- Github Link: https://github.com/DimitarAtanassov/CSE464

Part 1 Documentation

Dependencies Used (Imported VIA maven):

GraphViz
Slf4j-api
Junit

To import the dependencies add them to the pom.xml file like so:

```xml
<dependencies>
    <dependency>
        <groupId>guru.nidi</groupId>
        <artifactId>graphviz-java</artifactId>
        <version>0.18.1</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
        <version>2.0.6</version>
    </dependency>
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-simple</artifactId>
        <version>2.0.6</version>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.9.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.9.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>RELEASE</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

Features:

GraphParse(String filePath): This takes in a file path and creates a directed graph object

MAKE SURE TO PASS IN THE ABSOLUTE FILE PATH, if you do not it will throw an error if the file is not found in the root project folder
Using GraphParse(String filePath) with file in root project folder example:

```
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        String path = "input.dot";
        GraphParse graph;
        graph = new GraphParse( filePath: "input.dot");
    }
}
```

Using GraphParse with file NOT in root project folder example:

```
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        GraphParse graph;
        graph = new GraphParse( filePath: "C:\\Users\\Dimitar\\IdeaProjects\\CSE464P1\\src\\main\\java\\color.dot");
    }
}
```

String toString(): Returns the graph statistics as a string

```
System.out.println(graph.toString());
```

void outputGraph(String filePath): outputs the toString string to the specified file
MAKE SURE TO PASS IN THE ABSOLUTE FILE PATH, if you do not it will throw an error if the file is not found in the root project folder
Using outputGraph with file NOT in root project folder:

```
graph.outputGraph( filePath: "C:\\Users\\Dimitar\\IdeaProjects\\CSE464P1\\example\\outputgraphtest.txt");
```

void addNode(String label): adds a node to the graph with the specified label used as input, will not add the new node if the node already exists

```
graph.addNode( label: "x");
```

void removeNode(string label): removes the specified node if it exists, even if node does not exist it will still print that the node has been removed.

```
graph.removeNode( label: "x");
```

void addNodes(String[] labels): Takes in a string array of node labels and adds them to the graph, will not add duplicates.

```
String[] nodesToAdd = {"x","y","z"};
graph.addNodes(nodesToAdd);
```

void removeNodes(String[] labels): Takes in a string array of node labels and removes the nodes with those labels.

```
String[] nodesToRemove = {"x","y","z"};
graph.removeNodes(nodesToRemove);
```

void addEge(String srcLabel, String dstLabel): Takes in two node labels and creates an edge between them, if one of the nodes does not exist the node is created and then the edge is added

```
graph.addEdge( srcLabel: "a", dstLabel: "d");
```

void removeEdge(String srcLabel, String dstLabel): Takes in two node labels and removes the edge between them.

```
graph.removeEdge( srcLabel: "a", dstLabel: "d");
```

void outputDOTgraph(String filePath): Takes in the file path and outputs the DOT file of the graph, file is created if it does not exist.

