# 西 安 邮 电 大 学

（计算机学院）

# 数据结构课程设计报告

**题　目：全国交通咨询系统**

专业名称：软件工程

班　　级：软件 1602

学生姓名：赖伟峰

学号（8 位）：05168130

指导教师：乔　平　安

设计起止时间：2018 年 1 月 2 日—2018 年 1 月 5 日

## 一．设计目的

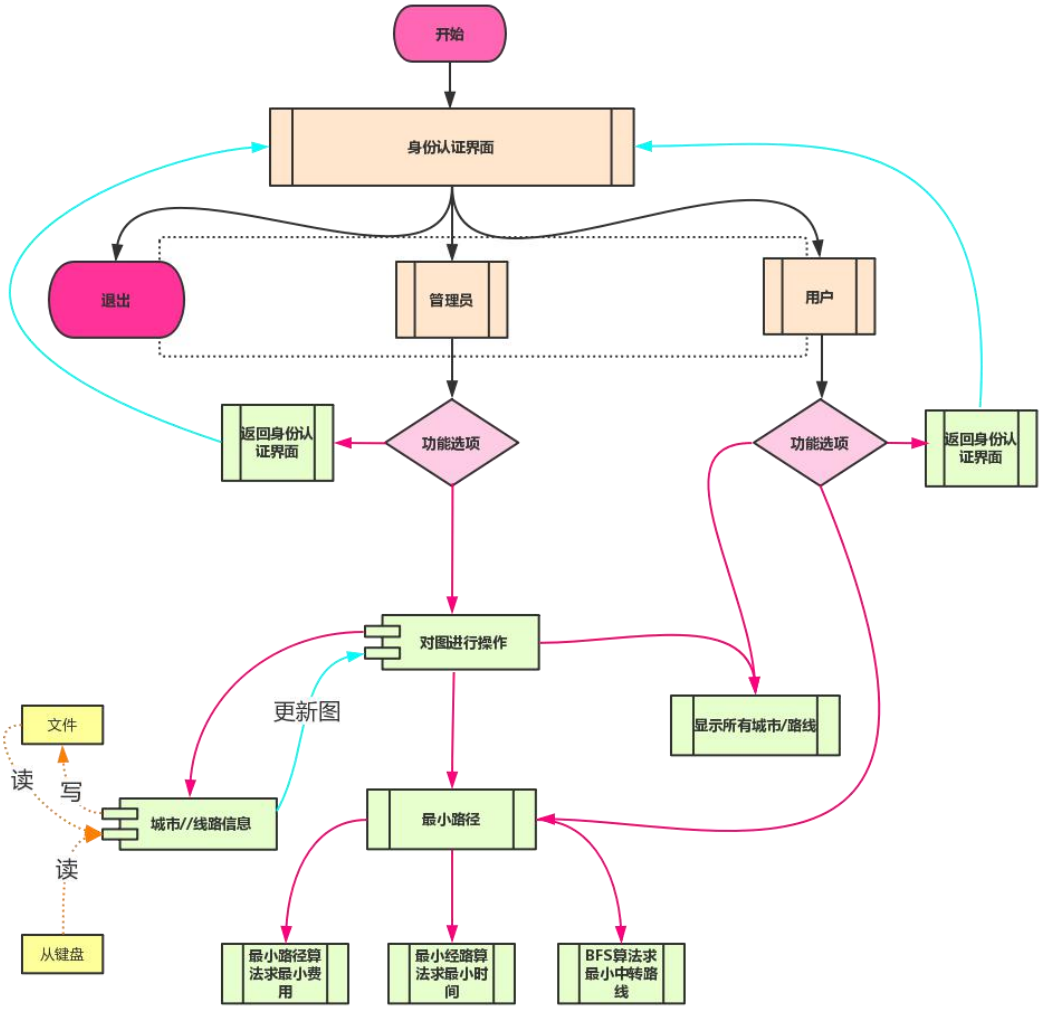
全国交通咨询模拟。处于不同目的的旅客对交通工具有不同的要求。例如，因公出差的旅客希望在旅途中的时间尽可能的短，出门旅游的游客则期望旅费尽可能省，而老年旅客则需要中转次数最少。编制一个全国城市间的交通咨询程序，为旅客提供两种或三种最优决策的交通咨询。

