

重庆交通大学信息科学与工程学院

综合性实验报告

实验名称	图算法应用实验
课程名称	数据结构 A
专业班级	计算机类 2105
学 号	632162010205
姓 名	周文浩
指导教师	鲁云平

2022 年 12 月

《数据结构 A》综合性实验评分标准

评分等级	综合性实验评分标准
优	程序演示完全正确，界面美观，能正确回答 90%及以上的问题；报告规范，分析清楚，严格按照要求条目书写，阐述清楚。能针对综合性题目进行分析，并设计合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
良	按要求完成 80%及以上功能，界面尚可，能正确回答 80%及以上的问题；报告规范，分析清楚，个别条目书写不完全符合要求，阐述基本清楚。能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
中	按要求完成 70%及以上功能，能回答 70%及以上的问题；报告基本规范，分析基本清楚，存在 30%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。
及格	按要求完成 60%及以上功能，能回答老师多数问题；报告基本规范，存在 40%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析和设计较合适的解决方案，并能使用合适的编程平台进行编程、调试、测试和实现，成功调试功能达 60%以上。
不及格	存在 40%以上功能未完成或抄袭；报告不规范或存在 40%以上条目书写不完全符合要求。无法采用正确的方法完成项目分析和解决方案设计或成功调试测试功能超过 40%未完成。

教师签名	
综合性实验成绩	

说明：

- 1、综合性实验报告总体要求为格式规范，内容丰富、全面、深入；
严禁大量拷贝。
- 2、请提交 word 文档和 PDF 文档，文件命名格式：学号-姓名。
- 3、报告主要内容参考下页示例

一、实验目的

根据所学内容，结合实际应用，使用邻接矩阵或邻接表实现图的存储结构，运用图的广度优先和深度优先遍历、最小生成树或最短路径等算法进行应用编程。

二、实验内容

a、最短路径实现校园地图查询

- 1、查询校园地点的信息
- 2、查询校园两点间的最短路径
- 3、查询校园任意两点的全部最短路径
- 4、更新校园间路径长度
- 5、在两个地点间添加路径
- 6、删除两个地点的一条路径
- 7、增加一个地点
- 8、去除一个地点

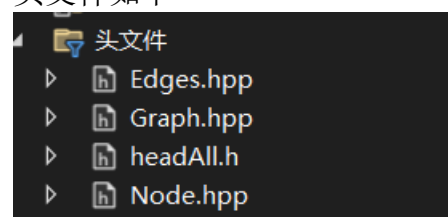
b、DFS 实现仓库最优取货路径（这个仓库系统是我参加物流设计大赛，这个十分契合本次大作业，因此我特此加了一个应用）

- 1、获得两点之间所有路径的数组
- 2、输出途径某些点两点之间的所有路径
- 3、输出两点之间的所有路径
- 4、输出两点之间途经点最少的路径

三、系统分析

（1）数据方面：

头文件如下



对于 Edges 这个类

准备定义起始点、终点、权重三个属性

对于 Graph 这个类

准备定义 记录结点数、边数、存放文件名、

判断是否为有向图的布尔值四个属性，加上邻接矩阵和顶点数组和边数组和标记数组

对于 headAll 这个头文件

准备放置各个类都需要导入的头文件

对于 Node 这个类

准备定义 顶点编号和顶点名称两个属性

（2）功能方面：

对于 Edges 这个类

准备定义 获取和设置起始点、终点、权重和一个有参构造函数

对于 Graph 这个类

公开属性：准备定义 读取文件、写文件、两个有参构造、一个析构函数、初始化、打印邻接矩阵、打印边数组、获得节点个数、给两个点之间的边来设计权重、删掉两点之间的边、取权重、进行标记、初始化标记数组这些基础方法

准备定义 输出两点之间的所有路径、从文件中读取校园节点信息进 节点数组、输出两点之间最短路径、输出两点路径之间的形式、获取结点 n 的名称、输出两点路径之间的形式、添加结点、保存结点名称、删除结点、规定一些途径点并输出路径、更新 Path 数组、获得路径数组、输出根据一些条件输出路径 必须经过一些点、输出经过点最少 的路径、输出全部路径这些核心方法

私有属性：准备设计初始化边数组、释放二维数组、适用于 Floyd 进行输出结果、初始化边数组、适用于 Dijkstra 进行输出结果这些依赖方法

对于 Node 这个类

准备定义 获取和设置顶点编号和顶点名称两个属性的方法

四、系统设计

(1) 数据结构的设计

```
private:  
    int from; //起始点 编号  
    int to; //终点 编号  
    double wt; //权重
```

对于 Edges 这个类

```
int numVertex, numEdge; //点数、边数  
double** matrix; //邻接矩阵  
Node* node; //顶点数组  
Edges eg[MAX]; //边数组  
int* mark; //指向存放有无访问该点的数组  
string fileName; //区分是否自己创建数组还是从文件中读取  
int flag; //1无向图 0有向图
```

对于 Graph 这个类

对于 Node 这个类

```
int No; //顶点的编号  
string Name; //顶点的名称
```

数据结构设计思路

先设计一个结点类，再设计一个边类，最后由这两个类组合而得图类，但是边类

我没有使用结点类作他的起点和终点，只是使用唯一编号将其联系在一起。

在图类中我是使用的二维指针创建的邻接矩阵，因此在其中边数组和结点数组我也是使用指针创建的，在其中我使用两个变量来记录边和点的数量；因为有两个应用案例因此我记录了文件名以此方便保存和读取；为了方便一些方法的实现我定义了一个标记数组

在边类中，因为权重可能为小数，我使用 double 类型存储权重

在点类中，因为编号不足以清晰向用户表明，因此我添加上了名称这个属性方便其表示

(2) 基本操作的设计

基本操作的抽象描述，关键算法的设计思路和算法流程图。

基本操作的抽象描述

a、最短路径实现校园地图查询

1、查询校园地点的信息

调用查询校园地点信息的方法并传入结点编号

2、查询校园两点间的最短路径

调用查询校园两点间的最短路径，并传入两点编号，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

3、查询校园任意两点的全部最短路径

调用查询校园任意两点间的最短路径，并传入两点编号，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

4、更新校园间路径长度

调用更新校园间路径长度，并传入两点编号，和想要修改的权重值，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

5、在两个地点间添加一条路径

跟步骤 4 一样的操作

6、删除两个地点的路径

调用删除两个地点的路径，并传入两点编号，和想要修改的权重值，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

7、增加一个地点

调用增加地点的方法，传入地点名称，增加地点在所有地点的后面

8、去除一个地点

调用去除地点的方法，不需要传入

b、DFS 实现仓库最优取货路径

1、获得两点之间所有路径的数组

不需要传入值，只需要调用获取两点间所有路径的数组的方法

2、输出途径某些点两点之间的所有路径

更新一下 Path 数组，调用输出途径某些点两点之间的所有路径，并传入两点编号，并输入路径点，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

3、输出两点之间的所有路径

更新一下 Path 数组，调用输出途径某些点两点之间的所有路径，并传入两点编号，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

4、输出两点之间途经点最少的路径

更新一下 Path 数组，调用输出两点之间途经点最少的路径，并传入两点编号，如果为无向图，两点的次序无关，如果为有向图，注意两点的次序

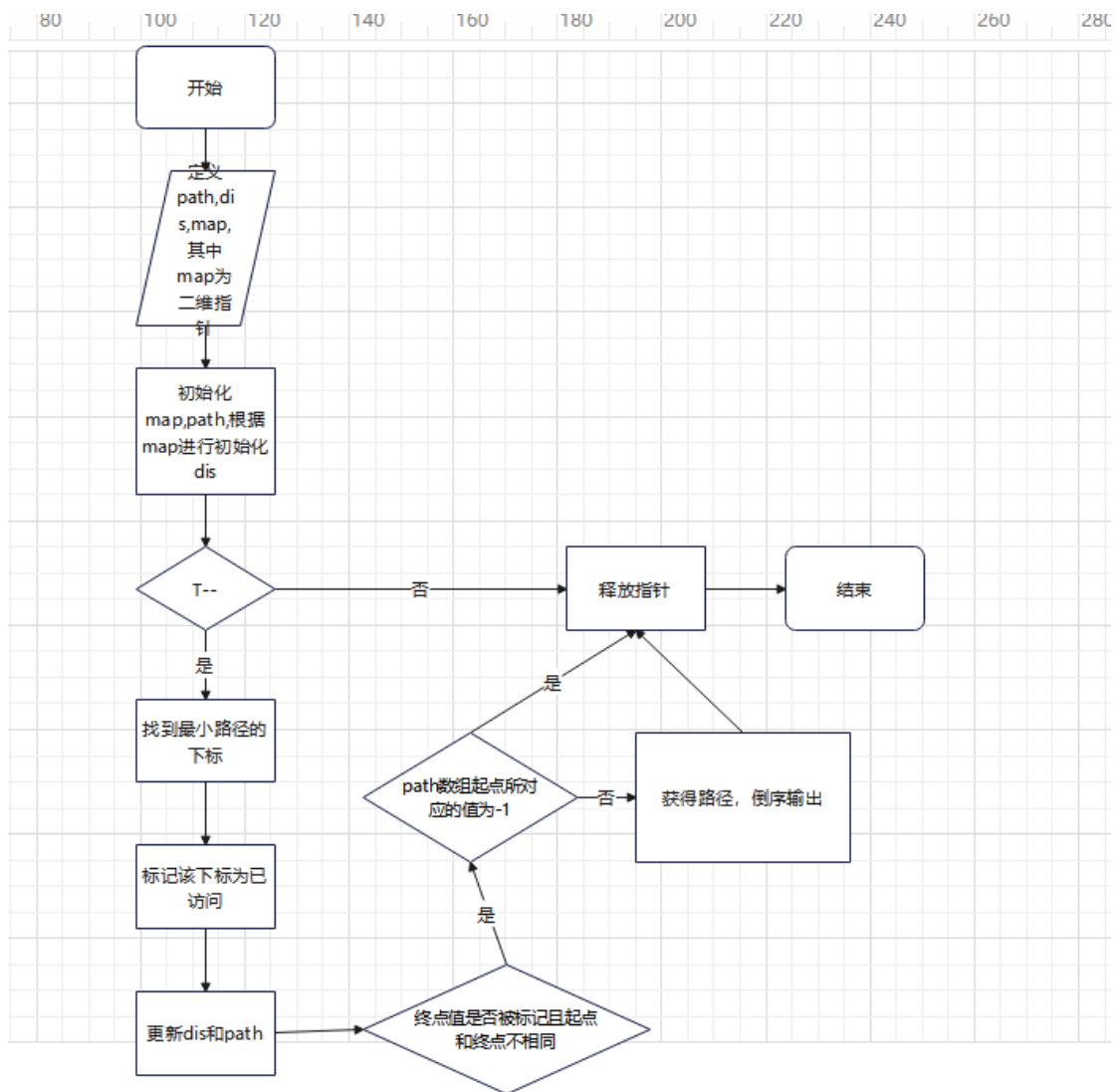
关键算法的设计思路和算法流程图。

校园

1、查询校园两点间的最短路径

设计思路：以 Dijkstra 算法为框架，先从当前实例中按条件初始化好一个 m 二维数组，叫 map，再将当前实例的标记数组全部设为未访问，初始化路径数组，若起点与某点之间有边且边的值不为 0x3f3f3f3f，在路径数组将该点设为起始点的值，否则就设为-1；初始化 dis 数组，将某点的 dis 数组设为起点至该点的边权重。初始化结束。下面是核心代码，将起点标记为已访问，进入循环，循环点数减一次，每一次循环，先计算出最小路径 dis 的下标，并标记此下标，更新 dis 的值和 path 的值，直至循环结束。调用输出函数，传入 dis, path, mark 数组和起点值、终点值。在这个函数中，先判断终点值是否被标记且起点和终点不相同，成立便进入，若 path 数组起点所对应的值为-1，输出无路径，退出函数；反而，先获得路径，存入 road，之后倒序输出 road 数组，最后释放所有指针。

