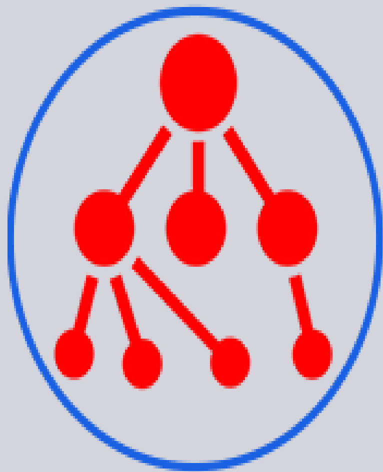


DATA STRUCTURE

Amin Jalilzadeh razin
Hossein Khalili

Data Structures

in

Data Types

Binary Trees

Arrays

Stacks and Queues

Linked Lists

Operations Associated

Chamran Vocational And Engineering College

فهرست مطالب

فصل اول: مفاهیم و مقدمات

خصوصیات الگوریتم........ ٦
انواع ساختمان داده ها........ ٦
پیچیدگی........ ٩
بررسی کارایی الگوریتم........ ١٤
نقاط رشد تابع........ ١٦
توابع بازگشتی........ ١٦

فصل دوم: آرایه رشته و ماتریس

آرایه........ ٢١
جست و جوی دودویی........ ٢٥
ماتریس........ ٢٧
رشته........ ٢٨

فصل سوم: پشته و صف

پشته........ ٣١
POP........ ٣١
PUSH........ ٣٢

درخت پارس.......... ۳۵
پارانتز بندی.......... 37
صف.......... 41
صف حلقوی.......... 44

فصل چهارم: لیست پیوندی

لیست پیوندی یک طرفه خطی.......... 49
لیست حلقوی یک طرفه.......... 50
لیست پیوندی دو طرفه.......... 51
لیست عمومی.......... 52

فصل پنجم: درخت یا tree

درخت.......... 54
درخت دودویی.......... 56
درخت عمومی.......... 62

فصل ششم: درختان ویژه

درخت heap.......... 65
درخت BST.......... 67
درخت با ارتفاع متوازن.......... 70

درختAVL...71
درخت تصمیم گیری...72
درخت دودویی گسترش یافته توی مسیر...71
الگوریتم هافمن...74

فصل هفتم:گراف

گراف تهی یا پوچ...81
گراف چندگانه...82
ماتریس مجاورتی...85
لیست مجاوتی...85
درخت پوشا...86
الگوریتم راشال کروسکال...87
الگوریتم پریم...88

فصل هشتم: مرتب سازی

مرتب سازی انتخابی...91
مرتب سازی حبابی...93
مرتب سازی درجی...94
مرتب سازی جا به جایی...95
مرتب سازی سریع...96

با تشکر ویژه از جناب آقای علی شکری

فصل اول

مفاهیم و مقدمات

تعریف الگوریتم: روش دقیق حل یک مسئله به صورت قدم به قدم که لزوما منحصر به فرد نیست

تعریف خصوصیات الگوریتم:

۱-ورودی (حداقل صفر ورودی).

۲-خروجی (حداقل یک خروجی).

۳-قطعی و عدم ابهام (هرکدام از دستورالعمل ها دقیق و بدون ابهام باشد).

٤-محدودیت (پایان پذیر بودن) یعنی هر الگوریتم پس از طی مراحل مشخص به پایان یابد.

٥-کارایی یا انجام پذیری (امکان پیاده سازی و اجرای الگوریتم روی کاغذ وجود داشته باشد و به عبارتی بهتر الگوریتم انجام شدنی باشد.)

تفاوت برنامه و الگوریتم: یک برنامه تمامی خصوصیات یک الگوریتم را به جز شرط پایان پذیر بودن شامل می شود به عنوان مثال سیستم عامل برنامه ای است که هیچ گاه پایان نمی پذیرد و همواره در حال اجراست تا برنامه بعدی وارد چرخه پردازش شود.

تعریف ساختمان داده: به ساختار هایی که جهت دریافت داده های خام به شکل مناسب توسط کامپیوتر برای پیاده سازی و اجرای الگوریتم های مختلف مورد استفاده قرار می گیرد ساختمان داده گفته می شود.

انواع ساختمان داده: ۱-ایستا-۲-نیمه ایستا-۳-پویا

۱-ایستا

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

۲-نیمه ایستا

پشته: پشته یا Stack به ساختمان داده ای گفته ای می شود که مجموعه ای از المان ها را بر اساس اصل LIFO (اولویت خروج با عناصر تازه وارد) در خود نگه داری می کند و از دو عمل Push برای افزودن آیتم و Pop برای حذف آیتم پشتیبانی می کند.
صف: صف یا Queue عنوان ساختمان داده ای است که مجموعه ای از المان ها را بر اساس اصل FIFO (خروج به ترتیب ورود) درخود نگه داری می کند.

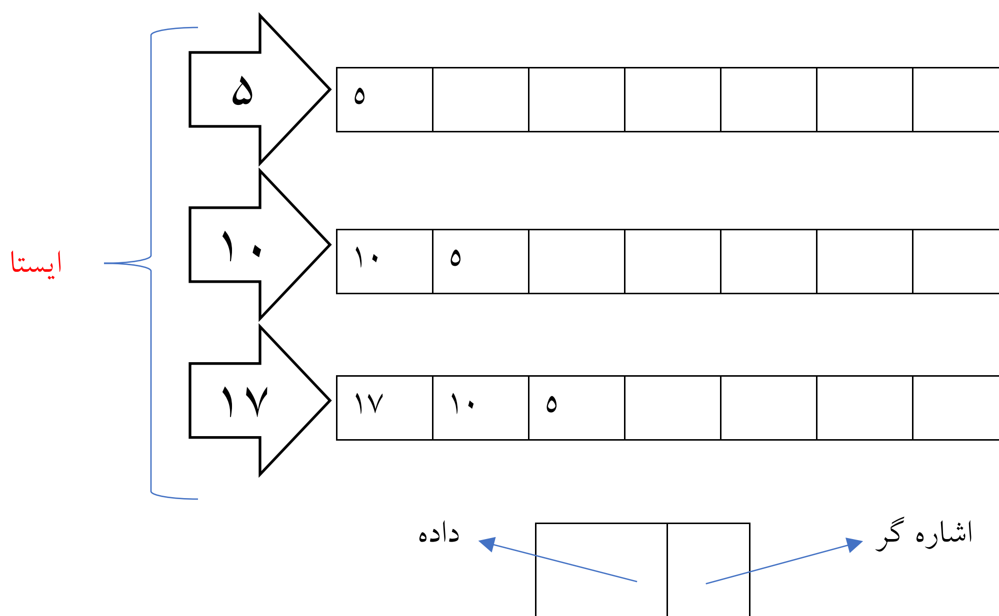
۳-پویا

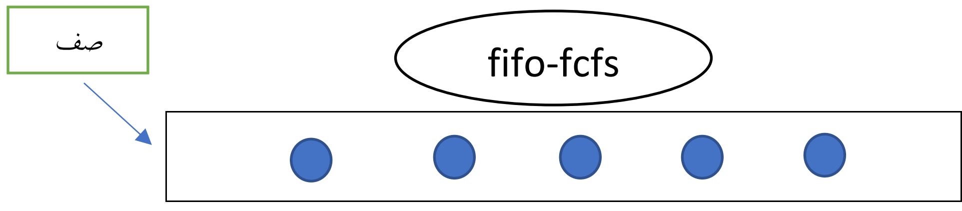
خطی: لیست پیوندی
غیرخطی: درخت و گراف

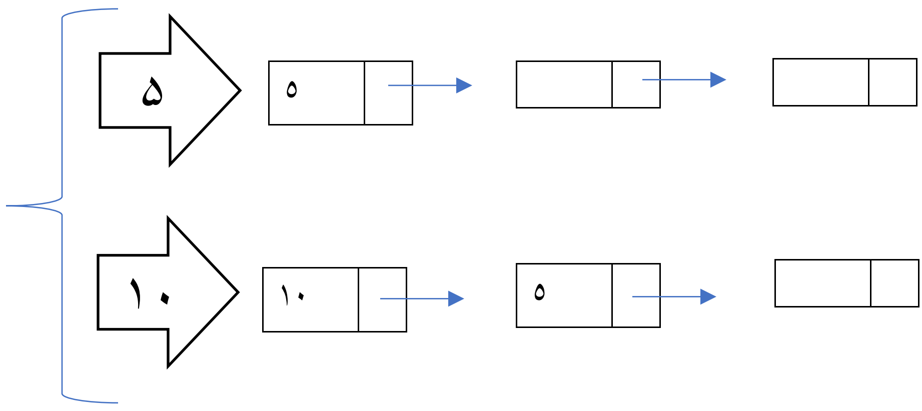
داده های ایستا: ساختار هایی هستند که از فضای محدود و از پیش تعریف شده استفاده می کنند و چنین فضایی در طول اجرای یک برنامه ثابت است.

داده پویا: ساختار هایی هستند که امکان تغییرات نامحدود و متناسب با داده هارو فراهم می کنند واین کار معمولا توسط اشاره گرها صورت می گیرد.

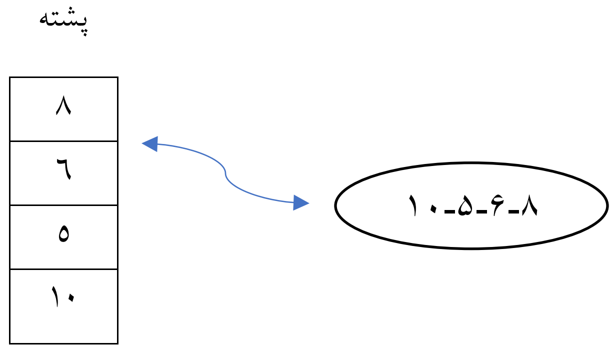
اشاره گر ها و آرایه ها از جمله مهم ترین ساختار های داده ای ایستا هستند. پشته و صف هم جزء ساختمان های داده ای نیمه ایستا هستند.



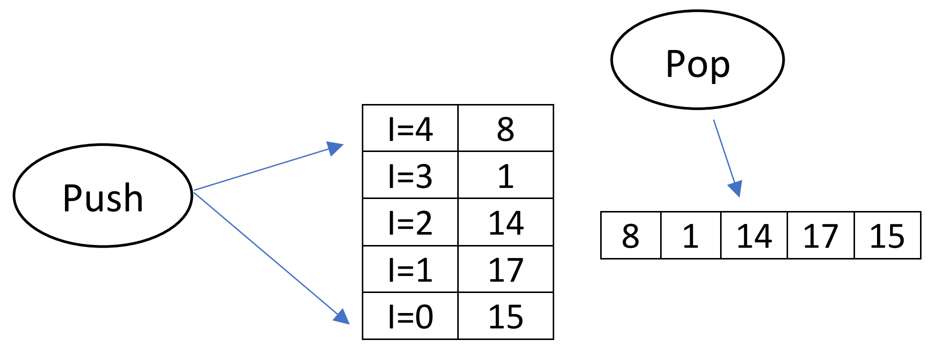




پشته




```
int x[5]={15,17,14,,1,8}
int y[5];
for(int i=0;i<5;i++)
{
  push(x[i])
```



```
}
for(int i=0;i<5;i++)
{
y[i]=pop()
}
```

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.

در تحلیل زمان اجرای یک الگوریتم تمام دستورات را شمارش نمی کنیم زیرا تعداد دستورات به نوع زبان برنامه نویسی بستگی دارد.برای محاسبه زمان اجرای یک الگوریتم فقط به عملیات اصلی نیاز داریم که مستقل از کامپیوتر و زبان برنامه نویسی هستند در عین حال هیچ قاعده مشخصی برای انتخاب عملیات اصلی وجود ندارد وانتخاب عمل اصلی به صورت تجربی انجام می شود.

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 .

مثال: اگر ورودی یک گراف باشد علاوه بر تعداد راس ها (n) یال ها نیز(n) یکی از مشخصه های ورودی است بنابراین زمان اجرای الگوریتم با T(n-m) شمارش می شود.

نکته: در محاسبه زمان T(n) یک الگوریتم محاسبه تعداد تکرارعملیات و توابع بازگشتی اهمیت بیشتری دارد.

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.

۴- درهردستوراجرایی به ازای هربار اجرا دارای یک گام است.

```
Procedure        P(........)      0 ←
Function         F(........)      0 ←
Void             F(........)      0 ←
                 int x;           0 ←
                 int x=1;         1 ←
```

مثال:

```
1)x=0 ← 1
2)for(i=0;i<n;i++) ← n+1
3){ ← 0
4)x++ ← 1*n=n
5)} ← 0
```

جواب: 2n+2 مرتبه

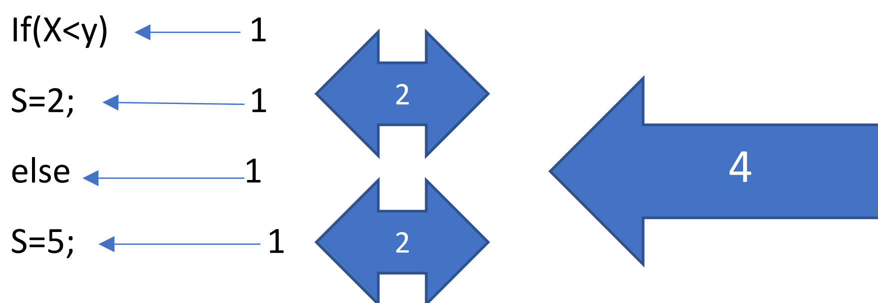
مثال:

```
for(i=1;i<n;i++) ← n+1
{ ← 0
S=s+1; ← n
} ← 0
```

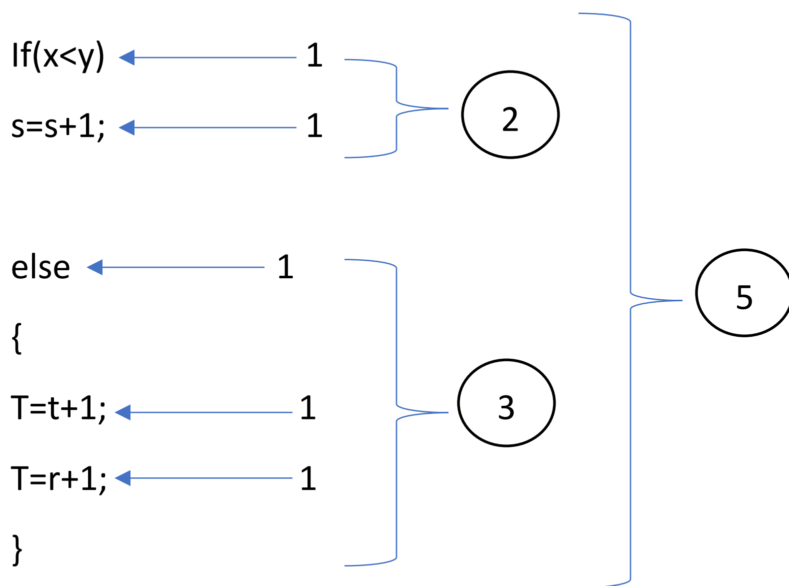
جواب: (n+1)+(n)=2n+1

عبارت شرطی if: عبارت شرطی یک گام دارد اما با توجه به درست و غلط بودن شرط 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 for به تعداد تکرار + یک گام اختیار می کند.

ج)جملات تکرار شونده داخل حلقه for به 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)

مثال:

```
for(i=10;i>1;i++) <----------- (10-1)+1=9+1=10
{
x++ <----------- 9
}
```

جواب:10+9=19

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 ۲ را برای تمامی حلقه ها انجام می دهیم.

```
for(i=0;i<n;i++)  ──────▶ n+1
for(j=0;j<m:j++)  ──────▶ n*(m+1)
{
S=s+1  ──────▶ m*n
}
```

جواب= ((n+1)+(n*(m+1))+(m*n)=n+1+nm+n+nm=2nm+2n+1

تعداد گام حلقه های whileمانند محاسبه می شود یعنی گام حلقه یک واحد از تعداد تکرار حلقه بیشتر است.

مثال؟

```
i=1;  ──────▶ 1
while(i<=n)  ──────▶ n+1
{
S=s+1;  ──────▶ n
i=i+1;  ──────▶ n
}
```

جواب=1+n+1+n+n=3n+2

بررسی کارایی الگوریتم:

bigO(۱

۲)bigΘ

3)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) O** وجود داشته باشد که برای همه ی مقادیر n>=nO داشته باشیم **T(n)<=C(f(n))**

T(n) ∈ O(f)n)) ⇔ ∃ c,n0>0 ∀ n ≥ n0 T(n)≤ cf(n)

T(n) ∈ Of(n) **f(n)** **T(n)** میباشد.

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

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

ثابت لگاریتمی رادیکالی مرتبه۲ مرتبهk نمایی توانی

مرتبه اجرایی: O(1)<log(n)<√n<n<n log n<n^2<n^2 log n<…<n^k<2^n<n!<n^n

مثال؟

T(n)=2n+2 O(n)
C=3 n=2

T(n)<=ef(n)
T(n)<=3*(n)

درستی یا نادرستی عبارت های زیر را مشخص کنید

T(n)=2n+1 ∈ O(n^2) √
T(n)=5n^2+n+1 ∈ O(n) ×

T(n)=(4*2^n*n^2) ∈ O(2^n) √

قضیه: اگر $T(n)=A_m R^m +...A_1 n+A_0$ زمان اجرای این الگوریتم عبارت است از؟

T(n) ∈ O(n^m)

:BigΩ گوییم T(n) ∈ Ω(f(n)) ⟵⟶ ∃ C\c, n0 > 0 ∀n > n0 T(n)>C f(n) اگر ثابت صحیح C,n0 وجود داشته باشد که به ازای همه ی مقادیر $n0 \geq m$ داشته باشیم.

T(n)≥C f(n)

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

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

مثال؟

T(n)=6n+4 ∈ Ω (n) √

∈ Ω (n^2)

نمادΘ: گوییم

$$T(n) \in \theta(fn) \leftrightarrow \exists\ c1, c2, n0 > 0\ \forall\ n \geq n0\ c1 * g(n) \leq T(n) \leq c2\ f(n)$$

اگر ثابت های c1,c2 وثابت صحیح n0 وجود داشته باشد به طوری که برای همه ی مقادیر n1≥ 0 بتوان گفت $c1\ f(n) \leq T(n) \leq c2\ f(n)$

با استفاده از نماد θ تابع T(n) هم ازبالا و هم ازپایین محدود می شود درجه رشد تابع f(n) T(n) ۱ است نماد θ زمانی به کار می رود که نرخ رشد دو تابع مساوی باشد نماد θ دقیق ترین بیان برای رشد 1 تابع است.

مثال؟

$$T(n) = \frac{1}{2}\, n^2 - 3n \in \theta(n^2) \qquad c2 \geq \frac{1}{2} \quad n \geq 1$$

$$T(n) \in \theta(n^2) \qquad c1 = \frac{1}{4} \quad n \geq 7$$

$$c1\ f(n) \leq T(n) \leq c2\ f(n) \qquad \in\ \theta\ (n^2)$$

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

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

نقاط رشد تابع

۱. لگاریتمN با هر پایه ای رشدشان باهم برابر است به شرطی که پایه پایشان ثابت باشد.

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

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

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

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

log n > log log n

مثال؟

$N^2 > \mathrm{Log}^3 N$　　　$N^{\frac{1}{2}} > \mathrm{LogN}$

مثال؟

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

توابع نمایی رشدشان از توابع چند جمله ای بیشتر است

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

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

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.

