دانشکده‌ی علوم ریاضی

ساختمان‌داده‌ها ۲۴ شهریور ۹۲

جلسه‌ی ۱: مقدمه–مسأله و الگوریتم

مدرّس: دکتر شهرام خزائی *نگارنده: محمد امین شعبانی*

۱ مقدمه

امروزه انسان‌ها همواره با مسائل گوناگونی مواجه می‌شوند که برای حل آن‌ها از روش‌های مختلفی استفاده می‌کنند. بعضی از این روش‌ها از روند خاصی برخوردارند که به ما کمک می‌کند تا با طی کردن آن به جواب مسأله دست پیدا کنیم. همچنین با پیشرفت فناوری و افزایش حجم اطلاعات، برای ذخیره و پردازش اطلاعات نیاز به روش‌هایی داریم که در این درس به بررسی آن‌ها می‌پردازیم.

۲ الگوریتم چیست؟

الگوریتم[۱] روشی برای حل یک مسأله محاسباتی[۲] مشخص است که مجموعه‌ای از ورودی‌ها را به مجموعه‌ای از خروجی‌ها تبدیل می‌کند. به عبارتی الگوریتم ابزاری برای حل مسأله‌های محاسباتی می‌باشد. در بخش ۲.۲ به نمونه‌هایی از مسائل و سپس در بخش ۳.۲ الگوریتم‌هایی برای حل آن‌ها معرفی خواهیم کرد.

۱.۲ تحلیل الگوریتم

در تحلیل یک الگوریتم باید به دو موضوع توجه کنیم:

۱. صحت اجرای الگوریتم

۲. راندمان یا کارایی الگوریتم

قبل از استفاده از هر الگوریتم باید مطمئن شد که جواب بدست آمده صحیح می‌باشد. همچنین الگوریتم‌هایی که برای حل یک مسأله‌یکسان ابداع شده‌اند اغلب به طور برجسته‌ای در کارایی با هم تفاوت دارند، به همین دلیل بدست آوردن تابعی که تعداد محاسبات مورد نیاز الگوریتم برای ورودی‌هایی با اندازه‌های مختلف را تعیین کند ضروری می‌باشد. از این تابع به عنوان پیچیدگی زمانی[۳] الگوریتم نام می‌بریم.

[۱] algorithm
[۲] computational problems
[۳] time complexity

۲.۲ مثال‌هایی از مسأله‌های محاسباتی

یک مسأله محاسباتی با یک ورودی و یک خروجی که باید شرایط خاصی را ارضا کند مشخص می‌شود. در ادامه مثال‌هایی از مسائل محاسباتی می آید.

مسأله مرتب‌سازی از جمله مواردی که چه به صورت مستقیم و چه غیر مستقیم به وفور به آن بر می‌خوریم مسأله مرتب‌سازی[۴] می‌باشد. مرتب‌سازی همچنین زمینه‌ی مناسبی برای معرفی بسیاری از تکنیک‌های طراحی استاندارد و ابزار تحلیل فراهم می‌کند. مسأله مرتب‌سازی با ورودی‌ها و خروجی‌های به صورت زیر تعریف می‌شود:

- **ورودی:** دنباله‌ای از اعداد مانند $\langle a_۱, a_۲, \ldots, a_n\rangle$
- **خروجی:** یک جایگشت $\langle a'_۱, a'_۲, \ldots, a'_n\rangle$ از دنباله‌ی ورودی به طوری که $a'_i \leq a'_j$ به ازای هر $۱ \leq i < j \leq n$.

نکته ۱ *در عمل ممکن است دنباله داده شده به جای دنباله‌ای از اعداد، دنباله‌ای از رکوردها باشند و هدف مرتب کردن رکوردها برحسب یک مؤلّفه خاص باشد. یادآوری می‌شود که هر رکورد دارای تعدادی مؤلّفه (یا کلید) است. مقدار مؤلّفه k رکورد x با $x.k$ یا $k(x)$ نشان داده می‌شود. لذا مسأله مرتب‌سازی رکوردها را می‌توان به صورت زیر در نظر گرفت:*

- **ورودی:** *دنباله‌ای از رکوردهای $\langle x_۱, x_۲, \ldots, x_n\rangle$ و کلیدهای key آن‌ها*
- **خروجی:** *یک جایگشت $\langle x'_۱, x'_۲, \ldots, x'_n\rangle$ از رکوردهای ورودی به طوری که $x'_i.key \leq x'_j.key$ به ازای هر $۱ \leq i < j \leq n$.*

مسأله محاسبه حاصل‌ضرب اعداد هرچند بدست آوردن حاصل‌ضرب دو عدد برای اعداد کوچک ساده و سریع است، برای محاسبه حاصل‌ضرب دو عدد با تعداد رقم‌های خیلی زیاد به روش های بهتری نیازمندیم. به عنوان مثال در رمزنگاری نیاز به محاسبه حاصل‌ضرب اعداد چندصد رقمی است. مسأله محاسبه حاصل‌ضرب اعداد با ورودی‌ها و خروجی‌های زیر تعریف می‌شود:

- **ورودی:** دو عدد n رقمی x و y
- **خروجی:** مقدار حاصل‌ضرب xy

مسأله محاسبه حاصل‌ضرب ماتریس‌ها مسأله محاسبه حاصل‌ضرب ماتریس‌های مربعی با ورودی‌ها و خروجی‌های زیر تعریف می‌شود:

- **ورودی:** دو ماتریس X و Y با ابعاد عدد $n \times n$
- **خروجی:** ماتریس حاصل‌ضرب XY

مسأله سه‌مجموع فرض کنید n میله با طول‌های مختلف داده شده است. می‌خواهیم برای t داده شده با استفاده از اتصال سه تا از این میله‌ها میله‌ای به طول t بسازیم. به عبارتی فرض کنید که n عدد داده شده است. آیا سه تا از آن‌ها وجود دارند که جمع آن‌ها برابر ثابت t شود؟ این مسأله که سه‌مجموع نامیده می‌شود به صورت زیر تعریف می‌شود:

- **ورودی:** دنباله‌ای از اعداد مانند $\langle a_۱, a_۲, \ldots, a_n\rangle$ و عدد t
- **خروجی:** در صورت وجود سه‌تایی a_i, a_j, a_k به صورتی که $a_i + a_j + a_k = t$ باشد

[۴] sorting problem

اندازه و نمونه مسأله. در عمل مسأله‌های محاسباتی برای یک ورودی خاص از مسأله که نمونه[۵] نامیده می‌شود مطرح می‌شوند. مثلا دنباله $\langle -10, 22, -12, 60, 3, 20, 18\rangle$ یک نمونه از مسأله مرتب‌سازی است. به طور دقیق‌تر هر مسأله محاسباتی مجموعه‌ای از نمونه‌های مختلف مسأله است. هر نمونه دارای یک اندازه[۶] است. منظور از اندازه یک نمونه از مسأله معمولا تعداد بیت‌های لازم برای نمایش ورودی مسأله است. اما در این درس ما اندازه مسأله را با n نمایش می‌دهیم که ممکن است یک ضریب ثابت با حداکثر تعداد بیت‌های لازم برای نمایش ورودی مسأله تفاوت داشته باشد. همانطور که در جلسات آینده خواهیم دید، ابزاری که برای تحلیل زمان اجرای الگوریتم‌ها استفاده خواهیم کرد، چشم‌پوشی از ضرایب ثابت را در خود لحاظ می‌کند. به عنوان مثال اندازه مسأله در هر چهار مسأله ذکر شده n است در حالی که تعداد بیت‌های لازم برای نمایش ورودی آنها به‌ترتیب nw، $2nw$، $2n^2w$ و $(n+1)w$ است که فرض کرده‌ایم هر رقم یا عدد را با یک کلمه w-بیتی نمایش داده می‌شود.

۳.۲ مثال‌هایی از الگوریتم‌ها

در مثال‌های ذکر شده در بخش قبلی می‌توانیم با طراحی الگوریتم‌های مختلف به جواب مسأله دست پیدا کنیم. همانطور که قبلا ذکر شد ممکن است برای یک مسأله روش‌های مختلفی وجود داشته باشد که با توجه به شرایط و نیازها یکی از آن‌ها را انتخاب می‌کنیم.

الگوریتم ۱ (مسأله مرتب‌سازی) *الگوریتم مرتب‌سازی درجی[۷] یک روش بدیهی برای حل مسأله مرتب سازی است که به همان صورتی عمل می‌کند که اکثر مردم دسته‌ای از کارت‌های بازی را مرتب می‌کنند. به این صورت که اولین کارت را در دست خود نگه داشته و بقیه کارت‌ها را بر روی زمین قرار می‌دهیم. در هر مرحله یک کارت از زمین برداشته می‌شود و درجای مناسب خود در برگ‌های مرتبی که در دست قرار دارند درج می‌شود. برای پیدا کردن محل مناسب کارت جدید، آن را به ترتیب با کارت‌های مرتبی که در دست قرار دارند مقایسه کرده تا مکان صحیح پیدا شود. در زیر شبه کدی از این الگوریتم نشان داده شده است:*

Algorithm 1 InsertionSort

```
function InsertionSort(array A[1 : n])
    for j = 2 to n do
        key ← A[j]
        i ← j − 1
        while i > 0 & A[i] > key do
            A[i + 1] ← A[i]
            i ← i − 1
        A[i + 1] ← key
```

هر عملیاتی (مانند مقداردهی، جایگذاری، خواندن، کپی، مقایسه و ..) نیازمند یک زمان ثابت مخصوص به خود برای اجرا است. همانگونه که در جلسه آینده خواهیم دید، می‌توان نشان داد که در اینصورت مرتب کردن یک دنباله به طول n با استفاده از الگوریتم مرتب‌سازی درجی نیازمند زمان $an^2 + bn + c$ است که a و b و c ثابت‌هایی هستند که به زمان اجرای دستورهای مختلف بستگی دارند. چنین تابعی نسبت به n از درجه ۲ است که ما آن را با نماد $\Theta(n^2)$ نشان می‌دهیم. در جلسات بعد با مفهوم این نماد و الگوریتم‌های مرتب‌سازی دیگری با زمان اجرای $\Theta(n \log n)$ که از کارایی بیشتری برخوردار هستند آشنا خواهیم شد.

[۵] instance
[۶] size
[۷] insertion sort

الگوریتم ۲ (مسأله حاصل‌ضرب) *در مثال دوم می‌توان از روشی که در دوران کودکی برای ضرب دو عدد آموخته‌ایم استفاده کرد. مثال زیر روند محاسبه حاصل‌ضرب دو عدد ۵۴۶۵ و ۱۴۲۳ را نشان می‌دهد:*

$$\begin{array}{rr} & ۵۴۶۵ \\ & ۱۴۲۳ \\ \hline & ۱۶۳۹۵ \\ + & ۱۰۹۳۰۰ \\ + & ۲۱۸۶۰۰۰ \\ + & ۵۴۶۵۰۰۰ \\ \hline & ۷۷۷۶۶۹۵ \end{array}$$

پیچیدگی زمانی الگوریتم معمولی حاصل‌ضرب برای عددهای n رقمی از $\Theta(n^۲)$ است. روش جالب دیگری نیز برای بدست آوردن حاصل‌ضرب دو عدد به صورت بازگشتی وجود دارد؛ به این صورت که دو عدد n رقمی x و y را به شکل $x = ۱۰^{n/۲}a + b$ و $y = ۱۰^{n/۲}c + d$ نوشته و سپس حاصل‌ضرب آنها را محاسبه می‌کنیم:

$$x = \boxed{a \mid b} \Rightarrow x = ۱۰^{n/۲}a + b$$
$$y = \boxed{c \mid d} \Rightarrow y = ۱۰^{n/۲}c + d$$

$$\Rightarrow xy = ۱۰^{n}ac + ۱۰^{n/۲}(ad + bc) + bd$$

متأسفانه پیچیدگی زمانی این روش باز هم $\Theta(n^۲)$ است. اثبات این موضوع را در جلسات بعد با استفاده از قضیه اصلی خواهیم دید. برای اینکه پیچیدگی زمانی الگوریتم تقسیم و حل را کم کنیم باید تعداد ضرب‌ها را از ۴ کمتر کنیم. برای این کار می‌توانیم به جای محاسبه $ad + bc$ با استفاده از دو حاصل‌ضرب ad و bc، علاوه‌بر محاسبه حاصل‌ضرب‌های $P_۱ = ac$ و $P_۲ = bd$، حاصل‌ضرب $P_۳ = (a + b)(c + d)$ را نیز بدست آوریم. سپس حاصل‌ضرب‌های $P_۱$ و $P_۲$ را از $P_۳$ کم کنیم تا به عبارت $ad + bc$ برسیم. این ایده اولین بار توسط گوس برای محاسبه حاصل‌ضرب اعداد مختلط مطرح شد و تعداد حاصل‌ضرب‌ها را از ۴ به ۳ کاهش می‌دهد. پیچیدگی الگوریتم حاصل $\Theta(n^{\log_۲ ۳})$ خواهد بود. آیا می‌توانید روش تقسیم و حل را به محاسبه حاصل‌ضرب ماتریس‌ها تعمیم دهید؟

الگوریتم ۳ (مسأله سه‌مجموع) *برای مسأله سه‌مجموع ساده‌ترین راهی که وجود دارد بررسی تمام سه‌تایی‌های موجود می‌باشد. می‌توانیم به ازای همه‌ی سه‌تایی‌های موجود و متمایز i, j, k حاصل جمع $a_i + a_j + a_k$ را حساب کرده و در صورت صفر بودن، آن سه عدد را به عنوان خروجی مسأله بازگردانیم. در غیر اینصورت الگوریتم یک مقدار قراردادی را به عنوان نبود هیچ سه‌تایی در خروجی نشان می‌دهد. در این روش برای بدست آوردن راه حل $\binom{n}{۳}$ حالت برای j, i و k داریم که در نتیجه برای بررسی همه‌ی حالات به زمانی معادل $\Theta(n^۳)$ نیاز خواهیم داشت. این مسأله را می‌توان با کمک گرفتن از الگوریتم مرتب‌سازی و الگوریتم جست‌وجوی دودویی[۸] در مرتبه زمانی $\Theta(n^۲ \log n)$ حل کرد. وجود الگوریتمی با مرتبه پایین‌تر و یا اثبات عدم وجود چنین الگوریتمی برای این مسأله تا این لحظه بدون حل می‌باشد.*

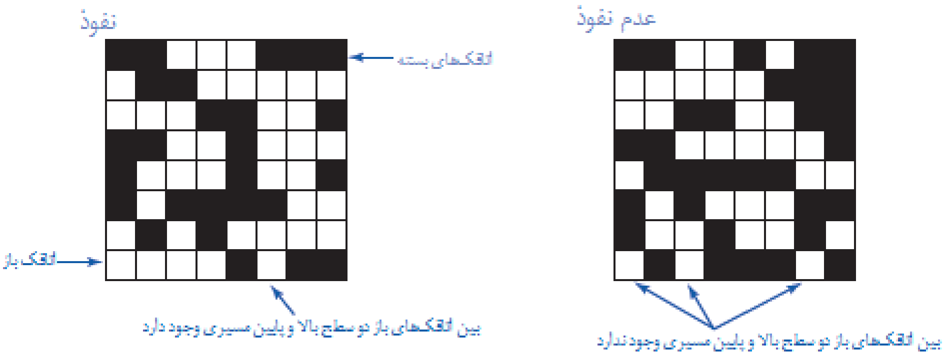
۳ ساختمان‌داده

ساختمان‌داده[۹] روشی برای ذخیره و مدیریت داده‌ها به منظور تسهیل در دسترسی و تغییر داده‌های مورد نظر می‌باشد. به عبارتی، ساختمان‌داده‌ها مدلی منطقی یا ریاضی برای سامان‌دهی داده‌ها به یک شکل خاص می‌باشند. استفاده از

[۸] binary search
[۹] data structure

ساختمان‌داده‌ها و الگوریتم‌ها می‌تواند علاوه بر بهبود سرعت محاسبه در کاهش حافظه‌ی مورد نیاز نیز مؤثر باشد. برای مثال از کاربردهای ساختمان‌داده‌ها می‌توانیم به سوال زیر اشاره کنیم:

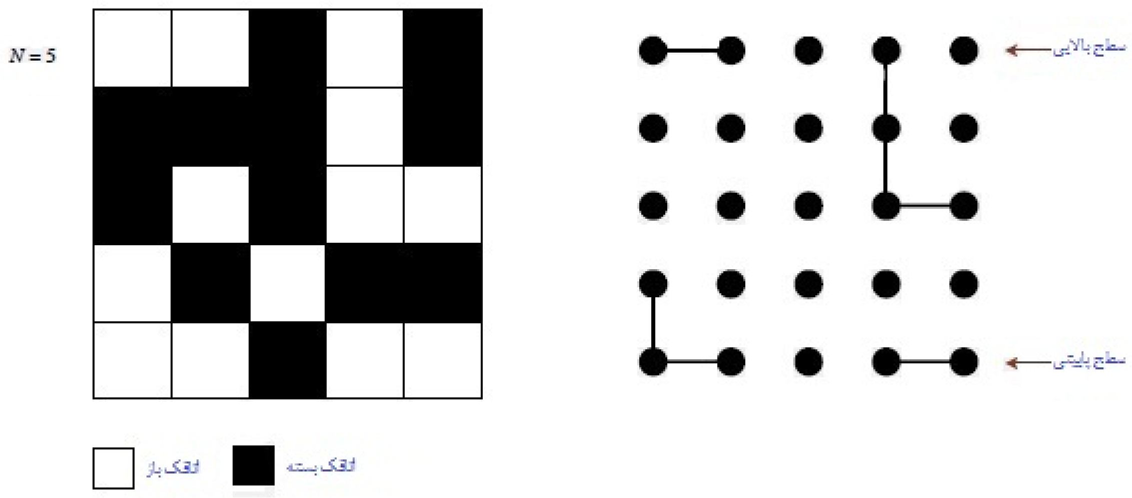
سؤال ۱ (تراوش) *فرض کنید مایعی در بالای جسمی منفذدار ریخته شده است. هر منفذ دارای چهار دیواره است و دو منفذ مجاور دارای یک دیواره مشترک هستند. بعضی از منافذ باز و بعضی بسته می‌باشند. تراوش مایع فقط به درون منافذ باز و از طریق دیواره‌ها صورت می‌گیرید. بدیهی است که تراوش مایع از سطح بالایی به سطح پایینی تنها در صورتی امکان‌پذیر می‌باشد که مسیر بازی از یک منفذ بالایی به یک منفذ پایینی وجود داشته باشد. اگر احتمال باز بودن هر منفذ p باشد، احتمال نفوذ مایع از سطح بالایی به سطح پایینی (وقتی ابعاد جسم خیلی زیاد است) چقدر است؟*



شکل ۱: وجود و عدم وجود مسیر بین دو سطح

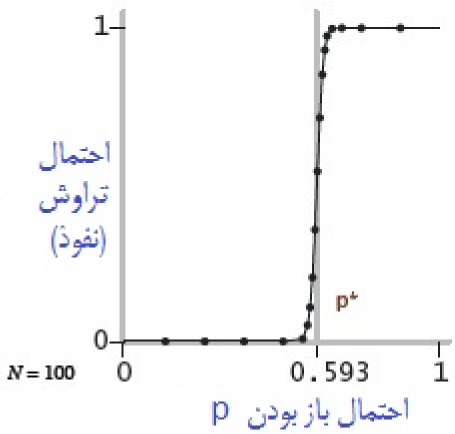
این سوال که در علم فیزیک در نظریه تراوش[10] مطرح می‌شود، در ریاضیات به صورت یک گراف با n^2 رأس مدل می‌شود. هر رأس بیانگر یک منفذ است و دو رأس توسط یک یال به هم متصل می‌شوند اگر هر دو منفذ متناظر با آن باز باشند و یک دیواره مشترک داشته باشند. حال مسأله امکان تراوش از سطح بالا به سطح پایین به تعیین وجود مسیری از مجموعه رئوس بالایی به مجموعه رئوس پایینی گراف تبدیل می‌شود.

[10] percolation theory



شکل ۲: تبدیل مسأله به مجموعه‌ای از راس‌ها و یال‌ها

متاسفانه هنوز راه حلی تئوری برای حل این مساله وجود ندارد. اما با استفاده از شبیه‌سازی‌های کامپیوتری می‌توان مسأله را حل کرد. نتایج شبیه‌سازی نشان می‌دهد که با افزایش ρ احتمال وجود مسیر از مجموعه رئوس بالایی به مجموعه رئوس پایینی گراف افزایش پیدا می‌کند. همچنین نقطه‌ی بحرانی ρ_c وجود دارد به صورتی که در بازه‌ای نزدیک ρ_c احتمال وجود چنین مسیری از نزدیکی‌های صفر تا نزدیکی‌های یک به شدت افزایش پیدا می‌کند. برای مثال این نقطه‌ی بحرانی برای $n = ۱۰۰$ چیزی در حدود "۰٫۵۹۳" بدست آمده است.



شکل ۳: احتمال وجود مسیر و نفوذ مواد به ازای $n = ۱۰۰$

در مسأله بالا برای مقادیر بزرگ جدول، ذخیره اطلاعات و پیدا کردن مسیرها به صورت معمولی سخت می‌باشد ولی با استفاده از ساختمان‌داده‌ها می‌توانیم چنین عملی را با هزینه و وقت بسیار کمتری انجام دهیم. برای چنین مسأله‌ای نیاز به روشی برای نگهداری رئوس و یال‌های گراف داریم به صورتی که بررسی و جست و جوی گراف به راحتی انجام شود. برای نگه‌داری تمامی اطلاعات یال‌ها و راس‌ها نیاز به ذخیره اطلاعات از مرتبه $n^۴$ خواهیم داشت که به ازای nهای

بزرگ حافظه‌ی زیادی را می‌طلبد ولی با استفاده از داده ساختارها می‌توانیم در هر لحظه تنها اطلاعات مجموعه‌های به هم متصل را نگه‌داری کنیم که به میزان حافظه‌ی کمتری نیاز دارد. همچنین بررسی وجود چنین مسیری نیز نیاز به روشی دارد که به آن ساختمان‌داده مجموعه‌های مجزا[۱۱] گفته می‌شود (که احتمالاً در این درس به آن نخواهیم پرداخت). در این روش منفذهایی که به هم متصل هستند را به صورت مجموعه‌هایی جدا از هم در نظر می‌گیریم و سپس مجموعه‌هایی که به هم متصل هستند را یکی می‌کنیم. در آخر برای فهمیدن وجود مسیری بین سطح بالایی و پایینی کافیست بررسی کنیم که آیا این رئوس در یک مجموعه قرار دارند یا خیر. البته همانطور که پیداست در اینجا فقط به وجود یا عدم وجود چنین مسیری پاسخ می‌دهیم که در صورت وجود، برای نمایش دادن چنین مسیری نیز نیاز به روشی برای نگهداری و پیمایش مسیرها خواهیم داشت. امکان شبیه‌سازی مسأله تراوش برای مقادیر بزرگ n فقط با استفاده از چنین داده‌ساختارهایی امکان‌پذیر است. در این درس با چندین ساختمان‌داده متفاوت با قابلیت‌های مختلف از قبیل پشته[۱۲]، صف[۱۳]، هرم[۱۴] (یا صف اولویت[۱۵])، لیست‌های زنجیره‌ای[۱۶]، تابع درهمساز[۱۷]، درخت‌دودویی‌جستجو[۱۸] و ... آشنا خواهیم شد.

[۱۱] disjoint-set
[۱۲] stack
[۱۳] queue
[۱۴] heap
[۱۵] priority queue
[۱۶] linked-list
[۱۷] hash function
[۱۸] binary search tree