哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础课（必修）

实验项目：图型结构及其应用

实验题目：**最短路径算法**

实验日期：2023/10/17

班级：2203202

学号：2022112266

姓名：魏圣卓

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 张岩 |

**一、实验目的**

最短路径问题研究的主要有：单源最短路径问题和所有顶点对之间的最短路径问题。在计算机领域和实际工程中具有广泛的应用，如集成电路设计、 GPS/游戏地图导航、智能交通、路由选择、铺设管线等。本实验要求设计和实现 Dijkstra算法和 Floyd-Warshall 算法，求解最短路径问题。

**二、实验要求及实验环境**

1． 实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长度和最短路径。

2． 实现全局最短路径的 Floyd-Warshall算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径。

3． 利用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每个顶点 *v* 到某个指定顶点 *c* 最短路径。

4． 利用 Dijkstra 或 Floyd-Warshall算法解决单顶点对间最短路径问题：对于某对顶点 *u* 和 *v*， 找出 *u* 到 *v* 和 *v* 到 *u* 的一条最短路径。

5． 以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点不少于 10 个，边不少于 13 个。

6． （选做）实现 Warshall 算法，计算有向图的可达矩阵，理解可达矩阵的含义；

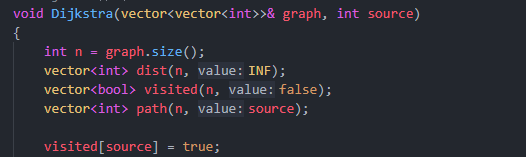
7． （选做）利用堆结构（优先级队列）改进和优化 Dijkstra 算法，实现改进和优化的 Dijkstra 算法，并与原算法进行实验比较；

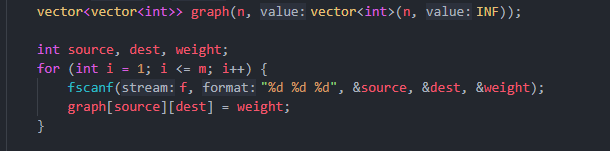
**三、设计思想**（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1．逻辑设计