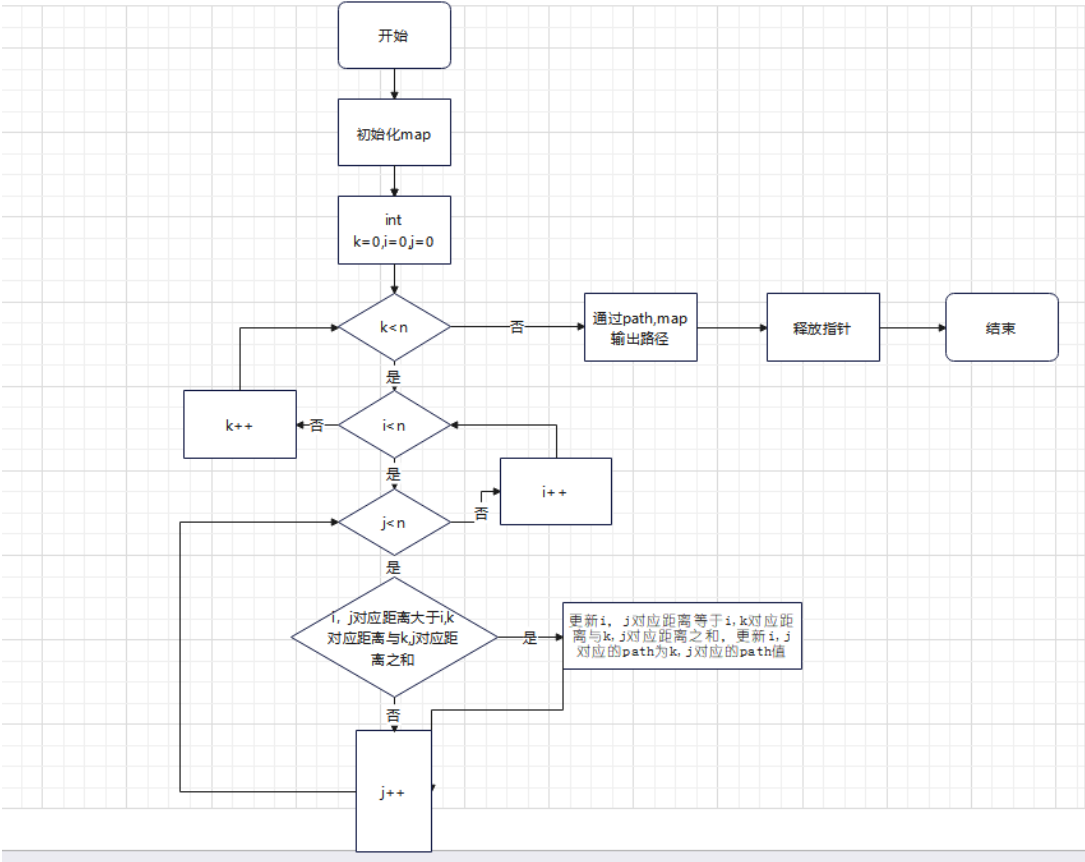
算法流程图



2、查询校园任意两点的全部最短路径

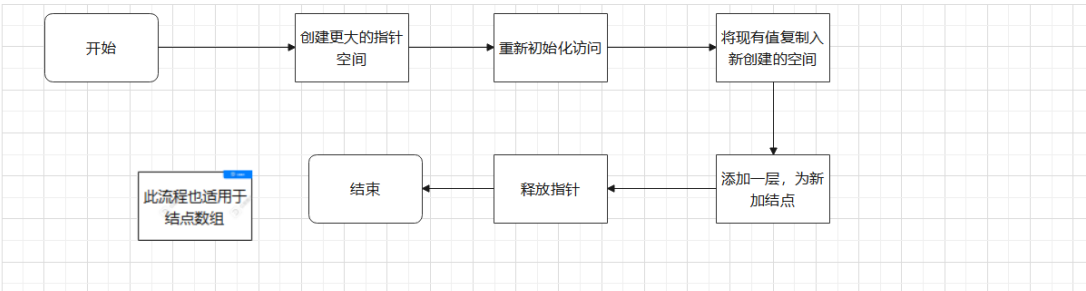
设计思路：本方法基于 Floyd 算法，先初始化 map，path，作为进行下一步的前提，使用三层循环，最外层循环的作为中继点，里面两层循环不断更换起点和终点，更新起点和终点的边的权重为 起点到中继点加上中继点到终点与起点到终点的最小值，更新路径数组为中继点到终点的边的权重，循环结束。开始输出路径，由于 path 数组是二维数组，因此我使用双层循环，在内层循环，如果两个下标不相同以及此时 map 数组的值不为无穷的，进入输入功能，初始化输出数组，按倒序输出输出数组。循环结束，开始释放创建指针。

算法流程图



3、添加结点

设计思路：我将新添加的结点依次向后添加，因此主要的工程就是要将此时的邻接矩阵更新即添加一层，又因为我是使用指针进行构建的，所以要重新分配一个更大的空间，将原先的复制进去，再按要求添加一层，再更新相应的值，最后将指针进行释放。对于结点数组也是这个思路

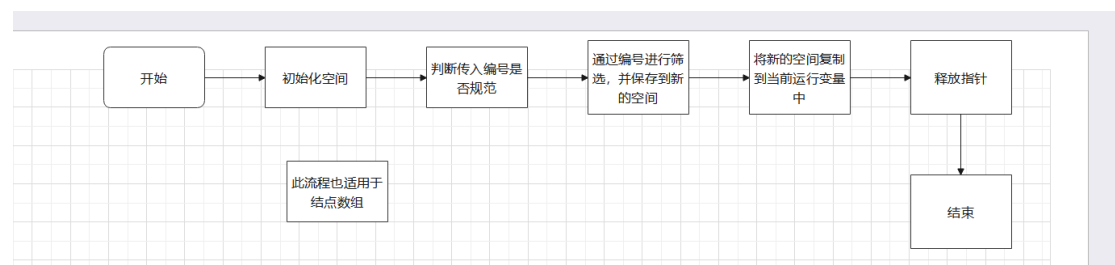


算法流程图：

4、删除结点

设计思路：对于删除结点，传入的为结点的编号，先要判断其编号是否规范，其次就是要新开一个空间进行对新邻接矩阵进行存储，对于传入的编号我们作为是否保存的依据。当邻接矩阵的横坐标、纵坐标等于传入编号就跳过，反之保存。下一步就是将其保存到邻接矩阵中，然后就是释放指针。对于结点数组也是这个思路

算法流程图

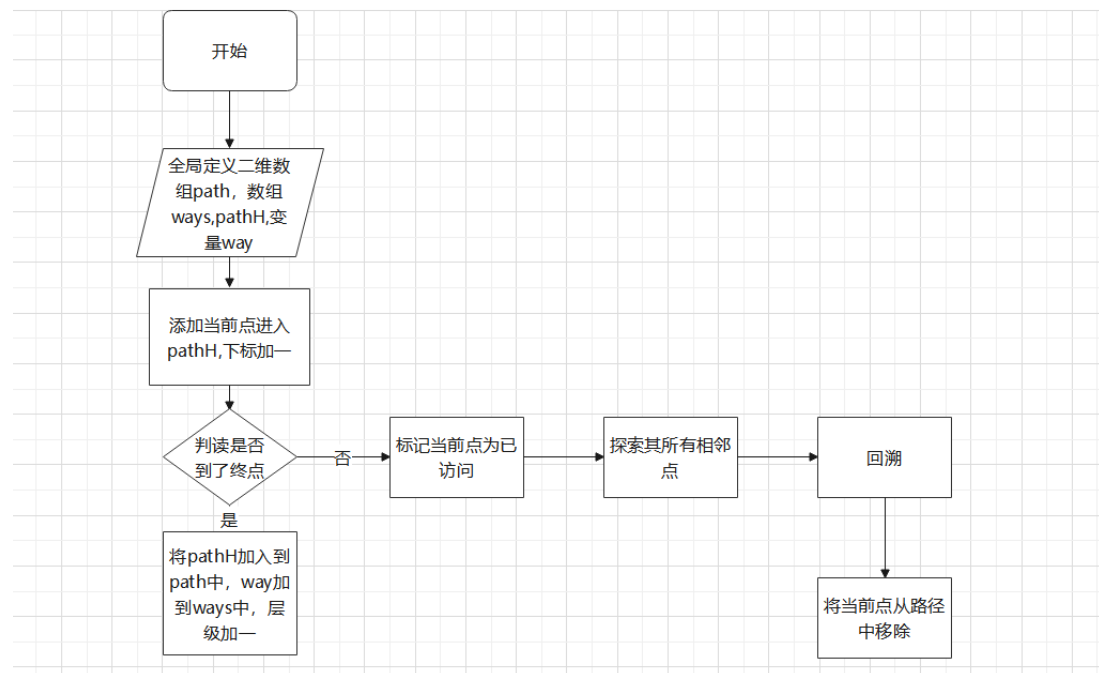


仓库

1、获得两点之间所有路径的数组

设计思路：本方法基于 DFS 算法，由于是要输出所有路径和对应的路径长度，因此我使用一个二维数组记录每一次路径经过的点，为全局定义，使用一个数组记录对应的路径长度，也为全局定义。使用一个一维数组记录当时某一层路径，一个变量记录该一层路径长度。按 DFS 的模板，首先记录此时的路径点，判断此时是不是到达了终点，到达终点便将这一层记录的数据进行保存，否则就标记此点为已访问，搜索所有相邻点，加上对应的路权值，后面就回溯，通过下标的减少将当前点从路径中移除，方法结束便得到了包含所有路径的数组和对应的路径长度

