数据结构课程设计

——COVID-19疫情环境下低风险旅行模拟系统的设计

2018211317班 2018211613 邱曦远

**一、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），具体旅行策略见后。

**二、功能需求说明及分析**

**需求说明：**

* 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。
* 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；汽车班次无限制；
* 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：
  + 最少风险策略：无时间限制，风险最少即可
  + 限时最少风险策略：在规定的时间内风险最少
* 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(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)；
* 不考虑城市内换乘交通工具所需时间
* 系统时间精确到小时
* 建立日志文件，对旅客状态变化和键入等信息进行记录
* 选做一：用图形绘制地图，并在地图上实时反映出旅客的旅行过程。
* 选做二：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。

**需求分析：**

此任务中，输入分为添加旅客和查询旅客信息两种形式。在添加旅客操作中，输入为一名旅客的起点、终点和选择的旅行策略。起点和终点的数据类型应为字符串，即一座城市的名字；旅行策略由于有两种：最少风险和限时最少风险，因此可以使用一个整型变量来表示，输入0表示执行最少风险策略，其他数字则表示执行限定时间（小时为单位）的最少风险策略。在旅客添加完毕之后，应当立即输出旅客的ID（整型变量）以及选择的路线信息。

若要查询旅客信息，则只需输入旅客的ID（整型变量），随后将会输出旅客的当前所在航班/车次号，或是所在的城市，为字符串类型，同时也会输出系统为旅客选择的路线信息。

城市的名字将使用大写字母A-Z来表示，因此最多会设置26个城市。

航班/车次的命名如下：飞机航班将使用Axx的形式命名（xx表示两位十进制数），如第一号航班A01，第二号航班A02……以此类推，火车车次使用Txx的形式命名，汽车车次使用Cxx的形式命名。因此每类航班/车次数量最多不超过99。

旅客ID将直接使用数字表示，值的范围即为int型变量表示范围。

本旅行模拟系统中，各航班车次的出发至到达耗时均有参考实际用时。

本旅行模拟系统的输出有三种：日志文件输出、命令行输出、图形界面输出。目前支持在命令行界面中通过键盘输入信息，来添加旅客或查询旅客信息。日志文件中将会对旅客状态变化和键入等信息进行记录。

**本次课设中，所有必须的需求以及选做一和选做二我均已完成。**图形界面中，城市地图已绘制，并根据城市风险等级的不同，使用了不同颜色的点来标识城市（高风险城市——红色，中风险城市——橙色，低风险城市——绿色）。地图上也能实时反应乘客的旅行过程。不同交通工具也对应着不同的风险值，旅客的旅行计划中将会考虑乘坐交通工具的风险。

**三、总体方案设计说明**

**软件开发环境：**

Windows 10下的Visual Studio 2017，已安装EasyX库。

**总体结构和模块划分：**



主函数模块为main.cpp，负责调用其余各个模块，运行系统，代码比较简短，如下：

CITY cities[CITYNUM];               //存储城市的数组

DPRT departureTable[DPRTNUM];       //存储车次的数组

PSGR passengerList[MAX\_PSGRNUM];    //存储旅客的数组

PLAN travelPlans[MAX\_PSGRNUM];      //存储旅客旅行计划的数组

int dayTime;                        //以天为单位计数的时间，初始为第0天

int hourTime;                       //以小时为单位计数的时间

int psgrNum;                        //乘客数量

int main()

{

    dayTime = 0;

    hourTime = 5;               //从第0天5:00开始运行程序

    psgrNum = 0;                //初始化乘客数量为0

    citiesInit();               //初始化城市，航班车次，旅客，旅行计划

    departureInit();

    memset(passengerList, 0, (sizeof(PSGR)) \* MAX\_PSGRNUM);

    memset(travelPlans, 0, sizeof(travelPlans));

    initgraph(1200, 700, SHOWCONSOLE);//初始化画布

    while (1)

    {

changeState();      //改变旅客状态

        drawCityMap();      //绘制城市地图

        drawInfo();         //绘制时间信息

drawPsgr();         //绘制旅客

        inputDetect();      //实时检测输入

        printPsgr();        //命令行打印旅客状态信息

        fPrintPsgr();       //日志文件打印旅客状态信息

        Sleep(SLEEPTIME);   //每隔10秒左右推移一个小时

        hourTime ++;        //时间推移

        dayTime = hourTime/24;

        cleardevice();      //清屏

    }

    closegraph();           //关闭画布

    return 0;

}

其余各个模块的功能如下：

State模块：根据当前时间和旅客的旅行计划来更改旅客状态。

Logfile模块：日志文件的输出以及基本的命令行输出，输出内容为航班信息、当前时间信息、旅客状态信息、旅客旅行计划等。

Input模块：用键盘或鼠标方式添加旅客，需要输入旅客的名字、起点、终点，以及旅行计划的时间限制（若时间限制为0表明将选择不限时最少风险策略）。在Input模块中，一旦添加了一名旅客，就会调用Route模块来生成旅客的旅行计划。

