ساختمان داده‌ها

الگوریتم:

تعریف ۱: مجموعهٔ محدودی از دستورالعمل‌ها که با دنبال کردن آن‌ها هدف خاصی دنبال می‌شود، که لزوماً منحصر به فرد نیست.

تعریف ۲: روش دقیق حل یک مسئله به صورت قدم به قدم که لزوماً منحصر به فرد نیست.

مثال: مطلوبست تغییر مقدار دو متغیر x و y $(x = 10 , y = 20)$

روش I	روش II	روش III
30 10 20 x ← x + y	10 10 temp ← x	200 10 20 x ← x * y
10 30 20 y ← x − y	20 20 x ← y	10 200 20 y ← x / y
20 30 10 x ← x − y	10 10 y ← temp	20 200 10 x ← x / y

مثال فوق نشان می‌دهد که روش حل یک مسئله ممکن است منحصر به فرد و یکتا نباشد.

□ خصوصیات الگوریتم:

۱) **ورودی:** حداقل 0 ورودی (ممکن است الگوریتم ورودی نداشته باشد.)

۲) **خروجی:** حداقل 1 خروجی (هر الگوریتم حداقل یک خروجی خواهد داشت.)

۳) **قطعیت (عدم ابهام):** هر کدام از دستورالعمل‌ها باید دقیق و بدون ابهام باشند.

۴) **محدودیت (پایان‌پذیر بودن):** هر الگوریتم پس از طی مراحل مشخصی باید اتمام پذیرد.

۵) **کارائی (انجام‌پذیر بودن):** امکان پیاده‌سازی و اجرای الگوریتم روی کاغذ وجود داشته باشد به عبارت بهتر الگوریتم انجام شدنی باشد.

برنامه:

مجموعه‌ای از دستورالعمل‌های یک زبان برنامه‌سازی که امکان پیاده‌سازی و اجرای یک الگوریتم در کامپیوتر را فراهم کند.

□ تفاوت برنامه و الگوریتم:

یک برنامه تمام خصوصیات یک الگوریتم را به جز "شرط پایان‌پذیر بودن" شامل می‌شود.

هر الگوریتم لزوماً پایان‌پذیر است اما هر برنامه لزوماً پایان‌پذیر نیست.

مثال: سیستم‌عامل برنامه‌ای است، که هیچ‌گاه پایان نمی‌پذیرد و همواره در یک سیکل انتظار قرار دارد تا برنامه بعدی وارد شود.

ساختمان داده‌ها:

به ساختارهائی که جهت دریافت داده‌های خام به شکل مناسب توسط کامپیوتر برای پیاده‌سازی و اجرای الگوریتم‌های مختلف مورد استفاده قرار می‌گیرند، ساختمان داده‌ها گفته می‌شود.

مسئله: 10 عدد صحیح را گرفته، به صورت مرتب نشان دهید.

ساختمان داده مورد نیاز: آرایهٔ صحیح 10تائی

ـ این مثال از ساختمان داده آرایه برای گرفتن 10 داده خام صحیح و حل مسئله استفاده می‌شود.

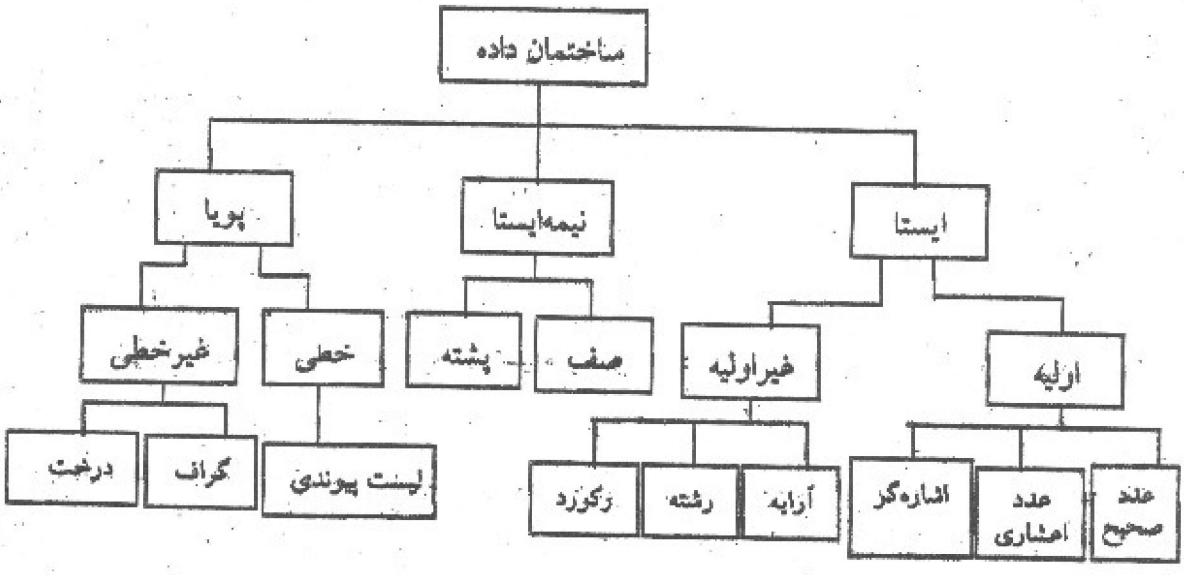
واع ساختمان داده‌ها:

ع ساختمان داده‌ها:

۱ـ ایستا
- ۱) **اولیه:** داده صحیح ـ داده اعشاری ـ داده کاراکتری
- ۲) **غیراولیه:** آرایه ـ رکورد ـ رشته ...

۲ـ نیمه ایستا
- ۱) پشته
- ۲) صف

۳ـ پویا
- ۱) **خطی:** لیست‌های پیوندی
- ۲) **غیرخطی:** درخت گراف



ه داده‌های ایستا:

ـ فضای محدود و از پیش تعریف شده استفاده کرده و این فضا در طول اجرای برنامه ثابت است.

انند:

(داده صحیح) int ← در طول اجرای برنامه به زبان C، همیشه 2 بایت است

(داده اعشاری) real ← در طول اجرای برنامه به زبان پاسکال، همیشه 6 بایت است.

داده‌های پویا (Heap):

داده‌های پویا با استفاده از اشاره‌گرها امکان تغییرات نامحدود و پویا متناسب با داده‌ها را فراهم می‌کند.

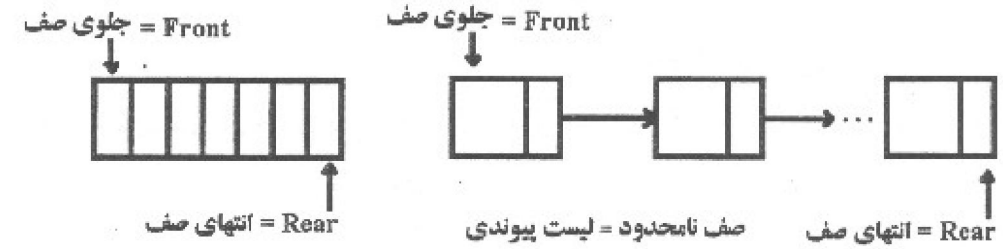
☑ داده‌های نیمه ایستا (پشته و صف):

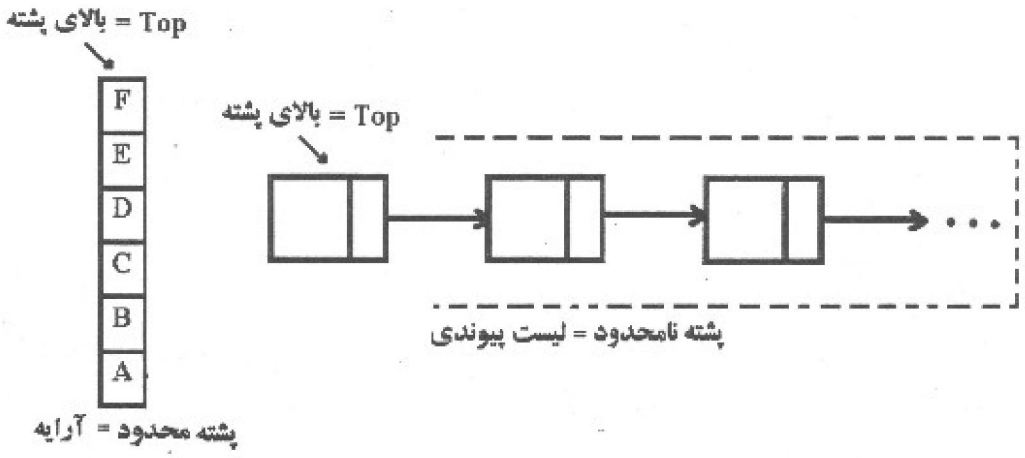
یادآوری:

۱) **صف:** لیستی که اولین ورودی در آن، اولین خروجی خواهد بود. (FIFO = First In First Out)

۲) **پشته:** لیستی که اولین ورودی در آن، آخرین خروجی خواهد بود. (FILO = First In Last out)

لیست مربوط به صف و پشته را می‌توان هم به صورت محدود با استفاده از آرایه‌ها (ساختارهای ایستا) و هم به صورت نامحدود با استفاده از لیست‌های پیوندی (ساختارهای پویا) پیاده‌سازی کرد، از این رو به آن‌ها نیمه ایستا می‌گویند.





■ مقدمه‌ای بر «تحلیل پیچیدگی زمانی و مرتبه اجرائی» الگوریتم‌ها:

در ارزیابی یک برنامه یا الگوریتم دو عامل اصلی وجود دارد، که باید مورد توجه قرار گیرد یکی حافظه مصرفی و دیگری زمان مصرفی الگوریتم است. یعنی الگوریتمی بهتر است که فضای مصرفی و زمان مصرفی کمتری را درخواست کند.

در الگوریتم‌هائی که در این مجموعه مورد بحث قرار می‌دهیم، «عامل زمان» مهمترین عامل به حساب می‌آید.

محاسبه زمان اجرائی یک الگوریتم متناسب یا تعداد دفعاتی است که عملیات اصلی انجام می‌شود. در تحلیل زمان اجرای یک الگوریتم تمام دستورات را شمارش نمی‌کنیم زیرا تعداد دستورات به نوع زبان برنامه‌نویسی بستگی دارد.

برای محاسبه زمان اجرای یک الگوریتم فقط به عملیات اصلی نیاز داریم که مستقل از کامپیوتر و زبان برنامه‌نوسی هستند، در عین حال، هیچ قاعده مشخص برای انتخاب عملیات اصلی وجود ندارد و انتخاب عمل اصلی به صورت تجربی انجام می‌پذیرد.

■ شمارش گام‌ها یا قدم‌ها یا مراحل یک برنامه:

در صورتیکه هر عمل اصلی در یک برنامه یک گام اختیار کند. گام‌های هر برنامه با استفاده از قواعد کلی زیر قابل محاسبه خواهد بود.

۱ـ تعاریف زیربرنامه‌ها و توابع دارای گام 0 است.

```
procedure  F(...);      →0
Function   P(...): ...; →0
void       F(...);      →0
نوع        P(...);      →0
```

۲ـ هر بلاک شامل { } یا begin و end دارای گام 0 است.

۳ـ تعاریف متغیرها در صورتیکه مقداردهی اولیه برای آن‌ها انجام گیرد، گام 1 و در غیر این‌صورت گام 0 دارند.

```
int   x; → 0
int   x = 3; → 1
float a,b = 5; → 1
```

```
var
j,i:integer; → 0
```

۴ـ درهر دستور اجرائی به ازاء هربار اجرا دارای گام 1 است.

```
y = x * y + z;   →1
```

مثال:

۱)

در حلقه زیر تعداد تکرار: 1+ ابتدا ـ انتها = n

```
for i:=1 to n do →  n+1
s:=s+1;          →+  n
                    2n+1
```

۲)

در حلقه زیر تعداد تکرار: $\boxed{n-2} = (n-1) - 2 + 1$ = 1+ ابتدا ـ انتها

```
for i:=2 to n-1 do →  n-1
s:=s+1;           →+ n-2
                     2n-3
```

۳)

در حلقه زیر تعداد تکرار: $\boxed{n-1} = n - 2 + 1$ = 1+ ابتدا ـ انتها

```
for i:=2 to n do ⟶   n
begin              +
s:=s+3;          ⟶+ n-1
r:=r+3;          ⟶  n-1
end;                3n-2
```

معادل مثال‌های ۱ و ۲ و ۳ در زبان C

۱

```
for (i = 1 ; i<=n ; i++)
    s = s + 1;
```

۲

```
for(i = 2 ; i<=n - 1 ; i++)
    s = s + 1;
```

۳

```
for(i = 2 ; i<=n ; i++)
    {
    s = s + 3;
    r = r + 3;
    }
```

۵ـ در دستور شرطی if عبارت شرط 1 گام دارد اما با توجه به درست یا غلط بودن شرط چون جملات مختلفی ممکن است اجرا شود، گام کل دستور شرط و البته به درست یا غلط بودن شرط خواهد بود (مگر در یک حالت خاص که در ازاء درست یا غلط بودن شرط فقط یک دستور اجرا شود که در این صورت 2 گام خواهد بود.)

مثال:

```
if (x<y) ⟶ 1
s = 2; → 1
   else
s = 5; → 1
```

شرط درست ⟶ 1 + 1 = [2]

شرط غلط ⟶ 1 + 1 = [2]

```
if (x < y) ⟶ 1
s = s + 1; ⟶ 1   } 1
else
{
t = t + 1; → 1   }
r = r + 1; → 1   } 2
}
```

شرط درست ⟶ 1 + 1 = [2]

شرط غلط ⟶ 2 + 1 = [3]

۶ـ دستور حلقه for:

مهمترین قسمت در شمارش گام‌های یک برنامه دستور حلقه for است که به ترتیب زیر گام آن را محاسبه می‌کنیم:

الف) ابتدا تعداد تکرار حلقه را محاسبه می‌کنیم.

ب) حلقه for به تعداد " تکرار +1 " گام اختیار می‌کند.

ج) جملات تکرار شونده داخل حلقه for به تعداد " تکرار " گام اختیار می‌کنند.

یادآوری:

۱ـ تعداد تکرار حلقه‌های زیر "b – a + 1" است.

for i : = a to b do یا for (i = a ; i <=b ; i++)

۲ـ تعداد تکرار حلقه زیر "b – a" است.

```
for (i = a ; i < b ; i++)
```

تفاوت مورد ۱ و ۲ آن است که در 1 شرط i <= b اما در 2 شرط i < b است.

■ روابط بالا در شرایطی عمل می‌کنند که به ازاء هر بار تکرار حلقه 1 واحد به متغیر شمارنده i اضافه شود.

۴)

در حلقه زیر چون شرط i < n است باتوجه به قسمت یادآوری، تعداد تکرار " ابتدا ـ انتها " بوده و برابر با n − 2 است.

```
for (i = 2 ; i < n ; i++)   ⟶   n − 1
    s = s + 1;              ⟶ + n − 2
                              ------
                              2n − 3
```

۵)

```
  0    ⟵  int  f(int x)
  0    ⟵  {
  1    ⟵  int i , j = 0;
  n    ⟵  for(i = 2 ; i<=n ; i++)
n − 1  ⟵  j = j + i;
  1    ⟵  return i;
------       }
2n + 1
```

تعداد تکرار در حلقه بالا $n - 1 = n - 2 + 1$ بار است.

■ حلقه‌های تو در تو:

برای محاسبه گام حلقه‌های تودرتو به صورت زیر عمل می‌کنیم:

۱ـ از بیرونی‌ترین حلقه شروع می‌کنیم.

۲ـ تعداد تکرار هر حلقه را برای تمام حلقه‌ها و دستورات تکرار شونده پائین در نظر گرفته و " تعداد تکرار + 1 " را برای خود حلقه در نظر می‌گیریم.

۳ـ قسمت 2 را برای تمام حلقه‌ها انجام می‌دهیم.

مثال:

۱)

```
for i : = 1 to m do   ⟶   (m + 1)
for j : = 1 to n do   ⟶ + m × (n + 1)
s = s + 1;            ⟶ + m × n
                          ---------------------
                          (m + 1) + m(n + 1) + mn = 2mn + 2m + 1
```

۲)

```
for  i : = 1  to  m  do   ⟶  (m + 1)
for  j : = 2  to  n − 1 do ⟶  (m) × (n − 1)
for  k : = 1  to  L  do   ⟶  (m) × (n − 2)(L + 1)
s : = s + 1;              ⟶  (m) × (n − 2)(L)
```

تکرار حلقه i: $m - 1 + 1 = m$

تکرار حلقه j: $n - 1 - 2 + 1 = n - 2$

تکرار حلقه k: $L - 1 + 1 = L$

۳)

```
for i:=1 to m do      ⟶  +(m+1)
for j:=2 to n do      ⟶  +(m)(n)
begin
s:=s+1;               ⟶  +(m)(n-1)
r:=r+1;               ⟶   (m)(n-1)
end;                       ―――――――
                           3mn-m+1
```

تکرار حلقه i: $m-1+1=m$ تکرار حلقه j: $n-2+1=n-1$

۴)

```
procedure add(var a,b,c:matrix; m,n:integer);
var
i,j:integer;
begin
for i:=1 to m do      ⟶  +(m+1)
for j:=1 to n do      ⟶  +(m)(n+1)
C[i,j]:=a[i,j]+b[i,j]; ⟶   (m)(n)
                           ―――――――
                           2mn+2m+1
end;
```

۵)

```
procedure P(x:integer);  ⟶  0
var
s,j,i:integer;           ⟶  0
begin                    ⟶  0
i:=0;                    ⟶  1
for i:=1 to m do         ⟶  +(m+1)
for j:=1 to n do         ⟶  +(m)(n+1)
s=s+1;                   ⟶  +(m)(n)
s=s+1;                   ⟶  +1
end;                     ⟶   0
                             ―――――――
                             2mn+2m+3
```

نکته ۱: تعداد گام حلقه‌های while مانند for محاسبه می‌شوند یعنی گام حلقه یک واحد از تعداد تکرار حلقه بیشتر است.

```
n+1 ⟵ for(i=1;i<=n;i++)        i=1;          ⟶1
n   ⟵ s=s+1;              ≡    while(i<=n)   ⟶n+1
―――                            {
2n+1                           s=s+1;        ⟶n
                               i=i+1;          n
                               }
                                             ―――
                                             3n+2
```

نکته۲: برای محاسبه تعداد گام حلقه‌های repeat ...until و while {...} do باید در نظر بگیریم که چون شرط حلقه انتهای حلقه کنترل می‌شود، تعداد گام حلقه برابر با تعداد تکرار حلقه است. برای محاسبه گام این نوع حلقه‌ها توصیه می‌شود برای یک مقدار فرضی n تکرار حلقه را محاسبه کرده و از روی آن گام را بدست می‌آوریم.

مثال:

$$2n-1 \leftarrow \begin{cases} 1 & \longleftarrow \text{ i:=1;} \\ & \text{repeat} \\ + & \\ + \quad n-1 & \longleftarrow \text{ i:=i+1;} \\ n-1 & \longleftarrow \text{ until i>=n;} \end{cases}$$

■ مرتبه اجرائی (O)

طبق تعریف $f(n) = O(g(n))$ می‌باشد در این حالت می‌گوئیم مرتبه اجرائی تابع f(n) ،g(n) می‌باشد. اگر و فقط اگر به ازاء توابع نامنفی f و g داشته باشیم:

$$f(n) = O(g(n)) \Leftrightarrow \exists\, C, n_0 > 0 : \forall\, n \geq n_0 \quad f(n) \leq C g(n)$$

برای انتخاب g(n) مناسب به صورت تجربی از f(n) جمله غالب را انتخاب می کنیم. به طور مثال:

$$f(n) = 3n + 3 = O(n)$$
$$f(n) = n^2 + 10n = O(n^2)$$
$$f(n) = \log_2^n + 1 = O(\log_2^n)$$

نکته۱: اگر $f(n) = a_m n^m + \ldots + a_1 n^1 + a_0$ باشد آن‌گاه $f(n) = O(n^m)$

$$f(n) = \frac{n+1}{2} = \frac{n}{2} + \frac{1}{2} = O(n)$$
$$f(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$
$$f(n) = 4 = 4n^0 = O(n^0) = O(1)$$
$$f(n) = a_0 = a_0 n^0 = O(n^0) = O(1)$$

نکته۲: جدول زیر مرتبه اجرائی توابع مهم را به ترتیب صعودی مطرح کرده است:

نام تابع	ثابت	لگاریتمی	خطی	-	مرتبه 2	توانی	فاکتوریل	توانی
مرتبه اجرا	$O(1) <$	$O(\log n) <$	$O(n) <$	$O(n\log n) <$	$O(n^2) <$	$O(2^n) <$	$O(n!)$	$< O(n^n)$

« بحث مربوط به مرتبه اجرائی به طور کامل در قسمت مرتب‌سازی شرح داده می‌شود»

■ توابع و زیر برنامه‌های بازگشتی (recursion)

استفاده از روند بازگشتی مربوط به مسائلی می‌شود که روند حل آن مسائل به شکل بازگشتی انجام می‌شود. به طور مثال:

۱) $n! = n \times (n-1)!$

۲) $x^n = x \times x^{n-1}$

در مثال ۱ اگر تابع Fact (n) را برای n! در نظر بگیریم رابطه بازگشتی آن

$$Fact(n) = n \times Fact(n-1)$$

و در مثال ۲ اگر تابع pow (x,n) را برای x^n در نظر بگیریم رابطه بازگشتی آن

Pow (x , n) = x * pow (x , n – 1)

می‌شود.

تعریف: به هر تابع یا زیربرنامه‌ای که بتواند داخل بدنه‌اش، خودش را دوباره فراخوانی کند یا صدا بزند، تابع یا زیربرنامه بازگشتی می‌گویند.

شرایط:

هر تابع بازگشتی نیاز به ۲ نقطه دارد.

۱ـ **نقطه بازگشت:** دستوری که دوباره تابع یا زیربرنامه را فراخوانی کند.

۲ـ **نقطه بن‌بست:** دستوری که با یک شرط مشخص باعث اتمام بازگشت‌ها شود.

مراحل بازگشت در ظرفی به نام stack یا پشته نگهداری می‌شود، و چون این ظرف محدود است، در صورت نبودن نقطه بن‌بست بعد از بازگشت‌های مکرر، این ظرف پرشده و با پیغام خطای stack over flow مواجه می‌شویم.

مثال: تابع بازگشتی زیر فاکتوریل عدد n را برمی‌گرداند.

(زبان C)

```
int Fact (int n)
 {
   if (n < = 1)
      return 1;(بن بست)
   else
    return n * Fact (n – 1); (بازگشت)
}
```

(زبان پاسکال)

```
Function Fact (n:integer): integer;
begin
        if  n < = 1 then  Fact: = 1 (بن بست)
        else  Fact:= n * Fact (n – 1); (بازگشت)
end;
```

دیده می‌شود که نقطه بن‌بست نقطه‌ای است که دیگر تابع فراخوانی نشده فقط یک مقدار برگردانده می‌شود و یا در بعضی مواقع در خروجی نمایش داده می‌شود. (در بعضی شرایط نیز با استفاده از دستور exit به روند بازگشت خاتمه داده و به مرحله قبل برمی گردیم)

مشخصات الگوریتم‌های بازگشتی:

۱ـ source (سورس کد) کوتاه: الگوریتم‌های بازگشتی مانند فاکتوریل را به راحتی می‌توان به صورت بازگشتی با دستورات کمتری نسبت به حالت غیربازگشتی پیاده‌سازی کرد.

۲ـ اتلاف حافظه بیشتر: به علت استفاده از حافظه اضافی stack که مراحل بازگشت را نگهداری می‌کند، اتلاف حافظه نسبت به حالت غیر بازگشتی بیشتر است.

۳ـ سرعت اجرا کمتر: سرعت اجرا در بیشتر الگوریتم‌های بازگشتی کمتر از الگوریتم غیربازگشتی می‌باشد.

❑ **مزایا:**

source (سورس) کوتاه ـ راحتی پیاده‌سازی برخی الگوریتم‌ها که روند حل آن‌ها بازگشتی است.

❑ **معایب:**

اتلاف حافظه ـ سرعت اجرای کمتر

■ **دلایل استفاده از توابع یا زیربرنامه‌های بازگشتی**

۱ـ راحتی تدوین و پیاده‌سازی مسائلی که روند حل آن‌ها به صورت بازگشتی (recursive) انجام می‌شود. مانند: فاکتوریل

۲ـ تعداد کم دستورات استفاده شده نسبت به حالت غیربازگشتی

الگوریتم بازگشتی فاکتوریل n

```
int Fact (int n)
{
if (n < = 1)
return 1;
else
return n * Fact (n);
}
```

الگوریتم غیربازگشتی فاکتوریل n

```
int Fact (int n)
{
int f = 1 , i;
for (i = 1 ; i < = n ; i++)
f = f * i;
return f;
}
```

نکته: توابع بازگشتی معمولاً می‌توانند به ۲ شکل بیان شوند.

۱ـ ضابطه ریاضی

۲ـ الگوریتم یا یک برنامه به زبان برنامه‌نویسی

مثال: فاکتوریل n

ضابطه ریاضی

$$fact = \begin{cases} 1 & n \le 1 \\ n \times fact(n-1) & \text{در غیر این صورت } (n > 1) \end{cases}$$

برنامه‌نویسی

```
function fact (n: integer): integer;
begin
if n <= 1 then
fact : = 1
else
fact : = n * fact ( n – 1)
end;
```

■ روش حل تابع و زیر برنامه‌های بازگشتی

۱ـ روش درختی: این روش زمانی استفاده می‌شود که تابع بازگشتی به صورت ضابطه ریاضی داده شود یا این که از مقدار بازگشتی در خطوط بعدی برنامه استفاده نشود.

۲ـ روش مرحله به مرحله: زمانی استفاده می‌شود که مقدار بازگشتی در خطوط بعدی زیربرنامه یا تابع بازگشتی به کار برده می‌شود، (معمولاً مربوط به زیربرنامه‌های بازگشتی است)

مثال ۱: اگر A و B دو عدد صحیح نامنفی باشند و تابع M به صورت زیر تعریف شده باشد، حاصل M (20 , 28) چیست؟

$$M(A,B) = \begin{cases} M(B,A) & A<B \\ A & B=0 \\ M(B,Mod(A,B)) & \text{وگرنه} \end{cases}$$

M(20 , 28)
↓↑
M(28 , 20)
↓↑
M(20 , 8)
↓↑
M(8 , 4)
↓↑
M(4 , 0)
↓↑
4 بن بست

مثال۲: فرض کنید که a و b اعداد صحیح مثبت بوده و تابع Q به صورت زیر به شکل بازگشتی تعریف شده باشد:

$$Q(a,b)=\begin{cases}0 & a<b\\ Q(a-b,b)+1 & a\geq b\end{cases}$$

حاصل Q (14 , 3) چیست؟

$$\overbrace{Q(14,3)}^{④}$$
$$\downarrow\uparrow$$
$$\overbrace{Q(11,3)}^{③}+1$$
$$\downarrow\uparrow$$
$$\overbrace{Q(8,3)}^{②}+1$$
$$\downarrow\uparrow$$
$$\overbrace{Q(5,3)}^{①}+1$$
$$\downarrow\uparrow$$
$$\overbrace{Q(2,3)}^{⓪}+1$$
$$\downarrow\uparrow$$
0 بن بست

مثال۳: خروجی تابع زیر برای فراخوانی Test (5 , 2) چقدر است؟

```
Function test (x , y: integer): integer;
begin
if (x < = y) or (y = 0)  then
test: = x
else if (y = 1) then test:= test (x – 1 , y) + 1
else test : = test (test (y , x) , y – 1) + 2
end;
```

البته می‌توان شکل ضابطه ریاضی تابع را به صورت زیر در نظر گرفت:

$$test(x,y)=\begin{cases}x & (x<=y)\ or\ (y=0)\\ test(x-1,y)+1 & y=1\\ test(test(y,x),y-1) & \text{وگرنه}\end{cases}$$

$$\overbrace{test(5,2)}^{④}$$
$$\downarrow$$
$$\overbrace{test(\underbrace{test(2,5)}_{2},1)}^{②}+2$$
$$\downarrow\uparrow$$
$$\overbrace{test(1,1)}^{①}+1$$
$$\downarrow\uparrow$$
1 بن بست

مثال: خروجی تابع زیر برپا فراخوانی ack (1 ,2) چقدر است؟

$$ack(m,n)=\begin{cases} n+1 & m=0 \\ ack(m-1,1) & (m<>0)\ And\ (n=0) \\ ack(m-1,ack(m,n-1)) & \text{وگرنه} \end{cases}$$

$\underbrace{ack(1,2)}$

↓ 4

ack (0 , $\underbrace{ack(1,1)}$)

↓ 3

ack (0 , $\underbrace{ack(1,0)}_{\downarrow 2}$)

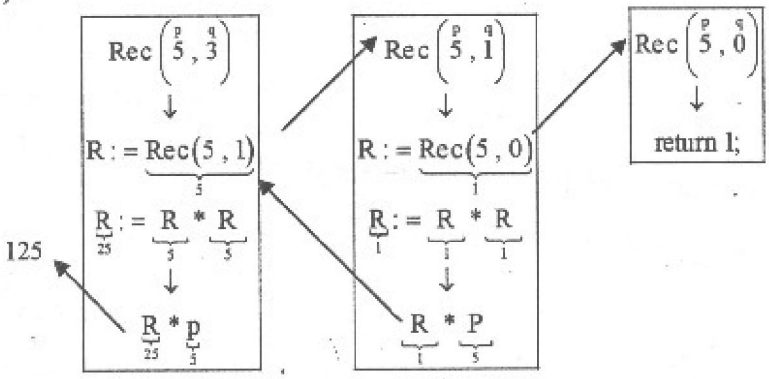
↓ 2

$\underbrace{ack(0,1)}$

2

مثال: در روال بازگشتی زیر مقدار Rec (5 , 3) کدام است؟

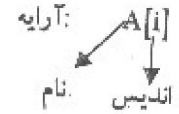
```
int Rec (int p , int q)
{
int R;
if (q < = 0)  return 1;
R = Rec (p , q/2);
R = R * R
if (q  mod  2 = = 0)
return R;
else
return  R * P ;
}
```

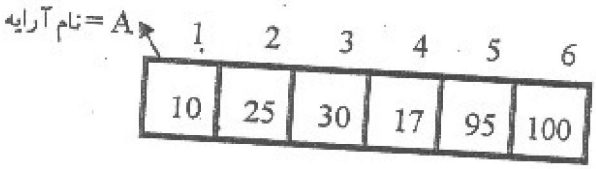


آرایه‌ها:

مجموعه‌ای از داده‌های پشت سرهم حافظه که همگی از یک نوع بوده و از آدرس مشخصی شروع می‌شوند.

نکته ۱: هر آرایه تحت یک نام واحد معرفی شده و هر عضو آن با یک اندیس مشخص قابل دستیابی است.





برای رسیدن به مقدار خانهٔ چهارم (17) باید از A[4] یا A (4) استفاده کنیم که در اینجا 4 اندیس خانه چهارم است.

نکته ۲: آرایه‌ها برحسب وضعیت‌های مختلف نمایش می‌توانند به صورت‌های یک بعدی، دو بعدی و ... مطرح شوند.

☑ **آرایه A می‌تواند به شکل‌های زیر مطرح شود:**

$A[L_1 .. U_1] \longrightarrow$ بردار = لیست = یک بعدی

$A[\underbrace{L_1 .. U_1}_{\text{سطر = بعد اول}} , \underbrace{L_2 .. U_2}_{\text{بعد دوم = ستون}}] \longrightarrow$ جدول = ماتریس = دوبعدی

$A[L_1 .. U_1 , L_2 .. U_2 , ... , L_n .. U_n] \rightarrow$ n بعدی

در حالت خاص اگر آرایه به صورت $A(U_1)$ و $A(U_1 , U_2)$ تعریف شود، اندیس پائین به صورت پیش فرض 1 در نظر گرفته می‌شود.

تعداد عناصر یک آرایه:

اگر آرایه به صورت یک یا چند بعدی تعریف شده باشد، تعداد عناصر آن برابر است با حاصلضرب زیر:

(تعداد عناصر بُعد n ام) × ... × (تعداد عناصر بُعد دوم) × (تعداد عناصر بُعد اول) = تعداد عناصر آرایه

یادآوری

تعداد عناصر بُعد i ام $= U_i - L_i + 1$

$A[L_1 .. U_1]$ تعداد عناصر $= U_1 - L_1 + 1$

$A[L_1 .. U_1 , L_2 .. U_2]$ تعداد عناصر $= (U_1 - L_1 + 1)(U_2 - L_2 + 1)$

$A[L_1 .. U_1 , L_2 .. U_2 , ... , L_n .. U_n]$ تعداد عناصر $= (U_1 - L_1 + 1) \times (U_2 - L_2 + 1) \times ... \times (U_n - L_n + 1)$

مثال:

تعداد عناصر آرایه $A(1..4,1..3) = (4-1+1)\times(3-1+1) = 12$ = تعداد عناصر آرایه $A(4,3)$

تعداد عناصر آرایه $A(1..4,-1..3,2..3) = (4-1+1)\times(3-(-1)+1)\times(3-2+1) = 40$

محاسبه فضای آرایه:

برای محاسبه فضای آرایه کافی است تعداد عناصر آرایه را در فضای نوع عناصر آن ضرب کنیم.

فضای نوع آرایه × تعداد عناصر آرایه = فضای آرایه

مثال:

A:Array $[-1..3, 2..5]$ of integer; (integer: 2 بایت)

تعداد عناصر آرایه $=(3-(-1)+1)\times(5-2+1)=20$

فضای نوع آرایه × تعداد عناصر آرایه = فضای آرایه $=20\times 2=40$ byte

محاسبه آدرس خانه دلخواه از یک آرایه (مهم)

مشخص کردن موقعیت عناصر یک آرایه با توجه به نحوه ذخیره‌سازی آنها به یکی از 2 روش سطری و ستونی متفاوت است.

10	20	30	70
5	15	45	32
12	13	2	24
7	9	8	4
1	2	6	14

مقدار 13 در آرایه (ماتریس) ↗ خانه 10 ام آرایه است = بصورت سطری

مقدار 13 در آرایه (ماتریس) ↘ خانه 8 ام آرایه است = بصورت ستونی

گاه ممکن است (در ماتریس‌های مربع) وضعیت ستونی و سطری با هم برابر شود.

10	20	30
40	50	60
70	80	90

مقدار 50 در آرایه (ماتریس) ↗ خانه 5 ام آرایه است = بصورت سطری

مقدار 50 در آرایه (ماتریس) ↘ خانه 5 ام آرایه است = بصورت ستونی

محاسبه آدرس خانه $A[i,j,k]$ در آرایه $A[L_1..U_1, L_2..U_2, L_3..U_3]$

α = آدرس شروع آرایه (در صورت عدم بیان آدرس شروع $\alpha = 0$ فرض می‌کنیم)

β = فضای نوع عناصر آرایه (در صورت عدم بیان فضای نوع عناصر آرایه $\beta = 1$ فرض می‌کنیم)

$$\text{آدرس سطری } A[i,j,k] = \alpha + \left[(i-L_1)(U_2-L_2+1)(U_3-L_3+1) + (j-L_2)(U_3-L_3+1) + (k-L_3) \right] \times \beta$$

حد پایین بعد اول — حد پایین بعد دوم

تعداد عناصر بعد سوم به بعد — تعداد عناصر بعد دوم به بعد — از چپ به راست

حد پایین بعد سوم حد پایین بعد دوم

$$\text{آدرس ستونی } A[\overrightarrow{i,j,k}] = \alpha + \left[(k - L_3)\underbrace{(U_2 - L_2 + 1)(U_1 - L_1 + 1)}_{\text{تعداد عناصر بعد اول به قبل}} + (j - L_2)\underbrace{(U_1 - L_1 + 1)}_{\text{تعداد عناصر بعد دوم به قبل}} + (i - L_1) \right] \times \beta$$

از راست به چپ

این روش را می‌توان برای حالت n بعدی نیز به راحتی تعمیم داد.

مثال۱: آدرس $A[i]$ در آرایه $A[L_1..U_1]$:

$$\alpha + [(i - L_1)] \times \beta$$

مثال۲: آدرس $A[i,j]$ در آرایه $A[L_1..U_1, L_2..U_2]$: (چون α و β مشخص نشده، $\alpha = 0$ و $\beta = 1$ در نظر می گیریم)

سطری $A[\overrightarrow{i,j}]$: $\alpha + [(i - L_1)(U_2 - L_2 + 1) + (j - L_2)] \times \beta = (i - L_1)(U_2 - L_2 + 1) + (j - L_2)$

ستونی $A[\overrightarrow{i,j}]$: $\alpha + [(j - L_2)(U_1 - L_1 + 1) + (i - L_1)] \times \beta = (j - L_2)(U_1 - L_1 + 1) + (i - L_1)$

مثال۳: آدرس $A[i,j,k]$ در آرایه $A[a,b,c]$ کدام است؟ (سطری و ستونی)

چون حد پایین بعدها مشخص نشده، $A[a,b,c]$ را بصورت $A[1..a,1..b,1..c]$ فرض می کنیم.

همچنین چون α و β بیان نشده به ترتیب $\alpha = 0$ و $\beta = 1$ در نظر می گیریم.

$$\text{آدرس سطری } A[\overrightarrow{i,j,k}] = \overset{0}{\alpha} + [(i-1) \times (b-1+1)(c-1+1) + (j-1) \times (c-1+1) + (k-1)] \times \overset{1}{\beta}$$

$$= (i-1)bc + (j-1)c + (k-1)$$

$$\text{آدرس ستونی } A[\overrightarrow{i,j,k}] = \overset{0}{\alpha} + [(k-1)(b-1+1)(a-1+1) + (j-1) \times (a-1+1) + (i-1)] \times \overset{1}{\beta}$$

$$= (k-1)ab + (j-1)a + (i-1)$$

مثال۴: آرایه ماتریسی A که 8×30 است را در نظر بگیرید که آدرس اولیه آن $A(1,1)=200$ و تعداد $w=4$ کلمه در حافظه وجود داشته باشد، مطلوبست آدرس سطری $A[12,4]$ ؟

α = آدرس شروع $=200$ ، $\beta=4$ byte

چون آرایه بصورت $A(30,8)$ مطرح شده، آنرا بصورت $A(1..30,1..8)$ در نظر می گیریم و خواهیم داشت:

$$\text{آدرس سطری } A[12,4] = \alpha + [(12-1)(8-1+1) + (4-1)] \times \beta$$

$$= 200 + [11 \times 8 + 3] \times 4 = \boxed{564}$$

مثال۵: آرایه $A[1..3,1..5]$ از آدرس 3000 به بعد ذخیره شده است و هر خانه 4 بایت اشغال کرده است مطلوبست آدرس $A[2,3]$ بصورت سطری و ستونی: $\alpha=3000$ ، $\beta=4$

سطری $A[\overrightarrow{2,3}] = 3000 + [(2-1)(5-1+1) + (3-1)] \times 4 = 3028$

ستونی $A[\overrightarrow{2,3}] = 3000 + [(3-1)(3-1+1) + (2-1)] \times 4 = 3028$

نکته: گاهی اوقات یک آرایه چند بعدی را به صورت سطری یا ستونی در یک آرایه یک بعدی ذخیره می‌کنند، در این صورت آدرس خانه مشخص در آرایه چند بعدی به شکل سطری یا ستونی در آرایه یک بعدی مورد نظر می‌باشد.

مثال: آرایه سه بعدی $M(10\,,\,20\,,\,30)$ را در آرایه یک بعدی $A(1..10\times 20\times 30)$ به صورت سطری ذخیره کرده‌ایم مطلوبست آدرس خانه $M(2\,,\,1\,,\,3)$؟

α = آدرس شروع آرایه مشخص نشده = 0

β = فضای نوع عناصر آرایه مشخص نشده = 1

$$M(2\,,\,1\,,\,3) \text{ آدرس سطری } = \overset{0}{\alpha} + \left[(2-1)(20-1+1)(30-1+1)+(1-1)(30-1+1)+(3-1)\right]\times \overset{1}{\beta}$$

$$= 602$$

□ **محاسبه شماره خانه** $A[i\,,\,j\,,\,k]$ **در آرایه** $A[L_1..U_1\,,\,L_2..U_2\,,\,L_3..U_3]$

برای این‌که محاسبه کنیم $A[i\,,\,j\,,k]$ چندمین خانه سطری یا ستونی ماتریس $A[L_1..U_1\,,\,L_2..U_2\,,\,L_3..U_3]$ است کافیست در شرایطی که $\alpha=0$ و $\beta=1$ در نظر گرفته‌ایم به آدرس $A[i\,,\,j\,,\,k]$ یک واحد اضافه کنیم.

$$A[\overline{i\,,\,j\,,\,k}] = \left(\overset{0}{\alpha} + \left[(i-L_1)(U_2-L_2+1)(U_3-L_3+1)+(j-L_2)(U_3-L_3+1)+(k-L_3)\right]\times \overset{1}{\beta}\right)+1$$

$$= \left[(i-L_1)(U_2-L_2+1)(U_3-L_3+1)+(j-L_2)(U_3-L_3+1)+(k-L_3)\right]+1$$

$$A[\overline{i\,,\,j\,,\,k}] = \left(\overset{0}{\alpha} + \left[(k-L_3)(U_2-L_2+1)(U_1-L_1+1)+(j-L_2)(U_1-L_1+1)+(i-L_1)\right]\times \overset{1}{\beta}\right)+1$$

$$= \left[(k-L_3)(U_2-L_2+1)(U_1-L_1+1)+(j-L_2)(U_1-L_1+1)+(i-L_1)\right]+1$$

مثال۱: در آرایه $A['a'..'e',1..4]$، $A['b',3]$ چندمین خانه آرایه است؟

چون وضعیت سطری یا ستونی مشخص نشده است پیش فرض به صورت سطری عمل می‌کنیم در ضمن فاصله 'a'..'e' را به شکل 1..5 در نظر می‌گیریم بنابراین وضعیت $A[2\,,\,3]$ در $A[1..5\,,\,1..4]$ مدنظر است.

$$A[2\,,\,3] = \left(0+\left[(2-1)(4-1+1)+(3-1)\right]\times 1\right)+1$$
$$= 7$$

$A[2\,,\,3]$ خانه 7ام به صورت سطری است

'a', 1	'a', 2	'a', 3	'a', 4
'b', 1	'b', 2	'b', 3	'b', 4
'c', 1	'c', 2	'c', 3	'c', 4
'd', 1	'd', 2	'd', 3	'd', 4
'e', 1	'e', 2	'e', 3	'e', 4

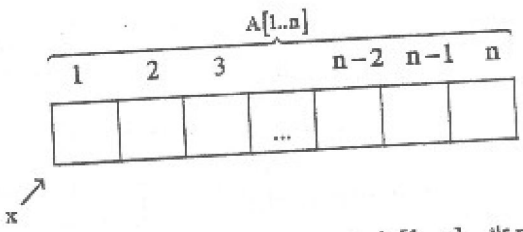
جستجو در آرایه‌ها:

به طور کلی برای جستجوی یک عنصر دلخواه در آرایه ها ۲ روش اساسی وجود دارد:

۱- روش جستجوی خطی (ترتیبی)
۲- روش جستجوی دودوئی (باینری)

۱- روش جستجوی خطی (ترتیبی)

در این روش برای جستجوی دادهٔ x در آرایه n تایی A [1..n]، جستجو را از یکی از دو طرف آرایه (پیش فرض از ابتدا) آغاز می‌کنیم و داده مورد نظر را پشت سر هم با عناصر آرایه مقایسه می‌کنیم، تا این‌که یا داده مورد نظر پیدا شود یا به طرف دیگر آرایه (انتهای آرایه) برسیم.



الگوریتم جستجوی خطی داده x در آرایه n تائی A [1.. n] :

```
flag : = flase;
For  i: =1  to  n  do
    if (A [ i ] = x ) then     (x پیدا شده و محل آن i است)
        begin
            Flag: = True;
            break;
        end;
 if Flag = True then
            write (' Location is:' , i ) ;
    else
            write (' not found');
```

در صورتی که شرط if برقرار شود (x = A [i]) عنصر x پیدا شده و محل آن i است در نتیجه Flag، True شده از حلقه خارج می‌شویم.

نکته ۱: در روش جستجوی خطی چون هیچ استراتژی خاصی جز جستجوی پشت سر هم از ابتدا تا انتها وجود ندارد، در نتیجه sort بودن یا نبودن آرایه تأثیری در روند جستجو ندارد.

تعداد مقایسات و مرتبه اجرائی

در الگوریتم جستجوی خطی مهمترین پارامتر، مقایسه است، در نتیجه تعداد مقایسات مهمترین عامل در محاسبه مرتبه اجرائی الگوریتم جستجوی خطی به حساب می‌آید.

مرتبه اجرائی و زمان جستجوی خطی مربوط به حالت متوسط است.

وضعیت مقایسات در جستجوی خطی

الف) بهترین حالت: داده مورد جستجو (x) در ابتدای لیست باشد (با فرض شروع جستجو از ابتدا). در این حالت حداقل مقایسه را داریم.

در نتیجه: تعداد مقایسه: 1 ـ مرتبه اجرائی O (1)

ب) بدترین حالت: داده مورد جستجو (x) در انتهای لیست باشد (با فرض شروع جستجو از ابتدا). در این حالت حداکثر مقایسه را داریم.

در نتیجه: تعداد مقایسه : n ـ مرتبه اجرائی O (n)

ج) حالت متوسط: متوسط مقایسات برابر است با: $\frac{\text{مجموع مقایسات}}{\text{تعداد عناصر آرایه‌ها}}$

در حالت کلی اگر داده x عنصر اول آرایه باشد با 1 مقایسه، اگر عنصر دوم باشد با 2 مقایسه و ... و اگر عنصر n ام باشد با n مقایسه بدست می‌آید.

در نتیجه:

$$\text{مجموع مقایسات} = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$\text{متوسط مقایسه} = \frac{\frac{n(n+1)}{2}}{n} = \frac{n+1}{2}$$

$$\text{مرتبه اجرائی متوسط} = O(n)$$

نکته مهم: زمان جستجوی خطی یا به عبارت بهتر مرتبه اجرائی آن مربوط به حالت متوسط بوده و برابر با O (n) است.

نتیجه‌گیری جستجوی خطی:

- حداقل مقایسه: 1 مقایسه
- متوسط مقایسه‌ها: $\frac{n+1}{2}$
- حداکثر مقایسه: n مقایسه
- زمان (مرتبه اجرائی) : O (n)

جستجوی دودوئی (binary search) :

شرط اولیه در این روش مرتب (sort) بودن آرایه است (پیش‌فرض صعودی)، در غیر این صورت این روش غیرقابل استفاده و تعریف نشده خواهد بود.

روش جستجو:

داده مورد جستجو (x) را با خانه میانی A [mid] آرایه مقایسه می‌کنیم، در صورتی که با آن خانه برابر باشد جستجو پایان می‌پذیرد در غیر این صورت در صورتی که x > A [mid] باشد به نیمه بالای آرایه می‌رویم و در صورتی که x < A [mid] باشد به نیمه پایین آرایه می‌رویم و دوباره با قسمت میانی آن نیمه عمل مقایسه را انجام می‌دهیم این عمل را تا زمانی انجام می‌دهیم که یا به داده مورد نظر برسیم که محل آن mid خواهد بود یا این‌که داده x در آرایه وجود ندارد که در این صورت Low (اندیس پائین نیمه آرایه) از high (اندیس بالای نیمه آرایه) **بیشتر** می‌شود.

الگوریتم جستجوی دودوئی:

مفروضات:

x = داده جستجو شونده

Flag = پیش‌فرض False و در صورت پیدا شدن x ، True می‌شود

Low = اندیس پایین آرایه

High = اندیس بالای آرایه

A [1.. n] = آرایه n تائی مرتب صعودی

$$mid = \left\lfloor \frac{low + high}{2} \right\rfloor =$$

اندیس وسط آرایه که عامل مقایسه x با A [mid] است.

I) الگوریتم غیر بازگشتی

```
Low: = 1 ;
high : = n ;
Flag : = False;
while (low < = high) AND (Flag < > True ) do
    begin
    mid : = (low + high) Div 2;
    if (A [mid] = x)          (x پیدا شده محل آن mid است)
                Flag: = True
  else if (A [mid] > x)       (باید به نیمه پایین برویم)
        High: = mid – 1
  else if (A [mid] < x)       (باید به نیمه بالا برویم)
        Low : = mid + 1
  end;
```

II) الگوریتم بازگشتی:

```
procedure binary search (a: elementary; x:element; var Left , Right , j:integer);
var
mid: integer;
begin
if (Left < = Right)  then
begin
mid: = compare (x , a[mid]) of
'>': binary search (a , x , mid + 1 , Right , j);
'<' : binary search (a , x , Left , mid – 1,j);
' = ' : j:=  mid;
end;
end;
```

مثال:

1	2	3	4	5	6	7	8
10	20	30	40	50	60	70	80
3	2		1				

Low	high	mid	A[mid]	x	Flag
1	8	4	40	10	False
1	3	2	20	10	False
1	1	1	10	10	False

Flag مقدار True گرفته بنابراین محل عنصر x، 1 = mid است.

1	2	3	4	5	6	7	8
10	20	30	40	50	60	70	80
			1		2	3	

Low	high	mid	A[mid]	x	Flag
1	8	4	40	75	False
5	8	6	60	75	False
7	8	7	70	75	False
8	8	8	80	75	False
8	7	(Low > high)			False

شرط اتمام حلقه اتفاق افتاده و Flag مقدار False داشته در نتیجه x در آرایه وجود ندارد.

نکته مهم: در الگوریتم‌های جستجو دودوئی:

الف) تعداد مقایسات برای یافتن دو عدد متفاوت با نتیجه موفق ممکن است متفاوت باشد.

ب) تعداد مقایسات برای نیافتن دو عدد متفاوت با نتیجه شکست ممکن است متفاوت باشد.

1	2	3	4	5	6	7	8
10	15	20	25	30	45	70	80
3	2	3	1	3	2	3	4

به طور مثال برای رسیدن به عدد 70 نیاز به 3 مقاسیه و برای رسیدن به عدد 15 نیاز به 2 مقایسه داریم، همین امر نیز برای دو نتیجه ناموفق نیز وجود دارد.

تعداد مقایسات و مرتبه اجرائی

در الگوریتم جستجوی باینری مهمترین پارامتر، مقایسه است، در نتیجه تعداد مقایسه مهمترین عامل در محاسبه مرتبه اجرائی الگوریتم جستجوی باینری به حساب می‌آید.

- مرتبه اجرائی و زمان جستجوی باینری مربوط به حالت متوسط است.

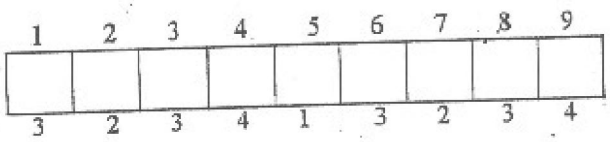
وضعیت مقایسات در جستجوی دودوئی

الف) بهترین حالت: عنصر در وسط لیست باشد.

در نتیجه: حداقل مقایسه ـ تعداد مقایسه: ۱ـ مرتبه اجرائی: O (1)

ب) بدترین حالت: می‌تواند در دو انتهای هر نیمه وجود داشته باشد.

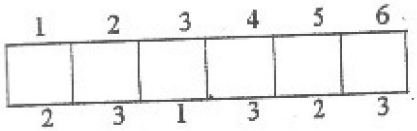
در نتیجه: حداکثر مقایسه ـ تعداد مقایسه: $\lfloor \log \frac{n}{2} \rfloor + 1$ ـ مرتبه اجرائی: O (n)



مقایسات

حداقل مقایسه = 1

حداکثر مقایسه $= \lfloor \log \frac{9}{2} \rfloor + 1 = 4$



مقایسات

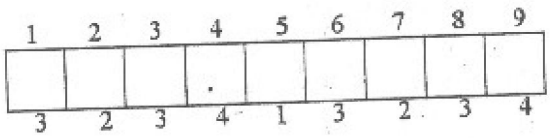
حداقل مقایسه = 1

حداکثر مقایسه $= \lfloor \log \frac{6}{2} \rfloor + 1 = 3$

ج) حالت متوسط

برای محاسبه متوسط تعداد مقایسات در روش جستجوی دودوئی 2 روش وجود دارد.

I) متوسط تعداد مقایسه‌ها $=$ $\frac{\text{مجموع مقایسات}}{\text{تعداد عناصر آرایه}}$

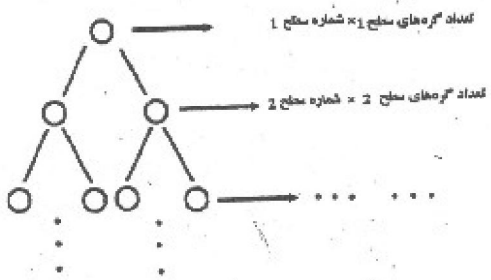


مقایسات

متوسط مقایسه $= \frac{3+2+3+4+1+3+2+3+4}{9} = \frac{25}{9}$

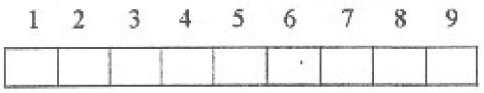
II) یک درخت دودوئی به تعداد عناصر آرایه تشکیل می‌دهیم و به شکل زیر عمل می‌کنیم.

■ سطح ریشه را 1 فرض می کنیم.

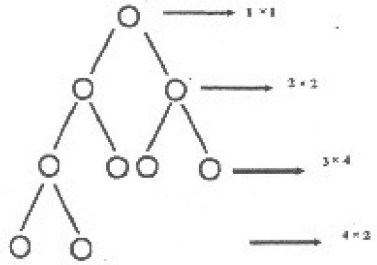


■ **متوسط مقایسه‌ها برابر است با:**

$$\text{متوسط مقایسات} = \frac{\sum \text{تعداد گره‌های سطح } i \text{ ام} \times \text{شماره سطح } i \text{ ام}}{\text{تعداد عناصر آرایه}}$$



درخت دودوئی ⟵ با 9 گره



$$\text{متوسط مقایسات} = \frac{1\times1+2\times2+3\times4+4\times2}{9} = \frac{25}{9}$$

■ زمان یا مرتبه اجرائی جستجوی دودوئی در حالت متوسط $O(\log_2^n)$ است.

نکته مهم: زمان جستجوی دودوئی یا به عبارت بهتر مرتبه اجرائی آن مربوط به حالت متوسط بوده و برابر با $O(\log_2^n)$ است.

مثال: اگر A آرایه‌ای مرتب از اعداد صحیح 1 تا 1024 باشد، الگوریتم جستجوی دودوئی با چند بار تکرار عدد 4 را پیدا می‌کند؟

۱) 8 ۲) 7 ۳) 9 ۴) 15

بار اول با مقایسه عدد 4 با وسط آرایه متوجه می‌شویم که عدد مذکور باید مابین خانه‌های 1 تا 512 یعنی نیمه پائینی آرایه باشد به همین ترتیب:

تعداد تکرار الگوریتم	عدد 4 در کدام محدوده است؟
1	1 تا 512
2	1 تا 256
3	1 تا 128
4	1 تا 64
5	1 تا 32
6	1 تا 16
7	1 تا 8

پس از 7 بار تکرار الگوریتم آرایه زیر در نظر گرفته می‌شود:

Low		mid				high	
1	2	3	4	5	6	7	8

$$\text{mid} = \left\lfloor \frac{1+8}{2} \right\rfloor = 4$$

بار هشتم هنگامی که عدد 4 با محتوای mid مقایسه می‌شود، پیدا می‌گردد پس الگوریتم 8 بار تکرار می‌شود.

نتیجه‌گیری جستجوی دودوئی:

- **حداقل مقایسه:** 1 مقایسه
- **حداکثر مقایسه:** $\lfloor \log_2^n \rfloor + 1$
- **زمان (مرتبه اجرائی) جستجوی دودوئی:** $O(\log_2^n)$

نتیجه‌گیری انواع جستجوها:

روش جستجو	زمان (مرتبه اجرائی)	حداقل مقایسه	حداکثر مقایسه	وضعیت آرایه
خطی (ترتیبی)	$O(n)$	1 مقایسه	n مقایسه	مرتب یا نامرتب
باینری (دودوئی)	$O(\log_2^n)$	1 مقایسه	$\lfloor \log_2^n \rfloor + 1$	مرتب

دیده می‌شود که روش جستجوی دودوئی به مراتب از روش جستجوی خطی از نظر تعداد مقایسه بهتر و مقرون به صرفه‌تر است.

ماتریس‌ها

به هر آرایه دو بعدی $m \times n$ یک ماتریس یا جدول با m سطر و n ستون گفته می‌شود. که تعداد mn خانه در آن وجود دارد.

ماتریس‌های مهم

۱ـ ماتریس مربع:

به هر ماتریس $n \times n$ با n^2 خانه ماتریس مربع گفته می‌شود.

در هر ماتریس مربع $n \times n$ مانند A روابط زیر برای اندیس خانه $A[i,j]$ وجود دارد:

$A[i,j]$ بالای قطر فرعی است $\Leftrightarrow i+j < n+1$

$A[i,j]$ روی قطر فرعی است $\Leftrightarrow i+j = n+1$

$A[i,j]$ پائین قطر فرعی است $\Leftrightarrow i+j > n+1$

$A[i,j]$ بالای قطر اصلی است $\Leftrightarrow i < j$

$A[i,j]$ روی قطر اصلی است $\Leftrightarrow i = j$

$A[i,j]$ پائین قطر اصلی است $\Leftrightarrow i > j$

قطر فرعی $(i+j=n+1)$ ↙ ↘ قطر اصلی $(i=j)$

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

3×3

۲ـ ماتریس بالا مثلث و پایین مثلث

در هر ماتریس مربع $n \times n$

الف) اگر عناصر زیر قطر اصلی 0 باشد ماتریس بالا مثلثی است.

ب) اگر عناصر بالای قطر اصلی 0 باشد ماتریس پائین مثلثی است.

$$\begin{bmatrix} X & X & X & \cdots & X \\ 0 & X & X & \cdots & X \\ 0 & 0 & X & \cdots & X \\ \vdots & \vdots & \cdots & \cdots & \vdots \\ 0 & 0 & 0 & \cdots & X \end{bmatrix}$$

(ب) بالا مثلثی

$$\begin{bmatrix} X & 0 & 0 & \cdots & 0 \\ X & X & 0 & \cdots & 0 \\ X & X & X & \cdots & 0 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ X & X & X & \cdots & X \end{bmatrix}$$

(الف) پایین مثلثی

شکل ۴ ـ ۱ : ماتریس‌های بالا مثلثی و پایین مثلثی

نکته مهم: در هر ماتریس بالا مثلث یا پایین مثلث $n \times n$:

n	×	×	⋯	×	×
+ n − 1	0	×	⋯	×	×
+ n − 2	0	0	×	×	×
+ ⋮	0	0	0	×	×
1	0	0	0	0	×

$n \times n$

$$\frac{n(n+1)}{2}$$

(تعداد خانه‌های مخالف صفر)

تعداد کل خانه‌ها $= n^2$

تعداد خانه‌های مخالف صفر $= \frac{n(n+1)}{2}$

تعداد خانه‌های صفر $= \frac{n(n-1)}{2}$

۳ـ ماتریس L قطری:

هر ماتریس $n \times n$ که در آن L قطر آن که قطر اصلی نیز شامل آن است مخالف صفر باشد ماتریس L قطری می‌گویند.

×	0	0
0	×	0
0	0	×

3×3

ماتریس 1 قطری = ماتریس قطری

×	×	0
×	×	×
0	×	×

3×3

ماتریس 3 قطری

ضرب ماتریس‌ها

بر روی هر ماتریس عملیات مختلفی می‌تواند انجام گیرد. مانند: جمع، تفریق، ضرب، ... که یکی از مهمترین آن‌ها عملیات ضرب ماتریس‌ها است.

□ شرایط ضرب دو ماتریس:

برای آن‌که ضرب دو ماتریس A و B امکان‌پذیر باشد باید بُعد وسط آن‌ها یکسان باشد به عبارت بهتر برای امکان‌پذیر بودن حاصلضرب $A \times B$ باید ستون ماتریس A با سطر ماتریس B یکسان باشد.

در نتیجه: $C_{m \times k} \leftarrow A_{m \times n} \times B_{n \times k}$

□ خواص ضرب ماتریس‌ها:

الف ضرب ماتریس‌ها خاصیت جابجایی ندارد. $AB \neq BA$

ب) ضرب ماتریس‌ها خاصیت شرکت‌پذیری دارد.

حالت‌های شرکت‌پذیری ۴ ماتریس D ، C ، B ، A

۱) (AB)(CD)
۲) ((A(BC))D)
۳) (A((BC)D))
۴) (((AB)C)D)
۵) (A(B(CD)))

حالت‌های شرکت‌پذیری ۳ ماتریس C ، B ، A

۱) A(BC)
۲) (AB)C

■ **محاسبه تعداد حالات شرکت‌پذیری ضرب ماتریس‌ها:**

بنابر قضیه کاتالان تعداد حالات شرکت‌پذیری ضرب $n+1$ ماتریس از رابطه زیر به‌دست می‌آید:

$$\frac{\binom{2n}{n}}{n+1}$$

مثال: تعداد حالات شرکت‌پذیری ۴ ماتریس کدام است؟

$$n+1=4 \Rightarrow n=3$$

$$\text{تعداد حالات شرکت‌پذیری} = \frac{\binom{6}{3}}{3+1} = 5$$

نکته:

هرگاه یک ماتریس بالا مثلث را در ماتریس بالا مثلث دیگری ضرب کنیم، حاصل ماتریس بالا مثلث است.

هرگاه یک ماتریس پائین مثلث را در ماتریس پائین مثلث دیگری ضرب کنیم، حاصل ماتریس پایین مثلث است.

هرگاه یک ماتریس قطری را در ماتریس قطری دیگری ضرب کنیم، حاصل ماتریس قطری است.

به عبارت دیگر ضرب یک ماتریس در ماتریس دیگر با همان مشخصه ماتریس اول خاصیت ماتریس را عوض نمی‌کند.

ماتریس اسپارس (تُنک ـ پراکنده ـ خلوت ـ sparse)

به هر ماتریس $m \times n$ که تعداد خانه‌های 0 و یا بدون ارزش (nonvalue) آن بیشتر از تعداد خانه‌های مخالف 0 آن باشد.ماتریس اسپارس می‌گویند.

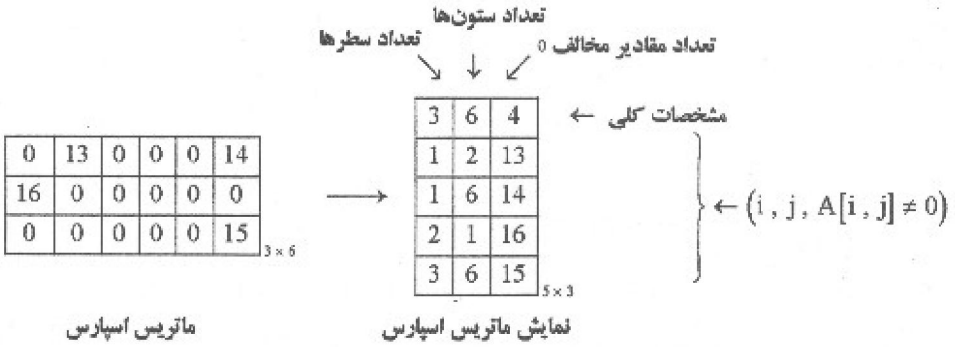
■ **روش عمومی برای نمایش ماتریس اسپارس: (ذخیره مختصات خانه‌های مخالف 0)**

در صورتیکه ماتریس $m \times n$ اسپارس r خانه مخالف 0 داشته باشد، برای نمایش آن از یک آرایه دو بعدی با 3 ستون و $r+1$ سطر استفاده می‌شود، به شرح زیر:

الف) سطر اول به ترتیب از چپ به راست، m (تعداد سطر) و n (تعداد ستون) و r (تعداد خانه‌های مخالف 0) گذاشته می‌شود.

ب) از سطر دوم به بعد، هر سطر شامل سه مؤلفه است که همان مختصات و مقدار خانه‌های مخالف 0 است، یعنی: $(i, j, A[i,j] \neq 0)$

نکته: این روش باعث صرفه‌جوئی در حافظه می‌شود چون فقط مختصات خانه‌های مخالف 0 نگهداری می‌شود.



0	13	0	0	0	14
16	0	0	0	0	0
0	0	0	0	0	15

3	6	4
1	2	13
1	6	14
2	1	16
3	6	15

چون خانه‌های مخالف 0، 4 تا است، بنابراین ماتریس اسپارس شامل 5 سطر و 3 ستون خواهد بود.

ترانهاده کردن ماتریس sparse: (transpose)

برای ترانهاده کردن ماتریس sparse به این شکل عمل می‌کنیم که ابتدا در سطر اول، جای سطر و ستون را عوض کرده و به همراه تعداد خانه‌های مخالف صفر در ماتریس ترانهاده قرار می‌دهیم. سپس از سطر دوم به بعد روی ستون دوم به ترتیب از بالا به پایین کوچکترین عنصر را پیدا کرده، جای سطر و ستون آن‌ها را عوض کرده و به همراه مقدارشان در ماتریس ترانهاده قرار می‌دهیم.

3	6	4
1	2	13
1	6	14
2	1	16
3	6	15

ترانهاده ⟶

6	3	4
1	2	16
2	1	13
6	1	14
6	3	15

نکته مهم: بعضی ماتریس‌ها مانند ماتریس‌های بالا مثلث و پائین مثلث، ماتریس اسپارس نیستند، اما زمانی که بعدهای آن‌ها زیاد شود، تعداد صفرهای ماتریس قابل توجه خواهد بود برای همین از روش زیر برای نمایش آن‌ها استفاده می‌کند:

۱ـ یک آرایه یک بعدی برای نگهداری مقادیر مخالف صفر آرایه (تعداد عناصر آرایه = تعداد مقادیر مخالف صفر ماتریس)

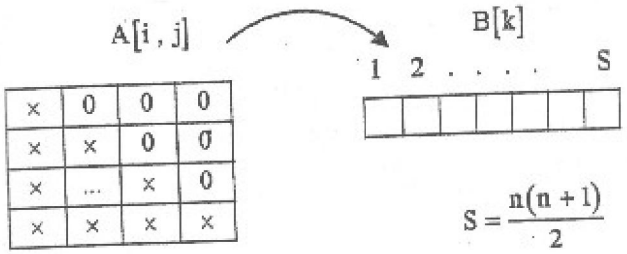
۲ـ یک رابطه یا فرمول که محل مقادیر مخالف صفر ماتریس را در آرایه مشخص کند.

مثال: ماتریس پائین مثلث $n \times n$ را در نظر می‌گیریم:

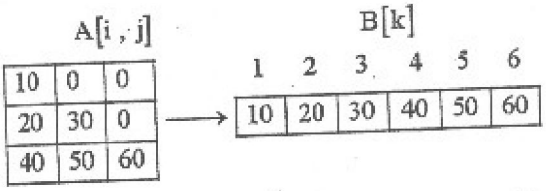
۱ـ در این ماتریس $\frac{n(n+1)}{2}$ خانه مخالف صفر وجود دارد. برای همین یک آرایه یک بعدی به تعداد خانه‌های مخالف صفر $\left(\frac{n(n+1)}{2}\right)$ اختیار می‌کنیم.

۲ـ برای هرخانه مخالف صفر $A[i,j]$ در ماتریس پائین مثلث، رابطه زیر محل آن خانه را در آرایه یک بعدی B مشخص می‌کند. (B [k])

$$k = \frac{i(i-1)}{2} + j$$



$$S = \frac{n(n+1)}{2}$$



به طور مثال $A[3,2] = 50$ در آرایه یک بعدی B [5] است و این ارتباط از طریق رابطه زیر فراهم می‌شود.

$$k = \frac{i(i-1)}{2} + j$$

$$\underbrace{A[3,2]}_{A[i,j]} \xrightarrow[j=2]{i=3} \underbrace{k = \frac{3(3-1)}{2} + 2 = 5}_{k = \frac{i(i-1)}{2} + j} \longrightarrow \underbrace{B[5]}_{B[k]}$$

مثال: برای هر خانه مخالف صفر A[i , j] در ماتریس بالا مثلث $n \times n$، رابطه زیر محل آن خانه را در آرایه یک بعدی B مشخص می‌کند. (B[k])

$$k = (i-1)\left(n - \frac{i}{2}\right) + j$$

نتیجه گیری:

بین دو روش بیان شده یعنی:

۱- روش عمومی نمایش ماتریس اسپارس (نگهداری مختصات مقادیر مخالف صفر با استفاده از آرایه 2 بعدی)

۲- نگهداری مقادیر مخالف صفر در آرایه یک بعدی که تعداد عناصر آن به تعداد خانه‌های مخالف 0 است.

روش دوم از نظر حافظه مقرون به صرفه‌تر است البته به شرط آن‌که بتوان رابطه یا فرمولی بین اندیس‌های آرایه B [k] و اندیس‌های مقادیر مخالف صفر A [i , j] پیدا کرد.

نکته مهم: حاصلضرب، جمع و تفریق ماتریس‌های sparse ممکن است که sparse نباشد.

جمع: $\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix}$

اسپارس نیست.

تفریق: $\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix} - \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} = \begin{bmatrix}1 & -1\\ 0 & 0\end{bmatrix}$

اسپارس نیست.

ضرب: $\begin{bmatrix}1 & 0 & 0\\ 1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} \times \begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} = \begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 0 & 0 & 0\end{bmatrix}$

اسپارس نیست.

محاسبه تعداد ضرب‌های حاصلضرب چند ماتریس:

اگر حاصلضرب دو ماتریس $C_{m\times k} = A_{m\times n} \times B_{n\times k}$ را در نظر بگیریم. قطعه برنامه‌زیر این عمل را انجام می‌دهد.

```
for i := 1  to  m  do
for j := 1  to  k  do
for L: = 1  to  n  do
C [i,j]: = C [i,j] + A [i,L] * B [L,j];
```

- تعداد ضرب‌های انجام شده به تعداد تکرار ۳ حلقه بوده و برابر است با: $m\times n\times k$
- تعداد جمع‌های انجام شده به تعداد تکرار ۳ حلقه بوده و برابر است با: $m\times n\times k$

البته اگر دو ماتریس $A_{m\times n}$ و $B_{n\times k}$ را در هم ضرب کنیم تعداد جمع‌ها $m(n-1)k$ می‌شود اما چون در کامپیوتر این جمع‌ها باید در ماتریس $C_{m\times k}$ که ماتریس حاصل از ضرب دو ماتریس است، قرار گیرد. یک جمع نیز به ازاء هر خانه $C_{m\times k}$ اضافه می‌شود که در نهایت $m(n-1)k + mk$ جمع خواهیم داشت که همان mnk است. (قبل از انجام ضرب تمام خانه‌های ماتریس $C_{m\times k}$ را 0 قرار می‌دهیم.)

مثال (I): تعداد ضرب‌های حاصلضرب 3 ماتریس $C_{5\times 4}$ ، $B_{10\times 5}$ ، $A_{3\times 10}$ با توجه به حالت‌های مختلف شرکت‌پذیری کدام است؟

$$A_{3\times 10}\,(BC)_{10\times 4} = (3\times 10\times 4) + (10\times 5\times 4) = 120 + 200 = 320$$

$$(AB)_{3\times 5}\,C_{5\times 4} = (3\times 10\times 5) + (3\times 5\times 4) = 150 + 60 = 210$$

دیده می‌شود که حالت‌های مختلف شرکت‌پذیری، تعداد ضرب‌های متفاوتی بوجود می‌آورد.

■ محاسبه بهینه‌ترین تعداد ضرب در ضرب چند ماتریس:

در ضرب چند ماتریس بهینه‌ترین حالت در مورد تعداد ضرب‌ها داشتن کمترین تعداد ضرب است تا عمل ضرب ماتریس‌ها سریعتر انجام شود.

قضیه: برای محاسبه ضرب 3 ماتریس $A_{m\times n} \times B_{n\times k} \times C_{k\times L}$ برای آن‌که:

$$\text{تعداد ضرب } A(BC) < \text{تعداد ضرب } (AB)\,C \longleftrightarrow \frac{1}{m} + \frac{1}{k} > \frac{1}{n} + \frac{1}{L}$$

طبق یک قاعده سرانگشتی می‌توان گفت که هنگام ضرب چند ماتریس، اگر ماتریس‌هایی را زودتر ضرب کنیم که بُعد وسط آن‌ها بزرگتر و بعد کناری آن‌ها به نسبت کوچکتر است، بدین ترتیب تعداد ضرب‌ها کوچکتر می‌شود. البته این قاعده همیشه درست نیست در عین حال قضیه بیان شده همیشه درست عمل می‌کند.

مثال:

(II) تعداد ضرب‌های حاصلضرب 3 ماتریس $C_{4\times 2}$ و $B_{5\times 4}$ و $A_{3\times 5}$ با توجه به حالت‌های مختلف شرکت‌پذیری کدام است؟

$$(AB)_{3\times 4}\,C_{4\times 2} = (3\times 5\times 4) + (3\times 4\times 2) = 60 + 24 = 84$$

$$A_{3\times 5}\,(BC)_{5\times 2} = (3\times 5\times 2) + (5\times 4\times 2) = 30 + 40 = 70$$

تحلیل:

در مثال (I)

$A_{3\times10} \times B_{10\times5} \times C_{5\times4}$

چون $\frac{1}{3}+\frac{1}{5} > \frac{1}{10}+\frac{1}{4}$ در نتیجه تعداد ضرب‌های (AB) C کمتر از تعداد ضرب‌های A (BC) است. در اینجا بُعد وسط AB بزرگتر بود.

در مثال (II)

$A_{3\times5} \times B_{5\times4} \times C_{4\times2}$

چون $\frac{1}{5}+\frac{1}{2} > \frac{1}{3}+\frac{1}{4}$ در نتیجه تعداد ضرب‌های A (BC) کمتر از تعداد ضرب‌های (AB) C است. در اینجا با این‌که بُعد وسط AB بزرگتر از BC بود اما تعداد ضرب (AB) C بیشتر از A (BC) است.

ولی قضیه همیشه درست عمل می‌کند.

نکته مهم: اگر در صورت یک مسئله برای ضرب چند ماتریس، بهینه بودن (تعداد ضرب کمتر - سرعت اجرا) مطرح نباشد، ماتریس‌ها را از سمت چپ به راست پشت سرهم ضرب می‌کنیم.

به طورمثال در ضرب ماتریس‌های $A_{m\times n} \times B_{n\times k} \times C_{k\times L}$ پیش فرض حاصلضرب را به صورت (AB) C در نظر می‌گیریم.

