2017 年复旦大学数据结构课程期末设计作业

交通咨询系统报告

院系：计算机科学技术学院

专业：保密管理

学号：16300240023

姓名：杨芗琳

1. 需求分析（每个模块的功能要求）

1.1 Main 函数

1.1.1.1 整合主要的函数

最主要功能由图中的函数实现，具体功能见 1.3

1.2 Guidance.h

1.2.1.1 界面设计

主要是 greeting 函数

```
***您好，欢迎来到这个精心设计的交通咨询系统！***
**我们承诺一定为您设计最符合您要求的出行路线！**
```

1.2.1.2 使用介绍

```
cout << "使用说明：" << endl;
cout << "1.输入（城市信息）对城市进行增加和删除操作" << endl;
cout << "2.输入（列车）对列车信息进行增加和删除操作" << endl;
cout << "3.输入（飞机）对飞机信息进行增加和删除操作" << endl;
cout << "4.输入（评估）获得最佳出行路线" << endl;
```

让用户明白操作方法

1.2.1.3 整合需要的功能

将主要的对城市的删减、列车飞机信息的删减等等功能整合在一起，实现循环输入，同时处理如果输入指令不正确的各种情况。

1.3 Graph.h

1.3.1.1 图的建立

一个包含所有信息的数据结构，可以记录城市名称、列车出发时间、列车到达时间、飞机出发时间、飞机到达时间、列车花费、飞机花费，通过邻接表实现

飞机和列车有两个图，分别记录

1.3.1.2 列车信息输入

将列车班次信息数据，出发地点、到达地点、时间、花费等信息记录到节点上

1.3.1.3 飞机信息输入

将飞机班次信息数据，出发地点、到达地点、时间、花费等信息记录到节点上

1.3.1.4 列车最短时间算法

将出发地点和到达地点之间最短时间走法记录下来，并输出

1.3.1.5 飞机最短时间算法

将出发地点和到达地点之间最短时间走法记录下来，并输出

1.3.1.6 列车最省钱算法

将出发地点和到达地点之间最省钱走法记录下来，并输出

1.3.1.7 飞机最省钱算法

将出发地点和到达地点之间最省钱走法记录下来，并输出

1.3.1.8 输出路线

将保存的路径和具体时间信息输出

1.3.1.9　　增加、删除城市信息

1.3.1.10　　增加、删除列车班次，飞机航班

2. 概要设计（每个部分的算法设计说明，流程图，数据结构说明）

2.1 城市信息输入、列车飞机班次输入

首先要建立一个图，将顶点放入。图采用邻接表表示，图里有一个 vector 放入每个城市的名称以及一个指向到达城市的指针 link。

把顶点放到图里之后，建立边的信息，每个边节点里有保存列车和飞机班次的时间以及到达城市的城市名称，还有一个指向下一个节点的指针 link。

完成后图的建立成功

2.2 最短时间算法

因为飞机和列车信息构建大致相同，故最短时间算法也是一样的。课堂中介绍了 Dijkstra、Floyd 等算法，但是这些算法都是基于每条路径都能走得通的情况下得到，在交通图里还要考虑到时间是否允许，即同时存在 1->2 和 2->3 的路径，但是如果时间不允许的话，这条路也走不通。所以在我的程序里，我采用了 DFS，先将每条可以到达目标城市的路径记录下来，然后判断时间上是否允许，如果这是一条可达路径，那么计算花费和时间，和当前最少花费和最短时间比较，如果该路径的时间比当前记录的最少时间还小，那么更新当前最短时间还有路径。遍历两个顶点之间所有的路径，找到最少时间的路径，输出。

这里用一个 vector 记录路径 path，还有（double）lowest_time 记录当前最短时间

2.3 最少花费算法

这里的算法和最少时间算法大致相同，在判断了一条路能走的通之后，和当前最少花费相比较，如果花费更少，就更新当前最少花费和路径

2.4 城市增加或删除

删除：只需要在图里 vector 加一个顶点或者删除一个顶点，同时如果有其他的边结点里有该城市，也删除该边结点

增加：NodeTable.push_back();

