石 家 庄 铁 道 大 学

**实 验 报 告**

**实习四 图的基本操作**

题 目： 图的基本操作

班 级： 信1901 – 4

姓 名： 闫竞存

学 号：20194127

日 期：2020.10.28

**1. 实验题目**

图的基本操作

[问题描述]

分别用邻接矩阵和邻接表实现以下操作：图的创建、遍历、插入、删除、最短路径。

[基本要求]

熟悉图的常用存储结构和基本操作。

[实现提示]

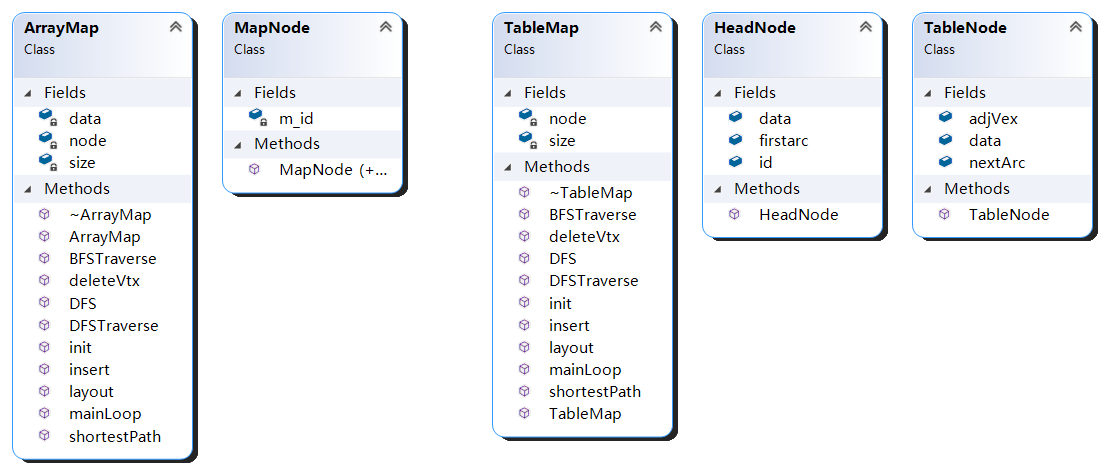
设图的结点不超过 30 个，每个结点用一个编号表示（如果一个图有 n 个结点，则它们的编号分别为 1,2,…,n）。通过输入图的全部边输入一个图，每个边为一个数对，可以对边的输入顺序作出某种限制。注意，生成树的边是有向边，端点顺序不能颠倒。

**2. 需求分析**

分别用邻接矩阵和临界表实现图的创建、遍历、插入和删除操作。

**3. 概要设计**

本程序的类图如下：

****

**4. 关键代码的详细设计**

这里以邻接表存储的为例

任意两点之间最短路径算法：

/\*

弗洛伊德算法求最短路径

\*/

void ArrayMap::shortestPath() {

    int dist[maxn][maxn] = {}; // 最短距离

    // 初始化最短距离矩阵

    for (int i = 0; i < this->size; i++) {

        for (int j = 0; j < this->size; j++) {

            if ((this->data[i][j] == 0 && i != j) || this->data[i][0] == -1) dist[i][j] = CANNOT\_REACH;

            // data[i][0] == -1 的时候，看作这个点不存在

            else dist[i][j] = this->data[i][j];

        }

    }

    // 打表,即为所求

    for (int k = 0; k < this->size; k++) {

        for (int i = 0; i < this->size; i++) {

            for (int j = 0; j < this->size; j++) {

                if (dist[i][j] > dist[i][k] + dist[k][j]) {

                    dist[i][j] = dist[i][k] + dist[k][j]; // 更新

                }

            }

        }

    }

    // 输入点两点

    cout << "输入两点, 判断其最短距离, 中间用空格隔开: ";

    int v1 = 0, v2 = 0;

    cin.clear();

    // 输入检查

    cin >> v1 >> v2;

    while (v1 < 1 || v1 > 30 || v2 < 1 || v2 > 30) {

        cout << "值不符合要求" << endl;

        cin >> v1 >> v2;

    }

    // 按照条件输出

    if (v1 == v2) {

        cout << "两点最短距离为 0 " << endl;

    } else if (dist[v1 - 1][v2 - 1] == 0 || dist[v1 - 1][v2 - 1] == CANNOT\_REACH) {

        cout << "两点不可达" << endl;

    } else {

        cout << "两点最短距离为 " << dist[v1 - 1][v2 - 1] << endl;

    }

}

深度优先搜索：

void ArrayMap::DFSTraverse() {

    cout << "------------\n深度优先遍历\n------------" << endl;

    // 已经遍历过的点

    int\* visit = new int[this->size];

    for (int i = 0; i < this->size; i++) {

        if (this->data[i][0] == -1) visit[i] = 1;

        else visit[i] = 0;

    }

    for (int i = 0; i < this->size; i++) {

        if (!visit[i])

            this->DFS(i, visit);

    }

    cout << endl;

