Programming Assignment 2

Programming assignments are to be done individually. You may discuss the problem and general concepts with other students, but there should be no sharing of code. **You may not submit code other than that which you write yourself or is provided with the assignment. This restriction specifically prohibits downloading code from the Internet.** If any code you submit is in violation of this policy, you will receive no credit for the entire assignment.

This lab is due **Friday, November 9th at 11:59PM.** If you are unable to complete the lab by this time you may submit the lab late until Monday, November 12th at 11:59PM for a 20 point penalty.

The goals of this lab are:

- Familiarize you with programming in Java
- Show an application of Dijkstra's Algorithm
- Implement a heap

Problem Description

In this project, you will implement a minimum heap by the `minDistance` variable in Node. You will also implement Dijkstra's shortest path algorithm on a directed graph using the same heap. It should run in $O((E+V)(\log(V))$. We will provide you with the following classes:

- `Node`
  Keeps track of `int minDistance` (the shortest distance to root), `int nodeName`, and ArrayList of `neighbors` with respective `weights`.
- `Heap`
  Needs to have all the properties of a heap. This class needs to function independently; the methods we ask you to implement will not all be needed for Dijkstra's algorithm, but we will test them separately. In terms of Dijkstra's algorithm, Heap is used to find the shortest path from the root to any other Node. Before Dijkstra runs, the Heap will consist of every node with the root node as the minimum. After termination, the Heap should be empty, and every Node should have the correct `minDistance`.
- `Graph`
  Holds a Heap and an ArrayList of all Nodes (vertices) in the graph.
- `Driver`
  Reads file input and instantiates each Node in the graph. It also populates ArrayLists `neighbors` and `weights` for each Node. You can modify `testrun()` for testing purposes, but we will use our own `Driver` to test your code.

You need to implement `Heap` and `Graph`. `Driver.testrun()` can be modified for your own testing purposes, but we will run our own `Driver` and test cases.

Part 1: Write a report [20 points]
Write a short report that includes the following information:

(a) Give the pseudocode for your implementation of Dijkstra's algorithm and Heap. The pseudocode for Dijkstra's algorithm should involve maintaining the Heap.

(b) Justify why your pseudocode runs in $O((E+V)(\log(V))$.

(c) You implemented Dijkstra's algorithm from a graph represented using an adjacency list. How would your algorithm change if the graph was represented by an adjacency matrix? Give and explain the runtime of this approach. For what reasons might it be better to use an adjacency list?. Use (5-9 sentences).

Part 2: Implement a Heap [30 points]
Complete `Heap` by implementing the following methods:

- `void buildHeap(ArrayList<Node>)`
  Given an ArrayList of Nodes, build a minimum heap in $O(n)$ time based on each Node's `minDistance`.
- `void insertNode(Node)`
  Insert a Node in $O(\log(n))$ time.
- `int findMin()`
  Find minimum in $O(1)$ time.
- `int extractMin()`
  Extract minimum in $O(\log(n))$ time.

Break any ties by `int nodeName`. For example, if Node 0 and Node 1 both have `minDistance = 4`, choose Node 0 to be "smaller".

Part 3: Implement Dijkstra's Algorithm [50 points]
Complete `Graph` by implementing the following methods:

- `int findShortestPathLength(Node root, Node x)`
  Returns distance of shortest path from root to x. If such a path does not exist, return `-1`.
- `ArrayList<Node> findAShortestPath(Node root, Node x)`
  Returns a shortest path from root to x. Any path works. If no paths exist, return `null`.
- `ArrayList<Node> findEveryShortestPathLength(Node root)`
  Returns an ArrayList of Nodes, where `minDistance` of each node is the shortest path from it to root. This ArrayList should contain every node in the graph. Note that `root.getMinDistance() = 0`.

Of the files we have provided, `Graph`, `Heap`, and `Node` are what will be used in grading. Feel free to add any helper methods or fields in `Graph` and `Heap` but be careful not to remove any to ensure that your classes remain compatible with your grading. Also, feel free to add any additional Java files (of your own authorship) as you see fit.

Input File Format

The first line of file is the number of total nodes in the graph, and the number of total edges in the graph. The first number on each even line is the name of a node, and every number that follows it is a neighbor in the graph. The first number on each odd line is the name of a node, and every number that follows it is the weight of the edge between it and the neighbor in the line above.

For example, if the input file is as follows:

4 5
0 1
0 9
1 0 2
1 9 8
2 1
2 8
3 1
3 10

Then there are 4 vertices, 5 edges.
Node 0 has neighbor 1 with weight 9.
Node 1 has neighbor 0 with weight 9 and neighbor 2 with weight 8.
Node 2 has neighbor 1 with weight 8.
Node 3 has neighbor 1 with weight 10.
Remember that this is a directed graph.

Instructions

- Download and import the code into your favorite development environment. We will be grading in Java 1.8 on the ECE LRC machines. Therefore, we recommend you use Java 1.8 and NOT other versions of Java, as we can not guarantee that other versions of Java will be compatible with our grading scripts. **It is YOUR responsibility to ensure that your solution compiles with Java 1.8 on the ECE LRC machines.** If you have doubts, email a TA or post your question on Piazza.

- Make sure you don't add the files into a package (keep them in the default package). Some IDEs will make a new package for you automatically. If your IDE does this, make sure that you remove the package statements from your source files.

- If you do not know how to download Java or are having trouble choosing and running an IDE, email a TA, post your question on Piazza or visit the TAs during Office Hours.

- There are several `.java` files, but you only need to make modifications to `Graph` and `Heap`. You can modify `Driver` if you'd like, but do not modify `Node` However, you may add additional

source files in your solution if you so desire. There is a lot of starter code; carefully study the code provided for you, and ensure that you understand it before starting to code your solution. The set of provided files should compile and run successfully before you modify them.

- `Driver.java` is the main driver program. It currently prints out your Graph and Heap. Modify `testrun()` to suit your liking. A main portion of the lab is debugging, be sure to leave time for that.
- Make sure your program compiles on the LRC machines before you submit it.
- We will be checking programming style. A penalty of up to 10 points will be given for poor programming practices (e.g. do not name your variables foo1, foo2, int1, and int2).
- Before you submit, be sure to turn your report into a PDF and name your PDF file `eid_lastname_firstname.pdf`.

What To Submit

You should submit a single zip file titled `eid_lastname_firstname.zip` that contains all of your java files and pdf report `eid_lastname_firstname.pdf`. Do not put these files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your solution must be submitted via Canvas BY 11:59 pm on the due date.