به نام خدا

دانشگاه تهران، دانشکده مهندسی برق و کامپیوتر

تحلیل و طراحی الگوریتم‌ها

تمرین کتبی ششم

موعد تحویل:

طراحان: سیدحامد میرامیرخانی ، ملیکا صادقی

---

۱. درستی یا نادرستی گزاره های زیر را با استدلال مشخص کنید :

الف) ترتیب مقادیر در یک سطر از جدول برنامه نویسی پویا برای مسئله کوله پشتی همیشه غیر نزولی است.

ب) ترتیب مقادیر در یک ستون از جدول برنامه‌نویسی پویا برای مسئله کوله پشتی همیشه غیر نزولی است.

راه حل

الف) این تساوی بیان می کند که $f(i,j-1) < f(i,j) \ for \ 1 < j < W$ که صحیح می باشد زیرا بیشترین مقدار از زیر مجموعه ای که به یک کوله پشتی با ظرفیت $j-1$ داده می شود نمی تواند از مقدار بیشینه زیر مجموعه ای که به یک کوله پشتی با ظرفیت j داده می شود بیشتر باشد.

ب) این تساوی بیان می کند $f(i-1,j) < f(i,j) \ for \ 1 < i < N$ که صحیح می باشد زیرا مقدار بیشینه زیرمجموعه‌ای از اولین $i-1$ مورد که به یک کوله‌پشتی با ظرفیت j می‌گنجد، نمی‌تواند از مقدار بیشینه زیرمجموعه‌ای از اولین i مورد که به یک کوله‌پشتی با همان ظرفیت j می‌گنجد، بیشتر باشد.

۲. اطلاعات زیر در رابطه با مسئله کوله پشتی به شما داده شده است. با رسم جدول و با استفاده از الگوریتم bottom-up مقدار بهینه را بدست بیاورید.

$$Weights = \{3,2,1,4,5\}, \quad Values = \{25,20,15,40,50\}, \quad capacity \ W = 6$$

راه حل

مطابق جدول رو به رو حالت بهینه انتخاب سومین و پنجمین شی است که برابر ۶۵ خواهد بود.

i	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	25	25	25	25
2	0	0	20	25	25	45	45
3	0	15	20	35	40	45	60
4	0	15	20	35	40	55	60
5	0	15	20	35	40	55	65

۳. در یک ردیف تعدادی کارت پشت سر هم چیده شده است و روی کارت i ام عدد v_i نوشته شده است. پارسا و پرویز در هر نوبت کارت ها را برمیدارند به طوریکه پارسا بعنوان نفر اول یکی از کارت ها را برمیدارد و در ادامه پرویز باید کارت قبلی و کارت بعدیِ پارسا را بردارد. بازی به همین شکل ادامه پیدا میکند. نفر اول یعنی پارسا یک محدودیت دارد و آن این است که مجموع ارزش کارت هایی که برمیدارد نباید بیشتر از k مقدار شود. بیشترین مقداری که کارت های پارسا میتواند داشته باشد چقدر خواهد بود؟

راه حل

جدول dp تعریف میکنیم به طوری که $dp[i][j]$ ماکزیمم مقداری است که نفر اول با i کارت اول بدست می‌آورد، در حالی که حداکثر بتواند j مقدار داشته باشد. در ادامه جدول را به این صورت پر میکنیم:

برای هر i : $dp[i][0] = 0,\ dp[0][i] = 0$

برای j های بزرگتر از $card[0]$: $dp[1][j] = card[0]$

در ادامه بررسی میکنیم اگر $card[i-1]$ از j کوچکتر یا مساوی باشد:

$$dp[i][j] = max(dp[i-1][j], card[i-1] + dp[i-2][j - card[i-1]])$$

در غیراینصورت: $dp[i][j] = dp[i-1][j]$

۴. سارا جدیدا به تئاتر علاقه مند شده و با خودش تصمیم گرفته در اوقات فراغت خود به تئاتر برود. در حال حاضر در تهران n نمایش برگزار میشود بطوریکه مدت زمان نمایش iام l_i دقیقه است و در کل c_i بار برگزار میشود. همچنین زمان های شروع نمایش iام $t_i[0], t_i[1], \dots, t_i[c_i - 1]$ است. سارا در مجموع میخواهد L دقیقه زمان صرف این کار کند. ممکن است او از نمایشی خوشش نیاید و هر زمانی در میانه ی نمایش سالن را ترک کند و همچنین میتواند هر زمانی دیدن نمایش را شروع کند و لزومی ندارد از ابتدای نمایش در سالن حضور داشته باشد.

