山东大学计算机科学与技术学院

数据结构与算法课程设计报告

学号：202000130143	姓名： 郑凯饶	班级： 20.1
上机学时： 6		日期：2022-5-19
课程设计题目：低风险出行系统		
软件环境：Windows 10 家庭中文版 64 位（10.0，版本 18363） Microsoft VS Code cmake version 3.23.0-rc3		

报告内容：

1.需求描述

1.1 问题描述

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）。

1.2 基本要求

（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。

（2）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；

（3）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。

（4）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)；

（5）不考虑城市内换乘交通工具所需时间。

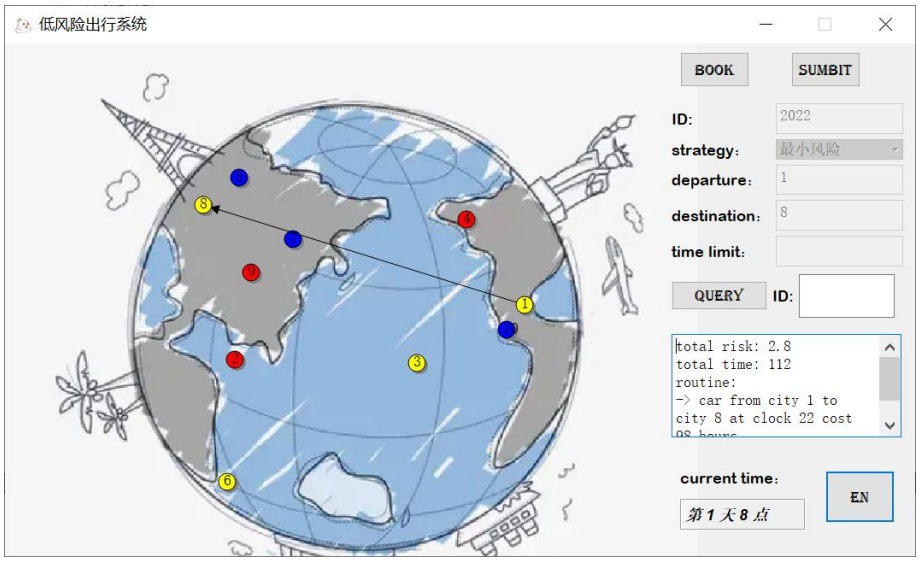
（6）系统时间精确到小时。

（7）建立日志文件，对旅客状态变化信息进行记录。

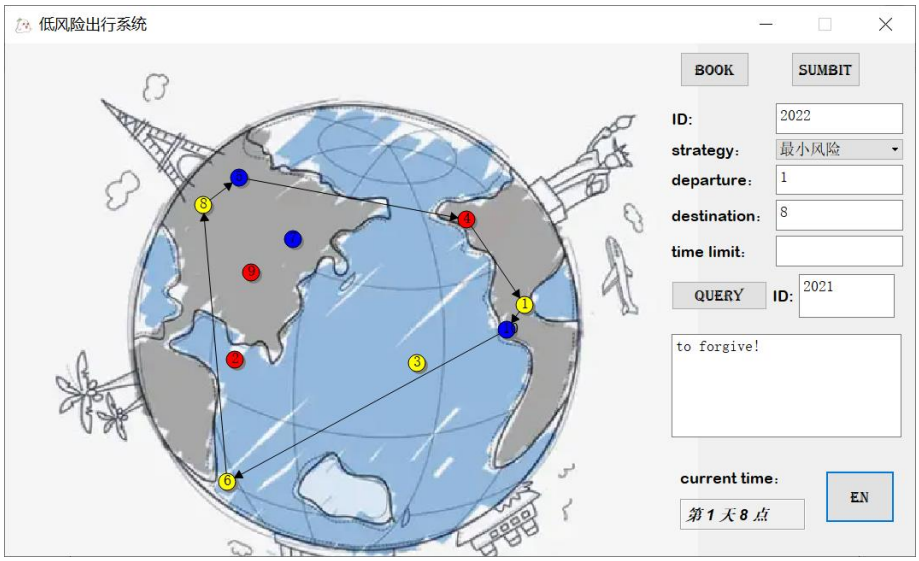
1.3 输入说明

输入界面设计

1. 输入通过右侧编辑框、按钮进行；



2. 也可以通过左侧 canvas 进行交互（我们可以尝试绘制一个“凸包”）。



输入样例

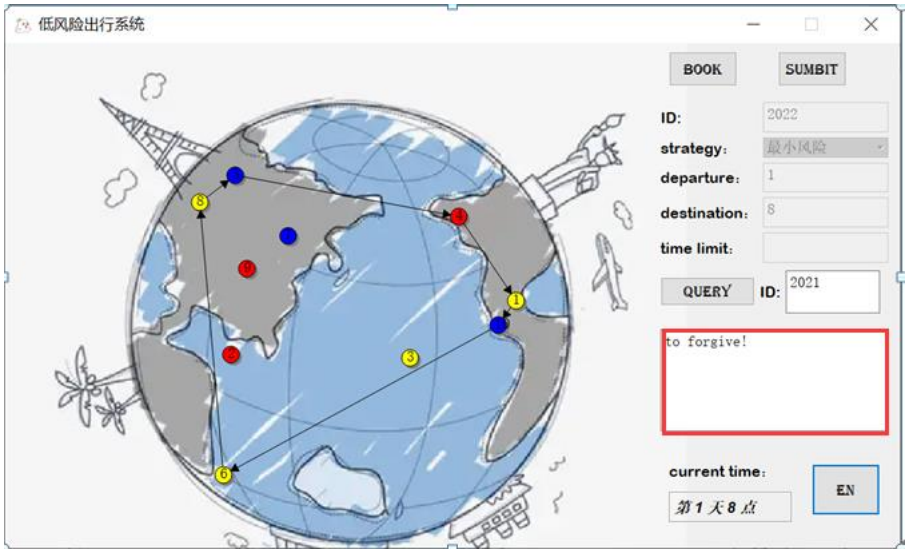
Items.in 文件

City.in 文件

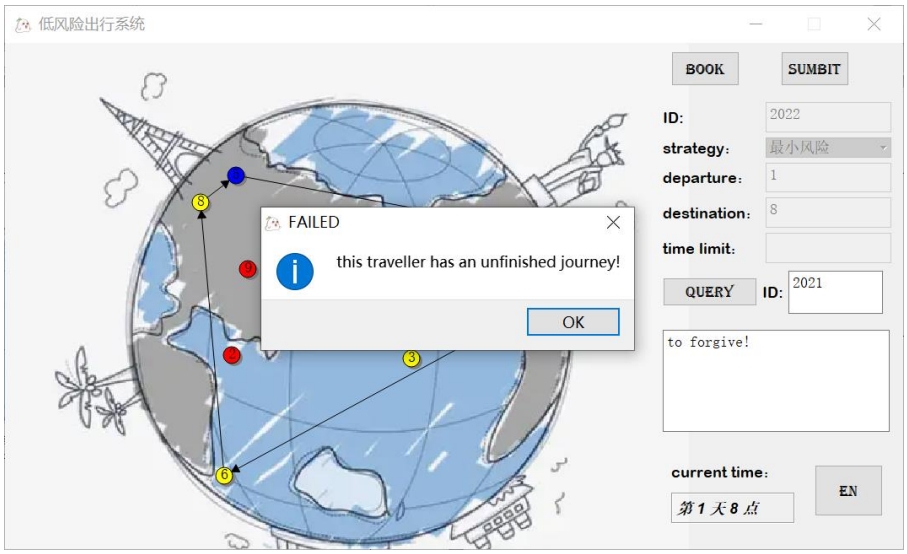
1.4 输出说明

输出界面设计

1. 日志及旅客状态输出在右侧红色框 QTextEdit；



2. 同时左侧可视化旅客推荐路线以及行进路线；
3. 对于用户不规范操作进行消息提示。



输出样例

Plan.out 文件

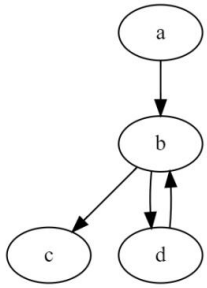
2. 分析与设计

