开发者文档

**目录：**

1. **设计任务的描述**
2. **功能需求说明及分析**
3. **总体方案设计说明**
4. **数据结构说明和数据字典**
5. **各模块设计说明**
6. **范例执行结果及测试情况说明**
7. **评价和改进意见**
8. **设计任务的描述**

城市之间有各种交通工具（汽车、火车和飞机）相连，有些城市之间无法直达，需要途径中转城市。在当前COVID-19疫情环境下，各个城市的风险程度不一样，分为低风险、中风险和高风险三种。

对于人们的出行需求，需要一个系统根据风险评估，为该旅客设计一条符合旅行策略的旅行线路并输出；系统能查询当前时刻旅客所处的地点和状态（停留城市/所在交通工具）。

1. **功能需求说明及分析**

需求说明：

1. 城市总数不少于10个，为不同城市设置不同的单位时间风险值：低风险城市为0.2；中风险城市为0.5；高风险城市为0.9。各种不同的风险城市分布要比较均匀，个数均不得小于3个。旅客在某城市停留风险计算公式为：旅客在某城市停留的风险=该城市单位时间风险值\*停留时间。
2. 建立汽车、火车和飞机的时刻表（航班表），假设各种交通工具均为起点到终点的直达，中途无经停。也不考虑城市内换乘交通工具所需时间。
3. 旅客的要求包括：起点、终点和选择的低风险旅行策略。其中，低风险旅行策略包括：
   1. 最少风险策略：无时间限制，风险最少即可
   2. 限时最少风险策略：在规定的时间内风险最少
4. 可以查询当前时刻旅客所处的地点和状态（停留城市/所在交通工具）。
5. 旅行模拟系统以时间为轴向前推移，每10秒左右向前推进1个小时(查询状态不计时，即：有键盘输入时系统不计时)
6. 建立日志文件，对旅客状态变化和键入等信息进行记录。
7. 选作一：图形绘制地图并展示
8. 选作二：为不同交通工具设置不同单位时间风险值，旅客乘坐某班次交通工具的风险 = 该交通工具单位时间风险值\*该班次起点城市的单位风险值\*乘坐时间。

需求分析：

1. 城市数量在程序启动时确定，可以选择自行输入，也可以选择使用默认数据。
2. 程序启动时还会要求输入时刻表生成规则或选择默认规则，时刻表由系统根据规则每小时进行更新。
3. 旅客旅行计划的计算使用变形的迪杰特斯拉算法。限时与不限时两种旅行要求则通过设置“无限时间”实现。即不限时相当于限定无限时间，从而统一算法。
4. 通过按下键盘特定按钮进入或退出查询模式。
5. 时间推移通过设计模拟计时器实现，具体的推移间隔也可以由用户自行设置。
6. 日志文件分三个：“systemLog”——系统日志文件；“interruptLog”——中断日志文件；“otherLog”——其他日志文件。分别记录：每小时各旅客的状态变化；查询记录；生成的其他数据。
7. 选作一图形化未完成。
8. 选作二完成。同时为了泛用性，是否设置交通工具的风险值由用户决定。设置为0即认为不考虑乘坐交通工具时的风险值。
9. **总体方案设计说明**
10. 软件开发环境：visual studio 2019
11. 总体结构：

程序启动时，会请求用户输入地图信息（即各城市以及其交通规则）；还会请求输入 旅客的要求。之后经过初始化，进入主体循环。主体循环由两部分组成，一是计时器检测，当计时器超时，则认为经过了一个小时，随即转入超时处理函数，更新交通工具表（时刻表）以及更新旅客状态；二是键盘中断检测，检测到键盘输入后先停止计时，然后判断输入了何种命令，并转向相应的中断处理函数。

