

**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：\_\_ 网安本硕博2101班\_\_\_\_**

**学 号：\_\_ U202112209\_\_\_ \_\_**

**姓 名：\_\_\_\_\_\_\_\_吴泽皓\_\_\_\_\_\_\_\_\_**

**指导教师：\_\_\_\_\_\_\_\_马晓静 \_\_\_\_\_\_\_\_**

**报告日期：\_\_\_2022年11月19日\_\_\_\_**

**网络空间安全学院**

**目 录**

[**1 基于链表的建图** **1**](#_Toc119868395)

[1.1 需求分析 1](#_Toc119868396)

[1.2 总体设计 1](#_Toc119868397)

[1.3 数据结构 1](#_Toc119868398)

[1.4 算法设计 2](#_Toc119868399)

[1.5 系统实现 4](#_Toc119868400)

[1.6 系统测试 5](#_Toc119868401)

[1.7 复杂度分析 6](#_Toc119868402)

[1.8 结果分析 6](#_Toc119868403)

[1.9 实验小结 6](#_Toc119868404)

[**2 增加站点，删除 7**](#_Toc119868405)

[2.1 需求分析 7](#_Toc119868406)

[2.2 总体设计 7](#_Toc119868407)

[2.3 数据结构 7](#_Toc119868408)

[2.4 算法设计 8](#_Toc119868409)

[2.5 系统实现 9](#_Toc119868410)

[2.6 系统测试 10](#_Toc119868411)

[2.7 复杂度分析 10](#_Toc119868412)

[2.8 结果分析 11](#_Toc119868413)

[2.9实验小结 11](#_Toc119868414)

[**3 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 12**](#_Toc119868415)

[3.1 需求分析 12](#_Toc119868416)

[3.2 总体设计 13](#_Toc119868417)

[3.3 数据结构 13](#_Toc119868418)

[3.4 算法设计 13](#_Toc119868419)

[3.5 系统实现 15](#_Toc119868420)

[3.6 系统测试 16](#_Toc119868421)

[3.7 复杂度分析 16](#_Toc119868422)

[3.8 结果分析 17](#_Toc119868423)

[3.9 实验小结 17](#_Toc119868424)

[**参考文献 18**](#_Toc119868425)

[**[1] 严蔚敏等. 数据结构(C语言版). 清华大学出版社 18**](#_Toc119868426)

[**附录 基于链表的树、图实现的源程序 19**](#_Toc119868427)

# 1 基于链表的建图

## 1.1 需求分析

### 1.1.1 功能需求

### 本关任务：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

根据提示，在右侧编辑器补充代码，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

整个程序分为读取数据，建立图建构，按顺序输出图三个部分。

## 1.3 数据结构

struct Edgenode{

int adj;

float len;

int seq;

struct Edgenode \*next;

}edge;

struct Vertexnode{

char name[20];

edge \*first;

}vertex, list[50];

struct{

list L;

int numvex, numedge;

}graph;

