**Task # 1:** Create a program to implement Graphs With Adjacency matrix

**Solution**

using System;

namespace BST

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter the number of Nodes : ");

int n = int.Parse(Console.ReadLine());

string[] arr = new string[n];

for (int i = 0; i < n; i++)

{

Console.WriteLine((i + 1) + " is connected to nodes?");

arr[i] = Console.ReadLine();}

int[,] array = new int[n, n];

for (int i = 0; i < array.GetLength(0); i++)

{

for (int j = 0; j < array.GetLength(1); j++)

{

string temp = "" + (j + 1);

if (arr[i].Contains(temp))

{

array[i, j] = 1;}

else

{

array[i, j] = 0;

}}}

Console.Write(" ");

for (int i = 0; i < n; i++)

{

Console.Write(i + 1 + " ");

}

Console.WriteLine();

for (int i = 0; i < array.GetLength(0); i++)

{

Console.Write((i + 1) + " ");

for (int j = 0; j < array.GetLength(1); j++)

{

Calendar

Description automatically generated Console.Write(array[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine("Good Bye..!");}}}

**Output**

**Task # 2:** Create a program to implement Graphs With Adjacency list

**Solution**

using System;

namespace BST

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter the number of nodes");

int n = int.Parse(Console.ReadLine());

string[] arr = new string[n];

for (int i = 0; i < n; i++)

{

Console.WriteLine((i + 1) + " is connected to nodes?");

arr[i] = Console.ReadLine();

}

List<int>[] obj = new List<int>[n];

for (int i = 0; i < n; i++)

{

obj[i] = new List<int>();

string[] temp = arr[i].Split(',');

for (int a = 0; a < temp.Length; a++)

{

obj[i].Add(int.Parse(temp[a]));

}

}

for (int i = 0; i < n; i++)

{

Console.Write((i + 1) + " => ");

for (int l = 0; l < obj[i].Count; l++)

{

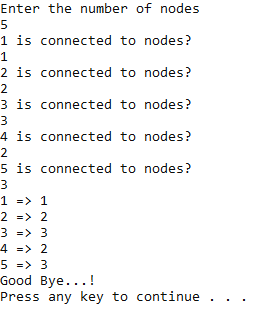
Console.Write(obj[i][l] + " ");

}

Console.WriteLine();

}

Console.WriteLine("Good Bye...!");}}}

**Output**

**Task # 3:** Create a program to implement BFS.

**Solution**

static void Main(string[] args)

{

Graph g = new Graph(6);

g.AddEdge(0, 1);

g.AddEdge(1, 2);

g.AddEdge(2, 3);

g.AddEdge(3, 4);

g.AddEdge(4, 5);

Console.WriteLine("Following is Breadth First");

g.BFS(1);

Console.WriteLine();

}

class Graph

{

private int \_V;

LinkedList<int>[] \_adj;

public Graph(int V)

{

\_adj = new LinkedList<int>[V];

for (int i = 0; i < \_adj.Length; i++)

{

\_adj[i] = new LinkedList<int>();

}

\_V = V;

}

public void AddEdge(int v, int w)

{

\_adj[v].AddLast(w);

}

public void BFS(int s)

{

bool[] visited = new bool[\_V];

for (int i = 0; i < \_V; i++)

visited[i] = false;

LinkedList<int> queue = new LinkedList<int>();

visited[s] = true;

queue.AddLast(s);

while (queue.Any())

{

s = queue.First();

Console.Write(s + " ");

queue.RemoveFirst();

LinkedList<int> list = \_adj[s];

foreach (var val in list)

{

if (!visited[val])

{

visited[val] = true;

queue.AddLast(val);

}

}}}}

**Output**

**Text

Description automatically generated with low confidence**

**Task # 4:** Create a program to implement DFS.

**Solution**

static void Main(string[] args)

{

Graph graph = new Graph(11);

graph.AddEdge(1, 2, false);

graph.AddEdge(2, 3, false);

graph.AddEdge(3, 4, false);

graph.AddEdge(1, 5, false);

graph.AddEdge(5, 6, false);

graph.AddEdge(6, 7, false);

graph.AddEdge(5, 8, false);

graph.AddEdge(1, 9, false);

graph.AddEdge(9, 10, false);

graph.DFS();

Console.WriteLine();

}

public class Graph

{

LinkedList<int>[] linkedListArray;

public Graph(int v)

{

linkedListArray = new LinkedList<int>[v];

}

public void AddEdge(int u, int v, bool blnBiDir = true)

{

if (linkedListArray[u] == null)

{

linkedListArray[u] = new LinkedList<int>();

linkedListArray[u].AddFirst(v);

}

else

{

var last = linkedListArray[u].Last;

linkedListArray[u].AddAfter(last, v);

}

if (blnBiDir)

{

if (linkedListArray[v] == null)

{

linkedListArray[v] = new LinkedList<int>();

linkedListArray[v].AddFirst(u);

}

else

{

var last = linkedListArray[v].Last;

linkedListArray[v].AddAfter(last, u);

}

}

}

internal void DFSHelper(int src, bool[] visited)

{

visited[src] = true;

Console.Write(src + " ");

if (linkedListArray[src] != null)

{

foreach (var item in linkedListArray[src])

{

if (!visited[item] == true)

{

DFSHelper(item, visited);

}

}

}

}

internal void DFS()

{

Console.WriteLine("\t\tDFS");

bool[] visited = new bool[linkedListArray.Length + 1];

DFSHelper(1, visited);

}}

**Output**

**A picture containing text

Description automatically generated**