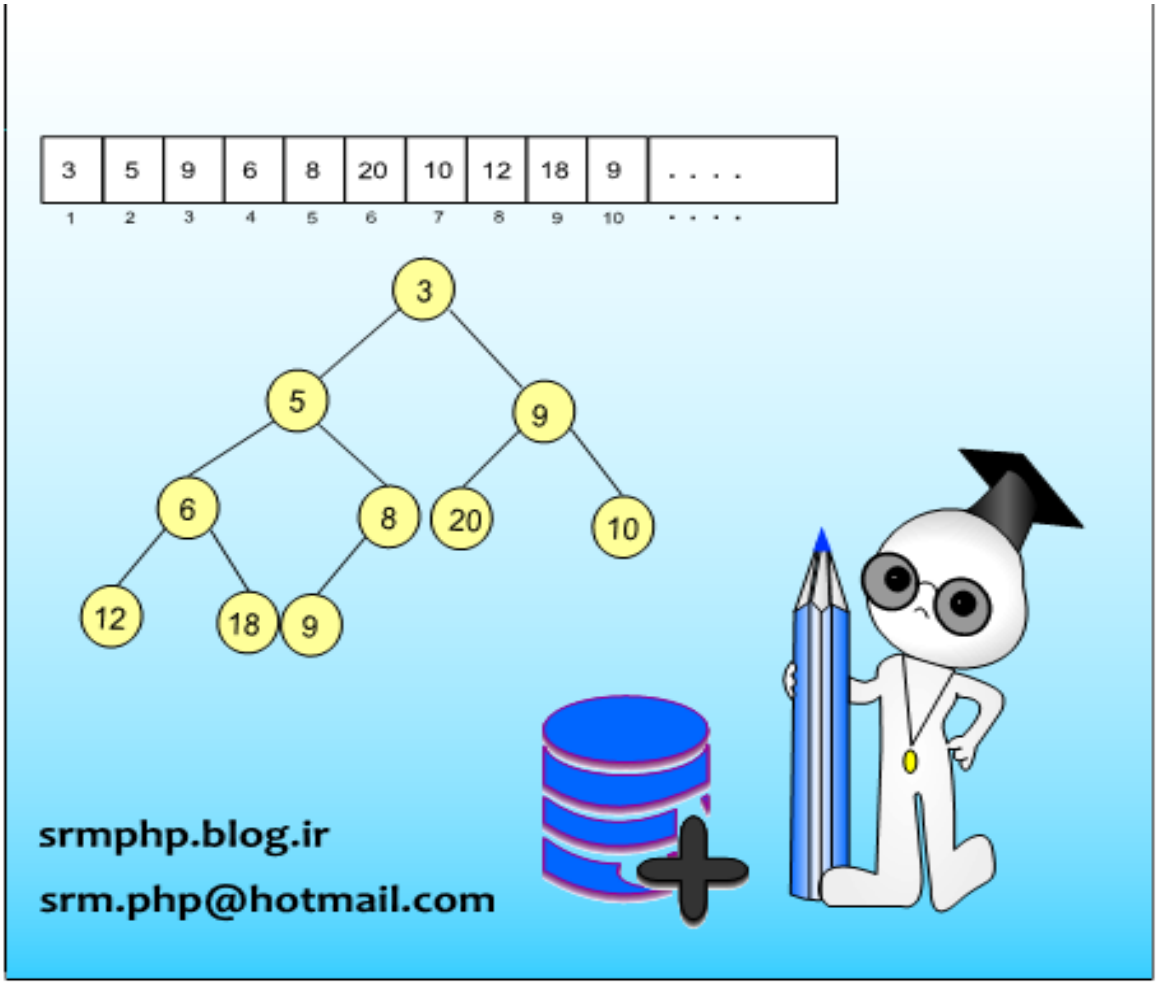
موسسه آموزش عالی مهراروند

ساختمان داده

استاد: مهندس سید رسول موسوی

تهیه و تنظیم: الهام صباحی



زندگی یعنی تکاپو

زندگی یعنی هیاهو

زندگی یعنی شب نو، روز نو، اندیشه نو

زندگی یعنی غم نو، حسرت نو، پیشه ی نو

زندگی بایست سرشار از تکان و تازگی باشد

زندگی بایست در پیچ و خم راهش ز الوان حوادث رنگ بپذیرد

زندگی بایست یکدم یک نفس حتی

زجنبش وانماند

گرچه این جنبش برای مقصدی بیهوده باشد

هوشنگ شفا

فصل اول : پیچیدگی زمانی

فصل دوم : آرایه

فصل سوم : درخت

فصل چهارم : گراف

فصل پنجم : مرتب سازی

فصل اول : پیچیدگی زمانی

الگوریتم ' algorithm '

مجموعه دستورالعمل هایی که برای رسیدن به یک هدف خاص ارائه می شود.

به بیان دیگر؛ مجموعه ای متناهی از دستور العمل هایی است که به ترتیب خاصی اجرا می شوند و مسئله را حل می کنند و لزوماً منحصر به فرد نیستند.


یک الگوریتم روشی گام به گام برای حل یک مسئله است.


تمام الگوریتم ها باید شرایط و معیار های زیر را دارا باشند:

- ورودی : یک الگوریتم می تواند هیچ یا چندین پارامتر را به عنوان ورودی بپذیرد.
- خروجی : یک الگوریتم باید حداقل یک کمیت را به عنوان خروجی "نتیجه عملیات" تولید کند.
- قطعیت : دستور های الگوریتم باید با زبانی دقیق و بدون ابهام بیان شود.(در اجرای آن هیچ ابهامی نباشد) هر دستورالعمل نیز باید انجام پذیر باشد.
- خاتمه پذیر باشد : الگوریتم باید دارای شروع و پایان مشخصی باشد، به نحوی که اگر دستورهای آن را دنبال کنیم پس از طی مراحلی که قابل شمارش و متناهی باشد، خاتمه یابد.

لازم به ذکر است مدت زمان لازم برای خاتمه یک الگوریتم باید به گونه‌ای معقول و کوتاه باشد.

- ✓ کارایی داشته باشد.(قابل تحلیل باشد.)

مهم ترین معیار های انتخاب یک الگوریتم

- ۱- زمان
  - پیچیدگی زمانی ' order '
  - مرتبه اجرایی
- ۲ - حافظه (در بحث ذخیره و بازیابی حائز اهمیت است).

پیچیدگی زمانی 'order'قطعه کد زیر را مشخص کنید.

قطعه کد	زمان اجرا
{	0
int x=y	0
x=10	1
y ← 1	1
}	0
	2

پیچیدگی زمانی' o ' = (2) o

زمانی order ، n می شود که ؛ حلقه for بازگشتی باشد.

$$\lceil 2.5 \rceil = 3 \qquad \lfloor 2.5 \rfloor = 2$$

کران بالا " سقف " کران پایین" کف "

مجموعه

از اجتماع چند عنصر هم نوع مجموعه به وجود می آید.

عملگر	نام	مثال
%	باقیمانده تقسیم'mod'	X % y

مضرب ها

نمایش مضارب اعداد طبیعی n به فرم زیر است؛

$$\sum_{n=1}^{10} n^2 = 1^2+2^2+3^2+...10^2 = ?$$

نمایش مضارب عدد 2 :

$$\sum_{n=1}^{10} = 2^n = 2^1 + 2^2 + 2^3 + \ldots + 2^{10} = ?$$

فاکتوریل ' ! '

هر عدد طبیعی از حاصل ضرب آن عدد در تمام اعداد صحیح و مثبت (اعداد طبیعی) کوچکتر از آن به دست می آید.

$$۳ ! = ۳ * ۲ * ۱$$

جایگشت

نحوه‌ی قرارگیری عناصر یک مجموعه با نظم و ترتیب خاص در کنار هم را جایگشت گویند.

به بیان دیگر؛ مرتب سازی یا تغییر ترتیب اعضای یک مجموعه را جایگشت گویند.

فرض کنید مجموعه M={a,b,c} باشد. جایگشت این مجموعه بصورت زیر خواهد بود؛

$$M=\{a,b,c\} \longrightarrow abc\ ,\ acb\ ,\ bac\ ,\ bca,\ cab,\ cba\ ,\ \ldots$$

لگاریتم logarithm

الگوریتم یک عدد در یک پایه، برابر با توانی از پایه است که آن عدد را می دهد.

مثلا؛ لگاریتم عدد ۱۰۰۰ در پایه ۱۰ برابر ۳ خواهد بود.

$$\text{Log}_{10}(1000) = 3 \quad \equiv \quad 1000 = 10 \times 10 \times 10$$

می توان نتیجه گرفت لگاریتم برعکس توان است.

$$\log_2^8 = 3 \iff 2^3 = 8 \qquad \log_2^{16} = 4 \iff 2^4 = 16$$

لگاریتم عدد صفر " $\log_n^0$ " و اعداد منفی " $\log_n^{-X}$ "در هر مبنایی که باشند؛ حاصلی برایشان تعریف نشده است.

قضیه ۱

فرمول جمع مقادیر بصورت زیر خواهد بود؛

formula $1+2+3+...+n = \frac{n(n+1)}{2}$

Example $1+2+3+...+100 = \frac{100\ (101)}{2}$

formula $1+2+3+...+(n-1) = \frac{n(n-1)}{2}$

Example $1+2+3+...+(n)+(n+1) = \frac{(n+1)(n+2)}{2}$

ساختار تکرار دستور For 'دستورات شرطی'

ساختارهای تکرار تحت شرایط خاصی یک یا چند دستور را چندین بار اجرا می کنند.

For شمارنده = مقدار اولیه to مقدارنهایی step گام افزایش

{

مجموعه دستورات

}

- روند عملکرد حلقه for :

شمارنده ◄ دستورات ◄ در صورت برقراری شرط For i شرط

پیچیدگی زمانی در الگوریتم

رفتار الگوریتم را در زمان اجرا با مجموعه‌ای از ورودی‌های منتخب توصیف می‌کند.

پیچیدگی زمانی قطعه کدهای زیر را بیابید.

حلقه for :

```
for   i=1   n        n+1
 {
   x=x+1             n×1=n
   y=y+1             n×1=n
   z=z+1             n×1=n
  }                  ----------
                     (n+1)+3n=n+1+3n=4n+1
```

پاسخ دقیق زمان اجرا ◄

پیچیدگی زمانی ' order ' : o (n)

حلقه for تو در تو :

```
for i 1 to n   | n+1
 {             |
  for j 1 to n | n*(n+1)=n^2 +n
   {           |
    print i * j| n*n=n^2
   }           |-------------------
 }             | 2 n+1 + 2n^2 =2n^2 +2n+1
```

پیچیدگی زمانی ' order ' : o (n^2)

قضیه ۲

فرض کنید دارای یک حلقه for و یک سری دستورات هستیم، تعداد تکرار حلقه for برابر است با تعدا تکرار دستورات درون حلقه.

یک + کران پایین – کران بالا

N – m + 1

Example : for i m to n

قضیه ۳

اگر تابع f(x)بصورت $f(x) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_2 n^2 + a_1 n^1 + a_0 n^0$؛ باشد آنگاه مرتبه اجرایی آن $o(n^m)$ خواهد بود.

اگر حلقه های for تو در تو باشند در یکدیگر ضرب می شوند. واگر بصورت مجذا از هم باشند با یکدیگر جمع می شوند.

مرتبه اجرایی هریک از شبه کد های زیر را بنویسید.

مثال۱ :

```
For  i=1  to  n        n+1
{
   X+=1;               n × 1 = n ---> 2n+1   تابع مرتبه اجرایی
 }
For  j=1 to  m         m+1
 {
   Y+=2;               m ---> 2m+1   تابع مرتبه اجرایی
 }
```

پیچیدگی زمانی ' order ' : $o(n + m)$

مثال۲ :

```
For   i=1   to   m     m + 1
  {
    For  o=1  to  n    m ( n+1 )= mn + 1
       {
         Y+=3;         m × n = mn
          }            2mn + 2m + 1   تابع مرتبه اجرایی
}
```

پیچیدگی زمانی ' order ' : o (mn)

مثال ۳ :

```
For  i  L  to  1          | n + 1
  {                       |
    X = X + I ;           | n
     }                    | 2n + 1
```

پیچیدگی زمانی' order ' : o (2n + 1)

حلقه for همیشه یکی بیشتر از خودش اجرا می شود. (هدف برسی شرط حلقه)

مثال ۴ :

```
For  i  5  to  m                  | فرمول ---> m – n + 2
 {                                | m – 5 + 2 = m – 3
   For  j  7  to  n               | یک واحد کاهش می یابد
    {                             | m – 2 (n – 7+ 2)
       A [ i , j ]=a [ i , j ] * b [ i , j ]   | m – 2n + 10
     }                            | m – 2 ( n – 4 )
                                  | m – 2n + 8
 }                                |
                                  |   m – 3
                                  |   m – 2n +10
                                  |   m – 2n + 8
                                  |   3m – 4n +15
```

پیچیدگی زمانی' order ' : o (m)

برای عبارت زیر شبه کد های برنامه نویسی بنویسید.

$$\sum_{i=1}^{n} i^2 + 2^i - 1$$

پاسخ

```
For  i  1  to  n                    | n + 1
  {                                 |
    X= x+ [ ( i ^ 2) +( 2 ^ i ) ] – 1 | n
  }                                 | 2n + 1
```

در این مثال متغییر x حاصل جمع های متوالی را در خود ذخیره می کند.

پیچیدگی زمانی' order ' : (n) o

حلقه while

این دستور تا زمانی که شرط برقرار باشد دستورات را اجرا می کند و شرط قبل از اجرای دستورات چک می شود.بدین معنا که اگر در وسط اجرای دستورات متغییرها طوری تغییر کنند که شرط برقرار نباشد، دستورات همچنان تا پایان اجرا می شوند.

```
while ( شرط یا شرایط )
 {
دستوراتی که می خواهیم تکرار شوند
 }
```

بیشتر بدانید!

بهترین راه برای حل کردن حلقه های شرطی آن است که؛ برای آن ها یک محدودیت (مثلا :یک عدد)در نظر گرفته و بر اساس محدودیت مد نظر حلقه را حل کرد.

اگر n = 16 و i را i = 1 , 2 , 4 , 8 , 16 در نظر بگیریم؛ مرتبه اجرایی حلقه زیر را بیابید.

```
I = 1
While ( I < n )
 {
    X = x + 1 ;
    i = i × 2 ;
    }
```

Analysis

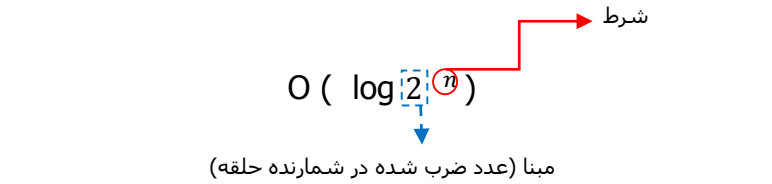
I	شـرط	مرتبه اجرایی
1	1 < 16 ✓	*
2	2 < 16 ✓	*
4	4 < 16 ✓	*
8	8 < 16 ✓	*
16	16 < 16 ×	

۴ ◄--- ۱۶ , $۲^۴=۱۶$

پیچیدگی زمانی' order ' : $o (\log_2 n)$

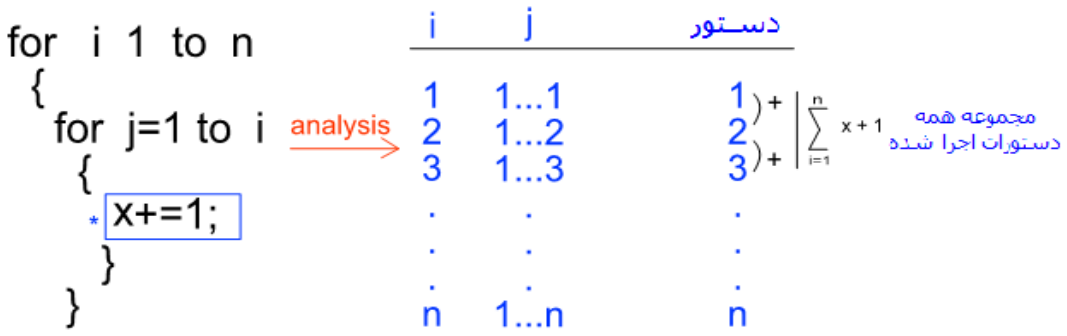
قضیه ۴

هرگاه در حلقه ای مانند :while شمارنده یا شرط حلقه در عددی ضرب و یا بر عددی تقسیم شود، مرتبه اجرایی آن الگوریتم (حلقه) لگاریتم انتهایی حلقه (شرط اتمام حلقه) در مبنای عدد ضرب شده یا تقسیم شده در شمارنده می باشد.



