COVID-19 疫情环境下低风险旅行查询模拟系统

数据结构说明和数据字典

姓　　名：刘立敏

学　　号：2018211398

学　　院：计算机学院

专　　业：计算机类

班　　级：2018211310

指导老师：张海旸

数据结构说明和数据字典

一、数据结构

本部分将列举系统中所有用到的主要的数据结构，包括时刻表，旅客计划，地图路线等等。该部分内容直接以代码示例和注释的方式列出，故与数据结构相关的内容不再在后面的数据字典部分重复列出。

1、时刻表（结构）

```
struct timetable
{
    QString route_seq;              //车次/航班号
    int origin;                     //起点
    int destination;                //终点
    int start_time;                 //出发时间
    int end_time;                   //到达时间
    int total_time;                 //历经时间
};
```

注 1: 由于出发时间 (start_time) 和到达时间 (end_time) 的取值范围均为 0-23，而实际的列车/航班的真正总历经时间可能跨天甚至超过 48 小时，直接用 end_time-start_time 显然不合理，故增加 total_time 字段。

注 2：QString 为 Qt 中的字符串基本类型，相当于标准 C++语言中的 string 类。

2、旅客计划（结构）

```
struct userplan
{
    //QString name;                 //用户名
    int origin;                     //出发地
    int destination;                //目的地
    strategy traveling_strategy;    //旅行策略
    int required_time;              //采用限时策略时要求的限制时间
};
```

注 1：strategy 为枚举类型，包含非限时和限时策略，具体如下

```
enum strategy { NON_TIME_LIMITED, TIME_LIMITED };
```

注 2: 开始设计时在该结构体中添加了用户名(name)字段，但实际中没有用到。

考虑到这个问题没有太大影响，并且也可以随时利用上这个字段，故未删去。

3、城市信息（结构）

```
struct cityinfo
{
    QString city_name;                    //城市名
    double city_risk_index;               //风险指数
};
```

4、地图（类）

```
class Map
{
private:
    int num_cities;     //城市数量
    int num_routes;     //所有的汽车、火车车次数，飞机航班数总和，统称为路线数

    typedef struct route           //路线
    {
        QString route_seq;         //车次/航班号
        int origin;                //起点
        int destination;           //终点
        int start_time;            //出发时间
        int end_time;              //到达时间
        int total_time;            //历经时间
        route* next;               //下一条路线
    }*Route;

    struct citylist                //城市列表
    {
        int name;                  //当前城市序号
        Route first_route;         //当前城市出发的第一条路线
    };

    citylist Citylist[MAX_CITIES];
```

```
public:
    //Map 类的方法将在“各模块设计说明”中详细展现，此处仅反映数据结构，故省略。
};
```

注：Map 类实际上相当于地图结构，且采用了邻接表方式存储数据。

其中 citylist 是顶点表结构，包含城市序号和该城市出发的第一条路线

route 是边表结构，包含了一条路线的所有信息（与时刻表结构完全相同）以及下一条路线（下一条边）

5、旅客状态表（结构）

该部分用于存储不同时刻旅客的状态和计划

```
struct travellers
{
    userplan plan;

    QString routelist;              //以字符串方式记录一个解决方案的所有班次
    bool go_or_arrive[5000]={0};    //用于记录是否有出发或到达事件发生
    QMap<QString,QString>ma;        //时间与事件的映射
    QMap<int,int>time_x;            //时间与旅客在地图上横坐标的映射
    QMap<int,int>time_y;            //时间与旅客在地图上纵坐标的映射
    QList<QString>list;

};
```

注 1：routelist 是用于记录一个解决方案的所有班次

例如，从北京到达广州，要先乘坐 A1 到达上海中转，再乘坐上海到广州的 T2 次列车。那么，我们称 A1 为一个 route（班次），T2 为另一个 route。

A1 T2 整体作为字符串被称为一个 routelist（解决方案），即一个解决方案包含了从起点到终点的所有班次。

注 2: QMap 和 QList 是 Qt 中的容器，对应于标准 C++语言的 map 类和 list 类。

二、数据字典

该部分主要对路线设计部分的算法作出说明，而与图形界面相关的部分由于变量等细节过于繁多且并非算法的核心内容故而省略。另外，诸如没有直接含义的循环计数变量（如 i，j，cnt 之类）的，同样在此省略。

