Assignment 4

Kailing Wang 521030910356

November 28.2022

Problem 1. (20 points) Usually, we want to improve the time complexity. But now, let us talk about space complexity.

(a) (10 points) Recall the knapsack DP algorithm in the lecture, which has nW subproblems totally. We need to maintain a subproblem table with size $n \times W$. Can we use only $O(W)$ space (i.e., maintain only a $1 \times W$ size subproblem table) to implement the DP algorithm (still runs in $O(nW)$) ?

(b) (10 points) Recall the Edit Distance DP algorithm in the lecture, which has nm subproblems totally. Can we use only $O(\min\{n, m\})$ space to implement the DP algorithm (still runs in $O(nm)$) ?

Answer 1.

(a) Suppose there are n objects, and we have W total capacity. The DP function is

$$f[i, v] = \max\{f[i-1, v], f[i-1, v-w[i]]\}$$

In the DP we talked in class, we used a full table with $O(nW)$ space to store every subproblem. However, most of the information in the table is useless. When calculating the $k+1$th row, only the kth row is used. In fact, only two rows of the table is useful. Each time use one of the rows as the kth row and the other as the $k+1$th row. The actual space complexity is $O(n)$. If we update from right to left, only n space is needed, which I find in PPT. That might be the intended solution.

(b) First review the DP edit distance. For two strings a and b with length m and n. The DP function is

$$d[i][j] = \begin{cases} d[i-1][j-1] & , a[i] == b[j] \\ \min\{d[i-1][j], d[i][j-1], d[i-1][j-1]\} + 1 & , a[i] != b[j] \end{cases}$$

Similar to the first problem, only the last one row is useful. Suppose $m < n$, we only need $2 \cdot m$ space $d[2][m]$. Initialize the first column and row, and for n times do:

In the $k-1$th loop, one line is initialized or calculated. In the kth loop, update from left to right using only the information from the last row and the left part.

The space complexity is $O(m)$. If $n < m$, transpose the process and the space complexity can be $O(n)$. The total time complexity is $O(\min\{m, n\})$.

We can also do it more dynamically, and can reach $\Theta(n)$. But that will sacrifice time moving data in the memory.

Problem 2. (25 points) Given a sequence of integers $a_1, a_2, \ldots, a_n$, a lower bound and an upper bound $1 \leq L \leq R \leq n$. An (L, R)-step subsequence is a subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_\ell}$, such that $\forall 1 \leq j \leq \ell - 1, L \leq i_{j+1} - i_j \leq R$. The revenue of the subsequence is $\sum_{j=1}^{\ell} a_{i_j}$. Design a DP algorithm to output the maximum revenue we can get from a (L, R)-step subsequence.

(a) (5 points) Suppose $L = R = 1$. Design a DP algorithm in $O(n)$ to find the maximum $(1, 1)$-step subsequence.

(b) (10 points) Design a DP algorithm in $O\left(n^2\right)$ to find the maximum (L, R)-step subsequence for any L and R.

(c) (10 points) Design a DP algorithm in $O(n)$ to find the maximum (L, R)-step subsequence for any L and R.

Answer 2.

(a) The subsequence is continuous if $L = R = 1$. The max revenue of the k-front sequence $(a_1, a_2, \ldots, a_k, k \leq n)$ is only based on the (k-1)-front sequence. This is:

$$R[k] = \begin{cases} R[k-1] & , a_k \leq 0 \\ R[k-1] + a_k & , a_k > 0 \end{cases}$$

Initialize R[0]=0, and then we can simply DP from the front to the end and select the largest $R[k]$. Correctness: for this problem, the answer is obviously the sum of all the positive numbers, and this DP can obviously achieve this.

