**计算机科学与技术学院课程设计成绩单**

课程名称：数据结构课程设计

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 夏红杰 | 性别 | 男 | 学号 | | 201813137205 | | 班级 | 计算机1805班 |
| 电话 | 18271822146 | | 综合成绩 | |  | | 成绩等级 | |  |
| 程序运行情况  （占总成绩20%） | |  | | | | | | | |
| 程序功能完善程度  （占总成绩20%） | |  | | | | | | | |
| 对问题的答辩情况  （占总成绩40%） | |  | | | | | | | |
| 学生的工作态度与独立工作能力  （占总成绩10%） | |  | | | | | | | |
| 设计报告的规范性  （占总成绩10%） | |  | | | | | | | |

A：90~100分 A-：85~89分 B+：82~84分 B：78~81分 B-：75~77分

C+：72~74分 C：68~71分 C-：64~67分 D：60~63分 F：<60分

武汉科技大学计算机科学与技术学院制表

**目 录**

**一、系统界面及功能介绍1**

（1）系统界面1

（2）程序结构2

**二、数据格式3**

（1）buses.txt文件3

（2）stations.txt文件3

（3）routes.txt文件4

**三、数据类型4**

（1）各类数据类型介绍4

（2）model.h头文件5

（3）公交线路图邻接表6

（4）公交线路图7

**四、测试使用的数据8**

（1）公交线路8

（2）站点8

**五、系统数据存储和管理9**

（1）C语言文件相关操作介绍9

（2）数据存储和管理流程图9

**六、查询公交线路和站点信息10**

（1）查询公交线路10

（2）查询站点信息11

**七、查询两站点之间至多换乘1次的路线并输出结果12**

（1）功能实现思路12

（2）算法描述13

核心算法：遍历两站点直接所有路径14

输出两站点之间至多换乘一次的所有路线流程图17

**八、新增公交线路、站点、路段信息18**

（1）新增路段信息18

（2）新增站点信息20

（3）新增公交线路21

**九、删除公交线路、站点、路段信息23**

（1）删除路段信息23

（2）删除站点信息25

（3）删除公交线路27

**九、修改公交线路、站点、路段信息28**

（1）修改路段信息28

（2）修改站点信息30

（3）修改公交线路31

**十、保存、重置、退出系统32**

（1）保存系统32

（2）重置系统32

（3）退出系统32

**十一、课程设计总结33**

（1）学到了什么33

（2）可以改进的地方35

**附录：基本操作（增、删、改）源代码36**

1.邻接表中新增节点AddNewNodeToMap 36

2.添加路段到ROUTES数组中相关位置AddNewRouteToArray 36

3.删除邻接表中节点RemoveNodeToMap 37

4.删除ROUTES数组中路段RemoveRouteFromArray 37

5.修改邻接表中节点ChangeNodeToMap 38

6.寻找两站点之间所有路线QueryRoute 38

**公交线路图的构建和查询**

计算机科学与技术学院 计算机1805班 夏红杰

**【摘要】**2020年是我们国家全面建成小康社会、实现第一个百年奋斗目标之际，随着人们生活水平的提高，出行工作、旅游等成为每日常态，而高铁、公交、地铁等公共交通设施更是最佳选择，然而城市公交、地铁等建设庞大复杂，人们急需一个便捷的方式找到最佳路线，公交系统应运而生，方便人们查询公交路线。本文介绍用C语言实现简易城市公交管理系统的原理及思想，能够实现基本查询、增加、删除、修改功能。

**【关键字】** 公交管理系统 数据结构 图

1. **系统界面及功能介绍**

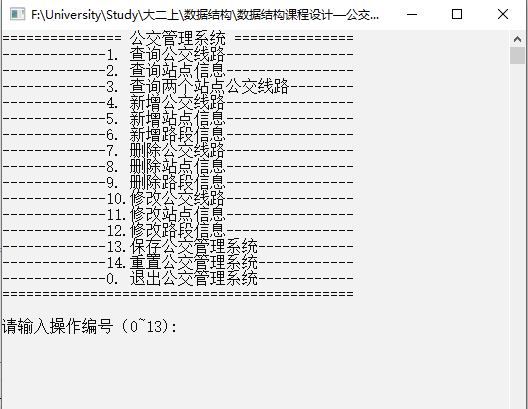
**（1）系统界面**

图1-1 公交管理系统界面

本公交管理系统界面显示功能选择编号，根据用户输入编号进行不同功能：

1-3均为查询功能，查询相关公交线路、站点、线路信息

4-6均为添加功能，添加公交线路、站点、路段信息

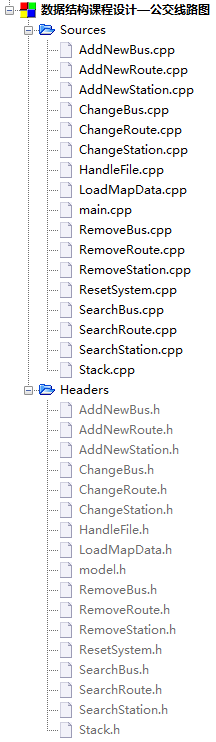
7-9均为删除功能，删除公交线路、站点、路段信息

10-12均为修改功能，修改公交线路、站点、路段信息

13为保存功能，保存用户进行增删改操作后的数据，并提示用户重启系统

14为重置功能，重置公交管理系统所有数据

0为退出公交管理系统，并提示用户是否保存修改数据

**（2）程序结构**

建立邻接表以及程序基本函数：

①查找公交索引号、站点索引号，

②查找指定站点在某一条公交线路中前驱站点、后继站点，

③确定指定站点是否在某一条公交线路中（起点站、终点站等）

数据结构定义头文件

查询路线信息辅助栈结构

查询功能文件，查询公交线路、站点信息、两站点至多换乘一次的路线

重置系统

删除功能文件，删除公交线路、站点信息、两站点路段信息

显示界面

文件处理，读取数据、存储数据

修改功能文件，修改公交线路、站点信息、两站点之间路段信息

添加功能文件，新增公交线路、站点信息、两站点路段信息

图1-2 公交管理系统程序结构图

**二、数据格式**

用三个txt文本存储数据，buses.txt文件存储公交信息，stations.txt文件存储站点信息，routes.txt文件存储路段信息。

1.buses.txt文件每一行为一条公交线路信息，每条公交线路信息的记录格式为：①公交线路编号(nBus) ②公交线路名称（busName） ③公交线路起点站（busStart） ④公交线路终点站（busEnd），文件格式与示例如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 公交线路1编号 | 公交线路1名称 | 公交线路1起点站 | 公交线路1终点站 |
| 公交线路1编号 | 公交线路1名称 | 公交线路1起点站 | 公交线路1终点站 |
| …… | …… | …… | …… |

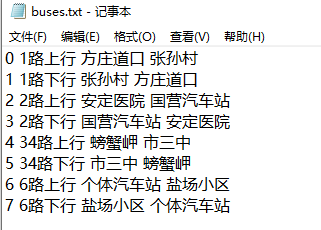


图2-1 buses.txt文件

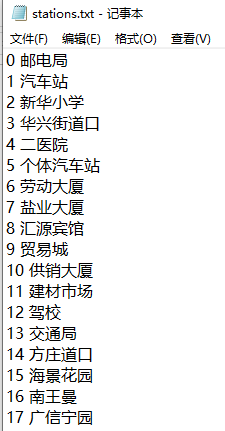
2.stations.txt文件中，每一行记录为一个站点信息。格式为：“站点编号（nStation） 站点名称（stationName）”。（每个字段使用空格符分隔）。

图2-2 stations.txt文件

|  |  |
| --- | --- |
| 站点1编号 | 站点1名称 |
| 站点2编号 | 站点2名称 |
| …… | …… |

3.routes.txt文件中，每一行记录为一个路段信息，表示两站点直接有公交线路经过。格式为：“公交线路编号（nBus），路段出发站点编号（nStart），路段目标站点编号（nEnd），两站点之间距离（distance）”。（每个字段使用“，”分隔）。

|  |  |  |  |
| --- | --- | --- | --- |
| 路段1公交线路编号 | 路段1出发站点编号 | 路段1目标站点编号 | 路段1两站点之间距离 |
| 路段2公交线路编号 | 路段2出发站点编号 | 路段2目标站点编号 | 路段2两站点之间距离 |
| …… | …… | …… | …… |