分别创建边节点，顶点节点和图结构，其中，边界点包含下一个顶点编号，路径长度，路径编号和由该起点衍生的下一个边节点。

## 1.4 算法设计

读取数据流程，建立图流程，按顺序输出图流程分别如图1-1，1-2，1-3所示。



图1-1 读取数据流程

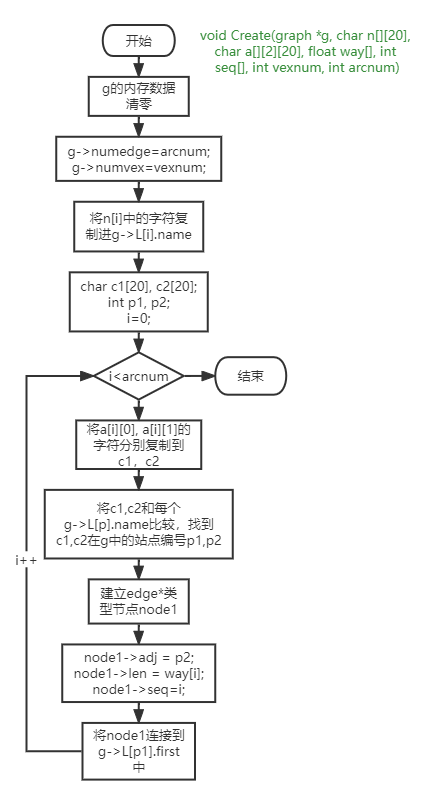


图1-2 建立图流程

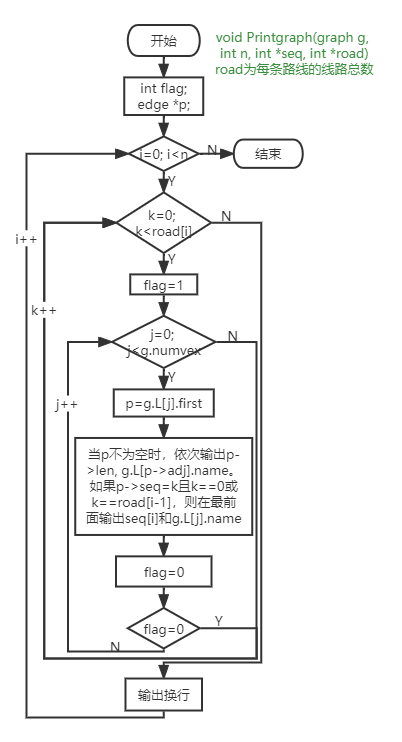


图1-3 按顺序输出图流程

## 1.5 系统实现

本程序全程在VScode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() | 读取输入的数据，转化为便于处理的方式 |
| void Create(graph \*g, char n[][20], char a[][2][20], float way[], int seq[], int vexnum, int arcnum) | 创建含有站名，站间距，线路编号的图 |
| void Printgraph(graph g, int n, int \*seq, int \*road) | 按顺序输出该图 |
| int get\_position(graph \*g, char\* c) | 查询c字符串在g中的编号 |
| int compare(char a[][20], char b[]) | 判断a中是否有和b字符串重合的元素 |

## 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1-4 通过所有测试

## 1.7 复杂度分析

读取数据并转换为便于处理的数组时间复杂度为或者，n、n’分别是站点数量和路线数量。

创建图需要查找节点对应的站点编号，故时间复杂度为。

数据存储的空间复杂度为，n’为路线数量。

## 1.8 结果分析

成功通过所有的给定测试用例，表明该链式循环队列操作程序设计成功实现使用需求。

## 1.9 实验小结

本次实验难点在于创建图结构，经过查阅资料和相关书籍后，成功创建图建构。此外，读取数据转化为便于处理的数据也是难点之一，我将读取的数据分为了三类，一类是单纯的站名，一类是所有路径的长度，一类是路径的起点站和终点站，将这三个数组导入函数创建图。

# 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

根据提示，在右侧编辑器补充代码，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

本程序在上一个程序基础上添加了读取输入的需要增加或删除站点的信息和增加、删除部分和删除一条边的函数。

## 2.3 数据结构

struct Edgenode{

int adj;

float len;

int seq;

struct Edgenode \*next;

}edge;

struct Vertexnode{

char name[20];

edge \*first;

}vertex, list[50];

struct{

list L;

int numvex, numedge;

}graph;

