# 数据结构说明和用户字典

## 数据结构定义

**1、旅行计划表结构：**

typedef struct{

int mean;//交通方式

int money;//该车次所需价格

int StartTime;//该车次起始时间

int EndTime;//该车次到达时间

int Time;//该车运行时间

}TIME;

**2、城市结点结构：**

typedef struct {

TIME Time[30];//一天最多一条路线十趟车，包含起始时间与到达时间

int i;//路线个数

}Path;

**3、交通图数据结构：**

typedef struct {

Path P[NN][NN];

}Graph;

**4、交通图边的结构：**

typedef pair <int,TIME> Road;

**5、旅行路线结构：**

typedef struct{

int z;

string s;

vector<Road> r;

}Travel;

## 配置文件定义

1. **时间表（航班表）：**

从左到右数据依次表示：

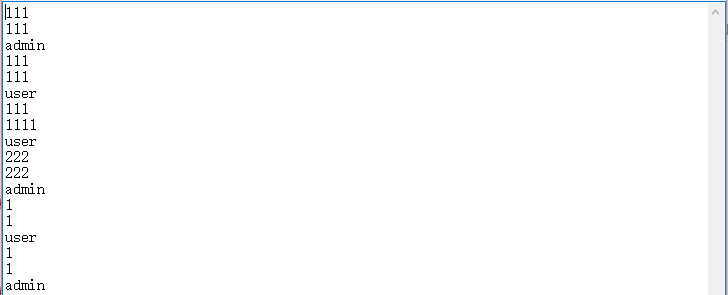
起始城市、终点城市、路线价格、出发时间、到达时间

（为了简便表示，我们在读取数据时用字母代替城市）

哈尔滨（A） 北京（B） 上海（C) 广州（D） 武汉（E) 成都（F） 拉萨（G） 乌鲁木齐（H） 西宁（I） 福州（J） 重庆（K）

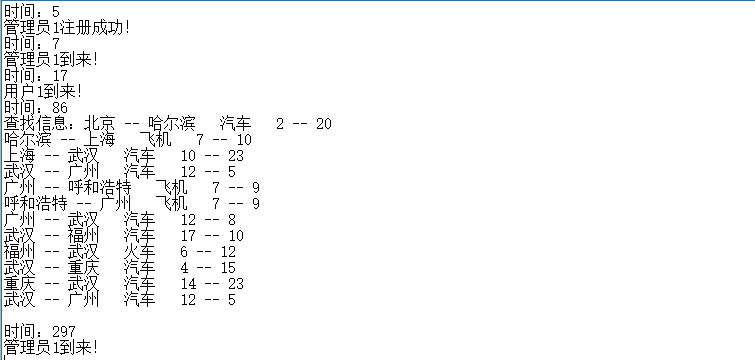


1. **密码文本（航班表）：**



第一行代表数据库中存入的用户信息的账号，第二行是账户密码，第三行是用户的身份信息，user代表是普通旅客，admin代表是管理员。

1. **日志文件：**



日志文件中包括事件发生的系统时间，余下一行是时间的类型，比如截图所示有“用户到来”、“管理员到来”、“用户注册”、“管理员注册”以及输出旅客查询的信息输出等。

## 接口函数以及类定义

Jmlog.h

void searchlog(string a,string b,int c);

void jmclear();

void usercome(string a,int c,int b);

add.h:

class add;

}

class add : public QDialog

{

Q\_OBJECT

public:

explicit add(QWidget \*parent = 0);

~*add*();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::add \*ui;

};

Administrators.h:

namespace Ui {

class administrators;

}

class administrators : public QDialog

{

Q\_OBJECT

public:

explicit administrators(QWidget \*parent = 0);

~*administrators*();

void MIMA(QString,QString);

void Open();

private slots:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

private:

Ui::administrators \*ui;

};

Chenggong.h:

namespace Ui {

class chenggong;

}

class chenggong : public QDialog

{

Q\_OBJECT

public:

explicit chenggong(QWidget \*parent = 0);

~*chenggong*();

private:

Ui::chenggong \*ui;

};

Global.h:

extern Graph g;

extern vector<Travel> TR;

Graph.h:

typedef struct{

int mean;//交通方式

int money;//该车次所需价格

int StartTime;//该车次起始时间

int EndTime;//该车次到达时间

int Time;//该车运行时间

}TIME;

typedef struct {

TIME Time[30];//一天最多一条路线十趟车，包含起始时间与到达时间

int i;//路线个数

}Path;

typedef struct {

Path P[NN][NN];

