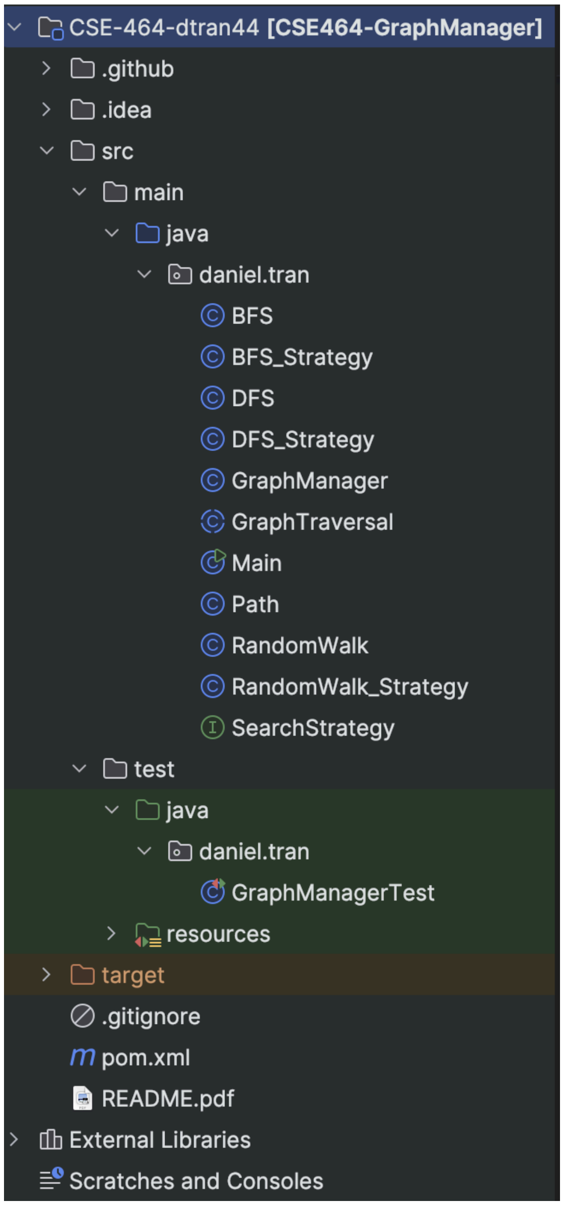
#README

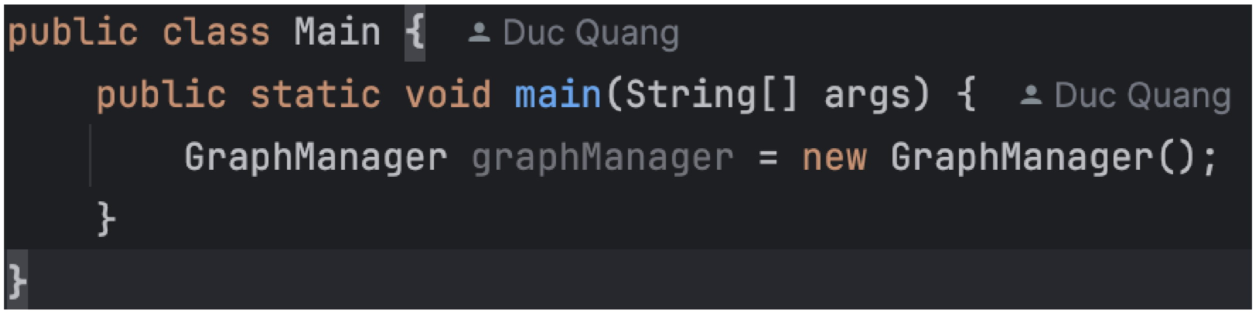
Project File Structure



```
CSE-464-dtran44 [CSE464-GraphManager]
  .github
  .idea
  src
    main
      java
        daniel.tran
          BFS
          BFS_Strategy
          DFS
          DFS_Strategy
          GraphManager
          GraphTraversal
          Main
          Path
          RandomWalk
          RandomWalk_Strategy
          SearchStrategy
    test
      java
        daniel.tran
          GraphManagerTest
      resources
  target
  .gitignore
  pom.xml
  README.pdf
External Libraries
Scratches and Consoles
```

Above is a screenshot of project's file structure should you open it in IntelliJ IDEA. You can see that the files have been placed in their intended structure where the source code will be in src -> main -> java -> daniel.tran whereas the test file in src -> test -> java -> daniel.tran along with the resources.