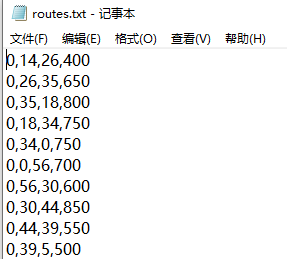


图2-3 routes.txt文件

**三、数据类型**

本系统整个公交线路图以邻接表为存储方式。

1. 整个公交线路图数据结构为BusMap，包括公交线路数组（buses）、站点数组（stations）、站点数（station\_num）、公交线路数（bus\_num）。
2. 公交数据结构为Bus，包括公交名（name），起点站（start），终点站（end）。
3. 站点数组为邻接表表头结点数组，站点数据结构为Station，包括站点名（station），从该站点出发的所有下行路线的链域（routes）。
4. 邻接表节点为路段，路段数据结构为Route，包括指向的站点索引号（station），公交索引号（bus），两站点之间的距离（distance），起始站点相同的下一条下行路线（next）以及遍历路径时辅助变量status。

在工程中，数据结构的定义位于model.h头文件中：

typedef struct Bus

{

char \*name;//公交名

int start;//起点站

int end;//终点站

}Bus;

typedef struct Station

{

char \*station;//站点名

struct Route \*routes;//从该站点出发的所有下行路线的链域

}Station;

typedef struct Route

{

int station;//指向的站点索引号

int bus;//公交索引号

int distance;//两站点之间的距离

struct Route \*next;//起始站点相同的下一条下行路线

int staus=0;//寻找路径时，若该route两站点均不在遍历的栈结构中，则为0

}Route;

typedef struct BusMap

{

Bus \*buses;//公交线路数组

Station \*stations;//站点数组

int station\_num;//站点数

int bus\_num;//公交线路数

}BusMap;

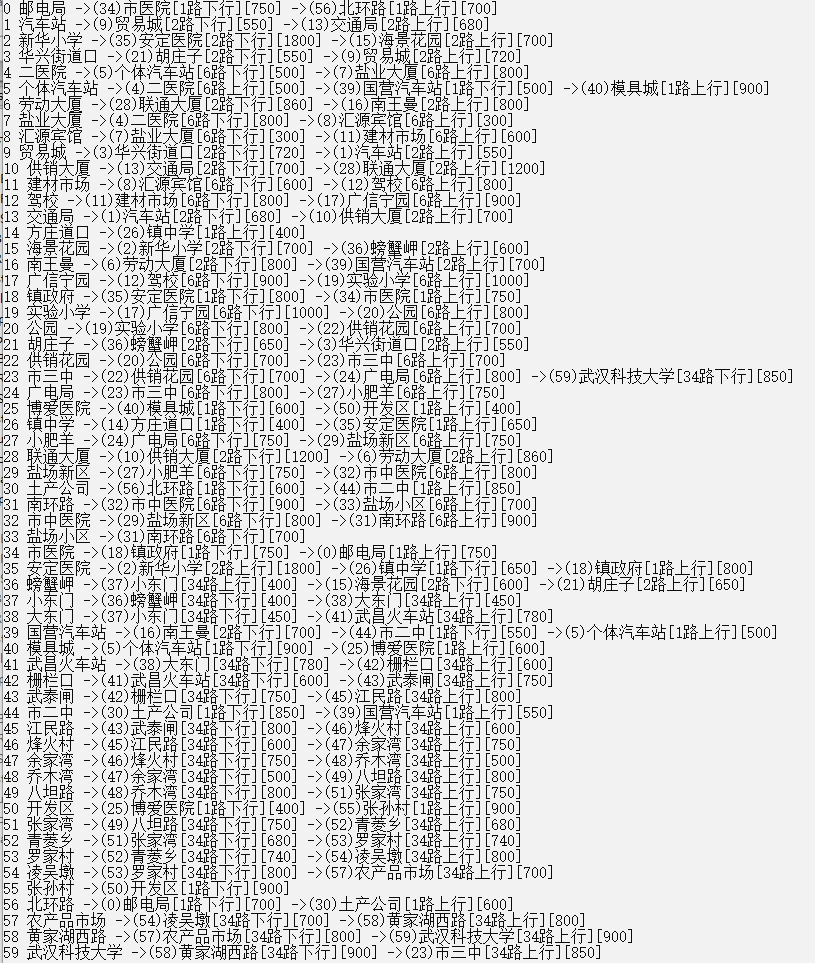
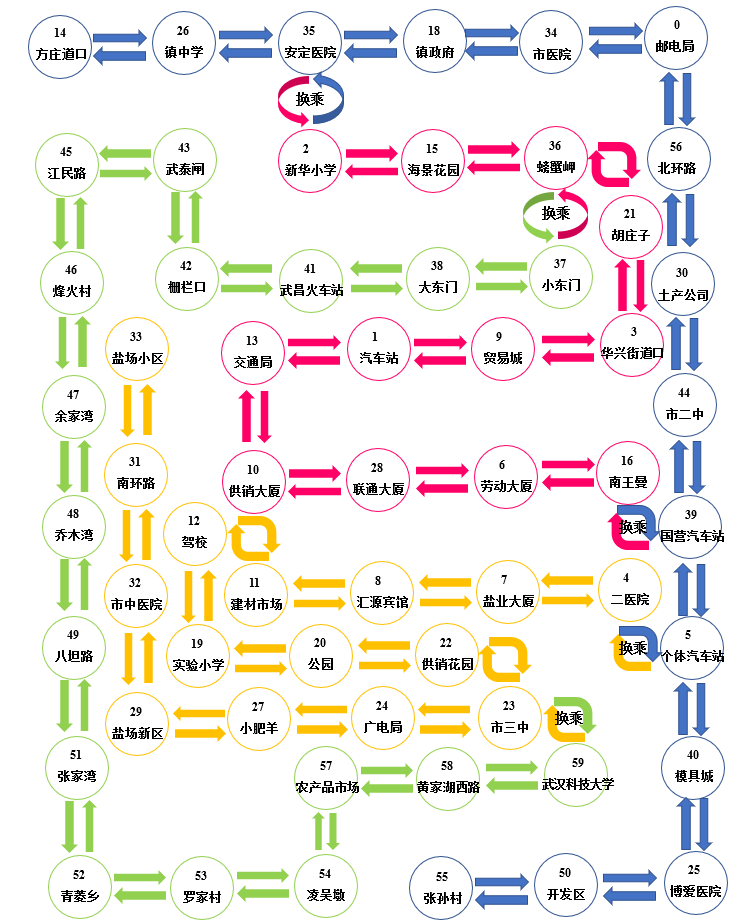
本系统初始公交线路邻接表如下：

图3-1 公交线路图邻接表

本系统初始公交线路图如下：

**图3-2 公交线路图**

**6路**

**34路**

**2路**

**1路**

**四、测试使用的数据**

为了便于更好的理解程序的运行过程，本系统测试数据为：

1. **公交线路：共设置八条公交线路，分别为：**

0 1路上行：从 [方庄道口] 开往 [张孙村]，共经过15个站点

1 1路下行：从 [张孙村] 开往 [方庄道口]，共经过15个站点

2 2路上行：从 [安定医院] 开往 [国营汽车站]，共经过14个站点

3 2路下行：从 [国营汽车站] 开往 [安定医院]，共经过14个站点

4 34路上行：从 [螃蟹岬] 开往 [市三中]，共经过19个站点

5 34路下行：从 [市三中] 开往 [螃蟹岬]，共经过19个站点

6 6路上行：从 [个体汽车站] 开往 [盐厂小区]，共经过17个站点

7 6路下行：从 [盐厂小区] 开往 [个体汽车站]，共经过17个站点

**2.站点：共设置60个站点，分别为：**

邮电局，汽车站，新华小学，华兴街道口，二医院，

个体汽车站，劳动大厦，盐业大厦，汇源宾馆，贸易城，

供销大厦，建材市场，驾校，交通局，方庄道口，

海景花园，南王曼，广信宁园，镇政府，实验小学，

公园，胡庄子，供销花园，市三中，广电局，

博爱医院，镇中学，小肥羊，联通大厦，盐场新区，

土产公司，南环路，市中医院，盐场小区，市医院，

安定医院，螃蟹岬，小东门，大东门，国营汽车站，

模具城，武昌火车站，栅栏口，武泰闸，市二中，

江民路，烽火村，余家湾，乔木湾，八坦路，

开发区，张家湾，青菱乡","罗家村","凌吴墩，

张孙村，北环路，农产品市场，黄家湖西路，武汉科技大学

**五、系统数据存储和管理**

程序对文本的数据管理和操作思路如下：程序定义三个全局数组，公交数组（BUSES）、站点数组（STATIONS）、路段数组（ROUTES），当程序开始运行时，首先执行函数ReadDataFromFile()，将buses.txt,stations.txt,routes.txt三个文本内容分别载入相应数组中，并执行函数LoadMapData()建立邻接表，在程序功能实现过程中，对邻接表结构和数组数据进行更改，最后执行函数WriteDataToFile()将数组中数据重新载入文件中。文件相关操作如下:

1. 打开文件 文件指针=fopen(文件名，打开方式)

**"r"：**以“只读”方式打开文件。只允许读取，不允许写入。文件必须存在，否则打开失败。

**"w"：**以“写入”方式打开文件。如果文件不存在，那么创建一个新文件；如果文件存在，那么清空文件内容（相当于删除原文件，再创建一个新文件）。

**"a"：**以“追加”方式打开文件。如果文件不存在，那么创建一个新文件；如果文件存在，那么将写入的数据追加到文件的末尾（文件原有的内容保留）。

1. 按行读取文件中数据 fgets(存储空间地址，存储空间大小，文件指针)
2. 按字节获取文件中数据 fscanf(文件指针，数据格式，存储变量)
3. 按字节写入数据到文件 fprintf(文件指针，数据格式，写入的数据)
4. 关闭文件 fclose(文件指针)

**ReadDataFromFile**

stations.txt

**WriteDataToFile**

STATIONS

增删

查改

建立

邻接表

**LoadMapData**

**ReadDataFromFile**

BUSES

buses.txt

**ReadDataFromFile**

**WriteDataToFile**

ROUTES

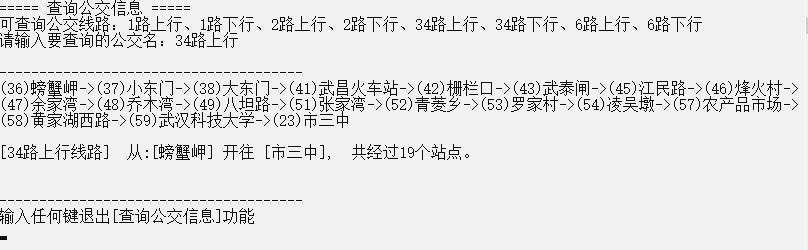
routes.txt

**WriteDataToFile**

图5-1 数据存储和管理流程图

**六、****查询公交线路和站点信息**

**（1）查询公交线路**

 系统第一个功能为查询公交线路，操作编号为1，系统首先会提示用户可查询的公交线路名称，根据用户需要查询的公交名显示公交线路以及公交线路信息：起点站、终点站、该线路经过总站点数。

输出查询的公交信息

结束

pStation=&Map.stations[i];

pStRoute=pStation->routes;

num++;

yes

i != nEnd

开始

输入查询的公交名

查找公交索引号、公交起点站、终点站

设置循环变量：i=nStart;num=0;

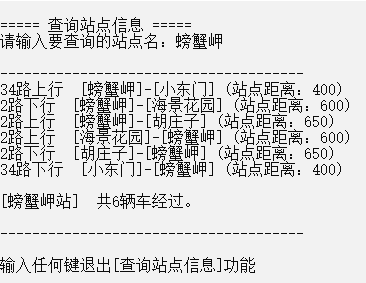
no

功能图1 查询公交线路

流程图1 查询公交线路

**（2）查询站点信息**

查询站点信息操作编号为2，系统会根据用户输入的站点名，依次显示经过该站点的路段，显示格式为：公交线路 [起始站点]-[目标站点] （站点距离），

并提示经过该站点的总车数。

功能图2 查询站点信息

开始

输入查询的站点名

pStation=&Map.stations[nStation];

pStRoute=pStation->routes;

pStRoute!=NULL

输出路段

i<Map.station\_num

i++;

nStation=i;

结束

yes

yes

no

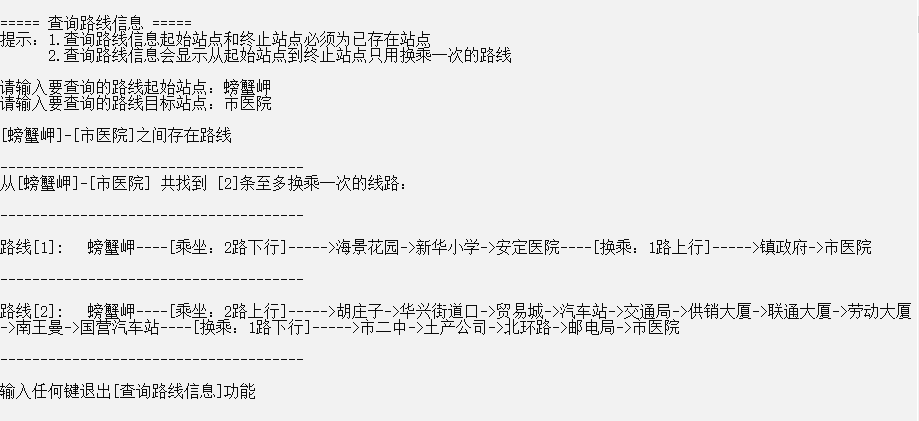
no

流程图2 查询站点信息

遍历邻接表

通用流程

**七、****查询****两站点之间的路线，找到至多换乘1次的路线，并输出结果**

查询两站点之间路线操作编号为3，系统会根据用户输入要查询的路线起始站点和目标站点，首先判断两站点之间是否存在路线，若两个站点之间有路线，则找到所有最多换乘1次的路线，然后依次输出。

功能图3 查询两站点之间的路线

功能实现思路：将整个公交线路图看做一个有向有环图（在换乘处形成环），因而找到两站点之间路线等价于寻找有向图中两顶点之间路径，首先通过寻找两顶点之间所有路线的过程判断两站点是否存在路线，并将路线记录下来，接下来对找到的所有路线依次判断换乘次数，最后输出两站点之间至多换乘一次的所有路线。

在寻找有向图中两顶点之间所有路线中，需要利用一个辅助数组：站点是否存在遍历栈数组（stationInStack），和一个辅助变量：路段状态（status），当一条路段两站点均已存在遍历栈中时，设置status=0，否则status=1。

在记录路径过程中，沿着起始站点链域依次将站点入栈，并进行深度优先搜索，当栈顶元素为目标站点时，即为找到一条路线，利用两个辅助栈：站点栈（stationStack）记录路径依次经过的站点，公交栈（busStack）记录路径依次乘坐的公交，并利用变量pathNum记录路线总数，pathStationNum数组记录每条路线经过站点数。

**算法描述：寻找两站点之间所有路径算法（QueryRoute）**

**（1）第一步：初始化Path数组**

int i;

**//初始化路线数位3，每条路线经过站点数20**

int initialPathNum=3,initialPathStationNum=20;

**//path数组为一个二维数组，每一行为一条路线，每一行列数为当前路线经过站点数**

**//path数组结构为 bus – station – bus – station**

int \*\*path=(int\*\*)malloc(sizeof(int\*)\*initialPathNum);

int \*pathStationNum=(int\*)malloc(sizeof(int)\*initialPathNum);

for(i=0; i<initialPathNum; i++)

**//分配空间\*2 与path结构有关，每一个站点对应一个公交**

path[i]=(int\*)malloc(sizeof(int)\*initialPathStationNum\*2);

**//每一条路线第一个bus元素初始化为-1**

path[0][0]=-1;

**//每一条路线第一个station元素初始化为起始站点**

path[0][1]=nStartStation;

**（2）第二步：初始化站点栈辅助数组**

**//在遍历路径的过程中，将访问过的站点入栈，stationInStack状态设为1**

int \*stationInStack=(int\*)malloc(sizeof(int)\*Map.station\_num);

for(i=0; i<Map.station\_num; i++)

stationInStack[i]=0;

**（3）第三步：初始化route status**

**//route结构的status为在寻找路径时，判断栈顶元素是否有下一个可入栈的后续站点辅助变量，只有当与需要扩展的站点相关的路段status状态为0（表示在之前遍历路径时未访问该边）时才可扩展该站点下一个站点，否则该站点出栈。**

Station \*pStation;