2．物理设计（即存储结构设计）

使用邻接矩阵存储无向图





**四、测试结果**（包括**测试数据、结果数据及结果的简单分析和结论，可以用截图得形式贴入此报告**）

测试用图：11 14

1 7 1

1 8 2

2 4 3

2 8 8

3 1 5

3 2 1

3 10 4

5 6 4

5 8 1

7 5 8

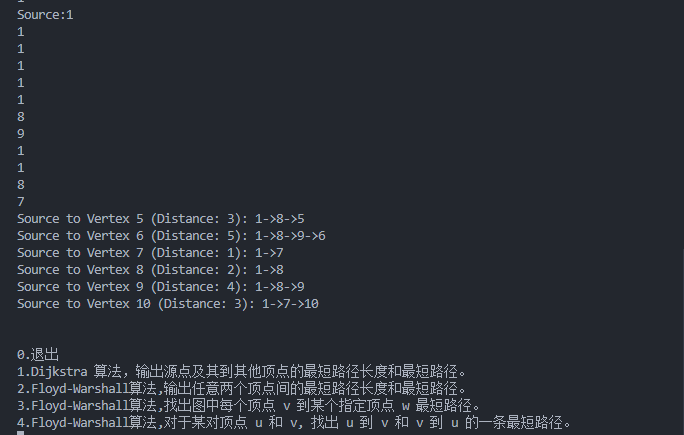
7 10 2

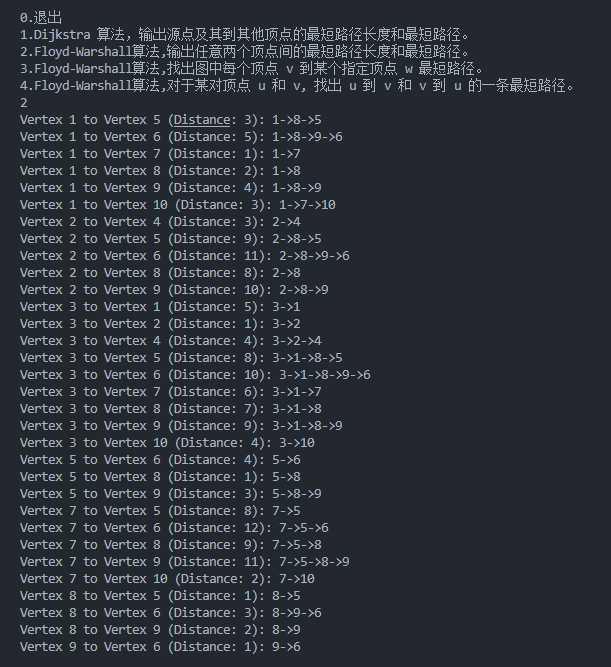
8 5 1

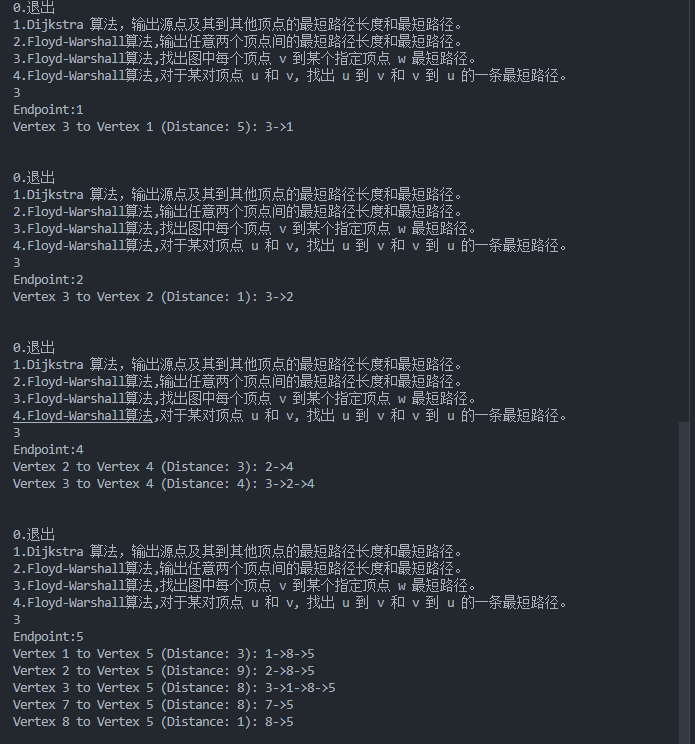
8 9 2

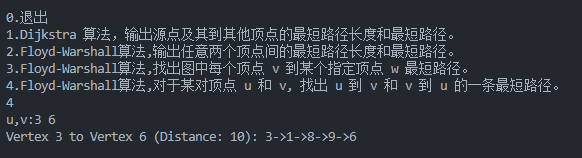
9 6 1

10 4 2









**五、经验体会与不足**

在本次实验中，我学会了图的存储方法以及不同的求最短路的相关算法。后续可以加强学习对于有向图的处理，同时深入理解可达矩阵的含义。

**六、附录：源代码（带注释）**

#include <fstream>

#include <iostream>

#include <limits>

#include <math.h>

#include <stack>

#include <vector>

using namespace std;

const int INF = numeric\_limits<int>::max(); // 代表无穷大

// const int INF = INFINITY;

// Dijkstra算法

void Dijkstra(vector<vector<int>>& graph, int source)

{

    int n = graph.size();

    vector<int> dist(n, INF);

    vector<bool> visited(n, false);

    vector<int> path(n, source);

    visited[source] = true;

    for (int i = 0; i < graph.size(); i++) {

        dist[i] = graph[source][i];

    }

    for (int i = 0; i < n - 1; i++) {

        int minDist = INF, minIndex = -1;

