



جزوه درس ساختمان داده ها

مدرس:

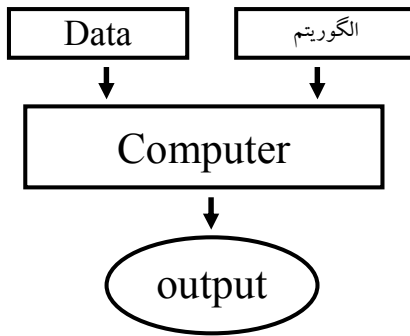
علی پاسبان اسدآبادی

وب سایت: www.alipasban.ir

ساختمان داده:

✓ به مدل ریاضی سازماندهی داده ها ، ساختمان داده گفته میشود.

✓ به ساختارهایی که جهت ذخیره سازی ، بازیابی و ... اطلاعات بکار می روند ساختمان داده گفته میشود.



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

✓ الگوریتم: که باید مناسب و کارا باشد.

✓ ساختمان داده مناسب: تا داده ها به درستی در آن سازماندهی شوند.

🚦 معیار برتری یک الگوریتم نسبت به الگوریتم دیگر چیست؟

با توجه به اینکه برای انجام هر مساله ای الگوریتمهای متفاوتی وجود دارد ، باید کاراترین راه حل برای حل مسئله را پیدا کنیم که تعیین کارایی یک الگوریتم با توجه به دو فاکتور زیر سنجیده میشود:

✓ زمان اجرای الگوریتم

✓ میزان حافظه مصرفی الگوریتم

🚦 سوال (۱) چرا به زمان صرف شده برای تولید الگوریتم و رفع اشکال آن و زمان تبدیل آن به برنامه توجه نشده است؟

معیار زمان اجرا وابسته به ماشینی است که الگوریتم بر روی آن اجرا می شود و معیار حافظه مصرفی نیز به روش مدیریت حافظه توسط سیستم عامل بستگی دارد. پس چنانچه بخواهیم دو الگوریتم را با همدیگر مقایسه نماییم باید شرایط کاملاً یکسانی برای آنها فراهم نماییم.

🚦 سوال (۲) اگر الگوریتمها توسط افراد متفاوت، در مکانهای متفاوت و با امکانات متفاوت نوشته شوند، چگونه می توان شرایط یکسانی فراهم کرد ؟

جهت فراهم نمودن شرایط یکسان، باید معیارهای فوق را به گونه ای تعریف نمود که مستقل از ماشین و سیستم عامل باشند. دو معیار زیر برای این منظور تعریف شده اند:

✓ مرتبه زمانی اجرای الگوریتم

✓ مرتبه مکانی اجرای الگوریتم

برای تشریح معیارهای فوق باید عاملی به نام اندازه مساله باید تعریف گردد:

اندازه مسئله:

تعداد داده های، تعداد نتایج و یا ترکیبی از آنهاست.

(مثال)

✓ مرتب سازی آرایه ای با n عنصر: اندازه ورودی n

✓ جستجوی عنصری در آرایه ای با n عنصر: اندازه ورودی n

✓ جمع دو ماتریس $n \times n$: اندازه ورودی n

✓ تعیین k عدد اول سری فیبوناچی: اندازه خروجی k

تعیین عمل اصلی در یک الگوریتم:

به دستور یا دستوراتی که کل کار انجام شده توسط الگوریتم، تقریباً متناسب با تعداد دفعاتی باشد که

توسط این دستور یا دستورات انجام می شود، عمل اصلی الگوریتم گویند.

مرتبه زمانی:

در اغلب مسائل تابعی از اندازه مساله می باشد و عبارت است از تعداد دفعاتی که عمل اصلی به ازای هر

مقدار از اندازه ورودی انجام می شود.

(مثال ۱) روش جستجوی ترتیبی (پیدا کردن y در آرایه x بطول n)

عمل اصلی: مقایسه

اندازه ورودی: n

تعداد دفعات انجام عمل اصلی: n

پس: $T(n) = n$

(مثال ۲) مرتب سازی آرایه x به طول n

```
For (i=1; i<=n-1; i++)
  For (j=1; j<=n-i; j++)
    If ( X[j]>x[j+1])
      Exchange X[j] and X[j+1]
```

عمل اصلی: مقایسه (if)

اندازه ورودی: n

تعداد اعمال مقایسه:

$$T(n) = (n-1) + (n-2) + (n-3) + \dots + 3 + 2 + 1 = \frac{n(n-1)}{2}$$

مثال ۳) ضرب ماتریسهای $Y_{K \times N}$ و $X_{M \times K}$

مثال ۴) فرض کنید دو الگوریتم با پیچیدگیهای متفاوت برای حل مساله ای داریم: n برای الگوریتم اول و n^2 برای الگوریتم دوم.

الف) کدام الگوریتم کارآمدتر است؟

ب) اگر کامپیوتر مفروضی برای پردازش عمل اصلی الگوریتم اول، ۱۰۰۰ برابر پردازش دستور اصلی الگوریتم دوم زمان لازم داشته باشد، کدام الگوریتم کارآمدتر است؟

حل) اگر t زمان لازم جهت یکبار اجرای دستورات اصلی الگوریتم دوم باشد، زمان لازم برای پردازش دستور اصلی الگوریتم دوم $1000t$ خواهد بود. بنابراین زمان لازم برای پردازش نمونه ای به اندازه n با الگوریتم نخست $n * 1000t$ و برای الگوریتم دوم $n^2 * t$ خواهد بود. الگوریتم اول زمانی بهتر خواهد بود که: $n^2 * t > n * 1000t$ در نتیجه به ازای $n > 1000$ الگوریتم اول کارایی بهتری خواهد داشت.

☑ پس الگوریتمی با پیچیدگی زمانی n از الگوریتمی با پیچیدگی زمانی n^2 به ازای مقادیر بزرگ n کارایی بیشتری دارد و این از زمان لازم جهت پردازش عملیات اصلی در دو الگوریتم مستقل می باشد.

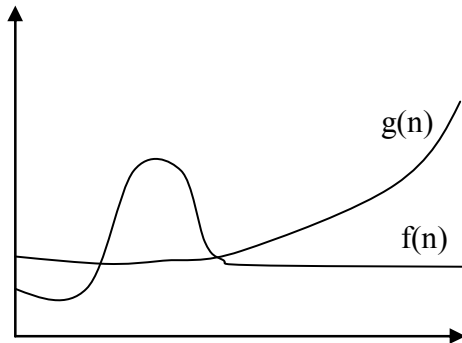
مثال ۵) مشابه مثال قبل: اگر پیچیدگی الگوریتم اول برابر $100n$ و پیچیدگی الگوریتم دوم برابر $0.01n^2$ باشد، باز هم به ازای $n > 10.000$ ، الگوریتم اول کارایی بهتری دارد. الگوریتمهایی با پیچیدگی زمانی n ، $100n$ ، $0.000n$ و امثال اینها را الگوریتمهای زمانی خطی گویند زیرا پیچیدگی زمانی آنها با اندازه ورودی رابطه خطی دارد.

توابعی نظیر n^2 ، $100n^2$ و $5n^2 + 10$ را توابع درجه دوم محض گویند و توابعی نظیر $5n^2 + 3n + 100$ را که در آنها علاوه بر درجه دو، درجه اول نیز وجود دارد، توابع درجه دوم کامل گویند.

معرفی Big O:

تعریف: می گوئیم تابع $f(n)$ از مرتبه $g(n)$ است (و بصورت $f(n) = O(g(n))$ نشان می دهیم)، اگر و فقط اگر ثابت حقیقی مثبت c و ثابت صحیح غیر منفی n_0 وجود داشته باشند که برای همه مقادیر $n \leq n_0$ انگاه $|f(n)| \leq c |g(n)|$.

در رابطه فوق $f(n)$ مبین تعداد اعمال انجام شده در زمان اجرای الگوریتم و $g(n)$ مرتبه زمانی اجرای الگوریتم نامیده می شود.



مثال ۶) درستی و یا عدم درستی روابط زیر را نشان دهید.

الف) $5n \in O(n^2)$

چون $5n \leq 5n^2$ پس به ازای $c=5, n_0=1$ می توان نتیجه گرفت که رابطه فوق درست می باشد.

ب) اگر $T(n) = \frac{n(n-1)}{2}$ انگاه $T(n) \in O(n^2)$

می توان نوشت: $\frac{n(n-1)}{2} \leq \frac{n(n)}{2} = \frac{1}{2}n^2$ پس $c = \frac{1}{2}$ و $n_0 = 0$

ج) $n^2 + 10n \in O(n^2)$

$$n^2 + 10n \leq n^2 + 10n^2 = 11n^2 \Rightarrow c = 11, n_0 = 1$$

د) $n \in O(n^2)$

$$c = 1, n_0 = 1$$

چنانچه پیچیدگی الگوریتمی به اندازه ورودی وابسته نباشد بلکه یک مقدار ثابت مستقل از اندازه ورودی باشد، در این حالت آنرا بصورت $O(1)$ نشان می دهیم.

مثال ۷) برای الگوریتم جستجوی خطی پیچیدگی همه حالات را بدست آورید.

1	2	3	.	.	.	.	.	N

بهترین حالت: $O(1)$

بدترین حالت: $O(n)$

حالت متوسط: $O((n+1)/2)$

مثال ۸) پیچیدگی الگوریتم زیر را تعیین کنید.

$x=0;$

$i=n;$

while ($i>1$)

تعداد دفعات تکرار حلقه $\log_2 n$

<pre>{ x--; i /= 2; }</pre>	$\rightarrow O(\log n)$
-------------------------------------	-------------------------

مثال ۹) پیچیدگی الگوریتم زیر را تعیین کنید.

<pre>x=0; i=n; while(i>1) { x--; i %= 2; }</pre>	a) $O(1)$ ✓ b) $O(n)$ c) $O(\log n)$ d) $O(n \log n)$
---	---

مثال ۱۰) با فرض $m=n$ پیچیدگی الگوریتم زیر را تعیین کنید:

<pre>For i:=1 to n For j:=1 to m x++;</pre>	$\rightarrow m*n \rightarrow O(n^2)$
---	--------------------------------------

مثال ۱۱) پیچیدگی الگوریتم زیر را تعیین کنید.

<pre>For i:=0 to n do For j:=1 to m do For k:=1 to j do x++;</pre>	$\sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^j (1) = \sum_{i=1}^n \sum_{j=1}^m (j) = \sum_{i=1}^n \frac{m(m+1)}{2} = \frac{mn(m+1)}{2} =$ $= \frac{n^2(n+1)}{2} = \frac{n^3 + n^2}{2}$ در چند جمله ای ها بالاترین توان مرتبه پیچیدگی است: 1) $O(n)$ 2) $O(n^2)$ $\rightarrow$ 3) $O(n^3)$ 4) $O(n^2 \log n)$
--	---

مثال ۱۲)

$$f(n) = 3 \log n + 10 n \log n + 7 n^2 \rightarrow O(n^2)$$

مثال ۱۳) در کدام گزینه توابع به ترتیب صعودی پیچیدگی مرتب شده اند؟

- | | |
|-----------------------------------|-----------------------------------|
| 1) n^{1000} , $n!$, $(1005)^n$ | 3) n^{1000} , $(1005)^n$, $n!$ |
| ✓ | |
| 2) $(1005)^n$, $n!$, n^{1000} | 4) $(1005)^n$, n^{1000} , $n!$ |

ترتیب اولویت ها:

$1, \log n, n, n \log n, n^2, n^3, \dots, n^k, 2^n, 3^n, \dots, k^n, n!$

مثال ۱۴) اگر زمان تست شرط برابر $\dagger$ باشد، پیچیدگی زمانی کد زیر چقدر است؟

if (test)	1) $t + s1 + s2$
s1	2) $s1 + s2$
else	3) $t + \min(s1+s2)$
s2	4) $t + \max(s1+s2)$ ✓

توضیح: همیشه باید بدترین حالت را در نظر گرفت.

مثال (۱۵)

if (test1)	1) $t1 + t2 + s1 + s2$
s1	2) $s1 + s2$
else if (test2)	3) $t1 + t2 + \min(s1+s2)$
s2	4) $t1 + t2 + \max(s1+s2)$ ✓

مثال (۱۶)

a=n; while(a>1) { a /= 2; b = n; while (b>1) { b /= 3; x++; } }	→ $\log_3 n$	→ $\log_2 n$	→ $O(\log_2 n * \log_3 n)$
---	--------------	--------------	----------------------------

مثال (۱۷)

for (k=0; k<=n-1 ; k++) for (i=0; i<=n-k ; i++) a[i][i+k] = k;	1) $n(n-1)/2$ 2) $n^2/2$ 3) $n(n+1)/2$ ✓ 4) n^2
--	--

راه حل:

$$\sum_{k=0}^{n-1} \sum_{i=1}^{n-k} (1) = \sum_{k=0}^{n-1} n - k = n \sum_{k=0}^{n-1} (1) - \sum_{k=0}^{n-1} (k) = n^2 - \frac{n(n-1)}{2} = \frac{2n^2}{2} - \frac{n^2}{2} + \frac{n}{2} = \frac{n^2}{2} + \frac{n}{2} = \frac{n(n+1)}{2}$$

مثال (۱۸)

k = ; for(i=1; i<=n ; i++) { for(j1; j<=n ; j++)	→ $O(m)$
---	----------

ساختمان داده ها

فصل اول	پیچیدگی الگوریتمها	Complexity	1) $n! = O(n^n)$
			2) $2n^2n^n + n^2 \log n = O(n^2 2^n)$
			3) $\sum_{i=1}^n i^2 = O(n^3)$
			4) $n^2 \log n = O(n^3)$

<pre> k=k+1; j = 1; while (j<n) { k++; j=2*j; } </pre>	→ $O(\log n)$	→ $O(m + \log n)$	→ $O(n(m + \log n))$
---	-------------------------------------	---	--

تمرین) از عبارات زیر کدامیک درست و کدامیک نادرست است؟

5) $n^2 \log n = O(n^2)$	
6) $n^2 / \log n = O(n)$	
7) $n^2 / \log n = O(n^2)$	
8) $n^2 = O(n^3)$	

مثال ۱۹) پیچیدگی الگوریتم زیر را بدست آورید:

```
S=0;
for(i=0 ; i<n; i++)
    for (j=0; j<i; j++)
```

$O(n^2)$

```
        for(k=0 ; k<3
;k++)
            S++;
```

چون ثابت است ، تاثیری در پیچیدگی ندارد

$O(n \log n)$ (۴)

$O(n^3)$ (۳)

$\rightarrow O(n^2)$ (۲)

$o(n)$ (۱)

مثال ۲۰)

```
S=0;
for(i=1 ; i<=n; i++)
    for (j=i+1 ; j<=n; j++)
        S++;
```

$$\sum_{i=1}^n \sum_{j=i+1}^n (1) = \sum_{i=1}^n n - i - 1 + 1 = \sum_{i=1}^n (n - i) = n^2 - \frac{n(n+1)}{2} \rightarrow O(n^2)$$

مثال ۲۱) کدامیک از عبارات زیر درست و کدامیک غلط است:

1) $n \log n = O(n \sqrt{n})$

$\sqrt{n} > \log n$ [true]

- | | |
|--|-------------------------------------|
| 2) $\sqrt{n} = O(\log n)$ | [false] |
| 3) $\log n = O(\sqrt{n})$ | [true] |
| 4) $n^2(1 + \sqrt{n}) = O(n^2 \log n)$ | $n^2 \sqrt{n} > n^2 \log n$ [false] |
| 5) $1/n = O(\sqrt{n})$ | [true] |
| 6) $1/n = O(\log n)$ | [true] |

توابع بازگشتی:

هر تابعی که بصورت مستقیم یا غیر مستقیم خودش را فراخوانی کند ، تابع بازگشتی نامیده میشود.

خواص:

- ✓ باید معیاری به عنوان معیار پایه (Base Case) وجود داشته باشد که تابع در این حالت خودش را فراخوانی نماید.
- ✓ در هر بار فراخوانی ، به معیار پایه نزدیکتر شود.

مثال (۱) تابع فاکتوریل:

$$n! = 1 * 2 * 3 * \dots * n$$

$$(n-1)! = 1 * 2 * \dots * (n-1)$$

$$n! + n * (n-1)!$$

$$f(n) = n! = 1 * \dots * n$$

$$f(n) = n * f(n-1)$$

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

```
int f(int n)
{
    if(n==0)
        return(1);
    else
        return(n*f(n-1));
}
```

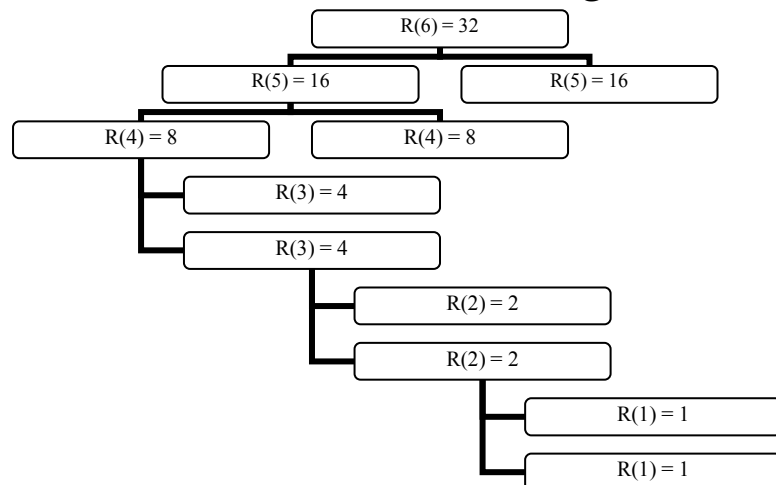
تابع نوشته شده به زبان سی //

$$f(5) = 5 * f(4) = 5 * 4 * f(3) = 5 * 4 * 3 * f(2) = 5 * 4 * 3 * 2 * f(1) = 5 * 4 * 3 * 2 * 1 * f(0) = 120$$

مثال (۲) مقدار تابع زیر چند میشود:

```
int Rec(int n)
{
    if (n==1)
        return(1)
    else
        return(Rec(n-1) + Rec(n-1))
}
```

$$Rec(6) = ? \rightarrow 32$$



مثال ۳) اگر $n=8$ باشد، چند عمل ضرب در تابع زیر انجام میشود. (هر عمل square یک ضرب دارد)

```
function count(n)
begin
  if n<=0 then return(1)
  if n=1 then return(2)
  if n=2 then return(3)
  return (count (n-2) * square (count (n-4)));
end
```

1) 4 2) 8 3) 9 4) 10

$c(8)$
↓
 $c(6) * sq(c(4)) : 8$
↓
 $c(4) * sq(c(2)) : 4$
↓
 $c(2) * sq(c(0)) : 2$

مثال ۴) مقدار $R(5,3)=?$

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

1) 15 2) 25 3) 75 4) 125

$R(5,3) : 125$
↓
 $R(5,1) : 5$
↓
 $R(5,0) : 1$

مثال ۵) تابع آکرمان:

$$Ack(x, y) = \begin{cases} y+1 & x=0 \\ Ack(x-1, 1) & y=0 \\ Ack(x-1, Ack(x, y-1)) & \text{else} \end{cases}$$

