// Zakery Cumbie

// CSE 464 Fall 2023

----------Project Part 3----------

NOTE: The calls to the various functions require a Graph<String, DefaultEdge> object that is provided by parseGraph(String filePath). parseGraph takes a string that represents the file path to a file with the .dot file, which should be a digraph. You should always run parseGraph and store it to a Graph<String, DefaultEdge> which can be used to test the various methods.

For the purposes of testing and displaying the inputs and output as seen below, I will be using the input2.dot file that is provided on Canvas and is required as part of the description for Random Walk Search.

Refactor Commits

Refactor 1:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/3442703498d29abf5fd4c14f7e811b5a0aba8aaa

Moved the removeNode code underneath the addNode function.

Refactor 2:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/93d46b736ab41997a3e5e4ba07986b57d4bcb9cd

Moved the removeNodes function underneath the addNodes function.

Refactor 3:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/5a7a22bd5263498dbab44fecb4991e1d0a56f2ea

Moved the removeEdge function underneath the addEdge function.

Refactor 4:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/a043d68bcda6482c6cb7e29777ad70f6e50ea22f

Moved outputGraph underneath parseGraph.

Refactor 5:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/e1f8816f7450f2173cb9e2ccd490713cd9d27b12

Moved the main function to the bottom of the main class.

Template Method Implementation

SearchAlgorithm Commit:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/959c2395541a79f673bd21efbe4aa9188b6c31d2

BFS Class Commit:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/bfe09b520e843677aa5b084eb4f4d29fa326bc87

DFS Class Commit:

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/86e772468c8174cfc53b8a4759aa7a9865b74468

```
static List<String> path1 = new ArrayList<>();

static BFS bfs = new BFS();

static DFS dfs = new DFS();


System.out.println("Template Pattern");

path1 = bfs.GraphSearch(graph, "a", "h");

System.out.println(path1.toString());


path1 = dfs.GraphSearch(graph, "a", "h");

System.out.println(path1.toString());
```

Bfs and dfs are instances of the bfs and dfs class, which are built off the template provided by SearchAlgorithm. SearchAlgorithm has the method GraphSearch, which takes a Graph<String, DefaultEdge> object, a string to represent the source node and a string to represent the destination node. GraphSearch runs a series of methods defined by the BFS and DFS classes. GraphSearch will return a List<String> object that represents the path between the source node and the destination node. The output screenshots are from the code above, which is included in the Main.java file for local testing.

```
Template Pattern
BFS: [a, e, f, h]
DFS: [a, e, g, h]
```

Strategy Pattern Implementation

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/9ca69e14c73ef4be2ad52358ba6b26d6ac136926

```
System.out.println("Strategy Pattern");

path1 = represent.GraphSearch(graph,"a", "h", BFSS);

System.out.println("BFS: " + path1.toString());


path1 = represent.GraphSearch(graph,"a", "h", DFSS);

System.out.println("DFS: " + path1.toString());
```

The strategy Pattern is implemented via an instance of the abstract class Path which is called StratRep. StratRep uses a GraphSearch method, which takes a Graph<String, DefaultEdge> object, a string representing a source node, a string representing a destination node, and an algorithm object based on the Algorithm interface by creating a new instance of either the BFSStrategy class or the DFSStrategy class. These instances are simply called BFSS and DFSS respectively.

StratRep's GraphSearch will then delegate the behavior of either BFS or DFS algorithms to BFSS or DFSS instances. These instances have their own implementations of the search function which is split into function calls much like the templates from the previous section. The search function will return a List<String> to StratRep and stored in the path and printed to the console. A sample of output from the input2.dot file provided on canvas is displayed below.

```
Strategy Pattern
BFS: [a, e, f, h]
DFS: [a, e, g, h]
```

Random Walk Search

https://github.com/Khrone99/CSE-464-2023-zcumbie/commit/d9e6493f299ce92321fb5573c014cc8cd860b0a9

The Random Walk Search is implemented through a class called randomWalk. randomWalkSearch is a method contained within and takes a Graph<String, DefaultEdge> object, a string representing the source node, and a string representing the destination node.

System.out.println();

System.out.println("Random Walk Test");

randomWalk randomGuy = new randomWalk();

path1 = randomGuy.randomWalkSearch(graph, "a", "c");

System.out.println(path1.toString());

For the purposes of this project, the test implemented in Main.java takes the input2.dot file from canvas and attempts to find the path from node “a” to node “c”, whilst printing the path it is currently on. If the randomizer takes the search down a path where a node has no children and the destination node has not been encountered, the search will begin again from the specified source node. This can result in any number of attempts that are random each time.

