دانشکده‌ی علوم ریاضی

ساختمان داده‌ها ۲ مهر ۱۳۹۲

جلسه‌ی ۴: تحلیل مجانبی الگوریتم‌ها

مدرّس: دکتر شهرام خزائی *نگارنده: شراره عزّت‌نژاد، آرمینا ثابتی‌اشرف*

۱ مقدمه

الگوریتم ابزاری است که از آن برای حل مسأله استفاده می‌شود. گاهی اوقات بیش از یک الگوریتم برای حل یک مسأله وجود دارد؛ بنابراین بهینه بودن ابزارها حائز اهمیت است. مهم‌ترین پارامترهای بهینه بودن یک الگوریتم، میزان حافظه و زمانی است که برای اجرا مصرف می‌کند و این دو پارامتر، معیارهایی برای سنجش کارآیی الگوریتم هستند.
در این درس به تحلیل زمانی الگوریتم‌ها می‌پردازیم. تحلیل زمانی الگوریتم معمولاً به بررسی زمان اجرا در بدترین حالت[۱] و یا حالت‌میانگین[۲] می‌پردازد. توجه کنید که بهترین زمان اجرای الگوریتم اهمیت چندانی ندارد، چرا که این حالت معمولاً برای ورودی‌های خاص و یا با احتمال خیلی کم اتفاق می‌افتد. در علوم کامپیوتر برای اشاره به زمان اجرای الگوریتم‌ها از لفظ پیچیدگی[۳] استفاده می‌شود. به طور دقیق‌تر منظور از پیچیدگی، تابعی است که زمان اجرای الگوریتم را برحسب اندازه‌ی ورودی بیان می‌کند. در بحث تحلیل الگوریتم‌ها بجای درنظر گرفتن رفتار دقیق توابع، رفتار مجانبی آنها مورد بررسی قرار می‌گیرد. در ادامه بحث به چگونگی محاسبه‌ی آن خواهیم پرداخت. بعضی مواقع پیچیدگی حالت میانگین و بدترین حالت با یکدیگر تفاوت چندانی نمی‌کند اما معمولاً پیچیدگی الگوریتم در حالت میانگین بهتر از پیچیدگی آن در بدترین حالت است؛ چراکه اغلب بدترین حالت الگوریتم به‌ندرت اتفاق می‌افتد. از این رو پیچیدگی حالت میانگین الگوریتم‌ها همان رفتار مورد انتظار ماست.
با استفاده از پیچیدگی الگوریتم‌ها، می‌توانیم آن‌ها را از نظر میزان کارآیی با یکدیگر مقایسه کنیم. برای مثال، زمان اجرای الگوریتم‌های مرتب‌سازی ادغامی[۴] و مرتب‌سازی درجی[۵] را مقایسه می‌کنیم. می‌دانیم الگوریتم مرتب‌سازی ادغامی از مرتبه‌ی $n \log n$ و الگوریتم مرتب‌سازی درجی از مرتبه $n^۲$ است. همان‌طور که مشاهده می‌شود، برای مقادیر کوچک الگوریتم مرتب‌سازی درجی سریع‌تر از الگوریتم مرتب‌سازی ادغامی عمل می‌کند اما هرچه مقدار n افزایش می‌یابد، الگوریتم مرتب‌سازی ادغامی بسیار سریع‌تر از الگوریتم مرتب‌سازی درجی عمل می‌کند. بنابراین از آن‌جا که اغلب برای ورودی‌های خیلی بزرگ از الگوریتم‌ها استفاده می‌کنیم، به تحلیل الگوریتم‌ها برای nهای بزرگ می‌پردازیم.
در این جلسه سعی بر آن است تا روش‌هایی برای مقایسه زمان اجرای الگوریتم‌ها بیان کنیم تا به درک روشنی از میزان بهینه بودن الگوریتم‌ها برسیم. در ابتدا به معرفی علائم قراردادی می‌پردازیم و سپس مفاهیم، تعاریف و مثال‌هایی از آن‌ها بیان می‌کنیم.