2.5 列车、飞机班次增加或删除

列车与飞机是一样的，这里只说一遍。

增加：找到出发城市的头节点，新建一个边结点 EdgeNode，加入到该头节点后面

删除：找到出发城市，然后找到含该到达城市的边结点，删除该节点

3. 详细设计（源程序，注释）

3.1 数据结构

图：

```
class Graph {
private:
    vector<VertexNode> NodeTable;
    int VerticesNum;//城市数量
public:
    Graph() { VerticesNum = 0; }
    bool create_city(istream &in);//必须考虑到如果输入的格式不对或者是多了城市
    char create_car_infomation(istream &in);
    char create_plane_infomation(istream &in);
```

```
    void lowest_car_cost(int v, int u);
    void lowest_car_time(int v, int u);
    void lowest_plane_cost(int v, int u);
    void lowest_plane_time(int v, int u);
    void car_printpath(int v, int u);
    void plane_printpath(int v, int u);
    int GetVertexPos(string name);
    void usage(istream &in,string command);
    void create_time(istream &in, double &i, double &j);
    void get_time(const double i, int &hour, int &minute);
    int GetFirstNeighbor(int v);
    int GetNextNeighbor(int v, int u);
    void access_car_cost(const vector<int> s, double &low, vector<int> &path);
    void access_car_time(const vector<int> s, double &low, vector<int> &path);
    void access_plane_cost(const vector<int> s, double &low, vector<int> &path);
    void access_plane_time(const vector<int> s, double &low, vector<int> &path);
    void car_printpath(const vector<int> s);
    void plane_printpath(const vector<int> s);
    void add_Vertices(string city);
    void erase_Vertices(string city);
    void access(string commute,istream &in);
};
```

头节点：

```
class VertexNode {
    friend class Graph;
    friend class EdgeNode;
private:
    string cityname;//城市名称
    EdgeNode *link;//出边表指针
};
```

边结点：

```
class EdgeNode {
    friend class Graph;
private:
    string cityname;  //到达城市的名称
    double car_start_time;//列车出发时间（出发的时刻，小时和分钟分别存储）
    double car_arrive_time;//列车到达时间
    double plane_start_time;//飞机出发时间
    double plane_arrive_time;//飞机到达时间
    double car_cost;//列车花费
    double plane_cost;//飞机花费
    EdgeNode *link;//边结点后继指针
public:
    EdgeNode() {//边界点初始化，-1代表没有列车或飞机到达该城市
```

```
        car_start_time = -1;
        car_arrive_time = -1;
        plane_start_time = -1;
        plane_arrive_time = -1;
        car_cost = 9999999.9;
        plane_cost = 99999999.9;
        link = NULL;
    }
    ~EdgeNode(){}
};
```

3.2 最短时间算法（以列车为例）

```
void Graph::lowest_car_time(int v, int u) {
    int *visited;
    visited = new int[Max];
    vector<int> path, s;//path用于保存当前最小花费时间的路径，s代表当前路径
    for (int i = 0; i < Max; i++) visited[i] = 0;
    s.push_back(v);
    visited[v] = 1;
    int w = -1;
    double lowest_time = 9999999;
    while (s.size()) {
        v = s.back();
        w = GetNextNeighbor(v, w);
        if (w == -1) {//表示下一个定点不存在，出栈
            w = v;
            s.pop_back();
            visited[v] = 0;
        }
        else if (w == u) {//如果发现了一条可达路径，那么计算该路径是否符合条件，
如果符合，则更新数据
            s.push_back(w);
            access_car_time(s, lowest_time, path);
            s.pop_back();
        }
        else if (!visited[w]) {//下一个顶点没访问过，入栈
            s.push_back(w);
            visited[w] = 1;
            w = -1;
        }
    }
    cout << "最省时间的路线是：";//输出路线
    car_printpath(path);
    cout << endl;
```

```
}
```

3.3 最小花费算法（以列车为例）

```
void Graph::lowest_car_cost(int v, int u) {
    int *visited;
    visited = new int[Max];
    vector<int> path, s;
    for (int i = 0; i < Max; i++) visited[i] = 0;
    s.push_back(v);
    visited[v] = 1;
    int w = -1;
    double lowest_cost = 9999999;
    while (s.size()) {
        v = s.back();
        w = GetNextNeighbor(v, w);
        if (w == -1) {
            w = v;
            s.pop_back();
            visited[v] = 0;
        }
        else if (w == u) {
            s.push_back(w);
            access_car_cost(s, lowest_cost, path);
            s.pop_back();
        }
        else if(!visited[w]){
            s.push_back(w);
            visited[w] = 1;
            w = -1;
        }
    }
    cout << "最省钱的路线是：";
    car_printpath(path);
    cout << "共花费了：" << lowest_cost;
    cout << endl;
};
```

