计算机学院

数据结构课程设计评分表

学号：183401050225　　**姓名：**黄远鹏　　**专业：**软件工程

题目	**交通换乘规划**		**成绩**	
序号	**评价内容**	**评价标准**	**分值**	**得分**
1	**系统分析 10 分**	对于课程设计题目要求理解准确，分析问题能力强	5	
		完成课程设计各阶段任务思路清晰	5	
2	**设计实现 40 分**	题目难度大、工作量饱满	10	
		系统功能齐备	20	
		系统界面设计合理	5	
		系统测试完备	5	
3	**报告质量 30 分**	报告内容完整，图表格式规范	10	
		报告内容条例清晰、论述充分，系统设计方案合理	20	
4	**课设答辩 20 分**	演示讲解思路清晰	10	
		设计方案理解深入	5	
		问题回答正确	5	
其它补充说明			累计得分	
指导教师签名			日期	

注：（1）成绩评定采用五级记分制:优秀(90～100 分)、良好(80～89 分)、中等(70～79 分)、及格(60～69 分)、不及格(60 分以下)。

沈阳航空航天大学

课 程 设 计 报 告

课程设计名称：数据结构课程设计

课程设计题目：交通换乘规划

学　　院：计算机学院

专　　业：软件工程

班　　级：软件**1802**

学　　号：**183401050225**

姓　　名：黄远鹏

指导教师：滕一平

沈阳航空航天大学

课程设计任务书

课程设计名称	数据结构课程设计			专业	软件工程
学生姓名	黄远鹏	班级	软件 1802	学号	183401050225
题目名称	交通换乘规划				
起止日期	2019 年 12 月 9 日起至 2019 年 12 月 20 日止				

课设内容和要求：

某城市的公共交通由地铁和公交车组成，其中公交车分为空调车（每次 2 元钱）和普通车（每次 1 元）两种，地铁票价计算方式为 0~7 站 3 元，7~14 站 5 元，大于 14 站 8 元，请设计程序规划一个交通工具换乘方案，具体要求如下：

1. 设计合适的数据结构存储站点信息和连接关系信息；
2. 交通图至少应有 20 个结点代表各站点信息；
3. 能够保留各站点之间的交通方式以及预计通行时间；
4. 设计不止一种个性化的换乘方案，如时间最短、换乘次数最少、花费最小等。

参考资料：

[1] 严蔚敏,吴伟民.数据结构[M].北京:清华大学出版社,2012

[2] 吕国英.算法设计与分析[M].北京:清华大学出版社,2006

[3] 徐宝文 李志. C 程序设计语言[M].北京: 机械工业出版社,2004

[4] Erich Gamma,Richard Helm. 设计模式（英文版）[M].北京:机械工业出版社, 2004

课程负责人审核意见：□同意 □不同意　　课程负责人签字（盖章）：

指导教师（签名）	年	月	日
学生（签名）	年	月	日

目　录

1 题目介绍

1.1 题目内容和要求

某城市的公共交通由地铁和公交车组成，其中公交车分为空调车（每次 2 元钱）和普通车（每次 1 元）两种，地铁票价计算方式为 0~7 站 3 元，7~14 站 5 元，大于 14 站 8 元，请设计程序规划一个交通工具换乘方案，具体要求如下：

1. 设计合适的数据结构存储站点信息和连接关系信息；
2. 交通图至少应有 20 个结点代表各站点信息；
3. 能够保留各站点之间的交通方式以及预计通行时间；
4. 设计不止一种个性化的换乘方案，如时间最短、换乘次数最少、花费最小等。

1.2 问题分析

- 设计合适的数据结构存储站点信息和连接关系信息；交通图至少应有 20 个结点代表各站点信息

  考虑到一般的公交和地铁线路都是双向的，可以建立一个带权无向图，用顶点表示站点，边表示从一个站点到另一个站点有直达线路，边上的权值表示从一个站点到另一个站点所需付出的代价。这里假设图中没有重边。

  在实际的交通线网中，边的个数与顶点的个数相比不会太少，交通线网应该是个稠密图，所以在图的实际存储中可采用邻接矩阵的存储结构。

- 能够保留各站点之间的交通方式以及预计通行时间；设计不止一种个性化的换乘方案，如时间最短、换乘次数最少、花费最小等

  根据题目要求，从实际情况出发考虑，应该满足用户的各种不同需求，如：(1) 仅考虑地铁线路，给出任意两地铁站之间的乘车方案；(2) 同时考虑地铁和公交线路，给出乘车方案。乘车方案应当包括具体的乘车路线（途经的所有站点以及乘坐的具体线路）、总用时以及乘车费用。

2　系统功能分析

总体设计思路如下：

(1) 首先要求用户加载地图文件，根据地图文件的内容在系统中建立图的存储结构。

(2) 用户输入起点和终点，并选择具体的乘车方案类型（时间最短或只坐地铁）。

(3) 根据用户的选择计算具体的乘车方案并输出相关信息。

系统应有一定的纠错性，比如用户输入了一个不存在的站点，系统能够输出提示信息。

系统总流程如图 2.1 所示。

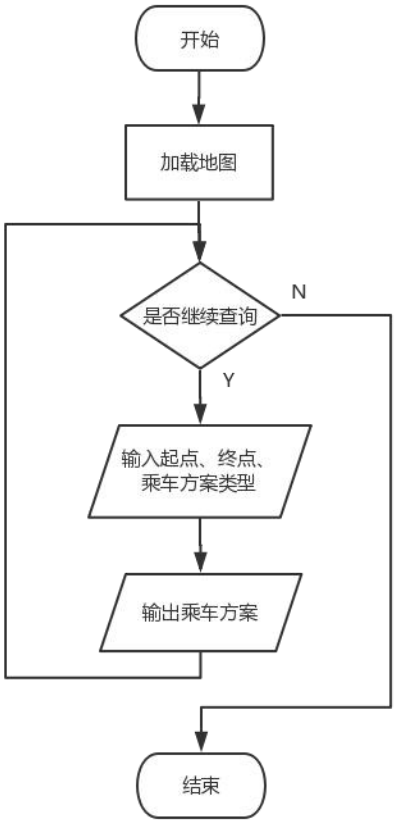

图 2.1　系统总流程

3　数据结构设计

3.1　边的数据结构

边结构体 edge 包含两部分：

- 数据部分：包括一个字符串 name 和车辆属性标志 type。name 用来存放这一条边所对应的公交（或地铁）的线路名称。车辆属性标志 type 用于区分不同的车辆类型，是自定义的枚举型变量：$\text{vehicle\_type}=\begin{cases}0, & \text{subway}\\ 1, & \text{normal\_bus}\\ 2, & \text{air\_bus}\\ \text{INF}, & \text{none}\end{cases}$，其中 INF 表示无穷大，在实际存储中可以用一个比较大的整型数字表示（如 0x3f3f3f3f）。
- 操作部分：包括一个无参构造函数和一个带参构造函数。

