## 1. 数组

### 二维矩阵：

注意m行n列 先行后列

### 总结：

遇到有序的问题可以想一下二分查找

遇到排序的问题可以试试从后面开始会不会简单很多

## 2. 链表

参考: <https://www.cnblogs.com/ly-leah/p/13700622.html> 链表刷题总结

* 朴素解法，head = head.next遍历链表来解决问题；（创建一个新链表，只需要创建一个头节点，然后head.next = ListNode(val)即可）
* 双指针，甚至是三指针，在很多的链表题中发挥很大的作用；
* 快慢指针，快指针和慢指针以不同的速度同时遍历链表；
* 递归，大多数链表题都能用递归来解决；
* 哑节点dummy，或者是哨兵节点，可以简化链表的极端操作；

### 双指针：

对于链表的题目，双指针的应用太多了。不论是删除链表的倒数第N个节点，还是回文链表，都会用到双指针。因为链表的随机访问效率差，当我们想要处于某个位置的节点的时，都需要去从头遍历，所以使用额外的指针来记录节点位置是非常有必要的。

双指针还是用 while 更清晰一点

例如：

83. 删除排序链表中的重复元素

82. 删除排序链表中的重复元素 II

### a. 快慢指针：

判断链表中是否有环，如果有环，那么快慢指针一定会相遇的

### b. 哨兵节点：

设定一个哨兵节点 dummy ，这可以在最后让我们比较容易地返回合并后的链表

同时比如当链表只有一个元素的特殊情况（删除啊等操作），也比较好处理，直接返回dummy.next

# 创建一个值为0的节点

dummy = ListNode(0)

dummy.next = head

等于 dummy = ListNode(0, head)

这样的话，返回链表就可以通过pre.next 来完成

19. 删除链表的倒数第 N 个结点

82. 删除排序链表中的重复元素 II

### c. 找中间节点

def middleNode(head):  
 slow = fast = head  
 while fast and fast.next:  
 slow = slow.next  
 fast = fast.next.next  
 return slow

### d. 反转链表

### d. 注意：

* python 中只有None（不用NULL），表示该值是一个空对象，空值是Python里一个特殊的值，用None表示。None不能理解为0，因为0是有意义的，而None是一个特殊的空值。
* 链表遍历结束的条件是while cur 还是 while cur and cur.next

## 3. 字典：

## 4. 动态规划：

解法模板：<https://www.cnblogs.com/L-shuai/p/14730103.html>

动态规划一般分为一维、二维、多维（使用状态压缩），对应形式为 dp(i)、dp(i)(j)、二进制dp(i)(j)。

如果是二维的话，涉及填表，可以从左到右，从上到下，也可以倒着填

经验：dp区域是正方形的话，通常左下角区域无效不需要再填，因为走过的区域不用再走

for j in range(1, size):  
 for i in range(j):

### 动态规划做题步骤

明确 dp(i)应该表示什么（二维情况： dp(i)(j)）；

根据 dp(i)和 dp(i−1) 的关系得出状态转移方程；

确定初始条件，如 dp(0)。

### 自己悟出来的道理：

传统的动态规划问题都是这一次的问题和上一次问题之间的联系，根据这个写出来状态转移方程。一般都是用数组来存储前i-1个状态，找出来初始条件的值，遍历递推，最后返回最后一个状态就是答案。空间优化的话，如果说当前状态之和前一两个状态有关，而不是和更早的状态有关。不需要整个数组来存储状态，那就只需要一两个变量来存储值了。（灵感来源<https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii/solution/mai-mai-gu-piao-de-zui-jia-shi-ji-ii-by-leetcode-s/> ）

比如

122. 买卖股票的最佳时机 II

## 5. 树

## 回溯法

深度优先遍历、递归、栈、回溯的本质都是后进先出

回溯的定义：采用试错的思想，它尝试分步的去解决一个问题。在分步解决问题的过程中，当它通过尝试发现现有的分步答案不能得到有效的正确的解答的时候，它将取消上一步甚至是上几步的计算，再通过其它的可能的分步解答再次尝试寻找问题的答案。回溯法通常用最简单的递归方法来实现，在反复重复上述的步骤后可能出现两种情况：

找到一个可能存在的正确的答案；

在尝试了所有可能的分步方法后宣告该问题没有答案

动态规划只要求评估最优解，回溯则可以搜索到所有的方案，本质是一种遍历方法。

## 最长xxx

注意区分最长公共子序列和最长公共子串

最长公共子串要求在原字符串中是连续的，而子序列只需要保持相对顺序一致，并不要求连续。

两者的基础都是二维动态规划

eg.涉及到两个字符串的相关题目，一般dp[i][j],表示第一个字符串到第二个字符串的操作，关键是找好状态转移关系，然后开始循环，填表。比如：

72. 编辑距离

## 二分法：

一般情况下，看到有序数组，就应该想到二分法，时间复杂度是log2(n)

二分法的关键是，细节，边界范围是开区间还是闭区间

mid = (left+right)//2 mid 作为分隔位置，将数组分为 [left,mid] 和 [mid+1,right]

然后不断改变left或者right，从而改变mid, 不断接近查找的目标

## 小知识点：

### 除法：

a = 5  
print(a/2)  
print(a//2)

一个是除法一个是整除

### 正则匹配：

规则：

<https://www.runoob.com/python/python-reg-expressions.html>

注意：Python正则表达式前的 r 表示原生字符串（rawstring），该字符串声明了引号中的内容表示该内容的原始含义，避免了多次转义造成的反斜杠困扰

eg. <https://leetcode-cn.com/problems/string-to-integer-atoi/>

### 方向问题

遇到二维矩阵的方向问题时，

# 定义上下左右四个行走方向

directs = [(0, 1), (0, -1), (1, 0), (-1, 0)]

#### [79. 单词搜索](https://leetcode-cn.com/problems/word-search/)