**实验5 图**

学号：2022217587姓名：党存远 专业：物联网工程

|  |  |  |  |
| --- | --- | --- | --- |
| **“数据结构课程实验”验收细则** | | | |
| **成绩等级** | **具体表现** | **教师评分**  **（百分制）** | |
| A（100-90] | 1）能较快完成实验；2）编码正确；3）能完整正确地回答教师提问； | □ |  |
| B（90-80] | 1）能够在规定时间内完成实验；2）编码正确；3）能正确地回答教师提问； | □ |  |
| C（80-70] | 1）能够在规定时间内完成实验； | □ |  |
| D（70-60] | 1）虽完成实验但未能在规定时间内完成验收的； | □ |  |
| E（<60） | 1）未能进行验收的； | □ |  |

**5.1 实验目的**

（1）掌握图的存储结构的设计与实现，基本运算的实现；

（2）掌握图的两种遍历算法、遍历生成树及遍历算法的应用；

（3）掌握基于图的关键路径、最短路径等实际问题求解的算法实现。

**5.2 实验要求**

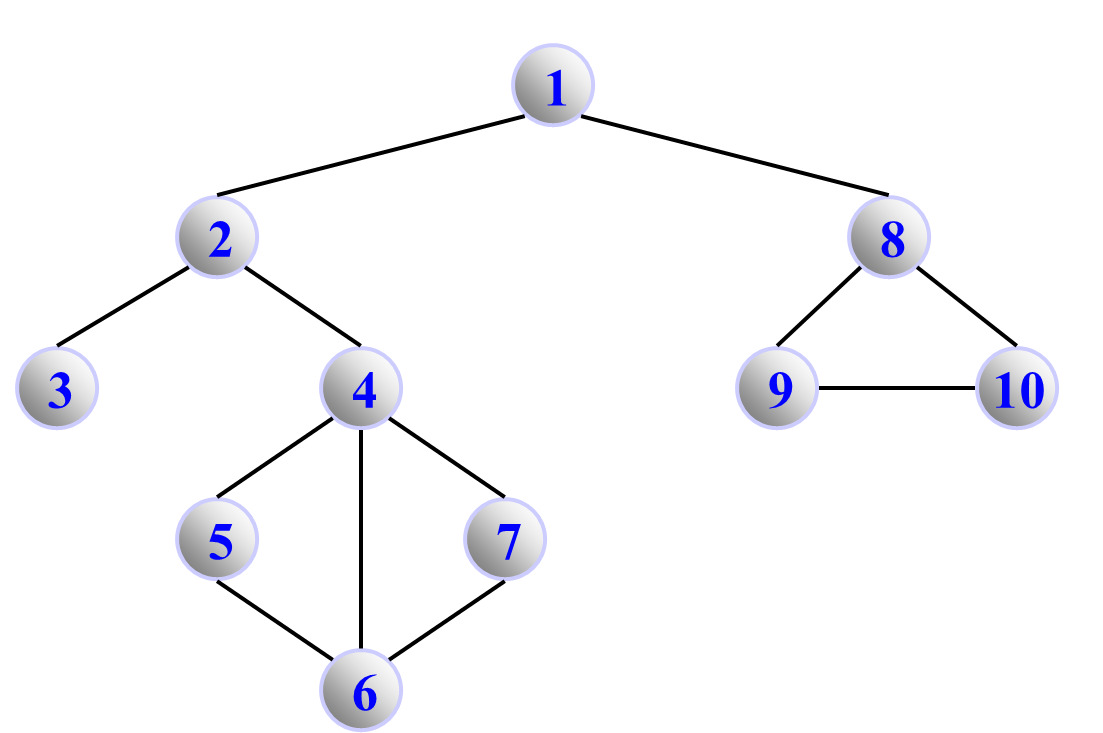
<1> 设计一个无向图的类，并基于此图类实现以下功能：

（1）求给定图中边的个数；

（2）判断该图是否是一棵树；

（3）完成DFS算法，输出DFS序列；

（4）完成BFS算法，输出BFS序列；



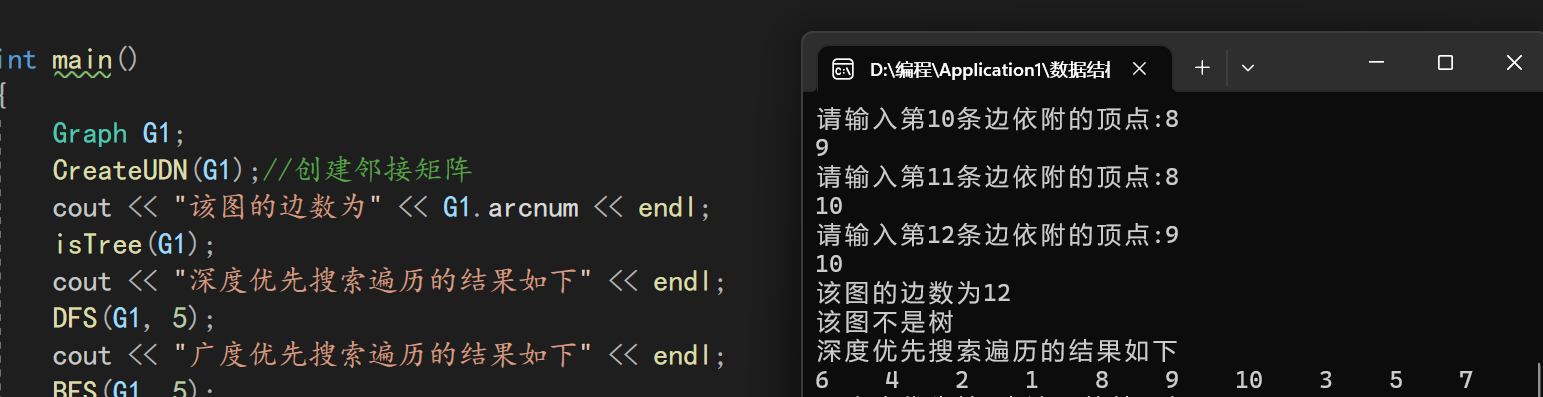
<2>用户自定义迷宫地图，指定入口和出口，采用图相关算法寻找一条出入口之间最短路径（选做）；