## 二．设计内容

提供用户以及管理员功能，用户可以对交通图进行查询，而管理员可以对交通图进行增删查改，同时管理员可以登陆、修改密码等待操作，界面采用字符界面。这样操作，更加真实地模拟了交通咨询系统。关于要求的功能，实现了城市线路的增加、删除、显示，基于 Dijkstra 的从源点到汇点的最小费用算法与最小时间算法。

## 三．概要设计

## 1．功能模块图；



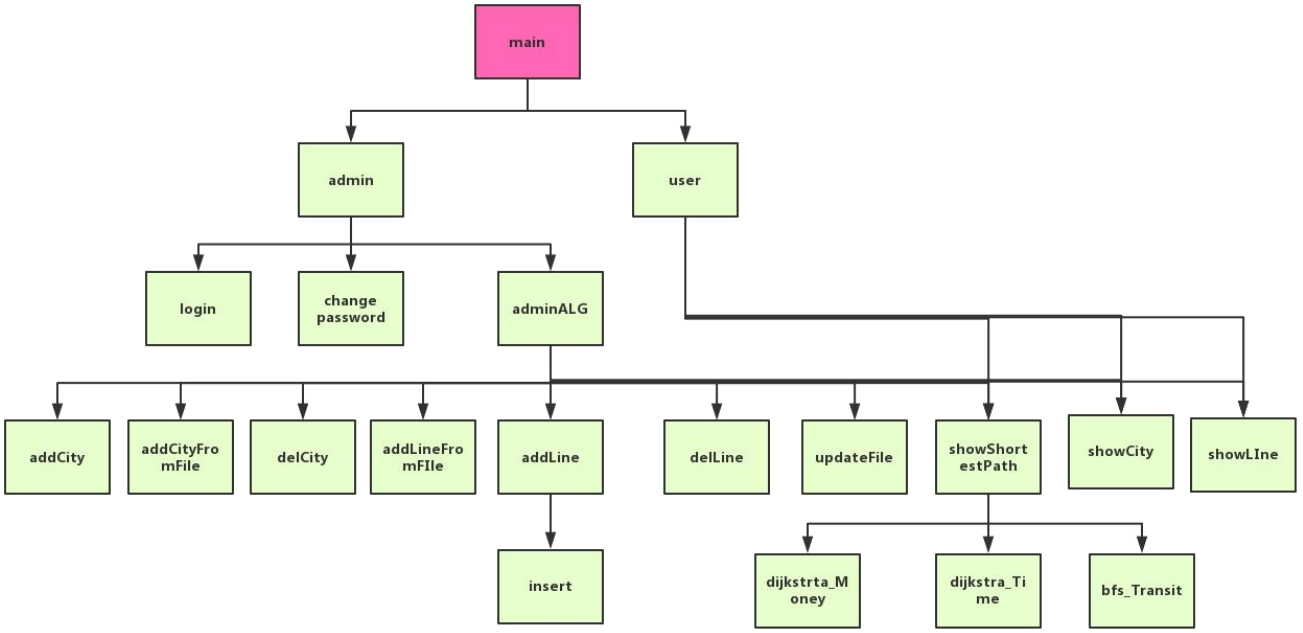

## 2．各个模块详细的功能描述。

查询城市编号：头结点建立顶点表时存储的是城市对应的序号
手动添加城市
从文件读取以添加城市
删除城市：删除城市时需要删除与该城市相关的所有线路
输出所有城市
更新城市列表：当新建城市个数加原本已存在城市个数大于 MAXSIZE 时，需要开辟空间存储新城市并 ++MAXSIZE
手动添加线路
插入线路：由于线路信息存于表结点里，所以需要新建表结点并加入对应起始城市的边表
从文件中读取线路

