## 题目思维或方法类似：

* 自顶向下140 403 638
* 多个状态量交替：188 309
* BFS搜索279 403
* O(n!)->O(n^3)三角形方法312 664
* O(n^2) 629

## 140. Word Break II

这道题目与546题的dp写法差不多，其实一道题目可以有很多种思想和解法，这道题目就是dp，之前TLE了，后来发现是return m[s]=res没有插入unordered\_map中。unordered\_map真是效率高的不行，绝对利器。

## 188. Best Time to Buy and Sell Stock IV

其实做了这么多DP问题，很多题其实都是特别的简单，首先分析这种题目数据特点，增加数据和减少数据会有什么影响和联系。

DP核心思维就在于“自顶向下，自底向上，逆向思维”，

关键词在于“最优子结构，重叠子问题，无后效性，状态及表示，初始值”

## 279. Perfect Squares

//解法四：Breadth-First Search: 80ms

思维值得学习！将广度优先搜索用于求 是否能满足某种条件或最快的方式 是一种全新的思维。时间复杂度为O（n)

另外一个类似的可用广度优先搜索解决问题的题目是403. Frog Jump

这类题目能用BFS的关键点在于：

1.节点是什么？

2.节点间的边是什么？（满足什么关系时可看做相连？）（边一般都不是事先预知的，而是动态呈现出来的）

## 309. Best Time to Buy and Sell Stock with Cooldown

非常好的DP题目，对于DP题目最重要的是由简单的一些例子由简到繁的进行分析，

找准状态量和分析清楚子问题（是否有后效性），对状态量进行从前到后的迭代递推或者从后到前的分析。

无论状态量是什么（一个整型值、结构体、字符串、集合等，或者有多个状态量），一定要分析找准状态量。

## 312. Burst Balloons

此题目应该是一类经典暴力需n!的题目中的典型。

一般暴力穷举结果需要n!的此类题目，一般都可以用DP算法n^3解决，类似题目还有前面的某个题目（题号忘记了）以及CCF的压缩编码，石子游戏等题目类似。

不过这个题目应该说更为一般。其他类似题目的子问题都可以分开考虑，但是这道题目有点不一样，其实从某种角度来看，是一样的。看怎么去理解！

思考方式一般有两种：

1.自底向上（从简单的子问题考虑，递推迭代到最终解）（此题的子问题容易被误解。nums[i]应该和两边1的作用等价）

2.自顶向下（从最后的问题出发，思考如果小规模的相同性质问题解决后，此问题可以如何得到解决。）

一般说来，两种方式都不是独立存在的，应该是相辅相成的，共同思考的。

这类题的写法有三种：

1.从左到右的三角形

2.从下到上的三角形

3.从斜边到直角的三角形

## 363. Max Sum of Rectangle No Larger Than K

总结： 对于求连续区域问题都可以采用DP区域减法，快速求出目标。

一维:后者-前者

二维:区域减法

...对于更多维也一样

## 466. Count The Repetitions

这个题目差一点就做出来了，方向还是算对的，看了一点思路提示后，靠自己完全解决了。

要怎样思考着一类的题目呢？首先暴力是不可能的，数据量太大；所以我们肯定应该去分析此题的特性，就是其字符串的重复性。但是要怎样转换成一个小规模问题，或者数学方法，光看是不能解决问题的，还要用一些实例从小处思考，启发，归纳。

考虑到每找一个s1，其第一个要找的s2中的字符是什么，不会超过100种情况，用鸽巢原理分析出答案。

注意两点：

1.本题不是求公共子序列，而是贪心的求包含个数。

2.当遇到内存越界报错时，首先应该看看是不是变量写错或写反了。

## 546. Remove Boxes

这道题很难吗？不思考为什么永远觉得题目难。我觉得此题的难点有：

1.摸索清楚子问题和状态量。不同于以前大多此类纯暴力需要O(n!)题目，基本套路一个二维数组， 从头到尾，有关位置顺序，二维数组下标可以有效存放相应状态量。此题目这样是行不通的。仅仅依赖二维状态量是肯定不能记录完整状态值。

2.如何有效的用代码实现。以前几乎所有的DP问题都是非递归按照行列更新。但是此题却要用递归写法最佳。能有效将问题规模变小，同时记录相应已经求解过的状态。

## 600. Non-negative Integers without Consecutive Ones

题目大义：求0-n中数个数，其中二进制没有连续1。

很好的DP题目，最开始估计到使用DP来解决此题目，但是用了几个小时都没有做出来。

在做此题目时体会到了一种一些编程比赛中会出现的状态；

感觉有思路，感觉方向可能是对的，感觉马上就要做出来了，但是一直处于一种紧张的状态，没有什么进展，持续了半个小时后，还是没有所获，这时就感觉到了一种失败无助的滋味，会影响到后面的题目正确心态。

纠正方法：一般一个题目不要花费超过20分钟，最多25分钟，如果还是没有想出来，一般要么超出实力，要么头皮发热没有正确认识题目，思路方法都是错的，但是有一种能做出的假象。但是不管怎样，停止，立即停止。先做其他题目，等到后面缓和以后，再重新认识题目，换一种角度和方法解决。

本题的错误：方法DP没有问题，但是对问题本身的认识不够，1-n中连续11，没有认识清楚这些数字的构成分析，一直扎身于错误中，没有反面思考错误，换方向，认识问题。

这道题目算是一种多次分割问题的题目，将原问题分割为多个小问题，每个问题又需其他方法来解决，但是问题类型一致。

## 629. K Inverse Pairs Array

有必要总结一下动规解题规范，明明自己很快知道了思路，但是用了很多时间才ac，究其原因：

1. 动规写法没有固定化，在一些题目汇总边界条件进行扩大，经初始后编程相对简介，但是有时其意义和题目中定义的有一点点偏差，容易出错，此时宜按照意义确定下标较好。

2. 读题马虎，将mod=1e9+7看成了1e9+9

3. j-i这种反应迟钝，太谨慎。

## 638. Shopping Offers

感觉比较有意思的题目，与以前做的一道消方块游戏一样，一开始准备用迭代方法做的，后来用的自顶向下的方法即递归式动规方法。同时利用bit位压缩状态。

## 664. Strange Printer

总的来说，这道题目还是不错，其实掌握方法后还是比较简单的。一看数据量，100， 复杂度一般为O(n^3)，必然是二维数组+之间的遍历。这道题目的大概思路还是知道的，但是最后还是只通过了101/211个case。

动态规划解此类问题的一个关键点在于：

1.求解过程中是否包含了所有的可能性，每一次的求最优值的过程代表着怎样的实际过程。

2.题意规则理解的深刻程度，这点很重要，一般的分析是多读题目，从最简单的一些例子分析入手，向复杂情况推广。

本题思路：

i k j

~ a ~ a ~ a

考虑上述的情况，dp[i][j]表示不考虑外部，仅s[i]-s[j]打印需要的最少步数。

i<=k<j 若s[k]==s[j], 那么dp[i][j]的一个可能值为dp[i][k]+dp[k+1][j-1]（j处的a同k处的a一起画出）