

## **GraphDot Project 3 - CSE 464**

**Brady Flahart**

**Repo:** <https://github.com/btflahar/CSE-464-btflahar-graph.git>

mvn package command works and will properly build

There should be a total of 15 Tests that run when package is run

Java 17

Apache Maven

GraphViz

Is required

mvn clean package to start

Place DOT file in the root (input.dot)

The tests can be run on either

src/test/java/btflahar/asu/edu/graphDot/FinalDemoTest.java

Or src/main/java/btflahar/asu/edu/graphDot/GraphApp

They can be run by simply going to the respective code block (main for either file) and running the green triangle (run button) for either.

The example input.dot given is very simple

```
digraph G {  
    a -> b;  
    b -> c;  
    c -> d;  
    d -> a;  
    a -> e;  
    e -> f;  
    e -> g;  
    f -> h;  
    g -> h;  
}
```

The expected outputs after running the TestDemo would be a successful BFS, DFS, and Random Walk

These outputs would look like

## BFS

BFS Execution

visiting a  
visiting a -> b  
visiting a -> e  
visiting a -> b -> c  
visiting a -> e -> f  
visiting a -> e -> g  
visiting a -> b -> c -> d  
visiting a -> e -> f -> h  
Final BFS path: a -> e -> f -> h

## DFS

DFS Execution

visiting a  
visiting a -> e  
visiting a -> e -> g  
visiting a -> e -> g -> h  
Final DFS path: a -> e -> g -> h

## RandomWalk

Random Walk output will vary but examples of it can be shown below

Random Walk Execution

visiting a  
visiting a -> e  
visiting a -> e -> f  
visiting a -> e -> f -> h  
Random Walk #1: a -> e -> f -> h  
visiting a  
visiting a -> e  
visiting a -> e -> f  
visiting a -> e -> f -> h  
Random Walk #2: a -> e -> f -> h

visiting a  
visiting a -> e  
visiting a -> e -> f  
visiting a -> e -> f -> h  
Random Walk #3: a -> e -> f -> h  
visiting a  
visiting a -> b  
visiting a -> b -> c  
visiting a -> b -> c -> d  
Random Walk #4: (dead end)  
visiting a  
visiting a -> b  
visiting a -> b -> c  
visiting a -> b -> c -> d  
Random Walk #5: (dead end)  
visiting a  
visiting a -> b  
visiting a -> b -> c  
visiting a -> b -> c -> d  
Random Walk #6: (dead end)  
visiting a  
visiting a -> e  
visiting a -> e -> f  
visiting a -> e -> f -> h  
Random Walk #7: a -> e -> f -> h  
visiting a  
visiting a -> b  
visiting a -> b -> c  
visiting a -> b -> c -> d  
Random Walk #8: (dead end)  
visiting a  
visiting a -> e  
visiting a -> e -> f  
visiting a -> e -> f -> h  
Random Walk #9: a -> e -> f -> h  
visiting a  
visiting a -> e  
visiting a -> e -> f  
visiting a -> e -> f -> h  
Random Walk #10: a -> e -> f -> h

The desired start/end node are entered at  
src/test/java/btflahar/asu/edu/graphDot/FinalDemoTest.java  
At “Start node” and “End Node”

Enter the desired node for each.

Here are images of each respective output  
BFS

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `FinalDemoTest.java` file, which contains Java code for testing graph search algorithms. The test method `demoAllFunctions()` uses the `GraphApp` class to perform Breadth-First Search (BFS) from node 'a' to node 'h'. The test passes, and the console output shows the BFS execution path: a -> e -> f -> h. The Maven tool window on the right shows the `test` lifecycle step selected.

```
package btflahar.asu.edu.graphDot;
import org.junit.jupiter.api.Test;
public class FinalDemoTest {
    @Test
    public void demoAllFunctions() throws Exception {
        GraphApp app = new GraphApp();
        String inputDotFile = "input.dot";
        app.parseGraph(inputDotFile);
        String start = "a";
        String dest = "h";
        System.out.println("\nBFS Execution");
        Path bfsPath = app.graphSearch(start, dest, Algorithm.BFS);
        System.out.println("Final BFS path: " + bfsPath);
        System.out.println("\nDFS Execution");
    }
}
```

