

**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：\_\_\_\_\_\_\_**密码2101**\_\_\_\_\_\_\_\_\_**

**学 号：\_\_\_\_\_\_\_**U202116003\_**\_\_\_\_\_\_**

**姓 名：\_\_\_\_\_\_\_**侯竣\_\_**\_\_\_\_\_\_\_\_\_\_\_**

**指导教师：\_\_\_\_\_\_**\_朱建新**\_\_\_\_\_\_\_\_\_\_\_**

**报告日期：\_\_\_\_\_**2022年11月11日**\_\_\_**

**网络空间安全学院**

**目 录**

**[1 基于链表的建图 1](#_Toc118025690)**

[1.1 需求分析 1](#_Toc118025691)

[1.2 总体设计 1](#_Toc118025692)

[1.3 数据结构 1](#_Toc118025693)

[1.4 算法设计 2](#_Toc118025694)

[1.5 系统实现 3](#_Toc118025695)

[1.6 系统测试 4](#_Toc118025696)

[1.7 复杂度分析 4](#_Toc118025697)

[1.8 结果分析 4](#_Toc118025698)

[1.9 实验小结 5](#_Toc118025699)

**[2 增加站点，删除 6](#_Toc118025700)**

[2.1 需求分析 6](#_Toc118025701)

[2.2 总体设计 6](#_Toc118025703)

[2.3 数据结构 6](#_Toc118025704)

[2.4 算法设计 7](#_Toc118025705)

[2.5 系统实现 10](#_Toc118025706)

[2.6 系统测试 10](#_Toc118025707)

[2.7 复杂度分析 11](#_Toc118025708)

[2.8 结果分析 11](#_Toc118025709)

[2.9实验小结 11](#_Toc118025710)

**[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](#_Toc118025711)**

[3.1 需求分析 12](#_Toc118025712)

[3.2 总体设计 12](#_Toc118025713)

[3.3 数据结构 13](#_Toc118025714)

[3.4 算法设计 14](#_Toc118025715)

[3.5 系统实现 15](#_Toc118025716)

[3.6 系统测试 16](#_Toc118025717)

[3.7 复杂度分析 16](#_Toc118025718)

[3.8 结果分析 17](#_Toc118025719)

[3.9 实验小结 17](#_Toc118025720)

**[参考文献 18](#_Toc118025731)**

**[附录 基于链式存储结构线性表实现的源程序 19](#_Toc118025733)**

# 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

读入线路号和站点信息, 以线路号为表头节点, 使用邻接表存储路线信息, 并输出

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

整个程序分为创建路线并初始化,读入路线和输出路线三个部分。

## 1.3 数据结构

typedef struct node {

char name[10]; // 站名

struct node \* next; //指向下一站

struct node \* pre; //指向上一站

double len; //到下一站的距离

}\* pnode;

typedef struct route {//线路

pnode head; //首站

pnode tail; //尾站

int num; //线路号s

}\*proute;

## 1.4 算法设计

主函数设计如图:

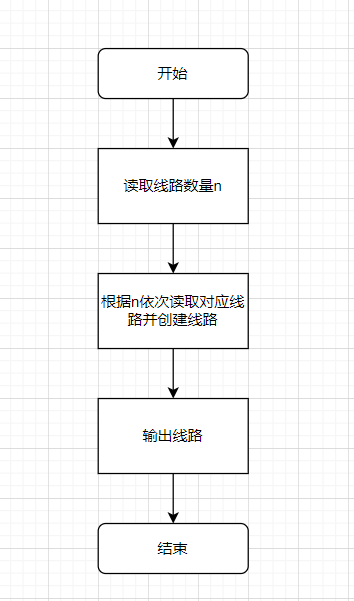


图1-1 主函数

CreateRoutes(int num)函数设计, num为线路条数

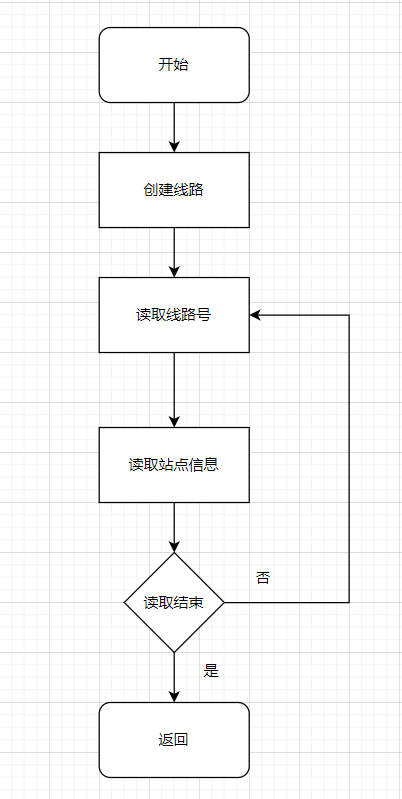


图1-2 CreateRoutes

## 1.5 系统实现

本程序全程在Visual Studio Code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() | 实现线路的读入 |
| proute CreateRoute(int num); | 完成route的初始化, num为线路号 |
| proute\* CreateRoutes(int num); | 读取输入信息, 并创建routes, num为线路条数 |
| void PrintAllRoute(proute\* p); | 按要求输出routes, p为线路指针, 指向四个route |

## 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1-2所示。



图1-3 通过所有测试

## 1.7 复杂度分析

读取线路只需要顺序一遍, 输出也只需要顺序操作一遍, 时间复杂度为O(n), n为站点数量

数据存储空间为站点数量的线性级别, 为O(n)。

## 1.8 结果分析

成功通过所有的给定测试用例，表明该设计成功实现使用需求。

## 1.9 实验小结

1. 需要设计好数据结构, 处理好输入输出关系.

2. 存储中文的问题, 中文占两个char, 需要注意存储空间分配.

# 2 增加站点，删除

## 2.1 需求分析

### 2.1.1 功能需求

本关任务：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

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

整个程序分为读取站点系统，输出站点系统, 增删站点系统三个部分.

读取站点系统和输出站点系统重用实验一.

增删站点系统首先读取操作, 然后对线路进行操作, 最后输出结果.

## 2.3 数据结构

与实验一相同

typedef struct node {

char \*name; // 站名

struct node \* next; //指向下一站

struct node \* pre; //指向上一站

double len; //到下一站的距离

}\* pnode;

typedef struct route {//线路

pnode head;

pnode tail;

int num; //线路号s

}\*proute;

