大运村电梯调度优化

15071162 王瑞烨

2018 年 12 月 29 日

建议你在这里查看和下载本项目:https://github.com/wokegrdws/Elevator_Optimization

1 背景介绍

大运村最高的楼层达到 18 层，对于住在高楼层的同学来说，大运村公寓最烦心的事情就是上下楼了。电梯上行的过程中需要在不同的低楼层停留多次，才能到达自己的楼层，下行也同理。

特别是在中午上楼的时候，电梯口会挤满下了课回寝室的同学。不仅电梯内会在上行的过程中很挤，不断在各个楼层停留，而且很多同学甚至挤不上第一班电梯，需要等待下一班。也有同学宁愿会选择步行走上十几层楼梯。中午时分这个拥挤的过程至少会持续 20 分钟，而同学们被浪费的时间是非常多的。

为了减少同学们中午等待电梯时间的浪费,本文对中午高峰期电梯上行的调度用 python 进行了模拟仿真，并提出了优化方案。

2 现状分析

2.1 先决条件

通过对大运村真实情况以及同学们心理的观察，本文将中午高峰期电梯上楼的真实情况简化，抽象出以下先决条件:

- 电梯最低在 1 层，最高在 18 层，所有同学只在 1 层上电梯
- 各个寝室分布在 2 层至 18 层
- 住在第 2、3 层的同学选择不坐电梯

- 同学只愿意去往自己住的楼层，而不愿意去相邻楼层再步行至自己楼层
- 同学去往不同楼层的可能性是均匀分布的
- 电梯每上行一层花费时间 2 秒
- 电梯每停一次（加减速、开门、进出、关门）花费时间 8 秒
- 同学平均在电梯停下后 4 秒离开电梯
- 共有两个电梯，每个电梯每次最多承载 12 人
- 高峰期上行的电梯始终能满载

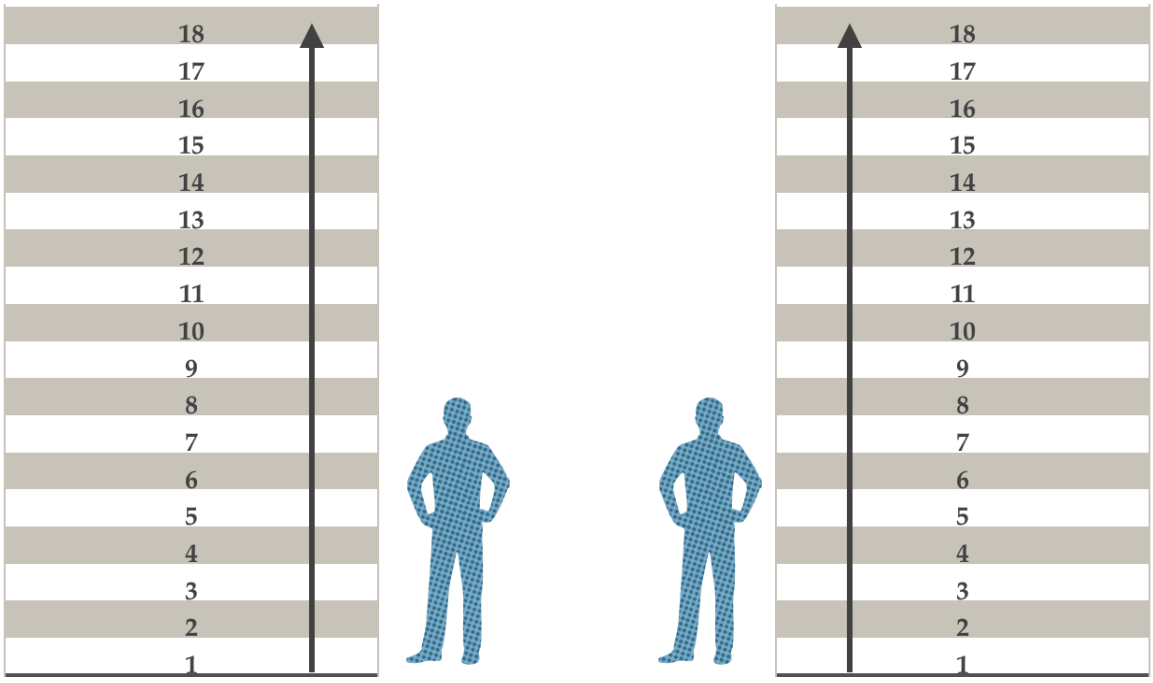

图 1: 现在情况电梯

2.2 仿真实现

将先决条件2.1用程序语言表示。在这里两个电梯是完全一样的，不必分开考虑。通过随机数随机产生每个同学的目标楼层，对每次电梯的上行进行仿真。本文采用每个同学在整个过程中花费时间的平均值来描述某种调度方法的效率。重复以上仿真若干次，可以提高结果的精度。整体算法的伪代码如 Algorithm1。完整源代码见附录A。

