**乘公交，看奥运**

**【摘要】该题目是2007年b组题，以图论为主实现交通系统的多条件路径规划，对于若干个联合条件（乘车时间、价格、步行+乘车时间），根据不同的优先度选出最优的路线，是一道非常贴合现实的题目，并且今天已经在百度、高德等地图上有类似运用。**

**对于这题，笔者一种通过重映射的方式，用多属性表达1个点，构建一个简洁高效模型，并且不进行任何近似解的求取，实现低时间复杂度，同时求出所有问题准确解，并且每次运行仅使用一次预处理后，能无限次数多询问（通过超级源汇点和低时间复杂度的单源，制造出”多源”的假象），并且选择了最优的那一条进行了路径还原。**

**在阅读一些分优秀论文后，没有发现想法相似的建模，因此选此为题并提供一个新模型。由于其他论文使用近似解没有求出最短需求，因此最终测试碾压了所有所读论文的最终答案。**

**问题一**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点  ->终点 | S3359  ->S1828 | S1557  ->S0481 | S0971  ->S0485 | S0008  ->S0073 | S0148  ->S0485 | S0087  ->S3676 |
| 耗时最优(min) | 64 | 99 | 103 | 59 | 102 | 46 |
| 费用最优(元) | 3 | 3 | 3 | 2 | 3 | 2 |

**问题二**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点  ->终点 | S3359  ->S1828 | S1557  ->S0481 | S0971  ->S0485 | S0008  ->S0073 | S0148  ->S0485 | S0087  ->S3676 |
| 耗时最优(min) | 62 | 99 | 95 | 53.5 | 86.5 | 33 |
| 费用最优(元) | 3 | 3 | 3 | 2 | 3 | 1 |

**问题三**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点  ->终点 | S3359  ->S1828 | S1557  ->S0481 | S0971  ->S0485 | S0008  ->S0073 | S0148  ->S0485 | S0087  ->S3676 |
| 耗时最优(min)  (带步行时间) | 68 | 105 | 104 | 63 | 95.5 | 41 |
| 费用最优(元) | 3 | 3 | 3 | 2 | 3 | 1 |

**此外，该模型不仅可以提供于交通系统，还可以延伸至其他方面，有较好的适用性。**

**关键词：**Dijkstra、重映射、虚拟点、图重建、多源汇

**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 符号使用及说明**

Sp ：第p个公交站点。

Ly ：第y辆公交车

Tx ：第x条地铁

Dy ：第y个地铁站点

Ai ：Ai=(Sp,Ly)，A表示一个汽车平面（你待在汽车上），Ai是上面一个点，表示乘坐第y辆公交车，此时开到第p个站点（补充：在代码中，由于AB实际当做同类点进行图转移，因此，会使用Ai=(Sp,(-1,Ly))来进行统一）

Bi ：Bi=(Sp,(Tx,Dy))，B表示一个地铁平面（待在地铁上），Bi是上面一个点，表示当乘坐第x条地铁在第y个地铁站点时，此时与第p站点相连（是这个地铁站点的出口）。

Timi,j ：边权其中一个，表示i,j两点之间的乘车时间代价（行驶+换s乘）

Costi,j ：边权其中一个，表示i,j两点的价格代价

Walki,j ：边权其中一个，表示i,j两点的**总时间(带步行)**，即**步行时间代价+Tim乘车时间**代价

Cnti,j ：边权其中一个，表示i,j两点的转乘次数

上面这些符号仅用于解释原理，由于代码需要大量边建立和点重映射，因此运用上非常多的数据结构，具体请在代码中查看注释，这里就不再放上来了。

**三 模型假设**

**3.1原题中提供的假设**

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

分段计价： 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）

**3.2 其它假设**

9、环形汽车和环形地铁没有说明单向还是双向。双向只是在单向基础上额外建一组对称边而已。由于其他数据都是双向，现实中也是双向，这里也统一当做双向。

10、我们的出发点和目标点都是在公交车站站点S。因此，即使一开始乘坐地铁，我们也假设要花费步行时间往下到地下通道做地铁（第二题由于没有步行时间，视为0）。同理，假设坐到的那个地铁站点，已经和站点S相连，不用再乘坐车，但最终还是要消耗步行时间到公交站点S（即，花时间上地面，同样仅第三题存在步行时间代价）。（当然，这里只是因为多源汇不包含这类地铁站点。如果希望地铁相连目标站点S时就算到达，把这一类点也算入多源汇即可，后面再提）。

11、已知存在数据，使得一个地铁站点D连接多个站点S。那么假如D连接着站点S1和S2（即对应现实中，一个地铁站有2个出入口）。那么，如果我们从S1到S2，应该从地面下到地铁通道，再上回去等车。也就是说，这里相当于是从公汽换乘到地铁再换乘到公汽，因此我们应该算两部分换乘时间7+6（步行4+4）。同时，这里所写的程序支持这种情况两个站口之间是否存在过闸费，由于没提，视为不过闸0元，其他地铁情况统一3元

12、当我们考虑步行时间时，我们可以独立考虑步行时间最优，或者视作求总时间=乘车时间+步行时间最优。两种均可实现，经过考虑认为后者更加贴近生活和题目，因此假设为第三题的询问是：联合代价、总时间两个元素，分别从代价最小和总时间最小来求取最优路线。

13、无额外情况（拥堵、不停靠等）

**四 问题的分析**

**问题提炼： （这里以时间代价为例，剩下的类似，在后文一并写齐）**

**公交：**

每辆车将给出一条链表（list），该链表可以是单向链表、双向链表、环形链表

**单辆车：**汽车移动到下一个站（链表相邻两个点之间）的时间代价是3。只能在同1条链表上单向移动（即使可以双向也不应该来回移动，否则不满足贪心算法思想）。

**换乘：**不同的链表可能会覆盖同一个点。如果要通过一个点，从一条链转换到另一条链上，需要花费5时间代价。

**地铁：**

每条地铁类似一颗树，每个地铁站点都会连接若干站点（树枝或者树叶）。

**单地铁：**地铁站点之间的移动（树干上移动到相邻点），花费2.5时间代价。单个地铁站点所相连的站点（树干与相邻叶子的移动），由于不需要经过乘车，假设成0代价。

**换乘：**不同的树会覆盖到相同的节点，从一棵树转化的另一棵树，花费4时间代价

**公交与地铁之间的换乘**

就是两种数据结构会覆盖同1个点，在那个点有换乘代价。具体请往后看。

**难点指出：**

这道题难点在于，他有四个地点属性{位置站点S，汽车路线L，地铁站口D，地铁线路T}，如果单独用任一个去表达图的节点，难以准确表达情况。

**五 模型的说明**

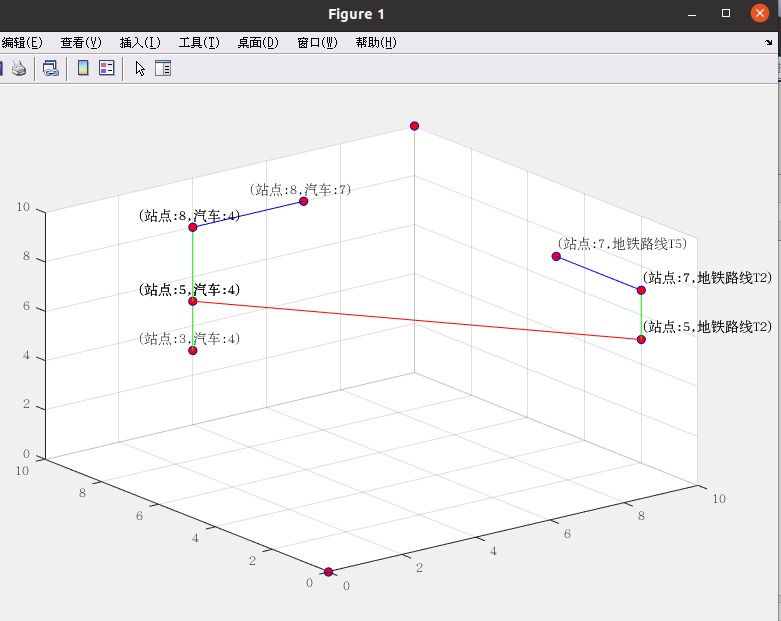
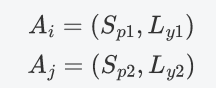


图5.1 交通系统整体三维模型图

（简述：在这里，记左汽车平面为A，右地铁平面为B，绿色线表示的是乘坐单辆交通工具移动到不同的站，蓝色线表示在同1个站点，汽车与汽车的换乘（A）或地铁与地铁的换乘（B），红色线表示汽车与地铁之间的换乘。当站点相同时，他们拥有相同的z高度（同一z平面））

首先要借重映射来建立简洁模型。

重映射这个技巧，比较巧妙的是，可以把(站点,汽车号)、(站点、地铁点)当做2个平面来看。对于汽车平面，假如



其中Sx是站点号码，Ly是汽车号码，Ai表示当你在Ly这辆车上并且开到Sp这个站点。

那么对于任意两点Ai，Aj，如果存在关系，就可以发现：

1.如果是行驶，同一辆车，那么Ly不变，Sp改变，即y1=y2，p1≠p2，建立行驶边（绿色）。

2.如果是换乘，同一站点，那么Sp不变，Ly改变，即p1=p2，y1≠y2，建立换乘边（蓝色）。

地铁平面也是同理的。只不过还要再套一个映射：假设在Tx条地铁，Dy的地铁站点，与那么可以用Bi=(Sp,(Tx,Dy))来代表地铁这一“平面”。

总结来说就是，先将原先的{Sp,Ly,Tx,Dy}数据，映射成{Ai,Bi}这两种节点。

Ai与Bi的连边，就是在两个平面之间的连边，表示地铁对公交或者公交对地铁的换乘（红色）。

但是这些边是已经转化过的，原始数据并没有那么简单。

对于汽车来说，我们要将一条“链”上，即每条公交路线上的任意两点之间都要建立连边。这是公交路线的图重构。还好“链”上任意两点的价格代价和时间代价都是线性的并与距离有关，且我们可以视每条公交独立，只需要再建立换乘边即可构建出来整个公交系统。

但是地铁就不一样了，整个地铁系统任意两点之间的价格代价都是3，这就导致了不能独立看待每一条地铁路线，因为如果我们使用地铁换乘，那就相当于跑了3段地铁（地铁段1-换乘-地铁段2，而且如果数据更大将更不合理），但是实际我们只应该付1份的价格（当然，似乎也想到一个比较反常的想法，我们把换乘边的价格设为负数，这样加起来似乎也相当于只付了1份）。最保险的方法就是，我们建立一个地铁小模型，在这个小模型上跑一遍dijkstra来构造出任意两地铁口之间的实际时间和价格即可。