3.4 建立图顶点

```
bool Graph::create_city(istream &in) {
    string city_name;
    vector<VertexNode>::size_type i;
    VertexNode p;
    while (in>>city_name) {
        for (i = 0; i < NodeTable.size(); i++) if (NodeTable[i].cityname ==
```

```
city_name) break;
        if (i == NodeTable.size()) {//判断输入几个城市，是否有重复
            p.cityname = city_name;
            NodeTable.push_back(p);
            VerticesNum++;
        }
    }
    return true;
}
```

3.5 建立班次信息（边结点输入，以列车为例）

```
char Graph::create_car_infomation(istream &in) {
    string start_city, arrive_city;
    double start_time, arrive_time;
    string is_no;
    vector<VertexNode>::size_type i, j;
    double cost;
    in >> start_city;
    if (start_city == "完成输入") return 'a';//判断完成输入
    in >> arrive_city;
    create_time(in,start_time,arrive_time);//时间以double形式存储
    in >> cost;
    for (i = 0; i < NodeTable.size(); i++) if (NodeTable[i].cityname ==
start_city) break;
    for (j = 0; j < NodeTable.size(); j++) if (NodeTable[j].cityname ==
arrive_city) break;
    //判断用户输入的城市是否存在于图顶点中，分情况讨论
    if (i == NodeTable.size()&&j!=NodeTable.size()) {
        cout << "该出发城市未出现在系统中，请确认是否加入该城市（是/否）：" <<
endl;
        in >> is_no;
        if (is_no == "否") return 'b';
        else {
            VertexNode q;
            q.cityname = start_city;
            VerticesNum++;
            NodeTable.push_back(q);
            EdgeNode *p;
            p = new EdgeNode;
            p->cityname = arrive_city;
            p->car_cost = cost;
            p->car_start_time = start_time;
            p->car_arrive_time = arrive_time;
            p->link = NodeTable[i].link;
```

```
                NodeTable[i].link = p;
                return 'c';
            }
        }
        if (j == NodeTable.size() && i!= NodeTable.size()) {
            cout << "该到达城市未出现在系统中，请确认是否加入该城市（是/否）：" <<
endl;
            in >> is_no;
            if (is_no == "否") return 'b';
            else {
                VertexNode q;
                q.cityname = arrive_city;
                VerticesNum++;
                NodeTable.push_back(q);
                EdgeNode *p;
                p = new EdgeNode;
                p->cityname = arrive_city;
                p->car_cost = cost;
                p->car_start_time = start_time;
                p->car_arrive_time = arrive_time;
                p->link = NodeTable[i].link;
                NodeTable[i].link = p;
                return 'e';
            }
        }
        if (j == NodeTable.size() && i == NodeTable.size()) {
            cout << "该出发城市和到达城市均未出现在系统中，请确认是否加入该城市（是/
否）：" << endl;
            in >> is_no;
            if (is_no == "否") return 'b';
            else {
                VertexNode q;
                q.cityname = start_city;
                VerticesNum++;
                NodeTable.push_back(q);
                q.cityname = arrive_city;
                VerticesNum++;
                NodeTable.push_back(q);
                EdgeNode *p;
                p = new EdgeNode;
                p->cityname = arrive_city;
                p->car_cost = cost;
                p->car_start_time = start_time;
                p->car_arrive_time = arrive_time;
```

```
                p->link = NodeTable[i].link;
                NodeTable[i].link = p;
                return 'f';
            }
        }
        EdgeNode *p;
        p = new EdgeNode;
        p->cityname = arrive_city;
        p->car_cost = cost;
        p->car_start_time = start_time;
        p->car_arrive_time = arrive_time;
        p->link = NodeTable[i].link;
        NodeTable[i].link = p;
        return 'd';

}
```