$Ack(1,1)=?$

$Ack(1,1) : 3$
↓
 $Ack(0, Ack(1,0)) : 3$
↓
 $Ack(0,1) : 2$

مثال ۶)

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

بازگشتی Recursive

$$f(m,n) = f(n-1,m) + f(n-1,m-1)$$

$$f(6,4)=?$$

مثال (۷)

```
Function w(n:word) : real;
begin
  if n=1 then
    w:=sqrt(2);
  else
    w:=sqrt(w(n-1)+2);
end;
```

$$n = 1384 \quad f(n) = ?$$

$$1)3 \quad 2)6 \quad \rightarrow 3)2 \quad 4)4$$

$$w = \sqrt{2 + \underbrace{\sqrt{2 + \sqrt{2 + \sqrt{2 + \dots}}}}_{1384}}$$

$$w^2 = \sqrt{2} + \underbrace{\sqrt{2 + \sqrt{2 + \sqrt{2 + \sqrt{2 + \dots}}}}}_{1383}$$

$$w^2 = 2 + w \rightarrow w^2 - w - 2 = 0 \rightarrow w = 2$$

مثال (۸)

```
void xyz(void)
{
  char ch;
  ch = getche();
  if (ch != '\n')
    xyz();
  putchar(ch);
}
```

input: ABC [enter]

output : CBA

مثال (۹) محاسبه مجموع اعداد صفر تا n:

$$S(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 1 \end{cases}$$

مثال (۱۰) محاسبه a^b :

$$P(a,b) = \begin{cases} 1 & b = 0 \\ a * p(a,b-1) & b \geq 1 \end{cases}$$

مثال (۱۱) محاسبه $a*b$:

$$m(a,b) = \begin{cases} a & b = 1 \\ a + m(a,b-1) & b > 1 \end{cases}$$

مثال (۱۲) محاسبه a/b :

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

مثال (۱۳) محاسبه $b \% a$:

$$M(a,b) = \begin{cases} a & a < b \\ M(a-b,b) & a \geq b \end{cases}$$

مثال (۱۴) $k(4,2)=?$:

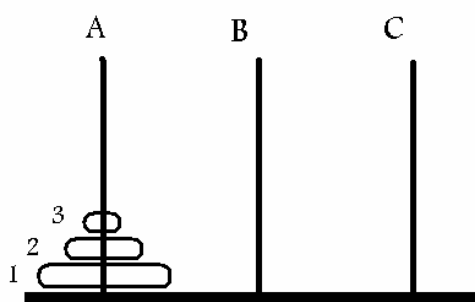
```
k(n,m)
{
    if (m==n) || (m==0)
        return(1)
    else
        return(k(n-1,m)+k(n-1,m-1))
}
```

$$\binom{m}{n} = \binom{4}{2} = \frac{4!}{2! \cdot 2!} = \frac{4 \cdot 3}{2} = 6$$

کوئیز ۱:

```
Algorithm FUN (n)
begin
    if(n>0) then
        if(n=1) then
            put "X"
        else if (n=2) then
            put "Y"
        else
            put "A"
            FUN (n-1)
            put "B"
            FUN (n-2)
            put "C"
        end if
    end
end
```

n = 3 : AYBXC
n = 4 : AAYBXCBYC
n = 5



برج هانوی (۱۸۸۳ لوکاس):

۱- حلقه بزرگتر روی حلقه کوچکتر قرار نگیرد.

۲- هر مرحله فقط یک حرکت انجام شود.

$$(2, C, B) \rightarrow \begin{cases} \overset{A}{3} \rightarrow C \\ \overset{A}{2} \rightarrow B \\ \overset{C}{3} \rightarrow B \\ \overset{A}{1} \rightarrow C \perp \\ \overset{B}{3} \rightarrow A \\ \overset{B}{2} \rightarrow C \perp \\ \overset{A}{3} \rightarrow C \perp \end{cases}$$

$$n=3 \rightarrow f(n)=7$$

۷ حرکت

$$n=4 \rightarrow f(n)=7+1+7=15$$

۱۵ حرکت

$$f(n) = 2f(n-1) + 1 \quad (n=1 \quad f(n)=1)$$

$$f(n) = 2^n - 1$$

راه حل بازگشتی:

(۱) $n-1$ حلقه را به کمک C از A به B منتقل می کنیم.

(۲) حلقه n ام را از A به C منتقل می کنیم.

(۳) حلقه باقیمانده را از B به C منتقل می کنیم.

$$f(n, A, C, B) \Rightarrow f(\text{کمکی, مقصد, مبدا, تعداد حلقه})$$

{
 $f(n-1, A, B, C)$
 انتقال دیسک آخر از مکان مبدا به مکان مقصد...
 $f(n-1, B, C, A)$
 }

رابطه بازگشتی مسأله برجهای هانوی:

$$f(n) = \begin{cases} 2f(n-1)+1 & n > 1 \\ 1 & n = 1 \end{cases}$$

حل رابطه بازگشتی هانوی:

$$\begin{aligned} f(n) &= 2f(n-1)+1 \\ &= 2(2f(n-2)+1)+1 \\ &= 2^2f(n-2)+1+2 \\ &= 2^3f(n-3)+1+2+4 \\ &\dots \\ &= 2^{n-1}f(1) + 1+2+2^2+\dots+2^{n-2} \\ &= 2^{n-1} + (2^{n-1}-1) = 2^n-1 \end{aligned}$$

$$1+2+2^2+\dots+2^n =$$

$$\sum_{i=0}^n 2^i = 2^{n+1} - 1$$

=>

$$O(2^n)$$

مثال)

$$f(n) = \begin{cases} 1 & n = 2 \\ 2f(n-2) & n > 2 \end{cases}$$

$$1) O(n) \quad 2) O(2^n) \quad 3) O(2^{n/2}) \quad \checkmark \quad 4) O(n \log n)$$

آرایه:

- ✓ ساختمان داده ای جهت ذخیره سازی عناصر همگن (همگن)
- ✓ عناصر آرایه بکمک یک اندیس قابل دسترس می باشند (چون پشت سرهم در حافظه ذخیره می شوند).

آرایه یک بعدی (لیست-بردار):

$X: \text{Array}[L..U] \text{ of } \langle \text{type} \rangle$

$w = \text{sizeof}(\text{type})$

تعداد عناصر = $U - L + 1$

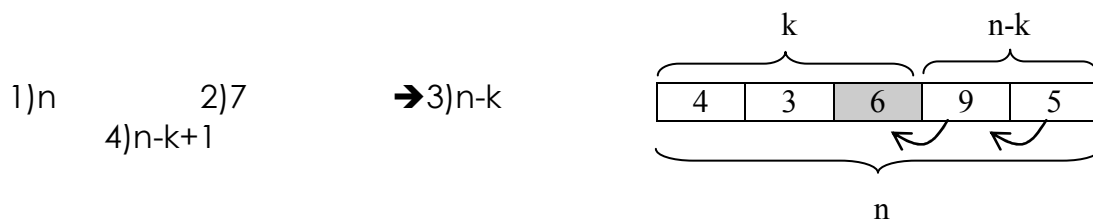
حافظه = $(U - L + 1) * w$

• جستجو:

- ترتیبی:
- بهترین حالت: $O(1)$
- بدترین حالت: $O(n)$
- دودویی:
- بهترین حالت: $O(1)$
- بدترین حالت: $O(\log(n))$ (مقدار دقیق: $\lceil \log(n) \rceil + 1$)

• مرتب سازی

مثال ۱) برای حذف عنصر k ام آرایه ای بطول n به چند عمل جابجایی نیاز داریم:



مثال ۲) اگر طول آرایه ۲۰۰ باشد، حداکثر اعمال مقایسه برای پیدا کردن یک عنصر به روش دودویی چقدر است؟

- 1) 200 2) 7 3) 8 4) 9

$$\lceil \log 200 \rceil + 1 = \lceil 7.64 \rceil + 1 = 8$$

مثال ۳) در صورتیکه آرایه مورد جستجو در روش جستجوی دودویی بصورت زیر باشد، متوسط تعداد مقایسه ها برای جستجوی موفق چقدر است؟

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

1) 27/9 → 2) 25/9 3) 31/9
4) 29/9

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

توضیح: تعداد همه مقایسه ها تقسیم بر تعداد کل عناصر

تابع جستجوی دودویی بصورت بازگشتی:

```
int BS(int x[], int a, int b, int item)
{
    int mid;
    mid = (a+b) / 2;
    if (a <= b)
    {
        if (x[mid] > item)
            return(BS(x, a, mid-1, item));
        elseif (x[mid] < item);
            return(BS(x, mid+1, b, item));
        else
            return (mid);
    }
    else
        return(-1);
}
```

$$T(n) = \begin{cases} 1 & n = 0 \\ T(n/2) + C & n \geq 1 \end{cases}$$

$$n = 2^k \rightarrow k = \log n$$

$$T(n) = T(n/2) + c \\ = T(n/2^2) + c + c$$

$$\dots \\ = T(n/2^k) + k * c \\ = T(0) + (k+1) * c$$

$$\rightarrow T(n) = 1 + (k+1) * c = 1 + (\log n + 1) * c$$

$$= 1 + c * \log n + c = c * \log n + d$$

$$\rightarrow O(\log n)$$

تمرین) روش جستجوی خطی را بصورت بازگشتی بنویسید و پیچیدگی آنرا بدست آورید.

```
linear (a[], int a, int b, int item)
{
    if(b>=a)
    {
        if(x[a]==item)
            return(a);
        else
            return(linear(x,a+1,b,item));
    }
    else
        return(-1);
}
```

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + 1 & n \geq 1 \end{cases}$$

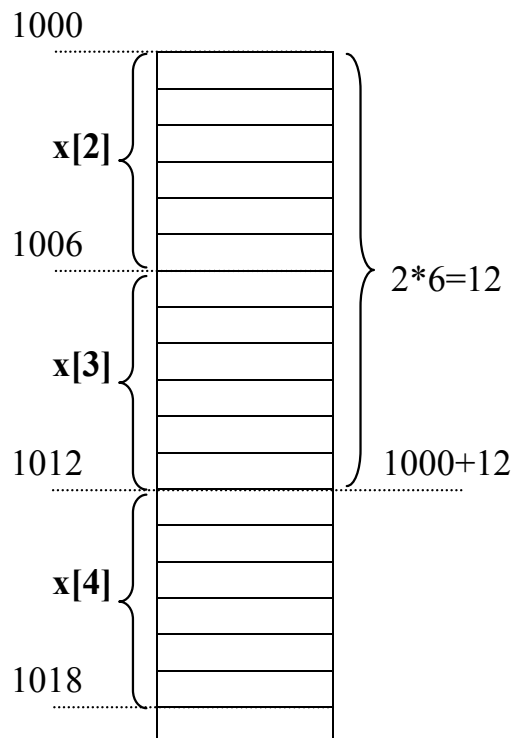
$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 2 \\ &\dots \\ &= T(n-n) + n \\ &= 0 + n \end{aligned}$$

$$\rightarrow O(n)$$

نحوه ذخیره سازی آرایه ها در حافظه:

X: [2..8] of real
w=6
size = 42 byte

$$\text{Add}(X[i]) = \text{Base} + (i-L)*w$$



(مثال) اگر آرایه X دارای ۱۰ عنصر باشد که هر عنصر آن یک رشته سه کاراکتری است و آرایه از موقعیت ۱۰۰ حافظه ذخیره شده باشد، موقعیت شروع عنصر ششم کدام است؟

- 1) 118 2) 112 **→ 3) 115** 4) 121

$$5 \times 3 = 15$$

$$\text{add} = 100 + 15 = 115$$

آرایه دو بعدی:

$X: \text{Array}[L_1..U_1, L_2..U_2]$ of type

تعداد سطرها: $U_1 - L_1 + 1 = r_1$

تعداد ستون ها: $U_2 - L_2 + 1 = r_2$

تعداد عناصر = $r_1 \times r_2$

میزان حافظه = $r_1 \times r_2 \times w$

				R.M		C.M
X	3	4	5			
2	1	2	3	R2 {	1	C3 {
3	4	5	6	R3 {	2	4
4	7	8	9	R4 {	3	7
					4	2
					5	5
					6	8
					7	3
					8	6
					9	9

نحوه ذخیره سازی در حافظه:

Row Major - روش سطر - (C, Pascal, ...)

Column Major - روش ستونی - (Cobol, Fortran, ...)

(مثال)

$X: \text{Array}[2..4, 3..5]$ of word;

$\text{Add}(x[2,3]) = \text{Base} = 100$

$W=2$

$\text{Add}(x[4,4])=?$

سطری: $14 = 2 \times 7 \leftarrow$ آدرس: $14 + 100 = 114$

114

ستونی: $10 = 2 \times 5 \leftarrow$ آدرس: $10 + 100 = 110$

110

بدست آوردن تعداد عناصر قبل از عنصر $x[i,j]$ در روش سطر:

عناصر سطرهای بالایی: $(i-L_1) \cdot (U_2-L_2+1)$

عناصر سطر i ام که قبل از عنصر $x[i,j]$ است: $j-L_2$

آدرس خانه اول = Base

آدرس $x[i,j] = ?$

تعداد کل عناصر: $\text{Num} = (i-L_1)(U_2-L_2+1) + j-L_2$

$\text{Add}(X[i,j]) = \text{Base} + [(i-L_1)(U_2-L_2+1) + j-L_2] \times w$

روش ستونی:

$\text{Add}(X[i,j]) = \text{Base} + [(j-L_2)(U_1-L_1+1) + i-L_1] \times w = \text{Add}(x[i,j])$

تمرین) فرمول های سطری و ستونی ۳ بعدی را بدست آورید.

$X: [L1..U1, L2..U2, L3..U3]$ of $\langle type \rangle$

روش سطری:

$$Add(X[i,j,k]) = Base + [(i-L1)*r2 + j - L2]*r3 + k - L3]*w$$

روش ستونی:

$$Add(X[i,j,k]) = Base + [(k-L3)*r2 + j-L2)*r1 + i - L1]*w$$

$X: Array[L1..U1, L2..U2, L3..U3]$

سطر $U_i - L_i + 1 = R_i$

سه بعدی سطری: $((i-L1)R2+(j-L2))R3 + k - L3$ $= (i-L1)R2R3+(j-L2)R3 + k - L3$ چهار بعدی سطری: $(i-L1)R2R3R4+(j-L2)R3R4 + (k - L3)R4 + l-L4$	سه بعدی سطری:
سه بعدی ستونی: $(k-L3)R2R1+(j-L2)R1 + i - L1$ چهار بعدی ستونی: $(l-L4)R3R2R1+(k-L3)R2R1 + (j-L2)R1 + i - L1$	سه بعدی ستونی:

مثال) اگر تخصیص حافظه به آرایه بصورت سطری باشد ، و دو آرایه $X[9,2,4]$ و $Y[24]$ از یک

نوع بوده و در حافظه روی هم تعریف شوند ، خانه $X(2,1,3)$ معادل کدام خانه Y می باشد؟

1) $Y[10]$ 2) $Y[11]$ 3) $Y[15]$ 4) $Y[20]$

$$(i-L1)R2R3+(J-L2)R3+k-L3$$

$$(2-1)*2*4+(1-1)4+3-1 = 8+2 = 10$$

مثال ۲) اگر $Base=20,000$ و $M[30*20*10]$ که بصورت ستونی ذخیره شده باشد و $w=4$ آنگاه آدرس $X[21,11,9]$ چند میشود؟

- 1) 20480 2) 40480 3) 40840 4) 20840

$$num = (k-3)R_2R_1 + (j-L_2)R_1 + i - L_1 = 5120$$

$$Add = 20,000 + 4*5120 = 40480$$

ماتریس های خلوت (اسپارس):

اگر در یک ماتریس، تعداد صفرها از تعداد یکها خیلی بیشتر باشد به آن ماتریس اسپارس گویند. روشهای ذخیره سازی ماتریسهای اسپارس:

۱- نامنظم

✓ هر عنصر بصورت یک سه تایی $(i, j, value)$ ذخیره میشود.

۲- منظم: مختص ماتریسهایی است که دارای شکل منظمی می باشند.

✓ قطری

✓ سه قطری

✓ پایین مثلثی

✓ بالا مثلثی

✓ متقارن

روش غیر منظم:

X:					عناصر	ستون	سطر
					غیر صفر		
	0	0	4	0	3	4	4
	0	1	0	5	1	3	4
	0	0	3	0	2	2	1
					2	4	5

🌈 اگر k تعداد عناصر غیر صفر باشد، نیاز به $3*(k+1)$ عنصر جهت ذخیره ماتریس اصلی داریم. در مثال فوق اگر $w=2$ (اندازه هر عنصر) آنگاه ماتریس اصلی ۲۴ بایت و ماتریس اسپارس ۳۰ بایت حافظه اشغال می کند. یعنی حافظه بیشتر بخاطر این است که تعداد عناصر غیر صفر از تعداد صفرها خیلی کمتر نیست.

بنابراین باید بینیم که چه زمانی ذخیره اسپارس به اینصورت بصرفه تر است.