在后面统一列出具体的建立发现

此时就会发现，单独选一个属性来建模，难以构建一个比较清晰的三维立体情况，但是用重映射造出面之后，情况会非常清晰，并且重映射是可以多次嵌套使用的。

**时间复杂度判断：**

**大图dijkstra,为O((V+E)logV)**

从现实情况来看，点的出度，就是这个站点的公交车经过数和地铁经过数，一般来说，1个公交站牌不会超过30辆车，一个地铁站点不会超过5个换乘。

一句话，每个点的出度都相对非常小，可以视作均<50，他会是一个稀疏图。从读入数据来看，也符合这个结论。

而映射出来总节点数，数据给出来的站点数S在4000以内，车辆和地铁在600以内，相乘就是总映射点数，达不到3e6个点。（实际运行时，V远达不到1e5个点，E远达不到1e6条边）

对于稀疏图，dijkstra的时间复杂度可以视为O((V+E)logV)

单次查询计算数最坏远远不会到达1e8，家用pc每秒计算数2e8左右，因此会跑的非常快，可以满足赛场做题情况。

**汽车的图重建，为O(L\*n^2)**

其中L是汽车路线数，n是路线下的汽车站点数。该数值相对较小，可忽视

**地铁的图重建，为O(n^3)**

建立地铁小模型后跑图，重新转化出距离边，和汽车边一起放到大模型的图上面。

坏消息是每个点与点之间都要求，多源，因此是n^3。

好消息是，根据网上的信息有[2]：

1.每个城市之间的地铁互不相同。

2.每个城市的地铁口最多不到500（截止至2022年12月31日统计）

此外，由于这题是2007年的题目，地铁站口只有39个，因此对于这道题的数据而言，地铁图重建的时间非常小。即便使用更大规模的数据（不远超现实），也依旧不会有非常大的运行时间，

**结论：**

使用了重映射后，1、2、3题均满足dijkstra的跑图条件，均可以实现路径还原。

此时解这道题已经非常的明了，第1题是单汽车平面（即只使用汽车部分数据），第2题是立体模型。

第3题优先考虑带步行的时间，或者价格第1，带步行时间第2。2和3类似。

**0边权：**

无论哪一道题，都存在0数值的边权。

对于单源最短路的求取，一旦引入了0数值的边，如果不使用dijkstra这种每个点都只跑一次的算法，是非常致命的。因此，在其他部分论文中，由于没有找出正确的模型，套不上dijkstra，那么所能做的只有近似解。

而dijkstra两种情况都能处理。其实也与一二问类似，只是将原来行驶对应成0，将换乘对应成步行时间或传承次数而已。

**多源汇：**

额外需要注意的是，对于最终需求，同一站点Sp有多个满足条件点(Sp,Ly1)、(Sp,Ly2)、(Sp,Ly3)……（即，你可以通过搭不同的车到达这里的）。这实际就是处理多源点多汇点，对于需要跑的起点，建”超级源点”连向他们即可，终点同理。（从三维模型来看，就是从1个z平面转移到另1个z平面，并由此很容易看出模型可延伸性，在后文继续提及）

同时，对于(Sp,(Tx,Dy))，表示在地铁上但与这个公交站相连，但理论上你是处在地铁里的，还要额外算上换乘时间走到任意的(Sp,Ly)，才表示到达这个公交站。因此这里不把(Sp,(Tx,Dy))算成源汇点，或者说即使和源汇点相连（为了防止存在地铁而不存在公交，导致到达地铁站点时不能连到源汇点，当然给出来的这几组需求路线没有这样数据），我们让源汇点到地铁站点的边附上换乘的代价即可。

**六 模型的建立与求解**

三道题的具体思路大致相同，因此我合起来一起讲。

**公交车行驶和换乘分析**

**行驶**

数据有4种情况，分别为上行、下行、环形和双向。其中下行给出来的数据形式与上行类似，处理方式相同。而环形的数据，他的开头和末尾是相同的站点S。

由于时间代价与距离呈线性，因此先求距离dis。

对于上行，下行，环形，他们都是从左侧节点移动到右侧节点，因此写双循环，令i<j，那么他们就是所有的从i到j，距离dis=j-i

环形比较特殊，考虑开头和结尾同站点，他的距离dis=min(j-i,len-(j-i)-1)，其中len表示这条路线的长度（站点数）。前者表示向右，后者是向左的环形距离。

对于环形和双向，他们都可以从j到i。实际这条边的权值与从i到j一样。因此就是建一个反向边而已

**换乘**

行驶只需要读入数据中每一条链即可。但换乘还要处理一下，我们使用数据结构set，记录每个站点有哪些公交车，这些公交车之间两两建边即可。

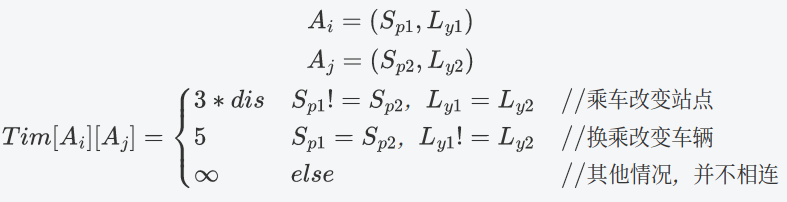


图6.1公交汽车移动和换乘的时间代价

对于第三题，如果要考虑步行，那么就是换乘有步行代价

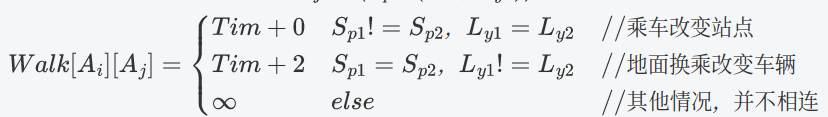


图6.2公交汽车移动和换乘的总时间代价

如果是统一1元，那么f(dis)=1，否则代入分段函数即可，换乘没有花费。

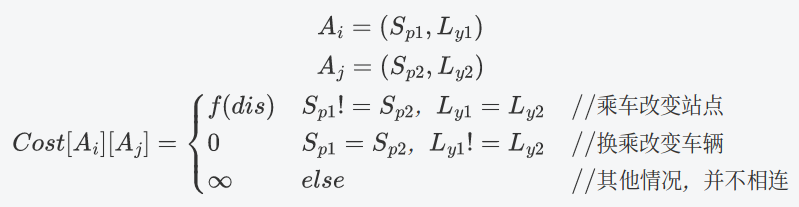


图6.3公交汽车移动和换乘的价格代价

转乘次数

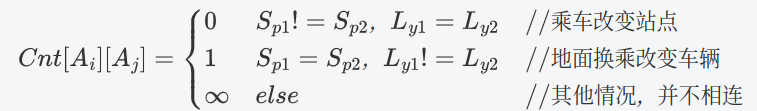


图6.4公交汽车移动和换乘的转乘次数

**地铁小模型的建立与图重建**

由于我们是联合4个元素，然后按照优先度来进行不同要求优先的最短路查询。但是地铁有一个元素价格Cost，他不是线性的，地铁任意两点直接价格都是3。因此我们不能当做简单链或其他形式直接根据原始数据逐个相连，否则他们跑出来价格不对。我们通过小模型跑出多源，重新构建任意两两之间的正确边权。

首先我们的目的是处理任意两两(Si1,Tj1,Dk1)、(Si2,Tj2,Dk2)的距离。我们再通过定义(-1,T,D)，具体的意义不好表达，你可以理解为(S,T,D)表示处在等待区（类似与树形结构的枝叶），(-1,T,D)表示当前位置处于地铁上（类似与树形结构的主干）。我们要做到是将这枝叶两两之间的距离算出来。那么就有

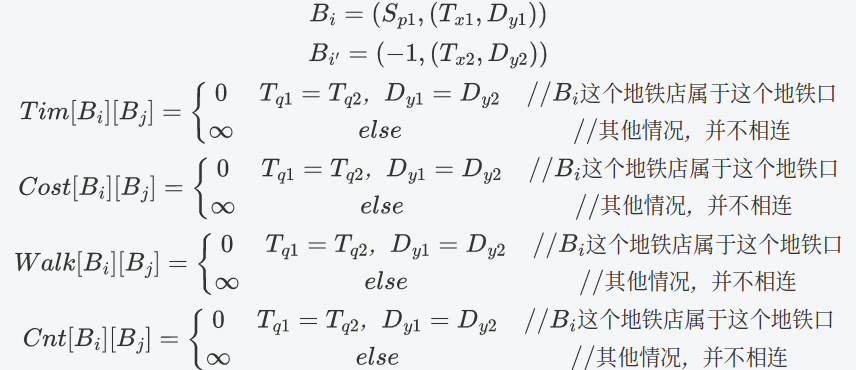


图6.5地铁类点Bi与地铁口虚拟点相连

然后就可以根据数据，将这些地铁口路线和换乘给连起来

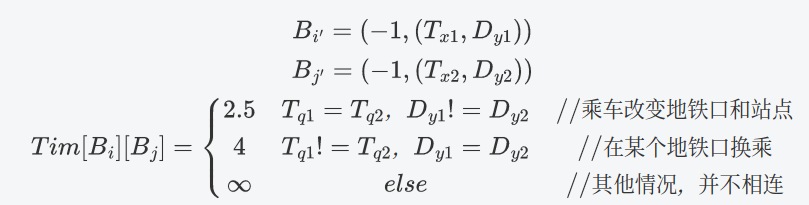
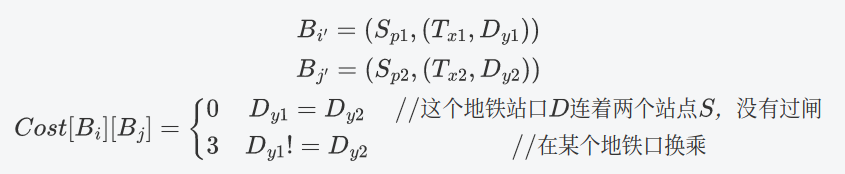


图6.6地铁类点Bi与地铁口虚拟点相连

由于并不需要计算中间(-1,T,D)的两两情况，只考虑(S,T,D)的情况，因此跑若干次单源可能优于跑一次多源。当然由于数据非常小他们相差不大。

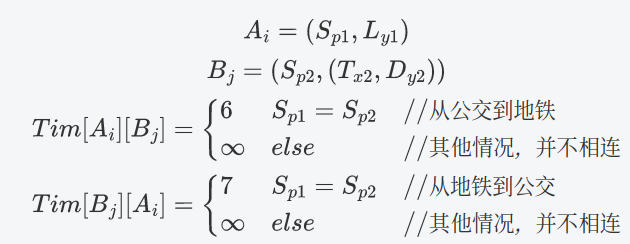
建立完小模型跑出来后，就可以得到任意两点的时间代价Tim，总时间代价Walk，转乘次数Cnt，然后他们的价格代价是：

