پاسخ امتحان دوم - برنامه نویسی پویا

طراحی الگوریتم - بهار ۱۴۰۰

• بخش اول

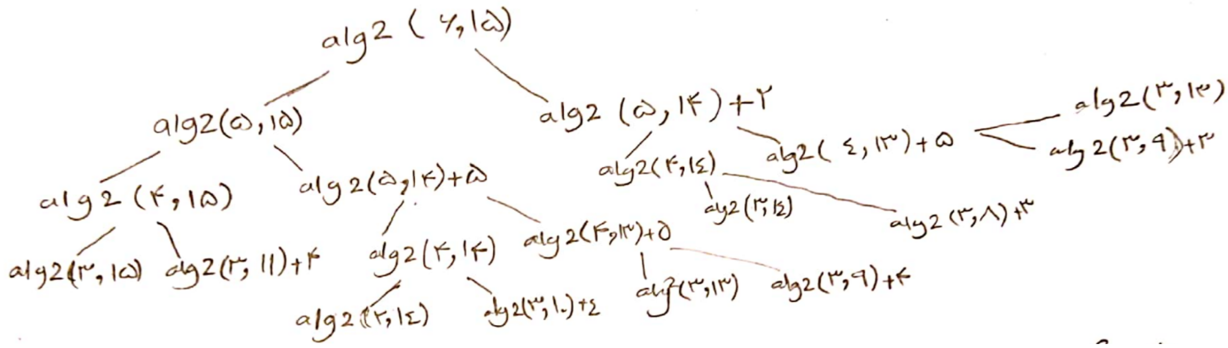
سوال اول:

این سوال را با شماره دانشجویی نمونه **810199214** حل خواهیم کرد.

اندیس (W)	۶	۵	۴	۳	۲	۱
ارزش (W)	۲	۵	۴	۳	۵	۷
حجم (V)	۱	۱	۴	۶	۳	۶

الف) الگوریتم ۱ و ۲ از نظر مرتبه زمانی یکسان هستند زیرا الگوریتم ۱ مسئله کوله پشتی را به صورت برنامه نویسی پویا حل می کند و الگوریتم ۲ مسئله را به صورت بازگشتی حافظه دار حل کرده است. اما تعداد اجرای الگوریتم ۲ کمتر است زیرا بعضی از درایه هایی که نیاز به آن ها نداریم در الگوریتم ۲ اصلا محاسبه نمی شوند ولی در الگوریتم ۱ تمام خانه ها ماتریس باید محاسبه شود.

ب) همانطور که مشخص است هیچکدام از درایه های $B[6,12]$ و $B[5,10]$ و $B[30,5]$ از این ماتریس توسط این تابع پر نمی شود. بنابراین مقدار آن ها NULL خواهد بود.



ج)

$$A = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{cccccccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \end{array}$$

$$\begin{array}{c|cccccccccccccccc|} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\ 2 & 0 & 0 & 0 & 5 & 5 & 5 & 7 & 7 & 7 & 12 & 12 & 12 & 12 & 12 & 12 & 12 \\ 3 & 0 & 0 & 0 & 5 & 5 & 5 & 7 & 7 & 7 & 12 & 12 & 12 & 12 & 12 & 12 & 15 \\ 4 & 0 & 0 & 0 & 5 & 5 & 5 & 7 & 7 & 9 & 12 & 12 & 12 & 12 & 14 & 14 & 14 \\ 5 & 0 & 5 & 5 & 5 & 10 & 10 & 10 & 12 & 12 & 14 & 17 & 17 & 17 & 17 & 21 & 21 \\ 6 & 0 & 2 & 7 & 7 & 7 & 12 & 12 & 12 & 14 & 14 & 17 & 19 & 19 & 19 & 21 & 23 \end{array}$$

● بخش دوم

سوال دوم:

برای حل این سوال از برنامه نویسی پویا و ایده الگوریتم بزرگترین زیر دنباله مشترک (LCS) استفاده می کنیم. $dp[i][j]$ را ضرب داخلی بیشینه برای i عدد ابتدای رشته a و j عدد ابتدای رشته b تعریف می کنیم. آن را با سه حالت زیر مانند LCS آپدیت می کنیم:

$dp[i][j]\ =$

$max(dp|$

واضح است که یا هر دوی i و j انتخاب می شوند و با هم مچ می شوند و یا یکی از آن ها انتخاب نمی شود. اگر i-1 یا j-1 معتبر نباشند (کمتر از ۰ شوند) آن ترم از فرمول به صفر تبدیل می شود.

حالت پایه این dp فقط حالت $dp[0][0] = 0$ است و همه استیت ها به این استیت می رسند.

جهت بعد اول از 1 تا n و جهت بعد دوم از 1 تا m است. (در این صورت هنگام محاسبه مقدار یک استیت همه استیت های حاضر در فرمول آن محاسبه شده اند)

جواب نهایی سوال در $dp[n][m]$ ذخیره خواهد شد.

حافظه مورد استفاده $O(nm)$ است که با $O(1)$ آپدیت می شود پس مرتبه زمانی نیز $O(nm)$ خواهد بود.

سوال سوم:

ابتدا $dp[i][j]$ را تعداد جایگشت های به طول i که دقیقا j وارونگی دارند تعریف می کنیم. روی مکان حضور عدد i (بزرگترین عدد این جایگشت) در جایگشت حالت بندی می کنیم و به این فرمول آپدیت می رسیم:

$$dp[i][j] = dp[i-1][j] + dp[i-1][j-1] + dp[i-1][j-2] + \ldots + dp[i-1][max(j-i-1, 0)]$$

یعنی روی i مکان حضور عدد i حالت بندی کردیم. (و وقتی که بعد دوم به صفر برسد طبیعتا باید متوقف شویم)

حالت پایه $dp[0][0] = 1$ و $dp[0][j] = 0,\ j > 0$ است.