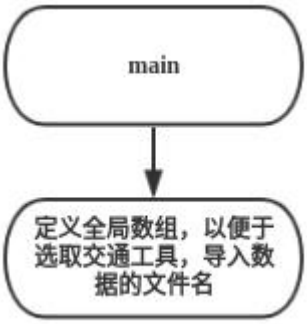
删除线路
求最少花费路径
求最少时间路径

## 四．详细设计

### 1．功能函数的调用关系图




### 2．各功能函数的数据流程图

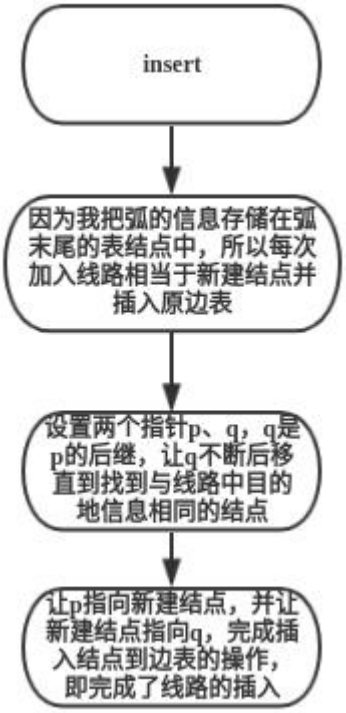


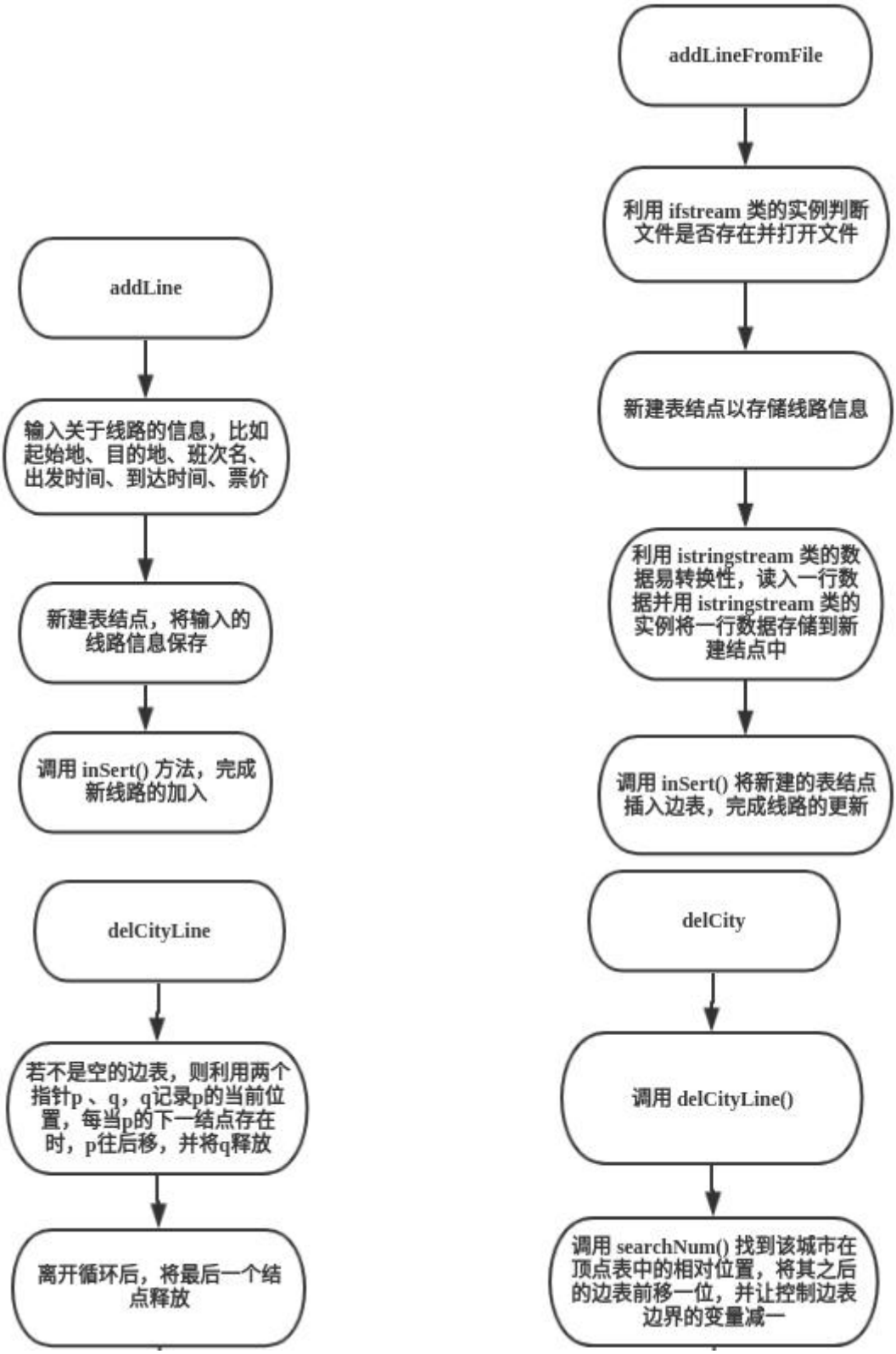


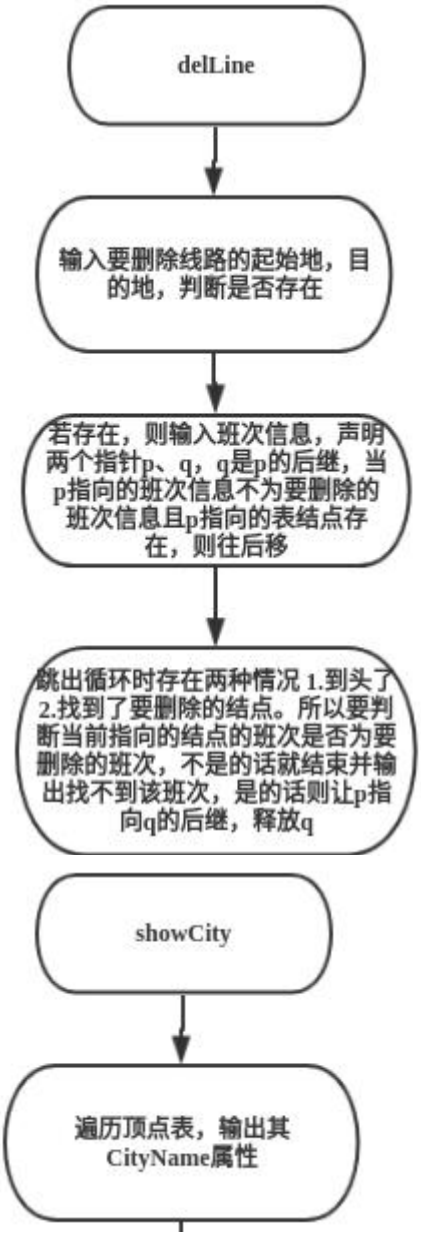
reSize
在原顶点表 CityList[] 开辟新的空间，并初始化


addCity
调用 searchNum，判断是否在顶点表里已存在该城市
若不存在，则在顶点表中添加新的顶点


addCityFromFile
利用 ifstream 类的实例判断文件是否存在并读取
设置 string 变量存储城市名，并调用 addCity() 函数将城市添加到顶点表中

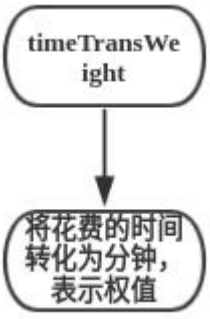

insert
因为我把弧的信息存储在弧末尾的表结点中，所以每次加入线路相当于新建结点并插入原边表
设置两个指针p、q，q是p的后继，让q不断后移直到找到与线路中目的地信息相同的结点
让p指向新建结点，并让新建结点指向q，完成插入结点到边表的操作，即完成了线路的插入

