CSE 464 Project Part 1
Joshua Russell

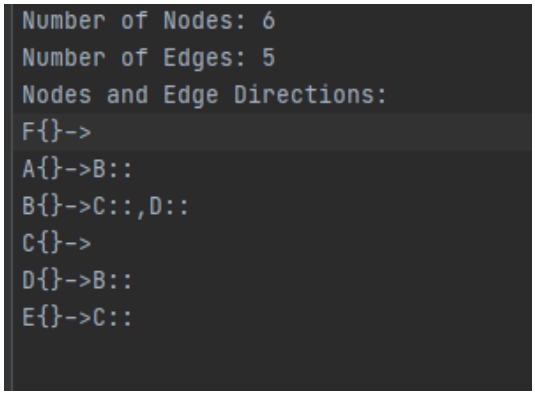
Link to Repo:
https://github.com/joshrussell170/CSE4642023jrrusse9/tree/master

To run the code you can simply do “mvn package” and the code will compile, run the tests, and spit out the evaluation. I also wrote examples in the main() of the DOTParser class which are commented out. You can uncomment these 1 at a time and test each specific feature.

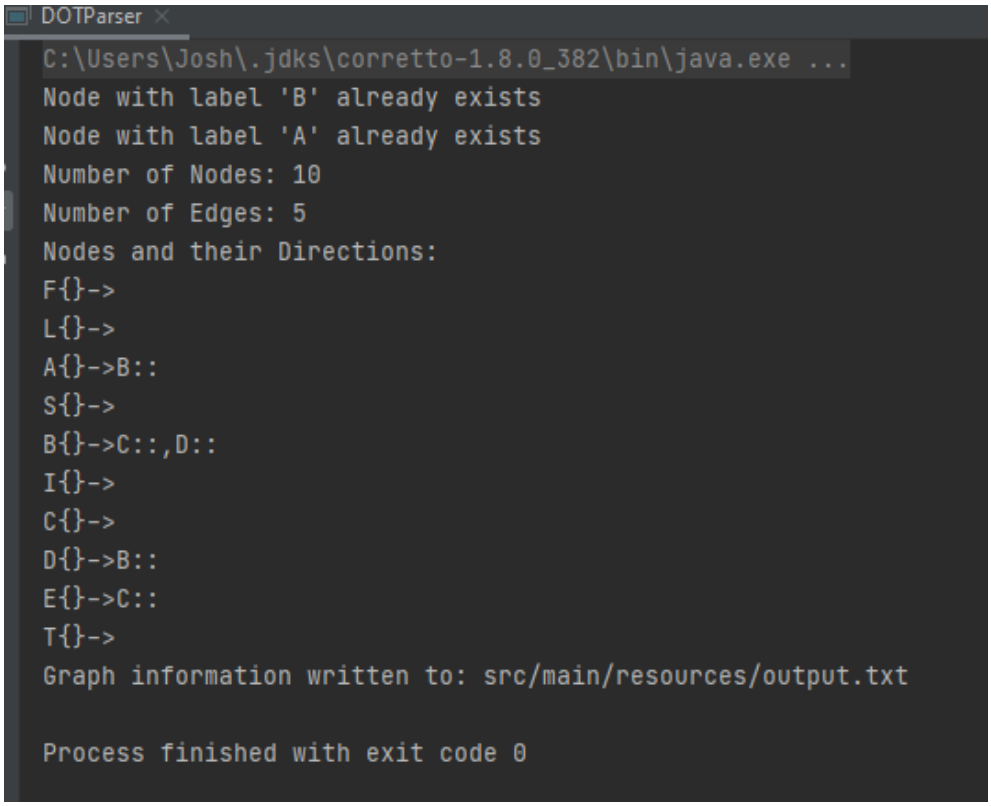
- All inputs are in files (mainly color.dot) or directly written into the code
- All outputs are presented in the form requested by the Project Documentation
- I created a toString() method in the Path class to use for output for the Search algorithm

Feature 1 Outputs:

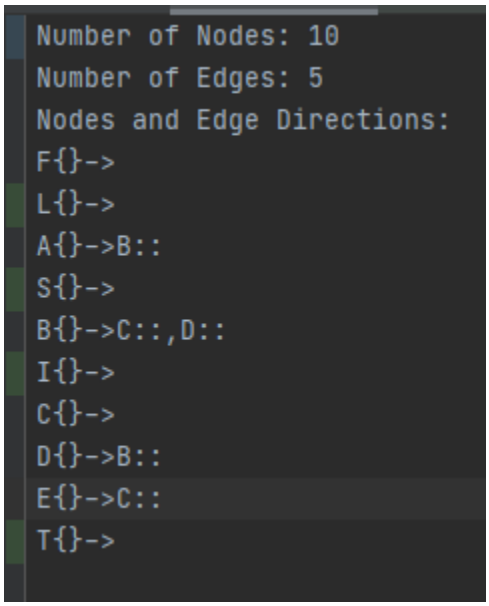
output.txt:



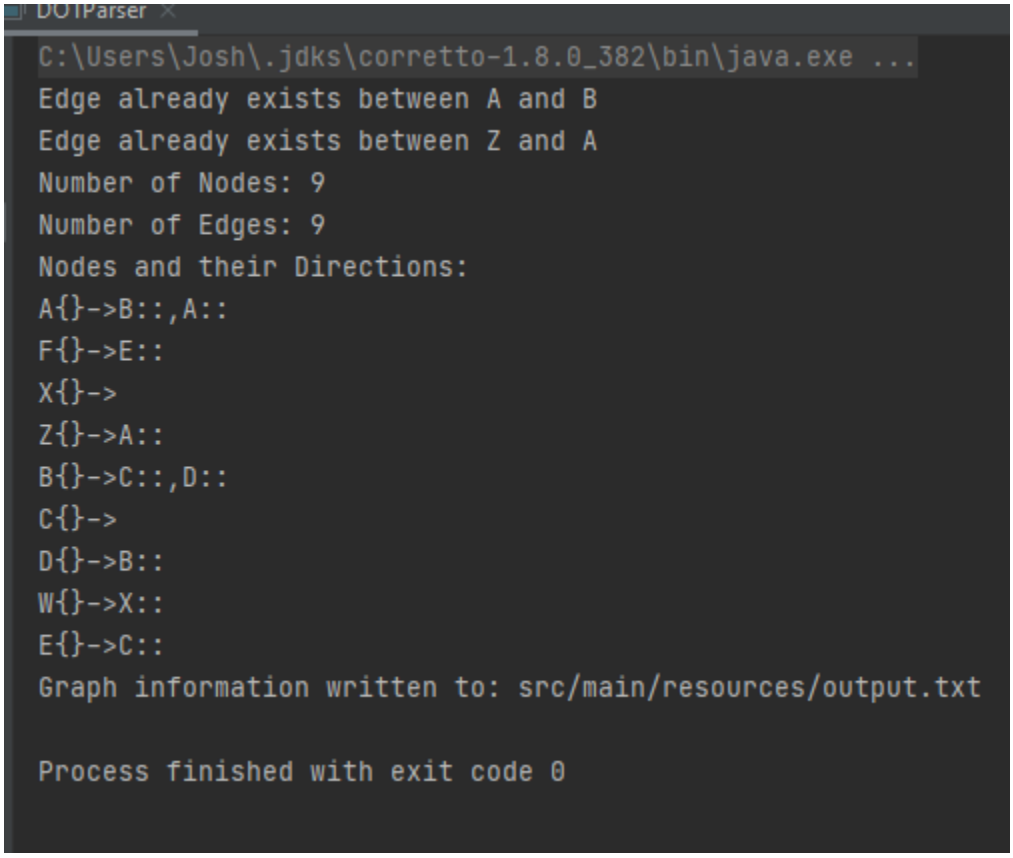
Feature 2 Outputs:



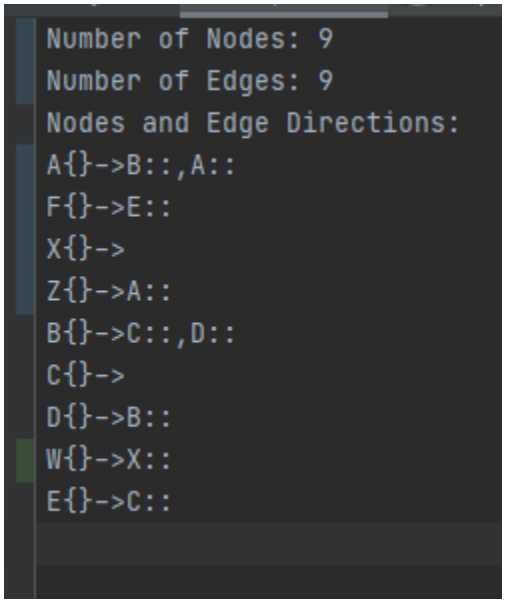
Output.txt:



Feature 3 Outputs:



Output.txt:



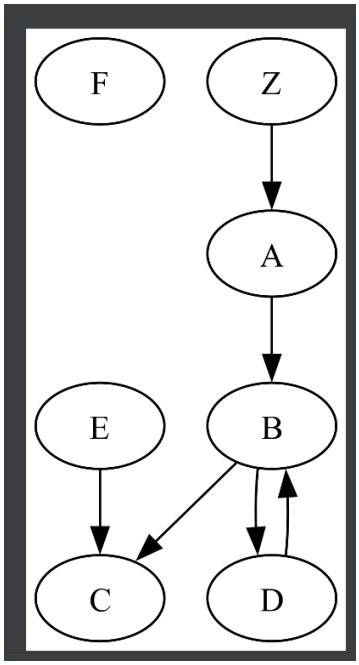
Feature 4 Outputs:

```
C:\Users\Josh\.jdks\corretto-1.8.0_382\bin\java.exe ...
Graph saved to src/main/resources/output.dot
Graph saved as PNG to src/main/resources/output.png

Process finished with exit code 0
```

Output.dot:

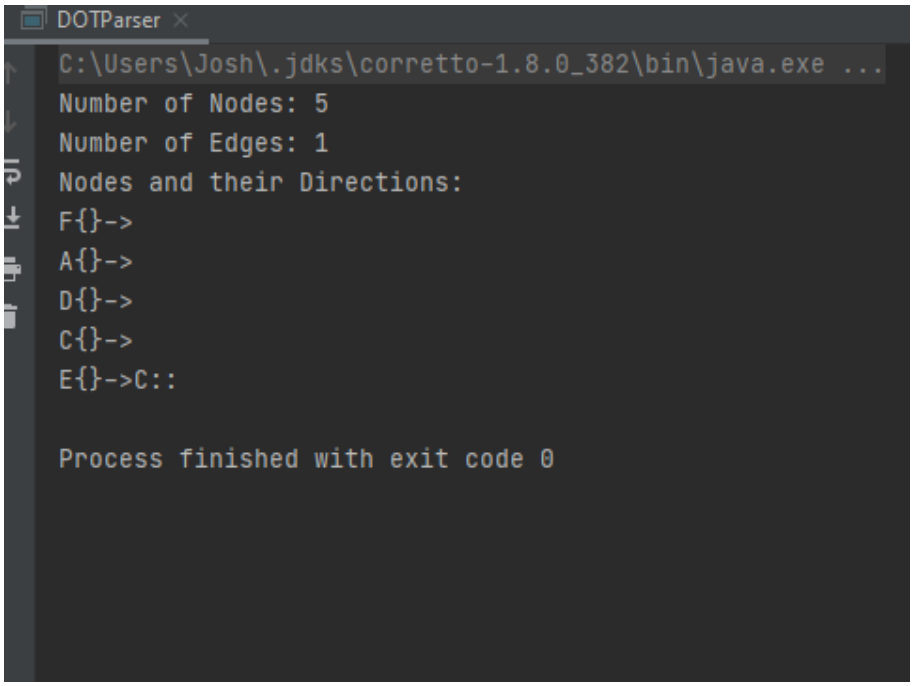
```
digraph "G" {
"F"
"A" -> "B"
"B" -> "C"
"B" -> "D"
"D" -> "B"
"E" -> "C"
"Z" -> "A"
}
```

Output.png:

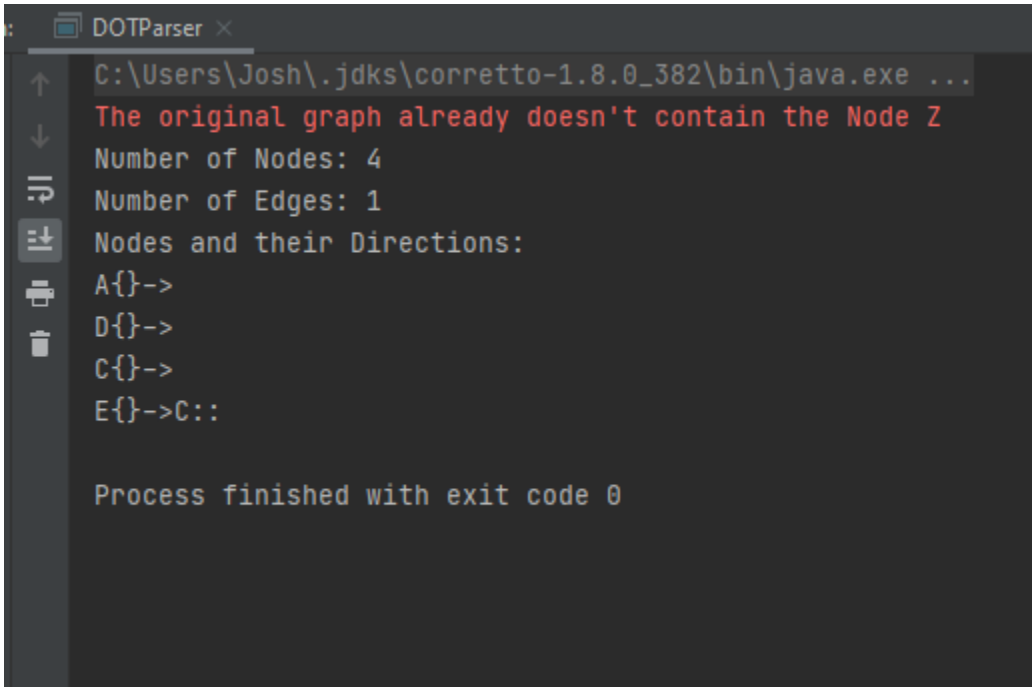


These next features were tested on the color.dot file which has a graphic example above without the Z node.

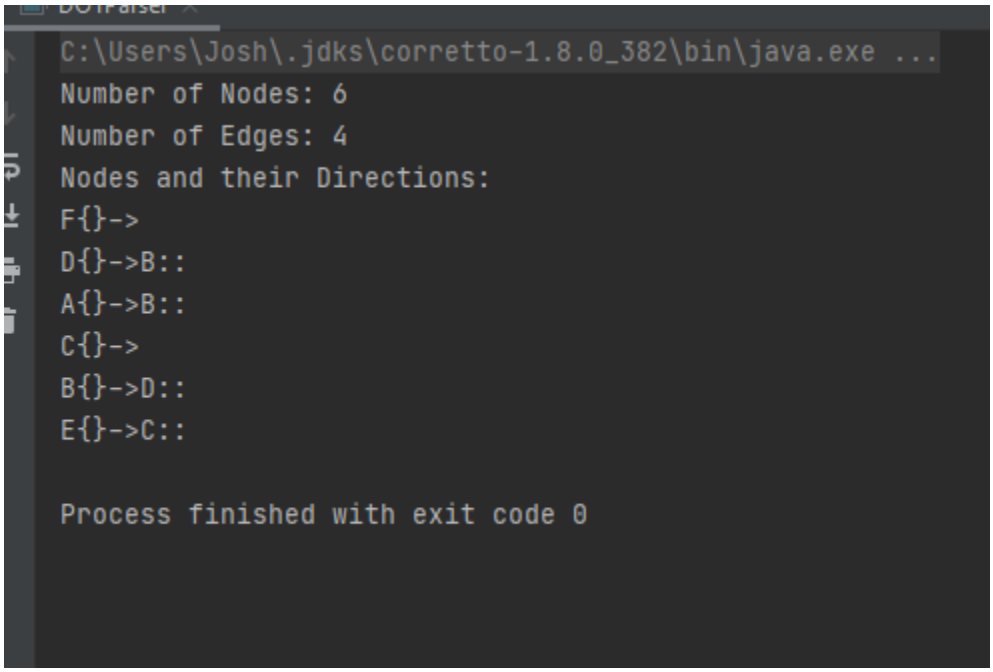
Remove Node Feature (removing just node B):



