# 实验5 图的表示与遍历

**实验日期**：2020年05月27日 20190521340 周吉瑞 **报告日期**：2020年05月27日

### 一、实验目的

1、掌握图的邻接矩阵和邻接表表示

2、掌握图的深度优先和广度优先搜索方法

### 二、实验预习

复习以下概念

1. 深度优先搜索遍历：DFS:遍历类似于树的先序遍历，是树的先序遍历的推广。

(1)从某个顶点V出发，访问顶点并标记为已访问

(2)访问V的邻接点，如果没有访问过，访问该顶点并标记为已访问，然后再访问该顶点的邻接点，递归执行

   如果该顶点已访问过，退回上一个顶点，再检查该顶点的邻接点是否都被访问过，如果有没有访问过的继续向下访问，如果全部都访问过继续退回到上一个顶点，继续同样的步骤。

1. 广度优先搜索遍历：BFS：遍历类似于树的按层次遍历的过程。

(1)从某个顶点V出发，访问该顶点的所有邻接点V1，V2..VN

(2)从邻接点V1，V2...VN出发，再访问他们各自的所有邻接点

(3)重复上述步骤，直到所有的顶点都被访问过

### 三、实验内容和要求

1、分析并完成下面程序，按照图示给出输入数据，并写出运行结果。

（1）**邻接矩阵.h**

//头文件：无向网的邻接矩阵表示法

#include <stdio.h>

#include <iostream>

using namespace std;

#define MaxInt 32767 //表示极大值，即∞

#define MVNum 100 //最大顶点数

#define OK 1

typedef char VerTexType; //假设顶点的数据类型为字符型

typedef int ArcType; //假设边的权值类型为整型

//- - - - -图的邻接矩阵存储表示- - - - -

typedef struct{

VerTexType vexs[MVNum]; //顶点表

ArcType arcs[MVNum][MVNum]; //邻接矩阵

int vexnum,arcnum; //图的当前点数和边数

}AMGraph;

int LocateVex(AMGraph G , VerTexType v); //确定点v在G中的位置

int CreateUDN(AMGraph &G); //采用邻接矩阵表示法，创建无向网G

void PrtAMGraph(AMGraph G); //输出邻接矩阵

int Degree(AMGraph G,int k); //求第k个顶点的度

int LocateVex(AMGraph G , VerTexType v){

//确定点v在G中的位置

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

if(G.vexs[i] == v)

return i;

return -1;

}//LocateVex

int CreateUDN(AMGraph &G){

//采用邻接矩阵表示法，创建无向网G

int i , j , k;

cout <<"请输入总顶点数，总边数，以空格隔开：";

cin >> G.vexnum >> G.arcnum; //输入总顶点数，总边数

cout << endl;

cout << "输入点的名称，如a" << endl;

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

cout << "请输入第" << (i+1) << "个点的名称:";

cin >> G.vexs[i]; //依次输入点的信息

}

cout << endl;

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

//初始化邻接矩阵，边的权值均置为极大值MaxInt

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

G.arcs[i][j] = MaxInt;

}

cout << "输入边依附的顶点及权值，如 a b 5" << endl;

for(k = 0; k < G.arcnum;++k) //构造邻接矩阵

{ VerTexType v1 , v2;

ArcType w;

cout << "请输入第" << (k + 1) << "条边依附的顶点及权值:";

cin >> v1 >> v2 >> w; //输入一条边依附的顶点及权值

i = LocateVex(G, v1) ; j = LocateVex(G, v2) ;

//确定v1和v2在G中的位置，即顶点数组的下标

G.arcs[i][j] = w; //边<v1, v2>的权值置为w

G.arcs[j][i] = G.arcs[i][j] ; //置<v1, v2>的对称边<v2, v1>的权值为w

}//for

return OK;

}//CreateUDN

void PrtAMGraph(AMGraph G) //输出邻接矩阵

{

int i,j;

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

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

{ if( G.arcs[i][j] != MaxInt )

cout << G.arcs[i][j] << "\t";

else

cout << "∞" << "\t";

} //for j

cout << endl;

}//for i

cout <<endl;

}//Ptrgraph

int Degree(AMGraph G,int k) //求第k个顶点的度

{ int j,d=0;

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

if (G.arcs[k][j] != MaxInt)

d = d + G.arcs[k][j];

}

return d;

}//Degree

**（2）主函数：**

/\*

Name:实验5-1 采用邻接矩阵表示法创建无向网并输出和统计各个结点的度

Author: JIA CHUNHUA

\*/

#include "邻接矩阵.h"

int main(){

cout << "\*\*\*\*采用邻接矩阵表示法创建无向网\*\*\*\*" << endl ;

AMGraph G;

CreateUDN(G) ;

cout <<endl;

cout << "该图的邻接矩阵如下：" << endl;

PrtAMGraph(G);

cout << endl<<"\*\*\*\*\*各个顶点的度：\*\*\*\*\*" << endl;

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

{

printf("第%d个顶点 %c 的度：%d\n",i, G.vexs[i] ,Degree(G,i));

}

return 0;

