2016

Assigment 1: Routh Criteria

Names :

Safaa Hssan wally (34)

Rewan Alaa Eldin (23)

4/14/2016

Problem statement:

Signal flow graph representation of the system

Main feature of the program:

- Graphical interface.
- signal flow graph showing nodes, branches, gains

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

- the value of delta to delta(m) where m is number of forward path

- Overall system transfer function

- **Additional option : the user can choose the sink node during calculation of forward path**

Data Structure used:

- 2D Array during determination of loops
- List for storing the groups non touching loops and for find and storing forward path.

Main modeuls:

class:

1- AllCycle: this class used to get cycle in graph.

method in this class:

```
a-private List<Object> calculateCost(List<Object>
cycles)//this method calculate the cost of loop
```

```
b-public List<Object> getCombination()
```

```
//this method calculate n-nontouching loop
```

2- StrongConnectedComponents:this class used to help in finding the loop in graph

method in this class:

```
a-private Vector getLowestIdComponent()
```

// this method find the component of lowest id node

---

b-private void getStrongConnectedComponents(**int** root)//get strong connected component from root node

3- CyclesSearch:this class used Johnson algorithm to find the loop in graph.

method in this class:

a-public List getElementaryCycles()

//this method prepare the data need for another method to find cycle.

b-private boolean findCycles(**int** v, **int** s, Vector[] adjList)

//this method take data from getElementaryCycles()

to find loop

4- Mason**:this class used to get forward path**

a-public find(int s, string path)

//this method get all the forward paths in tthe graph

Algorithms used :

// Forward Path Algorithm

```
findForwardPath (int s, String path) {
IF (s == sink) {
forwardPaths.add(path);
return;
}END IF
parent = s;
For (i=0  →  signalFlowGraph.size) {
IF (!visited.get(signalFlowGraph[i].getFirst())) {
visited.set(signalFlowGraph[i.getFirst(), true);
find(i.getFirst(), path + ", " + i.getFirst());
parent = s;
visited.set(signalFlowGraph[i].getFirst(), false);
}END IF
}END FOR
}
```

//Johnson algorithm to find all loops

```
sccs: input that make graphs groups of strongly connected component
Int s = 0

while (true) {
sccResult = sccs.getAdjacencyList(s)
if (sccResult != null && sccResult.getAdjList() != null) {
Vector[] scc = sccResult.getAdjList()
s = sccResult.getLowestNodeId()
for j = 0 to scc.length{
if ((scc[j] != null) && (scc[j].size() > 0)) {
blocked[j] = false
B[j] << array
}
}
findCycles(s, s, scc)
s = s+1
} else {
break
}
}
return cycles
```