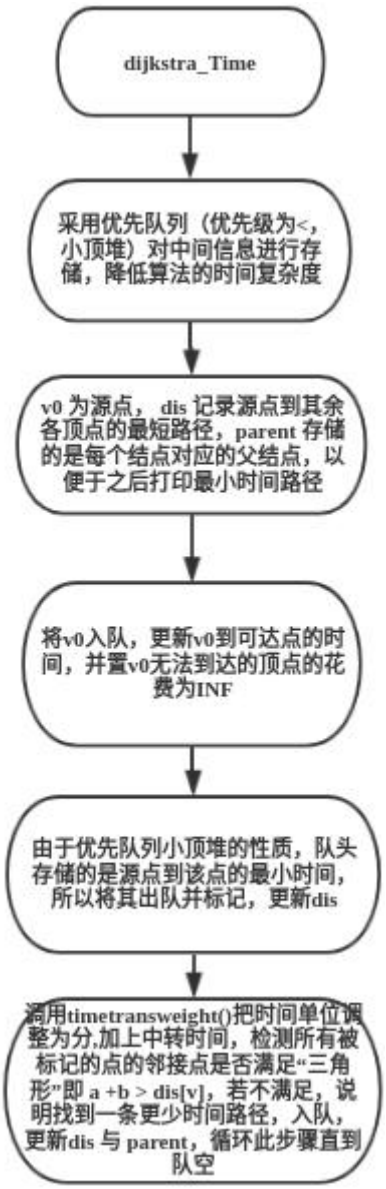

addLine
输入关于线路的信息，比如起始地、目的地、班次名、出发时间、到达时间、票价
新建表结点，将输入的线路信息保存
调用 inSert() 方法，完成新线路的加入
delCityLine
若不是空的边表，则利用两个指针p 、q，q记录p的当前位置，每当p的下一结点存在时，p往后移，并将q释放
离开循环后，将最后一个结点释放
addLineFromFile
利用 ifstream 类的实例判断文件是否存在并打开文件
新建表结点以存储线路信息
利用 istringstream 类的数据易转换性，读入一行数据并用 istringstream 类的实例将一行数据存储到新建结点中
调用 inSert() 将新建的表结点插入边表，完成线路的更新
delCity
调用 delCityLine()
调用 searchNum() 找到该城市在顶点表中的相对位置，将其之后的边表前移一位，并让控制边表边界的变量减一

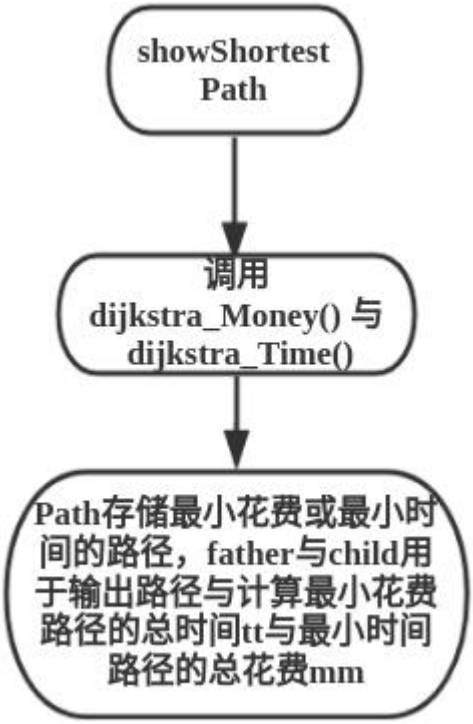

delLine
输入要删除线路的起始地，目的地，判断是否存在
若存在，则输入班次信息，声明两个指针p、q，q是p的后继，当p指向的班次信息不为要删除的班次信息且p指向的表结点存在，则往后移
跳出循环时存在两种情况 1.到头了 2.找到了要删除的结点。所以要判断当前指向的结点的班次是否为要删除的班次，不是的话就结束并输出找不到该班次，是的话则让p指向q的后继，释放q
showCity
遍历顶点表，输出其CityName属性


updateFile
利用类 ofstream 的实例实现更新到目标文件
showLine
遍历顶点表的同时遍历边表，从边表的首结点开始往后遍历


dijkstra_Money
采用优先队列（优先级为<，小顶堆）对中间信息进行存储，降低算法的时间复杂度
v0 为源点， dis 记录源点到其余各顶点的最短路径，parent 存储的是每个结点对应的父结点，以便于之后打印最小花费路径
将v0入队，更新v0到可达点的花费，并置v0无法到达的顶点的花费为INF
由于优先队列小顶堆的性质，队头存储的是源点到该点的最小花费，所以将其出队并标记，更新dis，
检测所有被标记的点的邻接点是否满足“三角形”即 a +b > dis[v]，若不满足，说明找到一条更小花费路，入队，更新dis 与 parent，循环此步骤直到队空


