

**课 程 实 验 报 告**

**课程名称： 数据结构实验**

实验名称： 基于链表的树、图实现

**专业班级：** 信安XXXX班

**学 号：** U2021XXXXX

**姓 名：** MTX

**指导教师：** 马晓静

**报告日期：** 2022.11.10

目 录

[**1 基于链表的建图 1**](#_Toc119482712)

[1.1 需求分析 1](#_Toc119482713)

[1.2 总体设计 1](#_Toc119482714)

[1.3 数据结构 1](#_Toc119482715)

[1.4 算法设计 2](#_Toc119482716)

[1.5 系统实现 3](#_Toc119482717)

[1.6 系统测试 4](#_Toc119482718)

[1.7 复杂度分析 4](#_Toc119482719)

[1.8 结果分析 4](#_Toc119482720)

[1.9 实验小结 5](#_Toc119482721)

[**2 增加站点，删除 6**](#_Toc119482722)

[2.1 需求分析 6](#_Toc119482723)

[2.2 总体设计 6](#_Toc119482724)

[2.3 数据结构 7](#_Toc119482725)

[2.4 算法设计 7](#_Toc119482726)

[2.5 系统实现 9](#_Toc119482727)

[2.6 系统测试 10](#_Toc119482728)

[2.7 复杂度分析 11](#_Toc119482729)

[2.8 结果分析 11](#_Toc119482730)

[2.9 实验小结 11](#_Toc119482731)

[**3 从指定站点出发，计算到另一站点的最短距离和途径站点序列 12**](#_Toc119482732)

[3.1 需求分析 12](#_Toc119482733)

[3.2 总体设计 12](#_Toc119482734)

[3.3 数据结构 13](#_Toc119482735)

[3.4 算法设计 14](#_Toc119482736)

[3.5 系统实现 16](#_Toc119482737)

[3.6 系统测试 17](#_Toc119482738)

[3.7 复杂度分析 17](#_Toc119482739)

[3.8 结果分析 18](#_Toc119482740)

[3.9 实验小结 19](#_Toc119482741)

[**参考文献 20**](#_Toc119482742)

[**附录 基于链表的树、图实现的源程序 21**](#_Toc119482743)

[1.基于链表的建图 21](#_Toc119482744)

[2.增加站点，删除 25](#_Toc119482745)

[3.从指定站点出发，计算到另一站点的最短距离和途径站点序列 34](#_Toc119482746)

# 1 基于链表的建图

## 1.1 需求分析

### 1.1.1 功能需求

使用邻接表构成有向图（或者双向链表）来表达地铁线路，存储武汉地铁1号线、2号线、6号线和7号线在前两站的站点信息。其中，地铁线路均为双向线路，相同站名的地铁站为转乘车站。

### 1.1.2 输入输出需求

使用邻接表构成有向图（或者双向链表）来表达地铁线路，存储武汉地铁1号线、2号线、6号线和7号线在汉口区域的站点信息。

输入形式：

总线路条数n 线路号1 站名1 到下一站的距离 站名2 …… 到下一站的距离 站名n 0(到下一站距离为0，代表该站是线路最后一站) 线路号n 站名1 到下一站的距离 站名2 …… 到下一站的距离 站名n 0(到下一站距离为0，代表该站是线路最后一站)

输出形式：

线路号 站名1 到下一站的距离 站名2 …… 到下一站的距离 站名n

## 1.2 总体设计

整个程序分为字符串读取系统，字符串输出系统2个部分。

字符串读取系统将读入的不同线路的字符串数据分别放在不同的双向循环链表中。

字符串输出系统会依次遍历多个链表，实现输出功能。

## 1.3 数据结构

typedef struct Node //双向链表节点 { char name[20]; double distance; struct Node\* pre; struct Node\* next;}Node,\*NodePtr;

使用结点内的指针连接结点，形成双向循环链表

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

## 1.4 算法设计

字符串读取系统，字符串输出系统分别如图1-1，1-2所示。

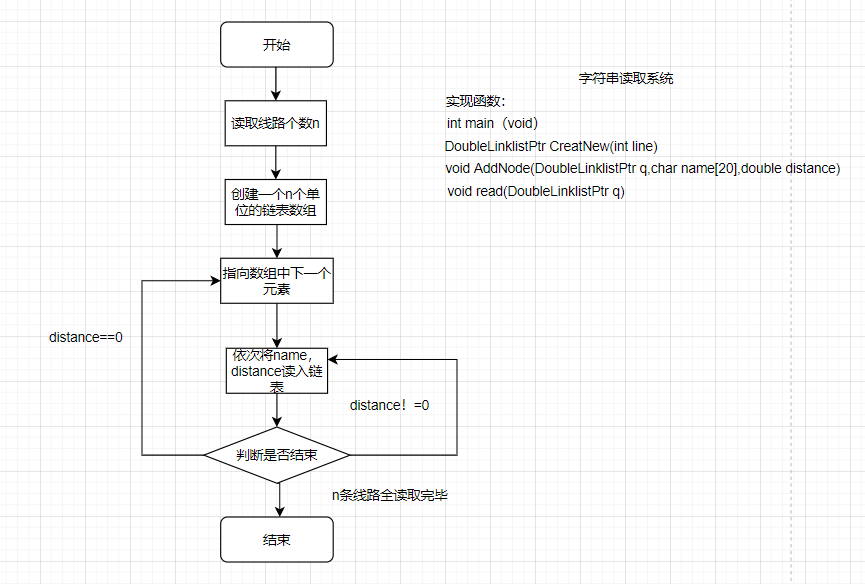


图1-1 字符串读取系统

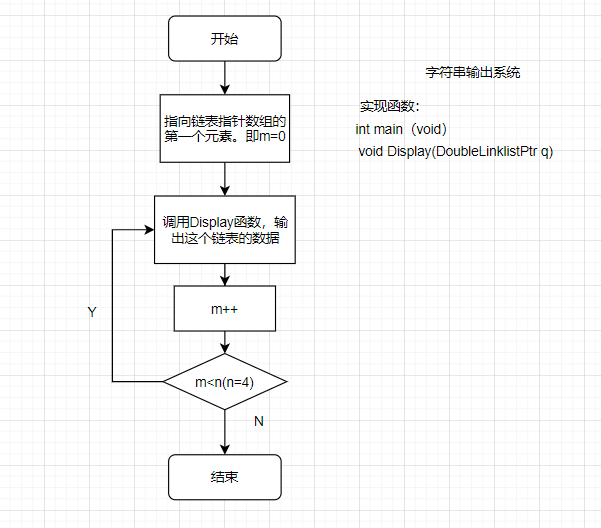


图1-2 字符串输出系统

## 1.5 系统实现

本程序全程在Microsoft Visual Studio 2022上编写、编译、调试、运行，并最终在Educoder平台上运行通过。

主要函数以及功能如表1-1所示。

表1-1 主要函数及功能

|  |  |
| --- | --- |
| 函数名 | 主要功能 |
| int main() | 实现读取地铁线路、输出地铁线路的功能 |
| DoubleLinklistPtr CreatNew(int line) | 实现根据地铁线路号 ,创建一个空的双向循环链表的功能 |
| void AddNode(DoubleLinklistPtr q,char name[20],double distance) | 实现双向链表中加入节点的功能 |
| void read(DoubleLinklistPtr q) | 实现将站点信息读入的功能 |
| void Display(DoubleLinklistPtr q) | 实现按要求格式输出一条地铁线路的各站点的功能 |

