

《数据结构》

实验报告

姓名： 庄李晨

学号： 10522238

成绩：

电子与计算机工程学院

School of Electronic & Computer Engineering

2023年11月

【实验名称】图的实现

【实验目的】

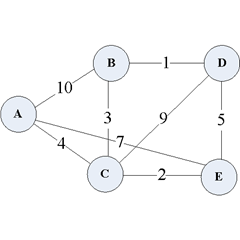
一、掌握图的邻接矩阵存储的实现，熟悉基本操作。

二、掌握图的邻接表存储的实现，熟悉基本操作。

【实验内容】

1. 实验要求：

1、仔细阅读图实验代码，实现无向网的邻接矩阵存储，补充下面8个函数的实现过程。补充完成main（）函数的定义，以下图所示图调试程序，给出测试结果。



（1）在图G的顶点数组中查找顶点V，返回顶点的下标

int LocateVex(MGraph G,VertexType v){ }

（2）采用邻接矩阵表示法，构造无向网G。

bool CreateUDN(MGraph &G){ }

（3）显示图G的邻接矩阵,即按行列输出二维数组

void Display(MGraph G){ }

（4）对图G进行深度优先遍历

void DFSTraverse(MGraph G){ }

（5）从v顶点出发对图G进行深度优先遍历的递归算法

void DFS(MGraph G,int v){ }

（6）对图G进行广度优先遍历

void BFSTraverse(MGraph G){ }

2、选作题目（七选一）

第一题：实现图的邻接表存储，完成题目1的要求；

第二题：实现图的最小生成树算法：普里姆算法(Prim)

第三题：实现图的最小生成树算法：克鲁斯卡尔算法 (Kruskal)

第四题：实现拓扑排序算法

第五题：实现求解关键路径算法

第六题：实现最短路径算法：Dijkstra算法

第七题：实现最短路径算法：Floyd算法

3、选做题

岛屿数量：给你一个由 '1'（陆地）和 '0'（水）组成的的二维网格，请你计算网格中岛屿的数量。岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。此外，你可以假设该网格的四条边均被水包围。

示例 1：

输入：

grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

]

输出：1

示例 2：

输入：grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

输出：3

1. 程序清单

1、

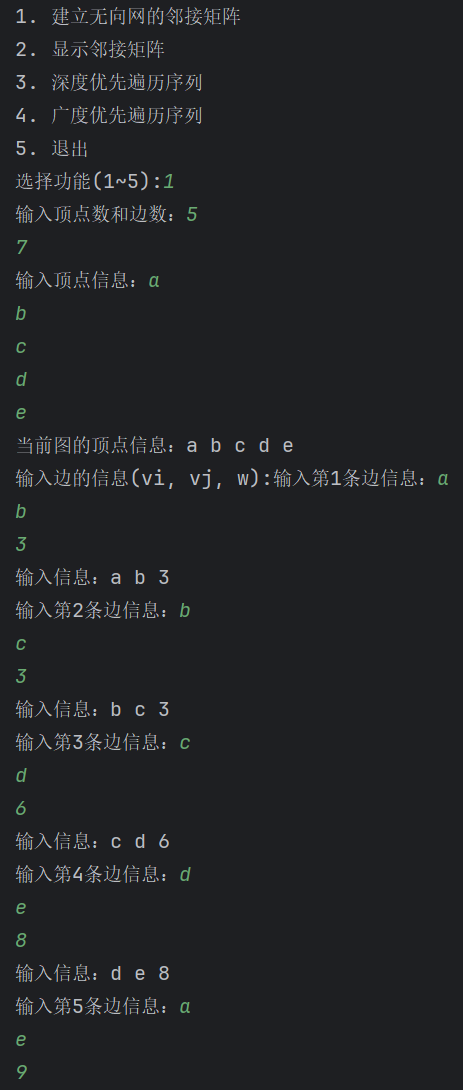
#include <iostream>  
#include <limits>  
#include <queue>  
using namespace std;  
  
