

Readme

I have implemented the features of the project by making a class for graph named myGraphClass.

GitHub Link: https://github.com/Divy2000/CSE464_2023_dplate164

Run the **src/main/application/Main.java** file to access the command-line interface.
To run the test cases run **test/myGraphClass.java**

Following are the commands I have used to run my code :-

“src/main/java/testGraph.dot” -> Parses the .dot file

“0” -> To print the graph details

“1” -> To enter the details of the graph to a file

“src/main/java/testGraph.txt” -> File-path for the output

“2” -> To add node to the graph

“node0” -> Label of the node to add

“3” -> To add multiple nodes to the graph

“node1,node2,node3” -> Three nodes to be added to the graph

“4” -> To add an edge to the graph

“white” -> First node of the edge

“node3” -> Second node of the edge

“5” -> To remove a node from the graph

“yellow” -> Label of the node to remove

“6” -> To remove multiple nodes from the graph

“node1,black” -> 2 nodes to remove

“7” -> To remove an edge from the graph

“blue” -> Label of the first node of the edge to remove

“cyan” -> Label of the second node of the edge to remove

“8” -> To output the graph to .dot file

“src/main/java/testGraph_o.dot” -> Path of the output file

“9” -> To output the graph to an image

“src/main/java/testGraph.png” -> Path of the output image file

“Q” -> To exit the program

In addition to this, now we have added feature for graph search :-

“10” -> Perform BFS search or DFS search in the graph

“bfs” -> For Breadth First Search

OR

“dfs” -> For Depth First Search

Running code screenshots and commit links :-

Feature 1

- Parsing a DOT graph file

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_dplate164' open. The 'Run' tool window is active, showing the command: '/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java ...'. Below it, the terminal output shows:

```
Enter the path to the dot file
src/main/testGraph.dot
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.

Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice
0

There are 8 nodes in the graph.
The label of the nodes are :-
red
pink
green
blue
white
yellow
black
cyan
```

- Print the details of the graph

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_dplate164' open. The 'Run' tool window is active, showing the command: '/Library/Java/JavaVirtualMachines/adoptopenjdk-8.jdk/Contents/Home/bin/java ...'. Below it, the terminal output shows:

```
Enter your choice
0

There are 8 nodes in the graph.
The label of the nodes are :-
red
pink
green
blue
white
yellow
black
cyan

There are 12 edges in the graph.
The edges are :-
green--black
white--pink
white--cyan
cyan--green
yellow--red
white--yellow
pink--blue
yellow--green
pink--red
blue--black
red--black
cyan--blue
```

- Output the details to a file.

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_dpage164' open. The code editor displays `Main.java` which reads a graph from a file and prints its details. A terminal window below shows the execution of the program, choosing option 0 to print graph details to `output.txt`. The output file is successfully created.

```

no usages  ↗ Divy2000 +1
public class Main {
    no usages  ↗ Divy2000 +1
    public static void main(String[] args) {
        myGraphClass g = new myGraphClass();
        boolean running = true;
        Scanner inputScanner = new Scanner(System.in);
        System.out.println("Enter the path to the dot file");
        String graphFilePath = inputScanner.nextLine();
        try {
            g.parseGraph(graphFilePath);
        }
        catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}

```

Run: application.Main

Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice

Enter the filepath of the output file:
output.txt
Details about graph are successfully written to the file

Git Run TODO Problems Terminal Services Build Dependencies

Commit link: https://github.com/Divy2000/CSE464_2023_dpage164/commit/992acecb2f21363762b58f52b81f04d088277959

Feature 2

- Add a node to the graph

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_dpage164' open. The code editor displays `Main.java` which reads a graph from a file and prints its details. A terminal window below shows the execution of the program, choosing option 2 to add a node to the graph. A message indicates that a new node has been added.

```

# Project exclude paths
/target/
.idea/
/src/main/resources/
src/main/testGraph_o.dot
.DS_Store
/out/
/example/
/ex1.png
output.txt

```

Run: application.Main

Details about graph are successfully written to the file

Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice

Enter the label of the node you want to add:
newNode1

Externally added files can be added to Git
View Files Always Add Don't Ask Again

Git Run TODO Problems Terminal Services Build Dependencies

- Add list of nodes to the graph

```

CSE464_2023_ddate164 / .gitignore
Project Commit Pull Requests Bookmarks Structure
Run TODO Problems Terminal Services Build Dependencies
Git pom.xml (CSE464-GraphManager) Maven Notifications

Run: application.Main

Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice
3

Enter the labels of node you want to add separated by comma ('a,b,c'):
newNode2, node1beRemoved

