۲۹ مهر
کوئیز (۳ نمره) ← ۲۷ آبان
۹ دی

میان ترم (۱۵ نمره) ← ۲۵ آذر
پایان ترم (۶ نمره) ←
تمرین (۴ نمره) ← حل تمرین + تمرین های فروش
پروژه پایانی (۲ نمره)

۱ ← (۶,۲۸) :

مسئله را نشناسیم = مسئله را تحلیل نکنیم = یک مدل اشتباه از مسئله میسازیم

تحلیل مسئله ← ورودی های مسئله / خروجی های مسئله / روش حل مسئله

الگوریتم ← مجموعه ای از دستورالعمل ها که به صورت مرحله به مرحله و به زبان دقیق و بدون ابهام (بدون متناهی) می شوند و هدف خاصی که همان حل مسئله است را دارد.

* دستورالعمل ها خاتمه پذیر باشند.
* دستورالعمل شروع و پایان مشخص باشد.
* جامع باشد ← به ازای تمام ورودی های ممکن خروجی مطلوب را تولید بکند.
* مختصر باشد
* قابل اجرا باشد (تقسیم بر صفر قابل اجرا نیست)

ورودی ← مجموعه دستورالعمل ها ← خروجی

Subject:

Year. Month. Date. ( )

- تحلیل الگوریتم ها:
  - زمانی ← محاسبهٔ مدت زمان لازم برای اجرای الگوریتم
  - فضایی یا حافظه ای ← محاسبهٔ میزان حافظه مصرف شده در طول اجرای الگوریتم

راهی برای اینکه ما بتونیم بهترین هندسا الگوریتم که برای حل مسئله داده ارائه شده یکی را انتخاب کنیم.

A_1 ، A_2 → A_2

A_1: یک ساعت، 1TB

A_2: دو ماه، 1GB

A_3 ، A_4 → A_4

A_3: نیم ساعت، 1GB

A_4: یک ساعت، 2GB

* حافظه در دسترس است پس اولویت زمان کمتری دارد بهره.

* اگر مشکل کمبود حافظه نداشته باشیم در این صورت الگوریتمی بهره‌ور است که زمان کمتری مصرف میکند.

اما اگر الگوریتمی میزان حافظه مورد استفاده‌اش خیلی زیاد باشد باید کنارش بذاریم.

- عوامل ساده کننده: مدت زمان لازم برای انجام عملیات اساسی با هم برابر است t

- مدت زمان لازم برای اجرای الگوریتم: تعداد عملیات اساسی $\times t$

t را به حذف کرد.

- تحلیل زمان الگوریتم: مجموع تعداد عملیات اساسی انجام شده

طراحی الگوریتم → ایده یک حل → مسئله

↑ تحلیل

Year. Month. Date. ( )

۲ ← (۶,۳۰) ۵

سرعت اجرای برنامه ها یا مدت زمان لازم برای اجرای برنامه ها وابسته به زبان برنامه نویسی است + ماشین مورد استفاده

برای اینکه زمان اجرای الگوریتم ها را محاسبه کنیم تعداد دستور العمل ها یا عملیات اساسی که توی الگوریتم وارد انجام شده را بشماریم.

مدت زمان لازم برای انجام یک الگوریتم تا حدی متناسب با تعداد عملیات انجام شده توی الگوریتم است.

A1 A2 → برای حل یک مسئله ارائه شده → A2

100*t 50*t

? x=x+1 $T_1(n)=1$

? for (int i=0 ; i<n ; i++)
x=x+1 ;

$T_2(n) = \underbrace{2}_{i=0} + \underbrace{2+5+5+\dots+5}_{i=1,\dots,n-1} + \underbrace{2+1}_{i=n}$

(i=0, i<n, x=x+1 / i++, i<n, x=x+1 / i++ / i<n)

$T_2(n) = 7 + 5(n-1) = 5n+2$

? for (int i=0 ; i<n ; i++)
{ for (int j=0 ; j<n ; j++)
x=x+1 ; }

$T_2(n)$

$T_3(n) = \underbrace{2+(5n+2)}_{i=0} + \underbrace{3(5n+2)}_{i=1,\dots,n-1} + \underbrace{2+1}_{i=n}$

(i=0, i<n, داخل حلقه / i++, i<n, داخل حلقه)

$T_3(n) = 5n+7+(n-1)(5n+3) = 5n^2+3n+4$

Subject:

Year. Month. Date. ( )

? for(int i=0 ; i<n ; i++)
  for(int j=0 ; j<n ; j++)
    for(int k=0 ; k<n ; k++)
      x=x+1 ;

$T_3(n)$

$T_4(n) = \underbrace{2 + (5n^2+3n+4)}_{i=0} + \underbrace{3 + (5n^2+3n+4)}_{i=1,m,n-1} + \underbrace{\quad}_{i=n}$

$T_4(n) = 5 + T_3(n) + (n-i)(5n^2+3n+7)$

? for(int i=0 ; i<n ; i++)
  for(int j=0 ; j<i ; j++)
    x=x+1 ;

$j<i, j++, x=x+1$; $j<i, j++$

$\underbrace{4}_{j=0} + \underbrace{5}_{j=1,m,n-1} ; \underbrace{3}_{j=i} \longrightarrow 5i+2$

$T_5(n) = \underbrace{2 + (5i+2)}_{i=0} + \underbrace{3 + (5i+2)}_{i=1,m,n-1} + \underbrace{3}_{i=n} \Rightarrow 2+2+\sum_{i=1}^{n-1}(3+5i+2)+3$

$T_5(n) = 7 + \sum_{i=1}^{n-1}(5i+5) = 7 + 5\sum_{i=1}^{n-1} i + 5\sum_{i=1}^{n-1} 1 = 7 + \frac{5n(n-1)}{2} + 5(n-1$

$T_5(n) = \frac{5n^2}{2} + \frac{5}{2}n + 2$

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ $\quad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$ $\quad \sum_{i=1}^{n} i^3 = \frac{n^2(n+1)^2}{4}$

? for(int i=0 ; i<n ; i++)
  for(int j=0 ; j<i ; j++)
    for(int k=0 ; k<j ; k++)
      x=x+1 ;

Year Month Date ( )

```
int linearSearch(int A[100], n){
    for(int i=1; i<=n; i++)
        if (A[i]==x)
            return i;
    return -1;}
```

* حلقوی خطی:

زمان اجرای این الگوریتم تنها به n یعنی تعداد عناصر موجود در آرایه A بستگی دارد نکته به چیدمان عناصر در آرایه A هم وابسته است.

B(n), W(n), E(n) / حالت بدترین، بهترین، میانگین

در ۳ حالت مختلف باید الگوریتم را تحلیل کنیم:

$B(n) = 4$ → i=1, i<=n, if, return

$W(n) = 3 + 4 + 3 + 1 = 4n+3$

- 3: i=1, i<=n, if
- 4: i=2,...,n; i<=n, i++, if
- 3: i=n+1
- 1: return -1

return ؛ اضافه حساب نشد چون در بدترین حالت است و شرط آن برقرار نیست.

۳ ← (۷،۴) :

از سمت چپ و به صورت صعودی مرتب می‌شود.

مقایسه‌ها از آخر به اول! (راست به چپ)

key=3
key=2
key=10
key=12
key=1
key=5
key=6

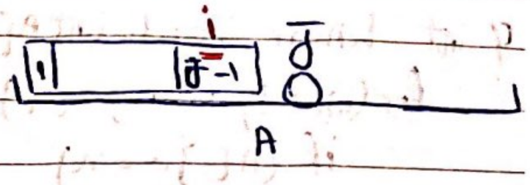
Insertion Sort = مرتب سازی درجی

4 3 2 10 12 1 5 6
3 4 2 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
2 3 4 10 12 1 5 6
1 2 3 4 10 12 5 6
1 2 3 4 5 10 12 6
1 2 3 4 5 6 10 12

Subject:

Year. Month. Date. ( )

```
void insertionSort (int A[1...n]) {
  for(int j=2; j<=n ; j++) {
      key = A[j] ;
        i = j-1 ;
   while (i>0 && A[i] > key) {
         A[i+1] = A[i];
            i-- ; }
   A[i+1] = key ; }}
```



key = A[j] ;

12 (10) 8 5 4 3 2	j [2]	n [7]	key [10] (A[j])	i [1] (j-1) → 0
10 12 (8) 5 4 3 2	j [3]		key [8]	i [2] → 1 → 0
8 10 12 (5) 4 3 2	j [4]		key [5]	i [3] → 2 → 1 → 0
5 8 10 12 (4) 3 2				

⋮

چون آرایه از اول نزولی بود عنصر کلید از تمام عناصر سمت چپ کوچک‌تر بود و باید جای قبل از عنصر قرار بگیرد و بدترین حالت را داریم ← بیشترین تعداد مقایسه

2 (3) 4 5 8 10 12	j [2]	n [7]	key [3]	i [1]
2 3 (4) 5 8 10 12	j [3]		key [4]	i [2]

فقط مقایسه کلید با عنصر قبلی است صورت می‌گیرد. در این مثال حلقه while (0) بار اجرا شود و تعداد مقایسه‌های صورت گرفته برابر با 1 است و کمترین حالت است.

j=2, j<=n, key, i ②, i>0, &&, A[i]>key, A[i+1]=key ②

$$B(n) = \underbrace{10}_{j=2} + \underbrace{11}_{j=3,\dots,n} = 10 + \sum_{j=3}^{n} 11 = 10 + 11(n-2) = 11n - 12$$

j<=n, j++ ②, key, i ②, شرط حلقه, A[i+1]=key ②

$$W(n) = \underbrace{15}_{j=2} + \sum_{j=3}^{n} \left(6 + (j-1) \times 7 + 1 + 2\right)$$

i>0, &&, A[i]>key, A[i+1]=A[i] ②, i-- ②

$$\frac{7}{2}n^2 + \frac{19}{2}n - 6$$

: (۷،۶) ← ۴

دوتا دوتا آرایه را مقایسه کرد.

*bubbleSort = مرتب‌سازی حبابی

8 2 4 9 3 6	2 4 8 3 6 [9]
2 8 4 9 3 6	2 4 8 3 6 [9]
2 4 8 9 3 6	2 4 8 3 6 [9]
2 4 8 3 9 6	2 4 3 8 6 [9]
2 4 8 3 6 9	2 4 3 6 [8] [9]

در هر مرحله بزرگ‌ترین عنصر را به آخر آرایه می‌برد.

```
? void BubbleSort (int A[1..n]) {
   for (int i=n-1; i>=2; i--)
     for (j=1; j<=i; j++)
       if (A[j] > A[j+1]) {
          temp = A[j];
          A[j] = A[j+1];
          A[j+1] = temp; }}
```

$$\sum_{i=m}^{n} i = (n-m+1)\left(\frac{m+n}{2}\right)$$

درونی $B(n) = \underbrace{4}_{j=1} + \sum_{j=2}^{i} 5 + 3 = 5i + 2$ وقتی آرایه از ابتدا صعودی باشد.

($j=1, j<=i, if$ ②; $j++, j<=i$; $j++$, $A[j]>A[j+1]$, $j<=i$ ②)

کل $B(n) = \underbrace{2 + 5(n-1) + 2}_{i=n-1} + \sum_{i=2}^{n-2} (5i+2) + 3 = 5n + 2 + \sum_{i=2}^{n-2} (5i+5)$

($5n, 5$; $3+$; $i--, i>=2$)

صفحه‌ای برحسب n از درجه ۲

$5n + 2 + 5\frac{(n-2)(n-1)}{2} - 1 + 5(n-3)$

درونی $W(n) = \underbrace{9}_{j=1} + \sum_{j=2}^{i} 10 + 3 = 12 + 10(i-1) = 10i + 2$

($j=1, j<=i, if$ ②, ⑤; $j++, j<=i$; $j++$)

کل $W(n) = \underbrace{2 + 10(n-1) + 2}_{i=n-1} + \sum_{i=1}^{n-2} 3 + (10i+2) + 3 = 10n - 3 + \sum_{i=2}^{n-2} (10i+5)$

($i--, i>=2$)

Subject:

Year. Month. Date. ( )

پیچیدگی زمانی الگوریتم‌ها:

- اگر یک کران بالا برای زمان اجرای الگوریتم بدست بیاریم منظم الگوریتم در بدترین حالت به اندازه‌ی کران بالا زمان اجرا احتیاج داره.

- و اگر یک کران پایین برای زمان اجرای الگوریتم بدست بیاریم منظم الگوریتم ما در بهترین حالت کمترین زمانی که احتیاج داره به اندازه‌ی کران پایینه.

$f(n) \in O(g(n))$: $f(n)$ متعلق است به مجموعه‌ی توابعی که $f(n) = Og(n)$ از مرتبه‌ی O بزرگ $g(n)$

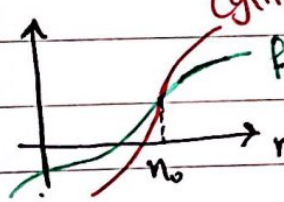
$g(n)$ یک کران بالا برای $f(n)$ اگر و فقط اگر

$f(n) \in O(g(n)) \leftrightarrow \exists c>0, n_0 \rightsquigarrow \forall n \geq n_0 \Rightarrow f(n) \leq cg(n)$

n_0 ای به قدر کافی بزرگ

n ای به قدر کافی بزرگ

* کران بالای مجانبی



* خاصیت تقارنی؟

$f(n) = n^2$? $\rightarrow f(n) \in O(g(n))$?

$g(n) = n^2 + 10n$

$\exists c>0, n_0 \rightsquigarrow \forall n \geq n_0 \quad 0 \leq f(n) \leq cg(n)$

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

$f(n) = n^2 + 10n$? $\rightarrow f(n) \in O(g(n))$?

$g(n) = n^2$

$c=2, n_0=10 \rightarrow 0 \leq n^2 + 10n \leq 2n^2$

$f(n) = n^2$? $\rightarrow f(n) \in O(g(n))$?

$g(n) = n^3$

$c=2, n_0=1 \rightarrow 0 \leq n^2 \leq n^3$

$g(n) \in O(f(n))$? ✗

$f(n) \in O(g(n)) \not\rightarrow g(n) \in O(f(n))$

* در حالت کلی O خاصیت تقارنی ندارد.

ω ← (۷,۱۱) : ω

* خاصیت انعکاسی:

$f(n) \in O(f(n))$?

$\exists c>0, n_0 \; / \; \forall n \geq n_0 \; 0 \leq f(n) \leq cf(n)$

کافیه $n_0 = 1$, $c = 1$ باشد.

* تابع مجانبی O دارای خاصیت انعکاسی است.

* خاصیت تعدی:

$f(n) \in O(g(n))$ ① , $g(n) \in O(h(n))$ ② $\xrightarrow{?}$ $f(n) \in O(h(n))$?

1: $\exists c_1>0, n_1>0 \; / \; \forall n \geq n_1 \; 0 \leq f(n) \leq cg(n)$

2: $\exists c_2>0, n_2>0 \; / \; \forall n \geq n_2 \; 0 \leq g(n) \leq c_2 h(n)$

1×2 : $\forall n \geq \max\{n_1, n_2\} \rightarrow 0 \leq f(n) \leq c_1 c_2 h(n)$

* تابع مجانبی O دارای خاصیت تعدی است.

$f(n) = f_1(n) + f_2(n) + \dots + f_m(n)$ / $f_i(n) \in O(g_i(n))$ $1 \leq i \leq m \rightarrow f(n) \in O(\;?\;)$

$\max_{1 \leq i \leq m} \{g_i(n)\}$

$\exists n_1, c_1>0 \; / \; \forall n \geq n_1 \; 0 \leq f_1(n) \leq cg_1(n)$

$\exists n_2, c_2>0 \; / \; \forall n \geq n_2 \; 0 \leq f_2(n) \leq c_2 g_2(n)$

$\exists n_i, c_i>0 \; / \; \forall n \geq n_i \; 0 \leq f_i(n) \leq c_i g_i(n)$

$\exists n_m, c_m>0 \; / \; \forall n \geq n_m \; 0 \leq f_m(n) \leq c_m g_m(n)$

$\forall n \geq \max\{n_1, n_2, \dots, n_m\} \; f_1(n) + f_2(n) + \dots + f_m(n) \leq c_1 g_1(n) + c_2 g_2(n) + \dots + c_m g_m(n) \leq$

$\max_{1 \leq i \leq m} \{c_i\} \max\{g_i(n)\} + \max\{c_i\} \max\{g_i(n)\} + \dots = \underbrace{m \max\{c_i\}}_{c} \max\{g_i(n)\}$

NEGAR

5 $\rightarrow$ $f(n) \leq \max_{1 \leq i \leq n} \{g_i(n)\}$

Subject:

Year. Month. Date. ( )

$\forall n \geq ?,\ \max\{n_1, n_2, \dots, n_k\} \rightsquigarrow 0 \leq f(n) \leq C \max\{g_1(n), \dots, g_m(n)\}$

$\longrightarrow f(n) \in O(\max_{1 \leq i \leq m} \{g_i(n)\})$

$? f(n) = n^2 + n^3 \longrightarrow f(n) \in O(n^3)$

$f(n) \in O(?)$

* قضیه اصلی:

$A(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_0$ یک چندجمله‌ای برحسب n از درجه m است.

$a_m > 0$ ↓

$A(n) \in O(n^m)$

* هم بدست آوردن کوچک‌ترین کران بالا است.

$f(n) = f_1(n) * f_2(n)$ / $f_1(n) \in O(g_1(n)),\ f_2(n) \in O(g_2(n)) \longrightarrow f(n) \in O(?)$

$? \leftarrow g_1(n) * g_2(n)$

$\exists n_1, C_1 > 0 \rightsquigarrow \forall n \geq n_1 \Rightarrow 0 \leq f_1(n) \leq C_1 g_1(n)$

$\exists n_2, C_2 > 0 \rightsquigarrow \forall n \geq n_2 \Rightarrow 0 \leq f_2(n) \leq C_2 g_2(n)$

$\forall n \geq \underbrace{\max\{n_1, n_2\}}_{n_0} \Rightarrow 0 \leq \underbrace{f_1(n) f_2(n)}_{f(n)} \leq \underbrace{C_1 C_2}_{C} g_1(n) g_2(n)$

$? f(n) = (4n^2 + 5n)(n \log n) \longrightarrow f(n) \in O(n^3 \log n)$

$4n^2 + 5n \in O(n^2)$, $n \log n \in O(n \log n)$

* ادامه مبحث O و تحلیل زمانی الگوریتم:

لازم نیست تک به تک عملیات اساسی را بشماریم و یک کران بالای مجانبی کافی است اما مهم کوچک‌ترین کران بالا است (پیدا کردن)

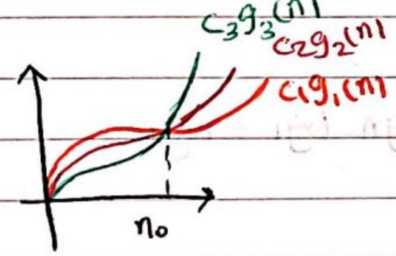
$T(n) \in O(g_1(n))$

* اثر ضرایب ثابت در اثر افزایش سرعت پردازنده قابل چشم‌پوشی است

A'_1 A'_2

n_2 $1000^{100} n_2$ → زمان اجرا تقریباً یکسان است.

* وقتی داریم الگوریتم‌های مختلف را از لحاظ زمان اجرا با هم مقایسه می‌کنیم n های کوچک (سایز ورودی کوچک) مهم نیست. کران بالای مجانبی مهم است و مهم که به ازای n های به قدر کافی بزرگ چه اتفاقی می‌افتد.



- مقایسه عملکرد زمانی برای n های کوچک ($n_0 > n$):

$c_1g_1(n) > c_2g_2(n) > c_3g_3(n)$

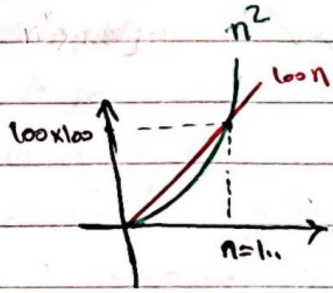
- مقایسه عملکرد زمانی برای n های بزرگ ($n_0 < n$):

$c_3g_3(n) > c_2g_2(n) > c_1g_1(n)$

* هر کدام پایین‌تر باشد بهتر است ← کوچک‌ترین کران بالای مجانبی

? $A_1 \to T_1(n) \in O(n^2)$

$A_2 \to T_2(n) \in (n \log n) \to$ ✓



? $A_1 \to T_1(n) \in O(100n) \to$ ✓

$A_2 \to T_2(n) \in O(n^2)$

? $T(n) = 100 \in O(?) \to O(1)$

* وقتی زمان اجرای الگوریتم یک عدد باشد یعنی مستقل از n و ربطی به سایز ورودی نداشته و به ازای هر سایز ورودی این الگوریتم 100 واحد زمانی مصرف می‌کند. الگوریتم در زمان ثابت اجرا شده یعنی به ازای هر سایز ورودی مستقل از اندازه ورودی مدت زمان لازم برای اجرا یک مقدار ثابت است.

$\forall n \geq n_0 \quad 100 \leq c_1 * 1$

$c_1 \downarrow 100$

Subject:

Year. Month. Date. ( )

؟ $\sum_{i=1}^{n} i \in O(?) \rightarrow O(n^2)$

$\sum_{i=1}^{n} i \rightarrow 1+2+\dots+n \leq n+n+\dots+n$ (n تا) $= n*n = n^2$

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

؟ $\sum_{i=1}^{n} i^2 \in O(?) \rightarrow O(n^3)$

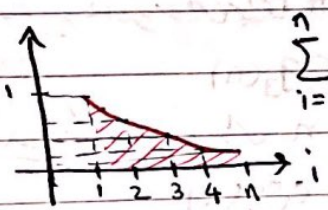
$\sum_{i=1}^{n} i^2 \rightarrow i^2+2^2+\dots+n^2 \leq n^2+n^2+\dots+n^2$ (n تا) $= n*n^2 = n^3$

$\sum_{i=1}^{n} i^2 \in \frac{n(n+1)(2n+1)}{6} \in O(n^3)$

؟ $\log n! \in O(?) \rightarrow O(n \log n)$

$\log n! = \log 1*2*3*\dots*n = \log 1 + \log 2 + \dots + \log n \leq \log n + \log n + \dots + \log n$ (n تا) $= n \log n$

؟ $\sum_{i=1}^{n} \frac{1}{i} \in O(?) \rightarrow O(\log n)$



$\sum_{i=1}^{n} \frac{1}{i} \leq \int_{1}^{n} \frac{1}{i} di = \log i \Big|_1^n = \log n - \log 1 = \log n$

$O(\log n) \rightarrow$ ۶

؟ $A_1 \rightarrow T_1(n) = 100n \rightarrow$ ✓

$A_2 \rightarrow T_2(n) = 2^n$

$n \leq 9 \rightarrow T_1(n) > T_2(n)$

$n=9 \rightarrow T_1(n) = 900 > T_2(n) = 512$

$n=10 \rightarrow T_1(n) = 1000 < T_2(n) = 1024$

$n \geq 10 \rightarrow T_1(n) < T_2(n)$

$T_1(100) = 10^4\ s \rightarrow$ حدود ضربت

$T_2(100) = 2^{100} = 2^{10 \times 10} = 10^{30} = 10^{20} * 10^{10} \rightarrow$

10^{20} سال طول میکشد تا اجرا شود ☹

10^{10} ثانیه $= 10^{10}$ (s)

* هدف ما بررسی زمان اجرا برای n های با اندازه بزرگ

7. $A_1 \rightarrow T_1(n) = 100n$ — در بررسی زمان اجرای الگوریتم‌ها ضریب ثابت اصلاً اهمیتی ندارد.

$A_2 \rightarrow T_2(n) = 100^{100} n$ — زیرا با افزایش سرعت پردازنده می‌توانیم اثر ضریب ثابت را از بین ببریم

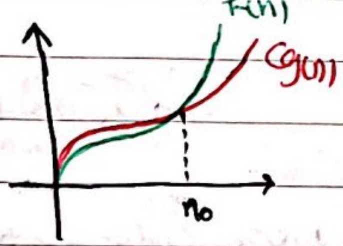
بطوریکه زمان اجرای A_1 و A_2 برابر شود.

* کران پایین مجانبی

Ω (امگا بزرگ): نماد مجانبی Ω ـ نماد مجانبی بهترین حالت

$f(n) \in \Omega\, g(n) \leftrightarrow \exists\, c>0,\ n_0>0$ که $\forall n \geq n_0 \quad 0 \leq c\,g(n) \leq f(n)$

یعنی $g(n)$ یک کران پایین برای $f(n)$.



رفتار تابع به ازای مقادیر کوچک n اصلاً مهم نیست و رفتار توابع به ازای nهای به قدر کافی بزرگ مهم است.

* پیدا کردن بزرگترین کران پایین مهم است.

$T(n) \in \Omega\, f(n)$ — یعنی $f(n)$ یک کران پایین مجانبی برای تابع $T(n)$ محسوب می‌شود.

پیچیدگی زمانی الگوریتم در بهترین حالت از مرتبه $f(n)$ است چون داریم کران پایین را پیدا می‌کنیم یعنی کمترین زمان لازم برای اجرای الگوریتم.

$T(n) \in \Omega\, f(n)$ زمان اجرای الگوریتم در بهترین حالت

$T(n) \in O\, g(n)$ " " " در بدترین حالت

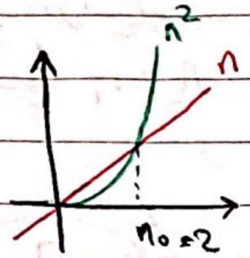
تحلیل زمانی الگوریتم: زمان اجرای الگوریتم‌ها را به طور دقیق محاسبه کنیم از طریق شمارش دستورالعمل‌ها.

پیچیدگی زمانی الگوریتم: نیازی نیست زمان را دقیق حساب کنیم و همه عملیات‌ها را بشماریم و کافیست از نمادهای مجانبی استفاده کنیم.

