فصل اول

کارایی، تحلیل و مرتبه الگوریتم ها

کارایی، تحلیل و مرتبه الگوریتم ها

- تکنیک ، روش مورد استفاده در حل مسائل است.
- مسئله ، پرسشی است که به دنبال پاسخ آن هستیم.
- بکار بردن تکنیک منجر به روشی گام به گام (الگوریتم) در حل یک مسئله می شود.
- منظورازسریع بودن یک الگوریتم، یعنی تحلیل آن از لحاظ زمان و حافظه.

- نوشتن الگوریتم به زبان فارسی دو ایراد دارد:

۱- نوشتن الگوریتم های پیچیده به این شیوه دشوار است.

۲- مشخص نیست از توصیف فارسی الگوریتم چگونه می توان یک برنامه کامپیوتری ایجاد کرد.

کارایی، تحلیل و مرتبه الگوریتم ها

تحلیل الگوریتم ها

- برای تعیین میزان کارایی یک الگوریتم را باید تحلیل کرد.

تحلیل پیچیدگی زمانی

- تحلیل پیچیدگی زمانی یک الگوریتم ، تعیین تعداد دفعاتی است که عمل اصلی به ازای هر مقدار از ورودی انجام می شود.

کارایی، تحلیل و مرتبه الگوریتم ها

- $T(n)$ را پیچیدگی زمانی الگوریتم در حالت معمول می گویند.
- $W(n)$ را تحلیل پیچیدگی زمانی در بدترین حالت می نامند.
- $A(n)$ را پیچیدگی زمانی در حالت میانگین می گویند.

تحلیل پیچیدگی زمانی برای حالت معمول برای الگوریتم(جمع کردن عناصرآرایه)

عمل اصلی: افزودن یک عنصر از آرایه به sum.

اندازه ورودی: n، تعداد عناصر آرایه.

$$T(n) = n$$

تحلیل پیچیدگی زمانی برای حالت معمول برای الگوریتم(مرتب سازی تعویضی)

عمل اصلی: مقایسه S [j] با S[i] .

اندازه ورودی: تعداد عناصری که باید مرتب شوند.

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

تحلیل پیچیدگی زمانی دربدترین حالت برای الگوریتم(جست و جوی ترتیبی)

عمل اصلی: مقایسه یک عنصر آرایه با x.

اندازه ورودی: n , تعداد عناصر موجود در آرایه.

$$W(n) = n$$

تحلیل پیچیدگی زمانی در بهترین حالت برای الگوریتم(جست وجوی ترتیبی)

عمل اصلی: مقایسه یک عنصر آرایه با x.

اندازه ورودی: n , تعداد عناصر آرایه.

$$B(n) = 1$$

کارایی، تحلیل و مرتبه الگوریتم ها

مثال ۱: برای مثال جمع عناصر یک آرایه ، تعداد کل مراحل برنامه را بنویسید.

```
float   sum  (float  list[  ], int  n)
{
  float  s=0 ;       int i;
    for (i = 0;  i<n;  i++)
              s = s + list[i];
       return    s;
  }
```

0
0
1
0
n+1
n
1
0
2n+3

نکته : هنگام محاسبه تعداد دفعاتی که یک دستور درون حلقه‌ها اجراء می‌گردد می‌توان از فرمولهای زیر استفاده کرد :

$$\sum_{i=1}^{n} 1 = n \qquad \sum_{i=1}^{n} kf(i) = k\sum_{i=1}^{n} f(i)$$

K عدد ثابتی است.

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

a_1 جمله اول، a_n جمله آخر و n تعداد جملات است : $\text{جمع تصاعد عددی} = \frac{n(a_1 + a_n)}{2}$

راه دیگر Trace کردن برنامه است.

مثال ۲: دستور اصلی در تکه برنامه زیر چند بار اجرا میشود؟ تعداد کل گامهای برنامه چند است؟

```
for  i: = 1 to m do
    for j: = 1 to n do
            x: = x+1;
```

راه حل اول:

تعداد اجرای دستور به دلیل وجود حلقه های مستقل برابر nm است.

