CSE 464: Software QA and Testing

Project Part #1

Sabthagirivasan Vellingiri

1225471286

svellin3@asu.edu

Introduction:

Part 1 of the project was built using **IntelliJ** as the IDE, **Java 8** as the Java Development Kit, **Maven** for Dependency Manager, and the **graphviz-java** library shared by the professor. The **graphviz-java** was used to parse the **.dot** file into objects in java and vice versa. Custom classes such as Graph, Node, and Edge were written to perform operations(such as adding or removing nodes or edges) on the given graph.

If there are any dependency issues, please resolve them using **mvn install** in the command line terminal or resolve them using maven in IntelliJ.

All the features required in part 1 are added into an Interface named "**GraphManager.java**" which is in the folder: **src/main/java/com/svellin3.** The implementation of all these features are implemented in **"GraphManagerImpl.java"** which resides in the folder: **src/main/java/com/svellin3/impl.**

How to run the application:

To run the application, run the main function that resides in the **Application.java** which can be located inside the folder: **src/main/java/com/svellin3**

The Command Line Terminal can be used to interact with the application. Initially, the application will ask for a **.dot** file to parse the graph. I have added my input file: **testInput.dot** to the project folder.

```
please enter the dot file to parse the graph:
testInput.dot
```

Once the dot file was parsed, the application will display a list of options to perform on the parsed graph. The screenshot of the options is attached below.

```
press the below options to perform actions:
1. print the graph
2. output to file
3. add a new node
4. add a list of new nodes
5. remove a node
6. remove a list of nodes
7. add an edge
8. remove an edge
9. output as DOT graph
10. output into graphics
11. exit
```

Option 1: print the graph

Selecting this option will print the graph in the command line terminal.

```
1
Number of Nodes: 4
Label of Nodes:[a, b, c, d]
Number of Edges: 4
digraph {
"a" -> "b"
"b" -> "c"
"c" -> "d"
"d" -> "a"
}
```

Option 2: output to a file

This option is used to write the output to a text file that describes the graph such as the number of nodes, the label of the nodes, the number of edges, the nodes, and the edge direction of the edges. The option asks for a filename as input.

```
2
Please enter the output filename:
graphDescription
```

This is the sample output file generated.

graphDescription.txt

```
1
2  Number of Nodes: 4
3  Label of Nodes:[a, b, c, d]
4  Number of Edges: 4
5  digraph {
6  "a" -> "b"
7  "b" -> "c"
8  "c" -> "d"
9  "d" -> "a"
10 }
```

Option 3: Add a new node

This option will add a new node to the graph. This option will ask for the node name. This feature is case-sensitive.

```
3
Please enter the node name:
A
```

Option 4: Add a list of new nodes

This option will add a list of new nodes to the graph. This option will ask for comma-separated node names. This feature is case-sensitive.

```
4
Please enter the list of node names as comma separated values:
f,g,h
```

Option 5: remove a node

This option will remove a node and its corresponding edges from the graph. This option will ask for the node name. This feature is case-sensitive.

```
5
Please enter the node name to be removed:
d
```

Option 6: remove a list of nodes

This option will remove a list of nodes and their corresponding edges from the graph. This option will ask for comma-separated node names. This feature is case-sensitive.

```
6
Please enter the list of node names to be removed as comma separated values:
a,b,f
```

Option 7: Add a new edge

This option will add a new edge to the graph. This option will ask for two inputs: source node name and destination node name. This feature is case-sensitive.

```
7
Please enter the source node:
b
Please enter the destination node:
d
```

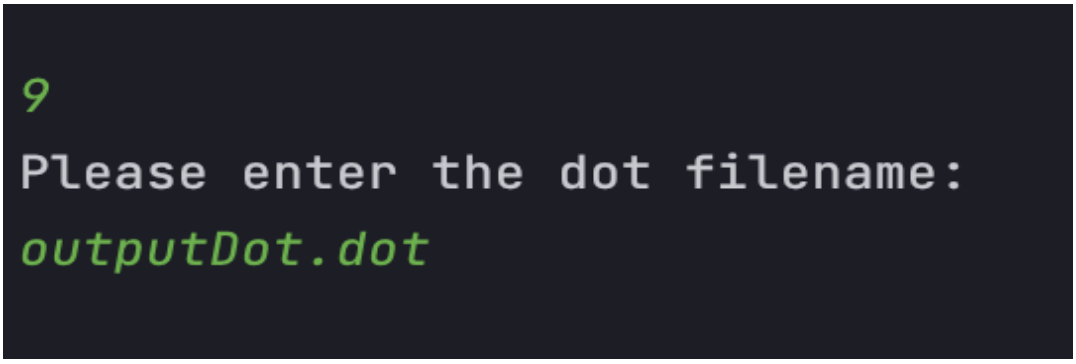
Option 8: Remove an edge

This option will remove an edge from the graph. This option will ask for two inputs: source node name and destination node name. This feature is case-sensitive.