۱. بدون بازگشتی(عادی)
۲. بازگشتی

مثال؟

```
Int n;
int f=1;
cin>>n;
for(i=1;i<=n;i++)
F=f*i;
cout<<f;
```

مثال؟

```
fact(n)
{
cin>>n;
if(n<=1)
Return 1;
else
Return n*fact(n-1);
}
```

توابع بازگشتی دو ویژگی دارند:

۱. تابع خودش خودش را صدا می زند البته اغلب با آرگومان های کوپکتر.
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 stack over flow نشان داده میشود(پشته همیشه از بالا به پایین پر میشود)

مثال؟

در برنامه زیر خروجی f(3,6) را بدست آورید.

```
int f(int m,int n)
{
if(m==1||n==0||m==n)
Return 1;
else
Return f(m-1,n)+f(m-1 , n-1)
}
```

جواب:

f(3,6)
f(2,6)+f(2+5)
f(1,6)+f(1,5)+f(1,5)+f(1,4)=4

مثال؟

M(A,B)	M(B,A)	A<B
	A	B=0 → M(20,28)

M(20,28)	A%B
M(28,20)	28%20=8
M(20,8)	20%8=4
M(8,4)	8%4=0
M(4,0)	A=4

فرض کنید که A,B عدد صحیح مثبت بوده و تابع Q به صورت زیر به شکل بازگشتی نعریف شده باشد.

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

حاصل=Q(14,3)

مثال؟

```
int test(int x,int y)
```

```
{
if(x<=y||y==0)
Test=x;
else
if(y==1)                                   test(x=5 , y=2)
Test=test(x-1,y)+1;
else
```

جواب=٤

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

فصل ۲

آرایه-رشته-ماتریس

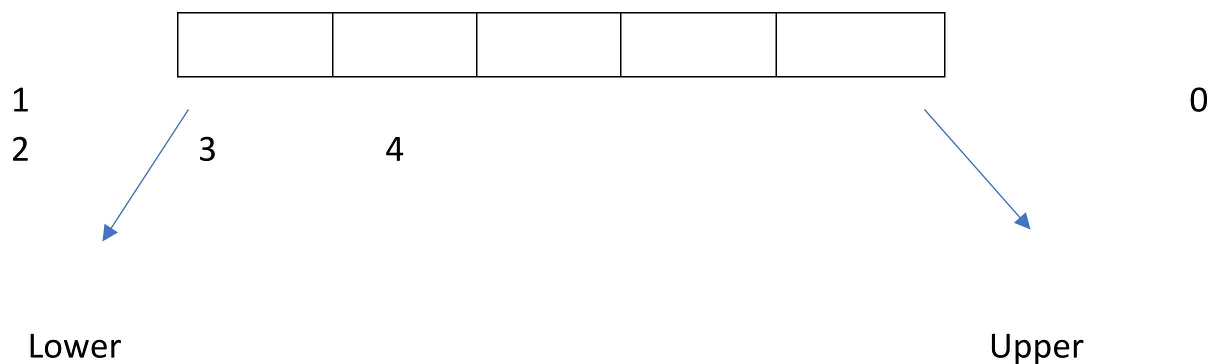
آرایه: مجموعه ای از داده های پشت سر هم که همگی از یک نوع بوده و از آدرس مشخص شروع می شوند.
عناصر یک آرایه با اندیسشان قابل دستیابی هستند آرایه میتواند یک بعدی دو بعدی یا چند بعدی باشد.
آرایه یک بعدی: آرایه یک بعدی برای ذخیره سازی مجموعه ای از عناصر هم نوع به کار میرود.
عناصر آرایه یک بعدی در محل های متوالی حافظه ذخیره میشوند در این آرایه برای دستیابی به عنصری از آرایه از اندیس استفاده می شود

آرایه ۱بعدی: int A[10]　　A[L1.....u1]

آرایه ۲ بعدی: int A[10][20]　　A[L1.....u1 , L2.....u2]

آرایه n بعدی: A[L1..u1] , [L2..u2 , Ln...un]

نکته: کوچکترین اندیس آرایه حد پایین آرایه یا Lower یا (L) بزرگترین مقدار اندیس آرایه را کران بالای آرایه میگویند و با Upper یا (U) نشان میدهند.



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

A=[L1......u1]　　تعداد خانه　　u1-L1+1

A=[L1......u1 , L2......u2]　　تعداد خانه　　(u1-L1+1)*(u2-L2+1)

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

عناصر نوع آرایه ❊ تعداد عناصر آرایه=فضای آرایه

مثال؟

int A[10];　　10*4=40 byte

int A[-1…3 , 2…5]　　حافظه =20*4=80 byte

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

آدرس اولین محل

Loc(i)=basc(a)+i*size

یا به این صورت هم می توان نوشت:

A[i]آدرس خانه =(i-L1)*n+∝ ⟶ آدرس شروع آرایه

مثال؟

آرایه A مساوی[5,…22] تعریف شده است اگر این آرایه از آدرس ۱۰۰ حافظه به بعد قرار گرفته باشد آدرس خانهA[16] کدام است؟

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

A[i]=(i-L1)*n+∝

A[16]=(16-5)*4+100==11*4+100=144

تعداد عناصر آرایه =u1-L1+1=20-5+1=16

16*4=64= تعداد عناصر آرایه* فضای نوع آرایه= فضای ذخیره سازی

فرض کنید آرایه A به صورتfloat A[10] تعریف شده و عناصر این آرایه از خانه ۳۰۰۰ به بعد در حافظه قرار گیرند. با فرض اینکه سایط هر عدد اعشاری float 4 byte است آدرس خانه A=[7] را محاسبه کنید.

A[i]=(i-L1)*n+∝

A[7]=(7-0)*4+3000=3028

۱۵	۷	۱۲	۷۰
۱۸	۹	۳	٦۱

محاسبه آدرس دلخواه از یک آرایه مشخص کردن موقعیت عناصر یک آرایه با توجه به نحوه ذخیره سازی آنها به یکی از دو روش سط... است.

١٩	٢٤	٦	٦٥
٢٠	٣٤	٥٢	٤٤
١	١٦	٦١	٣٩

ذخیره سطری

سطر اول	سطر دوم	سطر سوم	سطر چهارم	سطر پنجم

ذخیره ستونی

ستون اول	ستون دوم	ستون سوم	ستون چهارم

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 ($\propto$) آدرس شروع آرایه است و اگر بیان نشود (0) در نظر گرفته خواهد شد.

بتا (β) فضای نوع عناصر آرایه است و اگر بیان نشود (1) در نظر گرفته خواهد شد.

محاسبه آدرس A=(I,j,k) در آرایه A[L1...U1,L2...U2,L3...U3] ؟

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

سه بعدی [I,j,k]

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

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

دو بعدی [i,j]

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

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

ماتریس int x[30],[8] را در نظر بگیرید که آدرس اولیه آن یعنی x1=200 در نظر گرفته شده است مطلوب است محاسبه آدرس سطری A=[12][4]

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

$200+[(12-0)*(7-0+1)+(4-0)]*4=200+400=600$

آرایه[1…3 , 1…8] از آدرس ۳۰۰۰ به بعد ذخیره شده است و هر خانه ٤ بایت اشغال کرده است مطلوب است آدرس A[2,3] به صورت سطری و ستونی محاسبه کنید.

$\propto=3000$

$\beta=4$

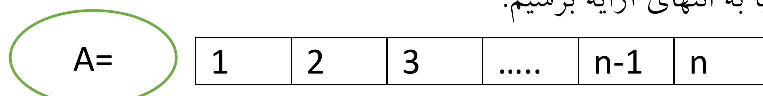
سطری = $A[2,3]=3000+[(2-1)(5-1+1)+(3-1)]*4=3028$

ستونی = $A[2,3]=3000+[(3-1)(3-1+1)+(2-1)]*4=3028$

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

۱. روش جست وجوی خطی(ترتیبی)
۲. جست و جوی دودویی(باینری)

روش جست و جوی خطی(ترتیبی): در این روش برای جست و جوی داده x در آرایه A[1…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.



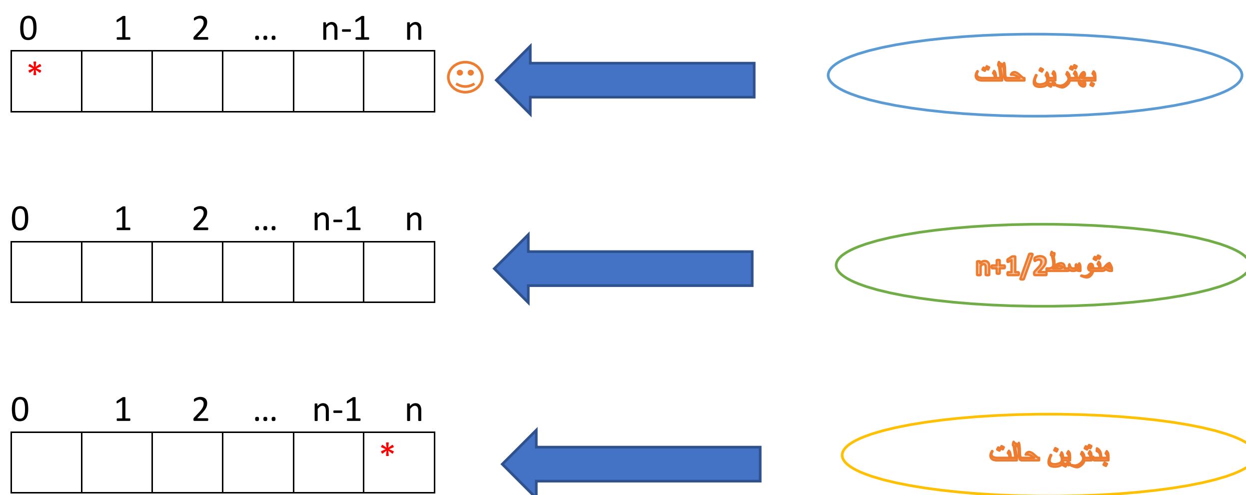
برنامه جست و جوی خطی

```
int seg-search(int A[] ; int n , int item)
{
for(int i=0 ; i<n ; i++)
{
if(a[i]==item)
return(i);
```

```
}
return(-1);
}
```

محاسبه زمان پیچیدگی:

1. بهترین حالت $1 \in T(n) = 0(1)$
2. متوسط $n + \frac{1}{2} \in T(n) = o(n)$
3. بدترین حالت $n \in T(n) = O(n)$



جست و جوی دودویی(باینری)

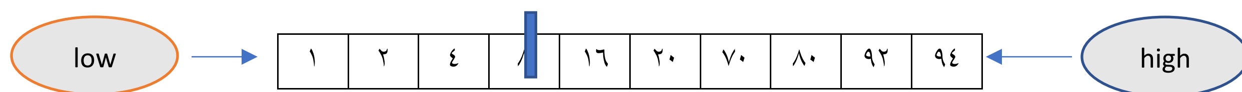
جست و جوی دودویی از یک آرایه مرتب امکان پذیر است که در ان هر آرایه به دو قسمت مساوی تقسیم می شود و در هر قسمت به جست و جو می پردازیم.

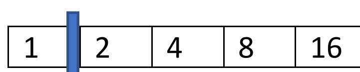
با توجه به بزرگتر یا کوچکتر بودن آیتم مورد نظر از بخش های انتخاب شده فضای جست و جو محدود تر شده و در نهایت عمصر مورد نظر پیدا می شود.

```
int binary-search(int a[] , int n , int item)
{
```

```
int mid , flag=0 , low=0, high=n-1;
while(low<=high && !flag)
{
mid=(low+high)/2;
}
if(item<a[mid])
{
high=mid-1;
else if(item>a[mid])
{
low=mid+1;
}
else
{
flag==1;
returm mid;
}
return -1;
}
```

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





8	16

8

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*n یک ماتریس یا جدول با m سطر و n ستون گفته می شود که تعداد m*n خانه در آن وجود دارد.

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

عناصر روی قطر اصلی: i==j

عناصر بالای قطر اصلی: i<j

عناصر پایین قطر اصلی: i>j

(۰و۰) ← ۱ ؛ ۵ → (۰و۴)

۱	۲	۳	٤	٥
٦	٧	٨	٩	١٠
١١	١٢	١٣	١٤	١٥
١٦	١٧	١٥	١٩	٢٠
٢١	٢٢	٢٣	٢٤	٢٥

تعریف ماتریس پایین مثلثی و بالا مثلثی: اگر عناصر زیر قطر اصلی ۰ باشند ماتریس بالا مثلثی است و اگر عناصر بالای قطر اصلی ۰ باشند ماتریس پایین مثلثی است.

شرایط ضرب دو ماتریس: برای اینکه ضرب دو ماتریس A,B امکان پذیر باشد باید ستون ماتریس A با سطر ماتریس B یکسان باشد.

$A[\ ]_{3*4} * B[\ ]_{4*5}$ 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_{20*4} B_{10*20} A_{5*10} را به چه ترتیبی انجام دهیم تا سریع تر اجرا شوند؟

$(A*B)*C \rightarrow 1400$

$A_{5*10} * B_{10*20} = AB_{5*2} \rightarrow 5*10*20=1000$

$(AB)_{5*20} * C_{20*4} = ABC_{5*4} \rightarrow 5*20*4=400$

$B_{10*20} * C_{20*4} = BC_{10*4} \rightarrow 10*20*4=800$

$A_{5*10} * BC_{10*4} = ABC_{5*4} \rightarrow 5*10*4=200$

$800+200=1000$

ترانه هادی یک ماتریس: اگر جای سطر و ستون یک ماتریس را عوض کنیم ماتریس ترانه هادی می شود.

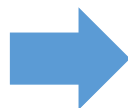
۱	۳	۰
۱۵	۲	۱
۱۲	۵	۱
۹	۱	۳
۱۸	۰	٤

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.

۴	۴	۲

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

5*6=30



18*4=72 byte

30*4=120 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+1 سطر استفاده می شود.

رشته ها

رشته ها در واقع آرایه ای از کاراکتر ها می باشند. در زبان پاسکال طول رشته در خانه ۰ ذخیره می شود و در زبان C در انتهای رشته کاراکتر (\0) ذخیره می گردد.

در زبان C

R	e	z	a	\0

H	a	s	a	n	\0

در زبان پاسکال

R	e	z	a

H	a	s	a	n

فصل ۳

پشته وصف

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.

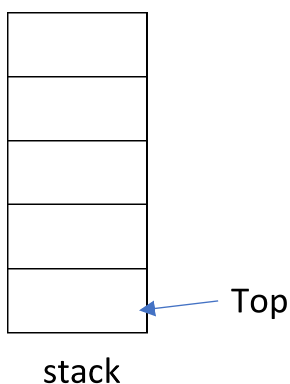
یکی از کاربرد های پشته ذخیره آدرس بازگشت و ساخت متغیر های محلی در صدا زدن توابع است.

به عمل ریختن اطلاعات در داخل پشته(پوش)و به برداشتن اطلاعات (پاپ) گفته می شود.

ساده ترین راه نمایش پشته استفاده از آرایه ۱ بعدی به طول n می باشد.

خانه های آرایه stack از عدد یک تا n شماره گذاری شده و در کنار آرایه متغیری به نام Top وجود دارد که عنصر بالایی به آن اشاره می کند.

Top از ۰ تا n تغییر می کند و در ابتدای کار Top=0 است.



if top == 0   خالی stack

if top == n   پر stack

الگوریتم pop:

```
int stack[n]
int pop()
{
int x;
if(top==0)
```

```
{
cout<<"پشته خالی است ";
return -1;
}
else
{
x=stack[top]
top--;
return x;
}
```

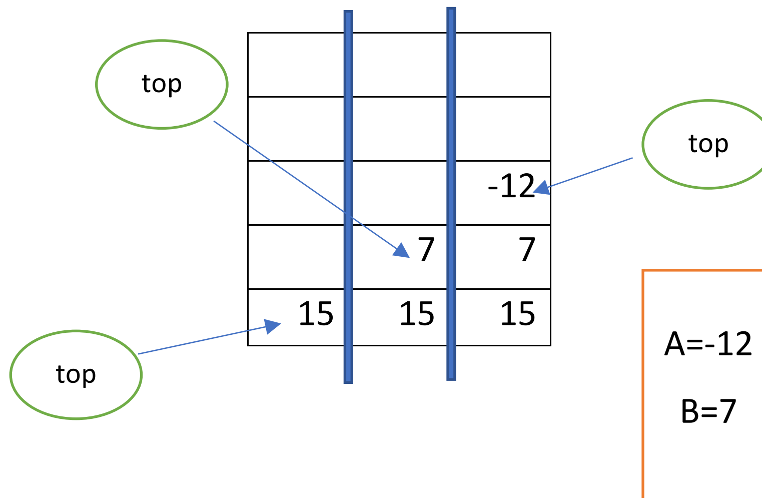
الگوریتم push :

```
int stack[n]
void push(int x)
{
if(top==n)
{
cout<<"پشته پر است ";
return -1
}
top++;
stack[top]=x;
}
```

نکته: stack از یک شروع می شود و تا n ادامه می یابد.

عملیات زیر را به ترتیب از سمت چپ به راست روی شکل نشان دهید(n = 5)

push(15)	A=pop()
push(7)	B=pop()
push(-12)	



مثال؟

مقدار نهایی A,B,C کدام است.(n = 5 , A = 10 , B = 2 , C = 5)

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()

کاربرد پشته:

۱. استفاده از توابع بازگشتی
۲. ارزیابی عبارت های محاسباتی یعنی(prefix , postfix , infix)
۳. الگوریتم مسیر حرکت maze
۴. پیمایش عمیقی گراف ها و درخت ها
۵. استفاده در روش های مرتب سازی (Quick sort , marge sort)

ارزشیابی عبارت های محاسباتی:

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

عملوند ← a*b → عملوند   Oprond

به یک عملوند نیاز دارند ← --a

عملگر:
1)یکانی (unary)
2)باینری (binary)

اولویت عملگر:

۱. () پارانتز	
۲. – منفی یا + مثبت	اولویت از چپ به راست
۳. توان	اولویت از چپ به راست
۴. * ضرب / تقسیم	اولویت از چپ به راست
۵. + جمع – منها	اولویت از چپ به راست

مثال؟

1. A+-b / c^d +e
2. A*b - c^d^ e +f
3. A* (b+c) -k/d +e
4. A / (b-c) -c * (f*g)

اولویت اول

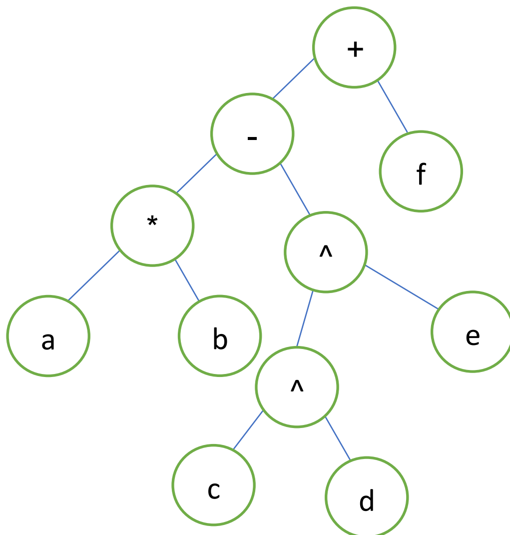
اولویت دوم

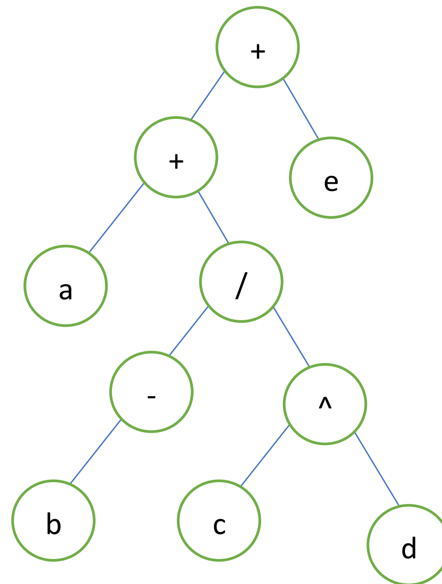
اولویت سوم

اولویت چهارم

اوبویت پنجم

درخت عبارت محاسباتی(درخت پارس)

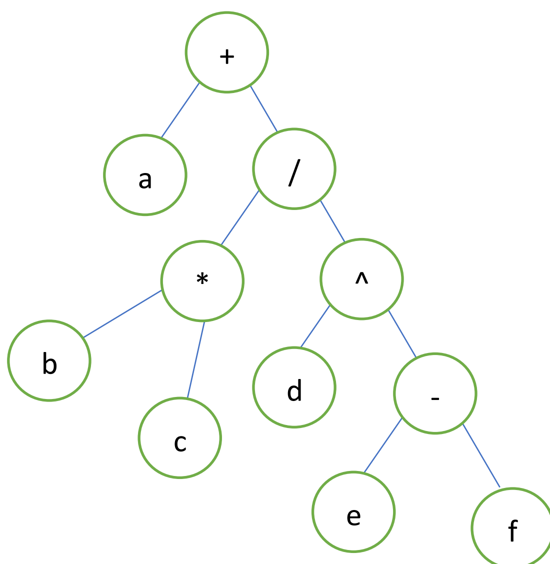




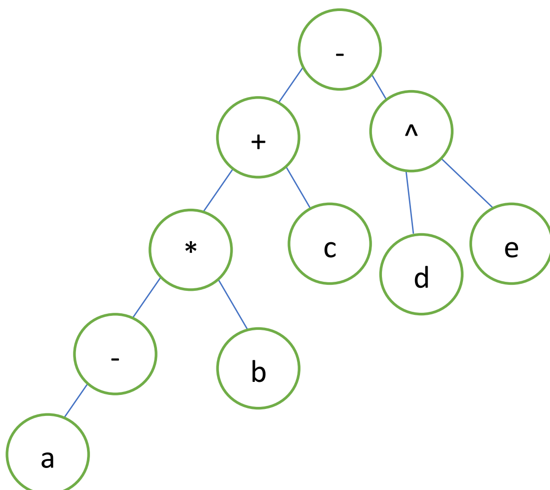
برای تشکیل درخت هر عبارت محاسباتی به صورت زیر عمل می کنیم.