## 1.6 系统测试

支持Educoder平台的所有可见测试用例与隐藏测试用例，均通过，如图1-4所示。

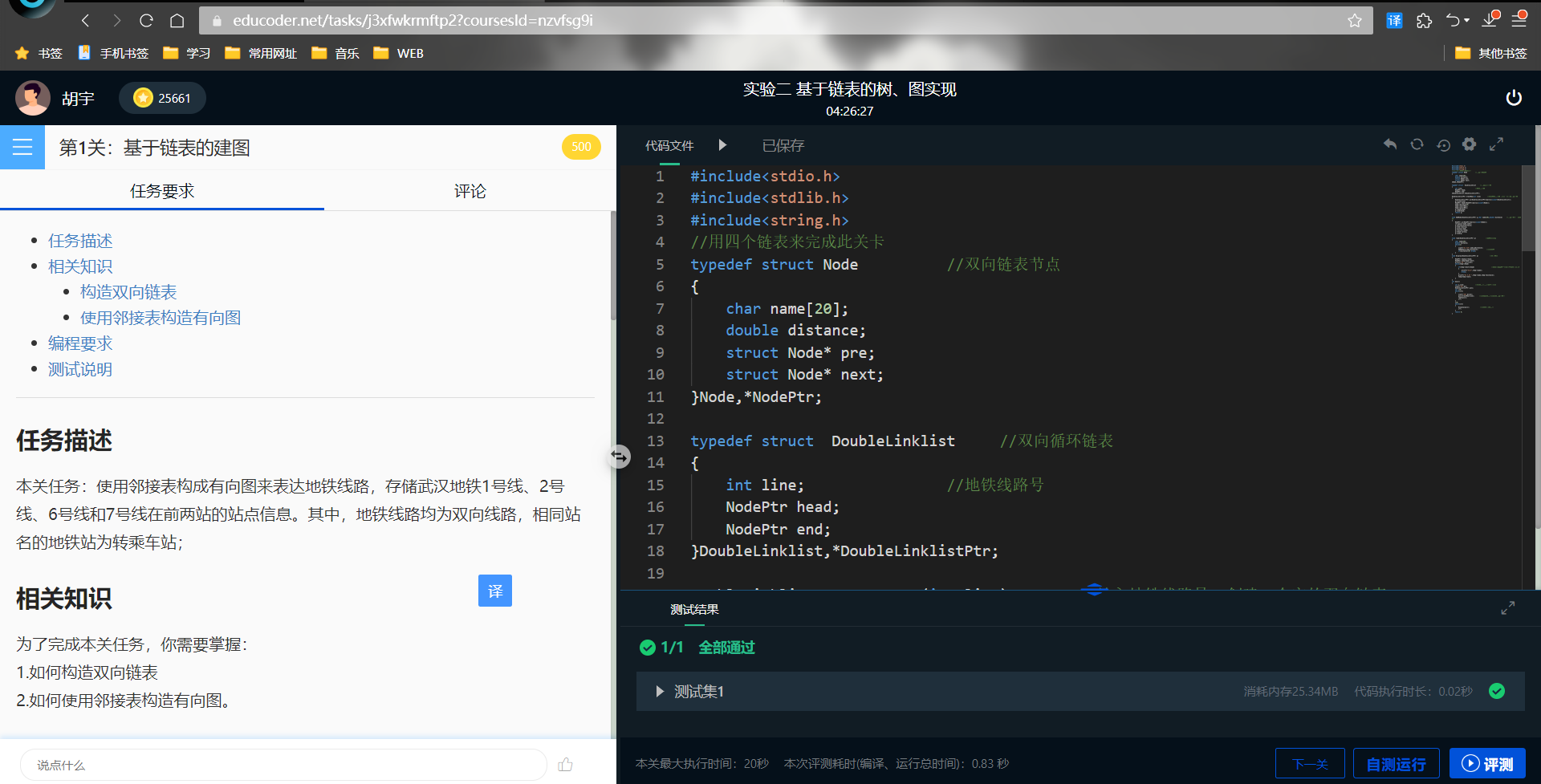


图1-3 测试通过

## 1.7 复杂度分析

设n为四条线路站点数之和

时间复杂度：

创建四个链表，将站点信息存入每个节点中，输出时也只需要遍历一遍，故时间复杂度为。

空间复杂度：  
 数据存储空间为站点个数的线性级别（包括距离和站点名称的信息），故空间复杂度为。

## 1.8 结果分析

成功通过所有的给定测试用例，表明该基于链表的建图成功，且能完成站点信息的读入，地铁线路的输出的任务。

## 1.9 实验小结

通过本此实验，我熟练的能够建立双向循环链表，定义该结构体类型的指针，用于指向各结点，分别建立具有添加、删除、修改、查询等功能的子函数，完成相应功能。这其中要用到对链表的删除、插入等知识。其中模块化的程序设计理念让我收益良多，现在拿到一个大的任务，我都会先画流程图，按照功能模块的不同将整个任务分解成易解决的小任务，再逐一编写每个小任务，这里就要尤其注意函数的封装，封装的好才能提高程序运行的安全性和可移植性。

# 2 增加站点，删除

## 2.1 需求分析

### 2.1.1 功能需求

对存储的线路信息，进行站点的增加、删除。

### 2.1.2 输入输出需求

进行站点的增加、删除。

输入格式：

add（选择进行增加操作） 线路号 要增加站点的距前一个站点的距离 要增加站点的距后一个站点的距离 要增加位置的前一个站点名称 站点名称 （若增加的站点是该线路上的第一个，则前一个距离为0，不需要输入增加位置的前一个站点名称；若增加的站点是该线路上的最后一个，则后一个距离为0） delete（选择进行删除操作） 线路号 要删除的站点名称

输出格式：

操作成功，线路号 站名1 到前一站的距离 站名2 …… 到前一站的距离 站名n/增加失败，已有同名站点/增加失败，没有与输入的增加位置前一站点同名的站点/删除失败，没有同名站点