        // 找到距离最近且未被访问的顶点

        for (int j = 0; j < n; j++) {

            if (!visited[j] && dist[j] < minDist) {

                minDist = dist[j];

                minIndex = j;

            }

        }

        if (minIndex == -1)

            break;

        visited[minIndex] = true;

        // 更新相邻顶点的距离

        for (int j = 0; j < n; j++) {

            if (!visited[j] && graph[minIndex][j] != INF && dist[minIndex] != INF && dist[minIndex] + graph[minIndex][j] < dist[j]) {

                path[j] = minIndex;

                dist[j] = dist[minIndex] + graph[minIndex][j];

            }

        }

    }

    for (int i = 0; i < n; i++) {

        cout << path[i] << endl;

    }

    // 输出最短路径长度和路径

    stack<int> finalpath;

    int temp;

    for (int i = 0; i < n; i++) {

        if (i == source) {

            continue;

        }

        if (dist[i] == INF) {

            continue;

        }

        temp = i;

        finalpath.push(temp);

        cout << "Source to Vertex " << i << " (Distance: " << dist[i] << "): ";

        while (1) {

            temp = path[temp];

            finalpath.push(temp);

            if (temp == source) {

                // finalpath.push(source);

                break;

            }

        }

        while (1) {

            cout << finalpath.top();

            finalpath.pop();

            if (finalpath.empty()) {

                break;

            }

            cout << "->";

        }

        cout << endl;

    }

}

void search(int s, int e, vector<vector<int>> path, vector<vector<int>> graph, vector<vector<int>> dist)

{

    int k = path[s][e];

    if (graph[s][e] == dist[s][e]) {

        cout << "->" << e;

        return;

    }

    search(s, k, path, graph, dist);

    search(k, e, path, graph, dist);

}

// Floyd-Warshall算法

void FloydWarshall(vector<vector<int>>& graph)

{

    int n = graph.size();

    vector<vector<int>> dist = graph;

    vector<vector<int>> path(n, vector<int>(n, -1));

    // 计算最短路径

    for (int k = 0; k < n; k++) {

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

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

                    path[i][j] = k;

                }

            }

        }

    }

    for (int i = 0; i < n; i++) {

        dist[i][i] = 0;

    }

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < n; j++) {

            if (i == j) {

                continue;

            }

            if (dist[i][j] == INF) {

                continue;

            }

            cout << "Vertex " << i << " to Vertex " << j << " (Distance: " << dist[i][j] << "): ";

            cout << i;

            search(i, j, path, graph, dist);

            cout << endl;

        }

    }

    //     for (int i=0; i<n; i++) {

    //         for (int j=0; j<n; j++) {

    //             cout<<dist[i][j]<<"  ";

    //         }

    //         cout<<endl;

    //     }

    // return;

    // 输出最短路径长度和路径

    // for (int i = 0; i < n; i++) {

    //     for (int j = 0; j < n; j++) {

    //     if (i==j) {

    //     continue;

    //     }

    //         cout << "Vertex " << i << " to Vertex " << j << " (Distance: " << dist[i][j] << "): ";

    //         vector<int> path;

    //         int v = i;

    //         while (v != j) {

    //             for (int k = 0; k < n; k++) {

    //                 if (graph[v][k] + dist[k][j] == dist[v][j]) {

    //                     path.push\_back(k);

    //                     v = k;

    //                     break;

    //                 }

    //             }

    //         }

    //         cout << i;

    //         for (int p : path)

    //             cout << " -> " << p;

    //         cout << endl;

    //         // cout << " -> " << j << endl;

    //     }

    // }

}

void FloydWarshall(vector<vector<int>>& graph, int end)

{

    int n = graph.size();

    vector<vector<int>> dist = graph;

    vector<vector<int>> path(n, vector<int>(n, -1));

    // 计算最短路径

    for (int k = 0; k < n; k++) {

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

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

                    path[i][j] = k;