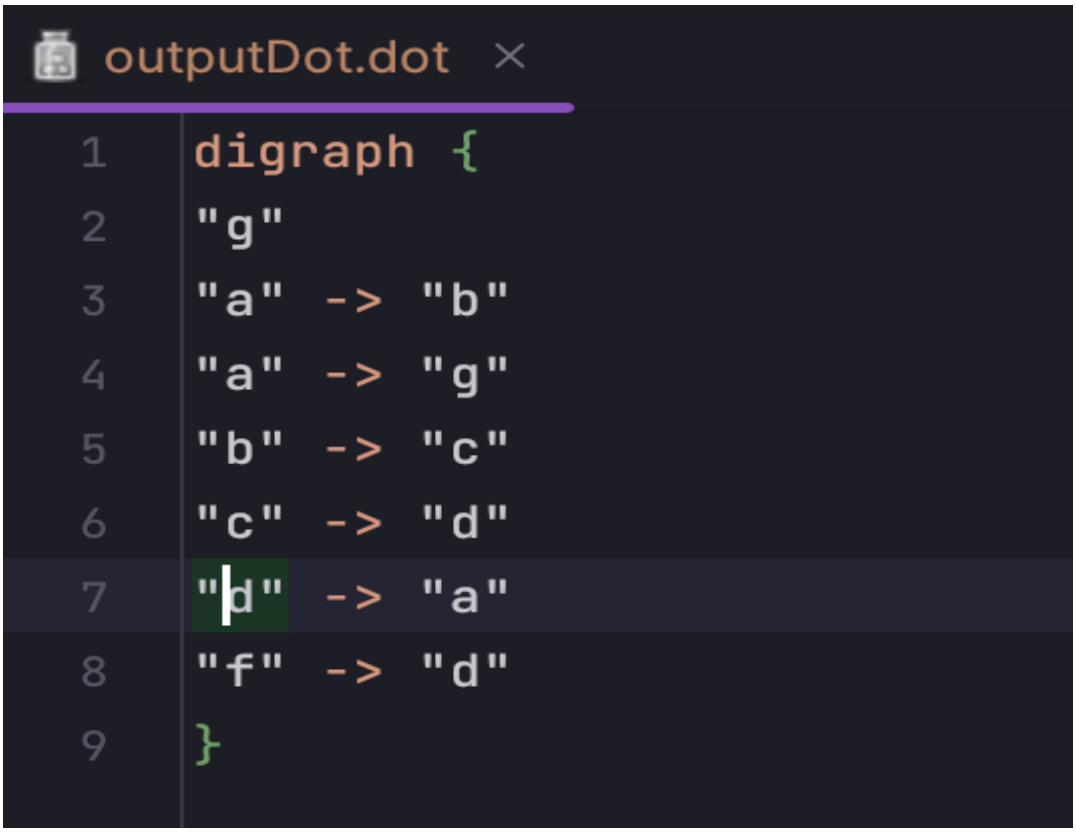
```
8
Please enter the source node:
d
Please enter the destination node:
a
```

Option 9: Outputs a dot graph file

This option will output the constructed graph in a .dot file. This option will ask for the desired output file name.



The sample generated .dot file is attached below.

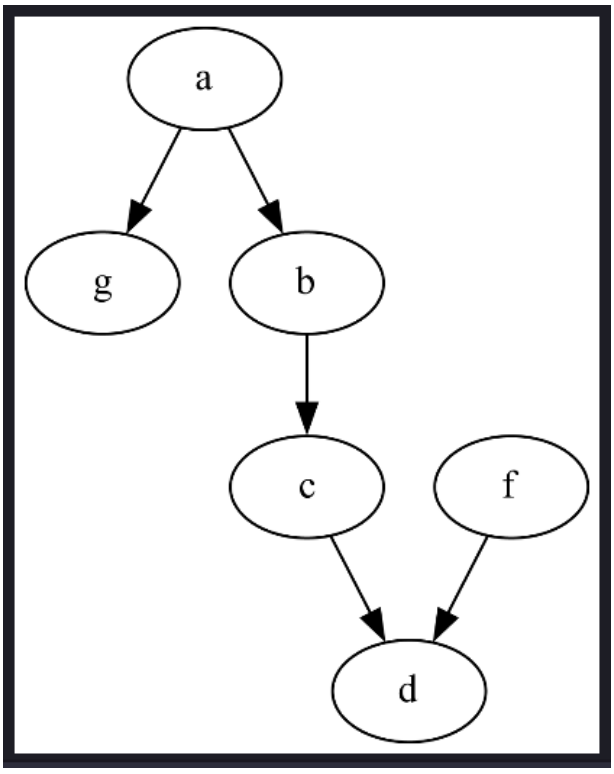


Option 10: Outputs into graphics file

This option will output the constructed graph into PNG or SVG file format. This option will ask for the desired output file name.

```
10
Please enter the graphics filename:
graphicsFormat
Please enter the format:
png
```

This is a sample-generated graphics output.



How to run the Test:

All the features required in part 1 are added into an Interface named "**GraphManager.java**" which is in the folder: **src/main/java/com/svellin3.** The implementation of all these features are implemented in **"GraphManagerImpl.java"** which resides in the folder: **src/main/java/com/svellin3/impl.**

The unit test cases were written for all the features implemented in part 1. The test class is added inside the folder "**src/test/java/com/svellin3**"**.** All the required features for part 1 are exclusively tested in the java file named "**GraphManagerTest.java**" which is located in the folder "**src/test/java/com/svellin3**". All other internal functions which are essential in achieving the required functionalities are tested in separate classes which reside inside **src/test/java/com/svellin3/impl**.

The test cases can be run using **IntelliJ** or **mvn test**.

I am hereby attaching the folder structure for reference.