## 2.2 总体设计

整个程序分为字符串读取系统、站点信息删增系统、字符串输出系统3个部分。

字符串读取系统将读入的不同线路的字符串数据分别放在不同的双向循环链表中。

站点信息删增系统会识别是删除指令还是增加指令，并针对特定地铁线路进行站点信息的删增处理。

字符串输出系统，先输出未修改的所有地铁线路。如果修改失败，则输出失败原因；如果站点信息修改成功，则输出修改后的地铁线路。

## 2.3 数据结构

typedef struct Node //双向链表节点 { char name[20]; double distance; struct Node\* pre; struct Node\* next;}Node,\*NodePtr;

使用结点内的指针连接结点，形成双向循环链表

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

## 2.4 算法设计

字符串读取系统、站点信息删增系统、字符串输出系统分别如图2-1，2-2，2-3所示。

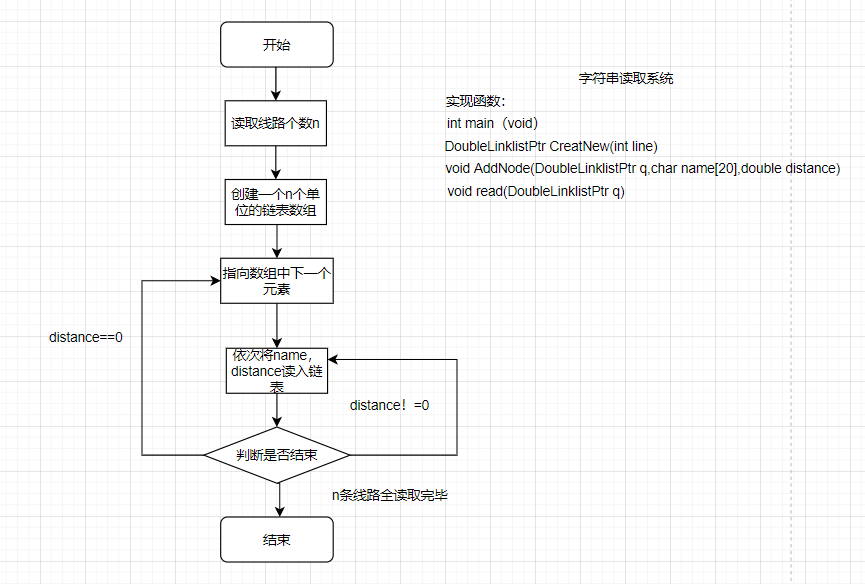


图2-1 字符串读取系统

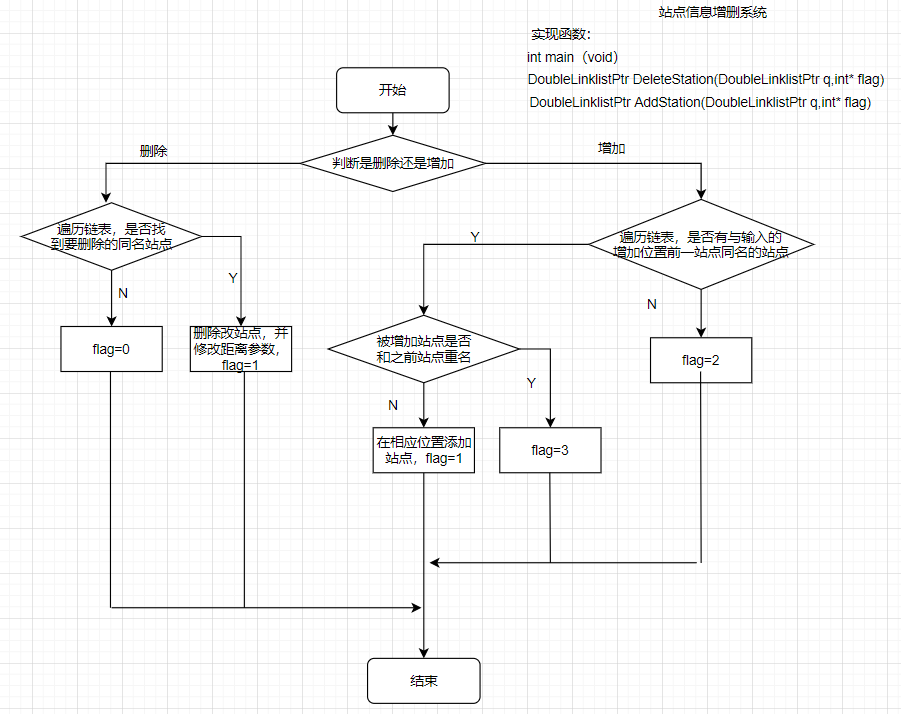


图2-2 站点信息增删系统

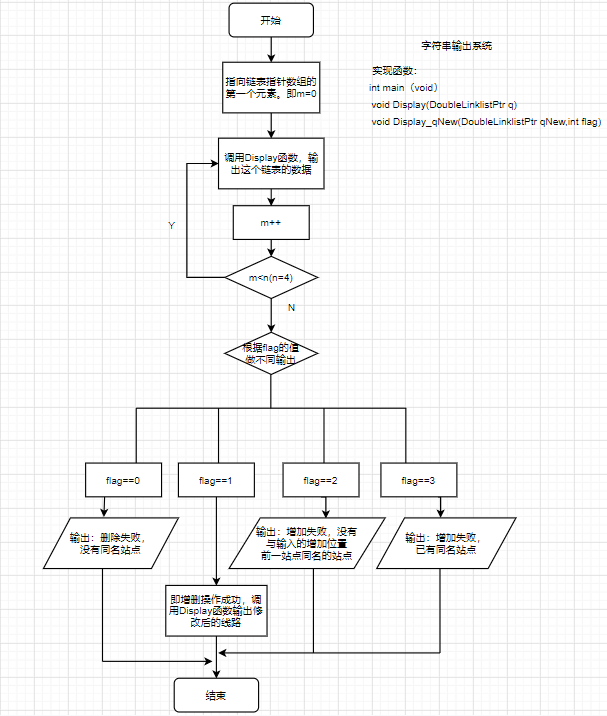


图2-3 字符串输出系统

## 2.5 系统实现

本程序全程在Microsoft Visual Studio 2022上编写、编译、调试、运行，并最终在Educoder平台上运行通过。

主要函数以及功能如表2-1所示。

表2-1 主要函数及功能

|  |  |
| --- | --- |
| 函数名 | 主要功能 |
| int main() | 实现读取地铁线路并增加或删除站点的功能，再输出修改后的地铁线路的功能 |
| DoubleLinklistPtr CreatNew(int line) | 实现根据地铁线路号 ,创建一个空的双向循环链表的功能 |
| void AddNode(DoubleLinklistPtr q,char name[20],double distance) | 实现双向链表中加入节点的功能 |
| void read(DoubleLinklistPtr q) | 实现将站点信息读入的功能 |
| void Display(DoubleLinklistPtr q) | 实现按要求格式输出一条地铁线路的各站点的功能 |
| DoubleLinklistPtr DeleteStation(DoubleLinklistPtr q,int\* flag) | 实现站点的删除的功能 |
| DoubleLinklistPtr AddStation(DoubleLinklistPtr q,int\* flag) | 实现站点的添加的功能 |
| void Display\_qNew(DoubleLinklistPtr qNew,int flag) | 实现输出修改后地铁线路的功能 |