}Graph;

typedef pair <int,TIME> Road;

typedef pair <int,string> TB;

typedef pair <TB,vector<Road> > Travel;

Init.h:

int ToInt(char s[10]);

void Print(Graph g);

void Add(void);

Graph Read(void);

Login.h:

namespace Ui {

class login;

}

class login : public QDialog

{

Q\_OBJECT

public:

explicit login(QWidget \*parent = 0);

~*login*();

protected:

void *timerEvent*(QTimerEvent \* event);

private slots:

void on\_denglu\_clicked();

void on\_close\_clicked();

void on\_zhuce\_clicked();

private:

Ui::login \*ui;

int id1;

};

Luxian.h:

namespace Ui {

class luxian;

}

class luxian : public QDialog

{

Q\_OBJECT

public:

explicit luxian(QWidget \*parent = 0);

~*luxian*();

void Print(QString s);

private:

Ui::luxian \*ui;

};

Plan.h:

extern int t;

extern int t1;

bool fn(TIME a, TIME b);

int minDistance(int dist[], bool sptSet[]);

int Low\_T(vector<Road> line);

TIME my\_min(Path p, int preT);

TIME my\_min\_lim(Path p, int preT, int limit);

vector<Road> dijkstraCost(Graph graph, int src, int dst);

Travel pivot\_inT(Graph graph, vector<int>p, int T,string &s);

vector<Road> dijkstraTime(Graph graph, int src, int dst, int T);

vector<Road> dijkstraCost\_T(Graph graph,int src,int dst,int T,int L);

Travel pivot\_inW(Graph graph, vector<int>p,string &s);

Travel pivot\_in\_limit(Graph graph,vector<int>p,int T,int L,string&s);

string Changecity(int a);

string Changetime(int m);

int Backcity(QString s);

int Backtime(QString s);

Search.h:

namespace Ui {

class searcha;

}

extern int t;

class searcha : public QDialog

{

Q\_OBJECT

public:

explicit searcha(QWidget \*parent = 0);

~*searcha*();

private slots:

void on\_pushButton\_2\_clicked();

void on\_pushButton\_clicked();

void on\_pushButton\_3\_clicked();

private:

Ui::searcha \*ui;

};

Users.h:

namespace Ui {

class Users;

}

class Users : public QDialog

{

Q\_OBJECT

public:

explicit Users(QWidget \*parent = 0);

~*Users*();

void MIMA(QString,QString);

private slots:

void on\_pushButton\_clicked();

void on\_queding\_clicked();

protected:

void *timerEvent*(QTimerEvent \*event);

private:

Ui::Users \*ui;

int id1;

};

Watch.h:

extern int t;//系统时间

extern string name;//用户名

namespace Ui {

class watch;

}

class watch : public QDialog

{

Q\_OBJECT

public:

explicit watch(QWidget \*parent = 0);

~*watch*();

protected:

void *timerEvent*(QTimerEvent \* event);

void *paintEvent*(QPaintEvent \* event);

void *mousePressEvent*(QMouseEvent \*e);

private:

Ui::watch \*ui;

int id2;

QPixmap \*pix;

};

void coordinate(int t,int &m,int &n);

void change(int a,int &m,int &n);

Wrong.h:

namespace Ui {

class wrong;

}

class wrong : public QDialog

{

Q\_OBJECT

public:

explicit wrong(QWidget \*parent = 0);

~*wrong*();

private:

Ui::wrong \*ui;

};

Zhanghu.h:

extern int t;

void write(string a,string b,string c);

int judge(string a,string b,string c);

Zhuce.h:

namespace Ui {

class zhuce;

}

class zhuce : public QDialog

{

Q\_OBJECT

public:

explicit zhuce(QWidget \*parent = 0);

~*zhuce*();

private:

Ui::zhuce \*ui;

};

## 核心算法描述

1. 费用最少策略：

以费用为权值，使用dijkstra算法寻找从出发点到其余点的最短路径，并在寻找过程中保存每一点的前驱，寻找完成后，通过前驱记录，构建出由源点到终点的路径。

1. 时间最少策略：

以时间为权值，使用dijkstra算法寻找从出发点到其余点的最短路径，并在寻找过程中保存每一点的前驱，寻找完成后，通过前驱记录，构建出由源点到终点的路径。

1. 限时费用最少：

基于时间最短策略求得的最短时间T，先对限制时间t进行判断，若满足T <= t，则以时间和费用为权值，分段寻找最短短路径，否则给出给出时间最少策略的路径。