README for CSE 464 Project Part 1

Shantanu Shishodia 1225590054

GitHub Repository link: https://github.com/shantanushishodia/cse-464-2023-sshishod

Instructions to Run

- Download the cse-464-sshishod.zip file from this repository
- Run mvn package
- This should run all tests for the project
- This command will build the project in the target folder as well
- Alternatively, unzip cse-464-sshishod.zip and then open the GraphHandler folder in IntelliJ

APIs

- void graphImporter(String filePath) import a directed graph in a dot file
- String toString() Graph information like number of nodes, edges and their directions
- void saveGraphToFile(String filePath) Write the graph information to a file
- void addOneNode(String label) Adds a new node to the graph with the given label if it does not exist

- void addMultipleNodes(ArrayList<String> labels) Add multiple nodes to the graph
- boolean addEdge(String initialNode, String targetNode) Returns true if edge is added otherwise returns false if edge exists
- void saveGraphDOT(String filePath) Outputs the modified graph in DOT format to the specified file
- void saveGraphPNG(String filePath) Output the modified graph to a PNG file (Graph Visualization)
- Path GraphSearch(String src, String dst, Algorithm algo) Find a path from src to dst node using BFS or DFS or Random Walk algorithm depending on enum specified.
 Possible values of the enum can be Algorithm.BFS, Algorithm.DFS or Algorithm.RANDOMWALK

1. Refactors for the project part 3:

 Refactor 1: Encapsulated pathMap and added getPathMap function. This makes the pathMap variable prive. The getPathMap function acts as a getter for it. LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/2ac900198 f9723d68b825cb6196b51e70de0a3ea

2. Refactor 2: renamed functions for better understanding of the code LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/723f8a049 a2cdcd7e088faec774f4a7fae1d43d6

3. Refactor 3: improved consistency of exception handling and removed redundancy of performing the same task:

LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/0005bf761 098dcb98dbdecbd2e9b39133e788d9e

4. Refactor 4: Streamlined graphSearchWithAlgo. Now uses switch case instead of if-else statements.

LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/b582226fc 82054af3234cb1bbd11d6f2a9c19b5b

5. Refactor 5: Used constants for file path for more maintainable code LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/07da7b1ecbc4ef1ea0d1c9e42778389c846bedf9

6. Refactor 6: Extracted logic in helper methods LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/a2ddc0ee7e5848a14b66101ed15f8a47dae355c7

2. Template design pattern for BFS and DFS:

a. The GraphSearchTemplate class implements the template pattern. This class acts as a template for graph search algorithms, providing the general framework for the search procedure while letting subclasses handle the actual search implementation.

The template method "findPath" in the template class is responsible for organizing the entire network search procedure. Common actions like verifying the source and destination and looking for nodes in the graph are included. Additionally, the abstract method (search), which is specified as abstract and intended to be implemented by concrete subclasses, receives the basic search logic from the template method.

LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/84a3895528471b 9d74252d25c4394be30d02ca8d

This commit also contains minor changes for better working of the code like converting any input in selecting the search algo to uppercase, etc.

b. A code snippet showing the use of template pattern:

```
Graph<String, DefaultEdge> currGraph = gh.getGraph();
BFS bfs = new BFS();
Path result = bfs.findPath(currGraph, "Google", "Tesla");
```

3. Strategy design pattern for BFS and DFS:

a First I created an interface "GrpahSearchStrategy" and a class "GraphSearchContext".

The interface serves as a common interface for all graph search algorithms, declaring a method (findPath) that encapsulates the algorithm's logic. Concrete classes implementing this interface, such as BFS and DFS, provide their specific implementations of the graph search algorithm.

The context class accepts a strategy (an object implementing the GraphSearchStrategy interface) and delegates the graph search task to this strategy.

LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/f91b776c9699fec 1048301780d3584802e4b8731

b A code snippet showing the use of strategy pattern:

```
GraphHandler gh = new GraphHandler();
gh.graphImporterFromDot("src/test/test/1.dot");
Path result = gh.graphSearchWithAlgo("Google", "Tesla", Main.Algorithm.BFS);
```

4. Tests for BFS and DFS using both patterns:

a I created different tests for BFS and DFS using both strategy and template patterns. You can see code snippets for both in the pattern's implementation section (2 and 3).

LINK:

 $\frac{https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/f1f4375f79effd205}{dc1a7ce797e0dc408e24428}$

5. Implementation for Random Walk Search Algo:

a Created a new for the search algo "RandomWalkSearch". The RandomWalkSearch class extends the "GraphSearchTemplate" and implements "GraphSearchStrategy".

The algorithm starts at a given source node and continues traversing the graph until it reaches the specified destination node. During each step, the algorithm selects a random neighbor of the current node, moves to that neighbor, and updates the traversal path accordingly. The process continues until the destination node is reached or there are no unvisited neighbors left for the current node.

LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/f1f4375f79effd205dc1a7ce797e0dc408e24428

b Code Snippet:

```
while (!neighbors.isEmpty()) {
   int randomIndex = new Random().nextInt(neighbors.size());

String randomNeighbor = neighbors.get(randomIndex);
   System.out.println(current+ "->"+randomNeighbor);

path.getPathMap().put(randomNeighbor, current);

boolean temp =search(graph, randomNeighbor, destination, visited);
   if(temp)
      return true;
   neighbors.remove(randomIndex);
}
```

- c I used the DOT file provided by the professor to use as the graph file for Random Walk search. The program automatically loads the DOT file whenever the user selects RANDOMWALK as the search algo.
- d Example output:

```
Choose Algo BFS/DFS/RandomWalk:

randomwalk

Graph Parsing Successful

Input source node

Input destination node

a -> b

b->c

a -> b -> c
```

```
Choose Algo BFS/DFS/RandomWalk:

randomwalk

Graph Parsing Successful

Input source node

Input destination node

a->e
e->f
f->h
e->g
g->h
a->b
b->c
a -> b -> c
```

- e The algorithm tries random paths and displays the path it's taking. When it finds the desired path, it displays it at last.
- f Didn't add any test cases as the part 3 doc did not ask to. Also since it is a random search, it did not make sense to do it.