مثال) ذخیره سازی اسپارس یک ماتریس قطری به ازای چه اندازه ای ($n=?$) بصرفه است؟
 $k=n \rightarrow n*n > (n+1)*3 \rightarrow n \geq 4$

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

مثال ۲) n مناسب را برای ماتریس سه قطری بدست آورید:

$$n^2 < [3n-2+1] * 3 \rightarrow n^2 - 9n + 3 > 0 \rightarrow n \geq 9$$

برخی عملیاتی که می توان روی اسپارس انجام داد:

✓ ترانهاد

✓ جمع

✓ ضرب

مثال ۱) بدست آوردن ترانهاد یک ماتریس اسپارس:

- $A[1,1] \rightarrow AT[1,2]$
- $A[1,2] \rightarrow AT[1,1]$
- $A[1,3] \rightarrow AT[1,3]$

```
L:=2;
for i:=1 to m do
  for j:=2 to k do
    if i=A[j,2] then
      begin
        AT[L,3] := A[j,3];
        AT[L,2] := A[j,1];
        AT[L,1] := A[j,2];
        L:=L+1;
      end;
```

مثال ۲) جمع دو ماتریس اسپارس:

- $X_{index1} > X_{index2} \Rightarrow K2$
- $X_{index1} < X_{index2} \Rightarrow K1$
- $X_{index1} = X_{index2} \Rightarrow Y_{index1} > Y_{index2} \Rightarrow K2$
- $Y_{index1} < Y_{index2} \Rightarrow K1$

$$Y_{index1} = Y_{index2} \Rightarrow K1 + K2$$

کوئیز:

در تابع زیر اگر $n=2^k$ آنگاه پیچیدگی را بدست آورید $O(?)$

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + 6n - 1 & n \geq 2 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + 6n - 1 \\ &= 2[2T(\frac{n}{4}) + 6 \cdot \frac{n}{2} - 1] + 6n - 1 \\ &= 2^2 T(\frac{n}{2^2}) + (6n - 2) + (6n - 1) \\ &= 2^3 T(\frac{n}{2^3}) + 6 \cdot \frac{n}{4} - 1 + (6n - 2) + (6n - 1) \\ &= 2^k T(\frac{n}{2^k}) + 6kn - (2^0 + 2^1 + 2^2 + \dots + 2^{k-1}) \\ &= nT(1) + 6n \log n - 2^k + 1 \\ &= 6n \log n + 1 \end{aligned}$$

→ $O(n \log n)$

ادامه مبحث اسپارس:

$$i \begin{bmatrix} \ddots & \ddots & 0 & 0 \\ \ddots & \ddots & x & 0 \\ 0 & \ddots & \ddots & \ddots \\ 0 & 0 & \ddots & \ddots \end{bmatrix} j$$

تعداد عناصر غیر صفر در ماتریس سه قطری:

سطر اول: ۲ سطر آخر: ۲

بقیه: ۳

تعداد عناصر: $4 + (n-2) \cdot 3$

عناصر سطرهای یک تا $i-1$: $2 + (i-1) \cdot 3$

سطر i ام و قبل از عنصر: $j-i+1$

تعداد کل: $2i+j-3$

آدرس عنصر: $2i+j-2$

مثال: اگر عنصر $x[1,1]$ در خانه اول آرایه y ذخیره شده باشد، آدرس عنصر زیر را بدست آورید.
(نوع عناصر دو آرایه یکسان است و x یک ماتریس سه قطری است)

X:Array[1..60, 1..60] of word;

1) $x[10,10] \rightarrow L=28$

2) $x[31,30] \rightarrow L=90$

3) $x[40,42] \rightarrow L=120 \rightarrow$ در آرایه ذخیره نمی شود $|i-j| < 1$

تمرین: ویژگیهای مربوط به تعداد عناصر قبل از عنصر i و آدرس عنصر دلخواه را برای ماتریس های اسپارس زیر بدست آورید:

(۱) بالامثلثی:

(۲) ۵ قطری:

(۳) k قطری:

مرتب سازی:

Bubble:

```
for i:=1 to n do
  for j:=1 to n-i do
    if  $x[j+1] < x[j]$ 
      interchange...
```

تعداد جابجایی:

تعداد مقایسه:

-بهترین: $O(1)$

-بدترین: $O(n^2)$

-بهترین: $\frac{n(n-1)}{2} = O(n^2)$

-بدترین: $O(n^2)$

Optimized Bubble!

```
f:=true;
i:=1
while (i<n) and f do
  begin
    f:=false;
    for j:=1 to n-i do
      if  $x[j] > x[j+1]$  then
        begin
          f:=true;
          Exchange;
        end
    end
  end
```

تعداد مقایسه:	تعداد جابجایی:
بهترین: $O(n)$	بهترین: $O(1)$
بدترین: $O(n^2)$	بدترین: $O(n^2)$

Selection:

```
for(i=0; i<size; i++)
{
    min=i;
    for(j=1; j<size; j++)
        if (arr[j]<arr[min])
            min=j;
    inchange(arr[i],arr[min]);
}
```

تعداد مقایسه:	تعداد جابجایی:
بهترین: $O(n^2)$	بهترین: $O(1)$
بدترین: $O(n^2)$	بدترین: $O(n)$

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

نکته: در روش انتخابی اگر آرایه به ترتیب عکس باشد، با $n/2$ جابجایی آرایه مرتب میشود

Insertion:

```
for(i=0; i<size; i++)
    for(j:=1; i<size; i++)
    {
        index=arr[i];
        j=i;
        while((j>0) && (arr[j-1]>index))
        {
            arr[j]=arr[j-1];
            j--;
        }
        arr[j]=index;
    }
```

تعداد مقایسه:	تعداد جابجایی:
بهترین: $O(n)$	بهترین: $O(1)$
بدترین: $O(n^2)$	بدترین: $O(n^2)$

نکته:

- ۱- روش درجی برای تعداد داده کم بهترین کارایی را دارد
- ۲- ترتیب عناصر هم کلید را عوض نمی کند.

سؤال) آیا میتوان در طی یکی از فازهای میانی Selection مرتب شدن آرایه را تشخیص داد؟

```
void SeletionSord(int arr[], int size)
{
    int i=0, j, min, temp, flag;
    do{
        min=i;
        flag=0;
        for(j=i+1 ; j<size; j++)
        {
            if(arr[j]<arr[min])
            {
                min=j;
                flag=1;
            }
            else if(arr[j-1]>arr[j])
                flag=1;
        }
        exchange(arr[i], arr[min]);
    }while(flag);
}
```

مرتب سازی سریع Quick:

۱- ابتدا یک عنصر به عنوان عنصر محور انتخاب میشود (pivot)

۲- کل عناصر آرایه به دو بخش تقسیم میشود

۱- عناصر کوچکتر یا مساوی محور به سمت چپ منتقل میشود

۲- عناصر بزرگتر از محور به سمت راست منتقل میشود

۳- همین روش به شیوه بازگشتی برای هر یک از دو نیمه راست و چپ انجام میشود.

50	40	60	30	45	70	65	55
pivot		L→		←R			

50	40	45	30	60	70	65	55
		L		R			

50	40	45	30	60	70	65	55
			←R	L→			

30	40	45	50	60	70	65	55
			pivot				

```
int partition(int arr[], int lx, int rx)
```

```
{
    int pivot = lx;

    while (lx < rx)
    {
        while (arr[lx] <= arr[pivot])
            lx++;
        while (arr[rx] > arr[pivot])
            rx--;
        if (lx < rx)
            exchange(&arr[lx], &arr[rx]);
        exchange(arr[rx], arr[pivot]);
    }
    return (rx);
}
```

$O(n)$

```
void quickSort(int arr[], int lx, int rx)
```

```
{
    if (lx < rx)
    {
        int pivot = partition(arr, lx, rx);
        quickSort(arr, lx, pivot - 1);
        quickSort(arr, pivot + 1, rx);
    }
}
```

بهترین حالت:

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + n & n > 1 \end{cases} \rightarrow$$

$O(n \log n)$

بدترین حالت:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n > 1 \end{cases} \rightarrow O(n^2)$$

نکته: اگر آرایه مرتب شده باشد، بدترین حالت برای QuickSort است.

اما بهترین حالت زمانی است که عنصر pivot بعد از جابجایی دقیقاً وسط قرار گیرد.

سؤال ۱) اگر بعد از تابع Partition بجای تابع QuickSort استفاده کنید، چه

وضعیتی پیش می آید؟ (برای اینکه آرایه بصورت صعودی مرتب شود)

۱) همواره از Quick بهتر است

۲) اگر آرایه بصورت صعودی مرتب شده باشد از Quick بهتر است

۳) اگر آرایه بصورت نزولی مرتب شده باشد از Quick بهتر است

۴) ...

سؤال ۲) آرایه ای داریم بطول $1..N$ و همچنین داریم $1 \leq k, L \leq N$ که $K+L \leq N$

با در نظر گرفتن موارد مقابل بگویید که L, K چه ویژگی داشته باشد تا این سه دستور همیشه آرایه را مرتب کنند.

Sort(X, K+1, N)

Sort(X, 1, L+K)

Sort(X, K+1, N)

1) $k \leq L$ → 2) $k = L$ 3) $n = 3k = 3L$ 4) $L > k$

سؤال ۳) آرایه ای داریم بطور $1..N$ آرایه از یک تا D تا $D \gg N/2$ مرتب شده است. کدام روش مرتب سازی بهترین کارایی را دارد.

1) Bubble → 2) Insertion 3) Selection 4) Quick

سؤال ۴) در کدامیک از روشهای زیر، پیچیدگی در حالت متوسط و بدترین حالت متفاوت است؟

1) Bubble (n^2) 2) Insertion (n^2) 3) Selection (n^2)
→ 4) Quick ($n \log n$)

سؤال ۵) الگوریتم ضرب دو ماتریس را در نظر بگیرید:

```
for i:=1 to n do
  for j:=1 to k do
    begin
      s:=0;
      for L:=1 to m do
        s += x[i][L] * y[L][j];
      z[i][j]=s;
    end;
```

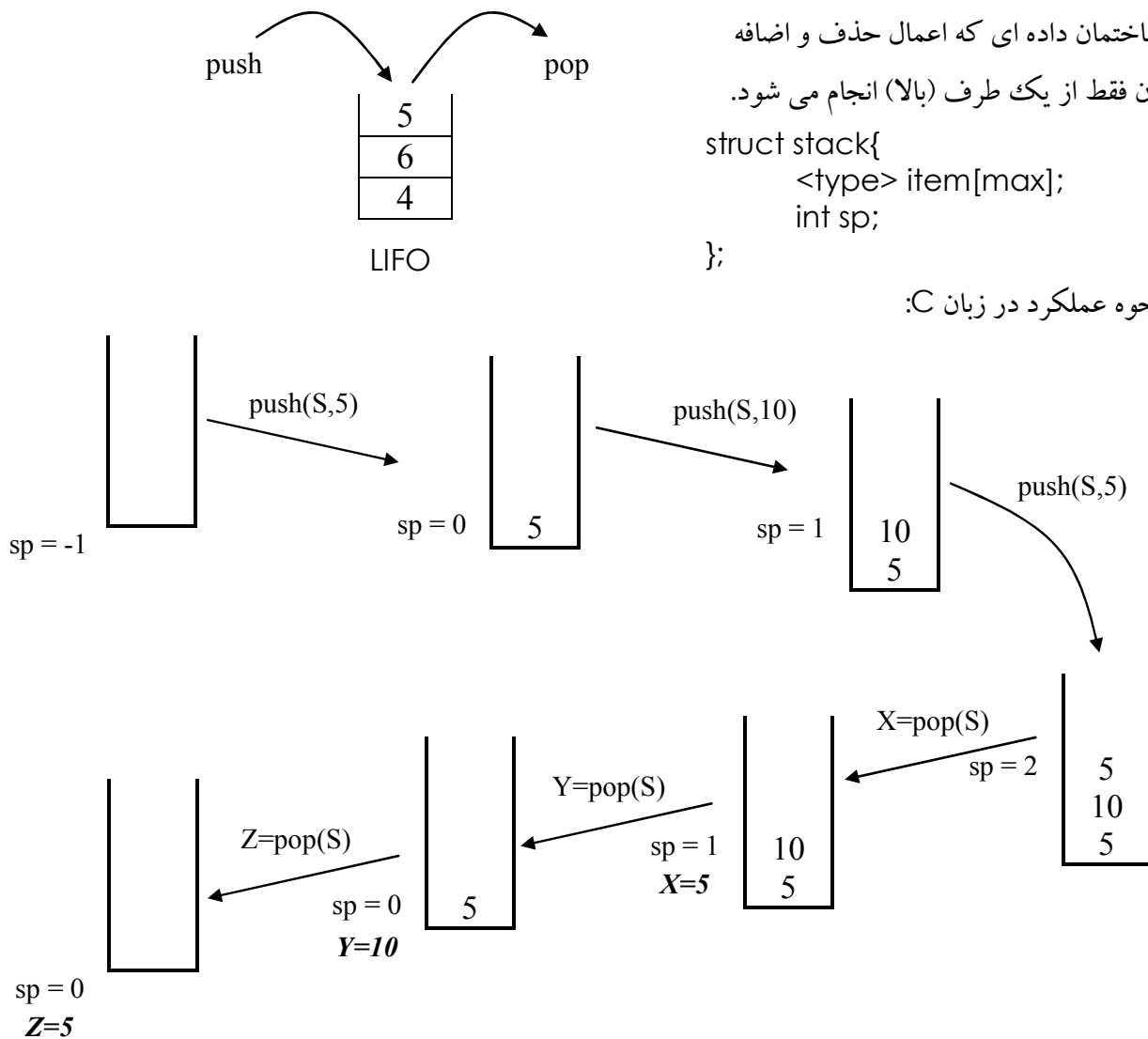
پیچیدگی این الگوریتم از مرتبه $O(n^3)$ است. روشی بنام starassen وجود دارد که پیچیدگی آن از مرتبه $O(n^{2.8})$ است. آنرا تحقیق کنید...

پشته Stack: (Last in First Out)

ساختمان داده ای که اعمال حذف و اضافه آن فقط از یک طرف (بالا) انجام می شود.

```
struct stack{
    <type> item[max];
    int sp;
};
```

نحوه عملکرد در زبان C:



مثال ۱) پشته S عناصر از نوع int است. در بازه [1,5] کدامیک از خروجیهای زیر را نمی توان با این

اعمال تولید کرد:

- ۱) 5,4,3,2,1
۲) 1,2,3,4,5
۳) 1,2,4,3,5
۴) 4,3,5,1,2 →

مثال ۲) کدامیک را نمی توان تولید کرد: (push کوچکترین مقدار ورودی را به پشته منتقل میکند)

- ۱) 2,1,5,3,4,6 →
۲) 1,2,3,5,6,4
۳) 3,2,4,6,5,1
۴) 4,3,2,1,6,3

مثال ۳) کم هزینه ترین روش از نظر حافظه مصرفی برای اینکه عناصر یک پشته S1 را به پشته S2

منتقل کنیم چگونه ای که ترتیب عناصر حفظ شود کدام است؟

(۱) یک متغیر کمکی (۲) چند متغیر اضافی (۳) دو پشته اضافی (۴) یک پشته اضافی
→

مثال ۴) اگر n عملی از اعمال مشخص شده با ترتیب دلخواه بر روی پشته S (که در ابتدا خالی است) باشد مجموع هزینه در بدترین حالت کدام است؟

$push(s,x)$: اضافه کردن x به پشته s با هزینه $O(1)$	(۲)	$O(n) \rightarrow (۱)$
$pop(s,x)$: حذف عنصر بالای پشته با هزینه $O(1)$	(۴)	$O(nk)$
$mpop(S,k)$: حذف k عنصر بالای پشته با هزینه $O(k)$		$O(\log n)$ (۳)
		$O(n \log n)$

$$(n-1)O(1) + O(n-1) \rightarrow O(n)$$

$$n/2O(1) + n/2O(1) \rightarrow O(n)$$

عبارات ریاضی:

چک توازن عبارات از نظر پرانتز:

- در حالت معمولی شمارنده صفر و در حین اجرا منفی نشود
 - در روش پشته باید در انتها پشته خالی شود
- برای تطبیق پرانتز و کروشه و آکولادهای عبارت زیر ، به یک پشته با حداقل چند عنصر نیازمندیم؟

$\{a-(b+[x-y]*z+[(m-n)])\}$

جواب: ۴ عنصر

Type	پیاده سازی پشته:
<code>stack=RECORD</code>	- آرایه
<code>item:Array[1..MAX];</code>	- لیست پیوندی
<code>sp:0..MAX;</code>	
<code>end;</code>	

مثال ۱) روالی بنویسید که پشته خالی را ایجاد نماید:

```

Procedure CreateStack(Var s:stack);
Begin
    s.sp := 0;
end;
```

مثال ۲) تشخیص خالی بودن پشته:

```

Procedure isFull(s:stack):Boolean;
Begin
    IF s.sp=MAX then return true
```



```
else return false;
end;
```

مثال ۳) افزودن به پشته:

```
Procedure push(Var s:stack; x:integer);
Begin
  if isFull(s) then
    begin
      // پیغام خطا
    end
  INC (s.sp);
  s.item[s.sp] := x;
end;
```

مثال ۴) خواندن از پشته

```
POP →
  اگر پشته خالی نباشد
  {
    x=s.item[s.sp]
    DEC (s.sp)
  }
```

مثال ۵) تابعی که عنصر بالای پشته را برمیگرداند (اما حذف نمی کند)

```
TOP →
  اگر پشته خالی نباشد
  {
    return (s.item[s.sp])
  }
```

نماد گذاری لهستانی:

in → post
in: A(B+C)/D

پرانتز گذاری:

[(A* (B+C)) / D]

انتقال علامت های داخل هر پرانتز به مقابل آن:

[((A * (B + C)) / D]

ABC+*D/

اولویت های محاسباتی یکسان از چپ به راست انجام میشوند بجز توان و انتساب:

2^3^4

x=y=z=2

infix: عملگر بین دو

عملوند

postfix: عملگر بعد از

دو عملوند

prefix: عملگر قبل از

دو عملوند

مثال ۱)

$[A - (((C + (D * A)) * (E - f)))]$ $\rightarrow$ $ACDA*+EF-*$
 in $\rightarrow$ pre

۱: پرانتز گذاری

۲: هر عملگر به قبل از پرانتز باز مربوطه منتقل شود

$A / (c - d * (f + e))$ $\rightarrow$ $[A / (c - (d * (f + e)))]$ $\rightarrow$ $/A - c * d + fe$

post $\rightarrow$ in

۱: بت پیمایش از چپ به راست ، هر عملگر ، با دو عملوند قبلی در یک پرانتز گذاشته شود.

۲: عملگر هر پرانتز بین دو عملوند قرار داده شود.

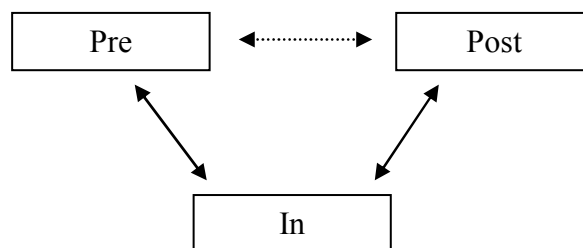
$ABC - * DE / +$ $\rightarrow$ $[(A(BC-)*)(DE/)+]$ $\rightarrow$ $A*(B-C)+D/E$

pre $\rightarrow$ in

۱: با پیمایش از راست به چپ ، هر عملگر ، با دو عملوند قبلی در یک پرانتز گذاشته شود.

۲: عملگر هر پرانتز بین دو عملوند قرار داده شود.

