0 前言

本单元主要讲了契约式编程和JML这种规格语言。但是在本单元结束时回首过去一个月，总感觉学的不如前两单元尽兴，可能是因为U3少了前两单元的刺激，没有那种决定架构时的挥斥方遒，而更多的是实现他人给我的需求，可能这才是将来工作的常态吧。不过不管怎么说，确实对契约式编程有了更多的理解。

1 测试过程

1.1 黑盒测试 & 白盒测试

我理解的黑盒测试是测试者不了解所测试程序具体是如何实现的，只能通过提供符合一定约束的输入然后来对输出进行合法性判断来确定所测试程序是否有bug。

白盒测试则是测试者需要去了解被测试程序具体的实现，需要测试者对源代码有更深层的了解。

很显然，白盒测试是更容易发现代码bug的(~~这么说可能不太负责，我感觉对于大型项目而言白盒测试并不一定更容易发现bug，因为白盒测试常常只会着眼于局部实现，有可能有些需求没有实现但是白盒测试发现不了,但是毕竟我们的作业还是挺简单的，很少出现这种情况~~)，但是却需要花费更多的时间成本与精力成本。

在这单元中，这两种测试方法我都用了。具体而言，我在互测时使用评测机来寻找同房间同学的bug，这属于黑盒测试（~~毕竟我基本没有认真看他们的代码~~）；在写完代码后，我与我的好朋友魏佬通过代码走查的方式来互相寻找bug，这属于白盒测试。

1.2 对单元测试等多种测试的理解

单元测试

对于单元测试，维基百科给出的解释是单元测试是针对软件设计的最小单位进行正确性检验的测试工作。我想对于我们所学习的面向对象而言，其实就是对于单个方法进行测试。前几次作业要求的Junit其实就是单元测试的一种方法。

功能测试

功能测试是对软件产品的各功能进行验证，确保系统执行用户期望它所执行的工作。对应到我们的作业中，就是测试群发消息等功能是否正确。

> **note**
> 很多时候，软件开发好比建筑房屋。
> 单元测试好比房屋建筑现场的建筑监理员。他关心房屋的各个内部系统，如地基、构架、供电系统和管道设备等。他确保（测试）房屋每一部分的工作都安全、正常，即符合建筑说明。
> 功能测试类似于视察同一建筑现场的房主。他假定内部系统将正常运作，并假定建筑监理员在执行其任务。房主关心的是住在这所房子里将会怎样。他关心房子的外观如何，各个房间的大小是否合适，房子能否满足家庭的需要，以及窗户的位置是否有利于采光。
> 总结来看其实就是房主对房子执行功能测试。他从用户的角度考虑问题。建筑监理员对房子执行单元测试。他从建筑工人的角度考虑问题。

集成测试

有时候方法也就是单元独立测试时是正确的，但是组合起来一起工作时就会出现bug，而集成测试就是对这种情况进行测试的。

压力测试

压力测试就是构建极端数据来对软件进行测试，以确定整个系统的稳定性。在这门课程中，压力测试是我们经常使用的。因为强测能否取得一个好成绩很大程度上取决于结果是否正确以及是否会tle，第一个很容易确保，但是否会tle则需要通过手动构造一些极端数据来进行测试。

回归测试

回归测试就是当我们修改代码后再将之前的测试数据全部跑一边来检验是否产生新的错误。我们的bug修复环节其实就是一种回归测试。

1.3 数据构造

主要有以下几点策略：

- 构建评测机，并提高所构建图的稠密度
- 手动构造一些极端数据
- 增大数据规模，比如课程组限制数据在1w条以内，但是我们自己做测试时可以提高到10w条，这样可以使得测试更加全面。