Formula : $\frac{n(n+1)}{2} \equiv \sum_{i=1}^{10} i = 1 + 2 + 3 + \dots + 10$

تعداد تکرار دستور ستاره دار چند مرتبه است ؟



پیچیدگی زمانی' order ' : $o(n^2)$

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

توابع

یکی از نماهایی که در آنالیز کردن استفاده می کنیم تابع است.

۱ - توابع بازگشتی

به توابعی که بر اساس شرایطی که دارند مجددا خود را فراخوانی می کنند گفته می شود.

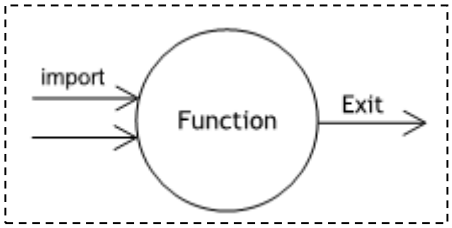
نحوه ی تعریف این توابع به فرم زیر است :

نام تابع (ورودی ها) --example--> f (x , y)

{

.....

}



ویژگی توابع بازگشتی :

۱- از معایب توابع بازگشتی این است که در نهایت سادگی حجم زیادی را اشغال کرده و نسبتا کند عمل می کنند و نیز؛ در زمان پردازش cpu را درگیر می کنند.

۲- خود را فراخوانی می کنند.(تعداد دفعاتی که خود را فراخوانی می کند باید محدود باشد.بدیهی است که در غیر این صورت اجرای آن خاتمه نمی یابد.)

۳- دارای شرط خاتمه می باشد.

```
Sig  ( n )                    n+1
  {
     If  ( n <=1 )            n+1
     Return 1;                n+1
Else  n + sig ( n + 1 );      n.n=n 2
                              ----------------
     }                        3n + n2 + 3  مرتبه اجرایی
```

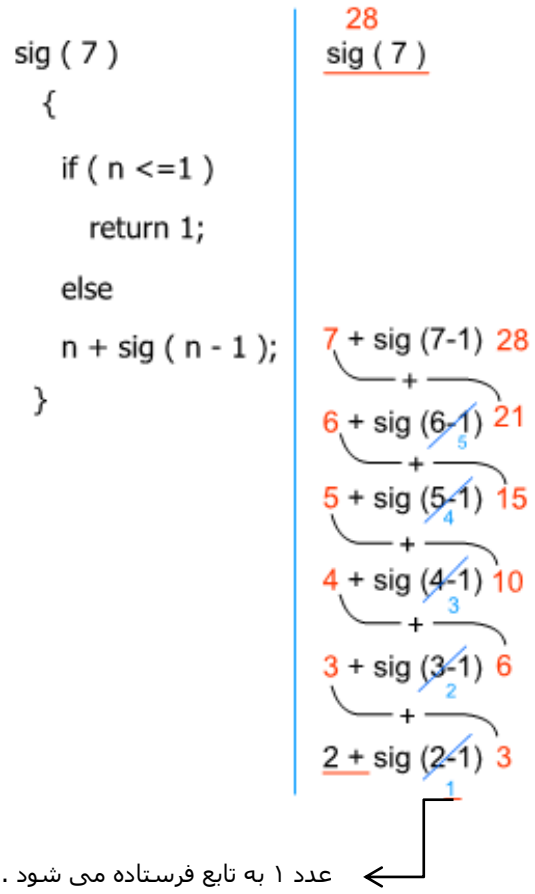
مرتب شده ⟶ $n^2 + 3n + 3$

'order' : $O(n^2)$ پیچیدگی زمانی

آنالیز توابع بازگشتی

از آنجایی که در توابع بازگشتی، توابع خود را فراخوانی کرده و برای انتقال جواب از روش پشته استفاده می کنند ، فلذا؛ تنها راه پیاده سازی پشته درخت است.

مثال:



توضیح :

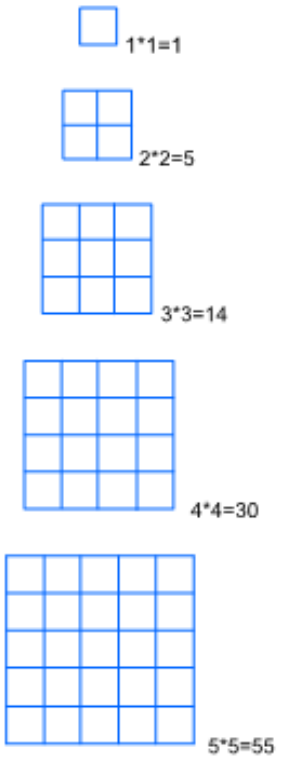
آیا ۷>۱است ؟ خیر. فلذا خط ' return 1 ' اجرا نمی شود. در غیر این صورت عدد (7)+n ، فراخوانی مجدد تابع(عدد انتزاعی)

مجددا تابع فراخوانی می شود برای عدد ۶.شرط برسی می شود .آیا ۶>۱؟ خیر. فلذا خط return 1 ' 'اجرا نمی شود. در غیر این صورت عدد (6)+n فراخوانی مجدد تابع(عدد انتزاعی)

مراحل فوق را تا زمانی تکرار می کنیم که شرط نقض شود (n<=1) باشد و عدد ۱ را return کند.


تعداد مربع های یک صفحه ی شطرنجی ۱۰×۱۰ چقدر است ؟

پاسخ :



n x n + f (n - 1)

10 x10 + (9 answer) = 385

9 x 9 + (8 answer) =285

8 x 8 + (7 answer) =204

7 x 7 + (6 answer) =140

6 x 6 + (5 answer) =91

5 x 5 + (4 answer) =55

4 x 4+ (3 answer) =30

3 x 3 + (2 answer) =14

2 x 2 + (1 answer) = 5

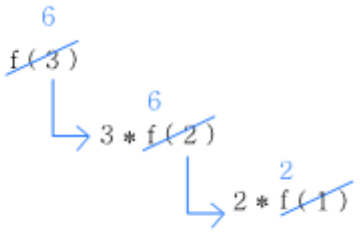
1 x 1 + (answer) = 1

توابع بازگشتی زیر چه عملی را انجام می دهند؟ مرتبه اجرایی آن را بنویسید.

```
f ( n )
if ( n < = 1 )
  return 1 ;
else
   return  n * f (n - 1)
```

n
n+1
1
n-1
3n - 1

example



o (n) : ' order '

در مثال فوق تابع بازگشتی بصورت پشته عمل کرده و هر بار خودش را فراخوانی می کند.

جواب این توابع همواره به مرحله قبل وابسته بوده تا زمانی که به خروجی پایانی برسد.

اعداد ثابت هیچ تاثیری در مرتبه اجرایی ندارند.(+ و – و / و X)

هر علامتی می تواند جایگزین شود(+ و – و / و X)

مرتبه اجرایی	تابع
O (n)	N + f (n – 1)
O ($\frac{n}{2}$)	N + f (n – 2)
O ($\frac{n}{3}$)	N + f (n – 3)
	.
	.
	.
O ($\frac{n}{m}$)	N + f (n – m)

مرتبه اجرایی	تابع
$\log_2 n$	N + f ($\frac{n}{2}$)
$\log_3 n$	N + f ($\frac{n}{3}$)
	.
	.
	.
$\log_m n$	N + f ($\frac{n}{m}$)
	N + f ($\frac{n}{2}$) + f ($\frac{n}{2}$)

مرتبه اجرایی	تابع
o ($_2 n$)	N + f (n - 1) + f (n – 1)
O ($2^{\frac{n}{2}}$)	N + f (n - 2) + f (n – 2)
	.
	.
	.
O ($2^{\frac{n}{m}}$)	N + f (n – m) + f (n – m)

مثال:

N + f (n – 1) + f (n – 2)

o ($2^{n+\frac{n}{2}}$) : ' order '

مرتبه اجرایی	تابع
$\mathbf{o}\,(_2\,\boldsymbol{n})$	$N \oplus f\,(\frac{n}{2}) + f\,(\frac{n}{2})$
$\mathbf{3^n}$	N + f (n - 1) + f (n – 1) + f (n – 1)

مرتبه اجرایی الگوریتم زیر چیست؟

```
F  ( n )
   If  ( n <= 1 )  return 1;
      Else
   Return  f ( n – 2 ) * f ( n – 2 );
```

$o\,(2^{\frac{n}{2}})$: ' order '

خروجی الگوریتم زیر به ازای عدد ۱۰۰ چیست ؟ فرمول ریاضی آن را بنویسید.

```
f ( n )
   if  ( n < = 1 )
       return 1;
    else
       return  n + f ( n – 1 );
```

$$\sum \frac{n\ (n+1)}{2} = \frac{100\ (101)}{2} = 5.050$$

خروجی تابع زیر به ازای ورودی عدد ۳ چیست ؟ مرتبه اجرایی آن را بدست آورید.

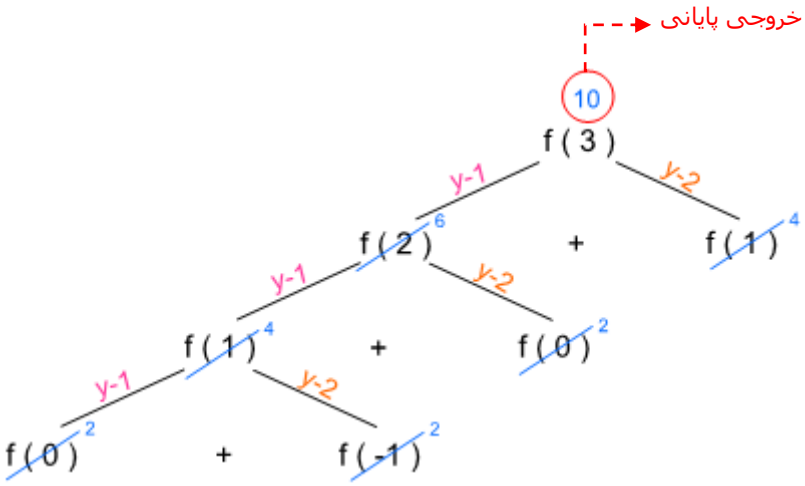
```
f  ( y )
if   ( y > 0 )
     return  f ( y – 1 ) + f ( y – 2 );
else
     return 2;
```

$$n + \frac{n}{2} = \frac{2n}{2} + \frac{n}{2} = \frac{3n}{2}$$

حال به ازای ورودی ۳ خروجی بصورت زیر خواهد بود؛

شرط برسی می شود ؛ آیا *y=3 > 0* است؟ بله ، فلذا یکبار *y-1* و یکبار *y-2*

درصورتی که حاصل بزرگتر از نبود بجای آن عدد ۲ را قرار میدهیم.



$$(2^{n+\frac{n}{2}}) = (2^n) \quad : \quad \text{'order' o}$$

خروجی تابع زیر به ازای ورودی های ۲ و ۴ چیست ؟

```
Bino   ( n , m )
    If ( n=0   OR   n = m )
        Return 1 ;
    Else
        Return  bino  ( n – 1 , m ) + bino  ( n - 1 , m - 1 )
```

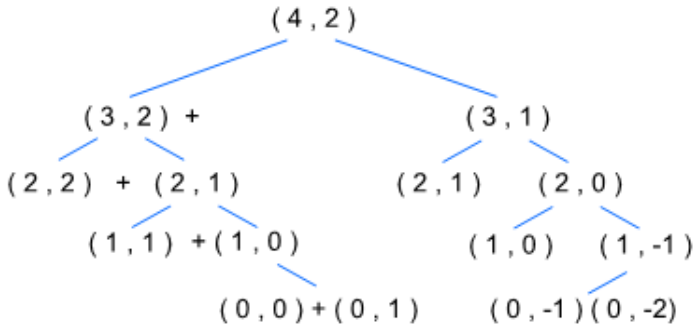
توضیح:

گام ۱ : آیا n=0 ؟ خیر ،آیا n = m ؟ خیر . فلذا یک واحد از n کم می شود ولی m با مقدار قبل باقی می ماند.
+ یک واحد از m و یک واحد از n کم می شود.

گام ۲ : n=0 نبوده و n=m نمی باشد فلذا n-1 و m با مقدار قبل باقی می ماند.

گام ۳ : n=0 نبوده اما n=m می باشد ، فلذا بجای آن عدد ۱ را قرار می دهیم.

در قسمت راست درخت n=0 نبوده و n=m نمی باشد. فلذا لازم است مراحل فوق مجددا تکرار شود تا به خروجی مناسب برسیم.



تابع اکرمن

یک تابع بازگشتی است که یک رایانه یا پردازشگر با حافظه بی‌کران می‌تواند آن را محاسبه کند.


اگر به عدد دوم تابع آکرمن دو واحد اضافه کنیم، حاصل نهایی بدست می آید.


شرط پذیرفتن تابع آکرمن :

در صورتی یک تابع را به عنوان تابع اکرمن در نظر می گیریم که ؛ عدد اول یک و عدد دوم x باشد.


خروجی تابع بازگشتی زیر چیست؟

$$f(m,n)\begin{cases} n+1 & m=0 \\ f(m-1,1) & n=0 \\ f(m-1,f(m,n-1)) & m,n>0 \end{cases}$$

$(1,3)$

$f(\underset{m}{1},\underset{n}{3})$ ⑤
$f(\underset{m-1}{0},f(\underset{m,n-1}{1},2))$ ④
$f(0,f(1,1))$ ③
$f(0,f(1,0))$ ②
$f(0,1)$ ①

اگر m = 0 باشد ⟵ یک واحد به n اضافه کن.

گر n = 0 باشد ⟵ رابطه بازگشتی را به f باز گردان.

اگر m , n > 0 باشد ⟵ رابطه را به تابع باز گردان.

روش حل:

گام ۱ : آیا m=0 است؟ خیر. آیا n=0 است؟ خیر. آیا m,n > 0 هستند ؟ بله ، فلذا حاصل m-1 را به عنوان خانه اول در نظر می گیریم. ((m-1 , f (m , n-1)) ⟵ (0 , f (1 , 2))

گام ۲ : مجددا یک تابع بازگشتی ایجاد می شود.

آیا m=0 است؟ خیر. آیا n=0 است؟ خیر. آیا m,n > 0 هستند ؟ بله ((m-1 , f (m , n-1)) .

گام ۳ : مجددا یک تابع بازگشتی ایجاد می شود.

