CSE 4810 – Algorithm Engineering Lab

Lab 5 | Tasks | Topic: Graphs

1.

a) Given an **m x n** 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water. Hint: use DFS or BFS

b) Analyze the complexity of your solution.

Example 1:

Input: grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

Output: 1

Example 2:

Input: grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

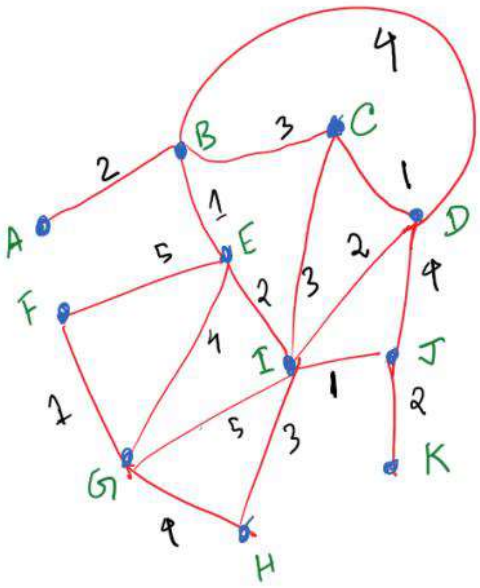
["0","0","0","1","1"]

]

Output: 3

2.

This problem is based on Dijkstra's algorithm. Dijkstra's algorithm is used to calculate the shortest path between two nodes in a graph. It works for both directed and undirected graphs. It is usually used for weighted graphs (in unweighted graphs, Breadth First Search would be more efficient). To recap, Dijkdtra's algorithm does the following:



(i) Record the shortest distance of each node from the source in an array (lets call this array dis).

(ii) dis_{source} is set to 0. For all other node i, dis_i is set to ∞.

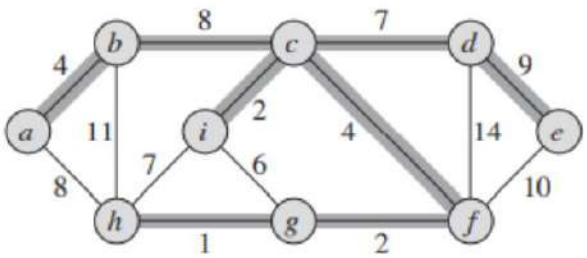
(iii) Among the unvisited nodes, the one with shortest distance from source is chosen. Let this node be u. Now for each adjacent node v, if $dis_v > dis_u + weight_{uv}$, then dis_v assigned to be $dis_u + weight_{uv}$.

Can you modify this algorithm to find the **second** shortest path from the source (A) to the destination (K)? Notice that just finding the path in the given graph will not suffice. You need to define an algorithm that works for any bidirectional weighted graph. The algorithm needs to work in $O(nlogn)$ time where n is the number of edges. Provide enough logical analysis to convince the reader that this solution is indeed the correct one. The second-best path is the shortest path whose length is longer than the shortest path(s) (i.e. if two or more shortest paths exist, the second-shortest path is the one whose length is longer than those but no longer than any other path). Follow this link if you want to check your solution by testing it on a dataset.

Link: https://lightoj.com/problem/not-the-best

3.

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. There are 2 very popular algorithms to calcualte MSTs of a graph. These algorithms are Kruskal's algorithm and Prim's algorithm.



(i) Can you come up with a technique to find the second best MST? Write the algorithm steps and necessary supporting logic. Analyze your algorithms time complexity in terms of the number of vertices V and number of edges E.

(ii) Can a Minimum Spanning Tree finding algorithm like Prim's algorithms be used to find out the Maximum Spanning Tree? If yes, then explain your steps. If not, explain why not with logic.

4.

a) There are a total of ***numCourses*** courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where ***prerequisites[i] = [ai, bi]*** indicates that you must take course **bi** first if you want to take course **ai**. For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1. Return true if you can finish all courses. Otherwise, return false.

b) Analyze the complexity of your solution.

Example 1:

Input: numCourses = 2, prerequisites = [[1,0]]

Output: true

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0. So it is possible.

Example 2:

Input: numCourses = 2, prerequisites = [[1,0],[0,1]]

Output: false

Explanation: There are a total of 2 courses to take.

To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.