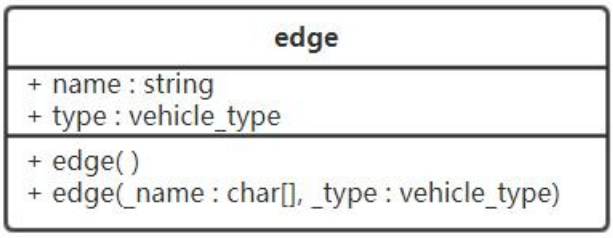

图 3.1　边的数据结构

3.2　图的数据结构

图类 graph 包含两部分：数据部分（见表 3.1）和操作部分（见表 3.2）。

表 3.1　graph 的数据部分

标识符	说明
vertexNum	总顶点数，初始值为 0
vertices	存放顶点信息，即每个站点的名称
index	从具体的顶点信息到 vertices 中下标值的映射（用 hash 表实现）
adjM	邻接矩阵，存放边的具体信息

表 3.2　graph 的操作部分

函数名	说明
insertVertex	功能：在图中添加一个顶点 参数：一个字符串，代表站点名 返回值：bool 类型，添加成功返回 true，添加失败返回 false
graph	功能：无参构造函数
isVertexInGraph	功能：判断一个站点是否已经在图里 参数：一个字符串，代表站点名 返回值：bool 类型，已经在图里返回 true，不在图里返回 false
insertEdge	功能：在图中添加一条边 参数：两个字符串，代表边的两个顶点；一个 edge 结构体，代表边的信息 返回值：bool 类型，添加成功返回 true，添加失败返回 false
lessTime	功能：输出“时间短”换乘方案 参数：两个字符串，分别代表起点和终点
findMin	功能：lessTime 的辅助函数，用于在 Dijkstra 算法中寻找没有确认最短路径的顶点中到达耗时最小的 参数：两个数组，对应 Dijkstra 算法中的距离数组和标记数组（详见 4.3.1 节） 返回值：如果找到满足要求的顶点，返回这个顶点对应的下标；否则返回-1
subwayOnly	功能：输出“只坐地铁”换乘方案 参数：两个字符串，分别代表起点和终点

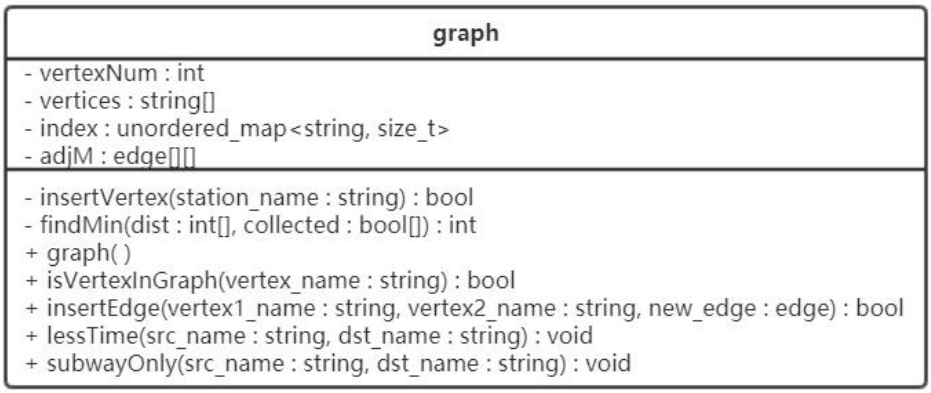

图 3.2　图的数据结构

4　功能模块设计

4.1　加载地图

4.1.1　主要函数

函数原型：`void loadMap()`

算法描述：通过调用 `insertEdge` 函数，以在图中插入边的方式建立起交通线网图。流程图见图 4.1。

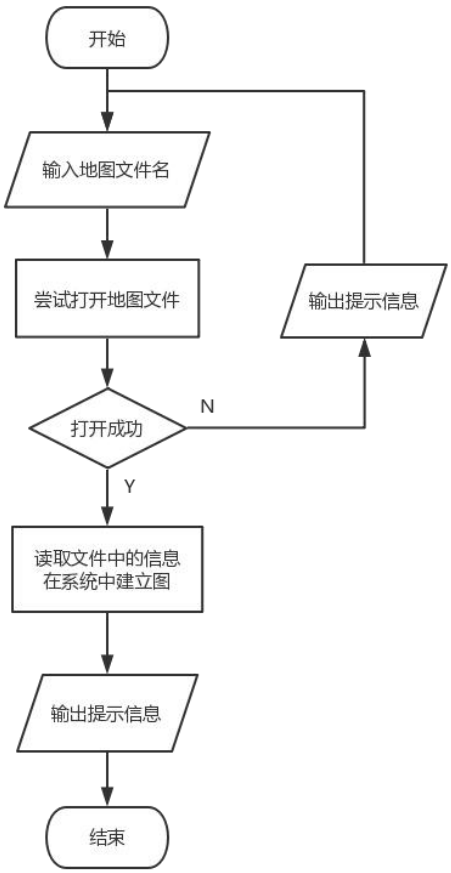

图 4.1　加载地图的流程图

4.1.2　调用的几个子函数

1. 插入边的函数

函数原型：`bool graph::insertEdge(string vertex1_name, string vertex2_name, edge new_edge)`

算法描述：先在图中插入两个顶点 vertex1 和 vertex2，再查询 vertex1 到 vertex2 是否有边相连。如果有边相连，则不需插入边，返回 false；如果没有边，则根据给定的边的信息添加边，并返回 true。