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.
۴. برگ های درخت عملوند ها هستند

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



-a*b+c-d^e



پرانتز بندی:

(a+((b*c)/(d^(e0f))))

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

۱. pre fix عبارت پیشوندی روش لهستانی -+abc , +ab
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 → ef – d ^ bc * / a+

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 پارانتز استفاده نمی شود.

عبارت post fix :

در این روش هر عملگر بعد از عملوند هایش قرار می گیرد.اولویت مطرح نیست و از پارانتز هم استفاده نمی شود.

نکته: در تبدیل عبارت in fix به pre fix و post fix ترنیب 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 پارانتز گذاری
۲. روش استفاده از پشته
۳. روش پیمایش درخت عبارت محاسباتی

روش پارانتز گذاری:

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

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

in fix به post fix

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

abc + * gd / -

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

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

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

مثال؟

1- ((a*b)+c)
+*abc

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

3- ((a+((b^c)*d)-e)
-+a*^ bc de

تبدیل in fix به post fix با استفاده از پشته

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

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

الف)اگر پارانتز بسته"(" بود آن را در استک push می کنیم.

ب)اگر پارانتز باز")" بود تا زمانی که در اسـتک به پارانتز بسـته برسـیمpop می کنیم و در سـر راه هرچه عملگر دیدیم آن را در خروجی چاپ می کنیم.

ج)اگر یک عملگر بود و در سر استک پارنتز بسته ")" 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 سـر اسـتک اسـت اولوبت کمتری داشـت عملگر سـر اسـتک را از اســتک 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 بیشـتری از عملگر سـر اسـتک برخوردار باشد در این صورت عملگر جدیدی را که دیده ایم در استک push می کنیم.

اگر تک تک اجزای عبارت را از سـمت راسـت به چپ دیدیم ولی اسـتک خالی نشـده بود تمام محتویات اسـتک را تا زمانی که خالی شود pop می کنیم و عملگر ها را در خروجی نمایش می دهیم.

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 می

نویسیم 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 پارانتز بسته باشد هر عملگر به راحتی روی آن قرار می گیرد.

ه)به هر پارانتز باز که رسیدیم آنقدر از استک عملگر خارج می کنیم و در خروجی می نویسیم تا به پارانتز بسته برسیم در این وضعیت پارانتز 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

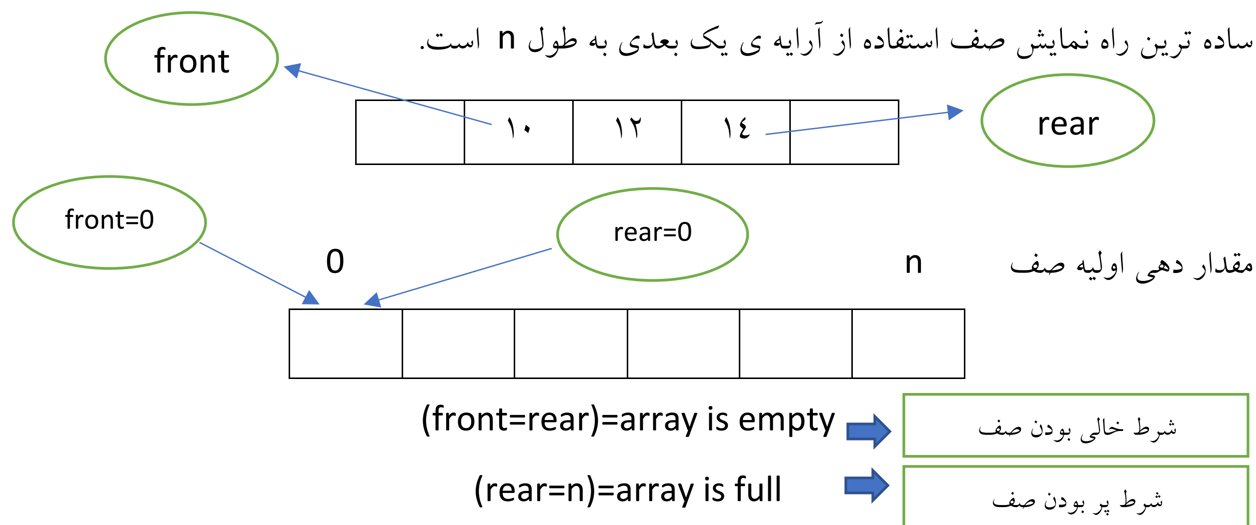
صف خطی یا صف معمولی(Queue) :

صف لیست خطی مرتبی از عناصر است 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 صف از دو متغیر اشاره گر به نام front که به عنصر قبل از عنصر ابتدایی اشاره می کند و دیگری به نام rear که همیشه به اخرین عنصر اشاره می کند.

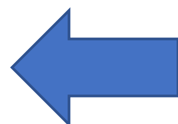
کاربرد مهم صف در زمان بندی برنامه ها در سیستم عامل است.نمایش صف با دو ساختار لیست پیوندی و آرایه ای می باشد.

ساده ترین راه نمایش صف استفاده از آرایه ی یک بعدی به طول n است.



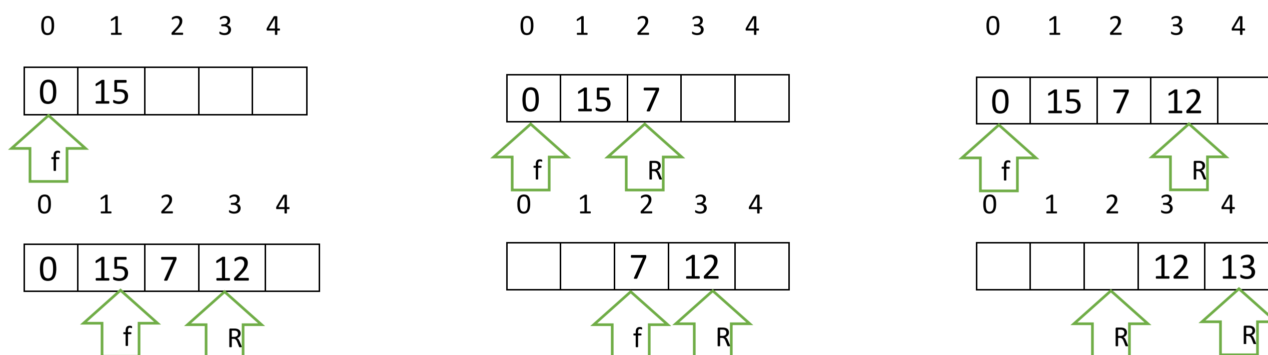
برای نوشتن زیر برنامه های اضافه و حذف از صف به صورت زیر عمل می کنیم:

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



اضافه کردن

```
double del Queue()
{
if(front==rear)
{
cout<<"Queue empty";
return 0;
}
else
{
front++;
}
return Queue[front];
}
```



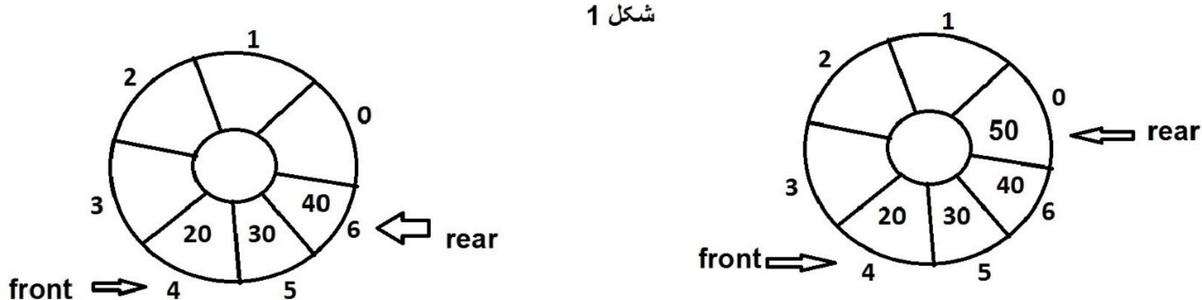
add Q(15) , add Q(7) , add Q(12) , A = del Q , B = del Q , add Q(13)

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

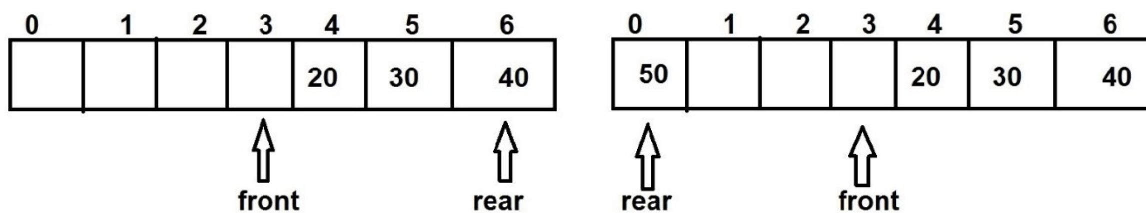
صف حلقوی:

آرایه انتخابی ۰ تا۱ [Q[0 , ... , n-1 را می توان به صـورت صـف 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.

مثال



شکل 2



زیر برنامه صف حلقوی:

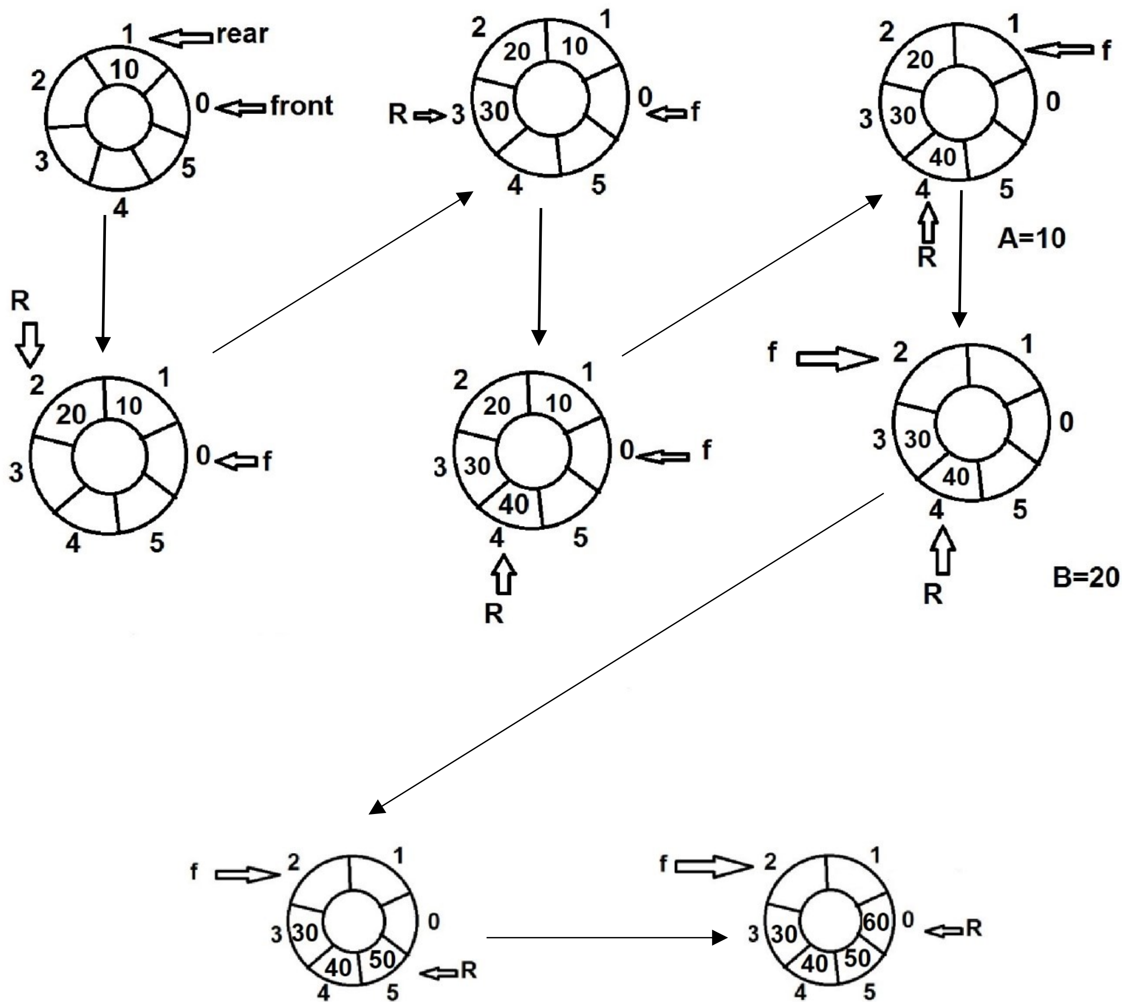
```
void add Q(double x)
{
if(front == (rear + 1) % n)
{
cout<<"Queue full";
}
else
{
rear = (rear + 1) % n);
Q[rear] = x;
}
```

```
double del Q()
{
if(front == rear)
{
cout<<"Queue empty";
return 0;
}
else
{
front = (front + 1) % n;
return Q[front];
```

مثال؟

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

add Q(10) , add Q(20) , add Q(30) , add Q(40) , A=del Q() , B=del Q() , add Q(50) , add Q(60)

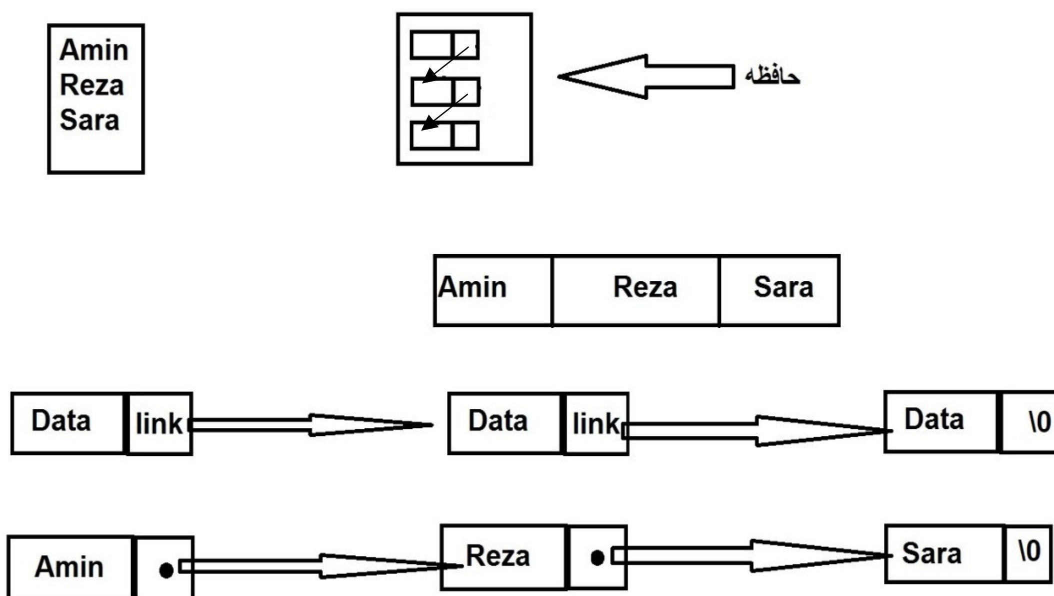


فصل ٤

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 ۳ دسته 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:

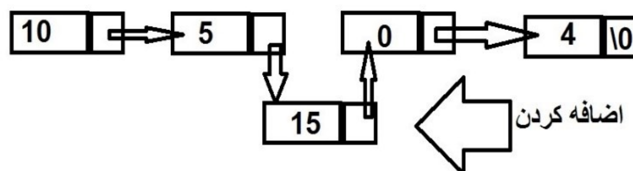
در لیست پیوندی یک طرفه خطی در هر گره فقط آدرس گره بعدی ذخیره می شود و اشاره گره آخرین گره نیز برابر **\0** هست

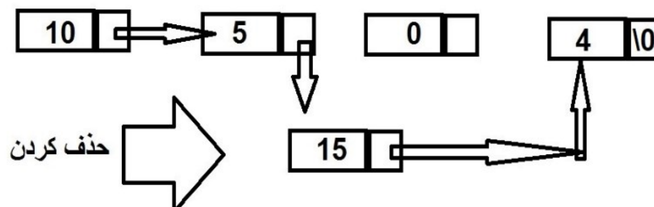
اضافه کردن به لیست یک طرفه:

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

ساختار تولید node

```
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.

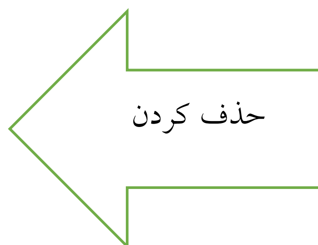
```
delete(x);
temp=head
while(temp.link != x)
temp=temp.link;
```

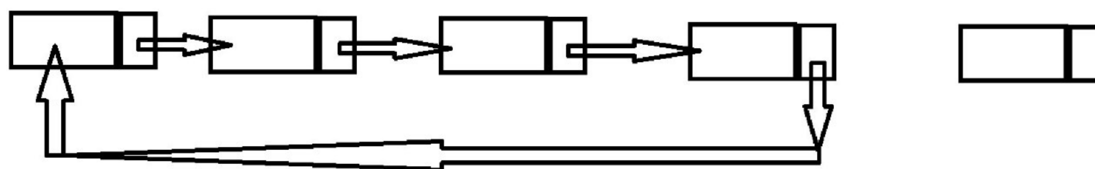
لیست حلقوی یک طرفه:

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.

