全国交通咨询模拟报告

小组：刘听雨、何咏哲、王嵩岳

一、需求分析

1、提供对城市信息进行编辑（添加或删除的功能）；提供对列车时刻表和飞机航班进行编辑（添加或删除）的功能。（提供文件形式输入和键盘输入两种方式）

2、提供两种最优决策：最快或最省钱到达。全程只考虑一种交通工具。

3、旅途中耗费的总时间应该包括中转站的等候时间

4、咨询以用户和计算机对话方式进行。由用户输入起始站、终点站、最优决策原则和交通工具，输出信息：最快需要多长时间才能到达或者最少需要多少旅费才能到达，并详细说明依次于何时乘坐哪一趟列车或哪一次班机到何地

二、概要设计

1、数据类型

```
3   #define MAXVEX 100      //图中最大的顶点数
16  struct edgeinfo{        //车次或者航班的信息
17      int ivex;           //起点城市号
18      int jvex;           //终点城市号
19      char number[10];    //车次号
20      int money;          //$
21      int starttime;      //出发时间（用一天中的分钟表示：比如7：30为450）
22      int endtime;        //到达时间（同上）
23      int time;           //中途时间
24  };
25
26  struct edgenode{        //边
27      struct edgeinfo elem;
28      struct edgenode *nextedge;
29  }*p;
30
31  struct vnode{           //城市信息
32      char *cityname;
33      int citynumber;
34      struct edgenode *firstedge;
35  };
36
37  struct graph{           //图的结构
38      struct vnode adjlist[MAXVEX];   //邻接表
39      int vexnum,edgenum; //图中的顶点数和边数
40      int flag[MAXVEX];   //标记是否为图中节点，0表示不是
41  };
```

2、基本操作

void editEdgeFromFile(struct graph *g);

//从文件读入并且编辑边

void editEdgeFromUser(struct graph *g);

//从用户读入并且编辑边

void editCityFromUser(struct graph *gt,struct graph *ga);

//从用户读入并且编辑城市

void editCityFromFile(struct graph *gt,struct graph *ga);

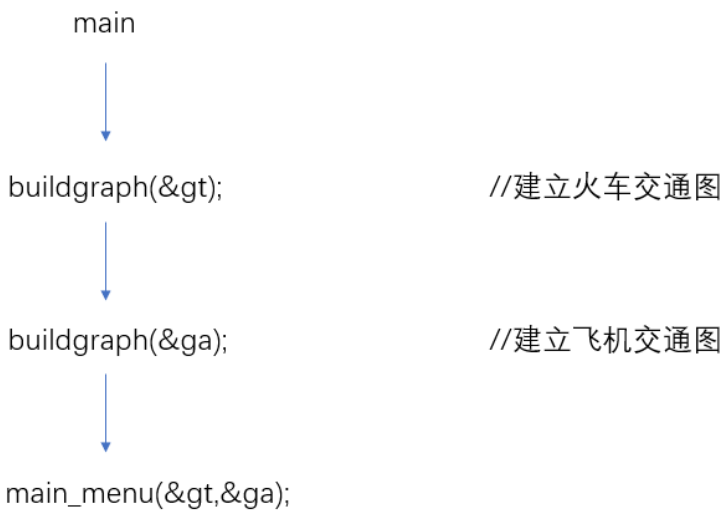
//从文件读入并且编辑城市

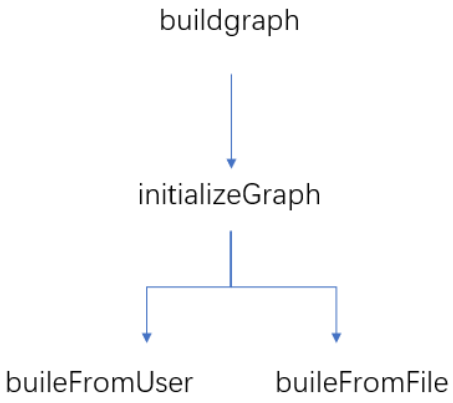
int getnum(int &i,char* tempstr);

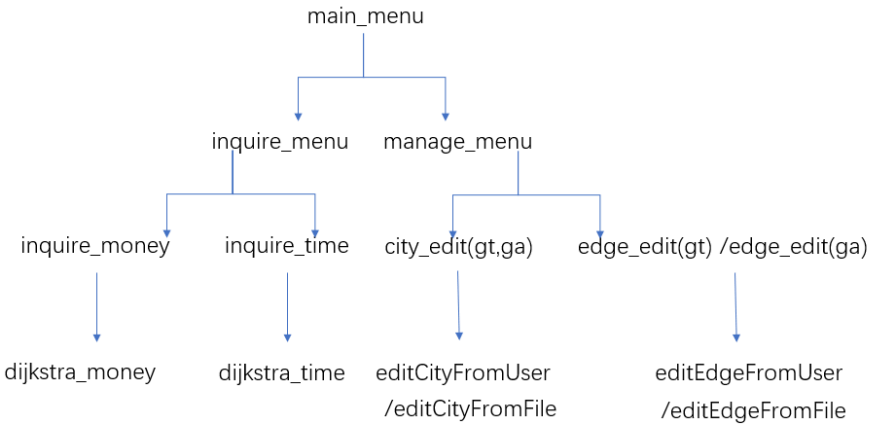
//读取从文件输入的数字

```
void buileFromUser(struct graph *g);
//根据用户输入的边建立一个图
void buileFromFile(struct graph *g);
//根据文件输入的边建立一个图
void initializeGraph(struct graph *g);
//初始化一个图
void city_edit(struct graph *gt,struct graph *ga);
//编辑城市
void edge_edit(struct graph *g);
//编辑边
void buildgraph(struct graph *g);
//建立一个图
void dijkstra_money(struct graph* g,int x,struct edgenode pre[])
//根据 dijkstra 算法计算最省钱路径
void inquire_money(struct graph* gt,struct graph* ga)
//查询最省钱路径
void dijkstra_time(struct graph* g,int x,struct edgenode pre[])
//根据 dijkstra 算法计算最快到达路径
void inquire_time(struct graph* gt,struct graph* ga)
//查询最快到达路径
void inquire_menu(struct graph* gt,struct graph* ga)
//查询菜单
void manage_menu(struct graph* gt,struct graph* ga)
//管理菜单
void main_menu(struct graph* gt,struct graph* ga)
//主菜单
```