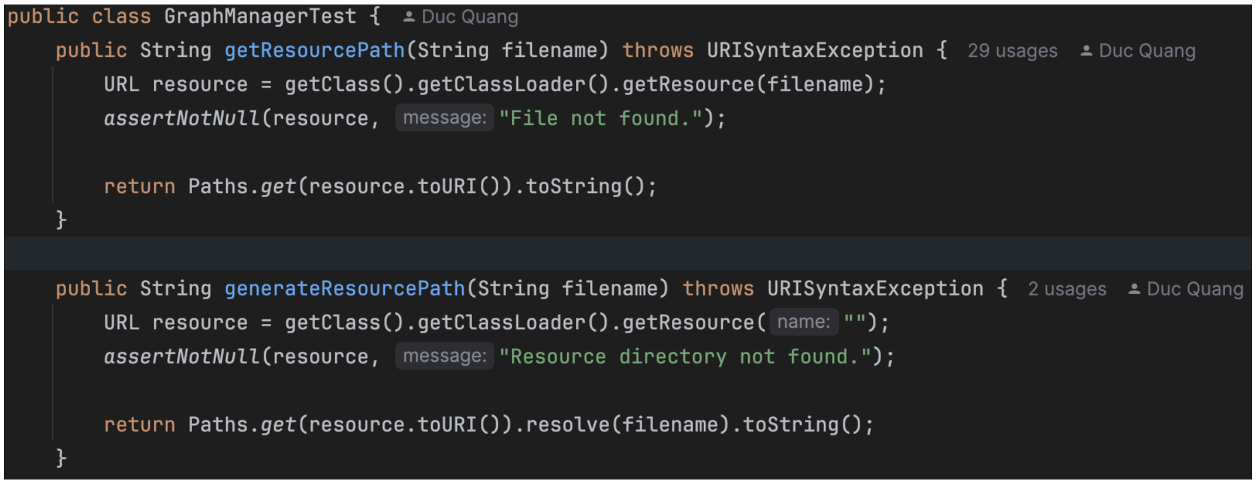
Main File



```java
public class Main {  Duc Quang
    public static void main(String[] args) {  Duc Quang
        GraphManager graphManager = new GraphManager();
    }
}
```

Above is a screenshot of the main file where you can test out all kind of APIs implemented. I have created a new GraphManager object above where you can use the functions by calling its variable name “graphManager” and then API/function name.

Test File



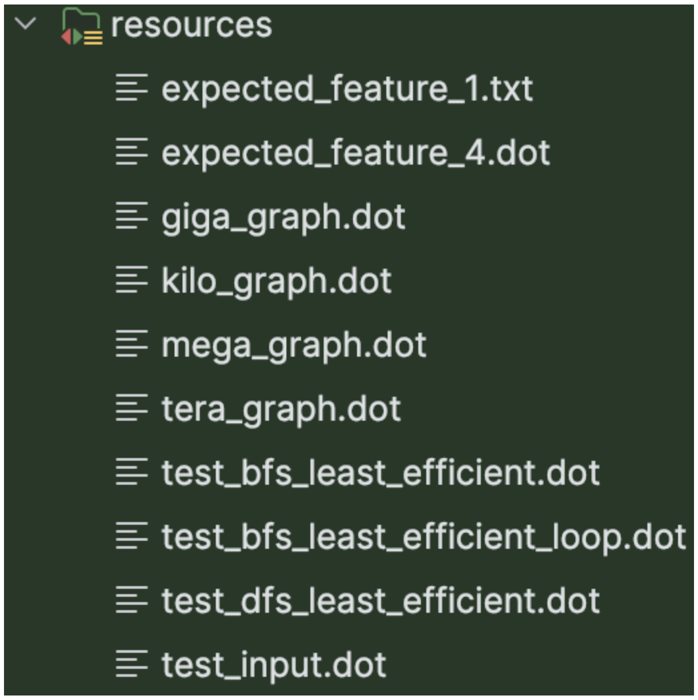
```java
public class GraphManagerTest {  Duc Quang
    public String getResourcePath(String filename) throws URISyntaxException {  29 usages  Duc Quang
        URL resource = getClass().getClassLoader().getResource(filename);
        assertNotNull(resource,  message: "File not found.");

        return Paths.get(resource.toURI()).toString();
    }

    public String generateResourcePath(String filename) throws URISyntaxException {  2 usages  Duc Quang
        URL resource = getClass().getClassLoader().getResource( name: "");
        assertNotNull(resource,  message: "Resource directory not found.");

        return Paths.get(resource.toURI()).resolve(filename).toString();
    }
```

