آرایه‌ها Array

آرایـه نوعی ساختمان داده است که عناصر آن هم نوع بوده و هر یک از عناصر با یک اندیس به صورت مسـتقیم قابـل دسـتیابی اسـت. آرایـه مـی‌تواند یک بعدی , دو بعدی و یا چند بعدی باشد. آرایه‌های دو بعدی را با نام ماتریس می‌شناسیم.

$[L_1 \dots U_1 , L_2 \dots U_2 , L_n \dots U_n]$

Array [L … U] of items

تعداد عناصر آرایه $= U - L + 1$

تعداد عناصر آرایه n بعدی $= [U_1 - L_1 + 1][U_2 - L_2 + 1][U_n - L_n + 1]$

فضای اشغال شده توسط آرایه (فضای مورد نیاز) $= (U - L + 1) \times n$

مـثال : در یـک آرایـه بـه نام [200] Float اگر آدرس شروع آرایه در حافظه 1000 باشد A25 در کدام آدرس قرار دارد.

$A[i] = (i - L) \times n + \alpha$

محل عنصر iام در حافظه $= (25 - 0) \times 4 + 1000 = 1100$

آرایه‌های دوبعدی یا ماتریس‌ها به دو روش در حافظه ذخیره می‌شوند.

$$\begin{bmatrix} 2 & 5 \\ 1 & 6 \\ 3 & 4 \end{bmatrix} \quad 3 \times 2$$

۱. روش سطری Row Major

سطری

0	1	2	3	4	5
2	5	1	6	3	4

۲. روش ستونی Column Major

ستونی

0	1	2	3	4	5
2	1	3	5	6	4

A : Array $[L_1 \dots U_1 , L_2 \dots U_2]$ of items

تعداد عناصری $= [U_1 - L_1 + 1][U_1 - L_2 + 1]$

آدرس $A[i , j]$ در روش سطری $= [(i - L_1) \times (U_2 - L_2 + 1) + (j - L_2)] \times n + \alpha$

آدرس $A[i , j]$ در روش ستونی $= [(j - L_2) \times (U_1 - L_1 + 1) + (i - L_1)] \times n + \alpha$

مثال : طبق آرایه زیر , آدرس‌های خواسته شده را محاسبه نمائید.

$$\overset{L_1 \ldots U_1 \;\; L_2 \ldots U_2}{A : [1 \ldots 3 \,,\, 1 \ldots 2]} \implies$$

در زبان C داریم $\implies A[3][2]$

$$\begin{bmatrix} 2 & 5 \\ 1 & 6 \\ 3 & 4 \end{bmatrix}$$

روش سطری $A\,[3\,,2] = (3-1)\times(2-1+1)+(2-1) = 2\times 2+1 = 5$

روش ستونی $A\,[3\,,2] = (2-1)\times(3-1+1)+(3-1) = 1\times 3+2 = 5$

روش سطری $A\,[1\,,2] = (1-1)\times(2-1+1)+(2-1) = 1$

روش ستونی $A\,[1\,,2] = (2-1)\times(3-1+1)+(1-1) = 3$

تمرین : در یک آرایه به شکل A [1 ... 100 , 1 ... 26] of integer اگر این آرایه از محل 1000 حافظـه شروع شده باشد محل داده [6 , 60] A در روش سطری و محل داده [4 , 20] A در روش ستونی کدام آدرس حافظه است.

$$A\,[60\,,6] = (60-1)\times(26-1+1)+(6-1)\times 2+1000 = 4078$$

$$A\,[20\,,4] = (4-1)\times(100-1+1)+(20-1)\times 2+1000 = 1638$$

در آرایــه‌هــای دو بعــدی مربعی یا ماتریس‌های مربعی که کلیه عناصر بالای قطر اصلی آن صفر باشند یــک ماتریس پایین مثلثی تشکیل می‌گردد و برعکس اگر کلیه عناصر پایین قطر اصلی آن صفر باشند یــک ماتــریس بــالا مثلثــی تشــکیل خــواهد شــد. در یــک ماتــریس پایین مثلثی یا بالا مثلثی حداکثر $\frac{n(n+1)}{2}$ عنصر غیر صفر داریم که n اندازه هر بعد ماتریس است.

$$\begin{bmatrix} 1 & 6 & 7 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{bmatrix}$$ بالا مثلثی

حداکثر عناصر غیر صفر $= \frac{3(3+1)}{2} = 6$

ماتریس بالا مثلثی $A\,[i\,,j] = 0 \qquad i > j \Longrightarrow$

ماتریس پایین مثلثی $A\,[i\,,j] = 0 \qquad i < j \Longrightarrow$

اگـر انـدازه ابعـاد ماتریس‌های مثلثی افزایش یابند این ماتریس‌ها حاوی تعداد زیادی صفر خواهند بود کـه ذخیـره کردن سطری یا ستونی ماتریس به طور کامل در حافظه باعث هدر رفتن بخشی از فضای حافظـه مـی‌گردد. به همین دلیل ماتریس‌های مثلثی را بصورت سطری یا ستونی بدون در نظر گرفتن صفرها در حافظه ذخیره می‌کنند.

پایین مثلثی =====> سطری $\frac{(i-1)\times i}{2}+j$

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 5 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$ =====> پایین مثلثی

1	2	3	4	5	6
1	2	5	3	1	1

بالا مثلثی =====> ستونی $\frac{(j-1)\times j}{2}+i$

$$\begin{bmatrix} 1 & 6 & 7 \\ 0 & 2 & 5 \\ 0 & 0 & 4 \end{bmatrix}$$ =====> بالا مثلثی

1	2	3	4	5	6
1	6	2	7	5	4

جمع ماتریس‌ها

در جمـع دو ماتریس , حتماً باید یک ماتریس $m \times n$ با یک ماتریس $m \times n$ جمع شده و نتیجه نیز یک ماتریس $m \times n$ خواهد شد. در این عملیات عناصر دو آرایه نظیر به نظیر با یکدیگر جمع خواهند شد.

$$A_{m\times n} + B_{m\times n} = C_{m\times n}$$

for (i = 0 , i < m , + + i)

for (j = 0 , j < n , + + j)

$C_{ij} = a_{ij} + b_{ij}$

ضرب ماتریس‌ها

در عمـل ضـرب , یـک ماتـریس A_{mL} و یک ماتریس B_{Ln} با یکدیگر ضرب شده و ماتریس بدست آمده نیز دارای سطر و ستونهایی می‌باشد که سطر ماتریس بدست آمده با تعداد سطرهای ماتریس اول و ستون ماتریس بدست آمده با تعداد ستونهای ماتریس دوم برابر است.

$$C_{mn} = A_{\underset{i}{m}\underset{k}{L}} \times B_{\underset{k}{L}\underset{i}{n}}$$

$$\begin{bmatrix} 3 & 4 & 1 & 2 \\ 2 & 5 & 3 & 1 \\ 1 & 0 & 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & 1 \\ 3 & 5 \end{bmatrix} = \begin{bmatrix} - & - \\ - & - \\ - & - \end{bmatrix}$$

$$3 \times 4 \quad \times \quad 4 \times 2 \quad = \quad 3 \times 2$$

for (i = 0 , i < m , + + i)

for (j = 0 , j < n , + + j)

{

$C_{ij} = 0$

for (k = 0 , k < L , + + k)

$C_{ij} = a_{ik} \times b_{kj} + C_{ij}$

}

تمرین : مقدار [1 ,0] C را در حاصلضرب دو ماتریس مثال قبل بدست آورید.

جــواب : برای بدست آوردن مقدار خواسته شده باید حلقه‌های for بالا را Trace کنیم. پس بنابراین داریم :

$$C_{ij} = 0$$
$$C_{ij} = a_{ik} \times b_{kj} + C_{ij}$$
$$C_{ij} = 2 \times 1 + 0 = 2$$
$$C_{ij} = 5 \times 0 + 2 = 2$$
$$C_{ij} = 3 \times 1 + 2 = 5$$
$$C_{ij} = 1 \times 3 + 5 = 8$$

i	j	k	L	C_{ij}
1	0	0	4	0
		1		2
		2		2
		3		5
				8

ترانهاده

برای اینکه ترانهاده یک ماتریس را بدست آوریم جای سطرها و ستونهای ماتریس عوض می‌شوند.

$$A\begin{bmatrix} 2 & 3 & 5 \\ 1 & 0 & 7 \end{bmatrix} \rightarrow A^T\begin{bmatrix} 2 & 1 \\ 3 & 0 \\ 5 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & 4 \\ 1 & 2 & 5 \\ 1 & 3 & 8 \end{bmatrix} \rightarrow \begin{bmatrix} 2 & 1 & 1 \\ 0 & 2 & 3 \\ 4 & 5 & 8 \end{bmatrix}$$

$A_{m \times n}$

```
for (i = 0 , i < m , + + i)
for (j = 0 , j < n , + + j)
A [j] [i] = A [i] [j]
```

جستجوی خطی در آرایه

```
Array A[n] , x
Int search (A[n] , x) ;
{
    int i = 1 ;
    while (i <= n && A[i] != x)
        i + + ;
    if (i > n ) return – 1 // داده پیدا نشده
    else return i // داده در محل اندیس آرایه است
}
```

1	2	3	4	5
1	5	2	8	6

n	x	i	A[i]
5	8	1	1
		2	5
		3	2
		4	8

x = A[4]

جستجوی دودویی برای آرایه‌های مرتب

```
Int bsearch (A[n] , int x , int L , int U)
{
    int i ;
    while
    {
        i = ⌊(L + U) / 2⌋;
        if ( x < A[i] ) U = i – 1
        else if ( x > A[i] ) L = i + 1
            else return i // داده در اندیس i است
    }
    return – 1 // داده پیدا نشده است
}
```

x	i	L	U	A[i]
8	5	1	10	12
	2	3	4	2
	4	4		5
	3			8

جمع دو چندجمله‌ای بوسیله آرایه

$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_2 x^2 + a_1 x + a_0$

a_n	a_{n-1}	a_{n-2}	….	a_1	a

$P(x) = 3x^2 + 5x + 2$

$P(x) = 3x^3 + 3$

0	3	5	2	+
3	0	0	3	
				=
3	3	5	5	

$P(x)\ x^{100} + 2$

ضریب	1	2
توان	100	0

Stack یا پشته

Stack لیستی است که اعمال ورودی و خروجی یا اضافه و حذف در آن از یک طرف لیست انجام می‌شود. به این جهت به آن لیست Last In First Owt (LIFO) می‌گویند. بدین معنی که آخرین ورودی به پشته , اولین خروجی خواهد بود. عنصر بالایی پشته را top پشته می‌گویند. با افزودن داده روی پشته , متغیر top یکی زیاد شده و داده در محل top از پشته قرار می‌گیرد. برای خارج کردن یک عنصر از پشته نیز داده‌ای که در محل top قرار گرفته از Stack خارج می‌گردد و متغیر top یکی کم می‌شود. مقدار اولیه top صفر است و با افزودن داده به یک پشته n عضوی , top می‌تواند تا مقدار n تغییر کند.

Top = 0 ===➔ پشته خالی است

Top = n ===➔ پشته پر است

دو عمل اصلی برای پشته‌ها را با push کردن و pop کردن می‌شناسیم. Push (x) داده x را در بالای پشته قرار می‌دهد و عمل pop عنصر بالای پشته را در متغیر x ذخیره می‌کند.

$x = pop \equiv pop\ (x)$

Stack : Array [1 .. n] of items

```
int pop ( )
{
    int x ;
    if (top = 0)
    {
        C out << " پشته خالی است " ;
        return – 1 ;
    }
    else
    {
        x = Stack [top] ;
        top = top – 1 ;
    }
    return x ;
}
```

```
void push (int x)
{
    if (top = = n)
    {
        C out << " پشته پر است " ;
        return – 1 ;
    }
    else
    {
        top + + ;
        Stack [top] ;
    }
}
```

مثال : مقدار نهایی A و B و C چقدر است؟

n = 5 A = 10 B = 2 C = 5

push (B)

push (A + B)

pop (C)

push (A – B)

push (C)

push (B)

pop (A)

pop (B)

push (A × B)

push (C)

push (A)

pop (B)

pop (C)

pop (A)

2	
2	12
12	24
12	8
2	

A	B	C
~~10~~	~~2~~	~~5~~
~~2~~	~~12~~	~~12~~
24	2	12

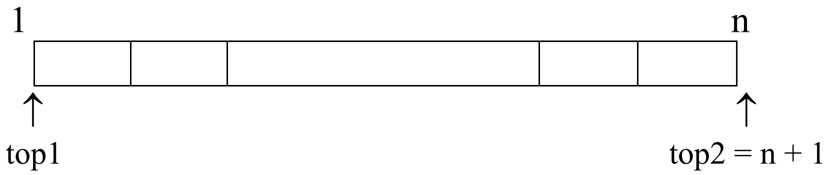
پشته‌های چندگانه

پشته دوگانه : برای پیداسازی دو پشته در یک آرایه نیاز به دو متغیر top1 برای نشان دادن بالاترین عنصر پشته اول و top2 برای بالاترین عنصر پشته دوم داریم. top1 و top2 در جهت عکس یکدیگر حرکت می‌کنند. مقدار اولیه $top1 = 0$ و مقدار اولیه $top2 = n + 1$ است.

$top1 = 0$ ===➔ پشته ۱ خالی است

$top2 = n + 1$ ===➔ پشته ۲ خالی است

$top2 = top1 + 1$ ===➔ آرایه پر است



دنباله‌های قابل قبول در پشته‌ها

هرگاه اعدادی را به صورت مرتب شده صعودی داشته باشیم و بخواهیم اعداد دیگری را از آن استخراج کنیم باید این قانون را راعایت کنیم که اعداد بزرگتر در صورتیکه اعداد کوچکتر در پشته قرار نگرفته‌اند حق قرار گرفتن در پشته را ندارند. مثلاً اعداد ۴ , ۳ , ۲ , ۱ را در نظر می‌گیریم. عدد ۲ در صورتی می‌تواند push شود که حتماً عدد ۱ push شده باشد و عدد ۳ زمانی می‌تواند push شود که اعدا ۱ و ۲ قبلاً push شده باشند.

مثال : چهار عدد ۴ , ۳ , ۲ , ۱ را داریم. کدامیک از اعداد زیر را می‌توانیم تولید کنیم ؟

۴ ۳ ۱ ۲	۴ ۲ ۳ ۱	۳ ۲ ۴ ۱	۳ ۱ ۴ ۲	۲ ۱ ۳ ۴
push 1 push 2 push 3 push 4 pop 4 pop 3	push 1 push 2 push 3 push 4 pop 4	push 1 push 2 push 3 pop 3 pop 2 push 4 pop 4 pop 1	push 1 push 2 push 3 pop 3	push 1 push 2 pop 2 pop 1 push 3 pop 3 push 4 pop 4
قابل تولید نیست	قابل تولید نیست		قابل تولید نیست	

اگر اعداد به صورت صعودی داده شوند (۴ ۳ ۲ ۱) و سه عدد a و b و c داشته باشیم بطوریکه $b < c < a$ در اینصورت دنباله abc قابل تولید نیست.

ارزشیابی عبارت

اولویت عملگر

بطور کلی اگر عبارت $a \times b + c / d$ را داشته باشیم اولویت عملگرها را به صورت زیر می‌نویسیم :

1. ( )
2. Not , – (قرینه) , توان
3. and , × , / , mod
4. OR , + , –
5. < , > , < = , > = , < > (! =)

نکته : بین عملگرهایی که اولویت مساوی دارند عملگری زودتر محاسبه می‌گردد که سمت چپ باشد.

روش نمایش عبارات محاسباتی

میانوندی	infix	a + b
پسوندی	postfix	ab +
پیشوندی	prefix	+ ab

تبدیل عبارات میانوندی به پسوندی و پیشوندی بدون استفاده از پشته

۱- پرانتز گذاری

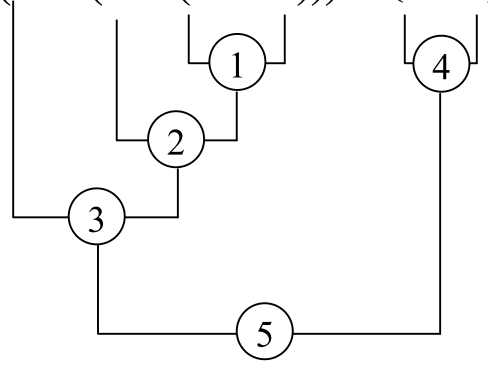
۲- برای تبدیل به پیشوندی , درون هر پرانتز عملگر را به سمت چپ منتقل می‌کنیم.

۳- برای تبدیل به پسوندی , درون هر پرانتز عمگلر را به سمت راست منتقل می‌کنیم.

۴- پرانتزها را حذف می‌کنیم.

