CSE 531 Homework Assignment 4

Due in class on Tuesday, Nov 06.

October 20, 2007

There are totally 6 problems, 10 points each. You should do them all. We will grade only 4 problems chosen at my discretion. If it so happens that you don't do one of the problems we don't grade, then no points will be deducted.

Note: this homework is on dynamic programming. To present a dynamic programming algorithm, please conform to the following format: (a) description of the idea and argue the correctness, (b) high-level pseudo code, (c) analysis of time **and** space complexity.

More note: also remember to describe how to construct the optimal solution if the problem specifically asked!

Example 1 (Alice in Disneyland)**.** Alice, a lazy CSE 531 student, plans to go to Disneyland during the winter break. She'd like to go to many events to make it worth the trip. However, being lazy she does not want to walk too far. At the start of each hour, there are many events going on at different locations. Events last for 30 minutes each. The other 30 minutes of each hour is sufficient for her to walk from any place to any place in Disneyland (OK, I admit, this is the only thing I made up). She does not mind sitting at one place for a few hours if it could save her some walking time.

Given a set of n events, their starting hours t_i, the distance $d(i,j)$ between event i and event j (note: $d(i,j) = 0$ if $i = j$), and the number k ($\leq n$) of events which Alice would like to participate in. She wants to schedule visits to events at distinct times, so as to minimize the total distance she must walk between events. Give a dynamic programming algorithm to solve lazy Alice's problem.

Sample Solution. Following the standard format:

(a) **Idea.** Re-order the events so that $t_1 \leq t_2 \leq \cdots \leq t_n$. Also, set $d(i,j) = \infty$ if $t_i = t_j$, namely the cost to attend two events at the same time is prohibitedly high. Let l_j be the distance between the entrance and the location of event j. (You can assume that $l_j = 0$.)

For $1 \leq i \leq k$ and $1 \leq j \leq n$, let l_{ij} be the minimum possible walking distance for Alice to visit exactly i events with j being the last event she visits. We are interested in l_{kn}.

Firstly, note that $l_{1j} = l_j, \forall j$.

Secondly, $l_{ij} = \infty$ when $i > j$ because there is no way to visit more events than the number of possible events up to the jth event.

For $i \geq 2$, consider an optimal walk scheduling for Alice to visit i events with j being the last. Let j' be the next to last event that Alice visits, then clearly $l_{ij} = l_{(i-1)j'} + d(j', j)$. In other words, to visit i events and finish at j, we have to visit $(i-1)$ events, finishing at some $j' < j$, and then go from j' to j. Obviously the walking distance up to j' also have to be optimal, thus when $j \geq i \geq 2$

$$l_{ij} = \min_{1 \leq j' \leq j-1} \left\{ l_{(i-1)j'} + d(j', j) \right\}.$$

In summary,

$$l_{ij} = \begin{cases} \infty & \text{if } j < i \\ l_j & \text{if } j \geq i = 1 \\ \min\limits_{1 \leq j' \leq j-1} \left\{ l_{(i-1)j'} + d(j', j) \right\} & \text{when } j \geq i \geq 2 \end{cases}$$

Since we knew $l_{(i-1)j'} = \infty$ when $j' < (i-1)$, we only need to iterate j' from i to $j-1$. The above definition can be rewritten as

$$l_{ij} = \begin{cases} \infty & \text{if } j < i \\ l_j & \text{if } j \geq i = 1 \\ \min\limits_{i-1 \leq j' \leq j-1} \left(l_{(i-1)j'} + d(j', j) \right) & \text{when } j \geq i \geq 2 \end{cases}$$

When filling out the table $L = (l_{ij})$, we are only interested in the entries at or above the main diagonal ($j \geq i$), since the rest are ∞. We first fill out the first row with l_j's ($i = 1$). Entries from the 2nd row and beyond ($d_{ij}, i \geq 2$) depend on entries on the previous row (row $i-1$) and strictly smaller column number ($j' \leq j-1$), hence the upper half of the table L can be filled out from left to right, top to bottom. Note that the table L is of dimension $k \times n$, hence it is not necessarily a square table.

(b) **Pseudo Code.**

LAZY-ALICE(D, n, k)

```
1: // We assume the events are sorted in starting time in advance.
2: for i ← 1 to k do
3:    for j ← i to n do
4:       if i = 1 then
5:          l_ij ← l_j
6:       else
7:          l_ij ← min_{i-1 ≤ j' ≤ j-1} (l_{(i-1)j'} + d(j', j))
8:       end if
9:    end for
10: end for
11: Return l_kn
```

Although we did not do it here, you should think about how to construct an optimal sequence of events to visit.

(c) **Analysis.**

As far as time is concerned, it takes $O(n \lg n)$ for the sorting, and then $O(kn^2)$ for the three loops we have (the outer loop is of size k, and two inner loops are at most n each). To be precise, the running time after sorting is

$$\Theta(n) + \sum_{i=2}^{k} \sum_{j=i}^{n} \sum_{j'=i-1}^{j-1} \Theta(1) = \Theta(kn^2).$$

Since we had two outer loops of size k and n each, and the min operation of the inner loop takes $O(n)$, we can conclude that the running time is $O(n^2 k)$, for a total of $O(n^2 k)$. Note here that we assume $k \geq 2$, otherwise the total running time for table filling would just be

$\Theta(n)$. In summary, the running time is $O(n \lg n)$ if $k = 1$, and $O(n^2 k)$ if $k \geq 2$. The space requirement is obviously $\Theta(nk)$.

