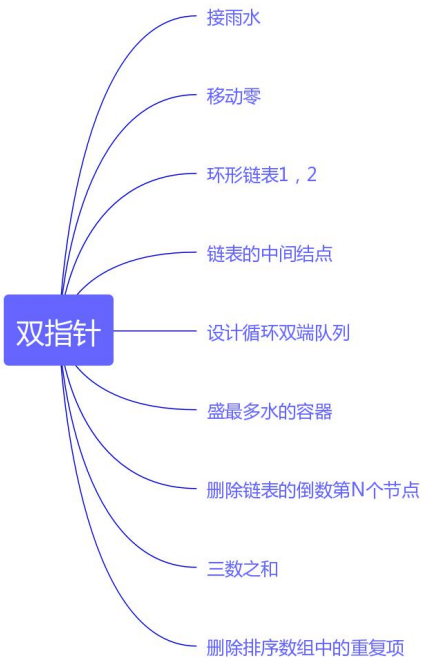
对于每周的学习总结，我想分两部分来进行，一部分是对本周所学知识的简单总结，第二部分是具体到某个题目，对自己做题中的代码的一些总结与反思。
PS：由于没用过 JAVA，就没有做那两个 JAVA 相关作业。
这里总结了本周所学的方法，所练习的题目，以及用这些方法可以解决的题目类型和解决某个题目要用什么办法。把它们作为解题的关键词，通过想到这些关键词来迅速回忆起完整的解题方法和过程。
1. 两数之和：哈希表(一遍)
11. 盛最多水的容器:双指针(夹逼)
15.三数之和：排序+双指针(夹逼)
19. 删除链表的倒数第 N 个节点：双指针(固定间隔)
20.有效的括号：栈(洋葱)
21.合并两个有序链表：迭代比较
26.删除排序数组中的重复项：双指针
42.接雨水：双指针(夹逼)(left and right)，依次找夹逼内的矩形作为底部高度
66.加一:取模，进位
70.爬楼梯：斐波那契数
84 柱状图中最大的矩形:栈
141.环形链表：双指针(快慢)
142 环形链表 2：双指针(快慢) + 双指针(固定间隔)
155.最小栈：使用两个栈（辅助栈）
189.旋转数组：环状替换；使用反转；（多做几遍）
206 反转链表：左->右依次迭代 or 递归
239.滑动窗口最大值:双向队列(单调队列)
283.移动零:双指针(快慢)
641.设计循环双端队列：双指针，注意判别队列为空和为满的条件不要冲突。
876.链表的中间结点：双指针(固定间隔)

可以看到双指针的题目是最多的，这里对双指针单独拉出来做个总结。此外还有想到括号就想到洋葱就想到栈，看到滑动窗口就想到双向单调队列这些固定套路。



本周的练习过程中，对双指针类型的题目练习颇多，熟练度++，掌握进度++，对于其余类型的题目，如反转链表，反而五天后回过头来再看有所生疏。这也告诫了自己不时要多回头看看做过的题目，熟练运用五毒神掌，并根据自身情况毒性可略加一些(多回顾几次)，练出符合自身特色的五毒神功。

26. 删除排序数组中的重复项

第一版:

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.empty())
            return 0;
        auto tail = nums.size() - 1;
        vector<int>::iterator iter = nums.end() - 1;
        for (auto i = tail; i > 0; i--){
            if (nums[i - 1] == nums[i])
                nums.erase(iter - 1);
            iter--;
        }
        return nums.size();
    }
};
```

这道题目在最一开始是采用了从后往前迭代的办法，从后往前迭代是当时考虑到数组中删除元素需要把该元素之后的所有元素往前移，所以从后往前迭代。但是后面看了题解中的解法后再思索，发现其实从后往前迭代尽管能减少一些时间，但是迭代到最后，每次删除还是要

移动大量的元素(比从前往后迭代略少)。但总体来说,for 循环的时间复杂度是 O(n)的，循环内的删除操作也是 O(n)的，总体是 $O(n^2)$的，真是糟糕的做法....

第二版:

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        if (nums.empty()) return 0;
        int first = 0,second = 1;
        for (; second < nums.size(); second++)
            nums[first] != nums[second] ? nums[++first] = nums[second] : first;
        return first + 1;
    }
};
```

考虑到删除操作的时间复杂度过高，思考有没有什么办法替换删除操作。考虑到数组中随机访问一个元素及修改它的时间复杂度是 O(1)的，再加上数组是排序好的，使用双指针是个很好的办法（看了题解，大雾....)。指针 i 与指针 j，一开始指针 i 在位置 0，指针 j 在位置 1，nums[i]等于 nums[j]时，i 不动，j 向前，直至 nums[i] != nums[j]，此时 nums[i+1]被指针 2 替换。(不是替换 nums[i],要保留重复元素的一个值)这个操作的时间复杂度是 O(1)，所以整体时间复杂度为 O(n)。

189. 旋转数组

汲取上题经验，牢记数组随机访问的时间复杂度为 O(1)的特性。初看此题，第一反应是考虑边界条件，第二反应是 k 有可能>数组 size，需要取模，第三反应是下标 i+k 有可能超出数组 size，也需要取模。做完题之后回过来看，我觉得我最基本的这三个考量是没问题的。

自己一开始的想法是用 temp 存储 nums[(i + k) % len]的，然后用上一轮存储的 temp 替换掉 nums[(i + k) % len]，然后 nums[i+1],nums[i+2]依次进行处理。但是很快就发现了愚蠢的地方，因为是依次处理 nums[i]，temp 存储的值并没有派上用处。看了题解之后才发现，与我思路相近的应该是题解中的“环状替换”。

环状替换核心：1.替换总次数为 n 次

2.n%（k%n) ==0 时，会在无法遍历所有数字的情况下回到出发数字，此时应从下一个数字开始再重复相同的过程。那么第一轮所有移动数字的下标 i 满足 i%k==0，因为我们每一步的步长都为 k，我们只会到达相距为 k 个位置下标的元素。每一轮移动的元素数量都是 n/k 个（理解为 n 里有几个 k)，下一轮会移动满足 i%k==1 的元素，直到再次 i%k==0，形成闭环。

3.当 n%（k%n) ！=0 时，只走一轮就可。

4.由 2，3 可得循环跳出条件为 count < nums.size()

写到这里我发现把每个题的详细思考都写出来跟写一篇题解差不多了....就先写这两道吧，其余的之后再总结