Subject:
Year. Month. Date. ( )

$? \; n^2 \in ?(n) \rightarrow \Omega$

$\exists c, n_0 > 0$ که $\forall n \geq n_0 \quad 0 \leq cn \leq n^2$ $\quad c=1, n=2$



$* \; n \in O(n^2)$

Ω ← چیزی که داخلش قرار گیرد کران پایین

O ← " " " " کران بالا

$* \; f(n) \in \Omega\, g(n) \leftrightarrow g(n) \in O\, f(n)$

* خاصیت تقارنی:

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

$n \notin \Omega(n^2)$

* Ω خاصیت تقارنی ندارد.

$f(n) \in \Omega\, g(n) \not\rightarrow g(n) \in \Omega\, f(n)$

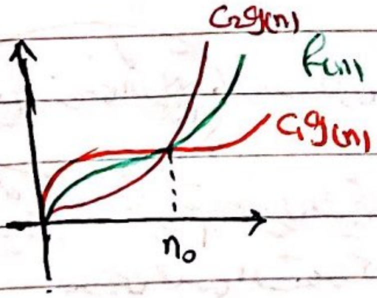
* قضیه اصلی:

اگر $A(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_0$ و $a_m > 0$ ← $A(n) \in \Omega(n^m)$

$A(n) \in O(n^m) \rightarrow A(n) \in \Theta(n^m)$

Θ: هم مرتبه بودن

$f(n) \in \Theta\, g(n) \leftrightarrow \exists c_1, c_2 > 0, n_0 > 0$ که $\forall n \geq n_0 \quad 0 \leq c_2 g(n) \leq f(n) \leq c\, g(n)$



g(n) هم یک کران پایینی و هم یک کران بالایی برای f(n) است.

یعنی: سرعت رشد f(n) و g(n) با هم یکسان است یعنی مقدار افزایش در تابع f و در تابع g یکسان است.

f(n) بین ضریبی از g(n) محصور شده.

$f(n) \in \Theta\, g(n)$ → سرعت رشد و نرخ رشد f و g یکسان و هم مرتبه است.

به ازای افزایش در مقدار n

مقدار ثابت = $\dfrac{\text{مقدار افزایش در } f}{\text{مقدار افزایش در } g}$

Year. Month. Date. ( )

* خاصیت تقارن:

$f(n)=n^2$, $g(n)=2n$ $\quad n\in\Theta(2n)$, $2n\in\Theta(n)$

$f(n)\in\Theta g(n) \leftrightarrow g(n)\in\Theta f(n)$ * Θ خاصیت تقارن دارد.

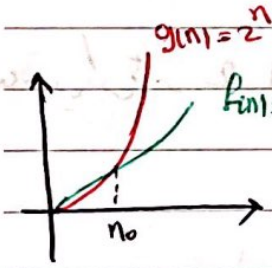
$f(n)\in O g(n) \leftrightarrow \forall c>0\ \exists n_0>0$ ک $\forall n\geq n_0$ $0\leq f(n)\leq cg(n)$: اثبات O

$0\leq \frac{f(n)}{g(n)}\leq c$

$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$

$f(n)$ از مرتبه O $g(n)$ است $\quad$ $g(n)$ سریعتر از $f(n)$ رشد می‌کند

یعنی سرعت رشد $g(n)$ خیلی بیشتر از سرعت رشد $f(n)$ است.



$f(n)\in o g(n)$ ← متناظر نزدیک شدن $f(n)$ به $g(n)$ خیلی بیشتر است.

O خیلی ضعیف‌تر از o است.

* $f(n)\in o g(n) \rightarrow f(n)\in O g(n)$

* $f(n)\in O g(n) \not\rightarrow f(n)\in o g(n)$

؟ $n^2\in O(n^2+10) \rightarrow n^2\notin o(n^2+10)$ $\quad$ $\lim_{n\to\infty}\frac{n^2}{n^2+10}=1\neq 0$

* اگر هم مرتبه با $f(n)$ متعلق به O $f(n)$ هم باشد آن‌گاه متعلق به O نیز می‌شود.

* خاصیت تعدی: $\Theta, O, \Omega, o, \omega$

* خاصیت انعکاسی: Θ, O, Ω

* تقارن: Θ

Subject:

Year. Month. Date. ( )

۷ـ (۷/۲۰):

: ω (برعکس o کوچک)

$f(n) \in \omega\, g(n) \leftrightarrow \forall c>0\ \exists n_0>0$ تا $\forall n \geq n_0 \quad 0 < c\,g(n) < f(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{\infty}{\text{عدد}} = \infty \qquad f(n) \in \omega\, g(n) \leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

$* f(n) \in o\, g(n) \longleftrightarrow g(n) \in \omega(f(n))$

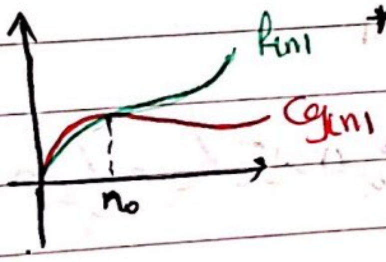
$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \longleftrightarrow \lim \frac{g(n)}{f(n)} = \infty$

↑ سرعت رشد

*ω خیلی قوی‌تر Ω است.

$* f(n) \in \omega\, g(n) \longrightarrow f(n) \in \Omega\, g(n)$

$* f(n) \in \Omega\, g(n) \not\longrightarrow f(n) \in \omega\, g(n)$



$? \; f(n) = n^2+1, \quad g(n) = n^2 \longrightarrow f(n) \in \Omega\, g(n)$ ✓

$f(n) \in \omega\, g(n)$ ✗ $\rightarrow \lim_{n\to\infty} \frac{n^2+1}{n^2} = 1 \neq \infty$

θ تعریف O بزرگ، Ω را هم دارد.

$* f(n) \in ?\ g(n)$

یک عدد ثابت : $f(n) \in \Theta\, g(n)$ یا $g(n) \in \Theta f(n)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)}$ →

0 : $f(n) \in o\, g(n)$, $f(n) \in O\, g(n)$

∞ : $f(n) \in \omega\, g(n)$, $f(n) \in \Omega\, g(n)$

Year. Month. Date. ( )

؟ $a^n \in$ ؟ (b^n) $\quad a,b>0$ ~ $0<a<b$

$0<\frac{a}{b}<1$

$\lim_{n\to\infty} \frac{a^n}{b^n} = \lim (a/b)^n = 0$

$0<1$ ✓ سرعت رشد صورت ↑

؟ $\log n \in$ ؟ (n)

$0<1$ ✓

$\lim_{n\to\infty} \frac{\log n}{n} = \frac{\infty}{\infty} \xrightarrow[(1)']{hop} \frac{\frac{1}{n \ln 2}}{1} = \lim_{n\to\infty} \frac{1}{n(\ln 2)} = 0$ سرعت رشد صورت ↑

؟ $\log_a^n \in$ ؟ $(\log_b^n)$

$\lim \frac{\log_a^n}{\log_b^n} = \frac{\infty}{\infty} \xrightarrow{hop} \frac{\frac{1}{n(\ln a)}}{\frac{1}{n(\ln b)}} = \lim \frac{\ln b}{\ln a} =$ عدد ثابت $\Rightarrow \Theta$

؟ $6*2^n + n^2 \in$ ؟ (2^n)

$\lim \frac{6*2^n + n^2}{2^n} = \frac{\infty}{\infty} \xrightarrow{hop} \frac{6*(\ln 2)2^n + 2n}{(\ln 2)*2^n} = \frac{6*(\ln 2)^2 2^n + 2}{(\ln 2)^2 2^n} =$

$\frac{6(\ln 2)^3 2^n}{(\ln 2)^3 2^n} = 6 =$ عدد ثابت $\Rightarrow \Theta$

$*(a^n)' = (\ln a)a^n \qquad *(\log_a^n)' = \frac{1}{n(\ln a)}$

؟ $x++;$ $\qquad T_1(n) \in O(1)$

؟ $for(int\ i=1;\ i<n;\ i++) \rightarrow O(n) \qquad T_2(n) \in O(n)$

$x++; \rightarrow O(1)$

تعداد دفعات تکرار حلقه‌ی for به ازای هر n برحسب n

$\in O(n)$ ← درجه‌ی 1

Subject:

Year. Month. Date. ( )

$f(n)$ → for تابع مربوط به تعداد دفعات تکرار حلقه

$g(n)$ → for تابع مربوط به زمان اجرای داخل دستورالعمل های

$T_2(n) = f(n) * g(n) \in O(n) * O(1) \in O(n*1) = O(n)$

1: $f_1(n) \in O\, g_1(n)$

2: $f_2(n) \in O\, g_2(n)$

$h(n) = f_1(n) * f_2(n) \longrightarrow h(n) \in O(g_1(n)\, g_2(n))$

$h(n) \in O(g_1(n) * g_2(n)) \overset{?}{=} O(g_1(n) * g_2(n))$

1: $\exists c_1 > 0, n_1 > 0 \;\ni\; \forall n \geq n_1 \quad 0 \leq f_1(n) \leq c_1 g_1(n)$

2: $\exists c_2 > 0, n_2 > 0 \;\ni\; \forall n \geq n_2 \quad 0 \leq f_2(n) \leq c_2 g_2(n)$

1x2:

$\forall n \geq \max\{n_1, n_2\} \longrightarrow 0 \leq f_1(n) f_2(n) \leq c_1 c_2 g_1(n) g_2(n) \longrightarrow h(n) \in O\, g_1(n) g_2(n)$

$O(g_1(n)) * O(g_2(n)) \in O(g_1(n) g_2(n))$.

؟

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

$T_2(n) \in O(n)$

$T_3(n) = a(n) * O(n) \in O(n^2)$

$h(n) =$ تعداد دفعات تکرار حلقه بیرونی $\longrightarrow O(n)$

؟

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

$f(i)$

$f_i \in O(i)$, $i \in O(n) \longrightarrow f_i \in O(a(n)) \in O(n)$

$T_4(n) \in O(n^2)$

Year: Month. Date. ( )

* $o(o(f(n))) \in o(f(n))$

?
```
for(int i=1; i<=n; i++)
    for(int j=1; j<=n; j++)
        for(int k=1; k<=n; k++)
            x++;
```
$T_3(n) \in O(n^2)$

$T_5(n) \in O(n^2) * O(n) \in O(n^3)$

?
```
for(int i=1; i<=n; i++)        --> O(n)
    for(int j=1; j<=i; j++)    --> O(n)
        for(int k=1; k<=j; k++) --> T4(n) ∈ O(n)
            x++;
```
$T_6(n) \in O(n^3)$

؟(۷،۲۵) ← ۸

$f(n) \in O(g(n)) \simeq a \leq b$ $\quad$ $f(n) \in o(g(n)) \simeq a < b$

$f(n) \in \Omega(g(n)) \simeq a \geq b$ $\quad$ $f(n) \in \omega(g(n)) \simeq a > b$

$f(n) \in \Theta(g(n)) \simeq a = b$

ترتیب رشد توابع : وقتی میگه n افزایش پیدا کنه میدونی کدوم تابع چقدر افزایش پیدا کنه.

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(4^n) < O(n!) < O(n^n)$

? $f(n) = \begin{cases} 2^n & n \leq 10^{10} \\ 100n & n > 10^{10} \end{cases}$, $g(n) = 2n^2 + 2$ *

$f(n) \in O(g(n))$? $\Leftrightarrow$ $g(n) \in O(f(n))$?

NEGAR

Subject:

Year. Month. Date. ( )

* مرتب سازی درجی:

```
void insertion_sort(int A[1000n]){
    for(int j=2; j<=n; j++){          → O(n)   Ω(n)
        key = A[j];                   → O(1)   Ω(1)
        i = j-1;                      → O(1)   Ω(1)
        while(i>1 && A[i]>key){       } O(i) = O(O(n)) = O(n)   } Ω(1)
            A[i+1] = A[i];            }
            i = i-1; }                }
        A[i+1] = key; }               → O(1)   Ω(1)
```

$O(n) * [O(1) + O(1) + O(n) + O(1)] = O(n^2)$ $T_n \in O(n^2)$ بدترین حالت

$O(n)$

$\Omega(n) * [\Omega(1) + \Omega(1) + \Omega(1) + \Omega(1)] = \Omega(n)$ $T_n \in \Omega(n)$ بهترین حالت

$\Omega(1) = \Omega(1)$

* مرتب سازی حبابی:

```
void Bubblesort(int A[1000n]){
    for(int j=n; j>1; j--)            → O(n)   Ω(n)
        for(int i=1; i<=j; i++)       → O(j) = O(O(n)) = O(n)   Ω(j)
            if(A[i] > A[i+1]){        } O(1)   } Ω(1)
                temp = A[i];          }        }
                A[i] = A[i+1];        }        }
                A[i+1] = temp; }      }        }
```

$O(n) * [O(n) * O(1)] = O(n^2)$ $T_n \in O(n^2)$

$\rightarrow T_n \in \Theta(n^2)$

$\Omega(n) * [\Omega(j) * \Omega(1)] = \Omega(n^2)$ $T_n \in \Omega(n^2)$

$\Omega(\Omega(n)) = \Omega(n)$

Year. Month. Date. ( )

```
int f1(int n){
    Sum = 1;                          → O(1)
    for(int i=1; i<=n; i++){          O(n)
      Count=0;                        → O(1)
      for(int j=1; j<=2^i; j++)       O(2^i) * [O(1)+O(1)] = O(1)O(2^i)
        Count++;                      O(1)
      Sum += Count;                   → O(1)
    return Sum; }
```

$T_{(n)} \in O(1) + O(n)\underbrace{[O(1)+O(1)*O(2^i)+O(1)]}_{O(n*2^n)} = O(n^2 2^n)$

: (V, W) ← 9

* $n = 2^m$ → تعداد دفعات تکرار حلقه → $m = \log_2^n$

```
? for(int i=1; i<n; i++)
    if(A[i]==x)
      return i;
```

$T_n \in O(n)$

$T_n \in \Omega(1)$

```
? for(int i=n; i>=1; i=i/3)
```

$n = 3^m$ → تعداد دفعات m → $\log_3^n$ → $O(\log_3^n) \in O(\log^n)$

```
? for(int i=1; i<=n; i=i*3)
```

$n/3$ → $O(n)$

Subject:

Year. Month. Date. ( )

```
? int f1 (int n) {
    int Count = 0;          → O(1)
    for(int i=0; i<n; i++)
      for(int j=1; j<=2^i; j++)
        Count++;            → O(1)
    return Count; }         → O(1)
```

$T(n) = O(1) + \sum_{i=0}^{n} (2^i + O(1)) + O(1)$

$O(1) + \sum_{i=0}^{n} 2^i + \sum_{i=0}^{n} O(1) + O(1)$

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

$T(n) = O(1) + 2^{n+1} - 1 + n\,O(1) + O(1)$

$O(1) + O(2^n) + O(n) + O(1) = O(2^n)$

```
? int f2 (int n) {
    int Sum = 1, m = 1;     → O(1)
    for(int i=1; i<=n; i++) → O(n)
      for(int j=1; j<=i; j++) → O(n)
        m = m + 2;
      Sum = m + Sum; }      → O(1)
    return Sum; }           → O(1)
```

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

$T(n) = O(1) + \sum_{i=1}^{n} (i + O(1)) + O(1) = O(1) + \underbrace{\sum_{i=1}^{n} i}_{\in O(n^2)} + \sum_{i=1}^{n} O(1) + O(1)$

$O(1) + O(n^2) + O(n) + O(1) \in O(n^2)$

```
? int f3 (int n) {
    int sum = 1, f = 1;     → O(1)
    for(int i=1; i<=n; i++) → O(n)
      f = f + 2;            → O(1)
      Sum = Sum + f; }
    return Sum; }           → O(1)
```

$T(n) \in O(n)$

* دنباله فیبوناچی:

بدترین حالت و بهترین حالت یکسان است.

```
int fibo(n){
if(n<=2) return 1;          → O(1)
int fn, f1=1, f2=1;         → O(1)
for(int i=3; i<=n; i++){    → O(n)
  fn = f1+f2;               O(1)
  f1 = f2;                  O(1)
  f2 = fn;}                 O(1)
  return fn;}               O(1)
```

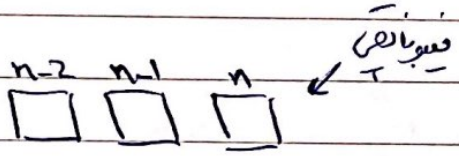
$T(n) = O(1) + O(n) * O(1) + O(1) = O(1) + O(n) + O(1) = O(n)$

$T(n) \in O(n)$

* الگوریتم های بازگشتی:

قسمت پایه

قسمت بازگشت




```
int fibo(n){
if(n<=2) return 1;
return (fibo(n-1) + fibo(n-2));}
```

پایه / بازگشت

$$T(n) \begin{cases} 1 & n<=2 \\ T(n-1)+T(n-2) & o.w \text{ (otherwise)} \end{cases}$$

مثال: $(n,k) \leftarrow$ ترکیب

* فاکتوریل:

```
int fact(n){
int f=1;                    O(1)
for(int i=1; i<=n; i++)     O(n)
  f=f*i;                    O(1)
return f;}                  O(1)
```

$T(n) \in O(n)$

```
int fact(int n){
if(n==1) return 1;
return (n*fact(n-1));}
```

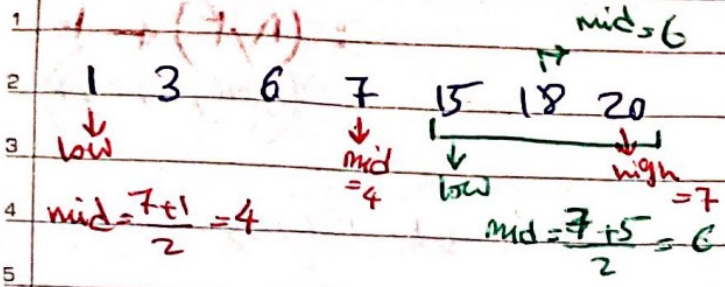
$$T(n) \begin{cases} O(1) & n==1 \\ T_{(n-1)}+O(n) & o.w \end{cases}$$

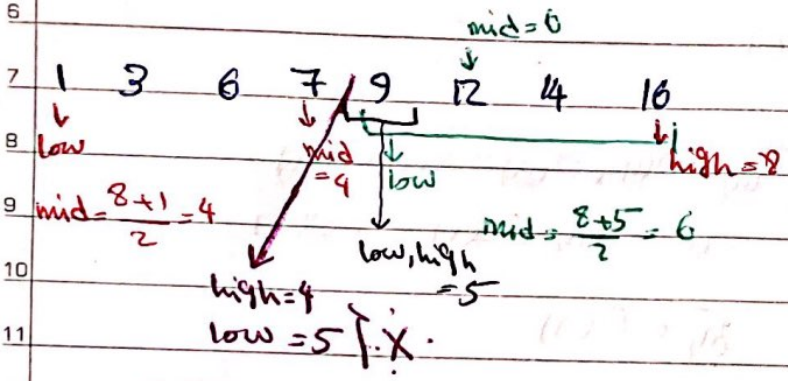
NEGAR

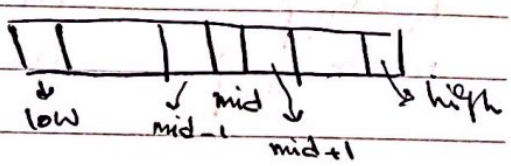
Subject:

Year. Month. Date. ( )

* جستجوی دودویی (binary Search)








```
int binarySearch(int A[100], int n){
  int low=1, high=n;
  if(low<=high){
  int mid = (low+high)/2;
  if(A[mid]==n) return mid;
  else if(A[mid]>n)
    return(binarySearch(A[low,...,mid-1],n);
  else
  return(binarySearch(A[mid+1,...,high],n);
  return -1;
```

بدترین حالت زمانی است که داده عضو نباشد!

$$T(n)\begin{cases} O(1) & n=1 \\ T(n/2)+O(1) & o.w \end{cases}$$

* روش‌های حل روابط بازگشتی
1. حدس و استقرا
2. جایگزاری مکرر
3. درخت بازگشت
4. معادله مشخصه
5. قضیه اصلی
6. قضیه متقدم

1) حدس و استقرا:

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

$T_0=0$

$T(1)=T(0)+1=1$

$T(2)=T(1)+1=2$

$T(n)\overset{?}{=}n$

پایه استقرا $\rightarrow n=1 \rightarrow T(1)\overset{?}{=}1 \rightsquigarrow T(1)=T(0)+1=1$ ✓

$n=2 \rightarrow T(2)\overset{?}{=}2 \rightsquigarrow T(2)=T(1)+1=2$ ✓

$\Rightarrow T(n)=n$ ✓

فرض استقرا $\rightarrow T(n)=n$

حکم استقرا $\rightarrow T(n+1)\overset{?}{=}n+1 \rightsquigarrow T(n+1)=T(n)+1=n+1$ ✓

$$T(n)=\begin{cases}1 & n=1\\ T(n/2)+1 & n\geq 1,\ n=2^k\end{cases}$$

$T(1)=1$

$T(2)=T(1)+1=2$

$T(4)=T(2)+1=3$

$T(8)=T(4)+1=4$

$T(16)=T(8)+1=5$

$T(n)\overset{?}{=}1+\log n$

پایه استقرا $\rightarrow n=1 \rightarrow T(1)\overset{?}{=}1 \rightsquigarrow T(1)=1+\log 1=1$ ✓

$n=2 \rightarrow T(2)\overset{?}{=}2 \rightsquigarrow T(2)=1+\log 2=2$ ✓

$\Rightarrow T(n)=1+\log n$

فرض استقرا $\rightarrow T(n)=1+\log n$

حکم استقرا $\rightarrow T(2n)\overset{?}{=}1+\log 2n \rightsquigarrow T(2n)=T(n)+1=1+\log n+1 \overset{\log 2}{=} \log 2n+1$

Subject:

Year. Month. Date. ( )

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

$T(1)=1$

$T(2)=2T(1)+1=3$

$T(3)=2T(2)+1=7$

$T(4)=2T(3)+1=15$

$T(5)=2T(4)+1=31$

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

برهان استقرا: $T(1)\overset{?}{=}1 \rightsquigarrow T(1)=2^1-1=1$ ✓

$T(2)\overset{?}{=}3 \rightsquigarrow T(2)=2^2-1=3$ ✓

$\boxed{T(n)=2^n-1}$

فرض استقرا: $T(n)=2^n-1$

حکم استقرا: $T(n+1)\overset{?}{=}2^{n+1}-1 \rightsquigarrow T(n+1)=2T(n)+1=2\cdot(2^n-1)+1=2^{n+1}-1$ ✓

$$T(n)=\begin{cases}0 & n=1\\ 2T(n/2)+n-1 & n>1,\ n=2^k\end{cases}$$

$T(n)=n\log^n-n+1$ به کمک استقرا

$T(n)\in O(n\log^n)$

$T(n)=2T(n/2)+n-1=2\left(2T(n/2^2)+n/2-1\right)+n-1=2^2T(n/2^2)+2n+(1-2)=$

$2^2\left(2T(n/2^3)+n/2^2-1\right)+2n-(1+2)=2^3T(n/2^3)+3n-(1+2+2^2)=$

$2^3\left(2T(n/2^4)+n/2^3-1\right)+3n-(1+2+2^2)=2^4T(n/2^4)+4n-(1+2+2^2+2^3)=\dots$

$2^iT(n/2^i)+in-\underbrace{(1+2+2^2+\dots+2^{i-1})}_{2^i-1}=2^iT(n/2^i)+in-(2^i-1)$

$2^kT(n/2^k)+kn-(2^k-1)=n\log^n-n+1$

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

برهان استقرا ②

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

$T(n)=n$

$T(n)=T(n-1)+1=T(n-2)+1+1=T(n-2)+2=T(n-3)+1+2=T(n-3)+3=T(n-4)+1+3$

$=T(n-4)+4=\dots=T(n-n)+n\Rightarrow$

$T(n)=n$

Year. Month. Date. ( )

$$T(n)=\begin{cases}1 & n=1\\ T(n/2)+1 & n>1,\ n=2^k\end{cases}$$

$T(n)=1+\log n$

$T(n)=T(n/2)+1=T(n/4)+1+1=T(n/4)+2=T(n/8)+1+2=T(n/8)+3=$
$T(n/16)+1+3=T(n/16)+4=\dots=T(n/2^k)+k=T(1)+k=1+k$

$T(n)=1+k$
$k=\log n$ $\Rightarrow T(n)=1+\log n$
چون $2^k=n$

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

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

$T(n)=2T(n-1)+1=2(2T(n-2)+1)+1=2^2T(n-1)+3=2^2(2T(n-3)+1)+3=$
$2^3T(n-3)+(2^2+2^1+2^0)=2^4T(n-4)+(2^3+2^2+2^1+2^0)=2^4T(n-4)+(2^4-1)=\dots=$
$2^{n-1}\underbrace{T(n-(n-1))}_{T(1)}+2^{n-1}-1=2^n-1$ (2^4-1 / $2-1$)

(۳) روش بازگشت: نحوه حالت‌بندی یک عبارت بازگشتی و چگونگی متناظر با آن توصیف از رابطه‌ی بازگشتی بدست میاد

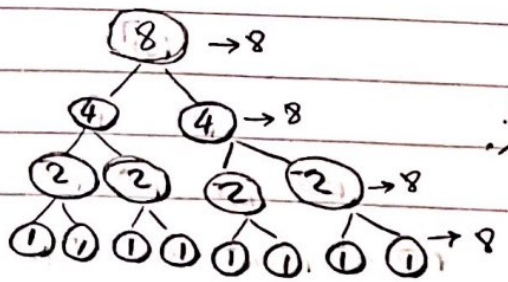
نمونه‌ی داده شده و با جمع آوردن مقادیر ثابت تمام سطوح جواب بدست میاد

$T(n,9)\leftarrow 11$:

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

$\log n$ = ارتفاع درخت بازگشت +
۱ = تعداد سطوح

$T(8)=2T(4)+8$
$T(4)=2T(2)+4$
$T(2)=2T(1)+2$



$T(8)=4*8=32$

* تابع درخت T وجود داشته باشد و
باید حد بالای درخت متناظر ثابت داشته باشد.

