软件开发文档

目录

1. 任务描述
2. 功能需求说明及分析
3. 概要设计
4. 数据结构和数据
5. 详细设计
6. 测试及结果说明
7. 评价和改进意见
8. 用户使用说明
9. 任务描述

* 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。某旅客于某一时刻向系统提出旅行要求。考虑在当前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。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）。
* 具体来说程序需根据旅客当前状态，旅客的旅行的地点，按照现实情况（即各个城市的风险值和时刻表），找到符合旅行策略的线路：最少风险策略或者显示最少风险策略。旅客按照此线路旅行，系统能查询当前时刻旅客所处的地点和状态。

1. 功能需求说明及分析

* 现实情况：

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。

所以设置了10个城市，风险分配：0.2\*4+0.5\*3+0.9\*3。城市数目和信息保存为数据文件。

1. 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。不能太简单，城市之间不能总只是1班车次；整个系统中航班数不得超过10个，火车不得超过30列次；汽车班次无限；不能任意两个城市间都有直达交通工具。

所以每个城市的时刻表和对应的城市保存在一起；时刻表每天都一样，整点出发，整点到达。

* 旅客要求：起点，终点，选择的低风险旅行策略。

最少风险策略：无时间限制，风险最少即可。（用户提出请求，可以默认是第二天0点开始旅行）

限时最少风险策略：在规定的时间内风险最少。（限定的时间是指旅途中花费的时间，即从起点出发开始计算，直到到达终点结束计算。

旅客输入策略，程序选择相应算法，得到旅行线路，输出相应策略下的线路。

* 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(有鼠标和键盘输入时系统不计时)。系统精确到一小时，不足一小时向上取整。
* 不考虑城市内换乘交通工具所需时间。
* 建立日志文件，对旅客状态变化和键入等信息进行记录。
* 当查询某个用户当前状态，系统能够返回该用户的状态，如果该用户在旅行中，给出用户所在位置，比在某个城市，或者在从某个城市到某个城市的火车上的描述。

1. 总体方案设计说明

* 软件开发环境：

开发系统：Windows10

开发语言：C语言

开发IDE：DEV

* 总体结构：本程序旨在解决一个旅客某一策略的的旅行计划问题，得到旅行路径。

程序最主要的部分是两个策略的算法问题。

最少风险策略每次确定两点之间的路径时都需要到达起点的时间，这就需要先完成上一个划分的计算才能进行这一次划分的计算，所以使用迪杰斯特拉算法找出起点和终点的最少风险路径。

限时最少风险策略特殊在无法将总问题的最优解划分为每个子问题的最优解，所以我们选取最优解的时候只能通过穷举去遍历所有的解来选取最优解。

* 因为旅客的旅行计划与当前状态查询请求是由键盘输入，且键盘输入不计时，所以将旅客状态变化及时间推进设为主进程，键盘键入为子进程。
* 模块划分

主模块：调用初始化函数，读取文件；主进程是旅行线路设计和输出，旅客状态控制，并以时间轴向前推进；子进程是接受键盘键入命令，分析并响应。

事件处理模块：初始化，建立地图邻接矩阵；日志文件写入；接受键盘键入，分析并响应；时间轴推进；调用旅行线路设计模块进行旅客状态控制（读取，处理，输出）。

旅行线路设计模块：根据旅客要求及当前旅客状态生成最优旅行线路。

1. 数据结构和数据

* 数据结构

struct passenger

{

int start;//起点

int finish;//终点，起点终点相等时，用户处于停留状态

char tool;//交通工具

}; //乘客信息

struct order

{

//char start[20];

int finish;//旅行终点

int strategy;//0为风险最小，1为时间限制下风险最小

int TimeLimit;//时间限制

struct order \*nextorder;//下一个命令的地址

}; //键入的命令，旅客需求

// 时刻表的结构体

typedef struct \_EdgeData

{

//char finish[30];

char tool;//交通工具

int DeTime;//出发时间

int ArTime;//到达时间 两者不超过24小时

}EData;

//地图矩阵

struct city

{

char vexs[10][30]; // 顶点集合名字

int vexnum; // 顶点数

int edgnum[10][10]; // 起点到终点的边数

EData matrix[10][10][6]; //两地之间时刻表的三维数组

float danger[10];//城市的风险值数组

};