路线设计部分主要位于两个文件：routedesign.cpp 和 routefunc.cpp。前者体现核心算法，后者体现辅助函数。

1、核心算法

①常量和全局变量

变量（常量）名	类型	作用及备注
inf	const double	常量，表示无穷大
MAX_TRANSITION_TIMES	const int	常量，表示最大中转次数（默认为 4），用于 DFS 的剪枝
city_info	cityinfo [ ]	数组，存储城市名和对应的风险系数
visited	static bool [ ]	静态，数组，用于记录 DFS 的访问
Timetable	timetable [ ]	数组，时刻表实例
cities	int	城市数量
routes	int	路线数量

②Map 类方法 Initialize：用于初始化地图

（返回类型 void，参数列表空）

变量（常量）名	类型	作用及备注
file，file2	QFile	Qt 中的文件类，用于绑定文件

fin，fin2	QTextStream	Qt 中的文本流，用于读写文件内容
R	Route	边表实例，这里的边是一个 route
num,from,to	QString	临时变量，分别表示从时刻表文件里读取出的班次序号，出发地，目的地
start,end,time	int	临时变量，分别表示从时刻表文件里读取出的出发时间，到达时间，历经时间

③Map 类方法 Extended_non_time_limited：扩展版本非限时最小风险策略算法

（返回类型 QString：一个解决方案，参数列表 userplan：旅客计划）

变量（常量）名	类型	作用及备注
src,des	int	保存旅客计划中起点和终点序号
total_risk	double [ ]	数组，记录从起点到各城市的总风险
arrive_time	int [ ]	数组，记录到达各城市的时间
path	int [ ]	数组，保存路径，记录路径中的上个城市
route_name	QString [ ]	数组，记录上个城市到该城市的车次
min_risk	double	最小风险值
min_risk_city	int	最小风险值对应的最小风险城市
trans_risk_index	double	交通工具上的风险系数
city_risk_index	double	城市的风险系数
stay_time	int	停留在一个城市的时间
tmp	double	临时变量，暂存当前路线上的总风险
route_list	QString	一个解决方案

④Map 类方法 DFS：深度优先搜索所有在规定中转次数内可能到达目的地的所有路径。

（返回类型 void；

参数列表：

参数名	参数类型	备注
str	QString	一个解决方案
src,des	int	出发城市序号，目的城市序号
hop	int &	引用，记录中转次数（跳数）
cnt	int &	引用，记录输出路线的条数

）

⑤Map 类方法 Extended_time_limited:扩展版本的限时最小风险策略算法

（返回类型 QString：一个解决方案，参数列表 userplan：旅客计划）

变量（常量）名	类型	作用及备注
src,des	int	出发城市序号，目的城市序号
hop	int	记录中转次数（跳数）
cnt	int	记录输出路线的条数
min_risk	double	最小风险值
min_risk_route	int	最小风险值对应的最小风险线路
min_risk_total_time	int	最小风险值对应的总历经时间
city_risk	double	城市风险系数
trans_risk	double	交通工具风险系数

timer	int	计时器
total_risk	double	风险累计值
arr_time	int	到达时间
stay_time	int	停留在一个城市的时间

⑥Map 类方法 Basic_non_time_limited:基础版本的非限时最小风险策略算法

（返回类型：QList<QString>routelists :同为风险值最小的路线列表

参数列表：userplan：旅客计划）

基本数据名称和用途同 Extended_time_limited，此外增加了一个 int 型变量 num 用于记录同为最小风险值的解决方案数量。

⑦Map 类方法 Basic_time_limited:基础版本的限时最小风险策略算法

（同上）

2、辅助函数

函数名	功能	参数列表	返回值
Get_seq	将城市名映射到序号	QString 类型的城市名	int 类型的序号
Get_name	将城市序号转换为城市名	int 类型的序号	QString 类型的城市名
Get_transport_risk_index	根据班次号首字母确定交通方式，并返回对应的风险值	char 类型的字符（班次号首字母）	double 类型的风险值

Get_route_info	以班次号为索引顺序查找时刻表	QString 类型的班次号和 timetable*类型的时刻表指针	timetable* 类型的时刻表指针
Get_path_list	提取出用 Dijkstra 算法得到的一个解决方案的车次列表	int*型路径，QString*类型的路线名，int 类型的目的城市序号和 QString&类型的解决方案	void
Get_path_info	得到最后要输出的解决方案	QString 类型的解决方案和 timetable*类型的时刻表指针	QString 类型的解决方案