3、流程以及调用关系







三、详细分析

1、插入删除操作（包括初始的建图）

（1）初始的建图（图的初始化）

```
char c;
int i=0,start=0;
start = getnum(i,tempstr);
struct edgenode* pointer, *temp;
if(!g->adjlist[start].firstedge){
    g->adjlist[start].firstedge = (struct edgenode*)malloc(sizeof(struct edgenode));
    g->adjlist[start].firstedge->nextedge = NULL;
}
pointer = NULL;
temp = g->adjlist[start].firstedge->nextedge;
pointer = (struct edgenode*)malloc(sizeof(struct edgenode));
g->adjlist[start].firstedge->nextedge = pointer;
pointer->nextedge = temp;
edgeinfo.ivex = start;
edgeinfo.jvex = getnum(i,tempstr);
int j=0;
while(isalnum(tempstr[i]))
    edgeinfo.number[j++] = tempstr[i++];
i++;
edgeinfo.money = getnum(i,tempstr);
edgeinfo.starttime = getnum(i,tempstr);
edgeinfo.endtime = getnum(i,tempstr);
edgeinfo.time = edgeinfo.endtime - edgeinfo.starttime;
count++;
```

（2）城市的插入删除

```
case 0:
    for(;k<gt->vexnum;k++){
        if(!strcmp(cityname,gt->adjlist[k].cityname))
            break;
        else if(k==gt->vexnum-1){
            cout<<"城市不存在，请重新尝试"<<endl;
            return;
        }
    }
    gt->flag[k] = 0;
    ga->flag[k] = 0;
    gt->vexnum--;
    ga->vexnum--;
    break;
case 1:
    gt->flag[gt->vexnum] = 1;
    ga->flag[ga->vexnum] = 1;
    gt->adjlist[gt->vexnum].cityname = (char*)malloc(sizeof(cityname));
    ga->adjlist[ga->vexnum].cityname = (char*)malloc(sizeof(cityname));
    memcpy(gt->adjlist[gt->vexnum].cityname,cityname,sizeof(cityname));
    memcpy(ga->adjlist[ga->vexnum].cityname,cityname,sizeof(cityname));
    gt->vexnum++;
    ga->vexnum++;
    break;
```

（3）边的插入删除

```
case 0:
    struct edgenode* pre,*next;
    pre = g->adjlist[startcity].firstedge;
    while(pre->nextedge!=NULL){
        if(strcmp(number,pre->nextedge->elem.number))
            break;
        else
            pre = pre->nextedge;
    }
    next = pre->nextedge;
    pre->nextedge = next->nextedge;
    free(next);
    g->edgenum--;
    break;
case 1:
    struct edgenode* pointer,*temp;
    pointer = g->adjlist[startcity].firstedge->nextedge;
    temp = (struct edgenode*)malloc(sizeof(struct edgenode));
    temp->elem.ivex = startcity; temp->elem.jvex = endcity; memcpy(temp->elem.number,number,sizeof(number));
    temp->elem.money = price; temp->elem.starttime = starttime; temp->elem.endtime = endtime;
    temp->elem.time = endtime - starttime;
    g->adjlist[startcity].firstedge->nextedge = temp;
    temp->nextedge = pointer;
    g->edgenum++;
    break;
```

可以看到对于图的编辑就是基本的图的节点和边等的插入和删除操作。
在建立初始的图和插入的时候，会分离节点（具体分析见下面）。

2、最省钱和最快路径

```
char c[10];
cout<<endl<<"请输入起点:"; scanf("%s",c);
for(int i=0;i<gt->vexnum;i++){
    if(strcmp(c,gt->adjlist[i].cityname)==0){st=i; break;}
}
cout<<endl<<"请输入终点:"; scanf("%s",c);
for(int i=0;i<gt->vexnum;i++){
    if(strcmp(c,gt->adjlist[i].cityname)==0){end=i; break;}
}
dijkstra_money(gt,st,pre);
int k=end;
struct edgenode list[MAXVEX];
int len=0;
while(k!=st){
    list[++len]=pre[k];
    k=pre[k].elem.ivex;
}
cout<<"最省钱路径:"<<endl;
for(int i=len;i>=1;i--){
    cout<<"车次号:"<<list[i].elem.number;
    cout<<"\t"<<gt->adjlist[list[i].elem.ivex].cityname<<"--"<<gt->adjlist[list[i].elem.jvex].cityname;
    cout<<"\t起止时间:";
    printf("%02d",list[i].elem.starttime/60);
    cout<<":";
    printf("%02d",list[i].elem.starttime%60);
    cout<<"--";
    printf("%02d",list[i].elem.endtime/60);
    cout<<":";
    printf("%02d",list[i].elem.endtime%60);
    cout<<"\t价钱:"<<list[i].elem.money<<endl;
}
break;
```