NEGAR

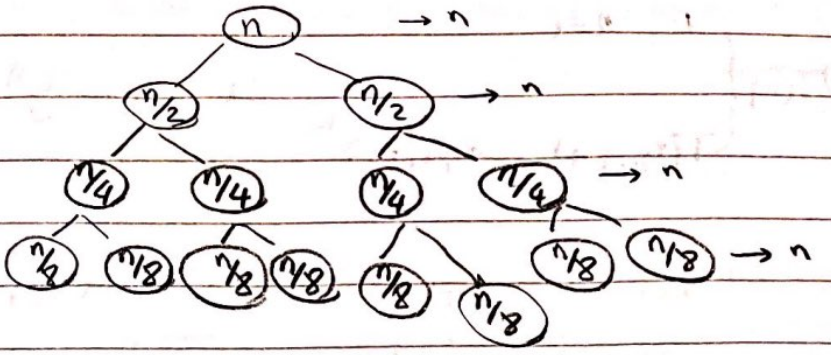
Subject:

Year. Month. Date. ( )

$T(n) = 2T(n/2) + n$

$T(n/2) = 2T(n/4) + n/2$

$T(n/4) = 2T(n/8) + n/4$



$2^i * \frac{n}{2^i} = n$

تعداد سطوح: $\log n + 1$ ←

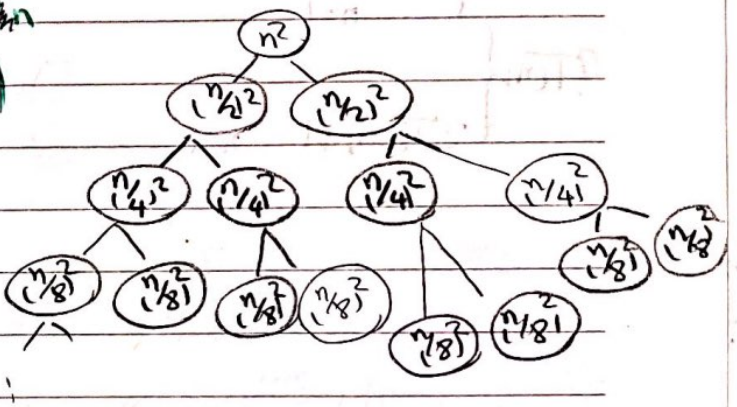
* تعداد گره‌های هر سطح دو برابر تعداد گره‌های سطح قبل از آن است اما مساحد هر سطح نصف مساحد گره‌های سطح قبلش است.

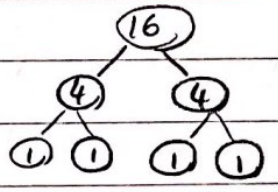
$T(n) = n(1 + \log n)$

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

$T(n) = 2n^2 - n$

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





$T(4) = 2T(2) + 16$

$T(2) = 2T(1) + 4$

$T(1) = 1$

$T(4) = 16 + 2*4 + 4*1 = 28$

مساحد گره‌ی در سطح i ام: $(n/2^i)^2$ — چند تا گره در سطح i ام؟ چند تا مساحد در سطح آخر؟ باید $1+\log n$ باشد — سطح آخر؟

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

$$\frac{n^2}{2^{\log n}}\left(2^{\log n} + 2^{\log n - 1} + 2^{\log n - 2} + \dots + 2^0\right) = 2^{1+\log n} - 1 = 2^n - 1 \longrightarrow n(2^n - 1)$$

n

Fibo

$?\ T(n)=T(n-1)+2T(n-2)$ → $n-(n-2)=2$

④ معادله مشخصه:
← رابطه بازگشتی همگن درجه دو
اختلاف بزرگ‌ترین متغیر با اندیس T و کوچک‌ترین متغیر با اندیس T برابر 2 است.

$?\ T(n)=5T(n-1)-8T(n-2)+\underbrace{n^2}_{f(n)}$ ← رابطه بازگشتی است. طرف راست ندارد.

$?\ T(n)=5T(n-1)+2T(n-2)+4T(n-3)$ → $n-(n-3)=3$ ← رابطه بازگشتی همگن درجه سه

$?\ T(n+1)=2T(n)-4T(n-1)$ → $(n+1)-(n-1)=2$ ← رابطه بازگشتی همگن درجه دو

① معادله مشخصه ② حل معادله مشخصه ③ جواب رابطه بازگشتی برای جواب معادله مشخصه

$?\ T(n)=T(n-1)+2T(n-2)\ ,\ T(0)=2\ ,\ T(1)=7$

① $\to T(n)-T(n-1)-2T(n-2)=0$ ② $r^2-r-2=0$ → معادله مشخصه

③ $\to (r+1)(r-2)=0 \to r=-1,\ r=2$ ④ $T(n)=C_1 2^n+C_2(-1)^n$

⑤ $\to T(0)=2 \to C_1 2^0+C_2(-1)^0=2 \to C_1+C_2=2$ → $C_1=3,\ C_2=-1$
$T(1)=7 \to C_1 2^1+C_2(-1)^1=7 \to 2C_1-C_2=7$

⑥ $T(n)=3*2^n+(-1)*(-1)^n \Rightarrow T(n)=3*2^n-(-1)^n$

$?\ T_{(n+2)}=4T(n+1)-4T(n)\ ,\ T(0)=1, T(1)=3$

① $\to T(n+2)-4T(n+1)+4T(n)=0$ ② $\to r^2-4r+4=0$

③ $\to \Delta=16-16=0\ ,\ r=-b/2a$ ④ $\to T(n)=C_1 r^n+C_2 n r^n \Rightarrow C_1 2^n+C_2 n 2^n$

⑤ $\to T(0)=1 \to C_1 2^0+C_2*0*2^0=1 \Rightarrow C_1=1$
$T(1)=3 \to C_1 2^1+2C_2=3 \Rightarrow C_2=1/2$

⑥ $T(n)=1(2^n)+\frac{1}{2}(n)(2^n) \Rightarrow T(n)=2^n+n(2^{n-1})$

NEGAR

Subject:

Year. Month. Date. ( )

$T(n) = c_1 r_1^n + c_2 r_2^n$ ← r_1, r_2

$T(n) = c_1 r^n + c_2 n r^n$ ← r

* حل رابطه بازگشتی خطی همگن مرتبه 2 ← معادله مشخصه ← حل

(۳) قضیه اصلی:

$T(n) = aT(n/b) + f(n) \rightsquigarrow a \geq 1, b > 1, b \neq 1$

$f(n) \square n^{\log_b^a}$

$T(n) \in \Theta(n^{\log_b^a})$ ← $f(n) < n^{\log_b^a}$ (1

$T(n) \in \Theta(f(n) \log n)$ ← $f(n) = n^{\log_b^a}$ (2

$T(n) \in \Theta(f(n))$ ← $f(n) > n^{\log_b^a}$ (3

؟ $T(n) = 4T(n/2) + \log n$

a ↙ b ↙ $f(n)$

$n^{\log_b^a} = n^{\log_2^4} = n^2$ $\quad \log n < n^2$

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

؟ $T(n) = 8T(n/9) + n \log n$

a ↙ b ↙ $f(n)$

$n^{\log_b^a} = n^{\log_9^8} < 1$ $\quad n \log n > n^{\log_9^8}$

$T(n) \in \Theta(n \log n)$

۱۲ ← (۱) و (۲) :

؟ $T(n) = T(\frac{2n}{3}) + 1 \rightsquigarrow T(n) = T(n/\frac{3}{2}) + 1$

$a=1, b=\frac{3}{2}, f(n)=1$

$n^{\log_b^a} = n^{\log_{3/2}^1} = n^0 = 1$ $\quad 1 = n^{\log_{3/2}^1}$

$T(n) \in \Theta(1 * \log n)$

؟ $T(n) = 9T(n/3) + n$

a ↙ b ↙ $f(n)$

$n^{\log_b^a} = n^{\log_3^9} = n^2$ $\quad n < n^2$

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

؟ $T(n) = 4T(n/2) + n^2\sqrt{n}$

a ↙ b ↙ $f(n) = n^2 * n^{1/2} = n^{5/2}$

$n^{\log_b^a} = n^{\log_2^4} = n^2$ $\quad n^2 < n^{5/2}$

$T(n) \in \Theta(n^{5/2})$

Year. Month. Date. ( )

؟ $T(n) = \underset{a}{3}T(\underset{b}{n/4}) + \underbrace{n \log n}_{f(n)}$ $\quad n^{\log_b a} = n^{\log_4 3} < 1$ $\quad n \log n > n^{\log_b a}$

$T(n) \in \Theta(n \log n)$

؟ $T(n) = \underset{a}{8}T(\underset{b}{n/4}) + \underbrace{5n^2}_{f(n)}$ $\quad n^{\log_b a} = n^{\log_4 8} = n^{3/2}$ $\quad n^{3/2} > 5n^2$

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

؟ $T(n) = \underset{a}{2}T(\underset{b}{n/2}) + \underbrace{n^3}_{f(n)}$ $\quad n^{\log_b a} = n^{\log_2 2} = n^1$ $\quad n^1 < n^3$

$T(n) \in \Theta(n^3)$

؟ $T(n) = \underset{a}{16}T(\underset{b}{n/4}) + \underbrace{n^2}_{f(n)}$ $\quad n^{\log_b a} = n^{\log_4 16} = n^2$ $\quad n^2 = n^2$

$T(n) \in \Theta(n^2 \log n)$

تمرین متفرقه: ①

؟ $T(n) = 2T(\sqrt{n}) + \log n \rightsquigarrow T(n) = 2T(n^{1/2}) + \log n$

$n = 2^m \rightsquigarrow m = \log n$ $\quad T(2^m) = 2T(2^{m/2}) + m \xrightarrow{S(m) = T(2^m)}$

$S(m) = \underset{a}{2}S(\underset{b}{m/2}) + \underbrace{m}_{f(n)}$ $\quad m^{\log_b a} = m^{\log_2 2} = m^1$ $\quad m^1 = m^1$

$S(m) \in \Theta(m \log m)$

$S(m) \in \Theta(m \log m) \xrightarrow[n = 2^m \;\&\; m = \log n]{S(m) = T(2^m) = T(n)} T(n) \in \Theta(\log n \log \log n)$

NEGAR

Subject:
Year. Month. Date. ( )

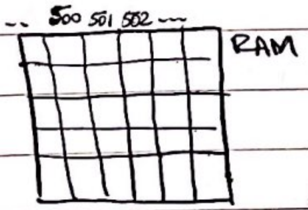
فصل سوم: ساختمان داده‌های پایه‌ای :

پیاده‌سازی ← ساختمان داده ← طراحی الگوریتم ← مدل انتزاعی ← مسأله

تست و ارزیابی — تحلیل

ساختمان داده: مجموعه‌ای دارای ساختار منظم از داده‌های مرتبط به هم

- ساده: متغیرها
- ترکیبی (ترکیب چند ساختمان داده ساده)
  - ترتیبی: ترتیب منطقی و فیزیکی عناصر یکی است. (منطقی: چیدمان عناصر در ساختمان داده / فیزیکی: چیدمان عناصر در حافظه) — آرایه / پشته ترتیبی — صف ترتیبی
  - غیرترتیبی: " " " " " نیست. — لیست پیوندی — پشته پیوندی / صف پیوندی



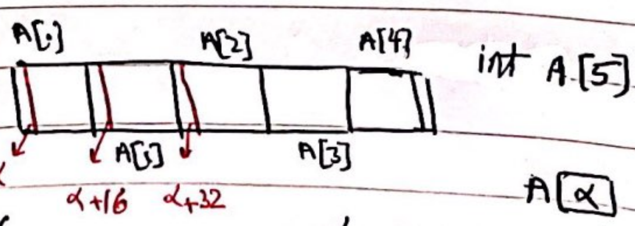
* بیت‌ها به صورت متوالی هستند. *شماره ردیف بیت = آدرس بیت

* چون بیت‌هایی که تشکیل یک متغیر را می‌دهند همه پشت سر هم هستند و هم‌چنین به صورت متوالی هستند دانستن شماره ردیف بیت آغازین کافی است.

* آدرس متغیر ← شماره ردیف بیت آغازین
* آدرس آرایه ← شماره ردیف بیت آغازین

* نوع آرایه چیه؟ چرا؟ به خاطر اینکه نوع آرایه مشخص می‌کند که نوع عناصری که تشکیل آرایه می‌دهند چیه و ما باید بدونیم که هرکدام از این خونه‌ها چند بایت از حافظه را اشغال می‌کند.

int ← ۲ بایت ← هرکدام از این خونه‌ها ۱۶ بیت از حافظه را اشغال کرده.



int A[5]

A [α]

↓

همنام با آرایه، یک اشاره‌گر وجود دارد که به ابتدای آرایه اشاره می‌کند.

* آرایه های معمولی (ایستا، غیرپویا) در زبان کامپایل بخشی از حافظه تخصیص داده شد اما آرایه های پویا در زبان اجرا بخشی از حافظه تخصیص داده شد.

آرایه: قدیمی ترین و رایج ترین ساختمان داده است.
ساختاری ترتیبی از عناصر هم نوع که در آن هر عنصر دارای یک اندیس منحصر به فرد است.

$a_0, a_1, \dots, a_{n-1}$

```
class array{
int A[K];
int n;
bool isFull(){
  if(n==k) return true;
  return false; }
int size(){
  return n; }
bool isEmpty{
  if(n==0) return true;
  return false; }
```

* تعداد عنصری که در آرایه هستند.
* قبل از اینکه اقدام کنیم به عنصر یک عنصر موجود باید چک کنیم که عنصری هست یا نه!

* آرایه های معمولی (استاتیک، غیرپویا) در زبان کامپایلر بهشون حافظه تخصیص داده شد اما آرایه های پویا در زبان اجرا بهشون حافظه تخصیص داده شد.

آرایه: متداول‌ترین و رایج‌ترین ساختمان داده است.
ساختاری ترتیبی از عناصر هم نوع که در آن هر عنصر دارای یک اندیس منحصر به فرد است.

$a_0, a_1, \dots, a_{n-1}$

```
class array{
int A[K];
int n;
⊖bool isfull(){                          Ω(1) O(1)
  if(n==k) return true;
  return false; }
⊖int size(){
  return n; }
⊖bool isEmpty{
  if(n==0) return true;
  return false; }
⊖void insert(int pos, int n){
  if (isfull()) {cout<<"the array is full"; return;}   Ω(1) O(1)
  if(pos<0 || pos>k-1) return;                          Ω(1) O(1)
  if (pos>n-1) {                                        Ω(1) O(1)
     A[pos]=n; n++; return;}
  for(int i=n-1; i>=pos; i--){
     A[i+1]=A[i];}
  A[pos]=n;
  n++;}
```

* تعداد عناصری که در آرایه هستند.

* قبل از اینکه اقدام بکنیم به حذف یک عنصر موجود باید چک بکنیم که عنصری هست یا نه!

(۸,۱۶)+۱۳

* در اینجا نیازی به شیفت دادن عناصر نیست.

* اگر از انتها شروع نمی‌کردیم هر بار عنصر بعدی را پاک می‌کردیم.

* عمل شیفت داریم

insert → {
Ω(1) → بهترین زمان در بهترین حالت → عنصر n درج بشود و لازم نشد for!
O(n) → بدترین زمان در بدترین حالت → اجرا کرد for, pos=0, pos<n
}

NEGAR

Subject:

Year. Month. Date. ( )

```cpp
int delete(int pos){
if (isEmpty())
{cout<<"the array is Empty";
  return;}
  if(pos<0 || pos>n-1) return;
  int x = A[pos];
  for(int i=pos+1; i<n; i++)
      A[i-1]=A[i];
      n--;   return x; }}
```

وقتی می‌خواهیم عنصری را از آرایه که در اندیس pos ذخیره شده را حذف کنیم باید تمام خانه‌هایی که بعد از این اندیس هستند یعنی اندیس بزرگ‌تر از pos هست و هم چنین کمتر یا مساوی با n-1 است را یک واحد به سمت چپ شیفت بدیم تا یک بعد اندیس را کم کنیم.

insert {
for اجرا نشود یا فقط یکبار اجرا شود (عنصر n-1 حذف شد) ← O(1) ← پیچیدگی زمانی در بهترین حالت
حذف اولین عنصر ← O(n) ← پیچیدگی زمانی در بدترین حالت

آرایه ۲ بعدی، ۳ بعدی و به طور کلی بیشتر از ۳ و ... و n بعدی:

```cpp
int A[u1];
int A[u1][u2];
int A[u1][u2]...[un];   →   u1*u2*...*un
```

تعداد کل خانه‌های آرایه

int A[4][5] →

int A[3][5][6] → عکس :)

Year. Month. Date. ( )

تبدیل آدرس منطقی به آدرس فیزیکی: با داشتن اندیس یک عنصر در آرایه برسیم به محل ذخیره سازی آن عنصر در آرایه

α ← آدرس شروع آرایه

int A[u1];
A[i] آدرس فیزیکی → $\alpha + i$ (طول نوع آرایه)

* هر کدام از خانه های آرایه به اندازه طول نوع آرایه حافظه اشغال می کنند.

int A[u1],[u2];
A[i][j] آدرس فیزیکی → $\alpha + [(i * u_2) + j] *$ طول نوع آرایه

int A[u1],[u2],[u3];
A[i1][i2][i3] آدرس فیزیکی → $\alpha + [i_1 * (u_2 * u_3) + i_2 * u_3 + i_3] *$ طول نوع آرایه

: (۸/۱۸) ← ۱۴

int A[3][5][6];
A[2][4][3] آدرس فیزیکی →

0 → 5x6
1 → 5x6
2,0 → 6
2,1 → 6
2,2 → 6
2,3 → 6
3

$2*(5*6) + 4*6 + 3$

int A[u1][u2]...[un]
A[i1][i2]...[in] آدرس فیزیکی → $\alpha + [i_1 * (u_2 * \dots * u_n) + i_2 * (u_3 * \dots * u_n) + \dots + i_{n-1} * (u_n) + i_n] *$ طول نوع آرایه

n-2 ضرب ، n-3 ضرب

$1 + 2 + \dots + n-2 \in O(n^2)$

$$a_j \begin{cases} u_{j+1} * u_{j+2} * \dots * u_n & j \neq n \\ 1 & j = n \end{cases}$$

آدرس فیزیکی: $\alpha + \left[\sum_{j=1}^{n} i_j a_j\right] *$ طول نوع آرایه

$\alpha + [i_1 * (u_2 * \dots * u_n) + i_2 (u_3 * \dots * u_n) + \dots + i_n] +$ طول نوع آرایه

$a_n = 1$
$a_{n-1} = u_n * a_n$
$a_{n-2} = u_{n-1} * a_{n-1}$
$a_{n-3} = u_{n-2} * a_{n-2}$
⋮
$a_1 = u_2 * a_2$

جمع i_n ←
تعداد اعمال ضرب مورد نیاز برای محاسبه a_j ← (n-1) ضرب
برای محاسبه $i_j * a_j$ ← (n) عمل ضرب

O(n)

ubject:

ear. Month. Date. ( )

تبدیل آدرس فیزیکی به منطقی : آدرس محل ذخیره سازی عنصر را در آرایه یک بعدی پیدا می‌کنیم.

int A[u1]

A[i] آدرس فیزیکی ← $\alpha + i * \text{طول نوع آرایه} = D \rightarrow i = \frac{D-\alpha}{\text{طول نوع آرایه}}$ آدرس منطقی

int A[u1][u2]

A[i][j] آدرس فیزیکی ← $\alpha + [i * u_2 + j] * \text{طول نوع آرایه} = D \rightarrow \frac{D-\alpha}{\text{طول نوع آرایه}} = d$ آدرس منطقی

$i * u_2 + j = d$

$i = d / u_2$ → خارج قسمت

$j = d \,\%\, u_2$ → باقی مانده

int A[u1][u2]...[un]

A[i1][i2]...[in] آدرس فیزیکی ← $\alpha + \left[\sum_{j=1}^{n} i_j a_j\right] * \text{طول نوع آرایه} = D \rightarrow \frac{D-\alpha}{\text{طول نوع آرایه}} = d$

$\sum_{j=1}^{n} i_j a_j = d \rightarrow i_1 * (u_2 * u_3 * \dots * u_n) + i_2 (u_3 * u_4 * \dots * u_n) + \dots + i_{n-1}(u_n) + i_n = d$

$[i_1 (u_2 * u_3 * \dots * u_{n-1}) + i_2 (u_3 * u_4 * \dots * u_{n-1}) + \dots + i_{n-1}] * u_n + i_n = d$

* = d / u_n ، $i_n = d \,\%\, u_n$ ، d_n

$i (u_2 * u_3 * \dots * u_{n-1}) + i_2 (u_3 * u_4 * \dots * u_{n-1}) + \dots + i_{n-1} = d_n$

$[i * (u_2 * u_3 * \dots * u_{n-2}) + i_2 (u_3 * u_4 * \dots * u_{n-2}) + \dots + i_{n-2}] * u_{n-1} + i_{n-1} = d_n$ ** = d_n / u_{n-1} ، $i_{n-1} = d_n \,\%\, u_{n-1}$

$i_n = d \,\%\, u_n$ $d_n = d / u_n$

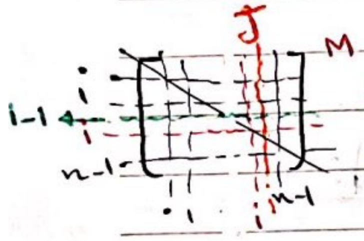
$i_{n-1} = d_n \,\%\, u_{n-1}$ $d_{n-1} = d_n / u_{n-1}$

ماتریس مثلثی: ۱) بالا مثلثی $\begin{bmatrix} & \diagdown \\ 0 & \end{bmatrix}_{n\times n}$

۲) پایین مثلثی $\begin{bmatrix} \diagdown & 0 \\ & \end{bmatrix}_{n\times n}$

$n^2 \rightarrow$ تعداد کل عناصر

$n+(n-1)+\dots+1 = \frac{n(n+1)}{2}$ $\qquad \frac{\frac{n(n+1)}{2}}{n^2} \approx 55\%$



A	a_{00}	a_{01}	...	a_{0n-1}	a_{11}	a_{12}	...	a_{1n-1}	...

درایه i,j در کدام اندیس از آرایه A ذخیره می‌شود؟

$$K = [(n^*i) - (1+2+\dots+i-1)] + (j-i)$$

↳ تعداد کل عناصر ماقبل از سطر i ام اندیس i ؛ $j-1-i+1$ تعداد عناصر قبل از عنصر مورد نظر در سطر i

(۴۹)

۱۵ ← (۸,۲۳):

ماتریس اسپارس (تنک) $_{m\times n}$: تعداد زیادی از عناصر صفر هستند. ابعاد بزرگ دارند.

$N << m^*n$ تعداد عناصر غیرصفر

۱) روش ترتیبی ۲) روش پیوندی

۱) روش ترتیبی: از یک آرایه دوبعدی استفاده می‌کنیم که سه ستون دارد. تعداد سطرها برابر است با تعداد درایه‌های غیرصفر +۱

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

	Row	Columns	Value
1	5 سطر	6 ستون	6 تعداد ناصفر
2	0	4	9
3	1	1	8
4	2	2	4
5	2	3	2
6	3	5	5
7	4	2	2

سطر ۱ ← فقط اطلاعات می‌دهد

* از سطر یک به بعد هر کدام از سطرها اطلاعات یک عنصر غیرصفر را نمایش می‌دهد به ترتیب سطر به سطر و به ترتیب ستون

$\frac{3^*(N+1)}{m^*n} \rightarrow$ مقدار صرفه‌جویی در مصرف حافظه

Subject:

Year. Month. Date. ( )

چندجمله‌ای کلی اسپارس:

$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ یک چندجمله‌ای درجه n

$k+1$ درایه‌ی ضرایب لازم است.

P | a_0 | a_1 | a_2 | ... | a_{n-1} | a_n |

سایز $n+1$

$a_i \equiv x^i$ ضریب $\leftarrow$ درجه با اندیس i

$P(x) = 5x^5 + 3x^4 + (-1)x^3 + 10x^2 + x - 1$ $\quad P$ | -1 | 1 | 10 | -1 | 3 | 5 |

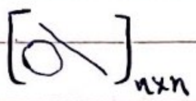
$P(x) = 4x^{1000} + 1$

توان	0	1000
ضریب	1	4

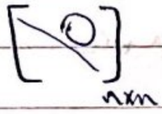
استفاده از لیست برای نگه‌داری اطلاعات

$P(x) = 5x^{100} + 3x^5 + 2x^1 + 10$

توان	0	1	5	100
ضریب	10	2	3	5

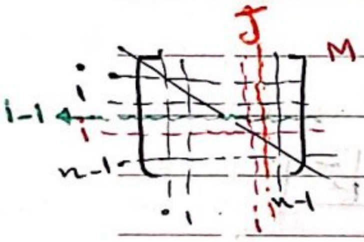


ماتریس مثلثی
بالا مثلثی:
پایین مثلثی:



تعداد کل عناصر ← n^2

$n+(n-1)+\dots+1 = \frac{n(n+1)}{2}$ $\qquad \frac{\frac{n(n+1)}{2}}{n^2} = 55\%$



A

a_{00}	a_{01}	...	a_{0n-1}	a_{11}	a_{12}	...	a_{1n-1}	...

