**CODE:-**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int time = 0;

int d[MAX], f[MAX];

int state[MAX];

int queue[MAX], front = -1, rear = -1;

struct Edge;

struct Vertex

{

int info;

struct Vertex \*nextVertex;

struct Edge \*firstEdge;

} \*start = NULL;

struct Edge

{

struct Vertex \*destVertex;

struct Edge \*nextEdge;

};

struct Vertex \*findVertex(int u);

void insertVertex(int u);

void insertEdge(int u, int v);

void deleteEdge(int u, int v);

void deleteIncomingEdges(int u);

void deleteVertex(int u);

void display();

void BFS\_Traversal();

void BFS(int v);

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

void DFS\_Traversal();

void DFS(int v);

int main()

{

int choice, u, origin, destin;

while (1)

{

printf("1. Insert an vertex.\n");

printf("2. Insert an Edge.\n");

printf("3. Delete an vertex.\n");

printf("4. Delete an Edge.\n");

printf("5. Display.\n");

printf("6. BFS Traversal.\n");

printf("7. DFS Traversal.\n");

printf("8. Exit.\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter a vertex to be inserted - ");

scanf("%d", &u);

insertVertex(u);

break;

case 2:

printf("Enter a edge to be inserted - ");

scanf("%d %d", &origin, &destin);

insertEdge(origin, destin);

break;

case 3:

printf("Enter the vertex to be deleted - ");

scanf("%d", &u);

deleteIncomingEdges(u);

deleteVertex(u);

break;

case 4:

printf("Enter a edge to be deleted - ");

scanf("%d %d", &origin, &destin);

deleteEdge(origin, destin);

break;

case 5:

display();

break;

case 6:

BFS\_Traversal();

break;

case 7:

DFS\_Traversal();

break;

case 8:

exit(1);

default:

printf("Entered wrong choice. Please re-enter choice");

}

}

return 0;

}

void insertVertex(int u)

{

struct Vertex \*tmp, \*ptr;

tmp = malloc(sizeof(struct Vertex));

tmp->info = u;

tmp->nextVertex = NULL;

tmp->firstEdge = NULL;

if (start == NULL)

{

start = tmp;

return;

}

ptr = start;

while (ptr->nextVertex != NULL)

ptr = ptr->nextVertex;

ptr->nextVertex = tmp;

}

void deleteVertex(int u)

{

struct Vertex \*tmp, \*q;

struct Edge \*p, \*temporary;

if (start == NULL)

{

printf("No vertices to be deleted\n");

return;

}

if (start->info == u)

{

tmp = start;

start = start->nextVertex;

}

else

{

q = start;

while (q->nextVertex != NULL)

{

if (q->nextVertex->info == u)

break;

q = q->nextVertex;

}

if (q->nextVertex == NULL)

{

printf("Vertex not found.\n");

return;

}

else

{

tmp = q->nextVertex;

q->nextVertex = tmp->nextVertex;

}

}

p = tmp->firstEdge;

while (p != NULL)

{

temporary = p;

p = p->nextEdge;

free(temporary);

}

free(tmp);

}

void deleteIncomingEdges(int u)

{

struct Vertex \*ptr;

struct Edge \*q, \*tmp;

ptr = start;

while (ptr != NULL)

{

if (ptr->firstEdge == NULL)

{

ptr = ptr->nextVertex;

continue;

}

if (ptr->firstEdge->destVertex->info == u)

{

tmp = ptr->firstEdge;

ptr->firstEdge = ptr->firstEdge->nextEdge;

free(tmp);

continue;

}

q = ptr->firstEdge;

while (q->nextEdge != NULL)

{

if (q->nextEdge->destVertex->info == u)

{

tmp = q->nextEdge;

q->nextEdge = tmp->nextEdge;

free(tmp);

continue;

}

q = q->nextEdge;

}

ptr = ptr->nextVertex;

}

}

struct Vertex \*findVertex(int u)

{

struct Vertex \*ptr, \*loc;

ptr = start;

while (ptr != NULL)

{

if (ptr->info == u)

{

loc = ptr;

return loc;

}

else

{

ptr = ptr->nextVertex;

}

}

loc = NULL;

return loc;

}

void insertEdge(int u, int v)

{

struct Vertex \*locu, \*locv;

struct Edge \*ptr, \*tmp;

locu = findVertex(u);

locv = findVertex(v);

if (locu == NULL)

{

printf("Start vertex not present, first insert vertex %d\n", u);

return;

}

if (locv == NULL)

{

printf("End vertex not present, first insert vertex %d\n", v);

return;

}

tmp = malloc(sizeof(struct Edge));

tmp->destVertex = locv;

tmp->nextEdge = NULL;

if (locu->firstEdge == NULL)