مجموعه سوالات

۱ـ آرایه دوبعدی زیر در آدرس 3000 به بعد حافظه قرار دارد، آدرس عنصر List[2 , 8] به روش سطری کدام است؟

List : Array[-3 .. 4,5 .. 20] of real;

6 بایت

۱) 3114 ۲) 3240 ۳) 3258 ۴) 3498

۲ـ بهترین راه نمایش مسیر در مساله Mazing (مسیر پرپیچ‌وخم) کدام است؟

۱) آرایه n×3 ۲) آرایه یک‌بعدی ۳) آرایه دوبعدی ۴) لیست پیوندی

۳ـ تعداد عناصر غیرصفر در یک ماتریس بالامثلثی و یا پائین مثلثی کدام است؟

۱) $\frac{n(n+1)}{2}$ ۲) $\frac{n(n-1)}{2}$ ۳) $n(n+1)$ ۴) $\frac{n^2+2n}{2}$

۴ـ تعداد عناصر صفر در یک ماتریس بالا مثلث و یا پائین مثلثی کدام است؟

۱) $\frac{n(n+1)}{2}$ ۲) $\frac{n(n-1)}{2}$ ۳) $n(n+1)$ ۴) هیچکدام

۵ـ برای نمایش یک ماتریس بالا یا پائین مثلثی در یک آرایه یک‌بعدی به حداقل چند خانه احتیاج داریم؟

۱) $n(n+1)$ ۲) $\frac{n(n+1)}{2}$ ۳) n^2 ۴) $\frac{n^2+2n}{2}$

۶ـ برای حذف عنصر Kام از یک آرایه N عنصری چند جابجائی لازم است؟

۱) N-K-1 ۲) K ۳) N-K ۴) N-K+1

۷ـ در ضرب سه‌آرایه $A(3,4), B(4,6), C(6,2)$ به ترتیب A*B*C چند عمل ضرب نیاز است؟

۱) 25 ۲) 108 ۳) 2592 ۴) 3456

۸ـ برای ضرب 5 ماتریس چند حالت مختلف برای ضرب در شرکت‌پذیری وجود دارد؟

۱) 24 ۲) 14 ۳) 36 ۴) هیچکدام

۹ـ آرایه ماتریسی A که 4×30 است را در نظر بگیرید. فرض کنید آدرس اولیه $A(1,1)=200$ و تعداد $w=4$ کلمه در حافظه وجود داشته باشد فرض هم بر این است که زبان برنامه‌سازی آرایه دو بعدی را با روش سطری ذخیره می‌کند آدرس A[12,6] کدام است؟ (سراسری ۸۲)

۱) 196 ۲) 356 ۳) 396 ۴) 844

۱۰ـ A['M',15] چندمین عنصر از آرایه تعریف شده زیر محسوب می‌شود؟

A : Array['K'..'N',11..50]

۱) 19 ۲) 23 ۳) 85 ۴) 125

۱۱ - هدف از فشرده‌کردن آرایه‌ها چیست؟

۱) نامحدودکردن آرایه ۲) دسترسی سریع‌تر به عناصر ۳) صرفه‌جوئی در حافظه ۴) کوچک‌شدن آرایه

۱۲ - کدامیک از روش‌های زیر برای نمایش یک ماتریس اسپارس از نظر صرفه‌جویی در حافظه بهتر عمل می‌کند؟

۱) لیست پیوندی ۲) ذخیره مختصات ۳) ذخیره مختصات با اشاره‌گر ۴) الگوی دودوئی

۱۳ - آرایه سه‌بعدی M[1..a,1..b,1..c] در یک آرایه یک‌بعدی N[1..a×b×c] به روش ستونی ذخیره شده است. آدرس عنصر M[i,j,k] در آرایه N کدام است؟

۱) $a\times b\times c+b\times c\times 1$

۲) $i\times b\times c+j\times c\times k$

۳) $(k-1)ab+(j-1)a+(i-1)$

۴) $(i-1)bc+(j-1)c+(k-1)$

۱۴ - ماتریس اسپارس را توسط کدام گزینه می‌توان تعریف کرد؟ (max تعداد عناصر غیر صفر)

۱) S[0..max,0..3] ۲) S[0..max,1..3] ۳) S[1..max,1..max] ۴) S[0..max,2..3]

۱۵ - A['c',13] چندمین عنصر از آرایه تعریف شده زیر محسوب می‌شود؟

A : array['a'..'z',10..20]

۱) 26 ۲) 25 ۳) 24 ۴) 27

۱۶ - درصورتیکه آرایه مورد جستجو در جستجوی دودوئی به‌صورت 7 , 6 , 5 , 4 , 3 , 2 , 1 , 0 , -1 باشد متوسط تعداد مقایسه‌ها برای جستجوی موفق چیست؟

۱) $\frac{27}{9}$ ۲) $\frac{25}{9}$ ۳) $\frac{31}{9}$ ۴) هیچکدام

۱۷ - در یک آرایه، n عدد به‌ترتیب فردی یا صعودی مرتب قرار دارد، اگر از روش جستجوی دودوئی (باینری) برای یافتن عددی استفاده کنیم، حداکثر تعداد مقایسه چقدر خواهد بود؟

۱) $n\log_2^n$ ۲) $\lceil\log_2^n\rceil+1$ ۳) $\log_2^{(n-1)}$ ۴) $n-\log_2^n$

۱۸ - در یک آرایه، n تائی مرتب شده (نزولی - صعودی) یا نامرتب حداکثر تعداد جستجو در حالت جستجوی خطی (ترتیبی) کدام است؟

۱) n ۲) n - 1 ۳) $n/2$ ۴) n^2

۱۹ - در یک آرایه nتایی نامرتب، حداکثر تعداد مقایسه برای جستجوی موفق چیست؟

۱) $\log_2^n$ ۲) $\lceil\log_2^n\rceil+1$ ۳) تعریف نشده ۴) هیچکدام

۲۰ - متوسط تعداد مقایسه در جستجوی خطی یک آرایه n عضوی کدام است؟

۱) $\frac{n+1}{2}$ ۲) $n/2$ ۳) $\log_2^n$ ۴) n^2

۲۱ - زمان جستجوی خطی و باینری در یک لیست n تائی از راست به چپ کدام است؟

۱) $o(n^2),o(n)$ ۲) $o(\log_2^n),o(n)$ ۳) $o(n^2),o(\log_2^n)$ ۴) هیچکدام

۲۲ـ آدرس عنصر $A[x]$ از یک آرایه یک‌بعدی که به صورت $A:array[m_1..m_2]$ و از جنس real تعریف می‌شود کدام است؟ (فرض کنید آدرس شروع H باشد.)

۱) $H+(x-m_1)*6$ ۲) $H+(m_1-m_2-x)*6$ ۳) $H+(x-m_2)*6$ ۴) $H+(x-m_2-m_1)*6$

۲۳ـ آرایه ۳بعدی $A[1..m,1..n,1..p]$ در یک آرایه یک‌بعدی $B[1..m\times n\times p]$ به روش سطر به سطر ذخیره شده است. آدرس عنصر $A[i,j,k]$ در B کدام است؟ (دولتی ۸۰)

۱) $(i-1)np+(j-1)m+(k-1)$ ۲) $(i-1)np+(j-1)p+(k-1)$

۳) $mnp+np\times 1$ ۴) $inp+jp+k$

۲۴ـ می‌خواهیم حاصلضرب ABCD را بدست آوریم، به‌طوری‌که کمترین تعداد عمل ضرب را داشته باشد، ترتیب ضرب ماتریس‌ها کدام است؟

$A_{13\times 5}B_{5\times 89}C_{89\times 3}D_{3\times 34}$

۱) $(A(BC))D$ ۲) $A((BC)D)$ ۳) $(AB)(CD)$ ۴) هیچکدام

۲۵ـ آرایه دوبعدی $X[-5..5,3..33]$ در آدرس 400 به بعد حافظه قرار دارد و هر خانه آرایه احتیاج به 4 بایت دارد. آدرس عنصر $X[4,10]$ به روش ستونی کدام است؟

۱) 1444 ۲) 1544 ۳) 844 ۴) 744

۲۶ـ دستور حذف‌شده برنامه جستجوی دودوئی زیر کدام است؟

```
procedure Binsearch (a: elementarray; x: element; var left, Right, j: integer)
   Vav
       middle : integer;
   begin
   if (Left < = right) then
                    begin
       middle : =(right + Left) Div 2;
       Case Compare (x , a [middle]) of
          '>' : Binsearch (a , x , middle +1 , right , j);
          '<' : دستور حذف شده
          '=' : j: = middle;
                   end;
end;
```

۱) Binsearch(a , x , middle – 1 , right , j);
۲) Binsearch (a , x, middle -1 , left , j)
۳) Binsearch (a , x , left , middle -1 ,j);
۴) هیچکدام

۲۷ـ تفاوت لیست با مجموعه کدام است؟

۱) در لیست داده تکراری می‌تواند وجود داشته باشد.
۲) در لیست ترتیب داده‌ها مهم است.
۳) در لیست داده‌ها به‌ترتیب صعودی یا نزولی داشته باشد.
۴) ۱ و ۲

۲۸- اگر آرایه A[1..m,1..n] در آرایه B[1..m×n] به روش سطربه‌سطر ذخیره شده باشد، آدرس عنصر A[i,j] در آرایه B کدام است؟

۱) (i-1) m+j
۲) (j -1)n+i
۳) (i – 1) n + j -1
۴) (j – 1) m + i

۲۹- آرایه ۳ بعدی A[1:15,-5:5,10:25] برای ذخیره اعداد صحیح به طول 2 بایت به کار گرفته است. اگر آرایه به‌صورت سطری از آدرس 2000 به بعد ذخیره شده باشد آدرس عنصر A[5,2,15] چیست؟

۱) 3868
۲) 3370
۳) 3420
۴) 3642

۳۰- کدام روش برای ذخیره ماتریس‌های پائین مثلثی مناسب‌تر می‌باشد؟

۱) ماتریس خطوط
۲) ماتریس دوبعدی
۳) آرایه یک‌بعدی
۴) لیست پیوندی یک طرفه یا دوطرفه

۳۱- فرض کنید یک آرایه 200عنصری مرتب شده باشد. زمان اجرای بدترین تابع برای پیداکردن عنصر معلوم x در آرایه A با استفاده از جستجوی دودوئی (باینری) چیست؟

۱) 200
۲) 8
۳) 7
۴) 12

۳۲- یک ماتریس بالا مثلث A را با یک آرایه یک‌بعدی B نمایش داده‌ایم. اگر عنصر A[i,j] معادل عنصر B[k] باشد، بین k , j , i چه رابطه‌ای وجود دارد؟

۱) $k=\frac{i(j+1)}{2}$
۲) $k=\frac{j(j-1)}{2}+i$
۳) $(i-1)\left(n-\frac{1}{2}\right)+j$
۴) هیچکدام

۳۳- می‌نیمم و ماکزیمم اعداد ذخیره‌شده در یک آرایه یک‌بعدی n خانه، با چند مقایسه بین اعداد ذخیره شده در خانه‌ها بدست خواهد آمد؟

۱) $\frac{3n}{2}$
۲) $\frac{3n}{2}-2$
۳) $\frac{n}{2}$
۴) $\frac{n+1}{2}$

۳۴- برای پیداکردن یک item مورد نظر در یک لیست مرتب شده شامل 30000 عنصر حداکثر چند مقایسه مورد نیاز است؟

۱) 100
۲) 45
۳) 30
۴) 15

۳۵- حاصلضرب، جمع، تفریق درماتریس اسپارس:

۱) ممکن است اسپارس نباشد.
۲) همواره یک ماتریس اسپارس است.
۳) همواره یک ماتریس 0 است
۴) همواره ماتریس غیراسپارس است.

۳۶- یک ماتریس پائین مثلث A را با یک آرایه یک‌بعدی B نمایش داده‌ایم اگر عنصر A[i,j] معادل عنصر B[k] باشد، بین k , j , i چه رابطه‌ای وجود دارد؟

۱) $k=\frac{i(j+1)}{2}$
۲) $k=\frac{i(i-1)}{2}+j$
۳) $k=i+j$
۴) $k=\frac{i(i-j)}{2}$

۳۷ـ تابع A(m, n) به شکل روبرو را درنظر بگیرید. حاصل A (1 , 3) کدام است؟ (سراسری ۸۲)

$$A(m,n)=\begin{cases} n+1 & \text{اگر } M=0 \\ A(m-1,1) & \text{اگر } N=0 \\ A(m-1,A(m,n-1)) & \text{بقیه حالات} \end{cases}$$

۱) 3 ۲) 4 ۳) 5 ۴) 6

۳۸ـ مجموع مراحل خطوط در برنامه زیر چند است؟

```
float   sum  (int num [ ] , int   n)
      {int  i , temp   = 0
       for ( i = 0   ; i < n  ;  i + + )
            temp + = num  [ i ] ;
            return   temp;
            }
```

۱) $2n+3$ ۲) $2n+1$ ۳) تعداد مراحل بستگی به n دارد و نامشخص است. ۴) $n+1$

۳۹ـ تعداد مراحل کل خطوط در برنامه زیر چقدر است؟

```
float  rsum  (float  list  [  ]  , int n )
      {
      if ( n )
      return  rsum  (list , n – 1) + list   [n – 1];
      return list [  0  ];
      }
```

۱) $2n+4$ ۲) $2n+2$ ۳) $2n^2$ ۴) $2(n-1)^2$

۴۰ـ کدامیک از روابط زیر نشان‌دهنده رابطه صحیح زمان محاسبه الگوریتم‌های مختلف است؟

۱) $o(\log n)<o(n)<o(n\log n)<o(2^n)<o(n^2)$

۲) $o(n)<o(\log n)<o(n\log n)o(2^n)<o(n^2)$

۳) $o(n)<o(\log n)<o(n\log n)<o(n^2)<o(2^n)$

۴) $o(\log n)\,o(n)<o(n\log n)<o(n^2)<o(2^n)$

۴۱ـ با توجه به تابع روبرو func (100) چه خواهد شد؟

```
int  func ( int n )
    if ( n = = 0)  return  0;
    return ( n + func ( n – 1 ) );
     }
```

۱) 199 ۲) 200 ۳) 5050 ۴) 10000

۴۲- الگوریتم عبارتست از مراحل ..... که هر مرحله ..... شده باشد.

۱) حل یک مسأله / به‌خوبی تعریف

۲) نوشتن یک برنامه / به‌خوبی تعریف

۳) نوشتن یک برنامه / از چند دستور تشکیل

۴) حل یک مسأله / از چند مرحله فرعی تشکیل

۴۳- با توجه به تعریف تابع مقابل مقدار $L(25)$ چیست؟

$$L(n)=\begin{cases}0 & \text{if } n=1\\ L\left(\left[\frac{n}{2}\right]\right)+1 & \text{if } n>1\end{cases}$$

۱) 6 ۲) 5 ۳) 4 ۴) 7

۴۴- خروجی تابع بازگشتی زیر در زبان پاسکال به ازاء y = 2 چه می‌شود.

```
function  f(y : integer) : integer;
Begin
    if  y<=0  then  f:=2
    Else f:=f(y-1)+f(y-2);
```

۱) 4 ۲) 8 ۳) 6 ۴) 2

۴۵- دلیل اصلی استفاده از توابع Recursive در برنامه‌سازی چیست؟

۱) ساختار تکرار بعضی از ساختمان داده‌ها

۲) راحتی تدوین برنامه‌های Recursive و تعداد کم دستورات استفاده شده در آن‌ها

۳) هزینه کم از نظر منبع کامپیوتر

۴) همه انتخاب‌ها.

۴۶- فرض کنید که a , b اعداد صحیح مثبت بوده و تابع Q به صورت زیر به شکل بازگشتی تعریف شده باشد. مقادیر $Q(2,3)$, $Q(14,3)$ را پیدا کنید.

$$Q(a,b)=\begin{cases}0 & \text{if } a<b\\ Q(a-b,b)+1 & \text{if } b\le a\end{cases}$$

۱) $Q(2,3)=0, Q(14,3)=0$

۲) $Q(2,3)=0, Q(14,3)=4$

۳) $Q(2,3)=0, Q(14,3)=3$

۴) $Q(2,3)=4, Q(14,3)=3$

۴۷- کدام گزینه یک ساختمان داده نیست؟

۱) گراف جهت‌دار ۲) صف حلقوی ۳) مجموعه ۴) لیست پیوندی

۴۸ـ کدامیک، از خصوصیات یک برنامه نمی‌باشد؟

۱) برنامه می‌تواند پایان‌ناپذیر باشد.

۲) برنامه اغلب دارای m ورودی $(M \geq 1)$، n خروجی $(n \geq 0)$ است.

۳) دستورات برنامه باید بدون ابهام باشد.

۴) دستورات برنامه توسط کامپیوتر باید قابل اجرا باشد.

۴۹ـ الگوریتم‌های بازگشتی چه معایبی دارند؟

۱) اتلاف حافظه، سرعت اجرای کمتر

۲) اتلاف حافظه، طولانی‌بودن کد (source)

۳) سرعت اجرای کمتر، طولانی‌بودن کد

۴) طولانی‌بودن کد، اتلاف حافظه، سرعت اجرای کمتر

۵۰ـ تابع زیر چه خروجی دارد؟

```
void    xyz (void){
        char ch;
        scanf ("%c" , &ch);
        if(ch! ='\n') xyz( );
        printf ("%c" , ch);}
```

۱) یک رشته را چاپ می‌کند.

۲) یک رشته ورودی را در خروجی برعکس چاپ می‌کند.

۳) یک رشته عددی ورودی را در خروجی به‌صورت برعکس چاپ می‌کند.

۴) یک رشته رقمی را از ورودی خوانده و در خروجی چاپ می‌کند.

۵۱ـ کار تابع f بر روی رشته s با n کاراکتر چیست؟ تابع $sub(s,i,n)$ تعداد n کاراکتر از موقعیت i در رشته s را برمی‌گرداند. (سراسری ۸۰)

$$f(s,n)=\begin{cases} s & \text{اگر } n=1 \\ f(sub(s,1,n-1),n-1)+sub(s,n,1) & \text{اگر } n>1 \end{cases}$$

۱) رشته s را برمی‌گرداند.

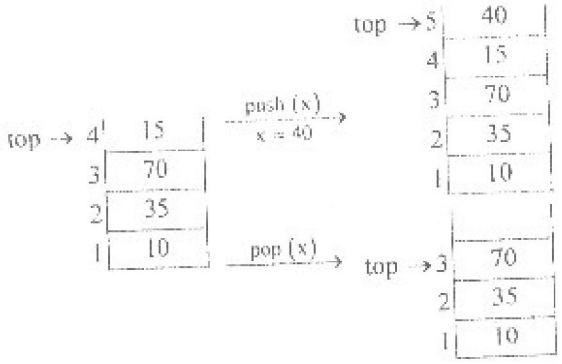
۲) معکوس رشته s را برمی‌گرداند.

۳) یک کاراکتر از انتها به رشته s اضافه می‌کند.

۴) یک کاراکتر به ابتدای رشته s اضافه می‌کند.

Stack (پشته‌ها)

هر لیست پشت سرهمی از داده‌ها که عمل حذف (pop) و درج (push) در آن از یک طرف لیست که همان عنصر بالا (top) نامیده می‌شود انجام می‌گیرد.



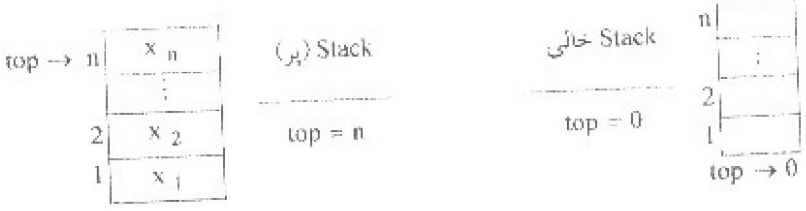
X = 15 عنصر حذف شده

دیده می‌شود که برای درج (push)، ابتدا اشاره‌گر top یک واحد به سمت بالا حرکت کرده، سپس عمل درج انجام می‌شود، و در عمل حذف (pop) ابتدا داده‌ای که top به آن اشاره می‌کند حذف شده سپس top یک واحد به سمت پائین حرکت می‌کند.

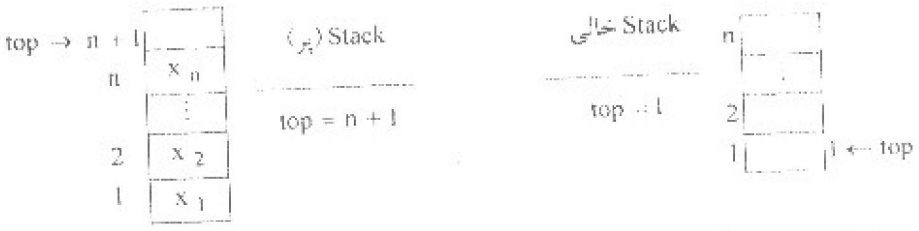
■ وضعیت اشاره‌گر top در شرایط مرزی خالی یا پر بودن stack

در صورتیکه آرایه stack [1 .. n] برای نگهداری عناصر پشته در نظر گرفته شود، در این صورت شرایط مرزی به صورت زیر خواهد بود.

۱- با فرض آن‌که top به خانه پر (آخرین خانه پر) اشاره کند.



۲- با فرض آن‌که top به خانه خالی (اولین خانه خالی) اشاره کند.



نکته مهم: همیشه به طور پیش فرض top به آخرین خانه پر اشاره می‌کند. (حالت 1)

الگوریتم‌های pop (حذف) و push (درج)

با فرض آن‌که top به آخرین خانه پر اشاره می‌کند، در صورتیکه بخواهیم عملیات حذف و درج را در stack (پشته) انجام دهیم می‌توانیم از زیربرنامه‌های زیر استفاده کنیم.

```
procedure   push (item : نوع );
    begin
        if   top = n   then
            write ('stack Full')
        else
            begin
                top : = top + 1;
                stack [top] : = item;
            end;
```

در این الگوریتم در صورتیکه در آرایه stack [1 .. n]، top = n نشده باشد (پشته پر) ابتدا، top به بالا حرکت کرده سپس داده (item) در محل خالی جدید درج می‌شود.

```
procedure   pop (item : نوع );
    begin
        if   top = 0   then
            write ('stack Empty')
        else
            begin
                item : = stack [top];
                top : = top – 1;
            end;
```

در این الگوریتم در صورتیکه در آرایه stack [1 .. n]، top = 0 نباشد (پشته خالی) ابتدا داده‌ای که top به آن اشاره می‌کند داخل item نگهداری شده سپس top یک واحد به سمت پائین حرکت می‌کند.

بهینه سازی قرار گرفتن چند stack در یک آرایه:

در صورتیکه بخواهیم 2 stack را داخل یک آرایه قرار دهیم، این امکان وجود دارد که 2 پشته در حالت‌های مختلفی کنار هم قرار گیرند،

1) 

3) 

2) 

4) 

با توجه به حالت‌های نشان داده شده دیده می‌شود که حالت 1 از تمام حالت‌ها بهتر پیاده‌سازی شده است این به آن علت است که در صورتیکه یکی از دو stack top1 و top 2 درجی نداشته باشند stack دیگر می‌تواند از فضای خالی آن استفاده کند. اما در بقیه حالت‌ها به ترتیب مشکلات زیر وجود دارد:

۱) در این حالت top 1 تا top2 و top 2 از وسط تا انتها می‌تواند حرکت کند.

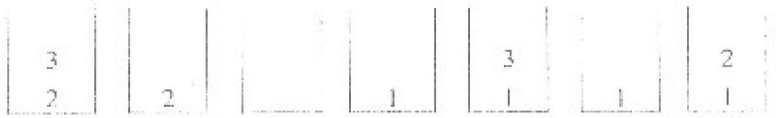
۲) در این حالت top 1 و top2 از وسط حرکت کرده و به ترتیب به ابتدا و انتهای آرایه محدود می‌شوند.

۳) در این حالت top2 از انتها تا top 1 و top 1 از وسط تا ابتدا حرکت می‌کند.

در حالت 1 که بهترین وضعیت است، top 1 و top 2 به سمت هم حرکت می‌کنند و این باعث می‌شود که اگر مثلاً top1 هیچ درجی انجام ندهد top2 تا ابتدای آرایه حرکت کند.

مثال: اگر یک پشته با وضعیت  وجود داشته باشد مطلوبست وضعیت پشته بعد از انجام عملیات زیر از راست به چپ:

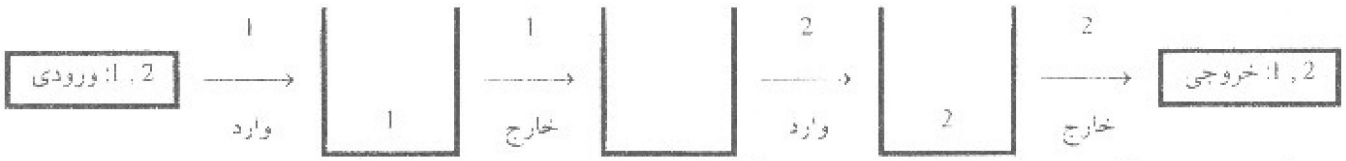
push (z) , push (y) , pop (x) , pop (z) , push (x) , pop (y) , pop (x)



ورودی‌ها و خروجی‌های مجاز برای یک پشته:

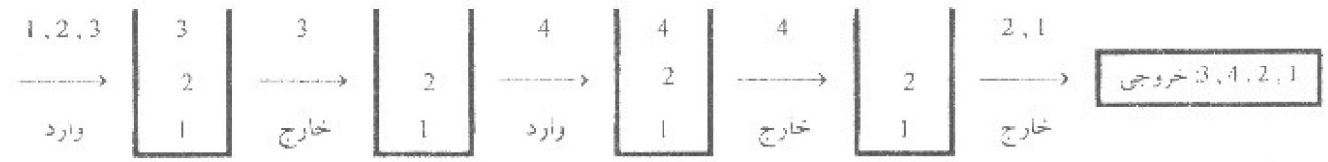
در یک پشته nتائی در صورتیکه داده‌های $a_1 , a_2 , \ldots , a_n$ به ترتیب وارد stack شوند، با رعایت قواعد زیر می‌توانیم خروجی‌های مجاز را تعیین کنیم:

الف) هر داده به محض ورود می تواند از پشته خارج شود.



ب) در هر مرحله هرگاه بخواهیم داده‌ای را در خروجی ببینیم که هنوز وارد stack نشده است می‌بایست داده‌ها را پشت سرهم وارد پشته کنیم تا به داده مورد نظر برسیم سپس داده را وارد پشته کرده و سپس خارج می‌کنیم.

4 , 3 , 2 , 1: ورودی



ج) در هر مرحله هرگاه بخواهیم داده‌ای را در خروجی ببینیم که پائین stack است و عنصر بالای پشته نیست ***این عمل غیرمجاز است.***

ورودی: 1 , 2 , 3 , 4

خروجی: 3 , 4 , 1 , 2

در این مرحله خارج کردن 1 غیرمجاز است چون در پائین پشته قرار دارد.

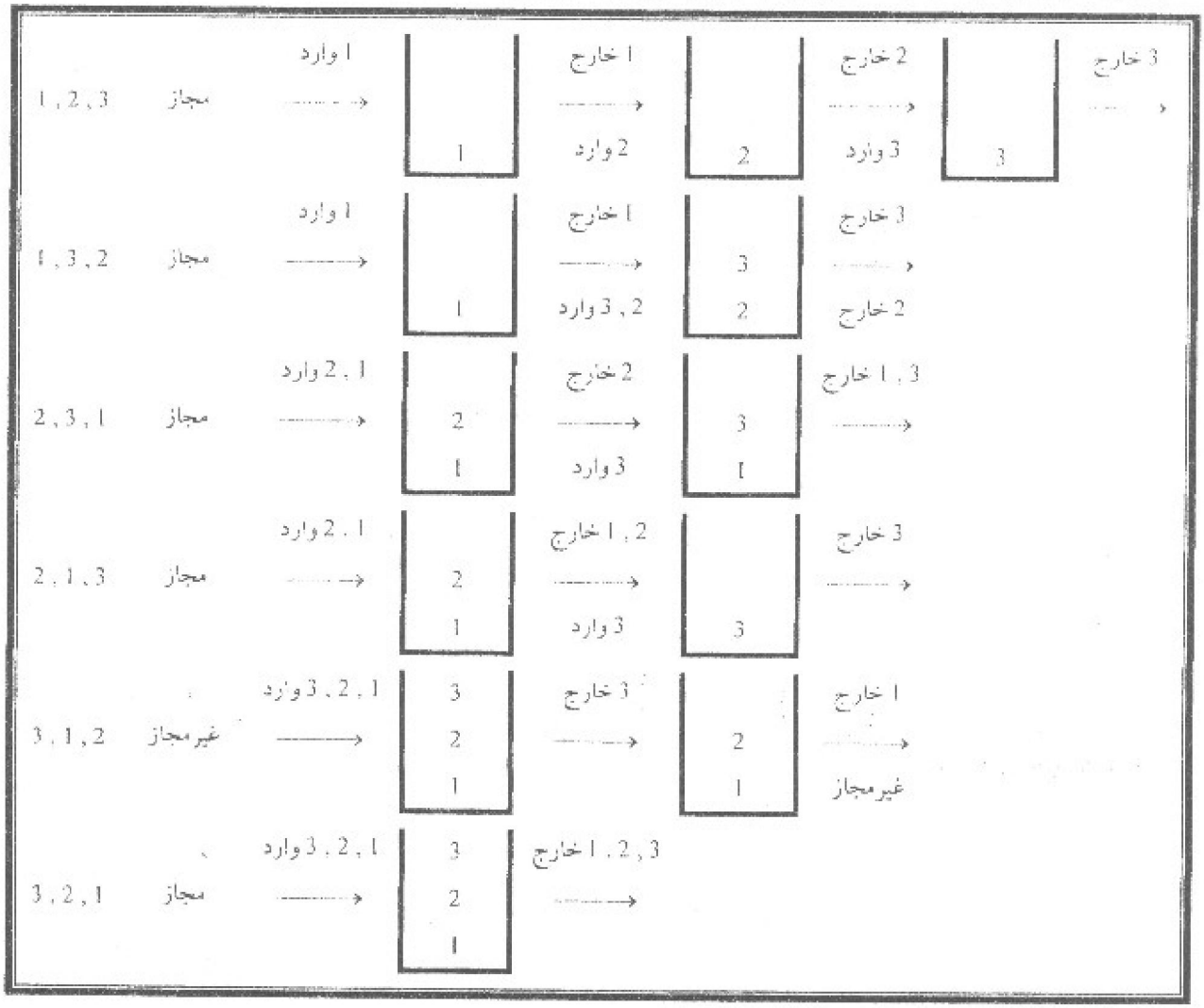
تعداد خروجی‌های مجاز در یک stack nتایی از رابطه زیر بدست می‌آید:

$$\frac{\binom{2n}{n}}{n+1}$$

مثال: در صورتیکه 1 , 2 , 3 به ترتیب وارد یک پشته شوند تعداد حالت‌های مجاز کدام هستند.

$$n = 3 \rightarrow \frac{\binom{6}{3}}{3+1} = 5$$ تعداد حالت‌های مجاز

در صورتیکه ورودی 1 , 2 , 3 باشد خروجی‌های مجاز و غیرمجاز به شرح زیر است:



■ stackهای مجاز برای ورودی های مختلف

یک stack از یک ورودی داده‌ها زمانی مجاز است که داده‌های قبلی روی داده‌های بعدی قرار نگیرند، چرا که داده‌های قبلی یا باید قبلاً وارد پشته شده و از آن خارج شده باشند یا این که خارج نشده باشند و پائین داده‌های بعدی باشند.

ابتدا A ، B وارد شده سپس B خارج شده، C ، D وارد شده سپس D خارج شده آ وارد می‌شود. (مجاز)

ابتدا A وارد سپس خارج شده سپس B ، C ، D ، E وارد شده سپس F ، C ، D خارج شده F وارد می‌شود. (مجاز)

A یا باید پائین B قرار گیرد یا قبلاً از پشته حذف شده باشد. (غیرمجاز)

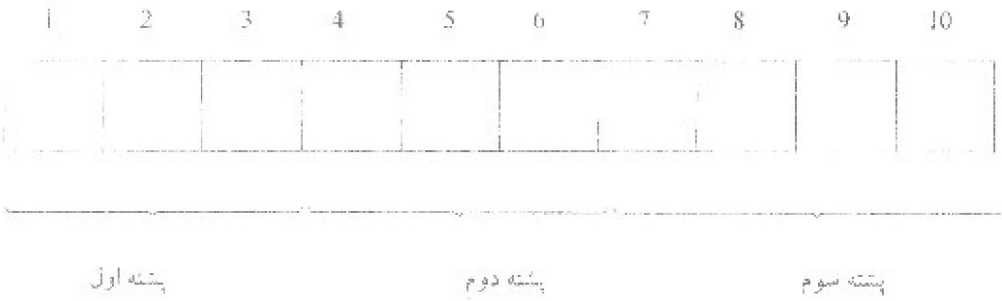
پشته چندگانه

برای نمایش n پشته در یک آرایه mتائی stack [1 .. m]، آرایه مورد نظر را به n قسمت مساوی تقسیم می‌کنیم و به هر پشته خانه‌های مشخص تخصیص داده می‌شود.

نکته: سعی داریم که فضای m خانه آرایه را به طور مساوی بین n پشته تقسیم کنیم در این حالت اگر n مضربی از m نباشد، فضای اضافی از آرایه m تائی به پشته آخر می‌رسد.

مثال: 3 پشته در آرایه 10تائی:

در این حالت برای هر پشته $\left\lfloor \frac{10}{3} \right\rfloor = 3$ خانه در نظر گرفته می‌شود اما یک خانه اضافی به پشته آخر می‌رسد.



پیاده‌سازی پشته‌ها

برای هر پشته دلخواه آ ام از دو اشاره‌گر h [i] (اشاره به پائین) و t [i] (اشاره به بالای) پشته استفاده می‌شود.

- اشاره‌گر b [i] برای این استفاده می‌شود که اشاره t [i - 1] مربوط به stack i ام به پشته i ام برخورد نکند.

شرایط مرزی برای stack ام i : (مهم)

۱) پشته i ام خالی $b[i] = t[i] = (i-1)\left\lfloor \frac{m}{n} \right\rfloor + 1 \qquad i = 1, 2, \ldots, n$

۲) پشته i ام پر $t[i] = b[i+1]$

مثال: می‌خواهیم در یک آرایه 10 تائی 3 پشته را قرار دهیم مطلوبست شرایط اولیه برای هر پشته؟

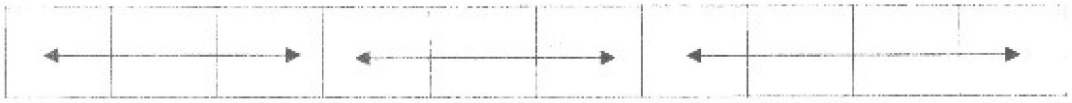
$$\left.\begin{matrix} m = 10 \\ n = 3 \end{matrix}\right\} \longrightarrow \begin{cases} i = 1 \rightarrow b[1] = t[1] = (1-1)\left\lfloor \frac{10}{3} \right\rfloor + 1 - 1 \\ i = 2 \rightarrow b[2] = t[2] = (2-1)\left\lfloor \frac{10}{3} \right\rfloor + 1 = 4 \\ i = 3 \rightarrow b[3] = t[3] = (3-1)\left\lfloor \frac{10}{3} \right\rfloor + 1 = 7 \end{cases}$$

$\left\lfloor \frac{m}{n} \right\rfloor = \left\lfloor \frac{10}{3} \right\rfloor = 3$

تعداد خانه‌های هر پشته

b[1]	b[2]	b[3]
t[1]	t[2]	t[3]

1 2 3 4 5 6 7 8 9 10



دیده می‌شود که یک خانه اضافی در انتهای آرایه به stack آخر (سوم) تعلق گرفته است.

ـ عملیات push (درج)

```
procedure  push (item : نوع);
if   t[i] = b[i + 1]   then
     write ('Full')
else      begin
            stack [t[i]] : = item;
            t[i] : = t[i] - 1;
          end;
     end;
```

چون در ابتدا $t[i]$ به خانه خالی اشاره می‌کند برای درج (push)، ابتدا عمل درج انجام می‌شود سپس $t[i]$ یک واحد به بالا حرکت می‌کند.

عملیات pop (حذف)

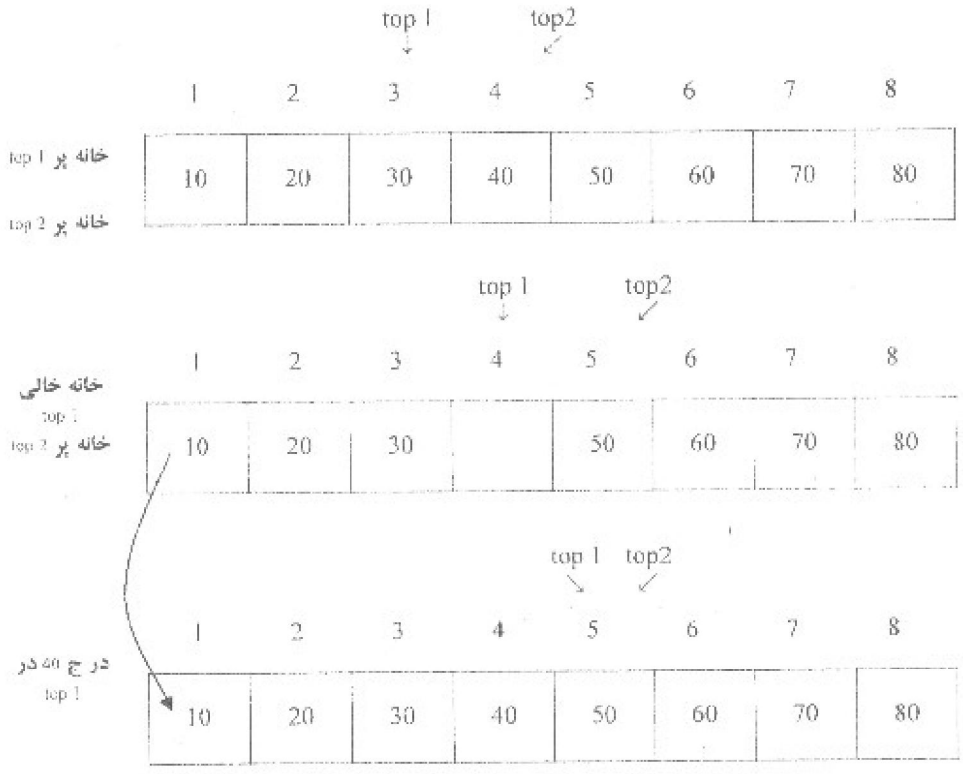
```
procedure   pop (item : داده );
    begin
    if   b [i] = t [i]   then
        write ('Empty')
    else
        begin
            t [i]: = t [i] – 1;
            item : = stack [t [i]];
        end;
```

چون در ابتدا t [i] 1 به خانه خالی اشاره می‌کند برای حذف (pop) ابتدا t [i] یک واحد به پائین حرکت می‌کند تا به خانه پر برسد سپس داده مورد نظر حذف شده و در item قرار می‌گیرد.

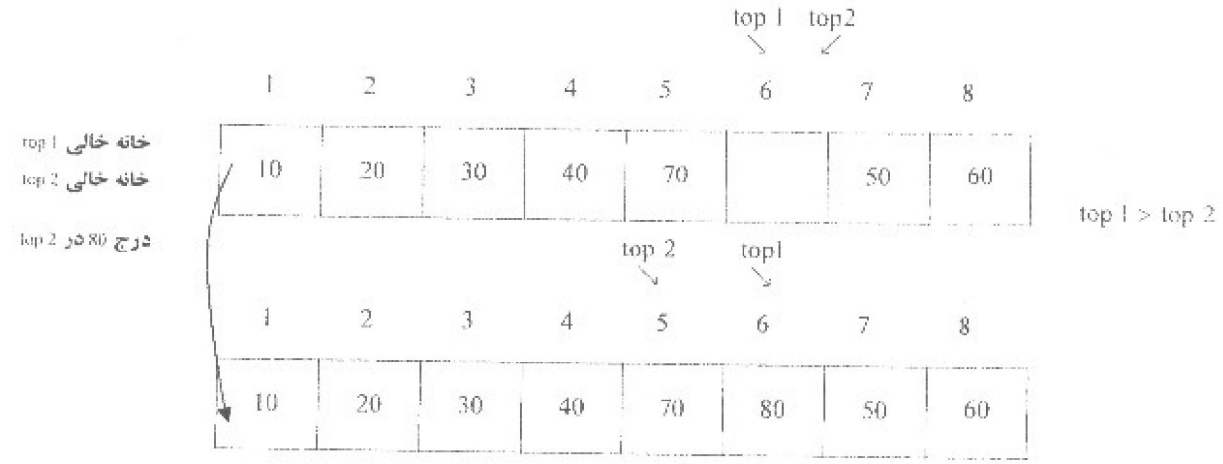
کاربرد پشته‌ها (مهم)

۱ـ استفاده در توابع بازگشتی (برج هانوی ـ تابع Ackerman ـ دنباله Fibonachi ـ ....)
۲ـ ارزیابی عبارت‌های محاسباتی (prefix – postfix – infix)
۳ـ الگوریتم مسیر حرکت Maze
۴ـ پیمایش عمقی درخت‌ها و گراف‌ها
۵ـ استفاده در روش‌های مرتب‌سازی quicksort و mergesort

■ وضعیت‌های مرزی در قرار گرفتن 2 پشته در یک لیست

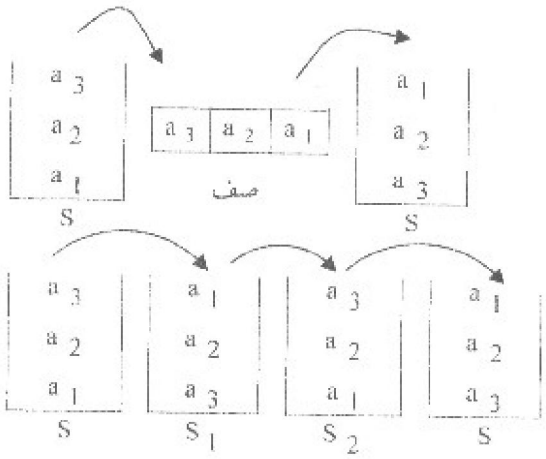


دقت می کنیم که در وضعیتی که top1 به خانه خالی اشاره می‌کند برای درج ابتدا عمل درج انجام شده سپس ۱ واحد به آن اضافه می‌شود که در این مثال بعد از عمل درج top 1 = top 2 = 5 می‌شود.

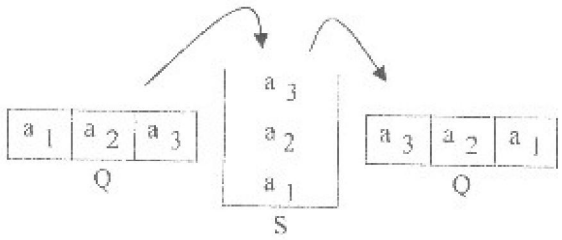


نکات مهم در مورد stack و صف:

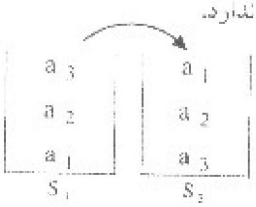
۱ـ می‌خواهیم یک stack را داخل خودش معکوس ببینیم از چه فضای اضافه‌ای استفاده کنیم: از یک صف اضافی یا از دو stack اضافی می‌توان استفاده کرد. که استفاده از صف مقرون به صرفه تر است.



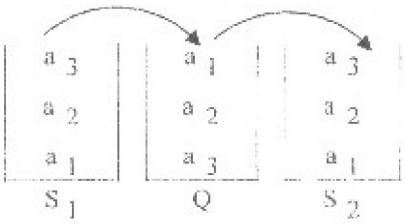
۲ـ در صورتی که بخواهیم عناصر یک صف را داخل خودش معکوس کنیم از چه فضای اضافه‌ای استفاده کنیم: از یک stack اضافی می‌توان استفاده کرد.



۳ـ می‌خواهیم یک stack را در stack دیگر معکوس ببینیم از چه فضای اضافه‌ای استفاده کنیم: هیچ فضایی نیاز ندارد.

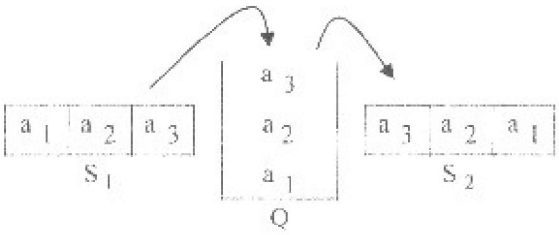


۴ـ می‌خواهیم یک stack را در stack دیگر بدون هیچ تغییر ببینیم از چه فضای اضافه‌ای استفاده کنیم: از یک stack اضافی می توان استفاده کرد.



۵ـ می‌خواهیم یک صف را داخل صف دیگر به همان شکل ببینیم از چه فضای اضافه‌ای استفاده کنیم: به فضای اضافی نیاز نداریم.

۶ـ می‌خواهیم یک صف را داخل صف دیگر معکوس ببینیم از چه فضای اضافه‌ای استفاده کنیم: از یک stack اضافی استفاده می کنیم.

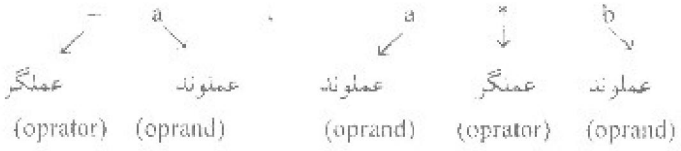


ارزشیابی عبارت‌های محاسباتی

هر عبارت محاسباتی با توجه به اولویت عملگرهایش قابل ارزیابی و محاسبه است.

یادآوری ۱:

در عبارت محاسباتی



یادآوری ۲:

با توجه به یادآوری ۱ دیده می‌شود که در عبارت‌های محاسباتی دو نوع عملگر وجود دارد:

۱ـ یکانی (unary)

۲ـ دوتایی (binary)

عملگرهای یکانی مانند: ـ (منفی) و + (مثبت) فقط به یک عملوند نیاز دارند.

عملگرهای دودوئی مانند: * (ضرب)، / (تقسیم)، + (جمع) و ـ (تفریق) و ^ (توان) به دو عملوند نیاز دارند.

اولویت عملگرها:

به طور کلی صرف‌نظر از هر زبان برنامه‌سازی اولویت عمومی عملگرها را می‌توان به شرح زیر در نظر گرفت:

اولویت	نام عملگر	
۱	( )	
۲	ـ (منفی)، + (مثبت)	
۳	^ یا ** (توان)	← اولویت توان‌های متوالی از راست به چپ
۴	* (ضرب)، / (تقسیم)	← هم اولویت، اولویت از چپ به راست
۵	+ (جمع)، ـ (تفریق)	← هم اولویت، اولویت از چپ به راست

نکته مهم: در هر عبارت محاسباتی اگر تعداد عملوندها ۱ واحد بیشتر از تعداد عملگرها باشد، در آن عبارت تمام عملگرها دودوئی است:

a * b + c → 3 عملوند و 2 عملگر

a * b - c / d → 4 عملوند و 3 عملگر

در عین حال اگر تعداد عملوندها و تعداد عملگرها با هم مساوی باشد یا تعداد عملگرها بیشتر از عملوندها باشد، در آن عبارت حتماً عملگر یکانی وجود دارد:

- a * b → 2 عملوند و 2 عملگر

- a * - b → 2 عملوند و 3 عملگر

مثال: مطلوبست اولویت‌بندی عبارت‌های زیر:

a * – b / c ↑ d + e
1 2
3
4
5

a * (b + c) – k / d + e
1 3
2
4
5

a * b – c ^ d ^ e + f
3 1
2
4
5

a / (b + c) – e * (f + g)
1 2
3 4
5

a – b * c / d + c ^ f ^ g – h
1 1
4 2
5
6
7

■ در هر عبارت محاسباتی اگر تعداد عملگرها از تعداد عملوندها بیشتر باشد

1 + تعداد عملوندها ـ تعداد عملگرها = تعداد عملگرهای یکانی

تنها عملگر یکانی ـ (منفی) است. تعداد عملگرهای یکانی = 3 – 3 + 1 = 1 ← – a * b + c → یکانی

تکرار عملگرهای یکانی — (منفی) 2 بار است. تعداد عملگرهای یکانی = 4 – 3 + 1 = 2 ← – a ^ – b + c → یکانی

یکانی

■ **درخت عبارت محاسباتی (درخت پارس)**

برای تشکیل درخت هر عبارت محاسباتی به صورت زیر عمل می‌کنیم:

۱ـ ابتدا عبارت را برحسب اولویت عملگرهایش اولویت‌بندی می کنیم.

۲ـ ریشه درخت، کم اولویت‌ترین از نظر اولویت است .

۳ـ ریشه زیر درختان چپ و راست به ترتیب کم اولویت‌ترین عملگر از نظر اولویت در چپ و راست ریشه درخت است.

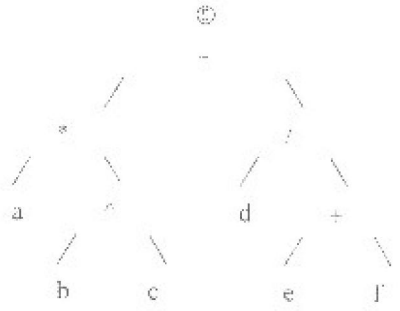
۴ـ برگ‌های درخت عملوندها هستند.

مثال :

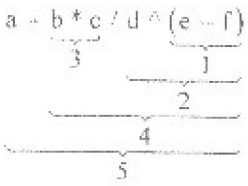
a * b – c / d
1 2
3

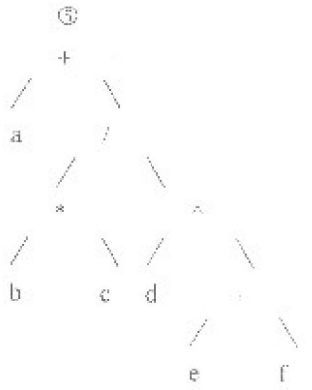
کم اولویت‌ترین عملگر ـ (تفریق) است که در ریشه قرار گرفته است.

مثال ۲:

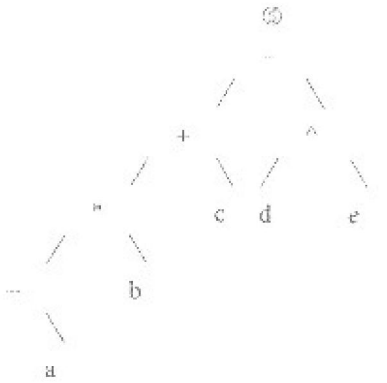


مثال ۳:





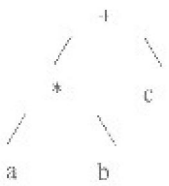
مثال ۴:



دیده می‌شود که عملوند مربوط به عملگر یکانی (ـ)، یعنی a فرزند راست آن قرار گرفته است.

نکته: تعداد گره‌های درخت عبارت محاسباتی در صورتیکه همه عملگرها دودویی باشد فرد است.

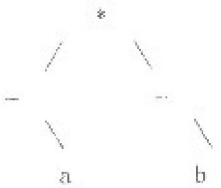
a * b + c



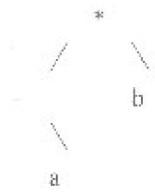
درخت روبرو 5 گره دارد.

تعداد گره‌های درخت عبارت محاسباتی در صورتیکه به تعداد فرد عملگر یکانی داشته باشد زوج و در صورتیکه به تعداد زوج عملگر یکانی داشته باشد فرد است.

درخت روبرو 5 گره دارد.



درخت روبرو 4 گره دارد.



عبارات infix (میانوندی) ـ postfix (پسوندی) ـ prefix (پیشوندی)

عبارت infix (میانوندی):

در هر عبارت عمومی محاسباتی هر عملگر دودوئی بین عملوندهایش قرار می‌گیرد، که به این نحوه قرار گرفتن عملگرها در بین عملوندها روش میانوندی گفته می‌شود.

عملگر * بین عملوندهای a , b قرار دارد. ← a * b

نکته ۱: در هر عبارت میانوندی (infix) اولویت عملگرها مطرح بوده و از ( ) برای تغییر اولویت عملگرها استفاده می‌شود.

نکته ۲: در هر عبارت میانوندی (infix) اولویت عملگرها باتوجه به جدول اولویت عملگرها مشخص می‌شود.

عبارت postfix (پسوندی):

در این روش هر عملگر بعد از عملوندهایش قرار می‌گیرد.

عملگر * بعد از عملوندهایش قرار گرفته است. ← ab *

نکته ۱: در عبارت‌های postfix، اولویت عملگرها مطرح نبوده و از ( ) نیز استفاده نمی‌شود.

نکته ۲: به روش postfix روش لهستانی معکوس یا polish وارونه یا RPN (Reverse polish Notation) نیز می‌گویند.

عبارت prefix (پیشوندی):

در این روش هر عملگر قبل از عملوندهایش قرار می‌گیرد.

عملگر * قبل از عملوندهایش قرار گرفته است. ← * ab

نکته ۱: در عبارت‌های prefix، اولویت عملگرها مطرح نبوده و از ( ) نیز استفاده نمی‌شود.

نکته ۲: به روش prefix روش لهستانی یا روش polish نیز می‌گویند چون اولین بار این روش توسط ریاضی‌دان لهستانی معرفی شد.

برای همین به روش postfix روش لهستانی وارونه گفته می‌شود.

نتیجه‌گیری: (مهم)

۱ـ در تبدیل عبارات infix و postfix و prefix به یکدیگر ترتیب عملوندها به هیچ شکل عوض نمی‌شود.

مثال:

a * b + c	infix
a b * c +	postfix
+ * a b c	prefix

دیده می‌شود که در هر سه روش ترتیب عملوندهای a , b , c تغییری نکرده است اما ترتیب عملگرها ممکن است عوض شود.

۲ـ در هر عبارت postfix همیشه سمت راست عبارت عملگر است.

۳ـ در هر عبارت prefix همیشه سمت چپ عبارت عملگر است.

تبدیل عبارت infix به عبارت‌های postfix و prefix

برای این عمل سه روش وجود دارد.
- ۱ـ روش پرانتزگذاری
- ۲ـ استفاده از stack
- ۳ـ پیمایش درخت عبارت محاسباتی

۱ـ روش پرانتزگذاری

تبدیل infix به postfix:

الف) عبارت را به طور کامل برحسب اولویت عملگرها پرانتزگذاری می‌کنیم در حین پرانتزگذاری عملگر مربوط به هر پرانتز را روی پرانتز بسته می‌گذاریم.

ب) عبارت را از چپ به راست (شامل عملوندها و عملگرهای روی پرانتزهای بسته) در خروجی می‌نویسیم.

■ قبل از قسمت (الف) ابتدا وضعیت پرانتزهای عبارت را مشخص می‌کنیم.

مثال ۱:

$$a * b + c \rightarrow \left((a * b)' + c\right)' \longrightarrow a\ ,\ b\ ,\ *\ ,\ c\ ,\ +$$

مثال ۲:

$$a * (b + c) - g / d \longrightarrow \left(\left(a * (b + c)'\right)' - (g / d)'\right)' \rightarrow a\ ,\ b\ ,\ c\ ,\ +\ ,\ *\ ,\ g\ ,\ d\ ,\ /\ ,\ -$$

مثال ۳:

$$a + b \uparrow c * d - e \rightarrow \left(\left(a + \left((b \uparrow c)' * d\right)'\right)' - e\right)' \rightarrow a\ ,\ b\ ,\ c\ ,\ \uparrow\ ,\ d\ ,\ *\ ,\ +\ ,\ e\ ,\ -$$

مثال ۴:

$$\sqrt{b^2 - 4ac} = (b \uparrow 2 - 4 * a * c) \uparrow (1 / 2) \rightarrow \left(\left((b \uparrow 2)' - \left((4 * a)' * c\right)'\right)' \uparrow (1 / 2)'\right)'$$

$$\rightarrow b\ ,\ 2\ ,\ \uparrow\ ,\ 4\ ,\ a\ ,\ *\ ,\ c\ ,\ *\ ,\ -\ ,\ 1\ ,\ 2\ ,\ /\ ,\ \uparrow$$

مثال ۵:

$$a * b + c \hat{} d \hat{} e - f \rightarrow \left(\left((a * b)' + \left(c \hat{} (d \hat{} e)' \right)' \right)' - f \right)'$$

$$\rightarrow a , b , * , c , d , e , \hat{} , \hat{} , + , f , -$$

مثال ۶:

$$a * - b \hat{} c \hat{} d - e / f \rightarrow \left(\left(a * \left((-b) \hat{} (c \hat{} d)' \right)' \right)' - (e / f)' \right)'$$

$$\rightarrow a , b , - , c , d , \hat{} , \hat{} , * , e , f , / , -$$

تبدیل infix به prefix:

الف) عبارت را به طور کامل برحسب اولویت عملگرها پرانتزگذاری می‌کنیم در حین پرانتزگذاری عملگر مربوط به هر پرانتز را روی پرانتز باز می‌گذاریم.

ب) عبارت را از چپ به راست (شامل عملوندها و عملگرهای روی پرانتزهای باز) در خروجی می‌نویسیم.

■ قبل از قسمت (الف) ابتدا وضعیت پرانتزهای عبارت را مشخص می‌کنیم.

مثال ۱:

$$a * b - c \rightarrow \left((a * b) - c \right) \rightarrow - , * , a , b , c$$

مثال ۲:

$$a * (b + c) - g / d \rightarrow \left((a * (b + c)) - (g / d) \right) \rightarrow - , * , a , + , b , c , / , g , d$$

مثال ۳:

$$a + b \hat{} c * d - e \rightarrow \left((a + ((b \hat{} c) * d)) - e \right) \rightarrow - , + , a , * , \hat{} , b , c , d , e$$

مثال ۴:

$$\sqrt{b^2 - 4ac} = (b \hat{} 2 - 4 * a * c) \hat{} (1 / 2) \rightarrow \left((b \hat{} 2) - ((4 * a) * c) \right) \hat{} (1 / 2)$$

$$\rightarrow \hat{} , - , \hat{} , b , 2 , * , * , 4 , a , c , / , 1 , 2$$

مثال ۵:

$$a * b + c \hat{} d \hat{} e - f \rightarrow \left(\left((a * b) + (c \hat{} (d \hat{} e)) \right) - f \right)$$

$$\rightarrow - , + , * , a , b , \hat{} , c , \hat{} , d , e , f$$

مثال ۶:

$$a * - b \hat{} c \hat{} d - e / f \rightarrow \left(\left(a * ((-b) \hat{} (c \hat{} d)) \right) - (e / f) \right)$$

$$\rightarrow - , * , a , \hat{} , - , b , \hat{} , c , d , / , e , f$$

■ ارزشیابی عبارت‌های postfix (پسوندی) و prefix (پیشوندی) مهم:

برای ارزشیابی عبارت‌های Postfix و prefix و تعیین مقدار نهائی آن‌ها به صورت زیر عمل می‌کنیم:

الف) ابتدا کنترل می‌کنیم که آیا تعداد عملوندها یک واحد از عملگرها بیشتر است یا خیر چون در غیر این صورت در عبارت عملگر یکانی وجود داشته و ارزشیابی آن شرایط خاصی خواهد داشت.

ب) عبارت را از سمتی بررسی می‌کنیم که با عملوند شروع می‌شود در postfix از چپ و در prefix از راست عبارت را بررسی می‌کنیم.

ج) با رسیدن به هر عملگر دو عملوند بر ی آن در نظر می‌گیریم (در postfix، سمت چپ عملگر و در prefix سمت راست عملگر)

د) باتوجه به عملگر عملیات را روی عملوندها انجام می‌دهیم (چپ به راست)

مثال ۱: از سمت عملوندها (چپ)

12 , 3 , / , 2 , *

→ ((12 , 3 , /) , 2 , *)

مثال ۲: از سمت عملوندها (چپ)

2 , 4 , * , 8 , + , 10 , 2 , − , /

→ (((2 , 4 , *) , 8 , +) , (10 , 2 , −) , /)

مثال ۳: از سمت عملوندها (چپ)

→ 12 , 7 , 3 , − , / , 2 , 1 , 5 , + , * , +

→ ((12 , (7 , 3 , −) , /) , (2 , (1 , 5 , +) * , +)

prefix:

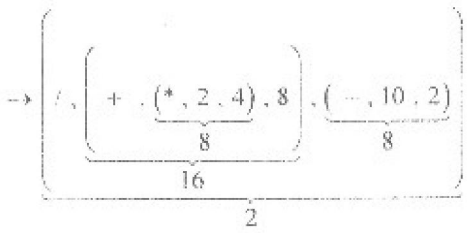
مثال ۱: از سمت عملوندها (راست)

* , / , 12 , 3 , 2

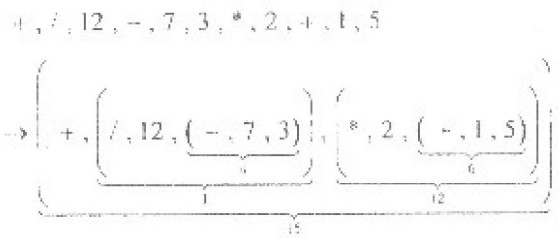
→ (* , (/ , 12 , 3) , 2)

مثال ۲: از سمت عملوندها (راست)

/ , + , * , 2 , 4 , 8 , − , 10 , 2



مثال ۳: از سمت عملوندها (راست)

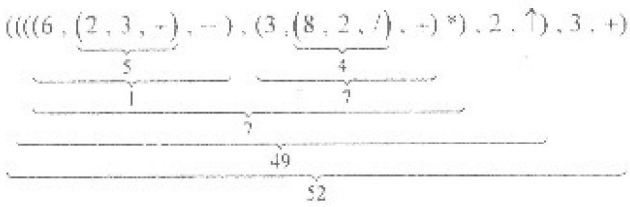


مثال ۴:

مطلوبست ارزش عبارت پسوندی زیر: (کنکور ۸۲)

6 , 2 , 3 , + , − , 3 , 8 , 2 , / , + , * , 2 , ↑ , 3 , −

۱) 48 ✓ ۲) 52 ۳) 34 ۴) 7



حالت خاص: (مهم)

زمانیکه در یک عبارت محاسباتی (prefix – postfix – infix)، تعداد عملگرها بیشتر یا مساوی تعداد عملوندها باشد. حتماً در آن عبارت محاسباتی عملگر یکانی (unary) وجود دارد.

یادآوری:

در هر عبارت محاسباتی تعداد عملگرهای یکانی همیشه از رابطه زیر بدست می‌آید:

(*) 1 + تعداد عملوندها - تعداد عملگرها = تعداد عملگرهای یکانی

در صورتیکه در عبارت محاسباتی همه عملگرها دودوئی باشد،

1 + تعداد عملگرها = تعداد عملوندها

و در نتیجه باتوجه به رابطه (*) تعداد عملگرهای یکانی 0 است.

مثال:

(1

infix : a * b + c
prefix : + * a b c
postfix : a b * c +

→ تعداد عملگرها = 2
تعداد عملوندها = 3

→ تعداد عملگر یکانی = 2 − 3 + 1 = 0

در صورتیکه در عبارت محاسباتی تعداد عملگرها = تعداد عملوندها باشد، باتوجه به رابطه (*) تعداد عملگرهای یکانی 1 است.

(II)

infix : − a − b − c
prefix : − − a b c
postfix : a − b − c

→ تعداد عملگرها = 3
تعداد عملوندها = 3

→ تعداد عملگرهای یکانی = 3 − 3 + 1 = 1

نکته مهم: همانطور که در مثال ۲ دیده می‌شود، به راحتی در عبارت Infix می‌توان فهمید که کدام عملگر یکانی است (a-) اما با مشاهدهٔ prefix یا postfix نمی‌توان وضعیت عملگر یکانی را به راحتی مشخص کرد، مگر آن‌که در عبارت فقط یک عملگر ـ یا + وجود داشته باشد.

دقت کنید که به طور مثال اگر عبارت infix به صورت (a − b) − c − مطرح می‌شد، عبارت prefix آن باز هم a b c − − − می‌شد و این نشان می‌دهد که از روی عبارت prefix یا postfix به راحتی نمی‌توان وضعیت عملگر یکانی را در صورت وجود تعیین کرد.

مثال:

$-13 = -10 - 3 \xrightarrow{\text{prefix}} -\ ,\ -\ ,10\ ,\ 3$

$-7 = -(10 - 3) \xrightarrow{\text{prefix}} -\ ,\ -\ ,\ 10\ ,\ 3$

در مثال بالا دیده می‌شود که با آن‌که عبارت prefix هر دو عبارت infix یکسان است، اما نتایج در حالت infix متفاوت است.

نکته: در خوشبینانه‌ترین حالت می‌توانیم امیدوار باشیم که در عبارت فقط یک عملگر ـ یا + باشد تا در صورتیکه (تعداد عملوندها = تعداد عملگرها) باشد آن‌گاه آن عملگر ـ یا + همان عملگر یکانی است. در عین حال تشخیص وضعیت عملگرهای یکانی در عبارت prefix و postfix به راحتی انجام نمی‌شود.

مثال: حاصل عبارت 3 , 2 , − , 5 , / , 10 , / , 1 , 1 , − , 3 , 2 , ↑ , * , − , − کدام است؟ (۸۳)

۱) −9 ۲) −2

۳) ✓ 1 ۴) 10

دیده می‌شود که تعداد عملگرها = 8 و تعداد عملوندها = 8 در نتیجه:

تعداد عملگر یکانی = 8 − 8 + 1 = 1

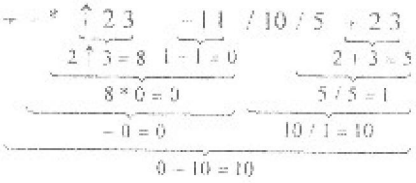
اگر عملگر − قبل از 1 به صورت یکتایی فرض کنیم:

$+ - * \underbrace{\uparrow 2\ 3}_{2 \uparrow 3 = 8} \underbrace{-1}_{8 * (-1) = -8}\ \underbrace{1 / 10 / 5}_{10 / 1 = 10}\ \underbrace{+ 2\ 3}_{5 / 5 = 1}$

$-8 - 1 = -9$

$-9 + 10 = 1$

و اگر عملگر − قبل از 1 را به صورت دوتایی فرض کنیم و − قبل از * را یکتایی فرض کنیم:



تبدیل infix به postfix و prefix با استفاده از پشته:

۱ـ infix به postfix: برای این تبدیل به ترتیب زیر عمل می‌کنیم:

الف) عبارت را از سمت چپ پیمایش می‌کنیم.

ب) عملوندها را در خروجی می‌نویسیم.

ج) به هر پرانتز '(' که رسیدیم آن‌را به راحتی داخل Stack قرار می‌دهیم.

د) به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای stack بیشتر باشد، آن را داخل stack قرار می‌دهیم، در غیر این صورت آن‌قدر از بالای stack عملگر خارج کرده و در خروجی می‌نویسیم تا یا stack خالی شده و یا به عملگری برسیم که بتوانیم عملگر مورد نظر را روی آن در stack قرار دهیم.

یادآوری:

باتوجه به مورد (د)، عملگر + نمی‌تواند روی + یا - قرار گیرد، همین‌طور - روی + یا - و / روی * یا / و * روی * یا / نمی‌توانند قرار گیرند. اما توان (↑) روی توان (↑) می‌تواند قرار گیرد؛ چون اولویت توان‌های پشت سرهم از راست به چپ بررسی می‌شود.

هـ) اگر بالای stack پرانتز '(' باشد هر عملگری به راحتی روی آن قرار می‌گیرد.

و) به هر پرانتز '(' که رسیدیم آن‌قدر از stack عملگر خارج کرده و در خروجی می‌نویسیم تا به '(' برسیم در این وضعیت '(' و ')' با هم خنثی می‌شوند.

ز) زمانیکه به انتهای عبارت رسیدیم در صورتیکه stack خالی نباشد، آن را به طور کامل در خروجی می‌نویسیم.

نکته ۱: در این تبدیل (تعداد pop = تعداد push) بوده و تعداد آن از رابطه زیر بدست می‌آید:

تعداد پرانتزهای '(' + تعداد عملگرها = تعداد Push = تعداد pop

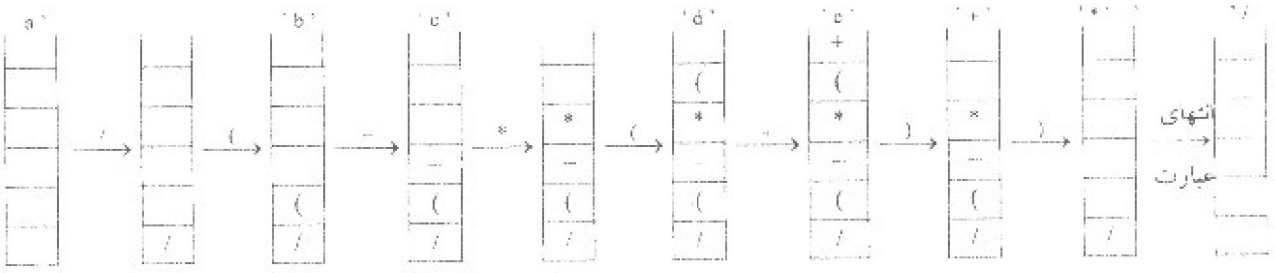
علت برابری تعداد pop و push این است که هر عملگر یا پرانتز '(' که وارد stack می‌شود (push) بالاخره باید از stack خارج شود. (pop)

نکته مهم: حداکثر فضائی که در طول اجرای برنامه استفاده می‌شود، برابر حداقل فضائی است که برای تبدیل مورد نیاز است.

مثال ۱: حداقل اندازهٔ پشته برای تبدیل عبارت میانوندی $a / (b - c * (d + e))$ به معادل پسوندی (postfix) کدام است؟

۱) 4 ۲) 5 ✓ ۳) 6 ۴) 3

از چپ به راست عبارت را پویش می‌کنیم.

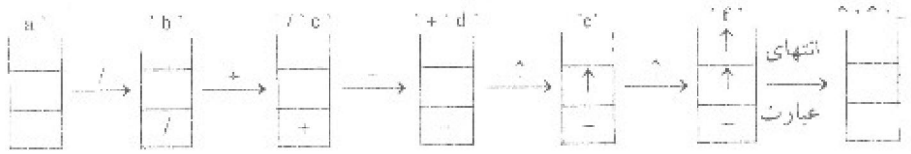


نتیجه را از چپ به راست می‌نویسیم: / ، - ، * ، (، + ،) ،) a , b , c , d , e , f

حداقل فضای مورد نیاز 6 بوده که همان حداکثر فضای استفاده شده است.

تعداد pop = تعداد push = تعداد عملگرها + تعداد (= 4 + 2 = 6

مثال ۲: حداقل اندازه پشته برای تبدیل عبارت میانوندی a / b – c – d ↑ e ↑ f به پسوندی کدام است؟



نتیجه از چپ به راست: ↑ ، ↑ ، - ، f ، e ، d ، + ، c ، / ، b ، a خواهد بود.

حداقل فضای مورد نیاز 3 بوده که همان حداکثر فضای مورد استفاده است.

تعداد pop = تعداد push = تعداد عملگرها + تعداد (= 5 + 0 = 5

مثال ۳: حداقل فضای پشته مورد نیاز (میانی) برای تبدیل عبارات میانوندی زیر به معادل پسوندی (postfix) کدام است؟

الف) عبارت میانوندی = a + b * (c – (d + e)) ^ m ^ n + k

ب) عبارت میانوندی = a – b – c * ((d – e ^ f) + g ^ h)

۳ـ تبدیل prefix به infix:

الف) عبارت را از سمت راست پیمایش می‌کنیم.

ب) عملوندها را در خروجی می‌نویسیم.

ج) به هر پرانتز ')' که رسیدیم آن‌را به راحتی داخل stack قرار می‌دهیم.

د) به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای stack بیشتر یا مساوی باشد، آن را داخل stack قرار می‌دهیم؛ در غیر این صورت آن قدر از بالای stack عملگر خارج کرده و در خروجی می‌نویسیم تا یا stack خالی شده و یا به عملگری برسیم که بتوانیم عملگر مورد نظر را روی آن در stack قرار دهیم.

یادآوری:

باتوجه به مورد (د)، عملگر ـ و + می‌توانند روی + یا ـ قرار گیرند، همین‌طور * و / می‌توانند روی * یا / قرار گیرند. اما توان ↑ روی توان ↑ نمی‌تواند قرار گیرد. چون اولویت توان‌های پشت سرهم از راست به چپ بررسی می‌شود.

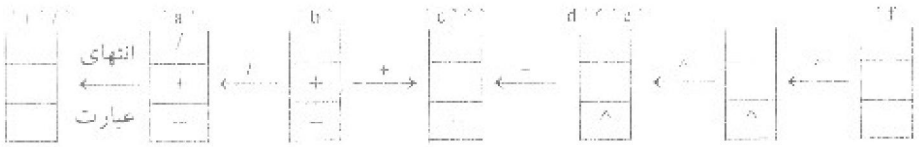
ها، اگر بالای stack پرانتز '(' باشد هر عملگری به راحتی روی آن قرار می‌گیرد.

و) به هر پرانتز ')' که رسیدیم آن‌قدر از stack عملگر خارج کرده و در خروجی می‌نویسیم تا به '(' برسیم در این وضعیت '(' و ')' با هم خنثی می‌شوند.

ز) زمانیکه به ابتدای عبارت رسیدیم (چون عبارات را از راست به چپ بررسی می‌کنیم) در صورتیکه stack خالی نباشد، آن را به طور کامل در خروجی می‌نویسیم.

نکته: تنها مورد متفاوت (مهم) در دو روش infix به postfix و infix به prefix به قسمت (د) در هر دو روش مربوط می‌شود. زمانیکه تبدیل به postfix را بررسی می‌کنیم چون عبارت از چپ به راست پویش می‌شود. به طور مثال در a + b - c، - نمی‌تواند روی + قرار گیرد چون اولویت عملگرهای هم اولویت از چپ به راست است و بنابراین اولویت + از - بیشتر است. اما در تبدیل prefix چون عبارت از راست به چپ پویش می‌شود، در a + b - c، + می‌تواند روی - قرار گیرد چون اولویتش از - بیشتر است. برای همین در تبدیل prefix عملگرهای هم اولویت می‌توانند روی هم قرار گیرند، البته به استثناء ↑ که اولویت توان‌های متوالی از راست به چپ بوده و نمی‌تواند در این وضعیت روی هم قرار گیرند.

مثال ۱: حداقل اندازهٔ پشته برای تبدیل عبارت میانوندی a / b + c - d ^ e ^ f به عبارت پیشوندی معادل کدام است؟



نتیجه از چپ به راست: - + / a b c ^ d ^ e f خواهد بود.

تشخیص صحت عبارات ریاضی از لحاظ علائم) یا } یا]:

تشخیص صحت عبارات ریاضی با توجه به علائم) یا } یا] با توجه به شرایط زیر حاصل می‌شود:

۱ـ تعداد ")" با "(" و تعداد "]" با "[" و تعداد "}" با "{" برابر باشد.

۲ـ تطبیق علائم ")" و "]" و "}": منظور این است که برای بستن ")" از "[" یا "{" استفاده نشده باشد و حتماً از "(" استفاده کرده باشیم.

مثال:

$\{((a + b) + [c / d]) - e\}$ عبارت صحیح

$\{((a + b) + [c / d)] - e\}$ عبارت ناصحیح

در عبارت فوق { انتهای عبارت اضافی بوده و همچنین برای بستن] از (و بستن) از [استفاده شده که نشان‌دهنده عدم تطبیق علائم است.

نحوه کار: عبارت را از سمت چپ به راست پویش کرده و علائم } و) و] را داخل stack قرار می‌دهیم تا به علائم [یا (یا { برسیم در این حالت اگر بالای stack علائم] یا) یا { منطبق با علائم [یا (یا { وجود داشته باشد علامت را از stack خارج می‌کنیم در غیر این صورت عملیات را متوقف کرده و عبارت صحیح نیست در عین حال اگر به انتهای عبارت رسیدیم و stack خالی باشد عبارت صحیح است در غیر این صورت علائم اضافی در stack وجود داشته و عبارت صحیح نیست

مثال ۱: حداقل فضای پشته برای تشخیص صحت عبارت روبرو کدام است؟

$$\{((a - b) + c - [d * e])\}$$

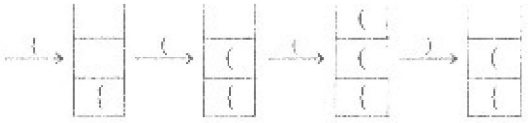
حداقل فضای مورد نیاز 3 بوده که همان حداکثر فضای استفاده شده است.

در انتهای عبارت stack خالی شده است بنابراین عبارت صحیح است.

تعداد pop با تعداد push برابر بوده و برابر تعداد علائمی از } و] و) که وارد stack شده‌اند می‌باشد.

مثال ۲: حداقل فضای پشته برای تشخیص صحت عبارت روبرو کدام است؟

$$\{((a + b) + c]\}$$



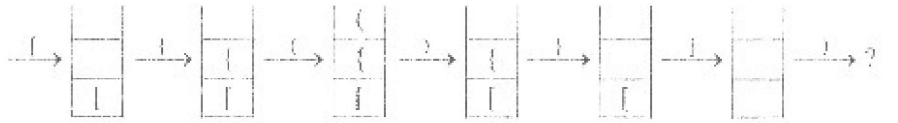
حداقل فضای مورد نیاز 3 بوده که همان حداکثر فضای استفاده شده است.

زمانیکه به "]" چون بالای stack ")" است بنابراین با عدم تطبیق روبرو می‌شویم و عبارت ناصحیح است.

تعداد push برابر 3 برای علائم } و) و) و تعداد pop ۱ و فقط برای (انجام می‌شود.و با رسیدن به [عملیات متوقف می‌شود.