}//main

* 根据下图的结构验证实验5-1

**A**

**B**

**C**

**E**

**D**

**1**

**2**

**3**

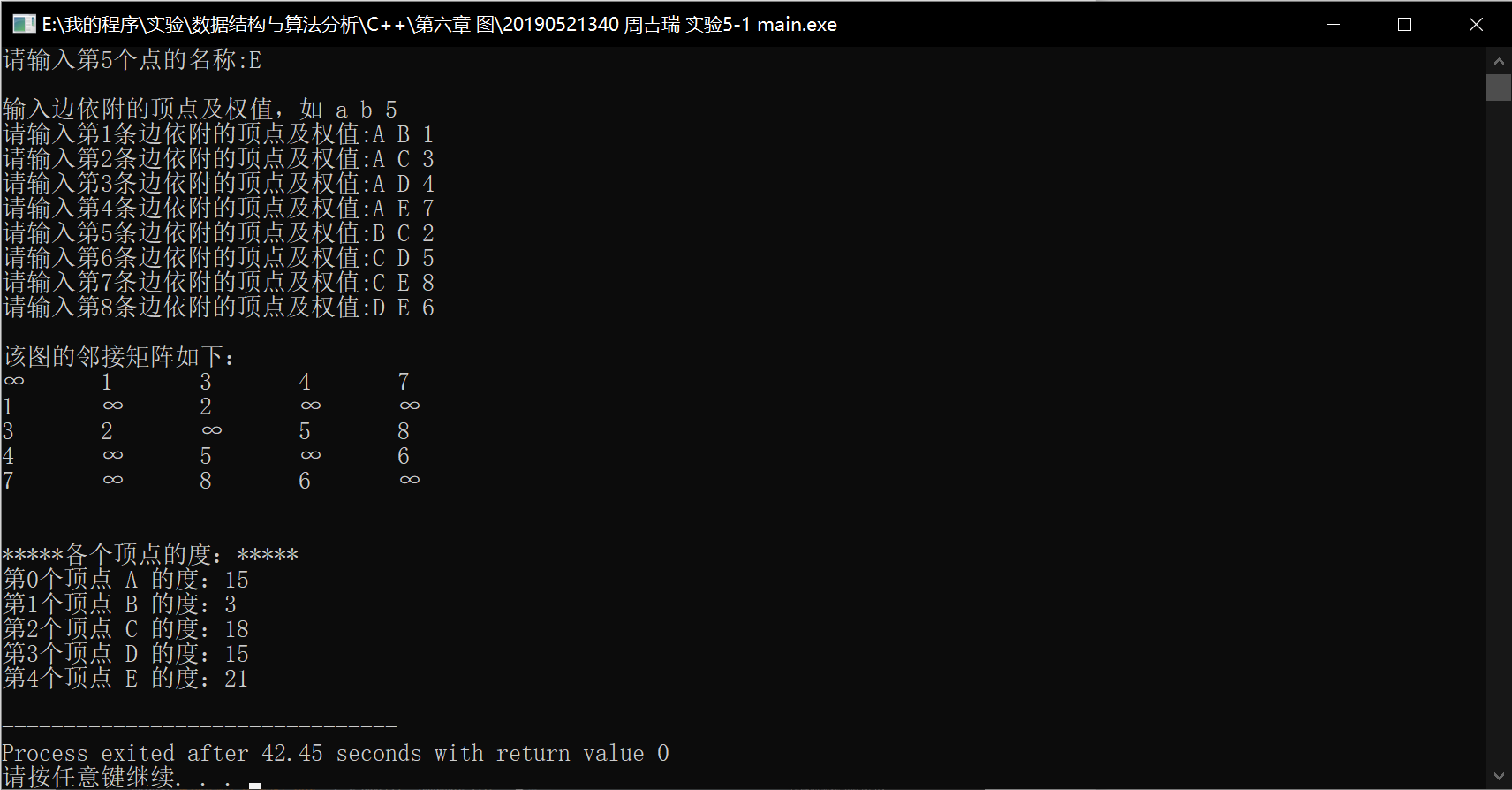
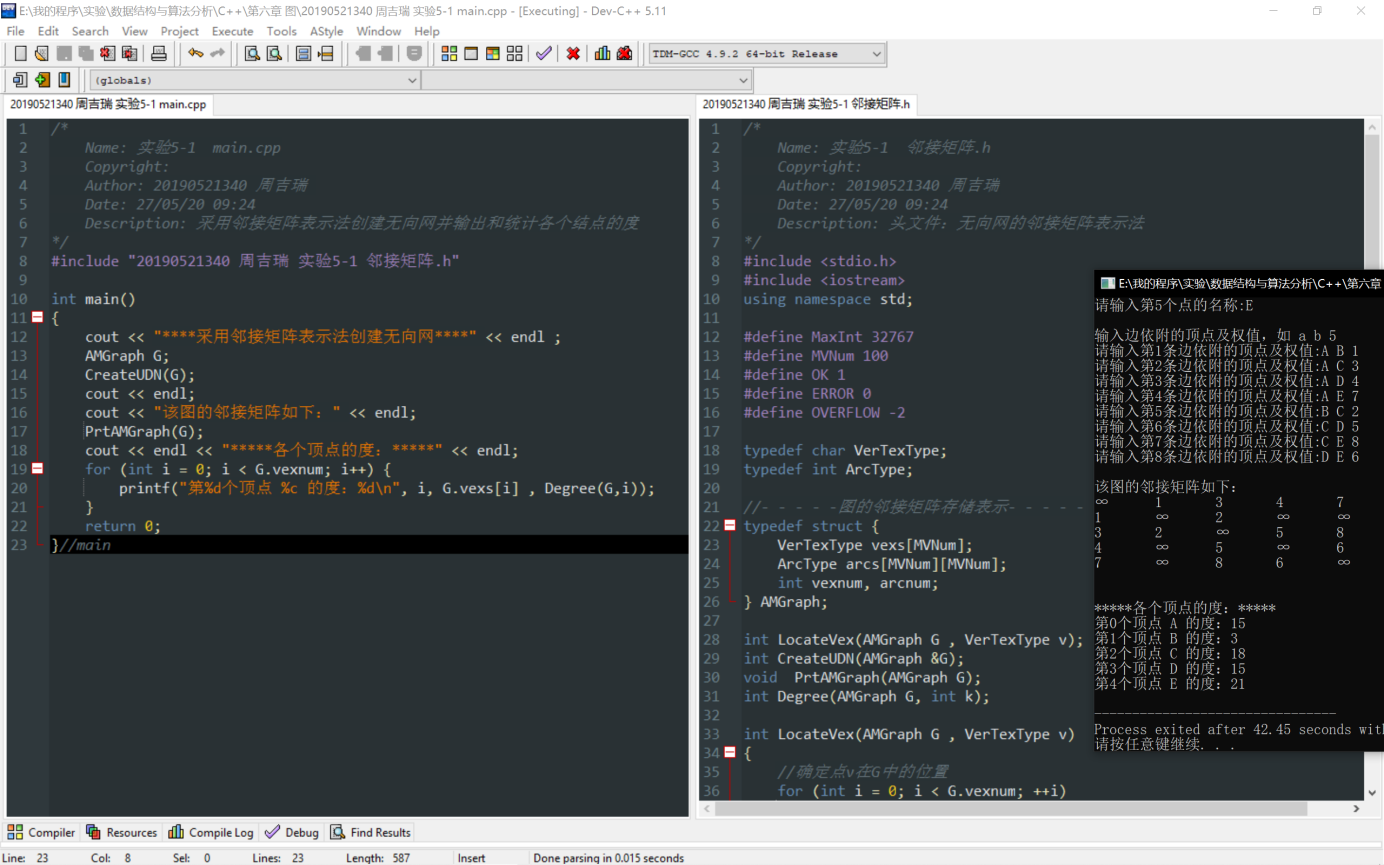
**5**

