دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۱۰

نگارنده: مائده کاملان

۳۰ مهر ۱۴۰۰

فهرست مطالب

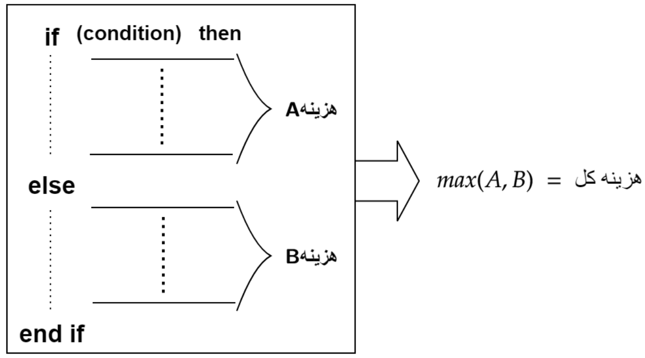
۱ نکاتی در تحلیل پیچیدگی زمانی واقعی

در تحلیل پیچیدگی زمانی واقعی برای هر عمل یک هزینه لحاظ کردیم و پیچیدگی زمانی کلی را از مجموع هزینه تمام خط‌های الگوریتم بدست آوردیم.

برخی نکات در رابطه با تحلیل پیچیدگی واقعی:

۱. دستوری برای کامنت گذاری در شبه کدها (comment//) معرفی کردیم در رابطه با تحلیل پیچیدگی واقعی، هزینه خط‌هایی از شبه کد که مربوط به توضیحات است صفر است.

۲. برای مثال‌هایی که تاکنون دیدیم برای هر خط هزینه‌ای درنظر می‌گرفتیم، با اینحال تمام دستورات داخل حلقه درون شبه کد ممکن است اجرا نشود. دراین راستا، اگر دستور if داشته باشیم، هزینه خط‌هایی که مربوط به دستور if است، به طور معمول به صورت زیر در نظر گرفته می‌شود:



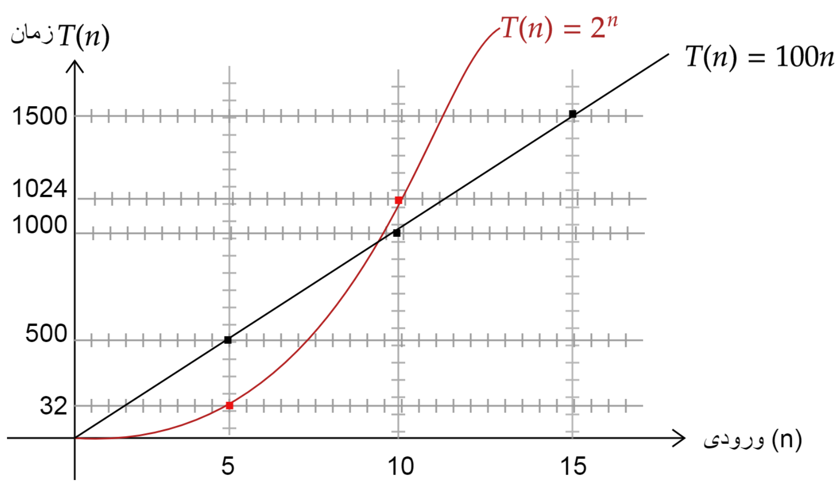
شکل ۱: هزینه شرط if در تحلیل پیچیدگی

۳. در تحلیل الگوریتم‌ها دیدیم که بهترین حالت اهمیت چندانی ندارد زیرا برای یک مجموعه ورودی، تحلیل الگوریتم در حالت کلی اهمیت دارد نه نمونه‌های خاصی که بهترین حالت را نتیجه بدهند. همچنین در حالت متوسط هم محاسبه هزینه و پیچیدگی زمانی واقعی، سختی محاسباتی دارد و غالبا با تحلیل احتمالاتی بدست می‌آید. با این اوصاف، ارزیابی کارایی یک الگوریتم به طور معمول برای نمونه‌های بزرگ ورودی لحاظ می‌شود.