**5.3 实验数据要求**

无。

**5.4 运行结果截图及说明**

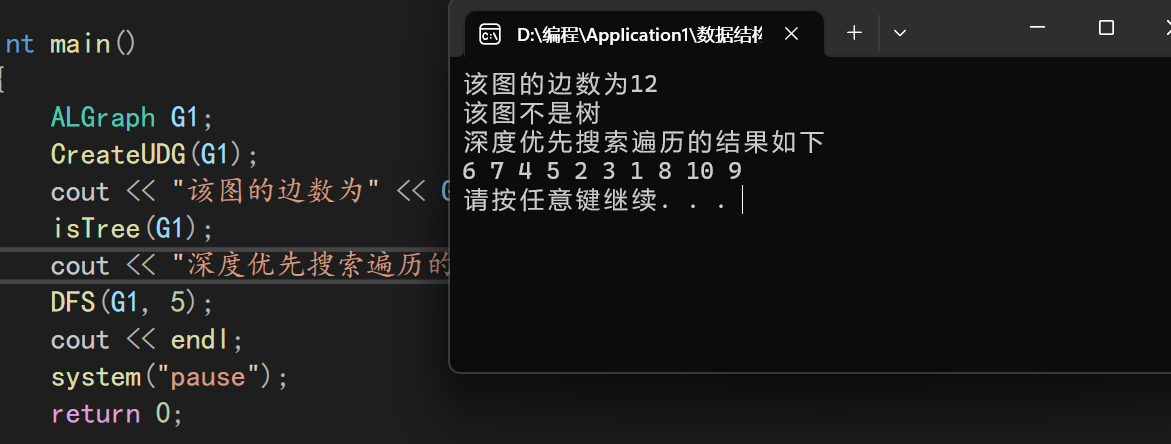
**邻接矩阵实现【深度优先搜索遍历】：**

****

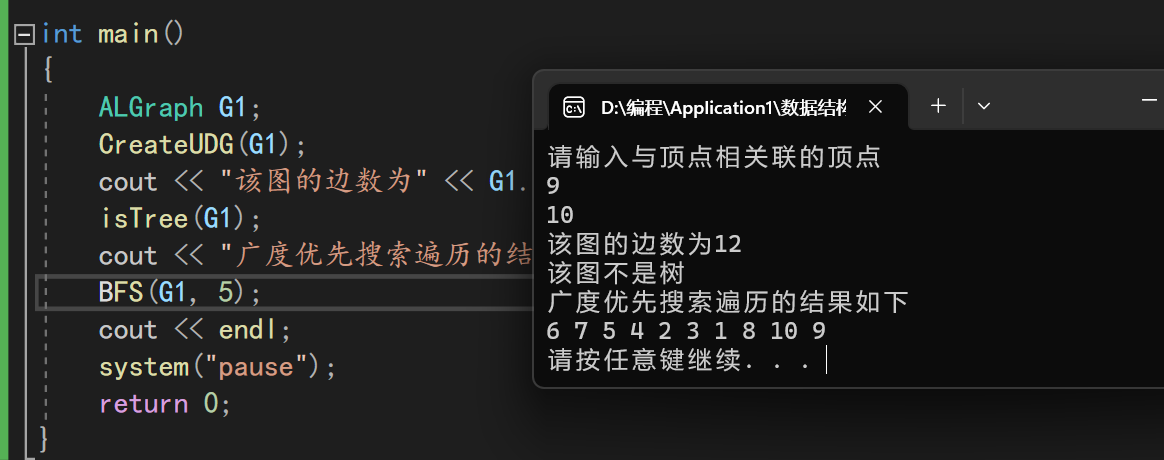
