



---

# CSANALYSIS

---

Ahmed Magdy Ibrahim

ID: 3003



## **Introduction:**

The Control System Analysis (CSAnalysis) Program is made in order to analyze systems represented by Signal Flow Graph or Block diagram.

## **Technical Overview:**

The program is based on Graphs as data structure to store the parsed data taken from the system representation. Also based on the back tracking algorithm as a traversal through all the nodes calculating both the forward paths and the loops then storing them in specific class named Path holding lists of the indexes of the nodes representing the path and its values.

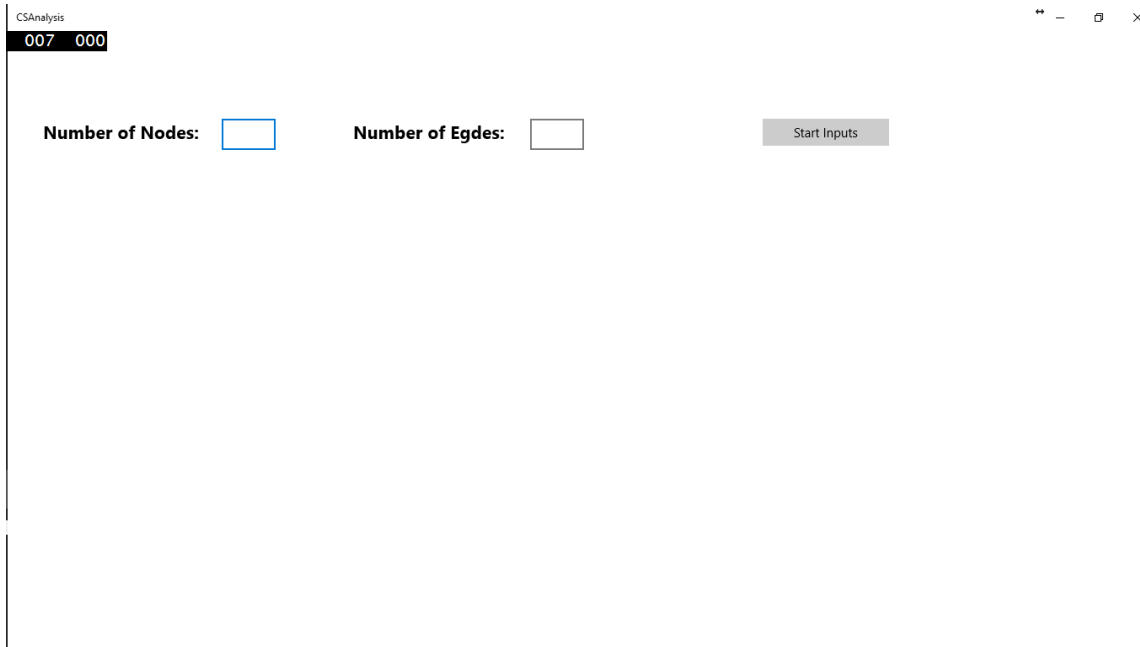
For the non-touching loops, a comparing algorithm is used to compare the indexes of the nodes representing each path and detecting if they are touched or not and storing the result in a matrix representing the relation between the paths.

## **Technical Details:**



The program starts with a screen to choose between Signal Flow Graph or Block Diagram.

**Note: Block Diagram is not implemented yet.**



For the Signal Flow Graph the program will ask for the number of nodes and the number of edges in your graph this will help the program initialize the Graph

```
1 reference | Ahmed Magdy Ibrahim, 37 days ago | 1 author, 3 changes
private void StartInput()
{
    theGraph = new Graph(NodesNum, EdgesNum);
    EdgesLeft = EdgesNum;
    AddEdgeText.Text = "Add the Edges " + EdgesLeft + " Left";
    for (int i = 0; i < NodesNum; ++i)
    {
        YourNodes.Text += " y" + (i+1);
    }
}
```

Next the program will ask for the connection of edges between the nodes and the value for each edge.

Number of Nodes:

Number of Edges:

[Start Inputs](#)

Your Nodes are: y1 y2 y3 y4 y5 y6 y7 y8 y9

Add the Edges 12 Left

From  To  Value  [Add Edge](#)

[Debug](#)

The program will print out your inputs so you don't miss anything and it will parse the input and store the data.

1 reference | Ahmed Magdy Ibrahim, 37 days ago | 1 author, 2 changes

```
private void EdgeButton_Click(object sender, RoutedEventArgs e)
{
    int From = 0;
    int To = 0;
    string Value = null;
    try
    {
        From = FromBox.Text[1] - 48;
        To = ToBox.Text[1] - 48;
        Value = ValueBox.Text;
    }
    catch (Exception) { return; }
    if (From == 0 || To == 0 || Value == null) return;
    theGraph.AddEdge(From - 1, To - 1, Value);
    EdgesLeft--;
    AddEdgeText.Text = "Add the Edges " + EdgesLeft + " Left";
    if(EdgesLeft == 0)
    {
        FromBox.IsEnabled = false;
        ToBox.IsEnabled = false;
        ValueBox.IsEnabled = false;
        EdgeButton.IsEnabled = false;
    }
    AddedEdgesText.Text += "- Edge From y" + From + " To y" + To + " with a value " + Value + "\n";
    FromBox.Text = "";
    ToBox.Text = "";
    ValueBox.Text = "";
}
```

After adding all your inputs press **Debug (Program still in developing mode)**. Then the program will start to analyze the data and print the output.