## 2.6 系统测试

支持Educoder平台的所有可见测试用例与隐藏测试用例，均通过，如图2-4所示



图2-4 测试通过

## 2.7 复杂度分析

设n为四条线路站点数之和

时间复杂度：

创建四个链表，将站点信息存入每个节点中，故时间复杂度为。修改节点时，最好情况判断1次即可找到要修改站点；最坏情况则是要遍历整条地铁线路才能找到是否有符合要求的修改线路，故时间复杂度为。输出节点时，遍历一整条地铁线路即可，故时间复杂度为。综合输入、查找修改、输出三个步骤，时间复杂度为。

空间复杂度：  
 数据存储空间为站点个数的线性级别（包括距离和站点名称的信息），故空间复杂度为。

## 2.8 结果分析

成功通过所有的给定测试用例，其中包括成功删除或增站点后的新线路输出；包括遇到同名站点时，增加失败的输出；包括没有找到同名站点，删除失败时的输出；包括没有与输入的增加位置前一站点同名的站点时，增加失败时的输出。表明增加、删除站点的功能成功实现使用需求。

## 2.9 实验小结

同过本次实验，我更关注细节的处理。站点信息的读取和存储，和实验一中大相径庭，这里就不过多赘述。本次编写我遇到的问题是对那些特殊情况的输出，如何判断“删除失败，没有同名站点”“增加失败，没有与输入的增加位置前一站点同名的站点”“增加失败，已有同名站点”这三种特殊情况，我采用了用一个flag变量记录 DoubleLinklistPtr DeleteStation(DoubleLinklistPtr q,int\* flag)函数和DoubleLinklistPtr DeleteStation(DoubleLinklistPtr q,int\* flag)函数中的结果，再将flag变量传给void Display\_qNew(DoubleLinklistPtr qNew,int flag)函数做相应输出。这样的好处是将输出变得十分简便、可读性很高。

# 3 从指定站点出发，计算到另一站点的最短距离和途径站点序列

## 3.1 需求分析

### 3.1.1 功能需求

从指定站点出发，计算出到另一个站点的最短距离和途径的地铁站序列。不同的线路之间可能存在相同的站点作为换乘车站，因此在两个站点之间路线不唯一。使用邻接表构成有向图来表达地铁线路，存储武汉地铁1号线、2号线、6号线和7号线的部分站点信息。其中，地铁线路均为双向线路，相同站名的地铁站为转乘车站。

### 3.1.2 输入输出需求

使用邻接表构成有向图来表达地铁线路，存储武汉地铁1号线、2号线、6号线和7号线的部分站点信息。尝试利用我们提供的部分站点信息，用关卡一的代码生成有向图后，查找两个站点之间的最短路径。

输入形式：

总线路条数n 线路号1 站名1 到下一站的距离 站名2 …… 到下一站的距离 站名n 0(到下一站距离为0，代表该站是线路最后一站) 线路号n 站名1 到下一站的距离 站名2 …… 到下一站的距离 站名n 0(到下一站距离为0，代表该站是线路最后一站) 站名i 站名j（要查找的两个站点）

输出形式：

最短距离s 站名i 到下一站的距离 站名i+1 …… 站名j-1 到下一站的距离 站名j

## 3.2 总体设计

整个程序分为字符串读取建图系统、求解图中最短路径系统、字符串输出系统3个部分。

字符串读取建图系统先将读入的不同线路的字符串数据分别放在不同的双向循环链表中，再用循环链表建立一个有向图。

求解图中最短路径系统将读入的两个站点转化为对应顶点表中下标，再在边表中利用狄克斯特拉算法找到这两个节点的最短路径。

字符串输出系统将最短路径长度，以及最短路径按要求输出。

## 3.3 数据结构

typedef struct Node //双向链表节点 { char name[20]; double distance; struct Node\* pre; struct Node\* next;}Node,\*NodePtr;

使用结点内的指针连接结点，形成双向循环链表

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

typedef struct //用邻接矩阵建立图 { VertexType vexs[MAXVEX]; EdgeType arc[MAXVEX][MAXVEX]; int numVertexes,numEdges; }MGraph;