2.1 问题分析

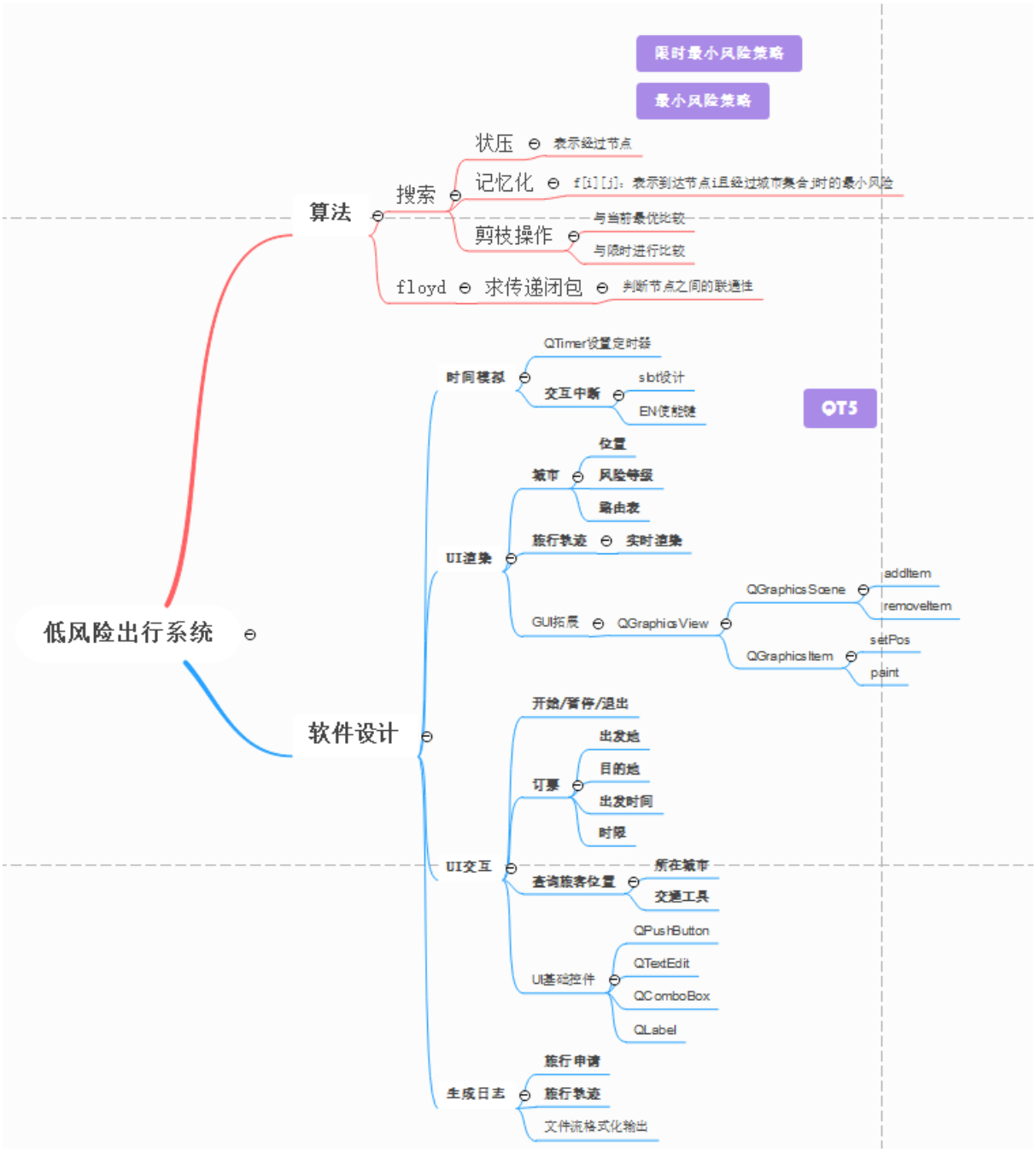
由于交通来往和时间密切相关，因此问题不可以简单抽象为求两点之间的一条路径，我们尝试为每个节点维护一张“路由表”，记录何时该节点有路通往其他节点，然后通过搜索算法，为旅行者制定一条满足指定条件的路线。

下面我们进行定义与假设：

1. 总风险为在起点城市以及中转城市停留风险的总和。
   每个城市的停留风险可以根据基本要求（1）计算。
2. 城市之间每天交通工具的来往是固定的。比如 A 城市每天 8，13，18 点有汽车开往 B 城市。而汽车通车的频率高于火车的，火车的高于飞机的。
   对基本要求（2）进行修改，实际生活中系统中航班数应该是随时间推进而不断增加的，不应该有参数设限。
3. 每个旅途中每个城市至多经过一遍。
   假设一种极端情况：
   旅客从a去往c，由于没有直达的交通工具，他必须途径b城市。他于9点到达b，而b往c的车每天8点出发，并且每天只有一趟。这时系统发现，b与另一城市d之间有车辆来往并且可以与明天8点前回到b，旅客在城市b逗留会增加风险，所以系统会推荐旅客在bd之间来回观光一次甚至多次！