$/ a - c * d + f e$ $\rightarrow$ $[/ a - (c * (d + f e)))]$ $\rightarrow$ $a/(c - d * (f + e))$



الگوریتم ارزیابی عبارات پسوندی:

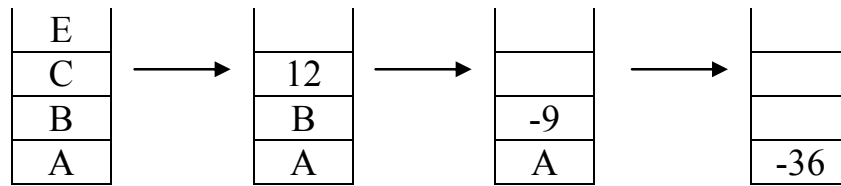
$A=4$, $B=3$, $C=6$, $E=2$

EX: $ABCE*-*$

- عبارت از چپ به راست پیمایش شود

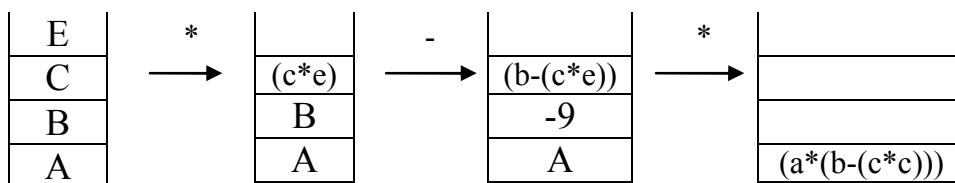
- عملوندها به ترتیب push شوند

- به ازای هر عملگر دو عملوند بالای پشته pop شود و پس از اعمال عملگر ، نتیجه push شود

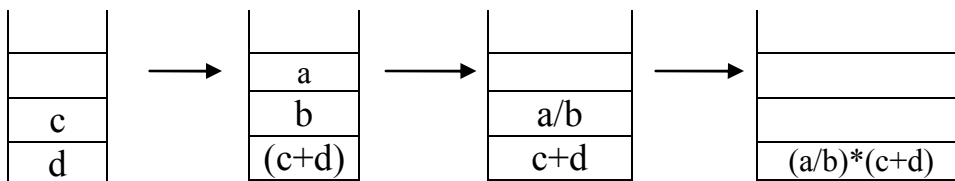


الگوریتم تبدیل post به in:

رشته: $ABCE^{*-}$



pre: $*/AB+CD \rightarrow$ in



تمرین) برنامه ای بنویسید که پرانتزهای اضافی را حذف کند.

پشته چندگانه:

روش اول: آرایه را به دو قسمت تقسیم کنیم:

$S1 : \{sp0.. Max/2\}$ $S2 : \{sp=Max/2..Max\}$

روش دوم: از دو طرف آرایه استفاده کنیم:

$S1 : \{sp0, \text{push} \rightarrow \text{INC}(sp)\}$ $S1 : \{sp \text{ max}, \text{push} \rightarrow \text{DEC}(sp)\}$

شرط پر شدن پشته در حالت دوم: $sp1 = sp$

مثال ۱) عبارت infix زیر را به polish تبدیل کنید:

$A * (B + D) / E - F * (G + H / K)$

جواب:

$\rightarrow ABC+*E/FGHK/+*-$

مثال ۲) تبدیل از infix به post:

$(A + B) * D + E / (F + A * B)$

مثال ۳) تبدیل از pre به in :

$+ - * \uparrow A B C D / E / F + G H$

جواب:

$\rightarrow A \uparrow B * C - D + E / F / (G + H)$

مثال ۴) نتیجه عبارت post مقابل چیست؟

$12, 7, 3, -, /, 2, 1, 5, +, *, +$

$\rightarrow 15(2) \quad 8(1)$

$0(3) \quad 4(4)$ هیچکدام

مثال ۵) مینیم تعداد متغیرهای میانی در محاسبه عبارت جبری زیر ، که بصورت polish می باشد، برابر چند است؟

$a b + c d * / a +$

$1(1) \quad 2(2) \quad 3(3) \rightarrow 4(4)$

مثال ۶) اگر اولویت عملگرها بصورت $/ \otimes \pm$ باشد ، شکل post عبارت مقابل چگونه است؟

$a + b \otimes c / (d - a \otimes (f + b) + b)$

جواب:

$\rightarrow a b c d a f b + \otimes - b + / \otimes +$

مثال ۷) معادل میانوندی (infix) عبارت مقابل چیست؟

$/ * + a b c - a b$

$\rightarrow (a+b)*c/(a-b)$

مثال ۸) تبدیل از پیشوندی به پسوندی:

$++a/b-cd*-ab-+c*d5/a-bc$

جواب:

$\rightarrow a b c d - / + a b - c d 5 * + a b c - / - / +$

مثال ۹) الگوریتم اشتباه برای چک متوازن بودن پرانتزها را در نظر بگیرید:

```

Declare a stack
while (more input is available)
{
    read a character  $\rightarrow$  ch
    if (ch='c') then
        push ch on the stack
    else if ((ch=')') and stack is not empty) then
        pop a char from stack
    else
        print "unbalanced" and exit
    print "Balanced"
}
    
```

}

کدامیک از رشته های نامتوازن زیر توسط الگوریتم فوق ، متوازن در نظر گرفته میشوند:

1: $((()))$

2: $((()))()$

3: $()()$

4: $((()))$ ← زیرا در انتها ، خالی بودن پشته بررسی نمیشود

مثال ۱۰ سه پشته داریم که هر کدام شامل دو عدد است:

$S1: 1, 2$

$S2: 3, 4$

$S3: 5, 6$

و دو عملگر داریم:

$pop(i)$: از پشته i -ام مقداری را pop میکند

$poppush(i,j)$: از پشته i -ام pop کرده و در پشته j -ام قرار میدهد

برای چاپ اعداد یک تا شش بصورت 1,3,5,2,4,6 عملگر $poppush$ حداقل چند بار مورد

استفاده قرار میگیرد:

$3(1) \quad 5(2) \quad 6(3) \quad \rightarrow 4(4)$

مثال ۱۱ N گلوله داریم که باید بصورت صعودی مرتب کنیم (به کمک ترازو). کدام گزینه صحیح

است؟

۱) ۳ گلوله را همواره با دوبار توزین می توان مرتب کرد

۲) ۴ گلوله را در بعضی مواقع می توان با سه

بار توزین مرتب کرد

۳) ۵ گلوله را همیشه با ۴ بار توزین می توان مرتب کرد

۴) هیچکدام

مثال ۱۲ حداقل n گلوله با چندبار عمل توزین مرتب میشوند؟

مثال ۱۳ ۱۰ عدد کبسه طلا داریم که ۹ عدد سالم (شامل ۱۰ سکه ۱۰ گرمی) و یکی تقلبی (شامل ۱۰

سکه ۹ گرمی). حداقل با یک بار وزن کردن کیسه تقلبی را پیدا کنید.

مثال ۱۴ ۱۲ گلوله داریم که یکی خراب است (وزن آن کمتر و یا بیشتر است). حداکثر با ۳ بار

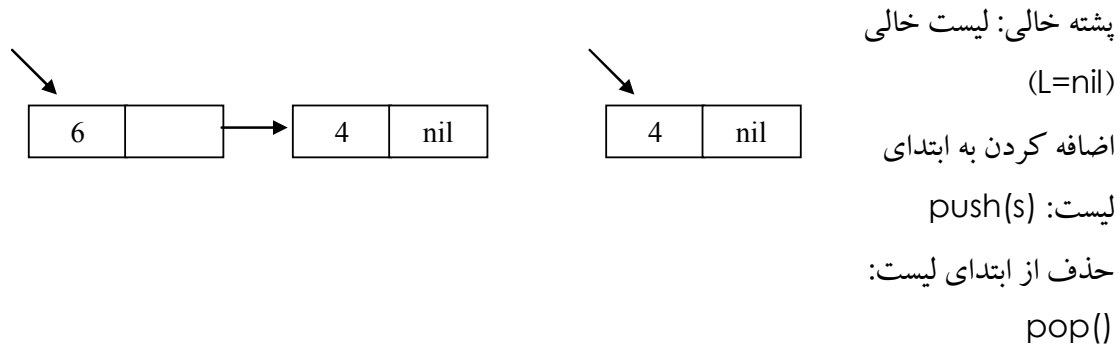
توزین آنرا پیدا کنید.

مثال ۱۵ حدس یک عدد: حداکثر با چند بار سؤال پرسیدن (و استفاده از عملگرهای $<$ و $>$) می توان

به جواب رسید؟

جواب: $\log n$

پیاده سازی پشته با لیست پیوندی (پشته پیوندی):



الگوریتم تبدیل in به post:

- (۱) پشته ای خالی با نام S در نظر میگیریم
 - (۲) عبارت میانوندی را از چپ به راست پیمایش می کنیم ، (تازمانی که به انتهای عبارت نرسیدیم)
 - (۳) کاراکتر بعدی از عبارت را دریافت می کنیم
 - (۱-۳) اگر کاراکتر (بود به داخل پشته push میشود
 - (۲-۳) اگر کاراکتر عملوند بود به عبارت پسوندی اضافه می شود
 - (۳-۳) اگر عملگر بود:
- چنانچه پشته خالی باشد یا اولویت عملگر از عملگر بالای پشته نیز بیشتر باشد ، این عملگر به درون پشته push میشود. در غیراینصورت عملگر بالای پشته pop و در خروجی نوشته میشود تا زمانی که به عملگر برسیم که اولویتش از عملگر خوانده شده کمتر است یا اینکه پشته خالی شود.
- (۴-۳) اگر کاراکتر (بود از پشته pop کرده و در عبارت postfix می نویسیم تا به یک پرانتز باز در پشته برسیم.
 - (۴) تا زمانی که به انتهای رشته ورودی نرسیدیم عملیات ۳ تکرار شود
 - در انتها کل محتوای باقیمانده رشته به عبارت پسوندی اضافه میشود.

نکته: پرانتز بازی که داخل پشته است ، کمترین و بیرون پشته بیشترین اولویت را دارد.

نمونه تمرین:

اولویت ها:

rtl	^
ltr	% / *
ltr	+ -

- 1) in $\rightarrow$ post
- 2) in $\rightarrow$ pre
- 3) post $\rightarrow$ pre
- 4) pre $\rightarrow$ post
- 5) post $\rightarrow$ in
- 6) pre $\rightarrow$ in
- 7) حذف پرانتزهای اضافی
- 8) پرانتز گذاری کامل عبارت
- 9) ارزیابی مقدار یک عبارت پسوندی
- 10) ارزیابی مقدار یک عبارت پیشوندی

صف Queue (FIFO):

ساختمان داده ای است که عناصر به انتها اضافه شده و از ابتدا حذف می شوند.

front: ابتدای صف	EnQ: اضافه کردن به صف (از انتها)
rear: انتهای صف	DeQ: حذف کردن از صف (از ابتدا)
	IsEmpty: تشخیص خالی بودن صف
	IsFull: تشخیص پر بودن صف
	CreateQ: ایجاد یک صف خالی

	A	B	C	D	
	front			rear	

EnQ('A') ; EnQ('B') ; EnQ('C') ;
EnQ('D')

سؤال) فرض می کنیم دو صف Q1 و Q2 با مقادیر زیر داریم:

Q1 = 20, 15, 14, 31, 29, 16

Q2 = 2, 10, 4, 1, 30, 7

اگر A و B عناصر صف باشند ، محتوی صف Q3 بعد از دستورات زیر چیست؟

make(Q3)

l:=0

while (not empty(Q1) and not
empty(Q2))

Begin

l:=l+1

A=DeQ(Q1)

B=DeQ(Q2)

If (B=l+1) then

AddQ(Q3,A-B)

End While

Q3	8	10	9
----	---	----	---

I	A	B
1	20	2
2	15	10
3	14	4
4	31	1
5	29	30
6	16	7

پیاده سازی صف با آرایه:

```

type
Queue=Record
    item: Array[0..(max-1)] of <type>;
    f, r: integer; {-1..max}
end;
```

0					max-1
---	--	--	--	--	-------

f=0: خالی بودن

مکان آخرین عنصر:

r=-1

↓					max-1
	4	6	8		
			↑		

EnQ(4)

EnQ(6)

صف حلقوی: در زمان رسیدن آبه انتها ، مجددا از اول شروع کرد.

r : اندایس اولین مکان خالی

اگر $r=f$ آنها صف خالی است. همچنین برای پر بودن باید شرط زیر بررسی شود:

پر بودن: $f=(r+1) \bmod \max$

حذف عنصر: بررسی خالی بودن ، $f=(f+1) \bmod \max$

اضافه کردن: بررسی پر نبودن ، $r=(r+1) \bmod \max$ ، اضافه کردن در مکان $r-am$

صف اولویت:

-آرایه

-لیست

-درخت

-Heap

عناصر به هر ترتیبی قابل اضافه کردن است. اما حذف عناصر تحت یک سری ضوابط انجام میشود

- صعودی : عناصر از کوچک به بزرگ حذف می شوند

- نزولی: عناصر از بزرگ به کوچک حذف می شوند

صعودی:

- پیدا کردن عنصر کوچک، $O(n)$

- حذف

- بعد از حذف ، بقیه عناصر shift داده شوند و r یکی کم شود. $O(n)$

- حذف منطقی : از عنصر شاخص استفاده کرد (مثلا عدد -1)

- (در صورت استفاده از حذف منطقی می توان روشهایی نظیر فشرده سازی و یا جایگزینی را نیز بکار برد)

- استفاده از صف دایره ای مرتب

ساختار های داده ای:

(۱) ایستا

✓	رکورد
✓	آرایه

(۲) پویا

✓	گراف
✓	درخت
✓	لیست پیوندی

(۳) نیمه ایستا

✓	صف
✓	پشته

مشکلات ساختارهای ایستا (static):

۱- اتلاف حافظه

۲- ذخیره سازی عناصر پشت سر هم (که در زمان درج و یا حذف باید عناصر دیگر shift داده شوند)

در ساختار های پویا هر جا نیاز باشد از سیستم حافظه گرفته و هر جا نیاز نبود به سیستم برگردانده میشود. در این ساختار از مفهومی بنام اشاره گر استفاده میشود.

$$C = \begin{cases} malloc \\ free \end{cases}$$

$$C++ = \begin{cases} new \\ delete \end{cases}$$

$$pascal = \begin{cases} new \\ dispose \end{cases}$$

لیست پیوندی:

به لیستی از عناصر که به کمک اشاره گر به هم مرتبط شده اند ، لیست پیوندی گفته میشود:

$$node = \begin{cases} Data \\ Link \end{cases}$$

Data	Link
------	------

در قسمت link آدرس گره ای که منطبقا بعد از گره فعلی قرار دارد ، باید مشخص شود.

گره ۲

گره ۳

گره ۱

گره ۴

برای پیاده سازی ساختار لیست پیوندی ، نیاز به رکورد یا ساختار داریم:

$$node = \begin{cases} info : integer \\ next : pointer \end{cases}$$

Heap

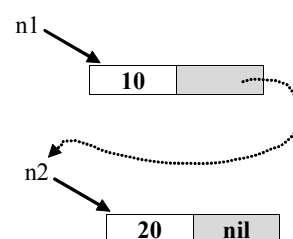
پاسکال:	C:
<pre> Type nptr= ^node; node= Record info: integer; next: nptr; end;</pre>	<pre> Struct Node{ int info; Node *next; }</pre>

Var n1,n2: nptr;

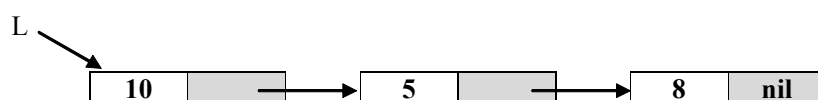
به اشاره گر n1 که جزء گره ها نیست اشاره گر خارجی گویند که به کمک آن به کل مجموعه دسترسی داریم.

```

new(n1);
n1^.info :=10 ;
new(n2);
n2^.info :=20 ;
n1^.next :=n2;
n2^.next :=nil ;
```



دسترسی به عناصر لیست و پیمایش:



writeln(L^.info); داده گره اول

writeln(L^.next^.info); داده گره دوم

اشاره گر برای پیمایش **P:=L;**

بین گره ها **P:=P^.next;**

مثال ۱) تابعی بنویسید که اشاره گر خارجی به لیست پیوندی را دریافت کرده و مجموع عناصر لیست را چاپ کند.

```

function sum(L:nptr):integer;
var
  s:integer;
  p:nptr;
begin
  s:=0; p:=L;
  while p<>nil do
  begin
    s:=s + p^.info;
    p:= p^.next;
  end
  writeln(s)
end;
```

مثال ۲) روالی بنویسید که عنصری با مقدار داده شده X را به ابتدای لیست اضافه کند.

```
Var
    n1:nptr;
begin
    new(n1);
    n1^.info:=x;
    n1^.next:=L;
    L:=n1;
end
```

مثال ۳) عنصری با مقدار X را به عنوان گره انتهایی لیست اضافه نمایید:

```
P:=L;
while P^.next<>nil do           // اشاره گر به گره آخر
    P:=P^.next;
new(n);
n^.info:=x;
n^.next=nil;
P^.next:=n;
```

مثال ۴) اضافه نمودن گره با مقدار X بعد از گره n ام لیست:

```
P:=L;
for i:=1 to n-1 do             // انتقال روی گره مورد نظر
    P:=P^.next;
new(n);
n^.info:=X;
n^.info:=P^.next;              // تنظیم اشاره گرها
P^.next:=n;
```

مثال ۵) چگونه میتوان گره ای با مقدار X را قبل از عنصری با اشاره گر Q اضافه کرد؟

```
P:=L
while P^.next<>q do
    P:=P^.next;
//...
// ترتیب تنظیم اشاره گرها فرق نمی کند
```

مثال ۶) حذف گره انتهایی لیست L و برگرداندن مقدار آن:

```
function RE(var L:nptr):integer;
var
    P:nptr;
    S:integer;
begin
    P:=L;
    While P^.next^.next<>nil do
        P:=P^.next;
    q:=P^.next;
```

```
P^.next=nil;
S:=q^.info;
dispose(q);
RE:=S;
```

end

مثال ۷) حذف عنصر از ابتدای لیست:

بهرتر است که در ابتدا پیوند عنصر اول را از بین ببریم

```
P:=L;
L:=L^.next;
P^.next=nil;
dispose(P);
```

مثال ۸) حذف عنصر K ام لیست:

```
P:=L;
for i:=1 to k-2 do
    P:=P^.next;
q:=P^.next;
P^.next:=q^.next;
q^.next=nil;
dispose(q);
```

مثال ۹) فرض کنید L1 و L2 اشاره گر به دو لیست پیوندی خطی باشند:

الف) دو لیست پیوندی را به همدیگر متصل کنید:

```
P:=L1;
while p^.next<>nil do
    P:=P^.next;
P^.next:=L2;
L2:=nil;
```

ب) لیست بنام L3 ایجاد نمایید که حاوی عناصر لیست ها L1 و L2 بصورت یک در میان باشد:

```
void alternateJoin(nptr *L, nptr *L1, nptr *L2){
    nptr *p=L,*q;
    while(L1 != NULL || L2 != NULL)
    {
        if(L1 != NULL){
            p->next = new nptr;
            q = p;
            p->data = L1->data;
            p = p->next;
            L1 = L1->next;
        }
        if(L2 != NULL){
            p->next = new nptr;
            q = p;
            p->data = L2->data;
            p = p->next;
            L2 = L2->next;
        }
    }
    q->next=NULL;
```

<pre> } void oddeven(nptr*L, nptr*L1, nptr*L2){ nptr *k = L, *p, *q, *endi; for(int i=0; i<2; i++) { p=L1; q=L2; while(p != NULL) { if(p->data % 2 != i) { k->data = p->data; k->next = new nptr; endi = k; k = k->next; } p = p->next; } while(q != NULL) { if(q->data % 2 != i) { k->data = q->data; k->next = new nptr; endi = k; k = k->next; } q = q->next; } } endi->next = NULL; } </pre>	ج) لیستی بنام L3 ایجاد نمایید که ابتدای لیست حاوی عناصر دو لیست L1 و L2 و بخش دوم لیست حاوی عناصر زوج L1 و L2 باشد.
---	--

<pre> void bubbleSort(nptr *L){ nptr *p,*q; int flag; do{ flag=0; p=L; q=p->next; while(q!=NULL){ if(p->data > q->data){ inchange(&p->data,&q- >data); flag=1; } p=q; q=q->next; } } } </pre>	د) لیست L1 را مرتب نمایید.
--	-----------------------------------

```

    }
    p=p->next;
    q=q->next;
}
}while(flag);

```

مثال ۱۰) عملکرد الگوریتم زیر چیست:

```

Procedure what (L:nptr);
begin
    if L<>nil then
    begin
        write (L^.info);    //1
        what (L^.next);    //2
    end;
end;

```

جواب: عناصر لیست را بصورت بازگشتی چاپ می کند
اگر جای ۱ و ۲ عوض شود، از آخر به اول چاپ خواهند شد.

