COMS 311: Homework 6
Due: Dec 1, 11:59pm
Total Points: 200

Late submission policy. Submission after Dec 1, 11:59PM will not be graded without explicit permission from the instructors.

Points. Total points for this homework assignment is 200pts. There will be extra credit test cases worth 40pts.

Learning outcomes.

Design, implement and evaluate algorithms following specifications.

0 Preamble

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistants for any questions/clarifications regarding the assignment.

Your programs must be in Java (as you have done in Homework 4; do not use packages from javafx, com.sun.*, or packages specifically available with some IDE). Please read submission requirements carefully before submitting.

Excerpt from the syllabus: For any assignment that involve submitting written code

(a) Students are expected to design tests and validate solution. Some example outputs for specific inputs may be provided as part of the assignment specification. Submissions that successfully produce the outputs for these specific inputs do not automatically get any points (or partial credits).

(b) Grading programming assignment will involve assessment based on test suit (T) designed by the teaching staff members. Failing to pass any of the tests in T will result in 0 points.

(c) Solutions must follow assignment specification; for instance, the assignment description may include specification of output in a specific format or the implementation of a specific method (with a specific signature). Submissions that do not conform to such specifications will be considered incorrect.

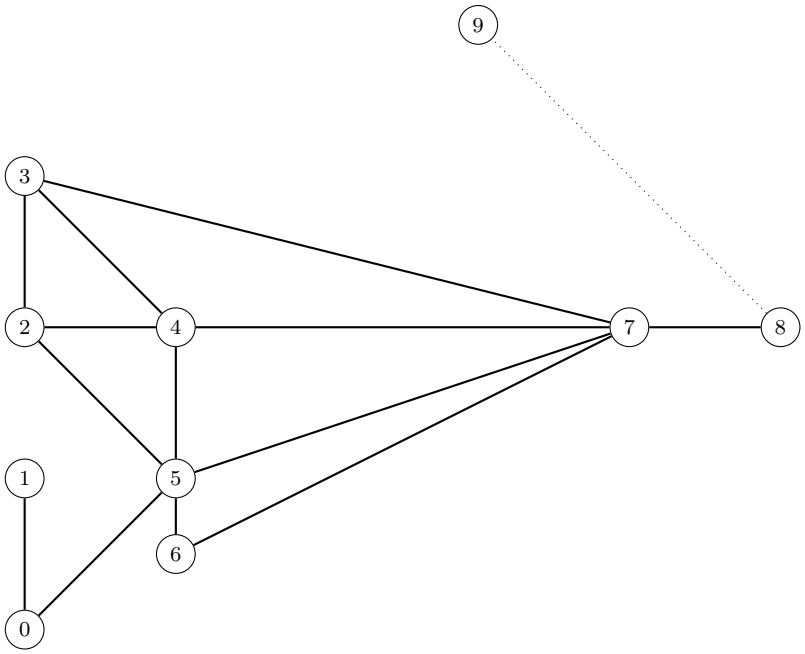
1 Problem Description

You are a software engineer developing a city- and intercity-planning software to used by urban planning agencies (e.g., road-works department, emergency management departments). As a result, the software provides answers to different type of queries such as shortest distances between intersections in the city, which allows for planning of bus-routes; (b) minimum overall roadways to be maintained to connect one intersection with others.

```
10 13
0    0 0
 1 0 10
 4   10 20
5   10 10
 6 5 5
7   80 20
 8 85 20
 2   0 20
  3   0 30
9 50 50

1 0
0   5
5 6
4 5
2 5
2 4
4 3
7 4
 7 3
5 7
6 7
  8 7
2 3
9 8
```

(a)



(b)

Figure 1: (a) Input file. (b) Graph Representation of Input roadways and intersections (not scaled). The dotted edge is not considered as the Input file mandates considering first 13 edges.

It has been decided that the intersections in the city should be identified using an integer identifier and their locations are described using coordinates. Furthermore, if there is a roadway between two intersections, the distance or length of the roadway can be approximated to be the Euclidean distance between the intersections.

2 Input File Format

The input file containing the information about intersections and roadways is presented in Figure 1 The first line states the number of intersections n (in the example, $n = 10$) and the number of roadways m (in the example, $m = 13$) that connect different intersections. The n lines immediately following the first line (there is no empty line after the first line) contains intersection identifiers followed by the coordinates of the location of the intersection. The value of the intersection identifiers is between 0 and $n - 1$.

Following the intersection information, there is a blank line, followed by $\geq m$ roadway information. In each line, there are two numbers, which are the intersections connected by a roadway. All roadways are bidirectional. For instance, in the above example, one can go from intersection 0 to 1 and from 1 to 0 using the same roadway connecting the two intersections 0 and 1. Note that, there may be $> m$ roadway information. This is because the planning agencies may include future roadway information in their files. For any analysis related to the current state of the city roadways, only the first m roadways need to be considered. None of the roadways will have intersection ids outside the range $[0, n - 1]$.

