面向对象设计与构造2022第二单元总结

零、任务简介

模拟一个多线程实时电梯系统。

系统基于一个类似北京航空航天大学新主楼的大楼，大楼有 A,B,C,D,E 五个座，每个楼座有对应的一台电梯，可以在楼座内 1-10 层之间运行。

系统从标准输入中输入请求信息，程序进行接收和处理，模拟电梯运行，将必要的运行信息通过输出接口进行输出。

具体而言，本次作业电梯系统具有的功能为：上下行，开关门，以及模拟乘客的进出。

电梯系统**可以采用任意的调度策略**，即在任意时刻，系统选择上下行动，是否在某层开关门，都可自定义，只要保证在**电梯系统时间不超过系统时间上限**的前提下将所有的乘客送至目的地即可。

电梯每上下运行一层、开关门的时间为固定值，仅在**开关门窗口时间内允许乘客进出。**

示例输入：

```
[2.3]1-FROM-D-2-TO-D-5
[3.6]2-FROM-D-3-TO-D-4
```

示例输出：

```
[ 2.7060]ARRIVE-D-2-4
[ 2.7070]OPEN-D-2-4
[ 2.7100]IN-1-D-2-4
[ 3.1080]CLOSE-D-2-4
[ 3.5120]ARRIVE-D-3-4
[ 3.9120]ARRIVE-D-4-4
[ 4.3130]ARRIVE-D-5-4
[ 4.3140]OPEN-D-5-4
[ 4.3140]OUT-1-D-5-4
[ 4.7140]CLOSE-D-5-4
[ 5.2130]ARRIVE-D-4-4
[ 5.6130]ARRIVE-D-3-4
[ 5.6140]OPEN-D-3-4
[ 5.6140]IN-2-D-3-4
[ 6.0150]CLOSE-D-3-4
[ 6.4160]ARRIVE-D-4-4
[ 6.4160]OPEN-D-4-4
[ 6.4180]OUT-2-D-4-4
[ 6.8170]CLOSE-D-4-4
```

一、Task1

1) 任务说明

- 不存在横向请求
- 同楼座有且仅有一台电梯
- 电梯参数固定

2) 整体结构

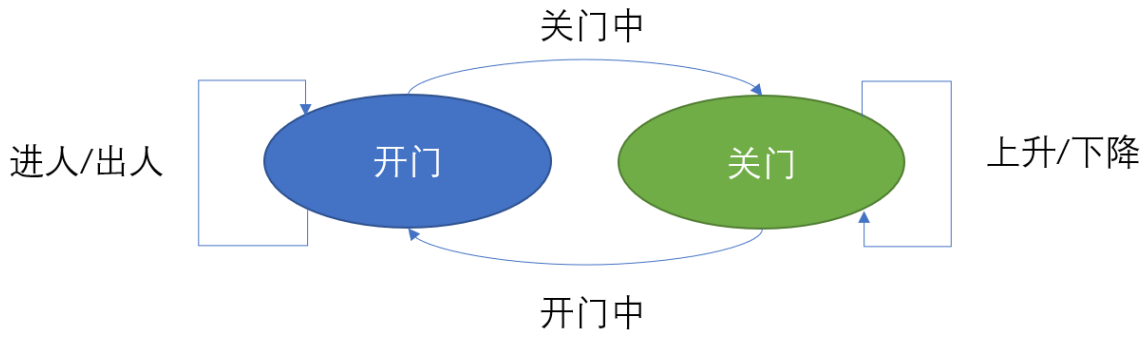
第一次作业选择了最短寻找楼层时间优先(SSTF-Shortest Seek Time First)算法，它注重电梯寻找楼层的优化。

