داده ساختارها و الگوریتم ها

مدرس :
محدثه محمدپور

فهرست مراجع :

۱. ساختمان داده ها به زبان , ++ Cنوشته الیس هورویتز سارتاج ساهنی، و دی. مژین ، ترجمه امیر علیخانزاده، انتشارات خراسان

۲. لیپشوتز ، سیمور ، اصول ساختمان داده ها ، ترجمه حسین ابراهیم زاده ، دانشگاه هرمزگان

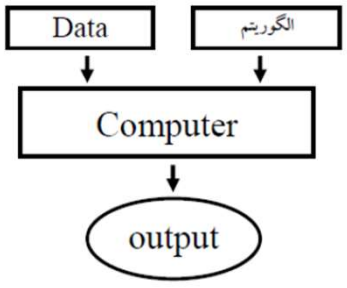
۳. محمد قدسی و آیدین نصیری شرق، " ۶۰۰مسئله ی چندگزینه‌ای از داده ساختارها و الگوریتم‌ها،" چاپ ششم، انتشارات فاطمی، ۱۳۹۷.

[۴]T. Cormen, C. Leiserson, R. Riverst, and C. Stein. Introduction to Algorithms. 3rd edition, MIT Press, 2011

فهرست مطالب :

فصل اول:
مفاهیم الگوریتم و مقدمات

برای انجام هر کاری در کامپیوتر نیاز به دو عنصر مهم داریم :



تعریف خصوصیات الگوریتم :

۱-ورودی (حداقل صفر ورودی)

۲-خروجی (حداقل یک خروجی)

۳-قطعی و عدم ابهام

٤-محدودیت (پایان پذیر بودن)

٥-کارایی یا انجام پذیری

تفاوت برنامه و الگوریتم :

تعریف ساختمان داده :

انواع ساختمان داده:

۱-ایستا-۲-نیمه ایستا-۳-پویا

۱-ایستا:

اولیه :داده صحیح و داده اعشاری.
غیر اولیه :آرایه-رکورد-رشته-داده کارکتری.

۲-نیمه ایستا

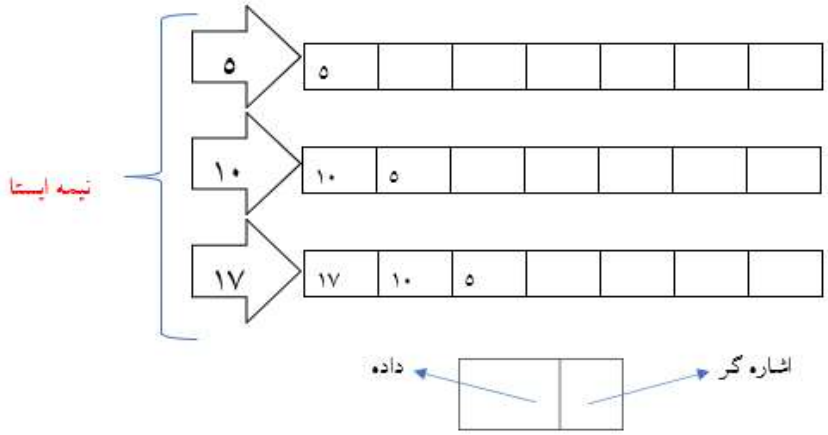
پشته :پشته یا Stackبه ساختمان داده ای گفته ای می شود که مجموعه ای از المان ها را بر اساس اصل(LIFOاولویت خروج با عناصر تازه وارد)در خود نگه داری می کند و از دو عمل Pushبرای افزودن آیتم و Popبرای حذف آیتم پشتیبانی می کند.
صف :صف یا Queueعنوان ساختمان داده ای است که مجموعه ای از المان ها را بر اساس اصل (FIFOخروج به ترتیب ورود)درخود نگه داری می کند.

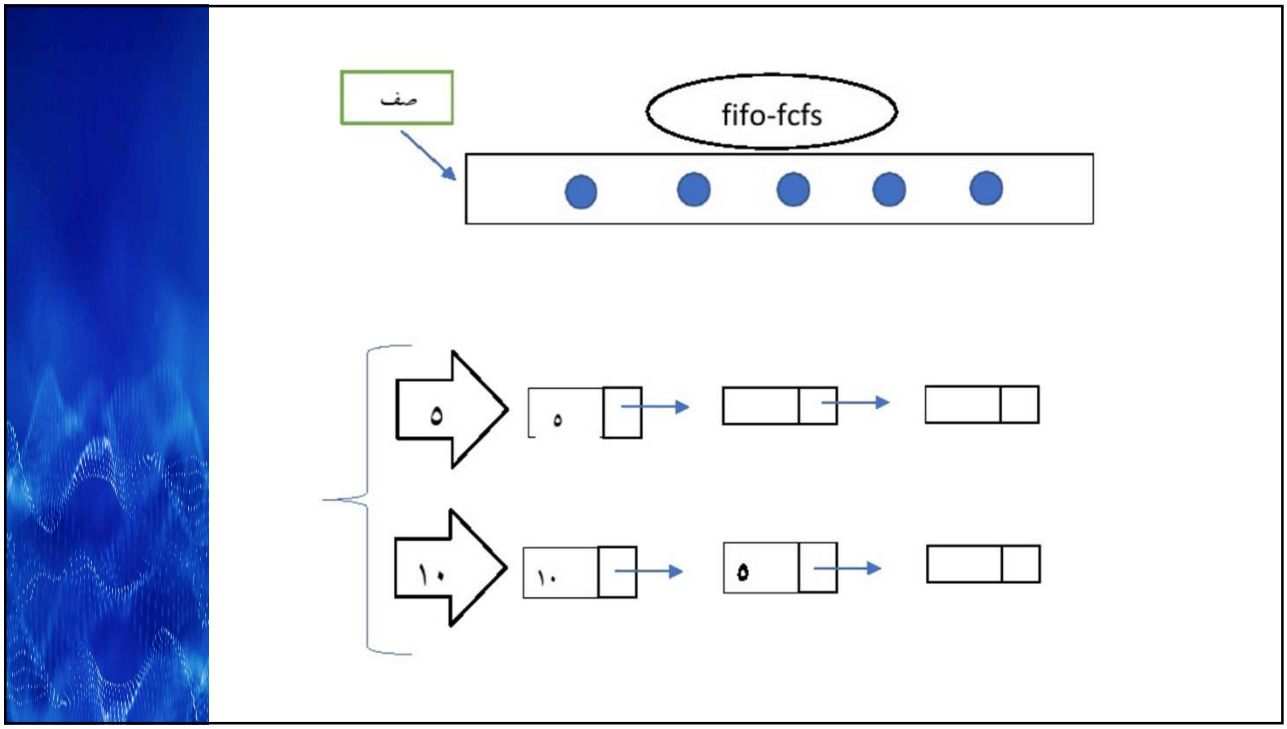
۳-پویا

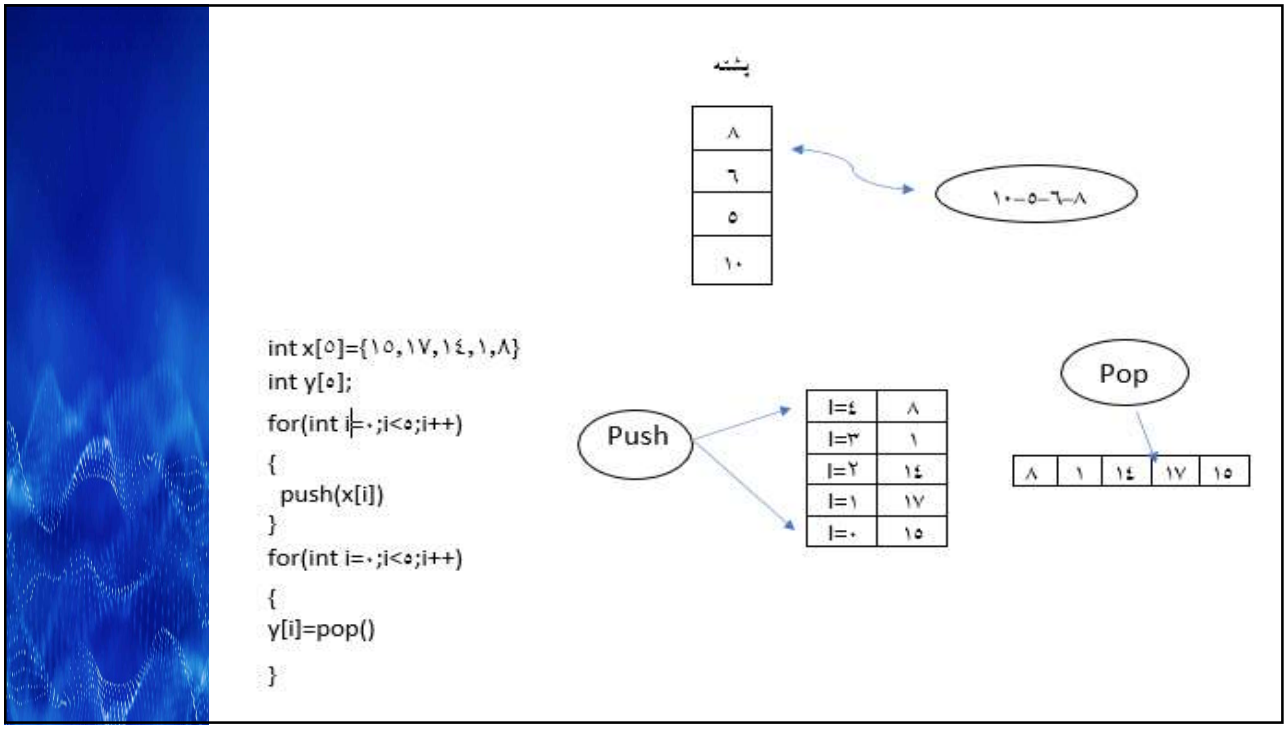
خطی: لیست پیوندی
غیرخطی :درخت و گراف

داده های ایستا :

داده پویا :




صف
fifo-fcfs
٥
٥
١٠
١٠
٥


پشته
٨
٦
٥
١٠
٨-٦-٥-١٠
int x[٥]={١٥,١٧,١٤,١,٨}
int y[٥];
for(int i=٠;i<٥;i++)
{
push(x[i])
}
for(int i=٠;i<٥;i++)
{
y[i]=pop()
}
Push
i=٤ ٨
i=٣ ١
i=٢ ١٤
i=١ ١٧
i=٠ ١٥
Pop
٨ ١ ١٤ ١٧ ١٥

مقدمه ای بر تحلیل پیچیدگی زمان و مرتبه اجرایی الگوریتم ها :

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

برای بررسی زمان اجرای یک الگوریتم به تابعی به نام T(n) که تابع زمان الگوریتم می شود درنظرمی گیریم که در آن (n) اندازه ورودی مسئله است که ممکن است شامل چندین داده ورودی باشد .

شمارش گام ها یا مراحل یک برنامه:

گام های یک برنامه با استفاده ازمراحل کلی و قواعد کلی زیر قابل محاسبه خواهد بود.

۱. تعاریف زیر برنامه ها و توابع دارای گام صفر هستند.
۲. هر بلاک شامل باز و بسته کردن { }دارای صفر گام است.
۳. تعاریف متغیر در صورتی که مقدار اولیه برای آنها درنظر گرفته نشود ۰ و اگر گرفته در نظر گرفته شود ۱ خواهد بود.
٤. درهردستوراجرایی به ازای هربار اجرا دارای یک گام است.

Procedure	P(.....)	۰ ←
Function	F(.....)	۰ ←
Void	F(.....)	۰ ←
	int x;	۰ ←
	int x=۱;	۱ ←

مثال:

١)x=٠ ← ١

٢)for(i=٠;i<n;i++) ← n+ ١

٣){ ← ٠

٤)x++ ← ١*n=n

٥)} ← ٠

جواب: ٢n+٢ مرتبه

مثال:

for(i=١;i≤n;i++) ← n+١

{ ← ٠

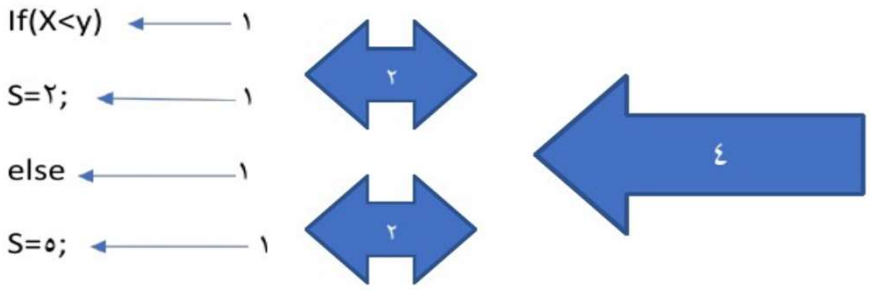
S=s+١; ← n

} ← ٠

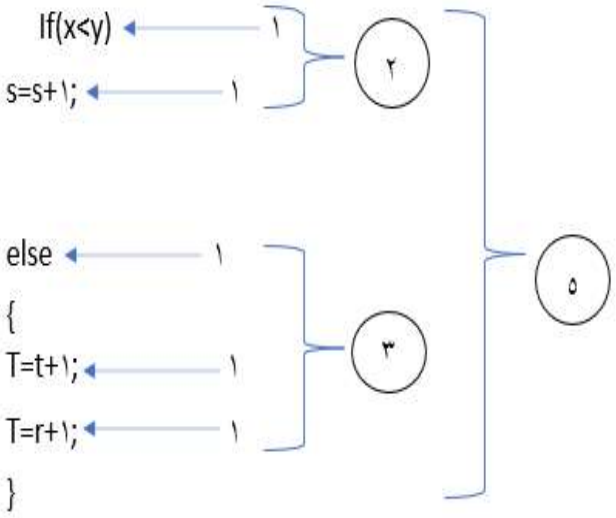
جواب: ٢n+١=(n)+(n+١)

عبارت شـرطی if: عبارت شـرطی یک گام دارد اما با توجه به درسـت و غلط بودن شـرط چون جملات مختلفی ممکن است اجرا شود کل دستور شرطی و به درست و غلط بودن شرط بستگی خواهد داشت

مثال:



مثال



دستور حلقه for:

مهم ترین قسمت در شمارش گام های یک برنامه حلقه for است که ترتیب زیر گام آن را محاسبه می کنیم.

الف)ابتدا تعداد تکرار حلقه را محاسبه می کنیم.

ب)حلقه forبه تعداد تکرار +یک گام اختیار می کند.

ج)جملات تکرار شونده داخل حلقه forبه تعداد تکرار گام اختیار می کند .

مثال:

```
for(i=a;i<b;i++)  <-- (b-a)+1
{
x++  <-- (b-a)
}
```

جواب: ((b-a)+1)+(b-a)=2(b-a)+1

مثال

```
for(i=a;i<=b;i++)  <-- (b-a)+1+1
{
x++  <-- (b-a)+1
}
```

جواب: ((b-a)+1+1)+(b-a)+1=2(b-a)+3

```
for(i=۱۰;i>۱;i--)  ← (۱۰-۱)+۱=۹+۱=۱۰
{
x++  ← ۹
}
```

جواب: ۱۰+۹=۱۹

حلقه تودرتو :

برای محاسبه گام حلقه های تودرتو به صورت زیرعمل میکنیم.

۱) از بیرون ترین حلقه شروع میکنیم.

۲)تعداد تکرار هر حلقه را برای تمام حلقه ها و دستورات تکرار شونده پایین در نظر گرفته و تعداد تکرار ۱ +را برای خود حلقه در نظر میگیریم.

۳)قسمت ۲را برای تمامی حلقه ها انجام می دهیم.

```
for(i=۰;i<n;i++)  → n+۱
for(j=۰;j<m:j++)  → n*(m+۱)
{
S=s+۱  → m*n
}
```

جواب= ((n+۱)+(n*(m+۱))+(m*n)=n+۱+nm+n+nm=۲nm+۲n+۱

تعداد گام حلقه های while مانند for محاسبه می شود یعنی گام حلقه یک واحد از تعداد تکرار حلقه بیشتر است.

مثال:

```
i=۱;          ──►  ۱
while(i<=n)   ──►  n+۱
{
S=s+۱;        ──►  n
i=i+۱;        ──►  n
}
```

جواب= ۱+n+۱+n+n=۳n+۲

بررسی کارایی الگوریتم:

۱) bigO

۲) BigΩ

۳) Θ

برای بررسی کارایی الگوریتم نماد هایی معرفی شده است که به بررسی آنها می پردازیم.

۱) **bigO**:برای بررسی میزان رشد توابع زمانی الگوریتم نماد bigO را به کارمی گیرند و آن را با علامت **O** بزرگ نمایش می دهند.

۲) **تعریف نماد O بزرگ**:گوییم **T(n) ∈ O(f(n))** است اگر فقط اگر ثابت C و ثابت صحیح **(n)** وجود داشته باشد که برای همه ی مقادیر **n0**=<n داشته باشیم **T(n)<=C(f(n))**

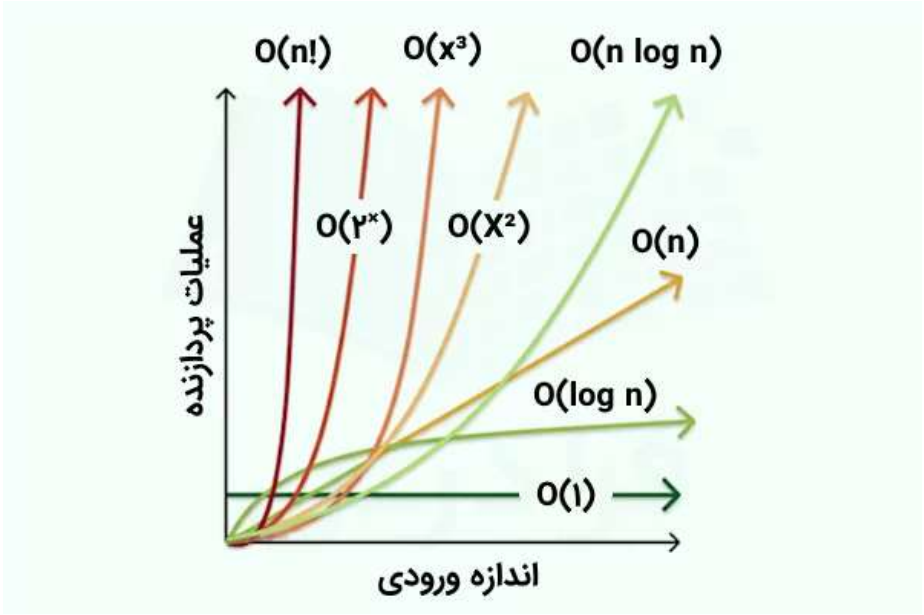
T(n) ∈ O(f)n)) ←→ ∃ c,n۰>۰ ⊎ n ≥ n۰ T(n)≤ cf(n)

T(n)=2n+2	O(n)
T(n)=2n^2+4n	O(n^2)
T(n)=3n^3+3n	O(n^3)
T(n)=4n+5 n log n+2	O(n log n)
T(n)= 2^ n +10n	O(2^ n)
T(n)=log 2n+1	O(log n)

جدول زیر مرتبه اجرایی توابع مهم را ترتیب صعودی مطرح کرده است.

ثابت لگاریتمی رادیکالی مرتبه۲ kمرتبه نمایی توانی

مرتبه اجرایی: O(۱)<log(n)<√n<n<n log n<n^۲<n^۲ log n<…<n^k<۲^n<n!<n^n



مثال:

درستی یا نادرستی عبارت های زیر را مشخص کنید:

$T(n)=۲n+۱ \in O(n^{۲})$ √

$T(n)=۵n^{۲}+n+۱ \in O(n)$ ×

$T(n)=(٤*۲^{n}*n^{۲}) \in O(۲^{n})$ √

قضیه : اگر $T(n)=Am\ R^{m}+\ldots A1\ n+A0$ زمان اجرای این الگوریتم عبارت است از:

$T(n) \in O(n^{m})$

BigΩ:

$T(n) \in \Omega(f(n)) \leftrightarrow \exists\ C\backslash c, n۰ > ۰ \forall n > n۰\ T(n) \rangle C\ f(n)$ اگر ثابت صحیح $C,n0$ داشته باشیم. **$n0 \geq m$. وجود داشته باشد که به ازای همه ی مقادیر**

$T(n) \geq C\ f(n)$

(Ω): یک کران پایین زمان اجرا برای $T(n)$ ارائه میدهد

در حالت کلی میتوان گفت که $f(n)(\Omega)$ بهترین حالت اجرا برای یک الگوریتم است.

مثال:

$T(n)=٦n+٤ \in \Omega\ (n)$ √

$\in \Omega\ (n^{2})$

نماد Θ:می گوییم

$$T(n) \in \Theta(f(n)) \leftrightarrow \exists\, c1, c2, n_0 > 0 \;\forall\, n \geq n_0 \; c1 * g(n) \leq T(n) \leq c2\, f(n)$$

اگر ثابت های c2و۱ c,وثابت صحیح n 0وجود داشته باشد به طوری که برای همه ی مقادیر $n_1 \geq 0$ بتوان‌گفت

$$c_1\, f(n) \leq T(n) \leq c_2\, f(n)$$

با استفاده از نماد Θ تابع T(n) هم ازبالا و هم ازپایین محدود می شود درجه رشد تابع T(n)f(n)۱ است نماد Θزمانی به کار می رود که نرخ رشد دو تابع مساوی باشد

نماد Θدقیق ترین بیان برای رشد 1 تابع است.

مثال :

$$T(n) = \frac{1}{2} n^2 - 3n \in \theta(n^2)$$

$$T(n) \in \theta(n^2)$$

$$c1\, f(n) \leq T(n) \leq c2\, f(n)$$

$$c2 \geq \frac{1}{2} \qquad n \geq 1$$

$$c1 = \frac{1}{4} \qquad n \geq 7$$

$$\in \theta(n^2)$$

$$\frac{c1 n^2}{n^2} \leq \frac{\frac{1}{2} n^2 - 3n}{n^2} \leq \frac{c2\, n^2}{n^2}$$

جواب: $c1 \leq \frac{1}{2} - \frac{3}{n} \leq c2$

نقاط رشد تابع

۱. لگاریتم Nبا هر پایه ای رشدشان باهم برابر است به شرطی که پایه پایشان ثابت باشد.

$\text{Log}_a\ n = \text{Log}_b\ n \rightarrow a,b>1$

۲. n به توان هر عدد ثابت رشدش از log n به هر توان ثابتی بیشتر است.

$n^2 > \log n^b \rightarrow n > 0$

۳. logn با هر توانی از log logn با هر توانی بیش تر است به شرطی که توانش ثابت باشد.

$\log n > \log \log n$

مثال؟

$N^{\frac{1}{2}} > \text{Log}N$ $\qquad$ $N^2 > \text{Log}^3 N$

مثال؟

$\text{Log}\frac{1}{2} > Log\ Logn^{300}$

توابع نمایی رشدشان از توابع چند جمله ای بیشتر است

$A^n > N^b \rightarrow 2^h > N^3$ $\qquad$ $a, b > 1$

$logN^{logn} < \log n^n$ $\qquad$ $3^n > n^8$

توابع و زیر برنامه های بازگشـــتی: به هر تابع یا برنامه ای که بتواند داخل بدنه اش خودش را دوباره فرا خوانی کند یا صدا بزند تابع یا زیر برنامه بازگشتی میگویند.

. ۱بدون بازگشتی(عادی)

2. بازگشتی

مثال:

```
int n;
int f=۱;
cin>>n;
for(i=۱;i<=n;i++)
f=f*i;
cout<<f;
```

مثال :

```
fact(n)
{
cin>>n;
if(n<=۱)
 Return ۱;
 else
Return n*fact(n-۱);
}
```

توابع بازگشتی دو ویژگی دارند:

۱. تابع خودش خودش را صدا می زند البته اغلب با آرگومان های‌کوچکتر
۲. . یک شرط جهت اتمام فراخوانی ها وجود دارد.

نکته:

مراحل بازگشت در استک یا پشته نگه داری می شود با پرشدن استک هنگام خطای داده stack over flow نشان داده می شود .
پشته همیشه از پایین به بالا پر میشود

مثال:

در برنامه زیر خروجی f(۳,٦) را بدست آورید.

```
int f(int m,int n)
{
if(m==١||n==٠||m==n)
 Return ١;
else
Return f(m-١,n)+f(m-١ , n-١)
}
```

جواب:

f(۳,٦)
f(2,٦)+f(2+5)
f(1,6)+f(1,5)+f(1,5)+f(1,4)=4

مثال:

M(A,B) → M(B,A) → A → A= A%B

A<B

B=٠ ⟹ M(٢٠,٢٨)

A > B

M(٢٠,٢٨)

M(٢٨,٢٠) ٢٨٪٢٠=٨

M(٢٠,٨) ٢٠٪٨=٤

M(٨,٤) ٨٪٤=٠

M(٤,٠) A=٤

مثال:

```
int test(int x,int y)
{
if(x<=y||y==٠)
test=x;
else
if(y==١)                                  test(x=٥ , y=2)
test=test(x-١,y)+1;
 else
   test=test(test(y,x),y-1)+2;
```

جواب=٤

Test(x=5,y=2)
Test(test(2,5),1)+2
Test(2,1)+2
Test(1,1)+1

تمرینات پایانی فصل ۱

سوالات تستی:

۱- پیچیدگی زمانی الگوریتم خاصی به صورت چند جمله ای زیر معین شده است. در این صورت زمان اجـرای آن کدام است؟ (فرض کنید n بسیار بزرگ است) (ارشد IT – آزاد ۸۹)

$$P(n) = 2^n + 100n^3 + n\log_2^n$$

۱) $O(100n^3)$ ۲) $O(n\log_2^n)$

۳) $O(2^n + n^3)$ ۴) $O(2^n)$

حل: گزینه ۴ درست است.

بزرگترین جمله در عبارات داده شده 2^n می باشد.

مرتبه زمانی شبه کد زیر کدام گزینه است؟ (ارشد IT – دولتی ۸۹)

```
for ( i=1; i<=n; i++)
  for ( j=1; j<=n; j=j+i)
        x=x+1;
```

۱) n ۲) $n\log n$ ۳) $\log n$ ۴) n^2

حل: گزینه ۲ درست است.

دستور x=x+1 ، به ازای i=1 ، n بار اجرا می شود. به ازای i=2 ، $\frac{n}{2}$ بار اجرا می شود و .... بنـابراین تعـداد اجـرای جملـه x=x+1 برابر است با:

$$n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \ldots + \frac{n}{n} = n(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}) \approx n.\ln n \Rightarrow O(n.\log n)$$

- فرض می گردد n و m مقادیری مثبت و بزرگتر از یک را دارا می باشند. تابع زیر چه عملی را انجام مـی دهـد؟ (مهندسی کامپیوتر - آزاد ۸۹)

```
int  f(int n , int m){
  if (m==2)
       return  n;
  else
       return  n*f(n,m-1);
}
```

۱) n^m ۲) n^{m-1} ۳) n^{m+1} ۴) m^n

حل: گزینه ۲ درست است.

در واقع همان تابع توان است، با این تفاوت که شرط اولیه آن تغییر کرده است.اگر شرط توقف به جای $f(2)=n$ برابر $f(1)=n$ بود، آنگاه گزینه ۴ درست بود.

۴- هزینه ی زمانی تکه برنامه زیر کدام است؟ (ارشد IT – دولتی ۹۹

```
int   i=n;
while ( i>1){
   i /= 2;
   j = i;
  while(j>1)
    {
      j /= 3;
    }
}
```

۱) $O(\lg n)$ ۲) $O(\lg^2 n)$ ۳) $O(n)$ ۴) $O(n^2)$

حل: گزینه ۲ درست است.

حلقه اول از مرتبه $O(\log_2^n)$ و حلقه دوم از مرتبه $O(\log_3^n)$ اسـت و چـون دو حلقـه تـو در تـو اسـت، جـواب ضـرب آنهـا $O(\log_2^n \times \log_3^n)$ است.

لازم به ذکر است که لگاریتم ها در هر پایـه ای از مرتبـه یکـدیگر هسـتند و مـی تـوان بـه جـای $O(\log_3^n)$ از $O(\log_2^n)$ استفاده کرد. بنابراین داریم:

$$O(\log_2^n \times \log_3^n) = O(\log_2^n \times \log_2^n) = O(\log_2^n)^2 = O(\lg^2 n)$$

سوالات تشریحی :

۱-زمان اجرا (T(n)) برای هر یک از الگوریتم های زیر را محاسبه نمایید .

الف)

```
(1) int Factorial(int n)
{
(2) int fact= 1 ;
(3) for( int i=2 ; i<= n ; i++ )
(4) fact*= i ;
(5) return fact ;
}
```

سطر	هزینه	تعداد
2	C_1	۱
3	C_2	n
4	C_3	n-۱
5	C_4	۱

$$T(n) = C_1 + C_2 n + C_3(n-1) + C_4$$

C را بیشترین مقدار برای C۴و C۳ و C۲ و C۱ در نظر می گیریم بنابراین خواهیم داشت:

$$T(n) = C(2n+1).$$

ب)

```
void Add(a, b, c, int m,n)
{
(1) for(int i=0 ; i<n ; i ++)
(2) for(int j=0 ; j<m ; j ++)
(3) c[i][j]= a[i][j] + b[i][j] ;
}
```

سطر	هزینه	تعداد
1	C_1	$n+1$
2	C_2	$n(m+1)$
3	C_3	nm

$$T(n) = C_1(n+1) + C_2 n(m+1) + C_3 nm$$

$$T(n) = C(n+1+n(m+1)+nm)$$
$$= C(2nm+2n+1)$$

۲-مرتبه زمانی T(n) تعدادی الگوریتم وجود دارد ، پیچیدگی زمانی آن را پیدا کنید.

i) $T_1(n) = 2n^2 + 4n$

$T_1(n) = 2n^2 + 4n \le 3n^2$

که در آن اگر $C = 3$ و $n_0 = 4$ باشد آنگاه $T_1(n) \in O(n^2)$ می‌باشد.

ii) $T_2(n) = 3n^3 + 3n$

$T_2(n) = 3n^3 + 3n$

$\le 4n^3$

که در آن اگر $C = 4$ و $n_0 = 2$ باشد، آنگاه $T_2(n) \in O(n^3)$.

iii) $T_{۳}(n) = ٤n + ٥n\,Logn + ٢$

$$T_{۳}(n) = ٤n + ٥n\,Log\,n + ٢ \le ٤n\,Log\,n + ٥n\,Log\,n + ٢n\,Log\,n$$
$$= ١١n \times Log\,n$$

که در آن اگر $C = ١١$ و $n_{۰} = ١$ باشد، آنگاه $T_{۳}(n) \in O(n\,Log\,n)$ خواهد بود.

توجه داشته باشید که می‌توانید ضرایب مختلفی از C را به‌دست آورید.

۳-درستی یا نادرستی عبارت های زیر را مشخص کنید .

i) $T(n) = (٢n + ١) \in O(n^{٢})$

i) $T(n) = (٢n + ١)$

$$\le ٢n^{٢}$$

که در آن اگر $C = ٢$ و $n_{۰} = ٢$ باشد آنگاه $T(n) \in O(n^{٢})$ خواهد بود. بنابراین رابطه بالا یک رابطه صحیح می‌باشد.

ii) $T(n) = (5n^2 + n + 1) \in O(n)$

ii) $T(n) = (5n^2 + n + 1)$

$\leq 6n^2$

همانطور که ملاحظه می‌کنید نمی توان C و n_0 معرفی کرد که رابطه

$5n^2 + n + 1 \leq Cn$

به ازای $n \geq n_0, C$ همواره برقرار باشد به عبارت دیگر $T(n)$ به هیچ وجه در حالت کلی نمی‌تواند کمتر از Cn باشد. در نتیجه رابطه بالا یک رابطه نادرست می‌باشد.

٤-زمان اجرای $T(n)$ الگوریتمی محاسبه شده است $\Omega\ f(n)$ آن را به دست آورید.

$T(n) = an^2 + bn + c \quad \text{and} \quad a, b, c > 0$

$an^2 + bn + c > an^2 + b + c$

$> an^2$

بنابراین اگر $n_0 = 1$ و $C = a$ باشد، آنگاه $T(n) \in \Omega(n^2)$ خواهد بود.

$$T(n) = n^{۴} + ۵n^{۲}$$

$$T(n) = n^{۴} + ۵n^{۲} \geq n^{۴}$$

بنابراین اگر $n_۰ = ۱$ و $C = ۱$ باشد، آنگاه $T(n) \in \Omega(n^{۴})$ خواهد بود.

پرسش های فصل ۱

۱- فرض کنید که A,B عدد 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)

۲-درستی یا نادرستی عبارت های زیر را مشخص کنید

iii) $T(n) = (۴ * ۲^{n} + n^{۲}) \in O(۲^{n})$

iiii) $T(n) = ۳n + ۲ \notin \Omega\left(n^{۲}\right)$

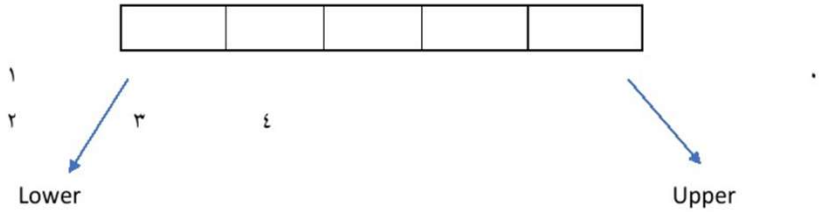
آرایه: مجموعه ای از داده های پشت سر هم که همگی از یک نوع بوده و از آدرس مشخص شروع می شوند .عناصر یک آرایه با اندیسشان قابل دستیابی هستند آرایه میتواند یک بعدی دو بعدی یا چند بعدی باشد.

آرایه یک بعدی :آرایه یک بعدی برای ذخیره سازی مجموعه ای از عناصر هم نوع به کار میرود.

عناصر آرایه یک بعدی در محل های متوالی حافظه ذخیره میشوند در این آرایه برای دستیابی به عنصری از آرایه از اندیس استفاده می شود

آرایه ۱ بعدی	int A[۱۰] A[L۱.....u۱]
آرایه ۲بعدی	int A[۱۰][20] A[L۱.....u۱ , L۲.....u۲]
آرایه n بعدی	A[L۱..u۱] , [L۲..u۲ , Ln...un]

نکته :کوچکترین انـدیس آرایـه حـد پـایین آرایـه یـا Lowerیـا (L) بزرگترین مقـدار انـدیس آرایـه را کران بـالای آرایه میگویند و با Upper یا (U)نشان میدهند.



تعداد عناصر یک آرایه :تعداد عناصر یک آرایه برابر است (u-L) می باشد.

A=[L۱.....u۱] تعداد خانه u۱-L۱+۱

A=[L۱.....u۱ , L۲.....u۲] تعداد خانه (u۱-L۱+۱)*(u۲-L۲+۱)

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

فضای آرایه=تعداد عناصر آرایه *نوع عناصر آرایه

مثال:

int A[۱۰]; 10*4=40 byte

int A[-۱...3 , 2...٥] حافظه=20*4=80 byte

محاسبه آدرس خانه : (i)اگر هر عنصر از آرایه با نام A به اندازه سایز (N)biteفضا اشغال کند محل عنصر **i** به صورت زیر محاسبه می شود.

آدرس اولین محل= Loc(i)=basic(a)+i*size

یا به این صورت هم می توان نوشت:

آدرس شروع آرایه ⟵ آدرس خانه A[i]=(i-L۱)*n+∝

مثال:

آرایه A مساوی[۲۰...۵] تعریف شده است اگر این آرایه از آدرس ۱۰۰ حافظه به بعد قرار گرفته باشد آدرس خانه[۱۶]A کدام است؟

تعداد عناصر آرایه را نیز به همراه فضای اشغال شده توسط آرایه بدست آورید.

A[i]=(i-L۱)*n+∝

A[۱۶]=(۱۶-۵)*٤+۱۰۰==۱۱*٤+۱۰۰=۱٤٤

تعداد عناصر آرایه=u۱-L۱+۱=۲۰-۵+۱=۱۶

فضای ذخیره سازی= تعداد عناصر آرایه* فضای نوع آرایه=۱۶*٤=٦٤

فرض کنید آرایه A به صورت[۱۰]A float تعریف شده و عناصر این آرایه از خانه ۳۰۰۰ به بعد در حافظه قرار گیرند. با فرض اینکه سایز هر عدد اعشاری float ٤ byte است آدرس خانه A=[۷] را محاسبه کنید.

A[i]=(i-L۱)*n+∝

A[۷]=(۷-۰)*٤+۳۰۰۰=۳۰۲۸

محاسبه آدرس دلخواه از یک آرایه مشخص کردن موقعیت عناصر یک آرایه با توجه به نحوه دخیره سازی آنها به یکی از دو روش سطری و ستونی صورت می‌پذیرد.