**4**

**6**

**8**

**7**

* **输入数据为：**
* 
* **运行结果：**
* 

2、阅读并运行下面程序。

//实验5-2 深度优先搜索遍历连通图的递归算法（邻接矩阵存储）

#include "邻接矩阵.h"

bool visited[MVNum]={false}; //访问标志数组，其初值为"false"

int FirstAdjVex(AMGraph G , int v); //返回v的第一个邻接点

int NextAdjVex(AMGraph G , int v , int w); //返回v相对于w的下一个邻接点

void DFS(AMGraph G, int v) **//从第v个顶点出发递归地深度优先遍历图G**

{ cout << G.vexs[v] << " "; visited[v] = true ;

//访问第v个顶点，并置访问标志数组相应分量值为true

int w;

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

//依次检查v的所有邻接点w ，FirstAdjVex(G, v)表示v的第一个邻接点

//NextAdjVex(G, v, w)表示v相对于w的下一个邻接点，w≥0表示存在邻接点

if(!visited[w]) DFS(G, w) ; //对v的尚未访问的邻接顶点w递归调用DFS

}//DFS

int FirstAdjVex(AMGraph G , int v) **//求v的第一个未被访问过的邻接点**

{

int i;

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

if(G.arcs[v][i] != MaxInt && visited[i] == false)

return i;

}

return -1 ;

}//FirstAdjVex

int NextAdjVex(AMGraph G , int v , int w) **//v相对于w的下一个未被访问过的邻接点**

{

int i;

for(i = w ; i < G.vexnum ; ++i){

if(G.arcs[v][i] != MaxInt && visited[i] == false )

return i;

}

return -1;

}//NextAdjVex

int main(){

cout << "\*\*深度优先搜索遍历连通图的递归算法\*\*" << endl;

AMGraph G;

CreateUDN(G);

cout << endl;

cout << "无向连通图G的邻接矩阵如下：" << endl ;

PrtAMGraph(G);

VerTexType c;

int i;

do{

cout << "请输入遍历连通图的起始点：";

cin >> c;

i= LocateVex(G , c) ; //求顶点c的序号

if(i<0)

cout << "该点不存在，请重新输入！" << endl;

}while(i<0);

cout << "深度优先搜索遍历连通图结果：" << endl;

DFS(G, i) ;

cout <<endl;

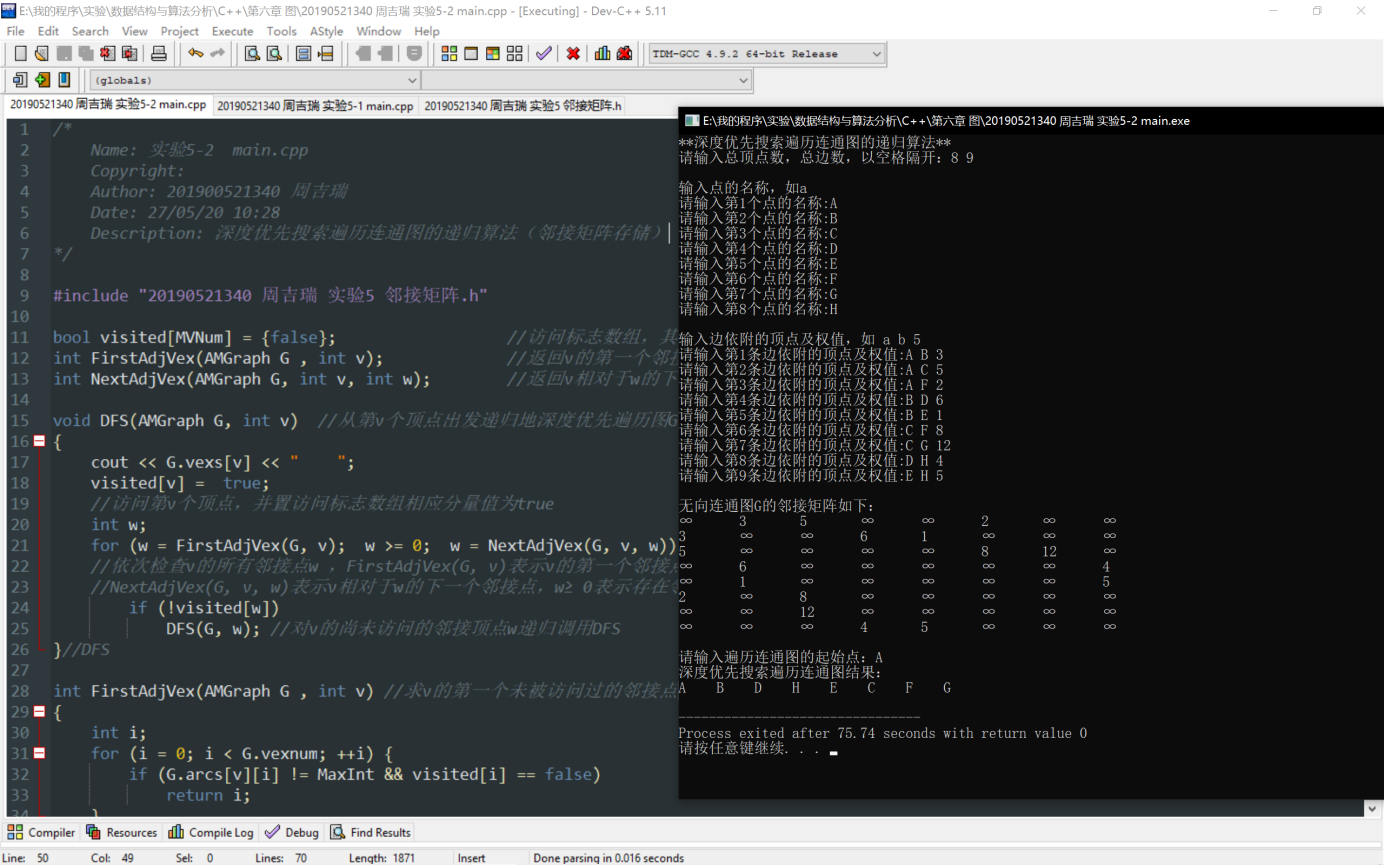
return 0;

