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：专业基础（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：2022.11.4

班级：2103103

学号：2021110608

姓名：孟悦琦

|  |  |  |
| --- | --- | --- |
| 设计成绩 | 报告成绩 | 指导老师 |
|  |  | 翟德明 |

**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）

本程序主要包含main函数，负责调用主要函数；Graph.h，Graph类的各种声明。Graph.cpp，Graph中的函数的实现。

下面主要介绍一下Graph类中包含的数据。

private:int edge[V\_NUMBER][V\_NUMBER] = {0}; // 图的邻接矩阵

// dijkstra算法的辅助记录

int visit[V\_NUMBER] = {0}; // 是否访问过的记录数组

int distance[V\_NUMBER] = {0}; // dijkstra算法最短路径辅助数组

int pass[V\_NUMBER] = {0}; // dijkstra算法路径辅助数组

// Floyd\_Warshall算法的辅助记录

int P[V\_NUMBER][V\_NUMBER]; // Floyd\_Warshall算法路径辅助数组

int D[V\_NUMBER][V\_NUMBER]; // Floyd\_Warshall算法最短路径辅助数组

int dijkstra\_chooseMin(int i); // dijkstra找到最小边的方法

void dijkstra\_print\_path(int i, int j); // dijkstra打印路径的算法

void Floyd\_Warshall\_path(int i,int j); // 打印Floyd\_Warshall算法计算出的路径的算法

void Print\_FW\_Single(int i); // 只打印关于一个节点的算法

void Floyd\_Warshall\_core(); // Floyd\_Warshall的核心算法

public:Graph(const char \*name = nullptr); // 初始化一个图

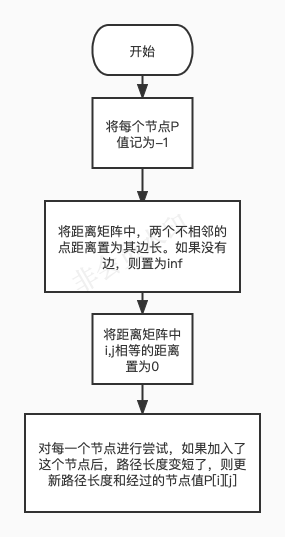
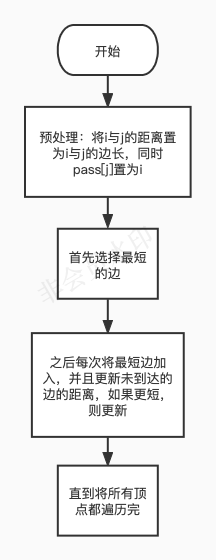
void printGraph(); // 打印一个图

void dijkstra(int i); // 对某个顶点展开dijkstra算法

void Floyd\_Warshall(); // 调用普遍意义上的Floyd\_Warshall

void find\_single(int i); // 关于某个顶点，找到到这个顶点的所有顶点的最短路

void find\_pair(int i,int j); // 找到两个顶点间来去的最短路



两个核心算法的流程图。（左侧是dijkstra算法，右侧是Floyd\_Warshall算法）

**四、测试结果**

0 5 2 4 5 0 0 0 7 0

0 0 3 0 0 0 0 0 0 6

0 0 0 1 0 0 0 0 0 0

0 0 0 0 7 0 0 0 0 0

0 0 6 0 0 0 9 0 0 0

0 0 0 0 9 0 0 0 0 0

0 0 0 0 0 8 0 0 0 0

0 0 0 0 0 0 0 0 0 10

0 0 0 0 0 0 9 0 0 0

0 0 0 0 0 0 0 0 0 0

The algorithm Floyd\_Warshall:

shortest distance: 5 path: 0 1

shortest distance: 2 path: 0 2

shortest distance: 3 path: 0 2 3

shortest distance: 5 path: 0 4

shortest distance: 22 path: 0 4 6 5

shortest distance: 14 path: 0 4 6

The path 0 to 7 :Not Find!

shortest distance: 7 path: 0 8

shortest distance: 11 path: 0 1 9

The algorithm Floyd\_Warshall:

The distance 0 to 1 is: 5 The path is:0 1

The distance 0 to 2 is: 2 The path is:0 2

The distance 0 to 3 is: 3 The path is:0 2 3

The distance 0 to 4 is: 5 The path is:0 4

The distance 0 to 5 is: 22 The path is:0 4 6 5

The distance 0 to 6 is: 14 The path is:0 4 6

The distance 0 to 7 is: not found!

The distance 0 to 8 is: 7 The path is:0 8

The distance 0 to 9 is: 11 The path is:0 1 9

The distance 1 to 0 is: not found!