2.2 主程序设计


限时最小风险策略
最小风险策略
低风险出行系统
算法
搜索
状压
表示经过节点
记忆化
f[i][j]：表示到达节点i且经过城市集合j时的最小风险
剪枝操作
与当前最优比较
与限时进行比较
floyd
求传递闭包
判断节点之间的联通性
软件设计
时间模拟
QTimer设置定时器
交互中断
sbt设计
EN使能键
QT5
UI渲染
城市
位置
风险等级
路由表
旅行轨迹
实时渲染
GUI拓展
QGraphicsView
QGraphicsScene
addItem
removeItem
QGraphicsItem
setPos
paint
UI交互
开始/暂停/退出
订票
出发地
目的地
出发时间
时限
查询旅客位置
所在城市
交通工具
UI基础控件
QPushButton
QTextEdit
QComboBox
QLabel
生成日志
旅行申请
旅行轨迹
文件流格式化输出

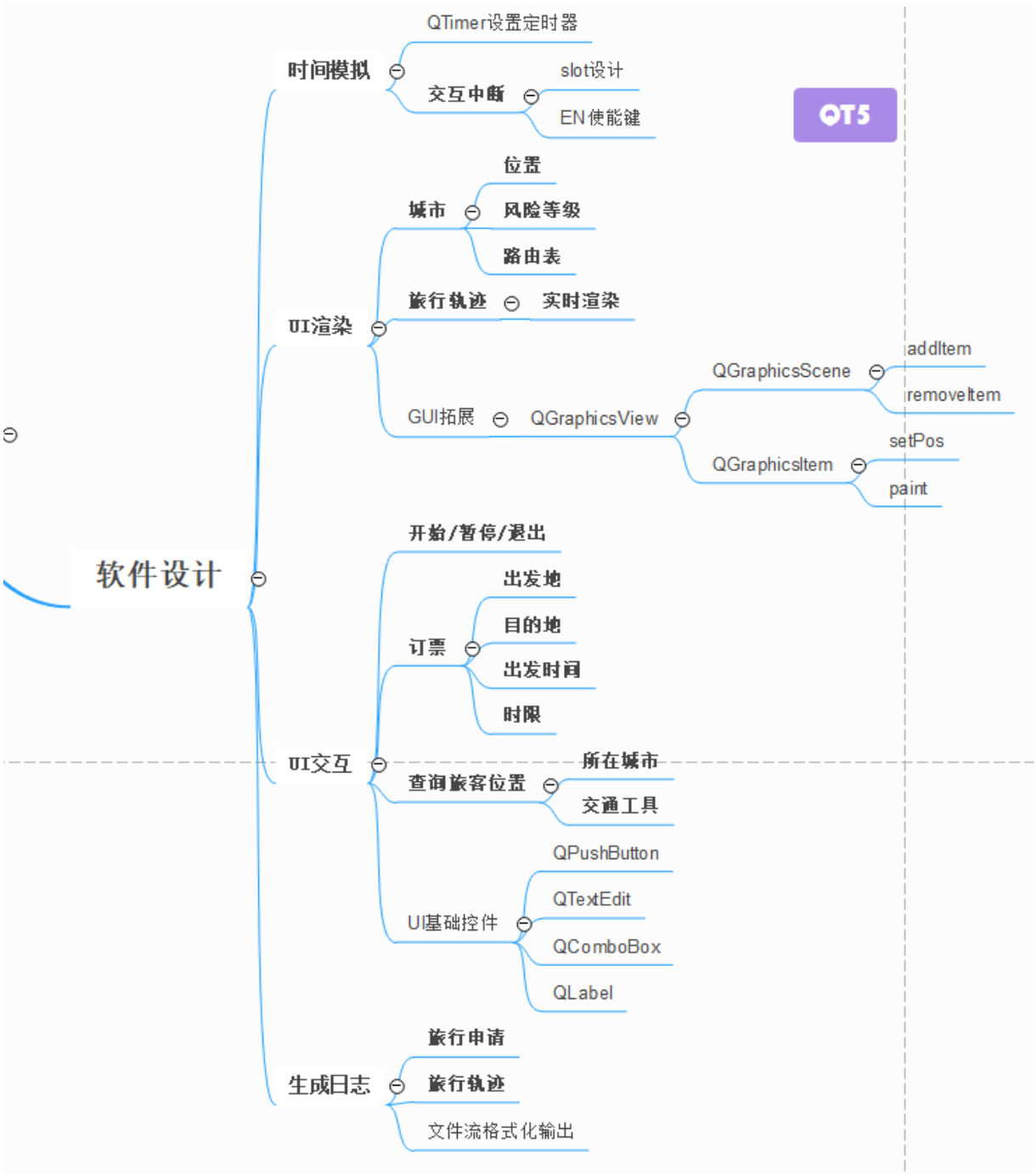
2.3　设计思路

算法部分：

总体答案的求解通过搜索，状态压缩简便地表达假设 3（每个旅途中每个城市至多经过 1 遍），记忆化通过记录阶段性子问题的解加速搜索（不做重复的搜索），同时持久化存储答案，下次查询时可以快速返回，最后剪枝不可能获得最优答案的分支加速搜索。

弗洛伊德算法用于计算城市之间的传递闭包，快速判断是否存在旅途从城市 A 到城市 B.