مثال ۱۱) خروجی زیر چیست؟

```

p:=L;
q:=nil;
while p<>nil do
begin
    r:=q;
    q:=p;
    p:=p^.next;
    q^.next:=r;
end;
L:=q;

```

جواب: لیست را معکوس می کند.
در مرحله آخر p به گره آخر اشاره می کند که به کمک آن مکان L را اصلاح می کنیم.

لیست های حلقوی:

مثال ۱) محاسبه مجموع مقادیر گره های لیست حلقوی:

```

P:=L;
Repeat
    S:=S+P^.info;
    P:=P^.next;
Until P=L;

```

مثال ۲) اضافه کردن عنصر به ابتدای لیست.

```

new(p);
p^.info:=x;
q:=L
while q^.next<>L do

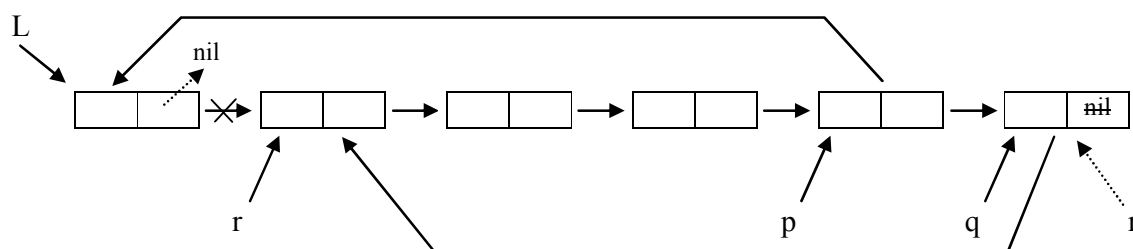
```

```
q:=q^.next;
q^.next:=p;
p^.next:=L;
L:=p;
```

هنگام اضافه کردن به انتها به این خط نیازی نیست //

نکته: اگر اشاره گر خارجی به گره آخر اشاره کند، پیچیدگی $O(1)$ می شود، زیرا نیاز به پیمایش نخواهد بود. در غیر اینصورت پیچیدگی $O(n)$ است.

تمرین (جابجایی عناصر اول و آخر لیست:



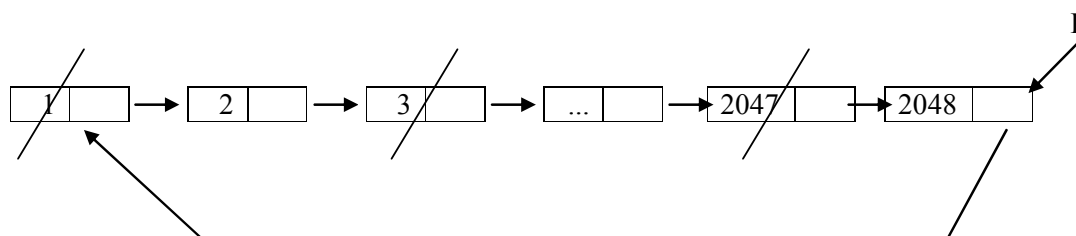
```
r := L^.next;
```

=> حلقه

- آخری q
- ماقبل آخری p

```
L^.next := nil;
```

مثال اگر لیست مقابل را داشته باشید، خروجی الگوریتم چیست؟



(۱) ۲۰۴۷

(۲) ۱

(۳) ۱۰۲۴

(۴) ۲۰۴۸ =>

```
while p^.next<>p do
```

```
begin
```

```
  p^.next := p^.next^.next;
```

```
  p:=p^.next;
```

```
end;
```

```
write(p^.info)
```

دفعه اول عنصر یک حذف میشود، و بعد روی عنصر دو قرار میگیرد سپس عنصر جلوی آنرا حذف می کند ... تا در نهایت همه عناصر فرد حذف شوند.
دفعه دوم یک در میان مضارب ۲ را حذف می کند (ضرب ۲ باشد و ضرب ۴ نباشد) دفعه بعد ضرب ۴ باشد و ضرب ۸ باشد...
... ضرب ۲۰۴۸

مثال ۲) در حلقوی گره های با مقادیر فرد را حذف کنید:

```
P:=L;
Repeat
    while(P^.next^.info) mod 2=1 do
        P^.next := P^.next ^.info;
    P:=P^.next;
Until P=L;
```

حالت خاص زمانی است که همه فرد باشند. پس باید بیرون حلقه آن را چک کنید.
همچنین اگر اولین عنصر فرد باشد، باید اشاره گر خارجی را نیز اصلاح کنید.

لیست دو پیوندی (Double Linked List):

هر گره **node** :

- داده : info
- اشاره گر به گره بعدی : next
- اشاره گر به گره قبلی : prev

ساختار در پاسکال:	ساختار در C:
<pre>Type Dnptr = ^node; node = Record; info: Integer; next, prev: pnptr; end;</pre>	<pre>typedef struct dnode *dnptr; struct dnode{ int info; dnptr, next, prev; }</pre>

مثال ۱) مجموعه دستوراتی بنویسید که گره ای با مقدار X را به ابتدای لیست دو پیوندی L اضافه نماید.

پاسکال:	C:
<pre>p:dnptr; new(p); p^.info:=x; p^.prev:=nil; p^.next:=L; L^.prev:=p; L:=p;</pre>	<pre>dnptr p; p=new struct dnode; p->info = x; p->next = L; p->prev = NULL; L=p;</pre>

مثال ۲) حذف یک گره:

```
p^.prev^.next := p^.next;
p^.nexy^.prev := p^.prev;
```

مثال ۳) خروجی برنامه زیر چیست؟

Procedure k(L)

begin

if $L \neq \text{nil}$ then

begin

k(Link(L));

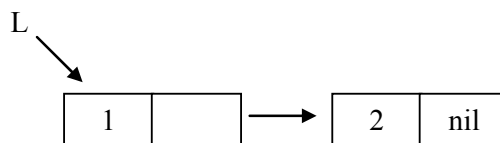
write(Data(L));

k(Link(L));

write(Data(L));

end;

end;



→ 221221 (۱)

121212 (۲)

221122 (۳)

221222 (۴)

مثال (۴) نتیجه p چه میشود؟

Procedure k(m, n: pointer; var p: pointer);

var

integer;

begin

t:=m;

while(t^.link ≠ nil) do

t:=t^.link;

// 1

t^.linkL:=n;

p:=m;

end;

(۱) الحاق دو لیست حلقوی

(۲) الحاق دو لیست پیوندی خطی

(۳) الحاق دو لیست حلقوی غیر تهی

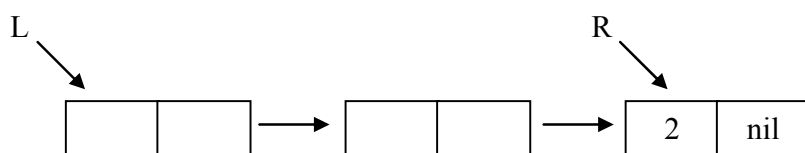
(۴) الحاق دو لیست پیوندی خطی که حداقل یکی از آنها غیر تهی باشد

نکته: اگر $m = \text{nil}$ خطا دارد

اگر در مکان ۱ دستور $t \wedge \text{link} = n$ if($m \neq \text{nil}$) قرار داده شود

، آنگاه گزینه ۲ صحیح است

مثال (۵) هزینه کدام یک از اعمال زیر وابسته به تعداد عناصر لیست است؟



(۱) حذف اولین عنصر

→ (۲) حذف آخرین عنصر

(۳) درج یک عنصر در انتهای لیست

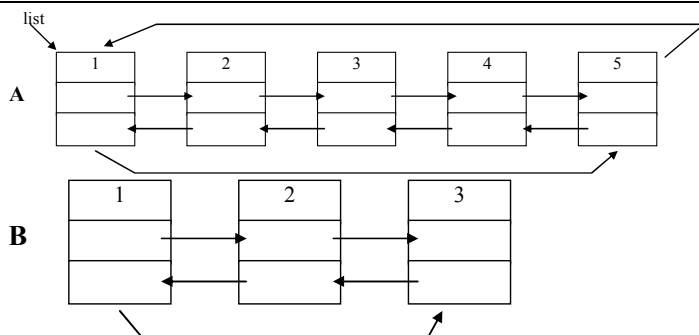
(۴) درج یک عنصر در ابتدای لیست

مثال (۶) کدام دستور لیست A را به B تبدیل می کند.

(۱) $\text{list} \rightarrow \text{next} \rightarrow \text{next} \rightarrow \text{next} = \text{list} \rightarrow \text{prev}$

→ (۲) $\text{list} \rightarrow \text{prev} = \text{list} \rightarrow \text{next} \rightarrow \text{next}$

(۳) $\text{list} \rightarrow \text{next} = \text{list} \rightarrow \text{prev} \rightarrow \text{next} \rightarrow \text{next}$



مثال (۷) برای حذف یک عنصر در لیست دو پیوندی چند آدرس باید جابجا شود؟ ۲ تا

مثال (۸) برای اضافه کردن یک عنصر حداکثر چند آدرس باید جابجا شود؟ ۴ تا

مثال ۹) اگر X اشاره گری به لیست دو پیوندی باشد ، دستورات زیر چه عملی انجام می دهند؟

- ۱) حذف گره قبلی $p \rightarrow \text{link} = x$
- ۲) حذف گره بعدی $p \rightarrow \text{rlink} = x \rightarrow \text{rlink}$
- ۳) حذف خود X $x \rightarrow \text{rlink} \rightarrow \text{llink} = p$
- ۴) درج X $x \rightarrow \text{rlink} = p$

مثال ۹) می خواهیم تغییراتی در لیست تک پیوندی اعمال کنیم که عمل افزودن عنصر به ابتدا یا انتهای لیست با عملیاتی از مرتبه

$O(1)$ قابل انجام باشد. لیست پیوندی را...

۱) دوطرفه می کنیم ۲) حلقوی می کنیم

۳) معکوس می کنیم ۴) حلقوی + اشاره گر خارجی به گره آخر

مثال ۱۰) سه تابع و سه اشاره گر داریم:

first: اشاره گر به گره اول را می دهد

next: اشاره گر به گره بعدی را می دهد

end: اشاره گر به گره انتهای لیست را می دهد

تعداد دفعات اجرای تابع first چقدر است؟

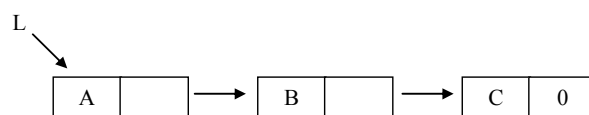
(طول لیست = n)

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

```
P1:=first(L);
While P1<>end(L) do
begin
  P2:=P1;
  while P2<>end(L) do
  begin
    P2:=next(P2,L);
    P3:=first(L);
    while P3<>P2 do
      P3:=Next(P3,L);
    endl;
    P1:=next(P1,L);
  end;
end;
```

مثال ۱۱) روال زیر چه کاری انجام میدهد؟

```
Procedure W(L)
begin
  if L<>0 then
  begin
    W(Link(L));
    Print(Data(L));
    W(Link(L));
    Print(Data(L));
  end;
end;
```



CCBCCBACCBCCBA

کاربرد لیست ها:

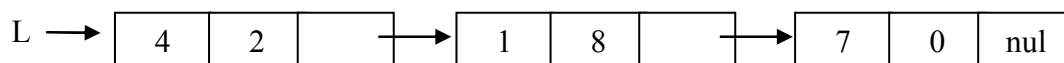
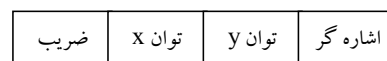
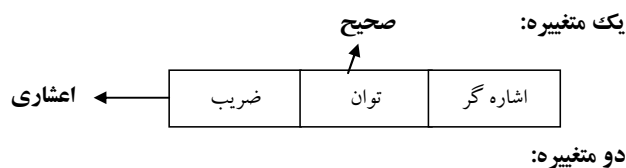
- ✓ چند جمله ای
- ✓ محاسبات اعداد بزرگ
- ✓ حل مسئله جوزف (Josephus)

محاسبات اعداد بزرگ:

ASSUME : $e_n > e_{n-1} > e_{n-2} > \dots > e_0$

$$P(x) = a_n X^{e_n} + a_{n-1} X^{e_{n-1}} + \dots + a_0 X^{e_0}$$

$$P(x) = 4X^{12} + X^8 + 7$$

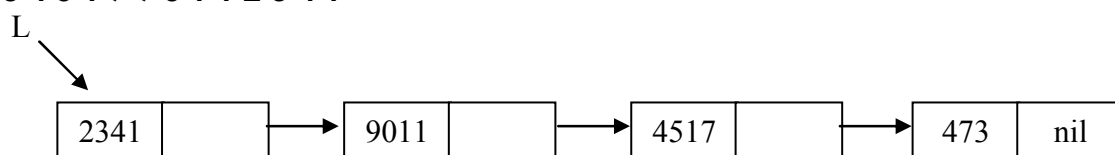


عملیات بر روی چند جمله ای:

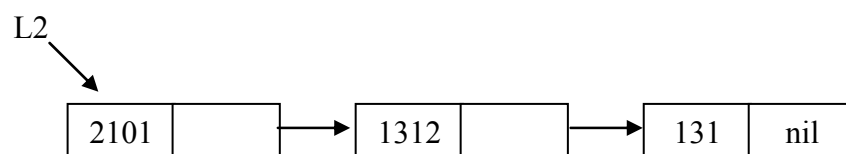
- ۱- ارزیابی چند جمله ای
- ۲- جمع دو چند جمله ای
- ۳- ضرب دو چند جمله ای

اعداد بزرگ:

X = 4 7 3 4 5 1 7 9 0 1 1 2 3 4 1



جمع با L2:



$C = 0$

```

{
    sum → { >
            <=
    }
    sum = L->Data + L2->Data + C;
    C = sum / 10000;
    sum = sum % 10000;
    AddLast(L3, sum);
}
    
```

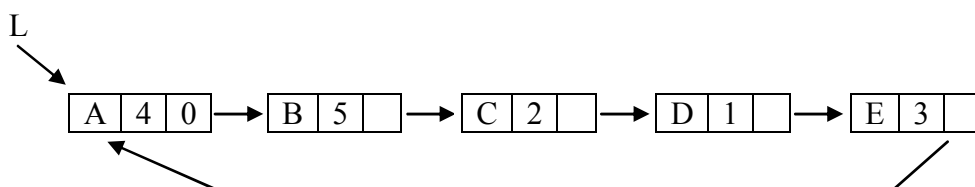
اضافه کردن بقیه گره ها + کری { L1

اضافه کردن بقیه گره ها + کری { L2

افزودن گره به انتها →

مساله جوزف:

ایجاد لیست حلقوی شامل n عنصر



از یک نقطه بصورت تصادفی شروع می کنیم به شمردن. اولین گره ای که sum را به بیش از S رساند حذف می کنیم و از گره بعدی (با sum=0) شمارش را مجددا ادامه می دهیم.
خروجی: عناصری که حذف می شوند.

تمرین) تابعی بنویسید که تعداد فراخوانی های فیبوناچی را حساب کند:

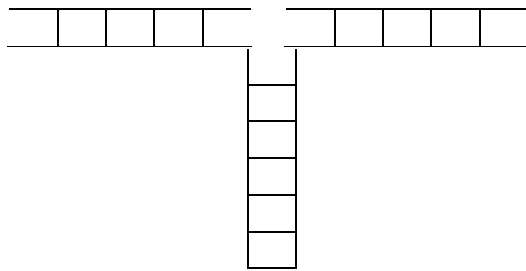
```

int fibo(int n)
{
    if(n==1 || n==2)
        return(1);
    else if(n>2)
        return(f(n-2)+f(n-1));
}
    
```

تمرین ۲) لیست خطی: تابعی بنویسید که اشتراک ، اجتماع و تفاضل دو لیست خطی را محاسبه کنند.

تمرین ۳) تابعی بنویسید که لیست خطی را بصورت بازگشتی معکوس کند.

تمرین ۴) برنامه ای بنویسید که برای n متغیر تعداد همه جایگشت های ممکن جهت ورود به پشته و سپس انتقال به صف دوم را بدست آورد:



e.g. n=2:

a1,a2

a2,a1

تمرین ۵) تابع زیر را بصورت بازگشتی و غیر بازگشتی بنویسید:

$$f(x) = \begin{cases} \frac{x}{2} & x \% 2 = 0 \\ f(f(3 * x + 1)) & \text{else} \end{cases}$$

e.g: $f(5) = 4$

$f(7) = f(f(22)) = f(11) = f(f(34)) = f(17) = f(f(52)) = f(26) = 13$

بازگشتی:

```
int f1(int n)
{
    if(n%2 == 0)
        return (n/2);
    else
        return (f(f(3*n+1)));
}
```

غیر بازگشتی:

```
int f2(int n)
{
    while(n%2!=0)
        n=(n*3+1)/2;

    return(n/2);
}
```

گراف:

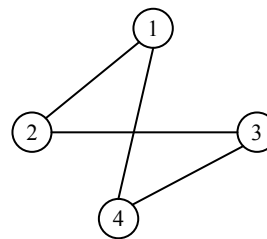
پیاده سازی گراف:

- مجموعه ها
- ماتریس همجواری
- لیست پیوندی

در گراف بدون جهت ماتریس مجاورت: متقارن است ، پس کافیت فقط بالا یا پایین آنرا ذخیره کرد:

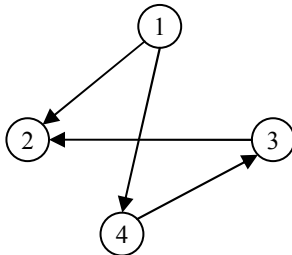
$$T_{n \times n} \rightarrow T[i, j] = \begin{cases} 0 & e \notin E \\ 1 & e \in E \end{cases}$$

$$T_{4 \times 4} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$



متقارن است: n^2

برای صرفه جویی در ذخیره سازی می توان آنرا بصورت بالا مثلثی و یا پایین مثلثی ذخیره کرد

$$(n(n+1)/2)$$


$$T_{4 \times 4} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

توجه : گراف جهت داری که در آن دور وجود داشته باشد DAG گفته میشود.