2. 查询某顶点在图中是否存在的函数

函数原型：`bool graph::isVertexInGraph(string vertex_name)`

算法描述：查找图中是否有与 vertex_name 同名的顶点，如果有返回 true，否则返回 false。

3. 插入顶点的函数

函数原型：`bool graph::insertVertex(string vertex_name)`

算法描述：先查找欲插入的顶点在图中是否存在，如果存在则无需插入，返回 false；若不存在则插入到图中，并返回 true。

4.2 输入起点、终点、乘车方案类型

4.2.1 输入起点

函数原型：`void inputSrc(string &src)`

算法描述：

(1) 输入起点。

(2) 在图中查找该顶点是否存在（调用 graph 类的 isVertexInGraph 函数）。

(3) 如果该顶点不存在，提示用户重新输入，转到(1)；如果存在，算法结束。

4.2.2 输入终点

函数原型：`void inputDst(string &dst, const string &src)`

算法描述：

(1) 输入终点。

(2) 在图中查找该顶点是否存在（调用 graph 类的 isVertexInGraph 函数）。

(3) 如果该顶点不存在或输入的终点与起点相同，提示用户重新输入，转到(1)；如果存在，算法结束。

4.2.3 选择乘车方案类型

输入乘车方案类型对应的代码，如果输入正确，调用相应的函数输出乘车方案；如果输入错误则输出提示信息。

4.3 输出乘车方案

4.3.1 输出“时间短”换乘方案

1. 主要函数

函数原型：`void graph::lessTime(string src_name, string dst_name)`

算法描述：用 Dijkstra 算法计算起点到其余各点的最短路径。算法中定义 time 数组记录从起点到其余各点的单源最短路径长度；path 数组记录最短路径经过的顶点，例如 path[i]=j 是指 j 是从起点到 i 的单源最短路径上 i 的直接前趋元素。

查询起点到终点是否存在最短路径，如果存在则输出该路径，并给出总用时与费用信息；如果不存在最短路径（起点到终点不可达）则输出提示信息。主要算法的流程图见图 4.2。算法中总用时和费用的计算基于以下假设：

(1) 相邻地铁站平均行驶时间：3 分钟。

(2) 相邻公交站平均行驶时间：7 分钟。

(3) 换乘时间不计。

(4) 地铁采用分段计价制：地铁票价 $=\begin{cases}3, & 0<\text{乘坐站数}\le 7\\ 5, & 8\le\text{乘坐站数}\le 14\\ 8, & \text{乘坐站数}>15\end{cases}$。

