CS 670 Spring 2015 - Solutions to HW 2

Problem 15.3-3

Yes, this variant exhibits optimal substructure (for exactly the same reason as the minimization version). Consider a product $A = a_1 a_2 \dots a_n$. Let $A_i^{left} = (a_1 a_2 \dots a_i)$ and $A_i^{right} = (a_{i+1} \dots a_n)$. Let *opt*() denote the maximal number of scalar multiplications that can be done in evaluating the product. Then

$$opt(A) = \max_i \{opt(A_i^{left}) + opt(A_i^{right}) + cols(a_i)rows(a_{i+1})\}$$

Problem 15.4-5

Let $OPT(i)$ be the length of the longest monotonically increasing sub-sequence whose last element is a[i] $(1 \leq i \leq n)$. We have the recurrence

$$OPT(i) = \max_{1 \leq j < i} c(j), (2 \leq i \leq n) \text{ where} \tag{1}$$

$$c(j) = \begin{cases} \text{OPT(j)+1} & \text{if } a[j] \leq a[i] \\ 1 & \text{if } a[j] > a[i] \end{cases}$$

The base case is $OPT(1) = a[1]$.
We can prove the correctness of this recurrence by contradiction. Assume there is a monotonically increasing sub-sequence $s(i)$ longer than $OPT(i)$ whose last element is $a[i]$. The length of $s(i)$ should be larger than 1 because $OPT(i)$ is at least 1. Consider the sub-sequence $s' = s(i) - a[i]$, suppose its last element is $a[k](1 \leq k < i)$. By construction $OPT(i) \geq OPT(k) + 1$, so we have $|s'| > OPT(k)$, which contradicts with the fact that $OPT(k)$ is the length of the longest increasing sub-sequence whose last element is $a[k]$.
After we have computed all the $OPT(1)...OPT(n)$, we can get the longest monotonically increasing sub-sequence by scanning these n numbers and pick the largest one. In every step of the recurrence, record the sub-sequence we chose to maximize $OPT(i)$, finally we can get the longest sub-sequence.
Every step of the recurrence needs to compare $i - 1$ numbers. So it will cost $O(n^2)$ to get the n $OPT(i)$ values. It costs $O(n)$ to pick a largest value from these n $OPT(i)$ values, so the total running time is $O(n^2)$.

Problem 15-3

Assume that we have n points (x_i, y_i). Sort them by increasing x coordinates, so that $x_{i-1} < x_i,\ 1 \leq i < n$. Call p_i the resulting set of points, which are strictly going left to right.

A bitonic tour starts at the leftmost point, goes strictly left to right to the rightmost point, and then goes strictly right to left back to the starting point. Hence, a bitonic tour can be thought of as a cycle where the vertices are points. Edges connect the points if they are 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 $p_1, \ldots, p_i$ that contains (p_k, p_i) must be a shortest bitonic tour on the points $p_1, \ldots, p_{k+1}$ minus the edge (p_k, p_{k+1}) plus the edge (p_k, p_i) and plus the path $\{(p_{k+1}, p_{k+2}), \ldots, (p_{i-1}, p_i)\}$. Consequently k can be chosen such that we end up with the shortest bitonic tour on $p_1, \ldots, p_i$

That the sub-problem on $p_1, \ldots, p_{k+1}$ exhibits the Optimal Substructure can be proved by a typical replacement strategy. Suppose we have an optimal solution on $p_1, \ldots, p_i$ points which uses a solution on $p_1, \ldots, p_{k+1}$ which is not optimal. Now by replacing the non-optimal solution on the $p_1, \ldots, p_{k+1}$ points by an optimal solution into the "$p_1, \ldots, p_i$" problem we obtain a solution which is better than the optimal solution on $p_1, \ldots, p_i$, which is 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

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$
$$D(i,j) = \textstyle\sum_{k=i}^{j-1} d(k, k+1)$$

The base cases are: $OPT(1) = 0, OPT(2) = 2d(1, 2)$. $OPT(n)$ 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 $O(n^2)$.

Problem 15-4

Let *space sum* denote the sum over all lines except the last, of the cubes of the numbers of extra space characters at the ends of lines. Let $OPT(i)$ be the minimum space sum of printing the words from l_i to l_n. So what we want to get is $OPT(1)$.

Consider the first line when we print the words $l_i, l_2, \ldots, l_n$. If we print k words in the first line, the optimum space sum would be the cube of the numbers of extra space characters in the first line plus $OPT(k+1)$. Say we can put at most the first p words from l_i to l_n in a line, that is, $\sum_{t=i}^{p+i-1} l_t + p - 1 \leq M$ and $\sum_{t=1}^{p+i} l_t + p > M$. Suppose the first k words are put in the first line, then the number of extra space characters is

$$s(i,k) := M - k + 1 - \sum_{t=i}^{i+k-1} l_t.$$

So we have the recurrence

$$OPT(i) = \begin{cases} 0 & \text{if } p \geq n - i + 1 \\ \min_{1\leq k\leq p}\{(s(i,k))^3 + OPT(i+k)\} & \text{if } p < n - i + 1 \end{cases}$$

Trace back k's value when $OPT(i)$ achieves minimal, we can get the number of words to be printed in each line. In every recurrence step, we need to compare p values, as $p < M$, so it will cost $O(M)$, we need to compute n $OPT(i)$ values, so the total running time is $O(Mn)$.