```
x=head
Temp=head
While(temp link!=null)
{
Temp=temp.link;
}
if(x.link==null)
{
Temp.link=x.link
}
delete(x);
```

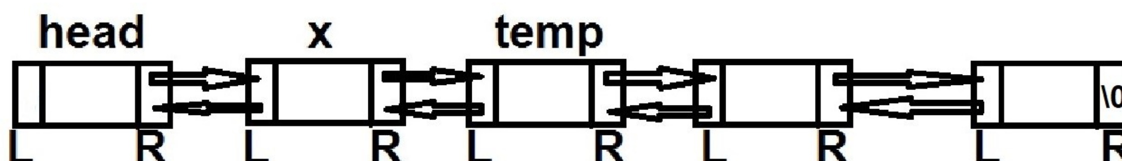




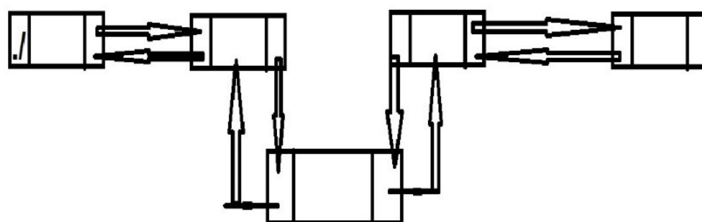
لیست پیوندی دو طرفه TWL (Two Way List):

در لیســت دو طرفه یا (TWL) در هر گونه 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:

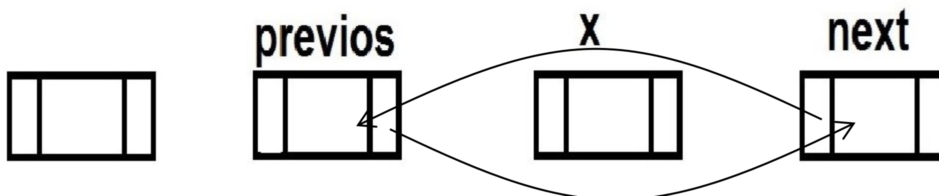


اضافه کردن به لیست دو طرفه

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

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

حذف از لیست پیوندی دوطرفه:

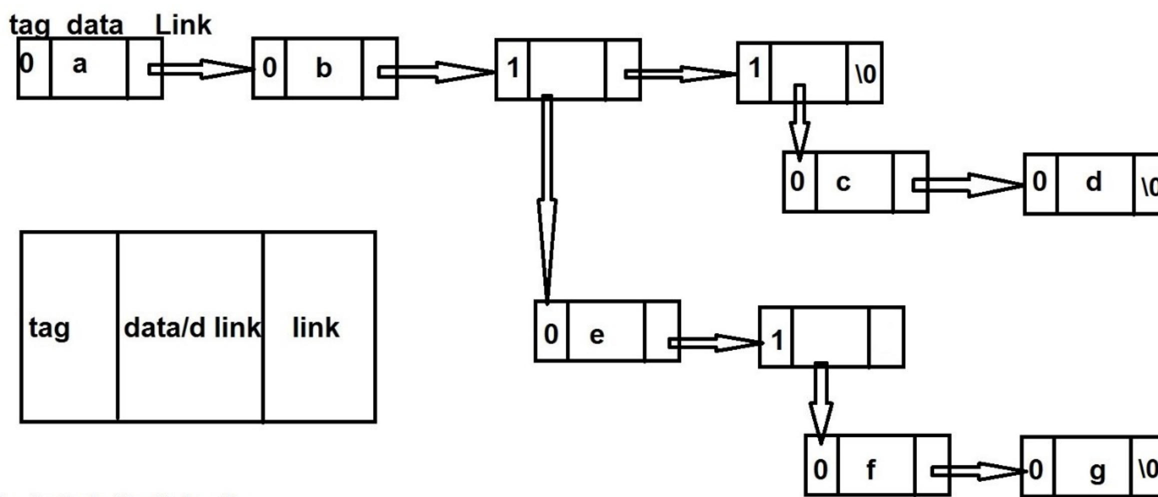


previos.R link=X.R link

next.L link=X.L link

لیست عمومی:

لیستی که هر گونه آن بتواند خود یک زیر لیست باشد لیست عمومی نامیده می شود.به عبارتی دیگر لیست عمومی (L) از تعداد محدودی عنصر A1,A2,...An تشکیل شده به گونه ای که هر(Al) یک عضو ساده و یا یک لیست میباشد.



فصل ۵

درخت یا tree

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

1)دارای گره خواصی به نام ریشه است.

2)بقیه گره ها به n>=0 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.

تعاریف

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.

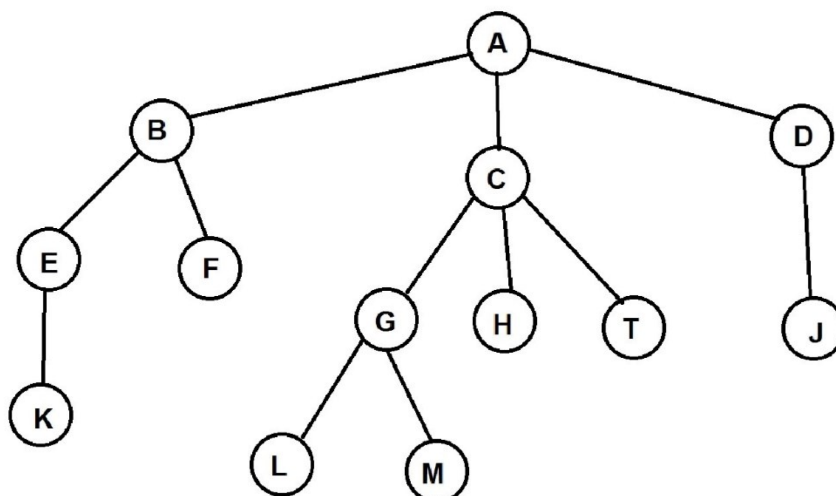
گره های هم ذات: فرزندان یک گره هم ذات نامیده می شود.

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 n+۱ قرار دارند. اگر گره ای در سطح k باشد فرزندان آن در سطح k+۱ 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 مسـیر: هر خط یا اتصـال از یک گره به گره دیگر یا هر انشـعاب 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 مشـابه: درخت های t1,t2 مشـابه هسـتند اگر دارای سـاختمان داده یکسـان باشـند یا به بیان دیگر شـکل مساوی داشته باشند.

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

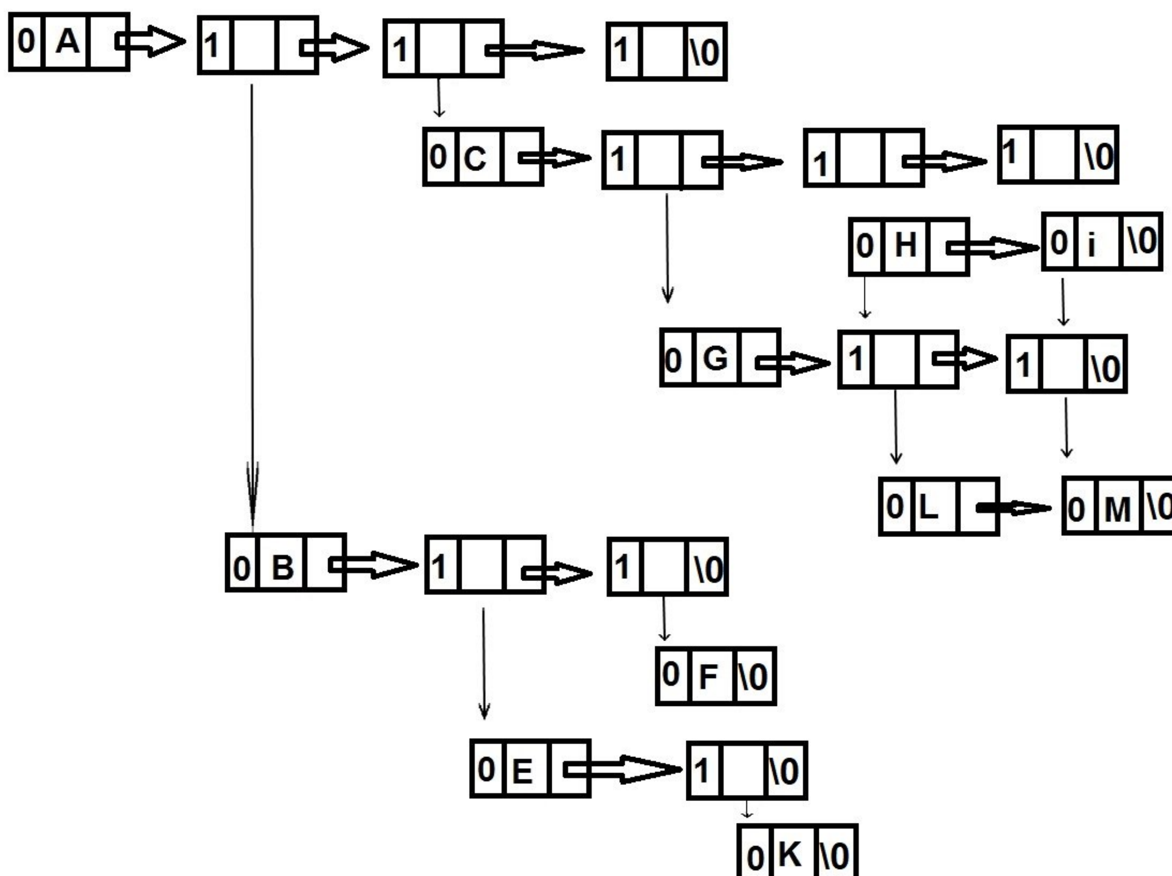
درخت k تایی: درختی که فرزندان هر گره در آن حداکثر k باشد.

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

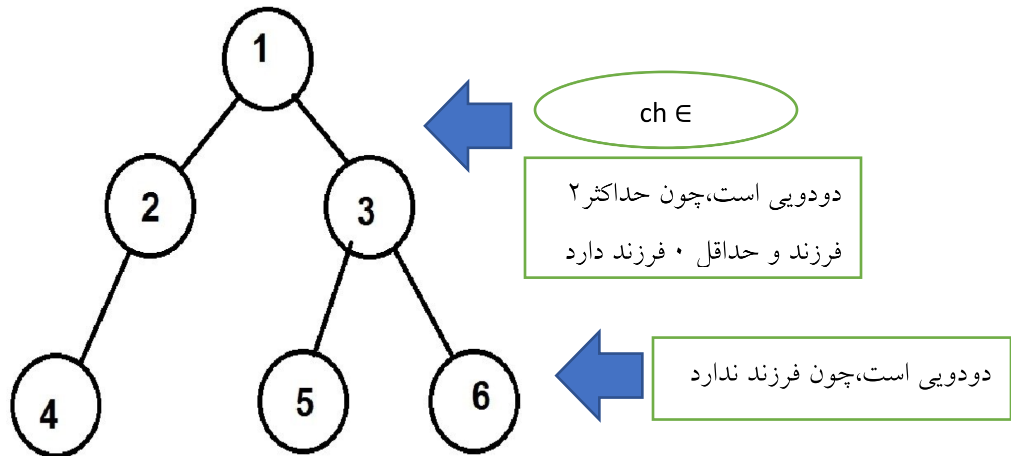
نمایش درخت: برای نمایش درخت می توان از فرم پرانتزی اسـتفاده کرد.در این فرم ابتدا اطلاعات ریشـه و سپس داخل پرانتز ها اطلاعات فرزندان آن گره به ترتیب از چپ به راست نوشته می شود.

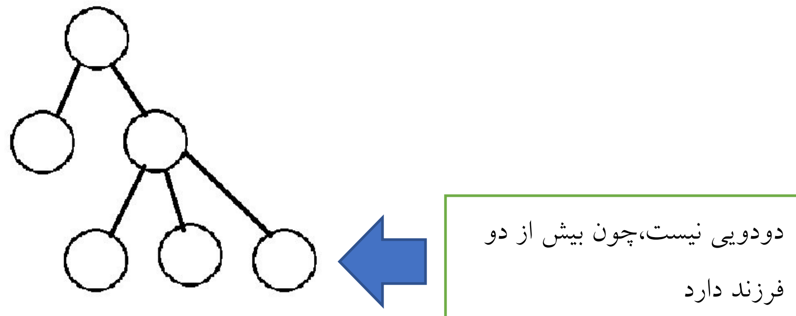
A(B(E(K),F),C(G(L,M)H,T),D(J))

درخت را می توان به صـورت یک لیسـت پیوندی عمومی نیز نمایش داد در این سـطح نمایش فرزندان هر گره در سـمت راست آن ترسیم شـده و هر فرزند که برگ نباشـد با اضـافه کردن یک سـطح به لیسـت به صـورت یک زیر لیست نمایش داده می شود.



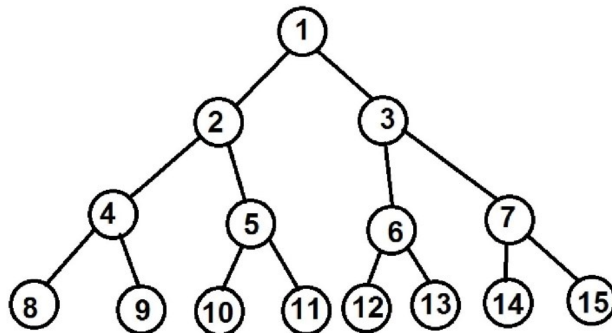
درخت دودویی: یک در خت دودویی یک سـاختمان داده درخت اسـت که در آن هر گره حداکثر دو گره فرزند دارد که فرزنـدان راســـت و چـپ نـامیـده می شـــونـد.در درخـت دودویی،درجـه هر گره حـداکثر می توانـد دو باشـد.درخت های دودویی برای پیاده سـازی درخت جسـت و جوی دودویی و انبوه دودویی و برای جسـت و جوی کارآمد و مرتب سـازی استفاده می شـود.درخت دودویی یک حالت خاص از یک درخت Kتایی اسـت که در آن K برابر ۲ است.در درخت دودویی ترتیب زیر درخت ها مهم است





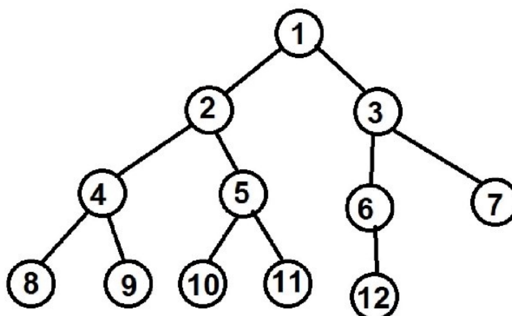
انواع درخت دودویی:

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

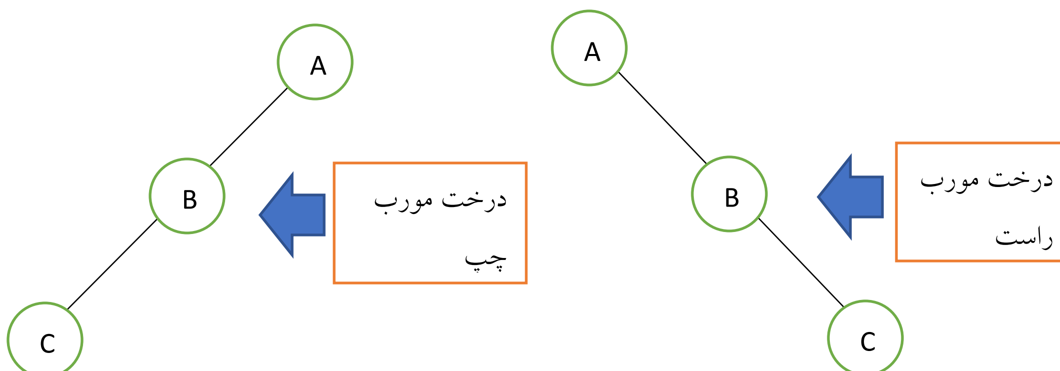


۲)درخت کامل: درختی را کامل گویند که تمام سـطوح آن به جز احتمالا سـطر آخر حداکثر تعداد گره های ممکن را داشته باشد. همچنین تمام گره های سطر تا حد ممکن در سمت چپ و دور ترین مکان آن باشد.

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.

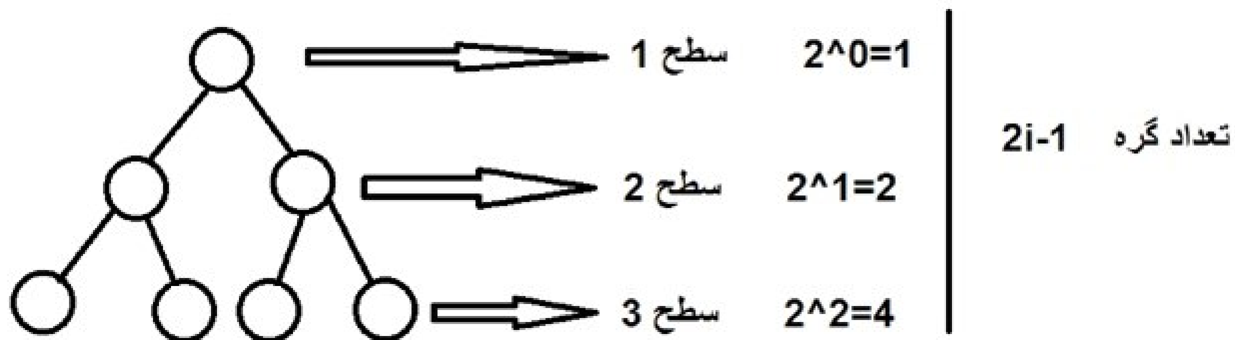


فورمول های مربوط به درخت دودویی:

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 با 2D-1 می باشد.(D<=1)

در هر درخت تعداد یال ها برابر با n-1 است که n تعداد گره هاست.



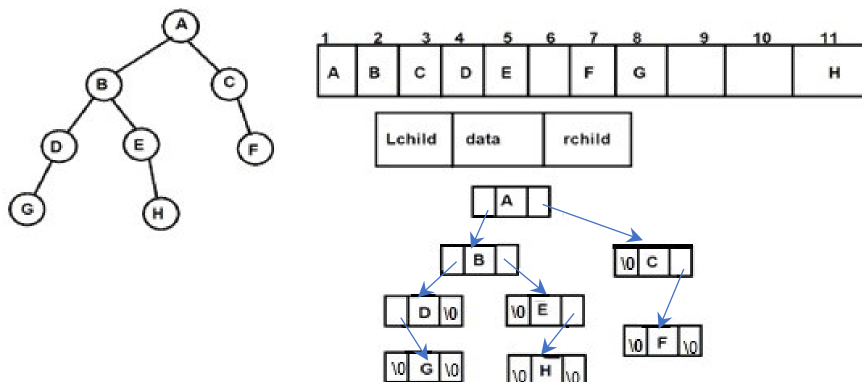
حداکثر کل گره ها = 2^d-1=2^3-1=7

تعداد یال = n-1=7-1=6

نمایش درخت دودویی:

س از 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.

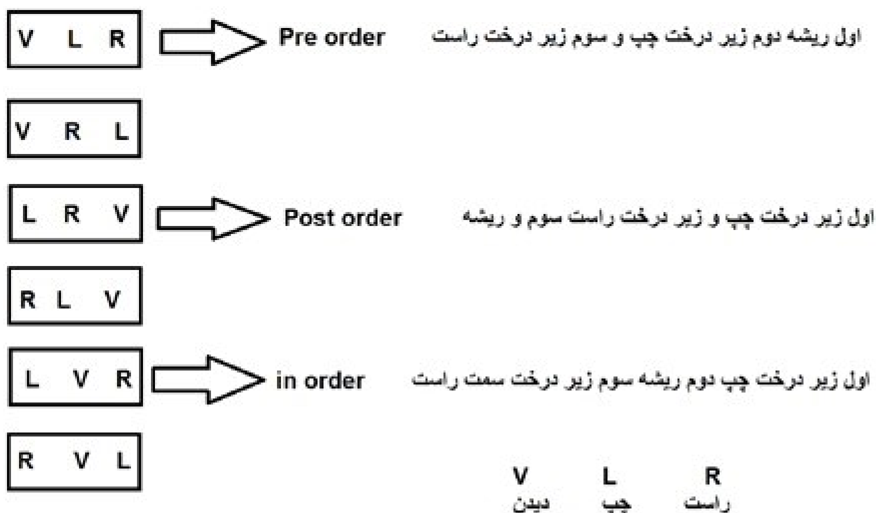
استفاده از فرم آرایه برای درختان کامل مناسب است چون هیچ خانه ای از حافظه هدر نمی رود ولی برای درخت های مورب کاملا نامناسب است چون فضای زیادی به هدر می رود ایراد نمایش درخت با آرایه این است که حذف یا درج گره ها و مستلزم جابه جایی گره هاست که خود باعث شماره سطر گره ها می شود و مشکل فوق با استفاده از نمایش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



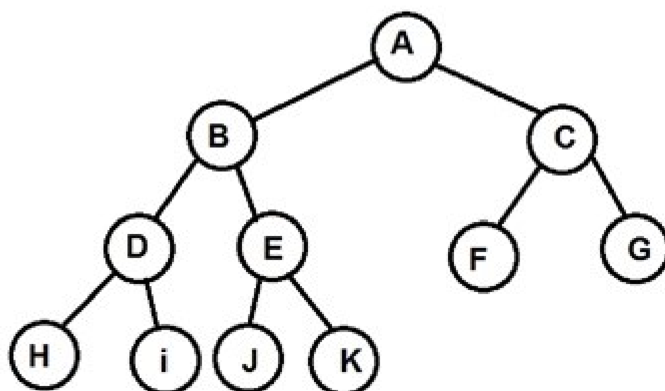
پیمایش درخت دودویی:

در پیمایش درخت می خواهیم هر گره درخت فقط یک بار ویزیت شـــود 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 راسـت باشـد) آنگاه ٦ ترکیب به دست خواهد آمد که سه ترکیب آن برای مهم است.



مثال



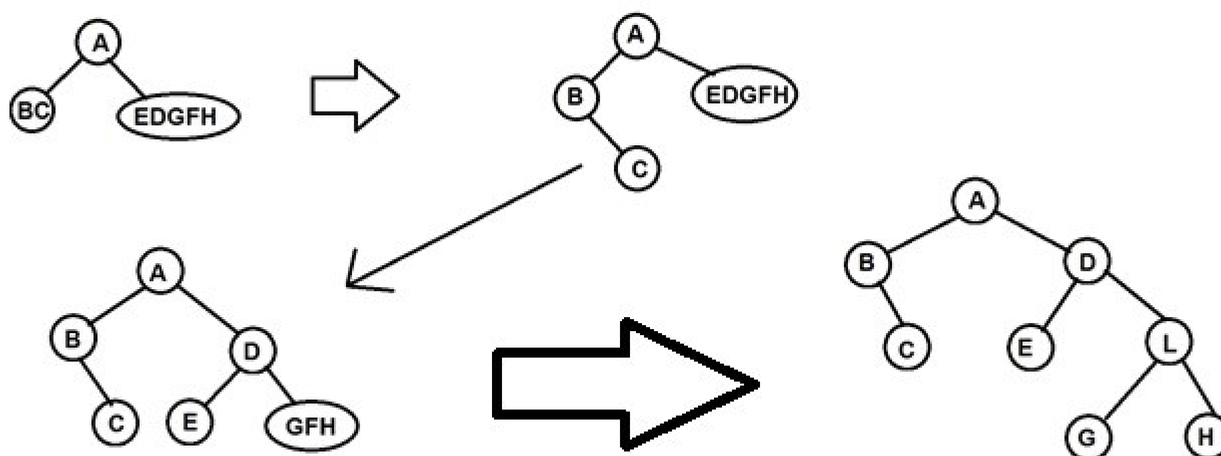
Pre order= ABDHiEJKCFG
Post order= HIDJKEBFGCA
in order= HDIBJEKAFCG

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.

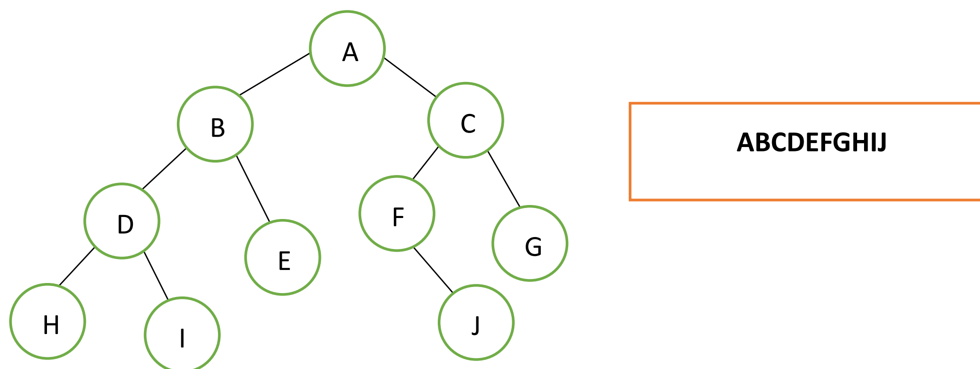
۲)اگر پیمایش های میانوندی و پیشوندی درخت دودویی را داشته باشیم می توانیم آن درخت را به صورت یکتا رسم کنیم.

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



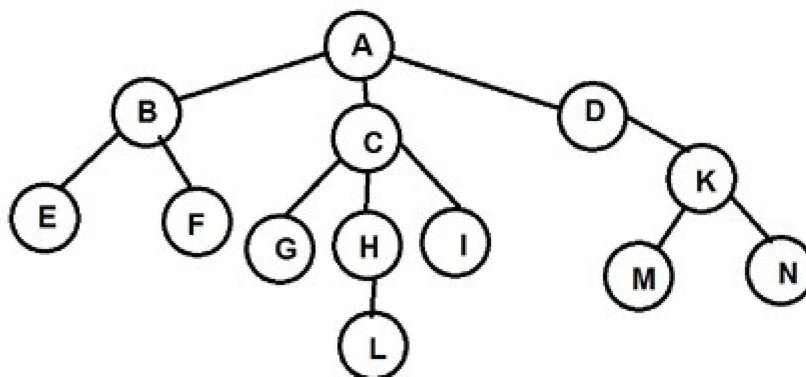
پیمایش ترتیبی سطحی:

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.



درخت عمومی(Genral tree) :

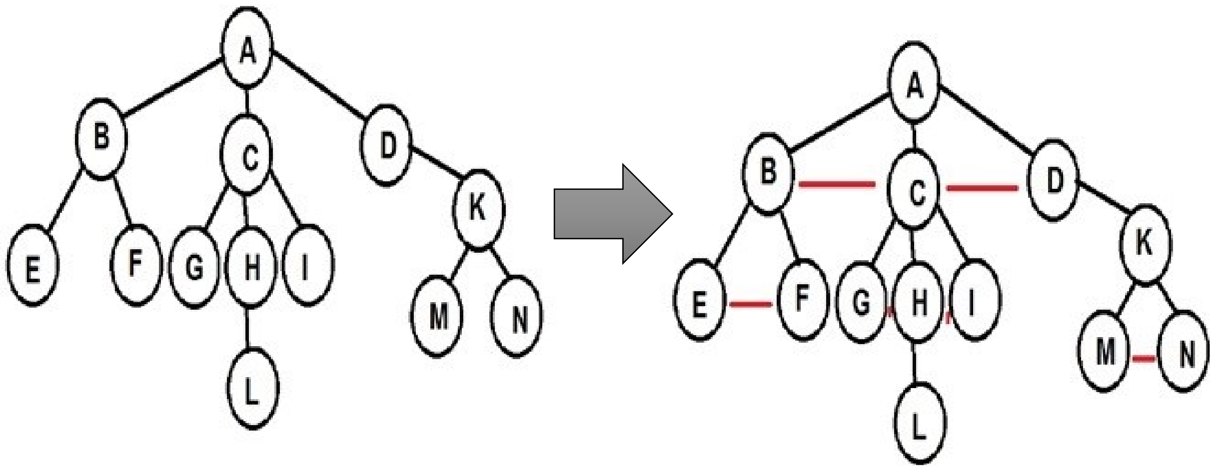
درخت عمومی یا درخت کلی یک درخت Kتایی است که در آن فقط یک گره به نام ریشه با درجه ورودی ۰ وجود دارد و سایر گروه ها دارای یک مکان ورودی هستند از این به بعد فرض می کنیم یک درخت عمومی مرتب است یعنی بچه های هر گره یک ترتیب مشخص دارند هر چند که با این خاصیت همواره برای تعریف درخت عمومی کافی نیست.



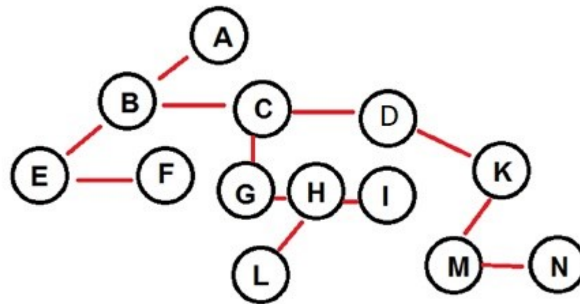
معمولا به دلیل اتلاف زیادی فضای حافظه بهتر است درخت های عمومی را با استفاده از الگوریتم به یک درخت دودویی تبدیل کرد.

تبدیل درخت عمومی به یک درخت دودیی:

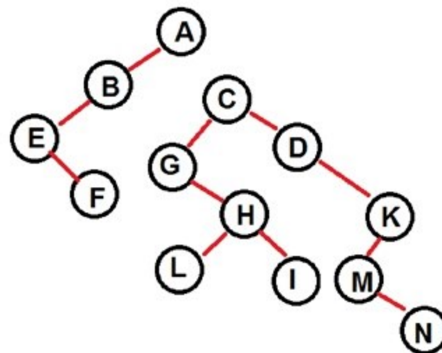
1)در هر سطح گره های کنار هم که فرزند یک پدر هستند را به یک دیگر وصل می کنیم.



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



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