Qt 软件部分(系统设计)：



要求（4）（6）对时间模拟进行了要求，我通过 QTimer 设置定时器，每 2s 触发一次并对当前时间变量增 1（以小时为单位），另一方面，当用户进行操作（键盘鼠标），停止计时器运行，模拟系统中断，可以通过使能键 EN 恢复系统运行。

UI 渲染是指图形化展示系统。首先渲染要素包括城市和旅行轨迹，两者用 QGraphicsItem 表达。城市主要有风险等级、位置和路由表 3 个特征。风险等级通过颜色标识，位置以一定比例尺映射至 QGraphicsScene 中，通过 setPos 方法设置。旅行轨迹要实时渲染，重写继承方法 paint。

UI 交互是指用户的操作界面。设置使能/订票/查询等按钮，订票要求输入出发地、目的地、出发时间（缺省设置）、时间限制（限时最小风险策略），查询将返回旅客所在城市或者其所在交通工具。整体操作界面依赖一系列 UI 基础控件实现。

生成日志主要是记录订票时生成的旅客出行路线（包含旅客在系统中的所有状态），通过文件流格式化输出。

2.4 数据及数据类(型)定义

算法部分：

基础数据结构定义：

航班、车次：

```
struct item
{
    int type;               // 交通工具 (0: 汽车 1: 火车 2: 飞机)
    int sCity, dCity;       // 出发地，目的地
    int sTime, time;        // 出发时间，耗时
};
```

城市：

```
const double rLevel[3] = {0.2, 0.5, 0.9};

struct city
{
    int seq;                // 城市序号
    int r;                  // 风险等级
    int x, y;               // 位置
    list<item> routeTable;  // 时刻表（路由表）
};
```

时刻表大小不确定，考虑可能还会动态修改，使用链表储存。

旅客申请：

```
struct travel
{
    int seq;            // 旅行序号
    int str;            // 策略 (0: 最低风险 1: 限时最小风险)
    int sCity, dCity;   // 出发地，目的地
    int sTime;          // 申请时间
    double totRisk;     // 总风险
    int totTime;        // 总时间
    list<item> routine; // 规划路线
};
```

记忆数组

```
trace f[35][35][24];   // f[i][j][t]: 于t点申请从城市i前往j的最低风险路线
```

定义“路径”：

```
struct trace
{
    trace():E(false), totRisk(1e10), totTime(0) {}
    bool E;
    double totRisk;
    int totTime;
    list<item> routine;
};
```

Trace 用于记忆化搜索。

将以上算法函数进行封装为 travelSystem，设计接口：

```
class travelSystem {
public:
    travelSystem();

    void setLimit(int t);
    void setTraveller(int id);
    void init();

    int ID(int id);                     // 创建新内部序号，或者查询内部序号
    int query(int rid, item &it);       // 通过内部序号查询旅客状态
    QString qState(int id, int t);      // 查询旅客精确状态

    void savePlan(int r);                   // 持久化旅客出行计划
    void updatePlan(int t);                 // 更新旅客状态
    QString outputPlan();                   // 将推荐路线输出至QString

    int getSize();
    city* getNodes();
    list<item> getRoutine();
    list<item> getTravel(int id);

    void dfs(int i, int s, int t, int stime, double r, int cost, int vis);         // 最小风险策略
    void dfs1(int i, int s, int t, int stime, double r, int cost, int vis);        // 限时最小风险策略
//    trace dfs2(int i, int s, int t, int stime, double r, int cost, int vis);     // 最小风险策略记忆化搜索

    int Dur(int a, int b);
```

```
private:
    // 静态变量
    const double rLevel[3] = {0.2, 0.5, 0.9};
    const static int N = 32;
    const QString type[3] = {"car", "train", "plane"};

    // 图
    int n;
    city gCity[N];

    bool reach[N][N];

    // 旅行计划
    int traveller;
    double totRisk;
    int totTime, timeLimit;
    list<item> routine;

    list<item> rt;

    // 系统中是否允许有多个旅客？允许
    // 持久化routine
    // 旅客状态记录
    map<int, int> mp;   // 映射id
    list<item> travel[15];
    int cnt;
    int clk[15];        //  行程沙漏值
    int wait[15];       //  等待沙漏值

    // 初始化子操作
    void readCity();
    void readItem();
    void floyd();       // floyd求传递闭包：判断城市之间是否可达
    void printReach();
};
```