در این زمینه هر کدام از این درایه ها در آرایه A در کجا ذخیره می شود؟

$K = [(n*i) - (1+2+\dots+i-1)] + (j-1)$

تعداد عناصر قبل از عنصر مورد نظر
تعداد کل عناصر ماقبل از سطر i ام:
در سطر i: $j-1-i+1$

۱۵ ← (۸/۲۳):

ماتریس اسپارس (تنک): تعداد زیادی از عناصر صفر هستند. ابعاد بزرگ دارند. $m \times n$

N = تعداد عناصر غیر صفر $\ll m*n$

① روش ترتیبی ② روش پیوندی

① روش ترتیبی: از یک آرایه دو بعدی استفاده می کنیم که سه ستون دارد. تعداد سطرها برابر است با تعداد درایه های غیر صفر +1

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

	Row	Columns	Value
1	5 سطر	6 ستون	6 مقدار ناصفر
2	0	4	9
3	1	1	8
4	2	0	4
5	2	3	2
6	3	5	5
7	4	2	2

← تعداد اطلاعات را می دهد

از سطر یک به بعد هر کدام از سطرها اطلاعات یک عنصر غیر صفر را نمایش می دهد به ترتیب سطرها و به ترتیب ستون

$\frac{3*(N+1)}{m*n}$ ← مقدار صرفه جویی در مصرف حافظه

Subject:

Year. Month. Date. ( )

چندجمله‌ای‌های اسپارس:

یک چندجمله‌ای از درجه n: $p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$

* نگه‌داشتن ضرایب مهم است.

P	a_0	a_1	a_2	$\dots$	a_{n-1}	a_n

سایز $n+1$

$a_i \equiv x^i$ ضریب ← i در خانه با اندیس

$p(x) = 5x^5 + 3x^4 + (-1)x^3 + 10x^2 + x - 1$

P	-1	1	10	-1	3	5

استفاده از آرایه دو بعدی که دو سطر دارد.

$p(x) = 4x^{1000} + 1$

0	1000	توان
1	4	ضریب

$p(x) = 5x^{100} + 3x^5 + 2x^1 + 10$

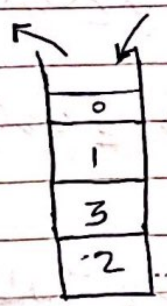
0	1	5	100	توان
10	2	3	5	ضریب

پشته (Stack):

ساختار داده‌ای ترتیبی که عمل حذف و درج عناصر از یک سمت آن صورت می‌گیرد.

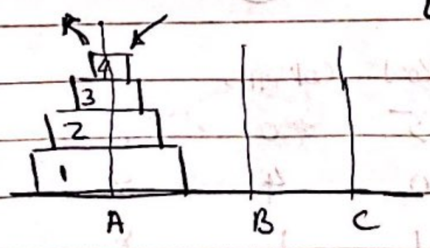
← بالای پشته

Top of Stack



LIFO = last input first output

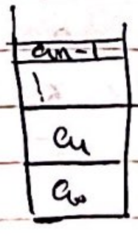
"آخرین ورودی اولین خروجی"



"برج هانوی"

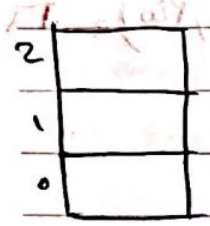
$S = \{a_0, a_1, \dots, a_{n-1}\}$

پایین‌ترین عنصر پشته



* a_n بالای استک قرار گرفته.

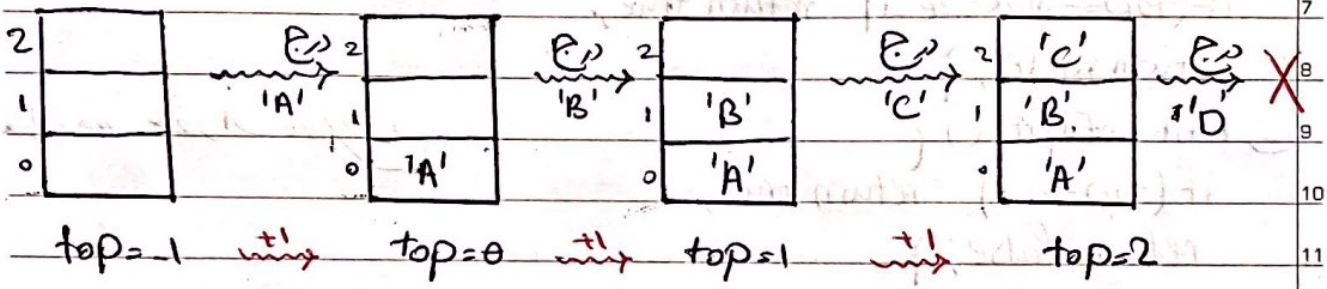
Const int MaxSize=3



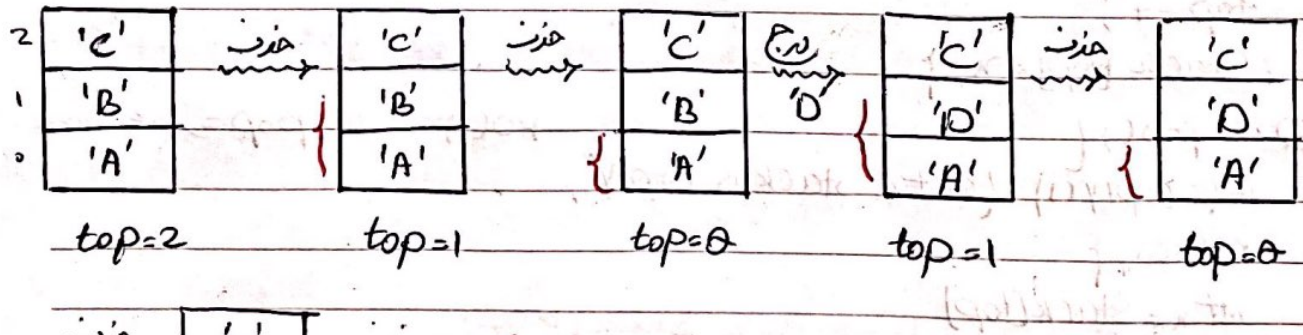
top = حاوی اندیس بالاترین عضو پشته

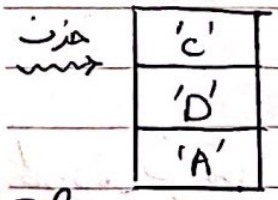
top=-1 ↔ پشته خالی است.

top = MaxSize-1 ↔ پشته پر است.



* همیشه در عمل حذف بالاترین عضو در پشته حذف می‌شود.





top=-1 ← پشته خالی است.

* تمام عناصر موجود در پشته باید هم نوع باشند. ← چرا؟
* پشته یک آرایه خاص است.

Subject:

Year. Month. Date. ( )

```
Class Stack{
  Const int maxsize=10;
  int top=-1;
  int stack[maxsize];
⊖bool isFull(){
  if(top==maxsize-1) return true;
  return false;}
⊖bool isEmpty(){
  if(top==-1) return true;
  return false;}
⊖void push(int x){
  if(isFull()){cout<<"Stack is full";
  return;}
  top++;
  Stack[top]=x;}
⊖int pop(){
  if(isEmpty()){cout<<"Stack is Empty";
  return;}
  int x=stack[top];
  top--;
  return x;}
};
```

→ int x=Stack[top--];
return x;

* برای درج عنصر جدید استفاده می‌شود

* برای حذف عنصر استفاده می‌شود

* درج عنصر جدید = push → O(1)

* حذف عنصر = pop → O(1)

کاربردهای پشته:

۱- استفاده در کامپایلرهای زبان‌های برنامه‌نویسی مثل بررسی عبارات پرانتزی

۲- فراخوانی توابع

نمایش یک عبارت ریاضی:

۱- میانوندی (infix) ← عملگر بین عملوندها است ... a*b

۲- پسوندی (postfix) ← عملگر بعد از عملوندهای خودش ظاهر شده ... ab*

۳- پیشوندی (prefix) ← عملگر قبل از عملوندها ... *ab

? a*b/c (infix → prefix)

a*b/c ⇝ *ab/c ⇝ /*abc

? a/b-c+d*e-a*c (infix → prefix)

a/b-c+d*e-a*c ⇝ /ab-c+*de-*ac ⇝ -/abc+*de-*ac

⇝ +-/abc*de-*ac ⇝ -+-/abc*de*ac

? -+-/abc*de*ac (prefix → infix) — از راست به چپ عمل می‌کنیم

-+-/abc*de*ac ⇝ -+-(a/b)c(d*e)(a*c) ⇝

-+(a/b-c)(d*e)(a*c) ⇝ -(a/b-c+d*e)(a*c) ⇝

a/b-c+d*e-a*c

* در نمایش پیشوندی یک عبارت ریاضی ترتیب اجرای عملگرها به ترتیب ظاهر شدن از راست به چپ است.

? a/b-c+d*e-a*c (infix → postfix)

a/b-c+d*e-a*c ⇝ (ab/)-c+(de*)-(ac*) ⇝ (ab/c-)+(de*)(ac*) ⇝

⇝ (ab/c-de*+)-(ac*) ⇝ ab/c-de*+ac*-

* دو عبارت با نمایش prefix متفاوت یک نمایش infix منحصر به فرد ندارند لزوماً.

Subject:
Year. Month. Date. ( )

* در نمایش پسوندی، ترتیب اجرای عملگرها به ترتیب ظاهر شدن آنها از چپ به راست است.

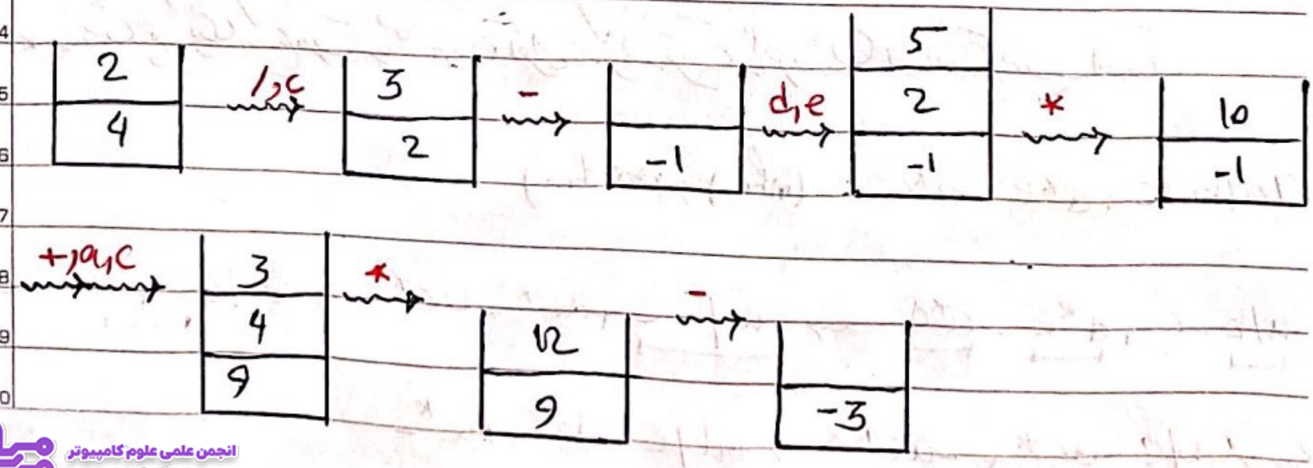
* الگوریتم ارزیابی با نمایش پسوندی خیلی راحت است.

الگوریتم ارزیابی یک عبارت ریاضی با نمایش پسوندی : { عملگر / عملوند / نشانه پایان رشته '\0' }

از یک پشته استفاده می‌کنیم، تمام نشانه‌ها را از چپ به راست مورد ارزیابی قرار می‌دهیم، اگر نشانه عملوند باشد محتویات عملوند را در پشته می‌ریزیم و اگر نشانه عملگر باشد (شامل دو عملگر یکانی یا دودویی) اگر عملگر یکانی باشد بالاترین عنصر موجود در پشته را از پشته بیرون می‌کشد و عملگر را روی آن اعمال می‌کند و حاصل را به پشته برمی‌گرداند و اگر عملگر دودویی باشد دو تا عنصر بالای پشته را از پشته بیرون می‌کشد و بعد عملگر را روی آنها اعمال می‌کند و حاصل را به پشته برمی‌گرداند.

و اگر نشانه، نشانه پایان رشته باشد یک عنصر در پشته موجود است، آن عنصر را برمی‌گرداند و حاصل عبارت ریاضی می‌شود.

? ab/c-de*+ae*-   a=4 , b=2 , c=3 , d=2 , e=5



* پشته این خصوصیت را دارد که هر چقدر عنصر دیرتر وارد پشته شود در بالای پشته قرار بگیرد و چون نمایش عبارت به صورت پسوندی هست بنابراین عملوندهای هر عملگر بلافاصله قبل از عملگر هستند.

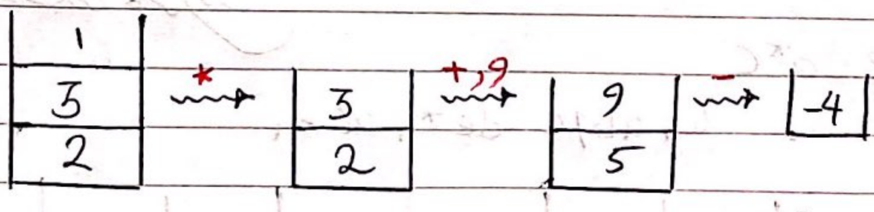
* به ازای هر عملوند ← $1pop + 1push$ ($O(1)$, $O(1)$)
* به ازای هر عملگر ← $1push$ ($O(1)$)

← همه = $O(1)$ * تعداد عملگرها + $O(1)$ * تعداد عملوندها = O(طول عبارت)

زمان مورد نیاز برای ارزیابی عبارت ریاضی با نمایش پسوندی

۱۷ ← (۸٫۳۰) :

? 2 3 1 * + 9 -



حداقل گنجایش پشته چقدر باشد ؟؟

الگوریتم تبدیل عبارت infix ← postfix :
- عملگر
- عملوند
- پرانتز باز یا بسته
- نشانه پایان رشته

از چپ به راست شروع می‌کند و این توکن‌ها را بررسی می‌کند و اگر نشانه عملوند باشد آن را به خروجی می‌برد یعنی کاری روی آن انجام نمی‌دهد، اگر پرانتز باز باشد آن را در پشته ذخیره می‌کند و اگر پرانتز بسته باشد از عناصری که در پشته هستند آنقدر pop می‌کند و به خروجی می‌برد تا به پرانتز باز برسد و وقتی به اولین پرانتز باز رسید آن پرانتز باز را هم از پشته خارج می‌کند ولی آن را به خروجی نمی‌برد. اگر نشانه یک عملگر باشد حالت‌های مختلفی ممکن است اتفاق بیافتد :

Subject:

Year. Month. Date. ( )

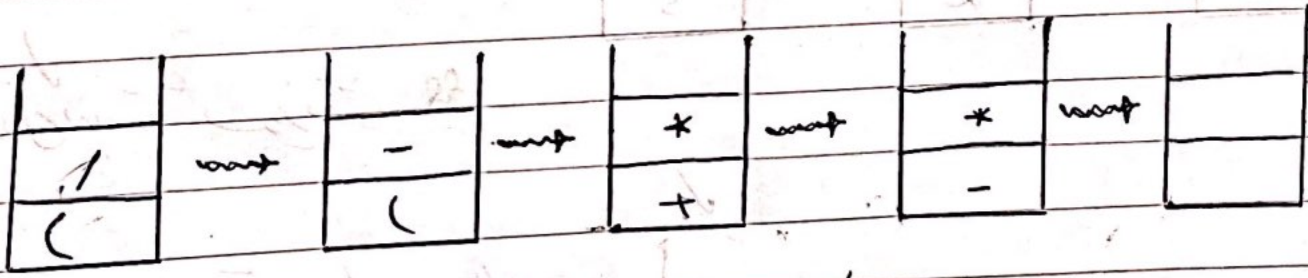
اگر عنصر بالای پشته پرانتز باز هم باشد عملگر وارد پشته می‌شود.

① اگر پشته خالی باشد عملگر وارد پشته می‌شود. ② اگر پشته خالی نباشد و اولویت عملگر جدیدی که می‌خواهد وارد پشته شود بیشتر از اولویت بالاترین عملگر موجود در پشته باشد آن عملگر وارد پشته می‌شود. ③ اما اگر اولویت عملگر جدید کمتر از اولویت بالاترین عملگر موجود در پشته باشد آنقدر از پشته pop انجام می‌شود و به خروجی برده می‌شود تا پشته خالی شود یا بالاترین عملگر موجود در پشته اولویتش از اولویت عملگر جدید کمتر شود و در این صورت عملگر جدید وارد پشته می‌شود.

در وقتی به انتها یا پایان رشته برسد هر چیزی که در پشته موجود باشد را pop می‌کند و به خروجی می‌برد یعنی پشته را هم سازی خالی می‌کند.

? (a/b-c)+d*e-a*c

→ ab/c-de*+ac*+

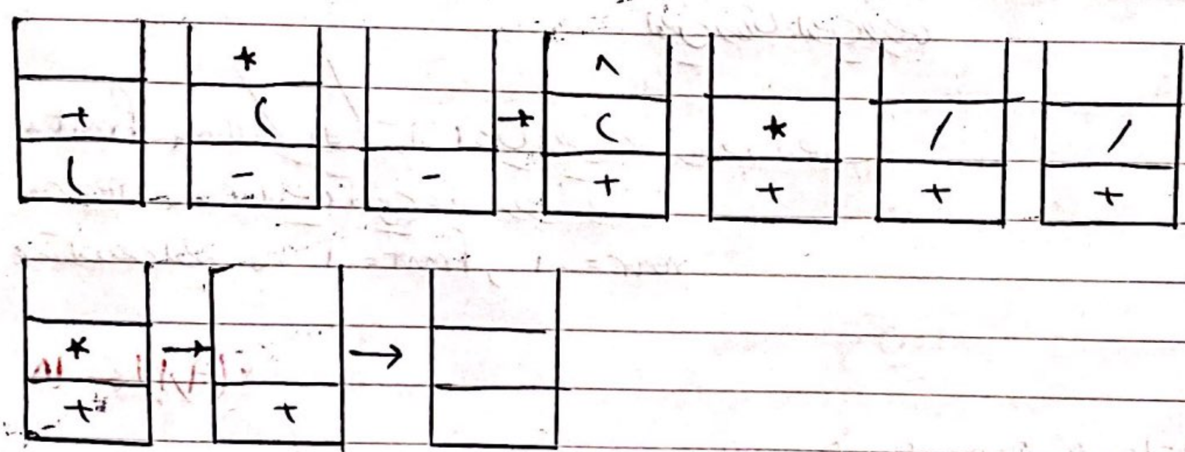


* با استفاده از پشته ترتیب ظاهر شدن عملگرها از چپ به راست در عبارت postfix معادل را مشخص می‌کند. آن عملگرهایی که اولویت‌های پایین‌تری دارند را در پشته نگه می‌دارد و بعد که نوبت آنها شد آنها را در خروجی نمایش می‌دهد.

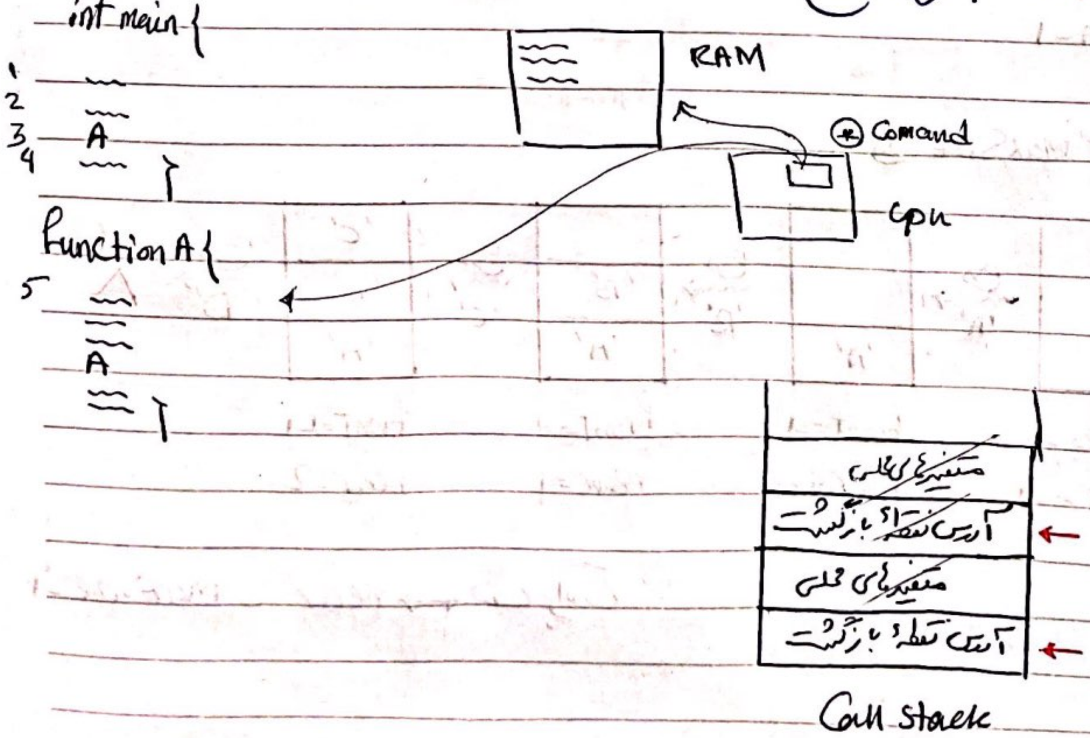
* با استفاده از پشته ترتیب ظاهر شدن عملگرها در خروجی را مدیریت می‌کنیم و هر عملگر دقیقاً یکبار داخل پشته شده و یکبار خارج می‌شود.

میزان مورد نیاز برای این الگوریتم = (طول عبارت) O
← مجموع تعداد عملوندها + عملگرها + پرانتز باز و بسته

? (K+L)-(M*N)+(O^P)*W/V/U*T+Q
→ KL+MN*-OP^W*V/U/T*+Q+



کاربرد پشته در هنگام فراخوانی توابع :



صف (queue):

یک ساختمان داده ترتیبی است (مثل آرایه و پشته) که درج عناصر از انتها و حذف عناصر از ابتدا صورت می‌گیرد. (صف نانوایی ...) سه خاصیت داره وظیفه

FIFO = first input, first output

اولین ورودی، اولین خروجی

* front ← حاوی یک واحد کمتر از اندیس اولین عنصر موجود در صف
* rear ← حاوی اندیس آخرین عنصر موجود در صف
* مقداردهی اولیه ← front = -1 , rear = -1

۱۸ ← (۹/۲):

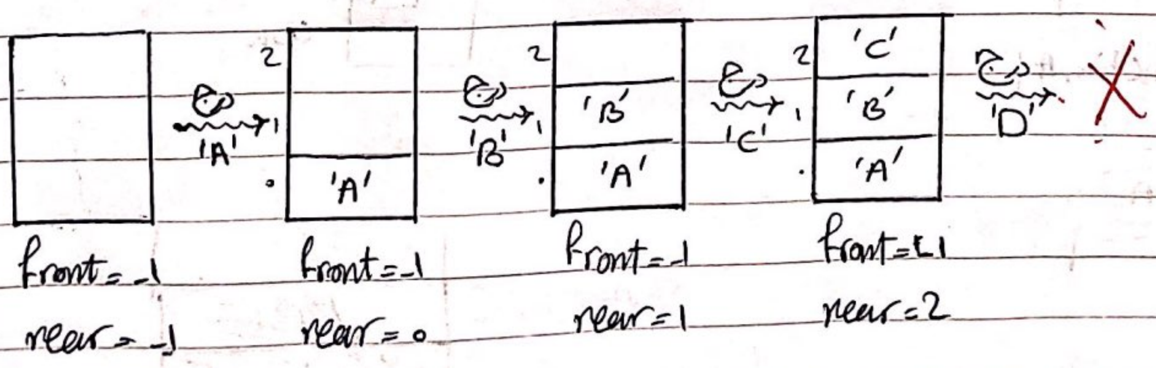
$S = \{a_0, a_1, a_2, \dots, a_{n-1}\}$

a_0 ← عنصر ابتدای صف ، a_{n-1} ← عنصر انتهای صف

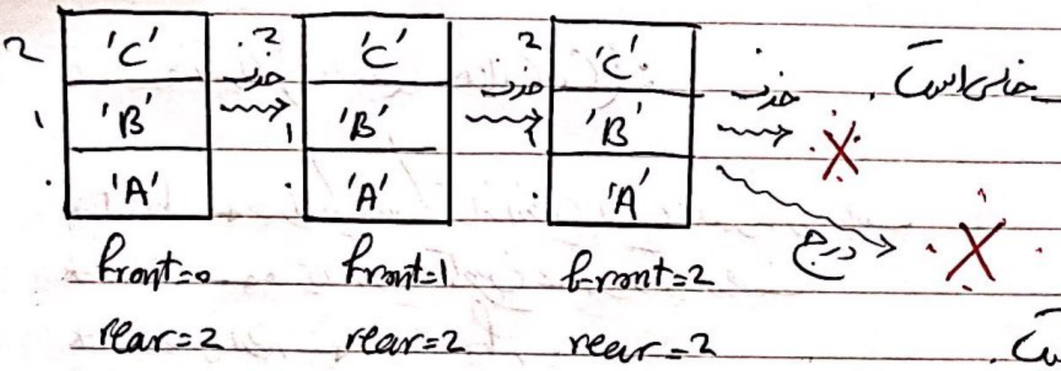
* منهای یک از a_0 قرار می‌گیرد.

front = -1
rear = n-1

Const int MaxSize = 3



rear = MaxSize-1 ←→ صف پر است.



front = rear ← صف خالی است .

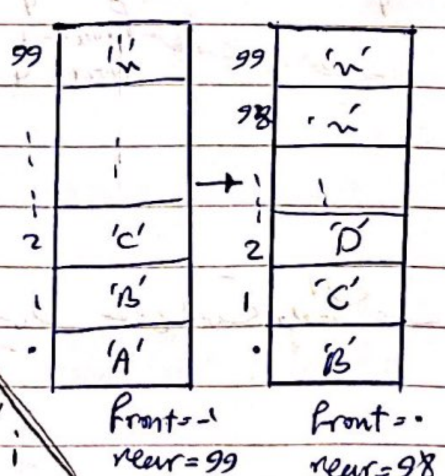
* صف خطی یکبار مصرف است .