مثال :

$$((a + (b \times (c \uparrow a))) - (b / c))$$



$$\text{postfix} = (a(b(ca)\uparrow)\times+(bc)/)- = abca\uparrow\times+bc/-$$

$$\text{prefix} = -+a\times b\uparrow ca/bc$$

استفاده از پشته در تبدیل عبارات infix به postfix

۱- عبارت infix را از چپ به راست پیمایش می‌کنیم.

۲- پرانتز باز را در پشته push می‌کنیم.

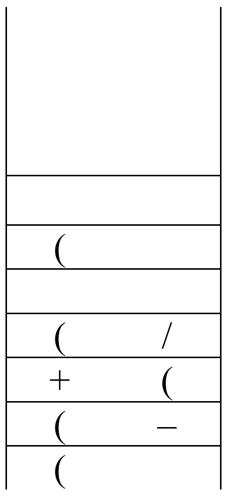
۳- عملوندها را در خروجی می‌نویسیم.

۴- در صـورتیکه به یک عملگر رسیدیم اگر top پشته دارای عملگری با اولویت بیشتر یا مساوی نـبود آنـرا push مـی‌کنـیم در غیـر اینصـورت عملگر top پشته را pop کرده و در خروجی می‌نویسیم.

۵- هرگاه به پرانتز بسته رسیدیم آنقدر pop می‌کنیم تا به اولین پرانتز باز برسیم.

مثال :

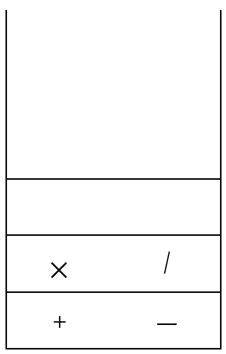
$$((a + (b \times (c \uparrow a))) - (b / c))$$



$abca\uparrow\times+bc/-$

مثال : با استفاده از پشته , عبارت زیر را به صورت postfix بنویسید.

$a+b\times c\uparrow a-b/c$



$abca\uparrow\times+bc/-$

مثال : با استفاده از پشته , عبارت زیر را به صورت postfix بنویسید.

$a+(b\times c)\uparrow a-b/c$

(	↑	/
+	−	

$abc\times a\uparrow +bc/-$

بـرای تـبدیل عـبارات infix بـه عبارات prefix از دو پشته استفاده می‌کنیم. یکی پشته عملوندها و دیگری پشته عملگرها. push کردن و pop کردن در پشته عملگرها مانند تبدیل infix به postfix است. با رسیدن به هر عملوند آنرا در پشته عملوندها push می‌کنیم. در صورت pop شدن هر عملگر از پشته عملگرها , دو عملوند بالای پشته عملوندها pop شده و با عملگر مربوطه به شکل prefix در پشته عملوندها push می‌شود. بقیه قوانین مانند قوانین infix به postfix است.

مثال : عبارت زیر را بوسیله پشتبه از infix به prefix بنویسید.

$a+(b\times c)\uparrow d/a-c\times b$

×			
(	↑	/	×
+	-		

c	d	a	d
b	$\times bc$	$\uparrow\times bcd$	$/\uparrow\times bcda$
a	$+a/\uparrow\times bcda$		

$-+a/\uparrow\times bcda\times cd$

مثال : عبارت infix زیر را بوسیله دو پشته به prefix تبدیل کنید.

$a+b\times c\uparrow(2-b)\times c/(d+a)$

-		
(	+	
↑	(	
×	×	/
+		

b					
2	-2b	a			
c	$\uparrow$c-2b	c	d	+da	
b	$\times b\uparrow$c-2b	$\times\times b\uparrow$c-2b	$/\times\times b\uparrow$c-2bc+da		
a	$+a/\times\times b\uparrow$c-2bc+da				

تبدیل عبارت postfix به infix

بـا اسـتفاده از یـک Stack مـی‌توان رشته postfix ورودی را به infix تبدیل کرد. برای این منظور

رشــته postfix را از چــپ پــردازش می‌کنیم. هر عملوند درون پشته push می‌شود. با رسیدن به هر عملگـر , دو عنصـر پشــته pop شده و بصورت infix نوشته می‌شود. سپس عبارت infix تولید شده درون پشــته push مــی‌شــود. در پایــان پردازش رشته ورودی , پشته حاوی یک عنصر است که شکل infix مــورد نظر می‌باشد. خروجی infix باید لزوماً پرانتزگذاری شده باشد. عملوند top پشته سمت راست عملگر نوشته می‌شود.

مثال :

abca↑×+bc/–

a			
c	c↑a	c	
b	b×(c↑a)	b	b/c
a	a+(b×(c↑a))	(a+b×(c↑a))–(b/c))	

<u>تبدیل عبارات prefix به infix</u>

بــرای تــبدیل عبارت prefix به infix باید رشته ورودی را از سمت راست پردازش کنیم. مانند روش قبل عملوندها در پشته push می‌شوند و با رسیدن به هر عملگر , دو عملوند بالای پشته pop شده و بـا عملگـر ورودی بصـورت infix نوشته می‌شود و نتیجه در پشته push می‌شود. عملوند top پشته سمت چپ عملگر قرار می‌گیرد.

–+a×b↑ca/bc

c	b	a		
b	a	(c↑a)	(b×(c↑a))	(a+(b×(c↑a))
c	(b/c)	(a+b×(c↑a) –(b/c)		

<u>تبدیل عبارات postfix به prefix و بالعکس</u>

بــرای تبدیل عبارات postfix و prefix به همدیگر می‌توان آنها را ابتدا تبدیل به حالت میانی infix کــرده و ســپس عبارت infix را با روشهای گفته شده به حالت مطلوب تبدیل نمود. همچنین می‌توان بصــورت مســقتیم عــبارت postfix و prefix را با استفاده از الگوریتم قبلی به یکدیگر تبدیل کرد. با این تفاوت که هنگامیکه در حین پردازش رشته ورودی به یک عملگر رسیدیم , دو عملوند بالای پشته pop شــده و بــه جــای اینکه به infix با عملگر ورودی در پشته push شوند به هر کدام از حالتهای مورد نظر postfix یا prefix در پشته push می‌شوند.

صف (queue)

صف لیستی است که عمل افزودن داده‌ها درون آن از یک طرف لیست یا انتهای لیست و عمل حذف داده‌ها از سمت دیگر یا ابتدای لیست انجام می‌شود. صف را لیست FIFO (First In First Out) می‌نامند. زیرا اولین عنصر ورودی , اولین عنصر خروجی از صف نیز هست. در ساختمان داده صف دو متغیر front و rear به ترتیب برای نشان دادن جلو و انتهای صف بکار می‌روند. صف را می‌توان با استفاده از آرایه‌ها یا لیست‌های پیوندی پیاده سازی کرد.

اگر صف را آرایه‌ای n عضوی از عناصر بدانیم مقادیر front و rear می‌تواند از صفر تا n تغییر کند که برای صف در ابتدا مقادیر اولیه صفر را برای front و rear تعریف می‌کنیم. front = rear = 0

در صورتیکه متغیر front با rear برابر باشد صف خالی است و در صورتیکه rear برابر با n باشد صف پر است.

rear = n ===➔ صف پر است

front = rear ===➔ صف خالی است

دو عمل اصلی برای صف , حذف کردن داده‌ها از صف و افزودن داده‌ها به صف است که به ترتیب با delqueue و Addqueue نمایش می‌دهیم. تابع Addqueue(x) به این معنی است که عنصر x به انتهای صف اضافه شده است و delqueue نیز مقدار جلوی صف را برداشته و در متغیر x قرار می‌دهد. x = delqueue

پیاده سازی تابع Addqueue و delqueue از صف

queue : Array [1 .. n] of item

برای اضافه کردن	برای حذف کردن

void Addqueue (int x)
{
    if (rear = = n)
        C out << " صف پر است " ;
    else
    {
        rear + + ;
        queue[rear] = x ;
    }
}
	int delqueue ( )
	{
	    if (front = = rear)
	    {
	        C out << " صف خالی است " ;
	        return 0 ;
	    }
	    else
	    {
	        front + + ;
	        x = queue[front] ;
	        return x ;
	    }
	}

مثال : با استفاده از توابع صفحه قبل مقادیر نهایی A و B و C را بدست آورید.

A = 5 B = 10 C = 2 n = 4

Addqueue (A + B)
Addqueue (C)
Addqueue (B × C)
A = delqueue ( )
B = delqueue ( )
Addqueue (B – A)

15	2	20	–13
1	2	3	4

Rear	Front	A	B	C
0	0	5	10	**2**
1	1	**15**	**2**	
2	2			
3				
4				

پس بنابراین داریم : **A = 15** **B = 2** **C = 2**

توابع Addqueue و delqueue به صورتیکه نوشته شد یک صف خطی را پیاده‌سازی می‌کنند. مشکل صف خطی این است که تنها یک بار قابل پر شدن است و در صورتیکه عناصر آن حذف شوند نیز با پیغام « صف پر است » مواجه می‌شوید به همین دلیل صف را بصورت حلقوی تعریف می‌کنیم. در صف حلقوی (دوار) rear و front بعد از رسیدن به آخرین مقدار خود در صورت وجود شرایط لازم مجدداً مقادیر اولیه را می‌توانند بگیرند. صف حلقوی n عضوی را بصورت آرایه صفر تا n – 1 تعریف می‌کنیم.

queue : Array [0 .. n – 1] of item

در این حالت وقتی rear = n – 1 عنصر بعدی در queue[0] قرار می‌گیرد.
در صف حلقوی front = rear به معنای خالی بودن صف است ولی شرط پر بودن صف بدین ترتیب تغییر می‌یابد.

front = (rear + 1) mod n ===➔ شرط پر بودن
front = rear ===➔ صف خالی است

برای اضافه کردن به صف حلقوی , rear یکی اضافه می‌شود و در صورتیکه rear = n – 1 باید صفر بشود. بدین منظور rear را با رابطه زیر در هر شرایطی مقداردهی می‌کنند.

rear = (rear + 1) mod n

این مسئله برای front نیز برقرار است.

front = (front + 1) mod n

برای حذف کردن

```
int delqueue ( )
{
	if (front = = rear)
	{
		C out << " صف خالی است " ;
		return 0 ;
	}
	else
	{
		front = (front + 1) mod n
		x = queue[front] ;
	}
}
```

برای اضافه کردن

```
void Addqueue (int x)
{
	rear = (rear + 1) mod n
	if (front = = rear)
		C out << " صف پر است " ;
	else
		queue[rear] = x ;
}
```

مثال :

			0	1	2	3
Addqueue [50]	r = 1	f = 0		50		
Addqueue [20]	r = 2	f = 0		50	20	
Addqueue [30]	r = 3	f = 0		50	20	30
delqueue ( )	r = 3	f = 1		50	20	30
Addqueue [10]	r = 0	f = 1	10	~~50~~	20	30

مثال : عبارت زیر را بصورت prefix و postfix بنویسید.

$$\sqrt{a^2 - bc} \Rightarrow (a \uparrow 2 - b \times c) \uparrow (1/2)$$

$$\text{postfix} = a2 \uparrow bc \times - 12 / \uparrow$$

$$\text{prefix} = \uparrow - \uparrow a2 \times bc / 12$$

مرتب‌سازی

در مرتب‌سازی تعدادی عنصر که از ورودی داده شده‌اند را بر اساس کلیدشان بصورت صعودی یا نزولی مرتب می‌کنیم.

مرتب‌سازی انتخابی (Selection Sort)

در مرتب‌سازی انتخابی یک آرایه n عنصری (A[1..n]) , n − 1 بار پیمایش می‌شود. در هر پیمایش بزرگترین عنصر در محل درست خود یعنی انتهای آرایه قرار می‌گیرد. با این روش آرایه از انتها مرتب می‌شود. در مرتب‌سازی انتخابی می‌توان با انتخاب کوچکترین عنصر در هر پیمایش و قرار دادن آن در محل درست خود یعنی ابتدای آرایه در هر پیمایش , مرتب‌سازی را از ابتدای لیست انجام داد.

مثال :

10	5	8	20	25	12	
10	5	8	20	12	25	پویش اول
10	5	8	12	20	25	پویش دوم
10	5	8	12	20	25	پویش سوم
8	5	10	12	20	25	پویش چهارم
5	8	10	12	20	25	پویش پنجم

برنامه کلی مرتب‌سازی انتخابی به شرح ذیل می‌باشد :

```
for (i = n ; i > 1 ; – – 1)
{
        max = A[1] ;
        index = 1 ;
        for (i = 2 ; j < = i ; + + j)
        if (A[j] > max)
        {
                max = A[j] ;
                index = j ;
        }
        A[index] = A[i] ;
        A[i] = max ;
}
```

مثال :

3	2	5	1
1	2	3	4

i	j	n	max	index
4	2	4	3	1
3	4		5	3
	4		3	1
	2			
	3			

نکته : در مرتب‌سازی انتخابی , حداکثر و حداقل n^2 مقایسه داریم. حداقل جابجایی صفر و حداکثر جابجایی نیز n بار خواهد بود.

مرتب‌سازی حبابی (Bubble Sort)

در مرتب‌سازی حبابی یک آرایه n عنصری (A[1..n]) , n − 1 بار پیمایش می‌شود و در هر پیمایش دو عنصـر متوالـی با یکدیگر مقایسه شده که در صورت لزوم جابجا خواهند شد. در هر پیمایش , طول آرایه پیمایش شده نسبت به مرحله قبل یکی کم می‌شود.

10	30	50	40	90	20	
10	30	~~50~~	~~40~~	~~90~~	~~20~~	مرحله اول
		40	50	20	90	
10	30	40	~~50~~	~~20~~	90	مرحله دوم
			20	50		
10	30	~~40~~	~~20~~	50	90	مرحله سوم
		20	40			
10	~~30~~	~~20~~	40	50	90	مرحله چهارم
	20	30				
10	20	30	40	50	90	مرحله پنجم

مرتب‌سازی از انتهای لیست

```
for (i = 1 ; i < n ; + + i)
        for (j = 1 ; j < = n ; + + j)
        if (A[j]) > A[j + 1])
        swap (A[j] , A[j + 1]) ;
```

مرتب‌سازی از ابتدای لیست

```
for (i = 1 ; i < n ; + + i)
        for (j = n ; j > = i ; – – j)
        if (A[j]) < A[j – 1])
        swap (A[j] , A[j – 1]) ;
```

مثال :

i	j	n
1	4	4
2	3	
3	2	
	1	
	4	

5	3	7	2	
1	2	3	4	
2	5	3	7	پویش اول
2	3	5	7	پویش دوم

« الگوریتم متعادل است »

در هر پویش امکان n^2 جابجایی وجود دارد. حداقل تعداد جابجایی نیز صفر است.

مرتب‌سازی حبابی بهینه شده

```
for (i = 1 ; i < = x ; + + i)
{
        sw = 0 ;
        for (j = 1 ; j < n – 1 ; + + j)
        if (A[j] > A[j + 1])
        {
                sw = 1 ;
                swap (A[j] , A[j + 1]) ;
        }
        if (sw = = 0) break ;
}
```

مرتب‌سازی دَرجی (Insertion Sort)

در مرتب‌سازی درجی فرض شده است که $i - 1$ عنصر اول لیست مرتب هستند. پس عنصر iام در جای صحیح خود قرار دارد.

```
For (i = 2 ; i < = n ; + + i)
{
        y = A[i] ;
        j = i – 1 ;
        while (j > 0 & & (y < A[j]))
        {
                A[j + 1] = A[j] ;
                j = j – 1 ;
        }
        A[j + 1] = y ;
}
```

i	n	y	j
2	6	60	1
3		40	2
4		20	1
5		10	0
			3
			2
			1
			0
			4
			3
			2
			1
			0

50	60	40	20	10	30
1	2	3	4	5	6

50	60	40			
50	40	60			
40	50	60			
10	40	50	60		

در این جابجایی حداقل n تا پویش داریم و در بهترین حالت نیز جابجایی نداریم.

مثال : آرایه زیر را به روش درجی مرتب کنید.

	1	2	3	4	5
A	4	8	5	2	6
	4	8			
	4	5	8		
	2	4	5	8	
	2	4	5	6	8

n	i	j	y
5	2	1	8
	3	2	5
	4	1	2
		3	6
		2	
		1	
		0	
		4	
		3	

مرتب سازی ادغامی (merge sort)

مـرتب سـازی ادغامـی مبتنـی بـر تقسـیم و حـل است و در روش ادغامی لیست n عنصری تبدیل به لیســت‌هــای یــک عنصــری شده (با تقسیمات متوالی بر ۲) و سپس لیست‌های یک عنصری که مرتب هســتند ادغــام شــده و لیســت‌های دو تایی مرتب تشکیل می‌دهند و سپس لیست‌های دو تایی مرتب شــده با هم ادغام می‌شوند و این فرآیند تا تولید لیست اولیه به صورت مرتب ادامه پیدا می‌کند که این حالت نیز بصورت بازگشتی است.

