Homework 7
COMS 311 Points: 250
Due: Nov 13, 11:59PM

Submission by Nov 12, 11:59PM: 20% credit.
Submission after Nov 13, 11:59PM and before Nov 14, 11:59PM: 20% penalty.

Learning outcomes.

Design, implement and evaluate algorithm following specifications.

0 Preamble

Description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. The assignment relies on several concepts, algorithms, data structures covered in class, discussed in lectures, developed in the textbook.

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).

Excerpt from syllabus.

For any assignment that involve submitting written code

1. 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).

2. 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.

3. Solutions must follow~~ing~~ 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 conducting some planning analysis on behalf of a city-planning commission. As part of the analysis, it is necessary to identify the (a) shortest distance between intersections of the city, which allows for understanding the minimum transportation units (bus-routes, metro, city-rails etc.) necessary to connect two intersections; and (b) minimum total distance from one intersection I to all intersections that are connected by available roadways, which allows for understanding the minimum amount of necessary road-maintenance to ensure connectivity between intersections.

Each intersection is described by an integer identifier and the coordinates of the location of the intersection. Due to prior careful planning of the city, all roadways between intersections follow the straight-line path between the intersections. In other words, the length of a road between two intersections is equal to the Euclidean distance between the intersections. All intersections may not be connected by roadways.

To facilitate the required analysis, you will write a software that reads from a file the description of the roadways in the city and computes answers to the queries related to planning analysis. To make your software future-proof, do not just hard-code the max roadway or intersections presented in this specification. Consider the scenario, where you may want to re-use the software for analysis of road-network containing 10k-90k intersections with 20k-120k roadways.

1.1 Shortest Path

A path between pair of intersections is described by sequence of zero/more roads. The distance covered in a path is the sum of the length of the roads present in the path; this is called the length of the path or the path-length. If one intersection is not reachable from another intersection, then the length of the path from the latter to the former is considered to be -1 (indicating unreachability). The roadway between a pair of intersections (that can reach each other) with the smallest path-length is called the M-path. The shortest roadway distance is the path-length of a M-path.

There may be multiple M-paths between pairs of reachable intersections. We refer to special M-path as low-M-path. It is constructed as follows (assume that M-path being considered is from intersection s to intersection t).

1. initialize an empty path P then
2. start at the destination t.
3. $I = t$
4. add I to P
5. Among all the intersections that have a roadway (direct link) to I and is part of some M-path from s to t, identify the intersection J whose id is the smallest.
6. $I = J$
7. Repeat from Step 4 until s is added to P; and then return P

You may have realized that the intersections can be modeled as vertices in a graph, the roadways as edges, and the weights on the edges can be computed from the Euclidean distance between vertices connected by the edges. In what follows, we will use the terms intersections and vertices interchangeably, and the terms roadway and edges interchangeably. You shall use Dijkstra's shortest path algorithm to solve problems related to M-paths.

1.1.1 Min-Heap as Priority Queue

Recall that Dijkstra algorithm depends on the **k-ary min-heap** representation of priority queue (in class you have reviewed min-heap where $k = 2$, i.e., binary min-heap). The min-heap keeps track of the vertices (the *key*) and an associated distance measure d (the *value*) in such a way that the vertex with the lowest d-value has the highest priority (i.e., is at the root of the tree-representation of the min-heap). If there are two keys with the same d-value, then consider the pair with smaller key to have higher priority.

In an array implementation for a k-ary min-heap, the root is located at index 0, its children will be located at indices 1 to k; the element at index 1 will have its children located at indices $k+1$ to $2k$; element at index 2 will have its children located at indices $2k+1$ to $3k$, and so on. As part of this assignment, you will implement a k-ary min-heap, where k will be an input for the min-heap specification.

1.1.2 Dijkstra Algorithm

We have noted that the Dijkstra algorithm associates with each vertex a property d, which captures the distance measure of the vertex from the source vertex s. Further recall, in Dijkstra algorithm, the vertex u with the smallest such measure is extracted from a priority queue[1], and the d-value for all u's neighbors (say, v) get updated as follows:

$$d(v) = \min\{d(v), d(u) + wt(u, v)\} \tag{1}$$

In the above, $wt(u, v)$ denotes the weight of the edge from u to v. We will refer to this algorithm as **Dijkstra-v1 algorithm**. Dijkstra-v1 algorithm will be used to find the shortest roadway distances from a given source to all vertices.

Dijkstra-v1 algorithm will be also used to compute specific M-paths, i.e., shortest distances from a given source s to a given destination t. Additionally, a new variant, referred to as **Dijkstra-v2**, will be used to address such problems. The algorithm for Dijkstra-v2 is identical to Dijkstra-v1 except the rule for updating the property d. As in Dijkstra-v1, the vertex u with the lowest d-value is extracted from the a priority queue[1], and the d-value of all u's neighbors (say, v) get updated as follows:

$$d(v) = \min\{d(v), d(u) + wt(u, v) + dist(v, t) - dist(u, t)\} \tag{2}$$

In the above, for any vertex x and y, $dist(x, y)$ denotes the Euclidean distance between x and y (note that, the $dist(x, y)$ does not require an edge between x and y; and the d-value in the Dijkstra-v2 does not capture the exact distance measure from the source to a vertex).