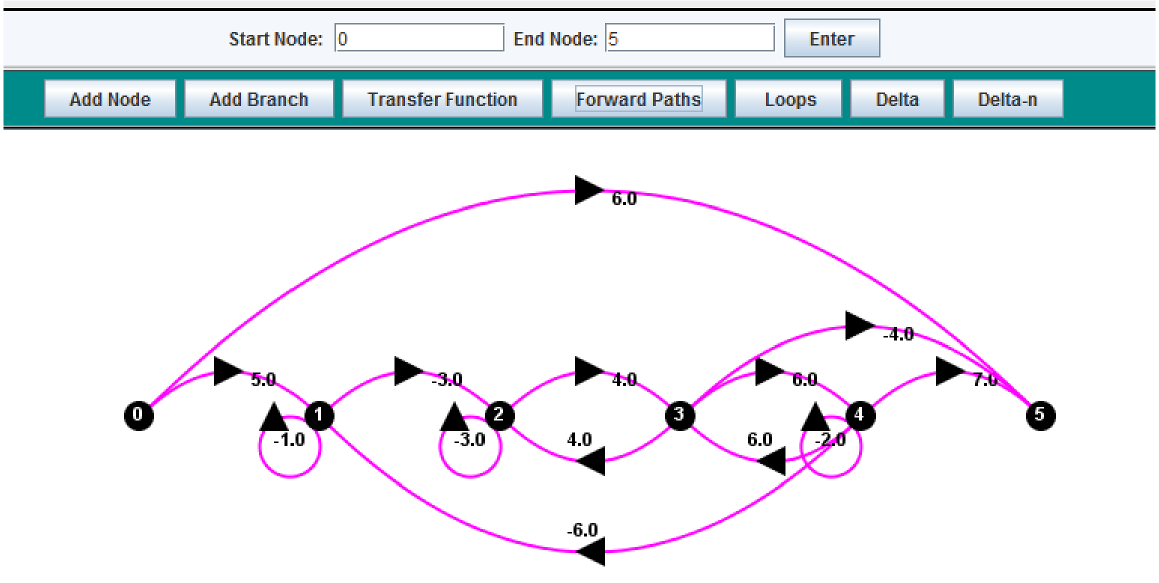
```
findCycles(int v, int s, Vector[] adjList) {

    boolean f = false
    stack.add(v)
    blocked[v] = true

    for i = 0 to adjList[v].size(){
        int w = adjList[v].get(i)
        found cycle
        if (w == s) {
            Vector cycle
            for j = 0 to stack.size(){
                int index = stack.get(j)
                cycle.add(index)
            }
            cycles.add(cycle)
            f = true
        } else if (!blocked[w]) {
            if (findCycles(w, s, adjList)) {
                f = true
            }
        }
    }
    if (f) {
        unblock(v)
    } else {
        for i = 0 to adjList[v].size(){
            int w = adjList[v].get(i))
            if (!B[w].contains(v)) {
                B[w].add(v)
            }
        }
    }

    stack.remove(v)
    return f;
}

unblock(int node) {
    blocked[node] = false
    Vector Bnode = B[node]
    while (Bnode.size() > 0) {
        Integer w = Bnode.get(0);
        Bnode.remove(0)
        if (blocked[w]) {
            unblock(w)
        }
    }
}
```

// algorithm to calculate delta

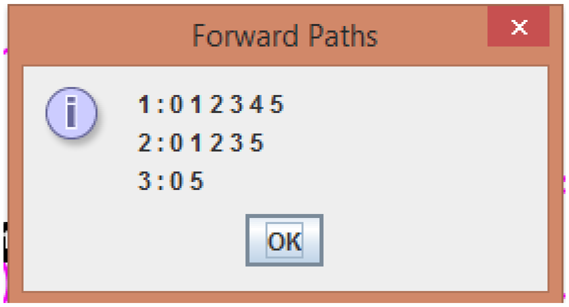
```
findDelta(){
delta = 1;
for i = 0 to i < cycle.getAllCost().size(){

List group = cycle.getAllCost().get(i)
double cost = 0
for j = 0 to j < group.size(){
cost = cost+ group.get(j)
}
if(i%2 == 0){
delta = delta - cost;
}
else{
delta = delta + cost;
}
return delta;
```

Sample runs:

First Example:

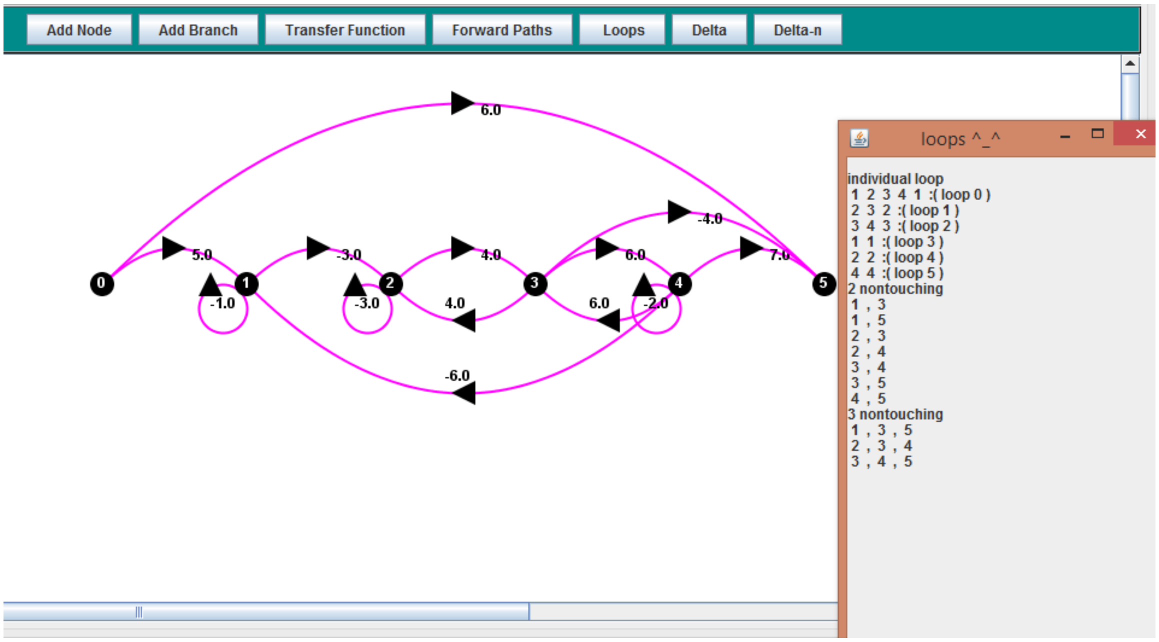


Start Node: 0 End Node: 5

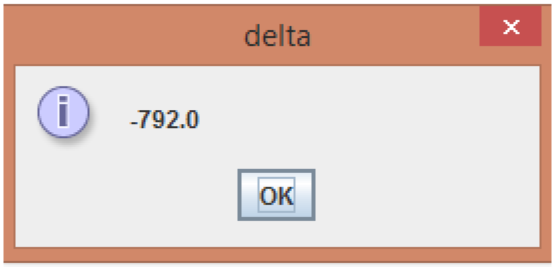


Forward Paths

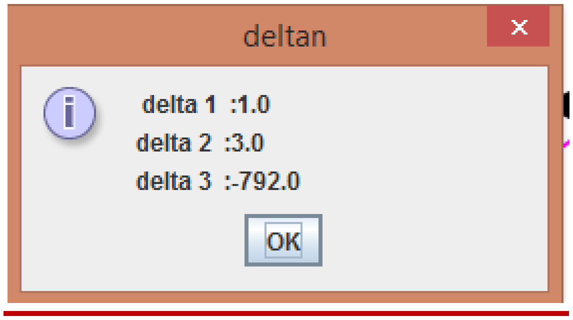
1:0 1 2 3 4 5
2:0 1 2 3 5
3:0 5



```
individual loop
 1  2  3  4  1  :( loop 0 )
 2  3  2  :( loop 1 )
 3  4  3  :( loop 2 )
 1  1  :( loop 3 )
 2  2  :( loop 4 )
 4  4  :( loop 5 )
2 nontouching
 1  ,  3
 1  ,  5
 2  ,  3
 2  ,  4
 3  ,  4
 3  ,  5
 4  ,  5
3 nontouching
 1  ,  3  ,  5
 2  ,  3  ,  4
 3  ,  4  ,  5
```