الگوریتمی با مرتبه زمانی $O(2^n \times n \times \log \max(c_i))$ ارائه دهید که سارا با استفاده از آن بتواند تشخیص دهد میتواند از زمان ۰ تا زمان L به صورت متوالی فیلم ببیند یا خیر.

راه حل

تابع dp را تعریف میکنیم. ورودی این تابع یک زیرمجموعه از نمایش هاست و خروجی اش بیشترین زمانی که یک نفر میتواند با آن زیرمجموعه از نمایش ها خود را مشغول کند. اگر مجموعه ی کل نمایش ها M باشد، درصورتی که $dp[M]$ بیشتر یا مساوی L باشد جواب بله است و درغیر اینصورت خیر.

حالت پایه بدیهی است، dp به ازای یک مجموعه تهی برابر صفر است.

نحوه محاسبه dp: وقتی میخواهیم $dp[A]$ را حساب کنیم بر روی آخرین نمایشی که دیدیم حالت بندی میکنیم بعبارتی به ازای هر v عضو A حساب میکنیم اگر آخرین نمایشی که دیدیم v باشد حداکثر تا چه زمانی میتوانیم به نگاه

کردن نمایش ها مشغول بوده باشیم. ابتدا به $dp[A - v]$ نگاه میکنیم، بدیهی است تا آن زمان میتوانیم خودمان را سرگرم کنیم. حالا میخواهیم ببینیم که اگر بعد از آن زمان بخواهیم نمایش v را ببینیم تا چه زمانی میتوانیم مشغول بوده باشیم. نگاه میکنیم آیا بازه ای از بازه های پخش نمایش v وجود دارد که زمان شروعش قبل از $dp[A - v]$ باشد و زمان اتمامش بعد از آن؟ اگر چنین بازه ای بود آن را انتخاب میکنیم که دیرتر از همه تمام میشود. پس اگر با مجموعه A بخواهیم خودمان را سرگرم کنیم به شرط اینکه آخرین نمایشی که دیده باشیم v باشد، راحت میتوانیم حساب کنیم بیشترین زمانی که میتوانیم تا آن موقع سرگرم باشیم چه زمانی است. حالا روی تمام حالات v حرکت میکنیم و بیشینه آن ها میشود مقدار $dp[v]$.

تحلیل مرتبه زمانی: 2^n تا زیرمجموعه از نمایش ها داریم. در هرکدام از این زیرمجموعه ها حداکثر n عضو وجود دارد. به ازای هر کدام از این اعضا یک باینری سرچ روی لیست بازه های نمایش باید بزنیم. باینری سرچ از مرتبه $\log \max(c_i)$ است.

۵. خیابان ولیعصر n درخت دارد که برخی از آن ها رنگ آمیزی شده اند، اما برخی دیگر بدون رنگ اند. درختان رنگی با m رنگ مختلف رنگ آمیزی شده اند. هم چنین، هزینه ی رنگ هر درخت با هر رنگ، مقداری مشخص است. به عنوان مثال، هزینه رنگ کردن درخت i با رنگ j برابر p_{ij} است. شهردار تصمیم گرفته است که درختان رنگ نشده ی خیابان را نیز رنگ آمیزی کند. از نظر شهردار، زیبایی یک رنگ‌آمیزی برابر حداقل تعداد گروه های یک رنگ متوالی است، به گونه ای که هر گروه شامل تعدادی درخت کنار هم و هم رنگ باشد؛ به عنوان مثال، اگر رنگ درختان به شکل: ۳, ۱, ۳, ۲, ۲, ۳, ۱, ۱, ۱, ۲ باشد، زیبایی رنگ آمیزی درختان برابر ۷ است که در آن گروه ها به شکل: [۳], [۱], [۲, ۲], [۳], [۱, ۱, ۱], [۲] هستند. شهردار میخواهد با صرف حداقل هزینه ی ممکن، درختان رنگ نشده را به گونه ای رنگ آمیزی کند که زیبایی رنگ آمیزی دقیقا برابر k شود. الگوریتمی برای این کار ارائه دهید.

راه حل

تابع $dp[i][j][t]$ را به گونه ای تعریف میکنیم که زیبایی درخت ها j باشد در حالی که i درخت داشته باشیم و رنگ آخرین درخت t باشد. جواب مسئله برابر کمینه مقدار بین همه $dp[i][j][t]$ هاست.