Route \*pRoute;

for(i=0; i<Map.station\_num; i++)

{

pStation=&Map.stations[i];

pRoute=pStation->routes;

while(pRoute!=NULL)

{

pRoute->staus=0;

pRoute=pRoute->next;

}

}

**（4）第四步：初始化栈结构**

**//stationStack为遍历过程中入栈的站点，busStack记录相应路段的公交**

Stack stationStack,busStack;

InitStack(stationStack);

InitStack(busStack);

**//nStartStation入栈**

PushStack(stationStack,nStartStation);

PushStack(busStack,-1);

stationInStack[nStartStation]=1;

**（5）第五步：（核心算法）遍历两站点之间所有路径**

pathNum=0;**//记录路线数**

flage=0;//两站点之间是否存在路径标志

While(!EmptyStack(stationStack))

{

nStation = GetTopStack(stationStack);**//获得栈顶元素**

if(nStation == nEndStation)**//找到了一条路径**

{

if(flag == 0)

flag=1;**//只要找到一条路径则说明两站点之间存在路径**

pathNum++;**//路线数增加**

if(pathNum > initialPathNum) **//动态增加数组path空间**

pathStationNum[pathNum-1]=GetStackNum(stationStack)**//当前路线站点数**

if(pathStationNum[pathNum-1]>initialPathStationNum)

**//动态增加数组path[pathNum-1]空间**

**//依次将栈内站点、公交记录到数组path[pathNum-1]中**

for(k=1,j=2; k<pathStationNum[pathNum-1]; k++,j+=2)**//注意j+=2**

{

path[i][j]=GetStackElement(busStack,k);

path[i][j+1]=GetStackElement(stationStack,k);

}

**//nEndStation出栈**

PopStack(stationStack); PopStack(busStack);

stationInStack[nEndStation]=0;**//站点入栈状态为0**

**//更新路段status状态，使得所有两个端点都不在栈内的路段的状态为0**

updateRoutesStatus();

}

else**//继续遍历站点**

{

**//沿着栈顶站点邻接表寻找下一个可入栈站点**

pStation=&Map.stations[nStation];

pRoute=pStation->routes;

while(pRoute!=NULL)

{

i=pRoute->station;

**//入栈要求：该站点未曾访问过，且该条路段也没有访问过**

if(stationInStack[i]==0 && pRoute->staus==0)

{

PushStack(stationStack,i); PushStack(busStack,pRoute->bus);

stationInStack[i]=1;**//站点入栈状态改为1**

pRoute->staus=1;**//路段访问状态改为1**

break;

}

pRoute=pRoute->next;

}

**//该栈顶站点没有符合要求的后续站点**

if(pRoute==NULL)

{

**//nStation出栈**

PopStack(stationStack); PopStack(busStack);

stationInStack[nStation]=0; **//站点入栈状态改为0**

**//更新与该站点相关所有路段访问状态为0**

updateRouteStatus(Map,nStation,stationInStack);

}

}

经过上述算法遍历公交线路图，即可将两站点之间所有路线保存在path中。

**（6）辅助算法updateRouteStatus:**

Station \*pStation=&Map.stations[station];**//更新所有以该站点为起始站点路段**

Route \*pRoute=pStation->routes;

while(pRoute!=NULL){

if(stationInStack[pRoute->station]==0)

pRoute->staus=0;

pRoute=pRoute->next;}

for(i=0; i<Map.station\_num && i!=station; i++){

pStation=&Map.stations[i];

pRoute=pStation->routes;

while(pRoute!=NULL){ **//更新所有以该站点为目标站点路段**

if(pRoute->station==station && stationInStack[i]==0)

pRoute->staus=0;

pRoute=pRoute->next;}

}

栈顶站点出栈

更新路段status状态

NO

NO

nEndStation出栈

更新路段status状态

开始

pathNum=0;

flag=0;

!EmptyStack(stationStack)

获得栈顶元素nStation

找到了一条路径

nStation=nEndStation？

flag=1;pathNum++;

pathStationNum[pathNum-1]=GetStackNum(stationStack);

数组空间不足？

依次将栈内站点、公交记录到数组

动态增加数组空间

YES

YES

YES

NO

沿着栈顶站点邻接表寻找下一个可入栈站点

stationInStack[i]==0 && pRoute->staus==0

站点入栈

YES

该栈顶站点没有符合要求的后续站点

pRoute!=NULL

YES

NO

NO

YES

结束

流程图3 遍历两站点之间所有路线

**（7）****输出两站点之间至多换乘一次的所有路线:**

在上述流程之后，得到两站点之间所有路径，并且保存在数组path中，path数组结构为：每一行为一条路线，每一行bus-station-bus-station……构成路线。接下来在该数组基础上找到至多换乘一次的所有路线，需要设置两个辅助变量，inBus表示当前乘坐公交，每次检索下一站点时会将公交与inBus比较，若不同则表示换乘，flag为换乘标志，当flag为2时表示该路线需要换乘两次不可取舍弃，设置transferBusOnceRoutes数组记录可行路线。

流程图4 输出两站点之间至多换乘一次的所有路线

开始

i=0;

i<pathNum?

flag=0;

inBus=path[i][2];

k=1,j=2;

k<pathStationNum[i]

path[i][j]!=inBus?

flag==1?

transferBusOnceRoutes[i]=-1;

flag=2;

YES

inBus=path[i][j];

flag=1;

YES

NO

flag!=2?

transferBusOnceRoutes[i]=i;

num++;

YES

YES

NO

输出所有至多换乘一次的路线

NO

结束

**八、新增公交线路、站点、路段信息**

**（1）新增路段信息**

新增路段信息操作编号为6，功能实现思路：系统首先提示用户现有公交路线并且用户只用输入需要新增路线所在上行公交线路即可，接下来根据用户输入的公交线路显示公交信息，提示用户需要新增的路段的起始站点和目标站点以及两站点之间距离，系统进行判断路线是否合理后将新增路线添加至公交线路系统中，更新邻接表，最后系统自动修改下行路线，完成新增路段功能。

功能结构：

1. AddNewRouteToMap：添加公交系统线路信息公共接口
2. JudgeRouteType：检测路段类型
3. AddNewRouteFlag0、AddNewRouteFlag3、AddNewRouteFlag12：根据路段类型进行新增路线、更改邻接表结构
4. AddNewRouteToArray：更改ROUTES数组实现数据修改功能

功能图4 新增路段信息

AddNewRouteFlag12

开始

用户输入新增路段所在公交线路、起始站点、目标站点

检测路线是否已存在

检测路线类型：JudgeRouteType

flag=0: 起始站点存在公交线路中，目标站点不存在

flag=1: 目标站点为当前公交线路起点站

flag=2: 起始站点为当前公交线路终点站

flag=3: 起始站点不存在公交线路中，目标站点存在

AddNewRouteFlag0

nStart1:新增路线起始站点

nEnd1:新增路线目标站点

nStart2=nEnd1;

nEnd2=起始站点原线路中下一站点

如果为上行公交添加路段，提示用户输入nStart2-nEnd2距离

修改邻接表、ROUTES数组：

nStart1-nEnd2->nStart1-nEnd1

在邻接表中、ROUTES数组中增加：

nStart2-nEnd2

AddNewRouteFlag3

nStart1:新增路线起始站点

nEnd1:新增路线目标站点

nStart2=起始站点原线路中上一站点

nEnd2=nStart1;

如果为上行公交添加路段，提示用户输入nStart2-nEnd2距离

修改邻接表、ROUTES数组：

nStart2-nEnd1->nStart2-nEnd2

在邻接表中、ROUTES数组中增加：

nStart1-nEnd1

修改当前公交线路

起点站或终点站

在邻接表中、ROUTES数组中增加：

nStart-nEnd

nStart:新增路线起始站点

nEnd:新增路线目标站点

自动修改下行路线

结束

流程图5 新增路段信息AddNewRouteToMap

**（2）新增站点信息**

新增站点信息操作编号为5，功能实现思路：系统首先提示用户输入新站点名称并检测是否合法，接下来提示用户依次输入新站点路段信息：1.公交线路，系统会显示输入的公交线路信息；2.与该新增站点相关路段信息（出发站点、目标站点、两站点之间距离），最后系统调用[AddNewRouteToMap]功能。