[۱] worst-case
[۲] average-case
[۳] complexity
[۴] merge sort
[۵] insertion sort

۲ نمادهای مجانبی

نمادهایی که ما از آن‌ها برای نشان دادن کران مجانبی زمان اجرای یک الگوریتم استفاده می‌کنیم، بر حسب توابعی تعریف شده‌اند که دامنه آن‌ها مجموعه‌ای از اعداد طبیعی است. دامنه‌ی این توابع اندازه‌ی ورودی الگوریتم را نشان می‌دهد.

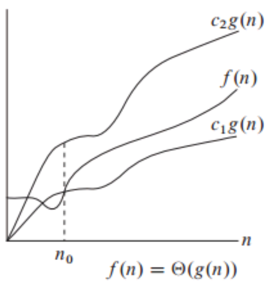
۱.۲ نماد Θ

تعریف ۱ *مجموعه‌ی توابع از مرتبه‌ی Θی تابع $g(n)$ با $\Theta(g(n))$ نشان داده می‌شود و به صورت زیر تعریف می شود:*

$$\Theta(g(n)) = \{f(n) : \exists\ c_1\ , c_2\ ,\ n_0 > 0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall\ n \geq n_0\}$$

به عبارت دیگر، اگر ضرایب ثابت c_1 و c_2 وجود داشته باشند که برای nهای به اندازه کافی بزرگ تابع $f(n)$ بین $c_1 g(n)$ و $c_2 g(n)$ قرار بگیرد، تابع $f(n)$ به مجموعه‌ی $\Theta(g(n))$ تعلق دارد.

چون $\Theta(g(n))$ یک مجموعه است، می‌توانیم بنویسیم $f(n) \in \Theta(g(n))$ تا نشان دهیم $f(n)$ عضوی از آن است، با این حال به جای آن از $f(n) = \Theta(g(n))$ استفاده می‌کنیم تا مفهوم مشابهی را نشان دهیم. تصویر ۱، نشان می‌دهد که $f(n) = \Theta(g(n))$ است. برای تمام n های بزرگ‌تر از n_0، تابع $f(n)$ در بالای $c_1 g(n)$ و در پایین $c_2 g(n)$ قرار دارد. به عبارت دیگر، برای تمام $n > n_0$ ها، تابع $f(n)$ برابر ضریبی از تابع $g(n)$ است که این ضریب همواره بین c_1 و c_2 قرار دارد. در نتیجه می‌توانیم بگوییم $g(n)$ یک **کران دوطرفه مجانبی**[۶] برای $f(n)$ است. هم‌چنین، در اصلاح می‌گوییم $f(n)$ از مرتبه‌ی Θی $g(n)$ است.



شکل ۱

نکته ۱ *برای این که تعریف $\Theta(g(n))$ درست باشد، باید هر $f(n) = \Theta(g(n))$ به طور مجانبی نامنفی باشد. یعنی، وقتی n به اندازه‌ی کافی بزرگ است، مقدار $f(n)$ نامنفی باشد. هم‌چنین، خود تابع $g(n)$ هم باید به طور مجانبی نامنفی باشد، وگرنه مجموعه $\Theta(g(n))$ تهی خواهد بود. در نتیجه ما فرض می‌کنیم هر تابعی که درون نماد Θ قرار می‌گیرد به طور مجانبی نامنفی است. این فرض را برای نمادهای دیگری هم که در ادامه معرفی می‌شوند در نظر می‌گیریم.*

مثال ۱ *ثابت کنید که $100n^2 + 5n - 4 \neq \Theta(n^3)$*
برهان. *باید نشان داد که نمی‌توان مقادیر مثبتی برای c_1 ، c_2 و n_0 یافت که رابطه‌ی $c_1 n^3 \leq 100n^2 + 5n - 4 \leq c_2 n^3$ برای همه مقادیر $n \geq n_0$ برقرار باشد. به‌وضوح به ازای هر مقدار $c_1 > 0$ و برای مقادیر بزرگ n داریم*

$$c_1 n^3 \nleq 100n^2 + 5n - 4$$

[۶] asymptotically tight bound

مثال ۲ *همان‌طورکه در جلسات قبل مشاهده کردیم، زمان اجرای مرتب‌سازی درجی $T(n) = an^۲ + bn + c$ است. می‌توان نشان داد $T(n) = \Theta(n^۲)$ است، زیرا به‌سادگی می‌توان مقادیر مثبتی برای $c_۱$ و $c_۲$ یافت که برای nهای به اندازه کافی بزرگ $c_۱n^۲ \leq an^۲ + bn + c \leq c_۲n^۲$ باشد.*