11	2	20	18	1	8	7	12	17	5
11	2	20	18	1	8	7	12	17	5
11	2	20	18	1	8	7	12	17	5
11	2	20	18	1	8	7	12	17	5
11	2		1	18	8	7		5	17
2	11				7	8			
2	11	20			7	8	12		
1	2	11	18	20	5	7	8	12	17
1	2	5	7	8	11	12	17	18	20

```
Void mergsort (int L , int U)
{
        int i ;
        if ( L < U )
        {
                i = ( L + U ) / 2 ;
                mergsort ( L , i ) ;
                mergsort ( i + 1 , U ) ;
                merg ( L , i , U ) ;
        }
}
```

مثال :

L = 3	U = 3	L = 4	U = 4
L = 3	U = 3	i = 3	
L = 1	U = 1	L = 2	U = 2
L = 1	U = 2	i = 1	
L = 1	U = 4	i = 2	

5	1	7	2

5	1		7	2

5	1	7	2

1	5		2	7

1	2	5	7

مثال : آرایه زیر را بوسیله merge sort مرتب کنید.

L = 4	U = 4	
L = 3	U = 4	
L = 3	U = 4	i = 3
L = 2	U = 2	
L = 1	U = 1	
L = 1	U = 2	i = 1
L = 1	U = 4	i = 2

	1	2	3	4
A	5	3	1	4

Merge sort (1 , 4)

1	2	3	4
3	5	1	4

1	3	4	5

تمرین : برنامه‌ای بنویسید که دو آرایه مرتب را بگیرد و در هم ادغام کند.

مرتب سازی سریع (quick sort)

در روش مـرتب‌سازی سریع , یک عنصر بعنوان عنصر محوری در نظر گرفته می‌شود که عنصر محوری را معمـولاً اولـین عنصـر آرایه در نظر می‌گیرند. بعد از اولین پیمایش , عنصر محوری در محل مناسب خـود در لیسـت قـرار مـی‌گیـرد و لیسـت به دو بخش مجزا تقسیم می‌گردد. عناصر سمت چپ عنصر محـوری که کوچکتر از عنصر محوری هستند و عناصر سمت راست عنصر محوری که همگی بزرگتر از عنصـر محـوری مـی‌باشند. این عمل مجدداً بر روی هر یک از دو بخش انجام می‌شود تا به لیست‌های یـک عنصری مرتب برسیم. متوسط زمان اجرای این الگوریتم $0(n \text{ Log } n)$ و بدترین زمان اجرای آن $0\ (n^2)$ می‌باشد که زمانی اتفاق می‌افتد که آرایه از پیش مرتب باشد.

1	2	3	4	5	6	7	8	9
12	10	2	8	15	7	3	1	14
محوری (Pivot)						j		I

3	10	2	8	1	7	12	15	14
	i	j	i	j				

2	1	3	8	10	7	12	14	15

```
Void quicksort (int L , int U )
{
        int i , j , pivot ;
        if ( L < U )
        {
                i = L + 1 ; j = U ; pivot = A[L] ;
                while ( i < j )
                {
                        while ( A [i] < pivot ) i + + ;
                        while ( A[j] > pivot ) j - - ;
                        if ( i < j ) swap ( A[i] , A[j]) ;
                }
                swap ( A[L] , A[j] ) ;
                quicksort ( L , j – 1 ) ;
                quicksort ( j + 1 , U ) ;
        }
}
```

لیست پیوندی (Link List)

لیست‌ها ساختمان داده‌ای هستند که اندازه آنها بصورت پویا تغییر می‌کند. پیمایش در لیست‌های پیوندی بصورت ترتیبی (خطی) است. بنابراین برای حذف , اضافه یا جستجو باید لیست را از ابتدا بصورت خطی پیمایش کرد. هر گره (node) در لیست پیوندی ساختاری با دو فیلد اصلی دارد. یکی فیلد داده که می‌تواند از هر نوع داده‌ای باشد و دیگری فیلد آدرس که به محل عنصر بعدی در لیست پیوندی اشاره می‌کند. در ساختمان داده لیست پیوندی اعمال اصلی حذف داده از لیست , اضافه کردن داده به لیست و جستجو در لیست انجام می‌شود. عنصر اول لیست پیوندی را هد (Head) یا هدر (Header) لیست می‌گویند و معمولاً این عنصر را برای سادگی پیمایش خالی نگه می‌دارند.

برای افزودن داده جدید به لیست پیوندی ۴ عمل اصلی انجام می‌گیرد.

۱- تشکیل گره (node) جدید بر اساس اطلاعات جدید افزوده شدنی

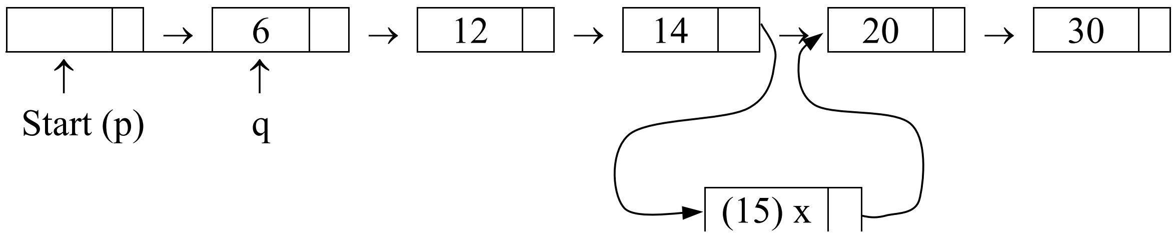
۲- بدست آوردن آدرس گره‌ای که باید قبل از گره جدید قرار گیرد (مثلاً گره p)

۳- آدرس گره جدید که به محل اشاره‌گر p اشاره می‌کند.

۴- آدرس گره p را به محل new node تغییر می‌دهیم.

```
Void insert ( int x , node * start )
{
        node * p , * q , * new node ;
        q = start → next ;
        p = start ; new node = new ( node ) ; new node → data = x ;
        while ( q → data < newnode → data)
                           x
        {
                p = q ;
                q = q → next ;
        }
        new node → next = p → next ;
        p → next = new node ;
}
```

مثال : می‌خواهیم گره ۱۵ را به لیست اضافه کنیم :



p	q
Start	6
6	12
12	14
14	20

<u>مراحل حذف یک گره از لیست پیوندی</u>

۱- یافتن گره قبل از گره مشخص شده برای حذف (q)

۲- تغییر دادن $q \rightarrow next$ به $p \rightarrow next$ (p حذف می‌شود)

۳- آزاد کردن حافظه‌ای که برای p در نظر گرفته‌ایم.

```
void dellinklist ( int x , node * store )
{
        node * p , * q ;
        p = start ;
        q = p ;
        while ( p → data ! = x )
        {
                q = p ;
                p = p → next ;
        }
        q → next = p → next ;
        delete (p) ;
}
```

مثال : گره شماره 3 را حذف کنید.

5 | → 8 | → 3 | → 2 | \

5 | → 8 | → 2 | \

p	q
5	5
8	8
3	2

تمرین : برنامه‌های زیر چه کاری انجام می‌دهند؟

```
node * f ( int x , node * start )
{
        node * p ;
        p = start ;
        while ( p ) ;
        if ( p → data ! = x && p ) p = p → next ;
        else return p ;
}
void g ( node * start )
{
        if (start ! = Null )
        {
                C out << start → data ;
                g ( start → next ) ;
        }
}
```

جـواب : در قسمت اول یعنی f عمل جستجو را انجام می‌دهد و در قسمت دوم یعنی g عناصر لیست را به ترتیب چاپ می‌کند.

<u>لیست پیوندی چرخشی</u>

اگر اشاره‌گر عنصر انتهای لیست به جای Null به هد لیست اشاره‌گر کند (start) لیست ما تبدیل به لیست تک پیوندی چرخشی می‌شود.

تمرین : عمل حذف و اضافه را در یک لیست پیوندی چرخشی بنویسید.

لیست دو پیوندی

در لیست دو پیوندی سه بخش وجود دارد.

۱- بخش data

۲- بخش سمت راست که به گره بعدی اشاره می‌کند.

۳- بخش سمت چپ که به گره قبلی اشاره می‌کند.

```
struct Linklist
{
        struct Linklist * left ;
        data ;
        struct Linklist * right ;
}
typedeg struct Linklist node
node * p , * q ;
```

مراحل اضافه کردن داده به لیست دو پیوندی

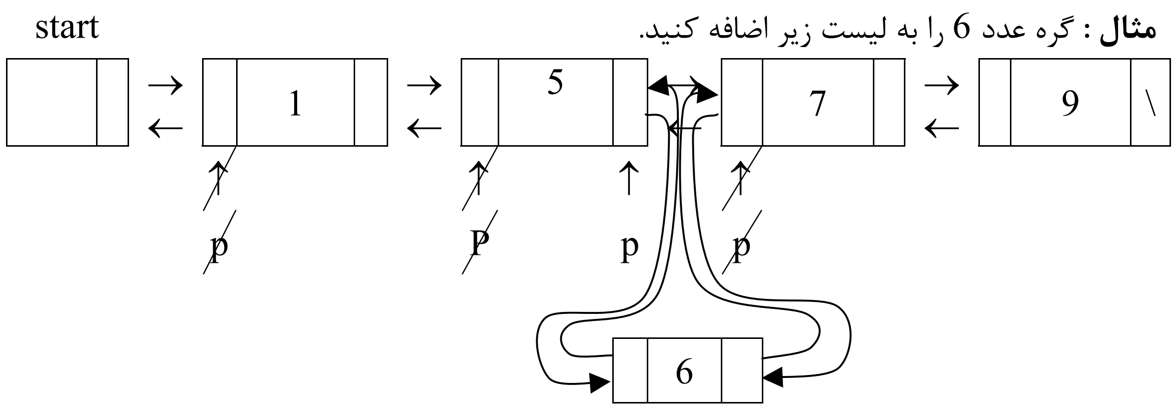
۱- تشکیل گره جدید با استفاده از اطلاعات اضافه شدنی (new node)

۲- پیدا کردن محل درج گره جدید

۳- انتساب مقادیر مورد نظر به بخشهای آدرس چپ و آدرس راست گره‌های p و new node

```
void insert (int x , node * start )
{
          node * newnode , * p ;
          newnode = new (node) ;
مرحله 1   newnode → data = x ;
          newnode → right = Null ;
          newnode → left = Null ;
          p = start → right ;
مرحله 2   while ( p → data < x ) p = p → right ;
          p = p → left ;
          ( p → right ) → left = newnode ;
مرحله 3   newnode → left = p ;
          newnode → right = p → right ;
          p → right = newnode ;
}
```

مثال : گره عدد 6 را به لیست زیر اضافه کنید.



حذف از لیست دو پیوندی

```
void delete ( int x , node * start )
{
        node *p ;
        p = start → right ;
        while ( p && p → data < x ) p = p → right ;
        if ( p = = Null | | p → data > x ) return “ داده در لیست نبوده است “ ;
        ( p → Left ) → right = p → right ;
        ( p → right ) → Left = p → Left ;
        delete ( p ) ;
}
```

نمایش چندجمله‌ایها با استفاده از لیست‌های پیوندی

همانطـور کـه چـندجمله‌ایهـا بوسـیله آرایـه‌هـا قابل نمایش بودند با استفاده از لیست‌های پیوندی نیز مـی‌تـوان چندجمله‌ایها را نمایش داد. برای این منظور باید ساختار متناسب با چندجمله‌ای تولید کرد. ایـن سـاختار شـامل یـک توان , یک ضریب و یک اشاره‌گر به جمله بعدی چند جمله‌ای است. به این ترتیب در نمایش چندجمله‌ایها بوسیله لیست‌های پیوندی از حافظه بصورت بهتری استفاده شده است ولـی چـون پـیمایش در لیست‌های پیوندی , ترتیبی یا خطی است زمان محاسبات روی چندجمله‌ایها در صورت استفاده از لیست پیوندی افزایش می‌یابد.

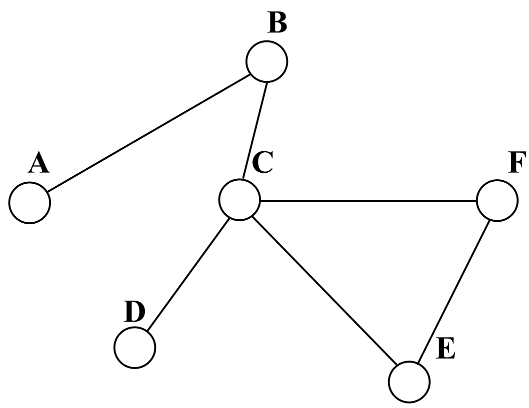
2	1		→	3	5		→	2	0	

$$2x + 3x^5 + 2$$

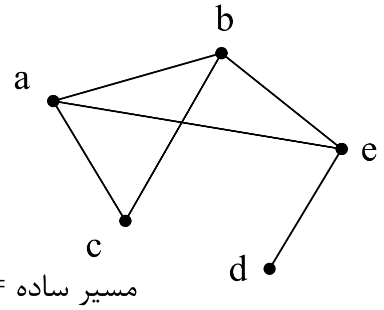
گراف (Graph)

گراف G مجموعه‌ای است از گره‌ها (گره = Vertex) که بوسیله یالهایی (لبه , یال = Edge) به یکدیگر متصل شده‌اند. گراف‌ها می‌توانند جهت‌دار یا غیر جهت‌دار باشند. اگر هر یال در گراف دارای جهت مشخصی باشد بدین معنی که مبدأ و مقصد آن مشخص بوده و غیرقابل جابجایی , این گراف , گراف جهت‌دار خواهد بود. هر دو گره که مستقیماً با یک یال به هم متصل باشند را دو گره مجاور یا همسایه گویند (Adjacement). گرافی که بین تمام گره‌ها مسیر مستقیم وجود داشته باشد را گراف کامل می‌گویند. گراف کامل با n گره را با k_n نمایش می‌دهند.

تعداد کل یالها در گراف کامل غیرجهت‌دار $n(n-\frac{1}{2})$ و در گراف کامل جهت‌دار $n(n-1)$ خواهد بود.



اگر از گره A با عبور از تعدادی یال و گره میانی به گره B در گراف برسیم گوئیم بین A و B یک مسیر از طول n وجود دارد. n تعداد یالهایی است که در مسیر پیموده می‌شوند. اگر در مسیری گره‌ای بیش از یکبار دیده شده باشد آن مسیر , مسیر غیر ساده است در غیر اینصورت مسیر ساده خواهد بود. اگر در یک مسیر , گره مبدأ و گره مقصد بر هم منطبق بودند یا گره مبدأ همان گره مقصد بود , گراف دارای سیکل یا دور است.



c - b - e = مسیر ساده

c - a - e - b = مسیر غیر ساده

a - b - e - a = سیکل

فاصله دو گره در گراف برابر است با کوتاهترین مسیر بین آن دو گره

قطر گراف برابر است با بزرگترین فاصله بین دو گره در گراف

گراف متصل : گرافی که بین هر دو گره مسیری وجود داشته باشد را گراف متصل گویند.

گراف غیر متصل : گرافی است که حداقل بین دو گره آن هیچ مسیر وجود نداشته باشد.

شرط لازم برای اینکه گرافی با n گره متصل باشد این است که حداقل n - 1 یال وجود داشته باشد.

گراف تهی : گرافی است که مجموعه‌ای از گره‌ها باشد و هیچ یالی بین گره‌ها وجود نداشته باشد.

درجه هر گره : تعداد یالهایی که از یک گره عبور می‌کند را درجه آن گره گویند.

تذکر : درجـه خروجی برای گراف‌های جهت‌دار تعداد یالهای خارج شده از یک گره را نشان می‌دهد و درجه ورودی برای گراف‌های جهت‌دار تعداد یالهایی که به یک گره وارد شده‌اند می‌باشد.

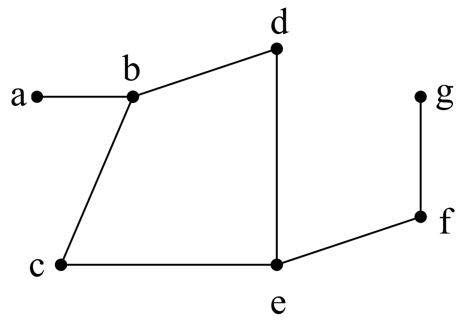
مجمـع درجـات گـره‌هـا در گراف‌های بدون جهت دو برابر تعداد یالهاست و مجموع درجات ورودی یا مجموع درجات خروجی در گراف‌های جهت‌دار تعداد یالها را نشان می‌دهد.

روشهای نمایش گراف‌ها

۱- ماتریس مجاورتی

ماتـریس مجاورتـی روشـی عمومی برای پیاده‌سازی گراف‌ها است. در این روش از یک ماتریس $n \times n$ برای نمایش گراف استفاده می‌کنیم که n تعداد گره‌های گراف است.

گراف وزن‌دار : گراف وزن دار گرافی است که به هر یال آن یک وزن (ارزش) منتصب شده باشد.