درجه راس:

بدون جهت:

$$v_i \rightarrow \begin{cases} row(i) \\ col(i) \end{cases}$$

جهت دار:

$$v_i \rightarrow \begin{cases} in & row(i) \\ out & col(i) \end{cases}$$

مثال ۱) آیا مسیری بطول ۲ بین آ و ژ وجود دارد یا نه؟

```
f:=false;
for k:=1 to n do
    f:=f OR (T[i,k] and T[k,j])
```

معادل است با:

$(T[i,1] \text{ and } T[1,j]) \text{ or } (T[i,2] \text{ and } T[2,j]) \text{ or } \dots \text{ or } (T[i,n] \text{ and } T[n,j])$

مثال ۲) پیدا کردن مسیر بطول k :

T = همجواری

$T_2[i,j] = T * T$ حاصلضرب ماتریس

...

$T_k = T * T_{k-1}$

پیچیدگی الگوریتم ضرب از مرتبه ۲: $O(n^4)$ پیچیدگی الگوریتم ضرب از مرتبه بالاتر
۲: $O(n^3)$

$P = T$; $adj = T$

for $k := 1$ to $n-1$ do

begin

$m \leftarrow T_{k+1}$ [$adj = adj.T$] $O(n^3)$

$p = p \text{ or } m$

end;

ماتریس مسیر: آیا مسیری با طول معین k بین دو گره i و j وجود دارد؟

$T[i,j] = 1$ وجود یال بین دو گره

$P[i,j] = 1$

وجود مسیر بین دو گره

$P[i,j]$ هنگامی یک است که مسیری بین گروه های i تا k وجود داشته باشد که i به j وصل کند:

آیا مسیری بین i و j وجود دارد؟

$T_k = T_{k-1} + T$

$P = T_1 \text{ or } T_2 \text{ or } T_3 \text{ or } \dots \text{ or } T_n$

for $k := 1$ to $n-1$ do

begin

$M = [T_{k+1}]$

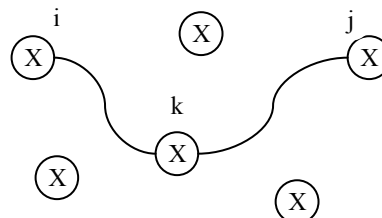
$P = P \text{ or } m$

end;

یالی وجود داشته باشد:

$P_{k-1}[i,j] = 1$

$P_k[i,j] = \begin{cases} P_{k-1}[i,j] = 1 \\ P_{k-1}[i,k] \& P_{k-1}[k,j] \end{cases}$



$P_k = P_{k-1}[i,j] \text{ or } (P_{k-1}[i,k] \text{ and } P_{k-1}[k,j])$

P_k هنگامی یک است که:

for $k := 1$ to N do

for $i := 1$ to n do

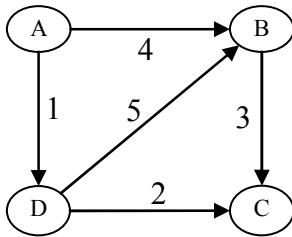
if $(p[i,k] = \text{True})$

for j:=1 to n do
p[i,j]=P[i,j] or (p[i,k] and p[k,j])

گرافهای وزن دار:

- وارشال: کوتاهترین مسیر بین کلیه گره ها
- Dijkstra: کوتاهترین مسیر (کم هزینه ترین) بین دو گره مشخص

وارشال:



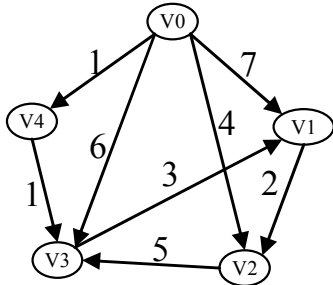
$$Q_0 = \begin{bmatrix} 100 & 4 & 100 & 1 \\ 100 & 100 & 3 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 5 & 2 & 100 \end{bmatrix}$$

کوتاهترین مسیر به ۱ و ۲ بطوریکه گره ۱ را میانی بگیریم:

$$Q_1[l,j] = \min\{Q_0[l,j], Q_0[l,1] + Q_1[1,j]\}$$

...

Dijkstra: پیدا کردن کوتاهترین مسیر بدون در نظر گرفتن گره های میانی



$$w = \begin{bmatrix} 100 & 7 & 4 & 6 & 1 \\ 100 & 100 & 2 & 100 & 100 \\ 100 & 100 & 100 & 5 & 100 \\ 100 & 3 & 100 & 100 & 100 \\ 100 & 100 & 100 & 1 & 100 \end{bmatrix}$$

d_0 : کوتاهترین فاصله بین گره صفر تا گره مبدا

d_4 : کوتاهترین فاصله بین گره ۴ تا گره مبدا

$$\begin{aligned} 1) \quad p &= \{V_0\} & T &= \{V_1, V_2, V_3, V_4\} \\ \min &= \{d_0 + w_{01}, d_0 + w_{02}, d_0 + w_{03}, d_0 + w_{04}\} \\ \min\{7, 4, 6, 1\} &= 1 \end{aligned}$$

$$\begin{aligned} 2) \quad p &= \{V_0, V_4\} & T &= \{V_1, V_2, V_3\} \\ \min &= \{d_0 + w_{01}, d_0 + w_{02}, d_0 + w_{03}, d_4 + w_{41}, d_4 + w_{42}, d_4 + w_{43}\} \\ \min\{7, 4, 6, 101, 101, 2\} &= 2 \end{aligned}$$

$$\begin{aligned} 3) \quad p &= \{V_0, V_4, V_3\} & T &= \{V_1, V_2\} \\ \min &= \{7, 4, 101, 101, 5, 102\} = 4 \end{aligned}$$

...

V0=0	V1=5	V2=4	V3=2	V4=1
------	------	------	------	------

با شروع از گره V0:

الگوریتم وارشال:

```
P=T
for k:=1 to N do
  for i:=1 to n do
    for j:=1 to n do
      P[i,j]=P[i,j] or (P[i,j] and P[k,j])
```

→ $O(n^3)$

الگوریتم بهبود یافته:

```
P=T
for k:=1 to N do
  for i:=1 to n do
    if P[i,j]=True then
      for j:=1 to n do
        P[i,j]=P[i,j] or (P[i,j] and P[k,j])
```

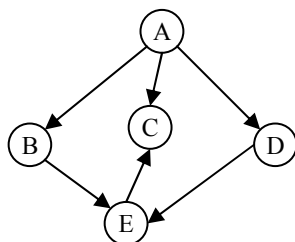
پیمایش گراف (تعیین درخت پوشا)

- پیمایش عرضی (BFS)

- پیمایش عمقی (DFS)

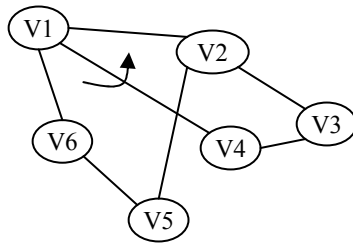
پیمایش عرضی bfs (پیاده سازی با صف):

```
EnQ(A);
while not Empty(Q) do
  Begin
    DeQ(Q)
    کلیه گره های همجوار گره حذف شده به صف اضافه شوند
  End;
```



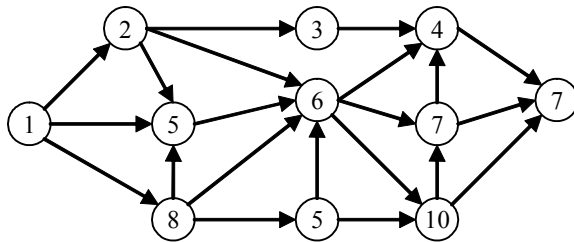
A	B	C	D	E		
---	---	---	---	---	--	--

→ ABCDE



از چپ به راست:

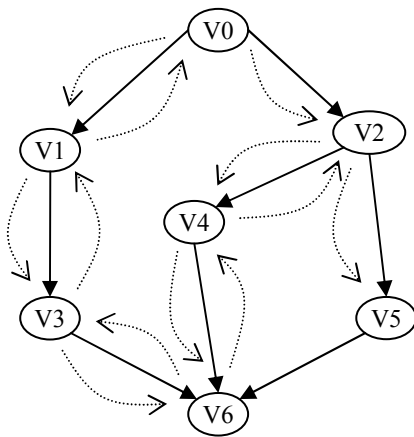
V1 V6 V4 V2 V5 V3



S=1

1,2,5,8,3,6,9,4,7,10,11

پیمایش عمق درخت dfs (depth first search):



visited:

f	f	f	f	f	f	F
0	1	2	3	4	5	6

V0 , V1 , V3 , V6 , V2 , V4 , V5

```
dfs(int n)
{
    visited[n]:=true;
    for each w vertex adjacent with n
        dfs(w);
}
```

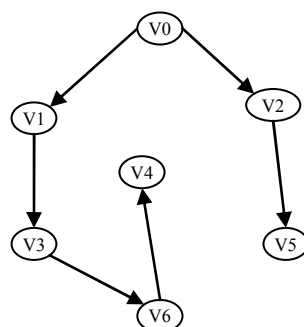
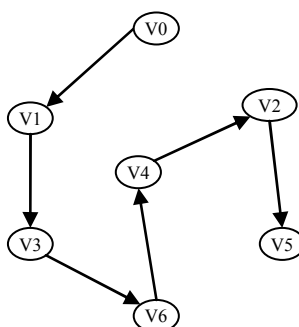
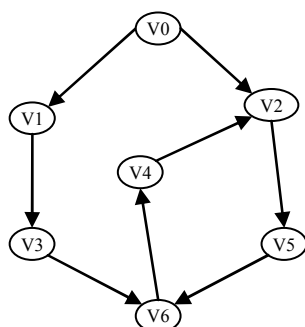
درخت پوشا (Spanning tree):

درخت همبندی که حداقل تعداد یال را داشته باشد.

Graph

DFS

BFS

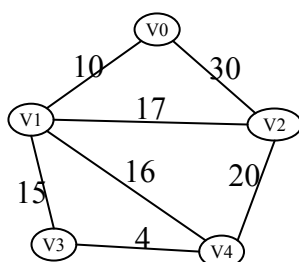


درخت پوشای بهینه: minimum spanning tree

(در گراف های وزن دار)

- کراسکال kruskal

- prim



→ در روش کراسکال →

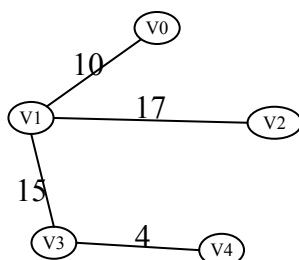
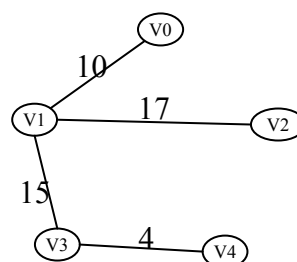
اگر جهت دار نباشد به

ترتیب یالها را از وزن کم تر

انتخاب می کنیم

تا جایی که سیکل ایجاد

نشود



prim: از یک راس شروع می کنیم و هربار کم وزن ترین یال با

گره بعدی انتخاب میشود (سیکل ایجاد نشود) ←

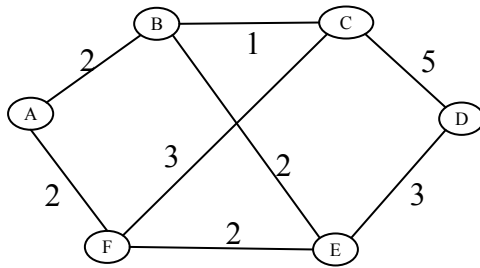
توجه: دو مجموعه داریم که گره مبدا از مجموعه اول و گره

مقصد از مجموعه دوم انتخاب می شوند.

مثلا در ابتدا این دو مجموعه بصورت $\{v_0\}$ و $\{V_1, \dots, V_4\}$ است.

سؤال ۱) اگر A مبدا باشد و از دو روش prim و کراسکال استفاده کنیم ، گره های انتخاب شده

چگونه خواهد بود:



K: BC,BE,FE,AF,ED (۱)

P: AF,FE,BE,BC,ED

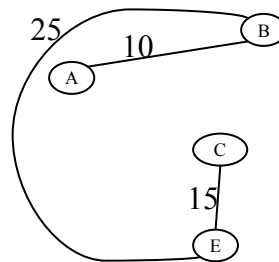
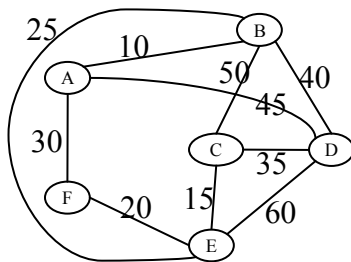
→ K: BC,BA,AF,FE,ED (۲)

→ P: AF,FE,EB BC,ED

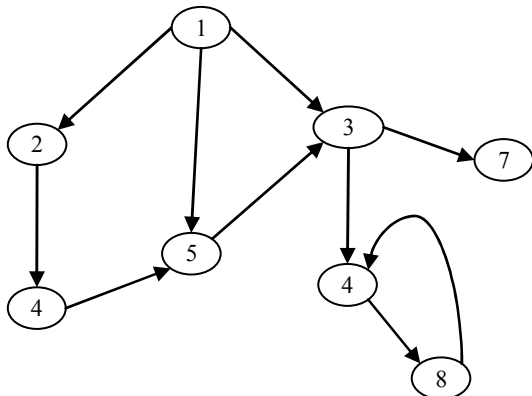
K: BC,FE,AF,AB,ED (۳)

P: AB,AF,FE,BC,ED

سؤال ۲) در انتهای مرحله سوم prim:



سؤال ۳) طبق الگوریتم زیر، شماره گره ۴ چند است؟



count:=0;

Dfs(v:vert x)

w:vert;

begin

mark v as visited

for all verties w adjacent

to v do

if (w is not visited)

then

Dfs(w);

count:=count+1;

DFN[V]:=count;

end;

۵ (۴)

→ ۶ (۳)

۳ (۲)

۴ (۱)

سؤال ۴) یک گراف کامل با ۱۰ راس داریم که بدون جهت است و یال های آن بصورت زیر

تعریف شده است:

$$w_{ij} = w_{ji} = \begin{cases} i+j & i+j \geq 5 \\ i^2 + j^2 & i+j < 5 \end{cases}$$

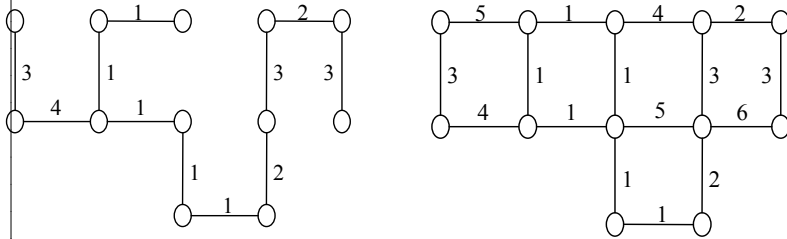
وزن درخت پوشای مینیمال (MST) چند میشود؟

۶۵ (۴)

۶۶ (۳)

۷۵ (۲)

۷۶ (۱)



سؤال ۵) وزن درخت پوشای مینیمال

چند میشود؟

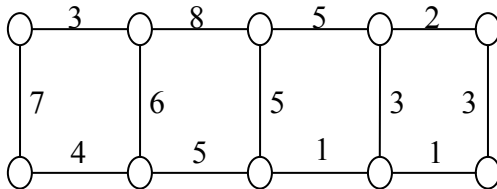
۲۴ (۲) → ۲۲ (۱)

۲۰ (۴)

۲۵ (۳)

سؤال ۶) گراف ساده غیر جهت دار بدون حلقه G را در نظر بگیرید که شامل ۱۰ مولفه همبند است.

اگر تعداد رئوس گراف برابر ۶۰ باشد در این صورت مجموع درجه های راس های گراف چند است؟



سؤال ۷) وزن درخت MST چند است؟

۳۵ (۲)

→ ۳۰ (۱)

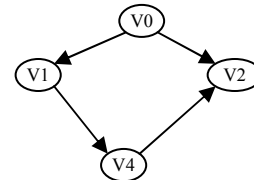
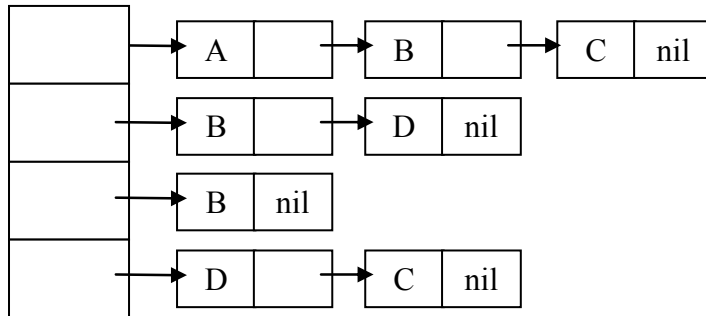
۳۳ (۴)

۲۵ (۳)

پیاده سازی گراف:

- آرایه: $M[n*n]$

- لیست پیوندی (n گره)



```

nptr = ^node;
node = RECORD
    item: <int>;
    adj: nptr;
end;
  
```

X: Array[1..n] of nptr;

پیاده سازی Dfs:

آرایه:

حافظه $O(n^2)$

پیمایش $O(n^2)$

لیست پیوندی:

حافظه $O(V+E)$

پیمایش $O(E)$

درخت:

مجموعه ی یک یا چند گره که:

- ۱- یکی از گره ها از بقیه متمایز بوده و ریشه نامیده میشود
- ۲- بقیه گره ها به صفر یا چند زیر مجموعه مجزا تقسیم میشوند ($T_0, \dots, T_n$) که هر کدام از T_i ها خود درخت بوده و زیر درخت ریشه نامیده میشود.
- درجه هر گره: تعداد زیر درخت ها
- درجه درخت: بیشترین درجه موجود بین کلیه نودها
- برگ: گره ای با درجه صفر
- فرزندان یک گره: ریشه های زیر درخت های یک گره
- نیاکان یک گره: کلیه گره های موجود در مسیر منتهی به ریشه (بجز خودش)
- گره های همزاد sibling (هم پدر) «گره هایی که یک ریشه مشترک داشته باشند».

سطح Level:

سطح ریشه = ۱

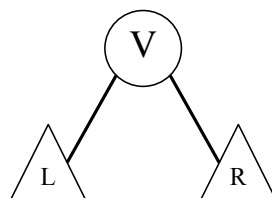
سطح هر گره = سطح گره پدر + ۱

ارتفاع درخت (depth) = ماکزیمم سطح موجود بین کلیه گره ها

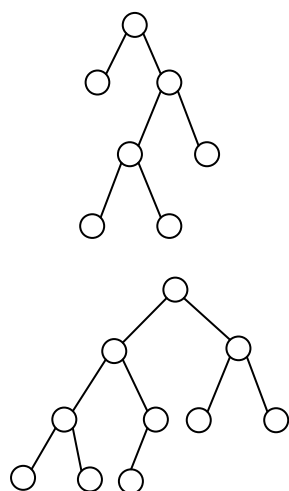
درخت دودویی:

درجه هر گره حداکثر ۲ است.