## 2.4 算法设计

主函数设计

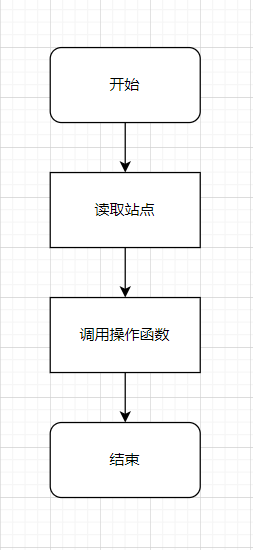


图2-1 主函数设计

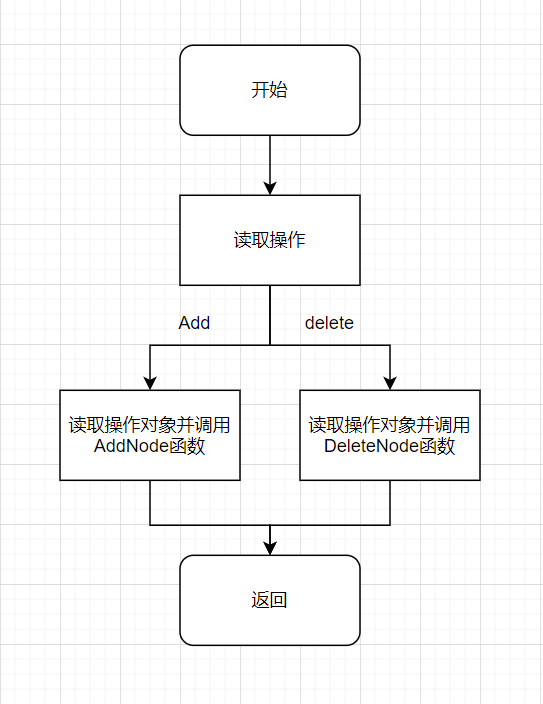


图2-2 操作函数Operate

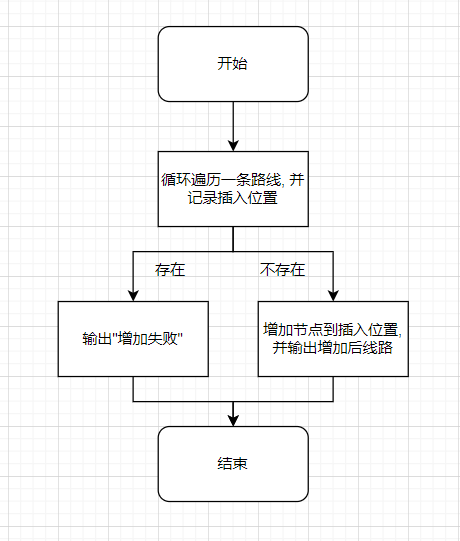


图2-3 增加节点函数AddNode

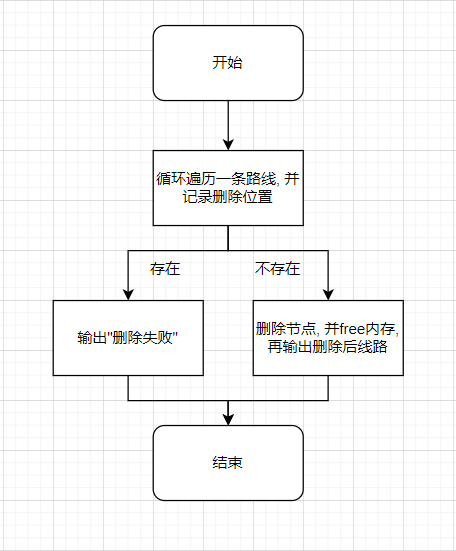


图2-4 删除节点函数 DeleteNode

## 2.5 系统实现

本程序全程在Visual Studio Code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() | 主函数 |
| proute CreateRoute(int num); | 完成route的初始化, num为线路号 |
| proute\* CreateRoutes(int num); | 读取输入信息, 并创建routes, num为线路条数 |
| void PrintAllRoute(proute\* p); | 按要求输出routes, p为线路指针, 指向四个route |
| proute AddNode(proute p, double len\_pre, double len\_next, char name\_pre[10], char name[10]); | 增加节点, 返回增加后的线路(proute) |
| proute DeleteNode(proute p, char \*name); | 删除节点 |
| int FromNumGetPos(proute\* p, int num); | 通过线路号得到线路存储的下标, 方便操作用 |
| PrintRoute(proute p); | 打印单条线路 |

## 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

读取线路只需要顺序一遍, 输出也只需要顺序操作一遍, 时间复杂度为O(n), n为站点数量, 之后读取操作, 进行线路的遍历并执行添加或者删除操作, 为O(n), 最后输出线路, 为O(n), 综合的时间复杂度为O(n)

空间复杂度, 只需要存储站点, 为O(n)

## 2.8 结果分析

成功通过所有的给定测试用例，表明该设计成功实现使用需求。

## 2.9实验小结

1. 对线路进行操作时, 线路号不等于存储的下标, 容易出错, 需要在线路中寻找一遍
2. 增加操作和删除操作可以用函数模块化编程, 并加入一个Operate函数, 读取操作, 实现不同功能之间的解耦

# 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

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

程序分为图操作系统和最短路径计算两大部分

图操作系统分为:创建图, 判断图某个节点是否存在, 增加图节点, 增加图边

最短路径计算使用了dijkstra算法

## 3.3 数据结构

typedef struct node {

char \*name; // 站名

struct node \* next; //指向下一站

struct node \* pre; //指向上一站

double len; //到下一站的距离

}\*pnode;

typedef struct route {//线路

pnode head;

pnode tail;

int num; //线路号s

double len; //线路长度

}\*proute;

typedef struct Edge {

char \*name; //被指向的顶点

double dis; //边权值

struct Edge\* next; //下一条边

}\*pEdge;

typedef struct {

char \*name;

pEdge next;

}Vertex; //顶点

typedef struct {

Vertex \*Vertex[VertexSize]; //顶点

int VertexNum; //顶点数量

int EdgeNum; //边数量

}Graph;

