Due : Dec. 10 (12/10), 13:00
Late submission due: Dec. 11 (12/11), 13:00

Overview

This assignment contains two parts, implementing a directed graph and topological soring.

General Notes

- *Read this homework guideline carefully.* If you do not follow the guidelines, you may receive a 0 regardless of whether your code works or not.
- Do not use any IDEs (Eclipse, IntelliJ IDEA, etc.)
  - We recommend Sublime Text (Linux/Mac/Windows), Atom (Linux /Mac/Windows), Notepad++ (Windows), or TextWrangler (Mac).
  - IDEs often create a "package" of your code, which breaks the autograder.
  - **If you know how to fix the package problem**, you can use any IDE you want. However, we will not answer any questions related to this problem since we have already recommended a solution.
- Do not change any method or class signatures. If your code changes any class or method names or signatures, you will receive an automatic 0.
- Make sure your code compiles. Non-compiling code will automatically receive a 0. If you have a problem that is causing you to not be able to compile, it may be better to just comment out the incorrect code and return a dummy value (something like null or -1) so the rest can compile.
- To ensure that your code will be accepted by the autograder, you should submit your code on YSCEC, download it again, recompile it and check the provided test suite. This way, you know that the file you are submitting is the correct one.
- You can use any course materials. However, if you do not cite the source you referred to, it might be checked as copied code. Please write the material (and page) you referred in comments.

1 Graph

A graph G is a pair of vertices V and edges E.

1.1 Adjacency Matrix

An adjacency matrix is one of many ways of representing a graph. It is a matrix with rows and columns labeled by graph vertices, and each cell (i, j) represents the positive weights (> 0) of the edge from v_i to v_j. The cell is 0 if the corresponding edge does not exist.

directed graph in a `txt` file, and you must return the graph as an adjacency matrix after performing some methods. The file will have the number of vertices N and the number of edges M on the first line separated by whitespace. The vertices are labeled from 0 to $N - 1$. Then on the following M lines, each line has vertices v_i,v_j and the weight w_{ij} separated by whitespace representing that there is an edge from v_i to v_j with weight w_{ij}.

You must implement the following methods as well as its constructor in Graph.java:

`constructor`: Given the name of a `txt` file, construct the corresponding graph. Think as performing `insertVertex` and `insertEdge` for the initial condition. The `txt` file's name will be given as a relative path from the working directory.

`insertVertex()`: Insert a new vertex as the last vertex in the graph.

`deleteVertex(`n`)`: Given a vertex n, delete the vertex n and the edges connected to n from the graph. The remaining vertices are relabeled. For example, if 1 is deleted from $\{0, 1, 2, 3\}$, $2, 3$ is relabeled to $1, 2$ respectively. If the vertex n is not in the graph, do nothing.

`insertEdge(`u, v, w`)`: Given a start vertex u and an end vertex v and weight w, insert the given edge to the graph. If the edge is already in the graph, update the weight.

`deleteEdge(`u, v`)`: Given a start vertex u and an end vertex v, delete the given edge from the graph. If the edge does not exist, do nothing.

`matrix()`: Return the adjacency matrix for the graph.

`size()`: Return the number of vertices in the graph.

1.2 Strong Connectivity

A graph G is strongly connected if for any two vertices u and v of G, u reaches v and v reaches u. In other words, each vertex in G can reach all vertices in G. Strongly connected components are maximal subgraphs that are strongly connected.

You must implement the following methods as well as its constructor in SCC.java:

`path(`G, u, v`)`: Given a graph G from section 1.1, and two vertices u, v return `true` if there is a path from u to v. Otherwise, return `false`.

`connectivity(`G`)`: Return the number of strongly connected components in G.

1.3 Shortest Path and Minimum Spanning Tree

Given an *undirected* graph G', a minimum spanning tree of G' is the subset of edges that connects all the vertices without any cycles and with the minimum possible total edge weight.

Define an undirected graph G' from a directed graph G by ignoring the directions of each edge in G. If edges (u, v) and (v, u) both exist in G, then the weight of the undirected edge (u, v) of G' is the sum of edge weights.

You must implement the following methods as well as its constructor in MST.java:

`shortestPath(`G, u, v`)`: Given a directed graph G, and two vertices u and v, return the total weight of the shortest path going from u to v. If the path does not exist, return -1.

`findMST(`G`)`: Given a graph G find the minimum spanning tree of G'. Return the total edge weight of the minimum spanning tree. If you cannot make a minimum spanning tree, return -1.

General Directions

- Write your name and student ID number in the comment at the top of the files you submit.
- Implement all of the required methods.
- You should not import anything that is not already included in the file.
- Pay careful attention to the required return type and edge cases.
- All the codes we provide can be found in src/base directory. If you are unsure what a class/method exactly does, please refer to the code.
- You are free to implement any algorithm that you wish, but you must code it yourself. If you referred to any course materials, you must write the name of the material and page in comments. We will only be testing that your code produces a correct result and terminates in a reasonable amount of time.

Submission Procedure

You *must* make a zip file for submission using Gradle build tool (refer to Compiling section). For this assignment, the zip file will contain only the following three files:

- Graph.java
- MST.java
- SCC.java
- your_student_id_number.txt

You must rename 2021xxxxxx.txt to your actual student ID number. Inside of that text file, you must include the following text and write your name at the bottom. Please be sure to write all the following text including the last period.

In completing this assignment, I pledge that I have not given nor received any unauthorized assistance.

If this file is missing, you will get 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2021123456.txt* is correct while something like *2021123456_pa8.txt* will receive 0.

Compiling

This assignment uses Gradle build tool to automate compiling and testing procedure. The following command will test your Java code against the provided testcases:

```
% ./gradlew -q runTestRunner
```

The following command will zip the files for your submission. The zip file will be named with your student id (the name of .txt file) and will lie in “build” directory. Be aware that the command will be interrupted if your pledge does not comply the guideline.

```
% ./gradlew -q zipSubmission
```

Since the testrunner blocks the standard output from printing, it is hard to test your code fragment while writing the code. For this purpose, we also provide an empty Main class. As this file is not for submission, you may use any features Java provides. The following command will run the Main class instead of the testcases.

```
% ./gradlew -q --console=plain runMain
```

On Windows, try `gradlew.bat` instead of `./gradlew` if you met an error. Moreover, you may omit the ‘`-q`’ option to review the compile log.

Testing

We have provided small testcases (src/test) for you to check your code. You can test your code by the means mentioned above.

Note that the testcases we will use to grade your code is `very much more` rigorous than the one provided here (and not necessarily a superset of the provided tests). You should consider making your own test cases to check your code more thoroughly.