طراحی الگوریتم ها

نیم‌سال دوم ۹۹-۹۸

دانشکده‌ی مهندسی کامپیوتر

گردآورندگان: مجید گروسی، پارسا اسکندر، اسرا کاشانی‌نیا

پاسخ‌نامه تمرین تئوری سری دوم

سوالات تحویلی

مسئله‌ی ۱. کوه سکه

تعدادی سکه داریم که در یک ردیف پشت سرهم چیده شده‌اند و سکه‌ی iام ارزش c_i دارد. دو نفر این سکه‌ها را برمی‌دارند به طوریکه نفر اول یک سکه برمی‌دارد و نفر دوم باید سکه‌ی قبلی و بعدی سکه‌ی برداشته شده را بردارد. نفر اول یک محدودیت هم دارد که جمع ارزش سکه‌هایی که برمی‌دارد نباید بیشتر از k شود. بیشترین مقداری که نفر اول می‌تواند ببرد چقدر است؟

حل. یک جدول dp تعریف می‌کنیم که $dp[i][j]$ ماکزیمم مقداری است که نفر اول از بازی با i سکه اول بدست می‌آورد، در صورتی که حداکثر j تومان بتواند برنده شود. حالا جدول را به این صورت پر می‌کنیم: طبعا برای هر i،

$$dp[i][0] = 0 dp[0][i] = 0$$

برای jهای از $coin[0]$ به بعد داریم:

$$dp[1][j] = coin[0]$$

و از آن به بعد داریم: اگر $coin[i1]$ از j کوچکتر یا مساوی باشد،

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

اگر $coin[i1]$ از j بزرگتر باشد،

$$dp[i][j] = dp[i-1][j]$$

$\triangleright$

مسئله‌ی ۲. خفنیت کوروش!

کوروش که در رشته‌ی مهندسی کامپیوتر درس می‌خواند، از بچگی نیاز مبرم! به گفتن حقایق علمی و به نوعی خفن جلوه دادن خود داشته‌است!

کوروش $m \times n$ تا دوست دارد که در پی برآورد این نیاز کوروش هستند! هم‌چنین دوستانش تصمیم گرفته‌اند برای طبیعی جلوه دادن کمک به کوروش، نیاز او را در یک بازی رفع کنند.

بازی‌ای که دوستان او طراحی کرده‌اند به‌این صورت است که در یک جدول $m \times n$، هر کدام در یک خانه می‌ایستند و کوروش از یکی از خانه‌های ستون اول با انتخاب خود شروع می‌کند و در آن خانه، میتواند به مقدار گنجایش هر شخص به او حقایق علمی بگوید (گنجایش هر شخص عددی طبیعی است)، سپس کوروش میتواند دو حرکت انجام دهد، یا به خانه‌ی راست و بالا یا به خانه‌ی راست و پایین خانه‌ای که در حال حاضر در آن است برود (حرکات کوروش قطری هستند) و شروع به گفتن حقایق خود کند!

الگوریتمی در زمان $\mathcal{O}(mn)$ ارائه دهید که کوروش با استفاده از آن بتواند بیشترین مقدار حقیقت علمی را به دوستان خود بگوید. (کوروش میزان گنجایش هر دوست خود را می‌داند).

حل. برای حل این مساله یک جدول $m \times n$ تعریف می‌کنیم به نام D و شروع به پر کردن خانه‌های آن می‌کنیم بدین صورت که خانه‌ی ردیف i و ستون j ماکسیمم مجموع خانه‌هایی در جدول اصلی(V) باشد که به خانه‌ی i و j جدول اصلی ختم می‌شود.
نحوه‌ی آپدیت کردن خانه‌ها هم به‌صورت زیر است:

