# 520\_Week 01 学习总结

第一部分，预习内容补充

首先，我是两天前看完课程的，现在来写总结的时候，脑海中仍然能清晰呈现出来老师在课堂上反复强调的内容（从预习周开始就反复强调的内容）：一个误区，两个优化方法——误区：刷题只刷一遍（应该刷至少五遍，五毒神掌）、两个优化方法：升维和空间换时间，或许是因为我刚起床，脑袋比较清醒^\_^。老师在课堂上重复的时候，提到说，“我一点也不觉得重复”，我是完全赞同的，这样的重复是很有效的。

所以我想说，老师这样的教学方式是很有成效的，让我消除了对算法题目的畏惧与恐惧。我本科是软件工程和电气工程双专业，本科的时候曾经想要参加算法竞赛之类的活动，但因为种种原因放弃了，但也曾经经历过刷题等阶段。如今硕士阶段面临找工作的节点，没理由放弃了，希望能在老师这醍醐灌顶的新方法之下，像之前备考雅思时候学习英语的时候那般，重复加思考，真正战胜未知和恐惧，让我的短板（或者大多数人觉得会是自己短板的数据结构和算法），转换成我的长处。

回归正题，首先是老师上课提到的五毒神掌的具体内容：

|  |  |
| --- | --- |
| 第一遍刷题 | 读题+思考，如果没有思路马上看题解！看多种解法，比较方法之间的优劣。背诵，默写好的方法。背诵和默写很重要 |
| 第二遍刷题 | 背诵之后自己默写，最好可以比较不同的解法之间的优劣 |
| 第三遍刷题 | 过了一天之后，重复练习，根据对不同解法的熟悉程度，进行专项练习 |
| 第四遍刷题 | 过了一周之后，反复练习之前的题目 |
| 第五遍刷题 | 面试之前一周，进行恢复性训练 |

和职业电竞选手训练类似，成为算法方面的职业选手，只需要进行一些跳出舒适区的训练的，这些训练内容一般是针对某一个点进行突破的。可能有一些反人性（学习本来就是反人性的），但是只有这样，才能拉开和业余选手的差距。

共勉。

刷题技巧切题四件套：

|  |  |
| --- | --- |
| 第一步：Clarification | 搞清楚题目具体问什么，如果碰上难题，题目本身很可能留存有具体细节的讨论和确定空间，这时候一定要和面试官交流，搞清楚题目需求。 |
| 第二步：Possible solutions  -Compare (time/space)  -optimal (加强) | 去想所有可能的解法。  这里其实是有套路的，就像覃超老师开学典礼分享他的面试经验，以及课上讲的方法一样，先考虑暴力法，提出来能解决问题的最简单的方案，然后再考虑从空间和时间复杂度上去优化。主要是时间复杂度上的优化。 |
| 第三步：Coding（多写） | 写代码，平时通过白板编程多写，这样面试的时候才不慌 |
| 第四步：Test cases | 写完代码之后自己想一些测试用例 |

第二部分，第一周内容

本周学习了数组、链表、跳表、栈、队列、优先队列、双端队列

有几道题留下较深刻印象：

第十五题，课上讲三种解法，第一种方法，暴力求解，用三重循环，复杂度为O(n^3)。

第二种方法，用哈希表，可以把复杂度降到O(n^2)

第三种方法，用左右下标夹逼的方法。这种方法其实是非常tricky，非常难以想象到的。双指针的方法很考验编码能力。

这种双指针的题目一定要做得滚瓜乱熟。虽然双指针理解起来比较晦涩，但是做熟练之后，会发现这种方法就是一种定式。

如果这道题，面试的时候，能把暴力求解、哈希表、双指针这三种方法都先概述出来，时间复杂度也能分析对，同时一气呵成地写出标准题解的20行的java代码，那么你会是最顶级的候选人，能拿到比较大的package。

分析Queue和Priority Queue源码。

Queue

Queue在java中是一个**接口**，具体实现方法是由实现类来写的。实际上它有比如LinkedList, PriorityQueue等许多实现类。

Queue**继承**了Collection接口，所以有遍历数据，判空的操作，还有一系列对Stream的相关操作。

Queue接口中定义了6个需要实现的方法：

|  |  |
| --- | --- |
| 方法名 | 作用 |
| boolean add(E e) | 将参数插入队列，插入成功返回true，如果插入失败返回false，但是基本没有返回机会因为会抛出异常（IllegalStateException） |
| boolean offer(E e) | 插入成功和add类似，但是插入失败不会抛出异常而是返回false |
| E remove() | 检索同时删除队列的头元素，若队列为空则抛出异常（NoSuchElementException） |
| E poll() | 和remove类似，检索同时删除队列的头元素，但是如果队列为空不会抛出异常，而是返回null |
| E element() | 仅仅检索队列的头元素但不会删除它。如果队列为空，抛出异常（NoSuchElementException） |
| E Peek() | 和element类似，仅仅检索队列的头元素但不会删除它。但如果队列为空，不抛异常而是返回null |

Priority Queue源码

优先队列会给每一个元素赋以权重，然后按照这个权重的大小进行排序，然后每次从队列中获取的元素都是队列中权重最大或者最小的元素。

源码内容：

// 应该是版本号

*private static final long* serialVersionUID = -7720805057305804111L;

// 默认容量大小为11  
*private static final int* DEFAULT\_INITIAL\_CAPACITY = 11;

// 存储元素

*transient* Object[] queue;

// 默认长度为0，即默认是空的

*private int* size = 0;

