

CS 1510 Midterm 1
Fall 2008

1. (40 points) We consider the following problem:

INPUT: A collection of jobs J_1, \dots, J_n , where the i th job is a 3-tuple (r_i, x_i, d_i) of non-negative integers.

OUTPUT: 1 if there is a preemptive feasible schedule for these jobs on one processor, and 0 otherwise. A schedule is *feasible* if every job J_i is run for x_i time units between its release time r_i and its deadline d_i .

We consider greedy algorithms for solving this problem that schedule times in an online fashion, that is the algorithms are of the following form:

$t = 0$;

while there are jobs left not completely scheduled

 pick a job J_m to schedule at time t ;

 increment t ;

One can get different greedy algorithms depending on how job J_m is selected. For each of the following method of selecting J_m , prove or disprove that the resulting greedy algorithms produce feasible schedules, if they exist for the jobs being considered. Your proof of correctness must use an exchange argument.

- (a) Among those jobs J_i such that $r_i \leq t$, and that have not been scheduled for enough time units, pick J_m to be the job with minimal deadline. Ties may be broken arbitrarily. We call this algorithm EDF.
- (b) Among those jobs such that $r_i \leq t$, and that have not been scheduled for enough time units, pick J_m to be the job that has minimal remaining processing time. Ties may be broken arbitrarily. If a job have been executed for y time units before time t , then its *remaining processing time* is $x_i - y$. We call this algorithm SRPT.

As an example of EDF and SRPT consider the following instance: $J_1 = (0, 100, 1000)$, $J_2 = (10, 10, 100)$, $J_3 = (10, 10, 101)$, and $J_4 = (115, 10, 130)$.

EDF runs J_1 from time 0 through time 10. From time 10 until time 20, EDF runs J_2 because J_2 's deadline, 100, is less than J_1 's deadline, 1000, and less than J_3 's deadline, 101. From time 20 until time 30, EDF runs J_3 because J_3 's deadline, 101, is less than J_1 's deadline, 1000. From time 30 to time 115, EDF runs J_1 . From time 115 to time 125, EDF runs J_4 since J_4 's deadline, 130, is less than J_1 's deadline, 1000. EDF then runs J_1 from time 125 to time 130. Thus for this input, EDF finishes all the jobs by their deadline.

SRPT runs J_1 from time 0 through time 10. From time 10 until time 30, SRPT runs J_2 and J_3 (either can go first) because throughout this time period, J_2 and J_3 's remaining processing times are always equal and less than J_1 's remaining processing time, 90. From time 30 to time 115, SRPT runs J_1 . From time 115 to time 120, SRPT runs J_1 since J_1 's remaining processing time throughout this period is smaller than J_4 's remaining processing time, 10. From time 120 to 130, SRPT runs J_4 . Thus for this input, SRPT finishes all the jobs by their deadline.

2. (20 points) We consider the problem of computing the longest increasing subsequence of a sequence x_1, \dots, x_n of integers. We considered this problem several times during class.
- (a) (10 points) Give a dynamic programming algorithm to compute the length of the longest increasing subsequence in time $O(n^2)$. Make sure to explain where to look in the table for the answer after the table is constructed. Read the next two parts of the question before answering this question.
- (b) (5 points) Show the table/array that your algorithm constructs on the input

12, 18, 24, 1, 15, 2, 3

- (c) (5 points) Explain how to reconstruct the actual shortest increasing subsequence from the filled in table in time $O(n)$.
3. (20 points) The input to this problem is a pair of strings $A = a_1 \dots a_m$ and $B = b_1 \dots b_n$. The goal is to convert A into B as cheaply as possible. The rules are as follows. For a cost of 1 you can delete any letter. For a cost of 2 you can insert a letter in any position. For a cost of 3 you can replace any letter by any other letter. For example, you can convert $A = abcabc$ to $B = abacab$ via the following sequence: $abcabc$ at a cost of 3 can be converted to $abaabc$, which at cost of 1 can be converted to $ababc$, which at cost of 1 can be converted to $abac$, which at cost of 2 can be converted to $abacb$, which at cost of 2 can be converted to $abacab$. Thus the total cost for this conversion would be 9. This is almost surely not the cheapest possible conversion. Give a polynomial time dynamic programming algorithm for this problem.
4. (20 points) The input to this problem consists of an ordered list of n words. The length of the i th word is w_i , that is the i th word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines, this is called a layout. Note that you can not reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L . No line may be longer than L , although it may be shorter. The penalty for having a line of length K is $L - K$. *The total penalty is the average cubed line penalty.* The problem is to find a layout that minimizes the total penalty. Give a polynomial time dynamic programming algorithm for this problem.

For example, if $w_1 = 3$, $w_2 = 4$, $w_3 = 8$, $w_4 = 5$, and $L = 10$, then the total penalty for putting the first two words on the first line, the third word the second line, and the fourth word on the third line would be

$$\left[(10 - (3 + 4))^3 + (10 - 8)^3 + (10 - 5)^3 \right] / 3$$

since there are three lines with line penalties $(10 - (3 + 4))$, $(10 - 8)$ and $(10 - 5)$.

For half credit, you can give an algorithm for the problem where the total penalty is the sum of the cubes of the line penalties. So in the example above, the total penalty is this case would be

$$\left[(10 - (3 + 4))^3 + (10 - 8)^3 + (10 - 5)^3 \right]$$

If you select this option, make sure to state this clearly at the start of your answer. A solution for the case when the objective is the sum of the cubes of the line penalties is a relatively straight-forward modification of the solution to one of the homework problems. If you think that you can just as easily solve the problem where the objective is the average cubed line penalty, you might want to reevaluate.