**Exercise 10:** **Graph traversal and its applications**

**ADT.h:**

#include<stdio.h>

#include "stackimpl.h"

#include "qimpl.h"

struct graph

{

int adj[10][10];

int vtx;

int edge;

int vis[100];

struct queue Q;

};

void create(struct graph \*G,int v,int e,int d);

void dispMat(struct graph \*G);

//void BFS(struct graph \*G,int);

void BFS(struct graph \*G);

void DFS(struct graph \*G;

void BFSprint(struct graph \*G,int V);//,struct queue Q,int vis[]);

void DFSprint(struct graph \*G,int);//,int[]);

**impl.h:**

#include "adt.h"

void vQ(struct graph \*G)

{

for(int i=0;i<100;i++)

{

G->vis[i]=0;

}

while(!isE(&G->Q))

{

deQ(&G->Q);

}

}

void create(struct graph \*G,int v,int e,int d)

{

G->vtx=v;

G->edge=e;

init(&G->Q,20);

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

{

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

{

G->adj[i][j]=0;

}

}

vQ(G);

char edge[2];

//if(d)

{

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

{

printf("Enter the edge: ");

scanf("%s",edge);

G->adj[edge[0]-'A'][edge[1]-'A']=1;

if(!d)

G->adj[edge[1]-'A'][edge[0]-'A']=1;

}

/\*G->adj[0][1]=1;

G->adj[1][2]=1;

G->adj[1][3]=1;

G->adj[2][0]=1;

G->adj[2][5]=1;

G->adj[3][4]=1;

G->adj[4][5]=1;\*/

/\*G->adj[0][4]=1;

G->adj[1][2]=1;

G->adj[1][3]=1;

G->adj[1][4]=1;

G->adj[2][3]=1;

G->adj[3][4]=1;\*/

/\*G->adj[1][3]=1;

G->adj[2][3]=1;

G->adj[0][4]=1;

G->adj[4][5]=1;

G->adj[5][6]=1;\*/

return;

}

//return;

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

{

printf("Enter the edge: ");

scanf("%s",edge);

G->adj[edge[0]-'A'][edge[1]-'A']=1;

G->adj[edge[1]-'A'][edge[0]-'A']=1;

}

}

void dispMat(struct graph \*G)

{

for(int i=0;i<G->vtx;i++)

{

for(int j=0;j<G->vtx;j++)

{

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

}

printf("\n");

}

}

void BFS(struct graph \*G)

{

for(int i=0;i<G->vtx;i++)

{

if(G->vis[i]==0)

BFSprint(G,i+65);

}

}

void BFS1(struct graph \*G)

{

for(int i=0;i<G->vtx;i++)

{

if(G->vis[i]==0)

{

BFSprint(G,i+65);

printf("\n");

}

}

}

void BFSprint(struct graph \*G,int V)

{

V-=65;

G->vis[V]=1;

enQ(&G->Q,V);

while(!isE(&G->Q))

{

int cur\_vtx=deQ(&G->Q);

printf("%c", cur\_vtx+65);

for(int i=0;i<G->vtx;i++)

{

if(G->adj[cur\_vtx][i])

{

int adj\_vtx=i;

if(G->vis[adj\_vtx]==0)

{

G->vis[adj\_vtx]=1;

enQ(&G->Q,adj\_vtx);

}

}

}

}

}

void DFS(struct graph \*G)

{

for(int i=0;i<G->vtx;i++)

{

if(G->vis[i]==0)

DFSprint(G,i+65);

}

}

void DFSprint(struct graph \*G,int V) //recursive

{

V-=65;

G->vis[V]=1;

printf("%c",V+65);

for(int i=0;i<G->vtx;i++)

{

if(G->adj[V][i])

{

int adj\_vtx=i;

if(G->vis[adj\_vtx]==0)

{

DFS(G,adj\_vtx+65);

}

}

}

}

/\*void DFSprint(struct graph \*g,int V) //using actual stack

{

int i=0;

g->vis[V]=1;

push(&g->s,V);

printf("%d", V);

while(!sIsEmpty(&g->s))

{

int cur\_vtx=peek(&g->s);

for(i=0;i<g->v;i++)

{

if(g->adj[cur\_vtx][i])

{

if(g->vis[i]==0)

{

push(&g->s,i);

g->vis[i]=1;

printf("%d", i);

break;

}

}

}

if(i==g->v)

pop(&g->s);

}

}\*/

**appl.c:**

#include<stdlib.h>

#include "impl.h"

int BFS2(struct graph \*G,int V,int S);

int main()