1. 模块划分：
   1. 初始化模块”initX.h”：定义了程序所需的各种数据结构，详见第四点“四、数据结构说明和数据字典”。
   2. 输入模块”inputX.cpp”及其头文件”inputX.h”：定义了如何获取城市数据以及旅客数据的函数。
   3. 输出模块”outputX.cpp”及其头文件”outputX.h”：定义了如何输出城市信息以及旅客旅行计划的函数。
   4. 日志模块”logX.cpp”及其头文件”logX.cpp”：定义了记录相关日志的函数以及部分格式化输出有关的转换函数。
   5. 路线设计模块”routeX.cpp”及其头文件”route.h”：通过对迪杰斯特拉算法的变形，实现适用于本设计任务的算法。同时还定义了相关辅助函数。
   6. 状态更新模块”updateX.cpp”及其头文件”update.h”：定义了每小时如何更新时刻表以及如何更新旅客状态。
   7. 中断模块”interrupt.cpp”及其头文件”interrupt.h”：定义了查询状态如何查询。
   8. 主模块”main.cpp”：在主模块完成初始化，定时器循环以及键盘中断检测。
2. **数据结构说明和数据字典**
3. 数据结构：
   1. 模拟定时器

typedef struct myTimer

{

char tName[20];//定时器名字

int tInterval;//定时器间隔

time\_t tLast;//定时器起始时间

void (\*pfunc)();//定时器超时处理函数

}MYTIMER;

* 1. 交通信息结构定义

typedef struct trans

{

char name[16];//名字

int type;//类型0-1-2-3:无-飞机-火车-汽车

int startCityID;//起始站ID

int startTime;//出发时间：小时

int endCityID;//终点站ID

int endTime;//到达时间：小时

int duration;//旅途耗时

float risk;//交通工具单位时间风险

int nflag;//屏蔽位，=1时屏蔽。主要用于路线设计时的回溯判断

}TRANS;

* 1. 旅客状态结构定义

typedef struct pass

{

char name[32];//名字

int status;//状态：0-无效 1-等待获得旅行方案 2-在城市中 3-在路上

int request[4];//要求：0Xab表示低风险a->b，1Xab表示限时X小时a->b

int position;//当前位置（在城市逗留或在交通工具上）

TRANS route[5];//换乘表，最多5趟

float risk;//截止到目前的累计风险值

struct pass \*next;//下一个乘客

}PASS;

* 1. 交通工具编排规则（即生成时刻表的方法）

typedef struct rule

{

int freq;//频率：多少小时一次

char name[16];//名字

int type;//类型1-2-3:飞机-火车-汽车

int startCityID;//起始站ID

int endCityID;//终点站ID

int duration;//旅途耗时

float risk;//交通工具单位时间风险

}RULE;

* 1. 城市结构定义

typedef struct city

{

char name[32];//城市名

float risk;//城市风险

RULE ruleTable[10];//此城市的交通工具编排规则

int transNum;//此城市当前时刻的时刻表表项数

TRANS timeTable[MAX\_TRANS];//城市1周内的时刻表

}CITY;

1. 数据字典（部分主要数据以及宏定义的说明）
   1. #define MAX\_PASS 5 //最大乘客数
   2. #define MAX\_CITY 20 //最大城市数
   3. #define MAX\_TRANS 1000 //每个城市最大航班数
   4. #define INFINITE 65535//无穷
   5. PASS \*passengers;//所有旅客存为一个链表，这是旅客链表头
   6. CITY citys[MAX\_CITY];//城市信息数组
   7. int activePass; //活跃乘客数
   8. int global\_hour; //系统时间
   9. int cityNum;//城市数量
   10. char systemLogPath[64];//系统日志文件路径（相对路径）
   11. char interruptLogPath[64];//中断日志文件路径（相对路径）
   12. char otherLogPath[64];//其他日志文件路径（相对路径）
   13. MYTIMER sTimer;//全局定时器，定时间隔可变
2. 函数说明：
   1. void getData();//控制台键盘输入城市信息
   2. void getDataDefault();//使用默认城市数据
   3. void getOrder();//控制台键盘输入旅客信息
   4. void getOrderDefault();//使用默认旅客数据
   5. void inquire();//查询旅客信息void GenerateCityLog();//程序开始时生成城市信息存入日志文件
   6. void timeChange(int\* date, int hours);//将系统时间转化为月-日-小时的形式
   7. void outputRoute(PASS\* tmpPass);//输出生成的乘客的旅行计划到控制台和日志文件
   8. void findRoute(PASS\* tmpPass);//为乘客tmpPass计算旅行计划
   9. void updateTimetable();//每小时更新所有城市的交通表
   10. PASS\* updatePassStage(PASS\* prePass, PASS\* tmpPass);//更新tmpPass的状态，其中prePass起辅助作用
3. **各模块设计说明**
4. 各模块算法思想：
   1. 初始化模块”initX.h”：定义了程序所需的各种数据结构，已在第四点“四、数据结构说明和数据字典”进行详细说明。
   2. 输入模块”inputX.cpp”及其头文件”inputX.h”：使用控制台输入流获取城市数据以及旅客数据。
   3. 输出模块”outputX.cpp”及其头文件”outputX.h”：使用控制台输出流输出城市信息以及旅客旅行计划。
   4. 日志模块”logX.cpp”及其头文件”logX.cpp”：使用C语言提供的文件操作函数生成日志文件。