## 3.4 算法设计

字符串读取建图系统、求解图中最短路径系统、字符串输出系统分别如图3-1，3-2，3-3所示。

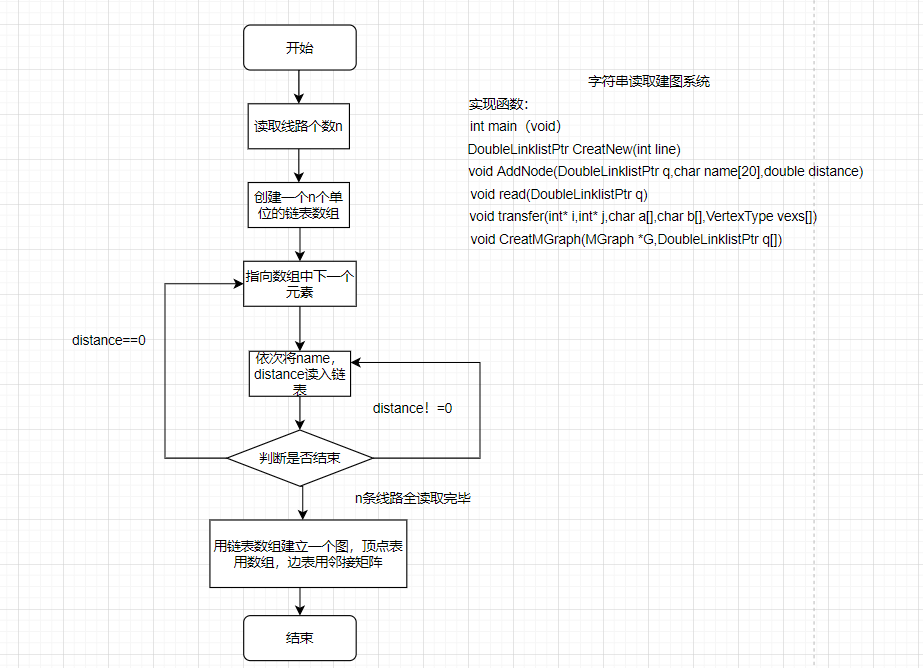


图3-1 字符串读取建图系统

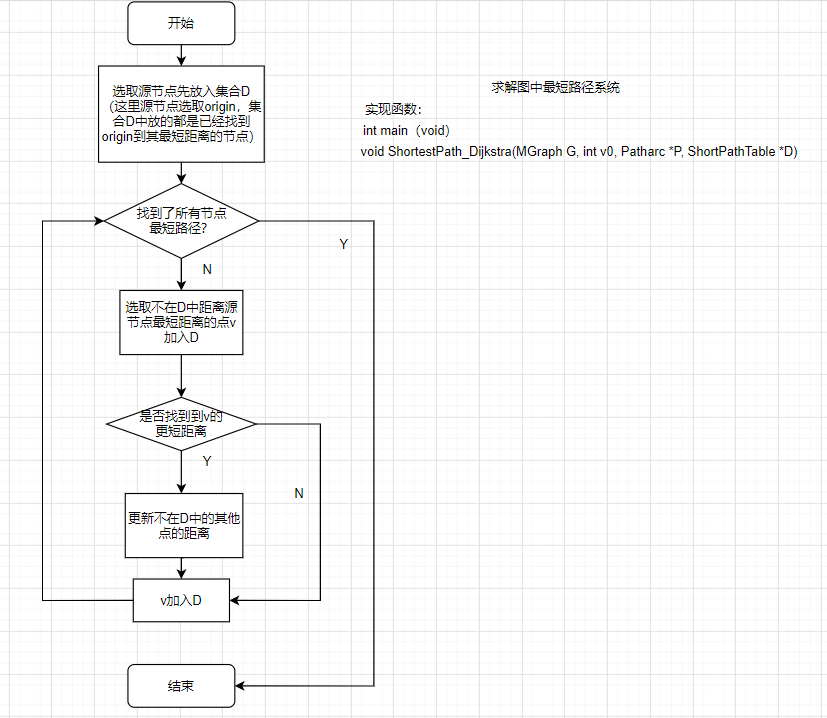


图3-2 求解图中最短路径系统

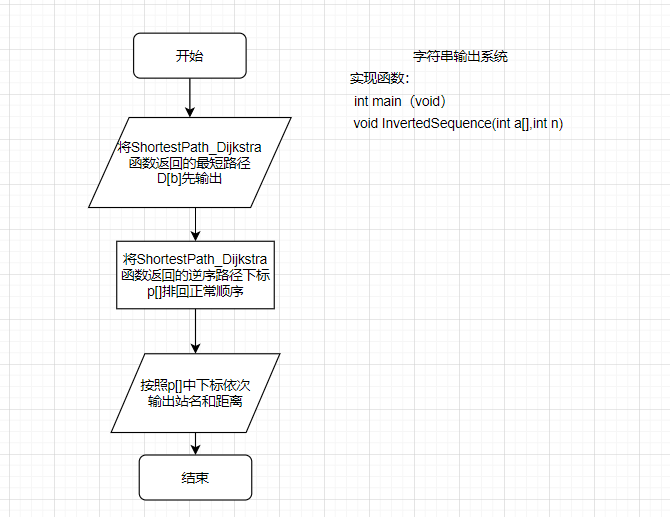


图3-3 字符串输出系统

## 3.5 系统实现

本程序全程在Microsoft Visual Studio 2022上编写、编译、调试、运行，并最终在Educoder平台上运行通过。

主要函数以及功能如表4-1所示。

表3-1 主要函数及功能

|  |  |
| --- | --- |
| 函数名 | 主要功能 |
| int main() | 实现读取地铁线路，并计算输出两个站点之间的最短路径的功能 |
| DoubleLinklistPtr CreatDoubleLinklist(int line) | 实现根据地铁线路号 ,创建一个空的双向循环链表的功能 |
| void AddNode(DoubleLinklistPtr q,char name[20],double distance) | 实现双向链表中加入节点的功能 |
| void read(DoubleLinklistPtr q) | 实现将站点信息读入的功能 |
| void transfer(int\* i,int\* j,char a[],char b[],VertexType vexs[]) | 实现找到站点在顶点表对应下标的功能 |
| void CreatMGraph(MGraph \*G,DoubleLinklistPtr q[]) | 实现将多个链表组成一张有向图的功能 |
| void ShortestPath\_Dijkstra(MGraph G, int v0, Patharc \*P, ShortPathTable \*D) | 利用狄克斯特拉算法，实现找到图中两个节点最短路径的功能 |
| void InvertedSequence(int a[],int n) | 实现将数组逆序的功能 |

## 3.6 系统测试

支持Educoder平台的所有可见测试用例与隐藏测试用例，均通过，如图3-4所示。



图3-4 测试通过

## 3.7 复杂度分析

设n为四条线路站点数之和

时间复杂度：

创建四个链表，将站点信息存入每个节点，再将节点建立图，建立图时，初始化整个邻接矩阵要遍历整个矩阵，故时间复杂度为。计算两个节点间最短路径时，使用的是迪杰斯特拉算法，用当前确定可达最短路来更新其他点，就是是点1到点x的距离是否比当前可达最短路点1到点t和点t到点x更远(有点像绕路比较)，如果更远则更新，每次迭代可确定一个可达最近点，然后将其标记，那么n个点n-1次迭代即可，因为n-1的点的最短路确定了，第n个点的最短路一定被前n-1次迭代所更新成最短路，故时间复杂度为。

输出时，根据迪杰斯特拉算法返回的下标逐一输出即可，复杂度为。综合输入、计算最短路径、输出三个步骤，时间复杂度为。

空间复杂度：  
 建立图时，包含图中包含顶点表（n）和邻接矩阵(n^2)，故空间复杂度为。

## 3.8 结果分析

成功通过给定测试用例，因oj平台测试案例只有一例，我在本地测试了其它案例如下图3-5，3-6，均通过，表明计算出到另一个站点的最短距离和途径的地铁站序列的需求成功实现。