系统部分：

窗口界面定义（沿用 Qt 的设计标准）：

```
namespace Ui {
    class Widget;    // ui_widget.h文件中定义的类，外部声明
}

class Widget : public QWidget
{
    Q_OBJECT     // 宏，使用信号与槽机制必须添加

public:
    explicit Widget(QWidget *parent = nullptr);
    ~Widget();
    void paintEvent(QPaintEvent *);      // 渲染&&可视化

    GraphWidget *mGW;
private:
    Ui::Widget *ui;                 // 指向界面
    void initUI();                  // 初始化界面
    void initConnect();             // 初始化信号槽

private slots:

    void book();     // 订票
    void sumbit();   // 提交
    void query();    // 查询

signals:

};
```

可视化渲染 canvas 定义（还有 Node、Edge 类的定义在源码中呈现不再赘述）：

```
class GraphWidget : public QGraphicsView
{
    Q_OBJECT

public:
    GraphWidget(QWidget *parent = nullptr, int cnt = 10, QSize size = QSize(1080, 600));  // 600,480
    void itemMoved(); // 定时检测节点是否移动
    int getCnt() { return Cnt; }
    void chooseNode(int); // 增删边

    void Set(city*);
    void reset();  // 刷新

    void setRoutine(list<item> rt);     // 当前旅客的计划路线或剩余路线渲染
    void clrRoutine();

    void addEdge(int, int); // 加边
    Edge *edge[15][15];

public slots:
    void shuffle();
    void zoomIn();
    void zoomOut();

protected:
    void keyPressEvent(QKeyEvent *event) override;      // 键盘输入事件
    void timerEvent(QTimerEvent *event) override;       // 设定一次定时器

    void wheelEvent(QWheelEvent *event) override;       // 鼠标滚轮事件

    void drawBackground(QPainter *painter, const QRectF &rect) override;

    void scaleView(qreal scaleFactor);    // 视图大小调整


private:
    QGraphicsScene *myscene;          // 平面，用于承载二维对象
    int timerId = 0;
    Node *node[15];      // 节点对象指针

    Node *crNode;
    int Cnt;             // 节点数
    QSize m_size;

    bool isWait;         // 是否已有选中点等待边操作
    int fn;              // 记录选中的第1个点
};
//! [0]
```

QGraphicsScene 是一个平面对象，可以承载诸如点、线的二维对象（QGraphicsItem），slot 中的方法可以对视图大小进行调整。

2.5. 算法设计及分析

最低风险策略

```
void dfs(int i, int s, int t, int stime, double r, int cost, int vis, list<item> rt) {
    if (s == t) {
        // 比较风险，更新路线
    }

    // 扫描当前城市的路由表
    for () {
        // 状压判断前往城市是否是经过城市，是则跳过
        if (vis & (1 << (it->dCity - 1))) {
            continue;
        }
        // 计算停留时间 dcost 以及停留增加风险 dr

        // 比较当前累积风险与当前已经完成最优路线的总风险
        // 更优则更新，否则剪枝
        // 若不进行剪枝，时间复杂度为 O(n!)
        if (r + dr > gTravel[i].totRisk) continue;

        // 标记前往城市已访问
        // 并将其加入路线
        vis ^= (1 << (it->dCity - 1));
        rt.push_back(*it);

        dfs(i, it->dCity, t, stime + dcost, r + dr, cost + dcost, vis, rt);

        // 恢复
        vis ^= (1 << (it->dCity - 1));
        rt.pop_back();
    }
}
```

时间复杂度分析：

搜索相当于一次“全表扫描”，但由于城市在单次旅途中至多经过 1 次，设城市数为 n，虽然城市之间有多种多躺交通工具来往，但其不会随 n 增长而增长，因此复杂度可以表达为 O(n!)。考虑到剪枝优化，复杂度实际更优秀。

限时最低风险

```
void dfs1(int i, int s, int t, int stime, double r, int cost, int vis, list<item> rt) {
    if (s == t) {
        // 比较风险，更新路线
    }

    // 扫描当前城市的路由表
    for () {
        // 状压判断前往城市是否是经过城市，是则跳过
        // 计算停留时间 dcost 以及停留增加风险 dr

        if (r + dr > gTravel[i].totRisk) continue;

        // 超过时间限制
        if (cost + dcost > TL) continue;


        // 标记

        dfs1(i, it->dCity, t, stime + dcost, r + dr, cost + dcost, vis, rt);

        // 恢复
    }
}
```