(5) 公交采用单一票价制：普通公交单程票价 1 元，空调公交单程票价 2 元。

时空复杂度分析：Dijkstra 算法的时间复杂度 $T(n)=O(n^2)$，空间复杂度 $S(n)=O(n)$。

2. Dijkstra 算法的子函数

函数原型：`int graph::findMin(int time[], bool collected[])`

算法描述：遍历所有顶点，寻找没有确认最短路径的顶点中到达耗时最小的。如果这样的顶点存在，返回该顶点下标；否则返回-1。

时空复杂度分析：时间复杂度$T(n)=O(n)$，空间复杂度$S(n)=O(1)$。

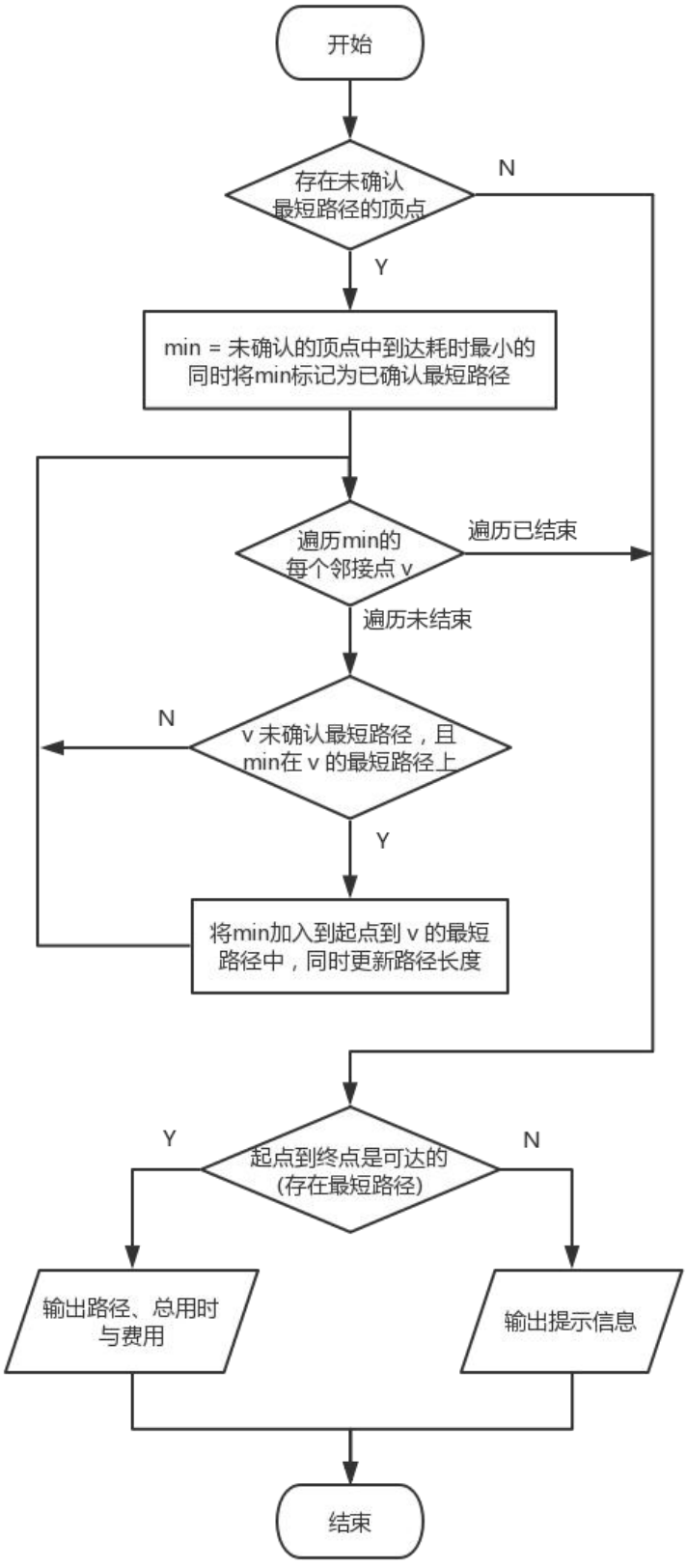

图 4.2　“时间短”换乘方案求解流程图

4.3.2 输出“只坐地铁”换乘方案

函数原型：`void graph::subwayOnly(string src_name, string dst_name)`

算法描述：在“只坐地铁”的前提下，为使总乘坐站数尽可能小，可将地铁线网视为无权图求最短路径。本算法是对图的广度优先搜索算法的改写，从起点开始一层一层地遍历每个顶点，更新它们的最短路径。查询起点到终点是否存在最短路径，如果存在则输出该路径，并给出总用时与费用信息；如果不存在（起点到终点不可达）则输出提示信息。主要算法的流程图见图 4.3。算法中总用时和费用的计算基于与 4.3.1 节相同的假设。

时空复杂度分析：时间复杂度 $T(n)=O(n^2)$，空间复杂度 $S(n)=O(n)$。

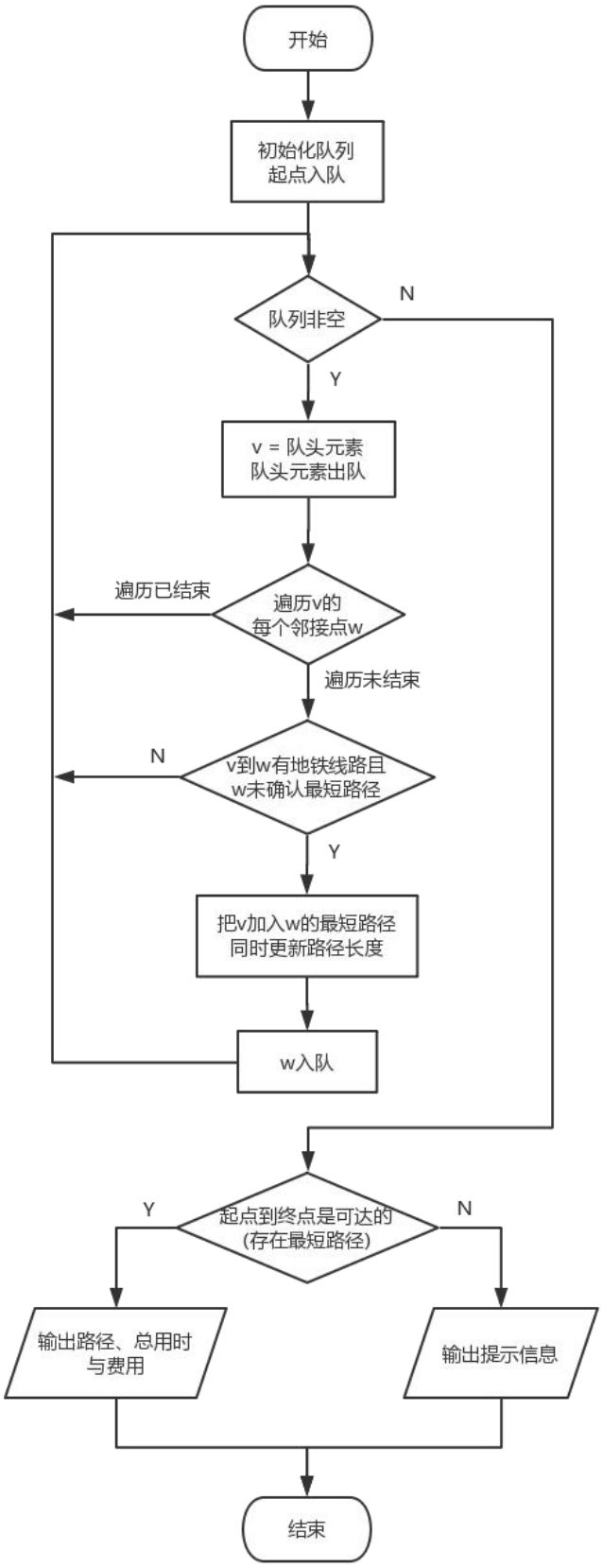

图 4.3 “只坐地铁”换乘方案求解流程图

5　系统测试与运行结果

5.1　调试及调试分析

进入系统后，首先要求用户加载地图文件，根据地图文件的内容在系统中建立图的存储结构。如果输入的文件名对应文件不存在，或文件名非法，则要求用户重新输入，直至输入正确。

