ASSIGNMENT 6

DUE: Tuesday November 16, 11:59 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read http://www.student.cs.uwaterloo.ca/~cs341 for general instructions and policies.

Note: “Giving” an algorithm means doing the four parts (i)–(iv) as described on the course web page. This goes without saying for all future assignments and exams.

1. [10 marks] Suppose you have a directed graph and a start vertex \( s \). You want to find paths from \( s \) to every other vertex. Rather than finding shortest paths (i.e., minimizing the sum of the weights of the edges in a path) you want to minimize the maximum weight of an edge in the path. We call these “max-edge shortest paths.” [For example, you are considering buying an electric car and wondering what battery power is good enough for your needs. The vertices of the directed graph are the charging stations, and the weight of an edge is the distance between the charging stations. To travel from your starting vertex \( s \) to some vertex \( v \), you want a route that minimizes the maximum distance between any two consecutive charging stations along the route, since that corresponds to the battery strength you need in order to get from \( s \) to \( v \), assuming you charge the battery at the charging stations.]

   (a) [2 marks] Give a linear time algorithm for the source-target decision version of the problem: given a threshold \( b \), and vertices \( s \) and \( t \), is there a path from \( s \) to \( t \) such that all edges on the path have weight at most \( b \).

       **Hint:** No shortest path algorithms are needed for this, and your answer can be brief.

   (b) [8 marks] Show how to modify Dijkstra’s algorithm to solve the max-edge shortest path problem.

       **Input:** a directed graph with edge weights (possibly negative) and a start vertex \( s \).

       **Output:** the max-edge shortest distance from \( s \) to every vertex \( v \).

       Your pseudocode should be high-level like the version of Dijkstra’s algorithm given in the lectures. In particular, you may assume an implementation of a Priority Queue with operations Find-and-Delete-Min, Insert, Delete, and Decrease-Key each taking \( O(\log k) \) time where \( k \) is the number of elements in the Priority Queue. (Such an implementation involves keeping a link from each edge or vertex of the graph to its location in the heap in order to avoid doing any Find operations. You may ignore these implementation issues.)

       **Note:** There are other algorithms that will solve this problem, but for this assignment you must modify Dijkstra’s algorithm.

2. [10 marks] You are given an array of \( n \) distinct natural numbers \( A[1..n] \), each containing \( k \) digits, where the most significant digit is in \( \{1..9\} \). Your goal is to find an optimal path from \( A[1] \) to \( A[n] \). What do we mean by a path? You can move from \( A[i] \) to \( A[j] \) if they differ in one digit, and you may only move from numbers in \( A \) to other numbers in \( A \).

   So, for example, if \( A = [1234, 4321, 3234, 5274, 3274, 5074] \), you cannot move directly from 1234 to 5074, because they differ in three digits: \( 1 \neq 5, 2 \neq 0, 4 \neq 7 \). Similarly, you cannot
move from 1234 to 4321 since they differ in four digits. However, there is a path from $A[1]$ to $A[n]$, namely, 1234 to 3234 to 3274 to 5274 to 5074.

What do we mean by optimal? Moving from $A[i]$ to $A[j]$ costs $2A[j] - A[i]$. An optimal path has the smallest total cost. Note that the cost might actually be negative (representing profit). In the example above, the stated path is also optimal.

The output should consist of entries in $A[1..n]$ that form an optimal path from $A[1]$ to $A[n]$. You can return any reasonable data structure (e.g., array/list); Just be clear about what you are doing. If there is no path from $A[1]$ to $A[n]$, you can return IMPOSSIBLE.

You may assume that you can compare two strings, or convert between integer and string representations, in $O(k)$ time. You may call graph algorithms described in class, and you may make (and clearly state) any reasonable input/output assumptions for such algorithms. You can write helper functions if you like. You can use simple data structures like arrays, lists, stacks, queues, etc. Clearly state your assumptions regarding their capabilities.

(a) [1 mark] Write an expression for the cost of a path starting and ending at the same number, in terms of $A[1..n]$.

(b) [6 marks] Give a high level description and pseudocode (at a higher level than actual code!) showing how to solve this problem using a graph. Be sure to specify whether your graph is directed or undirected, and describe your nodes, edges and weights. We recommend building adjacency lists, but you are free to build an adjacency matrix, or even both representations.

(c) [2 marks] Argue correctness for your algorithm. In particular, explain why any algorithms you use are applicable to the problem, and why the correct answer is returned. (There exists a two sentence answer warranting full marks.)

(d) [1 mark] Analyze the runtime of your algorithm in terms of $k$ and $n$ using big-O notation.

**Challenge Question.** This is for fun and enrichment only. Do not hand it in.

1. In the lectures, you saw how to test if a directed graph has a negative weight cycle. Show how to find a negative weight cycle if one exists. Can you find a simple negative weight cycle?