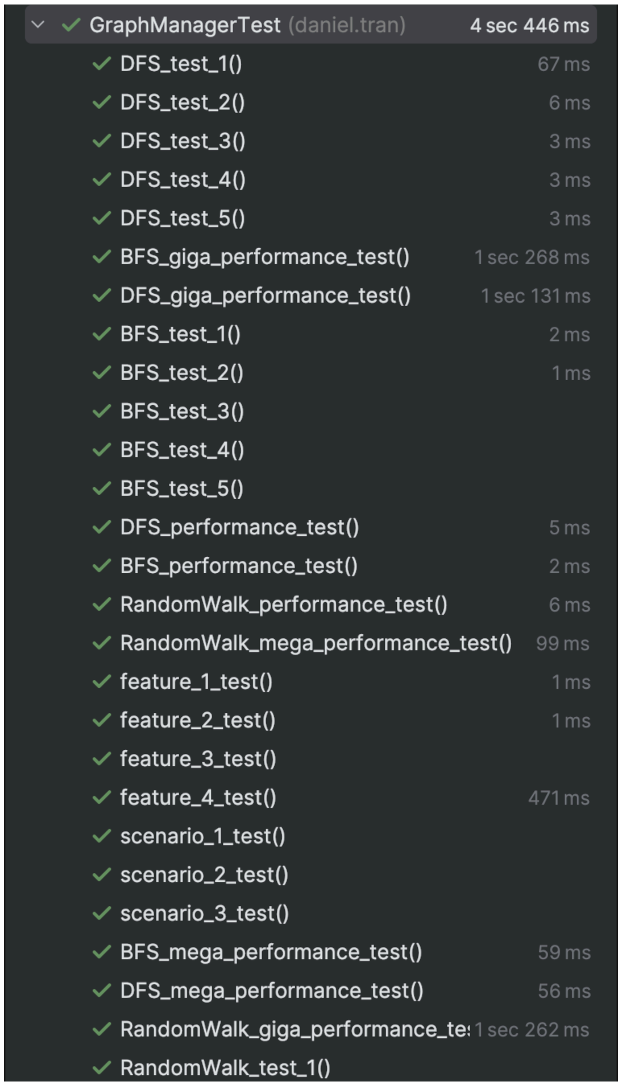
Above is a screenshot of part of the test file. With the new structure, any test inputs as file should be stored in src -> test -> resources



```
resources
    expected_feature_1.txt
    expected_feature_4.dot
    giga_graph.dot
    kilo_graph.dot
    mega_graph.dot
    tera_graph.dot
    test_bfs_least_efficient.dot
    test_bfs_least_efficient_loop.dot
    test_dfs_least_efficient.dot
    test_input.dot
```

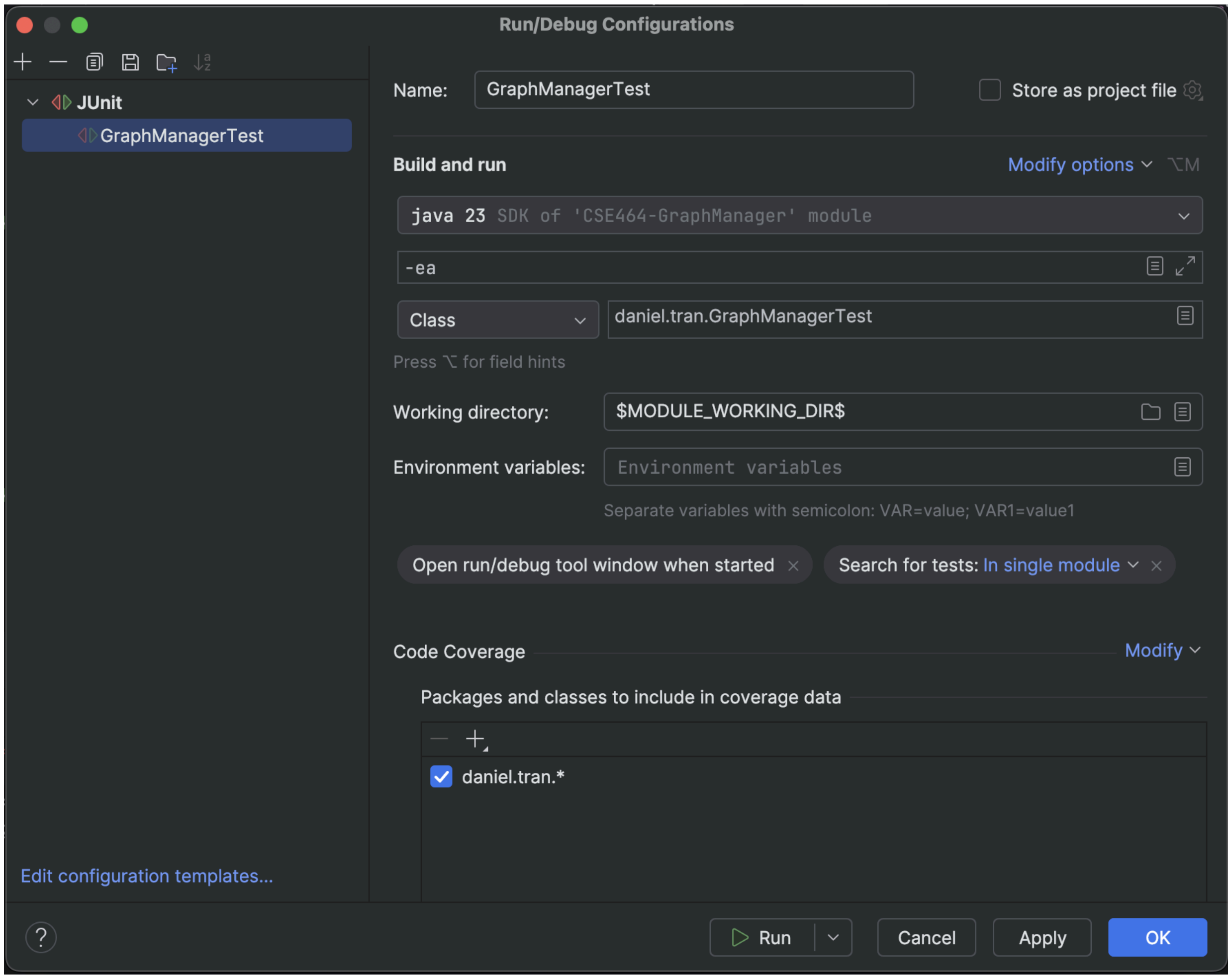
As seen in the first screenshot of the page, the test file now has 2 new functions to get the path of test inputs from the "resources" folder. The first one will throw exception if file doesn't exist, the second function will simply return the path to "resources" folder without checking for the file's existence. The first one is suitable for loading files in tests and the second one is more suitable for getting the path to "resources" folder to save a file (usage can be seen in `feature_4_test()` function)

