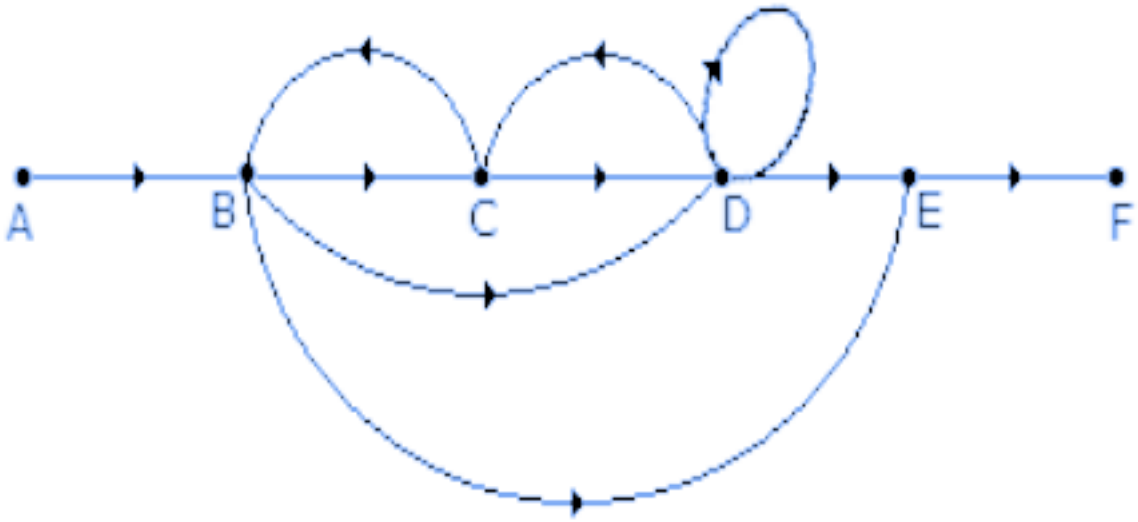


Single Flow Graph

Calculate overall Transfer Function Program.

Name: Linh Ahmed Mohamed Salah Eldin

ID: 50

Source code:

https://github.com/linhahmed/Single-Flow-Graph

Requirements

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

Description

The relation between an input variable and an output variable of a signal flow graph is given by Mason's Gain Formula. For determination of the overall system.

Main Features

The User can add the number of nodes and the starting and ending node then the graph will be shown in the right panel. Then adds an edge between two Nodes , While clicking the Results button, a window will be shown printing out All Forward Paths , All Loops , All information for each Delta and the Overall Transfer function.

Data Structure Used

Representing the graph with a class to build it taking the total number of Nodes from the user.

```
public Graph(int NumOfNodes) {
    Nodes = new ArrayList<Node>();
    allEdges = new ArrayList<Edge>();
    forwardPaths = new ArrayList<Integer[]>();
    Loops = new ArrayList<List<Integer>>();
    nonTouchingLoops = new ArrayList<List<List<List<Integer>>>>();
    nonTouchingLoopsGain = new ArrayList<List<String>>();
    this.NumOfNodes = NumOfNodes;
    visited = new HashSet<>();
    pointStack = new LinkedList<>();
    markedStack = new LinkedList<>();
    markedSet = new HashSet<>();
}
```

And then adding all nodes.

```
public boolean addNodes() {
    if (NumOfNodes == 1) {
        return false;
    }
    for (int i = 0; i < NumOfNodes; i++) {
        Node node = new Node(i + 1);
        Nodes.add(node);
    }
    return true;
}
```

Having a method to add an edge between two nodes with its weight.

```
public boolean addEdge(int from, int to, double gain) {
    if (Nodes.size() >= from && Nodes.size() >= to) {
        Edge g = Nodes.get(from - 1).addEdge(to, gain);
        allEdges.add(g);
        return true;
    }
    return false;
}
```

Using ArrayLists and arrays in the main Algorithms to save Paths and Loops and so on:

ArrayList<Node> Nodes → to hold all nodes.

ArrayList<Integer[]> forwardPaths → to hold all forward paths.

List<List<Integer>> Loops → to hold all loops.

List<List<List<List<Integer>>>> nonTouchingLoops → to hold all non touching loops.

Design

*Designing the program as **MVC Model**:*

1. *Model : which contains the edge and node classes that holds the edges and nodes given by the user*

```java
package Model;
import java.util.ArrayList;
public class Node {
    private int Number;
    private ArrayList<Edge> edges = new ArrayList<Edge>();

    public Node(int num) {
        Number = num;
    }

    public Edge addEdge(int to, double gain) {
        Edge g = new Edge(Number, to, gain);
         edges.add(g);
        return g;
    }

    public boolean removeEdge(int to, double gain) {
        Edge g = new Edge(Number, to, gain);
         if (edges.contains(g)) {
             edges.remove(g);
             return true;
         }
        return false;
    }

    public Edge getEdge(int to){
        for(Edge g : edges) {
            if(g.getTo()==to && g.getFrom()==Number) {
                return g;
            }
        }
        return null;
    }

    public ArrayList<Edge> getEdges(){
        return edges;
    }

    public ArrayList<Integer> getTos() {
        ArrayList<Integer> to = new ArrayList<Integer>();
        for(int i=0;i<edges.size();i++) {
            to.add(edges.get(i).getTo());
        }
        return to;
    }
}
```

```java
package Model;

public class Edge{

    private Integer From;
    private Integer To;
    private double Gain;

    public Edge(Integer from, Integer to, double gain) {
        From = from;
        To = to;
        Gain = gain;
    }

    public Integer getFrom() {
        return From;
    }
    public void setFrom(Integer from) {
        From = from;
    }
    public Integer getTo() {
        return To;
    }
    public void setTo(Integer to) {
        To = to;
    }
    public double getGain() {
        return Gain;
    }

    public void setGain(double gain) {
        Gain = gain;
    }

}

```

2. *View : Which contains the Classes which have the responsibility to show the GUI and give the user the opportunity to add data to draw the SFG.*
3. *Control : which is responsible to merge the first the Model and View , taking the Graph from the GUI and build an Graph from the Graph class and pass it to Model with the given start and end points to return the results .*

Algorithms

Using a Depth-First Search algorithm to find all forward paths from the start given node to the end given node.