**邻接矩阵实现【广度优先搜索遍历】：**

****

**邻接表实现【深度优先搜索遍历】:**

****

**邻接表实现【广度优先搜索遍历】:**

****

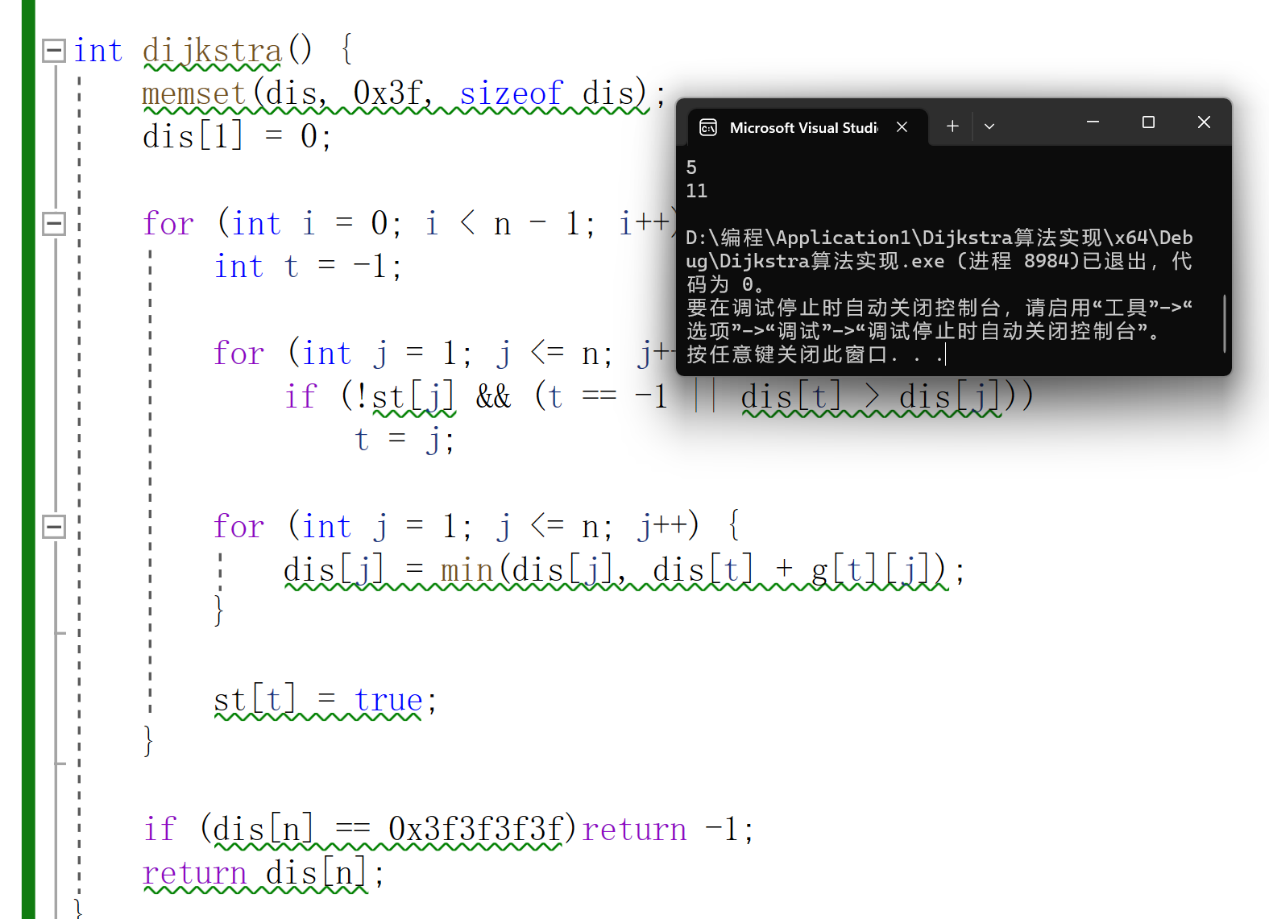
用户自定义迷宫地图，指定入口和出口，采用图相关算法寻找一条出入口之间最短路径（选做）:

测试图示:

【入口为1，出口为5，

输出最短路径:】





**5.5 附源代码**

bool isTree(Graph G)

{

int n = G.arcnum;

int m = G.vexnum;

if (n == m - 1)

{

cout << "该图是树" << endl;

return true;

}

else

{

cout << "该图不是树" << endl;

return false;

}

}

bool isTree(ALGraph G)

{

int n = G.vexnum;

int m = G.arcunm;

if (m == n - 1)

{

cout << "该图是树" << endl;

return true;

}

else

{

cout << "该图不是树" << endl;

return false;

}

}

**邻接矩阵实现【深度优先搜索遍历】：**

bool isTree(Graph G)

{

int n = G.arcnum;

int m = G.vexnum;

if (n == m - 1)

{

cout << "该图是树" << endl;

return true;

}

else

{

cout << "该图不是树" << endl;

return false;

}

}

oid DFS(Graph G, int v) {

//图G为邻接矩阵类型

int w;

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

for (w = 0; w < G.vexnum; w++) //依次检查邻接矩阵v所在的行

if ((G.arcs[v][w] != 0) && (!visited[w])) DFS(G, w); //G.arcs[v][w]!=0表示w是v的邻接点，如果w未访问，则递归调用DFS

}//DFS

**邻接矩阵实现【广度优先搜索遍历】：**

int FirstAdjVex(Graph G, int v) {

//返回v的第一个邻接点

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

if (G.arcs[v][i] == 1 && visited[i] == false)

return i;

}

return -1;

}//FirstAdjVex

int NextAdjVex(Graph G, int v, int w) { //w是在邻接矩阵中的位置

//v表示是哪一个顶点，w表示在与v相连一个邻接点

//返回v相对于w的下一个邻接点，也就是在邻接矩阵中访问第一个邻接点之后，找寻下一个邻接点

int i;

for (i = w; i < G.vexnum; ++i) {//寻找在w之后的第一个v的邻接点

if (G.arcs[v][i] == 1 && visited[i] == false)

return i;

}

return -1;

}//NextAdjVex

//广度优先搜索遍历需要用到队列

void BFS(Graph G, int v) {

//按广度优先非递归遍历连通图G

queue<int> que;//容器

ArcType u;

ArcType w;

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

que.push(v); //v进队

while (!que.empty()) { //队列非空

int u = que.front();

que.pop(); //队头元素出队并置为u

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

//依次检查u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]) { //w为u的尚未访问的邻接顶点

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

que.push(w); //w进队

}//if

}//for

}//while

}//BFS

**邻接表实现【深度优先搜索遍历】:**

void DFS(ALGraph G, int v)

