


ساختمان داده‌ها

تحلیل الگوریتم

الگوریتم‌ها از نظر حافظه و زمان مصرفی مورد ارزیابی قرار می‌گیرند. به عبارت بهتر الگوریتمی مطلوب تر است که حافظه و زمان کمتری مصرف کند. البته در اینجا تمرکز بیشتر بر روی زمان لازم برای اجرای یک الگوریتم است.

عوامل زیادی می‌توانند بر روی زمان اجرای یک الگوریتم تأثیر داشته باشند اما به دلیل اینکه نیازمند تحلیلی مستقل از سخت افزار و زبان برنامه نویسی هستیم، تغییرات زمان اجرای الگوریتم را در مقابل تغییر اندازه ورودی تحلیل می نماییم. ضمناً با توجه به اینکه ترکیب ورودی نیز بر زمان اجرای الگوریتم موثر است برای آن سه حالت در نظر می‌گیریم:

- بهترین حالت (best case): ترکیبی از ورودی که الگوریتم سریع‌ترین پاسخ را می‌دهد.
- بدترین حالت (worst case): ترکیبی از ورودی که الگوریتم کندترین پاسخ را می‌دهد.
- حالت متوسط (average case): ترکیب متداول و تصادفی از ورودی.

پس از تعیین اندازه ورودی و ترکیب آن، یک دستور یا گروهی از دستورها که «عمل اصلی» را انجام می‌دهند، تعیین می‌شوند. در حقیقت تحلیل پیچیدگی زمانی یک الگوریتم تعیین دفعاتی است که عمل اصلی به ازای هر مقدار اندازه ورودی انجام می‌شود.

مثال: دستور اصلی X:=X+1 در قطعه برنامه زیر چند بار اجرا می‌شود؟

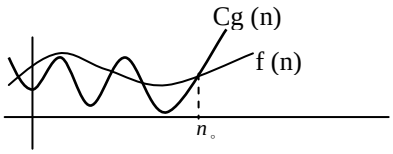
```
for i := ۱ to m do
    for j := ۱ to n do
        x := x + ۱
```

حل

$$\sum_{i=۱}^{m}\sum_{j=۱}^{n} ۱ = \sum_{i=۱}^{m} n = nm$$

تعیین دقیق تعداد دفعات اجرا شدن یک دستور در برخی موارد می‌تواند بسیار پیچیده و در عین حال غیر لازم باشد در حقیقت رفتار حدی الگوریتم با توجه به اندازه ورودی مورد اهمیت است. بنابراین نمادهای زیر تعریف می‌شوند.

1) $f(n) \in O(g(n)) \Leftrightarrow \exists c, n_o > ۰ : \forall n \geq n_o \quad f(n) \leq cg(n)$



عبارت بالا را می‌توان به صورت زیر نشان داد:

یعنی از n_o به بعد $cg(n)$ همواره بزرگ تر از $f(n)$ است. (حالت تساوی با در نظر گرفتن c مناسب به «بزرگ تر» تبدیل می‌شود.)

مثال: $5n+3 \in O(n)$

$c=6 \quad n_o=3 \quad \Rightarrow 5n+3 \leq 6n$

2) $f(n) \in \Omega(g(n)) \Leftrightarrow \exists c, n_o > ۰ : \forall n \geq n_o \quad f(n) \geq cg(n)$

با مقایسه این دو تعریف واضح است که O حد بالا و Ω حد پایین را برای $f(n)$ مشخص می‌کند. بنابراین در هنگام تعیین O باید کوچک‌ترین حد و در هنگام تعیین Ω بزرگ ترین حد را پیدا کنیم. (کوچک ترین حد بالا و بزرگ ترین حد پایین)

مثال: $3n+2 \in \Omega(n)$

$c=3 \quad , \quad n_0=1 \quad \Rightarrow \quad 3n+2 \geq 3n$

3) $f(n) \in \theta(g(n)) \Leftrightarrow \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0 \quad c_1 g(n) \leq f(n) \leq c_2 g(n)$

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

به عبارت بهتر:

مثال: $4n+1 \in \theta(n)$

$c_2 = 5, c_1 = 4, n_. = 1 \Rightarrow 4n \leq 4n+1 \leq 5n$

قضیه: اگر $f(n) = a_m n^m + ... + a_1 n + a_.$ **آنگاه** $f(n) \in \theta(n^m)$

مثال:

1) x:=x+1 $\rightarrow O(1)$

2) for i:=1 to n do
x:=x+1 $\rightarrow O(n)$

3) for i:= 2 to 100 do
x:= x+1 $\rightarrow O(1)$

حلقه به تعداد محدود و معین (مستقل از n) اجرا می‌شود.