آیا m=0 است؟ خیر. آیا n=0 است؟ خیر. آیا m,n > 0 هستند ؟ بله (F (0 , f (1 , 0))

گام ۴ : مجددا یک تابع بازگشتی ایجاد می شود.

m-1 و n را برابر ۱ قرار میدهیم. (f (0 , 1)

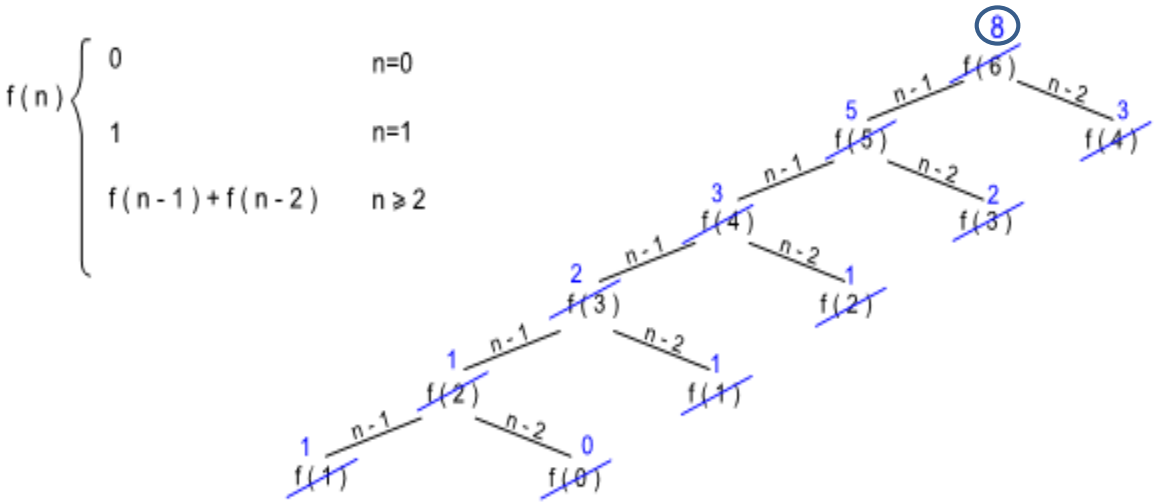
گام ۵ : مجددا یک تابع بازگشتی ایجاد می شود.

آیا m=0 است؟ بله. یک واحد به n اضافه کرده و به تابع قبل باز گردان.

شرط : هر گاه m=0 باشد، n+1 می شود.

' order ' : ($\frac{n}{2}$) o

خروجی تابع بازگشتی زیر به ازای ورودی عدد ۶ چقدر خواهد بود؟



روش حل:

گام ۱: آیا n=0 هست ؟خیر. آیا n=1 هست ؟خیر. آیا n>2 هست ؟ بله. فلذا ؛به دو تابع جمع n-1 و n-2 تقسیم می شود.

F(6) – 1 = f(5) , f(6) – 2 = f(4)

گام ۲ : آیا n=0 هست ؟خیر. آیا n=1 هست ؟خیر. آیا n>2 هست ؟ بله. فلذا ؛به دو تابع جمع n-1 و n-2تقسیم می شود.

F(5) – 1 = f(4) , f(5) – 2 = f(3)

گام ۳ : حال f (4) را در نظر می گیریم.

آیا n=0 هست ؟خیر. آیا n=1 هست ؟خیر. آیا n>2 هست ؟ بله. فلذا ؛به دو تابع جمع n-1 و n-2 تقسیم می شود.

F(4) – 1 = f(3) , f(4) – 2 = f(2)

گام ۴ : حال f (3) را در نظر می گیریم و شرایط فوق را چک می کنیم.

F(3) – 1 = f(2) , f(3) – 2 = f(1)

گام ۵ : حال f (2) را در نظر می گیریم و شرایط فوق را چک می کنیم.

$$F(2) - 1 = f(1) \quad , \quad f(2) - 2 = f(0)$$

تابع فوق از روش فیبوناچی می باشد.

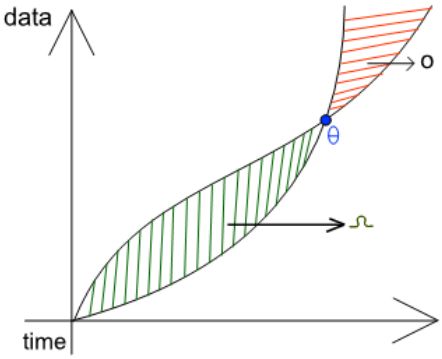
اعداد فیبوناچی نمونه اولیه ای از یک رابطه بازگشتی همگن با ضرائب ثابت می باشد.

$$F_n = f_{n-1} + f_{n-2}$$

- ✓ بیشتر بدانید...!؟!¿

به توالی اعداد فیبوناچی می رسیم که به شکل: ۰، ۱، ۱، ۲، ۳، ۵، ۸، ۱۳، ۲۱، ۳۴، ۵۵، ۸۹ و ...است.

در ادامه مباحث فوق، پیچیدگی زمانی دارای سه حالت ⟷ بد ترین ' o '، متوسط یا دقیق ' θ '، بهترین ' Ω ' می باشد.



اجرا

- ✓ زمان های اجرای یک الگوریتم به ترتیب از بهترین تا بدترین حالت ممکن:

$$O(1) < O(\log^n) < O(n) < O(n \log^n) < O(n^2) < O(n^3) < \ldots < O(2^n) < O(n!) < O(n^n)$$

فرض کنید دارای یک تابع با پیچیدگی زمانی n هستیم ؛

بد ترین حالت ' big-o 'تابع بصورت؛ $o(n^1)$, ..., $o(n^2)$, $o(n \log^n)$ می باشد.

حالت متوسط تابع بصورت؛ $\theta = o(n)$ می باشد.

بهترین حالت ' Ω ' بصورت؛ Ω = (o) log (n) , o (1) , o (n) می باشد.

اگر یک تابع داشته باشیم ،برای محاسبه ی بدترین حالت پیچیدگی زمانی آن تابع رابطه زیر صادق است:

$$F(n) = o(g(n)) \Leftrightarrow \exists c, n_0 > o : \forall n \geq n_0 \; f(n \leq c\, g(n))$$

$$\theta = o(n \log n), o(n^2) \dots o(n^n), o(n)$$

اومگا ' Ω '

اومگا، مگا به معنی بزرگ است .

$$f(n) = \Omega = \log(n), o(1), o(n)$$

$$F(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 > o : \forall n \geq n_0 \; f(n) \geq c\, g(n)$$

حالت ' θ '

حالت متوسط " دقیق "

$$f(n) = \Omega = \log(n), o(1), o(n)$$

$$F(n) = \Omega(g(n)) \Leftrightarrow \exists c, n_0 > o : \forall n \geq n_0 \; f(n) \geq c\,(g(n)) \; \Omega(g(n)) \cap o(g(n))$$

اشتراک اومگا و Big-o

$$\Omega(g(n)) \cap o(g(n))$$

نتیجه میگیریم : پیچیدگی زمانی big-o همواره باید بزرگتر یا مساوی خود تابع اصلی باشد.

کدام یک از تساوی های زیر درست است؟

الف) $5n^2 - 6n = o(n^3)$ √ $o(n^2) = o(n^3)$

بد ترین حالت برای یک تابع باید ؛اعداد بزرگتر همراه با خود پیچیدگی زمانی باشد.

ب) $5n^2 - 6n = \theta(n^3)$ X $o(n^2) \neq \theta(n^3)$

در حالت میانگین تتا باید مساوی درجه تابع باشد.

ج) $5n^2 - 6n = \Omega(n^3)$ X $o(n^2) \neq \Omega(n^3)$

بهترین حالت برای یک تابع باید ؛اعداد کوچکتر همراه با خود پیچیدگی زمانی باشد.

فصل دوم : آرایه

آرایه تعدادی متغیر از یک نوع داده و تحت یک نام می‌باشد. هر یک از متغیرهای درون آرایه با یک شماره که به آن «اندیس» می‌گوییم از یکدیگر متمایز می‌شوند. متغیرهای درون آرایه را «عناصر آرایه» می‌نامند که همگی قابلیت نگهداری فقط یک نوع داده را دارند.

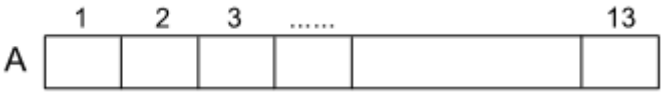
عناصر درون آرایه از نظر فیزیکی مکان‌های متوالی در حافظه اصلی رایانه را اشغال می‌کنند. بنا بر این تعداد عناصر درون آرایه محدود و ثابت می‌باشد.

خانه‌های آرایه توسط اندیس مشخص می شوند که یک عدد صحیح است، مثلا خانه شماره ۵ یعنی خانه ای که اندیس‌اش ۵ است. هر آرایه ای یک اندیس شروع و یک اندیس پایان دارد که شماره های مغتبر اندیس بین این دو خواهند بود. L1 اندیس شروع آرایه است و L اختصاری Low یعنی پایین است . مثلا اگر L1 برابر ۴ باشد، اولین اندیس آرایه ۴ است. U1 اندیس بالا آرایه است و U اختصاری Up یعنی بالا است. مقدار U1 همیشه مساوی با بزرگتر از L1 است.اگر اندیس شروع یک آرایه (L1) برابر ۱ و اندیس پایان آرایه (U1) برابر با ۵ باشد، اندیس های معتبر آرایه مقادیر ۱ و ۲ و ۳ و ۴ و ۵ خواهند بود.

تعداد عناصر آرایه برابر است با ۵ خانه که از فرمول زیر محاسبه می شود:

```
U1 - L1 + 1
```

example ⟵ 5 - 1 +1 = 5



برای دسترسی به یک خانه از آرایه، از شماره اندیس آن استفاده می کنیم.

A [3]

نام آرایه ⟵ ⟶ شماره اندیس

مزایای استفاده از آرایه:
- سادگی در استفاده
- دسترسی مستقیم به خانه ها

معایب استفاده از آرایه:
- ایستا بودن و محدود بودن (الزاما تعداد خانه ها باید مشخص باشد)
- به جزء حذف و اضافه کردن به انتهای آرایه و برای حذف و اضافه کردن به وسط آرایه نیاز به Shift است.

درج کردن در آرایه

برای درج کردن متحوی در یک آرایه از فرمول های زیر استفاده می کنیم :

نوع عمل	فرمول
محل عملیات	K = 1 . . . N
تعداد Shift مورد نیاز	N – K + 1
متوسط تعداد Shift	$\frac{n+1}{2}$

o (n) : ' order '

0	1	2	3	4	5	6
5	6	7	8	3	9	

مثال : 1 + 4 – 7

حذف کردن از آرایه

برای حذف کردن متحوی در یک آرایه از فرمول های زیر استفاده می کنیم :

نـوع عمل	فـرمول
محل عملیات	K = 1 . . . N
تعداد Shift مورد نیاز	N – K
متوسط تعداد Shift	$\frac{n-1}{2}$

بیشتر بدانید . . . ؟!¿!

برای دسترسی تصادفی به یک خانه از حافظه، یک واحد زمان لازم است.

برای اضافه کردن ' append 'یک مقدار به انتهای یک آرایه مرتب، یک واحد زمان لازم است. o (1)

برای حذف کردن یک مقدار از آرایه ، یک واحد زمان لازم است. o (1)

فضای اشغال شده توسط یک تابع = تعداد عناصر آرایه X اندازه نوع داده

جدول انواع عددی و میزان فضای اشغال شده در خافظه به فرم زیر است ؛

نوع	میزان حافظه (بایت)
Byte	1
Integer	2

4	Long
4	Single
8	Double
8	Currency '$ '

حالات یک آرایه سه بعدی بصورت زیر است ؛

$$A [L_I ..... U_I , L_j ..... U_j , L_k ..... U_k]$$

آدرس شروع آرایه را با α ، و اندازه هر خانه (تعداد بایت) آرایه را با β مشخص می کنیم.

سیستم ها به دو صورت آرایه ها را پیمایش می کنند :

۱- سطری (از ابتدا به انتها)

۲- ستونی

۱- پیمایش سطری :

$$A (i,j,k) = [(i-L_1) (U_2 - L_2 +1) (U_3 - L_3 +1) + (J - L_2) (U_3 - L_3 +1) + (K - L_3)] \chi \beta + \alpha$$

آرایه ی سه بعدی A [1:15 , -5:5 , 10:25] را برای ذخیره اعداد به طول 2 بایت بکار گرفته شده است. اگر آرایه به صورت سطری از خانه 2000 حافظه ذخیره شده باشد آدرس عنصر A [2 , 5 , 15] چقدر است ؟

$\alpha = 2000 \quad , \quad \beta = 2$

[(2-1) (5 - (-5) +1) (25-10+1) (5-5) (25-10) (15-10)] χ 2+2000= ???????

استاندارد یک آرایه سه بعدی به فرم زیر است :

A [3 , 4 , 5] OR A [3][4][5]

بعد اول سه تایی بعد دوم چهار تایی بعد سوم پنج تایی

بیشتر بدانید...!؟!؟

زبان های برنامه نویسی امروزه ، شماره خانه های یک آرایه را یا از صفر شروع می کنند یا از یک.

- در زبان های برنامه نویسی گذشته ،این قابلیت را داشتیم که آدرس شروع و انتهای آرایه را خودمان تعیین کنیم. به همین علت کران پایین و بالای آرایه ها را مشخص می کردیم.

$$A[L_1..U_1 , L_2..U_2 , L_3..U_3]$$

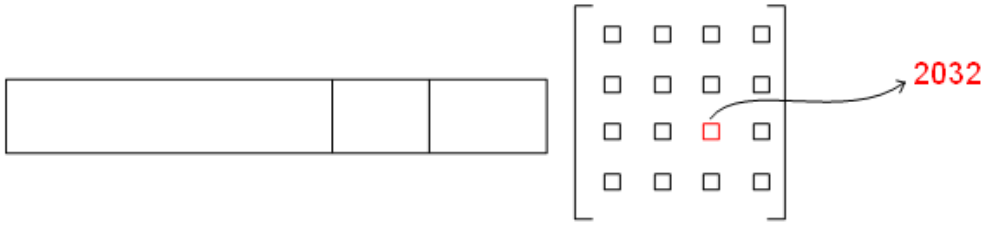
کران پایین و بالا بعد اول کران پایین و بالا بعد دوم کران پایین و بالا بعد سوم

۲- پیدا کردن آدرس یک خانه در حافظه بصورت پیمایش ستونی :

پیمایش یک آرایه ستونی از راست به چپ صورت می گیرد.

A [i , j , k] تعیین جهت پیمایش

$$[(K-L_3)(U_2-L_2+1)(U_1-L_1+1)(J-L_2)(U_1-L_1+1)(i-L_1)] \times \beta + \alpha$$



آرایه دو بعدی X [-5 . . 5 , 3 . . 33] در آدرس 400 به بعد حافظه ذخیره شده است. و هر خانه از آرایه احتیاج به 4 بایت دارد .آدرس X [4 , 10] به روش ستونی چقدر خواهد بود؟

$\beta = 4$

$\alpha = 400$

X [4 , 10]
i j

400 + [(10 - 3) (5 - (-5) +1) + (4 - (-5))] x 4=1944
7 11 9
77
86
486

در جستجوی یک آرایه به روش ستونی ، برای تعیین کردن شماره خانه از فرمول زیر استفاده می شود:

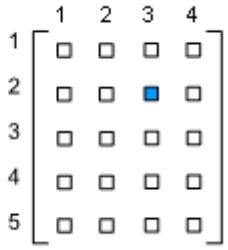
حذفα , $1+\beta$

در آرایه ی [4..1,'e'..'a'] A شماره ی درایه ی['3' , 'b'] A چند است؟

I j $L_1..u_1$ $L_2..u_2$

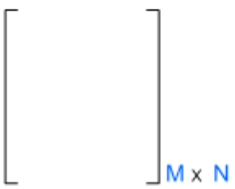
a	b	c	d	e
1	2	3	4	5

[(2 - 1)(4 - 1 + 1) + (3 - 1)] + 1 = 7
1 4 2



ماتریس

آرایه های دو بعدی m χ n بوده که m تعداد سطر ها و n تعداد ستون ها می باشد.