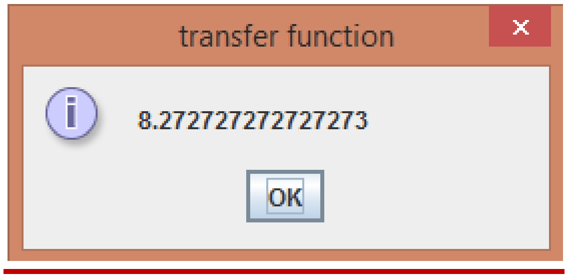


```
-792.0
```



deltan

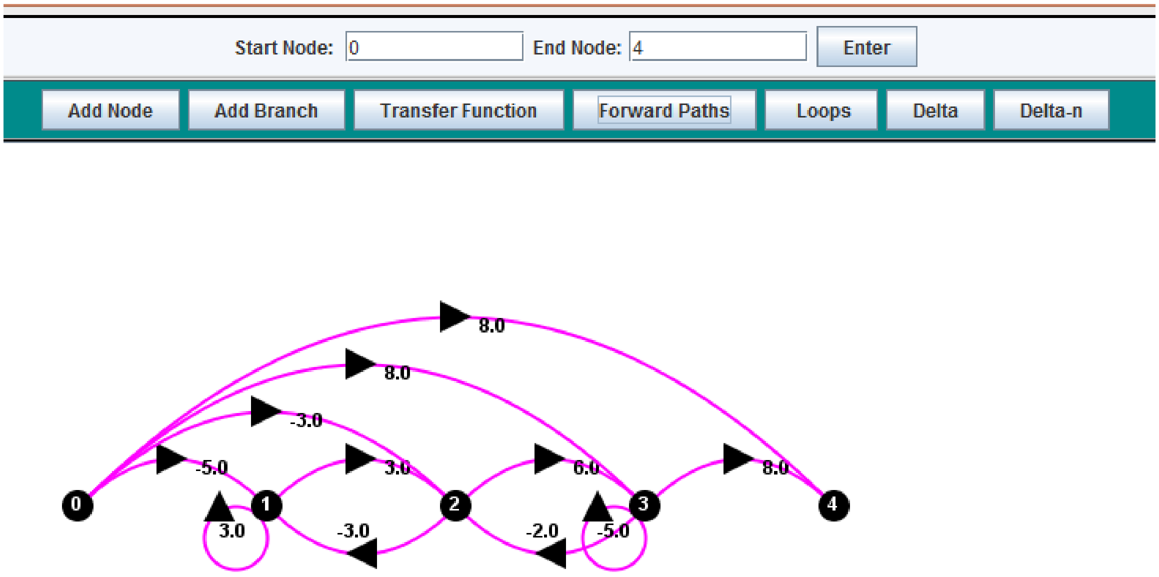
delta 1 :1.0
delta 2 :3.0
delta 3 :-792.0