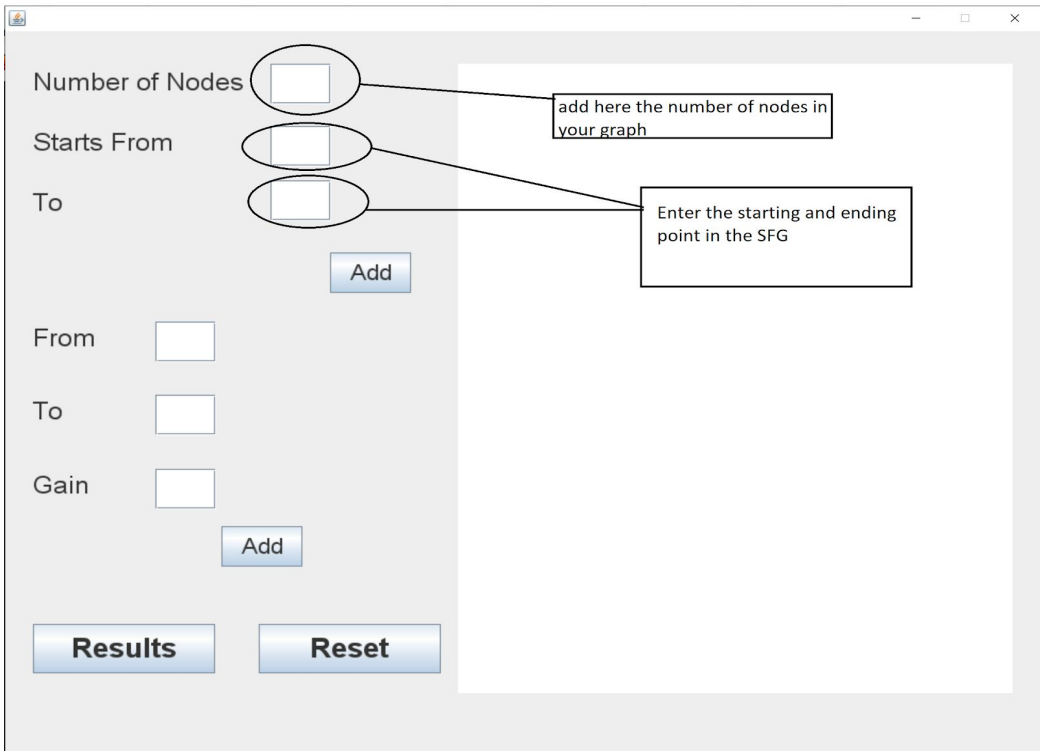
Using Tarjan algorithms to find all loops in the SFG

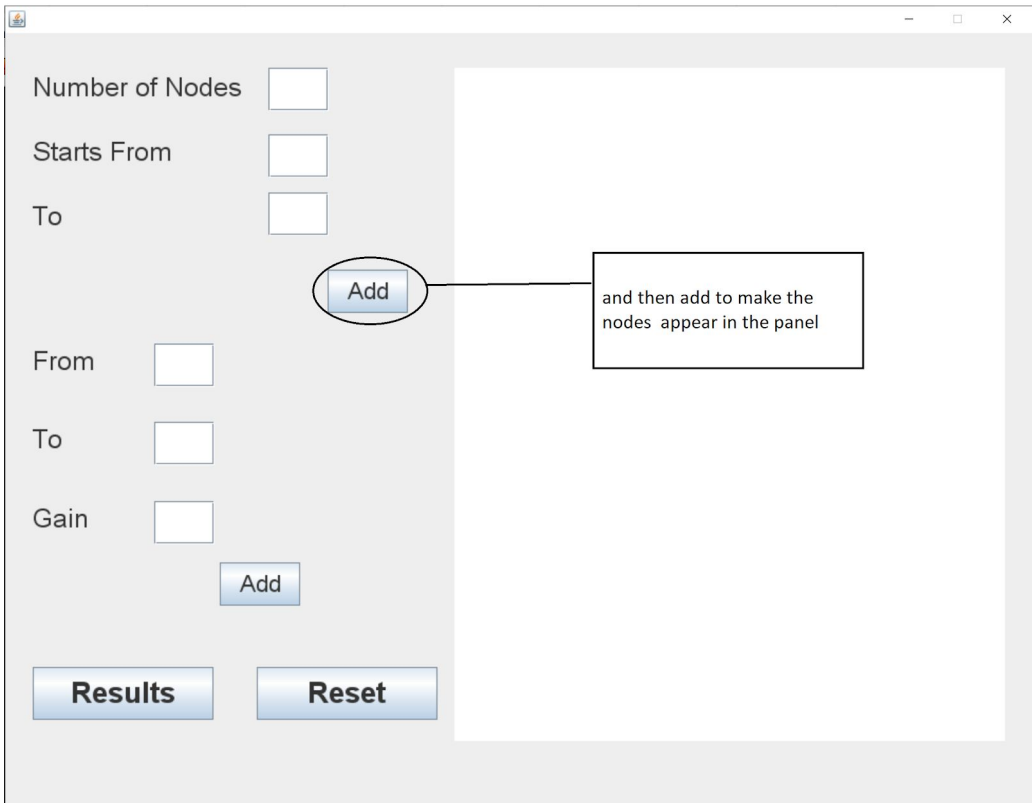
For Non-touched loops, first checking if 2 loops are touching if they are not touching, then put them in the nonTouchingLoops list in the first index list and pass it to complete the non touching loops function as tempLoops

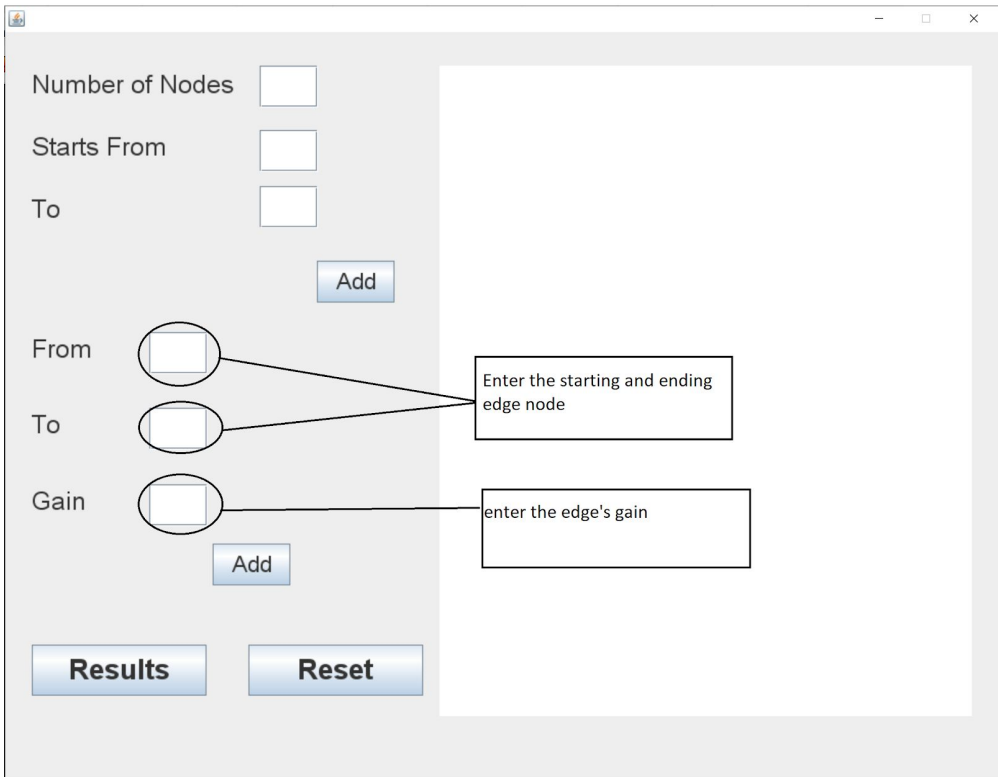
complete the non touching loops by taking one loop from the loop arraylist and the loops in the tempLoops and check if they are touching if they are not then add them to the nonTouchingLoops arraylist and so on

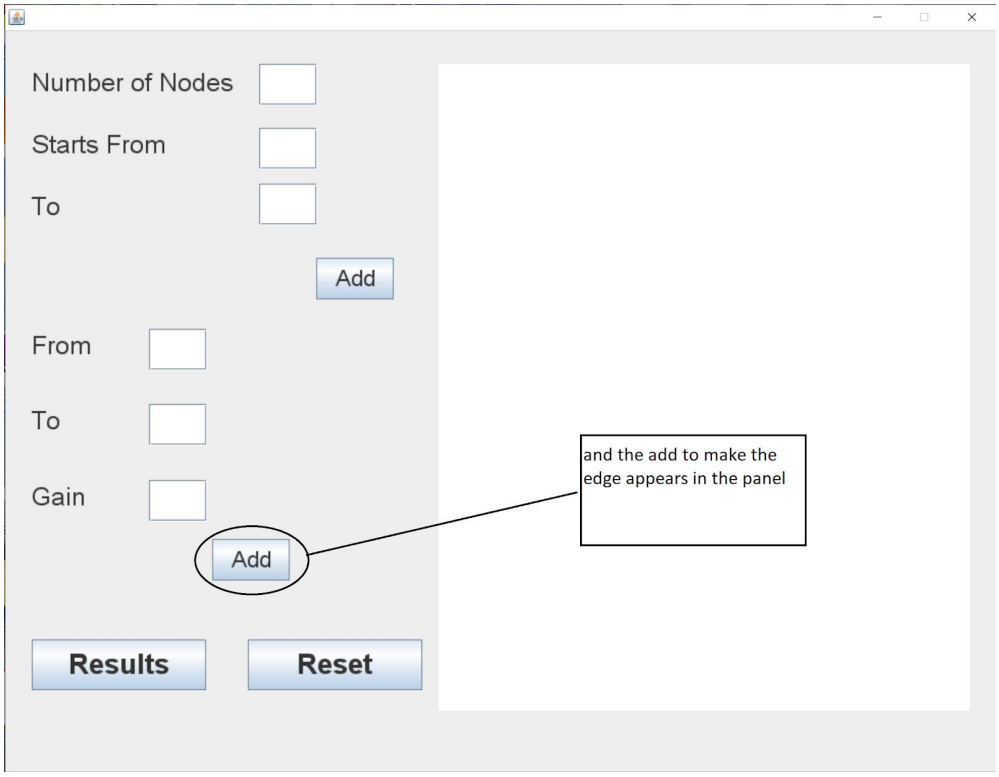
Finally remove the Non Touching Duplicate Loops

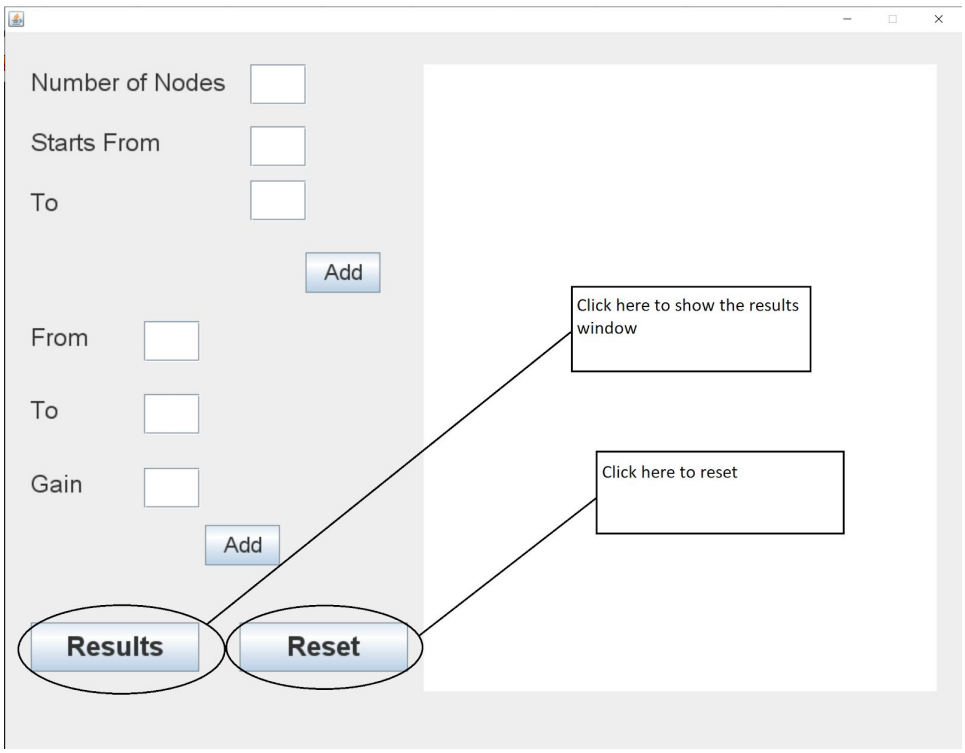
User Guide


Number of Nodes
Starts From
To
Add
From
To
Gain
Add
Results
Reset
add here the number of nodes in your graph
Enter the starting and ending point in the SFG

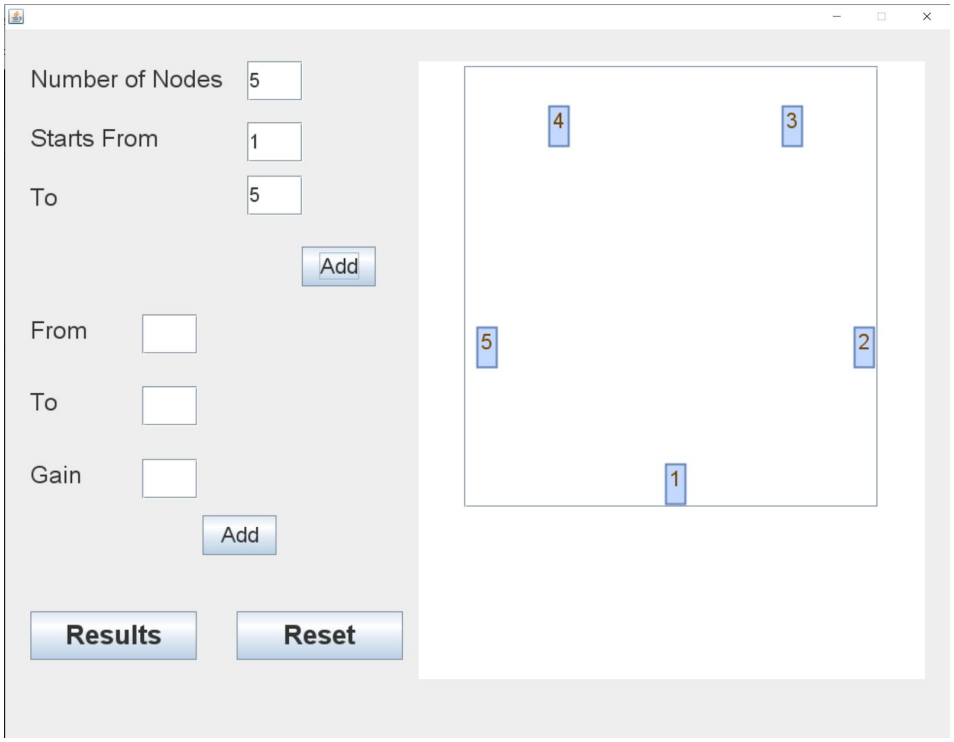

Number of Nodes
Starts From
To
Add
From
To
Gain
Add
Results
Reset
and then add to make the nodes appear in the panel

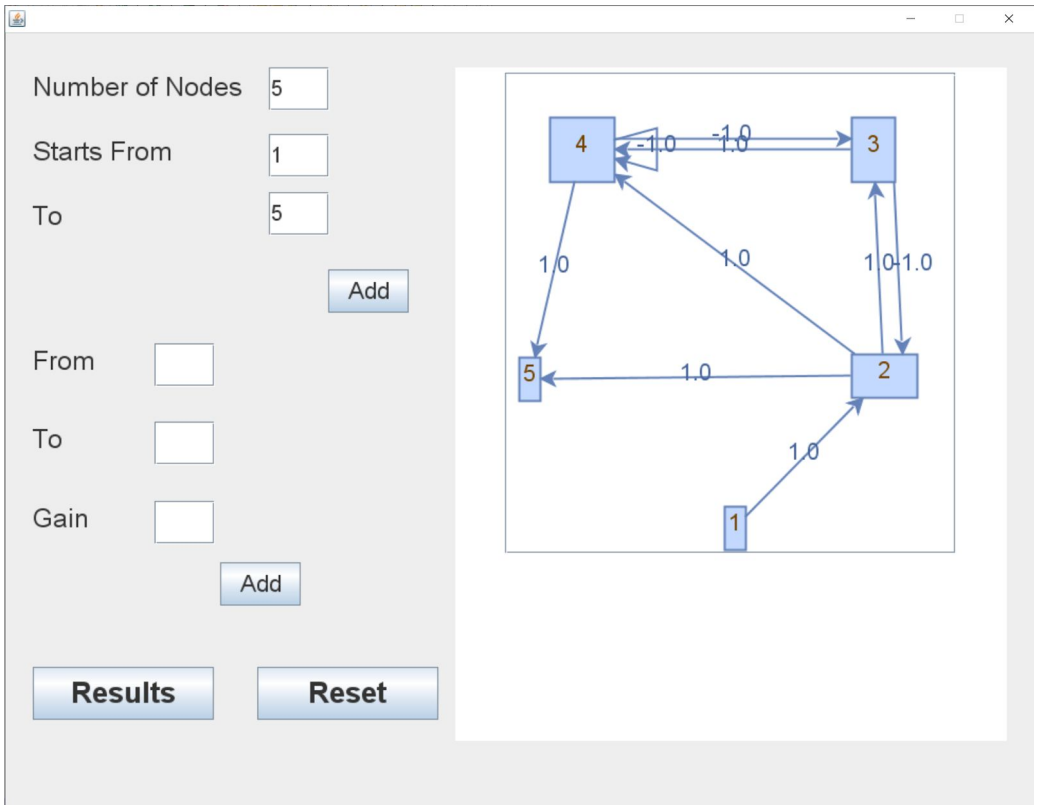

Number of Nodes
Starts From
To
Add
From
To
Gain
Add
Results
Reset
Enter the starting and ending edge node
enter the edge's gain

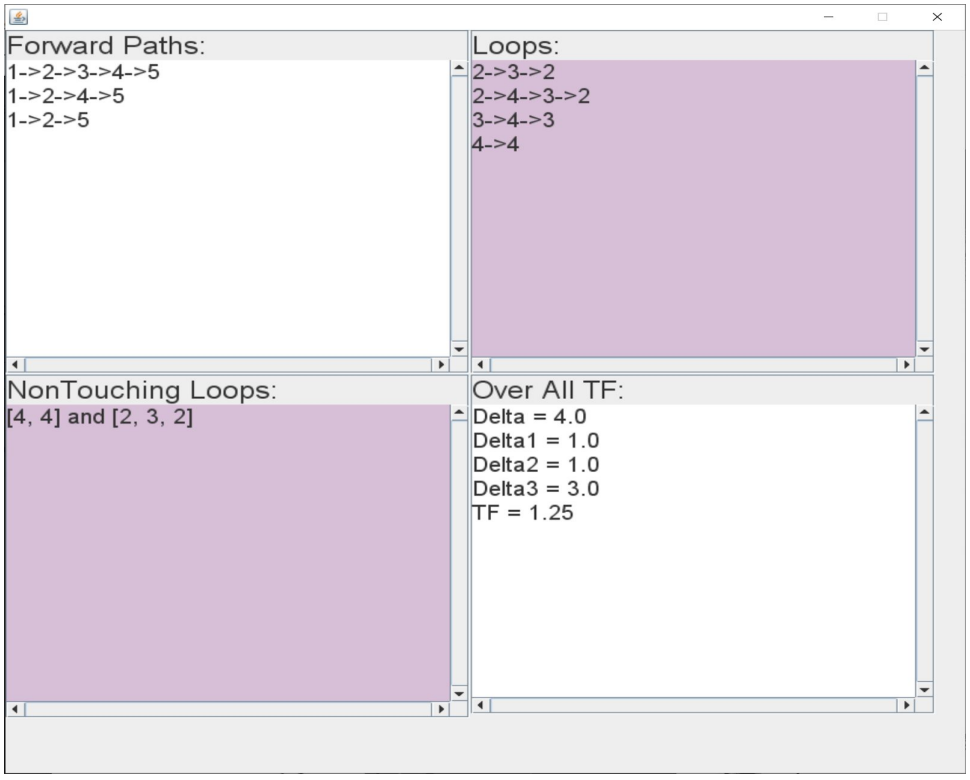

Number of Nodes
Starts From
To
Add
From
To
Gain
Add
Results
Reset
and the add to make the edge appears in the panel

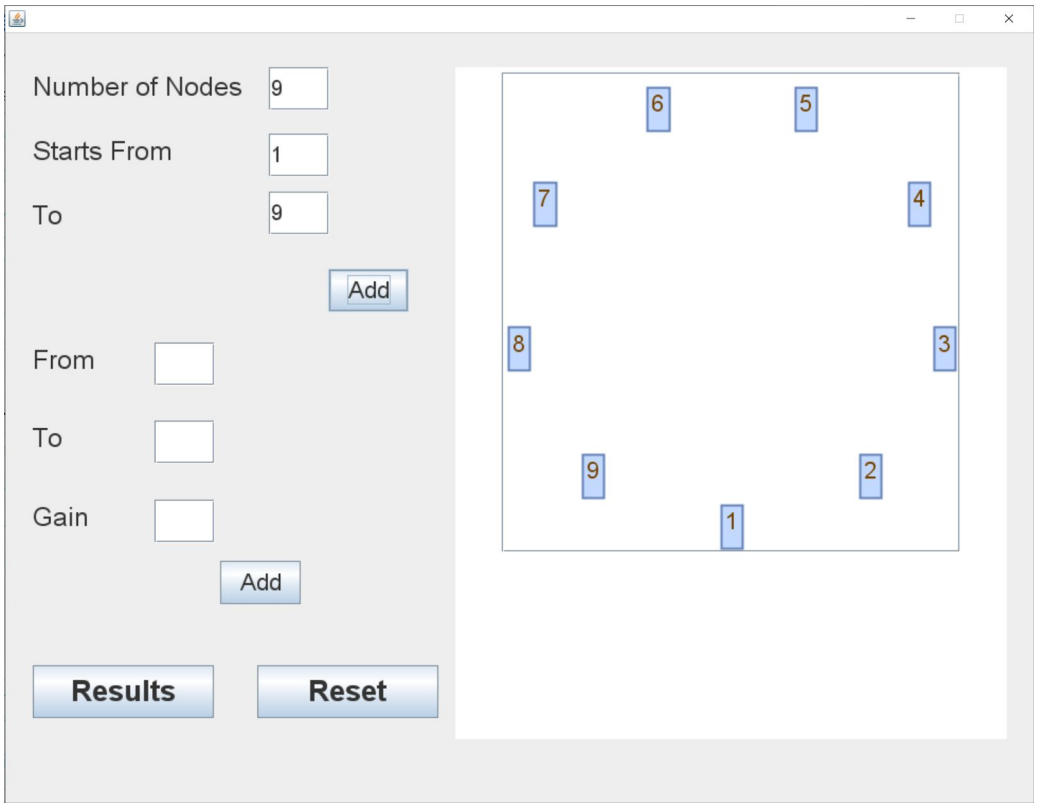

Number of Nodes
Starts From
To
Add
From
To
Gain
Add
Results
Reset
Click here to show the results window
Click here to reset

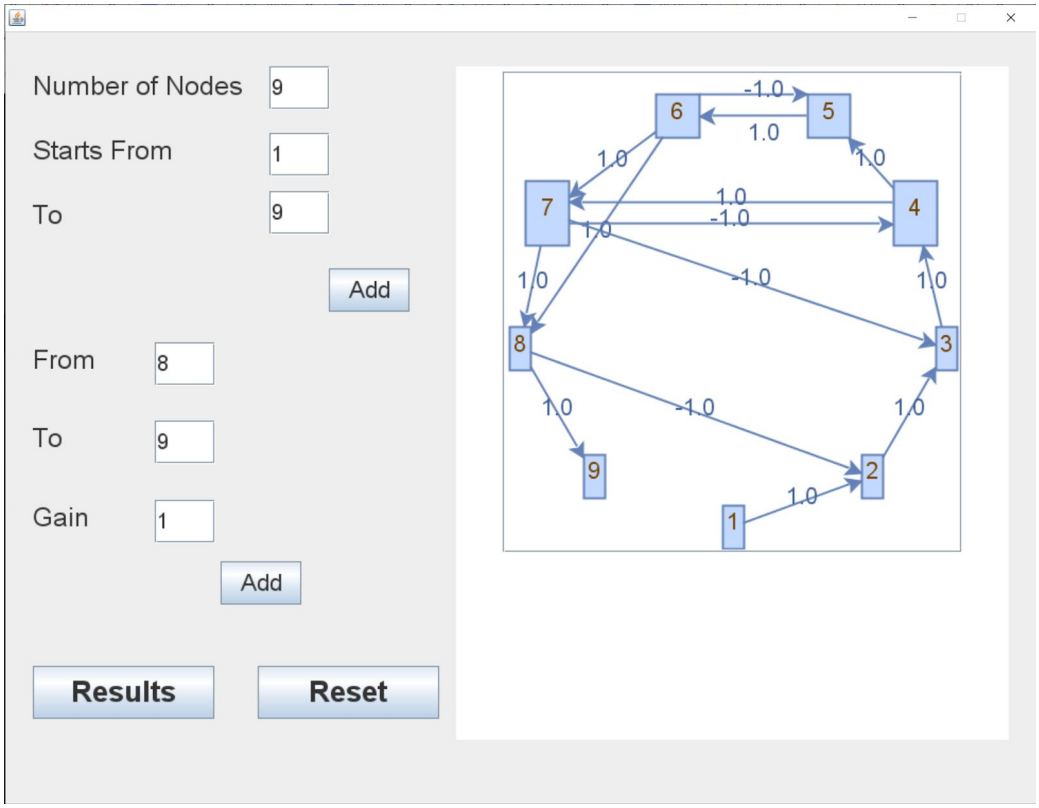
Sample Runs

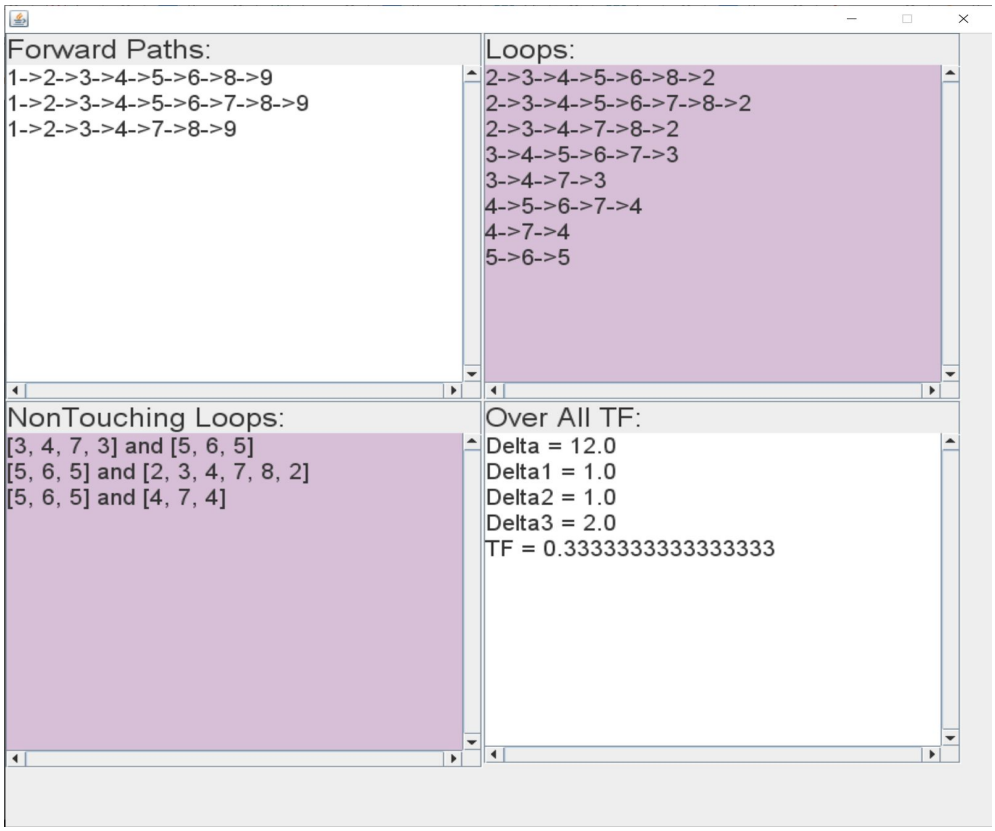

Number of Nodes
5
Starts From
1
To
5
Add
From
To
Gain
Add
Results
Reset
4
3
5
2
1

