

2007 高教社杯全国大学生数学建模竞赛

承 诺 书

我们仔细阅读了中国大学生数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们参赛选择的题号是（从 A/B/C/D 中选择一项填写）： B

我们的参赛报名号为（如果赛区设置报名号的话）：

所属学校（请填写完整的全名）： 重 庆 大 学

参赛队员（打印并签名）： 1. 熊国刚

2. 王 杰

3. 黎 明

指导教师或指导教师组负责人（打印并签名）： 龚 劬

日期： 2007 年 9 月 21 日

赛区评阅编号（由赛区组委会评阅前进行编号）：

2007 高教社杯全国大学生数学建模竞赛

编 号 专 用 页

赛区评阅编号（由赛区组委会评阅前进行编号）：

赛区评阅记录（可供赛区评阅时使用）：

评阅人										
评分										
备注										

全国统一编号（由赛区组委会送交全国前编号）：

全国评阅编号（由全国组委会评阅前进行编号）：

乘公交，看奥运

【摘要】本文要解决的问题是以即将举行的 08 年北京奥运会为背景而提出的。人们为了能现场观看奥运会，必然会面对出行方式与路线选择的问题。因此如何快速、高效地从众多可行路线中选出最优路线成为了解决此问题的关键。

鉴于公交系统网络的复杂性，我们没有采用常规的 Dijkstra 算法，而采用了高效的广度优先算法。其基本思想是从经过起（始）点的路线出发，搜寻出转乘次数不超过两次的可行路线，然后对可行解进行进一步处理。为满足不同查询者要求，我们对三个问题都分别建立了以时间、转乘次数、费用最小为目标的优化模型。

针对问题一（只考虑公汽系统），我们建立了模型一并通过 VC++ 编程得到了任意两个站点间的多种最优路线，并得出所求站点间最优路线的最优值，如下表所示：

出发站 ↓ 终点站	S3359 ↓ S1828	S1557 ↓ S0481	S0971 ↓ S0485	S0008 ↓ S0073	S0148 ↓ S0485	S0087 ↓ S3676
最短耗时 (min)	64	106	106	67	106	46
最少转乘次数 (次)	1	2	1	1	2	2
最少费用 (元)	3	3	3	2	3	3

模型二是根据问题二（同时考虑公汽和地铁系统）建立的，同样用 VC++ 编程得到所求站点间的最优路线，如下表所示：

出发站 ↓ 终点站	S3359 ↓ S1828	S1557 ↓ S0481	S0971 ↓ S0485	S0008 ↓ S0073	S0148 ↓ S0485	S0087 ↓ S3676
最短耗时 (min)	64	106	96	55	87.5	33
最少转乘次数 (次)	1	2	1	1	2	0
最少费用 (元)	3	3	3	2	3	3

对问题三（将步行考虑在内）我们建立了模型三的优化模型，然后在模型改进里又建立了图论模型。

本文的主要特点在于，所用算法的效率十分显著。在对原始数据仅做简单预处理的条件下，搜索任意站点间的最优路线所需的平均时间不超过 0.5 秒。另外，本文所建立的模型简单、所用算法比较清晰，易于程序实现，对公交线路自主查询计算机系统的实现具有现实指导作用。

关键字：转乘次数 广度优先算法 查询效率 实时系统

一 问题的重述

传承华夏五千年的文明，梦圆十三亿华夏儿女的畅想，2008 年 8 月 8 日这个不平凡的日子终于离我们越来越近了！在观看奥运的众多方式之中，现场观看无疑是最激动人心的。为了迎接 2008 年奥运会，北京公交做了充分的准备，首都的公交车大都焕然一新，增强了交通的安全性和舒适性，公交线路已达 800 条以上，使得公众的出行更加通畅、便利。但同时也面临多条线路的选择问题。为满足公众查询公交线路的选择问题，某公司准备研制开发一个解决公交线路选择问题的自主查询计算机系统。

这个系统的核心是线路选择的模型与算法，另外还应该从实际情况出发考虑，满足查询者的各种不同需求。需要解决的问题有：

1、仅考虑公汽线路，给出任意两公汽站点之间线路选择问题的一般数学模型与算法。并根据附录数据，利用模型算法，求出以下 6 对起始站到终到站最佳路线。

(1)、S3359→S1828 (2)、S1557→S0481 (3)、S0971→S0485

(4)、S0008→S0073 (5)、S0148→S0485 (6)、S0087→S3676

2、同时考虑公汽与地铁线路，解决以上问题。

3、假设又知道所有站点之间的步行时间，请你给出任意两站点之间线路选择问题的数学模型。

二 符号说明

L_i : 第 i 条公汽线路标号, $i=1,2 \dots 10400$, 当 $i \leq 520$ 时, L_i 表示上行公汽路线, 当

$i > 520$ 时, L_i 表示与上行路线 L_{i-520} 相对应的下行公汽路线;

$S_{i,g}$: 经过第 i 条公汽路线的第 g 个公汽站点标号;

T_j : 第 j 条地铁路线标号, $j=1,2$;

$D_{j,h}$: 经过第 j 条地铁线路的第 h 个地铁站点标号;

LS_n : 转乘 n 次的路线;

T_k : 选择第 k 种路线的总时间;

$N1_k$: 选择第 k 种路线公汽换乘公汽的换乘次数;

$N2_k$: 选择第 k 种路线地铁换乘地铁的换乘次数;

$N3_k$: 选择第 k 种路线地铁换乘公汽的换乘次数;

$N4_k$: 选择第 k 种路线公汽换乘地铁的换乘次数;

$W_{k,m}$: 第 k 种路线、乘坐第 m 辆公汽的计费方式, 其中:

$W_{k,m}=1$ 表示实行单一票价， $W_{k,m}=2$ 表示实行分段计价；

$CL_{k,m}$ ：第 k 种路线，乘坐第 m 辆公汽的费用；

C_k ：选择第 k 种路线的总费用；

$MS_{k,m}$ ：选择第 k 种路线，乘坐第 m 辆公汽需要经过的公汽站个数；

$MD_{k,n}$ ：选择第 k 种路线，乘坐第 n 路地铁需要经过的地铁站个数；

$FS_{k,m}$ ：表示对于第 k 种路线的第 m 路公汽的路线是否选择步行， $FS_{k,m}$ 为 0-1

变量， $FS_{k,m}=0$ 表示不选择步行， $FS_{k,m}=1$ 表示选择步行；

$FD_{k,n}$ ：对于第 k 种路线的第 n 路地铁的路线是否选择步行， $FD_{k,n}$ 为 0-1 变量，

$HD_{k,n}=0$ 表示不选择步行， $HD_{k,n}=1$ 表示选择步行；

三 模型假设

3.1 基本假设

- 1、相邻公汽站平均行驶时间(包括停站时间)： 3 分钟
- 2、相邻地铁站平均行驶时间(包括停站时间)： 2.5 分钟
- 3、公汽换乘公汽平均耗时： 5 分钟(其中步行时间 2 分钟)
- 4、地铁换乘地铁平均耗时： 4 分钟(其中步行时间 2 分钟)
- 5、地铁换乘公汽平均耗时： 7 分钟(其中步行时间 4 分钟)
- 6、公汽换乘地铁平均耗时： 6 分钟(其中步行时间 4 分钟)
- 7、公汽票价：分为单一票价与分段计价两种；
单一票价：1 元
其中分段计价的票价为：0 ~20 站：1 元
21~40 站：2 元
40 站以上：3 元
- 8、地铁票价：3 元（无论地铁线路间是否换乘）
- 9、假设同一地铁站对应的任意两个公汽站之间可以通过地铁站换乘，且无需支付地铁费

3.2 其它假设

- 10、查询者转乘公交的次数不超过两次；
- 11、所有环行公交线路都是双向的；
- 12、地铁线 T2 也是双向环行的；
- 13、各公交车都运行正常，不会发生堵车现象；
- 14、公交、列车均到站停车

四 问题的分析

在北京举行奥运会期间,公众如何在众多的交通路线中选择最优乘车路线或转乘路线去看奥运,这是我们要解决的核心问题。针对此问题,我们考虑从公交线路的角度来寻求最优线路。首先找出过任意两站点(公众所在地与奥运场地)的所有路线,将其存储起来,形成数据文件。这些路线可能包含有直达公交线路,也有可能是两条公交线路通过交汇而形成的(此时需要转乘公交一次),甚至更多公交线路交汇而成。然后在这些可行路线中搜寻最优路线。

对于路线的评价,我们可以分别以总行程时间,总转乘次数,总费用为指标,也可以将三种指标标准化后赋以不同权值形成一个综合指标。而最优路线则应是总行程时间最短,总费用最少或总转乘次数最少,或者三者皆有之。之所以这样考虑目标,是因为对于不同年龄阶段的查询者,他们追求的目标会有所不同,比如青年人比较热衷于比赛,因而他们会选择最短时间内到达奥运赛场观看比赛。而中年人则可能较倾向于综合指标最小,即较快、较省,转乘次数又不多。老年人总愿意以最省的方式看到奥运比赛。而对于残疾人士则总转乘次数最少为好。不同的路线查询需求用图 4.1 表示如下:

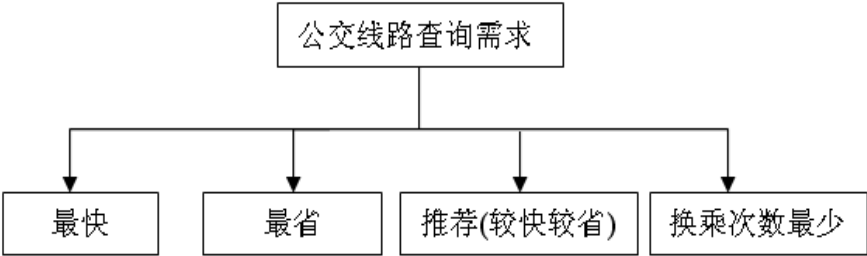


图 4.1 公交线路查询目标图

经分析,本问题的解决归结为一个求最短路径的问题,但是传统的 Dijkstra 最短路径算法并不适用于本问题,因为 Dijkstra 算法采用的存储结构和计算方法难以应付公交线路网络拓扑的复杂性,而且由于执行效率的问题,其很难满足实时系统对时间的严格要求。

为此我们在实际求解的过程中,采用了效率高效得广度优先算法,其基本思路是每次搜索指定点,并将其所有未访问过的近邻点加入搜索队列,循环搜索过程直到队列为空。此方法在后文中有详细说明。

五 建模前的准备

为了后面建模与程序设计的方便,在建立此模型前,我们有必要做一些准备工作。

5.1 数据的存储

由于所给的数据格式不是很规范,我们需要将其处理成我们需要的数据存储格式。从所给文件中读出线路上的站点信息,存入 txt 文档中,其存储格式为:两行数据,第一行表示上行线上的站点信息,第二行表示下行线的站点信息,其中下行路线标号需要在原标号的基础上加上 520,用以区分上行线和下行线。

如果上行线与下行线的站点名不完全相同,那么存储的两行数据相应地不完全相同,以公交线 L009 为例:

L009:3739 0359 1477 2159 2377 2211 2482 2480 3439 1920 1921 0180 2020 3027 2981

L529:2981 3027 2020 0180 1921 1920 3439 3440 2482 2211 2377 2159 1478

0359 3739

L529 为 L009 所对应的下行线路。

如果下行线是上行线原路返回，那么存储的两行数据中的站点信息刚好顺序颠倒，以公交线路 L001 为例：

L001:0619 1914 0388 0348 0392 0429 0436 3885 3612 0819 3524 0820 3914
0128 0710

L521:0710 0128 3914 0820 3524 0819 3612 3885 0436 0429 0392 0348 0388
1914 0619

如果是环线的情况（如图 5.1 所示），则可以等效为两条线路：

顺时针方向：S1→S2→S3→S4→S1→S2→S3→S4；

逆时针方向：S1→S4→S3→S2→S1→S4→S3→S2。

经过分析，此两条”单行路线”线路的作用等同于原环形路线

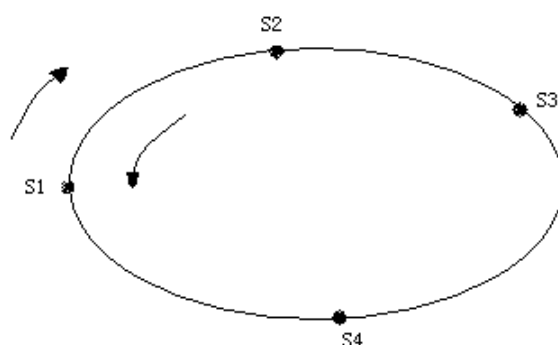


图 5.1 环行线路示意图

以环形公交线 L158 为例，此环形路线存储数据如下：

L153: 534 649 2355 1212 812 171 170 811 2600 172 1585 814 264 3513 1215
1217 251 2604 2606 534 649 2355 1212 812 171 170 811 2600 172
1585 814 264 3513 1215 1217 251 2604 2606

