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.

Feature 5: Part 2 Changes

`removeNode(String label)`

To remove a node, use the removeNode method and provide the label of the node you want to remove.

```
removeNodes(String[] label)
```

To remove multiple nodes, pass an array of strings and use the removeNodes method and provide the labels of the nodes you want to remove.

```
removeEdge(String srcLabel, String dstLabel)
```

To remove an edge, use the removeEdge method and provide the label of the source and destination nodes you want to remove.

```
graphSearch(srcNode, dstNode)
```

You can perform a graph search between two nodes using either a breadth-first or depth-first search algorithm. Use the graphSearch method with the desired search algorithm based on the enum in the main class. The path of the search algorithm will be stored in a class named Path which will output in the format "a -> b -> c"

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**

```
Feature 2: Adding Node(s): d, e, f
New node(s) added: true
```

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
```

6. **Remove Node(s):**

- To check node(s) removal functionality.

```
GraphManipulator manipulator = new GraphManipulator();
manipulator.parseGraph("path/to/your/graph.dot");
manipulator.addNode("Node1");
manipulator.addNodes(String[]{"Node2", "Node3"});
manipulator.removeNode("Node1");
manipulator.removeNode("NonexistentNode");
manipulator.removeNodes(String[]{"Node2", "Node3"});
manipulator.removeNodes(String[]{"NonExNode2", "NonExNode3"});
```

- **Console Output Remove Node(Test Case)**

```
✓ GraphManipulator 522 ms
  ✓ testParseGraph() 65 ms
  ✓ testOutputDOTG 16 ms
  ✓ testToGraphString 6 ms
  ✓ testAddEdge() 1 ms
  ✓ testAddNode() 1 ms
  ✓ testOutputGraph() 1 ms
  ✓ testRemoveNodes 2 ms
  ✓ testAddNodes() 1 ms
  ✓ testRemoveEdge() 1 ms
  ✓ testRemoveNode( 1 ms
  ✓ testOutputGrap 420 ms
  ✓ testGraphSearch() 7 ms