算法流程图

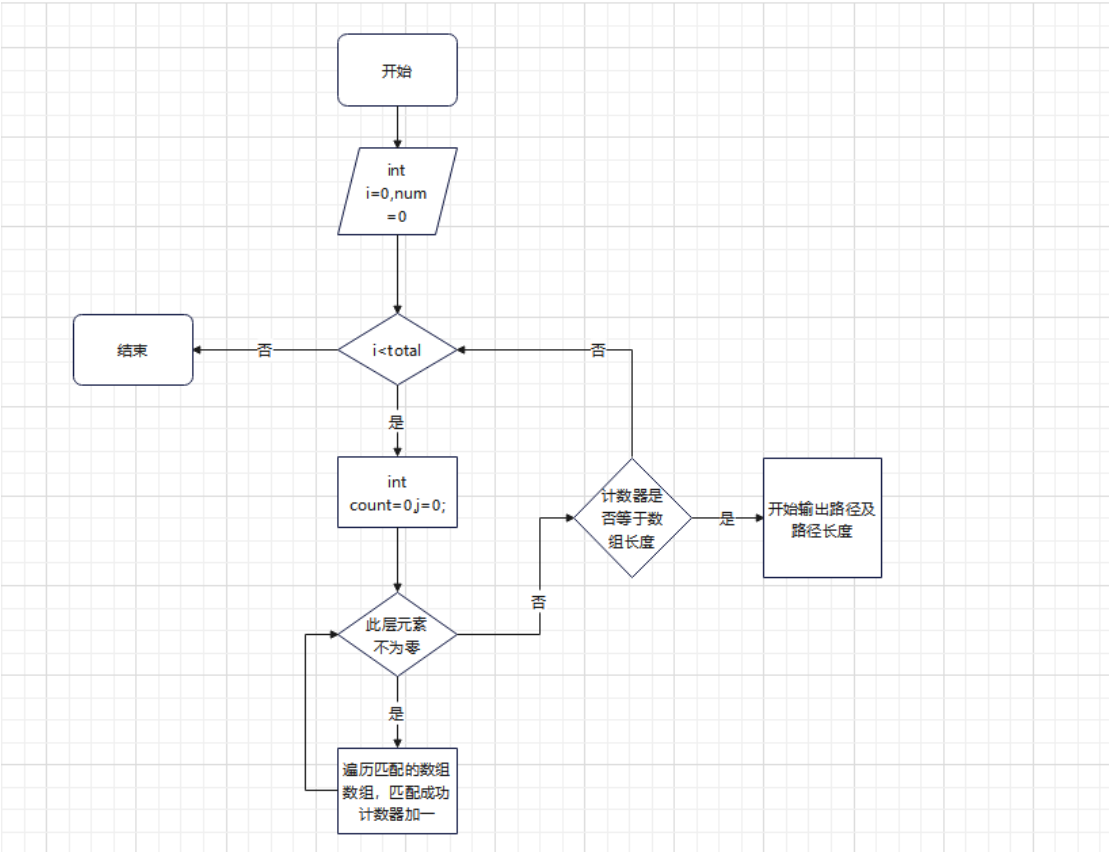


2、输出途径某些点两点之间的所有路径

设计思路：

由于通过第一个方法已经得到了所有路径的数组，接下来就是要路径包括我们传进来的点才能输出，先使用一层循环获得所有路径的数组其中一层数组，在遍历这一数组是否含有其中的点，用一个计数器来记录，如果达到传入数组的长度便按照一些格式进行输出这一层数组。

算法流程图

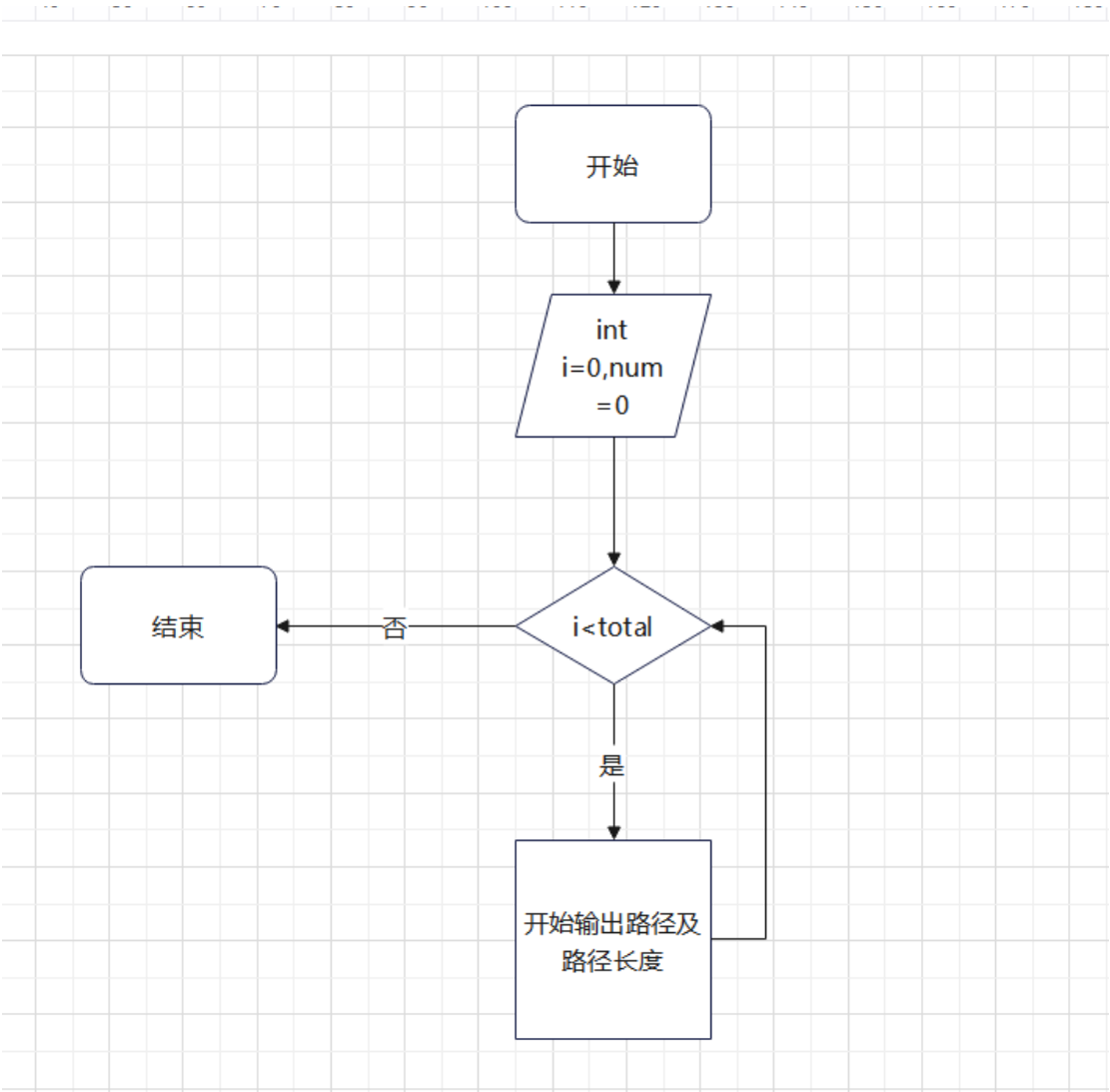


3、输出两点之间的所有路径

设计思路：

由于通过第一个方法已经得到了所有路径的数组，同时我们也实现了第二个方法，所以我们只要将传入参数全部设为 0, 0 即不作任何限制就可以全部输出了

算法流程图：

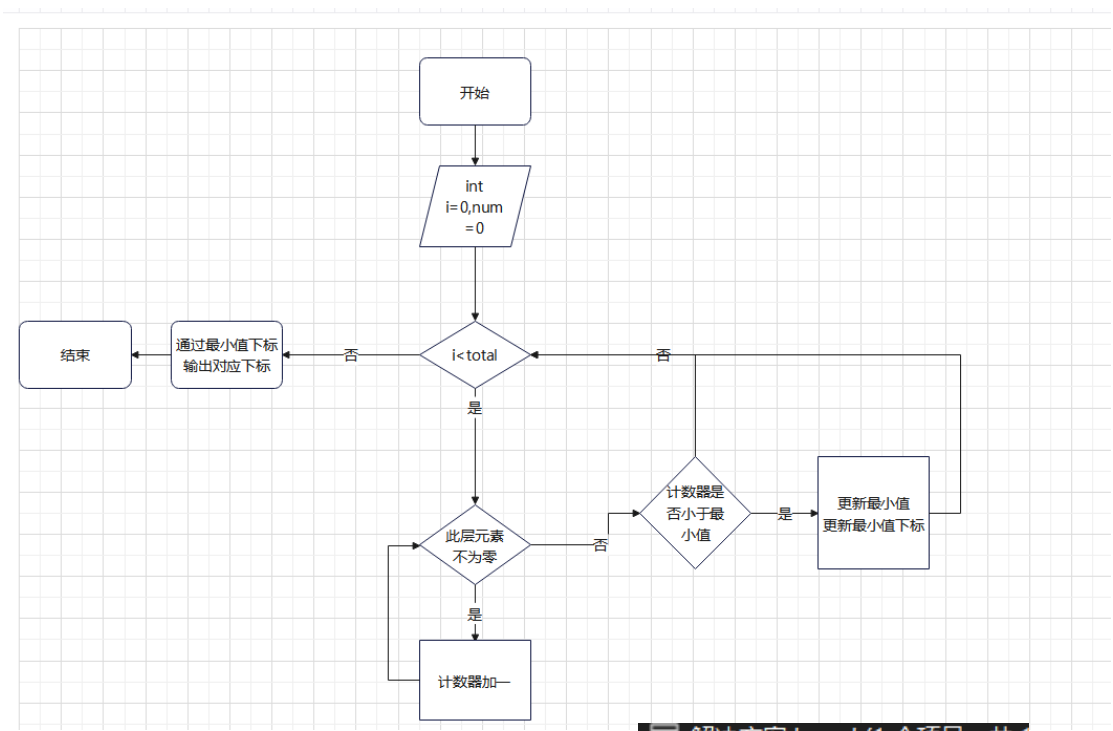


4、输出两点之间途经点最少的路径

设计思路：

通过第一个方法，我们已经得到了含有所有路径的数组，和其对应的路径长度数组，要实现途经点最少，那就是要找到对应路径长度数组最小值对应的下标，将下标对应的路径数组输出就行了

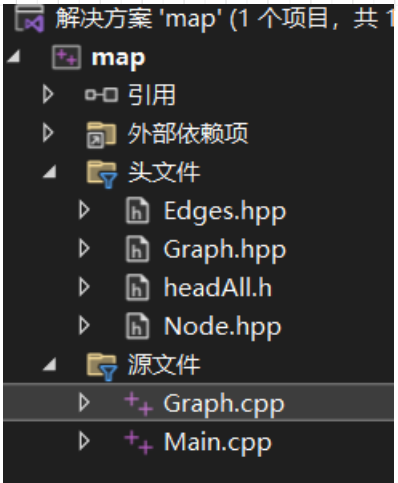
算法流程:



五、系统实现

主要功能的实现代码

项目结构



结点类定义及实现

```
#pragma once
#include "headAll.h"
class Node
{
private:
    int No; //顶点的编号
    string Name; //顶点的名称
public:
    Node() { No = -1; Name = "无顶点"; };
    Node(int No, string Name) : No(No), Name(Name) {}
    string GetName() { return this->Name; }
    int GetNo() { return this->No; }
    void SetName(string name) {
        this->Name = name;
    }
    void SetNo(int no) {
        this->No = no;
    }
};
```

用于规范包含头文件的头文件

```
#pragma once
#include<iostream>
using namespace std;

#include<string>
#include<string.h>
#include<fstream>
#include<chrono>
#define INF 0x3f3f3f3f
const int MAX = 155;
const int VISITED = 1;
const int UNVISITED = 0;
```

