Project 3

Sunanda Tummala

Part 1 : Refactor branch:

Commit 1: I have utilized the “Simiplifying methods Calls Refactoring” as removesNodes was directly calling the nodes.clear() without checking, so I have added a method removeNode(String node) which should be call and checks if node exists then remove and generate the msg for it**.**
Link: **https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/f0f568c3cc30b91100288001db402b76c682c842**

```
commit f0f568c3cc30b91100288001db402b76c682c842 (HEAD -> refactor)
Author: sunanda2004 <stumma10@asu.edu>
Date:   Fri Apr 18 07:19:33 2025 -0700

    update removeNodes method to restructure and call by removeNode method

commit dff028973f36a1d648d307ca427a05d617546211 (origin/main, origin/HEAD, main)
Author: sunanda2004 <stumma10@asu.edu>
Date:   Mon Mar 17 15:24:55 2025 -0700

    Update README2.pdf for project2 details
```

Figure 1: git status first commit in refactor branch

Commit 2: I have utilized the “Preparatory Refactoring” by adding the toString method for Path class and removing similar code by doing “Composing Method refactoring” in displayMethod, by just calling the getPath method and print it according to it.

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/b6af8d75770f13db62a86f3b6982051e857d094c

```
commit b6af8d75770f13db62a86f3b6982051e857d094c (HEAD -> refactor)
Author: sunanda2004 <stumma10@asu.edu>
Date:   Fri Apr 18 08:17:21 2025 -0700

    added toString method and update displayMethod
```

Figure 2: git status second commit in refactor branch

Commit 3: I have restructured the code by doing “Composing Method refactoring” to call GraphSearch by dividing the code into individual function for making code clearer for understanding.

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/8db3a340a485d14989628633524dfbc994dd51b5

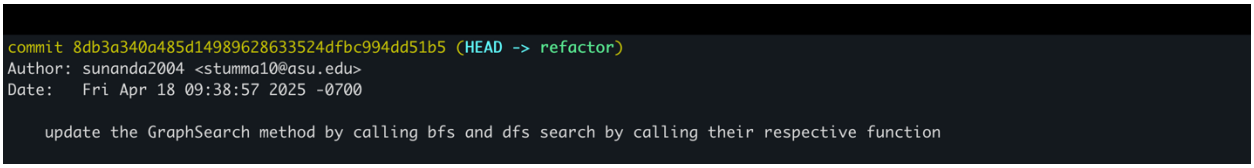

Figure 3: git status of third commit in refactor branch

Commit 4: I have utilized the "Moving Features Between Objects refactoring" by creating another class MyGraphIO which handles the read and write operation for the graph from dot file and also able to generate the graphics (png, jpg) file.

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/69e5e04477d1c4e0d9906ff64f8d3c95cf52bdb2

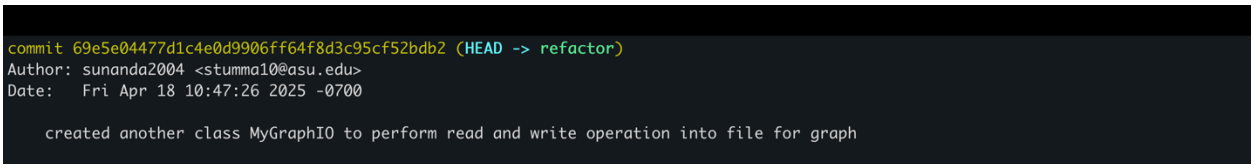

Figure 4: : git status for fourth commit into refactor branch

Commit 5: I have utilized the "Moving Features Between Objects refactoring" **by** creating MyGraph Class. I have moved defaultGraph variables into MyGraph which has function as addNode, addNodes, removeNode, removeNodes, toString, importGraph, exportGraph, containNode, getEdgeSource, getEdgeTarget and getEdgeSet from the Graph. Main Project class is MyGraphApp class, which will be used for calling for handling graph function call by MyGraph class object graph IO calls by MyGraphIO class object and handles search by calling bfsSearch and dfsSearch function.

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/b2bedd50a518da6fd9c6fe9683cc5e83383f3db1

Figure 5: git status for fifth commit in refactor branch

Part 2 : Template Pattern in Refactor branch:

Commit: I have created a template abstract class as SearchTemplate in which I have defined a searchPath as abstract method.

```
**********************************************************************
public abstract class SearchTemplate {

    //methods to be implemented by bfs, dfs and random path search
        public abstract Path searchPath(MyGraph graph, String srcLabel, String dstLabel);

}
**********************************************************************
```

After that I have created the BFSSearch subclass which extends the searchPath function by implementing the BFS search algorithm.

```
**********************************************************************
public class BFSSearch extends SearchTemplate {

  /**
   * searchPath method : it search for the path from srcLabel
   * to dstLabel by following the breadth for search algorithm
   */
  @Override
  public Path searchPath(MyGraph graph, String srcLabel, String dstLabel){

    // declaring path p as null
    Path p =  null;

    // declaring the visited nodes
    Set<String> visited = new HashSet<>();

    // declaring the map/dict to hold the parent -> connecting path
    HashMap<String, String> parent = new HashMap<>();

    // declaring the queue to hold unexplored nodes
    Queue<String> queue = new LinkedList<>();

    // Initializing the queue, visited nodes and parents dictionary for path
    queue.add(srcLabel);
    parent.put(srcLabel,null);
    visited.add(srcLabel);

    //Running the loop until queue becomes empty
    while (!queue.isEmpty()) {
```

```
        // removing the front element
        String currentNode = queue.remove();

        //System.out.println("Exploring Node: " + currentNode);

        // Checks if currentNode is the destination node
        if (currentNode.equals(dstLabel)) {
            p = new Path();
            while (currentNode != null) {
                p.addNode(currentNode);
                currentNode = parent.get(currentNode);
            }
            //System.out.println(p.getPath());
            return p;
        }

        for(DefaultEdge e : graph.getEdgeSet() ) {
            String source = graph.getEdgeSource(e);
            String neighbor  = graph.getEdgeTarget(e);
            if (source.equals(currentNode)) {
                // System.out.println(source + "=>" + neighbor);
                if (! visited.contains(neighbor)) {
                    visited.add(neighbor);
                    queue.add(neighbor);
                    parent.put(neighbor, currentNode);
                }
            }
        }
    }
    return p;
  }
}
**********************************************************************
```

Similarly I have implemented DFSSearch subclass which extends the SearchTemplate class searchPath method.

```
**********************************************************************
public class DFSSearch extends SearchTemplate {

  /**
   * searchPath method : it search for the path from srcLabel
   * to dstLabel by following the depth for search algorithm
   */
  @Override
  public Path searchPath(MyGraph graph, String srcLabel, String dstLabel){

    // declaring path p as null
    Path p =  null;

    // declaring the visited nodes
    Set<String> visited = new HashSet<>();
```

```
    // declaring the map/dict to hold the parent -> connecting path
    HashMap<String, String> parent = new HashMap<>();

    // declaring the stack to hold unexplored nodes
    Stack<String> stack = new Stack<>();

    // Initializing the stack, visited nodes and parents dictionary for path
    stack.push(srcLabel);
    parent.put(srcLabel,null);

    //Running the loop until stack becomes empty
    while (!stack.isEmpty()) {

      // removing the top element
      String currentNode = stack.pop();

      visited.add(currentNode);

      // Checks if currentNode is the destination node
      //System.out.println("Exploring Node: " + currentNode);
      if (currentNode.equals(dstLabel)) {
        p = new Path();
        while (currentNode != null) {
          p.addNode(currentNode);
          currentNode = parent.get(currentNode);
        }
        return p;
      }

      for(DefaultEdge e : graph.getEdgeSet() ) {
        String source = graph.getEdgeSource(e);
        String neighbor  = graph.getEdgeTarget(e);
        if (source.equals(currentNode)) {
          //System.out.println(source + "=>" + neighbor );
          if (! visited.contains(neighbor)) {
            stack.push(neighbor);
            parent.put(neighbor, currentNode);
          }
        }
      }
    }
    return p;
  }
}
```

****

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/45124625a07fbef118c4c7a104cd2171fe40ac6e

```
commit 45124625a07fbef118c4c7a104cd2171fe40ac6e (HEAD -> refactor, origin/refactor)
Author: sunanda2004 <stumma10@asu.edu>
Date:   Fri Apr 18 14:44:45 2025 -0700

    added SearchTemplate abstract class and BFSSearch, DSFSSearch subclass to complete the template Pattern
```

Figure 6: git commit for Template Pattern in refactor branch

Part 3 : Strategy Pattern in Refactor branch:

Commit: In the strategy pattern, I have created a SearchStrategy interface which has the method name searchPath which should be implement by subsequent classes which utilized this strategy.

****

```
public interface SearchStrategy {
        public Path searchPath();
}
```

****

In the next for BFS Search I have created a class named, "BFSSearchStrategy" in which I have implemented this searchPath function which runs the BFS algorithm.

****

```
public class BFSSearchStrategy implements SearchStrategy {

   private String srcLabel = "";
   private String dstLabel = "";
   private MyGraph graph = null;

   public BFSSearchStrategy(MyGraph graph, String srcLabel, String dstLabel ){
      this.srcLabel = srcLabel;
      this.dstLabel = dstLabel;
      this.graph = graph;
   }

   /**
    * searchPath method : it search for the path from srcLabel
    * to dstLabel by following the breadth for search algorithm
    */
   @Override
   public Path searchPath() {
      // declaring path p as null
      Path p =  null;

      // declaring the visited nodes
      Set<String> visited = new HashSet<>();

      // declaring the map/dict to hold the parent -> connecting path
      HashMap<String, String> parent = new HashMap<>();

      // declaring the queue to hold unexplored nodes
```

```
    Queue<String> queue = new LinkedList<>();

    // Initializing the queue, visited nodes and parents dictionary for path
    queue.add(srcLabel);
    parent.put(srcLabel,null);
    visited.add(srcLabel);

    //Running the loop until queue becomes empty
    while (!queue.isEmpty()) {

      // removing the front element
      String currentNode = queue.remove();

      //System.out.println("Exploring Node: " + currentNode);

      // Checks if currentNode is the destination node
      if (currentNode.equals(dstLabel)) {
        p = new Path();
        while (currentNode != null) {
          p.addNode(currentNode);
          currentNode = parent.get(currentNode);
        }
        //System.out.println(p.getPath());
        return p;
      }

      for(DefaultEdge e : graph.getEdgeSet() ) {
        String source = graph.getEdgeSource(e);
        String neighbor  = graph.getEdgeTarget(e);
        if (source.equals(currentNode)) {
          // System.out.println(source + "=>" + neighbor);
          if (! visited.contains(neighbor)) {
            visited.add(neighbor);
            queue.add(neighbor);
            parent.put(neighbor, currentNode);
          }
        }
      }
    }
    return p;
  }

}
```

****

Similarly, for DFS Search I have written the DFSSearchStrategy Class which implements the searchPath by utilizing the DFS Algorithm.

****

```
import java.io.*;
import java.util.*;
```

```
import org.jgrapht.graph.*;

public class DFSSearchStrategy implements SearchStrategy {

   private String srcLabel = "";
   private String dstLabel = "";
   private MyGraph graph = null;

   public DFSSearchStrategy(MyGraph graph, String srcLabel, String dstLabel ){
      this.srcLabel = srcLabel;
      this.dstLabel = dstLabel;
      this.graph = graph;
   }

   /**
    * searchPath method : it search for the path from srcLabel
    * to dstLabel by following the depth for search algorithm
    */
   @Override
   public Path searchPath() {

      // declaring path p as null
      Path p =  null;

      // declaring the visited nodes
      Set<String> visited = new HashSet<>();

      // declaring the map/dict to hold the parent -> connecting path
      HashMap<String, String> parent = new HashMap<>();

      // declaring the stack to hold unexplored nodes
      Stack<String> stack = new Stack<>();

      // Initializing the stack, visited nodes and parents dictionary for path
      stack.push(srcLabel);
      parent.put(srcLabel,null);

      //Running the loop until stack becomes empty
      while (!stack.isEmpty()) {

         // removing the top element
         String currentNode = stack.pop();

         visited.add(currentNode);

         // Checks if currentNode is the destination node
         //System.out.println("Exploring Node: " + currentNode);
         if (currentNode.equals(dstLabel)) {
            p = new Path();
            while (currentNode != null) {
               p.addNode(currentNode);
               currentNode = parent.get(currentNode);
            }
```

```
            return p;
        }

        for(DefaultEdge e : graph.getEdgeSet() ) {
            String source = graph.getEdgeSource(e);
            String neighbor  = graph.getEdgeTarget(e);
            if (source.equals(currentNode)) {
                //System.out.println(source + "=>" + neighbor );
                if (! visited.contains(neighbor)) {
                    stack.push(neighbor);
                    parent.put(neighbor, currentNode);
                }
            }
        }
      }
      return p;
   }
}
```

