# 520 第三周学习总结

本周学习了深搜，广搜，贪心，二分查找。

老师反复强调，我们现在写的算法实际上都可以归纳为最简单的逻辑（判断，循环，递归），不牵扯到更加复杂的人工智能神经网络相关。

在斯坦福大学的机器学习课程中，其实就是从搜索开始的，之后讲到了启发式搜索，再然后引入了神经网络。

总之，我们考虑的搜索大多数情况下处理的都是简单朴素的暴力搜索，没有智能情况的考虑。也就是说更多情况下要做的就是每个节点都访问一次。

广搜和深搜确实在本科期间就有学习过，但是当时要求的目标只是能了解怎么算的而对代码没有要求。研究生阶段也没有怎么涉及，现在是需要弄得更明白了。深搜一般使用方法是递归，非递归的话当然可以用栈去模拟。而广搜则一般就是用队列去模拟。

二叉树的层次遍历是经典问题，超哥说是硅谷面试排名前三的题目，重要程度不言而喻，代码个人感觉初学起来还是比较难的，需要多次反复啃。

岛屿题目之前面试有碰到过，当时是完全懵逼状态，看来当时面试没通过是很正常的……吃一堑长一智吧，在哪里跌倒得爬起来，不然不就成了王八了么？岛屿的题目看到很有意义的思维方式：“flood fill”，把某一类的扫描到之后全部变成另外一类，在扫雷等经典场景中也经常有使用到。

总之，关于广搜和深搜，本质就是递归，循环，找重复性，多练了之后发现它们就是在找重复性。

话说，超哥在这里传达了“神”论，作为一个共产党员，我肯定还是持保留意见的。

贪心算法有些抽象，应用场景一般来说也没有动规多。因为如果要是用傻贪心，你必须证明当前方法得到的就是最优解，而这种情况往往都必须伴随着假设条件（比如课上讲到的硬币问题，如果要用傻贪心，需要增加硬币币值的倍数关系的条件），而在实际面试中使用的可能较少。

贪心的一个变化，是一般都会思考从前往后贪心，而如果反过来思考，从后往前贪心，则可能更方便地解决一些问题，比如课上讲到的跨步子问题。

二分查找，有三个前提且需要形成肌肉式记忆：

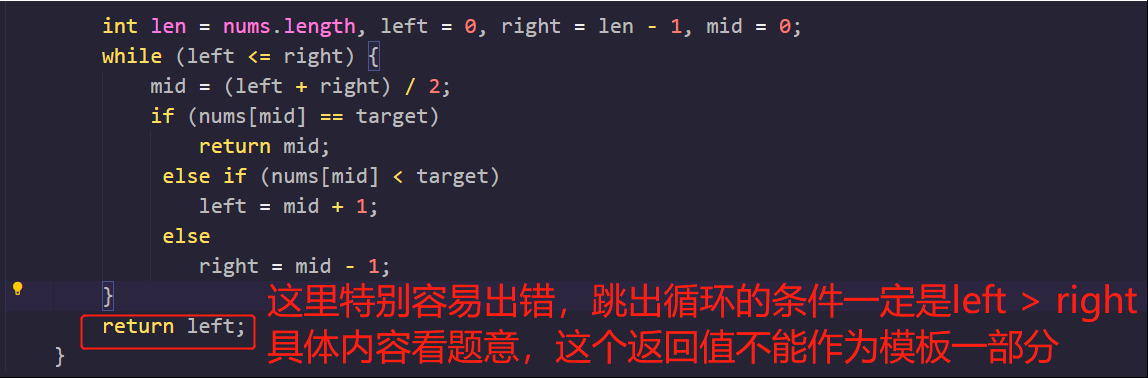
1. 目标函数存在单调性（单调递增或者单调递减）
2. 存在上下界（bounded）
3. 能够通过索引访问（index accessible）