分别创建边节点，顶点节点和图结构，其中，边界点包含下一个顶点编号，路径长度，路径编号和由该起点衍生的下一个边节点。

## 2.4 算法设计

删除一条边流程，插入站点流程分别如图2-1，2-2所示。

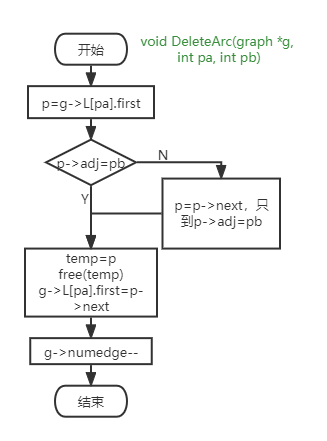


图2-1 删除一条边流程

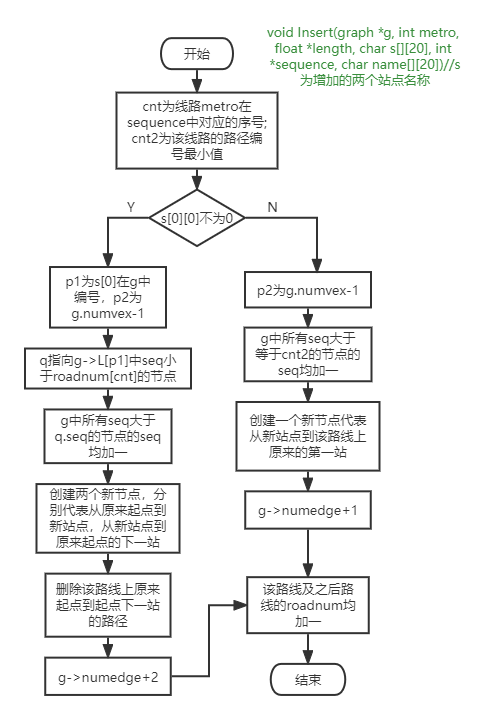


图2-2 插入站点流程

删除一个站点的思路和添加站点相似，将对应数据加一改为减一即可。

## 2.5 系统实现

本程序全程在VSCode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() | 读取新输入的增加/删除站点数据 |
| void DeleteArc(graph \*g, int pa, int pb) | 删除g中起点编号pa，终点编号pb的边 |
| void Insert(graph \*g, int metro, float \*length, char s[][20], int \*sequence, char name[][20]) | 插入一个站点 |
| void Delete(graph \*g, int metro, char \*s, int \*sequence) | 删除一个站点 |

## 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2-3所示。



图2-3 测试通过

## 2.7 复杂度分析

读取数据并转换为便于处理的数组时间复杂度为或者，n、n’分别是站点数量和路线数量。

创建图需要查找节点对应的站点编号，故时间复杂度为。

增加和删除站点时只需要将图按照站点顺序遍历一边。故程序总时间复杂度为。

数据存储的空间复杂度为，n’为路线数量。

## 2.8 结果分析

成功通过所有的给定测试用例，表明该链式栈操作程序设计成功实现使用需求。

## 2.9实验小结

本次实验耗时较长，原因为对插入站点的过程理解不透彻。事实上，插入一个站点不仅意味着要加入两条新的路径，还要删掉原有的路径。因此对应的路程编号，站点数目，路径数目均需对应更改。而且插入和删除站点考虑的要素较多，无论是输入还是输出都需要考虑是否是在一条线路的开头插入或删除。

# 3 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

## 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

根据提示，在右侧编辑器补充代码，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

参考信息：

1号线 六渡桥 5.00 循礼门 4.00 大智路 4.00 三阳路 5.00 黄浦路 0

2号线 汉口火车站 10.00 范湖 2.00 王家墩东 2.00 青年路 4.00 中山公园 2.00 循礼门 2.00 江汉路 5.00 积玉桥 0

6号线 三眼桥 3.00 香港路 2.00 苗栗路 2.00 大智路 5.00 江汉路 1.00 汉正街 0

7号线 武汉商务区 1.00 王家墩东 2.00 取水楼 2.00 香港路 6.00 三阳路 9.00 徐家棚 3.00 三角路 3.00 三层楼 3.00 积玉桥 0

## 3.2 总体设计

该程序在第一题程序的基础上，添加了查询路径函数和寻找节点的下一个节点的函数。

利用visited数组，path\_record数组，path\_next数组来记录是否经过该路径。

## 3.3 数据结构

struct Edgenode{

int adj;

float len;

int seq;

struct Edgenode \*next;

}edge;

struct Vertexnode{

char name[20];

edge \*first;

}vertex, list[50];

struct{

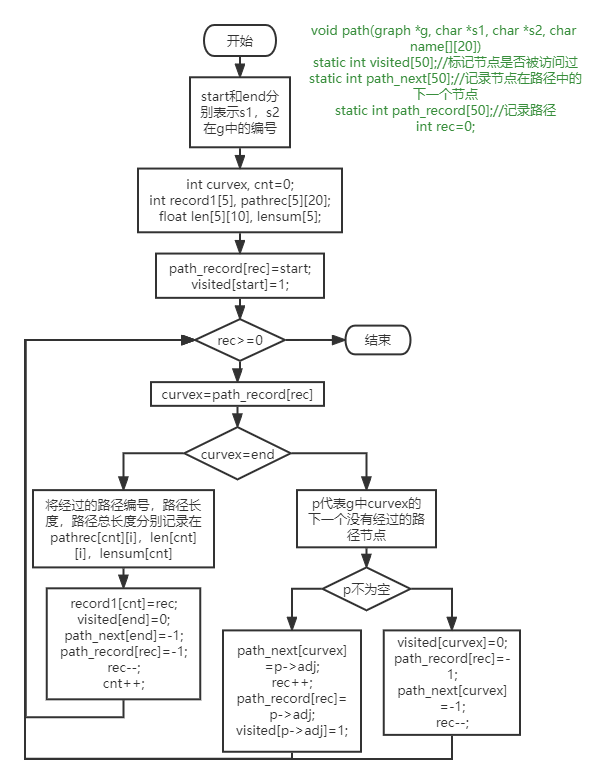
list L;

int numvex, numedge;