## 3.4 算法设计

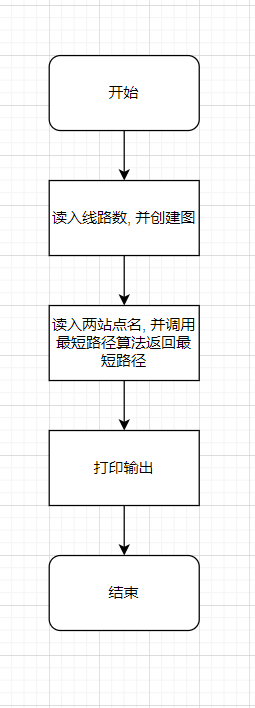


图3-1 主函数

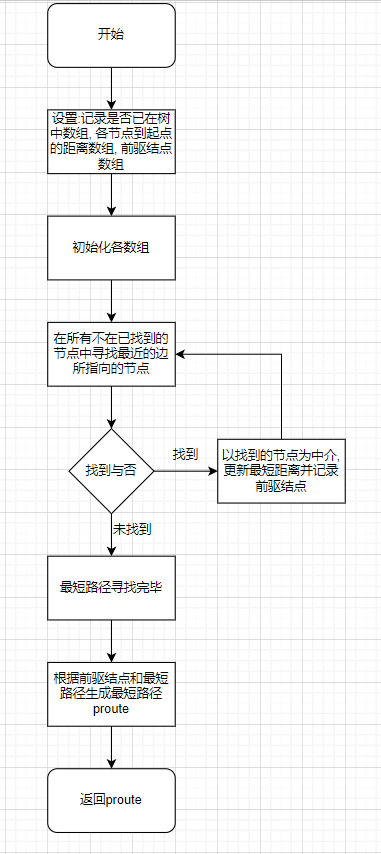


图3-2 dijkstra算法

## 3.5 系统实现

本程序全程在Visual Studio Code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() | 创建图, 调用最短路径算法, 输出路径 |
| void PrintRouteAndLen(proute p); | 按任务要求的格式输出最短路径 |
| void InsertVertex(char \*name); | 给图插入节点, 如果节点存在不会插入 |
| void InsertEdge(char \*start, char\* end, double dis); | 给图插入边(单向), 需要调用两次反向插入以构建无向图 |
| int FindVertex(char \*name); | 根据节点名字寻找节点的下标 |
| void PrintGraph(); | 打印所构建的邻接表, 调试用 |
| proute GetNeartest(char \*start, char \*end); | 使用dijkstra算法计算最短路径 |