Your Nodes are: y1 y2 y3 y4 y5 y6 y7 y8 y9

Add the Edges 0 Left

From  To  Value

- Edge From y3 To y4 with a value G1
- Edge From y4 To y5 with a value 1
- Edge From y5 To y6 with a value G2
- Edge From y6 To y7 with a value G3
- Edge From y7 To y8 with a value 1
- Edge From y8 To y9 with a value 1
- Edge From y8 To y2 with a value -1
- Edge From y8 To y5 with a value -H2
- Edge From y6 To y2 with a value -H1
- Edge From y3 To y7 with a value G4

Paths:

$P1 = G1 * G2 * G3$

$P2 = G4$

Loops:

$L1 = G1 * G2 * -H1$

$L2 = G1 * G2 * G3 * -1$

$L3 = G4 * -1$

$L4 = G4 * -H2 * G2 * -H1$

$L5 = G2 * G3 * -H2$

Two Non-Touching Loops:

1 reference | Ahmed Magdy Ibrahim, 20 days ago | 1 author, 2 changes

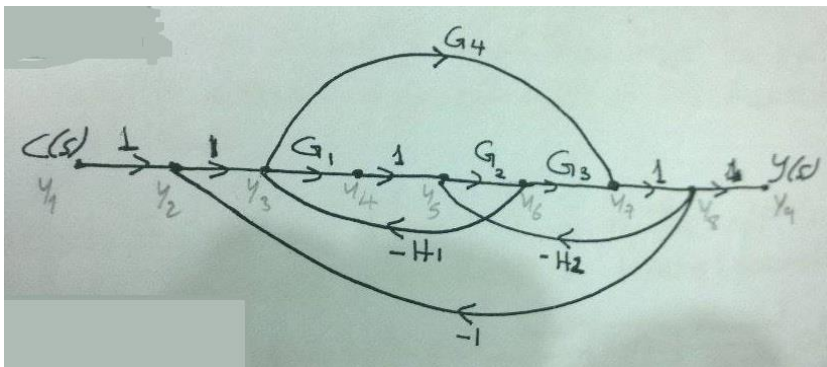
```
private void ShowResults()
{
    //-----Paths-----
    ResultsText.Text = "Paths:\n";
    foreach(Path path in theGraph.Paths)
    {
        ResultsText.Text += "P" + (theGraph.Paths.IndexOf(path)+1) + "=";
        foreach(string value in path.Values)
        {
            if(value != "1")
            {
                if (path.Values.IndexOf(value) != 0) ResultsText.Text += "*";
                ResultsText.Text += value;
            }
        }
        ResultsText.Text += "\n";
    }
    //-----Loops-----
    ResultsText.Text += "Loops:\n";
    foreach (Path loop in theGraph.Loops)
    {
        ResultsText.Text += "L" + (theGraph.Loops.IndexOf(loop) + 1) + "=";
        foreach (string value in loop.Values)
        {
            if (value != "1")
            {
                if (loop.Values.IndexOf(value) != 0) ResultsText.Text += "*";
                ResultsText.Text += value;
            }
        }
        ResultsText.Text += "\n";
    }
    //-----Non Touching-----
    ResultsText.Text += "Two Non-Touching Loops:\n";
    foreach (List<int> list in theGraph.NonTouching)
```

## The Back Tracking Algorithm:

3 references | Ahmed Magdy Ibrahim, 36 days ago | 1 author, 1 change

```
private void BackTracking(int start, int end, Track track)
{
    if(start == end && !startLoop)
    {
        if (PathGain.Count == 0) return;
        Path currentPath = new Path(new List<int>(PathNodes), new List<string>(PathGain));
        if (track == Track.Path) Paths.Add(currentPath);
        else if (track == Track.Loop) Loops.Add(currentPath);
        return;
    }
    visited[start] = true;
    if (startLoop)
    {
        visited[start] = false;
        startLoop = false;
    }
    PathNodes.Add(start);
    for (int i = 0; i < Nodes; ++i)
    {
        if (!visited[i] && theGraph[start, i] != null)
        {
            PathGain.Add(theGraph[start, i]);
            BackTracking(i, end, track);
            PathGain.Remove(theGraph[start, i]);
        }
    }
    PathNodes.Remove(start);
    visited[start] = false;
}
```

## The Input used for debugging:



**Paths:**

$$P1 = *G1*G2*G3$$

$$P2 = *G4$$

**Loops:**

$$L1 = *G1*G2*-H1$$

$$L2 = *G1*G2*G3*-1$$

$$L3 = *G4*-1$$

$$L4 = *G4*-H2*G2*-H1$$

$$L5 = G2*G3*-H2$$