# 数据结构大作业报告

## 题目：全国交通咨询模拟

班级：1510 姓名：陈俊廷 学号：2015K8009929019 （负责程序分析器）
班级：1510 姓名：李云志 学号：2015K8009929014 （搭建框架，交互界面，最低价格查询功能，制作 ppt）
班级：1510 姓名：朱钦霖 学号：2015K8009929017 （最短时间查询功能，编写报告）

## 一、需求分析

1. 以图的邻接表为基本数据结构，两张图分别存储城市间的车次连通与航班连通信息。
2. 需要用户以标准化输入的数据：车次与航班的全体信息、出发城市、到达城市、交通方式、查询项目。
3. 测试数据见文档 Data.txt,Data_Plane.dat。

## 二、概要设计

1. 邻接表的抽象数据类型定义

ADT ALGraph{

数据对象：$\mathrm{Cityset} = \{a_i \mid a_i\text{是交通网络中包含的城市}\}$

$\mathrm{Time} = [1440]$ 出发到达时间转换为分钟数的集合

$\mathrm{Period} = \mathbb{Z}^*$ 每趟列车航班运行时间

数据关系：$\mathrm{Line} \subseteq \mathrm{Cityset} \times \mathrm{Cityset} \times \mathrm{Time} \times \mathrm{Period}$

基本操作：

`int initialGraph(ALGraph &G)`

操作结果：初始化一个邻接表

`int addCity(string city, ALGraph &G)`

操作结果：向邻接表中添加一个城市

`int delCity(string city, ALGraph &G)`

操作结果：从邻接表中删去一个城市

`int addLine(Line l, ALGraph &G)`

操作结果：向邻接表中添加一班车次或航班

`int delLine(Line l, ALGraph &G)`

操作结果：从邻接表中删除一班车次或航班

`int minimalPrice(string src, string dest, ALGraph G)`

操作结果：打印出从 src 到 dest 的最低价格路径

`int minimalTime(string src, string dest, ALGraph G)`

操作结果：打印出从 src 到 dest 的最短时间路径

}ADT ALGraph

2. 本程序包含三个模块

1）主程序模块：

```
int main(void) {
  初始化
  while(ture) {
      选择不同的操作模块或者退出 while 循环
  }
}
```