边类定义及实现

```
#include "headAll.h"
#pragma once
class Edges
{
private:
    int from; //起始点 编号
    int to; //终点 编号
    double wt; //权重
public:
    Edges() {};
    Edges(int from, int to, int wt)
    { :from(from), to(to), wt(wt) {} }
    int getTo() { return to; }
    int getFrom() { return from; }
    int getWt() { return wt; }
    void setTo(int to) { this->to = to; }
    void setFrom(int from) { this->from = from; }
    void setWt(double wt) { this->wt = wt; }
};
```

图类的定义

```
1  #pragma once
2  #include "headAll.h"
3  #include "Queue.h"
4  #include "Node.hpp"
5  #include "Edges.hpp"
6  class Graph
7  {
8  private:
9      int numVertex, numEdge; //点数、边数
10     double** matrix; //邻接矩阵
11     Node* node; //顶点数组
12     Edges eg[MAX]; //边数组
13     int* mark; //指向存放有无访问该点的数组
14     string fileName; //区分是否自己创建数组还是从文件中读取
15     int flag; //1无向图 0有向图
16     void deleteDouble(double** map); //释放二维数组
17     void DisplayPath(double *dis, int *path, int* mark, int v, int t); //适用于Dijkstra进行输出结果
18     void DisPath(int** path, double** map); //适用于Floyd进行输出结果
19     void getEdges(); //初始化边数组
20 public:
21     bool readFile(string fileName); //读取文件
22     bool writeFile(string fileName); //写文件
23     Graph(int n, string fileName);
24     Graph(int n, int flag);
25     ~Graph();
26     void Init(int n); //初始化
27     void printMatrix(double **map); //打印邻接矩阵
28     void printMatrix(); //打印邻接矩阵
29     void printEdges(); //打印边数组
30     int getV(); //获得节点个数
31     void setEdge(int from, int to, int wt); //给两个点之间的边来设计权重
32     void delEdge(int v1, int v2); //删掉两点之间的边
33     double weight(int v1, int v2); //取权重
34     void setMark(int v, int val); //进行标记
35     void initMark(); //初始化标记数组
36 public:
37     //测试校园地图
38     void Floyd(); //输出两点之间的所有路径
39     void initNodeInfo(string fileName); //从文件中读取校园节点信息进 节点数组
40     void Dijkstra(int v, int t); //输出两点之间最短路径
41     string GetNodeInfo(int n); //获取结点n 的名称
42     void dbInfo(int from, int to); //输出两点路径之间的形式
43     void addNode(string NodeName); //添加结点
44     void saveNode(string fileName); //保存结点名称
45     void deleteNode(int v); //删除结点
46     void RouteSome(); //规定一些途径点 并输出路径
47     void initPath(); //更新Path数组
48     //仓库
49     void dfs(int u); //获得路径数组
50     void FindPath(int start, int n); //输出根据一些条件输出路径, 必须经过一些点
```

图类一些方法的实现

初始化

```
void Graph::Init(int n)
{
    int i, n1 = n + 1;
    fileName = "abc";
    numVertex = n;
    numEdge = 0;
    mark = new int[n1]; //不使用节点0
    node = new Node[n1]; //不使用节点0
    //开始都标记 为未访问
    //memset(mark, UNVISITED, n);
    //添加顶点名称
    for (i = 1; i < n1; i++) {
        node[i].SetNo(i);
        node[i].SetName("V" + to_string(i));
    }

    for (i = 0; i < n1; i++)
        mark[i] = UNVISITED;

    matrix = (double**) new double* [n1];
    for (i = 0; i < n1; i++)
        matrix[i] = new double[n1];
    for (i = 0; i < n1; i++)
        for (int j = 0; j < n1; j++)
            matrix[i][j] = 0;
}
```


读、写文件

```
bool Graph::readFile(string fileName)
{
    ifstream f;
    int n = this->numVertex + 1;
    double x;
    int num = 0, start, end;
    f.open(fileName, ios::in);
    if (!f.is_open()) {
        cout << "打开失败" << endl;
        return false;
    }
    f.seekg(0);
    if (!f.eof()) { //文件区分无向图还是有向图
        f >> x;
        if (x == 1) {
            flag = 1;
            cout << "无向图" << endl;
        }
        else if (x == 0) {
            flag = 0;
            cout << "有向图" << endl;
        }
    }
    //读取邻接矩阵
    while (!f.eof())
    {
        f >> x;
        if (num == n * n)
            break;
        start = num / n;
        end = num % n;
        this->matrix[start][end] = x;

        num++;
        if (f.fail()) break; //解决eof多读一行的问题
    }
    f.close();
    return true;
}

bool Graph::writeFile(string fileName)
{
    ofstream f;
    int n = this->numVertex + 1;
    f.open(fileName, ios::out);
    if (f) {
        f << flag << endl; //标记是无向图还是有向图
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                f << this->matrix[i][j] << "\t";
            }
            f << endl;
        }
        f.close();
        return true;
    }
    cout << "file open error!" << endl;
    return false;
}

Graph::Graph(int n, string fileName) { ... }
```

校园

1、查询校园两点 间的最短路径 求出路径数组

```
391 void Graph::Floyd()
392 {
393     int n = this->numVertex+1;
394     int i, j, k;
395     double** map = new double* [n];
396     int** path = new int* [n];
397     //初始化 map
398     for (i = 0; i < n; i++) {
399         map[i] = new double[n];
400         path[i] = new int[n];
401         for (j = 0; j < n; j++) {
402             if (i == j) {
403                 map[i][j] = 0;
404                 path[i][j] = -1;
405             }
406             else if (this->matrix[i][j] == 0) {
407                 map[i][j] = INF;
408                 path[i][j] = -1;
409             }
410             else {
411                 map[i][j] = this->matrix[i][j];
412                 path[i][j] = i;
413             }
414         }
415     }
416     //核心代码
417     for (k = 1; k < n; k++) {
418         for (i = 1; i < n; i++) {
419             for (j = 1; j < n; j++) {
420                 if (map[i][j] > map[i][k] + map[k][j]) {
421                     map[i][j] = map[i][k] + map[k][j];
422                     path[i][j] = path[k][j];
423                 }
424             }
425         }
426     }
427     //输出
428     DisPath(path, map);
429     //释放指针
430     deleteDouble(map);
431     for (int i = 0; i < n; i++)
432         delete[] path[i];
433     delete path;
434 }
```

输出路 径数组

```
49 void Graph::DisPath(int** path, double** map)
50 {
51     int i, j, k, s, n = this->numVertex + 1;
52     int* road = new int[n], d;
53     for (i = 1; i < n; i++) {
54         for (int j = 1; j < n; j++) {
55             if (i != j && map[i][j] != INF) {
56                 string f = node[i].GetName(), end = node[j].GetName();
57                 cout << f << "->" << end << "的最短路径: ";
58                 k = path[i][j];
59                 d = 1; road[d]=j;
60                 while (k != -1 && k != i) {
61                     road[++d] = k;
62                     k = path[i][k];
63                 }
64                 road[++d] = i;
65                 cout << node[road[d]].GetName();
66                 for (s = d - 1; s > 0; s--) {
67                     cout << "->" << node[road[s]].GetName();
68                 }
69                 cout << "\t" << "该路径的长度为: " << map[i][j] << "米" << endl;
70             }
71         }
72     }
73     delete[] road;
74 }
```

2、查询校园任意两点的全部最短路径

获得路径数组

```
void Graph::Dijkstra(int v, int t)
{
    int i, j, pos = 1, n = this->numVertex+1;
    double min, sum = 0;
    this->initMark(); //初始化为0, 表示开始都没走过
    double* dis = new double[n];
    int* path = new int[n];
    double** map = new double* [n];
    //初始化 map
    for (int i = 0; i < n; i++) {
        map[i] = new double[n];
        for (int j = 0; j < n; j++) {
            if (i == j) {
                map[i][j] = 0;
            }
            else if (this->matrix[i][j] == 0) {
                map[i][j] = INF;
            }
            else {
                map[i][j] = this->matrix[i][j];
            }
        }
    }

    //初始化 path
    for (int i = 1; i < n; i++) {
        if (map[v][i] != 0 && map[v][i] != INF) {
            path[i] = v;
        }
        else {
            path[i] = -1;
        }
    }

    //初始化 dis
    for (i = 1; i < n; i++)
    {
        dis[i] = map[v][i];
    }

    this->setMark(v, VISITED);
    dis[v] = 0;
    int T = n - 2;
    //核心代码
    while (T--)
    {
        min = INF;
        for (j = 1; j < n; j++)
        {
            if (mark[j] == 0 && min > dis[j])
            {
                min = dis[j];
                pos = j;
            }
        }
        mark[pos] = 1; //表示这个点已经走过
        for (j = 1; j < n; j++)
        {
            if (mark[j] == 0 && dis[j] > min + map[pos][j]) {
                //更新dis的值
                dis[j] = map[pos][j] + min;
                path[j] = pos;
            }
        }
    }

    DisplayPath(dis, path, mark, v, t);
    this->deleteDouble(map); //释放map指针
    delete[] path;
    delete[] dis;
}
```

输出路径


```
void Graph::DisplayPath(double* dis, int* path, int* mark, int v, int t)
{
    int j, k, d, n = this->numVertex + 1;
    int* road = new int[n];

    if (mark[t] == 1 && t != v) {
        string f = node[v].GetName(), end = node[t].GetName();
        cout << f << "-" << end << "的最短路径: ";
        d = 1, road[d] = t;
        k = path[t];
        if (k == -1) {
            cout << "不存在路径" << endl;
        }
        else {
            //获得路径
            while (k != v) {
                road[++d] = k;
                k = path[k];
            }
            //输出路径
            road[++d] = v;
            cout << node[road[d]].GetName();
            for (j = d - 1; j > 0; j--) {
                cout << "-" << node[road[j]].GetName();
            }
            cout << "\t" << "该路径的长度为: " << dis[t] << "米" << endl;
        }
    }

    delete[] road;
}
```

3 添加结点

```
void Graph::addNode(string NodeName)
{
    //中转值
    int n = this->numVertex + 1;
    Node* anode = new Node[n + 1]; //不使用0
    double** amatrix = new double* [n + 1];
    this->numVertex++;
    mark = new int[n + 1];
    this->initMark();
    for (int i = 1; i < n; i++) {
        anode[i] = node[i];
    }
    for (int i = 0; i < n; i++) {
        amatrix[i] = new double[n];
        for (int j = 0; j < n; j++) {
            amatrix[i][j] = this->matrix[i][j];
        }
    }
    node = new Node[n + 1];
    matrix = new double* [n + 1];
    for (int i = 1; i < n; i++)
        node[i] = anode[i];
    node[n].SetName(NodeName);
    node[n].SetNo(n);
    for (int i = 0; i < n + 1; i++) {
        matrix[i] = new double[n + 1];
        for (int j = 0; j < n + 1; j++) {
            if (i < n && j < n)
                this->matrix[i][j] = amatrix[i][j];
            else
                this->matrix[i][j] = 0;
        }
    }
    cout << "成功添加" << NodeName << "\t" << "编号为: " << n << endl;
    delete[] anode;
    for (int i = 0; i < n; i++)
        delete[] amatrix[i];
    delete amatrix;
}
```

