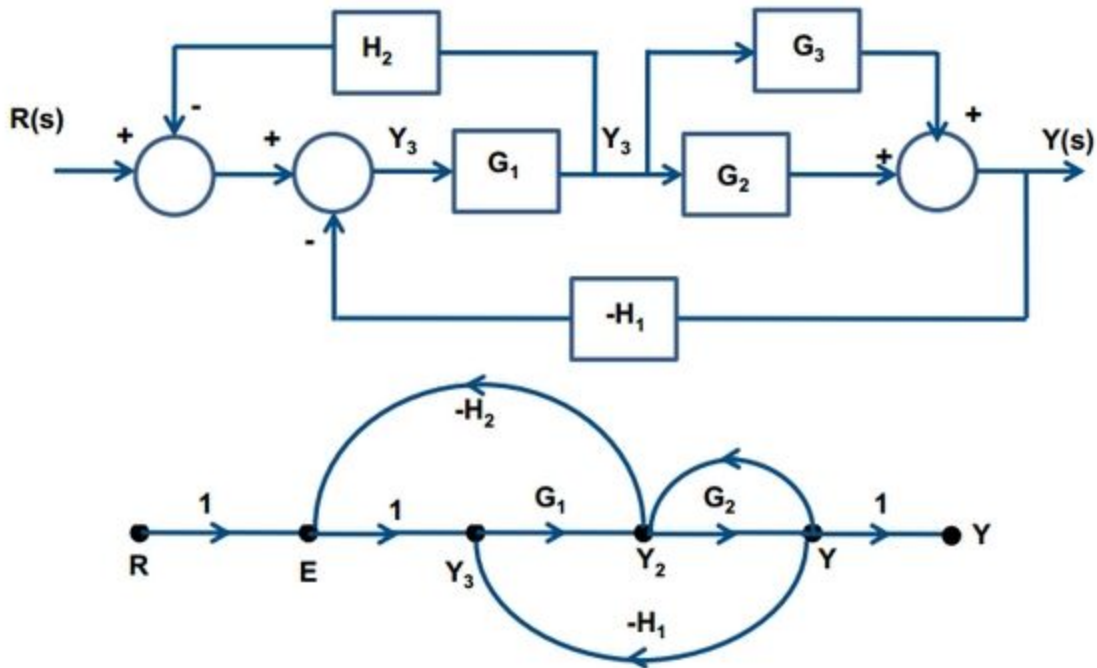


# SIGNAL FLOW GRAPH

*Linear Control System Project Report*



## Team Members:

Alaa Samir (1)

Bassant Ahmed (18)

# Table of Content:

- 1) Project Description and Specifications
- 2) Design and Data Structure
  - a) Modules
  - b) Data Structure
  - c) Libraries
- 3) Algorithm
- 4) Assumptions
- 5) GUI
  - a) Features
  - b) User Interface
  - c) Samples

# Project Description and Specifications

## DESCRIPTION

A signal-flow graph or signal-flow graph (SFG), invented by Claude Shannon, but often called a Mason graph

after Samuel Jefferson Mason who coined the term, is a specialized flow graph, a directed graph in which nodes represent system variables, and branches (edges, arcs, or arrows) represent functional connections between pairs of nodes. Thus, signal-flow graph theory builds on that of directed graphs (also called digraphs), which includes as well that of oriented graphs. This mathematical theory of digraphs exists, of course, quite apart from its applications.

SFGs are most commonly used to represent signal flow in a physical system and its controller(s), forming a cyber-physical system. Among their other uses are the representation of signal flow in various electronic networks and amplifiers, digital filters, state-variable filters and some other types of analog filters. In nearly all literature, a signal-flow graph is associated with a set of linear equations.

## SPECIFICATIONS

- Graphical User Interface
- Drawing Signal Flow Graph showing all nodes and edges
- Listing all Loops and paths including the 'N' non touching loops.
- Calculating the values of Delta for each path
- Calculating the overall transfer function
- 

## Design and Data Structure

### MODULES

The project consists of 3 main modules, Graph component module, Graph solver module and the Drawing Engine module.

### Graph component:

Contains all Graph components from:

- Edges
- Graphs
- Paths

As well as The corresponding interfaces.

These components are for the use of storing the graph data and displays the output to the user whether through tables or printed graphs.

### Graph Solver:

Graph Solver contains only one class which is the solver class, that solves the given graph producing all forward paths, cycles, non touching loops, and the required deltas for each path and for the overall formula.

### Draw Engine:

Connects the gui with the backend, it contains as well the **automatically drawn graph** given by the user and the output table containing all formulas' data.

## DATA STRUCTURE

### Graph component:

- IEdge-Interface, Edge-Class:  
It holds all of the edges information as:
  - From Node
  - To Node
  - Gain
- IGraph-Interface, Graph-Class:  
It holds all of the graph information as:
  - Remove edge
  - Add edge
  - Get graph
- IPath-Interface, Path-Class:  
In which all loops and cycles are stored with all the needed information such as:

- Add Node
- Remove Node
- Get Gain
- Set Gain
- Clone Path
- Get Path size
- List of Nodes - nodes that compose the path/cycle-.

### Graph Solver:

ISolver-Interface, Solver-Class:

The constructor takes the graph data structure and uses both built in data structures and graph component data structures to solve the given data. The solver class contains the following functions found necessary in computing mason's formula:

- Solve forward paths
- Solve loops
- Solve graph Delta
- Solve Non touching loops
- Get transfer function
- Check graph Connectivity
- New Graph

While the built in data structures used are:

- **Lists** -To store loops and paths
- **HashSet** -As in order to check if 2 loops touches through storing their vertices in a hashset, hashset was chosen for being faster and easier to use than lists or arrays.
- **StringBuilder** -stores all N non touching loops number in a string builder to save time during calculations and in addition to that it is faster than a normal string, as normal strings are re constructed every time in concatenation.
- **Double array** -stores the graph as a matrix representation.

### Draw Engine:

- Draw Class:

Contains the GUI layout which consists of :

- BoxLayout to organize the buttons within the GUI
- User input validation

- Graph canvas area
- Table Class:  
This Frame is called within the main GUI 'Draw' class when the user presses solve, it contains the linkage between the gui and the backend through passing the graph constructor to it and calling 'Solve' methods within the GraphSolver package then displaying the output in a table that contains the following fields:
  - Type whether path or cycle
  - The Vertices composing these cycles/paths
  - The gain of each cycle/path
  - The overall Transfer function

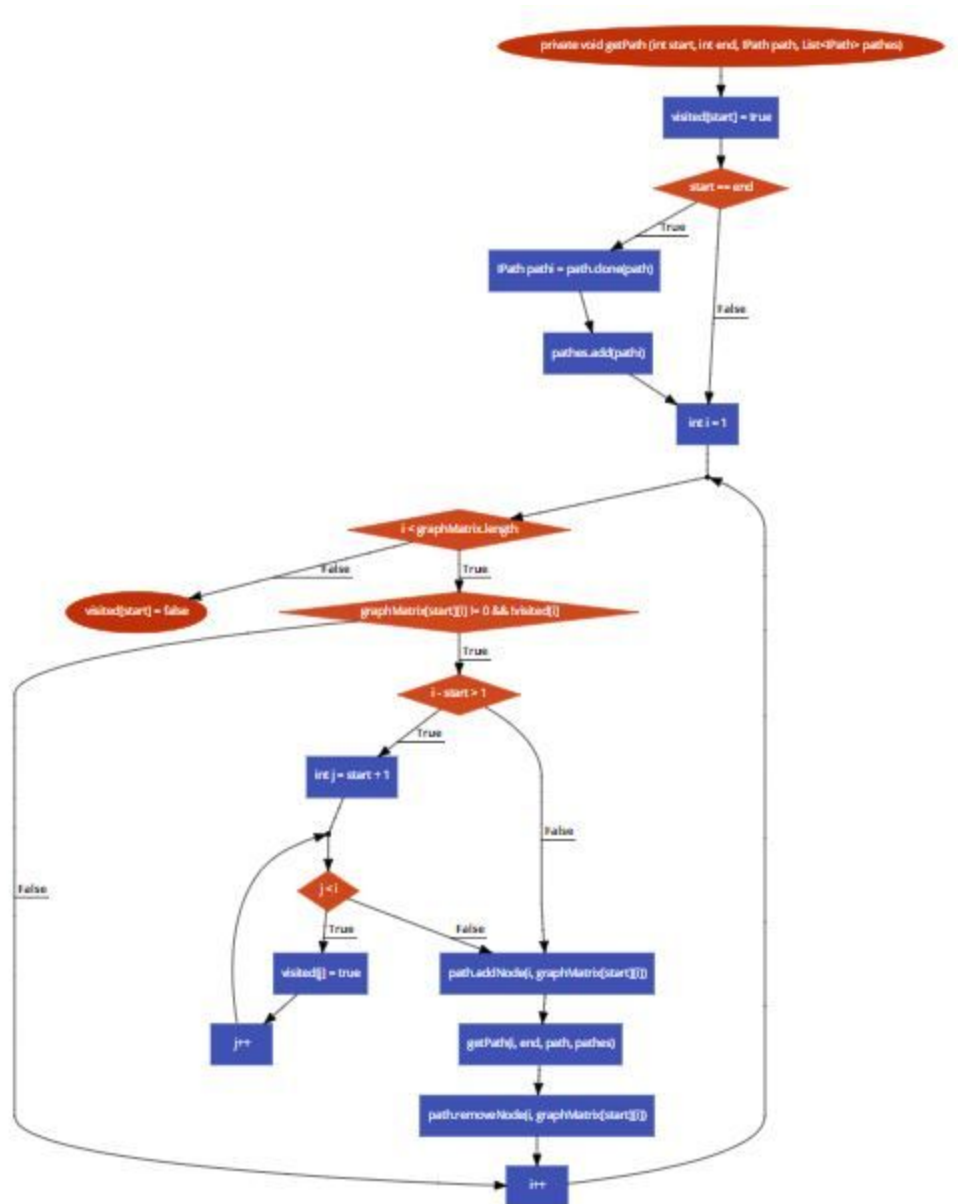
## LIBRARIES

Libraries used are:

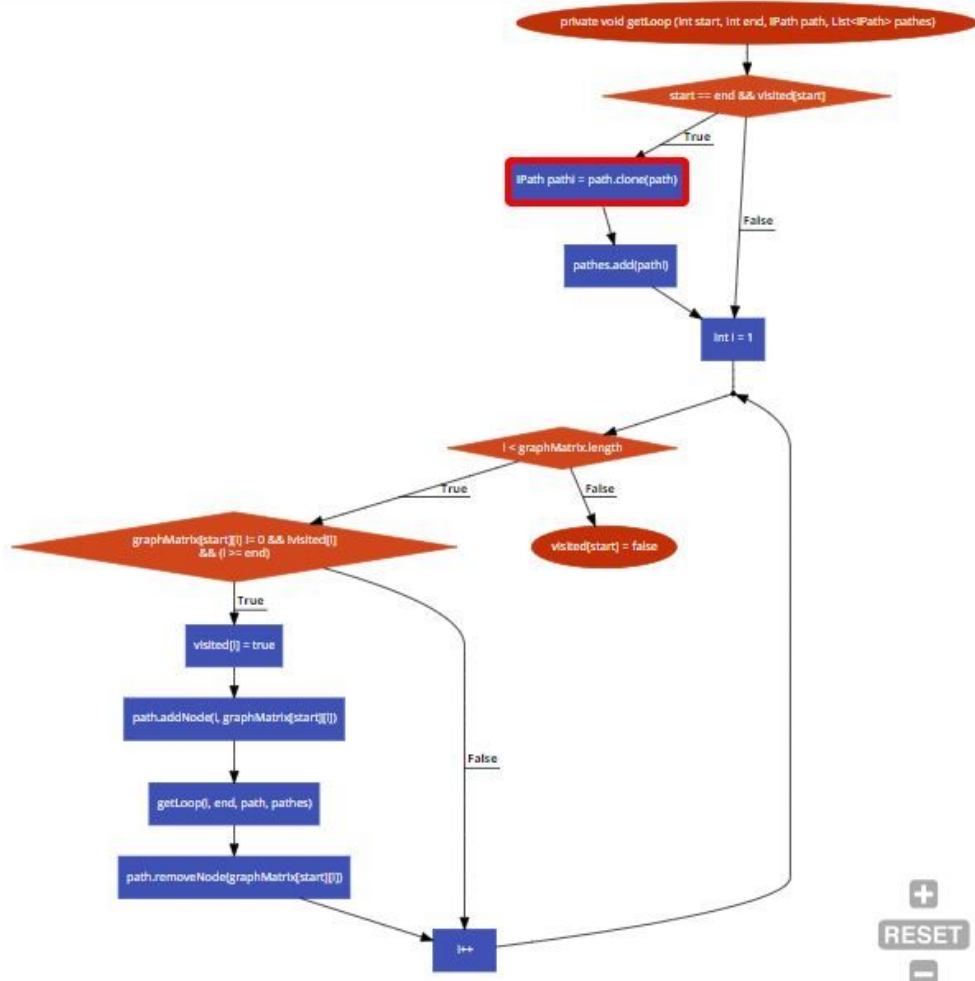
- Apache-Commons-Lang:  
Used for numerical parsing to handle user input.
- Jgrapht-Jgraphx:  
Used for the automatic drawing of the graph.

# Algorithms

## 1) Graph Solver - Get Path:

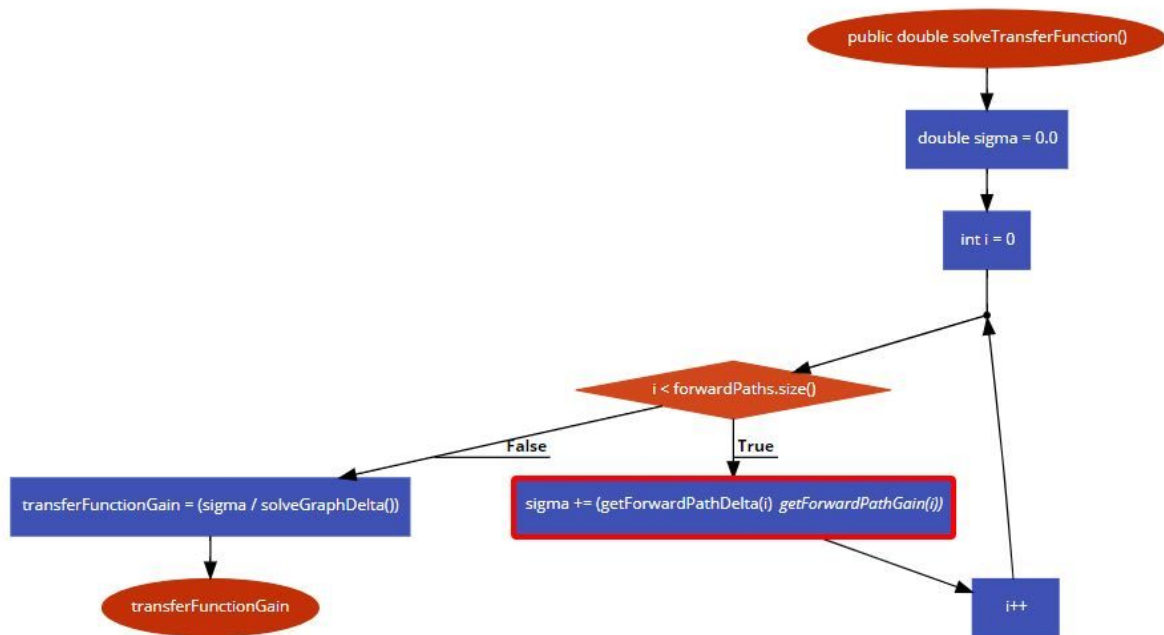


## 2) Graph Solver - Get Loop:

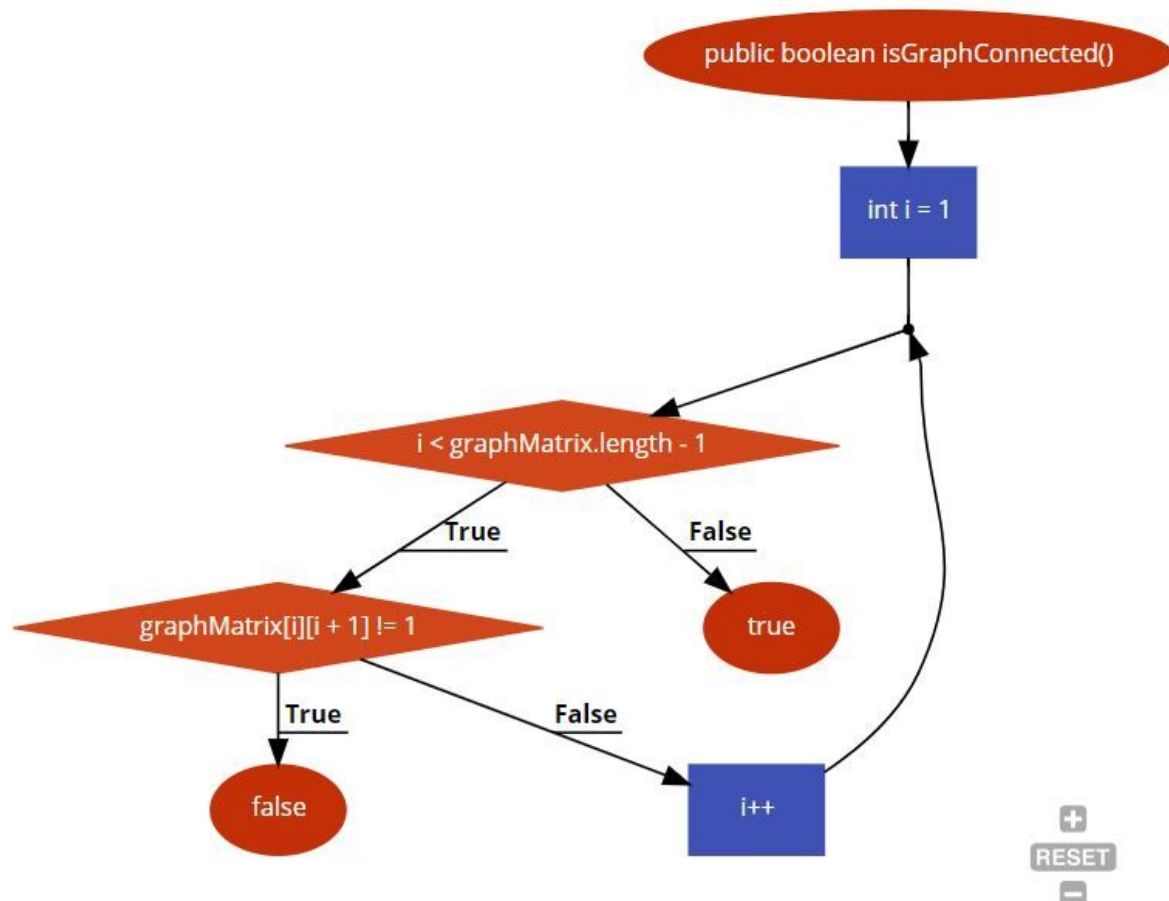




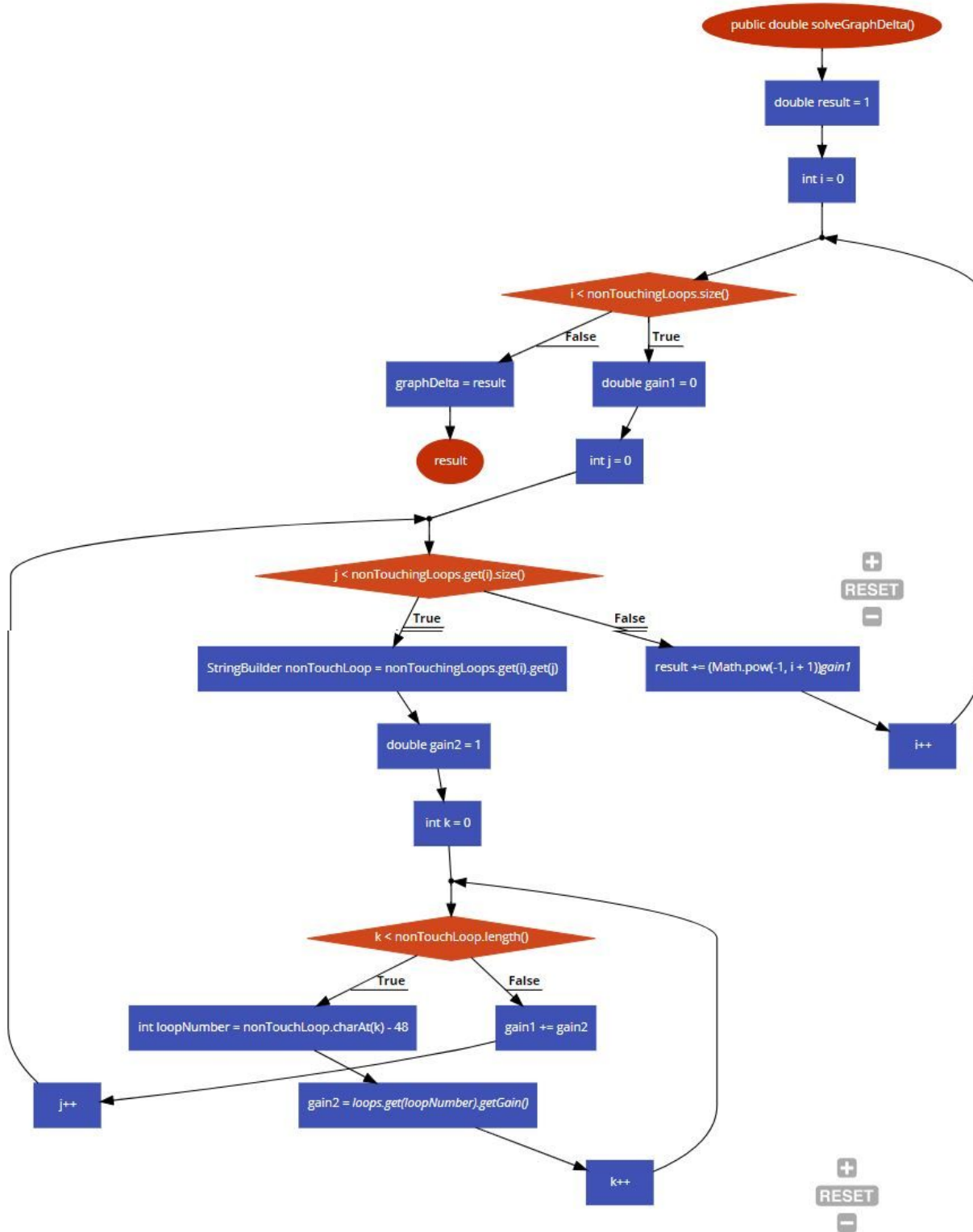
### 3) Graph Solver - Solve transfer Function:



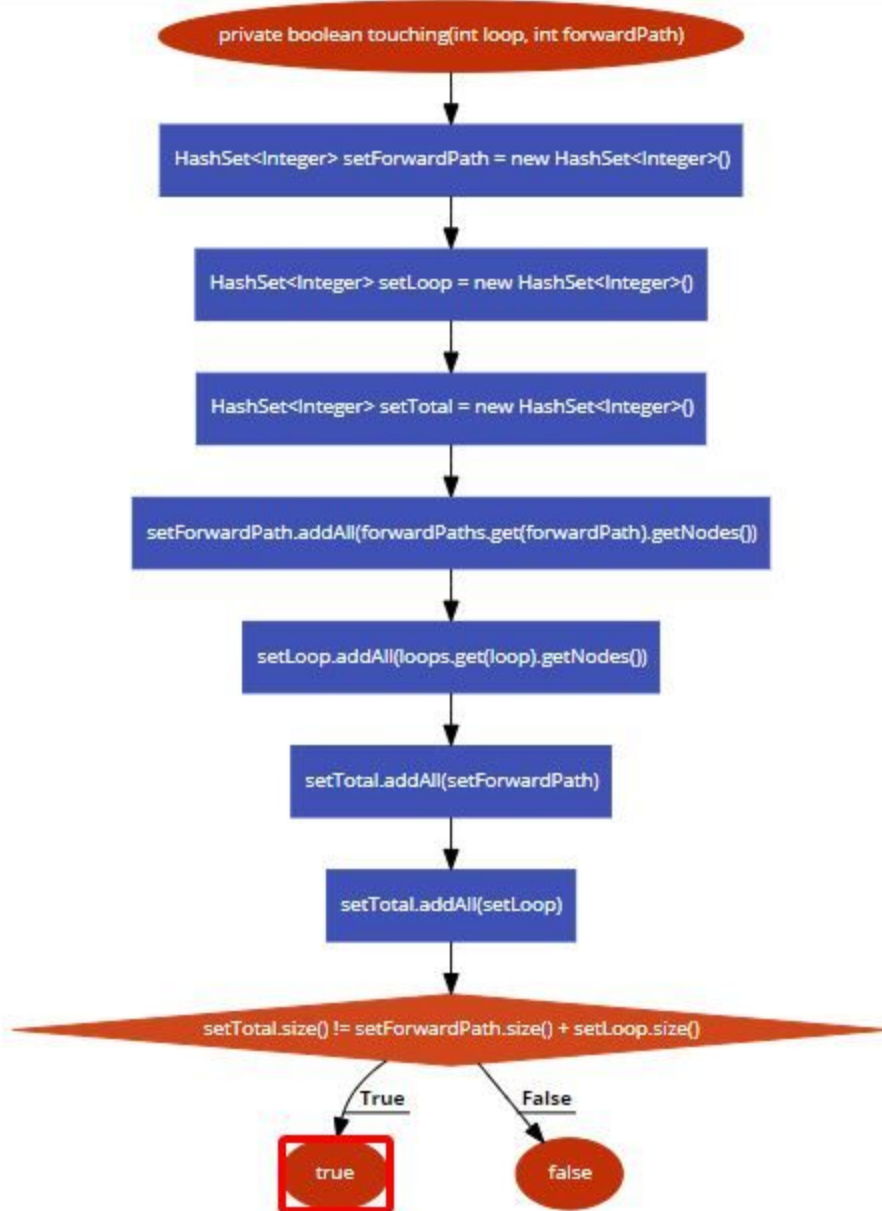
#### 4) Graph Solver - IsConnected:



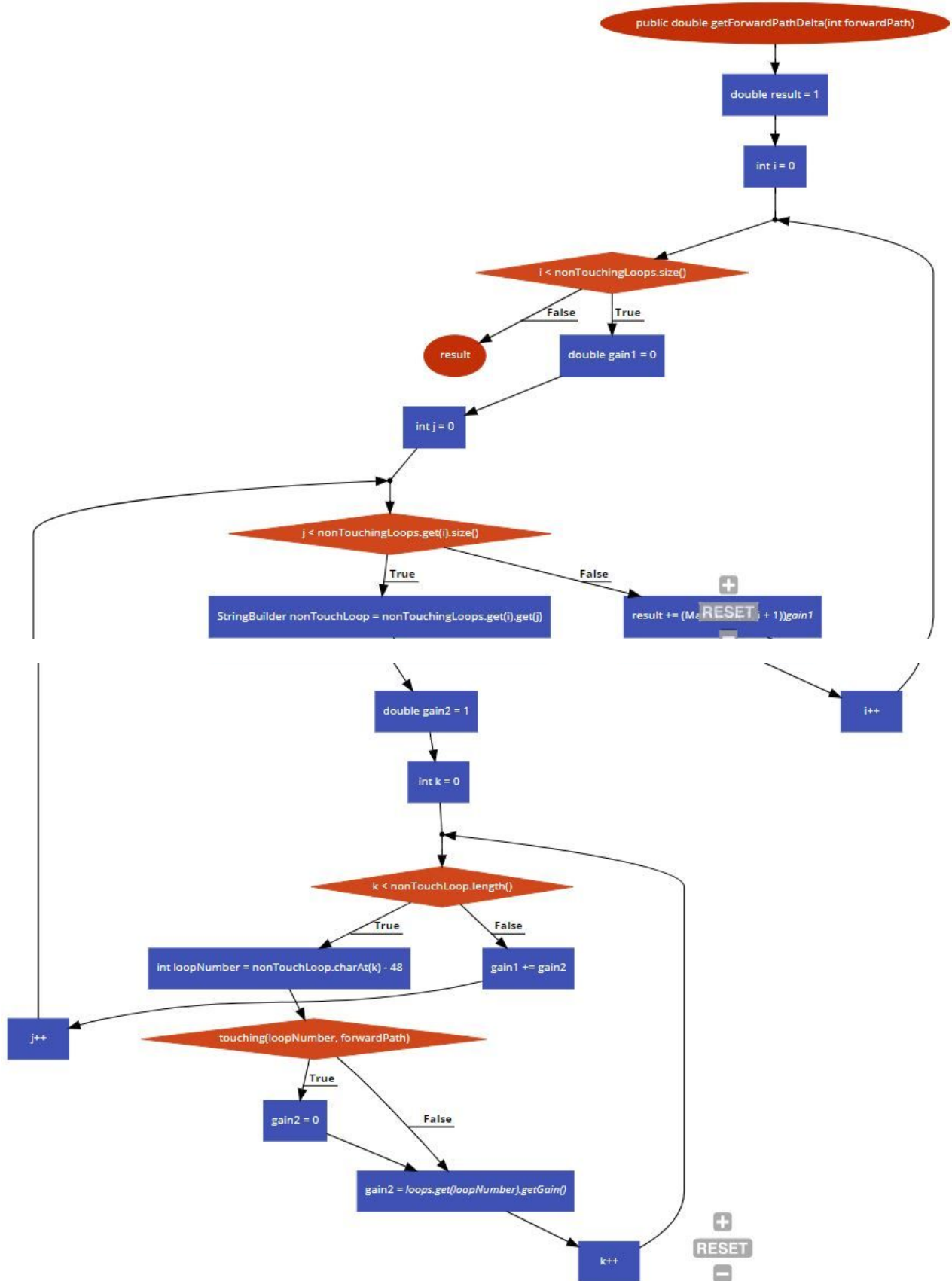
## 5) Graph Solver - Get Graph Delta:



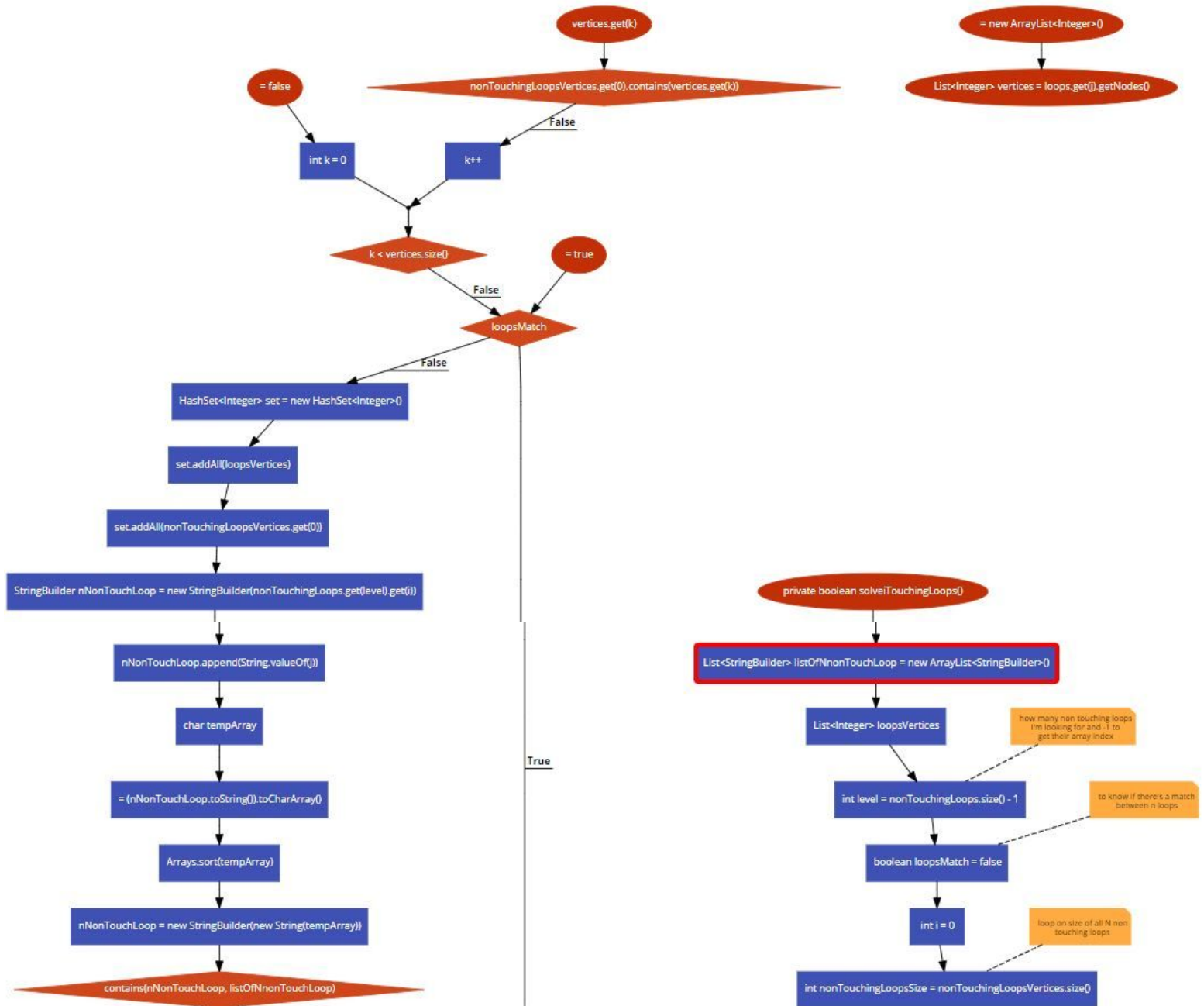
## 6) Graph Solver - Touching :



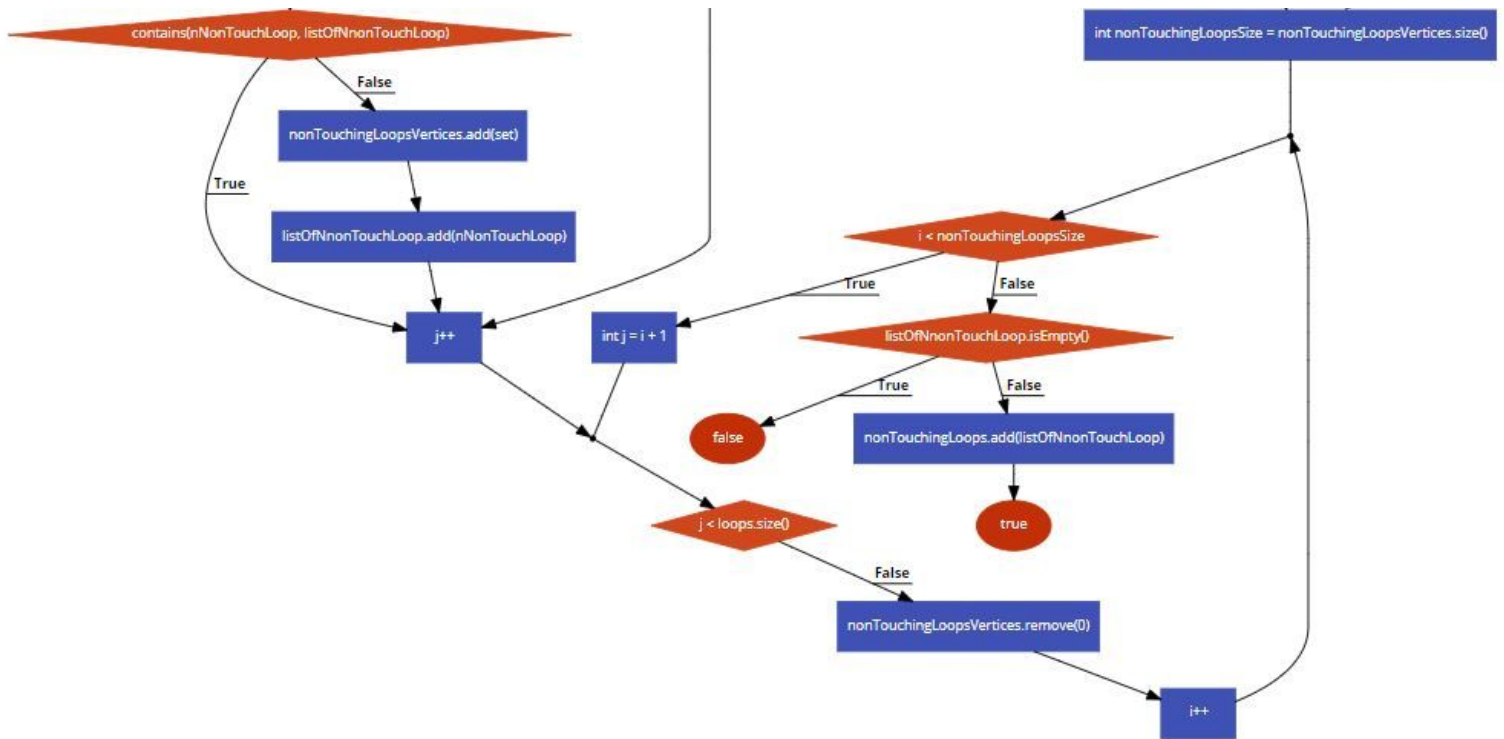
## 7) Graph Solver - Forward Path Delta:



## 8) Graph Solver - Find N non touching Loops:



Continue previous function:



## Assumptions

- Nodes are numbered from 1 to the input number of nodes entered by the user.
- 'To' and 'From' nodes must be greater than two to construct a graph.
- There can't be an edge with the same 'To' and 'From' nodes in the same direction as a previously entered one.
- Gain could be positive, negative or any real number except for the zero.

## GUI

### • FEATURES

- **Automatic** Graph **display** of the user input data
- **Reading** User graph from **file** (.txt extension is primited)
- Re-constructing the graph nodes/edges manually after drawing
- Displaying meaningful **error messages** on Invalid input
- Buttons are disabled until input is valid to prevent unexpected exceptions.

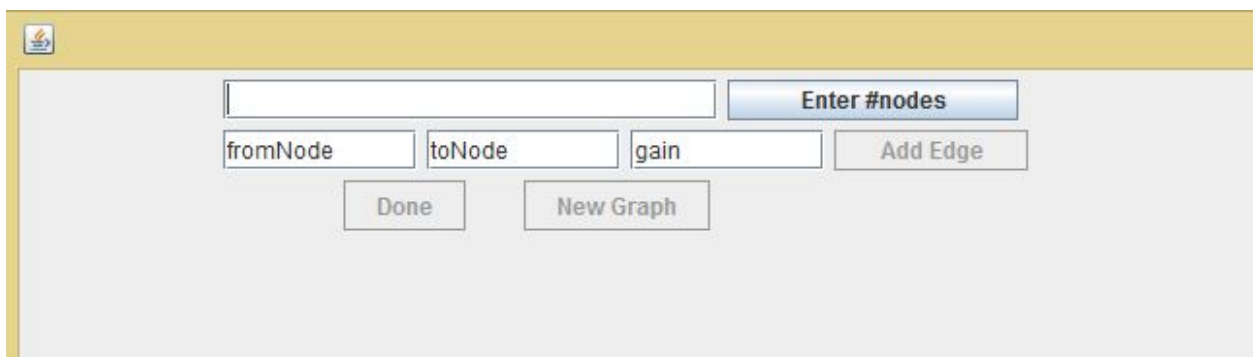
### • USER INTERFACE

- User should enter the number of nodes of the graph.
- The Add Edges button is now enabled and the user can enter the edges one by one (note:edge must contain to, from and gain)
- On pressing Done first the graph is painted and the user has the ability to add new edges or clear the current graph and make a new one.
- A table appears containing all loops and paths with their information from gain and vertices composing these loops/paths.
- User can upload a txt file to be read matching the following format :



- First line containing the number of nodes 'N'.
- The following at least 'N - 1' line contains the 'To', 'From' and 'Gain' of each edge separated by a space.
- To draw a new graph the user must clear the previously added graph by pressing 'New Graph' button.

## ● RUNNING SAMPLES



The screenshot shows the initial state of the graph application. It features a yellow header bar with a small icon on the left. Below the header, there is a light gray area containing several input fields and buttons. At the top, there is a text input field followed by a blue button labeled 'Enter #nodes'. Below this, there are three text input fields labeled 'fromNode', 'toNode', and 'gain', followed by a blue button labeled 'Add Edge'. At the bottom of this section, there are two gray buttons labeled 'Done' and 'New Graph'.

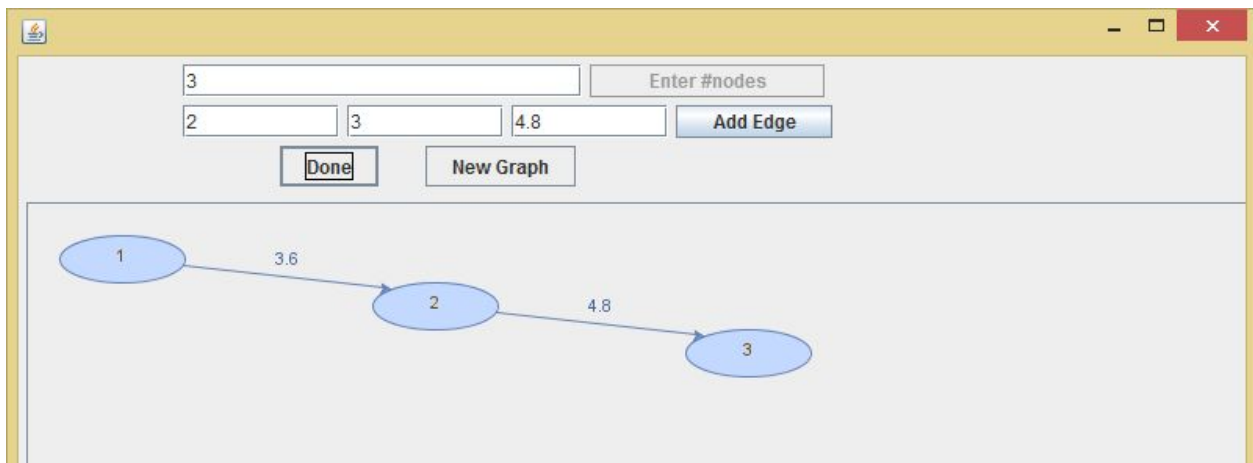


Table		
Transfer Function Gain= 17.28		
Type	Path/Loops	Gain
Forward Path '1'	123	17.28

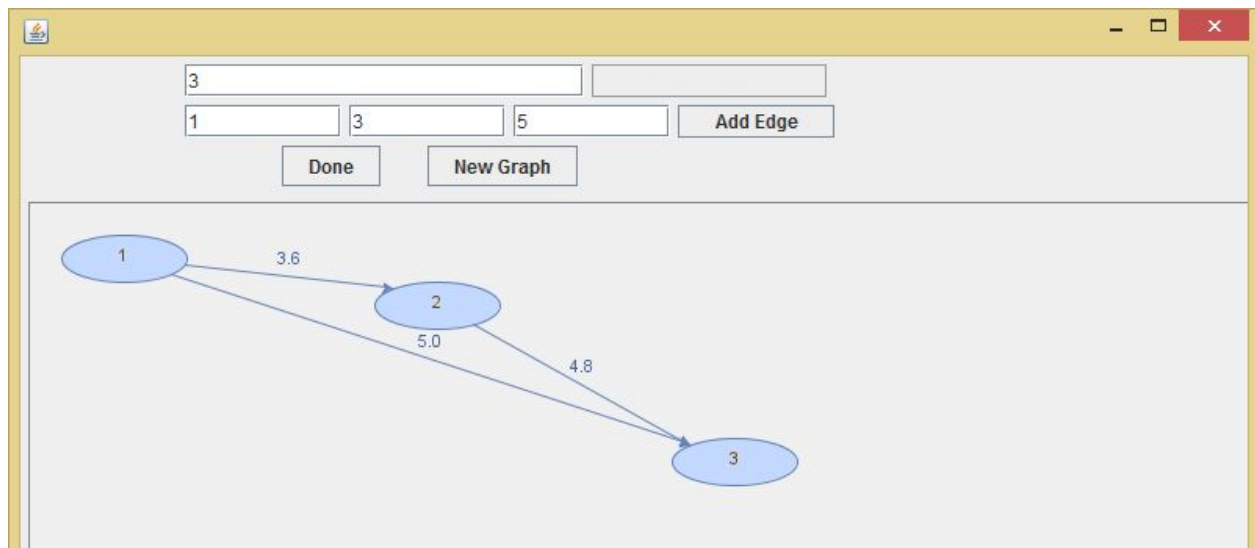
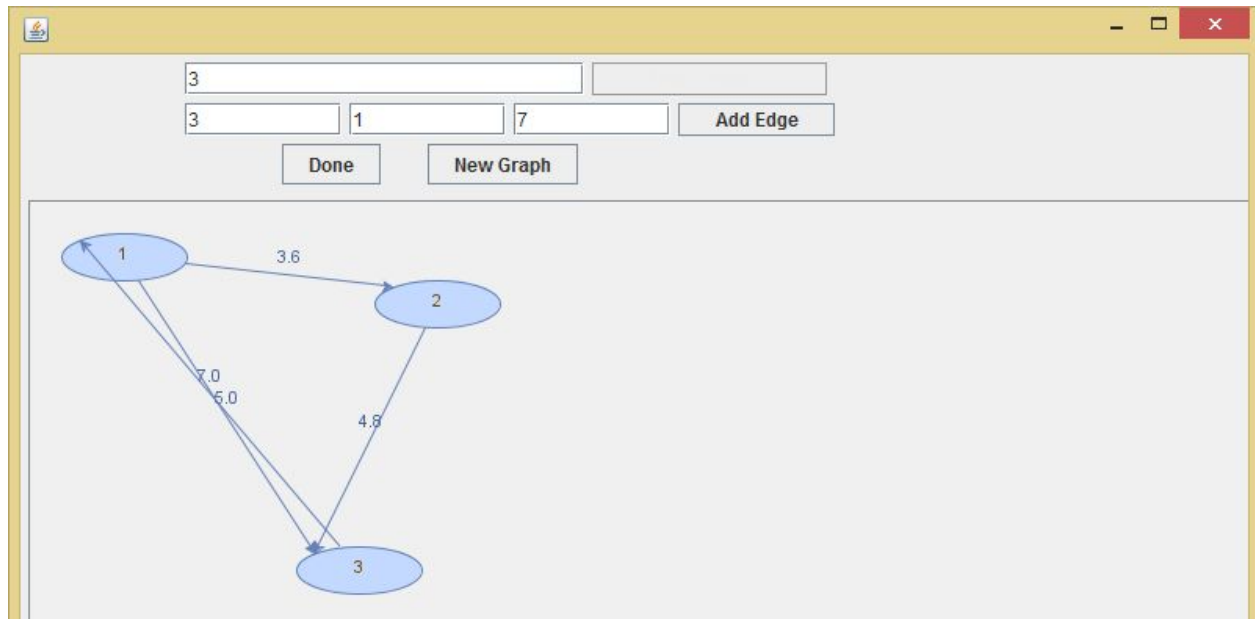


Table		
Transfer Function Gain= 22.28		
Type	Path/Loops	Gain
Forward Path '1'	123	17.28
Forward Path '2'	13	5.000000000000001



Table

Transfer Function Gain = -0.1437790397521941

Type	Path/Loops	Gain
Forward Path '1'	123	17.28
Forward Path '2'	13	5.000000000000001
Loop '1'	1231	120.96000000000001
Loop '2'	131	35.00000000000001

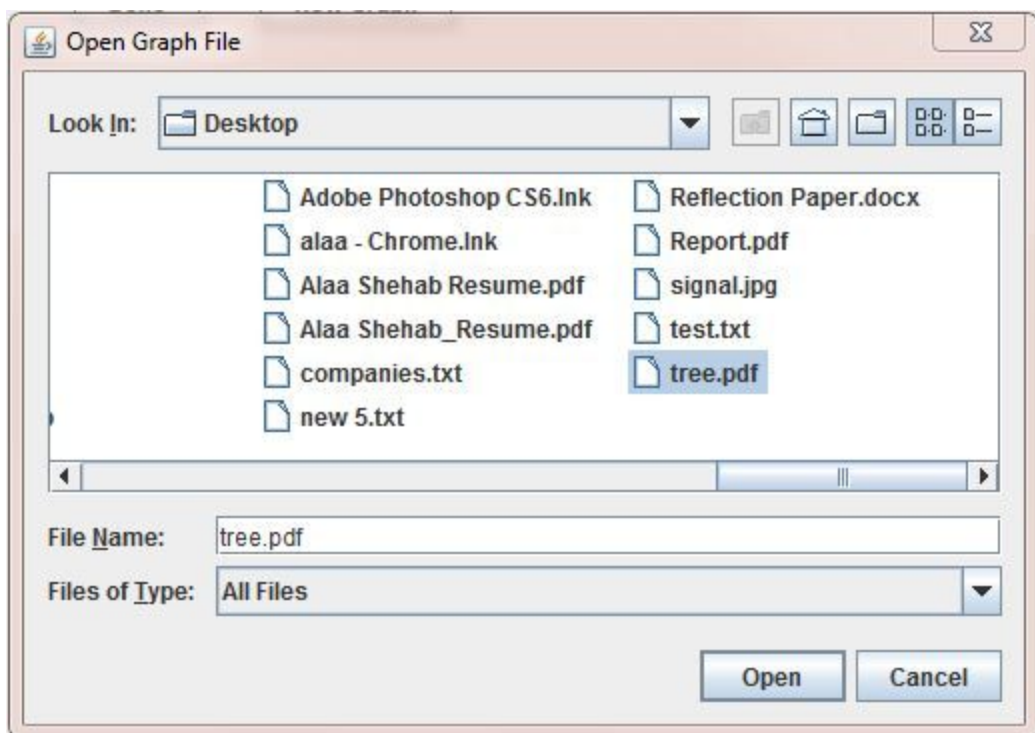
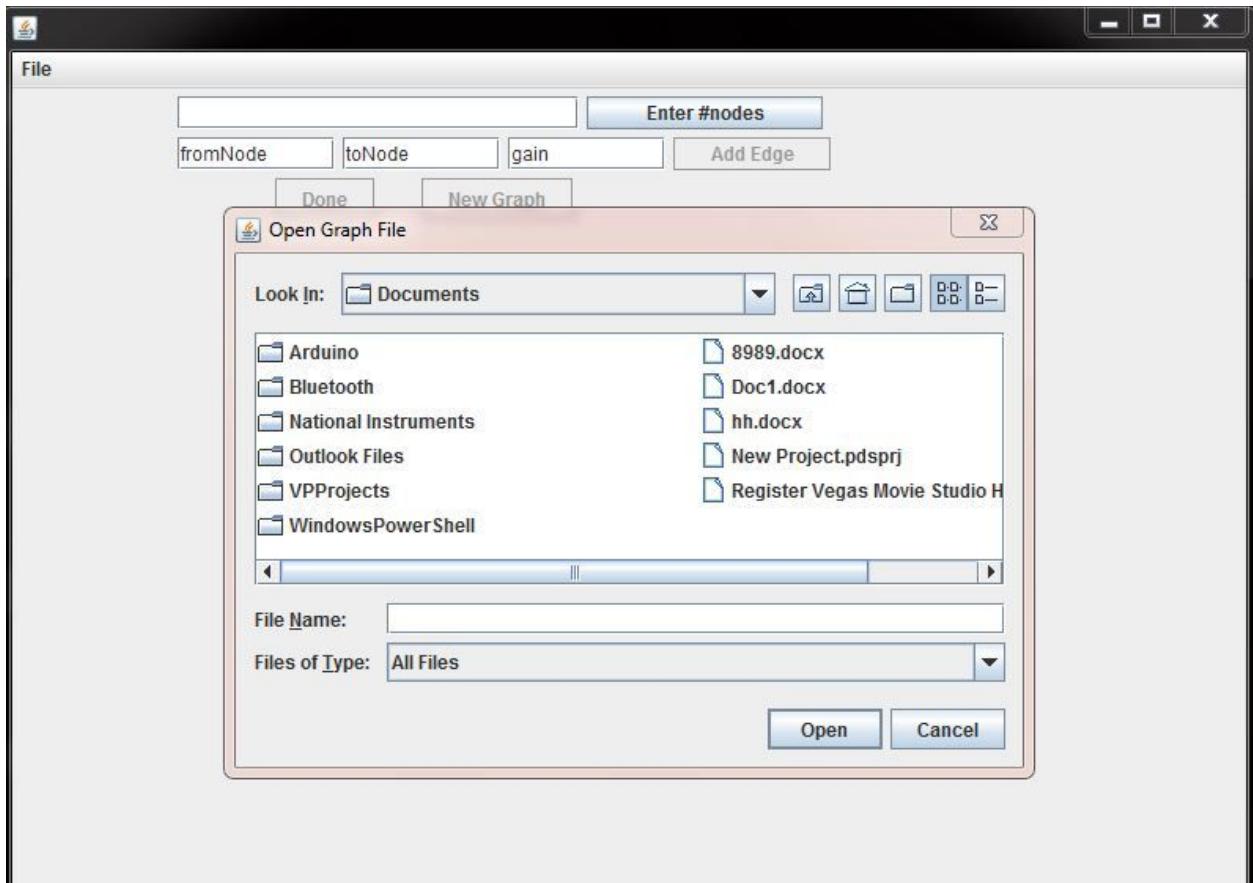
File

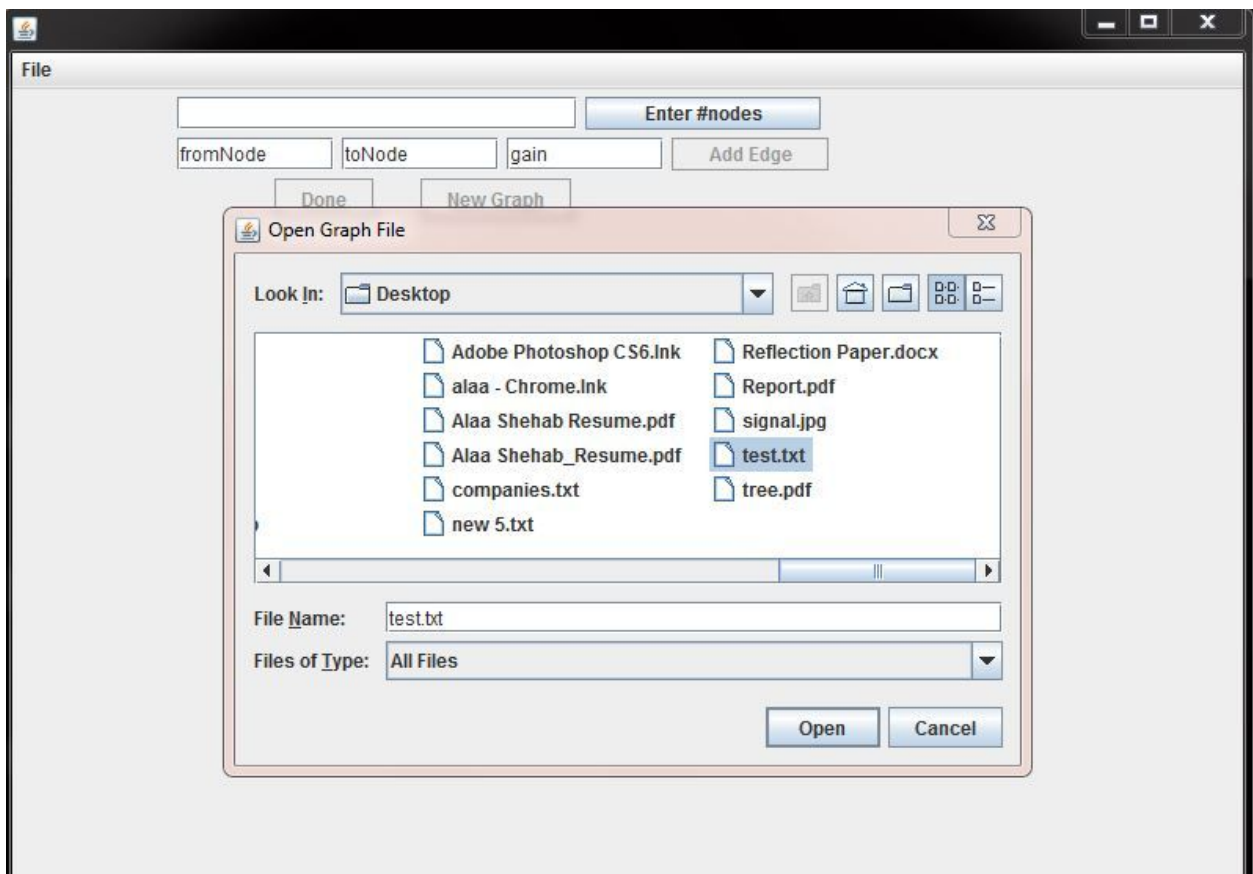
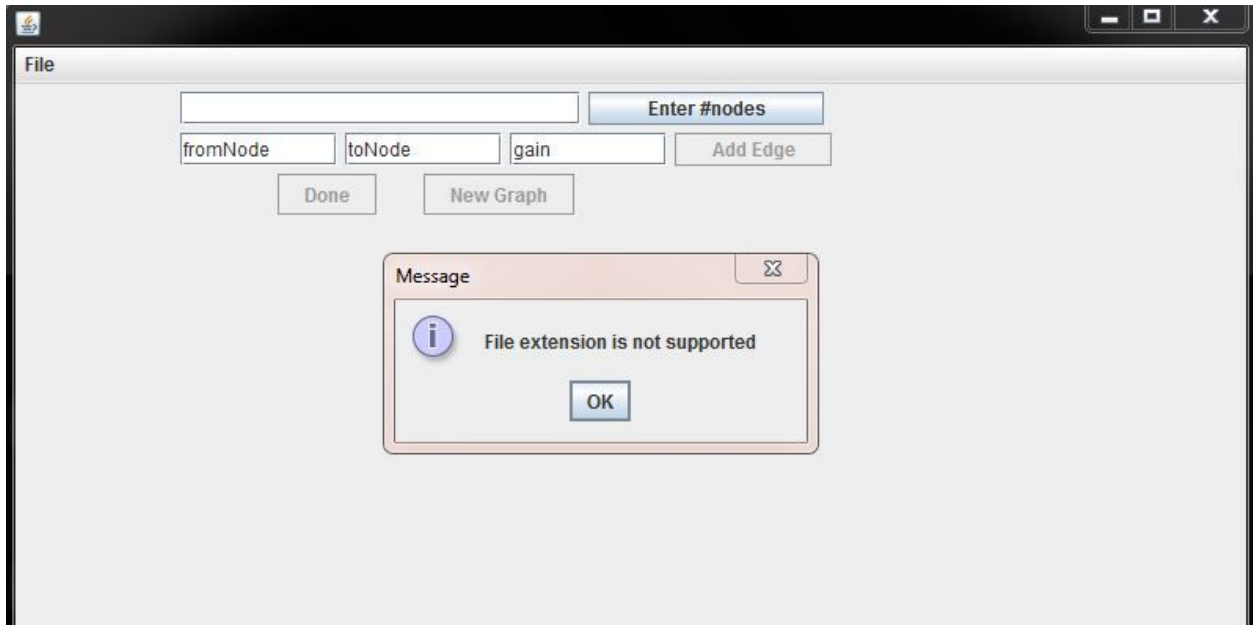
Add Graph

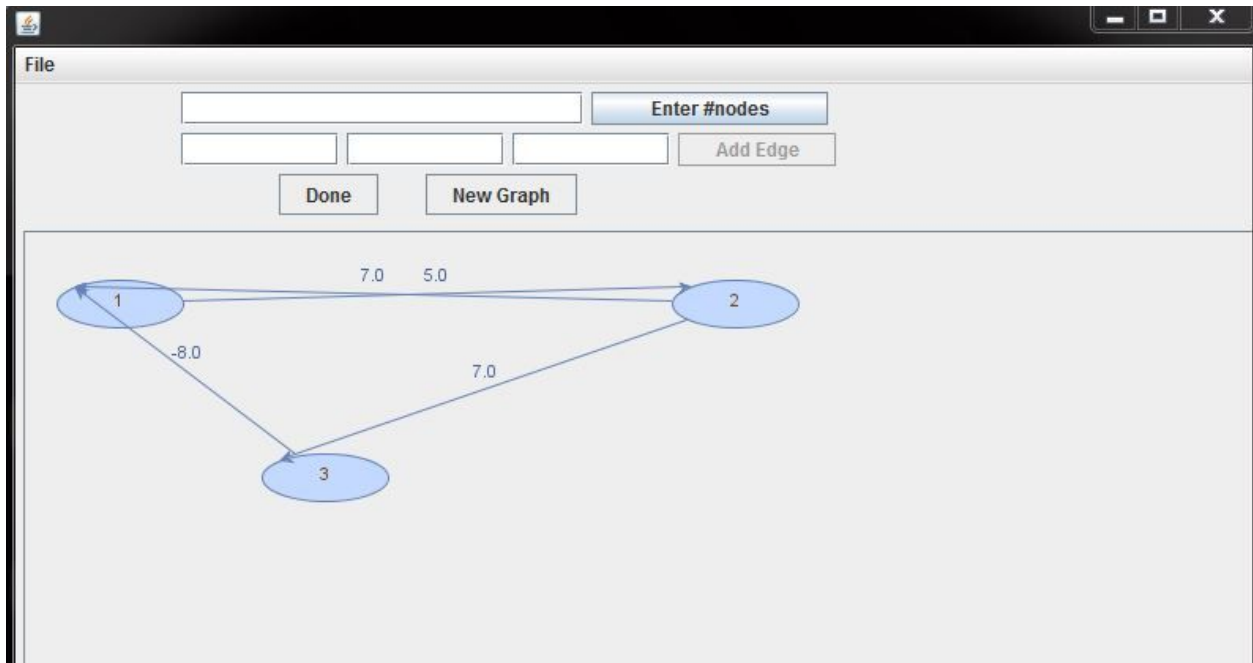
Enter #nodes

fromNode toNode gain Add Edge

Done New Graph







File

4 Enter #nodes

1 5 -5 Add Edge

Done

New Graph

Message

i Node must be smaller than 4

OK

