Algorithm Design and Analysis (Fall 2023)
Assignment 4
Deadline: Dec 26, 2023

1. (30 points) Consider that you are in a stock market and you would like to maximize your profit. Suppose the prices of the stock for the n days, $p_1, p_2, \ldots, p_n$, are given to you. On the i-th day, you are allowed to do exactly one of the following operations:

   - Buy one unit of the stock and pay the price p_i. Your stock will increase by 1.
   - Sell one unit of stock and get the reward p_i if your stock is at least 1. Your stock will decrease by one.
   - Do nothing.

   Design an $O(n^2)$ time dynamic programming algorithm.

   Remark: [Not for credits] There exits a clever greedy algorithm that runs in $O(n \log n)$ time. Can you figure it out?

   Algorithm:

   1. Initialize an array $DP[n+1][n+1]$ with $DP[0][0] = 0$, and other $DP[i][j] = -INF$.
   2. Repeatedly update DP with $DP[i][j] = max\{DP[i-1][j], DP[i-1][j-1]-p_i, DP[i-1][j+1]+p_i\}$
   3. After all updates are done, the maximum profit is $max\{DP[n][0], DP[n][1], ..., DP[n][n]\}$

   Time Complexity

   Updating the array takes n^2 rounds and each round takes $O(1)$. Choosing the final answer takes $O(n)$. Therefore, the time complexity is $O(n^2)$.

   Correctness

   Base Step: When $i = 1$, the only choice is either buy or do nothing. So $DP[1][0] = 0, DP[1][1] = -p_1$, which corresponds to the algorithm.

   Inductive step: Suppose $DP[x][y]$ is correct for all $x \leq i$. Then, consider calculating $DP[i+1][j]$, we traverse the three choices to select one with maximal profit. $DP[i][j]$ means doing nothing, $DP[i][j-1]-p_{i+1}$ means buying one and $DP[i][j+1]+p_{i+1}$ means selling one. $DP[i+1][j]$ means that you have j stocks in $i+1-th$ day. If $DP[i][j]$ is INF, it is impossible to have j stocks in $i-th$ day. So, by applying the algorithm on $DP[i+1][j]$ for all j has been traversed, it should give largest profit on $i+1-th$ day with all possible stocks. Therefore, we prove the correctness.

2. (30 points) Given two strings $x = x_1x_2 \cdots x_n$ and $y = y_1y_2 \cdots y_n$, we wish to find the length of their *longest common subsequence*, that is, the largest k for which there are indices $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_k$ with $x_{i_1}x_{i_2} \cdots x_{i_k} = y_{j_1}y_{j_2} \cdots y_{j_k}$. Design an $O(n^2)$ dynamic programming algorithm for this problem.

Algorithm

1. Initialize an array $DP[n+1][n+1]$ with $DP[0][j] = DP[i][0] = 0$ for $i, j = 0, 1, ..., n+1$ and other $DP[i][j] = -INF$.

2. Repeatedly update DP with $DP[i][j] = \begin{cases} DP[i-1][j-1] + 1 & \text{if } x_i = y_j \\ max\{DP[i-1][j], DP[i][j-1]\} & \text{if } x_i \neq y_j \end{cases}$

3. Finally, the answer is $DP[n][n]$.

Time Complexity

Updating the array takes n^2 rounds and each round takes $O(1)$. Therefore, the time complexity is $O(n^2)$.

Correctness

Base Step: When $i = j = 1$, we can only judge the length by x_1 and y_1, which corresponds to $DP[1][1]$ by the algorithm.

Inductive step: Suppose $DP[x][y]$ is correct for all $x < i$ and $y < j$. We need to prove that $DP[i][j]$ is also correct.

If $x_i = y_j$, then the length will increase by 1 since we could let $x_i = x_{i_{DP[i-1][j-1]+1}}$ and $y_j = y_{j_{DP[i-1][j-1]+1}}$.

If $x_i \neq y_j$, there is at most one element between x_i and y_j could increase the length because if both x_i and y_j increase the length, then $x_i = y_j$. So we find the larger one between $DP[i-1][j]$ and $DP[i][j-1]$, which means either y_j increases the length or x_i increases the length. If $DP[i-1][j] = DP[i][j-1]$, both x_i and y_j won't increases the length. Since $DP[x][y]$ is correct for all $x < i$ and $y < j$, $DP[i][j]$ is also correct. Therefore, we prove the correctness of the algorithm.