输入公交线路不为0

开始

输入新站点名称

输入新站点相关路段信息

调用新增路段功能

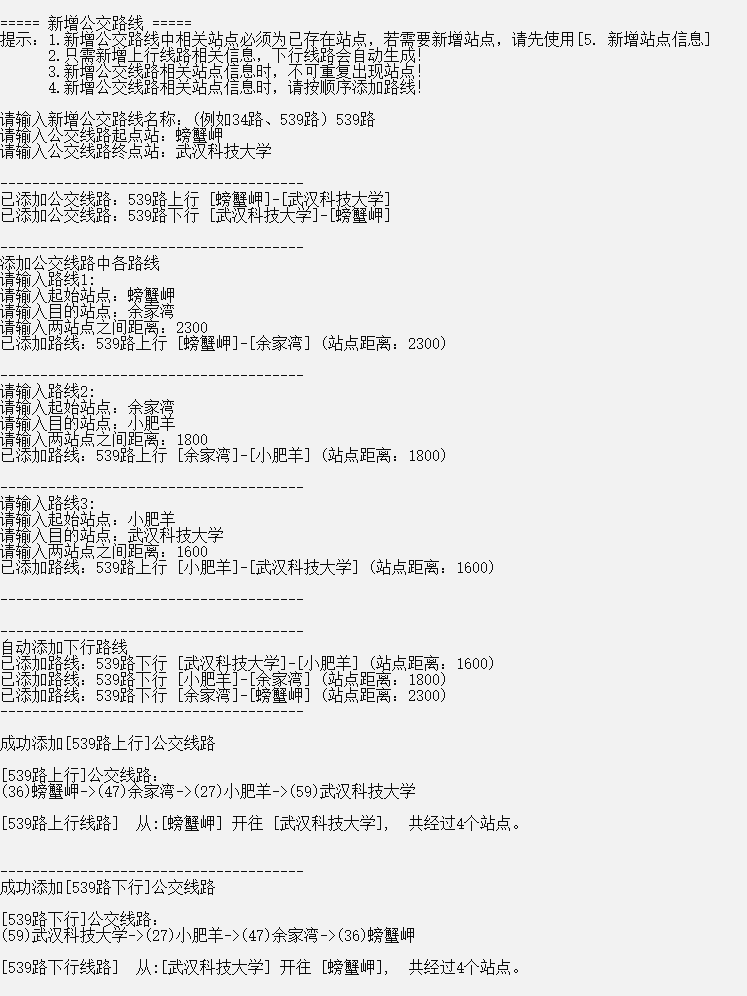
结束

功能图5 新增站点信息

流程图6 新增站点信息AddNewStationToMap

**（3）新增公交线路**

新增公交线路操作编号为4，功能实现思路：系统首先提示用户输入新增公交线路名称、起点站、终点站，并将将公交线路信息添加至BUSES数组，接下来系统提示用户依次输入上行公交线路路段，每一条新增路段调用[AddNew

RouteToMap]功能，最后系统自动添加下行公交路线，实现新增公交线路功能。

功能图6 新增公交线路

开始

输入新公交线路名、起点站、终点站

在BUS数组中新增公交线路信息（上下行两条），在邻接表新增路段：

起点站-终点站（上行即可）

用户依次输入新增公交线路中路段

nStart-nEnd (distance)

路段目标站点是否为公交终点站

检测路段是否合法

在邻接表和ROUTES数组中新增：nStart-nEnd (distance)

删除邻接表和ROUTES数组中：nStart-busEnd

在邻接表和ROUTES数组中新增：nEnd->busEnd

按照上行路线自动生成下行路线和ROUTES数组数据

检测路段是否合法

修改邻接表和ROUTES数组中：

nStart-busEnd(distance)

结束

NO

NO

YES

YES

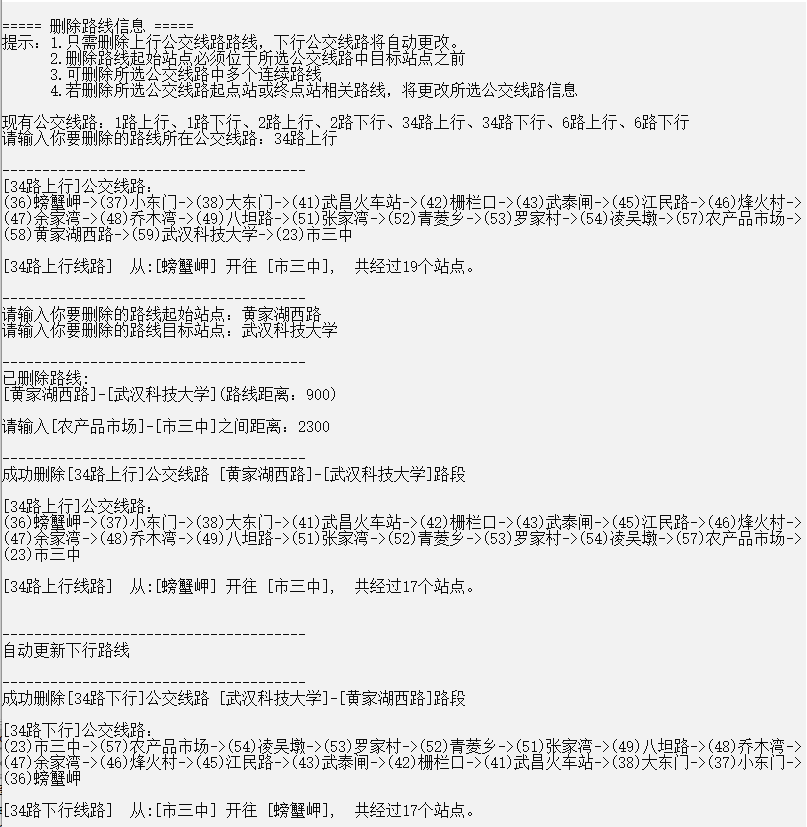
YES

NO

流程图7 新增公交线路AddNewBusToMap

**九、删除公交线路、站点、路段信息**

**（1）删除路段信息**

删除路段信息操作编号为9，功能实现思路：系统首先提示现有公交线路，根据用户输入要删除的路段所在公交线路显示公交信息，然后根据用户输入的起始站点与目标站点依次删除其间所有路段，注意其中的特殊情况（删除路段中站点可能为公交起点站或终点站），最后更新下行路线即可。

功能图7 删除路段信息

开始

输入删除路段所在公交线路

输入删除路段起始站点、目标站点

遍历邻接表，找到需要删除的总路段数routeNum，并分别记录到数组STARTS、ENDS、DISTANCE中

其中STARTS[0]=起始站点前一站点

ENDS[0]=目标站点后一站点

删除路线起始站点为公交线路起点站

删除路线目标站点为公交线路终点站

i<=routesNum

i=1;

从邻接表和ROUTES数组中删除：STARTS[i]-ENDS[i]

修改公交起点站为ENDS[i]

修改上行公交起点站

修改下行公交终点站

修改BUS数组

i<=routesNum

i=1;

从邻接表和ROUTES数组中删除：STARTS[i]-ENDS[i]

修改邻接表中路段：

STARTS[0]-STARTS[i]->

STARTS[0]-ENDS[i]

修改最后一条路段：

STARTS[0]-STARTS[1]

修改上行公交终点站

修改下行公交起点站

修改BUS数组

从邻接表和ROUTES数组中删除：STARTS[i]-ENDS[i]

修改邻接表中路段：

STARTS[0]-STARTS[i]->

STARTS[0]-ENDS[i]

