hw4 problem 1

Brian Wang-Chen

March 2024

1 Specification for Graph

```
/**
 * <b>Graph<b> Represents a mutable, directed graph
 * <b>Graph<b> representation in a hashmap where each key element represents a node
 * and each value of that node represents it outgoing edges or child nodes and the label for that edge
 *
 * Specified Node: the node or vertices of the graph, represented by a string
 * specified Edges: the edges of the graph, represented by a AbstractMap containing 2 strings: childnod
 *
 */
public class Graph {
    // Rep invariant:
    // graph != null
    // Every node and edge != null
    // handshake theorem applies to graph to ensure proper graph

    // Abstract Function:
    // this == directed graph g such that if g is empty then {}
    // if node a in nodes of graph g, {a = [outgoing edges]}
    // outgoing edges include the destination nodes node a reaches
    // each outgoing edge will have a weight value too

    // storing graph in adjacency list
    // Map where each key element is a string representing each node in graph
    // Each value of each key is a set of edges
    // edges represented in a AbstractMap.SimpleEntry, first element is the child node, second is edgel
    private Map<String, Set<Pair<String, String>>> graph;
    // sum of degree of all vertices
    private int sum_degree = 0;
    // number of edges in graph
    private int num_edges = 0;

    // representing a empty graph
    public static final Graph emptyGraph = new Graph();

     /**
     * Creates a empty directed graph
     * @param none
     * @effects Construct a new empty directed raph with no nodes or edges
     * @requires none
     * @modifies none
     * @returns none
     * @throws none
     */
    public Graph(){
        throw new RuntimeException("Graph constructor is not yet implemented");
    }
```

```
/**
 * Adds a node to the graph
 * @param nodeData the new node being added into graph
 * @effects Adds a new node to the graph if non existent, graph.keySet()
 * @requires nodeData != null
 * @modifies graph by adding a new node to the graph, graph.keySet
 * @returns none
 * @throws IllegalArgumentException if nodeData is null
 */
public void addNode(String nodeData){
    throw new RuntimeException("Graph addNode is not yet implemented");
}

/**
 * Creates a edge from parent to child with edge label in graph
 * @param parentNode the node in graph which new Edge is being made
 * @param childNode the child the parent node is being connected to, to create edge
 * @param edgeLabel label of the edge, or the weight
 * @requires all parameters != null
 * @modifies the edges of this graph, add new edge to node
 * @modifies the outgoing edges of parent node
 * @effects add edge with label to graph if the edge does not already exist
 * @throws IllegalArgumentException if either parentNode or childNode is not
 *                                 in graph, graph.keySet
 * @throws IllegalArgumentException if any param is null
 * @returns none
 */
public void addEdge(String parentNode, String childNode, String edgeLabel){
    throw new RuntimeException("Graph addEdge is not yet implemented");
}

/**
 * returns a iterator which represent nodes in lexicographical order
 * @param none
 * @returns a iterator of nodes in lexicographical order
 * @requires none
 * @modifes none
 * @effects none
 * @throws none
 */
public Iterator<String> listNodes(){
    throw new RuntimeException("Graph Iterator is not yet implemented");
}

/**
 * returns a iterator which represent the list of childNode(edgeLabel) in lexicographical order
 * @param parentNode a node in graph
 * @returns a iterator of childNodes(edgeLabel) of parentNode in lexicographical order
 * @returns a empty iterator if parentnode is not in graph or if parentNode is null
 * @requires parentNode != null
 * @modifies none
 * @effects none
 * @throws none
 */
public Iterator<String> listChildren(String parentNode){
    throw new RuntimeException("Graph Iterator is not yet implemented");
}
/**
 * Comparator to properly sort the children nodes or edges of a parentndoe
 * lexicographical order
 * @Override TreeSet<Pair<String,String>> comparator
```

```
     */
    Comparator<Pair<String, String>> compareEdge = new Comparator<Pair<String, String>>(){
        throw new RuntimeException("Comparator is not yet implemented");
    }

    @Override
    /**
     * Overidden equals method to compare two Graphs since by default
     * equals method in java compares memory addresses of objects, this
     * method will allow comparison by Graph value
     * @param obj object to compare to
     * @returns boolean indicating the equality of two Graphs
     * @modifies none
     * @effects none
     * @requires none
     * @throws none
     */
    public boolean equals(Object obj) {
        throw new RuntimeException("equals override is not yet implemented");
    }
    /**
     * Determine if a node is within graph, returns true if within, false otherwise
     * @param node the node to check if within graph
     * @returns boolean indicating if node is within graph
     * @modifes none
     * @effects none
     * @requires none
     * @throws none
     */
    public boolean containsNode(String node){
        throw new RuntimeException("containsNode is not yet implemented");
    }
    /**
     * Determine if a edge is within graph, returns true if within, false otherwise
     * @param parent the parent node of edge
     * @param edge the pair indicating child node and edge label of edge
     * @returns boolean indicating if edge is within graph
     * @requires none
     * @modifes none
     * @effects none
     * @throws none
     */
    public boolean containsEdge(String parent, Pair<String, String> edge){
        throw new RuntimeException("containsEdge is not yet implemented");
    }

}
```