L673: 534 2606 2604 251 1217 1215 3513 264 814 1585 172 2600 811 170
171 812 1212 2355 649 534 2606 2604 251 1217 1215 3513 264 814
1585 172 2600 811 170 171 812 1212 2355 649

在这里，L153 被看作成上行路线，L673 被当成下行路线。这样对于每条公交线路都可以得到两行线路存储信息。

5. 2 搜寻经过每个站点的公交线路

处理 5.1 所得信息，找出通过每个站点的所有公交线路，并将它们存入数据文件中。

例如，通过搜寻得出经过站点 S0001 的线路和经过站点 S0002 的线路如下：

经过 S0001 的线路有：L421

经过 S0002 的线路有：L027 L152 L365 L395 L485

5. 3 统计任意两条公交线路的相交（相近）站点

依次统计出任意两条公交线路之间相交（相近）的站点，将其存入 1040×1040 的矩阵 A 中，但是这个矩阵的元素是维数不确定的向量，具体实现的时候可以将用队列表示。

例如：公交线路 L001 与公交线路 L025 相交的站点为 $A[1][25] = \{S0619, S1914, S0388, S0348\}$ 。

六 模型的建立与求解

6.1 模型一的建立

该模型针对问题一，仅考虑公汽线路，先找出经过任意两个公汽站点 $S_{i,g}$ 与 $S_{1,g}$ 最多转乘两次公汽的路线，然后再根据不同查询者的需求搜寻出最优路线。

6.1.1 公汽路线的数学表示

任意两个站点间的路线有多种情况，如果最多允许换乘两次，则换乘路线分别对应图 6.1 的四种情况。该图中的 A、B 为出发站和终点站，C、D、E、F 为转乘站点。

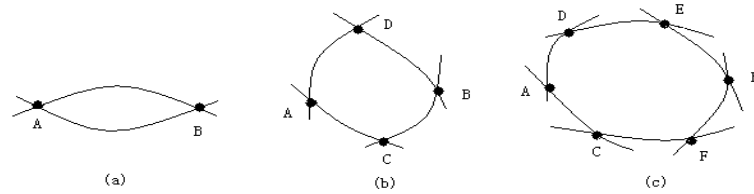


图 6.1 公汽路线图

对于任意两个公汽站点 $S_{i,g}$ 与 $S_{1,g}$ ，经过 $S_{i,g}$ 的公汽线路表示为 L_i ，有 $S_{i,g} \in L_i$ ；经过 $S_{1,g}$ 的公汽线路表示为 L_1 ，有 $S_{1,g} \in L_1$ ；

1) 直达的路线 LS_0 (如图 6.1 (a) 所示) 表示为：

$$LS_0 = \{L_i | S_{i,g} \in L_i, S_{1,g} \in L_i\}$$

2) 转乘一次的路线 LS_1 (如图 6.1 (b) 所示) 表示为：

$$LS_1 = \{L_i | S_{i,g} \in L_i, S_{1,g} \in L_j, L_i \cap L_j = \{C\}\}$$

其中：SC 为 L_{i1} ， L_{i2} 的一个交点；

3) 转乘两次的路线 LS_2 (如图 6.1 (c) 所示) 表示为：

$$LS_2 = \{L_i | S_{i,g} \in L_i, S_{1,g} \in L_j, L_i \cap L_j = \{C, D, E, F\}\}$$

通过以上转乘路线的建模过程，可以看出不同转乘次数间可作成迭代关系，进而对更多转乘次数的路线进行求寻。不过考虑到实际情况，转乘次数以不超过 2 次为佳，所以本文未对转乘三次及三次以上的情形做讨论。

6.1.2 最优路线模型的建立

找出了任意两个公汽站点间的可行路线，就可以对这些路线按不同需求进行

选择，找出最优路线了：

1) 以时间最短作为最优路线的模型：行程时间 T_k 等于乘车时间与转车时间之和。

$$\text{Min } T_k = 3 \times \sum_{m=1}^{N_k+1} (N_{k,m} - 1) + 5 \times N_k \quad (6.1 \text{ 式})$$

$$m=1, 2, \dots, N_k+1; k=1, 2, \dots, K$$

其中，第 k 路线是以上转乘路线中的一种或几种。

2) 以转乘次数最少作为最优路线的模型：

$$\text{Min } N_k \quad (6.2 \text{ 式})$$

此模型等效为以上转乘路线按直达、转乘一次、两次的优先次序来考虑。

3) 以费用最少作为最优路线的模型：

$$\text{Min } C_k = \sum_m C_{km} \quad (6.3 \text{ 式})$$

其中， $C_{km} = \begin{cases} 1 & W_{km}=1 \text{ 或 } W_{km}=2 \text{ 且 } t_{ism} \leq 0 \\ 2 & W_{km}=2, 2 \leq t_{ism} \leq 0 \\ 3 & W_{km}=2, t_{ism} > 0 \end{cases} \quad (6.4 \text{ 式})$

6. 1. 3 模型的算法描述

针对该问题的优化模型，我们采用广度优先算法找出任意两个站点间的可行路线，然后搜索出最优路线。现将此算法运用到该问题中，结合图 6.2 叙述如下：

（该图中的 $S_{i,g}$ 、 $S_{1,g}^0$ 、 $S_{1,1}$ 、 $S_{2,1}$ 、 $S_{2,2}$ 表示公汽站点， L_1 、 L_2 、 L_3 、 L_4 、 L_5 、 L_6 表示公汽线路。其中 (a)、(b)、(c) 图分别表示了从点 $S_{i,g}$ 到点 $S_{1,g}^0$ 直达、转乘一次、转乘两次的情况）

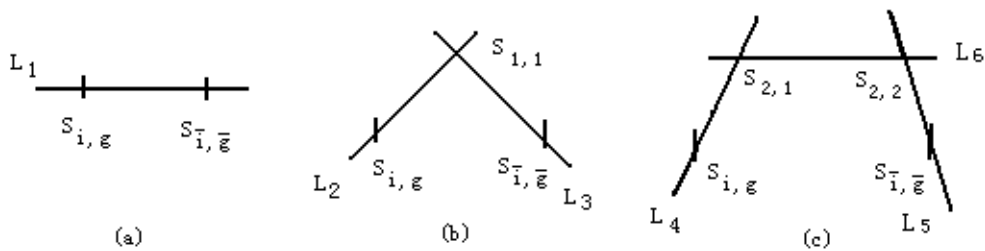


图 6.2 公交直达、转乘图

(1) 首先输入需要查询的两个站点 $S_{i,g}$ 与 $S_{1,g}^0$ （假设 $S_{i,g}$ 为起始站， $S_{1,g}^0$ 为终点站）；

(2) 搜索出经过 $S_{i,g}$ 的公汽线路 L_i ($i=1, 2, \dots, m$) 和经过 $S_{1,g}^0$ 的公汽线路 $L_{i'}^0$ ($i'=1, 2, \dots, n$)，存入数据文件；判断是 L_i 与 $L_{i'}^0$ 是否存在相同路线，若

有则站点 $S_{i,g}$ 与 $S_{i_1,g}$ 之间有直达路线(如图 6.2 中的 L_1)，则该路线是换乘次数最少(换乘次数等于 0)的路线，若有多条直达路线，则可以在此基础上找出时间最省的路线；这样可以找出所有直达路线，存入数据文件；

(3) 找出经过 $S_{i,g}$ 的公汽线路 L_i (如图 6.2 中的 L_2)中的另一站点 S_{i_1,g_1} 和经过 S_{i_1,g_1} 的公汽线路 L_{i_1} 中的另一站点 S_{i_1,g_1} 。判断 S_{i_1,g_1} 与 S_{i_1,g_1} 中是否存在相同的点，若存在(如图 6.2 中的 S_{i_1})则站点 $S_{i,g}$ 与 $S_{i_1,g}$ 间有一次换乘的路线(如图 6.2 中的 L_2 与 L_3)，该相同点即为换乘站点；这样又找出了一次换乘路线，存入数据文件；

(4) 再搜索出经过 L_i (如图 6.2 中的 L_4)线路上除了站点 $S_{i,g}$ 的另一站点 S_{i_2,g_1} (如图 6.2 中的 $S_{2,1}$)的公汽线路 L_{i_6} (如图 6.2 中的 L_6)，找出公汽线路 L_{i_6} 上的其他站点 S_{i_2,g_2} ；判断，如果 S_{i_2,g_2} 与经过 $S_{i,g}$ 的公汽线路 L_i 中的其他站点 S_{i_2,g_2} 存在相同的点(如图 6.2 中的 $S_{2,2}$)，则 $S_{i,g}$ 与 $S_{i_2,g}$ 间有二次换乘的路线(如图 6.2 中的 L_4 、 L_6 、 L_5)，该相同点和点 S_{i_2,g_1} 是换乘站点；将此二次换乘的路线存入数据文件中；

(5) 对上述存储的经过两个站点 $S_{i,g}$ 与 $S_{i_2,g}$ 的不同路线，根据不同模型进行最优路线进行搜索，得出查询者满意的最优路线。

6. 1. 4 模型一的求解

根据以上算法和前面建立的模型一，用 VC++进行编程（程序见附录）就可以得出不同目标下的最优路线。

1) 以耗时最少为目标的最优路线

起始站 S3359 到终到站 S1828 耗时最少为 64 min，耗时最少的最优路线（转乘次数较少，费用较省的路线）有 28 条(注：表 6.1 选择了其中的 10 条表示)；起始站 S1557 到终到站 S0481 耗时最少为 106 min，耗时最少的最优路线有 2 条；起始站 S0971 到终到站 S0485 耗时最少为 106 min，耗时最少的最优路线有 2 条；起始站 S0008 到终到站 S0073 耗时最少为 67 min，耗时最少的最优路线有 2 条；起始站 S0148 到终到站 S0485 耗时最少为 106 min，耗时最少的最优路线有 3 条；起始站 S0087 到终到站 S3676 耗时最少为 46 min，耗时最少的最优路线有 12 条；其耗时最少的最优路线如表 6.1 所示。

表 6.1 耗时最少的最优路线表

起始站	公汽线路	中转站	公汽线路	中转站	公汽线路	终到站	转乘次数	所需费用
S3359	L0535	S2903	L1005	S1784	L0687	S1828	2	3
S3359	L0535	S2903	L1005	S1784	L0737	S1828	2	3

S3359	L0123	S2903	L1005	S1784	L0687	S1828	2	3
S3359	L0123	S2903	L1005	S1784	L0737	S1828	2	3
S3359	L0652	S2903	L1005	S1784	L0687	S1828	2	3
S3359	L0652	S2903	L1005	S1784	L0737	S1828	2	3
S3359	L0844	S2027	L1005	S1784	L0687	S1828	2	3
S3359	L0844	S2027	L1005	S1784	L0737	S1828	2	3
S3359	L0844	S1746	L1005	S1784	L0687	S1828	2	3
S3359	L0844	S1746	L1005	S1784	L0737	S1828	2	3
S1557	L0604	S1919	L0709	S3186	L0980	S0481	2	3
S1557	L0883	S1919	L0709	S3186	L0980	S0481	2	3
S0971	L0533	S2517	L0810	S2480	L0937	S0485	2	3
S0971	L0533	S2517	L0296	S2480	L0937	S0485	2	3
S0008	L0198	S3766	L0296	S2184	L0345	S0073	2	3
S0008	L0198	S3766	L0296	S2184	L0345	S0073	2	3
S0148	L0308	S0036	L0156	S2210	L0937	S0485	2	3
S0148	L0308	S0036	L0156	S3332	L0937	S0485	2	3
S0148	L0308	S0036	L0156	S3351	L0937	S0485	2	3
S0087	L0541	S0088	L0231	S0427	L0097	S3676	2	3
S0087	L0541	S0088	L0231	S0427	L0982	S3676	2	3
S0087	L0541	S0088	L0901	S0427	L0097	S3676	2	3
S0087	L0541	S0088	L0901	S0427	L0982	S3676	2	3
S0087	L0206	S0088	L0231	S0427	L0097	S3676	2	3
S0087	L0206	S0088	L0231	S0427	L0982	S3676	2	3
S0087	L0206	S0088	L0901	S0427	L0097	S3676	2	3
S0087	L0206	S0088	L0901	S0427	L0982	S3676	2	3
S0087	L0974	S0088	L0231	S0427	L0097	S3676	2	3
S0087	L0974	S0088	L0231	S0427	L0982	S3676	2	3
S0087	L0974	S0088	L0901	S0427	L0097	S3676	2	3
S0087	L0974	S0088	L0901	S0427	L0982	S3676	2	3

2) 以转乘次数最少为目标的最优路线

起始站 S3359 到终到站 S1828 的最少转乘次数为 1 次, 转乘次数最少的最优路线(所需时间较短, 费用较省的路线)有 2 条;

起始站 S1557 到终到站 S0481 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 2 条与耗时最少的最优路线相同(表示在表 6.1 中, 下同);