struct route

{

//int start;//起点

int finish;//终点

char tool;//交通工具

int Detime;//出发时间 起点的出发时间

int Artime;//到达终点时间

};从起点到终点的路线

* 数据

struct passenger passer;//旅客的状态

struct city graph;//地图

int currenttime;//程序时间，小时为单位，不是24小时制

struct order \*command=NULL,\*end=NULL;//命令的链表头指针，尾指针

int over=0;//为一时，程序退出信号

int flag=0;//设计路线信号，上次旅行结束，还有命令时，flag置1

int fflag=1;//启动计时信号start

int st=0; //为一时，启动计时信号 tfinish

struct route line1[10];//迪杰斯特拉前驱节点数组

struct route line2[10];//旅行路线数组，一个命令对应一个数组

struct route temp;//中间值路线节点

int ll=0,middle=0;//ll是路线节点的个数，middle是当前路线在数组中的位置

int ffflag=0;//一个计划已结束

time\_t start,tfinish;//程序开始和中间时间值

int differtime;//从起点到终点花费时间

int fffflag=0;//时间加1时置1，进行旅客状态处理

int state=0;//旅客的状态，为0时停留状态，为1时旅行途中

int ptime=1;//计时数

* 宏定义

#define INF 9999//无穷大数

1. 详细设计

* 算法思想

最少风险策略：用Dijkstra算法找出起点和终点的最少风险路线。在此算法中我们首先要定义两个点集，分别储存我们已经计算过的点（已确认点集）和没有计算的点（未确认点集），下面是具体的运算过程。

1.将起点加入计算过的点集中，从没有计算的点集中删除，这个过程我们称为一个点的移动。

2.在已确认点集和未确认点集之间我们可以得到连接这两个点集的所有边，在所有边中筛选出最短的一条边（边值是两点所有路径风险值的最小值，由前一城市的风险值和出发时间决定）。

3.将第二步筛选出来的边所连接的未确认点移动到已确认点集中，并更新起点到所有未确认点的最短路径和前驱节点数组。

4.重复2,3,步，直到所有点都被移动到已确认点集。

最后得到所有城市的前驱节点结构数组，通过递归函数得出顶点即当前旅客所在城市到某一城市的最少风险顺序路线数组。

限制时间最少风险策略：由于时间限定问题的特性，所以我们没法在一个最小的子问题中确定一个最优的解，只有当某一特定路径子问题都计算结束之后才能找到这个问题的最优解。

在计算第一个子问题的划分的时候，我们会得到一棵树，这棵树包含了第一个子问题的所有路径。

建树过程利用了图的广度优先搜索，如下：

1.首先将起点作为根节点，并将起点入队列。

2.如果队列不为空，将一个节点出队列，遍历这个节点除了他的祖宗节点以外的所有能够到达的地点，将这些地点建立节点，记录五元组信息，称为这个节点的子节点集，子节点集中的所有节点中的父指针指向这个节点，然后将这个节点的子节点集入队列。

3.重复此过程，直到队列为空，或者出现终点。

4.终点不入队列，并将其信息复制到链表中保存。

建树的过程中，可以通过后续城市赘余原则对其进行剪枝，在建树之后，可以对链表按时间进行排序，并删除时间等价的基础上风险更多的链表节点，同时对树进行剪枝。

在计算下一个子问题的时候，需要上一个子问题已经计算完毕，这样才能针对上一个子问题的每个解都建一棵树来解决当前子问题。也就是说，对于上一个子问题链表的每一个节点都会建一棵新的树，这棵树将会被连接到这个链表所对应的叶子结点上。

对于同一个子问题我们会针对上一子问题的不同解建立多棵树和与其对应的多个链表，我们会将这些链表连接起来，组成一个大的链表，也可以对链表按时间进行排序，并删除时间等价的基础上价钱更多的链表节点，同时对树进行剪枝，这个新的链表作为这个子问题的解链表。