۱۵	۷	۱۲	۷۰
۱۸	۹	۳	۶۱
۱۹	۲۴	۶	۶۵
۲۰	۳۴	۵۲	۴۴
۱	۱۶	۶۱	۳۹

ذخیره سطری

سطر اول
سطر دوم
سطر سوم
سطر چهارم
سطر پنجم

ستون اول	ستون دوم	ستون سوم	ستون چهارم

ذخیره ستونی

نکته: وقتی عنصر ما مربعی باشد و عناصر هم فرد باشند میتوان گفت که عنصر وسطی هم از لحاظ ستونی و سطری با هم برابر است.

قرار داد : آلفا (α) آدرس شروع آرایه است و اگر بیان نشود (۰) در نظر گرفته خواهد شد.

بتا (β) فضای نوع عناصر آرایه است و اگر بیان نشود (۱) در نظر گرفته خواهد شد.

محاسبه آدرس A=(i,j,k) در آرایه A[L۱...U۱,L۲...U۲,L۳...U۳] ؟

$$\alpha + [(i-L۱)(U۲-L۲+۱)(U۳-L۳+۱)+(j-L۲)(U۳-L۳+۱)+(k-L۳)]*\beta$$

سطری $\alpha + [(i-L1)(U2-L2)+1)(U3-L3+1)+(j-L2)(U3-L3+1)+(k-L3)]* \beta$

سه بعدی[I,j,k]

ستونی $\alpha + [(k-L3)(U2-L3+1)(U1-L1+1)+(j-L2)(U1-L1+1)+(i-L1)]* \beta$

دو بعدی[i,j]

سطری $\alpha + [(i-L1)*(u2-L2+1)+(j-L2)]* \beta$

ستونی $\alpha + [(j-L2)*(u1-L1+1)+(i-L1)]* \beta$

ماتریس[۸],[۳۰]int x را در نظر بگیرید که آدرس اولیه آن یعنی ۲۰۰=x۱ در نظر گرفته شده است مطلوب است محاسبه آدرس سطری A=[۱۲][٤] β=٤

$A[12][4]= \alpha + [(i-L1)*(u2-L2+1)+(j-L2)]* \beta$

۲۰۰+[(۱۲-۰)*(۷-۰+۱)+(٤-۰)]*٤=۲۰۰+٤۰۰=٦۰۰

آرایه[۸...۱ , ۳...۱] از آدرس ۳۰۰۰ به بعد ذخیره شده است و هر خانه ٤ بایت اشغال کرده است مطلوب است آدرس A[۲,۳] به صورت سطری و ستونی محاسبه کنید.

α=۳۰۰۰ β=٤

سطری= A[۲,۳]=۳۰۰۰+[(۲-۱)(۸-۱+۱)+(۳-۱)]*٤=۳۰٤۰

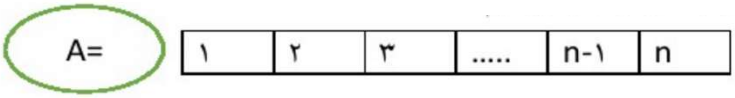
ستونی=A[۲,۳]=۳۰۰۰+[(۲-۱)(۳-۱+۱)+(۳-۱)]*٤=۳۰۲۸

جست و جو در آرایه :به طور کلی برای جست و جوی یک عنصر دلخواه در آرایه ها دو روش اساسی وجود دارد.

. ۱روش جست وجوی خطی(ترتیبی)

. ۲جست و جوی دودویی(باینری)

روش جست و جوی خطی(ترتیبی):در این روش برای جست و جوی داده Xدر آرایه A[۱...n] جست و جو را از یکی از دو طرف آرایه (پیشفرض)آغاز میکنیم و داده ی مورد نظر را پشت سر هم با عناصر آرایه مقایسه می کنیم تا اینکه یا داده مورد نظر پیدا شود یا به انتهای آرایه برسیم



```
int seg-search(int A[] ; int n , int item)
{
for(int i=۰ ; i<n ; i++)
{
 if(a[i]==item)
return(i);
}
return(-۱);
}
```

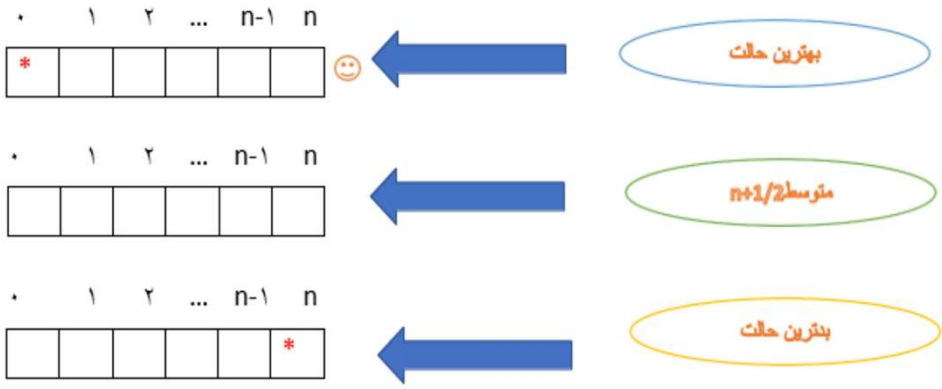
برنامه جست و جوی خطی

محاسبه زمان پیچیدگی:

۱. بهترین حالت $T(n) = O(1) \in$۱

۲. متوسط $T(n) = O(n) \in n * \frac{1}{۲}$

۳. بدترین حالت $T(n) = O(n) \in n$



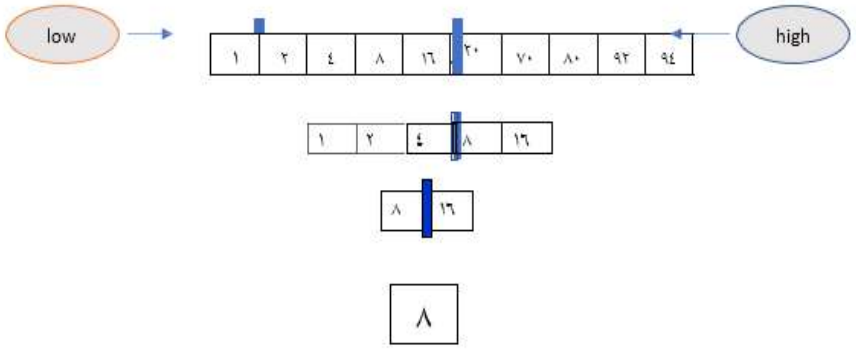
جست و جوی دودویی(باینری)

جست و جوی دودویی از یک آرایه مرتب امکان پذیر است که در ان هر آرایه به دو قسمت مساوی تقسیم می شود و در هر قسمت به جست و جو می پردازیم.

با توجه به بزرگتر یا کوچکتر بودن آیتم مورد نظر از بخش های انتخاب شده فضای جست و جو محدود تر شده و در نهایت عنصر مورد نظر پیدا می شود.

```
int binary-search(int a[] , int n , int item)
{
int mid , flag=۰ , low=۰, high=n-۱;
while(low<=high && !flag)
{
mid=(high +low)/2;
}
if(item<a[mid])
{
high=mid-۱;
else if(item>a[mid])
{
low=mid+۱;
}
else
{
flag==۱; returm mid;
}
return -۱;
}
```

میخواهیم عنصر ۸ را با استفاده از جست و جوی دودویی از آرایه بدست اوریم:



بهترین حالت: $T(n) \in 1$

متوسط : $\log \in O(\log n)$

بدترین حالت : $[\log + 1] + \in O(\log n)$

ماتریس ها

به هر آرایه دو بعدی m*nیک ماتریس یا جدول با mسـطر و n سـتون گفته می شـود که تعداد m*nخانه در آن وجود دارد.

عملیات رایج در ماتریس:

۱–جمع دو ماتریس :

برای جمع دو ماتریس A,B باید اندیس آنها یعنی اندازه سطر و ستون آن دو مانند هم باشد

۲- ضرب دو ماتریس :برای اینکه ضرب دو ماتریس A,Bامکان پذیر باشد باید ستون ماتریس Aبا سطر ماتریس B یکسان باشد

A [ ]$_{۳*٤}$ * B [ ]$_{٤*٥}$ A*B=C

خواص ضرب ماتریس ها:

۱. ضرب ماتریس خاصیت جابه جایی ندارد.

۲. ضرب ماتریس ها خاصیت شرکت پذیری دارند.

مثال:

ضرب سه ماتریس $C_{۲۰*٤}$ $B_{۱۰*۲۰}$ $A_{٥*۱۰}$را به چه ترتیبی انجام دهیم تا سریع تر اجرا شوند؟

(A*B)*C ⟶ ۱٤۰۰

A٥*۱۰ * B۱۰*۲۰ = AB٥*۲۰ ⟶ ٥*۱۰*۲۰=۱۰۰۰

(AB)٥*۲۰ * C۲۰*٤ = ABC٥*٤ ⟶ ٥*۲۰*٤=٤۰۰

B۱۰*۲۰*C۲۰*٤=BC۱۰*٤ ⟶ ۱۰*۲۰*٤=۸۰۰

A٥*۱۰ * BC۱۰*٤ =ABC٥*٤ ⟶ ٥*۱۰*٤=۲۰۰

۸۰۰+۲۰۰=۱۰۰۰

۳-ترانهاده یک ماتریس :اگر جای سطر و ستون یک ماتریس را عوض کنیم ماتریس ترانهاده می شود.

۰	۳	۱
۱	۲	۱۵
۱	۵	۱۲
۳	۱	۹
٤	۰	۱۸

0	1	1	3	4
3	2	5	1	0
1	15	12	9	18

انواع ماتریس:

ماتریس مربع: برای هر ماتریس مربع روابط زیر برای اندیس خانه ها وجود دارد.

۰و۰

٤و۰

۱	۲	۳	٤	
٦	۷	۸	۹	۱۰
۱۱	۱۲	۱۳	۱٤	۱۵
۱٦	۱۷	۱۵	۱۹	۲۰
۲۱	۲۲	۲۳	۲٤	۲۵

عناصر روی قطر اصلی: i==j

عناصر بالای قطر اصلی: i<j

عناصر پایین قطر اصلی: i>j

ماتریس پایین مثلثی و بالا مثلثی:

ماتریس مربعی است که اگر عناصر زیر قطر اصلی ۰باشند ماتریس بالا مثلثی است و اگر عناصر بالای قطر اصلی ۰ باشند ماتریس پایین مثلثی است.

ماتریس سه قطری:

ماتریس سه قطری یک ماتریس مربعی $n \times n$ می باشد که درایه های غیر صفر آن روی قطر اصلی و بلافاصله بالا و پایین قطر اصلی ظاهر می شوند . تعداد عناصر غیر صفر در این ماتریس $3n - 2$ می باشد

ماتریس اسپارس(ماتریس خلوت): به هر ماتریس m*n که تعداد خانه های ۰ یا بدون ارزش آن بیشتر از تعداد خانه های مخالف ۰ باشد ماتریس اسپارس می گویند.

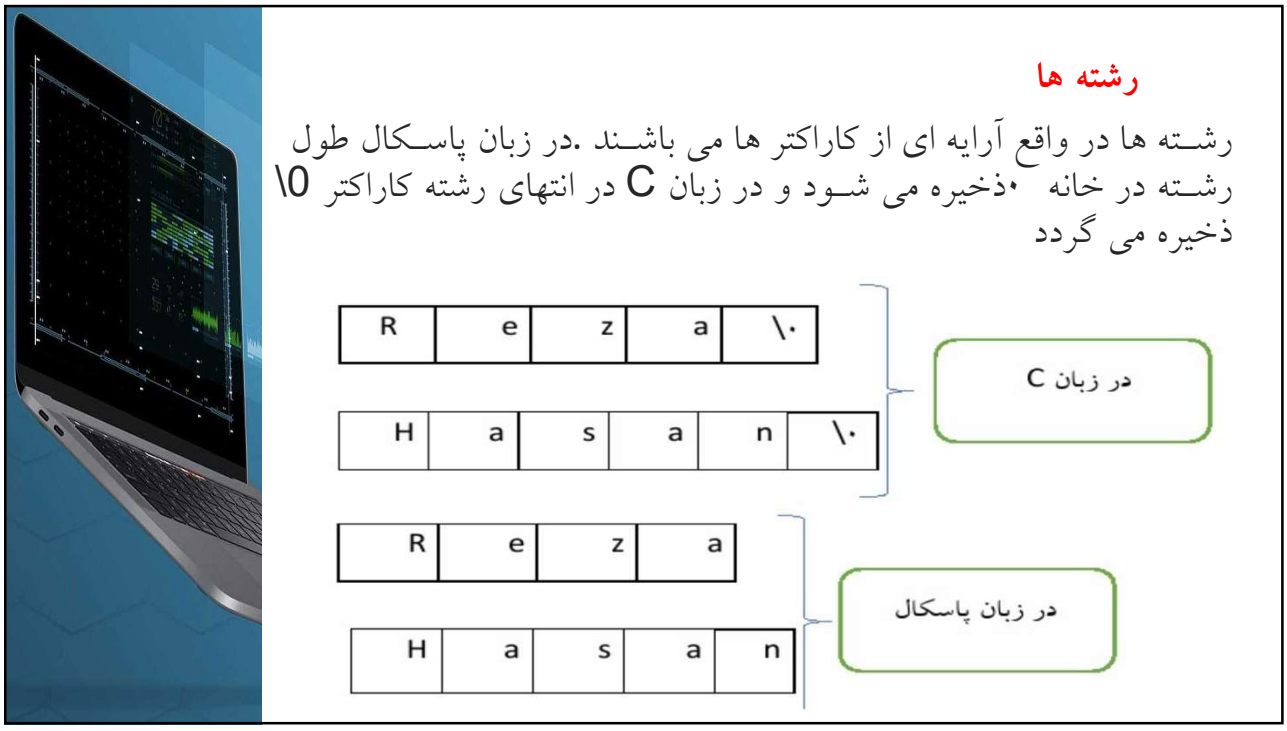
۰	۰	۰	۱	۰	۰
۰	۰	۱۵	۰	۰	۱۲
۰	۰	۰	۰	۰	۰
۰	۹	۰	۰	۰	۰
۱۸	۰	۰	۰	۲	۰

۵*۶=۳۰

۳۰*٤=۱۲۰ byte

در صورتی که ماتریس m*n اسپارس خانه مخالف ۰ وجود داشته باشد برای نمایش آن از یک آرایه دو بعدی با ۳ ستون و R+۱ سطر استفاده می شود.

۱۸*٤= ۷۲ byte


رشته ها
رشـته ها در واقع آرایه ای از کاراکتر ها می باشـند .در زبان پاسـکال طول رشـته در خانه ۰ذخیره می شـود و در زبان C در انتهای رشته کاراکتر \0 ذخیره می گردد
R e z a \۰
H a s a n \۰
در زبان C
R e z a
H a s a n
در زبان پاسکال


تمرینات فصل ۲

سوالات تستی

۱- اگر آرایه A به صورت int A[20][10][5] تعریف شده باشد (یعنی محدوده اندیس ها به صورت A[0..19][0..9][0..4] باشد). با فرض این که هر عدد int چهار بایت لازم دارد و آدرس شروع آرایه صفر باشد، آدرس عنصر A[3][4][2] را در حالت ستون محور (column major) عبارت است از:

(مهندسی IT- دولتی ۸۵)

۱) 1934 ۲) 1932 ۳) 484 ۴) 483

حل:جواب گزینه ۲ است.

$$0+[(3-0)+(4-0)\times 20+(2-0)\times 20\times 10]\times 4=1932$$

۴- الگوریتم زیر را برای جستجوی x در آرایه A شامل n عنصر درنظر بگیرید. پیچیدگی زمانی این الگوریتم چیست؟ (فرض کنیم در ابتدا down=1 , top =n , x ورودی باشد)

(مهندسی کامپیوتر- آزاد ۸۶)

```
int  search (int  down , int  top){
  int tmp;
  if (down > top)  return 0;
 else{
        tmp= (down+top) div 2;
        if (x== A[tmp])
            return tmp;
        else  if (x<A[tmp])
                    return  search (down,tmp-1);
               else
                    return  search (tmp+1,top);
        }
}
```

۱) $O(\log_{10}^{n})$ ۲) $O(n\log n)$ ۳) $O(\frac{n}{2})$ ۴) $O(\log_2 n)$

حل: جواب گزینه ۴ است. الگوریتم داده شده ، جستجوی دودویی می باشد که از مرتبه $O(\log_2 n)$ است.

- فرض کنید A یک ماتریس $m \times n$ خلوت باشد، عناصر غیر صفر آن را به ترتیب سطری در یک آرایه $t \times 3$ ذخیره می کنیم. تابع زمان عمل ترانهاده گیری کدام است؟ (t تعداد عناصر غیر صفر ماتریس است)

(علوم کامپیوتر- دولتی ۸۰)

۱) $\theta(t+m)$ ۲) $\theta(tm)$ ۳) $\theta(tn)$ ۴) $\theta(t+n)$

حل: گزینه ۴ درست است.

مرتبه زمانی الگوریتم ترانهاده گیری از این ماتریس برابر $\theta(t+n)$ می باشد.

- تعداد عناصر غیر صفر یک آرایه اسپارس سه قطری $n \times n$ چقدر است؟ (مهندسی IT – آزاد ۸۸)

۱) $3n-3$ ۲) $3n+2$ ۳) $3n-2$ ۴) $3n+3$

حل: گزینه ۳ درست است.

n عنصر در قطر اصلی و n-1 عنصر در بالای قطر اصلی و n-1 عنصر در پایین قطر اصلی.

$$n+(n-1)+(n-1)=3n-2$$

سوالات تشریحی:

آرایه دو بعدی x با 3 سطر و 4 ستون مفروض است. اگر خانه اول آرایه در محل 10 حافظه ذخیره شود. محل عنصر x[2,3] در حافظه کدام است؟(w=1)

x : array [1..3,1..4] of char ;

حل:

سطری : $10+[(2-1)(4-1+1)+(3-1)]\times 1=16$

ستونی : $10+[(2-1)+(3-1)(3-1+1)]\times 1=17$

آرایه سه بعدی A(1..11, 2..13 ,3..14) از آدرس 100 حافظه قرار گرفته می‌باشد. آدرس عنصر A(9,7,6) چه خواهد بود؟ (w=1)

حل:

سطری : $100+[(9-1)\times(13-2+1)\times(14-3+1)+(7-2)\times(14-3+1)+(6-3)]\times 1=1315$

ستونی : $100+[(9-1)+(7-2)\times(11-1+1)+(6-3)\times(13-2+1)\times(11-1+1)]\times 1=559$

برای پیدا کردن عدد 9 در آرایه مرتب زیر با روش جستجوی دودویی، به چند مقایسه نیاز است؟

1	2	3	4	5	6	7	8	9
5	**9**	12	20	**35**	50	82	88	97

حل:

ابتدا عدد 9 با عنصر وسط آرایه یعنی 35 مقایسه می شود و چون از آن کوچکتر است مقایسه به طور بازگشتی در زیر آرایه x[1..4] انجام می گیرد. بنابراین مقایسه بعدی با عنصر وسط این آرایه یعنی 9 است که برابر است. بنابراین با دو مقایسه به نتیجه می رسیم.

حداقل تعداد ضرب های لازم برای ضرب ۴ ماتریس زیر کدام است؟

$$A_{30\times1}, B_{1\times40}, C_{40\times10}, D_{10\times25}$$

حل: ابتدا دو ماتریس B و C را ضرب می کنیم، چون بعد مشترک آنها یعنی 40 از همه بیشتر است:

$$A_{30\times1}\ (BC)_{1\times10}\ D_{10\times25}$$

سپس ماتریس بدست آمده از مرحله قبل را در ماتریس D ضرب می کنیم:

$$A_{30\times1}(BCD)_{1\times25}$$

و در نهایت ماتریس A را در ماتریس بدست امده ضرب می کنیم:

$$(ABCD)_{30\times25}$$

تعداد ضرب ها برابر است با:

$$1\times40\times10+1\times10\times25+30\times1\times25=1400$$

ترانهاده ماتریس اسپارس s که به کمک ماتریس a نمایش داده شده را بدست آورید.

$$S=\begin{pmatrix}0 & 0 & 2 & 0\\ 0 & -6 & 0 & 0\\ 0 & 0 & 0 & 0\end{pmatrix}\Rightarrow a=\begin{pmatrix}3 & 4 & 2\\ 1 & 3 & 2\\ 2 & 2 & -6\end{pmatrix}$$

حل:

در ماتریس a، محل ستون اول و دوم را با یکدیگر جابجا کرده:

4	3	2
3	1	2
2	2	-6

و سپس سطر 2 و سطر 3 را بطور مرتب می نویسیم:

4	3	2
2	2	-6
3	1	2

ماتریس سه قطری 4×4 زیر را به صورت سطری در یک آرایه یک بعدی ذخیره نمایید.

$$\begin{bmatrix}6 & 23 & 0 & 0\\ 1 & 3 & 7 & 0\\ 0 & 2 & 4 & 83\\ 0 & 0 & 9 & 5\end{bmatrix}$$

حل:

فقط عناصر غیر صفر ماتریس را سطر به سطر در آرایه یک بعدی ذخیره می کنیم:

	1	2	3	4	5	6	7	8	9	10
B	6	23	1	3	7	2	4	83	9	5

پرسش های فصل دوم

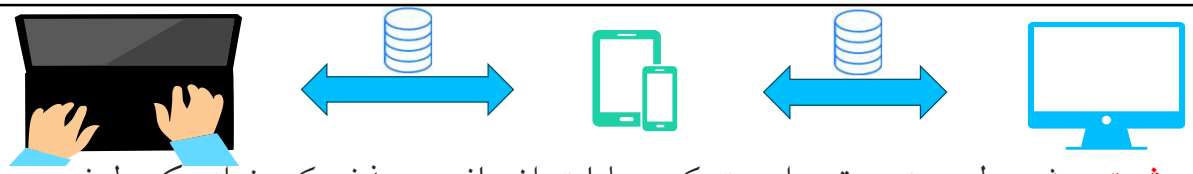
-1

آرایه چهار بعدی A[1..10,1..20,1..10,1..20] را در نظر بگیرید که به صورت سطری ذخیره شده است. اگر آدرس شروع آرایه صفر باشد،آدرس عنصر A[1,2,3,4] کدام است؟ (w=1)

۲-

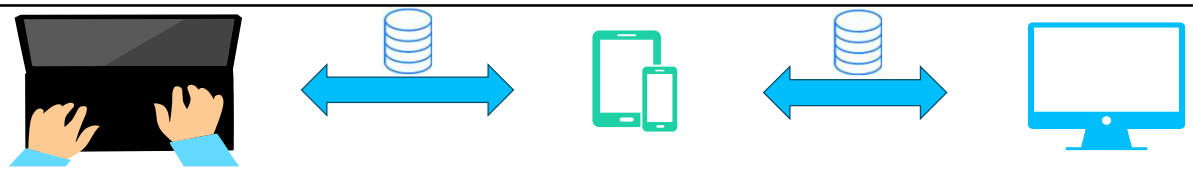
ضرب سه ماتریس را به چه ترتیبی انجام دهیم تا سریع تر اجرا شوند؟

A ۵*۱۵ B ۱۵*۴۰ C ۴۰*۸

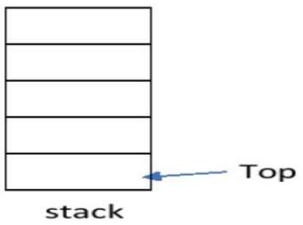


پشته: پشته لیست مرتبی است که عملیات اضافه و حذف کردن از یک طرف آن انجام می شود .پشته به صورت (Lifo)عمل می کند یعنی آخرین عنصر وارد شده اولین عنصری است که خارج می شود .به پشته (Lifo) نیز گفته می شود.

- یکی از کاربرد های پشته ذخیره آدرس بازگشت و ساخت متغیر های محلی در صدا زدن توابع است.
- به عمل ریختن اطلاعات در داخل پشته(پوش)
- به برداشتن اطلاعات (پاپ) گفته می شود.
- ساده ترین راه نمایش پشته استفاده از آرایه ۱بعدی به طول nمی باشد.

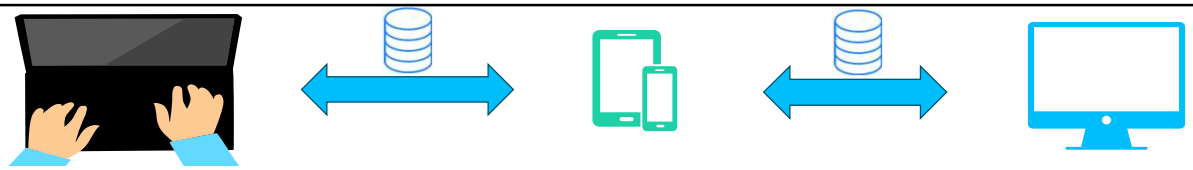


خانه های آرایه stack از عدد یک تا n شماره گذاری شده و در کنار آرایه متغیری به نام Topوجود دارد که عنصر بالایی به آن اشاره می کند.
Top از ۰تا n تغییر می کند و در ابتدای کارTop= 0است.



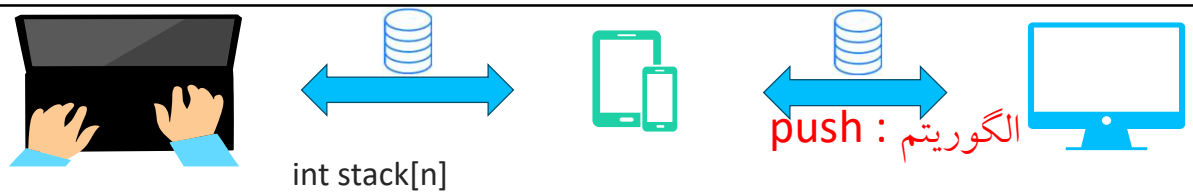
if top == ۰ stack خالی

if top == n stack پر



الگوریتمpop:

```
int stack[n]
 int pop()
{
int x;
if(top==۰) {
cout<<"پشته خالی است";
return -۱;  }
else
{
x=stack[top]
 top--;
 return x;
```



الگوریتم : push

```
int stack[n]
void push(int x)
{
if(top==n)
{
cout<<"پشته پر است";
return -۱
}
top++;
stack[top]=x;
}
```

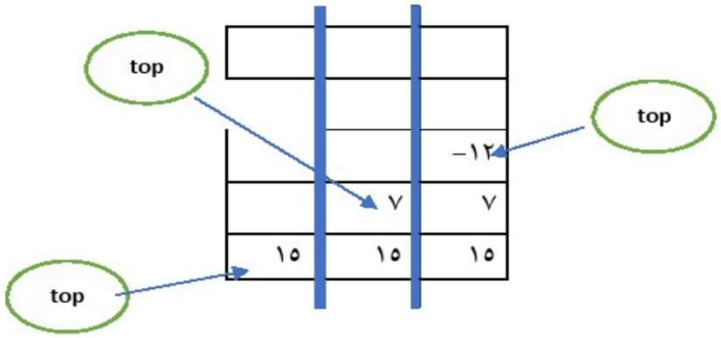
نکته stack از یک شروع می شود و تا nادامه می یابد.

نکته :تعداد خروجی های مجاز از یک پشته n تایی برابر است با $\frac{\binom{2n}{n}}{(n+1)}$

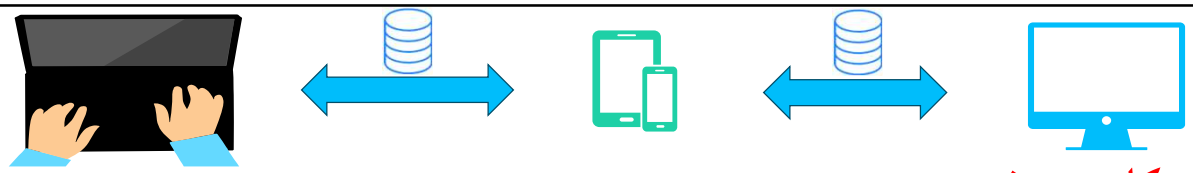
عملیات زیر را به ترتیب از سمت چپ به راست روی شکل نشان دهید(n = ۵)

push(۱۵)
push(۷)
push(-۱۲)

A=pop()
B=pop()



A=-۱۲
B=۷



کاربرد پشته:

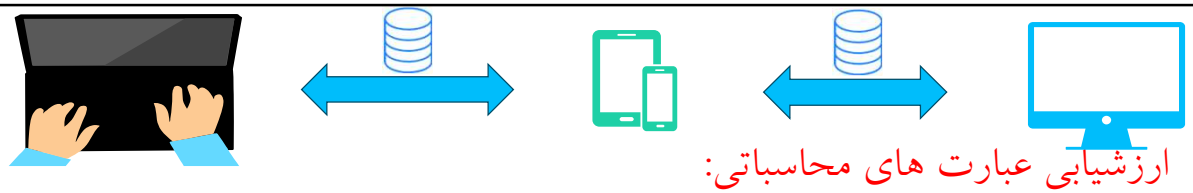
۱. استفاده از توابع بازگشتی

۲. ارزیابی عبارت های محاسباتی یعنی infix , postfix , prefix

۳. الگوریتم مسیر حرکت maze

٤. پیمایش عمیقی گراف ها و درخت ها

٥. استفاده در روش های مرتب سازی marge sort و Quick sort



ارزشیابی عبارت های محاسباتی:

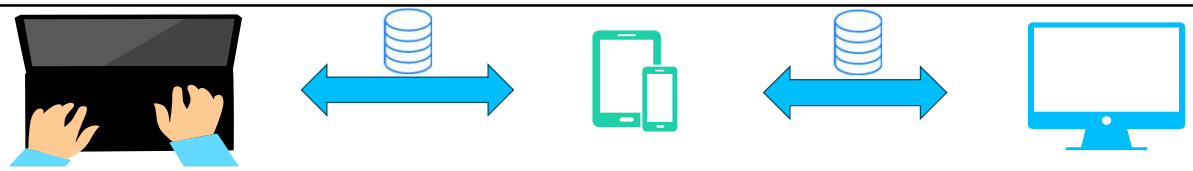
هر عبارت محاسباتی با توجه به اولویت عملگرهایش قابل ارزیابی و محاسبه است .

عملوند ← a*b → عملوند

Oprand

--a ← یه یک عملوند نیاز دارند

عملگر:
- ۱)یکانی (unary)
- ۲)باینری (binary)



۱. () پرانتز

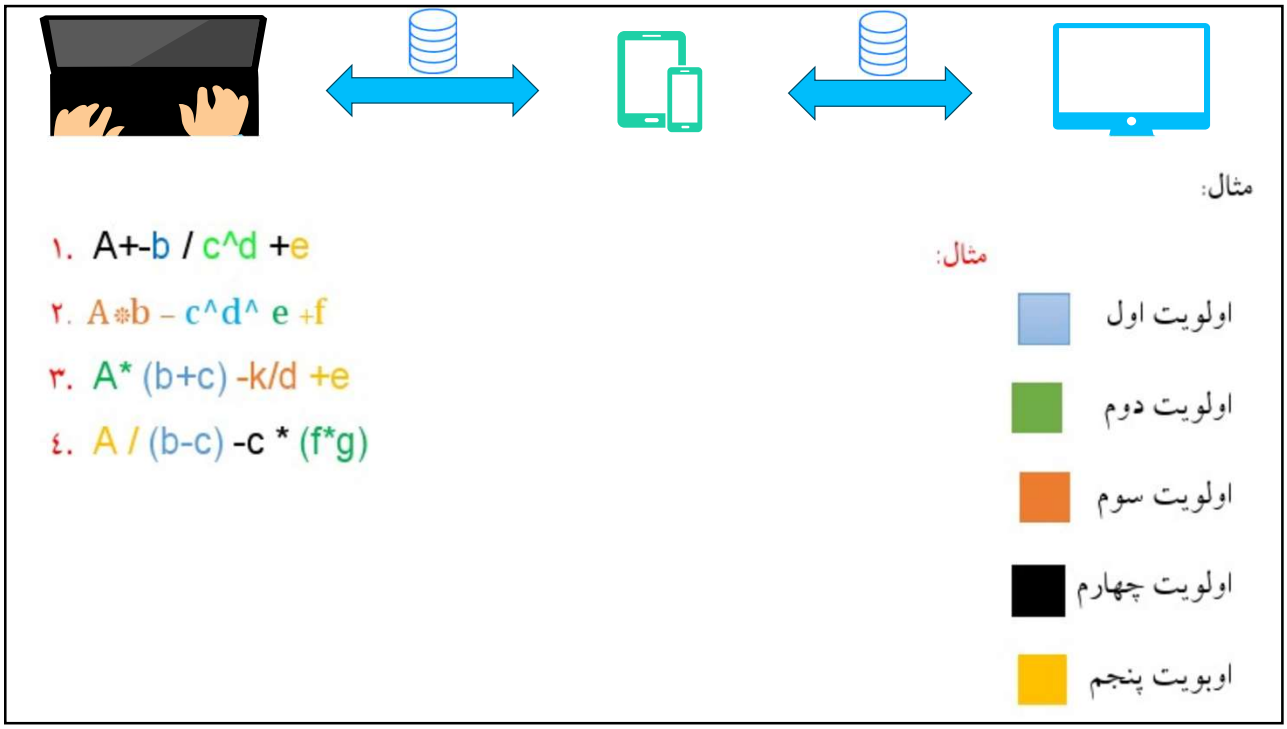
۲. - منفی یا + مثبت

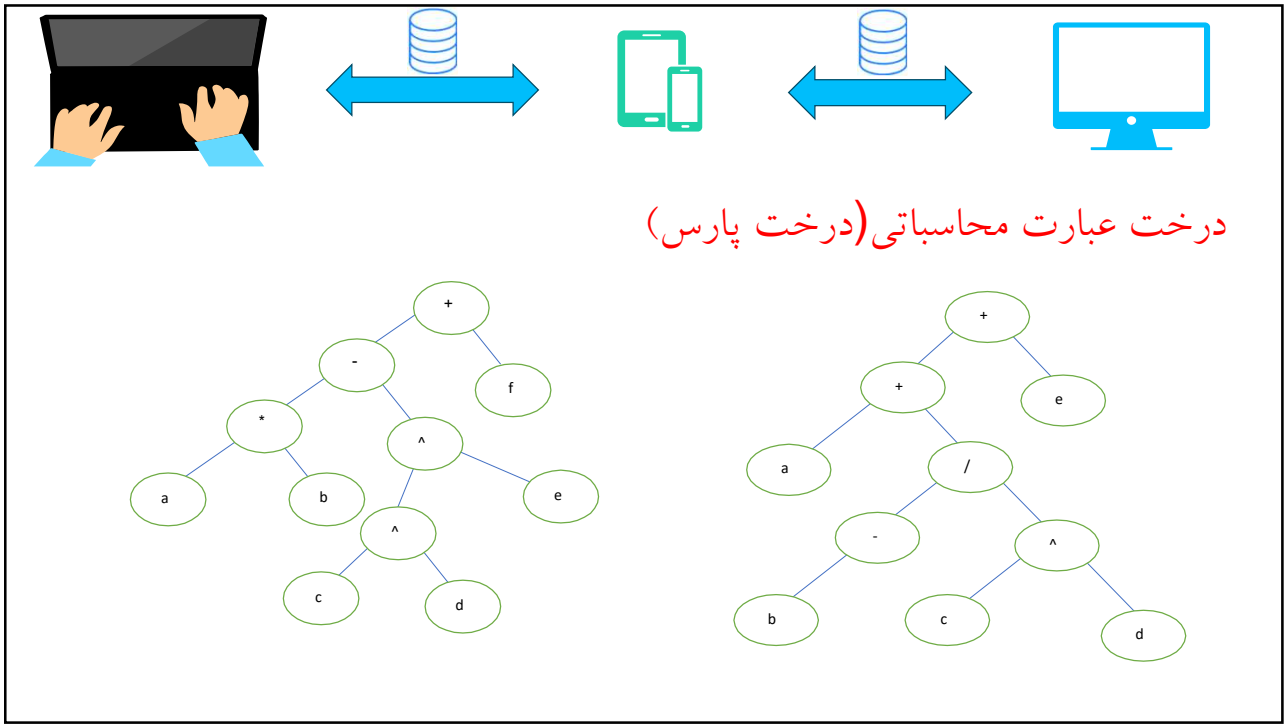
۳. توان

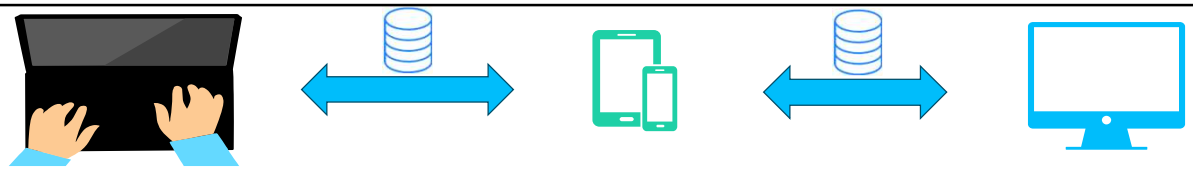
٤. * ضرب / تقسیم

٥. + جمع - منها

اولویت عملگر:
- اولویت از چپ به راست
- اولویت از چپ به راست
- اولویت از چپ به راست
- اولویت از چپ به راست


مثال:
۱. A+-b / c^d +e
۲. A*b – c^d^ e +f
۳. A* (b+c) -k/d +e
٤. A / (b-c) -c * (f*g)
مثال:
اولویت اول
اولویت دوم
اولویت سوم
اولویت چهارم
اوبویت پنجم


درخت عبارت محاسباتی(درخت پارس)
+
-
f
*
^
a
b
e
^
c
d
+
+
e
a
/
-
^
b
c
d



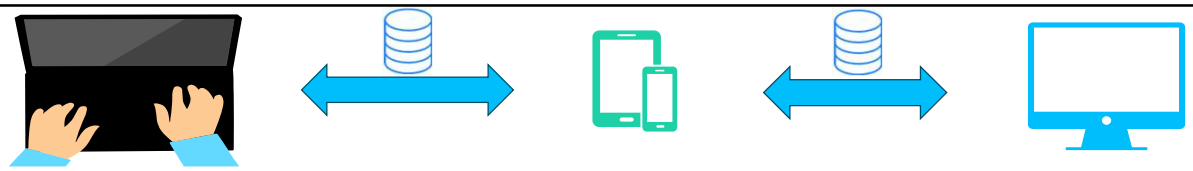
برای تشکیل درخت هر عبارت محاسباتی به صورت زیر عمل می کنیم.