{

cout << G.vertices[v].data << " "; //访问第v个顶点，v是这个顶点在顶点表中的数据

visited[v] = true;

ArcNode\* p = G.vertices[v].firstarc; //指向这个顶点所相连的第一个边结点

while (p != NULL)

{

if (visited[p->adjvex] != true) //若第一个邻接点未被访问则继续直到将所有的邻接点都访问结束

{

DFS(G, p->adjvex); //递归访问与下一个顶点相连接的顶点

}

p = p->nextarc; //寻找下一个邻接点

}

}**邻接表实现【广度优先搜索遍历】:**

void BFS(ALGraph G, int v)

{

queue<int>que;

ArcType u;

ArcType w;

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

visited[v] = true;

que.push(G.vertices[v].data);

while (!que.empty())

{

char m = que.front();

int u = LocateVex(G, m);

que.pop();

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

{

if (visited[w] != true)

{

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

visited[w] = true;

que.push(G.vertices[w].data);

}

}

}

}

用户自定义迷宫地图，指定入口和出口，采用图相关算法寻找一条出入口之间最短路径（选做）:

int dijkstra() {

memset(dis, 0x3f, sizeof dis);

dis[1] = 0;

for (int i = 0; i < n - 1; i++) {

int t = -1;

for (int j = 1; j <= n; j++)

if (!st[j] && (t == -1 || dis[t] > dis[j]))

t = j;

for (int j = 1; j <= n; j++) {

dis[j] = min(dis[j], dis[t] + g[t][j]);

}

st[t] = true;

}

if (dis[n] == 0x3f3f3f3f)return -1;

return dis[n];

}

int main() {

cin >> n >> m;

memset(g, 0x3f, sizeof g);

while (m--) {

int x, y, z;

cin >> x >> y >> z;

g[x][y] = min(g[x][y], z);

}

cout << dijkstra() << endl;

return 0;

}

**5.6 调试过程中出现的bug总结**

1、在进行本次实验中，分别选用了邻接矩阵表示法以及邻接表的方法来完成各项要求

2、求图中的边数时，在刚开始创建图的时候就需要设置顶点数和边数。

3、判断图是否为树，主要是顶点的前驱数是否唯一，若顶点的前驱数唯一，进而可以得到边的个数等于顶点的数目-1，通过此来判断该图是否为树。

4、利用邻接矩阵进行深度优先遍历算法，首先设置一个访问数组，每次访问顶点之后将标志设为True.

5、在创建邻接矩阵时数组下标为0的位置也存放元素，这样和Visited数组对应起来，保证每个顶点被访问之后不会再被访问。

6、在进行广度优先搜索遍历时，首先访问第一个顶点，将第一个顶点入队，如果队列不为空，则将与该顶点相连的所有顶点都如入队。

7、在编写求出该顶点所有顶点的邻接点时，编写了两个函数，一个函数是求出这个顶点的第一个未被访问邻接点，以及求出这个已经找到的邻接点后的下一个邻接点，之后每找到一个邻接点，都对这个邻接点进行访问，同时对该邻接点进行入队操作，直到将第一次传入的顶点全部访问后，再将第一个邻接点进行出队操作，求出这个出队的邻接点的所有临界点访问并且继续入队直到队为空则停止访问。

8、在求出所给迷宫指定入口与出口之间的最短路径时，使用了Dijiesitela算法，首先计算出所给的入口顶点到其余各个顶点的直接距离【如果能直接到达，就记录该条路的权值，如果不能直接到达，就记录为无穷大】，之后将第一次所有的路径中找出最短的一条路径以及这条路径所到达的第一个顶点。之后将该顶点视作已知顶点，计算出原来的入口顶点到其余各个顶点的路径是否因为该顶点的加入而减少，如果减少，则更新路径的权值，如果没有减少则不变，重复此操作直到找到该顶点到其余各个顶点的最短路径【若不能到达则输出为无穷】，若可以到达则输出最短路径的长度。

9、在编写Dijiesitela算法时，在记录初始入口顶点到其余顶点的路径时，需要设置一个数组来存放信息，之后如果新加入了顶点，则该顶点标记为已经找到，不在寻找到达此个顶点的最短路径。