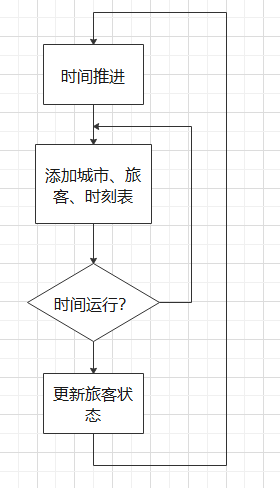
**一、算法思想**

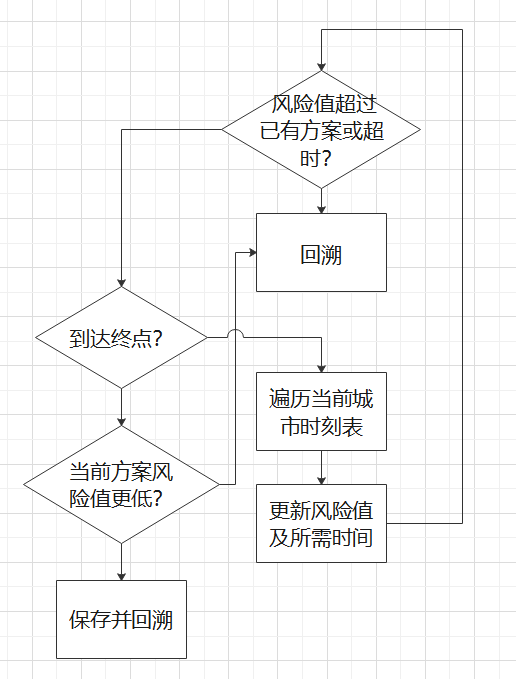
总体程序执行顺序为：



**二、算法**

该程序的核心问题是路线规划。这一部分的解决方案包含在Tourist类中，由main函数在时间推进时集中更新Tourist的状态信息，并在适当时机调用路线规划的深度优先搜索函数。

每当旅客可出发（位于出发点或中转站）时，进行一次深度优先搜索（遍历当前城市的时刻表，记录经过的时间和风险值，并递归寻找可能的下一站，直至找到能够到达终点的路线时保存。当寻找到风险值更低的方案时，更新当前最优方案；当寻找到风险值相同的方案时，保存多条可选方案），即判断流程大致如下：



核心递归函数：

void Tourist::dfs(City\* at, Route rut, int time)

{

if(this->limit == INT\_MAX) //不限时策略 不访问已经访问过的城市

{

if(rut.via.size() > 1)

{

for(unsigned i = 0; i < rut.via.size() - 1; i++) //检查路径中之前访问的所有城市

if(at->name == rut.via[i]->name) return;

}

}

if (time > this->limit || rut.risk > min\_risk) return; //当前规划超过最晚到达时间或风险超过已有方案

if (at->name == this->destination->name) //完成一次规划 检查是否能更新当前方案

{

if (rut.risk < min\_risk)

{

min\_plan.clear();

min\_plan.push\_back(rut);

}

else if(rut.risk == min\_risk) min\_plan.push\_back(rut); //若风险值相同 则记录多条符合线路

return;

}

for (auto i : at->everyday\_table) //deque<Transport> 查询当地城市的交通工具表

{

int nowtime = time % 24, real\_start\_time;

if (nowtime <= i->start\_time) real\_start\_time = time + i->start\_time - nowtime;

else real\_start\_time = time + i->start\_time - nowtime + 24;

if (rut.via.empty()) //第一个

{

rut.start\_time = real\_start\_time;

rut.time\_cost = i->time\_cost;

rut.transportKind = i->transportKind;

}

rut.via.push\_back(i->destination);

//新增风险值 = 等待时间风险(所在城市风险\*停留时间) + 旅行时间风险(乘坐时间\*交通工具风险值\*起点城市风险值)

rut.risk += at->risk \* (real\_start\_time - time) + i->time\_cost \* i->risk \* at->risk;

rut.mid\_arrive\_time.push\_back(real\_start\_time + i->time\_cost); //记录再次出发、到达时间 方便日志功能实现

rut.mid\_again\_time.push\_back(real\_start\_time);

dfs(i->destination, rut, real\_start\_time + i->time\_cost);

rut.risk -= at->risk \* (real\_start\_time - time) + i->time\_cost \* i->risk \* at->risk; //回溯

rut.via.pop\_back();

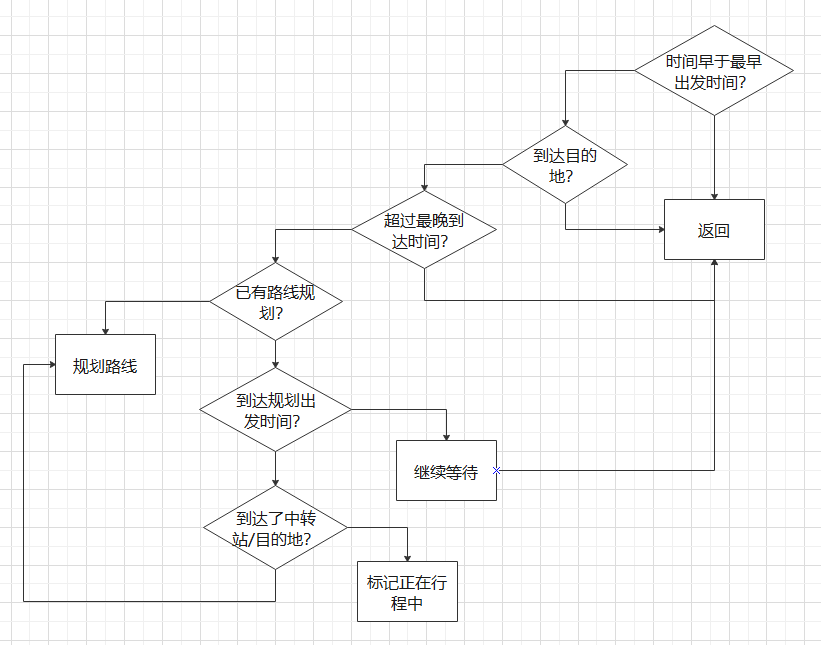
rut.mid\_again\_time.pop\_back();

rut.mid\_arrive\_time.pop\_back();

}

}

关于旅客状态更新方式：



**三、特点及与其他模块的关系**

|  |  |  |
| --- | --- | --- |
| 旅行线路设计模块 | city.h | 城市类。保存有基本信息和以此城市为起点的时刻表。用于向旅客类提供统一的处理城市信息的结构。 |
| tourist.h | 旅客类。保存有基本信息、更新方法和路径规划的算法。 |
| transport.h | 时刻表类。保存有基本信息。用于向城市类提供统一的保存时刻表信息的结构。 |
| route.h | 路径类。保存有基本信息和沿途所经城市。用于向旅客类提供统一的保存路径信息的结构。 |
| 状态动态查询显示模块 | mymap.h | 地图类。用于实现图形化，保存有通过当前信息绘制城市、旅客、交通工具的方法。用于向主窗口提供统一的更新图形化信息的结构。 |
| 主模块和输入输出模块 | mainwindow.h | 主窗口。保存有时间推进、处理输入信息、调用旅客更新、调用地图绘制、显示状态信息的方法。 |