Run

FinalDemoTest

FinalDemoTest (btflahar.asu.edu.graphDot) 58 ms ✓ 1 test passed 1 test total, 58ms

BFS Execution

visiting a

visiting a -> b

visiting a -> e

visiting a -> b -> c

visiting a -> e -> f

visiting a -> e -> g

visiting a -> b -> c -> d

visiting a -> e -> f -> h

Final BFS path: a -> e -> f -> h

# DFS

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "CSE464-btflahar-graph". It contains a "src" directory with "main" and "test" packages. "main" contains "btflahar.asu.edu.graphDot" which includes classes like AbstractGraphSearch, Algorithm, BFS, DFS, GraphApp, GraphContext, Path, RandomWalkSearch, and SearchStrategy. "test" contains "btflahar.asu.edu.graphDot" with a "java" directory containing "FinalDemoTest.java".
- Code Editor:** The code for "FinalDemoTest.java" is shown, demonstrating a DFS search from node 'a' to node 'h'. The code uses a GraphApp object to perform the search and prints the resulting path.
- Maven Tool Window:** The right side shows the Maven tool window with the "test" goal selected.
- Run Tab:** The "Run" tab shows the execution of "FinalDemoTest". The output shows two executions: one for Breadth-First Search (BFS) and one for Depth-First Search (DFS). Both executions start at 'a' and reach 'h'.

# RandomWalk 1 and 2

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "CSE464-btflahar-graph". It contains a "src" directory with "main" and "test" packages. "main" contains "btflahar.asu.edu.graphDot" with classes like BFS, RandomWalkSearch, GraphContext, DFS, GraphAppTest, and FinalDemoTest. "test" contains "btflahar.asu.edu.graphDot" with a "java" directory containing "FinalDemoTest.java".
- Code Editor:** The code for "FinalDemoTest.java" is shown, demonstrating random walks starting from node 'a'. The code uses a GraphApp object to perform the walks and prints the resulting paths.
- Maven Tool Window:** The right side shows the Maven tool window with the "Lifecycle" section expanded, showing various goals like clean, validate, compile, test, package, verify, install, site, and deploy.
- Run Tab:** The "Run" tab shows the execution of "FinalDemoTest". The output shows multiple random walks starting from 'a' and ending at various nodes, with some being dead ends.

```
Random Walk Execution
visiting a
visiting a -> b
visiting a -> b -> c
visiting a -> b -> c -> d
Random Walk #1: (dead end)
visiting a
visiting a -> b
visiting a -> b -> c
visiting a -> b -> c -> d
Random Walk #2: (dead end)
visiting a
visiting a -> e
visiting a -> e -> g
visiting a -> e -> g -> h
Random Walk #3: a -> e -> g -> h
visiting a
visiting a -> e
visiting a -> e -> g
visiting a -> e -> g -> h
Random Walk #4: a -> e -> g -> h
visiting a
visiting a -> b
visiting a -> b -> c
visiting a -> b -> c -> d
Random Walk #5: (dead end)
visiting a
visiting a -> e
visiting a -> e -> f
visiting a -> e -> f -> h
```

## Refactors

### Refactor 1:

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/e8962f645620b6ae4eb5f32d2da2b372ad5d9924>

### Refactor 2:

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/beefdccc793258fc255ce5450c8a654ca54d0eec>

### Refactor 3:

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/233b72533b0bf5b03ffa80adefc6ecd81840fe01>

Refactor 4:

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/ff2e1d0bc7ad6b810068208632a76644f2778b3f>

Refactor 5:

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/2578141c98517c857962f4ff3d01077896e9f358>

Last 2 Commits (one after pull request)

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/d76d07919a1cab1e061409049e5f01124880205>

<https://github.com/btflahar/CSE-464-btflahar-graph/commit/58b773eb0feb12672a56e8ae41066897d1a9cac9> (after pull request.)