For the purposes of demonstrating the true randomness of this method, I will be issuing the call 5 times, and their respective outputs.

Input:

```
System.out.println();
System.out.println("Random Walk Test 1");
randomWalk randomGuy = new randomWalk();
path1 = randomGuy.randomWalkSearch(graph, src: "a", dst: "c");
System.out.println(path1.toString());

System.out.println();
System.out.println("Random Walk Test 2");
path1 = randomGuy.randomWalkSearch(graph, src: "a", dst: "c");
System.out.println(path1.toString());

System.out.println();
System.out.println("Random Walk Test 3");
path1 = randomGuy.randomWalkSearch(graph, src: "a", dst: "c");
System.out.println(path1.toString());

System.out.println();
System.out.println("Random Walk Test 4");
path1 = randomGuy.randomWalkSearch(graph, src: "a", dst: "c");
System.out.println(path1.toString());

System.out.println();
System.out.println("Random Walk Test 5");
path1 = randomGuy.randomWalkSearch(graph, src: "a", dst: "c");
System.out.println(path1.toString());

System.out.println("\n...Ending Program... \n");
```

Output:

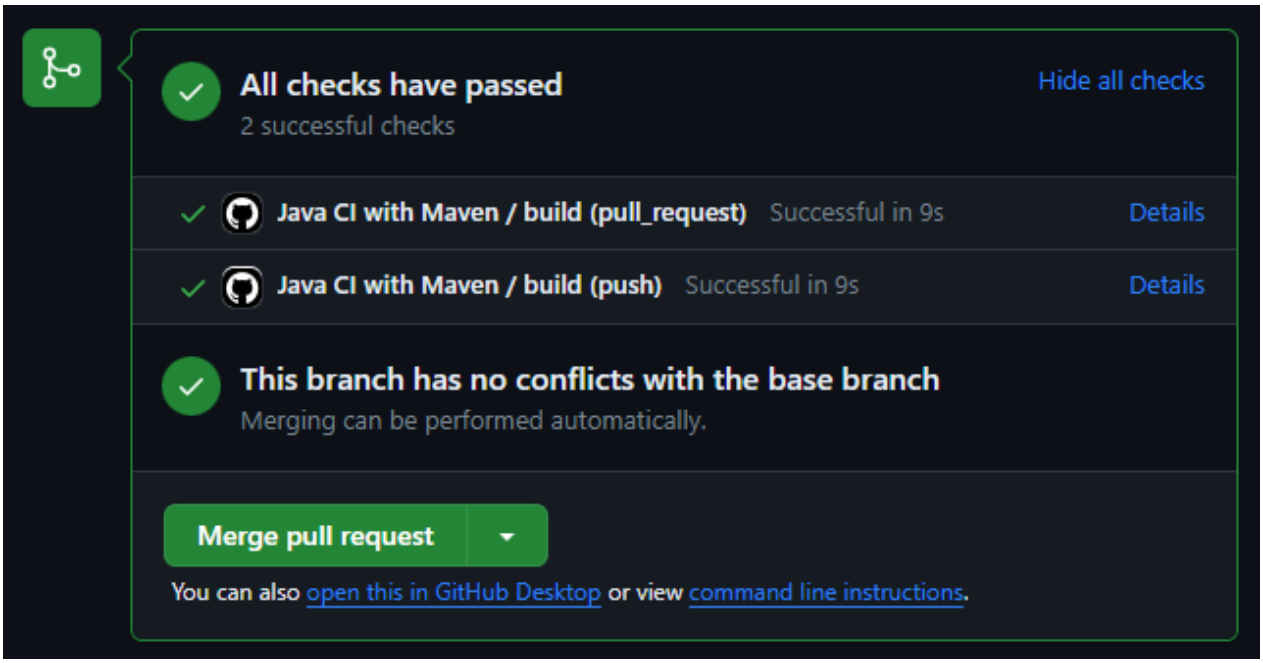
```
Random Walk Test 1
Path found: [a]
Path found: [a, e]
Path found: [a, e, g]
Path found: [a, e, g, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, e]
Path found: [a, e, g]
Path found: [a, e, g, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, e]
Path found: [a, e, g]
Path found: [a, e, g, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, e]
Path found: [a, e, f]
Path found: [a, e, f, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, e]
Path found: [a, e, f]
Path found: [a, e, f, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, e]
Path found: [a, e, g]
Path found: [a, e, g, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, b]
Path found: [a, b, c]
[a, b, c]
```

```
Random Walk Test 2
Path found: [a]
Path found: [a, e]
Path found: [a, e, f]
Path found: [a, e, f, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, b]
Path found: [a, b, c]
[a, b, c]

Random Walk Test 3
Path found: [a]
Path found: [a, b]
Path found: [a, b, c]
[a, b, c]

Random Walk Test 4
Path found: [a]
Path found: [a, e]
Path found: [a, e, f]
Path found: [a, e, f, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, e]
Path found: [a, e, g]
Path found: [a, e, g, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, b]
Path found: [a, b, c]
[a, b, c]
```