You are required to implement both Dijkstra-v1 and Dijkstra-v2 and k-ary min-heap (default value for k would be 2) in the context of this assignment.

1.2 Reachability Tree

Given an intersection I_S, the reachability tree from that intersection is the tree formed of the roadways such that all intersections reachable from I_S are present in the tree and the root of the tree is I_S. Recall that a tree does not have any cycle.

The cost of a reachability tree is the sum of the length of the roadways in the tree. The reachability tree with the smallest cost is referred to as the *mR-tree* from the given intersection. An mR-tree is denoted by an array, where the value at the index i denotes the parent of the intersection i. The parent of the source (I_S) will denoted by the source itself. The parent of intersections that are unreachable from I_S will be denoted by -1.

You are required to implement an algorithm to identify the mR-tree of intersections.

2 Encoding/Implementation Specification

2.1 Encoding Specification of Input: Intersections and Roadways

The city-planning commission provides the road network information using a text file. See Figure 1(a). The first line states the number of intersections n (in the example, $n = 10$) and the number of roadways m between intersections (in the example, $m = 13$).

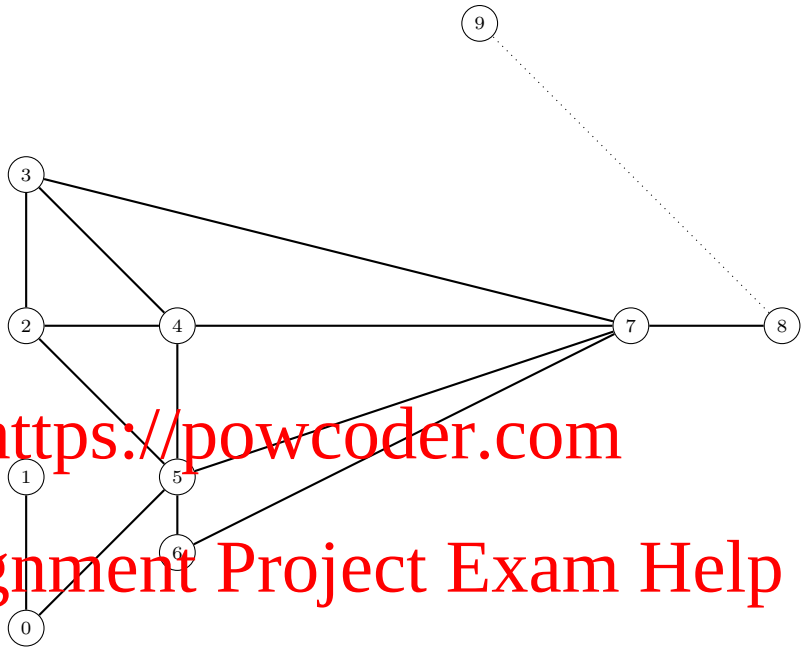
Each of the n ($n = 10$ in this example) lines immediately following the first line (there is no empty line after the first line) contains the intersection identifier followed by the coordinates of location. The value of the intersection identifier is between 0 and $n - 1$ (in the example, n is 10). Every intersection's location coordinate will be specified.

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

[1]Initially $d(s) = 0$ where s is the source vertex.

```
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.

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; city planning commission often includes future roadway information in their files. For analysis of the current state of the city, only the first m roadways should be considered. None of the entries for 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.

2.2 Class/Method Specification

2.2.1 MinHeap

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

1. It must have a default constructor, which creates a binary min-heap (value of k is 2).
2. It must have a constructor, which takes one int-type parameter, where the valuation k of the parameter results in creation of k-ary min-heap.
3. It must have a method `add`. It has parameters. The first parameter (int-type) describes the key 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 1-st till k indices in the array-list hold the keys of the children of the root of the heap, the $k+1$-th till $2k$-th indices of the array-list hold the keys of children of the 1-st element in the heap, and so on.

Usage listing. This is an example listing. It does not indicate any specific ordering in which the methods will 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
*/
```


2.2.2 PathFinder

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

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 2.1). The method does not return anything.
3. It must have a method `dist2Dest`. It has four formal parameters of type int:
   (a) first parameter denotes the source intersection s;
   (b) second parameter denotes the destination intersection t;
   (c) third parameter denotes the k value of the k-ary min-heap (see Section 1.1.1) to be used for the computing the shortest paths;
   (d) fourth parameter denotes the type of shortest path algorithm to consider (Dijkstra-v1 or Dijkstra-v2; see Section 1.1.2). When the value of the parameter is 1, the Dijkstra-v1 is applied; when the value of the parameter is 2, the Dijkstra-v2 is applied.