مثال ۳: حداقل فضای پشته برای تشخیص صحت عبارت زیر کدام است؟

$$[\{(a + b) + c\}])$$



در انتهای عبارت برای ")" علامت "(" در stack وجود ندارد بنابراین عبارت غلط است.

حداقل فضای مورد نیاز 3 بوده که همان حداکثر فضای استفاده شده است.

تعداد push = 3 و تعداد pop = 3 اما در انتهای عبارت "("، علامتی در stack وجود ندارد و عبارت ناصحیح است.

تبدیل عبارات prefix و Postfix به عبارات infix

برای این عمل 2 روش وجود دارد:
۱ـ روش پرانتز گذاری
۲ـ استفاده از stack

۱ـ روش پرانتزگذاری

الف) ابتدا کنترل می‌کنیم که آیا تعداد عملوندها یک واحد از عملگرها بیشتر است؛ چون در غیر این صورت در عبارت عملگر یکانی وجود داشته و ارزشیابی آن شرایط خاصی خواهد داشت.

ب) عبارت را از سمتی بررسی می‌کنیم که با عملوند شروع می‌شود، در postfix از چپ و در prefix از راست عبارت را بررسی می‌کنیم.

ج) با رسیدن به هر عملگر دو عملوند برای آن در نظر می‌گیریم (در postfix سمت چپ عملگر و در prefix سمت راست عملگر)

د) برای دو عملوند، عملگر را بین آن‌ها قرار داده و برای آن‌ها پرانتز در نظر می‌گیریم.

هـ) عبارت را از سمت چپ به راست به ترتیب شامل عملوندها و عملگرهای بین عملوندها ارزیابی می‌کنیم.

مثال ۱: عبارت را از سمت عملوندها (چپ) بررسی می‌کنیم.

$$A\ ,\ B\ ,\ C\ ,\ *\ ,\ D\ ,\ /\ ,\ +$$

$$\longrightarrow \Big(A + , \big((B\ ^{*}\ , C\ ,)^{'}\ , D\ , / \big) , + \Big) \longrightarrow A + B * C / D$$

مثال ۲: عبارت را از سمت عملوندها (راست) بررسی می‌کنیم.

$$+\ ,\ -\ ,\ a\ ,\ /\ ,\ b\ ,\ c\ ,\ d$$

$$\longrightarrow \Big(+ , \big(- , a\ ^{-} , \big(/ , b\ ^{/} , c \big) \big)^{'} , d \Big) \longrightarrow a - b / c + d$$

مثال ۳: عبارت prefix زیر داده شده است: (کارشناسی ارشد ـ آزاد۷۹)

$$+ - * \uparrow ABCD / E / F + GH$$

کدامیک از عبارات زیر معادل infix برای این عبارت است؟

۱) $A^{B*(C-D)} + E(F/(G+H))$

۲) $A^{B} * (C-D) + \frac{EF}{G} + H$

۳) $A^{B} * C - D + E / F / G + H$

۴) $A^{B} * C - D + E / (F / (G - H))$

حل: از سمت عملوندها (راست) عبارت را بررسی می‌کنیم:

$$\Big(+ \Big(- \big(* \big(\hat{} A \uparrow B \big)^{*} C \big)^{-} D \Big)^{-} \big(/ E^{/} \big(F^{/} \big(+ G^{-} H \big) \big) \big) \Big)$$

$$= A \hat{} B * C - D + E / (F / (G + H))$$

گزینه ۲ صحیح می‌باشد.

۲ـ روش stack (پشته)

با فرض آن‌که عملگرها باینری (دودوئی) هستند (تعداد عملگرها از عملوندها یک واحد کمتر باشد) به صورت زیر عمل می‌کنیم:

الف) عبارت را از سمتی بررسی می‌کنیم که با عملوند شروع می‌شود، در postfix از چپ و در prefix از راست عبارت را بررسی می‌کنیم.

ب) عملوندها را در stack قرار می‌دهیم و با رسیدن به هر عملگر (به جز عملگر آخر) دو عملوند خارج کرده و بعد از محاسبه دوباره حاصل عبارت را داخل stack قرار می‌دهیم. (دقت کنید که دو عملوندی که از stack خارج می‌کنیم از سمت چپ به راست عمل می‌کنیم)

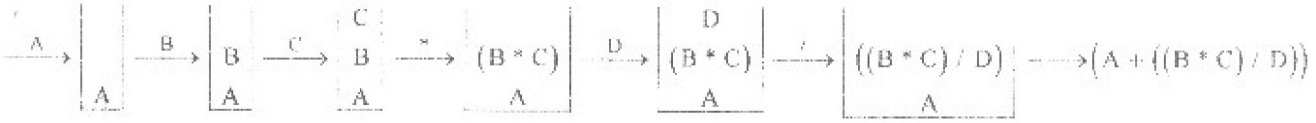
ج) به عملگر آخر که رسیدیم بعد از خارج کردن دو عملوند آخر و محاسبه آن را دیگر در stack قرار نمی‌دهیم و حاصل را در خروجی می‌نویسیم.

نکته: تعداد push و pop با هم برابر بوده و برابر با 2 برابر تعداد عملگرها می‌باشد.

مثال ۱:

A , B , C , * , D , / , +

از سمت چپ



نکته مهم:

۱ـ حداقل فضای مورد نیاز برای تبدیل بالا 3 می‌باشد که همان حداکثر فضای مورد استفاده است.

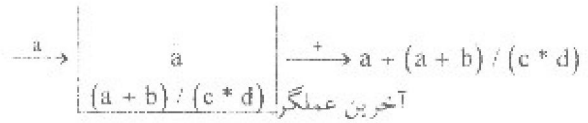
۲ـ تعداد push = تعداد pop = تعداد عملگرها × 2 = 2 × 3 = 6

مثال ۲: می‌نیمم تعداد متغیرهای میانی در محاسبه عبارت جبری a b + c d * / a − ؟

۱) 1 ۲) 2 ۳) 3 ۴) 4

حل: گزینه ۳ صحیح می‌باشد.

از سمت چپ بررسی می‌کنیم:



۱ـ حداقل فضای مورد نیاز برای تبدیل 3 می باشد که همان حداکثر فضای مورد استفاده است.

۲ـ تعداد push = تعداد pop = تعداد عملگرها × 2 = 2 × 4 = 8

تبدیل prefix به postfix و بالعکس:

۱ـ اگر در هنگام تبدیل prefix به infix در روش پرانتزگذاری به جای قرار دادن عملگر بین عملوندها، عملگر را روی پرانتز بسته بگذاریم به فرم postfix می‌رسیم.

۲ـ اگر در هنگام تبدیل postfix به infix در روش پرانتزگذاری به جای قرار دادن عملگر بین عملوندها، عملگر را روی پرانتز باز بگذاریم به فرم prefix می‌رسیم.

مثال ۱: معادل لهستانی عبارت پسوندی زیر کدام است؟

A B C * D / + :postfix (لهستانی وارونه)

حل: از سمت عملوندها عبارت را بررسی می‌کنیم (سمت چپ)

$$= + \left(A / \left(* (B C *) D / \right) \right)$$

$$= + A / * B C D$$

مثال ۲: معادل postfix عبارت پیشوندی زیر کدام است؟

+ a / b * - d e - a c

حل: از سمت عملوندها عبارت را بررسی می‌کنیم (سمت راست)

$$= \left(+ a \left(/ b \left(* (+ d e)^{+} (- a c)^{-} \right)^{*} \right)^{/} \right)^{+}$$

= a b d e - a c - * / +

مثال: معادل Postfix عبارت Prefix زیر کدام است؟ (مسابقات آموزشکده‌های فنی ۷۸)

++a/b-cd/-ab-+c*de/a-bc

۱) ab + cd - / ab - cd + / e * + ab / c - - + +

۲) a b - cd - ab - + / cde * + / abc - / -

۳) abcd - / + ab - cde * + abc - / - / +

۴) abcd / - - abc - de * abc - + / - / +

حل: از سمت عملوندها عبارت را بررسی می‌کنیم:

$$\left(+ \left(+ a \left(/ b (- c d)^{-} \right)^{/} \right)^{+} \left((- a b)^{-} \left(- \left(+ c (* d e)^{*} \right)^{+} \left(/ a (- b c)^{-} \right)^{/} \right) \right)^{-} \right)^{+}$$

صف (Queue):

هر ساختاری از داده‌های پشت سرهم ذخیره شده که در آن عمل درج از یک طرف (انتها) و عمل حذف از طرف دیگر (ابتدا) انجام می‌شود.

نکته ۱: به ساختار صف FIFO (First In First Out) نیز گفته می‌شود و در بعضی مراجع LIFO(Last In Last Out) نیز گفته می‌شود.

کاربرد صف

۱ـ در صف پردازش‌های سیستم عامل

۲ـ صف چاپگر

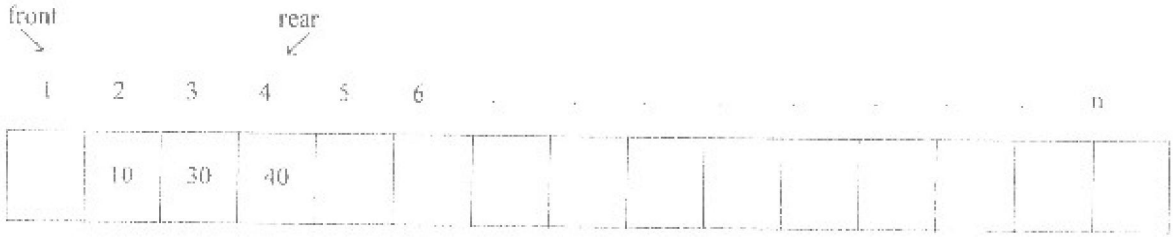
نکته ۲: برای نمایش یک صف از دو ساختار

۱ـ آرایه

۲ـ لیست پیوندی

استفاده می‌شود که نمایش پیوندی در بخش لیست پیوندی (link list) شرح داده می‌شود.

نکته ۳: در صورتیکه از یک آرایه n تائی Q[1 .. n] برای نمایش یک صف استفاده شود همیشه از دو اشاره‌گر Front و Rear که به ترتیب به ابتدا و انتهای صف اشاره می‌کند استفاده می‌کنیم.

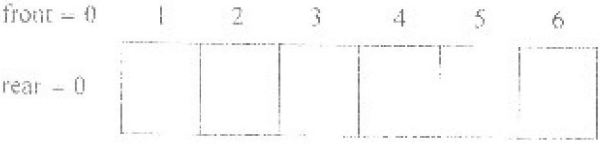


باتوجه به شکل دیده می‌شود که
- Front: همیشه به خانه قبل از خانه اول صف اشاره می‌کند.
- rear: همیشه به خانه آخر صف اشاره می‌کند.

صف خطی

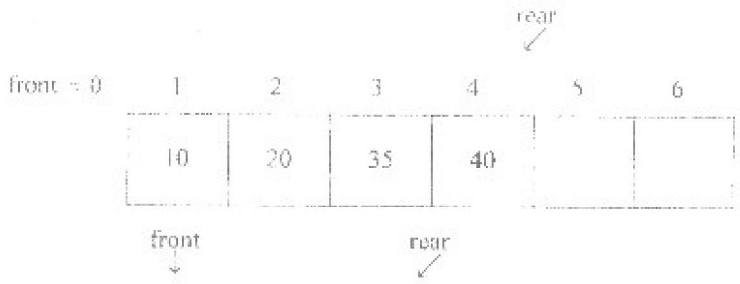
هر صف خطی یک آرایه n تائی Q[1 .. n] که عناصر از ابتدای صف (Front) حذف شده و به انتهای صف (rear) اضافه می‌شوند. و حرکت عناصر برای درج به سمت جلو است در عین حال عمل حذف نیز خانه‌های خالی در ابتدای آرایه ایجاد می‌کند که با توجه به حرکت عناصر برای درج به سمت جلو امکان استفاده از این خانه‌ها وجود ندارد.

مثال:



۱) صف 6 عضوی روبرو را در نظر بگیرید.

۲) درج عناصر 10 ، 20 ، 35 ، 40



۳) حذف 2 داده (به طور عادی داده‌ها از ابتدا حذف می‌شوند).

در وضعیت بالا با این که 2 داده از ابتدای صف حذف شده‌اند اما فقط دو خانه 5 ، 6 برای درج وجود دارند چون حرکت rear به سمت جلو بوده و امکان استفاده از خانه‌های قبلی را ندارد.

شرایط مرزی و اولیه: (مهم)

در صورتیکه
- Front: همیشه به خانه قبل از خانه اول صف اشاره کند.
- Rear: همیشه به خانه آخر صف اشاره کند.

انتخاب آرایه Q [1 .. n] برای پیاده‌سازی صف خطی دارای شرایط اولیه و مرزی زیر است:

وضعیت اولیه صف	خالی بودن صف خطی	پر بودن صف خطی
front = rear = 0	front = rear	rear = n

الگوریتم درج در صف

```
procedure  insert (item : نوع );
    begin
    if   rear = n   then
        write ('Queue Full')
    else
        begin
            rear := rear + 1;
            Q [rear] := item;
        end;
    end;
```

دیده می‌شود که چون rear به خانه آخر صف اشاره می‌کند برای عمل درج ابتدا rear یک واحد به جلو حرکت کرده سپس داده item در خانه خالی انتهای صف درج می‌شود.

الگوریتم حذف از پشته

```
procedure  delete (item : نوع );
    begin
    if   front = rear   then
        write ('Queue Empty')
    else
        begin
            front : = front + 1;
            item : = Q [front];
        end;
    end;
```

دیده می‌شود که چون front همیشه به قبل از اول صف اشاره می‌کند برای عمل حذف ابتدا front واحد به جلو حرکت کرده سپس داده ابتدای صف را خارج کرده در item قرار می‌دهد.

مشکل صف‌های خطی

مشکل هر صف خطی حرکت تدریجی عناصر به سمت انتهای صف و عدم امکان استفاده از خانه‌های ابتدای صف که در اثر حذف خالی شده‌اند.

راه حل رفع مشکل صف‌های خطی:

۱ـ شیفت خانه‌های انتهای صف به سمت ابتدای صف که این عمل در صورتیکه تعداد خانه‌های صف زیاد باشد هزینه بالائی خواهد داشت.

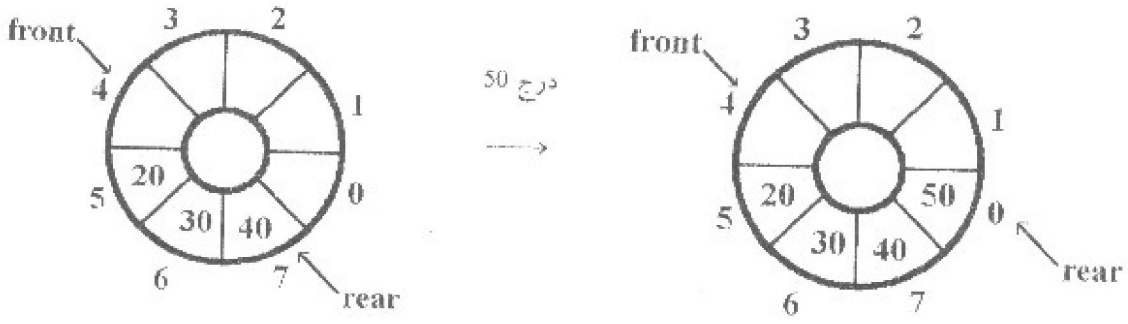
۲ـ استفاده از صف حلقوی به این مفهوم که زمانی که به انتهای صف رسیدیم عمل درج را دوباره از ابتدای صف انجام می‌دهیم و این حرکت چرخشی می‌باشد.

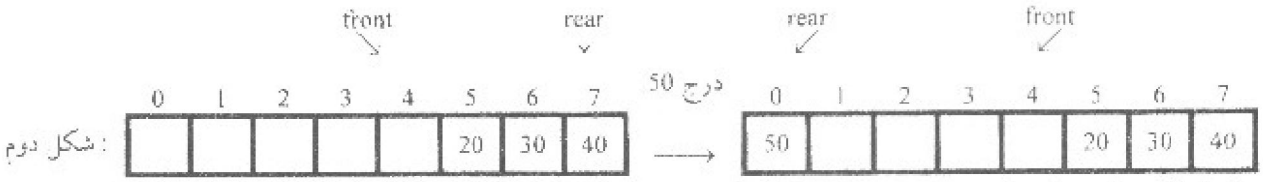
روش دوم در عمل مقرون به صرفه‌تر بوده و استفاده می‌شود.

صف حلقوی

آرایه متناظر Q [0 .. n - 1] را می‌توان به صورت یک صف حلقوی در نظر گرفت به طوریکه در این صف زمانیکه rear برابر n - 1 می‌شود، عنصر بعدی در خانه 0 قرار می‌گیرد.

شکل اول :





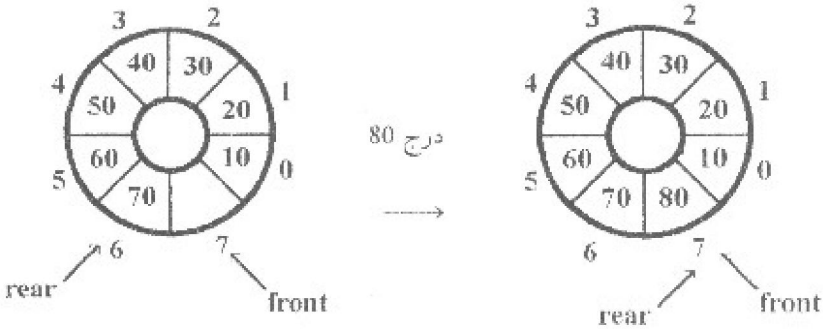
شرط خالی بودن صف حلقوی (front = rear)

در شرایطی که
- front: به قبل از اول صف اشاره کند، در صف حلقوی شرط خالی بودن صف (front = rear) خواهد بود.
- rear: به خانه آخر صف اشاره کند.

در هنگام شروع کار با صف هنگامی که صف حلقوی خالی باشد: front = rear = 0 است که یکی از حالت‌های خالی بودن صف در حین کار با صف می‌باشد.

نکته مهم: در هر صف حلقوی n عضوی همیشه یک خانه خالی وجود دارد، و حداکثر از n - 1 خانه صف می‌توانیم استفاده کنیم، این به آن علت است که بتوانیم وضعیت پر یا خالی بودن صف حلقوی را تشخیص بدهیم.

در صورتیکه در یک صف حلقوی n عضوی از همه خانه‌های صف بخواهیم استفاده کنیم و یک خانه را خالی نگذاریم وضعیت پر یا خالی بودن صف حلقوی قابل تشخیص نخواهد بود.



در مثال بالا بعد از درج 80 در صف حلقوی و استفاده از همان یک خانه‌ای که باید در صف خالی می‌ماند front = rear = 7 می‌شود که همان شرط خالی بودن صف است در صورتیکه صف پر شده و این یک تناقض است برای نبودن تناقض فوق و امکان تشخیص وضعیت پر یا خالی بودن صف حلقوی یک خانه را خالی نگه می‌داریم.

شرط پر بودن صف حلقوی $((rear - 1) \mod n = front)$

در صورتیکه صف حلقوی n عضو توسط آرایه $Q[0 .. n-1]$ پیاده‌سازی شود. شرط پر شدن صف حلقوی با در نظر گرفتن این نکته که حداکثر از $n-1$ خانه می‌توانیم استفاده کنیم و همیشه یک خانه صف باید خالی بماند از رابطه زیر بدست می‌آید:

شرط پر بودن صف حلقوی

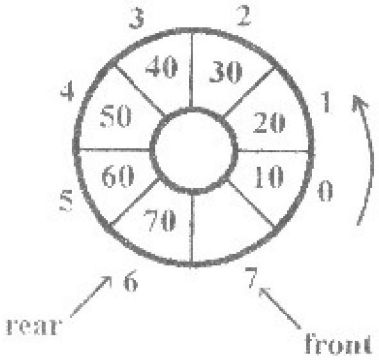
1) $(rear + 1) \mod n = front$

یا

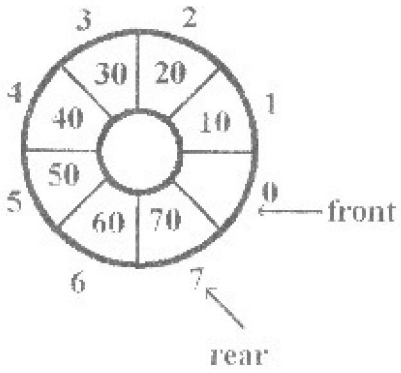
2) $((rear + 1) = front)$ or $(rear = n-1 , front = 0)$

مثال: در صف حلقوی زیر با $n = 8$ خانه:

(الف



(ب



الف) در این وضعیت $\begin{cases} rear = 6 \\ front = 7 \end{cases}$ و چون $(rear + 1) \mod n = front$ می‌شود صف پر است.

$$(6 + 1) \mod 8 = 7$$
rear — n — front

ب) در این وضعیت $\begin{cases} rear = 7 \\ front = 0 \end{cases}$ و چون $(rear + 1) \mod n = front$ می‌شود صف پر است.

$$(7 + 1) \mod 8 = 0$$
rear — n — front

رابطه ۱ یعنی $(rear + 1) \mod n = front$ رابطه‌ای است که هر دو وضعیت رابطه 2 را پوشش می دهد. برای همین کامترین رابطه می باشد.

علت استفاده از mod در رابطه بالا در شرایطی دیده می‌شود که در وضعیتی مانند حالت شکل (ب) قرار داریم که با حرکت rear به اندازه ۱ واحد به جلو باید از انتها به ابتدا برگردیم.

نتیجه‌گیری:

شرط پر و خالی بودن صف حلقوی n عضوی:

شرط خالی بودن: $front = rear$

شرط پر بودن: $(rear + 1) \mod n = front$

مثال: کدام مورد در ساختار یک صف حلقوی 1000 عضوی بیان کننده خالی و پر بودن صف است؟ (کنکور ۸۳)

۱) front = 0 , rear = 0 (خالی) و front = (rear + 1) mod n (پر)

۲) front = 0 , rear = 0 (خالی) و front = 0 , rear = 1000 (پر)

۳) front = 0 , rear = 1 (خالی) و rear = (rear + 1) mod 1000 (پر)

۴) front = 1000 ،rear = 1001 (خالی) و rear = 999 , front = 0 (پر)

گزینه ۱ صحیح می‌باشد.

front = rear = 0 خالی بودن (صحیح)

front = (rear + 1) mod n پر بودن صف (صحیح)

گزینه ۲ غلط می‌باشد.

front = rear = 0 خالی بودن صف (صحیح)

front = 0 , rear = 1000 پر بودن صف (غلط)

$$\underbrace{(rear + 1) \mod n}_{1} = \underbrace{(1000 + 1) \mod 1000}_{1} \neq \underbrace{front}_{0}$$

گزینه ۳ غلط می‌باشد.

front = 0 , rear = 1 خالی بودن صف (غلط)

rear = (rear + 1) mod 1000 پر بودن صف (غلط)

گزینه ۴ غلط می‌باشد.

front = 1000 , rear = 1001 خالی بودن صف (غلط)

rear = 999 , front = 0 پر بودن صف (صحیح)

$$\underbrace{(rear + 1) \mod n}_{(999 + 1) \mod 1000 = 0} = front$$

عملیات حذف از صف حلقوی

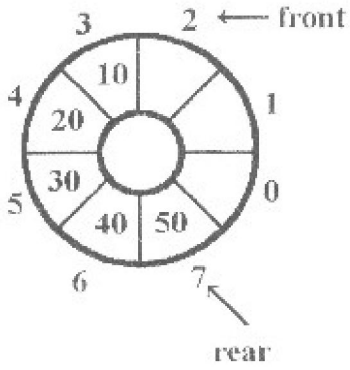
```
procedure  delqueue (item):
    begin
if  front = rear  then
    write ('Empty')
else
        begin
            front := (front + 1)  mod  n;
            item := Q [front];
        end;
    end;
```

عملیات درج در صف حلقوی

```
procedure  Addqueue (item)
    begin
if  (rear + 1)  mod  n = front  then
    write ('Full')
else
        begin
            rear : = (rear + 1)  mod  n;
            Q [rear] : = item;
        end;
    end;
```

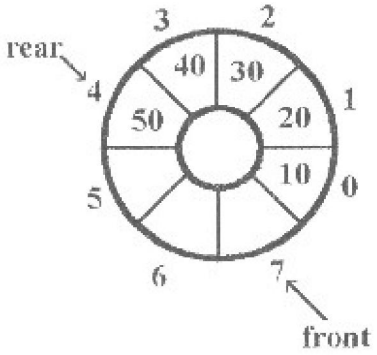
در هر دو روش حذف و درج به ترتیب از جملات front : = (front + 1) mod n و rear : = (rear + 1) mod n استفاده شده است که علت استفاده از mod در شرایطی است که front یا rear به خانه آخر اشاره کرده باشد و حرکت به جلو آن‌ها را به ابتدای صف منتقل می‌کند.

این حالت‌ها در شکل زیر دیده می‌شود.



$\underbrace{rear}_{0} := \underbrace{(rear + 1)}_{8} \quad mod \quad \underbrace{n}_{8}$;

Rear برای درج، یک واحد به جلو حرکت کرده به 0 اشاره می‌کند سپس مقدار جدید در فضای خالی درج می‌شود.



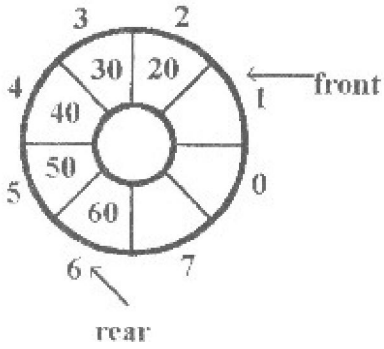
$\underbrace{front}_{0} := \underbrace{(front + 1)}_{8} \quad mod \quad \underbrace{n}_{8}$;

front برای حذف یک واحد به جلو حرکت کرده به 0 اشاره می‌کند و عنصر 10 حذف می‌شود.

تعداد خانه‌های پر و خالی یک صف حلقوی n تائی:

الف) front به قبل از اول صف اشاره می‌کند.
rear به انتهای صف اشاره می‌کند.

rear ≥ front ⟶
- تعداد خانه‌های پر : rear − front
- تعداد خانه‌های خالی : n − (rear − front)



n = 8
rear = 6
front = 1

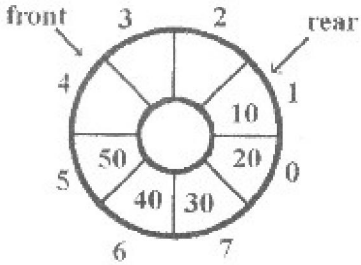
تعداد خانه‌های پر: rear − front = 6 − 1 = 5

تعداد خانه‌های خالی: n − (rear − front) = 8 − 5 = 3

rear < front ⟶

تعداد خانه‌های پر: n − (front − rear)

تعداد خانه‌های خالی: front − rear



n = 8
rear = 1
front = 4

تعداد خانه‌های پر: n − (front − rear) = 8 − 3 = 5

تعداد خانه‌های خالی: front − rear = 4 − 1 = 3

ب)

front: به ابتدای صف اشاره می‌کند

rear: به انتهای صف اشاره می‌کند.

rear ≥ front ⟶

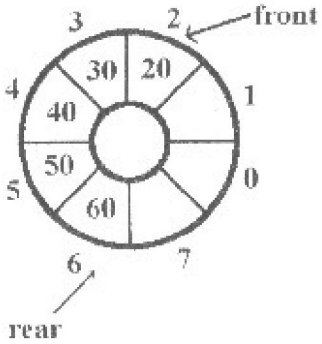
تعداد خانه‌های پر: rear − front + 1

تعداد خانه‌های خالی: n − (rear − front + 1)

n = 8
front = 2
rear = 6

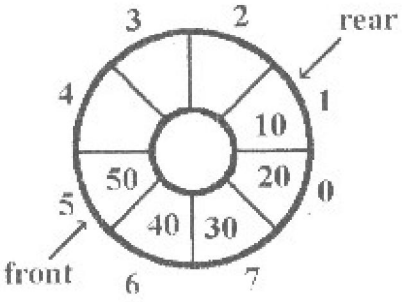
تعداد خانه‌های پر: 6 − 2 + 1 = 5

تعداد خانه‌های خالی: 8 − 5 = 3



rear < front ⟶
- تعداد خانه‌های پر: n − (front − rear − 1)
- تعداد خانه‌های خالی: front − rear − 1

n = 8
front = 5
rear = 1
⟶
- تعداد خانه‌های پر: 8 − 3 = 5
- تعداد خانه‌های خالی: 5 − 1 − 1 = 3



لیست‌های پیوندی (Linked list)

به طور کلی برای ذخیره‌سازی انواع داده‌ها در حافظه اصلی (RAM) از دو نوع ساختار می‌توان استفاده کرد:

۱ـ آرایه‌ها

۲ـ لیست‌های پیوندی

■ آرایه

هر آرایه لیست پشت سر هم از داده‌ها است که همگی داده‌ها از یک نوع بوده و ناحیه‌ای پیوسته از حافظه را در اختیار دارند.

نکات مهم در آرایه‌ها:

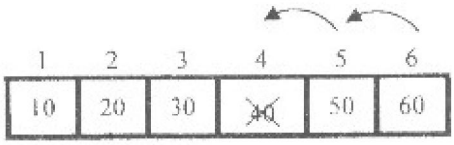
۱ـ تعداد عناصر هر آرایه همیشه محدود و مشخص و ایست است و در تمام طول برنامه ثابت می‌باشد. (عیب)

۲ـ عملیات حذف و درج یک داده دلخواه در ناحیه‌ای مشخص نیاز به شیفت دارد. (عیب)

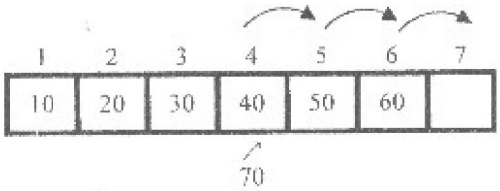
مثال:

الف) درج یک داده در محل kام یک آرایه N تائی نیاز به $N - K + 1$ شیفت دارد.

ب) حذف یک داده از محل kام یک آرایه N تائی نیاز به $N - K$ شیفت دارد.



به طور مثال برای حذف 40 از محل 4ام نیاز به $6 - 4 = 2$ شیفت است.



به طور مثال برای درج 70 در محل 4ام نیاز به $6 - 4 + 1 = 3$ شیفت داریم.

یادآوری:

عملیات همراه با شیفت عناصر همیشه هزینه بالائی دارد.

زمان حاصل از شیفت برای درج و حذف $O(n)$ است، چون به طور متوسط برای درج $\frac{n+1}{2}$ و برای حذف $\frac{n-1}{2}$ عمل شیفت نیاز داریم.

۳ـ عملیات جستجو به دنبال یک دادهٔ دلخواه در آرایه‌ها با یکی از دو روش زیر انجام می‌گیرد: (مزیت)

الف) جستجوی خطی (ترتیبی) در آرایه‌های مرتب یا نامرتب با زمان $O(n)$

ب) جستجوی دودوئی (باینری) در آرایه‌های مرتب با زمان $O(\log_2 n)$

مزیت این خصوصیت، تنوع روش‌های جستجو در آرایه‌ها است.

۴ـ دسترسی به داده‌های هر آرایه به صورت تصادفی و دلخواه به راحتی توسط اندیس آرایه انجام شده و دسترسی مستقیم و با زمان O(1) خواهد بود. (مزیت)

۵ـ روش‌های مختلفی برای مرتب‌سازی آرایه‌ها از قبیل: مرتب‌سازی حبابی ـ مرتب‌سازی انتخابی ـ مرتب‌سازی سریع ـ مرتب‌سازی درجی ـ ... وجود دارد. (مزیت)

■ لیست‌های پیوندی

لیست پیوندی: دارای عناصر (رکوردها) مختلفی از حافظه پویا (Heap) بوده که لزوماً عناصر آن کنار هم قرار نگرفته‌اند.

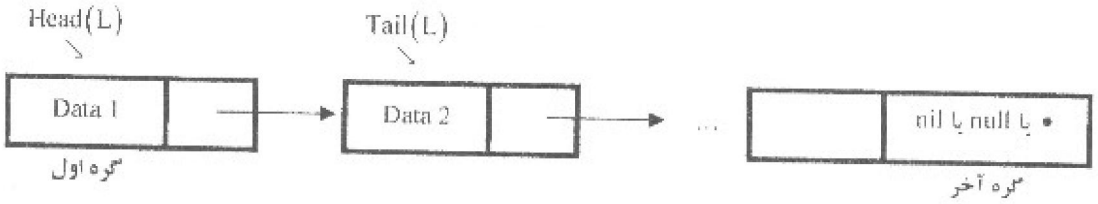
نکته:

هر عنصر از یک لیست پیوندی که معمولاً یک رکورد است به نام گره یا Node شناخته می‌شود و حداقل دارای 2 فیلد (قسمت) است

۱ـ داده (Data)

۲ـ اشاره‌گر (link)

که اشاره‌گر آدرس عنصر بعدی از لیست پیوندی را نگهداری می‌کند.



مهم:

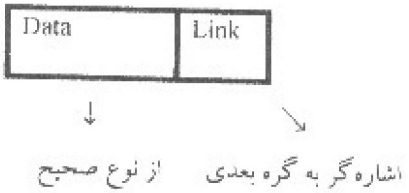
اشاره‌گر (link) گره آخر معمولاً با nil یا null یا • مقداردهی می‌شود.

Head (L): تابعی که اشاره‌گر به گره اول لیست را برمی‌گرداند.

Tail (L): تابعی که اشاره‌گر به گره بعد از گره اول لیست را برمی‌گرداند.

شکل کلی هر گره در ساختار لیست پیوندی در زبان‌های C و pascal:

در صورتیکه هر گره (node) به صورت:



باشد برای پیاده‌سازی آن در زبان‌های برنامه‌سازی C و pascal به شکل زیر عمل می‌کنیم:

pascal

```
Type
    list = ^ node;
    node = record
    Data : integer;
    link : list;
    end;
    :
var
    p : node;
    :
    New(p);
```

C

```
typedef  struct  node  *list;
typedef  struct  node {
int  Data;
list  link;
                    };
    :
struct  node  *p;
    :
p = malloc (sizeof (struct  node))
```

پیاده‌سازی یک گره به زبان الگوریتم به صورت زیر است:

1) if Avail = Null then write 'overflow' and exit

اگر فضای خالی در heap وجود نداشته باشد Avail = null شده و پیغام overflow ظاهر می‌شود. در غیر این صورت به خط بعدی می‌رویم و تخصیص انجام می‌شود.

2) p = Avail , Avail = link(Avail)

در صورتیکه فضای خالی در Heap وجود داشته باشد، Avail به عنوان یک گره آزاد در اختیار p قرار می‌گیرد، در این حالت گره مورد نظر می‌تواند توسط p مورد دسترسی واقع شود.

■ نحوه دسترسی به فیلدهای هر گره دلخواه

در صورتیکه p را اشاره‌گری به رکورد (گره یا node) دلخواه بدانیم در این صورت نحوه دسترسی به فیلدهای آن گره (node) فقط از طریق p به صورت زیر خواهد بود:

زبان پاسکال	زبان C	زبان الگوریتمی
P^.Data	P → Data	Data(p)
P^.link	P → link	link(p)
	(* p).Data	
	(* p).link	

در بعضی مراجع یا تست‌ها به جای Data از کلمه info استفاده می‌کنند.

مثال: برای آماده سازی گره p به صورت  می‌توانیم به صورت زیر عمل کنیم.

زبان الگوریتمی	زبان پاسکال	زبان C
Data (p) = 20;	P^.Data : = 20;	P → Data = 20
link (P) = 0;	P^.link : = nil;	P → link = null

null و nil و • یک مفهوم را دارند.

■ عملیات و انتساب‌ها در اشاره‌گرهای لیست‌های پیوندی (مهم)

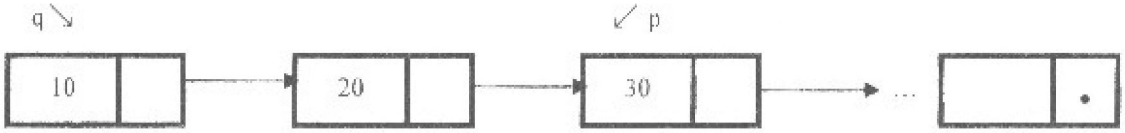
اگر p و q دو اشاره‌گر به گره‌های یک لیست پیوندی باشد وضعیت‌های زیر بروز می‌کند:

الف)

p = q اشاره کند به همان محلی که q اشاره می‌کند p

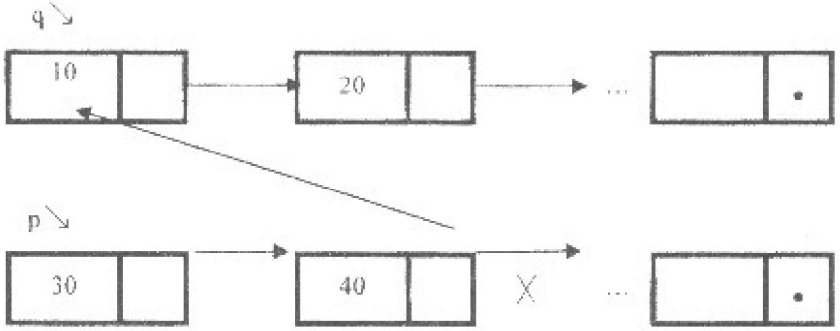
ب)

p : = q^.link^.link;
20
30



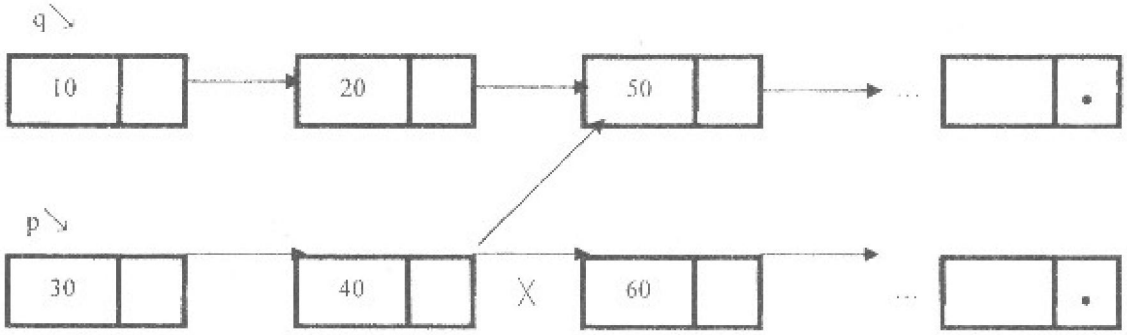
ج)

p^.link^.link : = q;
40



د)

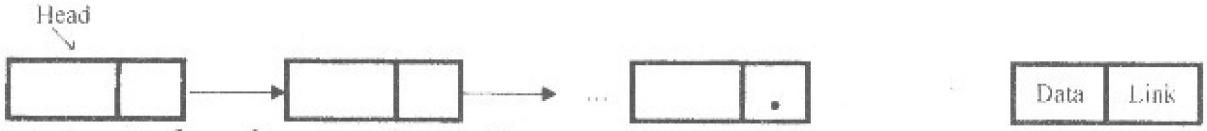
```
p^.link^.link := q^.link^.link;
   40          20      50
```



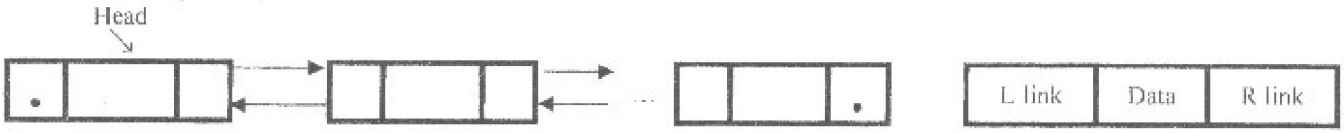
دیده می‌شود که هنگام انتساب اشاره‌گرهای لیست پیوندی در صورتیکه در سمت راست حکم انتساب مجموعه‌ای از linkها وجود داشته باشد، همه linkها محاسبه می‌شود ولی در سمت چپ حکم انتساب آخرین link را محاسبه نمی‌کنیم.

نکته: لیست‌های پیوندی می‌توانند به دو شکل عمومی یک طرفه و دو طرفه تعریف و استفاده شوند.

۱ـ **لیست پیوندی یک طرفه (خطی):** در هر گره فقط یک فیلد اشاره‌گر وجود دارد که آن هم آدرس گره بعدی را دارد.



۲ـ **لیست پیوندی دو طرفه (خطی):** در هر گره دو فیلد اشاره‌گر وجود دارد که یکی به گره بعدی و دیگری به گره قبلی اشاره می‌کند.

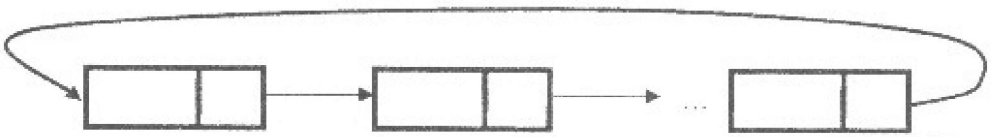


لیست‌های دوری (حلقوی)

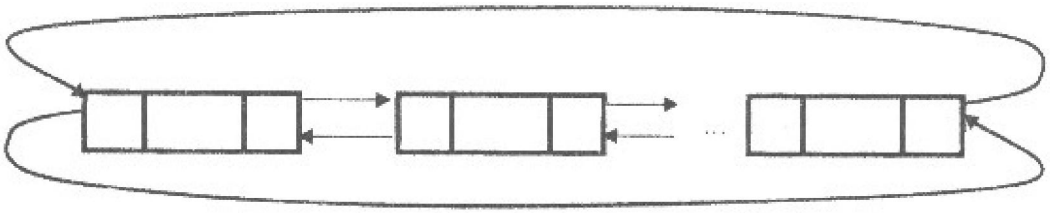
۱ـ در لیست حلقوی یک‌طرفه اشاره‌گر link گره آخر به گره اول اشاره می‌کند.

۲ـ در لیست حلقوی دو طرفه اشاره‌گر R link گره آخر به گره اول و اشاره‌گر L link گره اول به گره آخر اشاره می‌کند.

۱ـ **لیست حلقوی یک طرفه**

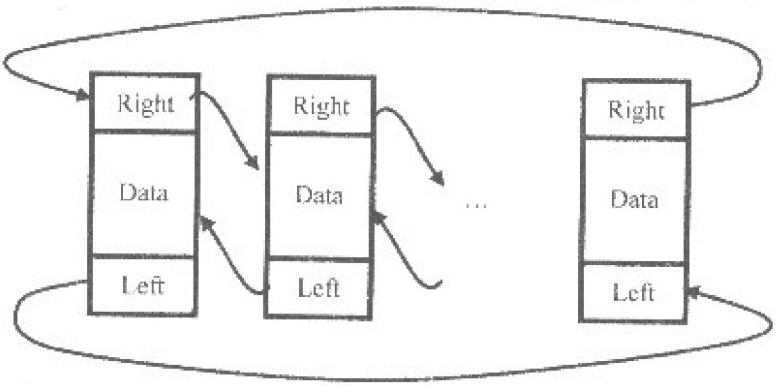


۲ـ **لیست حلقوی دوطرفه**

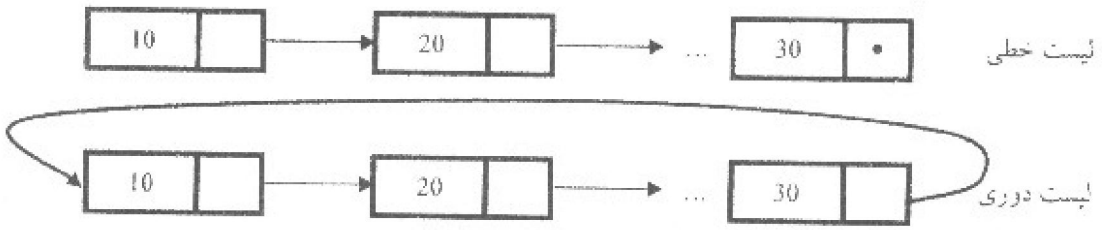


نکته مهم: بین لیست‌های دوری و غیردوری تفاوتی از نظر اتلاف حافظه وجود ندارد.

چون در لیست‌های دوری در حقیقت از اشاره‌گر (Link) گره‌هائی استفاده می‌شود. که در لیست‌های خطی nil یا null یا • هستند.



نکته مهم: یکی از مزایای مهم لیست‌های دوری یک طرفه نسبت به لیست‌های خطی یک‌طرفه این است که در لیست‌های دوری یک‌طرفه بدون استفاده از حافظه اضافی امکان رسیدن به گره قبلی از هر گره دلخواه وجود دارد که این عمل با یک دور کامل انجام می‌شود.



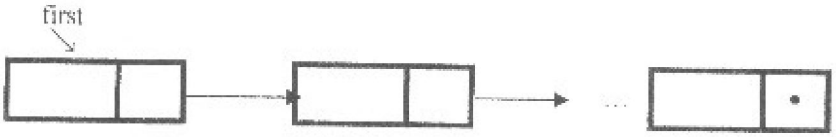
به طور مثال در لیست خطی بالا امکان رسیدن به گره 10 از موقعیت گره 20 وجود ندارد، در صورتیکه در لیست دوری امکان رسیدن از موقعیت گره 20 به گره 10 با رفتن به انتهای لیست و برگشت به ابتدای لیست وجود دارد.

یادآوری:

در لیست‌های دوطرفه رسیدن از یک گره به گره قبلی راحت‌تر از لیست ورودی انجام می‌شود. چون از هر گره یک اشاره‌گر L.link به گره قبل وجود دارد و مانند لیست حلقوی نیازی به رفتن به انتهای لیست و برگشت به ابتدای لیست نخواهد بود.

لیست‌های خطی یک‌طرفه

در این نوع لیست پیوندی، هر گره با یک اشاره‌گر link فقط آدرس گره بعدی را نگهداری می‌کند و امکان دسترسی به گره قبلی را ندارد؛ برای همین به آن لیست خطی یک‌طرفه می‌گویند.



یادآوری:

first را معادل head یا start می‌توانیم در نظر بگیریم.

در این ساختار اشاره‌گر first به گره اول لیست اشاره می‌کند و فقط از طریق این اشاره‌گر می‌توانیم به گره‌های لیست دسترسی پیدا کنیم. در این وضعیت اگر first را از دست بدهیم یا به هر نحوی آدرس آن را گم کنیم دیگر امکان دسترسی به عناصر لیست وجود ندارد.

■ **نکات مهم در لیست‌های پیوندی خطی یک‌طرفه:**

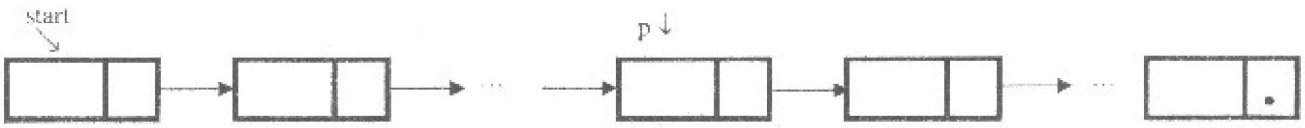
۱ـ امکان پیمایش لیست فقط از Head به Tail وجود دارد، آن‌هم به این علت که لیست یک‌طرفه است، در عین حال امکان پیمایش لیست از Tail به Head وجود ندارد.

۲ـ در صورتیکه شرط Head = null برقرار باشد، یا به عبارت بهتر متغیر Head مقدار null را داشته باشد لیست تهی خواهد بود، در این حالت نیازی به کنترل شرط Tail = null نداریم، چون لیست یک‌طرفه است.

```
if   Head = null   then   write('empty');
```

۳ـ از هر گره دلخواه p به راحتی می‌توان به گره بعدی دسترسی پیدا کرد، چون آدرس گره بعدی در اشاره‌گر link گره p ذخیره شده است، در عین حال امکان دسترسی به گره قبلی از گره p وجود نداشته و برای این کار باید از ابتدای لیست عمل پیمایش را تا رسیدن به گره قبل از p انجام دهیم.

مثال:



در این مثال:

الف) با استفاده از p ⟶ link به راحتی به گره بعد از p دسترسی خواهیم داشت.

ب) برای رسیدن به گره قبل از p، از موقعیت گره p نمی‌توان عمل کرد و باید از ابتدای لیست عمل پیمایش را تا رسیدن به موقعیت قبل از p انجام داد.

```
q = start;
while   q^.link < > p   do
            q = q^.link;
```

در این قطعه برنامه ابتدا q به گره اول اشاره کرده، سپس با شروع کار حلقه تا زمانی‌که به گره‌ای برسیم که Link آن گره p باشد، گره‌های لیست توسط q پیمایش می‌شوند، هنگام خروج از حلقه q به گره قبل از p اشاره می‌کند.

۴ـ برای حذف یک گره دلخواه نیاز به 1 عمل جایگزینی داریم با زمان O (1)، البته برای حذف هر گره باید از موقعیت قبل از آن این عمل را انجام دهیم.

۵ـ برای درج یک گره دلخواه نیاز به 2 عمل جایگزینی داریم با زمان O (1)، البته برای درج هر گره باید از موقعیت قبل از آن گره این عمل را انجام دهیم.

۶ـ برای حذف یک گره دلخواه در صورتیکه لیست خانه‌های حذف شده نیاز باشد، بعد از عمل حذف هر گره باید آن را به لیست مورد نظر اضافه کنیم که در این صورت نیاز به 3 عمل جایگزینی داریم، 1 عمل جایگزینی برای حذف و 2 عمل جایگزینی برای درج در لیست خانه‌های حذف شده نیاز است.

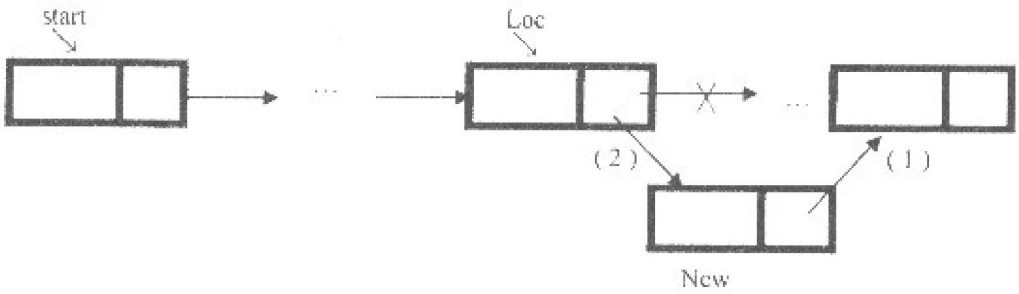
۷ـ به طور متوسط برای رسیدن به یک گره دلخواه در یک لیست پیوندی یک‌طرفه n عضوی، نیاز به $\frac{n}{2}$ پیمایش داریم.

■ درج گره در یک لیست پیوندی یک‌طرفه

برای درج یک گره دلخواه در موقعیت مشخص باید به قبل از موقعیت مورد نظر رسیده و با 2 عمل جایگزینی عمل درج را انجام داد.

می‌خواهیم گره New را بعد از گره معلوم Loc از لیستی که start به ابتدای آن اشاره می‌کند درج کنیم.

```
New : = getnode( );
New ^.data : = item;
if   first = nil   then
     begin                          } لیست تهی بوده و گره New به ابتدای لیست اضافه می‌شود.
        New ^.link : = start;
        start : = New;
     end
else
     begin
     1) New ^.link : = Loc ^.link;
     2) Loc ^.link : = New;
     end;
```



در صورتیکه دستورات (1) و (2) را جابجا کنیم قسمتی از لیست که start به آن اشاره می‌کند گم می‌شود. شکل الگوریتمی درج گره New می‌تواند به شکل روبرو باشد:

```
1) if   Avail = null   then   'overflow' and 'exit'
2) New = Avail , Avail = Link(Avail)
3) data(New) = item
4) if   (start = null)   then   link(New) = start , start = New
5) else
     Link(New) = Link(Loc)
     Link(Loc) = New
6) exit
```

نکته: قطعه برنامه زیر در هر وضعیتی که لیست تهی یا غیرتهی باشد گره New را به ابتدای لیست پیوندی اضافه می‌کند.

```
1) New ^.link = start
2) start = New
```

■ حذف گره از یک لیست پیوندی یک طرفه

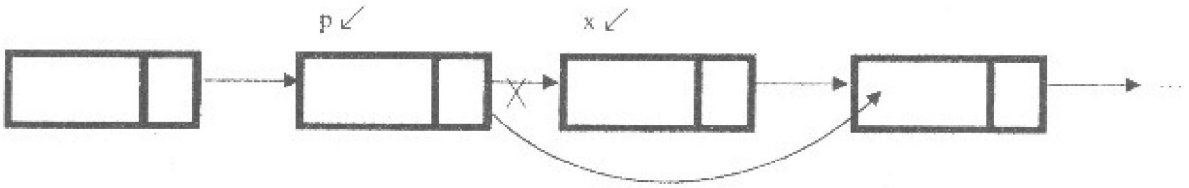
برای حذف گره با آدرس x باید لیست تا یافتن گره قبل از آن پیمایش شود، سپس با یک عمل جابجایی گره مورد نظر حذف می‌شود.

```
p = start;
while (p^.link < > x) do   → رسیدن به گره قبل از گره حذف شونده x
p = p^.link;
```

1) p^.link = x^.link;

یا

2) p^.link = p^link^.link;



بعد از رسیدن گره p به گره قبل از x توسط یکی از 2 عمل (1) و (2) گره مورد نظر حذف می‌شود.

نکته: با استفاده از دستور dispose (x) یا free (x) می‌توانیم بعد از حذف گره x، آن را به طور کامل تخریب کنیم.

پیمایش لیست‌های پیوندی خطی یک‌طرفه

پیمایش گره‌های یک لیست پیوندی خطی یک‌طرفه از ابتدای لیست انجام شده و به دو صورت بازگشتی و غیربازگشتی می‌تواند انجام گیرد.

الف) غیربازگشتی:

```
if  start < > nil  then
    begin
       p : = start;
       while  p < > nil  do
          begin
          write (p^.Data);
          p : = p^.link;
          end;
    end;
```

در شرط حلقه while اگر از شرط p^.link < > nil استفاده کنیم پیمایش لیست تا قبل از گره آخر انجام شده و با رسیدن اشاره‌گر p به گره آخر از حلقه خارج می‌شویم.

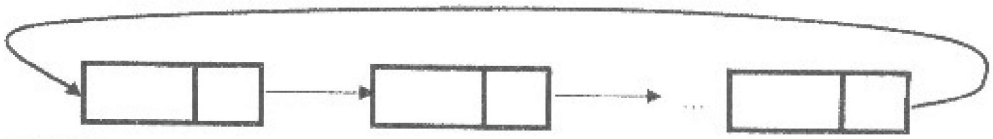
ب) بازگشتی:

```
procedure  show (L :List);
    begin
      if  (L < > nil)
         begin
         write(L^.Data);
         show(L^.link);
       end;
    end;
```

در شرط دستور if اگر از شرط L^.link < > nil استفاده کنیم پیمایش لیست تا قبل از گره آخر انجام شده و با رسیدن اشاره‌گر L به گره آخر از تابع بازگشتی خارج می‌شویم.

لیست حلقوی (لیست پیوندی یک‌طرفه دوری)

یک لیست پیوندی خطی یک‌طرفه که اشاره‌گر Link گره آخر به جای آن‌که مقدار null را اختیار کند به گره اول لیست اشاره می‌کند.



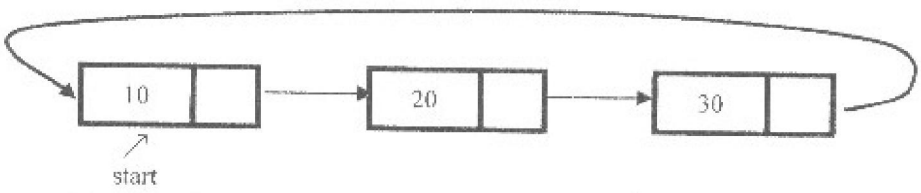
نکته ۱: مزیت لیست حلقوی (دوری) بر لیست خطی یک‌طرفه این است که در لیست حلقوی بدون نیاز به هیچ حافظه اضافی با داشتن آدرس یک گره دلخواه امکان دسترسی به گره قبلی یا به عبارت بهتر کلیه گره‌ها با حداکثر n - 1 پیمایش در لیست n عضوی وجود دارد ولی در لیست‌های یک‌طرفه این امکان وجود نداشته و دسترسی به گره‌های قبل از یک گره فقط از طریق آدرس شروع لیست و پیمایش لیست میسر خواهد بود.

نکته ۲: پیمایش یک لیست حلقوی در صورتیکه start اشاره‌گری به ابتدای لیست حلقوی باشد به شرح زیر خواهد بود:

```
if   start < > nil   then
     begin
     p : = start;
     repeat
     write (p ^.Data);
     p : = p^.link;
     until   p = start;
     end;
```

خروجی

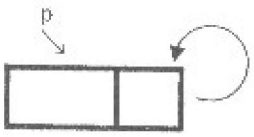
10
20
30



دیده می‌شود که شرط پایان حلقه آن است که اشاره‌گر p بعد از پیمایش کل لیست دوباره به ابتدای لیست اشاره کند.

نکته: در صورتیکه در شرط روبروی until عبارت p^.link = start ذکر می‌شد، آن‌گاه عمل پیمایش با رسیدن به گره آخر متوقف می‌شد و خروجی به صورت: 10 ، 20 نمایش داده می‌شد.

نکته ۳: در صورتیکه p اشاره‌گری به یک لیست حلقوی باشد جمله p^.link : = p یک لیست حلقوی یک عضوی را پیاده‌سازی می کند.



عملیات در لیست‌های حلقوی

عملیات در لیست‌های حلقوی مانند عملیات در لیست‌های خطی غیردوری است با این تفاوت که باید شرط پایان حلقه‌ها و نحوه اصلاح اشاره‌گر گره آخر با توجه به دوری بودن لیست تغییر یابد.

مثال: فرض کنید a اشاره‌گر انتهای یک لیست حلقوی و یک لیست غیرحلقوی باشد و بخواهیم گره x را بعد از آن درج کنیم.

لیست دوری

```
if (a < > nil) then
begin
   x^.link : = a^.link;
   a^.link : = x;
   end
else
   begin
   {a : = x;
   {x^.link : = x;
   end;
```

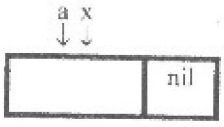
در صورتیکه لیست تهی باشد

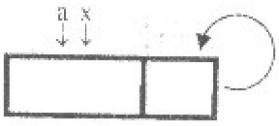
لیست غیر دوری

```
if (a < > nil) then
   begin
   x^.link : = a^.link;
   a^.link : = x;
   end
else
   begin
   {a : = x;
   {x^.link : = nil;
   end;
```

در صورتیکه لیست تهی باشد

دیده می‌شود برای درج درحالتی که لیست تهی نیست دو الگوریتم شبیه هم عمل می‌کنند اما زمانی‌که لیست تهی باشد در وضعیت متفاوت به صورت زیر بعد از درج گره به وجود می‌آید:

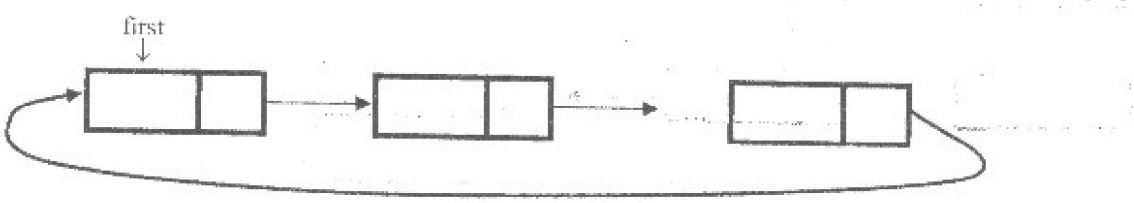




نکته مهم:

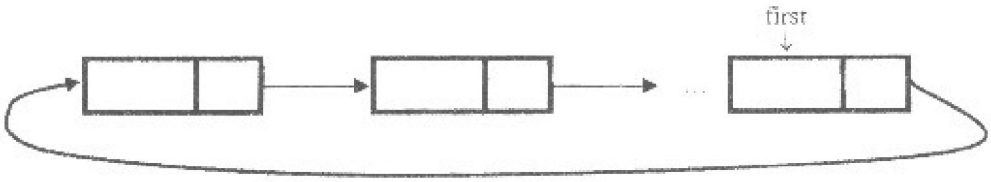
در صورتیکه اشاره‌گر first در لیست‌های دوری به گره اول یا آخر اشاره کند وضعیت‌های متفاوتی از نقطه نظر زمان انجام بعضی از عملیات‌ها بوجود می‌آید که به شرح زیر آن‌ها را بررسی می کنیم.

الف) اشاره‌گر first به ابتدای لیست اشاره کند:



در این وضعیت برای انجام عملیات: درج قبل از ابتدا ـ حذف از ابتدا ـ برای درج در انتها نیاز به عمل پیمایش داریم تا به گره آخر برسیم و بتوانیم عملیات فوق را از روی موقعیت گره آخر انجام دهیم.

ب) اشاره گر first به انتهای لیست اشاره کند:

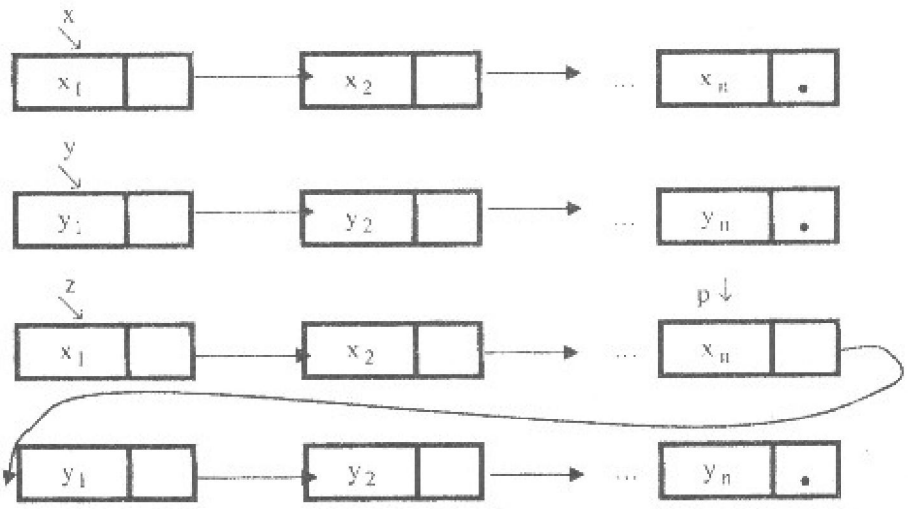


در این وضعیت برای انجام عملیات: درج قبل از ابتدا ـ حذف از ابتدا ـ برای درج در انتها نیاز به عمل پیمایش نداریم. و به راحتی می توانیم عملیات فوق را از روی گره آخر که first به آن اشاره می کند انجام دهیم.

اتصال دو لیست پیوندی

اگر دو لیست پیوندی x و y به شرح زیر وجود داشته باشند، با استفاده از قطعه برنامه زیر می توان این دو لیست را به هم متصل کرده و اشاره گر z را به ابتدای لیست حاصل از اتصال دو لیست x , y اشاره داد.

```
if   x = nil   then   z := y
else
     begin
     z := x;
if   y < > nil   then
     begin
     p := x;
     while   p^.link < > nil   do
     p := p^.link;
     p^.link := y;
     end;
end;
```



در شرط x = nil اگر x تهی باشد z به y اشاره خواهد کرد در غیر این صورت در قسمت else: z := x شده و z به ابتدای لیست x اشاره خواهد کرد.

در قسمت else بعد از آن‌که z := x انجام شد، با فرض آن‌که y تهی نباشد (در شرط y <> nil)؛ اشاره‌گر p در حلقه while، لیست x را پیمایش کرده تا روی گره آخر قرار گیرد، در این حالت از حلقه خارج شده و با جمله p^.link := y، اشاره‌گر (link) گره آخر x به گره اول y اشاره کرده و الگوریتم پایان می‌پذیرد.

لیست‌های پیوندی دوطرفه

در لیست‌های پیوندی دو طرفه در هر گره دو اشاره‌گر Left و Right وجود دارد که به ترتیب به گره قبلی و بعدی اشاره می کنند، با داشتن آدرس هر گره به راحتی می توان به گره قبلی یا بعدی دسترسی پیدا کرد.

مشخصات مهم لیست‌های پیوندی دو طرفه:

۱ـ امکان پیمایش لیست از Head به Tail و از Tail به Head به علت دوطرفه بودن وجود دارد.

۲ـ شرط تهی بودن لیست پیوندی دوطرفه Head = Tail = null است.

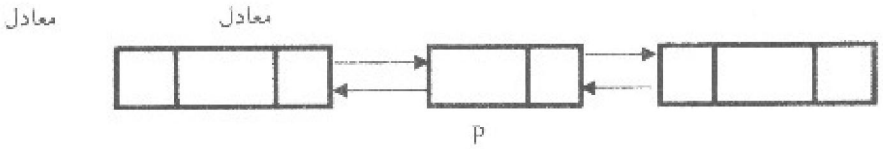
```
if  (head = null) And (Tail = null)  then  write('empty')
```

۳ـ از هر گره دلخواه p می‌توان به راحتی به گره بعدی p^.Right یا گره قبلی p^.Left دسترسی داشت.

۴ـ امکان درج و حذف هر گره در هر موقعیت دلخواه وجود دارد.

۵ـ در هر لیست پیوندی دو طرفه برای هر گره مانند p رابطه زیر برقرار است:

p^.Right^.left = p^.Left^.Right = p



۶ـ برای حذف یک گره دلخواه نیاز به 2 عمل جایگزینی داریم؛ با زمان O (1)

۷ـ برای درج یک گره دلخواه نیاز به 4 عمل جایگزینی داریم. با زمان O (1)

۸ـ برای حذف یک گره دلخواه در صورتیکه لیست خانه‌های حذف شده نیز نیاز باشد بعد از عمل حذف هر گره باید آن را به لیست مورد نظر اضافه کنیم که در این صورت نیاز به 6 عمل جایگزینی داریم.

۹ـ به طور متوسط برای رسیدن به یک گره دلخواه در یک لیست دوطرفه n عضوی نیاز به $\frac{n}{2}$ پیمایش است، که این عمل می‌تواند از ابتدای لیست انجام شود.

عمل درج در لیست‌های دوپیوندی

با داشتن آدرس یک گره دلخواه مانند p در یک لیست دوپیوندی می‌توان گره جدید New را در سمت راست یا چپ گره p درج کرد.

نکته ۱: (مهم)

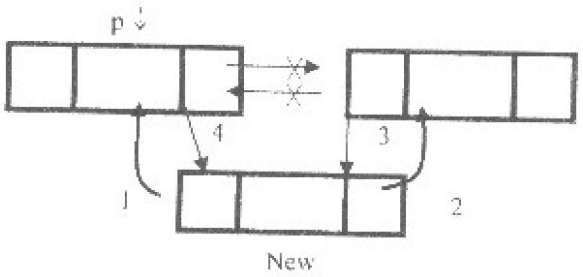
هر عمل درج در لیست دو پیوندی نیاز به 4 عمل جابگزینی دارد.

نکته ۲:

در صورتیکه عملیات لازم برای درج گره New در سمت راست یا چپ گره p نوشته شود کافیست در داخل عملیات به جای Left از Right و به جای Right از Left استفاده کنیم تا عملیات برای سمت دیگر بدست آید.

اضافه کردن گره New در سمت راست گره p:

1) New^.Left := p;
2) New^.Right := P^.Right;
3) p^.Right^.Left := New;
4) p^.Right := New;

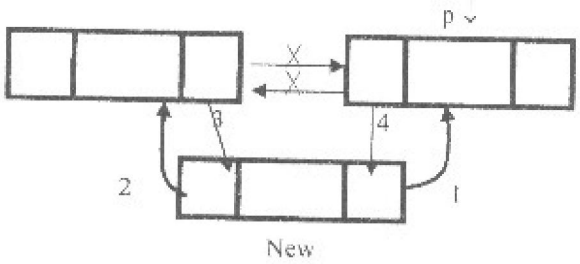


یادآوری:

در لیست‌های دو پیوندی اصطلاح‌های Next و Right و R link که اشاره‌گر به گره بعدی هستند با هم معادلند. همین‌طور Prev و Left و L link که اشاره‌گر به گره قبلی هستند نیز با هم معادلند.

اضافه کردن گره New در سمت چپ گره p:

1) New^.Right := p;
2) New^.Left := p^.Left;
3) p^.Left^.Right := New;
4) p^.Left := New;



نکته: همان‌طور که دیده می‌شود برای الگوریتم اضافه کردن گره New در سمت چپ گره p فقط جای Left و Right را در الگوریتم اضافه کردن گره New در سمت راست گره p عوض کردیم.

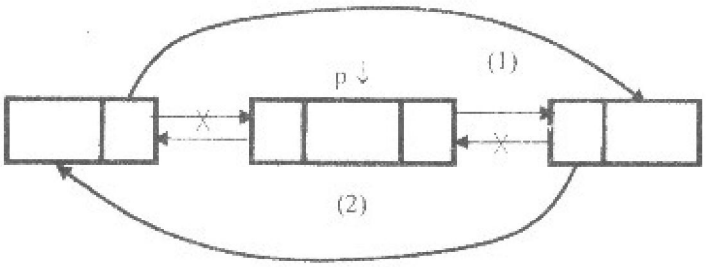
عمل حذف در لیست‌های دو پیوندی

با داشتن آدرس گره دلخواه p در یک لیست دوپیوندی می‌توان گره مورد نظر را حذف نمود.

نکته ۱: (مهم)

هر عمل حذف در لیست دو پیوندی نیاز به 2 عمل جایگزینی دارد.

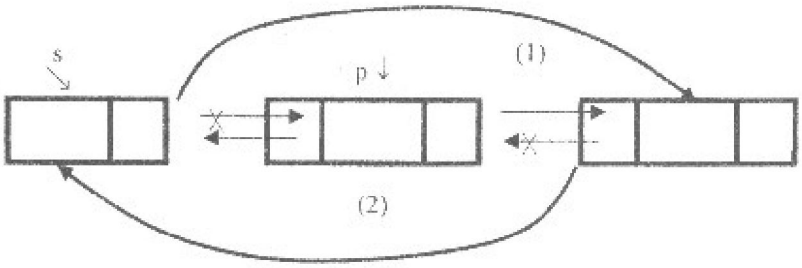
عملیات لازم برای حذف گره p در لیست دو پیوندی:




```
1) p^.Left^.Right := p^.Right;
2) p^.Right^.Left := p^.Left;
```

در عین حال عمل حذف گره می‌تواند از گره قبلی یا بعدی گره p انجام شود.

الف) در صورتیکه اشاره گر s قبل از گره حذف شوندهٔ p باشد.




```
1) s^.Right := p^.Right;
2) p^.Right^.Left := s;
```

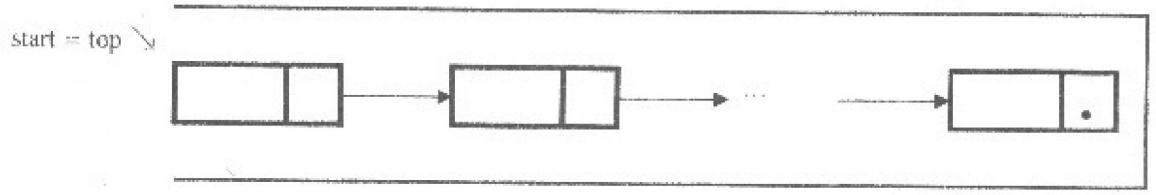
پیاده‌سازی پیوندی ساختارهای صف و پشته:

ساختارهای صف و پشته را می‌توان با استفاده از لیست‌های پیوندی پیاده‌سازی کرد.

نکته: استفاده از لیست‌های پیوندی در مقایسه با آرایه‌ها برای پیاده‌سازی صف و پشته این مزیت را دارد که با استفاده از حافظه پویا و تخصیص این حافظه به صورت متناسب با داده‌ها از فضای حافظه به طور بهینه استفاده می‌شود، این در حالی است که در آرایه‌ها فضا به صورت محدود و ایستا در اختیار ساختارهای صف و پشته قرار می‌گیرد.

■ **پشته پیوندی:**

هر لیستی که در آن عمل درج و حذف عناصر از یک‌طرف لیست (ابتدای لیست) انجام شود پشته پیوندی نامیده می‌شود.

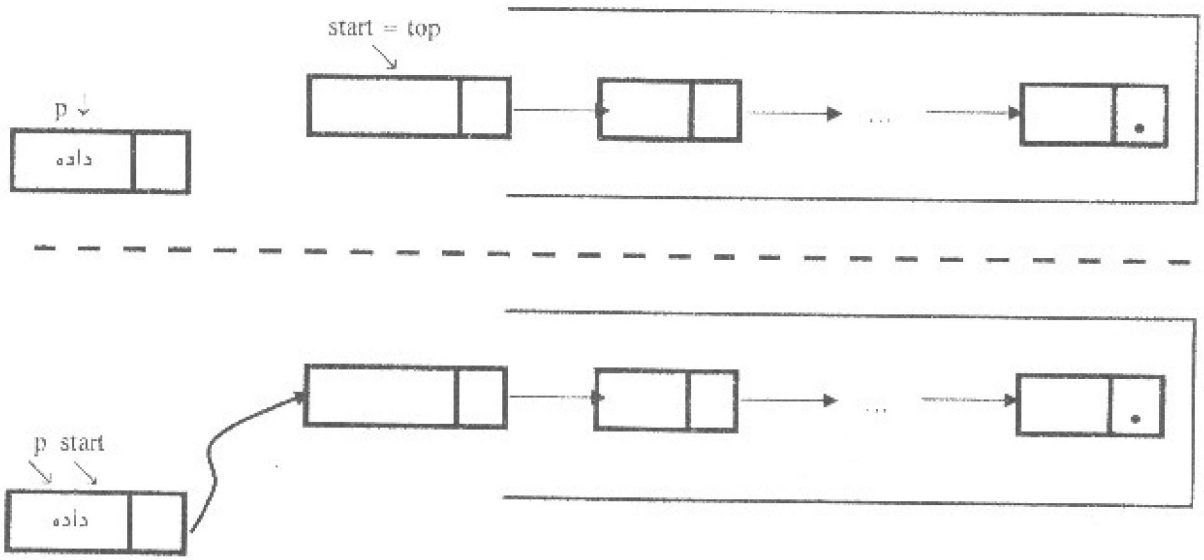


عملیات pop (حذف داده) و push (درج داده) به صورت زیر انجام می‌شود:

الف) push (درج داده)

این عمل در ابتدای لیست انجام می‌شود و به صورت مقابل:

```
1) New(p);
2) p^.data : = داده ;
3) p^.link : = start;
4) start : = p;
```



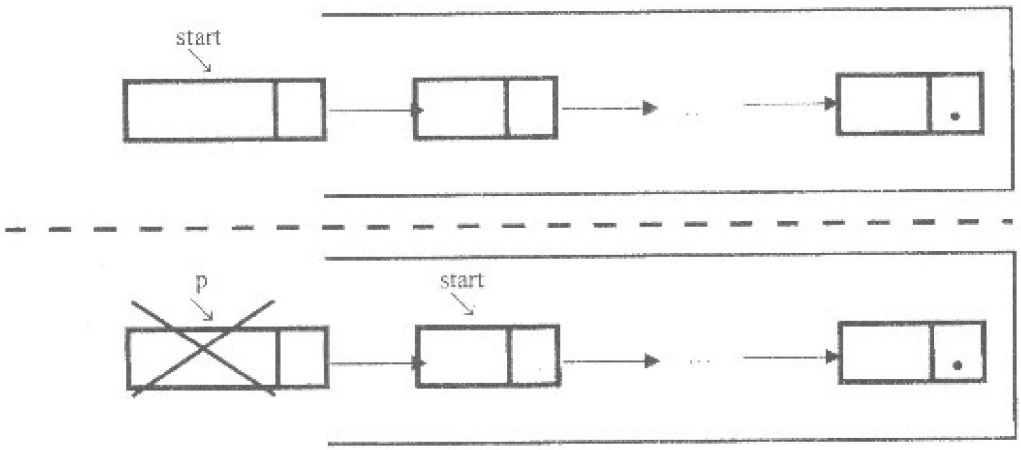
عمل درج در لیست پیوندی همان‌طور که قبلاً اشاره شد، 2 عمل جایگزینی نیاز دارد که در این‌جا عملیات 3 ، 4 می‌باشد.

