



تئوری و آزمایشگاه

# طراحی الگوریتم‌ها

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

فناوری اطلاعات

مؤلف:

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 ت ۴ ۷ الف / QA۷۶۹  
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 آرایبی ، 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 - ۵۱۱

این اثر مشمول قانون حمایت از مولفان و مصنفان و هنرمندان است. هر کس تمام یا قسمتی از این اثر را بدون اجازه مولف نشر یا پخش یا عرضه کند، مورد پیگرد قانونی قرار خواهد گرفت.

## فهرست مطالب

### فصل اول: پیچیدگی زمانی و مرتبه اجرا ..... ۱۰

- ۱-۱- مرتبه اجرا ..... ۱۰
- ۱-۲- تحلیل پیچیدگی زمانی برای حالات بهترین، بدترین و متوسط ..... ۱۴
- ۱-۳- مرتبه اجرای الگوریتم (O) ..... ۱۴
  - ۱-۳-۱- انواع توابع و مرتبه اجرایی آن ها به ترتیب صعودی ..... ۱۶
  - ۱-۳-۲- تعریف صوری تابع O ..... ۱۷
  - ۱-۴- نماد  $\Omega$  ..... ۱۸
  - ۱-۵- نماد  $\theta$  (حالت میانگین زمان اجرا) ..... ۱۹
  - ۱-۶- نمادهای O و  $\omega$  (ای کوچک و امگای کوچک) ..... ۲۱
  - ۱-۷- مقایسه خواص O،  $\Omega$ ،  $\theta$ ، و  $\omega$  ..... ۲۲
  - ۱-۸- سئوالات ..... ۲۴

### فصل دوم: توابع بازگشتی ..... ۲۵

- ۲-۱- مرتبه اجرای توابع بازگشتی ..... ۲۵
- ۲-۲- مسئله برج هانوی ..... ۲۷
  - ۲-۲-۱- محاسبه پیچیدگی تابع هانوی به روش جایگذاری ..... ۲۸
  - ۲-۳- ساختار برنامه کنسول در WIN32 در VISUAL C++ ..... ۲۹
  - ۲-۴- برنامه بزرگترین مقسوم علیه مشترک به روش بازگشتی ..... ۲۹
  - ۲-۵- روش های برهان خلف و استقراء ..... ۳۰

- ۱-۵-۲- سری فیبوناچی ..... ۳۱
- ۶-۲- الگوریتم تقسیم و غلبه ..... ۳۳
- ۷-۲- الگوریتم پویا ..... ۳۴
- ۸-۲- سئوالات ..... ۳۴

## فصل سوم: روش تقسیم و غلبه ..... ۳۵

- ۱-۳- معرفی ..... ۳۵
- ۱-۱-۳- تابع بازگشتی محاسبه فاکتوریل یک عدد صحیح ..... ۳۵
- ۲-۱-۳- الگوریتم اقلیدسی ..... ۳۶
- ۳-۱-۳- روش جست و جوی دودویی ..... ۳۷
- ۴-۱-۳- الگوریتمهای مرتب سازی ..... ۳۹
- ۱-۴-۳- مرتب سازی ادغامی ..... ۳۹
- ۲-۱-۴-۳- الگوریتم مرتب سازی سریع ..... ۴۱
- ۳-۱-۴-۳- مقایسه پیچیدگی محاسباتی مرتب سازی سریع و مرتب سازی ادغامی ..... ۴۲
- ۴-۱-۴-۳- نکات مرتب سازی ..... ۴۲
- ۵-۱-۳- ضرب ماتریس استراسن ..... ۴۳
- ۶-۱-۳- یافتن بزرگ ترین کلید ..... ۴۶
- ۲-۳- چه زمان هایی نباید از تقسیم و غلبه استفاده شود. .... ۴۷
- ۳-۳- سئوالات ..... ۴۸

## فصل چهارم: روش حریصانه ..... ۴۹

- ۱-۴- معرفی ..... ۴۹
- ۲-۴- الگوریتم پریم ..... ۴۹
- ۳-۴- الگوریتم کراسکال ..... ۵۱

- ۴-۴-الگوریتم دایجسترا (دیکسترا) ..... ۵۳
- ۴-۵-مسئله کوله پشتی به روش حرصانه ..... ۶۱
- ۴-۶-مسئله کوله پشتی ۰-۱ ..... ۶۶
- ۴-۷-سئوالات ..... ۶۷