لم ۱ *یک تابع چندجمله‌ای، از مرتبه‌ی دقیق جمله‌ی با بزرگترین توانش است. یعنی با فرض $k > ۰$ و $a_k > ۰$ داریم:*

$$a_k n^k + a_{k-۱} n^{k-۱} + \cdots + a_۰ = \Theta(n^k)$$

مثال ۳ *نامساوی زیر برای تابع فاکتوریل برقرار است:*

$$n! = \sqrt{۲\pi n}(\frac{n}{e})^n e^{\alpha_n}$$

که

$$\frac{۱}{۱۲n+۱}\alpha_n < \frac{۱}{۱۲n}.$$

بنابراین می‌توان نوشت:

$$n! = \sqrt{۲\pi n}(\frac{n}{e})^n(۱ + \Theta(\frac{۱}{n})).$$

(توجه کنید که برای $|x| \leq ۱$ داریم $۱ + x \leq e^x \leq ۱ + x + x^۲$.)

سؤال ۱ *تابع $\sum_{i=۱}^{n} \frac{۱}{i}$ از چه مرتبه‌ای است؟ (جواب $\Theta(\log n)$).*

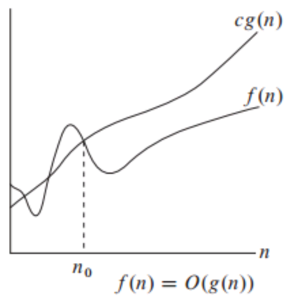
۲.۲ نماد $\mathcal{O}$

نماد Θ، یک تابع را از بالا و پایین به طور مجانبی کران‌دار می‌کند. هنگامی که فقط **کران بالای مجانبی**[۷] برای تابعی مطرح باشد، از نماد $\mathcal{O}$ استفاده می‌کنیم.

تعریف ۲ *مجموعه‌ی توابع از مرتبه‌ی $\mathcal{O}$ی تابع $g(n)$ با $\mathcal{O}(g(n))$ نشان داده می‌شود و به صورت زیر تعریف می شود:*

$$\mathcal{O}(g(n)) = \{f(n) : \exists\, c\ ,\ n_۰ > ۰ : ۰ \leq f(n) \leq cg(n),\ \forall\ n \geq n_۰\}$$

تصویر ۲، نشان می‌دهد که برای تمام n های بزرگ‌تر از $n_۰$، مقادیر تابع $f(n)$ در پایین $cg(n)$ قرار دارند. یعنی برای nهای به اندازه‌ی کافی بزرگ، ضریب ثابتی از تابع $g(n)$ کران بالای تابع $f(n)$ است. برای این که نشان دهیم $f(n)$ عضوی از $\mathcal{O}(n)$ است، $f(n) = \mathcal{O}(g(n))$ را می‌نویسیم.



شکل ۲

[۷]asymptotically upper bound

وقتی می‌گوییم "پیچیدگی زمانی یک الگوریتم $\mathcal{O}(g(n))$ است"، منظورمان این است که یک تابع $f(n)$ از $\mathcal{O}(g(n))$ وجود دارد که برای مقادیر به اندازه کافی بزرگ n، بدون توجه به این که چه ورودی‌ای با اندازه‌ی n انتخاب شده است، زمان اجرای الگوریتم برای آن ورودی از بالا با ضریب ثابت مثبتی از $f(n)$ کران‌دار شده است؛ یا به عبارتی زمان اجرا در بدترین حالت $\mathcal{O}(g(n))$ است.

مثال ۴ *روابط زیر برقرارند:*

$$n^2 = O(n^2)$$
$$n \log n = O(n^2)$$
$$n^2 \log n \neq O(n^2)$$

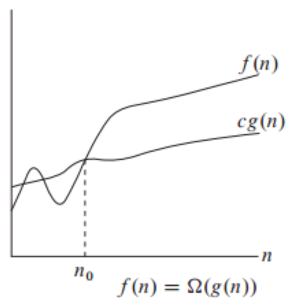
۳.۲ نماد Ω

همان‌طور که نماد $\mathcal{O}$ کران بالای مجانبی یک تابع را مشخص می‌کند، نماد Ω **کران پایین مجانبی**[۸] توابع را نشان می‌دهد.