Dot File Parsed at src/main/resources/test1.dot
Number of Nodes: 3
Nodes: [a, b, c]
Number of Edges: 3
Edges: [b -> c, c -> a, a -> b]
Adding node d
Number of Nodes: 4
Nodes: [a, b, c, d]
Number of Edges: 3
Edges: [b -> c, c -> a, a -> b]
Node d removed.
Number of Nodes: 3
Nodes: [a, b, c]
Number of Edges: 3
Edges: [b -> c, c -> a, a -> b]
Node z not found.
```

- **Console Output Remove Nodes(Test Case)**

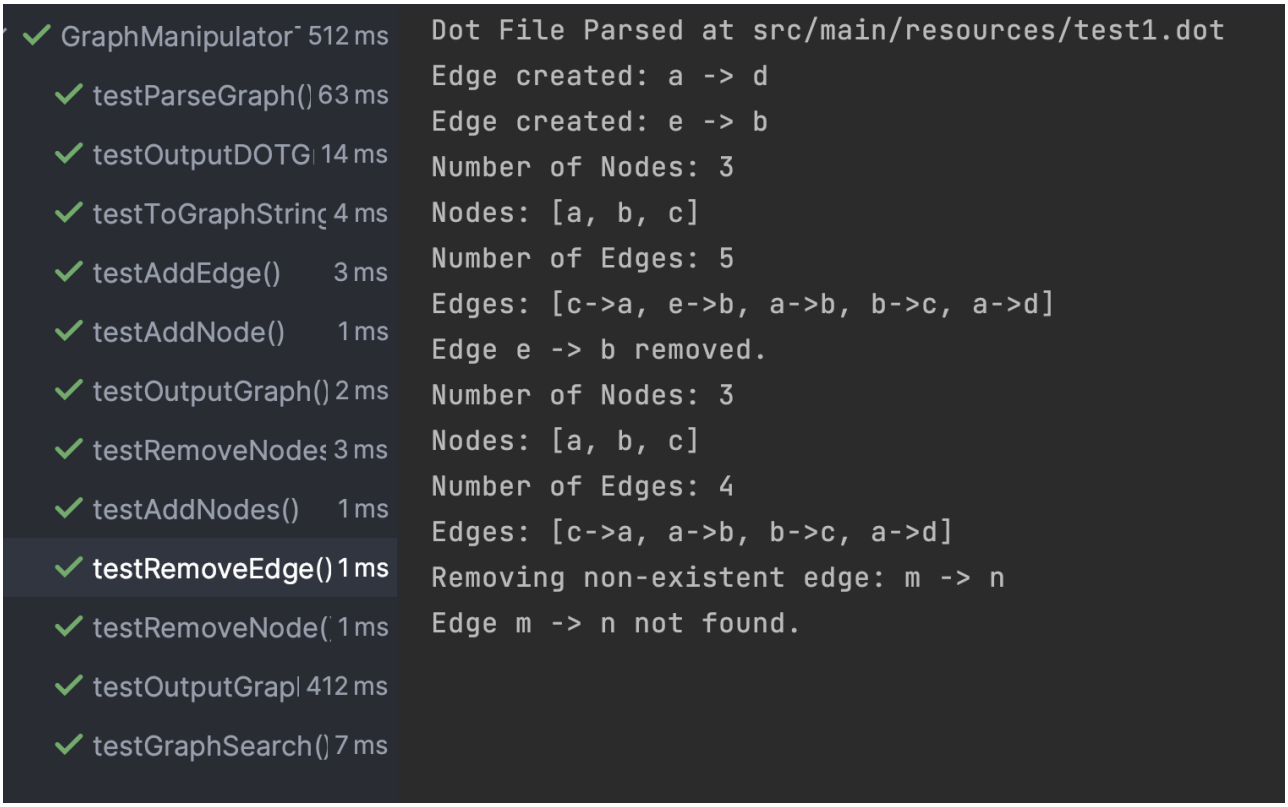
```
✓ GraphManipulator 522 ms
  ✓ testParseGraph() 65 ms
  ✓ testOutputDOTG 16 ms
  ✓ testToGraphString 6 ms
  ✓ testAddEdge() 1 ms
  ✓ testAddNode() 1 ms
  ✓ testOutputGraph() 1 ms
  ✓ testRemoveNodes 2 ms
  ✓ testAddNodes() 1 ms
  ✓ testRemoveEdge() 1 ms
  ✓ testRemoveNode( 1 ms
  ✓ testOutputGrap 420 ms
  ✓ testGraphSearch() 7 ms