最后调整整个邻接表和ROUTES数组

自动更新下行路线

结束

YES

YES

YES

YES

NO

NO

NO

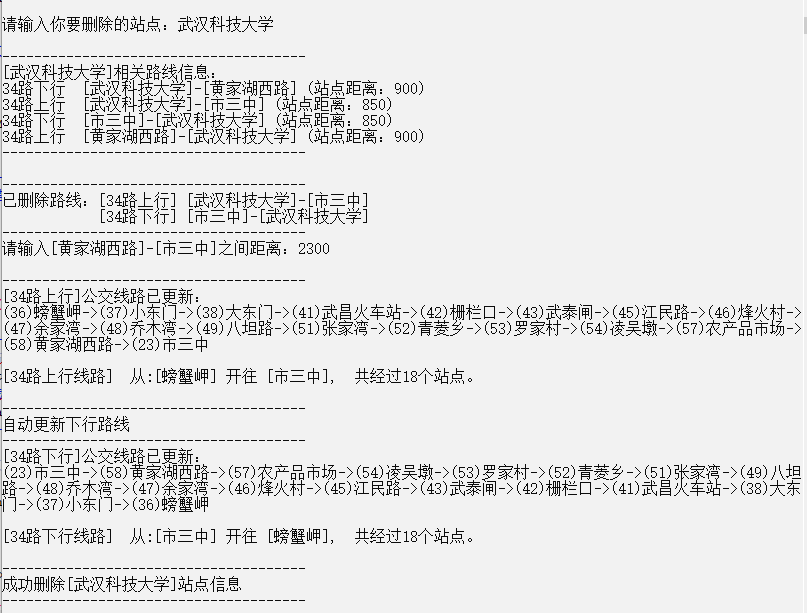
NO

流程图8 删除路段信息RemoveRouteToMap

**（2）删除站点信息**

删除站点信息操作编号为8，功能实现思路：系统首先提示用户输入要删除的站点名称，然后显示与该站点相关的所有路线信息，系统会先判断该站点是否为公交线路起点站或终点站，若是则修改公交信息，然后依次调用RemoveRouteToMap功能删除以该站点为起始站点的所有路段。

需要注意：在删除与站点相关路段时，为保证邻接表结构的完整性，需要连接删除路段前后站点并调整邻接表结构及所有站点编号。



功能图8 删除站点信息

YES

NO

YES

YES

NO

NO

删除站点为公交线路起点站

删除站点为公交线路终点站

开始

输入删除站点名称

遍历所有公交

删除路段：

起点站-下一站点

删除路段：

上一站点-终点站点

删除所有上行路线以该站点为起始站点的路线

pStation=&Map.stations[nStation]; pRoute=pStation->routes;

pRoute!=NULL

筛选上行路线

删除路段nStation->postStation

修改路段：preStation->nStation->preStation->postStation

自动更改下行路线

调整邻接表所有站点编号

结束

流程图9 删除站点信息RemoveStationToMap

**（3）删除公交线路**

删除站点信息操作编号为7，功能实现思路：系统首先提示现有公交线路，根据用户输入的要删除的公交线路，从起点站开始遍历公交线路，依次调用[RemoveNodeToMap]删除路段，注意每删除一条路段，都需要修改公交起点站。

功能图9 删除公交线路

开始

i=busStart

i!=busEnd

沿着邻接表遍历公交线路，依次删除路段，并修改公交起点站

i=pRoute->station;

自动删除下行路线

结束

输入要删除的公交线路

YES

NO

流程图10 删除公交线路RemoveBusToMap

**十、修改公交线路、站点、路段信息**

**（1）修改路段信息**

修改路段信息操作编号为12，系统首先会提示现有公交线路，根据用户输入的要修改的路段所在公交线路显示公交信息，然后提示用户输入要删除的路线起始站点、目标站点，系统自动识别需要修改的路段数，然后逐条提示用户是否需要修改每条路段的起始站点、目标站点、站点距离修改路段有四种情况：①修改路段的起始站点并且该站点为公交线路起点站：修改公交起点站

②修改路段的目标站点并且该站点为公交线路终点站：修改公交终点站

③修改路段起始站点：修改该站点与前一站点路段信息

1. 修改路段目标站点：删除该站点与后一站点路段信息

功能图10 修改路段信息

自动修改下行路线

结束

是否修改站点距离

修改站点距离

是否修改目标站点

目标站点为公交终点站

修改公交信息;

修改邻接表、ROUTES数组：

routeStart-routeEnd

->routeStart->newEnd

修改路段：routeEnd-postStation

->newEnd-postStation;

删除路段：routeStart-routeEnd

增加路段：routeStart-newEnd

是否修改起始站点

起始站点为公交起点站

修改公交信息;

修改邻接表、ROUTES数组：

routeStart-routeEnd

->newStart->routeEnd

修改路段：preStation-routeStart

->preStation-newStart;

删除路段：routeStart-routeEnd

增加路段：newStart-routeEnd

开始

输入要修改的公交线路

输入要修改的路线起始站点、目标站点

遍历需要修改的路段数changeNum

i=1;

i<=changeNum?

YES

YES

YES

YES

YES

YES

NO

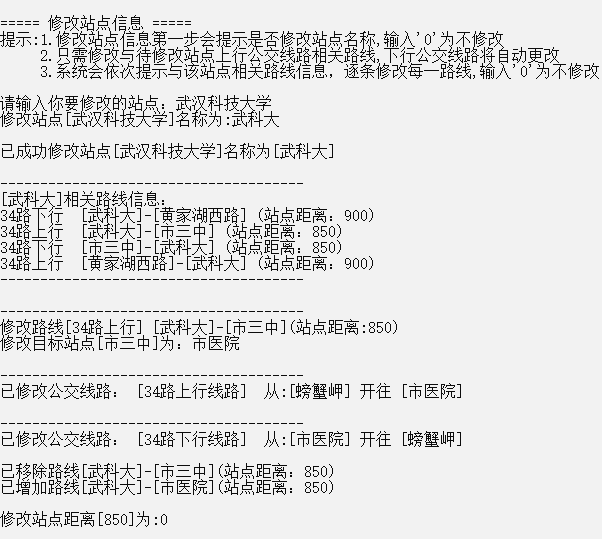
NO

NO

NO

流程图11 修改路段信息ChangeRouteToMap

**（2）修改站点信息**

修改路段信息操作编号为11，功能实现思路：系统首先提示用户输入要修改的站点，首先修改站点名称，如若修改站点名则会更新所有与该站点相关信息，然后显示与该站点相关所有路段，系统依次提示用户修改以该站点为起始站点的路段和以该站点为目标站点的路段，最后更新下行路线。

功能图11 修改站点信息

流程图12 修改站点信息ChangeStationToMap

开始

输入要修改的站点

是否修改站点名

修改站点名称

修改与该站点相关所有信息

以该站点为起始站点路段是否遍历完

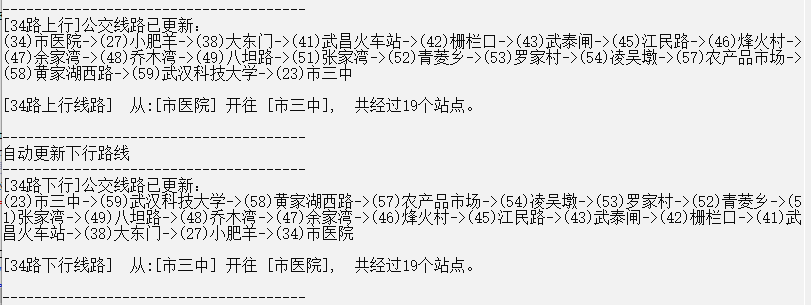
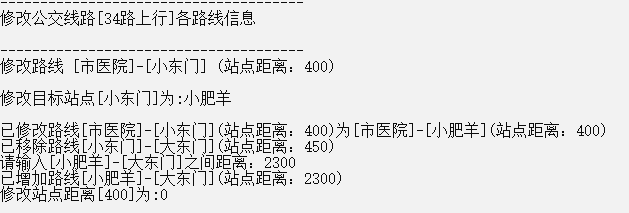
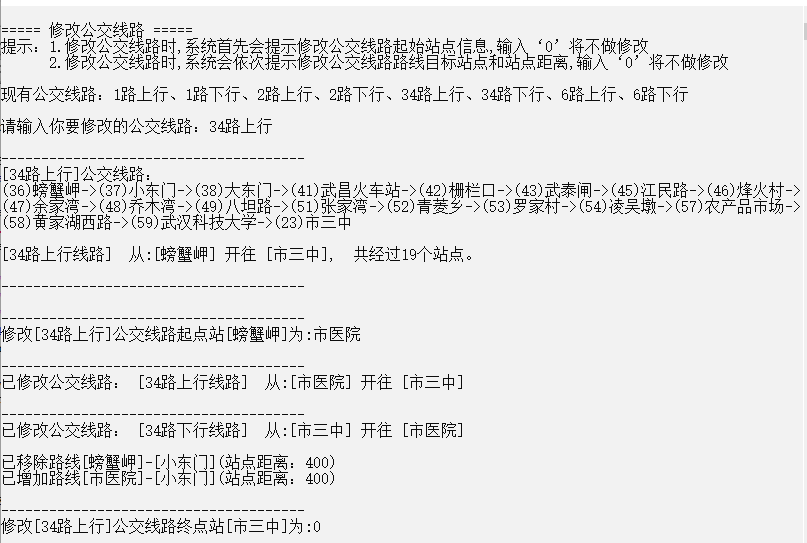
修改每条线路并更新下行公交路线