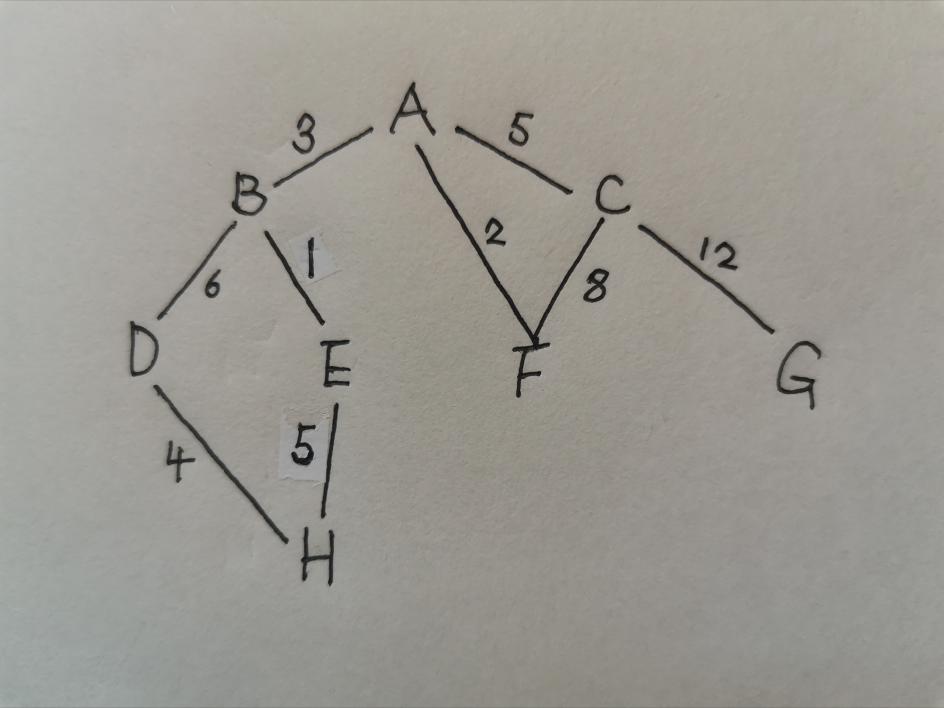
}//main

* 用下面的输入数据验证深度优先遍历算法，写出运行结果。（也可自选一个无向网作为输入数据）

8 9

ABCDEFGH

* A B 3
* A C 5
* A F 2
* B D 6
* B E 1
* C F 8
* C G 12
* D H 4
* E H 5
* **运行结果：**（并画出对应的图）
* 



1. （选做）使用邻接表存储结构实现深度优先遍历。

/\*

Name: 实验5 邻接表.h

Copyright:

Author: 20190521340 周吉瑞

Date: 27/05/20 13:24

Description: 头文件：无向网的邻接表表示法

\*/

#include <stdio.h>

#include <iostream>

using namespace std;

#define MVNum 100 //最大顶点数

#define OK 1

#define ERROR 0

#define OVERFLOW -2

typedef int Status;

typedef char VerTexType; //假设顶点的数据类型为字符型

typedef int ArcType; //假设边的权值类型为整型

//- - - - -图的邻接表存储表示- - - - -

typedef struct ArcNode { //边结点

int adjvex; //该边所指向的顶点的位置

struct ArcNode \*nextarc; //指向下一条边的指针

ArcType info; //和边相关的信息

} ArcNode;

typedef struct VNode { //顶点信息

VerTexType data;

ArcNode \*firstarc; //指向第一条依附该顶点的边的指针

} VNode, AdjList[MVNum]; //AdjList表示邻接表类型

typedef struct { //邻接表

AdjList vertices;

int vexnum, arcnum; //图的当前顶点数和边数

} ALGraph;

int LocateVex(ALGraph G, VerTexType v);

Status CreateUDG(ALGraph &G);

void PrtALGraph(ALGraph G);

int LocateVex(ALGraph G, VerTexType v)

{

//确定点v在G中的位置

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

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

return i;

return -1;