* front = 1 ← یعنی اولین عنصر موجود در صف در خانه ای با اندیس 2 است .

* front == rear ← یک واحد کمتر از اندیس اولین عنصر صف = اندیس آخرین موجود در صف = صف خالی است

① * چون که عمل حذف صورت بگیرد اندیس تمام عناصر موجود در صف یک واحد کم شود و این ترتیب تمام عناصر موجود در صف یک واحد به سمت ابتدای صف شیفت پیدا کنند و برای درج عناصر جدید مکان هست .

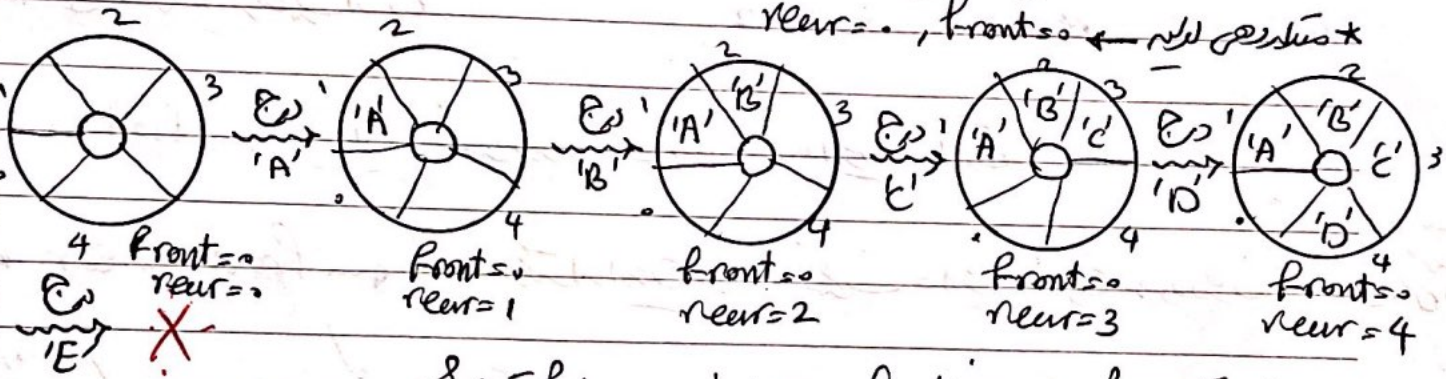
↓ O(n)



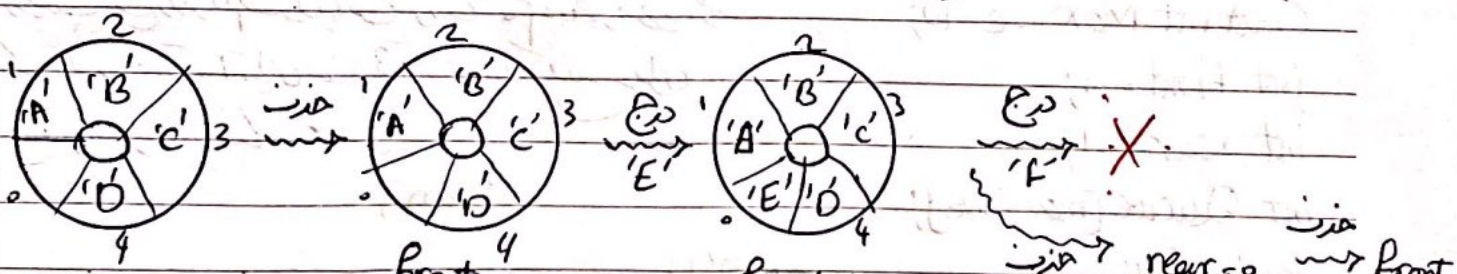

```
class Queue{
Const int MaxSize=10;
int front=-1;
int rear=-1;
int Queue[max_Size];
bool isFull(){                              → O(1)
if(rear==MaxSize-1) return true;
return false; }
bool isEmpty(){                             → O(1)
if(front==rear) return true;
return false; }
void insert(int n){                         → O(1)
if (isFull()) {Cout<<"the Queue is full";
return; }
Queue[++rear]=n; }
int delete (){                              → O(1)
if (isEmpty) {Cout<<"the Queue is Empty";
return; }
front++;
int n=Queue[front];
return n; }
```

صف حلقوی (Circular Queue):

* front ← جایی یک واحد کمتر از اندیس اولین عنصر موجود در صف
* rear ← جایی اندیس آخرین عنصر موجود در صف
* شروع اولیه ← rear=0 , front=0



* در صف حلقوی همیشه باید بین front, rear فاصله وجود داشته باشد.



در این حالت 2 خانه وجود دارد.

5%5=0 ← front

در آن درج کرد.

* اگر front==rear باشد صف خالی است و امکان حذف وجود ندارد.

* ((rear+1)%maxSize==front) ← صف پر است.

```
class cQueue{
Const int maxSize=6;
int front=0;
int rear=0;
int cQueue[maxsize];
bool isFull(){                    →O(1)
if((rear+1)%maxSize==front) return true;
return false;}
bool isEmpty(){                   →O(1)
if(front==rear) return true;
return false;}
void insert(int x){               →O(1)
if(isFull()){cout<<"Queue is full";
return;}
k=(rear+1)%maxSize;
cQueue[k]=x;
rear=k;}
int delete(){
if(isEmpty()){cout<<"the Queue is Empty";
return;}
front=(front+1)%maxSize;
int x=cQueue[front];
return x;}
```

جبرانی: 19 ← (9,4)

کاربردهای صف:

1- مدیریت صف برای منابع مشترک (CPU, Ram, hard, IO, ...) ← منابع مشترک

resource ← بازدیدکننده

2- صف‌های کامپیوتری (Printer)

Subject:

Year. Month. Date. ( )

صف اولویت دار (priority Queue):

۱- پیاده سازی با آرایه مرتب: درج: O(n)، حذف: O(1)

۲- پیاده سازی با آرایه نامرتب: درج: O(1)، حذف: O(n)

۳- پیاده سازی با آرایه نیمه مرتب: درج: O(log n)، حذف: O(log n)

← بهترین روش ← heap tree

پیاده سازی با آرایه مرتب:

صف اولویتی هست که یک آرایه مرتب از عناصر براساس اولویت به صورت نزولی باشد. اولین عنصر موجود در صف دارای بالاترین اولویت باشد.

12	10	8	7	5	9
'C'	'D'	'A'	'B'	'E'	'K'

✗

عمل درج یک عنصر جدید؟ O(n)

12	10	9	8	7	5
'C'	'D'	'K'	'A'	'B'	'E'

n ← تعداد عناصر موجود در صف ← O(n)

12	10	8	7	5	
'C'	'D'	'A'	'B'	'E'	

با استفاده از آرایه

حذف یک عضو؟ O(1)

پیاده سازی با آرایه نامرتب:

عناصر موجود در صف براساس اولویت مرتب نیستند.

8	7	12	6	10
'C'	'D'	'H'	'E'	'K'

✓

درج یک عضو جدید در صف؟ O(1) مثل صف معمولی چون جایگاه مهم نیست.

8	7	12	6	10
'C'	'D'	'H'	'E'	'K'

→

8	7	6	5
'C'	'D'	'E'	'K'

با استفاده از آرایه

حذف یک عضو؟ O(n)

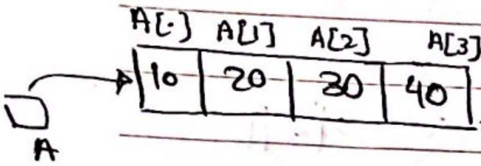
n ← تعداد عناصر موجود در صف

لیست پیوندی (linked list):

ساختاری است شامل ترتیبی از عناصر که ترتیب منطقی بین عناصر از طریق اشاره‌گرها تامین می‌شود.

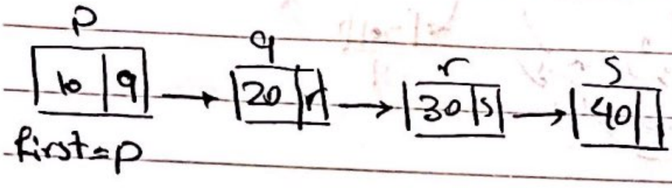
← هر عنصر حاوی آدرس عنصر بعدی می‌باشد.
← نظم فیزیکی و منطقی عناصر با هم یکسان نیست.

آرایه، ضمن لیستی:



A[0] → P
A[1] → P+1
A[2] → P+2
A[3] → P+3

لیست پیوندی:

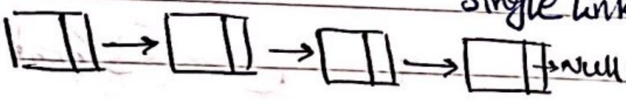


* ساختار تمام گره‌های موجود در یک لیست پیوندی با هم یکسان است.
* میزان حافظه مصرف شده توسط لیست پیوندی (برای ۴ عدد) بیشتر از میزان حافظه مصرف شده برای آرایه است.

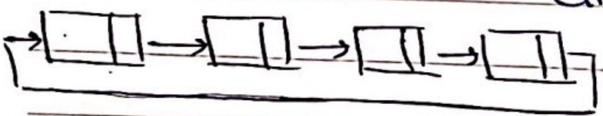
و در ترتیب با هم یکسان نمی‌شوند.

انواع لیست‌های پیوندی:

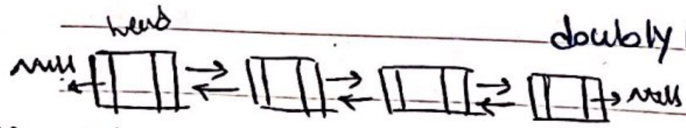
۱_ لیست تک پیوندی (زنجیره) Single linked list



۲_ لیست حلقوی Circular linked list

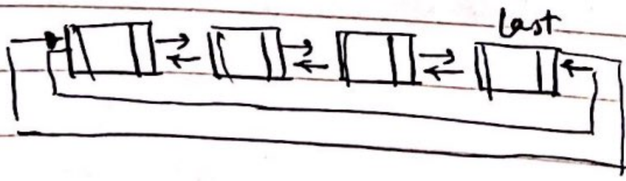


۳_ لیست پیوندی دوطرفه doubly linked list




```
class Node{
 int data;
 Node* next;
 Node* prev;}
```

۴_ لیست پیوندی دوطرفه حلقوی



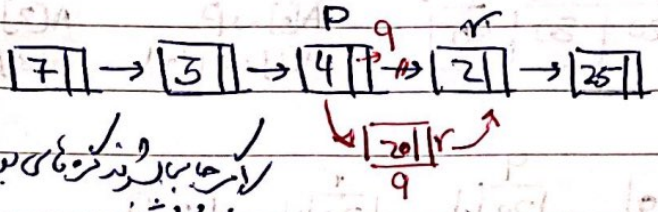
Subject:

Year. Month. Date. ( )

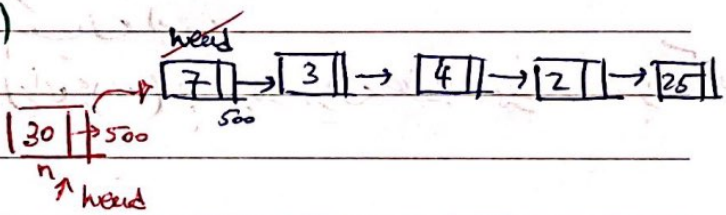
```
class node{
int data;
node* next;};
class linked list{
node* head = NULL;
int size = 0;
void insert(node*p, int n){
node* q = new node;
q->data = n;
q->next = p->next;
p->next = q;
size++;}
void insert(int n){
node* q = new node;
q->data = n;
q->next = head;
head = q;}
void delete (node*p){
if(head == NULL) return;
if(p == NULL){
node*r = head;
head = head->next;
delete r;}
else{
node*q = p->next;
p->next = q->next;
delete q;}
size--;}
```

لیست یک پیوندی: ساختمان داده پویا است.

لیست پیوندی خالی پوچ است.



اگر جای پیوند قبلی و بعدی p عوض شود.



آیا می‌توان قبل و بعدی را عوض کرد؟

```
void deleteR(node*p){   → O(n)
  if(p==head){
    head=p→next;
    delete p;
    size--;
    node*q=head
    while(q→next!=p)
      q=q→next;
    q→next=p→next;
    size--;
    delete p;}
```

آدرس خود گره را داریم! ←

n ← تعداد گره های موجود در لیست پیوندی

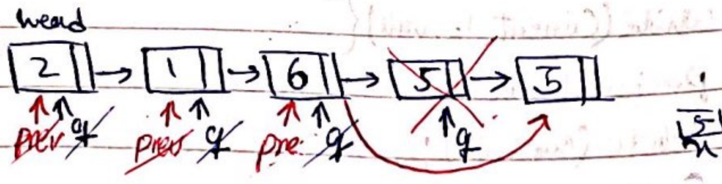
نمیشه هم که آدرس p حذف نشود و دیتا را جابجا کنیم و گره بعدی را حذف کنیم.

۲۱ ← (۹۹):

```
void delete(int n){   → O(n)
  node*q=head;
  node*prev=null;
  if(q!=null && q→data==n){   →
    head=q→next;
    delete q;
    size--;
    return}
  while(q!=null && q→data!=n){
  prev=q;
  q=q→next;}
  if(q==null) return;
  prev→next=q→next;
  delete q;
  size--;}
```

دیتای گره را داریم! ←

دیتای اولین گره برابر با n است.



Subject:
Year. Month. Date. ( )

```
void reverse(){                                   → O(n)
node *current=head;
node *prev=null, *next=null;
while(current!=null){
next=current->next;
current->next=prev;
prev=current;
current=next;}
head=prev;}
```

معکوس کردن لیست پیوندی


```
bool search(int x){                               → غیربازگشتی
node *current=head;
while(current!=null){
if(current->data==x)
return true;
current=current->next;}
return false;}
```

جستجوی عضو در لیست پیوندی

```
bool search(node *p, int x){
if(p==null) return false;
if(p->data==x) return true;
return search(p->next, x);}
```

بازگشتی

```
void travel(){                                    → غیربازگشتی  O(n)
node* current=head;
while(current!=null){
Cout<<"current->data";
current=current->next;}}
```

پیمایش لیست پیوندی

```
void travel(node *p){                             → O(n)
if(p==null) return;
Cout<<p->data;
return travel(p->next);}
```

بازگشتی

```
void deletelist(){                                → غیربازگشتی  O(n)
node* current=head;
node*p;
while(current!=null){
p=current->next;
delete(current);
current=p;}}
```

حذف لیست پیوندی

تابع کپی کردن لیست پیوندی ← O(n)

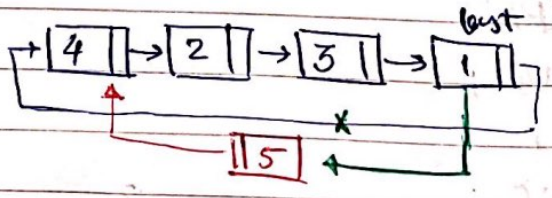
```
Node* Copy (){
if (head == null) return Null;
Node *p = new Node;
p->data = head->data;
Node* head2 = p;
Node* prev = p;
Node *q = head->next;
while (q != null){
Node *n = new Node;
n->data = q->data;
prev->next = n;
prev = n;
q = q->next;}
return head2;}
```

جواب: (۹,۹) ← ۴۲

لیست با دم حلقوی:

```
Class CircularLinkedList{
Node* last = Null;
int Size = 0;
void insert1(int n){        O(1)
Node *p = new Node;
p->data = n;
if (last == Null){
last = p;
last->next = last;
Size++;}
else{ p->next = last->next;
last->next = p;
Size++;}
```

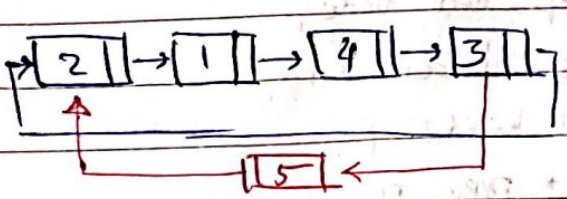
تابع درج عنصر به عنوان اولین گره جدید ←

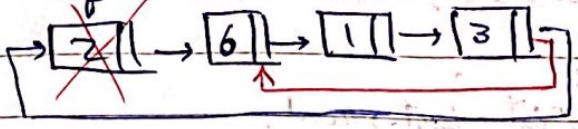


Subject:

Year. Month. Date. ( )

```
void insert2(int n){                                   O(1)
  Node *p = new Node;
  p->data = n;
  if(last == Null) {last = p;
  last->next = last;
  Size++;}
  else { p->next = last->next;
  last->next = p;
  last = p;
  Size++;}
void insert3(Node *p, int n){                          O(1)
  Node *q = new Node;
  q->data = n;
  q->next = p->next;
  p->next = q;
  Size++;}

void delete1(){                                        O(1)
  if(last == Null) return;
  if(last->next == last){
  Node *p = last;
  delete(P);
  last = Null;
  Size = 0;
  return;}
  Node *p = last->next;
  last->next = p->next;
  delete(P);
  Size--;}
```





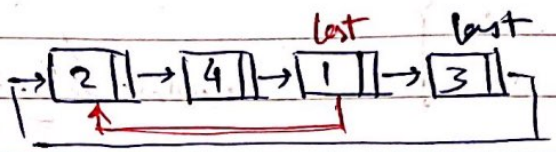
Year. Month. Date. ( )

```
void delete2 (){
  if (last==NULL) return;
  else if (last->next==last){
  node *p=last;
  delete (p);
  last=NULL;
  size=0;}
  else{
  node *q=last;
  while (q->next!=last)
  q=q->next;
  q->next=last->next;
  node *p=last;
  delete (p), size--;
  last=q;}
```

$\Theta(n)$ ← حذف آخرین گره

یک گره داریم !

بیشتر از یک گره داریم !



حذف گره ای ← حذف شده از لیست پیوندی یکره ای

```
void delete3 (int n){
  if (last==NULL) return;
  else if (last->next==last && last->data==n)
  node *q=last;
  delete (q);
  last=NULL;
  size=0;}
  else if (last->next!=last && last->data==n)
  node *q=last;
  while (q->next!=last)
  q=q->next;
  q->next=last->next;
  node *n=last;
  delete (n);
  last=q;
  size--;}
```

حذف گره با محتویات داده شده n ←

حذف آخرین گره وقتی یک گره داریم !

حذف آخرین گره وقتی بیشتر داریم !

NEGAR

Subject:
Year. Month. Date. ( )

```
else{                                                    O(n)
Node*p=last->next;
Node* prev=Null;
while(p!=last && p->data!=n){
 prev=p;
 p=p->next;}
if(p==last) return;
prev->next=p->next;
delete(p);
 size--;}}
```

هدف: پیمایش و جستجوی لیست حلقوی به لیست تک طرفه است.

```
bool isEqual(Node* last1, Node* last2){
if((last1==Null && last2!=Null) ||
   (last2!=Null && last2==Null)) return false;
if((last1==Null && last2==Null)) return true;
Node*p=last1->next;
Node*p=last2->next;
while(p!=last1 && q!=last2){
if(p->data!=q->data) return false;
 p=p->next;
 q=q->next;}
if(p==last1 && q==last2){
if(p->data==q->data) return true;
 else                 return false;
if((p==last1 && q!=last2) || (p!=last1 && q==last2)) return false;}
```

بررسی تساوی دو لیست حلقوی ← عناصر و ترتیب یک‌سان باشند.

O(min(طول لیست))

۱) ادغام دو لیست حلقوی صعودی ← عکس ت

۲۳ ← (۹٫۱۴):

لیست دوطرفه:

```
Class Node{
 int data;
 Node* next;
 Node* prev;}

 Class DoublyLinkedList{
 Node* head = Null;
 int size = 0;
```

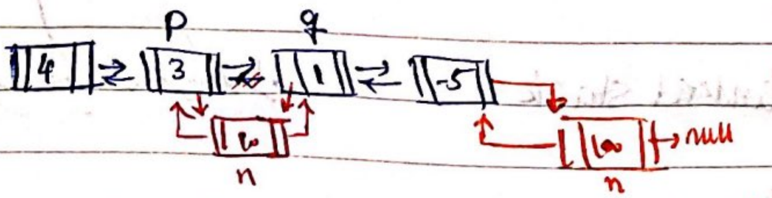
درج گره به عنوان اولین گره جدید با ریسک n ← Θ(1) ← O(1)

```
⊖void insert1 (int n){
 Node *p = new Node;
 p→data = n;
 p→next = head;
 p→prev = Null;
 if (head != Null)  head→prev = p;
 head = p;
 size++;
```

اولین گره موجود داریم یا خیر!

درج گره میانی ←

```
⊖ Void insert2 (int n, Node *p){      Θ(1)
 Node *n = new Node;
 n→data = n;
 Node *q = p→next;
 n→next = q;
if(q) { q→prev = n; }
 n→prev = p;
 p→next = n;
 size++;}
```



درج گره به عنوان آخرین گره ←

(1)

فرض کنید دو لیست حلقوی داریم که هر کدام ترتیب عناصر به صورت صعودی است. می‌خواهیم دو لیست را ادغام کنیم که لیست حاصل هم مرتب شده صعودی باشد.

```
Node * merge(Node * last1 , Node * last2) {
if ( last1 = = Null && last2 = = Null) return Null;
if ( last1 ! = Null && last2 = = Null) return last1
if ( last1 = = Null && last2 ! = Null) return last2

Node * first, * prev = Null, * last3 = Null
Node* p = last1 -> next ; Node * q = last2 -> next.
bool flag = true
while ( P! = last1 && q! = last2) {
Node * n = new Node;
 if ( p -> data < q -> data) { n -> data = p -> data; p = p -> next; }
elseif ( p -> data = = q -> data) {n -> data = p -> data; p = p -> next; q = q -> next; }
 else { n -> data = q -> data ; q = q -> next; }
 if (flag = = true) {
   first = n;
   flag = false }
 else { prev -> next = n; }
 prev = n
} // end of while loop.

if ( p = = last1 && q! = last2) {
  while ( q! = last2) {
    Node * n = new Node; n -> data = q -> data; q = q -> next
    if ( flag = = true) { first = n ; flag = false }
    else { prev -> next = n }
    prev = n } }
```

```
② else if (p! = last1 && q = = last2) {
    while( p! = last1) { Node * n = new node; n → data = p → dat
     if (flag = = true) { first = n; flag = false; }
     else { prev → next = n; }
     p = p → next
     prev = n } }
Node * n = new node;
 if (p → data <= q → data) { n → data = p → data;
     if ( flag = = true) { first = n; flag = false; }
     else { prev → next = n; }
     prev = n;
Node * n = new node; n → data = q → data; prev → next = n;
  last3 = n; } // end of if
else { n → data = q → data;
  if ( flag = = true ) { first = n; flag = false; }
  else { prev → next = n }
  prev = n;
Node * n = new node;
 n → data = p → data; prev → next = n;
   last3 = n;
} // end of else
last3 → next = first;
return last3 }
```

زمان اجرای در بدترین حالت:

O(مجموع طول دو لیست پیوندی)

زمان اجرا در بهترین حالت:

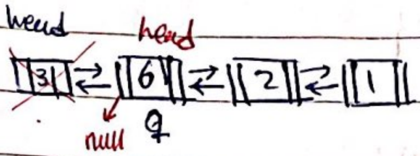
Ω(مجموع طول دو لیست پیوندی)

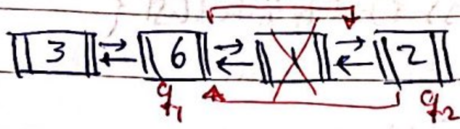
← زمان اجرا: Θ(مجموع طول دو لیست پیوندی)

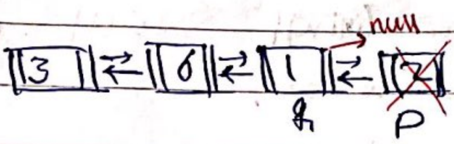
```
int delete(node *p){
 if(p==head){
  node *q=head->next;
  q->prev=null;
  head=q;
  size--;     int x=p->data;
  delete(p);   return x;
 else{     int x=p->data;
 node *q1=p->prev;
 node *q2=p->next;
 q1->next=q2;
 if(q2) {q2->prev=q1}
  delete(p);
  size--;
  return x;}
```

$\Theta(1)$

حذف اولین گره با آدرس p هست + میانی







لیست دوطرفه حلقوی:

* گره رأسی: یک گره ای هست که داده معتبر نیست و حتی وقتی لیست خالی است این گره وجود دارد و هر گره جدیدی که می خواهد اضافه شود بعد از این گره head اشاره شد و ثابت است.

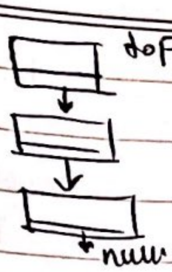
کاربردهای لیست پیوندی:

۱- پشته پیوندی: پیاده سازی با استفاده از لیست یک طرفه. linked stack

۲- صف پیوندی: " " " " " " " linked queue

۳- ذخیره چندجمله ای اسپارس: " " " " " "

۴- ماتریس های اسپارس:



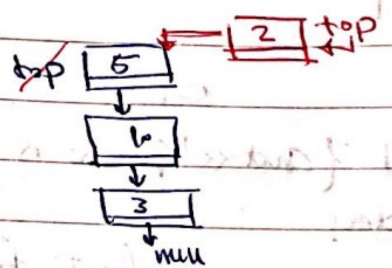
① پشته پیوندی: top = Null ← پشته خالی است.

پشته پیوندی بهتر از پشته ترتیبی است زیرا نیازی به (maxsize) ندارد و در حین اجرای برنامه می‌توان سایز پشته را تغییر داد.

* حذف و درج فقط از بالای پشته صورت می‌گیرد.

