Requirements specification:

Given a general signal flow graph, write a program to calculate the overall transfer function using Mason Formula.

Main data structures:

1- Arrays
2- ArrayLists

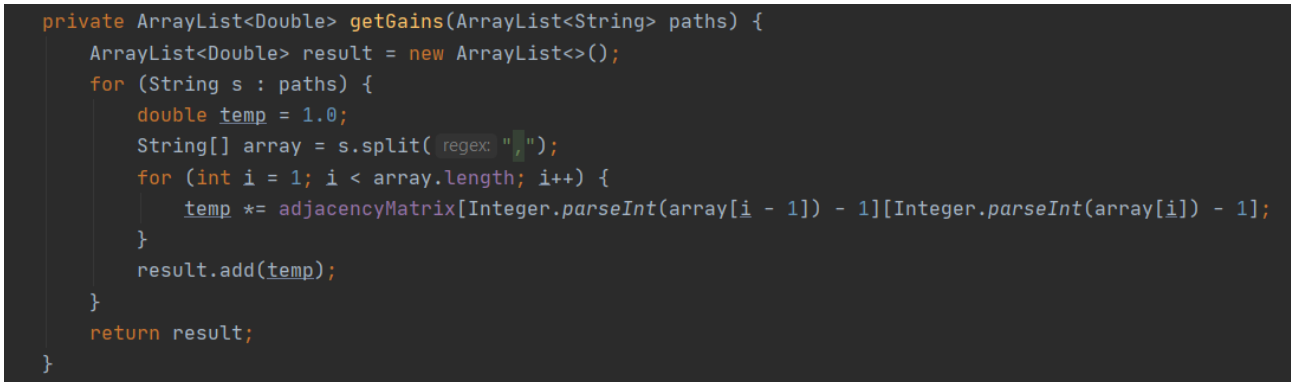
Algorithms description:

1- Saving all edges in 2D array, its lengths are the number of vertices.
2- Using DFS over edges to get all forward paths and loops and get their gains.
3- Saving all loops in 2D array, its lengths are the number of loops.
4- Using DFS over loops to get all non-touching loops.
5- Calculating the determinant, cofactors of the forward paths and the total transfer function using mason formula.

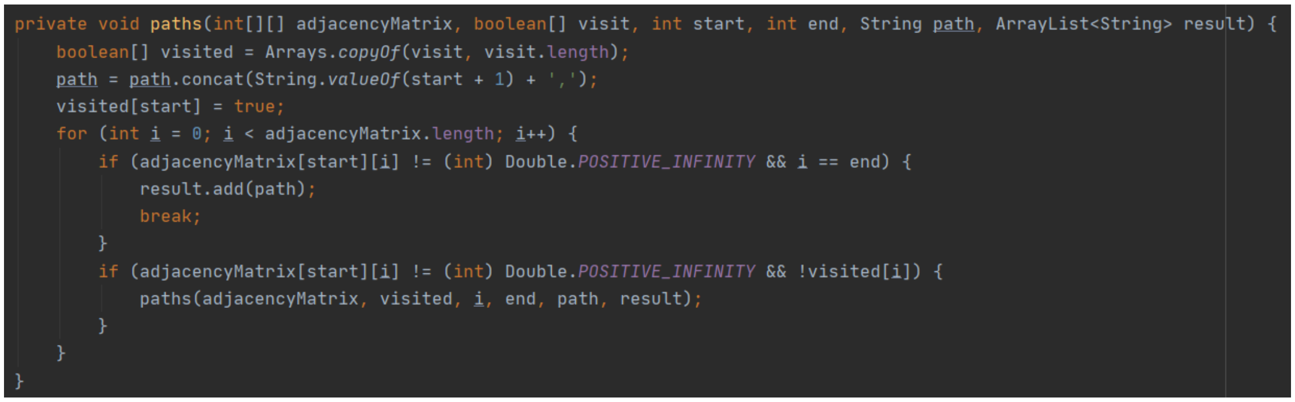
Assumptions:

1- The vertices names are numbers from 1 to the numbers of vertices.
2- The weight of the edges are integers.
3- It cannot be more than one edge between two vertices.

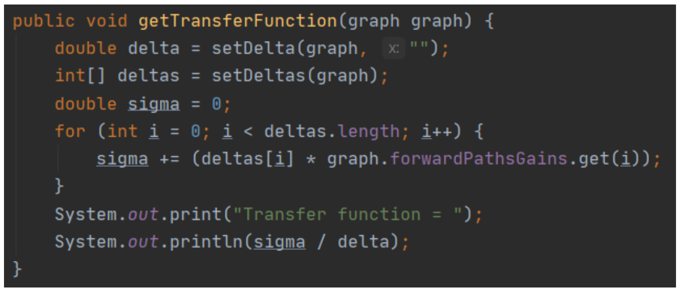
Code Snippets:



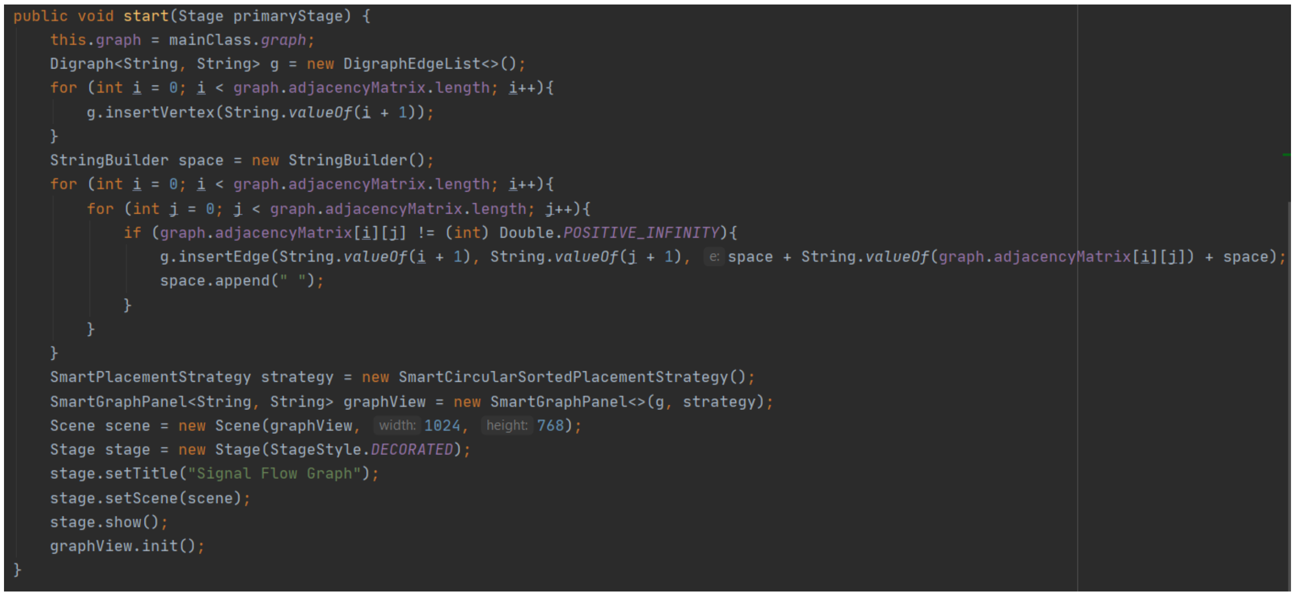
```java
private ArrayList<Double> getGains(ArrayList<String> paths) {
    ArrayList<Double> result = new ArrayList<>();
    for (String s : paths) {
        double temp = 1.0;
        String[] array = s.split( regex: ",");
        for (int i = 1; i < array.length; i++) {
            temp *= adjacencyMatrix[Integer.parseInt(array[i - 1]) - 1][Integer.parseInt(array[i]) - 1];
        }
        result.add(temp);
    }
    return result;
}
```



```java
private void paths(int[][] adjacencyMatrix, boolean[] visit, int start, int end, String path, ArrayList<String> result) {
    boolean[] visited = Arrays.copyOf(visit, visit.length);
    path = path.concat(String.valueOf(start + 1) + ',');
    visited[start] = true;
    for (int i = 0; i < adjacencyMatrix.length; i++) {
        if (adjacencyMatrix[start][i] != (int) Double.POSITIVE_INFINITY && i == end) {
            result.add(path);
            break;
        }
        if (adjacencyMatrix[start][i] != (int) Double.POSITIVE_INFINITY && !visited[i]) {
            paths(adjacencyMatrix, visited, i, end, path, result);
        }
    }
}
```



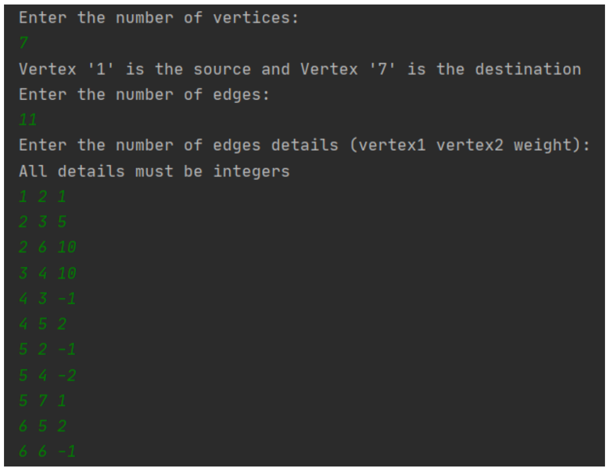
```java
public void getTransferFunction(graph graph) {
    double delta = setDelta(graph,  x: "");
    int[] deltas = setDeltas(graph);
    double sigma = 0;
    for (int i = 0; i < deltas.length; i++) {
        sigma += (deltas[i] * graph.forwardPathsGains.get(i));
    }
    System.out.print("Transfer function = ");
    System.out.println(sigma / delta);
}
```



```java
public void start(Stage primaryStage) {
    this.graph = mainClass.graph;
    Digraph<String, String> g = new DigraphEdgeList<>();
    for (int i = 0; i < graph.adjacencyMatrix.length; i++){
        g.insertVertex(String.valueOf(i + 1));
    }
    StringBuilder space = new StringBuilder();
    for (int i = 0; i < graph.adjacencyMatrix.length; i++){
        for (int j = 0; j < graph.adjacencyMatrix.length; j++){
            if (graph.adjacencyMatrix[i][j] != (int) Double.POSITIVE_INFINITY){
                g.insertEdge(String.valueOf(i + 1), String.valueOf(j + 1),  e: space + String.valueOf(graph.adjacencyMatrix[i][j]) + space);
                space.append(" ");
            }
        }
    }
    SmartPlacementStrategy strategy = new SmartCircularSortedPlacementStrategy();
    SmartGraphPanel<String, String> graphView = new SmartGraphPanel<>(g, strategy);
    Scene scene = new Scene(graphView,  width: 1024,  height: 768);
    Stage stage = new Stage(StageStyle.DECORATED);
    stage.setTitle("Signal Flow Graph");
    stage.setScene(scene);
    stage.show();
    graphView.init();
}
```

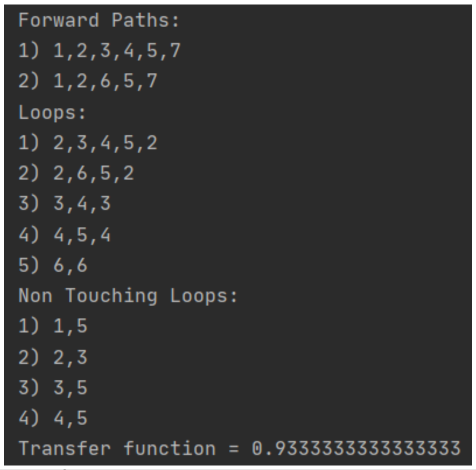
Sample Runs:

1- Input:

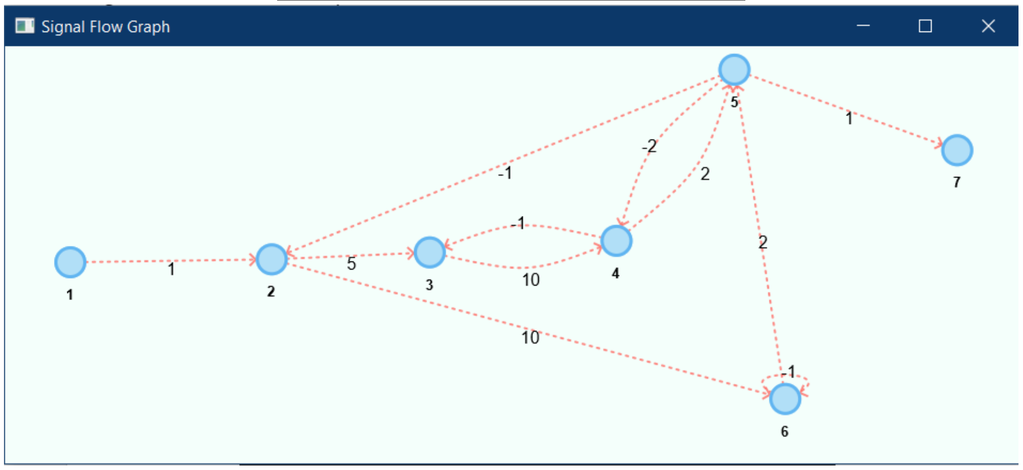


```
Enter the number of vertices:
7
Vertex '1' is the source and Vertex '7' is the destination
Enter the number of edges:
11
Enter the number of edges details (vertex1 vertex2 weight):
All details must be integers
1 2 1
2 3 5
2 6 10
3 4 10
4 3 -1
4 5 2
5 2 -1
5 4 -2
5 7 1
6 5 2
6 6 -1
```