 (字段) double**Graph::matrix
邻接矩阵
联机搜索

4 删除结点

```
void Graph::deleteNode(int v)
{
    //中转值
    if (v > this->numVertex || v < 1) {
        cout << "不存在" << endl;
        return;
    }
    string info = node[v].GetName();
    int n = this->numVertex - 1;
    int i, j, k = 1, m;
    Node* anode = new Node[n + 1]; //不使用0
    double** amatrix = new double* [n + 1];
    this->numVertex = n;
    mark = new int[n + 1];
    this->initMark();
    for (i = 1; i < n + 2; i++) {
        if (i == v) continue;
        anode[k++] = node[i];
    }
    k = 0;
    for (int i = 0; i < n + 2; i++) {
        amatrix[i] = new double[n + 1];
        if (i == v) continue;
        m = 0;
        for (int j = 0; j < n + 2; j++) {
            if (j == v) continue;
            amatrix[k][m++] = this->matrix[i][j];
        }
        k++;
    }
    node = new Node[n + 1];
    matrix = new double* [n + 1];
    for (int i = 1; i < n + 1; i++)
        node[i] = anode[i];
    for (int i = 0; i < n + 1; i++) {
        matrix[i] = new double[n + 1];
        for (int j = 0; j < n + 1; j++) {
            this->matrix[i][j] = amatrix[i][j];
        }
    }
    cout << "编号为: " << v << "的" << info << "被删除" << endl;
    delete[] anode;
    for (int i = 0; i < n + 1; i++)
        delete[] amatrix[i];
}
```

仓库

1、获得两点之间所有路径的数组

```
void Graph::dfs(int u)
{
    pathH[cnt++] = u;
    // 如果到达了终点，输出路径
    if (u == t) {
        ways[total] = way;
        for (int j = 0; j < cnt; j++) {
            path[total][j] = pathH[j];
        }
        ++total;
    }
    else {
        // 标记为已访问
        mark[u] = VISITED;
        // 搜索所有相邻点
        for (int v = 1; v <= this->numVertex; v++) {
            if (this->matrix[u][v] && !mark[v]) {
                way += this->matrix[u][v];
                dfs(v);
                way -= this->matrix[u][v];
            }
        }
        // 回溯
        mark[u] = UNVISITED;
    }
    // 将当前点从路径中移除
    --cnt;
}
```

2、输出途径某些点两点之间的所有路径

```
void Graph::FindPath(int *match, int n)
{
    int i, j, num = 0;
    for (i = 0; i < total; i++) {
        // 计数器，用于记录匹配的数字的数量
        int count = 0;
        // 遍历每一行中的每个元素
        j = 0;
        while (path[i][j] != 0) {
            // 遍历匹配的数字数组
            for (int k = 0; k < n; k++) {
                // 如果发现了匹配的数字，就将计数器加 1
                if (path[i][j] == match[k]) {
                    count++;
                    // 因为已经找到了匹配的数字，所以可以退出循环
                    break;
                }
            }
            j++;
        }
        if (count == n) {
            flag = true;
            for (int k = 0; k < j; k++) {
                if (k < j - 1)
                    cout << node[path[i][k]].GetName() << "->";
                else
                    cout << node[path[i][k]].GetName();
            }
            cout << "\t" << "路径长度为:" << ways[i] << "米" << endl;
            num++;
        }
    }
    if (!num) cout << "无匹配路径" << endl;
    else cout << "共有" << num << "条路径" << endl;
}

void Graph::lessNumPath()
{
    int i, j, final, min=INF;
    for (i = 0; i < total; i++) {
```

3、输出两点之间的所有路径

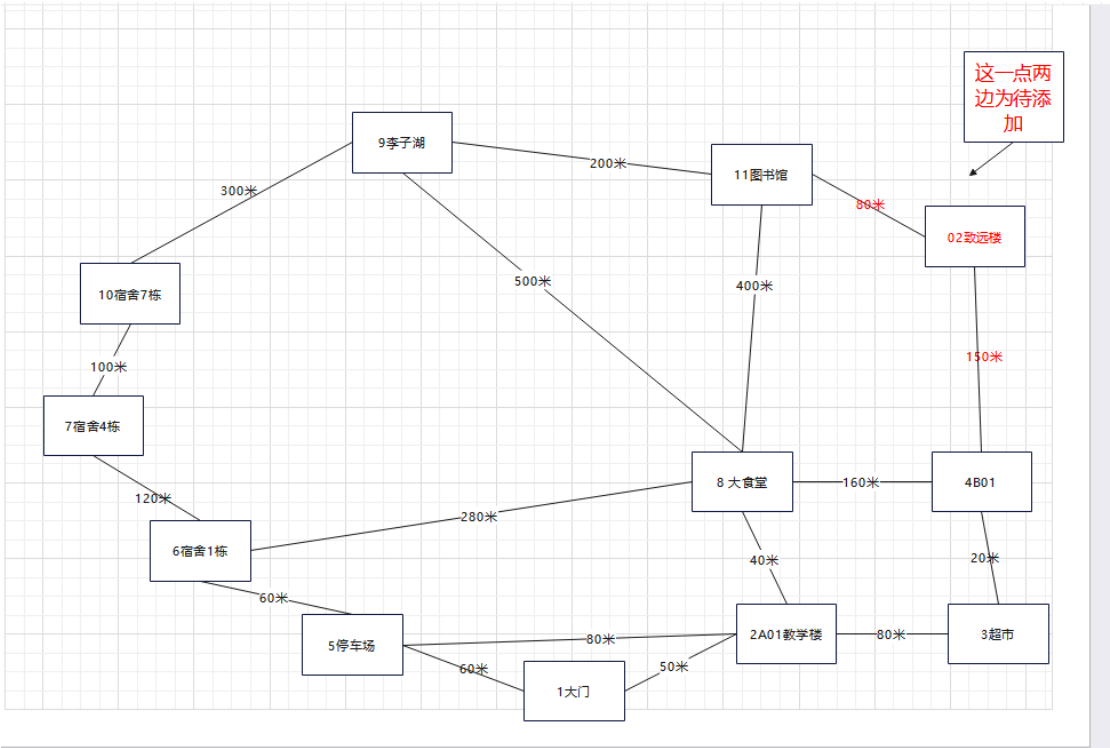
```
void Graph::AllPath()
{
    FindPath(0, 0);
}
```

```
1 1
2 0 0 0 0 0 0 0 0 0 0 0 0
3 0 0 50 0 0 60 0 0 0 0 0 0
4 0 50 0 80 0 80 0 0 40 0 0 0
5 0 0 80 0 20 0 0 0 0 0 0 0
6 0 0 0 20 0 0 0 0 160 0 0 0
7 0 60 80 0 0 0 60 0 0 0 0 0
8 0 0 0 0 0 60 0 120 280 0 0 0
9 0 0 0 0 0 0 120 0 0 0 100 0
10 0 0 40 0 160 0 280 0 0 500 0 400
11 0 0 0 0 0 0 0 500 0 300 200
12 0 0 0 0 0 0 100 0 300 0 0
13 0 0 0 0 0 0 0 0 400 200 0 0
14
```


xiaoyuanNodeInfo.txt

- 大门
- A01教学楼
- 超市
- B01教学楼
- 停车场
- 宿舍1栋
- 宿舍4栋
- 大食堂
- 李子湖
- 宿舍7栋
- 图书馆

抽象为



测试读取结点信息

```
auto starttime = system_clock::now();
g.initNodeInfo("xiaoyuanNodeInfo.txt");//从记录校园结点名称xiaoyuanNodeInfo.txt文件中读取结点名称
auto end = system_clock::now();
diff = end - starttime;
cout << "读入节点信息耗时" << diff.count() << "s" << endl;
```

结果

```
读入节点信息耗时0.0001445s
```

获得某点到某点的最短路径

我这里随机选择了两个点

```
starttime = system_clock::now();  
g.Dijkstra(1, 2); //获得某点到某点的最短路径  
end = system_clock::now();  
diff = end - starttime;  
cout << "计算两点最短路径信息耗时" << diff.count() << "s" << endl;
```

结果为

```
大门->A01教学楼的最短路径: 大门->A01教学楼    该路径的长度为: 50米  
计算两点最短路径信息耗时0.0002102s
```

任意两点的最短路径

```
starttime = system_clock::now();  
cout << g.GetNodeInfo(12) << endl; //获得某点的信息  
g.Floyd(); //任意两点的最短路径  
end = system_clock::now();  
diff = end - starttime;  
cout << "计算任意两点最短路径信息耗时" << diff.count() << "s" << endl;
```