Test Functions



```
✓ GraphManagerTest (daniel.tran)                4 sec 446 ms
    ✓ DFS_test_1()                                     67 ms
    ✓ DFS_test_2()                                      6 ms
    ✓ DFS_test_3()                                      3 ms
    ✓ DFS_test_4()                                      3 ms
    ✓ DFS_test_5()                                      3 ms
    ✓ BFS_giga_performance_test()               1 sec 268 ms
    ✓ DFS_giga_performance_test()               1 sec 131 ms
    ✓ BFS_test_1()                                      2 ms
    ✓ BFS_test_2()                                      1 ms
    ✓ BFS_test_3()
    ✓ BFS_test_4()
    ✓ BFS_test_5()
    ✓ DFS_performance_test()                            5 ms
    ✓ BFS_performance_test()                            2 ms
    ✓ RandomWalk_performance_test()                     6 ms
    ✓ RandomWalk_mega_performance_test()               99 ms
    ✓ feature_1_test()                                  1 ms
    ✓ feature_2_test()                                  1 ms
    ✓ feature_3_test()
    ✓ feature_4_test()                                471 ms
    ✓ scenario_1_test()
    ✓ scenario_2_test()
    ✓ scenario_3_test()
    ✓ BFS_mega_performance_test()                      59 ms
    ✓ DFS_mega_performance_test()                      56 ms
    ✓ RandomWalk_giga_performance_te:           1 sec 262 ms
    ✓ RandomWalk_test_1()
```

Above is a screenshot of the test performed on all test functions, where scenario_2 and 3 tests are failed on purpose according to the project description. And it is an example of all tests run on a decent computer (Apple M1 Pro) and their expected results.


Run/Debug Configurations

JUnit
GraphManagerTest

Name: GraphManagerTest
Store as project file

Build and run
Modify options

java 23 SDK of 'CSE464-GraphManager' module

-ea

Class | daniel.tran.GraphManagerTest

Press ⌥ for field hints

Working directory: $MODULE_WORKING_DIR$

Environment variables: Environment variables

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started
Search for tests: In single module

Code Coverage
Modify

Packages and classes to include in coverage data

- [x] daniel.tran.*

Edit configuration templates...

Run | Cancel | Apply | OK

Above is the test file's run configuration should you decide to run it inside InteliJ IDEA.

Merges

BFS to main: Github
DFS merge conflicts: Github
DFS to main: Github

Branches

BFS: Github
DFS: Github
Refactor: Github

Features

Feature 1: Github
Feature 2: Github
Feature 3: Github
Feature 4: Github
Remove node/ edge: Github
BFS: Github
DFS: Github

Refactoring

1. removeEdge refactor
   a. Github
   b. Reason: Readability
   c. Type: Extract Variable
2. addN refactor
   a. Github
   b. Reason: Readability
   c. Type: Extract Variable
3. addEdge refactor
   a. Github
   b. Reason: Readability
   c. Type: Extract Variable
4. Refactor printing node not exist
   a. Github
   b. Reason: Reduce redundant code
   c. Type: Extract Method
5. Automated refactor

a. Github
b. Reason: Unify formatting for readability
c. Type: automated

Template pattern design

The shared steps of the traversal algorithms are defined once in the base class, reducing code duplication and also act as placeholders for steps that differ between algorithms.New traversal strategies can be added by extending the base class without modifying existing code thanks to the base class being an abstract class

