LRU Cache

#容器；#链表；#散列表

要点：

使用链表保存数据（也可使用STL中的list）；

使用散列表保存Key和相应数据块指针的映射（比如使用map<int, list<Node>::iterator>结构）；

每次get时把取得的数据块移到链表头；若Cache空间满时，移除链表头元素

get操作的时间复杂度为O(1)；set操作的时间复杂度也是O(1)

Min Stack

#容器；#栈 #技巧

要点：

使用STL中的stack保存元素（使用vector会导致Memory Limited Exceeded，因为STL中的stack默认使用deque实现），使用另一个stack保存和当前该stack栈顶元素相等或更小的元素，保证getMin操作能以O(1)时间内返回最小元素。

Partition List

#链表

要点：

使用两个指针遍历链表。指针p串联比x小的元素，指针q串联不小于x的元素。遍历结束后先断开两个指针最末尾元素的原后继指针。注意各处指针可能为空的情况。

Remove Duplicates from Sorted List II

#链表

要点：

遍历链表，使用指针q记录当前元素的前驱的前驱，当发现当前元素和前驱元素相同时，q不再变化并记录发现重复元素的记号；当发现当前元素和前驱元素不同且有重复元素记号时，q的后继指向当前遍历指针并消去重复记号。

使用临时表头指针（因为原表头也可能是重复元素）；当遍历结束且有重复记号时，q的后继设为NULL。

Remove Duplicates from Sorted List

#链表

要点：

遍历链表，使用指针q记录当前元素的前驱，当发现当前元素和前驱元素相同时，q不再变化并记录发现重复元素的记号；当发现当前元素和前驱元素不同且有重复元素记号时，q的后继指向当前遍历指针并消去重复记号。

不需要临时表头指针（原表头不可能是被消去的重复元素）；当遍历结束且有重复记号时，q的后继设为NULL。

Search in Rotated Sorted Array

#数组；#递归

要点：

递归二分查找。如果A[start]<A[end]，说明是升序序列；否则是经过移动的序列：如果A[middle]>A[start]，说明中间元素在左侧子序列内，否则中间元素在右侧子序列内。再根据target和A[middle]的大小关系决定递归哪个子序列。

Search in Rotated Sorted Array II

#数组；#递归

要点：

递归二分查找。在上一题基础上，由于只需要多考虑A[start]、A[end]和A[middle]三者比较时相等的情况。其中，当A[start]和A[end]相等时，根本无法确定遍历哪个子序列，因此需要遍历左右子序列，然后取或。

Maximal Rectangle

#动态规划

要点：

从矩阵的左上角开始遍历，以当前元素作为子矩阵起点，dp[i][j]记录以i，j作为子矩阵右下角时子矩阵元素是否全为1。dp[i][j]=dp[i-1][j]&&dp[i][j-1]&&dp[i][j]&&dp[i-1][j-1]，如果dp[i][j]为true则计算面积值并更新max值，否则遍历下一个起点元素。[代码来源](http://blog.csdn.net/lanxu_yy/article/details/17533203)

联想：

解决最大子矩阵问题的两个思路：

1. 计算第i~j行的和（可以通过1~j行减去1~i行来解决），把最大子矩阵问题降为最大连续子串和的问题
2. 借用本题思路，dp[i][j]=dp[i-1][j]+dp[i][j-1]+dp[i][j]-dp[i-1][j-1]

注意：

这道题和Largest Rectangle in Histogram一题还有[联系](http://blog.csdn.net/doc_sgl/article/details/11832965)，其实质就是把二维问题变成一维问题。通过比较上下相邻两行的元素信息，把上下连续的为1的单元格数目记录在最低一行中。这样问题就变成了Largest Rectangle in Histogram中求最大连续矩形面积的问题。

Largest Rectangle in Histogram

#栈 #技巧

要点：

遍历高度数组。使用一个栈来保存最长连续递增子序列。栈内保存的是连续递增子序列的索引。如果当前遍历元素大于栈顶元素，则入栈；否则在弹出所有大于当前遍历元素的栈内元素，对于每个弹出元素，计算以弹出元素为高、弹出元素至当前元素前一个元素的矩形面积（因为栈内保存的是连续递增子序列的，所以保证有矩形），并更新最大矩形面积。[代码来源](http://www.cnblogs.com/avril/archive/2013/08/24/3278873.html)