首先输入起点和终点，再使用 dijktra 算法找到“最短路”。其中使用 pre 记录路径，方便接下来的依次打印。

```
int n=g->vexnum;
int t[MAXVEX],d[MAXVEX],endtime[MAXVEX];        //t表示有没有锁定，d表示最短时间，endtime表示到达时间
for(int i=0;i<n;i++) d[i]=99999;
d[x]=0; endtime[x]=-1; int min_d,num;
for(int i=0;i<n;i++){
    min_d=99999;
    for(int j=0;j<n;j++)
        if(t[j]!=1&&g->flag[j]==1&&d[j]<min_d){min_d=d[j];num=j;}
    t[num]=1;
    for(p=g->adjlist[num].firstedge->nextedge;p!=NULL;p=p->nextedge){
        if(p->elem.starttime<endtime[num]) continue;
        if(d[num]+p->elem.time<d[p->elem.jvex]){
            d[p->elem.jvex]=d[num]+p->elem.time;
            pre[p->elem.jvex]=*p;
            endtime[p->elem.jvex]=p->elem.endtime;
        }
    }
}
```

使用最短路算法，其中值得注意的是若起始时间在旅客到达该站之前，旅客是无法乘坐该趟火车/飞机的。加了一个判断，保证了算法的真实性。

值得注意的是为了保证在求最快到达的时候包括了等待时间，每个有两条或者两条以上路径交汇的节点要分成两个节点或者多个节点。并且将等待时间形成一条边从一个节点指向其分离出来的节点，然后再进行 dijkstra 算法。

四、设计和调试分析

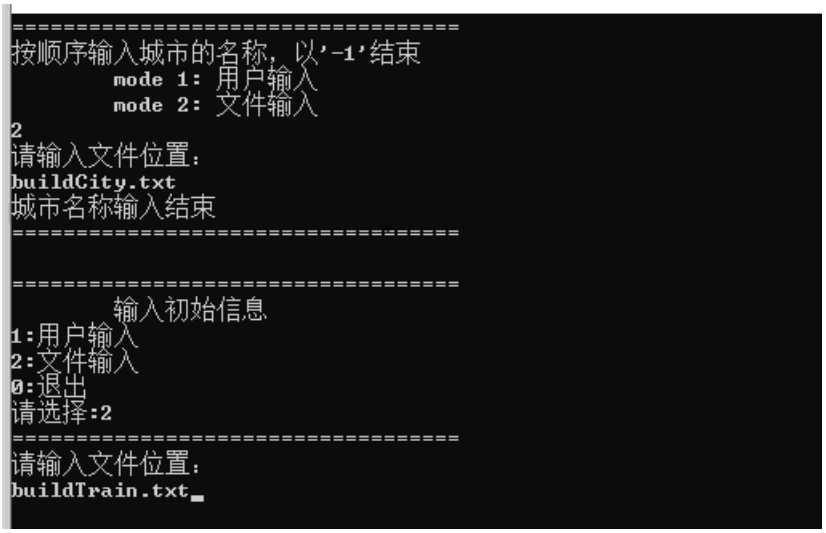
1、在文件读入的时候会出现一些问题，因为没有讲过文件读入，所以在文件读入时的内存分配上有些问题，经过排查，使用了正确的文件读入方式。

2、考虑道路网多是稀疏网，故采用邻接表作存储结构，其空间复杂度为 O(e)，此时的时间复杂度也为 O(e)。构建邻接表的时间度为 O(n+e)，多分离节点其时间复杂度不超过 O(e)。核心算法基于 dijkstra 算法，其时间复杂度为 $O(n^2)$。输出路径的时间复杂度不超过 O(e)。由此，本算法的总时间复杂度不超过 $O(n^2)$。

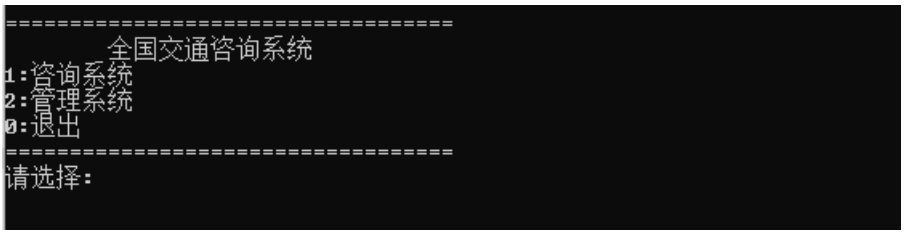
3、采用了较为简单的 dijkstra 算法，但是考虑到实际的情况（列车的时间冲突和等待时间）。在该算法上进行了一些修改，所以在处理实际问题的时候，不能简单的套用算法，要进行适当的修改。