Commit: Github

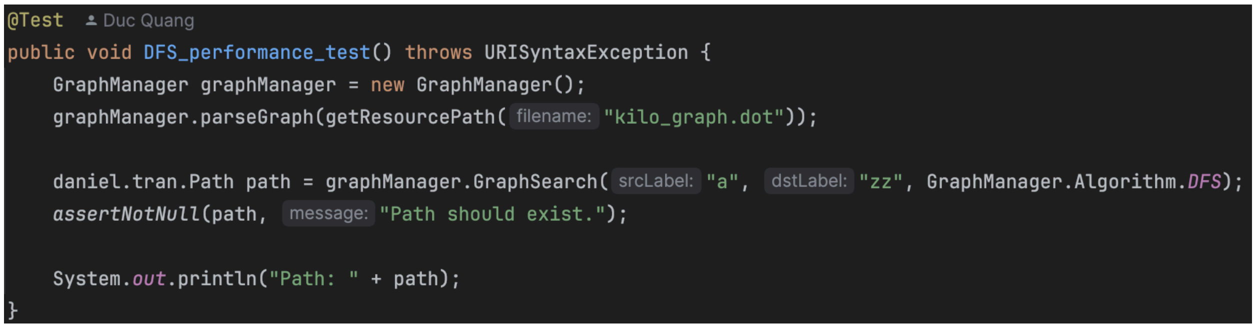
Strategy pattern design

We define the contract for the search algorithms with the search method where each class implements the SearchStrategy interface and encapsulates the logic specific to BFS, DFS, or RandomWalk. By using a SearchStrategy reference to execute the algorithm, the GraphSearch method acts as the context that selects and utilizes the appropriate strategy at runtime based on the algo parameter.

Commit: Github

How to run

An example of running the BFS, DFS, or RandomWalk algorithm can be seen in the test cases like below:



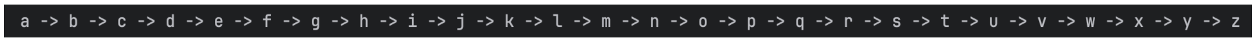
```
@Test  Duc Quang
public void DFS_performance_test() throws URISyntaxException {
    GraphManager graphManager = new GraphManager();
    graphManager.parseGraph(getResourcePath( filename: "kilo_graph.dot"));

    daniel.tran.Path path = graphManager.GraphSearch( srcLabel: "a",  dstLabel: "zz", GraphManager.Algorithm.DFS);
    assertNotNull(path,  message: "Path should exist.");

    System.out.println("Path: " + path);
}
```

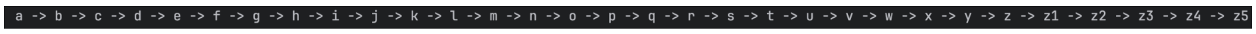
Expected output

DFS_test_1():



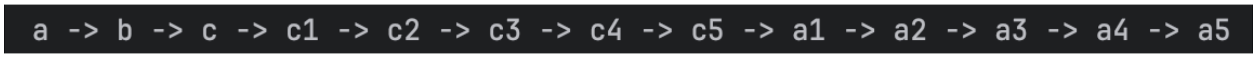
```
a -> b -> c -> d -> e -> f -> g -> h -> i -> j -> k -> l -> m -> n -> o -> p -> q -> r -> s -> t -> u -> v -> w -> x -> y -> z
```

DFS_test_2():



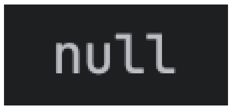
```
a -> b -> c -> d -> e -> f -> g -> h -> i -> j -> k -> l -> m -> n -> o -> p -> q -> r -> s -> t -> u -> v -> w -> x -> y -> z -> z1 -> z2 -> z3 -> z4 -> z5
```

DFS_test_3():



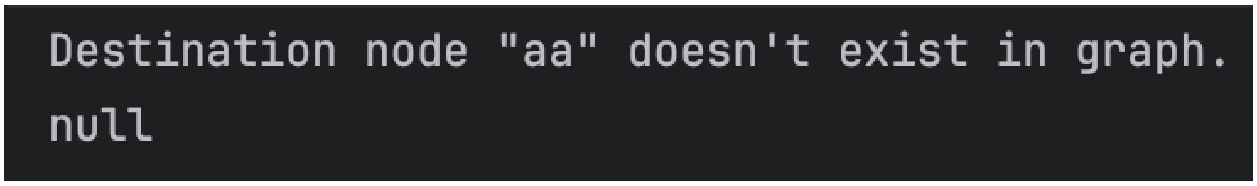
```
a -> b -> c -> c1 -> c2 -> c3 -> c4 -> c5 -> a1 -> a2 -> a3 -> a4 -> a5
```

DFS_test_4():



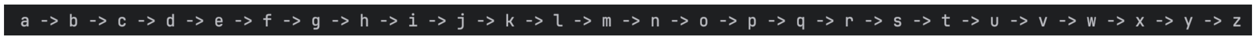
```
null
```

DFS_test_5():



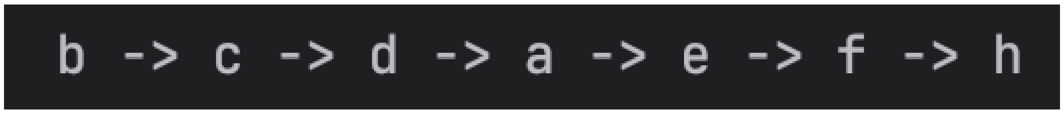
```
Destination node "aa" doesn't exist in graph.
null
```