## فصل پنجم: روش برنامه نویسی پویا ..... ۷۳

- ۵-۱-معرفی ..... ۷۳
- ۵-۲-مسئله سری فیبو ناچی ..... ۷۴
- ۵-۳-محاسبه ضرب دو جمله ای ..... ۷۶
- ۵-۴-الگوریتم فلوید ..... ۷۸
- ۵-۵-روش پویا برای حل دسته مسائل بهینه سازی ..... ۸۳
- ۵-۶-ضرب زنجیره ای ماتریس ها ..... ۸۴
- ۵-۷-فروشنده دوره گرد ..... ۹۲
- ۵-۸-بزرگترین زیر دنباله (رشته) مشترک (LCS) ..... ۹۶
- ۵-۹-مسئله ای که حل آنها معادل با جمله ای از سری کاتالان است ..... ۹۷
- ۵-۱۰-مسئله کوله پشتی ۰-۱ ..... ۹۹
- ۵-۱۱-سئوالات ..... ۱۰۳

## فصل ششم: تکنیک عقبگرد ..... ۱۰۷

- ۶-۱-مفهوم عقبگرد (BACK TRACKING) ..... ۱۰۷
- ۶-۲-مسئله N وزیر ..... ۱۰۸
- ۶-۲-۱-شبه کد مسئله n وزیر ..... ۱۱۲
- ۶-۲-۲-پیاده سازی مسئله n وزیر با Visual c++ ..... ۱۱۶

- ۱۱۸ ..... ۳-۶- رنگ آمیزی گراف (GRAPH COLORING)
- ۱۲۲ ..... ۴-۶- مسئله حاصل جمع زیر مجموعه‌ها
- ۱۲۵ ..... ۵-۶- حل مسئله کوله پشتی صفر و یک با تکنیک عقبگرد
- ۱۲۶ ..... ۶-۶- مسائل

## فصل هفتم: انشعاب و تحدید ..... ۱۲۹

- ۱۲۹ ..... ۱-۷- مقدمه
- ۱۳۲ ..... ۲-۷- بهترین جستجو با هرس شاخه و حد
- ۱۳۲ ..... ۱-۲-۷- ساختار کلی روش شاخه و حد
- ۱۳۳ ..... ۳-۷- مسئله فروشنده دورگرد به روش انشعاب و تحدید
- ۱۴۰ ..... ۴-۷- الگوریتم جستجوی اول بهترین
- ۱۴۲ ..... ۵-۷- جستجوی اول بهترین با هرس شاخه و حد برای مسئله کوله پشتی ۰ و ۱
- ۱۴۶ ..... ۶-۷- الگوریتم جستجوی اول بهترین: بهترین با هرس شاخه و حد برای مسئله کوله پشتی صفر و یک
- ۱۵۱ ..... ۷-۷- سئوالات

## فصل هشتم: مسائل رام نشدنی ..... ۱۵۳

- ۱۵۳ ..... ۱-۸- مقدمه
- ۱۵۳ ..... ۲-۸- مسائل رام نشدنی
- ۱۵۴ ..... ۱-۲-۸- مسائلی که الگوریتم‌های زمان چند جمله‌ای برای آنها پیدا شده است؟
- ۱۵۴ ..... ۲-۲-۸- مسائلی که رام نشدنی بودن آنها ثابت شده است؟
- ۱۵۵ ..... ۳-۲-۸- مسائلی که رام نشدنی بودن آنها ثابت نشده ولی تاکنون هیچ الگوریتم‌زمان چند جمله‌ای برای آنها پیدا نشده است؟
- ۱۵۵ ..... ۳-۸- نظریه NP

- ۱۵۶ ..... (Polynomial) P کلاس ۳-۸-۱
- ۱۵۶ ..... ( Nondeterministic polynomial ) NP کلاس ۳-۸-۲
- ۱۵۸..... NP-COMPLETE و NP-HARD ۴-۸-۴
- ۱۵۹..... NP-hard کلاس ۱-۴-۸-۴
- ۱۶۰ ..... NP-complete کلاس ۲-۴-۸-۴
- ۱۶۰ ..... NP-Complete چند نمونه مثال از مسائل ۱-۴-۲-۸-۴
- ۱۶۱ ..... ۵-۸-سوالات
- ۱۶۲ ..... مراجع