عملیات 1 ، 2 برای آماده سازی گره p شامل ایجاد گره (New (p);) و مقداردهی آن (:داده = p^.data) می‌باشد.

ب) pop (حذف داده)

این عمل مانند Push از ابتدای لیست پیوندی انجام شده و به صورت زیر در نظر گرفته می‌شود.

```
1) p : = start;
2) start : = start^.link;
3) x : = p^.data;
4) free(p);
```



عمل حذف در لیست‌های پیوندی همان‌طور که قبلاً اشاره شد، فقط نیاز به یک عمل جابگزینی دارد. که در این‌جا دستور دوم (start : = start^.link) همان دستور جابگزینی می باشد که اشاره‌گر start را برای حذف گره از ابتدای لیست یک گره به سمت جلو هدایت می کند.

دستورات 1 ، 3 ، 4 به ترتیب برای: اشاره‌گر به گره حذف شونده (p : = start;) نگهداری داده گره حذف شونده (x : = p^.data;) ـ تخریب کامل گره حذف شونده (free (p)) در نظر گرفته شده است.

■ صف پیوندی

هر لیستی که عمل حذف از آن از یک طرف (ابتدای لیست) و عمل درج در آن در طرف دیگر لیست (انتهای لیست) انجام شود به عنوان صف پیوندی نامیده می‌شود.

الف) عمل حذف از ابتدای لیست (front) به شکل زیر انجام می‌شود:

```
if  (front = null)  then  'queue empty'
    else
        begin
            1) p : = front;
            2) front : = front^.link;
            3) x : = p^.data;
            4) if  (front = null)  then  rear : = nil;
            5) free(p);
        end;
```

توضیحات:

در صورتیکه شرط front = null برقرار باشد صف خالی بوده و امکان حذف از آن وجود ندارد.

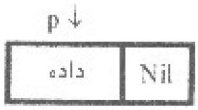
در صورتیکه به قسمت else برویم صف خالی نبوده اما اگر بعد ازعمل حذف با دستور (2) لیست خالی شود دستور شماره (4) به علت خالی شدن لیست rear را نیز null می کند.

در هر صف پیوندی بعد از هر عمل حذف ارزش گره حذف شده مورد ارزیابی قرار می‌گیرد برای همین در دستورات (1) و (3) مقدار گره حذف شده ارزیابی می‌شود.

آزادسازی گره حذف شده توسط دستور free (p) انجام می‌شود.

ب) عمل درج در صف پیوندی در انتها (rear) به شکل زیر انجام می‌شود:

```
1) p : = getnode( );
2) p ^.data : = داده ;   } → آماده سازی گره p برای درج در انتها
3) p ^.link : = nil;
4) if   front = nil   then   front : = p;
5) else   rear ^.link : = p;
6) rear : = p;
```

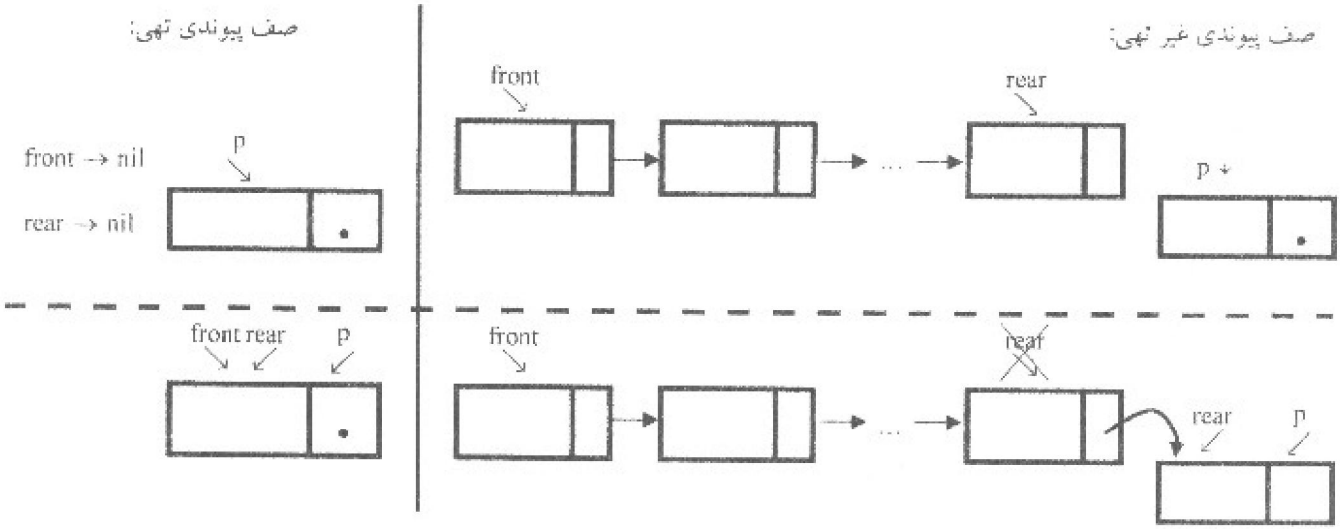


توضیحات:

4) در صورتیکه صف پیوندی تهی باشد (برقراری شرط front = nil) در نتیجه گره درج شونده تنها گره صف بوده و front هم باید به آن اشاره کند.

5) در صورتیکه صف پیوندی تهی نباشد (اجرای قسمت else) گره p انتهای لیست درج می‌شود جمله rear ^.link : = p باعث می‌شود اشاره‌گر Link گره آخر به p اشاره کند.

6) در هر وضعیتی بعد از درج p در انتهای صف پیوندی، اشاره‌گر rear باید به گره آخر که همان گره اضافه شده p است اشاره کند این عمل با دستور rear : = p انجام می‌شود.



معکوس کردن یک لیست پیوندی

برای معکوس هر لیست پیوندی با هر تعداد گره فقط با استفاده از 3 اشاره‌گر می‌توان این عمل را انجام داد.

در صورتیکه first اشاره‌گری به ابتدای یک لیست پیوندی باشد با استفاده از 3 اشاره‌گر p ، q ، r می‌توانیم لیست مورد نظر را معکوس کنیم:

در انتهای برنامه:

- اشاره‌گر q به ابتدای لیست معکوس شده اشاره می‌کند.
- اشاره‌گر p، null می‌شود.
- اشاره‌گر r، به یک گره بعد از q اشاره می‌کند.

```
p : = first;
q : = nil;
while   p < > nil   do
begin
1) r : = q;
2) q : = p;
3) p : = p^.link;
4) q^.link : = r;
end;
first : = q;
```

مثال: تابع زیر چه عملی انجام می‌دهد.

```
fun(s : list)
begin
if (s = nil)   then   return(nil)
else
return(cons(fun(tail(s)) , head(s)));
end;
```

cons (x , y) : لیست y را به انتهای لیست x متصل می‌کند.

حل: تابع بالا یک لیست پیوندی را معکوس می‌کند.

نتیجه‌گیری از لیست‌های پیوندی

مزایا:

۱ـ تخصیص پویا و متناسب برای داده‌ها برخلاف آرایه‌ها که تخصیص ایستا و محدود دارند.

۲ـ عملیات درج و حذف بدون نیاز به شیفت با $O(1)$ انجام می‌شود برخلاف آرایه‌ها که نیاز به شیفت برای این عمل دارند با $O(n)$.

معایب:

۱ـ اتلاف حافظه نسبت به آرایه‌ها به علت استفاده از فیلد اشاره‌گر بیشتر است.

۲ـ تنها روش جستجو: جستجوی خطی است و با زمان $O(n)$ برخلاف آرایه‌ها که جستجو دودویی را نیز دارند با زمان $O(\log_2 n)$.

۳ـ دسترسی به هر داده ترتیبی و از ابتدای لیست است با زمان $O(n)$ برخلاف آرایه‌ها که امکان دسترسی تصادفی با زمان $O(1)$ را دارند.

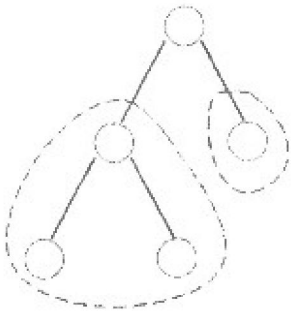
درخت (tree) : ساختمان دادهٔ غیرخطی

درخت: درخت مجموعه محدودی از یک یا چند گره به صورت زیر می‌باشد:

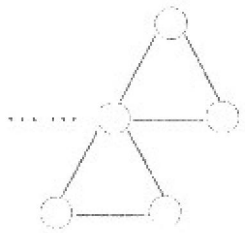
۱) دارای گره خاصی به نام ریشه

۲) بقیه گره‌ها به $n > 0$ مجموعه مجزا T_1 و T_2 و ... و T_n تقسیم شده که هر یک از این مجموعه‌ها خود یک درخت هستند. T_1 و T_{n-1} و ... و T_1 زیر درختان ریشه نامیده می‌شود.

شرط مجزا بودن زیر درختان T_1 و T_2 و ... و T_n آن است که هیچ اتصالی بین زیر درختان وجود ندارد.



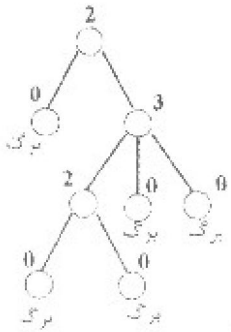
درخت است



(حلقه دارد) درخت نیست

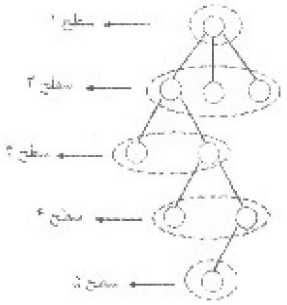
☑ **نکات و اصطلاح‌های مهم در مورد درختان:**

- **درجه یک گره:** تعداد زیر درخت‌های یک گره درجه آن نام دارد.
- **درجه درخت :** حداکثر درجه گره‌های درخت (بزرگترین درجه درخت)
- **برگ (گره‌های پایانی):** گره‌ها با درجه 0 برگ نام دارد.
- **گره‌های هم‌زاد (هم‌نیا):** فرزندان یک گره، گره‌های هم زاد نامیده می‌شود.
- **اجداد یک گره:** گره‌هایی هستند که در مسیر طی شده از ریشه تا گره دلخواهی وجود دارند.



درجه درخت = 3

- **سطح یک گره:** در صورتی که ریشه را در سطح ۱ بدانیم بقیه گره‌ها به تعداد کمانی که از ریشه فاصله دارند شماره سطح می گیرند.
- **ارتفاع یا عمق درخت:** به بیشترین سطح گره‌های یک درخت عمق یا ارتفاع درخت گفته می‌شود:

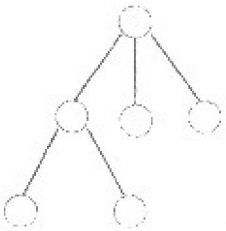


عمق یا ارتفاع = 5

در بعضی مراجع ریشه را در سطح 0 در نظر می گیرند. در این حالت ارتفاع درخت روبه‌رو به طور مثال 4 است.

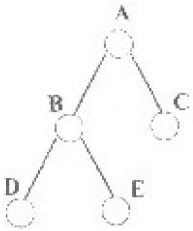
- **یال:** هر خط یا اتصال از یک گره به گره دیگر، یا به هر انشعاب از هر گره به گره دیگر.

نکته مهم: در هر درخت دلخواه | تعداد گره‌ها = تعداد یال‌ها + 1 |



(۵ + ۱ : **تعداد یال‌ها** = (۶ : **تعداد گره‌ها**

- **مسیر:** دنباله‌ای از یال‌های متوالی از یک گره به گره دیگر
- **شاخه:** مسیری که به یک برگ ختم شود شاخه نام دارد.



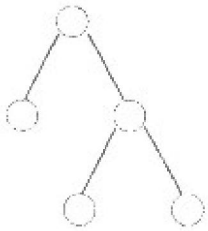
مسیر : A – B

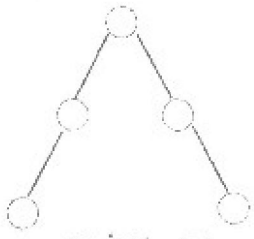
شاخه : A – B – D

درخت متوازن: درختی است که اختلاف سطح برگ‌های آن حداکثر 1 است.

درخت کاملاً متوازن: درختی است که اختلاف سطح برگ‌های آن 0 است.

درخت متوازن ⟸ هر درخت کاملاً متوازن


درخت متوازن

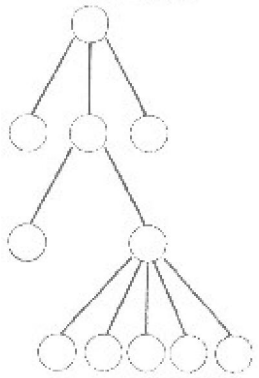

درخت کاملاً متوازن

هر درخت کاملاً متوازن، حتماً متوازن است.

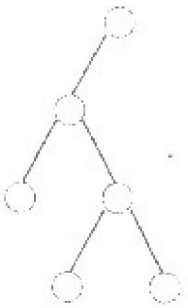
هر درخت متوازن، ممکن است کاملاً متوازن نباشد.

درخت K تایی: درختی که تعداد فرزندان هر گره حداکثر K است.

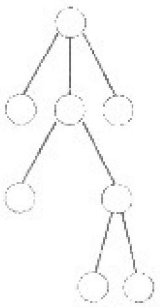
درخت 5 تائی



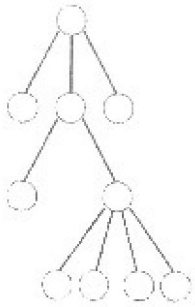
درخت 2 تائی



درخت 3 تائی



درخت 4 تائی



نکته مهم: در یک درخت K تائی با n گره:

nk ← تعداد کل اتصال‌ها

n - 1 ← تعداد اتصال‌های پر

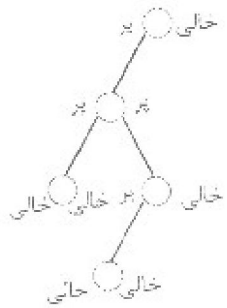
nk – (n – 1) ← تعداد اتصال‌های خالی

مثال: n = 5 گره از درجه K = 2:

کل اتصال‌ها nK = 2 × 5 = 10

اتصال‌های پر n – 1 = 4

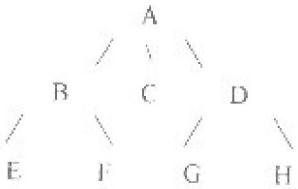
اتصال‌های خالی nK – (n – 1) = 6



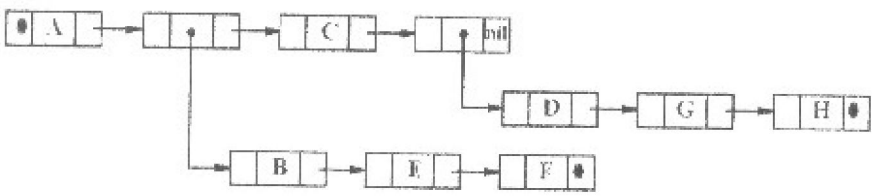
■ **نمایش درخت عمومی:**

۱ـ پرانتز گذاری:

(A (B (E , F) , C , D (G , H)))



۲ـ پیوندی:



در این نمایش فرزندان هر گره سمت راست آن هستند، اگر برگ باشد سمت راست و اگر برگ نباشند در یک سطح پایین‌تر سمت راست دیده می‌شوند.

☑ درخت K ثانی و ارتباط تعداد گره‌های یک فرزندی ـ دو فرزندی ـ ... با برگ‌ها

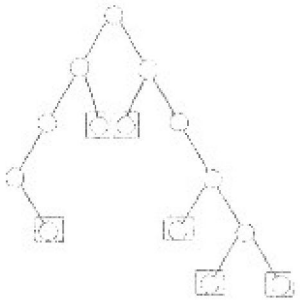
$\underbrace{\quad}_{n_0} \quad \underbrace{\quad}_{n_2} \quad \underbrace{\quad}_{n_K}$

در هر درخت تعداد برگ‌ها (n_0) مستقل از تعداد گره‌های تک فرزندی n_1 است و رابطهٔ زیر برای آن‌ها در یک درخت با n گره از درجه K وجود دارد.

$$n_0 = (K-1)n_K + (K-2)n_{K-1} + \cdots + 2n_3 + n_2 + 1$$

مهم: در هر درخت با n گره از درجه $K = 2$ $\boxed{n_0 = n_2 + 1}$

مثال ۱: در یک درخت دودوئی با $\begin{cases} n_2 = 5 \\ n_1 = 3 \\ n_0 = ? \end{cases}$ تعداد برگ‌ها کدام است؟



$n_0 = n_2 + 1 \rightarrow n_0 = 5 + 1 = \boxed{6}$

در هر درخت دودوئی همیشه: $\boxed{\text{تعداد برگ‌ها} = \text{تعداد گره‌ها با درجه 2} + 1}$

مثال ۲: در یک درخت از درجه 4 با $n_1 = 5$ و $n_2 = 6$ و $n_3 = 7$ و $n_4 = 9$ مطلوبست تعداد برگ‌ها؟

$$n_0 = 3n_4 + 2n_3 + n_2 + 1 = 3 \times 9 + 2 \times 7 + 6 + 1 = \boxed{48}$$

■ درخت دودوئی (binary):

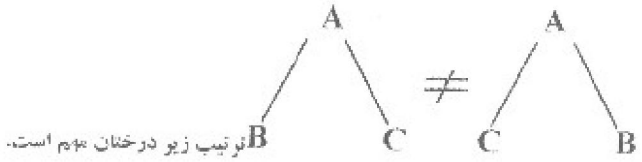
یک درخت دودوئی یا تهی است یا حاوی مجموعه‌ای محدود از گره‌ها شامل یک ریشه و هر گره در آن دارای حداکثر 2 فرزند است، برای هر گره دو زیر درخت چپ و راست می‌تواند وجود داشته باشد. (ترتیبشان مهم است)

در هر درخت دودوئی درجه هر گره حداکثر 2 است.

در هر درخت عادی ترتیب زیر درختان مهم نیست اما در درخت دودوئی ترتیب زیر درختان مهم است.

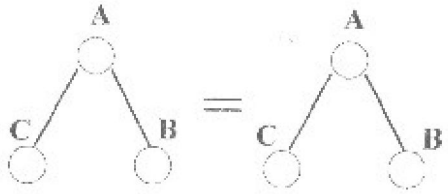
در هر درخت عادی گره صفر (تهی) وجود ندارد. اما در هر درخت دودوئی گره صفر (تهی) می‌تواند وجود داشته باشد.

✓ در هر درخت دودویی:

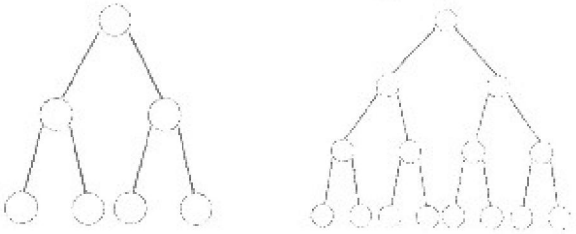


ترتیب زیر درختان مهم است.

✓ در درخت عادی:



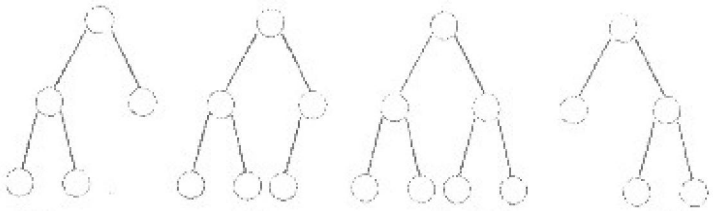
■ **درخت پر:** درخت دودویی، که همه گره‌های تمام سطوح در آن حداکثر فرزند یعنی دقیقاً 2 فرزند دارد



درخت دودویی پر به عمق 3

درخت دو دویی پر به عمق 4

■ **درخت کامل:** درختی را کامل گویند، اگر تمام سطح‌های آن به جز احتمالاً آخرین سطح حداکثر فرزندان را داشته باشد. و سطح آخر نیز از سمت چپ گره‌ها را پر می‌کند.



کامل — کامل — کامل (پر) — غیر کامل

درخت کامل
درخت کاملاً متوازن
درخت متوازن
درخت پر

متوازن
درخت کامل
کاملاً متوازن

درخت پر هم کامل است و هم کاملاً متوازن است.

☑ هر درخت پر هم کامل است؛ هم کاملاً متوازن، هم متوازن

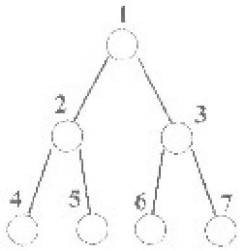
☑ هر درخت کامل، متوازن است. ولی همیشه پر یا کاملاً متوازن نیست.

■ **درخت مورب (اریب):** در درخت اریب به راست هر گره فرزند راست گره پدر خود است و در درخت اریب به چپ هر گره فرزند چپ گره پدر خود است.

نمایش درختان دودوئی:

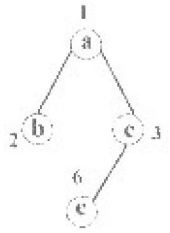
برای نمایش درختان دودوئی از دو روش ۱ـ آرایه ۲ـ لیست پیوندی استفاده می‌شود.

۱ـ **آرایه:** در این نمایش با فرض آن‌که ریشه را در سطح ۱ شماره ۱ بدهیم بقیه گره‌ها از سمت چپ به راست در سطوح بعدی می‌توانند شماره بگیرند.



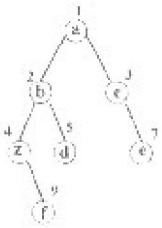
برای هر گره i فرزند راست \ 2i / فرزند چپ 2i + 1

مثال ۱:



1	2	3	4	5	6
a	b	c			e

مثال ۲:

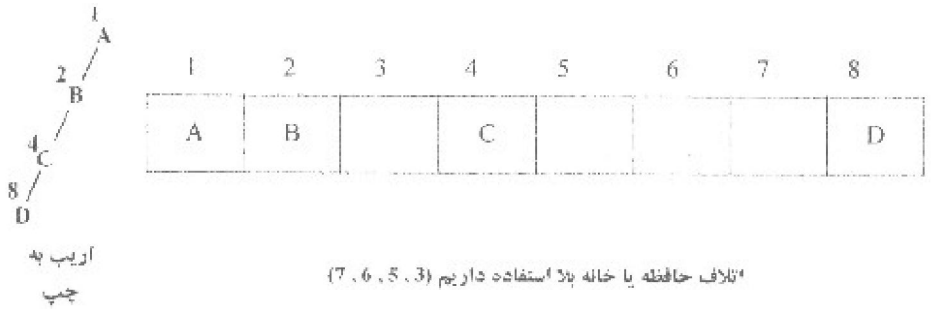


1	2	3	4	5	6	7	8	9
a	b	c	z	d		e		f

بهترین نمایش درخت با آرایه استفاده از درخت پر یا کامل است چون اتلاف حافظه و خانه بلا استفاده ندارد.

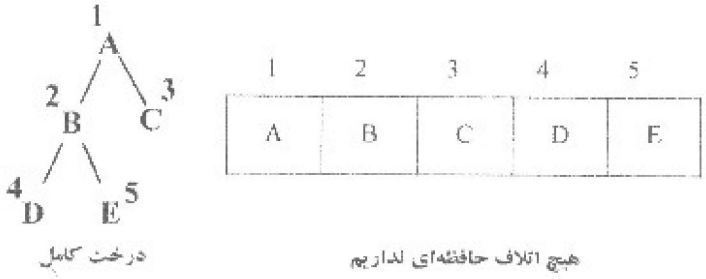
بدترین نمایش درخت با آرایه استفاده از درخت اریب به چپ یا راست است. که بیشترین اتلاف حافظه را در بر داشته است.

مثال:



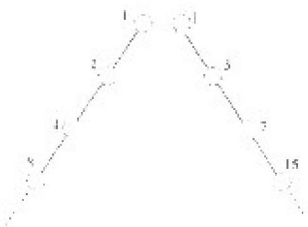
اتلاف حافظه با خانه بلا استفاده داریم (3، 5، 6، 7)

اریب به چپ



درخت کامل

هیچ اتلاف حافظه‌ای نداریم



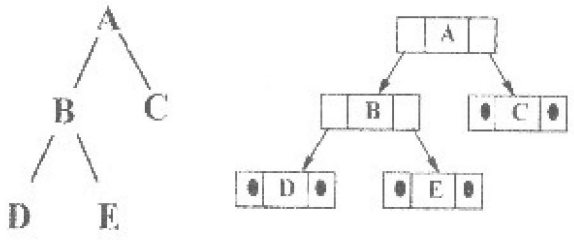
در درخت اریب به چپ با n گره احتیاج به 2^{n-1} خانه داریم که فقط از n خانه آن استفاده می‌شود.

در درخت اریب به راست با n گره احتیاج به $2^{n}-1$ خانه داریم که فقط از n خانه آن استفاده می‌شود. (اتلاف حافظه در این حالت بیشتر از اریب به چپ است.)

۱ـ نمایش پیوندی:

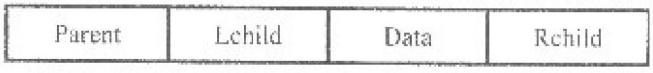
در نمایش آرایه دسترسی به خانه‌ها به راحتی انجام می‌شود اما حذف و درج نیاز به شیفت داشته و هزینه بر می‌باشد.

■ در نمایش پیوندی برای هر گره سه قسمت یا فیلد در نظر گرفته می‌شود. به صورت زیر:



آدرس فرزند راست داده آدرس فرزند چپ

✓ مهم: در بعضی شرایط اگر نیاز داشته باشیم از هر گره آدرس والد (پدر) آن را داشته باشیم در این صورت هر گره چهار فیلد خواهد داشت.



آدرس والد آدرس فرزند چپ داده آدرس فرزند راست

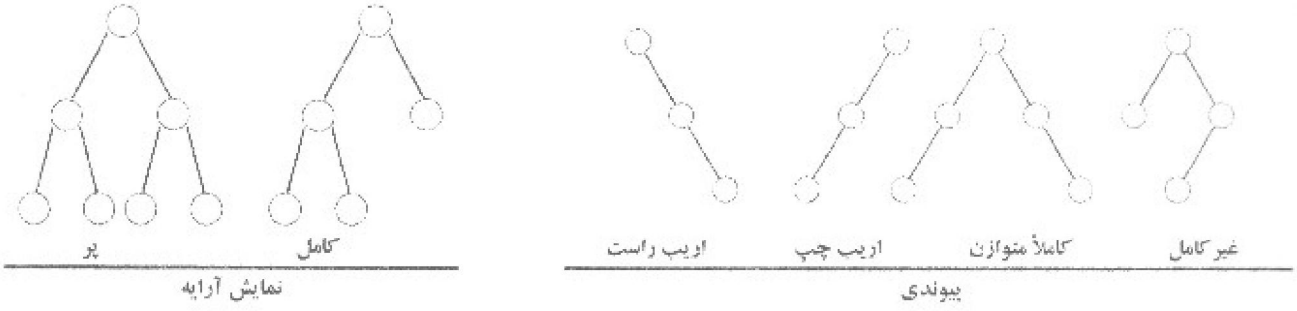
نکته تستی مهم: در نمایش درخت‌ها بهتر است از راه حل زیر حتماً استفاده کنیم.

نمایش آرایه: درخت کامل ـ درخت پر ـ درخت (max Heap ـ min Heap) Heap

نمایش پیوندی: درخت اریب ـ غیر کامل ـ جستجوی دودوئی (BST)

maxtree ـ mintree ـ متوازن ـ کاملاً متوازن

مثال:



نکات مهم درخت‌های دودوئی:

۱ـ حداکثر تعداد گره‌ها در سطح i ام یک درخت دودوئی 2^{i-1} (ریشه در سطح ۱)

۲ـ حداکثر تعداد کل گره‌ها در یک درخت دودوئی به عمق K برابر با $2^K - 1$ می‌باشد (ریشه در سطح ۱)

۳ـ حداقل تعداد کل گره‌ها در یک درخت دودوئی به عمق K، K است.

۴ـ حداقل تعداد کل گره‌ها در یک درخت دودوئی کامل به عمق K، 2^{K-1} است.

۵ـ در هر درخت دودوئی در صورتی که:

n_2 = تعداد گره‌های 2 فرزندی

n_0 = تعداد گره‌های پایانی (برگ‌ها) $\Rightarrow \quad n_0 = n_2 + 1$

۶ـ عمق درخت دودوئی پر با n گره: $\log_2(n+1)$

۷ـ عمق درخت دودوئی کامل یا پر: $\lfloor \log_2 n \rfloor = \lfloor \log_2 n - 1 \rfloor + 1$

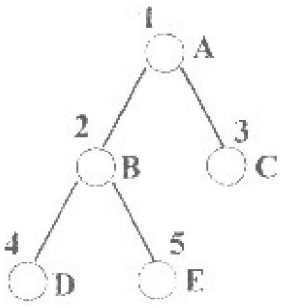
۸ـ اگر یک درخت دودوئی کامل با n گره داشته باشیم آنگاه برای هر $1 < i \leq n$

الف) اگر $i \neq 1$ باشد، پدر i در $\left\lfloor \frac{i}{2} \right\rfloor$ است. ($i = 1$ ریشه است و پدری ندارد.)

ب) اگر $2i \leq n$ باشد، آنگاه فرزند چپ i در $2i$ وجوددارد و گرنه اگر $2i > n$ باشد، i فرزند چپ ندارد.

ج) اگر $2i + 1 \leq n$ آنگاه فرزند راست i در $2i + 1$ است و گرنه اگر $2i + 1 > n$ باشد، i فرزند راست ندارد.

مثال ۱: $n = 5$ گره در یک درخت کامل به صورت روبرو وجود دارد.

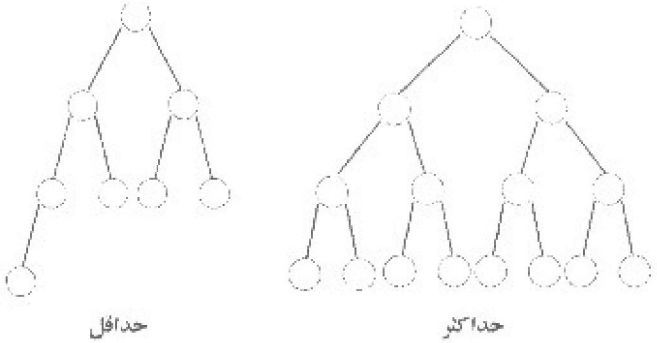


گره c با شماره $i = 3$ فرزند چپ یا راست ندارد.

چون $\boxed{5} > 2i = 6 >$ یا $\boxed{5} < 2i + 1 = 7 <$ است.

گره B با شماره $i = 2$ فرزند چپ و راست دارد. چون $\boxed{5} = > 2i = 4 < =$ و $\boxed{5} = > 2i + 1 = 5 < =$ است.

مثال ۲: در یک درخت دودوئی کامل به عمق 4 حداقل و حداکثر گره کدام است؟



حداکثر　　حداقل

حداقل (کامل) $2^{n-1} = 2^{4-1} = 2^3 = 8$

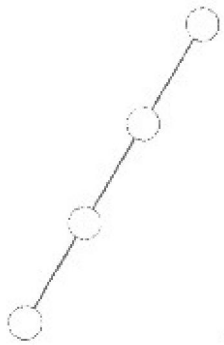
حداکثر (کامل و پر) $2^n - 1 = 2^4 - 1 = 16 - 1 = 15$

مثال ۳: در یک درخت دودوئی کامل با 1000 گره، عمق چقدر است؟

$$\lfloor \log_2 1000 \rfloor + 1 = \lfloor 9.97 \rfloor + 1 = 10$$

مثال ۴: در یک درخت دودوئی با 1000 گره،عمق چقدر است؟ نامعلوم

مثال ۵: در یک درخت دودوئی با عمق 4 حداقل گره،چقدر است؟ (کامل نیست)



4 گره

نکته مهم: در یک درخت K تائی کامل با n گره:

$$\text{تعداد برگ‌ها} = \left\lfloor \frac{nK - (n-1)}{K} \right\rfloor$$

کل اتصال‌ها　اتصال پر

در حالت خاص زمانی که درخت کامل از درجه دودویی باشد:

$$\text{تعداد برگ‌ها} = \left\lfloor \frac{n+1}{2} \right\rfloor$$

مثال ۱: در یک درخت دودویی کامل با 20 گره تعداد برگ‌ها کدام است؟

$$\text{تعداد برگ‌ها} = \left\lfloor \frac{20+1}{2} \right\rfloor = 10$$

مثال ۲: در یک درخت کامل از درجه $K = 3$ با $n = 10$ گره تعداد برگ‌ها کدام است؟

$$\left\lfloor \frac{10 \times 3 - (10-1)}{3} \right\rfloor = \left\lfloor \frac{21}{3} \right\rfloor = 7$$

✓ **یادآوری:** تعداد برگ‌ها همیشه مستقل از گره‌های تک‌فرزندی است.

نکته مهم: همیشه می‌توان رابطه $\frac{\binom{2n}{n}}{n+1} = \frac{\frac{(2n)!}{n!\,n!}}{n+1}$ را برای موارد زیر استفاده کرد:

۱) تعداد درخت‌های دودویی متفاوتی که می‌توان با n گره ساخت.

۲) تعداد حالت‌های مختلف خروجی که می‌توان برای n ورودی یک stack در نظر گرفت.

۳) تعداد حالت‌های مختلف ضرب $n+1$ ماتریس با استفاده از خاصیت شرکت‌پذیری.

مثال: تعداد درختان دودویی که می‌توان با 3 گره ساخت:

$$\frac{\binom{2n}{n}}{n+1} = \frac{\binom{6}{3}}{3+1} = \frac{\frac{6!}{3!\,3!}}{4} = 5$$

عدد 5 بدست آمده در بالا همین‌طور

- تعداد حالت‌های مختلف خروجی پشته با 3 ورودی
- تعداد حالت‌های مختلف ضرب 4 ماتریس

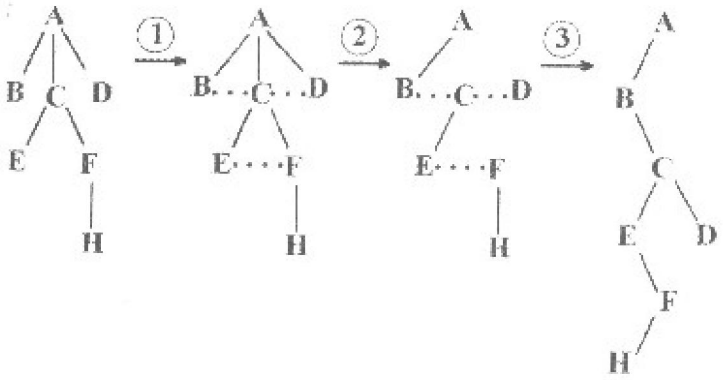
نیز می‌باشد.

☑ **تبدیل درخت عادی (عمومی) به درخت دودویی:**

۱ـ تمام فرزندان هر گره را به هم متصل می‌کنیم.

۲ـ برای هر گره اتصال کلیه فرزندان را به جز اتصال چپ حذف می‌کنیم.

۳ـ گره‌های متصل به هم در سطح افقی را 45 درجه موافق عقربه‌های ساعت حرکت می‌دهیم:



ظاهراً تمام اتصال‌های افقی فرزند راست می‌شوند و اتصال‌های عمودی فرزند چپ.

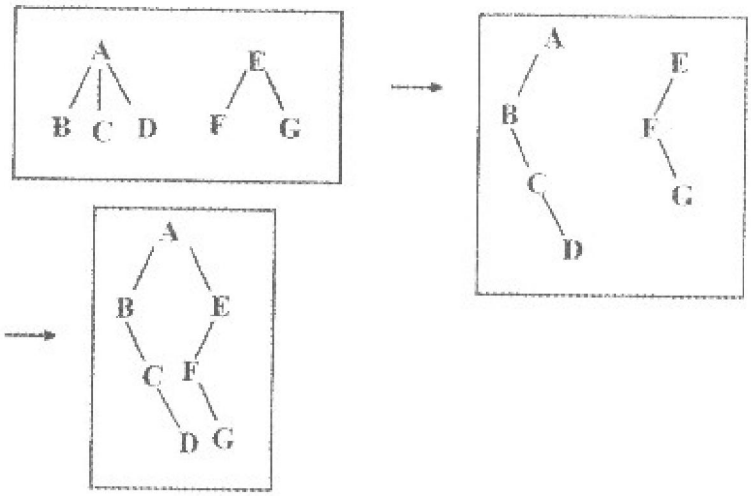
نکته: پیمایش میانوندی inorder (LVR) درخت دودوئی بدست آمده با پیمایش پسوندی LRV Postorder درخت عمومی آن متناظر است.

■ **جنگل** (forest): جنگل بنا بر تعریف مجموعه‌ای مرتب از صفر یا چند درخت متمایز است. به عبارت دیگر جنگل مجموعه‌ای از $n \geq 0$ درخت مجزا است.

اگر ریشه یک درخت را حذف کنیم آن‌گاه دارای جنگل یا درخت‌هایی به تعداد فرزندان ریشه است.

■ **تبدیل جنگل به درخت دودوئی:**

ابتدا نمایش دودوئی هر یک از درختان جنگل را بدست آورده سپس از چپ به راست ریشه هر درخت را به عنوان فرزند راست درخت سمت چپ در نظر می‌گیریم:

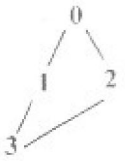


نکته مهم: یکی از کاربردهای درخت نمایش مجموعه‌ها است.

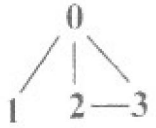
مثال مهم: کدام گزینه درخت است.

گزینه ۳ صحیح می‌باشد.

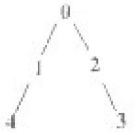
1) $\{\langle 0,1\rangle, \langle 0,2\rangle, \langle 1,3\rangle, \langle 3,2\rangle\}$ → (گراف) حلقه دارد



2) $\{\langle 0,1\rangle, \langle 0,2\rangle, \langle 0,3\rangle, \langle 2,3\rangle\}$ → (گراف) حلقه دارد



3) $\{\langle 0,1\rangle, \langle 0,2\rangle, \langle 2,3\rangle, \langle 1,4\rangle\}$ → (درخت) حلقه ندارد



4) هیچکدام

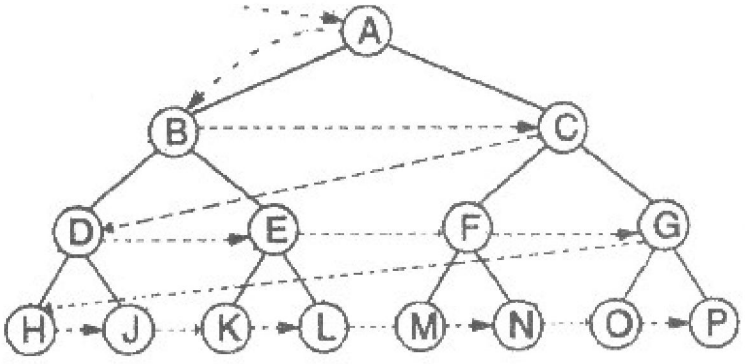
پیمایش درخت‌ها دودوئی

در پیمایش درخت دودوئی می‌خواهیم به هر گره فقط یک‌بار دسترسی داشته باشیم.

۱) پیمایش درختان در دو حالت عمقی و سطحی انجام می‌شود.

۲) در پیمایش سطحی (عرضی، ردیفی، level order) پیمایش را از ریشه آغاز کرده و از سطح دوم به بعد در هر سطح، گره‌ها را، از سمت چپ به راست رویت کرده و به سطح بعدی می‌رویم تا این‌که سطح آخر نیز دیده شود.

مثال:



A B C D E F G H J K L M N O P

۳) در پیمایش عمقی با توجه به آن‌که پیمایش از ریشه آغاز می‌گردد. برای هر گره از بالا به پائین سه حالت L (چپ) یا R (راست) یا V . D . N (مقدار Data) وجود دارد.

۴) با توجه به سه حالت بیان شده، 6 حالت پیمایش مختلف به‌وجود می‌آید.

اول زیر درخت چپ، دوم ریشه، سوم زیر درخت راست	میانوندی	inorder ⟵	LVR
اول زیر درخت چپ، دوم زیر درخت راست، سوم ریشه	پسوندی	postorder ⟵	LRV
اول ریشه، دوم زیر درخت چپ، سوم زیر درخت راست	پیشوندی	preorder ⟵	VLR
			VRL
			RVL
			RLV

الگوریتم پیمایش درخت:

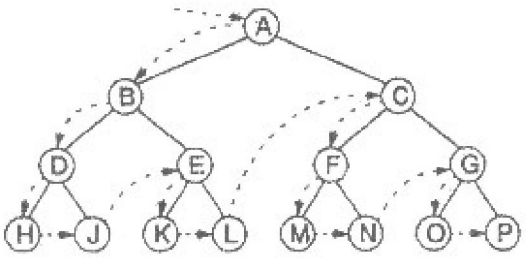
روش پیمایش preorder:

در این روش، درخت دودویی غیرخالی را به صورت زیر ملاقات کنید:

۱ـ ریشه درخت را ملاقات کنید.
۲ـ اگر زیر درخت چپ خالی نیست، آن را به روش preorder پیمایش کنید.
۳ـ اگر زیر درخت راست خالی نیست، آن را به روش preorder پیمایش کنید.

پیمایش preorder به این صورت است: از ریشه شروع کرده آن را ملاقات می‌کنیم. سپس به سمت چپ حرکت کرده اطلاعات این گره را می‌نویسیم و این روند را تا رسیدن به منتهی‌الیه سمت چپ ادامه می‌دهیم. پس از رسیدن به آخرین گره سمت چپ و ملاقات آن، به سمت راست حرکت می‌کنیم (چنانچه حرکت به سمت راست ممکن نباشد به گره بالاتر می‌رویم) و زیر درخت سمت راست آن را ملاقات می کنیم و این روند را برای تمام گره‌های درخت ادامه می‌دهیم. شکل ۱ شیوه پیمایش preorder یک درخت دودویی را نشان می‌دهد. تابع preorder ( ) که در زیر آمده است، درخت دودویی را به روش preorder (پیشوندی، VLR ، DLR ، NLR) پیمایش می‌کند:

```
Procedure preorder (t :treepointer );
 begin
    if (t< > nil) then
       begin
          write (data (t));
          preorder (Lchild (t));
          preorder (Rchild (t));
       end;
 end;
```



ABDHJEKLCFMNGOP

شکل ۱: شیوه پیمایش preorder درخت دودویی

روش پیمایش postorder

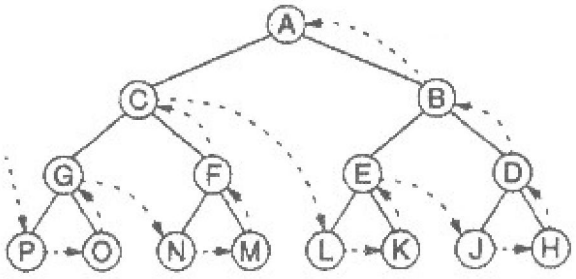
برای پیمایش postorder یک درخت دودویی غیرخالی، الگوریتم زیر را اجرا کنید:

۱ـ اگر زیر درخت چپ خالی نیست، آن را به روش postorder ملاقات کنید.
۲ـ اگر زیر درخت راست خالی نیست، آن را به روش postorder ملاقات کنید.
۳ـ ریشه درخت را ملاقات کنید.

پیمایش postorder به این صورت است: از ریشه شروع می‌کنیم و به طرف چپ حرکت می‌کنیم تا به منتهی‌الیه سمت چپ برسیم (یعنی به گره‌ای برسیم که فیلد Left آن تهی است). از این گره شروع کرده تا جایی که ممکن است به سمت راست حرکت می‌کنیم. چنانچه حرکت به سمت راست ممکن نباشد، محتویات این گره را می‌نویسیم و به گره بالایی می‌رویم و با این گره مانند ریشه رفتار می‌کنیم این روند را برای ملاقات تمام گره‌های درخت ادامه می دهیم. شکل ۲ شیوه پیمایش postorder یک درخت دودویی را نشان می‌دهد.

تابع ( ) postorder که در زیر آمده است، درخت را به شیوه postorder (پسوندی، LRV) پیمایش می کند:

```
Procedure postorder (t : treepointer);
 begin
    if (t< > nil)  then
       begin
          postorder (Lchild(t));
          postorder (Rchild(t));
          write (data (t));
       end;
 end;
```



POGNMFCLKEJHDBA

شکل ۳: شیوه پیمایش postorder درخت دودویی

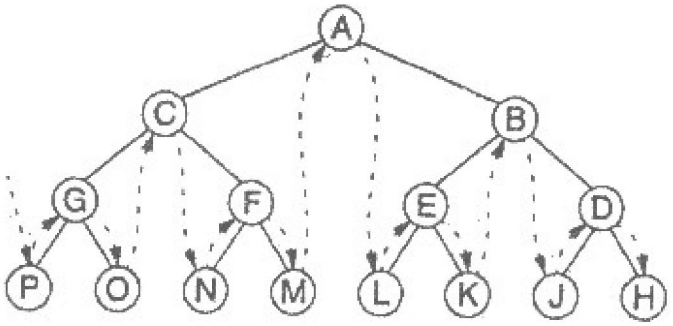
روش پیمایش inorder

برای پیمایش inorder یک درخت دودویی غیرخالی مراحل زیر را انجام دهید:

۱ـ اگر زیر درخت چپ خالی نیست، آن را به روش inorder پیمایش کنید.
۲ـ ریشه درخت را ملاقات کنید.
۳ـ اگر زیر درخت راست خالی نیست، آن را به روش inorder پیمایش کنید.

پیمایش inorder به این صورت است: از ریشه شروع کرده تا جایی که ممکن است به سمت چپ حرکت می‌کنیم. با رسیدن به آخرین گره سمت چپ (گره‌ای که فیلد Left آن تهی است)، محتویات آن گره را می‌نویسیم و سپس به سمت راست حرکت می‌کنیم و با آن مثل گره ریشه برخورد می‌کنیم و به منتهی الیه سمت چپ می‌رویم و آن گره را ملاقات می‌کنیم. اگر در گره‌ای حرکت به سمت راست ممکن نباشد، یک گره به سمت بالا حرکت کرده آن را ملاقات می‌کنیم و سپس به سمت راست حرکت می‌کنیم. این روند را تا ملاقات کردن کلیه گره‌های درخت ادامه می‌دهیم. شکل ۳ شیوه پیمایش درخت دودویی را به روش inorder نشان می‌دهد. تابع ( ) inorder که در زیر آمده است درخت را به روش inorder (میانوندی، LVR) پیمایش می‌کند:

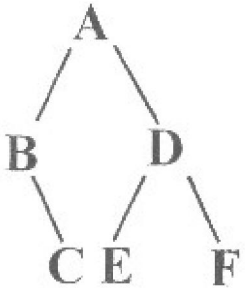
```
Procedure inorder (t:treepointer);
 begin
    if (t<>nil) then
       begin
          inorder(Lchild(t));
          write (data(t));
          inorder (Rchild (t));
       end;
 end;
```



PGOCNFMALEKBJDH

شکل ۳: شیوه پیمایش درخت دودویی به روش inorder

درهر سه پیمایش اول از چپ شروع می‌کنیم (یعنی جهت شروع چپ است)



LVR: BCAEDF

VLR: A BCDE F

ریشه ← سمت راست‌ترین برگ

LRV: C BEFD A

ریشه سمت چپ‌ترین برگ

نکته ۱: ترتیب دیدن برگ‌ها در هر سه پیمایش یکسان است.

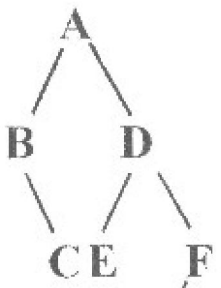
نکته۲: با داشتن (1) $\begin{cases}\text{inorder}\\ \text{postorder}\end{cases}$ یا (2) $\begin{cases}\text{inorder}\\ \text{preorder}\end{cases}$ می‌توان به درخت واحد رسید.

نکته۳: با داشتن preorder و postorder ممکن است به درخت واحدی نرسیم.

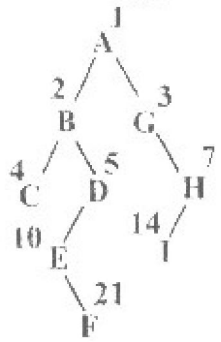
مگر آن‌که درخت پر باشد و همه گره‌ها 2 فرزندی باشند. چون وضعیت چپ یا راست بودن گره‌های تک فرزندی فقط با inorder مشخص می‌شود.

مثال: مطلوبست نمایش هر سه پیمایش درختان زیر:

	Left	Right	info
1	2	3	A
2	0	4	B
3	5	6	D
4	0	0	C
5	0	0	E
6	7	0	F
7	0	0	G



LVR: B C A E D G F
VLR: A B C D E F G
LRV: C B E G F D A



LVR: C B E F D A G 1 H
VLR: A B C D L F G H I
LRV: C F L D B I H G A

1	2	3	4	5	6	7	8	9	10	...	14	...	21
A	B	G	C	D		H			E	...	I	...	I

مشاهده می‌شود که در پیمایش LRV ابتدا برگ‌ها زیر درخت چپ دیده می‌شود سپس به سمت بالا و سپس برگ‌های زیر درخت راست سپس بالا و در آخر ریشه درخت دیده می‌شود.

نکته مهم:

RVL = معکوس LVR (inorder)

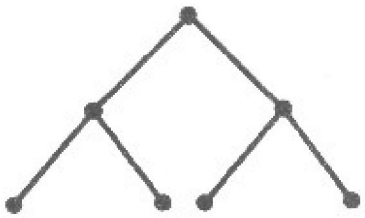
RLV = معکوس VLR (preorder)

VRL = معکوس LRV (postorder)

پس با سه پیمایش inorder و preorder و postorder می‌توانیم با معکوس کردن آن‌ها پیمایش‌های راست را بدست آوریم.

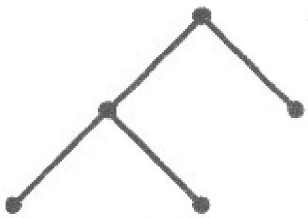
نکته بسیار مهم:

در بعضی مراجع و تست‌های کنکور تعاریف مختلفی از درخت‌های دودوئی کامل پر و محض وجود دارد به شرح زیر:



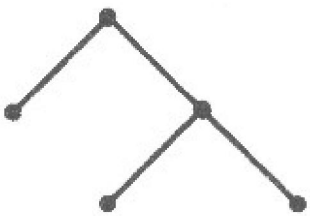
۱- پر

۲- کاملاً پر



کامل

پر



دودوئی محض (توسعه یافته)

کامل

تمام بحث‌های این مرجع براساس مورد ۱ می‌باشد.

بررسی برابری دو درخت دودویی

اگر دو درخت دودویی ساختاری نظیر هم داشته و اطلاعات موجود در گره‌های نظیرشان با هم برابر باشد، آن دو درخت برابر تلقی می‌شوند.

منظور از ساختار نظیر این است که هر انشعاب از درخت اول با انشعابی در درخت دوم مطابقت داشته باشد.

الگوریتم مطرح شده در این‌جا، درخت را به شیوهٔ پیشوندی پیمایش می‌کند ولی می‌توان از هر روش دیگری استفاده کرد.

```
function   equal (t1 , t2 : tree pointer): boolean;
    begin
       equal : = FALSE;
       if   (t1 = nil)   and   (t2 = nil)   then   equal : = TRUE
       else
          if   (t1 < > nil)   and   (t2 < > nil)   then
             if   data (t1) = data (t2)   then
                if   equal (Lchild (t1) , Lchild (t2))   then
                   equal : = equal (Rchild (t1) , Rchild (t2));
    end;
```

درخت نخی دودویی

در یک درخت دودویی با n گره، 2n اشاره‌گر وجود دارد که از این تعداد n - 1 اشاره‌گر مورد استفاده قرار گرفته‌اند و n + 1 اشاره‌گر مقدار nil دارند. با استفاده از این اشاره‌گرهای بدون استفاده می‌توان به عناصر قبلی یا بعدی در یک پیمایش خاص اشاره نمود و در نتیجه در هنگام پیمایش به نحو سریع‌تری درخت را پیمایش کرد. درختی که اشاره‌گرهای بدون استفاده آن بدین صورت مورد استفاده قرار گرفته است، درخت نخی یا درخت نخ کشی شده نام دارد.

البته با وجود این مزیت، اکنون باید اشاره‌گرهای واقعی از اشاره‌گرهای نخی به نحوی متمایز شوند و بدین منظور باید در هر گره بخشی برای مشخص کردن نوع اشاره‌گر موجود باشد.

معمولا اشاره‌گر سمت چپ (Lchild) در صورتی که بلااستفاده باشد، جهت اشاره به عنصری استفاده می‌شود که در یک پیمایش مورد نظر، قبل از این عنصر قرار دارد و اشاره‌گر سمت راست (Rchild) در صورت بدون استفاده بودن برای اشاره به عنصر بعدی در پیمایش مورد نظر، مورد استفاده قرار می‌گیرد.

برای امکان تشخیص اشاره‌گرهای واقعی از اشاره‌گرهای نخی دو فیلد منطقی به هر گره درخت افزوده می‌شود که Ltag و Rtag نامیده می‌شوند. ساختار گره در چنین درختی به صورت زیر خواهد بود:

Ltag	Lchild	Data	Rchild	Rtag

ساختار گره در درخت دودویی نخی

اگر Ltag = 1 باشد Lchild دارای یک اشاره‌گر نخی است و در غیر این صورت اشاره‌گر عادی به فرزند چپ است و به همین صورت اگر Rtag = 1 باشد Rchild از نوع نخ و در غیر این صورت (Rtag = 0) Rchild اشاره‌گر عادی به فرزند راست است.

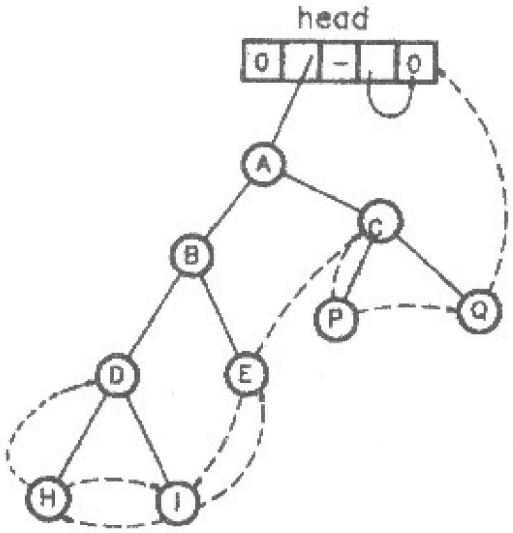
تشکیل درخت نخی براساس پیمایش پیشوندی

در این حالت برای ایجاد نخ‌ها از اتصالات تهی به صورت زیر استفاده می‌شود:

Rchild (p): به عنصری که در پیمایش پیشوندی بعد از p می‌آید، اشاره می‌کند.

Lchild (p): به عنصری که در پیمایش پیشوندی قبل از p می‌آید، اشاره می‌نماید.

preorder : A B D H I E C P Q



درخت نخی در پیمایش پیشوندی

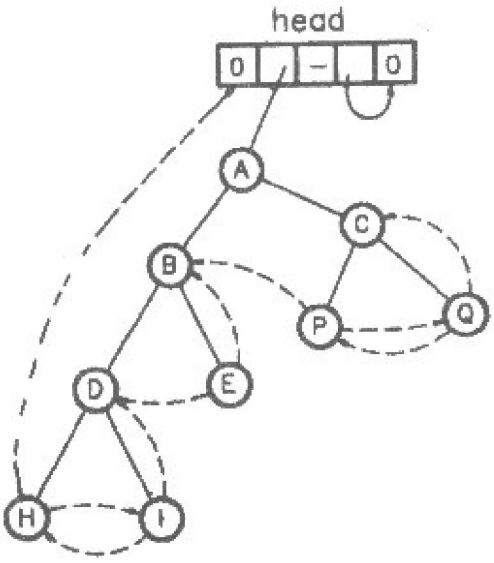
تشکیل درخت نخی براساس پیمایش پسوندی

در این حالت نیز از اتصالات تهی به صورت زیر استفاده می‌شود:

Rchild (p): به عنصری که در پیمایش پسوندی بعد از p می‌آید، اشاره می‌کند.

Lchild (p): به عنصری که در پیمایش پسوندی قبل از p می‌آید، اشاره می‌نماید.

postorder : H I D E B P Q C A



درخت تخی در پیمایش پسوندی

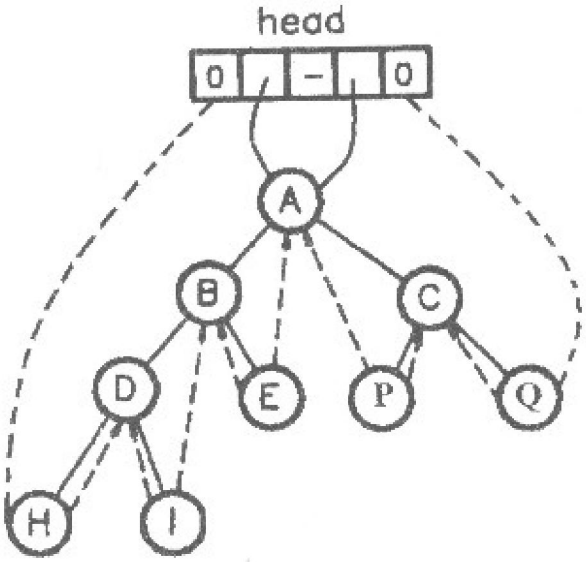
تشکیل درخت تخی براساس پیمایش میانوندی

در این حالت برای ایجاد تخ‌ها از اتصالات تهی به صورت زیر استفاده می‌شود:

Rchild (p): به عنصری که در پیمایش میانوندی بعد از p می‌آید، اشاره می‌کند.

Lchild (p): به عنصری که در پیمایش میانوندی قبل از p می‌آید، اشاره می‌نماید.

inorder : H D I B E A P C Q



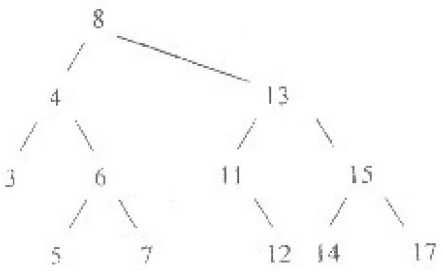
درخت جستجوی دودوئی (BST) ←——→ (binary search tree)

درخت جستجوی دودوئی، درخت دودوئی است که در آن:

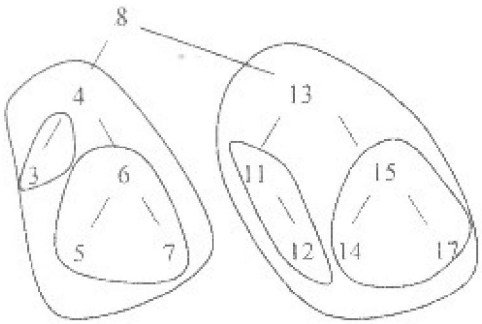
۱ـ مقدار هر گره از کلیه مقادیر گره‌های زیر درخت چپ بزرگتر و از کلیه مقادیر گره‌های راست کوچکتر باشد.

یا

۲ـ کلیه مقادیر گره‌های زیر درخت چپ هر گره از کلیه مقادیر گره‌های زیر درخت راست هر گره کوچکتر باشد.



نکته: دیده می‌شود که مقادیر زیر درخت چپ گره 8 از زیر درخت راست آن کوچکتر و همین امر برای گره 13 و گره 4 و ریشه بقیه زیر درختان صادق است.



■ مشخصات مهم در درخت BST

۱ـ هیچ عنصر تکراری در درخت وجود ندارد. (یکی از کاربردهای BST حذف عناصر تکراری است)

۲ـ پیمایش inorder (LVR) هر درخت BST عناصر را به صورت مرتب شده صعودی (sort) نشان می‌دهد. (یکی از کاربردهای BST مرتب سازی عناصر است)

■ پیمایش inorder در درخت BST شکل بالا به صورت زیر است:

3 , 4 , 5 , 6 , 7 , 8 , 11 , 12 , 13 , 14 , 15 , 17

۳ـ با شروع از ریشه درخت و حرکت به سمت چپ تا رسیدن به گره‌ای که سمت چپ آن گره تهی است به گره‌ای می‌رسیم که مقدار آن در درخت می‌نیمم است.

```
p = root
while (p ^ . Left ≠ null)
p = p ^ . Left;
write (p ^ . data) → (3 مقدار می‌نیمم (در مثال بالا
```

۴ـ با شروع از ریشه درخت و حرکت به سمت راست تا رسیدن به گره‌ای که سمت راست آن تهی (null) است به گره‌ای می‌رسیم که مقدار آن در درخت ماکزیمم است.

```
p = root
while (p ^ . Right ≠ null)
p = p ^ . Right
write (p ^ . data); → (مقدار ماکزیمم (در مثال بالا 17)
```

۵ـ مهمترین عمل در درخت BST جستجو می‌باشد که از ریشه آغاز شده و حداکثر تا عمق درخت ادامه می‌یابد به عبارت بهتر:

الف) حداقل مقایسه 1 مقایسه خواهد بود در صورتیکه عنصر جستجو شونده ریشه باشد. (در درخت بالا 8)

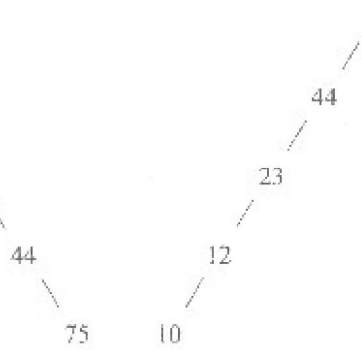
ب) حداکثر مقایسه به اندازهٔ عمق (تعداد سطوح) درخت خواهد بود در صورتیکه عنصر جستجو شونده یکی از عناصر گره‌ها در پائین‌ترین سطح باشد (در درخت بالا 5 , 7 , 14 , 17)

۶ـ در صورتیکه h عمق درخت باشد زمان جستجو $O(h)$ خواهد بود که در حالت متوسط $h = \log_2 n$ و زمان جستجو از مرتبه $O(\log_2 n)$ خواهد بود.

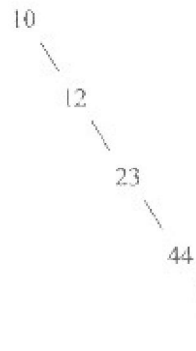
۷ـ در یک درخت BST با n گره با عمق k زمان جستجو $O(k)$ خواهد بود.

۸ـ در یک درخت BST با n گره حداکثر مقایسه n خواهد بود زمانیکه درخت به صورت اریب به راست یا چپ باشد.

در این درخت BST با 5 گره، گره 10 با 5 مقایسه بدست می‌آید.

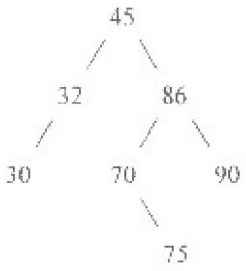


در این درخت BST با 5 گره، گره 75 با 5 مقایسه بدست می‌آید.



۹ـ در عین حال حداقل مقایسه زمانی، اتفاق می‌افتد که عنصر جستجو شونده ریشه باشد و با 1 مقایسه عنصر پیدا شود.

مثال: عنصر 75 در درخت BST زیر با چند مقایسه پیدا می‌شود.



جستجو از ریشه آغاز شده و به ترتیب با 4 مقایسه گره 75 یافت می‌شود: 45 راست 86 چپ 70 راست 75.

نتیجه‌گیری:

- در یک درخت BST با n گره با عمق h زمان جستجو $O(h)$ که متوسط این زمان وضعیتی است که $h = \log_2 n$ باشد و زمان $O(\log n)$ خواهد بود.
- در یک درخت BST به صورت اریب با n گره و عمق h حداکثر مقایسه n خواهد بود. که بدترین وضعیت است

۹ـ تعداد مقایسه برای رسیدن به این نتیجه که عنصری در درخت BST وجود ندارد حداکثر به عمق درخت خواهد بود و زمان آن $O(h)$ (h عمق درخت) خواهد بود. و بدترین وضعیت زمانی است که درخت اریب باشد در این حالت حداکثر مقایسه به تعداد گره‌های درخت (n) خواهد بود.

مثال: با چند مقایسه مشخص می‌شود عنصر 35 در درخت BST مثال بالا وجود ندارد. جستجو از ریشه آغاز شده و با 2 مقایسه زیر مشخص می‌شود گره 35 وجود ندارد: 45 $\xrightarrow{\text{چپ}}$ 32 $\xrightarrow{\text{راست}}$ تهی

درج در گره BST

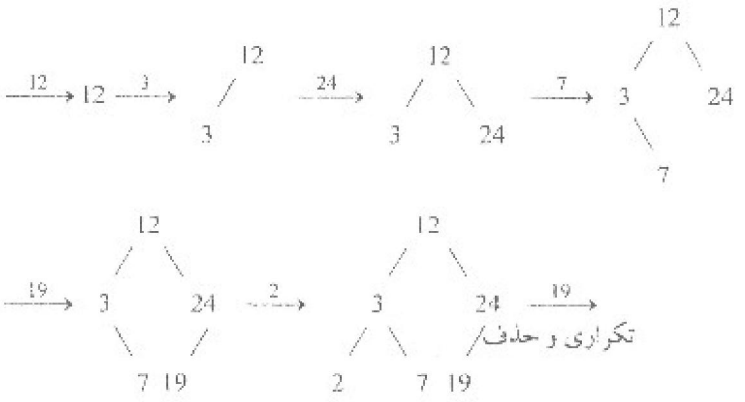
برای عمل درج ابتدا عمل جستجو برای گره درج شونده باید انجام شود تا مطمئن شویم گره مورد نظر در درخت BST وجود نداشته باشد. (در BST عنصر تکراری وجود ندارد) در این وضعیت 2 حالت پیش می‌آید.

۱ـ عنصر درج شونده در درخت پیدا شود در این صورت از لیست ورودی حذف شده و دیگر وارد درخت نمی‌شود.

۲ـ عنصر درج شونده در درخت وجود ندارد. در این صورت بعد از جستجو به محض رسیدن به بن بست (یعنی محلی که دیگر نمی توانیم به سمت چپ یا راست برویم) عنصر درج شونده را در همان محل درج می‌کنیم.

دقت می‌کنیم که برای درج هر گره جستجو مقایسه از ریشه آغاز شده و سعی داریم درخت، حالت BST بودن خود را حفظ کند.

مثال: مطلوبست درج آرایه زیر در یک درخت BST تهی: 12 , 3 , 24 , 7 , 19 , 2 , 19



نحوه عملیات :

12: بدون مقایسه در ریشه درخت

3: 1 مقایسه با 12 و سمت چپ 12

24: 1 مقایسه با 12 و سمت راست 12

7: 2 مقایسه به ترتیب با 12 , 3 و سمت راست 3

19: 2 مقایسه به ترتیب با 12 ، 24 و قرار گرفتن در سمت چپ 24

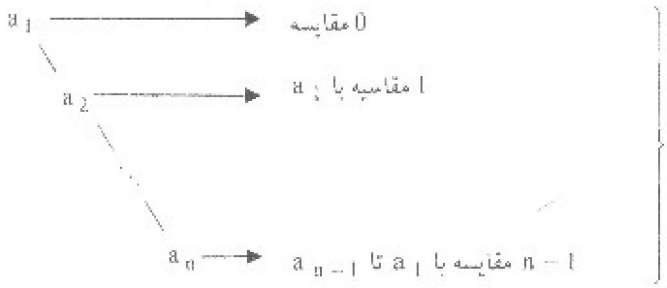
2: 2 مقایسه به ترتیب با 12 ، 3 و قرار گرفتن در سمت چپ 3

19: 3 مقایسه به ترتیب 12 ، 24 ، 19 و به علت تکراری بودن از لیست ورودی حذف می‌شود.

بدترین وضعیت:

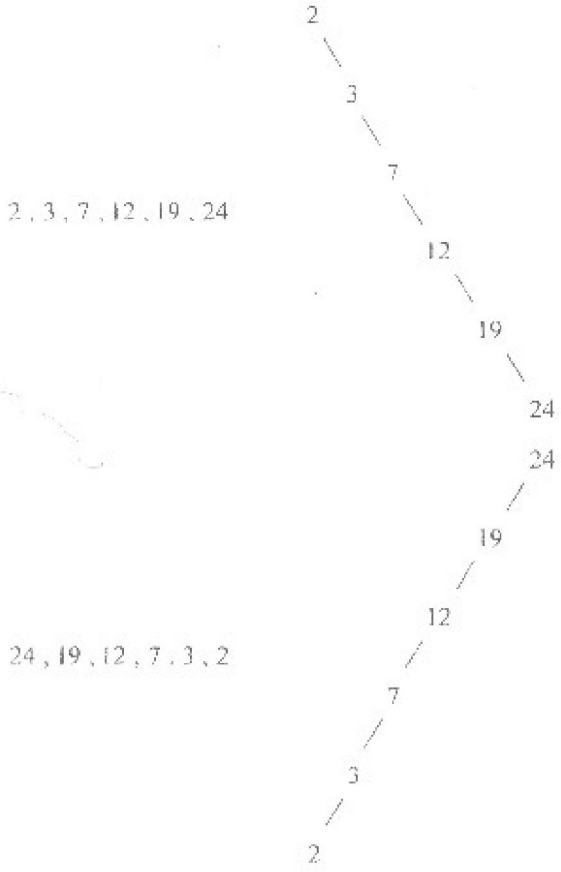
بدترین حالت در درخت BST ورود عناصر به صورت صعودی یا نزولی بوده که باعث می‌شود درخت به صورت اریب به راست یا چپ ایجاد شود.

فرض کنید $a_1 > a_2 > a_3 > \ldots > a_n$ در نتیجه اگر داده‌ها به صورت $a_1 , a_2 , \ldots , a_n$ وارد شوند.



در این حالت تعداد مقایسات برای درج n داده $\frac{n(n-1)}{2}$ می‌شود.

مثال:



نکته مهم: عمق درخت BST همیشه بستگی به نحوهٔ ورود داده‌ها دارد.

بهترین وضعیت:

زمانی که داده‌ها به صورت نامرتب وارد شوند، در این صورت درخت دارای ارتفاع متوازن $O(\log n)$ خواهد بود.

زمان درج در BST:

برای درج هر عنصر در BST دو عمل انجام می‌شود:

۱ـ جستجو برای عنصر درج شونده تا تکراری نباشد با زمان $O(h)$

۲ـ درج گره مورد نظر در صورت تکراری نبودن در محل تهی پست با زمان $O(1)$

در مجموع زمان درج هر گره $O(h)$ خواهد بود؛

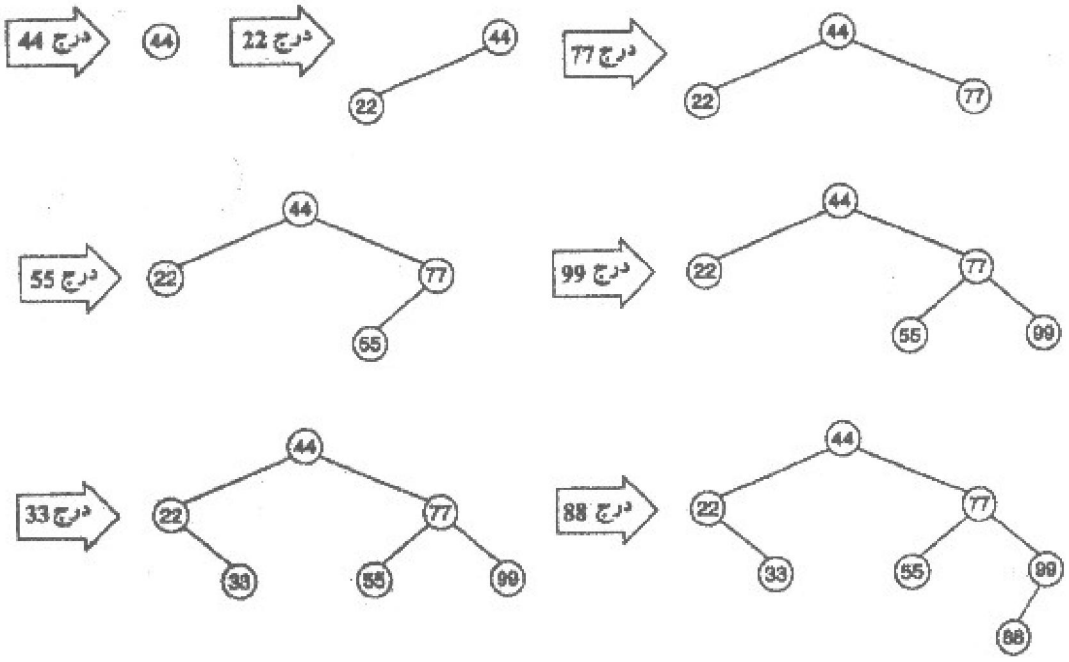
که دو حالت وجود دارد

۱ـ در حالت متوسط همچنین در بهترین حالت یعنی زمانیکه داده‌ها نامرتب وارد شود ارتفاع $h = \log n$ و درنتیجه زمان درج هر داده $O(\log n)$

۲ـ در بدترین وضعیت زمانیکه داده‌ها مرتب وارد شود درخت اریب به راست یا چپ شده و زمان درج هر داده $O(n)$ خواهد بود.

در نتیجه زمان برای درج n گره

- در حالت متوسط و (بهترین حالت) $O(n \log n)$
- در بدترین حالت $O(n^2)$



حذف گره‌ای از BST

برای حذف یک گره دلخواه از درخت BST ابتدا باید عمل جستجو انجام گیرد. که نیاز به زمان O (h) (h عمق درخت) خواهد داشت در این وضعیت 2 حالت پیش می آید:

الف) گره مورد نظر برای حذف از درخت BST وجود ندارد در این حالت هیچ عملی انجام نمی‌شود.

ب) گره مورد نظر برای حذف از درخت BST وجود داشته که در این حالت 3 وضعیت وجود خواهد داشت که به شرح زیر آن را بررسی می کنیم.

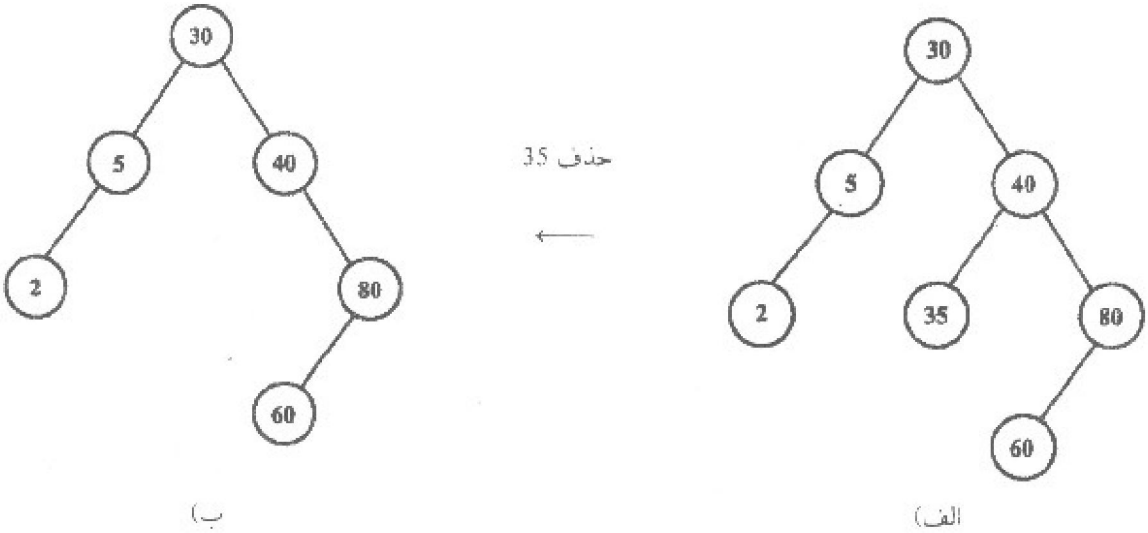
برای حذف گره‌ای مثل x از درخت BST بعد از آن‌که توسط رویه جستجو موقعیت آن بدست آمد،سه حالت را در نظر می‌گیریم:

۱ـ x برگ است.