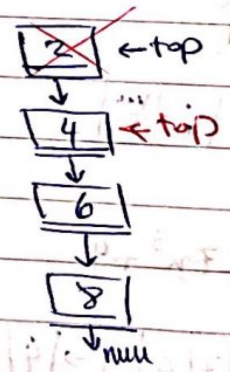
درج ←

```
void push(int n){          O(1)
  Node *p = New Node;
  p -> data = n;
  p -> next = top;
  top = p;}
```



حذف ←

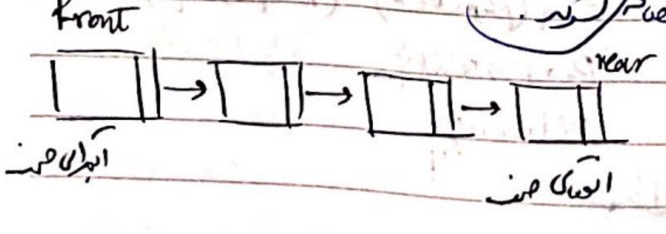
```
int pop(){                  O(1)
  if(top == Null)  {Cout << "stack is Empty" ;}
  Node *p = top;
  top = top -> next;
  int x = p -> data;
  delete(p);
  return x;}
```



② صف پیوندی:

صف پیوندی بهتر از صف ترتیبی است زیرا ظرفیت از پیش مشخص ندارد.

* از ابتدای صف عناصر حذف می‌شوند و از انتهای عناصر اضافه می‌شوند.

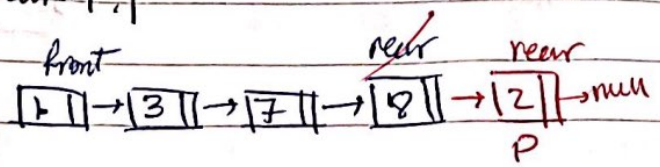


front = rear = null ← صف خالی است.

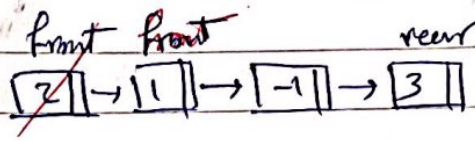
Subject:

Year. Month. Date. ( )

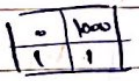
```
void add (int n){                                   Θ(1)
node *p = new node;
p->data = n;
if(front == null) {front = p; rear = p;}
else { rear->next = p;
       rear = p;}
     p->next = null;}
```




```
int delete (){                                      Θ(1)
if(front == null) {cout << "queue is empty"; return;}
int n = front->data;
node *p = front;
front = front->next;
delete (p);
return n;}
```

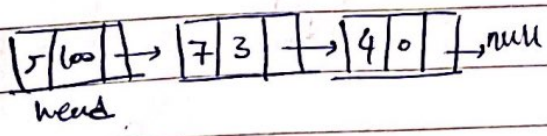


$p(x) = x^{1000} + 1$



③ نحوه ذخیره چند جمله‌ای اسپارس ← روش ترتیبی ← آرایه دو بعدی
روش پیوندی ← استفاده از لیست

$p(x) = 5x^{1000} + 7x^3 + 4$




```
class node{
int coef;      ضریب
int exp;       توان
node *next;}
```

```
float playeral (float x, node *head){
float S = 0;
node *p = head;
while(p != null){
S = S + (p->coef) * (x ^ p->exp);
p = p->next;}
returning;}
```

← طول جمله‌ای اسپارس ← زمان

:(۹,۱۶) ← ۲۴

① جمع دو چند جمله‌ای

```
Node* ADD(Node* first1, Node* first2){
if(first1==Null && first2==Null) return null;
if(first1!=Null && first2==Null) return first1;
if(first1==Null && first2!=Null) return first2;

Node *p=first1, Node *q=first2;
Node *prev=Null, first3;
bool flag=true;
while(p!=Null && q!=Null){
 if(p->exp==q->exp){
  if(p->coef+q->coef!=0){
   Node*n=new Node;
   n->exp=p->exp;
   n->coef=p->coef+q->coef;
   if(flag){
   first3=n; flag=false;}
   else{
   Prev->next=n;} prev=n;}
   p=p->next;
   q=q->next;}

else if(p->exp>q->exp){
Node *n=new Node;
n->exp=p->exp;
n->coef=p->coef;
if(flag){first3=n; flag=false;}
else{prev->next=n}
 prev=n;
 P=p->next;}
```

توان p و q باهم برابرند.

توان p بزرگتر بود.

مکان q بزرگتر باشد.

```
else {
Node *n = new Node;
n->exp = q->exp;
n->coef = q->coef;
if(flag) {first3 = n; flag = false;}
 else {prev->next = n;}
 prev = n;
 q = q->next; } }   // end of while

if(p != null){
 while(p != null){
 Node *n = new Node;
 n->exp = p->exp;
 n->coef = p->coef;
 if(flag) {first3 = n; flag = false;}
 else {prev->next = n;}
  prev = n;
  p = p->next; } }.

if(q != null){
while(q != null){
Node *n = new Node;
 n->exp = q->exp;
 n->coef = q->coef;
if(flag) {first3 = n; flag = true;}
 else{ prev->next = n;}
 prev = n;
 q = q->next;}}
return first3; }
```

④ ماتریس های اسپارس — روش ترتیبی، در دسترسی ... عکس ت.

— پیوندی ← ۱) روش اول
۲) ~ دوم
۳) ~ سوم.

(5)

suppose that A(m*t) and B(t*n) → we want to compute A*B

A, B هردو آرایه‌ی سه‌ستونی در حافظه ذخیره شده‌اند

ابتدا ترانهاده ماتریس B را حساب می‌کنیم:

A:
5	8	4
1	2	10
1	3	12
2	1	1
2	3	2

B:
8	5	4
1	1	2
1	2	5
2	2	1
3	1	8

→ B^T =
5	8	4
1	1	2
1	3	8
2	1	5
2	2	1

C =
5	5	
1	2	10
1	1	8417
2	1	2
2	2	5

8

درایه‌هایی که مؤلفه ستون دوم یکسان داشته باشند هم ضرب می‌شوند

```
Transpose(B);
for(int i=0; i<m; i++)
    for(int j=0; j<n; j++)
        C[i][j]=0
for(int i=1; i<=NA; i++){          // تعداد درایه‌های غیرصفر ماتریس A
    for(int j=1, j<=NB; ) ++){     // تعداد درایه‌های غیرصفر ماتریس B
        if (A[i][1] == B[j][1])
        {
            C[A[i][0]][B[j][0]] += A[i][2] * B[j][2]
        }
```

Summation of multiplied values:

result[1][1] = A[1][3]*B[1][3] = 12*8 = 96
result[1][2] = A[1][2]*B[2][2] = 10*1 = 10
result[2][1] = A[2][1]*B[1][1] + A[2][3]*B[1][3] = 2*1 + 2*8 = 18
result[2][2] = A[2][1]*B[2][1] = 1*5 = 5

Any other element cannot be obtained
by any combination of row in
Matrix A and Row in Matrix B.

Hence the final resultant matrix will be:

Row	Col	Val
1	1	96
1	2	10
2	1	18
2	2	5

1

Monday, November 2, 2020 11:38 PM

```
//transpose the matrix B:
For(int j=1,i<NB;j++)
      swap(B[-][0],B[-][1])
//Sort
For(int j=NB-1,j>1,j--)
  For(int j=1;j<;j++)
     {
                if(B[j][0]>B[j-1][0])
                  {
                        Swap(B[-][0],B[j-1][0])
                        Swap(B[-][1],B[j-1][1])
                        Swap(B[-][2],B[j-1][2])
                  }
     }

temp1=1
While(temp1<NB)
{
      For(int j=temp1,j=NB,j++)
      {
            temp2=temp1
            if(B[j][0]==B[j-1][0])
              temp2++;
            else
              break;

              For(int j=temp2-1,j>temp1,j--)
                For(int j=temp1,j<j,j++)
                  {
                    if(B[][1]>B[j-1][1])
                      {
                            Swap(B[][1],B[j-1][1])
                            Swap(B[][2],B[j-1][2])
                      }
                  }
      }
      Temp1=temp2+1
}
```

الف

ب

ج

50	50	22
0	1	2
0	5	6
1	3	8
1	5	10
1	10	6
5	2	1
5	9	3
5	16	6
5	32	5
8	5	-17
8	19	1
8	9	4
8	48	6
10	0	7
10	4	19
10	8	77
10	16	-96
10	5	-6
25	9	2
25	19	3
25	23	1
25	48	1

50	50	22
1	0	2
5	0	6
3	1	8
5	1	10
10	1	6
2	5	1
9	5	3
16	5	6
32	5	5
5	8	-17
19	8	1
9	8	4
48	8	6
0	10	7
4	10	19
8	10	77
16	10	-96
5	10	-6
9	25	2
19	25	3
23	25	1
48	25	1

50	50	22
0	10	7
1	0	2
2	5	1
3	1	8
4	10	19
5	8	-17
5	0	6
5	1	10
5	10	6
8	10	77
9	25	2
9	8	4
9	5	3
10	1	6
16	10	-96
16	5	6
19	25	3
19	8	1
23	25	1
32	5	5
48	8	6
48	25	1

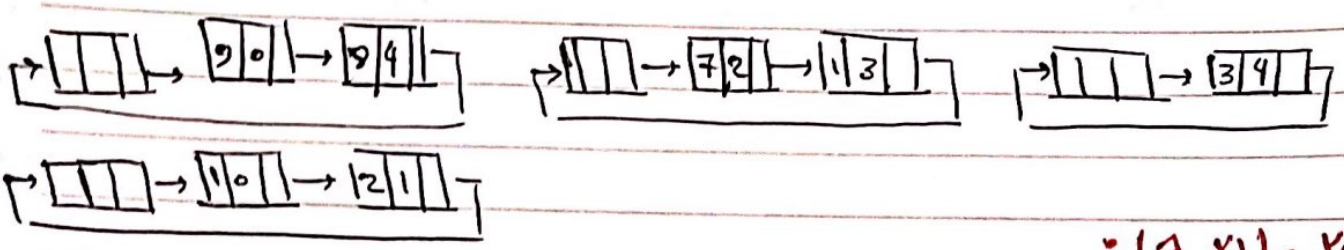
50	50	22
0	10	7
1	0	2
2	5	1
3	1	8
4	10	19
5	0	6
5	1	10
5	8	-17
5	10	-6
8	10	77
9	5	3
9	8	4
9	25	2
10	1	6
16	5	6
16	10	-96
19	8	1
19	25	3
23	25	1
32	5	5
48	8	6
48	25	1

۱- روش اول:

به ازای هر درایه ناصفر در ماتریس اسپرس یک گره خواهیم داشت.
روش خوبی نیست زیرا پردازش های دستوری سخت است و نمی توانیم به عناصر ستون ها دسترسی داشته باشیم.

```
class node{
int value;
int Col;
node *next;}
```

$$A=\begin{bmatrix} 9 & 0 & 0 & 8 \\ 0 & 0 & 7 & 1 & 0 \\ 0 & 0 & 0 & 3 \\ 1 & 2 & 0 & 0 & 0 \end{bmatrix}_{4\times 5}$$

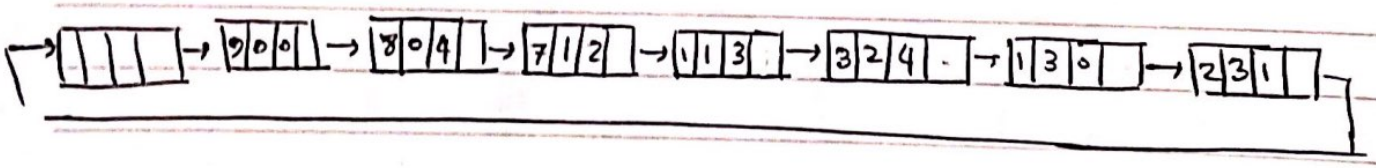


(۹,۲۱) ← ۲۵ :

۲- روش دوم:

از یک لیست برای ذخیره کل ماتریس اسپرس استفاده شود.
ساختار هر گره علاوه بر شماره ستون، شماره سطر را هم دارد.
روش خوب نیست زیرا مثل روش اول امکان دسترسی به درایه های موجود در یک ستون به طور مستقیم وجود ندارد.

```
class node{
int value;
int row, Col;
Node* next; }
```



۳- روش سوم:

بهترین روش!
عکس ت لیست دسترسی تقسیم بندی.

```
class node{
int value;
int row, Col;
Node *right;  اشاره گر به گره های درایه ناصر بعدی در سطر
Node* Down;}  اشاره گر به گره های درایه ناصر بعدی در ستون
```

NEGAR

h. Date. ( )

6:00

ضرب دو ماتریس با شکل روش سوم سه عکس تو

①

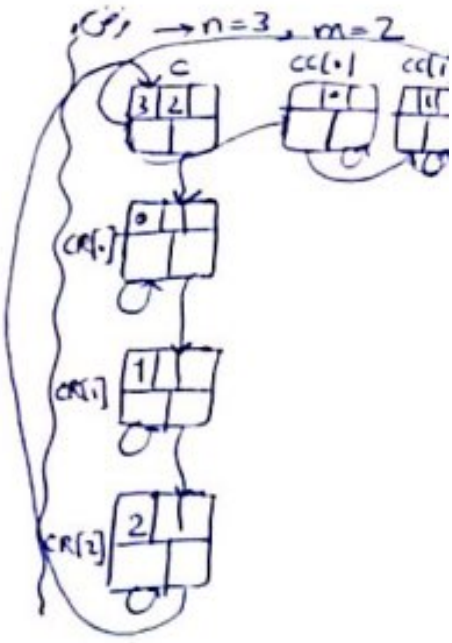
```
Node * multi( Node *A , Node *B){
if( A == NULL || B == NULL) return NULL
const int n = A -> Row;
const int m = B -> col;
if( A -> col != B -> Row) return NULL; //
Node *c = new Node;
c -> Row = n
c -> col = m
Node *CR[n];
Node *cc[m]
for (int i =0; i <n; i++){
    CR[i] = new Node
    CR[i] -> Row = i
    if (i>1)    CR[i-1] -> Down = CR[i]
    CR[i] -> Right = CR[i]
}
    CR[n-1] -> Down = C
    C -> Down = CR[i]
for( int j=0; j<m; j++) {
  cc[j] = new Node;
  cc[j] -> col = j
if(j>1)   cc[j-1] -> Right = cc[j]
  cc[j] -> Down = cc[j]
}
  C -> Right = cc[1];
  cc[m-1] -> Righ = C,
```

تعداد ستون‌های ماتریس اول برابر با تعداد سطرهای ماتریس دوم نیست

→ مقداردهی اولیه

سطرهای ماتریس حاصلضرب برابر با تعداد سطرهای ماتریس اول است

ستون‌های ماتریس حاصلضرب برابر با تعداد ستون‌های ماتریس دوم است



(2)

```
Node * lastRow[n];
Node * lastCol[m];
```

متغیرهایی برای ذخیره آدرس آخرین گره‌ها در سطر و ستون که تاکنون ساخته شده است.

```
for (int i=0; i<n; i++)
    lastRow[i] = cR[i];
for (int j=0; j<m; j++)
    lastCol[j] = cC[j]
```

در حال حاضر آخرین گره‌های ساخته شده در هر سطر و ستون

```
for (int i=0; i<n; i++)
    for (int j=0; j<m, j++){
        int sum=0
        Node * p = A -> Down;
        while (p -> Row != i)
            p = p -> Down
        Node * q = B -> Right
        while (q -> col != j)
            q = q -> Right
        Node * pR = p -> Right
        Node * qC = q -> Down
        while (pR != p && qC != q){
            if (pR -> col == qC -> Row){
                sum += (pR -> value) * (qC -> value)
                pR = pR -> Right
                qC = qC -> Down
            }
```

می‌خواهیم p حاوی آخرین گره‌ی واقع در سطر i در ماتریس A باشد

می‌خواهیم q حاوی آخرین گره‌ی واقع در ستون j در ماتریس B باشد.

(3)

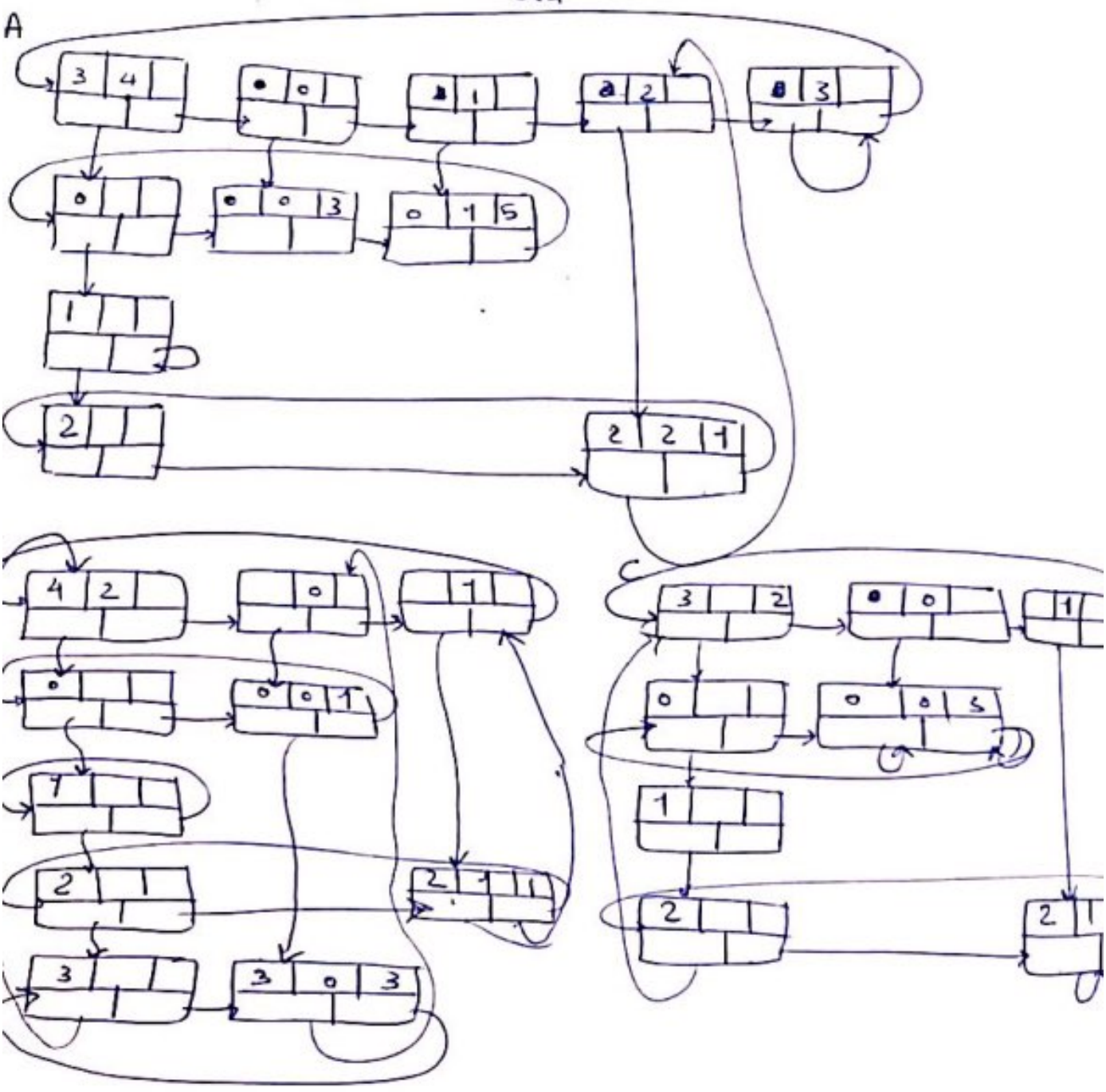
```
else if ( PR -> col < qc -> Row )
        PR = PR -> Right
else
        qc = qc -> Down
{

if ( sum != 0 ){
  Node * n = new Node ;
    n -> Row = i
    n -> col = j
  n -> value = sum
  lastRow[i] -> Right = n
  last col[j] -> Down = n
  lastRow[i] = n
  last col[j] = n
  n -> Right = CR[i]
  n -> Down = cc[j] ;
}
}
  return  c ;
}
```

④

$$A = \begin{bmatrix} 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 3 & 0 \end{bmatrix}_{4\times 2}$$

3×4

A



درخت ها

درخت دودویی (binary tree) حالت خاصی از درخت عمومی (general tree)

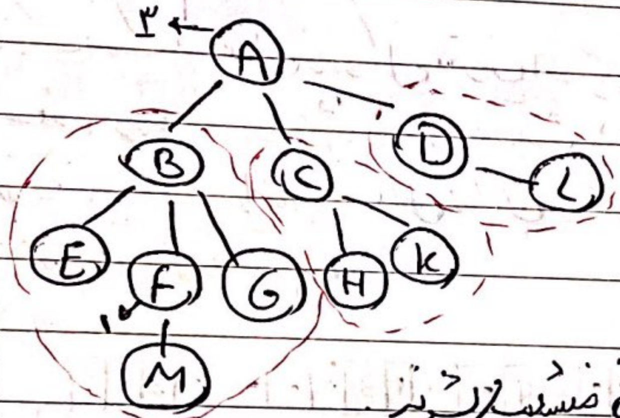
درخت مجموعه ای متناهی از عناصر که ساختار سلسله مراتبی دارند یعنی هر عنصر در یک سطح از این ساختار سلسله مراتبی قرار گرفته است.

درخت: مجموعه ی متناهی از یک یا چند گره: دارای یک گره ی خاص به نام ریشه (root)
دارای k زیر درخت ($T_1, T_2, ..., T_k$) ($k \geq 0$)

→ هر زیر درخت خودش یک درخت است.

درخت یک ساختمان داده ی بازگشتی است.

گراف همبند بدون دور ← درخت



* ریشه (root): گره ای که در بالاترین سطح و هر گره ای از آن منشعب می شوند

* درجه گره (تعداد فرزندان گره): تعداد زیر درخت های گره

* فرزند: گره هایی که یک سطح پایین تر اند و بطور مستقیم به گره متصل هستند (همه ی فرزندان در یک سطح هستند)

* عمق درخت (ارتفاع درخت): تعداد سطوح در ساختار سلسله مراتبی (یک واحد بیشتر از بیشترین عمق)

* والد / * هم زاد: گره هایی که والد یکسان دارند. / * درجه درخت: بزرگترین درجه گره های درخت

* برگ: گره هایی که فرزند ندارند (همه گره هایی که در آخرین سطح هستند برگ هستند)

درخت دودویی: درختی است که درجه هر گره حداکثر ۲ است، هر گره حداکثر ۲ فرزند دارد، فرزند چپ و فرزند راست.

① درخت دودویی می‌تواند تهی یا لانه‌ی درخت عمومی نمی‌تواند.
② در درخت دودویی می‌توان گفت فرزند چپ و راست دارد در درخت عمومی نمی‌توان گفت.

قضیه: n_0 ← تعداد گره‌های درخت دودویی با درجه صفر
n_1 ← تعداد گره‌های درخت دودویی با درجه ۱
n_2 ← تعداد گره‌های درخت دودویی با درجه ۲

$\rightarrow n_0 = n_2 + 1$

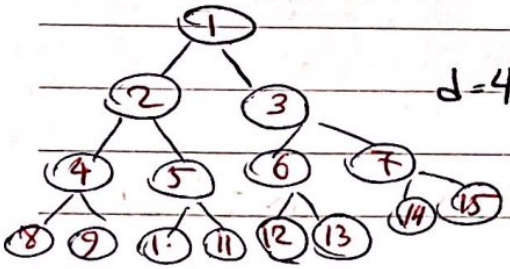
n ← تعداد گره‌های درخت: $n = n_0 + n_1 + n_2$

B ← تعداد کل یال‌ها (انشعاب‌ها): $B = n - 1$, $B = n_1 + 2n_2$

$\Downarrow$ $n - 1 = n_1 + 2n_2$

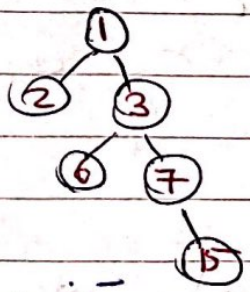
$n_0 + n_1 + n_2 - 1 = n_1 + 2n_2 \rightarrow n_0 = n_2 + 1$

درخت دودویی پر (Full binary tree):
هر گره غیر برگ آن درجه ۲ است و تمام سطوح پر است.



d=4

* برگ‌ها در پایین‌ترین سطح هستند.



غیر پر

قضیه: T درخت دودویی پر با عمق d یال؟

الف) تعداد گره‌ها در سطح i: 2^i
ب) تعداد کل گره‌های درخت $= 2^{d+1} - 1$

تعداد گره‌ها در سطح i: $N(i)$

$N(0) = 1$

$N(i) = 2 * N(i-1) = 2N(i-1) = 2^2 N(i-2) = 2^3 N(i-3) = \dots = 2^i * N(i-i)$

$N(i) = 2^i$

Subject:

Year. Month. Date. ( )

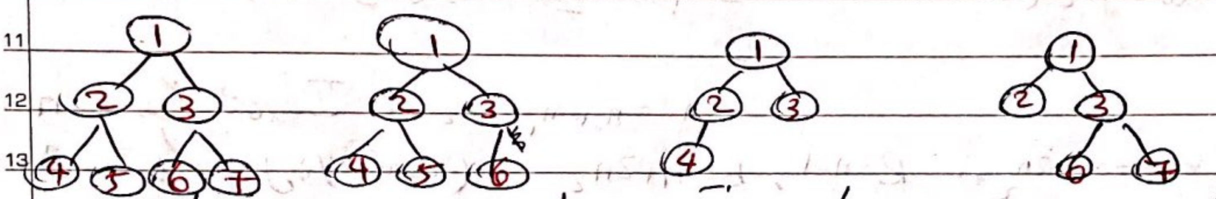
تعداد گره ها : $N(0) + N(1) + N(2) + \dots + N(d-1)$

$2^0 + 2^1 + 2^2 + \dots + 2^{d-1} = 2^d - 1$

درخت دودویی کامل

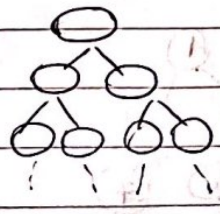
- اگر در یک درخت دودویی پر، برخی از گره های انتهایی که سمت راست اند حذف شوند، درخت دیگری حاصل می شود که به آن درخت کامل می گویند.