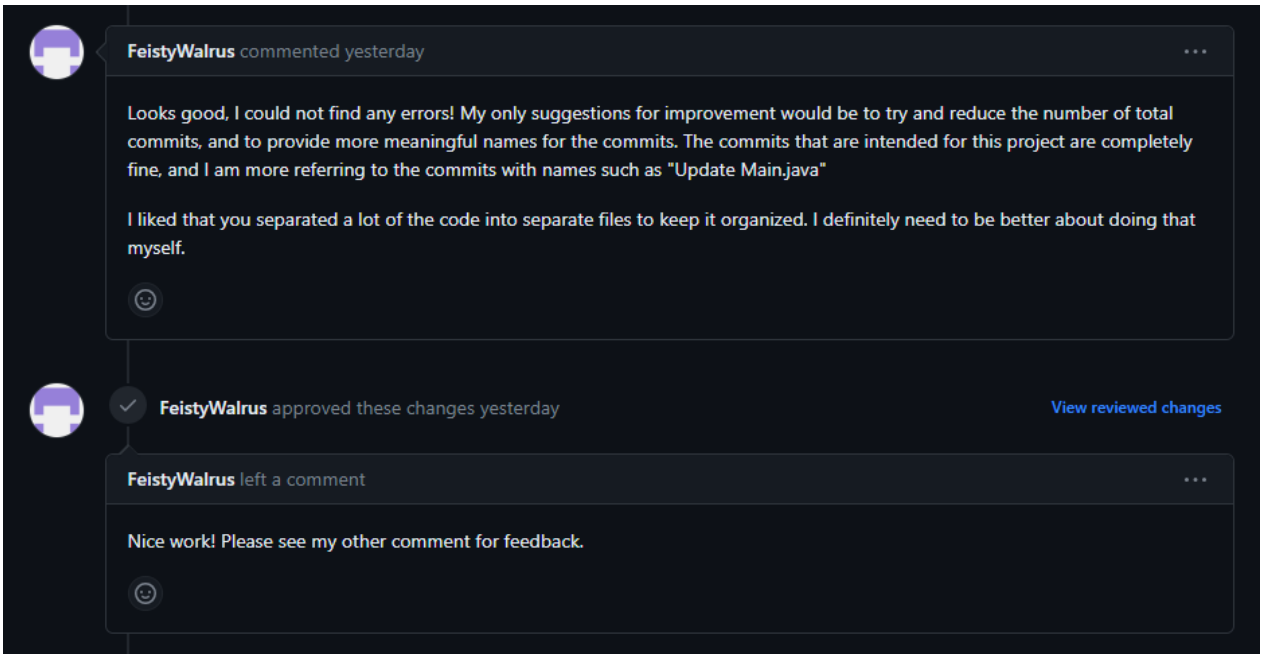
```
Random Walk Test 5
Path found: [a]
Path found: [a, e]
Path found: [a, e, f]
Path found: [a, e, f, h]
Path ended without find destination node, restarting search...
Path found: [a]
Path found: [a, b]
Path found: [a, b, c]
[a, b, c]
```

As can be seen in the screenshots above, the randomizer can be more or less efficient in finding the path at random.

Continuous Integration Checks Screenshot



A code review was done by Matthew Corey (mjcorey2@asu.edu) amd left this feedback:



I tried renaming the commits in the GitHub but there didn't seem to be any feature that allowed me to rename commits. Since it's a code review, I don't think it's very vital outside of the Refactoring commits, which I named appropriately, but I will keep this feedback in mind when using GitHub repos in the future.