

# 交通查询系统项目报告

## 一、需求分析：

大致分为五个模块：主模块、城市信息、打印表、读写文件、路线咨询。

### 1、主模块：

即 main 函数部分，负责反复读入需求，并调用函数。

### 2、城市信息：

即 city\_info 部分，负责对城市及路线进行管理。包含城市及班次的类或结构定义，以及一系列编辑（增加或删除）城市或班次的函数。

### 3、打印表：

即 print\_table 部分，负责在“打印时刻表”功能中进行输出，主要是对输出格式进行一些调整。

### 4、读写文件：

即 rw\_txt 部分，负责读写文件。

### 5、路线咨询：

即 which\_way 部分，负责读入需求并调用函数，计算用时最少、花费最少或中转最少的三种路径。

## 二、概要设计：

### 1、main：

（1）美观起见，为欢迎语增加星号方框。

```
1. string board (18, '*');
2. cout<<board<<endl<<"*欢迎进入交通咨询*"<<endl<<board<<endl;
```

（2）容错设计：

```
1. cout<<endl<<"0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询"<<endl
2. <<"请输入数字以选择相应功能"<<endl;
3. string func;
4. getline(cin, func);
5. char f=func[0];
6. if(!isdigit(f)){
7.     cout<<"需要输入一个数字！再试试看。"<<endl;
8.     continue;
9. }
```

读入整行，避免输入过多而未能完全读入，使得缓冲区中的内容对后续读入产生影响；用输入的第一个字符来决定功能。

三大功能：

```
1. cout<<endl<<"0、退出： 1、编辑信息： 2、打印时刻表： 3、路线咨询"<<endl
```

## 2、city\_info:

### (1) 城市及班次信息:

数据结构为有向图，城市为顶点，若城市 A 出发，有火车或飞机能到达城市 B，则存在弧(A,B)。可能有多重边，因为从 A 到 B 很可能不知一个班次。以出边邻接表的形式存储在 vector 中。

city\_info 头文件中定义两个类或结构，info 及 city\_node。每个城市有自己的 info 类，私有定义了城市名称和一存储该城市出发的班次的 vector，后一 vector 的成员即为 city\_node，包含目的地城市名、交通工具 (train or aeroplane)、花费、出发时间、到达时间。其中，时间以 int[2] 数组的形式存储时和分，便于后续对时间进行计算，同时也在一定程度上节省空间。

### (2) 管理:

用四个函数 add\_city、delete\_city、add\_f\_or\_t、delete\_f\_or\_t 来实现增删城市和增删班次的功能，考虑到增加班次有可能会同时增加城市，遂拆解该部分至 add\_city\_node 函数中。

①int add\_city\_node(vector<info>& c, string& name):

遍历 c，若找到 name 对应城市则返回 1，若没有找到，则将其加入 c 中，并返回 0。

②void add\_city(vector<info>& c):

输出提示性语句并读入 name，判断 name 的合法性:

```
1. if(name.size()>=20||name.size()==0){
2.     cout<<"请输入长度合适的城市名，再试试看！"<<endl;
3.     continue;
4. }
```

调用 add\_city\_node 函数，提示“该城市已存在”或表明“添加成功”。

③void delete\_city(vector<info>& c):

遍历并删除城市 info，同时删除其他城市到达该城市的班次，需要两个循环，O(n+e)。

④void add\_f\_or\_t(vector<info>& c):

增加班列，主要在于输入合法性的判断上。对于输入时间。

```
1. cout<<"按照 05:13 的格式输入时间,注意冒号使用英文符号."<<endl;
2.     string dt,at;
3.     int d0,d1,a0,a1;
4.     while(1){
5.         cout<<"出发时间: ";
6.         getline(cin,dt);
7.         if(dt.size()==5){
8.             d0=(dt[0]-'0')*10+(dt[1]-'0');
9.             d1=(dt[3]-'0')*10+(dt[4]-'0');
10.            if(d0<24&&d0>=0&&d1<60&&d1>=0) break;
11.        }
12.        cout<<"格式不规范，再试试！"<<endl;
13. }
```

读入一整行，若字符串长度为 5，再分别计算时、分，并判断时、分是否合理，即时应在 0-23 的整数中，分应在 0-59 的整数中，如不合规，则输出提示性语句再次录入。