BFS_test_1():



```
a -> b -> c -> d -> e -> f -> g -> h -> i -> j -> k -> l -> m -> n -> o -> p -> q -> r -> s -> t -> u -> v -> w -> x -> y -> z
```

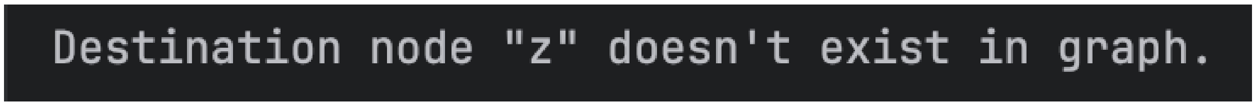
BFS_test_2():



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

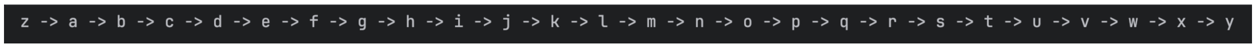
BFS_test_3():

BFS_test_4():



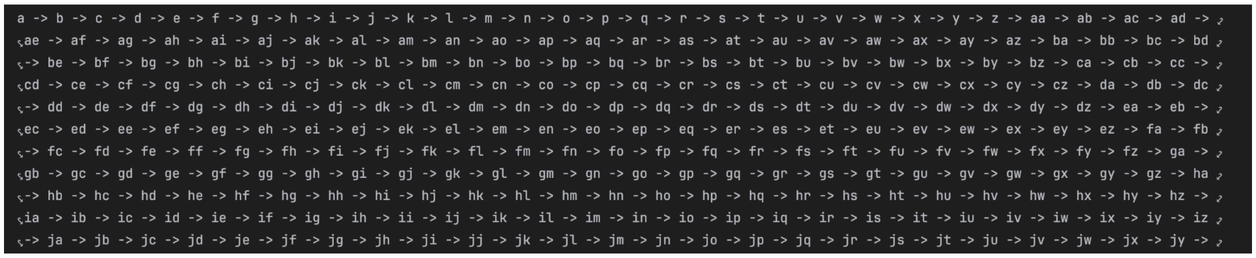
```
Destination node "z" doesn't exist in graph.
```

BFS_test_5():



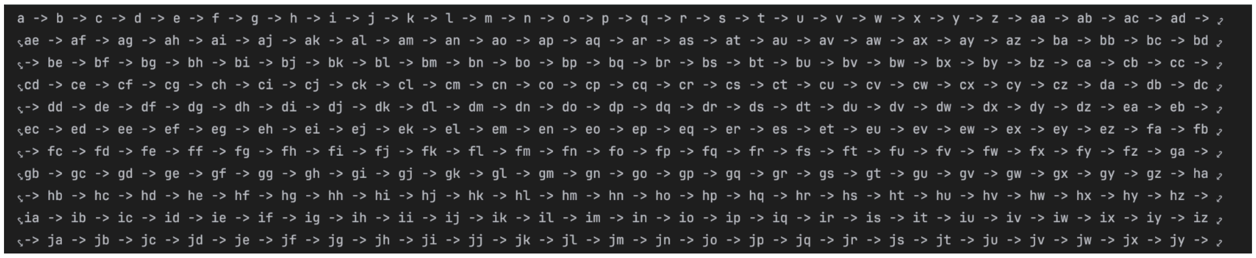
```
z -> a -> b -> c -> d -> e -> f -> g -> h -> i -> j -> k -> l -> m -> n -> o -> p -> q -> r -> s -> t -> u -> v -> w -> x -> y
```