۱. ابتدا اولویت های هر عبارت محاسباتی را بر حسب اولویت های عملگرهایش محاسبه می کنیم.

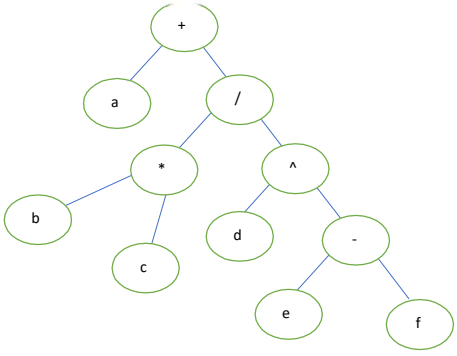
۲. ریشه درخت کم اولویت ترین از نظر اولویت است.

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

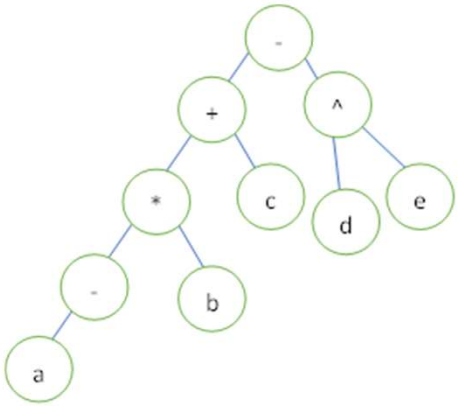
٤. برگ های درخت عملوند ها هستند

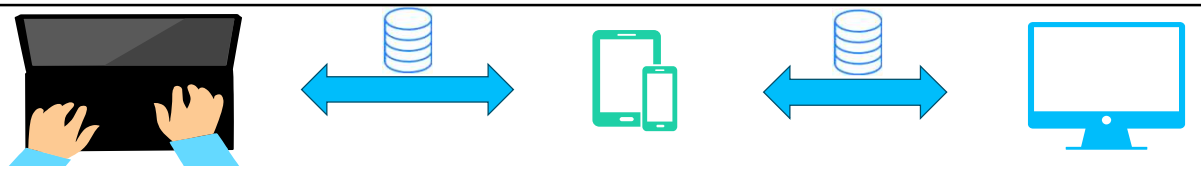


a+b*c/d^(e-f)



-a*b+c-d^e





پرانتز بندی:

(a+((b*c)/(d^(e-f))))

((((-a)*b)+c)-(d^e))

۱. pre fix عبارت پیشوندی روش لهستانی +ab -+abc ,

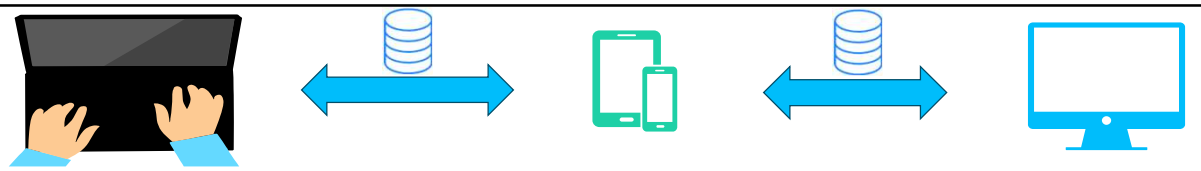
۲. in fix عبارت میانوندی a+b (a+b)-c

۳. post fix عبارت پسوندی ab+ ab+c-

a+ b*c / d ^ (e-f)

prefix ⇒ +a/*bc^d-ef

postfix ⇒ a+ bc * / d ^ ef -

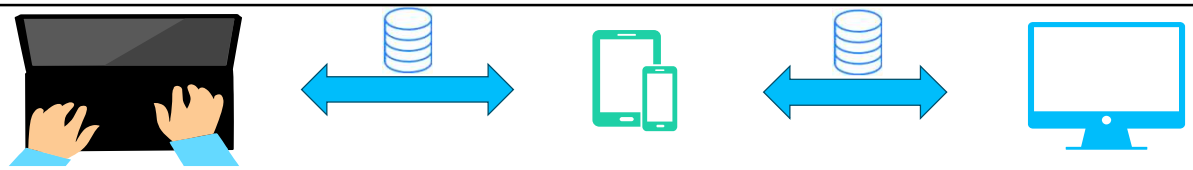


عبارت : in fix: در این روش عملگر وسط قرار گرفته و عملوند ها چپ و راست قرار می گیرند. اولویت عملگر ها مطرح می باشد و با توجه به جدول اولویت عملگر ها مشخص می گردد.

عبارت : pre fix : در این روش هر عملگر قبل از عملوند هایش قرار می گیرد اولویت عملگرها مطرح نیست و از پرانتز استفاده نمی شود.

عبارت : post fix : در این روش هر عملگر بعد از عملوند هایش قرار می گیرد.اولویت مطرح نیست و از پرانتز هم استفاده نمی شود.

نکته :در تبدیل عبارت in fix به pre fix و post fix ترتیب عملوند ها به هیچ شکل عوض نمی شود اما ترتیب عملگر ممکن است عوض شود



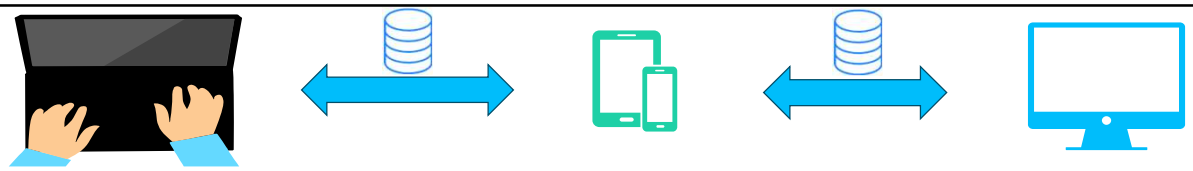
تبدیل عبارت in fix به عبارت های post fix و pre fix :

برای این عمل سه روش وجود دارد:

۱. روش پرانتز گذاری

۲. روش استفاده از پشته

۳. روش پیمایش درخت عبارت محاسباتی



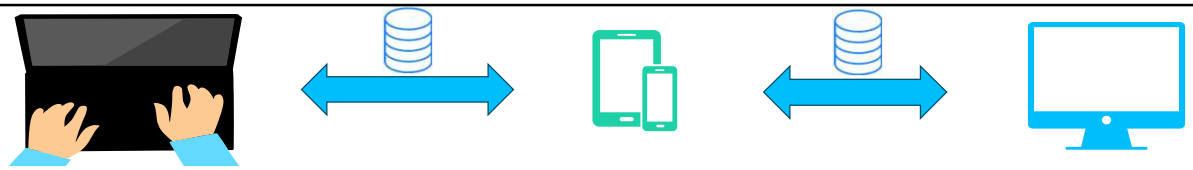
تبدیل عبارت های in fix به post fix به روش پرانتز گذاری

الف)عبارت را به طور کامل بر حسب اولویت عملگرها پرانتز گذاری می کنیم.در حین پرانتز گذاری عملگر مربوط به هر پرانتز را روی پرانتز بسته می گذاریم.

ب)عبارت را از چپ به راست شامل عملوند ها و عملگر های روی پرانتز بسته در خروجی می نویسیم.

((a * (b+c))-(g/d))

abc + * gd / -



تبدیل عبارت های in fix به pre fix با روش پرانتز گذاری

الف)عبارت را به طور کامل بر حسب اولویت عملگر ها پرانتز گذاری می کنیم در حین پرانتز گذاری عملگر مربوط به هر پرانتز را روی پرانتز باز می گذاریم.

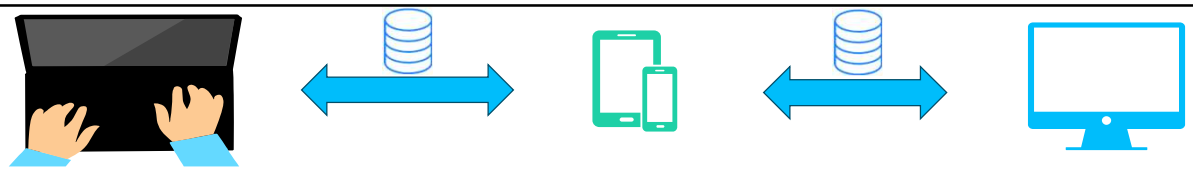
ب)عبارت را از چپ به راست شامل عملوند ها و عملگر های روی پرانتز باز در خروجی می نویسیم

مثال:

۱- ((a*b)+c)
+*abc

2- ((a*(b+c))-(g/d))
-*a+ bc / gd

۳- ((a+((b^c)*d)-e)
-+a*^ bc de



تبدیل in fix به post fix با استفاده از پشته

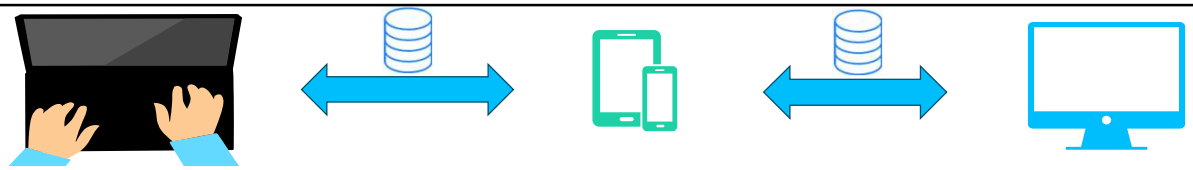
ابتدا از سمت چپ شروع به خواندن تک تک اجزای عبارت محاسباتی می کنیم که به صورت in fix است و به سمت راست حرکت می کنیم.

یک استک را برای نگه داری عملگر ها در نظر می گیریم.اگر جزئی از عبارت را که خوانده ایم یک عدد یا متغیر است(عملوند)آن را به صورت مستقیم در خروجی چاپ می کنیم.اگر عملگر یا پرانتز بود کار های زیر را انجام می دهیم.

الف)اگر پرانتز باز"("بود آن را در استک push می کنیم.

ب)اگر پرانتز بسته"("بود تا زمانی که در استک به پرانتز باز برسیم pop می کنیم و در سر راه هرچه عملگر دیدیم آن را در خروجی چاپ می کنیم.

ج)اگر یک عملگر بود و در سر استک پرانتز باز "("بود عملگر را در استک push می کنیم .

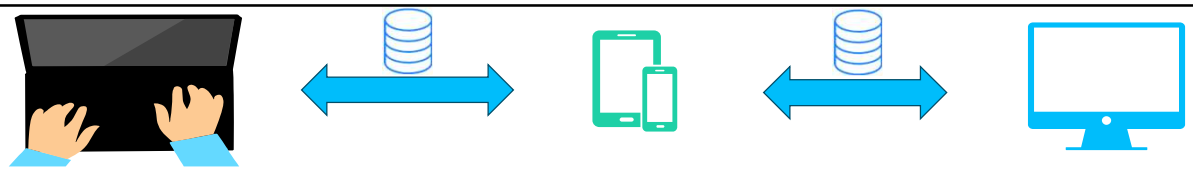


د)اگر عملگر بود و در سر استک یک عملگر دیگر موجود بود دو حالت وجود دارد:

۱)اگر عملگری که دیدیم از عملگری که در سر استک است اولوبت کمتری داشت عملگر سر استک را از استک pop می کنیم و آن را در خروجی چاپ می کنیم و دوباره چک می کنیم که آیا عملگری که دیده ایم با چیزی که سر استک است اولوبتش کمتر است یا نه.اگر کمتر بود دوباره همین کار را انجام می دهیم تا زمانی که اولویتش کمتر از عملگر سر استک نباشد و بعد عملگری را که دیده ایم در استک push می کنیم.

۲)حالت دوم زمانی اتفاق می افتد که عملگری را که دیده ایم اولوبت بیشتری از عملگر سر استک برخوردار باشد در این صورت عملگر جدیدی را که دیده ایم در استک push می کنیم.

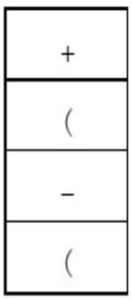
اگر تک تک اجزای عبارت را از سمت چپ به راست دیدیم ولی استک خالی نشده بود تمام محتویات می کنیم و عملگر ها را در خروجی نمایش می دهیم pop استک را تا زمانی که خالی شود

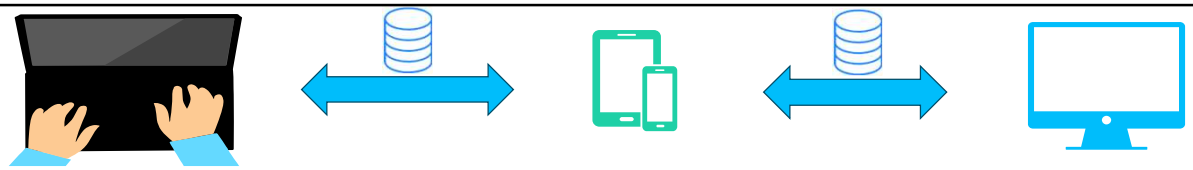


مثال:

(a-(b+c)) / (e+f))

abc+-ef+/

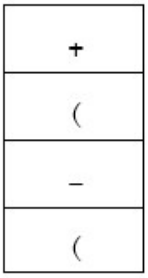




مثال:

(a+((b*(c/(d+e)))+f))

abcde+/*f++



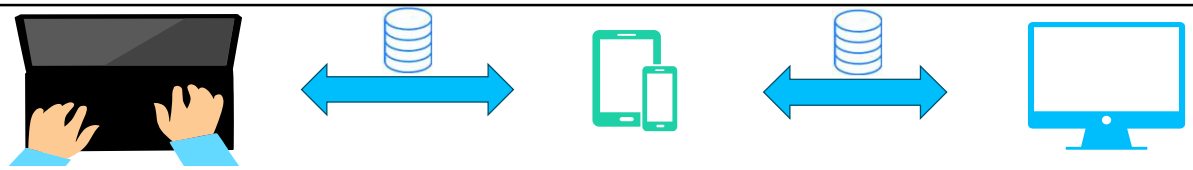
تبدیل عبارت های in fix به pre fix با استفاده از پشته

الف)عبارت را از سمت راست پیمایش می کنیم.

ب)عملوند ها را در خروجی می نویسیم

ج)به هر پارانتز بسته که رسیدیم آن را به راحتی در استک قرار می دهیم

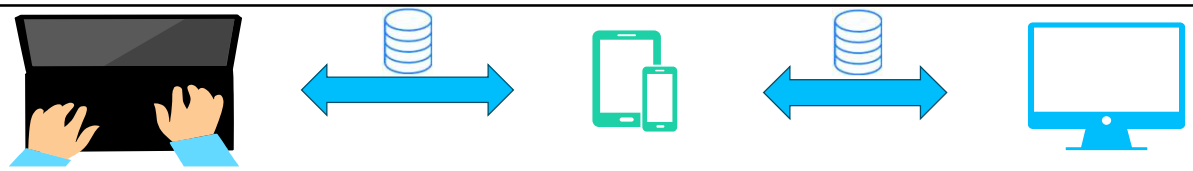
د)به هر عملگر که رسیدیم به شرطی که اولویت آن عملگر از عملگر بالای استک بیشتر یا مساوی باشد آن را داخل استک قرار می دهیم در غیر این صورت آن قدر از بالای استک عملگر خارج کرده و در خروجی می‌نویسیم تا یا استک خالی شده و یا به عملگر برسیم که بتوانیم عملگر مورد نظر را روی آن در استک قرار دهیم



د)اگر بالای استک پرانتز بسته باشد هر عملگر به راحتی روی آن قرار می گیرد.

ه)به هر پرانتز باز که رسیدیم آنقدر از استک عملگر خارج می کنیم و در خروجی می نویسیم تا به پرانتز بسته برسیم در این وضعیت پرانتز باز و بسته باهم خنثی می شوند.

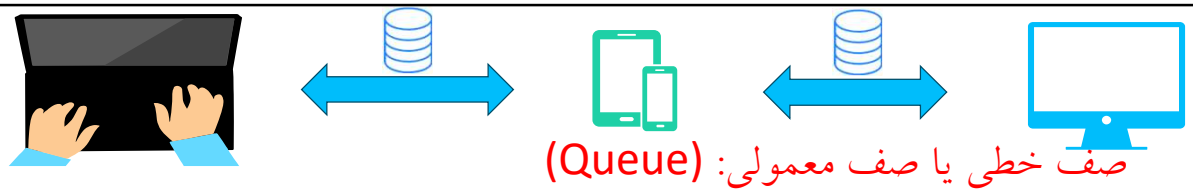
ی)زمانی که به ابتدای عبارت رسیدیم در صورتی که استک خالی نباشد آن را به طور خالی در خروجی می نویسیم



(a+((b+(c/(d+e)))*f))

(
(
*
)
)
*
)
)

+a*+b/c+def

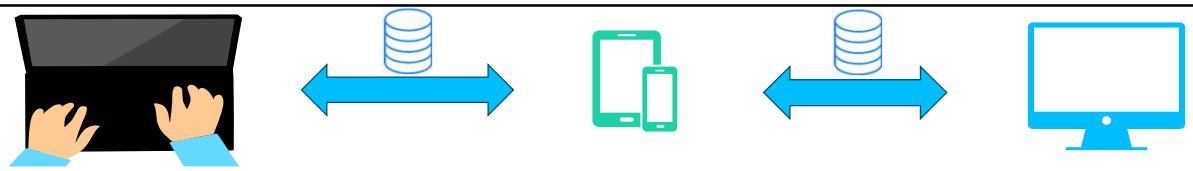


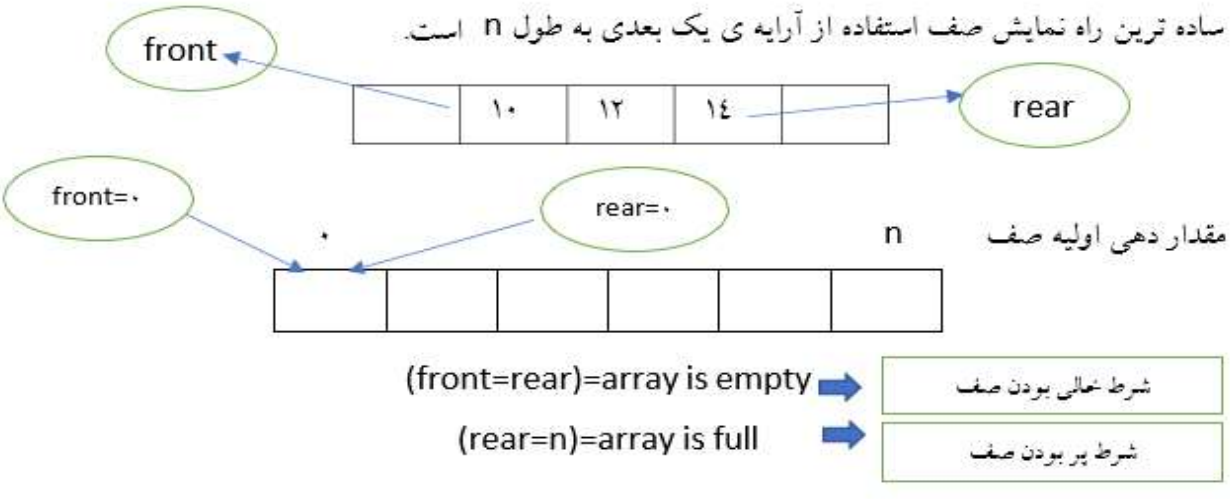
صف خطی یا صف معمولی: (Queue)

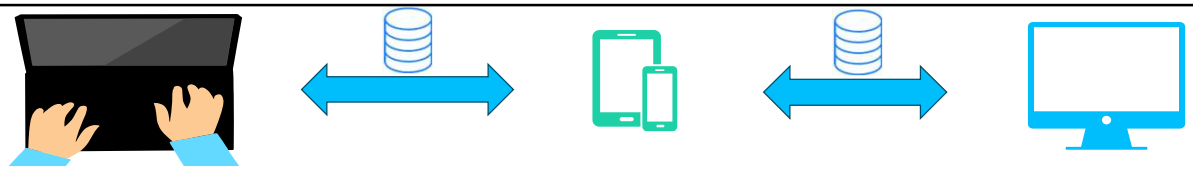
صـف لیسـت خطی مرتبی از عناصـر اسـت که در آن عمل درج از یک طرف و عمل حذف از طرف دیگر انجام می شود.به ساختار صف fifo نیز گفته می شود.

در صـف از دو متغیر اشـاره گر به نام front که به عنصـر قبل از عنصـر ابتدایی اشـاره می کند و دیگری به نام rear که همیشه به اخرین عنصر اشاره می کند

کاربرد مهم صـف در زمان بندی برنامه ها در سـیسـتم عامل اسـت.نمایش صـف با دو سـاختار لیسـت پیوندی و آرایه ای می باشد

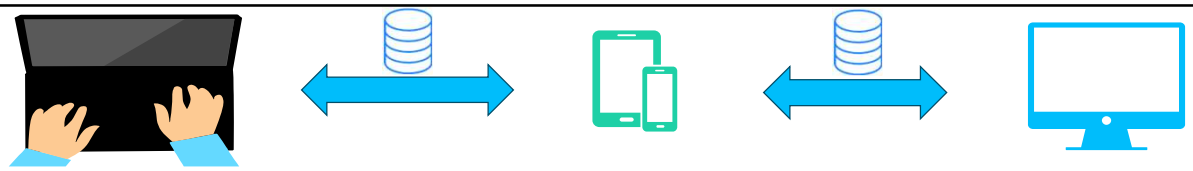




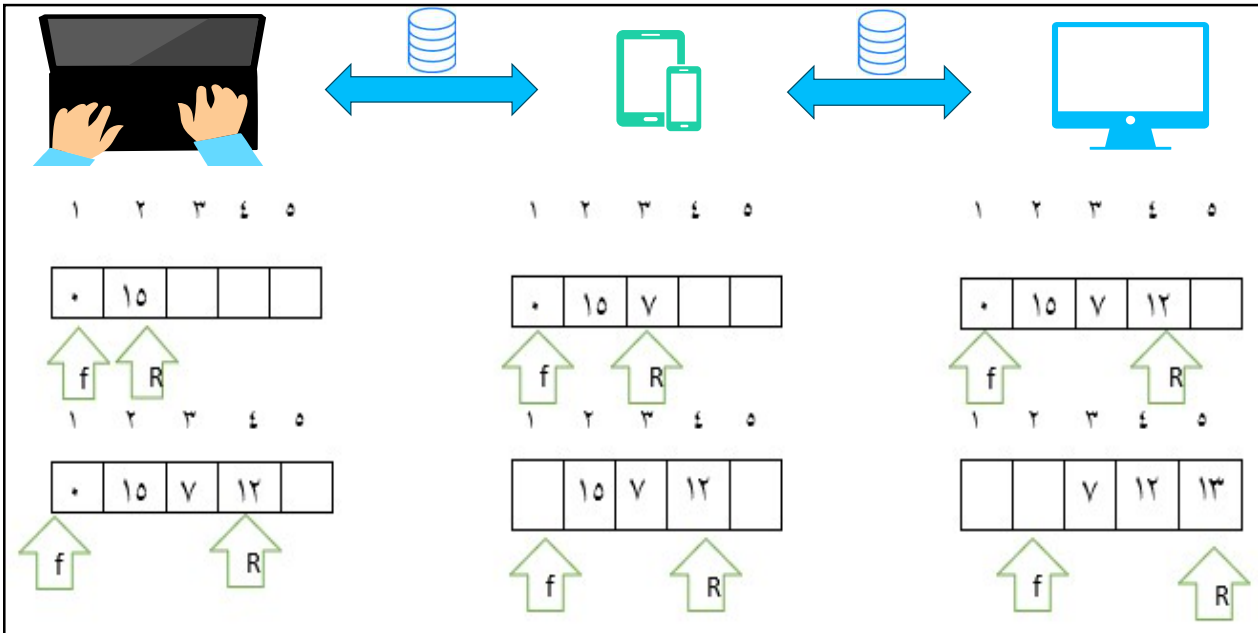


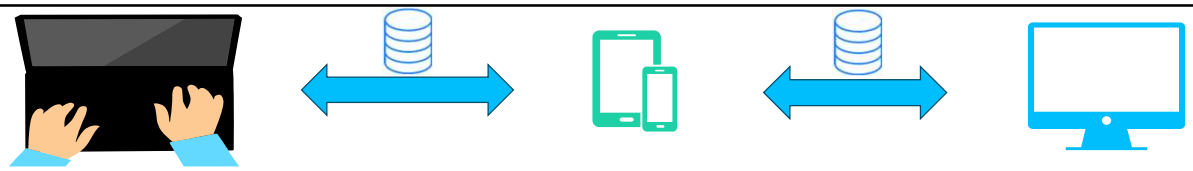
برای نوشتن زیر برنامه های اضافه و حذف از صف به صورت زیر عمل می کنیم:

```
void add Queue(double)
{
if(rear==n)
{
cout<<"Queue full";
}
else
{
rear++
Queue[rear]=x
}
```



```
double del Queue()
{
if(front==rear)
{
cout<<"Queue empty";
return ٠;
}
else
{
front++;
}
return Queue[front];
}
```

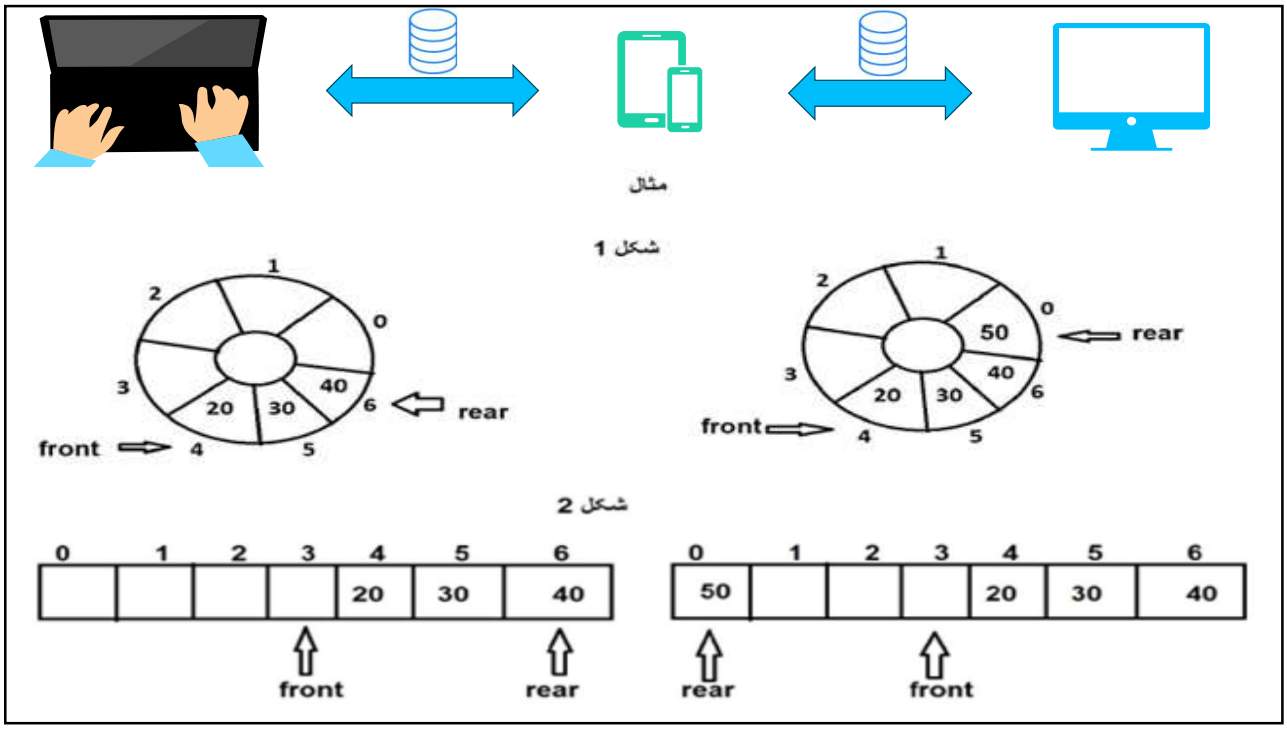


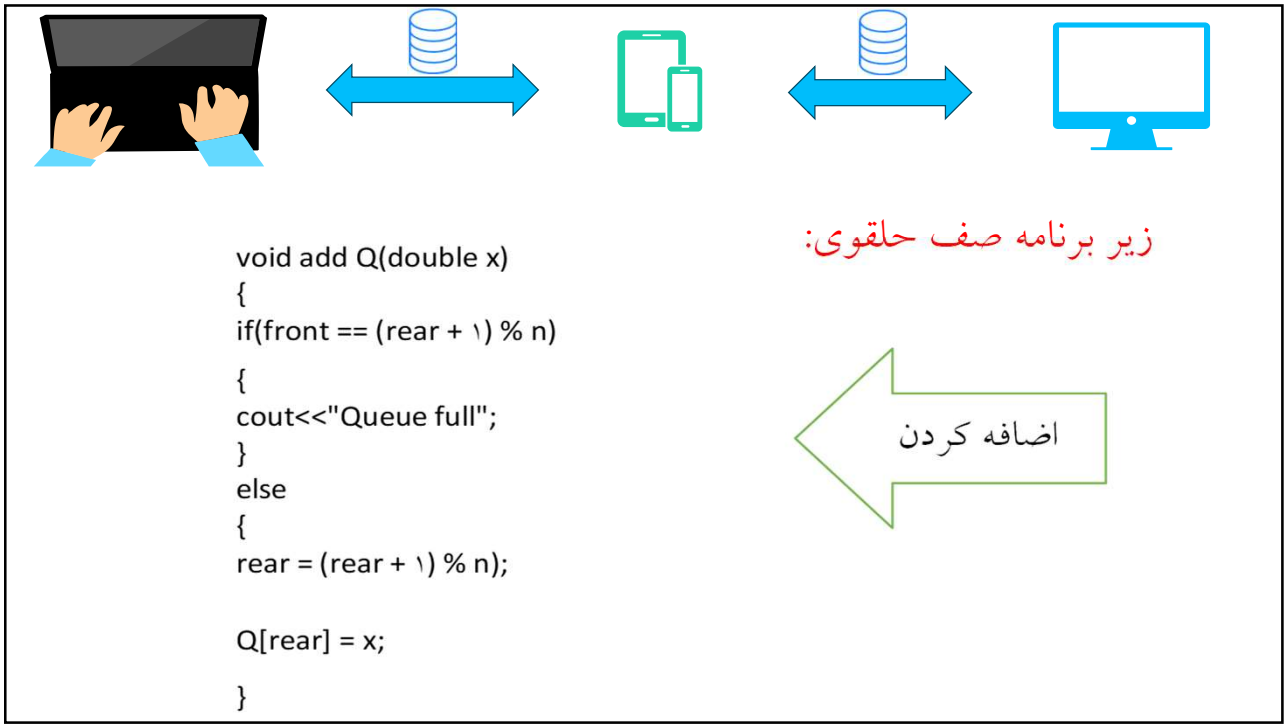


نکته:مشکل اصلی صف معمولی آن است که یک بار قابل استفاده است و هنگامی که rearبه انتها می رسد نمی توان در صف چیزی را ذخیره کرد برای حل این مشکل از صف حلقوی استفاده می کنیم.

صف حلقوی:

آرایه انتخابی ۰تا n-1 Q[0 , ... , n-1] را می توان به صورت صف حلقوی در نظر گرفت به طوری که در این صف زمانی که rear برابر n-1 می شود عنصر بعدی در خانه شماره ۰ قرار می گیرد

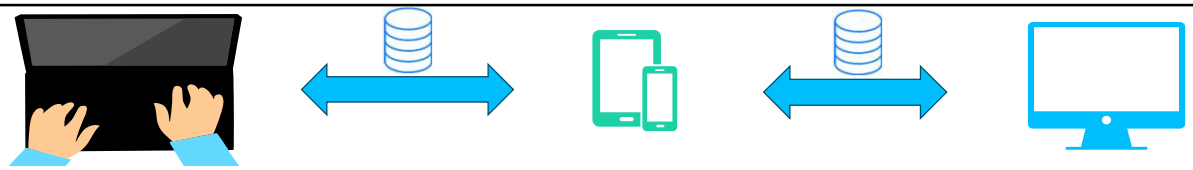




زیر برنامه صف حلقوی:

```
void add Q(double x)
{
if(front == (rear + ۱) % n)
{
cout<<"Queue full";
}
else
{
rear = (rear + ۱) % n);

Q[rear] = x;

}
```

اضافه کردن

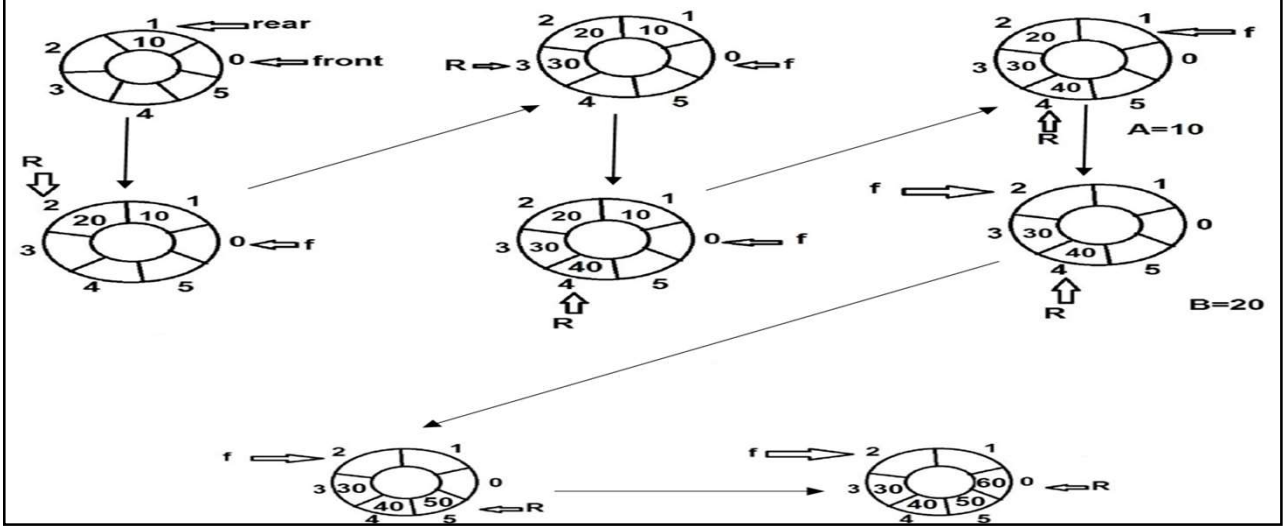


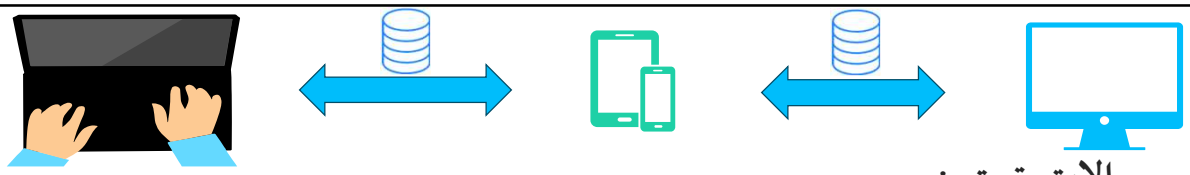
```
double del Q()
{
if(front == rear)
{
cout<<"Queue empty";
return ٠;
}
else
{
front = (front + ١) % n;

return Q[front];
```

حذف کردن

مثال: عملیات زیر را از چپ به راست روی یک صف حلقوی با رسم شکل نشان دهید n=5 .

add Q(١٠) , add Q(20) , add Q(٣٠) , add Q(٤٠) , A=del Q() , B=del Q() , add Q(٥٠) , add Q(٦٠)





سوالات تستی:

۱- سه پشته S_1 و S_2 و S_3 هر یک حاوی دو عدد بشکل زیر می باشند.