نکته:

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

$g(n) \in \theta(f(n)) \Leftrightarrow f(n) \in \theta(g(n))$

$f(n)+g(n) \in O(\max\{f(n), g(n)\})$

مثال:

۱) $6n^2 2^n + 3n \lg n \in \theta(n^2 2^n)$ ۲) $\lg(n!) \in \theta(n \lg n)$ ۳) $n^{1/01} + n \lg n \in O(n^{1/01})$

در بعضی مراجع دو نماد زیر نیز تعریف شده‌اند:

$o(f(n)) = \{g : N \rightarrow R^{>0} \mid \exists n_0 > 0 \forall c > 0 \forall n > n_0 \quad g(n) \leq c f(n)\}$

تفاوت اصلی o با O در سور مربوط به C است.

نکته:

$\omega(f(n)) = \{g : N \rightarrow R^{>0} \mid \exists n_0 > 0 \forall c > 0 \forall n > n_0 \quad c f(n) \leq g(n)\}$

$o(f(n)) \cap \omega(f(n)) = \phi$

الگوریتم های بازگشتی و تحلیل آنها:

برنامه بازگشتی برنامه‌ای است که برای اجرا نیاز به اجرای خودش در ابعاد کوچک‌تر دارد. برنامه‌های بازگشتی با استقرای ریاضی مرتبط هستند و بر این اساس دارای دو ویژگی هستند:

۱) قسمتی از برنامه خود را صدا می‌زند. ۲) یک شرط جهت اتمام فراخوانی ها وجود دارد (شرط پایه)

متداول ترین مثال برای این نوع برنامه‌ها، محاسبه n! است.

برای تحلیل یک برنامه بازگشتی باید رابطه بازگشتی مربوط به آن را حل کنیم.

مثال: محاسبه n!:

```
int fact(int n)
{
  if (n ==۱) return ۱;
  return (n * fact(n – ۱));
}
```

$$T(n)=\begin{cases} ۱ & n=۱ \\ T(n-۱)+c \end{cases}$$

T(n)=T(n-1)+c=T(n-2)+2c=...=T(1)+(n-1)c⇒ T(n)∈ O(n) (O(1)) c یک مقدار ثابت است

نکته: به طور کلی اگر رابطه بازگشتی به صورت T(n)=T(n-m)+O(g(n)) داشته باشیم (m عدد ثابت است) آنگاه:

T(n)∈ O(n*g(n))

مثال: برج‌های هانوی:

```
void Hanoi (int n, A, B, C)
{
  if (n ==۱) Move a disk from A to C;
  else{
      Hanoi (n – ۱, A, C, B);
      Move a disk from A to C;
      Hanoi (n – ۱, B, A, C):
   }
}
```

$$T(n)=\begin{cases} ۱ & n=۱ \\ T(n-۱)+T(n-۱)+۱ & n>۱ \end{cases}$$

$$T(n)=۲T(n-۱)+۱=۲(۲T(n-۲)+۱)+۱=۲^{۲}T(n-۲)+۲+۱=...$$

$$=۲^{n-۱}T(۱)+۲^{n-۲}+۲^{n-۳}+...+۲+۱$$ جمع تصاعد هندسی

$$=\frac{۱\times(۲^{n}-۱)}{۲-۱}=۲^{n}-۱$$

نکته: چنانچه یک رابطه بازگشتی به صورت T(n)=kT(n-m)+O(g(n)) باشد (k و m مقادیر ثابت هستند) آنگاه داریم:

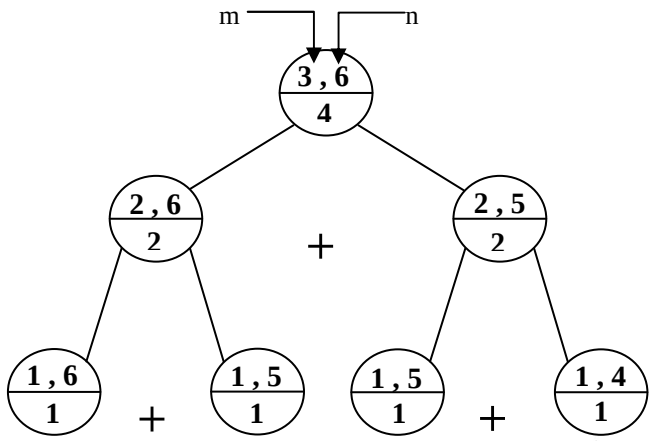
$$T(n)\in O\left(k^{\frac{n}{m}}g(n)\right)$$

برای تعیین پاسخ یک تابع بازگشتی بهترین روش استفاده از درخت بازگشتی است.

مثال: در برنامه زیر مقدار F(3,6) برابر است با: (مهندسی- سراسری ۷۵) (آزاد- ۸۱)

۱) ۲۰ ۲) ۱۰ ۳) ۱۸ ٤) ٤

حل




```
int F(int m, int n)
{
 if (m ==١||n ==٠||m ==n)
     return ١
 else
     return  F(m −١, n) + F(m −١, n −١)
}
```

⇐ گزینه 4

توجه: به عملگرها و متغیرها توجه کنید

مثال: خروجی برنامه زیر برای دو عدد صحیح مثبت x و y چیست؟ (علوم کامپیوتر- سراسری- ۸٤)

۱) x ۲) y ۳) y-x ٤) x+y

```
int  test (int x, int y)
{
 if  (x ==٠) return y;
 else
     return  test (−− x, y ++);
}
```

حل گزینه ۲ (توجه کنید که در اینجا test(- - x, y + +) معادل test (- - x, y) است.)

مثال: خروجی تابع زیر با F(a,5) کدام است؟ (علوم کامپیوتر- سراسری ۷۹)

۱) 0, 1, 2, 3, 4 ۲) 4, 3, 2, 1, 0 ۳) 1, 2, 3, 4, 5 ٤) 0, 0, 0, 0, 0

```
#define MAX ١٠
int a[max]
void F(int * a, int n){
  static int i = ٠; int k = ٠;
  if  (i < n){
          a[i] = k + +;
           print f ("%d", a[i + +]);
           F(a, n);
  }
}
```

حل گزینه ٤. توجه کنید که i از نوع Static است ولی k از نوع static نیست. لذا در هر بار صدا زدن تابع دوباره k در ابتدا صفر می‌شود. اگر متغیر k از نوع static بود جواب گزینه ۱ می‌شد.

آرایه و رشته:

آرایه یک بعدی: اگر آرایه $A:Array[L_1...U_1]of\ Type$ از آدرس α در حافظه شروع شود داریم:

آدرس خانه $A[i] = (i-L_1)*size\ of\ Type+\alpha$

تعداد عناصر آرایه $A = U_1 - L_1 + 1$

جستجو در آرایه یک بعدی: در آرایه های نا مرتب جستجو به صورت خطی (یا ترتیبی) و در آرایه های مرتب به صورت دودویی صورت می‌گیرد.

نکته ۱: مرتبه اجرایی جستجوی خطی $O(n)$ است. چرا که حداکثر لازم است تمام n خانه آرایه بررسی شود.

نکته ۲: مرتبه اجرایی جستجوی دودویی $O(lgn)$ است. چون هر بار نصف آرایه مورد بررسی قرار می‌گیرد.

نکته ۳: در روش جستجوی دودویی، در بدترین حالت با $\lfloor \lg n \rfloor + 1$ مقایسه می‌توان کلید را یافت.