۲ـ x یک فرزند دارد.

۳ـ x دو فرزند دارد.

حالت اول بسیار ساده است. اشاره‌گر مناسبی از گره والد x را تهی می‌کنیم. اگر x فرزند چپ باشد، اشاره‌گر چپ والد، وگرنه اشاره‌گر راست والد را تهی می کنیم. به عنوان مثال، در BST شکل ۱ ـ الف برای حذف گره برگ حاوی 35، باید فیلد فرزند چپ والد این گره 40 را برابر با تهی قرار دهیم تا شکل ۱ ـ ب به دست آید.



شکل ۱: حذف گرهٔ برگ از BST

حذف گره در حالت دوم که x فقط یک فرزند دارد نیز ساده است. در این حالت باید کاری کنیم که اشاره‌گر مناسبی از گره والد x به فرزند x اشاره نماید. به عنوان مثال، برای حذف گره حاوی 40 از شکل ۲ ـ الف، باید کاری کنیم که اشاره‌گر راست گره والد (30) به فرزند گره 40 (یعنی گره 80) اشاره نماید (شکل ۲ ـ ب) سپس گره 40 را به مخزن حافظه برمی گردانیم.

الف) ب)

شکل ۲: حذف گرهٔ یک فرزندی از BST

در حالت سوم، که در آن x دو فرزند دارد، گره‌ای را پیدا می‌کنیم که در پیمایش inorder بعد یا قبل از x قرار می‌گیرد. این گره را y نام‌گذاری می‌کنیم. دقت داشته باشید که گره y با این ویژگی، عنصر بعد از x فاقد فرزند چپ و عنصر قبل از x فاقد فرزند راست است. ابتدا گره y را با استفاده از حالت‌های 1 یا 2 از درخت حذف می‌کنیم و سپس گره y را به جای گره x در درخت قرار می‌دهیم.

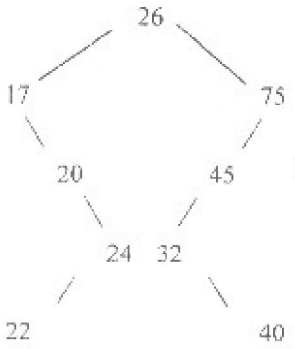
پیدا کردن عنصر بعدی یا قبلی برای هر گره در پیمایش inorder به شرح زیر خواهد بود.

الف) پیدا کردن موقعیت گره بعد از گره p در پیمایش Inorder: (کوچکترین بزرگتر بعد از p)

ابتدا به سمت راست p می‌رویم (بزرگتر از p) سپس برای بدست آوردن کوچکترین بزرگتر از p تا انتها به چپ می‌رویم.

```
T = p ^ . Right;
while (T ^ . Left ≠ null)
T : = T ^ . Left;
```

گره بعد از p (کوچکترین بزرگتر از p) در پیمایش inorder مطمئناً فرزند چپ ندارد چون اگر داشت کوچکتر از این گره بود.



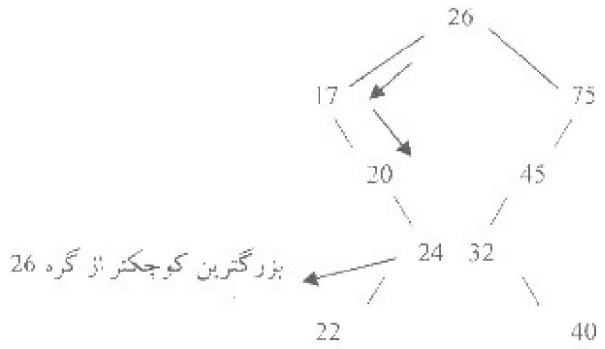
کوچکترین بزرگتر از گره 26

ب) پیدا کردن موقعیت گره قبل از گره p در پیمایش Inorder (بزرگترین کوچکتر قبل از p)

ابتدا به سمت چپ p می‌رویم (کوچکتر از p) سپس برای بدست آوردن بزرگترین کوچکتر از p تا انتها به راست می‌رویم.

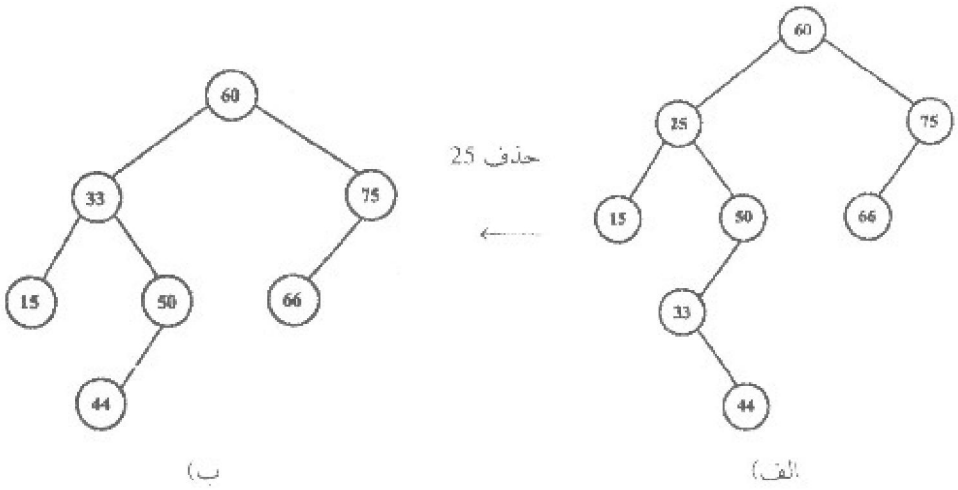
```
T = p ^ . Left;
while (T ^ . Right ≠ null)
T : = T ^ . Right;
```

گره قبل از p (بزرگترین کوچکتر از p) در پیمایش inorder مطمئناً فرزند راست ندارد چون اگر داشت بزرگتر از این گره بود.



پیمایش inorder درخت 75 ، 45 ، 40 ، 32 ، 26 ، 24 ، 22 ، 20 ، 17 در نظر گرفته می‌شود که با توجه به حالت‌های (الف و ب) گره بعد و قبل از 26 بدست می‌آید.

به عنوان مثال، درخت شکل ۳ را در نظر بگیرید. می‌خواهیم گره حاوی 25 را از درخت شکل ۳ ـ الف حذف کنیم تا درخت شکل ۳ ـ ب به دست آید. توجه دارید که اگر این درخت را به روش inorder پیمایش کنید، 33 بعد از 25 قرار می‌گیرد. بنابراین، 33 گره بعدی 25 در پیمایش inorder است. در نتیجه، باید گره 33 را حذف کرده گره 44 را به عنوان فرزند چپ گره 50 که والد گره 33 است وصل کنیم (حالت 2). سپس گره 33 را به جای گره 25 قرار دهیم تا شکل ۳ ـ ب به دست آید.



شکل ۳: حذف گرهٔ دو فرزندی از BST

زمان حذف هر گره از BST

برای حذف هر گره در BST دو عمل به صورت:

۱ـ جستجو به دنبال عنصر حذف شونده با زمان $O(h)$

۲ـ حذف گره مورد نظر $O(1)$

بنابراین زمان حذف گره $O(h)$ خواهد بود.

در نتیجه زمان برای حذف n گره
- در حالت متوسط و (بهترین حالت) $O(n \log n)$
- در بدترین حالت $O(n^2)$

کاربردهای درخت جستجوی دودوئی (BST) (مهم):

۱ـ جستجو به دنبال داده‌ها

۲ـ حذف داده‌های تکراری از لیست‌ها

۳ـ مرتب سازی (sort) لیست‌ها با پیمایش inorder (LVR)

۱ـ جستجو به دنبال داده‌ها

این عمل از ریشه آغاز شده و با مقایسه با هر گره در صورتیکه گره جستجو شونده بزرگتر باشد به سمت راست و در صورتیکه کوچکتر باشد به سمت چپ می‌رویم در نهایت یا گره مورد نظر پیدا می‌شود یا به null می‌رسیم (گره وجود ندارد). زمان جستجو نیز بستگی به ارتفاع درخت (h) دارد که O (h) خواهد بود و در حالت متوسط O (log n) می‌باشند.

دو الگوریتم زیر به صورت بازگشتی و غیربازگشتی عمل جستجو را انجام می‌دهند.

بازگشتی

```
function   find (t : tree pointer ; x : integer ) : boolean;
var
    flag : boolean;
begin
    flag : = False;
    if   (t < > nil)   then   begin
       if   t^.data = x   then   flag : = True
    else   if   x < t^.data   then
       flag : = find (t^. Lchild , x)
       else   if   x > t^.data   then
          flag : = find (t^. Rchild , x);
end;
          find : = flag;
end;
```

غیربازگشتی

```
function   find (t : tree pointer ; x : integer ) : boolean;
var
p , q : tree pointer ;
flag : boolean;
    begin
       p : = t;  flag : = False;
while   (p < > nil)   and   (not flag)   do
       begin
          if (p^. data = x)   then   flag : = True
          else   if (x < p^.data)   then
             begin
             q : = p;
             p : = p^. Lchild;
             end;
          else   if (x > p^. data)   then
             begin
             q : = p;
             p : = p^. Rchild;
             end;
       end;
       find : = flag;
end;
```

۲ـ حذف داده‌های تکراری لیست‌ها

برای آن‌که بخواهیم در لیست داده‌های $a_1, a_2, \ldots, a_n$ عناصر تکراری را هنگام ورود داده‌ها حذف کنیم روش‌های مختلفی وجود دارد.

الگوریتم عمومی: در یک الگوریتم ساده هنگام ورود داده a_k با $(k-1)$ مقایسه با داده‌های $a_1, a_2, \ldots, a_{k-1}$ تکراری بودن داده a_k مشخص می‌شود. در نتیجه تعداد مقایسات برای $a_1, a_2, \ldots, a_n$ از رابطه زیر بدست می‌آید.

$$a_1\ a_2\ \ldots\ a_n$$

تعداد مقایسات

$$0 + 1 + \ldots + (n-1) = \frac{n(n-1)}{2} = O(n^2)$$

الگوریتم درخت BST

با استفاده از داده‌های $a_1, a_2, \ldots, a_n$ یک درخت BST ایجاد می‌کنیم در این حالت هنگام درج a_k اگر در درخت تکراری باشد از لیست حذف می‌شود. در این وضعیت به طور متوسط زمان درج هر داده $O(\log n)$ و برای درج n داده $O(n \log n)$ نیاز است که زمان آن از الگوریتم عمومی کمتر خواهد بود.

بدترین وضعیت زمانی خواهد بود که داده‌ها به صورت مرتب وارد شود در این حالت مرتبه زمانی $O(n^2)$ خواهد بود و تعداد مقایسات در صورتیکه هیچ داده تکراری نداشته باشیم $\frac{n(n-1)}{2}$ خواهد بود.

مثال: مطلوبست تعداد مقایسات برای حذف داده‌های تکراری آرایه زیر به دو روش الگوریتم عمومی و درخت BST:

10 , 20 , 40 , 20 , 50 , 40

الگوریتم عمومی

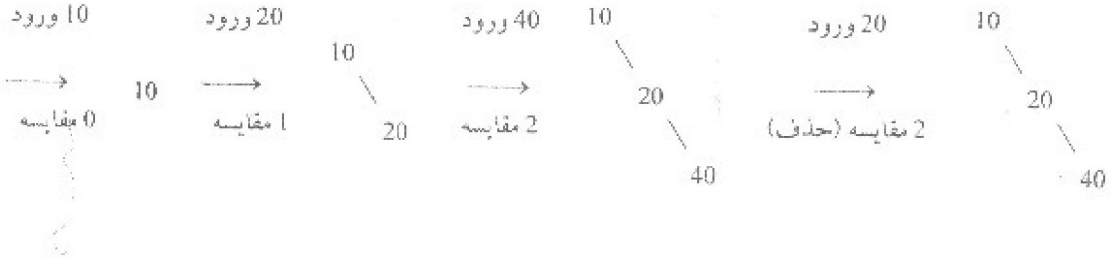
	10	20	40	20	50	40	
مقایسات	0	1	2	3	4	5	= 15

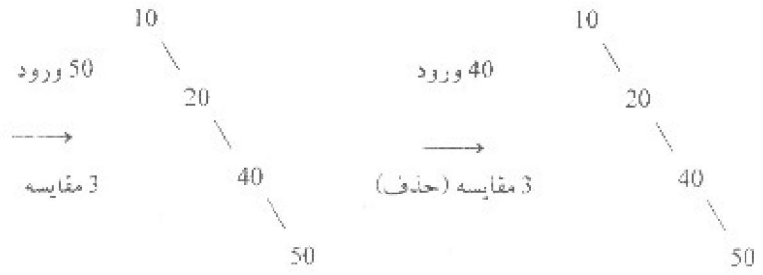
البته باتوجه به آن‌که $n = 6$ است در نتیجه از رابطه $\frac{n(n-1)}{2}$ نیز تعداد مقایسات در این روش بدست می‌آید.

$$\frac{6(6-1)}{2} = 15$$

الگوریتم درخت BST

درخت BST را از داده‌های فوق تشکیل می‌دهیم.





مجموع مقایسات = 0 + 1 + 2 + 2 + 3 + 3 = 11

که تعداد مقایسات کمتر از الگوریتم عمومی است.

۳ـ مرتب سازی داده‌ها

در صورتیکه آرایه n تائی را داشته باشیم می‌توانیم با
- الف) درج آرایه در درخت BST تهی
- ب) پیمایش inorder (LVR) درخت BST حاصل از مرحله (الف)

آرایه را sort کنیم.

زمان در حالت متوسط و بهترین وضعیت $O(n \log n)$

زمان در بدترین وضعیت (درخت اریب) $O(n^2)$

پیاده سازی BST

باتوجه به درج عناصر به صورت صعودی و نزولی و بوجود آمدن درخت BST به صورت اریب استفاده از نمایش BST به صورت آرایه اتلاف حافظه زیادی خواهد داشت در این حالت بهتر است از نمایش پیوندی استفاده کنیم.

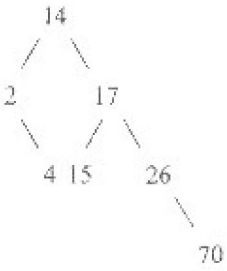
پیمایش Level order (سطح ترتیب) یک درخت BST:

در صورتیکه پیمایش سطح ترتیب یک درخت BST را داشته باشیم به راحتی می‌توانیم به درخت یکتائی BST برسیم.

مثال: فرض کنیم پیمایش سطح ترتیب یک درخت BST به صورت:

14 , 2 , 17 , 4 , 15 , 26 , 70

باشد به راحتی می‌توان درخت آن را بدست آورد.



ابتدا 14 را در ریشه قرار می‌دهیم سپس گره‌های بعدی را از سطح دوم به بعد از چپ به راست با ریشه هر زیر درخت مقایسه و در سمت چپ یا راست ریشه زیر درخت قرار می‌دهیم.

مثال: در صورتیکه اعداد 1 تا 1000 در یک درخت BST درج شده باشند کدام وضعیت‌های جستجو نمی‌تواند درست باشد؟

۱) 70 , 35 , 25 , 15 , 10

۲) 176 , 195 , 180 , 240 , 125

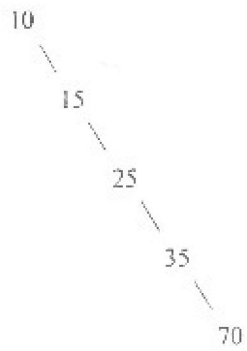
۳) 65 , 54 , 46 , 74 , 22

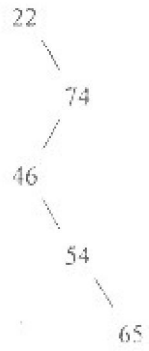
۴) 144 , 140 , 132 , 145 , 170

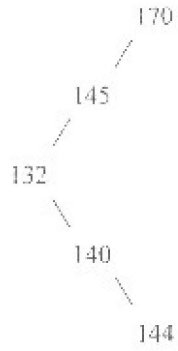
گزینه ۲ صحیح می‌باشد.

باتوجه به وضعیت سمت راست 180 باید از 180 بزرگتر باشد ولی 176 از 180 کوچکتر است و در عین حال سمت راست قرار دارد که این حالت صحیح نیست.

گزینه‌های دیگر همه حالت BST بودن خود را حفظ کرده‌اند.

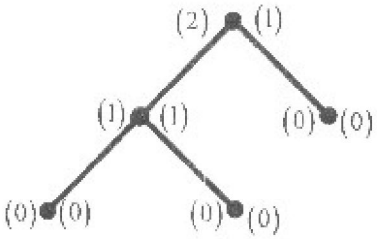




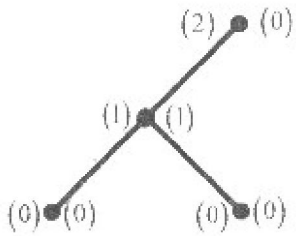


درخت با ارتفاع متوازن

درختی که در آن اختلاف ارتفاع در زیر درخت چپ و راست هر گره حداکثر 1 باشد، درخت با ارتفاع متوازن نامیده می‌شود.



الف) درخت با ارتفاع متوازن است.



ب) درخت با ارتفاع متوازن نیست.

در شکل ب اختلاف دو زیر ارتفاع چپ و راست گره ریشه، 2 است و در نتیجه درخت با ارتفاع متوازن نیست.

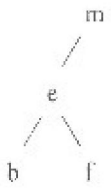
یادآوری:

درخت متوازن به درختی گفته می‌شود که اختلاف سطح برگ‌های آن حداکثر 1 باشد.

هر درخت با ارتفاع متوازن حتماً یک درخت متوازن است اما عکس این مطلب همیشه درست نیست یعنی هر درخت متوازن لزوماً درخت با ارتفاع متوازن نیست.

مثال ۱:

در شکل روبرو

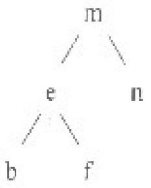


الف) اختلاف سطح برگ‌های درخت (b , f) صفر بوده و در نتیجه درخت متوازن است.

ب) ارتفاع زیر درخت چپ m، 2 و ارتفاع زیر درخت راست m، 0 و در نتیجه درخت با ارتفاع متوازن نیست.

مثال ۲:

در شکل روبرو:



الف) اختلاف سطح برگ‌های n , f , h حداکثر 1 است بنابراین درخت متوازن است.

ب) اختلاف ارتفاع زیر درخت چپ و راست هر گره حداکثر 1 است. بنابراین درخت با ارتفاع متوازن است.

نکته مهم:

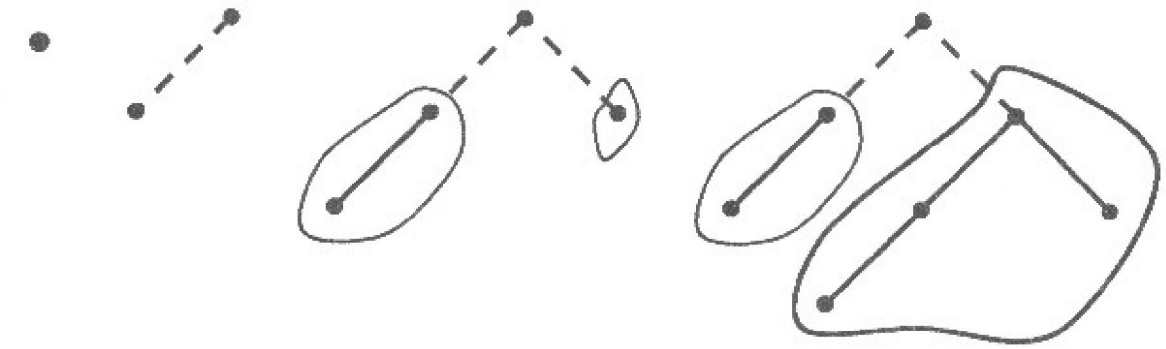
اگر AVL (h) حداقل تعداد گره مورد نیاز برای ساختن یک درخت با ارتفاع متوازن h باشد. رابطه زیر همیشه برقرار است.

$$AVL(h) = AVL(h-1) + AVL(h-2) + 1$$

مثال: در صورتیکه حداقل گره‌های لازم برای ساختن 2 درخت با ارتفاع متوازن 3 و 4، به ترتیب 4 و 7 باشد حداقل گره‌های لازم برای ساختن درخت با ارتفاع متوازن 5 کدام است؟

$$\begin{matrix} AVL(3) = 4 \\ AVL(4) = 7 \end{matrix} \Rightarrow \begin{matrix} AVL(5) = AVL(4) + AVL(3) + 1 \\ = 7 + 4 + 1 = 12 \end{matrix}$$

شکل‌های درخت با ارتفاع متوازن (حداقل گره)



درخت AVL

هر درخت AVL یک درخت جستجوی دودویی (BST) بوده که ارتفاع متوازن دارد.

درخت با ارتفاع متوازن درختی است که حداکثر اختلاف دو زیر درخت چپ و راست هر گره 1 باشد.

نکات مهم:

۱) حداکثر ارتفاع درخت AVL با n گره $\lfloor \log_2 n \rfloor + 1$ (در صورتیکه ریشه در سطح 1) و یا $\lfloor \log_2 n \rfloor$ (در صورتیکه ریشه در سطح 0) در نظر گرفته شود خواهد بود.

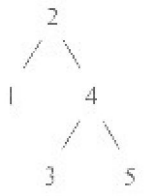
۲) زمان عمل درج و حذف و جستجو در AVL از BST بهتر بوده و زمان آن $O(\log_2 n)$ خواهد بود.

یادآوری:

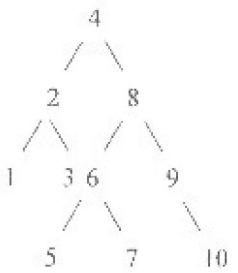
زمان درج و حذف و جستجو در AVL از BST، $O(h)$ بود که h عمق درخت بوده در بدترین حالت زمانیکه BST یک درخت با اریب با n گره بود زمان حذف و درج $O(n)$ بود.

مثال: درخت‌های زیر نمونه‌هایی از درخت AVL هستند.

الف)



ب)



دیده می‌شود که درخت‌های (الف) و (ب) در عین حال که BST هستند دارای ارتفاع متوازن نیز هستند.

نتیجه‌گیری:

بین درخت‌های BST، AVL، درخت AVL برای عملیات جستجو، درج و حذف بهتر است. چون ارتفاع آن متوازن بوده و در نتیجه هز عمل درج یا حذف یا جستجو حداکثر با زمان $O(\log_2 n)$ انجام می‌شود.

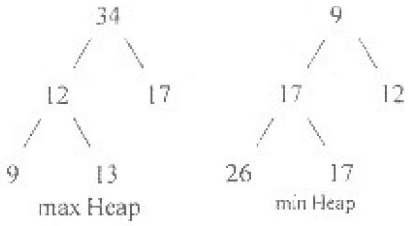
■ **درخت Heap (نیمه مرتب ـ هرم)**

maxtree: درختی که مقدار کلید هر گره آن بزرگتر یا مساوی فرزندانش باشد.

Mintree: درختی که مقدار کلید هر گره آن کوچکتر یا مساوی فرزندانش باشد.

maxHeap: درخت دودوئی کاملی که maxtree نیز باشد.

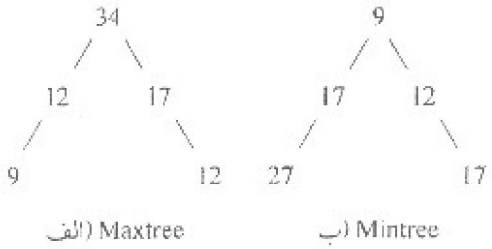
minHeap: درخت دودوئی کاملی که mintree نیز باشد.



مثال:

درخت (الف) maxtree است اما چون کامل نیست maxHeap نیست.

درخت (ب) mintree است اما چون، کامل نیست minHeap نیست.

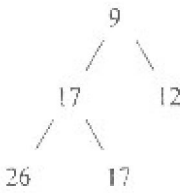


محاسبهٔ گره ماکزیمم و می‌نیمم در Heap

درخت minHeap

۱ـ عنصر می‌نیمم در ریشه وجود دارد که بدون مقایسه بدست می‌آید.

۲ـ عنصر ماکزیمم در برگ‌ها وجود دارد و چون درخت کامل است تعداد برگ‌ها ($\left\lfloor \frac{n+1}{2} \right\rfloor$) و تعداد مقایسات برای محاسبه عنصر ماکزیمم $\left\lfloor \frac{n+1}{2} \right\rfloor - 1$ خواهد بود.



در این درخت minHeap برای بدست آوردن عنصر ماکزیمم با 2 مقایسه بین برگ‌ها به نتیجه می‌رسیم.

تعداد گره‌ها $= n = 5$

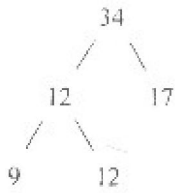
تعداد برگ‌ها $= \frac{n+1}{2} = 3$

تعداد برگ‌ها − 1 = 2 = تعداد مقایسات برای محاسبه ماکزیمم

درخت max Heap

۱ـ عنصر ماکزیمم در ریشه وجود دارد که بدون مقایسه بدست می‌آید.

۲ـ عنصر می‌نیمم در برگ‌ها وجود دارد و چون درخت کامل است تعداد برگ‌ها $\left\lfloor \frac{n+1}{2} \right\rfloor$ و تعداد مقایسات برای محاسبه می‌نیمم $\left\lfloor \frac{n+1}{2} \right\rfloor - 1$ خواهد بود.



در این درخت max Heap برای بدست آوردن عنصر می‌نیمم با 2 مقایسه بین برگ‌ها به نتیجه می‌رسیم.

تعداد گره‌ها $= n = 5$

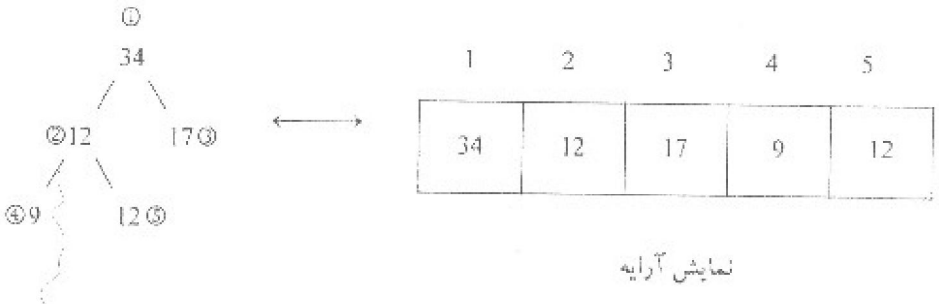
تعداد برگ‌ها $= \frac{n+1}{2} = 3$

تعداد برگ‌ها − 1 = 2 = تعداد مقایسات برای محاسبه می‌نیمم

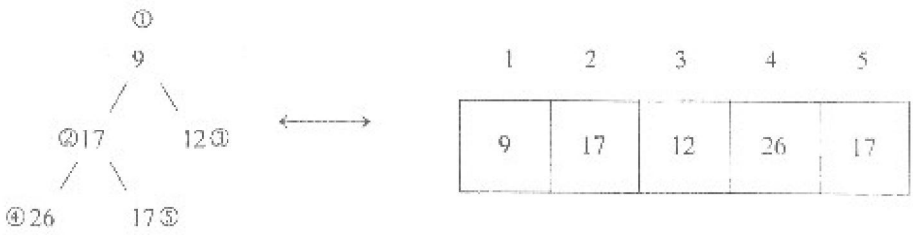
نمایش Heap در حافظه

۱ـ برای نمایش maxtree و mintree چون لزوماً درختان کاملی نیستند از نمایش پیوندی استفاده می‌شود.چون در صورت استفاده از آرایه اتلاف حافظه خواهند داشت.

۲ـ برای نمایش maxHeap و minHeap چون درختان کاملی هستند از آرایه‌ها استفاده کنیم که در این حالت اتلاف حافظه‌ای نخواهیم داشت.



نمایش آرایه



نمایش آرایه

یادآوری:

در شروع بحث درخت‌ها پیاده‌سازی درخت به صورت آرایه نشان داده شد. که شماره‌گذاری از 1 با گره ریشه آغاز می‌شد و تمام گره‌های دیگر از سطح بعدی از چپ به راست شماره‌گذاری می‌شدند.

درج و حذف در درخت Heap

درج گره در Heap

برای درج هر گره در Heap:

۱ـ گره جدید را در آخرین سطح در اولین موقعیت از چپ به راست قرار می‌دهیم.

۲ـ گره درج شده

الف) در maxHeap: تا جائیکه از والدش (parent) بزرگتر باشد باید با آن جابجا شود (حداکثر تا ریشه)

ب) در minHeap: تا جائیکه از والدش (parent) کوچکتر باشد باید با آن جابجا شود (حداکثر تا ریشه)

زمان درج گره در Heap: $O(\log_2 n)$

عمل درج گره با توجه به آن‌که گره درج شونده در آخرین سطح ممکن است با تمام والدهای خود تا ریشه جابجا شود زمان $O(h)$ خواهد داشت (h ارتفاع درخت Heap و چون این درخت کامل است $h = \log_2 n$) در نتیجه زمان درج گره در Heap $O(\log_2 n)$ خواهد بود.

الگوریتم درج: باتوجه به آن‌که برای نگهداری Heap از آرایه استفاده می‌کنیم در الگوریتم زیر در آرایه $H[1..n]$ که n گره Heap وجود دارد عمل درج عنصر x در maxHeap انجام می‌شود.

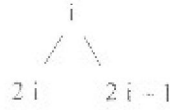
```
procedure  insertHeap (H : array Heap; n : integer; x : integer);
var
i , j : integer;
    begin
       i := n + 1;  j := i  div  2;
       while  (j > 0)  and  (H[j] < x)  do
          begin
             H[i] := H[j];
             i := j;
             j := i  div  2;
          end;
       H[i] := x;
    end;
```

در الگوریتم بالا
- i: گره جدید x را نشان می‌دهد.
- j: گره والد x را نشان می‌دهد.

تا زمانیکه $0 < j$ (به ریشه نرسیده باشیم) و $x > H[j]$ (x از والدش بزرگتر باشد) حلقه while تکرار شده و گره x (i) با والدش (j) جابجا می‌شود. در هر مرحله برای آن‌که به گره والد x (i) برسیم از جمله j := i div 2 استفاده می‌کنیم.

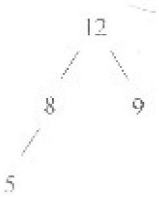
یادآوری:

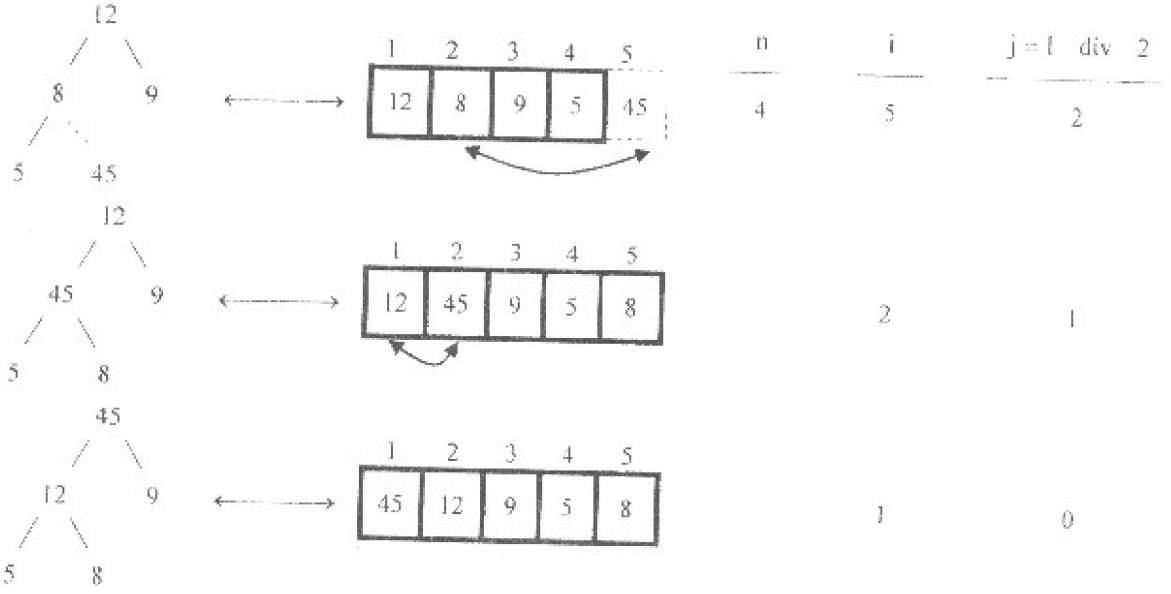
در بحث نمایش درخت به صورت آرایه بیان شد که



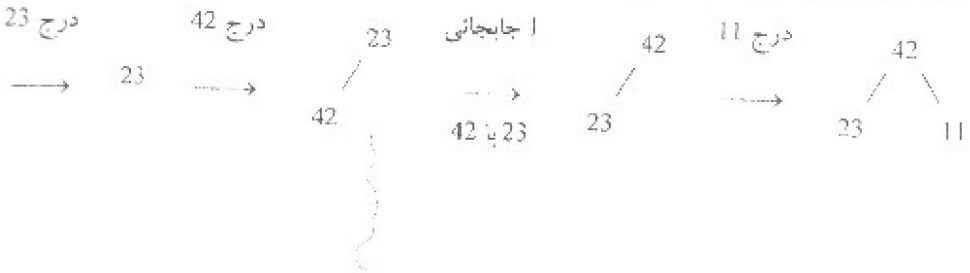
و همچنین والد گره i در برابر $\left\lfloor \frac{i}{2} \right\rfloor$ قرار دارد.

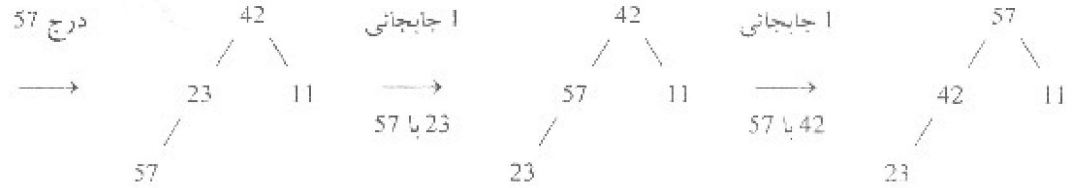
مثال: درج گره 45 در درخت maxHeap روبرو:

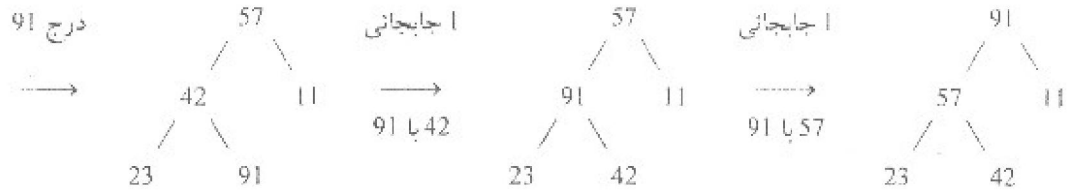


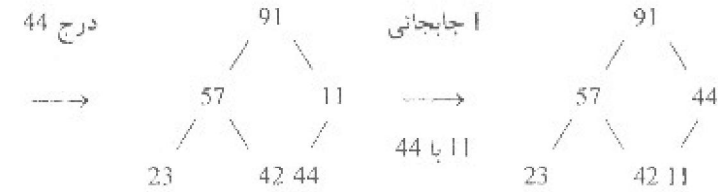


درج داده‌های 44 ، 91 ، 57 ، 11 ، 42 ، 23 در درخت max Heap تهی:









گره 23: درخت تهی بوده و گره 23 به عنوان اولین گره درج شود.

گره 42: با 1 جابجائی با گره 23 در موقعیت مناسب درخت max Heap قرار می‌گیرد.

گره 11: بدون جابجائی در موقعیت خود باقی می‌ماند.

گره 57: با 2 جابجائی با گره‌های 23 ، 42 در موقعیت مناسب درخت max Heap قرار می‌گیرد.

گره 91: با 2 جابجائی با گره‌های 42 ، 57 در موقعیت مناسب درخت max Heap قرار می‌گیرد.

گره 44: با 1 جابجائی با گره 11 در موقعیت مناسب درخت max Heap قرار می‌گیرد.

نکته: هر گاه درخت Heap به صورت آرایه مطرح شد، برای نمایش درخت عنصر اول را ریشه در نظر گرفته و عناصر بعدی گره‌های سطوح را از چپ به راست نشان می‌دهند.

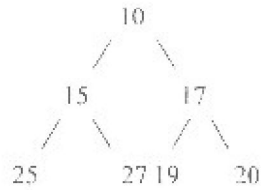
مثال: درخت Heap زیر را در نظر بگیرید.

1	2	3	4	5	6	7
10	15	17	25	27	19	20

وضعیت درخت Heap چگونه است؟

الف) عنصر اول = ریشه درخت (10) و از همه کوچکتر در نتیجه درخت minHeap است.

ب) نمایش درخت:



حذف گره از Heap

۱ـ ریشه حذف شده و آن را در خروجی می‌نویسیم.

۲ـ سمت راست‌ترین گره آخرین سطح جایگزین ریشه می‌کنیم.

برای حذف هر گره از Heap:

۳ـ گره جایگزین شده

الف) در maxHeap: تا جائیکه از فرزندانش کوچکتر است باید بزرگترین فرزند جابجا شود.

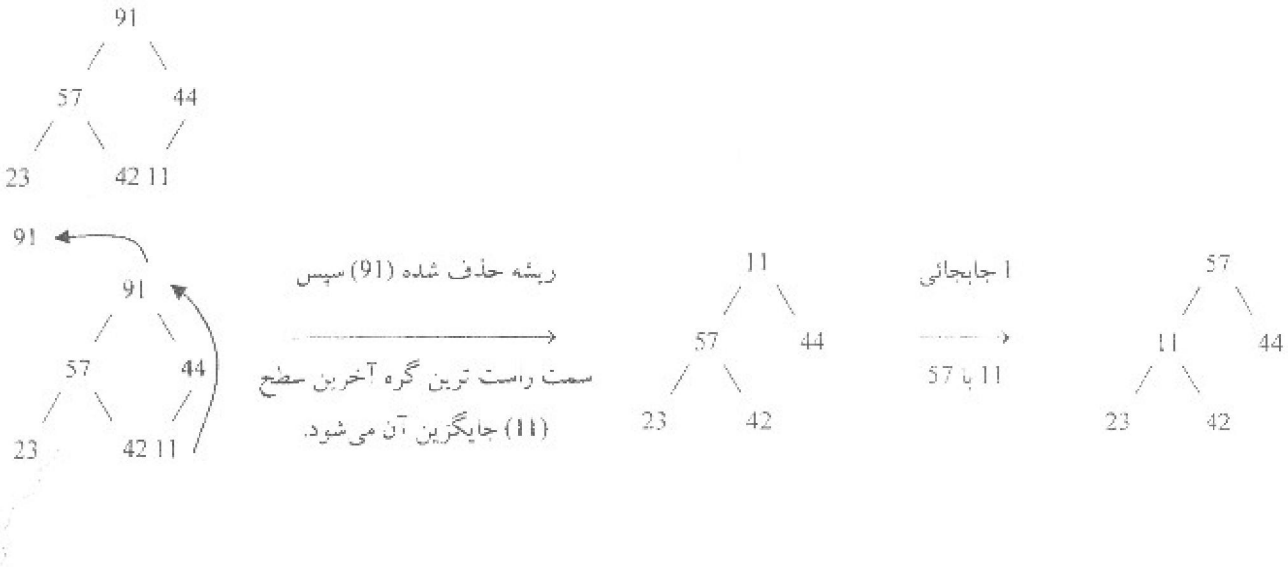
ب) در minHeap: تا جائیکه از فرزندانش بزرگتر است باید کوچکترین فرزند جابجا شود.

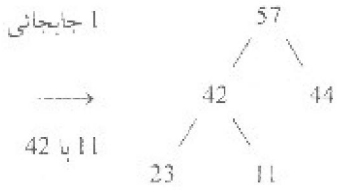
زمان حذف گره از Heap $O(\log n)$

عمل حذف گره باتوجه به آن‌که گره جایگزین در ریشه حداکثر ممکن است تا آخرین سطح با گره‌های دیگر جابجا شود زمان $O(h)$ خواهد داشت (h ارتفاع درخت Heap بوده که چون این درخت کامل است $h = \log_2 n$) در نتیجه زمان حذف گره از heap: $O(\log n)$ خواهد بود.

حذف گره‌های درخت max Heap درخت زیر:

الف) حذف اول (91)



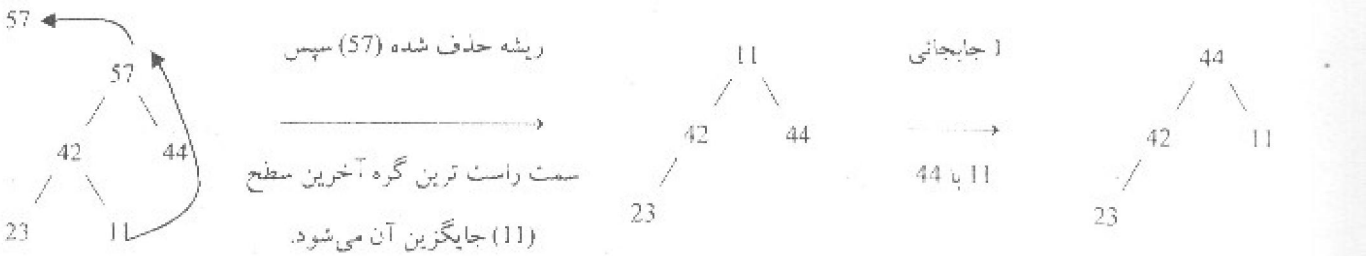


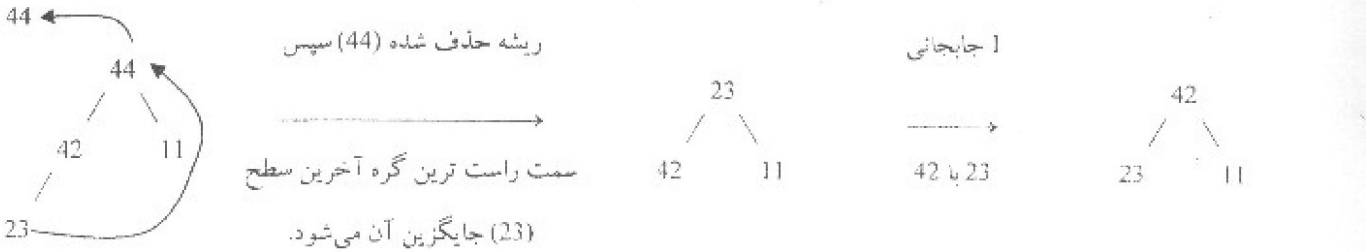
گره 91 با 2 جابجائی از درخت حذف می‌شود.

بعد از قرار گرفتن گره 11 به جای 91 برای آن که شرایط max Heap بودن درخت حفظ شود.

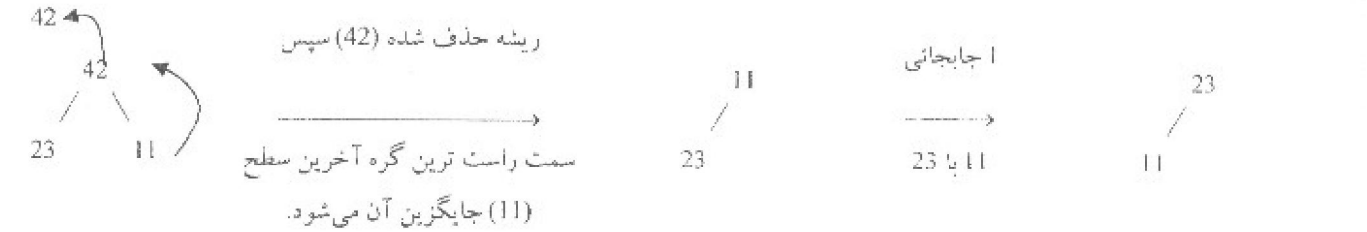
از بین فرزندان گره 11 یعنی 44 ، 57 فرزند بزرگتر یعنی (57) با گره 11 جابجا می‌شود سپس گره 11 والد گره‌های 42 ، 23 می‌شود که در بین این فرزندان 42 با 11 جابجا می‌شود تا max Heap بودن درخت حفظ شود.

ب) حذف دوم (57)

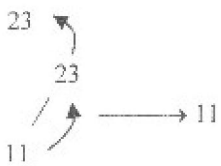




ج) حذف سوم (42)



د) حذف چهارم (23)



هـ) حذف پنجم (11)

کاربرد درخت Heap

۱ـ مرتب سازی آرایه‌ها
۲ـ پیاده سازی صف اولویت (priority Queue)
۳ـ ادغام لیست‌های مرتب

۱ـ مرتب سازی Heap

برای مرتب سازی یک آرایه n تائی به روش Heap به ترتیب باید 2 عمل انجام شود.

الف) درج تمام داده‌ها در یک درخت Heap تهی (maxHeap: مرتب سازی نزولی ، minHeap: مرتب سازی صعودی)

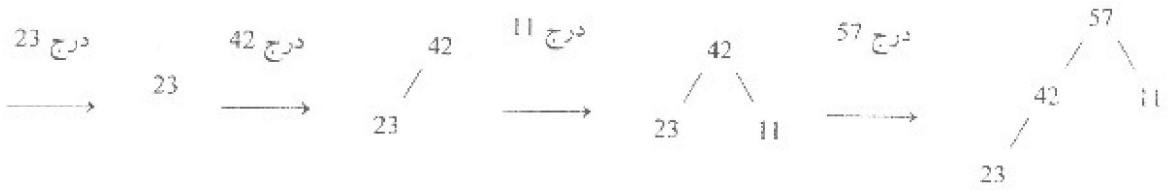
ب) حذف تمام داده‌ها از ریشه درخت Heap

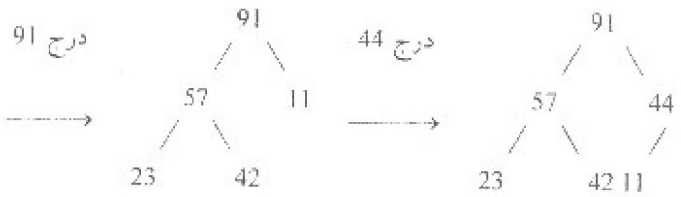
■ زمان مرتب سازی Heap

از آنجائیکه زمان درج یا حذف در Heap O (log n) می‌باشد، در نتیجه برای مرتب سازی یک آرایه n تائی زمان مورد نظر O (n log n) خواهد بود.

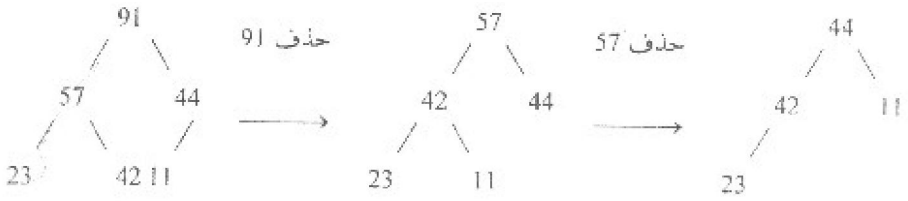
مثال: مرتب سازی داده‌های 44 ، 91 ، 57 ، 11 ، 42 ، 23 به سیولهٔ درخت Heap:

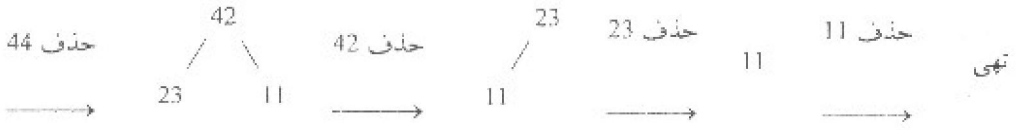
قسمت اول: درج داده‌ها در Heap





قسمت دوم: حذف داده‌ها از Heap





۲ـ صف اولویت

تعریف: صف اولویت به مجموعه‌ای از عناصر گفته می‌شود که در آن به هر عنصر یک اولویت داده می‌شود. و ترتیب حذف و پردازش داده‌ها بر طبق دو قاعده زیر است:

۱ـ عنصری که دارای اولویت بیشتر است قبل از بقیه عناصر حذف شده و پردازش می‌شود.

۲ـ دو عنصری که دارای اولویت یکسان هستند باتوجه به ترتیبی که به صف وارد شده‌اند پردازش می‌شود.

یادآوری:

در هر صف اولویت دو عمل حذف و درج عنصر انجام می‌شود.

پیاده سازی صف اولویت: به طور کلی سه روش برای پیاده‌سازی صف اولویت وجود دارد:

۱ـ آرایه‌ها
۲ـ لیست پیوندی
۳ـ درخت Heap

که درخت Heap از بقیه ساختارها بهتر است چون زمان کمتری برای درج و حذف نیاز دارد.

زمان‌های حذف و درج در این ساختارها به شرح زیر هستند:

ساختار	زمان درج	زمان حذف
آرایه نامرتب	$O(1)$	$O(n)$
آرایه مرتب	$O(n)$	$O(1)$
لیست نامرتب	$O(1)$	$O(n)$
لیست مرتب	$O(n)$	$O(1)$
✓ درخت Heap	$O(\log_2 n)$	$O(\log_2 n)$

در آرایه‌ها و لیست‌ها نامرتب:

۱ـ **درج:** عنصر به راحتی در ابتدای لیست یا در انتهای آرایه درج می‌شود. $O(1)$

۲ـ **حذف:** برای پیدا کردن بیشتر اولویت باید کل عناصر جستجو شوند و در آرایه‌ها نیاز به شیفت داریم. $O(n)$

در آرایه‌ها و لیست‌های مرتب:

۱ـ **درج:** برای درج عنصر باید باتوجه به اولویت عنصر موقعیت آن را بین داده‌ها پیدا کنیم و در آرایه نیز نیاز به شیفت داریم $O(n)$

۲ـ **حذف:** عنصر با اولویت بالا را در آرایه‌ها از انتها و در لیست‌ها از ابتدا با زمان $O(1)$ حذف می‌کنیم.

در درخت Heap

۱ـ **درج:** با زمان $O(\log_2 n)$ انجام می‌شود.

۲ـ **حذف:** با زمان $O(\log_2 n)$ انجام می‌شود.

دیده می‌شود که ساختار Heap در دو عمل حذف و درج زمان کمتری نسبت به دو ساختار آرایه و لیست پیوندی نیاز دارد.

۱ـ ادغام لیست‌ها

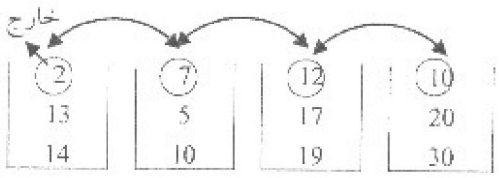
فرض کنید K آرایه مرتب (صعودی) داریم که تعداد کل خانه‌های K آرایه n می‌باشد. حال می‌خواهیم این K آرایه را ادغام کرده و در یک آرایه به صورت مرتب نگهداری کنیم.

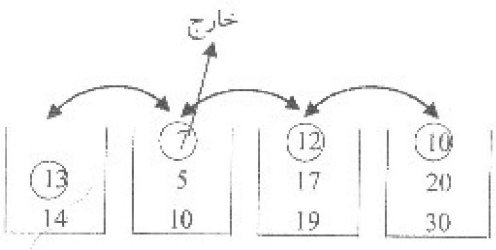
الف) روش عمومی:

۱ـ عضو اول همه آرایه‌ها را با هم مقایسه کرده و با $K-1$ مقایسه، عضو می‌نیمم را خارج می‌کنیم.

۲ـ در این حالت آرایه‌ای که عضو می‌نیمم از آن خارج شده عناصرش را یک واحد به سمت بالا حرکت می‌دهد (رانش ـ RUN).

۳ـ عمل ۱ را دوباره انجام می‌دهیم (برای $n-1$ عضو دیگر)





زمان اجرا: برای خروج هر عضو $(k-1)$ مقایسه انجام می‌شود در نتیجه برای n عضو آرایه $n(k-1)$ مقایسه انجام می‌شود و زمان ادغام $O(nk)$ می‌باشد.

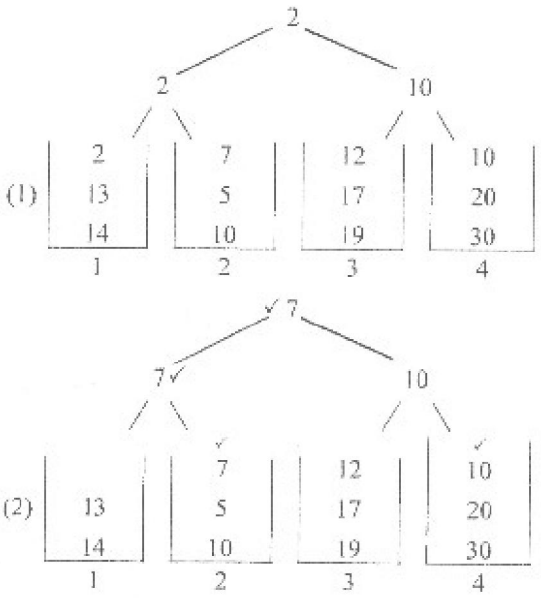
ب) روش درخت انتخابی

تعریف: هر درخت انتخابی درخت دودوئی کاملی است که در آن هر گره درخت از فرزندانش کوچکتر یا مساوی است و عنصر می‌نیمم در ریشه درخت است. (minHeap)

ادغام آرایه‌ها با درخت انتخابی

برای ادغام آرایه‌ها عنصر اول همه آرایه‌ها را به عنوان برگ‌های درخت انتخابی در نظر گرفته و درخت انتخابی را درست می‌کنیم (برنده (کوچکتر) ـ بازنده (بزرگتر))، به این صورت که بین هر دو برگ کنار هم گره‌ای را به عنوان والد انتخاب می‌کنیم (برنده) که کوچکتر باشد. این عمل را آن‌قدر تکرار می‌کنیم تا این که به ریشه برسیم و کوچکترین عنصر در ریشه قرار گیرد.

با مقایسه 2 با 7، 2 برنده و با مقایسه 10 با 12 ،10 برنده در نهایت با مقایسه 10 با 2، 2 برنده می‌شود و در ریشه قرار می‌گیرد.



روش کار:

۱ـ ریشه درخت انتخابی را حذف کرده و در آرایه نهائی قرار می‌دهیم (عنصر می‌نیمم) (گره 2 در شکل 1).

۲ـ برای تجدید ساختار در آرایه‌ای که عنصر می‌نیمم در مرحله اول از آن انتخاب شده یک واحد به بالا حرکت کرده (در آرایه اول در این مثال) و با مقایسات مناسب (علامت ✓) دوباره درخت انتخابی را ایجاد می‌کنیم (شکل ۲).

۳ـ عملیات 1 ، 2 را تا زمانیکه کلیه عناصر آرایه به صورت مرتب خارج شوند تکرار می‌کنیم.

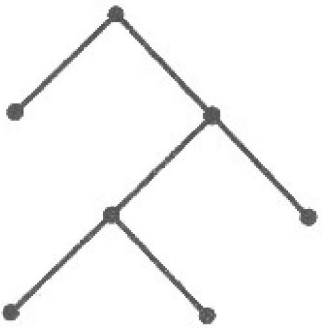
زمان اجرا: برای هر تجدید ساختار به اندازهٔ ارتفاع درخت زمان نیاز داریم $O(\log_2 k)$ و برای کل عناصر آرایه (n عنصر) زمان نهائی $O(n \log k)$ می‌باشد.

نتیجه‌گیری:

دیده می‌شود که روش انتخابی با استفاده از درخت minHeap از نظر زمانی مقرون به صرفه‌تر از روش عمومی می‌باشد چون $O(n \log_2 k) < O(n k)$ می‌باشد.

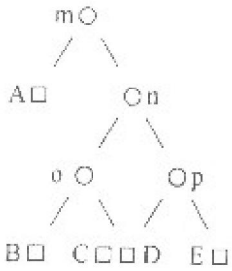
درخت توسعه یافته (دودوئی محض):

تعریف: به درختی دودوئی که در آن هر گره یا 2 فرزند داشته باشد یا 0 فرزند داشته باشد.



گره‌های خارجی و داخلی

در درخت دودوئی محض (توسعه یافته) به گره‌های 2 فرزندی گره‌های داخلی و به گره‌های 0 فرزندی گره خارجی می‌گوئیم.



O = گره داخلی

□ = گره خارجی

نکته:

اگر { N_E = تعداد گره‌های خارجی ; N_I = تعداد گره‌های داخلی } آن‌گاه $\boxed{N_E = N_I + 1}$ به عبارت بهتر: 1 + تعداد گره‌های 2 فرزندی = تعداد برگ‌ها

درمثالا بالا $N_E = 5$ و $N_I = 4$ و رابطه مورد نظر برقرار است.

طول مسیر خارجی (L_E) ـ طول مسیر داخلی (L_I)

بنابو تعریف:

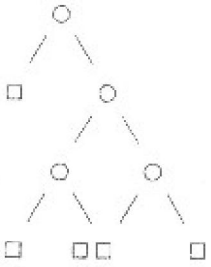
L_E = مجموع تمام طول مسیرها که طول هر مسیر برابر تعداد یال‌ها در مسیر ریشه تا یک گره خارجی می‌باشد.

L_I = مجموع تمام طول مسیرها که طول هر مسیر برابر تعداد یال‌ها در مسیر ریشه تا یک گره داخلی می‌باشد.

در درخت بالا

$$L_I = \underbrace{0}_{m} + \underbrace{1}_{n} + \underbrace{2}_{o} + \underbrace{2}_{p} = 5$$

$$L_E = \underbrace{1}_{A} + \underbrace{3}_{B} + \underbrace{3}_{C} + \underbrace{3}_{D} + \underbrace{3}_{E} = 13$$



نکته مهم:

در صورتیکه n تعداد گره داخلی باشد آن‌گاه $L_E = L_I + 2n$

در درخت بالا $n = 4$ گره داخلی وجود دارد. در نتیجه

$$L_E = L_I + 2n = 5 + 2 \times 4 = 13$$

طول مسیر وزن در گره‌های خارجی

اگر به هر گره خارجی یک وزن بدهیم (w_i)، طول مسیر وزن گره‌های خارجی به صورت زیر قابل محاسبه است:

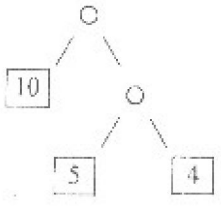
در صورتیکه m گره خارجی داشته باشیم:

$$p = W_1 L_1 + W_2 L_2 + \ldots + W_m L_m$$

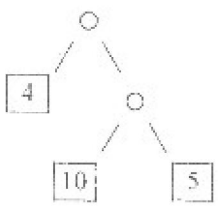
L_i = طول مسیر یک گره خارجی

W_i = وزن یک گره خارجی

نکته مهم: پیاده‌سازی مختلف گره‌های خارجی در قالب درخت دودوئی محض وزن‌های متفاوتی را از نظر طول مسیر وزن تولید می‌کند.



$p_1 = 1 \times 10 + 2 \times 5 + 3 \times 4 = 28$



$p_2 = 1 \times 4 + 2 \times 10 + 2 \times 5 = 34$

در دو درخت بالا با این‌که سه گره خارجی با وزن 4 ، 5 ، 10 وجود دارد اما طول مسیر وزن‌ها متفاوت است. دیده می‌شود که بهتر است ابتدا گره‌های خارجی با وزن کمتر را برای پایین‌ترین سطح انتخاب کنیم تا در مجموع وزن کمتری داشته باشیم.

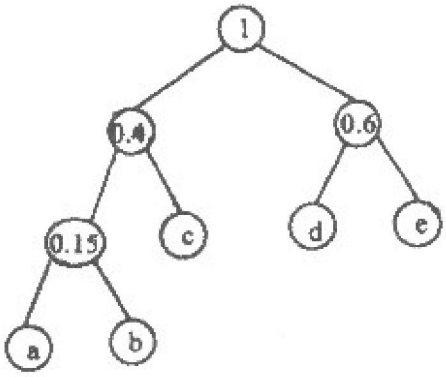
الگوریتم هافمن (تعیین درخت با حداقل طول مسیر وزن)

در این الگوریتم برای ایجاد درخت با حداقل طول مسیر وزن در هر مرحله دو درختی را که ریشه می‌نیمم دارند را انتخاب می‌کنیم (در شروع کار از گره‌های خارجی استفاده می‌کنیم.)

مثال: حروف a , b , c , d , e با جدول فراوانی زیر داده شده است:

حروف	a	b	c	d	E
فراوانی	0.05	0.1	0.25	0.28	0.32

درخت هافمن وابسته به این حروف به قرار زیر است:



مرحله اول: a , b می‌نیمم بوده و انتخاب می‌شود و ریشه با مقدار 0.15 ایجاد می‌کنند.

مرحله دوم: در بین {ریشه (a , b) , [0.15] c , [0.25] d , [0.28] e , [0.32]} ریشه (a , b) و c را که می‌نیمم هستند انتخاب می‌کنیم که ریشه ادغام آن‌ها 0.4 می‌شود.

مرحله سوم: در بین {ریشه ((a , b) , c) , [0.4] d , [0.28] e , [0.32]} c , d را که می‌نیمم هستند انتخاب می‌کنیم ریشه ادغام آن‌ها 0.6 می‌شود.

مرحله چهارم: به ریشه درخت می‌رسیم.

کاربرد الگوریتم هافمن در کدگذاری

یکی از کاربردهای مهم درخت می‌نیمم ایجاد شده به روش هافمن فشرده سازی داده‌ها با توجه تکرار آن‌ها می‌باشد، در این حالت برای هر داده کدی انتخاب می‌شود.

فرض کنید می خواهیم n عنصر اطلاعاتی $A_1 , A_2 , \ldots , A_n$ را به وسیله رشته‌هائی از بیت‌های 0 یا 1 کدگذاری کنیم، در این حالت برای هر عنصر 0 یک کد در نظر گرفته می‌شود. که این کد باتوجه به قواعدی ایجاد می‌شود.

نکته مهم:

برای کدگذاری n عنصر اطلاعاتی حداقل به r بیت نیاز داریم که از رابطه زیر بدست می‌آید.

$$\boxed{2^{r-1} < n \le 2^r}$$

مثال ۲: برای کدگذاری 5 کاراکتر حداقل به چند بیت نیاز داریم

$$4 = 2^{3-1} < 5 \le 2^3 = 8 \Rightarrow \boxed{r = 3}$$

حداقل به 3 بیت نیاز داریم.

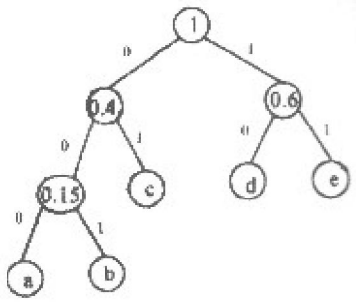
روش کار:

۱ـ درخت هافمن با حداقل طول مسیر وزن را تشکیل می‌دهیم.

۲ـ از ریشه شروع کرده به تمام اتصال‌های چپ 0 و تمام اتصال‌های راست 1 می‌دهیم.

۳ـ برای پیدا کردن کد هر عضو از ریشه تا آن عضو 0 و 1ها را پشت سرهم می‌نویسیم.

مثال ۳: در درخت مثال ۱ در صورت کدگذاری خواهیم داشت:



a = 000 c = 01 e = 11
b = 001 d = 10

در مثال بالا برای 5 کار کتر یا 3 بیت عملیات کدگذاری انجام شده است.

نکته مهم:

دیده می‌شود که عناصری که تکرار کمتری داشته‌اند کد آن‌ها دارای پیشوند یکسانی بوده و فقط در قسمت انتهائی با هم متفاوت هستند.

a = 0 0 0

b = 0 0 1

گراف

تعریف:

هر گراف G شامل دو مجموعه V و E است:

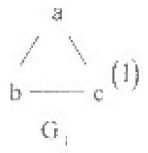
V = مجموعه محدود و غیرتهی از رئوس

E = مجموعه محدود و احتمالاً تهی از لبه‌ها

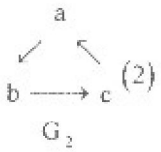
هر گراف G به صورت G (V , E) نمایش داده می‌شود.

نکته۱: هر گراف حداقل یک رأس دارد و نمی‌تواند کاملاً تهی باشد.

نکته۲: در گراف‌ها دو وضعیت جهت‌دار و بدون جهت وجود دارد که:



G_1



G_2

$V(G_1) = \{a , b , c\}$ $\quad$ $V(G_2) = \{a , b , c\}$

$E(G_1) = \{(a , b) , (a , c) , (b , c)\}$ $\quad$ $E(G_2) = \{\langle a , b\rangle , \langle b , c\rangle , \langle c , a\rangle\}$

دقت کنید که در گراف جهت‌دار G_2 $\langle v_1 , v_2\rangle \neq \langle v_2 , v_1\rangle$ اما در گراف بدون جهت $(v_1 , v_2) = (v_2 , v_1)$ است.

هر درخت حالت خاص از یک گراف است که سیکل یا دور ندارد. (هر گراف، درخت نیست اما هر درخت گراف است)

مثال:

G_3

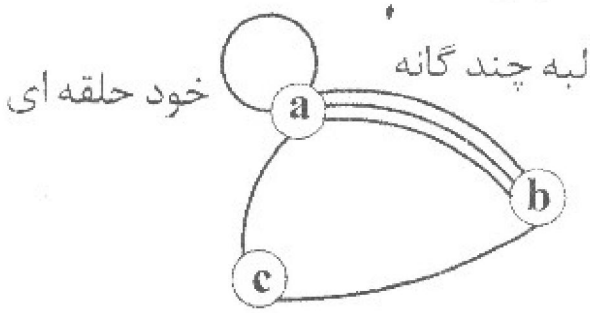
گراف G_3 به علت داشتن دور یا سیکل درخت نیست.

G_4

گراف G_4، درخت هم است.

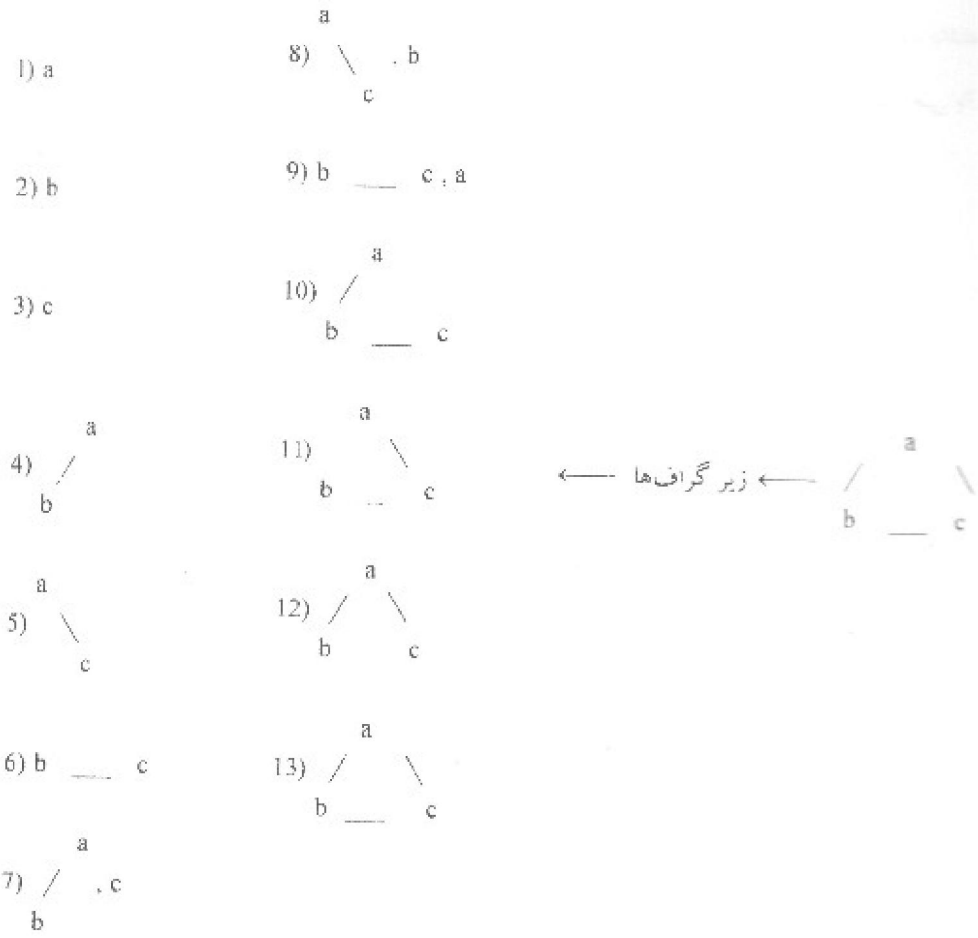
✓ **گراف چندگانه** (multi graph): گرافی که دارای لبه‌های چندگانه باشد.

✓ **گراف خود حلقه‌ای یا حلقه باز خوردی:** گرافی که در آن لبه یا یالی از یک رأس به خودش وجود داشته باشد.

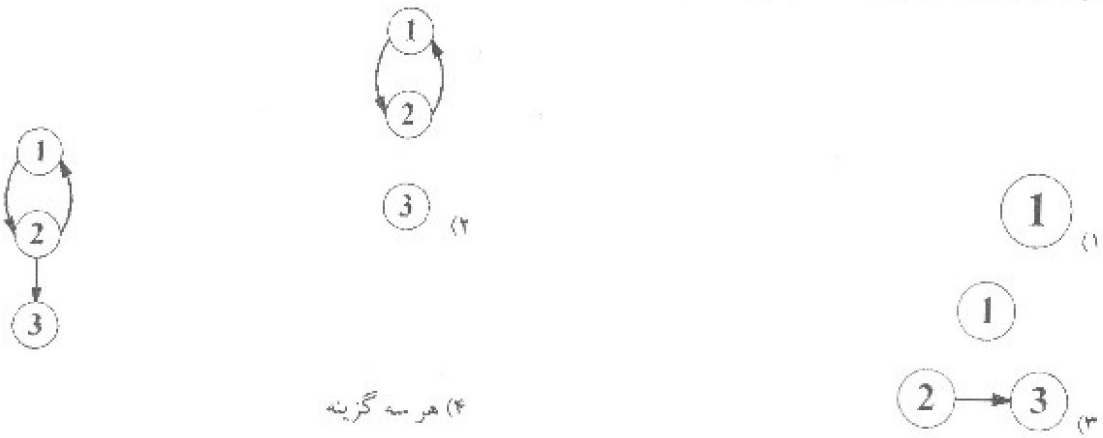


زیرگراف: هر زیر مجموعه از V (رئوس) و E (یال‌ها) به عنوان زیرگراف‌های G می‌توانند در نظر گرفته شوند.

مثال ۱:



۱ ـ گراف زیر را در نظر بگیرید. کدام گزینه، زیرگراف گراف فوق است؟ (آزاد۸۲)



۴) هر سه گزینه

گزینه ۴ صحیح می‌باشد.

□ **همسایگی (مجاور بودن) (adjacent)**

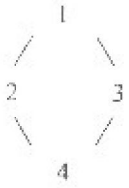
دو گره U ، V را در یک گراف مجاور (همسایه) گوئیم هرگاه لبه مستقیمی بین V, U وجود داشته باشد. به عبارت بهتر:

الف) در گراف بدون جهت: دو گره U ، V به شرطی مجاور (همسایه) هستند که لبه (یال) (U , V) وجود داشته باشد.

ب) در گراف جهت‌دار: هرگاه لبه ⟨U , V⟩ وجود داشته باشد می گوئیم V مجاور از رأس U است. یعنی یال مستقیمی از U به V وجود دارد،

به عبارت بهتر U مجاور به رأس V است.

مثال ۱: در گراف بدون جهت روبرو:



الف) گره‌های مجاور (همسایه) گره 1: گره‌های 3 ، 2

ب) گره‌های مجاور (همسایه) گره 2: گره‌های 1 ، 4

ج) گره‌های مجاور (همسایه) گره 3: گره‌های 1 ، 4

د) گره‌های مجاور (همسایه) گره 4: گره‌های 3 ، 2

در عین حال گره 4 مجاور گره 1 نیست، گره 2 مجاور 3 نیست.

مثال ۲: در گراف جهت‌دار روبرو:

$E(G) = \{\langle 1 , 2\rangle , \langle 2 , 1\rangle , \langle 2 , 3\rangle\}$

$N(G) = \{1 , 2 , 3\}$

الف) گره 2 مجاور به رأس 1 ، 3 است.

ب) گره 1 مجاور به رأس 2 است.

ج) گره 3 مجاور به رأس 2 نیست چون یال ⟨2 , 3⟩ وجود ندارد و فقط مجاور از رأس 2 است چون از 2 به 3 یال وجود دارد. ⟨3 , 2⟩

د) گره 3 نه مجاور به رأس 1 است یعنی یال ⟨1 , 3⟩ وجود ندارد و نه مجاور از رأس 1 است یعنی یال ⟨3 , 1⟩ وجود ندارد.

☑ **گراف ساده:** گرافی که فاقد خود حلقه‌ای و لبه‌های چندگانه باشد.

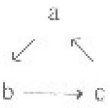
گراف کامل: گرافی که دارای حداکثر لبه‌ها باشد. به عبارت بهتر:

بین هر زوج گره دلخواه آن لبه مستقیمی وجود داشته باشد.

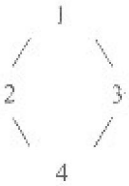
یا

هر گره U مجاور هر گره دیگر V در گراف باشد.

مثال: گراف‌های غیر کامل

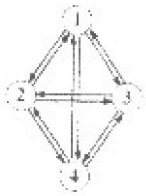


گراف غیر کامل جهت‌دار

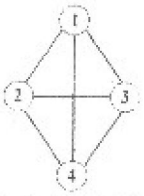


گراف غیر کامل بدون جهت

مثال: گراف‌های کامل



گراف کامل جهت‌دار $= G_2$



گراف کامل بدون جهت $= G_1$

فرمول:

- ☑ حداکثر تعداد لبه‌ها برای یک گراف بدون جهت با n رأس $\frac{n(n-1)}{2}$
- ☑ حداکثر تعداد لبه‌ها برای یک گراف جهت‌دار با n رأس $n(n-1)$

فرمول‌های فوق در حقیقت تعداد لبه‌های یک گراف کامل بدون جهت $\frac{n(n-1)}{2}$ و جهت‌دار $n(n-1)$ هستند.

مثال ۱: در گراف بدون جهت کامل G_1 با $n = 4$ رأس با توجه به رابطه $\frac{n(n-1)}{2}$ به تعداد زیر یال، وجود دارد:

$$n = 4 \rightarrow \text{تعداد یال‌ها} = \frac{4(4-1)}{2} = 6$$

مثال ۲: در گراف جهت‌دار کامل G_2 با $n = 4$ رأس باتوجه به رابطه $n(n-1)$ به تعداد زیر یال، وجود دارد:

$$n = 4 \rightarrow \text{تعداد یال‌ها} = 4(4-1) = 12$$

مسیر: هر مجموعه از لبه‌های پشت سرهم که دو رأس را به هم متصل می‌کنند مسیر نامیده می‌شود.

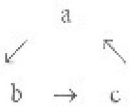
مثال ۱:

در گراف روبرو

a ___ b ___ c: مسیری از رأس a به رأس c

b ___ a ___ c: مسیری از رأس b به رأس c

مثال ۲: در گراف روبرو



a ⟵ b ⟵ c: مسیری از a به b

c ⟵ a: مسیری از c به a

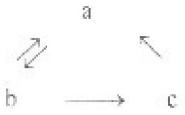
در عین حال b ⟵ c یا c ⟵ a ⟵ b مسیر نیستند چون از c به b و از b به a لبه مستقیم نداریم.

طول مسیر: تعداد لبه‌های موجود در مسیر را طول مسیر می‌گویند.

مسیر ساده: مسیری است که همه رئوس آن به جز احتمالاً اول و آخر، متفاوت است.

حلقه یا سیکل: مسیر ساده‌ای که اولین و آخرین رأس آن با هم برابر است.