其中不同操作模块见下

2）增加城市模块 addCity

3）删除城市模块 delCity

4）增加车次航班模块 addLine

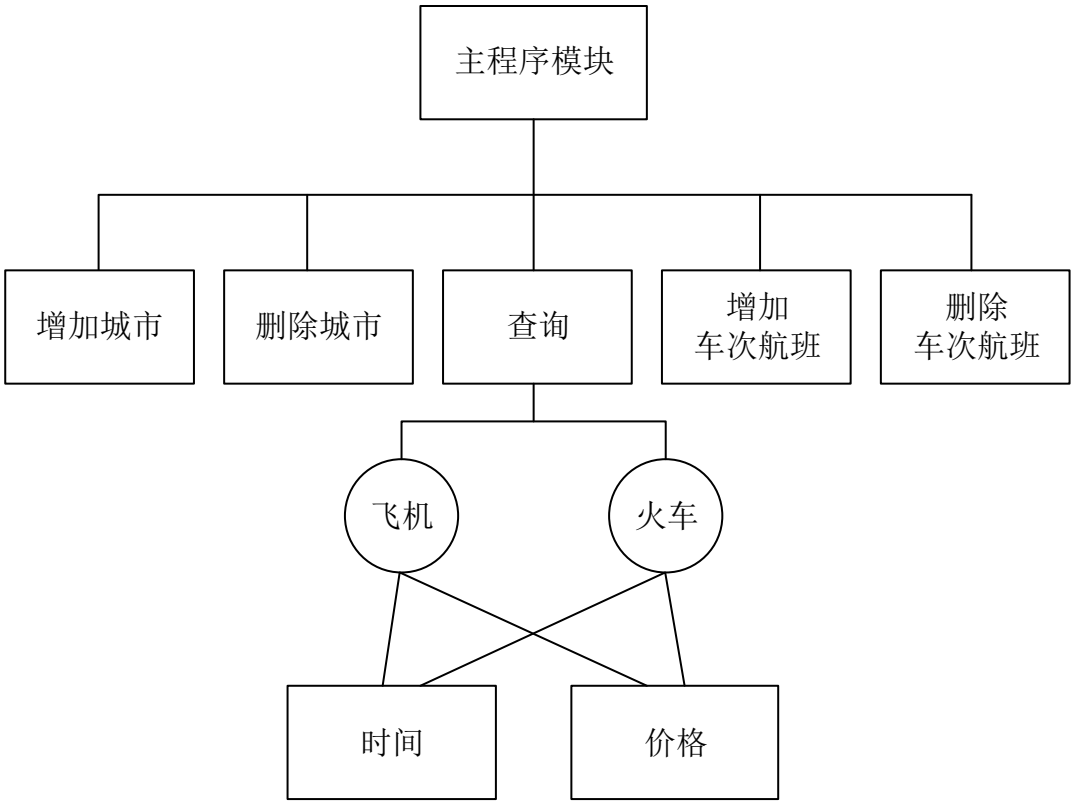
5）删除车次航班模块 delLine

6）查询模块

7）查询最低价格模块

8）查询最短时间模块

各模块之间的调用关系如下：



# 三、详细设计

1．邻接表数据类型

```
 车次信息结构
typedef struct Line {
    string Linename;//[Linelength];//列车名
    string src;// [Citylength];//起始站
    string dest;// [Citylength];//终点站
    unsigned int srcTime;//发车时间
    unsigned int destTime;//到达时间
    int period;//运行时长
    double price;//价格
}Line;
```

```
 邻接表中邻接点的结构
typedef struct adjNode {
    string adjCity;// [Citylength];//到达站名
    vector <Line> timetable;//时刻表
    struct adjNode *nextcity;
}adjNode;
```

```
 邻接表中城市列表的结构
typedef struct FirstNode {
    string City;// [Citylength];//城市名
    adjNode  *first;//邻接城市
    int status;
}FirstNode, AdjList[MAXNUMBER];
```

```
 邻接表
typedef struct {
    AdjList vertices;
    int vexnum, arcnum;
    int kind;
}ALGraph;
```

2．部分操作的算法

```
int initialGraph(ALGraph &G)//初始化图
{
    int i = 0;
    for (i = 0; i < MAXNUMBER; i++)
```

```
    {
        G.vertices[i].City = number_to_Cityname(i);
        G.vertices[i].first = NULL;
        G.vertices[i].status = 0;
    }
    G.vexnum = 0;
    G.arcnum = 0;
    G.kind = 0;
    return OK;
}
```

```
int addLine(Line l, ALGraph &G)//增加列车线路{
    int num = Cityname_to_number(l.src);
    if (G.vertices[num].first == NULL){
        G.vertices[num].first = new adjNode;
        adjNode *p = G.vertices[num].first;
        p->adjCity = l.dest;
        p->nextcity = NULL;
        p->timetable.push_back(l);
    }
    else{
        adjNode *p = G.vertices[num].first;
        while (p != NULL)
        {
            if (p->adjCity == l.dest)
            {
                p->timetable.push_back(l);
                break;
            }
            p = p->nextcity;
        }
        if (p == NULL)
        {
            p = new adjNode;
            p->adjCity = l.dest;
            p->timetable.push_back(l);
            p->nextcity = G.vertices[num].first;
            G.vertices[num].first = p;
        }
    }
    return OK;
