Assignment 1

1. Problem statement

1.1. Write an algorithm to check if a node, say A, is good to be the first node to traverse to all the nodes in the network.

1.2. Improve your solution to list all the nodes for the given use case, that are good to be initiator node for CL algorithm.

2. Source Code

>>Main_Frame.java

```
package chandi_lamport;

import java.util.Scanner;
import java.awt.FileDialog;
import java.awt.Frame;

public class Main_Frame {
   public static void main(String[] args) {

      // Choose graph file
      String filename = openFileDialog();
      if (filename == null) {
         System.out.println("No file selected.");
         return;
      }
      Graph graph = new Graph(filename);
      // Display Adjacency Matrix
      graph.printAdjacencyMatrix();
      Scanner scanner = new Scanner(System.in);

      int option;

      do {
         // Display menu
         System.out.println();
         System.out.println("Menu:");
         System.out.println("1. Find initiator with specified name");
         System.out.println("2. Find initiator without specifying name");
         System.out.println("0. Exit");
         System.out.print("Enter your choice: ");

         option = scanner.nextInt();
         scanner.nextLine(); // Consume newline character

         switch (option) {
                case 1:
                    System.out.print("Enter initiator name: ");
                    String initiatorName = scanner.nextLine();
                    graph.initiator(initiatorName);
                    break;
                case 2:
                    graph.initiator();
                    break;
                case 0:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid option. Please choose again.");
                    break;
                }
            } while (option != 0);

            scanner.close();
            return;

        }

        private static String openFileDialog() {
            FileDialog fd = new FileDialog((Frame) null, "Open", FileDialog.LOAD);
            fd.setFilenameFilter((dir, name) -> name.endsWith(".graph"));
            fd.setVisible(true);
            String filename = fd.getFile();
            // Validate file extension
            if (!filename.endsWith(".graph")) {
                System.out.println("Error: The selected file is not a .graph file.");
                filename = null;
            }
            if (filename != null) {
                return fd.getDirectory() + filename;
            }
            return null;
        }
    }
```

>> *Graph.java*

```
package chandi_lamport;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Graph {

//  Store the vertices from the input file
   private final List<String> vertices = new ArrayList<>();
//  Maps the input vertices with an index
   private final Map<String, Integer> vertexIndex = new HashMap<>();
   private int[][] adjacencyMatrix;
//  Store the edges from the input file
   private List<String> edges = new ArrayList<>();

   public Graph(String filename) {

      edges = readEdgesFromFile(filename);

      if (edges == null) {
         System.out.println("Error reading edges from file.");
         return;
      }

      // Identify unique vertices
      identifyVertices(edges);

      // Initialize adjacency matrix

      adjacencyMatrix = new int[vertices.size()][vertices.size()];
      // Populate adjacency matrix
      populateAdjacencyMatrix(edges, adjacencyMatrix);
   }

//  Read the edges from the file as input
   private static List<String> readEdgesFromFile(String filename) {
      List<String> edges = new ArrayList<>();
      try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
         String line;
         while ((line = br.readLine()) != null) {
            edges.add(line.trim());
         }
      } catch (IOException e) {
         e.printStackTrace();
         return null;
      }
      return edges;

   }

//  Maps the input vertices name with an index
   private void identifyVertices(List<String> edges) {
      for (String edge : edges) {
            String[] verticesInEdge = edge.split(",");
            for (String vertex : verticesInEdge) {
                if (!vertexIndex.containsKey(vertex)) {
                    vertexIndex.put(vertex, vertices.size());
                    vertices.add(vertex);
                }
            }
        }
    }

//Create adjacency matrix
    private void populateAdjacencyMatrix(List<String> edges, int[][] adjacencyMatrix) {
        for (String edge : edges) {
            String[] verticesInEdge = edge.split(",");
            int i = vertexIndex.get(verticesInEdge[0]);
            int j = vertexIndex.get(verticesInEdge[1]);
            adjacencyMatrix[i][j] = 1;
        }
    }

//  Print the created adjacency matrix
    public void printAdjacencyMatrix() {
        System.out.println("Adjacency matrix of given graph is : ");
        System.out.print("  ");
        for (String vertex : vertices) {
            System.out.print(vertex + " ");
        }
        System.out.println();

        for (int i = 0; i < adjacencyMatrix.length; i++) {
            System.out.print(vertices.get(i) + " ");
            for (int j = 0; j < adjacencyMatrix[i].length; j++) {
                System.out.print(adjacencyMatrix[i][j] + " ");
            }
            System.out.println();
        }
        System.out.println();
    }

//  Perform BFS on the given graph
    private Boolean BFS(String ini) {
        if (!vertexIndex.containsKey(ini)) {
            System.out.println(ini + " is not a valid vertex.");
            return false;
        }
        // Number of vertices in the graph
        int noOfNode = vertices.size();
        boolean[] visited = new boolean[noOfNode];
        Arrays.fill(visited, false);
        List<Integer> q = new ArrayList<>();
        int iniIndex = vertexIndex.get(ini);
        q.add(iniIndex);

        // Set source as visited
        visited[iniIndex] = true;

        int vis;
        while (!q.isEmpty()) {
            vis = q.get(0);
            // Dequeue
            q.remove(0);

            // For every adjacent vertex to the current vertex
            for (int i = 0; i < noOfNode; i++) {
                if (adjacencyMatrix[vis][i] == 1 && !visited[i]) {
                    // Enqueue the adjacent node to the queue
                    q.add(i);
                    visited[i] = true;
                }
            }
        }
        for (int i = 0; i < noOfNode; i++) {
            if (!visited[i]) {
                // If the given vertex is not an initiator return false
                return false;
            }
        }
        // If the given vertex is an initiator return true
        return true;
    }

// Check a given vertex is initiator or not
    public void initiator(String ini) {
        if (BFS(ini)) {
            System.out.println(ini + " is an initiator");
        } else
            System.out.println(ini + " is not an initiator");
    }

// List all initiator vertices
    public void initiator() {
        System.out.println("Initiators are: ");
        for (String ini : vertices) {
            if (BFS(ini)) {
                System.out.println(ini + " is an initiator");
            }
        }
    }
}
```

3. Pre-requisites & Assumptions

1) **Graph File Format**:

   It assumes that the input file containing graph edges is formatted correctly, with each line representing an edge between two vertices separated by a comma. The assumption includes consistency in formatting and absence of syntax errors within the file.

   <vertex_name1> , <vertex_name2>

2) **Error Handling**:

   The code assumes limited error scenarios, primarily focused on file I/O operations. It assumes that file reading operations will encounter no unexpected issues beyond standard IO Exceptions, neglecting potential errors such as missing files, insufficient permissions, or corrupted data. File name must contain ".graph" extension. <filename>.graph

3) **Graph Representation**:

   It assumes that the graph is relatively small or sparse enough to justify the memory usage of an adjacency matrix.

4) **Vertex Identification**:

   The program assumes that each vertex label is unique and identifiable by a string. It doesn't handle scenarios involving duplicate vertex labels or non-string vertex identifiers, presuming consistent labeling throughout the file.

5) **Initiator Definition**:

   The concept of an "initiator" is assumed to represent vertices from which a Breadth-First Search (BFS) traversal can reach all other vertices in the graph.

6) **User Interface**:

   It assumes users are accustomed to command-line interfaces and will provide valid inputs as prompted by the program's menu. The assumption includes user compliance with menu options and expected data formats, without considering potential input errors or misuse.

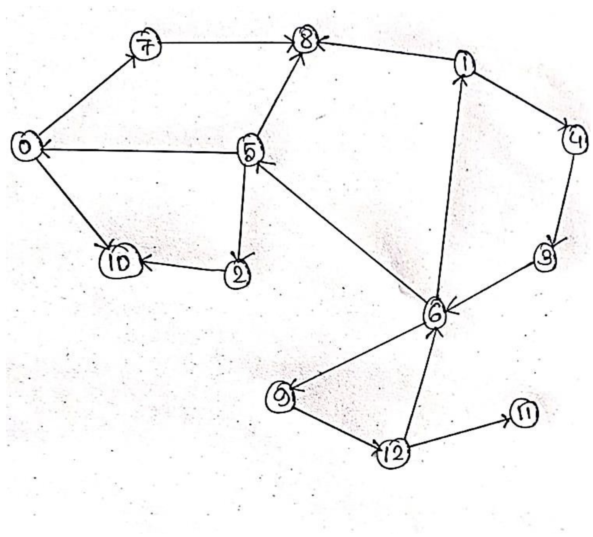
7) **Platform Dependency**:

   The program's reliance on AWT components for file dialogs assumes execution within desktop environments supporting AWT libraries. It may not accommodate platforms where AWT is unavailable or impractical, such as web or mobile environments.

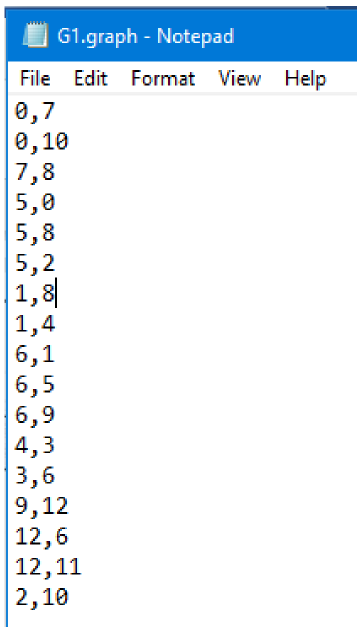
4. Results

4.1. Graph 1

(1) Visual Representation of Graph

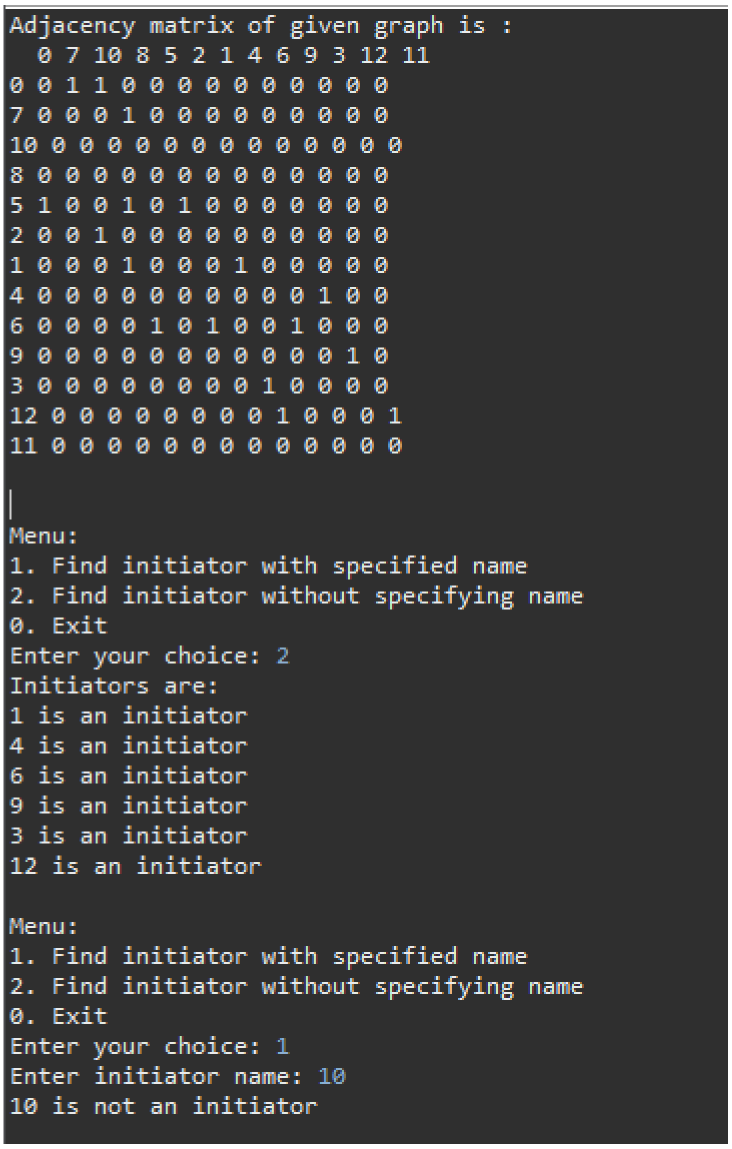


(2) Graph Input



```
0,7
0,10
7,8
5,0
5,8
5,2
1,8
1,4
6,1
6,5
6,9
4,3
3,6
9,12
12,6
12,11
2,10
```

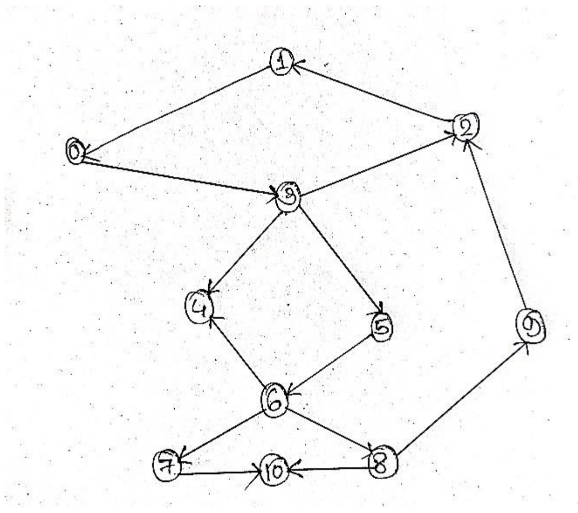
(3) Result



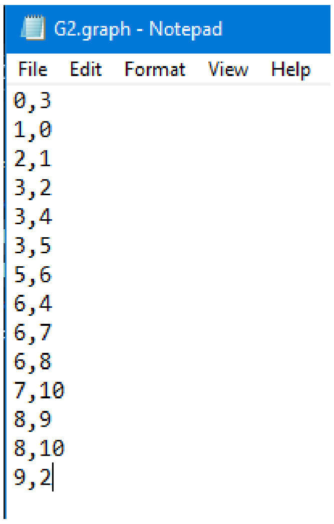
```
Adjacency matrix of given graph is :
  0 7 10 8 5 2 1 4 6 9 3 12 11
0 0 1 1 0 0 0 0 0 0 0 0 0 0
7 0 0 0 1 0 0 0 0 0 0 0 0 0
10 0 0 0 0 0 0 0 0 0 0 0 0 0
8 0 0 0 0 0 0 0 0 0 0 0 0 0
5 1 0 0 1 0 1 0 0 0 0 0 0 0
2 0 0 1 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0 1 0 0
6 0 0 0 0 1 0 1 0 0 1 0 0 0
9 0 0 0 0 0 0 0 0 0 0 0 1 0
3 0 0 0 0 0 0 0 0 1 0 0 0 0
12 0 0 0 0 0 0 0 0 1 0 0 0 1
11 0 0 0 0 0 0 0 0 0 0 0 0 0

|
Menu:
1. Find initiator with specified name
2. Find initiator without specifying name
0. Exit
Enter your choice: 2
Initiators are:
1 is an initiator
4 is an initiator
6 is an initiator
9 is an initiator
3 is an initiator
12 is an initiator

Menu:
1. Find initiator with specified name
2. Find initiator without specifying name
0. Exit
Enter your choice: 1
Enter initiator name: 10
10 is not an initiator
```

