# 设计任务的描述

任务的目标是为旅客设计一个COVID-19疫情环境下低风险旅行模拟系统。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，城市之间有汽车、火车和飞机三种交通工具相连，有些城市之间无法直达，需要途径中转城市。系统将评估风险，根据旅客需求输出一条线路，同时可查询当前时刻旅客所处的地点和状态。

# 功能需求说明及分析

* **需求分析**

1. 数据的合理性。城市的总数不得少于10个且各个风险等级的城市不得少于3个；建立的汽车、火车和飞机的时刻表和班次要贴近实际情况。为了简化问题，假设各种交通工具均为起点到终点的直达。
2. 策略的最优性。根据旅客的要求，系统能通过一定算法得出最少风险策略或限时最少风险策略。
3. 风险的量化。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。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。在考虑城市风险的情况下，得到线路的风险值。为不同交通工具设置不同单位时间风险值，交通工具单位时间风险值分别为：汽车=2；火车=5；飞机=9。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。将乘坐交通工具的风险考虑进来，实现前述最少风险策略和限时风险最少策略。
4. 输出的清晰性。能模拟时间推移，随时查询旅客状态，每10秒左右向前推进1个小时(非查询状态的请求不计时，即：有鼠标和键盘输入时系统不计时)。建立日志文件，对旅客状态变化和键入等信息进行记录。
5. 地图的绘制。能在地图上实时反映出旅客的旅行过程。

* **功能简介**

1. 能够根据旅客提供的起点和终点，找到限时与不限时情况下的最低风险策略。
2. 能够根据旅客提供的起点和终点，找到考虑城市停留风险的最低风险策略。
3. 能够在策略选定之后，自主选择是否考虑交通工具风险。
4. 输出的策略详细包含要经过的城市和乘坐的交通工具班次号，以及所要花费的时间。
5. 能够模拟旅客所选策略的时间推进过程，并且可以随时查询当前时刻旅客的状态、地点。
6. 能够直观地将路线及旅程展现在地图上。

# 总体方案设计说明

## 软件开发环境

Visual Studio 2017

## 总体结构

程序的总体结构是按照用户输入要求、程序输出结果的思想来设计的，包含三个状态：未出行状态、线路选择状态、出行状态。

1. 未出行状态：程序的初始状态。此时程序界面提示用户可选择不限时的最低风险出行策略或限时的最低风险出行策略。
2. 线路选择状态：出行前确定线路的状态。用户根据程序的提示输入出发时间、出发地、目的地，如果是限时策略还需输入限制时长，输入完毕后，程序内部运行算法将符合要求的最低风险路线输出。
3. 出行状态：按照最优路线出行的状态。程序会模拟旅客出行，时间和路线都将显示在界面上。系统时钟每10s推进一小时，每个时刻旅客的状态都会被记录在日志文件中，用户可以随时查询当前时刻所在的地点、交通工具等，查询时系统时钟暂停推进。