}

void ArrayMap::DFS(int vtx, int visited[]) {

    cout << "访问节点--->" << vtx + 1 << endl;

    visited[vtx] = 1;

    for (int i = 0; i < this->size; i++) {

        if (this->data[vtx][i] != 0 && !visited[i])

            this->DFS(i, visited);

    }

}

广度优先搜索：

void ArrayMap::BFSTraverse() {

    cout << "------------\n广度优先遍历\n------------" << endl;

    // 已经遍历过的点

    int\* visit = new int[this->size];

    for (int i = 0; i < this->size; i++) {

        if (this->data[i][0] == -1) visit[i] = 1;

        else visit[i] = 0;

    }

    // 结点队列

    queue<int> q;

    for (int i = 0; i < this->size; i++) {

        if (!visit[i]) {

            visit[i] = 1;

            cout << "访问结点--->" << i + 1 << endl;

            q.push(i);

            while (!q.empty()) {

                i = q.front();

                q.pop();

                for (int j = 0; j < this->size; j++) {

                    if (this->data[i][j] != 0 && !visit[j]) {

                        visit[j] = 1;

                        cout << "访问结点--->" << j + 1 << endl;

                        q.push(j);

                    }

                }

            }

        }

    }

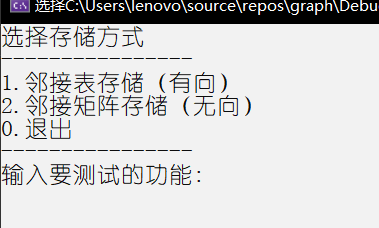
    cout << endl;

}

**5. 用户使用说明**

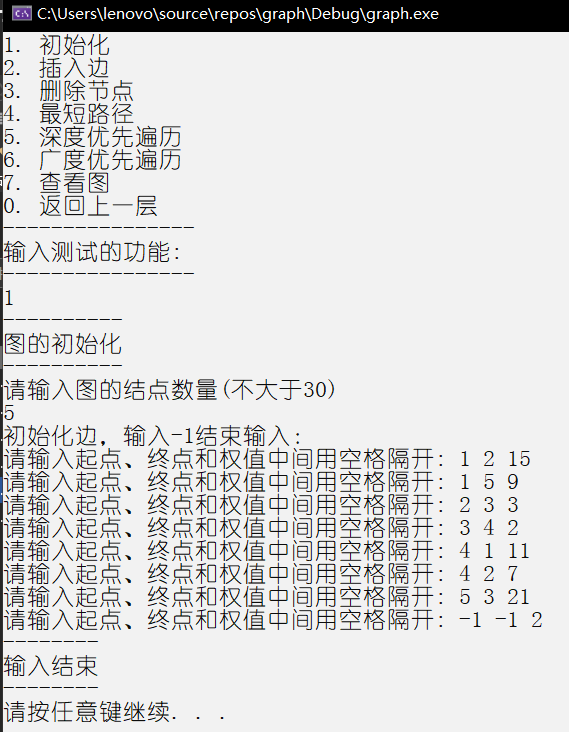
以邻接表存储为例

打开程序

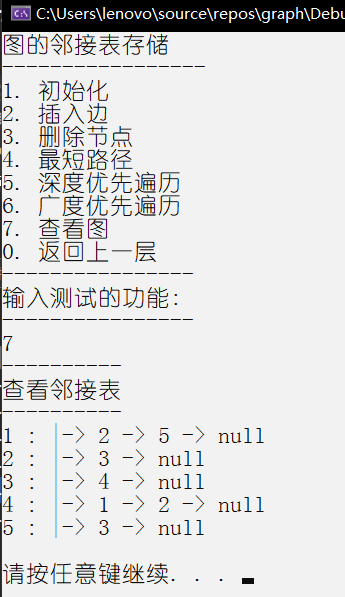


选择邻接表的存储方式：

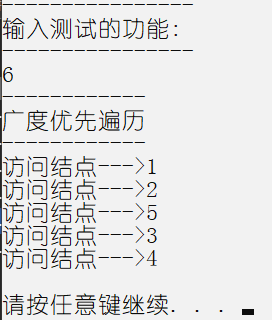
在菜单上选择相应的功能进行测试



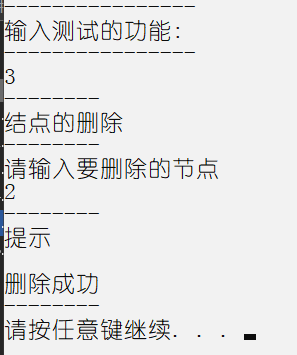
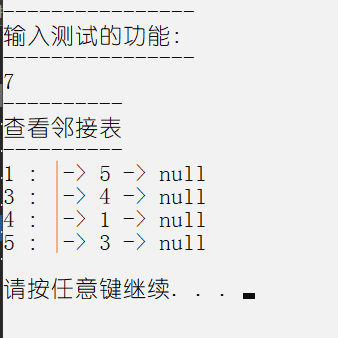
查看邻接表：



