```java
import javax.swing.plaf.nimbus.NimbusLookAndFeel;
import java.util.*;

public class graph {

    public int[][] matrix;
    public Node[] nodes;
    public int size;
    public int count=0;
    public List<Integer>[] adj;


        graph(){

        }
        public graph(int size){
            this.size=size;
            matrix= new int[size][size];
            nodes = new Node[size];
            adj = new ArrayList[size];
            for (int i = 0; i < size; i++) {
                adj[i] = new ArrayList<>();
            }
        }
        public void addNode(Node node){
                nodes[count++] = node;

        }
        public void addEdge(int src, int dst){
            matrix[src][dst]=1;
            matrix[dst][src]=1;
            adj[src].add(dst);
            adj[dst].add(src);
        }
        public boolean checkEdge(int src, int dst){
            if(matrix[src][dst]==1){
                return true;
            }
            else{
                return false;
            }
```

```
42          }
43  }
44      class Node{
45          String location;
46          int index;
47          Node(String location, int index){
48              this.location=location;
49              this.index=index;
50          }
51          public String getLocation(){
52              return location;
53          }
54          public int getIndex(){
55              return index;
56          }
57      }
58
59
```

```java
import java.util.*;


public class traversal {
    graph graph;

    public traversal(graph graph) {
        this.graph = graph;
    }

    // BFS
    public void bfs(int start) {
        boolean[] visited = new boolean[graph.size];
        Queue<Integer> queue = new LinkedList<>();
        visited[start] = true;
        queue.add(start);

        System.out.print("Breadth First Search: ");
        System.out.println(" ");
        while (!queue.isEmpty()) {
            int node = queue.poll();
            System.out.print(graph.nodes[node].
  getLocation() + " ");

            for (int neighbor : graph.adj[node]) {
                if (!visited[neighbor]) {
                    visited[neighbor] = true;
                    queue.add(neighbor);
                }
            }
        }
        System.out.println();
    }

    public void dfs(int start) {
        boolean[] visited = new boolean[graph.size];
        System.out.print("Depth First Search: ");
        System.out.println(" ");
        dfsRecursive(start, visited);
        System.out.println();
    }
```

```
41
42      private void dfsRecursive(int node, boolean[]
    visited) {
43          visited[node] = true;
44          System.out.print(graph.nodes[node].
    getLocation() + " ");
45
46          for (int neighbor : graph.adj[node]) {
47              if (!visited[neighbor]) {
48                  dfsRecursive(neighbor, visited);
49              }
50          }
51      }
52
53      public static void main(String[] args) {
54          graph graph = new graph(7);
55          graph.addNode(new Node("UC", 0)); //0
56          graph.addNode(new Node("GC", 1)); //1
57          graph.addNode(new Node("CC", 2)); //2
58          graph.addNode(new Node("DC", 3)); //3
59          graph.addNode(new Node("OC", 4)); //4
60          graph.addNode(new Node("SC", 5)); //5
61          graph.addNode(new Node("PC", 6)); //6
62
63          graph.addEdge(0, 1);
64          graph.addEdge(0, 2);
65          graph.addEdge(0, 3);
66          graph.addEdge(0, 4);
67          graph.addEdge(0, 5);
68          graph.addEdge(1, 2);
69          graph.addEdge(2, 3);
70          graph.addEdge(3, 4);
71          graph.addEdge(3, 6);
72          graph.addEdge(4, 6);
73          graph.addEdge(5, 4);
74
75          traversal t = new traversal(graph);
76
77          System.out.println("The search results are as
     follows:");
78          t.bfs(0);
```

```
79          t.dfs(0);
80
81
82      }
83  }
84
```