```
graph.outputDOTGraph( path: "output1.dot");
```

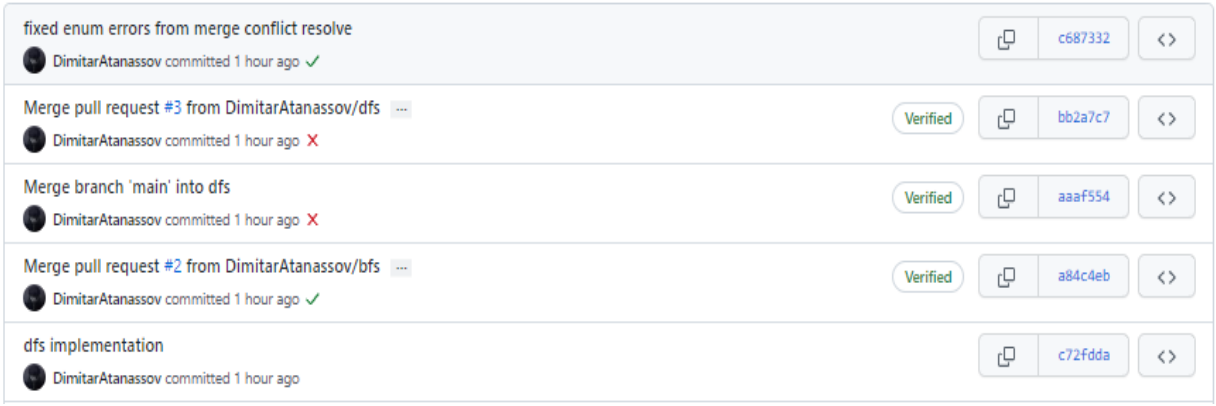
void outputGraphics(String filePath): Takes in the file path and outputs the visual created from the dot file.