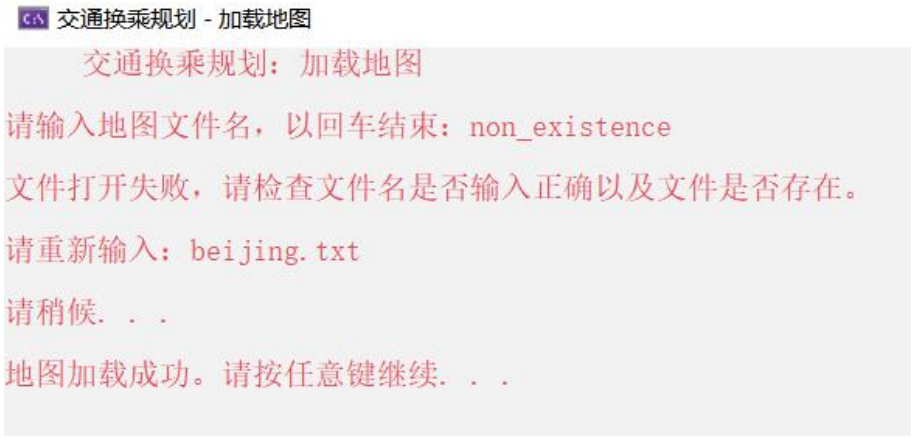

图 5.1　加载地图

成功加载地图之后，用户可以输入起点和终点，并选择需要的乘车方案类型，进行具体乘车方案的查询。如果过程中用户的输入不正确，系统会给出提示信息。

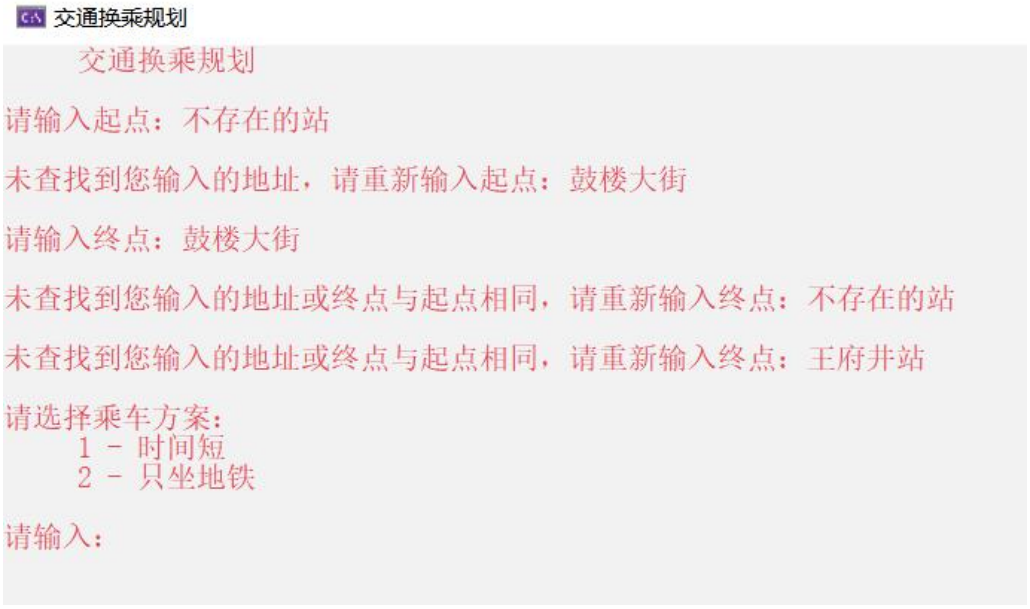

图 5.2　输入起点、终点、乘车方案类型

```
交通换乘规划 - 时间最短的乘车方案
从 鼓楼大街 到 王府井 时间最短的乘车方案：

鼓楼大街（地铁2号线） -> 安定门 -> 雍和宫（地铁5号线） -> 王府井
总用时：9分钟，费用：3元

请按任意键继续. . .
```

(a) 时间最短的乘车方案

```
交通换乘规划 - 只坐地铁的乘车方案
从 天坛 仅乘坐地铁无法到达 奥体东门，建议您更换乘车方案或采用其它出行方式。

请按任意键继续. . .
```

(b) 只坐地铁的乘车方案（方案不存在）

图 5.3　输出具体的乘车方案

5.2　测试用例

本系统选取北京市公共交通线网作为测试样本，包括 3 条地铁线路和 2 条公交线路，共 29 个站点。测试用例见表 5.1，具体的测试样本见附录 B。

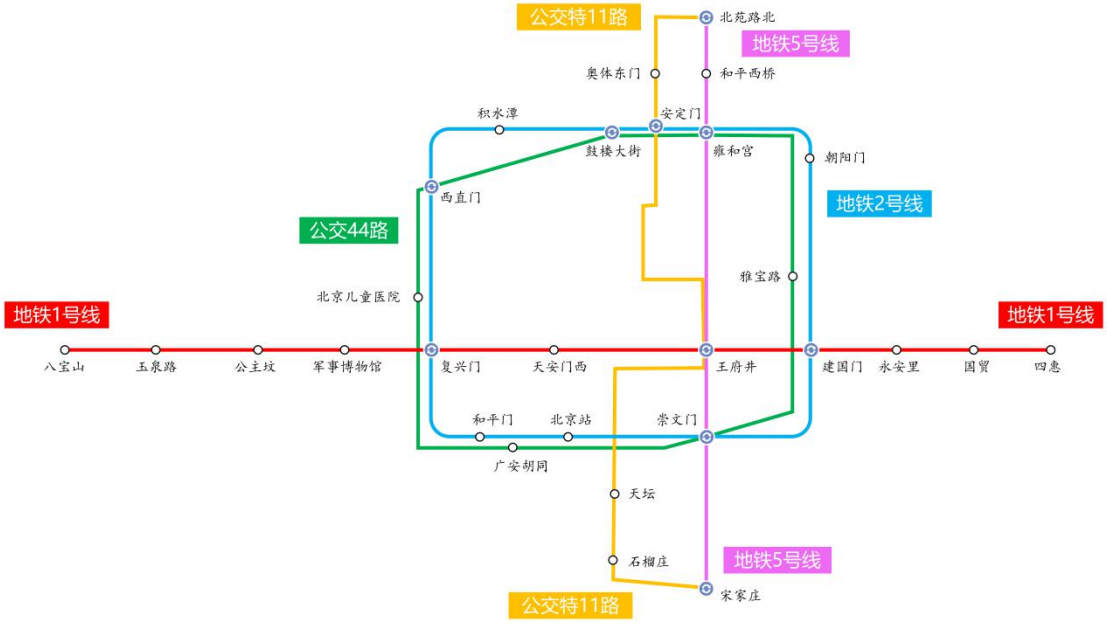

图 5.4　测试样本

表 5.1　测试用例

#	输入	样例说明	运行结果
1	起点：雅宝路 终点：广安胡同 方案代码：1	仅乘坐公交，无须换乘	从 雅宝路 到 广安胡同 时间最短的乘车方案： 雅宝路（公交44路） -> 崇文门 -> 广安胡同 总用时：14分钟，费用：1元
2	起点：四惠 终点：宋家庄 方案代码：1/2	仅乘坐地铁，无须换乘	从 公主坟 到 建国门 时间最短的乘车方案： 公主坟（地铁1号线） -> 军事博物馆 -> 复兴门 -> 天安门西 -> 王府井 -> 建国门 总用时：15分钟，费用：3元 从 公主坟 到 建国门 只坐地铁的乘车方案： 公主坟（地铁1号线） -> 军事博物馆 -> 复兴门 -> 天安门西 -> 王府井 -> 建国门 总用时：15分钟，费用：3元
3	起点：四惠 终点：宋家庄 方案代码：1/2	地铁间的换乘	从 四惠 到 宋家庄 时间最短的乘车方案： 四惠（地铁1号线） -> 国贸 -> 永安里 -> 建国门（地铁2号线） -> 崇文门（地铁5号线） -> 宋家庄 总用时：15分钟，费用：3元 从 四惠 到 宋家庄 只坐地铁的乘车方案： 四惠（地铁1号线） -> 国贸 -> 永安里 -> 建国门（地铁2号线） -> 崇文门（地铁5号线） -> 宋家庄 总用时：15分钟，费用：3元

