LAB 3 - Prim's Algorithm for Minimum Spanning Tree

Submission Deadline: March 9, 11:59 pm
Assessment: 5% of the total course mark.

Description:

In this assignment you are required to write a `Java` implementation of Prim's algorithm for finding a minimum spanning tree in an undirected graph. You have to implement the algorithm using a binary min-heap for the priority queue. For this, you have to write the `Java` classes `Graph`, and `MinBinHeap`. Class `Graph` represents undirected graphs using adjacency lists. Classes `Vertex` and `Edge`, which are needed for this representation, are given to you. **They have to be declared in the same package as `Graph`, and `MinBinHeap`.** Class `MinBinHeap` represents binary min-heaps customized for Prim's algorithm (in other words, they store items of type `Vertex` and they only support the operations needed for this algorithm). Class `Vertex` is also tailored for this specific application.

You are not allowed to use any predefined `Java` methods other than for input and output.

Your code has to run with the classes `Vertex` and `Edge` specified in this assignment.

Explanation:

◇ **The declaration of class `Vertex` is**

```
public class Vertex{
    Vertex prev;  //previous vertex in MST (variable v.p from lecture notes)
    int index; //the index in the array of vertices
    int heapIndex; //the index in the min-heap array
    int key=Graph.infinity; // key in min-heap; initialized as logical infinity
    boolean isInQ = true; //true when the vertex is in min-heap;
                          //when the Vertex object is created,
                          //it is assumed to be in the min-heap

    public Vertex(int n){
            index = n;
    }//end constructor
}//end class
```

Note that all the fields have package access. Therefore, they can be accessed directly from any method declared in the same package.

◇ **The declaration of class `Edge` is**

```
public class Edge {
    int weight;
    Edge next; //reference (or pointer) to the next edge in the adjacency list
    Vertex endPoint; //for edge (u,v) in the adjacency list of u, endPoint=v;
                     //for edge (u,v) in the adjacency list of v, endPoint=u

    public Edge(Edge e, Vertex v, int w) {
        weight = w;
        endPoint = v;
        next = e;
    }//end constructor
}//end class
```

Note that all the fields have package access. Therefore, they can be accessed directly from any method declared in the same package.

◇ **The partial declaration of class `Graph` is given next. You have to complete the declaration according to the specification given in the comments.**

```
import java.util.Scanner;
public class Graph {
    public static final int infinity = 10000; //logical infinity
    Vertex[] v; //array of vertices
    Edge[] adj; //array storing headers of adjacency lists  (adj[i] is
                //a reference to the first Edge object in the adjacency
                //list of vertex v[i])
    int size=0; //number of vertices

//Constructor: constructs the undirected graph described by the input string;
//the string contains only non-negative
//integers separated by white spaces; the first integer is
//the number of vertices (n); each of the following triples of integers
//specifies an edge, namely end1, end2 and weight, where end1 and end2
//are the indexes of the endpoints in the array of vertices
//and weight is the weight of the edge; you may assume that the input string
//respects the required format, that 0<=end1<=n-1, 0<=end2<=n-1,
//and that the input represents a connected graph

    public Graph(String inputString) {
        Scanner input = new Scanner(inputString);
        size = input.nextInt();
        v = new Vertex[size]; //alllocate the array of vertices
        //create the Vertex objects and place them in the array
                ...
```

```
     adj = new Edge[size]; //alllocate the array of headers to adjacency lists
    //read the info from the string
    int end1;
    int end2; int w;
    while(input.hasNext()){
        //read next edge
        end1 = input.nextInt();
        end2 = input.nextInt();
        w = input.nextInt();
    //create an edge with endPoint=end2 and
    //insert it in the adjacency list of  end1
    //create an edge with endPoint=end1 and
    //insert it in the adjacency list of end2
    //each insertion is made at the beginning of the list
              ...
    }//end while
}//end constructor

public String adjListString() {
    Edge p; //edge pointer
    String s = " ";
    for(int i=0; i<size; i++) {
        p = adj[i]; //p points to first edge in the adjacency list of v[i]
        //scan adjacency list of v[i]
        while(p != null) {
            s += " \n edge: (v" + i +", v" + p.endPoint.index + "), weight: "
                        + p.weight;
            p = p.next; //move to next edge in the current list
        }//end while
    }// end for
    return s;
} // end method

//minSTPrim(int r): finds a minimum spanning tree using Prim's algorithm
//implemented with a min-heap, starting at vertex v[r];
//returns a string that lists all edges in the MST,
//in the order they were found; see the output of the test class
//for clarification on the format of the string;
//you may assume that r is a valid index in the vertex array
//and that the graph is connected

public String minSTPrim (int r) {
              ...
}//end method
```

```
    //adjMatrix(): returns a two-dimensional array that represents
    //the adjacency matrix of the graph

    public int[][] adjMatrix(){
            ...
    }//end method
}//end class
```

Note that all the fields have package access. Therefore, they can be accessed directly from any method declared in the same package. You may declare additional methods in this class.

◇ **The partial declaration of class `MinBinHeap` is given next. You have to complete the declaration according to the specification given in the comments.**

```
public class MinBinHeap {
    Vertex[] heap;
    int size = 0;

    //Constructor: allocates the heap array, sets the key of v[r] to 0 and
    //places v[r] at the root; sets the keys of the remaining vertices
    //to logical infinity and copies them in the heap;
    //initializes heapIndex for each vertex appropriately

    public MinBinHeap(Graph g, int r) {
           ...
    }//end constructor

    //NOTE: When creating the min-heap in the method minSTPrim, you need to pass
    //a reference to this Graph object; use: new MinBinHeap(this, r);

    //extractMin: returns the vertex with the smallest key and removes it from
    //the heap; note that every time a change is made in the heap,
    //the heapIndex of any vertex involved in the change has to be updated

    Vertex extractMin() {
           ...
    }//end method

    //decreaseKey(int i, int newKey): decreases the key of the vertex stored
    //at index i in the heap; newKey is the new value of the key and it is
    //smaller than the old key; NOTE: after the change, the heap ordering property
    //has to be restored - use percolate up

    void decreaseKey(int i, int newKey) {
           ...
    }//end method
```

```
    public String toString(){
        String s = "\n The heap  size is " + size + "\n The items' labels are: \n";
        for(int i=1; i < size+1; i++) {
            s += heap[i].index + " key: ";
            s += heap[i].key + "\n";
        } //end for
        return s;
    }//end method
}//end class
```

You may declare additional private methods in this class.

SUBMISSION INSTRUCTIONS:
- Submit the source code for each of the `Java` classes `Graph` and `MinBinHeap` in a separate text file. Include the name of the class, your name and student number in the name of the file.

VERY IMPORTANT: **Your code has to run with the classes `Vertex` and `Edge` specified in this assignment.**