三个状态依次转换，出行状态结束后可转换回未出行状态，旅客编号加一。

## 模块划分

程序模块可由三大状态继续分为主模块、时刻表置入模块、线路生成模块、时钟推进输出模块。

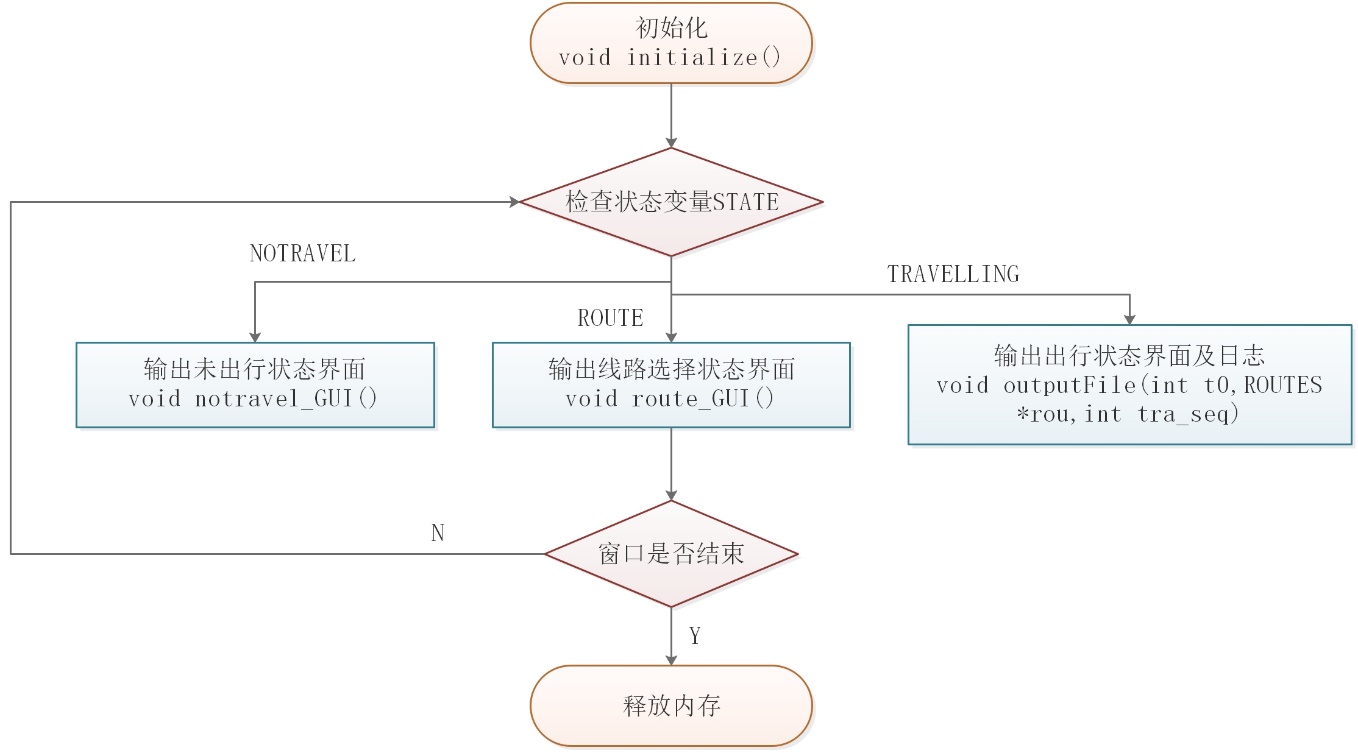
1. 主模块：程序流程推进的主要模块。在调用时刻表置入模块后，使程序进入未出行状态。程序运行期间，根据当前时刻状态变量STATE的值，将程序转换为相应状态。
2. 时刻表置入模块：读取“city.txt”文件，将所包含的交通工具时刻表转换为各个变量的值，存储在程序中。
3. 线路生成模块：用户选择不限时或限时最低风险策略时调用，将生成并输出符合要求的旅行线路。
4. 时钟推进输出模块：系统时钟每10s推进一小时，更新旅客当前时刻的旅行状态，并输出到“journal.txt”文件里，用户可以随时查询当前时刻所在的地点、交通工具等，查询时系统时钟暂停推进。