- درخت T به عمق d و n گره کامل است هرگاه گره های درخت را مطابق با شماره گذاری درخت پر شماره گذاری کنیم به ازای هر شماره بین 1 تا n، گره با آن شماره در درخت داریم.



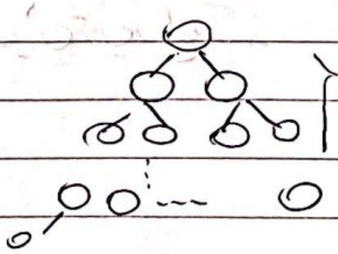
قضیه: اگر T یک درخت دودویی کامل با n گره باشد ←

$d = \lceil \log_2^{(n+1)} \rceil$



درخت پر به عمق d

$n \le 2^d - 1$



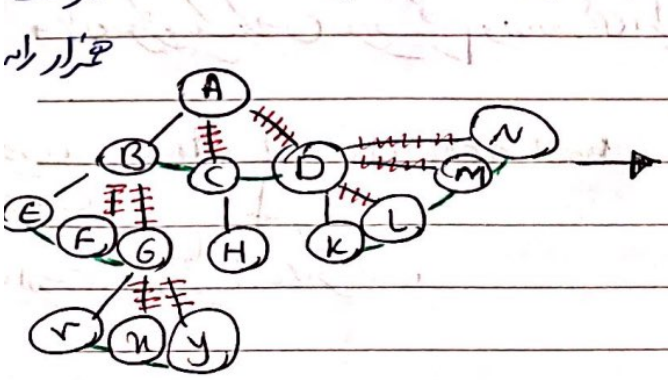
$n \ge 2^{d-1} - 1 + 1$

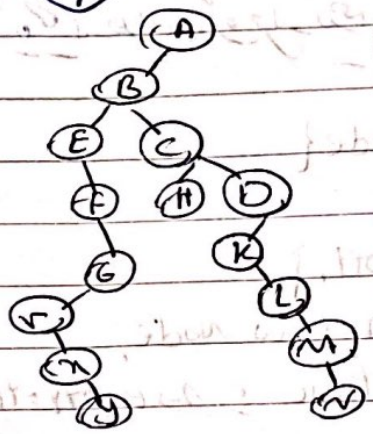
$2^{d-1} \le n \le 2^d - 1 \rightarrow 2^{d-1} \le n+1 \le 2^d \rightarrow d-1 \le \log_2^{n+1} \le d \rightarrow$

$d = \lceil \log_2^{n+1} \rceil$

تبدیل درخت عمومی به درخت دودویی:

ارتباط هر گره با بچه ی فرزندانش به غیر از چپ ترین را حذف می کنیم.
هر گره را به فرزند سمت راست متصل می کنیم. در این صورت هر گره حداکثر به یک خواهد داشت. (چپ ترین با فرزند راست)





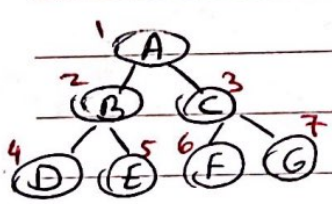
روش های نمایش یا ذخیره یک درخت دودویی در حافظه

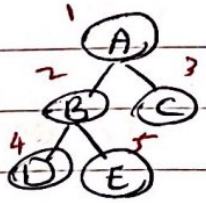
ترتیبی ← آرایه (ساده ترین)

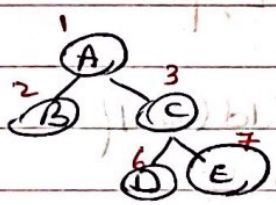
پیوندی ← لیست پیوندی

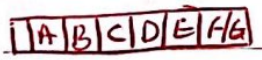
۱) روش ترتیبی ← استفاده از آرایه: (برای درخت دودویی پر و کامل بهتره)

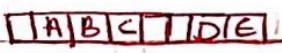
۱- گره های درخت دودویی را طبق شماره گذاری درخت دودویی پر شماره می زنیم.
۲- فرض کنید که اولین شماره ۱ باشد.
۳- آرایه ای به طول n+1 از نوع گره های درخت ایجاد می کنیم.
۴- محتویات گره با شماره i در خانه با اندیس i قرار می دهیم.



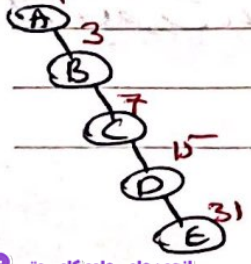


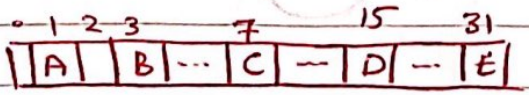






* بدترین حالت زمانی اتفاق می افتد که همه گره ها فرزند راست داشته باشند (اریب به سمت راست)
* زمانی این روش مفید است که درخت دودویی، درخت کامل باشد.





NEGAR

Subject:

Year. Month. Date. ( )

* از روی جنس داده‌ها می‌شود تشخیص داد که کدام خانه‌ها خالی است (با ۱- پر کنیم) ولی فقط برای اعداد صحیح ⊕ کاربرد دارد.

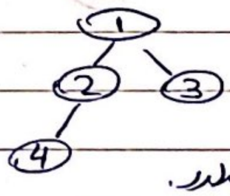
* برای هر گره یک کلاس تعریف کنیم و در خانه‌های آرایه اشیایی از کلاس گره قرار بدهیم → منطقی‌تر کلی

```
class Node{
char ch;
bool empty;};
Node *n=new Node;
n->empty=false  یا  n->empty=true;
```

bool isEmpty(int i) → true: خانه اندیس i حاوی گره نیست
→ False: خانه با اندیس i خالی نیست و حاوی گره است.

گره در خانه با اندیس i ذخیره شده است → والد؟ فرزند چپ؟ / فرزند راست؟

```
int parent(int i){                 O(1)
if(i==1)  return -1;   → ریشه پدر ندارد.
 return (i/2);}        → عدد صحیح رو بده.

int Lchild(int i){                 O(1)
 int L=2*i;
 if(L>n || isEmpty(L)) return -1;
 return L;}

int Rchild(int i){                 O(1)
 R=2*i+1;
 if(R>n || isEmpty(R)) return -1;
 return R;}
```



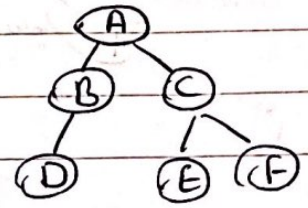
فرزند چپ ③ وجود ندارد.

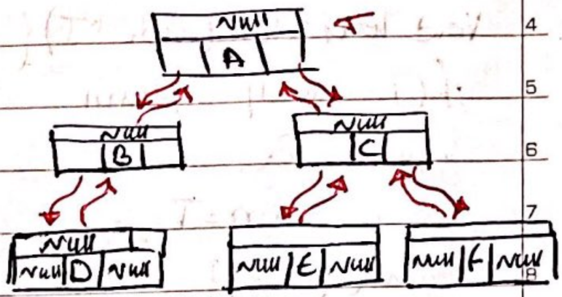
* عیب عمده استفاده از آرایه برای ذخیره‌سازی درخت‌ها زمانی هست که درخت اریب یا درخت غیر کامل باشد که در این صورت باعث هدر رفت حافظه می‌شود.

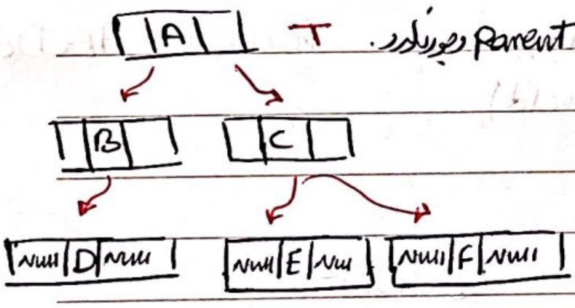
۲) روش پیوندی ← استفاده از لیست پیوندی

* بیشتر زمانی استفاده می‌شود که درخت دودویی غیرکامل را بخواهیم ذخیره کنیم.

```
Class Node{
int data;
Node *Lchild;
Node *Rchild;
Node *parent;}.
```



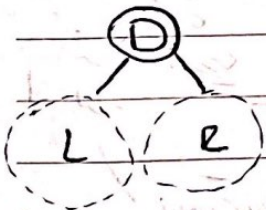




parent وجود ندارد.

* پیمایش درخت دودویی:
ملاقات کردن تمام گره‌های درخت و هر گره دقیقاً یک‌بار.

۱) پیمایش عمقی: پیشروی در عمق بر پیشروی در سطح اولویت دارد.
۲) پیمایش سطحی: پیشروی در سطح بر پیشروی در عمق اولویت دارد.

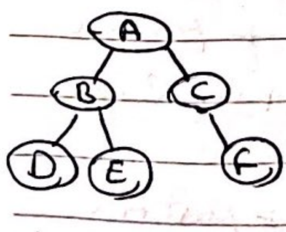


از پشته استفاده می‌کنیم

۱) پیمایش عمقی:
- inorder (میان ترتیب)
- postorder (پس ترتیب)
- preorder (پیش ترتیب)

inorder: LDR
preorder: DLR
postorder: LRD

* گره‌های چپ اولویت بر گره‌های راست دارند.



عکس داره :)

inorder: DBEACF → O(تعداد گره‌ها)
preorder: ABDECF → O(" ")
postorder: DEBFCA

Subject:

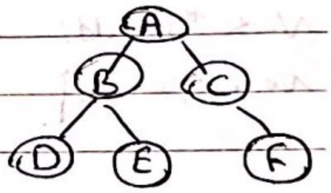
Year. Month. Date. ( )

۲۸ ←

* اگر در مسئله نحوه ذخیره سازی درخت مشخص نکنیم درخت دودویی با لیست پیوندی ذخیره شده است.

(۲) پیمایش سطحی: * از صف استفاده کنیم.

```
Void levelorder (Node *T){
if(T==NULL) return;
Queue Q;
Node *Curr=T;
while (Curr){
Cout << Curr->data;
if(Curr->Lchild) Q.insert(Curr->Lchild);
if(Curr->Rchild) Q.insert(Curr->Rchild);
Curr=Q.delete();}}
```



levelorder: ABCDEF

Curr: A, B, C, D, E, F, NULL

| B | C | D | E | F |

ABCDEF

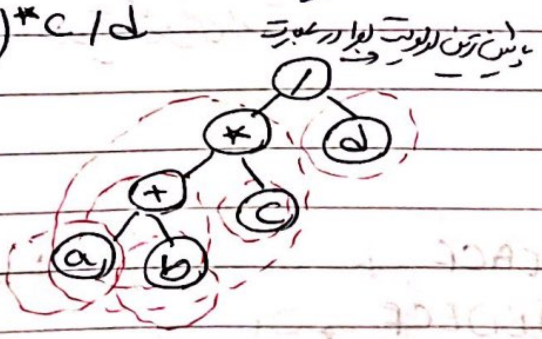
* درخت عبارت: یک درخت دودویی که یک عبارت ریاضی را نشان می‌دهد.

عملگر ← یکانی / دودویی ← گره‌های غیر برگ

عملوند ← گره‌های برگ

* هر چقدر که اولویت اجرای عملگر بیشتر باشد، گره متناظر با آن عملگر در سطوح پایین‌تری قرار دارد.

(a+b)*c/d



* inorder → infix

inorder: a+b*c/d

* Preorder → prefix

preorder: /*+abcd

* Postorder → postfix

postorder: ab+c*d/

```
class Node{
int value;
char operator;
Node *Lchild;
Node *Rchild;}
```

* value گره‌های داخلی را بدست بیاوریم و در نتیجه و حاصل را از ریشه بگیریم.

```
void postorderEval (node *T){
 if(T==Null) return;
 postorderEval (T->Lchild);
 postorderEval (T->Rchild);
 switch (T->operator){
 case '+';
 T->value = T->Lchild->value + T->Rchild->value;
 case '-' ; / case '*'; / case '/'; / ... }}
```

* اول باید پیمایش کنیم.
* تا value فرزند چپ و راست مشخص نباشد value ریشه مشخص نمی‌شود برای همین از postorder استفاده می‌کنیم. (اگر آخر هر ملاقات می‌شود)
* فقط با postorder می‌شود.

مسئله 3sat :

$x_1, x_2, \ldots, x_n \rightarrow 2^n$

$\wedge, \vee, \sim$

		$\wedge$	$\vee$
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

	$\sim$
F	T
T	F

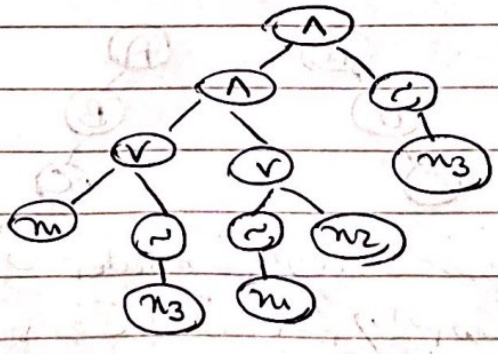
لیترال : متغیر منطقی یا نقیض متغیر منطقی

کلاز : ترکیب فصلی یک سری لیترال ($\vee$)

عبارت منطقی : ترکیب عطفی یک سری کلاز ($\wedge$)

3sat : در عبارت منطقی هر کلاز حاوی حداکثر ۳ لیترال است.

$(x_1 \vee \sim x_2) \wedge (\sim x_1 \vee x_2) \wedge \sim x_3$



Subject:

Year. Month. Date. ( )

۲۹ ← (۹,۳۰) :

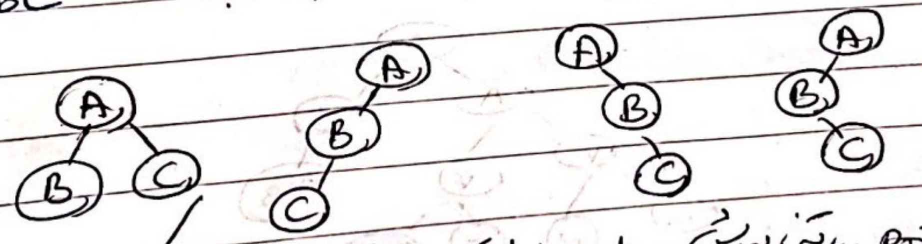
```
void postorderEval (Node *T){
  if(T==Null) return;                 y1,…yn
  postorderEval (T→LChild);           y1,…yn
  PostorderEval (T→RChild);           y1,…yn
  Switch(T→operator):{
  Case '¬': T→value = ¬(T→Rchild→value);
  Case '∧': T→value = (T→Rchild→value) ∧ (T→lchild→value);
  Case '∨': T→value = ( ~ ~ ~ ) ∨ ( ~ ~ ~ ) }}.
```

```
void SAT_Solution (Node *T){
  برای هر 2^n ترکیب ممکن برای متغیرها :
  { postorderEval (T);
  if(T→value==true) → جواب دارد SAT }
  SAT جواب ندارد. }
```

```
bool SATSolution (Node *T, bool S[n])
{for(i=1; i<=2^n; i++){
  یک ترکیب ممکن برای متغیرهای y1 تا yn
  y1…yn = y1 … yn
  PostorderEval (T,y1,…yn)
  if(T→value){S[n]=y1…yn, return true}.
} return false;}.
```

۷. آیا می‌توان درختی ساخت که ABC حاصل پیمایش preorder آن درخت باشد؟

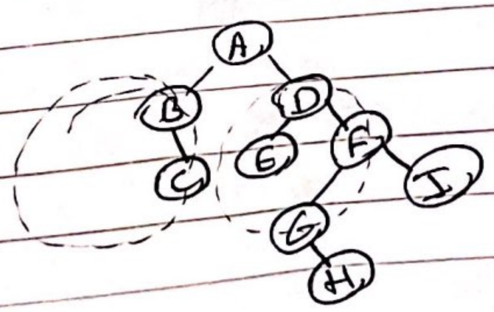
ABC



با داشتن تنها پیمایش postorder و یا تنها پیمایش inorder نمی‌توان درخت منحصربفردی را رسم کرد.
اما اگر هم پیمایش postorder / preorder و هم inorder درخت را داشته باشیم می‌توانیم درخت را مشخص کنیم.

preorder: ABCDEFGHI

Inorder: BC AEDGHFI



هرم ها (heap) — هرم حداقل (minheap) ← یک درخت دودویی کامل که کلید هر گره کوچکتر یا مساوی کلید فرزندانش است

هرم حداکثر (maxheap) ← یک درخت دودویی کامل که کلید هر گره بزرگتر یا مساوی کلید فرزندانش است

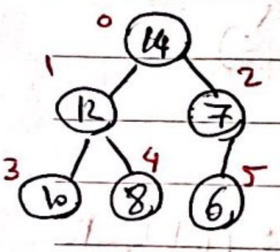
* از آرایه برای ذخیره کردن محتویات گره ها استفاده می کنیم.

* شماره گذاری را از صفر انجام می دهیم.

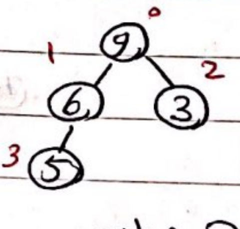
$i \xrightarrow{i \neq 0}$ پدر $= \frac{i-1}{2}$

$i \rightarrow$ فرزند راست $= 2*i+2$ (از صفر شروع شود)

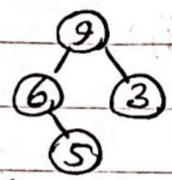
$i \rightarrow$ فرزند چپ $= 2*i+1$



maxheap



maxheap



X درخت دودویی کامل نیست.

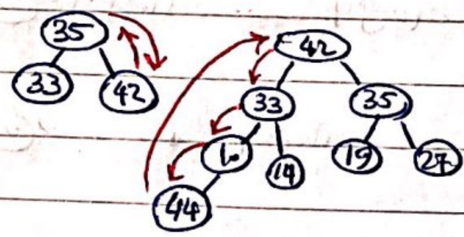
اضافه کردن گره:

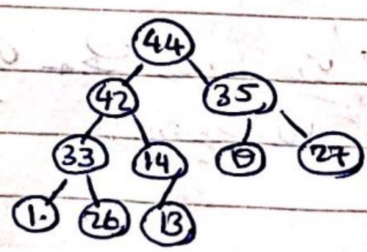
35, 33, 42, 10, 14, 19, 27, 44, 26, 13

```
int heap [maxSize]{
int n=0;
void insert (int n){
if(n==max_Size) {Cout<<"heap is full";
                 return;}
int i=n;
while (1){
if(i==0) break;
if(x<=heap[(i-1)/2]) break;
i= (i-1)/2; }      ▷ heap[i]=heap[(i-1)/2]
heap[i]=n;
n++;}}
```

$\rightarrow O(d)$ — ارتفاع $\rightarrow O(\log n)$

$d = \lceil \log_2^{n+1} \rceil$





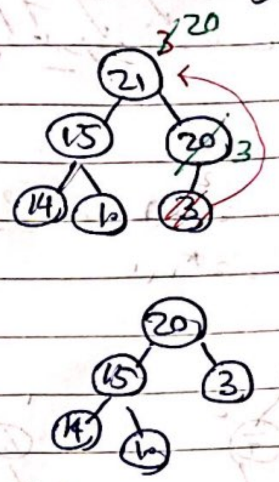
0	1	2	3	4	5	6	7	8	9
~~42~~ 44	~~33~~ 42	35	~~10~~ 33	14	19	27	~~44~~ 10	26	13

Subject:

Year. Month. Date. ( )

حذف کردن گره:

```
void delete(){                                  O(log n)
  if(n==0) {cout<<"heap is empty"; return;}
  int k=heap[n-1];
  int i=0,j=1;
  while(j<n){
    if(j<n && (heap[j]<heap[j+1])) j++;
    if(k>heap[j]) break;
    heap[i]=heap[j];
    i=j;
    j=2*j+1}
  heap[i]=k;}
```



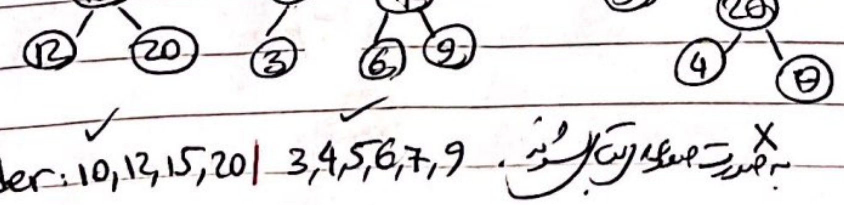
۳۰ ← (۱،۲) :

صف اولویت و روش‌های پیاده‌سازی:
- آرایه مرتب ← درج: O(n)، حذف: O(1)
- آرایه نامرتب ← درج: O(1)، حذف: O(n)
- آرایه غیر مرتب (max heap) ← درج: O(log n)، حذف: O(log n)

صف اولویت خاص: هر چقدر اولویت عنصر کمتر باشد زودتر از صف خارج شود ← minheap

BST (Binary Search tree): درخت جستجوی دودویی ← چون از درخت دودویی کامل استفاده کنیم لیست نمی‌تونیم برای ذخیره‌سازی استفاده کنیم، مشکل است.

1. هر گره حاوی کلید منحصر به فرد است.
2. مقدار کلید ریشه از کلید گره‌های موجود در زیر درخت چپ آن (البته در صورت وجود) بزرگتر است.
3. مقدار کلید ریشه از کلید گره‌های موجود در زیر درخت راست آن (البته در صورت وجود) کوچکتر است.
4. زیر درخت‌های چپ و راست ریشه هم به طور بازگشتی BST هستند.

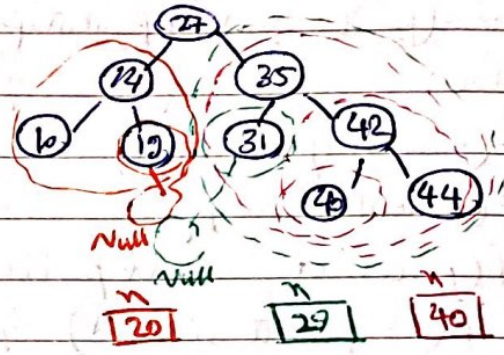


inorder: 10,12,15,20 | 3,4,5,6,7,9 . به صورت صعودی مرتب شود ✗

```
Node *Search(Node*T, int n){
  if(T==Null) return Null;
  if(n==T->data) return T;
  else if(n>T->data)
    Search(T->Rchild,n);
  else
    Search(T->Lchild,n);}
```

حستجو در BST : از بد حالت آن O(n) : بدترین حالت

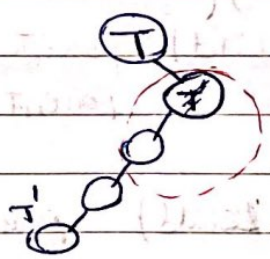
O(d) / O(log n) : بهترین حالت




```
Node *Next(Node*T){
  if(T==Null) return Null;
  T'=T->Rchild;
  while(T'&&T'->Lchild)
  T'=T'->Lchild;
  return T';}
```

پیدا کردن گره بعدی در پیمایش inorder : ته فرزند راست راس!

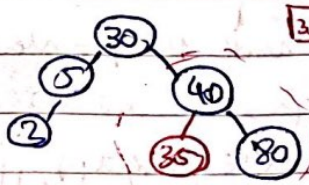
* هشتم ضرب کردن به این تابع نیاز داریم.




```
int insert(Node*T, int n){
node*p=T; Node*parent=Null;
while(p){
parent=p;
if(p->data==n) return 0;
else if(p->data>n) p=p->Lchild;
else p=p->Rchild;}
Node*n=new Node;
n->data=n;
n->Lchild=Null, n->Rchild=Null;
if(T==Null) T=n;
else if(n<Parent->data)  Parent->Lchild=n;
else                     Parent->Rchild=n;
```

درج یک گره جدید به BST :

نمونه



بدترین حالت : O(n)

بهترین حالت : O(d) { O(log n) / O(n)

عکس مثال ۱۱

Subject:

Year. Month. Date. ( )

```
int delete(node *T, int n){
node *p=T, node *parent=null;
while(p){
if(p->data==n) break;
elseif(p->data>n) {parent=p; p=p->lchild;}
else              {~ ~ ; p=p->Rchild;}
if(p==null) return 0;
if(parent!=null){
if(p->lchild==null && p->Rchild==null){
  if(p==parent->lchild) parent->lchild=null;
  else                  Parent->Rchild=null;
  return 1;}
elseif (p->lchd==null || p->Rchild==null){
  if(p->lchild!=null){
    if(p==parent->lchild) parent->lchild=p->lchild;
    else                  parent->Rchild=p->lchild;}
  else {
     if(p=parent->lchild) parent->lchild=p->Rchi;
     else                 parent->Rchild=p->Rchild;}
  return 1;}
else{
  node *ent p=ent(p);
  p->data=ent p->data;
  delete(ent p);}
else{
if(p->lchild==null && p->Rchild==null)
  root=null;
  return 1;}
```

حذف گره در BST :

کمترین حالت $O(\log n)$

$O(d)$

بیشترین حالت $O(n)$

① فرزند چپ و راست ندارد.

② یک فرزند دارد.

③ هر دو فرزند دارد.

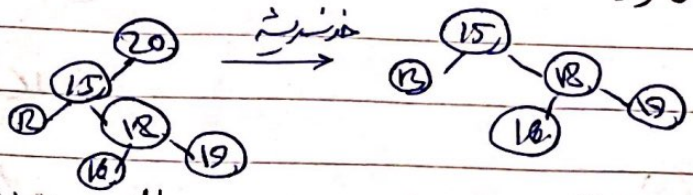
گره بعدی را در پیمایش inorder پیدا کنیم.