The distance 1 to 2 is: 3 The path is:1 2

The distance 1 to 3 is: 4 The path is:1 2 3

The distance 1 to 4 is: 11 The path is:1 2 3 4

The distance 1 to 5 is: 28 The path is:1 2 3 4 6 5

The distance 1 to 6 is: 20 The path is:1 2 3 4 6

The distance 1 to 7 is: not found!

The distance 1 to 8 is: not found!

The distance 1 to 9 is: 6 The path is:1 9

The distance 2 to 0 is: not found!

The distance 2 to 1 is: not found!

The distance 2 to 3 is: 1 The path is:2 3

The distance 2 to 4 is: 8 The path is:2 3 4

The distance 2 to 5 is: 25 The path is:2 3 4 6 5

The distance 2 to 6 is: 17 The path is:2 3 4 6

The distance 2 to 7 is: not found!

The distance 2 to 8 is: not found!

The distance 2 to 9 is: not found!

The distance 3 to 0 is: not found!

The distance 3 to 1 is: not found!

The distance 3 to 2 is: 13 The path is:3 4 2

The distance 3 to 4 is: 7 The path is:3 4

The distance 3 to 5 is: 24 The path is:3 4 6 5

The distance 3 to 6 is: 16 The path is:3 4 6

The distance 3 to 7 is: not found!

The distance 3 to 8 is: not found!

The distance 3 to 9 is: not found!

The distance 4 to 0 is: not found!

The distance 4 to 1 is: not found!

The distance 4 to 2 is: 6 The path is:4 2

The distance 4 to 3 is: 7 The path is:4 2 3

The distance 4 to 5 is: 17 The path is:4 6 5

The distance 4 to 6 is: 9 The path is:4 6

The distance 4 to 7 is: not found!

The distance 4 to 8 is: not found!

The distance 4 to 9 is: not found!

The distance 5 to 0 is: not found!

The distance 5 to 1 is: not found!

The distance 5 to 2 is: 15 The path is:5 4 2

The distance 5 to 3 is: 16 The path is:5 4 2 3

The distance 5 to 4 is: 9 The path is:5 4

The distance 5 to 6 is: 18 The path is:5 4 6

The distance 5 to 7 is: not found!

The distance 5 to 8 is: not found!

The distance 5 to 9 is: not found!

The distance 6 to 0 is: not found!

The distance 6 to 1 is: not found!

The distance 6 to 2 is: 23 The path is:6 5 4 2

The distance 6 to 3 is: 24 The path is:6 5 4 2 3

The distance 6 to 4 is: 17 The path is:6 5 4

The distance 6 to 5 is: 8 The path is:6 5

The distance 6 to 7 is: not found!

The distance 6 to 8 is: not found!

The distance 6 to 9 is: not found!

The distance 7 to 0 is: not found!

The distance 7 to 1 is: not found!

The distance 7 to 2 is: not found!

The distance 7 to 3 is: not found!

The distance 7 to 4 is: not found!

The distance 7 to 5 is: not found!

The distance 7 to 6 is: not found!

The distance 7 to 8 is: not found!

The distance 7 to 9 is: 10 The path is:7 9

The distance 8 to 0 is: not found!

The distance 8 to 1 is: not found!

The distance 8 to 2 is: 32 The path is:8 6 5 4 2

The distance 8 to 3 is: 33 The path is:8 6 5 4 2 3

The distance 8 to 4 is: 26 The path is:8 6 5 4

The distance 8 to 5 is: 17 The path is:8 6 5

The distance 8 to 6 is: 9 The path is:8 6

The distance 8 to 7 is: not found!

The distance 8 to 9 is: not found!

The distance 9 to 0 is: not found!

The distance 9 to 1 is: not found!

The distance 9 to 2 is: not found!

The distance 9 to 3 is: not found!

The distance 9 to 4 is: not found!

The distance 9 to 5 is: not found!

The distance 9 to 6 is: not found!

The distance 9 to 7 is: not found!

The distance 9 to 8 is: not found!

The distance 2 to 0 is: not found!

The distance 2 to 1 is: not found!

The distance 2 to 3 is: 1 The path is:2 3

The distance 2 to 4 is: 8 The path is:2 3 4

The distance 2 to 5 is: 25 The path is:2 3 4 6 5

The distance 2 to 6 is: 17 The path is:2 3 4 6

The distance 2 to 7 is: not found!

The distance 2 to 8 is: not found!

The distance 2 to 9 is: not found!