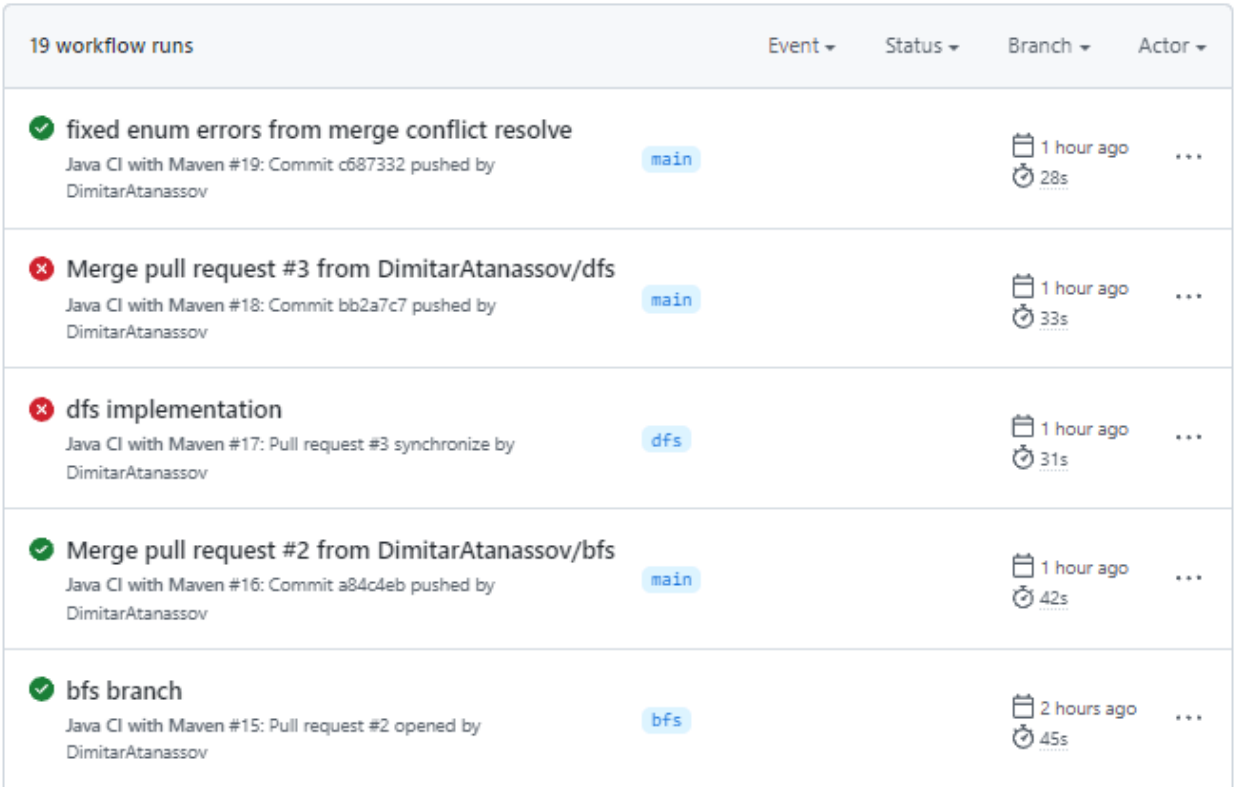
```
graph.outputGraphics( path: "output3.png");
```

Example of calling all features from main:

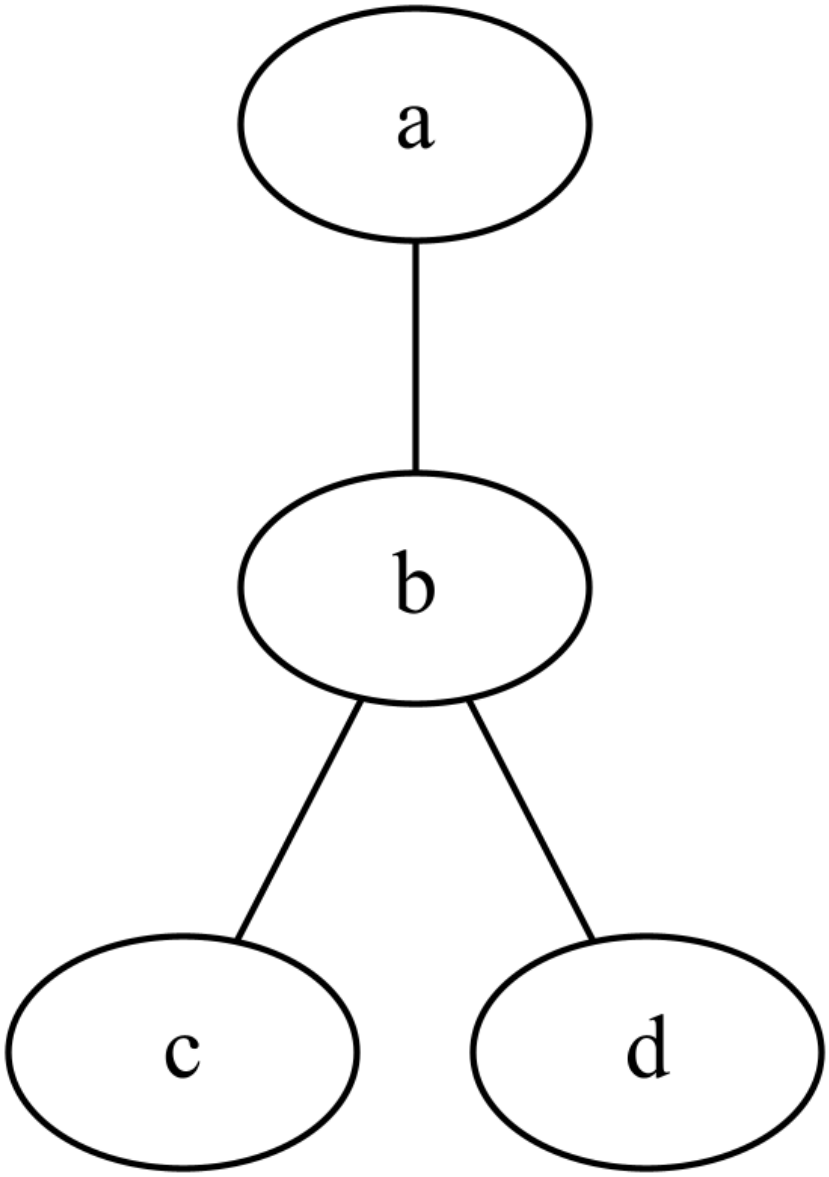
```java
import java.io.IOException;