Dot File Parsed at src/main/resources/test1.dot
Adding nodes e, f
Number of Nodes: 5
Nodes: [a, b, c, e, f]
Number of Edges: 3
Edges: [b -> c, c -> a, a -> b]
Removing nodes: e, f
Node e removed.
Node f removed.
Number of Nodes: 3
Nodes: [a, b, c]
Number of Edges: 3
Edges: [b -> c, c -> a, a -> b]
Removing non-existent nodes: x, y
Node x not found.
Node y not found.
```

7. **Remove Edge:**

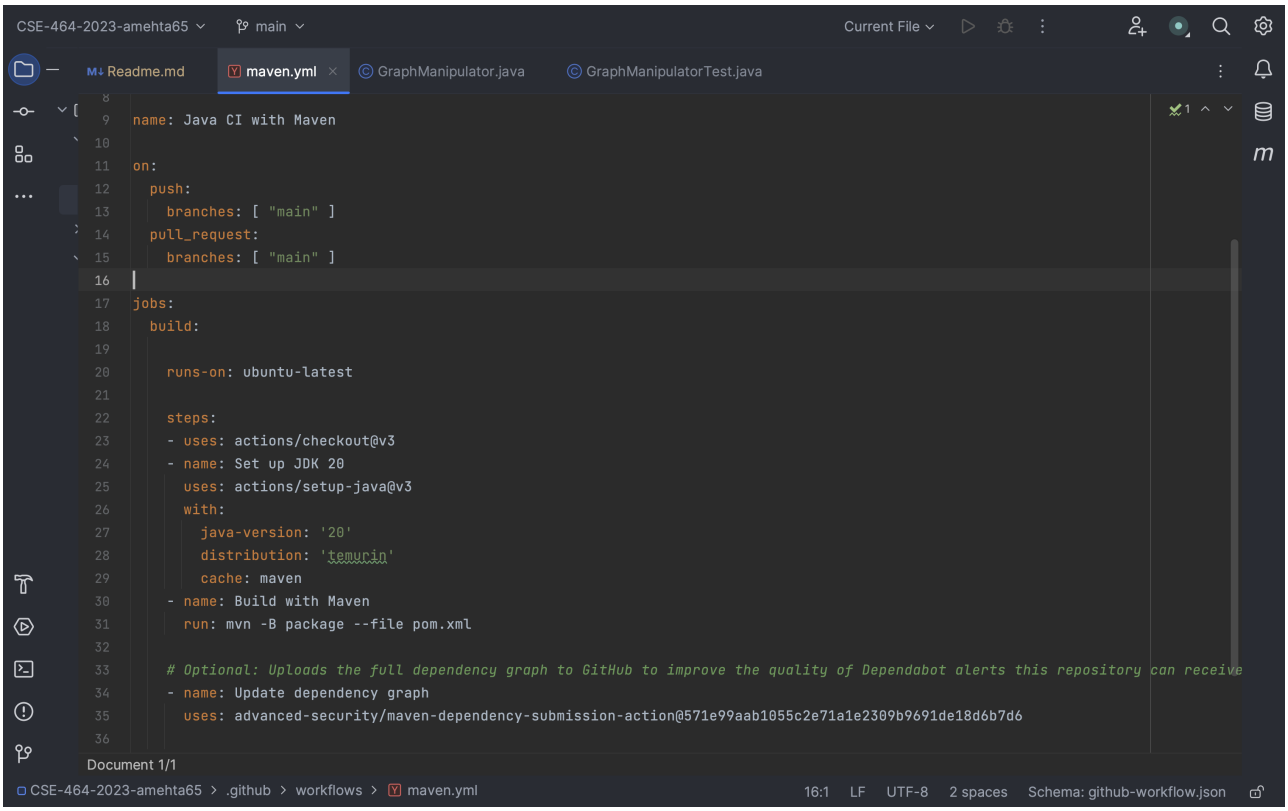
- To check remove edge functionality.

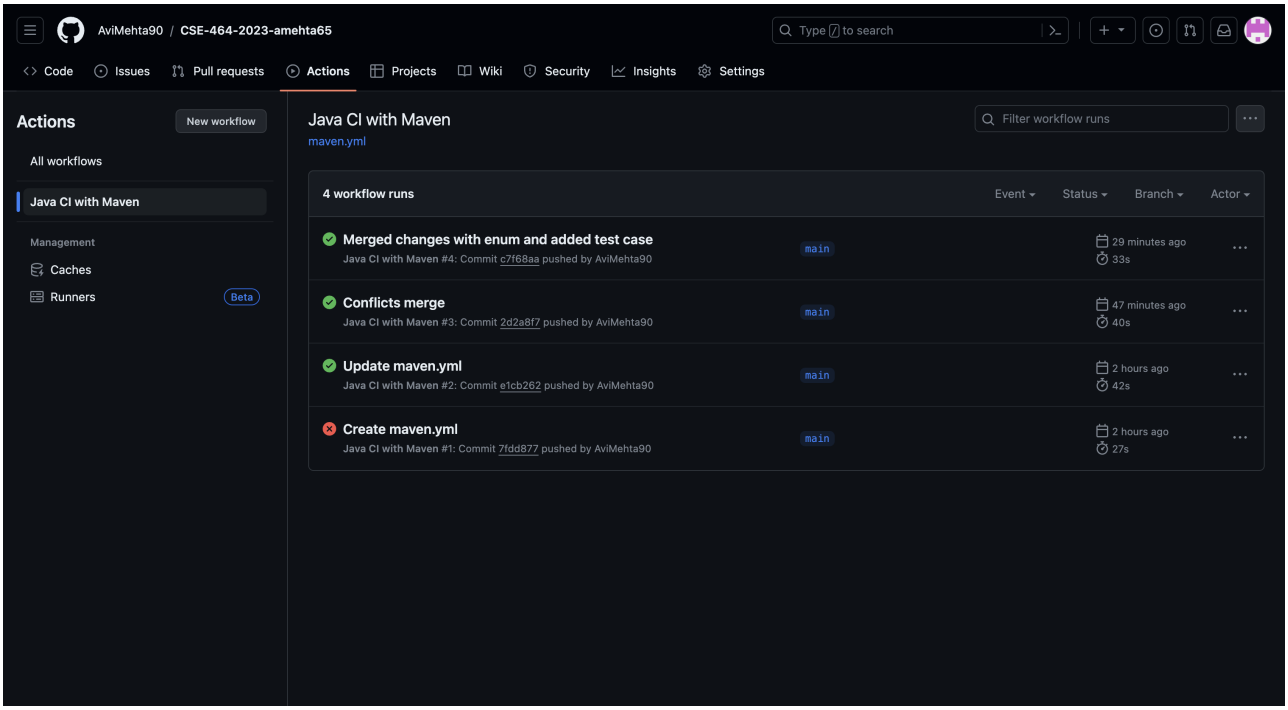
```java
GraphManipulator manipulator = new GraphManipulator();
manipulator.parseGraph("path/to/your/graph.dot");
manipulator.addNode("Node1");
manipulator.addEdge(String[]{"Node1", "Node2"});
manipulator.removeEdge(String[]{"Node1", "Node2"});
manipulator.removeEdge(String[]{"NonExNode2", "NonExNode3"});
```

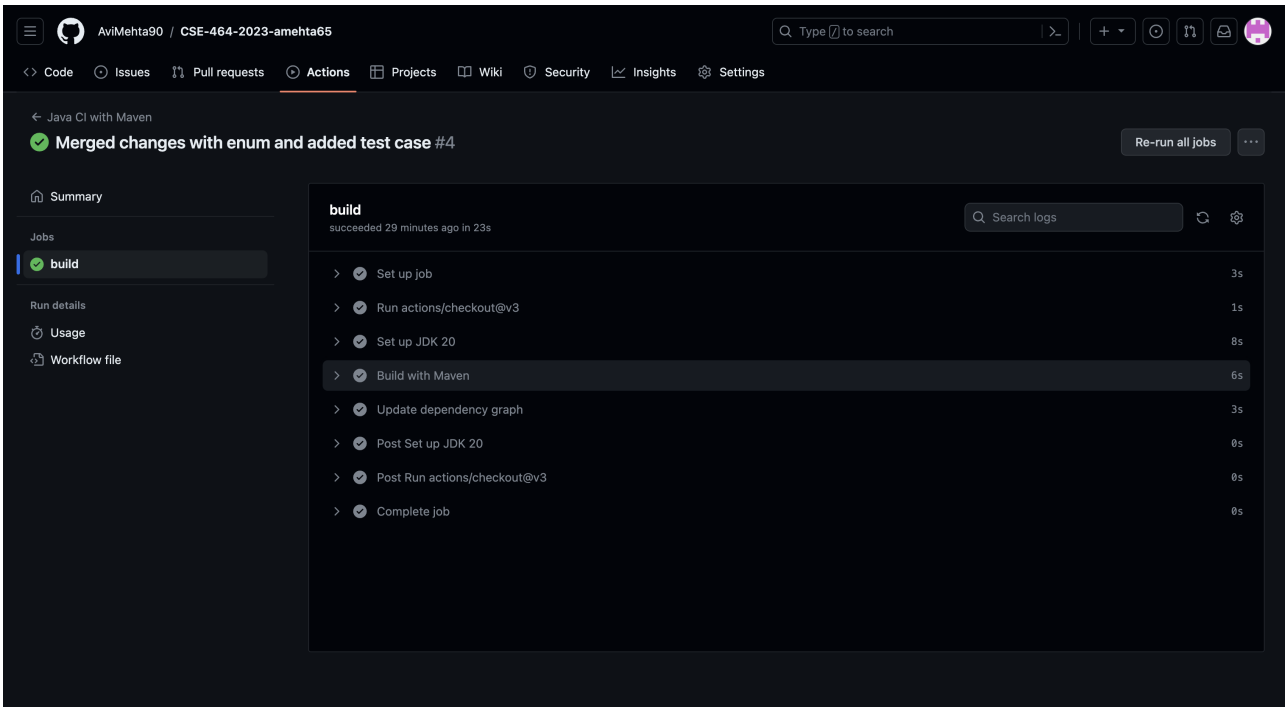