The distance 2 to 3 is: 1 The path is:2 3

The distance 3 to 2 is: 13 The path is:3 4 2

**五、经验体会与不足**

本次实验让我更好理解了图中最短路径的算法，同时对寻找最短路的算法思想有了一些认识。

在代码的编写中还有些瑕疵，之后会修改。

**六、附录：源代码（带注释）**

main.c

#include <iostream>

#include "Graph.h"

int main() {

Graph G = Graph("/Users/mengfanxing/Desktop/DS\_lab3/source.txt");

G.printGraph();

std::cout << "The algorithm Floyd\_Warshall:" << std::endl;

G.dijkstra(0);

std::cout << std::endl;

std::cout << "The algorithm Floyd\_Warshall:" << std::endl;

G.Floyd\_Warshall();

std::cout << std::endl;

G.find\_single(2);

std::cout << std::endl;

G.find\_pair(2,3);

return 0;

}

Graph.cpp

//

// Created by 孟繁兴 on 2022/10/28.

//

#include "Graph.h"

// 构造一个图

Graph::Graph(const char \*name) {

// 创建读取文件流

std::ifstream ifs;

ifs.open(name,std::ios::in);

// 读取文件并初始化图

int i,j,cost;

i = j = cost = 0;

if (ifs.is\_open()) {

while (!ifs.eof()) {

ifs >> i >> j >> cost;

edge[i][j] = cost;

}

ifs.close();

} else {

std::cerr << "Can't open the file!" << std::endl;

}

}

// 打印一个图

void Graph::printGraph() {

for (int i = 0; i < V\_NUMBER; ++i) {

for (int j = 0; j < V\_NUMBER; ++j) {

std::cout << edge[i][j] << "\t";

}

std::cout << std::endl;

}

}

// dijkstra算法，找到目前距离最短的节点

int Graph::dijkstra\_chooseMin(int i) {

int min = i;

for (int j = 0; j < V\_NUMBER; ++j) {

if (visit[j] == 0) {

if (min == i) {

min = j;

} else {

min = distance[min] < distance[j] ? min : j;

}

}

}

return min;

}

// 根据dijkstra算法的结结果，打印两个顶点之间的路

void Graph::dijkstra\_print\_path(int i, int j) {

// 辅助栈s

std::stack<int> s;

int k = j;

// 遍历获取路径

while (k!=i) {

s.push(k);

k = pass[k];

}

// 打印路径的算法

std::cout << "shortest distance: " << distance[j] << "\t";

std::cout << "path: " << i << "\t";

while (!s.empty()) {

std::cout << s.top() << "\t";

s.pop();

}

std::cout << std::endl;

}

// 运用dijkstra算法，找到从某一顶点开始，到其他顶点的最短路径

void Graph::dijkstra(int i) {

// 算法预处理

for (int j = 0; j < V\_NUMBER; ++j) {

if (edge[i][j] && i!=j) {

distance[j] = edge[i][j];

} else if (i!=j) {

distance[j] = inf;

}

}

for (int j = 0; j < V\_NUMBER; ++j) {

pass[j] = i;

}

visit[i] = 1;

for (int j = 0; j < V\_NUMBER; ++j) {

int v = dijkstra\_chooseMin(i);

visit[v] = 1;

for (int k = 0; k < V\_NUMBER; ++k) {

// 更新最短距离

if (visit[k] == 0 && edge[v][k]) {

// 如果加入新的边使之前的距离更短，就更新距离和走过的边

if (distance[k] > distance[v]+edge[v][k]) {

distance[k] = distance[v]+edge[v][k];

pass[k] = v;

}

}

}

}

// 打印路径

for (int j = 0; j < V\_NUMBER; ++j) {

if (j!=i && distance[j] != inf) {

dijkstra\_print\_path(i, j);

} else if (j!=i){

std::cout << "The path " << i << " to " << j << " :Not Find!" << std::endl;

}

}

}

// 运用Floyd\_Warshall打印路径的算法

void Graph::Floyd\_Warshall\_path(int i, int j) {

int k = P[i][j];

if (k!=-1) {

Floyd\_Warshall\_path(i,k);

std::cout << k << "\t";

Floyd\_Warshall\_path(k,j);

}

}

// 运用Floyd\_Warshall算法寻找最短路

void Graph::Floyd\_Warshall() {

// 调用核心算法

Floyd\_Warshall\_core();

// 打印路径

for (int i = 0; i < V\_NUMBER; ++i) {

for (int j = 0; j < V\_NUMBER; ++j) {

if (i!=j && D[i][j] != inf) {

std::cout << "The distance " << i << " to " << j << " is: " << D[i][j] << "\t";

std::cout << "\t" << "The path is:" << i << "\t";