以该站点为目标站点路段是否遍历完

结束

**（3）修改公交线路**

修改公交线路操作编号为10，功能实现思路为：首先修改公交起点站、终点站、然后依次修改公交线路中各路段。



依次修改公交线路各路段

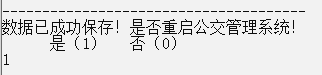
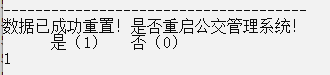
**… …**

功能图12 修改公交线路

**十、保存、重置、退出系统**

**（1）保存系统**

在对公交系统进行增加、删除、修改等操作后，系统会提供功能13：保存公交管理系统，将修改后的数据保存到文件中，即修改buses.txt、stations.txt、routes.txt三个文本文件中，实现思路为：在增删改三个功能实现中，修改邻接表的同时也修改了BUSES、ROUTES、STATIONS三个数组，只需将三个数组内容依次写入相应文件，即可实现保存功能。



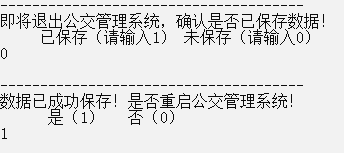
功能图14 重置公交管理系统

功能图13 保存公交管理系统

**（2）重置系统**

在对公交系统进行增加、删除、修改等操作后，系统会提供功能14：重置公交管理系统，恢复到初始数据状态，实现思路为：有三个备份bus.txt、stations.txt、routes.txt文件，只需将三个文件内容分别替换相应文件，即可实现重置功能。

**（3）退出系统**

在对公交系统进行查询、增加、删除、修改等操作后，系统会提供功能0：退出公交管理系统，在退出系统时会提示用户是否保存相关数据，未保存则自动调用保存功能。

功能图15 退出公交管理系统

**十一、课程设计总结**

当写到课程设计总结这一部分时，刚好是农历三月初一，从着手准备课程设计到现在课设基本完成将近用了三个星期的时间，可以说应该在这方面花费了不少心思吧，至少写到现在我的心情是愉悦的也是满足的。初始数据准备

阶段花费了一天时间，增、删、改三个功能的实现用了将近一个星期时间，查询功能实现用了两天，代码优化耗时一天，课程设计报告书写耗时三天时间，整个课程设计进行应该是比较顺利的，在这次课设的过程中，确实学到了很多。

1. **学到了什么**

**1.整体布局重要性**

首先，这次课设让我深刻地明白了整体布局的重要性，从最基本的数据结构设计和整个系统数据存储和管理的思路，还有内部各功能之间的联系，这些都需要在动手敲代码之前想好，有一个大体的框架。之所以将这一点放在第一点，是因为在这次课设中确实栽了一个大坑。

比如：在第一天准备好数据后，没有去考虑过程序向文本读取、存取数据的情况和实现可能性，将文本中的数据按照自己的方式排版，比如空格的缩进之类，然后在写第一个功能：读取数据的时候，发现非常困难，因为空格的存在不得不写很多代码去格式数据读取到程序中，这无疑是多此一举，因而不得不又将文本数据格式恢复到基本格式，这期间浪费了不少时间。

最大的坑应该就是数据管理部分，一开始的思路是在每次进行增加、删除、修改操作时，修改邻接表的结构的同时也直接修改文件内容，这样写第一实现难度有点大，因为进行不同操作时对文件的操作都不同，而且像文件中写入指定数据是一件非常麻烦的事情（因为要考虑文件的数据类型、找到插入的位置、以及如何插入数据），在一开始实现增加功能时，可以说在实现写入文件数据时头痛了几次，不过这不是让我回头是岸的原因，边实现功能边写入数据到文件，会造成程序莫名的崩溃，这是让我在最开始的几天百思不得其解的原因，在继续写下一个功能时，就会发现程序只能一次完成一个功能，我个人觉得应该是文本的修改会对当前运行的程序造成混淆引发错误，所以不得不选择放弃，于是决定采取中间数组存储的方式管理数据：在程序中定义三个全局数组，程序运行开始将文件中数据读取到数组中，然后程序的所有功能都在邻接表和数组上面进行操作，最后程序运行结束时，将数组中数据保存到文件中即可，这样只用对文件进行一遍操作，不会出现上述问题。

所以，这次课设让我真切明白在一个工程开始下手时，应该提前将数据结构、数据管理和整个系统中数据大致流向和操作有一个整体框架，才能保证编写相关功能时得心应手。

1. **程序实用性**

可能目前资历尚浅，谈不上考虑自己程序的实用性，但是这次课程设计中确实有所领悟，这次编写一个公交管理系统，实现每一个功能都或多或少会考虑到用户体验感，比如需要用户输入的地方该怎么处理，以及怎么和用户交互使得用户能清除理解程序的意思，所以在每个功能实现之前，都会提示用户该功能实现的数据输入要求，这些要求都是在功能代码测试时一次一次发现的问题，比如增加路段信息功能，一开始想的就是直接将一个节点插入邻接表中，而后发现还需要将邻接表连接起来，进而发现如果新增路段起始站点和目标站点有不同的约束，比如如果起始站点为公交终点站，就需要修改公交信息，并提示用户这样操作会导致的结果。同样在其他的功能中都有类似的情况，需要及时对用户的请求进行一系列的提示和反馈信息，从而使程序具有更好的实用性。

1. **熟悉邻接表、栈等数据结构的操作**

本次课程设计公交线路图的存储为邻接表存储，在查询公交、站点、路线功能中更加熟悉了利用邻接表进行遍历的方法，在增加、删除、修改功能中，也熟悉了邻接表相关增加、删除、修改节点的方法，在本系统中，相关信息的增删改不是简单的操作一个节点，而是在考虑多种情况的同时还要考虑邻接表结构的完整性，因而在编程中需要不断改进，同时在实现遍历路线时也利用到了栈数据结构，对其操作和功能有更加深刻的理解。

1. **寻找有向图中两点之间所有路线算法**

在本次课程设计中，涉及最核心的算法便是4. 寻找有向图中两点之间所有路线算法，利用普遍的深度优先或者广度优先搜索算法不能保证将所有路线都输出，而搜索资料中发现大部分都是无向图中两点之间所有路线的算法，然而公交线路图是有向有环图，因而利用普遍的算法会导致程序无限循环的问题。好在经过不断的搜集资料，找到了最终问题的算法：

涉及两个定义：1.Path的定义，是一个节点和边交叠出现的序列，并且在这个序列中节点不能重复，边也不能重复。2.DAG的定义，即不存在环路的有向图，显示本公交线路图是存在环路的有向图。

因而在上述两个定义的前提下，找到了寻找两站点之间所有路线的算法，核心是引入边状态变量：只有一条边的两个顶点都没有访问过，才能判定这条边没有访问过，这样就解决了在遍历过程中因为环路而无限循环的问题以及不会出现重复线路的问题，从而问题得到解决。

1. **可以改进的地方**

**1.程序存在大量重复**

在许多功能中，其实实现的方法大同小异，也有很多大量相同的代码，比如遍历公交路线的算法，重复出现在许多功能中，但因为一些小的不同，而重复书写了大量代码。我觉得可能是因为使用的是C语言编写，因此在编写程序过程中发现两个相似功能而细节不同时只能重复书写代码，而如果用面向对象的编程方法，或许会更好，这样在设计相似功能时，直接在类的代码中进行增改即可。

**2.函数参数过于复杂**

在一开始简单功能的实现时，没有觉得把结构体的数据一个一个传入有太麻烦，因而在一开始的各项功能实现中参数的传递都是单个的变量，没有直接将整个结构体变量传入，从而导致在后面复杂功能实现时参数传递带来的痛苦，每个函数的参数七八个甚至更多，而且含义有歧义，对于别人阅读代码和自己检查代码都带来了相当大的麻烦，比如修改路线函数，既要传入原路线各项属性，又要传入新路线各项属性，以及是哪一部分需要修改的标志，传入单个变量的参数过于复杂，如果传入的参数是结构体变量，那么只需要两个参数足矣。因此这部分是一个可以改进的地方，优化函数参数的形式。

