CS 355: Analysis of Algorithms (Fall 2004)
Homework 4: Dynamic Programming

Instructor (Section A): Mayur Thakur

Handed out 2004/10/21; Due 2004/10/28 9:29am.

Instructions:

- Maximum (regular credit) points: 100.
- Maximum extra credit points: 30.
- There are 7 questions in this homework.
- Answer each question in the space provided.
- You will **NOT** be rewarded for wordiness. At the same time, it might be a good idea to give an intuitive explanation of your proof before a technical full proof. This, however, is not a requirement; it is merely a guideline.
- CLRS stands for the course textbook: "T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. Second edition, McGraw Hill, 2001." **The older (first) edition of the book may not have some problems referred to below and may have different numbers for some problems referred to below.**
- Note that the following are merely answer sketches. Proofs of some claims have been omitted and some claims have been proved only informally.

1. **[EXTRA CREDIT]** Given a sequence X of n real numbers, $S = a_1, a_2, \ldots, a_k$ a *good* subsequence of X if S is a subsequence of X and the following holds: $a_1 \geq a_2 \geq a_3 \geq \ldots \geq a_k$ or $a_1 \leq a_2 \leq a_3 \leq \ldots \leq a_k$.

   (a) (10 points) Describe an efficient algorithm to compute the longest good subsequence of a given sequence of real numbers.

   Answer: Note that the longest good subsequence of a sequence of real numbers can be computed by computing the longest nondecreasing subsequence and computing the longest nonincreasing subsequence and then comparing their lengths. We will describe how to compute the longest nondecreasing subsequence. (The algorithm to compute the longest nonincreasing subsequence is analogous.) We will in fact describe how to compute the *length* of the longest nondecreasing subsequence. Our algorithm can be modified (in an obvious manner) to compute the longest nondecreasing subsequence. Let $A = a_1 a_2 a_3 \ldots a_n$ be a sequence of length n. For each i such that $1 \leq i \leq n$, $\text{len}(i)$ be the length of the longest nondecreasing subsequence in A that ends at i. Then, clearly the length of the longest nondecreasing subsequence in A is simply $\max\{\text{len}(i) \mid 1 \leq i \leq n\}$. The following is a recurrence for $\text{len}(i)$:

$$\text{len}(i) = \begin{cases} 1 & \text{if } i = 1, \text{ and} \\ \max\{\text{len}(j) + 1 \mid a_j \leq a_i \wedge j < i\} & \text{otherwise.} \end{cases}$$

   We can use memoization to turn the above recursive solution into a iterative solution. Roughly speaking, in the iterative (bottom-up) solution, $\text{len}(i)$'s are computed in increasing order of i's. Each computation of $\text{len}(i)$ takes $\mathcal{O}(n)$ time from the above recurrence. Since there are $\mathcal{O}(n)$ such steps, the total time is $\mathcal{O}(n^2)$.

(b) (20 points) Show that the length of the longest good subsequence of a sequence of n real numbers is at least $\sqrt{n}$.

Answer: Let $A = a_1 a_2 \ldots a_n$ be an arbitrary sequence of length n. For each i such that $1 \leq i \leq n$, let $\ell_1(i)$ be the length of the longest nondecreasing subsequence starting at a_i and let $\ell_2(i)$ be the length of the longest nonincreasing subsequence starting at a_i. The crucial point to note is that, for each i and j such that $i < j$, the following holds:

i. $\ell_1(i) = \ell_1(j) \implies \ell_2(i) \neq \ell_2(j)$ and
ii. $\ell_2(i) = \ell_2(j) \implies \ell_1(i) \neq \ell_1(j)$.

Why? (Consider what happens when $a_i \leq a_j$ and what happens when $a_i \geq a_j$.)