4. 调试分析

4.1 测试数据以及输出结果

测试图（以列车为例）

开始界面:

```
***您好，欢迎来到这个精心设计的交通咨询系统！***
**我们承诺一定为您设计最符合您要求的出行路线！**
```

输入图:

```
请输入城市（Ctrl+Z完成输入）：
北京 西安 厦门 重庆 上海 昆明
^Z
成功建立城市信息!
```

列车信息构建:

```
请按照（出发城市 到达城市 出发时间 到达时间 车票价格）输入列车信息:
/*输入（完成输入）结束构建列车信息*/
北京 西安 7:00 11:00 300
成功加入该班次!
北京 重庆 9:30 11:00 100
成功加入该班次!
北京 上海 7:05 12:10 200
成功加入该班次!
西安 昆明 12:00 18:00 800
成功加入该班次!
西安 厦门 10:00 19:00 100
成功加入该班次!
重庆 厦门 10:00 12:00 200
成功加入该班次!
上海 厦门 12:30 19:50 300
成功加入该班次!
厦门 昆明 20:00 23:00 400
成功加入该班次!
完成输入

完成构建列车信息!
```

最短时间方案和最省钱方案:

```
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 昆明 省钱 列车
最省钱的路线是：北京(07：04)->上海(12：09)        上海(12：30)->厦门(19：50)        厦门(20：00)->昆明(23：00)
共花费了：900
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 昆明 省时间 列车
最省时间的路线是：北京(07：00)->西安(11：00)      西安(12：00)->昆明(18：00)
```

增加城市、班次信息:

```
请输入操作：
城市信息
请输入（增加/删除）操作：
增加
请输入城市名称：
哈尔滨
加入成功！
请输入操作：
城市信息
请输入（增加/删除）操作：
增加
请输入城市名称：
沈阳
加入成功！
请输入操作：
列车
请输入（增加/删除）操作：
增加
请按照（出发城市 到达城市 出发时间 到达时间 车票价格）输入列车信息：
北京 沈阳 9:00 11:00 100
成功加入该班次！
请输入操作：
列车
请输入（增加/删除）操作：
增加
请按照（出发城市 到达城市 出发时间 到达时间 车票价格）输入列车信息：
沈阳 哈尔滨 12:00 14:00 100
成功加入该班次！
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 哈尔滨 省钱 列车
最省钱的路线是：北京(09：00)->沈阳(11：00)        沈阳(12：00)->哈尔滨(14：00)
共花费了：200
请输入操作：
```

删除城市:

```
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 昆明 省时间 列车
最省时间的路线是：北京(07：00)->西安(11：00)     西安(12：00)->昆明(18：00)

请输入操作：
城市信息
请输入（增加/删除）操作：
删除
请输入城市名称：
西安
成功删除城市！
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 昆明 省时间 列车
最省时间的路线是：北京(07：04)->上海(12：09)     上海(12：30)->厦门(19：50)       厦门(20：00)->昆明(23：00)
```

删除班次:

```
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 昆明 省时间 列车
最省时间的路线是：北京(07：00)->西安(11：00)     西安(12：00)->昆明(18：00)

请输入操作：
列车
请输入（增加/删除）操作：
删除
请按照（出发城市 到达城市）输入列车信息：
北京 西安
成功删除该班车！
请输入操作：
评估
请按照（起始城市 终点城市 最优决策原则 交通工具）输入要求：
北京 昆明 省时间 列车
最省时间的路线是：北京(07：04)->上海(12：09)     上海(12：30)->厦门(19：50)       厦门(20：00)->昆明(23：00)
```

额外的功能:

1）在（create_car_infomation）函数中如果输入格式不对，报错重输；如果多了的城市，多了出发城市，询问?

2）建立城市系统重复城市忽略

3）判断每次输入的内容是否合法（包括时间什么的）

4.2 时间复杂度分析

因为最短路径算法用栈来保存路径，故时间复杂度为 O(n).

4.3 出现问题及思考（有哪些问题？问题如何解决？）

4.3.1 时间记录

一开始我分别用两个数据记录时间，一个存小时一个存分钟，但是后来发现这样做会使空间复杂度提升而且计算起来数据繁杂容易出错。后来直接存在一个 double 的结构中，简化计算

```
int car_start_hour, car_start_minute;//列车出发时间（出发的时刻，小时和分钟分别存储）
int car_arrive_hour, car_arrive_minute;//列车到达时间
int plane_start_hour, plane_start_minute;//飞机出发时间
int plane_arrive_hour, plane_arrive_minute;//飞机到达时间
```

