COMS 311: Homework 5
Due: Nov 18, 11:59pm
Total Points: 100

Late submission policy. Submission after Nov 18, 11:59pm and before Nov 19, 11:59pm will have a penalty of 20%. That is, if your score is x points, then your final score for this homework after the penalty will be $0.8 \times x$. Submission after Nov 19 will not be graded without explicit permission from the instructors.

Points. Total points for this homework assignment is 100pts. There will be extra credit test cases worth 20pts.

Learning outcomes.

Design, implement and evaluate algorithms following specifications.

Shortest path algorithms for weighted graphs

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 submission must be in Java. Please read submission requirements carefully before submitting.

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.

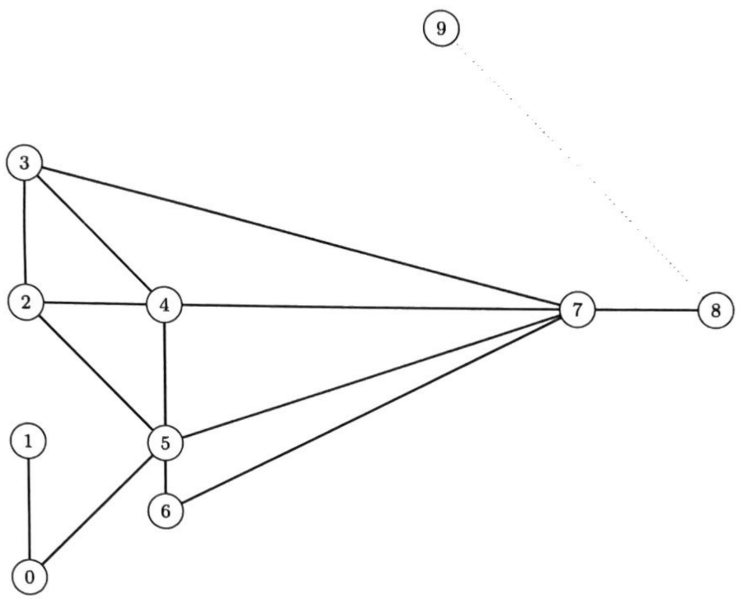
It has been decided that the intersections in the city should be identified using an integer identifier and their locations are described using coordinates. Furthermore, if there is a roadway between two intersections, the distance or length of the roadway can be approximated to be the Euclidean distance 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 $n - 1$.

```
10 13
0    0 0
 1 0 10
  4   10 20
5   10 10
 6 10 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.

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.

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

[(0,0),(1,0,10), , , ,0, , , , ,]

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 algorithms similar to Dijkstra's shortest path algorithm.

4.1 Priority Queue as Min-Heap

You will implement a k-ary min-heap. In the class, we have covered min-heap implementation using an array-based container where $k = 2$. In this assignment, k will be an input. In the array-based implementation, the root of k-ary min-heap is located at index 0 and its children are located at indices 1 to k; the element at index 1 has its children located at indices $k + 1$ to $2k$; the element at index 2 has its children located at indices $2k + 1$ to $3k$, and so on.

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 (i.e., default k value is 2).

2. It must have a constructor, which takes one int-type parameter. The parameter indicates the value of k for k-ary min-heap.

3. 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.

4. 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 $k + 1$-th and $2k$-th indices of the array-list contains the children of the root; and so on.

All the methods specified must have public modifier.

4.1.1 Usage listing

This is an example listing. It does not indicate any specific ordering in which the methodswill be invoked. Your code will be tested based on any allowable call-specifications.

```
MinHeap mH = new MinHeap();
mH.add(1, 5);
mH.add(11, 6);
mH.add(2, 6);
mH.add(3, 4);
mH.add(10, 3);
mH.add(9, 6);

ArrayList<Integer> elements = mH.getHeap();
System.out.println("MinHeap with default degree");

for (int i=0; i<elements.size(); i++)
    System.out.printf("%5s", elements.get(i));
System.out.println();

/* produce the output
MinHeap with default degree
10 3 2 11 1 9
*/

mH = new MinHeap(4);
mH.add(1, 5);
mH.add(11, 6);
mH.add(2, 6);
mH.add(3, 4);
mH.add(10, 3);
mH.add(9, 6);

elements = mH.getHeap();
System.out.println("MinHeap with non-default degree");

for (int i=0; i<elements.size(); i++)
     System.out.printf("%5s", elements.get(i));
System.out.println();
/* produce the output:
MinHeap with non-default degree
10 9 2 1 3 11
*/
```

4.2 Utility Class: PathFinder

All the major methods necessary to answer the queries of the urban planning agency (such as the one presented in Section 3) will be present in class `PathFinder`. The following sections outline these methods.

4.2.1 Shortest path computation from a source to a destination

Shortest path using Euclidean distance. Recall that Dijkstra's algorithm associates (and computes) with each vertex a property d. In Dijkstra's 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's 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.

In this assignment, you will implement a variant of Dijkstra's algorithm to compute the shortest path from a given source s to a given destination t where *d*-values are computed as follows:

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

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 road (i.e., an edge in the network) connecting u and t, or v and t.

4.2.2 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: `distToDest` with three int-type parameters. It returns the shortest path distance from a source intersection to a destination intersection.

   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 denotes the k value of k-ary min-heap to be used for computing the shortest path distance.

   The return type of the method is double. If the destination is unreachable from the source, then the method returns -1.
4. It must have a method: `noOfShortestPaths` with three 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 parameter denotes the k value of k-ary min-heap to be used for computing the number of shortest paths using Dijkstra's algorithm.

   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 three parameters. This method returns the shortest path from a source intersection to a destination intersection.

   The first parameter (type int) corresponds to the id of the source intersection and the second parameter (type int) corresponds to the id of a destination intersection. The third parameter denotes the k value of k-ary min-heap that you will be using for implementing the algorithm for computing shortest path from source to destination. You are required to implement the variant of Dijkstra's algorithm based on the desciption in Section 4.2.1.

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.

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

4.2.3 Usage listing

```
PathFinder pf = new PathFinder();

// read input
pf.readInput("sample1.txt");

//
System.out.println("Shortest path distance: " + pf.distToDest(0, 3, 2));

/* produce the output:
Shortest path distance: 38.2842712474619
*/

System.out.println("M-Path Distance: " + pf.distToDest(0, 9, 2));
/* produce the output:
Distance of shortest paths: -1.0
*/

// Number of shortest paths
System.out.println("Number of shortest paths: " + pf.noOfShortestPaths(0, 3, 2));
/* produce the output:
Number of shortest paths: 2
*/

ArrayList<Integer> path = pf.fromSrcToDest(0, 3, 2);
if (path == null)
     System.out.println("No path to destination");
else {
     for (int i=0; i<path.size(); i++)
          System.out.printf("%5s", path.get(i));
     System.out.println();
}
/* produce the output: If there are multiple shortest paths
                       then any one will be a correct output.
0 5 2 3
*/
```

5 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).
2. The PathFinder.java shall not have any `package` directive.
3. Please use the method signatures specified. If the return type of a method is void, please do not make it boolean. If the type of an argument is int, please do not make it Integer. If the name of a method is `myFunction`, please not make it `yourFunction`, `thisFunction`, `myFun`, etc.
4. Please comment your debug-print statements before submitting.
5. Please do not submit code that does not compile.
6. Please do not use packages from javafx, com.sun.*, and/or packages specifically available with some IDE.
7. 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 {
   ...
}
```

6 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 is 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.