$$D_{i,j} = \max(D_{i-۱,j-۱}, D_{i+۱,j-۱}) + V_{i,j}$$

▷

مسئله‌ی ۳. افسردگی مجید

مجید که اخیرا به شدت افسرده شده است تصمیم گرفته برای تخفیف ناراحتی خود L دقیقه متوالی فیلم ببیند. در حال حاضر n تا فیلم در سینما اکران می شوند که مدت زمان فیلم i ام l_i دقیقه است و در کل c_i بار پخش می شود. همچنین زمان های شروع پخش فیلم i ام $t_i[۰], t_i[۱], ..., t_i[c_i - ۱]$ است. از آن جایی که حوصله‌ی مجید خیلی زود سر می‌رود، ممکن است هر زمانی در میانه‌ی یک فیلم دیدن آن را رها کند و از سالن بیرون برود اما در این صورت دیگر نمی‌تواند تا انتهای فیلم به سالن اکران آن بازگردد زیرا مسئول سالن با دیدن دوباره‌ی یک فرد به شدت عصبانی می‌شود. همچنین او می تواند هر زمانی دیدن یک فیلم را شروع کند و لازم نیست از اول فیلم در سالن حضور داشته باشد.

الگوریتمی با مرتبه‌ی زمانی $O(n \times log(max(c_i)) \times ۲^n)$ ارائه دهید که مجید با استفاده از آن بتواند تشخیص دهد می تواند از زمان ۰ تا زمان L به صورت متوالی فیلم ببیند یا خیر.

حل. برای این سوال یک تابع به اسم dp تعریف می‌کنیم. ورودی این تابع یک زیرمجموعه از فیلم‌هاست و خروجی‌اش بیشترین زمانی است که یک نفر با آن مجموعه از فیلم‌ها می‌تواند خودش را مشغول کند. (بیشینه‌ی زمان t که بتوان با آن مجموعه از فیلم‌ها و قوانین گفته شده، در کل بازه‌ی زمانی صفر تا t سرگرم باشیم.) قاعدتا اگر مجموعه‌ی کل فیلم‌ها M باشد، اگر $dp(M)$ بیشتر یا مساوی L باشد جواب بله هست، وگرنه جواب خیر هست.

حالت پایه هم بدیهی است. dp به ازای یک مجموعه‌ی تهی برابر صفر هست.

اکنون این تابع dp چگونه محاسبه می‌شود؟ ما می‌دانیم یک فیلم را حداکثر یک بار می‌بینیم. حالا وقتی می خواهیم $dp(A)$ را محاسبه کنیم، حالت‌بندی می‌کنیم برروی آخرین فیلمی که دیدیم. به عبارتی به ازای هر v عضو A، حساب می‌کنیم اگر آخرین فیلمی که دیده باشیم v باشد، حداکثر تا چه زمانی می‌توانیم به نگاه کردن فیلم‌ها مشغول بوده باشیم. این جا اوّل می‌آییم یک نگاه به $dp(A - v)$ می‌اندازیم. بدیهی است که ما تا آن زمان می‌توانیم خودمان را سرگرم نگه داریم. حالا می خواهیم ببینیم که اگر بعد از آن زمان بخواهیم فیلم v را ببینیم تا چه زمانی می‌توانیم مشغول باشیم. این جا می‌آییم نگاه می‌کنیم ببینیم آیا بازه‌ای از بازه‌های پخش فیلم v وجود دارد که زمان شروعش قبل از $dp(A - v)$ باشد و زمان اتمامش بعد از آن؟ اگر چنین بازه‌ای وجود داشت، آنی را انتخاب می‌کنیم که دیرتر از همه تمام می‌شود. (این بازه را با باینری سرچ می‌توان محاسبه کرد چون گفتیم بازه‌های هر فیلمی را فرض کنید سورت شده هستند.) پس اگر با مجموعه‌ی A بخواهیم خودمان را سرگرم کنیم به شرط این که آخرین فیلمی که دیده باشیم v باشد، راحت می‌توانیم حساب کنیم که بیشترین زمانی که می‌توانیم تا آن موقع سرگرم باشیم چه زمانی هست. حالا روی تمام حالات مختلف v حرکت می‌کنیم و بیشینه‌ی آن‌ها می‌شود $dp(v)$.