}graph;

分别创建边节点，顶点节点和图结构，其中，边界点包含下一个顶点编号，路径长度，路径编号和由该起点衍生的下一个边节点。

## 3.4 算法设计

查询路径流程，查询节点的下一个节点流程分别如图3-1，3-2所示。 

Y

Y

Y

N

N

N

Y

Y

Y

图3-1 查询路径流程

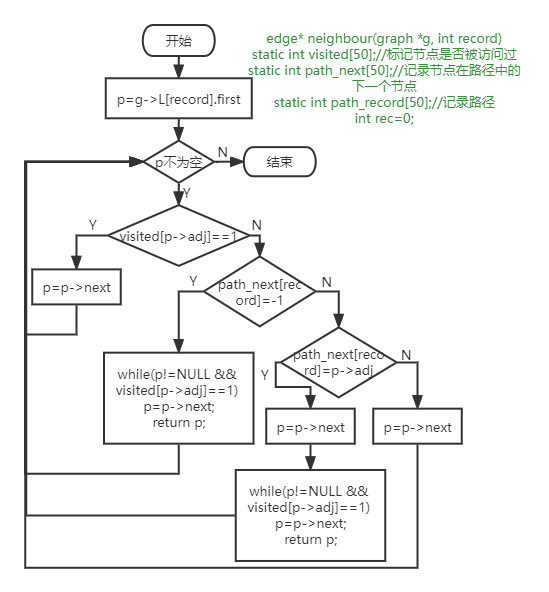


图3-2 查询下一个节点路径流程

## 3.5 系统实现

本程序全程在VSCode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3-1所示。

表3-1 主要函数及功能

|  |  |
| --- | --- |
| 函数名 | 主要功能 |
| int main() | 读取新输入的起点终点数据，输出题目要求内容 |
| edge\* neighbour(graph \*g, int record) | 查询节点的下一个没有被经过的路径 |
| void path(graph \*g, char \*s1, char \*s2, char name[][20]) | 记录所有能从起点到达终点的路径 |
| int MIN(float \*a) | 找到所有路径中长度最短的路径，返回序号 |

## 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3-3所示。



图3-3 测试通过

## 3.7 复杂度分析

读取数据并转换为便于处理的数组时间复杂度为或者，n、n’分别是站点数量和路线数量。

创建图需要查找节点对应的站点编号，时间复杂度为。

查询所有能从起点到达终点的路径的时间复杂度为，n为节点数目。

数据存储的空间复杂度为，n’为路线数量。

## 3.8 结果分析

成功通过所有的给定测试用例，表明该一元稀疏多项式加减运算器成功实现使用需求。

## 3.9 实验小结

本次实验运用了寻找图的最短路径算法，查询课本和相关资料后，顺利完成本次实验，对该算法有了更深入的理解，对其中的细节有了更加细致的掌握。

# 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**

第一题：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct Edgenode{

int adj;

float len;

int seq;

struct Edgenode \*next;

}edge;

typedef struct Vertexnode{

char name[20];

edge \*first;

}vertex, list[50];

typedef struct{

list L;

int numvex, numedge;

}graph;

int compare(char a[][20], char b[]){

int len=0;

while(a[len][0] != 0) len++;

for(int i=0; i<len; i++){

if(!(strcmp(a[i], b))){

return 0;

}

}

return 1;

}

int get\_position(graph \*g, char\* c){

int i;

for(i=0; i<g->numvex; i++)

if(strcmp(g->L[i].name, c) == 0) return i;

return -1;

}

void Create(graph \*g, char n[][20], char a[][2][20], float way[], int seq[], int vexnum, int arcnum){

int i;

memset(g, 0, sizeof(graph));

g->numedge = arcnum;

g->numvex = vexnum;

for(i=0; i<vexnum; i++){

strcpy(g->L[i].name, n[i]);

g->L[i].first=NULL;

}

char c1[20], c2[20];

int p1, p2;

for(i=0; i<arcnum; i++){

strcpy(c1, a[i][0]);

strcpy(c2, a[i][1]);

p1=get\_position(g, c1);

p2=get\_position(g, c2);

edge \*node1=(edge\*)malloc(sizeof(edge\*));

node1->adj = p2;

node1->len = way[i];

node1->seq=i;

node1->next=g->L[p1].first;

g->L[p1].first=node1;