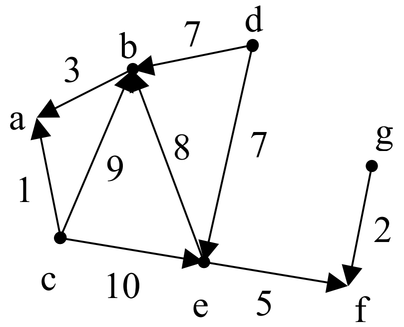
$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array}
\begin{array}{c}
\begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}$$

7×7

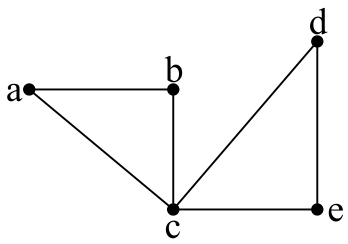
$A[i,j] = 1$ یا w if i, j متصل باشند

$A[i,j] = 0$ if i و j متصل نباشند

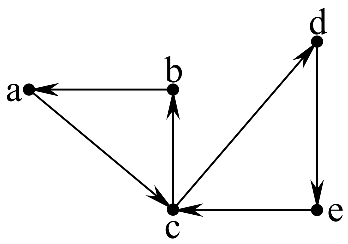
نکته : ماتریس گراف‌های غیر جهت‌دار , ماتریس متقارن است.



$$\begin{array}{c} \\ a\\ b\\ c\\ d\\ e\\ f\\ g \end{array}\begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 9 & 0 & 0 & 10 & 0 & 0 \\ 0 & 7 & 0 & 0 & 7 & 0 & 0 \\ 0 & 8 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \end{bmatrix} \end{array}$$



$$A = \begin{array}{c} \\ a\\ b\\ c\\ d\\ e \end{array}\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



$$\begin{array}{c} \\ a\\ b\\ c\\ d\\ e \end{array}\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

مجموع سطری یا ستونی هر گره در ماتریس برای گراف‌های بدون جهت برابر با درجه هر گره است و مجموع سطری هر گره در ماتریس برای گراف‌های جهت‌دار برابر است با درجه خروجی هر گره و مجموع ستونی هر گره در ماتریس برای گراف‌های جهت‌دار برابر است با درجه ورودی هر گره.

تمرین : ماتریس A صفحه قبل را در نظر گرفته و ماتریس‌های A^2 و A^3 آنرا حساب کنید.

$$\begin{array}{c|ccccc|} & a & b & c & d & e \\ \hline a & 0 & 1 & 1 & 0 & 0 \\ b & 1 & 0 & 1 & 0 & 0 \\ c & 1 & 1 & 0 & 1 & 1 \\ d & 0 & 0 & 1 & 0 & 1 \\ e & 0 & 0 & 1 & 1 & 0 \end{array} \times \begin{array}{c|ccccc|} & a & b & c & d & e \\ \hline a & 0 & 1 & 1 & 0 & 0 \\ b & 1 & 0 & 1 & 0 & 0 \\ c & 1 & 1 & 0 & 1 & 1 \\ d & 0 & 0 & 1 & 0 & 1 \\ e & 0 & 0 & 1 & 1 & 0 \end{array} \Rightarrow A^2 = \begin{array}{c|ccccc|} & a & b & c & d & e \\ \hline a & 2 & 1 & 1 & 1 & 1 \\ b & 1 & 2 & 1 & 1 & 1 \\ c & 1 & 1 & 4 & 1 & 1 \\ d & 1 & 1 & 1 & 2 & 1 \\ e & 1 & 1 & 1 & 1 & 2 \end{array}$$

$$\begin{array}{c|ccccc|} & a & b & c & d & e \\ \hline a & 2 & 1 & 1 & 1 & 1 \\ b & 1 & 2 & 1 & 1 & 1 \\ c & 1 & 1 & 4 & 1 & 1 \\ d & 1 & 1 & 1 & 2 & 1 \\ e & 1 & 1 & 1 & 1 & 2 \end{array} \times \begin{array}{c|ccccc|} & a & b & c & d & e \\ \hline a & 0 & 1 & 1 & 0 & 0 \\ b & 1 & 0 & 1 & 0 & 0 \\ c & 1 & 1 & 0 & 1 & 1 \\ d & 0 & 0 & 1 & 0 & 1 \\ e & 0 & 0 & 1 & 1 & 0 \end{array} \Rightarrow A^3 = \begin{array}{c|ccccc|} & a & b & c & d & e \\ \hline a & 2 & 3 & 5 & 2 & 2 \\ b & 3 & 2 & 5 & 2 & 2 \\ c & 5 & 5 & 4 & \boxed{5} & 5 \\ d & 2 & 2 & 5 & 2 & 3 \\ e & 2 & 2 & 5 & 3 & 2 \end{array}$$

حال برای اینکه ماتریس بدست آمده را امتحان کنیم تا درست بودن آن ثابت گردد یک عدد را در نظر گرفته (مثلاً عدد ۵ که از خانه‌های c - d بدست می‌آید) و حالات موجود را بررسی می‌کنیم :

c - d - c - d

c - b - c - d

c - e - c - d

c - a - c - d

c - d - e - d

نکته : قطر اصلی ماتریس A^2 درجه هر گره را نشان می‌دهد.

عنصر $[\,i\,,\,j\,]$ در ماتریس A^k نشان می‌دهد که چه تعداد مسیر به طول k بین i و j وجود دارد.

فضای لازم برای نمایش دادن یک گراف با استفاده از ماتریس مجاورتی از مرتبه n^2 است که n تعداد گره‌های گراف می‌باشد.

۲- لیست همجواری

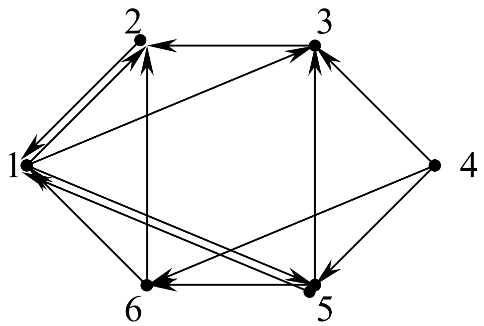
برای نمایش گراف‌ها می‌توانیم از لیست‌های همجواری استفاده کنیم. بدینصورت که آرایه‌ای به تعداد گره‌های گراف ایجاد می‌کنیم. هر عنصر این آرایه اشاره‌گری است به ابتدای یک لیست پیوندی. این لیست پیوندی شامل گره‌هایی است که به گره متناظر با عنصر آرایه متصل‌اند.

در گـراف‌هـای غیـر جهت‌دار تعداد کل گره‌ها در لیست‌های پیوندی دو برابر تعداد یالهای گراف است ولـی در گـراف‌هـای جهت‌دار مجموع تعداد گره‌های لیست‌های پیوندی برابر تعداد یالهای گراف است. فضـای مصـرفی در نمـایش بوسـیله لیسـت همجـواری در گـراف‌هـای جهت‌دار از مرتبه $n + e$ و در گراف‌های غیر جهت‌دار $n + 2e$ است. e تعداد یالهای گراف و n تعداد گره‌های گراف است.

$n = | V |$

$e = | E |$

1	→ 2	→ 3	→ 5 \	
2	→ 1 \			
3	→ 2 \			
4	→ 3	→ 5	→ 6 \	
5	→ 3	→ 6	→ 1 \	
6	→ 2	→ 1 \		

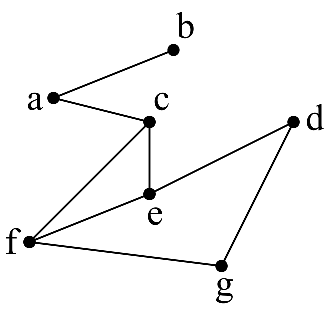


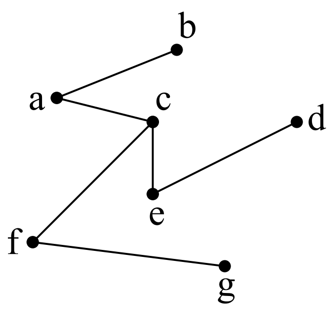
روشهای پیمایش گراف

دو روش کلی برای پیمایش گراف وجود دارد.

۱. اول سطح breadth first search (bfs)

۲. اول عمق depth first search (dfs)





a - b - c - f - e - g – d

(bfs)

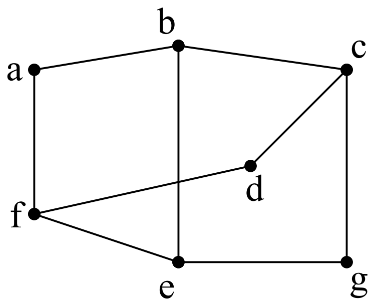
تعریف درخت : درخت گراف متصل بدون سیکل است.

روش اول سطح

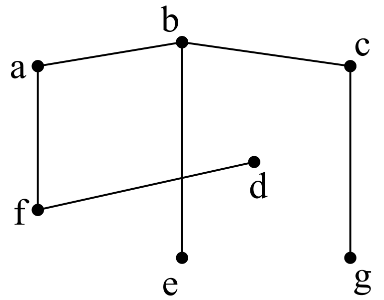
در پیمایش اول سطح با شروع از یک گره و ملاقات آن , کلیه گره‌های مجاور آن نیز ملاقات می‌شوند. سپس این رویه به ترتیب برای هر یک از گره‌های مجاور تکرار می‌شود. برای پیاده‌سازی پیمایش اول سطح (bfs) از ساختمان داده صف استفاده می‌کنیم. بدین ترتیب که رئوس مجاور هنگام ملاقات وارد صف می‌شوند , سپس از سر صف یک عنصر را حذف کرده و گره‌های مجاور آنرا ضمن ملاقات به صف اضافه مـی‌کنـیم. هر گره در صورتی ملاقات می‌شود (وارد صف می‌گردد) که قبلاً ملاقات نشده باشد. نتیجه پیمایش اول سطح , درخت پوشای اول سطح (درخت bfs گراف) می‌باشد.

پیمایش اول سطح از یک گراف و در نتیجه درخت پوشای bfs لزوماً منحصر به فرد نیست.

مثال : گراف زیر را بوسیله روش اول سطح پیمایش کرده و درخت پوشای آنرا بکشید.

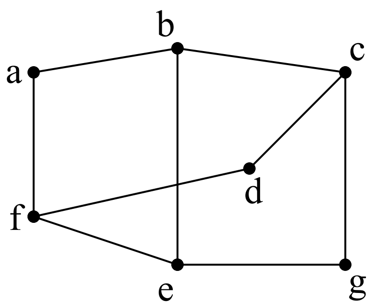


a - b - f - c - e - d - g

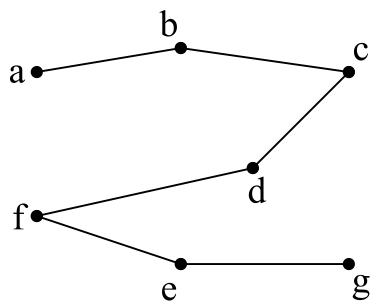


روش اول عمق

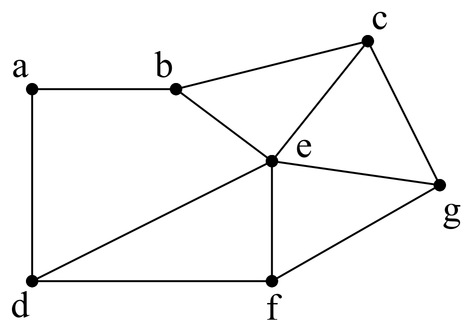
در پیمایش اول عمـق با شروع از یک گره و ملاقات آن , یکی از گره‌های مجاوری که ملاقات نشده را ملاقات می‌نمائیم و این عمل را متناوباً تکرار می‌کنیم. اگر در یک گره بودیم و همه گره‌های مجاور آن ملاقـات شـده بـود , یـک مـرحله بـه عقب برمی‌گردیم. برای پیاده‌سازی درپیمایش اول عمق از پشته استفاده می‌کنیم. نتیجه پیمایش اول عمق , درخت پوشای dfs است که لزوماً منحصر به فرد نیست.



a - b - c - d - f - e - g



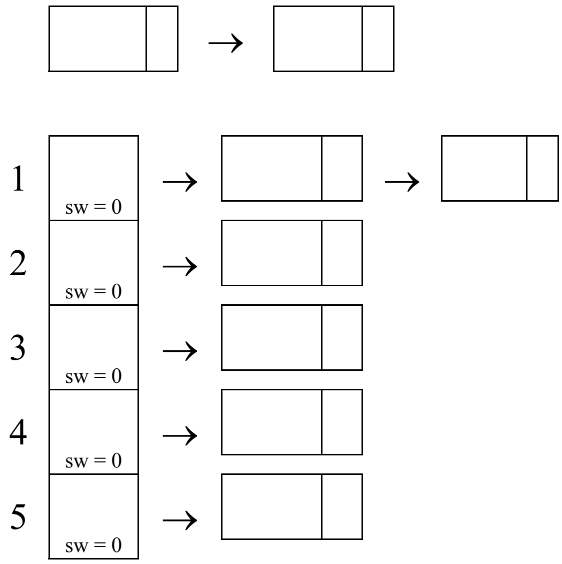
تمرین : از گره e شروع کرده و اول سطح و اول عمق گراف زیر را بنویسید.



اول سطح = e - b - c - d - f - g - a

اول عمق = e - c - b - a - d - f - g

کدنویسی روش اول سطح و اول عمق




```
Struct Linklist
{
        int data ;
        struct Linklist  *Next ;
}
typedef struct Linklist node ;
struct LinkArray
{
        int sw ;
        node  *Link ;
}
typedef struct LinkArray pointer ;
```

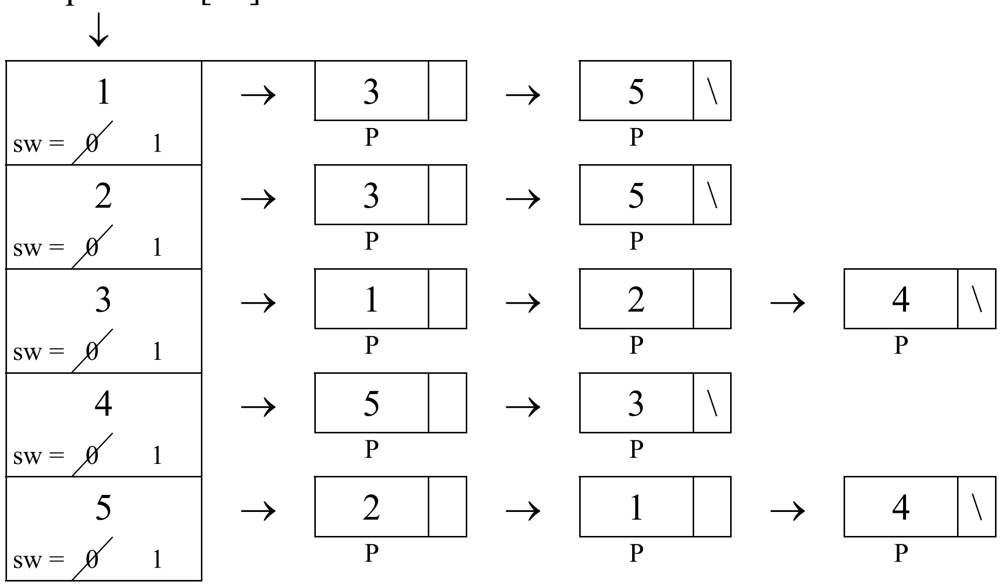
```
pointer  *graphnodes ;
graphnodes = New    pointer [n] ;
```

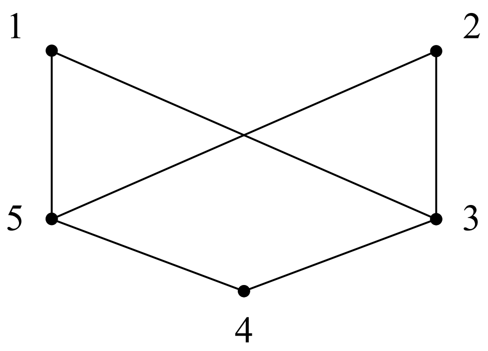
الگوریتم پیمایش اول سطح

در این پیمایش از رأس k شروع می‌کنیم. پس بنابراین داریم :

```
Void bfs ( pointer graphnodes [ ] , int k )
{
        node  *p ;
        graphnodes [ k ]  sw = 1 ;
        C out << k ;
        Addqueue ( k ) ;                        مقدار k را به انتهای صف اضافه می‌کند.
        while ( ! ( queue . empty ( )))         چک می‌کند که آیا صف خالی است یا خیر
        {
                k = delqueue ( ) ;      از صف یک مقدار حذف کرده و در k قرار می‌دهد.
                p = graphnodes [ k ] . Link ;
                do
                {
                        if ( graphnodes [ p → data ] . sw = = 0 )
                        {
                                addqueue ( p → data ) ; C out << p → data ;
                                graphnodes [ p → data ] . sw = 1 ;
                        }
                        p = p → Next ;
                }
                while ( p ) ;
        }
}
```