五、用户手册

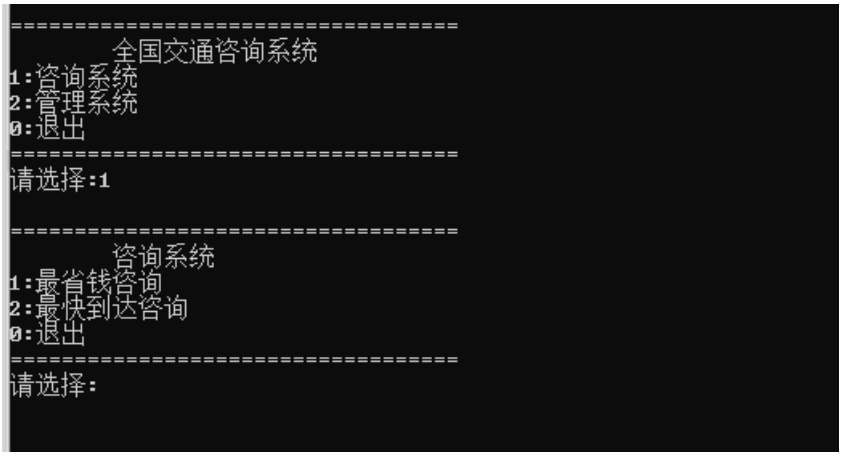
1、首先要求输入初始的图（以文件输入为例）



2、接着进入主菜单



3、可以选择查询或者管理，输入错误会报错



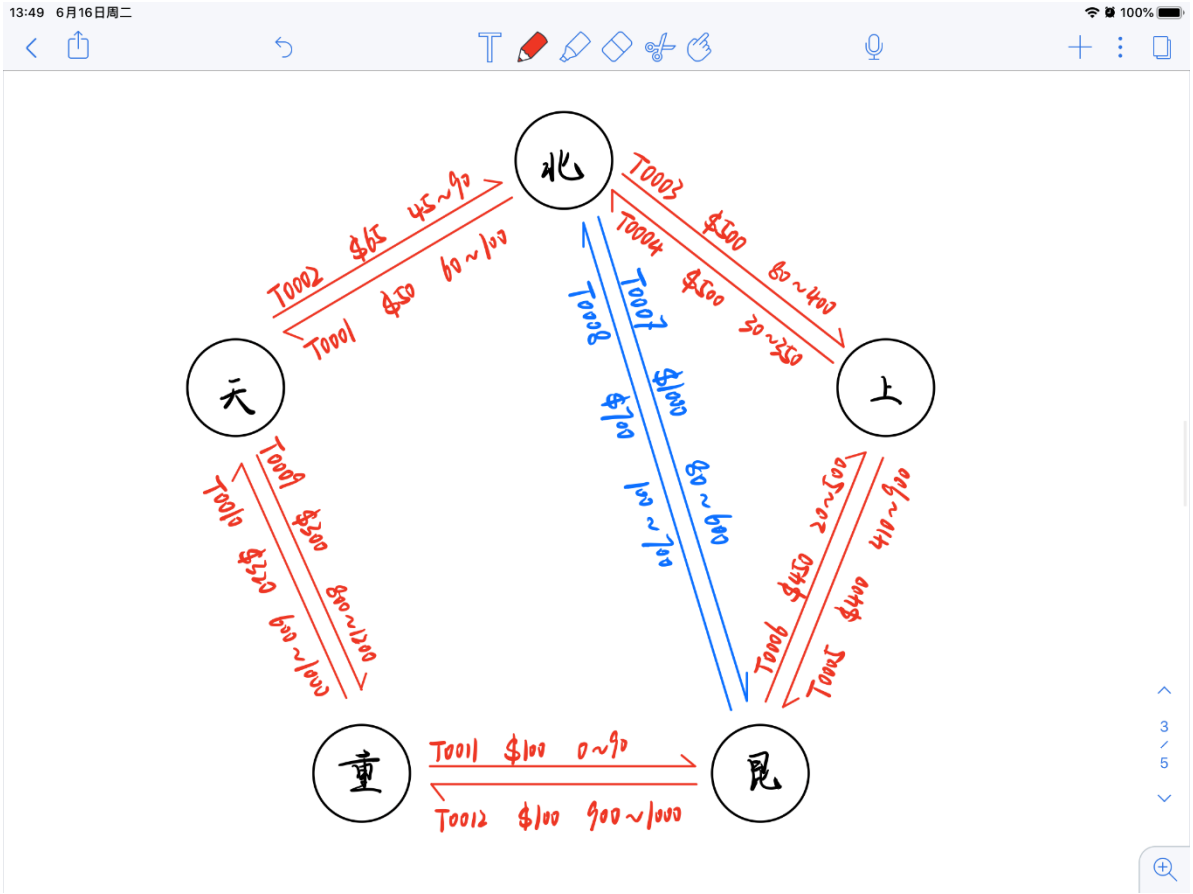
4、查询火车，最省钱，输入起点和终点

5、管理系统，可以选择用户输入或者文件输入

六、测试结果

0、交通图初始化

以火车为例，下图为根据附件 buildCity.txt 和 buildTrain.txt 所建立的交通图：



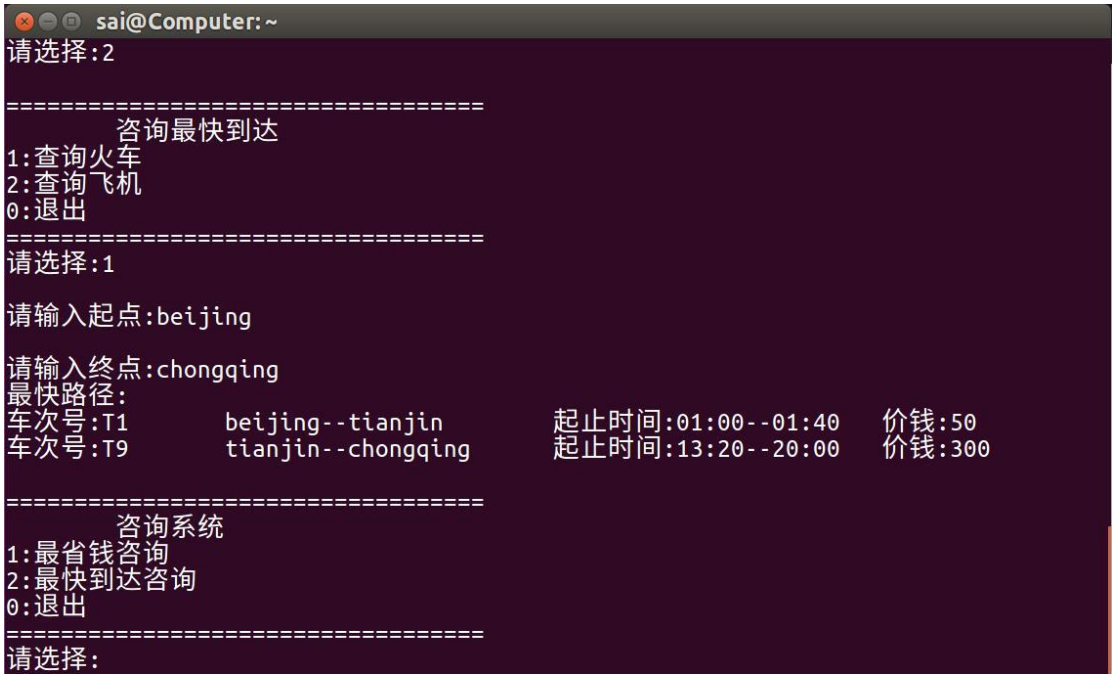
1、最省钱查询

```
sai@Computer:~
请选择:1

==================================
        咨询最省钱到达
1:查询火车
2:查询飞机
0:退出
==================================
请选择:1

请输入起点:beijing

请输入终点:kunming