}//LocateVex

Status CreateUDG(ALGraph &G)

{

//采用邻接表表示法，创建无向网G

int i , j , k;

cout << "请输入总顶点数，总边数，以空格隔开：";

cin >> G.vexnum >> G.arcnum; //输入总顶点数，总边数

cout << endl;

cout << "输入点的名称，如a" << endl;

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

cout << "请输入第" << (i+1) << "个点的名称:";

cin >> G.vertices[i].data; //依次输入点的信息

G.vertices[i].firstarc = NULL; //初始化表头结点的指针域为NULL

}

cout << endl;

cout << "输入边依附的顶点及权值，如 a b 5" << endl;

for (k = 0; k < G.arcnum; ++k) { //构造邻接矩阵

VerTexType v1 , v2;

ArcType w;

ArcNode \*p1, \*p2;

cout << "请输入第" << (k + 1) << "条边依附的顶点及权值:";

cin >> v1 >> v2 >> w; //输入一条边依附的顶点及权值

i = LocateVex(G, v1);

j = LocateVex(G, v2);

//确定v1和v2在G中的位置，即顶点G.vertices中的序号

p1 = new ArcNode;

p1->adjvex = j;

p1->info = w; //边<v1, v2>的权值置为w

p1->nextarc = G.vertices[i].firstarc;

G.vertices[i].firstarc = p1;

//将新结点\*p1插入顶点vi的边表头部

p2 = new ArcNode;

p2->adjvex = i;

p2->info = w;

p2->nextarc = G.vertices[j].firstarc;

G.vertices[j].firstarc = p2;

}//for

return OK;

}//CreateUDG

void PrtALGraph(ALGraph G) //输出邻接表

{

int i;

ArcNode \*p;

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

printf("%d %c", i, G.vertices[i].data);

p = G.vertices[i].firstarc;

while (1) {

if (p != NULL) {

printf(" -> [%d,%d]", p->adjvex, p->info);

p = p->nextarc;

}

else

break;

}

cout << endl;

}

}//Ptrglaph

/\*

Name: 实验5-3 main.cpp

Copyright:

Author: 201900521340 周吉瑞

Date: 27/05/20 14:28

Description: 深度优先搜索遍历连通图的递归算法（邻接表存储）

\*/

#include "20190521340 周吉瑞 实验5 邻接表.h"

bool visited[MVNum] = {false}; //访问标志数组，其初值为"false"

void DFS\_AL(ALGraph G, int v) //图G为邻接表类型，从第v个顶点出发深度优先搜索遍历

{

ArcNode \*p;

int w;

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

visited[v] = true;

p = G.vertices[v].firstarc;

while (p != NULL) {

w = p->adjvex;

if (!visited[w])

DFS\_AL(G, w);

p = p->nextarc;

}

}

int main()

{

cout << "\*\*深度优先搜索遍历连通图的递归算法\*\*" << endl;

ALGraph G;

CreateUDG(G);

cout << endl;

cout << "无向连通图G的邻接表如下：" << endl ;

PrtALGraph(G);

VerTexType c;

int i;

do {

cout << "请输入遍历连通图的起始点：";

cin >> c;

i = LocateVex(G, c); //求顶点c的序号

if(i < 0)

cout << "该点不存在，请重新输入！" << endl;

} while (i < 0);

cout << "深度优先搜索遍历连通图结果：" << endl;

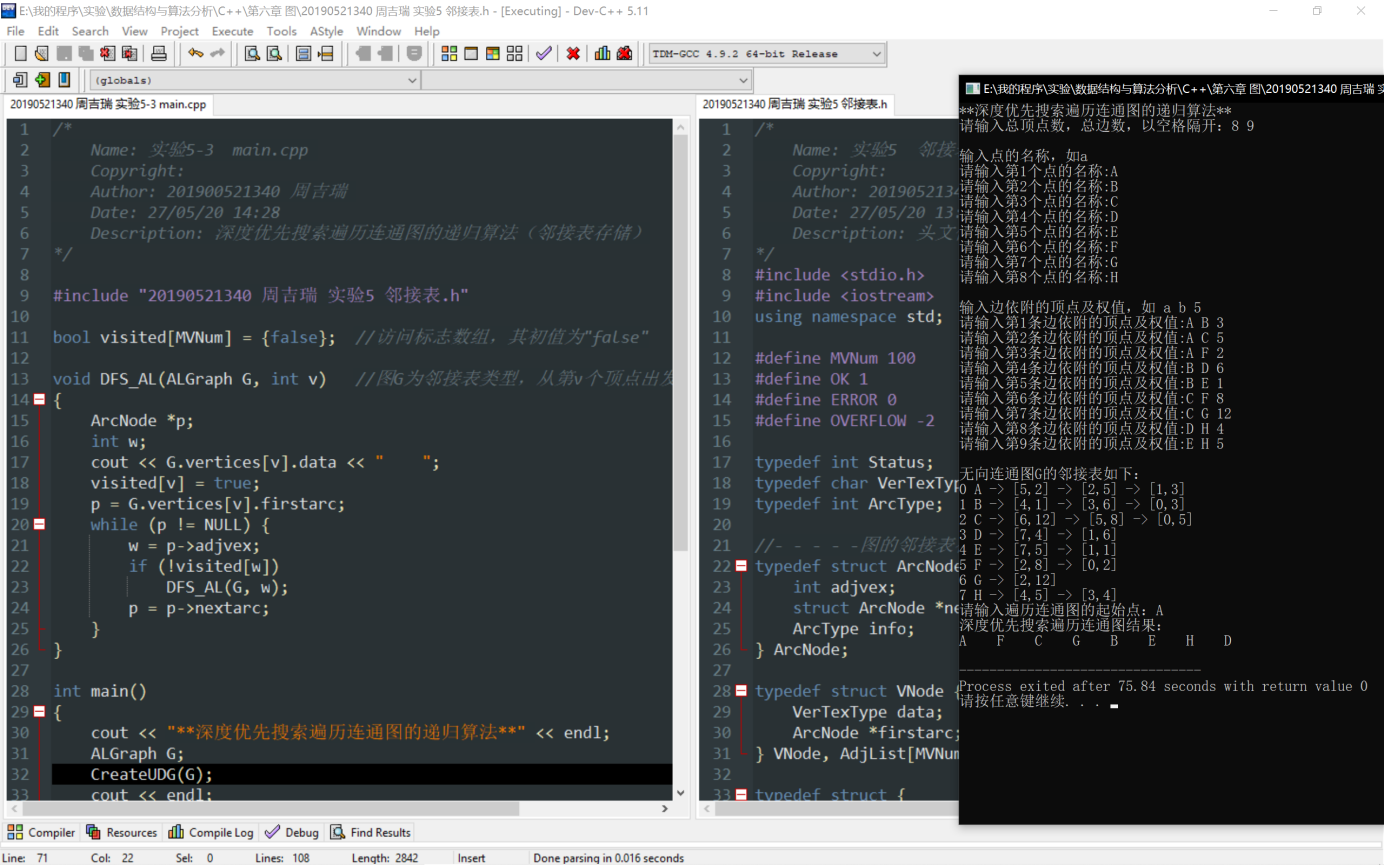
DFS\_AL(G, i);