## 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。

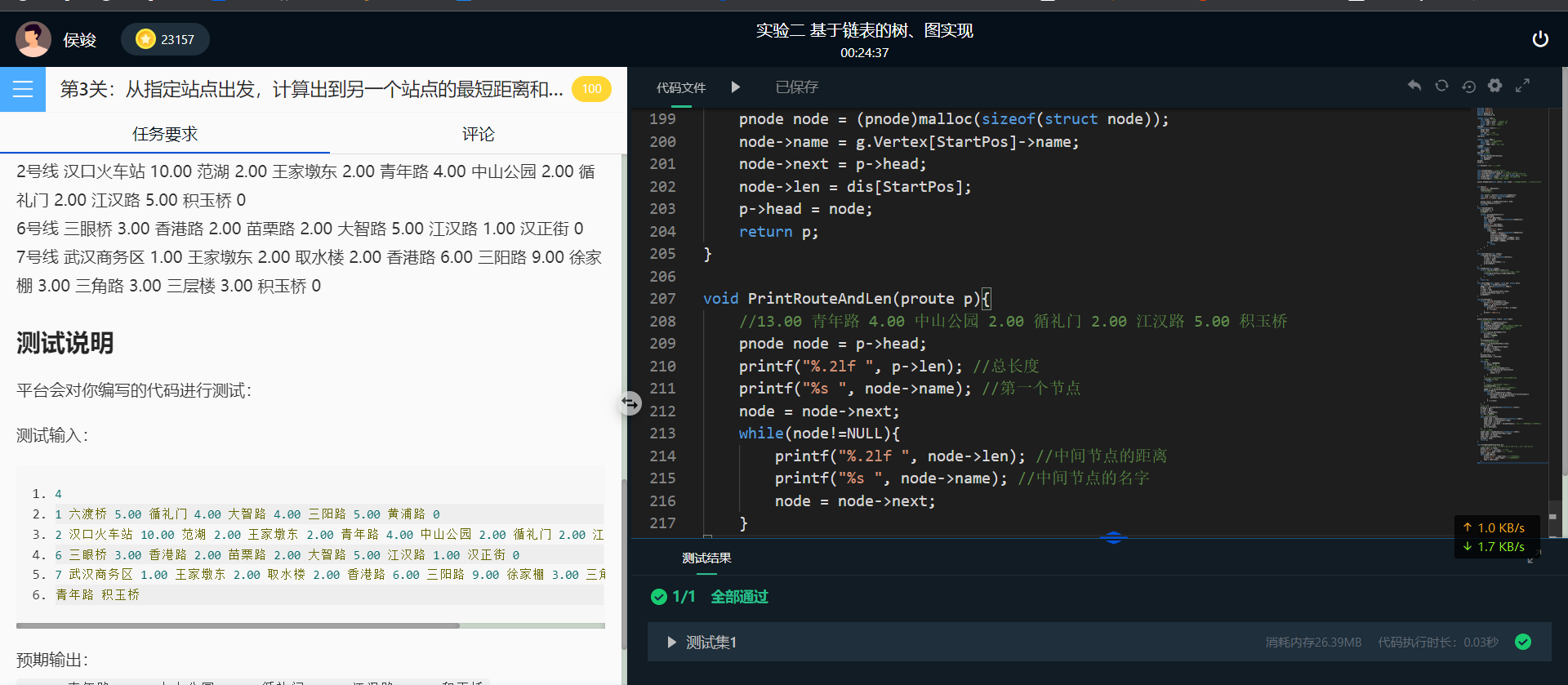


图3-4 测试通过

## 3.7 复杂度分析

创建图, 插入节点和边均为O(n); dijkstra算法分为两部分: 一部分要遍历未找到的节点,为O(n), n为节点数, 每一个遍历得到的节点还需要更新最短路径, 为O(n), 即O(n)\*O(n)=O(n^2), 故总时间复杂度为O(n)+O(n^2)=O(n^2)

空间复杂度: 存储图为O(n), 前驱结点数组, 标记未完成数组, 最短路径数组均为O(n), 故总空间复杂度为O(n)

## 3.8 结果分析

成功通过所有的给定测试用例，实现使用需求。

## 3.9 实验小结

1. 创建图时, 使用邻接表对一条边需要创建两次(u,v)和(v,u), 因为地铁站是双向图

2. 在dijkstra需要前驱结点记录路径

3. 输出路径时, 由于dijkstra算法得到的是基于起点的距离, 输出距离需要减去前一个节点, 得到站点之间的距离

# 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 基于链式存储结构线性表实现的源程序**

**任务1**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define NameSize 20

typedef struct node {

char \*name; // 站名

struct node \* next; //指向下一站

struct node \* pre; //指向上一站

double len; //到下一站的距离

}\* pnode;

typedef struct route {//线路

pnode head; //首站

pnode tail; //尾站

int num; //线路号s

}\*proute;

int RouteCnt = 0; //线路数

proute\* CreateRoutes(int num); //读入线路信息

proute CreateRoute(int num); //创建线路

void PrintAllRoute(proute\* p); //打印线路

int main(){

scanf("%d", &RouteCnt);

proute\* route = CreateRoutes(RouteCnt);

//printf("RouteCnt: %d\n", RouteCnt);

PrintAllRoute(route);

return 0;

}

void PrintAllRoute(proute\* p){

int num = RouteCnt;

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

printf("%d ", p[i]->num);

pnode temp = p[i]->head;

while(temp != NULL){

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

if(temp->next != NULL){

printf(" ");

}

if(temp->len != 0){

printf("%.2lf ", temp->len);

}

temp = temp->next;

}

printf("\n");

}

}

proute CreateRoute(int num){

proute p = (proute)malloc(sizeof(struct route));

p->num = num;

p->head = NULL;

p->tail = NULL;

return p;

}

proute\* CreateRoutes(int num){

proute\* route = (proute\*)malloc(sizeof(proute)\*num);

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

int number;

scanf("%d", &number);

route[i] = CreateRoute(number);

