**数据结构说明和数据字典（数据名称、用途等）**

**（一）结构体**

①保存制定的路线结果的结构体，包含路线总风险和路线总耗时：

typedef struct result

{

double idleRisk; //路线总风险

int idleTime; //路线总耗时

}Result;

②记录城市栈元素的结构体，该城市栈用于制定限时最少风险策略的路线，每个栈元素包含信息如下：

typedef struct stackinfo

{

int cityID; //城市的编号

int wayFromLast; //到上一个城市的路的下标号（城市间的路可能不止一条）

double riskFromLast; //该城市到上一个城市的风险值

int TimeFromlast; //该城市到上一个城市的耗时

}StackInfo;

**（二）类（变量+函数）**

①交通工具类Vehicle，用于记录交通工具（汽车/火车/飞机）的某个班次的信息，在图中相当于一条有向边，类中包含的信息如下：

class Vehicle

{

public:

int kind; //交通方式

int risk; //该交通工具的风险

int destID; //终点站标号，有向边的起点

int departID; //起始站标号，有向边的终点

int departTime; //出发时间

int arrivalTime; //到达时间

int days; //始发时间和到站时间之间跨越的天数

std::string number;//班次号

int transTime; //路途花费时间

double transRisk; //路途风险

int xSpeed; //经过多少毫秒x坐标更新一次，用于动画界面更新

int ySpeed; //经过多少毫秒y坐标更新一次，用于动画界面更新

Vehicle(int kind, int departID, int departTime, int destID, int arrivalTime, int days, std::string number); //交通工具类的构造函数

};

②城市类City，用于记录地图上的城市的信息，在图中相当于一个顶点，类中包含的信息如下：

class City

{

int id; //城市序号

std::string name; //城市名

int x; //该城市在地图上的x坐标

int y; //该城市在地图上的y坐标

double risk; //这个城市的风险值（高/中/低）

public:

std::vector<Vehicle> neighbor[CITY\_NUM];//该城市到邻居的路，记录所有以该城市为起点，邻居为终点的有向边

City(int id, std::string name, int x, int y, double risk);//该类的构造函数

void AddNeighbor(int kind, int departID, int departTime, int destID, int arrivalTime, int days, std::string number);//添加以该城市为起点，邻居为终点的有向边

int GetID() const;//返回城市的ID，用于类外访问

int GetX()const; //返回城市的x坐标，用于类外访问

int GetY()const; //返回城市的y坐标，用于类外访问

double GetRisk()const;//返回城市的风险值，用于类外访问

std::string GetCityName()const;//返回城市的名字，用于类外访问

};

③点类Vertex，用于制定最少风险策略和最少时间策略的路线，每个点对应到一个城市，每个点包含信息如下：

class Vertex

{

public:

bool isMarked; //表示该城市是否被标记，总共16个城市

int minWayIndex; //该城市的多重边中被选中的那条边的编号

int preVertex; //标记该城市在路线中的前一个城市

int startWaitTime; //上一个城市到达这个城市的时间

double riskToDepart; //距离起始点的风险值

int timeToDepart; //距离起始点的时间

Vertex();//该类的构造函数

void setVertex(bool isMarked, int preVertex, double riskToDepart, int timeToDepart, int startWaitTime); //更改某个点的信息

};

④旅客类Passenger

class Passenger

{

int seqNum; //整型，用于标记乘客序号

std::string name; //字符串型，乘客姓名

int startTime; //整型，乘客制定计划的时间

double preciseStartTIME;//精确到小数的乘客制定计划的时间

int departure; //起点城市的标号

int destination; //终点城市的标号

int strategy; //选择的策略 0:最小风险；1:最短时间；2：限时最小风险

int state; //乘客的状态 1等待出发/2到达目的地/3在路上

int x; //在图上的x坐标

int y; //在图上的y坐标

int curCity; //旅客当前所在城市,更改方案的时候会用到

int expectedTime; //限定的时间

Result idleResult; //理想的结果，包含路线总时间和路线总风险

int travelHours; //某一趟车已用耗时（小时

int usedHours; //已用时间

int printStartTIME; //打印用的出发时间

int res;

