CS2020: Data Structures and Algorithms (Accelerated)

Problems 16–17

Released: Tuesday, March 15th 2011, Due: Wednesday, March 23th 2011, 13:59

Overview. You have two programming tasks this week. Both are graph problems that are solvable with the knowledge that you have from Lecture 13-17 :).

Collaboration Policy. As always, you are encouraged to work with other students on solving these problems. However, you **must** write up your solution **by yourself**. In addition, when you write up your solution, you **must** list the names of every collaborator, that is, every other person that you talked to about the problem (even if you only discussed it briefly). Any deviation from this policy will be considered cheating, and will be punished severely, including referral to the NUS Board of Discipline.

Problem 16. (MST)

In Lecture15, Steven mentioned that one of the motivating example of MST problem is about building roads in rural villages. You are going to do exactly that in this task.

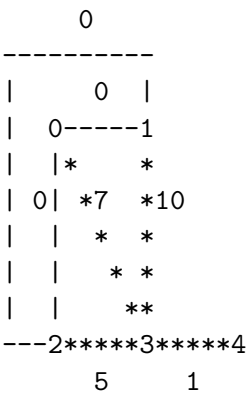
You are given the description of **N** villages, **M** roads that are *already exist* between some of these **N** villages, and another **K** additional roads that *may be built*.

We do not have to (re-)built the existing **M** roads, so their construction cost is 0. Note that the existing **M** roads form a *general graph*.

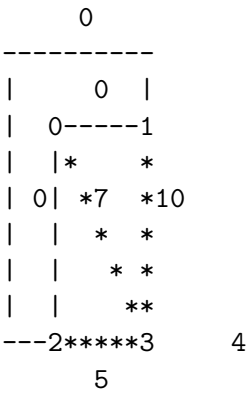
Each of the **K** additional roads is associated with a construction cost (a positive integer).

Your task is to determine what is the minimum cost to make all these **N** villages connected. If it is impossible to make all the **N** villages connected, output -1 (to say 'impossible').

Example 1: Suppose we have **N = 5** villages with **3** existing roads (denoted with —). Those existing roads are **0–1**, **0–2**, and **1–2**. Now, there are **K = 4** additional roads (denoted with ***) that may be built: road **2-3** with cost 5, **0-3** with cost 7, **1-3** with cost 10, and **3-4** with cost 1. For this example, we should choose road **2-3** with cost 5 and road **3-4** with cost 1 because selecting these two will make village 0-1-2-3-4 connected with minimum cost = 5+1 = 6.



Example 2: Suppose we have the same **N = 5** villages as above, but now we only have **K = 3** additional roads: road **2-3** with cost 5, **0-3** with cost 7, and **1-3** with cost 10. For this test case, no matter which additional roads that we choose to build, we will never be able to connect village 4 with the rest. Therefore, we have to output -1.



To get you started, please answer the following questions. You have to write your answers as comments in `MST.java`.

1. What is your strategy to deal with the **M** existing roads with cost 0?
   Remember that these **M** existing roads form a *general graph*!
2. How to determine whether it is possible to build a connected MST?
   Hint: This part is discussed in DG6.
3. Which algorithm that you use to solve this 'modified' MST: Prim's or Kruskal's?
   Why do you choose that particular algorithm?

Implementation: `MST.java`
Remember: never ever touch the `main` method to avoid unnecessary input/output errors.
The skeleton code (`MST.java`) written by Steven can already read in the villages and roads information and output the result in the correct format. You have to implement the following method inside (`MST.java`): `int minCost()`. The requirements for this method is explained in `MST.java`. Class 'iii' is used inside `MST.java` and it is used to model **Integer triple**. Class 'UnionFind' is also used inside `MST.java` in case you choose to implement Kruskal's algorithm (this data structure is discussed in Lecture15). You can use Java 'PriorityQueue' in case you choose to implement Prim's algorithm.

Problem 17. (Maze Navigation - Version 1)

We will explore an exciting (it is) implicit graph problem on 2-D grid (a.k.a. maze). In this maze, character '#' represents a wall (not pass-able), '.' represents a pass-able cell, '!' denotes your starting position, and finally 'E' denotes the exit. There is only one '!' in the maze but there can be *more than one* 'E' in the maze. There will be no '!' and '.' on the boundary of the maze.

You can move from your current cell A to another cell B if and only if cell B is either at the north/east/south/west of cell A.

The maze has size `R` $\times$ `C` where $2 \leq$ `R, C` ≤ 10. Now, your task is to identify the *nearest exit* and the number of steps to reach that nearest exit. The mazes will be designed in such a way that there will be *only one* unique nearest exit, if any. If there is no way you can reach a cell with character 'E', you have to output -1 (to say 'impossible') as your answer.

Several examples:

```
maze1 - there is only 1 path from '!'  to 'E' at row 1, column 0
(notice that we use zero based indexing here).  That path needs 17 steps

##########
E........#
########.#
#!.......#
##########

maze2 - you are trapped, output -1 (impossible)

##########
E........#
##########
#!.......#
##########

maze3 - there are 2 reachable exits, 'E' on row 3, column 9 is nearer:  8 steps

##########
E........#
########.#
#!.......E
##########

maze4 - also 2 reachable exits, 'E' on row 1, column 0 is nearer:  3 steps

##########
E........#
#.######.#
#!.......E
##########
```

To get you started, please answer the following questions. You have to write your answers as comments in `Maze1.java`.

1. Graph Modeling

   (a) What are the vertices and the edges of this implicit graph?

   (b) How many edges and how many vertices are there in an `R` $\times$ `C` maze?
   You do not have to count the precise number! Use big O notation in terms of R and C.

   (c) What is the graph problem that we are trying to solve here?

2. Graph Algorithm
   Remember: if we can reach more than one possible exits, we must select the *nearer one*.
   If there are several possible algorithms, choose the one with the fastest asymptotic running time. Marks will be deducted for implementing the inferior/wrong algorithm.
   There is no need to give formal proof why a certain algorithm can or cannot be used.
   Few logical sentences should be enough.

   (a) Can you use $O(V + E)$ Depth First Search (DFS) to solve this problem? Why?

   (b) Can you use $O(V + E)$ Breadth First Search (BFS) to solve this problem? Why?

   (c) Can you use $O(E \log V)$ Kruskal's algorithm to solve this problem? Why?

   (d) Can you use $O(VE)$ Bellman Ford's or $O((V + E) \log V)$ Dijkstra's algorithm to solve this problem? Why?

3. Implementation: `Maze1.java`
   Remember: never ever touch the `main` method to avoid unnecessary input/output errors.
   The skeleton code written by Steven can already read in the maze(s) and output the result in the correct format. You have to implement the following method: `iii nearestExit()`. The requirements for this method is explained in `Maze1.java`. As discussed in the lecture, class 'iii' is used to model **Integer triple**.