Thus, the number of elements in the set $X = \{(\ell_1(i), \ell_2(i)) \mid 1 \leq i \leq n\}$ is exactly n. However, if the length of the longest good subsequence is less than $\sqrt{n}$, then the number of elements in X is less than n (because each $\ell_1(i)$ and $\ell_2(i)$ can independently take less than $\sqrt{n}$ values), which is a contradiction. Thus, the length of the longest good subsequence must be at least $\sqrt{n}$.

2. (20 points) [Problem 15-1 of CLRS].

Answer: We want to compute a bitonic tour through a set of points n in a plane. Assume that the points have distinct x-coordinates. Let p_1, p_2, ..., p_n be the points in order of increasing x-coordinates. We say that u and v are neighbors in a tour T if u and v occur consecutively in T. (For example, if $p_1p_3p_4p_2$ is a tour, then p_1 has neighbors p_2 and p_3.)

For each $1 \leq i \leq n$, define $\text{right}(i) = \{p_i, p_{i+1}, p_{i+2}, \ldots, p_n\}$. That is, $\text{right}(i)$ is the set of points lying to the right of (and including) the point p_i. Given points p_k and p_ℓ such that $k < \ell$, let $\text{len}(p_k, p_\ell)$ be the length of an optimal bitonic *path* that starts at p_k, ends at p_ℓ, and passes through each point in $\text{right}(k)$.

We will give a dynamic programming algorithm for finding an optimal bitonic tour. The dynamic programming algorithm will make use of the following facts, the proofs of which are left as an exercise for the reader:

(a) p_1 and p_2 are neighbors in any bitonic tour through p_1, p_2, ..., p_n. Thus, we can assume w.l.o.g. that p_1 is followed by p_2 in the optimal bitonic tour we are looking for. Thus, the problem now becomes: Find an optimal bitonic *path* that starts at p_1 and ends at p_2 and goes through the following points: p_3, p_4, ..., p_n. Thus, the problem is to find $\text{len}(p_1, p_2)$.

(b) For any k and ℓ such that $k < \ell$, if $\ell > k + 1$, then in any bitonic tour starting at p_k, ending at p_ℓ, and passing through $\text{right}(k)$, p_{k+1} is a neighbor of p_k.

Using these two facts, we can write a recursive solution for $\text{len}(p_\ell, p_k)$ as follows (let $d(\cdot, \cdot)$ be the cartesian distance function):

$$\text{len}(p_k, p_\ell) = \begin{cases} d(p_k, p_\ell) & \text{if } k = n-1 \text{ and } \ell = n, \\ d(p_k, p_{k+1}) + \text{len}(p_{k+1}, p_\ell) & \text{if } \ell > k+1, \\ \min\{d(p_k, p_i) + \text{len}(p_i, p_\ell) \,\big|\, k+2 \leq i \leq n\} & \text{if } \ell = k+1. \end{cases}$$

We can now memoize this recursive solution to obtain an iterative solution that runs in time $\mathcal{O}(n^2)$.

3. (15 points) [Problem 15-2 of CLRS].

Answer: The words have lengths ℓ_1, ℓ_2, ..., ℓ_n. For each i and j such that $i \leq j$, let $s(i,j)$ denote the number of extra spaces when words i through j (both inclusive) are printed on a line.

We will give a recurrence to compute the minimum cost of printing neatly. It can be modified (in the obvious manner) to compute how the words are printing (that is, which words go on which lines). Let $c(i,j)$ denote the minimum cost of printing words i through j (both inclusive). Also, let $m(i)$ denote the largest integer k such that $i \leq k \leq n$ and $s(i,k)$ is nonnegative. Note that the if words i through j occupy more than one line in the solution corresponding to cost $c(i,j)$, then the last word on the first line of the solution could be any one of the following: word i, word $(i+1)$, ..., word $m(i)$. It cannot be any word that comes after word $m(i)$ because that would mean that on the first line we are printing more than M characters, which is illegal. Based on this, we can write the recurrence for $c(i,j)$ as follows:

$$c(i,j) = \begin{cases} 0 & \text{if } m(i) = n \\ \min\{s(i,r)^3 + c(r+1,j) \,\big|\, i \leq r \leq m(i)\} & \text{otherwise.} \end{cases}$$

Clearly, $c(1,n)$ gives us the minimum cost of printing ℓ_1, ℓ_2, ..., ℓ_n neatly. An iterative (bottom-up) implementation of c will compute $c(n,n)$, $c(n-1,n)$, $c(n-2,n)$, ..., $c(1,n)$ in order. This will take $\mathcal{O}(n^2)$ in the worst case.