先遍历匹配出发地名称，如找到，则在其内部 dest 中添加新班次，如无，则新建城市 info 并添加，再添加班次。最后，对目的地调用 add\_city\_node(c, dest)，不存在则添加。

⑤void delete\_f\_or\_t(vector<info>& c):

如果采用和添加一样的方法，录入需要删除的班次的信息，再进行匹配，工作量无疑是较大的，因而修改功能 2 打印时刻表部分，在打印时输出序号，那么，这里便由用户输入想要删除的班次的序号，从而进行删除。

```
1. stringstream ss;
2. string line;
3. getline(cin, line);
4. ss<<line;
5. int c0, c1;
6. ss>>c0>>c1;
7. if(c0>=c.size()||c1>=c[c0].dest.size()){
8.     cout<<"该序号不存在，再试试！"<<endl;
9.     continue;
10. }
11. c[c0].delete_node(c1);
```

A. delete\_node 函数为类内部定义，因采用输入相应序号进行删除的方式，而序号对应了其在 vector 内存储的下标，故删除较简单。

B. 输入的容错机制：同样地，读入一整行，在这里运用了字符串流，从读入的一整行中拿出两个 int 类型的序号 c0 和 c1，再行判断序号的合理性，即小于 vector 的大小。

⑥void edit\_citys(vector<info>& c):

功能引导部分，由 main 函数调用，可调用以上函数，包含类似 main 部分的容错机制。

### 3、print\_table:

该部分函数在头文件 city\_info.h 中声明，实现功能 2，打印时刻表，两层循环，O(n+e)，主要有格式问题和时间输出的问题。

(1) 格式：包含<iomanip>头文件，默认右对齐，用 setw 函数保证基本对齐。

(2) 时间输出：void print\_time(int a[])，如前所述，时间存储在 int[2] 数组中，a[0] 存储时，a[1] 存储分，此处用 print\_time 函数进行输出，同时对不足 10 的时或分补 0，如九点四十五，输出为“09:45”，以达到美观。

### 4、rw\_txt:

包含读、写文件两大部分。

(1) 从文件中读取信息并生成邻接表表示的有向图：

①void read\_from\_file(vector<info>& c):

创建文件流并打开文件 data.txt，读入一行至字符串 line，利用 read\_line 函数对 line 进行拆解并新建班次。

②string read\_line(city\_node\* t, string& line, int \*flag):

有两种情形，一是完整的班次信息，二是仅有城市名，因为没有从此出发的班次，需要在处理时做出区别。

```

1. string name;
2. int i=-1,j=0;
3. char temp[20];
4. while(line[++i]!=' ' && line[i]!='\0') temp[j++]=line[i];
5. temp[j]=0;
6. name=temp;
7. if(line[i]=='\0'){
8.     *flag=1; //flag 即是用来辨别是哪种情况的
9.     return name;
10. }

```

若为情形一，则接着往后读取内容，方法类似，详见 cpp 文件。

(2) 写文件以更新城市和班次信息

当增删城市和班次之后，同样地也应该重写文件。

`void rewrite(vector<info>& c):`

创建文件流并打开文件 data.txt，先用 clear 函数清空文件，然后重写，类似 print\_table 部分，只是将标准输出流换为文件流，且无须关注格式对齐，只需用空格间隔即可。print\_time 部分同样被拆出。

## 5、which\_way:

三种路径的核心算法都是用来求最小权路的 Dijkstra 算法，但需要一些改造，以适应不同的需求：

- ①关键在于权的不同，遂用 all\_cost 函数计算不同模式下的权值；
- ②考虑多重边的情况，增加 kth 数组记录是第几条弧，同时也因为多重边的情形，all\_cost 函数也需要再行改造；
- ③因时间的合理性，需要记录某城市的到达时间，以便之后以该城市为起点时，能选择时间合理的班次；
- ④需要对比交通工具，故将确定的交通工具字符串 vehi 以参数形式传入，如果不匹配则视作没有弧相连。