مثال: در گراف جهت‌دار روبرو



مسیرهای ساده به طول 2:
- a — b — c
- c — a — b
- a — b — a (سیکل)
- b — a — b (سیکل)

مسیرهای ساده به طول 3:
- a — b — c — a (سیکل)
- c — a — b — c (سیکل)
- b — c — a — b (سیکل)

مسیرهای a — b — a و b — a — b ساده نیستند چون تکرار گره‌ها در وسط مسیر دیده می‌شود.

مثال: در گراف غیر جهت‌دار روبرو:

مسیرهای ساده به طول 2:
- a — b — c
- c — a — b
- b — c — b (سیکل)
- b — a — b (سیکل)
- a — c — a (سیکل)
- a — b — a (سیکل)
- c — a — c (سیکل)
- c — b — c (سیکل)
- b — c — a
- b — a — c
- c — b — a
- a — c — b

سیرهای ساده به طول 3 | a — b — c — a (سیکل)
b — c — a — b (سیکل)
c — a — b — c (سیکل)

مسیرهای b ___ a ___ b و a ___ b ___ a و c ___ a ___ c ___ a و a ___ c ___ a ___ c و b ___ a ___ b ___ a و a ___ c ___ a ___ c

c ___ b ___ c ساده نیستند چون تکرار گره‌ها در وسط مسیر دیده می‌شود.

۱- اگر a, b دو گره در یک گراف بدون‌جهت G باشند و اگر دو مسیر P_1، P_2 از a به b وجود داشته باشد، آنگاه: (دولتی ۸۲)

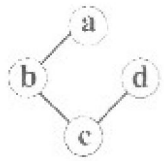
۱) a , b مجاورند.
۲) G نمی‌تواند یک گراف باشد.
۳) G دارای چرخه است.
۴) احتیاج به جهت مسیر داریم.

گزینه ۳ صحیح می‌باشد.

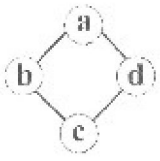
قضیه ۳: در گراف بدون جهت دو رأس دلخواه v_1 ، v_2 را متصل می‌گوئیم اگر مسیری از v_1 به v_2 و یا بالعکس وجود داشته باشد.

قضیه ۴: گراف بدون جهت را متصل یا همبند گوئیم اگر برای هر دو رأس v_1 و v_2 مسیری از v_1 به v_2 و یا بالعکس وجود داشته باشد.

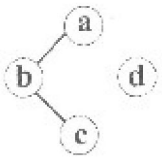
هر درخت یک گراف همبند (متصل) بدون دور یا حلقه است.



گراف متصل درخت است

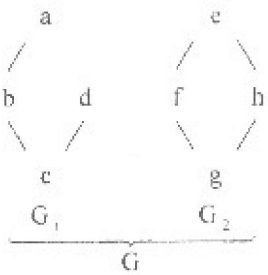


گراف متصل درخت نیست



گراف غیر متصل درخت نیست.

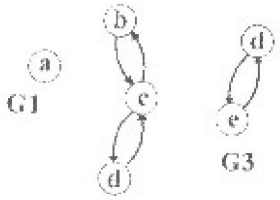
مؤلفه اتصال: یک مؤلفه اتصال یا به طور ساده‌تر یک مؤلفه در گراف بدون جهت بزرگترین زیر گراف متصل آن گراف است.



در گراف G روبرو G_1 و G_2 دو مؤلفه اتصال گراف G هستند.

قضیه ۵: یک گراف جهت‌دار کاملاً متصل نامیده می‌شود. هرگاه برای هر زوج رأس v_1 و v_2 مسیری جهت‌دار از v_1 به v_2 و همچنین از v_2 به v_1 وجود داشته باشد.

قضیه ۶: یک مؤلفه کاملاً متصل بزرگترین زیرگرافی است که کاملاً متصل باشد.



سه زیر گراف G_1 ، G_2 ، G_3 سه مؤلفه کاملاً متصل دارند.

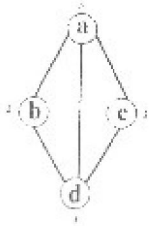
درجه یک رأس در گراف:

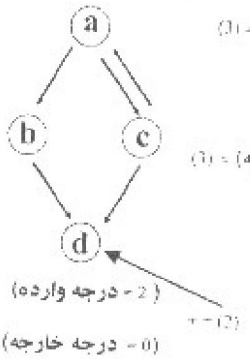
۱ـ درجه یک رأس از گراف بدون جهت تعداد لبه‌ها یا یال‌های متلاقی با آن رأس است.

۲ـ در گراف جهت‌دار درجه یک رأس برابر است با: درجه وارده + درجه خارجه

درجه وارده: تعداد یال‌هایی که به رأس وارد می‌شود.

درجه خارجه: تعداد یال‌هایی که از رأس خارج می‌شوند.



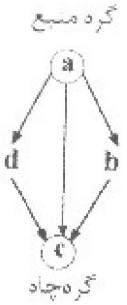


(3 - درجه خارجه) + (1 - درجه وارده) = (3)

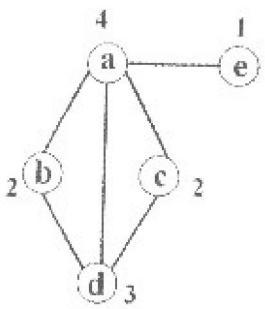
(1 - درجه وارده) + (2 - درجه خارجه) = (3)

گره چاه: به گره‌ای که درجه خروجی آن در گراف جهت‌دار 0 باشد.

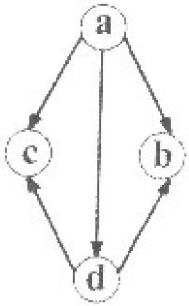
گره منبع: به گره‌ای که درجه ورودی آن در گراف جهت‌دار 0 باشد.



✓ **درجه گراف:** درجه هر گراف برابر با بزرگترین درجه گره‌های آن گراف است.



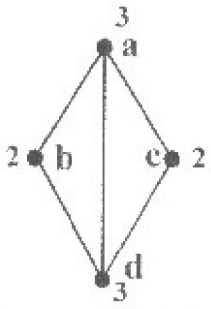
درجه گراف = 4



درجه گراف = 3

نکته: اگر درجه رأسی فرد باشد آن رأس را فرد گویند و اگر زوج باشد آن رأس را زوج گویند.

تعداد رأس‌های فرد یک گراف غیر جهت‌دار همواره عددی زوج است.

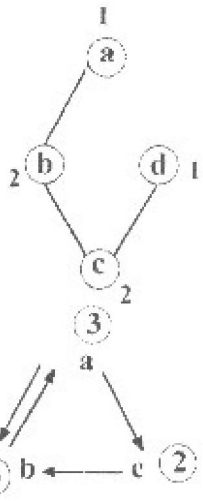


۲ رأس فرد a و d وجود دارد.

نکته مهم: اگر در گراف دلخواه G (جهت‌دار یا بدون جهت) درجه تمام رئوس را بدانیم آنگاه:

d_i: درجه رأس i $e = \frac{\sum d}{2}$ = تعداد یال‌ها یا تعداد لبه‌ها (یال‌ها) = $\frac{\text{مجموع درجه رئوس}}{2}$

مثال ۱:



$e = \frac{\sum d}{2} = \frac{1+2+2+1}{2} = 3$ تعداد یال‌ها

$e = \frac{\sum d_i}{2} = \frac{3+3+2}{2} = 4$ تعداد یال‌ها

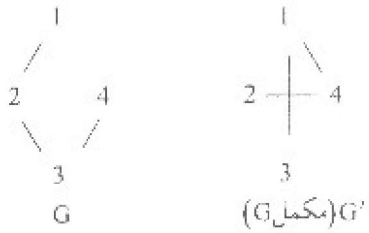
نکته مهم:

گراف مکمل: در صورتیکه G یک گراف دلخواه باشد گراف مکمل G گرافی است که با گراف G هیچ یال مشترکی نداشته ولی رئوس مشترک دارد و مجموع دو گراف یک گراف کامل تشکیل می‌دهند در این حالت

پیش فرض گراف بدون جهت

$\underbrace{\text{گراف مکمل } G'}_{\text{تعداد یال‌ها}} + \underbrace{\text{گراف } G}_{\text{تعداد یال‌ها}} = \frac{\overbrace{n(n-1)}^{\text{تعداد رئوس}}}{2}$

مثال:



در این حالت:

تعداد یال‌های G $= 3 +$ تعداد یال‌های G' $(3) = 6$

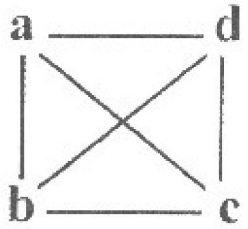
یعنی گراف کامل با 4 رأس $\frac{n(n-1)}{2}$ یال یعنی $\frac{4 \times 3}{2} = 6$ دارد.

مثال۲: اگر G یک گراف ساده از درجه n باشد. در صورتی که گراف G دارای 56 یال باشد و مجموع درجات رئوس مکمل گراف G برابر 160 باشد مقدار n کدام است؟

$$\frac{\sum d}{2} + 56 = \frac{n(n-1)}{2} \Rightarrow \frac{160}{2} + 56 = \frac{n(n-1)}{2} \Rightarrow \underline{n = 17}$$

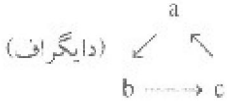
تعداد رئوس گراف $n = 17$ است اما در مسئله n = درجه گراف G خواسته شده و چون در یک گراف کامل با n رأس درجه گراف $n - 1$ است پس جواب مسئله 16 است.

نکته: در یک گراف کامل با n رأس درجه گراف $n - 1$ است.

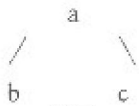


گراف کامل با 4 رأس از درجه 3

به گراف جهت‌دار، دایگراف (digraph) نیز گفته می‌شود.

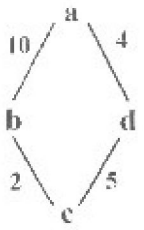


گراف G متصل است اگر و فقط اگر یک مسیر ساده بین هر دو گره آن وجود داشته باشد.



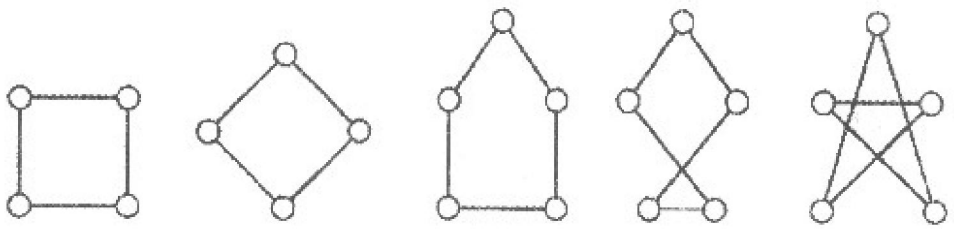
گراف G برچسب‌دار یا وزن‌دار یا شماره‌دار می گویند اگر اطلاعاتی به یال‌های آن نسبت داده شود.

اگر اطلاع نسبت داده شده به یال‌ها یک عدد غیرمنفی باشد به آن وزن یا هزینه یال‌ها می گویند.



○ گراف برچسب‌دار یا وزن‌دار

نکته مهم: اگر در گرافی درجه تمام رأس‌ها دقیقاً 2 باشد، گراف را منظم می‌گوئیم.



نمایش گراف:

به طور کلی روش‌های زیر برای نمایش گراف‌ها استفاده می‌شود:

۱) ماتریس مجاورتی
۲) لیست مجاورتی (همجواری)
۳) لیست مجاورتی معکوس

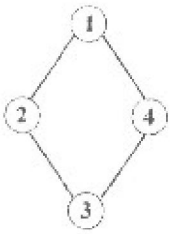
ماتریس مجاورتی:

فرض کنید $G(V , E)$ یک گراف با n رأس باشد، ماتریس مجاورتی گراف G یک آرایه دو بعدی است به صورت $n \times n$ که n^2 خانه دارد.

و به صورت زیر پیاده‌سازی می‌شود:

۱) گراف بدون جهت:

در صورتیکه لبه یا یالی به صورت (v_i , v_j) وجود داشته باشد $A[i][j] = 1$ است در غیر این‌صورت $A[i][j] = 0$ است.



⟶

	1	2	3	4
1	0	1	0	1
2	1	0	1	0
3	0	1	0	1
4	1	0	1	0

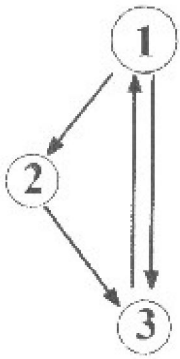
مشخصات:

الف) ماتریس مجاورتی یک گراف غیر جهت‌دار همیشه متقارن است،بنابراین همیشه می‌توانیم فقط بالا مثلث یا پائین مثلث را نگهداری کنیم.($\frac{n(n-1)}{2}$ خانه)

ب) تعداد 1های ماتریس همیشه 2 برابر تعداد یال‌ها است.

ج) مجموع عناصر سطری یا ستونی هر رأس درجه آن رأس را نشان می‌دهد.

۲ـ گراف جهت‌دار:



	1	2	3
1	0	1	1
2	0	0	1
3	1	0	0

الف) ماتریس مجاورتی یک گراف جهت‌دار همیشه متقارن نیست بنابراین همیشه باید $n \times n$ خانه ماتریس را نگهداری کرد.

ب) تعداد 1های ماتریس همیشه برابر تعداد یال‌ها است.

ج)

درجه رأس = مجموع عناصر ستونی هر رأس = درجه ورودی

مجموع عناصر سطری هر رأس = درجه خروجی

نکات مهم در مورد ماتریس مجاورتی:

۱) از نقاط قوت ماتریس مجاورتی این است که سرعت عملیات دسترسی به این که آیا دو گره i و j مجاور هم هستند یا نه (یال مستقیم از i به j هست یا خیر) از مرتبه $o(1)$ است.

۲) اغلب الگوریتم‌ها با استفاده از ماتریس مجاورتی با زمان $o(n^2)$ بدست می‌آیند.

۳) برای گراف اسپارس یعنی گرافی که دارای تعداد کمی لبه هست زمان لازم $o(n+e)$ است.

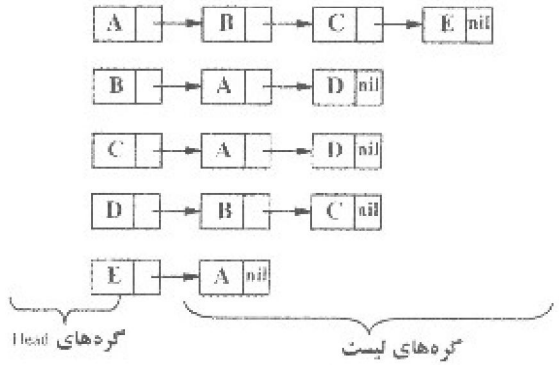
البته به شرطی که $e < \frac{n^2}{2}$ باشد.

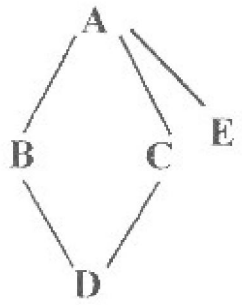
۴) اگر A ماتریس مجاورتی گراف G باشد. در این‌صورت در ماتریس A^k خانه A[i, j] تعداد مسیرهایی از i به j را نشان می‌دهد که طول k دارند.

لیست مجاورتی:

۱) گراف بدون جهت:

در این وضعیت برای یک گراف با n رأس و e لبه (یال) برای نمایش لیست مجاورتی به n گره Head و 2e گره لیست احتیاج داریم.

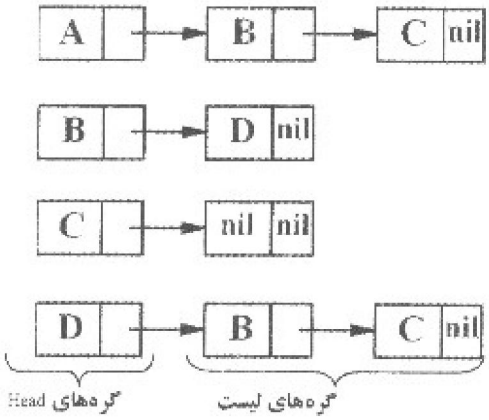


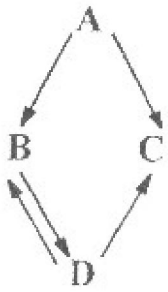


نکته: در شکل فوق درجه هر گره تعداد گره‌های لیست آن رأس می‌باشد. همسایه‌های هر گره را نیز می‌توانیم به راحتی در این وضعیت بدست آوریم.

۲) لیست مجاورتی برای گراف جهت‌دار:

در این وضعیت برای گراف جهت‌دار با n رأس و e لبه (یال) لیست مجاورتی احتیاج به n گره Head و e گره لیست دارد.





نکات مهم در مورد لیست مجاروتی:

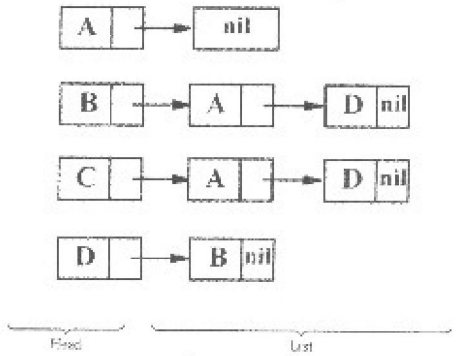
۱) اگر تعداد رئوس گراف G برابر n باشد. تعداد کل لبه‌ها در زمان $O(n+e)$ تعیین می‌شود.

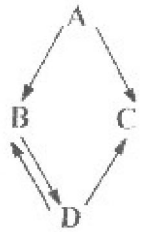
۲) برای یک گراف جهت‌دار، درجه خروجی هر رأس با شمارش تعداد گره‌ها در لیست مجاورتی آن گره به دست می‌آید این بدین معناست که می‌توانیم تعداد کل خطوط یک گراف جهت‌دار را در زمان $O(n+e)$ تعیین کنیم.

لیست مجاورتی معکوس

در لیست مجاورتی (در ماتریس جهت‌دار) به راحتی می‌توانیم درجه خروجی هر رأس را بدست آوریم که همان تعداد گره‌های لیست هر گرهٔ Head می‌باشد. برای پیدا کردن درجهٔ ورودی به راحتی نمی‌توانیم عمل کنیم. چون باید تمام گره‌های Head را جستجو کنیم به غیر از گره‌ای

که می‌خواهیم درجهٔ خروجی آن را بدست آوریم. برای آن‌که به راحتی بتوانیم این کار را انجام دهیم می‌توانیم از لیست مجاورتی معکوس استفاده کنیم که در این وضعیت باز هم n گره Head داریم.



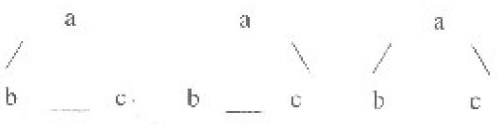


نکته: در لیست مجاورتی معکوس گره‌های لیست، درجهٔ ورودی را نشان می‌دهند و از روی آن می‌توانیم درجهٔ گرهٔ خروجی را بدست آوریم ولی به سختی.

درخت پوشا (Spanning tree)

در صورتیکه G یک گراف متصل (همبند) با n رأس، حداقل n – 1 لبه دارد. در این صورت درخت پوشا در این گراف درختی است که n رأس گراف را داشته و دقیقاً n – 1 لبه (یال) را اختیار می‌کند.

مثال: در گراف کامل روبرو با 4 رأس درخت‌های پوشای ممکن نشان داده شده‌اند که هر کدام 3 لبه دارند.



درخت‌های پوشا

مثال ۲: در یک گراف کامل با 10 رأس چند یال را کم کنیم تا به درخت پوشا برسیم:

۱) 45 ۲) 36 ۳) 9 ۴) 10

حل: تعداد یال‌های یک گراف کامل با n رأس (بدون جهت)

$$\text{تعداد یال‌ها} = \frac{n(n-1)}{2} \rightarrow n = 10 \quad \text{تعداد یال‌ها} = \frac{10 \times 9}{2} = 45$$

در یک درخت پوشا از یک گراف با n رأس n - 1 یال وجود دارد در نتیجه در گراف با 10 رأس درخت پوشا 9 یال دارد.

بنابراین 36 = 9 – 45 یال باید کم شود.

پیمایش گراف‌ها

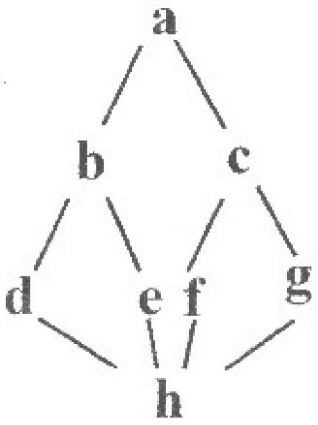
به طور کلی برای پیمایش گراف دو روش وجود دارد که منجر به درخت پوشا یا spanning tree می‌شود. درخت پوشای یک گراف که n راس دارد و حاصل از پیمایش گراف می‌باشد حتماً n-1 یال دارد.

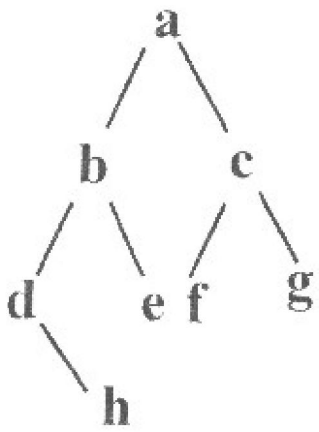
الف) پیمایش bfs:

این پیمایش را پیمایش سطحی (عرضی، ردیفی) می‌نامیم و برای شبیه‌سازی آن از صف استفاده می‌کنیم. به ترتیب زیر هر گراف دلخواه را می‌توانیم به فرم سطحی یا bfs پیمایش کنیم. ابتدا رأس شروع را در نظر می‌گیریم. سپس همسایه‌های مستقیم آن‌را ملاقات می‌کنیم. این کار را از سمت چپ به راست انجام می‌دهیم. بعد از این کار از سمت چپ‌ترین همسایه شروع کرده همسایه‌های مستقیم آن‌را می‌بینیم. دقت کنید اگر همسایه‌ای برای یک گره دیده شود، آن گره برای گره‌های دیگر دیده نمی‌شود. برای آن‌که حتماً باید درخت پوشا داشته باشیم.

مثال: مطلوبست پیمایش bfs گراف زیر از رأس a:

bfs (a): a b c d e f g h



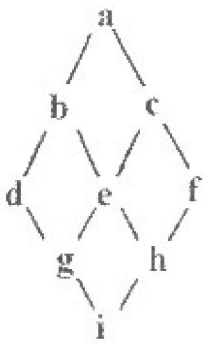


زمان اجرای الگوریتم bfs چنانچه از لیست‌های مجاورتی استفاده شود $O(n + e)$ می‌باشد.

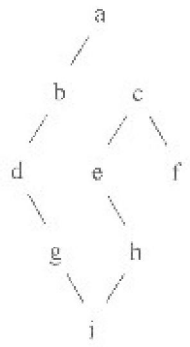
ب) پیمایش dfs: (depth first search) (عمقی)

در این پیمایش از هر گره که شروع کنیم با استفاده از سمت چپ‌ترین همسایه مستقیم آن در عمق حرکت می‌کنیم و مراحل در یک stack شبیه‌سازی و نگهداری می‌شود. در هر مرحله اگر به بن‌بست برخورد کنیم، یک عنصر را از پشته خارج کرده و به مرحله قبلی برمی‌گردیم تا شاید عنصر دیگری برای دیدن وجود داشته باشد. (در هر مرحله از پیمایش گره‌ای را انتخاب می‌کنیم که اولاً قبلاً رویت نشده باشند و ثانیاً در بین همسایه‌های چپ‌ترین باشند اگر به بن‌بست خوردیم به مرحله قبل برگشته تا همسایه‌های دیگر آن را بررسی کنیم.

مثال: پیمایش عمقی گراف زیر چیست؟ (مسابقات آموزشکده‌های فنی ۷۹)

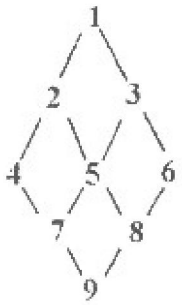


حل:

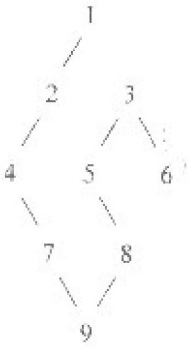


dfs : a , b , d , g , i , h , e , c , f

مثال: پیمایش عمقی گراف زیر کدام است؟ (دولتی ۸۲)



حل:



dfs : 1 , 2 , 4 , 7 , 9 , 8 , 5 , 3 , 6

پیمایش dfs با استفاده از لیست مجاورتی:

۱ـ از گره رأس ملاقات را شروع می کنیم.

۲ـ اولین گره لیست از گره رأس که تا به حال ملاقات نشده است را ملاقات کرده و سپس به لیستی می‌رویم که گره رأس آن گره لیست انتخاب شده است. هر زمان در یک لیست تمام گره‌ها قبلاً ملاقات شده بوده به مرحله قبل برگشته و گره دیگر را انتخاب می کنیم.

۳ـ عمل 2 را تا زمانیکه همه گره‌ها ملاقات شوند دنبال می کنیم.

مثال: لیست مجاورتی مربوط به گراف G به صورت زیر است. پیمایش عمقی این گراف کدام است:

Adjacent list
A:B,C,D
B:A,D,E
C:A,D,F
D:A,B,C
E:B,F
F:C,E

۱) A B D C F E

۲) A B E F C A

۳) A C F D E B

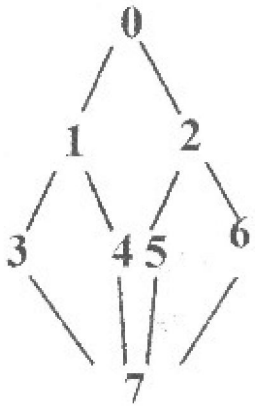
۴) A D C B E F

حل: گزینه ۱ صحیح می‌باشد.

از رأس A شروع می‌کنیم، و داخل لیست آن B را ملاقات کرده به رأس B می‌رویم چون A قبلاً ملاقات شده D را ملاقات کرده به رأس D می‌رویم B ، A قبلاً ملاقات شده‌اند پس C را ملاقات کرده به رأس C می‌رویم. D ، A قبلاً ملاقات شده‌اند پس F را ملاقات کرده به رأس F می‌رویم C قبلاً ملاقات شده ، E را ملاقات کرده به رأس E می‌رویم در این حالت F ، B قبلاً دیده شده‌اند و همین طور گره‌ها ملاقات شده پس پیمایش تمام شده است. خروجی: A B D C F E

اگر گراف G با لیست مجاورتی نمایش داده شود، زمان لازم توسط dfs برابر $O(e)$ است از آن‌جا که حلقه for به زمانی برابر $O(n)$ نیاز دارد، کل زمان برای تولید همه گراف‌های متصل $O(n+e)$ است. اگر گراف G توسط ماتریس مجاورتی ارائه شود آن‌گاه زمان لازم برای تعیین گراف‌های متصل برابر $O(n^2)$ می‌باشد.

مثال: پیمایش dfs گراف زیر کدام است؟



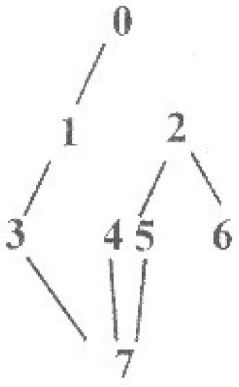
۱) 1 , 3 , 4 , 6 , 0 , 2 , 4 , 6

۲) 0 , 1 , 2 , 3 , 4 , 5 , 6 , 0 , 7

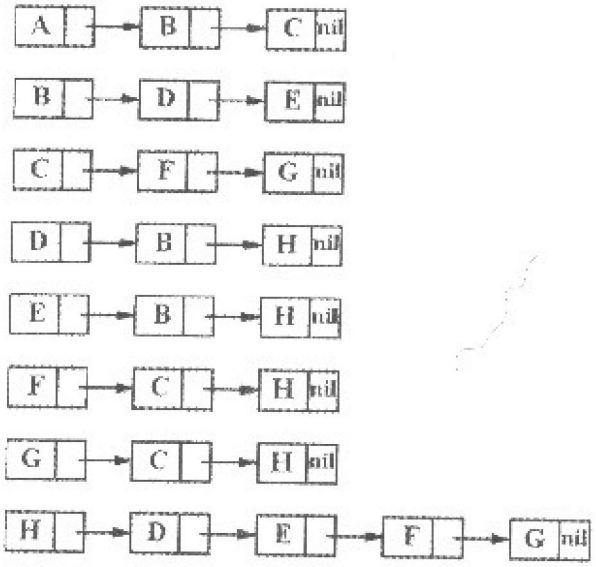
۳) 0 , 1 , 3 , 7 , 4 , 5 , 2 , 6

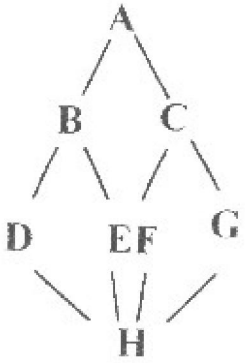
۴) 0 , 2 , 1 , 4 , 3 , 5 , 6 , 7

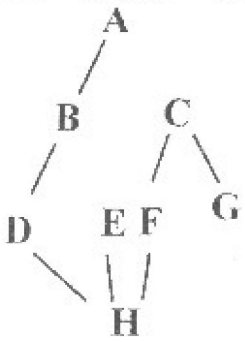
گزینه ۳ صحیح می‌باشد.



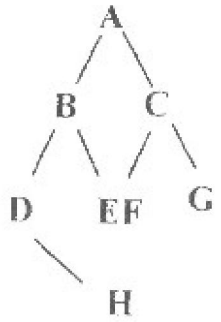
تمرین: مطلوبست پیمایش سطحی و عمقی گراف زیر:







bfs: A B C D E F G H



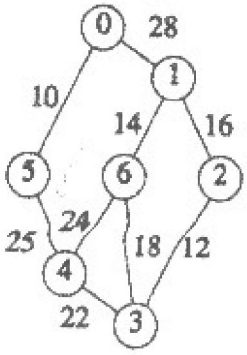
dfs: A B D H E F C G

درخت پوشای کم هزینه:

اساساً در گرافی که یال‌ها وزن داشته باشند به آن گراف وزن‌دار می‌گوییم. در این وضعیت پیمایش گراف وزن‌های متفاوتی را به وجود می‌آورد. برای بدست آوردن درخت پوشایی که هزینه یا وزن کمی را داشته باشد می‌بایست از الگوریتم‌های خاصی استفاده کنیم. تمام این الگوریتم‌ها بر دو اصل پیاده‌سازی می‌شود:

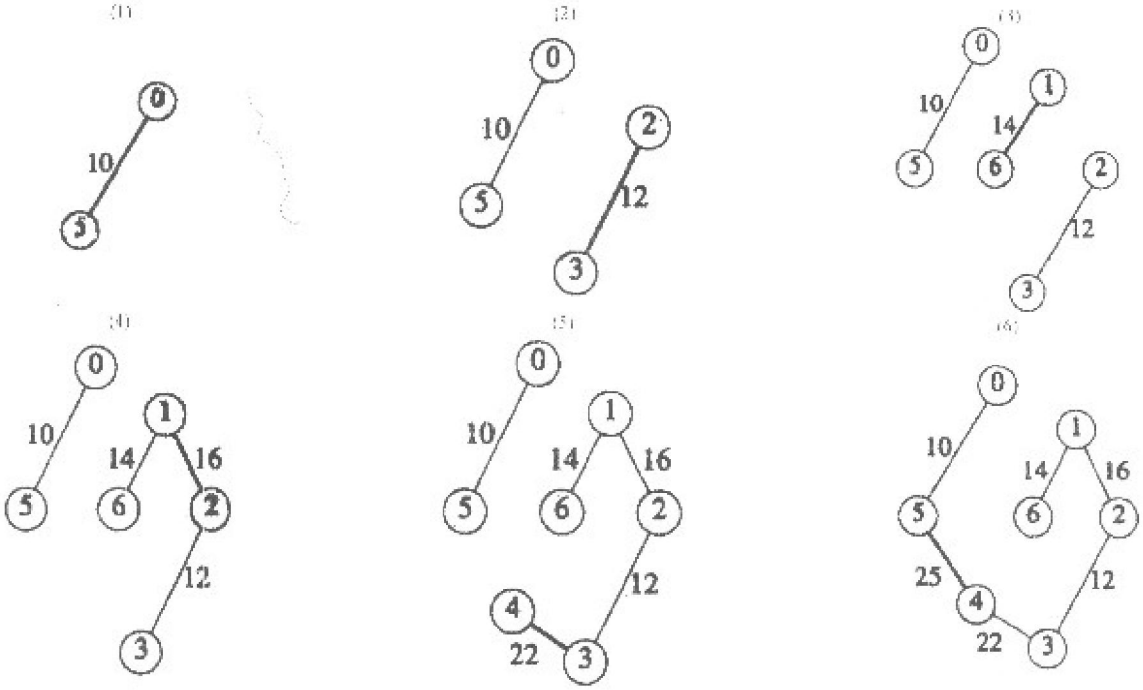
۱ـ همیشه در هر مرحله‌ای یالی را انتخاب می‌کنند که کمترین وزن را با مسیر انتخاب شده تا به حال داشته باشد.

۲ـ همیشه یالی که انتخاب می‌کنیم نباید با مسیر انتخاب شده تا به حال حلقه ایجاد کند.



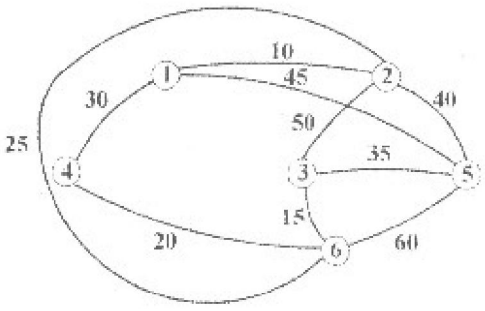
الف) الگوریتم راشال (کراسکال)

در این الگوریتم در هر مرحله یالی را انتخاب می‌کنیم که کمترین وزن را دارد. در این حالت رأس شروع مهم نیست چون ما از یال کوچک شروع می‌کنیم و فقط دقت می‌کنیم یالی که انتخاب می‌کنیم با درخت ایجاد شده تا به حال حلقه تولید نکند. نکتهٔ مهم این است که در هر مرحله از الگوریتم راشال ممکن است یک جنگل داشته باشیم. در مثال زیر هزینهٔ مینیمم گراف را برای درخت پوشا بدست آورده‌ایم. (گراف بالا)

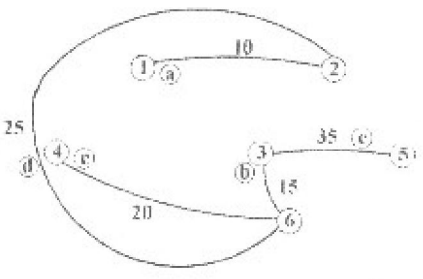


Rashal:

مثال:



حل:

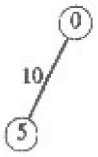


ب) الگوریتم پرایم:(prim)

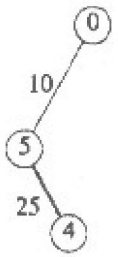
از یک رأس دلخواه شروع کرده و در هر مرحله رأسی را انتخاب می‌کنیم که با یکی از رئوس انتخاب شده تا به حال کمترین وزن را داشته باشد. تا درخت حاصل نیز کمترین وزن را داشته باشد. دقت می‌کنیم رأسی که انتخاب می‌کنیم نباید با مسیر انتخاب شده تا به حال ایجاد حلقه کند.

دقت کنید در هر مرحله از الگوریتم پرایم حتما یک درخت وجود دارد.

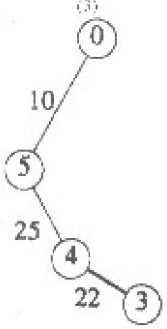
(1)



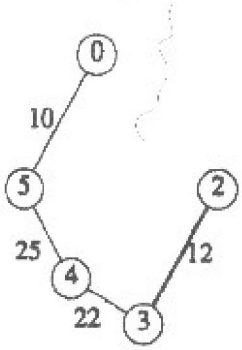
(2)



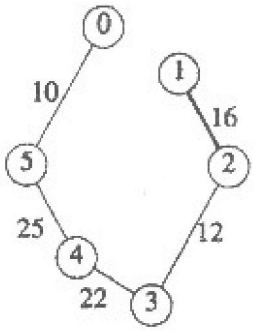
(3)



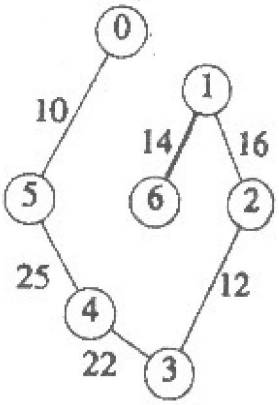
(4)



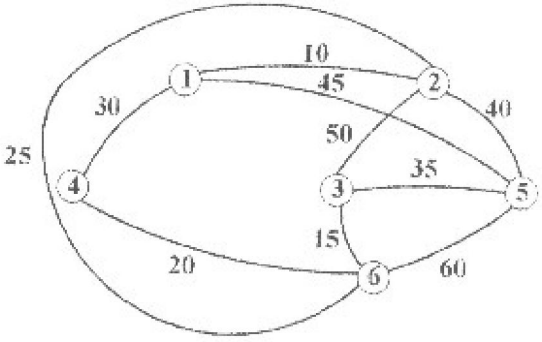
(5)



(6)

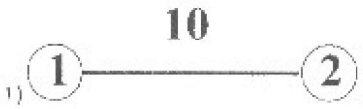


مثال:

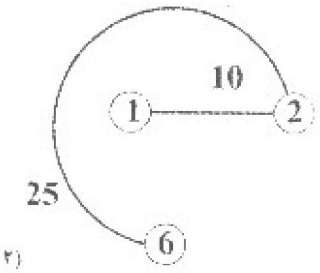


حل:

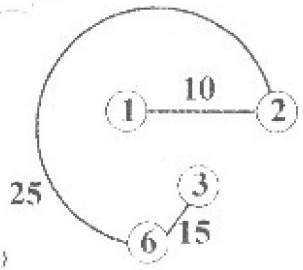
Prim:



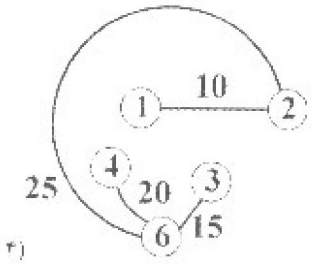
۱)



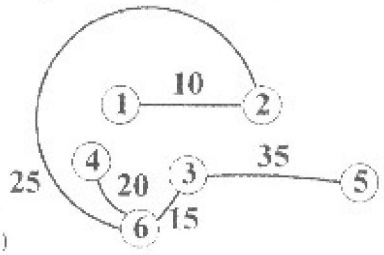
۲)



۳)



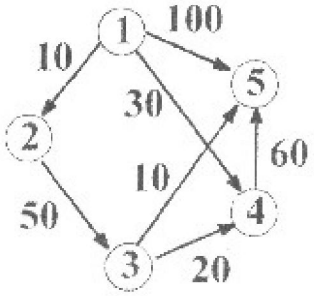
۴)



۵)

تمرین: در گراف زیر کمترین هزینه از گرهٔ 1 به 5 دارای مسیری به طول چند است؟

۱) 1　　۲) 2　　۳) 3　　۴) 4



$1 \to 2 \to 3 \to 5$: 70

$1 \to 4 \to 5$: 90　　$1 \to 5$: 100

$1 \to 2 \to 3 \to 4 \to 5$: 140

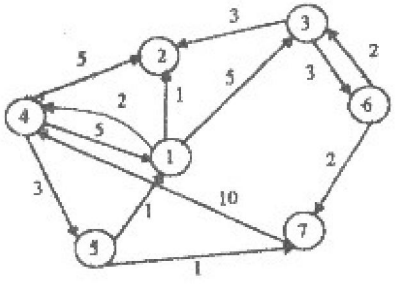
۳. **مثال:** در گراف زیر کم‌ترین هزینه از گره 1 به گره 7 عبارت است:

۱) 8

۲) 10

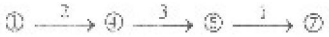
۳) 20

۴) 6



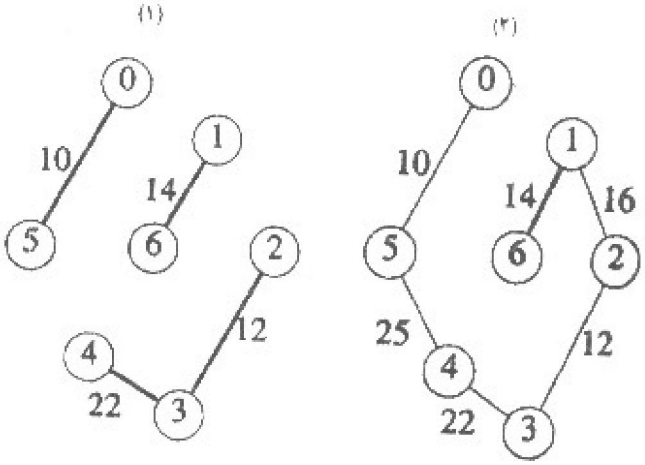
گزینه ۴ صحیح می‌باشد.

مسیر



ج) الگوریتم سالین: (sollin)

در این روش در مرحله اول یک جنگل متشکل از تمام رأس‌های گراف تشکل می‌دهیم به طوریکه هزینه آن می‌نیمم باشد. سپس با انتخاب یال‌هائی با هزینه می‌نیمم سعی می کنیم جنگل را به درختی با هزینه می‌نیمم تبدیل کنیم.



زمان الگوریتم‌ها

۱ـ حلقه‌ها

برای محاسبه زمان حلقه‌ها، تعداد تکرار آن‌ها را بدست آورده سپس از روی آن زمان الگوریتم را محاسبه می‌کنیم.

مثال ۱:

```
for i:=1 to n do
for j:=1 to n do  } (تکرار) n × n = O(n²)
...
```

مثال ۲:

```
for i:=1 to n do
    for j:=i+1 to n do
      ....
```

i	j	تکرار
1	2 .. n	n − 1
2	3 .. n	n − 2
⋮	⋮	⋮
n	n + 1 .. n	0
		$\frac{n(n-1)}{2} = O(n^2)$

مثال ۳:

```
for i:=1 to n do
    for j:=1 to n−i do
      for k:=1 to n do
        ...
```

i	j	k	تکرار
1	1 .. n − 1	n	(n − 1) n
2	1 .. n − 2	n	(n − 2) n +
⋮	⋮	⋮	⋮ +
n	1 .. n − n	n	(0) n +
			$O(n^2) = \left(\frac{n(n-1)}{2}\right) n$

مثال ۴:

```
for i:=1 to k do
    begin
    j:=1
while  j<=m  do
        j:=j+1;
    end;
```

i	j	تکرار
1	1 .. m	m
2	1 .. m	m
⋮	⋮	⋮
k	1 .. m	m
		k m = O (k m)

۲ـ مرتبه زمانی $O(\log_2 n)$

در دو وضعیت الگوریتم‌ها دارای این مرتبه زمانی خواهند بود

۱ـ متغیر دلخواهی مانند i دائمی بر مقدار ثابتی (k) تقسیم شود تا به حد پائینی برسد.

۲ـ متغیر دلخواهی مانند i دائمی در مقدار ثابتی (k) ضرب شود تا به مقدار n برسد.

مثال ۱: فرض کنید $n = 8$ مطلوبست تعداد تکرار حلقه زیر:

```
j: = 1;
while  j < = n  do
    begin
        j: = j * 2;
    end;
```

j	تکرار
1	1
$1 \times 2 = 2^1$	2
$2^1 \times 2 = 2^2$	3
$2^2 \times 2 = 2^3$	4
$2^3 \times 2 = 2^4$	خروج

تعداد تکرار $= 4 = \log_2 8 + 1$

$= \log_2 n + 1$

در حالت کلی در مثال بالا زمان $O(\log_2 n)$ خواهد بود.

نکته: در مثال بالا اگر شرط while به صورت $j < n$ باشد. تعداد تکرار $\log_2 n$ خواهد بود ولی زمان بازهم $O(\log_2 n)$ می‌باشد.

مثال ۲: تکرار حلقه زیر دارای چه مرتبه‌ای می‌باشد. (کنکور ۸۳)

```
i = 1;
while  (i < = n)
    {
    j = 1;
    while  (j < = n)
    {
    j = j * 2;
    }
    i = i + 1;
    }
```

تکرار $= (\log_2 n + 1)$

تکرار $= n(\log_2 n + 1)$

مثال ۳: برای $n = 8$ مطلوبست تعداد تکرار حلقه زیر:

```
i = n;
while  i > = 1  do
    i = i / 2;
```

n	تکرار
$8 = 2^3$	1
$4 = 2^2$	2
$2 = 2^1$	3
$1 = 2^0$	4
0	خروج

تکرار $= 4 = \log_2 8 + 1$

در حالت کلی زمان الگوریتم بالا $O(\log_2 n)$ خواهد بود.

نکته: در مثال بالا اگر شرط while به صورت $i > 1$ باشد. آن‌گاه تعداد تکرار $\log_2 n$ خواهد بود ولی زمان بازهم $O(\log_2 n)$ می‌باشد.

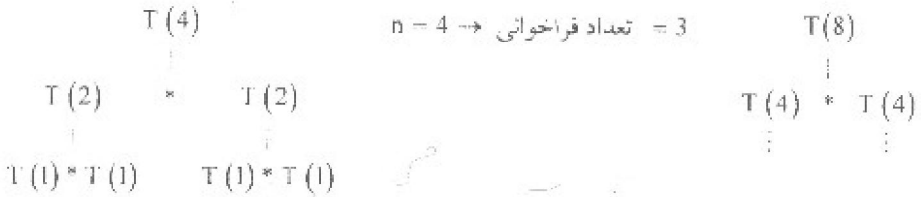
۳ـ مرتبه زمانی در توابع بازگشتی

در این توابع برای محاسبه مرتبه زمانی تعداد فراخوانی‌ها یا تعداد عملیات اصلی انجام شده نشان دهنده مرتبه زمانی خواهد بود که با محاسبه برای چند مقدار اولیه می‌توان مرتبه زمانی بدست آورد.

مثال ۱:

$$T(n) = \begin{cases} T(n-2) * T(n-2) & n > 1 \\ 1 & n = 1 \end{cases}$$

$n = 8 \rightarrow$ تعداد فراخوانی $= 7$ $\quad$ $n = 4 \rightarrow$ تعداد فراخوانی $= 3$

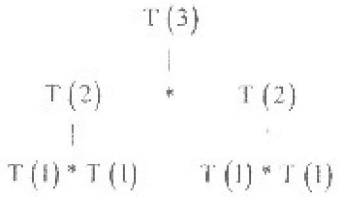


دیده می‌شود که تعداد فراخوانی برای هر $T(n)$ از رابطه $2^{n/2} - 1$ بدست می‌آید. بنابراین مرتبه زمانی $O\left(2^{n/2}\right)$ خواهد بود.

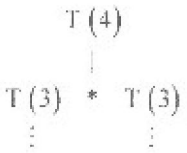
مثال ۲:

$$T(n) = \begin{cases} T(n-1) * T(n-1) & n > 1 \\ 1 & n = 1 \end{cases}$$

$n = 3 \rightarrow$ تعداد فراخوانی $= 3$



$n = 4 \rightarrow$ تعداد فراخوانی $= 7$



دیده می‌شود که تعداد فراخوانی برای هر $T(n)$ از رابطه $2^{n-1} - 1$ بدست می‌آید بنابراین مرتبه زمانی $O\left(2^n\right)$ می‌باشد.

مرتب سازی (sort)

✓ عملیات مرتب سازی برای هر آرایه از دو عمل مقایسه و تعویض تشکیل می‌شود.

نکته ۱: قسمتی از عملیات مرتب سازی که مقایسه براساس آن انجام می‌شود کلید نامیده می‌شود.

نکته ۲: الگوریتم‌های مرتب سازی انواع مختلفی دارند.

دسته بندی I

۱) **پایدار (متعادل)** (stable): الگوریتمی که ترتیب عناصر با مقدار مساوی را بعد از مرتب سازی حفظ کند.

۲) **ناپایدار (نامتعادل)** (non stable): الگوریتمی که ترتیب عناصر با مقدار مساوی را بعد از مرتب سازی حفظ نکند.

$$2 , 1_a , 3 , 5 , 1_b \xrightarrow[\text{پایدار}]{\text{مرتب سازی}} 1_a , 1_b , 2 , 3 , 5$$

$$2 , 1_a , 3 , 5 , 1_b \xrightarrow[\text{ناپایدار}]{\text{مرتب سازی}} 1_b , 1_a , 2 , 3 , 5$$

دسته بندی II

۱) **درجا** (inplace): استفاده از فضای کمکی ثابت برای مرتب سازی بدون وابستگی به تعداد عناصر.

۲) **برون ازجا** (Out place): استفاده از فضای کمکی متغیر برای مرتب سازی که وابسته به تعداد عناصر آرایه است.

دسته بندی III

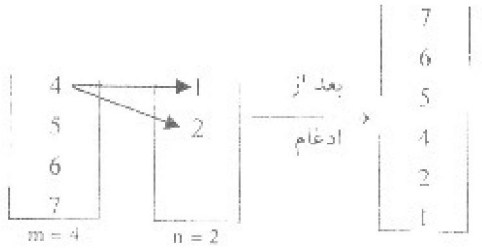
۱) **مرتب کننده داخلی** (Internal): عناصر مرتب شونده همگی در حافظه اصلی بوده و مقایسه مهم‌ترین عمل در این الگوریتم‌ها است و تعیین کننده زمان اجرا مقایسه است.

۲) **مرتب کننده خارجی** (External): عناصر مرتب شونده همگی در حافظه اصلی نبوده و دسترسی به عناصر مهم‌ترین عمل بوده و تعیین کننده زمان اجرا است.

نکته ۳: الگوریتم‌های مرتب سازی معمولاً از 2 حلقه تودرتو تشکیل می‌شوند.

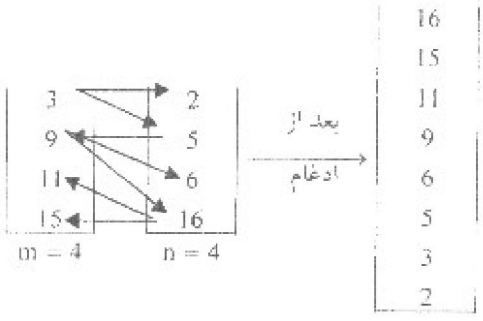
نکته ۴: اگر بخواهیم 2 آرایه مرتب شده را به فرم mتائی و nتائی بعد از ادغام بازهم مرتب ببینیم:

الف) حداقل مقایسه $\min(n , m)$



با 2 مقایسه

ب) حداکثر مقایسه $m - n - 1$



با 7 مقایسه

نکته۵: زمان‌های مختلف الگوریتم‌های مرتب سازی به شرح زیر است:

	بهترین حالت	متوسط	بدترین	توضیحات
۱ـ سریع (Quick sort)	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	نامتعادل و درجا
۲ـ انتخابی (Selection sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$	نامتعادل و درجا
۳ـ حبابی (Bubble sort)	$O(n)$	$O(n^2)$	$O(n^2)$	متعادل و درجا
۴ـ درجی (Insertion sort)	$O(n)$	$O(n^2)$	$O(n^2)$	متعادل و درجا
۵ـ Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	نامتعادل و درجا
۶ـ ادغامی (Merge)	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	متعادل و غیردرجا
۷ـ درختی (BST)	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	**غیردرجا**

همه الگوریتم‌های sort، زمانی بین $O(n)$ و $O(n^2)$ را اختیار می‌کنند.

چون عمل مهم در این الگوریتم‌ها که از نوع مرتب کننده داخلی هستند، مقایسه کردن است زمان‌های بالا با توجه به وضعیت مقایسه‌ای می‌باشد.

مثال: کدامیک از روش‌های مرتب سازی ذیل ترتیب اولیه رکوردهای هم کلید را حفظ می‌کند؟ (کنکور ۸۳ آزاد)

۱) selection ۲) Quick ۳) ✓ insertion ۴) Heap

مرتب سازی حبابی (Bubble sort) (متعادل و درجا)

در این مرتب سازی آرایه در چندین گذر (pass) عمل می‌کند، در هر گذر هر عنصر با عنصر بعدی مقایسه شده و در صورت نیاز با هم جابجا می‌شوند.

15 ⟷ 19 14 11
15 19 ⟷ 14 11
15 14 19 ⟷ 11
15 14 11 [19]

انتهای گذر اول بزرگترین عنصر در انتهای لیست

در یک آرایه n‌تایی به طور کلی در روش حبابی $n-1$ گذر وجود خواهد داشت. و در هر گذر تعداد مقایسه‌ها یکی کمتر از گذر قبلی است چون در هر گذر یک داده در مکان اصلی خودش قرار می‌گیرد.

	(حداقل) بهترین	متوسط	بدترین (حداکثر)
مقایسه	$O(n) = n-1$ مقایسه	$O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$ مقایسه
جابجایی	$O(1) = 0$ جابجایی	$O(n^2)$	$\frac{n(n-1)}{2} = O(n^2)$ جابجایی

متوسط تعداد جابجائی یا مقایسه $\frac{n(n-1)}{2}$ است. (حداکثر مقایسه یا جابجائی)

بهترین حالت = لیست مرتب باشد
- $n-1$ مقایسه (در گذر اول)
- 0 جابجائی

بدترین حالت = لیست مرتب معکوس باشد.
- $\frac{n(n-1)}{2}$ مقایسه در $n-1$ گذر
- $\frac{n(n-1)}{2}$ جابجائی

بهترین حالت: (لیست مرتب)

آرایه مرتب: $1, 2, 3, \ldots, n$ ← روش حبابی ←
- مقایسه در گذر اول $n-1$
- جابجائی در گذر اول 0

در صورتیکه از الگوریتم II استفاده کنیم در انتهای گذر اول با مقایسه دوبه‌دو در n داده ($n-1$ مقایسه) در آرایه مرتب چون هیچ جابجائی انجام نمی‌شود. نیازی به رفتن به گذر بعدی نداریم در نتیجه: **$n-1$ (مقایسه) و 0 (جابجائی)** داریم.

در صورتیکه از الگوریتم I استفاده کنیم در انتهای گذر اول در آرایه مرتب با آن‌که هیچ جابجائی انجام نشده ولی چون مانند الگوریتم II متغیر flag برای تشخیص وجود ندارد گذرهای بعدی نیز انجام می‌شود گرچه هیچ جابجائی نداریم. درنتیجه: **$\frac{n(n-1)}{2}$ (مقایسه) و 0 (جابجائی)** داریم.

بدترین حالت: (لیست نامرتب - مرتب معکوس)

آرایه مرتب معکوس: $n, n-1, n-2, \ldots, 1$ ← روش حبابی ←
- مقایسه $\frac{n(n-1)}{2}$
- جابجائی $\frac{n(n-1)}{2}$

مثال:

آرایه: $1, 2, \ldots, n-1, n$

	جابجائی	مقایسه
گذر اول: $n-1, n-2, \ldots, 2, 1, n$	$n-1$	$n-1$
گذر دوم: $n-2, n-3, \ldots, 2, 1, n-1, n$	$n-2$	$n-2$
⋮	⋮	⋮
گذر $n-1$ ام: $1, 2, \ldots, n-1, n$	1	1
	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$

الگوریتم

(I)

```
for  i := 1  to  n - 1  do
    for  j := 1  to  n - i  do
        if  A[j] > A[j + 1]  then
            begin
            temp := A[j];
            A[j] := A[j + 1];
            A[j + 1] := temp;
            end;
```

(II)

```
flag = true;   i := 1;
while (i <= n)  And  (flag = true)  do
    begin
        flag := false;
    for  j := 1  to  n - i  do
        if  A[j] > A[j + 1]  then
begin  temp := A[j]; A[j] := A[j + 1]; A[j + 1] := temp;  end;
    i := i + 1;
    end;
```

مرتب سازی درجی (insertion sort) (متعادل و درجا)

در این مرتب سازی آرایه در چندین گذر (pass) عمل می‌کند، در هر گذر به ترتیب زیر عمل می‌کنیم:

گذر 1: عنصر $A[1]$ به طور طبیعی مرتب است. (0 مقایسه)

گذر 2: عنصر $A[2]$ با 1 مقایسه قبل یا بعد از $A[1]$ قرار می‌گیرد. (1 مقایسه) (حداقل 0 جابجائی و حداکثر 1 جابجائی)

گذر 3: عنصر $A[3]$ با مقایسه طوری بین $A[1]$ و $A[2]$ قرار می‌گیرد که آرایه مرتب باشد. (حداکثر 2 مقایسه و حداقل 1 مقایسه) (حداکثر 2 جابجائی و حداقل 0 جابجائی)

⋮

گذر n: عنصر $A[n]$ در میان عناصر $A[1]$ ، ... $A[n-1]$ طوری قرار می‌گیرد که آرایه مرتب باشد. (حداکثر $n-1$ مقایسه و حداقل 1 مقایسه) (حداکثر $n-1$ جابجائی و حداقل 0 جابجائی)

مثال: آرایه روبرو به صورت درجی مرتب شده است:

12 , 14 , 5 , 9 , 7

12: گذر اول — ورود 12 →

12 , 14: گذر دوم — ورود 14 → — 14 با 12 مقایسه

5 , 12 , 14: گذر سوم — ورود 5 → — 5 با 14 و 12 مقایسه

5 , 7 , 12 , 14: گذر سوم — ورود 7 → — 7 با 14 , 12 , 5 مقایسه

5 , 7 , 9 , 12 , 14: گذر چهارم — ورود 9 → — 9 با 14 , 12 , 7 مقایسه

	بهترین (حداقل)	متوسط	بدترین (حداکثر)
مقایسه	$n-1=O(n)$ مقایسه	$O(n^2)$	$\frac{n(n-1)}{2}=O(n^2)$
جابجائی	$0=O(1)$ جابجائی	$O(n^2)$	$\frac{n(n-1)}{2}=O(n^2)$

حداکثر مقایسه و جابجائی $\frac{n(n-1)}{2}$ است.

بهترین حالت = لیست کاملاً مرتب

- $O(1)$ 0 جابجائی
- $O(n)$ $n-1$ مقایسه

بدترین حالت = لیست کاملاً مرتب معکوس

- $O(n^2)$ $\frac{n(n-1)}{2}$ جابجائی
- $O(n^2)$ $\frac{n(n-1)}{2}$ مقایسه

بهترین حالت: لیست کاملاً مرتب

1 , 2 , 3 , ... , n

	مقایسه	جابجائی	
2 با 1 مقایسه و بعد از آن درج می‌شود.	$-\ 0$	$+\ 0$	1: گذر اول
3 با 2 مقایسه و بعد از آن درج می‌شود.	$-\ 1$	$+\ 0$	2 , 1 : گذر دوم
n با $n-1$ مقایسه و بعد از آن درج می‌شود.	$+\ 1$	$+\ 0$	3 , 2 , 1 : گذر سوم
		$+\ 0$	n , ... , 2 , 1 : گذر nام
	$n-1$	0	

بدترین حالت: لیست مرتب معکوس

n , n − 1 , ... , 2 , 1

	مقایسه	جابجائی	
	0	0	n: گذر اول
n با $n-1$ مقایسه و با یک جابجائی قبل از آن درج می‌شود.	$-\ 1$	$+\ 1$	n , n − 1 : گذر دوم
$n-2$ با $n-1$ و n مقایسه و با 2 جابجائی قبل از آن‌ها درج می‌شود.	$+\ 2$	$+\ 2$	n , n − 1 , n − 2 : گذر سوم
	$+\ \vdots$	$+\ \vdots$	
1 با 2 تا $n-1$ مقایسه و با $n-1$ جابجائی قبل از آن‌ها درج می‌شود.	$n-1$	$n-1$	n , ... , 2 , 1 : گذر nام
	$\frac{n(n-1)}{2}$	$\frac{n(n-1)}{2}$	

این روش برای آرایه با تعداد عناصر کمتر یا مساوی 20 سریعترین روش است.

الگوریتم (I):

```
for  j ← 2  to  n  do
     key ← A[j]
Insert  A[j]  Into  the  sorted  sequence  A[1 .. j – 1]
     I ← j – 1
while  I > 0  and  A[I] > key
     do  A[I + 1] ← A[I]
I = I – 1
A[I + 1] ← key
```

الگوریتم (II):

```
for  i : = 2  to  n  do
     begin
     j : = i;
     while  A[j] < A[j – 1]  do
        begin
        T : = A[j – 1];
        A[j – 1] : = A[j];
        A[j] : = T;
        j : = j – 1;
        end;
     end;
```

الگوریتم مرتب سازی انتخابی (Selection sort) (نامتعادل و درجا)

در این روش با شروع از ابتدای آرایه در هر مرحله (گذر = pass) از مرتب سازی با مقایسه اولین داده با بقیه داده‌ها هنگام رسیدن به انتهای آرایه محل درست یک عنصر (بزرگترین یا کوچکترین) پیدا شده سپس در صورت نیاز با یک جابجائی آن را در محل درست خودش قرار می‌دهیم در این وضعیت برای یک آرایه n تائی به $n - 1$ گذر نیاز داریم چون در انتهای گذر $n - 1$ داده آخر نیز در مکان واقعی خودش قرار گرفته است.

10 15 20 7 14

10 15 14 7 20 انتهای گذر اول

10 7 14 15 20 انتهای گذر دوم

10 7 14 15 20 انتهای گذر سوم

7 10 14 15 20 انتهای گذر چهارم

نکته مهم:

۱ـ بهترین حالت و بدترین حالت برای این روش مفهومی ندارد چرا که اگر آرایه مرتب یا نامرتب باشد الگوریتم به صورت زیر عمل می‌کند.

۱ـ آرایه مرتب $O(n^2)$ { مقایسات: $\frac{n(n-1)}{2}$ ، جابجائی: 0 }

۲ـ آرایه نامرتب $O(n^2)$ { مقایسات: $\frac{n(n-1)}{2}$ ، جابجائی: حداکثر $n-1$ }

در هر دو وضعیت مقایسات باید انجام شود و در انتهای گذر اول امکان تشخیص مرتب بودن آرایه وجود ندارد چون فقط مقایسات برای یک عنصر انجام می‌شود.

```
for  i : = n  downto  1  do
     begin
     max : = x[1];
     index : = 1;
for  j : = 1  to  i  do
     if  (x[j] > max)  then
         begin
         max : = x[j];
         index : = j;
         end;
     x[index] : = x[i];
     x[i] : = max;
     end;
```

آرایه : 10 , 12 , 13 , 1 , 5

انتهای گذر اول: 10 , 12 , 5 , 1 , 13

انتهای گذر دوم : 10 , 1 , 5 , 12 , 13

انتهای گذر سوم: 5 , 1 , 10 , 12 , 13

انتهای گذر چهارم : 1 , 5 , 10 , 12 , 13

انتهای sort

در گذر اول: $max = x[1]$ در نظر گرفته شده و با مقایسه با $x[2]$ تا $x[n]$ در انتهای گذر بزرگترین مقدار در max قرار گرفته و محل آن در index قرار می‌گیرد. سپس در صورت نیاز با یک جابجائی مقدار $x[index]$ (ماکزیمم) با مقدار محل واقعی آن $x[n]$ عوض می‌شود.

گذر دوم: $max = x[1]$ در نظر گرفته شده و با مقایسه با $x[3]$ تا $x[n-1]$ در انتهای گذر بزرگترین مقدار در داده‌های باقیمانده در max قرار گرفته و محل آن در index ذخیره می‌شود سپس در صورت نیاز با یک جابجائی مقدار $x[index]$ (ماکزیمم) با مقدار محل واقعی آن $x[n-1]$ عوض می‌شود.

در این الگوریتم در $n-1$ گذر یک آرایه nتائی مرتب می‌شود.

بهترین و بدترین حالت با هم فرقی ندارد.

تعداد مقایسات: $\frac{n(n-1)}{2}$

بهترین حالت — متوسط — بدترین حالت

نکته:

در گذر اول $n-1$ مقایسه

در گذر دوم $n-2$ مقایسه

⋮

1

جمع مقایسه $\frac{n(n-1)}{2}$

نکته مهم:

در انتهای هر گذر در صورتیکه آرایه نامرتب باشد حداکثر 1 مقایسه را داریم اما در هر وضعیتی (مرتب بودن یا نبودن آرایه) مقایسات باید انجام شود.

مرتب سازی ادغام (Merge sort) (متعادل و غیردرجا)

این روش مرتب‌سازی عموماً بر روی عناصری که در فایل‌ها قرار دارند اجرا می‌شود اگرچه می‌توان این روش را در مورد بردارها نیز به کار برد. در این روش فایل n عضوی به n قسمت به طول یک تقسیم و سپس هر دو قسمت مجاور به صورت مرتب شده با هم ادغام می‌شوند. در این حالت $\frac{n}{2}$ فایل به طول 2 ایجاد شده است. روند ادغام تا رسیدن به یک فایل به طول n ادامه می‌یابد. به عنوان مثال مرتب سازی دنبالهٔ زیر در روش ادغام به صورت زیر است:

بردار اولیه: [17] [12] , [44] [12] , [23] [58] , [25] [9]

گذر اول: [17 , 21] [12 , 44] [23 , 58] [9 , 25]

گذر دوم: [12 , 17 , 21 , 44] [9 , 23 , 25 , 58]

گذر سوم: [9 , 12 , 17 , 21 , 23 , 25 , 44 , 58]

در هر مرحله برای ادغام دو قسمت، عناصر اول از دو قسمت در نظر گرفته شده با یکدیگر مقایسه می‌شوند، مقدار کوچکتر در دنبالهٔ حاصل قرار می‌گیرد و عنصر بعدی از دنباله‌ای که یک عنصر آن استفاده شده جهت مقایسه در نظر گرفته می‌شود و این روند ادامه می‌یابد تا دو دنباله در هم ادغام شوند.

نکته: این الگوریتم همواره با پیچیدگی $O(n \log n)$ انجام می‌شود.

پیچیدگی اجرا در الگوریتم مرتب سازی ادغام.

	بهترین حالت	حالت متوسط	بدترین حالت
پیچیدگی اجرا	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

BST sort: (غیر درجا)

وضعیت	بدترین	متوسط	بهترین
زمان	$O(n^2)$	O (n log n)	O (n log n)

همان‌طور که در بحث درخت BST مطرح شد، یکی از کاربردهای مهم درخت BST مرتب سازی لیست‌ها می‌باشد برای این کار در دو مرحله عملیات انجام می‌شود.

۱ـ درج عناصر لیست در یک درخت BST تهی. (درج هر داده در حالت متوسط $O(\log n)$ در نتیجه درج n داده $O(n \log n)$ خواهد بود)

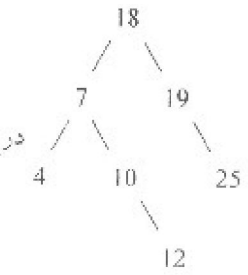
۲ـ پیمایش inorder (LVR) درخت BST که باعث نمایش لیست مرتب شده به صورت صعودی می‌باشد.

پیمایش RVL باعث نمایش لیست مرتب شده به صورت نزولی می‌باشد.

نکته: دقت می‌کنیم این روش مرتب سازی روی لیست‌هایی که داده تکراری دارند، ابتدا داده‌های تکراری را حذف کرده سپس در درخت نگهداری می‌کند. در این حالت متعادل بودن روش مرتب سازی مطرح نخواهد بود.

مثال: مرتب سازی آرایه 25 ، 19 ، 4 ، 12 ، 10 ، 7 ، 18 به روش BST خروجی عمل چیست؟

۱) درج آرایه در BST



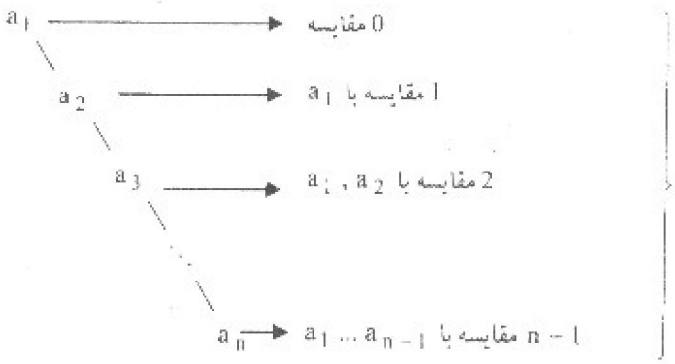
۲) درخت BST پیمایش inorder (LVR) : 4 ، 7 ، 10 ، 12 ، 18 ، 19 ، 25

درخت BST پیمایش RVL : 25 ، 19 ، 18 ، 12 ، 10 ، 7 ، 4

بدترین وضعیت در مرتب سازی BST (مهم)

بدترین حالت ورودی برای مرتب‌سازی BST، مرتب بودن آرایه ورودی (صعودی یا نزولی) است که درخت به صورت اریب ظاهر شده در این حالت در صورتیکه داده‌ها تکراری نباشند تعداد مقایسات برای درج $\frac{n(n-1)}{2}$ خواهد بود و زمان درج $O(n^2)$ خواهد بود.

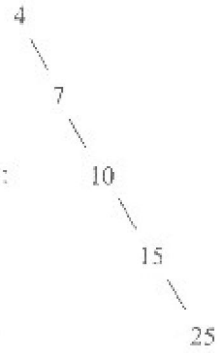
فرض کنید $a_1 > a_2 > a_3 > \ldots > a_n$ در نتیجه اگر داده به صورت $a_1 , a_2 , \ldots , a_n$ وارد شوند.



مثال ۱:

آرایه ورودی:

4 , 7 , 10 , 15 , 25



۱) درج در BST :

مقایسات	4	7	10	15	25	
	0	1	2	3	4	= 10

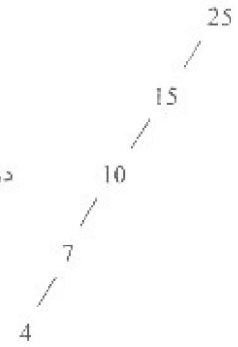
$n = 5 \rightarrow$ تعداد مقایسات $= \frac{n(n-1)}{2} = \frac{5 \times 4}{2} = 10$

۲) پیمایش inorder (LVR) : 4 , 7 , 10 , 15 , 25

مثال ۲:

آرایه ورودی:

25 , 15 , 10 , 7 , 4



۱) درج در BST

مقایسات	25	15	10	7	4	
	0	1	2	3	4	= 10

$n = 5 \rightarrow$ تعداد مقایسات $= \frac{n(n-1)}{2} = \frac{5 \times 4}{2} = 10$

۲) (LVR) inorder پیمایش : 4 , 7 , 10 , 15 , 25

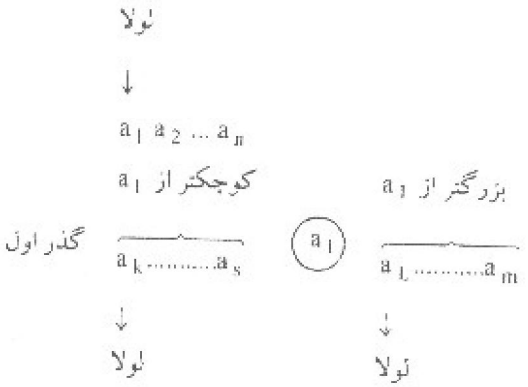
دیده می‌شود که برای یک آرایه n تائی در وضعیت بالا درخت با n سطح ایجاد شده است که برای درج n داده $\frac{n(n-1)}{2}$ مقایسه انجام می‌شود در نتیجه زمان درج $O(n^2)$ خواهد بود.

بهترین حالت در مرتب سازی BST

بهترین حالت ورودی برای مرتب سازی BST، نامرتب بودن آرایه ورودی می‌باشد که باعث می‌شود عمق درخت BST در حالت متوازن $(\log_2 n)$ قرار گرفته و در نتیجه زمان درج n داده $O(n \log n)$ و زمان مرتب سازی $O(n \log_2 n)$ خواهد بود.

Quick sort (مرتب سازی سریع) (نامتعادل و درجا)

الگوریتم مرتب سازی سریع از نوع الگوریتم‌های تعویضی بوده است که براساس جابجا کردن زیاد عناصر عمل می‌کند در این الگوریتم هر بار عنصری به عنوان pivot یا محور انتخاب می‌شود (پیش فرض داده اول آرایه) در این حالت بقیه داده‌های آرایه به گونه‌ای جابجا می‌شوند که کلیه عناصر کوچکتر از عنصر لولا در یک طرف آن و کلیه عناصر بزرگتر در طرف دیگر آن قرار گیرند. سپس دو آرایه ایجاد شده در دو طرف عنصر لولا (pivot) را به همین ترتیب دوباره مرتب می‌کنیم. (به صورت بازگشتی)



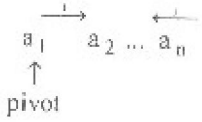
روش انجام کار در یک مرحله:

۱ـ عنصر اول را به طور پیش فرض به عنوان لولا (pivot) در نظر می‌گیریم.

۲ـ اندیس i: با شروع از حد پائین به سمت اولین داده‌ای در آرایه حرکت می‌کند که $A[i] > pivot$ باشد.

اندیس j: با شروع از حد بالا به سمت اولین داده‌ای در آرایه حرکت می‌کند که $A[j] < pivot$ باشد.

۳ـ بعد از مرحله 2
- الف) $i < j$ در نتیجه $A[i]$ را با $A[j]$ جابجا کرده و به مرحله 2 برمی‌گردیم.
- ب) $i >= j$ در نتیجه pivot را با $A[j]$ جابجا کرده گذر مورد نظر تمام شده است.



الگوریتم مرتب سازی سریع به شکل زیر خواهد بود.

```
procedure  Quicksort (var x : arraylist; Left , Right : integer);
    var
      i , j , pivot : integer;
        begin
          if  left < right  then
          begin
          i : = Left;  j : = Right + 1;  pivot : = x [left];
          repeat
              repeat

              i : = i + 1;

              until x [i] > = pivot;
              repeat
              j : = j - 1;
              until x [j] < = pivot;
          if  i < j  then  swap (x [i] , x [j]);
          until i > = j;
          swap (x [Left] , x [j]);
          Quicksort (x , Left , j - 1);
          Quicksort (x , j + 1 , Right);
          end;
        end;
```

مثال: فرض کنیم بخواهیم روش Quicksort را، در مورد آرایه زیر به کار ببریم. در اولین مرحله از کار شکل آرایه مطابق کدام گزینه خواهد بود: (کنکور آزاد ۸۳)

25 , 41 , 32 , 9 , 16 , 21

۱) 41 , 32 , 25 , 9 , 21 , 16

۲) 9 , 16 , 21 , 25 , 32 , 41

۳) 9 , 21 , 16 , 25 , 32 , 41

۴) 21 , 16 , 9 , 25 , 32 , 41

```
        i                 j
        ↓                 ↓
  25  , 41 , 32 , 9 , 16 , 21
  ↑
pivot
```

چون j > i است A [i] با A [j] جابجا می‌شود.

```
              i       j
              ↓       ↓
  25  , 21 , 32 , 9 , 16 , 41
  ↑
pivot
```

چون j > i است A [i] با A [j] جابجا می‌شود.

```
                    j   i
                    ↓   ↓
  25  , 21 , 16 , 9 , 32 , 41
  ↑
pivot
```

چون j = < i است A [j] با pivot و مرحله اول تمام می‌شود.

9 , 21 , 16 , 25 , 32 , 41

انتهای مرحله اول

گزینه ۳ صحیح می‌باشد.

بدترین وضعیت در مرتب سازی سریع (Quick sort)

بدترین وضعیت برای این الگوریتم زمانی است که داده به صورت مرتب یا مرتب معکوس وارد شوند چون هر بار که عنصر اول محور انتخاب می‌شود، بقیه داده‌ها یا سمت چپ و یا سمت راست آن قرار می‌گیرد.

در این وضعیت

۱ـ تعداد مقایسات $\frac{n(n-1)}{2}$

۲ـ مرتبه زمانی $O(n^2)$

مثال: لیست s که از $n = 6$ حرف تشکیل شده به صورت A , B , C , D , E , F می‌باشد مقایسه‌های لازم برای مرتب کردن s با الگوریتم Quick sort عبارتست از: (کنکور ۸۳)

۱) لیست مرتب و مقایسه‌ای نداریم.

