Programming Assignment 2

Programming assignments are to be done individually. Do not make your code publicly available (such as a Github repo) as this enables others to cheat and you will be held responsible. 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 **Tuesday, November 2nd at 11:59 PM.** If you are unable to complete the lab by this time, you may submit the lab late until Friday, November 5th at 11:59 PM for a 20 point penalty.

The goals of this lab are:

- Recognize algorithms you have learned in class in new contexts.
- Ensure that you understand certain aspects of graphs and greedy algorithms.

Problem Description

State A has multiple cities connected by roads. Suppose you are in charge of planning for a new bus route to connect two cities s and v, based on existing roads. You want to make the transportation as fast as possible, so you need to find the shortest route from s to v. You have a map that labels the length of the road between any two cities. How would you plan for the bus route?

You received a new task. The telecom company plans to install fiber-optic Internet for the whole state. For the convenience of building and maintaining, these fiber-optic cables should be installed by existing roads. Your task is to decide the most efficient set of roads to install fiber-optic cables, such that the fiber-optic internet is available at every city and the total length of fiber-optic cables is minimized.

In this project, you will implement a minimum heap by the `minDist` variable in the City class. You will also implement an algorithm to find the minimum distance route between two given cities. Finally, you will implement an algorithm to find which roads the fiber-optic cables should be installed by. You can assume that following the road between two Cities takes equal time regardless of direction (i.e. a **connected undirected weighted graph**). We will provide you with the following classes:

- `City`
  Keeps track of `int minDist` (the minimum total distance needed to get to this City from the starting City) and `int name` (the numerical id of the city, as written on the map). Also contains `ArrayList<City> neighbors` (adjacent Cities) and `ArrayList<Integer> weights` (distances to adjacent Cities).
- `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 your algorithm, but we will test them separately. Only valid parameters will be passed into heap functions so you do not need to handle invalid inputs.

- `Program2`
  The class that you will be implementing your algorithms in.

- `Driver`
  Reads file input and populates `ArrayList cities`. You can modify `testRun()` for testing purposes, but we will use our own `Driver` to test your code.

You need to implement the `Heap` and `Program2` classes. `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 an algorithm to find the length of the minimum distance route between two given Cities. Give the runtime of your algorithm and justify.

(b) Give the pseudocode for your implementation of an algorithm to find the minimum total length of fiber-optic cables that must be used to connect all Cities. Give the runtime of your algorithm and justify.

For both (a) and (b), make sure to give enough detail in your pseudocode so that the runtime can be accurately determined. (For example, when following an edge to update the City's `minDist`, how do you "find" the corresponding `City` in the heap?) When justifying the runtime, go into sufficient depth to show how your actual implementation maintains the desired runtime.

Part 2: Implement a Heap [20 points]
Complete `Heap` by implementing the following methods:

- `void buildHeap(ArrayList<City> cities)`
  Given an ArrayList of Citys, build a minimum heap based on each City's `minDist` in either $O(n\log(n))$ or $O(n)$ time. In testing, we won't be calling buildHeap more than once per test case.

- `void insertNode(City in)`
  Insert `City in` in $O(\log(n))$ time. Your heap should be able to grow beyond the original size.

- `City findMin()`
  Return minimum in $O(1)$ time.

- `City extractMin()`
  Return minimum and delete from heap in $O(\log(n))$ time.

- `void delete(int index)`
  Delete the City at `int index` in $O(\log(n))$ time.

- `void changeKey(City s, int newDist)`
  Change `minDist` of `City s` to `newDist` and update the heap in $O(\log(n))$ time.

Break any ties by `int cityName`**.**
For example if City 0 and City 1 both have `minDist = 4`**, choose City 0 to be "smaller".**

Part 3: Finding the Minimum Distance Route [30 points]
Implement an algorithm to find the minimum distance between two given Cities. You will be heavily penalized for a non-polynomial time (in terms of the number of Cities) algorithm. This means you can't brute force the solution. More specifically, your target runtime should be $O(m\log(n))$ where m is number of roads and n is number of cities. To achieve this you will need to use the heap. The specific inputs provided and outputs expected are provided below.