其中包含的日期转换函数采用简单的换算方式，即认为一年有12个月，一个月有30天，每天24小时。

* 1. 路线设计模块”routeX.cpp”及其头文件”route.h”：

首先是迪杰特斯拉算法的变形”findRoute”函数：从起始城市开始，每次寻找到一个已求出小风险路径的城市，并以此城市为起点，在加上到达此城市所需的时间偏移量后，更新计算图的各边权值。循环以上步骤直至到达目的城市。如果路径所花时间超过了旅客要求的时间限制，则逐步原路返回，每返回一步就重新计算是否有不超时的路径。

其次是权值更新函数”generateG”：以新找到的城市为主要参数，便利此城市的时刻表，计算直接到达各城市的最小风险值，并将信息存入相关数组。对于可能存在的超时路径，则通过每个交通信息中的”nflag”标志位决定是否参与计算。

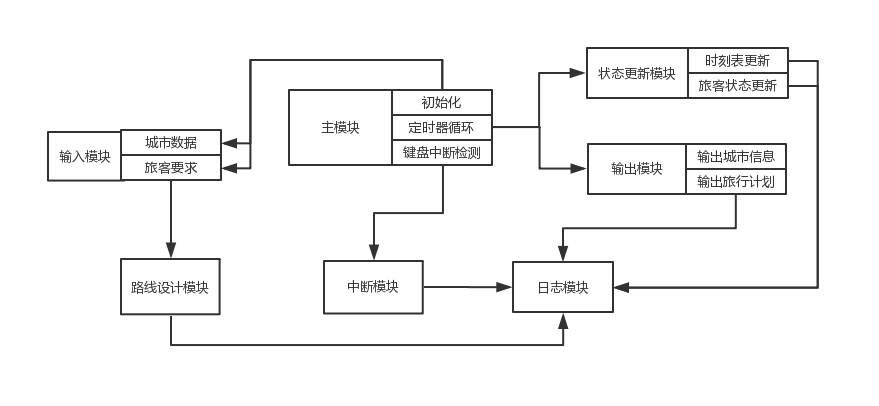
* 1. 状态更新模块”updateX.cpp”及其头文件”update.h”：

首先是时刻表更新函数：遍历每个城市的交通工具编排规则，生成未来一周内的所有交通。

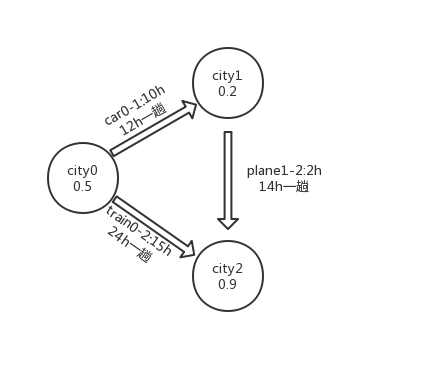
其次是旅客状态更新函数：先检测旅客当前位置，在城市里还是在交通工具上，再判断下一步要进行的操作，是下车还是上车还是等待。

* 1. 中断模块”interrupt.cpp”及其头文件”interrupt.h”：通过详细的引导，指导用户输入要查询的信息。
  2. 主模块”main.cpp”：调用上述各模块完成初始化；同时负责定时器循环以及键盘中断检测。