Route模块：根据车次表和旅客信息，生成符合条件的最优旅行计划，使用DFS+剪枝算法。

Draw模块：绘制地图，并在地图上反映旅客的实时状态，使用EasyX。

**四、数据结构说明和数据字典**

以下内容均在structure.h头文件中

#pragma once

//符号常量定义

#define CITY\_LOW\_RISK 0.2               //城市低风险值

#define CITY\_MID\_RISK 0.5               //城市中风险值

#define CITY\_HIGH\_RISK 0.9              //城市高风险值

#define CAR\_RISK 2                      //汽车单位时间风险值

#define TRAIN\_RISK 5                    //火车单位时间风险值

#define AIR\_RISK 9                      //飞机单位时间风险值

#define CAR 0                           //定义汽车常量值

#define TRAIN 1                         //定义火车常量值

#define AIR 2                           //定义飞机常量值

#define CITYNUM 11                      //城市数量

#define DPRTNUM 140                     //航班、车次总数

#define SLEEPTIME 8000                   //推进时间间隔为8秒

#define MAX\_PSGRNUM 100                 //最大乘客数量

typedef struct city {

    unsigned int ID;        //城市ID

    char name[20];          //城市名字

    double risk;            //城市风险

}CITY;

typedef struct departure {      //三种交通工具的车次统一命名为DPRT

    unsigned int ID;            //航班或车次ID

    unsigned int vehicle;       //交通工具类型

    unsigned int risk;          //风险值

    unsigned int startTime;     //出发时间

    unsigned int arriveTime;    //到达时间

    CITY \*start;                //始发城市

    CITY \*dest;                 //终点城市

}DPRT;

typedef struct passenger {      //乘客命名为PSGR

    char name[20];              //乘客名字

    unsigned int ID;            //乘客ID

    CITY \*start;                //乘客起点城市

    CITY \*dest;                 //乘客终点城市

    unsigned int startTime;     //乘客始发时间（输入乘客的时间）

    unsigned int timeLimit;

    //乘客旅行计划的时间限制：若为0表示无时间限制，将生成最少风险计划；否则根据限时的值生成限时最少风险计划

    bool isInDeparture;         //若乘客在航班或车次中此变量为1，停留在城市中则此变量为0

    DPRT \*inDeparture;          //乘客所在航班或车次

    CITY \*inCity;               //乘客所在城市

}PSGR;

typedef struct plan {           //乘客旅行计划

    unsigned int psgrID;        //乘客ID

    unsigned int startTime;     //起始时间

    double risk;                //总风险值

    unsigned int timeSpend;     //总耗时

    CITY \*dest;                 //计划目的地

    DPRT \*route[1000];          //此计划中的航班和车次

}PLAN;

主程序中的全局变量：

CITY cities[CITYNUM];               //存储城市的数组

DPRT departureTable[DPRTNUM];       //存储车次的数组

PSGR passengerList[MAX\_PSGRNUM];    //存储旅客的数组

PLAN travelPlans[MAX\_PSGRNUM];      //存储旅客旅行计划的数组

int dayTime;                        //以天为单位计数的时间，初始值为0，即第0天

int hourTime;                       //以小时为单位计数的时间，取值范围为0到正无穷

int psgrNum;                        //乘客数量

**五、各模块设计说明**

**模块关系图（箭头表示调用关系）：**



本模拟系统各个模块之间均具有相当高的独立性，耦合度较低。

**route模块及核心算法介绍：**

route模块中共声明了三个函数以及一些变量如下：

void findRoute(PSGR \*tmp);  //根据形参中旅客生成最佳旅行计划

void dfsRoute(DPRT \*tempDprt, PSGR \*tempPsgr, CITY \*tempCity, int step);    //带剪枝的dfs函数

unsigned int countTime(unsigned int a, unsigned int b);

//根据两个时间点计算耗时的简单函数

bool vis[CITYNUM+10];       //dfs需要用到的结点标记数组

double minR[CITYNUM][5010]; //minR[i][j]表示从起始点到城市i，花费时间为j的最优路径的风险值

double minRisk;             //最小风险值

double totalRisk;           //当前总风险值

unsigned int totalTime;     //当前花费总时间

DPRT \*tRoute[1000][1000];   //存储dfs找到的所有旅行路径

double tRisk[1000];         //存储每一条旅行路径的总风险值

unsigned int tTime[1000];   //存储每一条旅行路径的总时间花费

int routeNum;               //dfs找到的旅行计划数量

本模拟系统中最重要、最核心的算法函数便是findRoute和dfsRoute，findRoute函数会调用dfsRoute函数，使用dfs（深度优先搜索）+剪枝的算法，来对某一旅客生成其最优旅行计划，并存储在travelPlans数组中。