和最低风险策略相似。

伪代码

```
trace dfs2(int i, int s, int t, int stime, double r, int cost, int vis, list<item> rt) {
    if (f[s][t][daytime(stime)].E == true) {
        return f[s][t][daytime(stime)];
    }

    trace res;

    // 扫描当前城市的路由表
    for (; it != end; it++) {

        // 状压判断经过城市
        // 计算停留时间 dcost 以及停留增加风险 dr
        // 剪枝

        // 标记
        trace res1;
        res1 = dfs2(i, it->dCity, t, stime + dcost, r + dr, cost + dcost, vis, rt);

        // 回溯时已求解出 it->dCity 至 t 的最优路线
        // 记忆 s 至 t 于 stime + duration 时刻出发的最优路线
        f[s][t][daytime(stime + duration)] = res1;

        // 尝试更新 s 至 t 于 stime 时刻出发的最优路线
        if (res1.totRisk < res.totRisk) {
            res = res1;
        }
        // 恢复
    }

    return f[s][t][daytime(stime)] = res;
}
```

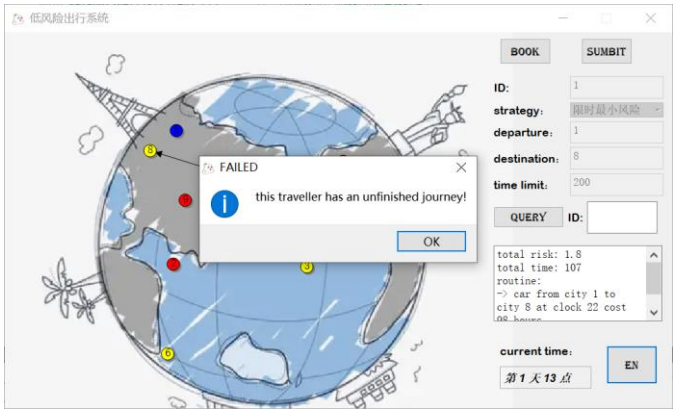
时间复杂度分析：

一开始单次的搜索同原始方法相同，但随着查询次数增加，在系统不发生改变的情况下，复杂度逐渐变为 O(1)（此时所有答案已经求解完成，实际为数组 O(1) 查询返回），均摊下来复杂度可以十分优秀！