Remove Nodes Feature (removing nodes B, F, and Z*):
* Z does not exist initially

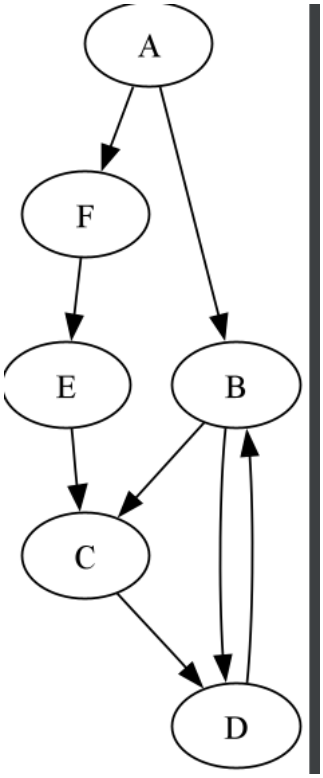


Remove Edge Feature (removed edge between B and C):

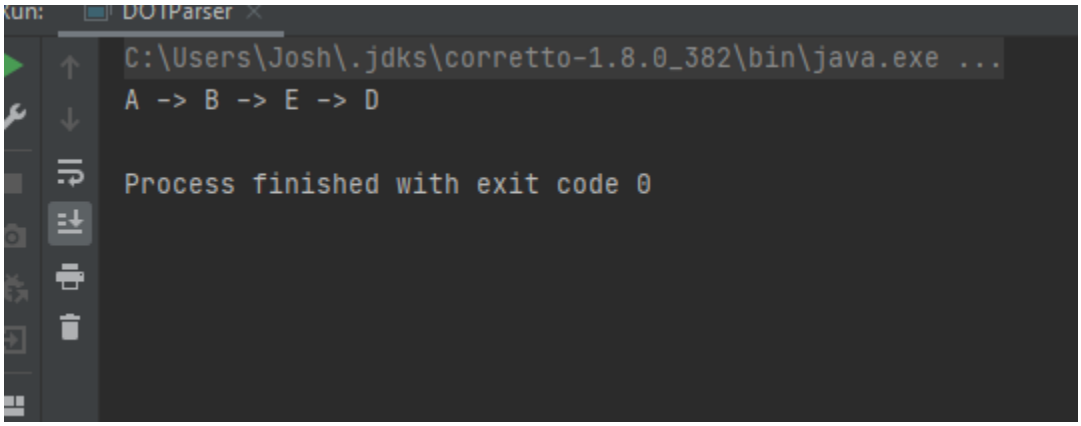


Graph search was done using this graph to show differences:

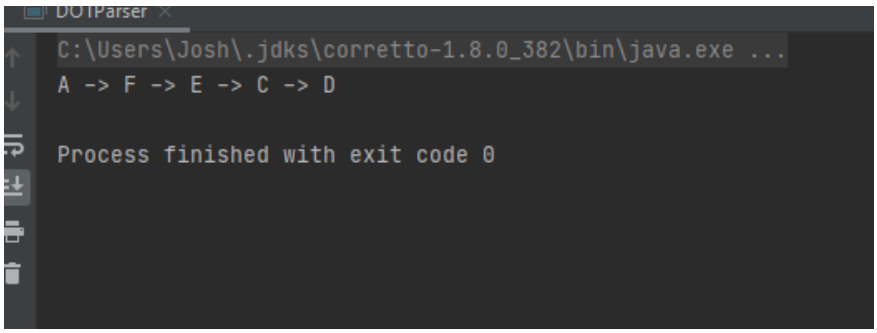
```
digraph G {
    // Define nodes
    A;
    B;
    C;
    D;
    E;
    F;

    // Define directed edges
    A -> B;
    B -> C;
    B -> D;
    E -> C;
    D -> B;
    C -> D;
    A -> F;
    F -> E;
}
```



