Due: Dec. 18th (12/18), 13:00

Overview

This assignment is to implement a weighted, undirected graph.

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), 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 your code does not 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 LearnUS, download it again, recompile it and check the provided test suite. This way, you know the file you are submitting is correct.
- If you use any code from the textbook, make a reference to the textbook through comments. If your code is flagged as copied and does not have a legitimate reference, you may get a 0 for plagiarism.

Graphs

In this assignment, you will implement an undirected, weighted graph and several methods related to graphs. You will implement the following methods:

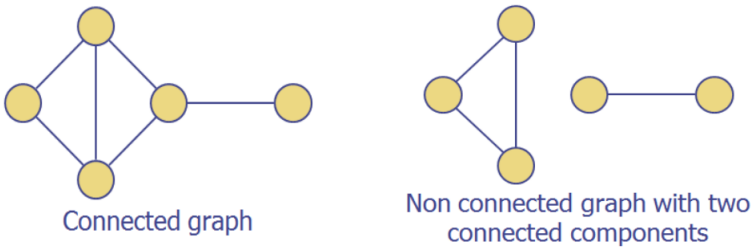
- `Node addNode()`: adds a new node to the graph. Each node in the graph has a label from 0 to $|V| - 1$, in the order of instantiation. Note that you should give labels to each node.
- `Edge addEdge(Node u, Node v, int weight)`: adds an edge with the given endpoints and weight to the graph.
- `int getNumNodes()`: returns the number of nodes in the graph.
- `int getNumEdges()`: returns the number of edges in the graph.
- `ArrayList<Edge> minSpanningTree()`: returns the list of edges in a minimum spanning tree of the graph. If there are multiple minimum spanning trees, you may return any of them. The method should run in $O((|V| + |E|) \log |V|)$ time.
- `int minSpanningTreeWeight()`: returns the weight of a minimum spanning tree of the graph. The method should run in $O((|V| + |E|) \log |V|)$ time.
- `boolean areUVConnected(Node u, Node v)`: checks if there is a path between nodes u and v. The method should run in $O(|V| + |E|)$ time.
- `boolean isConnected()`: checks if the graph is connected (meaning there is only one connected component). The method should run in $O(|V| + |E|)$ time.
- `int numConnectedComponents()`: returns the number of connected components of the graph. The method should run in $O(|V| + |E|)$ time.
- `HashMap<Node, Integer> dijkstra(Node source)`: runs Dijkstra's algorithm on the graph with the specified source node. The method should have the same time complexity as Dijkstra's algorithm. The returned HashMap should have each `Node` instance in the graph as key

and the distance from `source` as the value. Unreachable nodes should have distance ∞.

- `int shortestPathLength(Node source, Node target)`: returns the length of the shortest path between the specified nodes. Returns ∞ if the target is unreachable from the source. The method should have the same time complexity as Dijkstra's algorithm.

Definition

A connected graph is a graph where every pair of nodes has a path between them. Note that an isolated node (a node with no adjacent nodes) is a connected component by itself.



Implementation Notes

- We will always use valid graphs for each problem, so, in general, you do not have to worry about edge cases from invalid graphs.
- We will only be adding nodes and edges to the graph, so you do not have to worry about removing and relabeling the nodes.
- You do not have to worry about edge cases where the graph is empty.
- We will only have non-negative (≥ 0) integers as edge weights, so you can always run Dijkstra's algorithm.

- You do not need to consider the case of adding an edge with an endpoint that does not exist in the graph.
- You can use `Integer.MAX_VALUE` to represent ∞. We will not have test cases where the calculation overflows Java's `int`.
- Be sure to follow the guidelines on labeling the nodes in the `Graph.nodes` instance variable.
- The adjacency list is of type `HashMap<Node, HashMap<Node, Edge>>`, which means if you call `adjacencyList.get(u)`, it will return an object of type `HashMap<Node, Edge>`. To get the edge between nodes u and v (if it exists), you can write `adjacencyList.get(u).get(v)`.
- We imported `HashMap, PriorityQueue, ArrayList, HashSet` and `Iterator` for you to use. You may not import anything else.
- We have already implemented `Node` and `Edge` for you.

Directions

- Write your name and student ID number in the comment at the top of the Graph.java file.
- Implement all of the required functions in the Graph class.
- You are allowed to add any instance variables and methods that you want. However, you may not change any existing ones.
- Pay careful attention to the required return types.

General Directions

- Write your name and student ID number in the comment at the top of the files in src/main directory.
- 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 types and edge cases.
- All the codes we provided can be found in src/base directory. If you are unsure what a class/method exactly does, please refer to the code.

Submission Procedure

Submit the files in the src/main directory, excluding Main.java, as a zip file. You *must* make a zip file for submission using the Gradle build tool (refer to Compiling section).

For this assignment, you should submit only the following three files:

- Graph.java
- your_student_id_number.txt

You must rename 2023xxxxxx.txt to your actual student ID number. Inside that text file, you must include the following text. 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 a 0 on the assignment. It should be named *exactly* your student id, with no other text. For example, *2023123456.txt* is correct while something like *2023123456_pa6.txt* will receive 0.

Compiling

You can test your Java code using the following command:

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

You can also make a zip file for submission using the following command. The zip file named with your student id (the name of the .txt file) will lie in the "build" directory:

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

We provide an empty Main class for testing using standard input/output:

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

Since this file (src/main/Main.java) is not for submission, you can use any package in the file.

On Windows, use `gradlew.bat` instead of `./gradlew`.

Testing

We have provided a small test suite (src/test) for you to check your code. You can test your code by compiling and running the tester program.

Note that the test suite we will use to grade your code will be 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.