Graphnodes [5]





1	3	5	2	4

1　3　5　2　4

k
1
3
5
2
4

الگوریتم اول عمق

در این پیمایش از رأس k شروع می‌کنیم. پس بنابراین داریم :

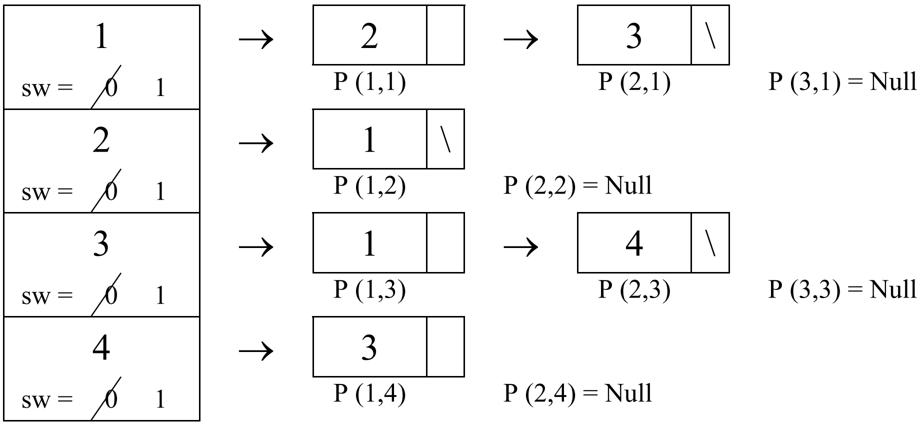
```
Void dfs ( pointer graphnodes [ ] , int k ) //
{
        node  *p ;
        graphnodes [ k ] . sw = 1 ;  C out << k ;
        for ( p = graphnodes [ k ] . Link ;  p ! = Null ;  p = → Next )
        if ( graphnodes [ p → data ] . sw = = 0 )
        dfs ( graphnodes , p → data ) ;
}
```

همان گراف بالا را در نظر می‌گیریم :

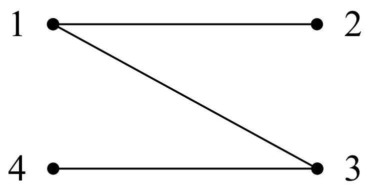
1 sw = ~~0~~ 1	→	3 P (1)	→	5 \\ P (12)		
2 sw = ~~0~~ 1	→	3 P (4)	→	5 \\ P (5)		
3 sw = ~~0~~ 1	→	1 P (2)	→	2 P (3)	→	4 \\ P (11)
4 sw = ~~0~~ 1	→	5 P (9)	→	3 \\ P (10)		
5 sw = ~~0~~ 1	→	2 P (6)	→	1 P (7)	→	4 \\ P (8)

1　3　2　5　4

مثال :



از یک شروع می‌کنیم.



1 2 3 4

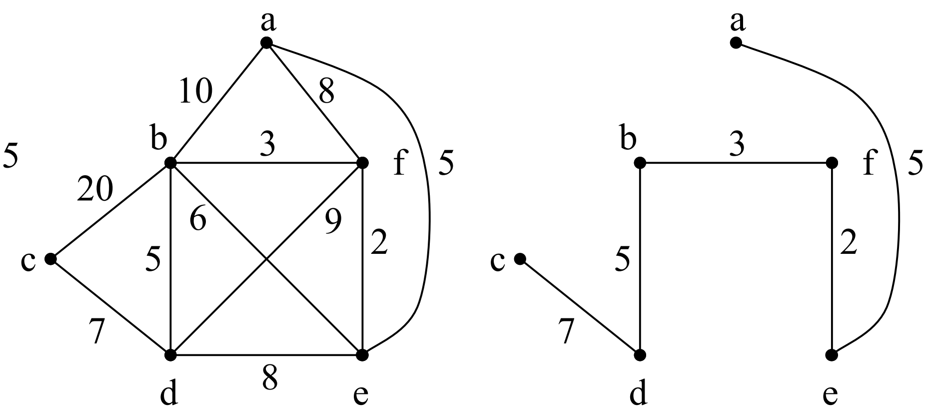
k
1
2
3
4

درخت پوشای بهینه (حداقل هزینه)

درخـت پوشای بهینه در گراف‌های ارزش‌دار (وزن‌دار) ساخته می‌شود و آن درختی است که اگر ارزش تمام گراف‌های آن را جمع کنیم کوچکترین عدد ممکن حاصل گردد.

- **روش اول الگوریـتم کراسـکال (kraskal)** : در الگوریـتم کراسـکال , یالهای گراف را به تـرتیب صـعودی مـرتب می‌کنیم. از اولین (کوچکترین) یال شروع کرده و هر یال را به گراف اضـافه مـی‌کنیم به شرط اینکه دور در گراف ایجاد نگردد. این روال را آنقدر ادامه می‌دهیم تا درخت پوشای بهینه تشکیل گردد.

✓ fe = 2
✓ bf = 3
✓ bd = 5 , ✓ ae = 5
× be = 6
✓ cd = 7
× de = 8 , af = 8
× df = 9
× ab = 10
× bc = 20



2 + 3 + 5 + 5 + 7 = 22

- **روش دوم الگوریــتم پــریم (prim) :** در این روش از یک رأس شروع می‌کنیم و کمترین یـال (یـال با کمترین وزن) که از آن می‌گذرد را انتخاب می‌کنیم. در مرحله بعد یالی انتخاب مـی‌شـود کـه کمتریـن وزن را در بین یالهایی که از دو گره موجود می‌گذرد داشته باشیم. به همیـن ترتیـب در مـرحله بعد یالی انتخاب می‌گردد که کمترین وزن را در بین یالهایی که از سه گره موجود می‌گذرد داشته باشد. این روال را آنقدر تکرار می‌کنیم تا درخت پوشای بهینه حاصـل شـود. بـاید توجه کرد که یال انتخابی در هر مرحله در صورتی انتخاب می‌شود که در گـراف دور ایجـاد نکـند. تفاوت روش پریم با روش کراسکال در این است که گراف حاصل در مـراحل میانـی تشـکیل درخـت پوشـای بهیـنه در روش پـریم همیشـه متصـل است ولی در الگوریتم کراسکال در آخرین مرحله قطعاً متصل است.
- **روش ســوم الگوریتم سولین :** در الگوریتم سولین برای هر گره یال با کمترین هزینه که از آن عـبور مـی‌کند را رسم می‌کنیم. در مرحله بعد ، گراف به مؤلفه‌هایی تقسیم می‌شود و یالی انـتخاب می‌گردد که با کمترین هزینه دو مؤلفه گراف را به همدیگر متصل نماید با شرط عدم وجود دور در گراف. آنقدر این مراحل را ادامه می‌دهیم تا درخت پوشای بهینه حاصل شود.

تمرین : درخت پوشای بهینه گراف زیر را به سه حالت رسم نمائید :

✓ ac = 2 , ✓ de = 2 , ✓ gh = 2

✓ be = 3

✓ eh = 4

✓ ab = 5 , ✓ gi = 5

✗ hi = 6

✓ ef = 7

✗ bc = 8

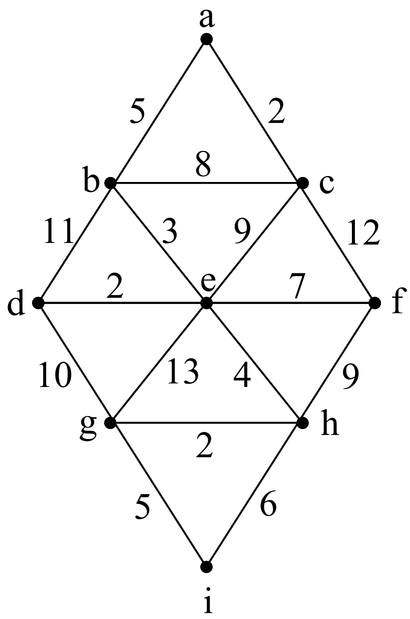
✗ fh = 9 , ✗ ce = 9

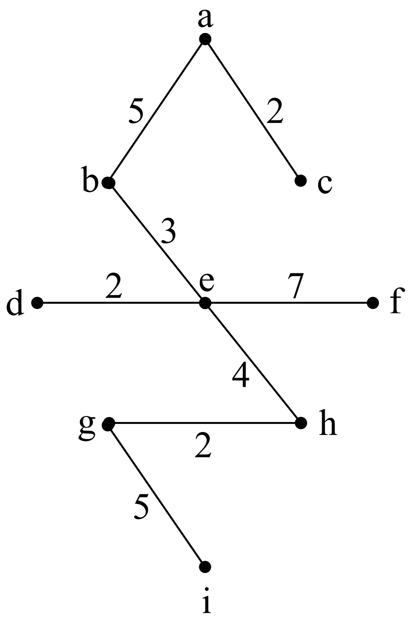
✗ dg = 10

✗ bd = 11

✗ cf = 12

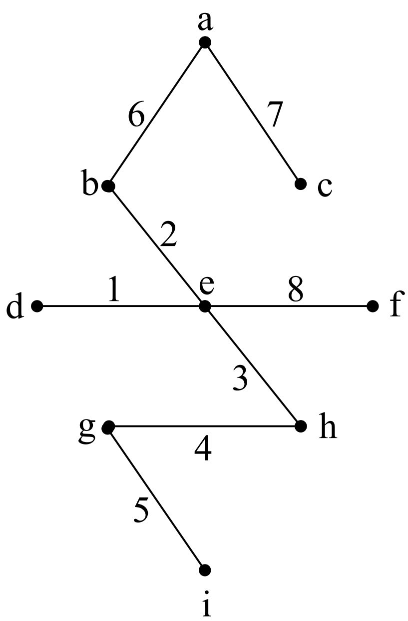
✗ eg = 13



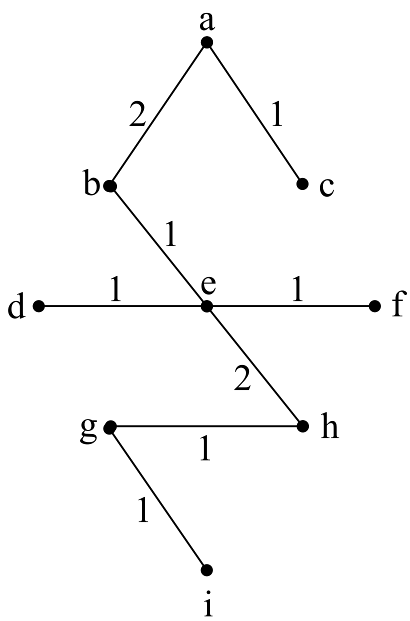


کراسکال

2 + 2 + 2 + 3 + 4 + 5 + 5 + 7 = 30



روش پریم (شروع از e)



روش سولین (شروع از e)

حداقل هزینه بین گره‌های گراف (الگوریتم دایکسترا)

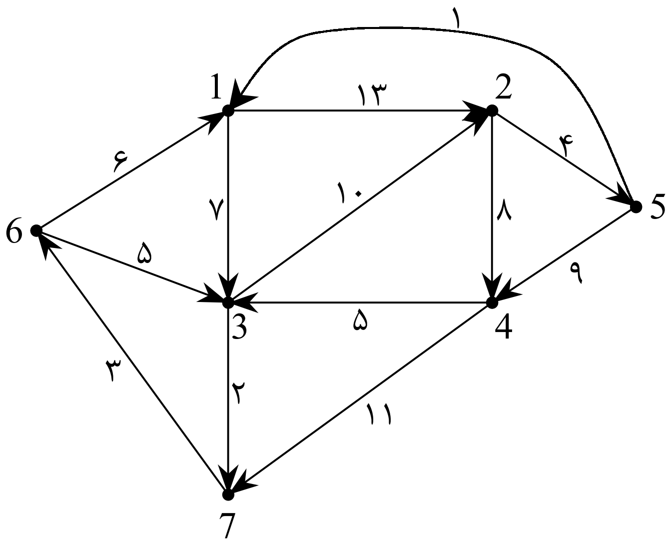
بـرای محاسـبه حداقـل هزیـنه‌هـا از یک گره به گره‌های دیگر در گراف وزن‌دار ، از الگوریتم دایکسترا اسـتفاده مـی‌کنیم. بدین منظور باید ابتا ماتریس هزینه‌های گراف را تشکیل دهیم و سپس با شروع از گـره مفـروض ، هزینه آن گره تا سایر گره‌ها را بدست آوریم. برای بدست آوردن هزینه حداقل بین دو گره دو انتخاب کلی وجود دارد :

۱- مسیر مستقیم بین دو گره W_{ij}

۲- استفاده از یک گره میانی $W_{ik} + W_{kj}$

آنقدر این روال را ادامه می‌دهیم تا تمام گره‌های گراف ملاقات شوند.

مثال : گراف زیر را در نظر بگیرید. از گره شماره ۱ شروع کرده و حداقل هزینه بین گره‌ها را نسبت به گره ۱ بدست آورید.



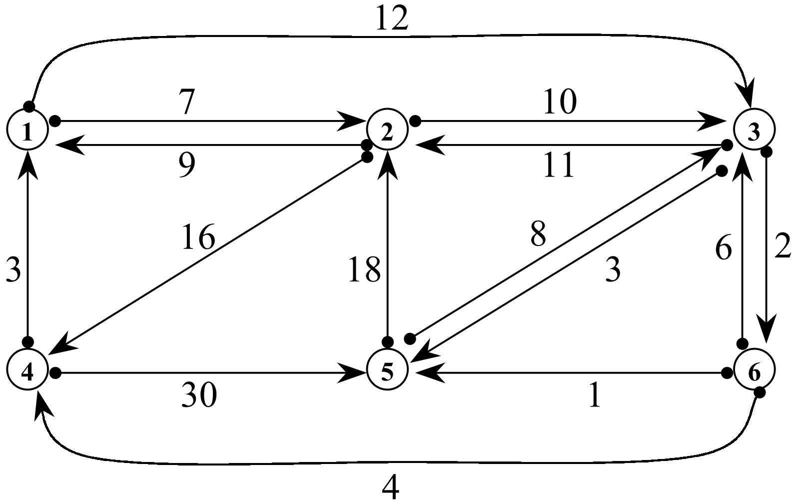
	1	2	3	4	5	6	7
1	0	13	7	∞	∞	∞	∞
2	∞	0	∞	8	4	∞	∞
3	∞	10	0	∞	∞	∞	2
4	∞	∞	5	0	∞	∞	11
5	1	∞	∞	9	0	∞	∞
6	6	∞	5	∞	∞	0	∞
7	∞	∞	∞	∞	∞	3	0

ماتریس

	✓	✓	✓	✓	✓	✓	✓
	1	2	3	4	5	6	7
1	0	13	**7**	∞	∞	∞	∞
3	0	13	7	∞	∞	∞	**9**
7	0	13	7	∞	∞	**12**	9
6	0	**13**	7	∞	∞	12	9
2	0	13	7	21	**17**	12	9
5	0	13	7	**21**	17	12	9
4							

جواب

مثال : حداقل هزینه بین گره‌های گراف زیر را در خصوص گره شماره ۲ محاسبه نمائید.



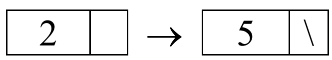
	1	2	3	4	5	6
1	0	7	12	∞	∞	∞
2	9	0	11	16	∞	∞
3	∞	10	0	∞	8	6
4	3	∞	∞	0	30	∞
5	∞	18	3	∞	0	∞
6	∞	∞	2	4	1	0

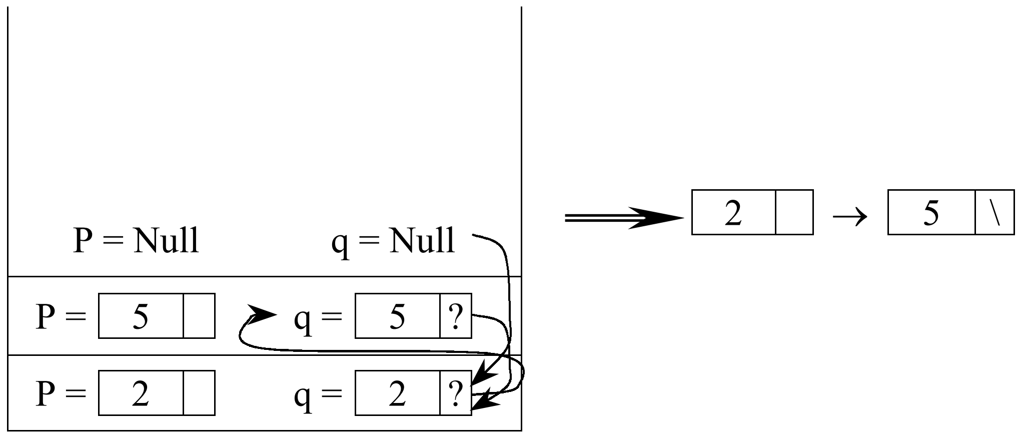
	✓ 1	✓ 2	✓ 3	✓ 4	✓ 5	✓ 6
2	$\boxed{\mathbf{9}}$	0	11	16	∞	∞
1	9	0	$\boxed{\mathbf{11}}$	16	∞	∞
3	9	0	11	$\boxed{\mathbf{16}}$	19	17
4	9	0	11	16	19	$\boxed{\mathbf{17}}$
5	9	0	11	16	$\boxed{\mathbf{18}}$	17

سؤال : تابع f چه عملی انجام می‌دهد؟

```
node * f( node  *p )
{
        node  *q ;
        q = Null ;
        if ( p )
        {
                q = New ( node ) ;
                q → data = p → data ;
                q = Next = f( p → Next ) ;
        }
        return q ;
}
```

جواب :





تابع بالا از لیست پیوندی کی کپی تهیه می‌کند.