راه حل دوم:

$$\text{تعداد اجرای دستور اصلی} = \sum_{i=1}^{m}\sum_{j=1}^{n} 1 = \sum_{i=1}^{m} n = n\left(\sum_{i=1}^{m} 1\right) = nm$$

$$\text{تعداد کل مراحل برنامه} = (m + 1) + m(n + 1) + mn$$

مثال ۳: دستور اصلی در تکه برنامه زیر چند بار اجرا میشود؟

```
for  j:= 1 to n do
    for  i: = 1 to j do
                x: = x+1;
```

j	تغییرات i	تعداد اجرا شدن دستور اصلی
1	1	1 بار
2	1,2	2 بار
3	1,2,3	3 بار
	.....	
n	1,2,3,...,n	n بار

$$\text{x: = x +1; تعداد اجرا شدن دستور } = 1+2+3+\ldots+n = \frac{n(n+1)}{2}$$

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

کارایی، تحلیل و مرتبه الگوریتم ها

مثال ۳: دستور اصلی در تکه برنامه زیر چند بار اجرا میشود؟

```
i:=n;
while (i>1) do begin
        x:=x+1;
         i: = i div 2;
end;
```

اگر n=16 باشد:

i	شرط i >1	تعداد اجرا شدن دستور اصلی
16	درست	۱ بار
8	درست	۱ بار
4	درست	۱ بار
2	درست	۱ بار
1	غلط	-
		جمعاً ٤ بار

کارایی، تحلیل و مرتبه الگوریتم ها

اگر **n=14** باشد:

تعداد اجرا شدن دستور اصلی	شرط i>1	i
۱ بار	درست	14
۱ بار	درست	7
۱ بار	درست	3
-	غلط	1
جمعاً ۳ بار		

دستور اصلی به تعداد $\lfloor \log_2^n \rfloor$ بار اجرا می‌شود.

کارایی، تحلیل و مرتبه الگوریتم ها

برخی مسائل برای همه موارد یک تابع پیچیدگی دارند مثل الگوریتم جمع عناصر یک آرایه :

```
A : Array [1 .. n]  of Integer;
S := 0;
For I := 1  To  n  do
    S := S + A[i] ;
```

در برنامه فوق عمل اصلی S := S + A[i] به تعداد n بار اجرا شده و همواره T(n) = n می‌باشد. ولی در الگوریتمی مثل جستجوی خطی (ترتیبی) تابع پیچیدگی برای حالات مختلف ممکن است متفاوت باشد.

```
A : Array [1 .. n] of Integer;
For  i := 1   to  n  do
     if (x = A[i])  {
              write ('Yes');
                exit ( ) ; → خروج از برنامه
        }
write ('No');
```

در برنامه فوق عمل اصلی شرط if (x = A[i]) می‌باشد.

در بدترین حالت عدد x ، در خانه آخر قرار دارد و یا اصلاً در آرایه نیست که در این حالت باید n بار عمل اصلی آزمایش کردن، انجام گیرد. برای بدترین حالت به جای نماد T(n) از نماد W(n) استفاده می‌کنیم که W مخفف Worst یعنی بدترین است. پس در برنامه فوق W(n) = n می‌باشد.

در بهترین حالت عدد x در اولین خانه قرار دارد و تنها به یک عمل if مورد نیاز است. بهترین حالت را با B نمایش می‌دهیم که مخفف Best است لذا در برنامه فوق داریم : B(n) = 1

برای تحلیل برنامه فوق در حالت متوسط که آن را با A (مخفف Average) نشان می‌دهیم باید به صورت زیر عمل کنیم. ابتدا فرض می‌کنیم x در آرایه A وجود داشته باشد. بدیهی است احتمال آنکه x در خانه i ام باشد برابر $\frac{1}{n}$ است و تعداد دفعات آزمایش کردن در این حالت برابر i است. لذا :

$$A(n) = \sum_{i=1}^{n} \frac{1}{n} \times i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}$$

پس به طور متوسط نیمی از عناصر آرایه باید جستجو شود.