}

}

void Printgraph(graph g, int n, int \*seq, int \*road){

int i, j, k=0;

int flag;

edge \*p;

// for(i=0; p->next!=NULL; i++){

// printf("%d\n", p->seq);

// }

for(i=0; i<n; i++){

for(; k<road[i]; k++){

flag=1;

for(j=0; j<g.numvex; j++){

p=g.L[j].first;

for(; p!=NULL; p=p->next){

if(p->seq==k && (k==0||k==road[i-1])){

printf("%d %s %.2f %s", seq[i], g.L[j].name, p->len, g.L[p->adj].name);

flag=0;

break;

}

else if(p->seq==k){

printf(" %.2f %s", p->len, g.L[p->adj].name);

flag=0;

break;

}

}

if(flag==0) break;

}

}

printf("\n");

}

}

int main(){

int n, route;

scanf("%d", &n);

float len;

char c;

int i, j, k;

int seq[n], roadnum[4]={4, 11, 16, 24};

char tmpname[30][20]={0}, name[30][20]={0};

char line[50][2][20]={0};

float tmplength[n][20], length[50]={0};

int cnt1=0, cnt2=0, cnt3=0;

for(i=0; i<n; i++){

scanf("%d ", &route);

seq[i]=route;

len=1;

for(k=0; len!=0; k++){

if(c==' ' && k!=0) getchar();

for(j=0; (c=getchar()) != ' ';j++) tmpname[k][j]=c;

scanf("%f", &len);

tmplength[i][k]=len;

if(len==0) break;

}

for(j=0; tmplength[i][j]!=0; j++){

strcpy(line[cnt1][0], tmpname[j]);

strcpy(line[cnt1][1], tmpname[j+1]);

cnt1++;

}

j=0;

do{

if(compare(name, tmpname[j])==1) strcpy(name[cnt2++], tmpname[j]);

}while(tmplength[i][j++]!=0);

for(j=0; tmplength[i][j]!=0; j++) length[cnt3++] = tmplength[i][j];

memset(tmpname, 0, sizeof(tmpname));

}

int arcnum=cnt3, vexnum=cnt2;

graph g;

Create(&g, name, line, length, seq, vexnum, arcnum);

Printgraph(g, n, seq, roadnum);

return 0;

}

第二题：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct Edgenode{

int adj;

float len;

int seq;

struct Edgenode \*next;

}edge;

typedef struct Vertexnode{

char name[20];

edge \*first;

}vertex, list[50];

typedef struct{

list L;

int numvex, numedge;

}graph;

static int roadnum[4]={4, 11, 16, 24};

int compare(char a[][20], char b[]){

int len=0;

while(a[len][0] != 0) len++;

for(int i=0; i<len; i++){

if(!(strcmp(a[i], b))){

return 0;

}

}

return 1;

}

int get\_position(graph \*g, char\* c){

int i;

for(i=0; i<g->numvex; i++)

if(strcmp(g->L[i].name, c) == 0) return i;

return -1;

}

void Create(graph \*g, char n[][20], char a[][2][20], float way[], int seq[], int vexnum, int arcnum){

int i;

memset(g, 0, sizeof(graph));

g->numedge = arcnum;

g->numvex = vexnum;

for(i=0; i<vexnum; i++){

strcpy(g->L[i].name, n[i]);

g->L[i].first=NULL;

}

char c1[20], c2[20];

int p1, p2;

for(i=0; i<arcnum; i++){

strcpy(c1, a[i][0]);

strcpy(c2, a[i][1]);

p1=get\_position(g, c1);

p2=get\_position(g, c2);

edge \*node1=(edge\*)malloc(sizeof(edge\*));

node1->adj = p2;

node1->len = way[i];

node1->seq=i;

node1->next=g->L[p1].first;

g->L[p1].first=node1;

}

}

void DeleteArc(graph \*g, int pa, int pb){

edge \*p, \*temp;

p=g->L[pa].first;

if(p->adj==pb){

temp=p;

free(temp);

g->L[pa].first=p->next;

}

while(p->next!=NULL){

if(p->next->adj==pb){

temp=p->next;

p->next=temp->next;

free(temp);

break;

}

p=p->next;

}

g->numedge--;