重复上述过程。直到这一排列计算完成。

* 模块关系

主模块

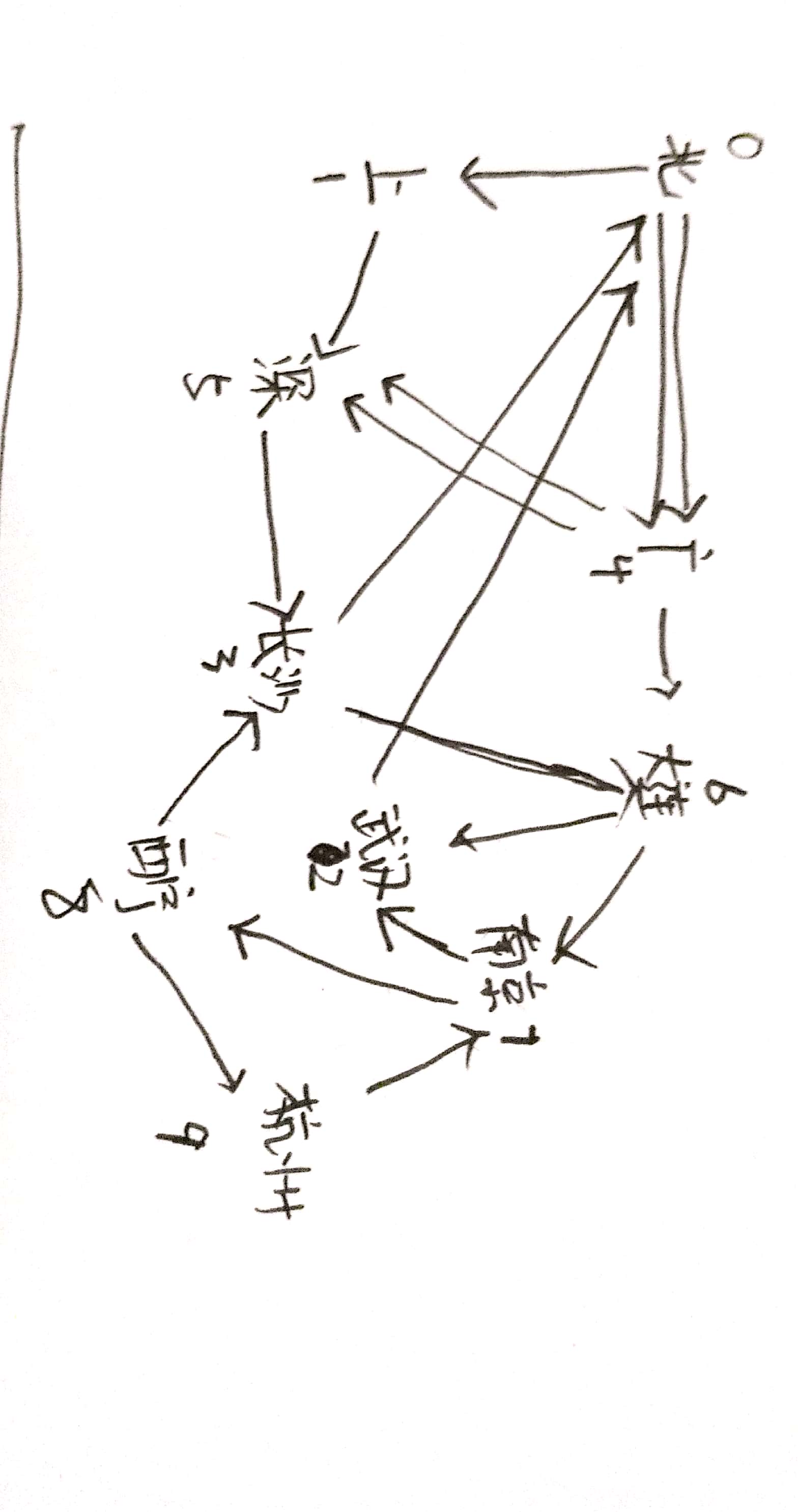
路线设计模块

事件处理模块

数据结构

1. 测试样例及结果说明

* 测试用例



城市信息及时刻表：

city Beijing 0.9 Shanghai 0.5 Wuhan 0.9 Changsha 0.5 Guangzhou 0.5 Shenzhen 0.2 Dalian 0.2 Nanjing 0.9 Xining 0.2 Hangzhou 0.2

trp p Beijing Guangzhou 2 4

trp p Beijing Guangzhou 3 2

trp c Beijing Shanghai 4 9

trp t Guangzhou Shenzhen 10 4

trp t Guangzhou Shenzhen 11 2

trp c Shanghai Shenzhen 8 12

trp c Shenzhen Changsha 3 5

trp c Guangzhou Dalian 2 6

trp c Changsha Dalian 3 8

trp c Dalian Nanjing 3 5

trp c Xining Hangzhou 3 2

trp c Hangzhou Nanjing 2 3

trp c Dalian Wuhan 3 2

trp c Nanjing Wuhan 3 5

trp c 31 Changsha Beijing 3 5

trp t 12 Wuhan Beijing 1 4

trp p Nanjing Xining 1 2

trp c Xining Changsha 3 5

* 结果说明，见功能介绍视频

1. 评价和改进意见

* 评价：有很多帮助性的说明，用户操作简单；

输出和日志文件写入条理清晰，直观反映旅客状态随时间变化情况；

输出效果不够直观，没有使用中文城市名称及交通工具，用了数字和字符代表，但是代表的含义已经提前说明。

时间原因，未能完善有时间限制最少风险策略代码。

* 改进意见：
* 改进输出模块代码，使输出符合用户习惯，比如使用汉语表示城市和交通工具；
* 用图形绘制地图，实时反映出旅客的旅行过程；
* 可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。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；
* 功能需求可以加上：可随意改变城市的数量及其信息，这样不拘泥于10个城市；
* 可以增加暂停机制，即键入暂停信号，除时间不再推进，害可进行其他键入命令，查询操作。
* 经验

充分了解任务和功能需求之后再开始进行模块划分，最后详细设计每个模块。

掌握了主进程和子进程的工作原理和模式以及建立过程。

面向用户，可读性要好，界面需友好。

1. 用户使用说明

程序开始运行，先输入旅客所在地点序号，输入P或Q或E，开始键入命令（终点序号，策略，限制时间）或查询旅客当前状态或退出程序。