Problem1

Yihang Wang

March 2020

1 Specification

1. package hw4;

```
/**
 * Node represents a immutable single element which composed to a whole graph
 * It is implemented as a single class
 * This class is invisible by the client
 */
public final class Node implements Comparable<Node>{
    private String value;

    // Abstraction Function:
    // a node represents a single part of graph

    // Representation Invariant for every Node n:
    // value != NULL

    /**
     * @effects Constructs a new empty node
     */
    public Node(){
        throw new RuntimeException("Constructor without parameter
        has not been implemented!");
    }

    /**
     * @effects Constructs a new node with value
     */
    public Node(String s){
        throw new RuntimeException("Constructor with parameter
        has not been implemented!");
    }

    /**
 * @return the value of node
 */
public String getValue(){
    throw new RuntimeException("getValue has not been impelemented!");
}

/**
 * check that representation invariant if holds
 */
private void checkRep(){
    if (value == null){
        throw new RuntimeException("there is no value");
    }
}

/** Standard hasCode function.
 *
 * @return an int that all objects equal to this will also return.
 */
@Override
public int hashCode(){
    int num = 0;
    for (int i = 0; i < value.length(); i++){
        num += (i+1)*value.indexOf(i);
    }
    return num;
}

/**
 * Standard equally operation
 * @param obj The object to be compared for equality
 * @return true if and only if 'obj' is an instance of a
 *         Node and 'this' and 'obj' represent the same
 *         node.
 */
@Override
public boolean equals(/*@Nullable*/Object obj) {
    if (obj instanceof Node) {
        Node n = (Node) obj;
        if (value.length() != n.value.length()){
            return false;
        }
        for (int i = 0; i < value.length(); i++){
            if (value.indexOf(i) != n.value.indexOf(i)){
                return false;
            }
```

```
                }
                return true;
            }
            return false;
        }

        /**
         * Compare two nodes' value
         * @param node
         * @return 0 if node's value smaller than this; other wise return 1.
         */
        @Override
        public int compareTo(Node node) {
            int i = 0;
            for (; i < this.value.length() && i < node.value.length(); i++){
                if (this.value.indexOf(i) < node.value.indexOf(i)){
                    return 1;
                }else if (this.value.indexOf(i) > node.value.indexOf(i)){
                    return 0;
                }
            }
            if (i < node.value.length()){
                return 1;
            }else{
                return 0;
            }
        }
    }
```

2. package hw4;

```
import java.util.Comparator;

/**
 * Edge represents mutable linked relationship from one node to another one
 * It is implemented as a single class
 * This class is invisible by the client
 */
public class Edge implements Comparable<Edge> {

    private Node root;
    private Node next;

    // abstraction function:
    // an edge e which have root and next represents
    // there is a path from root to next

    // representation invariant:
    // root != null && next != null

    /**
     *
     * @param r represents the node points to the next
     * @param n represents the node from the root
     * @effects Constructs an edge has root = r and next = n
     */
    public Edge(Node r, Node n){
        throw new RuntimeException("Constructor has not been implemented!");
    }

    /**
     * @return the root of this edge
     */
    public Node getRoot(){
        throw new RuntimeException("getRoot has not been implemented!");
    }

    /**
     * @return the next of this edge
     */
    public Node getNext(){
        throw new RuntimeException("getNext has not been implemented!");
    }

    /**
```

```
     * check if the representation invariant holds
     */
    // Throw a Runtime Exception if the representation invariant violates
    public void checkRap(){
        if (root == null || next == null){
            throw new RuntimeException("this edge is not valid!");
        }
    }

    /**
     * Compare two nodes' value
     * @param e The Edge to be compared
     * @requires e != NULL
     * @return a negative number if this < e,
     * 0 if this = e,
     * a positive number if this > e
     */
    @Override
    public int compareTo(Edge e) {
        return this.next.compareTo(e.next);
    }
}
```

3.
```
package hw4;
/**
 * Graph represents a mutable collection of nodes and edges
 * It is implemented as a single class.
 * This class is invisible by the client
 */
public class Graph {

    // abstraction function:
    // Utilize map to represent graph with node key and edge values to
    // correspond this node’s outdegree edge.
    // which is basically same idea with adjacency list

    // representation invariant:
    // The node number and edge number must be greater or equal to 0.

    private int nodeNumber;
    private int edgeNumber;

    /**
     * @effects Constructs a new empty graph
     */
    public Graph(){
        throw new RuntimeException("The graph constructor without parameters
        has not been implemented!");
    }

    /**
     * @param e The edge in the term which the graph constructs
     * @requires e != null
     * @effects Constructs a new Graph for node n has edge e.
     */
    public Graph(Edge e){
        throw new RuntimeException("The graph constructor with node and edge
        as parameters has not been implemeneted!");
    }

    /**
     * @param edges An array of edges to be contained in the new Graph
     * @requires ’edges’ is non-empty and it satisfies clauses
     * given in rep. invariant
     * @effects Constructs a new Graph using ’edges’ as part of the representation.
     */
    public Graph(Edge[] edges){
        throw new RuntimeException("The graph constructor with
        several edges has not been implemented!");
    }

    /**
     * @param n The node in the term which the graph add
     * @requires n != null
     * @effects add a new node n to the graph
     */
    public void addNode(Node n){
        throw new RuntimeException("The addNode method for Graph
        has not been implemented!");
    }

    /**
     * @param e The edge in the term which the graph add
     * @requires e != null
     * @effects add a new edge e to the graph
     */
    public void addEdge(Edge e){
        throw new RuntimeException("The addEdge method for Graph
        has not been implemented!");
    }
}
```