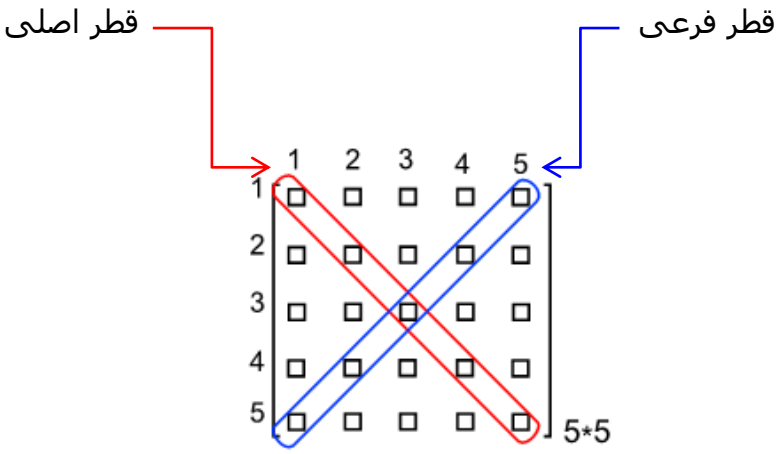


- ✓ قطر اصلی :

یکی از ویژگی ماتریس ها قطر اصلی بوده که ؛دارای شماره سطر و ستون یکسان می باشند.

- ✓ قطر فرعی :در قطر فرعی خانه ها برعکس یکدیگر می باشند.

رابطه قطر اصلی i = j می باشد.



ماتریس بالا مثلثی و ماتریس پایین مثلثی

ماتریسی که کلیه اعضای زیر قطر اصلی آن صفر باشد را ماتریس بالا مثلثی و ماتریسی که کلیه اعضای بالای قطر اصلی آن صفر باشد را ماتریس پایین مثلثی گویند.

- ✓ ماتریس بالا مثلثی

$$U=\begin{bmatrix} U_{1,1} & U_{1,2} & U_{1,3} & U_{1,4} \\ 0 & U_{2,2} & U_{2,3} & U_{2,4} \\ 0 & 0 & U_{3,3} & U_{3,4} \\ 0 & 0 & 0 & U_{4,4} \end{bmatrix}$$

- ✓ ماتریس پایین مثلثی

$$L=\begin{bmatrix} L_{1,1} & 0 & 0 & 0 \\ L_{2,1} & L_{2,2} & 0 & 0 \\ L_{3,1} & L_{3,2} & L_{3,3} & 0 \\ L_{4,1} & L_{4,2} & L_{4,3} & L_{4,4} \end{bmatrix}$$

بیشتر بدانید....؟!؟

نکاتی درمورد ماتریس بالا مثلثی و پایین مثلثی :

- ✓ تعداد کل خانه های یک ماتریس n^2 می باشد.
- ✓ تعداد خانه های مخالف صفر $\frac{N(N+1)}{2}$ می باشد.
- ✓ تعداد خانه های صفر $\frac{N(N-1)}{2}$ می باشد.

ماتریس های L قطری

در ماتریس هایی که در آن L قطر ،که قطر اصلی نیز شامل آن است مخالف صفر باشد،L قطری گویند.

ماتریس سه قطری

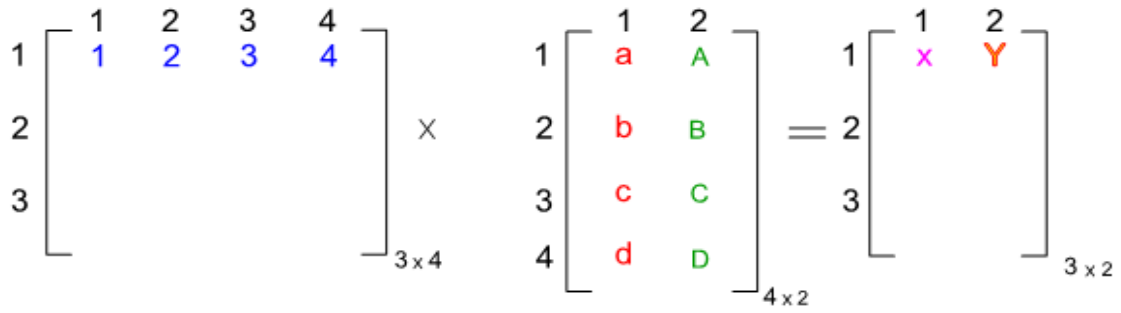
خانه های مخالف صفر را گویند که با فرمول 3n-2 نشان داده می شود.

محاسبه خانه های مخالف صفر در ماتریس L قطری:

$$nL - \frac{L^2-1}{4} = nL$$

ضرب ماتریس ها

ماتریس ها را می توان در یکدیگر ضرب کرد مشروط بر اینکه؛ تعداد ستون‌های ماتریس اول با تعداد سطرهای ماتریس دوم برابر باشد.



$[(1*a)+(2*b)+(3*c)+(4*d)] = X$

$[(1*A)+(2*B)+(3*C)+(4*D)] = Y$

Sparse Matrix

ماتریس خلوت (تنک یا کم پشت) یا 'ماتریس اسپارس 'ماتریسی است که اکثر عناصر آن صفر باشد.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- ✓ خلاصه شده ماتریس اسپارس: برای خلاصه کردن این ماتریس ها از آرایه دوبعدی استفاده می کنیم.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 2 \\ 0 & 0 & 0 & 4 \\ 3 & 0 & 0 & 0 \end{bmatrix}$$

تعداد خانه های غیرخالی | تعداد ستون ها | تعداد سطر ها

سطر کنترلی	۴	۴	۴
	۲	۲	۵
	۲	۴	۲
	۳	۴	۴
	۴	۱	۳

۵X۳

تعداد خانه های غیر خالی + یک سطر کنترلی ←

خلاصه شده ماتریس اسپارس همیشه نسبت به ماتریس اصلی یک سطر اضافه تر دارد.(سطرکنترلی)

بیشتر بدانید....؟!؟

حاصل جمع ، تفریق و ضرب دو ماتریس اسپارس الزاما" اسپارس نخواهد بود.

سطری که در آن صفر
باشد را حذف کرده و
ماتریس را مرتب می کنیم.

مثال:

حاصل این ماتریس اسپارس نخواهد بود.

$$\begin{bmatrix}1&0&0\\1&0&0\\1&0&0\end{bmatrix}\chi\begin{bmatrix}1&0&0\\1&0&0\\1&0&0\end{bmatrix}=\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

جمع ماتریس Sparce

در جمع ماتریس ها خانه ها نظیر به نظیر با هم جمع می شوند.

$$\begin{bmatrix}5&6&4\\1&3&7\\2&5&12\\4&3&-16\\5&2&10\end{bmatrix}+\begin{bmatrix}5&6&3\\2&5&-12\\3&4&18\\5&2&14\end{bmatrix}=\begin{bmatrix}5&6&5\\1&3&7\\2&5&0\\4&3&-16\\3&4&18\\5&2&24\end{bmatrix}$$

Row Remov

$$\begin{bmatrix}5&6&4\\1&3&7\\4&3&-16\\3&4&18\\5&2&24\end{bmatrix}=\begin{bmatrix}5&6&4\\1&3&7\\3&4&18\\4&3&-16\\5&2&24\end{bmatrix}$$

مرتب شده

در ضرب ماتریس ها خاصیت جا به جایی وجود ندارد.

$$A \chi B \neq B \chi A$$

در جمع ماتریس ها خاصیت جا به جایی وجود دارد.

$$A + B = B + A$$

ترانهاده Sparse Matrix [M^t , M^{-1}]

$$m = \begin{bmatrix} 5 & 6 & 4 \\ 1 & 3 & 7 \\ 3 & 4 & 18 \\ 4 & 3 & -16 \\ 5 & 2 & 24 \end{bmatrix} \rightarrow m^t = \begin{bmatrix} 6 & 5 & 4 \\ 3 & 1 & 7 \\ 4 & 3 & 18 \\ 3 & 4 & -16 \\ 2 & 5 & 24 \end{bmatrix} \xrightarrow{\text{مرتب شده}} \begin{bmatrix} 6 & 5 & 4 \\ 2 & 5 & 24 \\ 3 & 1 & 7 \\ 3 & 4 & -16 \\ 4 & 3 & 18 \end{bmatrix}$$

شبه کد ترانهاده کردن ماتریس دوبعدی :

```
For   j   1   to   m
      For   i   1   n
            Print   A [ j ] [ i ]
```

یا

```
For   i   1   to   m
      For   j   1   n
            Print   A [ i ] [ j ]
```

پشته یا stack

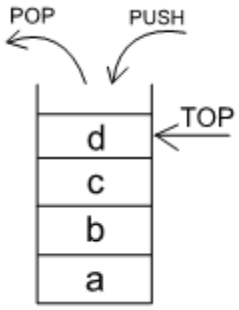
پشته لیست خطی است که عملیات حذف و اضافه تنها از یك طرف آن به نام Top صورت می گیرد. در هر لحظه فقط عنصر بالائی پشته قابل دسترس است .

آخرین عنصری که به پشته اضافه می شود اولین عنصری است که از آن حذف می شود. یعنی عناصر عکس ترتیب ورود از پشته خارج می شود به همین دلیل پشته یک ساختمان داده LIFO (last in first out)محسوب می شود .

می توان پشته را ساختمان داده FILO (first in last out)هم نامید زیرا اولین داده ای که در پشته ذخیره می شود آخرین داده ای است که خارج می شود .

در مبحث پشته کلمات کلیدی زیر وجود دارد :

- ✓ Push : عمل اضافه کردن عنصر جدیدی در پشته
- ✓ Pop : عمل حذف عنصری از پشته
- ✓ Top : آخرین عنصر پشته را مشخص می کند

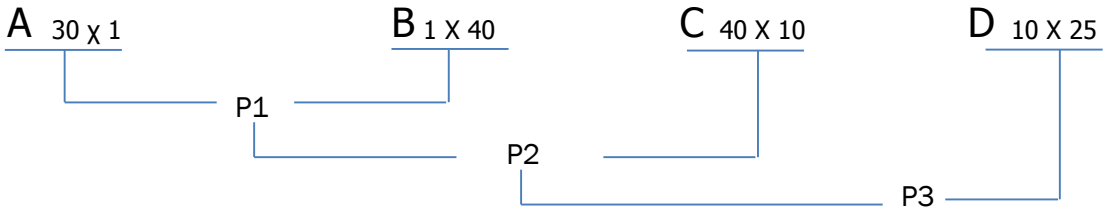


- ✓ TOPمی تواند دارای دو مقدار باشد
  - Top = n (پشته پر است)
  - Top = 0 (پشته خالی است)

بهینه کردن ضرب ماتریس ها :

۱- روش معمولی :

برای بهینه کردن ضرب ماتریس ها ، همواره ماتریس هایی را انتخاب می کنیم که اعداد داخلی آن ها بزرگتر باشد.



(30 X 40 X 1) + (30 X 10 X 40) + (30 X 10 X 25)=20.700

۲- حالت بهینه :

برای ماتریس بهینه اولویت با اعداد بزرگتر است .

A 30 X 1 B 1 X 40 C 40 X 10 D 10 X 25

$$(1 \times 10 \times 40) + (1 \times 25 \times 10) + (30 \times 25 \times 1) = 14000$$

زمانی که برای بهینه کردن ماتریس ها اولویت قرار می دهند حق جا به جایی نداریم.

چراکه در ضرب ماتریس ها جا به جایی نداریم.

شبه کد اضافه کردن مقداری به پشته ' push ' :

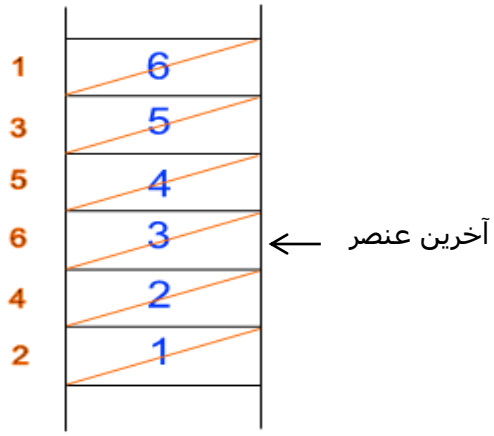
```
Array   s [ 0...n]
   Top = 0 ...n
   Top = 0;
   If  top==0    'stack  is  empty '
   If  top==n    'stack  is  full '
```

```
Int  Push  ( void  x )
 {
    If  ( top ==n )
       Stack  is  full
   Else
       S [ ++ top ] = x
```

شبه کد حذف کردن مقداری از پشته ' pop ' :

```
Array   s [ 0...n]
Int  pop  ( )
 {
If  (top==0 )
   'stack  is  empty '
Else
   X = s [ top - - ]
```

یک stack با حد اکثر تعداد خانه های ۶ موجود است. در هر خانه اندیس آن خانه ذخیره شده است. این پشته باهر بار pop کردن سر و ته می شود. آخرین عنصری که pop می شود چیست؟

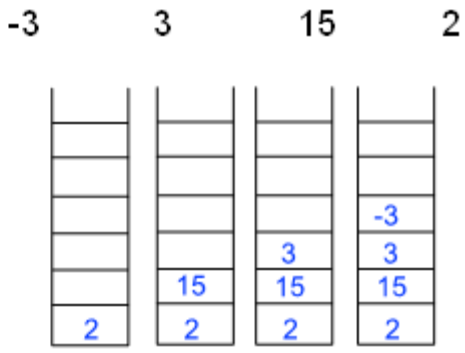


فرض کنید به یک stack ۶خانه اختصاص داده ایم.در ابتدا stack خالی است.

الف – خروجی برنامه زیر چیست؟

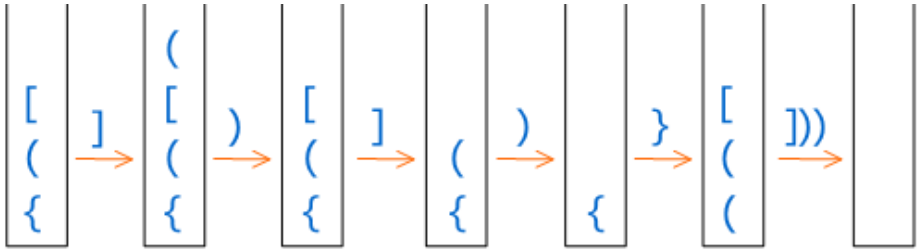
ب – حد اکثر تعداد خانه هایی که برا ی کد زیر نیاز است چقدر است؟

```
1 / set  A=2 , B=5 ;
2 / push (stack , A)
    push (stack , B+10)
    push ( stack , 3 )
    push ( stack , A-b )
3 / repeat  while  top <>0 pop ( stack , item )
      write : item ;
4 / return ;
```



تعداد خانه های مورد نیاز ۴ است.

عبارت { x + (y – [a+b] * c – [(d + e)]) } / (j – (k – [L – n])) را درنظر بگیرید .می خواهیم با استفاده از یک پشته بررسی کنیم که آیا پرانتزها ، کروشه ها و آکولادها به درستی تطبیق می شوند یا خیر؟ پشته موردنظر حد اقل باید گنجایش چند عنصر را داشته باشد؟



پشته حد اکثر باید دارای ۴ خانه باشد.