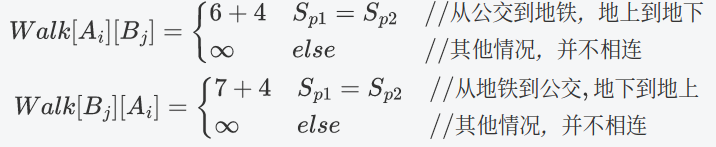
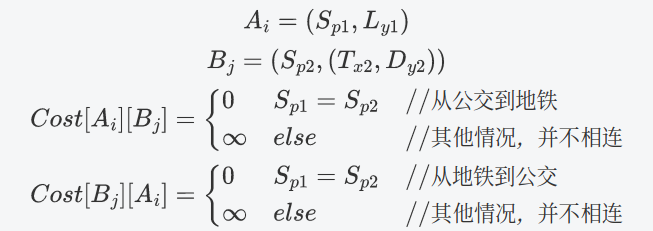


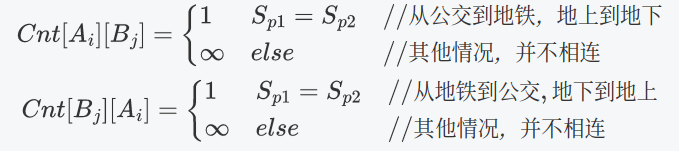
此外，地铁小模型被我跑了两次，因为第二题的需求是以乘车时间优先，第三题需求是以总时间(带步行)优先，尽管这道题数据较小，这两种情况跑出来重建边是一摸一样的，但是还是严格遵循了题目需求。

**汽车与地铁的换乘**

同样，我们用数据结构set记录S站点有哪些汽车和地铁，然后两两连边即可。







补充：

首先题目存在且仅存在一个2.5的时间小数。为了避免精度误差，我们将所有的时间\*2，再最后算出来后再/2即可。上文没有写入这个\*2，但是代码中有运用到。

**七 解的验证**

问题一

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点  ->终点 | S3359  ->S1828 | S1557  ->S0481 | S0971  ->S0485 | S0008  ->S0073 | S0148  ->S0485 | S0087  ->S3676 |
| 耗时最优(min) | 64 | 99 | 103 | 59 | 102 | 46 |
| 费用最优(元) | 3 | 3 | 3 | 2 | 3 | 2 |

问题二

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点  ->终点 | S3359  ->S1828 | S1557  ->S0481 | S0971  ->S0485 | S0008  ->S0073 | S0148  ->S0485 | S0087  ->S3676 |
| 耗时最优(min) | 62 | 99 | 95 | 53.5 | 86.5 | 33 |
| 费用最优(元) | 3 | 3 | 3 | 2 | 3 | 1 |

问题三

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 起点  ->终点 | S3359  ->S1828 | S1557  ->S0481 | S0971  ->S0485 | S0008  ->S0073 | S0148  ->S0485 | S0087  ->S3676 |
| 耗时最优(min)  (带步行时间) | 68 | 105 | 104 | 63 | 95.5 | 41 |
| 费用最优(元) | 3 | 3 | 3 | 2 | 3 | 1 |

具体的路线还原请看附录，或者查看代码进行测试。

**八 模型的优缺点和延伸**

**模型的优点：**

1. 模型简洁直观，并且贴合实际
2. 算法直观，编程容易实现
3. 低时间复杂度，高效查询速度(预处理5秒以内，每次查询1秒以内全结果)
4. 实现理论准确解，没有近似计算

**模型的缺点：**

1. 存在假设，结果与现实可能有差距

**模型的改进：**

为了最求实现速度，并没有运用指针等可以优化空间开销的操作，可以改进

**模型的延伸：**

图的搜索，也可以当做一种状态的转移，我们尝试推广这个交通模型。

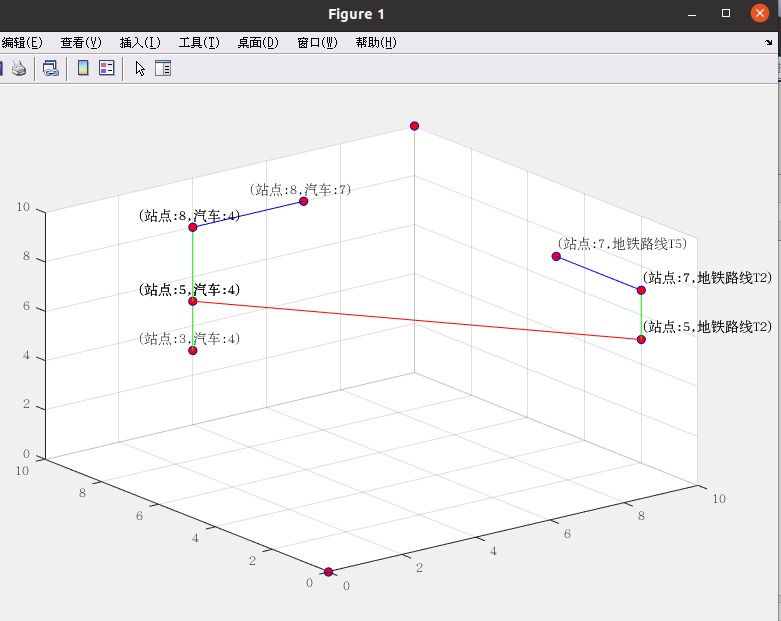


图7.1 交通系统整体三维模型图

对于每个z状态（这里表现为z平面），我们可以假设当拥有参数为(x1,x2,x3……)

对于这个交通模型，z状态就是处于哪个汽车站点S。其实有三个参数(汽车号L，地铁号T，地铁站点D)，但被我重新映射成了2个参数，其中一个是(-1,L)，另一个是(T,D)。这两个参数之间是互相独立不能同时共存的(可能表达不大准确，见谅)。

具体表现就是，这些点要么只存在与定x平面（汽车），要么只存在定y平面（地铁）。

交通模型是这种模型的一个特例。z状态不一定是平面，可以使三维、四维……我们可以简单总结出：

1. 一种状态z有多个参数。
2. 尽管参数不同，也可能代表同一个状态，可视为点在这个z状态里。
3. 对于同1个状态内，可以花费代价转移（花费代价改变参数，状态不变）
4. 同1个状态参数不同，会使得这个从状态变成另1个状态付出的代价不同。

因此，它有较好的适应性，通过有效建议能本质上满足各种多参数之间的转移。最容易想到的便是化学反应关系的建立。此外，这种状态的转移能在人工智能领域有所运用。

**九 参考文献**

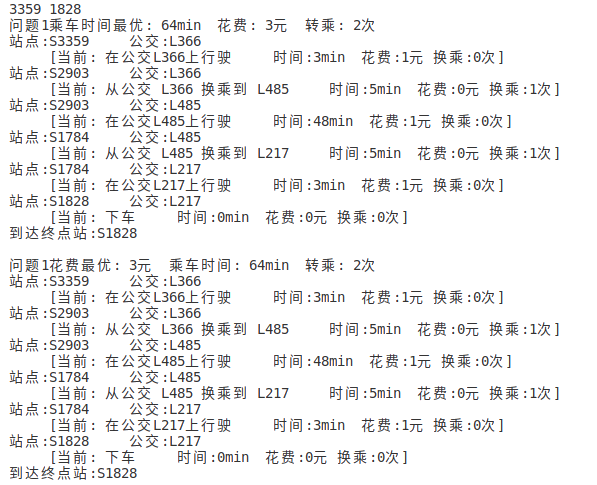
1. 熊国刚 王杰 黎明,乘公交 看奥运,百度文库,2007,可获取自: [https://wenku.baidu.com/view/81945a17cc7931b765ce15f6.html?\_wkts\_=1686057927399](https://wenku.baidu.com/view/81945a17cc7931b765ce15f6.html?_wkts_=1686057927399" \t "/home/beikuzi/文档\\x/_new).
2. 名品咨询,2022各城市地铁站数量排行,2022,可获取自:

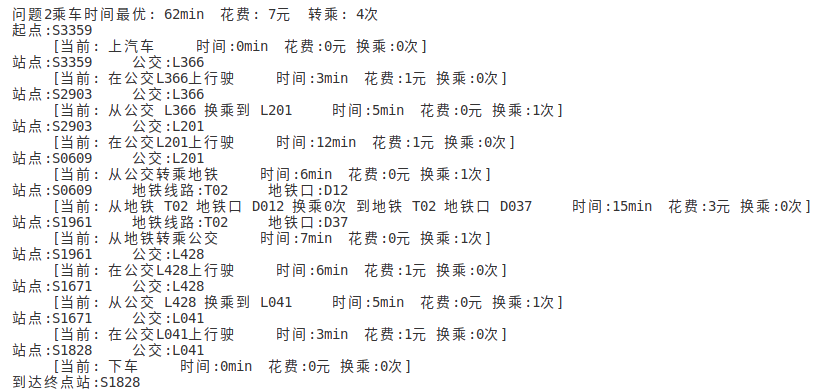
https://www.maigoo.com/news/662035.html

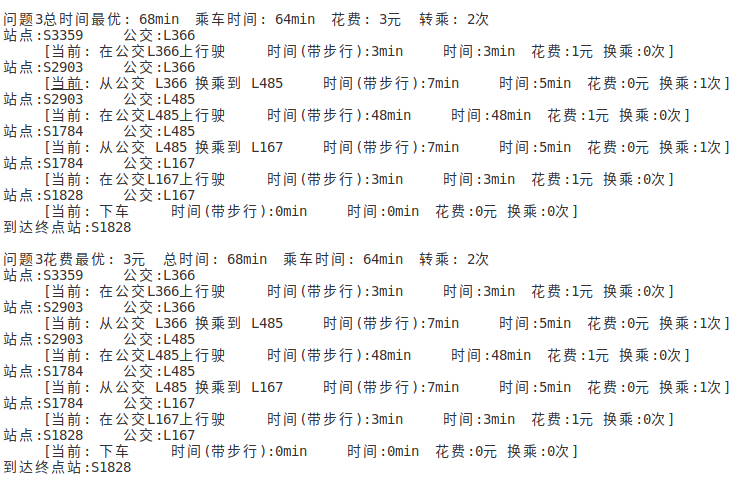
**十 附录**

题目要求六条路线的具体路径还原

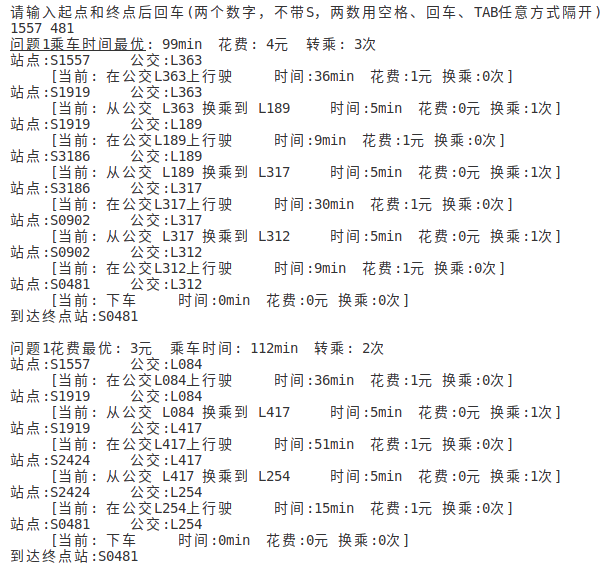
路线1

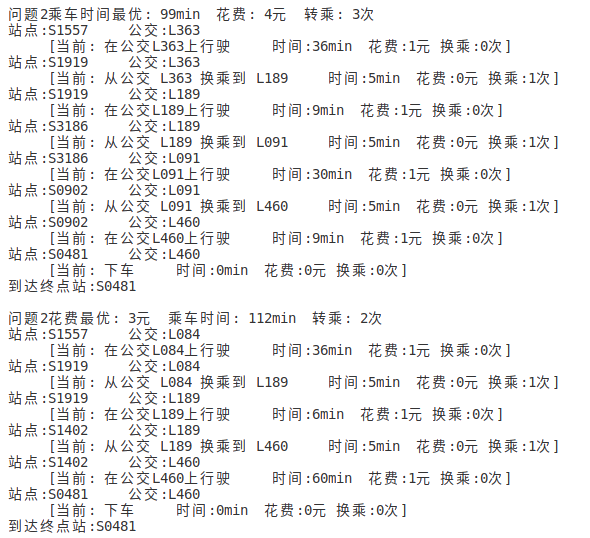


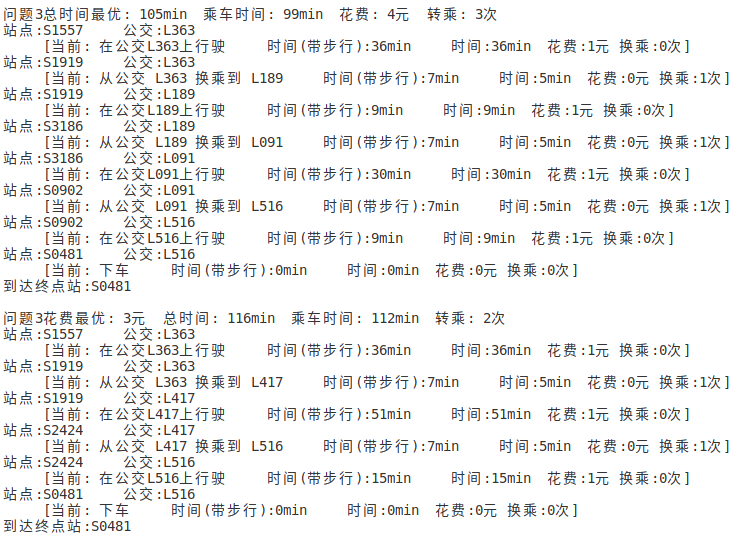




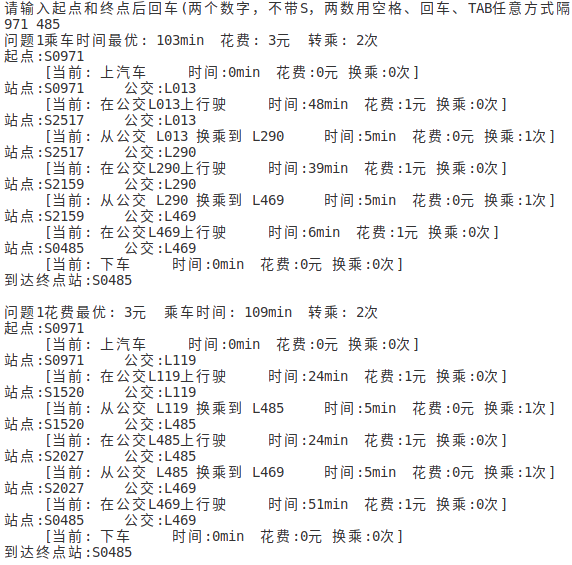
路线2

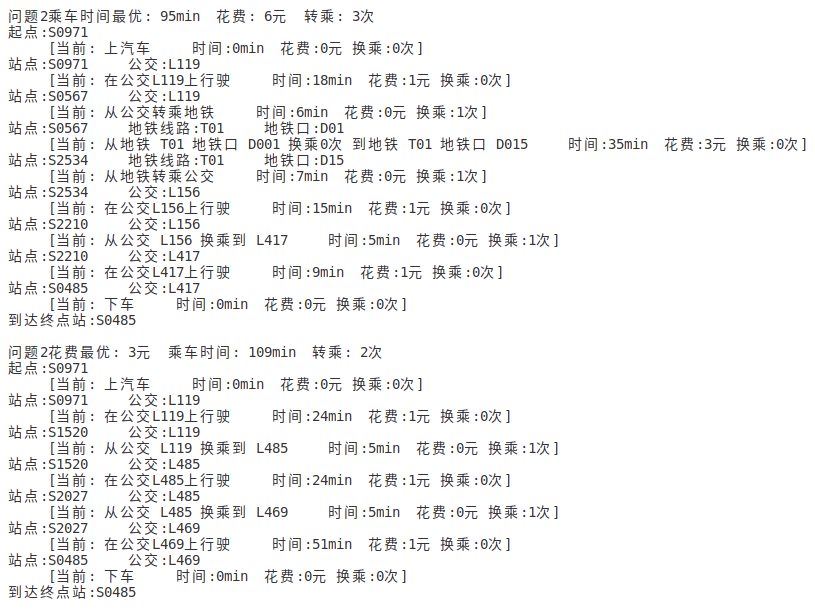


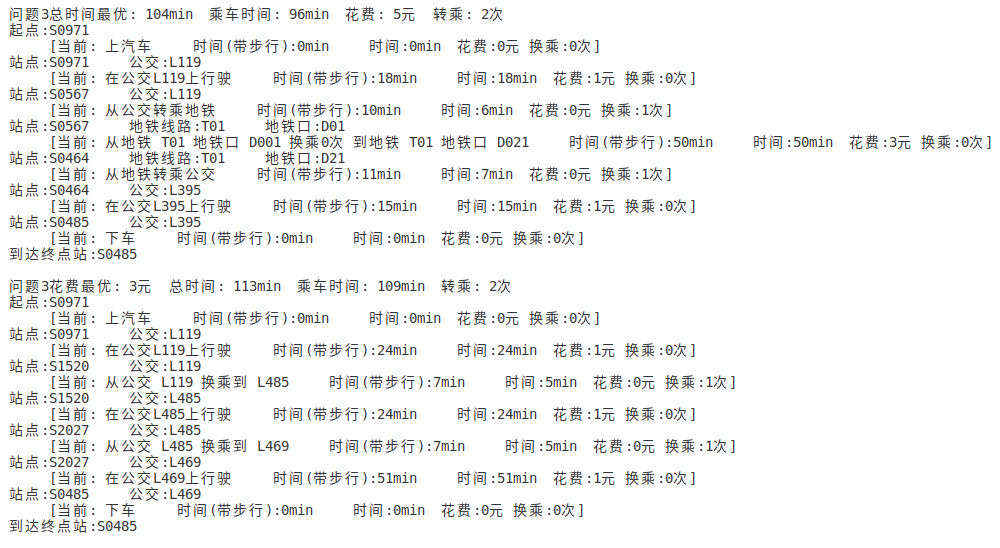
’