حذف ریشه

درخت هیچ عضو ندارد.

```
if (p->Lchild != NULL && p->Rchild == NULL){
    root = p->Lchild;
    return 1;}
else if(p->Lchild == NULL && p->Rchild != NULL){
    root = p->Rchild;
    return 1;}
else {
    node *nextp = Next(p);
    p->data = nextp->data;
    delete(nextp);}
```

فرزند چپ دارد.



فرزند راست دارد.

هر دو فرزند را دارد.

حذف آن.

۳۲ ← (۱۰,۷) :

گراف (Graph) ← $G(V,E)$ ← V : مجموعه‌ای غیرتهی از مجموعه رئوس هستند.

E مجموعه‌ای از زوج رئوس (a,b) که یال نامیده می‌شود.

گراف جهت دار: $e = \langle a,b \rangle \neq \langle b,a \rangle$

گراف بدون جهت یا بُد: $e = (a,b) = (a,b)$

درجه رأس v = $d(v)$ $\quad \forall v \in V \quad d(v)$

در گراف بدون جهت یا بُد:

$\sum_{v \in V} d(v) = 2|E|$ {دو برابر تعداد یال‌ها} $\quad d(v)$ = تعداد یال‌های گذرنده از رأس v

گراف جهت دار:

$d_{in}(v)$: تعداد یال‌هایی که رأس v رأس انتهایی آن‌ها است. (درجه ورودی) ← تعداد یال‌های وارد شده به رأس v

$d_{out}(v)$: تعداد یال‌هایی که رأس v، رأس ابتدایی آن‌ها است. (درجه خروجی) ← تعداد یال‌هایی که از رأس v خارج شده

$\sum_{v \in V} d_{in}(v) = \sum_{v \in V} d_{out}(v) = |E|$ {تعداد یال‌ها}

NEGAR

Subject:
Year. Month. Date. ( )

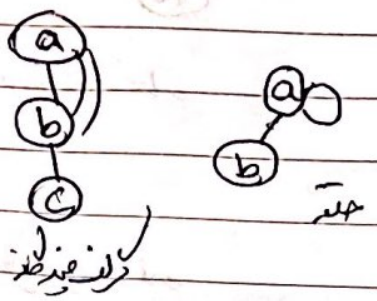
رئوس مجاور: دو رأس که توسط یک یال به هم وصل می‌شوند.

یال مجاور: دو یال که بر یک رأس مشترک هستند.

یال چندگانه (گراف چندگانه): اگر بین دو رأس بیش از یک یال وجود داشته باشد.

حلقه: یالی که دو سر آن یک رأس باشد.

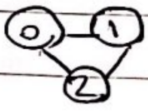
گراف ساده: گرافی که نه یال چندگانه و نه حلقه داشته باشد.

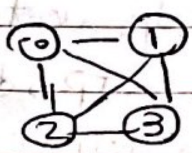


۱- گراف کامل: بین هر دو رأس یال داشته باشد ← n رأس ← K_n

(n-1) منتظم

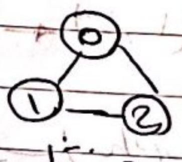
K_2 

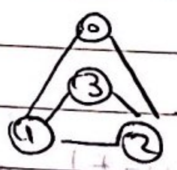
K_3 

K_4 

$|E| = \binom{n}{2} = \frac{n(n-1)}{2}$

۲- گراف k منتظم: درجه هر رأس برابر با k

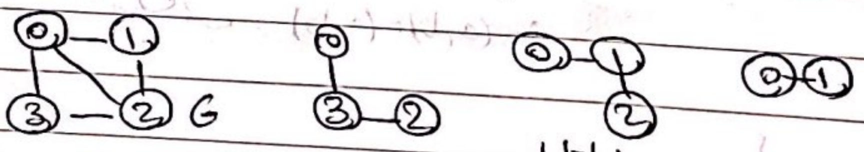




$|E| = \frac{nk}{2}$

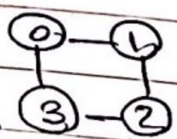
زیرگراف (Subgraph): گراف H را زیرگراف گراف G می‌گویند اگر

$V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$



زیرگراف فراگیر (پوشا) ← (Spanning subgraph): گراف H را زیرگراف پوشای G می‌گویند ←

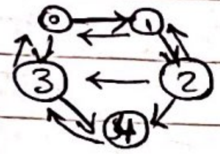
$V(H) = V(G)$, $E(H) \subseteq E(G)$



مسیر (Path): از رأس v به رأس u در گراف G یک دنباله از رئوس $v\, w_1, w_2, \ldots, w_k\, u$ که $(v,w_1), (w_1,w_2), \ldots, (w_k,u)$ ← یال‌های گراف باشند.

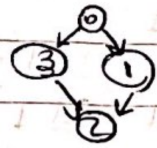
Year. Month. Date. ( )

گراف G همبند است درصورتی که بین هر دو راس آن مسیری باشد.



گراف جهت دار ← همبند قوی:

همبند ضعیف:



درخت: گراف همبند و بدون دور با n راس که (n-1) یال دارد.

درخت پوشا (Spanning tree): اگر G گراف باشد، T درخت پوشای G می‌گوییم هرگاه T درخت باشد. $V(T) = V(G)$, $E(T) \subseteq E(G)$

گراف وزن دار: گرافی که یال‌های آن وزن دار هستند.

مسئله minimum spanning tree (MST):

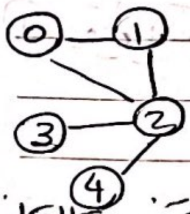
گراف وزن دار G داده شده ← هدف یافتن یک درخت پوشا از گراف G که کمترین وزن را داشته باشد.

* اگر G گراف کامل با n راس باشد تعداد درخت‌های پوشا n^{n-2} است.

روش‌های ذخیره کردن گراف در حافظه:

① ماتریس مجاورت:

n^2 حافظه ↑ کم‌ ... استفاده



	0	1	2	3	4
0		1	1	0	0
1	1	0	1	0	0
2	1	1	0	1	1
3	0	0	1	0	0
4	0	0	1	0	0

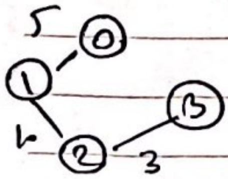
$a_{ij} \to 0 \quad (i,j) \notin E$

$a_{ij} \to 1 \quad (i,j) \in E$

* هر گراف بدون وزن n^2 مقدار صحیح باید ذخیره شود. * اگر گراف بدون جهت باشد مقادیر (۱و۰) هستند و می‌توان از ... استفاده کرد.

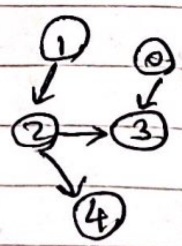
* در هر سطر مجموع عناصر سطر i در گراف بدون جهت درجه راس i می‌شود. (مجموع عناصر ستون i هم درجه راس i می‌شود)

* اگر گراف جهت دار باشد مجموع عناصر سطر i می‌شود درجه خروجی سطر i و مجموع درایه‌های ستون i می‌شود درجه ورودی راس i



	0	1	2	3
0	0	5	0	0
1	5	0	10	0
2	0	10	0	3
3	0	0	3	0

گراف وزن دار



	0	1	2	3	4
0	0	0	0	1	0
1	0	0	1	0	0
2	0	0	0	1	1
3	0	0	0	0	0
4	0	0	0	0	0

گراف جهت دار

NEGAR

* الگوریتم ... حداقل (n^2) است و ... است

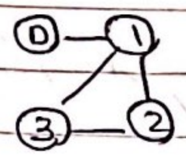
* حافظه مصرفی هم زیاد است ... n^2 ...

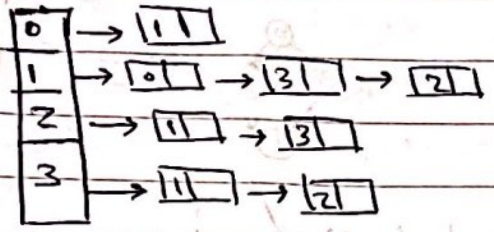
* حداکثر تعداد عناصر ناصفر = $n^2 - n$

Subject:
Year. Month. Date. ( )

② گراف اسپارس: ماتریس مجاورت آن اسپارس باشد.

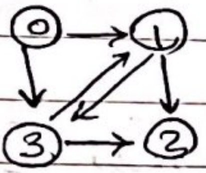
← از لیست مجاورت استفاده می‌کنیم.

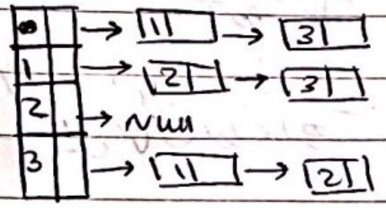




* تعداد گره‌های موجود در یک لیست مربوط به رأس i چیست؟ درجه i

* مجموع تعداد گره‌های لیست‌ها چه تعداد است؟ ۲ * تعداد یال





* مجموع تعداد گره‌های موجود در رأس i؟ درجه خروجی

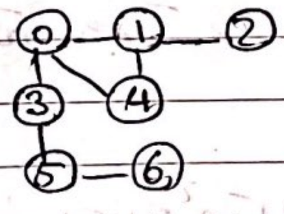
* مجموع تعداد گره‌های لیست‌ها؟ تعداد یال e

۳۳ ← (۱۰/۷):

عملیات گراف‌ها:

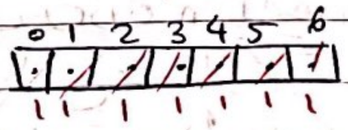
پیمایش گراف:
- عمقی، مشابه پیمایش preorder در درخت‌های دودویی (پشته)
- سطحی، مشابه پیمایش سطحی در درخت دودویی (صف)

① پیمایش عمقی:



```
int visited[n];
```

visited[i] ← 0 رأس i ملاقات نشده.
visited[i] ← 1 رأس i ملاقات شده.



0124356

```
int visited[n];
for(int i=0; i<n; i++)
    visited[i]=0;
```

```
void DFS (int v){
 visited [v]=1;
 Cout << v << " ";
 for( each vertex w adjacent to v)
   if(visited[w] !=1)
        DFS (w); }
```

گراف { ماتریس مجاورت: $O(n^2)$ ← برای هر راس ① پیدا کردن همسایه‌ها ② ملاقات کردن

اسپارس / لیست مجاورت: $O(e)$

برای هر راس لیستی که همسایه v است

گراف کامل ← $O(n^2)$

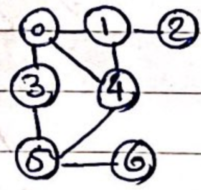
② پیمایش سطحی:

```
void BFS(int v){
 int visited[n]
 for(int i=0; i<n; i++)
  visited (i)=0;
  Queue Q;
  Q.insert(v);
 While(!Q.isempty()){
  w = Q.delete();  ──▶ if(visited[w]==0){
  visited[w]=1
 Cout << w << " ";
 for (به ازای هر همسایه w مانند z)
  if(visited[z]==0)
   Q.insert(z); }}
```

گراف { ماتریس مجاورت: $O(n^2)$ ← به ازای هر راس ① پیدا کردن همسایه‌ها ② ملاقات کردن

اسپارس / لیست مجاورت: $O(e)$

گراف کامل ← $O(n^2)$



0134256

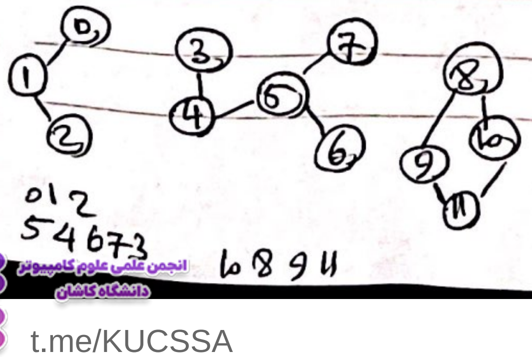
0	1	3	4	2	4	5	6

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

اگر گراف ناهمبند باشد و پیمایش BFS یا DFS انجام دهیم فقط همان رئوسی ملاقات می‌شوند که در مولفه‌ی همبندی آن راس شروع کننده باشند.

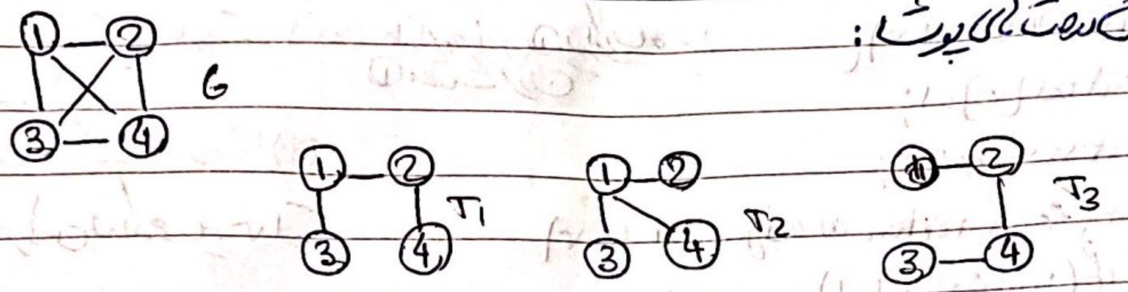
پیدا کردن مولفه‌های همبندی در یک گراف ناهمبند: اگر گراف همبند باشد که تمام رئوس با اعمال یکبار پیمایش، ملاقات می‌شوند ولی اگر گراف ناهمبند باشد و چند مولفه همبندی داشته باشد باید پیمایش را چندبار اعمال کرد روی راس‌هایی که تاکنون ملاقات نشده‌اند.

* مجموعه رئوس گراف در مولفه‌های همبندی افراز شوند.

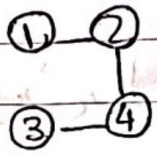


012
54673
10 8 9 11

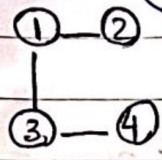
یافتن درخت های پوشا :



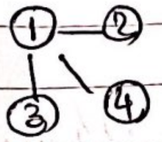
حاصل پیمایش عمقی (یا سطحی) از راس ۱ معمولا یافتن یک درخت پوشا می شود.



درخت پوشای عمقی



درخت پوشای عمقی



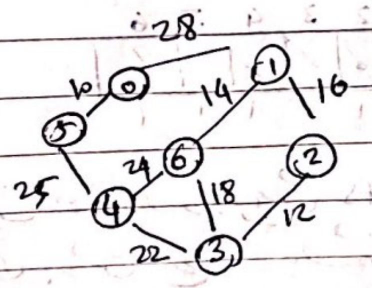
درخت پوشای سطحی

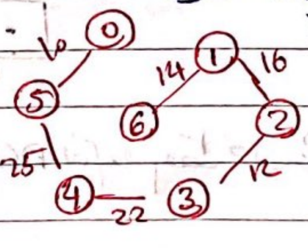
۳۴ـه :

درخت پوشای مینیمال: یک زیرگراف پوشا از این گراف است (معمولا رئوس همان مجموعه رئوس گراف است ولی مجموعه یال ها زیرمجموعه ی یال های گراف اصلی است و زیرگراف درخت است یعنی همبند است و دور ندارد و در میان تمام درخت های پوشای گراف کمترین وزن را دارد)

الگوریتم کراسکال: یال های موجود در گراف را به صورت صعودی مرتب می کند و هر بار به صورت تکراری عمل می کند در هر بار تکرار در بین یال هایی که هنوز باقی مانده است و انتخاب نشده یالی با کمترین وزن را انتخاب می کند.

n-1 یال باید اضافه شود.





$35+22+42=99$

* دور نباید ایجاد شود!

$S=E(G)$

$T=\emptyset$

الگوریتم کراسکال:

۱- $S = E(G)$, $T = \emptyset$

۲- تا وقتی که T شامل کمتر از n-1 یال باشد و S ناتهی باشد:

۲.۱: یالی از S انتخاب کنید که دارای کمترین وزن باشد.

۲.۲: اگر یال انتخاب شده با مجموعه یال های T ایجاد دور نکرد آن یال را به T اضافه کنید.

۲.۳: یال انتخابی را از S حذف می کنیم.

اگر T ، n-1 یال داشت یعنی درخت پیدا شده و اما اگر T کمتر از n-1 یال داشته باشد یعنی گراف مورد نظر درخت پوشا ندارد.

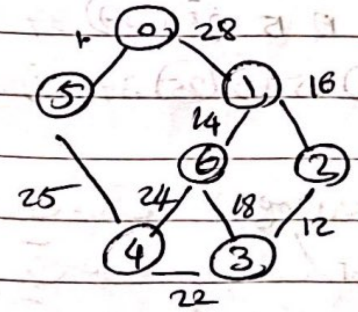
الگوریتم پریم:

۱- $T = \emptyset$, $S = \{0\}$ ← مجموعه رئوس پردازش شده

۲- تا وقتی T کمتر از n-1 یال داشته باشد:

۲.۱: یالی مانند (v_1, v_2) انتخاب می کنیم که $v_1 \in S$ و $v_2 \notin S$ و یال (v_1, v_2) در بین تمام یال هایی که از رأس ابتدایی آنها پردازش شده و رأس انتهایی پردازش نشده است کمترین وزن را داشته باشد.

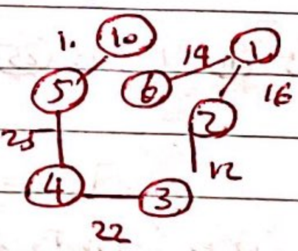
اگر چنین یالی وجود داشت به T اضافه می کنیم و v_2 را به S اضافه می کنیم. اگر T کمتر از n-1 یال داشته باشد گراف مورد نظر درخت پوشا ندارد.



$T = \emptyset$

$S = \{0\} \rightarrow \{0, 5, 4, 3, 2, 1\}$

مجموعه رئوس پردازش شده گراف



فصل مرتب سازی: ← با استفاده از آرایه

حافظه مصرفی اضافی ← درجا (inplace): حافظه کمکی زیادی استفاده نمی‌کند.
برون جا (outplace): " " " " می‌شود ← بیشتر بازگشتی

معیارها برای مقایسه روش‌های مرتب سازی:

۱- سرعت اجرای الگوریتم
۲- حافظه مصرفی
۳- رفتار طبیعی الگوریتم
۴- حفظ ترتیب عناصر با مقادیر مساوی ← الگوریتم مستقل پایدار (Stable)

۲۵ ← (۱۴/۱۰):

مرتب سازی سریع (Quick Sort): رفتار پایدار ندارد

pivot
25 5 19 1 61 11 59 15 48 37 → (25) 5 19 1 15 11 59 61 48 37

(11) 5 19 1 15 (25) (59) 61 48 37
→ (11) 5 1 19 15 (25) (59) 37 48 61

(1) 5 (11) (19) 15 (25) (48) 37 (59) 61 → (1) 5 (11) 15 (19) (25) 37 (48) (59) 61

بهترین حالت: $T(n) = 2T(n/2) + O(n) \longrightarrow T(n) \in \Theta(n \log n)$

بدترین حالت: زمانی است که pivot انتخاب شده لیست را نامتساوی تقسیم کند و pivot بزرگترین عنصر یا کوچکترین عنصر باشد.

$T(n) = T(n-1) + O(n) \longrightarrow T(n) \in \Theta(n^2)$

* رفتار طبیعی ندارد * متعادل نیست.

* out-place: غیر از فضای آرایه، پشته سیستم استفاده می‌کند و بازگشتی است.

Year. Month. Date.

```
void Quicksort(A[left...right]){
 int i,j,pivot;
 if(left<right){
  i=left+1; j=right-1;
  pivot=A[left];
  while(i<j){
  while(a[i]<pivot) i++;
  while(a[j]>pivot) j++;
  if(i<j) swap(a[i],a[j]);}
  Swap(pivot,a[j]);
  Quicksort(A[left...j-1]);
  Quicksort(A[j+1...right]);}
```

مرتب سازی ادغامی (megesort):

1)

26 5 77 1 61 11 59 15 48 19

→ 5 26 | 1 77 | 11 61 | 15 59 | 19 48

1 5 26 77 | 11 15 59 61 | 19 48

→ 1 5 11 15 26 59 61 77 | 19 48

1 5 11 15 19 26 48 59 61 77

2)

26 | 5 | 77 | 1 | 61 | 11 | 59 | 15 | 48 | 19

5,26 | 1,61 | 11,59 | 19,48

1,61,77 | 15,19,48

1,5,26,61,77 | 11,15,19,48,59 → 1,5,11,15,19,26,48,59,61,77

ubject:
ear. Month. Date. ( )

```
void Mergesort(int A[1..n]){
  if(n==1) return;
  int m=n/2;
  Mergesort(A[1..m]);
  mergesort(A[m+1..,n]);
  Merge(A[1..m], A[m+1..n], B[1..n]);
    for(int i=1; i<=n; i++){
      A[i]=B[i]; } }
```

$T(n) = 2T(n/2) + O(n)$

$T(n) \in O(n \log n)$

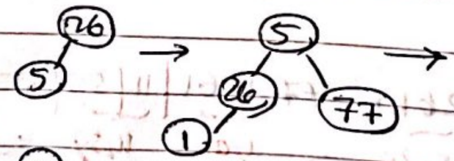
```
void Merge(A[S..m], A[m+1..,n], B[S..n])
{int i=S, j=m+1, k=S;
while(i<=m && j<=n){
  if(a[i] <= a[j]){ B[k]=a[i];
  k++; i++;}
  else { B[k]=a[j];
  k++; j++;}
if(i>m)
  while(j<=n){
    B[k]=a[j]; j++; k++;}
else
  while(i<=m){
  B[k]=a[i]; i++; k++;}}
```

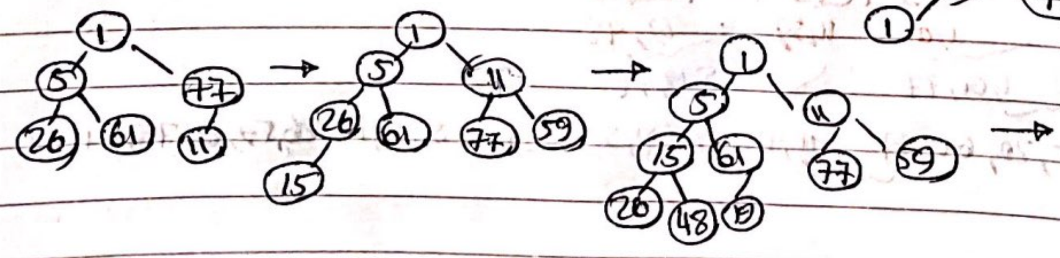
پیچیدگی زمانی : (مجموع طول دو آرایه) $O(n-S) \rightarrow O$

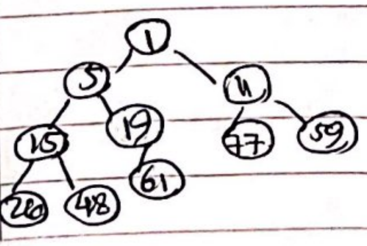
* رفتار طبیعی ندارد. * رفتار مشترک دارد. outplace

مرتب سازی هرمی (Heap-Sort) ← استفاده از minheap

26 5 77 1 61 11 59 15 48 19

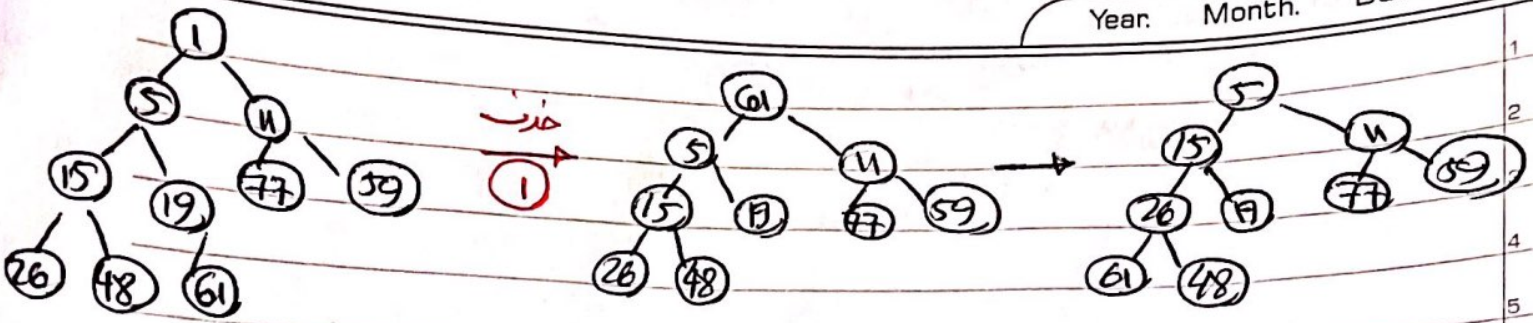




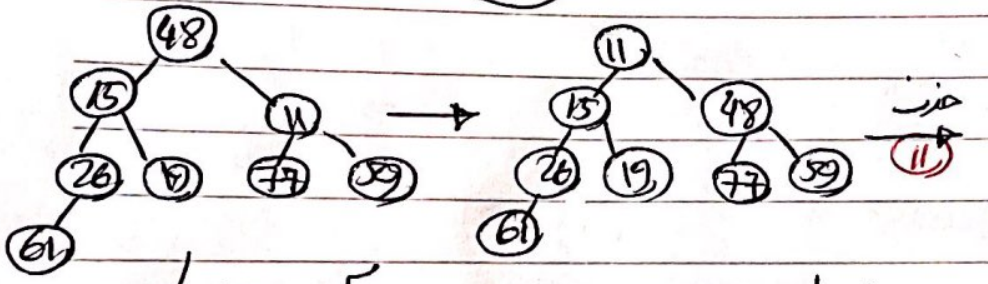


درج یک گره در minheap ← $O(\log n)$

← $O(n \log n)$



لیست مرتب شده صعودی است.
اگر بخواهیم به صورت نزولی باشد باید از maxheap استفاده کنیم.



پیچیدگی زمانی ← $O(n \log n)$
* روش طبیعی ندارد
* outplace ← از آرایه استفاده کردیم.

تمام :)