实验五 图的遍历

班级:软件工程一班 姓名:秦源 学号:1525161007

1. 需求分析

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。

1. 概要设计

设图的结点不超过30个，每个结点用一个编号表示（如果一个图有n个结点，则它们的编号分别为1,2,…,n）。通过输入图的全部边输入一个图，每个边为一个数对，可以对边的输入顺序作出某种限制，注意，生成树的边是有向边，端点顺序不能颠倒。

三、 详细设计

//

// main.cpp

// 图的遍历

//

// Created by 秦源 .

// Copyright © 2016年 QinYuan. All rights reserved.

//

#include <iostream>

using namespace std;

//-----图的邻接表储存表示-----

#define MAX\_VERTEX\_NUM 30 //图的节点

typedef struct ArcNode{

int adjvex; //该弧所指向的顶点的位置

struct ArcNode \* nextarc;//指向下一条弧的指针

// InfoType \* info;//该弧相关信息的指针

}ArcNode;

typedef struct VNode{

int data;//顶点信息

ArcNode \* firstarc;//指向第一条依附该顶点的弧的指针

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

typedef struct{

AdjList vertices;

int vexnum,arcnum;//图的当前顶点数和弧数

int kind;//图的种类标志

}ALGraph;

//----测试数据----

//由于取图7.33数据过多，测试数据选用自己的数据

ALGraph \*tempGraph=new ALGraph;

int vexnumber=7;

static void Init(){

tempGraph->vexnum=7;

tempGraph->arcnum=8;

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

tempGraph->vertices[i].data=i+1;

//--V1--

tempGraph->vertices[0].firstarc=new ArcNode;

tempGraph->vertices[0].firstarc->adjvex=1;

tempGraph->vertices[0].firstarc->nextarc=new ArcNode;

tempGraph->vertices[0].firstarc->nextarc->adjvex=2;

tempGraph->vertices[0].firstarc->nextarc->nextarc=new ArcNode;

tempGraph->vertices[0].firstarc->nextarc->nextarc->adjvex=3;

tempGraph->vertices[0].firstarc->nextarc->nextarc->nextarc=NULL;

//--V2--

tempGraph->vertices[1].firstarc=new ArcNode;

tempGraph->vertices[1].firstarc->adjvex=0;

tempGraph->vertices[1].firstarc->nextarc=new ArcNode;

tempGraph->vertices[1].firstarc->nextarc->adjvex=4;

tempGraph->vertices[1].firstarc->nextarc->nextarc=new ArcNode;

tempGraph->vertices[1].firstarc->nextarc->nextarc->adjvex=5;

tempGraph->vertices[1].firstarc->nextarc->nextarc->nextarc=new ArcNode;

tempGraph->vertices[1].firstarc->nextarc->nextarc->nextarc->adjvex=6;

tempGraph->vertices[1].firstarc->nextarc->nextarc->nextarc->nextarc=NULL;

//--V3--

tempGraph->vertices[2].firstarc=new ArcNode;

tempGraph->vertices[2].firstarc->adjvex=0;

tempGraph->vertices[2].firstarc->nextarc=new ArcNode;

tempGraph->vertices[2].firstarc->nextarc->adjvex=6;

tempGraph->vertices[2].firstarc->nextarc->nextarc=NULL;

//--V4--

tempGraph->vertices[3].firstarc=new ArcNode;

tempGraph->vertices[3].firstarc->adjvex=0;

tempGraph->vertices[3].firstarc->nextarc=NULL;

//--V5--

tempGraph->vertices[4].firstarc=new ArcNode;

tempGraph->vertices[4].firstarc->adjvex=1;

tempGraph->vertices[4].firstarc->nextarc=new ArcNode;

tempGraph->vertices[4].firstarc->nextarc->adjvex=5;

tempGraph->vertices[4].firstarc->nextarc->nextarc=NULL;

//--V6--

tempGraph->vertices[5].firstarc=new ArcNode;

tempGraph->vertices[5].firstarc->adjvex=1;

tempGraph->vertices[5].firstarc->nextarc=new ArcNode;

tempGraph->vertices[5].firstarc->nextarc->adjvex=4;

tempGraph->vertices[5].firstarc->nextarc->nextarc=NULL;

//--V7--

tempGraph->vertices[6].firstarc=new ArcNode;

tempGraph->vertices[6].firstarc->adjvex=1;

tempGraph->vertices[6].firstarc->nextarc=new ArcNode;

tempGraph->vertices[6].firstarc->nextarc->adjvex=2;

tempGraph->vertices[6].firstarc->nextarc->nextarc=NULL;

