CSE 464 Project Part 1

This Maven project, named "CSE-464-2023-amehta65", is a graph manipulator project which allows you to manipulate and work with graphs in the DOT file format. It offers several features to parse, modify, and visualize graphs. Below are instructions and example code to run the Maven project.

Prerequisites

Before running this project, ensure that you have the following prerequisites installed on your system:

1. Java Development Kit (JDK)
2. Apache Maven
3. Git (for cloning the project repository)

Getting Started

Follow these steps to get started with the Graph Manipulator project:

1. **Clone the Repository (optional):** You can clone the project repository from GitHub or download the source code as a ZIP archive.

```
git clone https://github.com/AviMehta90/CSE-464-2023-amehta65.git
```

2. **Navigate to the Project Directory:** Open a terminal or command prompt, and change your current directory to the project's root folder.

```
cd CSE-464-2023-amehta65
```

3. **Compile and Build the Project:** Use Maven to compile and build the project. This command will download the required dependencies and create an executable JAR file.

```
mvn clean package
```

4. **Run the maven project (JUnitTesting)**

```
mvn test
```

Features in `GraphManipulator` Class

Feature 1: Parse a DOT Graph File

`parseGraph(String filePath)`

This function parses a DOT graph file to create a graph. It takes the file path as an argument and reads the DOT file to build the graph. If successful, it sets the graph `g` and provides information about the number of nodes and edges in the parsed graph.

Information Functions

`getNumberOfNodes()`

This function returns the total number of nodes in the parsed graph.

`getNodeLabels()`

Returns a set of node labels in the parsed graph, including a label indicating the number of nodes.

`getNumberOfEdges()`

Returns the total number of edges in the parsed graph.

`getEdgeInfo()`

Returns a set of edge information in the parsed graph, where each entry represents an edge in the format "Source Node -> Destination Node."

`toGraphString()`

Returns a formatted string that combines information about the nodes and edges of the parsed graph.

`outputGraph(String filePath)`

This function writes the information about nodes and edges to a text file at the specified file path and returns the contents of the created file as a string.

Feature 2: Adding Nodes

`addNode(String label)`

Adds a node with the specified label to the graph. If a node with the same label already exists, it won't be added again. Returns `true` if the node is added successfully, or `false` if the node already exists.

`addNodes(String[] labels)`

Adds multiple nodes with the labels provided in the `labels` array. If any of the provided labels already exist in the graph, the function won't add them. Returns `true` if at least one new node is added, or `false` if all provided labels already exist.

Feature 3: Adding Edges

`addEdge(String srcLabel, String dstLabel)`

Adds an edge between two nodes in the graph based on their labels. The function ensures that both the source and destination nodes exist in the graph before adding the edge.

Feature 4: Output the Imported Graph

`outputDOTGraph(String filename)`

This function renders the imported graph as a DOT file and saves it with the specified filename. The rendered DOT file will be located in the specified directory.

`outputGraphics(String filePath)`

Visualizes the graph as a PNG image. It takes a path to a DOT file as input, reads the graph from that file, and generates a PNG image file. The generated image will be located in the specified directory.

These functions provide a wide range of functionalities for parsing, modifying, and visualizing DOT graphs. You can refer to the examples in the README to learn how to use these functions to work with graphs in your application.

Running the Application

Now that you've built the project, you can run the `GraphManipulator` application. Here are some examples of how to use it:

1. **Parsing a DOT File:**
    - To parse a DOT graph file and display node and edge information, use the following code:

```
GraphManipulator manipulator = new GraphManipulator();
manipulator.parseGraph("path/to/your/graph.dot");
System.out.println(manipulator.toGraphString());
```