入口1-A01教学楼的最短路径:	2门-A01教学楼	该路径的长度为: 50米
入口1-超市的最短路径:	入口1-A01教学楼-超市	该路径的长度为: 130米
入口1-B01教学楼的最短路径:	入口1-A01教学楼-超市-B01教学楼	该路径的长度为: 150米
入口1-停车场的最短路径:	入口1-停车场	该路径的长度为: 60米
入口1-宿舍1栋的最短路径:	入口1-停车场-宿舍1栋	该路径的长度为: 120米
入口1-宿舍4栋的最短路径:	入口1-停车场-宿舍1栋-宿舍4栋	该路径的长度为: 240米
入口1-食堂的最短路径:	入口1-A01教学楼-食堂	该路径的长度为: 90米
入口1-李子湖的最短路径:	入口1-A01教学楼-食堂-李子湖	该路径的长度为: 590米
入口1-图书馆的最短路径:	入口1-停车场-宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 340米
入口1-图书馆的最短路径:	入口1-A01教学楼-食堂-图书馆	该路径的长度为: 490米
A01教学楼-入口1的最短路径:	A01教学楼-超市	该路径的长度为: 50米
A01教学楼-B01教学楼的最短路径:	A01教学楼-超市-B01教学楼	该路径的长度为: 100米
A01教学楼-停车场的最短路径:	A01教学楼-停车场	该路径的长度为: 80米
A01教学楼-宿舍1栋的最短路径:	A01教学楼-停车场-宿舍1栋	该路径的长度为: 140米
A01教学楼-宿舍4栋的最短路径:	A01教学楼-停车场-宿舍1栋-宿舍4栋	该路径的长度为: 260米
A01教学楼-食堂的最短路径:	A01教学楼-食堂	该路径的长度为: 40米
A01教学楼-李子湖的最短路径:	A01教学楼-食堂-李子湖	该路径的长度为: 540米
A01教学楼-宿舍7栋的最短路径:	A01教学楼-停车场-宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 360米
A01教学楼-图书馆的最短路径:	A01教学楼-食堂-图书馆	该路径的长度为: 440米
超市-入口1的最短路径:	超市-A01教学楼-入口1	该路径的长度为: 130米
超市-A01教学楼的最短路径:	超市-A01教学楼	该路径的长度为: 80米
超市-B01教学楼的最短路径:	超市-B01教学楼	该路径的长度为: 20米
超市-停车场的最短路径:	超市-A01教学楼-停车场	该路径的长度为: 160米
超市-宿舍1栋的最短路径:	超市-A01教学楼-停车场-宿舍1栋	该路径的长度为: 220米
超市-宿舍4栋的最短路径:	超市-A01教学楼-停车场-宿舍1栋-宿舍4栋	该路径的长度为: 340米
超市-食堂的最短路径:	超市-A01教学楼-食堂	该路径的长度为: 120米
超市-李子湖的最短路径:	超市-A01教学楼-食堂-李子湖	该路径的长度为: 620米
超市-宿舍7栋的最短路径:	超市-A01教学楼-停车场-宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 440米
超市-图书馆的最短路径:	超市-A01教学楼-食堂-图书馆	该路径的长度为: 520米
B01教学楼-入口1的最短路径:	B01教学楼-超市-A01教学楼-入口1	该路径的长度为: 150米
B01教学楼-A01教学楼的最短路径:	B01教学楼-超市-A01教学楼	该路径的长度为: 100米
B01教学楼-超市的最短路径:	B01教学楼-超市	该路径的长度为: 20米
B01教学楼-停车场的最短路径:	B01教学楼-超市-A01教学楼-停车场	该路径的长度为: 180米
B01教学楼-宿舍1栋的最短路径:	B01教学楼-超市-A01教学楼-停车场-宿舍1栋	该路径的长度为: 240米
B01教学楼-宿舍4栋的最短路径:	B01教学楼-超市-A01教学楼-停车场-宿舍1栋-宿舍4栋	该路径的长度为: 360米
B01教学楼-食堂的最短路径:	B01教学楼-超市-A01教学楼-食堂	该路径的长度为: 140米
B01教学楼-李子湖的最短路径:	B01教学楼-超市-A01教学楼-食堂-李子湖	该路径的长度为: 640米
B01教学楼-宿舍7栋的最短路径:	B01教学楼-超市-A01教学楼-停车场-宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 460米
B01教学楼-图书馆的最短路径:	B01教学楼-超市-A01教学楼-食堂-图书馆	该路径的长度为: 540米
停车场-入口1的最短路径:	停车场-A01教学楼-入口1	该路径的长度为: 60米
停车场-A01教学楼的最短路径:	停车场-A01教学楼	该路径的长度为: 80米
停车场-超市的最短路径:	停车场-A01教学楼-超市	该路径的长度为: 160米
停车场-B01教学楼的最短路径:	停车场-A01教学楼-超市-B01教学楼	该路径的长度为: 180米
停车场-宿舍1栋的最短路径:	停车场-宿舍1栋	该路径的长度为: 60米
停车场-宿舍4栋的最短路径:	停车场-宿舍1栋-宿舍4栋	该路径的长度为: 180米
停车场-食堂的最短路径:	停车场-A01教学楼-食堂	该路径的长度为: 120米
停车场-李子湖的最短路径:	停车场-宿舍1栋-宿舍4栋-宿舍7栋-李子湖	该路径的长度为: 580米
停车场-宿舍7栋的最短路径:	停车场-宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 280米
停车场-图书馆的最短路径:	停车场-A01教学楼-食堂-图书馆	该路径的长度为: 520米
宿舍1栋-入口1的最短路径:	宿舍1栋-停车场-入口1	该路径的长度为: 120米
宿舍1栋-A01教学楼的最短路径:	宿舍1栋-停车场-A01教学楼	该路径的长度为: 140米
宿舍1栋-超市的最短路径:	宿舍1栋-停车场-A01教学楼-超市	该路径的长度为: 220米
宿舍1栋-B01教学楼的最短路径:	宿舍1栋-停车场-A01教学楼-超市-B01教学楼	该路径的长度为: 240米
宿舍1栋-停车场的最短路径:	宿舍1栋-停车场	该路径的长度为: 60米
宿舍1栋-宿舍4栋的最短路径:	宿舍1栋-宿舍4栋	该路径的长度为: 120米
宿舍1栋-食堂的最短路径:	宿舍1栋-停车场-A01教学楼-食堂	该路径的长度为: 180米
宿舍1栋-李子湖的最短路径:	宿舍1栋-宿舍4栋-宿舍7栋-李子湖	该路径的长度为: 520米
宿舍1栋-宿舍7栋的最短路径:	宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 220米
宿舍1栋-图书馆的最短路径:	宿舍1栋-停车场-A01教学楼-食堂-图书馆	该路径的长度为: 580米
宿舍4栋-入口1的最短路径:	宿舍4栋-宿舍1栋-停车场-入口1	该路径的长度为: 240米
宿舍4栋-A01教学楼的最短路径:	宿舍4栋-宿舍1栋-停车场-A01教学楼	该路径的长度为: 260米
宿舍4栋-超市的最短路径:	宿舍4栋-宿舍1栋-停车场-A01教学楼-超市	该路径的长度为: 340米
宿舍4栋-B01教学楼的最短路径:	宿舍4栋-宿舍1栋-停车场-A01教学楼-超市-B01教学楼	该路径的长度为: 360米
宿舍4栋-停车场的最短路径:	宿舍4栋-宿舍1栋-停车场	该路径的长度为: 180米
宿舍4栋-宿舍1栋的最短路径:	宿舍4栋-宿舍1栋	该路径的长度为: 120米
宿舍4栋-食堂的最短路径:	宿舍4栋-宿舍1栋-停车场-A01教学楼-食堂	该路径的长度为: 300米
宿舍4栋-李子湖的最短路径:	宿舍4栋-宿舍7栋-李子湖	该路径的长度为: 400米
宿舍4栋-宿舍7栋的最短路径:	宿舍4栋-宿舍7栋	该路径的长度为: 100米
宿舍4栋-图书馆的最短路径:	宿舍4栋-宿舍7栋-李子湖-图书馆	该路径的长度为: 600米
食堂-入口1的最短路径:	食堂-A01教学楼-入口1	该路径的长度为: 90米
食堂-A01教学楼的最短路径:	食堂-A01教学楼	该路径的长度为: 40米
食堂-超市的最短路径:	食堂-A01教学楼-超市	该路径的长度为: 120米
食堂-B01教学楼的最短路径:	食堂-A01教学楼-超市-B01教学楼	该路径的长度为: 140米
食堂-停车场的最短路径:	食堂-A01教学楼-停车场	该路径的长度为: 120米
食堂-宿舍1栋的最短路径:	食堂-A01教学楼-停车场-宿舍1栋	该路径的长度为: 180米
食堂-宿舍4栋的最短路径:	食堂-A01教学楼-停车场-宿舍1栋-宿舍4栋	该路径的长度为: 300米
食堂-李子湖的最短路径:	食堂-李子湖	该路径的长度为: 500米
食堂-宿舍7栋的最短路径:	食堂-A01教学楼-停车场-宿舍1栋-宿舍4栋-宿舍7栋	该路径的长度为: 400米
食堂-图书馆的最短路径:	食堂-图书馆	该路径的长度为: 400米
李子湖-入口1的最短路径:	李子湖-食堂-A01教学楼-入口1	该路径的长度为: 590米
李子湖-A01教学楼的最短路径:	李子湖-食堂-A01教学楼	该路径的长度为: 540米
李子湖-超市的最短路径:	李子湖-食堂-A01教学楼-超市	该路径的长度为: 620米
李子湖-B01教学楼的最短路径:	李子湖-食堂-A01教学楼-超市-B01教学楼</	

单独对边点进行测试

```
//单独对边点进行测试
int from, to, n;
from = 2; to = 4;
//对边进行修改或删除 前后对比
g.dbInfo(from, to);
g.setEdge(from, to, 70); //修改 2到4 路径权重
cout << "修改后: " << endl;
g.dbInfo(from, to);
cout << "-----" << endl;
g.delEdge(from, to); //删除 2到4 的路径权重
cout << "删除后: " << endl;
g.dbInfo(from, to);
cout << "-----" << endl;
//对点进行添加或删除 前后对比
g.addNode("超市");
n = g.getV();
cout << g.GetNodeInfo(n) << endl; //添加结点至结点数组尾部
cout << "-----" << endl;
g.deleteNode(12); //删除结点
cout << g.GetNodeInfo(n) << endl;
cout << "-----" << endl;
```

结果

```
计算任意两点最短路径信息耗时0.0191989s
A01教学楼到B01教学楼无路径
修改后:
A01教学楼到B01教学楼路径为: 70.000000
-----
删除后:
A01教学楼到B01教学楼无路径
-----
成功添加超市    编号为: 12
超市
-----
编号为: 12的超市被删除
输入有误
-----
```

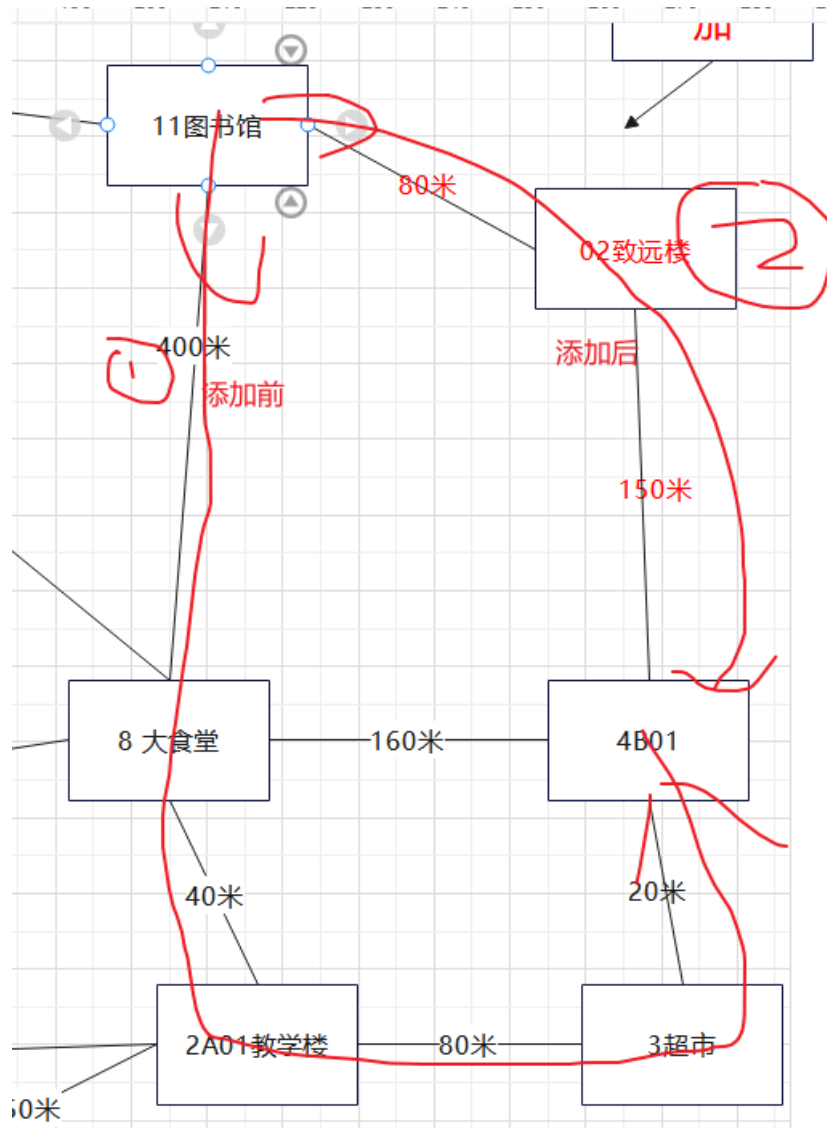
添加点 并为其附上边 查看 添加前后的路径变化
我这里添加了一条边和两个边

```
//添加点 并为其附上边 查看 添加前后的路径变化
//未修改边和点
starttime = system_clock::now();
g.Dijkstra(11, 4); //获得某点到某点的最短路径 添加前
end = system_clock::now();
diff = end - starttime;
cout << "计算两点最短路径信息耗时" << diff.count() << "s" << endl;
g.addNode("致远楼"); //添加一个点
n = g.getV();
g.setEdge(n, 11, 80); //为其添加两条边
g.setEdge(n, 4, 150);
starttime = system_clock::now();
g.Dijkstra(11, 4); //获得某点到某点的最短路径 添加后
end = system_clock::now();
diff = end - starttime;
cout << "计算两点最短路径信息耗时" << diff.count() << "s" << endl;
```