//读入站点

double distance = 1;

//读取站点

while(distance != 0){

pnode p = (pnode)malloc(sizeof(struct node));

p->name = (char\*)malloc(sizeof(char)\*NameSize);

scanf("%s", p->name);

scanf("%lf", &distance);

p->len = distance;

if(route[i]->head == NULL){

route[i]->head = p;

route[i]->tail = p;

p->next = NULL;

p->pre = NULL;

}else{

route[i]->tail->next = p;

p->pre = route[i]->tail;

route[i]->tail = p;

p->next = NULL;

}

}

}

return route;

}

**任务2**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define NameSize 20

typedef struct node {

char \*name; // 站名

struct node \* next; //指向下一站

struct node \* pre; //指向上一站

double len; //到下一站的距离

}\* pnode;

typedef struct route {//线路

pnode head;

pnode tail;

int num; //线路号s

}\*proute;

int RouteCnt = 0; //线路数

proute\* CreateRoutes(int num); //创建线路

proute CreateRoute(int num); //读入并创建线路

proute AddNode(proute p, double len\_pre, double len\_next, char name\_pre[10], char name[10]); //添加节点

proute DeleteNode(proute p, char \*name); //删除节点

int FromNumGetPos(proute\* p, int num); //根据线路号获取线路下标, 方便操作

void Operate(proute\* p); //操作

void PrintAllRoute(proute\* p); //打印线路

void PrintRoute(proute p); //打印线路

int main(){

scanf("%d", &RouteCnt);

proute\* route = CreateRoutes(RouteCnt);

//printf("RouteCnt: %d\n", RouteCnt);

PrintAllRoute(route);

Operate(route);

return 0;

}

void PrintAllRoute(proute\* p){

int num = RouteCnt;

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

PrintRoute(p[i]);

printf("\n");

}

}

void PrintRoute(proute p){

printf("%d ", p->num);

pnode temp = p->head;

while(temp != NULL){

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

if(temp->next != NULL){

printf(" ");

}

if(temp->len != 0){

printf("%.2lf ", temp->len);

}

temp = temp->next;

}

}

void Operate(proute\* p){

char \*op = (char\*)malloc(sizeof(char) \* NameSize);

scanf("%s", op);

if(op[0]=='a'){

//add 操作

int num;

scanf("%d", &num);

num = FromNumGetPos(p, num);

double len\_pre, len\_next;

char \*name\_pre = (char\*)malloc(sizeof(char) \* NameSize);

char \*name = (char\*)malloc(sizeof(char) \* NameSize);

scanf("%lf", &len\_pre);

if(len\_pre == 0){

scanf("%lf %s", &len\_next, name);

p[num] = AddNode(p[num], 0, len\_next, NULL, name);

}else{

scanf("%lf %s %s", &len\_next, name\_pre, name);

p[num] = AddNode(p[num], len\_pre, len\_next, name\_pre, name);

}

}else if(op[0] = 'd'){

// delete 操作

int num;

scanf("%d", &num);

num = FromNumGetPos(p, num);

char \*name = (char\*)malloc(sizeof(char) \* 20);

scanf("%s", name);

p[num] = DeleteNode(p[num], name);

}

}

proute CreateRoute(int num){

proute p = (proute)malloc(sizeof(struct route));

p->num = num;

p->head = NULL;

p->tail = NULL;

return p;

}

int FromNumGetPos(proute\* p, int \_num){

int cnt = RouteCnt;

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

if(p[i]->num == \_num){

return i;

}

}

return -1;

}

proute DeleteNode(proute p, char \*name){

pnode temp = p->head;

while(temp != NULL){

if(strcmp(temp->name, name) == 0){

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

if(temp->pre != NULL){

temp->pre->next = temp->next;

temp->pre->len = temp->pre->len + temp->len;

}else{

p->head = temp->next;

}

if(temp->next != NULL){

temp->next->pre = temp->pre;

}else{

p->tail = temp->pre;

}

free(temp);

PrintRoute(p);

return p;

}

temp = temp->next;

}

if(temp == NULL){

printf("删除失败，没有同名站点\n");

}

return p;