cout << endl;

return 0;

}//main

运行结果：



1. （选择）任选一种存储结构实现图的广度优先遍历。

/\*

Name: 实验5 队列.h

Copyright:

Author: 20190521340 周吉瑞

Date: 27/05/20 13:20

Description: 头文件：队列 （链队）

\*/

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

#define OK 1

#define ERROR 0

#define OVERFLOW -2

typedef char VerTexType;

typedef int Status;

typedef struct QNode {

VerTexType data;

struct QNode \*next;

} QNode, \*QueuePtr;

typedef struct {

QueuePtr front; //队头指针

QueuePtr rear; //队尾指针

} LinkQueue;

Status InitQueue(LinkQueue &Q) //构造一个空队列Q

{

Q.front = Q.rear = new QNode;

Q.front->next = NULL;

return OK;

}

Status EnQueue(LinkQueue &Q, VerTexType e) //插入元素e为Q的新队尾元素

{

QNode \*p;

p = new QNode;

p->data = e;

p->next = NULL;

Q.rear->next = p;

Q.rear = p;

return OK;

}

Status DeQueue(LinkQueue &Q, VerTexType &e) //删除Q的队头元素，用e返回其值

{

QNode \*p;

if (Q.front == Q.rear)

return ERROR;

p = Q.front->next;

e = p->data;

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

if (Q.rear == p)

Q.rear = Q.front;

delete p;

return OK;

}

Status QueueEmpty(LinkQueue Q) //判断队列为空

{

if (Q.front == Q.rear)

return OK;

else

return ERROR;

}

1. .邻接矩阵实现图的广度优先遍历

/\*

Name: 实验5-4 main.cpp

Copyright:

Author: 20190521340 周吉瑞

Date: 27/05/20 12:54

Description: 广度优先搜索遍历连通图的递归算法（邻接矩阵存储）

\*/

#include "20190521340 周吉瑞 实验5 邻接矩阵.h"

#include "20190521340 周吉瑞 实验5 队列.h"

bool visited[MVNum] = {false}; //访问标志数组，其初值为"false"

int FirstAdjVex(AMGraph G , int v); //返回v的第一个邻接点

int NextAdjVex(AMGraph G, int v, int w); //返回v相对于w的下一个邻接点

void BFS(AMGraph G, int v); //从第v个顶点出发递归地深度优先遍历图G

void BFS(AMGraph G, int v) //从第v个顶点出发递归地广度优先遍历图G

{

cout << G.vexs[v] << " ";

visited[v] = true;

//访问第v个顶点，并置访问标志数组相应分量值为true

LinkQueue Q;

InitQueue(Q); //辅助队列Q初始化，置空

EnQueue(Q, G.vexs[v]); //v进队

VerTexType u;

int n;

while (!QueueEmpty(Q)) { //队列非空

DeQueue(Q, u); //队头元素出队并置位u

n = LocateVex(G, u);

int w;

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

//依次检查v的所有邻接点w ，FirstAdjVex(G, u)表示u的第一个邻接点

//NextAdjVex(G, u, w)表示u相对于w的下一个邻接点，w≥0表示存在邻接点

if (!visited[w]) {

cout << G.vexs[w] << " "; //访问w，并置访问标志数组相应分量值为true

visited[w] = true;

EnQueue(Q, G.vexs[w]); //w进队

}//if

}//while

}//BFS

int FirstAdjVex(AMGraph G , int v) //求v的第一个未被访问过的邻接点

{

int i;

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

if (G.arcs[v][i] != MaxInt && visited[i] == false)

return i;

}

return -1;

}//FirstAdjVex

int NextAdjVex(AMGraph G , int v , int w) //v相对于w的下一个未被访问过的邻接点

{

int i;

for(i = w ; i < G.vexnum ; ++i){

if(G.arcs[v][i] != MaxInt && visited[i] == false)

return i;

}

return -1;

}//NextAdjVex

int main()

{

cout << "\*\*广度优先搜索遍历连通图的递归算法\*\*" << endl;

AMGraph G;

CreateUDN(G);

cout << endl;

cout << "无向连通图G的邻接矩阵如下：" << endl ;

PrtAMGraph(G);

VerTexType c;

int i;

do {

cout << "请输入遍历连通图的起始点：";

cin >> c;

i = LocateVex(G , c); //求顶点c的序号

if(i < 0)

cout << "该点不存在，请重新输入！" << endl;