You should not assume any specific number of blank spaces between numbers when reading this

file. You can assume that all coordinate values will be integers between 0 and 10,000.

3 Type of Queries Answered by the Software

As noted before, the software provides answers to several queries. They are of following types:

1. Number of shortest paths between two given intersections.
2. A shortest path between two given intersections.
3. Number of shortest paths from one intersection to another intersection via some other intermediate intersections.
4. A shortest path from one intersection to another intersection via some other intermediate intersections.
5. The shortest path distances from an intersection to all other intersections.
6. A set of roadways that connects a given intersection to all intersections that are reachable from the given intersection. There can be may such sets—the software outputs the one where the sum of the distances of the roadways is minimal.

In the following, we will present the implementation specification that you need to follow to realize the answers to the above queries.

4 Implementation Specification

You may have realized the intersections can be encoded as vertices, roadways between intersections can be encoded as undirected edges, and the weight of the edges are Euclidean distance between the vertices connected by the edges. Furthermore, you may be implementing Single-source-shortest path algorithms and spanning tree algorithms to answer the queries presented in the previous section.

4.1 Priority Queue as Binary Min-Heap

You will implement a binary min-heap. The elements in the heap have two attributes: a key of type integer and a value of type double. The ordering of the elements as per the min-heap property is maintained using the value of the elements. That is, the element with the smallest value will be present in the root of the heap. If there are two elements with the same value, then the element with smaller key will be considered to be "smaller".

Write a class `MinHeap` that conforms to the following requirements:

1. It must have a default constructor, which initiates a binary min-heap.
2. It must have a method `add`. It has two parameters. The first parameter (int-type) describes the key of the element being stored in the heap and the second parameter (double-type) is the value associated to the key.

   The return type of this method is void.

3. It must have a method `getHeap`. It has no formal parameter. The return type is `ArrayList<Integer>`. The method returns the keys of the elements stored in the heap in the form of an array-list. For instance, the key of the root of the heap must be at the 0-th index of the array-list, the 1-st and 2-nd index of the array-list contains the children of the root; and so on.

All the methods specified must have public modifier.

4.2 Utility Class: PathFinder

All the major methods necessary to answer the queries in Section 3 will be present in class `PathFinder`. The following sections outline these methods.

4.2.1 Path Computation using Dijkstra-based Heuristics

Dijkstra Algorithm for Shortest Paths from Source. Recall that Dijkstra algorithm associates (and computes) with each vertex a property d. In Dijkstra algorithm, for the vertex u with the smallest d-value, the d-value for all u's neighbors (say, v) is updated as follows

$$d(v) = min\{d(v),\ d(u) + wt(u, v)\}$$

This type of update is often referred to as *relaxation*. Dijkstra method, as we have learned, can be used to find the shortest paths and shortest path distances from a given vertex to all other vertices in the graph.

Adaptive Relaxation for Paths between Source and Target. Dijkstra algorithm can be also used to find the shortest path and shortest path distance from a source to a target. You are required to develop a variant of Dijkstra algorithm to answer queries related to finding a path between source and destination, and computing the distance for such a path. This variant uses adaptive relaxation, where the relaxation depends on two input parameters: A and B, and the target t. It is defined as follows:

$$d(v) = min\{d(v),\ A \times (d(u) + wt(u, v)) + B \times (distance(v, t) - distance(u, t)\}$$

Recall that, $wt(u, v)$ is the Euclidean distance between vertices u and v that are connected by an edge. The function $distance(u, t)$ and $distance(v, t)$ are Euclidean distances of target t from u and v, respectively; it does not imply the existence of a edge connecting u and t, or v and t.

Notice that, when $A = 1$ and $B = 0$, adaptive relaxation reduces to the relaxation used in Dijkstra algorithm.

4.2.2 Reachability Tree from a Source

For a given source vertex, you need to compute a tree such that the tree contains paths from the source to the reachable vertices. There can be many trees that satisfy this property. You are required to compute the tree such that the sum of the weight of the edges in the tree is minimal.

4.2.3 PathFinder Class Implementation

Write a class `PathFinder` that includes the following methods:

1. It must have a default constructor.

2. It must have a method `readInput`. It has a formal parameter of String-type. The parameter captures the name of the file, where the information regarding the intersections and roadways are present (see Section 2). The method does not return anything.

3. It must have a method `shortestPathDistances` with a int-type parameter. The method is invoked with an intersection id (say, i) as argument and returns an array of doubles containing shortest path distances to all other intersections. In the array, the value of the j-th element is the shortest path distance between i and j. If there is no path to some intersection, then for that intersection, the shortest path distance is set to -1.

