

CS 1510 Midterm 2
Fall 2003

1. (20 points) We consider the problem of computing the longest common subsequence of two sequences $A = a_1, \dots, a_m$ and $B = b_1, \dots, b_n$. Let $T(i, j)$ be the length of the longest common subsequence of a_1, \dots, a_i and b_1, \dots, b_j .
 - (a) Write a recursive function to compute $T(i, j)$ in the naive way. Don't forget the base case(s).
 - (b) Show that if you implement this recursion directly in say the C programming language, that the program could use time that is exponential in n .
 - (c) Write iterative array based code to compute $T(m, n)$ that runs in $O(n^2)$ time.
 - (d) Write code to actually find the longest common subsequence from your array.

2. (20 points) Consider the problem where the input is a collection of n train trips within Germany. For the i th trip T_i you are given the date d_i of that trip, and the non-discounted fare f_i for that trip. For simplicity we will assume that dates are nonnegative integers. The German railway system sells a class A Bahncard for Y Euros that entitles you to a 50% fare reduction on all train travel within Germany within A days of purchase. The German railway system also sells a class B Bahncard for M Euros that entitles you to a 60% fare reduction on all train travel within Germany within B days of purchase. You can apply at most one Bahncard discount to a particular trip. The problem is to determine the least you can spend on your travel.

In this paragraph we give an example. Assume that $Y = 500$, $A = 365$, and $M = 60$ and $B = 30$. Further assume, $d_1 = 1$, $f_1 = 20$ Euros, $d_2 = 40$, $f_2 = 200$ Euros, $d_3 = 80$, $f_3 = 200$ Euros, $d_4 = 120$, $f_4 = 100$ Euros, $d_5 = 200$, $f_5 = 200$ Euros, and $d_6 = 400$, and $f_6 = 600$ Euros. Then you might buy a class B Bahncard on day 40, and a class A Bahncard on day 200. This results in a total cost of

$$20 + 60 + (.4)(200) + 200 + 100 + 500 + (.5)(200) + (.5)(600)$$

Euros.

Give an $O(n^3)$ time algorithm for problem. Significant partial credit will be given for any polynomial time algorithm.

3. (20 points) This is an actual problem that arises in video processing, called temporally consistent assignment. You are given two videos, each represented as a sequence of images. Both videos were taken off of the same scene at roughly the same time period. Let $X = x_1, x_2, \dots, x_m$ be one sequence of images and $Y = y_1, y_2, \dots, y_n$ be the other sequence of images. The cameras are not synchronized and may run at significantly different speeds. The objective is to assign each image of the X video sequence with its most similar image in the Y video sequence, subject to the constraint that the assignments are consistent in time. In order to determine how similar two images are, as part of the input, you may assume that you are given a table $D[i, j]$ which contains a numeric dissimilarity value between these two images x_i and y_j . The lower $D[i, j]$ is, the more similar the images are. Do not worry how $D[i, j]$ is computed. An assignment of sequence X to sequence Y is a sequence of m indices $A = j_1, j_2, \dots, j_m$ meaning that, for $1 \leq i \leq m$, image x_i is assigned to image y_{j_i} . The cost of an assignment is a sum of the dissimilarities between the assigned images. That is, $cost(A) = \sum_{i=1}^m D[i, j_i]$. An assignment A is temporally consistent if $j_i \leq j_{i+1}$, for $1 \leq i < m$. In other words, if x_i is assigned to some image y_{j_i} then the next image in the sequence x_{i+1} must be assigned to an image appearing no earlier than y_{j_i} in the Y image sequence. (This makes sense, as time runs forward for both cameras.) We allow two images of X to be assigned to the same image of Y . The problem is: given the video sequences X, Y and the cost table $D[i, j]$, compute the minimum cost temporally consistent assignment of X to Y . Consider the following table for $m = 3$ and $n = 6$:

		$D[i, j]$					
		$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
$i = 1$		3	7	4	9	1	2
$i = 2$		5	4	2	5	8	6
$i = 3$		6	1	5	8	2	7

If x_1 is assigned to y_1 (that is $j_1 = 1$) and x_2 is assigned to y_3 (that is $j_2 = 3$), and x_3 is assigned to y_6 (that is $j_3 = 6$) then the cost of this assignment is $D[1, 1] + D[2, 3] + D[3, 6] = 3 + 2 + 2 = 7$. Give an $O(n^3)$ time algorithm to compute the lowest achievable cost. Significant partial credit will be given for any algorithm with polynomial running time.