تحلیل مرتبه‌زمانی: ۲ به توان n تا زیرمجموعه‌ی مختلف از فیلم‌ها وجود دارد. در هر کدام از این زیرمجموعه‌ها، حداکثر n عضو وجود دارند. به ازای هر کدام از این اعضا، یک باینری سرچ روی لیست بازه‌های یک فیلم باید بزنیم. می‌دانیم باینری سرچمان از مرتبه زمانی $log(maxc_i)$ است. (دقّت کنید مرتبه زمانی کران بالا است) پس محاسبه‌ی dp به ازای هر مجموعه حداکثر به

اندازه‌ی n برابر این لگاریتم زمان می‌خواهد. از آن جایی که ۲ به توان n تا مجموعه داشتیم، پس الگوریتممان از مرتبه زمانی $۲^n * n * log(maxc_i)$ خواهد بود.

حرف اضافه اندرباب پیاده‌سازی: موقع کد زدن، برای نشان دادن هر زیرمجموعه، از یک عددn بیتی استفاده می‌کنیم. بیت iام این عدد به ازای یک مجموعه‌ی A برابر ۱ هست، اگر و فقط اگر فیلم iام عضو A باشد. در این صورت فور زدن روی مجموعه‌های مختلف معادل است با یک فور زدن روی اعداد ۰ تا ۲^n و فور زدن روی اعضای یک مجموعه، تبدیل می‌شود به فور زدن روی بیت‌های برابر ۱ در یک عدد. به این مدلdpها $bitmask$ می‌گویند. $\triangleright$

سوالات اضافی

مسئله‌ی ۴. توالی

یک توالی از اعداد داریم، که می‌توانیم از آنها تعدادی را انتخاب کنیم به این صورت که از اول توالی شروع می‌کنیم و هر بار که عددی را برمی‌داریم، آن عدد باید از عدد قبلی‌اش بزرگتر باشد. به جز یک بار. یک بار اجازه داریم که عددی برداریم که از عدد قبلی‌ای که برداشته‌ایم کوچکتر باشد، اما از آن به بعد، ترتیب عوض می‌شود و تا آخر هر عددی که برمی‌داریم باید از عدد قبلی اش کوچکتر یا مساوی باشد. بیشترین تعداد عددی که می‌توانیم برداریم چندتاست؟

حل. به ازای هر i، برای عدد iام، یک $LIS(i)$ و یک $LDS(i)$ قرار می‌دهیم که به این صورت مقداردهی می‌شوند:

- $LIS(i) = max(LIS(j) + ۱)$ برای هر $j < i$ که در آن $array[i] > array[j]$ باشد. و اگر چنین jای وجود نداشته باشد $LIS(i) = ۱$.
- $LDS(i) = max(LDS(j) + ۱)$ برای هر $j > i$ که در آن $array[j] < array[i]$ باشد. و اگر چنین jای وجود نداشته باشد $LDS(i) = ۱$.

حالا برای هر کدام یک $LBS(i)$ تعریف می‌کنیم که برابر $LIS(i) + LDS(i)$ ۱ است. ماکزیمم $LBS(i)$ به ازای iهای مختلف جواب است. $\triangleright$

مسئله‌ی ۵. زیردرخت کوچک

الگوریتمی با استفاده از برنامه‌نویسی پویا ارائه دهید که در درختی با N راس، تعداد زیردرخت‌های با اندازه‌ی کمتر یا مساوی K را در زمان $O(NK)$ به‌دست آورد. هم‌چنین شبه‌کد مربوط به آن را نیز بنویسید.

حل. $S(v)$ را زیردرخت با ریشه‌ی v تعریف می‌کنیم (در $S(v)$ تمامی راس‌های زیردرخت و خود V وجود دارند).

$F(v)$ را تعداد زیردرخت‌های $S(v)$ تعریف می‌کنیم که شامل راس v هستند.