} while (i < 0);

cout << "广度优先搜索遍历连通图结果：" << endl;

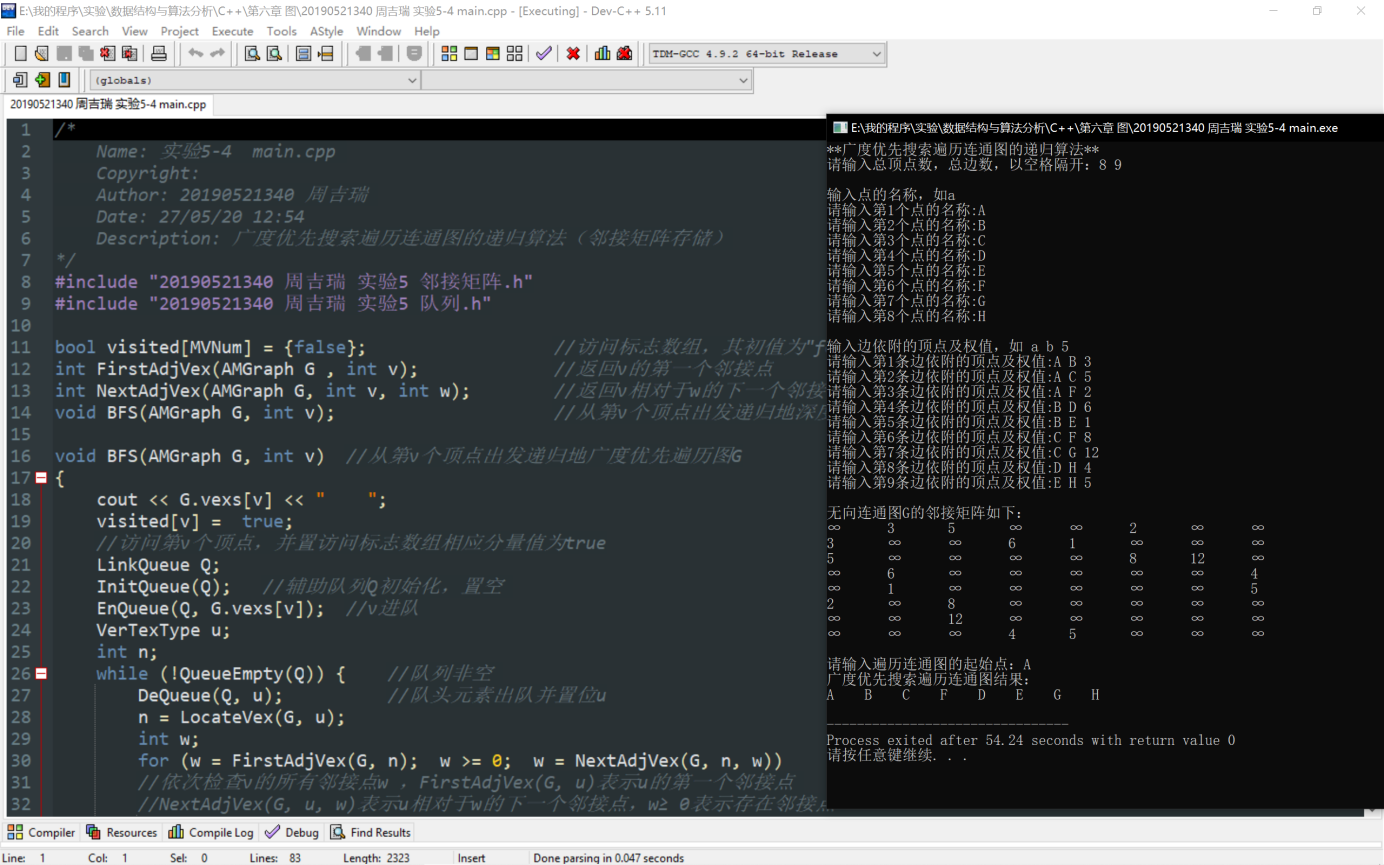
BFS(G, i);

cout << endl;

return 0;

}//main

实验结果：



1. .邻接表实现图的广度优先遍历

/\*

Name: 实验5-5 main.cpp

Copyright:

Author: 20190521340 周吉瑞

Date: 27/05/20 12:54

Description: 广度优先搜索遍历连通图的递归算法（邻接表存储）

\*/

#include "20190521340 周吉瑞 实验5 邻接表.h"

#include "20190521340 周吉瑞 实验5 队列.h"

bool visited[MVNum] = {false}; //访问标志数组，其初值为"false"

int FirstAdjVex(ALGraph G , int v); //返回v的第一个邻接点

int NextAdjVex(ALGraph G , int v , int w); //返回v相对于w的下一个邻接点

void BFS(ALGraph G, int v); //从第v个顶点出发递归地深度优先遍历图G

void BFS(ALGraph G, int v) //从第v个顶点出发递归地广度优先遍历图G

{

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

visited[v] = true;

//访问第v个顶点，并置访问标志数组相应分量值为true

LinkQueue Q;

InitQueue(Q); //辅助队列Q初始化，置空

EnQueue(Q, G.vertices[v].data); //v进队

VerTexType u;

int n;

while (!QueueEmpty(Q)) { //队列非空

DeQueue(Q, u); //队头元素出队并置为u

n = LocateVex(G, u);

int w;

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

//依次检查n的所有邻接点w ，FirstAdjVex(G, n)表示n的第一个邻接点

//NextAdjVex(G, n, w)表示n相对于w的下一个邻接点，w ≥0表示存在邻接点

if (!visited[w]) {

cout << G.vertices[w].data << " "; //访问w，并置访问标志数组相应分量值为true

visited[w] = true;

