Readme

Instructions

- Clone the repository, or download the zip file and unzip it.
- Run the command `mvn package`.
- This will build the project, create a `JAR` file in the `targets/` folder and run all tests.
- Opening the project folder in IntelliJ as a project will also enable running the tests and inspecting the code.

API Functions

The API is contained in the `GraphData.java` file at `src/main/java/` location. The various functions implemented are listed below:

- `boolean parseGraph(String filepath)` : Import DOT file to create a JGrapht graph object. Returns `true` if successful else `false`.
- `String toString()` : Display graph information such as node and edge number in string format. Returns a `String`.
- `boolean outputGraph(String filepath)` : Writes graph details to a file at `filepath`. Returns `true` if successful else `false`.
- `boolean addNode(String label)` : Adds a node to the graph. Returns `true` if successful else `false`.
- `boolean addNodes(String[] labels)` : Adds a list of nodes to the graph. Returns `true` if successful else `false`.
- `boolean addEdge(String srcLabel, String dstLabel)` : Adds an edge to the graph. Returns `true` if successful else `false`.
- `boolean outputDOTGraph(String path)` : Outputs the `JGraphT` graph object to a `DOT` file at the specified `path`. Returns `true` if successful else `false`.
- `void outputGraphics(String path, String format)` : Outputs the `JGraphT` graph object to a file with file format `format` at the specified `path`.

How to use (Example code)

- GraphData object creation

```
GraphData graphApi = new GraphData();
```

- Parse a graph

```
graphApi.parseGraph("src/main/resources/example.dot");
```

Expected Output:

```
Graph successfully parsed!
```

- Display graph data

```
graphApi.toString();
```

Expected Output:

```
Number of nodes: 4
Node labels: [A, B, C, D]
Number of edges: 3
Node and edge directions: (A -> B), (A -> C), (A -> D)
```

- Output graph data to file

```
graphApi.outputGraph("src/main/resources/output.txt");
```

Expected Output (in `output.txt`):

```
Number of nodes: 4
Node labels: [A, B, C, D]
Number of edges: 3
Node and edge directions: (A -> B), (A -> C), (A -> D)
```

- Add nodes

```
graphApi.addNode("Y");
graphApi.addNodes(new String[]{"Z", "X"});
```

- Add edge

```
graphApi.addEdge("Z", "C");
```

- Output graph in DOT file format

```
graphApi.outputDOTGraph("src/main/resources/gen_graph.dot");
```

Expected Output (in `gen_graph.dot`):

```
strict digraph G {
A;
B;
C;
D;
Y;
Z;
X;
A -> B;
A -> C;
A -> D;
Z -> C;
}
```

- Output graph as PNG file

```
graphApi.outputGraphics("src/main/resources/", "png");
```

- GraphSearch API

```
Path path = graphApi.GraphSearch("C","D", Algorithm.BFS);
// Algorithm.DFS can also be used.
path.printPath();
```

Expected Output

```
Using BFS
C->A->D
```

Commits

main

- Initial commit
- Add maven pom file.
- Update pom file.
- Implement first feature.
- Implement second feature.
- Implement third feature.
- Implement fourth feature.
- Handle IO exceptions.
- Add tests
- Update pom file.
- Update readme.
- Create maven.yml for buld automation.
- Update maven.yml to Java 21
- Add features to remove nodes.
- Add exception throws to functions.
- Add features to remove edges.
- Update maven.yml
- Add Path class.
- Add GraphSearch API with BFS algorithm.
- Add bfs GraphSearch tests.
- Add GraphSearch API with DFS algorithm.
- Add dfs GraphSearch tests.
- Merge commit.

bfs

- Add GraphSearch API with BFS algorithm.
- Add bfs GraphSearch tests.

dfs

- Add GraphSearch API with DFS algorithm.
- Add dfs GraphSearch tests.