Algorithm 1 现状仿真

Input: 仿真次数、电梯满载人数、电梯停留最低层、电梯停留最高层、电梯每层上升时间、电梯停留时间、学生出电梯平均时间

1: **for** $i = 0; i <$ 仿真次数; $i++$ **do**
2: 随机均匀产生每个学生目标楼层
3: 查找每个楼层需要出多少学生
4: 去除重复的楼层，排序得到电梯停留楼层
5: 对比每一个学生的层数和电梯停留的楼层，计算每个学生的等待时间
6: 计算所有学生平均等待时间
7: **end for**
8: 计算所有仿真的平均结果

Output: 仿真的平均结果

这里的输入是：

```
import random
import numpy as np

STUNUM = 12 #
FLOLOW = 4 #       4
FLOHIGH = 18 #       18
TRIP_DURATION = 2 #
STOP_DURATION = 8 #
OUT_DURATION = 4 #
times = 10000 #     10,000
```

每个学生随机均匀目标楼层的产生函数：

```
for i in range(STUNUM):
    stu[i].floor = random.randint(FLOLOW, FLOHIGH) #
```

2.3 仿真结果

由于仿真的次数取了 $times = 10,000$ 次，可以确定结果已经非常接近理论值。如图2，将每次仿真结果用散点图来表示。

```
                    53.78898333333308
```