}

void Insert(graph \*g, int metro, float \*length, char s[][20], int \*sequence, char name[][20]){

int i, cnt, cnt2;

for(cnt=0; ;cnt++) if(metro==sequence[cnt]) break;

if(metro==1) cnt2=0;

else cnt2=roadnum[cnt-1];

int p1, p2, tmp;

edge \*node1, \*node2, \*p, \*q;

if(s[0][0]!=0){

p1=get\_position(g, s[0]);

p2=g->numvex-1;

q=g->L[p1].first;

if(q->seq >= roadnum[cnt]) q=q->next;

tmp=q->adj;

for(i=0; i<g->numvex; i++){

p=g->L[i].first;

for(; p!=NULL; p=p->next) if(p->seq > q->seq) p->seq+=1;

}

node1=(edge\*)malloc(sizeof(edge\*));

node1->adj=p2;

node1->len=length[0];

node1->seq=q->seq;

node1->next=g->L[p1].first;

g->L[p1].first=node1;

g->numedge++;

strcpy(g->L[p2].name, name[p2]);

node2=(edge\*)malloc(sizeof(edge\*));

node2->adj=tmp;

node2->len=length[1];

node2->seq=node1->seq+1;

node2->next=NULL;

g->L[p2].first=node2;

g->numedge++;

DeleteArc(g, p1, tmp);

}else{

p2=g->numvex-1;

for(i=0; i<g->numvex; i++){

p=g->L[i].first;

for(; p!=NULL; p=p->next){

if(p->seq == cnt2) tmp=i;

if(p->seq >= cnt2) p->seq+=1;

}

}

strcpy(g->L[p2].name, name[p2]);

node2=(edge\*)malloc(sizeof(edge\*));

node2->adj=tmp;

node2->len=length[1];

node2->seq=cnt2;

node2->next=NULL;

g->L[p2].first=node2;

g->numedge++;

}

for(i=cnt; i<4; i++) roadnum[i]++;

}

void Delete(graph \*g, int metro, char \*s, int \*sequence){

int i, cnt, cnt2;

for(cnt=0; ;cnt++) if(metro==sequence[cnt]) break;

if(metro==1) cnt2=0;

else cnt2=roadnum[cnt-1];

int p1, p2=50, adjtmp, lentmp1=0, lentmp2=0, seqtmp;

edge \*node1, \*p, \*q;

p1=get\_position(g, s);

p=g->L[p1].first;

for(; p!=NULL; p=p->next){

if(p->seq<roadnum[cnt] && p->seq>=cnt2){

adjtmp=p->adj;

lentmp1=p->len;

seqtmp=p->seq;

}

}

for(i=0; i<g->numvex; i++){

q=g->L[i].first;

for(; q!=NULL; q=q->next){

if(q->adj==p1 && q->seq<roadnum[cnt] && q->seq>=cnt2){

lentmp2=q->len;

p2=i;

}

}

if(p2==i) break;

}

for(i=0; i<g->numvex; i++){

q=g->L[i].first;

for(; q!=NULL; q=q->next) if(q->seq > seqtmp) q->seq--;

}

node1=(edge\*)malloc(sizeof(edge\*));

node1->adj=adjtmp;

node1->len=lentmp1+lentmp2;

node1->seq=seqtmp-1;

node1->next=g->L[p2].first;

g->L[p2].first=node1;

g->numedge++;

DeleteArc(g, p2, p1);

DeleteArc(g, p1, adjtmp);

for(i=cnt; i<4; i++) roadnum[i]--;

}

void PrintPart(graph \*g, int \*seq, int metro){

int i, j, cnt, cnt2;

for(cnt=0; ;cnt++) if(metro==seq[cnt]) break;

if(metro==1) cnt2=0;

else cnt2=roadnum[cnt-1];

edge \*p;

int flag;

for(i=cnt2; i<roadnum[cnt]; i++){

flag=1;

for(j=0; j<g->numvex; j++){

p=g->L[j].first;

for(; p!=NULL; p=p->next){

if(p->seq==i && i==cnt2){

printf("%d %s %.2f %s", metro, g->L[j].name, p->len, g->L[p->adj].name);

flag=0;

break;

}

else if(p->seq==i){

printf(" %.2f %s", p->len, g->L[p->adj].name);

flag=0;

break;

}

}

if(flag==0) break;

}

}

printf("\n");

}

void Printgraph(graph g, int n, int \*seq){

int i, j, k=0;

int flag;

edge \*p;

for(i=0; i<n; i++){

for(; k<roadnum[i]; k++){

flag=1;

for(j=0; j<g.numvex; j++){

p=g.L[j].first;

for(; p!=NULL; p=p->next){

if(p->seq==k && (k==0||k==roadnum[i-1])){

printf("%d %s %.2f %s", seq[i], g.L[j].name, p->len, g.L[p->adj].name);

flag=0;

break;

