北京邮电大学软件学院

2016-2017学年第1学期实验报告

课程名称：\_算法与数据结构\_

实验名称：\_图及图的应用\_

实验完成人： \_刘浩博\_

姓名：\_刘浩博\_ 学号：\_2015212086\_ 成绩：\_\_\_\_\_\_\_\_\_\_\_

指导老师： \_贾红娓\_

日期： 2016 年 12 月 13 日

⼀.**实验⽬的**

熟悉图的储存方式和遍历算法。

**⼆. 实验内容**

以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。以用户指定的结点为起点，分别输出每种遍历下的结点访问序列和相应生成树的边集。

**三. 实验环境**

Dev-C++ 5.11

**四. 实验结果**

达到实验要求，程序可完成实验目的，且无错误出现。

**五. 附录**

**Graph\_Traverse.cpp:**

#include<stdio.h>

#include<stdlib.h>

#include<iostream>

#define MAX\_VERTEX\_NUM 20

#define INITSIZE 100

#define INCREAMENT 10

#define MAXQSIZE 100

using namespace std;

typedef struct ArcNode{ /\*边表节点\*/

int adjvex;

struct ArcNode\* nextarc;

}ArcNode;

typedef struct VNode{ /\*顶点表节点\*/

char vertex;

ArcNode\* firstarc;

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

typedef struct{ /\*图结构\*/

AdjList vertices;

int vexnum,arcnum;

}ALGraph;

typedef struct /\*栈\*/

{

int \*base;

int \*top;

int stacksize;

}SqStack;

typedef struct /\*队列\*/

{

int \*base;

int front;

int rear;

}SqQueue;

void Visit(ALGraph G,int v){

cout<<G.vertices[v].vertex;

}

int InitStack(SqStack &S)

{

S.base=(int \*)malloc(INITSIZE \* sizeof(int));

S.top=S.base;

S.stacksize=INITSIZE;

return 0;

}

int Push(SqStack &S,int e)

{

if(S.top-S.base>=S.stacksize)

{

S.base=(int \*)realloc(S.base,(S.stacksize+INCREAMENT)\*sizeof(int));

S.top=S.base+S.stacksize;

S.stacksize+=INCREAMENT;

}

\*S.top++=e;

return 0;

}

int Pop(SqStack &S,int &e)

{

if(S.base!=S.top)

e=\*--S.top;

return 0;

}

int StackEmpty(SqStack S)

{

if(S.base==S.top)

return 1;

else

return 0;

}

int InitQueue(SqQueue &Q)//构建空队列Q

{

Q.base=(int \*)malloc(MAXQSIZE \* sizeof(int));

Q.front=Q.rear=0;

return 1;

}

int QueueEmpty(SqQueue Q)//队列判空

{

if(Q.front==Q.rear)

return 1;

else

return 0;

}

int EnQueue(SqQueue &Q,int e)//入队

{

if((Q.rear+1)%MAXQSIZE==Q.front)

return 0;

Q.base[Q.rear]=e;

Q.rear=(Q.rear+1) % MAXQSIZE;

return 1;

}

int DeQueue(SqQueue &Q,int &e)//出队

{

if(Q.front==Q.rear)

return 0;

e=Q.base[Q.front];

Q.front=(Q.front+1)%MAXQSIZE;

return 1;

}

int CreateGraph(ALGraph \*alg){

int i,j,k;

ArcNode \*s;

cin>>alg->vexnum>>alg->arcnum;

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

cin>>alg->vertices[i].vertex;

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

}

for(k=0;k<alg->arcnum;k++){

cin>>i>>j;

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

s->adjvex=j;

s->nextarc=alg->vertices[i].firstarc; //插入表头

alg->vertices[i].firstarc=s;

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

s->adjvex=i;

s->nextarc=alg->vertices[j].firstarc;

alg->vertices[j].firstarc=s;