- **Console Output Remove Edge(Test Case)**



8. **Creating maven.yml for native CI/CD on github (under actions tab in the repository)**


CSE-464-2023-amehta65
main
Current File
Readme.md
maven.yml
GraphManipulator.java
GraphManipulatorTest.java
name: Java CI with Maven
on:
push:
branches: ["main"]
pull_request:
branches: ["main"]
jobs:
build:
runs-on: ubuntu-latest
steps:
- uses: actions/checkout@v3
- name: Set up JDK 20
uses: actions/setup-java@v3
with:
java-version: '20'
distribution: 'temurin'
cache: maven
- name: Build with Maven
run: mvn -B package --file pom.xml
Optional: Uploads the full dependency graph to GitHub to improve the quality of Dependabot alerts this repository can receive
- name: Update dependency graph
uses: advanced-security/maven-dependency-submission-action@571e99aab1055c2e71a1e2309b9691de18d6b7d6
Document 1/1
CSE-464-2023-amehta65 > .github > workflows > maven.yml
16:1
LF
UTF-8
2 spaces
Schema: github-workflow.json


AviMehta90 / CSE-464-2023-amehta65
Type / to search
Code
Issues
Pull requests
Actions
Projects
Wiki
Security
Insights
Settings
Actions
New workflow
All workflows
Java CI with Maven
Management
Caches
Runners
Beta
Java CI with Maven
maven.yml
Filter workflow runs
4 workflow runs
Event
Status
Branch
Actor
Merged changes with enum and added test case
Java CI with Maven #4: Commit c7f68aa pushed by AviMehta90
main
29 minutes ago
33s
Conflicts merge
Java CI with Maven #3: Commit 2d2a8f7 pushed by AviMehta90
main
47 minutes ago
40s
Update maven.yml
Java CI with Maven #2: Commit e1cb262 pushed by AviMehta90
main
2 hours ago
42s
Create maven.yml
Java CI with Maven #1: Commit 7fdd877 pushed by AviMehta90
main
2 hours ago
27s



9. **Creating branches and Merging Conflicts:**

- Creating bfs and dfs branch and merging with main branch