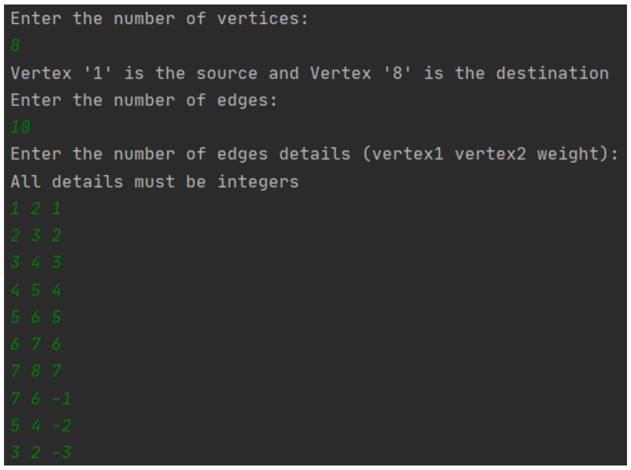
Output:



```
Forward Paths:
1) 1,2,3,4,5,7
2) 1,2,6,5,7
Loops:
1) 2,3,4,5,2
2) 2,6,5,2
3) 3,4,3
4) 4,5,4
5) 6,6
Non Touching Loops:
1) 1,5
2) 2,3
3) 3,5
4) 4,5
Transfer function = 0.9333333333333333
```

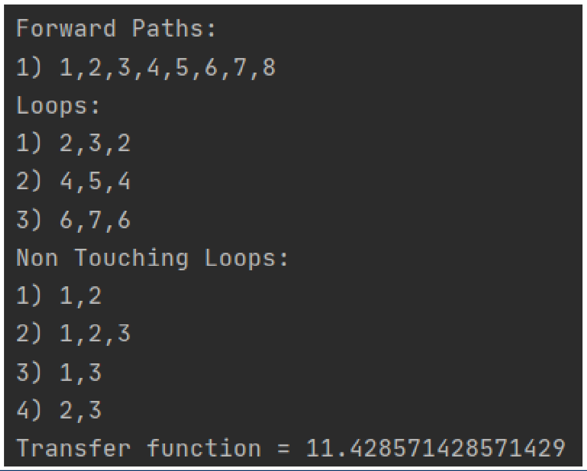


2- Input:

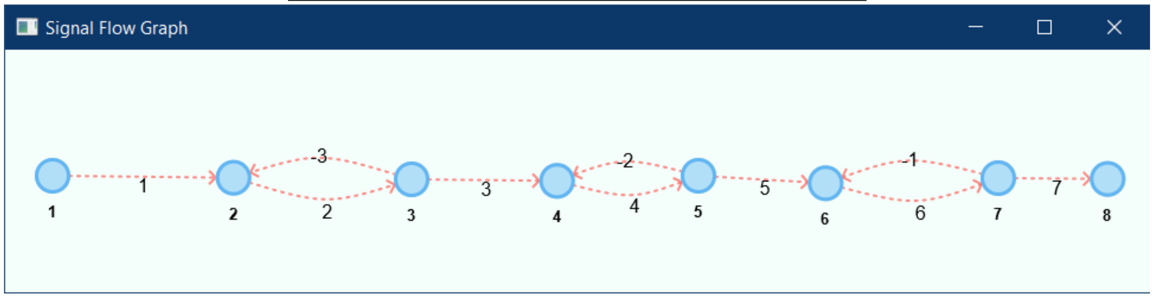


```
Enter the number of vertices:
8
Vertex '1' is the source and Vertex '8' is the destination
Enter the number of edges:
10
Enter the number of edges details (vertex1 vertex2 weight):
All details must be integers
1 2 1
2 3 2
3 4 3
4 5 4
5 6 5
6 7 6
7 8 7
7 6 -1
5 4 -2
3 2 -3
```

Output:



```
Forward Paths:
1) 1,2,3,4,5,6,7,8
Loops:
1) 2,3,2
2) 4,5,4
3) 6,7,6
Non Touching Loops:
1) 1,2
2) 1,2,3
3) 1,3
4) 2,3
Transfer function = 11.428571428571429
```



Project Link:

https://github.com/wzattout/mason-formula