timeTransWe ight
将花费的时间转化为分钟，表示权值


dijkstra_Time
采用优先队列（优先级为<，小顶堆）对中间信息进行存储，降低算法的时间复杂度
v0 为源点， dis 记录源点到其余各顶点的最短路径，parent 存储的是每个结点对应的父结点，以便于之后打印最小时间路径
将v0入队，更新v0到可达点的时间，并置v0无法到达的顶点的花费为INF
由于优先队列小顶堆的性质，队头存储的是源点到该点的最小时间，所以将其出队并标记，更新dis
调用timetransweight()把时间单位调整为分,加上中转时间，检测所有被标记的点的邻接点是否满足“三角形”即 a +b > dis[v]，若不满足，说明找到一条更少时间路径，入队，更新dis 与 parent，循环此步骤直到队空




### 3．重点设计及编码

用优先队列优化的基于 dijkstra 算法的最小费用与最小时间算法，代码如下：

```
//最少花费路径
struct Node {
    int id; //源顶点 id
    float money; //估算距离（费用）
    //由于 stl 中优先队列的第三个参数是 greater,而我们需要的是小顶堆，所以因重载运算符 <
    friend bool operator < (struct Node a, struct Node b) {
        return a.money > b.money;
    }
};
void ALGraph::dijkstra_Money (int v0, int *parent, Node *dis) {
    priority_queue<Node> q; //优化插入(更新)和取出最小值两个操作，队列存储最短距离与索引的编号
    //parent[]记录每个顶点的父亲结点
    //dis[]记录源点到每个估算距离，最后更新为源点到所有顶点的最短距离
    bool visited[MaxCityNum]; //判断下标对应的顶点是否算出最短路径或者说是否在最短路径树中
    //初始化
    int i;
    for (i = 0; i < CityNum; ++i) {
        dis[i].id = i;
        dis[i].money = INF;
        parent[i] = -1; //每个顶点都没有父结点
```

```
        visited[i] = false; //都未找到最短路
    }
    dis[v0].money = 0; //源点到源点最短路权值为 0
    q.push(dis[v0]); //压入队列
    while (!q.empty()) { //队列空说明完成了求解 v0 到其余各顶点的最短路径
        Node cd = q.top(); //取最小估算距离顶点
        q.pop();
        int u = cd.id;

        if (visited[u]) { //被标记了，就无需对其进行更新最短距离等等操作
            continue;
        }
        visited[u] = true;
        LineNode *p = CityList[u].FirstLine;
        //松弛操作
        while(p) { //找所有与它相邻的顶点，进行松弛操作，更新估算距离，压入队列
            int v = searchCityNum(p->EndName);
            float m = p->Info->SpendMoney;
            if (!visited[v] && dis[v].money > dis[u].money + m) {
                dis[v].money = dis[u].money + m;
                parent[v] = u;
                q.push(dis[v]);
            }
            p = p->NextLine;
        }
    }// while (!q.empty())
}//dijkstra_Money

//最少时间路径
struct Node1 {
    int id; //源顶点 id
    int tt; //估算距离(时间)
    Time et; //到达时间
    friend bool operator < (struct Node1 a, struct Node1 b){
        return a.tt > b.tt;
    }
};
int ALGraph::timeTransWeight (const Time& t) {
    return (t.day*24 + t.hour)*60 + t.minute;
}

void ALGraph::dijkstra_Time (int v0, int *parent, Node1 *dis) {
    priority_queue<Node1> q1;
    //parent[]记录每个顶点的父亲结点
    //dis[]记录源点到每个估算距离，最后更新为源点到所有顶点的最短距离
    bool visited[MaxCityNum]; //判断下标对应的顶点是否算出最短路径或者说是否在
最短路径树中
    int i;
    for (i = 0; i < CityNum; ++i) {
        dis[i].id = i;
        dis[i].tt = INF;
        dis[i].et = {0, 0, 0};
        parent[i] = -1; //都一个顶点都没有父结点
```

```
            visited[i] = false; //都未找到最短路径
        }
        dis[v0].tt = 0;
        q1.push(dis[v0]);
        while (!q1.empty()) {
            Node1 cd = q1.top(); //取出最短距离的点的序号
            q1.pop();
            int u = cd.id;

            if (visited[u]) {
                continue;
            }
            visited[u] = 1;

            LineNode *p = CityList[u].FirstLine;
            //找出所有相邻点，进行松弛操作，即更新 dis
            while (p) {
                int v = searchCityNum(p->EndName);
                int t = timeTransWeight(p->Info->SpendTime);
                Time st = p->Info->StartTime; //本条线路开始时间
                //计算中转的时间开销
                if (u != v0) { //注意源点到任何点都没有中转时间
                    int change = timeTransWeight(st - dis[u].et); //当前路线的开车时
间-始发站的上一到站时间
                    t += change;
                }
                if (!visited[v] && dis[v].tt > dis[u].tt + t) {
                    dis[v].tt = dis[u].tt + t;
                    dis[v].et = p->Info->EndTime;
                    parent[v] = u;
                    q1.push(dis[v]);
                }
                p = p->NextLine;
            }//while (p)
        }//while (!q1.empty())
}//dijkstra_Time
```