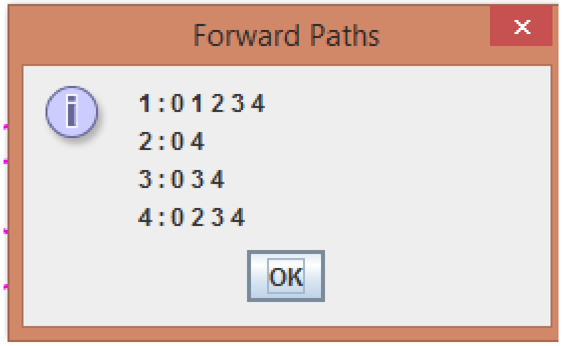
transfer function

8.272727272727273

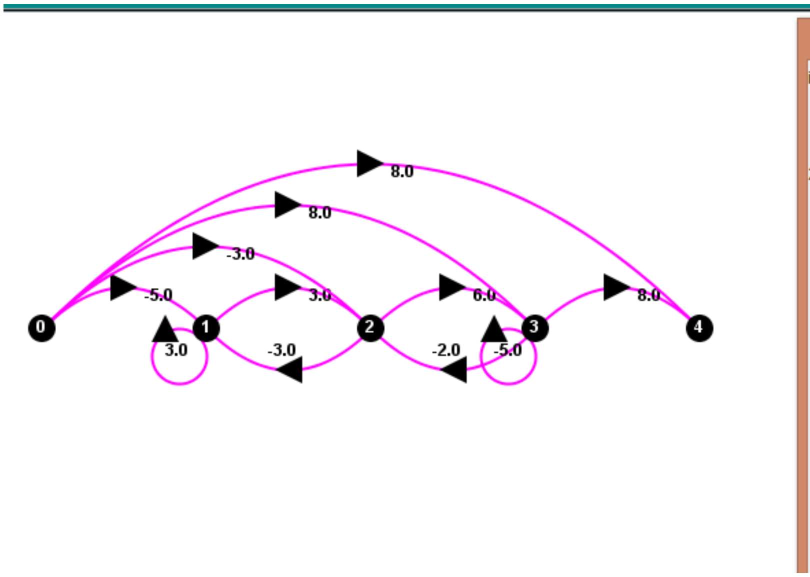
second Example:

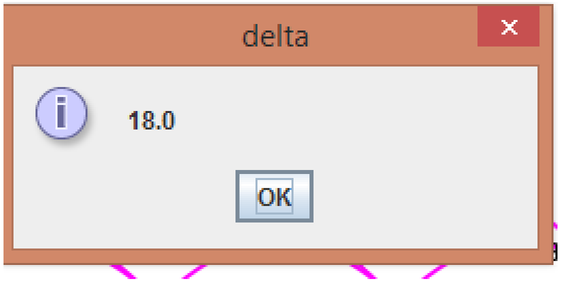


Start Node: 0 End Node: 4

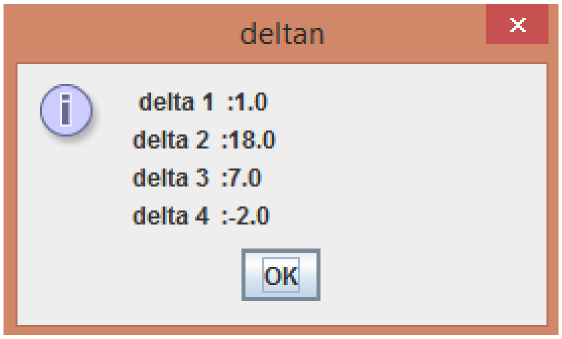


```
1:01234
2:04
3:034
4:0234
```

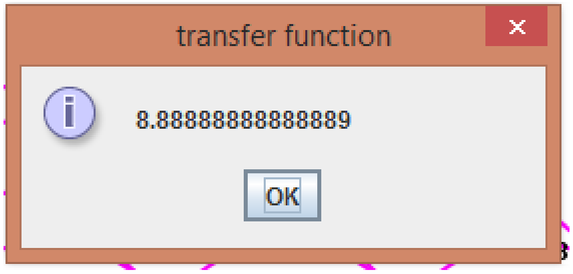




```
18.0
```



```
delta 1 :1.0
delta 2 :18.0
delta 3 :7.0
delta 4 :-2.0
```



```
8.88888888888889
```

assumption:

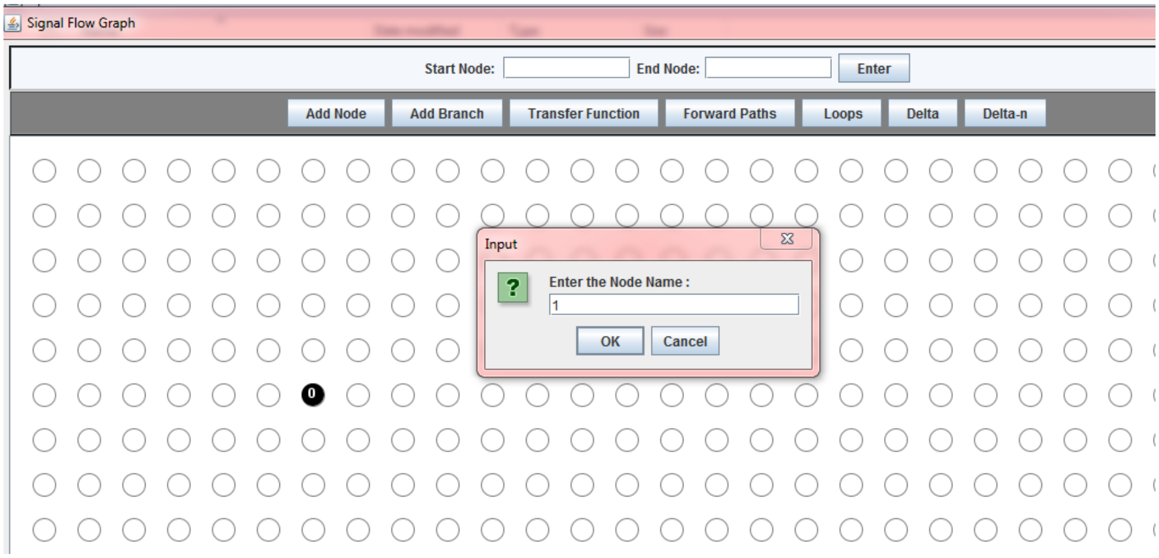
1- you can't exceed extra node after your calculation so if you need to run another example you must run the program again.

2- your start node must be source node but you can choose any node to be your sink node.

Simple user guide :

Add Node:

you press the button add node then choose the position through the guide circles then enter the name they shouldn't be duplicated.



Signal Flow Graph

Start Node:

End Node:

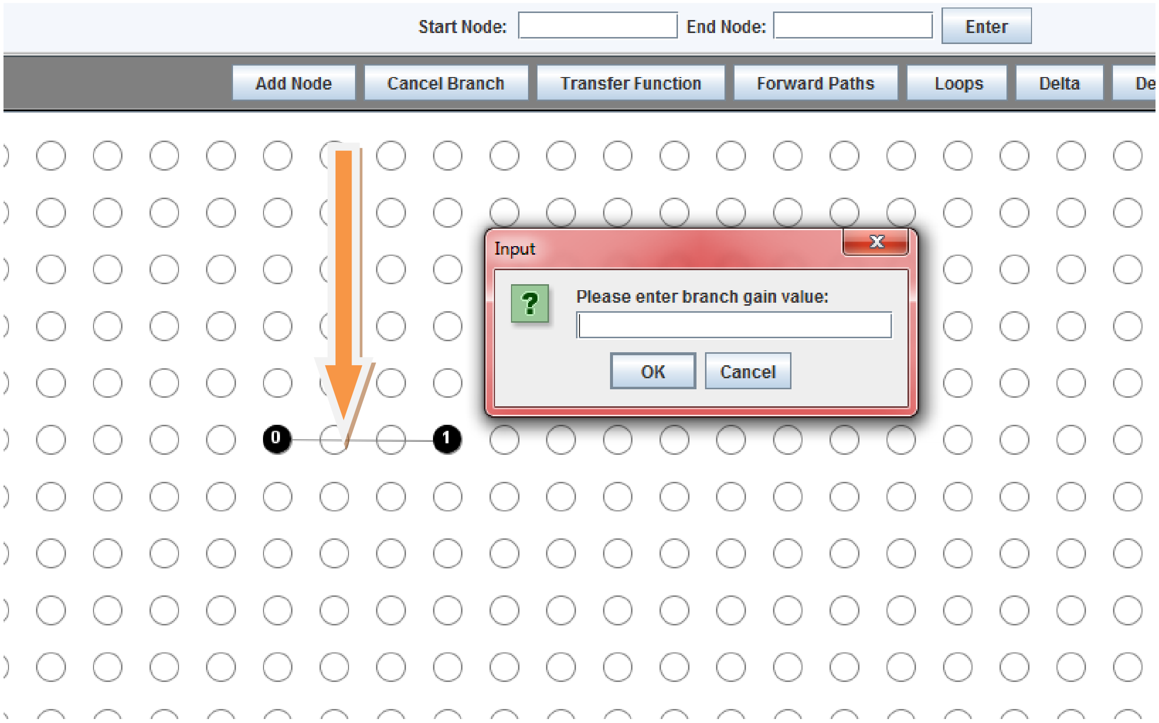
Input

Enter the Node Name :