فصل ٦

درختان ویژه

در این فصل 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 B tree 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 :

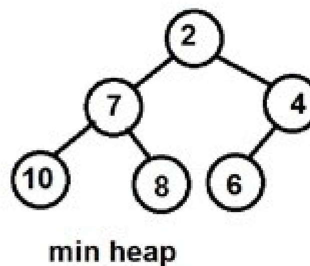
درخت کامل دودویی است که max tree نیز باشد.

تعریف min heap :

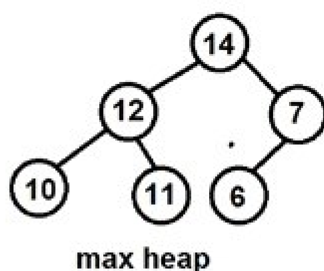
درخت کامل دودیی است که min tree نیز باشد.

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

max tree



min heap

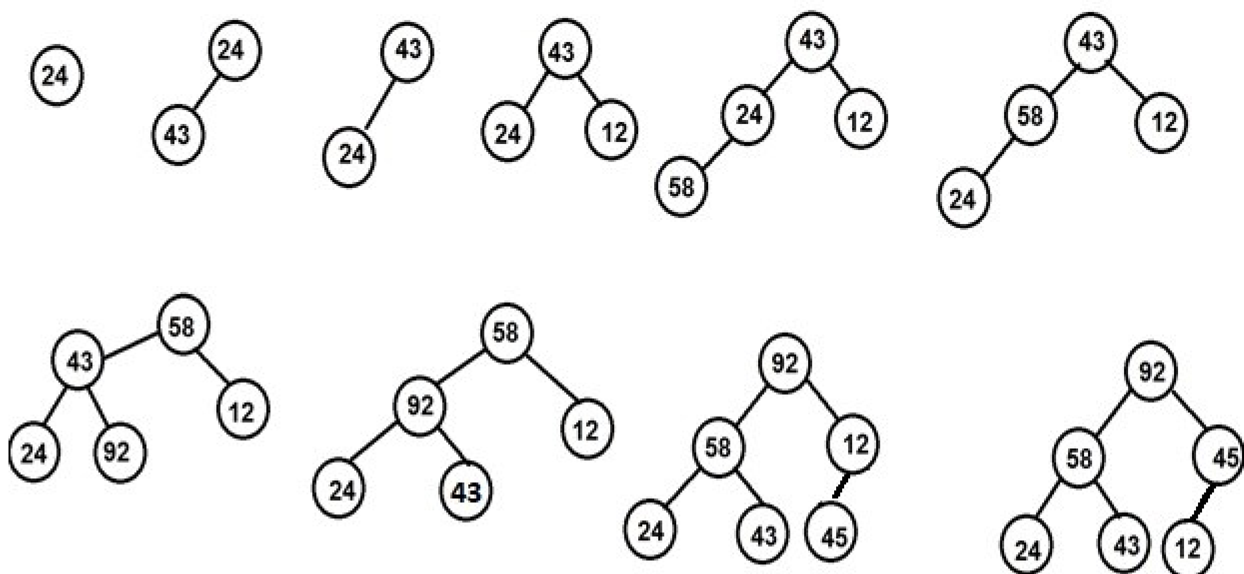


max heap

در اســـتفاده از درختان نیمه مرتب اعمال درج و حذف در درخت به صـــورتی انجام می گیرد که درخت به صورت نیمه مرتب باقی بماند عمل درج به این صورت است که همواره درخت از چپ به راست در هر سطح آخر پر شــده و پس از پر کردن از سـطح بعدی انجام می شـود.هنگام ورود یک گره جدید ابتدا این عنصــر در سـطح آخر(که هنوز پز 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.

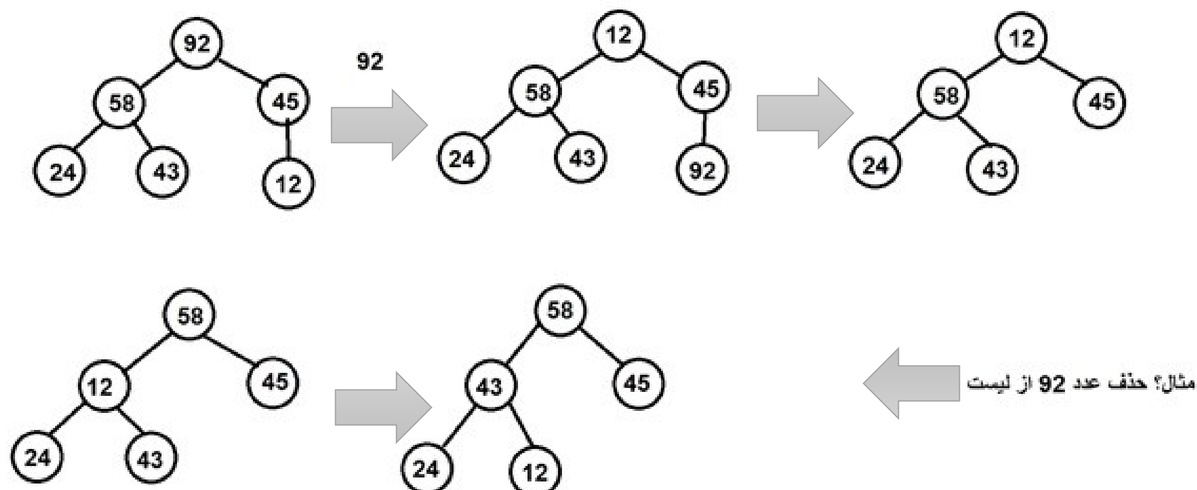
مثال؟

اگر اعداد ورودی از چپ به راست به صورت زیر باشد درخت max heap آن را رسم کنید.



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 میشـــود در تنظیم max heap عنصـــر ریشـــه 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 در 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:

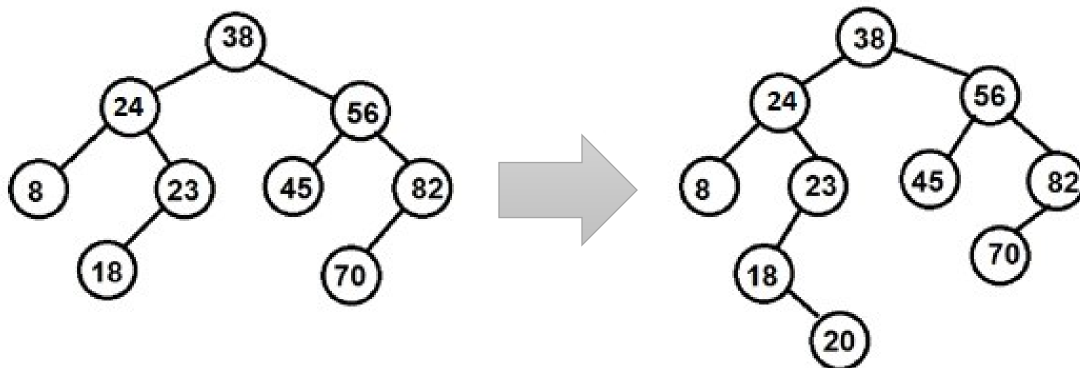
۱) x از گره ریشه کوچکتر است در این حالت هیچ عنصری در زیر درخت سمت راست وجود ندارد که مقدار کلید آن برابر با x باشد بنابراین جست و جو باید در زیر درخت سمت چپ ادامه یابد

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.

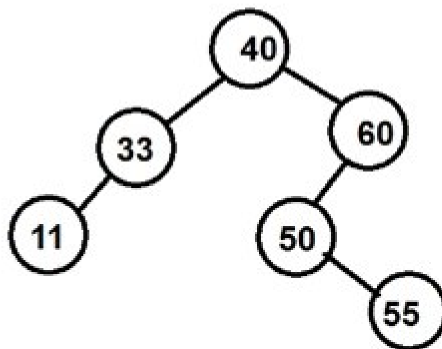
مثال؟

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



مثال؟

۱۱و۵۵و۳۳و۵۰و۶۰و۴۰



حذف یک عنصر از BST :

برای حذف یک گره دلخواه از درخت BST ابتدا باید عمل جست و جو انجام گیرددر 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.

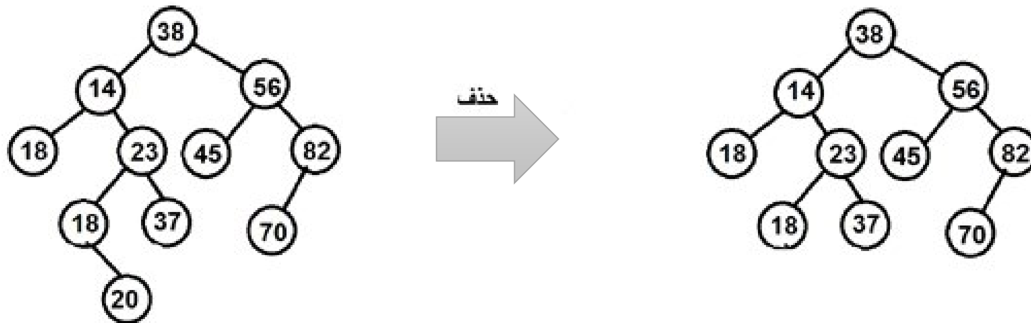
ب) گره مورد نظر برای حذف از درخت BST وجود دارد که برای این حالت ۳ وضعیت وجود خواهد داشت.

۱) عنصر x برگ است.

۲) عنصر x یک فرزند دارد.

۳) عنصر x دو فرزند دارد.

حذف عدد ۲۰ از درخت BST :



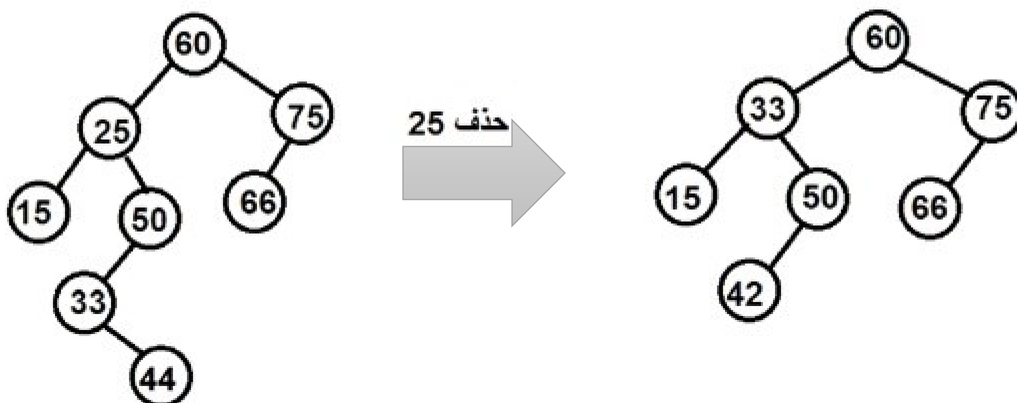
حالت ۲ :

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

حالت۳ : 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 چپ را جایگزین کینم.



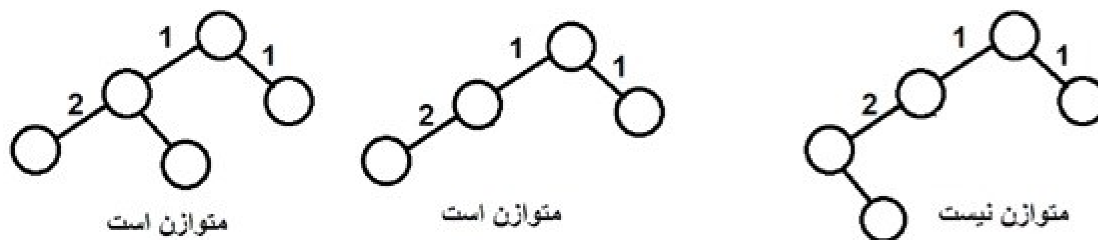
پیچیدگی الگوریتم:

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 بنا بر این زمان درج در BST برای این گره در حالت متوسط و بهترین حالت از مرتبه O(n log n) و در بدترین حالت O(n2) می باشد و زمان حذف نیز مشابه زمان درج است.

زمان درج در درخت heap برای هر عنصر On(log 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.

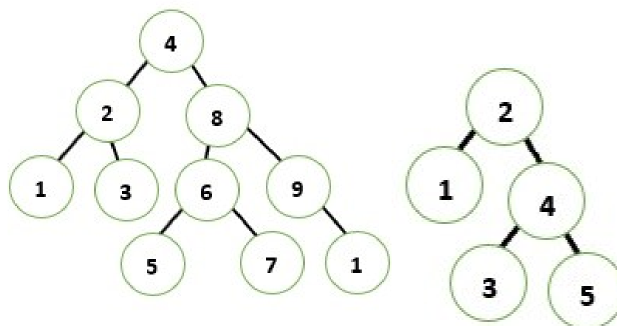


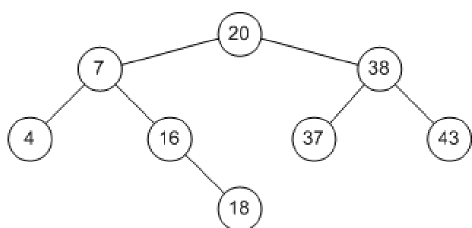
درخت AVL (ADELSON , VELSKII , LANDIS) :

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 نماد 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-2) + AVL(H-1) + 1

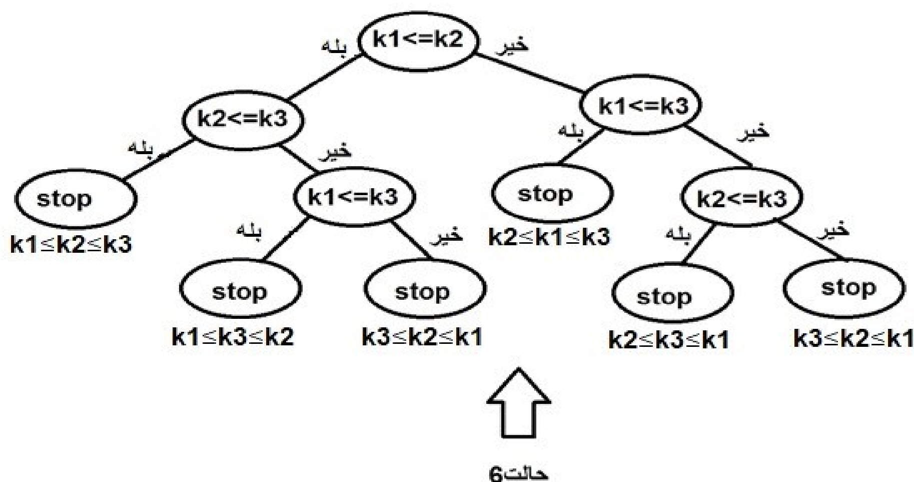
مثال؟

در صورتی که حداقل گره های لازم برای ساختن ۲ درخت با ارتفاع متوازن ۳و٤ به ترتیب ٤و٧ باشد حداقل گره های لازم برای ساختن درخت با ارتفاع متوازن ۵ کدام است؟

AVL(5) = AVL(5-2) + AVL(5-1) + 1 = AVL(3) + AVL(4) + 1= 4 + 7 + 1 = 12

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.



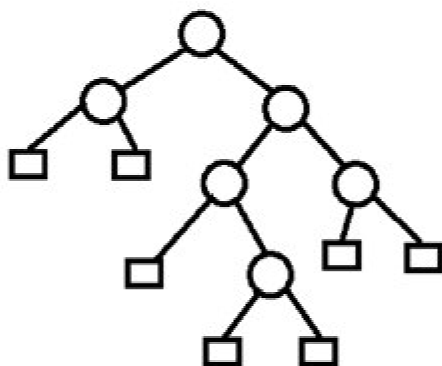
بررسی سه عدد نسبت به هم دیگر !3 نسبت به !6 دارد

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 عناصر ارتفاع درخت تصمیم برای حالت مرتب عناصر O log n بودن که این بهترین حالت 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 0 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+1　　　　　7 = 6+1

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.

$$le = 2 + 2 + 3 + 4 + 4 + 3 + 3 = 21$$

$$li = 0 + 1 + 1 + 2 + 2 + 3 = 9$$

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

$$le = li + 2 * ni$$

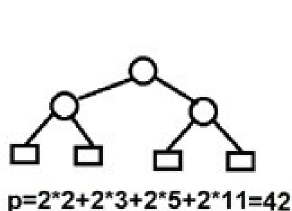
$$9 + 2 * 6 = 21$$

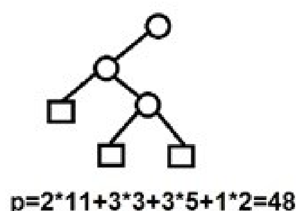
فرض کنید به هر گره خارجی یک وزن داده شده است طول مسیر وزن داده شده خارجی بنا به تعریف مجموع طول های مسیر وزن داده شده است و به صورت زیر تعریف می شود.

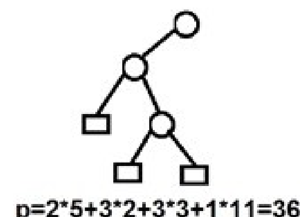
$$P = W1\ L1 + W2\ L2 + Wn\ Ln$$

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

مثال؟







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 اینجت 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.

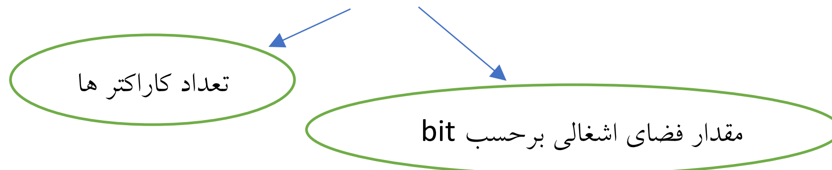
مثال؟

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					

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

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



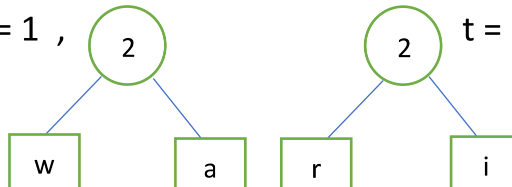
w = 1 , a = 1 , r = 1 , i = 1 , s = 1 , g = 1 , d = 1 , . = 1 , t = 2 , h = 2 , o = 2 , e = 3 , space = 3

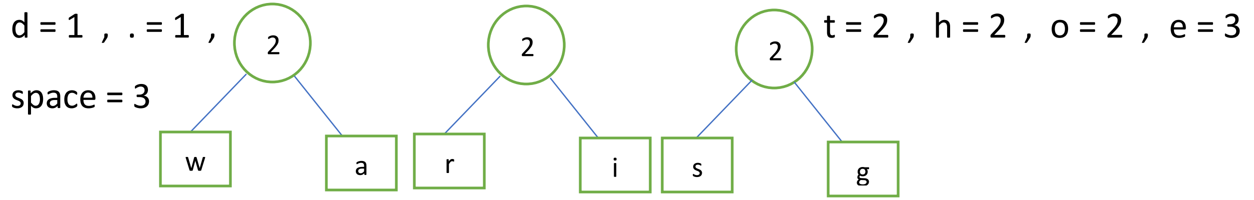
r = 1 , i =1 , s = 1 , g = 1 , d = 1 , . = 1 , 2 , t = 2 , h = 2 , o = 2 , e = 3

space = 3

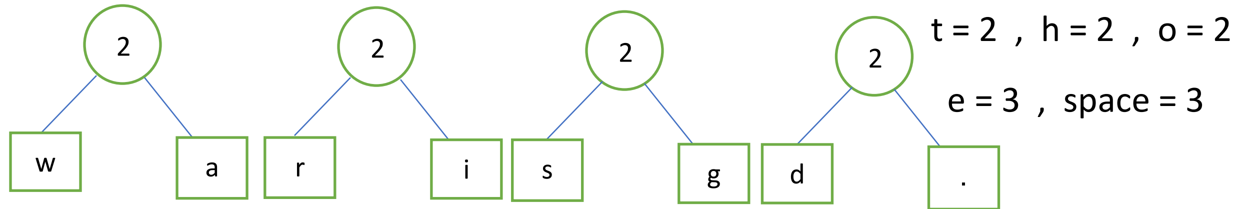
s = 1 , g = 1 , d = 1 , . = 1 , 2 2 t = 2 , h = 2 , o = 2 , e =3

space = 3

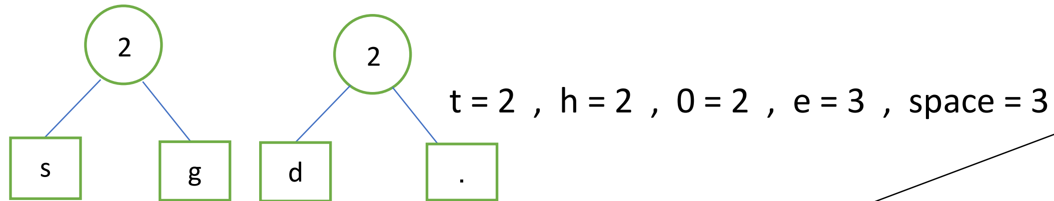




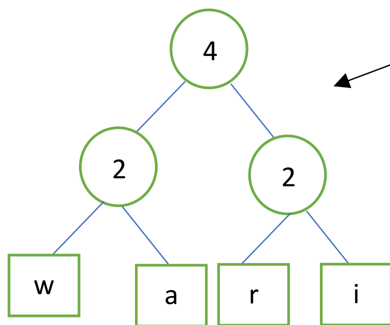
d = 1 , . = 1 ,
space = 3
t = 2 , h = 2 , o = 2 , e = 3

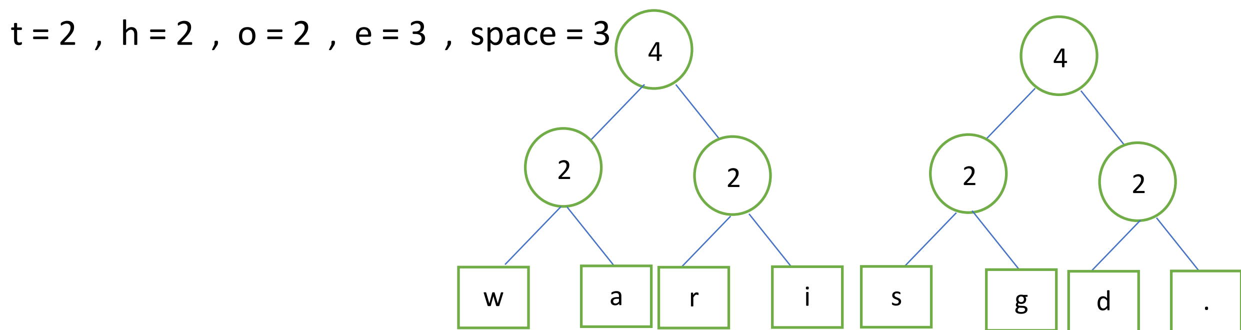


t = 2 , h = 2 , o = 2
e = 3 , space = 3



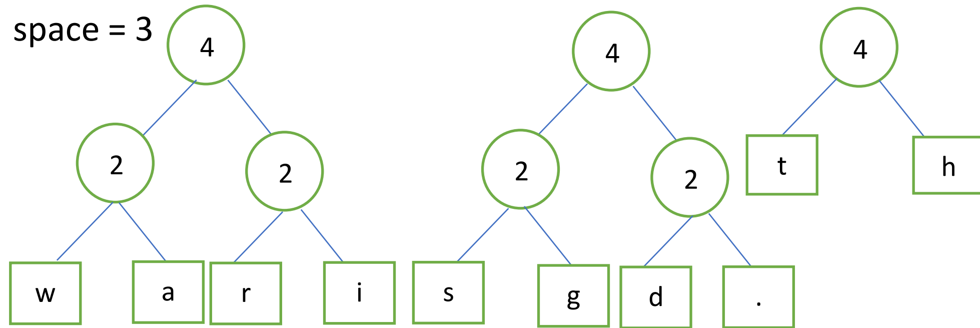
t = 2 , h = 2 , 0 = 2 , e = 3 , space = 3

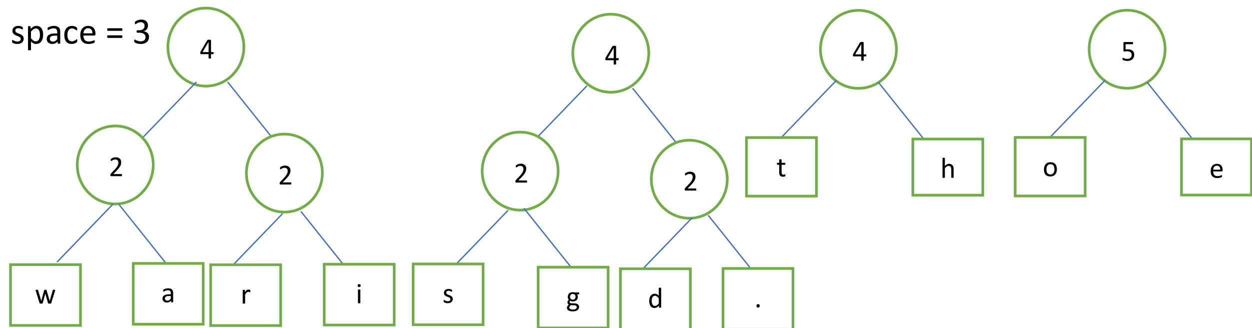


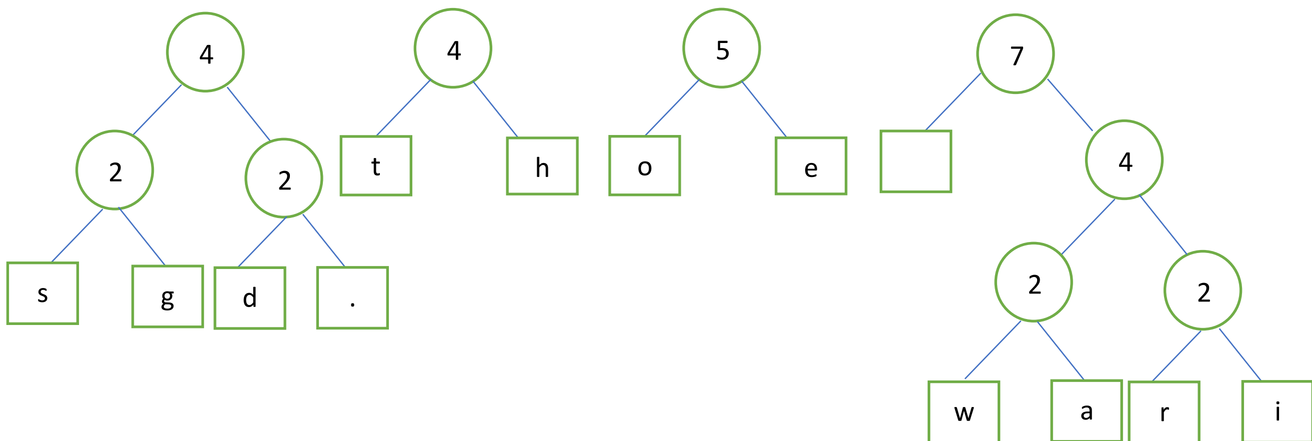


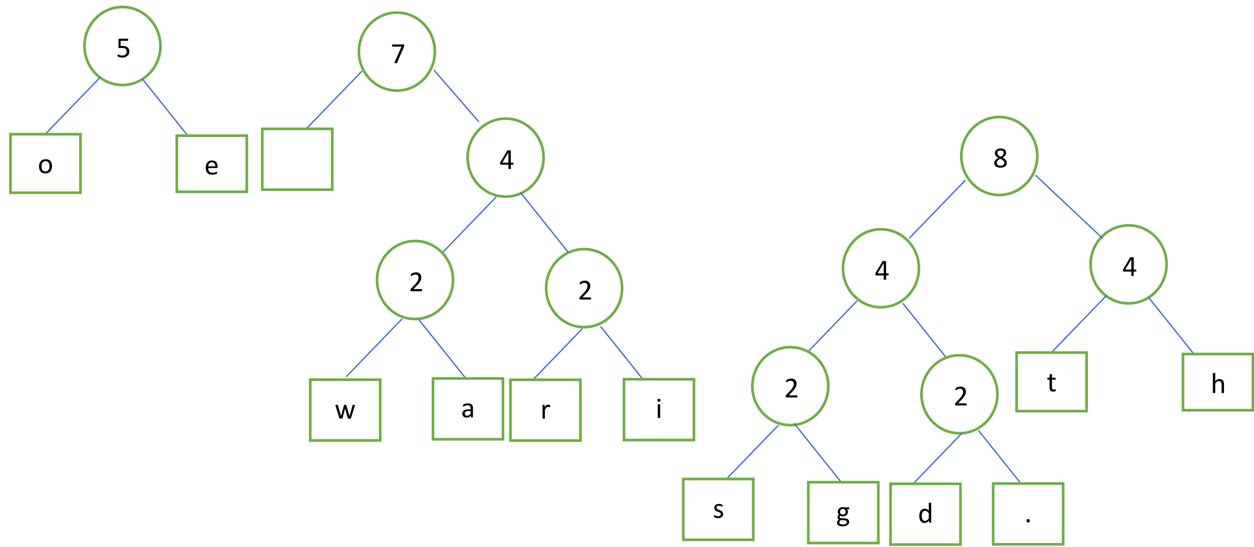
t = 2 , h = 2 , o = 2 , e = 3 , space = 3

o = 2 , e = 3 , space = 3

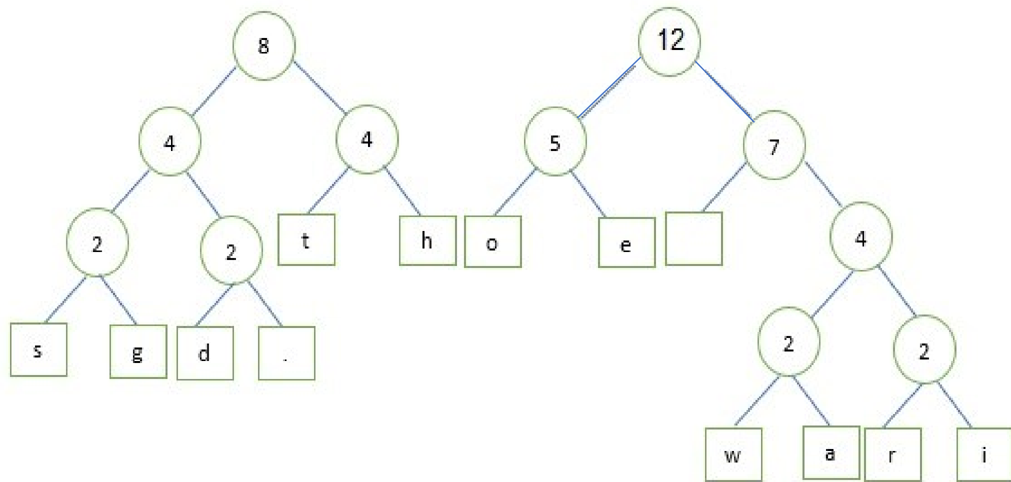




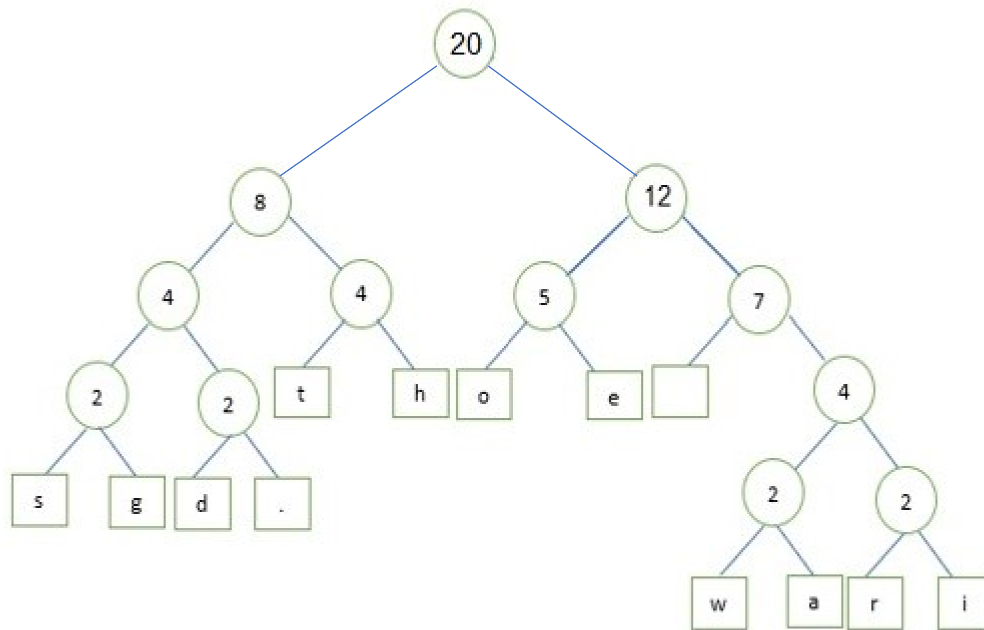


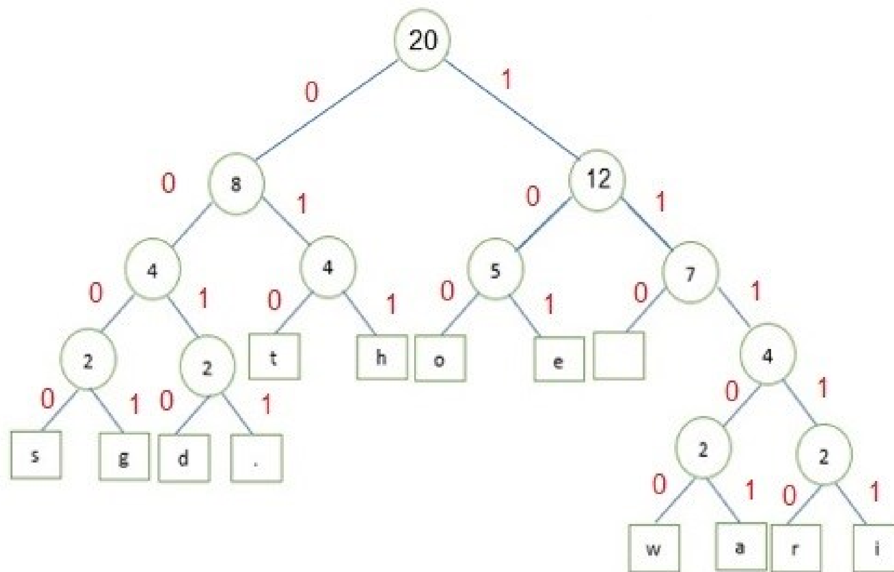


5
o
e
7
4
2
2
w
a
r
i
8
4
4
2
2
t
h
s
g
d
.



8
12
4
4
5
7
2
2
t
h
o
e
4
s
g
d
.
2
2
w
a
r
i





