实习报告：题目六

图及其应用——图的构造与遍历

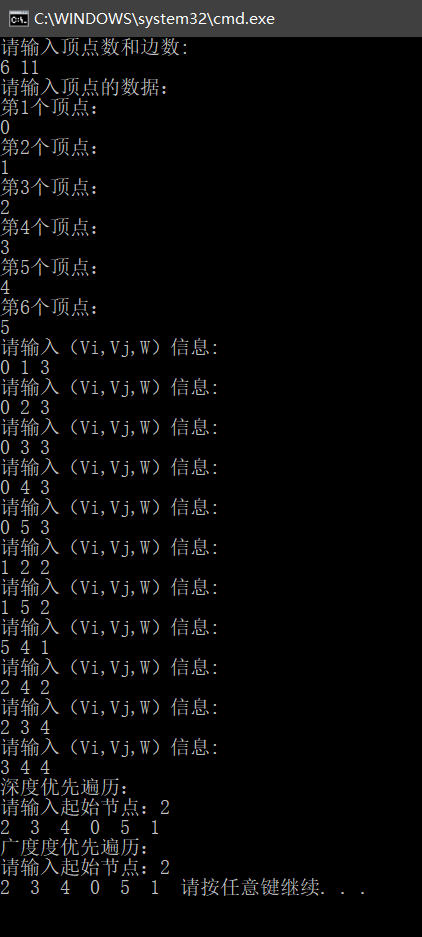
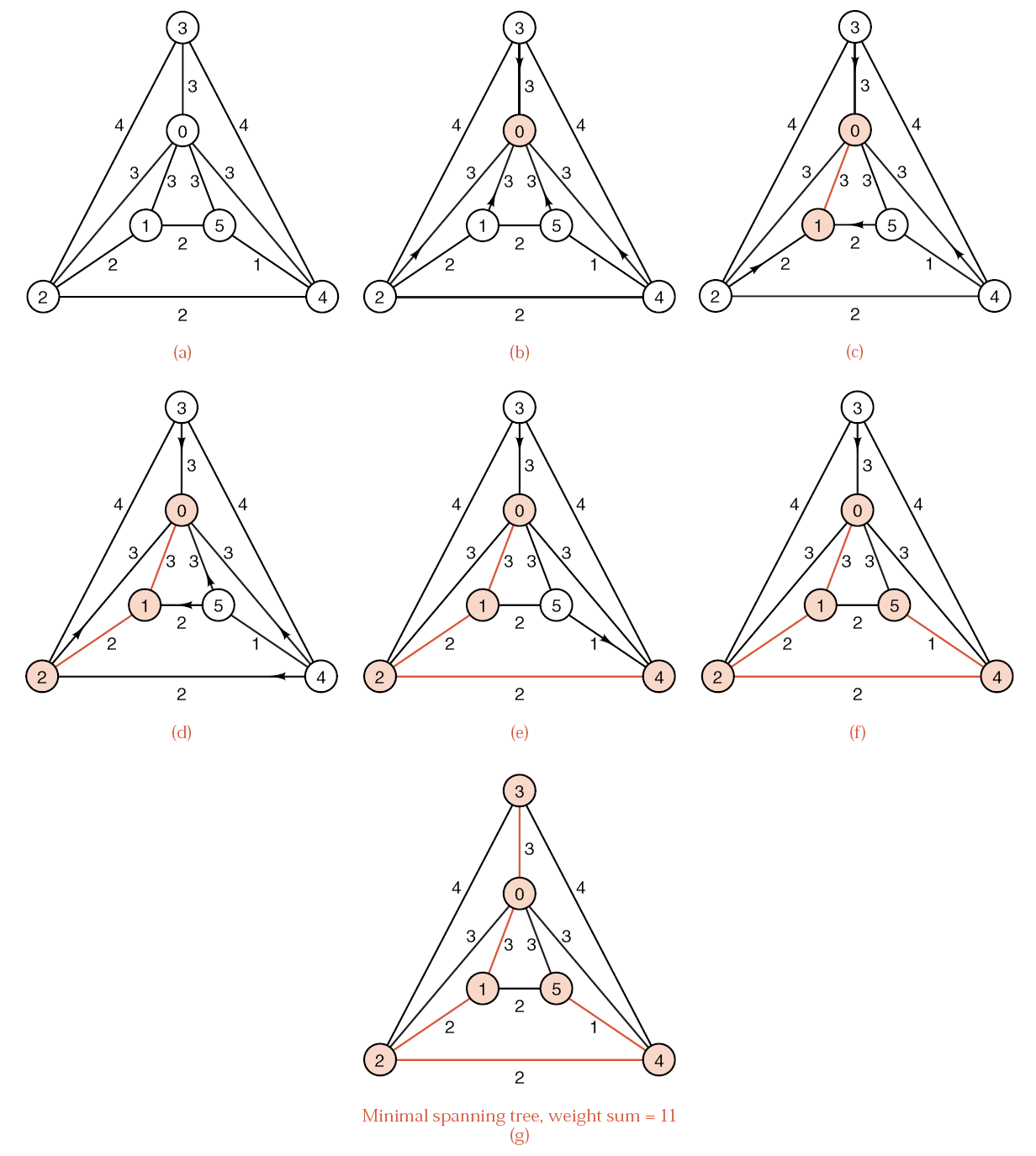
班级：17计算机一班 姓名：陈新朋 学号：1725111006 完成日期：2018.12.08