تعریف ۳ *مجموعه‌ی توابع از مرتبه‌ی Ωی تابع $g(n)$ با $\Omega(g(n))$ نشان داده می‌شود و به صورت زیر تعریف می شود:*

$$\Omega(g(n)) = \{f(n) : \exists\, c\ ,\ n_0 > 0 : 0 \leq cg(n) \leq f(n),\ \forall\ n \geq n_0\}$$

تصویر ۳، نشان می‌دهد که برای تمام n های بزرگ‌تر از n_0، مقادیر تابع $f(n)$ در بالای $cg(n)$ قرار دارند. یعنی برای nهای به اندازه‌ی کافی بزرگ، تابع $g(n)$ کران پایین تابع $f(n)$ است.



شکل ۳

وقتی می‌گوییم "پیچیدگی زمانی یک الگوریتم $\Omega(g(n))$ است"، منظورمان این است که برای هر مقدار به اندازه کافی بزرگ n، بدون توجه به این که چه ورودی‌ای با اندازه‌ی n انتخاب شده است، زمان اجرای الگوریتم برای آن ورودی حداقل ضریب ثابتی از $g(n)$ است؛ یا به عبارتی زمان اجرا در بهترین حالت $\Omega(g(n))$ است.

مثال ۵ *روابط زیر برقرارند:*

$$\sqrt{n} = \Omega(\log n)$$
$$n^3 = \Omega(n^2)$$

مثال ۶ *به ازای هر مقدار $\varepsilon > 0$، داریم*

$$\log n \neq \Omega(n^{\varepsilon}),$$

به عبارت دیگر، تابع لگاریتم دارای کران پایین مجانبی چندجمله‌ای نیست.

[۸] asymptotically lower bound

با توجه به تعاریف نمادهای مجانبی، اثبات قضیه‌ی مهم زیر آسان است:
با توجه به تعاریف نمادهای مجانبی، اثبات قضیه‌ی مهم زیر آسان است:

قضیه ۲ *شرط لازم و کافی برای این که $f(n) = \Theta(g(n))$ باشد آن است که $f(n) = O(g(n))$ و $f(n) = \Omega(g(n))$ باشند.*

۳ خواص نمادهای مجانبی

بسیاری از خواص روابط اعداد حقیقی، در هنگام مقایسه‌ی مجانبی هم برقرارند. در ادامه به تعدادی از این خواص اشاره شده است.

۱.۳ خاصیت تراگذاری

لم ۳ *نمادهای تعریف شده خاصیت تراگذاری[۹] دارند.*

۱. از $f(n) = \Theta(g(n))$ و $g(n) = \Theta(h(n))$ نتیجه می‌شود $f(n) = \Theta(h(n))$.

۲. از $f(n) = O(g(n))$ و $g(n) = O(h(n))$ نتیجه می‌شود $f(n) = O(h(n))$.

۳. از $f(n) = \Omega(g(n))$ و $g(n) = \Omega(h(n))$ نتیجه می‌شود $f(n) = \Omega(h(n))$.

۲.۳ خاصیت بازتابی

لم ۴ *نمادهای مجانبی تعریف شده خاصیت بازتابی[۱۰] دارند.*

۱. $f(n) = \Theta(f(n))$

۲. $f(n) = O(f(n))$

۳. $f(n) = \Omega(f(n))$

۳.۳ خاصیت تقارنی

از بین نمادهای مجانبی تعریف شده، تنها نماد Θ خاصیت تقارنی[۱۱] دارد.

لم ۵ *$f(n) = \Theta(g(n))$ است اگر و تنها اگر $g(n) = \Theta(f(n))$ باشد.*

۴.۳ خاصیت تقارنی ترانهاده

نمادهای مجانبی O و Ω خاصیت تقارنی‌ترانهاده[۱۲] زیر را دارند:

لم ۶ *شرط لازم و کافی برای این که $f(n) = O(g(n))$ باشد آن است که $g(n) = \Omega(f(n))$ باشد.*

[۹] transitive
[۱۰] reflexive
[۱۱] symmetry
[۱۲] transpose symmetry

۴ پیچیدگی زمانی حلقه‌های تودرتو

معمولا می‌توان با بررسی ساختار کلی الگوریتم، کران بالای آن را پیدا کرد. در ادامه مثال‌هایی از پیدا کردن پیچیدگی زمانی الگوریتم با بررسی ساختار کلی آن ارائه می‌گردند.

