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?

Solution:

Define the sub-problem $f(i,j)$ being the maximum of the profit we have earned after i days with j stocks left. For the first operation, $f(i,j) = f(i-1, j-1) - p_i$ when $j > 0$. For the second operation, $f(i,j) = f(i-1, j+1) + p_i$. For the third operation, $f(i,j) = f(i-1,j)$. Therefore, we have the following recurrence relation:

$$f(i,j) = \begin{cases} \max\{f(i-1,j-1) - p_i,\ f(i-1,j+1) + p_i,\ f(i-1,j)\}, & j > 0, \\ \max\{f(i-1,j+1) + p_i,\ f(i-1,j)\}, & \text{otherwise.} \end{cases}$$

However, $f(i,j)$ is invalid when $j > i$ since we cannot have more than i stocks after i days even if we buy one unit of the stock every day. For initialization, we can set $f(0,0) = 0$ and $f(i,j) = -\infty$ where $j > i$.

The algorithm is below.

Algorithm 1 Maximize the Profit

Input: the prices of the stock for n days, $p_1, p_2, ..., p_n$

```
Initialize f[0][0] = 0 and f[i][j] = −∞ for i < j
for i = 1, ..., n do
    f[i][0] = max{f[i − 1][1] + p_i, f[i − 1][0]}
    for j = 1, ..., i do
        f[i][j] = max{f[i − 1][j − 1] − p_i, f[i − 1][j + 1] + p_i, f[i − 1][j]}
return max_{j=0,...n} f[n][j]
```

The correctness of the algorithm is trivial by induction. For the base step, we do nothing before the first day, so $f(0,0)$ is 0. For the inductive step, suppose $f(i-1,j-1)$, $f(i-1,j)$, $f(i-1,j+1)$ are correct. We can only do three types of operations on i-th day, so the maximum profit we can earn after i days with j stocks left is the maximum of three operations. In the expressions of three operations, only $f(i-1,j-1)$, $f(i-1,j)$, $f(i-1,j+1)$ are involved and they are correct, so $f(i,j)$ is correct.

The initialization is $O(n^2)$ since it traverses half of the state space $O(n^2)$. The main part is $O(n^2)$ since it traverses another half. So the overall time complexity is $O(n^2)$.

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.

Solution:

This problem is similar to *longest palindrome sub-sequence* problem introduced in the lecture. Let $S[i]$ denotes the prefix $s_1s_2...s_i$. Define the sub-problem $f(i,j)$ being the length of the longest common sub-sequence (abbreviated as LCS) of $X[i]$ and $Y[j]$. We have the following recurrence relation:

$$f(i,j) = \begin{cases} f(i-1,j-1)+1, & x_i = y_j, \\ \max\{f(i,j-1),\ f(i-1,j)\}, & \text{otherwise.} \end{cases}$$

The algorithm is below.

Algorithm 2 Find the Length of LCS of Two Strings

Input: two strings $x = x_1x_2...x_n$ and $y = y_1y_2...y_n$

Initialize $f[i][0] = 0$ and $f[0][j] = 0$ for each $i \in [0,\ n]$ and $j \in [0,\ n]$.
for $i = 1, ..., n$ **do**
  for $j = 1, ..., n$ **do**
    if $x[i] = y[j]$ **then** $f[i][j] = f[i-1][j-1] + 1$
    else $f[i][j] = \max\{f[i-1][j],\ f[i][j-1]\}$
return $f[n][n]$

I'll show the correctness of the algorithm by induction. Claim that $f(i,j)$ gives the correct the length of LCS of $X[i]$ and $Y[j]$.

Base step:

If $i = 0$ or $j = 0$, at least one string is empty, which implies the length of LCS is 0. Therefore, $f(i,0) = f(0,j) = 0$ for any $i \in [1,n]$ and $j \in [1,n]$.

Inductive step:

Suppose that $f(m,n)$ is correct where $m \in [0,i]$, $n \in [0,j]$ except $m = i, n = j$. We need to show $f(i,j)$ is also correct. There are two scenarios when proceeding with $f(i,j)$:

- $x_i = y_j$:

  Suppose $a_1a_2...a_m$ (with length m) is the LCS of $X[i-1]$ and $Y[j-1]$, then $a_1a_2...a_mx_i$ (with length $m+1$) is the LCS of $X[i]$ and $Y[j]$. Otherwise, let OPT (with length $\geq m+1$) denotes the LCS of $X[i]$ and $Y[j]$.

If $x_i \notin OPT$, then OPT is a common sub-sequence of $X[i-1]$ and $Y[j-1]$. We can append x_i to OPT to get a better solution, which leads to contradiction.

If $x_i \in OPT$, then $OPT' = OPT \setminus \{x_i\}$ (with length $\geq m$) is a common sub-sequence of $X[i-1]$ and $Y[j-1]$. If $|OPT'| > m$, this contradicts to the assumption that $f(i-1, j-1)$ is correct. If $|OPT'| = m$, we can alternately choose OPT to be the LCS of $X[i]$ and $Y[j]$ whose length is also $m+1$.

Therefore, $f(i,j) = f(i-1, j-1) + 1$ holds for this scenario.