4.2. Graph 2

(1) Visual Representation of Graph

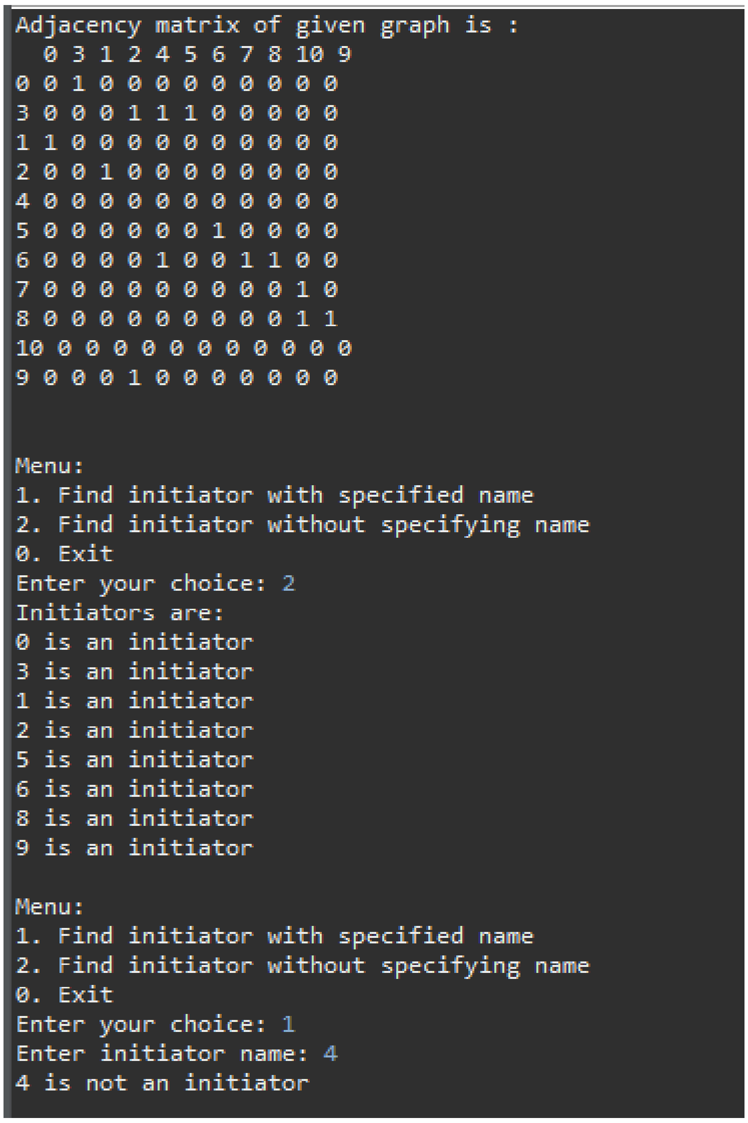


(2) Graph Input



```
0,3
1,0
2,1
3,2
3,4
3,5
5,6
6,4
6,7
6,8
7,10
8,9
8,10
9,2
```