注意：

需要在高度数组最后插入一个0元素，可以认为是数组最后一个元素的高度为零。

Combinations

#递归；#深搜

要点：

对数组进行深度优先搜索。当结果集合满足个数时返回；否则从当前起点开始，把之后的每个元素放入结果集中，递归调用搜索函数。

注意：

递归返回后，从结果集合中弹出当前放入的元素。

Word Search

#深搜

要点：

以所有和给定单词起始字母相同的字母开始按照单词序列进行深搜，使用一个散列表记录哪些元素已经被访问过，以免重复访问。

Minimum Window Substring

#字符串 #双指针 #技巧

要点：

使用两个指针（下标）指示窗口位置。如果窗口内还没有包含给定字符串中所有字符，则递增后一指针。如果窗口内已经包含所有必要字符，则前移前一指针，如果当前窗口长度最小，则记录最小长度和起始位置（前一指针位置）：如果窗口内没有包含给定字符串中所有字符，则再次前移前一指针；否则继续前移前一指针。

注意：

处理还没有移动前一指针、后一指针已经遍历到字符串末尾的情况。

Sort Colors

#字符串 #双指针 #技巧

要点：

两指针分别代表2放置的位置和1放置的位置，其初始值均指向字符串末尾。从前向后遍历字符串。如果碰到1，则交换指针1和当前值，且指针1向前移动一位；若碰到2，则交换指针2和当前值。直至遍历指针遇到指针1。

注意：

因为指针2是指明下一个2放置的位置，因此不能超过指针1。因此当指针2超过指针1时，指针2设为指针1所在位置。

Search a 2D Matrix

#二分查找

要点：

判断给定数字是否在矩阵某一层中，在的话对该层进行二分查找。

注意：

二分查找的结束条件和索引不要越界及死循环。

Set Matrix Zeroes

#技巧

要点：

第一次遍历：若(I, j)元素为0，则把(I, 0)和(0, j)标记为0；若(I, 0)/ (0, j)本身为0，则标记column/row

只访问第一行和第一列（不要访问(0, 0)），如果(I, 0)/ (0, j)为0，则把该行/该列全置为0

最后，若row被标记，则第一行设为0；若colum被标记，则第一列设为0

注意：

第二阶段，访问第一行和第一列时，不能访问(0, 0)，因为(0, 0)被标记为0可能是第一行有0，也可能第一列有0，不能保证两部分都是0

Edit Distance

#动态规划

要点：

如果word2的第i个字符和word1的第j个字符相等，那么dp[i][j]=dp[i-1][j-1]+1；

如果不相等，那么dp[i][j]=max(dp[i-1][j], dp[i][j-1], dp[i-1][j-1])+1

注意：

思路来源：自己举例：word1=abc；word2=bce；然后填充矩阵（增加word1和word2为空的情况）；然后先考虑两个字符相等情况，确定两字符相等时，和两字符串去掉这两个字符情况相同，因此dp[i][j]=dp[i-1][j-1]+1，然后当字符不同时，就是增加一步操作，而从三个来源（dp[i-1][j], dp[i][j-1], dp[i-1][j-1]）都可以转换到当前状态，因此取三者中最小再加1

简言之，举例（举例举得好很关键）🡪（确定矩阵元素定义）🡪填充矩阵🡪考虑状态转移方程（先考虑特殊情况，再考虑非特殊情况中如何转移）🡪在矩阵中进行验证🡪全部正确后写代码

最好是在问题规模本身的矩阵前加一行加一列，填充该行该列有助于理解状态转移情况。

Climbing Stairs

#动态规划

要点：

Dp[i]=dp[i-1]+dp[i-2]。理由：对于第i格台阶，可以从i-2格走两步上来，也可以从i-1格走一步上来

Sqrt(x)

#二分查找

要点：

以0为下界，x/2为上界进行二分查找。找到的middle最后还要判断，如果middle\*middle大于x，那么返回的是middle-1

注意：

要使用unsigned long long，否则会溢出。

Add Binary

#字符串

要点：

使用一个变量记录进位情况，然后模拟二进制加法。注意最后全部加完之后如果进位变量不为零，则把进位值加到结果的第一位。

Merge Two Sorted Lists

#链表

要点：遍历两个链表。使用特殊头指针方便代码编写。

Mininum Path Sum

#动态规划

要点：dp[i][j]=min(dp[i][j-1], dp[i-1][j])+matrix[i][j]