结果

```
图书馆->B01教学楼的最短路径: 图书馆->大食堂->A01教学楼->超市->B01教学楼 该路径的长度为: 540米
计算两点最短路径信息耗时0.0005483s
成功添加致远楼 编号为: 12
图书馆->B01教学楼的最短路径: 图书馆->致远楼->B01教学楼 该路径的长度为: 230米
计算两点最短路径信息耗时0.000502s
```

可以发现 两次路径长度发生了变化, 下面是变化



测试完全代码

```
#include "Graph.hpp"
void test3() { //无向图 求校园最短路径
    using namespace chrono;
    duration<double> diff;
    Graph g(11, "xiaoyuan.txt"); //从xiaoyuan.txt文件中读取数据邻接矩阵
    auto starttime = system_clock::now();
    g.initNodeInfo("xiaoyuanNodeInfo.txt"); //从记录校园结点名称xiaoyuanNodeInfo.txt文件中读取结点名称
    auto end = system_clock::now();
    diff = end - starttime;
    cout << "读入节点信息耗时" << diff.count() << "s" << endl;

    starttime = system_clock::now();
    g.Dijkstra(1, 2); //获得某点到某点的最短路径
    end = system_clock::now();
    diff = end - starttime;
    cout << "计算两点最短路径信息耗时" << diff.count() << endl;

    starttime = system_clock::now();
    cout << g.GetNodeInfo(12) << endl; //获得某点的信息
    g.Floyd(); //任意两点的最短路径
    end = system_clock::now();
    diff = end - starttime;
    cout << "计算任意两点最短路径信息耗时" << diff.count() << "s" << endl;

    //单独对边进行测试
    int from, to, n;
    from = 2; to = 4;
    //对边进行修改或删除 前后对比
    g.dbInfo(from, to);
    g.setEdge(from, to, 70); //修改 2到4 路径权重
    cout << "修改后: " << endl;
    g.dbInfo(from, to);
    cout << "-----" << endl;
    g.delEdge(from, to); //删除 2到4 的路径权重
    cout << "删除后: " << endl;
    g.dbInfo(from, to);
    cout << "-----" << endl;
    //对点进行添加或删除 前后对比
    g.addNode("超市");
    n = g.getV();
    cout << g.GetNodeInfo(n) << endl; //添加结点至结点数组尾部
    cout << "-----" << endl;
    g.deleteNode(12); //删除结点
    cout << g.GetNodeInfo(n) << endl;
    cout << "-----" << endl;

    //添加点 并为其附上边 查看 添加前后的路径变化
    //未修改边和点
    starttime = system_clock::now();
    g.Dijkstra(11, 4); //获得某点到某点的最短路径
    end = system_clock::now();
    diff = end - starttime;
    cout << "计算两点最短路径信息耗时" << diff.count() << "s" << endl;
    g.addNode("致远楼");
    n = g.getV();
    g.setEdge(n, 11, 80);
    g.setEdge(n, 4, 150);
    starttime = system_clock::now();
    g.Dijkstra(11, 4); //获得某点到某点的最短路径
    end = system_clock::now();
    diff = end - starttime;
    cout << "计算两点最短路径信息耗时" << diff.count() << "s" << endl;

    cout << "是否保存(保存 y)";
    string s; cin >> s;
    if (s == "y") g.saveNode("xiaoyuanNodeInfo.txt");
}
```


仓库

由于仓库系统和校园系统有些功能方法是相同的如，添加结点，计算两点的最短路径，因此我这里只对其具有的特有方法进行测试

利用文件进行初始化

```
Graph g(13, "changku.txt");//从changku.txt文件中读取数据邻接矩阵
g.initNodeInfo("changkuNodeInfo.txt");//从记录仓库结点名称changkuNodeInfo.txt文件中读取结点名称
```

changku.txt

changkuNodeInfo.txt

入口
1号货架
2号货架
3号货架
4号货架
5号货架
6号货架
7号货架
8号货架
9号货架
10号货架
11号货架
出口

读取 path 数组

```
auto starttime = system_clock::now();
g.initPath();//更新path数组
auto end = system_clock::now();
diff = end - starttime;
cout << "初始化path数组信息耗时" << diff.count() << "s" << endl;
```



```

void Graph::initPath()
{
    total = 0;
    memset(ways, 0, sizeof ways);
    this->initMark(); //设置全部点未访问
    //cout << "请输入起始点-终点" << endl;
    //cin >> s >> t; //起始点 终点
    s = 1; //入口
    t = 13; //出口
    dfs(s);
}

```

结果

```
初始化path数组信息耗时1.26e-05s
```

输出途径某些点的路径展示（这里自己规定了三个点

```

starttime = system_clock::now();
g.RouteSome(); //输出途径某些点的路径展示
end = system_clock::now();
diff = end - starttime;
cout << "途径某些点的路径展示信息耗时" << diff.count() << "s" << endl;

void Graph::RouteSome()
{
    const int n = 3;
    int match[n] = { 8, 7, 10 }; //途经点
    int i;
    cout << "从" << this->GetNodeInfo(s) << "->" << this->GetNodeInfo(t) << endl << "途径以下点:"
    for (i = 0; i < n; i++) {
        cout << this->GetNodeInfo(match[i]) << " ";
    }
    cout << endl;
    FindPath(match, n);
}

```

结果

```

从入口->出口
途径以下点:
7号货架 6号货架 9号货架
入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:388米
入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->9号货架->11号货架->出口 路径长度为:473米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:382米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->9号货架->11号货架->出口 路径长度为:467米
入口->10号货架->7号货架->6号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:370米
入口->10号货架->7号货架->6号货架->8号货架->9号货架->11号货架->出口 路径长度为:455米
共有6条路径
途径某些点的路径展示信息耗时0.0022496s

```

输出路径点最少的路径展示

```
starttime = system_clock::now();
g. lessNumPath(); //输出路径点最少的路径展示
end = system_clock::now();
diff = end - starttime;
cout << "最少路径点的路径展示信息耗时" << diff.count() << "s" << endl;
```

结果

途径点最少的路径如下：
入口->10号货架->9号货架->11号货架->出口 路径长度为:382米
最少路径点的路径展示信息耗时0.0002421s

全部路径展示

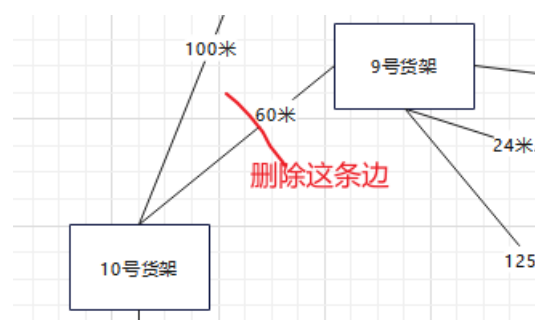
```
starttime = system_clock::now();
g.AllPath(); //输出全部路径的路径展示
end = system_clock::now();
diff = end - starttime;
cout << "全部路径展示信息耗时" << diff.count() << "s" << endl;
```

结果

入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:388米
入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->9号货架->11号货架->出口 路径长度为:473米
入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->11号货架->出口 路径长度为:388米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:382米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->9号货架->11号货架->出口 路径长度为:467米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->11号货架->出口 路径长度为:382米
入口->10号货架->7号货架->6号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:370米
入口->10号货架->7号货架->6号货架->8号货架->9号货架->11号货架->出口 路径长度为:455米
入口->10号货架->7号货架->6号货架->8号货架->11号货架->出口 路径长度为:370米
入口->10号货架->9号货架->3号货架->8号货架->11号货架->出口 路径长度为:370米
入口->10号货架->9号货架->8号货架->11号货架->出口 路径长度为:455米
入口->10号货架->9号货架->11号货架->出口 路径长度为:290米
共有12条路径
全部路径展示信息耗时0.0041311s

当我删除一条边

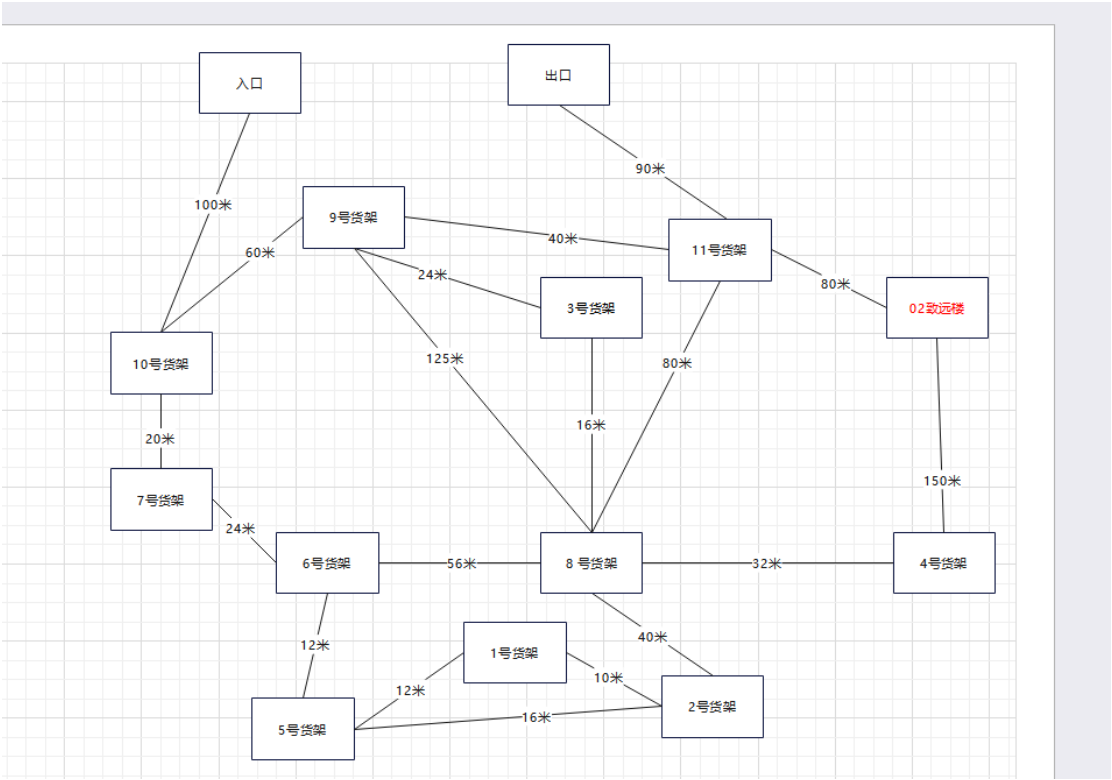
```
g.delEdge(10, 9); //删除一条边
g.initPath();
g.AllPath();
```