最短寻找楼层时间优先算法选择下一个服务对象的原则是最短寻找楼层的时间。这样请求队列中距当前能够最先到达的楼层的请求信号就是下一个服务对象。

在重载荷的情况下，最短寻找楼层时间优先算法的平均响应时间较短，但响应时间的方差较大，原因是队列中的某些请求可能长时间得不到响应，出现所谓的“饿死”现象。

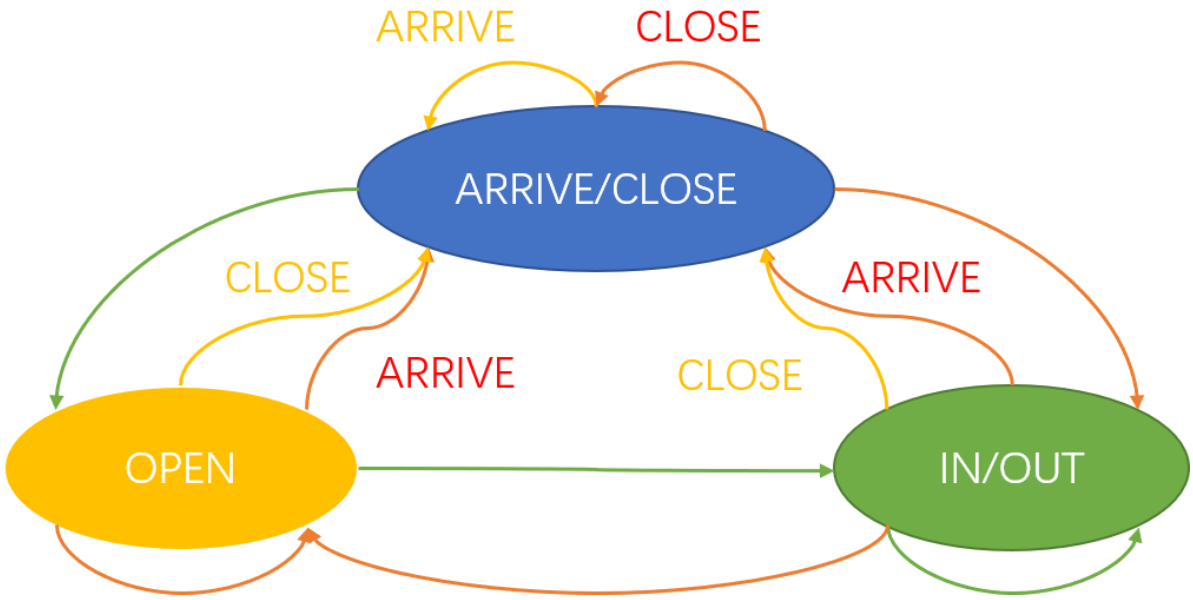
~~以上摘自百科~~

如何实现电梯流程：我选择了状态机的方法。如题目所言，电梯的状态实际上是一个有限自动机。从电梯运行角度看，有且仅有{正在上升/下降，开门中，关门中}。可以进行的操作也就如下所示。



具体证明略（不愧是懒癌的我）

为了便于处理时间，可以构建，基于前一个行动的自动机，如下所示：



不考虑特殊错误（电梯编号和楼座不匹配，满员上人/无人出人，前后楼层不一致）情况下，状态转义图如上，其中绿色表示可以转移，黄色需要判断，红色绝对禁止。当出现错误转移时，视为结果不正确。

以下是本任务的基本逻辑构成，以流程图形式给出。注意到此处多构建了一个调度器类，这一类是第一次作业冗余的，但在后续作业中有用武之地。

三次作业的基本逻辑构成区别不大（主要是第一次搭好了很多），且在第一次作业中，笔者就已经支持部分后续功能，因此在 Task2 及 Task3 中不再赘述，仅叙述重复部分。

UML类图如下所示，部分类中的私有函数与整体逻辑无关，已隐去。

3) 互测体验

强测AC

Hack五次。

- 发现bug：线程不安全，没有逐层输出，输出不安全。
- 事实上，这些发现的bug都是小的理解错误，除了线程不安全外都不涉及逻辑错误。

4) 性能优化

本人很讨厌量子电梯，觉得丧失了美感，于是就给自己懒癌的理由，没用做这个优化。

相应的SSTF算法并没有什么可优化的。因此在强测中得分很差。分析原因主要是大多数学生都使用Look算法，SSTF算法即使平均性能更优秀，在这个评分体系下，会出现大量的99+和85，总成绩反而难看。

个人认为这个性能分是不透明的。在未知数据分布特征的情况下，选择何种算法更优秀是不可预知的。而评分规则有进一步放大了赌博性，最后最好的策略是**使用大多数人使用的算法。**希望明年的课程组可以给出强测数据的分布特征（例如以 X 的频率，投放在[m,n]层之间，方差为Y等等信息）

5) 额外完成任务

- 实现调度器

二、Task2

1) 任务说明

- 允许添加电梯
- 允许横向请求

2) 整体结构

与Task1相同部分不再赘述。

添加的横向请求实际上和没有一样，将横向电梯视为第6第7第15座楼即可。

允许添加电梯则会设计到电梯分配算法，这里有两种不同的思路：

1. 同层电梯共享请求队列，通过算法或者自由竞争实现分配
2. 预分配请求队列，以做到最大效率

目前来看，采用策略1且自由竞争得到的性能分最高。不得不吐槽一下这个得分机制让复杂算法分数远落后于简单算法。

我是策略2，分配策略是优先考虑同向可捎带电梯，如果都不可以捎带，则选择人数最少的电梯，如果电梯人数都为0，则选择最近的电梯。

3) 互测体验

强测AC

Hack零次，被hack零次。

没错，全房都杀不动。。。。

4) 性能优化

前文已述

5) 额外完成任务

无

三、Task3

1) 任务说明

- 添加换乘请求
- 电梯参数可变

2) 整体结构

与Task2相同部分不再赘述。

注意到我并没有对可变参数进行处理。因为考虑到不同速度/载荷电梯，区别并不大，并不会对算法产生质的区别。

不难发现，Task3的架构与Task2基本相同。主要的时序图如下：

3) 互测体验

- 与HW2相同

4) 性能优化

选择加权最短路算法，可能由于权加的不好，最后效果很差。但我想这个思路应该是最好的。

加权为电梯平均速度 * 楼层差 * （1 + 当前人数（包括电梯内和等待中）/ 总载荷） + 开关门

5) 额外完成任务

无额外目标

四、互测

笔者在互测中，使用了测评机和对拍机。但由于主要工作并非本人完成，笔者仅仅是做了一些可用性的辅助，故而不再献拙，在此只提供一些Hack技巧。

评测机暴力Hack，大概率是Hack不下来的。假如这位同学有评测机，那么大概率在A房，地毯式轰炸并不一定能保证效果。对于高手互博，往往功能本身不会出错，只有想怪数据，反常识数据，才可克敌制胜。

既然如此，看代码就是必不可少的。和刀我的同学交流，发现他也是通过看代码才刀到。

据我不完全观察，防御力较低的代码遵循以下规律。大家在日后的互测中也可以针对以下部分着重分析。

- 优化多
- 父子类之间很少调用方法
- 重复代码过多

五、心得感想

~~（坐大牢）~~

第二单元的OO课程，比第一单元还硬核，我学到了许多面向对象编程的方法和思维，更加熟练地掌握了Java语言。当然，这一单元的训练也让我认识到了自己的许多不足，比如思路不清晰，在 Task1 到 Task2 中血泪重构。希望下一单元，可以吸取教训，更上一层楼。