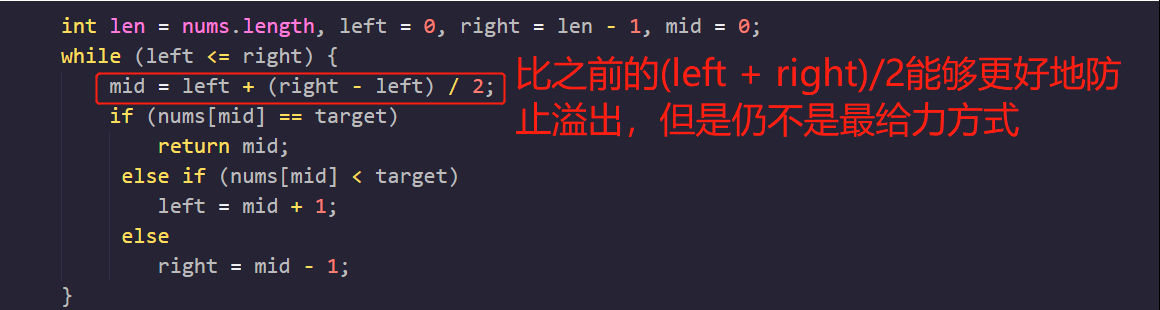
第三点也是指出，如果目标是链表，可能不太适合用二分查找，因为不能够直接通过索引访问。

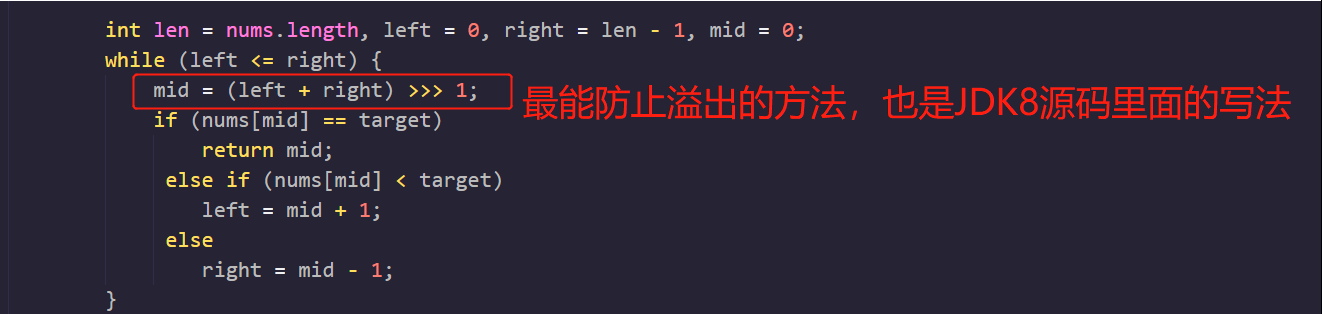
老师课上讲解了一个二分查找的代码模板：

计算机生成了可选文字:
代码模版
left,right=0/len(array)
whileleft
right
mid=(left
right)
ifarray[mid]
=target
#findthetarget!！
breakOrreturnresult
elifarray[mid]<target
left=mid
else
right=mid

根据35这道题，我在小程度上完善了这个二分查找的代码模板：







但是，集思广益，我在LeetCode上面看到了以为童鞋的题解和总结，确实写得很好也有收获，就是通篇对二分查找进行了解析。

二分查找的关键是循环的条件。如果设置的是left <= right，在跳出循环之后具体返回什么值这点上，比较容易搞错。所以这位同学写出的“二分查找神奇模板”，解决的方式就是判断条件直接写成 left < right，这样跳出循环的条件一定是left == right。如果这样可能漏掉left == right但是又满足题目需求的情况呢？可以单独对这种情况进行一次追加的判断就可以了。

作业，使用二分查找，寻找一个半有序数组[4, 5, 6, 7, 0, 1, 2]中间无序的地方

解答：

按照题意，需要找到变化的点的元素

这里要查找的是这个数组中变化的元素，也就是变化的点。我们把待求的变化后的数组成为a，可以总结出来这个点有两个特性：

1.该点左边所有的元素 > a的最后一个元素

2.该点右边所有的元素 < a的第一个元素

所以在进行二分查找时，与mid比较的时候，会有：

if (nums[mid] > nums[right]) -> mid落在了左半边，应该要往右移动，left = mid+1

if (nums[mid] < nums[right]) -> mid落在了右半边，应该要往左移动, right = mid-1

public int Solution(int[] nums) {

int len = nums.length;

int left = 0, right = len-1, mid = 0;

while(left < right) {

//位运算写法: mid = (left + right) >>>1;

mid = left + (right - left)/2 ;

if(nums[mid] > nums[right]) {

left = mid+1;

} else {

right = mid - 1;

}

}

return left;

}