S1	S2	S3
2	4	6
1	3	5

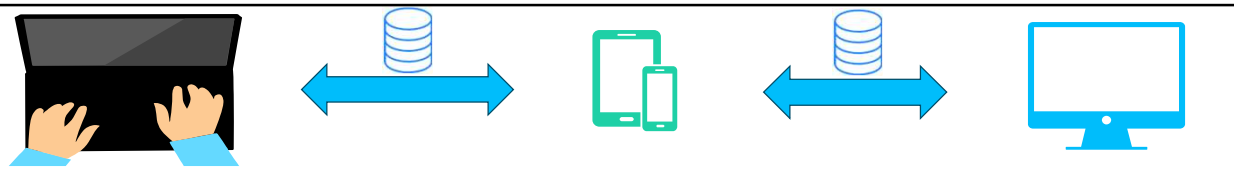
دو عملگر PopPush(i, j) و Pop(i) بصورت زیر تعریف شده اند.

PopPush(i, j) : یک قلم از پشته S_i حذف و به پشته S_j اضافه می کند.

Pop(i) : یک قلم از پشته S_i حذف و سپس آن را چاپ می کند.

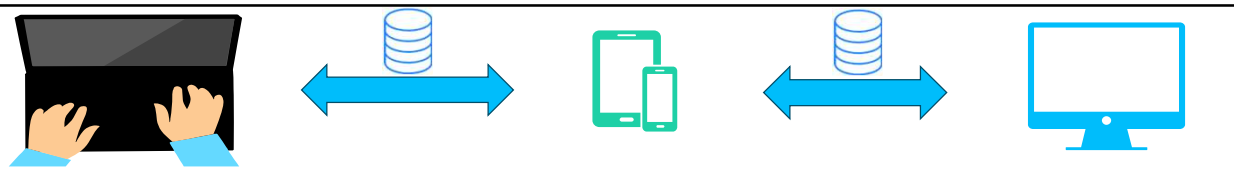
برای چاپ اعداد 1 تا 6 بصورت 6, 4, 2, 5, 3, 1 ، عملگر PopPush بایستی حداقل چند بار مورد استفاده قرار گیرد؟

۱) 3 ۲) 5 ۳) 6 ۴) 4



حل: گزینه ۴ درست است. عملیات لازم عبارتند از:

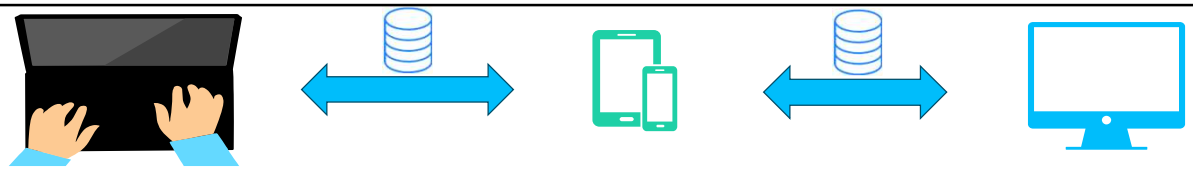
1- PopPush(1,3)
2- Pop(1)
3- PopPush(2,1)
4- Pop(2)
5- PopPush(3,1)
6- PopPush(3,2)
7- Pop(3)
8- Pop(1)
9- Pop(1)
10- Pop(2)



۲- تعداد دنباله های مجاز خروجی با k ورودی از یک پشته (stack) برابر با کدام گزینه است؟

۱) $\frac{1}{k+1}\binom{2k}{k}$ ۲) $\frac{1}{k}\binom{2k}{k}$ ۳) $2k+1$ ۴) $\frac{1}{k-1}\binom{2k}{k}$

حل: جواب گزینه ۱ است.



۳- بر روی پشته s اعمال زیر قابل انجام است. اگر n عمل از اعمال فوق به ترتیب دلخواه، بر روی پشته s که در ابتدای تهی است انجام شود، مجموع هزینه این n عمل در بدترین حالت کدام است؟

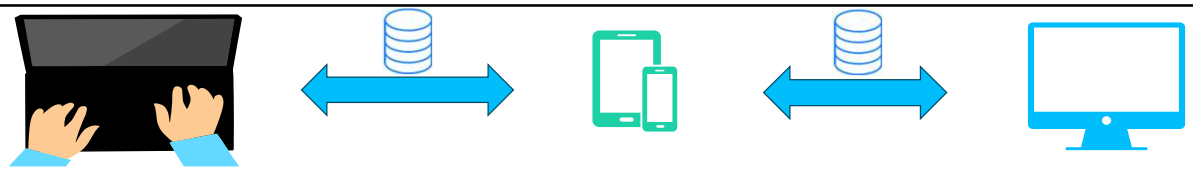
Push(s,x) : درج x در بالای پشته با هزینه $o(1)$.

Pop(s,x) : حذف عنصر بالای پشته با هزینه $o(1)$.

Multipop(s,k) : حذف k عنصر بالای پشته با هزینه $o(k)$.

۱) $o(n.\log n)$ ۲) $o(n^2)$ ۳) $o(nk)$ ۴) $o(n)$

حل: جواب گزینه ۴ است. چون هر عنصر به طور دقیق یکبار درج و مستقل از نحوه حذف شدنش توسط pop یا multipop ، حداکثر یکبار حذف می شود، به ازای هر عنصر، ۲ واحد هزینه پرداخت می شود. در نتیجه، در مجموع هزینه کل اعمال حداکثر برابر 2n خواهد بود.

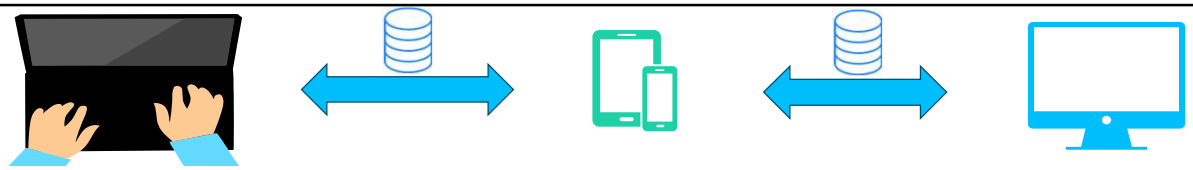


مقدار ارزیابی عبارت پسوندی 6 2 3 + – 3 8 2/+* 2 $ 3 + چیست؟($: توان)

۱) 50 ۲) 52 ۳) 48 ۴) 54

حل: جواب گزینه ۲ است.

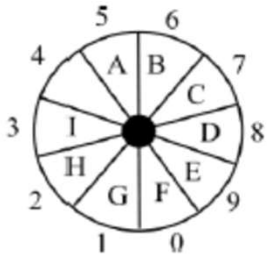
6 2 3 + - 3 8 2 / + * 2 $ 3 +
⇒ 6 5 - 3 4 + * 2 $ 3 +
⇒ 1 7 * 2 $ 3 +
⇒ 7 2 $ 3 +
⇒ 49 3 +
⇒ 52

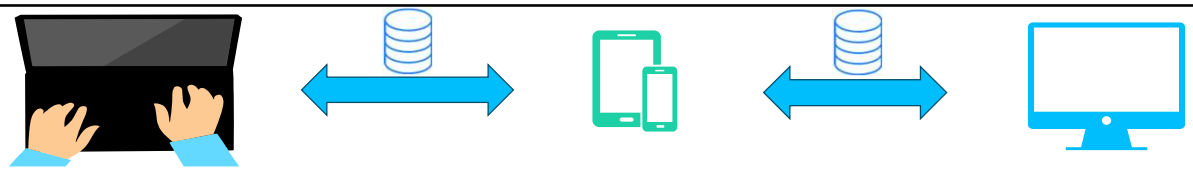


سوالات تشریحی:

یک صف حلقوی با ۱۰ خانه و front=4 , rear=3 ، چه وضعیتی دارد؟

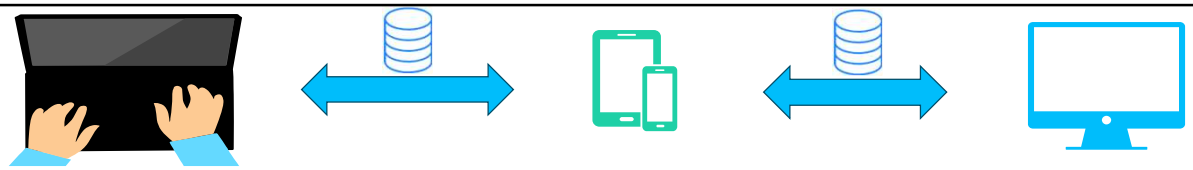
حل: صف پر است، چون مقدار (rear+1) % n برابر front است. شکل زیر مثالی از این حالت است:





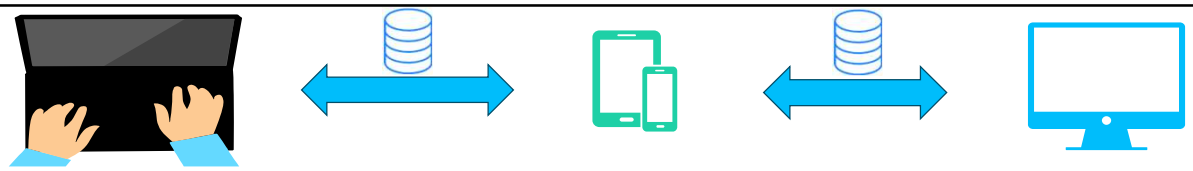
در صورتی که اعداد ۱و۲و۳ به ترتیب از راست به چپ وارد یک پشته شوند، چند حالت خروجی می تواند ایجاد شود؟

حل: با قرار دادن n=3 در رابطه $\frac{\binom{2n}{n}}{(n+1)}$ جواب برابر 5 خواهد شد.



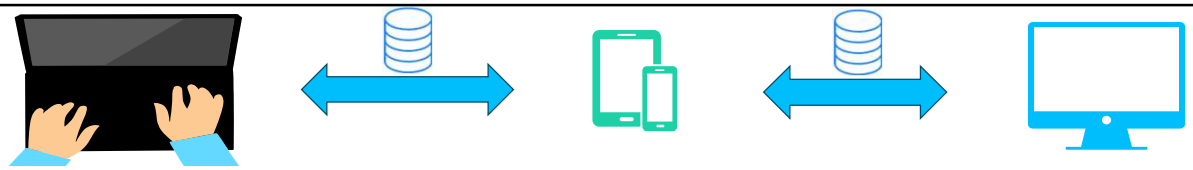
تبدیل عبارت پسوندی ABC*+ به فرم میانوندی :

$$AB^*C+ \Rightarrow A+B^*C$$



عبارت x+y*z-w را به فرم پیشوندی تبدیل نمایید.

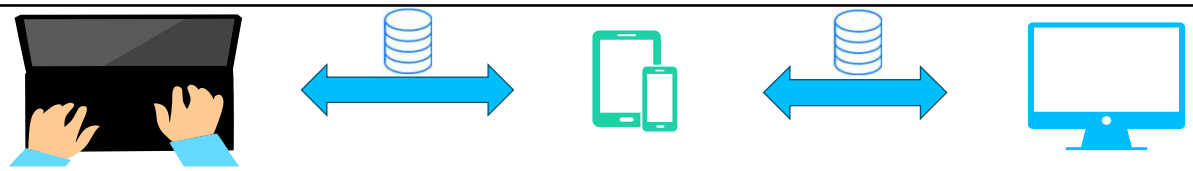
$$x+\ ^*yz\ -w \Rightarrow\ +x^*yz\ -\ w \Rightarrow\ -+x^*yzw$$



عبارت ((A+B)*D)^(E-F) را به فرم پسوندی تبدیل کنید.

پاسخ:

(AB+ *D) ^ (E-F)
⇒ AB+D* ^ (E-F)
⇒ AB+D*^ EF-
⇒ AB+D*EF-^

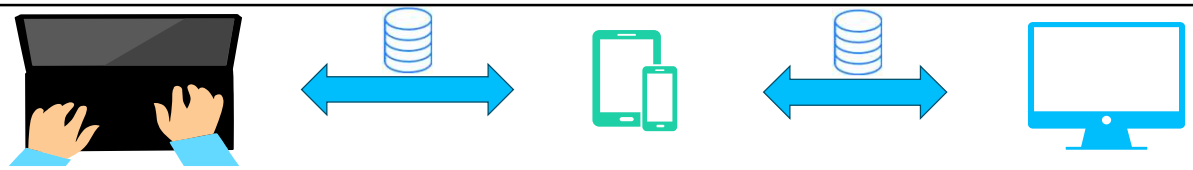


عبارت a+(b-c*d)^e-f^g^(h/ i*k) را به فرم پیشوندی و پسوندی تبدیل نمایید.

عملگر توان سمت راست،زودتر اجرا می شود.

پاسخ:

-+a^-b*cde^f^g*/hik ، abcd*-e^+fghi/k*^^-



عبارت prefix زیر را به infix تبدیل کنید؟

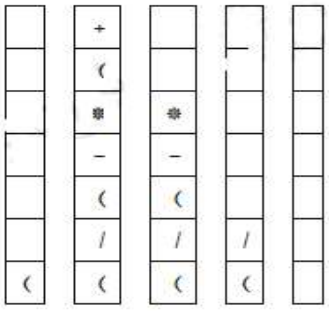
+-*↑ABCD/E/F+GH

پاسخ:

+ - * A^B CD/E/F +GH
+ - * A^B CD/E/F G+H
+- A^B*C D/E/F G+H
+-A^B*C D/E F/(G+H)
+-A^B*C D E/(F/(G+H))
+ A^B*C-D E/(F/(G+H))
A^B*C-D + E/(F/(G+H))

عبارت میانوندی (A / (B-C* (D+E) را به کمک پشته به معادل پسوندی تبدیل نمایید.

پاسخ: ابتدا یک پرانتز باز در پشته push کرده و یک پرانتز بسته به آخر عبارت میاوندی اضافه می کنیم و طبق الگوریتم بالا عملیات را انجام می دهیم:



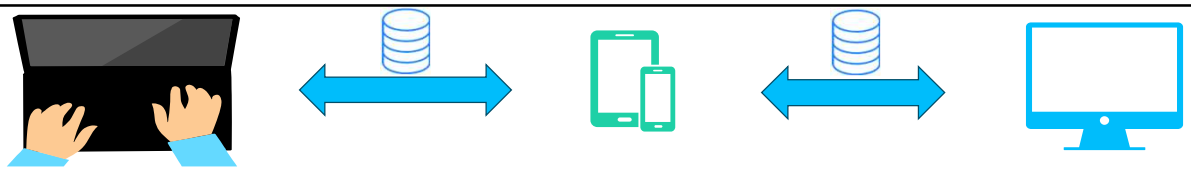
مقدار عبارت P در هر یک از حالت ها در زیر نشان داده شده است:

P = ABCDE
P = ABCDE+
P = ABCDE+*-
P = ABCDE+*-/

پرسش های فصل سوم

مقدار نهایی A,B,C کدام است.(n = ۵ , A = ۱۰ , B = ۲ , C = ۵)

push(B)	push(A*B)
push(A+B)	push(C)
C=pop()	push(A)
push(A-B)	B=pop()
push(C)	C=pop()
push(B)	A=pop()
A=pop()	B=pop()



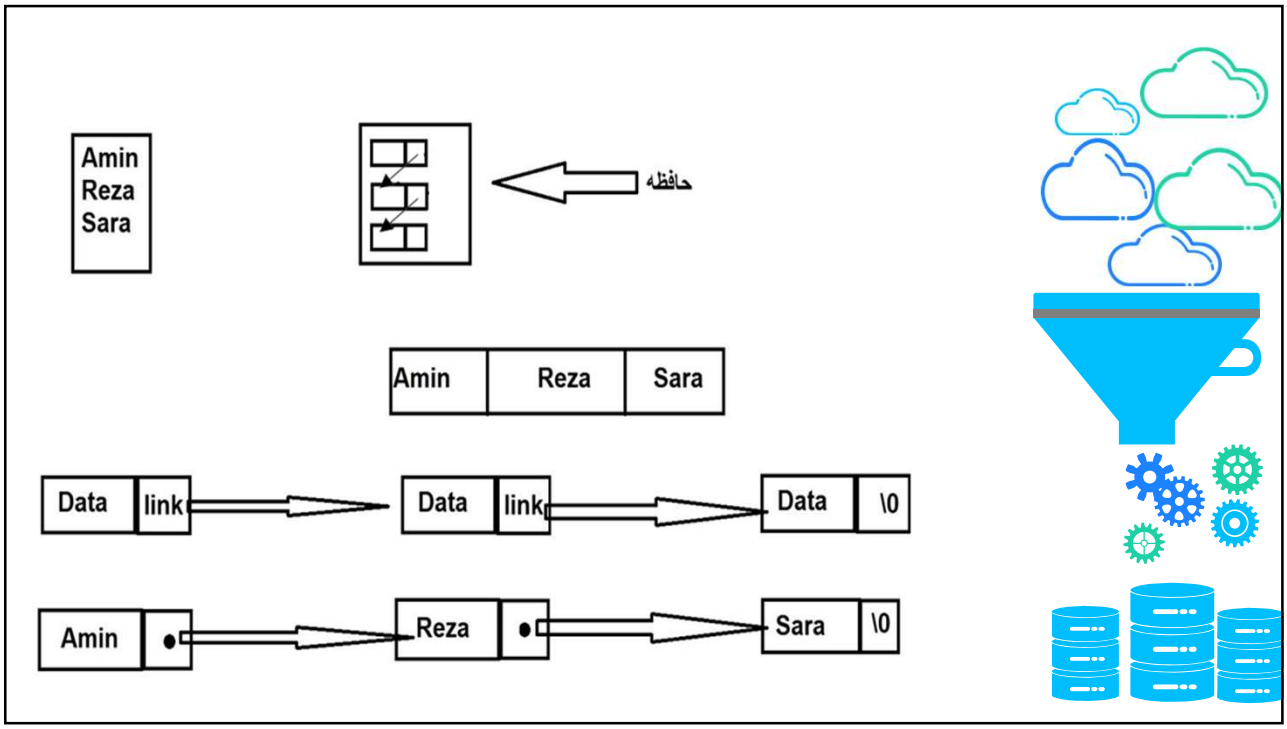
۲. هر یک از عبارات زیر را به عبارات prefix و postfix تبدیل کنید.

a. *A+B-C*
b. *(A+B)*(C-D)-E*F*
c. *A+B/C+D*
d. *A-(B-(C(D-E)))*

فصل ۴
لیست پیوندی

غالبا برای ذخیره داده ها هم می توان از آرایه استفاده کرد و هم از لیست پیوندی عناصر آرایه الزاما در حافظه پشت سر هم قرار گرفتند ولی عناصر لیست پیوندی از نوع پویا بوده و عناصر آن الزاما در کنار یک دیگر نمی باشند به همین دلیل اعمال درج و حذف در لیست پیوندی ساده تر و خیلی سریع تر از آرایه انجام می گیرد.

لیست پیوندی از مجموعه ای از عناصر تشکیل شده است هر عنصر یا گره یا نود در لیست پیوندی حداقل از دو فیلد داده (data) و اشاره گری به گره بعدی (link) تشکیل یافته است.



لیست های پیوندی را به ۳دسته تقسیم می کنند.

۱. لیست ها یک طرفه هستند یا دو طرفه

۲. لیست ها یا خطی هستند یا حلقوی.

۳. لیست ها یا بدون گره سر هستند یا گره سر دارند.

لیست پیوندی یک طرفه خطی:

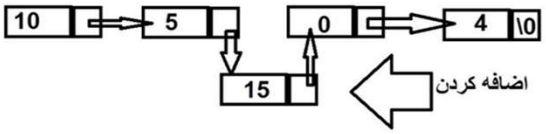
در لیست پیوندی یک طرفه خطی در هر گره فقط آدرس گره بعدی ذخیره می شود و اشاره گره آخرین گره نیز برابر **\0** هست

اضافه کردن به لیست یک طرفه:

لیستی که در آن هر عنصر فقط آدرس عنصر بعدی خود را نگه می دارد لیست یک طرفی یا لیست خطی نام دارد

ساختار تولیدnode

```
struct  node
{
int data
int * link
}
 int *head
add list(item)
stract node y;
y.data=item
if(head==null
{
nead=y;
y.link=null
}else
}
y.link=x.link
x.link=y
}
```

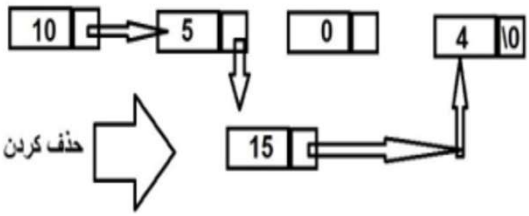


حذف از لیست یک طرفه

برای حذف گره با آدرس X ابتدا باید لیست را تا یافتن گره قبل از آن پیمایش کرد.

الگوریتم زیر این روش را نشان می دهد

```
temp=head
while(temp.link != x)
{
temp=temp.link;
}
temp.link=x.link
delete(x);
```



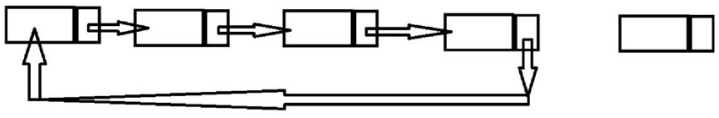
لیست حلقوی یک طرفه:

لیست حلقوی مشابه لیست یک طرفه است با این تفاوت که اشاره گر آخرین گره به جای اینکه **null** باشد به سر لیست اشاره می کند **(head)** در لیست خطی همواره باید آدرس سر لیست را داشته باشیم ولی در لیست حلقوی با داشتن هر گونه دلخواه می توان به تمامی گره ها دسترسی داشت.

کلیه الگوریتم ها لیست حلقوی و لیست خطی(مشابه یک دیگر)فرق دارند

```
struct node * deleteFirst() {
struct node *tempLink = head;
if(head->next == head) {
head = NULL;
return tempLink;
}
head Link= head->next;
return tempLink;
```

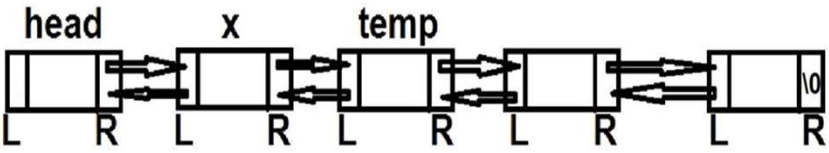
حذف کردن



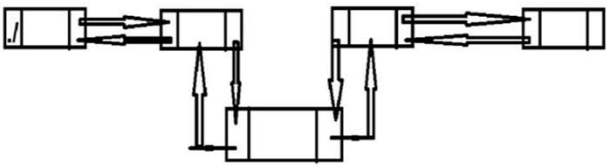
لیست پیوندی دو طرفه Two Way List (TWL)

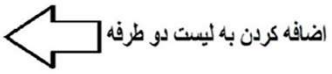
در لیســت دو طرفه یا (TWL)در هر گره دو اشـاره گر وجود دارد که یکی به گره بعدی و دیگری به گره قبلی در لیست اشاره می کند.

به کمک اشاره گر های سمت راست (R.link)و سمت چپ (L.link) می توان در هر دو طرف لیست حرکت کرد بنابراین با داشتن یک گره کلیه گره ها قابل دسترسی اند.



اضافه کردن به لیست پیوندی دو طرفه

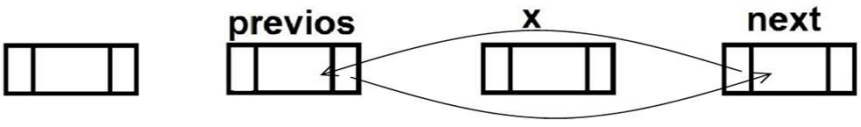




x.L link=previos
x.R link=next

previos.R link=x
next.L link= x

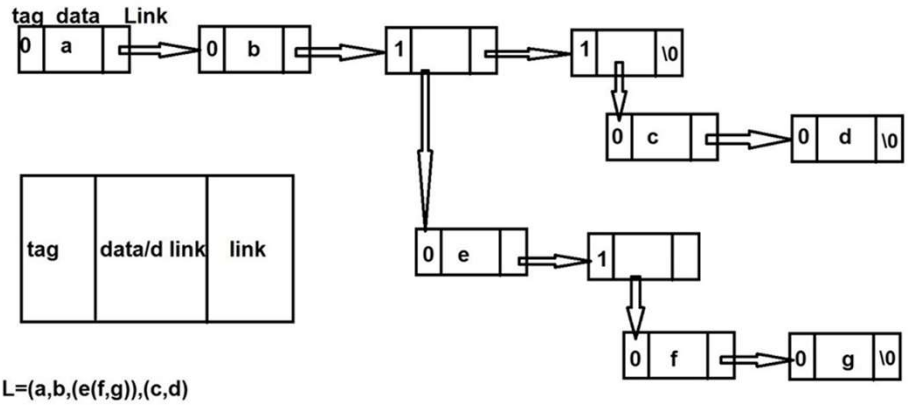
حذف از لیست پیوندی دوطرفه:



previos.R link=X.R link
next.L link=X.L link

لیست عمومی:

لیستی که هر گره آن بتواند خود یک زیر لیست باشد لیست عمومی نامیده می شود.به عبارتی دیگر لیست عمومی (L) از تعداد محدودی عنصر A۱,A2,...Anتشکیل شده به گونه ای که هر (AI)یک عضو ساده و یا یک لیست میباشد



L=(a,b,(e(f,g)),(c,d)

تمرینات فصل ۴

سوالات تستی :

۱- فرض کنید $x=(x_1,x_2,\cdots,x_n)$ و $y=(y_1,y_2,\cdots,y_m)$ دو لیست پیوندی خطی ساده باشند. مرتبه زمانی الگوریتمی که دو لیست را در لیست z ترکیب می‌کند، چیست؟

$$Z=\begin{cases}(x_1,y_1,x_2,\cdots x_m,y_m,x_{m+1},\cdots,x_n) & ,\ if\ m\le n\\ (x_1,y_1,x_2,\cdots x_n,y_n,x_{n+1},\cdots,x_m) & ,\ if\ m>n\end{cases}$$

۱) $O(\max(m,n))$ ۲) $O(m+n)$ ۳) $O(mn)$ ۴) $O(\min(m,n))$

پاسخ: جواب گزینه ۴ است.

با توجه به شرایط داده شده، دو لیست را به صورت یک درمیان ادغام کرده تا لیست کوچکتر تمام شود. سپس بقیه لیست بزرگتر را به انتهای لیست بدست آمده، اضافه می کنیم. بنابراین همواره به اندازه لیست کوچکتر، عمل پیمایش انجام می شود. بنابراین برای دو لیست به طول های n و m ، مرتبه الگوریتم برابر است با : $O(\min(m,n))$

۲- فرض کنید بخواهیم ترتیب اشاره گرها را در یک لیست تک پیوندی وارون نمائیم. به عبارت دیگر، هر اشاره گر به جای اشاره به عنصر بعدی به عنصر قبلی اشاره نماید. برای انجام این کار کدام یک از گزاره های ذیل درست است؟ (الگوریتم غیر بازگشتی)

۱) ۳ اشاره گر به جز نام لیست مورد نیاز است.

۲) ۲ اشاره گر به جز نام لیست مورد نیاز است.

۳) اشاره گر دیگری غیر از نام لیست، مورد نیاز نیست.

۴) کافی است از یک استک استفاده کنیم، لذا اشاره گر بالای استک و نام لیست، مورد نیاز است.

حل: جواب گزینه ۳ است.برای معکوس کردن یک لیست یکطرفه به ۳ اشاره گر نیاز است.

اگر X اشاره گری به لیست دو پیوندی باشد ، دستورات زیر چه عملی انجام می دهند؟

```
p->link=x
p->rlink = x->rlink
x->rlink ->llink= p
x->rlink = p
```

۱) حذف گره قبلی

۲) حذف گره بعدی

۳)حذف خود X

۴)درج X

گزینه ۳ درست است

می خواهیم تغییراتی در یک لیست پیوندی اعمال کنیم که عمل افزودن عنصر ابتدا و یا انتهای لیست با عملیاتی از مرتبه O(1) قابل انجام باشد. لیست پیوندی را .....

الف. حلقوی می کنیم.
ب. دو طرفه می کنیم.
ج. معکوس می کنیم
د. حلقوی می کنیم و آدرس آخرین عنصر را برای دسترسی به لیست ذخیره می کنیم.

گزینه ۴ درست است

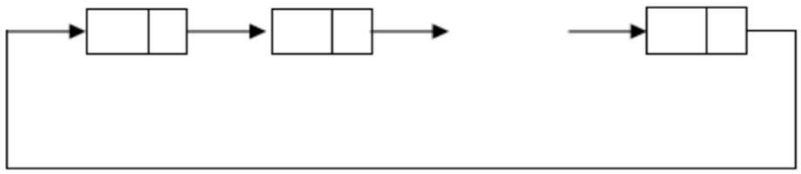
سوالات تشریحی

تابعی بنویسید که اشاره‌گر و لیست پیوندی را بگیرد و تعداد گره‌های لیست را برگرداند.

```
int count (Node  *ptr )
{
    Node   *list ;
    int  c=0;
    list = ptr;
    if( list ==NULL)
        return 0;
    while(list)
    {
        list = list → next;
        c ++;
    }
    return c;
}
```

می‌خواهیم تغییراتی در یک لیست پیوندی اعمال کنیم که عمل افزودن ابتدا یا انتهای لیست با عملیاتی از مرتبه O(1) قابل انجام باشد. به نظر شما از چه نوع لیستی برای این کار استفاده کنیم.

حل: برای حل این مسئله می‌بایست از یک لیست حلقوی استفاده کنیم:



آدرس آخرین گره را می‌بایست ذخیره کنیم. با داشتن آدرس آخرین گره می‌توان اعمال خواسته شده را در زمانی با مرتبه O(1) انجام داد.

قطعه کدی بنویسید که یک گره را به لیست پیوندی اضافه کند.

```
j= getnode();
j → inf o = item;
if ( head == NULL)
   {
      head=j;
      j → next = NULL;
}
else
   {
    j → next = list → next;
    list → next = j;
}
```

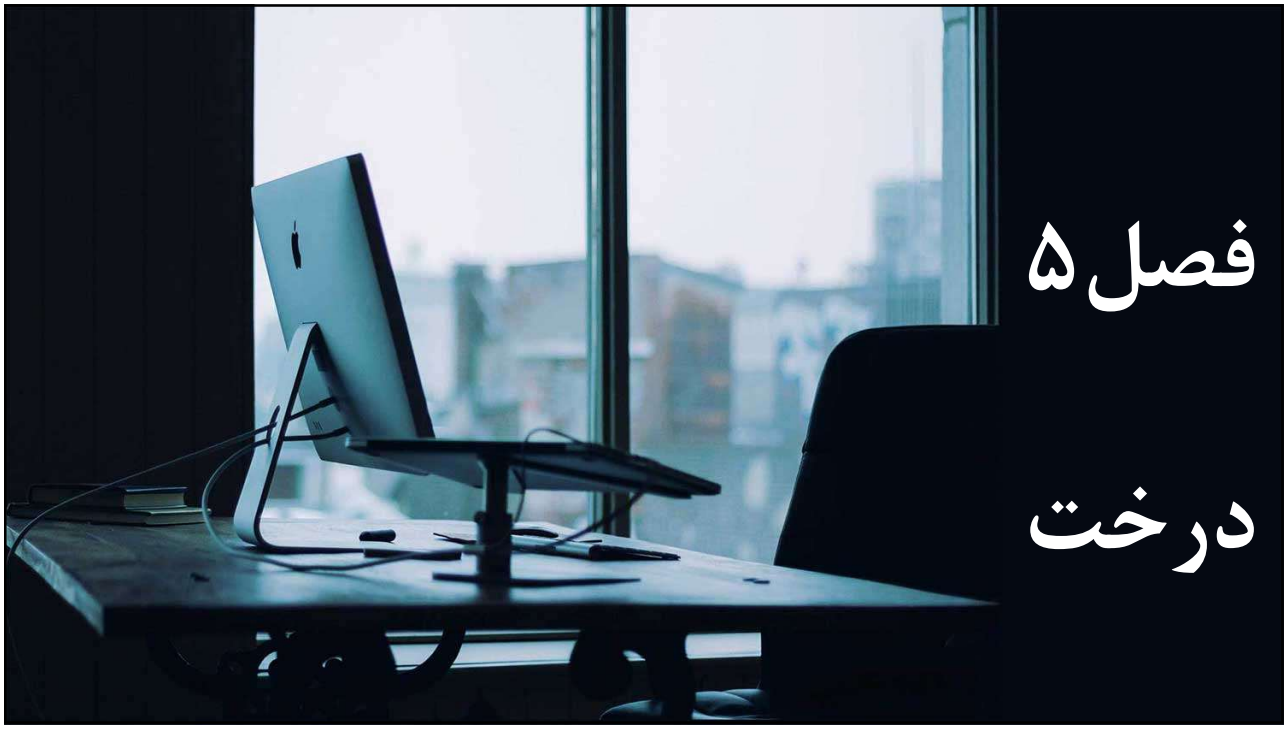
تابعی بنویسید که داده و لیست پیوندی را از آخر به اول چاپ کند.

```
void  print_reversal( Node  *list )
{
      if (list!= NULL)
       {
          print_reversal( list → next ) ;
          cout<< list → inf o ; // printf("%f", list → inf o );
       }
}
```

پرسش های فصل چهارم

الگوریتم زیر، لیست L را وارون می کند. اگر در دستور if ، قسمت دوم شرط or نباشد، آنگاه خروجی چه خواهد بود؟

```
reverse(L)
1  if L=null or next[L]=null
2     then return L
3  else r ← reverse(next[L])
4       next[next[L]] ← L
5       next[L] ← null
6  return r
```



درخت یک ساختمان داده غیر حقیقی است و مجموعه ای محدودی از یک یا چند گره به صورت زیر است:

)۱دارای گره خاصی به نام ریشه است.

)۲بقیه گره ها به n>= ۰ مجموعه مجزا یعنی t1….,tn تقسیم شده است که هر کدام از این مجموعه ها خود یک درخت هستند و t1….,tn نیز زیر درختان ریشه نامیده می شوند.

نکته :در درخت حلقه وجود ندارد .

درجه یک گره :تعداد زیر درخت های یک گره درجه آن نام دارد.

برگ :گره هایی که درجه ۰ دارند برگ یا گره هایی پایانی نامیده می شود.

درجه یک درخت: حداکثر درجه گره های آن درخت می باشد

گره های هم ذات: فرزندان یک گره هم ذات نامیده می شود

سطح یک گره:

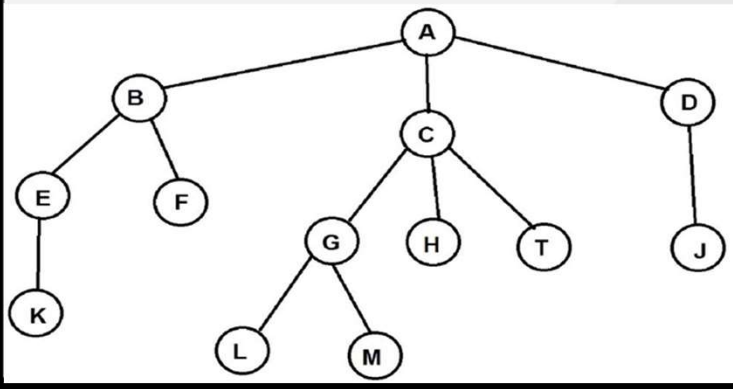
ریشه در سطح ۱ قرار دارد و سایز گره ها بر اساس تعداد مکانی که از ریشه فاصله دارند شماره سطح نامیده می شود.

گره هایی با فاصله n مکان از ریشه در سطح ۱ +n قرار دارند.

اگر گره ای در سطح k باشد فرزندان آن در سطح ۱ +k قرار دارند.

ارتفاع یا عمق یک درخت:

به بیش ترین سطح گره های آن درخت گفته می شود



عمق=4

B,C,Dهم ذات هستند

E,F,G,H,T,Jهم ذات هستند

تعریف یال و مسیر :هر خط یا اتصال از یک گره به گره دیگر یا هر انشعاب از گره به گره دیگر را یال می گوییم و دنباله ای از یال های متوالی یک مسیر نامیده می شود.

شاخه: مسیری که به یک برگ ختم می شود یک شاخه نام دارد

درخت مرتب :درختی است که ترتیب زیر درخت ها در آن مهم باشد

درخت مشابه :درخت های t ۱t ۲,مشابه هستند اگر دارای ساختمان داده یکسان باشند یا به بیان دیگر شکل مساوی داشته باشند.