1. 需求分析

（1）以邻接表为存储结构，创建右边的无向网。图中圆圈代表结点，结点中的数字代表结点数据

（2）依次输入并创建每条边（i，j，w），其中i和j是该条边的两个邻接结点数据，w是该条边上的权值。针对邻接表存储的右图，实现图的深度优先和广度优先两个遍历算法。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

（3）测试数据及结果见图



（6）程序执行的命令为：

1）建立带权无向图；2）进行深度优先遍历，输出结果；3）进行广度优先遍历，输出结果

二、 概要设计

1. 设定图的抽象数据类型定义：

ADT Graph{

数据对象：D={a i |ai∈IntSet,i=1,2,…,n,n>=0}

数据关系：R1={<a i-1 ,a i >|a i-1 ,a i ∈D,i=2,…,n}

基本操作：

/\*\*

\* @brief 创建一个图

\* 先输入顶点数、边数，之后输入各个顶点的数据，最后输入各个边的（Vi,Vj,W）信息

\*/

void CreateGraph(ALGraph & G)

/\*\*

\* @brief 深度优先遍历中进行递归调用、访问结点的函数

\* @condition 已有一个图G

\*/

void DFS(ALGraph G, int i)

/\*\*

\* @brief 深度优先遍历

\* 用户指定结点进行遍历

\* @condition 已有一个图G

\*/

void DFSTraverse(ALGraph G)

/\*\*

\* @brief 广度优先遍历

\* 使用了C++的queue库

\* @condition 已有一个图G

\* -通过标志数组来确定已经访问过的结点

\*/

void BFSTraverse(ALGraph G)

}}ADT list

三、 详细设计

// 邻接表建图遍历.cpp: 定义控制台应用程序的入口点。

//

#include "stdafx.h"

#include <iostream>

#include<queue>

#include <string>

using namespace std;

#define Max\_Vertex\_Num 30

struct ArcNode

{

int adjvex;

ArcNode \*nextarc;

int weight;

};

typedef struct VNode

{

int data;

ArcNode \*firstarc;

}vertices[Max\_Vertex\_Num];

struct ALGraph

{

vertices vertices;

int vexnum, arcnum;

};

int local(ALGraph G, int val)

{

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

{

if (G.vertices[i].data == val)

return i;

}

return -1;

}

void CreateGraph(ALGraph & G)

{

int i, j, k;

int v1, v2;

ArcNode \* e, \*p, \*q;

cout << "请输入顶点数和边数:" << endl;

cin >> G.vexnum >> G.arcnum;

cout << "请输入顶点的数据：" << endl;

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

{

cout << "第" << i + 1 << "个顶点：" << endl;

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

G.vertices[i].firstarc = NULL;

}

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

{

int wei;

cout << "请输入（Vi,Vj,W）信息:" << endl;

cin >> v1 >> v2>>wei;

i = local(G, v1);

j = local(G, v2);

if (G.vertices[i].firstarc == NULL)

{

e = new ArcNode;

e->adjvex = j;

e->nextarc = NULL;

G.vertices[i].firstarc = e;

G.vertices[i].firstarc->weight = wei;

}

else

{

p = G.vertices[i].firstarc;

while (p->nextarc != NULL)

{

p = p->nextarc;

}

e = new ArcNode;

e->adjvex = j;

e->nextarc = NULL;

p->nextarc = e;

G.vertices[i].firstarc = e;

G.vertices[i].firstarc->weight = wei;

}

if (G.vertices[j].firstarc == NULL)

{

e = new ArcNode;

e->adjvex = i;

e->nextarc = NULL;

G.vertices[j].firstarc = e;

G.vertices[j].firstarc->weight = wei;

}

else

{

p = G.vertices[j].firstarc;

while (p->nextarc != NULL)

{

p = p->nextarc;

}

e = new ArcNode;

e->adjvex = i;

e->nextarc = NULL;

p->nextarc = e;

G.vertices[j].firstarc->weight = wei;

}

}

}

bool DFSvisited[Max\_Vertex\_Num];

void DFS(ALGraph G, int i)

{

ArcNode \* p;

DFSvisited[i] = true;