起始站 S0971 到终到站 S0485 的最少转乘次数为 1 次, 转乘次数最少的最优路线有 1 条;

起始站 S0008 到终到站 S0073 的最少转乘次数为 1 次, 转乘次数最少的最优路线有 9 条;

起始站 S0148 到终到站 S0485 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 3 条与耗时最少的最优路线相同;

起始站 S0087 到终到站 S3676 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 6 条与耗时最少的最优路线相同;

其余转乘次数最少的最优路线如表 6.2 所示。

表 6.2 转乘次数最少的最优路线表

起始站	公汽线路	中转站	公汽线路	终到站	耗时	所需费用
S3359	L0956	S1784	L0687	S1828	101	3
S3359	L0956	S1784	L0737	S1828	101	3
S0971	L0533	S2184	L0937	S0485	128	3
S0008	L0679	S0291	L0578	S0073	83	2
S0008	L0679	S0491	L0578	S0073	83	2
S0008	L0679	S2559	L0578	S0073	83	2
S0008	L0679	S2683	L0578	S0073	83	2
S0008	L0679	S3614	L0578	S0073	83	2
S0008	L0875	S2263	L0345	S0073	83	2
S0008	L0875	S2303	L0345	S0073	83	2
S0008	L0875	S3917	L0345	S0073	83	2
S0008	L0983	S2083	L0057	S0073	83	2

3) 以费用最少为目标的最优路线

起始站 S3359 到终到站 S1828 的最少费用为 3 元，最少费用的最优路线（所需时间较短，转乘次数较少的路线）有 30 条，其中 28 条路线所需时间为 64 min，转乘次数为 2 次，另外两条路线所需时间为 101 min，转乘次数为 1 次；

起始站 S1557 到终到站 S0481 的最少费用为 3 元，最少费用的最优路线有 2 条，所需时间为 106 min，转乘次数为 2 次；

起始站 S0971 到终到站 S0485 的最少费用为 3 元，最少费用的最优路线有 3 条，其中两条所需时间为 106 min，转乘次数为 2 次，另外一条所需时间为 128 min，转乘次数为 1 次；

起始站 S0008 到终到站 S0073 的最少费用为 2 元，最少费用的最优路线有 9 条，所需时间为 83 min，转乘次数为 1 次；

起始站 S0148 到终到站 S0485 的最少费用为 3 元，最少费用的最优路线有 3 条，所需时间为 106min，转乘次数为 2 次；

起始站 S0087 到终到站 S3676 的最少费用为 3 元，最少费用的最优路线有 12 条，所需时间为 46 min，转乘次数为 2 次；

最少费用的最优路线表示在表 6.1 和表 6.2 中。

6. 2. 1 模型二的建立

该模型针对问题二，将公汽与地铁同时考虑，找出可行路线，然后寻找最优路线。对于地铁线路，也可以将其作为公交线路，本质上没有什么区别，只不过乘车费用、时间，换乘时间不一样罢了。因此地铁站可等效为公交站，地铁和公交的转乘站即可作为两者的交汇点。因此该模型的公交换乘路线模型与模型一中的基本相同。现建立模型二下的最优路线模型。

1) 以时间最短的路线作为最优路线的模型：可行路线的总时间为乘公交（公汽和地铁）时间与公汽与地铁换乘、公汽间、地铁间换乘时间之和。

$$T_k = \sum_{i=1}^n t_{i,i+1} + \sum_{i=1}^{n-1} t_{i,i+1}^h \quad (6.5 \text{ 式})$$

其中，第 k 路线为同时考虑公汽与地铁的转乘路线中的一种或几种。

2) 以转乘次数最少的路线作为最优路线的模型：

$$N_{ih} \in \sum_m C_{ih} \quad (6.6 \text{ 式})$$

此模型等效为以上转乘路线按直达、转乘一次、两次（包括公交与地铁间的转乘）的优先次序来考虑。

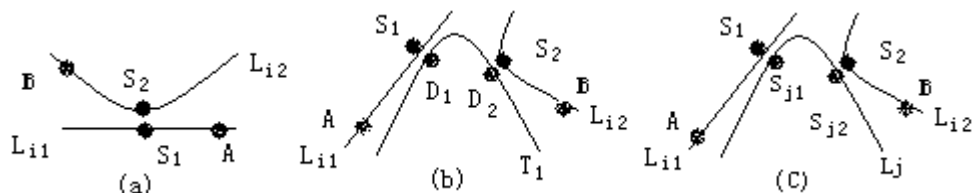
3) 以费用最少的路线作为最优路线的模型：可行路线的费用为乘公交和地铁费用的总和。

$$N_{ih} \in \sum_m C_{ih} \quad (6.7 \text{ 式})$$

其中， $CL_{k,m}$ 仍满足 (6.4 式)。

6. 2. 2 模型二的求解

不难发现，问题一是问题二解的一部分。在问题二中，新产生的最优解主要源于在通过换乘地铁、换乘附近相近站点的路线上，如下图所示：



从点 A 到 B，图(a)表示的是通过两公交线路上相邻公汽站 S1, S2 进行一次转乘；图(b)表示利用地铁站进行二次转乘；图(c)表示利用另一条公交线路为中介进行二次转乘。

铁路线路引入给题目的求解增加了难度,为了形象了解为数不多的两条铁路间的交叉关系，我们通过 matlab 编程（程序见附录）作出了两条铁路的位置关系图，如图 6.3 所示。

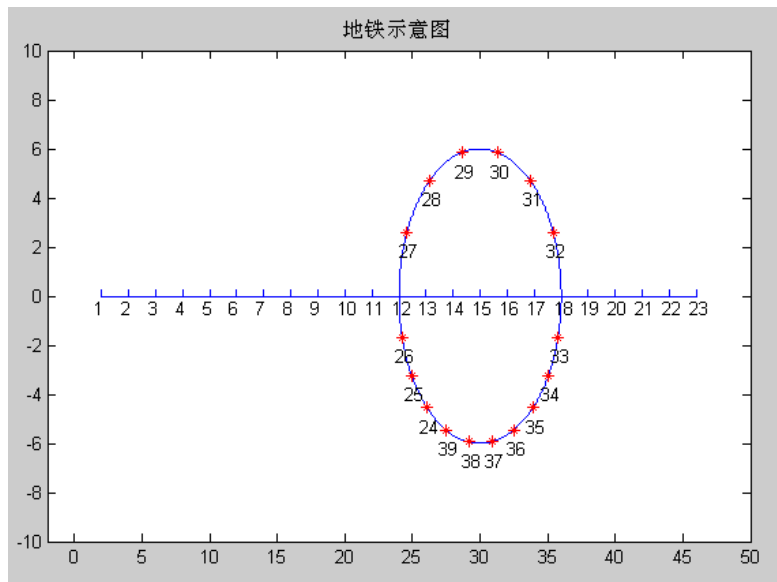


图 6.3 T1 与 T2 铁路位置关系图

注：图四中的直线表示 T1 铁路线，圆表示 T2 铁路线，数值表示站点，例如 1 表示 T1 铁路线上的 D1 铁路站，26 表示 T2 铁路线上的 D26 铁路站。此图与网上查询到的北京地铁示意图（如图 6.4 所示）相吻合。



图 6.4 北京地铁示意图

同样将地铁线路等效为公交线路得出任意两个站点间的可行线路,再将目标函数分别用模型二建立的模型表达式表达,用 VC++进行编程(程序见附录)求得考虑地铁情况的最优路线。

1) 以转乘次数最少为目标的最优路线

起始站 S0008 到终到站 S0073 的最少转乘次数为 1 次, 转乘次数最少的最优路线有 1 条:

起始站 S0087 到终到站 S3676 的最少转乘次数为 0 次, 即有直达路线, 直达下的最优路线有 1 条:

起始站 S0148 到终到站 S0485 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 10 条:

起始站 S0971 到终到站 S0485 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 20 条 (注表 6.4 中罗列其中 10 条):

起始站 S1557 到终到站 S0481 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 17 条 (注表 6.4 中罗列其中 10 条):

起始站 S3359 到终到站 S1828 的最少转乘次数为 2 次, 转乘次数最少的最优路线有 2 条。

2) 以耗时最少为目标的最优路线

起始站 S3359 到终到站 S1828 耗时最少为 64 min, 耗时最少的最优路线 (转乘次数较少, 费用较省的路线) 有 28 条 (注: 表 6.1 选择了其中的 10 条表示);

起始站 S1557 到终到站 S0481 耗时最少为 109 min, 耗时最少的最优路线有 17 条与转乘次数最少的最优路线相同;

起始站 S0971 到终到站 S0485 耗时最少为 96 min, 耗时最少的最优路线有 20 条与转乘次数最少的最优路线相同;

起始站 S0008 到终到站 S0073 耗时最少为 55 min, 耗时最少的最优路线有 3 条;

起始站 S0148 到终到站 S0485 耗时最少为 87.5 min, 耗时最少的最优路线有 10 条与转乘次数最少的最优路线相同;

起始站 S0087 到终到站 S3676 耗时最少为 33 min, 耗时最少的最优路线有 1 条与转乘次数最少的最优路线相同;

3) 最少费用的最优路线

起始站 S3359 到终到站 S1828 的最少费用为 3 元, 最少费用的最优路线 (所需时间较短, 转乘次数较少的路线) 有 2 条:

起始站 S1557 到终到站 S0481 的最少费用为 3 元, 最少费用的最优路线有

17 条；

起始站 S0971 到终到站 S0485 的最少费用为 5 元，最少费用的最优路线有 20 条；

起始站 S0008 到终到站 S0073 的最少费用为 2 元，最少费用的最优路线有 1 条；

起始站 S0148 到终到站 S0485 的最少费用为 5 元，最少费用的最优路线有 10 条；

起始站 S0087 到终到站 S3676 的最少费用为 2 元，最少费用的最优路线有 1 条；

在此种情况下，我们就只考虑可以通过地铁站换乘的情况，不通过地铁站的情况即为模型 1 的求解结果。模型 2 的求解结果见附件 1。

6. 3. 1 模型三的建立

该模型针对问题三，将步行方式考虑在了出行方式当中，更符合实际。因为当出发点与换乘点、终点站或转乘站与转乘站之间只相隔几个站时，当然该段选择步行方式更优。

因此作出如下假设：

一、如果存在某段路线，其两端点站之间相隔站点数小等于 2（即至多经过 4 个站点），则该段线路选择步行方式到达目的地。其他的情况用模型二来处理。其中路线的两端点站之间相隔站点数是根据公交直达换乘路线来确定的。

二、相邻公交站点（包括地铁站）间平均步行时间为 5 分钟。

三、如果在公汽线路上选择步行，则公汽间换乘次数减少 1；如果在地铁线路上选择步行，则地铁间换乘次数减少 1，直达线路除外。

直达和转乘一次、两次的路线需要步行的路段示意图如图 6. 5 所示。图中(a)表示出发点 A 与终点站 B 间能直达，相隔的站点数等于 2 所以选择步行；图中(b)表示出发点 A 与终点站 B 间通过一次换乘能到达，其中路段 AC 的站点数等于 2 所以选择步行，同样如果 CB 路段的站点数小等于 2，则也采取步行的方式；图中(c)选择步行方式的依据类似。

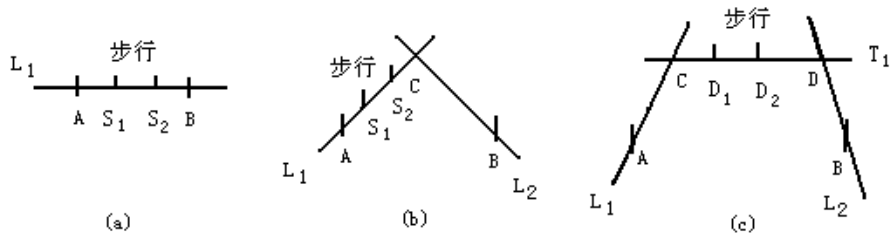


图 6. 5 步行示意图

是否选择步行方式的函数：

$$FS_{k,m} = \begin{cases} 1 & MS_{k,m} \leq 4 \\ 0 & \text{其他} \end{cases} \quad FD_{k,n} = \begin{cases} 1 & MD_{k,n} \leq 4 \\ 0 & \text{其他} \end{cases} \quad (6.8 \text{ 式})$$