****

For Strategy Pattern to work, I have made few changes into MyGraphApp class to call the method to work properly. I have written search method which will be call on run time object of strategy which can be call at run time according to user input for algorithms.

****

```
// search method is written which can be call on run time strategy object
public Path search(SearchStrategy searchMethod) {
      return searchMethod.searchPath();
   }

if (algorithm.equals("bfs"))  {
            return search(new BFSSearchStrategy(graph, srcLabel, dstLabel));
 }
 else if (algorithm.equals("dfs")) {
            return search(new DFSSearchStrategy(graph, srcLabel, dstLabel));
}
```

****

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/34a6accabaf88ab0f64906e459be8820a93e8ba5

Figure 7: git status for strategy pattern commit in refactor branch

Part 4 : Refactor branch on Random Walk:

Commit: In the strategy pattern, I have created a SearchStrategy interface which has the method name searchPath which should be implement by subsequent classes which utilized this strategy.

****

```
public interface SearchStrategy {
        public Path searchPath();
}
```

****

In the next for Random walk Search I have created a class named, “RWSSearchStrategy” in which I have implemented this searchPath function which runs the Random walk algorithm.

****

```
public class RWSSearchStrategy implements SearchStrategy {

  private String srcLabel = "";
  private String dstLabel = "";
  private MyGraph graph = null;

  public RWSSearchStrategy(MyGraph graph, String srcLabel, String dstLabel ){
    this.srcLabel = srcLabel;
    this.dstLabel = dstLabel;
    this.graph = graph;
  }

  public String getRandomElement(List<String> l) {
    Random r = new Random();
    return l.get(r.nextInt(l.size()));
  }

  /**
   * searchPath method : it search for the path from srcLabel
   * to dstLabel by following the depth for search algorithm
   */
  @Override
  public Path searchPath() {

    // declaring path p as null
    Path p =  new Path();

    // declaring the visited edges
    Set<DefaultEdge> visitedEdges = new HashSet<>();

    // declaring the node which is fully explored
    Set<String> exploredVertices = new HashSet<>();

    // Initializing the queue, visited nodes and parents dictionary for path
    String currentNode = srcLabel;
```

```
// adding current node into the path
p.addNode(currentNode);

// Printing the random testing
System.out.println("random testing");

//Running the loop until currentNode != dstLabel
while (!currentNode.equals(dstLabel)) {

   //System.out.println(currentNode);

   // Get all the neighbors of current Node
   List<String> neighbors = graph.getSuccessorNeighbors(currentNode);


   List<String> unvisitedVertices = new ArrayList<>();
   for (String v : neighbors) {
      DefaultEdge e = graph.getEdge(currentNode, v);
      if (!visitedEdges.contains(e)) {
         unvisitedVertices.add(v);
      }
   }

   if (unvisitedVertices.size() > 0 ) {
      String nextNode = getRandomElement(unvisitedVertices);
      visitedEdges.add(graph.getEdge(currentNode, nextNode));
      if (exploredVertices.contains(nextNode)) {
         if (!p.isEmpty()){
            currentNode = p.getLastNode();
            p.removeLastNode();
         }
      }
      else {
         currentNode = nextNode;
         p.addNodeInLast(currentNode);
         System.out.println("Visiting" + p.getPath());
      }
   }
   else {
      exploredVertices.add(p.getLastNode());
      if (p.getSize() == 1 ) {
         return null;
      }
      p.removeLastNode();
      currentNode =  p.getLastNode();
      while (true) {
         neighbors = graph.getSuccessorNeighbors(currentNode);
         unvisitedVertices.clear();
         for (String v : neighbors) {
            DefaultEdge e = graph.getEdge(currentNode, v);
            if (!visitedEdges.contains(e)) {
               unvisitedVertices.add(v);
            }
```

```
            }
            if (unvisitedVertices.size() == 0 ) {
               exploredVertices.add(p.getLastNode());
               if (p.getSize() == 1 ) {
                  return null;
               }
               p.removeLastNode();
               currentNode = p.getLastNode();
            }
            else {
               break;
            }
         }
      }
    }
    return p;
  }
}
```

****

Commit:

```
commit 713583580184c91c0313049cbe579ea3cf53915f
Author: sunanda2004 <stumma10@asu.edu>
Date:   Sun Apr 27 01:02:07 2025 -0700

    Random Walk Search by Strategy Techniques
```

Figure 8: Random Walk Search Strategy Pattern

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/713583580184c91c0313049cbe579ea3cf53915f

In the next for Random walk Search I have created a class named, “RWSSearch” in which I have implemented this searchPath function which runs the Random walk algorithm.

****

```
public class RWSSearch extends SearchTemplate {

  /**
   * searchPath method : it search for the path from srcLabel
   * to dstLabel by following the breadth for search algorithm
   */
  public String getRandomElement(List<String> l) {
     Random r = new Random();
     return l.get(r.nextInt(l.size()));
  }

  @Override
  public Path searchPath(MyGraph graph, String srcLabel, String dstLabel){
```

```
// declaring path p as null
Path p =  new Path();

// declaring the visited edges
Set<DefaultEdge> visitedEdges = new HashSet<>();

// declaring the node which is fully explored
Set<String> exploredVertices = new HashSet<>();

// Initializing the queue, visited nodes and parents dictionary for path
String currentNode = srcLabel;

// adding current node into the path
p.addNode(currentNode);

// Printing the random testing
System.out.println("random testing");

//Running the loop until currentNode != dstLabel
while (!currentNode.equals(dstLabel)) {

   //System.out.println(currentNode);

   // Get all the neighbors of current Node
   List<String> neighbors = graph.getSuccessorNeighbors(currentNode);


   List<String> unvisitedVertices = new ArrayList<>();
   for (String v : neighbors) {
      DefaultEdge e = graph.getEdge(currentNode, v);
      if (!visitedEdges.contains(e)) {
         unvisitedVertices.add(v);
      }
   }

   if (unvisitedVertices.size() > 0 ) {
      String nextNode = getRandomElement(unvisitedVertices);
      visitedEdges.add(graph.getEdge(currentNode, nextNode));
      if (exploredVertices.contains(nextNode)) {
         if (!p.isEmpty()){
            currentNode = p.getLastNode();
            p.removeLastNode();
         }
      }
      else {
         currentNode = nextNode;
         p.addNodeInLast(currentNode);
         System.out.println("Visiting" + p.getPath());
      }
   }
   else {
      exploredVertices.add(p.getLastNode());
      if (p.getSize() == 1 ) {
         return null;
      }
      p.removeLastNode();
      currentNode =  p.getLastNode();
```

```
        while (true) {
            neighbors = graph.getSuccessorNeighbors(currentNode);
            unvisitedVertices.clear();
            for (String v : neighbors) {
                DefaultEdge e = graph.getEdge(currentNode, v);
                if (!visitedEdges.contains(e)) {
                    unvisitedVertices.add(v);
                }
            }
            if (unvisitedVertices.size() == 0 ) {
                exploredVertices.add(p.getLastNode());
                if (p.getSize() == 1 ) {
                    return null;
                }
                p.removeLastNode();
                currentNode = p.getLastNode();
            }
            else {
                break;
            }
        }
      }
    }
    return p;
  }
}
**********************************************************************
```

Commit:

```
commit d59bce576046958305b7594b0f4711003e752f06
Author: sunanda2004 <stumma10@asu.edu>
Date:   Sun Apr 27 00:54:49 2025 -0700

    Random Walk in completed and updated with test cases for Template Version
```

Figure 9: Random Walk Search Template Pattern

Link: https://github.com/sunanda2004/CSE-464-2025-stumma10/commit/d59bce576046958305b7594b0f4711003e752f06