Floyd\_Warshall\_path(i, j);

std::cout << j << std::endl;

} else if (i!=j && D[i][j] == inf) {

std::cout << "The distance " << i << " to " << j << " is: not found!" << std::endl;

}

}

}

}

//运用Floyd\_Warshall算法计算每个顶点到某个顶点i的最短路径，需要提前调用Floyd\_Warshall算法

void Graph::Print\_FW\_Single(int i) {

for (int j = 0; j < V\_NUMBER; ++j) {

if (i!=j && D[i][j] != inf) {

std::cout << "The distance " << i << " to " << j << "\t" << " is: " << D[i][j] << "\t";

std::cout << "\t" << "The path is:" << i << "\t";

Floyd\_Warshall\_path(i, j);

std::cout << j << std::endl;

} else if (i!=j && D[i][j] == inf) {

std::cout << "The distance " << i << " to " << j << " is: not found!" << std::endl;

}

}

}

// 运用Floyd\_Warshall算法，计算各个顶点‍到达某个点的最短路径

void Graph::find\_single(int i) {

// 调用核心算法

Floyd\_Warshall\_core();

Print\_FW\_Single(i);

}

// Floyd\_Warshall的核心算法

void Graph::Floyd\_Warshall\_core() {

// 算法预备工作

for (int i = 0; i < V\_NUMBER; ++i) {

for (int j = 0; j < V\_NUMBER; ++j) {

P[i][j] = -1;

if (i!=j) {

if (edge[i][j]==0){D[i][j] = inf;}

else {D[i][j] = edge[i][j];}

} else {

D[i][j] = 0;

}

}

}

// 算法实现工作

for (int k = 0; k < V\_NUMBER; ++k) { // 对每一个节点进行尝试

for (int i = 0; i < V\_NUMBER; ++i) {

for (int j = 0; j < V\_NUMBER; ++j) {

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

P[i][j] = k;

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

}

}

}

}

}

// 运用Floyd\_Warshall算法，找到一对顶点之间，相通的两条最短路径

void Graph::find\_pair(int i, int j) {

// 首先调用Floyd\_Warshall算法

Floyd\_Warshall\_core();

if (D[i][j]!=inf && D[j][i]!=inf) {

// 打印i到j的路径

std::cout << "The distance " << i << " to " << j << "\t" << " is: " << D[i][j] << "\t";

std::cout << "\t" << "The path is:" << i << "\t";

Floyd\_Warshall\_path(i,j);

std::cout << j << std::endl;

// 打印j到i的路径

std::cout << "The distance " << j << " to " << i << "\t" << " is: " << D[j][i] << "\t";

std::cout << "\t" << "The path is:" << j << "\t";

Floyd\_Warshall\_path(j,i);

std::cout << i << std::endl;

} else {

std::cout << "Do not find the path!" << std::endl;

}

}

Graph.h

//

// Created by 孟繁兴 on 2022/10/28.

//

#ifndef DS\_LAB3\_GRAPH\_H

#define DS\_LAB3\_GRAPH\_H

#define V\_NUMBER 10

#define inf 13904789

#include <iostream>

#include <fstream>

#include <stack>

class Graph {

private:int edge[V\_NUMBER][V\_NUMBER] = {0}; // 图的邻接矩阵

// dijkstra算法的辅助记录

int visit[V\_NUMBER] = {0}; // 是否访问过的记录数组

int distance[V\_NUMBER] = {0}; // dijkstra算法最短路径辅助数组

int pass[V\_NUMBER] = {0}; // dijkstra算法路径辅助数组

// Floyd\_Warshall算法的辅助记录

int P[V\_NUMBER][V\_NUMBER]; // Floyd\_Warshall算法路径辅助数组

int D[V\_NUMBER][V\_NUMBER]; // Floyd\_Warshall算法最短路径辅助数组

int dijkstra\_chooseMin(int i);

void dijkstra\_print\_path(int i, int j);

void Floyd\_Warshall\_path(int i,int j);

void Print\_FW\_Single(int i);

void Floyd\_Warshall\_core();

public:Graph(const char \*name = nullptr);

void printGraph();

void dijkstra(int i);

void Floyd\_Warshall();

void find\_single(int i);

void find\_pair(int i,int j);

};

#endif //DS\_LAB3\_GRAPH\_H

source.txt

0 1 5

0 2 2

0 3 4

0 4 5

1 2 3

2 3 1

3 4 7

4 2 6

7 9 10

8 6 9

6 5 8

1 9 6

5 4 9

0 8 7

4 6 9