t = 010(2)	h = 011(2)	e = 101(3)	w =11100	a = 11101	r = 11110
i = 11111	s = 0000	g = 0001	o = 100(2)	d = 0010	space = 110(3)
. = 0010					

$$6+6+9+5+5+5+5+4+4+6+4+9+4=72$$

مقدار فضای اشغالی بعد از الگوریتم هافمن از ۱۶۰ بیت به ۷۲ بیت کاهش یافت

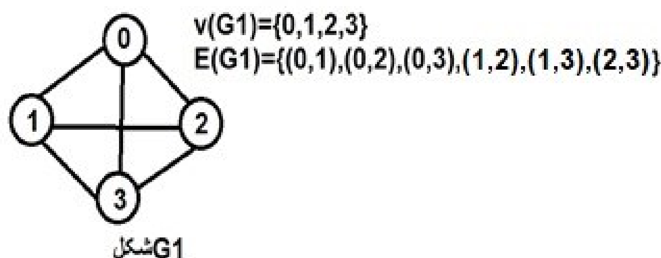
فصل ۷

گراف

هر گراف (g) شامل دو مجموعه V , E است. V مجموعه ای محدود از رئوس و E مجموعه ای محدود از لبه هاست.

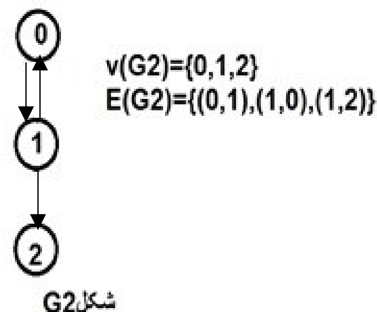
E(g) و V(g) مجموعه رئوس و لبه های گراف را نشان می دهد هر گراف به صورت G(V,E) نمایش داده می شود.در گراف دو وضعیت وجود دارد، وضعیت جهت دار و غیر جهت دار که در وضعیت غیر جهت دار رئوس زوج مرتب نیستند بنابراین زوج های (v0,v1) , (v1 , v0) یکسانند اما در گراف جهت دار این دو زوج دو لبه متفاوت را نمایش می دهد.گراف حداقل یک راس دارد و نمیتواند کاملا تهی باشد.

گراف غیر جهت

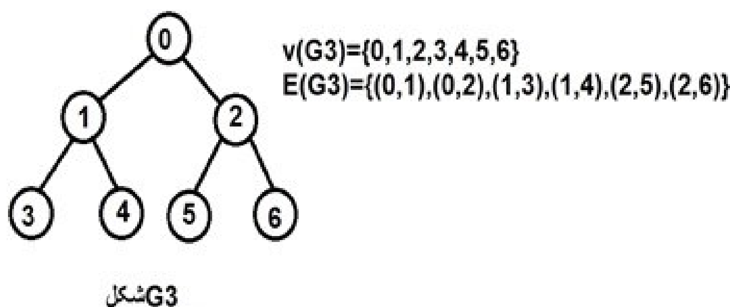


شکلG1

گراف جهت دار



شکلG2



شکلG3

گراف تهی یا پوچ

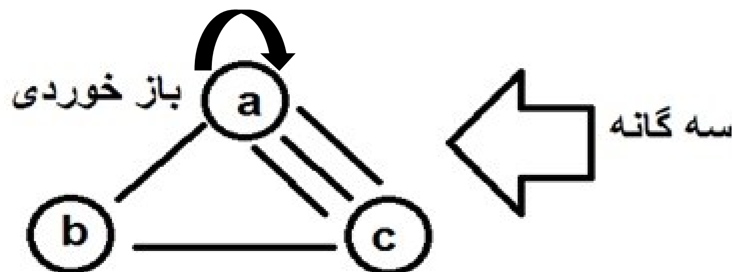
گراف که فقط شامل گره های منفرد است یا به عبارت دیگر مجموعه یال آن تهی است درخت را می توان حالت خاص از گراف در نظر گرفت که در آن حلقه وجود ندارد.

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.

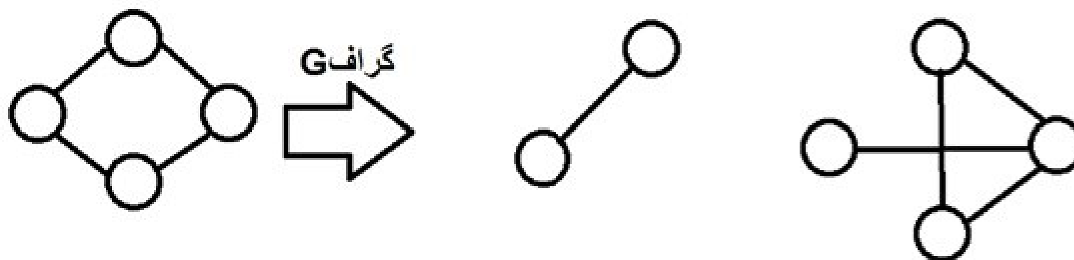
گراف کامل

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 n(n-1)/2 است.

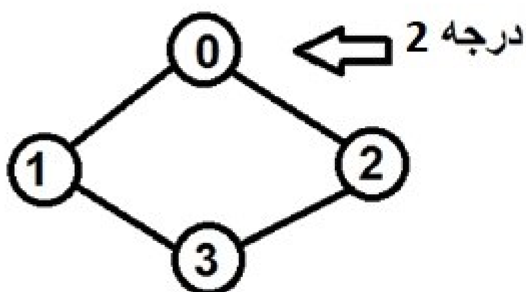
نکته: برای یک گراف جهت دار با n راس حداکثر تعداد لبه ها برابر با n(n-1) است.

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

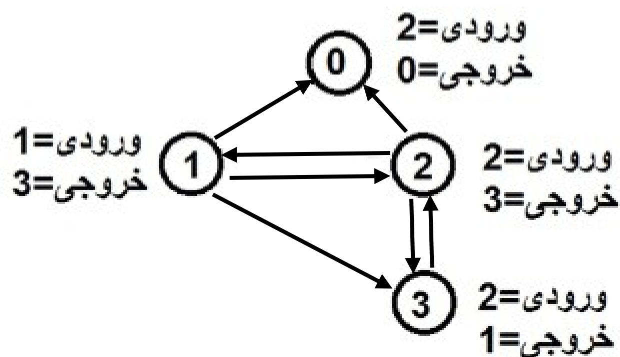


تعریف طول مسیر: تعداد لبه های موجود در مسیر را طول مسیر می گویند.

درجه۱ راس: در یک گراف بدون جهت تعداد لبه ها متقاطع با آن راس است.

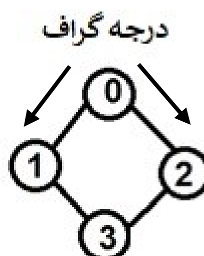


در گراف جهت دار هر راس دارای درجه ورودی و خروجی اســـت. تعداد یال هایی که به راس i وارد می شود،درجه ورودی و تعداد یال هایی که از آن خارج می شود،درجه خروجی آن است .



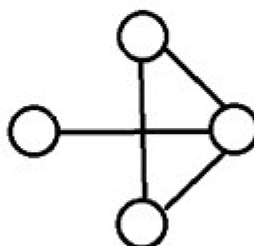
نکته:اگر درجه یک راس عددی فرد باشد آن راس را راس فرد و اگر زوج باشد آن را زوج گویند.

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



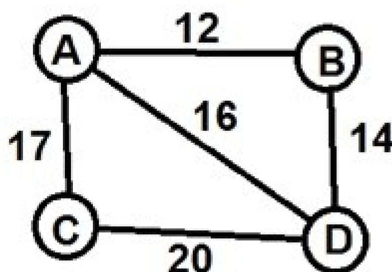
نکته:اگر در گراف (G) با n راس با شماره های ۰ تا (n-0) اگر Di درجه راس i باشد،آنگاه تعداد لبه ها یعنی E از فورمول زیر بدست می آید

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

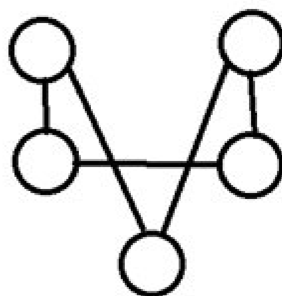


$$\frac{1}{2}(2+1+3+2)=\frac{1}{2}(8)=4$$

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.



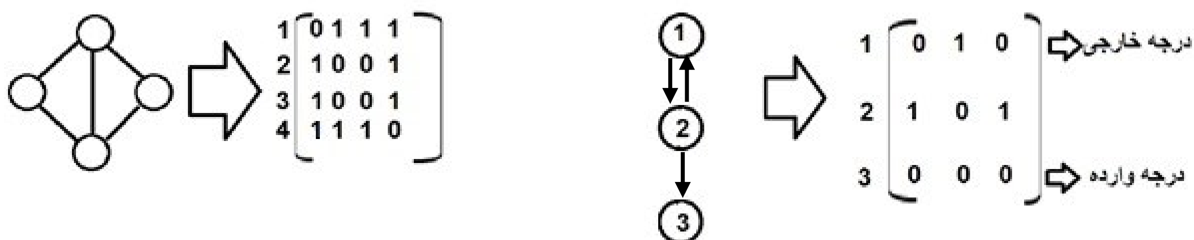
نمایش گراف: دو روش برای نمایش گراف وجود دارد

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 به di و vg در EG باشد عنصر خانه i,G برابر با i 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.

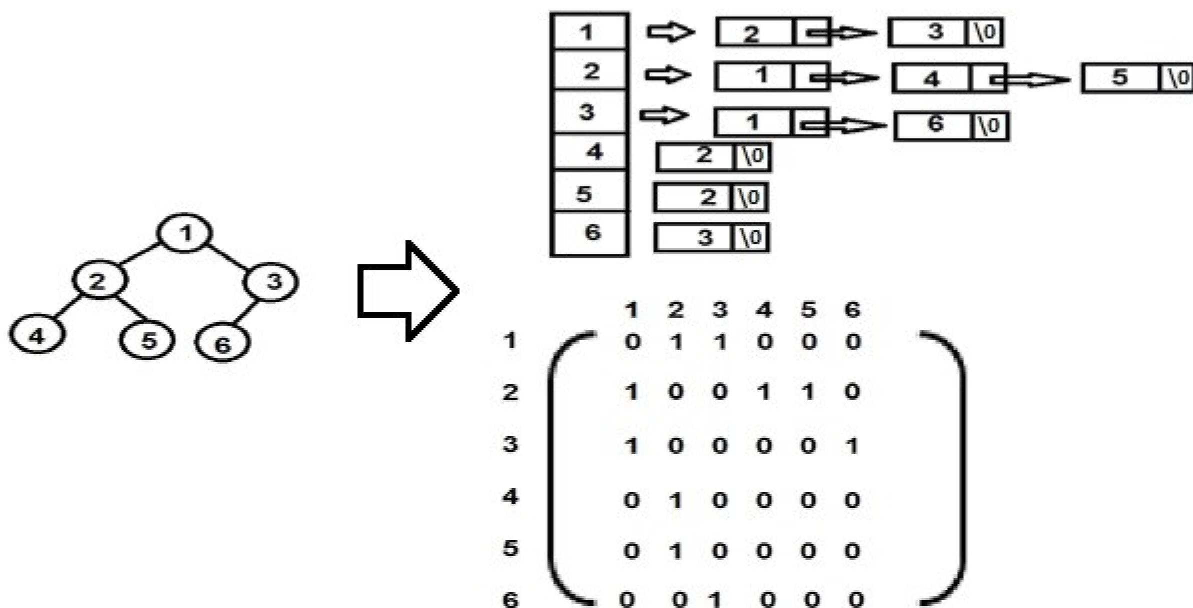
تعداد یک ها در ماتریس مجاورتی بدون جهت دو برابر تعداد یال ها و در گراف جهت دار برابر تعداد یال هاست.

لیست مجاورتی:

در این روش برای هر راس از گراف G یک لیست وجود دارد هر گره حداقل دو فیلد دارد.(راس و اتصال).

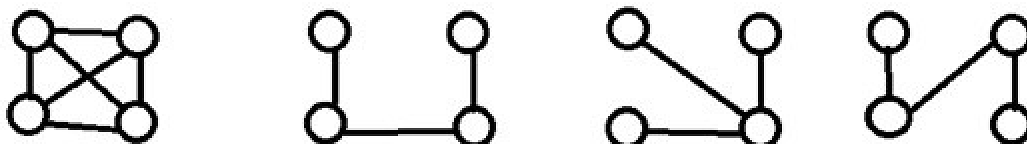
در هر لیست مشخص مانند i گره های لیست حاوی دو راس مجاور از راس i می باشد.

هر لیست یک گره هد دارد که به ترتیب شماره گذاری شده اند و این امر دستیابی سریع به لیست های مجاورتی برای راس خواص را به آسانی امکان پذیر می کند.نمایش لیست مجاورتی گراف زیر بدین صورت خواهد بود.



درخت پوشا یا (Spanning Tree):

درختی که تعدادی از لبه ها یا همان(یال ها) که تمام حروف(G)را در بر دارد که درخت پوشا نامیده می شود که یک گراف و سه درخت پوشای آن را در زیر مشاهده می کنید.



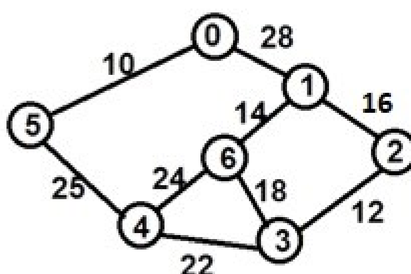
درخت پوشا(Minimum):

هزینه یک درخت پوشـــای یک گراف جهت دار دارای وزن مجموعه هزینه ها یعنی وزن های لبه ها در درخت پوشـا می باشـد.درخت پوشـای Minimum درختی اسـت که دارای کم ترین هزینه باشـد برای بدسـت آوردن درخت پوشـایMinimum،الگوریتم راشـال کروسـکال والگوریتم سـولین و 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

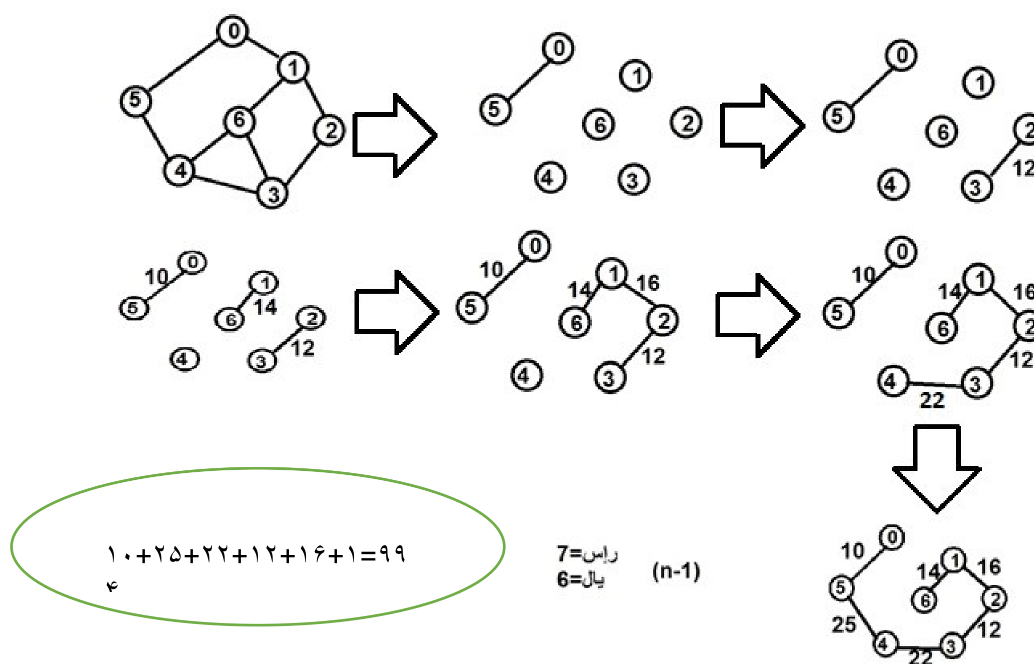
۲)باید دقیقا از n-1 از لبه استفاده کنیم.(تعداد گره=n)

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



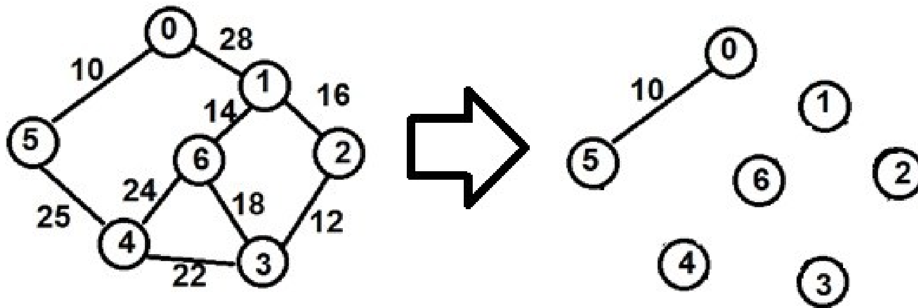
الگوریتم راشال(کروسکال):

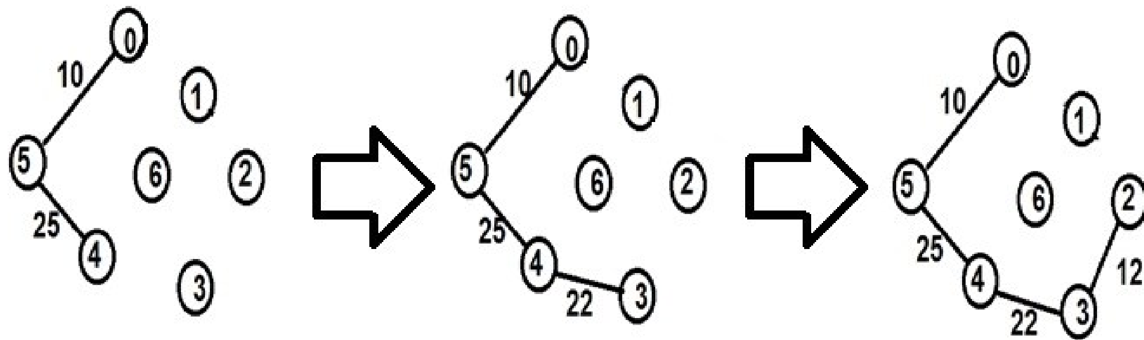
در این روش درخت پوشـــا با کمترین هزینه (+) لبه به لبه ســـاخته می شـــود لبه های مورد اســـتفاده در (+) به ترتیب سعودی وزن ها می باشد.یک لبه در (+) اگر با لبه های قبل که در (+) بودند تشکیل حلقه ندهد.

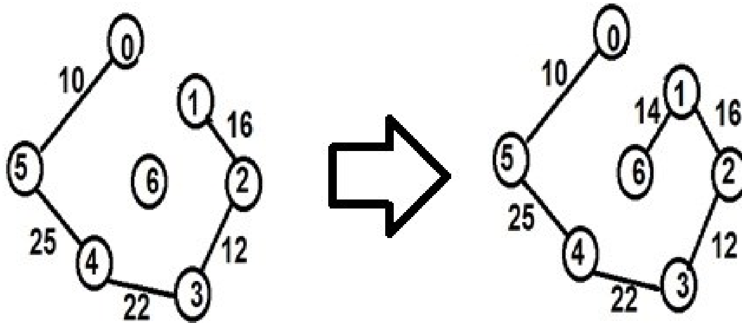


الگوریتم پریم(prim):

الگوریتم پریم همانند راشال(کروسکال) است با این تفاوت که در هر مرحله مجموعه لبه های انتخاب شده یک درخت را تشـــکیل 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.


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


0
10
1
5
6
2
25
4
3
0
10
1
5
6
2
25
4
22
3
0
10
1
5
6
2
25
12
4
22
3


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

فصل ۹

مرتب سازی

الگوریتم های متعددی برای مرتب سازی n عنصر وجود دارد که برخی از آن ها عبارتند از:

1)مرتب سازی انتخابی(selection sort)

2)مرتب سازی حبابی(bubble sort)

3)مرتب سازی درجی(insertion sort)

4)مرتب سازی جا به جایی(exchange sort)

5)مرتب سازی سریع(quick sort)

6)مرتب سازی هرمی(heap sort)

7)مرتب سازی ادغامی یا ترکیبی(merge sort)

8)مرتب سازی درختی(tree sort)

9)مرتب سازی مبنا(radius sort)

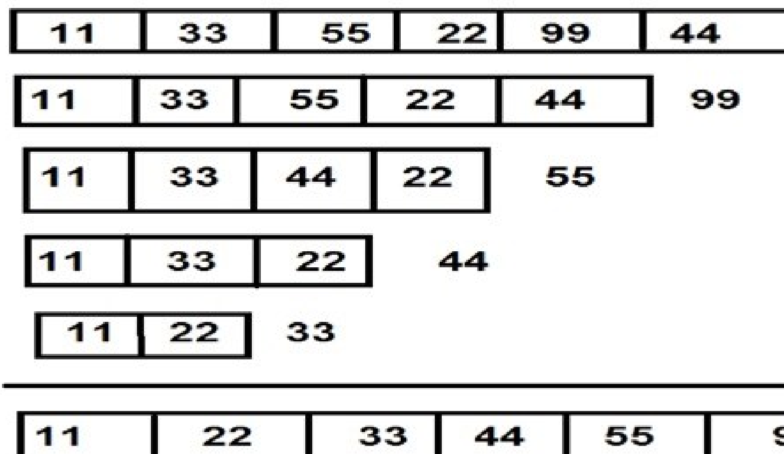
10)مرتب سازی پوسته(shell sort)

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:

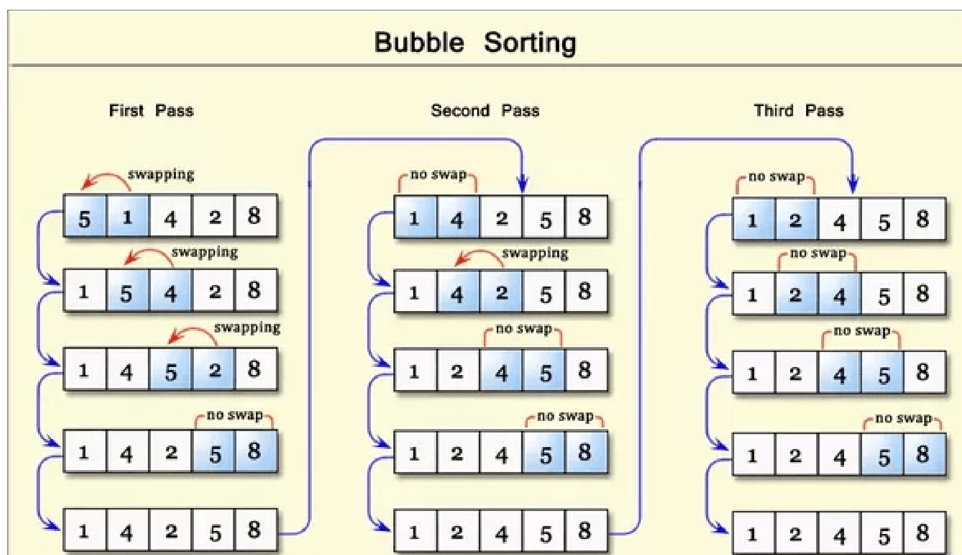
```
for(i=0 ; i<n ; i++)
{
max=x[i];
index=1;
for(j=0 ; j<n ; j++)
{
if(x[j] > max)
{
max=x[j];
index=j;
x[index]=x[i];
x[i]=max;
}
}
}
```



مرتب سازی حبابی:

در مرتب سازی حبابی باید چندین بار آرایه را پیمایش کرد و هر بار عنصری را با عنصر بعدی خودش مقایسه نمود در صورتی که عنصر اول از عنصر دوم بزرگتر باشد در مرتب سازی سعودی جای آنها را عوض می کنیم.

