CSCI 570 - Fall 2016 - HW 7 solution

1. (Chapter 6, Exercise 10)

   Solution:

   (a) Consider the following example: there are totally 4 minutes, the numbers of steps that can be done on the two machines in the 4 minutes are listed as follows (in time order):

   - Machine A: 2, 1, 1, 200
   - Machine B: 1, 1, 20, 100

   The given algorithm will choose A then move, then stay on B for the final two steps. However, the optimal solution will stay on A for the four steps.

   (b) An observation is that, in the optimal solution for the time interval from minute 1 to minute i, you should not move in minute i; because otherwise, you can keep staying on the machine where you are and get a better solution ($a_i > 0$ and $b_i > 0$). For the time interval from minute 1 to minute i, consider that if you are on machine A in minute i, you either (i) stay on machine A in minute $i-1$ or (ii) are in the process of moving from machine B to A in minute $i-1$. Now let $OPT_A(i)$ represent the maximum value of a plan in minute 1 through i that ends on machine A, and define $OPT_B(i)$ analogously for B. If case (i) is the best action to make for minute $i-1$, we have $OPT_A(i) = a_i + OPT_A(i-1)$; otherwise, we have $OPT_A(i) = a_i + OPT_B(i-2)$. Thus, we have

$$OPT_A(i) = a_i + \max\{OPT_A(i-1), OPT_B(i-2)\}$$
$$OPT_B(i) = b_i + \max\{OPT_B(i-1), OPT_A(i-2)\}$$

   Then the algorithm is illustrated in Algorithm 1.

Algorithm 1

$OPT_A(0) = 0$
$OPT_B(0) = 0$
$OPT_A(1) = a_1$
$OPT_B(1) = b_1$
for i from 2 to n **do**
    $OPT_A(i) = a_i + \max\{OPT_A(i-1), OPT_B(i-2)\}$
    Record the action (either stay or move) in minute $i-1$ that achieves the maximum
    $OPT_B(i) = b_i + \max\{OPT_B(i-1), OPT_A(i-2)\}$
    Record the action (either stay or move) in minute $i-1$ that achieves the maximum
end for
return $\max\{OPT_A(n), OPT_B(n)\}$
Track back through the arrays OPT_A and OPT_B by checking the action records from minute $n-1$ to minute 1 to recover the optimal solution

   It takes $O(1)$ time to complete the operations in each iteration; there are $O(n)$ iterations; the tracing backs takes $O(n)$ time. Thus, the overall complexity is $O(n)$.

2. (Chapter 6, Exercise 20)

Solution: Let the (i, h)-subproblem be the problem in which one wants to maximize one's grade on the first i courses, using at most h hours. Let $OPT(i, h)$ be the maximum total grade that can be achieved for this subproblem. Then $OPT(0, h) = 0$ for all h, and $OPT(i, 0) = \sum_{j=1}^{i} f_j(0)$. Now, in the optimal solution to the (i, h) subproblem, one spends k hours on course i for some value of $k \in \{0, 1, ..., h\}$; thus, we have

$$OPT(i, h) = \max_{0 \le k \le h} \{f_i(k) + OPT(i-1, h-k)\}$$

Then the algorithm is illustrated in Algorithm 2.

Algorithm 2

for h from 1 to H **do**
    $OPT(0, h) = 0$
end for
$OPT(1, 0) = f_1(0)$
for i from 2 to n **do**
    $OPT(i, 0) = f_i(0) + OPT(i-1, 0)$
end for
for i from 1 to n **do**
    for h from 1 to H **do**
        $OPT(i, h) = \max_{0 \le k \le h} \{f_i(k) + OPT(i-1, h-k)\}$
        Record the k that results in the maximum, denoted as $k^*(i, h)$
    end for
end for
return $OPT(n, H)$
Having obtained the $(n+1) \times (H+1)$ table with each entry filled with $OPT(i, h)$; in order to produce the optimal distribution of time, track back from the $(n+1, H+1)$th entry using $\{k^*(i, h)\}$ until a boundary entry of the table is reached

The total time to fill in each entry $OPT(i, h)$ is $O(H)$, and there are nH entries, resulting in a total time of $O(nH^2)$. Tracking back according to the records takes $O(n)$ time. Thus, the overall time is $O(nH^2)$.