سؤال : عبارت prefix زیر را بصورت postfix بنویسید.

+ + a / b – c d / – a b – + c × d 5 / a – b c

جواب :

a b c d – / + a b – c d 5 × + a b c – / – / +

سؤال : حاصل عبارت postfix زیر را بنویسید.

6 , 2 , 3 , + , – , 3 , 8 , 2 , / , + , × , 2 , ↑ , 3 , +

جواب :

$(6 - (2 + 3)) \times (3 + (8 / 2)) \uparrow 2 + 3 = 52$

سؤال : عبارت زیر را بصورت postfix و prefix بنویسید.

$(a / (b - c + d)) \times (e - a) \times c$

جواب :

Postfix = a b c – d + / e a – × c ×

Prefix = × × / a + – b c d – e a c

سؤال : خروجی تابع f را بنویسید.

```
void  f( node  *x )
{
        node  *p ;
        int i ;
        i = 0 ;
        if ( x ! = Null )
        {
                p = x ;
                do
                {
                        i + + ;
                        p = p →Next ;
                }
                while ( p ! = x )
        }
        C out << i ;
}
```

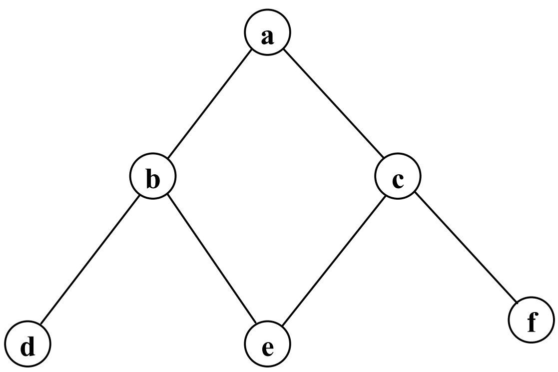
جواب : تعداد ندهای لیست پیوندی چرخشی را محاسبه و چاپ می‌کند.

سؤال : خروجی تابع g را بنویسید.

```
node  *g (node  *p )
{
        node  *m , *L ;
        m = Null ;
        while ( p )
        {
                L = m ; m = p ;
                p = p → Next ;
                m → Link = L ;
        }
        return m ;
}
```

جواب : لیست پیوندی را معکوس می‌کند.

سؤال : الف) از گره a شروع کرده و پیمایش‌های dfs گراف زیر را بنویسید.



a b d e c f

a b e c f d

a c f e b d

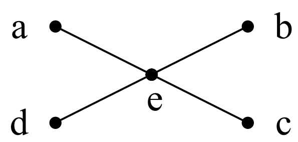
a c e b d f

ب) آیا می‌توان یک dfs و یک bfs در این گراف نوشت که با هم یکی باشند؟

جواب : خیر

ج) در حالت کلی گراف‌ها آیا می‌توان یک dfs و bfs نوشت که با هم برابر باشند یا خیر؟

جواب : بله مثلاً اگر در گراف زیر از گره e شروع کنیم dfs و bfs آن با هم یکی خواهد شد.



dfs = e a b c d

bfs = e a b c d

سؤالات میان ترم

۱- آرایـه‌ای ۱۱ عنصـری بشـکل زیـر موجود است. می‌خواهیم آن را به روش درجی مرتب کنیم. آرایه در مرحله پنجم پویش آن چگونه خواهد بود. (۱۰ نمره)

14	7	3	20	18	4	17	9	11	30	25
1	2	3	4	5	6	7	8	9	10	11

جواب :

3	4	7	14	18	20	17	9	11	30	25
1	2	3	4	5	6	7	8	9	10	11

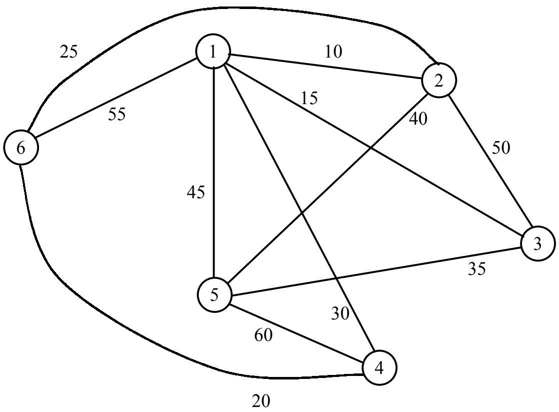
۲- زیربرنامه‌ای بنویسید که آدرس شروع دو لیست پیوندی مرتب را گرفته و آدرس شروع لیست پیوندی مرتب حاصل از ترکیب دو لیست پیوندی داده شده را برگرداند. (۲۰ نمره)

```
node ordermerg ( node * start 1 , node * start 2 )
{
        node * p , * q , * start , * s ;
        s = new ( node ) ; s → next = Null ;
        start = s ;
        p = start 1 → next ;
        q = start 2 → next ;
        while ( p && q )
        if ( p → data < q → data )
        {
                s → next = p ;
                s = p ;
                p = p → next ;
        }
        else
        {
                s → next = q ;
                s = q ;
                q = q → next ;
        }
        if ( p ) s → next = p ;
        else     s → next = q ;
        return start ;
}
```

۳- دو لیست پیوندی با آدرس‌های شروع Start 1 و Start 2 داریم. زیربرنامه‌ای بنویسید که آدرس شروع لیست پیوندی ترکیب این دو لیست را برگرداند. (۱۰ نمره)

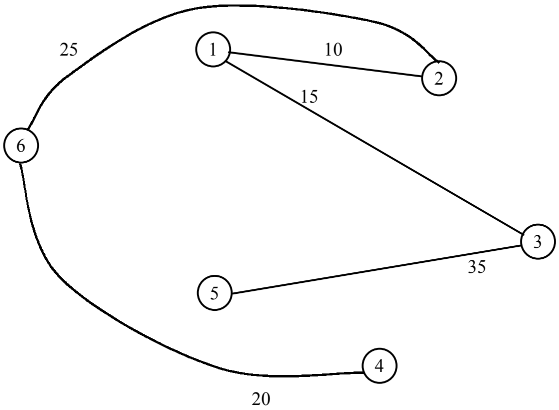
```
node * concatlist ( node * start 1 , node * start 2 )
{
        node * p ;
        p = start 1 → next ;
        while ( p → next )  p = p → next ;
        p → next = start 2 → next ;
        return start 2 ;
}
```

۴- هزینه درخت پوشای مینیمم گراف زیر را از روش پریم بدست آورید. درخت حاصل چند یال دارد؟ ترتیب رسم یالهای درخت پوشای بهینه را با استفاده از الگوریتم پریم ذکر کنید. (۱۵ نمره)



1 – 3 : 15	4 – 5 : 60
1 – 6 : 55	4 – 6 : 20
5 – 5 : 40	4 – 1 : 30
2 – 6 : 25	1 – 5 : 45
2 – 3 : 50	1 – 2 : 10
3 – 5 : 35	

جواب :



$$\left.\begin{array}{l} 1)\ 1-2:10 \\ 2)\ 2-3:15 \\ 3)\ 2-6:25 \\ 4)\ 6-4:20 \\ 5)\ 3-5:35 \end{array}\right\} = 105$$

۵- زیربرنامه‌ای بنویسید که آرایه‌ای از اعداد را به صورت انتخابی (selection sort) مرتب کند. یک آرایه مرتب شده را توسط کدام یک از روشهای مرتب‌سازی گفته شده مجدداً مرتب کنیم تا کندتر مرتب‌سازی انجام شود. (۲۰ نمره)

```
for (i = n ; i > 1 ; – – 1)
{
        max = A[1] ;
        index = 1 ;
        for (i = 2 ; j < = i ; + + j)
        if (A[j] > max)
        {
                max = A[j] ;
                index = j ;
        }
        A[index] = A[i] ;
        A[i] = max ;
}
```

جواب قسمت دوم : اگر یک آرایه مرتب شده داشته باشیم و با مرتب‌سازی درجی یا حبابی آن را مجدداً مرتب کنیم بهترین حالت مرتب‌سازی را انتخاب کرده‌ایم ولی اگر مرتب‌سازی سریع را انتخاب کنیم کندترین حالت را انتخاب کرده‌ایم. حال اگر یک آرایه نامرتب داشته باشیم بهترین حالت برای مرتب‌سازی حالت مرتب‌سازی سریع و یا ادغامی است.

۶- یک آرایه دو بعدی ۲۰ × ۱۰ را بصورت ستونی در حافظه از محل ۱۰۰۰ حافظه ذخیره کرده‌ایم. در صورتیکه هر عنصر آرایه ۲ بایت فضا مصروف کند آدرس عنصر [7 , 6] آرایه در حافظه چیست؟ (۵ نمره)

روش ستونی $[6,7]=[(6\times10)+5]\times2+1000=1130$

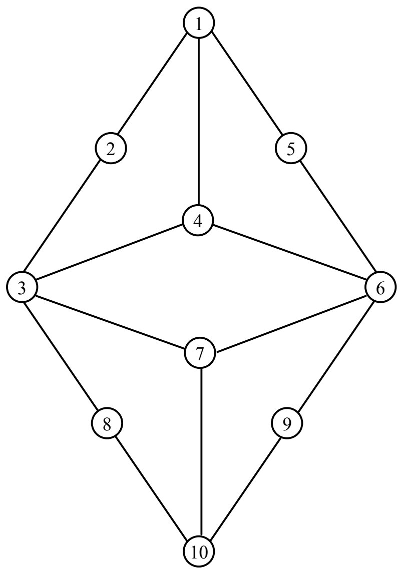
روش سطری $[6,7]=[(5\times20)+6]\times2+1000=1212$

۷- آرایه اعداد در سؤال ۱ را با استفاده از الگوریتم مرتب‌سازی سریع (Quick sort) مرتب می‌کنیم. در مرحله اول مرتب‌سازی (پویش اول آرایه) ، آرایه به چه شکل خواهد بود؟ (۱۰ نمره)

14	7	3	20	18	4	17	9	11	30	25
محور			i_1	i_2	J_3	i_3	j_2	j_1		
4	7	3	11	9	14	17	18	20	30	25

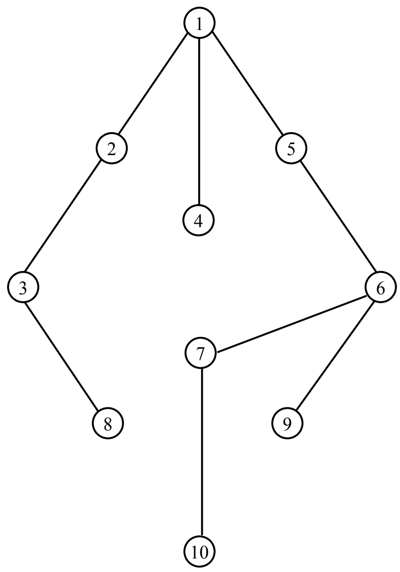
۸- پـیمایش اول عمـق و اول سـطح گراف زیر را بنویسید و درخت پوشای bfs و dfs هر یک را تشـکیل دهـید. پـیمایش اول سـطح را از گره ۵ و پیمایش اول عمق را از گره ۹ شروع کنید. (۲۰ نمره)

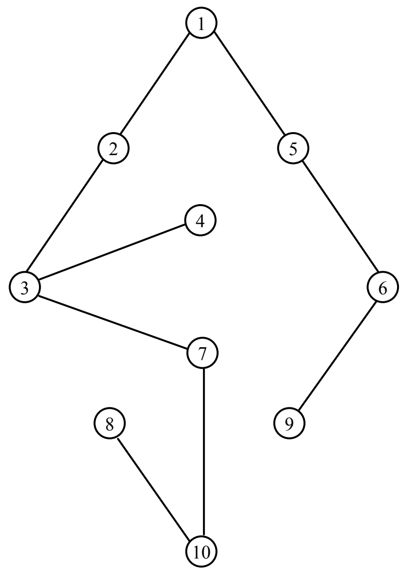
نکته : از بین سؤالات ۱ و ۷ تنها به یکی پاسخ دهید. همه مرتب‌سازی‌ها بصورت صعودی است.



bfs = 5 1 6 2 4 7 9 3 10 8

dfs = 9 6 5 1 2 3 4 7 10 8

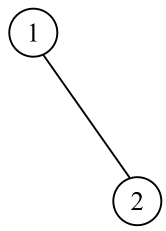




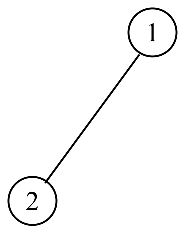
درخت (Tree)

درخت مجموعه‌ای است متناهی از یک یا چند گره که یک گره خاص را بنام ریشه مشخص کرده‌ایم و سایر گره‌ها به مجموعه‌های مجزایی تقسیم می‌شوند که هر مجموعه خود یک درخت است و زیر درخت ریشه نامیده می‌شود. تعداد زیر درخت‌های هر گره درجه آن گره است. فاصله هر گره تا ریشه درخت را سطح آن گره می‌نامند. بزرگترین درجه گره در درخت ، درجه درخت نامیده می‌شود. اگر درجه درخت m باشد درخت را m تایی می‌گویند. به گره‌هایی که درجه آنها صفر است برگ (Leaf) گفته می‌شود. برگها زیر درخت ندارند. برگها را گره‌های خارجی درخت و سایر گره‌ها غیر از برگها را گره‌های داخلی درخت می‌نامند. دو گره که دارای پدر مشترک هستند را گره‌های همزاد گویند. حداکثر سطح یک گره در درخت را ارتفاع (عمق) درخت گویند. پیش فرض سطح ریشه ۱ است.

درخت دودویی طبق تعریف درختی است که درجه آن ۲ باشد یعنی هر گره حداکثر ۲ فرزند داشته باشد. یکی فرزند سمت راست و یکی فرزند سمت چپ که با هم متفاوت (متمایز) هستند.

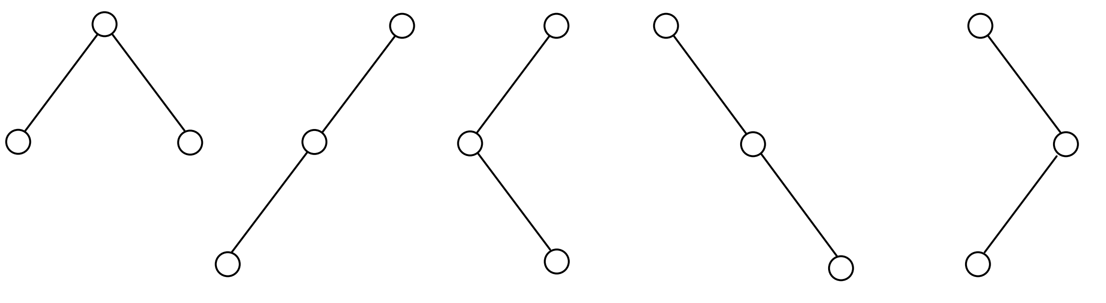


فرزند سمت راست است



فرزند سمت چپ است

مثال : با سه گره چند درخت دودویی می‌توان ساخت.



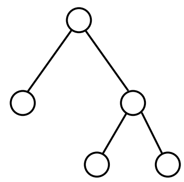
درخت اریب

درخت اریب

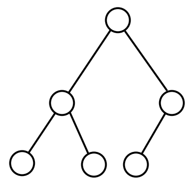
- **درخت Perfect (کاملاً پر)** : درختی است که همه گره‌ها بجز گره‌های سطح آخر (برگها) دارای حداکثر فرزندان بوده (حداکثر درجه درخت) و برگها هم سطح نیز باشند.
- **درخت Complete (کامل)** : درختی است که اگر گره‌های آنرا شماره‌گذاری کنیم ،

شـماره‌هـا بر درخت Perfect متناظرش منطبق باشند. یعنی می‌توان گفت درختی است که اگـر ارتفـاع آن d باشـد تـا ارتفـاع d - 1 ، درخـت Perfect بوده و برگها در سطح d تا حد ممکن در سمت چپ باشند.

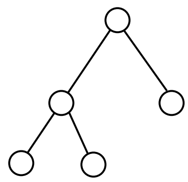
- **درخت Full (پر)** : درختی که در آن گره‌ها یا برگ هستند و یا به تعداد درجه درخت فرزند دارند را درخت full گویند.