```

Externally added files can be added to Git
View Files Always Add Don't Ask Again

Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (5 minutes ago)

- Remove a node from the graph

```

CSE464_2023_ddate164 / .gitignore
Project Commit Pull Requests Bookmarks Structure
Run TODO Problems Terminal Services Build Dependencies
Git pom.xml (CSE464-GraphManager) Maven Notifications

Run: application.Main

Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice
5

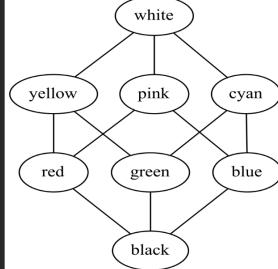
Enter the label of the node you want to remove:
node1beRemoved

```

Externally added files can be added to Git
View Files Always Add Don't Ask Again

Externally added files can be added to Git // View Files // Always Add // Don't Ask Again (6 minutes ago)

- Remove multiple node from the graph



The screenshot shows an IDE interface with a project named "CSE464_2023_dpage164". In the center is a graph visualization window displaying a network of nodes. The nodes are labeled with colors: white, yellow, pink, cyan, red, green, blue, and black. The graph structure is as follows: white is connected to yellow, pink, and cyan; yellow is connected to white, pink, and red; pink is connected to white, yellow, and green; cyan is connected to white, pink, and blue; red is connected to yellow, pink, and green; green is connected to pink, cyan, blue, and black; blue is connected to cyan and green; and black is connected to green. A terminal window below the graph shows the following interaction:

```

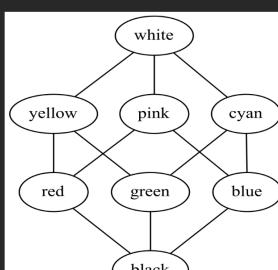
Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice
4
Enter the first/source node of the edge
newNode2, white
Choose one of the following options
  
```

A tooltip in the bottom right corner says: "Externally added files can be added to Git View Files Always Add Don't Ask Again".

Commit Link: https://github.com/Divy2000/CSE464_2023_dpage164/commit/c2e5e0723c60eed8c52ad7ac50c373458055bff

Feature 3

- Add an edge to the graph



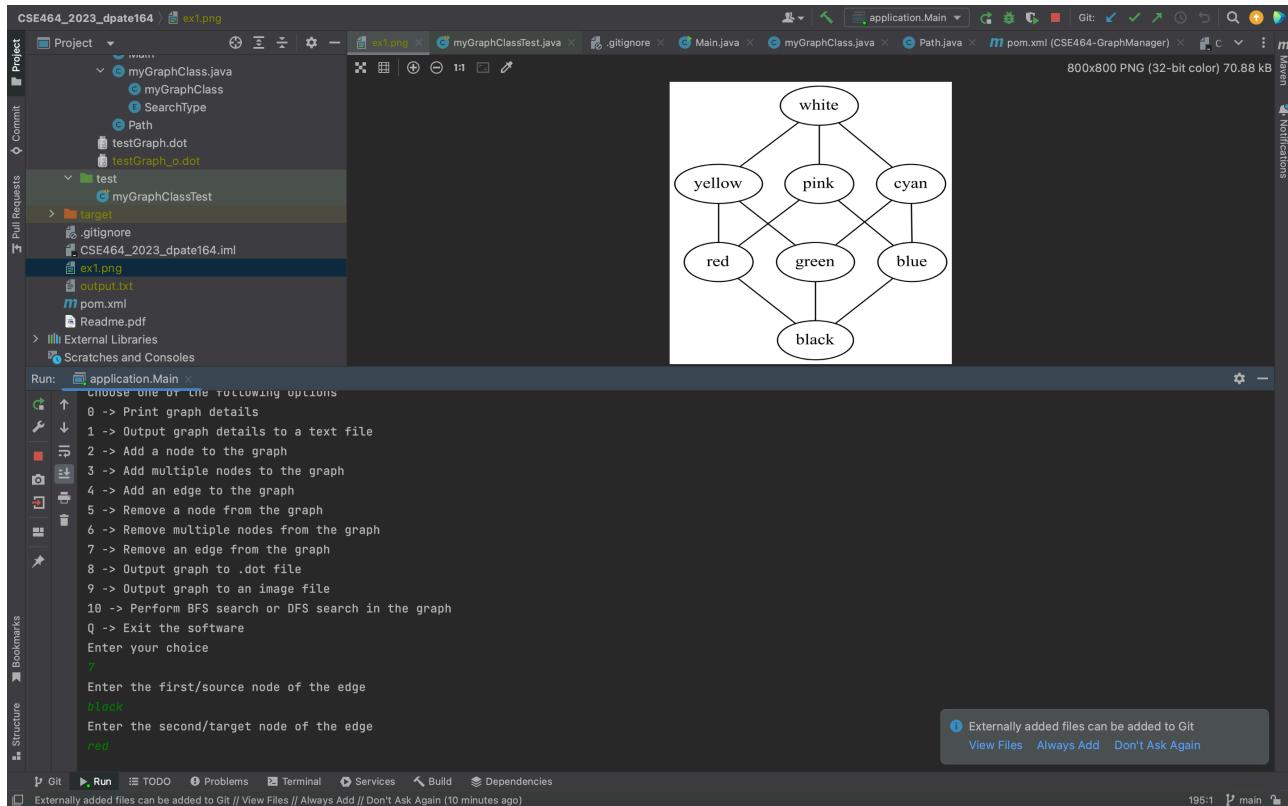
The screenshot shows the same IDE interface and graph as the previous one. The terminal window now displays:

```

Choose one of the following options
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice
4
Enter the first/source node of the edge
newNode1
Enter the second/target node of the edge
black
  
```

A tooltip in the bottom right corner says: "Externally added files can be added to Git View Files Always Add Don't Ask Again".

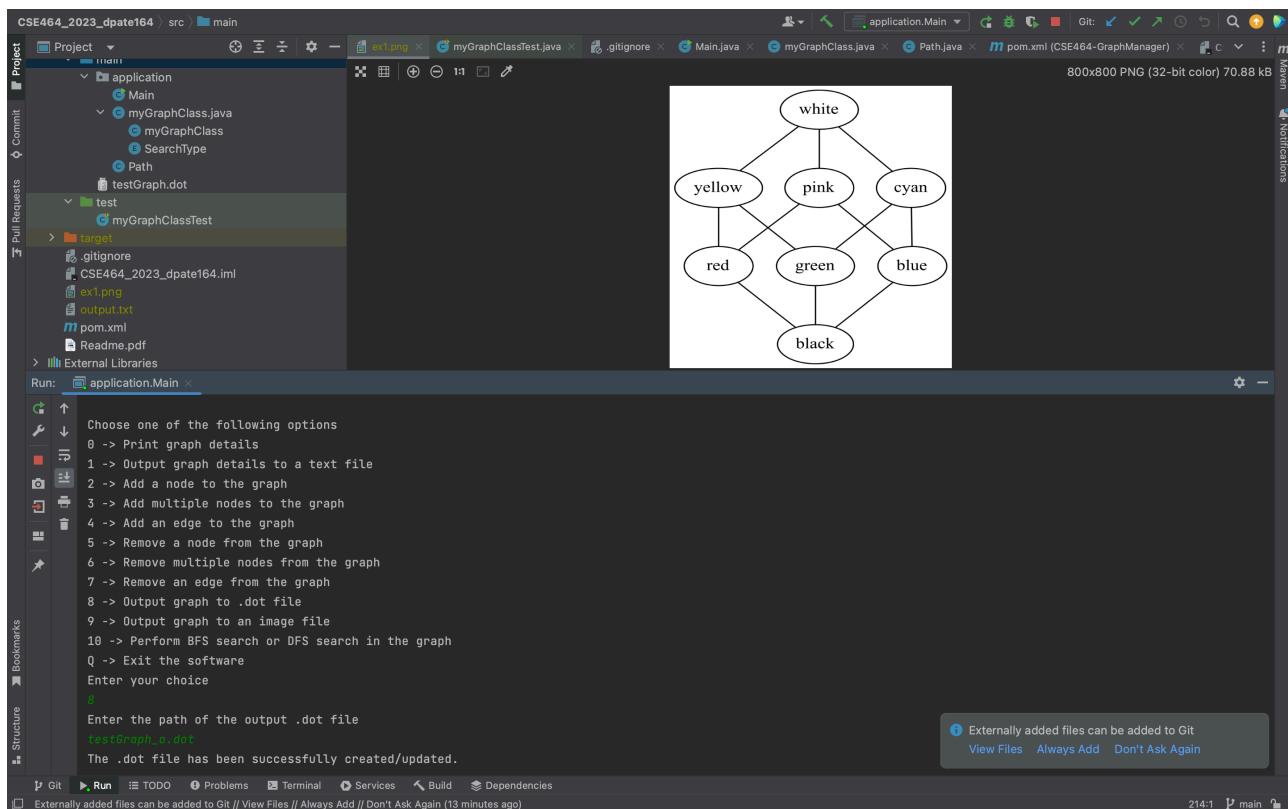
- Remove an edge from the graph



Commit Link: https://github.com/Divy2000/CSE464_2023_ddate164/commit/4fcf172714d5db0b8b19bc522782d1f3f1fa26b0

Feature 4

- Output the DOT graph file



- Output the image file

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_ddate164' open. The code editor displays `Main.java` which contains logic for handling user input and performing graph operations. Below the code editor is a terminal window showing the execution of the program. The terminal output includes a menu of options (e.g., Print graph details, Add node, Remove node, etc.) and a command to enter the path for the output image file. A tooltip in the bottom right corner provides information about externally added files.

```

case "6":
    System.out.println("Enter the labels of node you want to remove separated by comma ('a,b,c'):");
    nodeLabels = inputScanner.nextLine();
    nodeLabelList = nodeLabels.split( regex: "," );
    g.removeNodes(nodeLabelList);
    break;
case "7":
    System.out.println("Enter the first/source node of the edge");
    node1 = inputScanner.nextLine();
    System.out.println("Enter the second/target node of the edge");
    node2 = inputScanner.nextLine();
    g.removeEdge(node1, node2);
    break;
case "8":
    System.out.println("Enter the path of the output .dot file");
    filepath = inputScanner.nextLine();
}

Run: application.Main
0 -> Print graph details
1 -> Output graph details to a text file
2 -> Add a node to the graph
3 -> Add multiple nodes to the graph
4 -> Add an edge to the graph
5 -> Remove a node from the graph
6 -> Remove multiple nodes from the graph
7 -> Remove an edge from the graph
8 -> Output graph to .dot file
9 -> Output graph to an image file
10 -> Perform BFS search or DFS search in the graph
Q -> Exit the software
Enter your choice
9
Enter path to output the image of the final graph
output.png
Warning: the fonts "Times" and "Lucida Bright" are not available for the Java logical font "Serif", which may have unex
Warning: the fonts "Times" and "Lucida Bright" are not available for the Java logical font "Serif", which may have unex

Externally added files can be added to Git View Files Always Add Don't Ask Again

```

Commit Link: https://github.com/Divy2000/CSE464_2023_ddate164/commit/f36ef90b4ae229af14958bf57da1d9c8179503c5

Final commit of finishing the testing portion: https://github.com/Divy2000/CSE464_2023_ddate164/commit/82b8581aa5068c9804c18beb58abe7cc01e21766

Final commit for adding maven support: https://github.com/Divy2000/CSE464_2023_ddate164/commit/8c7363bda391c40d0865358f37e1ee66b59063dc

Final commit for completing the continuous integration: https://github.com/Divy2000/CSE464_2023_ddate164/commit/669e2d62a7a94fbc0a038a561ab0c1f0aa705246

BFS Branch: https://github.com/Divy2000/CSE464_2023_ddate164/tree/bfs

DFS Branch: https://github.com/Divy2000/CSE464_2023_ddate164/tree/dfs

Adding BFS implementation: https://github.com/Divy2000/CSE464_2023_ddate164/commit/c65d58291562086c2838ff501920aae9e405d5bf

Adding DFS implementation: https://github.com/Divy2000/CSE464_2023_ddate164/commit/b6c9a76f85e7a4d1f695e29629fe0a0943fe39b5

Merge branch BFS to main: https://github.com/Divy2000/CSE464_2023_ddate164/commit/119d64372419fbe526ec6756e19c5ca274fc16e6

Merge branch DFS to main: https://github.com/Divy2000/CSE464_2023_ddate164/commit/0192df91cf231a6e8ce38f5ef6508d49a49af2bc

- BFS search example

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_dpage164' open. The Main.java file is the active editor, displaying code for a graph search program. The code includes a switch statement for cases 6 through 8, which handle node removal, edge addition/removal, and path output respectively. A tooltip at the bottom right indicates externally added files can be added to Git.

```

case "6":
    System.out.println("Enter the labels of node you want to remove separated by comma ('a,b,c')");
    nodeLabelList = nodeLabels.split( regex: "," );
    g.removeNodes(nodeLabelList);
    break;
case "7":
    System.out.println("Enter the first/source node of the edge");
    node1 = inputScanner.nextLine();
    System.out.println("Enter the second/target node of the edge");
    node2 = inputScanner.nextLine();
    g.addEdge(node1, node2);
    break;
case "8":
    System.out.println("Enter the path of the output .dot file");
    filepath = inputScanner.nextLine();