   }
```

```
int addCity(string city, ALGraph &G)//添加城市
{
 int i = Cityname_to_number(city);
 if (G.vertices[i].status == 0)
 {
     G.vertices[i].status = 1;
     G.vexnum++;
 }
 else cout << "Has already in\n";
 return OK;
}
```

```
int delCity(string city, ALGraph &G)//删除城市
{
 int i = Cityname_to_number(city);
 if (G.vertices[i].status == 1)
 {
     G.vertices[i].status = 0;
     G.vertices[i].first = NULL;
     G.vexnum--;
     int count = 0;
     for (i = 0; i < MAXVERTEX; i++)
     {
         adjNode* p = G.vertices[i].first;
         adjNode* temp = p;
         while (temp&&temp->adjCity != city)temp = temp->nextcity;
         while (p&&p->adjCity!=city&&p->nextcity&&p->nextcity->adjCity != city)
          p = p->nextcity;
         if (p&&temp){
             if (p == G.vertices[i].first)
                 if (p->adjCity == city)
                     G.vertices[i].first = p->nextcity;
                 else
                     p->nextcity = temp->nextcity;
             else if (temp)
                 p->nextcity = temp->nextcity;
             else;
         }
     }
 }
 else cout << "Has already del\n";
 return OK;
}
```

```
int delLine(Line l, ALGraph &G)//删除线路
{
 int num = Cityname_to_number(l.src);
 adjNode *ptr = G.vertices[num].first;
 while (ptr&&ptr->adjCity != l.dest)ptr = ptr->nextcity;
 adjNode *pbefore = G.vertices[num].first;
 while (pbefore&&pbefore->adjCity!=l.dest&&pbefore->nextcity->adjCity != l.dest)
  pbefore = pbefore->nextcity;
 if (ptr == NULL)
     cout << "No such City\n";
 else
 {
     unsigned int i = 0;
     for (i = 0; i < ptr->timetable.size(); i++)
         if（两个 Line 相同）
         {
             ptr->timetable.erase(ptr->timetable.begin() + i);
             cout << "Delete complete\n";
             if (ptr->timetable.size() == 0)
             {
                 pbefore->nextcity = ptr->nextcity;
             }
             break;
         }
         else if (i == ptr->timetable.size() - 1)
             cout << "No such Line\n";
         else;
 }
 return 0;
}
```