۱.۴ $\Theta(n)$

هنگامی که در یک الگوریتم یک حلقه وجود دارد که از یک عدد ثابت تا n می‌رود، یا دو حلقه‌ی تودرتو وجود دارند یکی از آن‌ها به n وابسته نیست، پیچیدگی زمانی در بدترین حالت به وضوح از $\Theta(n)$ است، زیرا اعمال حلقه‌ی درونی، دارای کران بالای ثابت ($\Theta(1)$) هستند و در حالتی که دو حلقه وجود دارند، یکی از آن‌ها تعداد دفعات ثابتی اجرا می‌شود.

```
for i = 1 to n do
    do some operations in constant time.
```

```
for i = 1 to n do
    for j = 1 to c do
        do some operations in constant time.
```

۲.۴ $\Theta(n^2)$

هنگامی که در یک الگوریتم (مانند مرتب‌سازی درجی) دو حلقه‌ی تودرتو وجود دارند که از یک عدد ثابت تا n می‌روند، پیچیدگی زمانی در بدترین حالت به وضوح از $\Theta(n^2)$ است، زیرا اعمال حلقه‌ی درونی، دارای کران بالای ثابت ($\Theta(1)$) هستند. متغیرهای i و j هر کدام حداکثر به مقدار n می‌رسند، در نتیجه حلقه‌ی درونی حداکثر n^2 بار به ازای هر جفت از i و j های ممکن اجرا می‌شود. پس در کل حداکثر n^2 عملیات انجام می‌شوند.

```
for i = 1 to n do
    for j = 1 to n do
        do some operations in constant time.
```

```
for i = 1 to n do
    for j = i to n do
        do some operations in constant time.
```

۳.۴ $\Theta(n^3)$

هنگامی که در یک الگوریتم سه حلقه‌ی تودرتو وجود دارند که از یک عدد ثابت تا n می‌روند، پیچیدگی زمانی در بدترین حالت به وضوح از $\Theta(n^3)$ است، زیرا اعمال حلقه‌ی درونی، دارای کران بالای ثابت ($\Theta(1)$) هستند. متغیرهای

i و j و k هر کدام حداکثر به مقدار n می‌رسند، در نتیجه حلقه‌ی درونی حداکثر $n^۳$ بار به ازای هر سه تایی از i و j و k های ممکن اجرا می‌شود. پس در کل حداکثر $n^۳$ عملیات انجام می‌شوند.

```
for i = 1 to n do
    for j = 1 to n do
        for k = 1 to n do
            do some operations in constant time.
```

```
for i = 1 to n do
    for j = i to n do
        for k = j to n do
            do some operations in constant time.
```

۴.۴ $\Theta(\log n)$

هنگامی که در یک الگوریتم حلقه‌ای وجود دارد که از یک عدد ثابت تا n می‌رود و در هر مرحله متغیر حلقه دوبرابر می‌شود (یا از n شروع می‌شود و هر دفعه نصف می‌شود (مانند جست‌وجوی دودویی))، پیچیدگی زمانی در بدترین حالت به وضوح از $\Theta(\log n)$ است، زیرا اعمال حلقه‌ی درونی، دارای کران بالای ثابت ($\Theta(۱)$) هستند. متغیر i هر دفعه دوبرابر می‌شود، پس در $\log n$ مرحله به مقدار n می‌رسند، در نتیجه اعمال درونی حلقه حداکثر $\log n$ بار اجرا می‌شوند.

```
i ← 1
while i ≤ n do
    i ← 2i
    do some operations in constant time.
```

سؤال ۲ *اگر خط $i \leftarrow ۲i$ با $i \leftarrow i^۲$ جایگزین شود، پیچیدگی الگوریتم از چه مرتبه‌ای است؟ (جواب $\Theta(\log\log n)$).*