}

proute AddNode(proute p, double len\_pre, double len\_next, char \*name\_pre, char \*name){

pnode new\_node = (pnode)malloc(sizeof(struct node));

new\_node->name = (char\*)malloc(sizeof(char) \* NameSize);

strcpy(new\_node->name, name);

new\_node->len = len\_next;

new\_node->next = NULL;

new\_node->pre = NULL;

//插头

if(len\_pre == 0){

new\_node->next = p->head;

p->head->pre = new\_node;

p->head = new\_node;

PrintRoute(p);

return p;

}

if(p->head == NULL){

p->head = new\_node;

p->tail = new\_node;

}else{

pnode temp\_pre = NULL;

pnode temp = p->head;

while(temp != NULL){

if(strcmp(temp->name, name) == 0){

printf("增加失败，已有同名站点");

return p;

}

if(strcmp(temp->name, name\_pre) == 0 && temp\_pre == NULL){

temp\_pre = temp;

}

temp = temp->next;

}

if(temp\_pre == NULL){

printf("增加失败，没有与输入的增加位置前一站点同名的站点");

return p;

}

//考虑边界情况, 插到最后一站点

if(temp\_pre->next != NULL){

//temp\_pre->new\_node->temp

temp = temp\_pre->next;

temp->pre->next = new\_node;

new\_node->pre = temp\_pre;

new\_node->next = temp;

temp->pre = new\_node;

temp\_pre->len = len\_pre;

}else{

//temp\_pre->new\_node

temp\_pre->next = new\_node;

new\_node->pre = temp\_pre;

temp\_pre->len = len\_pre;

p->tail = new\_node;

}

}

PrintRoute(p);

return p;

}

proute\* CreateRoutes(int num){

proute\* route = (proute\*)malloc(sizeof(proute)\*num);

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

int number;

scanf("%d", &number);

route[i] = CreateRoute(number);

//读入站点

double distance = 1;

//读取站点

while(distance != 0){

pnode p = (pnode)malloc(sizeof(struct node));

p->name = (char\*)malloc(sizeof(char) \* 20);

char \*name = (char\*)malloc(sizeof(char) \* 20);

scanf("%s", name);

scanf("%lf", &distance);

strcpy(p->name, name);

p->len = distance;

if(route[i]->head == NULL){

route[i]->head = p;

route[i]->tail = p;

p->next = NULL;

p->pre = NULL;

}else{

route[i]->tail->next = p;

p->pre = route[i]->tail;

route[i]->tail = p;

p->next = NULL;

}

}

}

return route;

}

**任务3**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define NameSize 20

#define VertexSize 50

typedef struct node {

char \*name; // 站名

struct node \* next; //指向下一站

struct node \* pre; //指向上一站

double len; //到下一站的距离

}\*pnode;

typedef struct route {//线路

pnode head;

pnode tail;

int num; //线路号s

double len; //线路长度

}\*proute;

typedef struct Edge {

char \*name; //被指向的顶点

double dis; //边权值

struct Edge\* next; //下一条边

}\*pEdge;

typedef struct {

char \*name;

pEdge next;

}Vertex; //顶点

typedef struct {

Vertex \*Vertex[VertexSize];

int VertexNum;

int EdgeNum;

}Graph;

Graph g; //图

int RouteCnt = 0; //线路条数

void CreateGraph(); //读入图并创建

void PrintRouteAndLen(proute p); //打印线路和距离

void InsertVertex(char \*name); //插入节点, 如果存在则不插入

void InsertEdge(char \*start, char\* end, double dis); //插入边

int FindVertex(char \*name); //寻找节点, 返回节点位置

void PrintGraph(); //打印邻接表, 调试用

proute GetNeartest(char \*start, char \*end); //获取最近的站点, 使用Dijkstra算法

int main(){

scanf("%d", &RouteCnt);

CreateGraph();

//PrintGraph();

char \*start = malloc(sizeof(char)\*NameSize);

char \*end = malloc(sizeof(char)\*NameSize);

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

proute result = GetNeartest(start, end);

PrintRouteAndLen(result);

return 0;

}

void CreateGraph(){

g.VertexNum = 0;

g.EdgeNum = 0;

//读入图

for(int p=0;p<RouteCnt;p++){

int RouteNum;

scanf("%d",&RouteNum);

char \*StartName = malloc(sizeof(char)\*NameSize);

char \*EndName;

double Dis;

scanf("%s", StartName);