其中 $FS_{k,m}$ 表示第 m 路公交线路是否步行， $FD_{k,n}$ 表示第 n 路地铁线路是否步行；

对于直达路线，如果出发点与终点站之间相隔站点数小等于 2 则步行，否则乘车。对于需要转乘的路线的最优路线模型讨论如下：

1) 以时间最短的路线作为最优路线的模型：路线总时间等于乘车时间加上步行时间，再加上转乘时间。

$$\begin{aligned}
 \text{Min } T_k &= 3 \times \sum_m (1 - FS_{k,m}) \times (MS_{k,m} - 1) + 2.5 \times \sum_n (1 - HD_{k,n}) \times (MD_{k,n} - 1) \\
 &\quad + 5 \times \sum_m FS_{k,m} \times (MS_{k,m} - 1) + 5 \times \sum_n HD_{k,n} \times (MD_{k,n} - 1) \\
 &\quad + 5 \times (N1_k - \sum_m FS_{k,m}) + 4 \times (N2_k - \sum_n HD_{k,n}) + 7 \times N3_k + 6 \times N4_k \quad (6.9 \text{ 式}) \\
 &= \sum_m (3 + 2FS_{k,m}) \times (MS_{k,m} - 1) + \sum_n (2.5 + 2.5HD_{k,n}) \times (MD_{k,n} - 1) \\
 &\quad + 5 \times (N1_k - \sum_m FS_{k,m}) + 4 \times (N2_k - \sum_n HD_{k,n}) + 7 \times N3_k + 6 \times N4_k
 \end{aligned}$$

其中，第 k 路线为同时考虑公汽与地铁的转乘路线中的一种或几种。

2) 以转乘次数最少的路线作为最优路线的模型：每步行一次就少换乘一次车。

$$\text{Min } T_k = \sum_m (1 - FS_{k,m}) \times (MS_{k,m} - 1) + \sum_n (1 - HD_{k,n}) \times (MD_{k,n} - 1) \quad (6.20 \text{ 式})$$

此模型等效为以上转乘路线按直达、转乘一次、两次、三次（包括公交与地铁间的转乘）的优先次序来考虑。

3) 以费用最少的路线作为最优路线的模型：

$$\text{Min } T_k = \sum_m (1 - FS_{k,m}) \times (MS_{k,m} - 1) + \sum_n (1 - HD_{k,n}) \times (MD_{k,n} - 1) \quad (6.21 \text{ 式})$$

其中， $CL_{k,m}$ 仍满足 (6.4 式)。

七 模型的优缺点及改进

7.1 模型的评价

7.1.1 模型优点

- 1、模型是由简单到复杂一步步建立的，使得更贴近实际。
- 2、本文的模型简单，其算法直观，容易编程实现。
- 3、本文模型比较注重数据的处理和存储方式，大大提高了查询效率。
- 4、本文模型注重效率的提高，通过大量的特征信息的提取，并结合有效的算法，使其完全可以满足实时系统的要求。

7.1.2 模型缺点

在建模与编程过程中，使用的数据只是现实数据的一种近似，因而得出的结果可能与现实情况有一定的差距。

7.2 模型的改进

以上模型主要是从公交线路出发，寻找公交线路的交叉站作为换乘站点，进而找出经过任意两个站点的可能乘车路线。我们也可以从公交站点的角度出发，用图论的方法建立有向赋权图（如图 7.1 所示），此向赋权图是针对问题三建立

的图论模型，问题一、问题二只是此模型的简化。图 7.1 中 L_i 表示公交线路标号，该线路是公交线路 L_i 的上行线或下行线， s_1 、 ggg 、 s_{j-1} 、 s_j 、 s_{j+1} 、 ggg 、 s_n 是公交线路 L_i 上的站点标号； T_k 表示地铁线路标号，该地铁线路是双向行驶的， D_1 、 ggg 、 D_g 、 D_{g+1} 、 ggg 、 D_m 是地铁线路 T_k 上的站点标号；公交 L_i 与地铁 T_k 可以在公交站 s_j 和地铁站 D_g 间换乘。如果图 7.1 中的地铁线路替换成公交线路，为了表示公交间换乘所需的时间或者费用，应将同一个换乘站点用两个站点来表示。

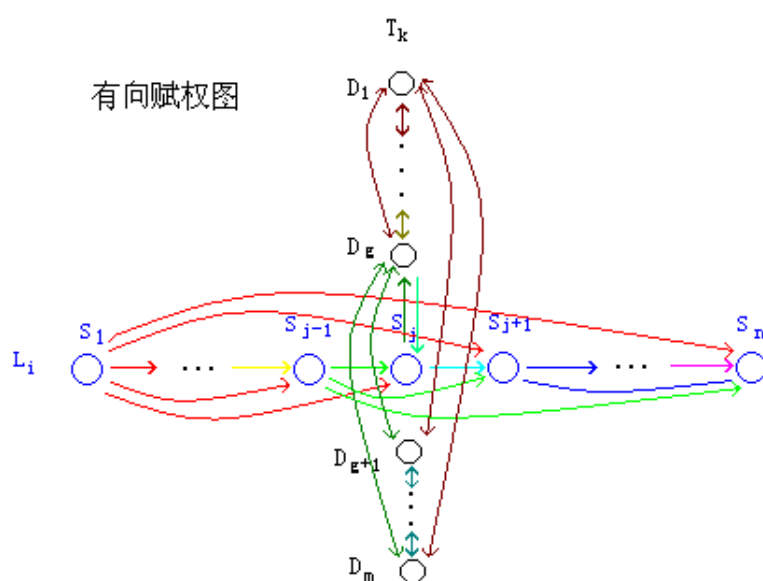


图 7.1 公交线路的有向赋权图

根据不同的目标, 给不同的站点间的边赋上不同的权值。然后利用图论的相关算法, 找出相应的最短路径。

1) 当以时间最短为目标时, 给每条边赋上时间的权值。给同一线路上任意两个站点间的边赋值时, 其权值等于站点间的公交线路段数与平均时间的乘积。当某段线路的两端点间隔站点数小于等于 3 时, 选择步行, 该线路的权值等于步行时间。不同公交和地铁间进行换乘时需要赋给不同的权值, 以表示换乘时间。


例如 (如图 7.1):

当 $j > 4$ 时, s_1 到 s_j 的边权值 $\langle s_1, s_j \rangle = 5j$;

从 s_{j-1} 到 s_j 不需要的转车, 但根据假设应选择步行, 其边权值 $\langle s_{j-1}, s_j \rangle = 5$;

从 s_{j-1} 到 D_j 要么乘公交, 然后转车, 要么步行, 根据步行的假设条件, s_{j-1}

到 D_j 的站点间隔数小于 2, 因此选择步行, 其边权值 $\langle s_{j-1}, D_j \rangle = 5$;

当 $g > 4$ 时, D_i 与 D_g 之间的边权值 ;

S_j 到 D_g 的边权值 $\langle S_j, D_g \rangle = 6$;

D_g 到 S_j 的边权值 $\langle D_g, S_j \rangle = 7$;

当 $j > 4$ 、 $g > 4$ 时, S_j 到 D_i 的路径长度为:



当 $j \leq 4$ 、 $g > 4$ 时, 则从 S_j 到 D_g 选择步行, 再乘地铁到 D_i , 其路径长度为:



找出任意两点间可行路线的路径长度后, 再搜索出其中的最短路径的可行路线作为时间的最优路线。

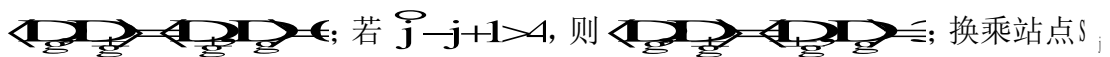
2) 当以费用最省为目标时, 则给每条边赋上费用的权值。
 公汽站点间的边权按 (6.4 式) 赋值。

当公汽线路 L_i 按单一票价计费, 对于 L_i 上任意两个公汽站点 S_j 和 S_g 间,

若 $j - j + 1 \leq 4$, 则选择步行 $\langle S_j, S_g \rangle = 0$; 若 $j - j + 1 > 4$, 则 $\langle S_j, S_g \rangle = 1$;


当公汽线路 L_i 按分段计价, 若 $j - j + 1 \leq 4$, 则 $\langle S_j, S_g \rangle = 0$; 若 $3 < j - j \leq 4$, 则 $\langle S_j, S_g \rangle = 1$; 若 $4 < j - j \leq 4$, 则 $\langle S_j, S_g \rangle = 2$; 若 $j - j > 4$, 则 $\langle S_j, S_g \rangle = 3$;

地铁线路 T_k 上任意两个站点 D_g 和 D_i 间, 若 $j - j + 1 \leq 4$, 则选择步行



与 D_g 间的边权值均为 0, 即 ; 则从 S_j 通过站点 S_j 换乘 D_g 到

D_i 的一条可行路线的路径长度为:

若 $j \leq 4$, $g > 4$, 则从 S_j 到 S_j 选择步行, ;

若 $j > 4$, $g > 4$, 则 ;

同样可以找出任意两点间可行路线的路径长度, 然后再搜索出最短路径作为费用的最优路线。

以上从公交站点出发, 将公交站点作为网络图中顶点, 得出公交的拓扑结构, 进而寻求不同目标下的最短路径, 为我们提供了另外一种思路。但是从以上图形的结构, 我们已经看得出其复杂程度是不可预知的, 尤其随着数据的增多, 图的复杂度随之上升。如果不寻求一个好的算法, 而用常规的Dijkstra算法, 将有可

能在可以忍受的时间范围内得不出有效结果。

经过参考相关资料，我们发现用蚂蚁算法可能比较有效。该算法利用了蚂蚁寻食出行路径选择的行为特点，通过线路激素强度的更新机制，实现了以换乘次数最少和公交出行站点最少的公交出行路径选择优化目标。

八 参考文献

- [1] 344000 温小文 臧德彦，城市公交信息查询系统设计初探，江西测绘，第65期，2006
- [2] 龚劬 图论与网络最优化算法 重庆大学出版社，2000
- [3] 1671-4512(2003)S1-0313 张帅 彭玉青 赵镇 李志强，蚂蚁算法在公交查询最短路径求法中的应用，华中科技大学学报（自然科学报），第31卷，2003

九 附件

换乘 0 次的情况

起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	换乘次数
S0087	L21	S0087	D27	T2	D36	S3676	L97	S3676	33	3	0

换乘 1 次的情况

起始站点: S0008 换乘站点: S0073								
起始站点	路线	站点	站点	路线	目标站点	耗时/min	车费/元	转乘次数
S0008	129	S0400	S2633	474	S0073	80	2	1
S0008	129	S0854	S0856	474	S0073	83	2	1
起始站点: S0087 换乘站点: S3676								
起始站点	路线	站点	站点	路线	目标站点	耗时/min	车费/元	转乘次数
S0087	454	S0541	S0540	462	S3676	44	2	1

换乘 2 次的情况

起始站点: S0008 目标站点: S0073											
起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	换乘次数
S0008	L150	S3874	D30	T2	D25	S0525	L103	S0073	55	5	2
S0008	L159	S3874	D30	T2	D25	S0525	L103	S0073	55	5	2
S0008	L259	S3874	D30	T2	D25	S0525	L103	S0073	55	5	2
S0008	L200	S2534	D15	T1	D12	S0609	L57	S0073	65.5	5	2
S0008	L159	S0400	S2633	L474	S0527	S0525	L103	S0073	67	3	2
起始站点: S0087 目标站点: S3676											
起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	换乘次数
S0087	L21	S0087	D27	T2	D36	S3676	L97	S3676	33	3	0
S0087	L28	S0608	D12	T2	D36	S3676	L97	S3676	33.5	5	2

S0087	L28	S0608	D12	T2	D36	S3676	L209	S3676	33.5	5	2
S0087	L28	S0608	D12	T2	D36	S3676	L209	S3676	33.5	5	2
S0087	L28	S0608	D12	T2	D36	S3676	L462	S3676	33.5	5	2
S0087	L28	S0608	D12	T2	D36	S3676	L462	S3676	33.5	5	2
S0087	L28	S0608	D12	T2	D36	S3676	L506	S3676	33.5	5	2
起始站点: S0148 目标站点: S0485											
起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	转乘次数
S0148	L24	S1487	D2	T1	D21	S0466	L51	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0464	L104	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0464	L395	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0466	L450	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0464	L469	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0466	L51	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0464	L104	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0464	L395	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0466	L450	S0485	87.5	5	2
S0148	L24	S1487	D2	T1	D21	S0464	L469	S0485	87.5	5	2
S0148	L308	S0302	S0303	L481	S2027	S2027	L469	S0485	130	3	2
起始站点: S0971 目标站点: S0485											
起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	转乘次数
S0971	L94	S0567	D1	T1	D21	S0466	L51	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0466	L51	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0464	L104	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0464	L104	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0464	L395	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0464	L395	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0466	L450	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0466	L450	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0464	L469	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0464	L469	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0466	L51	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0464	L104	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0464	L395	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0466	L450	S0485	96	5	2
S0971	L94	S0567	D1	T1	D21	S0464	L469	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0466	L51	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0464	L104	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0464	L395	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0466	L450	S0485	96	5	2
S0971	L119	S0567	D1	T1	D21	S0464	L469	S0485	96	5	2
起始站点: S1557 目标站点: S0481											
起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	转乘次数

