EC330 Applied Algorithms and Data Structures for Engineers
Spring 2022

Homework 7

Out: April 22, 2022
Due: May 4, 2022

This homework has a written part and a programming part. Both are due at 11:59 pm on May 4. You should submit both parts on Gradescope.

Problem 1 and 2 should be completed individually, whereas Problem 3 and 4 can be done in pairs. See course syllabus for policy on collaboration.

This assignment has a total of 120 points. Any points beyond 100 will be considered as bonus points, i.e. a maximum of 20 bonus points.

1. **Single-Source Shortest Path [20 pt]**
   Explain how to modify Bellman-Ford's algorithm to efficiently count the *number* of shortest paths (of equal length) from a source vertex to every other vertex in the graph. You can write the modified algorithm in pseudo code similar to the ones I gave in recent lectures, or write actual C++ code.

2. **All Paths from Source to Destination [20 pt]**
   Given a *directed, acyclic* graph G with n nodes, a source node s and a destination node t in G, how many possible paths can there be from s to t in the worst case? Given your answer in $\boldsymbol{O}$. Describe an algorithm that counts the total number of paths from s to t *efficiently*. The complexity of your algorithm should be asymptotically faster than enumerating all the paths (i.e. the $\boldsymbol{O}$ answer that you gave earlier on the number of possible paths from s to t). You can write the modified algorithm in pseudo code similar to the ones I gave in recent lectures, or write actual C++ code.

3. **Zoo Escape! [40 pt]**
   We are going to help Alex escape from the Central Park Zoo. Alex[1] is lost somewhere in the middle of the zoo and we want to help him find the quickest way out!

   The zoo is modeled as a $m \times n$ grid M, represented by `vector<vector<bool>>`, where $M[i][j] = 0$ iff cell $M[i][j]$ is free (i.e. a cell that Alex can occupy or move to).

[1] Here is a picture of Alex:
https://hero.fandom.com/wiki/Alex_(Madagascar)?file=Alex+The+Lion+%28Roar+Posing%29.png .

$M[i][j]$ is the cell in the i^{th} row and the j^{th} column of M. The top left cell has a coordinate of (0, 0).
The challenge we face is that Alex is limping due to an injured leg so he *can only walk straight or walk left. He cannot turn in place*. For instance, if Alex is in cell $M[2][2]$ and facing north, he cannot turn to face west or east on the same spot. However, if the cell $M[2][1]$ is free, then he can walk left to $M[2][1]$ and ends up facing west in $M[2][1]$ in a single move. Alternatively, if $M[1][2]$ is free, he can walk straight to $M[1][2]$ and remains facing north also in one move.

Write an efficient algorithm that finds the shortest escape route for Alex to escape from the zoo. You are given the initial coordinate of Alex as well as the direction that he is facing. You can assume that Alex can escape if he reaches a free cell on the periphery of the zoo. Below is an example.

x	x	x	x
x		↑	x
x	x		x

Alex is initially in coordinate (2, 2) and is facing north, as indicated by the arrow. The squares marked with 'x' are blocked (i.e. not free). Observe that Alex can escape the zoo by taking the following sequence of actions: *go_straight* → *go_left* → *go_straight*, as shown by blue arrows below. However, he cannot escape the zoo with the sequence *go_straight* → *go_right* since he can only go straight or go left.