```
for(i=0 ; i<n ; i++)
{
for(j=i+1 ; j<n ; j++)
{
if(x[i] > x[j])
{
temp=x[i];
x[i]=x[j];
x[j]=temp
}
}
```

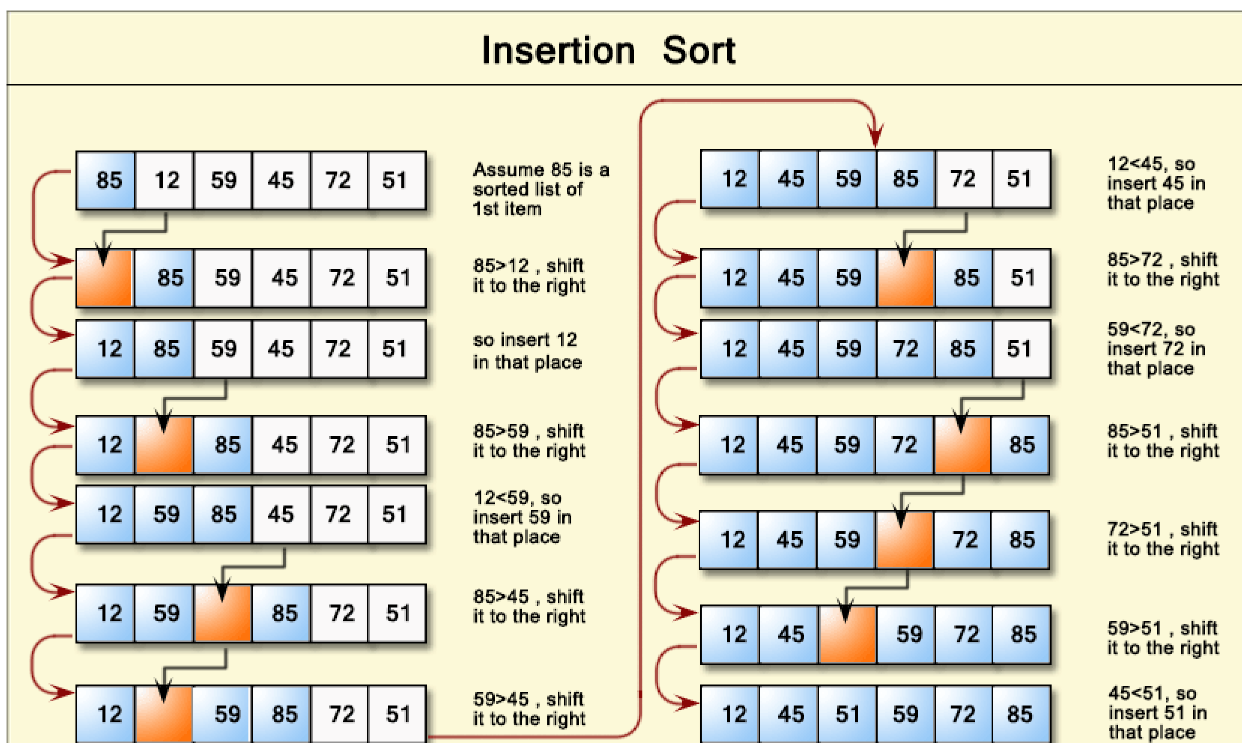


مرتب سازی درجی:

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

تکه کد الگوریتم به صورت زیر می باشد:

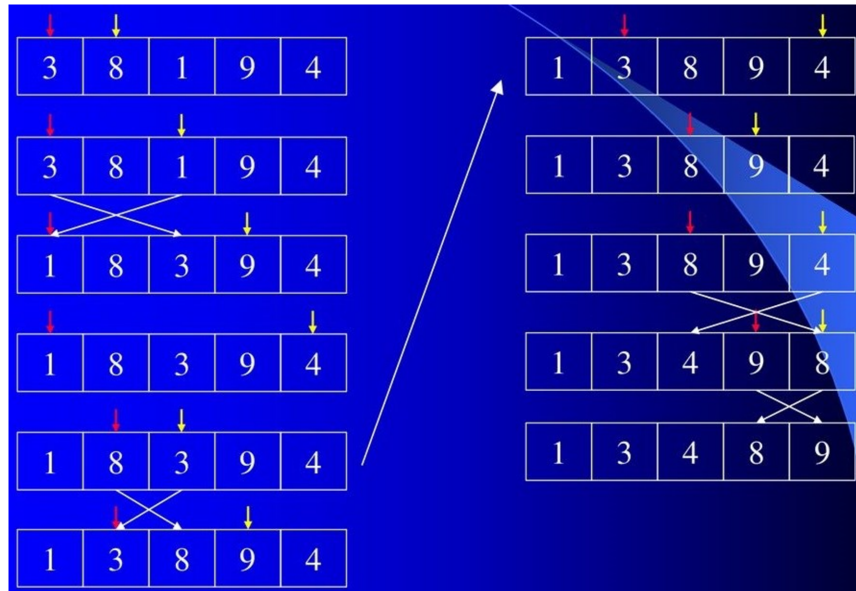
```
insertion sort(x[],h)
{
for(i=2 ; i<n ; i++)
{
Y=x[i];
j=i-1;
while(j>0 && Y<x[j])
{
x[j+1] = x[j];
j=j-1;
}
x[j+1]=Y;
}
```



مرتب سازی جا به جایی (تعویض):

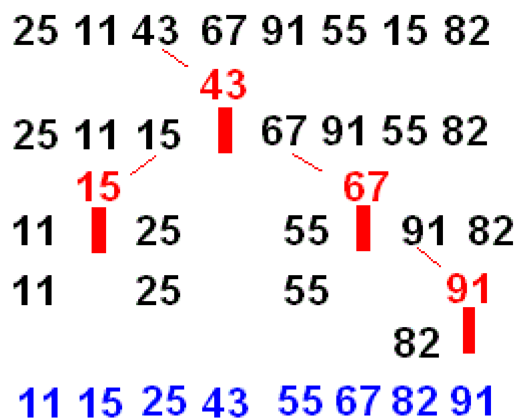
در این الگوریتم ابتدا خانه اول با خانه های دوم تا اخر مقایسه می شود و از هر کدام که بزرگتر بود با آن جا به جا می شود بدین ترتیب که در انتهای گذر اول کوچکترین عدد در خانه اول قرار گرفته است سپس خانه دوم با خانه های سوم تا n مقایسه می شود و عملیات مذکور انجام میگیرد و این روند تا مرحله (n-1) ادامه پیدا می کند.