2 架构设计

总体上我是采用邻接图来构建的图，然后主要使用HashMap或者HashSet容器来对数据进行管理，架构则是与JML基本保持一致，不必多谈。

具体的维护策略我们在下一节中再谈。

3 性能优化

由于我在第三单元强测互测中均未出现bug，所以再次不提修复情况，而重点谈一下对规格与实现分离的理解以及性能优化的措施。

我认为JML虽然说了一堆，看起来也给了具体的实现方法，但是这实际上只是一种对接口的约束。当需求者提出需求时，如果直接将需求用自然语言转交给编程者，是很容易出现问题的，这也就是为什么荣老师经常说理解客户的需求是很困难的，这是为什么呢？我想是因为不同的人对自然语言的理解是不一样的，也就是说自然语言是不严密的。所以我们需要JML这种规格语言来作为桥梁，传达需求者的需求。所以JML只是对接口的约束，需求者将需求转化为规格，编程者从规格中理解需求，并将其实现，也就是规格应当与实现相分离。那么自然地我们不应当直接翻译JML，~~要不然要程序员有啥用呢~~，而应当采用一些合理优秀的性能优化措施，这里我将自己采取的性能优化的手段列举如下：

- 选择合适的数据结构：
  - 对于以ID做索引的且ID唯一的数据类型，可以使用HashMap来进行管理；
  - HW11中Person对象内需要按序保存Message，并且还会有大量的删除操作以及插入操作，所以应当使用LinkedList这种底层是链表的数据结构进行管理；
- 选择合适的算法：
  - 对于`query_shortest_path`指令，即查询非加权图两点最短路径问题，可以使用bfs或者双向bfs或者使用堆优化的dijkstra算法进行查询。
  - 对于`query_block_sum`指令以及`query_circle`指令，可以使用路径压缩+按秩合并的并查集来进行查询。

    - 对于`query_best_acquaintance`以及`query_couple_sum`指令，可以使用堆排序、红黑树来维护某人的最好的朋友。
- 在合适的地方采用动态维护：对于`query_triple_sum`、`query_block_sum`、`query_tag_value_sum`、`query_tag_age_var`、`query_social_value`等指令，可以使用动态维护来减小时间开销。
- 在合适的地方使用`lazy tag`延迟维护：一般而言，我们会在每次删除关系时重建并查集，但是这其实是没有必要的，我们可以设置个`dirty`位来记录是否需要重建并查集，当需要用并查集进行查询时再进行查询。
- 采用合适的计算公式：对于`query_tag_age_var`指令，我们没必要用JML给出的计算公式，我们可以将公式化简成下式：

```
return (agePowSum - 2 * ageMean * ageSum + num * ageMean * ageMean) / num;
```

读者可能发现我说了好几个合适，这是因为我认为对于软件开发而言，性能优化符合需求即可，没必要“登峰造极”，更不要过度开发。

4 Junit测试

4.1 与规格信息相结合

在这单元中，我的Junit测试可以分为这么几步：

- 实例化对象并初始化一些较简单的指令；
- 搭建数据生成器，生成大量随机数据；
- 将翻译JML得到的标准答案与执行指令得到的答案进行比较；
- 检查执行指令前后是否符合JML中的ensures等要求。

值得一提的是我最后一次作业的Junit竟然写了1000行，后来与同学们进行比较，感觉自己写的测试代码多了好多没必要的部分，尤其是数据生成器写的过于臃肿了。

4.2 Junit效果

Junit对于代码实现与规格的一致性的检验以及单元测试有一些作用，但是我们的项目实在太小了，Junit很难有用武之地，只是检验正确性的话完全比不过用Python搭的评测机，所以我感觉写Junit付出的精力所能获得的收获似乎只有通过中测以及学会写简单的Junit测试，感觉这样或许与课程初心相背离。我感觉以后课程组可以在理论课上举一些使用Junit对大型项目的核心代码进行测试的例子，这样可能会有比较好的效果。

学习体会

本单元我学习了契约式编程以及JML这种规格语言，契约式编程我早有耳闻并且也极为赞同，但我认为契约式编程应当在小组合作中才能得到真正的训练，JML规格语言确实是第一次学到，希望将来能够用到或者能够帮助我对其他规格语言的学习。