Signal Flow Graph

By

1. Ahmed Nasser abdelkareem (9)
2. Zeyad Ahmed Elbanna (27)

# Problem Statement

* Signal flow graph representation of the system. Assume that total number of nodes and numeric branches gains are given.

**Required :-**

1- Graphical interface.

2- Draw the signal flow graph showing nodes, branches, gains.

3- Listing all forward paths, individual loops, all combination of *n* non-touching loops.

4- The values of Deltas.

5- Overall system transfer function.

# Main features

**Using Given Branches :**

* Listing Forward Paths.
* Listing Deltas.
* Listing non-Touching Loops.
* Output the overall Transfer Function.
* Visualization of Graph.

# Data structures

* ArrayList of Branches (From , to , gain).
* ArrayList of ArrayList of Integers : describes the number of loop and each loop component.
* ArrayList for forward paths.
* ArrayList of Nodes .
* ArrayList of Edges.

# Algorithms Used

We created three packages which has the parts of our whole project:

1. The first package is flowgraph:

This package has the main components of the project as a whole, it contains:

* Node : node class with all its components and attributes
* Edge: The class of the edge which is the connecting component between two nodes
* Graph: Which represents a loop or a forward path
* Flowgraph: Which is a class mainly used to update mason’s formulae and any new nodes added r removed from the main graph, update the graph and the whole formula

1. The second package is backend main package which was used to calculate the mason’s formula and the main transfer function of the program, This is done by finding out the forward paths and all the loops of the program together with all the repeated loops an getting rid of them afterwards. Its ain classes are :

* DeltaFunction: Mainly responsible for calculating the delta function for each loop as an input and using it afterward to calculate the transfer function
* Forwardpaths: This is mainly responsible for getting all the forward paths of the program and using them to caculate the TF
* Loops: all the program loops, redundant and main loops are generated

1. The third and final package was for the Graphical User interface created for the user by a friendly way and high-quality service.

Most important algorithm used:

Our most important used algorithm was to find the forward paths as well as the loops of the system by using the BFS algorithm in the program:

Its main aim is to fill the list of graphs (loops) of the system as well as the forward paths of the system and then removing the redundant loops away of the main loops The simplified algorithm was as follow:

**private** **static** **void** BFS(Node n) {

// visit every node that you can reach from this node

n.setVisited(**true**);// to avoid loops in graph

n.startLooping();

Edge e = n.getNextEdge();

Node next;

Edge ed;

Node nextn;

**while** (e != **null**) {

next = e.getTo();

**if** (!next.isVisited()) {

next.setParent(e);

*BFS*(next);

} **else** {// loop has been found

*generatePath*(n, next, e);

}

e = n.getNextEdge();

}

n.setVisited(**false**);// make it available for other searches

}

Fig 1.a

The algorithm in fig 1.a show the BFS used to find all the loops in the system and then generating the path associated with it.

Another main algorithm used to find the forward paths of the system was also BFS as follows:

**private** **static** **void** BFS(Node n) {

**if** (n.equals(*end*)) {// Successful forward path

*generatePath*(n);

**return**;

}

// visit every node that you can reach from this node

n.setVisited(**true**);// to avoid loops in graph

n.startLooping();

Edge e = n.getNextEdge();

Node next;

**while** (e != **null**) {

next = e.getTo();

**if** (!next.isVisited()) {

next.setParent(e);

*BFS*(next);

}

e = n.getNextEdge();

}

n.setVisited(**false**);// make it available for other searches

}

Fig1.b

# Main Modules

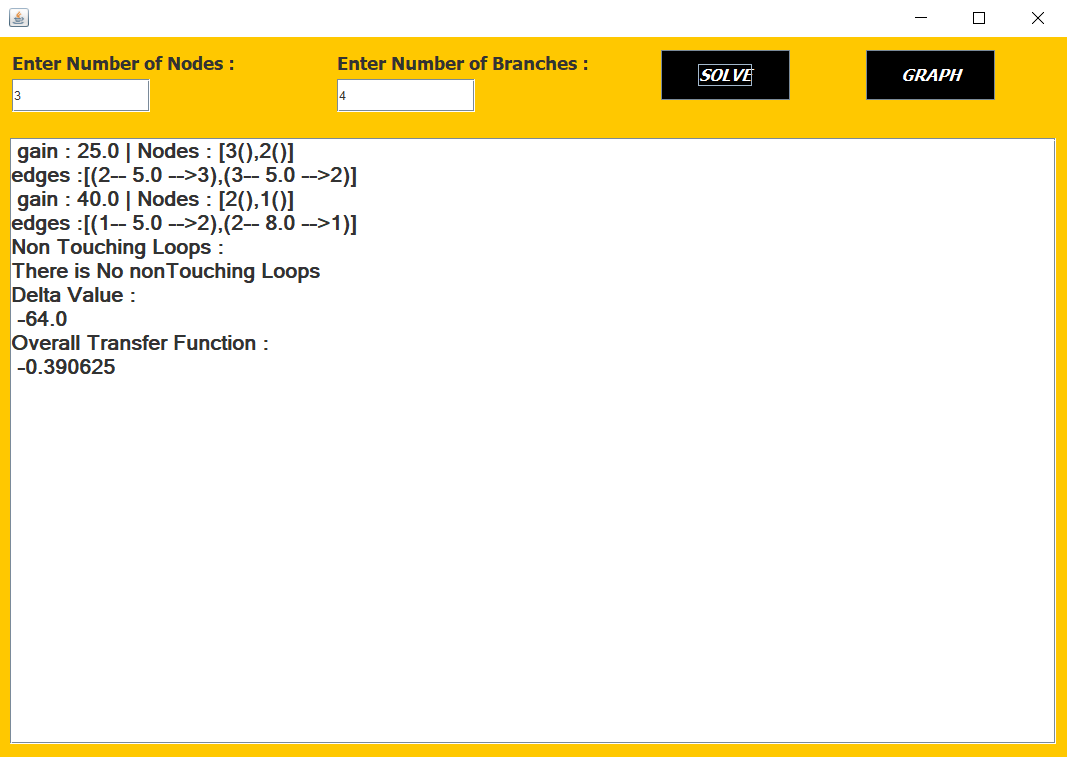
* Edge and Node Classes.
* Graph and Flow graph Module.
* Delta function.
* Forward paths.
* Loops module.
* Test Module.
* GUI module.
* Graph Visualization Module.

# Sample run

* Using problem with 3 nodes and 4 edges

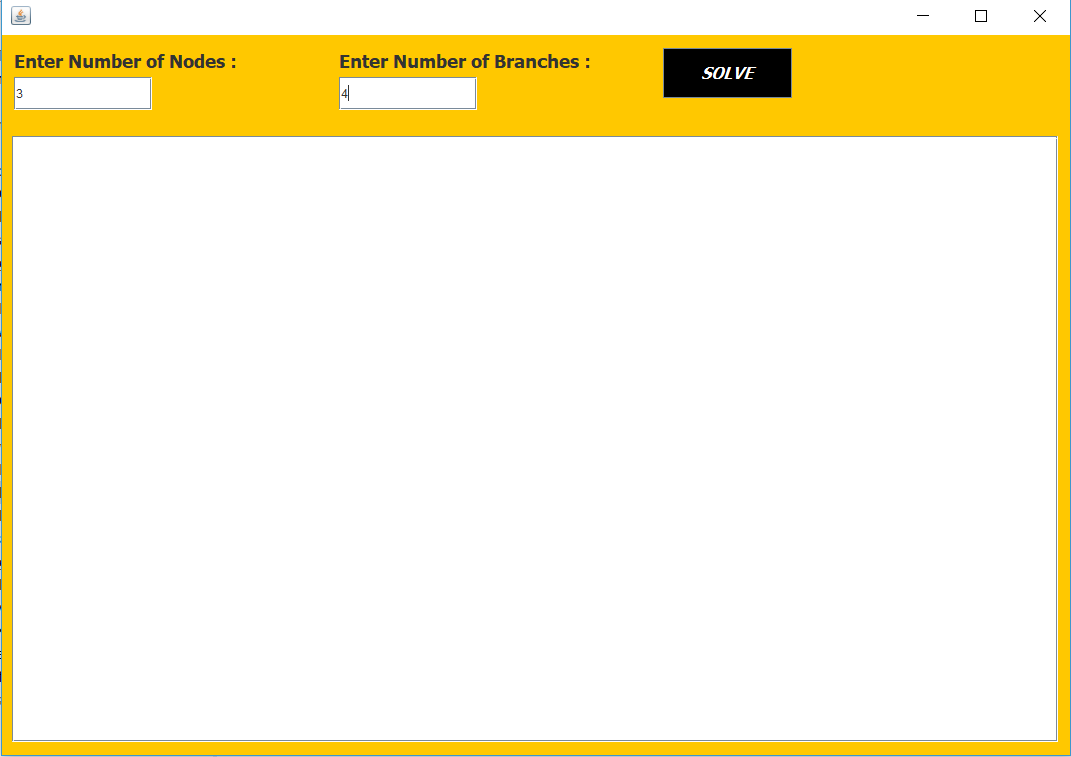
(1,2,5) , (2,3,10) , (3,2,15) , (2,1,20)

* The answer will be :

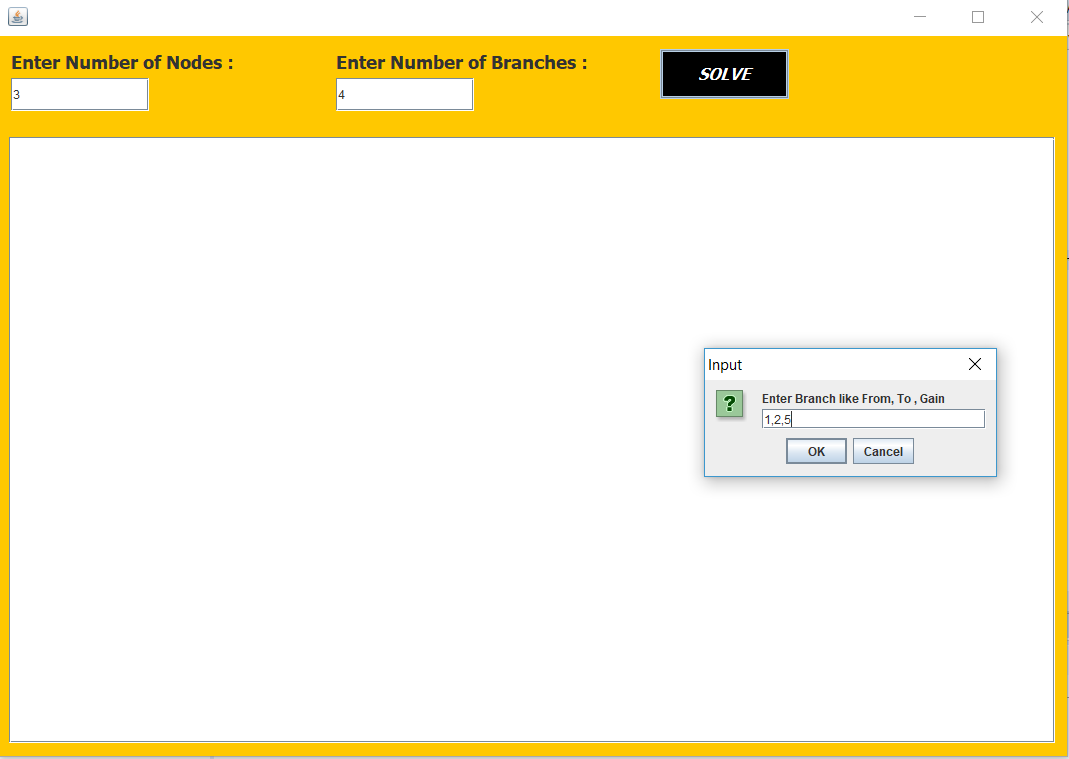


# USER GUIDE

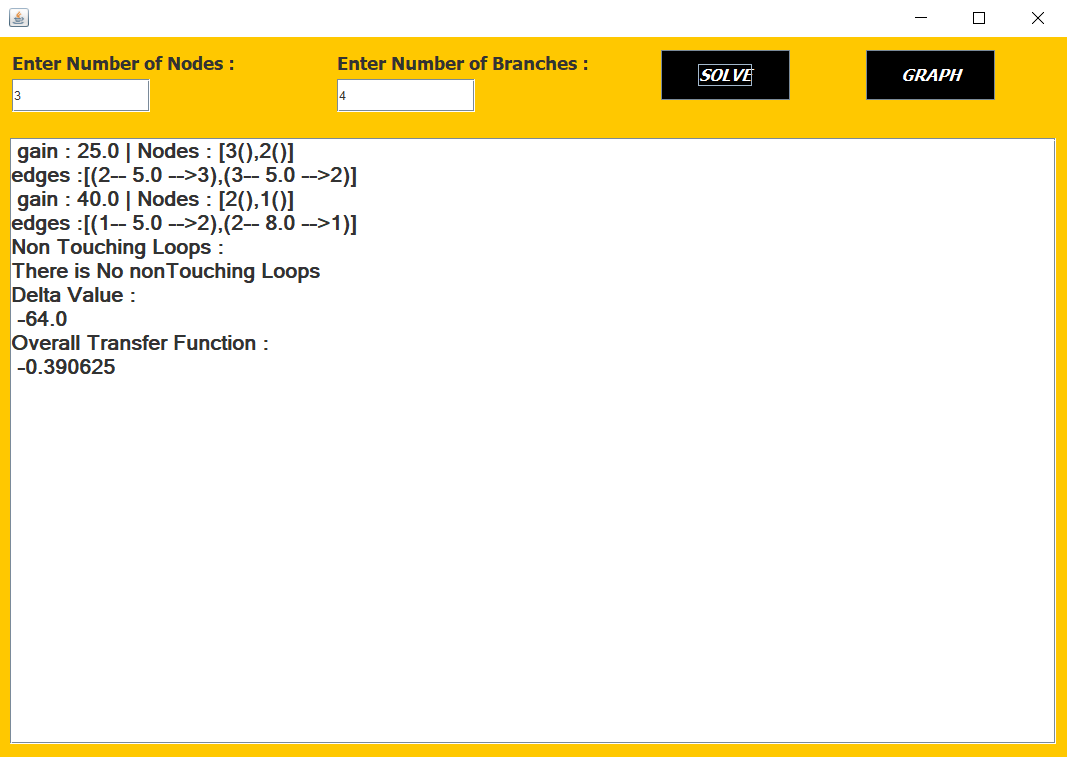
* Enter number of nodes and number of branches :



* Click SOLVE button to start writing edges with format (FROM,TO,GAIN)



* After Finishing edges Entry , The steps will shown in the bottom text area like this and the button GRAPH will be visible.



* When Click on GRAPH button the visualization of Graph will be show in different frame .

