

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:

1. Refactor 1: Encapsulated pathMap and added getPathMap function. This makes the pathMap variable private. The getPathMap function acts as a getter for it.
LINK:
<https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/2ac900198f9723d68b825cb6196b51e70de0a3ea>
2. Refactor 2: renamed functions for better understanding of the code
LINK:
<https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/723f8a049a2cdcd7e088faec774f4a7fae1d43d6>
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/0005bf761098dcb98dbdecdb2e9b39133e788d9e>
4. Refactor 4: Streamlined graphSearchWithAlgo. Now uses switch case instead of if-else statements.
LINK:
<https://github.com/shantanushishodia/cse-464-2023-sshishod/commit/b582226fc82054af3234cb1bbd11d6f2a9c19b5b>

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/84a3895528471b9d74252d25c4394be30d02ca8d>

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/f91b776c9699fec1048301780d3584802e4b8731>

- b A code snippet showing the use of strategy pattern:

```
GraphHandler gh = new GraphHandler();  
gh.graphImporterFromDot("src/test/test1.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:

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

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
a
Input destination node
c
a->b
b->c
a -> b -> c
```

```
Choose Algo BFS/DFS/RandomWalk:
randomwalk
Graph Parsing Successful
Input source node
a
Input destination node
c
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:

This screenshot shows a GitHub Actions workflow run for the file `maven.yml` triggered by a pull request. The workflow is named `synchronize #3` and is on the `refactor` branch. The status is **Success**, with a total duration of **36s** and a billable time of **1m**. There are no artifacts. A single job named `build` is shown with a green checkmark and a duration of **24s**.

Triggered via	Status	Total duration	Billable time	Artifacts
pull request 2 days ago	Success	36s	1m	—

maven.yml
on: pull_request

✓ **build** 24s

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

This screenshot shows a GitHub pull request for "Project Part 3 #3". The pull request is merged, with 11 commits merged into the `master` branch from the `refactor` branch, 8 hours ago. The pull request is reviewed by shreyas-147. The commit history shows 11 commits, including 6 refactor commits, 3 implementation commits, and 2 documentation updates. The pull request is closed, and the `refactor` branch can be safely deleted.

Project Part 3 #3

Merged shreyas-147 merged 11 commits into `master` from `refactor` 8 hours ago

Conversation 0 Commits 11 Checks 1 Files changed 13

shantanushishodia commented 2 days ago

No description provided.

shantanushishodia added 11 commits 2 days ago

- Refactor 1: Encapsulated pathMap and added getPathMap function 2ac9001
- Refactor 2: renamed functions for better understanding of the code 723f8a0
- Refactor 3: improved consistency of exception handling and removed re... 0005bf7
- Refactor 4: Streamlined graphSearchWithAlgo. Now uses switch case ins... b582226
- Refactor 5: Used constants for file path for more maintainable code 07da7b1
- Refactor 6: Extracted logic in helper methods a2ddc0e
- Implemented Template pattern for BFS and DFS. Some minor quality of l... 84a3895
- Implemented Strategy pattern for BFS and DFS. f91b776
- Implemented RandomWalkSearch using both Template and Strategy pattern... f1f4375
- Updated Java Doc ✓ 454882f
- Updated README.md ✓ f449ca6

shreyas-147 self-requested a review 8 hours ago

shreyas-147 merged commit `0dbeeaf6` into `master` 8 hours ago

1 check passed

✓ build Details

Pull request successfully merged and closed

You're all set—the `refactor` branch can be safely deleted.

Delete branch

7. Appendix

- a Final output after running the program using “mvn package” in IntelliJ:

```
[INFO] -----  
[INFO] T E S T S  
[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] -----

[INFO] 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