(b) For this condition, the a_k can inherit the revenue of front sequences that end from a_{k-R} to a_{k-L}. The $O(n^2)$ DP is to DP like the previous problem but each update takes $R - L$ time. The runtime is $O(n(R - L)) = O(n^2)$. The DP function is like:

$$R[k] = \max_{L \leq m \leq R} R[k - m] + \max\{a_k, 0\}$$

Select the largest $R[k]$. The correctness is, assume $R[1], R[2], \ldots, R[k-1]$ are optimal, $R[k]$ must be optimal because it inherits the max revenue it can.

(c) The DP structure is the same as the previous problem. To make it $O(n)$. Use potential list. The correctness of the potential list is proved in class. Anyway, I decide to prove the time complexity here. The potential list is a queue with length $R - L + 1$ and the update rule is:

- For the first R elements, each time pop the queue from the end until the tail is larger than the new element, and enqueue the new element.
- For the other elements, each time in addition pop the head if the head is R+1 distance away.

This ensures the list is in descending order. And with this structure we only need $O(1)$ to select the largest legal revenue. We need $O(n)$ time complexity, and the time to maintain a potential list is the bottleneck. Consider the overall operations concerning the potential list: Every element can only enqueue once and be popped once, so the total time complexity caused by the potential list is also $O(n)$. The total is $O(n)$.

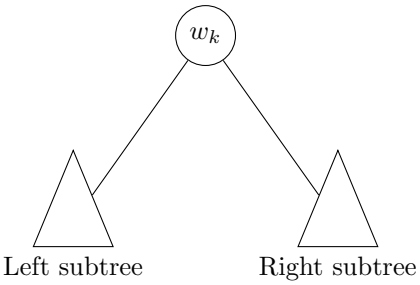
Problem 3. (25 points) Optimal Indexing for A Dictionary Consider a dictionary with n different words $a_1, a_2, \ldots, a_n$ sorted by the alphabetical order. We have already known the number of search times of each word a_i, which is represented by w_i. Suppose that the dictionary stores

all words in a binary search tree T, i.e., each node's word is alphabetically larger than the words stored in its left subtree and smaller than the words stored in its right subtree. Then, to look up a word in the dictionary, we have to do $\ell_i(T)$ comparisons on the binary search tree, where $\ell_i(T)$ is exactly the level of the node that stores a_i (root has level 1). We evaluate the search tree by the total number of comparisons for searching the n words, i.e., $\sum_{i=1}^{n} w_i \ell_i(T)$. Design a DP algorithm to find the best binary search tree for the n words to minimize the total number of comparisons.

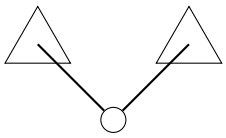
Answer 3.

I don't understand the trick of using words instead of simple number——it's the same problem. Consider how an optimal tree is constructed. Suppose there is an optimal tree. The left and right subtree are also optimal trees. Otherwise changing the subtrees into optimal tree the total comparison time will reduce. Therefore any subtree in the main tree is an optimal tree.

First sort the words. This way, the optimal tree is constructed by selecting a root word w_k each time and go on dp the left and right subtree. The words on the left of w_k is on the left subtree and the other on the right.



(Spent hours to learn TIKZ but failed. Want tutorial)



Now consider how to find the w_k on a range $[i, j] \in [1, n]$. The DP function is

$$C[i,j] = \min_{i \le k \le j} \{C[i, k-1] + c[k+1, j]\} + \sum_{n=i}^{j} w_i$$

Here since all the words in the subtree become 1 level deeper in the new tree, we need to add all the search time once. The naive way is to calculate all the subtrees.

- First initialize $C[i, i] = w_i :\ O(n)$
- Calculate all the subtree $C[i, i+1]$ with length 2 using the DP function: $O(n-1)$
- Calculate all the subtree $C[i, i+2]$ with length 3: $O((n-2) * 2)$
- Repeat, and calculate subtree $C[i, i+k]$ is $O((n-k) * k))$

The process can also be described in a table where we initialize the diagonal and DP towards the Upper-Right corner. The total time complexity is

$$\begin{aligned} & O(n) + \sum_{k=1}^{n-1} O((n-k) * k)) \\ =& O\left(n + \frac{n(n-1)}{2} - \frac{(n-1)(n-2)(2n-3)}{6}\right) \\ =& O(n^3) \end{aligned}$$

Sadly the time complexity is not that good.