2)



1) تهی

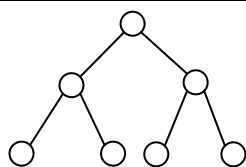


- پر Full: هر گره یا فرزند ندارد یا دقیقا دو فرزند دارد

- کامل Complete: تا سطح $h-1$ یک درخت perfect است و گره

های سطح آخر از چپ به راست چیده شده

- تمام Perfect: کلیه گره ها (بجز برگ) دقیقا دو فرزند داشته باشند و



همه برگها در سطح آخر هستند

نکته ۱) درختی کامل است که اگر آنرا شماره گذاری کنیم ، شماره ها متناظر با شماره های perfect باشد.

نکته ۲) در درخت perfect اگر یک گره در مکان i باشد فرزند چپ آن در مکان $2i$ و فرزند راست در مکان $2i+1$ است و پدر در مکان $i/2$ است.

نکته ۳) حداکثر گره های موجود در سطح i : 2^{i-1}

نکته ۴) تعداد گره های درخت perfect به عمق h :

$$1+q+q^2+\dots+q^k=\frac{1-q^{k+1}}{1-q} \quad \rightarrow \quad 2^0+2^1+2^2+\dots+2^{h-1}=2^h-1$$

نکته ۵) حداقل تعداد گره های درخت کامل به ارتفاع h :

$$2^{h-1}-1+1=2^{h-1}$$

پیاده سازی درخت کامل و تمام با آرایه:

$$X[1..2^{h-1}]$$

$$X[i] \rightarrow \begin{cases} X[i/2] \rightarrow Parent \\ X[2i] \rightarrow LeftChild \\ X[2i+1] \rightarrow RightChild \end{cases}$$

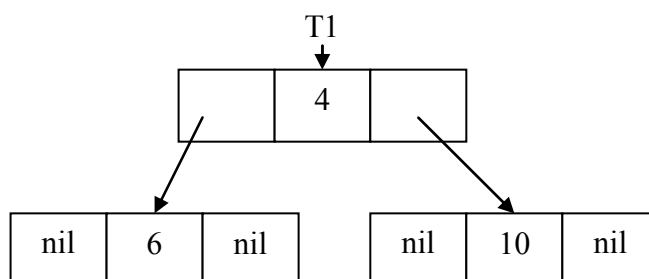
این روش برای درخت های مورب به هیچ وجه مقرون به صرفه نیست.

پیاده سازی با لیست:

Type

```
Tptr=^tree;
Tree=Record
    item: char;
    Left,Right: Tptr;
end;
```

```
T1,T2,T3:Tptr;
T^.item:=4;
new(T1);
T1^.item:=6;
T1^.Left:=nil;
T1^.Right:=nil;
T^.Left=T1;
```



روابط درخت:

n_0 = تعداد گره های برگ

n_1 = تعداد گره های تک فرزندی

n_2 = تعداد گره های دو فرزندی

$$n = n_0 + n_1 + n_2$$

$$\rightarrow B = n - 1 \quad \text{تعداد یال} \rightarrow B = n_1 + 2n_2 \rightarrow n_1 + 2n_2 + 1 = n \quad \rightarrow \quad n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$$

$$\rightarrow \quad n_0 = n_2 + 1$$

Perfect $\rightarrow$ تعداد کل گره $= n \rightarrow h = ?$

$$2^h - 1 = n \quad \rightarrow \quad h = \lceil \log_2(n+1) \rceil$$

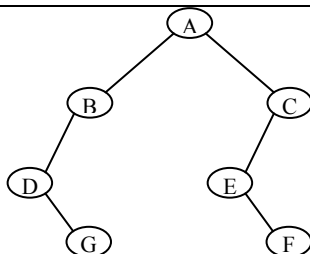
پیمایش درخت:

عمقی: (in) LVR ، (pre) VLR ، (post) LRV ، RLV ، VRL ، RVL

سطحی (Level) : **BFS** $\leftarrow$

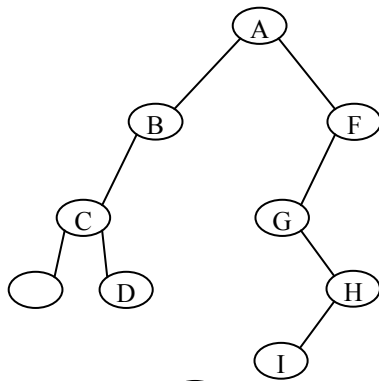
```

EnQ(Q,T);
while not IsEmpty(Q) do
Begin
    T1:=DeQ(Q);
    write(T1^.item);
    if T1^.left <> nil then
        EnQ(Q,T1^.Left);
    if T1^.Right <> nil then
        EnQ(Q,T1^.Right);
End;
    
```



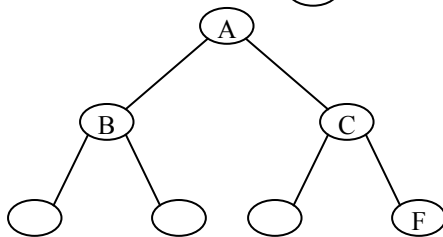
Inorder(LVR): DGBAEFC

Tree درخت



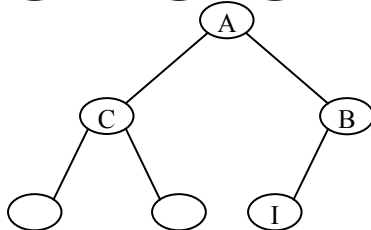
InOrder (LVR): ECDBAGIHF

(۲



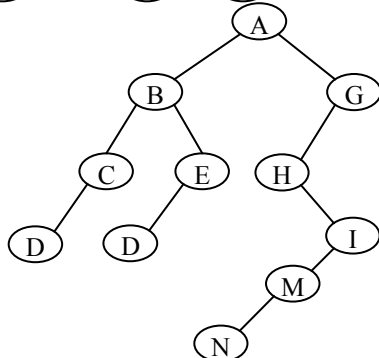
PreOrder (VLR) : ABEDCGF

(۳)



PostOrder (LVR) : DECFBA

(f)



LVR: DCBFEAHNMIG

LRV: DCFEBNMIHGA

VLR: ABCDEFGHIMN

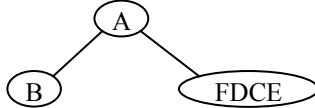
RLV: NMIHGFEDCBA

RVL: GIMNHAFCDB

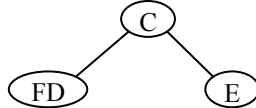
VRL: ABHIMNBEFCD

(5)

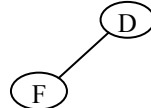
1) LRV: BFDECA



2) LRV: $FDE_{\underline{C}}$



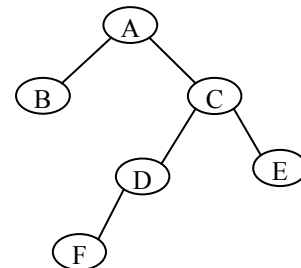
3) LRV: FD



IN(LVR): BAFDCE

Post(LRV): BFDECA

Graph $\rightarrow$?



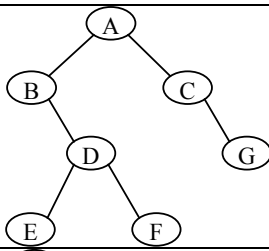
(۶) درخت را رسم کنید:

ساختمان داده ها

فصل هشتم

درخت Tree

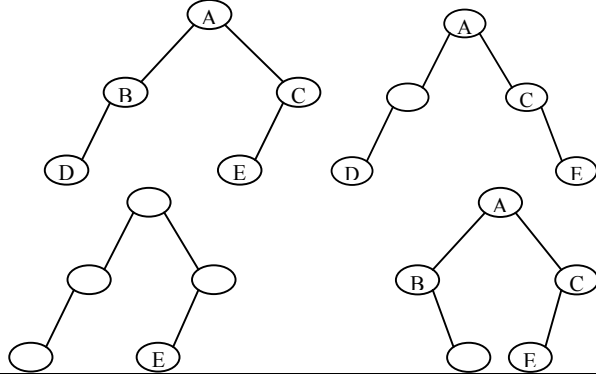
(۷) درخت را رسم کنید



IN: BEDFACG

Pre: ABDEF CG

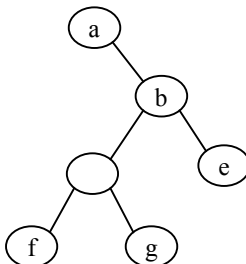
(۸) همه درخت های ممکن را رسم کنید:



Pre: ABDCE

Post: DBECA

(۹) پیمایش InOrder آرایه مقابل چیست؟



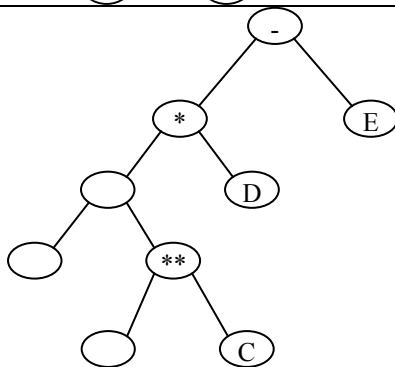
a		b		d	e		f	G	
---	--	---	--	---	---	--	---	---	--

فرزند چپ: r_i

afdgbe

فرزند راست: r_{i+1}

(۱۰) نتیجه پیمایش میانوندی درخت مقابل چیست؟



Posr: $ABC^{\wedge}+D^*E$

Pre: $-^*+A^{\wedge}BCDE$

In: $A+B^{\wedge}C^*D-E$

(۱۱) پیمایش post بصورت مقابل است. کدام گزینه

نمی تواند پیمایش pre باشد؟

1) ABDECF

2) ABCDEF

3) BADCEF

4) DBEFAC

DEBFCA

پیمایش بصورت بازگشتی:

Procedure InOrder(T:Tptr);

Begin

if $T \neq \text{nil}$ then

begin

InOrder($T^{\wedge}.\text{Left}$);

write($T^{\wedge}.\text{item}$);

InOrder($T^{\wedge}.\text{right}$);

end;

ساختمان داده ها

درخت Tree

فصل هشتم

end;

Procedure PreOrder(T:Tptr);

Begin

if T<>nil then

begin

write(T^.item);

PreOrder(T^.Left);

PreOrder(T^.right);

end;

end;

Procedure Traverse(T);

{

while T<>nil do

{

Traverse(Lchild(T));

visit(T);

T:=Rchild(T);

}

}

Procedure PostOrder(T:Tptr);

Begin

if T<>nil then

begin

PostOrder(T^.Left);

PostOrder(T^.right);

write(T^.item);

end;

end;

مثال ۱) الگوریتم مقابل کدام پیمایش را انجام میدهد؟

→ Inorder

Procedure Traverse(T)

Begin

P:=T;

F:=FALSE;

Repeat

while(P<>nil) do

Begin

push(P,s);

P:=Lchild(P);

End;

If not IsEmpty(S) then

Begin

pop(P,s);

write(data(P));

P:=Rchild(P);

End;

else

F:=TRUE;

Until F;

End;

→ LVR

مثال ۲) الگوریتم مقابل کدام روش پیمایش را انجام میدهد؟

Procedure tst(T)

مثال ۳)

Begin

push(s,T);

while(stack s is not empty)

begin

T:=pop(s);

while T<>nil do

begin

Push(s,T^.right);

visit(T);

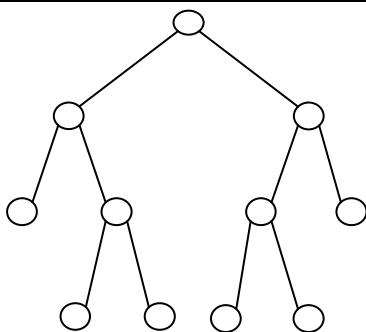
T:=T^.Left;

end;

end;

end;

→ VLR



مثال ۴) با توجه به درخت ، خروجی را مشخص کنید.

1) 12

2) 8

3) 10 ←

4) 6

Function k(T):Integer;

Begin

if T=nil then

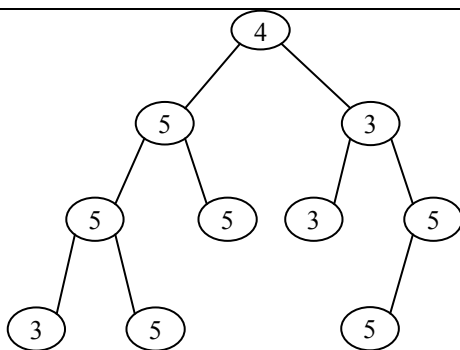
k:=0

else

k:=k(Rchild(Lchild(T)))

+3*k(Lchild(Rchild(T))+2;

End;



مثال ۵) این تابع چندبار مقدار ۵ را برمیگرداند؟

جواب: ۶ بار

function k(a):Byte;

Begin

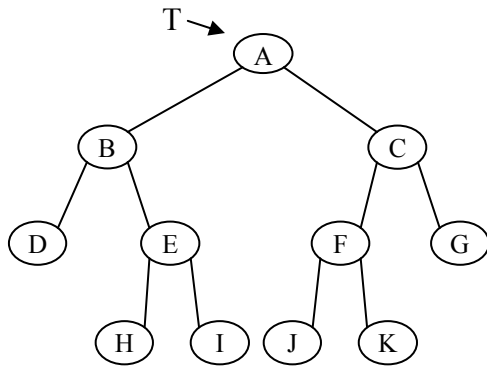
if a has weight then

k:=weight(a);

else

k:=k(parent(a));

End;



مثال ۶) مقدار بازگشتی تابعی زیر چند است؟

1) 4

2) 5

3) 6

4) 8

Function count(T);

Begin

if T<>nil then

count:=2count(Lchild(Rchild(T)));

+count(Rchild(Lchild(T)))+1;

else

count:=0;

End;

درخت عبارت جبری:

یکتایی ! ~ -

عملگر: دودویی / × - +

۱) تعداد عملگرهای دودویی همواره فرد است.

۲) تعداد عملگرهای دودویی همواره زوج است.

۳) تعداد عملگرهای یکتایی همواره زوج است

۴) تعداد عملگرهای یکتایی همواره فرد است →

مثال ۷) اگر تعداد کل گره های درخت ۱۴ باشد ، کدام

گزینه درست است؟

اثبات:

$$n = n_0 + n_1 + n_2$$

$$n_0 = n_2 + 1$$

در هر درخت تعداد برگ ها مستقل از تعداد گره های تک فرزندی است

$$14 = n_0 + n_1 + n_2$$

$$14 = n_2 + 1 + n_1 + n_2$$

$$13 = 2n_2 + n_1$$

فرد + زوج = فرد

مثال ۸) اگر $n_4 = 5$ و $n_3 = 6$ و $n_2 = 5$ آنگاه تعداد برگها چقدر است؟ ($n_0 = ?$)

$$n = n_4 + n_3 + n_2 + n_1 + n_0$$

$$B = n_1$$

$$B = n_1 + 2n_2 + 3n_3 + 4n_4$$

→

$$n = n_1 + 4n_2 + 3n_3 + 4n_4 + 1$$

$$\rightarrow n_0 = 3n_4 + 2n_3 + n_2 + 1$$

$$\rightarrow n_0 = 15 + 12 + 5 + 1 = 33$$

$$n_0 = n_2 + 2n_3 + 3n_4 + \dots + (k-1)n_k$$

درخت FULL:

هر گره: برگ - دو فرزندی

اگر تعداد برگ ها n باشد، تعداد گره ها $2n-1$ است. اگر تعداد گره های غیر برگ n باشد، تعداد گره ها $2n+1$ است.

مثال ۹) در حالت فوق اگر پیمایش Pre و برگ ها را داشته باشیم، کدام گزینه صحیح است؟

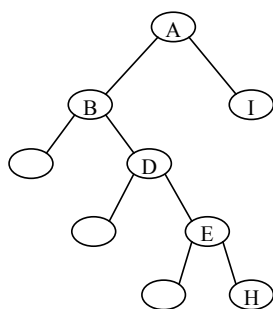
۱) میتوان درخت را ایجاد نمود ولی درخت حاصل واحد نیست.

۲) میتوان درخت را ایجاد نمود و درخت حاصل واحد است →

۳) نمی توان آنرا ایجاد نمود

۴) فقط اگر درخت بالانس باشد، میتوان آنرا ایجاد کرد.

مثال:



Pre: A B C D F E G H I

برگها: { C F I H G }

مثال ۱۰) عمق درخت عبارت جبری را بدست آورید: $h=6$

$$(-a)*b*c - d/e*g + h$$

اندازه درخت: تعداد گره های درخت.

```
Function TreeSize(T:Tptr):integer;
begin
    if T=nil then
        TreeSize:=0;
    else
        TreeSize= 1+TreeSize(T^. Left)+TreeSize(T^. Right);
    End;
```

تعیین تعداد گره های برگ:

```
function NumofLeaves(T:Tptr):integer;
Begin
    if T=nil then
        NumofLeaves:=0;
    else if (T^.Left=nil) and (T^.Right=nil) then
        NumofLeaves:=1;
    else
        NumofLeaves:=NumofLeaves(T^.Left)+NumofLeaves(T^.Right);
    End
```

آیا دو درخت مساوی هستند؟

```
function Cmp(Tptr T1,Tptr T2)
{
    if (T1==NULL && T2==NULL)
        return(1);
    else if (T1!=NULL && T2!=NULL && Data(T1)==Data(T2))
        return(Cmp(Left(T1),Left(T2) && Cmp(Right(T1),Right(T2)));
    else
        return(0);
}
```

تابع زیر چه مقداری را برمیگرداند؟ ارتفاع درخت

```
Function k(t:pointer):integer;
Begin
    if t=nil then
        k:=0;
    else
        k:=1 + max(k(T.Left),k(T.Right));
    End;
```

تابعی بنویسید که یک کپی از درخت T1 ساخته و اشاره گر T2 را به آن برگرداند.

```
Function CopyTree(T1:Tptr):Tptr;
Var
    T:Tptr;
```

```

Begin
  if T1<>nil then
    Begin
      new(T);
      T^.info = T1^.info;
      T^.left=CopyTree(T1^.Left);
      T^.Right=CopyTree(T1^.Right);
      CopyTree := T;
    End;
  else
    CopyTree:=nil;
End;

```

هر گره درخت :

- برگ

- k تا فرزند دارد

n: تعداد کل گره ها

- تعداد برگ: $n_0 = n - \left\lceil \frac{n-1}{k} \right\rceil$

- تعداد غیر برگ: $\left\lceil \frac{n-1}{k} \right\rceil$

مثال: $k=3$ $n_0 = 10 - \frac{9}{3} = 7$

اثبات:

$$n = n_0 + n_k, \quad B = n - 1, \quad B = kn_k \Rightarrow \frac{n-1}{k} = n_k \Rightarrow n_0 = n - \frac{n-1}{k}$$

۲) یک درخت ۳ تایی داریم (هر گره ۳ لینک دارد). اگر این درخت شامل ۱۰ گره باشد، چند اتصال null و چند غیر null دارد.