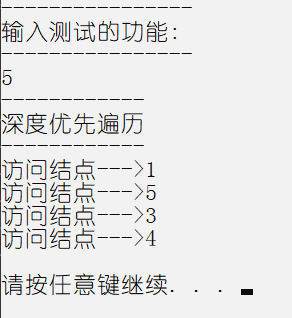
广度优先遍历



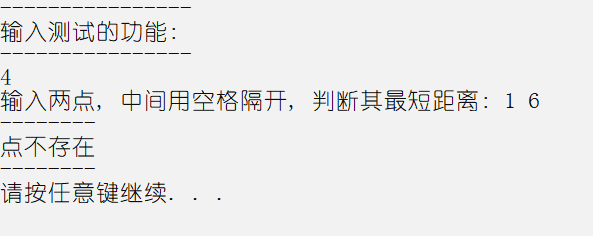
删除节点：

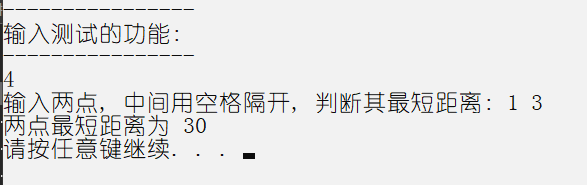
 

深度优先遍历：



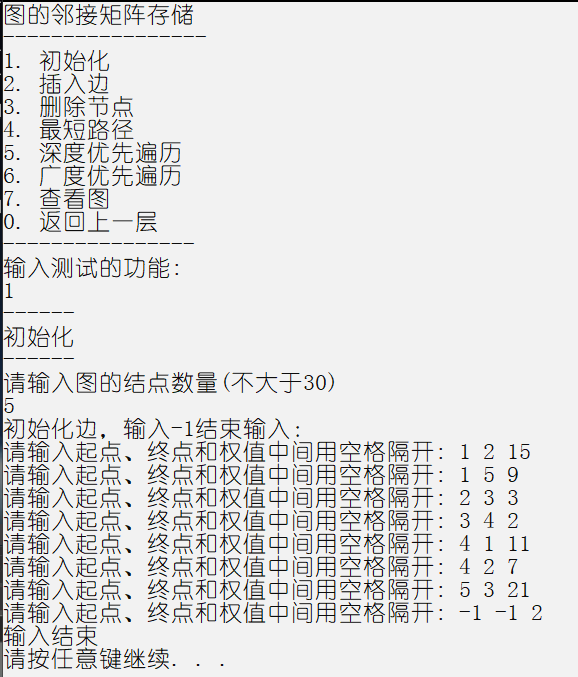
最短路径：



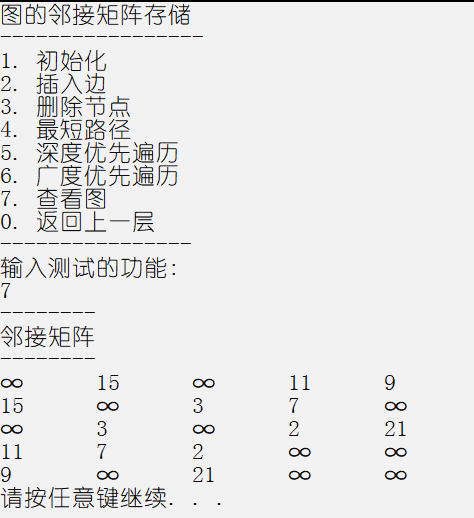


**6. 测试结果**

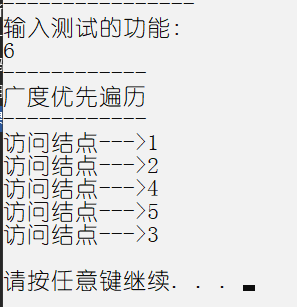
以邻接矩阵存储为例



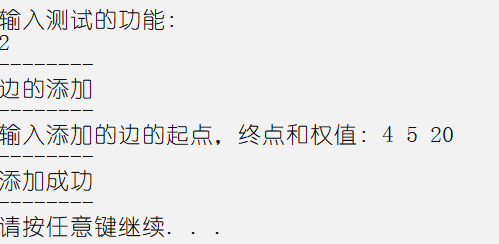
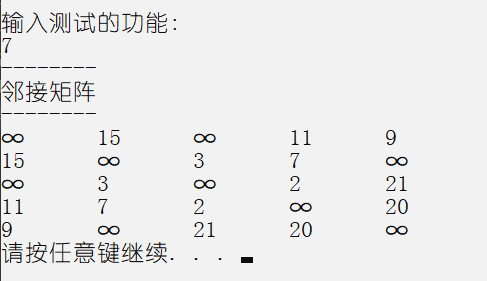
查看邻接矩阵：



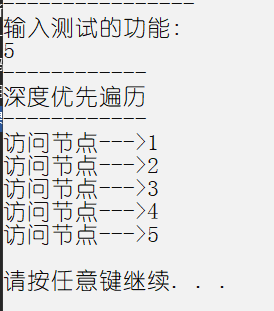
广度优先遍历



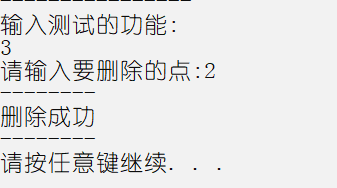
插入边:

深度优先遍历：



删除节点：



查看最短路径