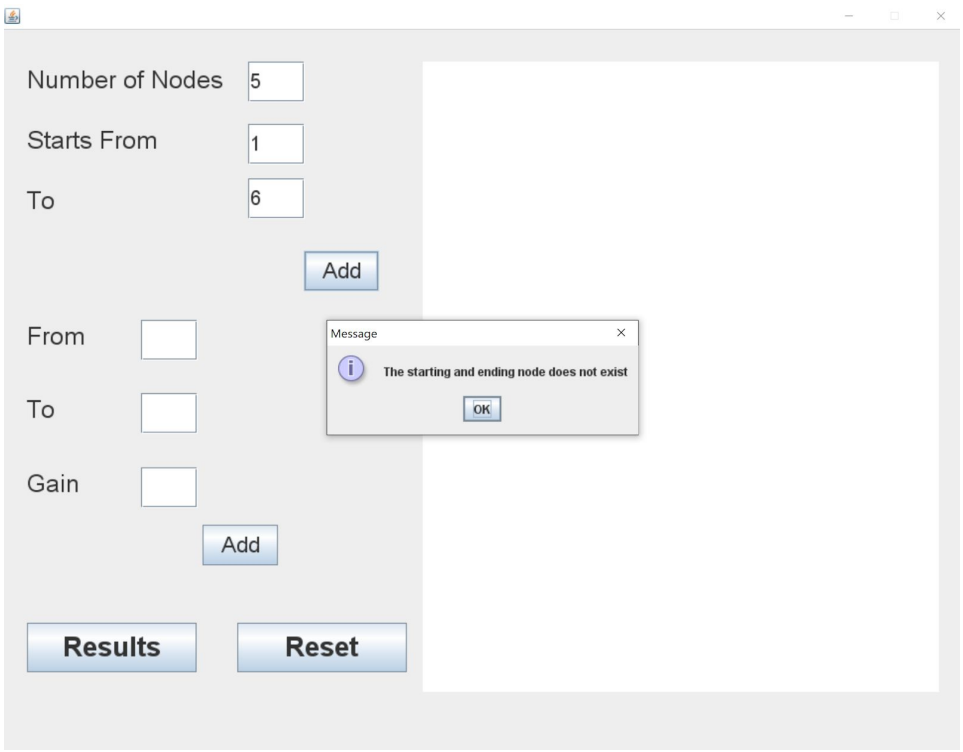

Number of Nodes
5
Starts From
1
To
5
Add
From
To
Gain
Add
Results
Reset
4
3
5
2
1
-1.0
-1.0
-1.0
1.0
1.0
1.0
1.0
1.0
1.0

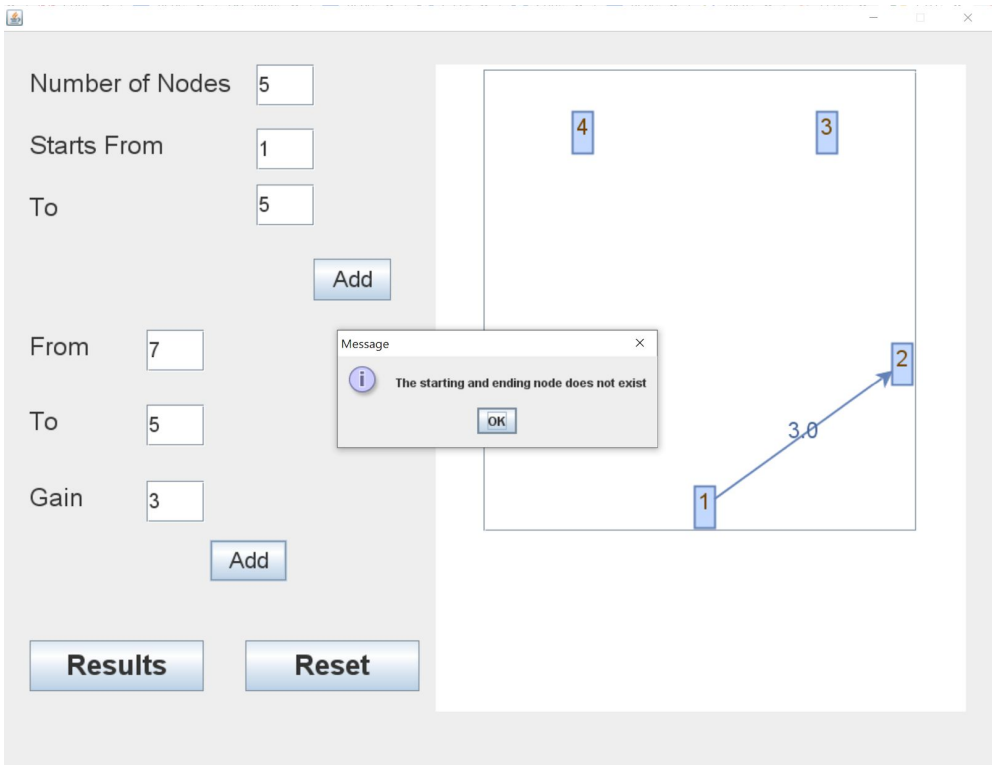

Forward Paths:
1->2->3->4->5
1->2->4->5
1->2->5
Loops:
2->3->2
2->4->3->2
3->4->3
4->4
NonTouching Loops:
[4, 4] and [2, 3, 2]
Over All TF:
Delta = 4.0
Delta1 = 1.0
Delta2 = 1.0
Delta3 = 3.0
TF = 1.25