}

//----DFS和BFS共用部分----

void VisiteFuc(int index){

cout<<"V"<<index<<" ";

}

bool \*visited=new bool[vexnumber];

void InitVisited(){

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

visited[i]=false;

}

//----DFS----

void DFS(ALGraph G, int v) {

//从序号为v的顶点出发，对图G做一次深度优先搜索遍历

ArcNode \*p;

VisiteFuc(G.vertices[v].data); //访问v

visited[v]=true; // 打上访问标记

for(p=G.vertices[v].firstarc; p!=NULL; p=p->nextarc )

if( !visited[p->adjvex] ) DFS(G, p->adjvex);

}

//----BFS----

typedef struct QNode{

int data;

QNode \*next;

}QNode,\*QueuePtr;

typedef struct{

QueuePtr front;//队头指针

QueuePtr rear; //队尾指针

}LinkQueue;

void InitQueue(LinkQueue \*Q){

Q->front=new QNode;

Q->rear=Q->front;

}

void EnQueue(LinkQueue \*Q,int v){

QNode \*p=new QNode;

p->data=v;

p->next=NULL;

Q->rear->next=p;

Q->rear=p;

}

int DeQueue(LinkQueue \*Q){

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

exit(-1);

QNode \*p=Q->front->next;

int e=p->data;

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

if(Q->rear==p)

Q->rear=Q->front;

delete p;

return e;

}

bool EmptyQueue(LinkQueue \*Q){

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

return true;

return false;

}

LinkQueue \*Q=new LinkQueue;

void BFS(ALGraph G) {

//从序号为v的顶点出发，对图G做一次广度优先搜索遍历

InitQueue(Q); //初始化一个空队列

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

if( !visited[v] ) {

VisiteFuc(G.vertices[v].data);//访问v

visited[v]=true; //打上访问标记

EnQueue(Q,v); //v入队

while( !EmptyQueue(Q) ) { //当队列非空

int u= DeQueue(Q); //队头元素u出队

ArcNode \*w;

for(w=G.vertices[u].firstarc; w!=NULL; w=w->nextarc)

if ( !visited[w->adjvex] ) {

VisiteFuc(G.vertices[w->adjvex].data);

visited[w->adjvex]=true;

EnQueue(Q, w->adjvex);

}

} //if

} //for

cout<<endl;

}

int main(int argc, const char \* argv[]) {

Init();//导入测试数据

InitVisited();//初始化visited

cout<<"深度遍历（DFS） ："<<endl;

DFS(\*tempGraph,0);//从V1点开始遍历测试数据

cout<<endl;

InitVisited();//初始化visited

cout<<"广度遍历（BFS） ："<<endl;

BFS(\*tempGraph);

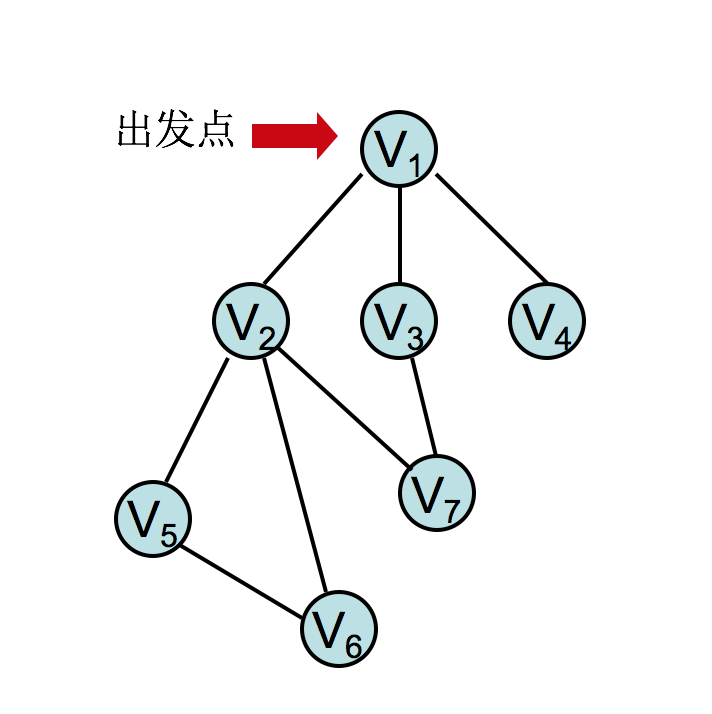
return 0;

}

四、用户使用说明

测试数据：见数据结构教科书Ｐ187图7.33测试数据。

\*由于测试数据过大，选用如下数据进行测试



五、 测试结果



六、 附录

1.深度遍历递归形式实现

2.广度遍历队列形式实现