Problem 15-7

(a) Start from vertex v_0, do a depth first search to find if there is a path having s as its label. The running time is $O(|V| + |E|)$.

(b) Let $OPT(v, i)$ be the probability of the most probable path starting from v and having label $s_i = \langle \sigma_i, \ldots, \sigma_k \rangle$ $(1 \leq i \leq k)$. Consider the set of neighbor nodes E_v that for each node u in E_v, (v, u) is an edge in the graph whose label is σ_i. If E_v is empty, obviously $OPT(v, i) = 0$. If not, every node in E_v will be a choice to go next in the path, and if we choose node u, the maximum probability will be the probability of edge (v, u) times $OPT(u, i+1)$, so we have the recurrence:

$$OPT(v, i) = \begin{cases} 0 & \text{if } E_v \text{ is empty} \\ \max_{u \in E_v} \{p(v, u) * OPT(u, i+1)\} & \text{if } E_v \text{ is not empty} \end{cases}$$

The base cases are: for every node u in the graph, $OPT(u, k+1) = 1$.
What we want to get is $OPT(v_0, 1)$, trace back the choice of each step when $OPT(v, i)$ achieves optimal, we can get the most probable path.
Suppose we have n nodes in the graph. In each step, it will cost $O(n)$ to get the set E_v, and $O(n)$ to compare different choices. In total we need to compute nk values, so the total running time is $O(n^2 k)$.

Problem 15-10

A strategy consists of $S = \{s_{ij}\}$, where s_{ij} is the amount invested on investment i in year j. Let D_j be the money left at the end of year j. Assume that the fee for each year is paid at the end of the year. A strategy that puts all the money in a single investment every year will be called "pure" (note: The investment picked can be different for different years). Further, without loss of generality assume that $f_1 = 0$ (every strategy pays at least f_1 after every year)

(a): We claim that for every j, there exists a pure strategy S^p such that for every $j' \leq j$ and every strategy S, we have $D^p_{j'} \geq D_{j'}$.

We prove the claim by induction on j. The case $j = 1$ is clearly true (S^p invests all the money in the investment with maximal return)

Induction Hypothesis The claim is true for all $k < j$

Assume there exists a strategy S^{opt} such that $D^{opt}_j > D^p_j$ for every pure strategy S^p.

case 1 S^{opt} moved money at the start of year j. From the hypothesis, there exists a strategy S^p such that $D^p_{j'} \geq D^{opt}_{j'}$ for all $j' \leq j-1$. Extend S_p such that it invests all the money in year j on the investment with maximal return that year. Clearly (they both paid f_2 for year j), $D^{opt}_j \leq D^p_j$ (a contradiction).

case 2 S^{opt} did not move money at the start of year j (Thus $s^{opt}_{ij} = s^{opt}_{i(j-1)} r_{i(j-1)}$)

Observe that

$$\sum_i s^{opt}_{ij} r_{ij} = \sum_i s^{opt}_{i(j-1)} r_{i(j-1)} r_{ij}$$

Consider a new problem with return rates

$$r'_{ik} = r_{ik}, \forall k \leq j-2$$

$$r'_{i(j-1)} = r_{i(j-1)}r_{ij}$$

One this new problem $D^{opt}_{j-1} > D^{p}_{j-1}$ for every pure strategy. This contradicts the induction hypothesis.

Thus no such strategy S_{opt} exists and our claim is true. Thus we infer that there always exists a pure strategy that maximizes the profit.

(b): From part (a) it is clear that it is sufficient to look at only pure strategies. Let S_{ij} be a pure strategy that maximizes the money left after j years (call D^i_j) with the additional constraint that in the year j all the money is invested on i.

We have the following recurrence

$$D^i_j = \max_{i'}\{D^{i'}_{j-1}r_{ij} - f_2\delta_{i,i'}\}$$

$$D^{opt}_j = \max_i D^i_j$$

Here, $\delta_{i,i'} = 0$ if $i = i'$ and 1 otherwise.

(c): Solve the recurrences in part b using dynamic programming with the initial condition the $D^i_1 = 10000r_{i1}$. If n is the number of investments and m the total number of years, then the number of sub-problems is mn. Computing each sub-problem takes $\mathcal{O}(n)$ time. Hence the total running time is $\mathcal{O}(mn^2)$.

(d): When there is a bound on the money that can be put on one investment, a pure strategy might not be possible at all. Hence all the sub-problems and relations derived break down and we no longer have the optimal substructure property(of this form).