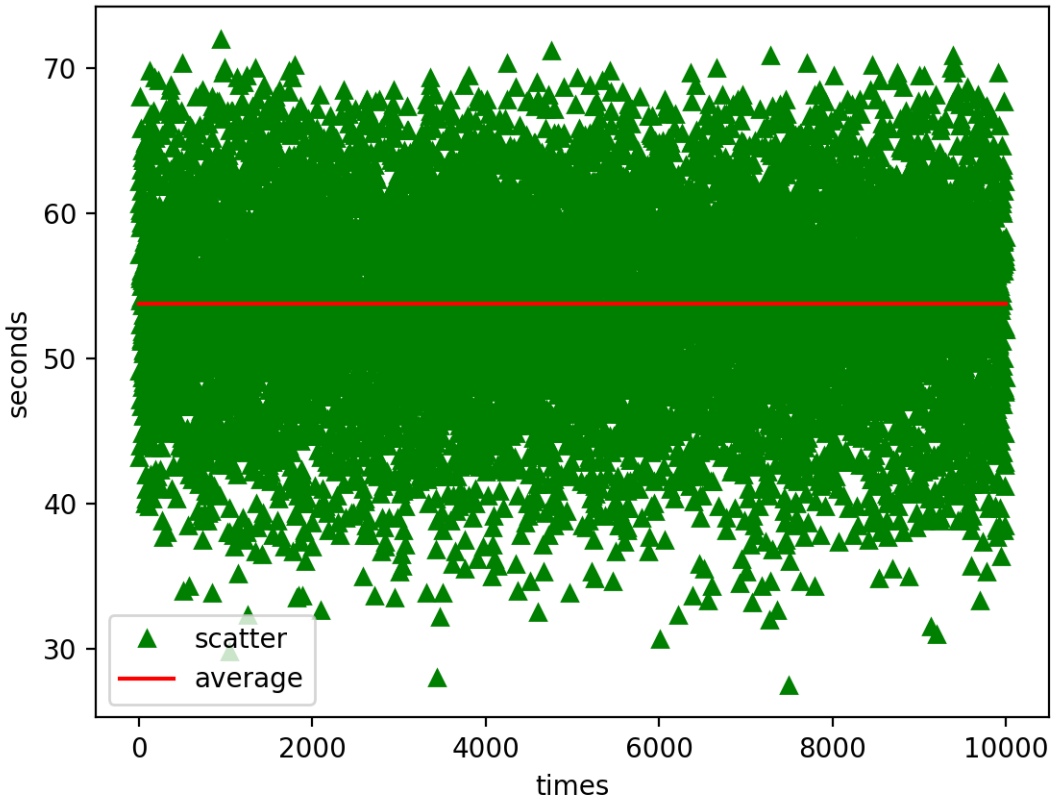

图 2: 现在情况仿真结果

即，未优化前，中午高峰期大运村电梯上行过程中每个学生平均花费的时间是 53.8 秒。我们将以这个未优化的数据为参考标准，衡量下面调度优化方案的效果。

3 单双层优化

本文对两个电梯进行设置：电梯 A 只能在单数层停留，电梯 B 只能在双数层停留。

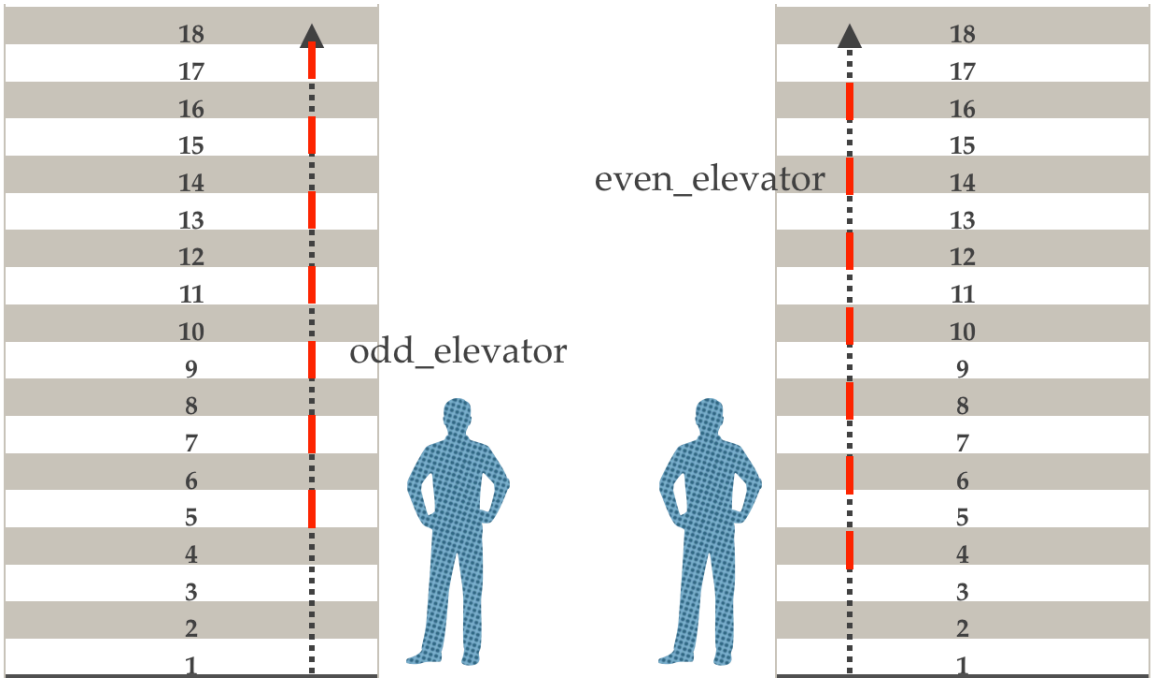

图 3: 单双层情况电梯

3.1 仿真实现

这里将对 A、B 两个电梯分别进行仿真，将分别得出的结果求平均即可作为最终结果。

每个学生随机均匀**单数**目标楼层的产生函数：

```
for i in range(STUNUM):
    stu[i].floor = random.randrange(FLOLOW, FLOHIGH, 2)
    stu[i].floor = stu[i].floor + 1
```

每个学生随机均匀**双数**目标楼层的产生函数：

```
for i in range(STUNUM):
    stu[i].floor = random.randrange(FLOLOW, FLOHIGH + 1, 2)
```