3. 部分伪码的算法

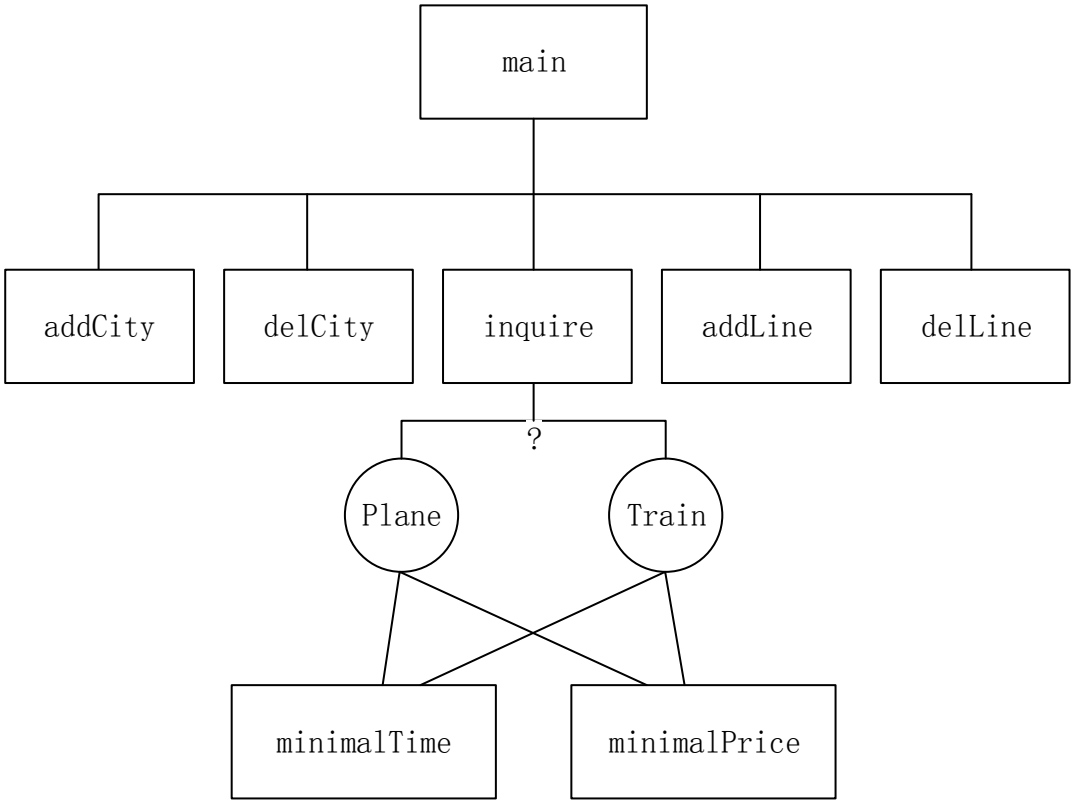
```
MinimalPrice() {   //minimalPrice 的伪码算法
  初始化 dist 数组；
  初始化 before 数组；//用于向前寻找节点，形成路径
  初始化 beforeLine 数组；//用于存储最低价格线路的各项信息
  初始化 visited 数组；//用于判断是否已经访问过
  int curr = root；
  while（true）
  {
    FirstNode f = G.vertices[curr];
    adjNode *p = f.first;//找到当前新加入节点
    while（p）
    {
      找到当前两个城市间的对低价格线路；
      保存线路信息到 tempLine 中；
      if（未访问过当前节点&& dist[]<dist[curr]+minimalPrice）
        更新距离；
       p = p->nextcity;
    }
    找到未访问过且距离最近的新节点 j；
    curr=j
    若全部访问过则 break；
  }
```

```
minimalTime() {//minimalTime 的伪码算法
    for(从 src 出发的可能的第一班车次，该车到达 city1) {
        创建辅助数组；
        src 和 city1 加入集合 S;
        while(S 集合仍可以扩展且 dest 还未加入 S) {

            对每一对v,u，其中v ∈ S 且 v ≠ s 并 u 与 v 邻接，找到 v 到 u 的最早到达

            的车次并记录在辅助数组中城市 u 对应的项中；
            比较刚刚记录的 u 中到达时间最早的，加入集合 S；
        }
    }
    对于上面不同数组描述的不同的到达 dest 的方式，找出其中耗时最短的，输出；
}   //minimalTime 的伪码算法
```