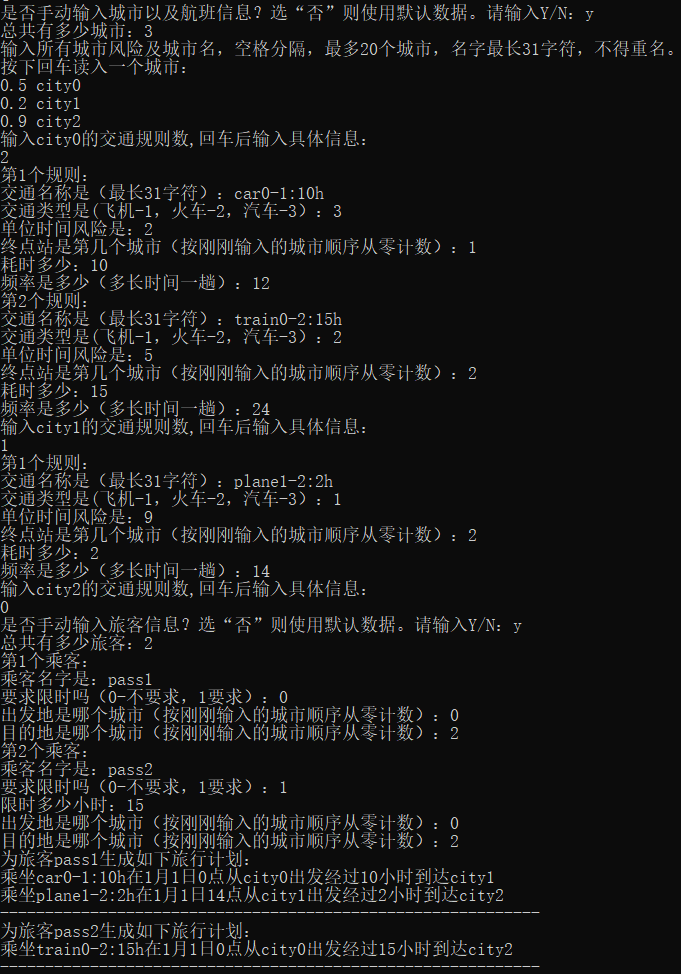
1. 与其它模块的关系：



1. **范例执行结果及测试情况说明**
2. 测试一，手动输入城市信息与旅客信息

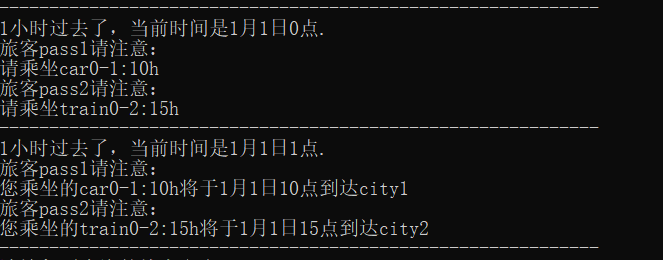


由于手动输入较慢，所以测试城市为3个，仅作功能演示。按上图所示拓扑关系输入：

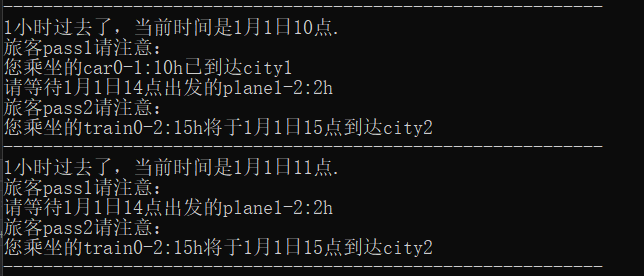


由输出可得，分别对两个乘客给出了相应的旅行计划：乘客”pass1”只要求风险最小，故给出city0->city1->city2的方案；”pass2”要求限时15h，而最小风险city0->city1->city2的方案超时，故选择city0->city2的方案。

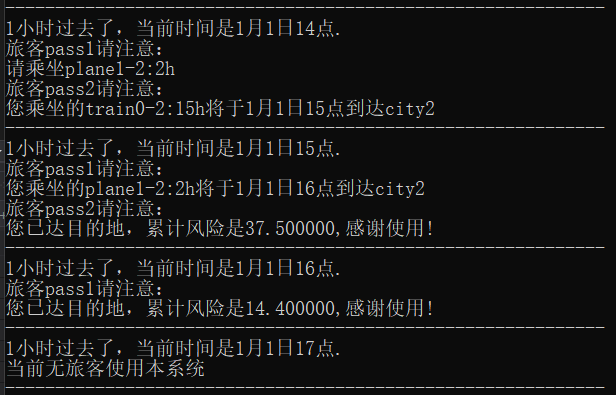
后续部分输出以及结果如下：



......