EnQueue(Q, G.vertices[w].data); //w进队

}//if

}//while

}//BFS

int FirstAdjVex(ALGraph G , int v) //求v的第一个未被访问过的邻接点

{

ArcNode \*p;

p = G.vertices[v].firstarc;

while (p != NULL) {

if (visited[p->adjvex] == false)

return p->adjvex;

else

p = p->nextarc;

}

return -1;

}//FirstAdjVex

int NextAdjVex(ALGraph G , int v , int w) //v相对于w的下一个未被访问过的邻接点

{

ArcNode \*p;

p = G.vertices[v].firstarc;

if (p != NULL) {

while (1) {

if (p->adjvex == w) {

p = p->nextarc;

break;

}

else

p = p->nextarc;

}//while

}//if

while (p != NULL) {

if (visited[p->adjvex] == false)

return p->adjvex;

else

p = p->nextarc;

}//while

return -1;

}//NextAdjVex

int main()

{

cout << "\*\*广度优先搜索遍历连通图的递归算法\*\*" << endl;

ALGraph G;

CreateUDG(G);

cout << endl;

cout << "无向连通图G的邻接表如下：" << endl ;

PrtALGraph(G);

VerTexType c;

int i;

do {

cout << "请输入遍历连通图的起始点：";

cin >> c;

i = LocateVex(G , c); //求顶点c的序号

if(i < 0)

cout << "该点不存在，请重新输入！" << endl;

} while (i < 0);

cout << "广度优先搜索遍历连通图结果：" << endl;

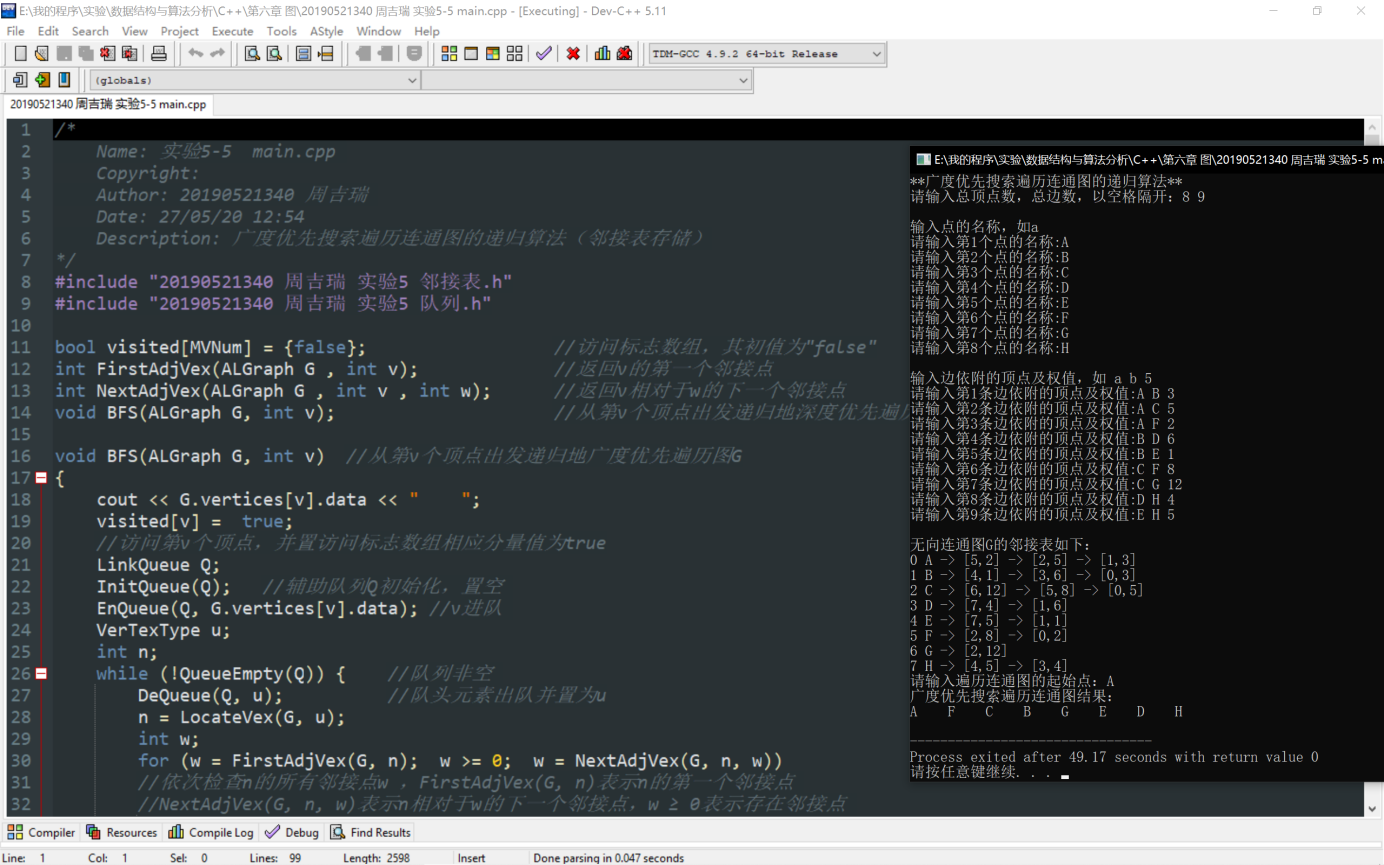
BFS(G, i);

cout << endl;

return 0;

}//main

实验结果：



四、实验小结

对于图的学习应该抓住本质，深刻理解好：

1.图的两种存储结构，以及对应的一些性质。

2.图的两种遍历方式及步骤，必要的代码要刻意记忆。

3.学会利用好作图法及流程图法来对图相关算法进行理解，对实验中的代码进行拆分分析。

4.对于数据结构中相关算法的学习主要要抓住算法思想及算法步骤，而不是去背代码，这样才能良好应用。