以下为做了修改的 Dijkstra 算法，说明以注释形式放在之后：

```

1. void short_path_dijkstra(vector<info>& c,int v,int mode,string& vehi,int kth
   [],int pre[]){
2.     int n=c.size();
3.     int k;
4.     int* kt[50]; //到达该城市的时间
5.     float min;
6.     float dist[50],s[50];
7.     for(int i=0;i<n;i++){
8.         if(i==v) continue;
9.         dist[i]=all_cost(mode,c,v,i,0,kth+i,kt+i,vehi);
10.        s[i]=0;
11.        if(dist[i]<MAX) pre[i]=v;
12.        else pre[i]=-1;
13.    }

```

```

14.     s[v]=1;pre[v]=-1;
15.     for(int i=0;i<n;i++){
16.         min=MAX;
17.         k=-1;
18.         for(int j=0;j<n;j++){
19.             if(j==v) continue;
20.             if(s[j]==0&&dist[j]<min){
21.                 min=dist[j];
22.                 k=j;
23.             }
24.         }
25.         if(k==-1) break;
26.         s[k]=1;
27.         for(int j=0;j<n;j++){
28.             if(s[j]==1) continue;
29.             int j_kth;
30.             int* j_kt;
31.             j_kth=kth[j];
32.             j_kt=kt[j];//all_cost 函数会对这两个值进行修改，故先保存
33.             int cost=all_cost(mode,c,k,j,kt[k],kth+j,kt+j,vehi);
34.             if(cost<MAX&&dist[k]+cost<dist[j]){
35.                 dist[j]=dist[k]+cost;
36.                 pre[j]=k;
37.             }else{
38.                 kth[j]=j_kth;
39.                 kt[j]=j_kt;//不应修改，遂还原
40.             }
41.         }
42.     }
43. }

```

对于 Dijkstra 算法，时间复杂度为  $O(n+e)$ 。

all\_cost 函数及相应说明如下所示：

```

1.  float all_cost(int d,vector<info>& c,int pre,int cur,int ar_time[],int* k,i
    nt** kt,string& vehi){
2.  const vector<city_node*>& t=c[pre].destination();
3.  //mode 0 最少中转; mode 1 最少花费; mode 2 最短用时
4.  vector<city_node*>::const_iterator i=t.begin();
5.  float min_cost=MAX;
6.  int min_time=MAX;
7.  int counter=0;
8.  *k=-1;//*k 用来记录是第几条弧，以应对多重边
9.  for(;i!=t.end();i++,counter++){

```

```

10.     if(ar_time==0||(*i)->de_time[0]>ar_time[0]||((*i)->de_time[0]==ar_time[0]
    ]&&(*i)->de_time[1]>ar_time[1]));
11.     else continue;
12.     //确认时间合理, 要么 ar_time=0, 代表第一班次, 要么后出发时间晚于前到达时间
13.     if((*i)->dest_city==c[cur].name()&&(*i)->vehicle==vehi){
14.         if(d==0){
15.             *k=counter;
16.             *kt=(*i)->ar_time;
17.             return 1;
18. //最少中转找到一条弧即可, 不用继续检索, 返回权为 1, 那么显然中转越少, 总权值越低
19.         }
20.         if(d==1&&(*i)->cost<min_cost){
21.             min_cost=(*i)->cost;
22.             *k=counter;
23.             *kt=(*i)->ar_time;//记录通过该班次到达该地的时间, 以备之后使用
24.         }//遍历以找到多重边当中最小的那一边
25.         if(d==2){
26.             int temp;
27.             if(ar_time==0) temp=calculate_time((*i)->de_time,(*i)->ar_time);
28.             else temp=calculate_time(ar_time,(*i)->ar_time);
29.             //用时分两种情况, 第一个班次即用自己的到达时间'减'出发时间
30.             //之后的班次用自己的到达时间'减'到达出发地的时间
31.             //'减'被拆解出来, 放到 calculate_time 函数里
32.             if(temp<min_time){
33.                 *kt=(*i)->ar_time;
34.                 min_time=temp;
35.                 *k=counter;
36.             }
37.         }//fast way
38.     }
39. }
40. if(d==0) return MAX;
41. if(d==1){
42.     if(*k==-1) return MAX;
43.     else return min_cost;
44. }
45. if(d==2){
46.     if(*k==-1) return MAX;
47.     else return min_time;
48. }

