实验五 图的遍历

#include <iostream>

#define MAX\_VERTEX\_NUM 20

#define MAXLEN 100

bool a[MAXLEN];

using namespace std;

typedef struct ArcNode

{

int adjvex;

struct ArcNode \*nextarc;

}ArcNode;

typedef struct VNode

{

char data;

ArcNode \*firstarc;

}VNode, AdjList[MAX\_VERTEX\_NUM];

typedef struct

{

AdjList vertices;

int vexnum,arcnum;

}Graph;

typedef struct QNode

{

int data;

struct QNode \*next;

}QNode, \*QueuePtr;

typedef struct

{

QueuePtr front;

QueuePtr rear;

}LinkQueue;

void InitQueue(LinkQueue \*Q)

{

Q->front=Q->rear=(QueuePtr)malloc(sizeof(QNode));

Q->front->next=NULL;

}

void EnQueue(LinkQueue \*Q,int e)

{

QueuePtr p;

p=(QueuePtr)malloc(sizeof(QNode));

p->data=e;

p->next=NULL;

Q->rear->next=p;

Q->rear=Q->rear->next;

}

int DeQueue(LinkQueue \*Q,int \*e)

{

QueuePtr p;

if(Q->front==Q->rear)

return 0;

p=Q->front->next;

\*e=p->data;

Q->front->next=p->next;

if(Q->rear==p)

Q->rear=Q->front;

free(p);

return 0;

}

int Queuejudge(LinkQueue \*Q)

{

if(Q->front==Q->rear)

return 0;

else

return 1;

}

void CreateGraph(Graph \*&G)

{

int i,j;

ArcNode \*p,\*q;

G=(Graph\*)malloc(sizeof(Graph));

cout<<"输入无向图的顶点数:"<<endl;

cin>>G->vexnum;

cout<<"输入无向图的边数:"<<endl;

cin>>G->arcnum;

cout<<"输入各顶点的数据:"<<endl;

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

{

cout<<"顶点"<<i<<":";

getchar();

cin>>G->vertices[i].data;

G->vertices[i].firstarc=NULL;

}

cout<<"请输入各个顶点所连的边:"<<endl;

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

{

int k,g;

cin>>k>>g;

p=(ArcNode\*)malloc(sizeof(ArcNode));

p->adjvex=g;

p->nextarc=G->vertices[k].firstarc;

G->vertices[k].firstarc=p;

q=(ArcNode\*)malloc(sizeof(ArcNode));

q->adjvex=k;

q->nextarc=G->vertices[g].firstarc;

G->vertices[g].firstarc=q;

}

}

void print(Graph \*G)

{

int i;

ArcNode \*p;

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

{

p=G->vertices[i].firstarc;

while(p)

{

cout<<"<"<<G->vertices[i].data<<","<<G->vertices[p->adjvex].data<<">";

p=p->nextarc;

}

cout<<endl;

}

}

void DFS(Graph \*G,int i)

{

ArcNode \*p;

a[i]=1;

cout<<G->vertices[i].data<<" ";

p=G->vertices[i].firstarc;

while(p)

{

if(!a[p->adjvex])

{

DFS(G,p->adjvex);

}

p=p->nextarc;

}

}

void DFSprint(Graph \*G)

{

int i,j;

cout<<"深度优先遍历序列:";

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

a[i]=0;

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

{

if(!a[j])

DFS(G,j);

}

cout<<endl;

}

void BFS(Graph \*G)

{

int i,j;

LinkQueue q;

int b[MAXLEN];

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

b[i]=0;

InitQueue(&q);

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

{

if(!b[j])

{

cout<<G->vertices[j].data<<" ";

b[j]=1;

EnQueue(&q,j);

while(!Queuejudge(&q))

{

DeQueue(&q,&j);

ArcNode \*p=G->vertices[j].firstarc;

while(p)

{

if(!b[p->adjvex]==1)

{

cout<<G->vertices[p->adjvex].data<<" ";

b[p->adjvex]=1;

EnQueue(&q,p->adjvex);

}

p=p->nextarc;

}

}

}

}

}

int main()

{

Graph \*g;

CreateGraph(g);

cout<<"无向图:"<<endl;

print(g);

DFSprint(g);

cout<<"广度优先遍历序列:";

BFS(g);

cout<<endl;

return 0;

}