Input(s):

- A starting `City start`
- A destination `City dest`

Output:

- A `distance` value where `distance` represents the minimum distance required to get from City `start` to City `dest`. It is NOT necessary to explicitly return the actual shortest path tree as long as your algorithm returns the correct distance.

Method Signature:

- `int findMinimumRouteDistance(City start, City dest);`

Part 4: Finding the Minimum fiber-optic cable Length [30 points]
Implement an algorithm to find the minimum fiber-optic cable length needed to connect every City, given a list of Cities and road distances between them. You will be heavily penalized for a non-polynomial time (in terms of the total number of Cities) algorithm. The specific inputs provided and outputs expected are provided below.

Input(s): none
Output:

- A `minLen` value where `minLen` represents the sum of the lengths of the roads under which the fiber-optic cables will be installed. It is NOT necessary to explicitly return the MST as long as your algorithm returns the correct `minLen`.

Method Signature:

- `int findMinimumLength();`

Of the files we have provided, `City`, `Heap`, and `Program2` are what will be used in grading. Feel free to add any methods or fields but be careful not to remove any to ensure that your classes remain compatible with our grading. Also, feel free to add any additional Java files (of your own authorship) as you see fit. **The files you turn in MUST compile and run with the provided, unmodified `Driver` to ensure it runs with the `Driver` used in grading.**

Input File Format
The first line of file is the total number of Cities and the total number of roads between the Cities. The

first number on each even line is the id of a city (natural number), and every number that follows it is a city that is connected to it by a road. The first number on each odd line is the id of a city, and every number that follows it is the distance of the road between it and the connected city in the line above.

For example, if the input file is as follows:
3 2
0 1
0 9
1 0 2
1 9 8
2 1
2 8
Then there are 3 cities, 2 roads.
City 0 has a road to City 1 with length 9.
City 1 has a road to City 0 with length 9 and City 2 with length 8.
City 2 has a road to City 1 with length 8.

We will parse the input files for you, so you don't need to worry too much about the input file format except when trying to make your own test cases.
We will use different inputs for testing, but test cases will have valid inputs so you do not need to handle invalid inputs throughout your whole program.

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.
- **Do not add any package statements to your code.** 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 before turning in the assignment.
- 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 `Heap` and `Program2`. 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).

⚠**NOTE:** To avoid receiving a 0 for the coded portion of this assignment, you MUST ensure that your code correctly compiles with the original, unmodified starter files on Java 1.8. Do not modify the signatures or remove existing methods of `Program2.java`, `Heap.java`, or `City.java`. Do not add package statements. Do not add extra imports. You must zip your code using the exact format described below with no spaces. We recommend testing compilation of your code using the ECE LRC Linux machines (using "javac *.java" and "java Driver [args] [inputfile.txt]") after redownloading the starter files from Canvas. We will not be allowing regrades on this assignment, so please be careful and double check that your final submission is correct.

What To Submit (please read carefully)

You should submit to Canvas a single ZIP file titled `pa2_eid_lastname_firstname.zip` that contains `Program2.java`, `Heap.java`, `City.java`, and any extra `.java` files you added. Do not submit `Driver.java`. Do not put your `.java` files in a folder before you zip them (i.e. the files should be in the root of the ZIP archive). Your ZIP file name MUST have the exact format: `pa2_eid_lastname_firstname.zip`. Be certain that there are no spaces in your zip file name. Failure to follow these instructions will result in a penalty of up to 10 points.

Your PDF report should be typed or legibly scanned and submitted to Gradescope. Both your zipped code and PDF report must be submitted by **Tuesday, November 2th at 11:59 PM.** If you are unable to complete the lab by this time, you may submit the lab late until Friday, November 5th at 11:59 PM for a 20 point penalty.