                }

            }

        }

    }

    for (int i = 0; i < n; i++) {

        dist[i][i] = 0;

    }

    for (int i = 0; i < n; i++) {

        if (i == end) {

            continue;

        }

        if (dist[i][end] == INF) {

            continue;

        }

        cout << "Vertex " << i << " to Vertex " << end << " (Distance: " << dist[i][end] << "): ";

        cout << i;

        search(i, end, path, graph, dist);

        cout << endl;

    }

}

void FloydWarshall(vector<vector<int>>& graph, int u, int v)

{

    if (u == v) {

        return;

    }

    int n = graph.size();

    vector<vector<int>> dist = graph;

    vector<vector<int>> path(n, vector<int>(n, -1));

    // 计算最短路径

    for (int k = 0; k < n; k++) {

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n; j++) {

                if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] < dist[i][j]) {

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

                    path[i][j] = k;

                }

            }

        }

    }

    for (int i = 0; i < n; i++) {

        dist[i][i] = 0;

    }

    if (dist[u][v] != INF) {

        cout << "Vertex " << u << " to Vertex " << v << " (Distance: " << dist[u][v] << "): ";

        cout << u;

        search(u, v, path, graph, dist);

        cout << endl;

    }

    if (dist[v][u] != INF) {

        cout << "Vertex " << v << " to Vertex " << u << " (Distance: " << dist[v][u] << "): ";

        cout << v;

        search(v, u, path, graph, dist);

        cout << endl;

    }

}

int main()

{

    int n, m;

    FILE\* f = fopen("graph.txt", "r");

    // FILE\* f = fopen("graph copy.txt", "r");

    fscanf(f, "%d %d", &n, &m);

    vector<vector<int>> graph(n, vector<int>(n, INF));

    int source, dest, weight;

    for (int i = 1; i <= m; i++) {

        fscanf(f, "%d %d %d", &source, &dest, &weight);

        graph[source][dest] = weight;

    }

    // cout << "Enter the number of vertices: ";

    // cin >> n;

    // cout << "Enter the number of edges: ";

    // cin >> m;

    // cout << "Enter edge information (source, destination, weight):" << endl;

    // for (int i = 0; i < m; i++) {

    //     int source, dest, weight;

    //     cin >> source >> dest >> weight;

    //     graph[source][dest] = weight;

    // }

    while (1) {

        printf("0.退出\n");

        printf("1.Dijkstra 算法，输出源点及其到其他顶点的最短路径长度和最短路径。\n");

        printf("2.Floyd-Warshall算法,输出任意两个顶点间的最短路径长度和最短路径。\n");

        printf("3.Floyd-Warshall算法,找出图中每个顶点 v 到某个指定顶点 w 最短路径。\n");

        printf("4.Floyd-Warshall算法,对于某对顶点 u 和 v, 找出 u 到 v 和 v 到 u 的一条最短路径。\n");

        char c;

        scanf("%d", &c);

        // cin>>c;

        if (c == 0)

            break;

        switch (c) {

        case 1:

            cout << "Source:";

            int source;

            cin >> source;

            Dijkstra(graph, source);

            break;

        case 2:

            FloydWarshall(graph);

            break;

        case 3:

            int end;

            cout << "Endpoint:";

            cin >> end;

            FloydWarshall(graph, end);

            break;

        case 4:

            int u, v;

            cout << "u,v:";

            cin >> u >> v;

            FloydWarshall(graph, u, v);

        }

        printf("\n\n");

    }

    // // Dijkstra算法

    // int sourceDijkstra;

    // cout << "Enter the source vertex for Dijkstra: ";

    // cin >> sourceDijkstra;

    // Dijkstra(graph, sourceDijkstra);

    // // Floyd-Warshall算法

    // FloydWarshall(graph);

    return 0;

    // 5 8 0 3 30 0 1 10 1 2 50 2 4 10 2 3 20 3 2 20 3 4 60 0 4 100

}