## 五・测试数据及运行结果

### 1・正常测试数据和运行结果

要求提供 3 组正常测试数据和运行结果

1.昆明 成都 （最小花费）

```
城市列表如上，请输入目的城市：最省钱路径
  昆明    成都 CZ1018 07:50,+0 09:25,+0 01:35,+0 500.000
一共花费500.000元和95分钟！
```

2.导入线路

```
    4.从文件中添加Flight 线路！
从Flight.txt中读取并导入线路！
线路导入完毕！
```

```
    8.显示所有线路！
系统中有 22 条线路的信息
出发城市|到达城市|班次名|出发时间|到达时间||||用时|||票价
  南昌    上海 CZ1000 14:20,+0 15:50,+0 01:30,+0 380.000
  上海    南昌 CZ1001 17:00,+0 18:45,+0 01:45,+0 380.000
  天津    沈阳 CZ1002 09:35,+0 10:50,+0 01:15,+0 450.000
  沈阳    天津 CZ1003 16:45,+0 18:25,+0 01:40,+0 450.000
  北京 呼和浩特 CZ1004 11:55,+0 13:00,+0 01:05,+0 370.000
  北京    郑州 CZ1005 21:10,+0 23:00,+0 01:50,+0 770.000
  郑州    北京 CZ1006 09:45,+0 11:25,+0 01:40,+0 460.000
呼和浩特    北京 CZ1007 13:45,+0 14:50,+0 01:05,+0 330.000
呼和浩特    兰州 CZ1008 19:25,+0 21:05,+0 01:40,+0 600.000
  兰州 呼和浩特 CZ1009 11:35,+0 13:10,+0 01:35,+0 700.000
  兰州 乌鲁木齐 CZ1010 10:10,+0 12:50,+0 02:40,+0 890.000
  兰州    西安 CZ1011 13:40,+0 15:00,+0 01:20,+0 500.000
乌鲁木齐    兰州 CZ1012 13:45,+0 16:15,+0 02:30,+0 890.000
  西安    兰州 CZ1013 22:25,+0 23:30,+0 01:05,+0 540.000
  西安    成都 CZ1014 21:45,+0 23:20,+0 01:35,+0 640.000
  成都    西安 CZ1015 07:20,+0 08:40,+0 01:20,+0 560.000
  成都    昆明 CZ1016 09:45,+0 11:30,+0 01:45,+0 500.000
  成都    贵阳 CZ1017 18:50,+0 20:10,+0 01:20,+0 670.000
  昆明    成都 CZ1018 07:50,+0 09:25,+0 01:35,+0 500.000
  昆明    贵阳 CZ1019 20:15,+0 21:30,+0 01:15,+0 600.000
  贵阳    成都 CZ1020 22:35,+0 23:55,+0 01:20,+0 620.000
  贵阳    昆明 CZ1021 22:10,+0 23:25,+0 01:15,+0 510.000
```

