1)DFS(for disconnected also)

#include <stdio.h>

#define MAX 100

int matrix[MAX][MAX], visited[MAX];

//STACK

int top = -1;

int stack[MAX];

void push(int item)

{

    stack[++top] = item;

}

int pop()

{

    return stack[top--];

}

int peek()

{

    return stack[top];

}

int isStackEmpty()

{

    return top == -1;

}

void init(int vertices)

{

    int i, j;

    for (i = 0; i < vertices; i++)

    {

        for (j = 0; j < vertices; j++)

        {

            matrix[i][j] = 0;

        }

    }

}

void addEdge(int vertices)

{

    int i, j, v;

    for (i = 0; i < vertices; i++)

    {

        visited[i] = -1;

        for (j = 0; j < vertices - 1; j++)

        {

            printf("Enter the adjacent node of %d(-1 to exit): ", i);

            scanf("%d", &v);

            if (v == -1)

            {

                break;

            }

            matrix[i][v] = 1;

        }

    }

    return;

}

int getAdjNode(int vertex, int vertices)

{

    int i;

    for (i = 0; i < vertices; i++)

    {

        if (matrix[vertex][i] == 1 && visited[i] == -1)

        {

            return i;

        }

    }

    return -1;

}

void DFS(int vertices,int vertex)

{

    int  i;

    visited[vertex] = 1;

    printf("%d\t", vertex);

    push(vertex);

    while (!isStackEmpty())

    {

        int adjunviNode = getAdjNode(peek(), vertices);

        if (adjunviNode == -1)

        {

            pop();

        }

        else

        {

            visited[adjunviNode] = 1;

            printf("%d\t", adjunviNode);

            push(adjunviNode);

        }

    }

}

void DFSdis(int v){

    for(int i=0;i<v;i++){

        if(visited[i]==-1){

            DFS(v,i);

            printf("\n");

        }

    }

}

void printMatrix(int vertices)

{

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

    {

        for (int j = 0; j < vertices; j++)

        {

            printf("%d\t", matrix[i][j]);

        }

        printf("\n");

    }

}

int main()

{

    int vertices;

    printf("Enter the number of vertices: ");

    scanf("%d", &vertices);

    printf("\n");

    init(vertices);

    addEdge(vertices);

    printf("The adjacency matrix is:\n");

    printMatrix(vertices);

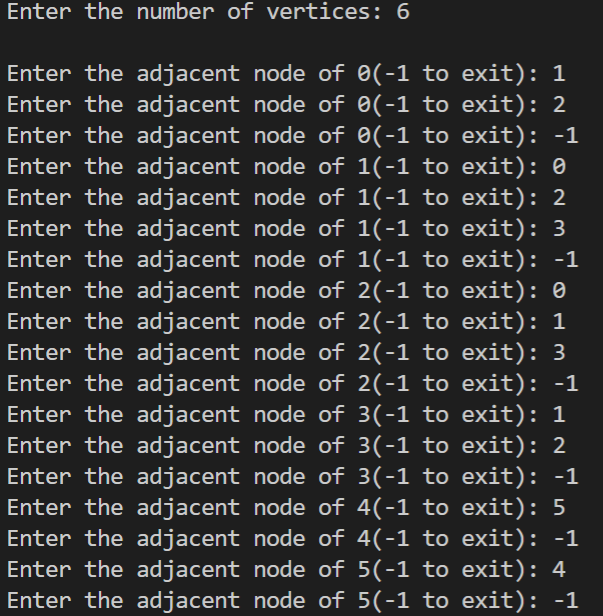
    printf("\n");

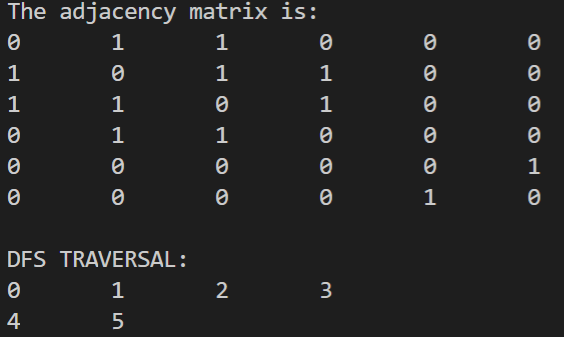
    printf("DFS TRAVERSAL:\n");

    DFSdis(vertices);

}

OUTPUT:





ANALYSIS:

Let V be the number of vertices and E be the number of edges.

We are using adjacency matrix here.During dfs traversal,each vertex v is called atleast once.From each vertex, all the vertices are checked until a negihbouring vertex is found.The complexity of this proces is O(V).Since this process is done for all the vertices,The Overall complexity is O(VxV)=O(V2).