#	输入	样例说明	运行结果
4	起点：北苑路北 终点：北京儿童医院 方案代码：1	公交和地铁间的换乘	从 北苑路北 到 北京儿童医院 时间最短的乘车方案： 北苑路北（地铁5号线） -> 和平西桥 -> 雍和宫 -> 王府井（地铁1号线） -> 天安门西 -> 复兴门（地铁2号线） -> 西直门（公交44路） -> 北京儿童医院 总用时：25分钟，费用：4元
5	起点：天坛 终点：奥体东门 方案代码：2	仅乘坐地铁不可达	从 天坛 仅乘坐地铁无法到达 奥体东门，建议您更换乘车方案或采用其它出行方式。

6 总结及建议

6.1 总结与体会

本次《数据结构课程设计》历时两周，经历了需求分析、概要设计、各模块详细设计、功能测试、报告撰写等几个环节。这一次的课程设计是我将书本知识运用于实际的过程，通过运用经典算法解决实际问题，加深了我对经典算法的理解，提升了解决实际问题的能力。

由于时间和能力有限，这个系统中还存在着一些不足之处，系统功能有待扩充，鲁棒性也有待提升。但我相信随着今后的深入学习，一定可以弥补这些不足之处。我将在追求完美的路上不断前行，不断鞭策自己努力学习，提高自己的程序设计能力。

6.2 系统改进建议

这个系统还存在提升的空间。例如：

(1) 系统可以提供更多可供选择的换乘方案，比如现在百度地图的公交导航就提供了“时间短”、“少换乘”、“少步行”、“地铁优先”、“不坐地铁”五种方案。

(2) 系统还可以增加更多查询功能，比如单个站点的信息查询（途经路线，周边推荐等）、特定线路的信息查询（途经站点、首末车时间、票价）等。

要完成这些功能可能要对系统做一定改造，比如不采用无向图，而是采用有向图的逻辑结构，同时将一条公交（或地铁）线路拆分成上行/下行（或内/外环）两条。

6.3 致谢

在我学习的道路上，遇到了很多认真、负责的老师。在他们的引领之下，我走进了计算机世界的大门。《高级程序设计》的任课教师许莉，是我程序设计的引路人。在她的悉心教导下，我练就了扎实的程序设计能力，初步建立起面向对象的程序设计思想并运用于实际。