S1557	L84	S1919	S2079	L417	S2424	S2424	L254	S0481	109	3	2
S1557	L363	S1919	S2079	L417	S2424	S2424	L254	S0481	109	3	2
S1557	L84	S1919	S2079	L417	S2424	S2424	L447	S0481	109	3	2
S1557	L363	S1919	S2079	L417	S2424	S2424	L447	S0481	109	3	2
S1557	L84	S1919	S2079	L417	S2424	S2424	L516	S0481	109	3	2
S1557	L363	S1919	S2079	L417	S2424	S2424	L516	S0481	109	3	2
S1557	L84	S1919	S2079	L417	S2424	S2424	L460	S0481	109	3	2
S1557	L84	S1919	S1920	L189	S3186	S3186	L460	S0481	109	3	2
S1557	L84	S1919	S2079	L417	S2424	S2424	L460	S0481	109	3	2
S1557	L84	S1919	S1920	L189	S3186	S3186	L460	S0481	109	3	2
S1557	L84	S1919	S2079	L417	S2424	S2424	L460	S0481	109	3	2
S1557	L84	S1919	S1920	L189	S3186	S3186	L460	S0481	109	3	2
S1557	L84	S1919	S2079	L417	S2424	S2424	L460	S0481	109	3	2
S1557	L363	S1919	S1920	L189	S3186	S3186	L460	S0481	109	3	2
S1557	L363	S1919	S2079	L417	S2424	S2424	L460	S0481	109	3	2
S1557	L363	S1919	S1920	L189	S3186	S3186	L460	S0481	109	3	2
S1557	L363	S1919	S2079	L417	S2424	S2424	L460	S0481	109	3	2
起始站点: S3359 目标站点: S1828											
起始站点	线路	站点	站点	线路	站点	站点	线路	站点	耗时/min	车费/元	转乘次数
S3359	L484	S0391	S0393	L485	S1784	S1784	L167	S1828	73	3	2
S3359	L484	S0391	S0393	L485	S1784	S1784	L217	S1828	73	3	2

十 附录

10.1 画地铁位置关系的程序

```

fplot('0',[2 46]); %画直线作为T1铁路线
y=0:0.01:0.3;
for i=2:2:46
    il=i*ones(size(y));
    hold on;
    plot(il,y); %给站点作标记
    istr=int2str(i/2);
    text(i-0.5,-0.5,istr); %给每个站点标上铁路站号
end

x1=24:0.005:36;y1=(36-(30-x1).^2).^0.5;y2=-y1;
plot(x1,y1,x1,y2,'b'); %画圆作为T2铁路线

```

```

for j=1:6
    x2(j)=30-6*cos(pi*j/7);y3(j)=6*sin(pi*j/7);
    plot(x2(j),y3(j),'r*'); %给站点作标记
    jstr=int2str(j+26);
    text(x2(j)-0.5,y3(j)-0.8,jstr); %给每个站点标上铁路站号
end

for j=1:10
    x2(j)=30-6*cos(pi*j/11);y3(j)=-6*sin(pi*j/11);
    plot(x2(j),y3(j),'r*'); %给站点作标记
end

for j=1:3
    jstr=int2str(27-j);
    text(x2(j)-0.5,y3(j)-0.8,jstr); %给每个站点标上铁路站号
end

for j=4:10
    jstr=int2str(43-j);
    text(x2(j)-0.5,y3(j)-0.8,jstr); %给每个站点标上铁路站号
end

axis([-2 50 -10 10]); %修改坐标轴
title('地铁示意图'); %注明标题
hold off;

```

10.2 问题一的相关源程序

10.2.1 (文件名: CalculateStandars.h)

```

#include <vector>
#include <map>
#include <fstream>
#include "LoadMessage.h"
#include <string>
#include <iomanip>
#include "CalulateWays.h"

using namespace std;

// 计算经过各路线的所花费的时间
multimap<float, vector<int> > TimeCalculate(
    vector<vector<int> > & FeasibleSolve,
    vector<LoadMessage> & AllLines
)

```

```

{
    multimap<float, vector<int> > Respond;          // 返回值

    for (int i=0; i<FeasibleSolve.size(); ++i)      // 遍历每一种方案
    {
        float fTotalTime = 0;                      // 总花时

        vector<int> CurrWay;                        // 行车路线

        int nLineName = FeasibleSolve[i][1];
                                                // 第一段路
        fTotalTime = 3.0*(FeasibleSolve[i][2] - FeasibleSolve[i][0]);

        int a = FeasibleSolve[i][2];
        int b = FeasibleSolve[i][0];

        CurrWay.push_back(AllLines[nLineName-1].LineInfo[FeasibleSolve[i][0]]);
        CurrWay.push_back(nLineName);

        if (FeasibleSolve[i].size() == 3)
        {

            CurrWay.push_back(AllLines[nLineName-1].LineInfo[FeasibleSolve[i][2]]);

            Respond.insert( make_pair(fTotalTime, CurrWay) );
            continue;
        }

        int nLineName1 = -1;

        for (int j=3; j<FeasibleSolve[i].size(); j+=3) // 计算剩余每一段路花费的时
间
        {
            float fEachTime = 0;

            fEachTime = 3.0*(FeasibleSolve[i][j+2] - FeasibleSolve[i][j]); // 乘车
用时
            fEachTime += 5.0;                      // 换乘时间
            fTotalTime+= fEachTime;

            nLineName1 = FeasibleSolve[i][j+1];

            CurrWay.push_back(AllLines[nLineName1-1].LineInfo[FeasibleSolve[i][j]]);

```

```

        CurrWay.push_back(nLineName1);
    }

    CurrWay.push_back(AllLines[nLineName1-1].LineInfo[FeasibleSolve[i][FeasibleSolve[i].size()-1]]);

    Respond.insert( make_pair(fTotalTime, CurrWay) );
}

return Respond;
}

// 文件输出花费时间
void OutPutTimeCal(ofstream & fout,
                   multimap<float, vector<int> > & FeasibleSolve)
{
    multimap<float, vector<int> >::iterator iter;
    iter = FeasibleSolve.begin();

    while ( iter != FeasibleSolve.end() )
    {
        fout << "用时: " << setw(4) << setfill(' ') << iter->first << " min : ";

        fout << " S" << setw(4) << setfill('0') << iter->second[0];

        for (int i=1; i<iter->second.size(); i+=2)
        {
            fout << " L" << setw(4) << setfill('0') << iter->second[i];
            fout << " S" << setw(4) << iter->second[i+1];
        }

        fout << endl;

        iter ++;
    }
}

// 计算经过各路线的所花费的费用
multimap<float, vector<int> > CostCalculate(
    vector<vector<int> > & FeasibleSolve,
    vector<LoadMessage> & AllLines
)
{

```



```

multimap<float, vector<int> > Respond;          // 返回值

for (int i=0; i<FeasibleSolve.size(); ++i)      // 遍历每一种方案
{
    int nTotalCost = 0;
    vector<int> CurrWay;                        // 行车路线

    int nFirstLineName = FeasibleSolve[i][1];

    CurrWay.push_back( AllLines[ nFirstLineName-1 ].LineInfo[ FeasibleSolve[i][0] ] );

    for (int j=0; j<FeasibleSolve[i].size(); j+=3) // 每一段路
    {
        int nEachCost = 0;
        int nLineName = -1;

        nLineName = FeasibleSolve[i][j+1];
                                                // 单一票价路段
        if ( AllLines[nLineName-1].strPrice == "单一票制1元。" )
        {
            nEachCost = 1;
        }
        else                                // 分段票价路段
        {
            int nDistance = FeasibleSolve[i][j+2]-FeasibleSolve[i][j];

            if (nDistance <= 20)
                nEachCost = 1;
            else if( nDistance <= 40)
                nEachCost = 2;
            else
                nEachCost = 3;
        }

        nTotalCost += nEachCost;
        CurrWay.push_back( nLineName );

        CurrWay.push_back( AllLines[ nLineName-1 ].LineInfo[ FeasibleSolve[i][j+2] ] );

    }

    Respond.insert( make_pair(nTotalCost, CurrWay) );
}

```

```

        return Respond;
    }

// 文件输出费用
void OutPutCostCal(ofstream & fout,
                  multimap<float, vector<int> > & FeasibleSolve)
{
    multimap<float, vector<int> >::iterator iter;
    iter = FeasibleSolve.begin();

    while ( iter != FeasibleSolve.end() )
    {
        fout << "费用: " << setw(4) << setfill(' ') << iter->first << " 元 : ";

        fout << " S" << setw(4) << setfill('0') << iter->second[0];

        for (int i=1; i<iter->second.size(); i+=2)
        {
            fout << " L" << setw(4) << setfill('0') << iter->second[i];
            fout << " S" << setw(4) << iter->second[i+1];
        }

        fout << endl;

        iter ++;
    }
}

// 综合处理时间，花费，转乘次数。并将结果分别存入文件中。
void Time_Cost_BusChangeTimes_Pro
(
    int nStart, int nEnd, // 起始点和目标
    vector<LoadMessage> & AllLines, // 所有路线
    vector<vector<int> > & AllBuses, // 所有公汽
    vector<vector<vector<int> > > & RoadCrossInfo // 任意两路线的
    交叉信息
)
{
    char char_1[20];

```

```

char    char_2[20];

itoa( nStart, char_1, 10 );
itoa( nEnd  , char_2, 10 );

string * pStr_1 = new string(char_1);
string * pStr_2 = new string(char_2);

string strPartName = "(" + * pStr_1 + "-" + * pStr_2 + ")";

ofstream fout_Cost(      (strPartName+"费用.txt").c_str() );
ofstream fout_Time(      (strPartName+"用时.txt").c_str() );

vector<vector<int> > FeasibleSolve_0;    // 直达
vector<vector<int> > FeasibleSolve_1;    // 一次转乘
vector<vector<int> > FeasibleSolve_2;    // 二次转乘

Direct_or_not( nStart, nEnd, FeasibleSolve_0, AllLines, AllBuses, RoadCrossInfo);
OnceBusChange( nStart, nEnd, FeasibleSolve_1, AllLines, AllBuses, RoadCrossInfo);
TwiceBusChange( nStart, nEnd, FeasibleSolve_2, AllLines, AllBuses, RoadCrossInfo);

multimap<float, vector<int> > Cost_0 = CostCalculate( FeasibleSolve_0, AllLines );
multimap<float, vector<int> > Cost_1 = CostCalculate( FeasibleSolve_1, AllLines );
multimap<float, vector<int> > Cost_2 = CostCalculate( FeasibleSolve_2, AllLines );
multimap<float, vector<int> > Time_0 = TimeCalculate( FeasibleSolve_0, AllLines );
multimap<float, vector<int> > Time_1 = TimeCalculate( FeasibleSolve_1, AllLines );
multimap<float, vector<int> > Time_2 = TimeCalculate( FeasibleSolve_2, AllLines );

fout_Cost << "起始点: " << nStart << endl;
fout_Cost << "目标点: " << nEnd  << endl;
fout_Cost << "#####" << endl;
fout_Cost << "直达情况下: " << endl;
fout_Cost << "总共有: " << FeasibleSolve_0.size() << " 种" << endl;
OutPutCostCal(fout_Cost, Cost_0);
fout_Cost << "#####" << endl;
fout_Cost << "一次换乘下: " << endl;
fout_Cost << "总共有: " << FeasibleSolve_1.size() << " 种" << endl;
OutPutCostCal(fout_Cost, Cost_1);
fout_Cost << "#####" << endl;
fout_Cost << "二次换乘下: " << endl;
fout_Cost << "总共有: " << FeasibleSolve_2.size() << " 种" << endl;
OutPutCostCal(fout_Cost, Cost_2);

```

```

fout_Time << "起始点: " << nStart << endl;
fout_Time << "目标点: " << nEnd << endl;
fout_Time << "#####" << endl;
fout_Time << "直达情况下: " << endl;
fout_Time << "总共有: " << FeasibleSolve_0.size() << " 种" << endl;
OutPutTimeCal(fout_Time, Time_0);
fout_Time << "#####" << endl;
fout_Time << "一次换乘下: " << endl;
fout_Time << "总共有: " << FeasibleSolve_1.size() << " 种" << endl;
OutPutTimeCal(fout_Time, Time_1);
fout_Time << "#####" << endl;
fout_Time << "二次换乘下: " << endl;
fout_Time << "总共有: " << FeasibleSolve_2.size() << " 种" << endl;
OutPutTimeCal(fout_Time, Time_2);

/* ofstream fout_BusChange( (strPartName+"转乘.txt").c_str() );

multimap<float, vector<int>> BusT_0 = CostCalculate( FeasibleSolve_0, AllLines );
multimap<float, vector<int>> BusT_1 = CostCalculate( FeasibleSolve_1, AllLines );
multimap<float, vector<int>> BusT_2 = CostCalculate( FeasibleSolve_2, AllLines );

fout_BusChange << "起始点: " << nStart << endl;
fout_BusChange << "目标点: " << nEnd << endl;
fout_BusChange << "#####" << endl;
fout_BusChange << "直达情况下: " << endl;
fout_BusChange << "总共有: " << FeasibleSolve_0.size() << " 种" << endl;
OutPutCostCal(fout_BusChange, BusT_0);
fout_BusChange << "#####" << endl;
fout_BusChange << "一次换乘下: " << endl;
fout_BusChange << "总共有: " << FeasibleSolve_1.size() << " 种" << endl;
OutPutCostCal(fout_BusChange, BusT_1);
fout_BusChange << "#####" << endl;
fout_BusChange << "二次换乘下: " << endl;
fout_BusChange << "总共有: " << FeasibleSolve_2.size() << " 种" << endl;
OutPutCostCal(fout_BusChange, BusT_2);
*/

/*
ofstream fout ("换乘二次的解决方案.txt");

ofstream fout1("测试费用.txt");

