NAME- Aniket Kalbhor

PRN- 12210601

ROLL- CSB-48

CODE- 7**.a) Write a program in C to traverse DFS and BFS on graph using Adjcency matrix.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 50

//bfs code

struct Queue

{

int data;

struct Queue\* next;

};

struct Queue\* front;

struct Queue\* rear;

int isEmpty()

{

return (front == NULL && rear == NULL);

}

void enQueue(int key)

{

struct Queue\* p = (struct Queue\*)malloc(sizeof(struct Queue));

if (p == NULL)

printf("Memory allocation failed\n");

p->data = key;

p->next = NULL;

if (isEmpty()){

front = p;

rear = p;

}

else{

rear->next = p;

rear = p;

}

}

int deQueue()

{

if (isEmpty())

printf("Queue is empty\n");

struct Queue\* temp = front;

int val = temp->data;

if (front == rear){

front = NULL;

rear = NULL;

}

else

front = front->next;

free(temp);

return val;

}

void BFS(int G[10][10],int vertex)

{

int visited[vertex], i, u, start;

for (i = 0;i < vertex;i++)

visited[i] = 0;

printf("Enter starting vertex: ");

scanf("%d",&start);

visited[start] = 1;

enQueue(start);

while(!isEmpty())

{

u = deQueue();

printf("%d ",u);

for (i = 0;i < vertex;i++)

{

if (G[u][i] == 1 && visited[i] == 0)

{

enQueue(i);

visited[i] = 1;

}

}

}

}

//dfs code

int stack[MAX];

int top = -1;

int emptyStack()

{

if (top == -1)

return 1;

else

return 0;

}

void push(int n)

{

if (top >= MAX-1)

printf("Stack overflow\n");

stack[++top] = n;

}

int pop()

{

if (emptyStack())

{

printf("Stack underflow\n");

return -1;

}

return stack[top--];

}

void DFS(int G[10][10], int vertex)

{

int visited[vertex], i, u, st;

for (i = 0;i < vertex;i++)

visited[i] = 0;

printf("Enter start vertex\n");

scanf("%d",&st);

push(st);

printf("%d ",st);

visited[st] = 1;

u = st;

while (!emptyStack())

{

for (i = 0;i < vertex;i++)

if (G[u][i] == 1 && visited[i] != 1)

{

push(i);

visited[i] = 1;

printf("%d ",i);

u = i;

break;

}

if (i == vertex)

{

pop();

u = stack[top];

}

}

}

void display(int G[10][10], int vertex)

{

int i, j;

for (i = 0;i < vertex;i++)

{

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

printf("%d ",G[i][j]);

printf("\n");

}

}

int main()

{

int G[10][10], vertex, e, i, j, u, v;

printf("Enter number of vertices: ");

scanf("%d",&vertex);

for (i = 0;i < vertex;i++)

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

G[i][j] = 0;

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

scanf("%d",&e);

for (i = 0;i < e;i++)

{

printf("Enter edge: ");

scanf("%d",&u);

scanf("%d",&v);

G[u][v] = G[v][u] = 1;

}

display(G, vertex);

int choice;

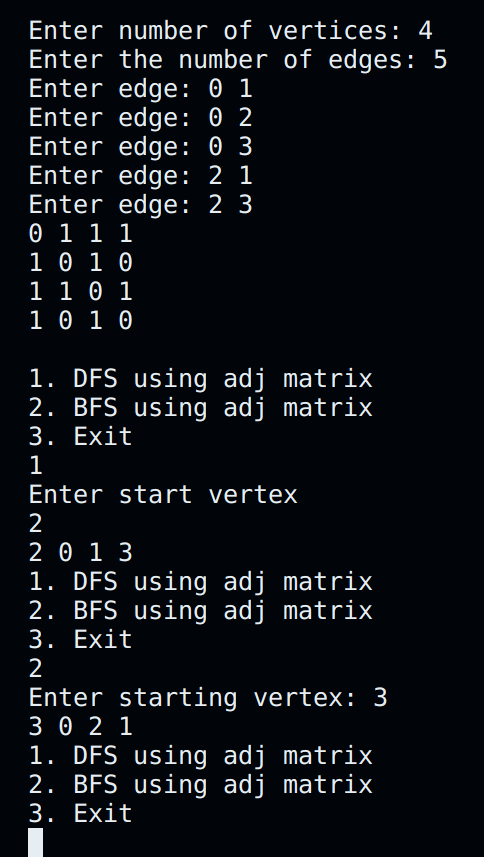
while (1)

{

printf("\n1. DFS using adj matrix\n2. BFS using adj matrix\n3. Exit\n");

scanf("%d",&choice);

switch (choice)

{

case 1:

DFS(G, vertex);

break;

case 2:

BFS(G, vertex);

break;

case 3:

exit(0);

}

}

}

**b) Write a program in C to traverse DFS and BFS on graph using Adjcency List.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 15

struct Node

{

int vertex;

struct Node\* next;

};

struct Node\* front;

struct Node\* rear;

struct Node\* a[MAX];

int emptyQueue()

{

return (front == NULL && rear == NULL);

}

void enqueue(int key)

{

struct Node\* p = (struct Node\*)malloc(sizeof(struct Node));

if (p == NULL)

{

printf("Memory allocation failed\n");

return;

}

p->vertex = key;

p->next = NULL;

if (emptyQueue())