اگر $v_۱, v_۲, ..., v_n$ فرزندان راس v باشند می‌دانیم $F(v) = \Pi_{i=۱}^{n}(۱ + f(v_i))$

و $G(v)$ را تعداد زیردرخت‌های $S(v)$ تعریف می‌کنیم که شامل راس v نیستند. می‌دانیم $G(v) = \sum_{i=۱}^{n}(F(i) + G(i))$ که به‌صورت بازگشتی به‌دست می‌آید. جواب مساله برابر $F(۱) + G(۱)$ خواهدبود.

هم‌چنین شبه‌کد این سوال را در این لینک می‌توانید ببینید. ▷

مسئله‌ی ۶. اعداد شلخته

به یک عدد شلخته می گوییم اگر هر دو رقم مجاور آن حداقل دو واحد اختلاف داشته باشند (دقت کنید رقم سمت چپ عدد نباید صفر باشد). دو عدد h و l داریم به طوری که $l < h$. الگوریتمی ارائه دهید که تعداد اعداد شلخته بین این دو عدد را در $O(log(h))$ بیابد.

حل. فرض کنید می‌توانیم تعداد اعداد شلخته‌ی کوچک‌تر از هر عدد X را با پیچیدگی زمانی $log(X)$ محاسبه کنیم. در این صورت مساله‌ی اصلی را هم می‌توانیم با پیچیدگی گفته شده حل کنیم. به این شکل که یک بار اعداد شلخته‌ی کم‌تر از $high$ را بشماریم و یک بار اعداد شلخته‌ی کم‌تر از low. تفاضل این مقدار برابر با جواب مساله خواهد بود.

پس کافی است اعداد شلخته‌ی کم‌تر از X دلخواه را بشماریم. به این منظور $dp[i][j]$ را برابر با تعداد اعداد شلخته‌ی i رقمی تعریف می‌کنیم که رقم سمت چپشان برابر با j باشد. علی الحساب فرض کنید این جا j صفر هم می‌تواند باشد. به وضوح آرایه به صورت زیر پر می‌شود:

$$dp[i][j] = \sum_{|k-j|>=۲} dp[i-۱][k] :$$

$$dp[۱][j] = ۱$$

فرض کنید تعداد ارقام X برابر با D هست. مشخص است که هر عدد شلخته‌ای که تعداد ارقامش از D کم‌تر باشد را باید در شمارشمان لحاظ کنیم. مقدار این برابر است با:

$$\sum_{i=۱}^{D-۱} \sum_{j=۱}^{۹} dp[i][j]$$

امّا مشکل شمردن اعداد شلخته‌ی D رقمی است. می‌دانیم که هر عدد شلخته‌ی D رقمی که از X کوچکتر باشد (اسم این عدد فرضی را Y می‌گذاریم)، اگر از پرارزش‌ترین رقمش شروع کنیم و به سمت کم‌ارزش‌ترین برویم، تا یه جایی این ارقام، با ارقام X مساوی هستن، امّا اوّلین جایی که این تساوی به هم می‌خورد، رقم Y، از رقم متناظرش در X کوچک‌تر است، چون خود Y قرار است از X کوچک‌تر باشد!

اکنون روی این که این عدد Y تا کجا با X مساوی است حالت بندی می‌کنیم. اگر فرض کنیم حداکثر e رقم اوّلش با X مساوی است، اگر e رقم ابتدایی X شلخته نباشند که تعداد چنین Yهایی صفر است. امّا اگر این e رقم شلخته باشند روی رقم $e + ۱$ ام حالت بندی می‌کنیم. این رقم می‌تواند هر چیزی باشد که اختلافش با رقم eام ایکس حداقل ۲ باشد و هم‌چنین از رقم $e + ۱$ ام X کوچک‌تر باشد. اگر مقدار چنین رقمی را j در نظر بگیریم، بدیهی است که تعداد حالات Y با این شرط‌ها برابر با $dp[D-e][j]$ خواهد بود.

واضح است که D از مرتبه‌زمانی $log(X)$ هست. ارقاممان هم که حداکثر ۱۰ حالت دارند. پس کل الگوریتم از مرتبه‌زمانی $log(X)$ خواهد بود. $\triangleright$

موفق باشید :)