图3-5 测试案例

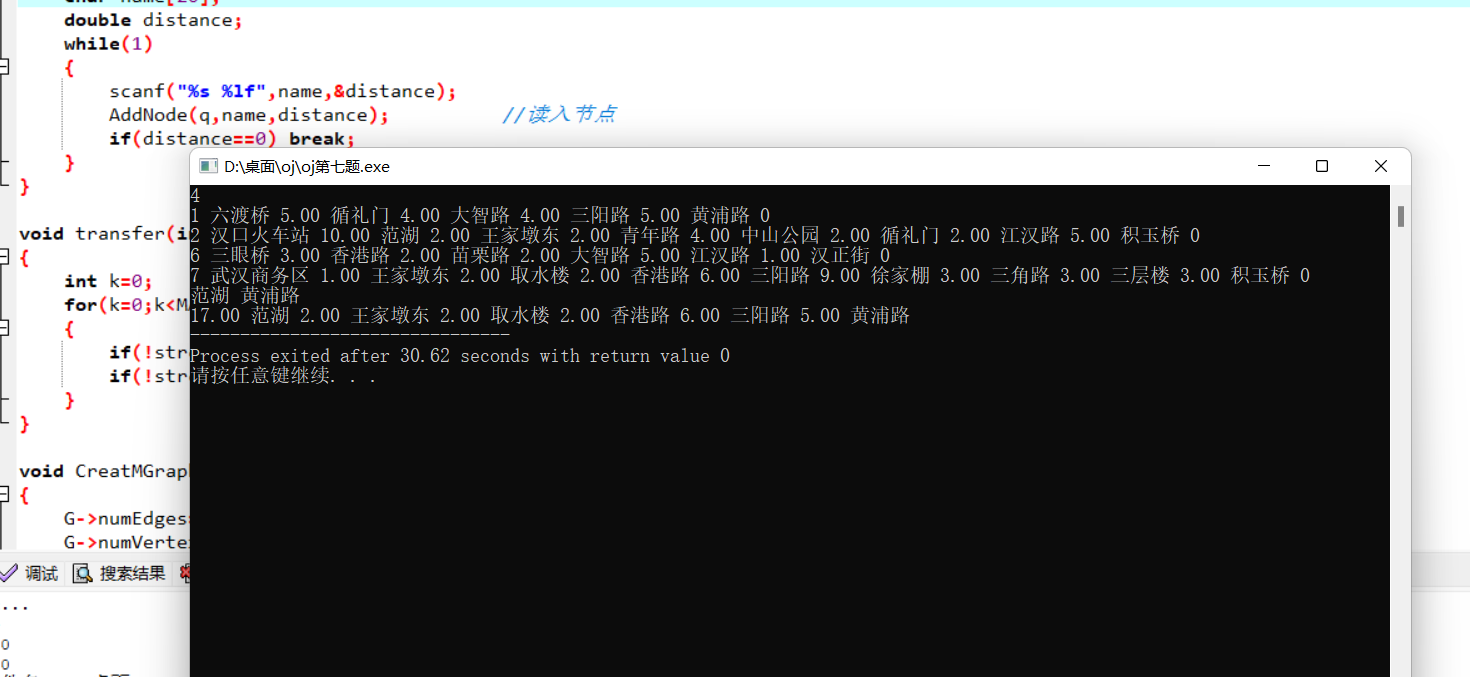


图3-6 测试案例

## 3.9 实验小结

通过本次实验，我最大的体会是：如何建立一个由顶点表和邻接矩阵组成的图，如何运用迪杰斯特拉算法求解最短路径。建立图的过程让我加深和熟悉了对图这个数据结构的理解，其实并没有什么思想难度。我还是想谈谈对迪杰斯特拉算法的理解，它的主要特点是以起始点为中心向外层层扩展，直到扩展到终点为止。本质上它采用的是贪心策略，将Graph中的节点集分为最短路径计算完成的节点集S和未计算完成的节点集T，每次将从T中挑选V0->Vt最小的节点Vt加入S，并更新V0经由Vt到T中剩余节点的更短距离，直到T中的节点全部加入S中，它贪心就贪心在每次都选择一个距离源点最近的节点加入最短路径节点集合。

# 参考文献

[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社

[2] [Larry Nyhoff](http://www.calvin.edu/~nyhl/index.html). [ADTs, Data Structures, and Problem Solving with C++.](http://vig.prenhall.com/catalog/academic/product/0,1144,0131409093,00.html)Second Edition, [Calvin College](http://cs.calvin.edu/), 2005

[3] 严蔚敏等.数据结构题集(C语言版). 清华大学出版社

# **附录** 基于链表的树、图实现的源程序

## 1.基于链表的建图

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//用四个链表来完成此关卡

typedef struct Node //双向链表节点

{

char name[20];

double distance;

struct Node\* pre;

struct Node\* next;

}Node,\*NodePtr;

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

DoubleLinklistPtr CreatNew(int line) //输入地铁线路号 ,创建一个空的双向链表

{

DoubleLinklistPtr q=(DoubleLinklistPtr)malloc(sizeof(DoubleLinklist));

q->line=line;

NodePtr head=(NodePtr)malloc(sizeof(Node));

head->distance=0;

head->next=NULL;

head->pre==NULL;

q->head=head;

q->end=head;

return q;

}

void AddNode(DoubleLinklistPtr q,char name[20],double distance) //双向链表中加入节点

{

NodePtr p=(NodePtr)malloc(sizeof(Node));

strcpy(p->name,name);

p->distance=distance;

q->end->next=p;

p->pre=q->end;

p->next=q->head;

q->head->pre=p;

q->end=p;

}

void read(DoubleLinklistPtr q) //将数据读入q

{

char name[20];

double distance;

while(1)

{

scanf("%s %lf",name,&distance);

AddNode(q,name,distance); //读入节点

if(distance==0) break;

}

}

void Display(DoubleLinklistPtr q) //输出数据

{

NodePtr head=q->head;

NodePtr temp=head->next;

printf("%d ",q->line);

while(temp!=head)

{

if(temp->next==head) //判断是否到表尾，若是则不输出距离0

{

printf("%s\n",temp->name);

break;

}

printf("%s %.2f ",temp->name,temp->distance);

temp=temp->next;

}

}

int main()

{

int n,line; //有n条线路,线路名字为line

scanf("%d",&n);

DoubleLinklistPtr q[n];

int i=0;

while(i<n)

{

scanf("%d",&line);

q[i]=CreatNew(line); //依次将n条线路读入n条双向链表中

read(q[i]);

i++;

}

i=0;

while(i<n)

{

Display(q[i]); //依次输出n条线路

i++;

}

return 0;

}

## 2.增加站点，删除

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

//用四个链表来完成此关卡

typedef struct Node //双向链表节点

{

char name[20];

double distance;

struct Node\* pre;

struct Node\* next;

}Node,\*NodePtr;

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

DoubleLinklistPtr CreatNew(int line) //输入地铁线路号 ,创建一个空的双向链表

{

DoubleLinklistPtr q=(DoubleLinklistPtr)malloc(sizeof(DoubleLinklist));

q->line=line;