```

- DFS search example

The screenshot shows the IntelliJ IDEA interface with the project 'CSE464_2023_dpage164' open. The Main.java file is the active editor, displaying code for a graph search program. The code includes a switch statement for cases 6 through 8, which handle node removal, edge addition/removal, and path output respectively. A tooltip at the bottom right indicates externally added files can be added to Git.

```

case "6":
    System.out.println("Enter the labels of node you want to remove separated by comma ('a,b,c')");
    nodeLabelList = nodeLabels.split( regex: "," );
    g.removeNodes(nodeLabelList);
    break;
case "7":
    System.out.println("Enter the first/source node of the edge");
    node1 = inputScanner.nextLine();
    System.out.println("Enter the second/target node of the edge");
    node2 = inputScanner.nextLine();
    g.addEdge(node1, node2);
    break;
case "8":
    System.out.println("Enter the path of the output .dot file");
    filepath = inputScanner.nextLine();

```

Project Part 3

Part 1 - Code Refactoring

1st Commit

Extract Method Refactoring: https://github.com/Divy2000/CSE464_2023_dplate164/commit/df43dd44ff8433b66ca8657331e6dc46126e20eb

2nd Commit

Inline Refactoring: https://github.com/Divy2000/CSE464_2023_dplate164/commit/d3f43769b68d95cb1adb44e9cc0315068fc2512b

3rd Commit

Renaming Variables: https://github.com/Divy2000/CSE464_2023_dplate164/commit/da24acdaf5b617fd184807aed35d445859a9f898

4th Commit

Replaced control flag with return statement: https://github.com/Divy2000/CSE464_2023_dplate164/commit/20f9b45535ba22629360444e9d89ee9c0f0cb905

5th Commit

Removing the dead code: https://github.com/Divy2000/CSE464_2023_dplate164/commit/9d85ac57609bcbfa33e74323049b1a3b75d63fd8

Part 2 - Template Design Pattern

- In implementing template design pattern, I have extracted the common steps in “**src/main/application/GraphSearch.java**”.
- Both BFS and DFS have only one difference, BFS uses Queue while DFS uses stack.
- Therefor I have implemented everything in GraphSearch.java except the part where we add and remove from the queue/stack and the part where we check if queue/stack is empty.
- I have made 3 different abstract methods for that part :-
 - addToDataStructure
 - removeFromDataStructure
 - isEmpty (**Note: I changed the method name to isEmpty in next commit**)
- In BFS class I declare a Queue and implement abstract methods according to that.
- In DFS class I declare a Stack and implement abstract methods according to that.

Commit Link: https://github.com/Divy2000/CSE464_2023_dplate164/commit/a79a63ad9249bf99742825b2c7acda88e04ec72f

Part 3 - Strategy Design Pattern

- For this implementation I have created an abstract class GraphSearch_Stragey, which takes MutableGraph and GraphSearch_interface as its input to constructor.
- GraphSearch_interface have 3 methods, addToDataStructure, removeFromDataStructure and isEmpty.
- Classes BFS and DFS implements these methods.
- The GraphSearch class extends the abstract class of GraphSearch_Stragey and calls the super constructor.
- In GraphSearch method in muGraphClass, I am creating objects of the type of search and passing them to constructor of GraphSearch class.

Commit link: https://github.com/Divy2000/CSE464_2023_dplate164/commit/e746659a58d4015f3e24ef76d6fa79e427f70aab

Part 4 - Random walk

- For this I have taken the common steps of the previous strategy pattern and that of the random walk and applied it accordingly.
- The BFS and DFS had the same iteration with only difference of the data structure, but random walk had different type of iterations, so I tried to see what's different in the iteration and put them separate keeping other steps common.
- To print the path for each of the visited node in random walk and implement the behaviour of looping till not finding the path, I used if else statement in the GraphSearch function.
- I again added the BFS and DFS Template implementation, which I removed while implementing the strategy pattern.

Commit Link: https://github.com/Divy2000/CSE464_2023_dpage164/commit/d7e93ca544637b9c1528202ade4cb161e1ccd803