// Zakery Cumbie

// CSE 464 Fall 2023

A Part 2 section has been created on page 5 that covers the changes made to the github for Project Part 2.

The Main.java file hosts the methods that implement the four features, those being parseGraph, addNode & addNodes, addEdge, and outputDOTGraph. The main

method is a driver that will manually put some inputs into the methods so that the outputs of the methods can be seen, some are errors and some are standard outputs.

For example,

parseGraph("C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

Will return the parsed information from the input.dot file, which returns

Number of Nodes: 4

Number of Edges: 5 Node

Labels: [A, B, C, D]

Edge Directions:

A -> B

B -> C

C -> A

A -> C

D -> B

The parseGraph method uses a file path in string form to find input. The input should follow the format "X -> X" or the information will not be stored.

So long as the given input is formatted correctly, it will sort for duplicate nodes and edges, and give numbers for both.

addNode(String s)

This method will add a Node labeled by the given input string. It will first check that the node does not already exist in the graph object.

addNodes(String[] labels)

This method performs the same function as addNode but accross multiple strings for various labels. Checks for repeat labels are done at each instance of a label being processed.

addEdge(source, destination)

This method adds an edge, starting from source and pointing towards destination. Checks for edges in the SAME direction will catch duplicates, but in the case of A -> B, B -> A, these edges will be added as they are valid.

outDOTGraph(String filePath)

This method will use a filePath to take the current instance of graph object and turn it into a dot file.

Below are pictures of sample input and output that I achieved through using the main function. In order to see the console output of each function, use the “main” method as a driver.

```java
public static void main(String[] args) {
    System.out.println("\n...Starting Program... \n");

    parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    addNode( label: "E");
    addNode( label: "E");

    String[] newNodes = {"F", "G", "H", "F"};
    addNodes(newNodes);

    addEdge( srcLabel: "F",  dstLabel: "G");
    addEdge( srcLabel: "F",  dstLabel: "G");
    addEdge( srcLabel: "G",  dstLabel: "F");

    outputDOTGraph( path: "C:/Users/ldf08/IdeaProjects/CSE464Project/input2.dot");

    System.out.println("\n...Ending Program... \n");
}
```

```
...Starting Program...

Number of Nodes: 4
Number of Edges: 5
Node Labels: [A, B, C, D]
Edge Directions:
A -> B
B -> C
C -> A
A -> C
D -> B

Node E already exists!
Node F already exists!
Node F and Node G already have an edge in that direction!
Graph exported to C:/Users/ldf08/IdeaProjects/CSE464Project/input2.dot

...Ending Program...
```

The file featureTester, contains the unit tests for each feature to be used by maven. These do not output anything unless an error is caught in testing, but will test for the expected outputs from each feature.

----------Part 2 Section----------

public static void removeNode(String label)

- Function that removes a specified node, represented as a string.

public static void removeNodes(String[] labels)

- Function that removes an array of nodes, all contained in the String Array labels.

public static void removeEdge(String srcLabel, String dstLabel)

- Function that takes two string representations of nodes and removes the edge between them.

These three functions all check that the given node(s) exist in the graph and removeEdge checks that the edge between nodes exists before being deleted. Otherwise, an exception is thrown.

featureTest.java----------

This file has been modified with additional tests for the new removal functions. These test both successful removal of some nodes, exception throwing for removal of node(s)/edges that don't exist in the graph.

```java
@Test
public void testRemoveNodeExistingNode() {
    Main.parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    Main.addNode( label: "V");
    Main.addNode( label: "P");
    Main.addEdge( srcLabel: "V",  dstLabel: "P");

    Main.removeNode( label: "V");

    assertFalse(Main.graph.containsVertex( v: "V"));
    assertTrue(Main.graph.containsVertex( v: "P"));
    assertFalse(Main.graph.containsEdge( sourceVertex: "V",  targetVertex: "P"));
}

@Test(expected = NodeNotFoundException.class)
public void testRemoveNodeNonExistingNode() {
    Main.parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    Main.removeNode( label: "Z");
}
```

These tests are for the remove Node function. The test is run on an input graph that I made with irrelevant nodes and edges. So long as a .dot file is parsed with parseGraph, the rest of the code will run. The inputs can also be changed to any string. The first test ensures that not only is the specified vertex/node deleted, but also that its relevant edge has been deleted too. The second test checks for exceptions thrown by the function when a node that doesn't exist in the graph is called for removal.

```java
@Test
public void testRemoveNodesExistingNodes() {
    Main.parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    String[] labels = {"L", "N", "U"};
    String[] labels2 = {"L", "U"};

    Main.addNodes(labels);

    Main.removeNodes(labels2);

    assertFalse(Main.graph.containsVertex( v: "L"));
    assertFalse(Main.graph.containsVertex( v: "U"));
    assertTrue(Main.graph.containsVertex( v: "N"));
}

@Test(expected = NodeNotFoundException.class)
public void testRemoveNodesNonExistingNodes() {
    Main.parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    String[] labels = {"K", "J", "R"};

    Main.removeNodes(labels);
}
```

These tests are for the remove Nodes function. The test is run on an input graph that I made with irrelevant nodes and edges. So long as a .dot file is parsed with parseGraph, the rest of the code will run. The inputs can also be changed to any string. The first test ensures that not only is the specified vertexs/nodes deleted, but also that its relevant edge has been deleted too. The second test checks for exceptions thrown by the function when a node that doesn't exist in the graph is called for removal. This second test also checks each string exists in each iteration.

```java
@Test
public void testRemoveEdgeExistingEdge() {
    Main.parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    String[] labels = {"X", "Y", "Z"};

    Main.addNodes(labels);

    Main.addEdge( srcLabel: "X",  dstLabel: "Z");
    Main.addEdge( srcLabel: "Z",  dstLabel: "Y");
    Main.addEdge( srcLabel: "Y",  dstLabel: "X");

    Main.removeEdge( srcLabel: "Z",  dstLabel: "Y");

    assertFalse(Main.graph.containsEdge( sourceVertex: "Z",  targetVertex: "Y"));
    assertTrue(Main.graph.containsEdge( sourceVertex: "Y",  targetVertex: "X"));
    assertTrue(Main.graph.containsEdge( sourceVertex: "X",  targetVertex: "Z"));
}

@Test(expected = EdgeNotFoundException.class)
public void testRemoveEdgeNonExistingEdge() {
    Main.parseGraph( filePath: "C:/Users/ldf08/IdeaProjects/CSE464Project/input.dot");

    Main.addNode( label: "M");
    Main.addNode( label: "O");

    Main.removeEdge( srcLabel: "M",  dstLabel: "O");
}
```

These tests are for the remove Edge function. The test is run on an input graph that I made with irrelevant nodes and edges. So long as a .dot file is parsed with parseGraph, the rest of the code will run. The inputs can also be changed to any string. The first test inserts new nodes with edges between them and then removes one of them. It then checks to make sure that the edges not removed are present and the one specified for removal is removed. The second test adds new nodes and attempts to remove an edge between them that does not exist, which throws an exception.

The link below contains the full history of commits for the GitHub repository relating to the main branch, in which a feature was added or modified.

https://github.com/Khrone99/CSE-464-2023-zcumbie/commits/main