}

else if(p->seq==k){

printf(" %.2f %s", p->len, g.L[p->adj].name);

flag=0;

break;

}

}

if(flag==0) break;

}

}

printf("\n");

}

}

int main(){

int n, route;

scanf("%d", &n);

float len;

char c;

int i, j, k;

int seq[n];

char tmpname[30][20]={0}, name[30][20]={0};

char line[50][2][20]={0};

float tmplength[n][20], length[50]={0};

int cnt1=0, cnt2=0, cnt3=0;

for(i=0; i<n; i++){

scanf("%d ", &route);

seq[i]=route;

len=1;

for(k=0; len!=0; k++){

if(c==' ' && k!=0) getchar();

for(j=0; (c=getchar()) != ' ';j++) tmpname[k][j]=c;

scanf("%f", &len);

tmplength[i][k]=len;

if(len==0) break;

}

for(j=0; tmplength[i][j]!=0; j++){

strcpy(line[cnt1][0], tmpname[j]);

strcpy(line[cnt1][1], tmpname[j+1]);

cnt1++;

}

j=0;

do{

if(compare(name, tmpname[j])==1) strcpy(name[cnt2++], tmpname[j]);

}while(tmplength[i][j++]!=0);

for(j=0; tmplength[i][j]!=0; j++) length[cnt3++] = tmplength[i][j];

memset(tmpname, 0, sizeof(tmpname));

}

int arcnum=cnt3, vexnum=cnt2;

graph g;

Create(&g, name, line, length, seq, vexnum, arcnum);

Printgraph(g, n, seq);

char jg[10]={0};

int metro, flag;

float length2[2]={0};

char sta[2][20]={0};

scanf("%s", jg);

if(!strcmp(jg, "add")){

scanf("%d%f%f ", &metro, &length2[0], &length2[1]);

if(length2[0]==0) scanf("%s", sta[1]);

else{

scanf("%s", sta[0]);

scanf("%s", sta[1]);

}

}else{

scanf("%d ", &metro);

scanf("%s", sta[0]);

}

for(i=0; name[i][0]!=0; i++){

if(sta[0][0]==0) flag=0;

if(!strcmp(sta[0], name[i])) flag=0;

}

if(flag != 0){

if(!strcmp(jg, "add")) printf("增加失败，没有与输入的增加位置前一站点同名的站点\n");

else printf("删除失败，没有同名站点\n");

}else{

if(!strcmp(jg, "add")){

flag=1;

for(i=0; name[i][0]!=0; i++){

if(!strcmp(sta[1], name[i])) flag=0;

}

if(flag==0) printf("增加失败，已有同名站点\n");

else{

strcpy(name[cnt2], sta[1]);

cnt2++;

g.numvex++;

Insert(&g, metro, length2, sta, seq, name);

PrintPart(&g, seq, metro);

}

}else{

Delete(&g, metro, sta[0], seq);

PrintPart(&g, seq, metro);

}

}

return 0;

}

第三题：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct Edgenode{

int adj;

float len;

int seq;

struct Edgenode \*next;

}edge;

typedef struct Vertexnode{

char name[20];

edge \*first;

}vertex, list[50];

typedef struct{

list L;

int numvex, numedge;

}graph;

static int visited[50];//标记节点是否被访问过

static int path\_next[50];//记录节点在路径中的下一个节点

static int path\_record[50];//记录路径

static int rec=0;

int compare(char a[][20], char b[]){

int len=0;

while(a[len][0] != 0) len++;

for(int i=0; i<len; i++){

if(!(strcmp(a[i], b))){

return 0;

}

}

return 1;

}

int get\_position(graph \*g, char\* c){

int i;

for(i=0; i<g->numvex; i++)

if(strcmp(g->L[i].name, c) == 0) return i;

return -1;

}

int MIN(float \*a){

int i, tmp2=0;

float tmp1=a[0];

for(i=0; a[i]!=0; i++){

if(a[i] < tmp1){

tmp1=a[i];

tmp2=i;

}

}

return tmp2;

}

void FreeGraph(graph \*g){

int i;

edge \*p, \*q;

for(i=0; i<g->numvex; i++){

p = g->L[i].first;

while(p){

q = p->next;

free(p);

p = q;