NodePtr head=(NodePtr)malloc(sizeof(Node));

head->distance=0;

head->next=NULL;

head->pre==NULL;

q->head=head;

q->end=head;

return q;

}

void AddNode(DoubleLinklistPtr q,char name[20],double distance) //双向链表中加入节点

{

NodePtr p=(NodePtr)malloc(sizeof(Node));

strcpy(p->name,name);

p->distance=distance;

q->end->next=p;

p->pre=q->end;

p->next=q->head;

q->head->pre=p;

q->end=p;

}

void read(DoubleLinklistPtr q) //将数据读入q

{

char name[20];

double distance;

while(1)

{

scanf("%s %lf",name,&distance);

AddNode(q,name,distance); //读入节点

if(distance==0) break;

}

}

void Display(DoubleLinklistPtr q) //输出数据

{

NodePtr head=q->head;

NodePtr temp=head->next;

printf("%d ",q->line);

while(temp!=head)

{

if(temp->next==head) //判断是否到表尾，若是则不输出距离0

{

printf("%s\n",temp->name);

break;

}

printf("%s %.2f ",temp->name,temp->distance);

temp=temp->next;

}

}

DoubleLinklistPtr DeleteStation(DoubleLinklistPtr q,int\* flag)

{

DoubleLinklistPtr qNew=CreatNew(q->line);

char a[20];

\*flag=0; //判断是否删除了值 ,flag为0没有删除

scanf("%s",a);

NodePtr temp=q->head->next;

while(temp!=q->head)

{

if(!strcmp(temp->name,a)) //如果相等就遍历下一个，不加入到qNew中

{

temp=temp->next;

\*flag=1; //标志找到了要删除的站 ，flag为1删除成功

}

if(!strcmp(temp->next->name,a)) //如果下个站点要删去，则这个站点的距离需要修改

{

AddNode(qNew,temp->name,temp->distance+temp->next->distance);

}

else AddNode(qNew,temp->name,temp->distance);

temp=temp->next;

}

return qNew;

}

DoubleLinklistPtr AddStation(DoubleLinklistPtr q,int\* flag)

{

DoubleLinklistPtr qNew=CreatNew(q->line);

\*flag=2; //flag=0删除失败；flag=1删除成功或增加成功；flag=2；flag=3增加失败，已有同名站点

float a,b;

char c[20],d[20];

NodePtr temp=q->head->next;

scanf("%f%f",&a,&b);

if(a==0) //即可能在第一个位置加入站点

{

scanf("%s",c) ;//将可能加入的站点读入c

while(temp!=q->head)

{

if(!strcmp(temp->name,c))

{

\*flag=3; //增加失败，已有同名站点,将flag设置为3.直接返回

return qNew;

}

AddNode(qNew,temp->name,temp->distance);

temp=temp->next;

}

\*flag=1;

NodePtr p=(NodePtr)malloc(sizeof(Node)); //没有同名站点，开始插入操作 ,且将flag设置为1

strcpy(p->name,c);

p->distance=b;

temp=qNew->head->next;

p->next=temp;

p->pre=qNew->head;

temp->pre=p;

qNew->head->next=p; //flag=1,插入成功，正常返回

return qNew;

}

else

{

scanf("%s%s",c,d); //在c站点后加入可能添加的d站点

while(temp!=q->head)

{

if(!strcmp(temp->name,d))

{

\*flag=3; //增加失败，已有同名站点,将flag设置为3.直接返回

return qNew;

}

temp=temp->next;

}

temp=q->head->next;

while(temp!=q->head) //先将原路线复制到qNew中

{

AddNode(qNew,temp->name,temp->distance);

temp=temp->next;

}

temp=qNew->head->next;

while(temp!=qNew->head) //再在qNew中做插入操作

{

if(!strcmp(temp->name,c)) //找到了c

{

\*flag=1;

temp->distance=a;

NodePtr p=(NodePtr)malloc(sizeof(Node)); //开始插入操作

strcpy(p->name,d);

p->distance=b;

NodePtr temp2=temp->next; //即在temp和temp2中插入一个p

p->next=temp2;

p->pre=temp;

temp->next=p;

temp2->pre=p; //flag=1,插入成功，正常返回

return qNew;

}

temp=temp->next;

} //flag为2，没有找到c

return qNew;

}

}

void Display\_qNew(DoubleLinklistPtr qNew,int flag) //flag=0删除失败；flag=1删除成功或增加成功；

{ //flag=2，没有与输入的增加位置前一站点同名的站点，flag=3增加失败，已有同名站点

if(flag==0) printf("删除失败，没有同名站点");

if(flag==1) Display(qNew);

if(flag==2) printf("增加失败，没有与输入的增加位置前一站点同名的站点");

if(flag==3) printf("增加失败，已有同名站点");

}

int main()

{

int n,line,flag; //有n条线路,线路名字为line,输出的几种情况用flag标识

char pattern[20];

scanf("%d",&n);

DoubleLinklistPtr q[n];

int i=0;

while(i<n)

{

scanf("%d",&line);

q[i]=CreatNew(line); //依次将n条线路读入n条双向链表中

read(q[i]);

i++;

}

scanf("%s%d",pattern,&line);

DoubleLinklistPtr qNew=NULL; //用一个qNew链表保存被改动后的那条线路的结果

for(i=0;i<n;i++) //找到对应line的的数组，然后赋给qNew传入函数中

{

if(q[i]->line==line)

{

qNew=q[i];

break;

}

}

if(!strcmp(pattern,"add")) qNew=AddStation(qNew,&flag);

if(!strcmp(pattern,"delete")) qNew=DeleteStation(qNew,&flag);

i=0;

while(i<n)

{

Display(q[i]); //依次输出n条线路

i++;

}

Display\_qNew(qNew,flag); //输出新的修改后的线路

return 0;

}

## 3.从指定站点出发，计算到另一站点的最短距离和途径站点序列

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAXVEX 21

#define INFINITY 65535

//用四个链表来完成此关卡

typedef struct Node //双向链表节点

{

char name[20];

double distance;

struct Node\* pre;

struct Node\* next;

}Node,\*NodePtr;

typedef struct DoubleLinklist //双向循环链表

{

int line; //地铁线路号

NodePtr head;

NodePtr end;

}DoubleLinklist,\*DoubleLinklistPtr;

typedef char VertexType[20]; //顶点类型为字符串，因为要传入站名

typedef float EdgeType;

typedef struct //用邻接矩阵建立图

{

VertexType vexs[MAXVEX];

EdgeType arc[MAXVEX][MAXVEX];

int numVertexes,numEdges;