ورودی A , B , C , D , E , F به یک پشته را در نظر بگیرید . کدام یک از پشته های زیر مجاز است؟

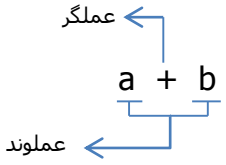
الف

ب

ج

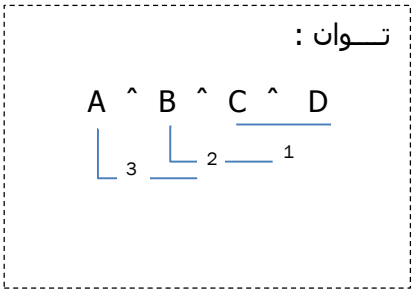
ارزشیابی عبارت های محاسباتی

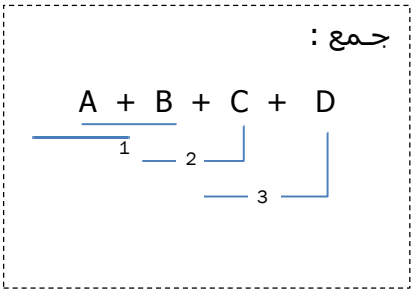
عملگر های محاسباتی ،اعمال محاسباتی را روی عملوند ها انجام می دهند.



ترتیب اولویت عملگر ها:

نام	عملگر
پرانتز	( )
مثبت و منفی	+ و -
توان	^
ضرب و تقسیم	* و /
جمع و منها	+ و -
باقیمانده	Mod





نمایش پسوندی و پیشوندی

در دنیای جبر و منطق اعمال ریاضی رابه دو فورم نمایش می دهیم:

۱- میانوندی ' in fix '

۲- لهستانی 'Post fix '

لهستانی خود شامل دو بخش می باشد

پیشوندی

پسوندی

← در مدل لهستانی اولویت عملگر ها هیچ گونه معنا و مفهومی ندارد.

← مدل لهستانی تنها شامل عملیات و عملوند ها می باشد.(نیاز به پرانتز گذاری ندارند.)

- نماد گذاری لهستانی:

Infix : عملگر بین دو عملوند

Postfix : عملگر بعد از دو عملوند

Prefix : عملگر قبل از دو عملوند

عبارت زیر را به پسوندی و پیشوندی تبدیل کنید .

(A – B / (C * D ^ E))

Post fix :

پسوندی: در این روش ابتدا پرانتز داخل را در نظر می گیریم. (دارای دو علامت * و ^ می باشد)

اولویت ^ بالاتر است. D E ^

حال عبارت قبلی را با عملوند بعد در نظر می گیریم. C D E ^ *

در پرانتز بعد دارای دو علامت – و / می باشیم.با در نظر گرفتن تقسیم : B C D E ^ * /

و در آخر با درنظر گرفتن عملگر - عبارت A B C D E ^ * / - حاصل می شود.

A – B / C * D ^ E

pre fix :

در pre fix با توجه به اینکه in fix ما پرانتز دار است ،لازم است عبارت را پرانتز گذاری کنیم.

گام ۱ – در پرانتز داخل ذارای دو علامت ^ و * هستیم. با توجه به اینکه اولویت ^ بالا تر است،فلذا ^ را در نظر می گیریم.

^ D E

گام ۲ – حال عبارت بعد را در نظر می گیریم.عبارت را نوشته و عملگر قبل از آن قرار می گیرد.

* C ^ D E

گام۳- حال B تقسیم بر یک عبارت می شود.

/ B * C ^ D E

در آخر A منهای کل عبارت قبل می شود.

- A / B * C ^ D E

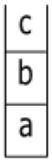
همواره آخرین علامت در Post fix و In fix یکسان خواهد بود.

عبارت Post fix زیر را به میانوندی (in fix) و پیشوندی (pre fix) تبدیل کنید.

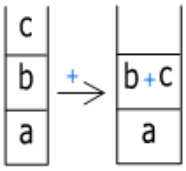
A b c + * d e / -

: ' in fix ' به ' Post fix '

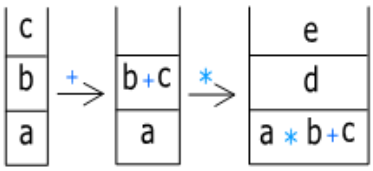
روش حل: هر گاه حروف بود درون پشته قرار می گیرد. a و b و c در پشته قرار می گیرد.



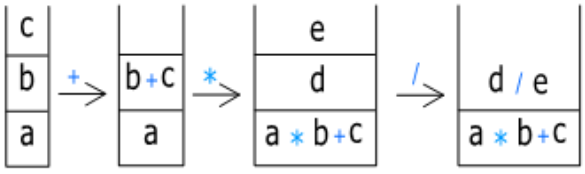
زمانی که به یک علامت برسیم،دو مقدار از بالای پشته برداشته شده و علامت ما بین دو مقدار قرار می گیرد.



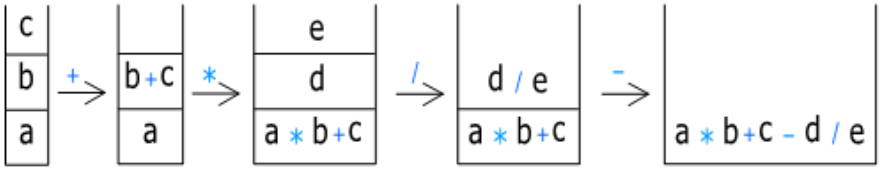
در مرحله بعد به ضرب می رسیم.مانند مرحله فوق ؛ دو مقدار از بالای پشته برداشته شده و علامت ما بین دو مقدار قرار می گیرد. و دو مقدار وارد پشته می شود.



دو حرف d و e وارد پشته می شوند. حال به عملگر / می رسیم.مانند مراحل فوق دو مقدار از بالای پشته برداشته شده و علامت ما بین دو مقدار قرار می گیرد. و دو مقدار وارد پشته می شود.



در ادامه به عملگر – می رسیم و طبق مراحل فوق عمل می کنیم.



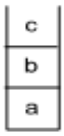
' Post fix ' به ' pre fix ' :

در این حالت عملگر ها قبل از عملوند ها قرار می گیرند.

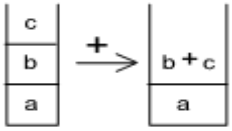
عبارت پسوندی زیر را به پیشوندی تبدیل کنید.

A b c + * d e / -

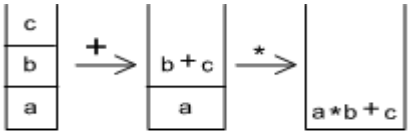
گام ۱- از سمت چپ به ترتیب حروف (عملوندها) را در پشته قرار می دهیم.



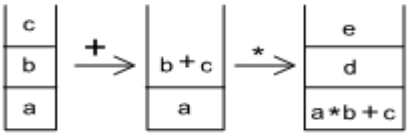
گام ۲ – زمانی که به عملگری برسیم دو عبارت از پشته خارج کرده و عملگر بین دو عبارت قرار می گیرد.



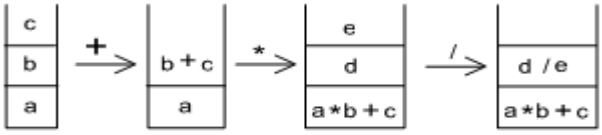
گام ۳- حال به عملگر ضرب می رسیم. دو عبارت بالای پشته را خارج کرده و عملگر بین دو عبارت قرار می گیرد.



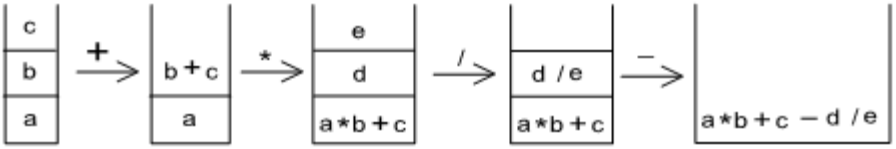
گام ۴ – حال d و e وارد پشته می شوند.



گام ۵ - حال به عملگر تقسیم می رسیم. دو عبارت بالای پشته را خارج کرده و عملگر بین دو عبارت قرار می گیرد.



گام۶- در آخر عملگر منها وارد شده و مانند مراحل فوق عمل می کنیم.



در تبدیل عبارت پسوندی به پیشوندی یا پسوندی به میانوندی یا ... همواره جای عملگرها ثابت است .

بیشتر بدانید...؟!¿

اگر می خواهیم یک عبارت pre fix را به post fix یا in fix تبدیل کنیم و از "پشته "برای ذخیره سازی عبارت استفاده کنیم،از سمت راست پیمایش را شروع می کنیم.

عبارت پیشوندی ' pre fix ' را به پسوندی ' post fix ' تبدیل کنید.

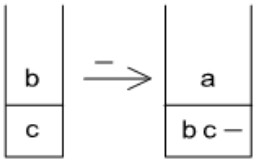
+ + a / b – c d / - a b - + c * d 5 / a – b c

روش حل:

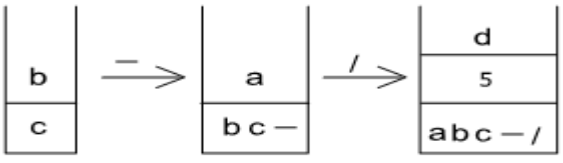
گام ۱ : عبارت ها را از سمت راست وارد پشته می کنیم.

گام ۲ : به علامت (عملگر) که رسیدیم، دو عبارت را از پشته خارج کرده و علامت را قبل از آن قرار می دهیم.

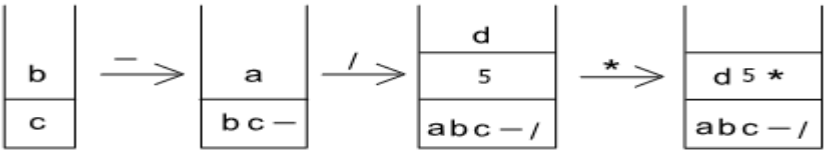
حال a وارد پشته می شود.



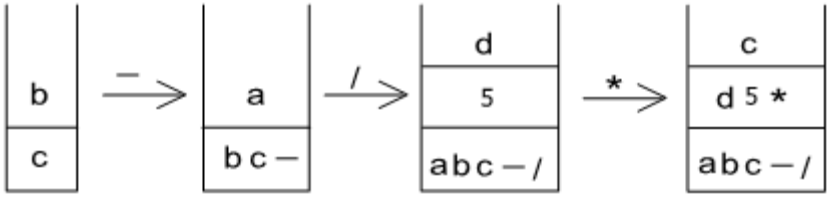
حال به یک علامت می رسیم.(هرگاه به علامتی می رسیم دو عبارت از پشته خارج شده و علامت بعد از آن قرار می گیرد.)عدد 5 و d وارد پشته می شوند.



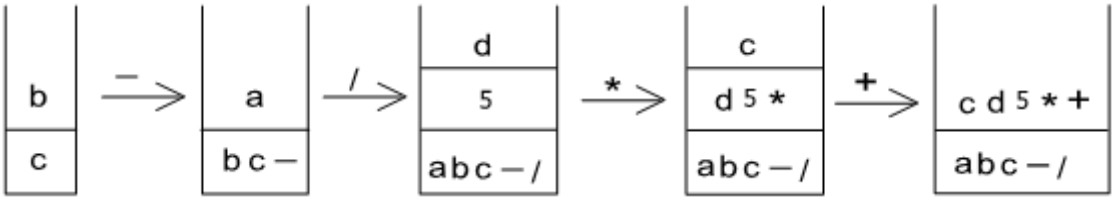
حال به عملگر * می رسیم. دو عبارت از پشته خارج شده و علامت بعد از آن قرار می گیرد.



حال c وارد پشته می شود.



به عملگر + می رسیم. دو عبارت از پشته خارج شده و علامت بعد از آن قرار می گیرد.



Post fixed : c d 5 * + a b c - / -

صف Queue

صف لیستی است مرتب که عناصر از یک سمت آن به نام ته صف (rear) به آن وارد و از جلوی صف (front) خارج می شوند.

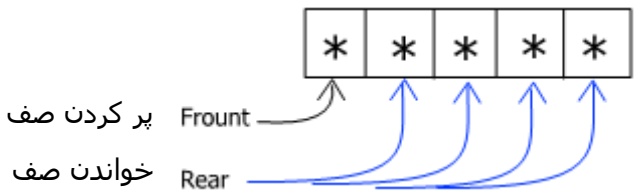
صف را لیست FIFO (First In First Out) یا LIFO (Last In First Out)می نامند.

صف Queue

آرایه : ساختمان داده ای از نوع استاتیک می باشد.(دارای ابتدا و انتها)

لیست پیوندی : می توان همواره مقداری را به آن افزود.

شبه کد یا الگوریتم پیاده سازی صف :



Q (n)

F , r = 0 ... n دامنه صف

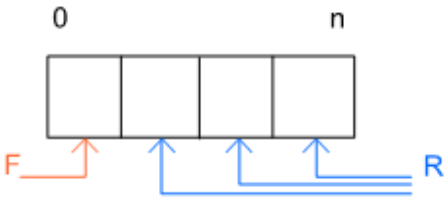
F, r =0 خالی بودن صف

If Q=R Q is empty

If R=N Q is full

شبه کد اضافه کردن به صف :

```
Add  q (x)
{
   If  R==n ⟶ A  is  full
  Else
   q [++r]=x ;
}
```




اگر F و R به n برسند، صف پر است.

ساختمان داده های حذف کردن از صف

مشکل صف خطی این است که آرایه فقط یکبار قابل استفاده است.یعنی ممکن است آرایه جایی خالی داشته باشد اما صف پر اعلام شود.

برای رفع این مشکل از صف حلقوی استفاده می کنیم.

```
Del  q ( )
{
   If  f==n ⟶  Q  is  empty;
  Else
   Return   q [ ++F ];
}
```

ساختمان داده های صف حلقوی

```
Q( n )
{
F , r = 0...n
F=r=0
If   f=r ⟶ Q  is  empty
If  f= (r+1) mod  n ⟶ Q  is   full
}
```

اضافه کردن به صف آرایه

```
Add  q   R ( x )
{
  If  f = ( r+1 )  mod  n ⟶ Q  is  full
Else
   {
     R =  ( r + 1) mod  n
     QR  [R]=x ;
     }
}
```

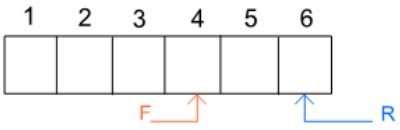
حذف کردن از صف حلقوی

```
Del   QR ( )
{
  If  F = R ⟶ q  is  empty
 Else
   {
     F = ( f + 1 ) mod  n
     Return  QR  [ F ]
    }
}
```

تعداد عناصر در صف معمولی از فرمول زیر محاسبه میشود:

R – F
↳ تعداد خانه های پر

مثال :



6 - 3 = 3

تعداد عناصر در صف حلقوی :

If $f \le R \longrightarrow R - F$

If $f > R \longrightarrow n - (f - r)$

فرض کنید یک صف حلقوی صد عنصری موجود باشد. اگر R=1 و F=100 باشد ،تعداد عناصر موجود در صف را مشخص کنید.

F= 100

R=1

فرمول $\longrightarrow n - (F - R)$

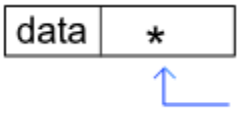
$$100 - (100 - 1) = 1$$

ساختارکلی LINK LIST در حافظه

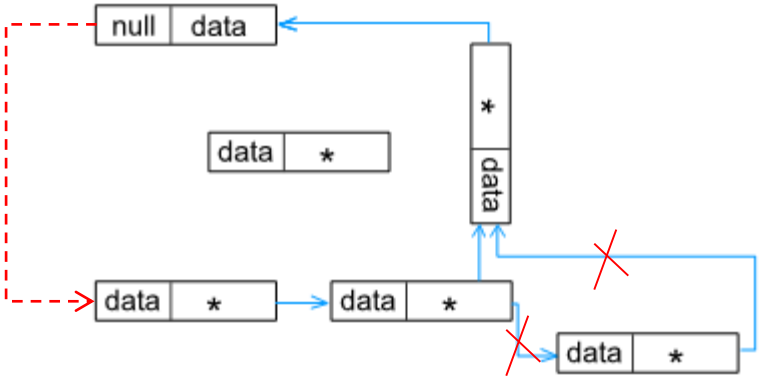
برای ذخیره سازی داده ها می توان از آرایه و لیست پیوندی استفتده کرد.