**3.功能实现的约束**

因为在设计各项功能的实现时，通常会因为很多特殊情况的处理而导致原本简单的代码变得非常冗杂，判断条件相当多，尤其是添加路段信息对新增路线的判断，长达8个if语句，这无疑是非常不易于理解的，如果能在用户输入数据的时候就进行检测会让整个程序的结构更加简易。

**附录：基本操作（增、删、改）源代码**

**1.邻接表中新增节点**

void AddNewNodeToMap(BusMap Map,int nBus,int nStart,int nEnd,int distance)

{

//插入起点的出边链域

Station \*pStation=&Map.stations[nStart];

Route \*pRoute=pStation->routes;

//创建新节点

Route \*pNewRoute=(Route\*)malloc(sizeof(Route));

pNewRoute->bus=nBus;

pNewRoute->station=nEnd;

pNewRoute->distance=distance;

pNewRoute->next=pRoute;

pStation->routes=pNewRoute;

}

**2.添加路段到ROUTES数组中相关位置**

void AddNewRouteToArray(int nBus,int nStart,int nEnd,int nDistance)

{

extern int \*ROUTES;

extern int ROUTE\_NUM;

ROUTES=(int\*)realloc(ROUTES,sizeof(int)\*(ROUTE\_NUM+1)\*4);

int i,j;

//寻找插入位置

for(i=0,j=0; i<ROUTE\_NUM; i++,j+=4)

{

if(ROUTES[j]==nBus && ROUTES[j+1]==nEnd)

{

//目标站点为当前公交线路起点站

i--;

break;

}

if(ROUTES[j]==nBus && ROUTES[j+2]==nStart)

break;

}

//反向连接数组

int n=ROUTE\_NUM\*4,k;

for(k=ROUTE\_NUM-1,j=k\*4; k>i; k--,j-=4)

{

ROUTES[n]=ROUTES[j];

ROUTES[n+1]=ROUTES[j+1];

ROUTES[n+2]=ROUTES[j+2];

ROUTES[n+3]=ROUTES[j+3];

n-=4;

}

ROUTES[n]=nBus;

ROUTES[n+1]=nStart;

ROUTES[n+2]=nEnd;

ROUTES[n+3]=nDistance;

ROUTE\_NUM++;

}

**3.删除邻接表中节点**

void RemoveNodeToMap(BusMap Map,int nBus,int nStart,int nEnd)

{

//以nStart为线索修改路线

Station \*pStation=&Map.stations[nStart];

Route \*pRoute=pStation->routes;

Route \*preRoure=pRoute;

while(pRoute!=NULL)

{

if(pRoute->bus==nBus && pRoute->station==nEnd)

break;

preRoure=pRoute;

pRoute=pRoute->next;

}

if(preRoure==pRoute)

pStation->routes=pRoute->next;

else

preRoure->next=pRoute->next;

free(pRoute);

}

**4.删除ROUTES数组中路段**

void RemoveRouteFromArray(int nBus,int nStart,int nEnd)

{

extern int ROUTE\_NUM;

extern int \*ROUTES;

int i,j;

for(i=0,j=0; i<ROUTE\_NUM; i++,j+=4)

{

if(ROUTES[j]==nBus && ROUTES[j+1]==nStart && ROUTES[j+2]==nEnd)

break;

}

for(; i<ROUTE\_NUM; i++,j+=4)

{

ROUTES[j]=ROUTES[j+4];

ROUTES[j+1]=ROUTES[j+5];

ROUTES[j+2]=ROUTES[j+6];

ROUTES[j+3]=ROUTES[j+7];

}

ROUTE\_NUM--;

ROUTES=(int\*)realloc(ROUTES,sizeof(int)\*ROUTE\_NUM\*4);

}

**5.修改邻接表中节点**

void ChangeNodeToMap(BusMap &Map,int nBus,int nStart,int changeEnd,int changeDistance)

{

//以nStart为线索修改路线

Station \*pStation=&Map.stations[nStart];

Route \*pRoute=pStation->routes;

while(pRoute!=NULL)

{

if(pRoute->bus==nBus)

{

pRoute->station=changeEnd;

pRoute->distance=changeDistance;

break;

}

pRoute=pRoute->next;

}

}

**6.寻找两站点之间所有路线**

int QueryRoute(BusMap &Map,int nStartStation,int nEndStation)

{

//初始化path 路径数组

//path结构为 bus-station-bus-station

int i;

int initialPathNum=3,initialPathStationNum=20;

int \*\*path=(int\*\*)malloc(sizeof(int\*)\*initialPathNum);

int \*pathStationNum=(int\*)malloc(sizeof(int)\*initialPathNum);

for(i=0; i<initialPathNum; i++)

path[i]=(int\*)malloc(sizeof(int)\*initialPathStationNum\*2);

path[0][0]=-1;

path[0][1]=nStartStation;

//初始化站点栈辅助数组

int \*stationInStack=(int\*)malloc(sizeof(int)\*Map.station\_num);

for(i=0; i<Map.station\_num; i++)

stationInStack[i]=0;

//初始化route status

Station \*pStation;

Route \*pRoute;

for(i=0; i<Map.station\_num; i++)

{

pStation=&Map.stations[i];

pRoute=pStation->routes;

while(pRoute!=NULL)

{

pRoute->staus=0;

pRoute=pRoute->next;

}

}

//初始化栈结构

Stack stationStack,busStack;

InitStack(stationStack);

InitStack(busStack);

//nStartStation入栈

PushStack(stationStack,nStartStation);

PushStack(busStack,-1);

stationInStack[nStartStation]=1;

int j,k,nStation;

int flag=0,pathNum=0;

while(EmptyStack(stationStack)==false)

{

nStation=GetTopStack(stationStack);//栈顶元素出栈

if(nStation==nEndStation)//找到一条路径

{

if(flag==0)

{

printf("\n[%s]-[%s]之间存在路线\n",Map.stations[nStart

Station].station,Map.stations[nEndStation].station);

flag=1;

}

pathNum++;

if(pathNum>initialPathNum)//扩容

{

initialPathNum+=2;

path=(int\*\*)realloc(path,sizeof(int\*)\*initialPathNum);

for(i=pathNum-1; i<initialPathNum; i++)

{

path[i]=(int\*)malloc(sizeof(int)\*initialPathStationNum\*2);

path[i][0]=-1;

path[i][1]=nStartStation;

}

pathStationNum=(int\*)realloc(pathStationNum,sizeof(int)\*initialPathNum);

}

//将栈内站点依次输入path[pathNum]数组

pathStationNum[pathNum-1]=GetStackNum(stationStack);

if(pathStationNum[pathNum-1]>initialPathStationNum)

path[pathNum-1]=(int\*)realloc(path[pathNum-1],sizeof(int)\*pathStationNum[pathNum-1]\*2);

i=pathNum-1;

path[i][0]=-1;

path[i][1]=nStartStation;

for(k=1,j=2; k<pathStationNum[pathNum-1]; k++,j+=2)

{

path[i][j]=GetStackElement(busStack,k);

path[i][j+1]=GetStackElement(stationStack,k);

}

//nEndStation出栈

PopStack(stationStack);

PopStack(busStack);

stationInStack[nEndStation]=0;

//更新route status

updateRouteStatus(Map,nEndStation,stationInStack);

}

else

{

pStation=&Map.stations[nStation];

pRoute=pStation->routes;

while(pRoute!=NULL)

{

i=pRoute->station;

if(stationInStack[i]==0 && pRoute->staus==0)

{

//入栈

PushStack(stationStack,i);

PushStack(busStack,pRoute->bus);

stationInStack[i]=1;

pRoute->staus=1;

break;

}

pRoute=pRoute->next;

}

if(pRoute==NULL)//该节点没有符合要求的后续节点

{

//nStation出栈

PopStack(stationStack);

PopStack(busStack);

stationInStack[nStation]=0;

updateRouteStatus(Map,nStation,stationInStack);

}

}

}

if(flag==1)

PrintRoute(Map,nStartStation,nEndStation,pathNum,pathStationNum,path);

else if(flag==0)

{

printf("[%s]-[%s]之间不存在路线！\n\n",Map.stations[nStart

Station].station,Map.stations[nEndStation].station);

return NONE;

}

return 1;

}