```
git checkout -b bfs
```

Do the changes in bfs branch and type the following command:

```
git add .
git commit -m "Added BFS Search Algorithm"
git push origin bfs
```

Go back to main branch

```
git checkout main
```

Create dfs branch

```
git checkout -b dfs
```

Do the changes in dfs branch and type the following command:

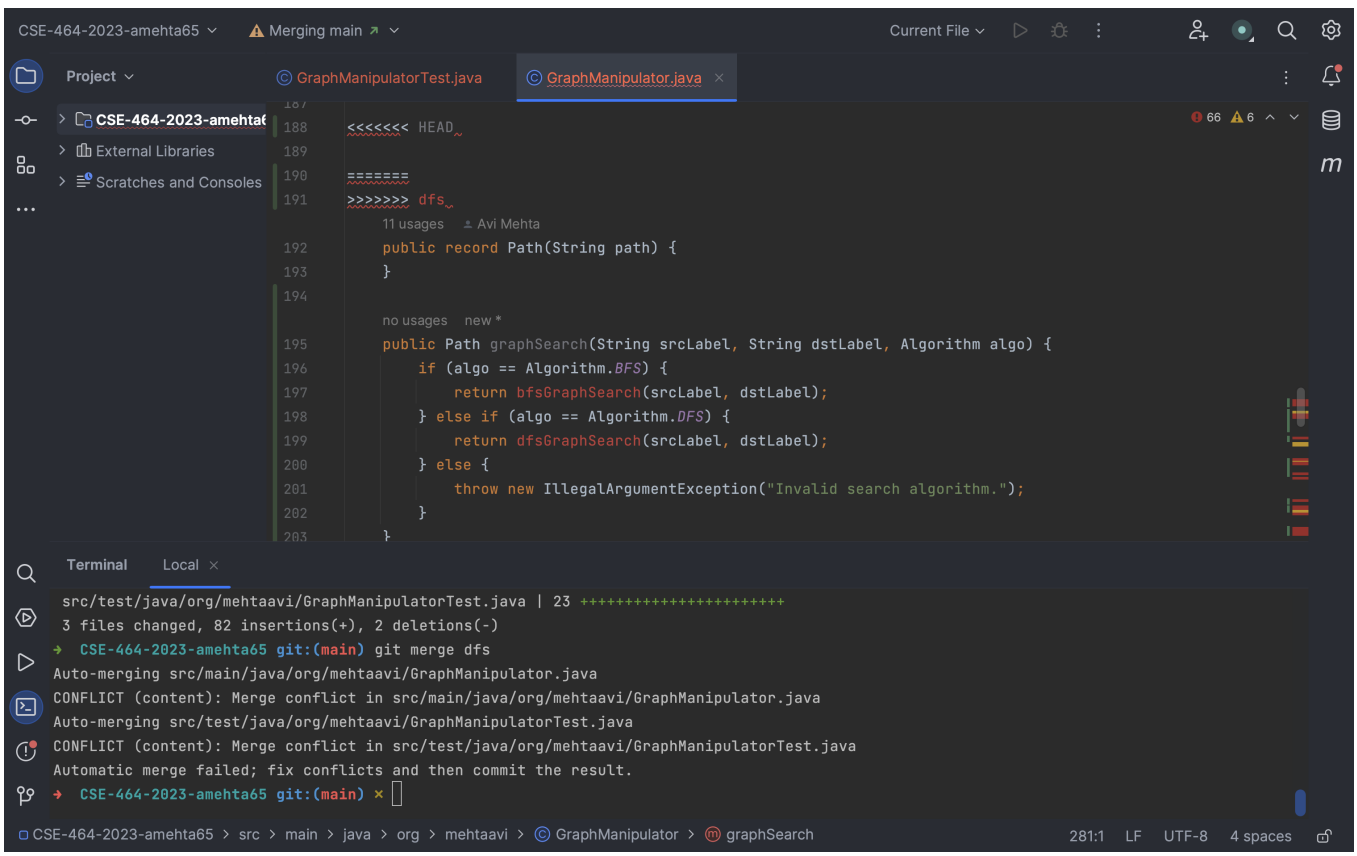
```
git add .
git commit -m "Added DFS Search Algorithm"
git push origin dfs
```

Now merge the dfs and bfs with main branch by checking in individual branches

```
git checkout main
git merge bfs
git merge dfs
```

Merge conflicts and then commit and push to main origin

```
git add .
git commit -m "Resolved merge conflicts and added algorithm selection in gra
```



10. **Graph Search(BFS):**

- Checking the BFS search algorithm.