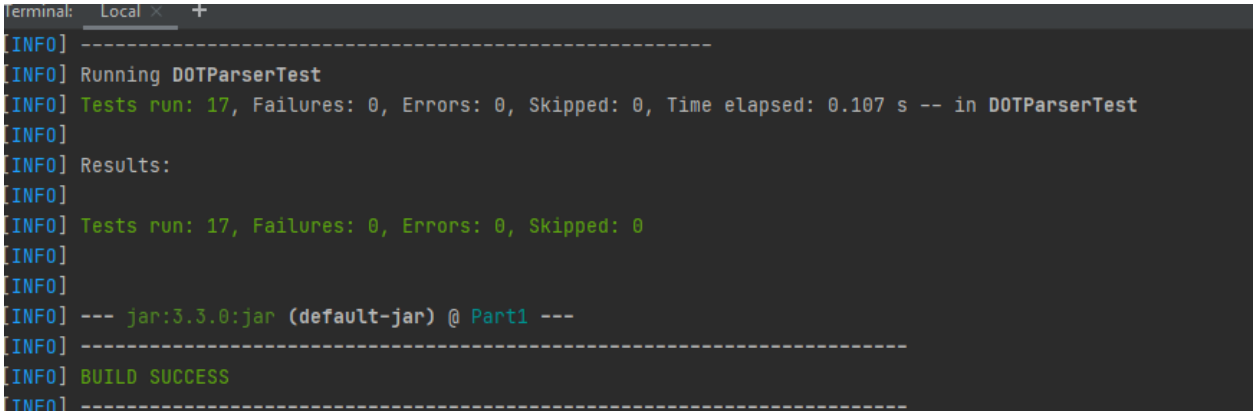
Graph Search Output (bfs):



Graph Search Output (dfs):



Output from running “mvn package”:



1st Commit for Feature 1:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/8c3a1f6f68486f0ee337b64a06c890f946351464

2nd Commit for Feature 2:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/0a0b681cf68fb1daf093d59da0dbeb2d0db0ab2f

3rd Commit for Feature 3:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/7a4b56919122f277f76b4ec46df448954d6da001

4th Commit for Feature 4:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/1ee337e758df92e28e0c3ba98930196c9565feab
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/4f35e950aa71cd9345b3e7ecd16269c76f6ffbc6

Commit for Test Cases:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/e33ccdc9dd967866e90db5c6441138c1b621141d
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/91f5467cc1d6bfc35bd924b248116c03a45a43fa

Final Part 1 Commit:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/eecfd85e76bf2c8479b6661563c39df1eff00649

Commit for new features:

https://github.com/joshrussell170/CSE4642023jrrusse9/commit/cc5132f7c66809a1c6fa4bf5510a8aba34dad81c

BFS Commit:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/67c243f609e6ea5b3a330e082277b14f3a6a6cc6

DFS Commit:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/e76661ed046ce63aaaf11660c4df7030b444b650

Commit for enum search:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/375e14b5f2f1ad1cb1267da00de5ebe8024437e4

Continuous integration:
https://github.com/joshrussell170/CSE4642023jrrusse9/actions/workflows/maven.yml

BFS Branch:
https://github.com/joshrussell170/CSE4642023jrrusse9/tree/bfs

DFS Branch:
https://github.com/joshrussell170/CSE4642023jrrusse9/tree/dfs

Merge Build:
https://github.com/joshrussell170/CSE4642023jrrusse9/actions/runs/6766759120/job/18388304476

Refactor #1:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/db4eb5372f40acfd2cb89f6d87d69a039e61f25c

Refactor #2:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/569e6234a2180bf2aa229e670eaae98857fdd727

Refactor#3:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/4e3b73f2f1aa8938d4c25f0939e85e1cc1b37d22

Refactor #4:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/4cadeb8a1c43ae5e3e70c88af225efcfb825c2c6

Refactor #5:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/9a4f21994dde1a7cfc5f2a116f607d28158fd7d3

Template Method commit:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/e6e07b57e602dc8b2d4b87a275100022b4079a08
I applied the template method by creating an abstract class called “Search” which holds a few helper methods for the classes “dfsSearch” and “bfsSearch”. I found the overlapping logic between the two search methods and extracted the logic into 4 helper methods within the “Search” class. These methods are:

- moveNode()
  - Responsible for selecting the next node that will be used in the specific algorithms
- returnPathCheck()
  - Responsible for checking if the search is successful and returns the organized path object
- reconstructPath()
  - Responsible for reorganizing the Path object in the order that the search algorithm traversed the graph; used in returnPathCheck() method
- nodesExist()
  - Used to check that both nodes exist within the graph and returns the starting node for both specific algorithms

This left only algorithm-specific logic within the “dfsSearch” and “bfsSearch” classes. Each would use the helper methods but dfs would use a stack where bfs uses a queue.