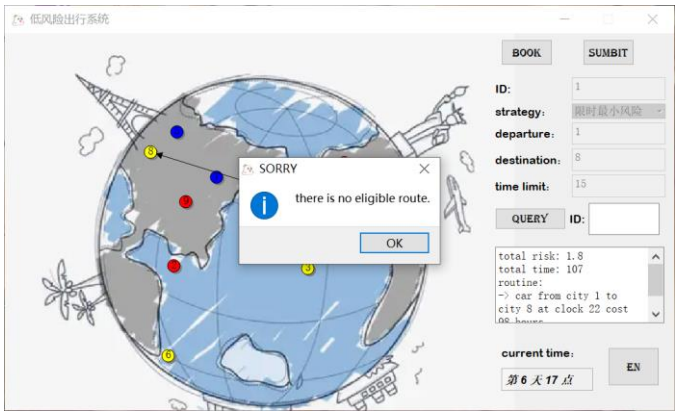
3. 测试

用户交互测试：

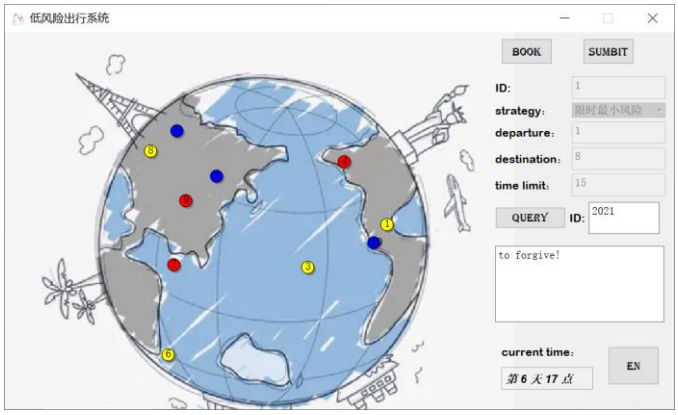
（1） 旅行有未完成行程，QMessage 进行提示：



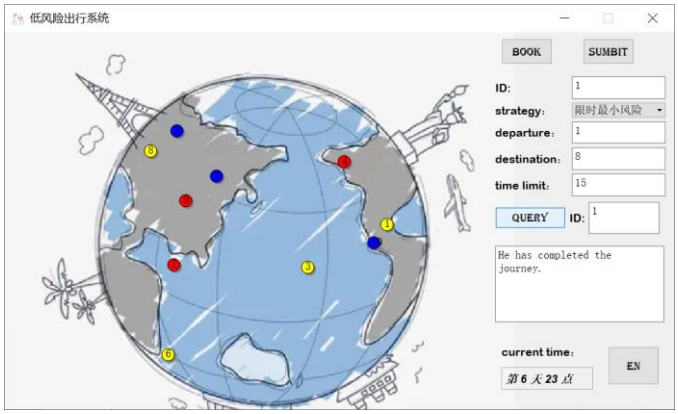
（2） 没有符合旅客需求的旅行方案：



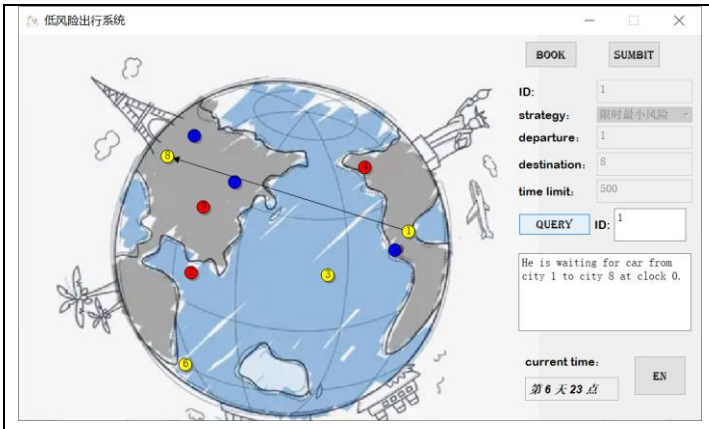
（3） 查询时，没有相应旅客信息，在右侧消息框中显示 to forgive!（查无此人）：



（4） 正常查询：

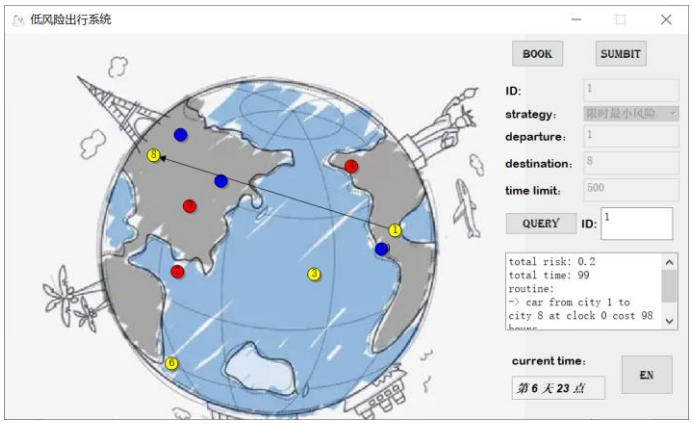


（查询到该旅客已经完成行程）



（ID 为 1 的旅客正在城市 1 等待前往城市 8 的  ）

（5） 正常订票：



消息框中显示旅行总风险、总用时以及具体路线，左侧 canvas 渲染路线。

算法正确性验证：

与暴力方法进行比较，同时各个方法之间也可以相互佐证正确性。

4. 分析与探讨

这次大作业我分为算法与系统（Qt）两部分完成，两个部分前前后后都用去了一个星期，过程中我回顾了 DSA 搜索算法的相关设计、C++面向对象的设计方法以及学习了 Qt 工程的编写。

但是 covid-19（我对于该项目的命名）距离现实中的软件工程项目还有很大差距，从算法设计方面，我的算法鲁棒性较低，只适用于一类情况，而且考量指标单一，远远未及现实问题的复杂性。希望有机会学习途游等等旅游平台的路线规划算法，不过在新冠疫情的背景下，covid-19 也对低风险出行规划提出了思考（虽然是课设题目要求的 233）。

已将代码开源，可能 DSA 课设的题目不会人人都一样，但是我觉得我魔改的拓扑图可视化渲染的 GraphWidget 组件在图类问题的可视化大有用处，可以作为大家在图形化渲染方面的入门参考。我会积极维护，不断完善其功能。

以上。

5. 附录：实现源代码 （另附）