3. (40 points) In the *Traveling Salesman Problem* (TSP), we are given an undirected weighted complete graph $G = (V, E, w)$ (where $(i, j) \in E$ for any $i \neq j \in V$). The objective is to find a cycle of length $|V|$ with minimum total weight, i.e., to find a tour that visit each vertex exactly once such that the total distance traveled in the tour is minimized. Obviously, the naïve exhaustive search algorithm requires $O((n-1)!)$ time. In this question, you are to design a dynamic programming algorithm for the TSP problem with time complexity $O(n^2 \cdot 2^n)$.

   (a) (10 points) Show that $n^2 \cdot 2^n = o((n-1)!)$, so that the above-mentioned algorithm is indeed faster than the naïve exhaustive search algorithm.

   (b) (30 points) Design this algorithm. Hint: label all vertices as $1, 2, \dots, n$; given $i \in V$ and $S \subseteq V \setminus \{1, i\}$, let $d(S, i)$ be the length of the shortest path from 1 to i where the intermediate vertices are *exactly* those in S; show that the minimum weight cycle/tour is $\min_{i=2,3,\dots,n}\{d(V \setminus \{1, i\}, i) + w(i, 1)\}$.

(a) To show that $n^2 \cdot 2^n = o((n-1)!)$, we just need to show that $\lim\limits_{n\to\infty} \frac{n^2 \cdot 2^n}{(n-1)!} = 0$. Consider $\sum_{n=1}^{\infty} \frac{n^2 \cdot 2^n}{(n-1)!}$. By ratio test, $\lim\limits_{n\to\infty} \frac{\frac{(n+1)^2 \cdot 2^{(n+1)}}{n!}}{\frac{n^2 \cdot 2^n}{(n-1)!}} = \lim\limits_{n\to\infty} \frac{(n+1)^2 \cdot 2}{n^3} = 0 < 1$, so the series converges. Therefore, by the nature of convergence, $\lim\limits_{n\to\infty} \frac{n^2 \cdot 2^n}{(n-1)!} = 0$, which means $n^2 \cdot 2^n = o((n-1)!)$.

(b)**Algorithm**

1. Initialize an array $DP[n+1][2^{n-1}]$ where $DP[i][S_j]$ means the length of the shortest path from 1 to i where the intermediate vertices are exactly those in $S_j (1, i \notin S_j)$. $S_0, S_1, ..., S_{2^{n-1}}$ are all combinations of all vertices(1 is not in any of S). Let $DP[i][\{\}] = w(1, i)$ for all $i \in V \setminus \{1\}$ and other $DP[i][S_j] = INF$.
2. Repeatedly update $DP[i][S_j] = min\{DP[k][S_j \setminus \{k\}] + w(k, i)\}$ for all $k \in S_j$.
3. After all the updates are done, the final answer is $min_{i=2,3,\dots,n}\{DP[i][V \setminus \{1, i\} + w(i, 1)\}$.

Time Complexity

The algorithm takes $n \cdot 2^{n-1}$ rounds, and each round takes $O(n)$. To find the optimal answer takes $O(n)$, so the overall time complexity is $O(n^2 \cdot 2^n)$

Correctness

Base step: $DP[i][\{\}]$ means the direct distance between i and 1 $w(1, i)$, which corresponds to the algorithm.

Inductive step: Suppose all $DP[i][S_j]$ are correct, where S_j means all S with $|S| < k$. We need to show that $DP[i][S_{j+1}]$ is correct for all $|S_{j+1}| = k$. By the algorithm, $DP[i][S_{j+1}] = min\{DP[x][S_{j+1} \setminus \{x\}] + w(x, i)\}$ for all $x \in S_{j+1}$. $DP[x][S_{j+1} \setminus \{x\}] + w(x, i)$ means the shortest path from 1 to x passing $S_{j+1} \setminus \{x\}$ adding the length of (x, i). And they make up a set of all subminimum paths from 1 to i. The minimum of them means the shortest path from 1 to i passing S_{j+1} because all $DP[i][S_j]$ is correct. Therefore, the algorithm is correct.

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

   1 days.

   4, 4, 5.

   No.