4. 函数的调用关系反映了演示程序层次结构




## 四、调试分析

程序设计过程中遇到的问题：

1、我们需要增加、删除城市的功能以及增加、删除时刻表的功能，该功能与链表的增加删除联系紧密，而我们也在开发过程中遇到了此方面许多的问题，如指针运用不当、逻辑错误等问题，也导致了程序多次运行失败。但此问题最终得到了很好的解决。

难点：该程序的大部分功能都可以基于 Dijkstras 算法来实现，但是查询最短时间的算法中，由于对不同的路径选择，出发时间也会变化，不像路径问题，出发时走过的距离永远是 0。所以程序对不同的出发时间要分类讨论。在这种分类讨论中也会造成一些重复计算，排除这种重复计算也需要一定的手段。

优点：1.数据中火车车次为 3502，飞机航班数为 1064，数据量较大，程序完成效果正常

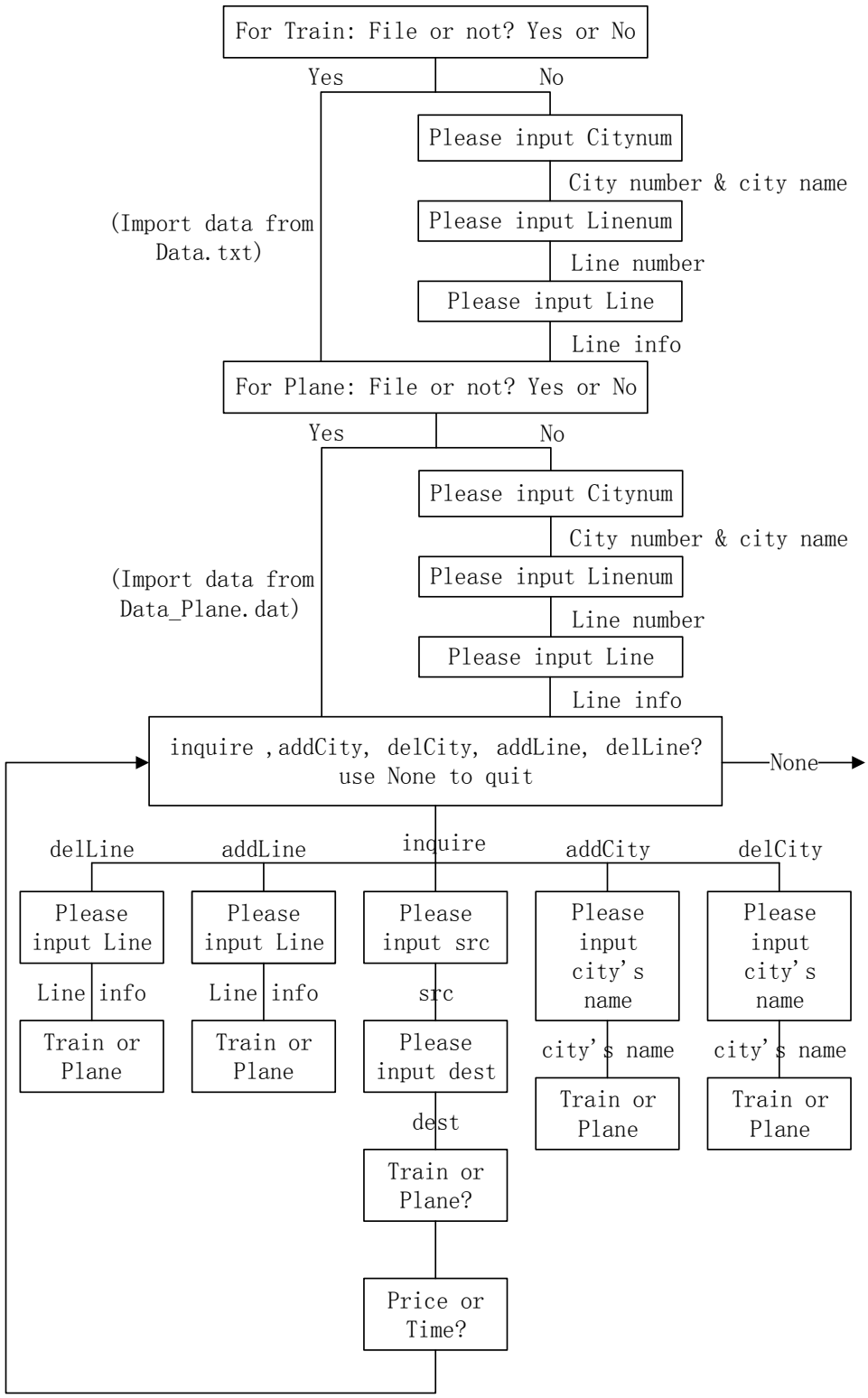
2.交互过程较为简单易懂，对于未操作过程序的人来说也可以较快的上手

缺点：1. 控制台用户界面，基于命令行的交互方式，给操作带来了一定的不便，对于习惯了图形界面的用户来说，不十分友好。

2.对于增加和删除线路这一功能要求车次信息精确，对于录入信息有一定的难度

# 五、用户手册

1. 本程序的运行环境为 window 10 操作系统，执行文件为：Route_Inquire.exe。
2. 进入演示程序后，即显示命令行方式的用户界面。
3. 根据屏幕上出现的提示信息输入：



a．提示信息：For Train: File or not? Yes or No
输入 Yes 并按回车则从程序所处的路径处的 Data.txt 导入车次信息
输入 No 并按回车则在控制台手动输入车次信息

b．提示信息：Please input Citynum
该提示信息出现在用户选择手动输入车次或航班信息的情况下，输入一个正整数，表示交通网络中涉及的城市数目，并继续依次输入这些城市的名字

c．提示信息：Please input Linenum
输入一个正整数，表示交通网络中的车次或航班数目

d．提示信息：Please input Line
按照格式输入一班车或航班的详细信息，格式为：
车次名 发车站 到达站 发车时间 到达时间 经历时间 价格
例如 T296 WAR LZJ 14:34 10:22 19:48 217.0

