数据结构实验报告
张恒硕　　2212266　　智科 2 班

实验目标：　实现迪杰斯特拉算法

声明：代码是学习老师的 ppt 写的!

迪杰斯特拉算法

实验原理：

迪杰斯特拉算法（Dijkstra）是解决权图最短路径问题的一个有效算法。其特点是从起始点开始，采用贪心算法的策略，每次遍历到始点距离最近且未访问过的顶点的邻接节点，直到扩展到终点为止。算法流程初始时，S 只包含起点 s；U 包含除 s 外的其他顶点，且 U 中顶点的距离为起点 s 到该顶点的距离。从 U 中选出距离最短的顶点 k，并将顶点 k 加入到 S 中；同时，从 U 中移除顶点 k。更新 U 中各个顶点到起点 s 的距离。重复上述步骤，直到遍历完所有顶点。

代码：

```
#include <iostream>
#include <climits>
using namespace std;
void dijkstra(int** graph, int n, int root) { //迪杰斯特拉算法
    int* distance = new int[n]; //距离数组
    bool* visited = new bool[n]; //访问数组
    int* path = new int[n]; //路径
    for (int i = 0; i < n; i++) { //初始化
        distance[i] = graph[root][i]; //距离数组初始化为起点到各点权重
        visited[i] = false; //访问数组初始化为未访问
        path[i] = 0; //路径初始化为空
        if (distance[i] < INT_MAX) { //与起点相邻的各点和起点本身，路径起点为起点
            path[i] = root;
        }
    }
    visited[root] = true; //起点标记为已访问
    distance[root] = 0; //起点距离设为0
    for (int i = 0; i < n - 1; i++) { //循环（n-1）次
        int min_distance = INT_MAX; //找距离最小的顶点
        int v = 0; //距离最小顶点标记
        for (int j = 0; j < n; j++) {
            if (!visited[j] && distance[j] < min_distance) {
                min_distance = distance[j];
                v = j;
            }
        }
```

```
            visited[v] = true; //标记距离最小顶点为已访问
            for (int j = 0; j < n; j++) { //更新相邻顶点的距离
                if (!visited[j] && min_distance < INT_MAX && graph[v][j] < INT_MAX &&
min_distance + graph[v][j] < distance[j]) {
                    //两个限制条件防越限
                    distance[j] = min_distance + graph[v][j];
                    path[j] = v;
                }
            }
        }
        cout << "从起点" << root << "到各顶点的最短路径和距离为:" << endl;
//打印从源点到所有顶点的最短距离
        for (int i = 0; i < n; i++) {
            cout << "顶点" << i << ": ";
            if (i != root) {
                cout << "逆路径: " << i;
                int tmp = path[i];
                while (tmp != root) {
                    cout << tmp;
                    tmp = path[tmp];
                }
                cout << root << ", 距离为: " << distance[i] << endl;
            }
            else {
                cout << "为起点! " << endl;
            }
        }
        delete[] visited;
        delete[] distance;
}
int main() {
        int n, m;
        cout << "请输入顶点数和边数:";
        cin >> n >> m;
        int** graph = new int* [n]; //创建以邻接矩阵表示的图
        for (int i = 0; i < n; i++) {
            graph[i] = new int[n];
            for (int j = 0; j < n; j++) { //初始化邻接矩阵
                if (i == j) {
                    graph[i][j] = 0; //0表示自身
                }
                else {
                    graph[i][j] = INT_MAX; //极大值表示不相邻
                }
```

```
            }
        }
        cout << "请输入边和权重（注意，序号从0开排）:" << endl;
        for (int i = 0; i < m; i++) {
            int u, v, weight;
            cin >> u >> v >> weight;
            graph[u][v] = weight;
        }
        int root;
        cout << "请输入起始顶点:";
        cin >> root;
        dijkstra(graph, n, root);
        for (int i = 0; i < n; i++) { //释放内存
            delete[] graph[i];
        }
        delete[] graph;
        return 0;
}
```

输入样例:

1、

3 3
0 1 2
0 2 4
1 2 1
0

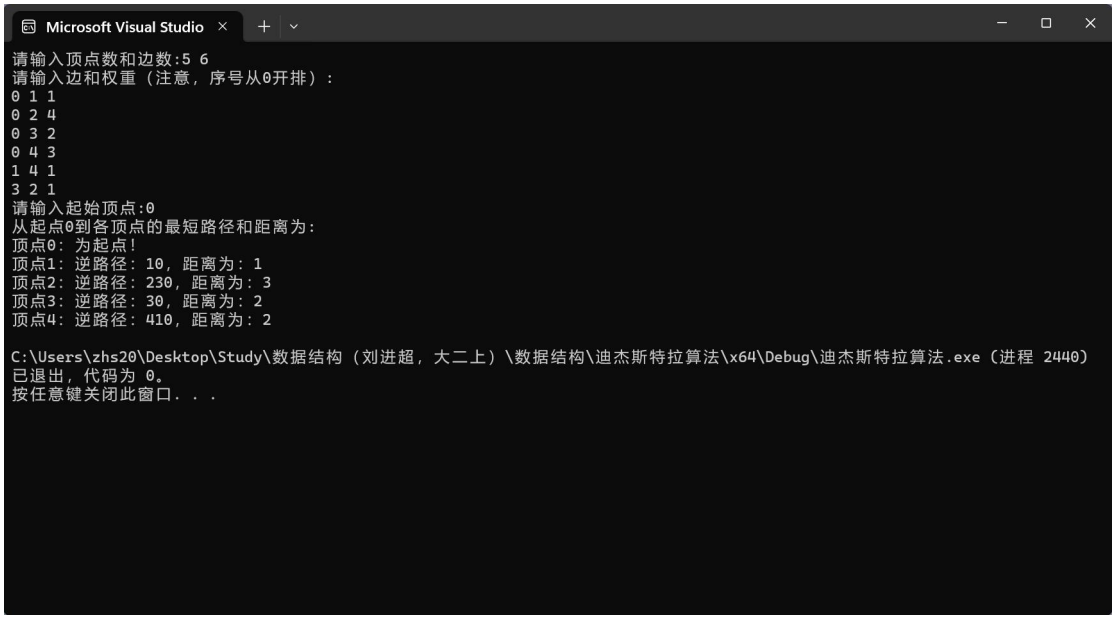
2、

5 6
0 1 1
0 2 4
0 3 2
0 4 3
1 4 1
3 2 1
0

运行结果:

1、

2、



实现操作

迪杰斯特拉算法（能给出逆路径和距离值）

分析

迪杰斯特拉算法适用于所有边权值为正的情况，能求出图中从源点到其余各顶点的最短路径。以|V|表示顶点数，在最坏情况下，时间复杂度为 O(|V|^2)， 空间复杂度为 O(|V|)。在边权值为负的情况下，可以使用贝尔曼-福特算法（Bellman-Ford algorithm）来求解最短路径问题。

在本代码中，利用 path 数组储存顶点所连接的上一个点的序号，这样可以按图索骥，寻回逆的路径。