## مقدمه:

طراحی الگوریتم‌ها یکی از دروس اصلی و مهم در رشته‌های مهندسی کامپیوتر و فناوری اطلاعات است. این درس به دانشجویان کمک می‌کند که روش‌های کارآمد حل مسائل مختلف را بیاموزند. استفاده از این تکنیک‌ها در برنامه‌نویسی موجب می‌شود برنامه‌های نوشته شده سرعت اجرا و کارایی مناسبی داشته باشند. تاکید بیشتر کتاب‌های موجود در این حوزه بر روی شبه کدهای C یا C++ است. ضمناً حجم این کتاب‌ها نیز زیاد است و تدریس آنها در طول یک نیمسال دشوار است.

این کتاب به منظور سهولت کار اساتید و دانشجویان گرامی و همچنین افزایش کیفیت آموزشی درس طراحی الگوریتم‌ها تدوین گردیده است. این کتاب علاوه بر اینکه سرفصل‌های مصوب وزارت علوم، تحقیقات و فناوری را پوشش می‌دهد، حجم مناسبی نیز دارد. ضمناً الگوریتم‌ها نیز با زبان C در محیط Visual C++ ارائه شده‌اند. به این ترتیب دانشجویان می‌توانند الگوریتم‌ها را به صورت عملی اجرا و خروجی آنها را مشاهده کنند. این روش خصوصاً برای قسمت 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 مجموعه‌ای از سئوالات مهم آورده شده است تا دانشجویان گرامی بهتر بتوانند خود را برای آزمون‌هایی که در پیش رو دارند آماده نمایند. امید است این اثر مورد توجه همکاران و دانشجویان گرامی قرار گیرد. از اساتید و دانشجویان گرامی تقاضا دارم نقطه نظرات خود را از طریق ایمیل [m.a.torkamani@gmail.com](mailto:m.a.torkamani@gmail.com) با اینجانب در میان بگذارند تا انشاءالله 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 می‌باشد. به طور کلی زمان اجرای یک الگوریتم با افزایش اندازه ورودی ( $n$ ) زیاد می‌شود. چنانچه عملیات اصلی الگوریتم تکرار شود (استفاده از حلقه)، 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 المان 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 می‌شود. معمولاً

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؟

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

نکته ۱: 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 حلقه‌ها اجرا می‌شود می‌توان از رابطه‌های زیر استفاده نمود [۱].

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

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

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

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

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:

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

بنابراین دستور  $x=x+1$  به اندازه  $mn$  اجرا می‌شود. حال باید تعداد اجرای حلقه‌ها را نیز در نظر بگیریم. حلقه داخلی اگر به تنهایی بود،  $n+1$  بار اجرا میشود. اما چون در داخل یک حلقه دیگر با 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:

$$T(n) = m + 1 + m(n + 1) + mn = 2m(n + 1) + 1$$

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

$$x=x+1 \text{ دستور } \Rightarrow 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

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  $[x]$  و سقف آن به صورت  $\lceil x \rceil$  نمایش داده می‌شود.  
مثال:

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

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

$$\lfloor 3 \rfloor = \lceil 3 \rceil = 3$$

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 می‌شود.

نکته: اغلب در کتاب‌های ساختمان داده و طراحی الگوریتم عبارت  $\log_2 n$  با  $\log n$  یا  $\lg n$  نمایش داده می‌شود.

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

فرض کنید یک لیست نامرتب داریم که دارای  $n$  عدد می‌باشد می‌خواهیم یک عدد را در آن پیدا کنیم. بهترین حالت زمانی اتفاق می‌افتد 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 می‌دهند 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 هم‌ارزی‌ها در مسائل ریاضیات است، مثلاً در ریاضیات خواننده‌اید که  $\lim_{n \rightarrow \infty} 5n^2 + 4n - 3$  هم‌ارز است با  $\lim_{n \rightarrow \infty} 5n^2$ . یعنی هنگامی که  $n$  افزایش می‌یابد، عبارت  $4n - 3$  در مقابل  $5n^2$  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(n^2)
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  $i=i \times k$  از ۱ 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  $T_1(n)$  و  $T_2(n)$  زمان اجرای دو قطعه برنامه باشد: