Team Members

1-Ahmed Reda Amin ID:9

2-Mohamed Elsayed Helmy ID:60

Signal Flow Graph.

Signal flow graph is a systematic way to calculate the overall transfer function for a system.

And this way calculate overall transfer function using “Mason Formula”

$$G = \frac{y_{\text{out}}}{y_{\text{in}}} = \frac{\sum_{k=1}^{N} G_k \Delta_k}{\Delta}$$

$$\Delta = 1 - \sum L_i + \sum L_i L_j - \sum L_i L_j L_k + \cdots + (-1)^m \sum \cdots + \cdots$$

where:

- Δ = the determinant of the graph.
- y_{in} = input-node variable
- y_{out} = output-node variable
- G = complete gain between y_{in} and y_{out}
- N = total number of forward paths between y_{in} and y_{out}
- G_k = path gain of the kth forward path between y_{in} and y_{out}
- L_i = loop gain of each closed loop in the system
- L_iL_j = product of the loop gains of any two non-touching loops (no common nodes)
- $L_iL_jL_k$ = product of the loop gains of any three pairwise nontouching loops
- Δ_k = the cofactor value of Δ for the k^{th} forward path, with the loops touching the k^{th} forward path removed. *

And this make it easier to calculate overall gain than the block diagram.

And we can make a program to calculate gain as we do but in block diagram we can not do this.

Main Features.

This program enable user to enter his system .

He enter input ,output and gain for this transformation.

Datastructure.

1-array of array list *to store transformation between nodes “adjacency matrix”*

2-two dimensional array *to store gain for each transformation between 2 nodes.*

3-stack of array list *to store paths between input and output node, and another one to store individual loops.*

4-stack of stack of array list *to store all loops (individual ,two non touched loops , three non touched loops,……..)*

5-one dimensional array *to store gain for each forward path.*

6-array list *:to store one forward path.*

Main Modules

1-Forward Path: *it extract all forward paths from input node to the output node and store them in an array list , and also using it to extract individual loops.*

```
private void forwardPaths(int i, int j) {
            if (adjacencyMatrix[i].get(j) == outNode) {
                  oneForwardPath.add(outNode);
                  paths.push(new ArrayList(oneForwardPath));
                  oneForwardPath.remove(adjacencyMatrix[i].get(j));
            } else {
                  if (visitedNodes[adjacencyMatrix[i].get(j)] == false) {
                        visitedNodes[adjacencyMatrix[i].get(j)] = true;
                        oneForwardPath.add(adjacencyMatrix[i].get(j));
                        forwardPaths(adjacencyMatrix[i].get(j), 0);
                        visitedNodes[adjacencyMatrix[i].get(j)] = false;
                        oneForwardPath.remove(adjacencyMatrix[i].get(j));
                  } else {
                        loops(adjacencyMatrix[i].get(j));
                  }
```

```
            }
            for (int counter = 1; j == 0 && counter <
adjacencyMatrix[i].size(); counter++) {
                  forwardPaths(i, counter);
            }
      }
```

2-Loops: *add an individual loop to a stack which contain all individual loops.*

```
      private void loops(int value) {
            boolean flag = true;
            int index = 0;
            ArrayList<Integer> loop = new ArrayList(oneForwardPath);
            loop.add(value);
            while (flag == true) {
                  if (loop.get(index) != loop.get(loop.size() - 1)) {
                        loop.remove(index);
                  } else {
                        flag = false;
                  }
            }
            individualLoops.push(loop);
      }
```

3-Non Touched Loops:

This module extract (two non touched,three non touched,.....)

And we can say this is the most important module in this program.

```
private void nonTouched(int i, Stack<ArrayList<Integer>> temp,Stack<Stack<ArrayList<Integer>>> table,Stack<ArrayList<Integer>> individual) {
    if (i == individual.size()) {
        if (temp.size() >= 2) {
            boolean flag = false;
            if (temp.size() - 1 <=  table.size()) {

                for (int counter = 0; counter < temp.size() && !flag; counter++) {
                    for (int j = 0; j < visitedNodes.length; j++) {
                        visitedNodes[j] = false;
                    }
                    for (int count = 0; count < temp.get(0).size(); count++) {
                        visitedNodes[temp.get(counter).get(count)] = true;
                    }
                    for (int j = counter + 1; j < temp.size() && !flag; j++) {
                        for (int count = 0; count < temp.get(j).size(); count++) {
                            if (visitedNodes[temp.get(j).get(count)]) {
                                flag = true;
                                break;
                            }
                        }
                    }
                }
                if (!flag) {
                    for (int j = 0; j < temp.size(); j++) {
                        if (temp.size() >  table.size()) {
                            table.add(new Stack<ArrayList<Integer>>());
                        }
                        table.get(temp.size() - 1).add(new ArrayList<Integer>(temp.get(j)));
                    }
                }

            }
        }
        return;
```



```
}
nonTouched(i + 1, temp,table,individual);
temp.add(new ArrayList<Integer>(individual.get(i)));
nonTouched(i + 1, temp,table,individual);
temp.remove(temp.size() - 1);
```