}MGraph;

typedef int Patharc[MAXVEX]; //用于存储最短路径下标的数组

typedef float ShortPathTable[MAXVEX]; //用于存储到各点最短路径的权值和

DoubleLinklistPtr CreatDoubleLinklist(int line) //输入地铁线路号 ,创建一个空的双向链表

{

DoubleLinklistPtr q=(DoubleLinklistPtr)malloc(sizeof(DoubleLinklist));

q->line=line;

NodePtr head=(NodePtr)malloc(sizeof(Node));

head->distance=0;

head->next=NULL;

head->pre==NULL;

q->head=head;

q->end=head;

return q;

}

void AddNode(DoubleLinklistPtr q,char name[20],double distance) //双向链表中加入节点

{

NodePtr p=(NodePtr)malloc(sizeof(Node));

strcpy(p->name,name);

p->distance=distance;

q->end->next=p;

p->pre=q->end;

p->next=q->head;

q->head->pre=p;

q->end=p;

}

void read(DoubleLinklistPtr q) //将数据读入q

{

char name[20];

double distance;

while(1)

{

scanf("%s %lf",name,&distance);

AddNode(q,name,distance); //读入节点

if(distance==0) break;

}

}

void transfer(int\* i,int\* j,char a[],char b[],VertexType vexs[]) //将a，b在顶点表中的下标找到，并以i，j返回

{

int k=0;

for(k=0;k<MAXVEX;k++)

{

if(!strcmp(a,vexs[k])) \*i=k;

if(!strcmp(b,vexs[k])) \*j=k;

}

}

void CreatMGraph(MGraph \*G,DoubleLinklistPtr q[]) //将四个链表创立一个图

{

G->numEdges=0;

G->numVertexes=0;

int i,j,k;

int flag=1; //flag=0则表示之前有这个站了，不用加入顶点表

//建立顶点表

for(i=0;i<MAXVEX;)

{

for(j=0;j<4;j++) //依次将四个链表读入

{

NodePtr head=q[j]->head;

NodePtr temp=q[j]->head->next;

while(temp!=head)

{

for(k=0;k<G->numVertexes;k++)

{

if(!strcmp(G->vexs[k],temp->name))

{

flag=0;

break;

}

}

if(flag==1) //之前没有相等的站

{

strcpy(G->vexs[i],temp->name);

G->numVertexes++;

i++;

}

temp=temp->next;

flag=1;

}

}

}

//初始化邻接矩阵

for(i=0;i<G->numVertexes;i++)

{

for(j=0;j<G->numVertexes;j++)

{

G->arc[i][j]=INFINITY;

}

}

//建立邻接矩阵

for(k=0;k<4;k++) //将四个链表的边是关系输入邻接矩阵中

{

NodePtr head=q[k]->head;

NodePtr temp=q[k]->head->next;

while(temp->next!=head)

{

transfer(&i,&j,temp->name,temp->next->name,G->vexs);

G->arc[i][j]=temp->distance;

G->arc[j][i]=temp->distance;

G->numEdges++;

temp=temp->next;

}

}

}

void ShortestPath\_Dijkstra(MGraph G, int v0, Patharc \*P, ShortPathTable \*D)

{

int v,w,k,min;

int final[MAXVEX]; // final[w]=1表示求得顶点v0至vw的最短路径

for(v=0; v<G.numVertexes; v++) //初始化数据

{

final[v] = 0; //全部顶点初始化为未知最短路径状态

(\*D)[v] = G.arc[v0][v]; //将与v0点有连线的顶点加上权值

(\*P)[v] = -1; //初始化路径数组P为-1

}

(\*D)[v0] = 0; //v0至v0路径为0

final[v0] = 1; // v0至v0不需要求路径

//开始主循环，每次求得v0到某个v顶点的最短路径

for(v=1; v<G.numVertexes; v++)

{

min=INFINITY; //当前所知离v0顶点的最近距离

for(w=0; w<G.numVertexes; w++) //寻找离v0最近的顶点

{

if(!final[w] && (\*D)[w]<min)

{

k=w;

min = (\*D)[w]; //w顶点离v0顶点更近

}

}

final[k] = 1; //将目前找到的最近的顶点置为1

for(w=0; w<G.numVertexes; w++) // 修正当前最短路径及距离

{

//如果经过v顶点的路径比现在这条路径的长度短的话

if(!final[w] && (min+G.arc[k][w]<(\*D)[w]))

{ // 说明找到了更短的路径，修改D[w]和P[w]

(\*D)[w] = min + G.arc[k][w]; //修改当前路径长度

(\*P)[w]=k;

}

}

}

}

//Dijkstra算法最后得到的结果为：

//顶点a（下标）到其余顶点v的最短路径P[v]以及带权长度D[v]

// P[v]的值为前驱顶点下标，D[v]表示a到v的最短路径之和

void InvertedSequence(int a[],int n) //将有n个元素的数组逆序排列

{

int temp=0;

int i=0;

for(;i<n-1;i++,n--)

{

temp=a[i];

a[i]=a[n-1];

a[n-1]=temp;

}

}

int main()

{

int n,line; //有n条线路,线路名字为line

scanf("%d",&n);

DoubleLinklistPtr q[n];

int i=0; //i就起一个计数的作用

while(i<n)

{

scanf("%d",&line);

q[i]=CreatDoubleLinklist(line); //依次将n条线路读入n条双向链表中

read(q[i]);

i++;

}

char origin[20],terminal[20]; //起点和终点

scanf("%s%s",origin,terminal);

MGraph G;

CreatMGraph(&G,q);

int a,b; //用a，b接受起始和终点对应的数组下标

Patharc P; //用于存储最短路径下标的数组

ShortPathTable D; //用于存储到各点最短路径的权值和

transfer(&a,&b,origin,terminal,G.vexs);

ShortestPath\_Dijkstra(G,a,&P,&D);

printf("%.2f ",D[b]); //将到b的总路程输出

int m=0;

int b\_temp1=b; //b的值需要保存下来，所以后面两个要用到b的地方，分别用b\_temp1和b\_temp2

int b\_temp2=b;

while(P[b\_temp1]!=-1) //判断中间经过了几个站点 ,记录在m中

{

m++;

b\_temp1=P[b\_temp1];

}

int index[m];

i=0;

while(P[b\_temp2]!=-1) //将i个站点的index逆序存入数组中

{

index[i]=P[b\_temp2];

b\_temp2=P[b\_temp2];

i++;

}

InvertedSequence(index,m); //将数组逆序，变成正常的顺序

printf("%s %.2f ",origin,G.arc[a][index[0]]);

for(i=0;i<m-1;i++)

{

printf("%s %.2f ",G.vexs[index[i]],G.arc[index[i]][index[i+1]]);

}

printf("%s %.2f ",G.vexs[index[m-1]],G.arc[index[m-1]][b]);

printf("%s",terminal);

return 0;

}