تئوری و آزمایشگاه

طراحی الگوریتم‌ها

مطابق با سرفصل وزارت علوم، تحقیقات و فتاوری برای رشته های کامپیوتر و فناوری اطلاعات

مولف:

دکتر محمد علی ترکمانی

سرشناسه : ترکمانی ، محمد علی، ۱۳۵۴
عنوان و پدید آور : تئوری و آزمایشگاه طراحی الگوریتم ها/ مولف محمدعلی ترکمانی.
مشخصات نشر : مشهد: ارسطو، ۱۳۹۵.
مشخصات ظاهری : ۱۶۰ ص. مصور، جدول، نمودار.
شابک: ۵ - ۰۲۴- ۴۳۲- ۶۰۰ - ۹۷۸
وضعیت فهرست نویسی : فیپا
موضوع : الگوریتم‌های کامپیوتری -- راهنمای آموزشی (عالی)
موضوع : Computer algorithms -- Study and teaching (Higher)
موضوع : الگوریتم‌های کامپیوتری-- مسایل، تمرینها و غیره (عالی)
موضوع : Computer algorithms -- Problems, exercises, etc. (Higher)
موضوع : شبکه‌های کامپیوتری -- تدابیر ایمنی
موضوع : Computer networks -- Security measures
رده بندی کنگره : ۱۳۹۵ ۴ ت ۷ الف /QA۷۶/۹
رده بندی دیویی : ۰۰۵/۱۰۷۶
شماره کتابشناسی ملی : ۴۲۳۸۶۵۳

نام کتاب : تئوری و آزمایشگاه طراحی الگوریتم ها
مولف : دکتر محمد علی ترکمانی
ناشر : ارسطو (با همکاری سامانه اطلاع رسانی چاپ و نشر ایران)
صفحه آرایی ، تنظیم و طرح جلد : محمدعلی ترکمانی و علی بیات
تیراژ : ۱۰۰۰ جلد
نوبت چاپ : دوم – ۱۳۹۸
چاپ : مدیران
قیمت : ۳۵۰۰۰ تومان
شابک : ۵ - ۰۲۴- ۴۳۲- ۶۰۰ - ۹۷۸
تلفن های مرکز پخش : ۵۰۹۶۱۴۵ – ۵۰۹۶۱۴۶ - ۰۵۱۱

فهرست مطالب

مقدمه:

طراحی الگوریتم ها یکی از دروس اصـــلی و مهم در رشـــته های مهندســـی کامپیوتر و فناوری اطلاعات ا ست. این درس به دانشجویان کمک می‌کند که روش‌های کارآمد حل مسائل مختلف را بیاموزند. ا ستفاده از این تکنیکها در برنامه نویسی موجب میشود برنامه های نو شته شده سرعت اجرا و کارایی مناسـبی داشـته باشـدند. تاکید بیشـتر کتابهای موجود در این حوزه بر روی شـبه کدهای C یا C++ است. ضمنا حجم این کتابها نیز زیاد است و تدریس آنها در طول یک نیمسال دشوار است.

این کتاب به منظور سهولت کار ا ساتید و دان شجویان گرامی و همچنین افزایش کیفیت آموز شی درس طراحی الگوریتمها تدوین گردیده ا ست. این کتاب علاوه بر اینکه سرفصلهای مصوب وزارت علوم، تحقیقات و فناوری را پو شش میدهد، حجم منا سبی نیز دارد . ضمنا الگوریتم‌ها نیز با زبان C در محیط Visual C++ ارائه شـــده اند. به این ترتیب دانشـــجویان میتوانند الگوریتمها را به صـورت عملی اجرا و خروجی آنها را مشـاهده کنند. این روش خصـوصـا برای قسـمت کارگاهی و عملی درس طراحی الگوریتم در دانشگاه علمی و کاربردی مفید است.