// 用来给队列元素排序的比较器

*private final Comparator*<? *super* E> comparator;

// 存储元素被修改的次数

*transient int* modCount = 0;

// 这个队列能被分配的最大空间（maximum size of array to allocate）

*private static final int* MAX\_ARRAY\_SIZE = Integer.MAX\_VALUE - 8;

具体方法：

省略所有构造方法。

*//给优先队列增加容量的方法(官方：Increase the capacity of the array)*

*private void* grow(*int* minCapacity) {

*int* oldCapacity = queue.length;

//从这里可以看出，如果队列长度小于64，那么每次扩容一倍，也就是增加到原来的2倍。如果队列长度已经大于64，每次增加原来的一半，也就是增加到原来的1.5倍  
 *// Double size if small; else grow by 50%  
 int* newCapacity = oldCapacity + ((oldCapacity < 64) ?  
 (oldCapacity + 2) :  
 (oldCapacity >> 1));  
 *// overflow-conscious code  
 if* (newCapacity - MAX\_ARRAY\_SIZE > 0)  
 newCapacity = *hugeCapacity*(minCapacity);  
 queue = Arrays.*copyOf*(queue, newCapacity);  
}

*// add使用了offer 方法*

*public boolean* add(E e) {  
 *return* offer(e);  
}

*//给优先队列插入元素*

*public boolean* offer(E e) {

//判断插入目标是否为空，否则空指针异常  
 *if* (e == *null*)  
 *throw new* NullPointerException();  
 modCount++;  
 *int* i = size;

//若队列长度比要插入的元素个数还多，进行扩容  
 *if* (i >= queue.length)  
 grow(i + 1);  
 size = i + 1;  
 *if* (i == 0)  
 queue[0] = e;  
 *else*

*//很重要，这里看到了插入逻辑的实现函数，是siftUp* siftUp(i, e);  
 *return true*;  
}

那接下来就看看siftUp

*//插入操作的主要实现部分*

*private void* siftUp(*int* k, E x) {  
 *if* (comparator != *null*) //如果有comparator，就使用自定义的比较器，否则一般情况使用默认的siftUpComparable比较器  
 siftUpUsingComparator(k, x);  
 *else* siftUpComparable(k, x);  
}

//默认比较器的插入元素的算法  
@SuppressWarnings("unchecked")  
*private void* siftUpComparable(*int* k, E x) {  
 *Comparable*<? *super* E> key = (*Comparable*<? *super* E>) x;  
 *while* (k > 0) {

//为了获取父元素下标，无符号右移一位  
 *int* parent = (k - 1) >>> 1;  
 Object e = queue[parent];

//优先队列需要把权重大的元素放在上面，所以通过判断把权重大的元素放在最上面。如果当前元素的权重大，不用做，break。反之，把父元素覆盖掉当前元素  
 *if* (key.compareTo((E) e) >= 0)  
 *break*;  
 queue[k] = e;  
 k = parent;  
 }

//k为上述迭代后最终位置  
 queue[k] = key;  
}

通过三重位移分析得知，java中的优先队列时使用二叉堆来实现的。

下面分析删除功能相关的代码。

*//从队列中删除元素，使用removeAt方法*

*public boolean* remove(Object o) {  
 *int* i = indexOf(o);  
 *if* (i == -1)  
 *return false*;  
 *else* {  
 removeAt(i);  
 *return true*;  
 }  
}

@SuppressWarnings("unchecked")

*private* E removeAt(*int* i) {  
 *// assert i >= 0 && i < size;* modCount++;  
 *int* s = --size;  
 *if* (s == i) *// removed last element* queue[i] = *null*;  
 *else* {  
 E moved = (E) queue[s];  
 queue[s] = *null*;  
 siftDown(i, moved);  
 *if* (queue[i] == moved) {  
 siftUp(i, moved);  
 *if* (queue[i] != moved)  
 *return* moved;  
 }  
 }  
 *return null*;  
}

和remove功能类似的poll方法

//检索并删除优先队列的头部，空的话返回Null而不报错。

@SuppressWarnings("unchecked")  
*public* E poll() {  
 *if* (size == 0)  
 *return null*;  
 *int* s = --size;  
 modCount++;  
 E result = (E) queue[0];  
 E x = (E) queue[s];  
 queue[s] = *null*;  
 *if* (s != 0)

//关键操作，为siftDown  
 siftDown(0, x);  
 *return* result;  
}

//仅查看元素而不出队的方法

@SuppressWarnings("unchecked")  
*public* E peek() {  
 *return* (size == 0) ? *null* : (E) queue[0];  
}  
  
//查找元素位置的方法indexOf

*private int* indexOf(Object o) {  
 *if* (o != *null*) {  
 *for* (*int* i = 0; i < size; i++)  
 *if* (o.equals(queue[i]))  
 *return* i;  
 }  
 *return* -1;  
}

分析总结：

整个使用过程中插入删除操作还是最多被使用的。Java中优先队列通过堆来实现，值得学习。

Code Review

005 - 或许你是市面上很抢手的PHP人才，PHP实现算法的样本代码挺少的

495–从你的代码和分析中学到了很多，你是我的榜样。你提交了很多题，看来很有经验，膜拜一下大神！

100 – 源码分析部分很有干货，值得学习，解题代码注释很多，方法很多，也有分析，看得出来很认真。

295 – 提交了很多题目，完成得很认真，可以适当增加一些注释和思考

115 – 完成了很多题目，认真态度值得学习，甚至把leetCode上面运行的结果贴出来了（虽然覃超老师在群里也提了说这个数据其实只能参考）