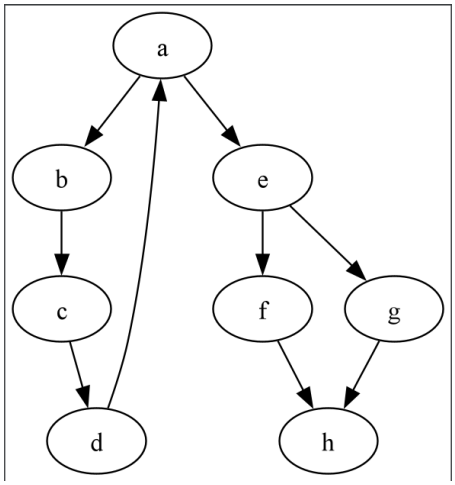
Strategy Method commit:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/7d93c4cff2e6ec1885a1f802e6ef3d31fe95541d
I applied the Strategy Method by creating a “Strategy” interface that defines the search() method. Then “dfsSearch” and “bfsSearch” classes implement the Strategy and define the implementation of search() with the algorithm-specific approach. They use overriding to create their own version of the search() method. Then I created a “Context” class that is used to actually execute the algorithms within the main class. So you need to create a Context object by giving it a Strategy object where the strategy is a particular search algorithm (bfs/dfs). Then you use the Context object to execute the strategy search().

Random Walk Commit:
https://github.com/joshrussell170/CSE4642023jrrusse9/commit/0995d41b8b7744d3ae2fc36db87a66247bd52207
This feature does a random search from a source node to a destination node. It does this by doing a random walk of the graph from nodes accessible to the source node. If the graph comes across the destination node, then a path is created and returned to be printed to the console. This function also logs all the attempted paths as the program runs to show the randomness of

the program. Note: if a path doesn't exist between nodes, then a message will be returned saying that no path exists and that the path was null.

To run this feature, there is a section of code in the main() function under the comment "feature 10" which runs the random walk algorithm. To change the different nodes you want to search between, you need to change the first and second arguments to the string of the name of the node you want to search. The first argument is the source node, the second is the destination node.

Examples:
Run on this graph



Searching from a -> d

```
Visiting Path: a -> e -> g -> h
Visiting Path: a -> e -> g
Visiting Path: a -> b -> c -> d
Path Found: a -> b -> c -> d

Process finished with exit code 0
```

```
Visiting Path: a -> e -> g -> h
Visiting Path: a -> b -> c
Visiting Path: a -> e
Visiting Path: a -> b
Visiting Path: a -> b -> c -> d
Path Found: a -> b -> c -> d

Process finished with exit code 0
```

Searching from a-> g

```
Visiting Path: a -> b
Visiting Path: a -> e -> g
Path Found: a -> e -> g

Process finished with exit code 0
```

```
Visiting Path: a -> b
Visiting Path: a -> b -> c -> d
Visiting Path: a -> e
Visiting Path: a -> b -> c
Visiting Path: a -> e -> g
Path Found: a -> e -> g

Process finished with exit code 0
```

Searching from d -> c

```
Visiting Path: d -> a -> e -> g -> h
Visiting Path: d -> a
Visiting Path: d -> a -> b
Visiting Path: d -> a -> e -> f
Visiting Path: d -> a -> e
Visiting Path: d -> a -> e -> g
Visiting Path: d -> a -> b -> c
Path Found: d -> a -> b -> c

Process finished with exit code 0
```

```
Visiting Path: d -> a -> b -> c
Path Found: d -> a -> b -> c

Process finished with exit code 0
```

Searching from f -> c (no path exists)

```
There exists no path between nodes
Path is null

Process finished with exit code 0
```