.درخت هارا کپی می گویند اگر مشابه بوده و محتوای گره های متناضر آن نیز یکی باشد

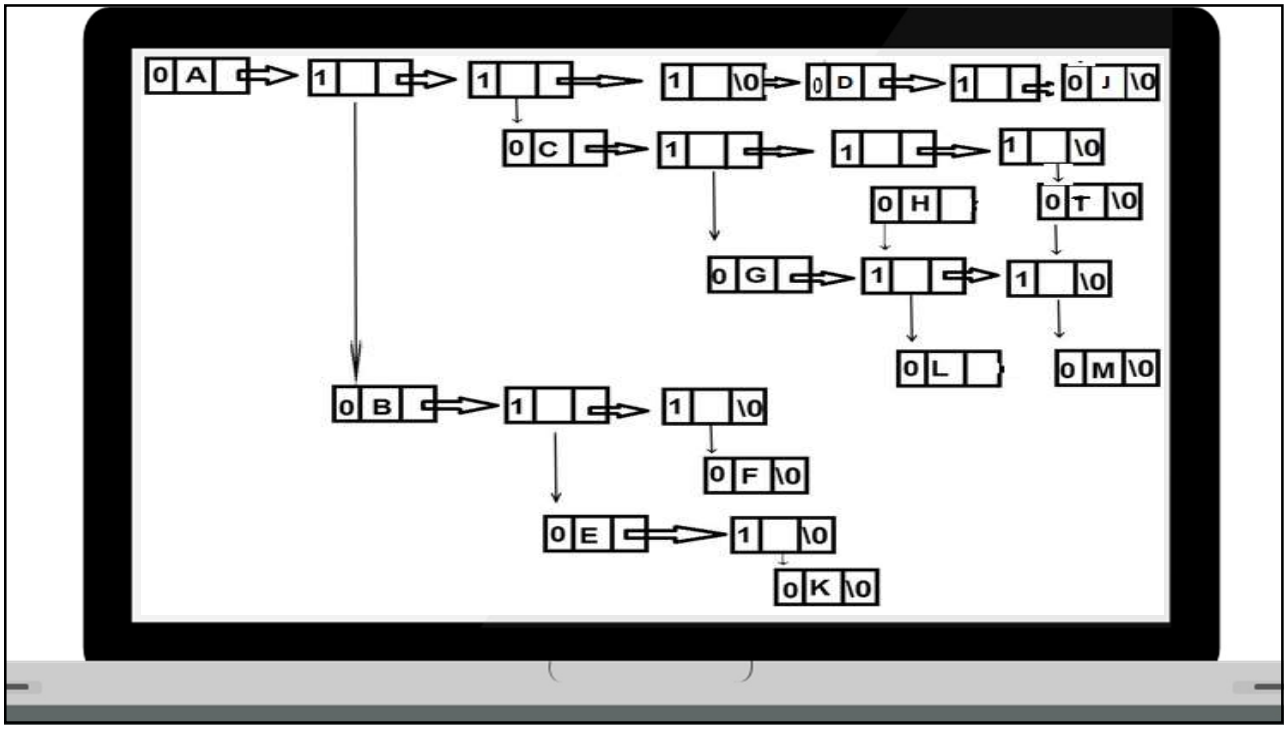
درخت k تایی: درختی که فرزندان هر گره در آن حداکثر k باشد.

درخت متوازن :درختی است که اختلاف سطح برگ های آن حداکثر یک باشد و اگر اختلاف سطح برگ ها ۰ باشد آنگاه درخت کاملا متوازن است.

نمایش درخت :برای نمایش درخت می توان از فرم پرانتزی استفاده کرد.در این فرم ابتدا اطلاعات ریشه و سپس داخل پرانتز ها اطلاعات فرزندان آن گره به ترتیب از چپ به راست نوشته می شود.

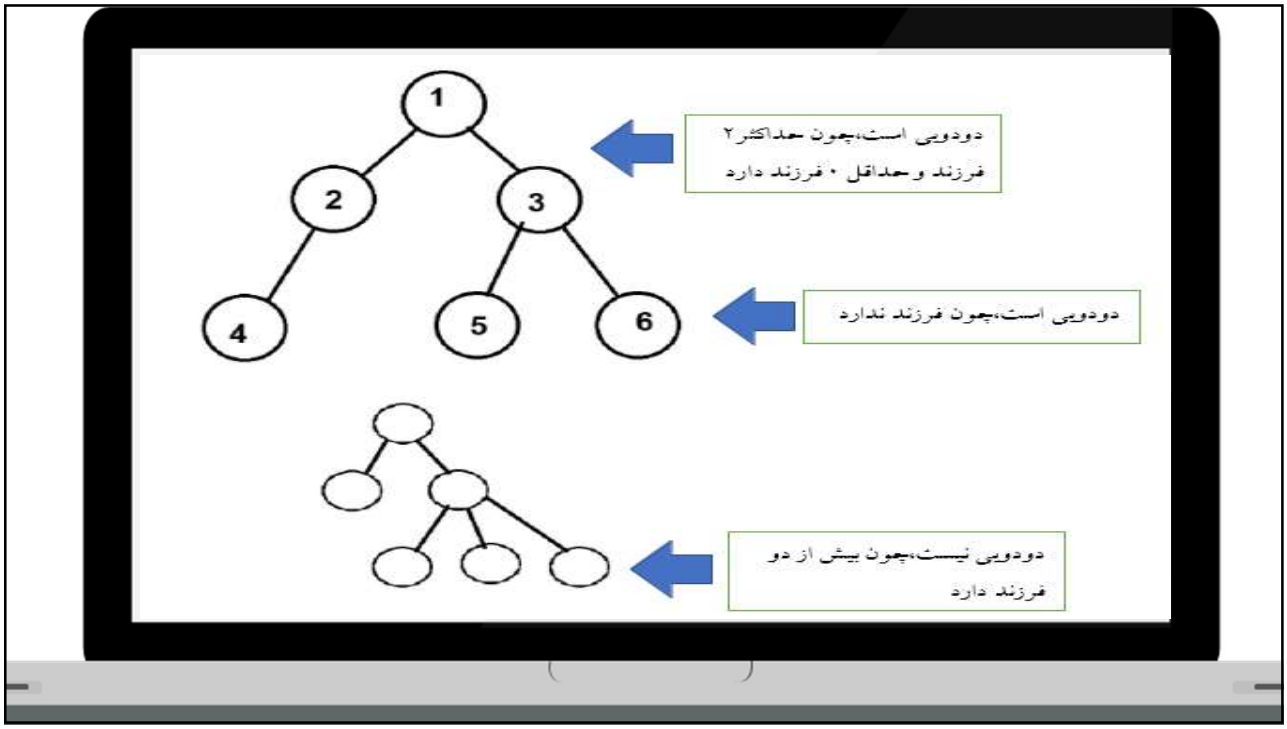
A(B(E(K),F),C(G(L,M)H,T),D(J))

درخت را می توان به صورت یک لیست پیوندی عمومی نیز نمایش داد در این سطح نمایش فرزندان هر گره در سمت راست آن ترسیم شده و هر فرزند که برگ نباشد با اضافه کردن یک سطح به لیست به صورت یک زیر لیست نمایش داده می شود.



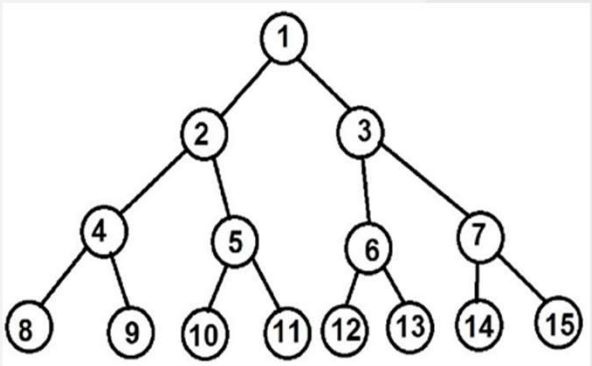
درخت دودویی : یک در خت دودویی یک ساختمان داده درخت است که در آن هر گره حداکثر دو گره فرزند دارد که فرزندان راست و چپ نامیده می شوند.

- در درخت دودویی،درجه هر گره حداکثر می تواند دو باشد.
- درخت های دودویی برای پیاده سازی درخت جست و جوی دودویی و انبوه دودویی و برای جست و جوی کارآمد و مرتب سازی استفاده می شود.
- درخت دودویی یک حالت خاص از یک درخت K تایی است که در آن K برابر ۲ است.
- در درخت دودویی ترتیب زیر درخت ها مهم است.



انواع درخت دودویی:

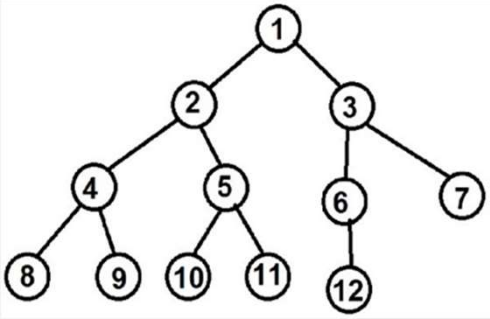
)۱درخت پر: درختی است که به جز گره های سطح آخر دقیقا دو فرزند داشته باشد.



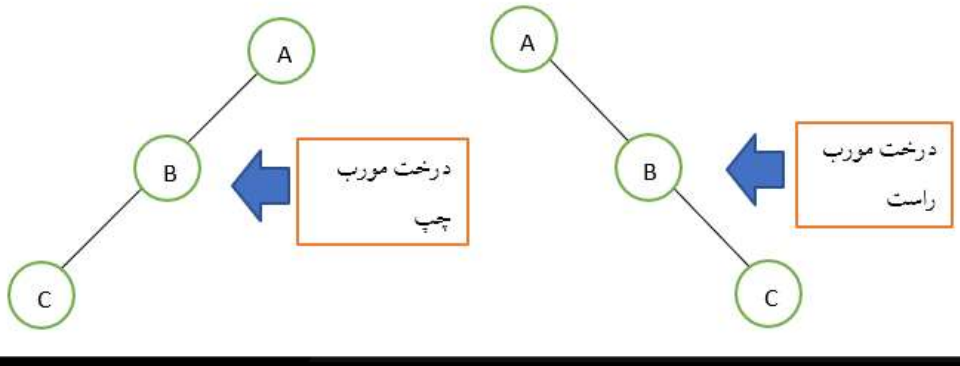
۲درخت کامل :درختی را کامل گویند که تمام سطوح آن به جز احتمالا سطر آخر حداکثر تعداد گره های ممکن را داشته باشد.

همچنین تمام گره های سطر تا حد ممکن در سمت چپ و دور ترین مکان آن باشد.

نکته :درخت پر هم کامل است هم متوازن



۳)درخت مورب :در درخت مورب چپ هر گره فرزند چپ والد خود می باشـد و در درخت مورب راسـت هر گره فرزند راست والد خود می باشد.



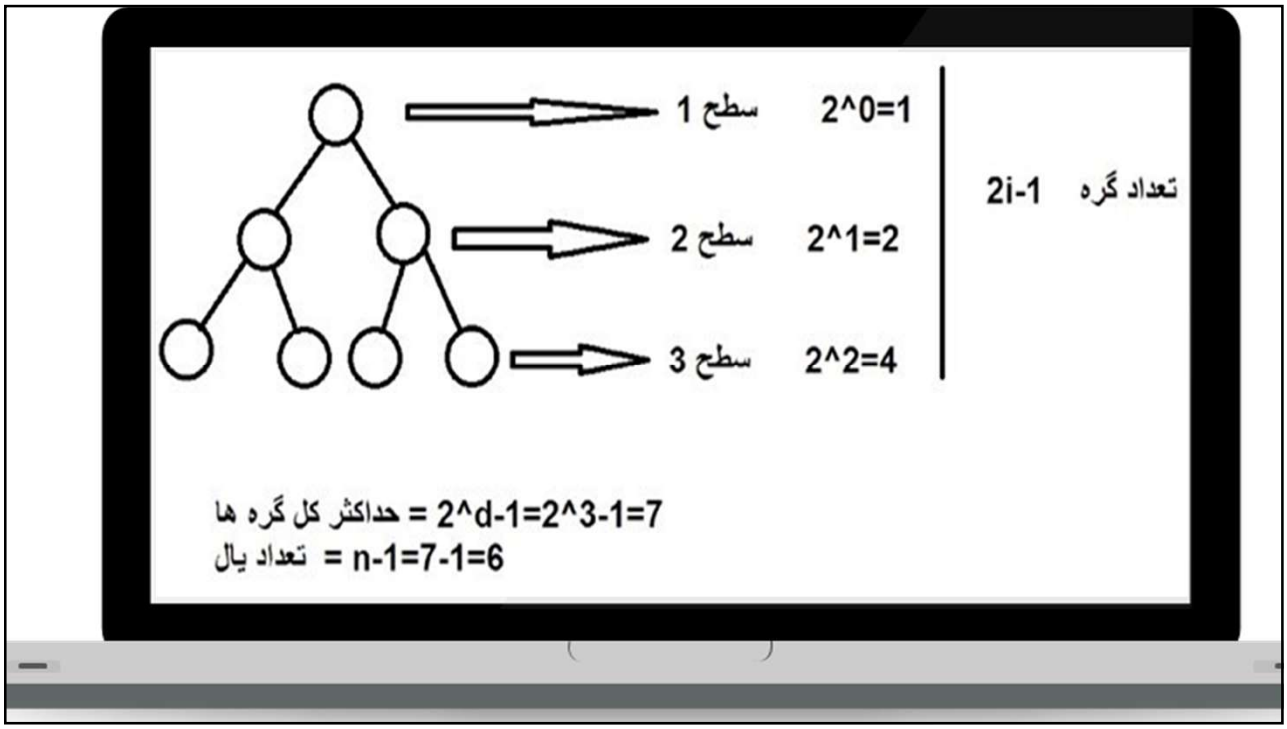
فرمول های مربوط به درخت دودویی:

حداکثر تعداد گره ها در سطح (i) ام یک درخت دودویی 2^{i-1} می باشد که i>= 0 می باشد .

حداکثر تعداد کل گره ها در یک درخت دودویی به عمق D برابر با۱- 2^{D} می باشد D>=۱

در هر درخت تعداد یال ها برابر با ۱ -n است که n تعداد گره هاست.

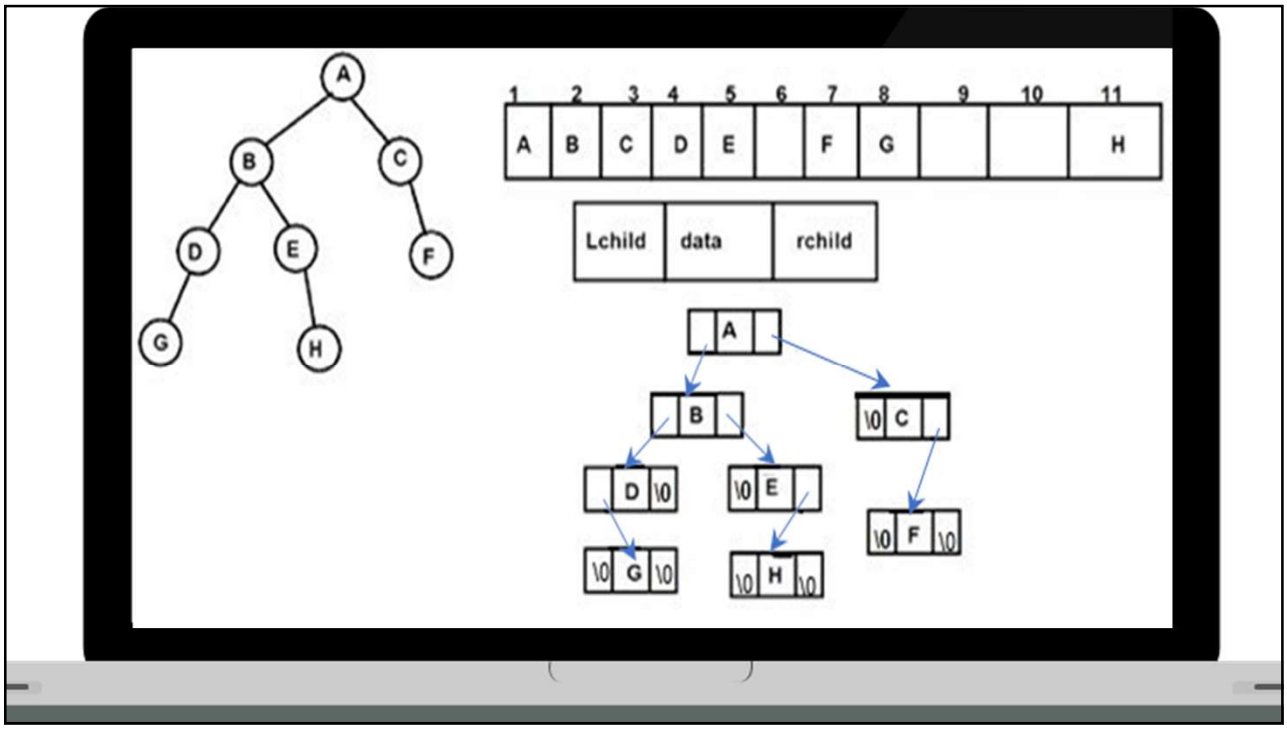
تعداد درخت های دودویی مجاز با n گره برابر $\frac{1}{n+1}\binom{2n}{n}$



نمایش درخت دودویی:

شماره گذاری گره های درخت دودویی می توان درخت را در آرایه ذخیره کرد به طوری که محتوای هر گره در خانه ای از آرایه با همان شماره ذخیره شود.

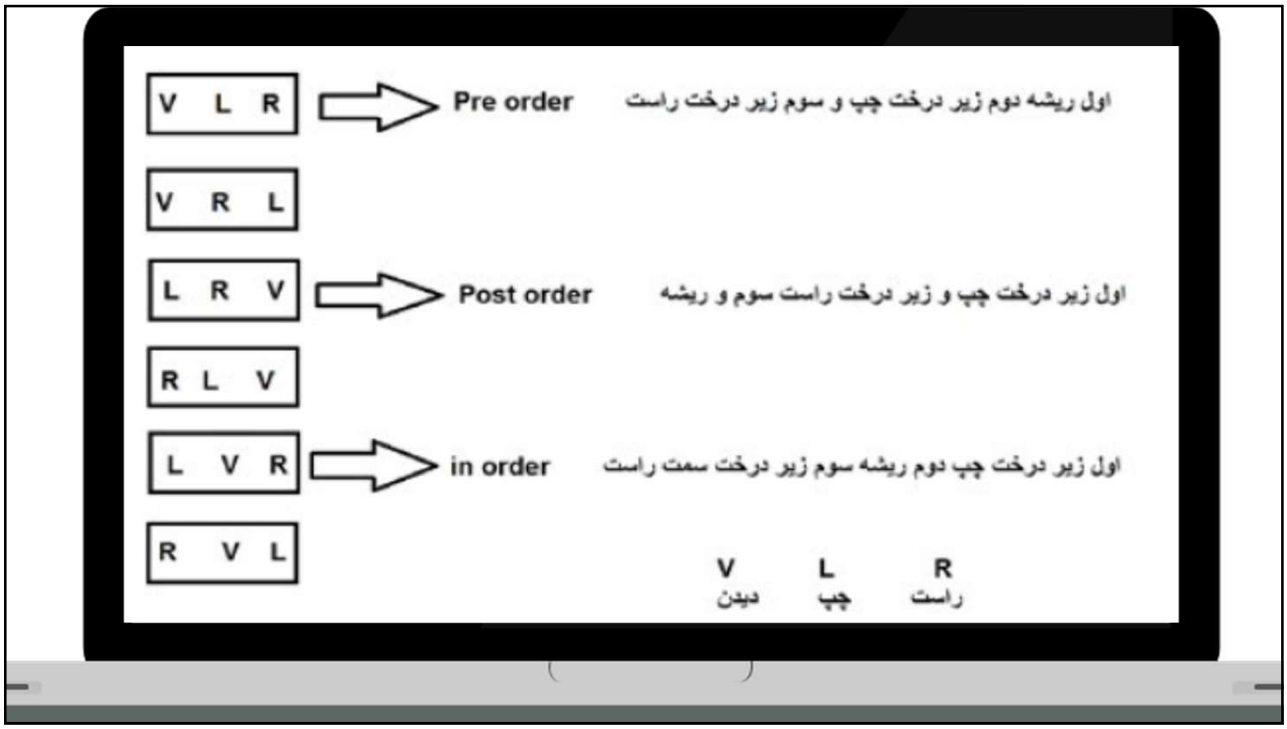
استفاده از فرم آرایه برای درختـان کامل مناسب است چون هیچ خانه ای از حـافظـه هدر نمی رود ولی برای درخت های مورب کاملا نامناسب است چون فضای زیادی به هدر می رود ایراد نمایش درخت با آرایه این است که حذف یا درج گره ها و مستلزم جابه جایی گره هاست که خود باعث شماره سطر گره ها می شود و مشکل فوق با استفاده از نمایش پیوندی قابل حل است.

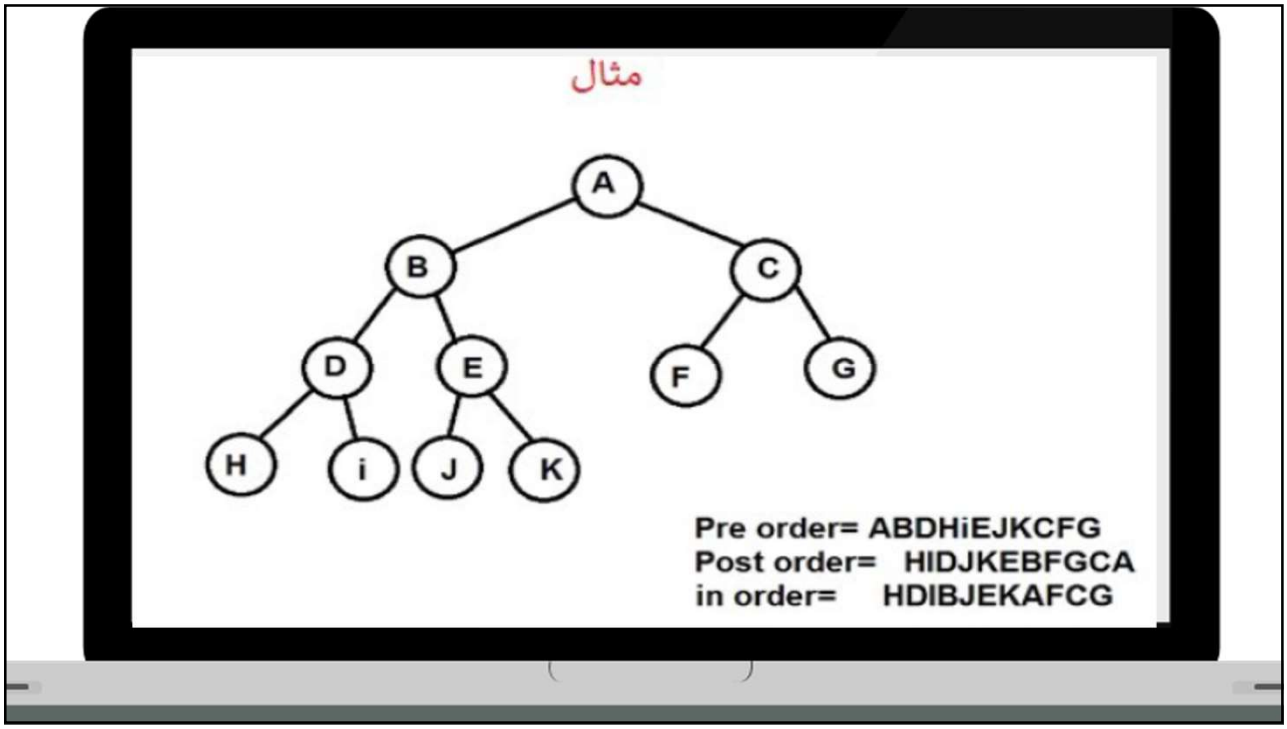


پیمایش درخت دودویی:

در پیمایش درخت می خواهیم هر گره درخت فقط یک بار ویزیت شود به هنگام پیمایش یک درخت با هر گره و زیر درختانش به طرز مشابهی رفتار کنیم.

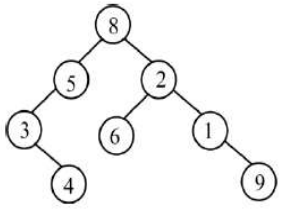
اگر L,V,R (L به ترتیب حرکت به چپ، V ملاقات کردن یک گره و R حرکت کردن به راست باشد)آنگاه ٦ ترکیب به دست خواهد آمد که سه ترکیب آن برای ما مهم است


V L R
Pre order
اول ریشه دوم زیر درخت چپ و سوم زیر درخت راست
V R L
L R V
Post order
اول زیر درخت چپ و زیر درخت راست سوم و ریشه
R L V
L V R
in order
اول زیر درخت چپ دوم ریشه سوم زیر درخت سمت راست
R V L
V دیدن
L چپ
R راست


مثال
A
B
C
D
E
F
G
H
i
J
K
Pre order= ABDHiEJKCFG
Post order= HIDJKEBFGCA
in order= HDIBJEKAFCG

مثال

حاصل پیمایش های معمول درخت زیر را بدست آورید.



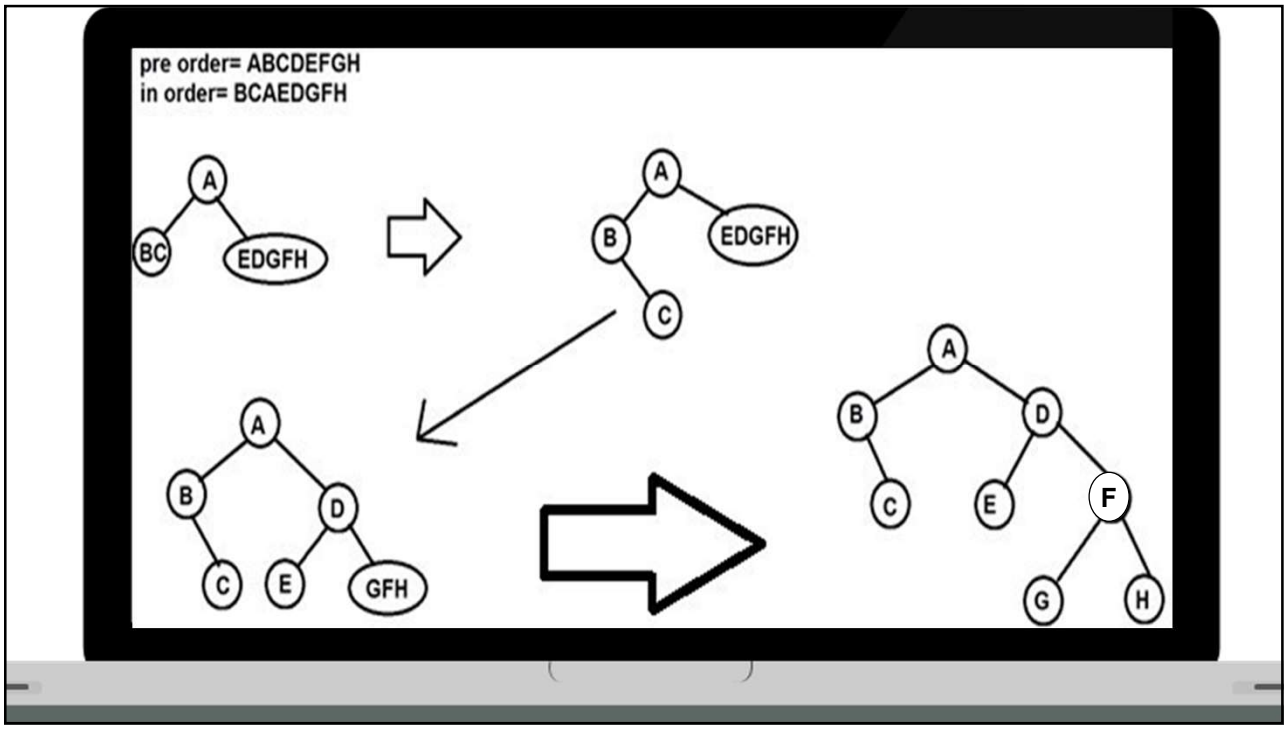
inorder : 34586219
preorder : 85342619
postorder : 43569128

ترسیم درخت به کمک پیمایش آن ها:

۱)اگر پیمایش های میانوندی وپسوندی درخت دودویی را داشته باشیم می توانیم آن درخت را به صورت یکتا رسم کنیم.

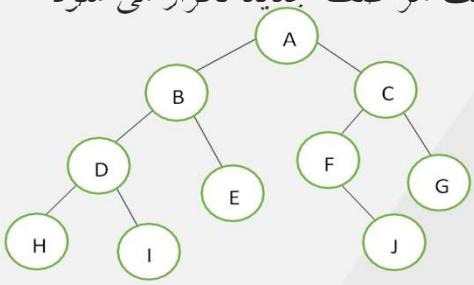
۲)اگر پیمایش های میانوندی و پیشوندی درخت دودویی را داشته باشیم می توانیم آن درخت را به صورت یکتا رسم کنیم.

3)اگر پیمایش های پیشوندی یا پسوندی یک درخت دودویی را داشته باشیم ممکن است نتوانیم آن درخت را به صورت یکتا رسم کنیم



پیمایش ترتیبی سطحی:

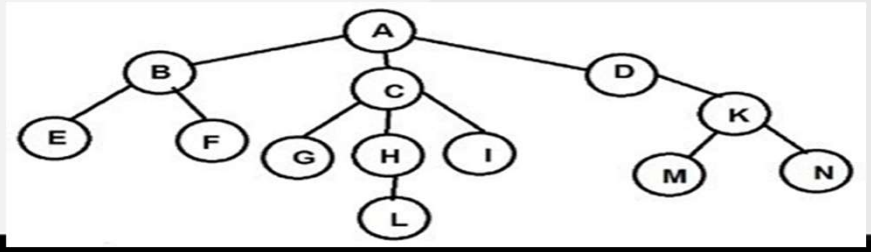
پیمایش های post order , pre order , in order در واقع پیمایش های عمقی هستند که برای آنها از پشته استفاده می شود.درپیمایش سطحی از یک صف استفاده می شود و در این روش ابتدا ریشه سپس فرزند چپ ریشه و به دنبال آن فرزند راست ریشه بازیابی می گردد.این شیوه با بازیابی از گره های منتها علیه سمت چپ به راست هر صف جدید تکرار می شود



ABCDEFGHIJ

درخت عمومی: General tree

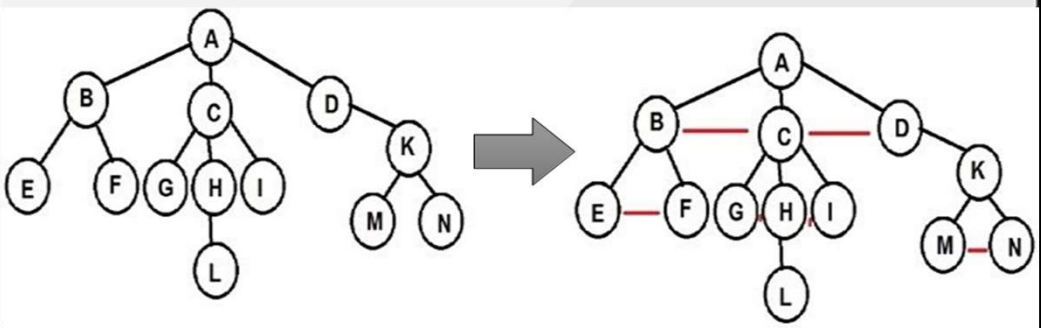
درخت عمومی یا درخت کلی یک درخت K تایی است که در آن فقط یک گره به نام ریشه با درجه ورودی ۰ وجود دارد و سایر گروه ها دارای یک مکان ورودی هستند از این به بعد فرض می کنیم یک درخت عمومی مرتب است یعنی بچه های هر گره یک ترتیب مشخص دارند هر چند که با این خاصیت همواره برای تعریف درخت عمومی کافی نیست.



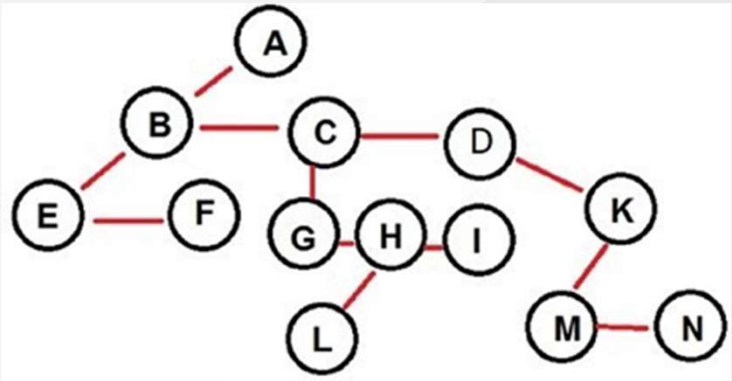
معمولا به دلیل اتلاف زیادی فضای حافظه بهتر است درخت های عمومی را با استفاده از الگوریتم به یک درخت دودویی تبدیل کرد

تبدیل درخت عمومی به یک درخت دودیی:

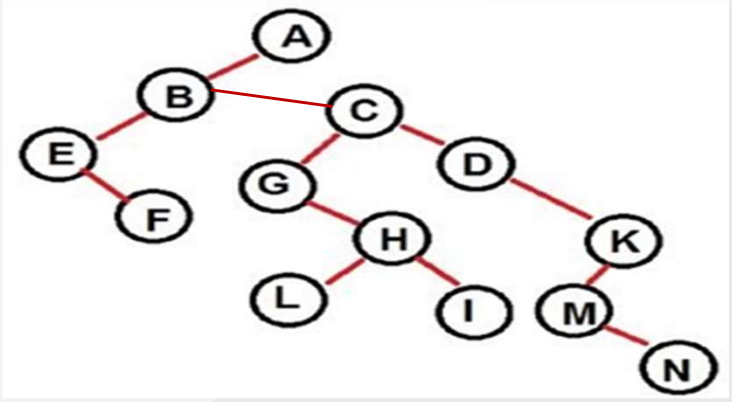
۱در هر سطح گره های کنار هم که فرزند یک پدر هستند را به یک دیگر وصل می کنیم)



۲)ارتباط کلیه گره ها به جز اتصال سمت چپ ترین فرزند و اتصال هم سطح را قطع می کنیم.



۳گره های متصل به هم در هر سطح افقی را ٤٥درجه در جهت حرکت عقربه های ساعت می چرخانیم.



درختان ویژه:

برخی از درختان از جمله درخت Huff man , Heap , BCD , AVL ،انتخابی و درخت تقسیم را مورد بررسی قرار خواهیم داد.یکی دیگر از درخت های پر استفاده Btree در مباحث مربوط به فایل ها به کار برده می شود.

درخت (Heap هرم نیمه مرتب)

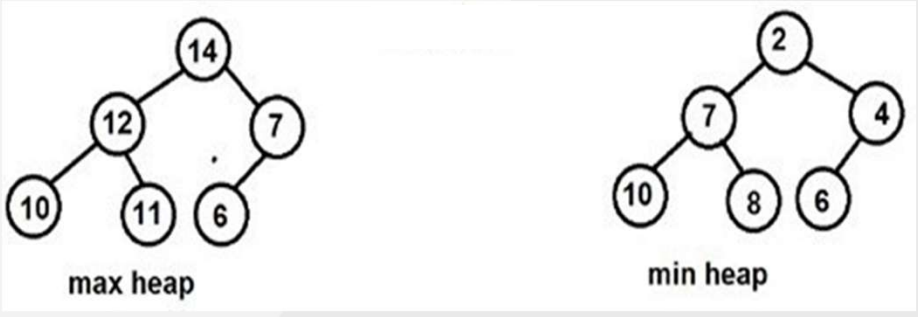
تعریف max tree: درختی است که مقدار هر کلید گره آن بزرگتر یا مساوی کلید های فرزند هایش باشد.

تعریف min tree: درختی است که مقدار هر کلید گره آن کوچکتر یا مساوی کلید های فرزند هایش باشد .

تعریف max heap :درخت کامل دودویی است که max treeنیز باشد

تعریف min heap : :درخت کامل دودویی است که min Tree نیز باشد

- توجه کنید که کامل بود شرط لازم برای heap بودن درخت است.
- ریشه درخت min heap حاوی کوچک ترین کلید درخت است.
- ریشه درخت max heap حاوی بزرگترین کلید درخت است.

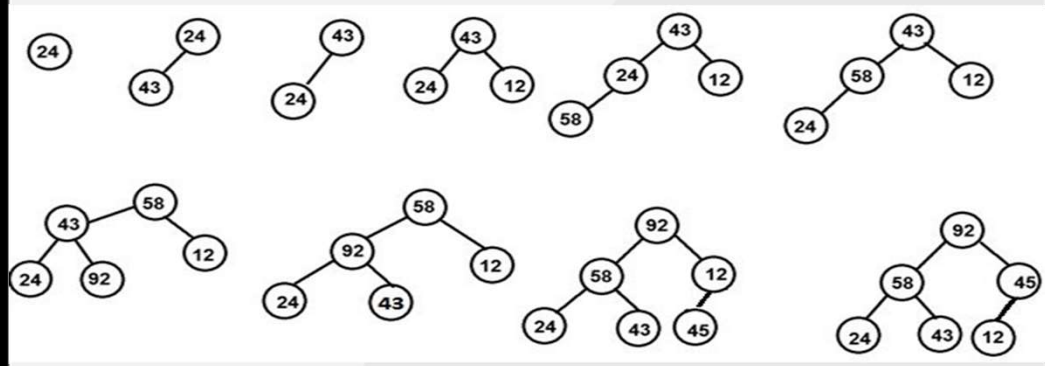


در اســـتفـاده از درختـان نیمـه مرتـب اعمـال درج و حـذف در درخـت بـه صـــورتی انجـام می گیرد کـه درخـت بـه صـورت نیمه مرتب باقی بماند.