نکته: الگوریتم‌هایی که پیچیدگی نمایی دارند ممکن است برای نمونه‌های کوچک، حتی از یک الگوریتم با پیچیدگی خطی هم بهتر عمل کند. در این رابطه به مثال زیر دقت کنید.

مثال: الگوریتم A_1 با تابع رشد $T(n) = 100n$ و الگوریتم A_4 با تابع رشد$T(n) = 2^n$ در نظر بگیرید. همچنین زوج مرتب $(A_1{,}A_4)$ را که برابر است با زمان اجرای هر یک از الگوریتم‌ها در نظر بگیرید:

$$(A_1{,}A_4) = (100\text{n},2^n) = \{(500, 32 : n = 5), (1000, 1024 : n = 10), (1500, 32768 : n = 15), \cdots\}$$



نکته: در عمل ممکن است نمونه‌ی شما کوچک باشد، با توجه به نمودار بالا، متوجه می‌شویم که برای نمونه‌های کوچک، الگوریتم با پیچیدگی نمایی بهتر جواب می‌دهد. پس الگوریتم ترجیحی شما برای کاربردتان می‌تواند الگوریتم نمایی باشد. یا ممکن است ترجیح دهید ترکیبی از الگوریتم‌ها را به کار گیرید، مثلا الگوریتمی مثل A^* داشته باشیم که برای نمونه‌های بزرگ از A_1 و برای نمونه‌های کوچک از A_4 استفاده کند.

۲ حرکت از تحلیل واقعی به سوی تحلیل مجانبی

تحلیل واقعی به دلیل انتساب هزینه به هر عمل، دارای اشکالات و ابهامات زیر است:

۱. آیا می شود شبه کد ساده‌تری نوشت که هزینه ی کمتری داشته باشیم؟

۲. الگوریتم روی چه سیستمی (با چه منابع محاسباتی و ذخیره‌سازی) اجرا شده است؟ برای مقایسه دو الگوریتم روی یک سیستم جدید، باید هر دو را دوباره اجرا و هزینه عملیاتی هر کدام را محاسبه کرد تا بتوان مقایسه را انجام داد. این شرایط از سیستمی به سیستم دیگر ممکن است متفاوت باشد.

۳. کار و دقت بیشتر در تحلیل،

۴. عدم ضرورت، چرا که در نمونه‌های بزرگ ورودی، ضرایب ثابت (هزینه‌ها) بی‌تاثیر می‌شود.

بنابراین از تحلیل مجانبی استفاده می‌کنیم چرا که یک دید کلی از الگوریتم ارائه می‌دهد. تحلیل مجانبی کار تحلیل الگوریتم را ساده‌تر می‌کند.
به یاد آورید برای الگوریتم مرتب‌سازی درجی سه حالت با فرم کلی زیر را در اختیار داشتیم:

۱. بهترین حالت: $An + B$

۲. بدترین حالت:$An + B + cn^2$

۳. حالت متوسط: $An + B + cn^2$

به دنبال مفهومی می‌گردیم که حتی از این فرم هم کلی‌تر باشد. مثلا به جای $An + B$در بهترین حالت و $An + B + cn^2$در بدترین حالت و حالت متوسط، بگوییم پیچیدگی در بهترین حالت n و در بدترین حالت و حالت متوسط n^2 است. در واقع بزرگترین مرتبه n در چندجمله‌ای را مد نظر قرار دهیم و ضرایب و مرتبه‌های کمتر را حذف کنیم.

۳ مقایسه رشد توابع

شبیه عملگرهای مقایسه‌ای برای اعداد حقیقی ($\geq, >, =, \leq, <$)، برای توابع پیچیدگی زمانی نیز می‌توان (نمادها) عملگرهایی را تعریف کرد:

۱. o: أی کوچک (معادل $>$ کوچکتری)،

۲. O: أی بزرگ (معادل کوچکتر و مساوی $\leq$)،

۳. Θ: تتا (معادل تساوی $=$) ،

۴. Ω: امگا بزرگ (معادل بزرگتر و مساوی $\geq$)،

۵. ω: امگا کوچک (معادل $<$ بزرگتری).