#define INFINITY numeric\_limits<int>::max() //最大值 无穷  
#define MAX\_VERTEX\_NUM 20 //最大顶点个数  
typedef char VertexType; //图的顶点的类型  
typedef enum{*DG*,*DN*,*UDG*,*UDN*}GraphKind; //{有向图，有向网，无向图，无向网}  
bool visited[MAX\_VERTEX\_NUM]; //遍历算法使用的数组  
typedef struct { // 图的定义  
 VertexType vexs[MAX\_VERTEX\_NUM]; // 顶点信息  
 int arcs[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM]; // 弧的信息  
 int vexnum, arcnum; // 顶点数，弧数  
 GraphKind kind; // 图的种类标志  
} MGraph;  
  
// 在图G的顶点数组中查找顶点V，返回顶点的下标  
int LocateVex(MGraph G,VertexType v) {  
 for(int i = 0; i < G.vexnum; ++i) {  
 if(G.vexs[i] == v) {  
 return i;  
 }  
 }  
 return -1;  
}  
  
//采用邻接矩阵表示法，构造无向网G。算法7.2（注：改代码省略IncInfo的定义，请省略算法中对IncInfo的操作  
bool CreateUDN(MGraph &G) {  
 cout << "输入顶点数和边数：";  
 cin >> G.vexnum >> G.arcnum;  
  
 cout << "输入顶点信息：";  
 for (int i = 0; i < G.vexnum; ++i) {  
 cin >> G.vexs[i];  
 }  
  
 for (int i = 0; i < G.vexnum; ++i) {  
 for (int j = 0; j < G.vexnum; ++j) {  
 G.arcs[i][j] = INFINITY; // 初始化邻接矩阵，表示无穷大  
 }  
 }  
  
 cout << "当前图的顶点信息：";  
 for (int i = 0; i < G.vexnum; ++i) {  
 cout << G.vexs[i] << " ";  
 }  
 cout << endl;  
  
 cout << "输入边的信息(vi, vj, w):";  
 const int MAX\_TRIES = 3; // 设置最大尝试次数  
 int tries = 0;  
  
 for (int k = 0; k < G.arcnum; ++k) {  
 VertexType vi, vj;  
 int w;  
 cout << "输入第" << k + 1 << "条边信息：";  
 cin >> vi >> vj >> w;  
  
 int m = LocateVex(G, vi);  
 int n = LocateVex(G, vj);  
  
 cout << "输入信息：" << vi << " " << vj << " " << w << endl;  
  
 if (m != -1 && n != -1) {  
 G.arcs[m][n] = G.arcs[n][m] = w;  
 } else {  
 cout << "输入的顶点不存在，请重新输入有效的边信息。" << endl;  
 k--;  
  
 tries++;  
 if (tries >= MAX\_TRIES) {  
 cout << "连续多次输入无效边信息，退出输入。" << endl;  
 break;  
 }  
 }  
 }  
  
 G.kind = *UDN*; // 图的种类为无向网  
 return true;  
}  
  
//显示图G的邻接矩阵,即按行列输出二维数组  
void Display(MGraph G) {  
 cout << "邻接矩阵：" << endl;  
 for(int i = 0; i < G.vexnum; ++i) {  
 for(int j = 0; j < G.vexnum; ++j) {  
 if(G.arcs[i][j] == INFINITY) {  
 cout << "∞" << "\t";  
 } else {  
 cout << G.arcs[i][j] << "\t";  
 }  
 }  
 cout << endl;  
 }  
}  
  
// 对图G进行深度优先遍历 算法7.4（注：省略Visit（）函数的定义，遍历操作为输出）  
void DFSTraverse(MGraph G, int v) {  
 visited[v] = true;  
 cout << G.vexs[v] << " ";  
  
 for (int w = 0; w < G.vexnum; ++w) {  
 if (G.arcs[v][w] != INFINITY && !visited[w]) {  
 DFSTraverse(G, w);  
 }  
 }  
}  
  