## 各模块详细说明

下面介绍的算法均为每个模块各自的核心算法思想，其他过程就省略了，实际每个模块具体实现以及各个状态转换的时间点不再赘述。

1. **主模块**

* 算法流程图

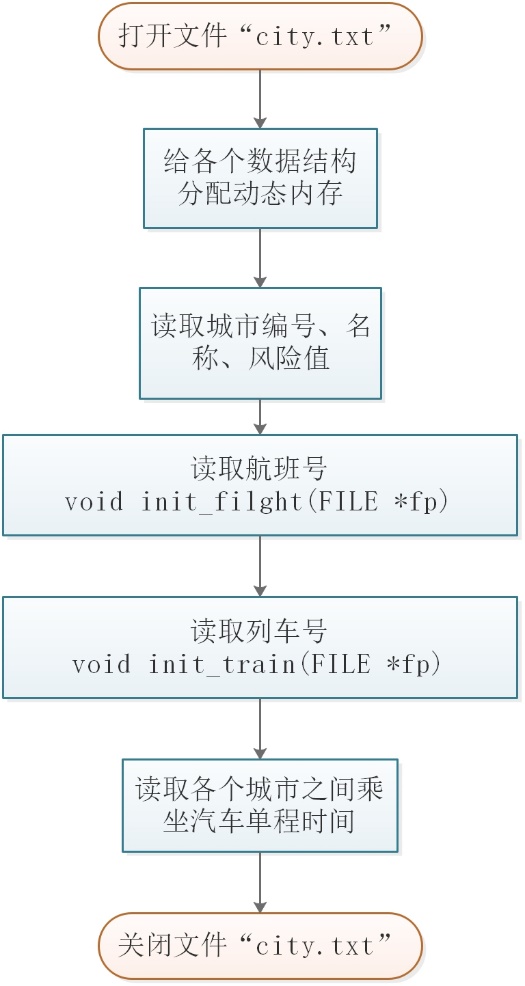
****

* 算法思想及特点

主模块的作用即为按照逻辑调用其他模块，推进程序运行。因此应用状态转换自动机的思想，设置全局变量STATE，在各个模块内部完成任务时改变STATE的值使其指向下一状态，而主模块只需要判断当前STATE的值，并推动程序进入相应状态执行操作即可。