لیست پیوندی بر خلاف آرایه یک ساختار غیر خطی دارد . به این معنا که عناصر لیست پیوندی می توانند در هر مکانی در حافظه کامپیوتر قرار گیرند و لزوما مانند آرایه پشت سر هم نمی باشد.(با استفاده از اشاره گر به هم اشاره می کنند.)

لینک لیست ها از نوع ساختمان داده های داینامیک بوده و فاقد محدودیت می باشند.



(محل قرار گیری آدرس خانه بعد در حافظه)



با افزودن یک مسیر دیگر مدار یک طرفه خواهد شد.

اضافه کردن به انتهای لیست :

دلیل Dynamic بودن لیست پیوندی:

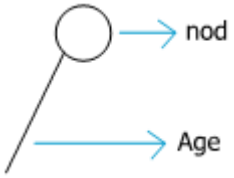
هرگاه در آرایه خانه ها بصورت متوالی نباشند، نمی توان مقداری را در آرایه درج کرد.

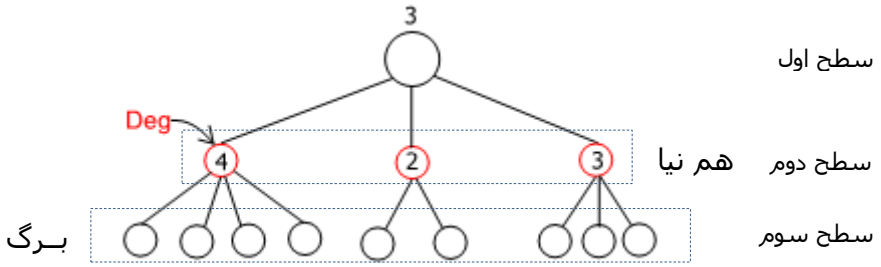
اما؛ در link list یک مکان از حافظه را در نظر گرفته و به آدرس خانه مد نظر در آرایه ارجاع می دهد.

فصل سوم : درخت ' tree '

درخت یک حالت خاص از یک گراف می باشد.

درخت یک لیست متناهی است که دارای تعدادی گره ' node ' و یال یا ' Age ' می باشد که ؛ همه به ریشه (گره والد) متصل هستند.





با توجه به اینکه بالا ترین درجه درخت فوق ۴ است فلذا ، این درخت چهارتایی است.

برگ :گره های آخر یک درخت را که فاقد فرزند هستند، برگ می نامند.

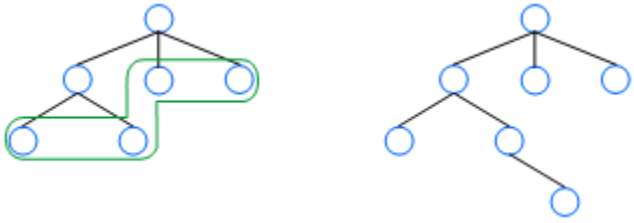
درجه گره : تعداد فرزندان یک گره را درجه گره گویند.

درجه درخت : بزرگ ترین درجه فرزندان یک درخت را گویند.

هم زاد یا هم نیا : فرزندان یک گره را گویند.

مسیر : دنباله ای از یال های متوالی یک گره که ما را به گره مقصد هدایت کند.

درخت متوازن : درختی است که اختلاف سطح برگ های آن حداکثر یک است.

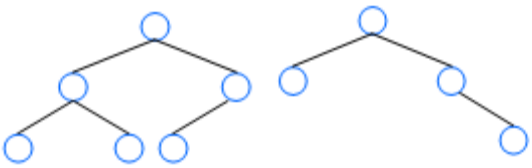


متوازن است　　متوازن نیست

درخت کاملا متوازن : درختی است که اختلاف سطح برگ های آن صفر است.

درخت k تایی : درختی است که درجه گره های آن کوچکتر یا مساوی k (درجه گره درخت) باشد.

درخت مرتب : درختی است که برگ ها آن از چپ به راست قرار گرفته باشند.

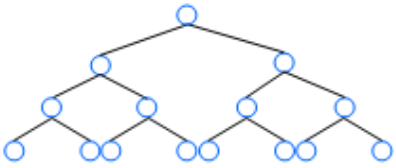


مرتب　　غیر مرتب

درخت دودویی : درخت مرتبی است که درجه گره های آن کوچکتر یا مساوی دو باشد.

به بیان دیگر ؛ حد اکثر تعداد یال های هر گره آن دو می باشد.

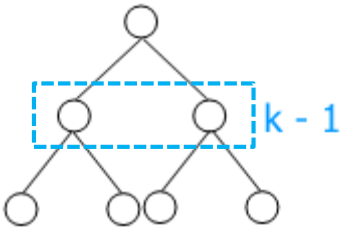
درخت دودویی پر : درخت دودویی مرتبی است که بجز برگ های آن تمام گره ها دقیقا دارای دو فرزند باشند و کاملا متوازن باشد.



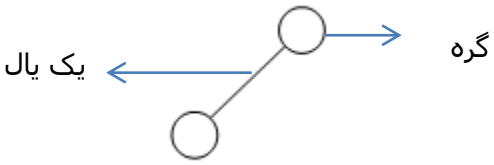
درخت توسعه یافته یا محض ' two tree ': درخت دودویی است که درجه کلیه گره های آن بجز برگ ها دقیقا دو باشد.

یک درخت پر می تواند محض باشد.

درخت دودویی کامل : درخت دودویی است به عمق k ،که تا سطح k-1 پر است.و گره های موجود در سطح kام آن ،از چپ به راست قرار گرفته باشد.



تعداد یال های یک درخت ، همواره یکی کمتر از گره ها می باشد.



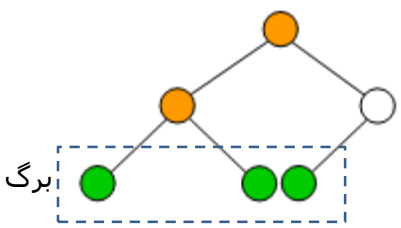
حد اکثر تعداد گره ها برای ایجاد یک درخت ،یک است و برای ایجاد یک درخت دودویی صفر است.

- تعداد برگ ها در یک درخت k تایی کامل با n گره برابر است با ؛ $n - \frac{n-1}{k}$.

تعداد گره های یک درخت سه تایی که تعداد سطح های آن ۱۰ است ،چقدر است؟

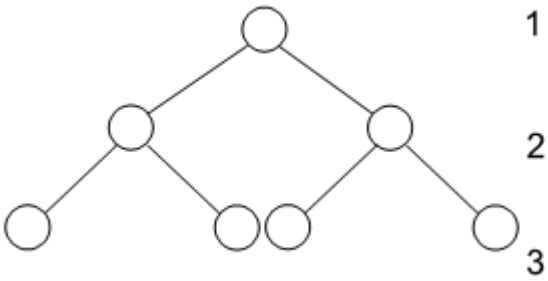
$$10 - \frac{10-1}{3} = 10 - \frac{9}{3} = 7$$

- در یک درخت دودویی همواره تعداد برگ ها یک واحد بیشتر از تعداد گره های درجه دو است.

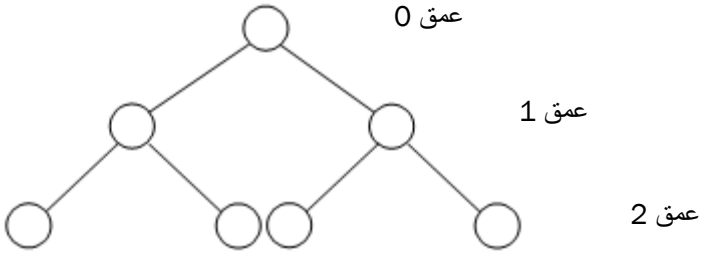


- حد اکثر تعداد گره ها در سطح i ام یک درخت دودویی برابر است با : 2^i - 1

مثال:

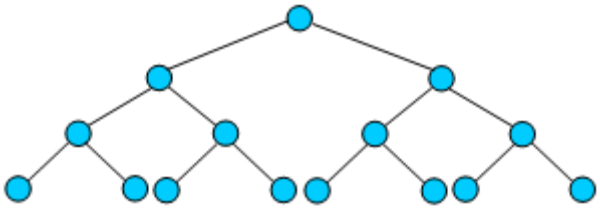


- حد اکثر تعداد گره ها در یک درخت دودویی در عمق k برابر است با : 2^k - 1



حد اکثر تعداد گره ها در عمق چهارم چقدر است؟

2^4 -1 =15



- عمق یک درخت " دودویی پر "با n گره برابر است با :

فرمول ⟵ $\log_2(n+1)$ $\log_2(15+1) = 4$

- تعداد برگ ها در یک درخت دودویی کامل برابر است با :

$$N - \left\lceil \frac{n-1}{2} \right\rceil$$

پیاده سازی درخت

درخت به دو صورت قابل پیاده سازی است :

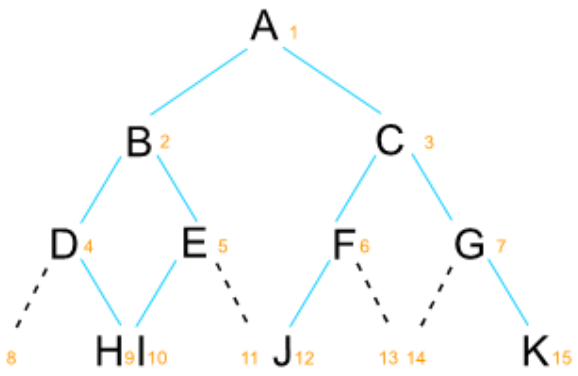
۱- آرایه

۲- لیست پیوندی

۱- آرایه

برای درخت هایی که دارای گره زیاد(پر یا کامل)هستند ، آرایه مناسب خواهد بود.چرا که تعداد خانه های خالی موجود درآرایه کمتر است.

فلذا ایجاد یک درخت کامل با استفاده از آرایه اگر درخت پر یا کامل باشد مناسب است. اما ؛اگر از نوع دیگر درخت مثلا : اوریب به راست باشد مقرون و به صرفه نخواهد بود.

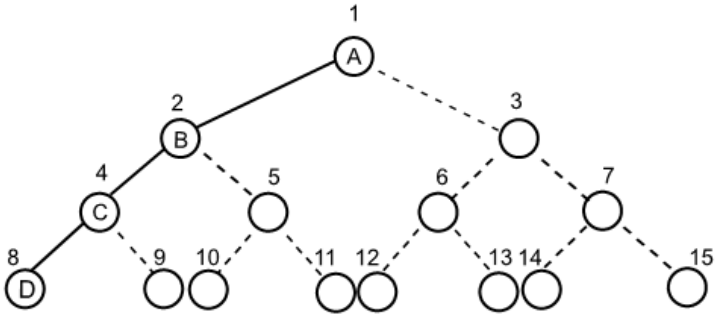


برای قرار دادن درخت فوق در آرایه لازم است برای هر گره ' NOD ' آن یک عدد در نظر گرفت.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B	C	D	E	F	G		H	I		J			K

درخت اریب

درخت اریب به چپ زیر را در نظر بگیرید.



برای رسم آرایه این درخت لازم است به تعداد یک درخت کامل و پر برای این درخت خانه در نظر گرفت.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	B		C				D							

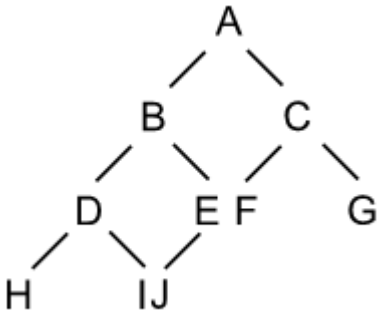
با توجه به مثال فوق می توان نتیجه گرفت در درخت اریب استفاده از آرایه مقرون و به صرفه نمی باشد.

پیمایش درخت :

درخت به دو حالت پیمایش می شود
- سطحی ' BFS '
- عمقی ' DFS '

پیمایش سطحی توسط یک صف صورت می گیرد. و پیمایش عمقی توسط یک پشته پیاده سازی خواهد شد.

پیمایش سطحی ' BFS '



سطحی : A B C D E F G H I J

پیمایش عمقی ' DFS '

پیمایش عمقی خود به سه حالت انجام می شود:

۱- میانوندی LVR

۲- پیشوندی VLR

۳ – پسوندی LRV

۱- میانوندی LVR

از آنجایی که درخت ها بصورت بازگشتی عمل می کنند در هر مرحله تا یک گام قبل را در نظر می گیریم.

در ابتدا ریشه A را در نظر گرفته که دارای دو فرزند B و C می باشد. فرزند سمت چپ را در نظر می گیریم.

حال B خود دارای دو فرزند D و E می باشد. فرزند سمت چپ را در نظر می گیریم.

D نیز خود دارای دو فرزند I و H می باشد. آخرین فرزند سمت چپ را به خروجی انتقال می دهیم.

H

حال ریشه را در نظر می گیریم.

H D

حال فرزند سمت چپ را در نظر می گیریم.

H D I

زمانی که به انتها رسیدیم به گره والد رفته و سمت راست والد را در نظر می گیریم.

H D I B

حال سمت راست گره B را در نظر گرفته و آن را تا انتها پیمایش می کنیم.

H D I B J

از آنجایی که J هیچ فرزندی ندارد به والد آن باز می گردیم.

H D I B J E

از آنجایی که گره B از قبل خوانده شده است به والد آن باز می گردیم.

H D I B J E A

حال سمت راست را در نظر می گیریم. (گره C خود ریشه می باشد.)

سمت چپ گره ریشه F بوده که هیچ فرزندی ندارد فلذا خود به خروجی انتقال می یابد.

H D I B J E A F

حال به والد رفته و آن را به خروجی ارسال می کنیم.

H D I B J E A F C

و در آخر گره سمت راست به خروجی ارسال می شود.

H D I B J E A F C G

۲- پیشوندی VLR

در VLR اولویت با ریشه است.(اول ریشه خوانده می شود.)

A خوانده می شود.از آنجایی که A خود ریشه است به خروجی می رود.

A

حال برسی می شود؛ آیا A فرزند چپ دارد ؟ بله

آیا ریشه است؟بله. فلذا به خروجی می رود.

A B

مجددا مراحل فوق برسی می شوند

A B D

طبق شرط فوق :

A B D H

حال سمت راست درخت در نظر گرفته می شود.

A B D H I

از آنجایی که Iفرزند دیگری ندارد به ریشه قبل از خود می رود. حال والد (B) را در نظر گرفته و سمت راست را پیمایش می کنیم.

A B D H I E

E خود دارای یک فرزند می باشد.

A B D H I E J

حال به والد کل بازگشته و سمت راست را پیمایش می کنیم.

A B D H I E J C

C دارای دو فرزند می باشد.(اولویت با فرزند سمت چپ می باشد)

A B D H I E J C F

و در آخر

A B D H I E J C F G

۳- پسوندی LRV

در ابتدا A را در نظر می گیریم.A خود دارای چند ریشه می باشد.(ریشه سمت چپ را پیمایش می کنیم تا به آخرین ریشه سمت چپ که هیچ فرزندی ندارد برسیم.)

H به خروجی می رود.

H

به ریشه قبل خود بازگشته و سمت راست را پیمایش می کند.

H I

حال ریشه را چاپ می کند.

H I D

از گره Dبه والد خود بازگشته و سمت راست را پیمایش می کند.

