Homework 4 Problem 1

Glen Madsen

March 8, 2018

Problem 1.

Overview: I will be using a HashMap with String keys and a HashMap as its value set. Then, in the second HashMap, i will be using again String keys, but an ArrayList<String> as the value. This will allow me to map each node to another map of its children and their corresponding edgelabels. It will also natively prevent duplicate nodes from being added, allow the program to run quickly, and be easy to implement and test.

```
/**
* Abstraction Function:

* Graph represents a directed labeled multigraph where each key (a String) in the nodes keySet is a node in the graph and each nodes' keys map to a HashMap of all of it's children as keys (Strings) and an ArrayList¡String¿ as a list of edgeLabels for every edge.
*
* So there is a set of nodes (possibly empty) in the first level of the map, with each node containing a HashMap with set of its children (nodes it directs to in a one way manner and possibly empty) as keys. This maps to an array of all of the edges from the parentnode to the childnode, which are labeled via the edgeLabels. This represents the directed labeled multigraph. There should be no duplicate nodes but duplicate edges between nodes are allowed.
*

* Representation Invariant for every Graph g:
* (g.nodes.containsKey(null) == false)
* && (g.nodes.keySet() != null)
* && (g.nodes.values() != null)
* && (g.nodes.containsValue(null))
* && (for all keys in keyset g.nodes.get(key).containsKey(null) == false)
* && (for all keys in keyset g.nodes.get(key).keySet() != null)
* && (for all keys in keyset g.nodes.get(key).values() != null);
```

* && (for all keys in keyset g.nodes.get(key).get(key in other keyset).contains(null) == false);
* && (for all keys in keyset of g there are no duplicate keys)

* In other words
* *The keyset of nodes does not contain null and is not null.
* *The values of the nodes HashMap is not null and does not contain null.
* *The keyset of every HashMap mapped to by a key in nodes does not contain null and is not null.
* *The values of every HashMap mapped to by a key in nodes is not null and does not contain null.
*/

Graph Class Implementaion Followed by Operations and specifications for each.

public class Graph

private HashMap<String, HashMap<String, ArrayList<String>>> nodes = new HashMap<String, HashMap<String, ArrayList<String>>>();

/**
* @param N/A
* @returns N/A
* @throws N/A
* @requires N/A
* @effects constructs a new HashMap<String, HashMap<String, ArrayList<String> > > equal to null
* @modifies nodes variable is created
*/
public Graph()
{
this.nodes = new HashMap¡String, HashMap<String, ArrayList<String>
}

/**
* @param nodeData is the label of the new node
* @returns N/A
* @throws N/A
* @requires nodeData to be a non-null String

```
* @effects places a new HashMap<String, ArrayList<String> > equal to
null in nodes values with a key of nodeData mapped to it.
* If nodeData is already in the keySet of nodes then nothing is
modified.
* @modifies nodes values and keySet if nodeData is not in the keySet of nodes.keySet()
*/
public void addNode(String nodeData)
{

}

    /**
* @param parentNode is the String key of the nodes HashMap, childNode is the key to
be added or a key already of the HashMap mapped to by parentNode, and edgeLabel is
the string to be added to either a prior existing or new ArrayList<String> mapped to by
childNode in the HashMap mapped to by the parentNode key in the nodes HashMap.
* @returns N/A
* @throws N/A
* @requires parentNode, childNode, and edgeLabel to be a non-null String. If either par-
entNode or childNode do not exist in the keySet of nodes then nothing is modified.
* @effects If either the nodes keySet does not contains parentNode or childNode the it does
nothing
* else:
* If the childNode exists in the keySet of the nodes.get(parentNode).keySet() HashMap then
it adds a String edgeLabel to the ArrayList mapped by the childNode in the values of that
HashMap.
*
* If the childNode does not exist in the keySet of the nodes.get(parentNode).keyset() HashMap,
then it adds a String childNode to that hashmaps keyset and maps it to a new ArrayList<String>
containing only the string edgeLabel.
*
* Any duplicate edges going from the same parentNode to childNode with the same edge-
Label will still be added to the ArrayList¡String¿ so duplicates are allowed.
*
* @modifies If the nodes keySet contains parentNode and childNode, it either modifies
* the ArrayList<String> in the values of the HashMap mapped by nodes.get(parentNode)
if
* childNode already exists in nodes.get(parentNode).keySet()
* or it modifies ArrayList<String> in the values of the HashMap mapped by nodes.get(parentNode)
* and the corresponding keySet of the Hashmap
*/
public void addEdge(String parentNode, String childNode, String edgeLabel)
```

{

}
/**
* @param N/A
* @returns returns an iterator to a new TreeSet containing the keySet of nodes
* This iterator returns the nodes in lexicographical (alphabetical) order.
* @throws N/A
* @requires
* @effects N/A
* @modifies N/A
*/
public Iterator¡String¿ listNodes()
{

}
/**
* @param
* @returns returns If parentNode is not in the keySet then it returns null, if the keySet is empty then it returns null. If parentNode is in the keySet then it returns an iterator to a new TreeSet containing each edgeLabel in every ArrayList<String> mapped to by each key in the keySet of the HashMap mapped to by the parentNode.
*
* The iterator returns the list of childNode(edgeLabel) in lexicographical (alphabetical) order by node name and secondarily by edge label. childNode(edgeLabel) means there is an edge with label edgeLabel from parentNode to childNode. This includes reflexive edges and duplicate edges, listing each edge as a seperate entry.
* @throws N/A
* @requires parentNode to be a non-null String
* @effects N/A
* @modifies N/A
*/
public Iterator¡String¿ listChildren(String parentNode)
{

}

private void checkRep() throws RuntimeException
{

}
}