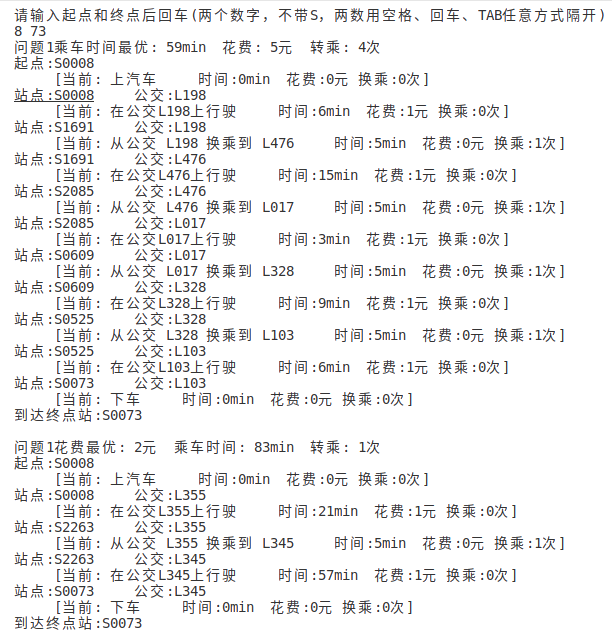
路线3

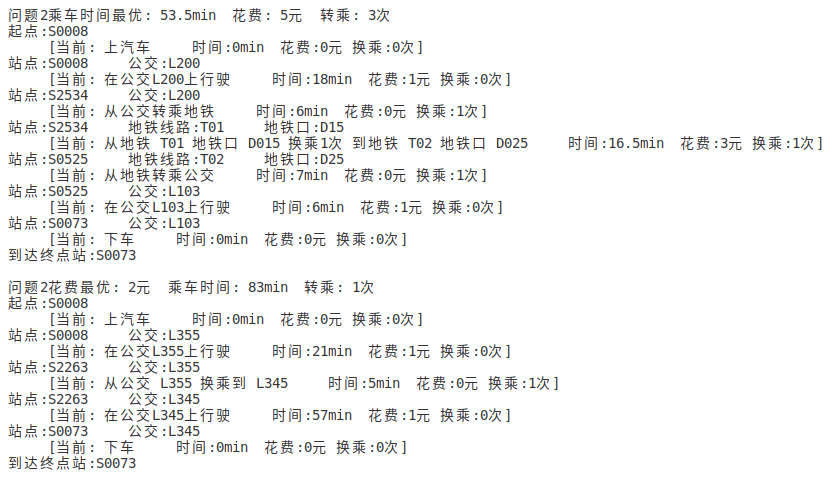


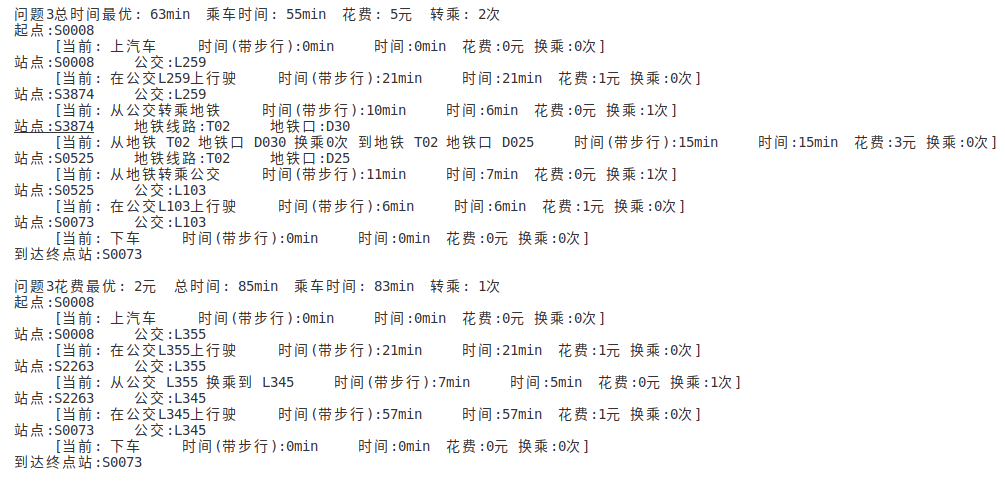




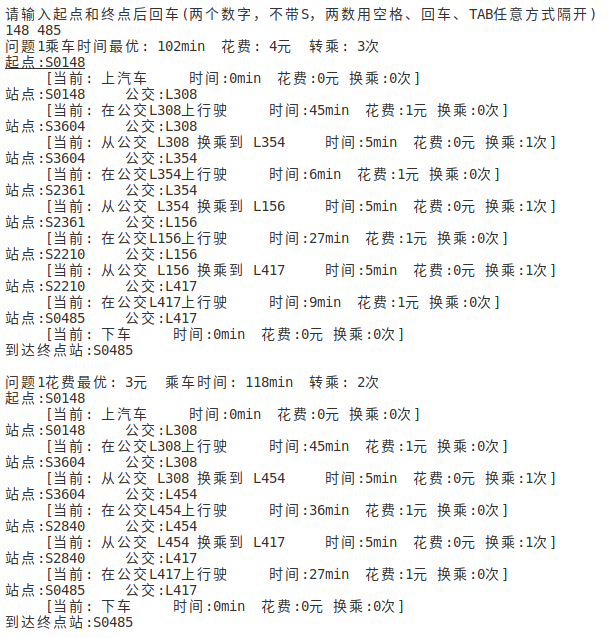
路线4

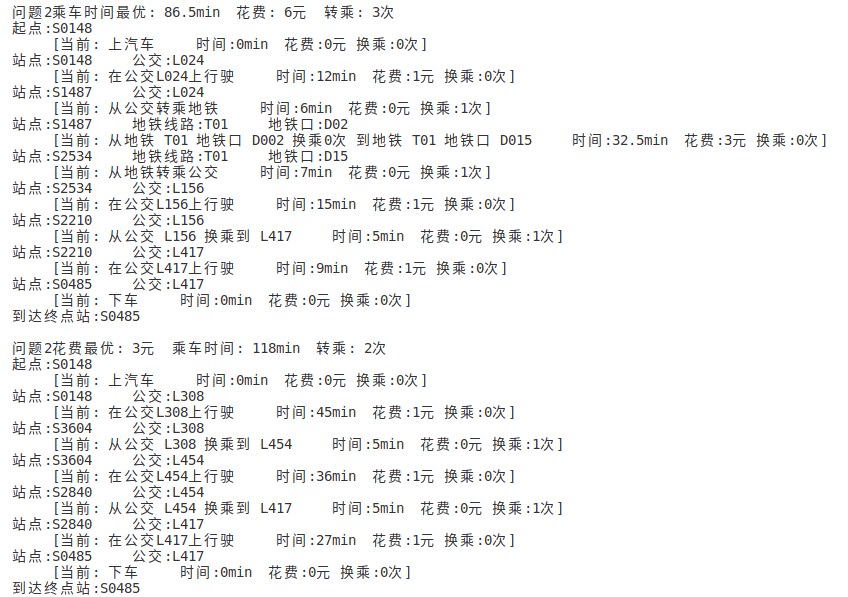


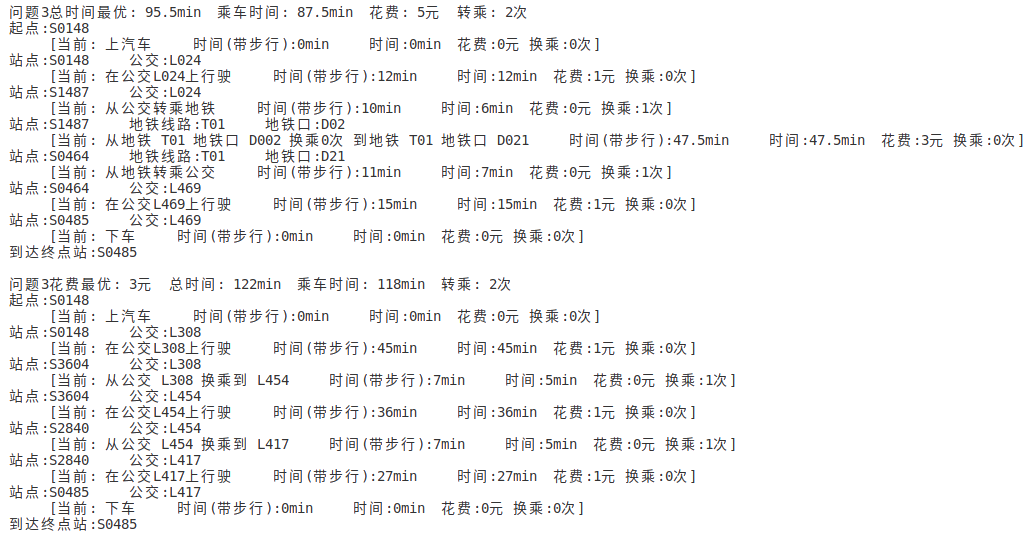




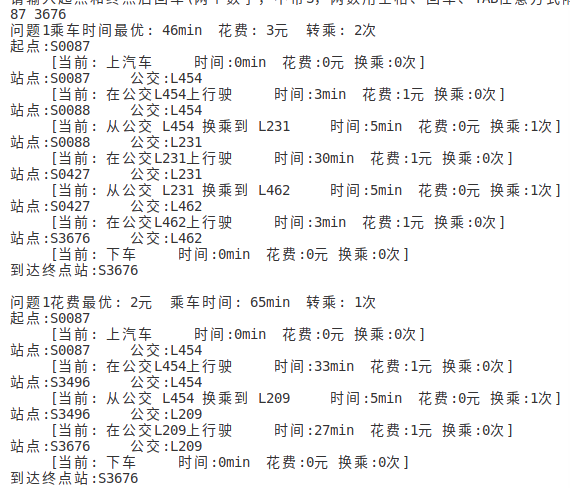
路线5

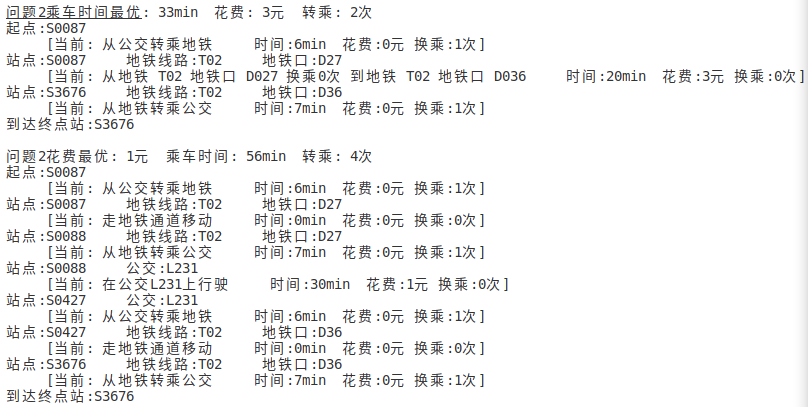


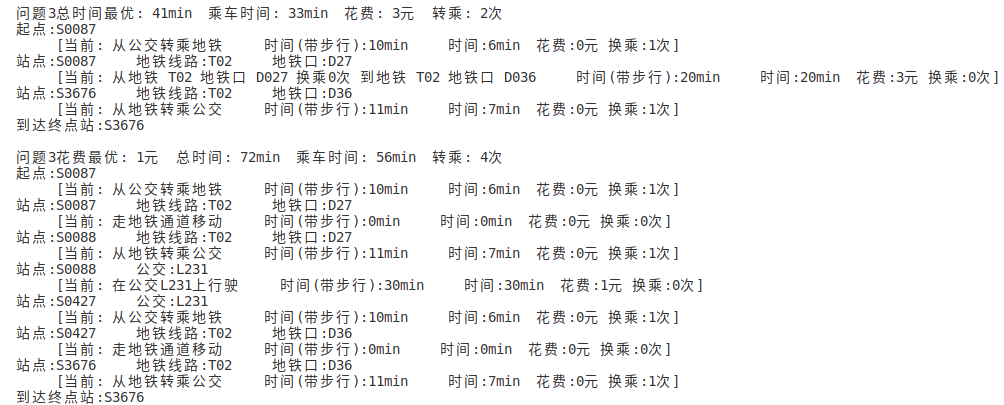




路线6







matlab空间模型

figure(1);

grid on;

title('交通系统');

xlabel('X');

ylabel('Y');

zlabel('Z');

global x y z

x = [];

y = [];

z = [];

addP(0,0,0);

addP(10,10,10);

addP(4,-1,3);

addP(4,-1,5);

addP(-1,2,5);

addP(-1,2,7);

addP(4,-1,8);

addP(7,-1,8);

addP(-1,5,7);

scatter3(x, y, z, 'filled', 'Marker', 'o', 'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'r');

linkP(4,5,'r');

linkP(3,4,'g');

linkP(5,6,'g');

linkP(4,7,'g');

linkP(7,8,'b');

linkP(6,9,'b');

function linkP(i,j,color)

global x y z

line([x(i),x(j)],[y(i),y(j)],[z(i),z(j)],'color',color);

if y(i)==10

text(x(i)-1.5,y(i),z(i)+1,"(站点:"+num2str(z(i))+",汽车:"+num2str(x(i))+")");

else

text(x(i),y(i),z(i)+0.5,"(站点:"+num2str(z(i))+",地铁口"+num2str(y(i))+")");

end

i=j;

if y(i)==10

text(x(i)-1.5,y(i),z(i)+1,"(站点:"+num2str(z(i))+",汽车:"+num2str(x(i))+")");

else

text(x(i),y(i),z(i)+0.5,"(站点:"+num2str(z(i))+",地铁口"+num2str(y(i))+")");

end

end

function addP(i,j,k)

if i==-1,i=10;end

if j==-1,j=10;end

global x y z

x=[x i];

y=[y j];

z=[z k];

end

三个问题写在同一程序下：

#include<iostream>

#include<cstdio>

#include<algorithm>

#include<stack>

#include<queue>

#include<map>

#include<set>

#include<vector>

#include<string>

#include<cstring>

#include<ctime>

#include<cstdlib>

#include<iomanip>

#include<fstream>

#include<sstream>

using namespace std;

ofstream fout;

void tst(string s){

//cout<<time(nullptr)<<'\n';