no usages
public class Main {
    no usages
    public static void main(String[] args) throws IOException {
        GraphParse graph;
        graph = new GraphParse( filePath: "input.dot");
        System.out.println(graph.toString());
        graph.outputGraph( filePath: "outputgraphtest.txt");
        graph.addNode( label: "x");
        graph.removeNode( label: "x");
        String[] nodesToAdd = {"x","y","z"};
        graph.addNodes(nodesToAdd);
        String[] nodesToRemove = {"x","y","z"};
        graph.removeNodes(nodesToRemove);
        graph.addEdge( srcLabel: "a", dstLabel: "d");
        graph.removeEdge( srcLabel: "a", dstLabel: "d");
        graph.outputDOTGraph( path: "output1.dot");
        graph.outputGraphics( path: "output3.png");

    }
}
```

Part 2 Documentation

- Screenshot of commits
  - 

- Screenshot of workflow
  - 

- Screenshot of Input Used for BFS and DFS testing (filename: input.dot)



  -
  - Expected output BFS: `a->b->d`
  - Expected output DFS: `a->b->c->d`
- Steps to run BFS:
  - Need to create a Path object
  - Steps for using with User Input:
    - Get user input to choose between which algorithm to use for bfs the user must input **bfsAlgo**
    - Pass in graph dot file and user input for algorithm choice into path constructor
    - Call path.GraphSearch with source node and destination node labels (the function parameters are strings)

        - Example Snippet
        - ```
          Path path;
          Scanner myObj = new Scanner(System.in);
          System.out.println("Enter (bfsAlgo) for BFS traversal and Enter (dfsAlgo) for DFS traversal ");
          algoChoice userInput = algoChoice.valueOf(myObj.nextLine());
          path = new Path( filePath: "input.dot", userInput);
          path.GraphSearch( src: "a", dst: "d");
          ```
    - To use without user input
        - ```
          Path path;
          //Scanner myObj = new Scanner(System.in);
          //System.out.println("Enter (bfsAlgo) for BFS traversal and Enter (dfsAlgo) for DFS traversal ");
          // algoChoice userInput = algoChoice.valueOf(myObj.nextLine());
          path = new Path( filePath: "input.dot", algoChoice.bfsAlgo);
          path.GraphSearch( src: "a", dst: "d");
          ```
        - Create a path object
        - Pass in input file and algoChoice enum for bfs it would be algoChoice.bfsAlgo
        - Call path.GraphSearch with source node and destination node labels (the function parameters are strings)
- Steps to run DFS:
    - With user input
        - Get user input to choose between which algorithm to use for dfs the user must input **dfsAlgo**
        - Pass in graph dot file and user input for algorithm choice into path constructor
        - Call path.GraphSearch with source node and destination node labels (the function parameters are strings)
        - Example Snippet
        - ```
          Path path;
          Scanner myObj = new Scanner(System.in);
          System.out.println("Enter (bfsAlgo) for BFS traversal and Enter (dfsAlgo) for DFS traversal ");
          algoChoice userInput = algoChoice.valueOf(myObj.nextLine());
          path = new Path( filePath: "input.dot", userInput);
          path.GraphSearch( src: "a", dst: "d");
          ```
    - Without user input
        - Create a path object
        - Pass in input file and algoChoice enum for dfs it would be algoChoice.dfsAlgo
        - Call path.GraphSearch with source node and destination node labels (the function parameters are strings)
        - ```
          Path path;
          //Scanner myObj = new Scanner(System.in);
          //System.out.println("Enter (bfsAlgo) for BFS traversal and Enter (dfsAlgo) for DFS traversal ");
          // algoChoice userInput = algoChoice.valueOf(myObj.nextLine());
          path = new Path( filePath: "input.dot", algoChoice.dfsAlgo);
          path.GraphSearch( src: "a", dst: "d");
          ```

- Links
  - CI Link: https://github.com/DimitarAtanassov/CSE464/actions/workflows/maven.yml
  - Commits Link: https://github.com/DimitarAtanassov/CSE464/commits/main
    - Commit link for bfs: https://github.com/DimitarAtanassov/CSE464/commit/f3520ecb7c7b444a316cc698f4632a9461cfe65c
    - Commit link for dfs: https://github.com/DimitarAtanassov/CSE464/commit/c72fdda122fb4cb0b1d36c26bffd6066fb33fee1
    - Commit link for merge bfs: https://github.com/DimitarAtanassov/CSE464/commit/a84c4eb4715b108d851b080d8b05b52bbe220328
    - Commit link for merge dfs: https://github.com/DimitarAtanassov/CSE464/commit/aaaf5547d210f8da0f138a7bd1877976d66c0998
    - Commit link for merge conflict resolve: https://github.com/DimitarAtanassov/CSE464/commit/bb2a7c7c39caf47949bf3952df44290904fc94c3
    - Commit link for enum eros resolve:https://github.com/DimitarAtanassov/CSE464/commit/c6873321468f951f6bad5fc33ee879c255c1de3e
  - Main Branch: https://github.com/DimitarAtanassov/CSE464
  - BFS Branch:https://github.com/DimitarAtanassov/CSE464/tree/bfs
  - DFS Branch:https://github.com/DimitarAtanassov/CSE464/tree/dfs
  -