DFS_performance_test(): Too big to include all



```
a -> b -> c -> d -> e -> f -> g -> h -> i -> j -> k -> l -> m -> n -> o -> p -> q -> r -> s -> t -> u -> v -> w -> x -> y -> z -> aa -> ab -> ac -> ad -> ↩
↪ae -> af -> ag -> ah -> ai -> aj -> ak -> al -> am -> an -> ao -> ap -> aq -> ar -> as -> at -> au -> av -> aw -> ax -> ay -> az -> ba -> bb -> bc -> bd ↩
↪-> be -> bf -> bg -> bh -> bi -> bj -> bk -> bl -> bm -> bn -> bo -> bp -> bq -> br -> bs -> bt -> bu -> bv -> bw -> bx -> by -> bz -> ca -> cb -> cc -> ↩
↪cd -> ce -> cf -> cg -> ch -> ci -> cj -> ck -> cl -> cm -> cn -> co -> cp -> cq -> cr -> cs -> ct -> cu -> cv -> cw -> cx -> cy -> cz -> da -> db -> dc ↩
↪-> dd -> de -> df -> dg -> dh -> di -> dj -> dk -> dl -> dm -> dn -> do -> dp -> dq -> dr -> ds -> dt -> du -> dv -> dw -> dx -> dy -> dz -> ea -> eb -> ↩
↪ec -> ed -> ee -> ef -> eg -> eh -> ei -> ej -> ek -> el -> em -> en -> eo -> ep -> eq -> er -> es -> et -> eu -> ev -> ew -> ex -> ey -> ez -> fa -> fb ↩
↪-> fc -> fd -> fe -> ff -> fg -> fh -> fi -> fj -> fk -> fl -> fm -> fn -> fo -> fp -> fq -> fr -> fs -> ft -> fu -> fv -> fw -> fx -> fy -> fz -> ga -> ↩
↪gb -> gc -> gd -> ge -> gf -> gg -> gh -> gi -> gj -> gk -> gl -> gm -> gn -> go -> gp -> gq -> gr -> gs -> gt -> gu -> gv -> gw -> gx -> gy -> gz -> ha ↩
↪-> hb -> hc -> hd -> he -> hf -> hg -> hh -> hi -> hj -> hk -> hl -> hm -> hn -> ho -> hp -> hq -> hr -> hs -> ht -> hu -> hv -> hw -> hx -> hy -> hz -> ↩
↪ia -> ib -> ic -> id -> ie -> if -> ig -> ih -> ii -> ij -> ik -> il -> im -> in -> io -> ip -> iq -> ir -> is -> it -> iu -> iv -> iw -> ix -> iy -> iz ↩
↪-> ja -> jb -> jc -> jd -> je -> jf -> jg -> jh -> ji -> jj -> jk -> jl -> jm -> jn -> jo -> jp -> jq -> jr -> js -> jt -> ju -> jv -> jw -> jx -> jy -> ↩
```

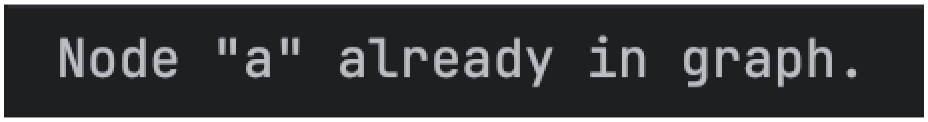
BFS_performance_test(): Too big to include all



```
a -> b -> c -> d -> e -> f -> g -> h -> i -> j -> k -> l -> m -> n -> o -> p -> q -> r -> s -> t -> u -> v -> w -> x -> y -> z -> aa -> ab -> ac -> ad -> ↩
↪ae -> af -> ag -> ah -> ai -> aj -> ak -> al -> am -> an -> ao -> ap -> aq -> ar -> as -> at -> au -> av -> aw -> ax -> ay -> az -> ba -> bb -> bc -> bd ↩
↪-> be -> bf -> bg -> bh -> bi -> bj -> bk -> bl -> bm -> bn -> bo -> bp -> bq -> br -> bs -> bt -> bu -> bv -> bw -> bx -> by -> bz -> ca -> cb -> cc -> ↩
↪cd -> ce -> cf -> cg -> ch -> ci -> cj -> ck -> cl -> cm -> cn -> co -> cp -> cq -> cr -> cs -> ct -> cu -> cv -> cw -> cx -> cy -> cz -> da -> db -> dc ↩
↪-> dd -> de -> df -> dg -> dh -> di -> dj -> dk -> dl -> dm -> dn -> do -> dp -> dq -> dr -> ds -> dt -> du -> dv -> dw -> dx -> dy -> dz -> ea -> eb -> ↩
↪ec -> ed -> ee -> ef -> eg -> eh -> ei -> ej -> ek -> el -> em -> en -> eo -> ep -> eq -> er -> es -> et -> eu -> ev -> ew -> ex -> ey -> ez -> fa -> fb ↩
↪-> fc -> fd -> fe -> ff -> fg -> fh -> fi -> fj -> fk -> fl -> fm -> fn -> fo -> fp -> fq -> fr -> fs -> ft -> fu -> fv -> fw -> fx -> fy -> fz -> ga -> ↩
↪gb -> gc -> gd -> ge -> gf -> gg -> gh -> gi -> gj -> gk -> gl -> gm -> gn -> go -> gp -> gq -> gr -> gs -> gt -> gu -> gv -> gw -> gx -> gy -> gz -> ha ↩
↪-> hb -> hc -> hd -> he -> hf -> hg -> hh -> hi -> hj -> hk -> hl -> hm -> hn -> ho -> hp -> hq -> hr -> hs -> ht -> hu -> hv -> hw -> hx -> hy -> hz -> ↩
↪ia -> ib -> ic -> id -> ie -> if -> ig -> ih -> ii -> ij -> ik -> il -> im -> in -> io -> ip -> iq -> ir -> is -> it -> iu -> iv -> iw -> ix -> iy -> iz ↩
↪-> ja -> jb -> jc -> jd -> je -> jf -> jg -> jh -> ji -> jj -> jk -> jl -> jm -> jn -> jo -> jp -> jq -> jr -> js -> jt -> ju -> jv -> jw -> jx -> jy -> ↩
```