Full است
Complete نیست
Perfect نیست



Complete است
Full نیست
Perfect نیست

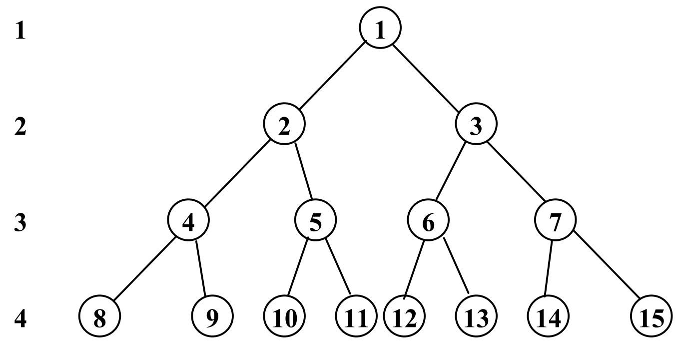


Perfect نیست
Full است
Complete است

- درختی که Perfect باشد حتماً Complete و Full هم هست.
- درختی که Complete است لزوماً Perfect نیست.
- درختی که Complete است لزوماً Full نیست.
- درختی که Full است لزوماً Complete نیست.

- حداکثر تعداد گره‌ها در یک درخت دودویی به ارتفاع d براب با $2^d - 1$ خواهد بود.
- حداثر تعداد برگها در یک درخت دودویی به ارتفاع d برابر با 2^{d-1} خواهد بود.
- حداکثر تعداد گره‌های غیر برگ یک درخت با ارتفاع d برابر با $2^{d-1} - 1$ خواهد بود.
- درختی دودویی با n گره دارای ارتفاع Log_2^{n-1} خواهد بود.

سؤال : درخت دودویی زیر را در نظر بگیرید به سؤالهای آن پاسخ دهید.



۱- حداکثر چند برگ وجود دارد؟

حداکثر تعداد برگها از رابطه 2^{d-1} بدست می‌آید. اگر بعنوان مثال ارتفاع 4 را در نظر بگیریم داریم :

$2^{d-1} = 2^{4-1} = 2^3 = 8$ حداکثر تعداد برگهای موجود در این درخت

۲- حداکثر چند گره غیر از برگ داریم؟ (گره‌های داخلی)

حداکثر گره‌های داخلی از رابطه $2^{d-1} - 1$ بدست می‌آید. باز هم در ارتفاع 4 داریم :

$$2^{d-1} - 1 = 2^{4-1} - 1 = 2^3 - 1 = 8 - 1 = 7$$

۳- حداکثر چند گره وجود دارد؟

حداکثر تعداد گره‌ها از رابطه $2^d - 1$ بدست آمده و بعنوان مثال باز هم در ارتفاع 4 داریم :

$$2^d - 1 = 2^4 - 1 = 16 - 1 = 15$$

۴- چند درخت کامل متمایز به ارتفاع d داریم؟

حداکـثر تعـداد درخـت کـامل مـتمایز نیز از همان رابطه تعداد برگها بدست می‌آید. پس در ارتفاع 4 داریم :

$$2^{d-1} = 2^{4-1} = 2^3 = 8$$

سؤال : اگر n تا گره داشته باشیم :

الف) حداکثر عمق چقدر است؟

حداکـثر عمـق برابـر بـا n خواهد بود. در این حالت درخت بصورت کاملاً اریب خواهد بود. یعنی تمام فرزندان از یک سمت (چپ یا راست) رشد می‌کنند.

ب) حداقل عمق چقدر است؟

حداقل ارتفاع یک درخت دودویی با n گره از رابطه زیر بدست می‌آید :

$$\left[\mathrm{Log}_2^n\right] + 1 \Rightarrow \left[\mathrm{Log}_2^8\right] + 1 = 3 + 1 = 4$$

نکته : همه روابط گفته شده برای درخت‌های m تایی نیز قابل تعمیم است.

اگر n_0 تعداد برگهای در یک درخت دودویی و n_2 تعداد گره‌های دو فرزندی باشند رابطه $n_0 = n_2 + 1$ برقرار است.

روشهای پیمایش درخت

۱- آرایه

برای نمایش درخت‌های دودویی می‌توان از آرایه‌ها استفاده کرد. بدین منظور به تعداد گره‌های درخت کامل متناظر با درخت مفروض برای یک آرایه حافظه نیاز داریم . در آنصورت داریم :

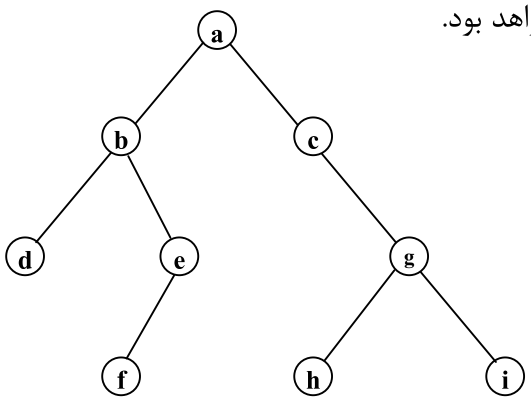
- ریشه در خانه اول آرایه قرار می‌گیرد.
- فرزند سمت چپ گره‌ای با اندیس i در آرایه درون خانه $2i$ قرار می‌گیرد که $2i \le n$ خواهد بود. اگر $2i > n$ بود یعنی گره i فرزند سمت چپ ندارد.

فرزند سمت چپ $= 2i$

- فرزند سمت راست گره‌ای با اندیس i در آرایه درون خانه $2i + 1$ آرایه قرار می‌گیرد که $2i + 1 \le n$ خواهد بود. اگر $2i + 1 > n$ باشد یعنی گره i فرزند سمت راست ندارد.

فرزند سمت راست $= 2i + 1$

در نمایش درخت‌های دودویی بوسیله آرایه‌ها اگر درخت کامل نباشد اتلاف حافظه خواهیم داشت ولی اگر درخت کامل باشد روش خوبی خواهد بود.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	b	c	d	e		g			f				h	I

فرزند سمت چپ $= 2i$

فرزند سمت راست $= 2i + 1$

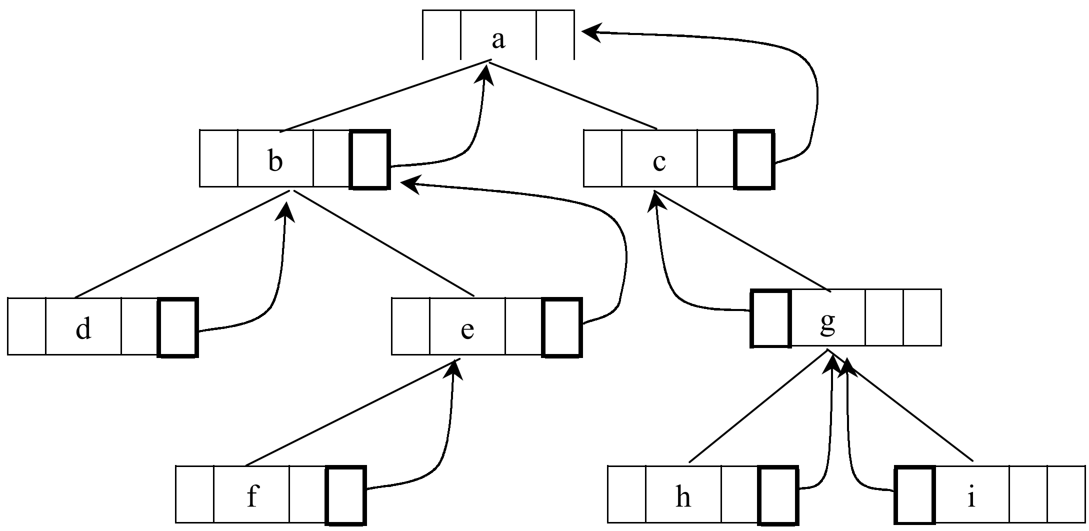
$\Rightarrow$ i پدر هر گره $= \dfrac{i}{2}$

۲- لیست‌های پیوندی

برای نمایش درخت‌ها به روش لیست پیوندی باید گره‌هایی با ساختار زیر تعریف کنیم.

هر گره یک فرزند سمت چپ و یک فرزند سمت راست و یک داده دارد. در ساختار تعریف شده مشخص کردن پدر هر گره به سادگی امکان پذیر نیست. برای رفع این مشکل می‌توان در ساختار هر گره یک فیلد جدید به نام Parent که به پدر آن گره اشاره می‌کند تعریف نمود.

```
Struct Treetype
{
        int data ;
        struct Treetype * Lchild ;
        struct Treetype * Rchild ;
        struct Treetype * Parent ;
}
typedef struct Treetype Tree ;
```

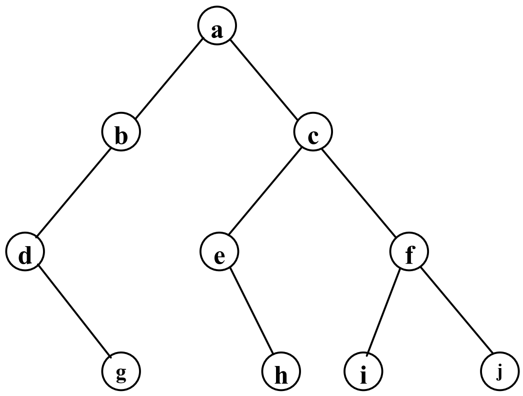


روشهای پیمایش درخت

۱- پیمایش اول عمق (عمقی)

سه روش پیمایش عمقی به شرح ذیل می‌باشد :

	سوم	دوم	اول	
Preorder (پیش ترتیب)	فرزند سمت راست	فرزند سمت چپ	ریشه	1.
Postorder (پس ترتیب)	ریشه	فرزند سمت راست	فرزند سمت چپ	2.
Inorder (میان ترتیب)	فرزند سمت راست	ریشه	فرزند سمت چپ	3.



Preorder = a b d g c e h f i j

Postorder = g d b h e i j f c a

Inorder = d g b a e h c i f j

الگوریتم پیمایش عمقی به روش Inorder

```
Void  Inorder  ( Tree * T )
{
        if ( T ! = Null )
        {
                Inorder ( T → Lchild ) ;
                C out << T → data ;
                Inorder ( T → Rchild );
        }
}
```

الگوریتم پیمایش عمقی به روش Preorder

```
Void  Preorder  ( Tree * T )
{
        if ( T ! = Null )
        {
                C out << T → data ;
                Preorder  ( T → Lchild ) ;
                Preorder  ( T → Rchild );
        }
}
```

الگوریتم پیمایش عمقی به روش Postorder

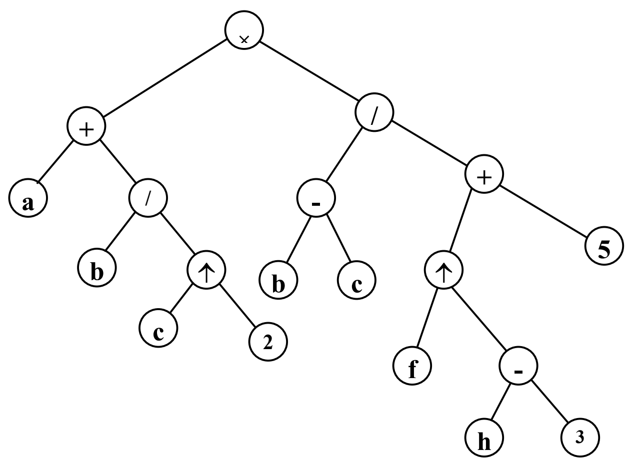
```
Void  Postorder ( Tree * T )
{
        if ( T ! = Null )
        {
                Postorder ( T → Lchild ) ;
                Postorder ( T → Rchild );
                C out << T → data ;
        }
}
```

درخت متناظر با عبارت infix

عبارت infix زیر را در نظر می‌گیریم.

$$\left(a + b / \left(c \uparrow 2\right) \times \left(b - c\right) / \left(f \uparrow \left(h - 3\right) + 5\right)\right)$$

هر عبارت infix یک درخت دودویی دارد.



$\text{Inorder} = a + b / c \uparrow 2 \times b - c / f \uparrow h - 3 + 5$

این همان عبارت infix بدون در نظر گرفتن پرانتزها است.

$\text{Preorder} = \times + a / b \uparrow c2 / - bc + \uparrow f - h35$

این همان عبارت prefix بدون در نظر گرفتن پرانتزها است.

$\text{Postorder} = abc2 \uparrow / + bc - fh3 - \uparrow 5 + / \times$

این همان عبارت postfix بدون در نظر گرفتن پرانتزها است.

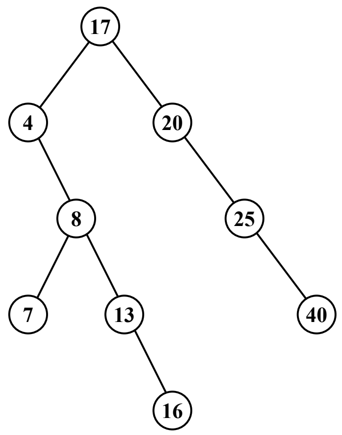
۲- پیمایش اول سطح (سطحی)

```
Void Levelorder ( Tree * T )
{
        while ( T )
        {
                C out << T → data ;
                if ( T → Lchild ) addqueue ( T → Lchild ) ;
                if ( T → Rchild ) addqueue ( T → Rchild ) ;
                T = delqueue ( ) ;
        }
}
```

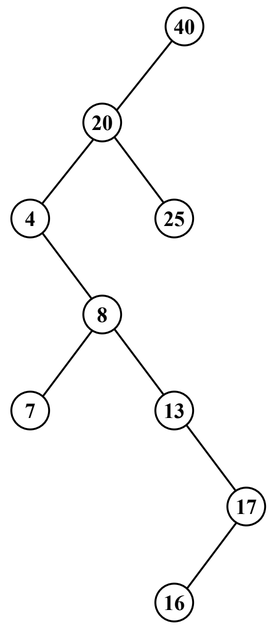
درخت جستجوی دودویی BST (Binary Search Tree)

ساختمان داده‌هایی که تا کنون بررسی شده‌اند هر یک دارای نقاط ضعفی هستند. مثلاً درج در آرایه مرتب مستلزم شیفت دادن داده‌ها و در نتیجه کندتر شدن الگوریتم است. پیمایش‌های مختلف روی لیست‌های پیوندی نیز بصورت خطی انجام می‌شود که هزینه انجام اعمال را بالا می‌برد. درخت‌های جستجوی دودویی راهکاری پیشنهاد می‌کنند که هزینه انجام اعمال اصلی مانند حذف ، اضافه و جستجو با زمان متوسط بهتری انجام می‌شود. این زمان برابر است با ارتفاع درخت که از Log_2^n تا n متغیر است. ترتیب ورود عناصر یا کلیدها برای تشکیل درخت BST از آنها کاملاً مؤثر است. کلیدهای یکسان با ترتیب متفاوت ، درخت‌های BST متفاوتی ایجاد می‌کنند.

17 20 25 40 4 8 7 13 16



40 20 4 8 13 7 17 25 16



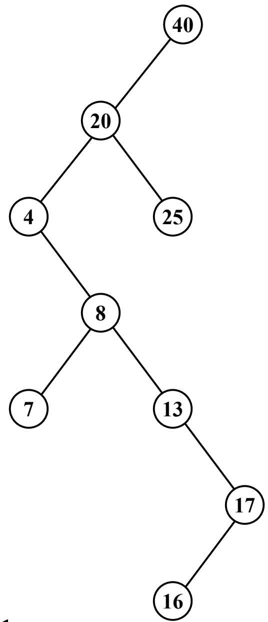
اگـر درخـت BST را بصـورت inorder پـیمایش کنـیم در خروجـی لیسـت مرتب صعودی خواهیم داشت.

هزینه ساخت یک درخت دودویی BST از یک آرایه n تایی ورودی (نامرتب) $n \times Log_2^n$ است.

الگوریتم جستجو در یک درخت دودویی BST

```
int  BSTSearch ( node * T , int * x )
{
        int founded = 0 ;
        if ( T )
        {
                if ( T → data < x )
                        founded = BSTSearch ( T → Right , x ) ;
                else if ( T → data > x )
                        founded = BSTSearch ( T → Left , x ) ;
                else founded = 1 ;
        }
        return founded ;
}
```

مثال : می‌خواهیم ببینیم عدد ۱۳ در درخت زیر وجود دارد یا خیر؟



13 ← T	x = 13	F =	~~0~~ = 1
8 ← T	x = 13	F =	~~0~~ = BSTSearch (13 , 13)
4 ← T	x = 13	F =	~~0~~ = BSTSearch (8 , 13)
20 ← T	x = 13	F =	~~0~~ = BSTSearch (4 , 13)
40 ← T	x = 13	F =	~~0~~ = BSTSearch (20 , 13)

خروجی = founded = 1

وقتی خروجی برابر با 1 باشد یعنی داده پیدا شده و در صورتیکه 0 باشد یعنی داده پیدا نشده است.