全部路径展示结果

入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:388米
入口->10号货架->7号货架->6号货架->5号货架->1号货架->2号货架->8号货架->11号货架->出口->出口 路径长度为:388米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->3号货架->9号货架->11号货架->出口 路径长度为:382米
入口->10号货架->7号货架->6号货架->5号货架->2号货架->8号货架->11号货架->出口->11号货架->出口 路径长度为:382米
入口->10号货架->7号货架->6号货架->8号货架->3号货架->9号货架->11号货架->出口->出口 路径长度为:370米
入口->10号货架->7号货架->6号货架->11号货架->出口->11号货架->出口 路径长度为:370米
入口->10号货架->9号货架->3号货架->8号货架->11号货架->出口->11号货架->出口 路径长度为:370米
入口->10号货架->9号货架->11号货架->出口->9号货架->11号货架->出口 路径长度为:290米

抽象出来的图：



七、实验分析

(1) 算法的性能分析

主要针对于关键算法进行分析。

校园

1 查询校园两点间的最短路径

本算法基于 Dijkstra 算法

此算法的时间复杂度为 $O(n^2)$ ，因为有两个嵌套的 for 循环，每个循环在 $O(n)$ 时间内运行。第一个 for 循环运行所有顶点以查找最小距离顶点，第二个 for 循环运行所有顶点以更新最小距离。

此算法的空间复杂度为 $O(n)$ ，因为它使用三个大小为 n 的数组：dis、path 和 mark。

此外，还有一个嵌套循环，用于创建大小为 $n \times n$ 的 2 维 map 数组，这为空间复杂性贡献了额外的 $O(n^2)$ 。因此，**整体空间复杂度为 $O(n^2)$** 。

2、查询校园任意两点的全部最短路径

本算法基于 Floyd 算法

算法的时间复杂度为 $O(n^3)$ ，其中 n 是图中的顶点数。这是因为该算法使用嵌套的 for 循环，每个循环遍历所有 n 个顶点。

算法的空间复杂度也是 $O(n^2)$ ，因为它需要一个包含 n 行和 n 列的 2 维数组 (map) 来存储顶点对之间的最短路径距离，以及另一个具有 n 行和 n 列的 2 维数组 (path) 来存储顶点对之间的最短路径中的中间顶点。

3、增加一个地点

本算法时间复杂度为 $O(n^2)$ ，因为有两个嵌套的 for 循环，每个循环迭代 2 维数组的所有元素。**空间复杂度也是 $O(n^2)$** ，因为 2 维数组的大小随着顶点数的平方而增加。

4、去除一个地点

本算法的时间复杂度为 $O(n^2)$ ，因为有两个嵌套的 for 循环，每个循环迭代 2 维数组的所有元素。**空间复杂度也是 $O(n^2)$** ，虽然说 2 维数组的大小减少了 1，但由于分配了新的空间所以空间复杂度仍为 $O(n^2)$

仓库

1、获得两点之间所有路径的数组

本算法的时间复杂度为 $O(n^2)$ ，因为内部 for 循环迭代 2 维数组的所有元素，并且对图中的每个顶点调用此循环。空间复杂度为 $O(n)$ ，因为路径数组的大小与图中的顶点数成正比。

2、输出途径某些点两点之间的所有路径

本算法的时间复杂度为 $O(n^2)$ ，因为有两个嵌套的 for 循环，每个循环迭代数组的所有元素。空间复杂度为 $O(1)$ ，因为所用数组的大小不依赖于输入大小。

3、输出两点之间途经点最少的路径

此代码的时间复杂度为 $O(n)$ ，因为只有一个 for 循环遍历数组的所有元素。空间复杂度为 $O(1)$ ，因为所用数组的大小不依赖于输入大小

(2) 数据结构的分析

通过性能分析总结此种结构的优缺点，并说明其适用场景。

优点

- 1、图可以很好地表示复杂的关系和连接，比如社交网络、交通网络、图书管理系统等。
- 2、图可以方便地描述有向和无向关系。
- 3、图可以用于对图中节点的数据进行搜索和遍历。

缺点

- 1、通过上面的算法分析不难看出图的时间复杂度和空间复杂度都很高，只要数据量大，时间花费和空间花费将很高
- 2、图的存储和操作可能比其他数据结构复杂。

使用场景

图相对于其他数据结构在生活中应用广泛。

图书管理系统：图可以用于表示图书管理系统中的图书、类别和作者之间的关系。

电子电路设计：图可以用于表示电子电路中的电路元件和连接之间的关系。

计算机网络：图可以用于表示计算机网络中的节点和连接之间的关系。

其他应用：图还可以用于表示生物网络、知识图谱、Web 图谱等复杂的关系和连接。

八、实验总结

主要针对本实验的分析、设计、实现、测试等环节进行总结，包含收获与不足，此部分的阐述应较为详细与全面。

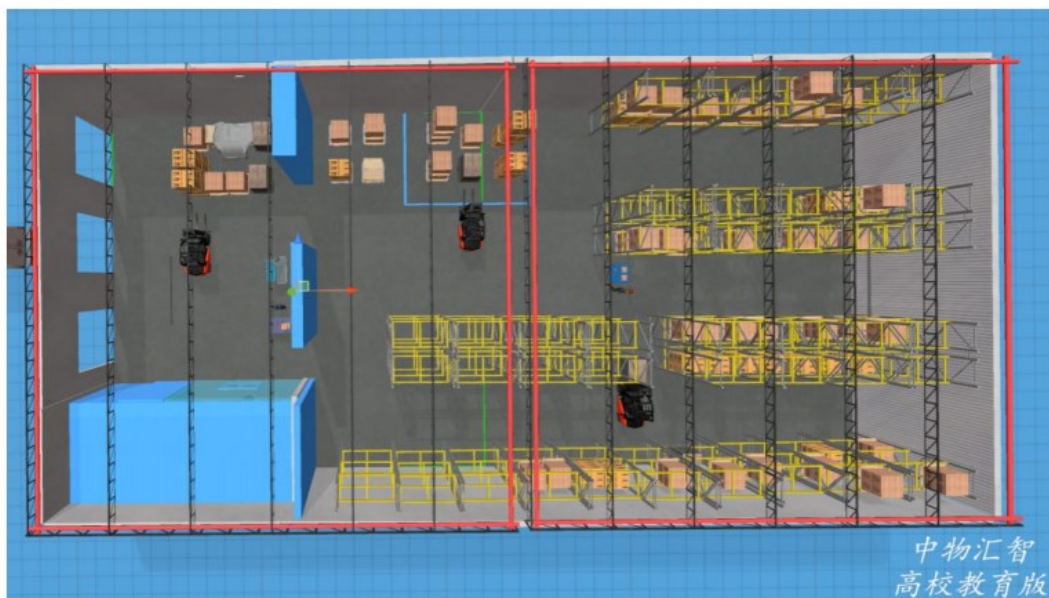
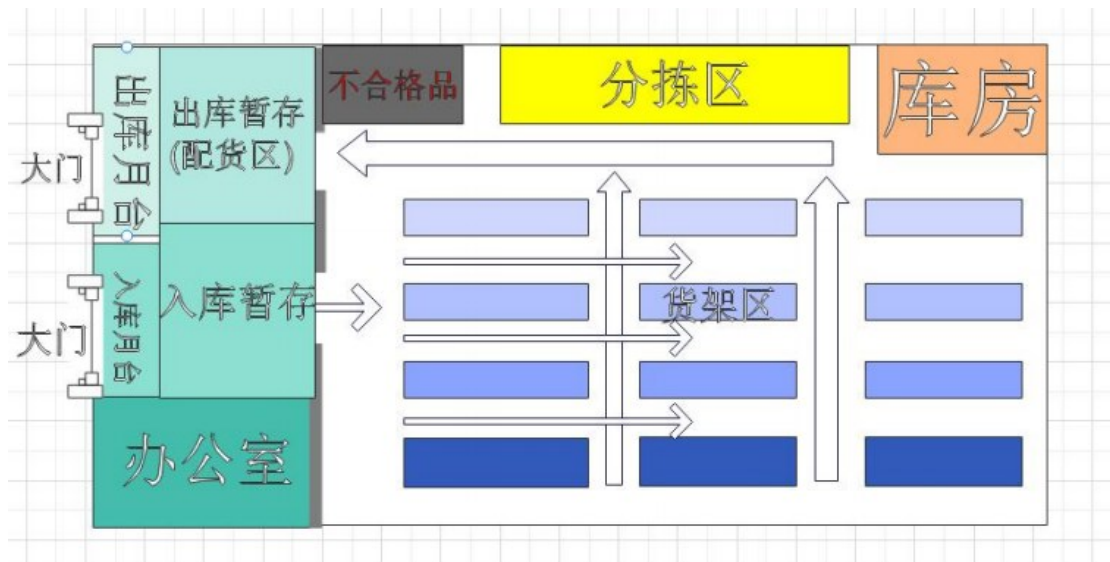
一开始并没有理解题目的意思，就按自己的想法将图的一些算法都实现了，如 DFS BFS, Dijkstra, Floyd, Prim, Kruskal, 拓扑排序。再加上利用 DFS, Dijkstra, Floyd 进行应用编程，一个文件中的代码量达到了 900 多行。我一看，这么多代码，是不是有点不对，就去问问其他同学，经过一段交流。原来只要应用一些算法进行应用编程就好了。得了，白花了一两天时间。然后就是删除一些不必要的算法了，最终的代码行数在 500 行左右。

在此基础上，我修修补补最终实现了两个应用编程，一个是校园地图查询，另一个就是仓库系统。校园地图查询是我根据学校大致地点分布抽取的一部分地图实现的，在实现了它的全部功能后，我来到一个很尴尬的时间点，就是截止时间快到了，而我又想再加一个应用编程，就是仓库系统，但是没什么时间来写了，还好鲁大师贴心的为我们延长了时间。这才我有时间去完成我特别想完成的东西。

这个仓库系统是我参加物流设计大赛，这个十分契合本次大作业，因此我特此加了一个应用。那个案例是要我们解决仓库的两个痛点，一个是货架摆放问题（货物该怎么更有效率放置在货架上，另一个是就是员工怎么更有效率从起点到终

点途径某些个货架取货。我是负责第二个痛点的。当时我是使用的 python 的遗传算法进行解决的，现在我在此次编程中我使用的是 DFS 算法去解决的。这两种算法的不同之处有以下两点，一个是实现难度，另一个是数据规模较大时，两种算法的应用。遗传算法实现难度很大，但是他对解决数据规模大很有成效，而 DFS 实现简单，但是数据规模一大就 t1l 了。

下面是当时的建模图像



本次收获就是，第一次使用课上学到的数据结构解决了比赛中的问题，还有就是要看清楚题目要求

不足之处：就是没有合理安排时间在本该截止时间提交作业

参考文献：

- 1.
- 2.