۲) 36

۳) 15 ✓

۴) 10

باتوجه به مرتب بودن لیست بدترین وضعیت را داریم در نتیجه تعداد مقایسات

$$\frac{n(n-1)}{2} = \frac{6 \times 5}{2} = 15$$

بهترین وضعیت در مرتب سازی سریع (Quicksort)

بهترین وضعیت برای این الگوریتم زمانی است که داده به صورت نامرتب وارد شوند تا در هر بار در دو طرف عنصر لولا به طور متعادل داده‌ها توزیع شوند در این وضعیت زمان مرتب سازی $O(n \log_2 n)$ خواهد بود که زمان وضعیت متوسط نیز خواهد بود.

مثال: الگوریتم Quick sort یک رشته n تائی را با چه سرعتی مرتب می‌نماید؟ (کنکور ۸۳ آزاد)

۱) $O(n \log n)$ ✓

۲) $O(n)$

۳) $O(n^2)$

۴) $\log_2 n$

زمان هر الگوریتم مرتب سازی در صورت عدم بیان (بدترین یا بهترین حالت) مربوط به حالت متوسط آن می‌شود که در این مسئله نیز منظور زمان متوسط است که همان $O(n \log n)$ است.

مرتب سازی Heap (نامتعادل و درجا)

برای مرتب سازی یک آرایه n تائی به روش Heap به ترتیب باید 2 عمل انجام شود.

الف) درج تمام داده‌ها در یک درخت Heap تهی (maxHeap مرتب سازی نزولی ؛ minHeap مرتب سازی صعودی)

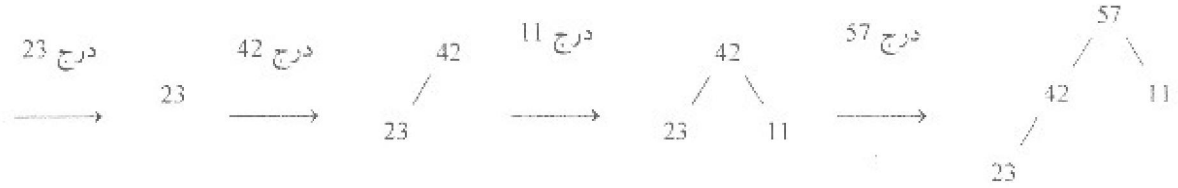
ب) حذف تمام داده‌ها از ریشه درخت Heap

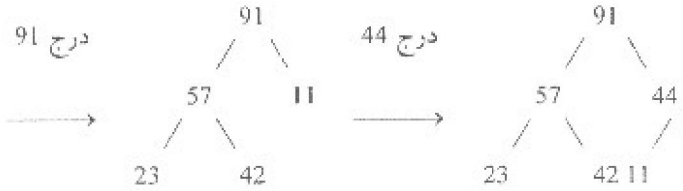
■ **زمان مرتب سازی Heap**

از آنجائیکه زمان درج یا حذف در Heap $O(\log n)$ می‌باشد، در نتیجه برای مرتب سازی یک آرایه n تائی زمان مورد نظر $O(n \log n)$ خواهد بود.

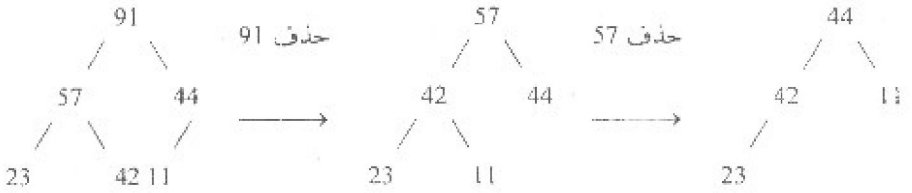
مثال: مرتب سازی داده‌های 44 ، 91 ، 57 ، 11 ، 42 ، 23 به شیوهٔ درخت Heap:

قسمت اول: درج داده‌ها در Heap





قسمت دوم: حذف داده‌ها از Heap



مجموعه سوالات کنکوری

پشته و صف

۱ ـ یک پشته خالی با اعداد از ۱ تا ۶ در ورودی داده شده است. اعمال زیر بر روی پشته قابل انجام هستند:

PUSH : کوچکترین عدد ورودی را برداشته و وارد پشته می کنیم.

POP : عنصر بالای پشته را در خروجی نوشته و سپس آن را حذف می کنیم.

کدامیک از گزینه‌های زیر را نمی‌توان با هیچ ترتیبی از اعمال فوق به دست آورد؟ (اعداد را از چپ به راست بخوانید.) (کارشناسی ارشد دولتی ۷۴)

۱) 123564 ۲) 324651 ۳) 432165 ۴) 215346

۲ ـ کدام گزینه درست است؟

۱) یک پشته و یا صف را فقط به‌کمک خانه‌های مجاور هم در حافظه می‌توان پیاده‌سازی کرد.

۲) یک روش مناسب برای پیاده‌سازی 2 پشته آن است که به سمت همدیگر رشد کنند.

۳) یک پشته همواره به دو اشاره‌گر برای مشخص کردن ابتدا و انتهایش نیاز دارد.

۴) یک صف را تنها به‌صورت حلقوی می‌توان پیاده‌سازی کرد.

۳ ـ کاراکترهای رشته ABC به‌ترتیب جهت پردازش وارد پشته‌ای می‌شوند، کدام خروجی نمی‌تواند پدید بیاید؟

۱) ABC ۲) CBA ۳) CAB ۴) ACB

۴ ـ برای محاسبه عبارت $(A+B)^*(A-B)$ ابتدا آن را به postfix تبدیل کرده و از پشته استفاده می‌کنیم. برای این کار پشته مورد نیاز حداقل چند خانه باید داشته باشد و چند بار عمل push در آن صورت می‌گیرد؟

۱) 4 خانه ـ 10 بار ۲) 5 خانه ـ 7 بار ۳) 4 خانه ـ 7 بار ۴) 5 خانه ـ 10 بار

۵ ـ برای تعیین ترتیب پردازش برنامه‌های کامپیوتری دسته‌ای (batch) که به مرکز محاسبات ارائه می‌شوند. کدام ساختار مناسب‌تر است؟(مسابقات آموزشکده‌های فنی ۷۷)

۱) صف ۲) پشته ۳) درخت ۴) آرایه

۶ ـ اگر رشته اعداد 1 ، 2 ، 3 ، 4 ، 5 را به‌ترتیب به یک stack وارد کنیم کدامیک از خروجی‌های زیر از این stack امکان‌پذیر خواهد بود(خروجی‌های پشته را از سمت چپ به راست بخوانید) (مسابقات آموزشکده‌های فنی ـ ۷۸) (کارشناسی ارشد ـ آزاد ۷۱)

۱) 4 - 1 - 5 - 3 - 2 ۲) 2 - 1 - 3 - 4 - 5 ۳) 2 - 4 - 5 - 3 - 1 ۴) 4 - 2 - 3 - 1 - 5

۷ ـ اگر رشته اعداد 1 ، 3 ، 4 ، 5 ، 7 را به‌ترتیب از سمت راست وارد یک stack کنیم کدامیک از خروجی‌های زیر از این stack امکان‌پذیر نیست.(خروجی‌ها را از سمت راست بخوانید) (کارشناسی ارشد ـ آزاد ۷۲)

۱) 13457 ۲) 45731 ۳) 45371 ۴) 57341

۸ـ push , pop کردن به‌ترتیب به چه معناست؟ (مسابقات آموزشکده‌ها ـ ۷۶)

۱) برداشتن عنصر بالایی پشته ـ گذاشتن عنصر جدید روی پشته
۲) خالی کردن پشته ـ پر کردن پشته
۳) دیدن عنصر بالایی پشته ـ برداشتن عنصر بالایی پشته
۴) گذاشتن عنصر در پشته ـ خواندن عنصر بالایی پشته

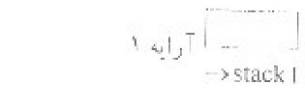
۹ـ لیستی که در آن آخرین عنصر وارده، اولین عنصر حذف شده ازآن می‌باشد چه نام دارد؟ (مسابقات آموزشکده‌ها ـ ۷۶) (مشابه مسابقات آموزشکده‌ها ـ ۸۰)

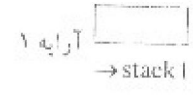
۱) پشته
۲) درخت دودوئی
۳) صف
۴) لیست پیوندی حلقوی

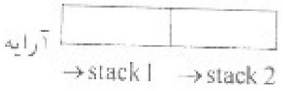
۱۰ـ اگر یک صف با آرایه نمایش داده شود، مشکل حرکت تدریجی صف در آرایه پیش می‌آید برای حل آن باید: (مسابقات آموزشکده‌ها ـ ۷۶)

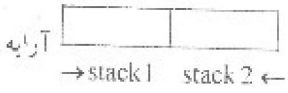
۱) از صف حلقوی استفاده می‌کنیم.
۲) از لیست پیوندی استفاده می‌کنیم.
۳) با تابعی پرشدن آرایه بررسی شود.
۴) عناصر داخل صف به ابتدای آرایه انتقال داده شوند.

۱۱ـ کدام روش برای نمایش دو پشته در یک برنامه با استفاده از آرایه مناسب‌تر است؟ (مسابقات آموزشکده‌ها ـ ۷۶)

۱)


۲)


۳)


۴)


۱۲ـ مناسب‌ترین ساختار داده جهت ثبت آدرس محل بازگشت در موقع فراخوانی زیر برنامه‌ها کدام است؟ (مسابقات آموزشکده‌ها ۸۰)

۱) صف (Queue)
۲) پشته (Stack)
۳) درخت (Tree)
۴) آرایه (Array)

۱۳ـ عبارت $\{x+(y-[a+b]*c-[(d+e)])\}/(j-(k-[L-n]))$ را در نظر بگیرید. می‌خواهیم با استفاده از یک پشته بررسی کنیم که آیا پرانتز، کروشه و آکولادها بدرستی تطبیق می‌شوند یا نه. پشته مورد استفاده حداقل بایستی گنجایش چند عنصر را داشته باشد؟ (مسابقات آموزشکده‌ها ـ ۷۹)

۱) 4
۲) 9
۳) 18
۴) 27

۱۴ـ اگر داده ورودی به‌ترتیب از چپ به راست A , B , C , D , E , F باشد کدام پشته زیر می‌تواند یک حالت ایجاد شده باشد؟

۱)

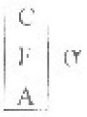

۲)


۳)


۴)


۱۵ ـ اگر داده ورودی به‌ترتیب از چپ به راست A ، B ، C ، D ، E ، F باشد کدام پشته نمی‌تواند یک حالت ایجادشده باشد؟

۱) 

۲) 

۳) 

۴) 

۱۶ ـ در push کردن به یک stack:

۱) ابتدا Top یک واحد اضافه می‌شود و بعد داده وارد پشته می‌شود.

۲) ابتدا Top یک واحد کم می‌شود و بعد داده وارد پشته می‌شود.

۳) داده وارد می‌شود، بعد Top یک واحد اضافه می‌شود.

۴) داده وارد می‌شود، بعد Top یک واحد کم می‌شود.

۱۷ ـ کدام عبارت صحیح است؟

۱) در عمل POP عبارت Top = 0 به معنای Stack Full است.

۲) در عمل POP عبارت Top = n به معنای Stack Empty است.

۳) در عمل PUSH عبارت Top = n به معنای Stack Empty است.

۴) در عمل PUSH عبارت Top = n به معنای Stack Full است.

۱۸ ـ کدامیک از تعاریف ذیل برای صف (Queue) درست است؟ (مسابقات آموزشکده‌های فنی ۷۸)

۱) یک لیست منظم که همهٔ اضافه‌شدنی‌ها یا حذف‌شدنی‌ها از یک‌طرف انجام می‌شود.

۲) یک لیست منظم که همهٔ اضافه‌شدنی‌ها یا حذف از دوطرف انجام می‌شود.

۳) یک لیست منظم که همهٔ اضافه‌شدنی‌ها از انتهای لیست و همهٔ حذف‌شدنی‌ها از ابتدای لیست صورت می‌گیرد.

۴) یک لیست منظم که دارای سه اشاره‌گر است.

۱۹ ـ دوصف در آرایه Q(n) طوری ذخیره شده‌اند که سر صف اولی در ابتدا و سرصف دومی در انتهای Q قراردارد. انتهای صف‌ها به‌ترتیب در متغیرهای R1 ، R2 قراردارند. زیر برنامه زیر چه عملی انجام می‌دهد؟

```
proc f (i , y);
if i=1 then R1 = R1+1 else R2 = R2-1;
if R1 = R2 then callfull;
case i of
1 : q(R1) = y;
2 : q(R2) = y;
end f;
```

۱) از برخورد دوصف جلوگیری می‌کند.

۲) عنصر y را به صف مشخص‌شده اضافه می‌کند.

۳) هردو

۴) هیچکدام

۲۰ ـ اگر یک صف در یک آرایه به طول N به‌صورت حلقوی تعریف شده باشد و Front ابتدای صف و Rear انتهای صف را نشان دهد می‌توان گفت:

۱) اگر $Front \le Rear$ تعداد عناصر برابر $Rear - Front + 1$ است.

۲) اگر $Front \le Rear$ تعداد عناصر برابر $N - Front + Rear$ است.

۳) اگر $Rear < Front$ تعداد عناصر برابر $Front - Rear - 1$ است.

۴) اگر $Rear < Front$ تعداد عناصر برابر $N - Front - Rear$ است.

۲۱ ـ کدام گزینه در ساختار یک صف حلقوی n عنصری، بیان‌کنندهٔ خالی و پربودن صف می‌باشد؟ (کارشناسی ناپیوسته ـ آزاد۷۹)

۱) خالی: $Front = 0 , Rear = 1$ و پر $Front = 0 , rear = n - 1$

۲) خالی: $Front = 0 , Rear = 0$ و پر $Front = 0 , rear = n - 1$

۳) خالی: $Front = n , Rear = n + 1$ و پر $Front = n - 1 , rear = n$

۴) خالی: $Front = 0 , Rear = 0$ و پر $Front = 0 , rear = n$

۲۲ ـ در ساختمان MAZE زمان اجرای مسیر طراحی شده (راهیابی) چگونه تعیین می‌شود؟ (کارشناسی ناپیوسته ـ آزاد۷۹)

۱) توسط اندازه مسیر پرپیچ‌وخم ۲) توسط طول کل مسیر

۳) فقط توسط سرعت کامپیوتر ۴) توسط الگوریتم

۲۳ ـ معادل میانوندی عبارت پسوندی ab + c * de / - چیست؟ (مسابقات آموزشکده‌ها ـ ۷۶)

۱) $a-(b*d)/(c+e)$ ۲) $a-b*c/(d-e)$ ۳) $a-(b/c)*(d+e)$ ۴) $a*(b+c)-d/e$

۲۴ ـ اگر $a = 2 , b = 4 , c = 8 , d = 10$ باشد، ارزش عبارت پسوندی ab * c - da - / چیست؟ (مسابقات آموزشکده‌ها ـ ۷۹)

۱) -2 ۲) -1 ۳) 1 ۴) 2

۲۵ ـ معادل postfix عبارت ریاضی infix روبرو کدام است؟ $x + y / (u - t * w) * y$ (مسابقات آموزشکده‌ها ـ ۷۹)

۱) xyutwy * - * / + ۲) xyutw * - / y * + ۳) utw * - xy / + y * ۴) tw * u - y / x + y *

۲۶ ـ عبارت پسوندی (Postfix) معادل عبارت ریاضی a / b - c + d * e - a * c / d کدام است؟ (با توجه به اولویت عملگرها) (کارشناسی ناپیوسته ـ آزاد۷۹)

۱) ab / c - de * + ac * d / - ۲) ab / c - de * ac * - d / ۳) ab / c - d * e + a - c * d / ۴) abc / - d + e * a - c * d /

۲۷ ـ عبارت پیشوندی (prefix) معادل عبارت ریاضی a / b - c + d * e - a * c کدام است؟ (با توجه به اولویت عملگرها) (کارشناسی ناپیوسته ـ آزاد۷۹)

۱) + - / abcd * e - ac * ۲) / ab - cd + ea - c * ۳) - + - / abc * de * ac ۴) + - / abcde * - ac *

۲۸ ـ معادل پسوندی $(A - B / (C * D \hat{} E))$ کدام است؟ (مسابقات آموزشکده‌های فنی ـ ۸۰)

۱) ABCD * E ^ - / ۲) ABCDE ^ * / - ۳) AB - CD * E ^ / ۴) AB / CDE ^ * -

۲۹ ـ مزایای عبارت پسوندی یا پیشوندی نسبت به عبارت میانوندی چیست؟

۱) اولویت‌ها مطرح نیستند.
۲) نیاز به پرانتزگذاری ندارد.
۳) گزینه ۱ و ۲
۴) هیچکدام

۳۰ ـ معادل پسوندی (postfix) عبارت میانوندی (infix) زیر کدام است؟ (مسابقات آموزشکده‌های فنی ـ ۷۷) $(A/B\hat{}C)*D+E$

۱) $AB\hat{}C/DE*+$
۲) $ABC/\hat{}DE*+$
۳) $ABC\hat{}/D*E+$
۴) $AB/C\hat{}DE*+$

۳۱ ـ معادل پیشوندی (prefix) عبارت میانوندی (infix) زیر چیست؟

$(A+B)*(C-D)$

۱) $*+ABCD-$
۲) $*+AB-DC$
۳) $*AB+-CD$
۴) $*+AB-CD$

۳۲ ـ معادل پیشوندی عبارت پسوندی زیر چیست؟

$AB\hat{}C*D-EF/GH+/+$

۱) $+-*\hat{}ABCD//EF+GH$
۲) $+-AB*\hat{}CD//EF+GH$
۳) $+--*\hat{}ABCDEF//+GH$
۴) $+-*ABC\hat{}//EF+GH$

۳۳ ـ معادل پسوندی عبارت پیشوندی زیر چیست؟

$-A/B*C\hat{}DE$

۱) $ABC\hat{}DE*/-$
۲) $ABCD\hat{}E*/-$
۳) $ABCDE\hat{}*/-$
۴) $A\hat{}BCDE*/-$

۳۴ ـ معادل پسوندی عبارت پیشوندی زیر چیست؟

$\hat{}-*+ABC-DE+FG$

۱) $AB+C*DE-F-G+\hat{}$
۲) $ABC+*DE--FG*\hat{}$
۳) $AB+C*DE--FG+\hat{}$
۴) $AB+C*DE-\ F+\hat{}G$

۳۵ ـ در رابطه با پشته‌های چندگانه کدام گزینه نادرست است؟ (t[i] انتها و b[i] ابتدای پشته i را نشان می‌دهد.)

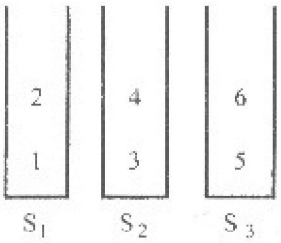
۱) شرط پربودن پشته iام آن است که $t[i]=b[i+1]$ باشد.

۲) شرط خالی بودن پشته iام آن است که $t[i]-b[i]$ باشد.

۳) مقادیر اولیه عبارتند از: $b[i]=t[i+1]=\frac{m}{n}*(i+1)-1$. (m تعداد خانه‌های کل آرایه و n تعداد پشته‌ها)

۴) ممکن است پشته‌ای پر شود در حالیکه پشته‌های دیگر خالی باشند.

۳۶ ـ سه پشته s1 و s2 و s3 هر یک حاوی دو عدد به شکل زیر می‌باشند:

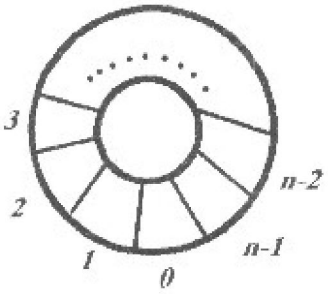


دو عملگر poppush (i ,j) و pop(i) به صورت زیر تعریف شده‌اند.

poppush(i,j) یک قلم از پشته s_j حذف و به پشته s_i اضافه می‌کند. pop(i) یک قلم از پشته s_i حذف و سپس آن را چاپ می‌کند. برای چاپ اعداد 1 تا 6 به صورت 6,4,2,5,3,1 عملگر poppush بایستی حداقل چندبار مورد استفاده قرار گیرد؟ اولین عددی که چاپ می‌شود عدد 1، دومین عدد، عدد 3 و ... می‌باشد.(کارشناسی ارشد ـ آزاد۸۰)

۱) 3 بار ۲) 5 بار ۳) 6 بار ۴) 4 بار

۳۷ ـ کدامیک از فرمول‌های زیر، N، تعداد اقلام در یک صف دایره‌ای را محاسبه می‌کند؟ متغیر F به خانه‌ای که بلافاصله قبل از جلوی صف قرار دارد، (در جهت عکس عقربه‌های ساعت) اشاره می‌کند و متغیر R به عقب صف اشاره می‌نماید.



۱) N=R-F

۲) N= n-(R-F)

۳) $N = \begin{cases} n-(F-R) & \text{if } F > R \\ R-F & \text{if } R > F \end{cases}$

۴) $N = \begin{cases} n-(R-F) & \text{if } F < R \\ R-F & \text{if } R > F \end{cases}$

۳۸ ـ عبارت infix زیر را به polish وارونه تبدیل کنید.(کارشناسی ارشد ـ آزاد۷۸)

A*(B-D)/E-F*(G+H/K)

۱) ABD-*E/FGHK/-*-

۲) A + B * DEF / GHK / * -

۳) -*+/KHG/FED*B-A

۴) AB*+DEF/G+HK/*

۳۹ ـ عبارت postfix زیر مساوی است با: + * + 5 1 2 / - 3 7 12 (کارشناسی ارشد ـ آزاد۷۶)

۱) 8 ۲) 15

۳) 0 ۴) هیچکدام

۴۰ ـ عبارت infix زیر معادل با کدامیک از عبارات postfix است؟ (کارشناسی ارشد ـ آزاد۷۶)

((A+B)*D)↑(E-F)

۱) AB+D*EF-↑

۲) ABD+*EF-↑

۳) +AB*D-↑EF

۴) هیچکدام

۴۱ ـ می‌خواهیم عبارت میانوندی (infix) زیر را به کمک پشته به عبارت پسوندی (postfix) تبدیل کنیم: a+b*(c/(d+e))*f برای این کار چندبار باید پردازه PUSH را برای گذاشتن اجزاء این عبارت در پشته، فراخوانی کنیم؟ (مسابقات آموزشکده‌های فنی ـ ۷۷)

۱) 15 ۲) 5 ۳) 9 ۴) 7

۴۲ ـ پشته (stack) ساختمان داده‌ای است از نوع: (مسابقات آموزشکده‌های فنی ـ ۷۸)

۱) IFOF ۲) FILO ۳) FIFO ۴) هیچکدام

۴۳ ـ برای پیاده‌سازی اعداد صحیح بسیار بزرگ (30 رقمی و یا بیشتر) کدام ساختار مناسب‌تر است؟ (به عنوان مثال: 2^n که n، عدد نسبتاً بزرگی باشد.) (مسابقات آموزشکده‌های فنی ـ ۸۰)

۱) Stack ۲) Queue ۳) Long integer ۴) Linked List

۴۴ ـ کدامیک از اصطلاحات زیر یک صف یا queue را توصیف می‌نماید: (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) LIFO ۲) LILO ۳) FIFO ۴) هیچکدام

۴۵ ـ کم‌هزینه‌ترین (از نظر تخصیص حافظه) برای این‌که عناصر یک پشته (stack) s1 را به پشته دیگر s2 بدون این‌که ترتیب عناصر تغییر یابند، انتقال دهیم کدام است؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) از طریق یک متغیر

۲) از طریق یک پشته (stack) اضافی

۳) از طریق دوپشته (stack) اضافی ۴) از طریق چند متغیر

۴۶ ـ کم‌هزینه‌ترین (از نظر تخصیص حافظه) راه برای این‌که ترتیب عناصر یک stack را برعکس کنیم کدام است؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) از طریق 2 پشته (stack) اضافه ۲) از طریق یک صف اضافی (queue)

۳) از طریق یک پشته اضافی و چندین متغیر ۴) هیچکدام

۴۷ ـ کدام مورد جزو کاربردهای پشته نمی‌باشد؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) ارزشیابی عبارات ریاضی پسوندی (postfix)

۲) فراخوانی زیر برنامه‌ها در یک برنامه

۳) مدیریت درخواست‌های چاپ برای چاپگر

۴) مسأله مسیر پر پیچ و خم (Maze)

۴۸ ـ معادل پسوندی (postfix) عبارت ریاضی $a+b\otimes c/(d-a\otimes(f+b)+b)$ کدام است؟ فرض کنید اولویت + و ـ یکسان و کمتر از اولویت ⊗ و اولویت ⊗ کمتر از / باشد.(کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) $abcdafb+b\otimes-+/\otimes+$ ۲) $abcdafb+b\otimes-/\otimes+$

۳) $abcdafb-+\otimes-b/\otimes-$ ۴) $abcdafb-\otimes-b+/\otimes+$

۴۹ ـ در نمایش صف حلقوی به کمک آرایه، چرا از یک خانه استفاده نمی‌شود؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) اندیس خانه مزبور صفر است.

۲) به عنوان رزرو برای مواقع خاص نگه داشته می‌شود.

۳) برای ارتباط خانه آخر با خانه اول آرایه باید از یک خانه استفاده کنیم.

۴) در صورت استفاده، پروخالی بودن صف با یکدیگر اشتباه می‌شود.

۵۰ ـ معادل میانوندی (infix) عبارت پیشوندی (prefix) روبرو چیست؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

/ * + abc-ab

۱) (a-b)*c/a-b ۲) (a+b)*c/(a-b) ۳) (a+b)*(c/a-b) ۴) a+b*c/a-b

۵۱ ـ اعداد 1 تا 6 به ترتیب وارد stack می‌شوند، کدامیک از گزینه‌ها را نمی‌توان در خروجی نمایش داد؟

۱) 1 2 3 5 6 4 ۲) 3 2 4 6 5 1 ۳) 4 3 2 1 6 5 ۴) 2 1 5 3 6 4

۵۲ ـ اگر رشته اعداد 7 ، 5 ، 4 ، 3 ، 1 را به ترتیب وارد stack کنیم کدام خروجی درست نخواهد بود؟

۱) 1 3 4 7 5 ۲) 4 5 7 3 1 ۳) 4 5 3 7 1 ۴) 5 7 3 4 1

۵۳ ـ کدام گزینه در ساختار یک صف حلقوی با n=7 بیان کننده خالی و پر بودن صف می‌باشد؟

۱) خالی: R=7 و F=0 ؛ پر: R=6 و F=0

۲) خالی: R=2 و F=2 ؛ پر: R=6 و F=0

۳) خالی: R=0 و F=0 ؛ پر: R=6 و F=7

۴) خالی: R=0 و F=0 ؛ پر: R=7 و F=6

۵۴ ـ معادل پسوندی عبارت روبرو چیست؟

+ - * ^ ABCD / EF + GH

۱) AB^CD*E-FG/H+/+ ۲) A^B*CDEF-/G+H/+

۳) A^BCD*E-F/+GH/+ ۴) AB^C*D-EF/GH-/+

۵۵ ـ در کدام عملیات ابتدا خواندن و یا نوشتن صورت می‌گیرد و سپس اشاره‌گر تغییر می‌کند؟

۱) Push داخل Stack ۲) pop از Stack

۳) خواندن از صف حلقوی ۴) نوشتن در صف حلقوی

۵۶ـ نتیجه عبارت postfix روبرو چیست؟

8 2 / 5 - 7 3 * +

۱) 28.6 ۲) -28 ۳) 20 ۴) 8.2

۵۷ـ کدام گزینه معرف حذف کردن از صف است؟

۱) front: = front mod n ۲) front : = (front +1) mod n

۳) rear: = rear mod n ۴) rear:= (rear+1) mod n

۵۸ـ اگر a=1 و b=2 و c=3 و d=4 باشد، آنگاه ارزش عبارت پسوندی ab*c+dc-/ کدام است؟

۱) 7 ۲) 6 ۳) 5 ۴) 4

۵۹ـ یک پشته خالی و یک صف با محتویات 3 ، 2 ، 1 (عدد 1 ابتدای صف است) داریم. پس از انجام سری دستورات زیر از چپ به راست محتوای صف چه می‌شود؟

delq(A), push(A), delq(A), push(A), delq(A), push(A), pop(A), addq(A), pop(A), addq(A)

۱) 3 ، 1 ،2 ۲) 3 ، 2 ،1 ۳) 2 ، 1 ۴) 3 ، 2

۶۰ـ اگر به جای پشته s1، از دو پشته s1، s2 و عملیات push و pop آن‌ها استفاده کنیم، کدامیک از پشته‌های زیر با داده‌های ورودی A , B , C , D , E , F امکان‌پذیر است؟

۱) همه پشته‌های ممکن با A , B , C , D , E , F ۲) فقط همان پشته‌هایی که به تنهایی با s1 امکان‌پذیر است.

۳) دو برابر تعداد پشته‌هایی که به تنهایی با s1 امکان‌پذیر است. ۴) نصف همه پشته‌های ممکن با A , B , C , D , E , F

۶۱ـ اگر یک صف دوار با گنجایش n عنصر و اشاره‌گرهای head و tail داشته باشیم، حداکثر چه تعداد عنصر می‌توان داخل این صف قرار داد؟

۱) n-1 ۲) n ۳) n+1 ۴) n+2

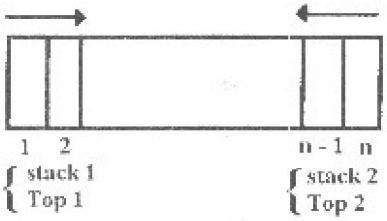
۶۲ـ در هنگام محاسبه عبارت میانوندی (5+3)-4 کدامیک از پشته‌های زیر غیرممکن است؟

۱) $\begin{vmatrix} \\ 4 \end{vmatrix}$ ۲) $\begin{vmatrix} 8 \\ - \\ 4 \end{vmatrix}$ ۳) $\begin{vmatrix} 5 \\ - \\ 1 \end{vmatrix}$ ۴) هر سه امکان‌پذیرند.

۶۳ـ مینیمم تعداد متغیرهای میانی در محاسبه عبارت جبری ab+cd*/a+ که به صورت postfix است، برابر است با ... (کارشناسی ارشد ـ علوم کامپیوتر ۷۹)

۱) 1 ۲) 2 ۳) 3 ۴) 4

۶۴ـ دو پشته در یک آرایه به اندازه (1..n) پیاده‌سازی شده‌اند. اگر Top یکی به خانه خالی و Top دیگری به خانه پر اشاره کند، شرط پر بودن بردار کدام گزینه می‌باشد؟



۱) Top1 = Top2-1

۲) Top1 = Top2

۳) Top1>Top2

۴) هیچ کدام

۶۵ـ عبارت postfix معادل عبارت $(A+B)*D+E/(F+A*D)$ برابر است با ..... (کارشناسی ارشد ـ علوم کامپیوتر ۸۰)

۱) AB+D*E-F/A+D*

۲) AB+D*EF AD*+/+

۳) ABDEFAD-*+/-*

۴) AB+DE+FAD*+/

۶۶ـ عبارت prefix زیر داده شده است: (کارشناسی ارشد ـ آزاد ۷۹)

+-*↑ABCD/E/F+GH

کدامیک از عبارات زیر معادل infix برای این عبارت است؟

۱) $A^B*(C-D)+\frac{EF}{G}+H$

۲) $A^{B*(C-D)}+E(F/(G+H))$

۳) $A^B*C-D+E/F/G+H$

۴) $A^B*C-D+E/(F/(G+H))$

۶۷ـ چنانچه بخواهیم k تا صف را در یک بردار پیاده‌سازی کنیم برای کدامیک از مقادیر k هزینه پیاده‌سازی شده همواره Q(1) خواهد بود؟

۱) تنها k=1

۲) $k\le 2$

۳) $k\le n$ و n طول بردار

۴) $k\le\frac{n}{2}$ و n طول بردار

۶۸ـ عبارت پیشوندی (prefix) زیر داده شده است: (کارشناسی ارشد ـ دولتی ۷۷)

++a/b-cd/-ab--+c*d5/a-bc

معادل پسوندی (postfix) آن کدام است؟

۱) abcd-/+ab-cd5*+abc--/-/+

۲) ab+cd-/ab-cd-/5*+ab/c---

۳) ab+cd-ab--/cd5*+/abc--/-

۴) abcd/-+abc-d5*abc-+/-/+

۶۹ـ عبارت +ABC*D/ در نماد لهستانی وارونه معادل کدام عبارت ریاضی زیر است؟

۱) A*D+B/C

۲) C*D/B+A

۳) C*B/D+A

۴) D*C/A+B

۷۰ـ عبارت prefix معادل عبارت میانوندی زیر کدام است؟

A*B^(C+D^-E*F)/G-H*K

۱) -/*A^B+C*^D-EFG*HK

۲) -*^AB C+^DEF*/G-*HK

۳) -/*^ABC+*D^-EFG*HK

۴) ^^*ABC+-EF*/G-HK*

۷۱ـ با توجه به صف حلقوی زیر چنانچه عناصر AA، BB و CC و DD به ترتیب از سر چپ آن حذف و عناصر FF، GG و HH و JJ به ترتیب به سر راست آن اضافه شوند مقدار Left و Right به ترتیب از راست به چپ چه خواهد بود؟ (مسابقات آموزشکده‌ها-۸۱)

Left : 5					AA	BB	CC	DD	EE	
Right : 9	1	2	3	4	5	6	7	8	9	10

۱) 4 – 8 ۲) 3 – 8 ۳) 3 – 9 ۴) 10 – 9

۷۲ـ برای تشخیص صحت عبارات ریاضی، از لحاظ علامتهای) }] کدامیک از ساختمان‌های زیر را ترجیح می‌دهید: (مثل عبارت $\{[(3+7)*2]/5\}$) (مسابقات آموزشکده‌های فنی ۸۱)

۱) آرایه ۲) پشته ۳) صف ۴) لیست پیوندی

۷۳ـ عبارت پیشوندی (Prefix) روبرو داده شده است ++a/b–cd/–ab–+c*d5/a–bc معادل پسوندی آن کدام است؟

۱) ab+cd–/ab–cd+/*ab/c–+ ۲) ab+cd–ab–+/cd5*+/abc–/–

۳) abcd–/+abc–d5*abc–+/–/+ ۴) abcd–/+ab–cd5*+abc–/–/+

۷۴ـ معادل پیشوندی (Prefix) عبارت ریاضی $a+b/((d+c)*(a-c))$ کدام است؟ (کارشناسی ناپیوسته دلتی ۸۱)

۱) +a/b*+de–ac ۲) +a/b+de*–ac

۳) +a/b*+–deac ۴) +a/b+de–*ac

۷۵ـ حداقل اندازه پشته برای تبدیل عبارات میانوندی $a/(b-c*(d+e))$ به معادل پسوندی چقدر است؟ (کارشناسی ناپیوسته آزاد ۸۱)

۱) 7 ۲) 5 ۳) 4 ۴) 6

۷۶ـ شرط خالی بودن پشته‌ای که با آرایه stack[1..Max] نمایش داده شده است، کدام است؟ (کارشناسی ناپیوسته آزاد ۸۱)

۱) Top = Max ۲) Top = Max – 1 ۳) Top = –1 ۴) Top = 0

۷۷ـ کدام گزینه رابطه $\sqrt{b^2-4ac}$ را به‌صورت پسوندی لهستانی نمایش می‌دهد؟ (علامت ^ نماینده توان است) (کارشناسی ناپیوسته علمی کاربردی ۸۱)

۱) ^ , – , ^ , b , 2 , * , – , 4 , a , c , / , 1 , 2

۲) b , 2 , ^ , 4 , a , *, c , * , – , 1 , 2 , / , ^

۳) b , 2 , ^ , 4 , a , c , * , * , – , 1 , / , 2 , ^

۴) b , 2 , ^ , 4 , a , c, * , * , – , 1 , 2 , ^ , /

۷۸ـ برای ارزیابی عبارت پسوندی (لهستانی) زیر با الگوریتمی که از یک پشته استفاده می‌کند چند بار عملیات Push رخ می‌دهد؟ (کارشناسی ناپیوسته علمی کاربردی ۸۱)

4 , 2 , 6 , / , * , 7 , 8 , + , –

۱) 8 ۲) 5 ۳) 4 ۴) 9

۷۹ـ اگر اعداد 1 ، 2 ، 3 به ترتیب قرار داشته باشند (3 بالاترین) و عملیات Pop (x) به معنای قرار گرفتن بالاترین عنصر پشته در متغیر x و عملیات push (x) به معنای ذخیره x در پشته باشد پس از انجام عملیات زیر (از راست به چپ) وضعیت پشته چگونه است؟ (کارشناسی ناپیوسته علمی کاربردی ۸۱)

$$Pop(x)\ ,\ Push(z)\ ,\ Push(x)\ ,\ Push(y)\ ,\ Pop(z)\ ,\ Push(x)\ ,\ Pop(y)\ ,\ Pop(x)$$

۱) 2 ، 1 ، 3　　۲) 1 ، 2 ، 3　　۳) 3 ، 2 ، 1　　۴) 2 ، 3 ، 1

۸۰ـ عناصر صف‌های Q_1 و Q_2 از چپ به راست به صورت زیر است (عنصر سمت چپ ابتدای صف است) : (کارشناسی ارشد ۷۹ دولتی)

$Q_1 = 10\ ,\ 25\ ,\ 17\ ,\ 41\ ,\ 19\ ,\ 26\ ,\ 75$

$Q_2 = 1\ ,\ 5\ ,\ 7\ ,\ 4\ ,\ 9\ ,\ 6$

اگر x ، y عناصر صف باشند، پس از اجرای قطعه برنامه زیر :

```
make NULL (Q3)
i = 0
While (not empty (Q1) and not empty (Q2))
i = i + 1
x = DeleteQ (Q1)
y = DeleteQ (Q2)
if (y = i) then Add Q (Q3 , x)
end while
```

محتوی صف Q_3 برابراست با :

۱) $Q_3 = 1,4,6$　　۲) $Q_3 = 10\ ,\ 25\ ,\ 17$　　۳) $Q_3 = 10\ ,\ 41\ ,\ 26$　　۴) $Q_3 = 1\ ,\ 5\ ,\ 7$

۸۱ـ معادل Postfix عبارت Prefix زیر کدام است؟ (مسابقات آموزشکده‌های فنی ۷۸)

$$++a/b-cd/-ab-+c*de/a-bc$$

۱) $ab+cd-/ab-cd-/e*+ab/c-+$　　۲) $ab+cd-ab-+/cde*+/abc-/-$

۳) $abcd-/-ab-cde*+abc-/-/+$　　۴) $abcd/-+abc-de*abc-+/-/+$

۸۲ـ عبارت میانوندی روبرو را به‌صورت نمادگذاری لهستانی تبدیل کنید؟ (مسابقات آموزشکده‌های فنی ـ ۸۲)

$$(A+B)/(C-D)$$

۱) $/-AB-CD$　　۲) $AB+/CD$ ـ　　۳) $(AB+)/(CD-)$　　۴) $(\ )/+AB(\ )-CD$

۸۳ـ هنگامی که به کمک آرایه یک صف حلقوی را نمایش می‌دهیم، چرا از یک خانه آرایه استفاده نمی‌شود؟ (کارشناسی ناپیوسته ـ دولتی ۸۲)

۱) برای اشتباه‌نشدن پروخالی بودن صف　　۲) برای ارتباط خانهٔ آخر با خانهٔ اول

۳) به عنوان آدرس استفاده می‌شود　　۴) در زبان برنامه‌نویسی خانه اول آرایه صفر است.

۸۴ـ حاصل Postfix عبارت زیر چیست؟ (کارشناسی ناپیوسته ـ دولتی ۸۲)

6,2,3,+,-,3,8,2,/,+,*,2,↑,3,+

۱) 7 ۲) 25 ۳) 49 ۴) 52

۸۵ـ شکل Postfix عبارت $(a - b \uparrow c \uparrow d) * (e + f / d)$ کدام است؟ (کارشناسی ناپیوسته دولتی ۸۲)

۱) abc ↑ d ↑ +efd/+* ۲) abcd ↑↑ +efd+*

۳) abcd ↑↑ + e/fd+* ۴) abcd ↑↑ +*e/fd-

۸۶ـ شکل Postfix عبارت زیر چیست؟ (کارشناسی ناپیوسته ـ آزاد ۸۲)

$A + (B * C - (D / E \uparrow f) * G) * H$

۱) ABC*DEFG↑/*-H*+ ۲) ABCD*EF↑/G*-H*-

۳) ABCDEF*↑G*-H*- ۴) ABC*DEF↑/G*-H*+

۸۷ـ فرض کنید به stack N = 6 خانه اختصاص داده‌ایم، در ابتدا Stack خالی است، خروجی برنامه زیر چیست؟ از چپ به راست (کارشناسی ناپیوسته آزاد ۸۲)

```
1/. Set A = 2, B = 5
2/. Push(Stack, A)
Push(Stack, B + 10)
Push (Stack,3)
Push (Stack, A-B)
3/. Repeat while Top < >0  POP (Stack, item)
      write : item
4/. Return
```

۱) -3,3,15,2 ۲) 2,15,3,-3

۳) 2,15,-3,3 ۴) 2,5,-3

۸۸ـ اگر رشته c,d,c,b,a را به‌ترتیب از سمت راست وارد Stack کنیم، کدامیک از خروجی‌های زیر از این Stack امکان‌پذیر نیست؟ خروجی‌ها از سمت راست خوانده می‌شود؟ (کارشناسی ناپیوسته ـ آزاد ۸۲)

۱) c,d,e,b,a ۲) c,d,b,e,a ۳) a,b,c,d,e ۴) d,e,b,c,a

۸۹ـ فرض کنید ساختار صف را با دو پشته پیاده‌سازی کرده‌ایم. اگر صف حاوی n عنصر باشد چند عمل Push برای درج یک عنصر به این صف لازم است؟ (کارشناسی ناپیوسته ـ علمی کاربردی ۸۲)

۱) 1 ۲) 2n ۳) 2n+1 ۴) n+1

۹۰ ـ درچه صورت دستور write در قطعه برنامه زیر اجرا می‌شود: فرض کنید n عدد خوانده شده متمایز باشد؟ (Inqueue روتین درج در صف و Outqueue روتین حذف از صف، Push درج در پشته و POP حذف از پشته است) (کارشناسی ناپیوسته ـ علمی کاربردی ۸۲)

```
for I . = 1 To n Do
Begin
    Read (x);
    Push (x) ; Inqueue (x);
End;
for I: = 1 To n Do
Begin
  A : = POP; B: = Outqueue:
  if (A = B ) then write ('fcund');
End;
```

۱) اعداد صعودی وارد شوند.
۲) اعداد نزولی وارد شوند.
۳) n فرد باشد.
۴) اعداد یک دنباله متقارن را تشکیل دهند.

لیست پیوندی

۹۱ ـ در یک لیست پیوندی حلقوی درستی شرط start^.link = start نمایانگر کدام است؟

۱) لیست هیچ عنصری ندارد و کاملاً خالی است.
۲) به آخرین عنصر لیست رسیده‌ایم.
✓ ۳) لیست فقط یک عنصر (سرلیست) دارد.
۴) لیست پر شده است.

۹۲ ـ تابع زیر بر روی یک لیست یک طرفه چه می کند؟

```
function T(L : pointer );
Var x : pointer;
Begin
    T:=0;
    if  L<> nil  then  Begin
        x:=L;
        while  x<>nil  do  Begin
            T:=T+1;
            x := x^.link;
        end;
    end;
end;
```

✓ ۱) تعداد گره‌های لیست را بر می گرداند.
۲) اشاره‌گر به گره آخر را بر می گرداند.
۳) لیست را معکوس می کند.
۴) محتویات داده گره‌ها را با هم جمع می کند.

۹۳ـ تابع زیر چه عملی انجام می‌دهد؟ توضیح این‌که list نشان دهنده لیستی از اعداد بوده و منظور از تابع Head، تابعی است که مقدار اولین عنصر در لیست را برمی‌گرداند و تابع tail لیستی حاوی همه عناصر لیست ورودی به استثنای اولین عنصر را برمی‌گرداند. مکان‌های لیست را از مکان ۱ در نظر بگیرید: (کارشناسی ارشد ـ دولتی ۷۶)

```
function what(L:List):integer;
begin
    if L=nil then
        what:=(0)
    else
        if Tail(L)≠nil then
        what:=(Head(L)+what(Tail(Tail(L))))
    else
        what:=Head(L)
end;
```

۱) تعداد عناصر لیست را برمی‌گرداند.

۲) مجموع عناصر در مکان‌های فرد لیست ورودی را برمی‌گرداند.

۳) تعداد عناصر در مکان‌های فرد لیست را برمی‌گرداند.

۴) مجموع عناصر لیست ورودی را برمی‌گرداند.

۹۴ـ روال زیر چه عملی انجام می‌دهد؟ (کارشناسی ارشد دولتی ۷۱)

```
Procedure f(x,y:ptr; var z:ptr);
var p:ptr;
begin
    p:=x; z:=x;
    while p↑.Link<>nil do
        p:=p↑.Link;
    p↑.Link:=y;
end;
```

۱) دولیست پیوندی را به هم وصل (Concat) می‌کند.

۲) دو لیست حلقوی (Circular) را به هم وصل می‌کند.

۳) دو لیست پیوندی که حداقل یکی از آن‌ها غیرتهی می‌باشد را به هم وصل می‌کند.

۴) دو لیست حلقوی غیرتهی را به هم وصل می‌کند.

۹۵ـ اگر p گرهی از یک لیست دو پیوندی (Doubly Linked List) با پیوندهای llink و rlink باشد، کدام گزینه همواره صحیح است؟

۱) P=P↑.rlink↑.llink

۲) P=P↑.llink↑.rlink

۳) هیچکدام ✓

۴) هر دو

۹۶ـ اگر x آدرس شروع یک لیست پیوندی زنجیری باشد؟ روال زیر چه عملی انجام می‌دهد؟

```
Proc S(x)
    Define P,Q,r:pointer;
    p=x;q=nil;
    while (p<>nil) do
    r=q; q=p;
    p=p↑.link;
    q↑.link=r;
    end;
    x=q;
end;
```

۱) لیست را به ترتیب صعودی مرتب می‌کند.

۲) آخرین گره لیست را بازگشت می‌دهد.

۳) لیست را وارون می‌کند.

۴) لیست را به ترتیب نزولی مرتب می‌کند.

۹۷ ـ مزیت لیست پیوندی نسبت به آرایه چیست؟ (مسابقات آموزشکده‌های فنی ـ ۸۱)

۱) مصرف حافظه کمتر

۲) ساده‌تر بودن عملیات حذف و درج

۳) سریعتر بودن عمل پیمایش

۴) سریعتر بودن عمل جستجو

۹۸ ـ برای حذف کردن یک عنصر از یک لیست پیوندی یک طرفه n عضوی چند دستور انتساب (جایگزینی) و چند دستور جابجایی لازم است؟ (مسابقات آموزشکده‌های فنی ۷۶)

۱) یک دستور انتساب و هیچ جابجایی

۲) یک دستور انتساب و n جابجایی

۳) $\frac{n}{2}$ دستور انتساب و جابجایی

۴) n دستور انتساب و $\frac{n}{2}$ جابجایی

۹۹ ـ برای امکان وجود پیمایش یک لیست در دو جهت از کدام مورد استفاده می‌شود؟ (مسابقات آموزشکده‌های فنی ـ ۷۶)

۱) تک پیوندی با گره آغازین

۲) لیست دوار تک پیوندی

۳) لیست پیوندی دوگانه

۴) دوار با گره آغازین

۱۰۰ ـ برای حذف کردن یک عنصر از یک لیست پیوندی دوطرفه چند آدرس باید جابگزین شود؟ (مسابقات آموزشکده‌های فنی ـ ۷۶)

(کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) 4 ۲) 3 ۳) 2 ۴) 1

۱۰۱ ـ برای حذف کردن یک عنصر از یک لیست یک طرفه چند آدرس باید جابگزین شود؟

۱) 1 ۲) 2 ۳) 3 ۴) 4

۱۰۲ ـ برای اضافه کردن یک گره به یک لیست پیوندی دوطرفه چند جابگزینی لازم است؟ (کارشناسی ناپیوسته ـ علمی کاربردی ۸۱)

۱) 1 ۲) 2 ۳) 3 ۴) 4

۱۰۳ ـ برای حذف کردن یک عنصر از یک لیست پیوندی یک طرفه n عضوی، چند دستور پیمایش و چند دستور جابگزینی لازم است؟

۱) یک دستور پیمایش و یک دستور جابگزینی

۲) یک دستور پیمایش و n دستور جابگزینی

۳) n دستور پیمایش و 1 دستور جابگزینی

۴) $\frac{n}{2}$ دستور پیمایش و 1 دستور جابگزینی

۱۰۴ ـ برای تعریف صحیح نوع لیست پیوندی، از کدام دستورات می‌توان استفاده کرد؟ (کارشناسی ناپیوسته ـ آزاد ۷۹)

۱) `typedef struct node *list; typedef struct node{char ch;list link;};`

۲) `typedef struct node list; struct node{char ch;list link;};`

۳) `struct node *list struct node{char ch;list link;};`

۴) `struct node{char ch;struct node link;};`

۱۰۵ ـ عبارت روبرو را در نظر بگیرید. طبق این عبارت کدام گزینه صحیح است؟ (کاشناسی ناپیوسته ـ آزاد ۷۹) $a \to b \, . \, c \to d$

۱) b ساختاری از نوع اشاره‌گر دارد.

۲) d و b هر دو از نوع اشاره‌گرند.

۳) c ساختاری از نوع اشاره‌گر دارد.

۴) از طریق دستور c.d می‌توان به عنصر d دسترسی داشت.

۱۰۶ ـ تابع زیر چه عملی انجام می‌دهد؟ (با فرض این که نوع list اشاره‌گر است) (کارشناسی ناپیوسته ـ آزاد ۷۹)

```
list x(list L)
{ list m,t;
  m=NULL;
  while(L){
  t=m; m=L;
  L=L.→link;
  m→link=t;}
  return m;}
```

۱) محل دو عنصر در لیست L را جابه‌جا می‌کند.

۲) لیست پیوندی L را معکوس می‌کند.

۳) عنصری را از لیست L جابه‌جا می‌کند.

۴) لیست L را مرور می‌کند.

۱۰۷ ـ مزیت لیست یک طرفه دوار (چرخشی) نسبت به لیست غیردوار یک طرفه چیست؟ (مسابقات آموزشکده‌های فنی ـ ۷۹)

۱) پیمایش لیست را سریعتر می‌کند.

۲) فیلد nil در این لیست وجود ندارد.

۳) بدون مصرف حافظه اضافی، حرکت به گره قبلی را میسر می‌سازد.

۴) با مصرف حافظه‌ای به اندازه یک فیلد پیوند، حرکت به گره قبلی را امکان‌پذیر می‌کند.

۱۰۸ ـ اگر first اشاره‌گر به ابتدای یک لیست یک طرفه باشد. اثر اجرای قطعه دستورات زیر چیست؟ (مسابقات آموزشکده‌های فنی ـ ۷۹)

(مشابه کارشناسی ناپیوسته ـ دولتی ۸۰) (مشابه مسابقات آموزشکده‌های فنی ۷۷ و ۸۱)

```
p:=first;
q:=nil;
while p<>nil do
begin
    r:=q;
    q:=p;
    p:=p^.link;
    q^.link=r;
end;
first:=q;
```

۱) لیست را sort می‌نماید.

۲) لیست را معکوس می‌نماید.

۳) لیست یک طرفه را به لیست دوار تبدیل می‌کند.

۴) لیست را با سه پوینتر پیمایش کرده و first را به گره‌آخر اشاره می‌دهد.

۱۰۹ ـ روال زیر یک گره با فیلد داده‌ای 10 را به بعد از گره‌ای که آدرس آن در x می‌باشد درج می‌کند. اگر جای دستورات (۱) و (۲) عوض شود کدام گزینه رخ می‌دهد؟ (مسابقات آموزشکده‌های فنی ـ ۷۹) (توضیح : first به ابتدای یک لیست پیوندی اشاره می‌کند.)

```
procedure insert(var first : pointer ; x : pointer );
var
  t:pointer ;
begin
  new(t);
  t^.data:=10;
  if first=nil then begin
     first:t;
  t^.link=nil;
  end
  else begin
(1)→t^.link:=x^.link;
(2)→x^.link:=t;
  end
end;
```

۱) t = nil می‌شود.

۲) first = nil می‌شود.

۳) خانه‌ای که x به آن اشاره می‌کند گم می‌شود.

۴) قسمتی از لیستی که first به آن اشاره می‌کند گم می‌شود.

۱۱۰ ـ زیر روال زیر چه عملی انجام می‌دهد؟ (مسابقات آموزشکده‌های فنی ـ ۷۶)

```
procedure f(x,y:pointer ;var z:pointer );
var
   p:pointer;
   begin
      p:=x;z:=x;
      while p^.link<>nil do
         p:=p^.link;
      p^.link:=y;
   end;
```

۱) دو لیست پیوندی y و x را به همین ترتیب به هم وصل می‌کند و لیست پیوندی z را بوجود می‌آورد.

✓ ۲) دو لیست پیوندی y و x را به همین ترتیب به هم وصل کرده و لیست پیوندی با نام z را درست می‌کند.

۳) دو لیست دایره‌ای x و y را به همین ترتیب وصل می‌کند و لیست z را بوجود می‌آورد.

۴) z را فقط معادل لیست پیوندی x در نظر می‌گیرد.

۱۱۱ ـ مهمترین مزیت لیست پیوندی نسبت به آرایه در نمایش یک لیست چیست؟ (مسابقات آموزشکده‌های فنی ـ ۷۶ و ۸۰)

۱) مصرف حافظه کمتر

۲) جستجوی عنصری از لیست

۳) پیمایش (Traverse) لیست

✓ ۴) سادگی عمل حذف و درج عنصر از لیست

۱۱۲ ـ مزیت لیست پیوندی نسبت به آرایه در نمایش یک لیست کدام است؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) پیمایش لیست

۲) جستجوی عنصری از لیست

۳) سادگی عمل حذف و درج عنصر از لیست

۴) مصرف حافظه کمتر

۱۱۳ ـ کدام گزینه جزو معایب روش پیوندی (linked) (لیستهای پیوندی) است؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) اتلاف حافظه

۲) بالابردن سرعت دسترسی به گره‌ها

۳) پیچیدگی انجام عمل حذف یا درج

۴) سادگی دسترسی به گره‌ها

۱۱۴ ـ کدام یک از عملیات زیر در لیست دو طرفه، ساده‌تر از لیست یک طرفه انجام می‌شود؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

۱) پیمایش لیست از ابتدا تا انتها

۲) حذف گره با آدرس p از لیست

۳) درج گره در انتهای لیست

۴) درج گره در ابتدای لیست

۱۱۵ ـ روش جستجو در یک لیست پیوندی یک طرفه با n گره که داده‌های آن به ترتیب صعودی مرتب شده‌اند چیست؟ (کارشناسی ناپیوسته ـ دولتی ۸۰) (مشابه کارشناسی ناپیوسته ـ آزاد ۸۱)

۱) ترتیبی

۲) دودونی

۳) ترتیبی یا دودونی

۴) ترتیبی و دودونی

۱۱۶ ـ کار قطعه برنامه زیر چیست؟ (start به ابتدای لیست اشاره می‌نماید) (کارشناسی ناپیوسته ـ دولتی ۸۰)

```
new(p)
p^.data:=data;
P^.link:=start;
start:=p;
```

۱) درج در انتهای لیست

۲) درج در صف پیوندی

۳) درج در پشته پیوندی

۴) درج در ابتدای یک لیست غیرتهی

۱۱۷ ـ شرط پایان در حلقه until قطعه برنامه پاسکال زیر که به منظور پیمایش یک لیست یک طرفه دوار بدون گره آغازین (Headernode) نوشته شده است چیست؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

```
p:=start;
if p<>nil then
    Repeat
        writeln(p^.data);
        p:=p^.link;
until  شرط ;
```

۱) P<>start

۲) p=start

۳) p^.link=start

۴) p^.link<>start

۱۱۸ ـ زیربرنامه g بر روی یک لیست پیوندی یک طرفه کدام عمل را انجام می‌دهد؟ (کارشناسی ناپیوسته ـ دولتی ۸۰)

```
Procedure g(Var start:Nodeptr);
Var
   p,q,r:Nodeptr;
Begin
   p:=start;
   q:=nil;
   while p<>nil Do
   Begin
      r:=q;
      q:=p;
      p:=p^.link;
      q^.link:=r;
   end;
   start:=q;
End;
```

۱) پیمایش لیست
۲) حذف کردن لیست
۳) معکوس کردن لیست ✓
۴) مرتب کردن لیست

۱۱۹ ـ کدامیک از زیر روال‌ها عنصر جدید را بعد از عنصر p درج می‌کند؟ (مسابقات آموزشکده‌های فنی ـ ۷۶)

```
Procedure insert list(Var x,p,head:pointer);
Var
  y,z:pointer;
Begin
  y:=head; z:=head;
  while y<>p do begin
     z:=y; y:=y^.link
  end;
  x^.link:=z^.link; z^.link:=x;
end;
```

(۱

```
Procedure insert list(Var x,p,head:pointer);
Var
  y:pointer;
Begin
  y:=head;
  while y<>p do
     y:=y^.link; x^.link:=y^.link; y^.link:=x;
  end;
```

(۲

```
Procedure insert list(x,p:pointer);
Begin
  x^.link:=p^.link;
  p^.link:=x;
end;
```

(۳

```
Procedure insert list(Var x,p:pointer);
Begin
  p^.link:=x;
  x^.link:=p^.link;
end;
```

(۴

۱۲۰ ـ کدام گزینه درباره لیست پیوندی دوطرفه درست می‌باشد؟

۱) حرکت از ابتدا به انتها و بالعکس به سادگی امکان‌پذیر است.

۲) برای حذف گره‌ای که آدرس آن را داریم نیاز به پیمایش لیست وجود ندارد.

۳) درج پس از گره دلخواه x در آن سریعتر از لیست یک طرفه است.

۴) ۲ و ۱ ✓

۱۲۱ ـ یک لیست خطی یک طرفه با دو اشاره‌گر F و R که به ترتیب به عنصر اول و آخر لیست اشاره می‌کنند پیاده‌سازی شده است. هزینه کدامیک از اعمال زیر وابسته به تعداد عناصر لیست است؟ (کارشناسی ارشد ـ دولتی ۸۰)

۱) حذف اولین عنصر

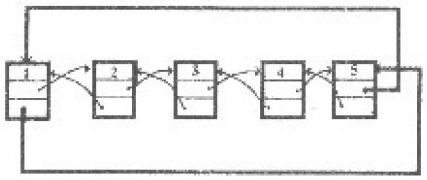
۲) حذف آخرین عنصر

۳) درج یک عنصر در انتهای لیست

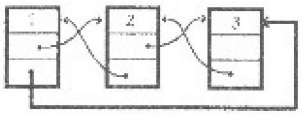
۴) درج یک عنصر در ابتدای لیست

۱۲۲ ـ کدامیک از گزینه‌های زیر لیست دو طرفه را از حالت A به حالت B تغییر می‌دهد؟ (کارشناسی ناپیوسته آزاد ۸۰)

لیست A



لیست B



۱) list → next → next → next = list → prev

۲) list → next → next → next → next = list → next

۳) list → prev = list → next → next

۴) list → next → prev → next = list → next → next

۱۲۳ ـ در لیست پیوندی حلقوی اغلب به جای ذخیره آدرس ابتدای لیست آدرس انتهای لیست ذخیره می‌شود مزیت این کار در چیست؟

۱) زمان حذف یک گره از اول لیست از مرتبه $o(n)$ می‌شود.

۲) زمان درج یک گره قبل از گره ابتدایی ساده می‌شود.

۳) زمان اضافه کردن و یا حذف یک گره از اول لیست از مرتبه $o(n)$ می‌شود.

۴) هر سه گزینه

۱۲۴ ـ در یک صف که به‌صورت یک لیست پیوندی ساخته شده است حذف یک عنصر مطابق با کدام انتخاب زیر انجام می‌گردد؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) حذف عنصر از انتهای لیست انجام گرفته و ارزش ذخیره شده در آن بازگردانده می‌گردد.

۲) حذف عنصر از انتهای لیست انجام گرفته و ارزش بازگردانده نمی‌شود.

۳) حذف عنصر از ابتدای لیست انجام گرفته و هیچ ارزشی نیز بازگردانده نمی‌شود.

۴) هیچ یک از حالات بالا درست نیست.

۱۲۵ ـ یک لیست پیوندی را چگونه می‌توانیم پیمایش کنیم؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) از انتها به ابتدای لیست.

۲) از طریق متغیر Tail در ساختمان داده‌های لیست پیوندی.

۳) از طریق متغیر Head و از ابتدای لیست به انتهای آن.

۴) همه حالات گفته شده در بالا

۱۲۶ ـ در کدامیک از انتخاب‌های زیر یک لیست پیوندی خالی است؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) متغیرهای Head ، Tail برابر باشند و مقدار آن‌ها چیزی به غیر NULL باشد.

۲) Head مقدار NULL داشته باشد.

۳) Tail مقدار NULL داشته باشد.

۴) متغیرهای Head و Tail برابر باشند و مقدار آن‌ها NULL باشد.

۱۲۷ ـ در لیست پیوندی دوطرفه (double link list) امکان پیمایش لیست چگونه است؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) از طرف Head به طرف Tail

۲) از طرف Tail به طرف Head

۳) انتخاب ۱ و ۲

۴) هیچکدام

۱۲۸ ـ برای اضافه نمودن یک گره (node) در یک لیست پیوندی در ابتدای لیست در کدامیک از انتخاب‌های زیر ارزش متغیر Head به کار برده می‌شود؟ (کارشناسی ناپیوسته ـ آزاد ۸۰)

۱) در ارزش‌دهی متغیر Tail

۲) در ارزش دهی قسمت اشاره‌گر در گره جدید

۳) در ارزش‌دهی استفاده نمی‌گردد.

۴) در ارزش‌دهی آخرین گره (node) در لیست پیوندی

۱۲۹ ـ کدامیک از کدهای زیر دو گره بعد از گره x را از یک لیست پیوندی خطی حذف می‌کند؟

۱) $link(x):=link(link(x));$

۲) $x:=link(link(x));$

۳) $link(link(x)):=x;$

۴) $link(x):=link(link(link(x)));$

۱۳۰ ـ صف اولویت‌دار (Priority Queue) شامل کدامیک از اعمال زیر می‌شود؟

۱) جستجو، درج، حذف کوچکترین عضو

۲) درج، حذف کوچکترین عنصر

۳) جستجو، درج، حذف

۴) درج، حذف

۱۳۱ ـ تابع head در برنامه زیر مصرف برگرداندن اولین عنصر لیست و تابع tail مصرف مابقی list به غیر از اولین عنصر را می‌رساند و همچنین تابع Cons(x , y) به معنی ترکیب دو لیست x و y به صورت x.y می‌باشد زیر برنامه زیر چه عملی انجام می‌دهد؟

```
Fun (s:list)
begin
  if (S=nil) then return(nil);
  else return(cons(fun(tail(s)).head(s)));
end.
```

۱) معکوس سازی ترتیب عناصر list

۲) جابجایی عنصر اول و آخر لیست

۳) عمل خاصی انجام نمی‌دهد.

۴) حذف عناصر اول و آخر لیست

۱۳۲ ـ پردازه f چه عملی روی لیست یک طرفه انجام می‌دهد؟ (کارشناسی ناپیوسته ـ دولتی ۸۱)

```
procedure f(Var start:Nodeptr);
Begin
  if start<>nil then
    if start^.link=nil then
    begin
      Dispose(start);
      start:=nil;
    End
    Else
      f(start^.link)
End;
```

۱) حذف اولین گره

۲) حذف دومین گره

۳) حذف آخرین گره

۴) حذف گره ما قبل

۱۳۳ ـ کدام گزینه صحیح است؟ (کارشناسی ناپیوسته ـ دولتی ۸۱)

۱) اتلاف حافظه و ایستا بودن لیست‌های پیوندی از معایب آن محسوب می‌شود.

۲) اتلاف حافظه در آرایه به مراتب از لیست‌های پیوندی بیشتر است.

۳) دسترسی به عناصر لیست پیوندی نسبت به آرایه سریع‌تر انجام می‌شود.

۴) زمان دسترسی به عنصر اول و عنصر آخر آرایه با یکدیگر یکسان است.

۱۳۴ ـ کدام گزینه از مزایای صف پیوندی نسبت به صف ایجاد شده در آرایه است؟ (کارشناسی ناپیوسته ـ آزاد ۸۱)

۱) اختصاص حافظه به صورت پویا و متناسب با داده‌ها

۲) سرعت زیاد در دسترسی به داده سرصف

۳) امکان دسترسی به داده سر و انتهای صف

۴) جلوگیری از اتلاف حافظه

۱۳۵ ـ مزیت لیست یک طرفه دوار نسبت به لیست یک طرفه غیردوار کدام است؟ (کارشناسی ناپیوسته ـ آزاد ۸۱)

۱) سرعت دسترسی به گره وسط بیشتر است.

۲) اتلاف حافظه کمتری دارد.

۳) امکان دسترسی به گره قبل وجود دارد.

۴) مزیتی ندارد.

۱۳۶ ـ کدام گزینه نادرست است؟ (کارشناسی ناپیوسته ـ آزاد ۸۱)

۱) اتلاف حافظه در لیست دو طرفه بیشتر از لیست یک طرفه است.

۲) لیستی که درج و حذف در ابتدای آن صورت می‌گیرد پشته است.

۳) لیستی که درج از یک طرف و حذف از طرف دیگر آن صورت می‌گیرد صف است.

✓۴) پیمایش لیست دو طرفه سریعتر از لیست یک طرفه انجام می‌شود.

۱۳۷ ـ یک لیست پیوندی یک طرفه در آرایه‌های موازی info و link پیاده‌سازی شده است و start به شروع لیست اشاره می‌کند. قطعه برنامه زیر چه کاری روی لیست انجام می‌دهد؟ (کارشناسی ناپیوسته ـ علمی کاربردی ۸۱)

```
ptr:=start;
while link[ptr]≠NULL
info[ptr]:=info[ptr]+1
ptr:=link[ptr]
end while
```

۱) داده‌های هر گره را با گره بعدی عوض می‌کند به‌جز گره آخر که تعویض نمی‌کند.

۲) به داده‌های هر گره یک واحد اضافه می‌کند.

۳) به داده‌های هر گره به جز گره آخر یک واحد اضافه می‌کند.

۴) داده‌های هر گره را با گره ماقبل عوض می‌کند.

۱۳۸ ـ در لیست پیوندی یک طرفه کدامیک از الگوریتم‌های زیر را نمی‌توان مورد استفاده قرار داد؟ (کارشناسی ناپیوسته ـ علمی کاربردی ۸۱)

۱) مرتب‌سازی حبابی ۲) جستجوی خطی ۳) جستجوی دودویی ۴) الگوریتم بازگشتی

درخت

۱۳۹ ـ در یک درخت دودوئی کامل با n گره، برای هر گره با اندیس i داریم: (آزاد۷۹)

۱) اگر $i <> 1$ باشد، i ریشه است و پدری نخواهد داشت.

۲) اگر $2i > n$ باشد آن‌گاه i فرزند راست ندارد.

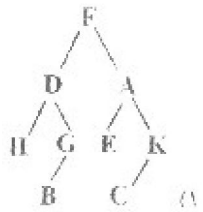
۳) اگر $i <> 1$ باشد آن‌گاه پدر i در $[i/2]$ است.

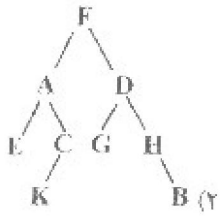
۴) اگر $2i - 1 > n$ باشد آن‌گاه i فرزند چپ ندارد.

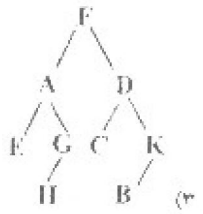
۱۴۰ ـ درخت دودوئی که پیمایش‌های LVR و VLR آن به‌صورت زیر است کدام است؟ (دولتی ۸۰)

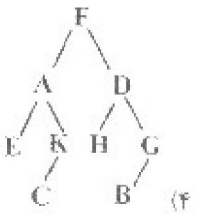
LVR : E, A, C, K, F, H, D, B, G

VLR : F, A, E, K, C, D, H, G, B

۱)


۲)


۳)


۴)


۱۴۱ ـ کار تابع H به روی یک درخت دودوئی چیست؟ (دولتی ۸۰)

```
procedure h (root: Treeptr; var  i: integer);
var
   x , y: Integer;
Begin
  if  root = nill then
     i : = 0
Else Begin
    h(root ^.Left, x);
    h(root ^.Right, y);
    IF x > y then
        i : = x + 1
   else
        i : = y + 1
   End
End;
```

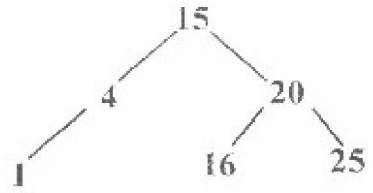
۱) شمارش تعداد گره‌های درخت

۲) شمارش تعداد گره‌های سطح آخر

۳) شمارش تعداد سطوح درخت

۴) شمارش تعداد شاخه‌های درخت

۱۴۲ ـ کدامیک از انتخاب‌های زیر درخت روبرو را به‌صورت Inorder پیمایش نموده است؟ (آزاد ۸۰)



۱) 1,4,15,16,20,25
۲) 25,20,16,15,1,4
۳) 15,1,4,20,16,25
۴) 4,1,15,16,20,25

۱۴۳ ـ چند روش پیمایش در درخت‌های دودویی به‌صورت depth – first وجود دارد؟ (آزاد ۸۰)

۱) 4 ۲) 8 ۳) 6 ۴) 16

۱۴۴ ـ در روش Preorder کدامیک از انتخاب‌ها درست است؟ (آزاد ۸۰)

۱) گره را ویزیت کنیم بعد زیر درخت راست را پیمایش کنیم سپس زیر درخت چپ را پیمایش کنیم.

۲) گره را ویزیت کنیم بعد زیر درخت چپ را پیمایش کنیم سپس زیر درخت راست را پیمایش کنیم.

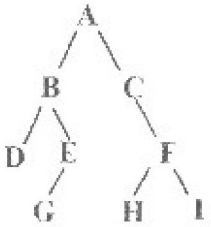
۳) زیر درخت چپ را پیمایش کنیم بعد زیر درخت راست را پیمایش کنیم سپس گره را ویزیت کنیم.

۴) زیر درخت راست را پیمایش کنیم بعد زیر درخت چپ را پیمایش کنیم سپس گره را ویزیت کنیم.

۱۴۵ ـ عمق یک درخت کامل با 1000 گره کدام است؟ (دولتی ۸۱)

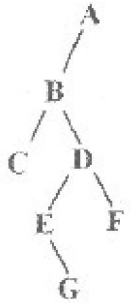
۱) 9 ۲) 10 ۳) 11 ۴) 1000

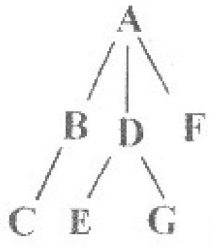
۱۴۶ ـ پیمایش RLV درخت روبرو کدام است؟ (دولتی ۸۱)

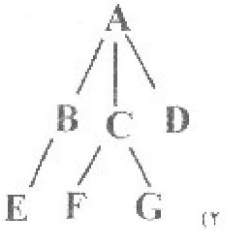


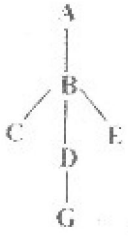
۱) A,B,D,E,G,C,F,H,I
۲) A,C,B,F,E,D,I,H,G
۳) I,H,F,C,E,G,B,D,A
۴) I,H,F,C,G,E,D,B,A

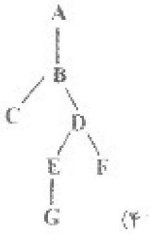
۱۴۷ ـ کدام درخت با استفاده از الگوریتم فرزند چپ ـ برادر راست به درخت دودویی شکل روبرو تبدیل می‌شود؟ (دولتی ۸۱)



۱)


۲)


۳)


۴)


۱۴۸ ـ درخت دودوئی پر، کدام است؟ (دولتی ۸۱)

۱) درخت دودوئی به عمق h که دارای 2^{h-1} گره است.

۲) درخت دودوئی به عمق h که دارای $2^h - 1$ گره است.

۳) درخت دودوئی که حداقل گره‌های سطح آخر آن پر باشد.

۴) درخت دودوئی که فقط گره‌های سطح آخر آن پر باشد.

۱۴۹ ـ در کدام پیمایش می‌توان با استفاده از دستور Dispose به جای دستور Writeln ، تمام گره‌های یک درخت دودوئی را حذف کرد؟ (دولتی ۸۱)

۱) LRV

۲) LVR

۳) VLR

۴) هر سه گزینه

۱۵۰ ـ یک درخت دودوئی در سه آرایه موازی به‌نام‌های info , left , Right پیاده‌سازی شده به‌طوری که info به اطلاعات هر گره و left , Right اشاره‌گر به فرزندان راست و چپ آن می‌باشند و Root به ریشه درخت اشاره می‌کند. زیربرنامه بازگشتی زیر چه کاری روی درخت فوق انجام می‌دهد؟(علمی کاربردی ۸۱)

```
A (left , Right , Root , s)
if Root = NULL then S : = 0 & Return
call A (left, Right , left [Root], s1]
call A (left , Right , Right [Root] , s2]
S : = S1 + S2 + 1
Return
```

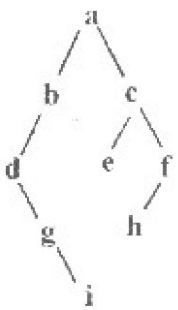
۱) تعداد برگ‌های درخت را پیدا می‌کند.

۲) عمق درخت را پیدا می‌کند.

۳) سطح درخت را پیدا می‌کند.

۴) تعداد گره‌های درخت را پیدا می‌کند.

۱۵۱ ـ کدام گزینه پیمایش Postorder درخت روبروست؟ (علمی کاربردی ۸۱)



۱) dgibehfca

۲) igdbehfca

۳) dgibaechf

۴) abdgicefh

۱۵۲ ـ روش پیمایش درخت دودوئی (Binary tree) زیر را که به صورت بازگشتی تعریف می‌شود درنظر بگیرید:

۱) ریشه را ملاقات (Visit) می‌کنیم.

۲) درخت فرعی سمت راست را پیمایش می‌کنیم.

۳) درخت فرعی سمت چپ را پیمایش می‌کنیم.

بین گره‌های ملاقات شده توسط روش فوق‌الذکر و postorder , Inorder , preorder چه رابطه‌ای وجوددارد؟ (۷۶)

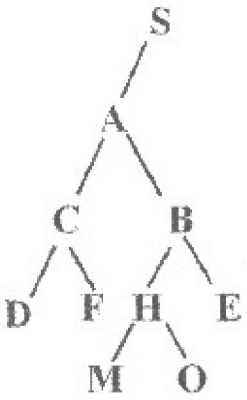
۱) گره‌های ملاقات‌شده معکوس گره‌های ملاقات شده توسط preorder است.

۲) گره‌های ملاقات شده معکوس گره‌های ملاقات شده توسط Inorder است.

۳) گره‌های ملاقات شده معکوس گره‌های ملاقات شده توسط postorder است.

۴) هیچ ربطی بین گره‌های ملاقات شده و گره‌های ملاقات شده توسط روش‌های کلاسیک فوق نیست.