1. **时刻表置入模块**

* 算法流程图

****

* 算法思想及特点

“city.txt”文件的数据格式为：（假设城市总数为n）

城市数量 航班数量 列车数量

城市序号 城市名 城市风险值

……

航班号 航班时长 每天的航班数量

出发时刻-到达时刻 出发城市序号 到达城市序号

出发时刻-到达时刻 出发城市序号 到达城市序号

……

……

列车序号

列车时长 每天的班次数量

出发时刻-到达时刻 出发城市序号 到达城市序号

出发时刻-到达时刻 出发城市序号 到达城市序号

……

……

城市1与城市n 城市2与城市n …… 城市n-1与城市n

汽车单程时间 汽车单程时间 汽车单程时间

城市1与城市n-1 …… 城市n-2与城市n-1

汽车单程时间 汽车单程时间

…… ……

城市1与城市2

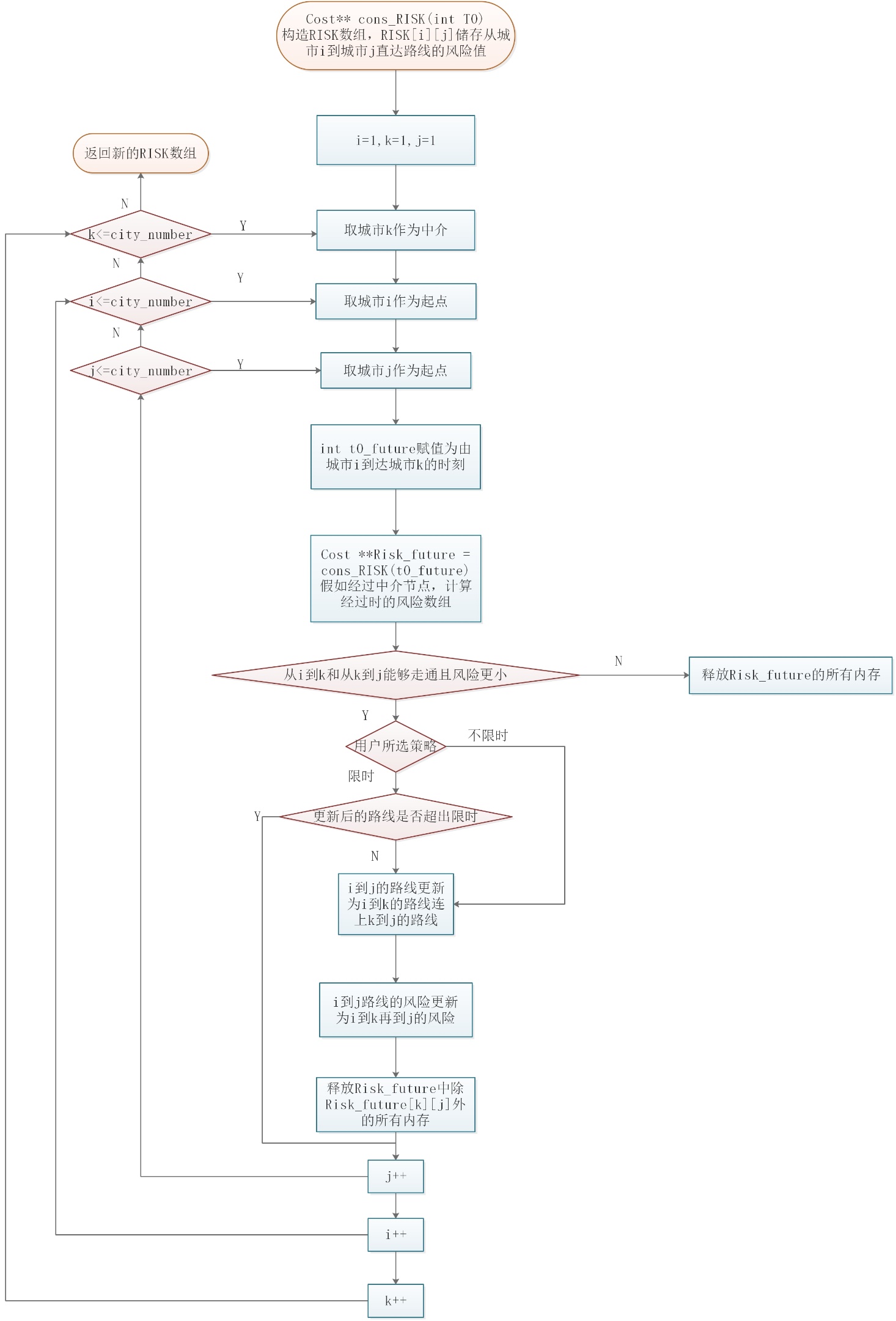
汽车单程时间

依据现实情况可知，汽车往往班次较多，因此文件中不再给出汽车的时刻表，程序的时间单位为小时，因此若两地乘汽车可达，则每小时都有一班汽车，若两地不可达，则其汽车单程时间为-1。