{

locu->firstEdge = tmp;

return;

}

ptr = locu->firstEdge;

while (ptr->nextEdge != NULL)

ptr = ptr->nextEdge;

ptr->nextEdge = tmp;

}

void deleteEdge(int u, int v)

{

struct Vertex \*locu;

struct Edge \*tmp, \*q;

locu = findVertex(u);

if (locu == NULL)

{

printf("Start Vertex not present\n");

return;

}

if (locu->firstEdge == NULL)

{

printf("Edge not present\n");

return;

}

if (locu->firstEdge->destVertex->info == v)

{

tmp = locu->firstEdge;

locu->firstEdge = locu->firstEdge->nextEdge;

free(tmp);

return;

}

q = locu->firstEdge;

while (q->nextEdge != NULL)

{

if (q->nextEdge->destVertex->info == v)

{

tmp = q->nextEdge;

q->nextEdge = tmp->nextEdge;

free(tmp);

return;

}

q = q->nextEdge;

}

printf("This Edge not present in the graph\n");

}

void display()

{

struct Vertex \*ptr;

struct Edge \*q;

ptr = start;

while (ptr != NULL)

{

printf("%d -> ", ptr->info);

q = ptr->firstEdge;

while (q != NULL)

{

printf("%d -> ", q->destVertex->info);

q = q->nextEdge;

}

printf("\n");

ptr = ptr->nextVertex;

}

}

void BFS\_Traversal()

{

int v;

struct Vertex \*ptr;

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

state[v] = initial;

printf("Enter starting vertex for BFS - ");

scanf("%d", &v);

BFS(v);

ptr = start;

while (ptr != NULL)

{

if (state[ptr->info] == initial)

BFS(ptr->info);

ptr = ptr->nextVertex;

}

printf("\n");

}

void BFS(int v)

{

struct Vertex \*locw;

struct Edge \*ptr;

int w;

insert\_queue(v);

state[v] = waiting;

while (!isEmpty\_queue())

{

w = delete\_queue();

printf("%d ", w);

state[w] = visited;

locw = findVertex(w);

ptr = locw->firstEdge;

while (ptr != NULL)

{

if (state[ptr->destVertex->info] == initial)

{

insert\_queue(ptr->destVertex->info);

state[ptr->destVertex->info] = waiting;

}

ptr = ptr->nextEdge;

}

}

}

void insert\_queue(int vertex)

{

if (rear == MAX - 1)

{

printf("Queue overflown\n");

return;

}

else

{

if (front == -1)

front = 0;

rear = rear + 1;

queue[rear] = vertex;

}

}

int isEmpty\_queue()

{

if (front == -1 || front > rear)

return 1;

else

return 0;

}

int delete\_queue()

{

int del\_item;

if (isEmpty\_queue())

{

printf("Queue Underflown\n");

exit(1);

}

del\_item = queue[front];

front = front + 1;

return del\_item;

}

void DFS\_Traversal()

{

int v;

struct Vertex \*ptr;

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

state[v] = initial;

printf("Enter starting vertex for DFS - ");

scanf("%d", &v);

DFS(v);

ptr = start;

while (ptr != NULL)

{

if (state[ptr->info] == initial)

DFS(ptr->info);

ptr = ptr->nextVertex;

}

printf("\n");

}

void DFS(int v)

{

struct Vertex \*locv;

struct Edge \*ptr;

d[v] = ++time;

printf("%d ", v);

state[v] = visited;

locv = findVertex(v);

ptr = locv->firstEdge;

while (ptr != NULL)

{

if (state[ptr->destVertex->info] == initial)

DFS(ptr->destVertex->info);