武卫东老师给我们讲授了《数据结构与算法》这门课程。通过他在课堂上认真、细致的讲授，我感受到了数据结构与算法之美，也坚定了自己今后努力学习、研究算法的决心。

滕一平老师是我这一次数据结构课程设计的指导老师，他在程序设计以及报告的撰写方面指出了我存在的问题并提出修改建议，这些建议帮助我将程序和报告打造的更加完美。

在两周的课程设计过程中，我和室友、同学们互相鼓励、互相督促，室友在我程序的编写过程中也给了一些建议。课程设计过程中参考了一些资料，这些资料给我带来了一些启发。

最后，同时也是最应该感谢的是我的父母。他们培育我茁壮成长，付出了许多心血。我一定会努力学习，不辜负他们的期望。

参考文献

[1] 严蔚敏, 吴伟民. 数据结构[M]. 北京: 清华大学出版社, 2012

[2] 吕国英. 算法设计与分析[M]. 北京: 清华大学出版社, 2006

[3] 徐宝文, 李志. C 程序设计语言[M]. 北京: 机械工业出版社, 2004

[4] Erich Gamma, Richard Helm. 设计模式（英文版）[M]. 北京: 机械工业出版社, 2004

[5] 陈越. 数据结构. 第 2 版[M]. 北京: 高等教育出版社, 2016

附　录

附录 A　关键部分程序清单

```
/* ************************************************
 *                  数据结构课程设计
 *    题    目  交通换乘规划
 *    姓    名  黄远鹏      学    号  183401050225
 *    班    级  软件 1802
 *    指导教师  滕一平
 * ************************************************ */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <iostream>
#include <unordered_map>
#include <string>
#include <stack>
#include <queue>
using namespace std;

#define MAXSIZE 100   // 最大顶点数
#define INF 0x3f3f3f3f   // 无穷大
enum vehicle_type { subway, normal_bus, air_bus, none = INF };

struct edge
{
    string name;
    vehicle_type type;
    edge():type(none) {}
    edge(char _name[], vehicle_type _type):name(_name), type(_type) {}
};

class graph
{
private:
    int vertexNum;   // 总顶点数
    string vertices[MAXSIZE];   // 顶点信息
    unordered_map<string, size_t> index;   // 根据顶点信息找到对应下标
```

```
    edge adjM[MAXSIZE][MAXSIZE];   // 邻接矩阵
    bool insertVertex(string station_name);   // 增加顶点
    int findMin(int dist[], bool collected[]);   // lessTime()的辅助函数
public:
    graph():vertexNum(0) {};
    bool isVertexInGraph(string vertex_name);   // 判断某点是否在图内
    bool insertEdge(string vertex1_name, string vertex2_name, edge new_edge);
// 插入边
    void lessTime(string src_name, string dst_name);   // 输出"时间短"换乘
方案
    void subwayOnly(string src_name, string dst_name);   // 输出"只坐地铁"
换乘方案
};

bool graph::isVertexInGraph(string vertex_name)
{
    return (index.find(vertex_name) != index.end());
}
bool graph::insertVertex(string vertex_name)
{
    if(isVertexInGraph(vertex_name))   // 已有该顶点，无需插入
        return false;
    vertices[vertexNum] = vertex_name;   // 增加一个顶点并初始化相关信息
    index[vertex_name] = vertexNum;
    vertexNum++;
    return true;
}
bool graph::insertEdge(string vertex1_name, string vertex2_name, edge
new_edge)
{
    insertVertex(vertex1_name);
    insertVertex(vertex2_name);
    size_t vertex1_index = index[vertex1_name], vertex2_index =
index[vertex2_name];
    if(adjM[vertex1_index][vertex2_index].type != none)   // 已有这条边，无
需插入
        return false;
    adjM[vertex1_index][vertex2_index] = new_edge;   // 插入新边
    adjM[vertex2_index][vertex1_index] = new_edge;
    return true;
}
int graph::findMin(int time[], bool collected[])
{
```

```
    int min_index, min_time = INF;
    for(int i = 0; i < vertexNum; i++)
    {
        if(collected[i] == false && time[i] < min_time)
        {
            min_time = time[i];
            min_index = i;
        }
    }
    return min_time == INF ? -1 : min_index;   // 未找到返回-1
}
void graph::lessTime(string src_name, string dst_name)
{
    system("title 交通换乘规划 - 时间最短的乘车方案");
    size_t src = index[src_name], dst = index[dst_name];
    int time[MAXSIZE], path[MAXSIZE];
    bool collected[MAXSIZE];
    for(int i = 0; i < vertexNum; i++)
    {
        switch(adjM[src][i].type)
        {
            case subway:     time[i] = 3; break;   // 地铁 3min 走一站
            case normal_bus: time[i] = 7; break;   // 公交 7min 走一站
            case air_bus:    time[i] = 7; break;
            case none:       time[i] = none; break;   // 没有直达路径，初始化
为 none(用无穷大表示)
        }
        if(time[i] != none)   // 与起点直接相连的点
            path[i] = src;
        else
            path[i] = -1;
        collected[i] = false;
    }
    time[src] = 0;
    collected[src] = true;
    while(true)
    {
        int min = findMin(time, collected);
        if(min == -1)
            break;
        collected[min] = true;
        for(int i = 0; i < vertexNum; i++)
        {
```

```
            if(collected[i] == false && adjM[min][i].type != none)
            {
                int new_time;
                switch(adjM[min][i].type)
                {
                    case subway:     new_time = 3; break;
                    case normal_bus: new_time = 7; break;
                    case air_bus:    new_time = 7; break;
                }
                if(time[min] + new_time < time[i])
                {
                    time[i] = time[min] + new_time;
                    path[i] = min;
                }
            }
        }
    }
    if(time[dst] == INF)
    {
        printf("从 %s 到 %s 暂无换乘路径，建议您采用其它出行方式。\n\n",
            src_name.c_str(), dst_name.c_str());
        system("pause");
        return;
    }
    size_t temp = dst;
    stack<size_t> st;   // 用于暂存最短路径
    st.push(dst);
    while(path[temp] != -1)
    {
        st.push(path[temp]);
        temp = path[temp];
    }
    printf("从 %s 到 %s 时间最短的乘车方案：\n\n", src_name.c_str(),
dst_name.c_str());
    temp = st.top();
    st.pop();
    string line;
    vehicle_type type = none;
    int cost = 0, subwayStation_cnt = 0;   // 所需费用，乘坐地铁站数
    while(true)
    {
        cout << vertices[temp];
        if(st.empty() || adjM[temp][st.top()].name != line)
```

```
        {
            if(st.empty() || adjM[temp][st.top()].type != type)
            {   // 在终点处或改换交通方式时计算费用
                if(type == subway)   // 地铁按乘坐站数收费
                {
                    if(subwayStation_cnt <= 7)   // 0-7 站，3 元
                        cost += 3;
                    else if(subwayStation_cnt <= 14)   // 8-14 站，5 元
                        cost += 5;
                    else   // 大于 14 站，8 元
                        cost += 8;
                    subwayStation_cnt = 0;
                }
                else if(type == normal_bus)   // 普通公交 1 元
                    cost += 1;
                else if(type == air_bus)   // 空调公交 2 元
                    cost += 2;
            }

            if(st.empty())   // 在终点处退出循环
                break;
            line = adjM[temp][st.top()].name;   // 在换乘站更新换乘信息
            type = adjM[temp][st.top()].type;
            cout << "（" << line << "）";
        }
        if(adjM[temp][st.top()].type == subway)   // 乘坐地铁要累计站数
            subwayStation_cnt++;
        temp = st.top();
        st.pop();
        cout << " -> ";
    }
    cout << endl << "总用时：" << time[dst] << "分钟，";
    cout << "费用：" << cost << "元" << endl << endl;
    system("pause");
}
void graph::subwayOnly(string src_name, string dst_name)
{
    system("title 交通换乘规划 - 只坐地铁的乘车方案");
    size_t src = index[src_name], dst = index[dst_name];
    int dist[MAXSIZE], path[MAXSIZE];
    memset(dist, -1, sizeof(dist));
    memset(path, -1, sizeof(path));
    dist[src] = 0;
```

```
    queue<size_t> qu;
    qu.push(src);
    while(!qu.empty())
    {
        size_t t = qu.front();
        qu.pop();
        for(int i = 0; i < vertexNum; i++)
        {
            if(adjM[t][i].type == subway && dist[i] == -1)
            {
                dist[i] = dist[t] + 1;
                path[i] = t;
                qu.push(i);
            }
        }
    }
    if(dist[dst] == -1)
    {
        printf("从 %s 仅乘坐地铁无法到达 %s，建议您更换乘车方案或采用其它出行方
式。\n\n",
            src_name.c_str(), dst_name.c_str());
        system("pause");
        return;
    }
    size_t temp = dst;
    stack<size_t> st;
    st.push(dst);
    while(path[temp] != -1)
    {
        st.push(path[temp]);
        temp = path[temp];
    }
    printf("从 %s 到 %s 只坐地铁的乘车方案：\n\n", src_name.c_str(),
dst_name.c_str());
    temp = st.top();
    st.pop();
    string line;
    while(true)
    {
        cout << vertices[temp];
        if(st.empty())   // 到达终点，退出循环
            break;
        if(adjM[temp][st.top()].name != line)
```

```
        {
            line = adjM[temp][st.top()].name;   // 在换乘站更新换乘信息
            cout << "（" << line << "）";
        }
        temp = st.top();
        st.pop();
        cout << " -> ";
    }
    cout << endl << "总用时：" << dist[dst] * 3 << "分钟，";
    int cost;
    if(dist[dst] <= 7)   // 0-7站，3元
        cost = 3;
    else if(dist[dst] <= 14)   // 8-14站，5元
        cost = 5;
    else   // 大于14站，8元
        cost = 8;
    cout << "费用：" << cost << "元" << endl << endl;
    system("pause");
}

graph map;
void loadMap()   // 加载地图
{
    system("title 交通换乘规划 - 加载地图");
    printf("     交通换乘规划：加载地图\n\n");
    printf("请输入地图文件名，以回车结束：");
    char mapFileName[256];
    cin.getline(mapFileName, 256);
    FILE *fp = fopen(mapFileName, "r");
    while(fp == NULL)   // 若文件打开失败，则重新输入文件名直至打开成功
    {
        printf("\n文件打开失败，请检查文件名是否输入正确以及文件是否存在。
\n\n");
        printf("请重新输入：");
        cin.getline(mapFileName, 256);
        fp = fopen(mapFileName, "r");
    }
    printf("\n请稍候. . . \n\n");
    int n;
    fscanf(fp, "%d", &n);
    char name[256];   // 线路名称
    int num, type, isCycle;   // 线路站点数  地铁0/普通公交1/空调公交2  是否
环线
```

```
    for(int i = 0; i < n; i++)
    {
        fscanf(fp, "%s%d%d%d\n", name, &num, &type, &isCycle);
        char v1[256], v2[256];
        for(int j = 0; j < num - 1 + isCycle; j++)
        {
            fscanf(fp, "%s%s\n", v1, v2);
            map.insertEdge(v1, v2, edge(name, (vehicle_type)type));
        }
    }
    fclose(fp);
    printf("地图加载成功。");
    system("pause");
}
void inputSrc(string &src)   // 输入起点
{
    printf("\n 请输入起点：");
    getline(cin, src);
    while(!map.isVertexInGraph(src))   // 未找到这个起点
    {
        if(src.size() > 2   // 长度大于 2
            && src.substr(src.size() - 2, src.size()) == "站"   // 最后一个字是"站"
            && map.isVertexInGraph(src.substr(0, src.size() - 2)))   // 去掉"站"字后该点存在
        {
            src = src.substr(0, src.size() - 2);   // 用户输入多了一个"站"字，去掉即可
            break;
        }
        printf("\n 未查找到您输入的地址，请重新输入起点：");
        getline(cin, src);
    }
}
void inputDst(string &dst, const string &src)   // 输入终点
{
    printf("\n 请输入终点：");
    getline(cin, dst);
    while(!map.isVertexInGraph(dst) || dst == src)   // 未找到这个终点或起点与终点相同
    {
        if(dst.size() > 2 && dst.substr(dst.size() - 2, dst.size()) == "站
"
```

```
            && map.isVertexInGraph(dst.substr(0, dst.size() - 2))
            && dst.substr(0, dst.size() - 2) != src)   // 去掉"站"字后不与起
点相同
        {
            dst = dst.substr(0, dst.size() - 2);   // 用户输入多了一个"站"字，
去掉即可
            break;
        }
        printf("\n 未查找到您输入的地址或终点与起点相同，请重新输入终点：");
        getline(cin, dst);
    }
}
int main()
{
    system("color fc");
    loadMap();
    string src, dst, choice;
    while(true)
    {
        system("cls");
        system("title 交通换乘规划");
        printf("     交通换乘规划\n");
        fflush(stdin);
        inputSrc(src);
        inputDst(dst, src);
        printf("\n 请选择乘车方案：\n");
        printf("     1 - 时间短\n");
        printf("     2 - 只坐地铁\n");
        printf("\n 请输入：");
        getline(cin, choice);
        switch(choice[0])
        {
            case '1':system("cls"); map.lessTime(src, dst);     break;
            case '2':system("cls"); map.subwayOnly(src, dst);   break;

            default:printf("输入错误，请重新输入。"); system("pause"); break;
        }
    }
    return 0;
}
```