```
exchange sort (a[n])
{
for(i=1 ; i<=n-1 ; i++)
if(a[i] > a[j])
swap(a[i] , a[j])
}
```

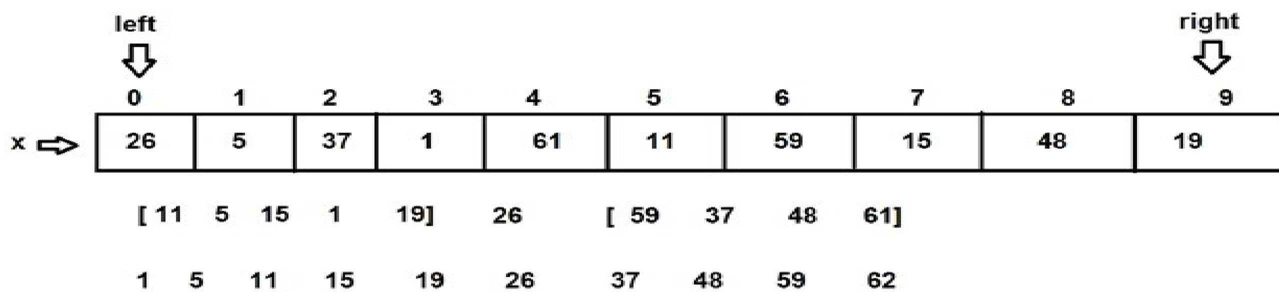


مرتب سازی سریع:

در این روش در هر گذر یک عنصـر محوری یا (pivot)انتخاب شـده وسـایر عناصـر به گونه ای جا به جا می شـود که کلیه عناصـر بزرگتر از آن در یک طرف وکلیه عناصـر کوچک تر در طرف دیگر قرار گیرد بدین ترتیب عنصـر در مکان درسـت خود قرار خواهد گرفت سپس دو بردار دو طرف این عنصـر با توجه به یکی از عناصـر به صـورت بازگشـتی به همین ترتیب مرتب می شـود این الگوریتم جزء الگوریتم های تقسـیم بوده که با کوچک کردن بازه عمل به چند بازه سپس ترکیب جواب های آنها مسعله را حل می کند.



```
Quick sort(x[ ] , left , right)
{
int I , j , pivot;
if(left < right)
{
i=left;
j=right+1;
pivot=x[left];
while(i > j)
while(x[i] >= pivot)
i=i+1;
while(x[j] <= pivot)
j--;
if(i<j)
swap(x[i] , x[j])
swaple(left , x[j])
Quick sort(x , left , j-1)
Quick sort(x , j+1 , right)
}
}
```



این جزوه آموزشی توسط دانشجوی کامپیوتر،گرایش نرم افزار آموزشکده فنی و حرفه ای شهید چمران اهربه نام حسین خلیلی زیر نظر جناب اقای مهندس امین جلیل زاده رزین مدرس همین آموزشکده طراحی شده است و هدف از آن سهولت در امر آموزش می باشد.امید است دانشجویان بتوانند با اراده و پشتکار مضاعف تمامی مطالب این جزوه را مطالعه وآن را یاد بگیرند.

حسین خلیلی