حال موردی را در نظر می‌گیریم که x ممکن است اصلاً در آرایه نباشد. فرض کنید احتمال وجود x در آرایه برابر p است. پس احتمال آنکه x در یکی از خانه‌های آرایه (مثل خانه i ام) باشد برابر $\frac{p}{n}$ است. احتمال آنکه x در آرایه نباشد برابر 1-p است. اگر x در خانه i ام باشد دستور اصلی i بار اجرا می‌شود و اگر x در آرایه نباشد دستور اصلی n بار اجرا می‌شود، لذا :

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

توجه کنید که اگر p = 1 باشد همان حالت قبلی $A(n)=\frac{n+1}{2}$ بدست می‌آید و اگر $p=\frac{1}{2}$ باشد $A(n)=\frac{3n}{4}+\frac{1}{4}$ می‌شود و این بدان معناست که حدود $\frac{3}{4}$ آرایه به طور میانگین باید جستجو شود.

مرتبه الگوریتم

- الگوریتم ها یی با پیچیدگی زمانی ازقبیل n و100n را الگوریتم های زمانی خطی می گویند.

- مجموعه کامل توابع پیچیدگی را که با توابع درجه دوم محض قابل دسته بندی باشند، $\theta(n^2)$ می گویند.

- مجموعه ای ازتوابع پیچیدگی که با توابع درجه سوم محض قابل دسته بندی باشند، $\theta(n^3)$ نامیده می شوند.

- برخی از گروه های پیچیدگی متداول در زیر داده شده است:

$$\theta(\lg n) < \theta(n) < \theta(n \lg n) < \theta(n^2) < \theta(n^3) < \theta(2^n)$$

- برای یک تابع پیچیدگی مفروض $f(n)$، $O(f(n))$ "O" بزرگ" مجموعه ای از توابع پیچیدگی $g(n)$ است که برای آن ها یک ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی $n >= N$ داریم:

$$g(n) >= c \times f(n)$$

- برای یک تابع پیچیدگی مفروض $f(n)$ ، $\Omega(f(n))$ مجموعه ای از توابع پیچیدگی $g(n)$ است که برای آن ها یک عدد ثابت حقیقی مثبت c و یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی $n >= N$ داریم:

$$g(n) =< c \times f(n)$$

برای یک تابع پیچیدگی مفروض $f(n)$، داریم:

$$\theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

یعنی $\theta(f(n))$ مجموعه ای از توابع پیچیدگی $g(n)$ است که برای آن ها ثابت های حقیقی مثبت c و d و عدد صحیح غیر منفی N وجود دارد به قسمی که :

$$c \times f(n) <= d \times f(n)$$

- برای یک تابع پیچیدگی $f(n)$ مفروض، $o(f(n))$ "o کوچک" عبارت ازمجموعه کلیه توابع پیچیدگی $g(n)$ است که این شرط را برآورده می سازند : به ازای هرثابت حقیقی مثبت c ،یک عدد صحیح غیر منفی N وجود دارد به قسمی که به ازای همه ی $n >= N$ داریم:

$$g(n) =< c \times f(n)$$

ویژگی های مرتبه

۱- $g(n) \in O(f(n))$ اگروفقط اگر $f(n) \in \Omega(g(n))$.

۲- $g(n) \in \theta(f(n))$ اگروفقط اگر $f(n) \in \theta(g(n))$.

۳- اگر $b > 1$ و $a > 1$، در آن صورت:

$$\log_a n \in \theta(\log_b n)$$

۴- اگر $b > a > 0$،در آن صورت:

$$a^n \in o(b^n)$$

۵- به ازای همه ی مقادیر $a > 0$ داریم :

$$a^n \in o(n!)$$

۶- اگر $c >= 0$، $d > 0$ ، $g(n) \in o(f(n))$ و $h(n) \in \theta(f(n))$ باشد، در آن صورت:

$$c \times g(n) + d \times h(n) \in \theta(f(n))$$

۷- ترتیب دسته های پیچیدگی زیر را در نظربگیرید:

$\theta(\lg n)\ \theta(n)\ \theta(n \lg n)\ \theta(n^2)\ \theta(n^j)\ \theta(n^k)\ \theta(a^n)\ \theta(b^n)\ \theta(n!)$

که در آن $2 < j < k$ و $1 < a < b$ است. اگر تابع پیچیدگی $g(n)$ در دسته ای واقع در طرف چپ دسته ی حاوی $f(n)$ باشد، در آن صورت:

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