حال به گره E رسیده ایم. گره E خود دارای فرزند می باشد.فرزند E (J) را چاپ می کند.

H I D J

حال ریشه را چاپ می کند.

H I D J E

به ریشه قبل باز گشته و B را چاپ می کند.

H I D J E B

حال سمت راست گراف را پیمایش می کنیم.گره C خود دارای دو فرزند می باشد که، فرزند سمت چپ را در نظر می گیریم.

H I D J E B F

حال به ریشه بازگشته و سمت راست را پیمایش می کنیم.

H I D J E B F G

به ریشه قبل باز گشته و C را چاپ می کند.

H I D J E B F G C

و در آخر والد کل را چاپ می کنیم

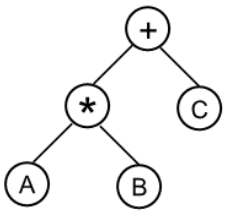
H I D J E B F G C A

ترتیب ملاقات برگ ها در هر سه الگوریتم میانوندی ، پیشوندی و پسوندی یکسان است.

درخت Parse

درخت Parse در کامپایلرها مورد استفاده قرار می گیرد.(برای عبارات ریاضی و سینتکس های برنامه نویسی)

مدل ریاضی:



میانوندی : A * B + C

برای رسم یک درخت به دو پیمایش که حد اقل یکی از آن ها میانوندی باشد نیاز است.

اگر LRV باشد : پیمایش ←— صورت می گیرد.

اگر VLR باشد : پیمایش —→ صورت می گیرد.

مثال :

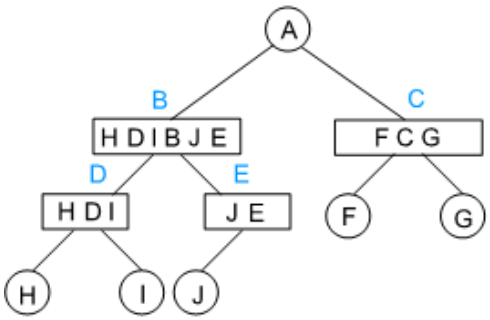
LVR : h d i b j e a f c g

VLR : a b d h i e j c f g

اولین ورودی در vlr ' a ' می باشد .(نکته : همیشه در VLR اولین ورودی ریشه است و در LRV آخرین ریشه)

هر چه در سمت راست گره باشد در یک گره و هر چه در سمت چپ باشد در یک گره دیگر قرار می گیرد.

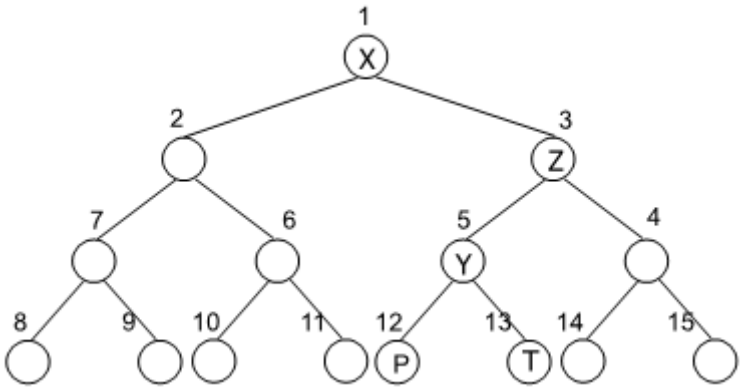
حال حرف b را در نظر می گیریم و طبق گره فوق عمل می کنیم.



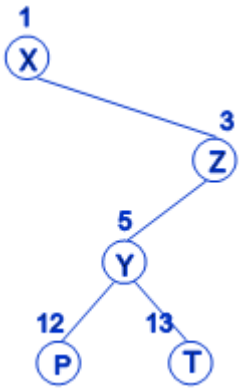
درخت زیر را در نظر گرفته و پیمایش میانوندی ' inorder ' آن را بنویسید.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x		z			y						P	t		

در ابتدا درخت آرایه فوق را رسم می کنیم.

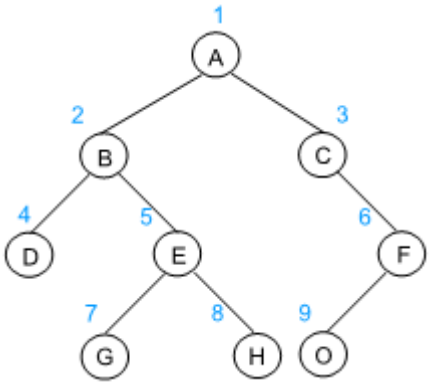


حال گره های اضافه را حذف می کنیم.



LVR : X P Y T Z

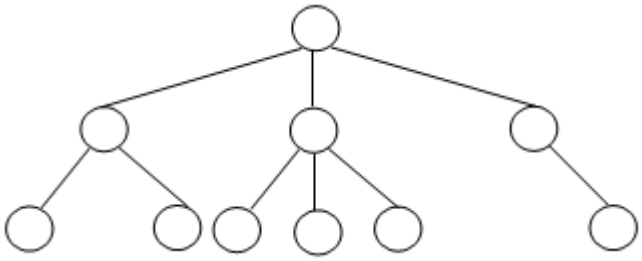
پیمایش پسوندی درخت زیر را بنویسید.



	in to	left	right
Root → 1	A	2	3
2	B	4	5
3	C	0	6
4	D	0	0
5	E	7	8
6	F	9	0
7	G	0	0
8	H	0	0
9	O	0	0

درخت عمومی

دارای درجه ثابت و معینی نمی باشد.

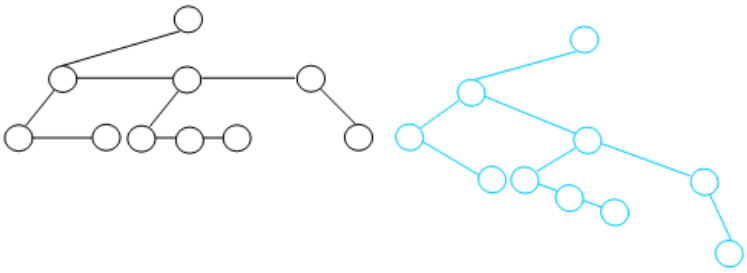


تبدیل درخت عمومی به درخت دودویی

روش اول : فرزند چپ برادر راست

گام ۱ : کلیه ی گره های همزاد را به یکدیگر وصل می کنیم.

گام ۲ : ارتباط تمامی گره ها به پدرشان را به جزء گره سمت چپ را قطع می کنیم.



فصل چهارم : گراف

مختصری از تعاریف گراف :

گراف :

مجموعه ای از روس ها و یال ها که هم دارای جهت هستند و هم بدون جهت .که در برخی گراف ها یال ها دارای وزن هستند.

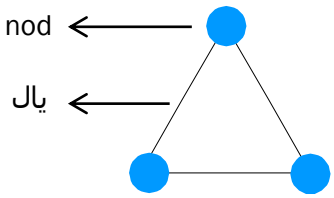
یک گراف شامل دو مجموعه است؛ مجموعه غیر تهی از گره ها یا رئوس (vertex) و مجموعه ای از یال ها (edge) که راس ها را به هم متصل می کنند.

مثال: می توان شهر های یک کشور را رئوس و جاده های بین آن ها را یال های یک گراف تصور کرد.

به هر راس یا هر یال گراف نامی اختصاص داده می شود.

node :

یال : به مسیر ارتباطی بین دو گره یال گفته می شود.



گراف جهت دار : اگر یال های گراف جهتی داشته باشند،گراف جهت دار است.

گراف بدون جهت : اگر یال های گراف جهتی نداشته باشند،گراف بدون جهت است.

گراف وزن دار :گرافی است که روی هر یال آن عدد یا علامتی باشد که نشان دهنده ی وزن آن یال است.

گراف کامل : گرافی است که بدون درنظر گرفتن جهت بین هر دو گره یک یال وجود داشته باشد.

درجه گراف جهت دار: درجه ی راس یک گراف جهت دار یال های ورودی و خروجی به آن راس هستند.

درجه گراف بدون جهت : درجه ی راس های یک گراف، در یک گراف بدون جهت برابر است با تعداد یال هایی که از آن راس می گذرد.

گراف تهی : گرافی که فقط راس یا **nod** داشته باشد،گراف تهی " { } " یا " Ø " می باشد.

مسیر در یک گراف : در یک گراف مسیر وجود دارد اگر و فقط اگر از هر راس به راس دیگر حداقل یک راه وجود داشته باشد.

مدار : یک گراف بصورت یک مدار تصور می شود اگر و فقط اگر پیمایش را از یک راس آغاز ،و پس از طی مسیری دوباره به راس آغازین باز گردیم.

گراف همبند جهت دار : گرافی است که با حذف جهت های آن ، یک گراف هم بند بدون جهت بوجود می آید.

گراف همبند بدون جهت: گرافی است که بین هر دو راس آن مسیر وجود داشته باشد.

گراف همبند قوی :گرافی است که به ازای هر دو راس دلخواه مثل **A** و **B** هم مسیر از **A** به **B** موجود باشد هم **B** به **A** .

گراف مکمل : برای یافتن مکمل گراف **G** ،یال های **G** را حذف کرده و یال هایی که در گراف کامل هستند را رسم می کنیم.

گراف k تایی : گرافی که بین هر دو راس بیش از یک یال باشد ،گراف چند گانه نامیده می شود.

مدار اویلری : مداری است که از هر یال یک و فقط یکبار عبور کند و به خود باز گردد.

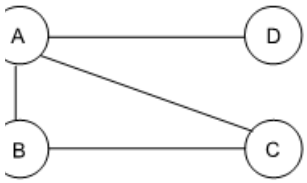
مسیر اویلری : مسیری است که از هر یال یک و فقط یکبار عبور کند. (غیر بازگشتی)

مدار هامیلتونی : مداری است که از هر یال یک و فقط یکبار عبور کند و به خود باز گردد.

مسیر هامیلتونی : مسیری است که از هر گره یک گراف یک و فقط یک بار عبور کند.

حلقه : مسیری که راس اول و آخر آن یکی باشد.

درجه هر گره: تعداد یال های ورودی یا خروجی از هر گره را گویند.



Deg (A) = 3

Deg (D) = 1

پیمایش گراف

گراف ها به دو صورت پیمایش می شوند

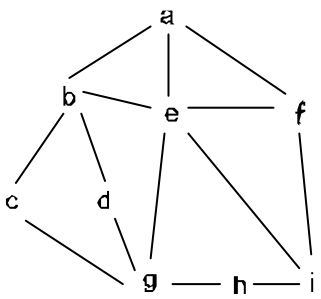
۱- سطحی 'BFS'

۲- عمقی 'DFS'

لازم به ذکر است اگر در پیمایش گراف نقطه شروع تعیین نشده باشد اولویت با گره ای است که کوچکترین اسم را دارد.

سطحی ' BFS '

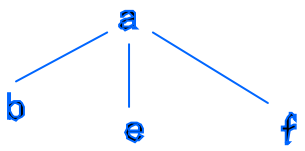
اگر گراف زیر را داشته باشیم



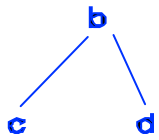
برای پیمایش یک گراف به روش سطحی بصورت زیر عمل می کنیم ؛

- ✓ هر گره که در نظر گرفته می شود تمامی فرزندان آن گره نوشته می شوند.

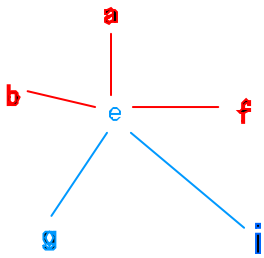
گام ۱: در این نوع پیمایش اولین گره (گره شروع) ' a ' را در نظر گرفته و تمامی ارتباطات گره a را می نویسیم.



گام ۲: حال گره b را در نظر گرفته و تمامی ارتباطات آن را می نویسیم.



گام ۳ :حال گره e را در نظر می گیریم. (از گره هایی که از قبل خوانده شده اند صرفه نظر می کنیم)

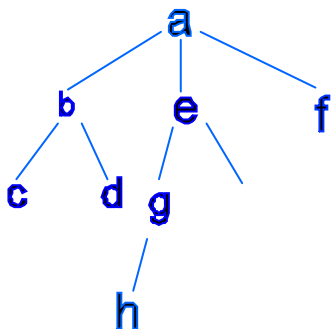


گام ۴ : حال به f می رسیم . از آنجایی که تمامی مسیر های موجود از f به مابقی گره ها از قبل نوشته شده اند از آن صرفه نظر می کنیم.

گام ۵ : حال به c می رسیم . از آنجایی که تمامی مسیر های موجود از c به مابقی گره ها از قبل نوشته شده اند از آن صرفه نظر می کنیم.

گام ۶ : حال به d می رسیم . از آنجایی که تمامی مسیر های موجود از d به مابقی گره ها از قبل نوشته شده اند از آن صرفه نظر می کنیم.

گام ۷ : گره g با e و h در ارتباط است .از آنجایی که گره e از قبل ذکر شده از آن صرفه نظر کرده و h را در نظر می گیریم.

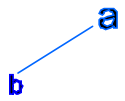


درخت پوشای سطحی

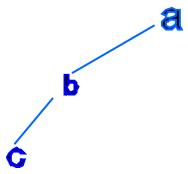
' BFS ' سطحی : a b e f c d g i h

عمقی ' DFS '

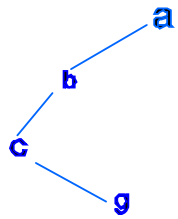
گام ۱ : در این نوع پیمایش اولین گره (گره شروع) را در نظر گرفته و آن گره را تا عمق (انتها) پیمایش می کنیم.(اولویت با ارتباط کوچکتر می باشد.)



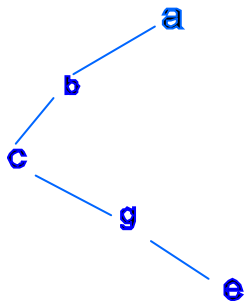
گام ۲ :گره b با c و d و e در ارتباط است.در اینجا گره c را در نظر می گیریم.



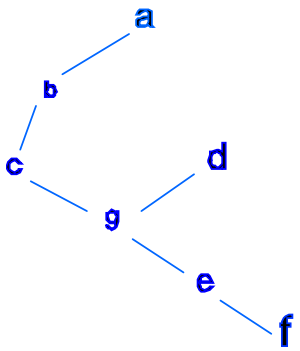
گام ۳ : گره c با گره g در ارتباط است.



گام ۴ : گره g با e و h در ارتباط است .بر اساس تعاریف فوق گره e را در نظر می گیریم.

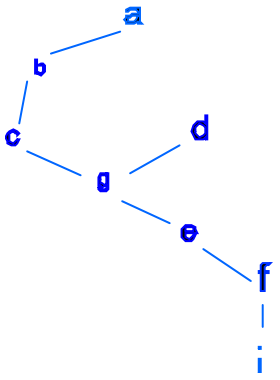


گام ۵ :گره e با گره های a و b و f و g و I در ارتباط است ولی؛ از آنجایی که گره های a و b و c و g و d از قبل پیمایش شده اند و f از i کوچکتر است ،f را در نظر می گیریم.

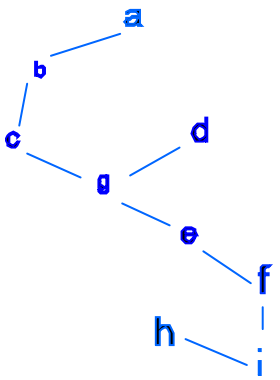


گام۶- گره f با a در ارتباط است ولی از آنجایی که از قبل پیمایش شده است از آن صرفه نظر می کنیم.

گام ۷- حال گره i را در نظر می گیریم.



