طراحی الگوریتم-تکلیف سوم

-1

برای حل این سوال سعی می‌کنیم از قسمت‌های بدیهی سوال شروع کنیم و با اطلاعات به دست آمده به مرور به جواب برسیم، واضح است که اگر $k = 0$ باشد یعنی گاو نتواند علف خود را در حین خوردن تغییر دهد حداکثر می‌تواند m کیلو علف بخورد پس اگر جدول داشته باشیم که تعداد حالت‌های خوردن را براساس وزن و k در آن بنویسیم سطر اول آن به این صورت پر خواهد شد:

0	1	1	...	1	0	...	0
	1	2	...	m	m+1	...	n

برای محاسبه سطرهای بعدی جدول یعنی تعداد حالت‌هایی که می‌تواند با یک‌بار تغییر بشتر مقداری علف را بخورد باید از سطرهای قبلی استفاده کنیم و نیازمند یک رابطه بازگشتی هستیم که هر سطر را به سطرهای قبل مرتبط کند.

اگر گاو بخواهد j کیلو علف را با k بار تغییر بخورد می‌توانیم بررسی کنیم که $j-1$ کیلو را با $k-1$ تغییر به چند حالت می‌توانسته بخورد پس الان قطعا می‌تواند با یکبار تغییر نوع علف j کیلو علف را نیز بخورد. به همین صورت گاو اگر میتوانسته $j-m$ کیلو علف را با $k-1$ بار تغییر بخورد پس الان هم می‌تواند j کیلو علف را با k بار تغییر بخورد.

با نوشتن همین روابط توصیف شده به رابطه بازگشتی زیر می‌رسیم:

$$\mathrm{DP[i][j]} = \mathrm{sum(DP[i-1][j-p]\ \ for\ p\ in\ \{1,\ 2,\ \ldots,\ m\})}$$

حال کافیست این جدول که m در n است را پر کنیم و درایه $DP[k][n]$ پاسخ مسئله خواهد بود، چون برای پر کردن هر کدام از خانه‌ها باید m خانه از ردیف قبلی را با هم جمع کنیم پس پیچیدگی زمانی این الگوریتم $O(mnk)$ است.

-2

هر بار از رشته‌ی اصلی زیر رشته‌ای که از i تا j است را انتخاب می‌کنیم و بررسی می‌کنیم. اگر کاراکتر i ام با کاراکتر j ام برابر بود بزرگترین زیر رشته متقارن از i تا j برابر است با

$$s[i][j] = s[i+1][j1] + 2$$

در غیر این صورت (اگر دو کاراکتر برابر نبود)

$$s[i][j] = max(s[i][j1], s[i+1][j])$$

مقدار دهی اولیه:

هر کاراکتر خود یک زیررشته ی متقارن به طول ۱ است

(هر دو کاراکتر متوالی در صورت برابری یک زیررشته ی متقارن به طول ۲ است)

-3

We use dynamic programming to compute the optimal solution to the two taxi cab problem. We first note that when p_i is serviced one of the taxi is in p_i while the other taxi is in one of the location $p_0 \dots p_{i-1}$, where p_0 denotes the origin. Therefore at stage i we only need to keep one solution (the best one) for each possible location $p_0 \dots p_{i-1}$ of the second taxi.

For $i = 1 \dots n$ and $j = 0 \dots i-1$ let $A(i,j)$ be the minimum cost of a routing that serves the points $p_1 \dots p_i$ and leaves one taxi in p_i and the other in p_j. Given the best solutions for stage i we can compute the solutions for stage $i+1$ by considering the following two possibilities:

(a) the taxi that was in p_i is used to serves p_{i+1} and the other taxi remains where it was. That is, for $j = 0 \dots i-1$,

$$A(i+1, j) = |p_i p_{i+1}| + A(i, j).$$

(b) the taxi in p_i remains at p_i, and the other taxi serves p_{i+1}. Only the solution with minimum cost is kept.

$$A(i+1, i) = \min_{k=0 \dots i-1} |p_k p_{i+1}| + A(i, k),$$

Using the above relations it is easy to fill the table. At the end the minimum routing cost is given by the minimum of $A(n, i)$, $i = 0 \dots n-1$. The actual routing can be reconstructed by tracing the "path" back in the table starting from the final cell.

Since there are n stages and each stage takes $O(n)$ time, the total time is $O(n^2)$ which is polynomial in n.

-4

ابتدا dpn را تعریف میکنیم هزینه ای که با صرف آن میتوان از مبدأ به قایق i ام رسید. برای رسیدن به هر قایق اگر در قایق ابتدایی نباشیم باید در گام قبل سوار یک قایق دیگر شده باشیم. به این ترتیب با یک حلقه حالت های مختلفی را که برای انتخاب قایق قبلی داشتیم همه را بررسی میکنیم و بهینه ترین پاسخ را برای dp می یابیم.

-5

اگر به ازای هر عضو بتوانیم زیردنباله ی متوالی که در آن این عضو کوچکترین عضو آن زیر دنباله است را به دست بیاوریم مسأله به راحتی قابل حل است (در این حالت حاصل این زیردنباله برابر مقدار آن عضو در طول زیر دنباله است. حال کافی است به ازای تمام زیر دنباله ها که تعداد آن‌ها n تا است این مقدار را محاسبه کرده و max بگیریم)

برای این کار به شکلی که در زیر گفته شده است از stack استفاده میکنیم.

We traverse all bars from left to right, maintain a stack of bars. Every bar is pushed to stack once. A bar is popped from stack when a bar of smaller height is seen. When a bar is popped, we calculate the area with the popped bar as smallest bar. How do we get left and right indexes of the popped bar – the current index tells us the ‘right index’ and index of previous item in stack is the ‘left index’. Following is the complete algorithm.

1) Create an empty stack.
2) Start from first bar, and do following for every bar ‘hist[i]’ where ‘i’ varies from 0 to n-1.
……**a)** If stack is empty or hist[i] is higher than the bar at top of stack, then push ‘i’ to stack.
……**b)** If this bar is smaller than the top of stack, then keep removing the top of stack while top of the stack is greater. Let the removed bar be hist[tp]. Calculate

area of rectangle with hist[tp] as smallest bar. For hist[tp], the ‘left index’ is previous (previous to tp) item in stack and ‘right index’ is ‘i’ (current index).
3) If the stack is not empty, then one by one remove all bars from stack and do step 2.b for every removed bar.

(در این مسأله به جای bar مقادیر عددی آن‌ها در یک آرایه داده شده است)

-6

فرض کنید پاسخ مسئله longest increasing subsequence را می‌دانیم در اینصورت صورت مسئله را می‌توان به اینصورت بازنویسی کرد:
بزرگترین مجموع بلندترین زیردنباله صعودی و بلندترین زیردنباله نزولی با شروع از یکی از عناصر دنباله را می‌خواهیم پس کافیست از راه حل مسئله longest increasing subsequence دو بار برای هر عنصر از آرایه خودمان استفاده کنیم.
حل longest increasing subsequence :
به اولین درایه از آرایه مورد نظر عدد یک را نظیر می‌کنیم و از این به بعد درصورتی که یک درایه از هر کدام از درایه‌های قبلش بزرگتر بود عدد نظیر شده به آن برابر با عدد درایه مورد نظر بعلاوه یک خواهد بود و در غیر اینصورت به این درایه هم عدد یک می‌دهیم.