CS 1510 Midterm 1
Fall 2019

Instructions

- If you are unsure how to proceed at some point, you will receive more partial credit for admitting this, than for proceeding in an incorrect direction.
- If you are asked to provide an algorithm (so this doesn't pertain to questions where you are asked to provide pseudo-code), if your algorithm description is predominantly pseudo-code, and does not contain sufficient English explanation, then you will receive no credit.
- Algorithm descriptions that use an array data structure must contain a statement of the intended meaning of the array entries. For example "The intended meaning of $A[i, j]$ is the length of the longest increasing subsequence of $x_i, \ldots, x_i$ that ends at x_j. " Algorithm descriptions that do not include such a statement, will receive no credit.

1. (25 points) Consider the following problem.

   INPUT: Positive integers $r_1, \ldots, r_n$ and $c_1, \ldots, c_n$.

   OUTPUT: An n by n array A such that for all $i \in [1, n]$ and all $j \in [1, n]$ it is the case that either $A[i, j] = 0$ or $A[i, j] = 1$, for all $i \in [1, n]$ it is the case that $\sum_{j=1}^{n} A_{i,j} = r_i$, and for all $j \in [1, n]$ it is the case that $\sum_{i=1}^{n} A_{i,j} = c_j$. If such an array doesn't exist, the algorithm should output a statement that the instance is infeasible.

   For each of the following two algorithms, either prove that it is incorrect, or prove that it is correct using an exchange argument.

   (a) This greedy algorithm constructs A row by row. Assume that the first $i - 1$ rows have been constructed. Let a_j be the number of 1's in the jth column in the first $i - 1$ rows. Now the r_i columns with with maximum $c_j - a_j$ are assigned 1's in row i, and the rest of the columns are assigned 0's. That is, the columns that still needs the most 1's are given 1's. At the end, if this doesn't lead to a feasible solution, then the algorithm declares the instance is infeasible.

   (b) This greedy algorithm constructs A one entry at a time. Let $\hat{r}_i$ be the number of remaining 1's that need to be placed in row i. So initially $\hat{r}_i$ is equal to r_i, but it will decrease as the algorithm runs, ideally ending at zero. Let $\hat{c}_j$ be the number of remaining 1's that need to be placed in column j. So initially $\hat{c}_j$ is equal to c_j, but it will decrease as the algorithm runs, ideally ending at zero. The algorithm repeats the following:

      > Let (i, j) be an arbitrary pair such that $A[i, j]$ has not been previously set to 1, $\hat{r}_i > 0$, $\hat{c}_j > 0$, and such that $\hat{r}_i + \hat{c}_j$ is maximum; Then $A[i, j]$ is set to 1, and $\hat{r}_i$ and $\hat{c}_j$ are decremented.

      The algorithm terminates and declares success if and when each $\hat{r}_i$ and $\hat{c}_j$ is zero. The algorithm terminates and declares infeasibility if there is a positive $\hat{r}_i$ or a positive $\hat{c}_j$, but it is not possible to find an (i, j) that meets the other conditions.

2. (25 points) Recall that the input to the longest common subsequence (LCS) problem is two strings $A = A_1 \dots A_m$ and $B = B_1 \dots B_n$. The LCS is the longest string $C = C_1, \dots, C_k$ such that C is a subsequence of both A and B.

   (a) Give simple recursive pseudo-code to compute the length of the LCS of two strings A and B.

   (b) If your ran your pseudo-code on an instance in which $n = m = 100$ on an average laptop, will your recursive pseudo-code:

      i. definitely terminate before our sun supernovas,

      ii. maybe terminate before our sun supernovas, or maybe not terminate before our sun supernovas, depending on the structure of the instance, or

      iii. definitely not terminate before our sun supernovas.

      Justify your answer.

   (c) Let the $m+1$ by $n+1$ table T be defined as follows: $T[i, j]$, for $0 \leq i \leq m$ and $0 \leq j \leq n$, is the length of the longest common subsequence of $A_1 \dots A_i$ and $B = B_1 \dots B_j$. Give iterative pseudo-code, which runs in time $O(mn)$, to fill in this this table T.

   (d) Show the table T constructed by your iterative pseudo-code for the two strings $A = xxyy$ and $B = xyxy$.

   (e) Augment your iterative pseudo-code to actually compute the actual LCS C by tracing back through the table T. (Your pseudo-code doesn't have to be particularly formal or close to actual code. Feel free to use English if you feel this communicates the idea more clearly.)

3. (25 points) The Pew Research Center states that

> Republicans and Democrats are more divided along ideological lines — and partisan acrimony is deeper and more extensive — than at any point in recent history. And these trends manifest themselves in myriad ways, both in politics and in everyday life.

This division has even started to manifest itself in the realm of computer data structures. In particular, *The Onion*, America's finest news source, has advocated the use of Leftist Trees, which are binary search trees that dogmatically lean left. More precisely a Leftist Tree is a binary search tree with the property that every nonleaf node has balance factor 1 or 2. The balance factor of a node is the height of its left subtree minus the height of its right subtree.

Consider the problem where the input consists of n keys $K_1, \ldots, K_n$, with $K_1 < K_2 < \ldots, K_n$, and associated probabilities $p_1, \ldots, p_n$. The feasible solutions are leftist trees. The objective is to minimize the expected depth of a key in the leftist tree. Give an algorithm that outputs the optimal objective value (so the least possible expected depth of a key over all leftist trees). You algorithm should run in time $O(n^3)$. Note that you need not output the actual optimal leftist tree, you just need to output the optimal objective value.

4. The input to this problem consists of n words $W_1, \ldots, W_n$ with n associate positive integer lengths $s_1, \ldots, s_n$ and a positive integer X. Conceptually these words form a paragraph that needs to broken into lines. This is a problem that has to be solved within LaTeX. So a feasible solution is a partition of the words into lines $L_1, \ldots L_h$, each of length at most X. Note that there are no constraints on the number h of lines. That is, for all lines L_i it must be the case that $\sum_{W_j \in L_i} s_j \leq X$. The discrepancy d_i of a line L_i is defined to be $X - \sum_{W_j \in L_i} s_j$. Further words can not be reordered. So if $W_i \in L_m$ and $W_k \in L_m$ and $i < j < k$ then $W_j \in L_m$.

   We consider two different problems, in which the objectives are different. Solve at most one of the following two problems. It is your choice which problem you pick. The first problem is easier, but worth less points. The second problem is harder, but worth more points.

   (a) (15 points) In this problem the objective function is to minimize the sum of the cubes of the discrepancies, that is $\min \sum_{i=1}^{h} d_i^3$. So the output for this problem is the optimal objective value (you do not need to compute the actual line breaks). Don't forget to explain how one will extract the objective value from the table. The running time of your algorithm should be $O(nX)$.

   (b) (25 points) In this problem the objective function is to minimize the average cubed discrepancy, that is $\min \sum_{i=1}^{h} d_i^3/h$. So the output for this problem is the optimal objective value (you do not need to compute the actual line breaks). Don't forget to explain how one will extract the objective value from the table. The running time of your algorithm should be $O(n^2X)$.