{

struct graph \*G=(struct graph\*)malloc(sizeof(struct graph));

int v,e;

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

scanf("%d%d",&v,&e);

create(G,v,e,1);

dispMat(G);

printf("\nBFS:\n");

BFS(G);

printf("\n");

vQ(G);

printf("\nDFS:\n");

DFS(G);

printf("\n");

vQ(G);

printf("\nD to F: ");

int ans=BFS2(G,'D','F');

if(ans)

printf("Yes!\n");

else

printf("No!\n");

vQ(G);

printf("\nF to B: ");

ans=BFS2(G,'F','B');

if(ans)

printf("Yes!\n");

else

printf("No!\n");

printf("\n");

struct graph \*G1=(struct graph\*)malloc(sizeof(struct graph));

//int v1,e1;

int v1=5,e1=3;//ENTER AB,BC,DE

/\*printf("Enter the number of verties and edges: ");

scanf("%d%d",&v1,&e1);\*/

create(G1,v1,e1,0);

dispMat(G1);

printf("\nConnected Comps:\n");

BFS1(G1);

printf("\n");

return 0;

}

int BFS2(struct graph \*G,int V,int S)

{

V-=65;

G->vis[V]=1;

enQ(&G->Q,V);

while(!isE(&G->Q))

{

int cur\_vtx=deQ(&G->Q);

for(int i=0;i<G->vtx;i++)

{

if(G->adj[cur\_vtx][i])

{

int adj\_vtx=i;

if(adj\_vtx+65==S)

return 1;

if(G->vis[adj\_vtx]==0)

{

G->vis[adj\_vtx]=1;

enQ(&G->Q,adj\_vtx);

}

//i++;

}

}

}

return 0;

}

**queueADT:**

struct queue

{

int max;

int f,r;

int data[100];

};

void init(struct queue \*,int);

int isF(struct queue \*);

int isE(struct queue \*);

void enQ(struct queue \*,int);

int deQ(struct queue \*);

void disp(struct queue);

**qimpl.h:**

#include<stdio.h>

#include "qADT.h"

void init(struct queue \*Q,int val)

{

Q->max=val;

Q->f=-1;

Q->r=-1;

}

int isF(struct queue \*Q)

{

if((Q->r+1)%Q->max==Q->f)

return 1;

return 0;

}

int isE(struct queue \*Q)

{

if(Q->f==-1 && Q->r==-1)

return 1;

return 0;

}

void enQ(struct queue \*Q,int val)

{

if(isF(Q))

{

printf("\nOverflow!\n");

return;

}

else if(isE(Q))

{

Q->f=0;

Q->r=0;

}

else

Q->r=(Q->r+1)%Q->max;

Q->data[Q->r]=val;

}

int deQ(struct queue \*Q)

{

int item=Q->data[Q->f];

if(isE(Q))

{

printf("\nUnderflow!\n");

return -1;

}

else if(Q->f==Q->r)

{

Q->f=-1;

Q->r=-1;

}

else

Q->f=(Q->f+1)%Q->max;

return item;

}

void disp(struct queue Q)

{

if(isE(&Q))

{

printf("\nUnderflow!\n");

return;

}

int i;

printf("Queue:");

for(i=Q.f;i!=Q.r; i=(i+1)%Q.max)

{

printf("\n%d",Q.data[i]);

}

printf("\n%d\n",Q.data[i]);

}

**stackADT.h:**

#include<stdio.h>

#include<stdlib.h>

struct stack

{

int data[10];

int top,size;

};

void sInit(struct stack \*s,int size);

int sIsFull(struct stack \*s);

int sIsEmpty(struct stack \*s);

void push(struct stack \*s,int x);

int pop(struct stack \*s);

int peek(struct stack \*s);

**stackimpl.h:**

#include "stackADT.h"

void sInit(struct stack \*s,int size)

{

s->top=-1;

s->size=size;

}

int sIsFull(struct stack \*s)

{

return s->top==s->size-1;

}

int sIsEmpty(struct stack \*s)

{

return s->top==-1;

}

void push(struct stack \*s,int x)

{

if(sIsFull(s))

return;

s->data[++s->top]=x;

}

int pop(struct stack \*s)

{

if(sIsEmpty(s))

return -1;

return s->data[s->top--];

}

int peek(struct stack \*s)

{

if(sIsEmpty(s))

return -1;

return s->data[s->top];

}

**Sample I/O:**