عمل درج به این صـورت اسـت که همواره درخت از چپ به راسـت در هر سـطح آخر پر شـده و پس از پر کردن از سـطح بعدی انجام می شـود.هنگام ورود یک گره جدید ابتدا این عنصـر در سـطح آخر (که هنوز پر نشـده)در چپ ترین جای خالی قرار می گیرد سـپس عمل مرتب سـازی درخت انجام می شود در این عمل یک گره در پایین ترین سطح تا جایی که لازم باشد با گره پدرش جا به جا می شود

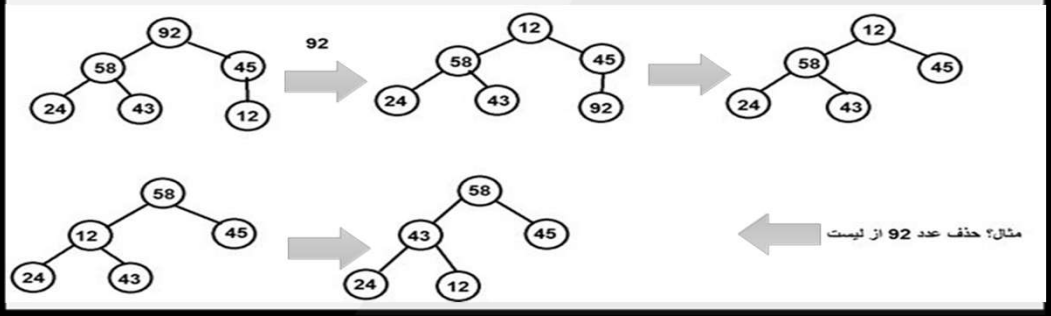
مثال:

اگر اعداد ورودی از چپ به راست به صورت زیر باشد درخت max heap آن را رسم کنید ٤٥-٩٢-٥٨-١٢-٤٣-٢٤



حذف عنصر از : max heap

در عمل حذف همواره مقدار ریشه حذف شده وسمت راست ترین عنصر موجود در پایین ترین سطح در ریشه قرار گرفته و درخت مجددا تنظیم میشود در تنظیم max heapعنصر ریشه تا جایی که از فرزندان خود کوچکتر است با بزگترین آن جای خود را عوض می کند و درخت min heap برعکس این قضیه است



جست و جوی عنصر دردرخت جستجوی دودویی : BST

برای پیدا کردن گرهی با یک کلید خاص مانند x در درخت ابتدا باید از ریشه درخت شروع کنیم اگر ریشه تهی باشد،درخت فاقد هر عنصری بوده و جست و جو ناموفق خواهد بود.

در غیر این صورت x را با مقدار گره ریشه مقایسه می کنیم اگر برابر بودند جست و جو موفق است و گره ریشه همان گره مورد نظر است

در غیر این صورت دو حالت پیش خواهد آمد:

۱) x ۱ از گره ریشه کوچکتر است در این حالت هیچ عنصری در زیر درخت سمت راست وجود ندارد که مقدار کلید آن برابر با x باشد بنابراین جست و جو باید در زیر درخت سمت چپ ادامه یابد

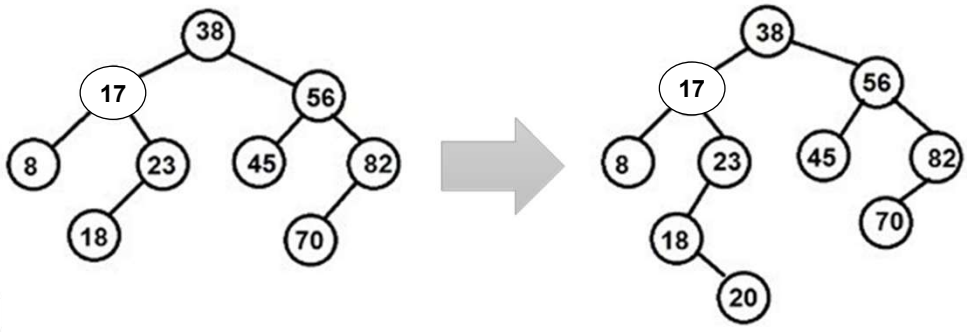
۲)x بزرگتر از ریشه است در این حالت هیچ عنصری در زیر درخت سمت چپ وجود ندارد که مقدار کلید آن برابر با x باشد بنابراین جست و جو باید در زیر درخت سمت راست ادامه یابد.

سپس بسته به حالت یک و دو زیر درخت سمت چپ یا زیر درخت سمت راست را به روش بازگشتی و با استفاده از الگوریتم بالا جست و جو می کنیم.عمل جست و جو در درخت جست و جوی دودویی از مرتبه O(h)است که در آن h ارتفاع درخت است چرا که حداکثر باید به میزان عمق درخت به طرف پایین حرکت کنیم.

اضافه کردن یک عنصر به BST

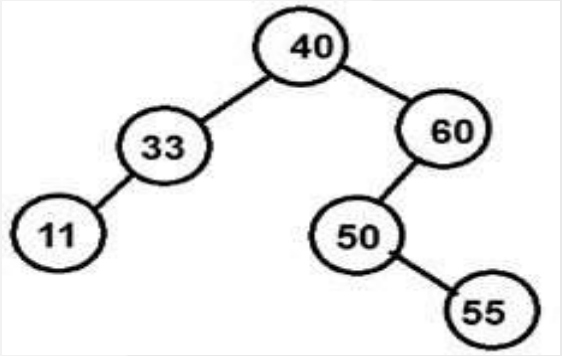
مثال:

فرض کنید می خواهیم x=20 را در درخت درج کنیم عملیات به صورت زیر خواهد بود.



مثال:

٤٠,٦٠,٥٠,٣٣,٥٥,١١



حذف یک عنصر از : BST

برای حذف یک گره دلخواه از درخت BST ابتدا باید عمل جست و جو انجام گیرد ،در این وضعیت دو حالت پیش می آید.

الف)گره مورد نظر برای حذف از درخت BST وجود ندارد در این حالت هیچ عملی انجام نمی شود.

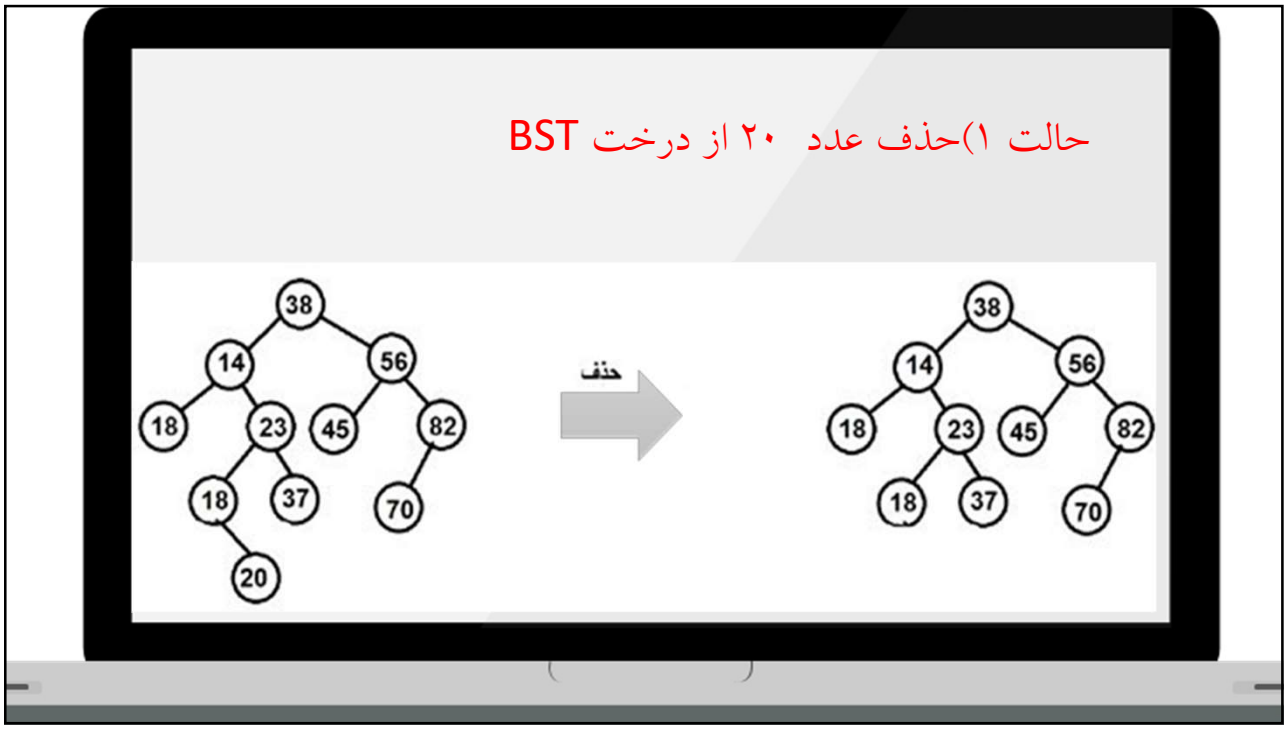
ب)گره مورد نظر برای حذف از درخت BST وجود دارد که برای این حالت ۳ وضعیت وجود خواهد داشت.

۱-عنصر x برگ است.

۲- عنصر x یک فرزند دارد.

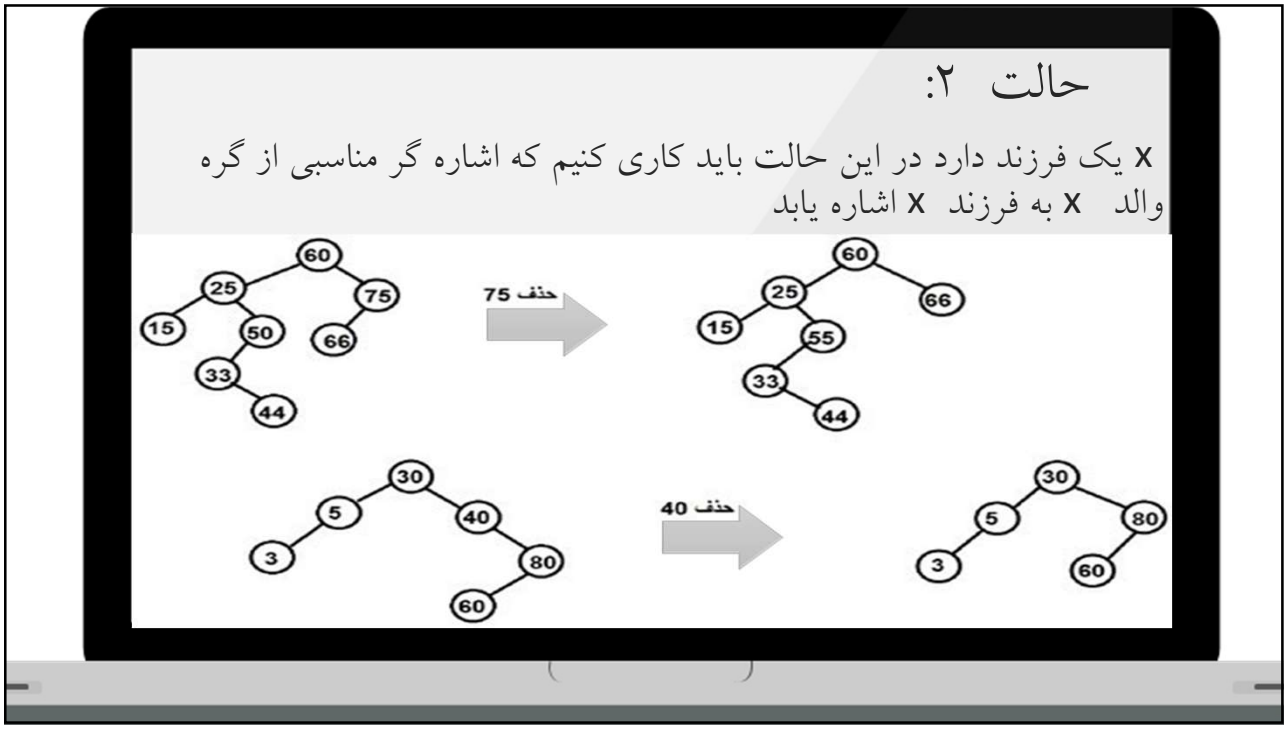
۳- عنصر x دو فرزند دارد.

حالت ۱)حذف عدد ۲۰ از درخت BST



حالت ۲:

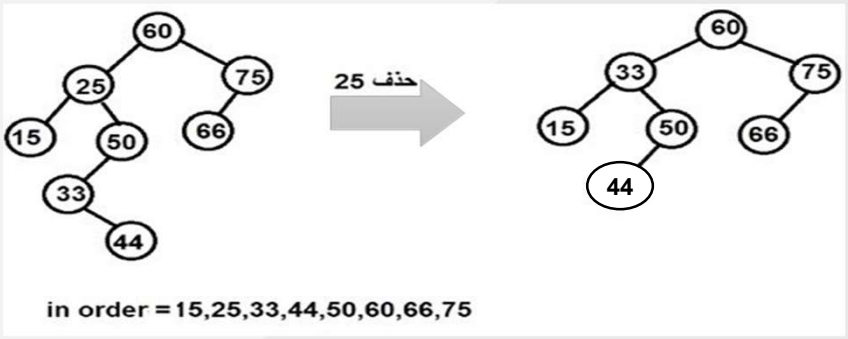
X یک فرزند دارد در این حالت باید کاری کنیم که اشاره گر مناسبی از گره والد X به فرزند X اشاره یابد



حالت ۳: x دو فرزند دارد در این حالت گره پیدا می کنیم که در پیمایش in order بعد یا قبل x قرار گیرد که باز در این حالت دو حالت پیش می آید

۱) می توانیم بزرگترین عنصر سمت چپ را جایگزین کنیم.

۲) می توانیم کوچک ترین عنصر سمت راست را جایگزین کنیم.

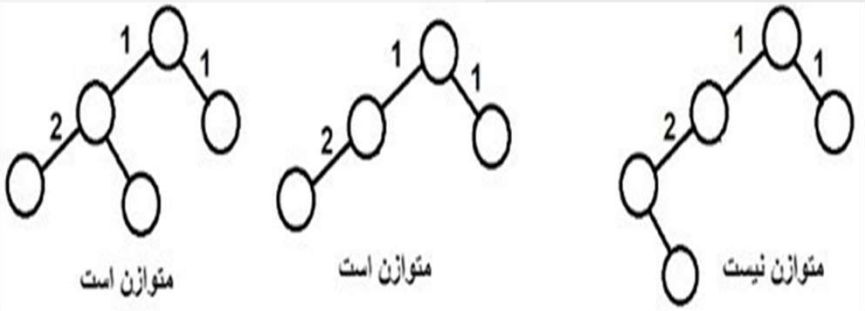


پیچیدگی الگوریتم:

- زمان درج در BST برای یک گره O(logn) که در حالت متوسط در بهترین حالت می باشد.در بدترین حالت زمان درج هر داده O(n) می باشد .
- بنابراین زمان درج در BST برای n گره در حالت متوسط و بهترین حالت از مرتبه(O(n log n و در بدترین حالت $O(n^2)$ می باشد و زمان حذف نیز مشابه زمان درج است.
- زمان درج در درخت heap برای هر عنصر(O (nlog n می باشد.

درخت با ارتفاع متوازن

درختی که در آن اختلاف در زیر درخت چپ و راست هر گره حداکثر یک باشد درخت با ارتفاع متوازن نامیده می شود.



درخت AVL (ADELSON , VELSKII , LANDIS) :

یک نوع درخت جستجوی دودویی خود متوازن‌کننده‌است(هر درخت AVL یک درخت BST بوده که ارتفاع متوازن دارد) و اولین ساختار داده‌ای از این نوع می‌باشد که اختراع شد .در یک درخت ای‌وی‌ال اختلاف ارتفاع دو زیر شاخه هر گره حداکثر برابر یک است؛ بنابراین به این درخت، درخت با ارتفاع متوازن نیز گفته می‌شود.

توجه کنید که عملیات درج، حذف و جستجو در یک درخت ای‌وی‌ال در بدترین حالت و حالت متوسط به اندازه O(logn)خواهد بود به‌طوری‌که n تعداد گره‌های درخت می‌باشد .در عملیات درج و حذف ممکن است نیاز باشد که درخت به وسیله چرخش درخت‌ها، یک یا چند بار متوازن گردد.

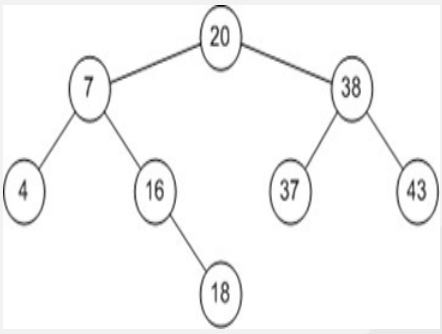
درخت AVL

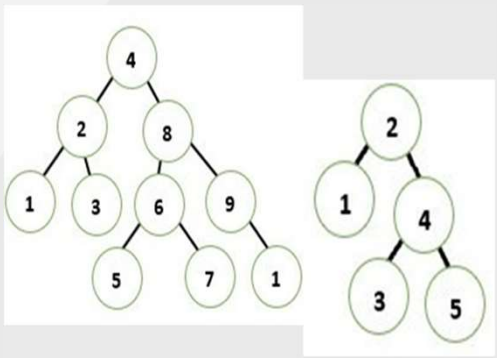
نوع	درخت
سال اختراع شدن	۱۹۶۲
اختراع شده توسط	جرجی اندلسون-ولسکی و اوگنی لاندیس

پیچیدگی زمانی بر حسب تعداد اوی بزرگ

	میانگین	بدترین حالت
فضا	O(n)	O(n)
جستجو	O(log n)	O(log n)
درج	O(log n)	
حذف	O(log n)	O(log n)

درخت های زیر AVL هستند یعنی در عین حال که BST هستند دارای ارتفاع متوازن نیز میباشند.





نکته:

اگر AVL(H) که در آن (H) برابر ارتفاع می باشد حداقل گره مورد نیاز برای ساختن یک درخت با ارتفاع متوازن (H) از رابطه زیر بدست می آید.

$$AVL(H) = AVL(H-۲) + AVL(H-۱) + ۱$$

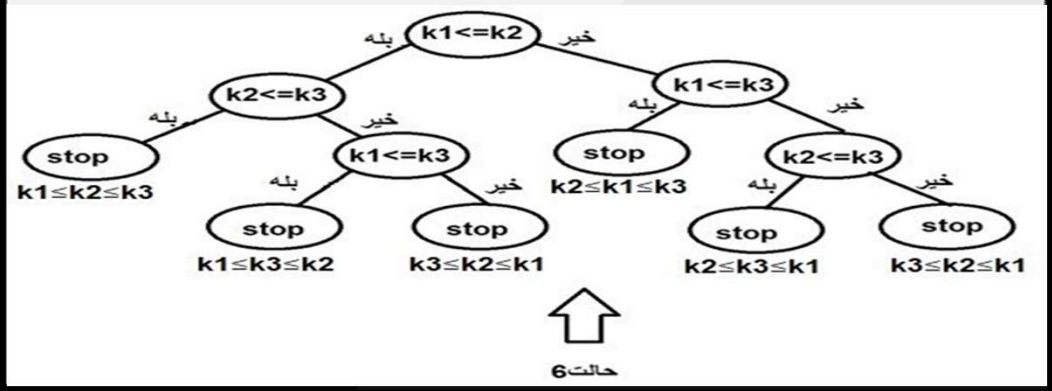
مثال:

در صورتی که حداقل گره های لازم برای ساختن ۲ درخت با ارتفاع متوازن ۳و٤ به ترتیب ٤و۷ باشد حداقل گره های لازم برای ساختن درخت با ارتفاع متوازن ۵ کدام است؟

$$AVL(۵) = AVL(۵-۲) + AVL(۵-۱) + ۱ = AVL(۳) + AVL(٤) + ۱ = ٤ + ۷ + ۱ = ۱۲$$

درخت تصمیم گیری:

۳ عدد k1 , k2 , k3 را در نظر بگیرید.منظور از درخت تصمیم تمام حالاتی است که این ۳ عدد می توانند نسبت به هم دیگر داشته باشند

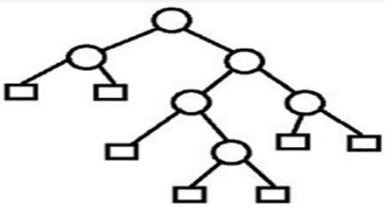


- بررسی سه عدد نسبت به هم دیگر ۳! است
- که برابر حالت های توقف می باشد به طور کلی بررسی n عضو دارای n! حالت است، که یکی از این حالات همان حالت مرتب شده است.
- ارتفاع عناصر در درخت تصمیم برای حالت مرتب عناصر O(log n) است که این بهترین‌حالت نیز می باشد.

درخت دودویی گسترش یافته توی مسیر Extended Binary Tree

درخت دودویی گسترش یافته یک درخت دودیی است که در آن هر گره ۰ یا 2 بچه دارد.گره هایی که ۰ بچه دارند گره های خارجی و گره هایی که ۲ بچه دارند گره های داخلی نامیده می شوند.

شکل زیر یک درخت EBT است که گره های داخلی را با دایره و گره های خارجی را با مربع نشان می دهیم



L = طول مسیر
N=تعداد گره ها
i= گره های داخلی
E=گره های خارجی
ne=تعداد گره های خارجی
ni=تعداد گره های داخلی
LE=طول مسیر خارجی
Li=طول مسیر داخلی

نکته:تعداد گره های خارجی ne همواره یک واحد بیشتر از تعداد گره های داخلی ni است.

ne = ni+۱ ۱+٦ = ۷

طول مسیر خارجی یا LE به صورت مجموع طول تمام مسیر هایی تعریف می شود که حاصل جمع طول تمامی مسیر ها از ریشه درخت تا یک گره خارجی است.

طول مسیر داخلی یا li به صورت مجموع طول تمام مسیر هایی تعریف میشود که حاصل طول تمامی مسیر ها از ریشه درخت تا یک گره داخلی است.

le=۲+۲+۳+٤+٤+۳+۳=۲۱

li=۰+۱+۱+۲+۲+۳=۹

نکته:اگر ni و li را داشته باشیم میتوانیم le را محاسبه کنیم.

$le=li+۲*ni$

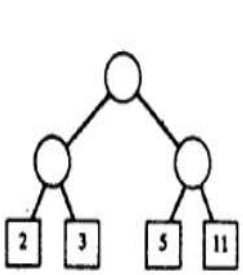
۹+۲*٦=۲۱

فرض کنید به هر گره خارجی یک وزن داده شده است طول مسیر وزن داده شده خارجی بنا به تعریف مجموع طول های مسیر وزن داده شده است و به صورت زیر تعریف می شود.

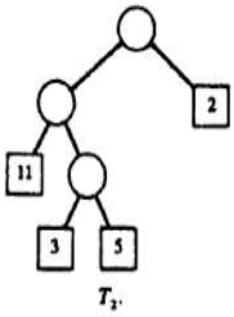
$P=W1\ L1+W2\ L2+Wn\ Ln$

که در آن w وزن گره های خارجی و L طول مسیر خارجی است و N تعداد گره های خارجی است

مثال:

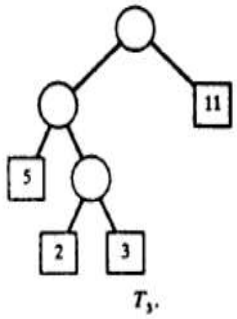


p=2*2+2*3+2*5+2*11=42



T_2.

p=2*11+3*3+3*5+1*2=48



T_3.

p=2*5+3*2+3*3+1*11=36

الگوریتم هافمن:

فرض کنید یک لیست با n وزن داده شده است در میان تمام EBT های دارای این گره خارجی و n وزن داده شده تعیین کنید یک درخت با حداقل طول مسیر وزن در اینجهت مهم است.در الگوریتم هافمن برای ایجاد درخت با حداقل طول مسیر وزن در مرحله دو درختی را که ریشه minimum دارند را انتخاب می کنیم و وزن آنها را باهم جمع می کنیم و وزن کمتر گره ها در سمت چپ قرار می گیرند بدین ترتیب هم از لحاظ حافظه و زمان جست و جو بهینه سازی و کاهش در اشغال فضا خواهیم داشت

مثال؟

The Weather Is Good.

T=2	H=2	E=3	Space=3	W=1	A=1
R=1	I=1	S=1	G=1	O=2	D=1
.=1					

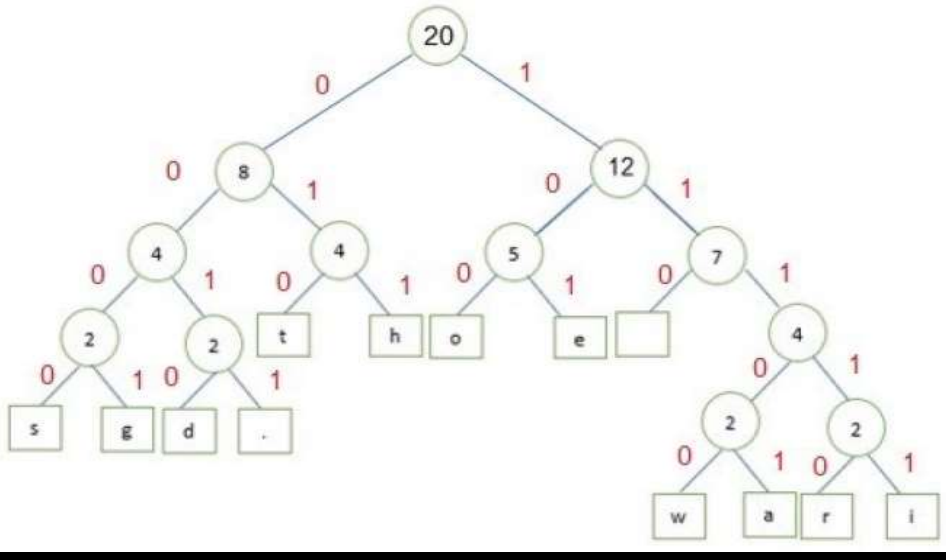
$$2+2+3+3+1+1+1+1+1+1+2+1+1=20$$

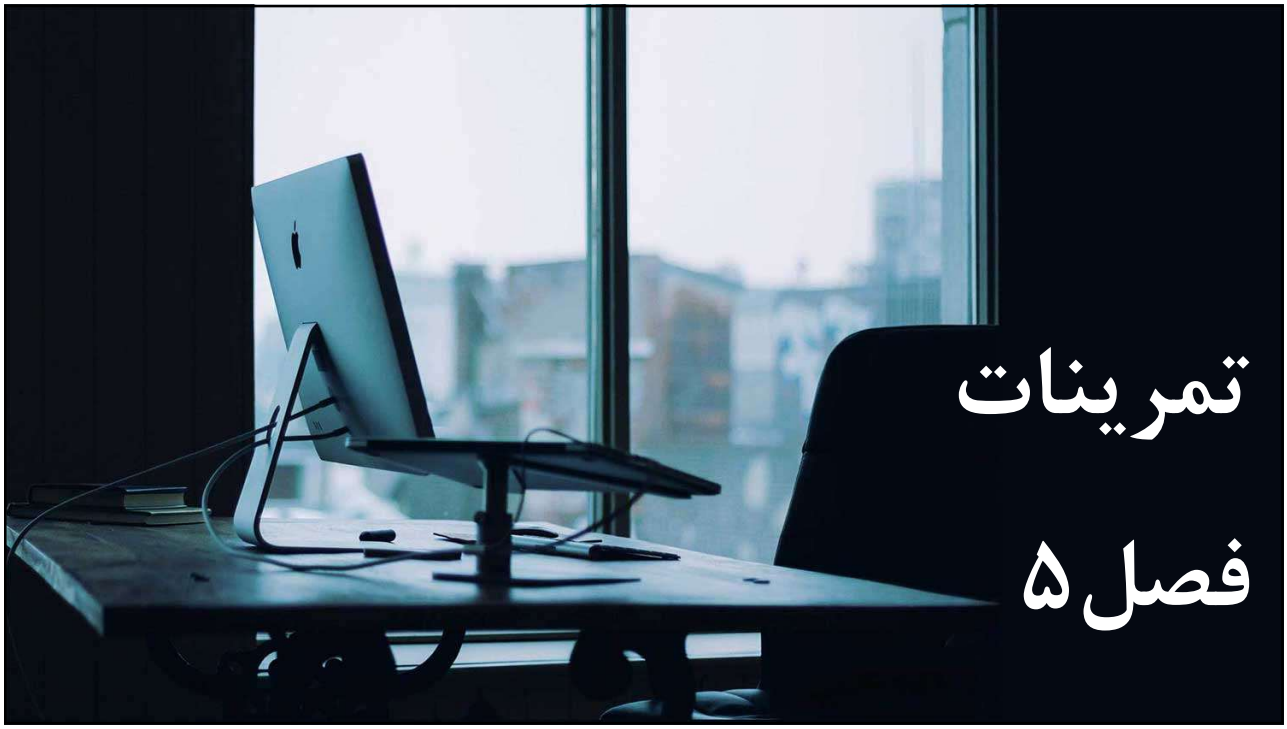
$$20*8=160\text{bit}$$

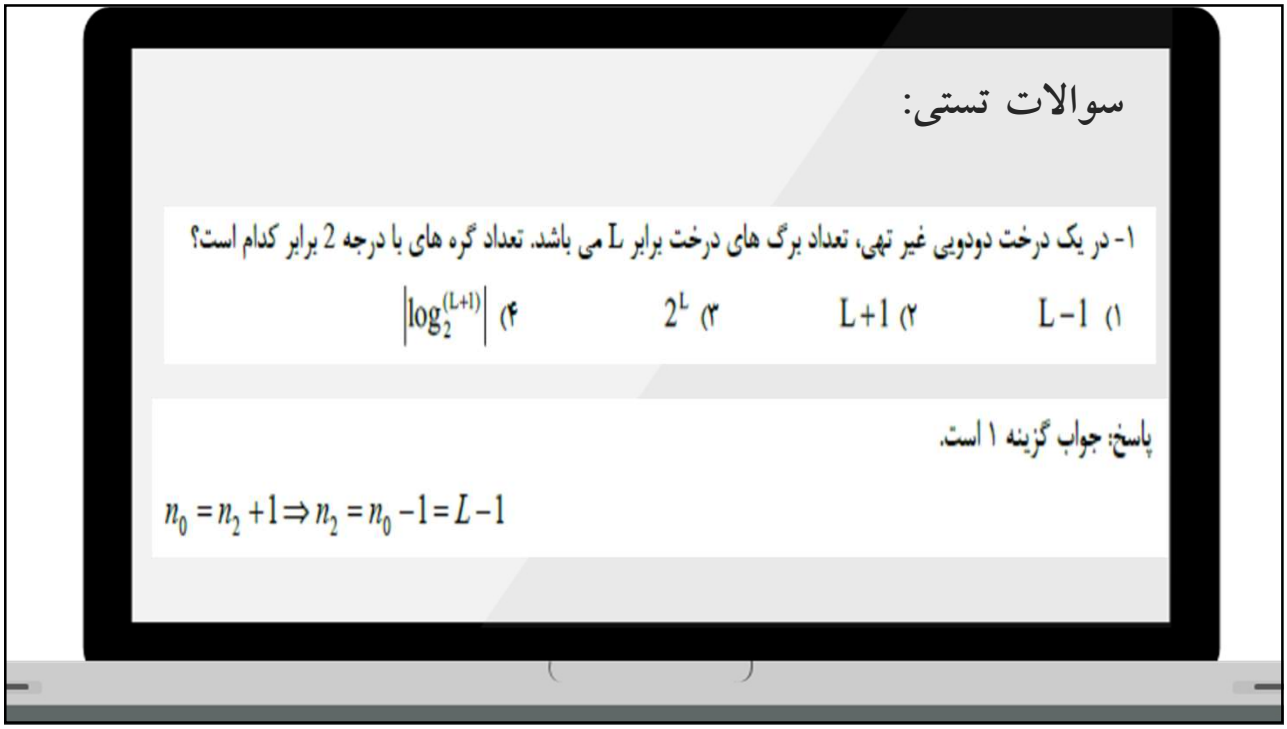
تعداد کاراکتر ها

مقدار فضای اشغالی برحسب bit

w = 1 , a = 1 , r = 1 , i = 1 , s = 1 , g = 1 , d = 1 , . = 1 , t = 2 , h = 2 , o = 2 , e = 3 , space = 3




تمرینات
فصل ۵


سوالات تستی:
۱- در یک درخت دودویی غیر تهی، تعداد برگ های درخت برابر L می باشد. تعداد گره های با درجه 2 برابر کدام است؟
۱) L-1 ۲) L+1 ۳) 2^L ۴) $\left|\log_2^{(L+1)}\right|$
پاسخ: جواب گزینه ۱ است.
$n_0 = n_2 + 1 \Rightarrow n_2 = n_0 - 1 = L - 1$

خروجی الگوریتم زیر برای یک درخت باینری معادل کدام یک از پیمایش‌های زیر می‌باشد؟

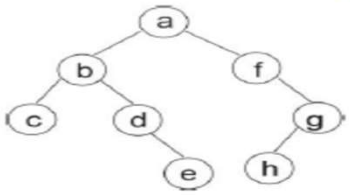
```
void  rco(ptr  p){
  if ( p !=NULL) {
       printf( p-> data );
       rco( p – > right );
       rco( p – > left );
    }
}
```

۱) inorder ۲) reverse preorder ۳) reverse postordor ۴) هیچکدام

پاسخ : جواب گزینه ۳ است.

درخت داده شده به صورت "ریشه –راست – چپ" پیمایش می شود که معکوس postorder است. پس postorder است.

۳- نمایش درخت زیر به کدام صورت درست می باشد؟



۱) (a,b(c,d),e,f(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,)))

پاسخ: جواب گزینه ۴ است.

عبارت a(b,) بیانگر این است که b فرزند چپ a است و عبارات a(,b) معرف این است که b فرزند راست a می باشد. با توجه به این موضوع عبارت (a(b(c,d(,e)),f(,g(h,))) معادل درخت داده شده است.

۴- با ۴ گره چند درخت دودویی متفاوت (از لحاظ فرم درخت و بدون توجه به داده های آن) می توان ساخت؟

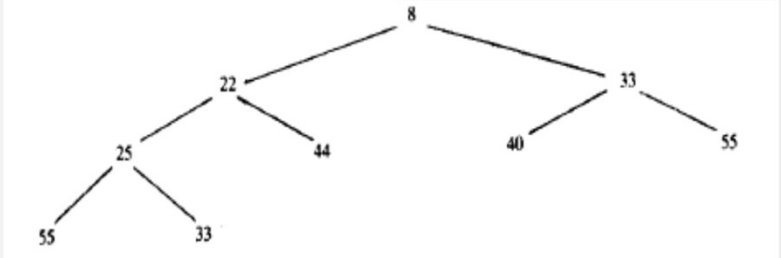
۱) 5 ۲) 42 ۳) 18 ۴) 14

پاسخ: جواب گزینه ۴ است.

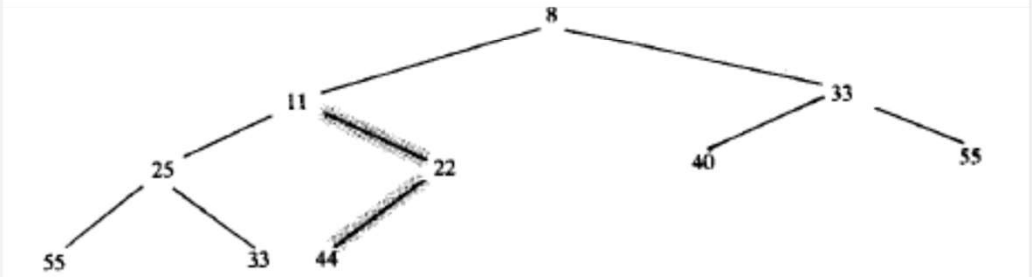
$$\frac{\binom{8}{4}}{5} = \frac{\frac{8!}{4!\times 4!}}{5} = \frac{\frac{8\times 7\times 6\times 5\times 4!}{4!\times 4!}}{5} = \frac{\frac{8\times 7\times 6\times 5}{4\times 3\times 2\times 1}}{5} = \frac{70}{5} = 14$$

سوالات تشریحی :

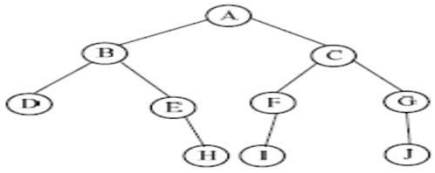
H : MinHeap شکل (الف) را درنظر بگیرید. (H یک MinHeap است چون عنصرهای کوچکتر به عوض عنصرهای بزرگتر، بالای Heap هستند.) وضعیت Heap را پس از اضافه‌شدن ITEM = 11 در H توضیح دهید.