در این کتاب سعی شده است که مطالب با زبانی ساده و به همراه مثال‌های متنوعی ارائه گردد. در انتهای هر فصل مجموعه ای از سئوالات مهم آورده شده ا ست تا دانشجویان گرامی بهتر بتوانند خود را برای آزمون‌هایی که در پیش رو دارند آماده نمایند. امید ا ست این اثر مورد توجه همکاران و دان شجویان گرامی قرار گیرد. از ا ساتید و دان شجویان گرامی تقا ضا دارم نقطه نظرات خود را از طریق ایمیل m.a.torkamani@gmail.com با اینجانب در میان بگذارند تا انشـــاالله در ویرایش‌های بعدی کتاب اشکالات یا کاستی‌های احتمالی آن، مورد تجدید نظر قرار گیرد. در پایان وظیفه خود می‌دانم از زحمات آقای مهندس علی بیات به خاطر طراحی جلد کتاب و همچنین مدیریت انتشارات ارسطو جناب آقای حسین قنبری تشکر و قدردانی نمایم.

محمد علی ترکمانی

تابستان ۱۳۹۵

تقدیم به:

دانشجویان عزیز

فصل اول

پیچیدگی زمانی و مرتبه اجرا

۱-۱-مرتبه اجرا

در ارزیابی یک الگوریتم دو فاکتور حافظه مصرفی و زمان اجرای الگوریتم دارای اهمیت می‌باشد. در طراحی الگوریتم زمان اجرای الگوریتم برای ما مهم تر است. در واقع تأکید این درس بر روی پیچیدگی زمان اجرا می باشد. به طور کلی زمان اجرای یک الگوریتم با افزایش اندازه ورودی (n) زیاد می‌شود. چنانچه عملیات اصلی الگوریتم تکرار شود (استفاده از حلقه[۱])، زمان اجرای الگوریتم نیز افزایش می‌یابد. بنابراین کارایی الگوریتم را با تعیین تعداد دفعاتی که یک عمل انجام می‌شود به عنوان تابعی از ورودی تحلیل می‌کنیم[۱].

مثال:

```
float sum(float list[], int n)
{
       float s = 0; // 1
              int  i,
              for (i = 0, i < n, i + +) // n + 1
                     s = s + list[i]; // n
                     return s // 1
}
```

در این مثال یک تابع به نام sum تعریف شده که دو پارامتر یا آرگومان را به عنوان ورودی دریافت پارامتر اول یک آرایه اعشاری (float) به نام list و پارامتر دوم یک متغیر از نوع int به نام n در این جا n تعداد اِلمان آرایه است. در این برنامه n مرتبه اجرا می‌باشد. به طور کلی تحلیل پیچیدگی زمانی یک الگوریتم بر اساس تعداد دفعاتی که عمل اصلی تکرار می‌گردد، محاسبه می شود. معمولاً

1 loop

پیچیدگی زمانی الگوریتم را با $T(n)$ نمایش می‌دهند. البته ممکن است زمان اجرا یک الگوریتم وابسته به چند ورودی باشد مانند $T(n,m)$. در مثال فوق $T(n)= n$ است[۱].

تعداد کل مراحل مثال فوق را محاسبه کنید؟

$1+n+1+n+1=2n+3$

نکته۱: اگر یک حلقه for ، n مرتبه اجرا شود، هنگام محاسبه تعداد کل مراحل برنامه باید $n+1$ را برای آن در نظر بگیریم. علت آن است که وقتی یک حلقه For اجرا می‌شود، بعد از هر بار اجرا به ابتدای حلقه باز می‌گردد و مجدداً شرط حلقه را چک می‌کند. لذا در برنامه فوق، بعد از n امین مرتبه که دستور s= s + list [i]; اجرا می‌شود، مجدداً شرط حلقه برای n امین مرتبه نیز بررسی خواهد شد.

نکته ۲: هنگام تعریف یک متغیر اگر مقدار دهی اولیه داشته باشد، یک واحد اجرا در نظر گرفته می‌شود. در غیر این صورت صفر را برای آن در نظر می‌گیریم. مانند دستور float و int در مثال فوق.