e．提示信息：inquire ,addCity, delCity, addLine, delLine?
use None to quit
选择接下来要进行的操作，或者退出

f．提示信息：Please input src(dest)
输入查询起点(终点)城市名

g．提示信息：Train or Plane? Use 'None' to quit
输入交通方式，或选择退出

h．提示信息：Price or Time?
输入查询项目，是最低价格或是最短时间

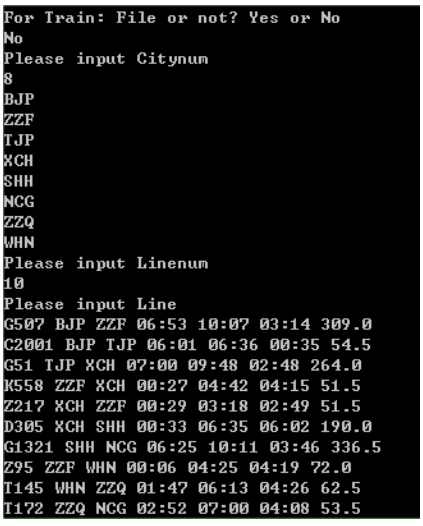
i．提示信息：Please input city's name
输入城市名字

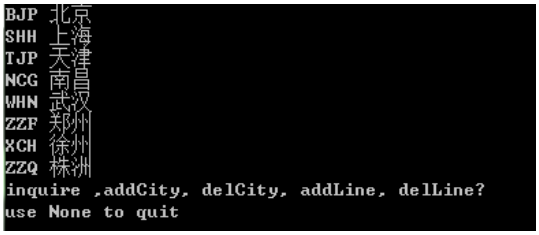
4. 信息输入完毕后，会依次输出交通网络中涉及的城市以及其代号
5. 查询结果输出在屏幕上，倒序依次指示用户应该乘坐的车次航班。