附录 B　测试样本

地图文件输入格式：

第 1 行给出总线路数 L。接下来 L 块，每一块第 1 行按照 线路名称 线路站点数 n 车辆属性 是否环线 的格式给出相关信息。其中线路名称为字符串，长度不超过 256 个字符；线路站点数为整型；车辆属性 0 代表地铁，1 代表普通公交，2 代表空调公交；是否环线取值为 0 或 1。接下来 n-1 行（非环线）或 n 行（环线），按照 站点 1 站点 2 的格式，给出该线路的所有站点以及前后站点之间的联系。

测试样本输入文件如下：

```
5
地铁 1 号线 11 0 0
八宝山 玉泉路
玉泉路 公主坟
公主坟 军事博物馆
军事博物馆 复兴门
复兴门 天安门西
天安门西 王府井
王府井 建国门
建国门 永安里
永安里 国贸
国贸 四惠
地铁 2 号线 11 0 1
鼓楼大街 安定门
安定门 雍和宫
雍和宫 朝阳门
朝阳门 建国门
建国门 崇文门
崇文门 北京站
北京站 和平门
和平门 复兴门
复兴门 西直门
西直门 积水潭
积水潭 鼓楼大街
地铁 5 号线 6 0 0
北苑路北 和平西桥
和平西桥 雍和宫
雍和宫 王府井
```

王府井 崇文门
崇文门 宋家庄
公交 44 路 7 1 1
鼓楼大街 雍和宫
雍和宫 雅宝路
雅宝路 崇文门
崇文门 广安胡同
广安胡同 北京儿童医院
北京儿童医院 西直门
西直门 鼓楼大街
公交特 11 路 7 2 0
北苑路北 奥体东门
奥体东门 安定门
安定门 王府井
王府井 天坛
天坛 石榴庄
石榴庄 宋家庄