{

front = p;

rear = p;

}

else

{

rear->next = p;

rear = p;

}

}

int dequeue()

{

struct Node\* temp = front;

if (emptyQueue())

{

printf("Queue is empty\n");

return -1;

}

int value = temp->vertex;

if (front == rear)

{

front = NULL;

rear = NULL;

}

else

front = front->next;

free(temp);

return value;

}

void BFS(struct Node\* a[MAX], int v)

{

int visited[MAX], st, u, i;

for (i = 0; i < v; i++)

visited[i] = 0;

printf("Enter start vertex\n");

scanf("%d", &st);

visited[st] = 1;

enqueue(st);

while (!emptyQueue())

{

u = dequeue();

printf("%d ", u);

struct Node\* p = a[u];

while (p != NULL)

{

if (visited[p->vertex] == 0)

{

enqueue(p->vertex);

visited[p->vertex] = 1;

}

p = p->next;

}

}

}

//----------------------------------DFS----------------------------------

int stack[MAX];

int top = -1;

int emptyStack()

{

if (top == -1)

return 1;

else

return 0;

}

void push(int key)

{

if (top >= MAX - 1)

printf("Stack overflow\n");

stack[++top] = key;

}

int pop()

{

if (emptyStack())

{

printf("Stack underflow\n");

return -1;

}

return stack[top--];

}

void DFS(struct Node\* a[MAX], int v)

{

int visited[MAX], i, u, st;

struct Node\* p;

for (i = 0; i < v; i++)

visited[i] = 0;

printf("Enter start vertex\n");

scanf("%d", &st);

push(st);

printf("%d ", st);

visited[st] = 1;

u = st;

p = a[u];

do

{

p = a[u];

while (p != NULL)

{

if (visited[p->vertex] == 0)

{

push(p->vertex);

visited[p->vertex] = 1;

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

u = p->vertex;

break;

}

else

p = p->next;

}

if (p == NULL)

{

pop();

u = stack[top];

}

} while (!emptyStack());

}

void display\_list(struct Node\* a[MAX], int v)

{

int i;

struct Node\* p;

printf("\nAdjacency list:\n");

for (i = 0; i < v; i++)

{

p = a[i];

printf("%d", i);

while (p != NULL)

{

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

p = p->next;

}

printf("\n");

}

}

int main()

{

struct Node\* p;

struct Node\* newV;

int u, v, vertex,e, i;

char ch;

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

scanf("%d", &vertex);

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

a[i] = NULL;

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

scanf("%d",&e);

for (i = 0;i < e;i++)

{

printf("Enter edge: ");

scanf("%d",&u);

scanf("%d",&v);

newV = (struct Node\*)malloc(sizeof(struct Node));

newV->vertex = v;

newV->next = NULL;

p = a[u];

if (p == NULL)

a[u] = newV;

else

{

while (p->next != NULL)

p = p->next;

p->next = newV;

}

newV = (struct Node\*)malloc(sizeof(struct Node));

newV->vertex = u;

newV->next = NULL;

p = a[v];

if (p == NULL)

a[v] = newV;

else

{

while (p->next != NULL)

p = p->next;

p->next = newV;

}

}

int choice;

while (1)

{

display\_list(a, vertex);

printf("\n1. DFS using adj list\n2. BFS using adj list\n3. Exit\n");

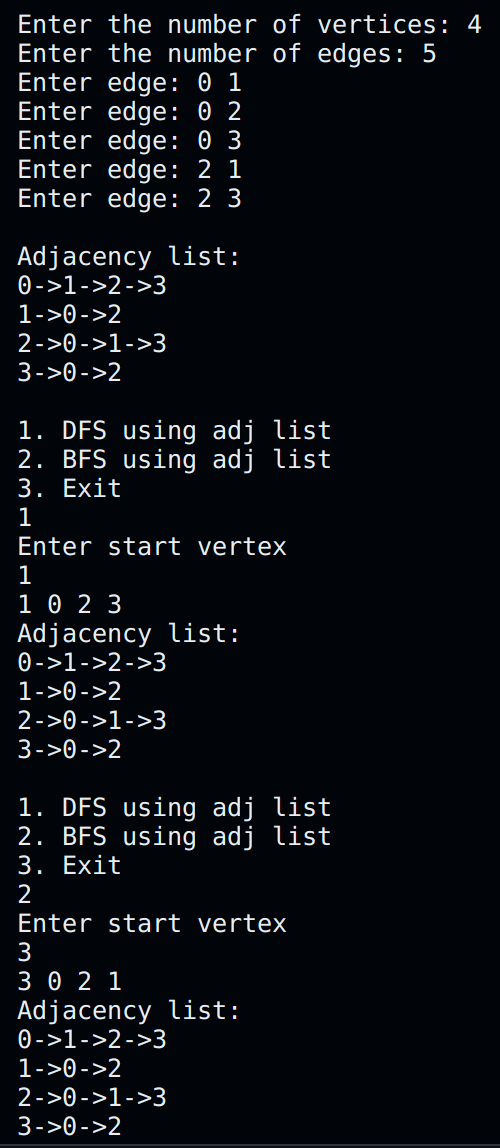
scanf("%d",&choice);

switch (choice)

{

case 1:

DFS(a, vertex);

break;

case 2:

BFS(a, vertex);

break;

case 3:

exit(0);

}

}

}