# 六、测试结果

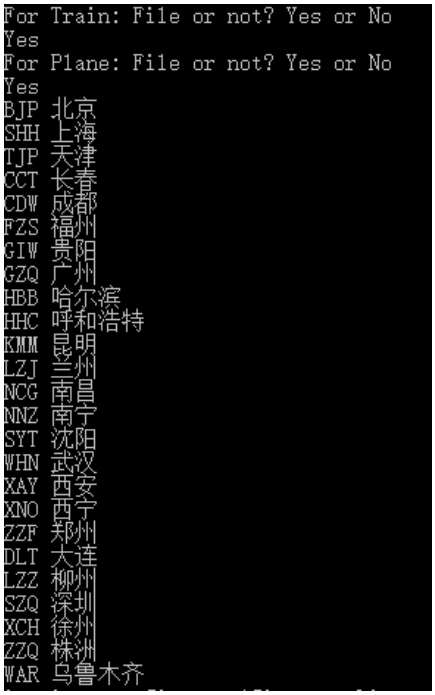
三组测试数据和输出结果如下：

1. 从 Data.txt 读入交通网络信息，导入成功后显示城市列表
(1)手动输入

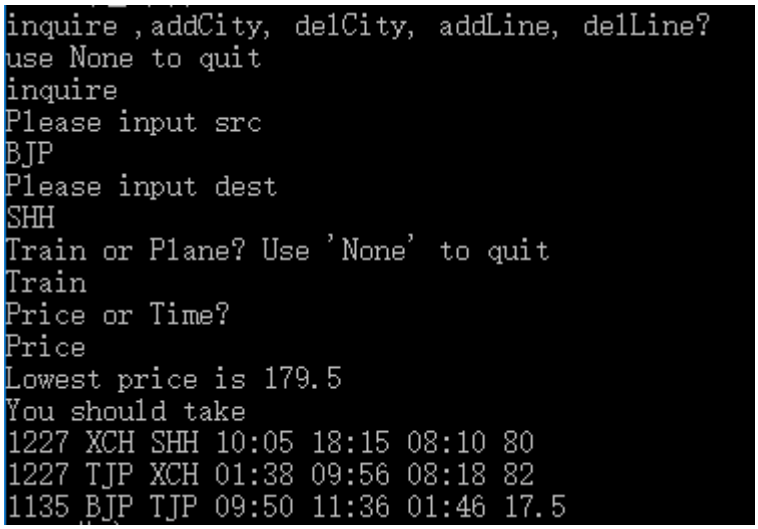





(2)文件录入




2．查询北京到上海的最低价格路径



3．查询北京到上海的最短时间路径

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
inquire
Please input src
BJP
Please input dest
SHH
Train or Plane? Use 'None' to quit
Train
Price or Time?
Time
You should take
This trip takes 05:14
G21 XCH SHH 18:47 21:14 02:27 279
G21 TJP XCH 16:36 18:44 02:08 259
G21 BJP TJP 16:00 16:34 00:34 54.5
```

4．删除上面最低价格路径中用到的一班车次

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
delLine
Please input Line
1227 TJP XCH 01:38 09:56 08:18 82
Train or Plane
Train
Delete complete
```

5．再次查询北京到上海的最低价格路径

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
inquire
Please input src
BJP
Please input dest
SHH
Train or Plane? Use 'None' to quit
Train
Price or Time?
Price
Lowest price is 188.5
You should take
1227 XCH SHH 10:05 18:15 08:10 80
T31 TJP XCH 18:21 00:33 06:12 91
1135 BJP TJP 09:50 11:36 01:46 17.5
```

6．删除上面最短时间路径中用到的一班车次

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
delLine
Please input Line
G21 BJP TJP 16:00 16:34 00:34 54.5
Train or Plane
Train
Delete complete
```

7．再次查询北京到上海的最短时间路径

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
inquire
Please input src
BJP
Please input dest
SHH
Train or Plane? Use 'None' to quit
Train
Price or Time?
Time
You should take
This trip takes 05:37
G113 XCH SHH 11:53 14:30 02:37 279
G113 TJP XCH 09:34 11:51 02:17 259
G113 BJP TJP 08:53 09:27 00:34 54.5
```

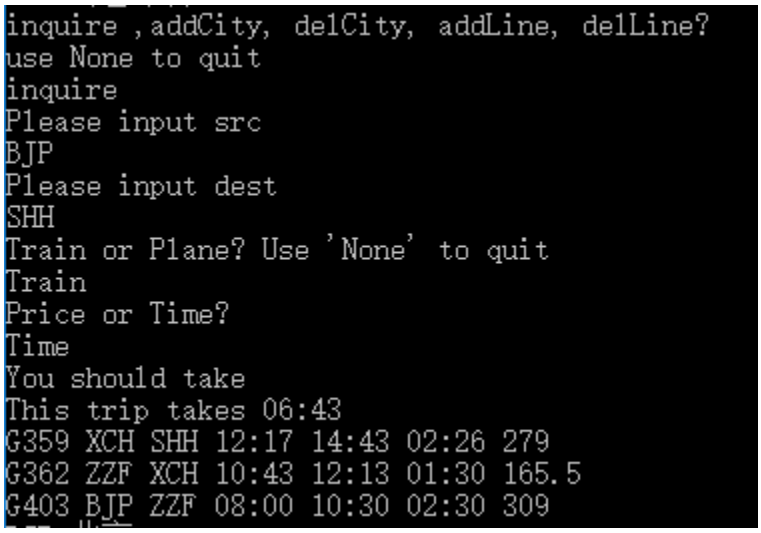
8．删除上面最低价格和最短路径用到的一个城市

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
delCity
Please input city's name
TJP
Train or Plane
Train
```

9．再次查询北京到上海的最低价格路径

```
inquire ,addCity, delCity, addLine, delLine?
use None to quit
inquire
Please input src
BJP
Please input dest
SHH
Train or Plane? Use 'None' to quit
Train
Price or Time?
Price
Lowest price is 218.5
You should take
1227 XCH SHH 10:05 18:15 08:10 80
1148 ZZF XCH 02:14 06:53 04:39 45.5
K599 BJP ZZF 05:14 14:04 08:50 93
```

10. 再次查询北京到上海的最短时间路径

11. 


# 七、附录

源程序文件名清单：

Route_Inquire.cpp