1

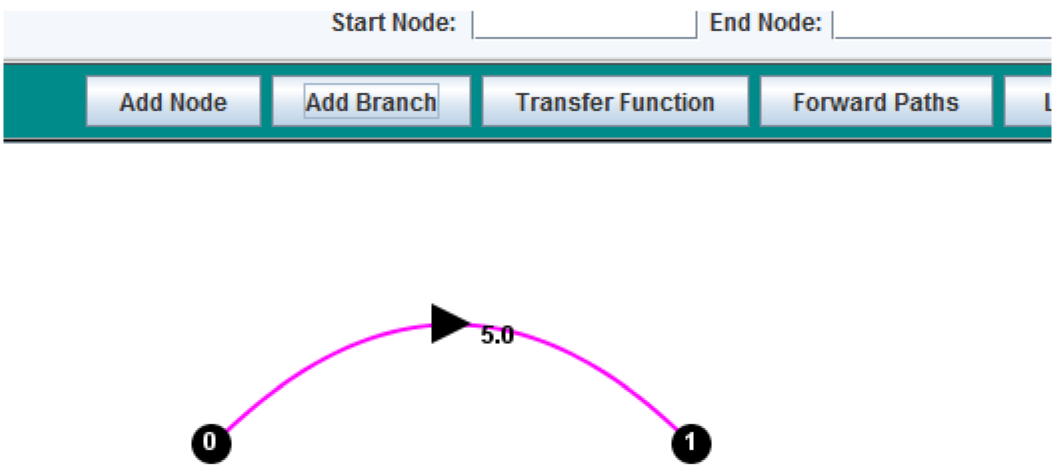
Add Branch:

You press the button [Add Branch] then choose the first position and second position through the guide circles then enter the gain



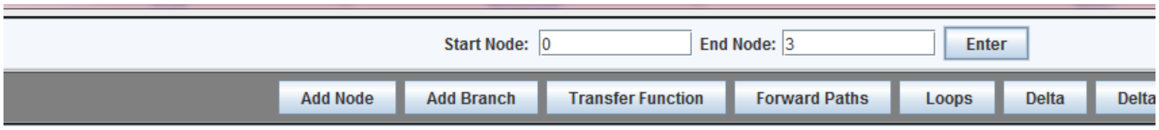
Please enter branch gain value:

Hint :an arrow guide will be drawn while drawing branch to show the start and the end



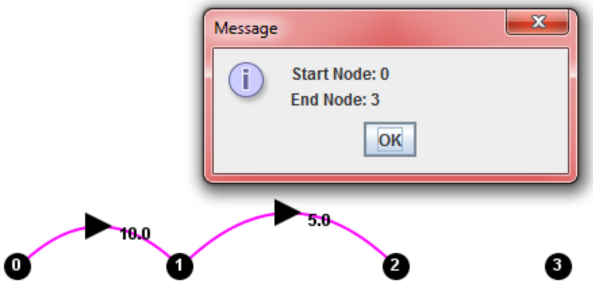
Enter the source Node and sink Node:

You enter the first node and then end node to get the transfer function make sure that the start node is source and the end one is sink then press enter



Start Node: 0

End Node: 3



Message

Start Node: 0
End Node: 3

The transfer function:

press button transfer function to get it .

Forward Paths:

to get forward paths press button forward paths.

Loops:

to get Loops (individuals and n-non touching loops) press button loops note that in n-non touching loop the number is the number of loop .

Delta and Delta-n:

to get delta and delta -n press button .