```

```

for (int j=0; j<200; ++j)
{
    vector<vector<int> > FeasibleLine;

    cout << "请输入起始站点和目标站点: " ;
    cin >> nStart >> nEnd;

    fout << "#####" << endl;
    fout << "起始站点: " << nStart << endl;
    fout << "目标站点: " << nEnd << endl;

    //      Direct_or_not(nStart, nEnd, FeasibleSolve, AllLines, AllBuses,
RoadCrossInfo);

    OnceBusChange(nStart, nEnd, FeasibleSolve, AllLines, AllBuses,
RoadCrossInfo);

    //      TwiceBusChange(nStart, nEnd, FeasibleSolve, AllLines, AllBuses,
RoadCrossInfo);

    multimap<float, vector<int> > SolveList = CostCalculate( FeasibleLine,
AllLines );

    OutPutCostCal(fout1, SolveList);

    fout << "连线情况有 " << FeasibleLine.size() << " 种: " << endl;

    // 输出线路
    for (int i=0; i<FeasibleLine.size(); ++i)
    {
        for (int k=0; k<FeasibleLine[i].size(); ++k)
        {
            fout << FeasibleLine[i][k] << " ";
        }
        fout << endl;
    }
}

*/
}

```

10.2.2 (文件名: CalculateWays.h)

```
#include <map>
```

```

#include <iostream>
#include <vector>
#include "LoadMessage.h"
#include <fstream>

using namespace std;

// 判断一条线上两点间是否是上下层次关系，如果是，确定位置
bool find(vector<int> &Line, const int NFIRST, const int NSECOND,
          int &pos_1, int &pos_2)
{
    if (NFIRST == NSECOND)
    {
        int i = 0;

        while (Line[i] != NFIRST)
        {
            ++i;
        }

        pos_1 = pos_2 = i;

        return true;
    }

    bool bHaveFound = false;
    bool IsDirectTo_1 = false;

    for (int i=0; i<Line.size(); ++i)
    {
        if (Line[i] == NFIRST)
        {
            IsDirectTo_1 = true;
            pos_1 = i;
            continue;
        }

        if (IsDirectTo_1 && Line[i] == NSECOND)
        {
            pos_2 = i;
            bHaveFound = true;
            break;
        }
    }
}

```

```

    if ( !bHaveFound )
    {
        pos_1 = pos_2 = -1;
    }

    return bHaveFound;
}

// 找出两点间是否有连线直接连接
// 如果有，则返回所有可能的连线

bool Direct_or_not(const int NSTART, const int NEND,           // 起始点和目标
点
                    vector<vector<int> > &FeasibleSolve,        // 可行的解
决方案，是返回值
                    vector<LoadMessage> & AllLines,            // 所有路线
信息
                    vector<vector<int> > & AllBuses,            // 所有公汽
站信息
                    vector<vector<vector<int> > > &RoadCrossInfo) // 任意两路线的
交叉信息
{
    int i, j ;
    i = j = 0;

    bool bHaveFound = false;

    vector<int> PossibleLine; // 存储可能的解

    // 搜寻各自经过起始站点、目标站点线路间是否为同一线路
    for (i=0; i<AllBuses[NSTART-1].size(); ++i)
    {
        for (j=0; j<AllBuses[NEND-1].size(); ++j)
        {
            if (AllBuses[NSTART-1][i] == AllBuses[NEND-1][j])
            {
                PossibleLine.push_back(AllBuses[NSTART-1][i]);
            }
        }
    }

    bool bIsDirect_1 = false;
    int nCurLineName = -1; // 可行路线的编号

```

```

int  nStart_pos   = -1;      // 起始点在可行线路中的位置
int  nEnd_pos     = -1;      // 目标点在可行线路中的位置
vector<int> FeasibleLine(3, 0);

for (i=0; i<PossibleLine.size(); ++i)
{
    nCurLineName = PossibleLine[i];

    for (j=0; j<AllLines[ nCurLineName-1 ].LineInfo.size(); ++j)
    {
        if (AllLines[nCurLineName-1].LineInfo[j] == NSTART)
        {
            bIsDirect_1 = true;
            nStart_pos   = j;
            continue;
        }

        if (bIsDirect_1 &&
            AllLines[nCurLineName-1].LineInfo[j] == NEND)
        {
            bHaveFound = true;
            nEnd_pos    = j;

            FeasibleLine[0] = nStart_pos;
            FeasibleLine[1] = nCurLineName;
            FeasibleLine[2] = nEnd_pos;

            FeasibleSolve.push_back( FeasibleLine );

            break;
        }
    }

    bIsDirect_1 = false;
}

return bHaveFound;
}

// 一次换乘的情况
bool OnceBusChange(const int NSTART, const int NEND,           // 起始点和目标
点
                    vector<vector<int> > & FeasibleSolve,       // 可行的解决方

```


案，是返回值

信息

站信息

交叉信息

{

```
bool bHaveFound = false;
```

```
vector<int> FeasibleLine(6, 0);
```

```
int i, j, k;
```

```
i = j = k = 0;
```

```
int nFirstLine = -1;
```

```
int nSecondLine = -1;
```

```
int nCurBridge = -1;
```

点

```
bool IsDirectToBridge = false;
```

// 起始点是否能直达中转

点

```
bool IsBridgeDirectToDest = false;
```

// 中转点是否能直达目标

// 搜索相交的两条线路

```
for (i=0; i<AllBuses[NSTART-1].size(); ++i)
```

// 经过起始点的所有线路

```
{
```

```
    nFirstLine = AllBuses[NSTART-1][i];
```

```
    for (j=0; j<AllBuses[NEND-1].size(); ++j)
```

// 经过终点的所有线路

```
    {
```

```
        nSecondLine = AllBuses[NEND-1][j];
```

// 两路线的公有站点

```
        int nLine1 = -1;
```

```
        int nLine2 = -1;
```

```
        if ( nFirstLine < nSecondLine )
```

```
        {
```

```
            nLine1 = nFirstLine;
```

```
            nLine2 = nSecondLine;
```

```
        }
```

```
        else
```

```
        {
```

```
            nLine2 = nFirstLine;
```

```

        nLine1 = nSecondLine;
    }

    for (k=0; k<RoadCrossInfo[nLine1-1][nLine2-1].size(); ++k)
    {
        nCurBridge = RoadCrossInfo[nLine1-1][nLine2-1][k];

        int a = 0, b = 0, c = 0, d =0;
        bool IsDirectToBridge
            = find(AllLines[nFirstLine-1].LineInfo, NSTART, nCurBridge, a,
b);

        bool IsBridgeDirectToDest = false;

        if ( IsDirectToBridge )
        {
            IsBridgeDirectToDest
                = find(AllLines[nSecondLine-1].LineInfo, nCurBridge, NEND, c,
d);
        }

        if ( IsDirectToBridge && IsBridgeDirectToDest)
        {
            FeasibleLine[0] = a; FeasibleLine[1] = nFirstLine;
            FeasibleLine[2] = b; FeasibleLine[3] = c;
            FeasibleLine[4] = nSecondLine; FeasibleLine[5] = d;

            FeasibleSolve.push_back(FeasibleLine);
        }

        IsDirectToBridge = false;
        IsBridgeDirectToDest = false;
    }
}

// cout << FeasibleSolve.size() << endl;

if ( !FeasibleSolve.empty() )
    bHaveFound = true;

return bHaveFound;
}

```

```

// 二次换乘的情况
bool TwiceBusChange(const int NSTART, const int NEND,           // 起始点和目标
点
                        vector<vector<int> > & FeasibleSolve,      // 可行的解决方
案，是返回值
                        vector<LoadMessage> & AllLines,           // 所有路线
信息
                        vector<vector<int> > & AllBuses,           // 所有公汽
站信息
                        vector<vector<vector<int> > > & RoadCrossInfo) // 任意两路线的
交叉信息
{
    bool    bHaveFound = false;

    vector<int> FeasibleLine(9, 0);
    int        nCurLineName = -1;
    int        nFirstBusChangeStation = -1;           // 第一次换乘的车站名

    int i, j, k;
    i = j = k = 0;

    for (i=0; i<AllBuses[NSTART-1].size(); ++i)       // 经过起始站点的所有线
路
    {
        nCurLineName = AllBuses[NSTART-1][i];

        j=0;
        int nStart_pos = -1;

        while (AllLines[nCurLineName-1].LineInfo[j] != NSTART)
            ++j;

        nStart_pos = j;

                                                // 当前线路上的所有站点
        for (j=j+1; j<AllLines[nCurLineName-1].LineInfo.size(); ++j)
        {
            nFirstBusChangeStation = AllLines[nCurLineName-1].LineInfo[j];

            vector<vector<int> > temp;

            bool IsConnected
                = OnceBusChange(nFirstBusChangeStation,
                                NEND, temp, AllLines, AllBuses, RoadCrossInfo );

```

```

        if ( IsConnected )
        {
            int m = temp[0][0];
            int n = temp[0][1];
            int d = AllLines[n-1].LineInfo[m];

            for (k=0; k<temp.size(); ++k)
            {
                FeasibleLine[0] = nStart_pos; FeasibleLine[1] = nCurLineName;
                FeasibleLine[2] = j           ; FeasibleLine[3] = temp[k][0];
                FeasibleLine[4] = temp[k][1]; FeasibleLine[5] = temp[k][2];
                FeasibleLine[6] = temp[k][3]; FeasibleLine[7] = temp[k][4];
                FeasibleLine[8] = temp[k][5];

                FeasibleSolve.push_back( FeasibleLine );
            }
        }
    }
}

cout << FeasibleSolve.size() << endl;

if ( !FeasibleSolve.empty() )
    bHaveFound = true;

return bHaveFound;
}

```

10.2.3 (文件名: LoadMessage.h)

```

#ifndef    ROADMESSAGE_H
#define    ROADMESSAGE_H

#include <string>
#include <vector>
using namespace std;

// 线路信息
class LoadMessage
{
public:
    LoadMessage()
    {
        nLineName    = -1;
    }
}

```

```

        strPrice = "";
        bRoundLoad = false;
    }

    int    nLineName;        // 公交线路名
    string strPrice ;        // 计费信息
    vector<int> LineInfo;    // 线路信息, 里面存储的是当前公交线路
                                // 沿线公汽站点。

    bool    bRoundLoad;      // 判别是否为环形路。
};

#endif

```

10.2.4(文件名: ShortPathCalculate.cpp)

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 此程序计算最小公交线路
//
// 有关要求: 用户输入起始站点和终点, 在满足换乘次数不超过某一限度的情况下
//            程序输出所有可能的行车路线
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#pragma warning(disable: 4786)

#include <map>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <queue>
#include <vector>
#include "Utility.h"
#include "CalculateStandars.h"
// #include "CalulateWays.h"

using namespace std;

void main()
{
    ifstream fin("1.1 公汽线路信息.txt");

    // 经过实现的计算, 总共有 520 条公汽线路, 3957 个公汽站点

```

```

//
// 将同一路线的上、下行路线看成两个不同的路线
// 将环形路线换算成两个与之等价的单行路线
vector<LoadMessage> AllLines(1040);
vector<vector<int> > AllBuses(3957);

string strCurrLine = " ";

string strBusLine = ""; // 公交线路
string strPrice = ""; // 票价信息
string strUpLoad = ""; // 上行路线
string strDownLoad = ""; // 下行路线
string strRoundLoad = ""; // 环形路线
bool IsRoundWay = false; // 判断当前读入的路线是否为环形路线

for(int nCount=0; nCount<520; ++nCount)
{
    IsRoundWay = false;

    // 读入路线名
    do {
        getline(fin, strCurrLine, '\n');
    } while(strCurrLine == "");

    strBusLine = strCurrLine;

    // 读入计费标准
    do {
        getline(fin, strCurrLine, '\n');
    } while(strCurrLine == "");

    strPrice = strCurrLine;

    // 读入上行路线
    do {
        getline(fin, strCurrLine, '\n');
    } while(strCurrLine == "");

    string temp(strCurrLine, 0, 6);

    // 环形线路
    if ( temp == "环行: " )
    {
        strUpLoad = "";
    }
}

```

```

        strDownLoad = "";
        strRoundLoad = string(strCurrLine, 6, strCurrLine.length());
        IsRoundWay = true;
    }
    // 下行线路不是上行线路的逆方向
    else if( temp == "上行: " )
    {
        strUpLoad = string(strCurrLine, 6, strCurrLine.length());
        strRoundLoad = "";