حل : نخست **ITEM** را به عنوان گره بعدی یعنی به عنوان بچهٔ چپ گرهٔ **44** در درخت کامل اضافه کنید. آنگاه بطور مکرر **ITEM** را با پدرش مقایسه کنید. چون **44>11**، جای **11** و **44** را عوض کنید. از آنجا که **22>11**، جای **11** و **22** را عوض کنید. چون **8>11**، **ITEM = 11** مکان مربوطه‌اش را در **Heap** پیدا کرده است. شکل ۵۳-۷ (ب) **Heap** تازه شده **H** را نشان می‌دهد. یالهای سایه‌زده شده بیانگر مسیر **ITEM** به طرف بالای درخت است.



مثال ۲-۶: سه روش پیمایش را در نظر گرفته و درخت زیر را با سه روش پیمایش نمائید.



حل: همان‌طور که می‌دانید در روش Postorder اول زیر درخت چپ سپس زیر درخت راست و بعد ریشه ملاقات می‌شود. بنابراین خواهیم داشت:

DHEBIFJGCA

در روش Inorder اول زیر درخت چپ بعد ریشه و بعد زیر درخت راست ملاقات می‌شوند لذا خواهیم داشت:

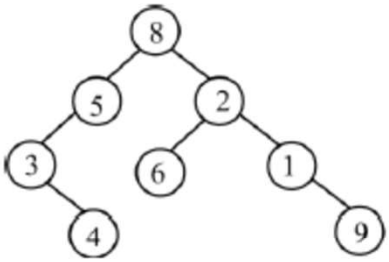
DBEHAIFCJG

در روش preorder همان‌طور که می‌دانید اول ریشه بعد زیر درخت چپ و بعد زیردرخت راست ملاقات می‌شود. بنابراین:

ABDEHCFIGJ

خروجی پیمایش به روش preorder خواهد بود.

حاصل پیمایش های معمول درخت زیر را بدست آورید.

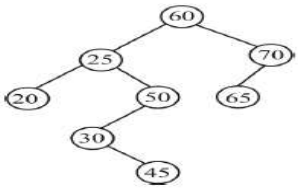


inorder : 34586219

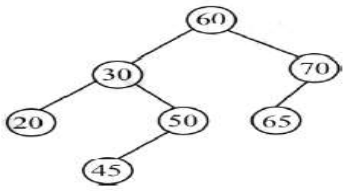
preorder : 85342619

postorder : 43569128

گره با داده 25 را از درخت زیر حذف کنید.

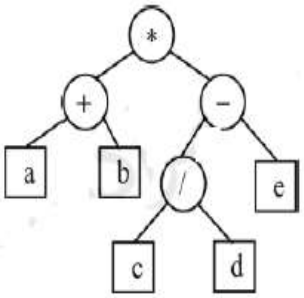


سخ: برای حذف گره با داده 25 ، می توان گره 20 یا 30 را جایگزین کرد. (در این مثال گره 30)



کر: در واقع گره 20 یا 30 ، گرههای قبل و بعد از گره 25 در پیمایش میانوندی درخت می باشند.

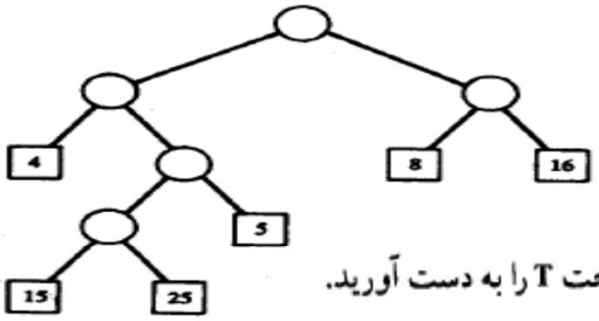
عبارت جبری متناظر با 2-TREE زیر کدام است؟



رخت داده شده برابر است با :

E =(a+b)*((c/d)–e)

درخت T وزن داده شدهٔ شکل ۵۷ـ ۷ را درنظر بگیرید.



P طول مسیر وزن داده شدهٔ درخت T را به دست آورید.

حل : هر وزن W_i را در طول L_i مسیر از ریشهٔ T تا گره‌ای که دارای وزن است ضرب کنید و آنگاه تمام این حاصلضرب‌ها را با هم جمع کنید تا P بدست آید. بنابراین :

$$P = 4.2 + 15.4 + 25.4 + 5.3 + 8.2 + 16.2 = 8 + 60 + 100 + 15 + 16 + 32 = 231$$

پرسش های فصل پنجم

1 فرض کنید دنبالهٔ زیرلیست گره‌های یک درخت دودویی T به ترتیب در پیمایش PreOrder و InOrder باشند :

Preorder: G, B, Q, A, C, K, F, P, D, E, R, H
Inorder: Q, B, K, C, F, A, G, P, E, D, H, R

نمودار درخت را رسم کنید.

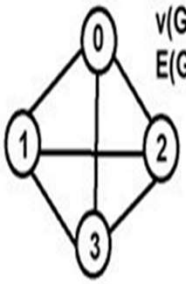
2 فرض کنید هشت عدد زیر به ترتیب به یک درخت جستجوی دودویی خالی T اضافه شده‌اند.

50, 33, 44, 22, 77, 35, 60, 40

درخت را رسم کنید.

- هر گراف (G) شـامل دو مجموعه V , E اسـت.
- V مجموعه ای محدود از رئوس و E مجموعه ای محدود از لبه هاست.
- E(g) و V(g) مجموعه رئوس و لبه های گراف را نشـان می دهد هر گراف به صـورت G(V,E) نمایش داده می شـــود.
- در گراف دو وضـــعیت وجود دارد، وضـــعیت جهت دار و غیر جهت دار که در وضـــعیت غیر جهت دار رئوس زوج مرتب نیســـتند بنابراین زوج های (v0,v1) , (v1,v0) یکســـانند اما در گراف جهت دار این دو زوج دو لبه متفاوت را نمایش می دهد.
- گراف حداقل یک راس دارد و نمیتواند کاملا تهی باشد.

گراف غیر جهت



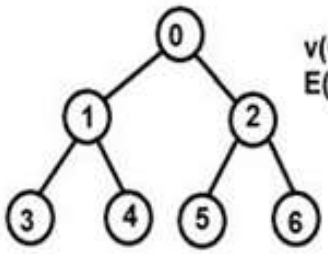
v(G1)={0,1,2,3}
E(G1)={(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)}

شکلG1

گراف جهت دار

v(G2)={0,1,2}
E(G2)={(0,1),(1,0),(1,2)}

شکلG2



شکلG3

v(G3)={0,1,2,3,4,5,6}
E(G3)={(0,1),(0,2),(1,3),(1,4),(2,5),(2,6)}

نکته : درخت را می توان حالت خاص از گراف در نظر گرفت که در آن حلقه وجود ندارد.

گراف تهی یا پوچ

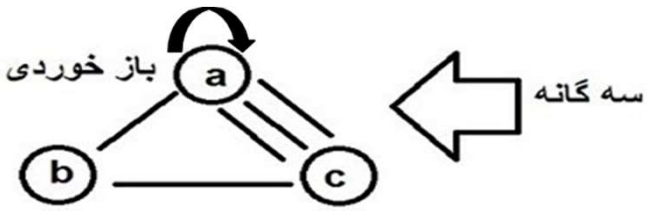
گراف که فقط شـــامل گره های منفرد اســـت یا به عبارت دیگر مجموعه یال آن تهی اســـت .

گراف چندگانه

گرافی را که دارای لبه های چندگانه هست را گراف چندگانه می گویند.

گراف خود حلقه ای یا بازخوردی

گرافی که در آن لبه یا یالی از یک راس به خود آن راس وجود داشته باشد را می گویند.



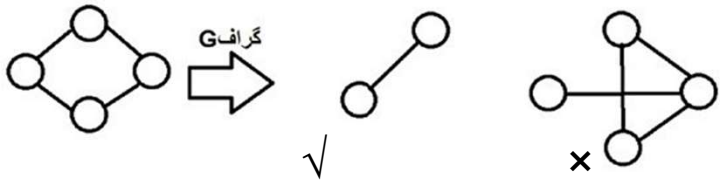
گراف ساده:گرافی است که فاقد خود حلقه و لبه های چندگانه است .

گراف کامل:گرافی که دارای حداکثر لبه باشد یعنی بین هر گره زوج،لبه ی مستقیمی وجود داشته باشد .

نکته : برای یک گراف بدون جهت با n راس حداکثر تعداد لبه ها برابر با $\frac{n(n-1)}{2}$ است.

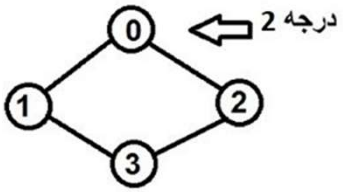
نکته: برای یک گراف جهت دار با n راس حداکثر تعداد لبه ها برابر با n(n-1) است.

تعریف G' گراف: 'G زیر گراف G است اگر $\acute{E}(G) \subseteq E(G), V(\acute{G}) \subseteq V(G)$ باشد

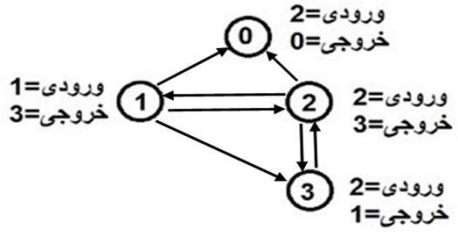


تعریف طول مسیر :تعداد لبه های موجود در مسیر را طول مسیر می گویند.

درجه ۱ راس :در یک گراف بدون جهت تعداد لبه ها متقاطع با آن راس است.



در گراف جهت دار هر راس دارای درجه ورودی و خروجی است .تعداد یال هایی که به راس i وارد می شود، درجه ورودی و تعداد یال هایی که از آن خارج می شود،درجه خروجی آن است

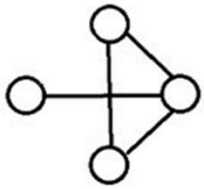


نکته:اگر درجه یک راس عددی فرد باشد آن راس را راس فرد و اگر زوج باشد آن را زوج گویند.

درجه گراف:درجه گراف برابر با بزرگترین درجه گره های آن است

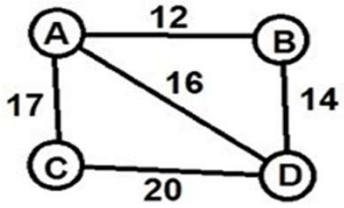
نکته:اگر در گراف (G) با n راس با شماره های ۰ تا (n-1) اگر Di درجه راس i باشد،آنگاه تعداد لبه ها یعنیE از فرمول زیر بدست می آید

$$E = \frac{1}{2}\sum_{i=0}^{n-1} di$$

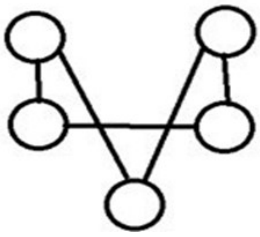


$\frac{1}{2}(2+1+3+2)=\frac{1}{2}(8)=4$

تعریف گراف شماره دار یا برچسب دار:گراف (G) را شماره دار گویند اگر اطلاعاتی به یال های آن نسبت داده شده باشد



گراف منظم:اگر در گرافی درجه تمام راس ها برابر ۲ باشد آن گراف را منظم می گویند



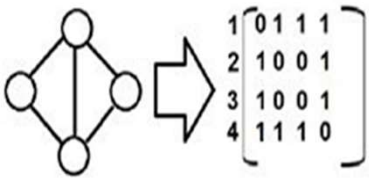
نمایش گراف : دو روش برای نمایش گراف وجود دارد

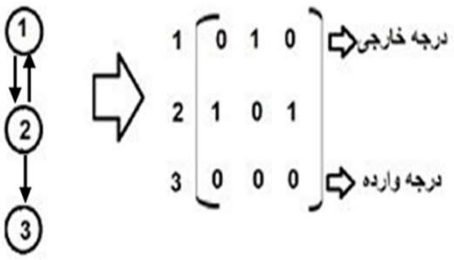
۱ ماتریس مجاورت(همسایگی)

۲ لیست مجاورتی

ماتریس مجاورتی:

فرض کنید G(V,E) یک گراف باشد ماتریس مجاورتی گراف (G) یک آرایه دو بعدی n*n خواهد بود که اگر vi و vg در EG باشد عنصر خانه i,g برابر با 1 خواهد بود و در غیر این صورت ۰ در نظر گرفته می شود





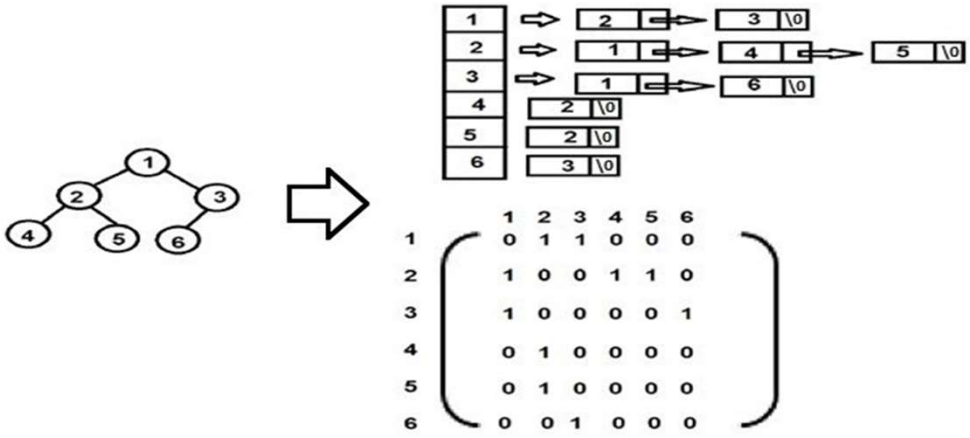
نکته:برای گراف بدون جهت درجه هر راس مانند i مجموعه عناصر سطری آن است و برای گراف جهت دار مجموعه سطری درجه خارجی و مجموعه ستونی درجه وارده خواهد بود.

تعداد یک ها در ماتریس مجاورتی بدون جهت دو برابر تعداد یال ها و در گراف جهت دار برابر تعداد یال هاست.

لیست مجاورتی:

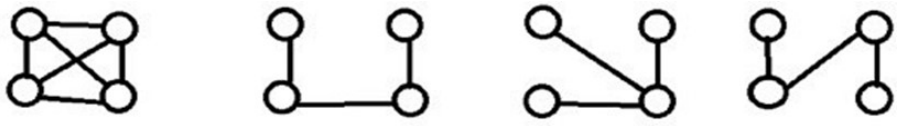
در این روش برای هر راس از گراف G یک لیست وجود دارد هر گره حداقل دو فیلد دارد.(راس و اتصال). در هر لیست مشخص مانند i گره های لیست حاوی دو راس مجاور از راس i می باشد

هر لیســـت یک گره هد دارد که به ترتیب شـــماره گذاری شـــده اند و این امر دســـتیابی ســـریع به لیســـت های مجاورتی برای راس خواص را به آســـانی امکان پذیر می کند.نمایش لیســـت مجاورتی گراف زیر بدین صـورت خواهد بود



درخت پوشا یاSpanning Tree):

- درختی که تعدادی از لبه ها یا همان(یال ها) ولی تمام راس های (G) را در بر دارد که **درخت پوشـــا** نامیده می شـــود .
- درخت پوشای یک گراف با n گره ، دارای حداقل n-1 یال است .
- یک گراف و سه درخت پوشای آن را در زیر مشاهده می کنید

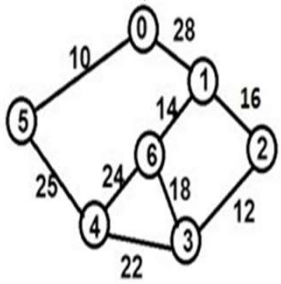


درخت پوشا(Minimum):

هزینه یک درخت پوشـــای یک گراف دارای وزن مجموعه هزینه ها یعنی وزن های لبه ها در درخت پوشـا می باشـد.

درخت پوشـای Minimum(MST) درختی اسـت که دارای کم ترین هزینه باشـد برای بدسـت آوردن درخت پوشـایMinimum،الگوریتمراشـال کروسـکال والگوریتم پریم را شـرح می دهیم.

.برای درخت پوشایMinimum باید سه شرط زیر را در نظر بگیریم.



۱ فقط از لبه های داخل گراف استفاده کنیم

۲ باید دقیقا از n-1 از لبه استفاده کنیم.(تعداد گره=n)

۳ نباید از لبه هایی که ایجاد حلقه می کنند استفاده کنیم.

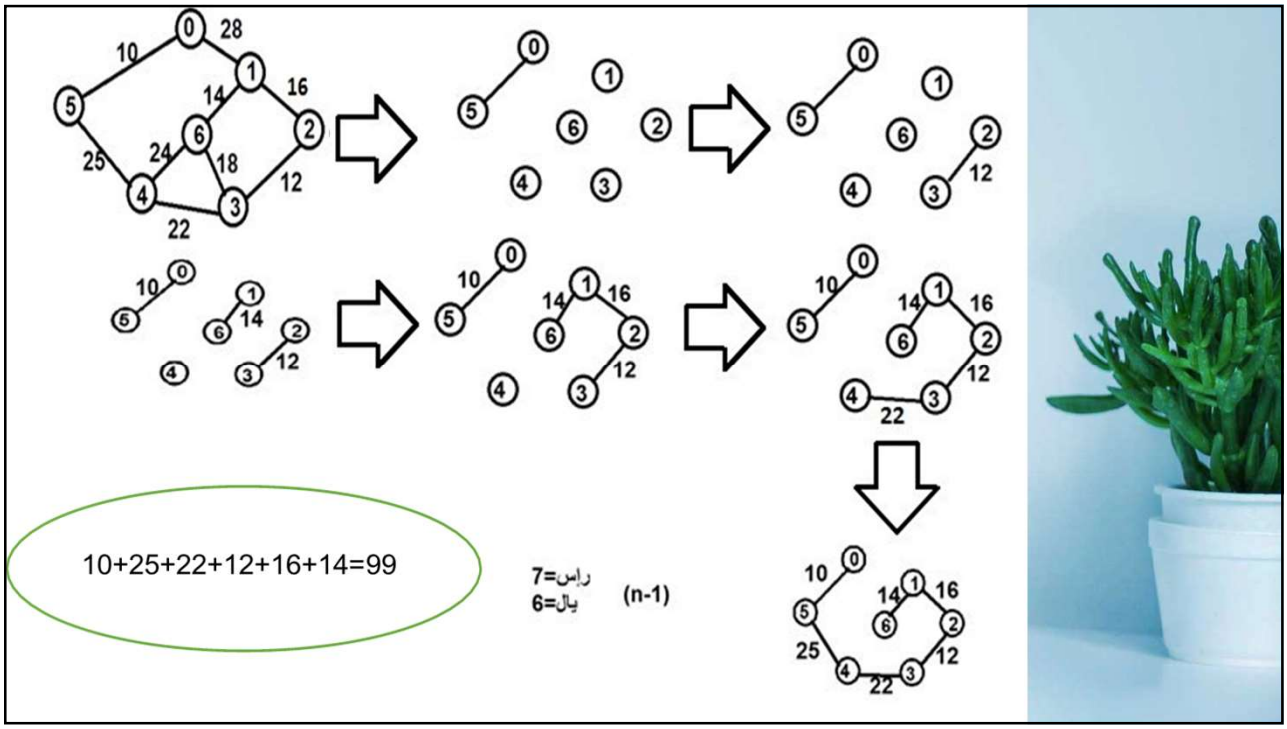
الگوریتم راشال(کروسکال):

در این روش درخت پوشـــا با کمترین هزینه لبه به لبه ســـاخته می شـــود.

۱-لبه های مورد اســـتفاده در گراف G به ترتیب صعودی وزن ها مرتب می شود.

۲-درخت T با تمام راس های گراف G بدون یال می کشیم.

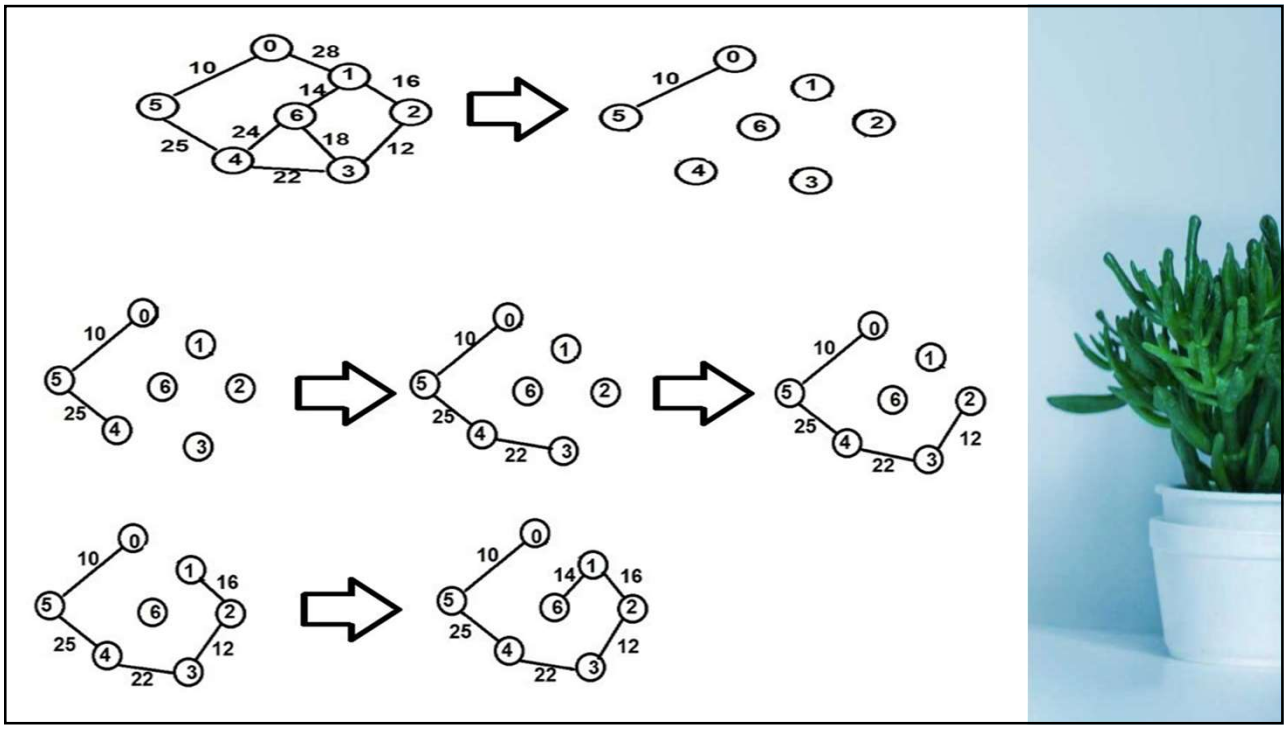
۳-به درخت T یک یال با حداقل وزن از گراف G اضافه می کنیم به طوری که در T حلقه ایجاد نشود .این کار را n-1 مرتبه تکرار می کنیم.



الگوریتم پریم(prim):

الگوریتم پریم همانند راشال(کروسکال)است با این تفاوت که در هر مرحله مجموعه لبه های انتخاب شده یک درخت را تشکیل می دهد به عبارت دیگر می توان گفت در این الگوریتم ابتدا گره ای به دلخواه انتخاب می شود سپس از بین یال های متصل به آن یال با کمترین وزن انتخاب می شود به گونه ای که حلقه ایجاد نکند.در هر مرحله یالی انتخاب می شود که حتما یکی از دو سر آن جزء مسیر جواب بوده و وزن حداقل داشته باشد

مرتبه اجرایی پریم برابر $o(n^2)$ و مرتبه اجرایی کراسکال برابر $o(e \times \log e)$ می باشد.


0 28 1 16 2 12 3 22 4 25 5 10 6 14 24 18
0 1 2 3 4 5 6 10
0 1 2 3 4 5 6 10 25
0 1 2 3 4 5 6 10 25 22
0 1 2 3 4 5 6 10 25 22 12
0 1 2 3 4 5 6 10 16 25 12 22
0 1 2 3 4 5 6 10 14 16 25 12 22


تمرینات
فصل ۶

سوالات تستی:

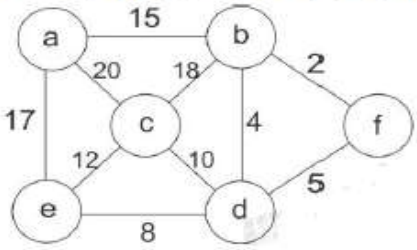
۱- در نمایش یالی (Adjacency Multilist) یک گراف G=(V,E) با $|E| = m, |V| = n$ ، چند Linked list و چه تعداد node وجود دارد (بدون در نظر گرفتن head node)؟ (گزینه ها از چپ به راست)

۱) $|V|+|E|, |V|$ ۲) $|V|, |E|$ ۳) $|E|, |E|$ ۴) $|V|, |V|+|E|$

حل: جواب گزینه ۲ است.

در نمایش یالی گراف G=(V,E) ، به تعداد گره ها یعنی $|V|$، لیست پیوندی و به تعداد دو برابر یالها که از مرتبه $|E|$ می باشد، node وجود دارد.

۲- ترتیب انتخاب یال ها در الگوریتم Prim روی گراف زیر با شروع از راس a کدام مورد است؟



۱) (b,f),(b,d),(e,d),(c,d),(a,b)
۲) (a,b),(b,f),(f,d),(d,e),(e,c)
۳) (a,b),(b,f),(b,d),(d,e),(d,c)
۴) هیچکدام

حل: جواب گزینه ۳ است.

با شروع از گره a، در مرحله اول یال (a,b) با وزن 15 انتخاب می شود. در مرحله دوم یال (b,f) با وزن 2، در مرحله سوم یال (b,d) با وزن 4، در مرحله چهارم یال (e,d) با وزن 8 و در مرحله پنجم یال (d,c) با وزن 10 انتخاب خواهد شد.

۳- یک گراف بسیار متصل شامل n گره مفروض است. می خواهیم با استفاده از الگوریتم kruskal کوچکترین درخت پوشای این گراف را بدست آوریم. در این صورت پیچیدگی زمانی این الگوریتم برابر است با:

۱) $o(n^2 \log n)$ ۲) $o(n \log n)$ ۳) $o(n^2)$ ۴) $o(\frac{n^2}{2})$

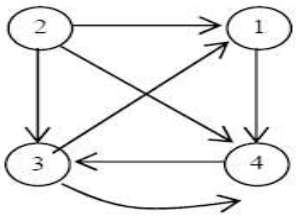
حل: گزینه ۱ جواب است.

مرتبه اجرایی الگوریتم کراسکال برابر $\theta(e \log e)$ می باشد که در یک گراف کامل که تعداد یالها برابر $\frac{n(n-1)}{2}$ از مرتبه n^2 می باشد، خواهیم داشت:

$$\theta(e \log e) = \theta(n^2 \log n^2) = \theta(2n^2 \log n) = \theta(n^2 \log n)$$

سوالات تشریحی :

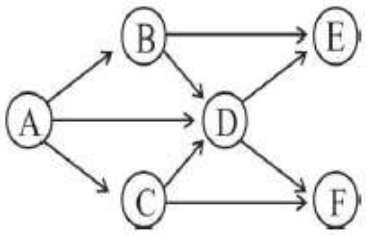
ماتریس مجاورتی گراف جهت‌دار G، شکل را در نظر بگیرید:



ماتریس مجاورتی گراف جهت‌دار G چنین است:

$$A = \begin{pmatrix} ۰ & ۰ & ۰ & ۱ \\ ۱ & ۰ & ۱ & ۱ \\ ۱ & ۰ & ۰ & ۱ \\ ۰ & ۰ & ۱ & ۰ \end{pmatrix}$$

چند پیمایش BFS و DFS برای گراف زیر مشخص کنید.



پاسخ:

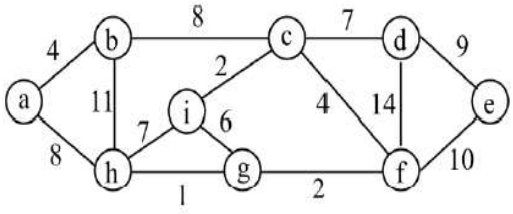
چند پیمایش BFS به صورت زیر است:

ABCDEF , ADBCEF , ADBCFE

و چند پیمایش DFS :

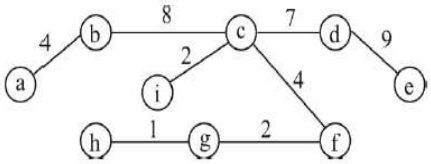
ABEDFC , ABDFEC , ADEFBC

وزن درخت پوشای مینیمم گراف در صورت استفاده از الگوریتم کراسکال چه می باشد؟



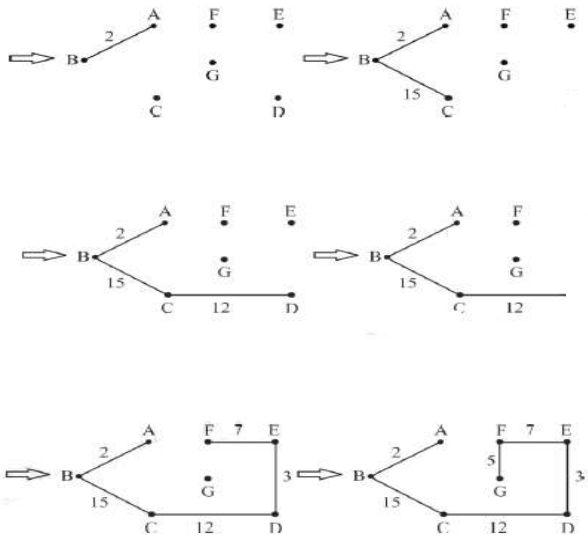
حل:

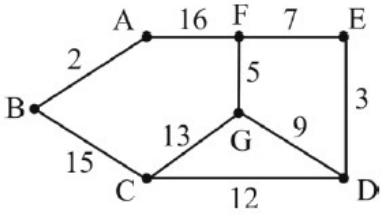
ترتیب اضافه شدن یالها در روش کراسکال برابر است با: 1,2,2,4,4,7,8,9 ، بنابراین وزن یال ها در درخت پوشای مینیمم برابر 37 می باشد.



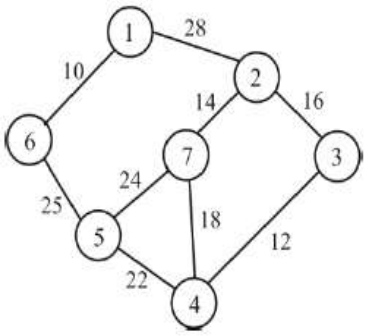
به کمک الگوریتم پریم، درخت پوشای حداقل گراف زیر را بدست آورید.(با شروع از گره a)

پاسخ:





ترتیب اضافه شدن یالها را در صورت استفاده از الگوریتم پریم و کراسکال مشخص نمایید.



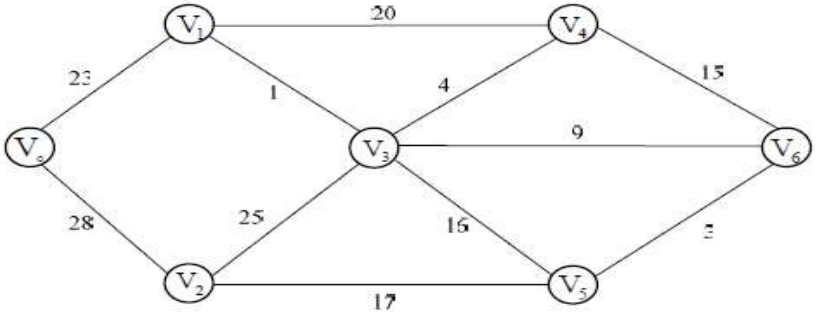
حل:

ترتیب اضافه شدن یالها در روش کراسکال برابر است با: 10,12,14,16,22,25

و در روش پریم برابر است با : 10,25,22,12,16,14

پرسش فصل ششم

گراف وزن‌دار زیر را در نظر بگیرید:



با استفاده از الگوریتم های راشال و پریم درخت پوشای کمینه ، ترتیب اضافه شدن یالهای آن و وزن درخت به دست آمده در هر روش را بدست آورید

مقدمه

مرتب کردن و جستجوی اطلاعات از عملیات اساسی و اصلی در علم کامپیوتر است. مرتب کردن عبارت است از، عمل تجدید آرایش داده‌ها با یک ترتیب مشخص، مثلا برای داده‌های عددی به ترتیب صعودی یا نزولی اعداد یا برای داده های کاراکتری ترتیب الفبایی آنها مرتب‌سازی صورت می‌گیرد. به همین ترتیب، جستجو کردن عبارت است از عمل پیدا کردن محل یک عنصر داده شده در بین مجموعه ای از عناصر می‌باشد.

مرتب کردن و جستجوی اطلاعات، اغلب روی یک فایل از رکوردها به کار می‌رود، از این رو لازم است چند اصطلاح استاندارد را یادآوری کنیم. هر رکورد در یک فایل می‌تواند چند فیلد داشته باشد اما فیلد ویژه‌ای وجود دارد که مقدارهای آن به‌طور منحصر به فردی، رکوردهای داخل فایل را معین می‌کند. چنین فیلدی یک کلید اولیه یا اصلی نامیده می شود، مرتب‌کردن فایل معمولا به مرتب کردن نسبت به کلید اولیه خاصی گفته می شود و جستجوی اطلاعات در فایل به جستجوی رکورد با مقدار کلیدهای معین گفته می‌شود.

نکته :

اگر عمل مرتب سازی بر روی رکوردهای موجود در حافظه انجام شود، مرتب‌سازی را مرتب‌سازی داخلی(internal) می‌نامند و اگر بر روی رکوردهای موجود در حافظه جانبی (مانند دیسک‌ها)صورت گیرد، مرتب سازی را مرتب‌سازی خارجی (External) می‌نامند. در این فصل بیشتر مرتب‌سازی های داخلی مورد بحث و بررسی قرار خواهد گرفت.

الگوریتم های مرتب سازی را از نظر نحوه مرتب سازی داده ها می توان به صورت زیر دسته بندی کرد:

۱- مرتب سازی مقایسه ای

در مرتب سازی مقایسه ای، اطلاعات دیگری از داده های ورودی وجود ندارد و فقط با مقایسه بین کلیدهای عناصر، ترتیب نسبی آنها پیدا می شود.

۲- مرتب سازی غیر مقایسه ای(خطی)

در مرتب سازی غیر مقایسه ای، بدون مقایسه کلیدهای عناصر با هم، عمل مرتب سازی انجام می شود. این الگوریتم ها با استفاده از اطلاعاتی که از قبل در خصوص نوع کلیدها موجود است و در شرایط خاص، از روش های سریعی استفاده می کند که در آنها کلیدهای عناصر با هم مقایسه نمی شوند.

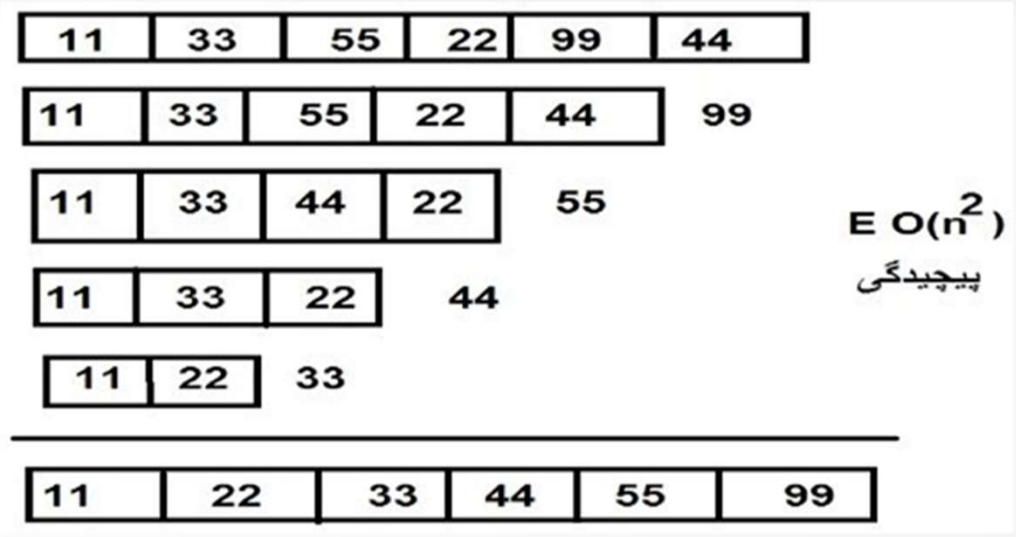
در این فصل مرتب سازی های مقایسه ای را بررسی می کنیم. از مرتب سازی های غیر مقایسه ای مانند مرتب سازی "مبنایی، شمارشی و سطلی"، مرتب سازی مبنایی را بررسی می کنیم .