最省钱路径:
车次号:T3       beijing--shanghai       起止时间:01:20--06:40   价钱:500
车次号:T5       shanghai--kunming       起止时间:06:50--15:00   价钱:400

==================================
        咨询系统
1:最省钱咨询
2:最快到达咨询
0:退出
==================================
请选择:
```

由之前构建的交通图可以看到，从北京出发到达昆明可以有三条不同的路径，其一是【北京->上海->昆明】，花费 900 元；其二是【北京->昆明】，花费 1000 元，其三是【北京->天津->重庆->昆明】，花费 450 元。但是事实上第三条途径

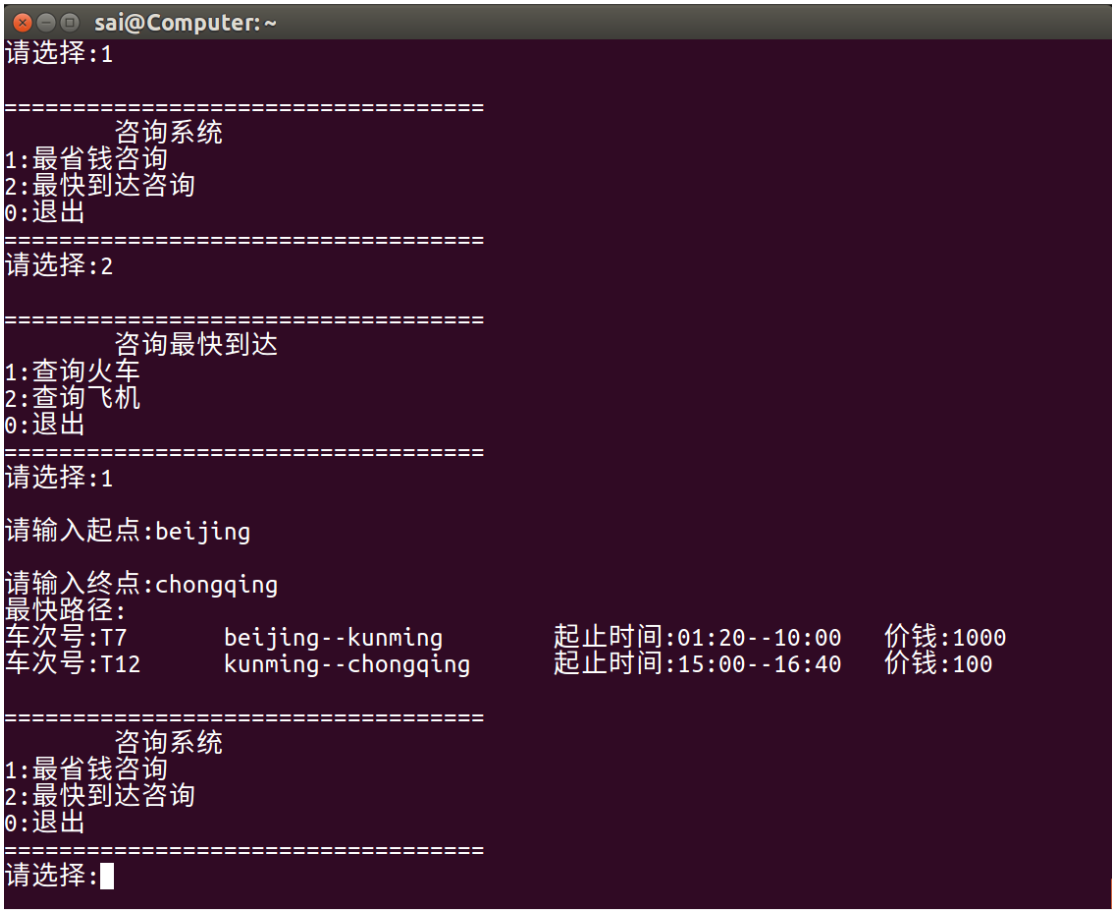
不存在，因为重庆->昆明的发车时间比天津->重庆的到达时间还要早，故不可能走这条途径。因此在最省钱路径中我们选择了【北京->上海->昆明】，结果正确。