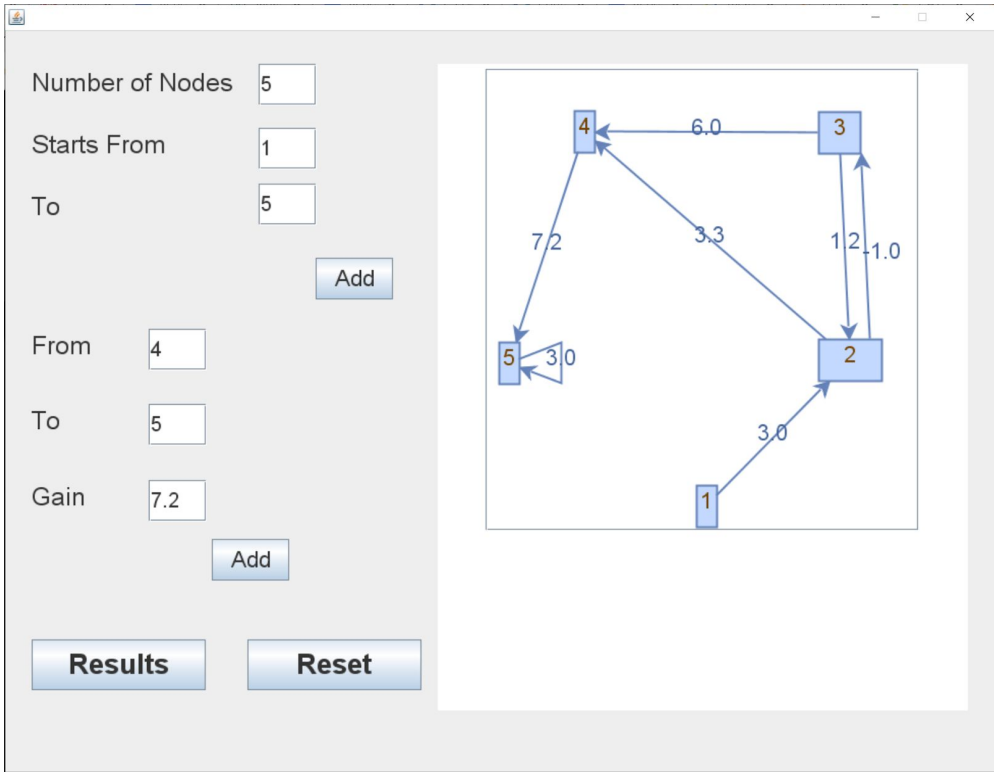

Number of Nodes
9
Starts From
1
To
9
Add
From
To
Gain
Add
Results
Reset
6
5
7
4
8
3
9
2
1

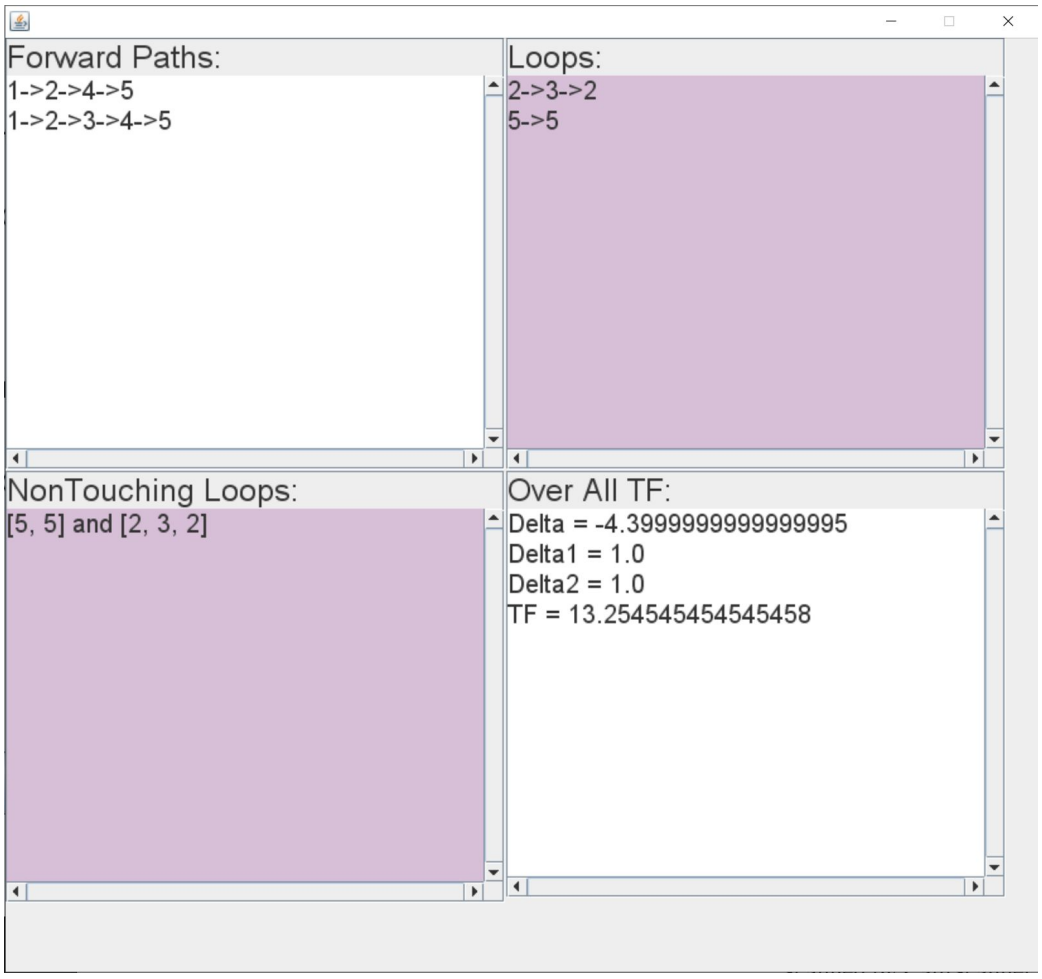

Number of Nodes
9
Starts From
1
To
9
Add
From
8
To
9
Gain
1
Add
Results
Reset
-1.0
1.0
1.0
1.0
1.0
1.0
-1.0
1.0
1.0
-1.0
1.0
1.0
-1.0
1.0
1.0
6
5
7
4
8
3
9
2
1


Forward Paths:
1->2->3->4->5->6->8->9
1->2->3->4->5->6->7->8->9
1->2->3->4->7->8->9
Loops:
2->3->4->5->6->8->2
2->3->4->5->6->7->8->2
2->3->4->7->8->2
3->4->5->6->7->3
3->4->7->3
4->5->6->7->4
4->7->4
5->6->5
NonTouching Loops:
[3, 4, 7, 3] and [5, 6, 5]
[5, 6, 5] and [2, 3, 4, 7, 8, 2]
[5, 6, 5] and [4, 7, 4]
Over All TF:
Delta = 12.0
Delta1 = 1.0
Delta2 = 1.0
Delta3 = 2.0
TF = 0.3333333333333333


Number of Nodes
5
Starts From
1
To
6
Add
From
To
Gain
Add
Results
Reset
Message
The starting and ending node does not exist
OK


Number of Nodes
5
Starts From
1
To
5
Add
From
7
To
5
Gain
3
Add
Results
Reset
Message
The starting and ending node does not exist
OK
4
3
2
1
3.0


Number of Nodes
5
Starts From
1
To
5
Add
From
4
To
5
Gain
7.2
Add
Results
Reset
4
6.0
3
7.2
3.3
1.2
-1.0
5
3.0
2
3.0
1


Forward Paths:
1->2->4->5
1->2->3->4->5
Loops:
2->3->2
5->5
NonTouching Loops:
[5, 5] and [2, 3, 2]
Over All TF:
Delta = -4.3999999999999995
Delta1 = 1.0
Delta2 = 1.0
TF = 13.254545454545458