}

}

int FirstAdjVex(ALGraph G,int v){

if(!G.vertices[v].firstarc)

return 0;

else

return (G.vertices[v].firstarc->adjvex);

}

int NextAdjVex(ALGraph G,int v,int w){

ArcNode \*p;

p=G.vertices[v].firstarc;

while(p&&p->adjvex!=w) p=p->nextarc;

if(p->adjvex==w && p->nextarc) return(p->nextarc->adjvex);

else return 0;

}

bool visited[1000];

void DFS(ALGraph G,int v);

void DFSTraverseR(ALGraph G){

int k;

for(k=0;k<G.vexnum;++k) visited[k]=false;

for(k=0;k<G.vexnum;++k)

if(!visited[k]) DFS(G,k);

}

void DFS(ALGraph G,int v){

visited[v]=true;

Visit(G,v);

int w;

for(w=FirstAdjVex(G,v);w>0;w=NextAdjVex(G,v,w))

if(!visited[w]) DFS(G,w);

}

void DFSTraverseS(ALGraph G){

SqStack S;

InitStack(S);

int i, j;

int node;

int count = 1;

node = 0;

Push(S,0);

for(j=0; j<G.vexnum; j++) visited[j]=false;

while(count < G.vexnum) //still has node not visited

{

/\* 所有被访问的节点依次入栈，只有node当找不到下一个相连的节点时，才使用出栈节点 \*/

for(j=0; j<G.vexnum; j++)

{

if(visited[j] == false)

{

visited[j] = true;

count++;

Push(S,j); //push node j

break;

}

}

if(j == G.vexnum) //与node相连的节点都已经被访问过，所以需要从stack里取出node的上一个节点，再看该节点是否有相连的节点未被访问

{

Visit(G,node);

Pop(S,node);

}

else //找到与node相连并且未被访问的节点，

node = j;

}

}

void BFSTraverse(ALGraph G){

int v,u,w;

SqQueue Q;

for ( v=0; v<=G.vexnum; ++v ) visited[v] = false;

InitQueue(Q);

for( v=0; v<=G.vexnum; ++v )

if ( !visited[v] ){

visited[v]=true; Visit(G,v); EnQueue(Q, v);

while (!QueueEmpty(Q)){

DeQueue(Q,u);

for(w=FirstAdjVex(G, u); w>0; w=NextAdjVex(G, u, w))

if( !visited[w] ){

visited[w]=true; Visit(G,w); EnQueue(Q, w);

}

} // end while

} // end if

}

int main(){

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

int i;

printf("Please input data in form as follows\n");

printf("|vex arc|\n");

printf("|a b c,etc.(Several vexs)|\n");

printf("|0 1 0 2 2 3 ,etc.(Several arcs)|\n\n");

CreateGraph(G);

printf("\n");

while(1)

{

printf("\nGraph Traverse\n");

printf("===============\n");

printf("1.Print Graph(Adjacency List)\n");

printf("2.Depth\_First Search(Recursion)\n");

printf("3.Depth\_First Search(Stack)\n");

printf("4.Breadth\_First Search(Queue)\n");

printf("5.Exit\n");

printf("\n");

printf("Enter your choice : ");

if(scanf("%d",&i)<=0){

printf("Enter only an Integer\n");

exit(0);

} else {

printf("\n");

switch(i)

{

case 1:

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

printf("%c->",G->vertices[i].vertex);

while(G->vertices[i].firstarc!=NULL){

printf("%d->",G->vertices[i].firstarc->adjvex);

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

}

printf("\n");

}

break;

case 2:

cout<<"Traverse result:";

DFSTraverseR(\*G);

cout<<endl;

break;

case 3:

cout<<"Traverse result:";

DFSTraverseR(\*G);

cout<<endl;

break;

case 4:

cout<<"Traverse result:";

BFSTraverse(\*G);

cout<<endl;

break;

case 5: return 0;

default: printf("Invalid option\n");

}

}

}

return 0;

}

**结果截图：**