feature_1_test(): Blank

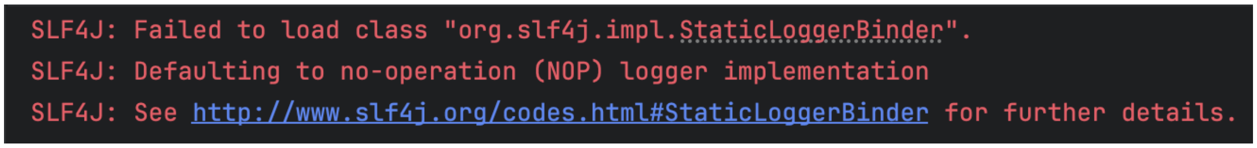
feature_2_test():



```
Node "a" already in graph.
```

feature_3_test(): Blank

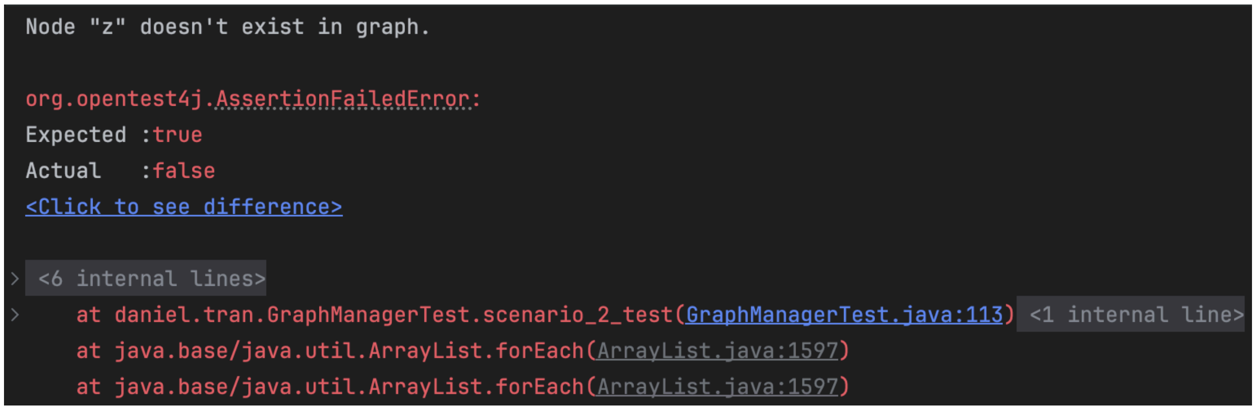
feature_4_test():



```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
```

scenario_1_test(): Blank

scenario_2_test():

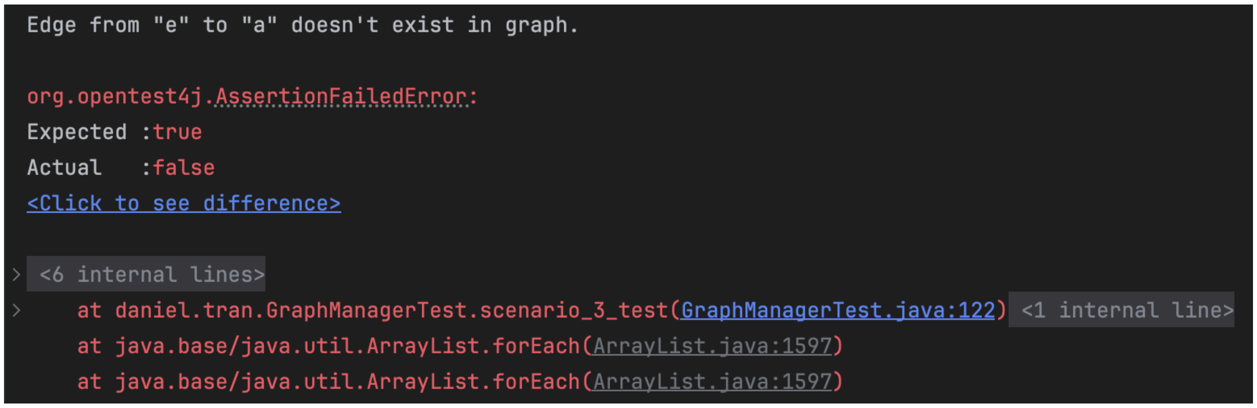


```
Node "z" doesn't exist in graph.

org.opentest4j.AssertionFailedError:
Expected :true
Actual   :false
<Click to see difference>

> <6 internal lines>
>     at daniel.tran.GraphManagerTest.scenario_2_test(GraphManagerTest.java:113) <1 internal line>
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
```

scenario_3_test():



```
Edge from "e" to "a" doesn't exist in graph.

org.opentest4j.AssertionFailedError:
Expected :true
Actual   :false
<Click to see difference>

> <6 internal lines>
>     at daniel.tran.GraphManagerTest.scenario_3_test(GraphManagerTest.java:122) <1 internal line>
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1597)
```

DFS/BFS mega/giga/ tera_performance_test(): Too big to include all of the outputs