3. Given a sequence $\{a_1, a_2, ..., a_n\}$ of n numbers, describe an $O(n^2)$ algorithm to find the longest monotonically increasing sub-sequence.

Solution: Let l_i denote the length of the longest monotonically increasing sub-sequence that ends with a_i ($l_1 = 1$). Compute the sequences S_i, S_{ij} using the following recurrences.

- Initialize $S_1 = a_1$.
- For $1 \leq j < i$, if $a_j > a_i$ then $S_{ij} = a_i$. Otherwise, S_{ij} is set to $\{S_j$ concatenated with $a_i\}$.
- S_i is set to the longest sequence among all the sequences S_{ij} , $1 \leq j < i$.

Claim: The length of S_i, $l(S_i) = l_i$.

Assume otherwise. Let k be the smallest index such that $l(S_k) < l_k$. Let O_k be a sequence of length l_i ending with a_k. Let a_j be the second last element of the sequence O_k. As $j < k$, $l_j = l(S_j)$

$$l(S_k) < l_k = l_j + 1 \Rightarrow l(S_j) < l_j$$

This is a contradiction as $j < k$ and k is the smallest index such that $l(S_k) < l_k$. Thus our claim is true. Clearly the longest monotonically increasing sub-sequence is by definition the longest of the sequences S_i, $1 \leq i \leq n$.

4. You are given n points $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ on the real plane. They have been sorted from left to right and no two points have the same x-coordinate. That means $x_1 < x_2 < ... < x_n$.
A bitonic tour is defined as follows. The tour starts from (x_1, y_1), goes through some intermediate points and reaches (x_n, y_n). Then it goes back to (x_1, y_1) through every one of the rest of the points. All points except (x_1, y_1) are thus visited exactly once. Further, from (x_1, y_1) to (x_n, y_n), you have to keep going right at every step. Similarly, you have to keep going left from (x_n, y_n) to (x_1, y_1). Describe an $O(n^2)$ algorithm to compute the shortest bitonic tour.

Solution: Let $p_j = (x_j, y_j)$ denote the j^{th} point. A shortest bitonic tour can be thought of as a cycle where the vertices are points. Edges connect the points if they are visited one after another. Consider the shortest bitonic tour on the first i points. Observe that such a tour must contain an edge (p_k, p_i) with $k < i-1$. For $k < i-1$, a shortest bitonic tour on the points $p_1, \cdots, p_i$ that contains (p_k, p_i) must be a shortest bitonic tour on the points $p1, \cdots, p_{k+1}$ minus the edge (p_k, p_{k+1}) plus the edge (p_k, p_i) and plus the path $\{(p_{k+1}, p_{k+2}), \cdots (p_{i-1}, p_i)\}$. Consequently k can be chosen such that we end up with the shortest bitonic tour on $p_1, \cdots, p_i$.

The fact that the subproblem on $p_1, \cdots, p_{k+1}$ exhibits the optimal substructure property can be proved by a simple replacement strategy. Suppose we have an optimal solution on $p_1, \cdots, p_i$ points which uses a solution on $p_1, \cdots, p_{k+1}$ and is not optimal. Now by replacing the non-optimal solution on the $p_1, \cdots, p_{k+1}$ points by an optimal solution into the "$p_1, \cdots, p_i$" problem, we obtain a solution which is better than the optimal solution on $p_1, \cdots, p_i$, resulting in a contradiction.

Let $OPT(i)$ be the length of the shortest bitonic tour on the first i points, we can write the following recursion.

$$OPT(i) = \min_{1 \leq k \leq i-2} \{OPT(k+1) + D(k+1, i) + d(k, i) - d(k, k+1)\}$$

for all $3 \leq i \leq n$, where

$$d(i, j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

$$D(i, j) = \sum_{k=i}^{j-1} d(k, k+1)$$

The base cases are: $OPT(1) = 0, OPT(2) = 2 \times d(1, 2)$. $OPT(n)$ is the length of the shortest bitonic tour.

Each step of the iterations costs $O(n)$ and we need to compute n values in total, so the total running time is $O(n^2)$.