InsertVertex(StartName);

while(1){

scanf("%lf", &Dis);

if(Dis!=0){

EndName = malloc(sizeof(char)\*NameSize);

scanf("%s",EndName);

InsertVertex(EndName);

InsertEdge(StartName, EndName, Dis);

InsertEdge(EndName, StartName, Dis);

StartName = EndName;

}else{

break;

}

}

}

}

void InsertVertex(char \*name){

if(FindVertex(name)==-1){

Vertex \*v = malloc(sizeof(Vertex));

v->name = name;

v->next = NULL;

g.Vertex[g.VertexNum] = v;

g.VertexNum++;

}

}

int FindVertex(char \*name){

for(int i=g.VertexNum-1;i>=0;i--){

//printf("%s, %s\n", Vertexs[i]->name, name);

if(strcmp(g.Vertex[i]->name,name)==0){

return i;

}

}

return -1;

}

void InsertEdge(char \*start, char\* end, double dis){

int StartPos = FindVertex(start);

pEdge e = malloc(sizeof(struct Edge));

e->dis = dis;

e->name = end;

e->next = g.Vertex[StartPos]->next;

g.Vertex[StartPos]->next = e;

g.EdgeNum++;

}

void PrintGraph(){

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

printf("%s",g.Vertex[i]->name);

pEdge e = g.Vertex[i]->next;

while(e!=NULL){

printf("-%.2lf->%s",e->dis, e->name);

e = e->next;

}

printf("-->NULL\n");

}

}

proute GetNeartest(char \*start, char\* end){

//dijkstra算法

int StartPos = FindVertex(start);

int EndPos = FindVertex(end);

int visited[g.VertexNum]; //记录是否已经在最短路径中

double dis[g.VertexNum]; //记录到起点的距离

int path[g.VertexNum]; //记录前驱节点

//初始化

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

visited[i] = 0;

dis[i] = 1000000;

path[i] = -1;

}

visited[StartPos] = 1;

//初始化起点附近的节点

pEdge e = g.Vertex[StartPos]->next;

while(e!=NULL){

int pos = FindVertex(e->name);

dis[pos] = e->dis;

path[pos] = StartPos;

e = e->next;

}

dis[StartPos] = 0;

path[StartPos] = StartPos;

//开始遍历

while(1){

int min = 100000000;

int minPos = -1;

//找到距离最近的节点

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

if(visited[j]==0 && dis[j]<min){

min = dis[j];

minPos = j;

}

}

//没找到距离最近的节点, 说明已经遍历完了

if(minPos == -1){

break;

}

//找到了距离最近的节点, 更新距离

visited[minPos] = 1;

//以minPos为中间节点，更新其他节点的距离

pEdge e = g.Vertex[minPos]->next;

while(e!=NULL){

int pos = FindVertex(e->name);

if(visited[pos]==0 && dis[minPos]+e->dis<dis[pos]){

dis[pos] = dis[minPos]+e->dis;

path[pos] = minPos;

}

e = e->next;

}

}

//输出路径

proute p = (proute)malloc(sizeof(struct route));

p->head = NULL;

p->tail = NULL;

p->len = dis[EndPos];

int pos = EndPos;

//在开始节点之前插入

while(pos!=StartPos){

pnode node = (pnode)malloc(sizeof(struct node));

node->name = g.Vertex[pos]->name;

node->next = p->head;

node->len = dis[pos] - dis[path[pos]]; //距离为前驱节点到当前节点的距离

p->head = node;

pos = path[pos];

}

//插入开始节点

pnode node = (pnode)malloc(sizeof(struct node));

node->name = g.Vertex[StartPos]->name;

node->next = p->head;

node->len = dis[StartPos];

p->head = node;

return p;

}

void PrintRouteAndLen(proute p){

//13.00 青年路 4.00 中山公园 2.00 循礼门 2.00 江汉路 5.00 积玉桥

pnode node = p->head;

printf("%.2lf ", p->len); //总长度

printf("%s ", node->name); //第一个节点

node = node->next;

while(node!=NULL){

printf("%.2lf ", node->len); //中间节点的距离

printf("%s ", node->name); //中间节点的名字

node = node->next;

}

}