其余不变进行仿真。

3.2 仿真结果

单数电梯的仿真结果是：

```
43.54270000000004
```

双数电梯的仿真结果是：

```
45.633866666666734
```

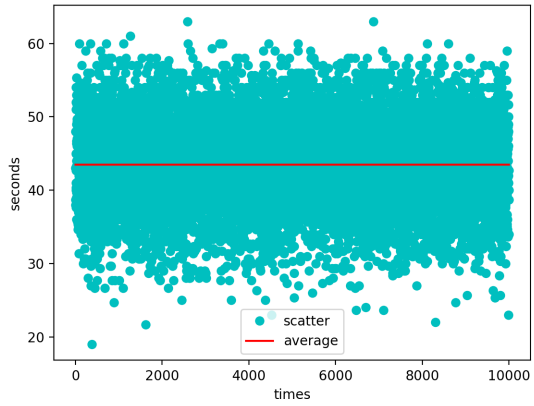

图 4: 单数情况仿真结果

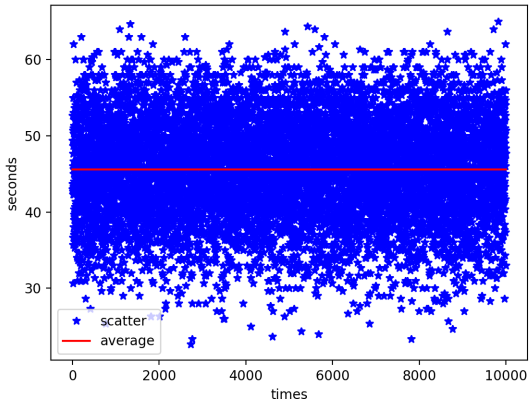

图 5: 双数情况仿真结果

对结果取平均得每个学生平均花费的时间是 44.6 秒。相比未优化前的 53.8 秒节约了 17.1% 的时间。

4 高低层优化

本文对两个电梯进行设置：电梯 A 只能在 4 至 10 层停留，电梯 B 只能在 11 至 18 层停留。

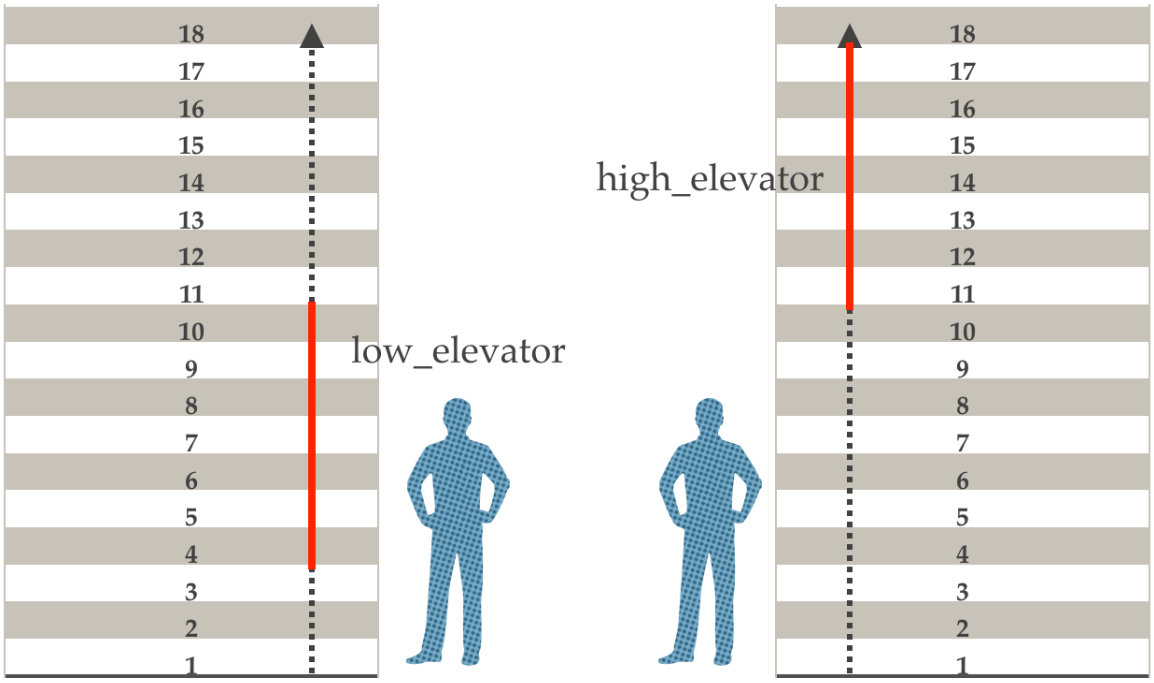

图 6: 高低层情况电梯

4.1 仿真实现

这里将对 A、B 两个电梯分别进行仿真，将分别得出的结果求平均即可作为最终结果。

高层电梯的输入为：

```
FLOLOW = 11 #       11
FLOHIGH = 18 #       18
```

低层电梯的输入为：

```
FLOLOW = 4 #       4
