Algorithms (CSCI-261-03, CSCI-264), Fall 2023/24
Homework 5, due Tuesday, Nov 21, 2023, 11:59pm

Problem 1

An alphabet contains letters A, B, C, D, E, F. The frequencies of the letters are 35%, 20%, 15%, 15%, 8%, and 7%. We know that the Huffman algorithm always outputs an optimal prefix-free code. However, this code is not always unique (obviously we can, for example, switch 0's with 1's and get a different code – but, for some inputs, there are two optimal prefix-free codes that are significantly different). For the purposes of this exercise, we consider two Huffman codes to be different if there exists a letter for which one of the codes assigns a shorter codeword than the other code.

(a) Trace the Huffman algorithm and construct two different Huffman codes for the above input.

(b) Compute the expected codeword length (i.e., the weighted average codeword length) for both codes.

(c) Does there exist a prefix-free code with a smaller expected codeword length? Reason your answer in a sentence or two, or provide such a prefix-free code.

Problem 2

Given is an undirected graph G that represents a map of a town: each vertex corresponds to an intersection and each edge corresponds to a (two-way) road between two intersections. After living happily in this town for years, its inhabitants suddenly realized that they cannot get to all the places in their town! Help them to find the smallest number of roads they need to build to be able to go from any intersection to any other intersection in this town. In other words, find the smallest k such that, after adding k edges to the graph, it becomes connected. Design an $O(m+n)$ algorithm for this problem. (Recall that n is the number of vertices and m the number of edges in G.)

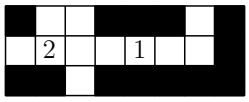
Problem 3

Given are n courses and for each course given are its prerequisites. Let P_i be the set of prerequisite courses for the i-th course and let $m = |P_1| + |P_2| + \ldots + |P_n|$. Give an $O(m+n)$ algorithm that finds the size of the longest prerequisite chain, i.e., the longest sequence of courses for which for every element in the sequence the previous element is its prerequisite. You may assume that the data is consistent, i.e., there are no "prerequisite loops."

Problem 4

Given is an $a \times b$ matrix where every cell corresponds to either an empty space or a wall. This matrix represents a maze and in it are also two Things: Thing One and Thing Two. The Things start at different empty spaces. And, somehow, they got synchronized: They do exactly the same movements—when Thing One goes west, so does Thing Two; when Thing

One goes east, so does Thing Two; and the same happens when they go south or north. They move at the same time: For example, if Thing One is east of Thing Two and they move east, both Things move. The Things cannot go to a location where there is an obstacle (or the other Thing, unless it is moving away). For example, if there is an obstacle east of Thing One but not of Thing Two, if they go east only Thing Two moves. Or, if east of Thing One there is an obstacle and west of it there is Thing Two, if they try to move east, they will stay in their current locations as Thing One is blocked by the obstacle and Thing Two by Thing One.



Things One and Two want to get out of the house, and they want to do so at the very same time (that is, they want to leave the house with the same move). The Things get out of the house if they move north in row 1, south in row a, west in column 1, or east in column b. This is proving to be very tricky. Please help them by designing an $O((ab)^2)$ algorithm that will tell them how to get out in the smallest number of moves (or it will tell them that the task is impossible).

For example, for the 3×8 input in the figure above, where dark entries represent walls and the numbers the starting position of the respective Thing, they can get out in 6 moves, for example: east-north-east-south-north-north.

Problem 5 (CSCI-264 only)

Given is an undirected graph and two of its vertices s and t. Give an $O(n + m)$ algorithm that computes the number of shortest paths from s to t. (For the running time, assume that each basic arithmetic operation (addition, multiplication, etc) takes constant time. Our final counts will fit in `int`, so this is a reasonable assumption.)