در مورد جهت آپدیت شدن dp هم بعد اول از 1 تا n و بعد دوم از 1 تا k است. پس مموری از مرتبه $O(nk)$ است و چون مرتبه آپدیت نیز $O(n)$ است پس مرتبه زمانی الگوریتم $O(n^2k)$ خواهد بود. این مرتبه ۳۰ نمره داشت و می توان آن را به $O(nk)$ بهینه کرد.

اولا چون هر استیت فقط از سطر قبلی آپدیت می شود. (همیشه بعد اول i-1 است) پس می توان با ایده کاهش حافظه فقط سطر قبلی و سطر فعلی را نگه داشت و مرتبه مموری $O(k)$ می شود. در آرایه سطر قبلی واضح است که ما جمع یک زیربازه از آرایه را می خواهیم (از $max(j - i - 1, 0)$ تا j) پس می توانیم از ایده پارشیال سام استفاده کنیم. (پارشیال سام آرایه a برابر آرایه s است که $s[i] = \sum_{j=0}^{i-1} a[j]$) پس با منها کردن دو خانه آرایه پارشیال سام در $O(1)$ می توان مقدار $dp[i][j]$ را محاسبه نمود. پس همزمان خود آرایه dp آرایه پارشیال سام نیز محاسبه می کنیم. حال مرتبه زمانی ما $O(nk)$ است.

جواب نهایی سوال در خانه $dp[n][k]$ است.

• بخش سوم

سوال چهارم:

ابتدا $dp[l][r]$ را حداقل هزینه برای پیدا کردن جواب در بازه $[l, r)$ تعریف می کنیم. (دقت کنید که بازه را بسته-باز تعریف می کنیم) روی اینکه برای چه عددی تابع isMissing را صدا کنیم حالت بندی می کنیم. به این فرمول آپدیت می رسیم:

$$dp[l][r] = min(l + max(dp[l][l], dp[l + 1][r]),$$

$$(l + 1) + max(dp[l][l + 1], dp[l + 2][r]),$$

$$(l + 2) + max(dp[l][l + 2], dp[l + 3][r]), \dots ,$$

$$(r - 1) + max(dp[l][r - 1], dp[r][r]))$$

در واقع برای عدد تابع isMissing از اعداد l تا r-1 می توانیم انتخاب کنیم و بعد از آن بازه $[l, r)$ به دو قسمت قبل و بعد از این عدد تقسیم می شود. چون باید بدترین حالت را در نظر بگیریم بین dp این دو بازه ماکزیمم می گیریم. در بین همه این حالات باید کمینه را انتخاب کنیم که هزینه ما حداقل شود. پس به فرمول بالا خواهیم رسید.

حالت پایه $dp[i][i] = 0$ است. (می دانیم جواب عدد i است پس هزینه ای نداریم)

برای پر کردن این نوع dp که دو بعدی هستند و اول و آخر یک بازه را نشان می دهند ($dp[l][r]$) معمولا ابتدا حلقه ای برای طول بازه (length) می گذاریم و سپس اول بازه (l) را با حلقه ای دیگر فیکس می کنیم. در واقع تبدیل به $dp[l][l + length]$ خواهد شد. با این جهت مقداردهی، هنگام محاسبه هر خانه حتما مقادیر خانه های دیگر محاسبه شده اند. (اینگونه جدول dp به صورت اریب پر خواهد شد)

جواب نهایی سوال در خانه $dp[0][n]$ خواهد بود.

مرتبه حافظه مشخصا برابر $O(n^2)$ است و چون آپدیت با مرتبه $O(n)$ داریم پس مرتبه زمانی الگوریتم برابر $O(n^3)$ خواهد بود.

سوال پنجم:

ابتدا $dp[i][k]$ را حداقل تعداد صدا کردن تابع isMissing برای مطمئن شدن از پیدا کردن x با داشتن k تومان تعریف می کنیم. روی اولین حالت بندی می کنیم و با این فرمول dp را آپدیت می کنیم:

$$dp[i][k] = min(max(dp[i-1][k], dp[1][k-1]),$$
$$max(dp[i-2][k], dp[2][k-1]),$$
$$\dots , max(dp[1][k], dp[i-1][k-1]))$$

در واقع اگر $isMissing(i)$ را صدا بزنیم اگر جواب تابع ۰ باشد باید با k تومان بین اعداد i تا n بگردیم و اگر جواب تابع ۱ باشد، ۱ تومان می پردازیم و باید بین اعداد ۱ تا i-1 دنبال جواب بگردیم.

حالت های پایه $dp[1][k] = 0$ (یک عدد وجود دارد پس نیاز به صدا کردن تابع نداریم) و همینطور $dp[n][1] = n - 1$ است. (چون مجبوریم از ۱ دونه دونه بالا برویم و اعداد را صدا کنیم تا به جواب برسیم!)

اگر بعد اول را از 1 تا n و پس از آن بعد دوم را از 1 تا k پر کنیم جهت مقداردهی درست خواهد بود.

جواب نهایی در خانه $dp[n][k]$ خواهد بود.

مرتبه حافظه $O(nk)$ است که با $O(n)$ آن را آپدیت می‌کنیم پس در مجموع مرتبه زمانی $O(n^2k)$ است.