```

以上两个函数为核心所在, 遂完整展示, 其余还有函数:

```

void fast_way(vector<info>& c, int d, int a, string& v)
void short_way(vector<info>& c, int d, int a, string& v)

```

```
void cheap_way(vector<info>& c, int d, int a, string& v)
```

这三个函数除调用 short\_path\_dijkstra 函数时的 mode 值不同外，一模一样，不再赘述。在判断有无可达班次及输出班次、总用时、总花费、总中转站上，调用函数

```
void show_way(vector<info>& c, int d, int a, int kth[], int pre[])
```

因 pre 数组需倒序查看，故在该函数中运用数据结构，栈，从而得到正序结果，并在此过程中计算用时、花费和中转站。考虑多重边情形，故压入栈的为自定义的结构 print\_node，如下所示：

```
1. struct print_node{
2.     int city; //序号记录的城市
3.     int kth; //该城市出发的班次中的第 k 条
4. ;
```

show\_way 函数中的打印部分类似于 print\_table 部分，且调用了那一部分的 print\_time 函数，不再赘述。

### 三、详细设计：

如所附源文件所示，此处不作展示。

### 四、调试分析：

#### 1、初始测试数据：

```
Chongqing Guangzhou train 210 13:10 18:01
Chongqing Sichuan train 210 09:10 12:01
Guangzhou Hubei train 300 01:12 05:30
Guangzhou Hubei train 400 19:12 21:30
Sichuan Hubei train 720 13:00 15:00
Sichuan Beijing aeroplane 1000 13:00 15:00
Beijing Chongqing aeroplane 800 15:12 16:40
Sichuan Chongqing aeroplane 1000 08:44 14:00
Hubei
```

如所附文件 data.txt。

设置了多条线路，且有多重边和孤立城市的存在，便于测试各功能是否正确执行。

#### 2、功能测试：

##### (1) 打印时刻表：

```
*****
*欢迎进入交通咨询*
*****

0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询
请输入数字以选择相应功能
2

Time Table
0 0      Chongqing      Guangzhou      train      210      13:10      18:01
0 1      Chongqing      Sichuan        train      210      09:10      12:01
1 0      Guangzhou      Hubei          train      300      01:12      05:30
1 1      Guangzhou      Hubei          train      400      19:12      21:30
2 0      Sichuan        Hubei          train      720      13:00      15:00
2 1      Sichuan        Beijing        aeroplane   1000     13:00      15:00
2 2      Sichuan        Chongqing      aeroplane   1000     08:44      14:00
3 0      Beijing        Chongqing      aeroplane   800      15:12      16:40
4 0      Hubei          暂无航班或火车车次从这里出发
```

## （2）增删城市：

### ①增加城市：

```
0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询
请输入数字以选择相应功能
1

您希望做些什么？
0、增加城市； 1、删除城市； 2、增加班次； 3、删除班次； 4、回到上一步
请输入数字以选择相应功能
0

请输入城市名（不超过20个字符）： Hangzhou
添加成功！

0、增加城市； 1、删除城市； 2、增加班次； 3、删除班次； 4、回到上一步
请输入数字以选择相应功能
4

0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询
请输入数字以选择相应功能
2

Time Table
0 0      Chongqing      Guangzhou      train      210      13:10      18:01
0 1      Chongqing      Sichuan        train      210      09:10      12:01
1 0      Guangzhou      Hubei          train      300      01:12      05:30
1 1      Guangzhou      Hubei          train      400      19:12      21:30
2 0      Sichuan        Hubei          train      720      13:00      15:00
2 1      Sichuan        Beijing        aeroplane   1000     13:00      15:00
2 2      Sichuan        Chongqing      aeroplane   1000     08:44      14:00
3 0      Beijing        Chongqing      aeroplane   800      15:12      16:40
4 0      Hubei          暂无航班或火车车次从这里出发
5 0      Hangzhou
```

可以看到，不论是输出，还是右侧的 data 文件中，均出现了新增的城市 Hangzhou。输入城市名不超过 20 个字符，增加相应机制检测，如长度不合理，则重输。