۱۵۳ ـ نتیجه پیمایش درخت زیر به روش LNR چیست؟ (۷۹ و ۸۰)



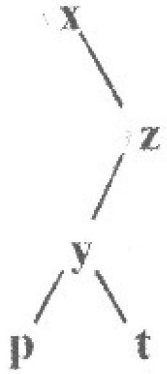
۱) DCFAMHOBES

۲) DFCMOHEBSA

۳) SACDFBHMOE

۴) SABEHOMCFD

۱۵۴ ـ پیاده‌سازی درخت زیر با آرایه به کدام صورت صحیح است؟ (۷۹)



۱)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x		z			y						p	t		

۲)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	z	y	p	t										

۳)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		t	p						y			x		z

۴)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x		z	y	p	t									

۱۵۵ ـ کدام گزینه نادرست است؟ (۷۶)

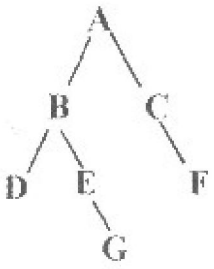
۱) بیشترین تعداد گره‌ها در یک درخت دودوئی به عمق k، $2^k - 1, k >= 1$ است

۲) بیشترین تعداد گره‌ها روی سطح iام یک درخت دودوئی 2^{i-1} است $(i>=1)$

۳) در هیچ درخت عادی گره صفر وجود ندارد.

۴) در هیچ درخت دودوئی گره صفر وجود ندارد.

۱۵۶ ـ پیمایش ABDEGCF کدامیک از گزینه‌های زیر است؟ (۷۶)



۱) LRD

۲) DLR

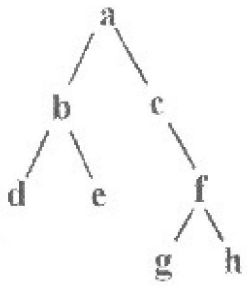
۳) LDR

۴) RDL

۱۵۷ ـ در یک درخت دودوئی کامل با 5 سطح حداکثر چند گره وجود دارد؟ (سطح ریشه برابر یک فرض شود) (۸۰)

۱) 15 ۲) 16 ۳) 31 ۴) 32

۱۵۸ ـ پیمایش درخت زیر به روش postorder کدام است؟ (۷۷)



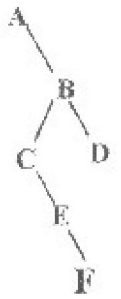
۱) dbeacgfh

۲) acfhgbed

۳) abdecfgh

۴) debghfca

۱۵۹ ـ پیمایش ABCEFD کدام یک از گزینه‌های زیر است؟ (۷۷)



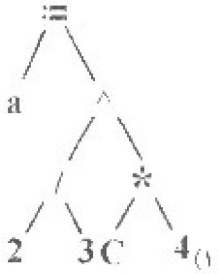
۱) LDR

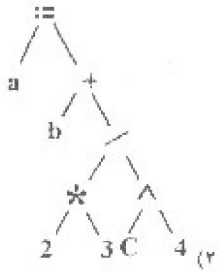
۲) DLR

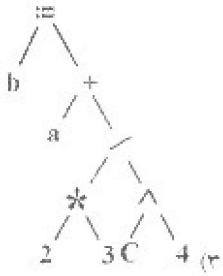
۳) DRL

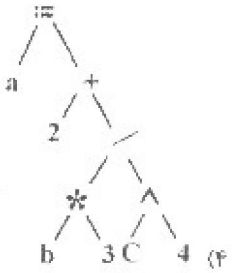
۴) RDL

۱۶۰ ـ عبارت a + b/*23^c4 := پیمایش preorder کدامیک از درخت‌های زیر است؟ (۷۸)

۱)


۲)


۳)


۴)


۱۶۱ ـ درخت دودویی T به‌صورت آرایه‌ای نمایش داده شده است. پیمایش inorder (LDR) درخت کدام است؟ (۷۸)

a	b	c		d					e
1	2	3	4	5	6	7	8	9	10

۱) bedac ۲) bdaec ۳) badec ۴) baecd

۱۶۲ ـ در نمایش و ارزیابی عبارت‌های ریاضی با استفاده از درخت‌های دودویی کدام یک از روش‌های طی کردن درخت فرم طبیعی عبارت را بدست می‌دهند؟ (۷۸)

۱) postorder ۲) preorder ۳) family order ۴) inorder LVR

۱۶۳ ـ درجه یک گره (node) در درخت شامل چیست؟ (۷۸)

۱) تعداد فیلدها ۲) تعداد انشعاب‌ها ۳) تعداد ریشه‌ها ۴) تعداد گره‌ها

۱۶۴ ـ کدام گزینه نادرست است؟ (۷۷)

۱) در هر درخت تعداد یال‌ها یکی کمتر از تعداد رأس‌هاست.

۲) یک درخت، یک گراف همبند بدون دور است.

۳) در هر درخت هر دو رأس با یک مسیر منحصر به فرد به هم متصل هستند.

۴) در هر درخت تعداد یال‌ها یکی بیشتر از تعداد رأس‌هاست.

۱۶۵ ـ بیشترین تعداد گره‌ها در یک درخت دودویی (Binary tree) با عمق h برابر کدام است؟ (۷۷)

۱) 2^{h-1} ۲) $2^h + 1$ ۳) $2^h - 1$ ۴) 2^{h-1}

۱۶۶ ـ پردازه زیر را برای پیمایش درخت دودویی نوشته‌ایم: (۷۷)

```
procedure trav (r , tree);
begin
   if r< > nil then begin
      trav(r^.left );
      writeln(r^.info );
      trav(r^.right);
   end;
end;
```

کدامیک از عبارت‌های زیر درست است؟

۱) اگر جای سطرهای 4 ، 5 پردازه r، عوض کنیم پردازه برای پیمایش پس‌ترتیبی به کار می‌رود.

۲) این پردازه برای پیمایش پس‌ترتیبی (Postorder) به کار می‌رود.

۳) این پردازه برای پیمایش میان‌ترتیبی (inorder) به کار می‌رود.

۴) اگر جای سطرهای 5 ، 6 پردازه را عوض کنیم پردازه برای پیمایش پیش‌ترتیبی (preorder) به کار می‌رود.

۱۶۷ ـ کدام گزینه نادرست است؟ (۷۶)

۱) تعداد زیردرخت‌های یک گره درجه آن گره نامیده می‌شود.

۲) تعداد زیردرخت‌های یک گره درجه آن درخت نامیده می‌شود.

۳) فرزندان یک گره، گره‌های همزاد یا هم‌نیا نامیده می‌شوند.

۴) یک جنگل شامل n درخت مجزا است $(n \geq 0)$

۱۶۸ ـ به یک درخت دودویی T یک درخت توسعه‌یافته گفته می‌شود اگر : (۸۱)

۱) دارای دو گره ریشه باشد.

۲) دارای 4 گره ریشه باشد.

۳) هر گره آن دارای صفر یا دو فرزند باشد.

۴) هر گره آن دارای 4 فرزند باشد.

۱۶۹ ـ به فرض داشتن یک درخت دودویی کامل با N گره، برای هر گره با اندیس i به‌گونه‌ای که $1 \leq i \leq N$ باشد کدام گزینه نادرست است؟ (۸۱)

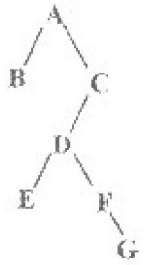
۱) اگر $i \neq 1$ آنگاه فرزند i در $i / 2$ است.

۲) اگر $2i \leq N$ باشد آنگاه فرزند چپ i در $2i$ است.

۳) اگر $2i > N$ باشد آنگاه i فرزند چپ ندارد.

۴) اگر $2i+1 \leq N$ باشد آنگاه فرزند راست i در $2i+1$ است.

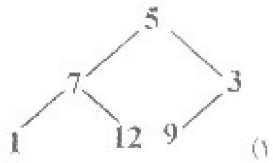
۱۷۰ ـ پیمایش ABCDEFG کدامیک از گزینه‌های زیر است؟ (۸۱)

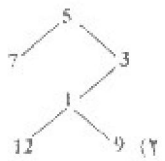


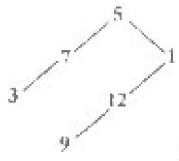
۱) DLR (NLR)
۲) LDR (LNR)
۳) LRD (LRN)
۴) RDL (RNL)

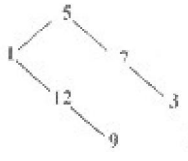
۱۷۱ ـ اگر نمایش ترتیبی ذخیره‌سازی یک درخت دودویی به شکل زیر باشد، آن درخت کدام است؟ (مسابقات آموزشکده‌های فنی ۸۲)

5	7	3			1						12	9	
1	2	3	4	5	6	7	8	9	10	11	12	13	14

۱)


۲)


۳)


۴)


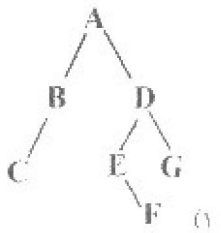
۱۷۲ ـ اگر آدرس گره درخت دودویی به‌وسیله t مشخص شود، الگوریتم بازگشتی زیر نشان‌دهنده کدام پیمایش است؟ LPTR اشاره‌گر چپ و RPTR اشاره‌گر سمت راست هستند. (دولتی ۸۲)

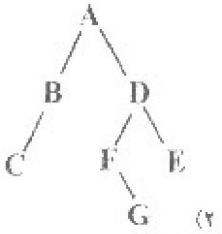
```
Recursive (t)
IF t = NULL then 'empty tree' return
IF LPTR (t) ≠NULL then Recursive (LPTR (t))
IF RPTR (t) ≠ NULL then Recursive (RPTR(t))
Write (DATA(t))
Return
```

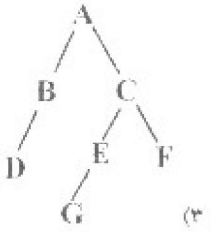
۱) infix ۲) inorder ۳) preorder ۴) postorder

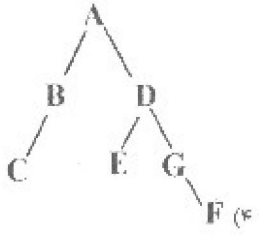
۱۷۳ ـ کدام درخت مربوط به پیمایش روبرو است؟ (دولتی ۸۲)

preorder :ABCDEFG
Inorder : CBAEFDG

۱)


۲)


۳)


۴)


۱۷۴ ـ کدامیک از مجموعه‌های زیر نمایش‌دهندهٔ یک درخت است؟ (آزاد۸۲)

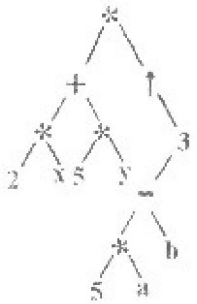
۱) $A=\{(0,1),(0,3),(3,2),(3,3),(4,2)\}$

۲) $A=\{(0,2),(0,3),(2,1),(2,3)\}$

۳) $A=\{(0,2),(0,3),(3,1),(3,4)\}$

۴) $A=\{(0,1),(0,3),(0,4),(4,1)\}$

۱۷۵ ـ شکل زیر نمایش کدام گزینه است؟ (آزاد۸۲)



۱) $(2x+5y)(5a-b)$

۲) $(2x+5y)(b-5a)^3$

۳) $(2x+5y)(5a-b)^3$

۴) $(5a-b)(2x+5y)^3$

۱۷۶ ـ لیست مربوط به گره‌های درخت دودویی T، درنظر بگیرید. root (ریشه درخت) left (ریشه چپ) و Right (ریشه راست) است. پیمایش inorder درخت T عبارت است از: (آزاد۸۲)

		INFO	left	right
root →	1	A	2	3
	2	B	4	5
	3	C	0	6
	4	D	0	0
	5	E	7	8
	6	F	9	0
	7	G	0	0
	8	H	0	0
	9	O	0	0

۱) D B G E H A C O F

۲) A B D E G H C F O

۳) D B E G H A C F O

۴) O F C A D G H E B

۱۷۷ ـ اگر پیمایش بین ترتیب (Inorder) یک درخت دودویی پر (FULL) به‌صورت زیر باشد، پیمایش پیش‌ترتیب آن کدام است؟ (علمی کاربردی۸۲)

پیمایش بین ترتیب = dbeagcf

۱) abdecgf ۲) gfedeba ۳) adbegcf ۴) این پیمایش منحصربه‌فرد نیست.

۱۷۸ ـ کدامیک از موارد از خواص «درخت جستجوی دودویی» نیست؟ (آزاد۷۹)

۱) کلیدهای واقع در زیر درخت غیرتهی چپ باید کمتر از مقدار واقع در ریشه زیردرخت راست باشد.

۲) هر عضو دارای یک کلید است و کلیدها منحصربه‌فرد نیستند.

۳) زیر درختان چپ و راست نیز خود، درختان جستجوی دودویی می‌باشند.

۴) کلیدهای زیر درخت غیر تهی راست، باید بزرگتر از مقدار کلید ریشه زیردرخت چپ باشد.

۱۷۹ ـ کدامیک از مجموعه‌های زیر نمایش‌دهندهٔ یک درخت است؟ (آزاد۷۹)

۱) $E(G)=\{(0,1)(0,2)(1,3)(1,4)(2,5)\}$

۲) $E(G)=\{(0,1)(0,2)(0,3)(1,2)\}$

۳) $E(G)=\{(0,1)(0,3)(0,2)(2,0)\}$

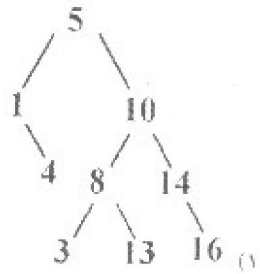
۴) $E(G)=\{(0,1)(0,2)(4,0)(4,3)(4,2)\}$

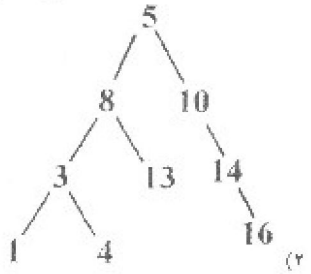
۱۸۰ ـ چنانچه بخواهیم داده‌های تکراری را از لیستی حذف کنیم، از کدام ساختار داده‌ای برای لیست مزبور استفاده می‌کنیم؟ (دولتی ۸۰)

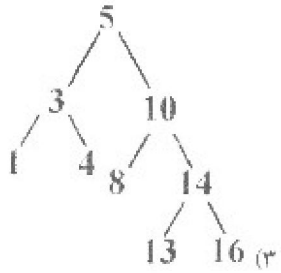
۱) درخت جستجوی دودویی ۲) درخت Heap ۳) پشته ۴) صف

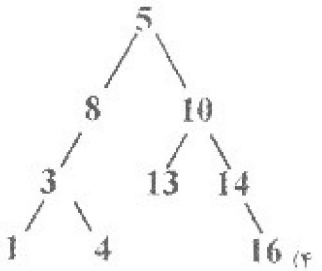
۱۸۱ ـ درخت جستجوی دودویی، درختی است که در آن اولاً برای هر گره از درخت، تمام گره‌های زیر درخت سمت چپ آن کوچک تر یا مساوی با گره ریشه و تمام گره‌های زیر درخت سمت راست آن بزرگ‌تر از گره ریشه باشند و ثانیاً تمام گره‌های زیردرخت سمت چپ کوچک‌تر از تمام گره‌های زیر درخت راست باشند. فرض کنید اعداد زیر به‌ترتیب از سمت چپ وارد یک درخت جستجوی دودویی شوند. درخت حاصل کدام است؟(مسابقات آموزشکده‌های فنی۷۸)

5 10 - 8 - 3 – 14 – 1 - 13 - 4 – 16

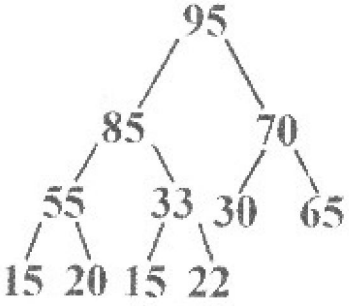
۱)


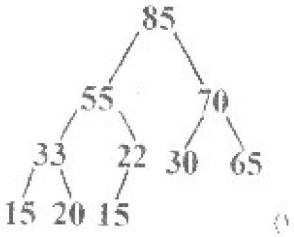
۲)


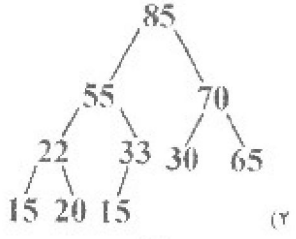
۳)


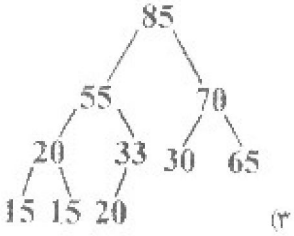
۴)


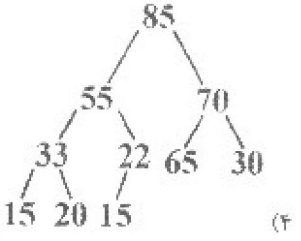
۱۸۲ ـ باتوجه به درخت (Max Heap) زیر چنانچه ریشه آن (95) حذف شود شکل جدیدش کدام است؟ (مسابقات آموزشکده‌های فنی ۸۱)



۱)


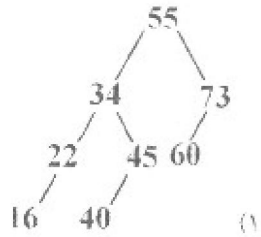
۲)


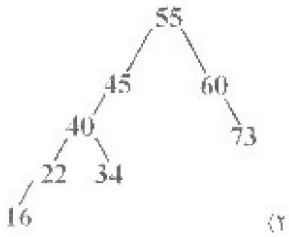
۳)


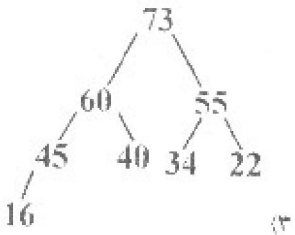
۴)


۱۸۳ ـ فرض کنید اعداد زیر به‌ترتیب در یک درخت جستجوی دودویی خالی درج شوند. درخت نهایی کدام است؟ (مسابقات آموزشکده‌های فنی ۸۱)

55 , 73 , 34 , 45 , 22 , 16 , 60 , 40

۱)


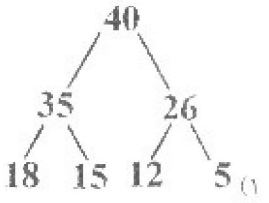
۲)


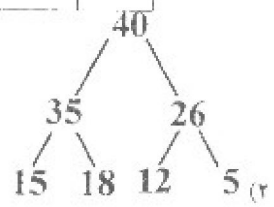
۳)


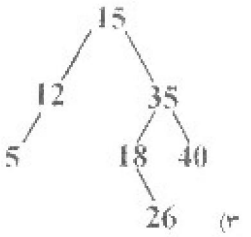
۴) هیچکدام

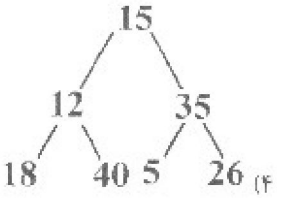
۱۸۴ ـ فرض کنید آرایه زیر را با روش heap sort می‌خواهیم مرتب کنیم پس از اجرای مرحله اول الگوریتم، درخت heap حاصل شده کدام است؟ (مسابقات آموزشکده‌های فنی ۸۱)

15	35	12	18	40	26	5

۱)


۲)


۳)


۴)


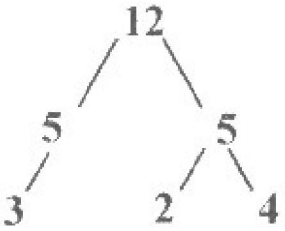
۱۸۵ ـ حداکثر تعداد مقایسه برای یافتن کلیدی در یک درخت جستجوی دودویی با n گره کدام است؟ (دولتی ۸۱)

۱) به اندازهٔ عمق درخت
۲) به اندازهٔ تعداد گره‌های سطح آخر
۳) $n \log \frac{n}{2}$
۴) $\frac{n}{2}$

۱۸۶ ـ روش متعارف نمایش درخت Max Heap چگونه است؟ (دولتی ۸۱)

۱) صف پیوندی
۲) پشته پیوندی
۳) آرایه
۴) لیست مجاورتی

۱۸۷ ـ درخت روبه‌رو چه نوع درختی است؟ (دولتی ۸۱)



۱) کامل

۲) جستجوی دودویی

۳) max – tree

۴) Max – Heap

۱۸۸ ـ کاربرد درخت Heap کدام است؟ (دولتی ۸۱)

۱) جستجوی سریع

۲) صف و پشته

۳) مرتب‌کردن داده‌ها

۴) مرتب‌کردن داده‌ها ـ صف اولویت‌دار

۱۸۹ ـ گزینه غلط کدام است؟ (دولتی ۸۱)

۱) درخت‌پر، درخت کامل نیز هست.

۲) درخت دودویی می‌تواند تهی باشد.

۳) درخت Max-tree که کامل نیز باشد، Max-Heap است.

۴) گراف نوع خاصی از درخت است.

۱۹۰ ـ کدام درخت را نمی‌توان با آرایه نشان داد؟ (دولتی ۸۱)

۱) اریب

۲) پر

۳) کامل

۴) Max - Heap

۱۹۱ ـ برای نمایش کدام درخت، از روش پیوندی استفاده می‌شود؟ (آزاد ۸۱)

۱) Max Heap

۲) Min tree

۳) درخت کامل (complete)

۴) درخت پر (FULL)

۱۹۲ ـ کدام گزینه صحیح است؟ (دولتی ۸۱)

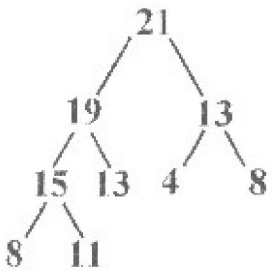
۱) درخت min Heap، درخت کامل است.

۲) درخت min tree، درخت پر است.

۳) درخت اریب، یک درخت min tree است.

۴) درختی که min tree نباشد، max tree است.

۱۹۳ ـ اگر یک گره از درخت max Heap زیر حذف شود، داده آخرین سطح درخت چیست؟ (آزاد ۸۱)



۱) 11

۲) 15

۳) 8

۴) 21

۱۹۴ ـ کاربرد درخت جستجوی دودویی چیست؟ (آزاد ۸۱)

۱) حذف داده‌های تکراری از یک لیست

۲) پیمایش VLR آن سبب مرتب‌شدن داده‌ها می‌شود.

۳) اگر داده‌ها به ترتیب صعودی در آن وارد شوند، برای عمل جستجو بسیار مناسب است.

۴) اگر داده‌ها به‌ترتیب نزولی در آن وارد شوند، برای عمل جستجو بسیار مناسب است.

۱۹۵ ـ کدام گزینه نادرست است؟ (دولتی ۸۰)

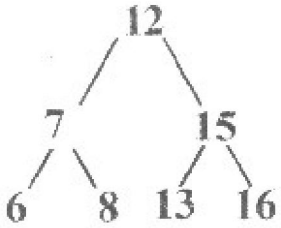
۱) درخت min heap درخت min tree است که کامل نیز باشد.

۲) روش نمایش درخت کامل، آرایه است.

۳) تعداد گره‌های سطح i ($i \geq 1$) درخت کامل (complete tree) 2^{i-1} گره است.

۴) از روش پیوندی برای نمایش درخت اریب استفاده می‌شود.

۱۹۶ ـ درخت جستجوی دودویی روبرو نشان‌دهندهٔ کدامیک از رشته عددهای زیر می‌باشد؟ (مسابقات آموزشکده‌های فنی ۸۲)



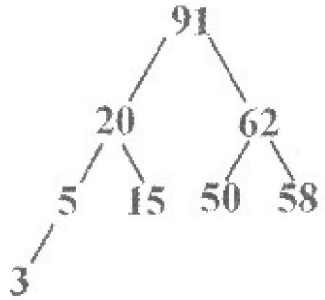
۱) 12,6, 7,8,13,15,16

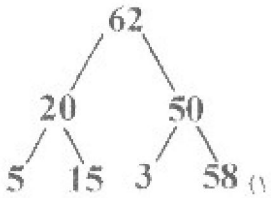
۲) 12,6,7,8,15,16,13

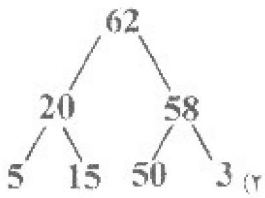
۳) 12,7,6,8,13,15,16

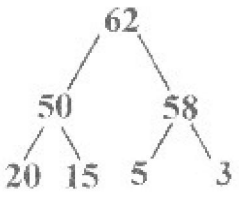
۴) 12,15,7,6,8,16,13

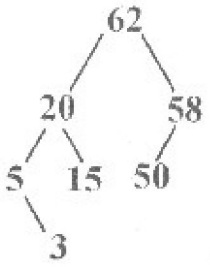
۱۹۷ ـ اگر در درخت Max heap زیر ریشه حذف شود، درخت جدید کدام خواهد بود؟ (مسابقات آموزشکده‌های فنی ۸۲)



۱)


۲)


۳)


۴)


۱۹۸ ـ در کدام نوع از درخت‌های دودویی پیمایش درخت به‌صورت inorder باعث پدیدآمدن یک لیست مرتب می‌شود؟ (آزاد ۸۲)

۱) heap ۲) binary tree ۳) binary search tree ۴) spanning tree

۱۹۹ ـ کدام گزینه نادرست است؟ (آزاد ۸۲) (دولتی ۸۲)

۱) یک درخت دوتایی کامل همیشه یک heap است.

۲) یک heap همیشه یک درخت دوتایی کامل است.

۳) یک heap همیشه از نوع درخت جستجوی دودویی نیست.

۴) یک درخت جستجوی دودویی (Binary Search) همیشه یک heap نیست.

۲۰۰ ـ درخت دودویی T این خاصیت را دارد که هر گره از فرزند چپش بزرگ‌تر و از فرزند راستش کوچک‌تر است در این صورت T:

(علمی کاربردی ۸۲)

۱) یک درخت دودویی جستجو است
۲) max Heap است.
۳) min heap است.
۴) هیچکدام.

۲۰۱ ـ فرض کنید زیر برنامه Test روی یک درخت دودویی جستجو اعمال گردد. پس از انجام تغییرات داده شده کدام گزینه صحیح است؟

(علمی کاربردی ۸۲)

```
Procedure Test (var T:tree);
var
  p:tree
Begin
  If T < > nil then
    Begin
      Test (T^.right);  writeln (T^.data);
       Test (T^.left);
      P:=T^.right;
      T^.right : = T^.left;
      T^.left = p;
    End;
End;
```

۱) پیمایش پس ترتیب postorder درخت، اعداد را به‌صورت نزولی چاپ می‌کند.

۲) پیمایش پس ترتیب postorder درخت، اعداد را به‌صورت صعودی چاپ می‌کند.

۳) پیمایش بین ترتیب inorder درخت، اعداد را به‌صورت نزولی چاپ می‌کند.

۴) پیمایش پیش ترتیب preorder درخت اعداد را به صورت صعودی چاپ می‌کند.

۲۰۲ ـ خروجی تابع زیر با فرض آن‌که T یک درخت دودویی جستجو باشد چیست؟ (علمی کاربردی)

```
Function what (T:Tree) Integer;
Var
P: tree;
Begin
P: = T;
while   P ^.left <>nil  Do
      P : = P ^.left;
what : = P ^.element;
end;
```

۱) عنصر وسط از لحاظ بزرگی
۲) ماکزیمم عنصر موجود در درخت
۳) مینیمم عنصر موجود در درخت
۴) بستگی به نحوهٔ قرارگرفتن اعداد در درخت دارد.

۲۰۳ ـ حداکثر تعداد لبه‌های یک گراف جهت‌دار شامل n گره برابر است با: (آزاد ۷۹)

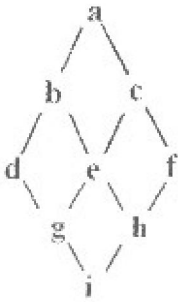
۱) $2n-1$ ۲) n^2-n ۳) n^2-1 ۴) $2n-n$

۲۰۴ ـ کدامیک از مجموعه‌های زیر نمایش‌دهنده یک درخت است؟ (آزاد ۷۹)

۱) $E(G)=\{(0,1),(0,2),(1,3),(1,4),(2,5)\}$ ۲) $E(G)=\{(0,1),(0,2),(0,3),(1,2)\}$

۳) $E(G)=\{(0,1),(0,3),(0,2),(2,0)\}$ ۴) $E(G)=\{(0,1),(0,2),(4,0),(4,3),(4,2)\}$

۲۰۵ ـ پیمایش عمقی (depth - first search) گراف زیر چیست؟ اگر از گره a شروع کنیم (به‌ترتیب از چپ به راست) (مسابقات آموزشکده‌های فنی ۷۹)



۱) a,b,c,d,e,f,g,h,i

۲) a,b,d,g,i,h,c,e,f

۳) a,b,c,f,h,e,i,g,d

۴) a,b,d,g,i,h,f,c,e

۲۰۶ ـ کدام گزینه نادرست است؟ (مسابقات آموزشکده‌های فنی ۷۶)

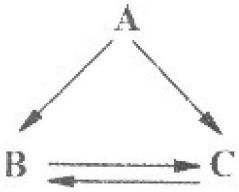
۱) تفاوتی میان گراف و گراف چندگانه وجودندارد.

۲) حلقه یک مسیر ساده است که اولین و آخرین راس آن یکی باشد.

۳) در یک گراف بدون جهت با n رأس بیشترین تعداد لبه‌ها $n(n-1)/2$ است.

۴) در یک گراف جهت‌دار با n رأس بیشترین تعداد لبه‌ها $n(n-1)$ است.

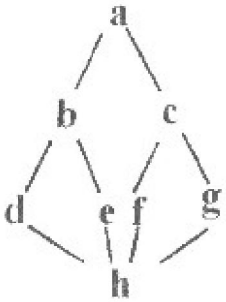
۲۰۷ ـ در گراف زیر چند مسیر به طول 2 وجود دارد؟ (مسابقات آموزشکده‌های فنی ۷۷ و ۸۰)



۱) 1 ۲) 2

۳) 3 ۴) 4

۲۰۸ ـ پیمایش عمقی گراف زیر چه خواهد بود؟ اگر از گره a شروع کنیم. (پاسخ‌ها از چپ به راست نوشته‌شده‌اند.) (دولتی ۸۰)



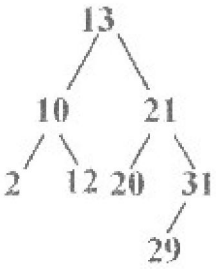
۱) h,d,e,f,g,b,c,a

۲) a,c,g,h,f,b,e,d

۳) a,b,c,d,e,f,g,h

۴) a,b,d,h,e,f,c,g

۲۰۹ ـ در شکل زیر کدامیک از انتخاب‌ها روش Breadth – First – Traversal یک درخت است؟ (آزاد ۸۰)



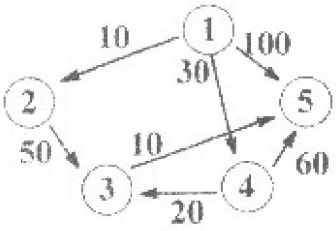
۱) 13,21,10,31,20,12,29,2

۲) 13,10,21,31,20,12,2,29

۳) 13,10,2,12,21,20,31,29

۴) 13,10,21,2,12,20,31,29

۲۱۰ ـ در گراف زیر کمترین هزینه از گره ۱ به 5 دارای مسیری با طول؟ (مسابقات آموزشکده‌های فنی ۸۱)



۱) 1

۲) 2

۳) 3

۴) 4

۲۱۱ ـ از کدام روش برای نمایش گراف استفاده نمی‌شود؟ (دولتی ۸۱)

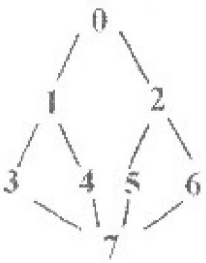
۱) ماتریس مجاورتی

۲) لیست یک طرفه دوار

۳) لیست مجاورتی

۴) لیست چندگانه مجاورتی

۲۱۲ ـ پیمایش dfs(0) روبرو کدام است؟ (دولتی ۸۱)



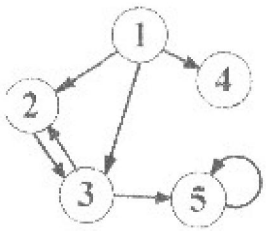
۱) 1,3,4,6,0,2,4,6

۲) 0,1,2,3,4,5,6,7

۳) 0,1,3,7,4,5,2,6

۴) 0,2,1,4,3,5,6,7

۲۱۳ ـ ماتریس مجاورتی (همجواری) گراف زیر کدام است؟ (مسابقات آموزشکده‌های فنی ۸۲)



۱)
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

۲)
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

۳)
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

۴)
$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

۲۱۴ ـ کدام گزینه نادرست است؟ (مسابقات آموزشکده‌های فنی۸۲)

۱) در یک گراف بدون جهت با n رأس بیشترین تعداد لبه‌ها $\frac{n(n-1)}{2}$ است.

۲) حلقه یک مسیر ساده است که اولین و آخرین رأس آن یکی می‌باشد.

۳) تفاوتی میان گراف و گراف چندگانه وجود ندارد.

۴) در یک گراف جهت‌دار با n رأس بیشترین تعداد لبه‌ها $n(n-1)$ است.

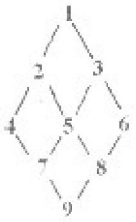
۲۱۵ ـ اگر G یک گراف ساده از درجه n باشد، درصورتی که گراف G دارای 56 یال و مجموع درجات رئوس مکمل گراف G برابر 160 باشد مقدار n کدام است؟ (دولتی۸۲)

۱) -16 ۲) 16 ۳) 17 ۴) -17

۲۱۶ ـ اگر a, b دو گره در یک گراف بدون‌جهت G باشند و اگر دو مسیر P_1, P_2 از a به b وجود داشته باشد، آنگاه: (دولتی۸۲)

۱) a , b مجاورند. ۲) G نمی‌تواند یک گراف باشد. ۳) G دارای چرخه است. ۴) احتیاج به جهت مسیر داریم.

۲۱۷ ـ پیمایش عمقی (depth – First – Search) گراف زیر کدام است؟ (دولتی۸۲)



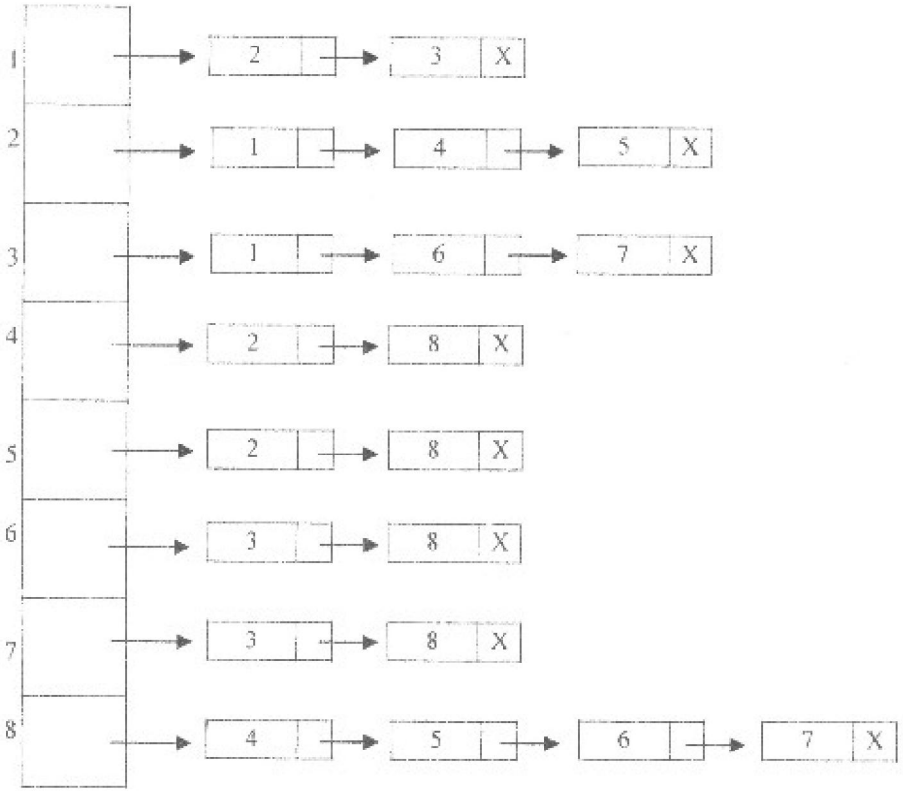
۱) 1,2,3,4,5,6,7,8,9

۲) 1,2,4,7,9,8,5,3,6

۳) 1,2,3,6,8,5,9,7,4

۴) 1,2,4,7,9,8,6,3,5

۲۱۸ ـ لیست مجاورتی گراف غیرجهت‌دار زیر را در نظر بگیرید. جستجوی عمقی (dfs) عبارت است از: (آزاد۸۲)



۱) 1,2,4,8,5,6,3,7

۲) 1,2,3,4,5,6,7,8

۳) 1,2,3,6,7,8,5,4

۴) 1,3,2,6,8,4,5,7

۲۱۹ ـ گراف زیر را در نظر بگیرید. کدام گزینه، زیرگراف گراف فوق است؟ (آزاد۸۲)

۱) ۲) ۳) ۴) هر سه گزینه

۲۲۰ ـ خروجی کدام الگوریتم درخت پوشا برای گراف نیست؟ (علمی کاربردی۸۲)

۱) Prim ۲) Dijkestra ۳) kruskal ۴) BFS

۲۲۱ ـ اگر A ماتریس مجاورت یک گراف جهت‌دار از مرتبه n بوده و v یک آرایه n عنصری که تمام عناصر آن در ابتدا True هستند باشد، تابع ذیل با دریافت دو راس i , j چه کاری انجام می‌دهد ؟ (علمی کاربردی۸۲)

```
Function F (i , j : Byte) Boolean;
Var
T : Boolean;
Begin
V [i] : = False;
if A[i , j] = 1 then F : = True
else Begin
T : = False ;
for k : = 1 To n Do
if (A[i , k] = 1) and V [k] then
T : = T  or  F (i , j);
F : = T ;
End;
End;
```

۱) تشخیص می‌دهد آیا از راس i تا به راس j مسیر وجود دارد یا خیر؟

۲) تشخیص می‌دهد آیا دوری با شروع از راس i و با گذر از راس j در گراف وجود دارد یا خیر؟

۳) تشخیص می‌دهد آیا یالی از راس i به راس j در گراف می‌باشد یا خیر؟

۴) تشخیص می‌دهد آیا مسیری به طول n از i به j وجود دارد یا خیر؟

مرتب سازی

۲۲۲ ـ اگر یک لیست مرتب شده با n خانه را با استفاده از الگوریتم Binary search برای یک مقدار خاص جستجو کنیم، تعداد دفعات مقایسه چه خواهد بود؟

۱) $O(n/2)$ ۲) $O(n^2)$ ۳) $O(\log n)$ ۴) $O(\log_2 n)$

۲۲۳ ـ برای مرتب سازی یک آرایه nتایی به روش Bubble – sort حداکثر چند تعویض لازم است؟ (مسابقات آموزشکده‌های فنی۷۹)

۱) $\frac{N(N-1)}{2}$ ۲) $\log_2(N-1)$ ۳) $\frac{N^2}{4}+3(N-1)$ ۴) $\frac{N^2-3N-4}{2}$

۲۲۴ ـ کدام کار روی همه ساختارهای داده‌ی انجام می‌شود؟ (مسابقات آموزشکده‌های فنی ۸۰)

۱) ادغام (Merge)
۲) پردازش Process
۳) مرتب‌سازی (sort)
۴) درج (Insert)

۲۲۵ ـ نماد $O(\log n)$ نشان‌دهنده پیچیدگی کدام الگوریتم است؟ (مسابقات آموزشکده‌ها ۸۰)

۱) جستجوی دودویی (Binary Search)
۲) جستجوی خطی (Liner Search)
۳) مرتب‌سازی حبابی (Bubble sort)
۴) مرتب‌سازی سریع (Quick sort)

۲۲۶ ـ بدترین حالت در روش مرتب‌سازی سریع (Quick sort) چیست؟ (دولتی ۸۰)

۱) عناصر لیست به ترتیب معکوس باشند.
۲) عناصر لیست از قبل مرتب باشد.
۳) یک نیمه لیست مرتب باشد.
۴) یک نیمه لیست معکوس باشد.

۲۲۷ ـ چنانچه بخواهیم داده‌های تکرار را از لیستی حذف کنیم، از کدام ساختار داده‌ای برای لیست مذبور استفاده می‌کنیم؟ (دولتی ۸۰)

۱) درخت جستجوی دودویی ۲) درخت Heap ۳) پشته ۴) صف

۲۲۸ ـ الگوریتم Quick sort یک رشته n تایی را با چه سرعتی مرتب می‌کند؟

۱) $O(n \log n)$ ۲) $O(n)$ ۳) $O(n^2)$ ۴) $\log 2^n$

۲۲۹ ـ الگوریتم Quick sort یک رشته n تایی را در بدترین حالت با چه سرعتی مرتب می‌کند؟ (مسابقات آموزشکده‌های فنی)

۱) $O(n)$ ۲) $O(n^2)$ ۳) $O(\log n)$ ۴) $O(n \log n)$

۲۳۰ ـ اگر آرایه مرتب A دارای r عنصر و آرایه مرتب B دارای p عنصر باشد، حداکثر تعداد مقایسه برای ترتیب (Merge) دو آرایه کدام است؟

۱) $\frac{r+p}{2}$ ۲) $r+p$ ۳) $\text{Max}(r, p)$ ۴) $\log r * p$

۲۳۱ ـ اگر N رکورد داشته باشیم تعداد کل مقایسه در روش liner Insertion sort برابراست با :

۱) $\frac{N^2}{4}$ ۲) $\frac{N(N-1)}{2}$ ۳) $\frac{N(N+1)}{4}$ ۴) $\frac{N^2}{4}$

۲۳۲ ـ در کدام روش مرتب‌سازی، لیست داده‌ها همواره به دو نیمه تقسیم می‌شود؟

۱) Bubble sort ۲) Merge sort ۳) Quick sort ۴) Selection sort

۲۳۳ ـ آرایه زیر یک Heap است. برای درج عدد 95 در آرایه به گونه‌ای که آرایه نهایی نیز وضعیت Heap داشته باشد، چند عمل exchange (تعویض دو کمیت) لازم است؟

100
90
82
85
74
75
73
68
70

۱) دو
۲) چهار
۳) شش
۴) هشت

۲۳۴ ـ کاربرد درخت Heap کدام است؟

۱) جستجوی سریع
۲) صف و پشته
۳) مرتب کردن داده‌ها
۴) مرتب کرده داده‌ها ـ صف اولویت‌دار

۲۳۵ ـ برای مرتب‌سازی یک آرایه n تایی به روش مرتب‌سازی حبابی، حداکثر چند تعویض لازم است؟

۱) n^2
۲) $n \log n$
۳) $\log_2 (n-1)$
۴) $\frac{n(n-1)}{2}$

۲۳۶ ـ الگوریتم زیر چه نوع مرتب‌سازی است و تعداد مقایسه‌های آن برحسب تعداد داده‌ها (n) کدام است؟

1. Repeat step 2 and 3 for k = 1 to N-1
2. Set PTR : = 1
3. Repeat while PTR ≤ N – k
if DATA [PTR] > DATA [PRT + 1] then
swap DATA [PRT] and DATA [PTR + 1]
PTR : = PTR + 1
End of step 3 loop
End of step 1 loop
4. Exit

۱) حبابی ـ $\frac{n(n+1)}{2}$
۲) حبابی ـ $\frac{n(n-1)}{2}$
۳) درجی ـ $\frac{n(n-1)}{2}$
۴) درجی ـ $\frac{n(n+1)}{2}$

۲۳۷ ـ می‌خواهیم یک آرایه 5 عضوی را با استفاده از روش Selection sort مرتب کنیم. اگر آرایه در ابتدا، به ترتیب عکس مرتب شده باشد، چه تعداد مقایسه برای مرتب کردن آن مورد نیاز است؟

۱) 1
۲) 10
۳) 15
۴) 20

۲۳۸ ـ چنانچه یک بردار کاملاً مرتب را به الگوریتم liner insertion sort بدهیم تعداد جابجایی و تعداد مقایسه چگونه است؟

۱) تعداد جابجایی صفر و حداکثر مقایسه را خواهیم داشت.
۲) تعداد جابجایی و مقایسه در بالاترین سطح می‌باشند.
۳) تعداد جابجایی صفر و حداقل مقایسه را خواهیم داشت.
۴) تعداد جابجایی و تعداد مقایسه تغییری با حالت نامرتب ندارند.

۲۳۹ ـ اگر بخواهیم یک لیست متصل (linked list) که آدرس اول آن first می‌باشد را با کمترین تعداد عملیات مرتب نماییم (sort) به جای علامت ؟ در الگوریتم زیر چه مقادیری به ترتیب قرار دهیم؟

```
for (P = first ; ? ;  P = P → link)
for (q = ? ; q ; q = q → link)
if (P → Data > q → Data)
swap (& P → Data , & q → Data)
```

۱) P , P → link
۲) P → link , P
۳) P , P
۴) P → link , P → link

۲۴۰ ـ کدامیک از روشهای sort زیر در بدترین حالت از $o(n*n)$ است؟

۱) Bubble sort
۲) Insertion sort
۳) Quick sort
۴) هر سه

۲۴۱ ـ آرایه A به طول n را در نظر بگیرید. الگوریتم زیر بیانگر کدام نوع عمل مرتب‌سازی است؟ (دولتی ۸۲)

```
For j← 2 to n do
key ← A[j]
Insert A[j] Into the sorted sequence A[1:j-1]
        I ← j -1
while   I > 0  and  A[I] > key
        Do A[I+1] ← A[I]
I = I - 1
A[I+1] ← key
```

۱) Bubble sort
۲) Insertion sort
۳) selection sort
۴) Quick sort

۲۴۲ ـ کدام الگوریتم مرتب‌سازی یک آرایه تقریباً مرتب شده را سریعتر مرتب می‌نماید؟ (علمی کاربردی ۸۲)

۱) Selection sort
۲) Quick sort
۳) Bubble sort
۴) Insertion sort

۲۴۳ ـ برای ادغام (merge) دو آرایه مرتب شده A و B که به ترتیب m و n عنصری هستند حداکثر چند مقایسه لازم است؟ (علمی کاربردی ۸۲)

۱) $n+m$
۲) $n+m-1$
۳) $\max(n,m)$
۴) $\frac{(m+n)(m+n-1)}{2}$

۲۴۴ ـ اگر آرایه‌ای مرتب از اعداد صحیح 1 تا 1024 باشد، الگوریتم جستجوی دودوئی با چند بار تکرار عدد 4 را پیدا می‌کند؟

۱) 8
۲) 7
۳) 9
۴) 10

دولتی ۸۳

۲۴۵ـ الگوریتم مقابل کدام عمل را انجام می‌دهد؟

```
function  L (x : pointer ) : integer;
var   p : pointer;
begin
L : = 0;
if   x < > nil   then   begin
p : = x;
repeat
L : = L + 1;
p : = p^.link;
until   p = x;
end;
end;
```

۱) پیمایش لیست پیوندی خطی
۲) پیمایش لیست پیوندی چرخشی
۳) تعیین طول لیست پیوندی خطی
۴) تعیین طول لیست چرخشی

۲۴۶ـ باتوجه به تکه برنامه مقابل مرتبهٔ اجرایی (8)f عبارت است از:

```
i = 1;
while   (i < = n)
{
j = 1;
while   (j < = n)
{
j = j * 2;
}
i = i + 1;
}
```

۱) $\log_2 8$
۲) $\log_2 8 + 1$
۳) $8\left(\frac{8+1}{2}\right)$
۴) $8\left(\log_2 8 + 1\right)$

۲۴۷ـ کدام مورد در ساختار یک صف حلقوی 1000 عنصری بیان کننده خالی و پر بودن صف است؟

۱) rear = 0 ، front = 0 (خالی) و front = (rear − 1) mod 1000 (پر)
۲) rear = 0 ، front = 0 (خالی) و rear = 1000 ، front = 0 (پر)
۳) rear = 1 ، front = 0 (خالی) و rear = (rear + 1) mod 1000 (پر)
۴) rear = 1001 ، front = 1000 (خالی) و front = 0 ، rear = 999 (پر)

۲۴۸ ـ ساختار یک صف پیوندی (نمایش ترتیبی صف به وسیلهٔ لیست پیوندی) اگر front و rear به ترتیب اشاره به بخش‌های آغازین صف داشته باشند، کدام الگوریتم نمایش اضافه کردن به یک صف پیوندی است؟

۱)
```
procedure  add (i , y : integer);
var  x : pointer;
begin
new (x);
x ^ . data : = y;   y ^ . link : = nil;
if  front [i] = nil  then  rear [i] : = x
else  rear [i] ^ . link : = x;
front [i] : = x;
end;
```

۲)
```
procedure  add (i , y : integer);
var  x : pointer;
begin
new (x);
x ^ . data : = x;   x ^ . link : = y;
if  front [i] = nil  then  rear [i] : = x
else  rear [i] : = y;
end;
```

۳)
```
procedure  add (i , y : integer);
var  x : pointer;
begin
new (x);
y ^ . data : = x;   x ^ . link : = nil;
rear [i] : = x;
end;
```

۴)
```
procedure  add (i , y : integer);
var  x : pointer;
begin
new (x);
x ^ . data : = y;   x ^ . link : = nl;
if  front [i] = nil  then  front [i] : = x
else  rear [i] ^ . link : = x;
rear [i] : = x;
end;
```

۲۴۹ ـ حداقل تعداد عناصر یک درخت دودویی کامل کدام است؟

۱) 1　　۲) 2　　۳) 3　　۴) 4

۲۵۰ ـ لیست s که از $n = 6$ حرف تشکیل شده به صورت A , B , C , D , E , F می‌باشد، مقایسه‌های لازم برای مرتب کردن s با الگوریتم Quicksort عبارت است از:

۱) لیست مرتب است و مقایسه‌ای نداریم.　　۲) 36

۳) 15　　۴) 10

۲۵۱ ـ لیست مجاورت مربوط به گراف G را در نظر بگیرید، پیمایش عمقی این گراف عبارتست از:

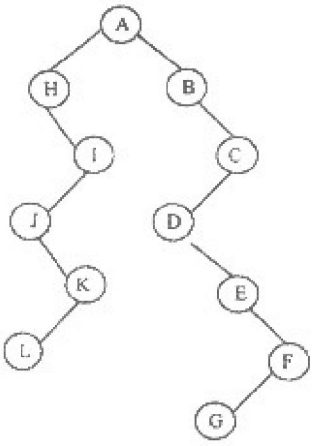
adj list
A : B , C , D
B : A , D , E
C : A , D , F
D : A , B , C
E : B , F
F : C , E

۱) A B D C F E　　۲) A B E F C A　　۳) A C F D E B　　۴) A D C B E F

۲۵۲ - حاصل عبارت 3 , 2 , + , 5 , / , 10 , / , 1 , 1 , − , 3 , 2 , ↑ , * , − , − کدام است؟

۱) 9 - ۲) 2 - ۳) 1 ۴) 10

۲۵۳ - پیمایش LVR درخت دودویی مقابل کدام است؟



۱) A H I J K L B C D E F G

۲) H J L K I A B D E G F C

۳) H J L K I A D E C F G

۴) L K J I H A G F E D C B

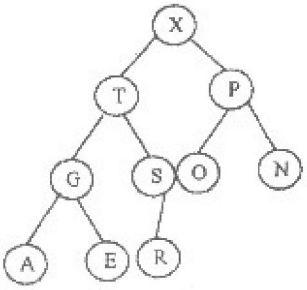
۲۵۴ - روال بازگشتی مقابل را در نظر بگیرید مطلوب است M [30 , 5]

```
M (A , B)
P ← 0
while   A < > 0
do  if   A   mod   2 = 1
then   P ← P + B
A ← ⌊A / 2⌋
B ← 2B;
return   p
```

۱) 20 ۲) 40 ۳) 70 ۴) 150

۲۵۵ - درخت max Heap به شکل زیر با روال قید شده مشخص شده است. آرایه A نمایش درخت فوق به صورت ترتیبی است و Length (A) طول آرایه و heapsize طول درخت heap می‌باشند کدام گزینه صحیح است؟

```
parent (A , i)
return (i / 2)
left (A , i)
return 2i
right (A , i)
return (2i + 1)
```



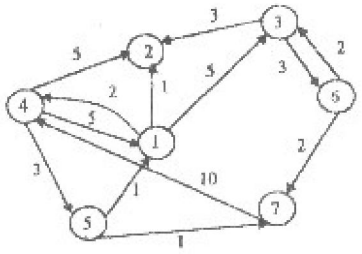
۱) length (A) = 10 , heapsize = 10

۲) length (A) = 10 , heapsize = 11

۳) length (A) = 11 , heapsize = 10

۴) length (A) = 11 , heapsize = 11

۲۵۶ ـ در گراف زیر کمترین هزینه از گره 1 به گره 7 عبارت است از:



۱) 8

۲) 10

۳) 20

۴) 6

۲۵۷ ـ شکل یک زیر رشته substring به صورت زیر است:

substring (string , initial , lenght)

که در آن string نام رشته اصلی و initial مکان اولین کاراکتر زیر رشته در رشته اصلی و length طول زیر رشته می‌باشد. متن داده شده T را در نظر بگیرید می‌خواهیم رشته (String) را طوری به آن اضافه کنیم که string از مکان K شروع می‌شود insert (t , k , string) دو رشته s_1 و s_2 به شکل مقابل به یکدیگر متصل می‌شوند: $s_1 \| s_2$

s_1 = 'AL'
s_2 = 'reza' $\Rightarrow$ $s_1 \| s_2$ = 'ALreza'

اگر عمل insert به شکل زیر تعریف شود:

insert (T , K , string) = substring (T , 1 , K − 1) ‖string‖
substring (T , K , lenght (T) − K + 1)

حاصل insert ('A B C D E F G', 6 , 'x y z') چیست؟

۱) A B C D x y z E F G

۲) A B C D E F G x y z

۳) A B C D x y z F G E

۴) A B C D E x y z F G

۲۵۸ ـ فرض کنید Maze یک آرایه سه بعدی است که به صورت Maze (2 .. 8 , − 4 .. 1 , 6 .. 10) تعریف می‌شود، زبان برنامه‌نویسی Maze را به روش سطری در حافظه ذخیره می‌کند و Base (Maze) = 200 و w = 4 کلمه در حافظه موجود است، Loc (Maze [5 , − 1 , 8]) چیست؟

۱) 200

۲) 207

۳) 428

۴) 628

۲۵۹ ـ در عمل حذف در یک لیست پیوندی با در نظر گرفتن لیست در دسترس حافظه (حاوی خانه‌های خالی لیست) چند آدرس جایگزینی انجام می‌شود؟

۱) 4

۲) 3

۳) 2

۴) 1

پاسخ تشریحی

پشته و صف

۱ ـ گزینه ۴ صحیح می‌باشد.

۲ ـ گزینه ۲ صحیح می‌باشد.

۳ ـ گزینه ۳ صحیح می‌باشد.

۴ ـ گزینه ۳ صحیح می‌باشد.

۵ ـ گزینه ۱ صحیح می‌باشد.

۶ ـ گزینه ۳ صحیح می‌باشد.

۷ ـ گزینه ۳ صحیح می‌باشد.

۸ ـ گزینه ۱ صحیح می‌باشد.

۹ ـ گزینه ۱ صحیح می‌باشد.

۱۰ ـ گزینه ۱ صحیح می‌باشد.

۱۱ ـ گزینه ۴ صحیح می‌باشد.

۱۲ ـ گزینه ۲ صحیح می‌باشد.

۱۳ ـ گزینه ۱ صحیح می‌باشد.

۱۴ ـ گزینه ۳ صحیح می‌باشد.

۱۵ ـ گزینه ۲ صحیح می‌باشد.

۱۶ ـ گزینه ۱ صحیح می‌باشد.

۱۷ ـ گزینه ۴ صحیح می‌باشد.

۱۸ ـ گزینه ۳ صحیح می‌باشد.

۱۹ ـ گزینه ۳ صحیح می‌باشد.

۲۰ ـ گزینه ۴ صحیح می‌باشد.

۲۱ ـ گزینه ۲ صحیح می‌باشد.

۲۲ ـ گزینه ۱ صحیح می‌باشد.

۲۳ ـ گزینه ۴ صحیح می‌باشد.

۲۴ ـ گزینه ۴ صحیح می‌باشد.

۲۵ ـ گزینه ۲ صحیح می‌باشد.

۲۶ ـ گزینه ۱ صحیح می‌باشد.

۲۷ ـ گزینه ۳ صحیح می‌باشد.

۲۸ ـ گزینه ۲ صحیح می‌باشد.

۲۹ ـ گزینه ۳ صحیح می‌باشد.

۳۰ ـ گزینه ۳ صحیح می‌باشد.

۳۱ ـ گزینه ۴ صحیح می‌باشد.

۳۲ ـ گزینه ۱ صحیح می‌باشد.

۳۳ ـ گزینه ۳ صحیح می‌باشد.

۳۴ ـ گزینه ۳ صحیح می‌باشد.

۳۵ ـ گزینه ۳ صحیح می‌باشد.

۳۶ ـ گزینه ۴ صحیح می‌باشد.

۳۷ ـ گزینه ۳ صحیح می‌باشد.

۳۸ ـ گزینه ۱ صحیح می‌باشد.

۳۹ ـ گزینه ۲ صحیح می‌باشد.

۴۰ ـ گزینه ۱ صحیح می‌باشد.

۴۱ ـ گزینه ۴ صحیح می‌باشد.

۴۲ ـ گزینه ۲ صحیح می‌باشد.

۴۳ ـ گزینه ۴ صحیح می‌باشد.

۴۴ ـ گزینه ۳ صحیح می‌باشد.

۴۵ ـ گزینه ۲ صحیح می‌باشد.

۴۶ ـ گزینه ۴ صحیح می‌باشد.

۴۷ ـ گزینه ۳ صحیح می‌باشد.

۴۸ ـ گزینه ۴ صحیح می‌باشد.

۴۹ ـ گزینه ۴ صحیح می‌باشد.

۵۰ ـ گزینه ۲ صحیح می‌باشد.

۵۱ ـ گزینه ۴ صحیح می‌باشد.

۵۲ ـ گزینه ۴ صحیح می‌باشد.

۵۳ ـ گزینه ۲ صحیح می‌باشد.

۵۴ـ گزینه ۳ صحیح می‌باشد.

۵۵ـ گزینه ۲ صحیح می‌باشد.

۵۶ـ گزینه ۳ صحیح می‌باشد.

۵۷ـ گزینه ۲ صحیح می‌باشد.

۵۸ـ گزینه ۳ صحیح می‌باشد.

۵۹ـ گزینه ۴ صحیح می‌باشد.

۶۰ـ گزینه ۱ صحیح می‌باشد.

۶۱ـ گزینه ۱ صحیح می‌باشد.

۶۲ـ گزینه ۳ صحیح می‌باشد.

۶۳ـ گزینه ۳ صحیح می‌باشد.

۶۴ـ گزینه ۲ صحیح می‌باشد.

۶۵ـ گزینه ۲ صحیح می‌باشد.

۶۶ـ گزینه ۴ صحیح می‌باشد.

۶۷ـ گزینه ۲ صحیح می‌باشد.

۶۸ـ گزینه ۳ صحیح می‌باشد.

۶۹ـ گزینه ۳ صحیح می‌باشد.

۷۰ـ گزینه ۱ صحیح می‌باشد.

۷۱ـ گزینه ۳ صحیح می‌باشد.

۷۲ـ گزینه ۲ صحیح می‌باشد.

۷۳ـ گزینه ۴ صحیح می‌باشد.

۷۴ـ گزینه ۱ صحیح می‌باشد.

۷۵ـ گزینه ۴ صحیح می‌باشد.

۷۶ـ گزینه ۴ صحیح می‌باشد.

۷۷ـ گزینه ۲ صحیح می‌باشد.

۷۸ـ گزینه ۱ صحیح می‌باشد.

۷۹ـ گزینه ۳ صحیح می‌باشد.

۸۰ـ گزینه ۳ صحیح می‌باشد.

۸۱ـ گزینه ۳ صحیح می‌باشد.

۸۲ـ گزینه ۱ صحیح می‌باشد.

۸۳ـ گزینه ۱ صحیح می‌باشد.

۸۴ـ گزینه ۴ صحیح می‌باشد.

۸۵ـ گزینه ۲ صحیح می‌باشد.

۸۶ـ گزینه ۴ صحیح می‌باشد.

۸۷ـ گزینه ۱ صحیح می‌باشد.

۸۸ـ گزینه ۲ صحیح می‌باشد.

۸۹ـ گزینه ۱ صحیح می‌باشد.

۹۰ـ گزینه ۳ صحیح می‌باشد.

لیست پیوندی

۹۱ـ گزینه ۳ صحیح می‌باشد.

۹۲ـ گزینه ۱ صحیح می‌باشد.

۹۳ـ گزینه ۲ صحیح می‌باشد.

۹۴ـ گزینه ۳ صحیح می‌باشد.

۹۵ـ گزینه ۳ صحیح می‌باشد.

۹۶ـ گزینه ۳ صحیح می‌باشد.

۹۷ـ گزینه ۲ صحیح می‌باشد.

۹۸ـ گزینه ۱ صحیح می‌باشد.

۹۹ـ گزینه ۳ صحیح می‌باشد.

۱۰۰ـ گزینه ۳ صحیح می‌باشد.

۱۰۱ـ گزینه ۱ صحیح می‌باشد.

۱۰۲ـ گزینه ۴ صحیح می‌باشد.

۱۰۳ـ گزینه ۴ صحیح می‌باشد.

۱۰۴ـ گزینه ۱ صحیح می‌باشد.

۱۰۵ـ گزینه ۳ صحیح می‌باشد.

۱۰۶ـ گزینه ۲ صحیح می‌باشد.

۱۰۷ـ گزینه ۳ صحیح می‌باشد.

۱۰۸ ـ گزینه ۲ صحیح می‌باشد.

۱۰۹ ـ گزینه ۴ صحیح می‌باشد.

۱۱۰ ـ گزینه ۲ صحیح می‌باشد.

۱۱۱ ـ گزینه ۴ صحیح می‌باشد.

۱۱۲ ـ گزینه ۳ صحیح می‌باشد.

۱۱۳ ـ گزینه ۱ صحیح می‌باشد.

۱۱۴ ـ گزینه ۲ صحیح می‌باشد.

۱۱۵ ـ گزینه ۱ صحیح می‌باشد.

۱۱۶ ـ گزینه ۳ صحیح می‌باشد.

۱۱۷ ـ گزینه ۲ صحیح می‌باشد.

۱۱۸ ـ گزینه ۳ صحیح می‌باشد.

۱۱۹ ـ گزینه ۳ صحیح می‌باشد.

۱۲۰ ـ گزینه ۴ صحیح می‌باشد.

۱۲۱ ـ گزینه ۲ صحیح می‌باشد.

۱۲۲ ـ گزینه ۳ صحیح می‌باشد.

۱۲۳ ـ گزینه ۲ صحیح می‌باشد.

۱۲۴ ـ گزینه ۴ صحیح می‌باشد.

۱۲۵ ـ گزینه ۳ صحیح می‌باشد.

۱۲۶ ـ گزینه ۲ صحیح می‌باشد.

۱۲۷ ـ گزینه ۳ صحیح می‌باشد.

۱۲۸ ـ گزینه ۲ صحیح می‌باشد.

۱۲۹ ـ گزینه ۴ صحیح می‌باشد.

۱۳۰ ـ گزینه ۲ صحیح می‌باشد.

۱۳۱ ـ گزینه ۱ صحیح می‌باشد.

۱۳۲ ـ گزینه ۳ صحیح می‌باشد.

۱۳۳ ـ گزینه ۴ صحیح می‌باشد.

۱۳۴ ـ گزینه ۱ صحیح می‌باشد.

۱۳۵ ـ گزینه ۳ صحیح می‌باشد.

۱۳۶ ـ گزینه ۴ صحیح می‌باشد.

۱۳۷ ـ گزینه ۳ صحیح می‌باشد.

۱۳۸ ـ گزینه ۳ صحیح می‌باشد.

درخت

۱۳۹ ـ گزینه ۳ صحیح می‌باشد.

۱۴۰ ـ گزینه ۴ صحیح می‌باشد.

۱۴۱ ـ گزینه ۳ صحیح می‌باشد.

۱۴۲ ـ گزینه ۱ صحیح می‌باشد.

۱۴۳ ـ گزینه ۳ صحیح می‌باشد.

۱۴۴ ـ گزینه ۲ صحیح می‌باشد.

۱۴۵ ـ گزینه ۲ صحیح می‌باشد.

۱۴۶ ـ گزینه ۴ صحیح می‌باشد.

۱۴۷ ـ گزینه ۱ صحیح می‌باشد.

۱۴۸ ـ گزینه ۲ صحیح می‌باشد.

۱۴۹ ـ گزینه ۱ صحیح می‌باشد.

۱۵۰ ـ گزینه ۴ صحیح می‌باشد.

۱۵۱ ـ گزینه ۲ صحیح می‌باشد.

۱۵۲ ـ گزینه ۳ صحیح می‌باشد.

۱۵۳ ـ گزینه ۱ صحیح می‌باشد.

۱۵۴ ـ گزینه ۱ صحیح می‌باشد.

۱۵۵ ـ گزینه ۴ صحیح می‌باشد.

۱۵۶ ـ گزینه ۲ صحیح می‌باشد.

۱۵۷ ـ گزینه ۳ صحیح می‌باشد.

۱۵۸ ـ گزینه ۴ صحیح می‌باشد.

۱۵۹ ـ گزینه ۲ صحیح می‌باشد.

۱۶۰ ـ گزینه ۲ صحیح می‌باشد.

۱۶۱ ـ گزینه ۱ صحیح می‌باشد.

۱۶۲ ـ گزینه ۴ صحیح می‌باشد.

۱۶۳ ـ گزینه ۲ صحیح می‌باشد.

۱۶۴ ـ گزینه ۴ صحیح می‌باشد.

۱۶۵ ـ گزینه ۳ صحیح می‌باشد.

۱۶۶ ـ گزینه ۳ صحیح می‌باشد.

۱۶۷ ـ گزینه ۲ صحیح می‌باشد.

۱۶۸ ـ گزینه ۳ صحیح می‌باشد.

۱۶۹ ـ گزینه ۱ صحیح می‌باشد.

۱۷۰ ـ گزینه ۱ صحیح می‌باشد.

۱۷۱ ـ گزینه ۲ صحیح می‌باشد.

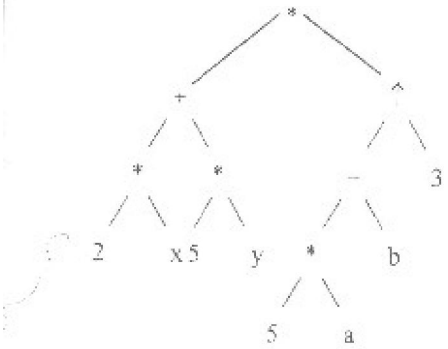
۱۷۲ ـ گزینه ۴ صحیح می‌باشد.

۱۷۳ ـ گزینه ۱ صحیح می‌باشد.

۱۷۴ ـ گزینه ۳ صحیح می‌باشد.

۱۷۵ ـ گزینه ۳ صحیح می‌باشد.

شکل صحیح سؤال به صورت زیر بوده است:



۱۷۶ ـ گزینه ۱ صحیح می‌باشد.

۱۷۷ ـ گزینه ۱ صحیح می‌باشد.

۱۷۸ ـ گزینه ۲ صحیح می‌باشد.

۱۷۹ ـ گزینه ۱ صحیح می‌باشد.

۱۸۰ ـ گزینه ۱ صحیح می‌باشد.

۱۸۱ ـ گزینه ۳ صحیح می‌باشد.

۱۸۲ ـ گزینه ۲ صحیح می‌باشد.

۱۸۷ ـ گزینه ۳	صحیح می‌باشد.
۱۸۸ ـ گزینه ۴	صحیح می‌باشد.
۱۸۹ ـ گزینه ۴	صحیح می‌باشد.
۱۹۰ ـ گزینه ۱	صحیح می‌باشد.
۱۹۱ ـ گزینه ۲	صحیح می‌باشد.
۱۹۲ ـ گزینه ۱	صحیح می‌باشد.
۱۹۳ ـ گزینه ۳	صحیح می‌باشد.
۱۹۴ ـ گزینه ۱	صحیح می‌باشد.
۱۹۵ ـ گزینه ۱	صحیح می‌باشد.
۱۹۶ ـ گزینه ۴	صحیح می‌باشد.
۱۹۷ ـ گزینه ۲	صحیح می‌باشد.
۱۹۸ ـ گزینه ۳	صحیح می‌باشد.
۱۹۹ ـ گزینه ۱	صحیح می‌باشد.
۲۰۰ ـ گزینه ۴	صحیح می‌باشد.
۲۰۱ ـ گزینه ۳	صحیح می‌باشد.
۲۰۲ ـ گزینه ۳	صحیح می‌باشد.
۲۰۳ ـ گزینه ۲	صحیح می‌باشد.
۲۰۴ ـ گزینه ۱	صحیح می‌باشد.
۲۰۵ ـ گزینه ۲	صحیح می‌باشد.
۲۰۶ ـ گزینه ۱	صحیح می‌باشد.
۲۰۷ ـ گزینه ۴	صحیح می‌باشد.
۲۰۸ ـ گزینه ۴	صحیح می‌باشد.
۲۰۹ ـ گزینه ۴	صحیح می‌باشد.
۲۱۰ ـ گزینه ۳	صحیح می‌باشد.

۲۱۱ ـ گزینه ۲ صحیح می‌باشد.

۲۱۲ ـ گزینه ۳ صحیح می‌باشد.

۲۱۳ ـ گزینه ۱ صحیح می‌باشد.

۲۱۴ ـ گزینه ۳ صحیح می‌باشد.

۲۱۵ ـ گزینه ۲ صحیح می‌باشد.

۲۱۶ ـ گزینه ۳ صحیح می‌باشد.

۲۱۷ ـ گزینه ۲ صحیح می‌باشد.

۲۱۸ ـ گزینه ۱ صحیح می‌باشد.

۲۱۹ ـ گزینه ۴ صحیح می‌باشد.

۲۲۰ ـ گزینه ۲ صحیح می‌باشد.

۲۲۱ ـ گزینه ۱ صحیح می‌باشد.

۲۲۲ ـ گزینه ۴ صحیح می‌باشد.

۲۲۳ ـ گزینه ۱ صحیح می‌باشد.

۲۲۴ ـ گزینه ۴ صحیح می‌باشد.

۲۲۵ ـ گزینه ۱ صحیح می‌باشد.

۲۲۶ ـ گزینه ۲ صحیح می‌باشد.

۲۲۷ ـ گزینه ۱ صحیح می‌باشد.

۲۲۸ ـ گزینه ۱ صحیح می‌باشد.

۲۲۹ ـ گزینه ۲ صحیح می‌باشد.

۲۳۰ ـ گزینه ۲ صحیح می‌باشد.

۲۳۱ ـ گزینه ۲ صحیح می‌باشد.

۲۳۲ ـ گزینه ۳ صحیح می‌باشد.

۲۳۳ ـ گزینه ۱ صحیح می‌باشد.

۲۳۴ ـ گزینه ۴ صحیح می‌باشد.

۲۳۵ ـ گزینه ۴ صحیح می‌باشد.

۲۳۶ ـ گزینه ۲ صحیح می‌باشد.

۲۳۷ ـ گزینه ۲ صحیح می‌باشد.

۲۳۸ ـ گزینه ۳ صحیح می‌باشد.

۲۳۹ ـ گزینه ۴ صحیح می‌باشد.

۲۴۰ ـ گزینه ۴ صحیح می‌باشد.

۲۴۱ ـ گزینه ۲ صحیح می‌باشد.

۲۴۲ ـ گزینه ۳ صحیح می‌باشد.

۲۴۳ ـ گزینه ۲ صحیح می‌باشد.

۲۴۴ ـ گزینه ۱ صحیح می‌باشد.

پاسخ تشریحی دولتی ۸۳

۲۴۵ ـ گزینه ۴ صحیح می‌باشد.

۲۴۶ ـ گزینه ۴ صحیح می‌باشد.

۲۴۷ ـ گزینه ۱ صحیح می‌باشد.

۲۴۸ ـ گزینه ۴ صحیح می‌باشد.

۲۴۹ ـ گزینه ۱ صحیح می‌باشد.

۲۵۰ ـ گزینه ۳ صحیح می‌باشد.

۲۵۱ ـ گزینه ۱ صحیح می‌باشد.

۲۵۲ ـ گزینه ۳ و ۴ صحیح می‌باشد.

۲۵۳ ـ گزینه ۲ صحیح می‌باشد.

۲۵۴ ـ گزینه ۴ صحیح می‌باشد.

A	B	P	A < > 0
30	5	0	TRUE
15	10	10	TRUE
7	20	30	TRUE
3	40	70	TRUE
1	80	150	TRUE
0	160		FALSE

۳ ـ صورت سوال اشتباه است.

۲۵۸ ـ گزینه ۴ صحیح می‌باشد.

۲۵۹ ـ گزینه ۲ صحیح می‌باشد.

پاسخ تشریحی ۵۰٪ اول

آرایه

۱ ـ گزینه ۴ صحیح می‌باشد.

۲ ـ گزینه ۳ صحیح می‌باشد.

۳ ـ گزینه ۱ صحیح می‌باشد.

۴ ـ گزینه ۲ صحیح می‌باشد.

۵ ـ گزینه ۲ صحیح می‌باشد.

۶ ـ گزینه ۳ صحیح می‌باشد.

۷ ـ گزینه ۲ صحیح می‌باشد.

۸ ـ گزینه ۲ صحیح می‌باشد.

۹ ـ گزینه ۳ صحیح می‌باشد.

۱۰ ـ گزینه ۳ صحیح می‌باشد.

۱۱ ـ گزینه ۳ صحیح می‌باشد.

۱۲ ـ گزینه ۲ صحیح می‌باشد.

۱۳ ـ گزینه ۳ صحیح می‌باشد.

۱۴ ـ گزینه ۲ صحیح می‌باشد.

۱۵ ـ گزینه ۱ صحیح می‌باشد.

۱۶ ـ گزینه ۲ صحیح می‌باشد.

۱۷ ـ گزینه ۲ صحیح می‌باشد.

۱۸ ـ گزینه ۱ صحیح می‌باشد.

۱۹ ـ گزینه ۳ صحیح می‌باشد.

۲۰ ـ گزینه ۱ صحیح می‌باشد.

۲۱ ـ گزینه ۲ صحیح می‌باشد.

۲۲ ـ گزینه ۱ صحیح می‌باشد.

۲۳ ـ گزینه ۲ صحیح می‌باشد.

۲۴ ـ گزینه ۱ صحیح می‌باشد.

۲۵ ـ گزینه ۴ صحیح می‌باشد.

۲۶ ـ گزینه ۳ صحیح می‌باشد.

۲۷ ـ گزینه ۴ صحیح می‌باشد.

۲۸ ـ گزینه ۳ صحیح می‌باشد.

۲۹ ـ گزینه ۴ صحیح می‌باشد.

۳۰ ـ گزینه ۳ صحیح می‌باشد.

۳۱ ـ گزینه ۲ صحیح می‌باشد.

۳۲ ـ گزینه ۳ صحیح می‌باشد.

۳۳ ـ گزینه ۲ صحیح می‌باشد.

۳۴ ـ گزینه ۴ صحیح می‌باشد.

۳۵ ـ گزینه ۱ صحیح می‌باشد.

۳۶ ـ گزینه ۲ صحیح می‌باشد.

توابع بازگشتی

۳۷ ـ گزینه ۳ صحیح می‌باشد.

۳۸ ـ گزینه ۱ صحیح می‌باشد.

۳۹ ـ گزینه ۲ صحیح می‌باشد.

۴۰ ـ گزینه ۴ صحیح می‌باشد.

۴۱ ـ گزینه ۳ صحیح می‌باشد.

۴۲ ـ گزینه ۱ صحیح می‌باشد.

۴۳ ـ گزینه ۳ صحیح می‌باشد.

۴۴ ـ گزینه ۳ صحیح می‌باشد.

۴۵ ـ گزینه ۲ صحیح می‌باشد.

۴۶ ـ گزینه ۲ صحیح می‌باشد.

۴۷ ـ گزینه ۳ صحیح می‌باشد.

۴۸ ـ گزینه ۲ صحیح می‌باشد.

۴۹ ـ گزینه ۱ صحیح می‌باشد.

۵۰ ـ گزینه ۲ صحیح می‌باشد.

۵۱ ـ گزینه ۱ صحیح می‌باشد.