//存路径，终点的数组下标是0，起点在栈顶

std::vector<Vehicle\*> minRiskPath;//最少风险策略的路径

std::vector<Vehicle\*> minTimePath;//最少时间策略的路径

std::vector<Vehicle\*> limitTimeMinRiskPath;//限时最少风险策略的路径

std::vector<Vehicle\*> curPath;//当前还未完成的路径

public:

clock\_t lastXMoveTime;//上一次x移动的时间（非小时，指毫秒

clock\_t lastYMoveTime;//上一次y移动的时间（非小时，指毫秒

Passenger(int seqNum, int startTime,double pTIME, int departure, int destination, int strategy, int expectedTime, clock\_t time, std::string name); //构造函数，添加乘客时使用

int GetStrategy()const; //返回该乘客选择的策略

int GetCurCity()const; //返回该乘客当前所在城市

int GetDepart()const; //返回出发地标号

int GetDest()const; //返回目的地标号

int GetExpectedTime()const;//返回限定时间

int GetUsedHours()const; //返回已用时间

int GetState()const; //返回该乘客的状态

std::string GetName()const;//返回该乘客的姓名

void ChangeDest(int newDest, int newTime);//旅客变更目的地时的操作

void ChangeStrategy(int newStrategy, int newTime);//旅客变更策略时的操作

void UpdateHours();//更新已经用的时间（从制定计划的那个时间算起）

void MakePlan(); //制定路线前的准备，并根据策略选择相应的算法制定路线

void MovePassenger();//更新乘客位置及状态

Result MakeMinPath(int stra); //制定最少时间/风险策略的路线方案

int MakelimitTimeMinRiskPath();//制定限时最少风险策略的路线方案

void Print()const; //常规模式下打印乘客信息（到动画界面）

void PrintWhenChange()const;//更改旅行方案模式下打印的乘客信息到动画界面

void PrintLog()const; //打印乘客信息到日志中

bool whetherRemake()const; //判断更改旅行计划后是否需要重新制定路线

};

**（三）全局变量**

std::vector<City> cities;//城市列表，保存所有的城市

std::vector<Passenger> passengerList;//保存所有添加进系统的乘客

bool isStart; //用于表示系统是否启动，true启动，false未启动

bool isStop; //用于表示系统是否暂停计时（用户键入信息时会暂停）

int curState; //用于表示系统当前的状态

int TIME; //全局时间（显示的时间），整型

double preciseTIME; //全局时间，精确到小数

clock\_t startTime; //系统启动的时间，更新时序时会用到

clock\_t stopTime; //暂停开始的时间

clock\_t endStopTime; //结束暂停的时间

clock\_t stopInterval; //某次暂停间隔

clock\_t totStopInterval;//总共暂停时间

clock\_t currentTime; //当前时间

clock\_t lastUpdateTime; //上次更新动画界面的时间，用于计算是否需要更新界面

int curPassID; //当前界面上显示的乘客的序号

int passStartTime; //当前操作的乘客出发时间，用于生成新乘客

double passPreciseTIME;//精确到小数的当前操作的乘客出发时间，用于生成新乘客

int passDeparture; //当前操作的起点城市，用于生成新乘客和打印

int passDestination;// 当前操作的终点城市，用于生成新乘客和打印

int passStrategy; //当前操作的选择的策略 ，用于生成新乘客和打印

int passExpectedTime; //当前操作的限定的时间，用于生成新乘客

**（四）函数**

void InitTable(); //用于初始化时刻表

void InitCityList(); //用于初始化城市列表

int ToCityID(std::string name);//用于将时刻表中的城市转换成他的标号

void Continue(); //用于将系统恢复计时并变回常规状态

void Stop(); //用于将系统暂停

int InputTime(); //用于输入乘客的限定时间

std::string InputPassName();//用于输入乘客的姓名

void StartSystem(); //用于启动系统

void TransferSystemState(); //用于转换系统状态

void Update();//用于更新时序和所有乘客的信息（状态、坐标等）

void LoadImg(); //加载图片

void InitCanvas(); //初始化画布

void Draw(int curPassId);//打印动画界面

bool Between(int left, int right, int top, int bottom, int x, int y);//判断用户点击的位置是否在某个区域内

int MatchCity(int x, int y); //匹配用户所点击的城市

int MatchStrategy(int x, int y);//匹配用户所点击的策略