FLOHIGH = 10 #       10
```

其余不变进行仿真。

4.2 仿真结果

高层电梯的仿真结果是：

```
52.58893333333316
```

低层电梯的仿真结果是：

```
35.61570000000019
```

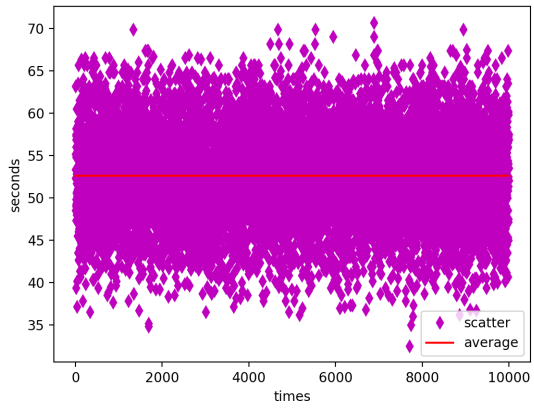

图 7: 高层情况仿真结果

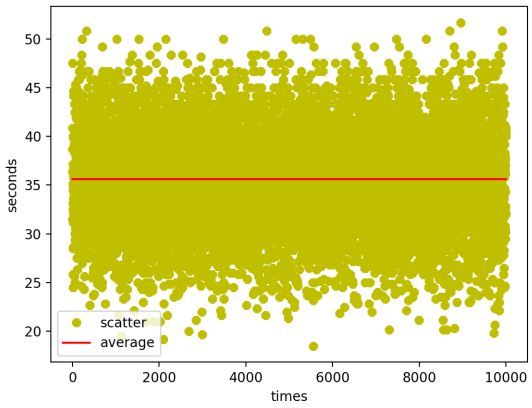

图 8: 低层情况仿真结果

对结果取平均得每个学生平均花费的时间是 44.1 秒。相比未优化前的 53.8 秒节约了 18.0% 的时间。

5 不足及改进

本文的仿真是非常粗略的，主要体现在对现实问题的抽象非常粗略，因此有非常多的不足和非常大的改进空间。主要有以下：

- 大运村不同楼的楼层数有一定的不同
- 需要上楼的同学并不是随机均匀分布在各个楼层的
- 不同楼层学生出电梯速度是不同的，越往高层人越少，速度越快
- 单双层、高低层的优化方案是根据经验得出来的
- 显然存在更好的优化方案，尽管最优方案很难得出
- 如果采用了优化方案，学生可能会选择乘电梯到达相近的楼层
- 尽管是粗略的抽象，也需要严格的数学证明与仿真结果配合