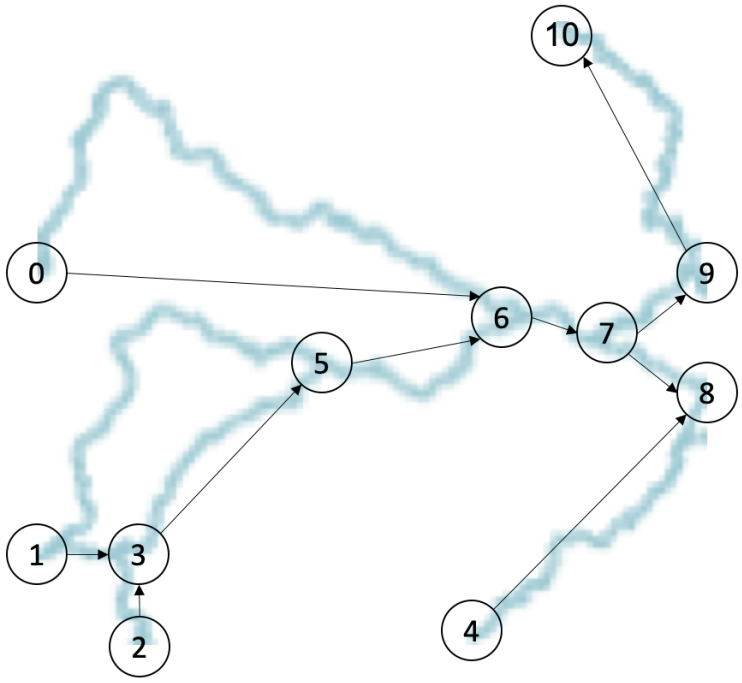
x	x	x	x
←	←	↑	
x		↑	x
x	x		x

Implement the `escape_route` method in `zoo_escape.cpp` which takes inputs the zoo model M, the initial coordinate of Alex, the initial direction that he is facing, and returns the *shortest* sequence of actions (stored in a vector with the first action at index 0) that will allow Alex to escape the zoo. The method should return an empty vector if it is impossible to escape the zoo. You can assume that the Alex does not start on the periphery of the zoo. Submit only the `.cpp` file on Gradescope.

Hint: This is obviously a graph problem so you will want to think about how to construct the graph first, that is, what the vertices and edges are.

4. River Rafting [40 pt]

Ryan decides to go river rafting in the Yosemite National Park. We are going to model the river network in Yosemite as an *unweighted, directed, acyclic* graph, such as the one given below.



Nodes represent starting points, end points, forking points, or converging points in the river network. There is an edge from node *u* to node *v* if there is a water stream that flows directly from *u* to *v*. The nodes are numbered from *0* to *n-1* for a graph with *n* nodes.

a) **[20 pt]** Given a target node *t*, determine all the starting points (nodes with an in-degree of 0) that Ryan can start at so that he can reach *t*. You can assume that the raft cannot go up against the water stream but Ryan can decide which way to go at any forking point.

Implement your algorithm in the `start` function in `river.cpp`. The inputs to `start` are an adjacency matrix representation *r* of the graph (*r*[*i*][*j*] = 1 iff there is an edge from node *i* to node *j*) and the number of the target node *t*. You should output the starting points in *increasing order* of their node numbers.

As an example, in the river network (r1) above, if the target node is 7, then `start` should return [0, 1, 2]. If the target node is 8, then `start` should return [0, 1, 2, 4].

Your algorithm should run in time $\boldsymbol{O}(n^2)$ or better.

b) **[20 pt]** Mira decides to join Ryan in this rafting adventure. Given two nodes that represent the initial locations of Ryan and Mira in the river network respectively, determine efficiently all the *earliest possible meeting points* (EPMs) that they can meet in the river. If a node is an EPM, then it cannot have a predecessor that is also an EPM.

Implement your algorithm in the `meet` function in `river.cpp`. It should output the EPMs in *increasing order* of their node numbers.

For the graph (r2) below, if Ryan starts at 0 and Mira starts at 1, then `meet` should return [4], since Ryan can go from 0 through 2 to 4 and Mira can go from 1 to 4. However, node 7 is not an EPM even though Ryan and Mira can meet at 7 because 4 is a predecessor of 7.
If there is no EPM (e.g. Ryan starting at 0 and Mira starting at 8), then `meet` should output an empty vector.

Note that one of them can *wait* for the other person to come down the river to meet. For example, if Ryan starts at 1 and Mira starts at 3, then `meet` should return [3].

Your algorithm should run in time $\boldsymbol{O}(n^2)$ or better.