ابتدا همه مقادیر تابع را بینهایت قرار میدهیم. برای هر $dp[i][j][t]$ سه حالت زیر وجود خواهد داشت:

الف) اگر درخت iام قبلا رنگ شده باشد و رنگی به جز t داشته باشد، نمیتوانیم کاری کنیم که رنگ آخرین درخت i شود. پس مقدار این درایه بینهایت خواهد ماند.

ب) اگر درخت iام قبلا رنگ شده باشد و رنگ t داشته باشد، دو حالت به وجود میاید،

ب.الف) درخت $i-1$ ام نیز رنگ t دارد: هزینه برابر است با $dp[i-1][j][t]$

ب.ب) درخت $i-1$ ام رنگی $m(m \neq t)$ دارد: باید $i-1$ درخت اول زیبایی $j-1$ داشته باشند. مینیمم مقدار $dp[i-1][j-1][m]$ به ازای همه مقادیر m خواهد بود.

پس در نهایت در این حالت هزینه به صورت مقابل است:

$$min(dp[i-1][j][t], min(dp[i-1][j][m] \ \forall m \mid m \neq t)$$

ج) اگر درخت iام رنگ نشده باشد تنها تفاوتی که با حالت قبل خواهد داشت این است که باید درخت iام را به رنگ t در بیاوریم. پس به هزینه حساب شده در حالت قبل p_{ij} اضافه میشود.

۶. یک گراف را «خوب» می نامیم اگر تعداد یال های آن حداکثر ۸ تا بیش تر از تعداد راس های آن باشد. فرض کنید یک گراف «خوب» وزن دار با وزن های متمایز روی یال هایش به شما داده شده است. الگوریتمی ارائه دهید که در زمان $O(n)$ درخت پوشای کمینه ی این گراف را پیدا کند.

راه حل

با DFS میتوان ۹ دور در گراف پیدا کرد به عبارتی از $n+8$ یال میخواهیم به $n-1$ یال برسیم و سنگین ترین یال هر دور را حذف کنیم(طبق قضیه $cycle\ property$) و در نهایت به گرافی با $n-1$ یال برسیم که درخت پوشای مورد نظر ما خواهد بود. عملا راه حل مشابه اثبات قضیه کروسکال است.

۷. سینا n مرغ دارد که وزن مرغ iام برابر w_i است. هم چنین می دانیم که w_i ها طبیعی و بزرگتر از ۳ هستند. تا مسابقه ی بزرگ تنها یک هفته مانده است و او برای آماده سازی مرغ های خود از رژیم به خصوصی از گندم استفاده میکند. او می تواند به هر مرغ آن قدر گندم بدهد که وزن مرغ ۱ واحد اضافه شود یا می تواند وزن هر مرغ را همان مقدار نگه دارد و یا مدتی به مرغش فقط آب بدهد که وزنش ۱ واحد کم شود. او برای بالا بردن شانس خود در مسابقات نیاز دارد مرغ هایی با وزنهای متفاوت داشته باشد. الگوریتمی طراحی کنید که در مرتبه زمانی $O(n \times log\ n)$ بتواند حداکثر تعداد مرغ هایی که در روز مسابقه وزنشان با یک دیگر متفاوت است را حساب کند.

راه حل

مرغ ها را بر اساس وزنشان مرتب میکنیم. متغیر $last$ خواهیم داشت که نشان می دهد آخرین مرغی که وزن نهایی اش را مشخص کرده ایم، چه وزنی داشته است و مقدار اولیه آن را ۰ = last در نظر میگیریم. حال از کم وزن ترین مرغ شروع میکنیم و هر دفعه اگر وزن مرغ مورد بررسی w باشد و $last < w-1$ باشد، به مرغ آب میدهیم و وزن آن $w-1$ میشود. درصورتی که $last = w-1$ باشد، مرغ در همین وزن مانده و در صورتی که $last = w$ باشد، وزن مرغ را یک واحد زیاد میکنیم. توجه کنید که در انتهای هر تصمیم گیری باید متغیر $last$ را به وزن نهایی به روز رسانی کنیم. تعداد مرغ های متفاوت نیز برابر با تعداد مرغ هایی خواهد بود که $last \neq w$ است. در این الگوریتم تنها یک مرتب کردن و یک حرکت روی اعضای آرایه را داشتیم. پس مرتبه ی زمانی $O(n\ log\ n)$ خواهد بود.