4-Sum Of Gain

This module calculate the some of gain for each type of loops

Like sum of gain of individual loops, and sum for two non touched ,.....

```
private float sumOfGain(Stack<ArrayList<Integer>> temp, int index) {
            float gain = 1;
            float tempgain = 1;
            float gainSum = 0;
            if (temp != null) {
                  for (int i = 0; i < temp.size(); i += index) {
                        for (int k = 0; k < index; k++) {
                              for (int j = 0; j < temp.get(i + k).size() - 1;
j++) {
```

```
                                    gain *= weight[temp.get(i + k).get(j)]
[temp.get(i + k).get(j + 1)];
                              }
                              tempgain *= gain;
                              gain = 1;
                        }
                        gainSum += tempgain;
                        gain = 1;
                        tempgain = 1;
                  }
            }
            return gainSum;
      }
```

5-Delta

this part calculate delta using some of gain module

```
      private float delta(Stack<Stack<ArrayList<Integer>>> temp) {
            float delta = 0;
            if (temp != null) {
                  int factor = -1;
                  for (int i = 0; i < temp.size(); i++) {
                        delta += (sumOfGain(temp.get(i), i + 1) * factor);
                        factor *= -1;
                  }
            }
            return delta + 1;
      }
```

6-Overall Transfer Function:

this part calculate the gain of overall transfer function .

```
      private float overAllTransferFunction(){
            gainOfForwardPaths();
            float sum=0;
            float delta;
            for (int i=0;i<paths.size();i++){
                  sum+=(gainOfPaths[i]*calculateSecondaryDelta(i));
            }
            delta=delta(allLoops);
            if (delta!=0){
                  return sum/delta;
            }
             return Float.MAX_VALUE;
      }
```

7-Controller

Control the flow of the program

```
private void controller(){
    forwardPaths(inputNode, 0); // first element in the array of input node
    removeRepeatdLoops(); // after extracting loops and paths
    allLoops.add(individualLoops); //add individual loops to the table of loops.
    nonTouched(0, new Stack<ArrayList<Integer>>(),allLoops,individualLoops); //
}
```

Algorithms

1-Recursion And Back Track

This algorithm used in forward path module

And this happens when make the node visited and then make it un visited.