2、最快查询



搜索北京到重庆的最快路径，得到的结果是【北京->天津->重庆】，表面上看似乎没有问题，因为【北京->天津】和【天津->重庆】分别耗时 40 和 400，比起另一条路径【北京->昆明】和【昆明->重庆】所耗时间 520+100 来说更短。但是在这次尝试中我们并没有考虑等待时间的问题，若选择第一条路径，旅客需要在天津等待 700 的时间！而选择【北京->昆明->重庆】的话只需要等待 300 的时间。

在意识到算法的缺陷之后，我们对原来的程序进行了修正。由于 dijkstra 算法并不能够直接计算等待时间，因此我们采取了“曲线救国”战略，在建立初始图的时候就将有两条或者两条以上路径交汇的节点分成了两个节点或者多个节点。借助“节点分离”的思想，dijkstra 算法现在能够将等待时间一同考虑在内了。修正后的重新尝试如下：



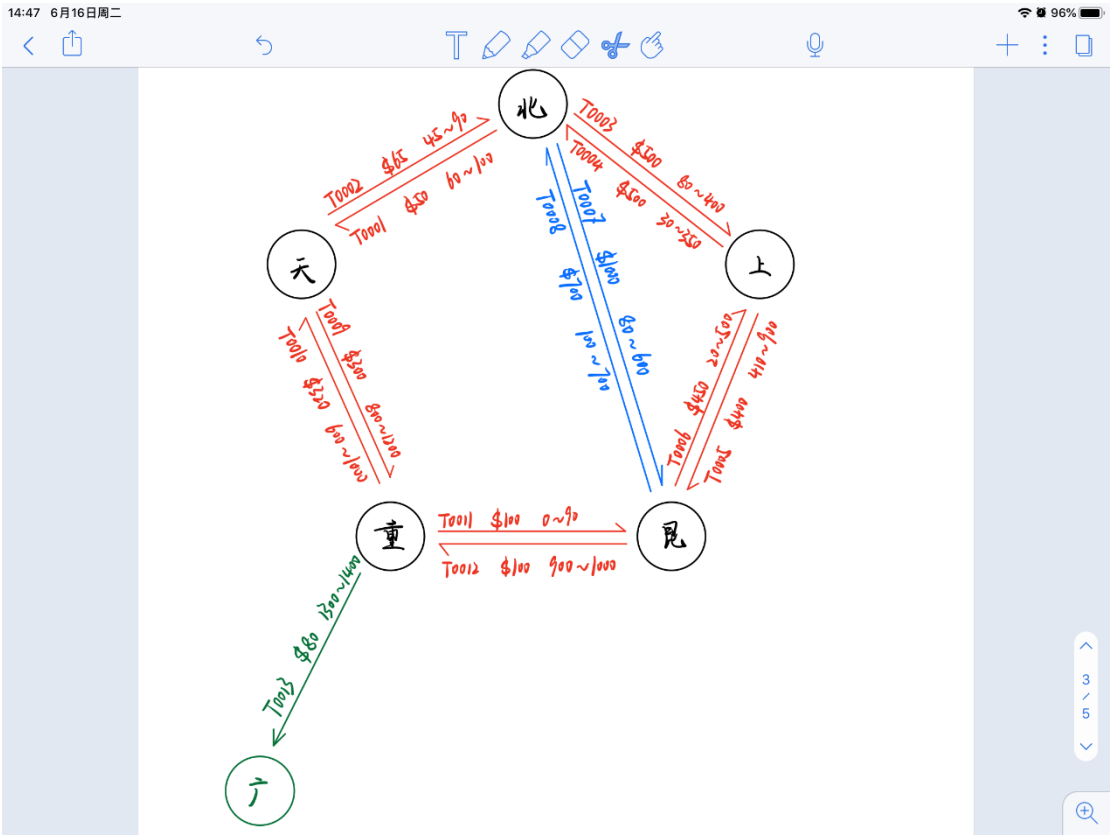
所以可以看到，原始图像简单的使用 dijkstra 算法会出错，但是将节点分离之后则可以使用 dijkstra 算法计算最快到达