گام۸- گره i با f و e و h در ارتباط است.از آنجایی که e و f از قبل پیمایش شده اند h را در نظر می گیریم.



درخت پوشای عمقی ←

درخت پوشا

درختی است که کلیه ی گره های رئوس گراف را با حد اقل یال ممکن در بر گیرد.

درخت پوشای حداقل ' minimum '

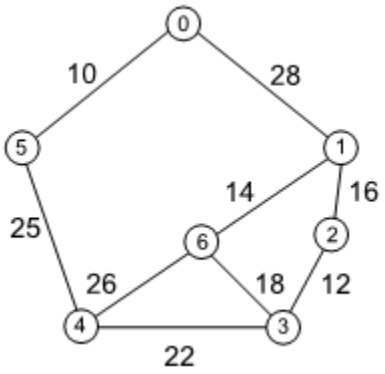
درخت پوشایی است که مجموع وزن یال های آن حداقل باشد.

الگوریتم
- ۱- راشال (راسل ، کروسال ،کروشال)
- ۲- پریم ' Prim '

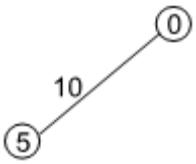
۱- راشال

در این روش در هر مرحله یالی که کمترین هزینه را دارد انتخاب نموده ،و درصورتی که باعث بوجود آمدن حلقه نشود آن را رسم می کنیم.

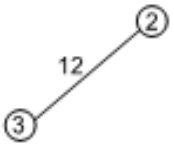
این عمل را تا بدست آمدن یک درخت پوشا ادامه می دهیم.



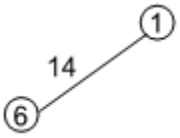
گام۱: در این مرحله از راس ۱ به راس ۵ می رویم ،با هزینه ی ۱۰.



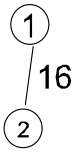
گام ۲ : در این مرحله کمترین هزینه موجود در گراف ۱۲ بوده که از ارتباط راس ۲ به ۳بوجود آمده است.



گام ۳ : کمترین هزینه بعد ۱۴ بوده که از پیمایش یال ۱ به ۶ بوجود آمده است.

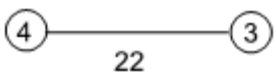


گام ۴ : کمترین هزینه بعد ۱۶ بوده که از پیمایش یال ۱ به ۲ بوجود آمده است.

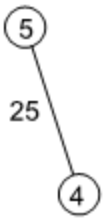


⟵ از یال ۶ به ۳ صرفه نظر می کنیم. (باعث ایجاد حلقه می شود.)

گام ۵ : کمترین هزینه بعد ۲۲ بوده که از پیمایش یال ۳ به ۴ بوجود آمده است.

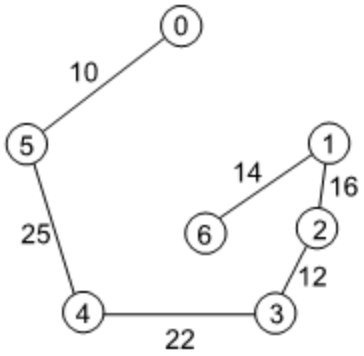


گام ۶ : کمترین هزینه بعد ۲۵ بوده که از پیمایش یال ۴ به ۵ بوجود آمده است.



⟵ از یال ۱ به 0 صرفه نظر می کنیم. (باعث ایجاد حلقه می شود.)

⟵ از یال ۶ به ۴ صرفه نظر می کنیم. (باعث ایجاد حلقه می شود.)



مرتبهاجرایی الگوریتم راشال : o (e log e)

۲- پریم

در این روش با شروع از یک گره دلخواه ، ابتدا آن گره و سپس نزدیکترین گره مجاور آن را انتخاب می کنیم.

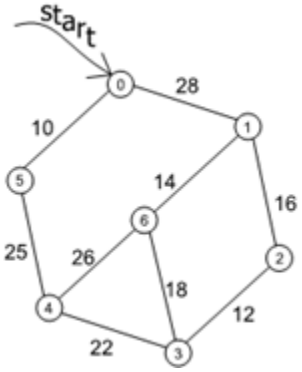
حال نزدیک ترین گره را به یکی از دو گره ملاقات شده انتخاب می کنیم.

به همین ترتیب ادامه می دهیم تا درخت پوشا به وجود آید.

⟵ از یالی که باعث ایجاد حلقه شود ،صرفه نظر می کنیم.

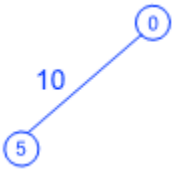
مرتبهاجرایی الگوریتم preem : o (n^2)

درخت پوشای شکل زیر را رسم کنید.

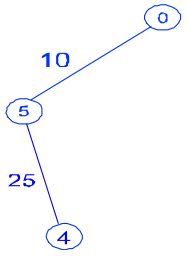


گام ۱ : راس شروع (0) را در نظر می گیریم.

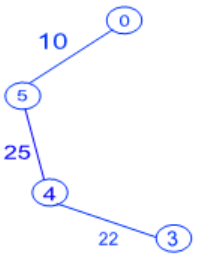
از راس صفر کوچکترین یال موجود 10 می باشد.



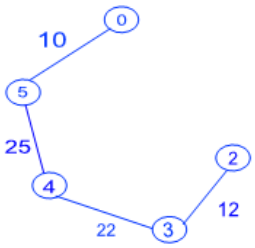
گام ۲ :حال راس های 0 و 5 را در نظر می گیریم. در اینجا کوچکترین یال 25 است.



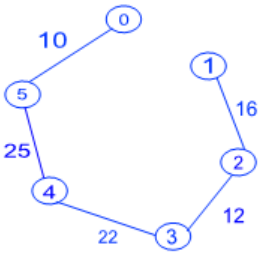
گام ۳ : حال از بین راس های 5 و 0 و 4 کوچکترین یال را در نظر می گیریم.(یال های ارتباطی 22 و 26 و 28) هستند.



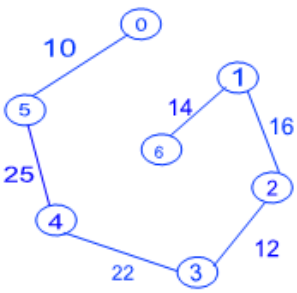
گام ۴ : حال از بین راس های 0 و 5 و 4 و 3 ، کوچکترین یال را انتخاب می کنیم.



گام ۵ : حال از بین راس های 0 و 5 و 4 و 3 و 2 کوچکترین یال را انتخاب می کنیم.



گام ۶ : حال از بین راس های 0 و 5 و 4 و 3 و 2 و 1 کوچکترین یال را انتخاب می کنیم.



فصل پنجم : مرتب سازی

در بحث مرتب سازی ابتدا لازم است الگوریتم های جستجو را برسی کنیم.

الگوریتم های جستجو خود شامل دو بخش می باشند

۱- خطی (آرایه نا مرتب)

۲- دودویی (آرایه مرتب)

۱- خطی

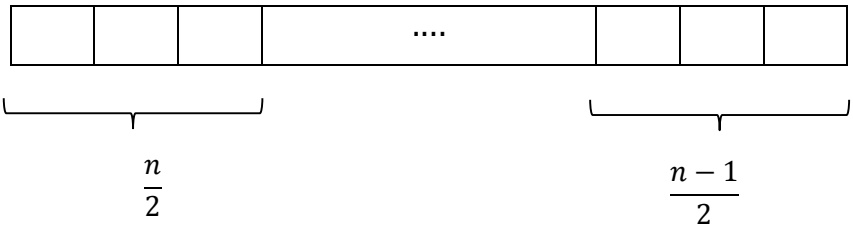
در این روش خانه های آرایه را یک به یک برسی می کنیم تا به مقدار مدنظر برسیم.

تنها مزیت این روش این است که الزامی بر مرتب بودن آرایه ندارد.

۲- دودویی

فرض کنید دارای یک آرایه ی n تایی دودویی هستیم.برای جستجو در این آرایه ابتدا لازم است آرایه مرتب شود.

برای این کار لازم است آرایه دا به دو قسمت مساوی تقسیم کنیم.



حال برسی می شود مقدار مد نظر در سمت راست آرایه قرار دارد یا در سمت چپ.

با فرض بر اینکه مقدار مد نظر در سمت چپ آرایه قرار دارد، مجددا سمت چپ را به دو قسمت تقسیم می کنیم.

حال طبق مرحله فوق برسی می شود تا زمانی که به خانه مد نظر برسیم.

اگر مقدار مدنظر یافت نشود یعنی چنین مقداری در آرایه ثبت نشده است.

⟵ مرتبه اجرایی این الگوریتم (دودویی) ($\log n$) o می باشد.

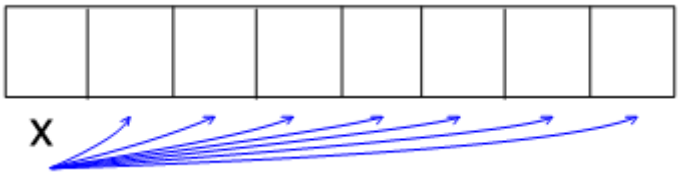
مرتب سازی

الگوریتم های مرتب سازی خود شامل چندین بخش می باشند؛

۱- مرتب سازی حبابی

در این نوع الگوریتم مقدار اول را در نظر گرفته ،و آن را با تک تک خانه های آرایه مقایسه می کنیم تا محل اصلی درج آن یافت شود.

نتیجـــــه ← در مرتب سازی حبابی هر مقداری که در نظر گرفته شود،با تمام مقادیر چک می شود.سپس مقدار بعد را انتخاب می کند.



از آنجایی که برای انجام این عمل دو حلقه ی for لازم است ، فلذا مرتبه اجرایی این الگوریتم o (n^2) می باشد.

۲- مرتب سازی درجی ' Insertion Sort '

یک الگوریتم مرتب‌سازی ساده بر مبنای مقایسه است. این الگوریتم برای تعداد داده‌های زیاد، کارآمد نیست.

در این الگوریتم هر مقداری که انتخاب شود در محل اصلی خود درج می شود.

مثال : لیست زیر را از طریق مرتب سازی درجی مرتب کنید.

20 , 12 , 10 , 23 , 40

گام ۱ – در مرحله اول ۲۰ را در نظر می گیرد.

20

گام ۲- در مرحله دوم ۲۰ را با عدد ۱۲ در نظر می گیرد.

20 12

گام ۳- در مرحله سوم عدد بعد را با لیست مرتب شده قبل در نظر می گیرد.

10 12 20

گام۴- در مرحله چهارم مقدار ۲۳ را درج می کند .(در این مرحله جا به جایی صورت نمی گیرد.)

10 12 20 23

گام ۵-در مرحله آخر همه اعداد را در نظر می گیریم.

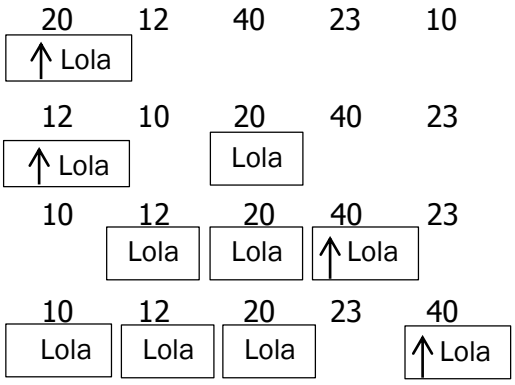
10 12 20 23 40

مرتب سازی درجی در بهترین حالت دارای مرتبه اجرایی (n) o و در بد ترین حالت(n^2) o خواهد بود.

۳- مرتب سازی سریع ' Quick Sort '

در این نوع مرتب سازی ابتدا لازم است یک لولا انتخاب کنیم.

⟵ همیشه مقدار اول به عنوان لولا در نظر گرفته می شود سپس ، هر چه از لولا کوچکتر باشد سمت چپ و هر چه از لولا بزرگتر باشد سمت راست قرار می گیرد.



هر مقداری که به عنوان لولا در نظر گرفته شود، دیگر تغییر مکان نخواهد داد.

مرتب سازی سریع دارای مرتبه اجرایی n^2 برای بدترین حالت و n log n برای بهترین حالت و حالت متوسط می باشد.

۴-مرتب سازی ادغامی ' merge Sort '

در این نوع مرتب سازی هر بار دو مقدار را در نظر می گیریم. (دودویی)

20 12 40 23 10 9 31

12 20 23 40 9 10 31

مرتب سازی ادغامی در همه حالات دارای مرتبه اجرایی($n \log n$) o می باشد.

9 10 12 20 23 31 40

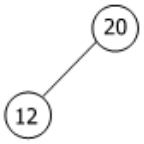
۵-مرتب سازی درختی ' Tree Sort '

در این نوع مرتب سازی هر بار هر خانه را به عنوان ریشه در نظر گرفته و مقادیر بعد اگر بزرگتر از مقدار ریشه باشد سمت راست و در غیر این صورت سمت چپ ریشه قرار می گیرند.

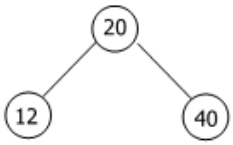
20 , 12 , 40 , 23 , 10 , 14

گام ۱ : ۲۰ را به عنوان ریشه در نظر می گیرد.

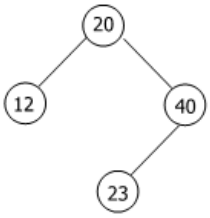
گام ۲ : از آنجایی که ۱۲ از ۲۰ کوچکتر است سمت چپ ریشه قرار می گیرد.



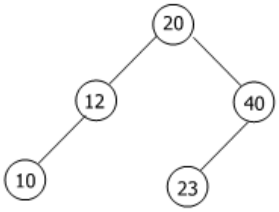
گام ۳: حال عدد ۴۰ را در نظر می گیریم،۴۰ از ریشه بزرگتر است فلذا سمت راست قرار می گیرد.



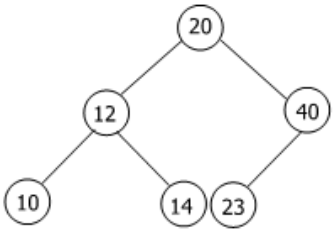
گام۴ :حال عدد ۲۳ را در نظر می گیریم،۲۳ از ریشه بزرگتر است (سمت راست قرار می گیرد) اما از آنجایی که از ۴۰ کوچکتر است سمت چپ ۴۰ قرار می گیرد.



گام ۵ : حال عدد ۱۰ را در نظر می گیریم،۱۰ از ریشه کوچکتر است (سمت چپ قرار می گیرد) از آنجایی که از ۱۲نیز کوچکتر است سمت چپ قرار می گیرد.



گام ۶ : در آخر عدد ۱۴ را در نظر گرفته و طبق مراحل فوق عمل می کنیم.



پیمایش میانوندی ' LVR ' درخت فوق به شکل زیر خواهد بود ؛

LVR : 10 , 12 , 14 , 20 , 23 , 40

۶-مرتب سازی مبنایی ' Radix Sort '

20 , 12 , 40 , 23 , 10 , 14

در این نوع مرتب سازی ابتدا لازم است مبنای اعداد را در نظر بگیریم. (بزرگ ترین رقم موجود در اعداد مبنا محسوب می شود.)

به تعداد مبنا سطر در نظر گرفته می شود.

گام۱ : اعداد را بر اساس یکان مرتب می کنیم.

[0]: 20 , 40 , 10

[1]:

[2]: 12

[3]: 23

[4]: 14

گام۲ : اعداد را بر اساس دهگان مرتب می کنیم.

[0]: 20 , 40 , 10	
[1]:	10 , 12 , 14
[2]: 12	20 , 23
[3]: 23	

[4]: 40

حال همه را بصورت پیمایش سطری چاپ می کند.

10 , 12 , 14 , 20 , 23 , 40