The 1-length subtrees are obviously optimal. If the (k-1)-length subtrees are optimal. The DP function must get the optimal because it includes all the cases that the two subtrees are optimal, which is required by the optimal main tree. So the final tree must be optimal according tp induction.

Problem 4. (30 points) Collecting Gift On a Grid Given n gifts located on a $(m \times m)$ grid. The i-th gift is located at some point (x_i, y_i) (integers chosen in $1 \cdots m$) on the grid. A player at $(1, 1)$ is going to collect gifts by several Upper-Right Move. In particular, assuming the player is currently located at (x, y), he can make one Upper-Right Move to another point (x', y') where $x' \geq x$ and $y' \geq y$. The cost of this movement is $(x' - x)^2 + (y' - y)^2$. The player will collect the i-th gift when he is at point (x_i, y_i). There is no restriction for the number of Upper-Right Move and the final location of the player.

(a) (15 points) Design an $O\left(m^2\right)$ algorithm to maximize the player's profit, i.e., the sum of value he collects minus the sum of cost he pays for his Upper-Right Move.

(b) (15 points) Sometimes, n can be much smaller than m. Can you design another algorithm that runs in $O\left(n^2\right)$ for this situation?

(c) (0 Points. It is for fun, you can discuss your idea with me.) Is there any difference if the player can only make Upper-Right Move among gifts? Can we still design efficient algorithm runs in $O\left(n^2\right), O(nm)$, and $O\left(m^2\right)$?

Answer 4.

(a) The movement cost $(x' - x)^2 + (y' - y)^2$ means any movement can be decomposed into x and y directions. However, since $x^2 \geq x$, we achieve the smallest cost when going step by step.

Now consider the grid. To any place we should walk step by step or it cost us more. This means and point is reached by the point under or left to it. View the value of gift is 0 if there is not a gift. The DP function is

$$P[i, j] = \max\{P[i - 1, j], P[i, j - 1]\} - 1 + v[i, j]$$

Here the -1 means the step cost us 1 and v[i,j] is 0 or the value of the gift.

We update the table either row by row or col by col until the grid is full. This DP process include all the possible route so the correctness is obvious. Since each update is $O(1)$, the time complexity equals the number of points $O(m^2)$.

(b) When there is not so many gift, many step in the m*m only adds the cost by one and can be replaced with a single step. Now we only care about the relative positions of the gift. Sort the gift positions twice by x and y respectively. The the sorted index is their position in the n*n grid, and we add the 0-th row and col and put (1,1) with value 0 on the position (0,0). For example, the gift on the point (114,514) have the 19th largest x and 81 largest y. So it should be on the (19,81) point on the new grid. No two gift are on the same row or col: each row or each col has one and only gift. The distance of the two rows i and j is calculated as $|y_i - y_j|$, the difference of y of the two gift on the rows. The distance between cols is calculated by the difference of x.

Now the problem is rather the same as the previous one.

$$P[i,j] = \max\{P[i-1,j] - d_{row}[i-1], P[i,j-1] - d_{col}[j-1]\} + v[i,j]$$

The correctness and time complexity is also the same. Since now we have a n*n grid and we use $O(n \log n)$ to sort and $O(n)$ to initialize the new grid, the total time complexity is $O(n)$.

Problem 5. How long does it take you to finish the assignment (including thinking and discussing)? Give a score $(1, 2, 3, 4, 5)$ to the difficulty. Do you have any collaborators? Write down their names here.

Answer 5.

Today(28. Nov, at around 8 p.m.) I saw Professor Yuhao and his fellows(?) in GIMLID (The 5th restaurant building, SJTU). The sandwich there worth 4 score.

Bujiazi. After viewing his previous assignment and his handwriting, I found myself lack the ability of English and making clear explanation.