//fout<<"处理： "<<s<<'\n';

cout<<"处理： "<<s<<'\n';

}

struct edge{

int u,v,next;

int Tim,Cost,Walk,Cnt;

};

vector<edge>e,//用来跑图的边

ebus,//第一题的边

eTim,//第二题，地铁以时间为第一优先度进行图重构之后，汽车+地铁所有边

eWalk;//第三题，步行+乘坐第1的重构

const int maxn=2e6,maxm=5e6;

vector<int>head,head1,head2,head3;//跑图头边，第1，2，3题的头个边

//int head[maxn],head1[maxn],head2[maxn];

int k,k1,k2,k3;

void add(int u,int v,int Tim,int Cost,int Walk,int Cnt){

edge etemp;

etemp.next=head[u];

head[u]=++k;

etemp.u=u,etemp.v=v,

etemp.Tim=Tim,etemp.Cost=Cost,etemp.Walk=Walk,etemp.Cnt=Cnt;

e.push\_back(etemp);

}

void add(int u,edge etemp){

etemp.u=u;

etemp.next=head[u];

head[u]=++k;

e.push\_back(etemp);

}

typedef pair<int,int> pii;

map<pii,int>TP;//transportMap，交通映射

//1.将地铁路线T，地铁站点D映射成点对p

//2.或者将-1，公交路线L映射成点对p

map<int,pii>RTP;//reverseTransportMap，逆还原

int turn(int x,int y){//生成一个新的映射交通点

static int Tcnt=0;//交通点数量

if(TP[{x,y}])return TP[{x,y}];

RTP[++Tcnt]={x,y};

return TP[{x,y}]=Tcnt;

}

map<pii,int>PP;//planeMap，平面点映射

map<int,pii>RPP;//逆还原

int Pcnt=0;//总共有多少个虚拟点

int turn2(int S,int tp){//公交站和交通点，生成一个新的映射平面点

if(PP[{S,tp}])return PP[{S,tp}];

RPP[++Pcnt]={S,tp};

return PP[{S,tp}]=Pcnt;

}

struct text{//文本

ifstream fin;

string file1="1.1 公汽线路信息.txt";

string file2="1.2 地铁线路信息.txt";

string file3="2.1 地铁T1线换乘公汽信息.txt";

string file4="2.2 地铁T2线换乘公汽信息.txt";

static const int maxL=1000,maxM=100;

string Lmsg[maxL][3];//文本1读入

string Mmsg[maxM][3];//文本2读入

vector<string>Dmsg[maxM];//文本3和文本4读入，不同的地铁站点D存在的站点S

int Lcnt=0;//读入有多少条公交车路线

int Mcnt=0;//有多少条地铁路线

int Scnt=0;//经过读入处理后，得出来有多少个汽车站点S

int Dcnt=0;//有多少个地铁站点D

void read(){

fin.open(file1);

string s;

while(getline(fin,s)){

if(s[0]!='L')break;

Lcnt++;

for(int i=0;i<3;i++){

getline(fin,s);

Lmsg[Lcnt][i]=s;

}

}

fin.close();fin.open(file2);

while(getline(fin,s)){

if(s[0]!='T')break;

Mcnt++;

for(int i=0;i<3;i++){

getline(fin,s);

Mmsg[Mcnt][i]=s;

}

}

fin.close();fin.open(file3);

int now=1;//第一条地铁路线

while(getline(fin,s)){

if(s[0]!='D')break;

Dmsg[now].push\_back(s);

}

fin.close();fin.open(file4);

now=2;//第二条地铁路线

while(getline(fin,s)){

if(s[0]!='D')break;

Dmsg[now].push\_back(s);

}

fin.close();

}

}txt;

int sget(string s="缺省"){//从这个字符串中获取数字，直到读入字符串结束

static string sread;

static int now=0;

static int cnt=0;

if(s!="缺省"){//初始化

sread=s;

now=0;

return -1;

}

int res=0;

bool flag=0;

while(sread[now]<'0' or sread[now]>'9'){//将所有非数字去掉

if(sread[now]=='\0')break;

now++;

}

while(sread[now]>='0' and sread[now]<='9'){

if(sread[now]=='\0')break;

flag=1;

res=(res<<3)+(res<<1)+sread[now]-48;//res\*10+sread[now]

now++;

}

if(sread[now]=='\0'){//通过让sread清空，来表示已经读取完所有的数据

sread="";

now=0;

}

if(flag)return res;

return -1;

}

const int maxs=5000;//数据中共有4000个S站点

//SL使用于源点汇点，SL和SM组合使用于地铁和公交换乘,MS用于站点通过地铁地上通达换站

set<int>SL[maxs];//与经过S站点的公交L有哪些

int addcnt=0;

void Bus(int L,int cas,int dir,string msg){//第L条公交车，计价模式cas和方向dir

sget(msg);

vector<int>vt;

int res=0,len=0;

while((res=sget())!=-1){

vt.push\_back(res);

len++;

SL[res].insert(L);

txt.Scnt=max(txt.Scnt,res);

}

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

for(int j=i+1;j<len;j++){

edge etemp;//转乘cnt=0

etemp.Cnt=0;

int dis=(dir==3)?min(j-i,len-(j-i)-1):j-i;//距离站数

//环形min(j-i,len-(j-i)-1)，-1是因为文本额外将头尾给出，其他j-i，自身相连无所谓

int u=turn2(vt[i],turn(-1,L));

int v=turn2(vt[j],turn(-1,L));

etemp.v=v;

etemp.Tim=dis\*3\*2;//时间都算2倍来去掉小数

etemp.Walk=etemp.Tim+0;//无步行时间，总时间就是Tim

if(cas==1)etemp.Cost=1;//同一1元

else{

int ticket;

if(dis<21)ticket=1;

else if(dis<41)ticket=2;

else ticket=3;

etemp.Cost=ticket;

}

add(u,etemp);

addcnt++;

if(dir==3 or dir==4){//对于双向，两个点之间的计算值一样，直接反相边

etemp.v=u;

add(v,etemp);

addcnt++;

}

}

}

}

void dealBus(){//处理公交车移动,当选择第三题时，要一并算入步行时间和转乘

int Lcnt=txt.Lcnt;//公交路线数

int cas=0;//计价模式

string temp;

for(int i=1;i<=Lcnt;i++){

temp=txt.Lmsg[i][0].substr(0,6);

if(temp=="单一"){

cas=1;

}

else if(temp=="分段"){

cas=2;

}

else{

cout<<"数据格式错误: "<<Lcnt<<' '<<txt.Lmsg[Lcnt][0]<<'\n';

continue;

}

temp=txt.Lmsg[i][1].substr(0,6);

if(temp=="上行"){

Bus(i,cas,1,txt.Lmsg[i][1]);

Bus(i,cas,2,txt.Lmsg[i][2]);

}

else if(temp=="环形"){

Bus(i,cas,3,txt.Lmsg[i][1]);

}

else{

Bus(i,cas,4,txt.Lmsg[i][1]);

}

}

}

void dealBus2(){//处理公交车换乘,当选择第三题时，要一并算入步行时间和转乘

edge etemp;

int maxS=txt.Scnt;

etemp.Cost=0;//没有换乘费用

etemp.Tim=5\*2;//汽车换乘等待5s，所有时间\*2

etemp.Cnt=1;//换乘1次

etemp.Walk=7\*2;//步行+换乘7分钟

set<int>::iterator it1,it2;

for(int i=1;i<=maxS;i++){//同一位置的公交车两两互通换乘

for(it1=SL[i].begin();it1!=SL[i].end();++it1){

for(it2=SL[i].begin();it2!=SL[i].end();++it2){

if((\*it1)==(\*it2))continue;

//cout<<(\*it1)<<' '<<(\*it2)<<'\n';

int u=turn2(i,turn(-1,(\*it1)));

int v=turn2(i,turn(-1,(\*it2)));

etemp.v=v;

add(u,etemp);

addcnt++;

}

}

}

}

const int maxd=maxs;

set<int>DT[maxd];//记录地铁站点D有哪些地铁路线经过

set<int>DS[maxd];//记录地铁站点D有哪些汽车站点S

set<int>SM[maxs];//经过S站点的地铁M(T,D)有哪些

set<int>sNode;//记录了所有的(S,(T,D))点

//set<int>MS[maxs];//与地铁口M相连的S有哪些

void Metro(int T,int cas,int dir,string msg){//第T条地铁，计价模式cas和方向dir

sget(msg);

vector<int>vt;

int res=0,len=0;

while((res=sget())!=-1){

vt.push\_back(res);

len++;

DT[res].insert(T);//记录这个站点

txt.Dcnt=max(txt.Dcnt,res);

}

edge etemp;

etemp.Tim=5;//2.5\*2

etemp.Cost=0;

etemp.Walk=etemp.Tim+0;

etemp.Cnt=0;

if(dir==1 or dir==2 or dir==3 or dir==4){//向右侧相连

for(int i=0;i<len-1;i++){

int u=turn2(-1,turn(T,vt[i]));

int v=turn2(-1,turn(T,vt[i+1]));

etemp.v=v;

add(u,etemp);

addcnt++;

}

}

if(dir==3 or dir==4){//向左侧相连

for(int i=len-1;i>0;i--){

int u=turn2(-1,turn(T,vt[i]));

int v=turn2(-1,turn(T,vt[i-1]));

etemp.v=v;

add(u,etemp);

addcnt++;

}

}

}

void dealMetro(){//实现地铁行驶边建立和地铁换乘边建立

int Mcnt=txt.Mcnt;//地铁路线数

int cas=0;//计价模式

string temp;

//(-1,T,D)之间相连

for(int i=1;i<=Mcnt;i++){

temp=txt.Mmsg[i][0].substr(0,6);

if(temp=="票价"){//统一3元

cas=1;

}

else{

cout<<"数据格式错误: "<<i<<' '<<txt.Mmsg[Mcnt][0]<<'\n';

continue;

}

temp=txt.Mmsg[i][1].substr(0,6);

if(temp=="上行"){

Metro(i,cas,1,txt.Mmsg[i][1]);

Metro(i,cas,2,txt.Mmsg[i][2]);

}

else if(temp=="环形"){

Metro(i,cas,3,txt.Mmsg[i][1]);

}

else{

Metro(i,cas,4,txt.Mmsg[i][1]);

}

}

//(-1,T,D)与(S,T,D)相连

for(int i=1;i<=Mcnt;i++){//读取并记录所有与D相连的S

vector<string>dmsg=txt.Dmsg[i];//第i条地铁信息

int len=dmsg.size();

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

sget(dmsg[j]);

int D=sget();

int S=-1;

while((S=sget())!=-1){

DS[D].insert(S);