2)BFS(including disconnected graph)

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

typedef struct Node

{

    int vertex;

    struct Node \*next;

} node;

typedef struct Graph

{

    int visited;

    struct Node \*adjList;

} graph;

//QUEUE

int front = -1;

int rear = -1;

int queue[MAX];

void enqueue(int v)

{

    if (front == -1 && rear == -1)

    {

        queue[++front] = queue[++rear] = v;

        return;

    }

    queue[++rear] = v;

    return;

}

int dequeue()

{

    return queue[front++];

}

int getFront()

{

    return queue[front];

}

int isQueueEmpty()

{

    if (front > rear)

    {

        return 1;

    }

    return 0;

}

node \*createNode(int v)

{

    node \*newnode = (node \*)malloc(sizeof(node));

    newnode->vertex = v;

    newnode->next = NULL;

    return newnode;

}

void createGraph(graph \*p, int v)

{

    int i;

    for (i = 0; i < v; i++)

    {

        p[i].adjList = (node \*)malloc(sizeof(node));

        p[i].visited = -1;

        p[i].adjList = NULL;

    }

    return;

}

void addEdge(graph \*graph, int src, int dest)

{

    node \*newNode = createNode(dest);

    newNode->next = graph[src].adjList;

    graph[src].adjList = newNode;

    return;

}

int getAdjunvisited(graph \*graph, int vertex)

{

    node \*temp = graph[vertex].adjList;

    while (temp)

    {

        if (graph[temp->vertex].visited == -1)

        {

            return temp->vertex;

        }

        temp = temp->next;

    }

    return -1;

}

void BFS(graph \*graph, int v,int vertex)

{

    int i;

    graph[vertex].visited = 1;

    enqueue(vertex);

    while (!isQueueEmpty())

    {

        node \*temp = graph[vertex].adjList;

        while (temp)

        {

            if (graph[temp->vertex].visited == -1)

            {

                enqueue(temp->vertex);

                graph[temp->vertex].visited = 1;

            }

            temp = temp->next;

        }

        printf("%d\t", getFront());

        dequeue();

        vertex = getFront();

    }

    printf("\n");

}

void BFSdis(graph \*g,int v){

    for(int i=0;i<v;i++){

        if(g[i].visited==-1)

        BFS(g,v,i);

    }

}

void printAdjNodes(graph \*g, int v)

{

    for (int i = 0; i < v; i++)

    {

        node \*temp = g[i].adjList;

        printf("%d->", i);

        while (temp != NULL)

        {

            printf("%d,", temp->vertex);

            temp = temp->next;

        }

        printf("\n");

    }

}

int main()

{

    int vertices, i, adj, j;

    printf("\nEnter the number of vertices: ");

    scanf("%d", &vertices);

    printf("\n");

    graph g[vertices];

    createGraph(g, vertices);

    for (i = 0; i < vertices; i++)

    {

        for (j = 0; j < vertices - 1; j++)

        {

            printf("Enter the adjacent node of %d(-1 to exit): ", i);

            scanf("%d", &adj);

            if (adj == -1)

            {

                break;

            }

            addEdge(g, i, adj);

        }

    }

    printAdjNodes(g, vertices);

    printf("\n");

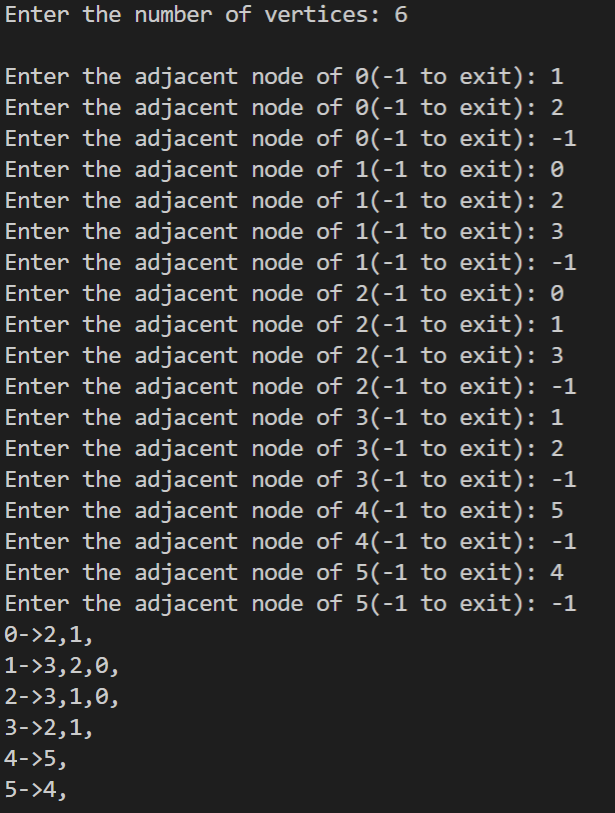
    printf("BFS TRAVERSAL:\n");

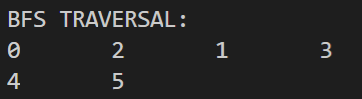
    BFSdis(g, vertices);

    return 0;

}

OUTPUT:





ANALYSIS:

Let V be the number of vertices and E be the number of edges.

We are using adjacency list here.During bfs traversal,each vertex v is called atleast once.From,each vertex all of its neighbouring vertices are checked,it’s complexity (summing over all vertices) is O(E).Constant operation of changing the visited flag to 1 happens for every vertex.It’s complexity is O(V).The overall complexity is O(V)+O(E)=O(V+E).

3)Threaded Binary Tree