

CS 1510 Midterm 1
Fall 2011

1. (40 points) Consider the following problem. The input is a collection $A = \{a_1, \dots, a_n\}$ of n points on the real line. The problem is to find a minimum cardinality collection S of unit length intervals that cover every point in A .

Another way to think about this same problem is the following. You know a collection of times A that trains will arrive at a station. When a train arrives there must be someone manning the station. Due to union rules, each employee can work at most one hour at the station (and this hour must be consecutive). The problem is to find a scheduling of employees that covers all the times in A and uses the fewest number of employees.

For each of the following two greedy algorithms, prove or disprove that the algorithm correctly solves the problem. Your proof of correctness must use an exchange argument. Make sure it is clear which algorithm you are proving correct, and which algorithm you are proving incorrect.

- (a) Let I be the interval that covers the most number of points in A . Add I to the solution set S . Then recursively continue on the points in A not covered by I .
 - (b) Let a_j be the smallest (leftmost) point in A . Add the interval $I = (a_j, a_j + 1)$ to the solution set S . Then recursively continue on the points in A not covered by I .
2. (20 points) We consider the shortest common supersequence (SCS) problem. Recall that the input to the SCS problem is two strings $A = A_1 \dots A_m$ and $B = B_1 \dots B_n$. And the output is the length of the shortest string that contains both A and B as subsequences.
 - (a) Give simple recursive pseudo-code to compute the length of the shortest common supersequence of two strings A and B .
 - (b) For each n , give strings A and B , each of length n , on which the running time of this recursive code will be polynomial in n . For each n , give strings A and B , each of length n , on which the running time of this recursive code will be exponential in n . In each case, justify your answer from first principles.
 - (c) Let the m by n table T be defined as follows: $T[i, j]$ is the length of the shortest common supersequence of $A_1 \dots A_i$ and $B = B_1 \dots B_j$. Give pseudo-code, which runs in time $O(mn)$, to fill in this table.
 - (d) Show the table T constructed by your code from the previous part for the two strings $A = yxxyx$ and $B = xyxy$.

3. (20 points) The input consists of n intervals $[a_1, b_1], \dots, [a_n, b_n]$ over the real line. The output should be a collection C of non-overlapping intervals such the sum of the lengths of the intervals in C is maximized. Give a $O(n^4)$ time dynamic programming algorithm for this problem. You can get significant partial credit for giving an algorithm that outputs only the aggregate lengths of the intervals in C .

4. (20 points) Give a polynomial time dynamic programming algorithm for the following problem. The input consists of two dimensional array R of non-negative integers, and an integer k . The value $R_{t,p}$ gives the number of of users requesting page p at time t (say from a www server). At each integer time, the server can broadcast an arbitrary collection of pages. If the server broadcasts page p at time t , the the requests of all the users who requested page p strictly before time t are satisfied. The server can make at most k broadcasts. The goal is to pick the k times to broadcast, and the pages to broadcast at those k times, in order to minimize the total time (over all requests) that requests/users have to wait in order to have their requests satisfied.

For example, if $k = 3$ and the array R was:

time	1	2	3
page A	0	2	0
page B	1	3	4

One schedule (which is presumably optimal) is to broadcast pages A and B at time 3, and page B again at time 4. This gives $k=3$ total broadcasts. The waiting time for the 2 page requests to page A at time 2 would be 1. The page request to B at time 1 would wait 2. The 3 page requests to B at time 2 would wait 1. And the 4 page requests to B at time 3 would wait 1. This gives total waiting time $2*1 + 1*2 + 3*1 + 4*1 = 11$.

You can get significant partial credit for giving an algorithm that outputs only the optimal total waiting time. You can get moderate partial credit for giving an algorithm for the special case that there is only one page.