```
GraphManipulator gM = new GraphManipulator();
```

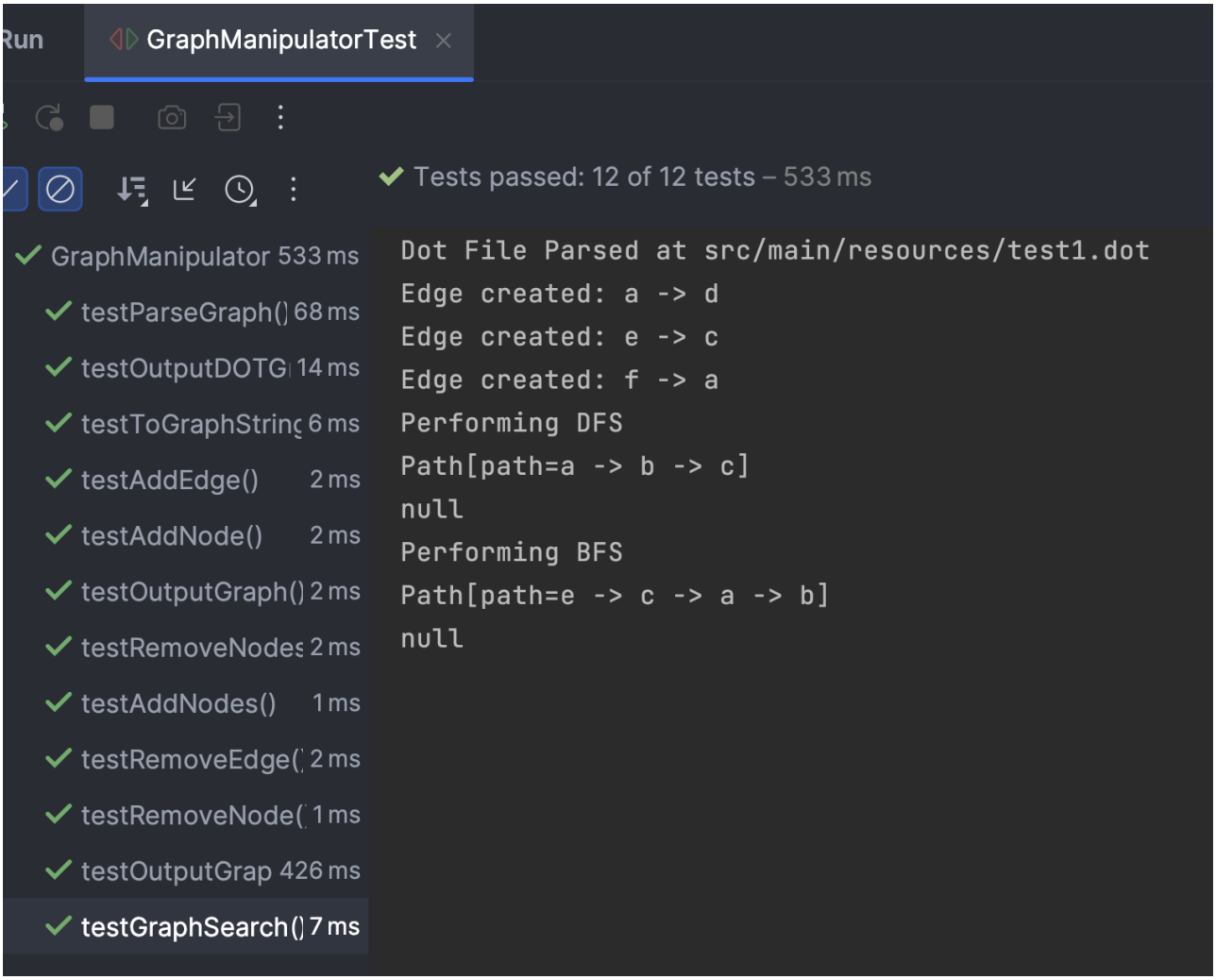
```
gM.addNode("d");
gM.addNodes(new String[]{"e", "f"});
gM.addEdge("a", "d");
gM.addEdge("e", "c");
gM.addEdge("f", "a");
GraphManipulator.Path path1 = gM.graphSearch("a", "c", GraphManipulator.
System.out.println("Performing BFS");
System.out.println(path1);
```

11. **Graph Search(DFS):**

- Checking the DFS search algorithm.

```
GraphManipulator gM = new GraphManipulator();
gM.addNode("d");
gM.addNodes(new String[]{"e", "f"});
gM.addEdge("a", "d");
gM.addEdge("e", "c");
gM.addEdge("f", "a");
GraphManipulator.Path path1 = gM.graphSearch("a", "c", GraphManipulator.
System.out.println("Performing DFS");
System.out.println(path1);
```

- **Console Output Graph Search(Test Case)**



Project Structure

The project's source code is organized as follows:

- `src/main/java/org/mehtaavi` : Contains the Java source code.
- `src/test/java/org/mehtaavi` : Contains the Java test 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
9. Added 3 APIs to support node and edge removal with test unit
10. Create maven.yml
11. Update maven.yml
12. Added BFS graph search API
13. Name corrected for BFS API graphsearch
14. Implemented DFS graph search API
15. Conflicts merge
16. Merged changes with enum and added test case
17. Formatting changes