3、插入一个新城市，和到它的边

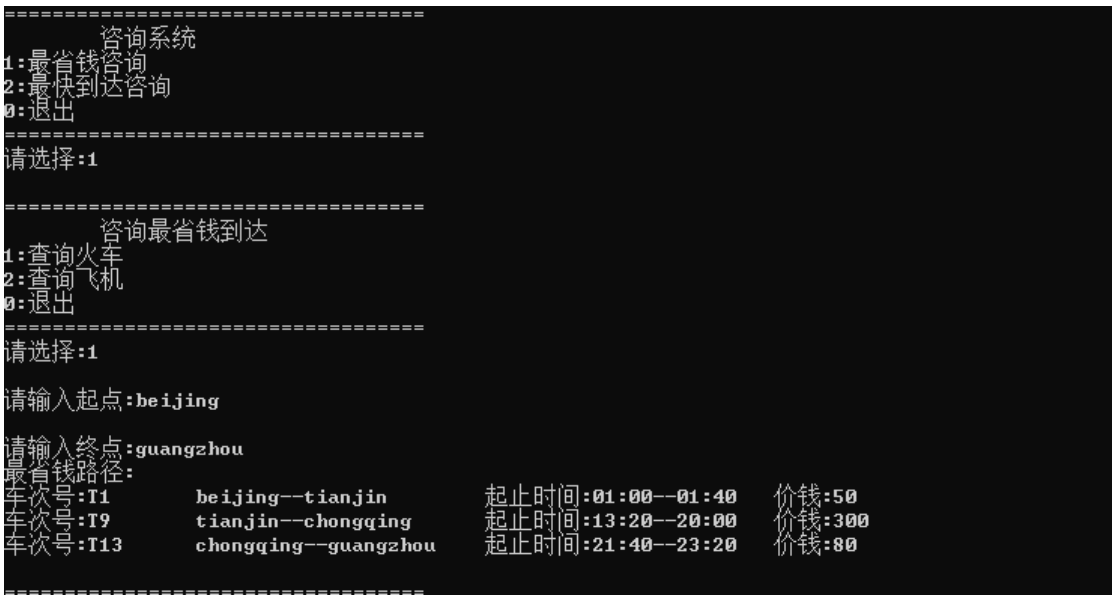
插入城市广州，插入一条从重庆到广州的列车

```
sai@Computer:~
==================================
请依次输入 操作 城市名
(其中1代表插入，0代表删除)
输入以2开头结束
1 guangzhou
2
输入结束
==================================

==================================
        管理系统
1:城市编辑
2:列车时刻表编辑
3:航班时刻表编辑
0:退出
==================================
请选择:2
==================================
选择修改模式
        1：用户输入
        2：文件输入
        3：退出
1
==================================
请依次输入：操作 起点 终点 车次 票价 出发时间 到达时间
(其中1代表插入，0代表删除)
输入以2开头结束
1 chongqing guangzhou T13 80 1300 1400
2
输入结束
==================================
```