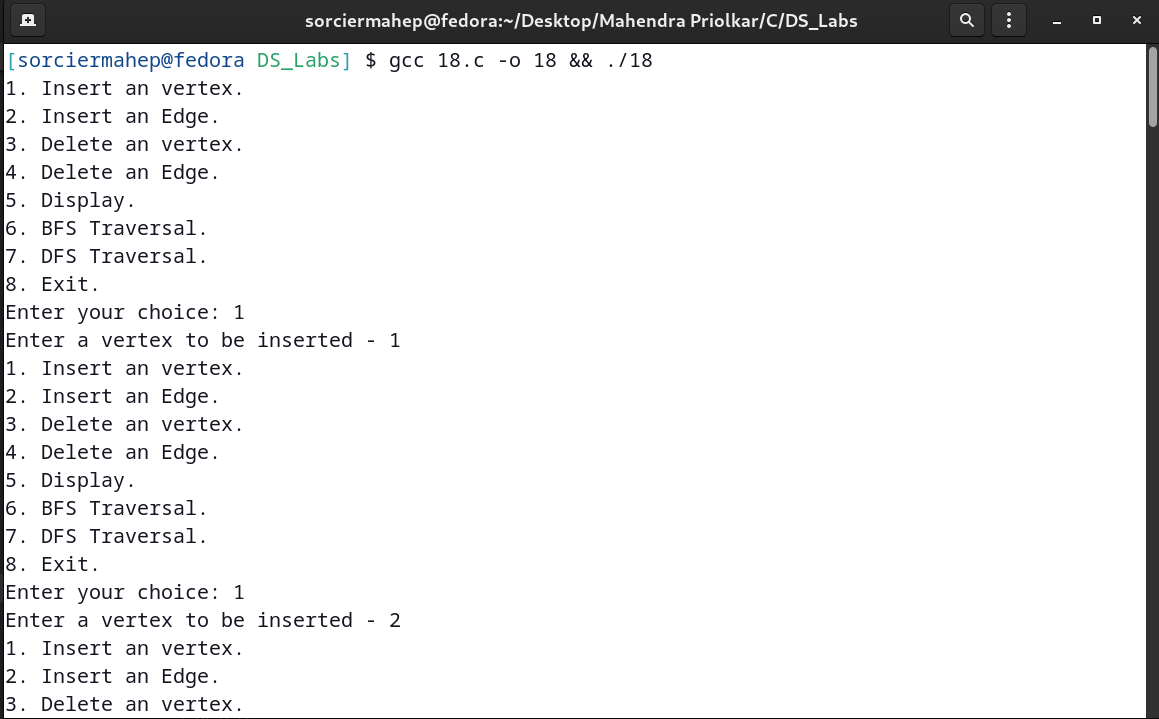
ptr = ptr->nextEdge;

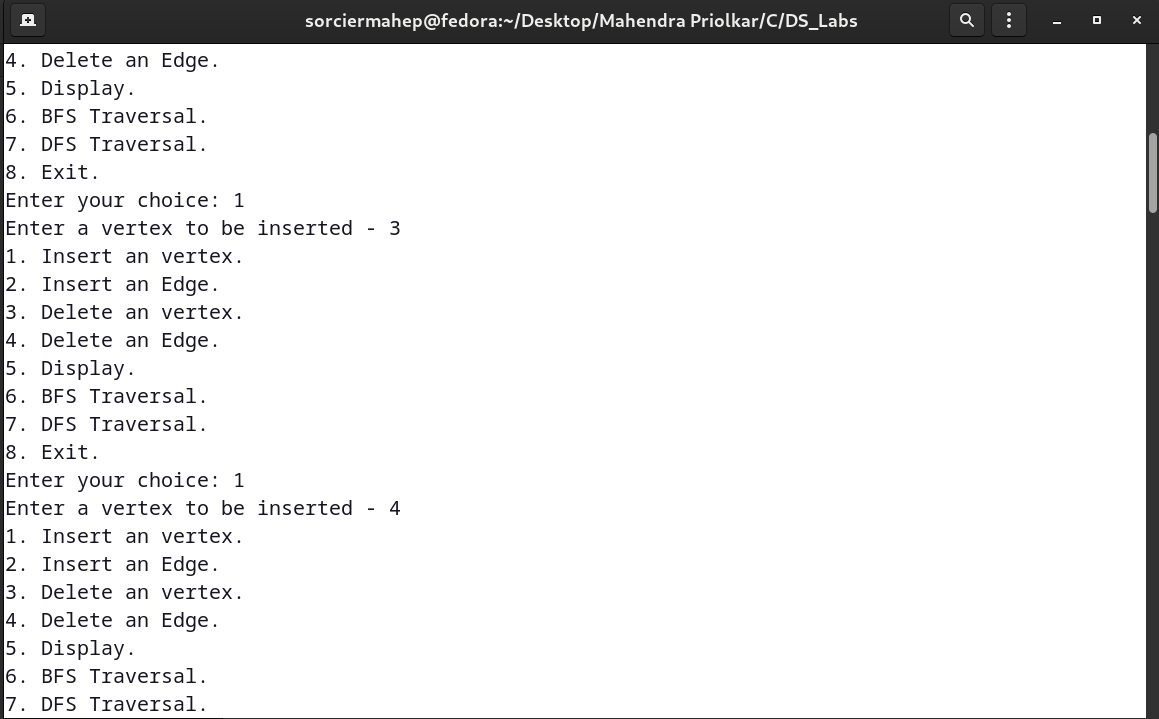
}

f[v] = ++time;

}

**OUTPUT:-**



****