4.3.2 输入时间格式

对于 10：00 这样的时间输入方式，中间的冒号再程序中我用 char 表示，后再输入的时候在中文的环境下输入就会报错。后来发现中文的冒号不是 char 类型的，必须在英文的环境下才能正确读入。

4.3.3 错误调试

经常出现 cout 不明确，原因是前面函数的括号少了或者多了。

```
//开始的界面
void guidance_greeting() {
    cout << "***您好，欢迎来到这个精心设计的交通咨询系统！***" << endl;
    cout << "**我们承诺一定为您设计最符合您要求的出行路线！**" << endl;
    cout << endl;
}

//初始建立整个系统
void guidance_create_system(istream &in,Graph &city_system) {//界面美观问题
//城市信息
    bool istrue;
    cout << "请输入城市（Ctrl+Z完成输入）：" << endl;
    istrue = city_system.create_city(in);
    while (!istrue) {
        cout << "未成功建立城市信息！" << endl;
        in.clear();
        istrue = city_system.create_city(in);
    }
    cout << "成功建立城市信息！" << endl;
    in.clear();
    cout << endl;
//列车时间表
    cout << "请按照（出发城市 到达城市 出发时间 到达时间 车票价格）输入列车信息:" << endl;
    cout << "/*输入（完成输入）结束构建列车信息*/" << endl;
```

4.3.4 图的建立

因为飞机和列车是两个不同的系统，所以应该有两个图，一开始我只建了一个图，导致飞机和列车不能同时评估。所以后来我用了两个图，分别记录信息

4.4 改进思路

4.4.1 平行边

这个算法还不能满足平行边，即如果两个城市之间有两辆以上的班次，则无法判断。还可以再改进一下，邻接表里出现两个以上相同的节点也遍历一遍，可以实现。

4.4.2 继承

因为飞机和列车是类似的，所以可以采用继承的写法减小复杂度。

4.4.3 表头节点

在邻接表里，如果采用了表头节点，可以减少很多判断的工作，有助于简化程序

5. 课程设计总结

5.1 收获

这次建立这个交通咨询系统，从无到有，体验了编程的乐趣，特别是看到输出结果是正确的时候，非常有成就感。这次的编程体验让我更加喜欢这一学科。

还有知识方面，之前对图有很多不理解的地方，在构建图的过程中，加深了对图的理解，同时对课堂上讲的几个算法也有了更深的理解。

5.2 思考

本来在找最少时间算法的时候，我想用课堂上介绍的 Dijkstra 算法，但是后来发现有很多问题，Dijkstra 算法没有办法判断时间上允许的最小路径，于是 DFS 算法进入我的脑海中。这个算法基于穷举，将所有可能的情况都记录在内，最终选出最合适的一条路径。

在思考怎么得到最短路径问题时，我对图的知识也有了一个更深刻的体会。

我觉得图这一章节的内容就是如何用数据结构表示一个图，以及在不同的数据结构下如何对图进行遍历，后面的最短路径算法也是基于在图的遍历上的。最后的拓扑排序也是堆土的遍历。

我觉得图的 DFS 算法和树的前序遍历本质上是一样的，进而，如果要将图和树进行比较，有很多相似的地方。树其实就是一个没有圈的图，树可以算是图的一个分支，有自己的很多特点。

5.3 对编程调试的体会

在写代码的过程中出现了很多错误，有时候没有输出，有时候输出错误。在 VS 的环境下，很多错误会直接提示，可以直接改掉，但是更多时候错误找不到。

我觉得调试可以在代码中 cout 变量，这样可以看到过程，体会到错在哪里。还有就是函数应该边写边调试，而不要等到最后综合起来，这个时候错误就很难找了。

5.4 对课程的体会

这门课循序渐进，我也慢慢体会到数据结构对于解决问题的重要性。编程就是用编程语言将自己的算法表达出来，往往借助于一些有用的数据结构可以大大减少时间复杂度或者是空间复杂度。学好这门课还需要耐心和恒心，慢慢体会编程过程中的乐趣才能学好这门课。

6. 参考资料（论文书籍网站等）

Dijkstra

http://blog.csdn.net/qq_35644234/article/details/60870719

Floyd

http://blog.csdn.net/qq_35644234/article/details/60875818

DFS

http://blog.csdn.net/lysc_forever/article/details/17500959