نکته ۳: هنگام محاسبه تعداد دفعاتی که یک دستور درون حلقه ها اجرا می‌شود می‌توان از رابطه های زیر استفاده نمود[۱].

$$\sum_{i=1}^{n} 1 = \mathbf{n}$$

$$\sum_{i=1}^{n} i = \mathbf{1} + \mathbf{2} + \mathbf{3} + \cdots + \mathbf{n} = \frac{n(n+1)}{2}$$

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

$$\sum_{i=1}^{n} kf(i) = k\sum_{i=1}^{n} f(i), \qquad k \text{ یک عدد ثابت است}$$

مثال) تعداد اجرای دستور اصلی در قطعه برنامه زیر و همچنین تعداد کل گامهای برنامه زیر چقدر است؟

```
for i = 1 to m do
   for  i = 1  to  n do
      x = x + 1; //   اصلي دستور
```

پاسخ: طبق چیزی که در زبانهای برنامه نویسی خوانده‌ایم، تعداد اجرای این قطعه برنامه برابر با mn است. اما اگر بخواهیم از نظر ریاضی محاسبه کنیم به صورت زیر عمل میکنیم. در اینجا دستور اصلی اگر به تنهایی بود فقط یک بار اجرا می‌شد اما چون داخل دو حلقه است، از رابطه زیر استفاده می‌شود:

$$T(n) = \sum_{i=1}^{m}\sum_{j=1}^{n} 1 = \sum_{i=1}^{m} n = \mathbf{n}\sum_{i=1}^{m} 1 = \mathbf{mn}$$

بنابراین دستور x=x+1; به اندازه mn اجرا می‌شود. حال باید تعداد اجرای حلقه ها را نیز در نظر بگیریم. حلقه داخلی اگر به تنهایی بود، n+1 بار اجرا میشد. اما چون در داخل یک حلقه دیگربا تکرار m است، به تعداد m(n+1) اجرا می‌شود. خود حلقه بیرونی نیز به اندازه m+1 اجرا خواهد شد. بنابراین تعداد کل اجرا برابر است با:

$$T(n) = m + 1 + m(n+1) + \mathbf{mn} = \mathbf{2m(n+1)} + \mathbf{1}$$

مثال) تعداد اجرای دستور را بیابید؟

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

روش اول[۱]:

j	تغییرات i	تعداد اجرا
1	1	1
2	1,2	2
.	.	.
.	.	.
.	.	.
3	1,2,3	3
.	.	.
.	.	.
.	.	.
n	1,2,…,n	N

تعداد اجرای حلقه داخلی دستور x=x+1 $= 1 + 2 + 3 + \cdots + n \Rightarrow \frac{n(n+1)}{2}$

روش دوم[۱]:

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

یادآوری:کف عدد x با علامت $\lfloor x \rfloor$ و سقف آن به صورت $\lceil x \rceil$ نمایش داده می‌شود.

مثال:

$\lfloor 3.2 \rfloor = 3 , \lfloor 3.7 \rfloor = 3$
$\lceil 3.2 \rceil = 4, \lceil 3.7 \rceil = 4$
$\lceil 3 \rceil = \lfloor 3 \rfloor = 3$

مثال) تعداد اجرای دستور اصلی را به ازای n=16 و n=14 بیابید؟

```
i = n;
while (i>1)
{
	i = i / 2;
	x++;
}
```

الف- n=16

I	شرط i>1	تعداد اجرای دستور اصلی	x
16	T	1	1
8	T	1	2
4	T	1	3
2	T	1	4
1	F		

$T(n) = \lfloor \log_2 16 \rfloor = 4$

ب-n=14

i	شرط i>1	تعداد اجرای دستور اصلی	X
14	T	1	1
7	T	1	2
3	T	1	3
1	F		

$T(n) = \lfloor \log_2 14 \rfloor = 3$

در حالت کلی دستور اصلی در این مثال به تعداد $\lfloor \log_2 n \rfloor$ بار اجرا می شود.