Plus One

#数学 #字符串

要点：使用一个变量记录进位情况，然后模拟二进制加法。一开始即可把进位量置为1，简化代码编写。进位值=（某位上的数字+进位值）/10；结果上的某位数字=（某位上的数字+进位值）%10；注意最后如果进位值不为零则把进位值加到结果最高位。

Unique Path

#动态规划

要点：

dp[i]=dp[i]+dp[i-1]

Unique Path II

#动态规划

要点：

如果矩阵中某个值为1，则动规矩阵中该处值为0，其余的和Unique Path中一样。

Insert Interval

#遍历

要点：

遍历。如果插入区间和当前区间有交集，则合并两个区间。然后继续遍历，此时如果当前区间和后一区间有交集，则合并两区间。

Merge Interval

#遍历

要点：

遍历。和Insert Interval后半部分相同。

Jump Game

#遍历

要点：

题意是每次“至多”走元素个数步。因此遍历数组，记录当前能走到的最远位置，如果最远位置超过数组末尾，则为true；如果到达最远位置时仍未达到数组末尾，则返回false。

Length of Last Word

#字符串

要点：

题意是如果最后一个为空格，则返回空格前一个单词。因此遍历字符串，记录当前最后一个单词。

Maximum Subarray

#动态规划

要点：

一维动态规划。不需要0状态。动规状态表示以当前元素为末尾的和最大的子串。同时使用一个变量记录全局最大的子串和。dp[0]=A[0],dp[i]=max(dp[i-1]+A[i],A[i])。最后返回全剧最大子串和。

Permutation Sequence

#数学 #技巧 #排列

要点：

题意是返回第k个全排列（第k个是指所有全排列可能中第k小个）。把全排列全输出会超时。思路是设第k个全排列的第一位数字为x，那么x\*（n-1）！=k，x=k/（n-1）！（原因：后n-1位全排列是（n-1）！，因为第一位是从1开始的，所以x也表示从1开始到x）。然后k=k- x\*（n-1）！，迭代。

注意：

要实现上面的思路，关键是x\*（n-1）！=k式子的解x是取上整。有非常蛋疼的+1和-1的细节。

Rotate List

#链表

要点：

先遍历一遍，知道链表长度，然后找到分割点。

Anagrams

#字符串

要点：

对于每个单词排序，按排序后的字符串投入散列表，散列表的值记这样的词在原序列中第一次出现的位置。

Pow(x, n)

#数学

要点：

二分，递归求解n/2规模的解。主要特判n为负和n为0的情况。

Rotate Image

#矩阵

要点：

顺时针转换：(i, j)🡪(m-j, i)，注意代码实现时候是反向替换。还有如果矩阵行列是奇数个的话，不是完整的四分之一进行旋转，而是四分之一缺右边一列或下面一行，否则这两部分会别替代两次。

Permutations

#深搜 #排列

要点：

遍历。交换start和i，深搜，交换回来。

Permutations II

#深搜 #排列

要点：

首先排序。然后和Permutations类似，遍历。交换start和i。关键是当i>start并且num[i]==num[start]时，不要交换start和i。

Jump Game ||

#技巧

要点：

本题需要除了需要一个值需要记录当前能到达的最远位置外，还要一个值记录当前所查看的元素所对应的最远位置。因为如果当前查看的元素超过该值时，说明必须要多往前走一步，此时最小步数增加，并且更新该值为当前能到达的最远距离。而当当前能到达的最远位置抵达最后一个元素时，即返回最小步数加1。

Two Sum

#散列表

要点：

先把所有元素存入散列表，有多个元素相同的则记录该元素个数。然后遍历所有数字，如果发现目标和当前遍历元素相减后的元素也在散列表中时，记录一个索引。当再次碰到时记录第二个索引。注意：存在当前遍历元素和相减后元素相同的情况，这时需要检查散列表中该元素在原集合中有几个。

Add Two Numbers

#数学

要点：

遍历，模拟加法。注意进位。

Longest Substring Without Repeating Characters

#散列表

遍历，使用头尾指针记录子串头尾。子串中的元素放入散列表，并记录元素对应的索引。如果当前元素在散列表中出现，那么从子串头到散列表中该元素所对应的索引之间所有元素舍弃，并在散列表中删除。头指针指向重复元素的索引的下一个位置。

Scramble String

#深搜 #技巧

要点：

对于两个字符串，使用散列表比较其所含字符是否一致，不一致返回false，否则返回true。然后从第一个元素开始遍历到最后一个元素，每次以当前遍历位置为中心对两个字符串进行分割，递归本函数。

String to Integer (atoi)