الگوریتم اضافه کردن داده به درخت دودویی BST

```
Void  insertBST ( node * T , int x )
{
        node * p , * q , * S ;
        p = new (node) ; p → data = x ;
        p → Right = Null ; p → Left = Null ;
        S = T ;
        While ( S && S → data != x )
        {
                if ( S → data > x ) { q = S ; S = S → Left ; }
                else if ( S → data < x ) { q = S ; S = S → Right ; }
        }
        if ( !S ) if ( q → data > x ) q → Left = p ;
                else q → Right = p ;
}
```

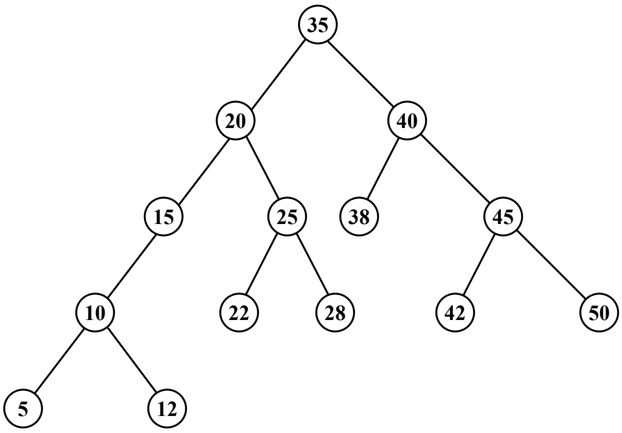
حذف

بـرای حـذف یک گره از درخت جستجو دودویی ابتدا باید آن گره را در درخت BST پیدا کنیم. حال یکی از وضعیتهای زیر رخ می‌دهد :

۱- اگـر گـره مـورد نظـر برگ باشد حذف می‌شود یعنی حافظه گرفته شده برای گره آزاد شده و اشاره‌گر پدرش Null می‌شود.

۲- اگـر گـره حـذف شدنی فقط یک فرزند داشته باشد فرزند آن گره جایگزین گره حذف شدنی می‌گردد و یا می‌توان مورد بعدی را انجام داد.

۳- اگـر گـره دارای دو فـرزند باشد یک قدم به راست و سپس آنقدر به چپ می‌رویم تا به Null برسـیم و یـا بـرعکس یک قدم به چپ و سپس آنقدر به راست می‌رویم تا به Null برسیم. با دنـبال کردن هر یک از حالات فوق گره آخر را جایگزین گره حذف شدنی کرده و حافظه آنرا آزاد می‌کنیم.



❖ اگـر بخواهـیم گره شماره ۴۰ را حذف کنیم هم می‌توانیم گره شماره ۳۸ و هم می‌توانیم گره شماره ۴۲ را جایگزین آن کنیم.

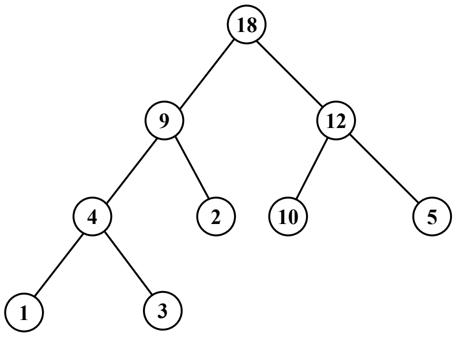
❖ اگـر بخواهـیم گره شماره ۲۰ را حذف کنیم هم می‌توانیم گره شماره ۱۵ و هم می‌توانیم گره شماره ۲۲ را جایگزین آن کنیم.

❖ اگـر بخواهـیم گره شماره ۱۵ را حذف کنیم هم می‌توانیم گره شماره ۱۰ و هم می‌توانیم گره شماره ۱۲ را جایگزین آن کنیم.

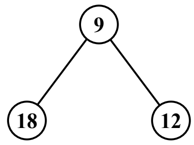
درخت heap (کپه)

درختـی اسـت دودویـی کامل (Complete) که تعداد موجود در هر گره از مقدار موجود در گره‌های فـرزندانش کوچکـتر نباشـد. ایـن کـپه ، کـپه بیشـترین (maxheap) اسـت. در صورتیکه در درخت دودویـی کامل مقدار هر گره از مقدار گره فرزندانش بیشتر نباشد کپه کمترین (minheap) خواهیم داشت.

maxheap



minheap



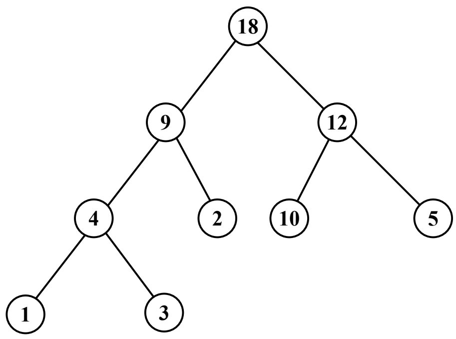
در maxheap بزرگترین عنصر را می‌توان با مرتبه زمانی $O(1)$ بدست آورد (یعنی بدون محاسبه زیرا ریشه درخت بزرگترین عنصر است) و متقابلاً در minheap کمترین عنصر را می‌توان با مرتبه زمانی $O(1)$ بدست آورد (در این حالت نیز کمترین عنصر همان ریشه درخت است).

افزودن داده به درخت heap

برای افزودن داده جدید به درخت heap داده جدید را در انتهای لیست آرایه قرار می‌دهیم (توجه اینکه درخت heap را درون آرایه نگهداری می‌کنیم). روال زیر تا رسیدن به ابتدای آرایه و یا بر قرار بودن شرط درخت heap انجام می‌شود :

- ✓ داده موجود در خانه i (برای اولین بار آخرین عنصر آرایه) با پدر خویش در خانه $\frac{i}{2}$ مقایسه می‌شود. در صورت جابجایی مجدداً این مقایسه برای خانه جدید ($\frac{i}{2}$) انجام می‌گردد. این روش را افزودن به طریقه درج در heap می‌نامند.
- ✓ درج در درخت heap برای هر عنصر با مرتبه زمانی Log_2^n انجام می‌شود.

درخت زیر را در نظر بگیرید :



آرایه این درخت به شکل زیر خواهد شد.

1	2	3	4	5	6	7	8	9
18	9	12	4	2	10	5	1	3

حال می‌خواهیم داده شماره 8 را به این آرایه اضافه کنیم و درخت heap نیز برقرار باشد.

1	2	3	4	5	6	7	8	9	10
18	9	12	4	~~2~~	10	5	1	3	~~8~~
18	9	12	4	8	10	5	1	3	2

حال می‌خواهیم ابتدا داده شماره 7 و سپس داده شماره 25 را به آرایه اضافه کنیم.

1	2	3	4	5	6	7	8	9	10	11	12
18	9	12	4	8	~~10~~	5	1	3	2	7	~~25~~
18	9	~~12~~	4	8	~~25~~	5	1	3	2	7	10
~~18~~	9	~~25~~	4	8	12	5	1	3	2	7	10
25	9	18	4	8	12	5	1	3	2	7	10

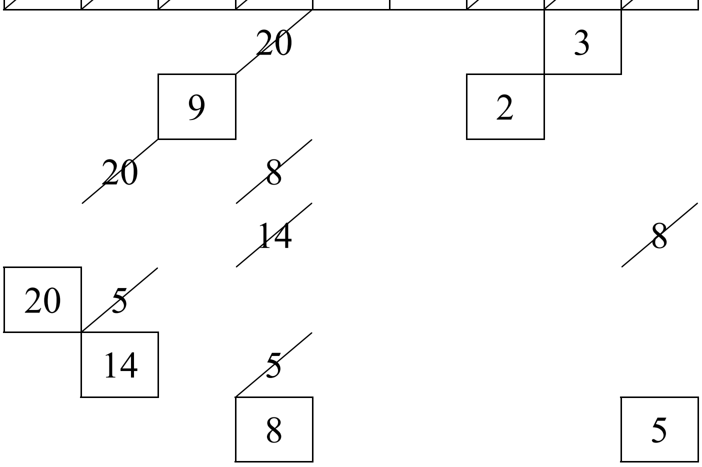
چــون داده شــماره 7 بــر ســر جــای خود درست قرار گرفته است پس آنرا دست نمی‌زنیم و فقط داده شماره 25 را آنقدر جابجا کرده تا درخت heap برقرار باشد.

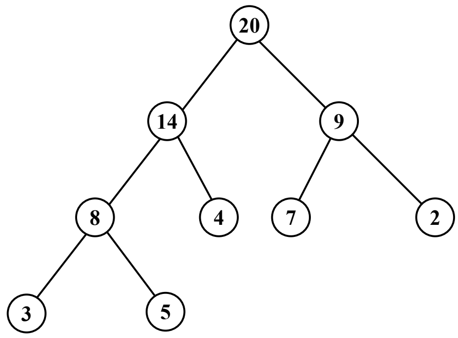
<u>روش ساخت درخت دودویی heap به روش جوان‌ترین پدر</u>

در ایجـاد کـپه بـه روش جوان‌ترین پدر ابتدا همه n عنصر ورودی را در یک آرایه قرار دهید. سپس از پائیــن درخــت شروع کرده ، هر پدر و فرزندانش را بصورت کپه تنظیم می‌کنیم و به سمت بالا (ریشه) حرکـت مــی‌کنــیم. همیــنطور کـه به سمت ریشه می‌رویم زیردرخت‌ها بصورت کپه درآمده‌اند. در این روش چـون بـرگها خودبخـود بـه تنهایی یک heap هستند باید از جوان‌ترین پدر شروع کنیم که اگر عناصر آرایه i تا باشند از عنصر $\frac{i}{2}$ شروع می‌کنیم.

مثال : درخت زیر را به روش جوان‌ترین درخت به صورت درخت دودویی heap درآورید.

1	2	3	4	5	6	7	8	9
~~5~~	~~8~~	~~2~~	~~3~~	4	7	~~9~~	~~20~~	~~14~~

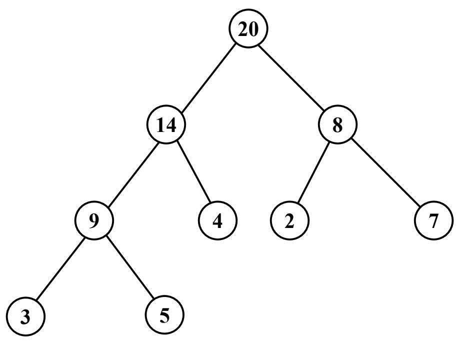




20	14	9	8	4	7	2	3	5

مثال : آرایه زیر را به روش درج بصورت درخت heap بنویسید.

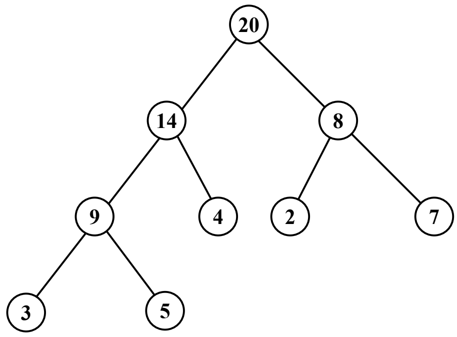
1	2	3	4	5	6	7	8	9
5	8	2	3	4	7	9	20	14



~~5~~	~~8~~							
8	5	~~2~~	3	4	~~7~~			
8	5	~~7~~	3	4	2	~~9~~		
8	5	9	3	4	2	7	20	
~~8~~	5	~~9~~	20	4	2	7	3	14
9	~~5~~	8	~~20~~	4	2	7	3	14
9	20	8	~~5~~	4	2	7	3	14
~~9~~	~~20~~	8	14	4	2	7	3	5
20	~~9~~	8	~~14~~	4	2	7	3	5
20	14	8	9	4	2	7	3	5

حذف یک عنصر

برای حذف یک عنصر از درخت heap ، ریشه درخت خارج شده و داده آخر به جای آن قرار می‌گیرد. سـپس از ابـتدای آرایه (i =) شروع می‌کنیم و بین 2i و 2i + 1 عنصر ماکزیمم را در صورت لزوم با عنصـر i عـوض می‌کنیم. به همین ترتیب جابجایی انجام گرفته تا زمانیکه به انتهای آرایه برسیم (یک گره فرزندی نداشته باشد).



1	2	3	4	5	6	7	8	9
~~20~~	14	9	8	4	7	2	3	~~5~~
~~5~~	~~14~~	9	8	4	7	2	3	
14	~~5~~	9	~~8~~	4	7	2	3	
14	8	9	5	4	7	2	3	
14	8	9	5	4	7	2	3	

در این مثال عنصر 20 را حذف کرده و عنصر 5 را جایگزین کرده و بقیه مراحل ساخت درخت heap را انجام داده‌ایم.

سؤال : تابع f چه کاری انجام می‌دهد؟

```
int f (node * T )
{
        int L , r ;
        if ( T )
        {
                L = f ( T → Left ) ;
                R = f ( T → Right ) ;
                if  L > r  return  L + 1 ;
                else return  r + 1 ;
        }
        return  0 ;
}
```

جواب : ارتفاع درخت را نشان می‌دهد.

سؤال : تابع f چه کاری انجام می‌دهد؟

```
node  * f ( node * T )
{
        node  * r , * s , * q ;
        q = Null ;
        if ( T )
        {
                r = f ( T → Left ) ;
                s = f ( T → Right ) ;
                q = New ( node ) ;
                q → Left = r ;
                q → Right = s ;
                q → data = T → data ;
        }
        return  q ;
}
```

جواب : از یک کپی تهیه می‌کند.

سؤال : تابع g چه کاری انجام می‌دهد؟

```
int  g ( node * T )
{
        if ( T )
        {
                if ( ! T → Left ) && ( ! T → Right )
                        return  1 ;
                else
                        return ( g ( T → Left ) + g ( T → Righe ) + 1 ) ;
        }
        return  0 ;
}
```

جواب : تعداد گره‌ها را چاپ می‌کند.

سؤال : تابع h چه کاری انجام می‌دهد؟

```
int  h ( node * T )
{
        if ( T )
        {
                if ( T → Left = = Null ) && ( T → Right = = Null )
                        return  1 ;
                else
                        return ( h ( T → Left ) + h ( T → Righe ) ) ;
        }
        return  0 ;
}
```

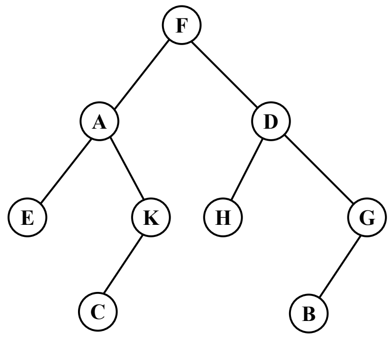
جواب : تعداد برگها را چاپ می‌کند.

سؤال : پیمایش Inorder و Preorder زیر را داریم. درخت دودویی آنرا تشکیل داده و پیمایش Postorder آنرا بنویسید.

Inorder :   E A C K F H D B G

Preorder :   F A E K C D H G B

Postorder :   E C K A H B G D F

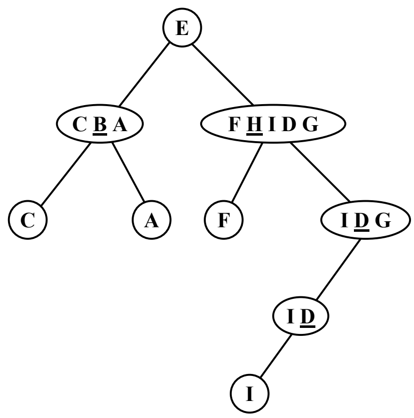


سؤال : پیمایش Inorder و Postorder زیر را داریم. درخت دودویی آنرا تشکیل داده و پیمایش Preorder آنرا بنویسید.

Inorder :   C B A E F H I D G

Postorder :   C A B F I D G H E

Preorder :   E B C A H F G D I



سؤال : درخت maxheap آرایه زیر را از دو روش درج و جوان‌ترین پدر بدست آورید.

1	2	3	4	5	6	7	8
~~10~~	15	~~22~~	~~4~~	11	~~23~~	~~19~~	~~14~~

به روش جوان‌ترین پدر

			14				4
		~~23~~			~~22~~	~~19~~	
23	15	~~10~~					
		22			10	19	

23	15	22	14	11	10	19	4

1	2	3	4	5	6	7	8
10	15	22	4	11	23	19	14

به روش درج

~~10~~	~~15~~						
~~15~~	10	~~22~~					
22	~~10~~	15	4	~~11~~			
22	11	~~15~~	4	10	~~23~~		
22	11	23	~~4~~	10	15	19	~~14~~
~~22~~	11	~~23~~	14	10	15	19	4
23	~~11~~	22	~~14~~	10	15	19	4
23	14	22	11	10	15	19	4

23	14	22	11	10	15	19	4