}

}

}

void Create(graph \*g, char n[][20], char a[][2][20], float way[], int seq[], int vexnum, int arcnum){

int i;

memset(g, 0, sizeof(graph));

g->numedge = arcnum;

g->numvex = vexnum;

for(i=0; i<vexnum; i++){

strcpy(g->L[i].name, n[i]);

g->L[i].first=NULL;

}

char c1[20], c2[20];

int p1, p2;

for(i=0; i<arcnum; i++){

strcpy(c1, a[i][0]);

strcpy(c2, a[i][1]);

p1=get\_position(g, c1);

p2=get\_position(g, c2);

edge \*node1=(edge\*)malloc(sizeof(edge\*));

node1->adj = p2;

node1->len = way[i];

node1->seq=i;

node1->next=g->L[p1].first;

g->L[p1].first=node1;

}

}

edge\* neighbour(graph \*g, int record){

edge \*p;

p=g->L[record].first;

while(p!=NULL){

if(visited[p->adj]==1) p=p->next;

else{

if(path\_next[record]==-1){

while(p!=NULL && visited[p->adj]==1) p=p->next;

return p;

}else if(path\_next[record] == p->adj){

p=p->next;

while(p!=NULL && visited[p->adj]==1) p=p->next;

return p;

}else p=p->next;

}

}

}

void path(graph \*g, char \*s1, char \*s2, char name[][20]){

int start, end;

start=get\_position(g, s1);

end=get\_position(g, s2);

int i, curvex, cnt=0;

int record1[5], pathrec[5][20];

float len[5][10], lensum[5];

edge \*p, \*q;

memset(lensum, 0, sizeof(lensum));

path\_record[rec]=start;

visited[start]=1;

while(rec>=0){

curvex=path\_record[rec];

if(curvex == end){

for(i=0; i<rec; i++){

q=g->L[path\_record[i]].first;

while(q->adj!=path\_record[i+1]) q=q->next;

len[cnt][i]=q->len;

}

for(i=0; i<rec; i++) lensum[cnt]+=len[cnt][i];

for(i=0; i<=rec; i++) pathrec[cnt][i]=path\_record[i];

record1[cnt]=rec;

visited[end]=0;

path\_next[end]=-1;

path\_record[rec]=-1;

rec--;

cnt++;

}else{

p=neighbour(g, curvex);

if(p!=NULL){

path\_next[curvex]=p->adj;

rec++;

path\_record[rec]=p->adj;

visited[p->adj]=1;

}else{

visited[curvex]=0;

path\_record[rec]=-1;

path\_next[curvex]=-1;

rec--;

}

}

}

int cnt2=MIN(lensum);

for(i=0; i<=record1[cnt2]; i++){

if(i == 0) printf("%.2f %s %.2f", lensum[cnt2], name[pathrec[cnt2][i]], len[cnt2][i]);

else if(i == record1[cnt2]) printf(" %s", name[pathrec[cnt2][i]]);

else printf(" %s %.2f", name[pathrec[cnt2][i]], len[cnt2][i]);

}

printf("\n");

}

int main(){

int n, route;

scanf("%d", &n);

float len;

char c;

int i, j, k;

int seq[n];

char tmpname[30][20]={0}, name[30][20]={0};

char line[50][2][20]={0};

float tmplength[n][20], length[50]={0};

int cnt1=0, cnt2=0, cnt3=0;

for(i=0; i<n; i++){

scanf("%d ", &route);

seq[i]=route;

len=1;

for(k=0; len!=0; k++){

if(c==' ' && k!=0) getchar();

for(j=0; (c=getchar()) != ' ';j++) tmpname[k][j]=c;

scanf("%f", &len);

tmplength[i][k]=len;

if(len==0) break;

}

for(j=0; tmplength[i][j]!=0; j++){

strcpy(line[cnt1][0], tmpname[j]);

strcpy(line[cnt1][1], tmpname[j+1]);

cnt1++;

}

j=0;

do{

if(compare(name, tmpname[j])==1) strcpy(name[cnt2++], tmpname[j]);

}while(tmplength[i][j++]!=0);

for(j=0; tmplength[i][j]!=0; j++) length[cnt3++] = tmplength[i][j];

memset(tmpname, 0, sizeof(tmpname));

}

int arcnum=cnt3, vexnum=cnt2;

graph g;

Create(&g, name, line, length, seq, vexnum, arcnum);

char start[20], end[20];

for(i=0; i<g.numvex; i++){

visited[i] = 0;

path\_next[i] = -1;

path\_record[i] = -1;

}

scanf("%s%s", start, end);

path(&g, start, end, name);

FreeGraph(&g);

return 0;

}