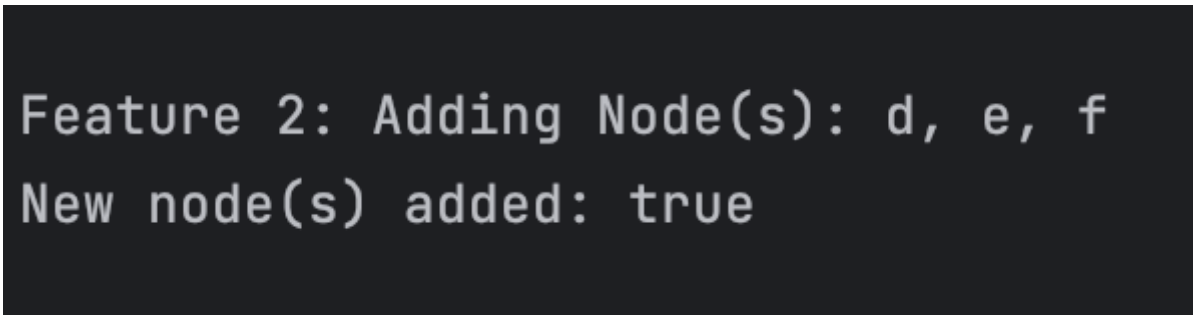
- **Console Output**

```
Feature 1: Parsing DOT Graph File
Dot File Parsed at src/main/resources/test1.dot

Dot File looks like:
digraph "G" {
"a" -> "b"
"b" -> "c"
"c" -> "a"
}

Number of Nodes: 3
Nodes: [a, b, c]
Number of Edges: 3
Edges: [b -> c, c -> a, a -> b]
Output String Graph Info at src/main/resources/GraphInfoAsString.txt
```

2. **Adding Nodes:**
   - You can add nodes to the imported graph using the `addNode` method. If the label already exists, it won't be added again.

```
GraphManipulator manipulator = new GraphManipulator();
manipulator.parseGraph("path/to/your/graph.dot");
manipulator.addNode("NewNode");
```

     - **Console Output**



3. **Adding Edges:**
   - To add edges between existing nodes in the graph, use the `addEdge` method. Provide the labels of the source and destination nodes.

```
GraphManipulator manipulator = new GraphManipulator();
manipulator.parseGraph("path/to/your/graph.dot");
manipulator.addEdge("Node1", "Node2");
```

- **Console Output**

```
Feature 3: Adding Edges
Edge created: a -> d
Edge created: e -> c
Edge created: f -> a
New edge(s) added

digraph "G" {
"a" -> "d"
"a" -> "b"
"b" -> "c"
"c" -> "a"
"e" -> "c"
"f" -> "a"
}
```

4. **Output as DOT File:**
   - You can output the imported graph as a DOT file with the following code. This will create a DOT file in the specified location.

```
GraphManipulator manipulator = new GraphManipulator();
manipulator.parseGraph("path/to/your/graph.dot");
manipulator.outputDOTGraph("output_graph.dot");
```

5. **Output as Graphics:**
   - To visualize the graph as a PNG image, provide the path to the DOT file and specify the output file path.

```
GraphManipulator manipulator = new GraphManipulator();
manipulator.outputGraphics("path/to/your/graph.dot");
```

- **Console Output**

```
Feature 4: Output the Imported Graph as DOT File
DOT file created at: graph_for_graphics.dot

Feature 4: Output the Imported Graph as Graphics
15:26:50.661 [main] INFO guru.nidi.graphviz.engine.GraphvizCmdLineEngine
15:26:50.663 [main] INFO guru.nidi.graphviz.service.CommandLineExecutor -
15:26:50.665 [main] DEBUG guru.nidi.graphviz.service.CommandLineExecutor
15:26:50.769 [main] INFO guru.nidi.graphviz.engine.GraphvizCmdLineEngine
Graph graphics generated at : src/main/resources/graph_for_graphics.dot

Process finished with exit code 0
```

Project Structure

The project's source code is organized as follows:

- `src/main/java/org/mehtaavi` : Contains the Java source code.
- `src/main/resources` : Place your DOT files here and specify the output directory for graphics.

Commit Links

1. First Commit
2. First Feature Added
3. Second Feature Added
4. Third Feature Added
5. Fourth Feature Added
6. Deleted 'example' directory
7. code formatted
8. Final Upload