接下来我将自顶向下，介绍算法模块中的各个函数以及其中的核心算法思想：

首先介绍findRoute函数，此函数唯一的参数为一个指向某名旅客的指针。调用此函数后，会根据此旅客的起始时间、始发地、目的地以及时间限制来生成一个最优的旅行计划，如果旅客的时间限制为0，则会生成的旅行路径采用的是不限时最少风险策略，时间限制不为0则会采用限时最少风险策略。生成的旅行计划会直接存储在travelPlans数组中。函数流程图如下：



然后介绍dfsRoute函数，此函数有四个参数，依次为：当前航班的指针、当前旅客的指针、当前城市的指针、深度优先搜索的步数（即旅行中的第几趟车次）。

此函数使用了dfs+剪枝的算法思想，其中剪枝包括根据最少风险来进行的最优性剪枝，以及根据旅行时间限制进行的可行性剪枝，这种算法经过实践是比较优秀的，复杂度要明显优于不带剪枝的普通dfs算法。

函数流程图如下：



此函数中算法的特点就是使用了逻辑清晰易懂的dfs算法，并用剪枝有效减少了dfs的复杂度。但众所周知dfs算法往往时间复杂度不太理想，因此笔者从风险、耗时两方面进行了最优性剪枝和可行性剪枝，这样就预先去除了大量的多余分支和计算过程，有效地减少了算法的时间复杂度。笔者认为此函数中使用的带剪枝的dfs算法和一些启发式的路径搜索算法（如A\*算法等）实质上是殊途同归的。

最后介绍countTime函数，它的功能仅仅是计算两个时刻的时间差值，比较简单，在此展示一下代码即可。

//计算a时刻与b时刻的时间差值，a为较前的时刻,b为较后的时刻

unsigned int countTime(unsigned int a, unsigned int b)

{

    if (int((int)b - (int)a) >= 0) return (b - a);

    else return (b + 24 - a);

}

Route算法模块中只有findRoute函数会被input模块中的inputDetect函数调用，除此之外不存在其他耦合，具有很高的独立性。

**logfile模块设计介绍：**

此模块负责完成日志文件输出以及命令行输出，共声明了六个函数如下：

//日志文件输出：车次、旅客、旅行计划

void fPrintDprt();

void fPrintPsgr();

void fPrintPlan(PSGR \*t);

//命令行输出：车次、旅客、旅行计划

void printDprt();

void printPsgr();

void printPlan(PSGR \*t);

logfile模块会被主函数模块以及input模块调用，除此之外不存在其他耦合。

**input模块设计介绍：**

此模块负责初始化城市和车次，并实时检测信息输入：

void inputDetect();     //输入实时检测

void citiesInit();      //城市初始化

void departureInit();   //车次初始化

input模块仅被主函数模块调用，与其他模块不存在耦合。

城市列表按风险从高到低的顺序排列如下，共有11个城市：

    cities[0] = { 0, "武汉", CITY\_HIGH\_RISK };

    cities[1] = { 1, "北京", CITY\_HIGH\_RISK };

    cities[2] = { 2, "深圳", CITY\_HIGH\_RISK };

    cities[3] = { 3, "上海", CITY\_HIGH\_RISK };

    cities[4] = { 4, "南昌", CITY\_MID\_RISK };

    cities[5] = { 5, "福州", CITY\_MID\_RISK };

    cities[6] = { 6, "长沙", CITY\_MID\_RISK };

    cities[7] = { 7, "重庆", CITY\_MID\_RISK };

    cities[8] = { 8, "大连", CITY\_LOW\_RISK };

    cities[9] = { 9, "衡水", CITY\_LOW\_RISK };

    cities[10] = { 10, "兰州", CITY\_LOW\_RISK };

航班车次时刻表较长，飞机、火车、汽车三种交通工具的车次（航班）数量分别为10，20,110，共有140车次。航班车次时刻表文件在同一目录中，可自行查阅，故不在此列出。

**state模块设计介绍：**

void changeState(); //改变旅客状态函数

state模块中仅有一个改变全体旅客状态的函数，此函数会根据travelPlans数组中存储的每名旅客的旅行计划以及系统当前时间来改变每名旅客的状态（在哪个城市或是在哪个航班车次中）。

此模块仅被主函数模块调用，与其他模块无耦合。

**draw模块设计介绍：**

void drawCityMap(); //绘制城市地图

void drawPsgr();    //根据状态绘制旅客

void drawInfo();    //绘制信息

draw模块负责图形输出，采用easyX，其中有三个函数，分别绘制城市地图、旅客、全局信息。

此模块仅被主函数模块调用，与其他模块无耦合。