۸. یک آرایه به طول n داریم . در هر خانه از آرایه یا یک پلیس وجود دارد یا یک دزد هر پلیس می تواند حداکثر یک دزد را که در خانه ای با فاصله ی کمتر از k از او قرار دارد دستگیر کند الگوریتمی با پیچیدگی زمانی $o(n)$ ارایه دهید که حساب کند بیشترین تعداد دزدی که می توانیم دستگیر کنیم چقدر است.

راه حل

دو اشاره گر یکی به اولین خانه ای که در آن پلیس قرار دارد و یکی به اولین خانه ای که در آن دزد قرار دارد در نظر میگیریم .اگر دزد فعلی در فاصله ی کمتر از k از پلیس بود پلیس دزد را دستگیر. میکند و اشاره گر ها را به اولین دزد و پلیس بعدی به روزرسانی میکنیم. اگر فاصله ی دزد و پلیس بیشتر از k بود اشاره گر کوچکتر را به روزرسانی دزد یا پلیس بعدی اشاره کند و این کار را تکرار میکنیم. چون هرکدام از اشاره گر ها یک بار طول آرایه را طی میکنند هزینه این الگوریتم به صورت سرشکن از مرتبه $(O(n))$ است.

۹. قرار است تعدادی کارگاه در کلاسهای یک ساختمان برگزار شوند، کارگاه iام زمان s_i شروع میشود و تا زمان f_i ادامه دارد. در هر زمان هر کلاس فقط میتواند میزبان یک کارگاه باشد. الگوریتمی از مرتبه $O(n \times lg\ n)$ طرح کنید که حداقل تعداد کلاسهای مورد نیاز را بدست آورد و درستی الگوریتم خود را اثبات کنید.

راه حل

حل کردن مسئله داده شده معادل با این است که به هر بازه یک رنگ اختصاص دهیم به گونه ای که هیچ دو بازه ای که اشتراک دارند رنگ یکسانی دریافت نکرده باشند. هر رنگ را با یک عدد بزرگتر از صفر نشان می دهیم. بازه ها را طبق زمان شروع در نظر می گیریم و به هر بازه کوچکترین رنگی که به هیچکدام از بازه هایی که تا الان رنگ شده اند و با آن اشتراک دارند داده نشده را اختصاص می دهیم. واضح است که این الگوریتم یک رنگ آمیزی صحیح از بازه ها ارایه میکند (اگر دو بازه اشتراک داشته باشند رنگ یکسانی نمی گیرند).

حال اثبات می کنیم این الگوریتم حداقل تعداد رنگ را استفاده می کند. از روی بازه های داده شده گراف G را به این شکل می سازیم که به ازای هر بازه در G یک راس قرار می دهیم و بین دو راس یک یال اضافه می کنیم اگر و تنها اگر بازه های متناظر آن دو راس با هم اشتراک داشته باشند . اجرای الگوریتم داده شده روی این گراف را در نظر بگیرید. ادعا می کنیم اگر این الگوریتم به یک راس رنگ k را نسبت دهد آنگاه این گراف یک خوشه با اندازه k دارد. راسی را در نظر بگیرید که به آن رنگ k داده ایم. با توجه به اینکه بازه ها را به ترتیب شروع بررسی می کنیم ، هر بازه ای که رنگ شده باشد و با این بازه اشتراک داشته باشد، حتما با نقطه شروع آن اشتراک دارد، با توجه به اینکه این بازه رنگ خورده است حداقل $k-1$ بازه با ابتدای آن اشتراک داشته اند که به همراه خود این بازه تشکیل یک خوشه با اندازه k می دهند. با توجه به اینکه برای رنگ آمیزی رئوس یک گراف حداقل به تعداد اندازه بزرگترین خوشه آن رنگ نیاز داریم، چنین رنگ آمیزی بهینه خواهد بود. تنها بخشی که از مسئله می ماند این است که روشی ارایه دهیم که برای هر بازه کوچکترین رنگی که می توان به آن داد را در زمان مورد نیاز پیدا کند. برای انجام این کار ابتدا تمام اعداد ۱ تا n را در یک درخت جستجوی دودویی قرار میدهیم و سپس تمام نقاط (شروع و پایان) را به در نظر می گیریم و یک لیست پیوندی شامل تمام بازه هایی که به ابتدای آنها رسیده ایم ولی به انتهایشان نه را نگه می داریم. با دیدن هر نقطه، اگر نقطه شروع بود کوچکترین عدد داخل درخت را حذف کرده و آن رنگ را به بازه ای که دیده ایم نسبت می دهیم اگر نقطه پایان بود، آن را از لیست پیوندی حذف کرده و رنگی که به آن داده بودیم را به درخت اضافه می کنیم(هر نقطه شروع و پایان به بازه متناظر با خود در لیست پیوندی یک اشاره گر دارد.) زمان مورد نیاز برای مرتب سازی نقاط $O(n \times lg\ n)$ است و به ازای هر بازه یک بار به درخت اضافه یک بار در آن جستجو و یک بار از آن حذف می کنیم که زمان مورد نیاز برای جمع اینها نیز $O(n \times log\ n)$ است.