该模块依据此数据文件的格式依次读入所有城市与交通工具的信息。数据格式明确精简，读入较为便捷，无须复杂的算法。

1. **线路生成模块**

* 算法流程图



* 算法思想及特点

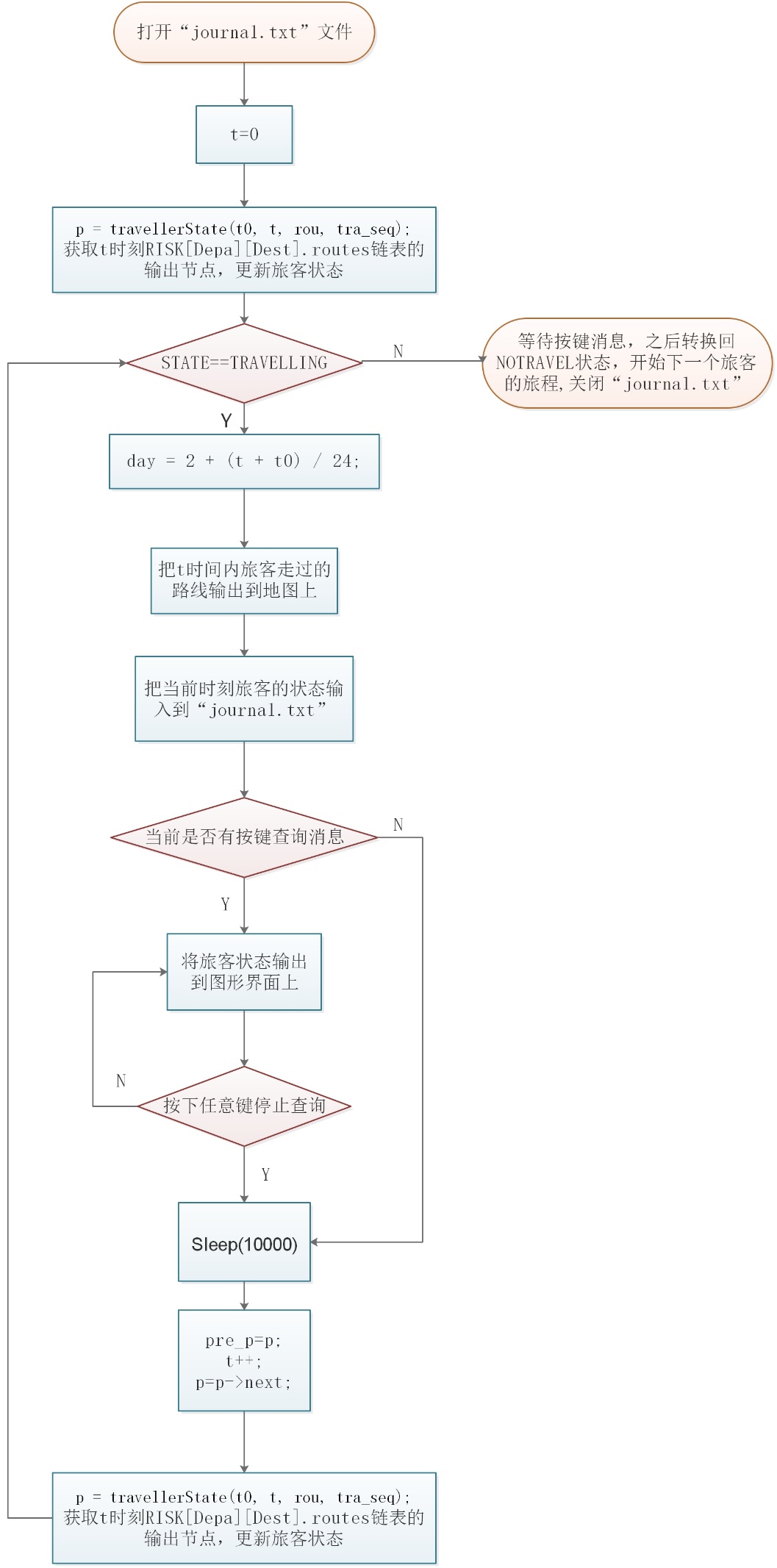
考虑到每一时刻出发路线以及路线的风险值都会不同，我基于弗洛伊德算法思想，对最低风险RISK数组的构造过程进行了变形。RISK初始态即为各个地点之间的直达或不可达（风险值为MAXRISK），在判断途径其他节点能否使当前旅程的风险值更小时，计算到达中介的时刻，再次构造那一时刻的直达风险数组Risk\_future，从而求得中介节点到终点间的风险值。

这种做法存在两个问题：一是虽然保证了结果的最优性，但空间和时间上的开销过大，二是得出的RISK数组所包含的路线顶多只有两条。针对第一个问题，我尽可能缩小Cost结构体（风险数组每个元素的数据结构类型）所占的空间大小，比如把原本的int型改成了char型，同时及时释放Risk\_future数组。设置多个条件判断语句，减少不必要的循环。针对第二个问题，我尝试了在该函数中使用递归，但造成的空间开销让程序处于崩溃的边缘，于是将递归转变为了迭代，即连续i次的RISK数组构造，每次都将上次构造完成的RISK数组继续代入。进行了多次试验后，我发现i取大于3次时就不会造成最低风险路线的大变动了，而多次的构造却会导致空间飞快消耗，因此最后确定i的值为3。在实际情况的基础上达到了路线相对最优的效果。

此模块的算法为整个程序的核心算法，主体思想是Floyd算法。空间复杂度为O(n2)，时间复杂度为O(n3)（n为城市数）。当城市之间可达率高、距离不远时较适用。同种情况适用的算法还有Dijkstra算法，该算法的空间复杂度为O(m)，时间复杂度为O((m+n)logn)（m为车次总数），该算法在时间上的开销较小，但由于车次的不确定性，空间上不一定优于本程序的算法。总体来说，程序的核心算法开销大，编码复杂度低，易于理解，并且能够保证路线的最优性。

1. **时钟推进输出模块**

* 算法流程图



* 算法思想及特点

进入时钟推进模块时，实际上完整的路线已经生成，保存在以RISK[Depa][Dest].routes为头结点的链表中。此模块定义变量t来模拟时间的推进，每过10s t值加一，代表系统时间推进1h，通过将t值与路线结构体中的出发时间、到达时间作比较，来确定旅客当前时刻的状态，输出到界面和文件中。

## 各模块之间的关系