        // 读入下行路线
        do {
            getline(fin, strCurrLine, '\n');
        } while(strCurrLine == "");

        strDownLoad = string(strCurrLine, 6, strCurrLine.length());
    }
    // 下行线路是上行线路的逆方向
    else
    {
        strUpLoad = strCurrLine;
        strDownLoad = "";
        strRoundLoad = "";
    }

    // 构造同一路线分成的两个路线，并把它们存入容器中
    LoadMessage currLoad_up; // 当前路线的上行路线
    LoadMessage currLoad_down; // 当前路线的下行路线

    currLoad_up.nLineName = StrToInt( string(strBusLine, 1, 3) );
    currLoad_up.strPrice = strPrice;
    currLoad_up.bRoundLoad = IsRoundWay;

    currLoad_down.nLineName = currLoad_up.nLineName + 520;
    currLoad_down.strPrice = strPrice;
    currLoad_down.bRoundLoad = IsRoundWay;

    // 环形路
    if ( IsRoundWay )
    {
        RoundRoadCal(strRoundLoad, currLoad_up.nLineName, AllBuses,
            currLoad_up.LineInfo, currLoad_down.LineInfo);
    }

```

```

        // 非环形路的上行路
        if (strUpLoad != "")
        {
            BusRoadCal(strUpLoad, currLoad_up.nLineName, AllBuses,
currLoad_up.LineInfo);
        }

        // 非环形路的下行路
        if (strDownLoad != "")
        {
            BusRoadCal(strDownLoad, currLoad_down.nLineName, AllBuses,
currLoad_down.LineInfo);
        }
        else if( !IsRoundWay )
        {
            ReverseRoadCal(strUpLoad, currLoad_down.nLineName, AllBuses,
currLoad_down.LineInfo);
        }

        AllLines[currLoad_up.nLineName-1] = currLoad_up;
        AllLines[currLoad_down.nLineName-1] = currLoad_down;
    }

    // 520*520 的矩阵，里面存储相对应的线路之间的重合站点信息
    vector<vector<int> > vTemp(520);
    vector<vector<vector<int> > > RoadCrossInfo(520, vTemp);

    // 任意两条线路间相交间相交顶点信息
    RoadCrossInfo = CalRoadCrossInfo( AllBuses );

    int nStart = -1;
    int nEnd = -1;

    for (int i=0; i<200; ++i)
    {
        cout << "请输入起始站点: " ;
        cin >> nStart >> nEnd;

        Time_Cost_BusChangeTimes_Pro(nStart, nEnd, AllLines, AllBuses,
RoadCrossInfo);
    }
}

```



```

/* ofstream fout("各公汽站信息.txt");

for (int i=0; i<AllBuses.size(); ++i)
{
    fout << i+1 << ": ";

    for (int j=0; j<AllBuses[i].size(); ++j)
    {
        fout << AllBuses[i][j] << " ";
    }

    fout << endl;
}

ofstream fout1("各公路线路重合站点信息.txt");

for (int i=0; i<520; ++i)
{
    for (int j=i; j<520; ++j)
    {
        for (int k=0; k<RoadCrossInfo[i][j].size(); ++k)
        {
            fout1 << RoadCrossInfo[i][j][k] << " ";
        }

        fout1 << endl;
    }
}
*/

```

10.3 问题二的相关程序

10.3.1 (文件名: CalculateStandars.h)

```

#include <vector>
#include <map>
#include <fstream>
#include "LoadMessage.h"
#include <string>
#include <iomanip>
#include "CalulateWays.h"

```

```

using namespace std;

float BusPrice(int nLineName, int nDistance,
               vector<vector<int> > & FeasibleSolve,
               vector<LoadMessage> & AllLines)
{
    float nEachCost = 0;

    if ( AllLines[nLineName-1].strPrice == "单一票制 1 元。" )
    {
        nEachCost = 1;
    }
    else // 分段票价路段
    {
        if (nDistance <= 20)
            nEachCost = 1;
        else if( nDistance <= 40)
            nEachCost = 2;
        else
            nEachCost = 3;
    }

    return nEachCost;
}

// 计算经过各路线的所花费的时间
vector<vector<float> > Time_Cost_Calculate(
                                   vector<vector<int> > & FeasibleSolve,
                                   vector<LoadMessage> & AllLines
                                   )
{
    vector<vector<float> > Respond; // 返回值

    for (int i=0; i<FeasibleSolve.size(); ++i) // 遍历每一种方案
    {
        float fTotalTime = 0; // 总花时
        float fTotalCost = 0; // 总花费

        float fCost_A = 0;
        float fCost_B = 0;
        float fCost_C = 0;

        vector<float> CurrWay; // 行车路线
    }
}

```

```

int A_1 = FeasibleSolve[i][0];
int A   = FeasibleSolve[i][1];
int A_2 = FeasibleSolve[i][2];

float Time_A = 3.0*(A_2 - A_1);

int B_1 = FeasibleSolve[i][3];
int B   = FeasibleSolve[i][4];
int B_2 = FeasibleSolve[i][5];

float Time_B = 0.0;

int C_1 = FeasibleSolve[i][6];
int C   = FeasibleSolve[i][7];
int C_2 = FeasibleSolve[i][8];

float Time_C = 3.0*(C_2 - C_1);

float fTime_B_C = 0;
float fTime_A_B = 0;

if (B > 1040)
{
    fTime_A_B = 6.0;
    fTime_B_C = 7.0;
    Time_B     = 2.5*(B_2-B_1);

    fCost_B     = 3.0;
}
else
{
    fTime_A_B = 5.0;
    fTime_B_C = 5.0;
    Time_B     = 3.0*(B_2-B_1);

    fCost_B = BusPrice(B, B_2-B_1, FeasibleSolve, AllLines);
}

if (A_1 != A_2)
    fCost_A = BusPrice(A, A_2-A_1, FeasibleSolve, AllLines);
else
    fCost_A = 0;

```

```

        if (C_1 != C_2)
            fCost_C = BusPrice(C, C_2-C_1, FeasibleSolve, AllLines);
        else
            fCost_C = 0;

        fTotalTime = Time_A + Time_B + Time_C + fTime_A_B + fTime_B_C;
        fTotalCost = fCost_A + fCost_B + fCost_C;

        int nA_1 = AllLines[A-1].LineInfo[A_1];
        int nA_2 = AllLines[A-1].LineInfo[A_2];

        int nB_1 = AllLines[B-1].LineInfo[B_1];
        int nB_2 = AllLines[B-1].LineInfo[B_2];

        int nC_1 = AllLines[C-1].LineInfo[C_1];
        int nC_2 = AllLines[C-1].LineInfo[C_2];

        CurrWay.push_back(nA_1); CurrWay.push_back(A); CurrWay.push_back(nA_2);
        CurrWay.push_back(nB_1); CurrWay.push_back(B); CurrWay.push_back(nB_2);
        CurrWay.push_back(nC_1); CurrWay.push_back(C); CurrWay.push_back(nC_2);
        CurrWay.push_back(fTotalTime);
        CurrWay.push_back(fTotalCost);

        Respond.push_back( CurrWay );
    }

    return Respond;
}

// 文件输出花费时间
void OutPutTime_Cost_Cal(ofstream & fout,
                        vector<vector<float> > & FeasibleSolve)
{
    for (int i=0; i<FeasibleSolve.size(); ++i)
    {
        for (int j=0; j<FeasibleSolve[i].size(); ++j)
        {
            fout << setw(3) << left << setfill(' ') << FeasibleSolve[i][j] << "    ";
        }

        if (FeasibleSolve[i][3] == 3984 &&
            FeasibleSolve[i][5] == 3993 &&
            FeasibleSolve[i][2] == 87    &&
            FeasibleSolve[i][6] == 3676)

```

```

        {
            int a = i+1;
            cout << a << endl;
        }

        fout << endl;
    }
}

// 综合处理时间，花费，转乘次数。并将结果分别存入文件中。
void Time_Cost_BusChangeTimes_Pro
(
    int nStart, int nEnd, // 起始点和目标
    点
    vector<LoadMessage> & AllLines, // 所有路线
    信息
    vector<vector<int> > & AllBuses, // 所有公汽
    站信息
    vector<vector<vector<long> > > &RoadCrossInfo // 任意两路线的
    交叉信息
)
{
    char char_1[20];
    char char_2[20];

    itoa( nStart, char_1, 10 );
    itoa( nEnd , char_2, 10 );

    string * pStr_1 = new string(char_1);
    string * pStr_2 = new string(char_2);

    string strPartName = "(" + * pStr_1 + "-" + * pStr_2 + ")";

    ofstream fout( (strPartName+"用时_费用.txt").c_str() );

    vector<vector<int> > FeasibleSolve_2; // 二次转乘

    TwiceBusChange(nStart, nEnd, FeasibleSolve_2, AllLines, AllBuses, RoadCrossInfo);

    vector<vector<float> > A
        = Time_Cost_Calculate(FeasibleSolve_2, AllLines);

    OutPutTime_Cost_Cal(fout, A);
}

```

```
}
```

10.3.2(文件名: CalulateWays.h)

```
#include <map>
#include <iostream>
#include <vector>
#include "LoadMessage.h"
#include <fstream>
#include <queue>

using namespace std;

int aaaaaa ;
int bbbbbb ;

// 判断一条线上两点间是否是上下层次关系, 如果是, 确定位置
bool find(vector<int> &Line, const int NFIRST, const int NSECOND,
          int &pos_1, int &pos_2)
{
    if (NFIRST == NSECOND)
    {
        int i = 0;

        while (Line[i] != NFIRST)
        {
            ++i;
        }

        pos_1 = pos_2 = i;

        return true;
    }

    bool bHaveFound = false;
    bool IsDirectTo_1 = false;

    for (int i=0; i<Line.size(); ++i)
    {
        if (Line[i] == NFIRST)
        {
            IsDirectTo_1 = true;
            pos_1 = i;
            continue;
        }
    }
}
```

```

    }

    if (IsDirectTo_1 && Line[i] == NSECOND)
    {
        pos_2 = i;
        bHaveFound = true;
        break;
    }
}

if ( !bHaveFound )
{
    pos_1 = pos_2 = -1;
}

return bHaveFound;
}

// 找出两点间是否有连线直接连接
// 如果有，则返回所有可能的连线

bool Direct_or_not(const int NSTART, const int NEND,           // 起始点和目标
点
                    vector<vector<int>> &FeasibleSolve,          // 可行的解
决策方案，是返回值
                    vector<LoadMessage> & AllLines,             // 所有路线
信息
                    vector<vector<int>> & AllBuses,              // 所有公汽
站信息
                    vector<vector<vector<long>>> &RoadCrossInfo) // 任意两路线的
交叉信息
{
    int i, j ;
    i = j = 0;

    bool bHaveFound = false;

    return bHaveFound;
}

// 一次换乘的情况
bool OnceBusChange(const int NSTART, const int NEND,           // 起始点和目标
点

```

```

        vector<vector<int>> & FeasibleSolve,           // 可行的解决方
案，是返回值
        vector<LoadMessage> & AllLines,             // 所有路线
信息
        vector<vector<int>> & AllBuses,               // 所有公汽
站信息
        vector<vector<vector<long>>> &RoadCrossInfo) // 任意两路线的
交叉信息
{
    bool    bHaveFound = false;

    vector<int> FeasibleLine(6, 0);

    int i, j, k;
    i = j = k = 0;

    int nFirstLine = -1;
    int nSecondLine = -1;
    long nCurrValue = -1;
    long nCurBridge1 = -1; // 上行方向的桥接点
    long nCurBridge2 = -1; // 下行方向的桥接点

    bool IsDirectToBridge = false;           // 起始点是否能直达中转
点
    bool IsBridgeDirectToDest = false;       // 中转点是否能直达目标
点

    // 搜索相交的两条线路
    for (i=0; i<AllBuses[NSTART-1].size(); ++i) // 经过起始点的所有线路
    {
        nFirstLine = AllBuses[NSTART-1][i];

        for (j=0; j<AllBuses[NEND-1].size(); ++j) // 经过终点的所有线路
        {
            nSecondLine = AllBuses[NEND-1][j];

            // 两路线的公有站点

            int nLine1 = -1;
            int nLine2 = -1;

            if ( nFirstLine < nSecondLine )
            {
                nLine1 = nFirstLine;
                nLine2 = nSecondLine;
            }
        }
    }
}

```