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

p = G.vertices[i].firstarc;

while (p)

{

if (!DFSvisited[p->adjvex])

DFS(G, p->adjvex);

p = p->nextarc;

}

}

void DFSTraverse(ALGraph G)

{

cout << "请输入起始节点：";

int n;

cin >> n;

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

DFSvisited[i] = false;

for (int i = n; i<G.vexnum; i++)

{

if (!DFSvisited[i])

DFS(G, i);

}

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

{

if (!DFSvisited[i])

DFS(G, i);

}

}

bool BFSvisited[Max\_Vertex\_Num];

void BFSTraverse(ALGraph G)

{

ArcNode \* p;

queue<int>q;

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

BFSvisited[i] = false;

cout << "请输入起始节点：";

int n;

cin >> n;

for (int i = n; i<G.vexnum; i++)

{

if (!BFSvisited[i])

{

BFSvisited[i] = true;

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

q.push(i);

while (!q.empty())

{

int count = q.front();

q.pop();

p = G.vertices[count].firstarc;

while (p)

{

if (!BFSvisited[p->adjvex])

{

BFSvisited[p->adjvex] = true;

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

q.push(p->adjvex);

}

p = p->nextarc;

}

}

}

}

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

{

if (!BFSvisited[i])

{

BFSvisited[i] = true;

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

q.push(i);

while (!q.empty())

{

int count = q.front();

q.pop();

p = G.vertices[count].firstarc;

while (p)

{

if (!BFSvisited[p->adjvex])

{

BFSvisited[p->adjvex] = true;

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

q.push(p->adjvex);

}

p = p->nextarc;

}

}

}

}

}

int main()

{

ALGraph test;

CreateGraph(test);

cout << "深度优先遍历：" << endl;

DFSTraverse(test);

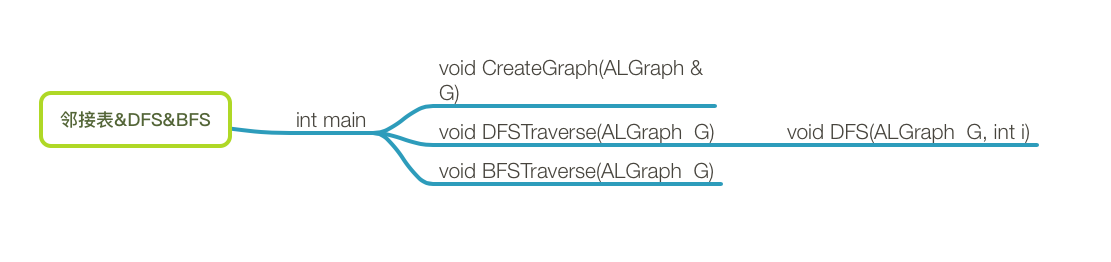
cout << endl;

cout << "广度度优先遍历：" << endl;

BFSTraverse(test);

return 0;

}

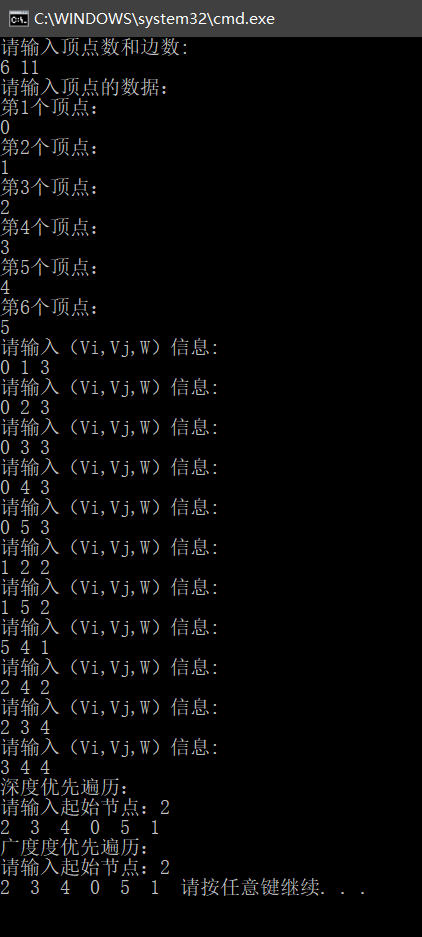


四、 调试分析

1．此次算法难度中等，所需代码量较少，故调试时可以较快找到错误。

3．当一个函数中涉及到多个计数变量的使用时，需要注意不可调用错误。而且这种错误比较隐蔽。

五、 测试结果



六、 用户手册

1.运行代码后，提示“请输入顶点数和边数”

2.输入顶点数和边数后提示继续输入顶点和边的信息

3.分别输入深度优先遍历和广度优先遍历的起始结点后会输出结果。

七、 附录

源程序文件名清单：

邻接表建图遍历.cpp//主程序