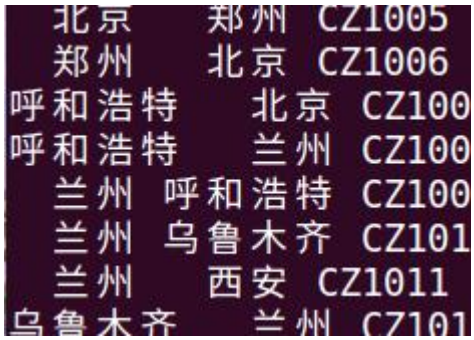
3.删除线路

```
    6.删除Flight 线路！
请输入起点城市：南昌
请输入终点城市：上海
起点城市与终点城市的路线有：
出发城市|到达城市|班次名|出发时间|到达时间||||用时|||票价
  南昌    上海 CZ1000 14:20,+0 15:50,+0 01:30,+0 380.000
请输入你想要删除的航班号：
```

```
    8.显示所有线路！
系统中有 21 条线路的信息
出发城市|到达城市|班次名|出发时间|到达时间||||用时|||票价
  上海    南昌 CZ1001 17:00,+0 18:45,+0 01:45,+0 380.000
  天津    沈阳 CZ1002 09:35,+0 10:50,+0 01:15,+0 450.000
  沈阳    天津 CZ1003 16:45,+0 18:25,+0 01:40,+0 450.000
  北京 呼和浩特 CZ1004 11:55,+0 13:00,+0 01:05,+0 370.000
  北京    郑州 CZ1005 21:10,+0 23:00,+0 01:50,+0 770.000
  郑州    北京 CZ1006 09:45,+0 11:25,+0 01:40,+0 460.000
呼和浩特    北京 CZ1007 13:45,+0 14:50,+0 01:05,+0 330.000
呼和浩特    兰州 CZ1008 19:25,+0 21:05,+0 01:40,+0 600.000
  兰州 呼和浩特 CZ1009 11:35,+0 13:10,+0 01:35,+0 700.000
  兰州 乌鲁木齐 CZ1010 10:10,+0 12:50,+0 02:40,+0 890.000
  兰州    西安 CZ1011 13:40,+0 15:00,+0 01:20,+0 500.000
乌鲁木齐    兰州 CZ1012 13:45,+0 16:15,+0 02:30,+0 890.000
  西安    兰州 CZ1013 22:25,+0 23:30,+0 01:05,+0 540.000
  西安    成都 CZ1014 21:45,+0 23:20,+0 01:35,+0 640.000
  成都    西安 CZ1015 07:20,+0 08:40,+0 01:20,+0 560.000
  成都    昆明 CZ1016 09:45,+0 11:30,+0 01:45,+0 500.000
  成都    贵阳 CZ1017 18:50,+0 20:10,+0 01:20,+0 670.000
  昆明    成都 CZ1018 07:50,+0 09:25,+0 01:35,+0 500.000
  昆明    贵阳 CZ1019 20:15,+0 21:30,+0 01:15,+0 600.000
  贵阳    成都 CZ1020 22:35,+0 23:55,+0 01:20,+0 620.000
  贵阳    昆明 CZ1021 22:10,+0 23:25,+0 01:15,+0 510.000
```

**2．异常测试数据及运行结果**

显示所有线路：输出格式异常




**六．调试情况，设计技巧及体会**

## 1・改进方案

打印最小路径的方法过于冗长，不易修改，在 Debug 时屡屡出现困难，需要再进行优化，去除一些不必要的变量。由于只有三天的开发时间，完成的进度不令我满意，最小中转还没有写完，之后将添加上基于 BFS 的最小中转算法。

## 2・体会

对于常见的算法比如单源最短路经算法 Dijkstra 要有自己的理解，不能止于记住步骤，只有这样才可能高效地去开发并优化基于它的实际算法。写函数的时候注意“高内聚，低耦合”，尽可能写出易懂、出错后易修改的代码。注意给程序适当增加注释，方便之后的修改。

## 七・参考文献

[1] Thomas H.Cormen Charles E.Leiserson Ronald L.Rivest Clifford Stein 著. 《算法导论》. 第三版. P383 Dijkstra 算法 机械工业出版社.

[2] Stanley B.Lippman joseeLajoie Barbara E.Moo 著. 《C++ Primer》. 第五版. P373 关联容器 电子工业出版社.

[3]王曙燕 主编 王春梅 副编. 《数据结构与算法》. 第二版. P169 邻接表 P188 最短路径算法 人民邮电出版社.