```

else
{
    nLine2 = nFirstLine;
    nLine1 = nSecondLine;
}

for (k=0; k<RoadCrossInfo[nLine1-1][nLine2-1].size(); ++k)
{
    // 桥接点
    nCurrValue = RoadCrossInfo[nLine1-1][nLine2-1][k];

    if (nCurrValue >= 10000)
    {
        if ( nFirstLine == nLine1 )
        {
            nCurBridge2 = nCurrValue % 10000;
            nCurBridge1 = (nCurrValue-nCurBridge2) / 10000;
        }
        else
        {
            nCurBridge1 = nCurrValue % 10000;
            nCurBridge2 = (nCurrValue-nCurBridge1) / 10000;
        }
    }
    else
    {
        nCurBridge1 = nCurBridge2 = nCurrValue;
    }

    int  a = 0, b = 0, c = 0, d =0;
    bool IsDirectToBridge
        = find(AllLines[nFirstLine-1].LineInfo, NSTART, nCurBridge1, a,
b);

    bool IsBridgeDirectToDest = false;

    if ( IsDirectToBridge )
    {
        IsBridgeDirectToDest
            = find(AllLines[nSecondLine-1].LineInfo, nCurBridge2, NEND, c,
d);
    }

    if ( IsDirectToBridge && IsBridgeDirectToDest)

```

```

        {
            FeasibleLine[0] = a; FeasibleLine[1] = nFirstLine;
            FeasibleLine[2] = b; FeasibleLine[3] = c;
            FeasibleLine[4] = nSecondLine; FeasibleLine[5] = d;

            FeasibleSolve.push_back(FeasibleLine);
        }

        IsDirectToBridge      = false;
        IsBridgeDirectToDest = false;
    }
}

if ( !FeasibleSolve.empty() )
    bHaveFound = true;

return bHaveFound;
}

////////////////////////////////////
//
// 二次换乘的情况
//
// 由于第一问解是第二题解的一部分，所以在这里只需求出通过地铁站换乘的情况。
//
bool TwiceBusChange(const int NSTART, const int NEND,           // 起始点和目标
点
                    vector<vector<int> > & FeasibleSolve,       // 可行的解决方
案，是返回值
                    vector<LoadMessage> & AllLines,           // 所有路线
信息
                    vector<vector<int> > & AllBuses,           // 所有公汽
站信息
                    vector<vector<vector<long> > > & RoadCrossInfo) // 任意两路线的
交叉信息
{
    bool      bHaveFound = false;

    vector<int> FeasibleLine(9, 0);
    int         nCurLineName = -1;
    int         nFirstBusChangeStation = -1;           // 第一次换乘的车站名

    int i, j, k;

```

```

i = j = k = 0;

for (i=0; i<AllBuses[NSTART-1].size(); ++i)           // 经过起始站点的所有线
路
{
    nCurLineName = AllBuses[NSTART-1][i];           // 当前路线名

    for (j=1040; j<1044; ++j)
    {
        map<int, int> ToRailWay;
        bool IsExist = false;

        for (k=0; k<RoadCrossInfo[nCurLineName-1][j].size(); ++k)
        {
            if ( RoadCrossInfo[nCurLineName-1][j][k] >= 10000)
            {
                long nValue  = RoadCrossInfo[nCurLineName-1][j][k];
                long nSecond = nValue % 10000;
                long nFirst  = (nValue-nSecond) / 10000;

                ToRailWay.insert( make_pair(nFirst, nSecond) );

                if (nFirst==87 && nSecond==3984)
                {
                    int y=0;
                }

                IsExist = true;
            }
        }

        map<int, int>::iterator iter;

//
// //////////////////////////////////////
//      int nSize_ = ToRailWay.size();
//      iter = ToRailWay.begin();
//      cout << "#####" << endl;
//      while (iter != ToRailWay.end())
//      {
//          cout << iter->first << " " << iter->second << endl;
//          iter ++;
//      }

```

```

if ( !IsExist )
    continue;

int m=0;
int nStart_pos = -1;

iter = ToRailWay.begin();

for (m=0; m<AllLines[nCurLineName-1].LineInfo.size() ; ++m)
{
    if (AllLines[nCurLineName-1].LineInfo[m] == NSTART)
    {
        nStart_pos = m;
        break;
    }

    iter = ToRailWay.find(AllLines[nCurLineName-1].LineInfo[m]);

    if ( iter != ToRailWay.end() )
    {
        ToRailWay.erase( iter );
    }
}

while ( !ToRailWay.empty() )
{
    int a = ToRailWay.size();

    iter = ToRailWay.find( AllLines[nCurLineName-1].LineInfo[m] );

    int _a = iter->first ;
    int _b = iter->second;

//      if (iter->second == 3984 && iter->first == 87)
//      {
//          int y = 0;
//      }

    if ( iter == ToRailWay.end() )
    {
        ++m;
        continue;
    }
}

```

```

vector<vector<int> > temp;
OnceBusChange(iter->second,
    NEND, temp, AllLines, AllBuses, RoadCrossInfo );

int c = temp.size();

for (int kk=0; kk<temp.size(); ++kk)
{
    FeasibleLine[0] = nStart_pos ; FeasibleLine[1] = nCurLineName;
    FeasibleLine[2] = m ; FeasibleLine[3] = temp[kk][0];
    FeasibleLine[4] = temp[kk][1]; FeasibleLine[5] = temp[kk][2];
    FeasibleLine[6] = temp[kk][3]; FeasibleLine[7] = temp[kk][4];
    FeasibleLine[8] = temp[kk][5];

    // int s = temp[kk][1];
    // int s1 = temp[kk][0];
    // int s2 = temp[kk][1];
    // int s3 = temp[kk][2];
    // int s4 = temp[kk][3];
    // int s5 = temp[kk][4];
    //
    // if (AllLines[FeasibleLine[4]-1].LineInfo[FeasibleLine[3]] ==
3984 &&
    // AllLines[FeasibleLine[4]-1].LineInfo[FeasibleLine[5]] ==
3993 )
    // {
    //     int a = 0;
    // }

    FeasibleSolve.push_back( FeasibleLine );
}

ToRailWay.erase( iter );
}
}

if ( !FeasibleSolve.empty() )
    bHaveFound = true;

return bHaveFound;
}

```

10.3.3(文件名: DataPreProcess.h)

```
#include <vector>
#include <iostream>
#include <string>
#include "LoadMessage.h"
#include "Utility.h"

using namespace std;

// 将公汽线路信息及公汽站点信息存入数组中。
void AddBusStationsAndLines(vector<LoadMessage> & AllLines,
                           vector<vector<int> > & AllBuses)
{
    ifstream fin("1.1 公汽线路信息.txt");

    string strCurrLine = " ";

    string strBusLine = ""; // 公交线路
    string strPrice = ""; // 票价信息
    string strUpLoad = ""; // 上行路线
    string strDownLoad = ""; // 下行路线
    string strRoundLoad = ""; // 环形路线
    bool IsRoundWay = false; // 判断当前读入的路线是否为环形路线

    for(int nCount=0; nCount<520; ++nCount)
    {
        IsRoundWay = false;

        // int c = AllLines[1403].LineInfo.size();
        //
        // if (c == 2)
        // {
        //     c = c;
        // }

        // 读入路线名
        do {
            getline(fin, strCurrLine, '\n');
        } while(strCurrLine == "");

        strBusLine = strCurrLine;
```

```

// 读入计费标准
do {
    getline(fin, strCurrLine, '\n');
} while(strCurrLine == "");

strPrice = strCurrLine;

// 读入上行路线
do {
    getline(fin, strCurrLine, '\n');
} while(strCurrLine == "");

string temp(strCurrLine, 0, 6);

// 环形线路
if ( temp == "环行: " )
{
    strUpLoad    = "";
    strDownLoad  = "";
    strRoundLoad = string(strCurrLine, 6, strCurrLine.length());
    IsRoundWay   = true;
}
// 下行线路不是上行线路的逆方向
else if( temp == "上行: " )
{
    strUpLoad    = string(strCurrLine, 6, strCurrLine.length());
    strRoundLoad = "";

    // 读入下行路线
    do {
        getline(fin, strCurrLine, '\n');
    } while(strCurrLine == "");

    strDownLoad = string(strCurrLine, 6, strCurrLine.length());
}
// 下行线路是上行线路的逆方向
else
{
    strUpLoad    = strCurrLine;
    strDownLoad  = "";
    strRoundLoad = "";
}

```

```

// 构造同一路线分成的两个路线，并把它们存入容器中
LoadMessage          currLoad_up;          // 当前路线的上行路线
LoadMessage          currLoad_down;        // 当前路线的下行路线

currLoad_up.nLineName      = StrToInt( string(strBusLine, 1, 3) );
currLoad_up.strPrice       = strPrice;
currLoad_up.bRoundLoad     = IsRoundWay;

currLoad_down.nLineName    = currLoad_up.nLineName + 520;
currLoad_down.strPrice     = strPrice;
currLoad_down.bRoundLoad  = IsRoundWay;

// 环形路
if ( IsRoundWay )
{
    RoundRoadCal(strRoundLoad, currLoad_up.nLineName, AllBuses,
        currLoad_up.LineInfo, currLoad_down.LineInfo);
}

// 非环形路的上行路
if (strUpLoad != "")
{
    BusRoadCal(strUpLoad,          currLoad_up.nLineName,          AllBuses,
currLoad_up.LineInfo);
}

// 非环形路的下行路
if (strDownLoad != "")
{
    BusRoadCal(strDownLoad,        currLoad_down.nLineName,        AllBuses,
currLoad_down.LineInfo);
}
else if( !IsRoundWay )
{
    ReverseRoadCal(strUpLoad,      currLoad_down.nLineName,      AllBuses,
currLoad_down.LineInfo);
}

AllLines[currLoad_up.nLineName-1] = currLoad_up;
AllLines[currLoad_down.nLineName-1] = currLoad_down;
}
}

// 将地铁线路信息及地铁站点信息存入数组中。

```



```

void AddRailWayStationsAndLines(vector<LoadMessage> & AllLines,
                                vector<vector<int> > & AllBuses)
{
    // T1 线路
    for (int i=1; i<24; ++i)
    {
        AllLines[1040].LineInfo.push_back(3957 + i);
        AllLines[1041].LineInfo.push_back(3957 + 24 - i);

        AllBuses[3956+i].push_back(1041);
        AllBuses[3956+i].push_back(1042);
    }

    ifstream fin("地铁线路 T2 信息.txt");
    int nValue = -1;

    for (i=3956+24; i<3996; ++i)
    {
        AllBuses[i].push_back(1043);
        AllBuses[i].push_back(1044);
    }

    AllBuses[3956+12].push_back(1043);
    AllBuses[3956+12].push_back(1044);
    AllBuses[3956+18].push_back(1043);
    AllBuses[3956+18].push_back(1044);

    vector<int> temp_1;
    vector<int> temp_2;

    // T2 线路
    for (i=0; i<18; ++i)
    {
        fin >> nValue;
        temp_1.push_back( 3957+nValue );
    }

    int nLSize = temp_1.size();

    AllLines[1042].LineInfo.resize( nLSize*2, -1 );
    AllLines[1043].LineInfo.resize( nLSize*2, -1 );

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

```

```

        AllLines[1042].LineInfo[i] = temp_1[i];
        AllLines[1042].LineInfo[nLSize+i] = temp_1[i];

        AllLines[1043].LineInfo[nLSize-1-i] = temp_1[i];
        AllLines[1043].LineInfo[nLSize+nLSize-1-i] = temp_1[i];
    }
}

```

10.3.3(文件名: ShortPathCalculate.cpp)

```

////////////////////////////////////
//
//  此程序计算最小公交线路
//
//  有关要求: 用户输入起始站点和终点, 在满足换乘次数不超过某一限度的情况下
//             程序输出所有可能的行车路线
//
////////////////////////////////////

#pragma warning(disable: 4786)

#include <map>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <queue>
#include <vector>
//#include "Utility.h"
#include "DataPreProcess.h"
#include "CalculateStandars.h"
//#include "CalulateWays.h"

using namespace std;

void main()
{
    // 经过事先的计算, 总共有 520 条公汽线路, 3957 个公汽站点
    // 2 条地铁线路, 39 个地铁站点
    //
    // 将同一路线的上、下行路线看成两个不同的路线
    // 将环形路线换算成两个与之等价的单行路线
    // 将地铁线等同于公汽车站
    vector<LoadMessage>      AllLines(1040+4);
}

```

```

vector<vector<int> > AllBuses(3957+39);

// 从文件读入公交线路和站点信息，并存入数组。
AddBusStationsAndLines(AllLines, AllBuses);

// 将地铁站点信息和地铁线路信息存入数组中。
AddRailWayStationsAndLines(AllLines, AllBuses);

// 1044*1044 的矩阵，里面存储相对应的线路之间的重合站点信息
vector<vector<vector<long> > > RoadCrossInfo;

// 任意两条线路间相交间相交顶点信息
RoadCrossInfo = CalRoadCrossInfo( AllBuses );

int nStart = -1;
int nEnd   = -1;

for (int i=0; i<200; ++i)
{
    cout << "请输入起始站点: " ;
    cin >> nStart >> nEnd;

    Time_Cost_BusChangeTimes_Pro(nStart,      nEnd,      AllLines,      AllBuses,
RoadCrossInfo);
}
}

```