دو هدف از معرفی این نمادها دنبال می‌کنیم:
۱.دو تابع پیچیدگی چه وضعیتی نسبت به هم دارند؟
۲.یک دسته‌بندی و کلاس‌بندی برای الگوریتم‌ها از نظر پیچیدگی بدست آید.
حال هر یک از نمادهای مقایسه‌ای برای توابع پیچیدگی زمانی را به تفصیل معرفی می‌کنیم:

۱. **نماد O (Big-O):**

- **تعریف ۱**:فرض کنید که f(n) و g(n) دو تابع پیچیدگی باشند، می‌گوییم رشد تابع f(n) حداکثر به اندازه رشد تابع g(n) است و به صورت $f(n) = O(g(n))$می‌نویسیم اگر:

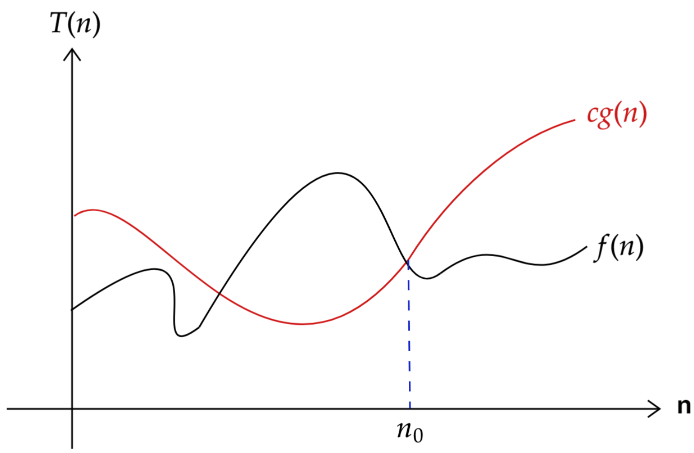
$$\exists c, n_0 \quad \text{s.t.} \quad \forall n > n_0 \quad 0 \leq f(n) \leq cg(n)$$

که c عدد ثابت و مثبت حقیقی و n_0 عدد ثابت و مثبت طبیعی هستند. در واقع O یک حد بالا (Upper Bound) را مشخص می‌کند.
در شکل ۲ مشاهده می‌شود که از n_0 به بعد، رشد تابع f(n) همواره کمتر از cg(n) است.

- **تعریف ۲:** به صورت کلی کلاس همه‌ی توابعی که به صورت f(n)=O(g(n)) هستند به صورت زیر است:

$$O(g(n)) = \{f(n) : \exists c, n_0 \quad s.t. \quad \forall n \geq n_0 \qquad 0 \leq f(n) \leq cg(n)\}$$

با مقایسه تعریف اول و دوم، سوالی که مطرح می‌شود این است که چرا به جای $f(n) \in cg(n)$ تساوی می‌گذاریم؟ در جواب می‌توان گفت که چون تمامی اعضای این مجموعه یک مفهوم را منتقل می‌کنند پس می‌توان آن ها را یکی گرفت و رابطه تساوی را نوشت.



شکل ۲: نمودار رشد توابع $f(n) = O(g(n))$

مثال ۱: توابع f(n) و g(n) به صورت $f(n) = 2n^2 - 3n + 4$ و $g(n) = n^3$ هستند. آیا رابطه $f(n) = O(g(n))$ برقرار است؟
طبق تعریف باید دو ثابت c و n_0 پیدا کنیم که شرایط لحاظ شده را برآورده کند:

$$\exists c, n_0 \quad \forall n \geq n_0 \quad 0 \leq 2n^2 - 3n + 4 \leq cn^3$$

مثلا $\left.\begin{array}{r} 2n^2 < 2n^3 \leftarrow c = 2 \\ -3n + 4 < 0 \leftarrow n_0 = 2 \end{array}\right\}$

پس به ازای $n_0 = 2$ رابطه $f(n) = O(g(n))$ برقرار می‌شود. توجه کنید که اگر c کوچکتر شود، n_0 بزرگتری باید انتخاب شود.
سوال: آیا عکس رابطه بالا یعنی $g(n) = O(f(n))$ برقرار است؟
فرض کنید برقرار باشد، پس طبق تعریف باید ثابت‌های c و n_0 وجود داشته باشند که شرایط موجود در تعریف را برآورده کنند:

$$\exists c, n_0 \quad \forall n \geq n_0 \quad 0 \leq n^3 \leq c(2n^2 - 3n + 4)$$