### ②删除城市：

为了更好的展示相应功能，选择删除同时也在班次目的地的城市，Hubei，效果如下。

```
0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询
请输入数字以选择相应功能
1

您希望做些什么？
0、增加城市； 1、删除城市； 2、增加班次； 3、删除班次； 4、回到上一步
请输入数字以选择相应功能
1

请输入城市名（不超过20个字符）： Hubei
删除成功！

0、增加城市； 1、删除城市； 2、增加班次； 3、删除班次； 4、回到上一步
请输入数字以选择相应功能
4

0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询
请输入数字以选择相应功能
2

Time Table
0 0      Chongqing      Guangzhou      train      210      13:10      18:01
0 1      Chongqing      Sichuan        train      210      09:10      12:01
1 0      Guangzhou      暂无航班或火车车次从这里出发
2 0      Sichuan        Beijing        aeroplane   1000     13:00      15:00
2 1      Sichuan        Chongqing      aeroplane   1000     08:44      14:00
3 0      Beijing        Chongqing      aeroplane   800      15:12      16:40
4 0      Hangzhou      暂无航班或火车车次从这里出发
```

在输出和右侧的 data 文件中，Hubei 被删除，以 Hubei 为目的地的班次也被删除。有一定容错机制，对城市名长度做出限制，并会检索反馈是否存在该城市。

### ③增加班次：

在前方基础上，预备增加：

Hubei Nanjing aeroplane 320.5 13:01 15:00



Time Table						
0 0	Chongqing	Guangzhou	train	210	13:10	18:01
0 1	Chongqing	Sichuan	train	210	09:10	12:01
1 0	Guangzhou	暂无航班或火车车次从这里出发				
2 0	Sichuan	Beijing	aeroplane	1000	13:00	15:00
2 1	Sichuan	Chongqing	aeroplane	1000	08:44	14:00
3 0	Beijing	Chongqing	aeroplane	800	15:12	16:40
4 0	Hangzhou	暂无航班或火车车次从这里出发				
5 0	Hubei	Nanjing	aeroplane	320.5	13:01	15:00
6 0	Nanjing	暂无航班或火车车次从这里出发				

0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询  
请输入数字以选择相应功能

Chongqing Guangzhou train 210 13:10 18:01  
Chongqing Sichuan train 210 09:10 12:01  
Guangzhou  
Sichuan Beijing aeroplane 1000 13:00 15:00  
Sichuan Chongqing aeroplane 1000 08:44 14:00  
Beijing Chongqing aeroplane 800 15:12 16:40  
Hangzhou  
Hubei Nanjing aeroplane 320.5 13:01 15:00  
Nanjiing

在输出和右侧的 data 文件中，成功增加班次 5 0，同时因目的地 Nanjing 未出现过，也增加为城市，见 6 0。该班次中各数据为一个一个在提示性语句后输入，如果输入不合理，一定程度上能够察觉并要求重输，具体见概述部分或源文件。

#### ④删除班次：

在上方基础上，可以先尝试输入非法序号 3 1，再输入 2 0，删除该班次，结果如下。

请输入时间表中的对应序号以删除该班次，以空格间隔  
3 1  
该序号不存在，再试试！  
2 0  
删除成功！

0、增加城市； 1、删除城市； 2、增加班次； 3、删除班次； 4、回到上一步  
请输入数字以选择相应功能  
4

0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询  
请输入数字以选择相应功能  
2

Time Table						
0 0	Chongqing	Guangzhou	train	210	13:10	18:01
0 1	Chongqing	Sichuan	train	210	09:10	12:01
1 0	Guangzhou	暂无航班或火车车次从这里出发				
2 0	Sichuan	Chongqing	aeroplane	1000	08:44	14:00
3 0	Beijing	Chongqing	aeroplane	800	15:12	16:40
4 0	Hangzhou	暂无航班或火车车次从这里出发				
5 0	Hubei	Nanjing	aeroplane	320.5	13:01	15:00
6 0	Nanjing	暂无航班或火车车次从这里出发				

0、退出； 1、编辑信息； 2、打印时刻表； 3、路线咨询  
请输入数字以选择相应功能

data - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Chongqing Guangzhou train 210 13:10 18:01  
Chongqing Sichuan train 210 09:10 12:01  
Guangzhou  
Sichuan Chongqing aeroplane 1000 08:44 14:00  
Beijing Chongqing aeroplane 800 15:12 16:40  
Hangzhou  
Hubei Nanjing aeroplane 320.5 13:01 15:00  
Nanjing