- $x_i \neq y_j$:

  Suppose $f(i-1, j) \geq f(i, j-1)$ w.l.o.g., and $a_1 a_2 ... a_m$ (with length m) is the LCS of $X[i-1]$ and $Y[j]$. Then $a_1 a_2 ... a_m$ is the LCS of $X[i]$ and $Y[j]$. Otherwise, let OPT (with length $\geq m$) denotes the LCS of $X[i]$ and $Y[j]$.

  If $x_i \notin OPT$, then OPT is a common sub-sequence of $X[i-1]$ and $Y[j]$. If $|OPT| > m$, this contradicts to the assumption that $f(i-1, j)$ is correct; if $|OPT| = m$, we can alternately choose OPT to be the LCS of $X[i]$ and $Y[j]$ whose length is also m.

  If $x_i \in OPT$, then $y_j \notin OPT$ since $x_i \neq y_j$, which implies that OPT is a common sub-sequence of $X[i]$ and $Y[j-1]$. If $|OPT| > m$, this contradicts to the assumption that $f(i-1, j) = m \geq f(i, j-1)$. If $|OPT| = m$, we can also choose $|OPT|$.

  Therefore, $f(i,j) = \max\{f(i, j-1),\ f(i-1, j)\}$ holds for this scenario.

Therefore, $f(i,j)$ is correct, and the inductive step is completed. The correctness of the algorithm is also proven.

The initialization is $O(n)$. In the main part, the algorithm traverse each $i \in [1, n]$ and $j \in [1, n]$ to compute $f[i][j]$ which is $O(1)$, so this part is $O(n^2)$. The overall time complexity is $O(n^2)$.

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 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 $1, 2, \ldots, n$; 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 $\min_{i=2,3,\ldots,n}\{d(V \setminus \{1, i\}, i) + w(i, 1)\}$.

Solution:

(a) According to Stirling's approximation, we have that

$$(n-1)! = \sqrt{2\pi(n-1)}\left(\frac{n-1}{e}\right)^{n-1}.$$

Take the logarithm of the values, we have

$$\log(n^2 \cdot 2^n) = 2\log n + n\log 2 = O(n),$$
$$\log((n-1)!) = \frac{1}{2}\log 2\pi(n-1) + (n-1)\log(n-1) - (n-1)\log e = O(n\log n).$$

These two equations imply $n^2 \cdot 2^n = o((n-1)!)$.

(b) Following the hint, define the sub-problem $d(S, i)$ be the length of the shortest path from 1 to i where the intermediate vertices are *exactly* those in S. Then we have the following recurrence relation:

$$d(S, i) = \min_{j \in S}\{d(S \setminus \{j\}, j) + w(j, i)\}.$$

For initialization, we can set $d(\emptyset, i) = w(1, i)$. The complete algorithm is below.

Algorithm 3 Solve TSP with Dynamic Programming

Input: an undirected weighted complete graph $G = (V, E, w)$

Initialize $d(\emptyset, i) = w(1, i)$.
for $m = 1, 2, ..., n-2$ **do**
  for $S \subseteq V \setminus \{1\}$ with size $|S| = m$ **do**
    for $i \in V \setminus (S \cup \{1\})$ **do**
      $d(S, i) = \min_{j \in S}\{d(S \setminus \{j\}, j) + w(j, i)\}$
return $\min_{i=2,3,\ldots,n}\{d(V \setminus \{1, i\}, i) + w(i, 1)\}$

The correctness of the algorithm is below.

First, claim that $d(S,i)$ produced by the algorithm is correct, i.e., $d(S,i)$ is the length of the shortest path from 1 to i where the intermediate vertices are exactly those in S. I'll prove this by induction.

Base step:

$d(\emptyset, i) = w(1,i)$ holds trivial since the path from 1 to i without intermediate vertices is exactly the edge whose endpoints are 1 and i.

Inductive step:

For $d(S,i)$, suppose $d(S \setminus \{j\}, j)$ is correct for any $j \in S$. Suppose the **optimal** path from 1 to i with intermediate vertices formed by S is $1s_1s_2...s_{|S|}i$, whose length is $d(S,i) = d(S \setminus \{s_k\}, s_k) + w(s_k, i)$. This is one of the terms in $\{d(S \setminus \{j\}, j) + w(j,i),\ j \in S\}$. Since $d(S \setminus \{j\}, j)$ is correct, which implies all these terms are true distance of paths, therefore $d(S, s_k) + w(s_k, i) \leq d(S \setminus \{j\}, j) + w(j,i), j \in S$ according our assumption. Therefore $d(S,i)$ is correct and furthermore the claim is also correct.

The left we have to do is to show that the minimum weight cycle is $\min_{i=2,3,\ldots,n}\{d(V \setminus \{1,i\}, i) + w(i,1)\}$. This is similar to the inductive step. The minimum weight cycle must be one of the terms in $d(V \setminus \{1,i\}, i) + w(i,1)$, take the minimum for this will gives the correct answer.

The first loop traverses with size of $O(n)$, the second and third loop traverses all subset of V with size of $O(2^n)$, and computing the minimum from $j \in S$ requires $O(n)$ time. So the overall time complexity is $O(n^2 \cdot 2^n)$.

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.

Solution:

These three questions cost 1.5h, 2.5h, 3h respectively (typing also included).

The difficulty scores are 2, 3, 3 respectively.

I complete this assignment independently.