آرایه دو بعدی: برای آرایه $A:Array[L_1..U_1,L_2..U_2]\ of\ Type$ داریم:

تعداد عناصر $A = (U_1-L_1+1)*(U_2-L_2+1)$

هم چنین برای پیدا کرن آدرس $A[i,j]$ با توجه به روش ذخیره (سطری یا ستونی) و با فرض اینکه آرایه از آدرس α شروع شود، داریم:

محل $A[i,j]$ در روش سطری $= [(i-L_1)(U_2-L_2+1)+(j-L_2)]*size\ of\ Type+\alpha$

محل $A[i,j]$ در روش ستونی $= [(j-L_2)(U_1-L_1+1)+(i-L_1)]*size\ of\ Type+\alpha$

به همین ترتیب در آرایه سه بعدی $A[L_1...U_1,L_2...U_2,L_3...U_3]\ of\ Type$ که از آدرس α شروع می‌شود داریم:

محل $A[i,j,k]$ در روش سطری =

$[(i-L_1)(U_2-L_2+1)(U_3-L_3+1)+(j-L_2)(U_3-L_3+1)+(k-L_3)]\ *size\ of\ Type+\alpha$

محل $A[i,j,k]$ در روش ستونی =

$[(k-L_3)(U_2-L_2+1)(U_1-L_1+1)+(j-L_2)(U_1-L_1+1)+(i-L_1)]\ *size\ of\ Type+\alpha$

ماتریس اسپارس: برای نمایش ماتریس اسپارس می‌توان از یک ماتریس $(N+1)\times 3$ استفاده نمود که N تعداد عناصر غیر صفر است. در این حالت هر عنصر از ماتریس اسپارس به وسیله مکانش و مقدارش و به صورت (i,j,value) مشخص می‌شود.

برای ذخیره ماتریس‌های بالا یا پایین مثلثی و ماتریس های قطری (سه قطری و ...) می‌توان از آرایه یک بعدی استفاده نمود.

مثال: اگر ماتریس پایین مثلثی به صورت زیر ذخیره شود داریم:

1	2	3	4	5	6	
a_{11}	a_{12}	a_{22}	a_{31}	a_{32}	a_{33}	...

عنصر $A[i,j]$ ماتریس پایین مثلثی معادل است با عنصر $B\left[\frac{i(i-1)}{2}+j\right]$ در آرایه یک بعدی فوق

لیست پیوندی:

لیست پیوندی بر خلاف آرایه پویاست. به همین دلیل درج و حذف در آن ساده تر و سریع تر از آرایه است. اما جستجو و مرتب‌سازی می‌توانند در آرایه بسیار سریع‌تر انجام شوند.

هر عنصر در لیست یک طرفه خطی از دو بخش تشکیل شده است که یکی برای ذخیره مقدار و دیگری برای اشاره به عنصر بعدی لیست است. در لیست یک طرفه خطی نمی‌توان جستجوی دودویی انجام داد جستجوی خطی در لیست یک طرفه از مرتبه O(n) است.

برای حذف گره با آدرس مشخص از لیست یک طرفه ابتدا باید لیست را تا یافتن گره قبل از آن پیمایش نمود. بنابراین عملیات حذف یک عنصر دلخواه از لیست یک طرفه خطی از مرتبه O(n) است. در حالی که درج با فرض دانستن مکان آن از O(1) است.

برای دسترسی به عناصر لیست یک طرفه خطی لازم است آدرس عنصر ابتدایی را داشته باشیم. بنابراین گاهی برای هماهنگی این عنصر با دیگر عناصر آن را مقدار دهی نمی‌کنند و در حقیقت اولین عنصر حاوی مقدار بعد از عنصر ابتدایی قرار می گیرد. در این حالت به عنصر ابتدایی «سر لیست» گفته می شود.

آخرین اشاره گر لیست یک طرفه خطی null است. با تعریف لیست حلقوی یک طرفه این اشاره گر به عنصر اول اشاره می کند. در چنین لیستی با داشتن آدرس هر گره می توان به کلیه گره ها دسترسی داشت و معمولاً به خاطر ساده نمودن درج در ابتدای لیست آدرس گره انتهایی نگه داری می شود. لیست حلقوی نیز می تواند سر لیست داشته باشد.

در لیست پیوندی دو طرفه در هر گره دو اشاره گر وجود دارد که یکی به گره بعدی و دیگری به گره قبلی در لیست اشاره می کند. بنابراین با داشتن آدرس یک گره کلیه گره ها قابل دستیابی هستند. حذف و درج در این لیست با لیست یک طرفه تفاوت دارد و از مرتبه O(1) است. (بررسی شود)