}

}

}

set<int>::iterator it1,it2,it3;

int Dcnt=txt.Dcnt;

for(int i=1;i<=Dcnt;i++){//内部已经重构

for(it1=DT[i].begin();it1!=DT[i].end();it1++){

for(it2=DS[i].begin();it2!=DS[i].end();it2++){

int D=i,T=\*it1,S=\*it2;

SM[S].insert(turn(T,D));

txt.Scnt=max(txt.Scnt,S);

int u=turn2(S,turn(T,D));

sNode.insert(u);

int v=turn2(-1,turn(T,D));

add(u,v,0,0,0,0);

add(v,u,0,0,0,0);

addcnt+=2;

}

}

}

}

void dealMetro2(){//地铁换乘

set<int>::iterator it1,it2;

edge etemp;

etemp.Tim=4\*2;//换乘4分钟

etemp.Walk=6\*2;//加上步行换乘6分钟

etemp.Cnt=1;

etemp.Cost=0;

int Dcnt=txt.Dcnt;

for(int i=1;i<=Dcnt;i++){

for(it1=DT[i].begin();it1!=DT[i].end();it1++){

for(it2=DT[i].begin();it2!=DT[i].end();it2++){

if((\*it1)==(\*it2))continue;

int u=turn2(-1,turn(\*it1,i));

int v=turn2(-1,turn(\*it2,i));

etemp.v=v;add(u,etemp);

etemp.v=u;add(v,etemp);//换乘双向

addcnt+=2;

}

}

}

}

void dealBus\_Metro(){//处理公交和地铁的换乘,以及通过相同地铁口来到达不同公交点

int Scnt=txt.Scnt;

set<int>::iterator it1,it2;

edge etemp1,etemp2;

etemp1.Cost=etemp2.Cost=0;

etemp1.Cnt=etemp2.Cnt=1;

etemp1.Tim=7\*2;//从地铁到公交

etemp1.Walk=(7+4)\*2;

etemp2.Tim=6\*2;

etemp2.Walk=(6+4)\*2;

for(int i=1;i<=Scnt;i++){

for(it1=SL[i].begin();it1!=SL[i].end();it1++){

for(it2=SM[i].begin();it2!=SM[i].end();it2++){

int u=turn2(i,turn(-1,(\*it1)));

int v=turn2(i,(\*it2));

etemp1.v=u;//到公交

etemp2.v=v;//到地铁

add(v,etemp1);

add(u,etemp2);

addcnt+=2;

}

}

}

}

struct node{//映射的平面虚拟点最终跑出来的距离

int id;

int Tim,Cost,Walk,Cnt;

}nd[maxn];

struct sortTim{//时间优先

bool operator()(const node &n1,const node &n2){

if(n1.Tim!=n2.Tim)return n1.Tim>n2.Tim;

if(n1.Cost!=n2.Cost)return n1.Cost>n2.Cost;

if(n1.Cnt!=n2.Cnt)return n1.Cnt>n2.Cnt;

return false;

}

};

struct sortCost{//不考虑步行的价格优先

bool operator()(const node &n1,const node &n2){

if(n1.Cost!=n2.Cost)return n1.Cost>n2.Cost;

if(n1.Tim!=n2.Tim)return n1.Tim>n2.Tim;

if(n1.Cnt!=n2.Cnt)return n1.Cnt>n2.Cnt;

return false;

}

};

struct sortCost2{//考虑步行的价格优先

bool operator()(const node &n1,const node &n2){

if(n1.Cost!=n2.Cost)return n1.Cost>n2.Cost;

if(n1.Walk!=n2.Walk)return n1.Walk>n2.Walk;

if(n1.Tim!=n2.Tim)return n1.Tim>n2.Tim;

if(n1.Cnt!=n2.Cnt)return n1.Cnt>n2.Cnt;

return false;

}

};

struct sortWalk{//步行优先

bool operator()(const node &n1,const node &n2){

if(n1.Walk!=n2.Walk)return n1.Walk>n2.Walk;

if(n1.Tim!=n2.Tim)return n1.Tim>n2.Tim;

if(n1.Cost!=n2.Cost)return n1.Cost>n2.Cost;

if(n1.Cnt!=n2.Cnt)return n1.Cnt>n2.Cnt;

return false;

}

};

int preu[maxn],prew[maxn];

int SP,EP;//输入起点终点

int ST,ED;//重映射后跑图的起点终点

int vis[maxn];

void dijk\_init(){

memset(vis,0,sizeof(vis));

memset(nd,0x3f,sizeof(nd));

memset(preu,-1,sizeof(preu));

memset(prew,-1,sizeof(prew));

nd[ST]={0,0,0,0,0};

nd[ST].id=ST;

}

bool dijkTim(){

dijk\_init();

priority\_queue<node,vector<node>,sortTim>q;

q.push(nd[ST]);

while(!q.empty()){

node u=q.top();q.pop();

// cout<<u.id<<'\n';

if(vis[u.id])continue;vis[u.id]=1;

if(u.id==ED)return 1;

for(int i=head[u.id];~i;i=e[i].next){

int vv=e[i].v;

node v=nd[vv];

edge w=e[i];

if(v.Tim>u.Tim+w.Tim){

v.id=vv;

v.Tim=u.Tim+w.Tim;

v.Cost=u.Cost+w.Cost;

v.Walk=u.Walk+w.Walk;

v.Cnt=u.Cnt+w.Cnt;

nd[vv]=v;

q.push(v);

preu[vv]=u.id,prew[vv]=i;//记录前向

}

}

}

return 0;

}

bool dijkCost(){

dijk\_init();

priority\_queue<node,vector<node>,sortCost>q;

q.push(nd[ST]);

while(!q.empty()){

node u=q.top();q.pop();

// cout<<u.id<<'\n';

if(vis[u.id])continue;vis[u.id]=1;

if(u.id==ED)return 1;

for(int i=head[u.id];~i;i=e[i].next){

int vv=e[i].v;

node v=nd[vv];

edge w=e[i];

//cout<<u.id<<' '<<u.Tim<<' '<<v.id<<' '<<v.Tim<<' '<<Tim<<'\n';

if(v.Cost>u.Cost+w.Cost){

v.id=vv;

v.Tim=u.Tim+w.Tim;

v.Cost=u.Cost+w.Cost;

v.Walk=u.Walk+w.Walk;

v.Cnt=u.Cnt+w.Cnt;

nd[vv]=v;

q.push(v);

preu[vv]=u.id,prew[vv]=i;//记录前向

}

}

}

return 0;

}

bool dijkWalk(){

dijk\_init();

priority\_queue<node,vector<node>,sortWalk>q;

q.push(nd[ST]);

while(!q.empty()){

node u=q.top();q.pop();

if(vis[u.id])continue;vis[u.id]=1;

if(u.id==ED)return 1;

for(int i=head[u.id];~i;i=e[i].next){

int vv=e[i].v;

edge w=e[i];

node v=nd[vv];

//cout<<u.id<<' '<<u.Tim<<' '<<v.id<<' '<<v.Tim<<' '<<Tim<<'\n';

if(v.Walk>u.Walk+w.Walk){

v.id=vv;

v.Tim=u.Tim+w.Tim;

v.Cost=u.Cost+w.Cost;

v.Walk=u.Walk+w.Walk;

v.Cnt=u.Cnt+w.Cnt;

nd[vv]=v;

q.push(v);

preu[vv]=u.id,prew[vv]=i;//记录前向

}

}

}

return 0;

}

bool dijkCost2(){

dijk\_init();

priority\_queue<node,vector<node>,sortCost2>q;

q.push(nd[ST]);

while(!q.empty()){

node u=q.top();q.pop();

// cout<<u.id<<'\n';

if(vis[u.id])continue;vis[u.id]=1;

if(u.id==ED)return 1;

for(int i=head[u.id];~i;i=e[i].next){

int vv=e[i].v;

node v=nd[vv];

edge w=e[i];

//cout<<u.id<<' '<<u.Tim<<' '<<v.id<<' '<<v.Tim<<' '<<Tim<<'\n';

if(v.Cost>u.Cost+w.Cost){

v.id=vv;

v.Tim=u.Tim+w.Tim;

v.Cost=u.Cost+w.Cost;

v.Walk=u.Walk+w.Walk;

v.Cnt=u.Cnt+w.Cnt;

nd[vv]=v;

q.push(v);

preu[vv]=u.id,prew[vv]=i;//记录前向

}

}

}

return 0;

}

void dijk2(){//第二题的地铁小模型跑图

set<int>::iterator it;

for(it=sNode.begin();it!=sNode.end();it++){

ST=\*it;

dijk\_init();

priority\_queue<node,vector<node>,sortTim>q;

q.push(nd[ST]);

while(!q.empty()){

node u=q.top();q.pop();

if(vis[u.id])continue;vis[u.id]=1;

if(RPP[u.id].first!=-1 and u.id!=ST){

edge etemp;

etemp.u=ST;

etemp.v=u.id;

etemp.Cnt=nd[u.id].Cnt;

etemp.Tim=nd[u.id].Tim;

etemp.Walk=nd[u.id].Walk;

etemp.next=head2[ST];

etemp.Cost=3;//固定3元

//如果这两个站是连同一个地铁口d，他们的费用设为多少。这里设无过闸费=0

if(RTP[RPP[ST].second].second==RTP[RPP[u.id].second].second){

etemp.Cost=0;

}

head2[ST]=++k2;

eTim.push\_back(etemp);

addcnt++;

}

for(int i=head[u.id];~i;i=e[i].next){

int vv=e[i].v;

node v=nd[vv];

edge w=e[i];

if(v.Tim>u.Tim+w.Tim){

v.id=vv;

v.Tim=u.Tim+w.Tim;

v.Cost=u.Cost+w.Cost;

v.Walk=u.Walk+w.Walk;

v.Cnt=u.Cnt+w.Cnt;

nd[vv]=v;

q.push(v);

}

}

}

}

}

void dijk3(){//第三题的地铁小模型跑图

set<int>::iterator it;

for(it=sNode.begin();it!=sNode.end();it++){

ST=\*it;

dijk\_init();

priority\_queue<node,vector<node>,sortWalk>q;

q.push(nd[ST]);

while(!q.empty()){

node u=q.top();q.pop();

if(vis[u.id])continue;vis[u.id]=1;

if(RPP[u.id].first!=-1 and u.id!=ST){

edge etemp;

etemp.u=ST;

etemp.v=u.id;

etemp.Cnt=nd[u.id].Cnt;

etemp.Tim=nd[u.id].Tim;

etemp.Walk=nd[u.id].Walk;

etemp.next=head3[ST];

etemp.Cost=3;//固定3元

//如果这两个站是连同一个地铁口d，他们的费用设为多少。这里设无过闸费=0

if(RTP[RPP[ST].second].second==RTP[RPP[u.id].second].second){

etemp.Cost=0;

}

head3[ST]=++k3;

eWalk.push\_back(etemp);

addcnt++;