更新后的交通图如下

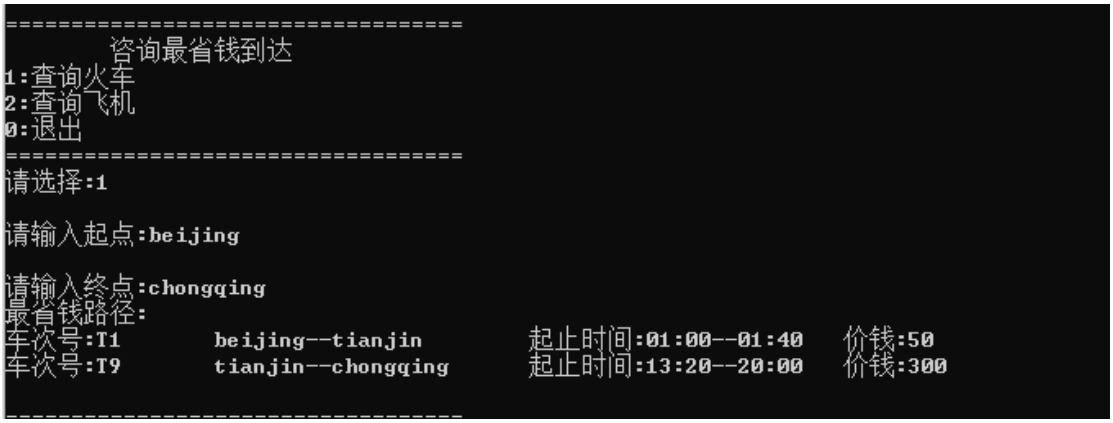


现在我们进行查询测试，查询北京到广州的最省钱路径，得到了【北京->天津->重庆->广州】的路线，由交通图可知结果正确。



4、删除一个城市

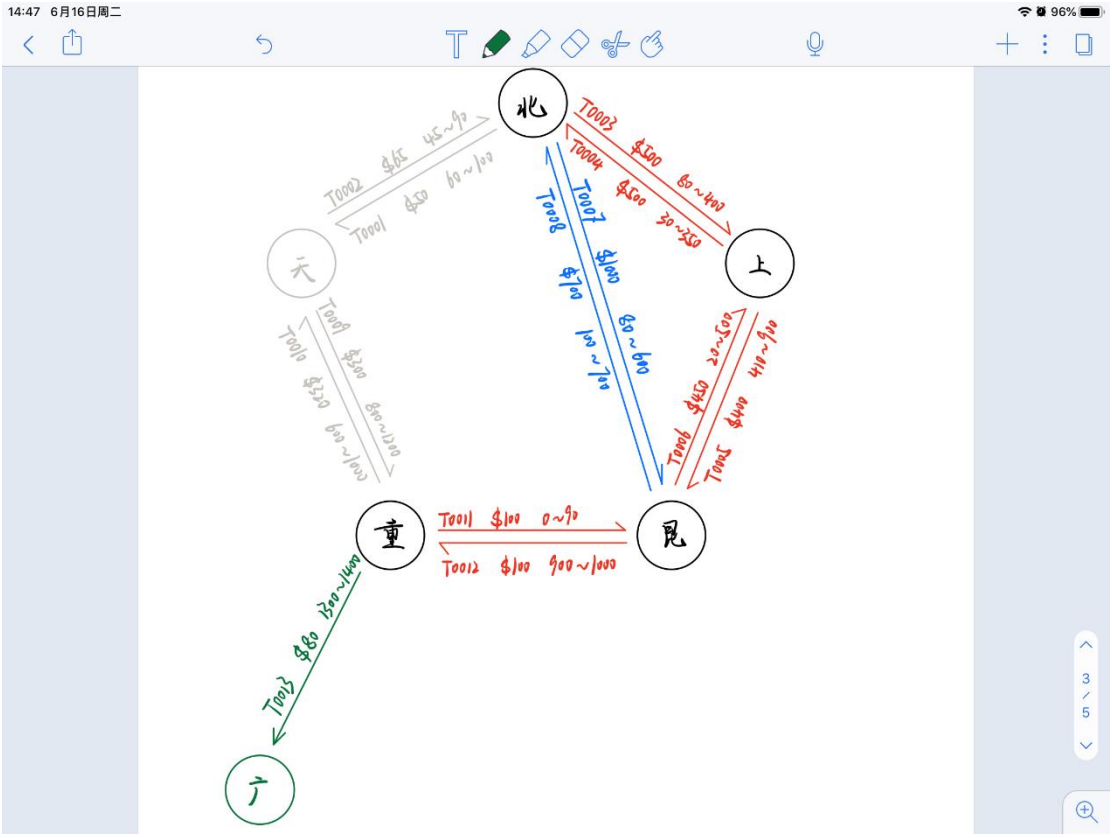
删除之前查询北京到重庆的最省钱路径，得到【北京->天津->重庆】



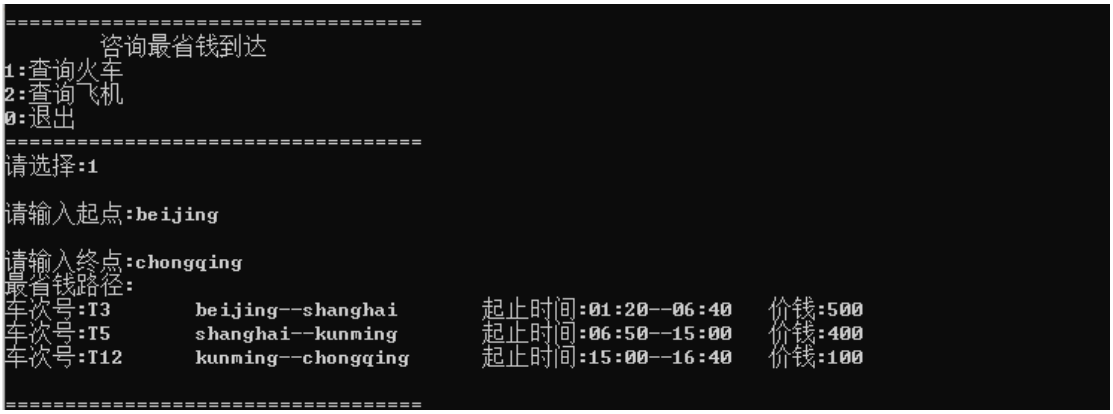
我们将城市天津删除。

```
==================================
        管理系统
1:城市编辑
2:列车时刻表编辑
3:航班时刻表编辑
0:退出
==================================
请选择:1
==================================
选择修改模式
        1: 用户输入
        2: 文件输入
        3: 退出
1
==================================
请依次输入 操作 城市名
(其中1代表插入，0代表删除)
输入以2开头结束
0 tianjin
2
输入结束
==================================
```

删除天津后的交通图如下：



删除之后，此时若再进行北京到重庆的最省钱路径查询，预测所得结果会是【北京->上海->昆明->重庆】，验证如下：



结果正确无误！

七、附录

traffic – final.c //主程序
buildTrain.txt //火车数据
buildCity.txt //城市数据
buildAir.txt //飞机数据

注解：

火车和飞机的数据格式：

起始号，终点号，车次，费用，起始时间，终止时间（按分钟计）