4. It must have a method: `noOfShortestPaths` with two int-type parameters. The first parameter corresponds to the id of the source intersection and the second parameter corresponds to the id of a destination intersection. The method returns the number of shortest paths between the source and destination. The return type is int. If there is no path, then the method returns 0.

5. It must have a method: `fromSrcToDest` with four int-type parameters. The first parameter corresponds to the id of the source intersection and the second parameter corresponds to the id of a destination intersection. The third and fourth parameters are of int-type and correspond to A and B respectively for the adaptive relaxation. The method computes a path from source to destination using adaptive relaxation. The return type is `ArrayList<Integer>`. The i-th element in the ArrayList is the id of the intersection that is i edges away from the source in the path. Note that the 0-th element in the ArrayList is the id of the source. If there is no path, the method returns null.

6. It must have a method: `fromSrcToDestVia` with five parameters. The first parameter corresponds to the id of the source intersection and the second parameter corresponds to the id of a destination intersection. The third parameter is of type `ArrayList<Integer>`. It corresponds to a list of ids of intersection that needs to be visited on the way from source to the destination intersection. This parameter will not be empty/null. It will not contain the ids of the source or the destination. The fourth and fifth parameters are of int-type and correspond to A and B respectively for the adaptive relaxation. The i-th element in the ArrayList is the id of the intersection that is i edges away from the source in the path. Note that the 0-th element in the ArrayList is the id of the source. If there is no path, the method returns null.

   You can think of constructing the path by first considerng the construction of path from source to the first intersection (say, i) to be visited on the way, then constructing the path from i to the second intersection to be visited on the way, and so on.

7. It must have a method: `minCostReachabilityFromSrc` with one parameter of int-type. The parameter corresponds to the id of an intersection – the source intersection. The method computes the minimum cost reachability tree from that intersection (see Section 4.2.2). The output is of type `int []`. The i-th element in the array is the *parent* of the intersection with

id i. If the id of the source is i, then the i-th element in the array is also i. If i is the id of an intersection not reachable from the source, then the i-th element in the array is equal to -1.

8. It must have a method: `minCostOfReachabilityFromSrc` with one parameter of int-type. The parameter corresponds to the id of an intersection – the source intersection. The method computes the cost of the minimum cost reachability tree from that intersection (see Section 4.2.2). The return type is `double`.

9. It must have a method: `isFullReachableFromSrc` with one parameter of int-type. The parameter corresponds to the id of an intersection – the source intersection. The method determines whether all intersections are reachable from the source. The return type is `boolean`.

Whenever any of the above methods are invoked, the relevant arguments will contain valid intersection ids. All the methods specified must have public modifier.

5 Listing

A sample listing of output for each of the methods for a given input is given as a separate file.

6 Submission File Requirements

1. The name of your submission file must be PathFinder.java. You will not submit any other file or file(s) of any other format (zipped, unzipped or otherwise). Please comment your debug-print statements before submitting.

2. The PathFinder.java shall not have any `package` directive.

3. You can write as many helpers (classes, methods) you want. All helpers and all classes specified in this assignment must be in one file - PathFinder.java. For instance,

```
// import directives

public class PathFinder {
   ...
}

class MinHeap {
   ...
}

class ArbitraryHelper {
  ...
}
```

7 Postscript

1. Euclidean distance. Use the following.

```
public double distance(int x1, int y1, int x2, int y2) {
   return Math.sqrt((x1 - x2)* (x1 - x2) + (y1 - y2) * (y1 - y2));
}
```

2. For testing: There will be no formatting and semantic errors in the input file.

3. For testing: The methods will be invoked with correct parameter values in the context of the input test files.

4. For testing: The expected answers will be such that they will not lead to data-type overflow. For instance, number of shortest paths for a destination will not be over the maximum value the int-type can handle.

5. You must follow the given specifications. Method names, classnames, return types, input types. Any discrepancy may result in lower than expected grade even if "everything works".

6. There are several data structure/organization that are left for you to decide. Do not ask questions related to such data structure/organization. Part of the exercise to understand and assess a good way to organize data that will allow effective application of methods/algorithms.

7. Start reading and sketching the strategy for implementation as early as possible. That does not mean starting to "code" without putting much thought on what to code and how to code it. This will also help in resolving all doubts about the assignment before it is too late. Early detection of possible pitfalls and difficulties in the implementation will help in reducing the finishTime-startTime for this assignment.

8. Both correctness and efficiency are important for any algorithm assignment. Writing a highly efficient incorrect solution *will* result in low grade. Writing a highly inefficient correct solution *may* result in low grade. In most cases, the brute force algorithm is unlikely to be the most efficient. Use your knowledge from lectures, notes, book-chapters to design and implement algorithms that are correct and efficient.

9. Test your code extensively (starting with individual methods). Your submission will be assessed using test cases that are different from the ones provided as part of this assignment specification. Your grade will primarily depend on the number of these test cases for which your submission produces the correct result.