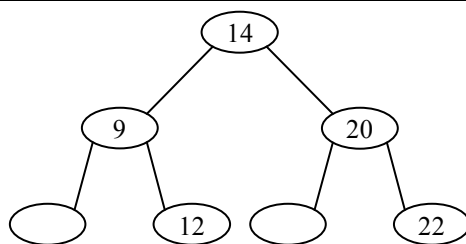
$$n-1 = \text{nil} = 9 \Rightarrow n \cdot 3 = 30 \Rightarrow \text{کل اشاره گرها} = 30 \Rightarrow \text{nil} = 30 - 9 = 21$$

نتیجه: n گره k تایی: کل اشاره گرها: nk، اشاره گرهای غیر nil: n-1، اشاره گرهای nil: nk-(n-1)

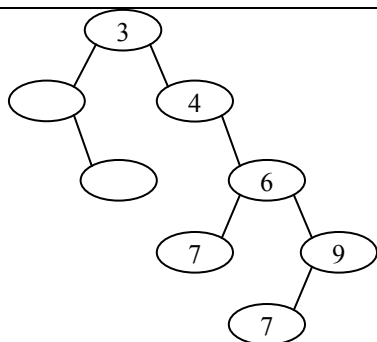
:(binary search tree) BST

درخت جستجوی دودویی:

مقدار گره ریشه از مقدار فرزند چپ بیشتر و از مقدار فرزند راست کمتر باشد:



مثال ۱) با اعداد مقابل یک درخت BST ایجاد کنید.



3,1,4,6,9,2,5,7

مثال ۲) با k کلید چند درخت BST میتوان ساخت؟



جواب: ۲ تا

با سه تا کلید چگونه؟ ۵ تا

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

رابطه مربوط به تعداد درخت های BST متفاوت که با n کلید میتوان ساخت؟

چند درخت ارتفاع n دارند؟ 2^{n-1}

اگر داده ها مرتب شده باشند ، ارتفاع همیشه n است.

حذف گره در BST:

- برگ (گره = nil)

- تک فرزندی: کافیت اشاره گری از پدر به فرزند متصل شود

- دو فرزندی: پیمایش inorder را نگاه میکنیم. گره ای که در inorder بعد از گره موردنظر آمده است

بجای عنصر حذف شده می نشیند که حتما یا برگ است یا تک فرزندی است.

LVR: 7 8 9 10 12 13 15 16 17 20 24 25 (نتیجه پیمایش به ترتیب صعودی است)

برای ترتیب نزولی می توان از RLV استفاده کرد.

مثال ۳) اگر اعداد یک تا هزار در یک درخت BST درج شده باشند و بدنبال عدد ۳۶۳ بگردیم ، کدامیک از

ترتیب های زیر نمی تواند بیانگر ترتیب دسترسی به عناصر درخت در این جستجو باشد؟

924,220,911,244,898,258,362,363(۱

→ 925,202,911,240,912,245,363 (۲

...

پیاده سازی:

Tptr: left , right , info

```
function find(X:integer; T:Tptr):Tptr;
Begin
    if T=nil then
        find:=nil
    else if T^.info>X then
        find:=find(T^.Left)
    else if T^.info<X then
        find:=find(T^.Right)
    else
        find:=T
End;
```

بصورت غیر بازگشتی:

```
While(T!=nil && T^.info!=x)
{
    if(T^.info>X)
        T=T^.Left;
    else(T^.info<X);
        T=T^.Right;
}
```

مثال ۱) تابعی بنویسید که مقدار min را بدست آورد

```
Tptr K(T)
{
    if (T==NULL) return NULL
    else if (T.Left==NULL)
        return T
    else Return (K(T.Left))
}
```

```
function findmin(T:Tptr):Tptr
Begin
    while(T^.Left!=nil)
        T=T^.Left;
    return T;
End
```

افزودن به درخت BST:

- ابتدا گره را جستجو می کنیم
- اضافه کردن در صورت وجود نداشتن گره
- اگر $q^.item < X$ سپس $q^.Left = temp$
- در غیر اینصورت $q^.Right = temp$

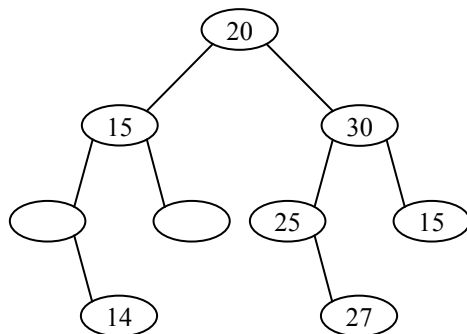
```
int Insert(Tptr T,int X)
```

```

{
    Tptr P= T, temp;
    while(T!=NULL && Data(T)!=X)
    {
        if(Data(T)>X)
        {
            P=T;
            T=Left(T);
        }
        else if(Data(T)<X)
        {
            P=T;
            T=Right(T);
        }
    }
    if (T==NULL)
    {
        temp=new Tptr;
        Data(temp)=X;
        Right(temp)=NULL;
        Left(temp)=NULL;
        if ( Data(p)<X )
            Right(P)=temp;
        else
            Left(P)=temp;
    }
    else
        پیغام خطا
}

```

تمرین ۱) در صورت حذف ریشه ، کدام گره می توان بجای ریشه قرار گیرد تا درخت BST باقی بماند (هدف پیدا کردن آدرس یکی از گره های درخت آمیاشد که اگر بجای ریشه درخت بنشیند ، درخت T یک درخت جستجوی دودویی باقی بماند) کدام یک از کدهای زیر این کار را انجام میدهد.



باید یا کوچکترین فرزند سمت راست و یا بزرگترین فرزند سمت

(۱)
 P:=Left(T);
 While right(p)!=nil do
 P:=Right(P);

(۲)
 P:=Right(T);
 While Left(P)!=nil do
 P=Left(P);

تمرین ۲) درج بصورت بازگشتی:

```

Procedure Insert(X:Integer; Var T:Tptr);
var
    Tptr *temp;
Begin
    if T=nil then
        Begin
            new(temp);
            temp^.info:=X;
            temp^.Left:=nil;
            temp^.Right:=nil;
            T:=temp;
        End;
    Else
        Begin
            if X<T^.info then
                Insert(X,T^.Left);
            else if X>T^.info then
                Insert(X,T^.right);
            Else
                پیغام خطا
        End;
    End;
End;

```

تابع حذف از BST:

```

Procedure Delete(X:Integer; Var T:Tptr);
Var
    temp:Tptr;
Begin
    if T=nil then
        Begin
            Error Halt
        End
    Else if X<T^.info then
        Delete(X,T^.Left)
    Else if X>T^.info then
        Delete(X,T^.Right)
    Else
        { //x=T^.info }
        Begin
            if T^.Left=nil then
                Begin
                    temp:=T;
                    T:=T^.Right;

```

```

        dispose (temp);
    End
    Else if T^.right=nil then
    Begin
        temp:=T;
        T:=T^.Left;
        dispose(temp);
    End;
Else                                     { // 2 child }
    Begin
        temp:=findmin(T^.Right);
        T^.info:=temp^.info;
        Delete(T^.info, T^.Right);
    End;
End;
End;
```

ساختمان داده ها

درخت Tree

فصل هشتم

اگر درخت BST کاملاً بالانس باشد ارتفاع آن $\log_2 n$ خواهد بود.

این درخت AVL نام دارد که در آن حداکثر اختلاف هر زیر درخت آبرابر یک است.

$$|h(L) - h(R)| \leq 1$$

حداقل گره مورد نیاز برای ایجاد درخت AVL به ارتفاع h :

$h=2$	$n=2$
$h=3$	$n=4$
$h=4$	$n=7$

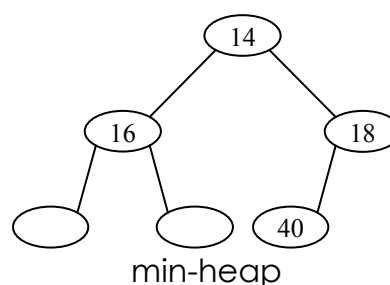
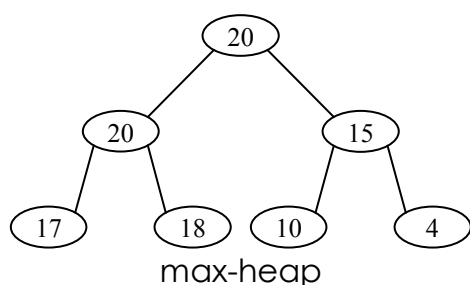
$$n(h) = n(h-1) + n(h-2) + 1 \quad (n(1)=1, n(2)=2)$$

هرم کپه (HEAP):

درخت دودویی کامل:

min-heap: اگر مقدار گره ریشه از مقدار فرزندان بزرگتر نباشد

max-heap: اگر مقدار گره ریشه از مقدار فرزندان کوچکتر نباشد



اگر تعداد کل گره ها n باشد: $h = \lfloor \log_2 n \rfloor + 1$

همیشه تعداد گره های برگ: $\lfloor \frac{n}{2} \rfloor + 1$

غیربرگ: $\frac{n}{2}$

جستجوی بزرگترین عنصر را باید در برگها انجام داد.

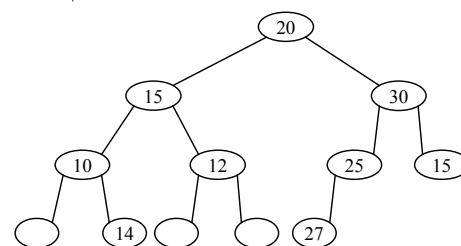
در بدترین حالت $n/2$ مقایسه انجام میشود. !!

1	2	3	4	5	6	7	8	9	10	11	12
1	6	4	10	15	7	9	17	20	16	30	10

ریشه: ۱

فرزند چپ گره i : $2i$ ، فرزند راست گره i : $2i+1$

پدر گره i : $i/2$

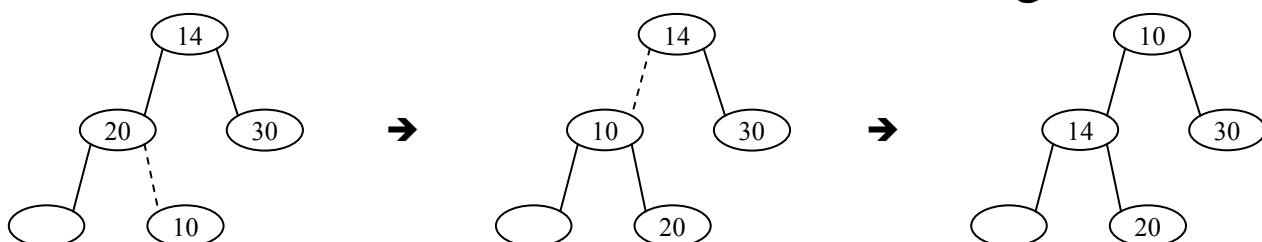


Insert -

Delete -

اضافه کردن:

گره جدید را به انتها اضافه می کنیم. در این حالت درخت کامل است اما از حالت heap خارج شده است. پس باید آنرا به مکان صحیح منتقل کرد...



پیچیدگی عمل اضافه کردن $\log_2 n$ است (همان ارتفاع درخت است)

پیچیدگی ایجاد درخت $n \log_2 n$ است (برای n عنصر)

سؤال ۱) با اضافه کردن عنصری با مقدار ۹۵ چند عمل جابجایی انجام میشود.

					۸(۴	۶(۳	۴(۲	۲(۱ ←	
1	2	3	4	5	6	7	8	9	10
100	90	82	85	74	75	73	68	70	

سؤال ۲) یک max heap شامل n عنصر که با آرایه پیاده سازی شده است داریم. مناسبترین گزینه برای پیدا کردن عنصر min کدام است.

۱) $O(\log n)$ ۲) حداکثر $n-1$ مقایسه

۳) ← حداکثر $n/2$ مقایسه ۴) هیچکدام

سؤال ۳) یک max-heap شامل n عنصر متمایز داریم. چهارمین بزرگترین عنصر در کدامیک از دایره های زیر می تواند باشد؟

۱) ۲ یا ۳ ۲) ۸ تا ۱۵ ۳) ۴ تا ۷ ۴) همه موارد →

Delete (حذف از heap):

همیشه ریشه حذف میشود. پس آخرین عنصر بجای آن قرار گرفته و درخت مجدداً بازسازی میشود.

مرتب سازی Heap:

ابتدا عناصر در یک heap درج شده و سپس به ترتیب حذف میشوند. (پیچیدگی $n \log n$)

صعودی: minheap نزولی: maxheap

ساختمان داده ها

درخت Tree

فصل هشتم

تمرین ۱) آرایه ای شامل n عنصر داریم که برای ذخیره سازی عناصر درخت دودویی کامل مورد استفاده واقع شده است. الگوریتمی بنویسید که بررسی کند درخت heap است یا نه؟ (پیچیدگی را بدست آورید)

تمرین ۲) کدام گزینه صحیح است؟

- ۱) حداکثر عمق درخت BST برابر $O(\log n)$ است
- ۲) حداکثر عمق درخت heap برابر $O(n)$ است →
- ۳) حداکثر عمق درخت BST برابر $O(n)$ است
- ۴) حداکثر عمق درخت Heap و BST برابر $O(\log n)$ است

تمرین ۳) یک درخت BST داریم. X : برگ $a = \text{value}(x)$ و Y : پدر گره x ، $b = \text{value}(y)$ کدام گزینه صحیح است؟

- ۱) b بزرگترین کلید در T است که کوچکتر از a می باشد.
- ۲) a کوچکترین کلید در T است که بزرگتر از b می باشد
- ۳) b کوچکترین کلید در T است که بزرگتر از a می باشد
- ۴) هیچکدام →

تمرین ۴) درخت را با توجه به الگوریتم داده شده پیمایش کنید:

Procedure tst(T)

Begin

if $T \neq \text{nil}$ then

Begin

write (Data(T))

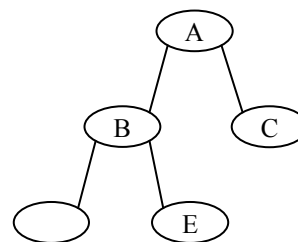
tst (Rchild(T))

write (Data(T))

tst (Lchild(T))

End

End



ACCABEEBDD

تمرین ۵) n عنصر با مقادیر مختلف داریم که میتوان با استفاده از یک درخت BST که در ابتدا تهی است آنها را مرتب کرد. الف) درج عناصر به ترتیب در درخت T ب) پیمایش به روش inorder پیچیدگی این الگوریتم را بدست آورید: **بهترین حالت n^2 بدترین حالت: $n \log n$**

ادامه مبحث Heap:

```

Procedure Insert (X:<Type>; Var H:Heap);
var
    i:Integer;
Begin
    Inc(H.size);
    i:=H.size;
    while H.Element[i Div 2] > x do           // تا زمانی که مقدار ریشه از فرزند بیشتر است
    begin                                     // باید گره فرزند را به بالا انتقال داد
        H.Element[i]:=H.Element[i Div 2];
        i:=i Div 2;
    end;
    H.Element[i]:=x;

```

End;

```

Function DeletMin (var H:Heap): <Type>;

```

```

var
    l,child: Integer;
    flag: Boolean;
    tmp, last : <type>;
Begin
    tmp:=H. Element[1];
    last := H.Element[H.size];
    Dec(H.size);
    i:=1;
    flag:=false;
    while (2*i < H.size) and (not flag) do
    begin
        child:=2*i;
        if child <> H.size then
            if H.Element[child+1] < H.Element[child] then
                child :=child +1;
            if lastElement>H.Element[child] then
                bgin
                    H.Element[i] := H.Element[child];
                    i:=child;
                end;
            else
                flag:=True;
        end; {while}
        H.Element[i]:=lastElement;

```

End;

مرتب سازی Heap:

در این روش ابتدا کلیه عناصر در یک min-heap یا max-heap درج میشوند. با توجه به اینکه بخواهیم عناصر را صعودی مرتب نماییم یا نزولی؟

مثلا جهت مرتب سازی صعودی کلیه عناصر را به ترتیب با n بار فراخوانی تابع Insert در یک min-heap درج می کنیم ، سپس n بار تابع Delete را فراخوانی می کنیم که در هر مرحله عنصر min را به ما میدهد.

- پیچیدگی عمل درج حداکثر $O(\log n)$ است
- پیچیدگی عمل حذف نیز $O(\log n)$ است
- ساخت heap از مرتبه $O(n \log n)$ است.

مرتب سازی Merge:

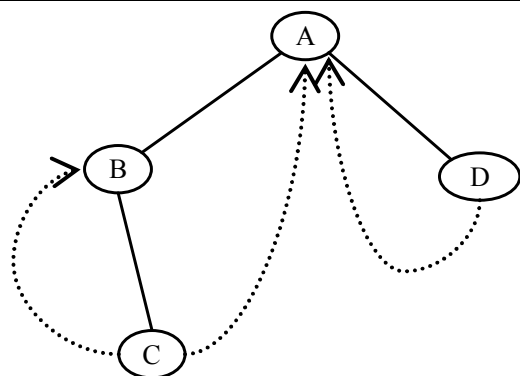
در این روش آرایه به دو تکه مساوی تقسیم شده سپس هر قسمت مرتب شده و نتایج مرتب سازیها با هم ادغام میشوند. در این حالت به یک آرایه کمکی با حداکثر اندازه n نیاز داریم.

```
mergeSort(A, L, U)
begin
    اگر تعداد عناصر کمتر از ۲ باشد
        return ;
    else
        mid := (L+U) Div 2;
        mergeSort(A,L,mid);
        mergeSort(A,mid+1,U);
        merge(A,L,mid,U);
end;
```

روتین merge آرایه های مرتب شده $A[L..mid]$ و $A[mid+1..U]$ را در هم ادغام می نماید. بگونه ای که پس از فراخوانی آن آرایه A از اندیس $L..U$ مرتب شده است

درخت نخه:

چون در یک درخت دودویی از میان $2n$ اشاره گر فقط $n-1$ اشاره گر آن مورد استفاده قرار میگیرد و عملا از بقیه استفاده نمی شود ، میتوان از این اشاره گر ها جهت ساده سازی در اعمال پیمایش ها استفاده نمود. مثلا میتوان جهت استفاده در پیمایش inorder استفاده نمود که در اینصورت اشار گر سمت راست اگر nil باشد کاری می کنیم که به گره بعدی در پیمایش inorder اشاره نماید و اگر اشاره گر چپ nil باشد کاری می کنیم که به گره قبلی در پیمایش inorder اشاره نماید.



InOrder: BCAD

جهت تشخیص اینکه آیا اشاره گر یک گره اصلی هستند یا اینکه اشاره گر نخی ، میتوانیم از دو متغیر اضافی در ساختار گره ها استفاده نماییم.

lf	left	info	right	rf
----	------	------	-------	----

left : rf=0 یک اشاره گر اصلی است

left : rf=1 یک اشاره گر نخی است