۱-مرتب سازی انتخابی selection sort)
۲-مرتب سازی حبابی bubble sort)
۳-مرتب سازی درجی insertion sort)
٤-مرتب سازی جا به جایی exchange sort)
٥-مرتب سازی سریع quick sort)
٦-مرتب سازی هرمی heap sort)
۷-مرتب سازی ادغامی یا ترکیبی merge sort)
۸- مرتب سازی درختی tree sort)
۹-مرتب سازی مبنا radius sort)
۱۰-مرتب سازی پوسته shell sort)

۱-مرتب سازی انتخابی:

در این الگوریتم در هر مرحله پیمایش آرایه،محل درست یک عنصر(بزرگترین و کوچکترین عنصر)پیدا شده و سپس با یک جا به جایی در محل درست خود قرار می گیرد.
کد الگوریتم به صورت زیر می باشد:

```
for(i=۰ ; i<n ; i++)
{
max=x[i]; index=۱;
for(j=۰ ; j<n ; j++)
{
if(x[j] > max)
{
max=x[j]; index=j;
 x[index]=x[i]; x[i]=max;
}
```



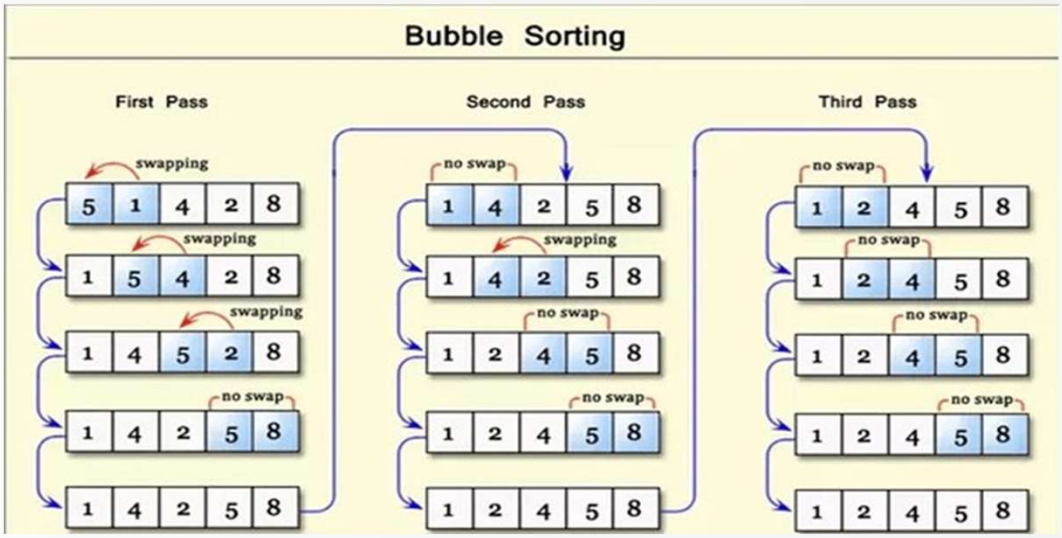
۲-مرتب سازی حبابی:

در مرتب سـازی حبابی باید چندین بار آرایه را پیمایش کرد و هر بار عنصـری را با عنصـر بعدی خودش مقایسـه نمود در صورتی که عنصر اول از عنصر دوم بزرگتر باشد در مرتب سازی صعودی جای آنها را عوض می کنیم.

تعداد مقایسه ها در روش مرتب سازی حبابی $\frac{n(n-1)}{2}$ می باشد که در بدترین حالت نیز به همین تعداد تعویض انجام می شود.

کد الگوریتم حبابی:

```
 for(i=۰ ; i<n ; i++)
{
for(j=i+۱ ; j<n ; j++)
{
if(x[i] > x[j])
{
temp=x[i];
 x[i]=x[j];
x[j]=temp
}
}
```



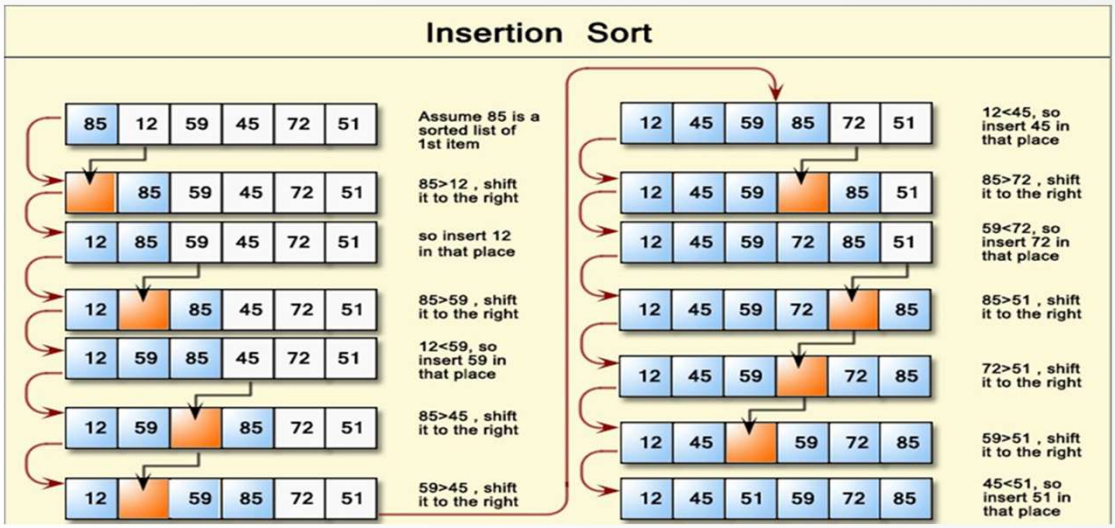
۳-مرتب سازی درجی:

در مرتب سازی درجی عناصر به ترتیب خوانده شده و هر عنصر در مکان صحیح در زیر آرایه از قبل مرتب شده درج میشود.

- در مرتب سازی درجی، پیچیدگی زمانی تعداد انتساب های رکوردها، در بدترین حالت برابر $o(n^2)$ و در حالت میانگین $o(n)$ می باشد
- مرتب سازی درجی، برای یک آرایه مرتب، بهترین عملکرد و برای یک آرایه مرتب معکوس، بدترین عملکرد را دارد.
- مرتب سازی درجی برای $n<=20$، سریع‌ترین روش مرتب‌سازی است.
- اگر n بزرگ باشد و رکوردها هم بزرگ باشند (زمان انتساب آنها چشمگیر باشد)، مرتب سازی انتخابی باید بهتر از مرتب سازی درجی عمل کند.
- در مرتب سازی درجی، یک آرایه n عنصری را می توان با $n-1$ عمل درج مرتب کرد

تکه کد الگوریتم به صورت زیر می باشد

```
insertion sort(x[],h)
{
for(i=2 ; i<n ; i++)
{ Y=x[i];
j=i-۱;
while(j>۰ && Y<x[j])
{
x[j+۱] = x[j]; j=j-۱;
}
x[j+۱]=Y;
```

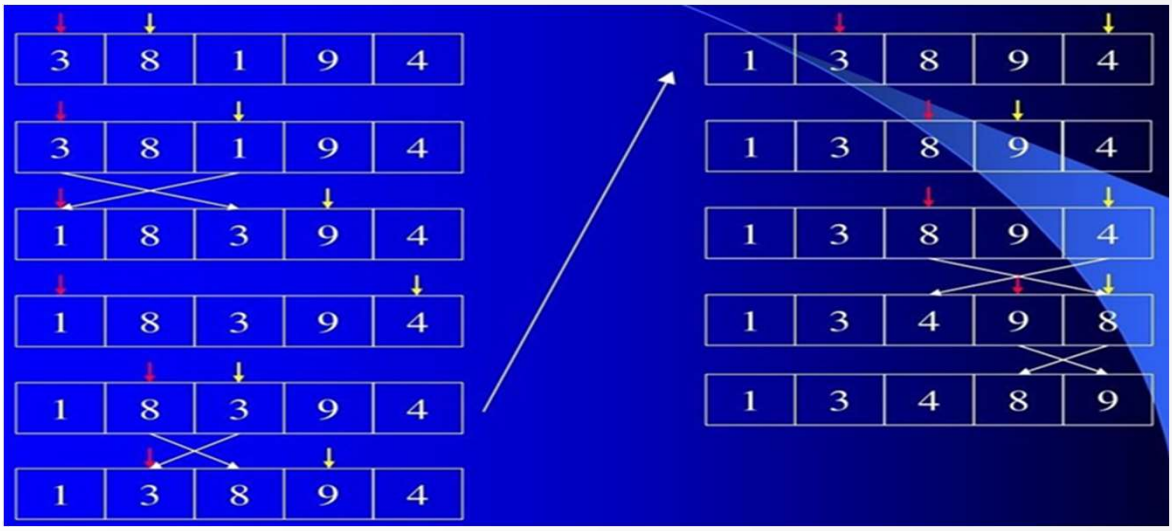


٤-مرتب سازی جا به جایی(تعویض):

در این الگوریتم ابتدا خانه اول با خانه های دوم تا اخر مقایسه می شود و از هر کدام که بزرگتر بود با آن جا به جا می شود بدین ترتیب که در انتهای گذر اول کوچکترین عدد در خانه اول قرار گرفته است سپس خانه دوم با خانه های سوم تا nمقایسه می شود و عملیات مذکور انجام میگیرد و این روند تا مرحله(n-۱) ادامه پیدا می کند.

این الگوریتم با وجود سادگی، به دلیل پیچیدگی زمانی(O(n^2 ، برای آرایه‌های بزرگ کارایی مناسبی ندارد، اما به دلیل تعداد کم جابه‌جایی‌ها، در مواردی که جابه‌جایی داده‌ها هزینه‌بر است، می‌تواند مفید باشد.

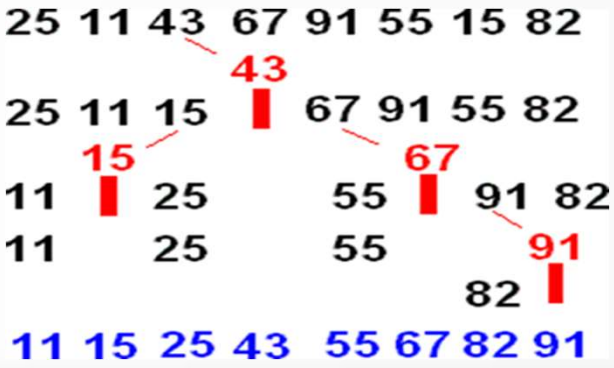
```
exchange sort (a[n])
{
for(i=١ ; i<=n-١ ; i++)
 if(a[i] > a[j])
swap(a[i] , a[j])
}
```



۵-مرتب سازی سریع:

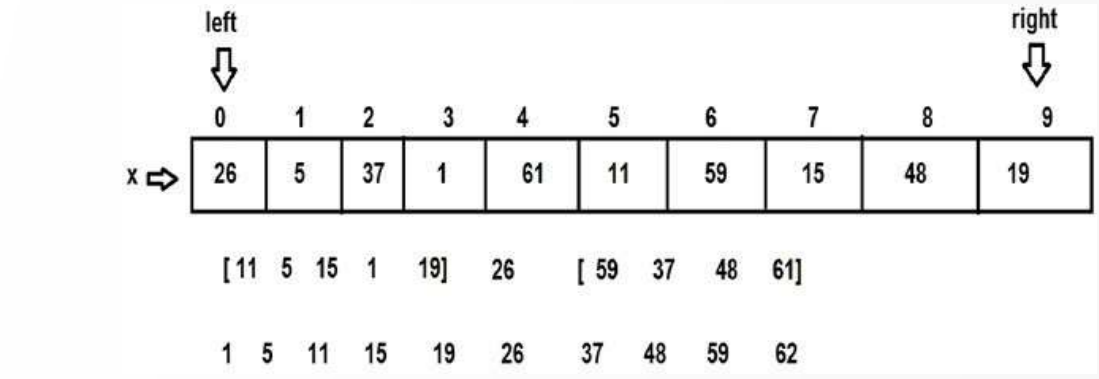
در این روش در هر گذر یک عنصر محوری یا (pivot)انتخاب شده وسایر عناصر به گونه ای جا به جا می شود که کلیه عناصر بزرگتر از آن در یک طرف وکلیه عناصر کوچک تر در طرف دیگر قرار گیرد بدین ترتیب عنصر در مکان درست خود قرار خواهد گرفت سپس دو بردار دو طرف این عنصر با توجه به یکی از عناصر به صورت بازگشتی به همین ترتیب مرتب می شود این الگوریتم جزء الگوریتم های تقسیم بوده که با کوچک کردن بازه عمل به چند بازه سپس ترکیب جواب های آنها مساله را حل می کند

در روش مرتب سازی سریع مرتبه اجرایی آن$o(n.\log n)$ می باشد البته ممکن است در بدترین حالت به صورت (آرایه مرتب) آرایه به دو قسمت به طول صفر و$n-1$ تقسیم شده که در این حالت مرتبه اجرایی آن $o(n^2)$ می باشد.



الگوریتم مرتب سازی سریع:

```
.Quick sort(x[ ] , left , right)
{
int I , j , pivot;
 if(left < right)
{
i=left;
 j=right+١;
 pivot=x[left];
 while(i > j)
while(x[i] >= pivot)
i=i+١;
while(x[j] <= pivot)
 j--;
if(i<j)
 swap(x[i] , x[j])
swaple(left , x[j])
 Quick sort(x , left , j-١)
Quick sort(x , j+١ , right)
```



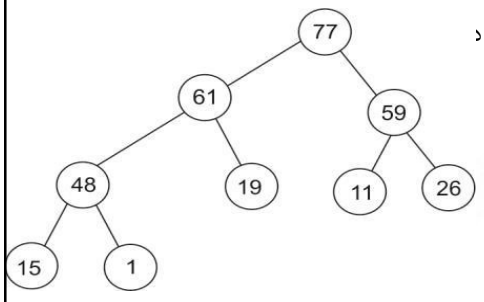
۶-مرتب‌سازی هرمی (Heap Sort)

در مرتب سازی heap، ساختار maxheap مورد استفاده قرار می گیرد. یک درخت heap با n رکورد با استفاده از تابع adjust ایجاد می شود. سپس رکوردها یکی یکی از heap خارج می شوند.

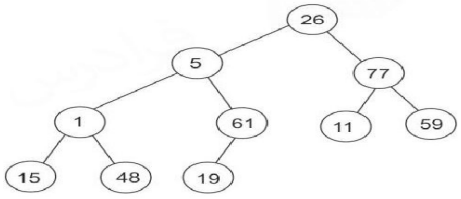
تابع adjust از یک درخت دودویی که زیردرختهای چپ و راست آن خاصیتheap را برآورده می کنند، استفاده می کند. (بـه غیر از ریشه) همچنین این تابع درخت را بگونه ای تنظیم می کند که کل درخت دودویی خاصیت heap را برآورده می کند. برای مرتب سازی یک لیست، n-1 گذر بر روی لیست اعمال می گردد. در هر گذر، اولین رکـورد در heap بـا آخـرین رکـورد تعویض می شود. چون اولین رکورد، همیشه شامل بزرگترین کلید است، این رکورد با بزرگترین کلیـد را در محـل n ام قـرار می دهیم. در گذر دوم، رکورد با دومین کلید بزرگ را در موقعیت n-1 قرار می دهیم و در نهایت، در i امین گذر، رکـورد بـا i امین کلید را در در موقعیت قرار می دهیم .

مثال

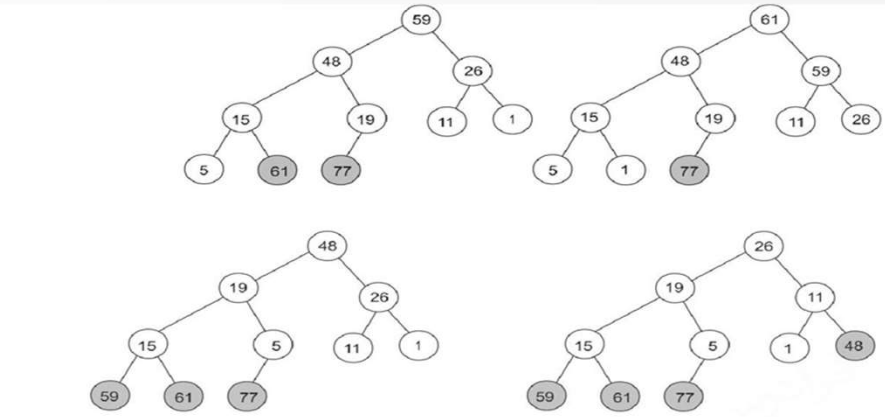
لیست ورودی (19 , 48 , 15 , 59 , 11 , 61 , 1 , 77 , 5 , 26) را در نظر بگیرید. شکل زیر درخت دودویی اولیـه را نشـان می دهد



بعد از تبدیل آن به maxheap د



فرایند مرتب سازی در زیر نشان داده شده است. دایره های پر رنگ رکوردهایی که در موقعیت مرتب شده خود قرار گرفته انـد را نشان می دهد.



تابع مرتب سازی Heap

```
void heapsort ( int  a[ ] , int  n ){
    int   i , j ;
for ( i = n/۲ ; i>۰ ; i--)
adjust ( a , i , n); for( i = n- ۱ ; i>۰ ; i--)
}

adjust ( a , i , n);
 for( i = n- 1 ; i>0 ; i--)
{
swap( a[1] , a[i+1] , temp );
 adjust ( a , 1, i );
}
}
```

```
void  adjust( int  a[ ] , int  root , int n )
{ int  child , rootkey, temp;
temp = a[root];
rootkey = a[root];
 child =2 * root;
while (child <= n)
{ if (( child <n) && (a[child] < a[child+1]) )
child++;
if ( rootkey > a[child] )
 break;
else{
a[child / 2] = a[child];  child = child *2;
} }
a[child / 2] = temp;
```

در حلقه for اول در تابع heapsort ، تابع adjust یکبار به ازای همه گره ها که دارای یک فرزند هستند، فراخوانی می شود. زمان این حلقه بیشتر از O(n) نمی باشد. در حلقه for دوم، تابع adjust به تعداد (n -1) مرتبه با حداکثر عمق log(n +1) فراخوانی می شود. بنابراین زمان آن O(n log n) می باشد. در نتیجه کل زمان محاسباتی O(n log n) خواهد شد.

۷-مرتب سازی ادغام (Merge Sort)

در این روش آرایه n عنصری به n قسمت به طول یک تقسیم و سپس هر دو قسمت مجاور به صورت مرتب شده با هم ادغام می‌شوند. در این حالت آرایه به طول ۲ ایجاد شده است. روند ادغام تا رسیدن به یک آرایه به طول n ادامه می‌یابد. این مرتب سازی، برای ادغام دو آرایه مرتب در یک آرایه مرتب جدید بسیار مناسب است.

پیچیدگی زمانی این الگوریتم o(n.log n) است

الگوریتم مرتب سازی ادغام

```
void merge-sort(int a[ ] , int n)
{
int len=1;
int extra[ MAX];
while(len < n) {
merge-pass(a , extra , n , len);
len=len *2;
merge-pass(extra , a , n , len);
len = len *2;
}
}
```

```
void merge-pass(int a[ ] , int s [ ] , int n
 , int len){
int i , j;
for ( i=0 ; i <= n-2*len ; i = i+2*len )
merge( a , s , i , i+len-1 , i+2*len-1 );
if ( i+len < n)
merge( a , s , i , i+len-1 , n-1 );
else
for ( j = i ; j < n; j++) s[j] = a[j];
}
```

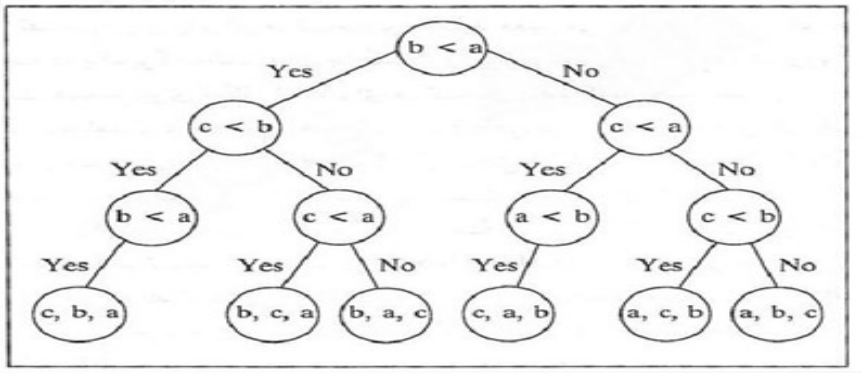
```
void merge( int a[ ] , int s[ ] , int i
 , int m , int n){
int j , k , t;
j = m+1;
k = i;
while( i <= m && j <= n ) {
if ( a[i] <= a[j] )
s[k++] = a[i++];
else
s[k++] = a[j++]; }
if ( i > m)
for( t = j ; t <= n ; t++ )
s[k+t-j] = a[t];
else
for( t = i ; t <= m ; t++ ) s[k+t-i] =
a[t];
```

۸-مرتب سازی درختی (Tree Sort)

ابتدا با اعداد داده شده یک BST ساخته و سپس با پیمایش میانوندی درخت حاصل، رشته صعودی حاصل می شود. این موضوع در مبحث BST بررسی شد.

درخت تصمیم گیری برای الگوریتم های مرتب سازی

متناظر با هر الگوریتم قطعی برای مرتب سازی n کلید، حداقل یک درخت تصمیم گیری معتبر وجود دارد. در شکل زیر درخت تصمیم گیری متناظر با مرتب سازی تعویضی، در هنگام مرتب سازی سه کلید رسم شده است:



در درخت بالا، گره سطح ۲ حاوی مقایسه "b<a" هیچ فرزند راستی ندارد. چون پاسخ No به به آن مقایسه، پاسخ هایی را نقض میکند که روی مسیر منتهای به آن گره به دست می آید

نکته:

۱- تعداد مقایسه های انجام شده توسط یک درخت تصمیم گیری در بدترین حالت برابر با عمق درخت است.

۲- هر الگوریتم قطعی که n کلید متمایز را فقط با مقایسه کلیدها مرتب سازی می کند، باید در بدترین حالت حداقل$\lg(n!)$ مقایسه کلیدها را انجام دهد.

۳- درخت تصمیم گیری که فقط از طریق مقایسه کردن عمل مرتب سازی را انجام می دهد، دارای $n!$ برگ است

۹-مرتب سازی مبنا (Radix Sort)

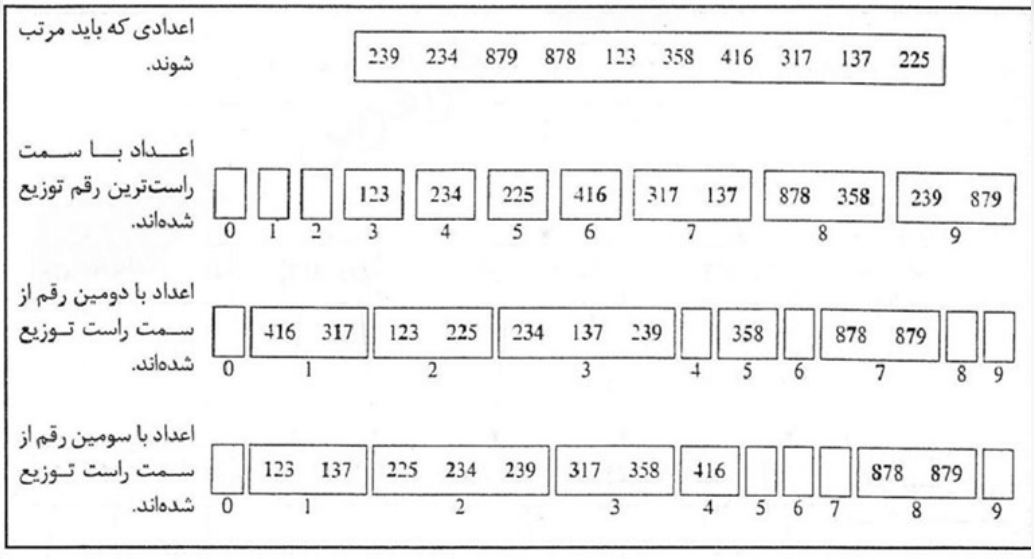
در مرتب سازی مبنا (توزیعی)، با فرض اینکه اعداد در مبنای 10 باشند، در گذر اول اعداد بر حسب اولین رقم سمت راست مرتب می شوند و در گذر دوم بر طبق دومین رقم از سمت راستشان و به همین روال مرتب سازی ادامه می یابد.(این نوع مرتب سازی مقایسه ای نمی باشد)

مثال

اعداد زیر را به روش مرتب سازی مبنایی، مرتب نمایید.

۲۳۹ , ۲۳۴ , ۸۷۹ , ۸۷۸ , ۱۲۳ , ۳۵۸ , ۴۱۶ , ۳۱۷ , ۱۳۷ , ۲۲۵

حل: چون مبنا ۱۰ است، ده گروه از ۰ تا ۹ در نظر می گیریم. ارقام را از راست به چپ بازرسی کرده و هر کلید را در گروه متناظر با رقمی که در حال حاضر بازرسی می شود، قرار می دهیم. بعد از پایان گذر اول، اعداد بر حسب اولین رقم سمت راست مرتب شده اند. بعد از پایان گذر دوم، اعداد بر حسب دومین رقم سمت راست مرتب شده اند و .... این روال در شکل زیر نشان داده شده است.



با هر بار گذر، اگر دو کلید باید در یک گروه قرار گیرند، کلیدی که از گروه واقع در منتها الیه سمت چپ در گذر قبلی می آید، در طرف چپ کلید دیگر قرار می گیرد. به عنوان مثال بعد از اولین گذر، کلید 416 در گروه واقع در چپ کلید 317 قرار دارد. بنابراین، هنگامی که در دومین گذر، هر دو آن ها در گروه اول قرار بگیرند، کلید 416 در طرف چپ کلید 317 قرار می گیـرد. ولی در گذر سوم، کلید 416 به طرف راست کلید 317 می رود، چون در گروه چهارم قرار داده می شود، سپس کلیـد 317 در گروه سوم قرار داده می شود.

پیچیدگی زمانی: مرتب سازی مبنا n عدد d رقمی که در آنها هر رقم می تواند تا k مقدار ممکن را بگیرد در زمان $\theta(d(k + n))$ مرتب می کند.

10-الگوریتم مرتب سازی شِل(Shell sort)

مرتب سازی شل یکی از الگوریتم‌های مرتب سازی بسیار بهینه است که بر مبنای الگوریتم مرتب سازی درجی عمل می‌کند. این الگوریتم از جابجایی‌های بزرگی که در آن مقادیر کوچک‌تر در موقعیت‌های دورتر در سمت راست یا چپ قرار دارند و در مرتب سازی درجی وجود داشت، اجتناب می‌کند.

این الگوریتم از مرتب سازی درجی بر روی عناصر بسیار دور از هم استفاده می‌کند و ابتدا آن‌ها را مرتب سازی می‌کند و سپس عناصری که فاصله اندکی با هم دارند را مرتب می‌سازد. این فاصله‌گذاری به نام بازه (Interval) نامیده می‌شود.

این الگوریتم برای مجموعه داده‌های با اندازه متوسط بسیار بهینه است، زیرا سناریوی حالت متوسط آن دارای پیچیدگی $O(n\log n)$ است که n تعداد آیتم‌های لیست است. توجه داشته باشید که بدترین حالت آن نیز برابر با $O(n^2)$ است.

مقایسه روشهای مرتب سازی

از بین روشهای مختلف مرتب سازی هیچ کدام روشی ایده آل و مناسب همه شرایط نیست اما کارایی مرتب سازی درجی از مرتب سازی حبابی بیشتر است . در فایل های کوچک مرتب سازی انتخابی توصیه می شود ، چون عمل مقایسه کلیدها ارزان تر است .

ولی کارایی مرتب سازی heapsort برای مقادیر بزرگ n از کارایی بهتری برخوردار است.

مرتبه اجرایی مرتب سازی های مقایسه ای در جدول زیر نشان داده شده است.

نام روش	بهترین	میانگین	بدترین
Bubble	o(n)	$o(n^2)$	$o(n^2)$
Selection	$o(n^2)$	$o(n^2)$	$o(n^2)$
Insertion	o(n)	o(n)	$o(n^2)$
exchange	o(n)	$o(n^2)$	$o(n^2)$
Quick	o(n.log n)	o(n.log n)	$o(n^2)$
Merge	o(n.log n)	o(n.log n)	o(n.log n)
Heap	o(n.log n)	o(n.log n)	o(n.log n)
Tree	o(n.log n)	o(n.log n)	$o(n^2)$
Radix Sort	o(k+n)	o(k+n)	o(kn)
shell	o(n.log n)	o(n.log n)	$o(n^2)$

تمرینات فصل ۷

سوالات تستی :

۱- در الگوریتم مرتب سازی حبابی (یا تبادلی) فرض می کنیم T(n) تعداد دستورالعمل مقایسه باشد. در این صورت T(n) کدام است؟

۱) $T(n)=\frac{n(n+1)}{2}$ ۲) $T(n)=2n-1$

۳) $T(n)=\frac{n(n-1)}{2}$ ۴) $T(n)=\frac{n^2}{2}-1$

حل: جواب گزینه ۳ است.

تعداد مقایسه ها در روش مرتب سازی حبابی برابر $\frac{n(n-1)}{2}$ می باشد.

- اگر در مرتب سازی سریع، زیر لیست های، کوچکتر در ابتدا مرتب شوند، پشته بازگشتی دارای چه عمقی خواهد بود؟

۱) $o(n)$ ۲) $o(n^2)$ ۳) $o(\log n)$ ۴) $o(1)$

حل: جواب گزینه ۱ است.

اگر در روش Quick Sort ، داده ها هر بار به طور مساوی تقسیم شوند، حداکثر عمق درخت بازگشتی logn بوده و به فضای پشته $o(\log n)$ نیاز خواهد بود. اما در بدترین حالت یعنی وقتی داده ها به قسمتهای به طول n-1 و صفر تقسیم شده باشد، فضای پشته $o(n)$ خواهد بود.

اگر برای مرتب سازی آرایه ی زیر به صورت صعودی از الگوریتم Bubble Sort استفاده شود، چند عمل مقایسه و چند عمل جابه جایی صورت می گیرد؟ {3 , 5 , 2 , 1, 4 , 7}

۱) 3 عمل جابه جایی و 30 عمل مقایسه صورت می گیرد.

۲) 3 عمل جابه جایی و 15 عمل مقایسه صورت می گیرد.

۳) 6 عمل جابه جایی و 14 عمل مقایسه صورت می گیرد.

۴) 30 عمل جابه جایی و 30 عمل مقایسه صورت می گیرد.

حل: جواب گزینه ۳ است.

تعداد مقایسه ها در روش مرتب سازی حبابی برابر $\frac{n(n-1)}{2}$ می باشد که در بدترین حالت به همین تعداد، تعویض انجام می گیرد. بنابراین با قرار دادن 6 به جای n در این رابطه، عدد 15 حاصل می شود. بنابراین گزینه های ۱و۴ که اعداد بزرگتر از 15 دارند، حتما نادرست می باشند. این آرایه در ۳ گذر مرتب می شود.

گذر اول:

آرایه اولیه : 3 , 5 , 2 , 1 , 4 , 7

3 , 2 , 5 , 1 , 4 , 7
3 , 2 , 1 , 5 , 4 , 7
3 , 2 , 1 , 4 , 5 , 7

گذر دوم:

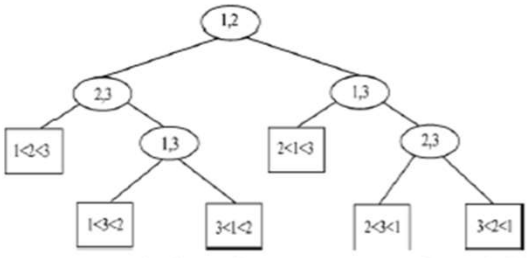
2 , 3 , 1 , 4 , 5 , 7
2 , 1 , 3 , 4 , 5 , 7

گذر سوم:

1 , 2 , 3 , 4 , 5 , 7

مشخص است که 6 جابه جایی لازم است. بنابراین گزینه ۳ درست است.

یک درخت دودویی تصمیم‌گیری را در نظر بگیرید که از آن برای مرتب کردن عناصر یک مجموعه براساس عمل مقایسه، استفاده می‌شود. برای مثال درخت روبرو، برای مرتب کردن یک فایل با سه عضو، استفاده می‌شود. کدام یک از گزاره‌های ذیل، در مورد این نوع درخت‌ها درست است؟



۱) عمق این نوع درخت‌ها، حداقل $\log_2(n!)$ است و در گروه $o(n.\log n)$ قرار می‌گیرد.

۲) عمق این نوع درخت‌ها، حداکثر $\log_2(n!)$ است و در گروه $o(n.\log n)$ قرار می‌گیرد.

۳) عمق این نوع درخت‌ها، حداکثر $\log_2(n!)$ است و حداقل $o(n.\log n)$ مقایسه در آنها انجام می‌شود.

۴) عمق این نوع درخت‌ها، حداقل $\log_2(n!)$ است و حداکثر $o(n.\log n)$ مقایسه در آنها انجام می‌شود.

حل: جواب گزینه ۱ است.

عمق درخت تصمیم گیری، حداقل برابر $\log_2(n!)$ می باشد. همچنین می دانیم که به جای $n!$ می توان n^n نوشت و به همین علت $\log_2(n!)$ در گروه $o(n.\log n)$ قرار می‌گیرد.

سوالات تشریحی :

مثال

اعداد 2 ، 3 ، 8 ، 4 را به روش حبابی، صعودی نمایید.

حل: گذر اول به صورت زیر می باشد:

مقایسه ۴ با ۸ : 4 ، 8 ، 3 ، 2

مقایسه ۳ با ۸ و تعویض آنها : 4 ، 3 ، 8 ، 2

مقایسه ۸ با ۲ و تعویض آنها : 4 ، 3 ، 2 ، 8

در پایان گذر اول ، بزرگترین عنصر در خانه آخر قرار گرفته است. گذر دوم به صورت زیر است:

مقایسه ۴ با ۳ و تعویض آنها : 3 ، 4 ، 2 ، 8

مقایسه ۴ با ۲ و تعویض آنها : 3 ، 2 ، 4 ، 8

گذر سوم:

مقایسه ۳ با ۲ و تعویض آنها : 2 ، 3 ، 4 ، 8

عناصر داده شده بعد از ۳ گذر با ۶ مقایسه که ۵ تا از آنها منجر به تعویض شد، مرتب شدند.

آرایه زیر را به روش درجی مرتب سازید.

20 , 5 , 35 , 8 , 2

حل:

درج ۲۰					20
درج ۵				20	5
درج ۳۵			35	20	5
درج ۸		35	20	8	5
درج ۲	35	20	8	5	2

به عبارتی الگوریتم مرتب سازی درجی الگوریتمی است که مرتب سازی را با درج رکوردها در یک آرایه مرتب شده موجود مرتب سازی می کند.

لیست داده شده در روش Merge Sort ، بعد از چند گذر مرتب می شود؟

2 , 3 , 1, 7 , 6 , 5, 4, 0

حل:

[2] [3] [1] [7] [6] [5] [4] [0]
[2,3] [1,7] [5,6] [0,4]
[1,2,3,7] [0,4,5,6]
[0,1,2,3,4,5,6,7]

تعداد مقایسه های لازم برای مرتب کردن لیست زیر به روش انتخابی را محاسبه نمایید.

10 , 30 , 17 , 12 , 1 , 21 , 15

حل: در ابتدا محل کوچک ترین عنصر یعنی 1 را با عنصر اول تعویض کرده و سپس محل کوچکترین عنصر دوم یعنی 10 با عنصر دوم تعویض می‌شود. به عبارتی در مرحله دوم کوچکترین عنصر آرایه 2 تا n پیدا شده و محل آن با عنصر اول این آرایه تعویض می شود. عملیات به همین روال ادامه می یابد.

لیست اولیه	10	30	17	12	1	21	15
مرحله اول	1	30	17	12	10	21	15
مرحله دوم	1	10	17	12	30	21	15
مرحله سوم	1	10	12	17	30	21	15
مرحله چهارم	1	10	12	15	30	21	17
مرحله پنجم	1	10	12	15	17	21	30
مرحله ششم	1	10	12	15	17	21	30

در مرحله اول برای پیدا کردن کوچکترین عنصر نیاز به شش مقایسه می باشد. در مرحله دوم برای پیدا کردن کوچکترین عنصر لیست جدید (از خانه ۲ تا انتها) به پنج مقایسه نیاز است و به همین ترتیبی تعداد مقایسه ها را بدست آورده و خواهیم داشت: 6+5+4+3+2+1=21

تمرینات های فصل هفتم:

برای اعداد زیر، مرتب‌سازی درجی را دنبال کنید:

a. 13, 57, 85, 70, 22, 64, 48
b. 13, 57, 12, 39, 40, 54, 55, 2, 68
c. 70, 57, 99, 34, 56, 89

برای اعداد زیر، مرتب‌سازی حبابی را دنبال کنید:

d. 13, 57, 85, 70, 22, 64, 48
e. 13, 57, 12, 39, 40, 54, 55, 2, 68
f. 70, 57, 99, 34, 56, 89

برای اعداد زیر، مرتب‌سازی سریع را دنبال کنید:

j. 13, 57, 85, 70, 22, 64, 48
k. 13, 57, 12, 39, 40, 54, 55, 2, 68
l. 70, 57, 99, 34, 56, 89

موفق باشید