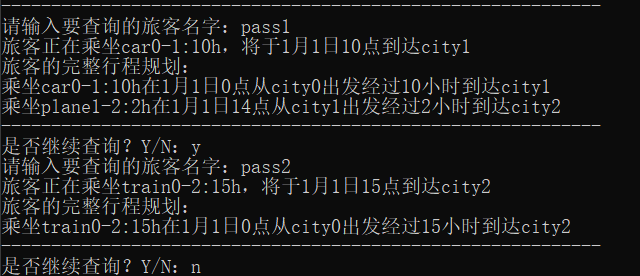


......



经计算pass1的旅行方案累计风险为0.5\*2\*10+0.2\*4+0.2\*9\*2=14.4；pass2的旅行方案累计风险为0.5\*5\*15=37.5。与输出结果相符，同时时满足限时条件的最小风险。

此外，还测试了查询旅客信息：也正确输出了结果。

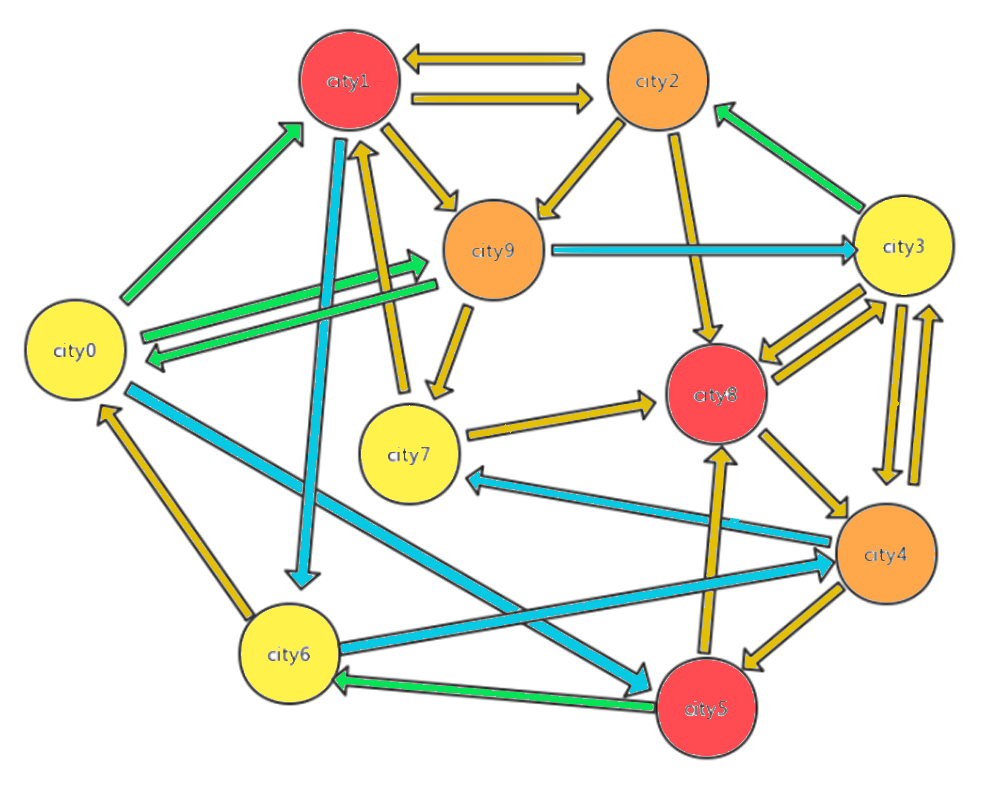


测试一通过。

1. 测试二，使用默认城市信息与默认旅客信息

默认城市信息内置了10个城市，每时刻存在的航班数上限为：5趟飞机，7趟火车，15趟汽车。

各城市拓扑图如下：

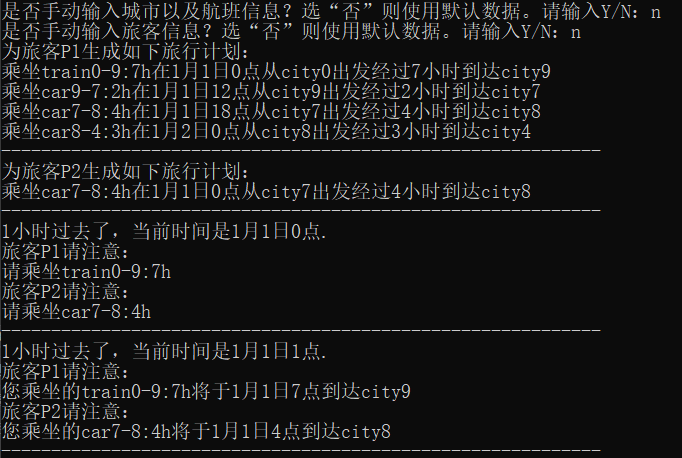


红色城市为高风险，橙色城市为中风险，黄色城市为低风险。

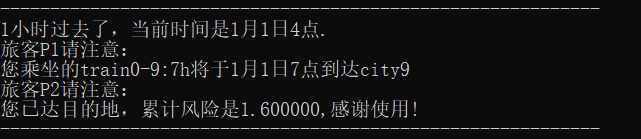
蓝色箭头代表飞机，绿色箭头代表火车，黄色箭头代表汽车。

具体航班信息见生成的系统日志文件。

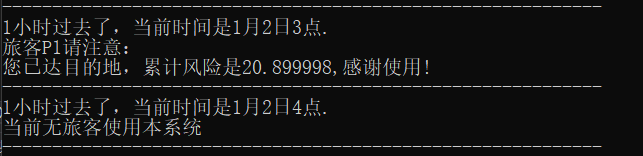
程序运行截图：



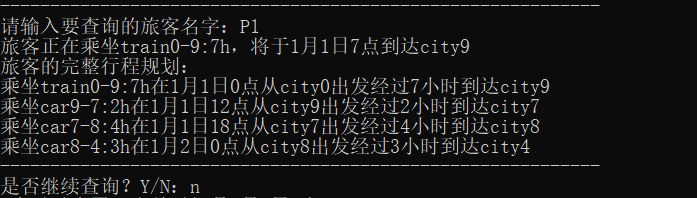
......



......



查询功能测试：



1. **评价和改进意见**
2. 评价
   1. 软件输入提示详尽，易于操作。
   2. 输出信息简洁明了，包含主要信息。