   The method outputs the M-path distance (see Section 1.1) from s to t; the return type is `double`.
4. It must have a method `path2Dest`. It has four formal parameters. The description of the formal parameters is same as `dist2Dest`.

The method returns a list of integers denoting a low-M-path (see Section 1.1) from s to t. The return type is `ArrayList<Integer>`. The i-th element in the ArrayList has the intersection that is i-edges away from s in the low-M-path between s and t. Note that 0-th element in the ArrayList is the intersection s.

5. It must have a method `dist2All`. It has two formal parameters of type int.
   (a) first parameter denotes the source intersection s;
   (b) second parameter denotes the k value of the k-ary min-heap (see Section 1.1.1) to be used for the computing the shortest paths;

   The method returns shortest roadway distances from s to all intersections. The return type of this method is `double[]`. The i-th element in the array is the shortest roadway distance from s to intersection i. (Intersections unreachable from s will have a distance -1.)

6. It must have a method `noOfMPaths2Dest`. It has three formal parameters of type int.
   (a) first parameter denotes the source intersection s;
   (b) second parameter denotes the destination intersection t;
   (c) third parameter denotes the k value of the k-ary min-heap (see Section 1.1.1) to be used for the computing the shortest paths;

   The method returns the number of M-paths from s to t. The return type of this method is `int`.

7. It must have a method `mRtreeFromSource`. It has two formal parameters of type int.
   (a) first parameter denotes the source intersection s;
   (b) second parameter denotes the k value of the k-ary min-heap (see Section 1.1.1) to be used for the computing the shortest paths;

   This method returns mR-tree from s; the return type of this method is `int[]` (see Section 1.2).

8. It must have a method `mRtreeCostFromSource`. It has two formal parameters of type int.
   (a) first parameter denotes the source intersection s;
   (b) second parameter denotes the k value of the k-ary min-heap (see Section 1.1.1) to be used for the computing the shortest paths;

   The method returns the cost of the mR-Tree from s (see Section 1.2). The return type is `double`.

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

```
PathFinder pf = new PathFinder();

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

// M-Path distance
System.out.println("M-Path Distance: " + pf.dist2Dest(0, 3, 2, 1));

/* produce the output:
   Distance of shortest paths: 38.2842712474619
*/


// A variant
System.out.println("M-Path Distance: " + pf.dist2Dest(0,  3, 2, 2));

/* produce the output:
   Distance of shortest paths: 38.2842712474619
*/
```

```
System.out.println("M-Path Distance: " + pf.dist2Dest(0,  9, 2, 2));

/* produce the output:
   Distance of shortest paths: -1.0
*/

// Number of M-Paths
System.out.println("Number of shortest paths: " + pf.noOfMPaths2Dest(0, 3, 2));

/* produce the output:
   Number of shortest paths: 2
*/


// low-M-Path
ArrayList<Integer> path = pf.path2Dest(0, 3, 2, 1);
if (path == null) System.out.println("No path to destination");
else {
  for (int i=0; i<path.size(); i++) System.out.printf("%s ", path.get(i));
  System.out.println();
}
/* produce the output:
      0    5    2    3
*/

// M-Path distance to all vertices
double[] distances = pf.dist2All(0, 2);
for (int i=0; i<distances.length; i++) System.out.println("distance to "  + i + " is " + distances[i]);
System.out.println();

/*  produce the output: (NOTE DISTANCE TO 9 IS -1 AS 9 IS UNREACHABLE)
    distance to 0 is 0.0
    distance to 1 is 10.0
    distance to 2 is 28.284271247461902
    distance to 3 is 38.2842712474619
    distance to 4 is 24.14213562373095
    distance to 5 is 14.142135623730951
    distance to 6 is 19.142135623730951
    distance to 7 is 84.85281374238571
    distance to 8 is 89.8528137423857
    distance to 9 is -1.0

*/

/*
 *  R-Tree
 */
// cost mR-Tree
System.out.println("Cost of mR-Tree: " + pf.mRtreeCostFromSource(0, 2));

/*  produce the output:
    Cost of mR-Tree: 134.14213562373095

*/


// mR-tree
int[] parents = pf.mRtreeFromSource(0, 2);
for (int i=0; i<parents.length; i++) System.out.println("parent of "  + i + " is " + parents[i]);

/* produce the output: NOTE PARENT OF 9 IS -1 AS 9 IS UNREACHABLE
   parent of 0 is 0
   parent of 1 is 0
   parent of 2 is 4
   parent of 3 is 2
   parent of 4 is 5
   parent of 5 is 0
   parent of 6 is 5
   parent of 7 is 4
   parent of 8 is 7
   parent of 9 is -1
*/
```

3 Submission File Specification

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. 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.

You can develop your code in a way that works for you; write classes in multiple files, use package directives, etc. However, make sure when you submit, you submission adheres the above specifications.

4 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 M-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.