4. (20 points) [Problem 15-4 of CLRS].

Answer:

Let T be the tree representing the company hierarchy. Let R be the root of T. Let X be an arbitrary node in T. Let $c(X)$ denote the the conviviality of employee X. Let guest(X) denote an optimal selection of guests from among employees in the subtree rooted at X. For any set S of nodes, let $C(S)$ denote the sum of conviviality of guests in S. Let X_1, X_2, ..., $X_k(X)$ denote immediate suboridinates of X.

We want to find guest(R). For any node X, guest(X) is computed as follows. We (recursively) find guest(X_1), guest(X_2), ..., guest(X_k), where X_i's are immediate subordinates of X. If, for each $1 \leq i \leq k$, it holds that $X_i \notin$ guest(X_i), then we let guest(X) $= \bigcup_{i=1}^{k}$ guest(X_i) $\bigcup\{X\}$. (In this case, it is "safe" to add X to the union of guest lists of subtrees rooted at X's subordinates.) Otherwise, let Y_1, Y_2, ..., Y_ℓ be the set of immediate subordinates of X such that, for each $1 \leq j \leq \ell$, guest(Y_j) contains Y_j. For each $1 \leq j \leq \ell$, let G_j be the union of the optimal guest lists of subtrees rooted at Y_j's immediate subordinates. Consider the following two sets:

(a) S_1 consists of X and all the guests in G_1, G_2, ..., G_ℓ, and for each Z in $\{X_1, X_2, \ldots, X_k\} - \{Y_1, Y_2, \ldots, Y_\ell\}$, all the guests in guest(Z).

(b) S_2 consists of all the guests in guest(X_1), guest(X_2), ..., guest(X_k).

Note that S_1 is an optimal solution (that is, set of guests with maximum conviviality rating) for the subtree rooted at X when X is invited to the party, while S_2 is the optimal solution solution for the subtree rooted at X when X is not invited to the party. Note that when X is invited to the party, then none of X's immediate subordinates can be invited.

If $C(S_1) \geq C(S_2)$, then guest(X) $= S_1$, otherwise guest(X) $= S_2$. (The base cases are when X is a leaf node, in which case guest(X) $= \{X\}$.)

Note that in order to implement the abovementioned algorithm in a bottom-up manner, we need to compute guest(X) in a bottom-up manner in the tree representing the company hierarchy.

5. (15 points) [Problem 15-6 of CLRS].

Answer:

Let x and y denote squares on the checkerboard. Let d_1, d_2, ..., d_n denote "destination" squares, that is, squares on the top row. For each square x, let $P(x)$ denote the maximum profit one can make while via a sequence of legal moves from a square on the bottom row to square x. We want to find $\max\{P(d_j) \mid 1 \leq j \leq n\}$. The recurrence relation for $P(x)$ can be described as follows (valid(a, b) is true if and only if going from a to b is a valid move):

$$P(x) = \begin{cases} 0 & \text{if } x \text{ is on the bottom row} \\ \max\{P(z) + p(x, z) \mid \text{valid}(x, z)\} & \text{otherwise.} \end{cases}$$

To compute the maximum profit, it is sufficient to compute $P(d_j)$, for each $j \in \{1, 2, \ldots, n\}$. The bottom-up algorithm will compute $P(x)$ for squares x in rows $n-1$, $n-2$, ..., 2, 1 (in order). It is easy to see that each such $P(\cdot)$ computation takes $\mathcal{O}(1)$ time and the number of such computations is $\mathcal{O}(n^2)$. Thus, the total time taken by the algorithm is $\mathcal{O}(n^2)$.