每个小时都会提示每个旅客接下来应该注意的事情，例如还有多长时间到达本趟交通工具的终点站；什么时候应该乘坐下一趟交通工具；以及到达最终目的地时的提示。

* 1. 查询功能方便易用。

运行期间可随时按下键盘进入查询模式；查询模式设置了输入合法性检查；还可以连续查询并随时退出查询模式。

* 1. 时间推进准确。

计时器采用系统时钟的数据模拟而成，计时精度良好，还可根据不同需求更改定时间隔。

* 1. 日志文件分类明确。

区分系统日志和中断日志，便于日后有条理地分类查看程序运行过程。

1. 修改意见
   1. 程序初始化存在大量输入，但输入合法性检查并未全部覆盖，不按规定输入可能导致程序异常。如果输入错误需要重新启动。
   2. 程序容错性较低。若用户输入错误的数据或程序内部计算时出现问题，容易导致程序内存溢出进而崩溃。
   3. 模块划分耦合性较高

设计初期未考虑周全，导致各模块依赖性较强。同时有些功能分散于程序各处，无法整合为一个合适的函数去处理。例如生成系统日志的功能和更新旅客状态的功能高度依赖，所以只能在更新状态的同时随之生成日志。

* 1. 算法效率与容错性还需优化

在限时最小风险策略中，涉及到城市的回溯过程，有时要多次回溯，现有的算法按时刻表依次尝试并判断回溯的方式效率就较为低下；同时，若时刻表过于庞大，可能导致回溯次数过多而计算失败。

* 1. 用户界面美观性仍可改善

本程序没有使用图形化界面，所有操作与信息展示全部由命令行实现。尽管信息的排布方式经过设计，达到了有条理，但界面美观度与直观程仍度不是很理想。