□

Problem 1. A climatologist has a large data set of temperatures recorded daily for more than a century. To study global warming trend, he would like to find a period during which the daily average temperature was increased the most. Specifically, he has an array of average temperatures $\mathbf{t} = [t_1, t_2, \ldots, t_n]$, where t_i is the average temperature of the ith day on record. He would like to find a pair of day (i, j) for which $i < j$ and $t_j - t_i$ is the largest among all such pairs. Help him design an $O(n)$-time algorithm to find such a pair.

Problem 2. As a graduate student, renting cheaply is always a plus. Two landlords Alice and Bob run a pretty weird business model. For each of the next n months, they pre-set their monthly rent prices. In the ith month, Alice's apartments are set at a_i dollars each, while Bob's apartments are set at b_i dollars each. Each of them also allows tenants to specify which months in the next n months they want to sign the lease for. For example, if you want, you can arrange your leases so that you will stay in an Alice's apartment for 3 months, then switch to Bob's for 2 month, and then switch back to Alice's for 1 more, and so on. The drawback, of course, is the moving cost, which is fixed at c dollars. For instance, if you arrange your leases to be

Month number	1	2	3	4	5	6
Rent from	Alice	Alice	Bob	Alice	Bob	Bob

Then, your total cost for the next 6 months is $a_1 + a_2 + c + b_3 + c + a_4 + c + b_5 + b_6$. (We ignore the initial moving-in cost, because that has to be paid no matter where we stay.)

Problem. given the rental prices $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$, along with the moving cost c, determine the cheapest way to sign leases with Bob and Alice. The output is an array $\mathbf{D}$ of n elements, in which each element is either Alice or Bob.

Questions.

(a) Consider the following algorithm:

GREEDY APARTMENT HUNTING

```
for i = 1 to n do
  if a_i < b_i then
    D[i] ← Alice
  else
    D[i] ← Bob
  end if
end for
```

Give an instance for which GREEDY APARTMENT HUNTING does not give an optimal solution. Also indicate what the optimal solution is for that instance.

(b) Devise an efficient algorithm for solving the above problem. The output is not only the cost, but also the array $\mathbf{D}$, i.e. the renting strategy.

Problem 3. Let $\mathbf{A} = (a_1, \ldots, a_n)$ be a sequence of n real numbers, and $m \leq n/2$ be a given integer. We want to find a set $S = \{i_1, \ldots, i_{2k}\}$ of $2k$ integers, where $0 \leq k \leq m$ and $1 \leq i_1 < i_2 < \cdots < i_{2k} \leq n$, so that the following sum is maximized:

$$(a_{i_2} - a_{i_1}) + (a_{i_4} - a_{i_3}) + \cdots + (a_{i_{2k}} - a_{i_{2k-1}})$$

(When $k = 0$, the sequence is empty with "cost" zero.) Devise an efficient algorithm to find the best such set S. The running time should be a polynomial in n and m.

Problem 4. The owners of an independently operated gas station are faced with the following situation. They have a large underground tank in which they store gas; the tank can hold up to L gallons at one time. Ordering gas is quite expensive, so they want to order relatively rarely. For each order, they need to pay a fixed price P for delivery in addition to the cost of the gas ordered. However, it costs c to store a gallon of gas for an extra day, so ordering too much ahead increases the storage cost.

They are planning to close for a week in the winter, and they want their tank to be empty by the time they close. Luckily, based on years of experience, they have accurate projections for how much gas they will need each day until this point in time. Assume that there are n days left until they close, and they need g_i gallons of gas for each of the days $i = 1, \ldots, n$. Assume that the tank is empty at the end of day 0. Give an algorithm to decide on which days they should place orders, and how much to order so as to minimize their total cost.

Problem 5. Let $G = (V, E)$ be a directed graph with given edge costs $c(e)$, $e \in E$. The costs may be positive or negative, but every directed cycle in the graph has strictly positive cost. Given a source node s and a destination node d, devise an efficient algorithm which counts the number of shortest paths from s to d. (No need to list all the shortest paths, just output the number of shortest paths.)

Problem 6. Alice and Bob go to Disneyland. For better or worse, they come up with the following strategy to visit all different event locations $l_0, l_1, \ldots, l_n$:

- They both start at l_0.
- Then, they go separate ways to visit the locations, as long as each location is visited by at least one of them.
- Each person also has to visit his/her locations in increasing index order, namely if Alice visits l_i before l_j then $i < j$. The same holds for Bob.
- Finally, after covering all $l_1, \ldots, l_n$, they both come back to l_0.

Given the distance $d(i, j)$ between event locations l_i and l_j, devise a dynamic programming algorithm to determine a visit schedule for Alice and Bob to minimize their total walking distance.