6. (15 points) [Problem 15-7 of CLRS].

Answer: Note that since we get zero profit for completing a job after its deadline, there is an optimal solution where all jobs whose deadlines are met are scheduled before any job whose deadline is not met. Also note that the profit for meeting a deadline is the same regardless of when the job is scheduled. Thus, if there is a subset X of jobs for which there is a feasible schedule (that is, a schedule in which deadlines for all jobs is met), then the following is a feasible schedule: schedule the jobs in order of increasing deadlines. (Why? Consider that there is a feasible schedule but the "increasing deadline" schedule is not a feasible schedule. In particular, the deadline for the job whose deadline is ranked k among all deadlines in X is not being met in the "increasing deadline" schedule. Then we can easily show that in any schedule, there will be at least one job x among jobs whose deadline occur in the first k deadlines, such that x's deadline is not met.) Thus, we simply need to find out a set of jobs having a feasible schedule such that the total profit of these jobs is maximized.

Sort the jobs in increasing order of their deadlines. Let this sorted order of jobs be a_1, a_2, ..., a_n. $P(j)$ will be the maximum profit from scheduling some of the jobs in a_1, a_2, ..., a_j. (Note that as mentioned above, w.l.o.g. all the jobs scheduled meet their deadlines.) Also, let $S(j)$ be the set of jobs scheduled in the optimal schedule of a_1, a_2, ..., a_j. (Given a set $X \subseteq \{a_1, a_2, \ldots, a_n\}$, X is *feasible* if and only if scheduling jobs in X one after the other in increasing order of the deadlines results in deadlines of all jobs being met. For any $j \in \{2, \ldots, n\}$, let $\alpha(j)$ be the largest integer less than j such that $S(\alpha(j)) \bigcup \{a_j\}$ is feasible. Note that given $S(1)$, $S(2)$, ..., $S(j-1)$, d_j (the deadline for a_j), and t_j (the time takes for a_j), it is easy to find $\alpha(j)$.)

$$P(j) = \begin{cases} p_j & \text{if } j = 1 \wedge t_j \leq d_j, \\ 0 & \text{if } j = 1 \wedge t_j > d_j, \\ \max\{P(S(\alpha(j))) + p_j, P(S(j-1))\} & \text{otherwise.} \end{cases}$$

In a bottom-up implementation of this algorithm, we would compute in order $P(1)$, $S(1)$, $P(2)$, $S(2)$, ..., $P(n)$, $S(n)$. An optimal schedule consists of scheduling the jobs in $S(n)$ in order of their deadlines. Each computation of $P(j)$ would require $\mathcal{O}(n)$ time in the worst case, so the total time is $\mathcal{O}(n^2)$ in the worst case.

7. (15 points) Consider the generalized assembly-line scheduling problem, which is a generalization of the 2 assembly-line scheduling problem mentioned in Chapter 15 of CLRS. In the generelized assembly-line scheduling problem, the input is the following:

   (a) k, the number of assembly lines,

   (b) n, the number of stations on each line, and

   (c) $t_{i,j,\ell}$ (for each i, j, and ℓ such that $1 \leq i \leq k$, $1 \leq j \leq k$, and $1 \leq \ell \leq n-1$), the time it takes to move the car from the i-th line to the j-th line after it has been processed at station ℓ.

   Assume that the entry and exit times for each line are zero. Describe an efficient algorithm that finds the fastest route through the factory consisting of these k assembly lines. What is the asymptotic running time of your algorithm in terms of n and k?

   Answer:

   The solution is a straightforward generalization of algorithm for the 2-line assembly line scheduling problem given in CLRS.