6. Code Review

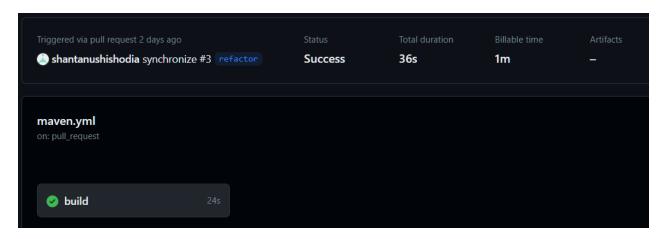
a After pushing my commits to the refactor branch, I raised a pull request for merging the refactor to master branch.

LINK: https://github.com/shantanushishodia/cse-464-2023-sshishod/pull/3

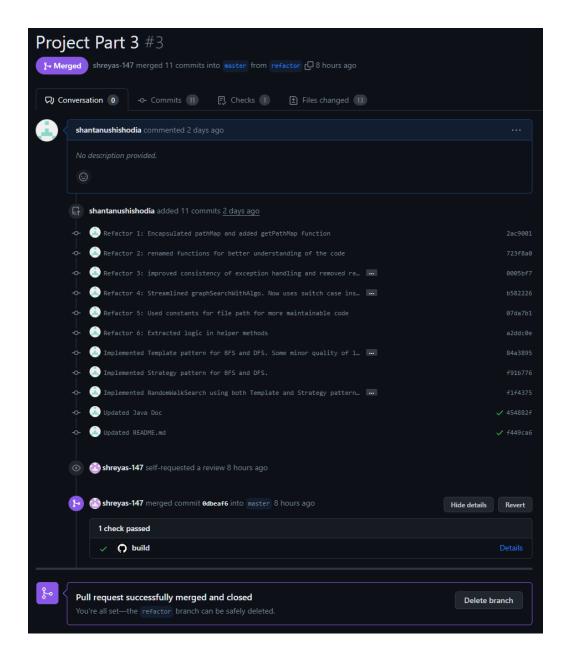
b Verified the working of CI. Tests passed and there were no conflicts. LINK:

https://github.com/shantanushishodia/cse-464-2023-sshishod/actions/runs/6985883549/job/19010700636

c Screenshot for the workflow build success:



d The PR was merged successfully. [Reviewed by: Shreyas Kolte (shreyas-147)] LINK: https://github.com/shantanushishodia/cse-464-2023-sshishod/pull/3



7. Appendix

a Final output after running the program using "mvn package" in IntelliJ:

[INFO] -----[INFO] TESTS [INFO] -----[INFO] Running GraphHandlerTest **Graph Parsing Successful** output: src/outputDOTFile.dot **Graph Parsing Successful** Nodes Count: 6 Label of nodes: Google Meta Ford Tesla **NXP** Asus Edges count: 6 Directional edges with nodes: Google -> Meta Meta -> Ford Tesla -> NXP NXP -> Asus Ford -> Tesla Google -> Ford Edge already present in the graph **Graph Parsing Successful** File save is a success src/testSaveGraphFile.txt Graph Parsing Successful **Graph Parsing Successful** Google -> Meta -> Ford -> Tesla No Path found **Graph Parsing Successful Graph Parsing Successful** Google -> Meta -> Ford -> Tesla No Path found **Graph Parsing Successful Graph Parsing Successful Graph Parsing Successful** Nodes Count: 6 Label of nodes: Google

Meta Ford Tesla **NXP** Asus Edges count: 5 Directional edges with nodes: Google -> Meta Meta -> Ford Tesla -> NXP NXP -> Asus Ford -> Tesla **Graph Parsing Successful Graph Parsing Successful Graph Parsing Successful** Google -> Meta -> Ford -> Tesla No Path found Graph Parsing Successful Nodes Count: 7 Label of nodes: Google Meta Ford Tesla NXP Asus NASA Edges count: 5 Directional edges with nodes: Google -> Meta Meta -> Ford Tesla -> NXP NXP -> Asus Ford -> Tesla **Graph Parsing Successful** Nodes Count: 6 Label of nodes: Google Meta Ford Tesla **NXP** Asus Edges count: 4

```
Directional edges with nodes:
Google -> Meta
Tesla -> NXP
NXP -> Asus
Ford -> Tesla
Graph Parsing Successful
Nodes Count: 5
Label of nodes:
Meta
Ford
Tesla
NXP
Asus
Edges count: 4
Directional edges with nodes:
Meta -> Ford
Tesla -> NXP
NXP -> Asus
Ford -> Tesla
Graph Parsing Successful
Graph Parsing Successful
Google -> Meta -> Ford -> Tesla
No Path found
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.248 s -
in GraphHandlerTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ SS P1 ---
[INFO] Building jar:
C:\Users\shant\IdeaProjects\SS P1\target\SS P1-1.0-SNAPSHOT.jar
[INFO] ------
IINFOI BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.493 s
[INFO] Finished at: 2023-11-25T01:46:49-07:00
[INFO] -----
Process finished with exit code 0
```