}

for(int i=head[u.id];~i;i=e[i].next){

int vv=e[i].v;

node v=nd[vv];

edge w=e[i];

if(v.Walk>u.Walk+w.Walk){

v.id=vv;

v.Tim=u.Tim+w.Tim;

v.Cost=u.Cost+w.Cost;

v.Walk=u.Walk+w.Walk;

v.Cnt=u.Cnt+w.Cnt;

nd[vv]=v;

q.push(v);

}

}

}

}

}

stack<int>pre1;//前向点，即上一个状态

stack<int>pre2;//前向边，即乘车或者换乘

void path(int v){//路径还原

pre1.push(v);

if(v==-1 or v==ST)return;

pre2.push(prew[v]);

path(preu[v]);

}

string cplt(int num,int bit) {//to complete，补全

stringstream ss;

ss << setw(bit) << setfill('0') << num;

return ss.str();

}

void pathout(int x){

path(ED);

while(!pre1.empty()){

int u=pre1.top();pre1.pop();

pii temp1=RPP[u];

pii temp2=RTP[temp1.second];

if(temp2.second==-1){

if(u==ST)cout<<"起点:S"<<cplt(temp1.first,4)<<'\n';

else cout<<"到达终点站:S"<<cplt(temp1.first,4)<<'\n';

}

else if(temp2.first==-1){

cout<<"站点:S"<<cplt(temp1.first,4)

<<" 公交:L"<<cplt(temp2.second,3)<<'\n';

}

else{

cout<<"站点:S"<<cplt(temp1.first,4)

<<" 地铁线路:T"<<cplt(temp2.first,2)

<<" 地铁口:D"<<cplt(temp2.second,2)<<'\n';

}

if(!pre2.empty()){

int w=pre2.top();pre2.pop();

pii temp1=RPP[e[w].u];

pii temp2=RTP[temp1.second];

int u=e[w].u,v=e[w].v;

cout<<" [当前: ";

if(RTP[RPP[u].second].first==-1 and RTP[RPP[v].second].first!=-1){

cout<<"从公交转乘地铁";

}

else if(RTP[RPP[u].second].first!=-1 and RTP[RPP[v].second].first==-1){

cout<<"从地铁转乘公交";

}

else if(RTP[RPP[u].second].first==-1 and RTP[RPP[v].second].first==-1){

if(RTP[RPP[u].second].second==RTP[RPP[v].second].second){

cout<<"在公交L"<<cplt(RTP[RPP[u].second].second,3)<<"上行驶";

}

else {

if(RTP[RPP[v].second].second==-1)cout<<"下车";

else{

if(RTP[RPP[u].second].second==-1)cout<<"上汽车";

else cout<<"从公交 L"<<cplt(RTP[RPP[u].second].second,3)

<<" 换乘到 L"<<cplt(RTP[RPP[v].second].second,3);

}

}

}else{

if(RTP[RPP[u].second].second!=RTP[RPP[v].second].second){

cout<<"从地铁 T"<<cplt(RTP[RPP[u].second].first,2)

<<" 地铁口 D"<<cplt(RTP[RPP[u].second].second,3)

<<" 换乘"<<e[w].Cnt<<"次 "

<<"到地铁 T"<<cplt(RTP[RPP[v].second].first,2)

<<" 地铁口 D"<<cplt(RTP[RPP[v].second].second,3);

}

else {

cout<<"走地铁通道移动";

}

}

if(x==3)cout<<" 时间(带步行):"<<double(e[w].Walk)/2<<"min";

cout<<" 时间:"<<double(e[w].Tim)/2.0<<"min 花费:"<<e[w].Cost<<"元 换乘:"<<e[w].Cnt<<"次]\n";

// if(x==3)

// cout<<" 总时间(带步行):"<<double(nd[v].Walk)/2.0<<"min";

// cout<<" 时间:"<<double(nd[v].Tim)/2.0<<"min"

// <<" 花费:"<<nd[v].Cost<<"元"

// <<" 换乘:"<<nd[v].Cnt<<"次"<<"]\n";

}

}

}

vector<edge>eMetro;//地铁图重构的边

vector<int>headMetro;//地图重构的头边

int kMetro;

void init(){

stringstream ss;

cout<<"初始化开始"<<'\n';

k=-1;

//memset(head,-1,sizeof(head));

head.resize(maxn,-1);

tst("读入");

txt.read();

tst("公交行驶边换算中");

dealBus();

ss<<"建立了"<<addcnt<<"条公交行驶边";

tst(ss.str());ss.str("");

addcnt=0;

tst("公交换乘边换算中");

dealBus2();

ss<<"建立了"<<addcnt<<"条公交换乘边";

tst(ss.str());ss.str("");

addcnt=0;

ss<<"建立了"<<Pcnt<<"个公交虚拟点";

tst(ss.str());ss.str("");

//对于不同地铁重构图方式进行分支

/\*

将记有汽车的e传给ebus，eTim，eWalk

head传给head1，head2，head3

k也要拷贝

清空e和head来记录地铁小模型eMetro和headMetro

2.跑地铁小模型时，使用eMetro和headMetro和kMetro

跑出来边的记在eTim和head2和k2

然后将eTim和head2传给e和head

然后再往上面记载地铁与公交的换乘边

3同理

\*/

ebus=eTim=eWalk=e;

head1=head2=head3=head;

k1=k2=k3=k;

e.clear();

head.clear();

head.resize(maxn,-1);

k=-1;

tst("构建地铁小模型");

tst("正在构建地铁行驶边");

dealMetro();

tst("正在构建地铁换乘边");

dealMetro2();

eMetro=e;

headMetro=head;

kMetro=k;

ss<<"总共有"<<addcnt<<"个地铁小模型构建边";

tst(ss.str());ss.str("");

addcnt=0;

tst("第二问：以乘车时间为优先，重构地铁模型");

// e=eMetro;

// head=headMetro;

// k=kMetro;

//跑Tim记录在eTim中和head2中

dijk2();

ss<<"建立了"<<addcnt<<"个重构边";

tst(ss.str());ss.str("");

addcnt=0;

e=eTim;

head=head2;

k=k2;

dealBus\_Metro();

ss<<"建立了"<<addcnt<<"个地铁-公交换乘边";

tst(ss.str());ss.str("");

addcnt=0;

eTim=e;

head2=head;

k2=k;

tst("第三问：以乘车时间+步行时间优先，重构地铁模型");

e=eMetro;

head=headMetro;

k=kMetro;

//跑Tim记录在eWalk中

dijk3();

ss<<"建立了"<<addcnt<<"个重构边";

tst(ss.str());ss.str("");

addcnt=0;

e=eWalk;

head=head3;

k=k3;

dealBus\_Metro();

ss<<"建立了"<<addcnt<<"个地铁-公交换乘边";

tst(ss.str());ss.str("");

addcnt=0;

eWalk=e;

head3=head;

k3=k;

eMetro.clear();

headMetro.clear();

ss<<"总共有"<<Pcnt<<"个虚拟点";

tst(ss.str());ss.str("");

cout<<"初始化完成";

}

void solve(int x){

if(x==1){

e=ebus;//第一问，只处理公交站

head=head1;

k=k1;

}

else if(x==2){//第二问，地铁优先考虑乘车

e=eTim;

head=head2;

k=k2;

}

else if(x==3){//第三问，地铁优先考虑步行+乘车，由于题目所给数据较小，实际上二三问的是一样的

e=eWalk;

head=head3;

k=k3;

}

set<int>::iterator it;

ST=turn2(SP,turn(-1,-1));

ED=turn2(EP,turn(-1,-1));

//对于所有与SL相连的公交，与一个虚拟超级源点ST相连，表示这些车辆开始可上

//tst("建立超级源汇点\n");

for(it=SL[SP].begin();it!=SL[SP].end();it++){

int u=turn2(SP,turn(-1,\*it));

add(ST,u,0,0,0,0);

}

for(it=SM[SP].begin();it!=SM[SP].end();it++){//可能会存在只有地铁没有公交，我们先下地铁花费6分钟

int u=turn2(SP,\*it);

edge etemp;

etemp.Cost=etemp.Tim=0;

etemp.Cnt=1;

etemp.Tim=6\*2;

etemp.Walk=10\*2;

etemp.v=u;

add(ST,etemp);

}

//超级汇点，最终可以在这些车上下来

for(it=SL[EP].begin();it!=SL[EP].end();it++){

int v=turn2(EP,turn(-1,\*it));

add(v,ED,0,0,0,0);

}

for(it=SM[EP].begin();it!=SM[EP].end();it++){

int v=turn2(EP,\*it);

edge etemp;

etemp.Cost=etemp.Tim=0;

etemp.Cnt=1;

etemp.Tim=7\*2;

etemp.Walk=11\*2;

etemp.v=ED;

add(v,etemp);

}

bool res=0;

if(x==1 or x==2)res=dijkTim();

else res=dijkWalk();

if(!res){

cout<<"起点和终点之间不存在可行路!"<<'\n';

return;

}

if(x==1 or x==2)

cout<<"问题"<<x

<<"乘车时间最优: "<<double(nd[ED].Tim)/2.0<<"min "

<<"花费: "<<nd[ED].Cost<<"元 "

<<"转乘: "<<nd[ED].Cnt<<"次\n";

else

cout<<"问题"<<x

<<"总时间最优: "<<double(nd[ED].Walk)/2.0<<"min "

<<"乘车时间: "<<double(nd[ED].Tim)/2.0<<"min "

<<"花费: "<<nd[ED].Cost<<"元 "

<<"转乘: "<<nd[ED].Cnt<<"次\n";

pathout(x);

cout<<"\n";

if(x==1 or x==2)dijkCost();

else dijkCost2();

if(x==1 or x==2)

cout<<"问题"<<x

<<"花费最优: "<<nd[ED].Cost<<"元 "

<<"乘车时间: "<<double(nd[ED].Tim)/2.0<<"min "

<<"转乘: "<<nd[ED].Cnt<<"次\n";

else

cout<<"问题"<<x

<<"花费最优: "<<nd[ED].Cost<<"元 "

<<"总时间: "<<double(nd[ED].Walk)/2.0<<"min "

<<"乘车时间: "<<double(nd[ED].Tim)/2.0<<"min "

<<"转乘: "<<nd[ED].Cnt<<"次\n";

pathout(x);

cout<<"\n";

return;

}

void run(){

stringstream ss;

while(1){

cout<<"\n请输入起点和终点后回车(两个数字，不带S，两数用空格、回车、TAB任意方式隔开)\n";

//SP=1557,EP=481;

cin>>SP>>EP;

if(SP<=0 or EP<=0 or SP>txt.Scnt or EP>txt.Scnt){

ss<<"不满足读入数据的站点S范围：1到"<<txt.Scnt<<",将结束";

tst(ss.str());ss.str("");

break;

}

solve(1);

solve(2);

solve(3);

}

}

int main(){

// fout.open("test.txt");

init();

run();

return 0;

}