نکته: اغلب در کتاب های ساختمان داده و طراحی الگوریتم عبارت $\log_2 n$ با $\log n$ یا $\lg n$ نمایش داده می شود.

۲-۱-تحلیل پیچیدگی زمانی برای حالات بهترین، بدترین و متوسط

فرض کنید یک لیست نامرتب داریم که دارای n عدد می باشد می‌خواهیم یک عدد را در آن پیدا کنیم. بهترین حالت زمانی اتفاق می افتد که عدد مورد نظر ما در ابتدای لیست باشد. در این حالت با یک مقایسه می‌توانیم عدد مورد نظر را بیابیم. بدترین حالت زمانی است که عدد مورد نظر ما در انتهای لیست باشد. در این حالت تعداد مقایسه‌ها برابر است با n است. در حالت متوسط دفعات مقایسه $\frac{n+1}{2}$ خواهد بود یعنی به طور متوسط نیمی از عناصر آرایه باید جستجو شود. بنابراین می‌توانیم یک الگوریتم را از سه جنبه مورد بررسی قرار دهیم:

- تحلیل پیچیدگی زمانی برای بهترین حالت
- تحلیل پیچیدگی زمانی برای بدترین حالت
- تحلیل پیچیدگی زمانی برای حالت متوسط

۳-۱-مرتبه اجرای الگوریتم (O)

نماد مرتبه اجرای الگوریتم را با O بزرگ (big o) نمایش می دهند و آنرا order نیز می‌خوانند. در قسمت‌های قبل به طور دقیق محاسبه کردیم که دستور اصلی دقیقاً چند مرتبه اجرا می‌شود. به جای محاسبات دقیق، به روشی نیاز داریم که زمان اجرای الگوریتم‌ها را به صورت تقریبی نشان دهد. این بحث تا حدودی شبیه به بحث هم ارزی ها در مسائل ریاضیات است، مثلاً در ریاضیات خوانده‌اید که $\lim_{n\to\infty} 5n^2+4n-3$ هم ارز است با $\lim_{n\to\infty} 5n^2$. یعنی هنگامی که n افزایش می‌یابد، عبارت 4n-3 در مقابل 5n2 صرفنظر کردن می‌باشد.

در طراحی الگوریتم نیز مرتبه اجرایی یک الگوریتم به همین صورت محاسبه می‌کنیم. به عنوان مثال الگوریتمی که پیچیدگی آن $T(n) = 6n^2 + 7n - 4$ است را از مرتبه n^2 در نظر

گرفته و با $O(n^2)$ نمایش می‌دهیم. به همین ترتیب الگوریتمی که پیچیدگی زمانی آن $T(n) = 5n - 4$ است، از مرتبه n بوده و آن را با $O(n)$ نمایش می‌دهیم.

به طور کلی اگر $T(n)$ تابع زمانی الگوریتم باشد و $g(n)$ نیز یک کران بالا برای $T(n)$ باشد، آنگاه می‌نویسیم:

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

مثال:

الف-

```
x = x + 1 ; // O(1)
```

نکته: O (1) زمان محاسبات ثابتی را نشان میدهد که تابعی از n نیست.

ب-

```
for (i = 1; i <= n; i++) // O(n)
x + +;
```

ج-

```
for (j = 1; j <= n; j++) // O(n2)
for (i = 1; i <= n; i++)
x + +;
```

نکته: دو حلقه تودر تو از $O(n^2)$ و سه حلقه تو در تو از $O(n^3)$ است.

نکته: در حلقه While اگر شمارنده حلقه با دستور i=i/k از n تا ۱ تغییر کند و یا اگر شمارنده حلقه با دستور i=i×k از 1 تا n تغییر کند ، مرتبه اجرایی آن $O(log_k^n)$ است.

مثال:

```
x = 0;
i = n;
while (i>1)
{
       x++;
       i = i / z;
}
```

$$O(\log_2 n)$$

قانون جمع: فرض کنید T1(n) و T2(n) زمان اجرای دو قطعه برنامه باشد: