بخش اول:

آنالیز زمانی

مدرسان شریف

فصل اول

«پیچیدگی زمانی الگوریتم‌ها»

مقدمه

می‌توان گفت بخش گسترده‌ای از مباحث موجود در علم کامپیوتر بر پایهٔ مطالعه و بررسی الگوریتم‌ها بنا شده است. در نوشتار پیش رو سعی بر این است که به مطالعه، بررسی و مقایسه روش‌های طراحی و آنالیز الگوریتم‌ها پرداخته شود. در زیر تعریف الگوریتم بیان شده است (واژه الگوریتم از نام ریاضی‌دان مشهور ایرانی «الخوارزمی» اقتباس شده است).

❖ **تعریف:** الگوریتم مجموعه‌ای از دستورالعمل‌هایی است که اگر با ترتیب خاصی اجرا شود، هدف مشخصی را برآورده می‌کند. یک الگوریتم، دارای پنج خصوصیت زیر می‌باشد:

۱) **ورودی:** یک الگوریتم می‌تواند چندین کمیت را به عنوان ورودی داشته باشد و یا هیچ ورودی نداشته باشد (یعنی از دنیای خارج هیچ ورودی دریافت نکند).

۲) **خروجی:** یک الگوریتم باید حداقل یک کمیت به عنوان خروجی داشته باشد.

۳) **عدم ابهام (قطعیت):** هر دستور باید بدون ابهام باشد.

۴) **خاتمه‌پذیری:** هر الگوریتم باید پس از طی مراحل متناهی به پایان برسد.

۵) **قابل اجرا بودن:** هر دستورالعمل موجود در یک الگوریتم باید قابل اجرا باشد (بتوان آن را با استفاده از کاغذ و قلم و به صورت دستی اجرا کرد).

نکته۱: تفاوت اصلی الگوریتم با برنامه در این است که الگوریتم باید پایان‌پذیر باشد، اما برنامه ممکن است پایان‌پذیر نباشد.

درسنامه (۱): نمادهای مجانبی

به‌دست آوردن مرتبه اجرایی الگوریتم

در یک تعریف کلی می‌توان یک الگوریتم را ایده‌ای برای حل یک مسأله دانست، اما برای حل یک مسأله خاص ایده‌های مختلفی وجود دارد. به عنوان مثال برای مسأله مرتب‌سازی (Sorting Problem) الگوریتم‌های متفاوتی مانند مرتب‌سازی سریع (Quick Sort)، مرتب‌سازی درجی (Insertion Sort)، مرتب‌سازی حبابی (Bubble Sort)، مرتب‌سازی هرمی (Heap Sort)، مرتب‌سازی درختی (Tree Sort)، مرتب‌سازی ادغامی (Merge Sort)، مرتب‌سازی مبنایی (Radix Sort)، مرتب‌سازی پوسته‌ای (Shell Sort)، مرتب‌سازی شمارشی (Counting Sort)، مرتب‌سازی پیمانه‌ای (Bucket Sort)، مرتب‌سازی پن کیک (Panckake Sort) و ... وجود دارد. حال اگر بخواهیم در یک مسأله که زمان در آن نقش تعیین‌کننده‌ای دارد از یک الگوریتم مرتب‌سازی استفاده کنیم، منطقی است که از بین الگوریتم‌های موجود سریع‌ترین را انتخاب نماییم (البته در بسیاری از موارد عملی، مقدار حافظه مورد نیاز الگوریتم نیز اهمیت خاص خود را دارد). در نتیجه برای مقایسه الگوریتم‌های موجود باید معیارهایی داشته باشیم که مستقل از سخت‌افزار یا نرم‌افزار مورد استفاده، بتواند تعیین کند که در کل کدام الگوریتم سریع‌تر می‌باشد، هدف این بخش به‌دست آوردن ابزاری برای انجام این کار است.

لازم به ذکر است که الگوریتم‌های مختلف مقادیر مختلفی از حافظه را نیاز دارند و میزان حافظه مورد نیاز نیز یک عامل مهم در کارایی الگوریتم‌ها می‌باشد. اما در مقابله بین حافظه و زمان، عامل زمانی تأثیر بیشتری دارد و از اهمیت بالاتری برخوردار است (هر جا که دو الگوریتم مقایسه می‌شوند در حقیقت از نظر سرعت آنها را مقایسه می‌نماییم، مگر این که صراحتاً چیزی غیر از آن ذکر شود).

تابع پیچیدگی و گام محاسباتی:

در این قسمت، هدف تعیین معیارهایی است که بتوان با استفاده از آن‌ها، الگوریتم‌ها را مستقل از نرم‌افزار و سخت‌افزار از نظر زمانی تحلیل کرد.

❖ **تعریف:** کوچکترین واحد زمانی برای اجرای الگوریتم‌ها را یک **step** یا یک **گام محاسباتی** می‌گوییم.

در زیر انواع عباراتی که در یک الگوریتم می‌تواند وجود داشته باشد، به همراه تعداد گام‌های لازم برای اجرای آنها بیان شده است:

۱) **Comments**: توضیحاتی می‌باشند که برای فهم بهتر الگوریتم اضافه شده است و هیچ گام زمانی ندارد بنابراین step لازم برای اجرای آنها صفر است.

۲) **Declarations**: عبارات تعیین نوع (type) می‌باشند (نظیر int , char ،const ، . . .). تعداد stepهای لازم برای اجرای این دستورات نیز صفر است.

۳) **Assignments**: این دستورات یک مقدار را به یک متغیر نسبت می‌دهند و شکل کلی آنها به صورت $y = f$ می‌باشد که f می‌تواند یک تابع، یک روال و یا یک عبارت محاسباتی باشد، بنابراین تعداد step های آن به f بستگی دارد.

۴) **Iteration statements**: این دستورات شامل حلقه‌های تکرار مانند حلقه‌های زیر است:

1) for i =< expr1 > to < expr2 > do{body}
2) for j =< expr1 > down to < expr2 > do{body}
3) while < expr1 > do{body}
4) until < expr1 > repeat{body}
5) do {body} while < expr >

تعداد step های لازم برای هر یک از این عبارات به تعداد دفعات تکرار حلقه‌ها و گام‌های بدنه بستگی دارد.

۵) **فراخوانی روال و تابع (Call)**: دستوری که شامل یک فراخوانی است، گامی را لازم ندارد، اما در عوض باید تعداد گام‌های مورد نیاز برای تابع یا روال فراخوانی شده محاسبه گردد.

❖ **تعریف تابع پیچیدگی زمانی (Complexity Function)**: برای یک الگوریتم تابعی مانند $f(n)$ است که: $f : N \longrightarrow R \geq 0$

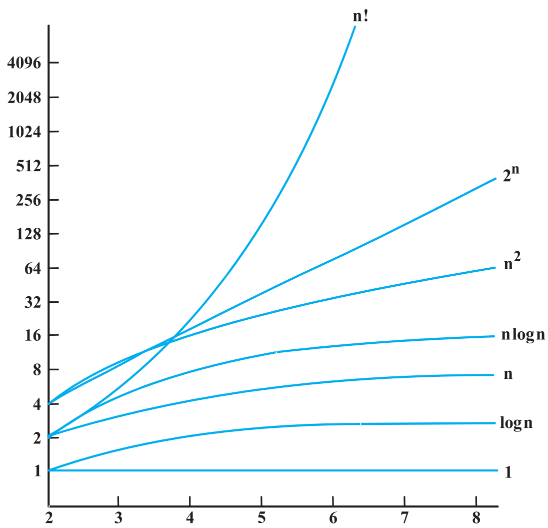
که در آن n اندازهٔ ورودی و $f(n)$ برابر با تعداد گام‌های زمانی مورد نیاز الگوریتم برحسب n می‌باشد.

به همین طریق می‌توان تابع پیچیدگی حافظهٔ یک الگوریتم را یک تابع مانند $g(n)$ در نظر گرفت که n اندازه ورودی است و $g(n)$ نشان‌دهنده مقدار حافظه‌ای است که الگوریتم مصرف می‌کند.

در حالت کلی، مقایسه دو الگوریتم به ازای ورودی‌های کوچک چندان معنادار نیست و مقایسه به ازای ورودی‌های به اندازه کافی بزرگ (در حد بی‌نهایت) انجام می‌پذیرد. دلیل این نوع مقایسه را می‌توان در این دانست که تفاوت توابع پیچیدگی به ازای ورودی‌های بزرگ، تعیین‌کننده و حیاتی است. به عنوان مثال، در جدول زیر شش تابع پیچیدگی به ازای چند ورودی مختلف مقایسه شده‌اند.

2^n	n^3	n^2	$n\lg n$	n	$\lg n$	اندازه ورودی
2.0	1.0	1.0	0.0	1.0	0.0	1
4.0	8.0	4.0	2.0	2.0	1.0	2
32.0	125.5	25.0	11.6	5.0	2.3	5
1024.0	1000.0	100.0	33.2	10.0	3.3	10
32768.0	3375.0	225.0	58.6	15.0	3.9	15
1048576.0	8000.0	400.0	86.4	20.0	4.3	20
1073741824.0	27000.0	900.0	147.2	30.0	4.9	30
1099511627776.0	64000.0	1600.0	212.9	40.0	5.3	40
1125899906842620.0	125000.0	2500.0	282.2	50.0	5.6	50
1152921504606850000.0	216000.0	3600.0	354.4	60.0	5.9	60
1180591620717410000000.0	343000.0	4900.0	429.0	70.0	6.1	70
1208925819614630000000000.0	512000.0	6400.0	505.8	80.0	6.3	80
1237940039285380000000000000.0	729000.0	8100.0	584.3	90.0	6.5	90
1267650600228230000000000000000.0	1000000.0	10000.0	664.4	100.0	6.6	100

همان‌گونه که مشخص است با بزرگ شدن اندازه ورودی تفاوت بین توابع بیشتر می‌گردد و بنابراین به ازای ورودی‌های بزرگ، انتخاب الگوریتم مناسب برای یک مسأله خاص از اهمیت به‌سزایی برخوردار است. شکل زیر، نمودار برخی از توابع پیچیدگی را نشان می‌دهد:



دقت کنید که این شکل فقط به ازای ورودی‌های کوچک ترسیم گردیده، اما باز هم تفاوت رشد توابع مشخص است. حال اگر ورودی‌های بزرگ در نظر گرفته شوند، آنگاه حتی در صورتی که توابع با رشد کم مانند $n\log n$ در ضرایب بزرگی نیز ضرب شوند، باز هم رشد آن‌ها کمتر از توابعی مانند 2^n یا $n!$ خواهد بود.
اگر فرض کنیم توابع بررسی شده در شکل فوق، مربوط به الگوریتم‌های موجود برای یک مسأله خاص باشند و روی یک کامپیوتر سریع اجرا شوند، به طوری که این کامپیوتر در هر ثانیه یک میلیارد عملیات پایه مربوط به مسأله مورد بررسی را انجام دهد، در این صورت زمان مورد نیاز برای اجرای الگوریتم‌های مختلف این مسأله روی کامپیوتر بیان شده، به صورت زیر خواهد بود:

<table>
<tr><th>اندازه ورودی</th><th colspan="6">تابع پیچیدگی</th></tr>
<tr><th>n</th><th>Log n</th><th>n</th><th>n Log n</th><th>n^2</th><th>2^n</th><th>n!</th></tr>
<tr><td>10</td><td>$3\times10^{-9}s$</td><td>$10^{-8}s$</td><td>$3\times10^{-8}s$</td><td>$10^{-7}s$</td><td>$10^{-6}s$</td><td>$3\times10^{-3}s$</td></tr>
<tr><td>10^2</td><td>$7\times10^{-9}s$</td><td>$10^{-7}s$</td><td>$7\times10^{-7}s$</td><td>$10^{-5}s$</td><td>بیش از صد میلیارد قرن</td><td rowspan="5">بیش از صد هزار میلیارد قرن</td></tr>
<tr><td>10^3</td><td>$1.0\times10^{-8}s$</td><td>$10^{-6}s$</td><td>$1\times10^{-5}s$</td><td>$10^{-3}s$</td><td rowspan="4">بیش از صد هزار میلیارد قرن</td></tr>
<tr><td>10^4</td><td>$1.3\times10^{-8}s$</td><td>$10^{-5}s$</td><td>$1\times10^{-4}s$</td><td>$10^{-1}s$</td></tr>
<tr><td>10^5</td><td>$1.7\times10^{-8}s$</td><td>$10^{-4}s$</td><td>$2\times10^{-3}s$</td><td>10s</td></tr>
<tr><td>10^6</td><td>$2\times10^{-8}s$</td><td>$10^{-3}s$</td><td>$2\times10^{-2}s$</td><td>17 min</td></tr>
</table>

همان‌گونه که می‌بینید، مدت زمان مورد نیاز برای توابع نمایی و فاکتوریل حتی به ازای ورودی‌های نه چندان بزرگ چند هزار برابر عمر کره زمین می‌باشد! (عمر کره زمین برابر 4.54 میلیارد سال و عمر حیات حدود یک میلیارد سال تخمین زده می‌شود). اگر طی سال‌های آینده سرعت کامپیوترها چند هزار برابر نیز شود، باز هم توابع نمایی و فاکتوریل، توابعی مهار نشدنی می‌باشند و می‌توان گفت الگوریتم‌های چند جمله‌ای بیشترین سود را از پیشرفت در سرعت سخت‌افزار خواهند برد.

نماد O (O بزرگ یا Big O)

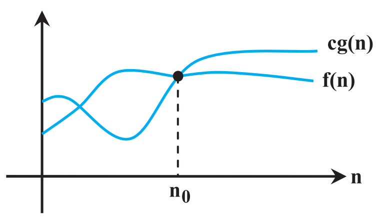
❖ **تعریف:** اگر $f(n)$ و $g(n)$ دو تابع پیچیدگی باشد، آنگاه می‌نویسیم $f(n)\in O(g(n))$ (می‌خوانیم $f(n)$ عضو O بزرگ $g(n)$ است) هرگاه داشته باشیم:

$$\exists c>0,\exists n_0\geq 0 \text{ s.t. } (\forall n\geq n_0 : f(n)\leq cg(n))\ ;\ c\in\mathbb{R}^+, n_0\in\mathbb{N}$$

در حقیقت $g(n)$ یک حد مجانبی بالا روی $f(n)$ می‌گذارد، به این معنی که برای n های به اندازه کافی بزرگ $f(n)$ همواره از $cg(n)$ کوچکتر است و بنابراین می‌توانیم بگوییم که در این حالت رشد $g(n)$ بیشتر از $f(n)$ خواهد بود و در نتیجه $f(n)$ سریع‌تر از $g(n)$ می‌باشد.

تذکر: در برخی منابع رابطه $f(n)\in O(g(n))$ به صورت $f(n)=O(g(n))$ استفاده می‌گردد.

نکته۲: در حقیقت $O(g(n))$ مجموعه تمام توابعی مانند $f(n)$ است که $f(n)\in O(g(n))$ شکل روبه‌رو رابطه بین $f(n)$, $g(n)$ را در حالتی که $f(n)\in O(g(n))$ باشد، نمایش می‌دهد.



اگر $f(n)=3n+14$ و $g(n)=n+\log n$، آنگاه: $f(n)\in O(g(n))$

اگر داشته باشیم $f(n)\in O(g(n))$ آنگاه یکی از دو حالت زیر اتفاق می‌افتد:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0 \quad \text{یا} \quad \lim_{n\to\infty}\frac{f(n)}{g(n)}=\alpha>0 \quad \alpha\in R$$

اگر $g(n)=2^n$ و $f(n)=n^3$ آنگاه: $f(n)\in O(g(n))$

اما $g(n)\notin O(f(n))$ زیرا داریم: $\lim_{n\to\infty}\frac{g(n)}{f(n)}=\infty$

نماد Ω (امگای بزرگ)

همان‌گونه که بیان شد، با استفاده از نماد "O" یک حد بالا برای رشد تابع مورد بررسی تعیین می‌گردد. با استفاده از نماد Ω، یک حد پایین روی رشد تابع مورد نظر قرار داده می‌شود.

❖ **تعریف:** می‌نویسیم $f(n)\in\Omega(g(n))$ (می‌خوانیم $f(n)$ امگای بزرگ $g(n)$ است)، هرگاه داشته باشیم:

$$\exists c>0,\ \exists n_0>0 \quad (\forall n\geq n_0 :\ f(n)\geq cg(n))\ ;\ c\in\mathbb{R}^+,\ n_0\in\mathbb{N}$$

فصل دوم

«آنالیز سرشکن»

مقدمه

در این فصل به بررسی میانگین زمان اجرای الگوریتم‌هایی می‌پردازیم که مرتبه اجرای آن‌ها برای ورودی‌های مختلف، متفاوت است. به داوطلبانی که آشنایی کافی با مطالب این کتاب ندارند و این مطالب را برای اولین بار مطالعه می‌کنند، توصیه می‌شود که این فصل را بعد از به اتمام رساندن کتاب مطالعه کنند.

درسنامه: تحلیل هزینه میانگین برای بدترین دنباله مقادیر

آنالیز سرشکن شده (Amortized Analysis)

ساختمان داده‌ای با n عنصر را در نظر بگیرید که هزینه عمل درج در آن در حالتی که تعداد عناصر آرایه به صورت توانی از 2 ($n = 2^m$) باشد، برابر $\theta(n)$ است در غیر این صورت، هزینه درج برابر $\theta(1)$ می‌باشد. برآورد هزینه عمل درج به اندازه $\theta(n)$ تا حدی بدبینانه و غیرواقعی است؛ زیرا به عنوان مثال در یک میلیون عمل درج کمتر از 20 بار چنین حالتی رخ می‌دهد و در بقیه حالات زمان $\theta(1)$ می‌باشد. همچنین نمی‌توان به راحتی زمان $\theta(1)$ را برای هزینه عمل درج در نظر گرفت، زیرا می‌دانیم در برخی حالات زمان بدتر است. برای تحلیل زمانی این گونه مسائل از آنالیز سرشکن شده استفاده می‌شود.

در حالت کلی، می‌توان گفت آنالیز سرشکن شده در مورد مسائلی مفید است که در آنها دنباله‌ای از عملیات روی یک ساختار داده انجام می‌پذیرد و در این ساختار زمانی که پرهزینه‌ترین عمل انجام می‌شود، اغلب حالت مسأله طوری تغییر می‌یابد که مجدداً، رخ دادن بدترین عملیات در آینده نزدیک اتفاق نمی‌افتد. به‌عنوان مثال، می‌توان نمودار زیر را برای نشان دادن تعداد عملیات لازم به‌ازای دنباله‌ای متوالی از عملیات درج در مثال بیان شده، در نظر گرفت:

تحلیل سرشکن شده، میانگین کارایی هر عمل در بدترین حالت را تضمین می‌کند و با آنالیز حالت میانگین و آنالیز آماری حالت میانگین متفاوت است:

۱ـ در آنالیز حالت میانگین (average case) هزینه کل به‌ازای تمام ورودی‌ها میانگین گرفته می‌شود.

۲ـ در آنالیز آماری هزینه کل روی تمام انتخاب‌های تصادفی ممکن انجام می‌پذیرد.

۳ـ در حالت سرشکن شده بدترین ورودی در نظر گرفته می‌شود و با توجه به زمان موردنیاز در دنباله‌ای از عملیات، بدترین زمان به ازای هر عمل به‌دست می‌آید.

سه روش اصلی زیر برای انجام آنالیز سرشکن شده استفاده می‌شود:

۱ـ روش آنالیز جمعی (aggregate analysis)

۲ـ روش حسابداری (accounting method)

۳ـ روش پتانسیل (potential method)

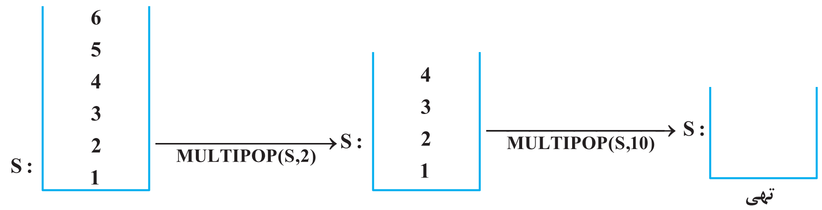
برای بررسی این سه روش دو مثال کلی براساس مطلب بیان شده در مرجع [1] در نظر گرفته شده است، یکی استفاده از ساختار پشته و دیگری شمارنده دودویی. سه روش بیان شده در ادامه به ترتیب مورد بحث قرار گرفته است.

آنالیز جمعی

ساختار داده پشته (stack) را در نظر بگیرید. می‌دانیم دو عمل اصلی POP و PUSH که به ترتیب برای حذف و اضافه نمودن یک عنصر به پشته استفاده می‌گردند، دارای زمان $O(1)$ می‌باشند. حال عمل جدید MULTI POP(S,k) را در نظر بگیرید. در این عمل، k عنصر بالای پشته حذف می‌شوند و اگر پشته کمتر از k عنصر داشته باشد، آنگاه تمام عناصر آن حذف می‌گردند.

```
MULTIPOP(S,K)
While not STACK – EMPTY(S) and k > 0
      POP (S)
      k = k-1
```

به عنوان مثال شکل زیر یک نمونه از انجام این عمل را نشان می‌دهد:



با توجه به الگوریتم بیان شده، هزینه انجام MULTIPOP برابر $\min(s,k)$ می‌باشد که s نشان‌دهنده تعداد عناصر موجود در پشته است. حال فرض کنید یک دنباله از n عمل POP، PUSH و MULTIPOP روی یک پشته انجام شده است. در تحلیل سرشکن جمعی هزینه $T(n)$ برای انجام یک دنباله از n عملیات در نظر گرفته شده و سپس هزینه $\frac{T(n)}{n}$ به ازای هر عمل در نظر گرفته می‌شود. حال اگر n عمل در پشته را در نظر بگیریم می‌توان گفت که تعداد کل POPهای انجام شده (هر MULTI POP را نیز می‌توانیم به عنوان دنباله‌ای از POPها در نظر بگیریم) برابر تعداد عملیات PUSH انجام شده می‌باشد. با توجه به این‌که در یک دنباله از n عملیات، هزینه عملیات PUSH از مرتبه $O(n)$ است، بنابراین هزینه عملیات POP و MULTIPOP نیز در مجموع $O(n)$ خواهد بود. در نتیجه هزینه کلی $T(n)=O(n)$ است و هزینه سرشکن شده جمعی برای هر عمل برابر $\frac{O(n)}{n}=O(1)$ می‌باشد.

مثال دوم به هزینه تغییر بیت‌ها در عمل افزایش در یک شمارنده دودویی می‌پردازد. فرض کنید یک عدد دودویی k رقمی در آرایه $A[0...k-1]$ ذخیره شده باشد، به طوری‌که $A[i]$ نشان‌دهنده بیت شماره i از عدد مورد نظر است، به عنوان مثال اگر آرایه به صورت زیر باشد:

A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
0	1	1	0	1	0

آنگاه آرایه A عدد 26 را نشان می‌دهد، بنابراین در کل عدد متناظر با آرایه A برابر $\sum_{i=0}^{k-1} A[i]\times 2^i$ است. در این حالت برای افزایش یک واحد مقدار ذخیره شده در آرایه از الگوریتم زیر استفاده می‌شود:

```
INCREMENT (A)
i = 0
While i < A . length and A[i] = =1
      A[i] = 0
      i = i +1
if i < A.length
      A[i]=1
```

اگر هدف، یافتن مرتبه زمانی بر اساس تعداد تغییرات بیت‌ها در n عمل افزایش (increment) در یک آرایه که با صفر مقداردهی اولیه شده است، باشد آنگاه با توجه به این که در بدترین حالت مرتبه زمانی الگوریتم فوق $\theta(k)$ می‌باشد، یک حد بالای $O(nk)$ را می‌توان برای انجام n عمل افزایش متوالی در نظر گرفت، اما این حد بالا، یک حد بالای غیر دقیق است.

به عنوان مثال تعداد تغییر بیت‌ها در 16 عمل افزایش متوالی به صورت زیر می‌باشد:

مقدار شمارنده پس از هر عمل افزایش	بیت‌های آرایه A: A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	تعداد تغییر بیت‌ها در هر مرحله	تعداد کل تغییر بیت‌ها
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	1	0	2	3
3	0	0	0	0	0	1	1	1	4
4	0	0	0	0	1	0	0	3	7
5	0	0	0	0	1	0	1	1	8
6	0	0	0	0	1	1	0	2	10
7	0	0	0	0	1	1	1	1	11
8	0	0	0	1	0	0	0	4	15
9	0	0	0	1	0	0	1	1	16
10	0	0	0	1	0	1	0	2	18
11	0	0	0	1	0	1	1	1	19
12	0	0	0	1	1	0	0	3	22
13	0	0	0	1	1	0	1	1	23
14	0	0	0	1	1	1	0	2	25
15	0	0	0	1	1	1	1	1	26
16	0	0	1	0	0	0	0	5	31

به عبارت دیگر می‌توان گفت:

بیت A[0] در هر افزایش تغییر می‌کند.

بیت A[1] در هر دو افزایش متوالی یک بار تغییر می‌کند.

بیت A[2] در هر چهار افزایش متوالی یک بار تغییر می‌کند.

بیت A[3] در هر هشت افزایش متوالی یک بار تغییر می‌کند.

⋮

بنابراین اگر n عمل افزایش متوالی را در نظر بگیریم، آنگاه تعداد بیت‌ها به صورت زیر خواهد بود:

بیت	تعداد تغییرات در n عمل افزایش متوالی
A[0]	n
A[1]	[n/2]
A[2]	[n/4]
A[3]	[n/8]
A[4]	[n/16]
⋮	⋮
A[i]	$[n/2^i]$
⋮	⋮

در نتیجه، اگر n عمل افزایش متوالی روی یک آرایه با مقدار اولیه صفر انجام شود (که منجر به ایجاد تغییر در $[\log_2^n]$ بیت ابتدایی می‌گردد)، تعداد کل تغییرات انجام شده روی بیت‌ها برابر است با:

$$\text{تعداد کل تغییر بیت‌ها در n عمل متوالی افزایش بر روی آرایه با مقدار اولیه صفر} = \sum_{i=0}^{[\log_2^n]} \left[\frac{n}{2^i}\right] \le \sum_{i=0}^{\infty} \frac{n}{2^i} = 2n$$

بنابراین، هزینه تغییر بیت‌ها در n عمل متوالی $O(n)$ می‌باشد و هزینه سرشکن شده جمعی برابر $\frac{O(n)}{n} = O(1)$ خواهد بود.

نکته: اگر عمل کاهش روی آرایه انجام شود، آنگاه هزینه n عمل می‌تواند $O(nk)$ باشد (زیرا کاهش می‌تواند باعث ایجاد رقم فرضی در بین تمام بیت‌ها گردد) و بنابراین هزینه سرشکن شده $O(k)$ خواهد بود.

مثال ۱: اگر n عمل افزایش متوالی روی یک شمارنده دودویی k بیتی با مقدار اولیه صفر انجام شود، آنگاه تعداد دفعات تغییر بیت شماره i کدام است؟

۱) 2^i ۲) $\frac{n}{2^i}$ ۳) $\log_2^i$ ۴) $\frac{n}{\log_2^i}$

پاسخ: گزینه «۲» حداکثر تعداد دفعات تغییر بیت i ام برابر با $\left\lceil \frac{n}{2^i} \right\rceil$ و حداقل آن $\left\lfloor \frac{n}{2^i} \right\rfloor$ است.

مثال ۲: در مثال قبل میانگین تعداد کل تغییرات بیت‌ها در هر عمل افزایش کدام است؟

۱) $O(1)$	۲) $O(n)$	۳) $O(\log_2^n)$	۴) $O(2n)$

پاسخ: گزینه «۱» تعداد کل تغییر بیت‌ها کمتر از $2n$ می‌باشد، بنابراین میانگین تغییر بیت‌ها $O(1)$ خواهد بود.

مثال ۳: اگر n عمل کاهش و افزایش روی یک شمارنده دودویی k بیتی انجام شود، میانگین تعداد تغییر بیت‌ها کدام است؟

۱) $O(n)$	۲) $O(nk)$	۳) $O(k)$	۴) $O(1)$

پاسخ: گزینه «۳» در این حالت تعداد کل تغییر بیت‌ها از مرتبه $O(nk)$ می‌باشد، بنابراین میانگین تعداد تغییر بیت‌ها عبارت است از:

$$\frac{O(nk)}{n} = O(k)$$

مثال ۴: یک ساختمان داده را در نظر بگیرید که n عمل متوالی روی آن انجام می‌پذیرد و هزینه عمل شماره i برابر i است، اگر i توانی از 2 باشد و در غیر این‌صورت هزینه $O(1)$ می‌باشد. در این حالت هزینه سرشکن شده جمعی به ازای هر عمل کدام است؟

۱) $O(3)$	۲) $O(1)$	۳) $O(n)$	۴) $O(2^n)$

پاسخ: گزینه «۱» اگر n عمل متوالی را در نظر بگیریم، آنگاه هزینه این اعمال به صورت زیر خواهد بود:

$$1+2+1+4+1+1+8+1+1+1+1+1+1+16+1+\dots$$
$$= \sum_{i=1}^{[\log_2^n]} 2^i + n - [\log_2^n] = 2^{[\log_2^n]+1} - 2 + n - [\log_2^n] < 2n + n = 3n$$

بنابراین هزینه کل از مرتبه $O(3n)$ می‌باشد و در نتیجه هزینه سرشکن شده جمعی برابر $\frac{O(3n)}{n} = O(3)$ خواهد بود.

روش حسابداری

در روش حسابداری ممکن است به اعمال مختلف، هزینه‌های سرشکن شده متفاوتی انتساب داده شود (برخلاف روش جمعی که در آن هزینه سرشکن شده برای تمام اعمال یکسان در نظر گرفته می‌شود). در روش حسابداری اگر هزینه سرشکن شده برای یک عمل بیش از هزینه واقعی آن باشد، آنگاه تفاوت این دو مقدار به عنوان موجودی در نظر گرفته می‌شود که این موجودی می‌تواند برای انجام عملیات بعدی که هزینه سرشکن شده آن‌ها کمتر از هزینه واقعی برآورد شده، در نظر گرفته شود.

در هر لحظه باید مجموع هزینه سرشکن شده برای انجام n عمل متوالی بزرگتر یا مساوی هزینه واقعی باشد، به عبارت دیگر اگر $\hat{c}_i$ هزینه سرشکن شده عمل iام و c_i هزینه واقعی آن باشد، آنگاه باید داشته باشیم:

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

در نتیجه میزان موجودی کل یعنی $\sum_{i=1}^{n}(\hat{c}_i - c_i)$ در هر لحظه، نامنفی می‌باشد. حال اگر در مثال پشته هزینه‌های سرشکن شده زیر به اعمال مربوطه انتساب داده شود، داریم:
$PUSH$: هزینه $= 2$ و $MULTIPOP$: هزینه $= 0$ و POP : هزینه $= 0$

آنگاه شرایط موجود در روش حسابداری برآورده می‌گردد. با توجه به این که هزینه واقعی PUSH برابر 1 است، بنابراین با هر عمل PUSH مقدار موجودی یک واحد افزایش می‌یابد و در نتیجه تعداد عناصر پشته نشان‌دهنده موجودی می‌باشد. با توجه به این که تعداد عناصر پشته همواره یک عدد نامنفی است، در نتیجه موجودی نیز همواره نامنفی خواهد بود (در حقیقت یک واحد افزایش موجودی در عمل PUSH به عنوان هزینه برای حذف عناصر مصرف می‌شوند). بنابراین هزینه سرشکن شده n عمل متوالی برابر $O(n)$ خواهد بود.

اگر در شمارنده دودویی هزینه سرشکن شده تغییر یک بیت از صفر به یک را 2 و هزینه تبدیل بیت از یک به صفر را برابر 0 در نظر بگیریم، آنگاه شرایط مورد نیاز در روش حسابداری برآورده می‌گردد، زیرا در هر لحظه تعداد یک‌های موجود در آرایه، نشان‌دهنده مقدار موجودی است و با توجه به این که همواره تعداد یک‌ها نامنفی است، بنابراین موجودی کل همواره بزرگتر یا مساوی صفر می‌باشد. در نتیجه، هزینه سرشکن شده n عمل متوالی $O(n)$ است و از این رو هزینه واقعی نیز $O(n)$ خواهد بود.

روش پتانسیل

در روش پتانسیل به هر حالت از ساختار داده یک پتانسیل نسبت داده می‌شود که میزان این پتانسیل نشان‌دهنده هزینه‌ای است که می‌تواند برای انجام اعمال بعدی استفاده شود، در این صورت اگر D_i را حالت ساختار داده پس از انجام i امین عمل در نظر بگیریم، آنگاه هزینه سرشکن شده به صورت زیر تعریف می‌گردد:

$$\hat{c}_i = \underbrace{c_i}_{\text{هزینه واقعی}} + \underbrace{\Phi(D_i)}_{\text{پتانسیل ساختار داده پس از انجام عمل ام}} - \underbrace{\Phi(D_{i-1})}_{\text{پتانسیل ساختار داده قبل از انجام عمل ام}}$$

بنابراین کل هزینه سرشکن شده برای انجام n عمل متوالی به صورت زیر به‌دست می‌آید:

هزینه سرشکن شده حاصل از انجام n عمل متوالی: $\sum_{i=1}^{n}\hat{c}_i = \sum_{i=1}^{n}[c_i + \Phi(D_i) - \Phi(D_{i-1})] = \sum_{i=1}^{n}c_i + \Phi(D_n) - \Phi(D_0)$

در نتیجه، برای این که هزینه سرشکن شده یک حد بالا برای هزینه کل باشد، باید شرط $\Phi(D_n) \ge \Phi(D_0)$ برقرار شود که برای سادگی می‌توان $\Phi(D_0) = 0$ را در نظر گرفت. بنابراین شرط $\Phi(D_i) \ge 0$ به عنوان شرط مورد نیاز برای تعریف تابع پتانسیل خواهد بود. حال اگر مثال پشته را در نظر بگیریم و پتانسیل پشته را تعداد عناصر موجود در پشته تعریف کنیم، آنگاه:

$\Phi(D_0) = 0 \quad , \quad \Phi(D_i) \ge 0$ (همواره تعداد عناصر پشته نامنفی است)

در این حالت هزینه سرشکن شده اعمال مورد بررسی به صورت زیر خواهد بود:

هزینه سرشکن شده عمل PUSH:

$$\hat{c} = \underbrace{1}_{\text{هزینه واقعی}} + \underbrace{\Phi(D_i) - \Phi(D_{i-i})}_{\text{تغییر پتانسیل}} = 1 + 1 = 2$$

هزینه سرشکن شده MULTIPOP:

اگر قرار دهیم $m=\min(s,k)$، آنگاه:

$$\hat{c} = m + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{تغییر پتانسیل}} = m + (-m) = 0$$

هزینه سرشکن شده عمل POP:

$$\hat{c} = 1 + \underbrace{(-1)}_{\text{تغییر پتانسیل}} = 0$$

بنابراین هزینه سرشکن شده انجام n عمل متوالی برابر $O(n)$ می‌باشد که یک حد بالا برای هزینه واقعی است.

در شمارنده دودویی مقدار پتانسیل را تعداد یک‌های موجود در آرایه پس از انجام عمل i ام در نظر می‌گیریم. اگر تعداد یک‌های موجود در آرایه پس از انجام عمل iام را b_i در نظر بگیریم، آنگاه در صورتی‌که در انجام عمل i ام، تعداد بیت‌هایی که از یک به صفر تغییر می‌کنند را t_i در نظر بگیریم، هزینه واقعی انجام عمل iام برابر $t_i + 1$ خواهد بود؛ زیرا یک بیت نیز از صفر به یک تغییر می‌کند. در این حالت تعداد یک‌های موجود در آرایه به صورت زیر خواهد بود:

$$b_i = \underset{\text{تعداد یک‌های قبل از عمل ام}}{\underset{\uparrow}{b_{i-1}}} - \underset{\text{تعداد تغییرات بیت‌ها از یک به صفر}}{\underset{\uparrow}{t_i}} + \underset{\text{یک تغییر بیت از صفر به یک}}{\underset{\uparrow}{1}}$$

بنابراین تغییر پتانسیل در هر حالت عبارت است از:

$$\Phi(D_i) - \Phi(D_{i-1}) = b_i - b_{i-1} = b_{i-1} + t_i + 1 - b_{i-1} = 1 - t_i$$

و در نتیجه هزینه سرشکن شده به صورت مقابل خواهد بود:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = (t_i + 1) + (1 - t_i) = 2$$

بنابراین هزینه سرشکن شده n عمل متوالی $O(n)$ است.

مثال ۵: **در روش پتانسیل باید کدام شرط در مورد تابع پتانسیل برقرار باشد؟**

۱) $\Phi(D_o) = \Phi(D_n)$ ۲) $\Phi(D_n) \ge \Phi(D_o)$ ۳) $\Phi(D_n) \le \Phi(D_o)$ ۴) $\Phi(D_n) \le \Phi(D_{n+1})$

☑ **پاسخ:** گزینه «۲» میزان پتانسیل همواره باید نامنفی باشد.

مثال ۶: **دستگاهی داریم که می‌تواند قطعات چوب را به دو قسمت مساوی تقسیم نماید. در صورتی که n قطعه چوب با طول‌های $\{1,2,3,...,n\}$ برحسب متر داشته باشیم و بخواهیم آن‌ها را با دستگاه مذکور برش دهیم، به‌طوری که طول همه چوب‌ها عددی فرد برحسب متر باشد، تعداد چوب‌هایی که پس از پایان یافتن تمام برش‌ها در اختیار داریم از چه مرتبه‌ای است؟**

۱) $\theta(n)$ ۲) $\theta(n\lg n)$ ۳) $\theta(n\lg\lg n)$ ۴) $\theta(n^2)$

☑ **پاسخ:** گزینه «۲» هر چوب به طول $c2^n$ (فرض می‌کنیم c عددی فرد باشد) به 2^n چوب با طول c قابل تقسیم شدن است. اگر n را برابر 2^m درنظر بگیریم، 1 چوب به 2^m تکه، 1 چوب به 2^{m-1} تکه، 2 چوب به 2^{m-2} تکه، ... ، 2^{i-1} چوب به 2^{m-i} تکه، ... و 2^{m-2} چوب به 2 تکه تقسیم می‌شوند و 2^{m-1} چوب تقسیم نخواهند شد.

به عنوان مثال، چوب‌هایی که به 2^k تکه تقسیم می‌شوند، طولشان برابر یکی از مقادیر $\{2^k, 3\times 2^k, 5\times 2^k, 7\times 2^k, \dots, (2^{m-k}-1)\times 2^k\}$ خواهد بود که دقیقاً k عامل 2 دارند. مجموع مقادیر از مرتبه $\theta(m2^m)$ است که با $\theta(n\lg n)$ برابر خواهد بود.

مثال ۷: تابع $\lg n(x)$ را به فرم زیر تعریف می‌کنیم. (منظور از $\mathbb{N}$ مجموعه اعداد طبیعی است.) در این صورت مقدار $m=\sum_{i=1}^{n}\lg n(i)$ از چه مرتبه‌ای خواهد بود؟

$$\lg n(x)=\begin{cases}\lg n(\frac{x}{2})+1 & x\in\mathbb{N}\\ 0 & x\notin\mathbb{N}\end{cases}$$

۱) $\theta(n^2)$ ۲) $\theta(\lg n)$ ۳) $\theta(n)$ ۴) $\theta(n\lg n)$

☑ **پاسخ:** گزینه «۳» در حالت کلی اگر عدد x به تعداد i عامل 2 داشته باشد، مقدار $\lg n(x)$ برابر $i+1$ خواهد بود. در تحلیل سرشکن مقدار m با توجه به اینکه مجموع عامل‌های اعداد 1 تا n از مرتبه $\theta(n)$ است، مقدار m نیز از مرتبه $\theta(n)$ خواهد بود.

مثال ۸: آرایه‌ی A به طول n داده شده است. درایه‌های A در ابتدا صفرند. عمل درج زیر را n بار روی A انجام می‌دهیم. هزینه سرشکنی هر عمل درج (یعنی مجموع هزینه‌ی n عمل درج تقسیم بر n) کدام است؟ بهترین گزینه را انتخاب کنید. (مهندسی نرم‌افزار ـ سراسری ۹۱)

```
INSERT(x)
    n ← n+1
    t ← x
    for i ← 0 to[lg n]
        do if A[i]≠0
                then t ← t+A[i]
                     A[i] ← 0
                else A[i] ← t
                     stop and return
```

۱) $O(1)$ ۲) $O(\lg n)$ ۳) $O(n)$ ۴) $O(\lg^2 n)$

☑ **پاسخ:** گزینه «۱» نحوه اجرای این الگوریتم برای چند عدد ابتدایی به صورت زیر خواهد بود:

0	1	2	⋯	n		
x	0	0	⋯	0		مرحله ۱ $\Rightarrow$ تعداد دفعات اجرای حلقه $=1$
0	2x	0	0⋯	0		مرحله ۲ $\Rightarrow$ تعداد دفعات اجرای حلقه $=2$
x	2x	0	0⋯	0		مرحله ۳ $\Rightarrow$ تعداد دفعات اجرای حلقه $=1$
0	0	4x	0⋯	0		مرحله ۴ $\Rightarrow$ تعداد دفعات اجرای حلقه $=3$
x	0	4x	0⋯	0		مرحله ۵ $\Rightarrow$ تعداد دفعات اجرای حلقه $=1$
0	2x	4x	0⋯	0		مرحله ۶ $\Rightarrow$ تعداد دفعات اجرای حلقه $=2$
x	2x	4x	0⋯	0		مرحله ۷ $\Rightarrow$ تعداد دفعات اجرای حلقه $=1$
0	0	0	8x	0	 0	مرحله ۸ $\Rightarrow$ تعداد دفعات اجرای حلقه $=4$

بنابراین:

$$T(n)=2T(\frac{n}{2})+1 \Rightarrow T(n)\in\theta(n)$$

در نتیجه هزینه سرشکن شده عبارت است از:

$$\frac{O(n)}{n}=O(1)$$

مثال ۹: هریک از گزاره‌های زیر درست است یا نادرست؟ (مهندسی کامپیوتر، نرم‌افزار ـ دکتری ۹۲)

الف) اگر انجام هر عمل روی داده ساختاری به اندازه‌ی n به صورت سرشکنی $O(1)$ باشد، هزینه انجام n تا از این اعمال در بدترین حالت $O(n)$ است.

ب) در یک درخت دودویی با n عنصر که n بر سه بخش‌پذیر است، همیشه یک گره به نام x هست که تعداد گره‌های موجود در زیر درخت به ریشه‌ی x حداقل $\frac{n}{3}$ و حداکثر $\frac{2n}{3}$ باشد.

۱) الف: نادرست، ب: نادرست ۲) الف: نادرست، ب: درست ۳) الف: درست، ب: نادرست ۴) الف: درست، ب: درست

پاسخ: گزینه «۴» به‌طور کلی اگر روی یک ساختمان داده‌ها، مرتبه اجرایی هر عملی $O(1)$ باشد، n تا از این اعمال با مرتبه $O(n)$ انجام می‌شود و اگر در یک درخت جستجوی دودویی با n عنصر، n بر 3 بخش‌پذیر باشد، می‌توان گرۀ x را یافت، به طوری که تعداد گره‌های موجود در زیر درخت‌های x حداقل $\frac{n}{3}$ و حداکثر $\frac{2n}{3}$ باشد.

مثال ۱۰: می‌خواهیم مجموعه‌ای از n لیست خطی داشته باشیم که بتوانیم اعمال زیر را بر روی آن‌ها انجام دهیم:

$Insert(x,i)$: درج عنصر جدید x در لیست i، هزینه‌ی این کار 1 واحد است.

$Sum(i)$: جمع همه عناصر لیست i را به دست آورده و کل لیست را با یک عنصر با مقدار جمع به‌دست آمده جایگزین می‌کند. هزینه این کار برابر تعداد عناصر موجود در لیست i هنگام اجرای عمل فوق است.

اگر با لیست‌های تهی آغاز کنیم و اعمال گفته شده را به ترتیب دلخواه انجام دهیم. هزینه سرشکن هر یک از اعمال بالا متناسب با کدام گزینه است؟

(مهندسی کامپیوتر ـ سراسری ۹۴)

۱) درج: 2، جمع: 1
۲) درج: 1، جمع: 2
۳) درج: 1، جمع n
۴) درج: n ، جمع: 1

پاسخ: گزینه «۲» عمل $insert(x,i)$، دارای هزینه 1 واحد است، حال اگر n عمل درج انجام شود، هزینه سرشکن این عمل همواره 1 واحد خواهد بود. در مورد $sum(i)$، اگر لیست‌ها تهی باشد، باید ابتدا تعدادی عمل درج انجام شود. سپس با هر عمل $sum(i)$ تمام عناصر لیست i با یک عنصر جایگزین می‌شود که دارای هزینه‌ای برابر تعداد عناصر لیست i خواهد بود. به‌طور مثال فرض کنید n عمل درج در لیست i انجام شده باشد که در کل دارای هزینه n واحد می‌باشد. بعد از اجرای $sum(i)$ به‌جای کل این n عنصر، یک عنصر در لیست درج شده و هزینه این عمل نیز n واحد خواهد بود، بنابراین هزینه سرشکن این $n+1$ عمل را می‌توان به صورت مقابل محاسبه کرد:

$$\frac{n+n}{n+1}=\frac{2n}{n+1}\cong 2$$

مثال ۱۱: بر روی یک هرم کمینه تهی n عمل درج و حذف (با فرض داشتن محل حذف در هرم کمینه) با ترتیب دلخواه انجام می‌دهیم. هزینه سرشکنی هر یک از این دو عمل چقدر است؟

(مهندسی کامپیوتر، نرم‌افزار ـ دکتری ۹۴)

۱) درج: $O(\log n)$ و حذف: $O(\log n)$
۲) درج: $O(1)$ و حذف: $O(\log n)$
۳) درج: $O(\log n)$ و حذف: $O(1)$
۴) درج: $O(1)$ و حذف: $O(1)$

پاسخ: گزینه «۱» هزینه عمل درج و حذف در یک هرم با n عنصر از مرتبه $O(n)$ است. فرض کنید در یک هرم کمینه تهی، $\frac{n}{2}$ مرتبه عمل درج و $\frac{n}{2}$ مرتبه عمل حذف را انجام می‌دهیم، به‌طوری که در هر درج، عنصری کمتر از عنصر کمینه هرم به هرم اضافه شود و در هر عمل حذف، عنصر کمینه حذف شود. در این صورت، هزینه سرشکن شده هر عمل جمع (مجموع هزینه عمل‌ها تقسیم بر تعداد) برابر است با:

$$c=\frac{\sum_{i=1}^{\frac{n}{2}}\log i}{\frac{n}{2}} \quad , \quad \frac{1}{2}\log\frac{n}{2}<c<\log n \quad \Rightarrow \quad c\in O(\log n)$$

هزینه سرشکن شده هر عمل حذف نیز به همین شکل خواهد بود.

مثال ۱۲: فرض کنید آرایه $A[0,\cdots,n-1]$ آرایه‌ای است که در آن عددهای صفر و یک قرار دارد و عدد $x=\sum_{i=0}^{n-1}A[i]\times 2^i$ در آن نمایش داده شده است. برای اینکه x را یک واحد افزایش دهیم، از الگوریتم زیر استفاده می‌کنیم. هزینه سرشکن (Amortized cost) این الگوریتم چقدر است؟

(علوم کامپیوتر ـ سراسری ۹۶)

```
inc(A,n){
   i = 0;
   while(i < n and A[i] == 1){
      A[i] = 0
      i = i + 1
   }
   if i < n then A[i] = 1
}
```

۱) $O(n)$
۲) $O(\sqrt{n})$
۳) $O(1)$
۴) $O(\log n)$

خلاصه فصل چهارم

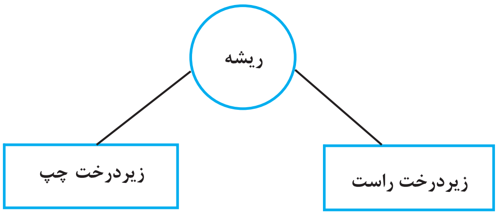
مطالب مطرح شده در این فصل را می‌توان به شکل زیر طبقه‌بندی نمود:

- داده‌ساختارهای مقدماتی
  - آرایه
    - ذخیره‌سازی آرایه در حافظه
    - به‌دست آوردن آدرس عناصر یک آرایه
    - ماتریس‌های بالامثلثی و پایین‌مثلثی
    - ماتریس‌های اسپارس
    - لیست مرتب
  - لیست پیوندی
    - پیاده‌سازی لیست پیوند یک طرفه
    - اعمال اصلی بر روی لیست‌های پیوندی
    - لیست‌های پیوندی حلقوی
    - لیست‌های پیوندی دوطرفه
  - پشته و صف
    - پشته
    - صف
    - پیاده‌سازی صف و پشته با استفاده از لیست پیوندی
    - انتقال داده‌ها در پشته و صف
    - استفاده از پشته در پیاده‌سازی صف
    - ارزشیابی عبارت‌ها
    - خروجی‌های ممکن در پشته

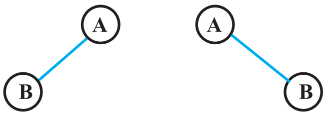
درسنامه (۲): درخت دودویی

دیدیم که هر درخت را می‌توانیم به صورت یک درخت دودویی معادل نمایش دهیم. درخت دودویی، یکی از انواع مهم ساختارهای درخت است که کاربرد زیادی دارد. درخت دودویی، درختی است که هر گره آن، حداکثر دو انشعاب دارد (یعنی، گره‌هایی که درجه بیشتر از دو داشته باشند، در آن وجود ندارد). برای درخت‌های دودویی، مفاهیم زیردرخت راست و چپ را تعریف می‌کنیم، زیرا این درخت یک درخت مرتب است و برای فرزند راست و چپ هر گره تمایز قائل هستیم. حتی درصورتی که یکی از فرزندان، گره‌ای null باشد، برای این‌که فرزند دیگر آن گره فرزند چپ یا راست باشد، تفاوت قائل هستیم. هر اشاره‌گر null را می‌توان یک درخت دودویی با 0 گره درنظر گرفت. در این صورت می‌توان تعریف زیر را برای درخت دودویی ارائه داد.

❖ **تعریف درخت دودویی:** درخت دودویی درختی است که هر گره از آن حداکثر دو فرزند دارد. شکل کلی آن در مقابل آمده است. هر درخت دودویی شامل ریشه و زیردرخت چپ و زیردرخت راست می‌باشد که این زیردرخت‌ها می‌توانند تهی باشند.

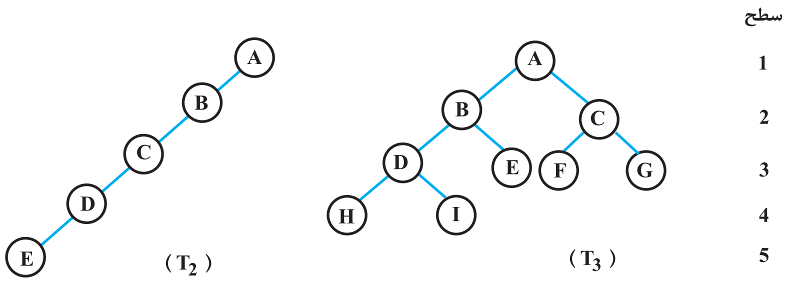


شکل مقابل دو درخت دودویی با دو گره را نشان می‌دهد که با توجه به اهمیت قرار گرفتن زیردرخت‌های چپ و راست متمایز هستند.



دو درخت دودویی متفاوت

شکل زیر دو نوع درخت دودویی خاص را نشان می‌دهد. شکل T_2 یک درخت مورب یا بهتر بگوییم، یک درخت مورب به چپ است، درخت مورب به راست نیز وجود دارد. شکل T_3 یک درخت دودویی کامل نامیده می‌شود. تعاریفی که برای درخت‌ها داشتیم، از جمله درجه، سطح، ارتفاع، برگ، والد و فرزند، همه روی درخت‌های دودویی نیز تعریف می‌شوند.



درخت دودویی مورب و کامل

خواص درخت‌های دودویی

قبل از اینکه به شیوه نمایش داده‌ها، در درخت دودویی بپردازیم، باید از دیدگاه مناسبی به درخت نگاه کنیم. می‌خواهیم حداکثر تعداد گره‌ها را در یک درخت دودویی به عمق K ، تعیین کنیم و رابطه‌ای بین تعداد گره‌های برگ و تعداد گره‌های درجهٔ دو درخت دودویی، پیدا نماییم.

بیشترین تعداد گره‌ها روی عمق i ام (سطح $i+1$ام) یک درخت دودویی 2^i است، $(i \geq 0)$.

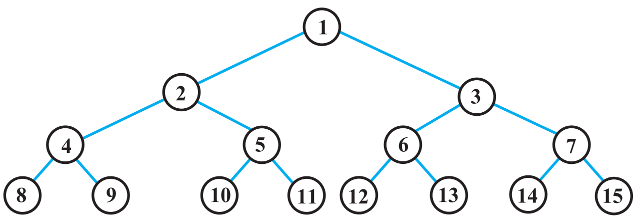
بیشترین تعداد گره‌ها در یک درخت دودویی به عمق k ، $2^{k+1}-1$ است، $(k \geq 0)$.

برای هر درخت دودویی غیرتهی، مانند T ، اگر n_0 تعداد گره‌های برگ و n_2 تعداد گره‌هایی باشد که درجهٔ 2 دارند، آنگاه $n_0 = n_2 + 1$ است. در درخت T_2، $n_0 = 1$ ، $n_2 = 0$ و در درخت T_3 (ب)، $n_0 = 5$ و $n_2 = 4$ است. اکنون می‌خواهیم درخت‌های دودویی پر و کامل را تعریف کنیم.

یک درخت دودویی پر به عمق k ، یک درخت دودویی به عمق k است که $2^{k+1}-1$ گره دارد، که در آن $k \geq 0$ می‌باشد.

بیشترین تعداد گره یک درخت دودویی با عمق k برابر $2^{k+1}-1$ است.

در شکل مقابل، درخت دودویی پر به عمق 3 نمایش داده شده است. فرض کنید که گره‌های یک درخت دودویی پر را شماره‌گذاری کنیم. گره ریشه که روی سطح یک (عمق صفر) است، شمارهٔ 1 و بعد گره‌های روی سطح 2 و به همین ترتیب. گره‌های هر سطح از چپ به راست، شماره‌گذاری می‌شوند. این شِمای عددگذاری شده، تعریف درخت دودویی کامل را روشن می‌کند.



درخت دودویی پر با عمق 3 ، گره‌ها به ترتیب شماره‌گذاری شده‌اند.

یک درخت دودویی با n گره و به عمق k، **کامل (complete)** است، اگر و تنها اگر، گره‌هایش مطابق با گره‌های شماره‌گذاری از 1 تا n، یک درخت دودویی پر به عمق k باشند، می‌توان گفت که عمق یک درخت دودویی کامل با n گره، $\log_2 n$ است. تعداد درختان دودویی برابر جمله nام از اعداد کاتالان یعنی $C_n = \frac{\binom{2n}{n}}{n+1}$ است.

مثال ۱۴: تعداد درخت‌های دودویی با n عنصر به ارتفاع $n-1$ برابر است با: (مهندسی نرم‌افزار ـ آزاد ۹۱)

۱) 1 ۲) 2 ۳) 2^{n-1} ۴) 2^n

پاسخ: گزینه «۳» گره اول در ریشه قرار می‌گیرد و تمام سطوح دارای یک گره می‌باشند، بنابراین به ازای هر یک از $n-1$ گره باقی‌مانده 2 حالت وجود دارد که در نهایت منجر به 2^{n-1} حالت خواهد شد.

مثال ۱۵: تعداد درختان برچسب‌دار متفاوت با n گره با برچسب‌های 1 تا n چند تا است؟ (علوم کامپیوتر ـ سراسری ۹۳)

۱) n^{n-2} ۲) $n!$ ۳) n^n ۴) $\frac{1}{n+1}\binom{2n}{n}$

پاسخ: گزینه «۱» تعداد درختان آزاد (غیرریشه‌دار) و برچسب‌دار با n رأس برابر n^{n-2} می‌باشد.

مثال ۱۶: با 25 عنصر چند درخت دودویی با ارتفاع کمینه می‌توان ساخت؟ (مهندسی کامپیوتر، نرم‌افزار ـ دکتری ۹۱)

۱) $\binom{16}{10}$ ۲) $56+\binom{14}{4}+\binom{16}{6}$ ۳) $\binom{8}{2}+8\binom{14}{3}+\binom{16}{6}$ ۴) $\binom{8}{2}+\binom{8}{2}\binom{14}{3}+\binom{16}{10}$

پاسخ: گزینه«۳» برای آن‌که درخت دودویی با 25 عنصر ارتفاع کمینه داشته باشد، باید ارتفاع آن معادل ارتفاع درخت دودویی کامل با 25 عنصر باشد که در این صورت، ارتفاع آن برابر $[\log_2^n]=[\log_2^{25}]=4$ است (ارتفاع ریشه صفر است) که در سطح‌های 1 و 2 و 3 باید تمامی گره‌ها باشد و در سطح‌های 4 و 5 باید 18 گره قرار گیرد که برای به‌دست آوردن ترکیب درخت‌ها، می‌توان از فرمول $\binom{8}{2}+8\binom{14}{3}+\binom{16}{6}$ استفاده کرد.

مثال ۱۷: کدام گزینه در مورد درخت‌های دودویی صحیح می‌باشد؟ (تعداد کل گره‌ها بزرگ‌تر یا مساوی یک فرض می‌شود.)
(علوم کامپیوتر ـ سراسری ۹۴)

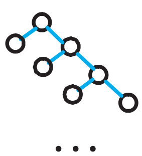
۱) همواره تعداد برگ‌ها و تعداد گره‌های تک‌فرزندی دو عدد متوالی می‌باشند.
۲) همواره تعداد برگ‌ها و گره‌های تک‌فرزندی برای درخت‌های کامل، دو عدد متوالی می‌باشند.
۳) همواره تعداد برگ‌ها و تعداد گره‌های دو فرزندی دو عدد متوالی می‌باشند.
۴) همواره تعداد گره‌های تک‌فرزندی و تعداد گره‌های دو فرزندی، دو عدد متوالی می‌باشند.

پاسخ: گزینه «۳» اگر در یک درخت دودویی با n گره، n_0 را تعداد گره‌های برگ و n_2 را تعداد گره‌های درجه 2 در نظر بگیریم، همواره $n_0 = n_2 + 1$ می‌باشد. پس همواره تعداد برگ‌ها و تعداد گره‌های دو فرزندی، دو عدد متوالی هستند.

مثال ۱۸: یک درخت 2- کامل درختی است که هر گره‌ی آن صفر یا 2 فرزند دارد. اگر $n(h)$ و $N(h)$ به ترتیب بیشینه و کمینه‌ی تعداد گره‌های یک درخت 2- کامل به ارتفاع h باشد، برای $h>0$، مقادیر $N(h), n(h)$ به ترتیب کدامند؟ (مهندسی کامپیوتر، نرم‌افزار ـ دکتری ۹۱)

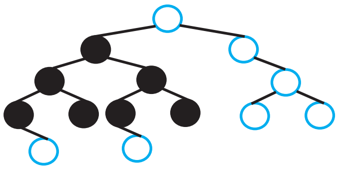
۱) $2^{h-1}, h+1$ ۲) $2^{h+1}, h+1$ ۳) $2^h-1, 2h+1$ ۴) $2^{h+1}-1, 2h+1$

پاسخ: گزینه «۴» در این سؤال توجه به این نکته ضروری می‌باشد که ریشه صفر در نظر گرفته شده است.



$(h>0)$ در متن سؤال نیز اشاره به همین موضوع دارد. لذا جواب به صورت زیر خواهد بود:
برای $n(h)$ کافی است درخت شبیه به یک مسیر باشد که از هر گره آن یک فرزند اضافه نیز خارج شده است. (مانند شکل مقابل)
در این صورت تعداد گره‌های هر سطح 2 است که اگر با ریشه جمع شود، $2h+1$ خواهد شد.
برای $N(h)$ نیز کافی است درخت دودویی کامل باشد (مانند هرم)، که در این صورت مجموع تعداد گره‌ها برابر $2^{h+1}-1$ خواهد بود.

مثال۱۹: یک درخت دودویی T با n گره داده شده است. می‌خواهیم بزرگ‌ترین زیردرخت پر T را به دست آوریم. درخت پر، یک درخت کاملاً متوازن است. یک زیردرخت پر در شکل زیر با گره‌های سیاه نمایش داده شده است. مرتبه بهترین الگوریتم برای محاسبه زیر درخت پر T با بیشترین تعداد عناصر کدام است؟ (مهندسی نرم‌افزار ـ سراسری ۹۱)



۱) $O(n)$

۲) $O(n\lg n)$

۳) $O(n\lg^2 n)$

۴) $O(n^2)$

☑ **پاسخ:** گزینه «۱» با یک بار پیمایش درخت و انتساب ویژگی full به هر یک از گره‌ها که به صورت زیر محاسبه می‌شود، می‌توان بزرگ‌ترین زیردرخت پر را تعیین نمود:

$$\text{full(node)} = \text{full(node} \rightarrow \text{left)} \;\&\; \text{full(node} - \text{right)} \;\&\; \text{height(node} \rightarrow \text{left)} == \text{height(node} \rightarrow \text{right)}$$

نمایش درخت دودویی

۱) نمایش با آرایه

طرح شماره‌گذاری شکل فوق مربوط به درخت دودویی پر، اولین ایده برای نمایش درخت دودویی در حافظه است. چون همه گره‌ها از 1 تا n شماره‌گذاری شده‌اند، می‌توانیم از یک آرایه یک‌بعدی، برای ذخیره گره‌ها استفاده کنیم. با استفاده از رابطه زیر، به آسانی می‌توان مکان گره‌های والد، فرزند چپ و راست گره i ، در یک درخت دودویی را مشخص کرد.

اگر یک درخت دودویی کامل با n گره به صورت ترتیبی، نمایش داده شود، برای هر گره با اندیس i ، $(1 \le i \le n)$ داریم:

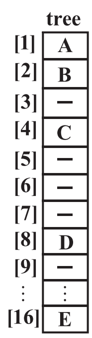
۱) parent (i) : اگر $i \ne 1$ باشد، والد i در $\left\lfloor \frac{i}{2} \right\rfloor$ است. اگر $i = 1$ باشد، i ریشه است و والدی ندارد.

۲) LeftChild (i) : اگر $2i \le n$ ، آنگاه فرزند چپ i در 2i است. اگر $2i > n$ باشد، i فرزند چپ ندارد.

۳) RightChild (i) : اگر $2i+1 \le n$ ، آنگاه فرزند راست i در 2i+1 است. اگر $2i+1 > n$ باشد، i فرزند راست ندارد.

این شیوه ارائه درخت دودویی، می‌تواند به آسانی برای همهٔ درخت‌های دودویی، به کار گرفته شود، مخصوصاً در حالتی که فضای زیادی از حافظه را در اختیار داریم. در شکل زیر، نمایش آرایه‌ای درخت‌های مورب و کامل T_3, T_2 آورده شده است. برای درخت دودویی کامل، نظیر آنچه که در T_3 داشتیم، این نمایش ایده‌آل است، هیچ حافظه‌ای هدر نمی‌رود. برای درخت مورب T_2 کمتر از نصف آرایه استفاده می‌شود.

در بدترین حالت، یک درخت مورب به عمق k، $2^k - 1$ حافظه نیاز دارد که از این مقدار، فقط k محل استفاده خواهد شد.



نمایش درخت T_2

نمایش درخت T_3

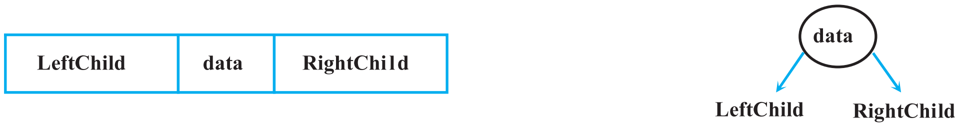
نمایش آرایه‌ای درخت‌های دودویی T_3, T_2

۲) نمایش با لیست پیوندی

گرچه نمایش با آرایه، برای درخت‌های دودویی کامل مناسب به نظر می‌رسد، اما برای بسیاری از درخت‌های دودویی دیگر، باعث اتلاف حافظه می‌شود. علاوه بر این، این روش از نارسایی‌های موجود در نمایش ترتیبی نیز برخوردار است. درج یا حذف گره‌ای از وسط درخت، مستلزم جابه‌جایی دیگر گره‌هاست، که باعث تغییر شماره سطح گره‌ها می‌شود. این مسائل را می‌توان با روش نمایش با لیست پیوندی حل کرد. در این روش هر گره سه فیلد دارد، LeftChild، data و RigthChild که در C چنین تعریف می‌شود:

```
typedef struct treenode * treepointer;
typedef  struct  treenode {
      treepointer LeftChild;
      char data;
      treepointer RightChild;
} tree;
```

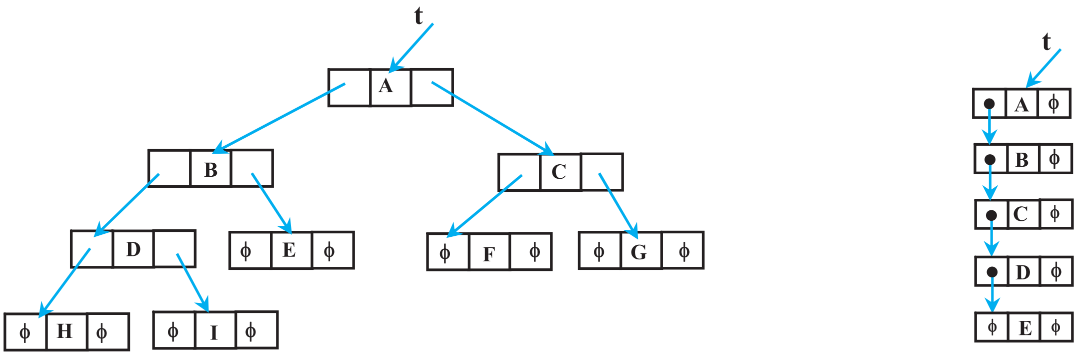
و یک گره، نظیر آنچه در شکل زیر آورده شده، نمایش داده می‌شود.



نمایش یک گره

هر چند که با این ساختار برای گره، تعیین والد گره مشکل می‌شود، اما بعداً خواهیم دید که در اکثر کاربردها مناسب است. در حالتی که تعیین والد یک گره اهمیت دارد، می‌توان فیلد چهارمی از نوع اشاره‌گر به نام parent نیز اضافه کرد.

درخت‌های T_2, T_3 که با این ساختار تعریف شده، در شکل زیر نمایش داده شده است. یک درخت با متغیری که به ریشه‌اش اشاره می‌کند، شناخته می‌شود.



(الف) نمایش لیست پیوندی درخت T_2 (ب) نمایش لیست پیوندی درخت T_3

نمایش لیست پیوندی درخت‌های دودویی T_2 و T_3 (ϕ = Null)

مثال ۲۰: بیشترین تعداد گره‌ها در یک درخت دودویی (Binary tree) با عمق h برابر کدام است؟

۱) 2^{h+1} ۲) $2^{h+1}+1$ ۳) $2^{h+1}-1$ ۴) 2^{h-1}

☑ **پاسخ:** گزینه «۳» درخت دودویی پر به عمق h همواره بیشترین تعداد گره‌ها را دارد که برابر $2^{h+1}-1$ است.

مثال ۲۱: در یک درخت دودویی کامل با 5 سطح حداکثر چند گره وجود دارد؟ (سطح ریشه برابر یک فرض شود.)

۱) 15 ۲) 16 ۳) 31 ۴) 32

☑ **پاسخ:** گزینه «۳» اگر h تعداد سطح‌های درخت دودویی باشد، بیشترین تعداد گره‌ها از فرمول زیر به دست می‌آید:

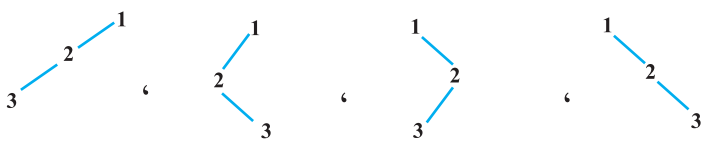
$$2^h-1=2^5-1=32-1=31$$

مثال ۲۲: با سه گره، چند درخت دودویی متمایز (از لحاظ توپولوژی) به ارتفاع 2 می‌توان ساخت؟

(ارتفاع یک درخت دودویی با یک گره را صفر در نظر بگیرید.)

۱) 4 ۲) 3 ۳) 5 ۴) 6

☑ **پاسخ:** گزینه «۱» یکی از گره‌ها باید به عنوان ریشه انتخاب شود و گره بعدی یا در زیردرخت چپ و یا در زیردرخت راست ریشه در نظر گرفته می‌شود که دو حالت دارد و گره بعدی نیز به همین صورت، که در کل با n گره می‌توان 2^{n-1} درخت دودویی متمایز با ارتفاع $n-1$ ساخت. بنابراین با سه گره، $2^{3-1}=2^2=4$ درخت دودویی متمایز به ارتفاع 2 می‌توان ساخت که عبارتند از:



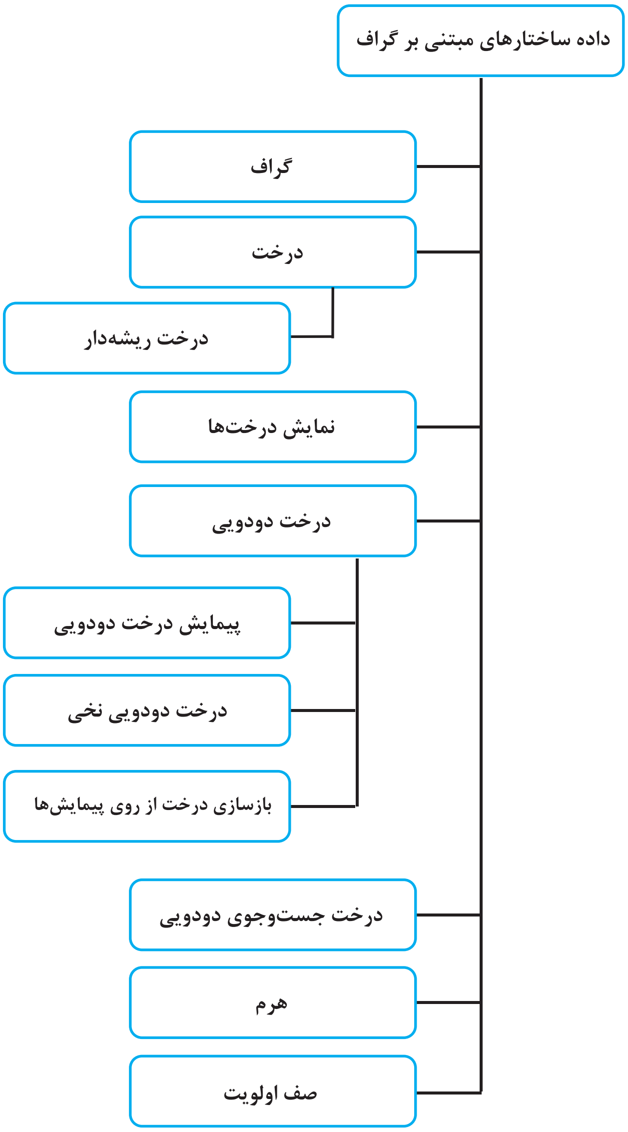
مثال ۲۳: یک درخت دودویی کامل با ارتفاع 6 چند گره دارد؟

۱) 128 ۲) 127

۳) 64 ۴) حداقل 64 و حداکثر 127 گره دارد.

خلاصه فصل پنجم

مطالب بیان شده در این فصل را می‌توان به زیر خلاصه کرد:



گراف: گراف مجموعه‌ای ناتهی از رئوس V است که با مجموعه‌ای از یال‌های E به هم متصل شده‌اند. هر یال دو رأس (گره) را به یکدیگر متصل می‌کنند. یال‌ها می‌توانند جهت‌دار یا بدون جهت و وزن‌دار یا بدون وزن باشند. گراف را ساده می‌نامند، اگر یال‌های آن بدون جهت باشد و هر یال دقیقاً بین دو رأس متمایز قرار گیرد. گراف را همبند می‌گویند، اگر بین هر دو رأس از آن مسیری وجود داشته باشد.

درخت: درخت‌ها گراف‌های همبند و بدون دور هستند. هر درخت n رأسی، $n-1$ یال دارد. بین هر دو رأس متمایز از درخت دقیقاً یک مسیر وجود دارد. درخت‌ها می‌توانند جهت‌دار یا بدون جهت باشند.

درخت ریشه‌دار: درختان ریشه‌دار نوع خاصی از درختان جهت‌دار هستند که دقیقاً یک گره با درجه ورودی 0 دارند و درجه ورودی سایر گره‌ها برابر 1 است. گره منحصربه‌فرد با درجه ورودی 0 از این درخت ریشه نام دارد. گره‌های با درجه خروجی 0 برگ نامیده می‌شوند. عمق هر گره برابر با طول مسیر ریشه تا آن گره است و ارتفاع گره برابر با بیشترین طول مسیر آن گره تا برگ‌ها است. عمق یا ارتفاع درخت برابر با ارتفاع ریشه درخت است.

یک درخت kتایی نامیده می‌شود، به‌شرطی که حداکثر درجه (خروجی) هر رأس آن برابر با k باشد. یک درخت پُر خوانده می‌شود، اگر تمام برگ‌های آن در یک سطح باشند و درجه تمام گره‌های غیربرگ آن برابر باشد. یک درخت کامل، درخت پری است که تعدادی از گره‌های برگ سمت راست آن می‌توانند حذف شده باشند.

نمایش درخت‌ها: درخت‌ها را می‌توان با لیست یا مجموعه‌ای از گره‌ها نمایش داد. ساختار گره‌ها می‌تواند شامل k فیلد فرزند باشد که هر فیلد یک فرزند را نمایش دهد و یا شامل 2 فیلد باشد که یک فیلد بیانگر اولین فرزند چپ و فیلد دیگر بیانگر اولین همزاد (هم‌نیا) راست باشد. این نمایش با نمایش درخت فرزند چپ ـ هم‌نیای راست معرفی می‌شود. درخت فرزند چپ ـ هم‌نیای راست را می‌توان به صورت درخت فرزند چپ ـ فرزند راست معادل نمایش داد.

درخت دودویی: درخت دودویی نوع خاصی از درختان است که هر گره از آن حداکثر دو فرزند دارد که ترتیب فرزندان مهم است.

پیمایش درخت دودویی: در پیمایش سطح ترتیب از داده ساختار صف برای پیمایش گره‌ها استفاده می‌شود و ترتیب ملاقات گره‌ها باتوجه به فاصله گره از ریشه است. در پیمایش پیش‌ترتیب (preorder) ابتدا گره سپس زیردرخت چپ و پس از آن زیردرخت راست ملاقات می‌شود. در پیمایش میان‌ترتیب (inorder) ابتدا زیردرخت چپ، سپس گره کنونی و پس از آن زیردرخت راست ملاقات می‌شود. در پیمایش پس‌ترتیب (postorder) گره کنونی پس از زیردرخت چپ و زیردرخت راست ملاقات می‌شود.

درخت دودویی نخی: در درخت دودویی نخی، اشاره‌گرهای Null گره‌ها به اشاره‌گرهای نخی تبدیل می‌شوند که به عنصر بعد و قبل پیمایش میان‌ترتیب اشاره می‌کنند. با این اشاره‌گرها برای پیمایش میان‌ترتیب درخت دودویی نیاز به استفاده از پشته نخواهیم داشت.

بازسازی درخت دودویی: درخت دودویی با در اختیار داشتن پیمایش میان‌ترتیب و یکی از دو پیمایش پیش‌ترتیب یا پس‌ترتیب به‌صورت یکتا قابل بازسازی است. با در اختیار داشتن پیمایش‌های پیش‌ترتیب و پس‌ترتیب، اگر درخت دارای k گره تک فرزندی باشد، 2^k درخت متمایز قابل بازسازی است.

درخت جست‌وجوی دودویی: درخت جست‌وجوی دودویی، درختی دودویی است که مقدار هر گره از آن از تمام گره‌های زیردرخت چپ آن بیشتر و از تمام گره‌های زیردرخت راست آن کمتر است. هیچ دو گره‌ای در این ساختار، مقدار برابر ندارند. تعداد درختان جست‌وجوی دودویی با n گره برابر جمله nام از اعداد کاتالان است.

جمع‌بندی:

داده ساختارها	جست‌وجو	درج	درج (با معلوم بودن جایگاه)	حذف (با معلوم بودن گره)	یافتن کمینه	یافتن بیشینه	استخراج کمینه
درخت دودویی	$O(n)$	-	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
هرم کمینه	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(n)$	$O(\log n)$
B.S.T با ارتفاع h	$O(h)$	$O(h)$	$O(h)$	$O(h)$	$O(h)$	$O(h)$	$O(h)$

هرم: هرم یک درخت دودویی کامل است که باتوجه به نوع کمینه یا بیشینه بودن آن، مقدار هر گره از دو فرزندش نابیشتر یا ناکمتر است. ریشه هرم کمینه، کمترین عنصر از هرم و ریشه هرم بیشینه، بیشترین عنصر از هرم است.

صف اولویت: صف اولویت نوعی ساختار داده انتزاعی است که برای عناصر، اولویت قائل می‌شود؛ به‌گونه‌ای که عنصر با اولویت بیشتر، زودتر از صف خارج می‌شود. جدول زیر هزینه توابع صف اولویت پیاده‌سازی شده با چند داده‌ساختار را نشان می‌دهد.

داده‌ساختار	یافتن کمینه	حذف کمینه	استخراج کمینه	درج	ادغام	کاهش مقدار
آرایه مرتب	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
آرایه نامرتب	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(1)$
لیست پیوندی مرتب	$O(1)$	$O(1)$	$O(1)$	$O(n)$	$O(n)$	$O(n)$
لیست پیوندی نامرتب	$O(n)$	$O(1)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$
هرم دودویی	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n)$	$O(\log n)$
هرم دوجمله‌ای	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(\log n)$	$O(\log n)$
هرم فیبوناچی (سرشکن شده)	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$

فصل ششم

«داده‌ساختارهای پیشرفته»

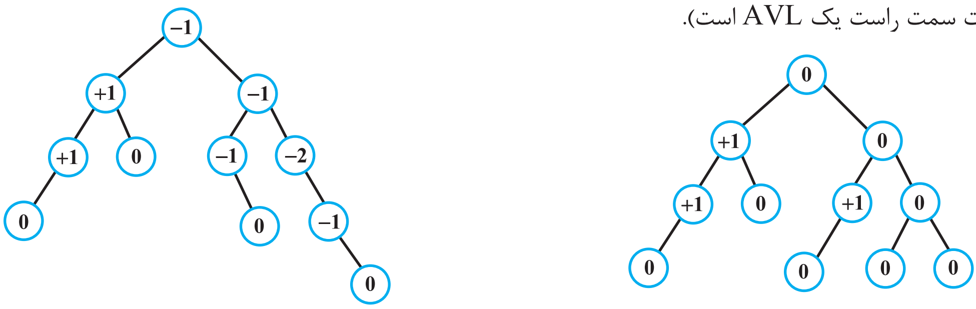
مقدمه

در فصل‌های قبل داده‌ساختارهای مقدماتی و داده‌ساختارهای پرکاربرد مبتنی بر گراف را بررسی کردیم. این فصل به بررسی مجموعه دیگری از داده‌ساختارهای مبتنی بر گراف اختصاص دارد که به نسبت داده‌ساختارهایی که تا به حال معرفی کردیم، ساختار پیچیده‌تری دارند و تعداد سؤالاتی که از این مباحث مطرح می‌شود به مراتب کمتر از مباحث پیشین است. به همین دلیل توصیه می‌کنیم در صورتی که برای مطالعه منابع آزمون، محدودیت زمانی دارید، پس از خواندن سایر فصل‌های کتاب، مطالعه این فصل را شروع کنید.

درسنامه (۱): درخت‌های دودویی متوازن

درخت‌های AVL

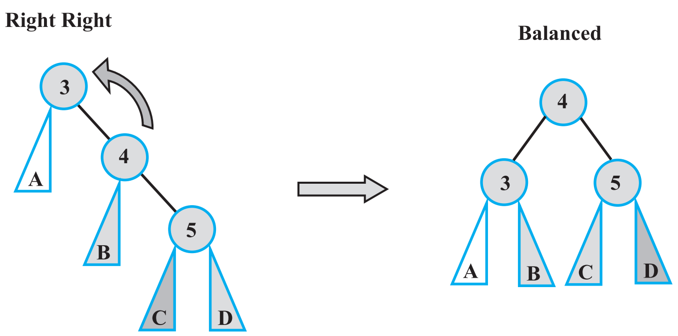
یک درخت AVL درخت جست‌وجوی دودویی متوازنی است که در آن عمق درخت همواره از مرتبه $\theta(\log_2^n)$ می‌باشد. در این نوع درخت‌ها برای هر گره، یک فاکتور توازن (BF) تعریف می‌شود که نشان‌دهنده تفاضل عمق زیردرخت چپ و راست گره مربوطه می‌باشد و برای هر گره، BF می‌تواند دارای یکی از سه مقدار $+1$، 0 یا -1 باشد. به عنوان مثال، در شکل زیر دو درخت آمده که روی هر گره فاکتور توازن آن نوشته شده است (درخت سمت چپ AVL نمی‌باشد و درخت سمت راست یک AVL است).



برای این که عمق درخت از مرتبه $\theta(\log_2 n)$ باقی بماند، کافی است در هر درج و حذف با استفاده از اعمال چرخش (rotation) فاکتور توازن گره‌هایی که فاکتور توازن آن‌ها $+2$ یا -2 شده را اصلاح نماییم. در عمل درج چرخش‌ها حول پایین‌ترین عنصری که درجه آن ± 2 شده است انجام می‌شود، در کل چهار چرخش زیر را می‌توان در نظر گرفت (فرض کنید p پایین‌ترین گرهی می‌باشد که درجه آن $+2$ یا -2 شده است):

۱ـ چرخش RR (راست ـ راست)

اگر گره جدید در زیردرخت راست فرزند راست p درج شده باشد (فاکتور توازن فرزند راست p برابر -1 است). در این صورت چرخش به صورت زیر خواهد بود:



مدرسان شریف

فصل دهم

«الگوریتم‌های حریصانه (Greedy Algorithms)»

مقدمه

یکی از روش‌های حل مسائل بهینه‌سازی، الگوریتم‌های حریصانه می‌باشد. این روش در هر مرحله از بین انتخاب‌های ممکن، گزینه‌ای را انتخاب می‌کند که در همان لحظه بهترین انتخاب به نظر می‌رسد و در انتخاب‌ها به عواقب کار اندیشیده نمی‌شود. این روش، نسبت به سایر روش‌های بررسی شده ساده‌تر و کم‌هزینه‌تر است و اصولاً در مسائلی که سرعت یافتن یک راه‌حل تقریبی از یافتن حل دقیق مهم‌تر باشد، استفاده می‌گردد.

الگوریتم‌هایی که مبنای حریصانه دارند همواره حل بهینه را نمی‌یابند، اما جواب‌های یافته شده توسط آنها معمولاً به حل بهینه نزدیک است (یافتن حل بهینه توسط الگوریتم‌های حریصانه برای برخی از مسائل خاص که در این فصل بررسی می‌شوند، تضمین شده است).

الگوریتم‌های پریم، کروسکال و دایکسترا که در فصل سیزدهم از کتاب آمده‌اند، مثال‌های آموزشی خوبی برای این فصل هستند.

درسنامه (۱): حل مسائل زمان‌بندی (Scheduling)

زمان‌بندی (Scheduling)

۱ـ زمان‌بندی بدون مهلت

فرض کنید تعدادی کار (job) موجود است که هرکدام زمان t_i را برای کامل شدن لازم دارند و در هر زمان فقط می‌توانیم یک کار را انجام دهیم. (بنابراین زمانی که یک کار در حال انجام شدن است، بقیه کارها در حالت انتظار به سر می‌برند). هدف انجام کارها به ترتیبی است که کمترین زمان انتظار برای کل کارها به‌دست آید. این مسأله به عنوان یکی از صورت‌های مسأله زمان‌بندی مطرح می‌شود و در ادامه با یک مثال شرح داده شده است.

مثال ۱: فرض کنید سه کار j_1 و j_2 و j_3 که هر کدام زمان t_i را برای تکمیل شدن لازم دارند داده شده‌اند که t_iها عبارتند از: $t_1 = 5, t_2 = 10$ ، $t_3 = 4$ در این صورت زمان‌بندی‌های زیر برای این کارها می‌تواند صورت پذیرد.

زمان بندی	زمان کل مورد نیاز برای انجام کارها
$[1,2,3]$	$5+(5+10)+(5+10+4)=39$
$[1,3,2]$	$5+(5+4)+(5+4+10)=33$
$[2,1,3]$	$10+(10+5)+(10+5+4)=44$
$[2,3,1]$	$10+(10+4)+(10+4+5)=43$
$[3,1,2]$	$4+(4+5)+(4+5+10)=32$
$[3,2,1]$	$4+(4+10)+(4+10+5)=37$

(در این جدول، زمان‌بندی $[i_1, i_2, i_3]$ نشان‌دهنده این است که ابتدا کار j_{i_1} سپس کار j_{i_2} و بعد از آن کار j_{i_3} انجام می‌شود.)

به عنوان مثال زمان‌بندی موجود در سطر اول یعنی، $[1,2,3]$ نشان‌دهنده این است که ابتدا کار j_1 ، بعد از آن کار j_2 و در نهایت کار j_3 انجام می‌شود. کار j_1 به پنج واحد زمانی برای انجام شدن نیاز دارد، سپس زمانی که کار j_2 برای تکمیل شدن نیاز دارد برابر است با 10 واحد زمانی به اضافه پنج واحد زمانی که در انتظار بوده است و با همین استدلال کار j_3 به اندازه 15 واحد زمانی در انتظار خواهد بود و سپس چهار واحد نیز برای تکمیل شدن نیاز دارد؛ بنابراین ترتیب $[1,2,3]$ برای زمانبندی به زمان کل 39 واحد نیاز خواهد داشت.

همان‌گونه که مشخص است برای این که تمام حالات ممکن را بررسی کنیم، باید $n!$ حالت را بررسی نماییم (n تعداد کارهاست). اما طبق قضیه‌ای ثابت می‌شود که همواره بهترین زمان‌بندی در حالتی به‌دست می‌آید که کارها را به ترتیب غیرنزولی زمان آن‌ها مرتب کنیم. به عنوان نمونه، در این مثال قابل مشاهده است که بهترین زمان‌بندی، زمان‌بندی $[3,1,2]$ است که: $t_3 < t_1 < t_2$

بنابراین قسمت اصلی حل این مسأله مرتب نمودن کارها بر حسب زمان اجرای آن‌ها است، در نتیجه زمان الگوریتمی که جواب بهینه را می‌یابد برابر $\theta(n\log n)$ خواهد بود.

مثال۲: فرض کنید چهار کار J_4 و J_3 و J_2 و J_1 با زمان‌های $t_1 = 7, t_2 = 5, t_3 = 13, t_4 = 20$ داده شده است در این صورت، کمترین زمان کل برای تکمیل کارها کدام است؟

۱) 87　　۲) 45　　۳) 57　　۴) 78

پاسخ: گزینه «۱» برای به‌دست آوردن حل بهینه کافیست کارها را براساس مدت زمان ارائه مرتب نماییم:

Job	زمان
J_2	5
J_1	7
J_3	13
J_4	20

در نتیجه زمان کل تکمیل در حالت بهینه به صورت زیر است:

زمان تکمیل برای کار اول : 5

زمان تکمیل برای کار دوم : $5+7=12$

زمان تکمیل برای کار سوم : $5+7+13=25$

زمان تکمیل برای کار سوم : $5+7+13+20=45$

بنابراین کمترین زمان تکمیل در کل برابر با 87 است.

در زیر الگوریتم حریصانه مربوط به مسأله زمان‌بندی ارائه شده است.

در این الگوریتم آرایه S شامل زمان مربوط به کارها (t_iها)، n تعداد کارها و F مجموعه شامل حل می‌باشد. همچنین دستور Sort (S) برای مرتب کردن آرایه S استفاده شده است.

```
Scheduling (S,n,F)
1   Sort (S);
2   F = ϕ;
3   For i = 1 to n
4       F = F ∪ S[i];
```

زمان‌بندی با هدف انجام بیش‌ترین تعداد کارها

در این مسأله تعدادی کار با زمان‌های شروع و پایان مشخص داده شده‌اند و یک پردازنده برای انجام کارها موجود است، هدف انجام بیش‌ترین تعداد کار ممکن توسط پردازنده می‌باشد (واضح است که با توجه به زمان‌های شروع و پایان در بسیاری از موارد، تمام کارها قابل انجام نمی‌باشند؛ زیرا در هر زمان بیش از یک کار قابل انجام نیست). بنابراین می‌توان مسأله‌ی زمان‌بندی با مهلت معین را به‌صورت زیر درنظر گرفت:

مسأله زمان‌بندی با هدف انجام بیش‌ترین تعداد کارها
ورودی: کارهای $j_1, j_2, \ldots, j_n$ که هریک با یک بازه شروع و پایان (s_i, e_i) مشخص می‌شوند.
خروجی: بیش‌ترین تعداد کار ممکن به‌طوری که با یکدیگر سازگار باشند (بازه شروع و پایان هیچ دو کاری با یکدیگر تداخل نداشته باشد).

برای حل این مسأله کافیست کارها براساس زمان پایان به‌صورت صعودی مرتب شوند و سپس بیش‌ترین تعداد کار سازگار انتخاب شوند. به مثال زیر دقت کنید:

مثال۳: شش کار زیر با زمان‌های شروع و پایان داده شده را درنظر بگیرید:

j_i	j_1	j_2	j_3	j_4	j_5	j_6
s_i (زمان شروع)	2	1	4	5	3	1
e_i (زمان پایان)	5	3	6	7	4	2

در این صورت بیش‌ترین تعداد کار قابل انجام بر روی یک پردازنده چقدر است؟

پاسخ: براساس توضیح ارائه شده، ابتدا کارها را براساس زمان پایان و به صورت صعودی مرتب می‌کنیم:

$\xrightarrow{\text{مرتب‌سازی}}$

j_i	j_6	j_2	j_5	j_1	j_3	j_4
s_i	1	1	3	2	4	5
e_i	2	3	4	5	6	7

در این صورت نحوه‌ی انتخاب کارها به شکل زیر خواهد بود:

کار j_6 انتخاب می‌شود.

$\Downarrow$

کار j_2 انتخاب نمی‌شود زیرا با j_6 تداخل دارد.

$\Downarrow$

کار j_5 انتخاب می‌شود (با j_6 تداخل ندارد).

$\Downarrow$

کار j_1 انتخاب نمی‌شود زیرا با j_5 تداخل دارد.

$\Downarrow$

کار j_3 انتخاب می‌شود (با j_5 و j_6 تداخل ندارد).

$\Downarrow$

کار j_4 انتخاب نمی‌شود زیرا با j_3 تداخل دارد.

بنابراین کارهای j_6، j_5 و j_3 انتخاب شده‌اند و حداکثر تعداد کارهای قابل اجرا توسط یک پردازنده سه کار می‌باشد.

۲ــ زمان‌بندی با مهلت معین

حال فرض کنیم در مسأله‌ی مذکور علاوه بر مقادیر داده شده برای هر کار، یک مهلت معین (deadline) نیز وجود داشته باشد و هر کار فقط در صورتی که تا قبل از مهلت معین موردنظر کامل شود، قابل انجام است و در غیر این‌صورت، دیگر آن کار از دست رفته است. اگر علاوه بر مهلت برای هر کار مقدار بهره را نیز داشته باشیم که سود حاصل از انجام آن کار را نشان می‌دهد، آنگاه به این مسأله، مسأله زمان‌بندی با مهلت معین می‌گوییم.

در این مسأله با توجه به مهلت معین برای کارها هر ترتیبی نمی‌تواند امکان‌پذیر باشد.

بنابراین می‌توان مسأله زمان‌بندی با مهلت معین را به صورت زیر در نظر گرفت:

مسأله زمان‌بندی با مهلت معین
ورودی: کارهای $j_1,\dots,j_n$ که به ترتیب دارای مهلت معین $d_1,\dots,d_n$ برای اجرا می‌باشند و سود حاصل از اجرای این کارها (در صورتی که پس از اتمام مهلت اجرا نشوند) به ترتیب $p_1,\dots,p_n$ می‌باشد (زمان اجرای هر کار یک واحد زمانی در نظر گرفته می‌شود).
خروجی: زیرمجموعه‌ای از کارها با یک ترتیب مشخص که در آن مهلت هر کار رعایت شده و این زیرمجموعه در کل بیشترین سود ممکن را داشته باشد.

به عنوان مثال اگر چهار کار زیر را داشته باشیم.

کار	مهلت	سود
1	3	20
2	1	15
3	1	10
4	3	25

آنگاه ترتیب $[4 \quad 1 \quad 3 \quad 2]$ یک ترتیب ممکن نیست، زیرا کار 3 دارای مهلت 1 است در حالی که پس از $1+1=2$ واحد زمانی کامل می‌شود.

برای حل این مسأله کافیست ابتدا کارها براساس سود به‌صورت نزولی مرتب شوند و سپس بیش‌ترین تعداد کار ممکن (به‌گونه‌ای که در مهلت معین قابل اجرا باشند) انتخاب شود. برای انجام عملیات بیان شده، مرتب نمودن کارها به سادگی توسط یک الگوریتم مرتب‌سازی در زمان $\theta(n\log n)$ قابل انجام است، اما نکته مهم چگونگی بررسی انجام کارها در مهلت معین می‌باشد. اگر برای این هدف، تمام ترتیب‌های ممکن از کارهای انتخاب شده بررسی شوند، آنگاه زمان کلی الگوریتم از مرتبه $O(n!)$ می‌باشد. اما طبق قضیه زیر (بدون اثبات) می‌توان در زمان بسیار کمتری این کنترل را انجام داد.

قضیه: **یک مجموعه از کارها با مهلت معین تنها در صورتی قابل انجام است (به‌گونه‌ای که مهلت معین تمام کارها رعایت شده باشد) که ترتیب صعودی کارها براساس مهلت معین، قابل انجام باشد.**

به عنوان مثال اگر جدول قبلی کارها را درنظر بگیریم، آنگاه برای کنترل این‌که آیا کارهای j_1، j_2 و j_4 یک مجموعه‌ی امکان‌پذیر از کارها می‌سازند، نیازی به بررسی تمام حالات نداریم و کافیست تنها ترتیب $[j_2, j_4, j_1]$ بررسی شود (ترتیب صعودی براساس مهلت معین) و با توجه به این‌که ترتیب مورد نظر یک ترتیب امکان‌پذیر برای انجام کارها می‌باشد، در نتیجه کارهای j_1، j_2 و j_4 قابل انجام هستند که مقدار سود 60 را دربر خواهند داشت.

الگوریتم زمان‌بندی با مهلت معین در زیر ارائه شده است، در این الگوریتم آرایهٔ [ ]d شامل مهلت کارها و آرایه [ ]p شامل سود مربوط به کارها می‌باشد:

```
Job Sequence (d[ ] , p [ ] , n)
1         j[n]: array of integer,
2         j[0]=0;   d[0]=0;
3         j[1]=1;
4         job-no=1;
5         for i=2  to  n  do
6         {
7                 k = job-no;
8                  while ((d[i] < d[j[k]]) and (d [j [k]] ≠ k)
9                         k = k - 1;
10                if ((d[i] ≥ d[j[k]]) and (d [i] > k))
11                {
12                        for ℓ = job-no  downto  k + 1   Do
13                            j[ℓ + 1] = j[ℓ];
14                        j[k + 1]=i;
15                        job-no = job-no + 1;
16                }
17        }
```

در این الگوریتم، فرض بر این است که آرایه [ ]p به صورت غیرصعودی مرتب است و همچنین در نهایت آرایه [ ]j شامل ترتیب بهینه نهایی برای کارها خواهد بود.

در ابتدا با دستور $j[1] = 1$ ترتیب بهینه شامل کار اول خواهد بود سپس، در حلقه for اصلی هر بار مکان درج کار بعدی (که با job no مشخص می‌شود) به کمک حلقه while و با توجه به مهلت آن تعیین می‌گردد، که منظور از $d[j[k]]$ مهلت مقرر مربوط به کار k ام در لیست j می‌باشد. سپس در دستور if بررسی می‌شود که لیست j حاصل از درج کار شماره i یک لیست قابل اجرا (feasible) از کارها می‌باشد یا خیر و در صورتی که قابل اجرا باشد، عمل درج در حلقه for (خط ۱۸) انجام می‌شود.

مثال ۴: فرض کنید هفت کار با مهلت‌های d_i و بهره‌های p_i طبق جدول زیر داده شده است. در این صورت در زمان‌بندی بهینه کارها مقدار سود چقدر خواهد بود؟

سود	مهلت
15	**3**
50	**1**
10	**1**
5	**2**
60	**3**
30	**1**
20	**2**

۱) 110　　۲) 120　　۳) 130　　۴) 145

پاسخ: گزینه «۳» ابتدا کارها را براساس ترتیب غیرصعودی بهره مرتب می‌کنیم:

سود	مهلت	کار
60	3	1
50	1	2
30	1	3
20	2	4
15	3	5
10	1	6
5	2	7

بنابراین آرایه $p[\ ]$ به صورت زیر مرتب می‌شود:

p[1]	p[2]	p[3]	p[4]	p[5]	p[6]	p[7]
60	50	30	20	15	10	5

و آرایه $d[\ ]$ به شکل زیر خواهد بود:

d[0]	d[1]	d[2]	d[3]	d[4]	d[5]	d[6]	d[7]
0	3	1	1	2	3	1	2

جدول زیر روند اجرای الگوریتم در تعیین تعداد نهایی آرایه j که حاوی زمانبندی بهینه می‌باشد را نشان می‌دهد:

امکان‌پذیر بودن	مقدار سود	نحوه تغییر آرایه J	
✓	60	[1]	مرحلهٔ اول
✓	110	[2,1]	مرحلهٔ دوم
×	—	[2,3,1]	مرحلهٔ سوم
✓	130	[2,4,1]	مرحلهٔ چهارم
×	—	[2,4,1,5]	مرحلهٔ پنجم
×	—	[2,4,1,6]	مرحلهٔ ششم
×	—	[2,4,1,7]	مرحلهٔ هفتم

بنابراین مقدار نهایی آرایه j به صورت [2,4,1] است و بیشترین مقدار سود 130 می‌باشد.

مثال۵: **کارهای موجود در جدول زیر با جریمه‌های متناظر با هر کار داده شده است. هدف انجام کارها به ترتیبی است که کمترین جریمه را متحمل شویم. در این صورت کمترین مقدار جریمه ممکن چقدر است؟**

جریمه	مهلت
20	4
70	4
60	2
50	4
30	1
10	6
40	3

۱) 50 ۲) 40 ۳) 70 ۴) 60

پاسخ: گزینه «۱» مسأله مطرح شده در حقیقت همان مسأله زمانبندی با مهلت معین می‌باشد که هدف به‌دست آوردن کمترین جریمه ممکن است، بنابراین باید کارها با جریمه‌های بیشتر زودتر انجام شوند تا متحمل جریمه آنها نشویم. در نتیجه، الگوریتم حریصانه بررسی شده در مثال قبل برای این مسأله نیز جواب بهینه را تولید خواهد کرد که این بار براساس جریمه کارهای داده شده را به صورت غیرصعودی مرتب می‌نماییم و بنابراین جدول زیر به‌دست می‌آید.

جریمه	مهلت	کار
70	4	1
60	2	2
50	4	3
40	3	4
30	1	5
20	4	6
10	6	7

مراحل اجرای الگوریتم به شکل زیر خواهد بود:

	نحوه تغییر آرایه J	امکان‌پذیر بودن
مرحلهٔ اول	$[1]$	✓
مرحلهٔ دوم	$[2,1]$	✓
مرحلهٔ سوم	$[2,1,3]$	✓
مرحلهٔ چهارم	$[2,4,1,3]$	✓
مرحلهٔ پنجم	$[5,2,4,1,3]$	×
مرحلهٔ ششم	$[1,4,1,3,6]$	×
مرحلهٔ هفتم	$[2,4,1,3,7]$	✓

بنابراین ترتیب بهینه نهایی به صورت $[2,4,1,3,7]$ می‌باشد که کارهای شماره 5 و 6 در مهلت مقرر انجام نشده‌اند. بنابراین به ترتیب دارای جریمه 30 و 20 می‌باشند و در نهایت کل جریمه برابر 50 است.

نکته۱: الگوریتم حریصانه برای مسأله زمانبندی با مهلت معین همواره بهترین حل با بیشترین بهره را به‌دست می‌آورد.

نکته۲: بدترین زمان الگوریتم حریصانه برای مسألهٔ زمانبندی با مهلت معین برابر $w(n)=\theta(n^2)$ است.

مثال۶: در ساعت صفر، 10 نفر با شماره‌های 1 تا 10 برای پر کردن سطل خود در مقابل یک شیر آب صف کشیده‌اند. به محض این که سطل فردی که در جلوی شیر آب است پر می‌شود، او کنار می‌رود و نفر بعدی در صف جای او را می‌گیرد. فرض کنید سطل نفرم iام به اندازه‌ای است که پر کردن آن i دقیقه طول می‌کشد. زمانی که نفر iام سطل خود را کاملاً پر می‌کند را «زمان معطلی» نفر iام می‌نامیم. نحوه قرار گرفتن افراد در صف ابتدایی مجموع زمان معطلی را تعیین می‌کند. کمینه مجموع زمان معطلی این 10 نفر چند است؟ (مهندسی کامپیوتر، نرم‌افزار ـ دکتری ۹۳)

۱) 55　　۲) 110　　۳) 220　　۴) 302

پاسخ: گزینه «۳» مسأله مطرح شده همان مسأله زمان‌بندی بدون مهلت با هدف کمینه کردن زمان انتظار است. بنابراین کافی است کارها به ترتیب صعودی زمان مورد نیاز اجرا شوند و در نتیجه زمان معطلی کمینه عبارت است از:

$$1+(1+2)+(1+2+3)+\ldots+(1+2+3+\ldots+9+10)=\sum_{i=1}^{10}\frac{i(i+1)}{2}=220$$

مثال۷: فرض کنید تعدادی بازه به صورت $[s_i,f_i]$ داده شده و هدف، انتخاب بیشترین تعداد بازه‌هایی است که دوبه‌دو ناهمپوشان باشند. از بین چهار ایده حریصانه زیر چند مورد همواره حل بهینه را ایجاد می‌کند؟

الف) انتخاب بازه‌ها براساس s_i ها از کوچک به بزرگ و حذف بازه‌هایی که با بازه انتخاب شده همپوشانی دارند.

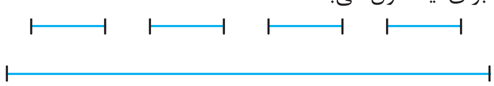
ب) انتخاب بازه‌ها براساس طول آن‌ها از کوچک به بزرگ و حذف بازه‌هایی که با بازه انتخاب شده در هر مرحله تداخل دارند.

ج) انتخاب بازه‌ها براساس f_i ها از کوچک به بزرگ و حذف بازه‌هایی که با بازه انتخاب شده در هر مرحله تداخل دارند.

د) انتخاب بازه‌ها براساس داشتن کمترین تداخل با بازه‌های دیگر و حذف بازه‌هایی که با بازه انتخاب شده در هر مرحله تداخل دارند.

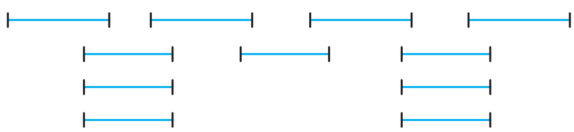
۱) یک مورد　　۲) دو مورد　　۳) سه مورد　　۴) چهار مورد

پاسخ: گزینه «۱» شکل زیر یک مثال نقض برای ایده اول می‌باشد:



شکل زیر یک مثال نقض برای ایده دوم می‌باشد:

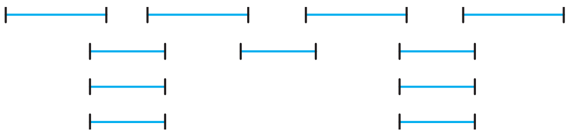
شکل زیر یک مثال نقض برای ایده چهارم می‌باشد:



اما ایده بیان شده در مورد «ج» همان روش حریصانه بیان شده در متن درس برای مسألهٔ زمان‌بندی بدون مهلت می‌باشد که همواره حل بهینه را ایجاد می‌کند.

مثال۸: **از بین ایده‌های بیان شده در مثال قبل چند مورد بر روی بازه‌های موجود در شکل زیر حل بهینه را ایجاد می‌کنند؟**

۱) یک مورد
۲) دو مورد
۳) سه مورد
۴) هر چهار ایده



پاسخ: گزینه «۲» حل بهینه شامل انتخاب چهار بازه موجود در سطر اول می‌باشد که توسط ایده‌های اول و سوم این بازه‌ها انتخاب می‌شوند.

مثال۹: **فرض کنید می‌خواهیم برای تعدادی کلاس درس که هر کدام ساعت شروع و خاتمه‌شان مشخص است، اتاق رزرو کنیم. هدف کمینه کردن تعداد اتاق‌هاست. به این منظور از الگوریتم حریصانه‌ی زیر استفاده می‌کنیم:**

- **درس‌ها را براساس زمان خاتمه‌شان صعودی مرتب می‌کنیم.**
- **به‌ترتیب لیست صعودی به درس‌ها بدین شکل اتاق اختصاص می‌دهیم: اگر در میان اتاق‌هایی که تا الان از آن‌ها استفاده شده اتاقی باشد که بتوان این درس را در آن جا برگزار کرد (یعنی با درس‌هایی که قبلاً به این اتاق تخصیص داده شده‌اند، همپوشانی ندارد)، این کار را انجام می‌دهیم. در غیر این‌صورت، اتاق جدیدی به این درس اختصاص می‌دهیم و این اتاق نیز به مجموعه اتاق‌های ما اضافه می‌شود.**

اگر n تعداد درس‌ها باشد، کوچک‌ترین n که الگوریتم فوق جواب بهینه تولید نمی‌کند، کدام است؟ (مهندسی کامپیوتر ـ دکتری ۹۵)

۱) 2 ۲) 3 ۳) 4 ۴) این الگوریتم به‌ازای هر n همیشه جواب بهینه تولید می‌کند.

پاسخ: گزینه «۳» به ازای $n=2$ و $n=3$ روش بیان شده حل بهینه را ایجاد می‌کند. اما برای $n=4$ اگر بازه‌های زیر را به عنوان زمان‌های مورد نیاز کلاس‌ها در نظر بگیرید، روش بیان شده حل بهینه را ایجاد نمی‌کند: $[2,4],[3,5],[6,7],[4,8]$

مثال۱۰: **دو پردازنده‌ی مشابه داریم و n عدد کار t_1 تا t_n که زمان انجام کار i ام روی هر کدام از این پردازنده‌ها برابر d_i است. می‌خواهیم این کارها را طوری زمان‌بندی کنیم که:**

حالت۱) متوسط زمان پاسخ کارها کمینه شود.

حالت۲) آخرین زمانی که همه پردازنده‌ها بیکار می‌شوند، کمینه شود.

زمان پاسخ یک کار زمانی است که آن کار از یکی از پردازنده‌ها خارج شود. وضعیت گزاره‌های زیر کدام است؟

الف) برای حالت ۱ یک الگوریتم چندجمله‌ای حریصانه وجود دارد.

ب) برای حالت ۲ یک الگوریتم چندجمله‌ای حریصانه وجود دارد. (مهندسی کامپیوتر، هوش مصنوعی ـ سراسری ۹۵)

۱) الف: درست، ب: درست. ۲) الف: نادرست، ب: درست.
۳) الف: درست، ب: نادرست. ۴) الف: نادرست، ب: نادرست.

پاسخ: گزینه «۳» برای کمینه شدن متوسط زمان پاسخ کارها کافی است کارها به ترتیب زمان مورد نیازشان از کم به زیاد مرتب شوند، دو کار اول به پردازنده‌ها داده شود و هر پردازنده پس از اتمام کار کنونی‌اش، کار جدید را با توجه به ترتیب اعمال شده دریافت کند. مسأله مرتب‌سازی نیز یک مسأله با راه‌حل چندجمله‌ای است.

برای کمینه کردن اختلاف زمان پایان کار دو پردازنده باید مجموعه $D=\{d_1,d_2,\cdots,d_n\}$ را به دو زیرمجموعه مانند D_1 و D_2 افراز کنیم، به‌طوری که اختلاف مجموع مقادیر اعضای دو مجموعه D_1 و D_2 کمینه شود. این مسأله راه‌حل چندجمله‌ای ندارد.

مثال۱۱: **n بازه‌ی $[l_i,u_i]$ برای $i=1..n$ داده شده‌اند. می‌خواهیم مجموعه‌ای از بازه‌های دوبه‌دو ناهم‌پوشان را با بیش‌ترین مجموع طول پیدا کنیم. برای این کار الگوریتم زیر را اجرا می‌کنیم.**

هر بار یکی از بازه‌ها را طبق یک ترتیب مشخص انتخاب کن، بازه‌هایی را که با این بازه هم‌پوشانی دارند حذف و این کار را تکرار کن.

این الگوریتم برای کدام‌یک از ترتیب‌های زیر همیشه درست کار می‌کند؟ (مهندسی کامپیوتر، نرم‌افزار ـ دکتری ۹۳)

۱) به ترتیب l_i ها ۲) به ترتیب u_i ها
۳) به ترتیب طول بازه‌ها ۴) هیچ‌کدام از ترتیب‌های بالا درست نیست.

پاسخ: گزینه «۴» مسأله مورد نظر دارای راه‌حل حریصانه نمی‌باشد.

مثال۱۲: می‌خواهیم n پردازه را بر روی یک پردازنده اجرا کنیم. اجرای پردازه‌ی iام p_i ثانیه طول می‌کشد و اجرای آن باید حداکثر تا زمان d_i به پایان برسد؛ در غیر این صورت، باید به میزان $t_i - d_i$ جریمه دیرکرد پرداخت شود که t_i زمان اتمام اجرای پردازه iام است. هدف پیدا کردن الگوریتمی برای زمان‌بندی پردازه‌ها است که مجموع جریمه‌ی دیرکردها کمینه شود. اجرای پردازه‌ها براساس کدام‌یک از ترتیب‌های زیر مجموع جریمه دیرکردها را کمینه می‌کند؟ (مهندسی کامپیوتر، هوش مصنوعی ـ سراسری ۹۳)

۱) به ترتیب غیرنزولی d_iها   ۲) به ترتیب غیرنزولی p_iها   ۳) به ترتیب غیرنزولی $d_i - p_i$ها   ۴) هیچ‌کدام

پاسخ: گزینه «۴» هیچ‌یک از راه‌کارهای بیان شده صحیح نیست.

برای گزینه اول می‌توان مثال نقض زیر را در نظر گرفت:

$p_1=4, d_1=5$
$p_2=3, d_2=4$

برای رد گزینه دوم می‌توان مثال زیر را در نظر گرفت:

$p_1=4, d_1=5$
$p_2=3, d_2=14$

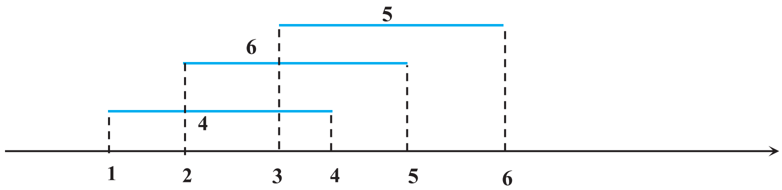
برای گزینه سوم می‌توان مثال نقض زیر را در نظر گرفت:

$p_1=8, d_1=8$
$p_2=1, d_2=2$

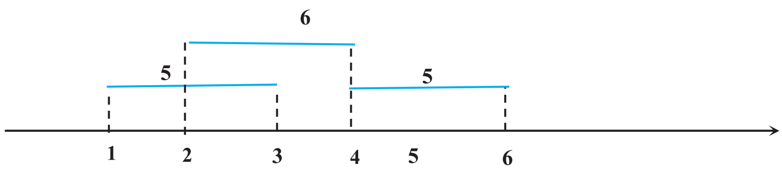
مثال۱۳: n پردازه داریم که به هرکدام یک زمان شروع، یک زمان پایان و یک ارزش نسبت داده شده است. پردازنده‌ای را نیز در اختیار داریم که در هر لحظه می‌تواند یک پردازه را اجرا کند. می‌خواهیم تعدادی پردازه با بیش‌ترین سود (مجموع ارزش) را انتخاب کنیم که بازه‌ی اجرای آن‌ها با هم هم‌پوشانی نداشته باشند. برای این‌کار، از یک الگوریتم حریصانه استفاده می‌کنیم که براساس «اولویت» پردازه‌ها کار می‌کند. در هر گام پردازه‌ی با بیش‌ترین اولویت مورد بررسی قرار می‌گیرد. اگر این پردازه با پردازه‌های انتخاب شده هم‌پوشانی نداشته باشد، انتخاب وگرنه از آن صرفنظر می‌شود. با کدام‌یک از اولویت‌های زیر سود بیشینه به دست می‌آید؟ (مهندسی فناوری اطلاعات (IT) ـ سراسری ۹۲)

۱) زمان شروع پردازه‌ها   ۲) زمان پایان پردازه‌ها   ۳) ارزش پردازه‌ها   ۴) هیچ‌کدام

پاسخ: گزینه «۴» مثال زیر یک مثال نقض برای گزینه‌های (۱) و (۲) می‌باشد (ارزش هر پردازه بر روی آن نوشته شده است).



مثال زیر یک مثال نقض برای گزینه (۳) می‌باشد.



مثال۱۴: n کیسه شن با حجم‌های v_1 تا v_n $(\forall i, 0 < v_i < 1)$ ولی نه لزوماً مرتب) داریم، می‌خواهیم آن‌ها را در جعبه‌هایی به حجم یک قرار داده و بسته‌بندی کنیم. برای این کار الگوریتم زیر پیشنهاد می‌شود:

ابتدا کیسه‌ها را به همان ترتیب می‌چینیم. سپس یک جعبه برداشته و کیسه‌ی اول، دوم و ... (تا جایی که می‌شود) را در آن قرار می‌دهیم. در صورتی که کیسه‌ی i اُم دیگر در جعبه جا نشد، یک جعبه جدید برداشته و کیسه i، $i+1$ و ...(تا جایی که جا بشود) را در آن قرار می‌دهیم.

اگر تعداد جعبه‌های استفاده شده در این روش M_a و تعداد جعبه‌های استفاده شده در بسته‌بندی کمینه (که کم‌ترین تعداد جعبه را استفاده کند) M_b باشد، در آن صورت کدام یک از گزاره‌های زیر درست است؟ (مهندسی نرم‌افزار ـ دکتری ۹۱)

۱) همواره $M_a > M_b$ ۲) همواره $M_a = M_b$ ۳) همواره $M_a < 2 \times M_b$ ۴) هیچ‌کدام

پاسخ: گزینه «۳» فرض کنید کیسه‌های زیر به ترتیب از چپ به راست داده شده است: $\frac{1}{3}, \frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}$

در این حالت روش a (روش حریصانه بیان شده در صورت تست) و روش b (روش بهینه) به شکل زیر عمل می‌کنند:

روش a: $\underbrace{\frac{1}{3},\frac{1}{2}}_{\text{جعبه } 4},\underbrace{\frac{1}{2},\frac{1}{3}}_{\text{جعبه } 2},\underbrace{\frac{1}{3}}_{\text{جعبه } 3}$

روش b: $\underbrace{\frac{1}{3},\frac{1}{3},\frac{1}{3}}_{\text{جعبه } 1},\underbrace{\frac{1}{2},\frac{1}{2}}_{\text{جعبه } 2}$

بنابراین $M_a=3$ و $M_b=2$ و $M_a>M_b$. حال اگر ترتیب داده شده را به صورت زیر در نظر بگیریم:

$$\frac{1}{3},\frac{1}{3},\frac{1}{3},\frac{1}{2},\frac{1}{2}$$

آنگاه $M_a=M_b=2$ خواهد بود و بنابراین گزینه‌های اول و دوم نادرست می‌باشند. اما در حالت کلی اگر جعبه‌های تخصیص داده روش a را به صورت $M_1M_2\cdots M_a$ در نظر بگیریم، آنگاه میزان حجم هدر رفته در هر دو جعبه متوالی کمتر از یک خواهد بود، زیرا حاصل‌جمع کیسه‌های قرار داده شده در دو جعبه بیشتر از یک می‌باشد (اگر کمتر از یک باشد همگی در جعبه اول قرار می‌گیرند) و مجموع ظرفیت دو جعبه 2 می‌باشد. در نتیجه حداکثر میزان فضای هدر رفته توسط روش a از $\frac{M_a}{2}$ کمتر می‌باشد.

مثال ۱۵: n بازه‌ی $I_i=(x_i,y_i)$ که $1\le i\le n$ داده شده‌اند. هر بازه نشان‌دهنده زمان یک کلاس درس است و هر کلاس به صورت مستقل احتیاج به یک اتاق دارد. می‌خواهیم کم‌ترین تعداد اتاق‌های لازم را پیدا کنیم تا بتوان بدون تداخل زمانی، به همه کلاس‌ها اتاق تخصیص داد. سریع‌ترین الگوریتم برای یافتن این تعداد اتاق کمینه، از چه مرتبه زمانی است؟ (الگوریتم و محاسبات ـ دکتری ۹۱)

۱) $O(n\lg n)$

۲) $O(n^2)$

۳) $O(n^3)$

۴) راه‌حل چندجمله‌ای ندارد.

☑ **پاسخ:** گزینه «۱» مقادیر ابتدا و انتهای بازه‌ها را در نظر می‌گیریم و در آرایه 2n تایی قرار می‌دهیم، سپس آرایه را مرتب می‌کنیم $O(n\log n)$. متغیر S را صفر در نظر می‌گیریم: $(S=0)$ روی آرایه 2n تایی مرتب جلو می‌رویم. به ازای هر ابتدای بازه، S را یک واحد اضافه و به ازای هر انتهای بازه، S را یک واحد کم می‌کنیم. بیشترین مقدار S، تعداد کلاس‌های مورد نیاز را نشان می‌دهد.

زمانی که با شروع یک بازه، S را یک واحد اضافه می‌کنیم معادل اختصاص دادن یک کلاس به آن بازه است. بیشترین مقدار S، زمانی را نشان می‌دهد که S تا بازه هم‌پوشانی دارند و کمتر از آن تعداد نمی‌توان کلاس داشت.

مثال ۱۶: هر یک از کارهای زیر در یک واحد زمان قابل اجرا است. هر یک از این کارها دارای یک موعد خاتمه (Deadline) است و در صورتی که بعد از موعد خاتمه انجام شود، مشمول یک جریمه (Penalty) خواهد شد. اگر این کارها را برای اجرا و رسیدن به کم‌ترین جریمه زمان‌بندی کنیم، مقدار جریمه چقدر است؟ (مهندسی IT ـ سراسری ۸۹)

Work :	W1	W2	W3	W4	W5	W6	W7
Deadline	7	2	3	3	2	5	1
Penalty	10	40	50	60	70	30	20

۱) 30

۲) 40

۳) 50

۴) 60

☑ **پاسخ:** گزینه «۴» مسأله طرح شده، شبیه به مسأله زمان‌بندی با مهلت معین است که به جای سود (profit) از جریمه (penalty) استفاده شده است. با توجه به مطلب مطرح شده در متن درس، روش حریصانه (greedy) بهترین راه حل را برای این مسأله پیدا می‌کند. بنابراین کافیست کارها را براساس جریمه مرتب کنیم و سپس آن‌ها را انتخاب نماییم که ترتیب زیر ایجاد می‌گردد:

$$w_5,w_4,w_3,\underset{\downarrow}{w_2},w_6,\underset{\downarrow}{w_7},w_1$$

جریمه = 20   جریمه = 40 $\Rightarrow$ کل جریمه $=60$

مثال ۱۷: در مسأله زمان‌بندی کارها، n پردازنده در اختیار داریم و باید m کار را پردازش کنیم (n<<m). با فرض اینکه همه کارها مشابه بوده و زمان پردازش یک کار برای پردازنده i برابر i واحد زمان باشد و تخصیص کار به‌صورت حریصانه با اولویت کندترین پردازنده باشد (پردازنده‌ای که زمان بیشتری نیاز دارد اولویت دارد)، بیشینه زمان پردازش این کارها چه مقدار خواهد بود؟

۱) $\dfrac{m+n-1}{\sum_{i=1}^{n}\frac{1}{i}}$

۲) $\dfrac{m+\frac{1}{n}}{\sum_{i=1}^{n}\frac{1}{i}}$

۳) $\dfrac{m-1+\frac{1}{n}}{\sum_{i=1}^{n}\frac{1}{i}}+n-1$

۴) $\dfrac{m-1}{\sum_{i=1}^{n}\frac{1}{i}}+n$

مدرسان شریف

فصل دوازدهم

«الگوریتم‌های پیمایش گراف»

مقدمه

در فصل پنجم با مفاهیم اولیه گراف آشنا شدیم. در این فصل، ابتدا به بیان الگوریتم‌های جست‌وجوی عمقی (DFS) و جست‌وجوی سطحی (BFS) در گراف، پرداخته شده است که در بسیاری دیگر از الگوریتم‌های گراف استفاده می‌شوند. در ادامه فصل الگوریتم‌های مربوط به مرتب‌سازی توپولوژیک، مؤلفه‌های همبند قوی که کاربردهایی از الگوریتم‌های پیمایش گراف هستند، مورد مطالعه و بررسی قرار گرفته‌اند.

درسنامه: پیمایش گراف

جست‌وجوی سطحی (BFS) در گراف

در الگوریتم جست‌وجوی سطحی، یک رأس خاص به عنوان رأس منبع (Source) یا رأس آغازین در نظر گرفته می‌شود و هدف ملاقات تمام رئوسی است که از رأس منبع قابل دسترسی می‌باشند. در این جست‌وجو ترتیب ملاقات رئوس با توجه به فاصله آن‌ها از رأس منبع می‌باشد (رئوس گراف براساس تعداد یال‌هایی که با رأس منبع فاصله دارند ملاقات می‌شوند). ایده موجود در الگوریتم جست‌وجوی سطحی در الگوریتم‌های دیگری مانند الگوریتم دایکسترا و الگوریتم پریم (فصل سوم) مورد استفاده قرار گرفته است.

در الگوریتم BFS هر رأس گراف می‌تواند دارای سه رنگ سفید، خاکستری و سیاه باشد که این رنگ‌آمیزی برای پیمایش صحیح رئوس گراف و همچنین تشخیص پایان الگوریتم استفاده می‌گردد. مفهوم رنگ‌های انتساب داده شده به صورت زیر است:

سفید: نشان‌دهنده این است که رأس مورد نظر کشف (discover) یا ملاقات نشده است.

خاکستری: نشان‌دهنده این است که رأس مورد نظر کشف شده است و ممکن است همسایه‌ای با رنگ سفید داشته باشد (بنابراین ممکن است همسایه‌ای داشته باشد که هنوز کشف نشده است).

سیاه: نشان‌دهنده این است که رأس مورد نظر کشف شده و هیچ همسایه سفیدی ندارد (بنابراین تمام همسایه‌های آن نیز کشف شده‌اند و نمی‌توان از این رأس به رئوس جدید دیگری رسید).

در نهایت، یال‌های پیمایش شده در الگوریتم BFS درختی را ایجاد می‌کنند که ریشه آن رأس منبع می‌باشد و فاصله هر رأس موجود در این درخت از ریشه نشان دهنده طول کوتاه‌ترین مسیر از رأس منبع تا رأس مورد نظر می‌باشد.

الگوریتم BFS در زیر آمده است. در این الگوریتم از صف Q که دارای خاصیت FIFO می‌باشد، برای ذخیره‌سازی رئوس خاکستری گراف (به ترتیب ملاقات) استفاده گردیده است. همچنین برای هر رأس دلخواه مانند u سه مشخصه color، d و π در نظر گرفته شده است که هرکدام یکی از ویژگی‌های مورد نیاز برای رأس u را به صورت زیر نشان می‌دهد:

مشخصه u.color نشان‌دهنده رنگ رأس u می‌باشد (که سه حالت WHITE، GRAY و BLACK را می‌پذیرد).

مشخصه u.d نشان‌دهنده فاصله رأس u از رأس منبع (تعداد یال‌های موجود در مسیر) می‌باشد.

مشخصه $u.\pi$ نشان‌دهنده رأس والد (گره ماقبل) برای رأس u در مسیر موجود از رأس منبع به رأس u می‌باشد (این ویژگی برای حالتی که u همان رأس منبع باشد و یا u هنوز کشف نشده باشد برابر NIL می‌باشد).

BFS (G , s)

```
1   for each vertex u ∈ G.V − {s}
2       u.color WHITE
3       u.d = ∞
4       u.π = NIL
5   s.color = GRAY
6   s.d = 0
7   s.π = NIL
8   Q = ∅
9   ENQUEUE (Q , s)
10  while Q ≠ ∅
11      u = DEQUEUE (Q)
12      for each v ∈ G.Adj[u]
13          if v .color == WHITE
14              v .color = GRAY
15              v .d = u.d + 1
16              v.π = u
17              ENQUEUE (Q , v )
18      u.color = BLACK
```

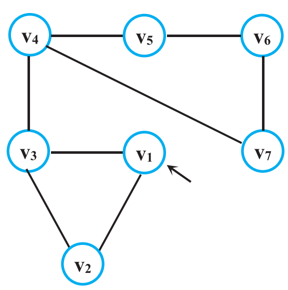
همان‌گونه که ملاحظه می‌گردد در ابتدا رنگ تمام رئوس (به جز رأس منبع) سفید در نظر گرفته می‌شود، فاصله تمام رئوس از رأس منبع برابر بی‌نهایت قرار داده می‌شود و والد تمام رئوس با NIL مقداردهی می‌گردد.

مرتبه زمانی حلقه for موجود در خط 1 برابر $\theta(V)$ می‌باشد (زیرا ویژگی‌های مربوط به تمام رئوس به غیر از رأس منبع مقداردهی می‌شوند) و حلقه while به‌ازای تمام رئوسی که از رأس منبع قابل دسترسی می‌باشند بررسی می‌گردد که اگر گراف همبند باشد زمان آن $\theta(E)$ خواهد بود. بنابراین زمان کل الگوریتم BFS برابر با $O(V+E)$ می‌باشد. دقت نمایید که اگر گراف مورد نظر به وسیله ماتریس مجاورتی نمایش داده شود زمان $O(V^2)$ خواهد بود.

تذکر: الگوریتم BFS را می‌توان روی گراف‌های جهت‌دار و بدون جهت اجرا نمود.

نکته۱: در الگوریتم BFS هر رأس تنها یک بار ملاقات می‌گردد و بنابراین هر رأس فقط می‌تواند یک والد داشته باشد.

گراف مقابل را در نظر بگیرید:



اگر رأس v_1 به عنوان رأس منبع در نظر گرفته شود، مراحل انجام الگوریتم پیمایش BFS به صورت زیر خواهد بود:

ابتدا ویژگی‌های مربوط به هر رأس مقداردهی اولیه می‌گردند و رأس منبع به عنوان یک رأس خاکستری در صف Q قرار می‌گیرد.

Q

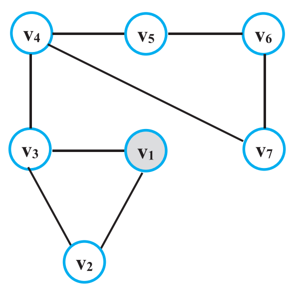
v_1

color

v_1	v_2	v_3	v_4	v_5	v_6	v_7
G	W	W	W	W	W	W

π

v_1	v_2	v_3	v_4	v_5	v_6	v_7
NIL	NIL	NIL	NIL	NIL	NIL	NIL



d

v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	∞	∞	∞	∞	∞	∞

همانگونه که مشخص است رنگ رأس v_1 (رأس منبع) خاکستری و رنگ سایر رئوس سفید است و همچنین فاصله رأس v_1 از منبع برابر صفر و فاصله سایر رئوس برابر بی‌نهایت در نظر گرفته شده است. سپس رأس موجود در ابتدای صف Q یعنی، v_1 حذف می‌گردد و تمام همسایه‌های سفید آن (یعنی v_2 و v_3) به صف اضافه می‌شوند. ویژگی‌های آن‌ها به صورت زیر تغییر می‌کند:

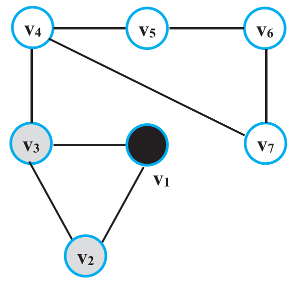
ویژگی color برابر GRAY قرار داده می‌شود. ویژگی π برابر v_1 قرار می‌گیرد (زیرا رأس ماقبل آن v_1 می‌باشد).

ویژگی d برابر 1 قرار می‌گیرد (یک واحد بیشتر از فاصله والد).

سپس با توجه به این که تمام همسایه‌های v_1 ملاقات شده‌اند، رنگ v_1 سیاه خواهد شد.

Q

v_2	v_3



color

v_1	v_2	v_3	v_4	v_5	v_6	v_7
B	G	G	W	W	W	W

π

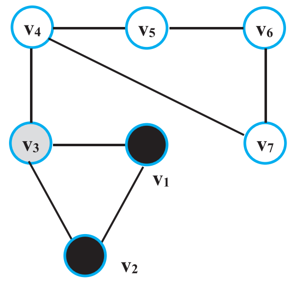
v_1	v_2	v_3	v_4	v_5	v_6	v_7
NIL	v_1	v_1	NIL	NIL	NIL	NIL

d

v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	1	1	∞	∞	∞	∞

در مرحله بعد، رأس موجود در ابتدای صف Q، یعنی v_2 حذف می‌گردد و با توجه به این که v_2 هیچ همسایه‌ای با رنگ سفید ندارد، بنابراین هیچ رأسی به صف Q اضافه نمی‌گردد و رنگ رأس v_2 به سیاه تغییر می‌کند و غیر از این، ویژگی‌های هیچ رأسی تغییر نمی‌کند. صف Q به صورت زیر خواهد بود:

Q: v_3



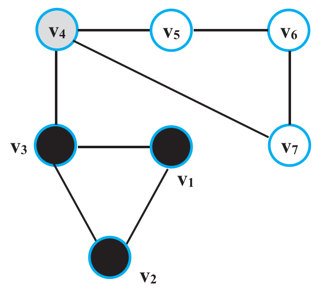
color

v_1	v_2	v_3	v_4	v_5	v_6	v_7
B	B	G	W	W	W	W

در این مرحله رأس V_3 از صف حذف می‌شود و تنها همسایه سفید آن، یعنی رأس V_4 به صف اضافه می‌گردد. فاصله V_4 برابر 2 قرار می‌گیرد و والد آن به V_3 و رنگ آن به خاکستری تغییر می‌یابد و همچنین رنگ رأس V_3 به سیاه تغییر داده می‌شود:

Q

v_4



color

v_1	v_2	v_3	v_4	v_5	v_6	v_7
B	B	B	G	W	W	W

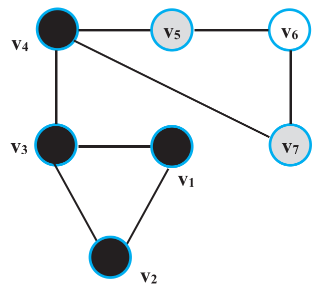
π

v_1	v_2	v_3	v_4	v_5	v_6	v_7
NIL	v_1	v_1	v_3	NIL	NIL	NIL

d

v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	1	1	2	∞	∞	∞

حال رأس v_4 از صف حذف می‌گردد و تمام همسایه‌های سفید آن (یعنی v_7, v_5) به صف اضافه می‌شوند، در این حالت رنگ رأس‌های v_5 و v_7 برابر خاکستری، فاصله آن‌ها برابر 3 و والد آن‌ها برابر v_4 قرار داده می‌شود و سپس رنگ رأس v_4 سیاه می‌گردد:



Q

v_5	v_7

color

v_1	v_2	v_3	v_4	v_5	v_6	v_7
B	B	B	B	G	W	G

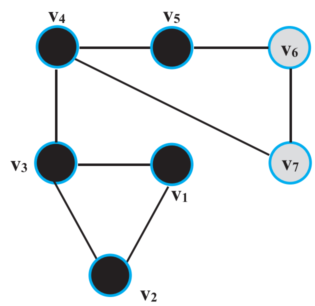
π

v_1	v_2	v_3	v_4	v_5	v_6	v_7
NIL	v_1	v_1	v_3	v_4	NIL	v_4

d

v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	1	1	2	3	∞	3

سپس رأس v_5 از صف حذف می‌گردد و تنها همسایه سفید آن یعنی v_6 به صف اضافه می‌گردد و ویژگی رئوس v_5, v_6 به صورت زیر تغییر می‌کند.



Q

v_7	v_6

color

v_1	v_2	v_3	v_4	v_5	v_6	v_7
B	B	B	B	B	G	G

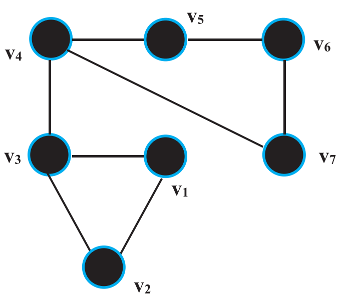
π

v_1	v_2	v_3	v_4	v_5	v_6	v_7
NIL	v_1	v_1	v_3	v_4	v_5	v_4

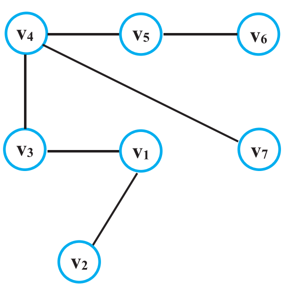
d

v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	1	1	2	3	4	3

سپس رأس v_7 از صف حذف می‌گردد و با توجه به این که هیچ همسایه سفیدی ندارد، رنگ آن به سیاه تغییر می‌یابد و همین اتفاق در مورد رأس v_6 نیز رخ خواهد داد و در نهایت، رنگ تمام رئوس سیاه خواهد بود و صف نیز خالی می‌گردد و الگوریتم پایان می‌پذیرد.



ویژگی d برای هر رأس، نشان‌دهنده تعداد یال‌های موجود در کوتاه‌ترین مسیر از منبع تا رأس مورد نظر می‌باشد که مسیر را می‌توان با استفاده از ویژگی π به دست آمده برای رئوس گراف، تعیین نمود. در نهایت درخت ایجاد شده توسط الگوریتم BFS در این مثال به صورت مقابل می‌باشد.



مدرسان شریف

فصل چهاردهم

«مرتب‌سازی‌های مقایسه‌ای»

مقدمه

مسأله مرتب‌سازی یکی از مسائل بسیار پرکاربرد در دسته وسیعی از الگوریتم‌ها می‌باشد و بهبود در الگوریتم‌های مرتب‌سازی می‌تواند منجر به بهبود بسیاری از الگوریتم‌ها گردد. مسأله مرتب‌سازی در حالت کلی به صورت زیر مطرح می‌گردد:

مسأله مرتب‌سازی (Sort)
ورودی: لیست n عنصری $[a_1, a_2, \cdots, a_n]$ از کلیدها
خروجی: لیست $[b_1, b_2, \cdots, b_n]$ که از تغییر ترتیب عناصر لیست ورودی به دست می‌آید و در آن داریم: $b_i \le b_{i+1} : i = 1, 2, \cdots, n-1$

قبل از شروع معرفی مرتب‌سازی‌های جدید، ابتدا نشان می‌دهیم که کران پایین برای مسأله مرتب‌سازی در الگوریتم‌هایی که عمل مرتب‌سازی را با مقایسه عناصر انجام می‌دهند، $\Omega(n \log n)$ است.

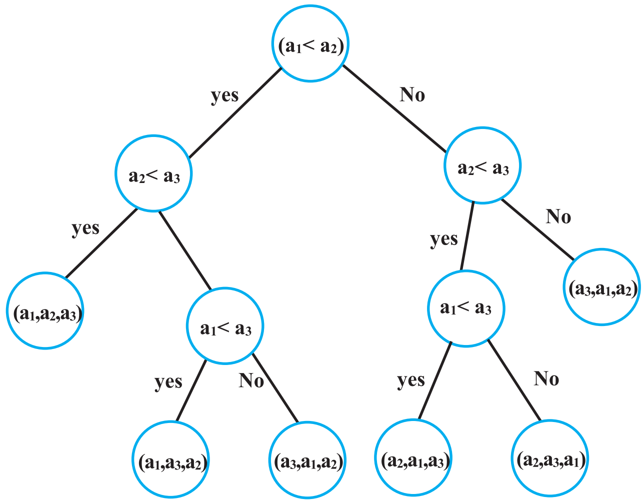
درسنامه: مرتب‌سازی با مقایسه عناصر

درخت تصمیم مسأله مرتب‌سازی

❖ **تعریف:** یک درخت تصمیم‌گیری برای دنبالهٔ $a_1 \ldots\ldots a_n$ درختی است که در هر گره داخلی آن یک مقایسه صورت می‌گیرد و هر برگ حاوی یک جایگشت از دنبالهٔ $a_1 \ldots\ldots a_n$ است.

نکته۱: با توجه به تعداد جایگشت‌های ممکن یک لیست n عنصری می‌توان گفت که درخت تصمیم‌گیری برای مرتب‌سازی دنبالهٔ $a_1 \ldots\ldots a_n$ دارای $n!$ برگ است.

درخت تصمیم‌گیری برای مرتب‌سازی دنبالهٔ a_1, a_2, a_3 به صورت زیر خواهد بود.



بنابراین با توجه به این که درخت تصمیم‌گیری برای مرتب‌سازی n کلید متمایز دارای $n!$ برگ است و باید برای مرتب نمودن لیست ورودی یکی از مسیرهای موجود از ریشه به برگ را طی کنیم، در نتیجه در بدترین حالت، باید عمق درخت یعنی $\log_2(n!)$ را طی کنیم.

نکته۲: طبق تقریب استرلینگ رابطه $\log(n!) \in \theta(n\log n)$ برقرار است.

با توجه به بحث بالا (که در بدترین حالت باید $\log n!$ مقایسه انجام پذیرد) می‌توان گفت در بدترین حالت هر الگوریتم مرتب‌سازی که با مقایسه عناصر، عمل مرتب‌سازی را انجام می‌دهد، دارای زمان $\Omega(n\log n)$ خواهد بود.

مرتب‌سازی درجی (insertion sort)

این مرتب‌سازی از $n-1$ مرحله تشکیل شده است، که در مرحلهٔ iام عناصر اول تا $i-1$ مرتب می‌باشند و عنصر a_i در بین عناصر مرتب قبلی که یک جایگشت از $a_1.......a_{i-1}$ است طوری جای می‌گیرد که i عنصر ابتدایی مرتب باشند. بنابراین مراحل کار به صورت زیر خواهد بود:

a_1 به تنهایی مرتب است.

مرحلهٔ اول: a_2 با a_1 مقایسه می‌شود. اگر $a_2 < a_1$، آنگاه a_2 را قبل از a_1 درج می‌کنیم و در غیر این صورت هیچ تغییری صورت نمی‌گیرد و لیست a_1a_2 مرتب است.

مرحلهٔ دوم: a_3 را با عنصر قبلی مقایسه می‌کنیم. اگر کوچکتر از عنصر دوم بود آن را با عنصر اول نیز مقایسه می‌کنیم، اگر از عنصر اول نیز کوچکتر بود آن را در مکان اول قرار می‌دهیم وگرنه آن را در بین عناصر اول و دوم درج می‌نماییم. اگر از عنصر دوم کوچکتر نبود هیچ تغییری ایجاد نمی‌کنیم.

⋮

مرحلهٔ n − ۱: عنصر a_n را با عناصر قبلی مقایسه می‌کنیم و اولین عنصری را می‌یابیم که عنصر a_n از آن عنصر بزرگتر یا مساوی باشد. سپس a_n را در مکان بعد از آن عنصر درج می‌نماییم. الگوریتم مرتب‌سازی درجی در زیر آمده است:

```
Insertion sort (A [1,...,n])
for (i = 2 to n)   {
    temp = A[i]
    j = i – 1
    while (j > 0 & & A[j] > temp)   {
        A[j + 1] = A[j]
        j = j – 1 }// end of while
    A[j + 1] = temp }// end of for
```

به نکات زیر در مورد مرتب‌سازی درجی دقت کنید:

۱ـ مرتب‌سازی درجی یک مرتب‌سازی درجا است (زیرا فقط از یک مقدار ثابت حافظه به صورت کمکی استفاده می‌کند).

۲ـ مرتب‌سازی درجی یک مرتب‌سازی پایدار است.

۳ـ تعداد مقایسه‌های مرتب‌سازی درجی در بدترین حالت برابر $\frac{n(n-1)}{2}$ و در حالت متوسط برابر با $\frac{n(n-1)}{4}$ خواهد بود.

۴ـ مرتب‌سازی درجی برای مرتب‌کردن لیست‌های کوچک و یا لیست‌هایی که به صورت تقریبی مرتب می‌باشند، بهترین مرتب‌سازی محسوب می‌شود.

۵ـ بدترین ورودی برای مرتب‌سازی درجی ورودی است که به ترتیب عکس مرتب شده باشد.

۶ـ بهترین ورودی برای مرتب‌سازی درجی ورودی است که از قبل مرتب باشد. در این حالت فقط n مقایسه انجام می‌شود و هیچ جابجایی صورت نمی‌پذیرد.

مراحل مرتب‌سازی لیست $77,33,44,11,88,22,66,55$ را با استفاده از مرتب‌سازی درجی آمده است:

مرحله اول: 33 با 77 مقایسه می‌شود و با توجه به این که $33 < 77$ بنابراین، لیست به صورت زیر در می‌آید:

$,77,33,44,11,88,22,66,55$

مرحله دوم: 44 با 77 مقایسه می‌شود و با توجه به این که $44 < 77$ بنابراین، 44 با 33 مقایسه می‌شود و با توجه به این که $33 < 44$ در نتیجه 44 بین 77,33 درج می‌شود.

$33,\ ,77,44,11,88,22,66,55$

مرحله سوم: عدد چهارم یعنی، عدد ۱۱ در بین سه عدد قبلی در مکان درست خود یعنی، در مکان اول درج می‌شود.

$,33,44,77,11,88,22,66,55$

مرحله چهارم: عدد پنجم یعنی، عدد ۸۸ در بین چهار عدد قبلی در جای درست خود یعنی، در مکان پنجم قرار می‌گیرد:

$11,33,44,77,88,22,66,55$

مرحله پنجم: عدد ششم یعنی، عدد ۲۲ در بین پنج عدد قبلی در مکان دوم درج می‌گردد:

$11,\ ,33,44,77,88,22,66,55$

مرحله ششم: عدد 66 در جای مناسب در بین شش عدد قبلی یعنی، در مکان پنجم درج می‌شود:

11,22,33,44, ,77,88, 66 ,55

مرحله هفتم: (مرحلهٔ آخر) عدد 55 در جای مناسب در بین هفت عدد ابتدایی یعنی، در مکان پنجم درج می‌گردد:

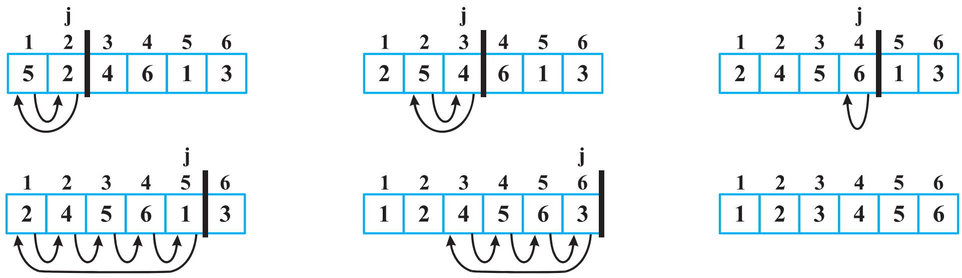
11,22,33,44, ,66,77,88, 55

مثال ۱: **تعداد انتساب‌هایی که برای مرتب نمودن لیست زیر توسط مرتب‌سازی درجی انجام می‌شود، چقدر است؟**

5	2	4	6	1	3

۱) 9
۲) 10
۳) 14
۴) 15

پاسخ: گزینه «۳» نحوه انجام مرتب‌سازی و انتساب‌ها به صورت زیر می‌باشد:



که مجموع تعداد انتساب‌ها برابر 14 است.

مرتب‌سازی حبابی (Bubble sort)

در این مرتب‌سازی با شروع از ابتدای آرایه، عناصر مجاور دوبه‌دو با یکدیگر مقایسه می‌شوند و در صورتی که یک عنصر از عنصر بعد از خود بزرگتر باشد، دو عنصر مورد مقایسه با یکدیگر تعویض می‌شوند.

اگر به همین ترتیب عمل کنیم، در پایان مرحله اول بزرگترین عنصر در مکان آخر قرار می‌گیرد. سپس در مرحله دوم با انجام عملیاتی مشابه دومین عنصر بزرگ در مکان ماقبل آخر جای می‌گیرد و اگر به همین شکل کار را ادامه دهیم، در مرحله i ام، i امین بزرگترین عنصر در مکان درست خود قرار می‌گیرد.

تذکر ۱: در برخی مراجع الگوریتم مرتب‌سازی حبابی طوری نوشته می‌شود که در مرحله iام، iامین کوچکترین عنصر در جای خود قرار می‌گیرد.

الگوریتم مرتب‌سازی حبابی به صورت زیر است:

Bubble sort (A[1,…,n])

```
for (i = 1 to n - 1)
    for (j = 1 to n - i)
        if (A[j] > A[j + 1]
            A[j] ↔ A[j + 1]
```

فرض کنیم لیست ورودی 77,33,44,11,88,22,66,55 را داریم، مراحل انجام مرتب‌سازی حبابی به صورت زیر می‌باشد:

گذر اول:

۱) 77 را با 33 مقایسه می‌کنیم و با توجه به این‌که $77 > 33$ پس جای آنها را عوض می‌کنیم.

77 , 33 ,44,11,88,22,66,55

۲) 77 را با 44 مقایسه می‌کنیم و با توجه به این‌که $77 > 44$ پس جای آنها را عوض می‌کنیم.

33, 77 , 44 ,11,88,22,66,55

۳) 77 را با 11 مقایسه می‌کنیم و با توجه به این‌که $77 > 11$ پس جای آنها را عوض می‌کنیم.

33,44, 77 , 11 ,88,22,66,55

۴) 77 را با 88 مقایسه می‌کنیم و با توجه به این‌که $77 < 88$ پس هیچ جابجایی صورت نمی‌گیرد.

33,44,11, 77 , 88 ,22,66,55

۵) 88 را با 22 مقایسه می‌کنیم و با توجه به این‌که $88 > 22$ پس جای آنها را عوض می‌کنیم.

33,44,11,77, 22 , 88 ,66,55

۶) 88 را با 66 مقایسه می‌کنیم و با توجه به این‌که $88 > 66$ پس جای آنها را عوض می‌کنیم.

33,44,11,77,22, 66 , 88 ,55

۷) 88 را با 55 را مقایسه می‌کنیم و با توجه به این‌که $88 > 55$ بنابراین جای آنها را عوض می‌کنیم.

$33,44,11,77,22,66,55,88$

بنابراین در انتهای گذر اول لیست به شکل مقابل خواهد بود.

$33,44,11,77,22,66,55,88$

همان‌گونه که مشخص است، در پایان گذر اول بزرگترین عنصر در مکان آخر قرار گرفته است.

گذر دوم: اگر این گذر را نیز مانند گذر اول انجام دهیم پس از اتمام آن، دومین بزرگترین عنصر یعنی، 77 در مکان ماقبل آخر قرار می‌گیرد. بنابراین در پایان گذر دوم لیست به شکل زیر خواهد بود.

$33,11,44,22,66,55,77,88$

گذر سوم: در پایان گذر سوم لیست به صورت زیر خواهد بود (66 در جای درست خود قرار می‌گیرد).

$11,33,22,44,55,66,77,88$

گذر چهارم: در پایان گذر چهارم لیست به صورت زیر می‌باشد.

$11,22,33,44,55,66,77,88$

در گذرهای پنجم تا هشتم هیچ تغییر دیگری در لیست اتفاق نمی‌افتد.

نکته۳: زمان مرتب‌سازی حبابی در هر حالت $O(n^2)$ است.

همان گونه که در مثال۳ مشاهده شد اگر در یک گذر از مرتب‌سازی هیچ جابه‌جایی صورت نگیرد، آنگاه لیست مرتب است؛ یعنی گذرهای ششم، هفتم و هشتم در مثال ۳ مقایسه‌های اضافی انجام می‌دهند و هیچ جابه‌جایی صورت نمی‌دهند. بنابراین می‌توان الگوریتم مرتب‌سازی حبابی را با قرار دادن یک متغیر منطقی طوری اصلاح نمود که مقایسه‌های اضافی را انجام ندهد که به این الگوریتم، مرتب‌سازی حبابی اصلاح شده می‌گوییم.

به چند نکته در مورد مرتب‌سازی حبابی توجه کنید:

۱ـ زمان الگوریتم مرتب‌سازی حبابی اصلاح شده در بدترین حالت $\theta(n^2)$ و در بهترین حالت $\theta(n)$ است.

۲ـ بهترین ورودی برای الگوریتم مرتب‌سازی حبابی ورودی است که از قبل مرتب باشد.

۳ـ بدترین ورودی برای الگوریتم مرتب‌سازی حبابی ورودی است که به ترتیب عکس مرتب شده باشد.

۴ـ الگوریتم مرتب‌سازی حبابی از لحاظ زمانی الگوریتم مناسبی نیست و مزیت این الگوریتم ساده بودن آن است.

۵ـ مرتب‌سازی حبابی یک مرتب‌سازی درجا می‌باشد.

۶ـ مرتب‌سازی حبابی یک مرتب‌سازی متعادل است.

مرتب‌سازی انتخابی (selection sort)

مرتب‌سازی انتخابی برای مرتب کردن یک لیست ابتدا کوچکترین عنصر لیست را می‌یابد و آن را در مکان اول لیست قرار می‌دهد سپس، دومین کوچکترین عنصر لیست را می‌یابد و آن را در مکان دوم قرار می‌دهد. بنابراین با انجام $n-1$ مرحله تمام لیست مرتب خواهد شد. الگوریتم این مرتب‌سازی در زیر آمده:

Selection sort (A[1,…,n])

```
for i = 1 to n
    min = i
    for j = i + 1 to n
        if (A[j] < A[min])
            min = j
    A[i] ↔ A[min]
```

نکته۴: مرتب‌سازی انتخابی یک مرتب‌سازی درجا می‌باشد.

نکته۵: مرتب‌سازی انتخابی زمان $\theta(n^2)$ را برای مرتب‌سازی نیاز دارد.

فرض کنیم لیست ورودی $77,33,44,11,88,22,66,55$ داده شده، مراحل انجام مرتب‌سازی انتخابی در ادامه روی این لیست نشان داده شده است.

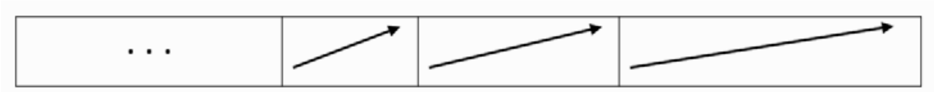
گذر اول: کوچکترین عنصر یعنی 11 به اولین مکان انتقال می‌یابد.

$77,33,44,11,88,22,66,55$

گذر دوم: دومین کوچکترین عنصر یعنی 22 به مکان دوم منتقل می‌شود.

$11,33,44,77,88,22,66,55$

مثال ۸۳: آرایه‌ای به طول n در اختیار داریم که عناصر یک‌سوم انتهایی آن به‌صورت صعودی مرتب هستند (این اعداد الزاماً از همه اعداد دوسوم ابتدایی بزرگتر نیستند). دوسوم ابتدایی این آرایه نیز خاصیت آرایه به طول n را دارد. یعنی یک‌سوم انتهایی آن به‌صورت صعودی مرتب است. این روند به‌صورت بازگشتی برای بخش‌های کوچکتر نیز صادق است. در این شرایط، مرتب‌سازی این آرایه از چه مرتبه‌ای خواهد بود؟ (شمایی از ترتیب عناصر آرایه در شکل زیر آمده است.)



۱) $\theta(n\log\log n)$ ۲) $\theta(n)$ ۳) $\theta(n\sqrt{\log n})$ ۴) $\theta(n\log n)$

پاسخ: گزینه «۲» می‌توان الگوریتم مرتب‌سازی ادغامی را به شکلی تغییر داد که در تابع ادغام، یک آرایه مرتب k عضوی را با یک آرایه مرتب 2k عضوی ادغام کند. پیچیدگی محاسباتی این عمل از مرتبه $\theta(n)$ خواهد بود. در نتیجه کافی است به‌صورت بازگشتی، دوسوم ابتدایی آرایه را مرتب کرده و با یک‌سوم انتهایی آرایه (که از قبل مرتب بوده است) ادغام نماییم. رابطه بازگشتی مربوط به پیچیدگی محاسباتی این الگوریتم به شکل زیر خواهد بود.

$$T(n)=T(\frac{2n}{3})+\theta(n)\in\theta(n)$$

خلاصه فصل چهاردهم

مطالب مرتبط با مفاهیم این فصل را می‌توان به صورت زیر در نظر گرفت:

مرتب‌سازی درجی

۱ـ مرتب‌سازی درجی یک مرتب‌سازی درجا است (زیرا فقط از یک مقدار ثابت حافظه به صورت کمکی استفاده می‌کند).

۲ـ مرتب‌سازی درجی یک مرتب‌سازی پایدار است.

۳ـ تعداد مقایسه‌های مرتب‌سازی درجی در بدترین حالت برابر $\frac{n(n-1)}{2}$ و در حالت متوسط برابر با $\frac{n(n-1)}{4}$ خواهد بود.

۴ـ مرتب‌سازی درجی برای مرتب‌کردن لیست‌های کوچک و یا لیست‌هایی که به صورت تقریبی مرتب می‌باشند، بهترین مرتب‌سازی محسوب می‌شود.

۵ـ بدترین ورودی برای مرتب‌سازی درجی ورودی است که به ترتیب عکس مرتب شده باشد.

۶ـ بهترین ورودی برای مرتب‌سازی درجی ورودی است که از قبل مرتب باشد، در این حالت فقط n مقایسه انجام می‌شود و هیچ جابه‌جایی صورت نمی‌پذیرد.

مرتب‌سازی حبابی

ویژگی‌های مهم مرتب‌سازی حبابی را می‌توان به صورت زیر در نظر گرفت:

۱ـ زمان مرتب‌سازی حبابی در هر حالت $O(n^2)$ است.

۲ـ زمان الگوریتم مرتب‌سازی حبابی اصلاح شده در بدترین حالت $\theta(n^2)$ و در بهترین حالت $\theta(n)$ است.

۳ـ بهترین ورودی برای الگوریتم مرتب‌سازی حبابی ورودی است که از قبل مرتب باشد.

۴ـ بدترین ورودی برای الگوریتم مرتب‌سازی حبابی ورودی است که به ترتیب عکس مرتب شده باشد.

۵ـ الگوریتم مرتب‌سازی حبابی از لحاظ زمانی الگوریتم مناسبی نیست و مزیت این الگوریتم ساده بودن آن است.

۶ـ مرتب‌سازی حبابی یک مرتب‌سازی درجا می‌باشد.

۷ـ مرتب‌سازی حبابی یک مرتب‌سازی متعادل است.

مرتب‌سازی انتخابی

۱ـ مرتب‌سازی انتخابی یک مرتب‌سازی درجا می‌باشد.

۲ـ مرتب‌سازی انتخابی در بدترین حالت زمان $\theta(n^2)$ را برای مرتب‌سازی نیاز دارد.

۳ـ مرتب‌سازی انتخابی روی انواع ورودی‌های مختلف دارای زمان $\theta(n^2)$ خواهد بود، حتی لیست مرتب.

مرتب‌سازی ادغامی

این مرتب‌سازی با استفاده از الگوریتم کمکی ادغام، لیست ورودی را مرتب می‌کند.

۱ـ زمان ادغام دو آرایه مرتب به طول m و n برابر $O(m+n)$ است.

۲ـ بهترین زمان ادغام دو آرایه مرتب به طول m و n برابر $\theta(\min(m,n))$ است.

۳ـ رابطه بازگشتی مربوط به مرتبه زمانی مرتب‌سازی ادغامی به صورت مقابل می‌باشد: $T(n)=2T(\frac{n}{2})+\theta(n)$

بنابراین: $T(n)\in\theta(n\log_2^n)$

۱ـ مرتب‌سازی ادغامی از فضای کمکی $\theta(n)$ استفاده می‌کند. ۲ـ مرتب‌سازی ادغامی پایدار (stable) است. ۳ـ مرتب‌سازی ادغامی درجا (inplace) نیست.

مرتب‌سازی سریع

این مرتب‌سازی با استفاده از الگوریتم افراز (partition)، به صورت بازگشتی روی لیست ورودی اجرا می‌شود و لیست را مرتب می‌کند.

۱ـ مرتبه زمانی الگوریتم افراز برابر $\theta(n)$ می‌باشد.

۲ـ اگر آرایه ورودی از قبل به صورت صعودی یا نزولی مرتب باشد، آنگاه بدترین حالت در الگوریتم مرتب‌سازی سریع رخ خواهد داد و رابطه بازگشتی مربوط به مرتبه زمانی الگوریتم در این حالت عبارت است از: $T(n)=T(n-1)+\theta(n)$

بنابراین: $T(n)=\theta(n^2)$

۳ـ بهترین حالت زمانی رخ می‌دهد که عنصر محوری در هر مرحله تقریباً میانه لیست باشد که در این حالت داریم: $T(n)\simeq 2T(\frac{n}{2})+\theta(n)$

بنابراین: $T(n)\in\theta(n\log_2^n)$

۴ـ مرتب‌سازی سریع پایدار نیست. مرتب‌سازی سریع یک الگوریتم مرتب‌سازی درجا است. اگر عنصر محوری به صورت تصادفی انتخاب شود، زمان میانگین مورد انتظار از مرتبه $\theta(n\log n)$ می‌باشد.

مرتب‌سازی درختی

در این روش ابتدا یک درخت BST با استفاده از کلیدهای داده شده ایجاد می‌شود و سپس با پیمایش میانوندی درخت موردنظر، لیست مرتب شده ایجاد می‌گردد. زمان مرتب‌سازی درختی به صورت زیر قابل محاسبه است:

زمان ساخت درخت + زمان پیمایش = زمان ساخت $+O(n)$

بنابراین اگر درخت ایجاد شده تقریباً متوازن باشد (که در حالت میانگین این اتفاق رخ می‌دهد)، آنگاه زمان کلی $O(n\log_2 n)$ خواهد بود. اما در بدترین حالت زمان ساخت یک درخت BST از مرتبه $O(n^2)$ می‌باشد و در نتیجه در بدترین حالت زمان الگوریتم $O(n^2)$ خواهد بود.

مرتب سازی هرمی

در این مرتب‌سازی داده‌ها با ساخت یک heap مرتب می‌شوند.

۱ـ مرتبه زمانی الگوریتم موردنظر در هر حالت برابر $O(n\log n)$ است.

۲ـ حافظه مصرفی این روش از مرتبه $O(n)$ است. (با فرض این که تمام کلیدها از ابتدا در دسترس باشند).