۱۰. یک شرکت مخابراتی میخواهد به خانه هایی که در طول یک مسیر و به موازات آن قرار دارند، خدمات ارتباطی ارائه کند. برای این منظور، شرکت از دکل هایی با طول برد ۸ کیلومتر استفاده میکند. الگوریتمی بیابید که این شرکت به کمک آن بتواند با کم ترین تعداد دکل، به تمام خانه های این مسیر خدمات ارائه

دهد و در ادامه درستی الگوریتم خود را ثابت کنید.

راه حل

الگوریتم به این صورت خواهد بود که اولین خانه را در نظر میگیریم و ۸ کیلومتر بعد از آن اولین دکل را نصب میکنیم. سپس هر ۱۶ کیلومتر یکبار یک دکل گذاشته تا به آخرین خانه برسیم. برای اثبات درستی هم فرض میکنیم راه حل بهینه نباشد پس میتوان جواب دیگر را در نظر گرفت و در ادامه جواب را به حالت بهینه تبدیل کرد.

۱۱. تعداد n سکه در اختیار دارید که ارزش سکه ی iام برابر c_i تومان است (i یک عدد طبیعی است). دقت کنید سکه ها مرتب شده اند و برای هرسکه داریم $c_i \leq c_i + 1$ هم چنین دقت کنید که ارزش بعضی سکه ها می تواند برابر باشند. هدف شما این است که بتوانید با سکه های که در اختیار دارید هر مقداری بین ۱ تومان تا k تومان را بپردازید ولی ممکن است این امکان وجود نداشته باشد. برای مثال اگر ۲ سکه با ارزشهای ۱ و ۳ تومان داشته باشید و $k = 4$ باشد این امکان وجود ندارد. فرض کنید شما میتوانید تعدادی سکه به سکه های خود اضافه کنید. همچنین انتخاب با شماست و می توانید ارزش هر سکه را مشخص کنید(البته ارزش هر سکه باید یک عدد طبیعی باشد). هدف این است که کمترین تعداد سکه را اضافه کنید که بتوانید هر مقداری بین ۱ تومان تا k تومان را پرداخت کنید. الگوریتمی با زمان اجرای $O(n + \log k)$ برای حل مسئله طراحی کنید و درستی الگوریتم خود را اثبات کنید.

راه حل

سکه ها را به صورت صعودی مرتب می کنیم . روی اعداد حرکت میکنیم. در هر لحظه عدد m را نگه می که نشان دهنده آن است که با سکه های موجود میتوان هزینه ۱ تا m را پرداخت کرد. فرض کنید عدد بعدی k باشد داریم:

الف) اگر $k : k < m$ تنها $m = m + k$ می کنیم.

ب) اگر $k : k > m$ ۱ + m را به مجموعه جواب ها اضافه کرده و $m = 2m + 1$

با توجه به به توجه به میزان تغییر m در حالت دوم (دو برابر شدن) و همچنین حرکت بر روی مجموعه اعداد هزینه این الگوریتم برابر با $O(n + \log k)$ خواهد بود.

اثبات درستی: بدین منظور فرض کنید راه ما با راه بهینه تفاوت داشته باشد. راه بهینه را شامل مجموعه سکه $\{X_1, X_2, \ldots, X_i, \ldots X_n\}$ باشد و فرض کنید پاسخ حریصانه مجموعه $\{y_1, y_2, y_3, y_4, \ldots\}$ باشد . فرض کنید اولین اختلاف بین دو پاسخ در ایندکس iام باشد . حال دو حالت در اینجا داریم : یا اینکه $y_i < x_i$ باشد یعنی $X > m$

پس خود m را نمی توان ساخت که این تناقض است. در غیر اینصورت در راه بهینه می دانیم میتوان تا مقدار

$m + X_i - 1$ ساخت و در راه حریصانه تا مقدار $m + y_i - 1$ را ساخت که می دانیم دومین عبارت از عبارت اول بزرگتر است پس با تعویض X_i و y_i این راه همچنان درست باقی می ماند اما اختلاف آن با راه حریصانه یک واحد کمتر می شود که این اشتباه است.