红线 1 部分展示输入 3 1 时，输出“该序号不存在”信息，并要求重输，重输 2 0 后，在输出的时刻表的 data 文件中，原 2 0 班次被删除，原 2 1 班次变为了 2 0 班次。

#### （3）路线咨询：

**注意：**此处采用的数据仍为 1 中所示的初始测试数据！如下所示。

Time Table						
0 0	Chongqing	Guangzhou	train	210	13:10	18:01
0 1	Chongqing	Sichuan	train	210	09:10	12:01
1 0	Guangzhou	Hubei	train	300	01:12	05:30
1 1	Guangzhou	Hubei	train	400	19:12	21:30
2 0	Sichuan	Hubei	train	720	13:00	15:00
2 1	Sichuan	Beijing	aeroplane	1000	13:00	15:00
2 2	Sichuan	Chongqing	aeroplane	1000	08:44	14:00
3 0	Beijing	Chongqing	aeroplane	800	15:12	16:40
4 0	Hubei	暂无航班或火车车次从这里出发				

测试：

##### ①Chongqing to Hubei, train

有两条路线：0 0->1 1，花费 610 元，耗时 500min，中转 1 站

0 1->2 0，花费 930 元，耗时 350min，中转 1 站

出发地：Chongqing  
目的地：Hubei  
选择交通工具：可供选择的交通方式只有火车和飞机，输入英文，train or aeroplane。  
选择交通工具：train

0、最快； 1、最少中转； 2、最便宜； 3、回到上一步  
请输入数字以选择相应功能  
0

Chongqing	Sichuan	train	210	09:10	12:01
Sichuan	Hubei	train	720	13:00	15:00

总计用时：350分钟 花费：930元 中转站数：1站

0、最快； 1、最少中转； 2、最便宜； 3、回到上一步  
请输入数字以选择相应功能  
2

Chongqing	Guangzhou	train	210	13:10	18:01
Guangzhou	Hubei	train	400	19:12	21:30

总计用时：500分钟 花费：610元 中转站数：1站

成功选择合适线路。

②Sichuan to Chongqing, aeroplane

有两条线路：2 1->3 0，花费 1800 元，耗时 220min，中转 1 站

2 2，花费 1000 元，耗时 316min，中转 0 站

```
出发地: Sichuan
目的地: Chongqing
选择交通工具: 可供选择的交通方式只有火车和飞机, 输入英文, train or aeroplane.
选择交通工具: aeroplane

0、最快; 1、最少中转; 2、最便宜; 3、回到上一步
请输入数字以选择相应功能
1
Sichuan Chongqing      aeroplane      1000      08:44      14:00
总计用时: 316分钟    花费: 1000元    中转站数: 0站

0、最快; 1、最少中转; 2、最便宜; 3、回到上一步
请输入数字以选择相应功能
2
Sichuan Chongqing      aeroplane      1000      08:44      14:00
总计用时: 316分钟    花费: 1000元    中转站数: 0站

0、最快; 1、最少中转; 2、最便宜; 3、回到上一步
请输入数字以选择相应功能
0
Sichuan Beijing aeroplane      1000      13:00      15:00
Beijing Chongqing      aeroplane      800      15:12      16:40
总计用时: 220分钟    花费: 1800元    中转站数: 1站
```

成功选择合适路线。

(4) 过程中遭遇的问题及分析解决:

①打印时刻表:

最开始使用 ' \t ' 作为间隔, 但发现因为各部分长短不一, 所以不太对得齐, 后改用 <iomanip>头文件中的 setw 函数, 并右对齐。

左侧的两个序号以空格间隔, 即为后续删除班次阶段所用。事实上最开始预备使用一个序号以表示, 但发现两部分很可能位数不一, 难以划分, 如 121, 难确定是 1 城市的 21 班次, 还是 12 城市的 1 班次, 存在歧义, 遂改为间隔的两部分。

②增删城市:

事实上, 在开始着手此部分之前, 并没有将没有出边的点也放在 vector<info>中, 即只存储了有从此地出发的班次的城市的信息。但既然能够单独地增加城市, 说明城市信息应当更加独立, 而不是依赖于弧, 所以对 data.txt 数据进行修改, 并且修改了读文件的相应逻辑。

在删除城市中, 遍历加 vector 的 erase 函数, 可以较容易做到删除。在访问 info 类成员, 去删除具体含相应城市作目的地的班次时, 关于类的私密性, 出了些问题, 原先以便输出的类函数 `const vector<city_node*>& destination() const { return dest; }` 当然不再适用, 因为无法修改相应内容。考虑到将删除城市作为类函数不太方便, 遂将其声明为 info 类的友函数, 解决问题。

③增删班次:

增加班次, 在容错性上做了一些考虑, 具体如源文件所示。

删除班次, 最开始考虑如同增加班次一般, 依次输入对应班次的相应内容, 比对, 再删除。但这过于麻烦。直觉上来讲, 让用户直接选择删除哪一趟班次肯定是较好的方式, 遂修改 print\_table 函数, 增加输出序号, 且使得序号直接对应了该出发城市及班次在相应 vector 中的下标, 使得遍历也不再需要, 可快速访问删除。

④路线咨询

可以很快发现，无论是最短用时、最少花费，还是最少中转，本质上都是最小权路的问题，那么都采用 Dijkstra 算法，通过调用一个 `short_path_dijkstra` 达到目的，可减少重复代码量，也更加模块化。权值的计算放入 `all_cost` 函数中，先暂时不管。

在这样修改 Dijkstra 算法的过程中，初始化各数组的部分没问题，但在后续选最小权边，且利用它更新各 `dist`、`pre` 函数过程中发现，先前到达某城市的时间 `ar_time` 在计算下一权值过程中是需要的：

一是为了保证时间的合理性，即下一班次的出发时间肯定得晚于 `ar_time`；

二是在计算最短用时时，需要考虑中转时间，那么上一班次的到达时间肯定也是必须的。综合考虑，增加 `int* kt[50]` 记录到达城市 `k` 的时间 `kt[k]`，如前所述，采用数组表示时间，故 `kt[k]` 是指向时间数组首元素的指针。

并考虑多重边的情况，增加 `int kth[50]` 记录到达该城市 `k`，是乘坐的 `pre[k]` 城市出发的第 `kth[k]` 趟班次。

基于上述两点，完善 `all_cost` 函数，根据 `ar_time` 确认时间合理性和计算用时，其余选择最小权多重边的过程无需赘述。关于时间复杂度，在前文有所述。

特别提及 `show_way` 函数中，总时间的计算是记录首尾时间相减，而非如总花费一般，对各权值求和，考虑到计算时间的复杂性，此举避免了一些相对繁杂的计算或者记录。

## 五、总结：

- 1、体验了从仅拿到功能需求，到最后实现相应需求，且效果尚可的项目全过程，学会了去分析需求，拆分模块，选用算法等；
- 2、主动去完善了容错机制，平常完成作业，大多数时候考虑的是输入合理的情况，此次项目促使站在用户角度，思考可能会输入的错误内容，以及明确应当怎样去测试，是有益的进步；
- 3、更加深入地体会到 C++ 面向对象的特性，用类来组织某些数据，留下接口以保证私密性，避免类的对象受到随意的访问及修改；
- 4、学习如何更好地分割功能置于不同函数，使得代码模块化，减少重复代码，以便于更广泛的使用，以及更为容易进行修改；
- 5、熟练并复习了 C++ 中类的定义，了解了一些字符串流及文件流的相关内容，以及在处理输入输出时一些需要注意的点也得到重温；
- 6、能够较为熟练地运用课内所学的栈、图等相关内容，并且认识到，能够掌握合宜的数据结构和算法，对于编写程序来说是大有助益的。

限于个人水平和时间，整个项目仍简单粗糙，但从无至有完成整个项目的过程是有意义的，从中收获许多进步。

## 六、参考资料：

课内 PPT 第八章图的 Dijkstra 算法部分，为适应需求有改动。

《C++ Primer》，了解文件流和字符串流，以及复习类的相关定义时用到。