//从v顶点出发对图G进行深度优先遍历的递归算法 算法7.5  
void DFS(MGraph G) {  
 cout << "深度优先遍历序列：";  
 for(int i = 0; i < G.vexnum; ++i) {  
 visited[i] = false;  
 }  
  
 for(int i = 0; i < G.vexnum; ++i) {  
 if(!visited[i]) {  
 DFSTraverse(G, i);  
 }  
 }  
 cout << endl;  
}  
  
//对图G进行广度优先遍历 算法7.6（使用之前队列代码，求邻接点的方式如上个函数）  
void BFSTraverse(MGraph G) {  
 cout << "广度优先遍历序列：";  
 for(int i = 0; i < G.vexnum; ++i) {  
 visited[i] = false;  
 }  
  
 for(int i = 0; i < G.vexnum; ++i) {  
 if(!visited[i]) {  
 queue<int> Q;  
 Q.push(i);  
 visited[i] = true;  
  
 while(!Q.empty()) {  
 int v = Q.front();  
 Q.pop();  
 cout << G.vexs[v] << " ";  
  
 for(int w = 0; w < G.vexnum; ++w) {  
 if(G.arcs[v][w] != INFINITY && !visited[w]) {  
 Q.push(w);  
 visited[w] = true;  
 }  
 }  
 }  
 }  
 }  
 cout << endl;  
}  
  
  
int main(void) {  
 MGraph G;  
 int c = 0;  
 while (c != 5) {  
 cout << endl << "1. 建立无向网的邻接矩阵";  
 cout << endl << "2. 显示邻接矩阵";  
 cout << endl << "3. 深度优先遍历序列";  
 cout << endl << "4. 广度优先遍历序列";  
 cout << endl << "5. 退出";  
 cout << endl << "选择功能(1~5):";  
 cin >> c;  
 switch (c) {  
 case 1:  
 CreateUDN(G);  
 break;  
 case 2:  
 Display(G);  
 break;  
 case 3:  
 DFS(G);  
 break;  
 case 4:  
 BFSTraverse(G);  
 break;  
 case 5:  
 cout << "结束操作" << endl;  
 break;  
 default:  
 cout << "无效选择，请重新输入。" << endl;  
 break;  
 }  
 }  
  
 return 0;  
}

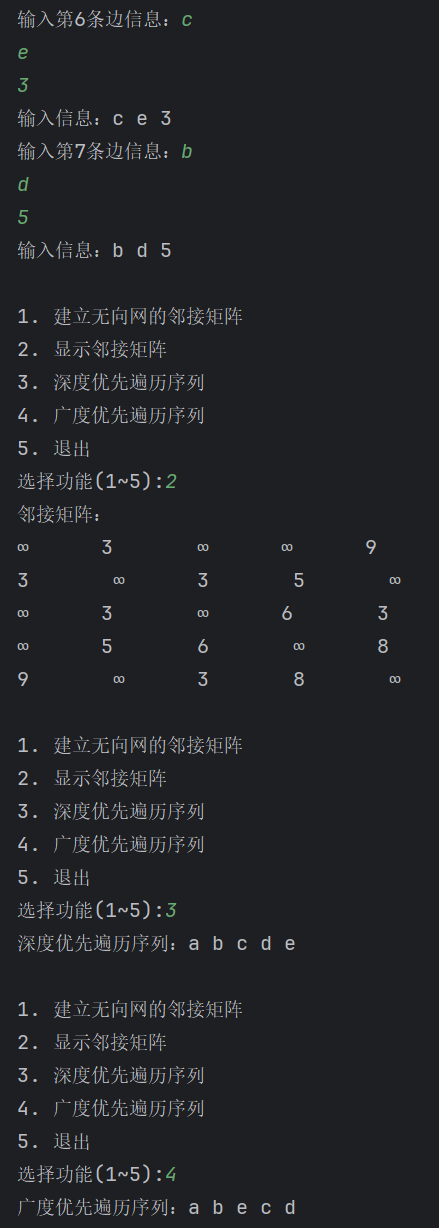
2、

#include <iostream>  
#include <vector>  
#include <stack>  
using namespace std;  
  