6 结论

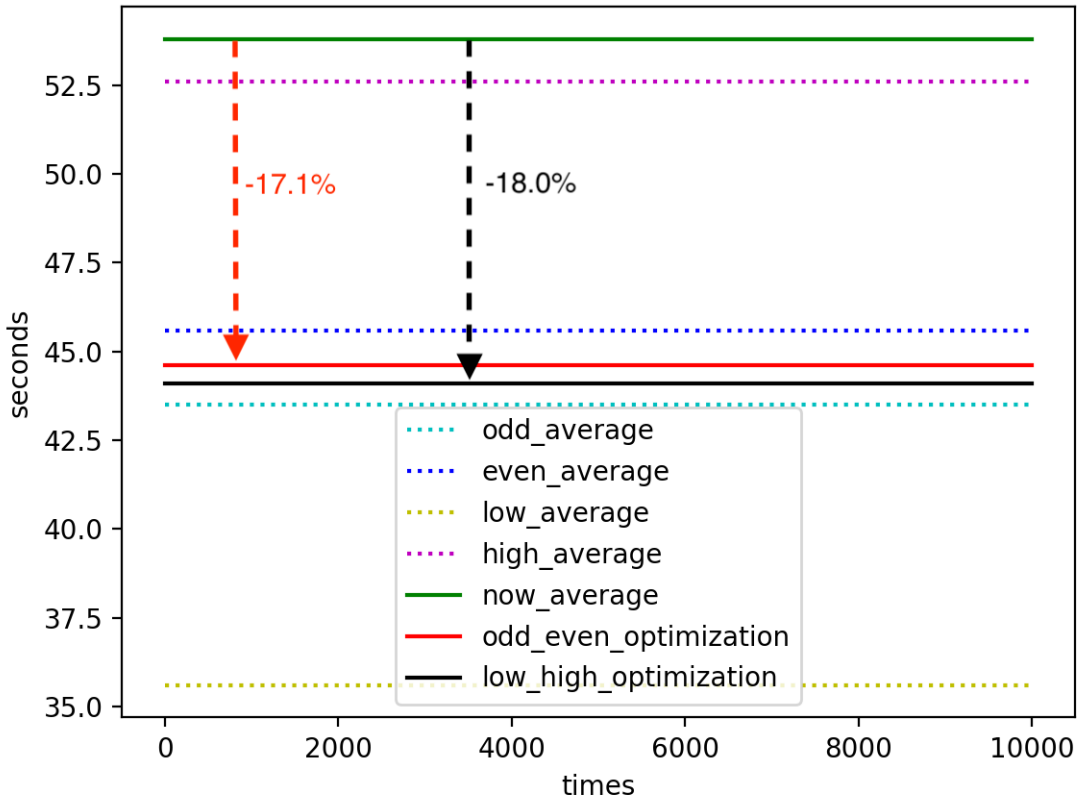

图 9: 各情况比较

本文针对大运村中午高峰期电梯上行的情况，提供了两种经验优化方案，如图9。单双层优化（见3）能为学生节约 17.1% 的时间，高低层优化（见4）能节约 18.0%。显然对于大运村的电梯，有更好的优化方案，也需要严密的数学证明。扩展来看，很多其他地方的电梯同样也需要得到优化。

A 现状仿真源代码

```
import random
import numpy as np

STUNUM = 12 #
FLOLOW = 4 #      4
FLOHIGH = 18 #      18
TRIP_DURATION = 2 #
STOP_DURATION = 8 #
OUT_DURATION = 4 #

#
class Student(object):
    def __init__(self, num, floor, time):
        self.num = num
        self.floor = floor
        self.time = time

#
def deleteDuplicatedElementFromList(list):
        list.sort();
        print("sorted list:%s" % list)
        length = len(list)
        lastItem = list[length - 1]
        for i in range(length - 2,-1,-1):
                currentItem = list[i]
                if currentItem == lastItem:
                        list.remove(currentItem)
                else:
                        lastItem = currentItem
        return list

#
def calculatemean():
    #12
    stu0 = Student(0, random.randint(FLOLOW, FLOHIGH), 0)
    stu1 = Student(1, random.randint(FLOLOW, FLOHIGH), 0)
    stu2 = Student(2, random.randint(FLOLOW, FLOHIGH), 0)
    stu3 = Student(3, random.randint(FLOLOW, FLOHIGH), 0)
    stu4 = Student(4, random.randint(FLOLOW, FLOHIGH), 0)
```

```
stu5 = Student(5, random.randint(FLOLOW, FLOHIGH), 0)
stu6 = Student(6, random.randint(FLOLOW, FLOHIGH), 0)
stu7 = Student(7, random.randint(FLOLOW, FLOHIGH), 0)
stu8 = Student(8, random.randint(FLOLOW, FLOHIGH), 0)
stu9 = Student(9, random.randint(FLOLOW, FLOHIGH), 0)
stu10 = Student(10, random.randint(FLOLOW, FLOHIGH), 0)
stu11 = Student(11, random.randint(FLOLOW, FLOHIGH), 0)

#12
stu = (stu0, stu1, stu2, stu3, stu4, stu5, stu6, stu7, stu8, stu9, stu10,
                                        stu11)

#
list = [stu[0].floor, stu[1].floor, stu[2].floor, stu[3].floor, stu[4].
                                        floor, stu[5].floor, stu[6].floor,
                                        stu[7].floor, stu[8].floor, stu[9].
                                        floor, stu[10].floor, stu[11].floor
                                        ]


#
myset = set(list)
for item in myset:
    print("  %d      %d      " %(item,list.count(item)))

#                                       list
deleteDuplicatedElementFromList(list)

#
print('                ', list)

#                  i                              j

for i in range(STUNUM):
    for j in range(len(list)):
        if stu[i].floor == list[j]:
            stu[i].time = (stu[i].floor - 1) * TRIP_DURATION + (j + 1) *
                                                STOP_DURATION - (
                                                STOP_DURATION -
                                                OUT_DURATION)
    print('  ',stu[i].num,'                  ', stu[i].time)
```

```
    #
    sum_time = 0
    for i in range(STUNUM):
        sum_time = sum_time + stu[i].time

    mean_time = sum_time / STUNUM
    return mean_time

#
add_time = 0
times = 10000
for i in range(times):
    add_time = add_time + calculatemean()
ave_time = add_time / times
print('                    ', ave_time)
```