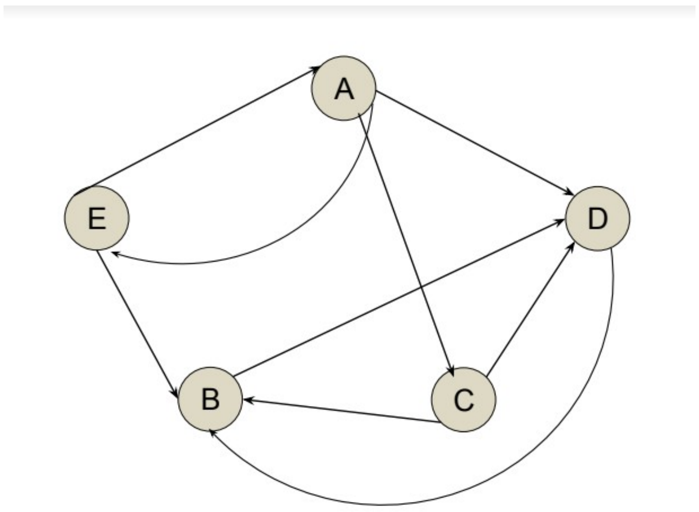
(3) Result



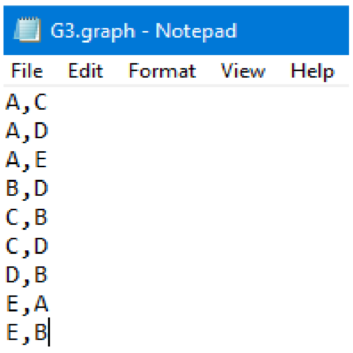
```
Adjacency matrix of given graph is :
  0 3 1 2 4 5 6 7 8 10 9
0 0 1 0 0 0 0 0 0 0 0 0
3 0 0 0 1 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0 0 0 0 0
2 0 0 1 0 0 0 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 1 0 0 0 0
6 0 0 0 0 1 0 0 1 1 0 0
7 0 0 0 0 0 0 0 0 0 1 0
8 0 0 0 0 0 0 0 0 0 1 1
10 0 0 0 0 0 0 0 0 0 0 0
9 0 0 0 1 0 0 0 0 0 0 0


Menu:
1. Find initiator with specified name
2. Find initiator without specifying name
0. Exit
Enter your choice: 2
Initiators are:
0 is an initiator
3 is an initiator
1 is an initiator
2 is an initiator
5 is an initiator
6 is an initiator
8 is an initiator
9 is an initiator

Menu:
1. Find initiator with specified name
2. Find initiator without specifying name
0. Exit
Enter your choice: 1
Enter initiator name: 4
4 is not an initiator
```

4.3.Graph 3

(1) Visual Representation of Graph

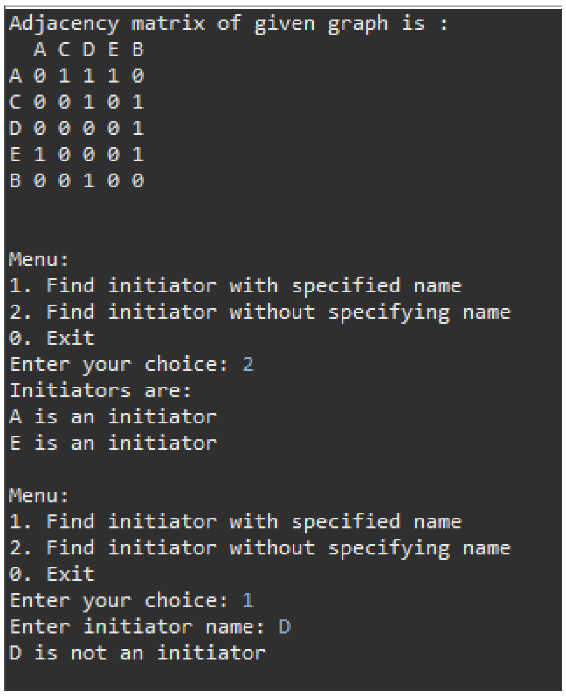


(2) Graph Input



```
A,C
A,D
A,E
B,D
C,B
C,D
D,B
E,A
E,B
```

(3) Result



```
Adjacency matrix of given graph is :
  A C D E B
A 0 1 1 1 0
C 0 0 1 0 1
D 0 0 0 0 1
E 1 0 0 0 1
B 0 0 1 0 0


Menu:
1. Find initiator with specified name
2. Find initiator without specifying name
0. Exit
Enter your choice: 2
Initiators are:
A is an initiator
E is an initiator

Menu:
1. Find initiator with specified name
2. Find initiator without specifying name
0. Exit
Enter your choice: 1
Enter initiator name: D
D is not an initiator
```

5. Remarks

5.1. Error Handling Improvement:

Implement more comprehensive error handling to handle various exceptional scenarios, such as missing or inaccessible files, malformed input data, or unexpected runtime errors. Provide detailed error messages to assist users in troubleshooting issues.

5.2. Graph Representation Optimization:

Consider alternative graph representations (e.g., adjacency lists) to reduce memory consumption, especially for large or sparse graphs. Choose a representation that balances memory efficiency with the required operations for the specific application.

5.3. Input Validation Enhancement:

Strengthen input validation to ensure robustness against unexpected user inputs. Validate user input formats, handle edge cases gracefully, and provide clear feedback to guide users in providing correct inputs.

5.4. Code Modularization:

Further modularize the codebase to enhance maintainability and code reuse. Identify cohesive modules and refactor the code into smaller, reusable components with well-defined responsibilities, adhering to principles such as separation of concerns and single responsibility.