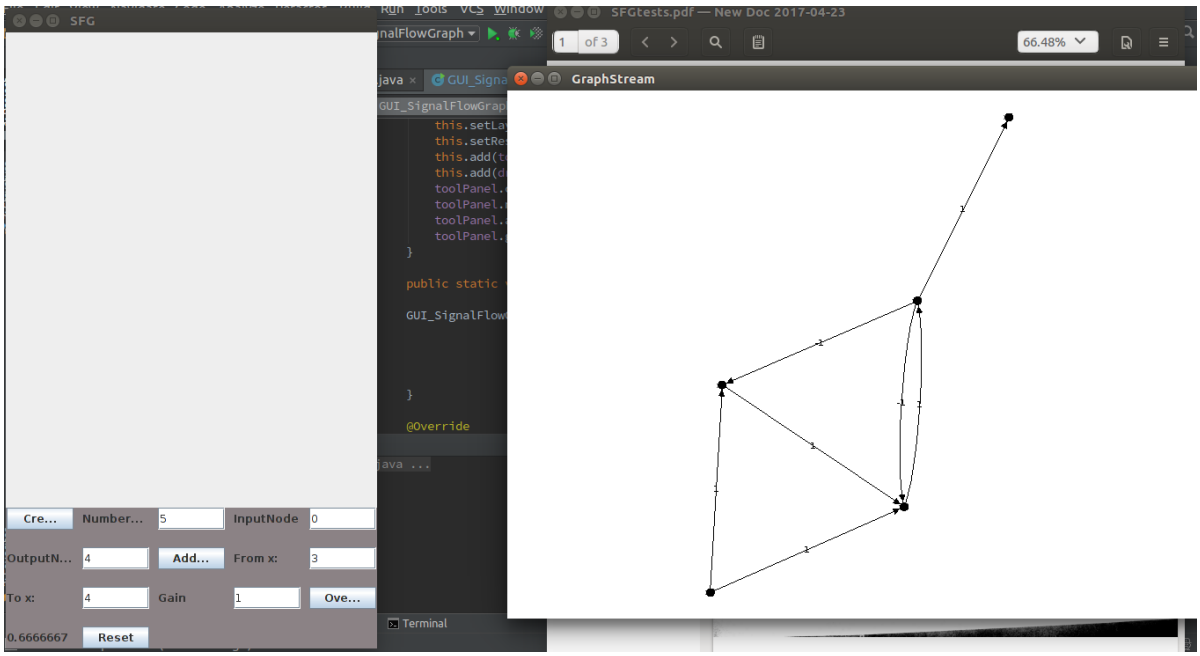
2-Complete Search

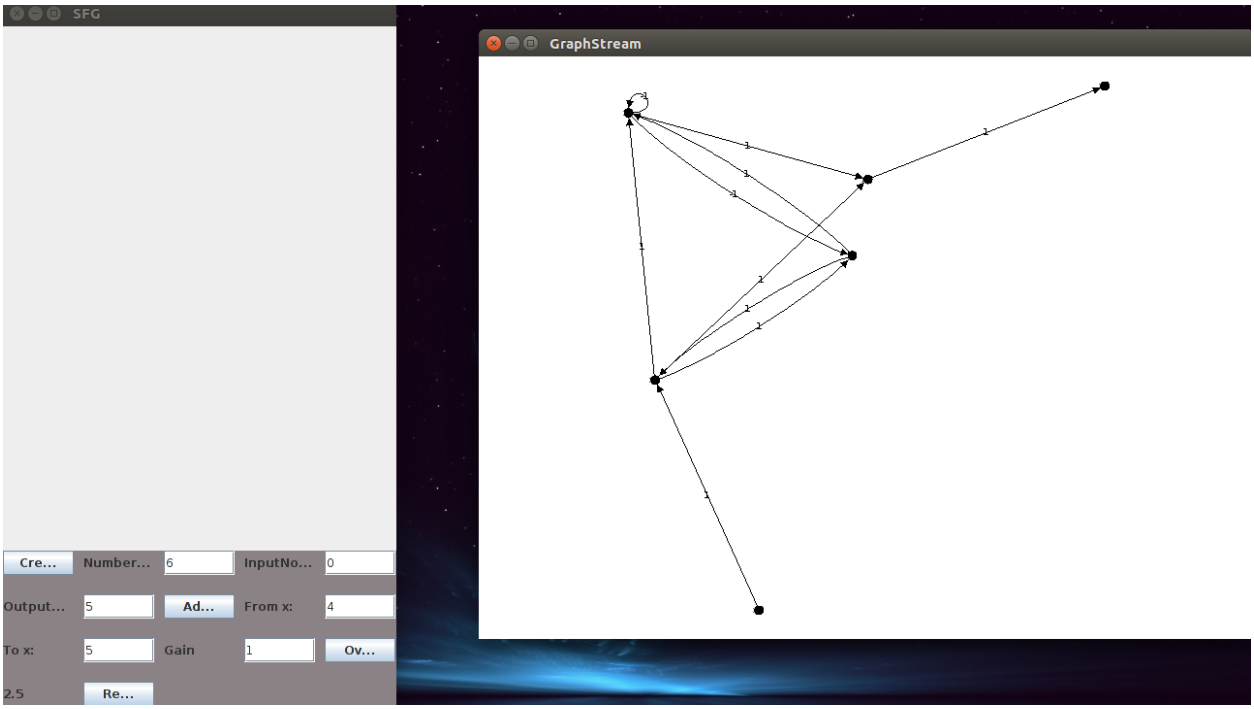
This algorithm used in nontouched loops module

As we take all 2 power n combinations of individual loops ,where n number of individual loops

For each combination we check if they can be nontouched with each other or not.

Sample Runs:





Simple User Guide:

1 – User specifies number of nodes of the graph , the index of the input node and the index of the output node and then presses on the create button.

2- User then manually constructs all edges by putting the from and to index and the gain of the edge .

3- Finally the User presses the Overall TF button to calculate the Mason Formula of the SFG which is displayed in a label next to it .

4- The user can add another graph by pressing the reset button which resets everything back to its original form.