#define MAX\_VERTEX\_NUM 20  
  
typedef char VertexType;  
typedef struct {  
 VertexType vexs[MAX\_VERTEX\_NUM];  
 int arcs[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM];  
 int vexnum, arcnum;  
} MGraph;  
  
bool visited[MAX\_VERTEX\_NUM];  
  
// 在图G的顶点数组中查找顶点V，返回顶点的下标  
int LocateVex(MGraph G, VertexType v) {  
 for (int i = 0; i < G.vexnum; ++i) {  
 if (G.vexs[i] == v) {  
 return i;  
 }  
 }  
 return -1;  
}  
  
// 采用邻接矩阵表示法，构造有向图G  
bool CreateDG(MGraph &G) {  
 cout << "输入顶点数和边数：";  
 cin >> G.vexnum >> G.arcnum;  
  
 cout << "输入顶点信息：";  
 for (int i = 0; i < G.vexnum; ++i) {  
 cin >> G.vexs[i];  
 }  
  
 for (int i = 0; i < G.vexnum; ++i) {  
 for (int j = 0; j < G.vexnum; ++j) {  
 G.arcs[i][j] = 0; // 初始化邻接矩阵  
 }  
 }  
  
 cout << "输入有向边的信息(vi, vj):";  
 for (int k = 0; k < G.arcnum; ++k) {  
 VertexType vi, vj;  
 cout<<"输入第"<<k+1<<"条边信息：";  
 cin >> vi >> vj;  
  
 int m = LocateVex(G, vi);  
 int n = LocateVex(G, vj);  
  
 if (m != -1 && n != -1) {  
 G.arcs[m][n] = 1; // 有向边，用1表示  
 } else {  
 cout << "顶点不存在，重新输入边的信息。" << endl;  
 k--;  
 }  
 }  
  
 return true;  
}  
  
// 拓扑排序  
void TopologicalSortDFS(MGraph G, int v, stack<int> &s) {  
 visited[v] = true;  
  
 for (int w = 0; w < G.vexnum; ++w) {  
 if (G.arcs[v][w] && !visited[w]) {  
 TopologicalSortDFS(G, w, s);  
 }  
 }  
  
 s.push(v);  
}  
  
// 对有向图G进行拓扑排序  
void TopologicalSort(MGraph G) {  
 stack<int> s;  
  
 for (int i = 0; i < G.vexnum; ++i) {  
 visited[i] = false;  
 }  
  
 for (int i = 0; i < G.vexnum; ++i) {  
 if (!visited[i]) {  
 TopologicalSortDFS(G, i, s);  
 }  
 }  
  
 cout << "拓扑排序序列：";  
 while (!s.empty()) {  
 cout << G.vexs[s.top()] << " ";  
 s.pop();  
 }  
  
 cout << endl;  
}  
  
int main(void) {  
 MGraph G;  
  
 if (CreateDG(G)) {  
 TopologicalSort(G);  
 }  
  
 return 0;  
}

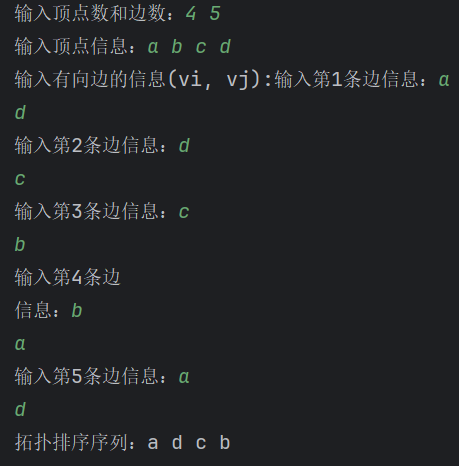
1. 结果截图

第一问：

1



2、



【实验体会】