دو طرف را بر n^3 تقسیم می‌کنیم: $1 \leq \underbrace{c(\underbrace{\frac{2}{n} - \frac{3}{n^2} + \frac{4}{n^3}}_{(n\to\infty)\to 0})}_{<1}$

پس طرف راست نامساوی بالا کمتر از ۱ می‌شود. پس رابطه $g(n) = O(f(n))$ برقرار نیست.

مثال۲: فرض کنید $f(n) = 10^9n^2, g(n) = n^2$ باشند، نشان دهید: $f(n) = O(g(n))$.

$$\exists c, n_0 \quad \forall n \geq n_0 \quad 0 \leq 10^9n^2 \leq cn^2$$

کافی است که $c = 10^9, n_0 = 1$ قرار دهیم.

سوال: آیا $g(n) = O(f(n))$ برقرار است؟

کافی است که c=1 قرار دهیم و رابطه برای هر n_0 انتخابی درست است.

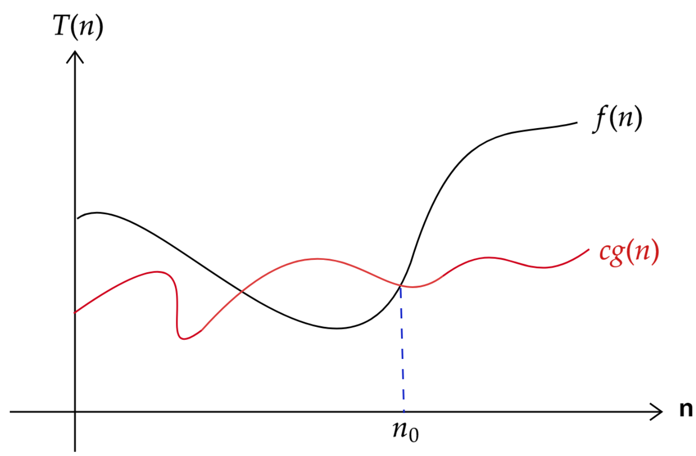
نکته:این مثال نشان می‌دهد که ضرایب ثابت تاثیری بر کلاس توابع پیچیدگی زمانی ندارند. در واقع نماد O به نوعی ثابت c را در خودش پنهان می‌کند و اهمیتی به آن نمی‌دهد. در مثال جدولی جلسه قبل، چنین هدفی را دنبال می‌کردیم که همانطور که دیدید نماد O آن را برایمان برآورده می‌کند.

۲. **نماد** Ω $(Big - Omega)$:

- **تعریف ۱:** فرض کنید f(n) و g(n) دو تابع باشند، می‌گوییم رشد تابع f(n) حداقل به اندازه تابع g(n) است و به صورت $f(n) = \Omega(g(n))$ می‌نویسیم، اگر:

$$\exists c, n_0 \quad \text{s.t.} \quad \forall n > n_0 \quad 0 \leq cg(n) \leq f(n)$$

که c عدد ثابت و مثبت حقیقی و n_0 عدد ثابت و مثبت طبیعی هستند. در واقع Ω یک حد پایین (Lower Bound) را مشخص می‌کند.



شکل ۳: نمودار رشد توابع: $f(n) = \Omega(g(n))$

- **تعریف ۲:** به طور کلی کلاس همه‌ی توابعی که به صورت $f(n) = \Omega(g(n))$ هستند، به صورت زیر است:

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

مثال: فرض کنید $f(n) = 10^9n^2, g(n) = n^2$ باشند، نشان دهید: $f(n) = \Omega(g(n)), g(n) = \Omega(f(n))$.

$$\exists c, n_0 \quad \forall n \geq n_0 \quad 0 \leq cn^2 \leq 10^9n^2$$

کافی است که $c = 10^9, n_0 = 1$ قرار دهیم تا هر دو معادله بدست آیند.