#字符串 #数学

要点：

考虑几种边界情况：头部存在空字符（跳过）；数字前存在多个正负号（返回0）；数字间存在无效字符（中断，只考虑无效字符前的数字）；数字太大或太小（返回INT\_MAX或INT\_MIN）。注意对于数字越界，判断的方法是result>INT\_MAX/10|| (result==INT\_MAX/10)&&num>INT\_MAX%10。尤其是后半部分的条件。原因是如果满足result==INT\_MAX/10，且数字为负，那么说明num>7，那么返回INT\_MIN也没错；如果数字为正数，那么num>7，那么当然返回INT\_MAX。

Reverse Integer

#数学

要点：

类似atoi，要考虑整数最大值和最小值时的边界情况。判断条件也是result>INT\_MAX/10|| (result==INT\_MAX/10)&&num>INT\_MAX%10。

Palindrome Number

#数学 #回文

要点：

对原数迭代的取余和求最大值，然后比较。要一开始对原数就求出其最大位数是10的几次方，使用一个值记录，然后每次迭代对该值除以100（因为要取头和尾）。

注意：

如果每次在迭代中计算除以几个10的次方，会在类似100021的数字中出错。因为头尾1去掉之后，数字一下子变2了，和121无法分辨。

Median of Two Sorted Arrays

#技巧 #二分查找

要点：

这个问题可以用跟广谱的在有序数列中求第k个数解决。比较A[k/2-1]和B[k/2-1]，如果A[k/2-1]>B[k/2-1]，那么可以安全地丢掉B[0]~B[k/2-1]。同时k变成k/2。

注意：

在实际编程中，可以保持第一个参数的数组永远比第二个参数的数组短，那么当第一个参数的数组长度为空时，直接返回第二个参数数组中第k个就行。

另外，要保证k/2比m大，可以使用min(k/2, 数组A长度)来保证，这样的话数组B的截断索引就变为k- min(k/2, 数组A长度)，总之要保证两个数组的子序列元素个数合起来要等于k。

Longest Palindromic Substring

#技巧 #动态规划 #回文

要点：

动态规划方法：dp[i][j]表示以i为开头、j为结尾的子串是否为回文串。矩阵初始化为初始dp[i][i]和dp[i][i+1]，分别代表一个字母自己是回文串和两个相邻字母是否是回文串。然后dp[i][j]=dp[i+1][j-1]&&s[i]==s[j]，表示以i+1为开头、j-1为结尾的子串是回文串，并且字符s[i]和字符s[j]相同，即以i为开头、j为结尾的子串也是回文串。同时设立变量记录当前最长回文串的位置即可。

Mancher算法：是O(n)级别求回文子串算法。其算法大概思想为：假设当前考虑s[j]。已知s[j]前面以s[k]为中心的回文字符串最长，向后延伸至s[j+3]。因此，s[j]和s[2k-j]（即以k为中心的j的镜像）必然是一样的（因为以k为中心是回文串），而且至少到j+3为止，以j为中心的回文串长度和以2k-j为中心的回文串长度是一样的（同理，因为以k为中心是回文串）。而因为以2k-j为中心的回文串长度是已知的（因为2k-j在j前面），所以对于j来说，只要考虑j+3以后的元素和j-3以前的元素是否相同就行了。如果相同，更新p[j]（p记录每个字符为中心的最长回文串长度）。

注意：

Mancher算法难度较大，网上有很多介绍资料。结合上文具体理解。在实现时，在原字符串每个字符中间加入“#”，可以避免分别考虑回文串是单数还是双数的情况，简化代码。

Container With Most Water

#技巧 #双指针

要点：

两个指针代表当前考虑的左右边界，初始时分别指向序列的头和尾，并分别向中间移动。记录当前边界的盛水量。然后移动两个指针中指向的数字较小的那个指针。如果某个指针当前移动后的值比开始移动之前的值小，那就继续移动，直至新的值比开始移动前的值大。回到上述循环。

注意：

最大盛水结果的两个边界必须满足如下条件：左侧边界的左侧都比左边界小；右侧边界的右侧都比右边界小。

Longest Common Prefix

#字符串 #技巧

要点：

公共前缀不可能比最短的字符串长。取第一个字符串开始比较。当出现不一致时则截断后面部分，并取下一个字符串。

3Sum

#技巧 #双指针

要点：

先排序。然后遍历取每个元素作为第一个元素。然后在当前所取元素之后的子串设置两个指针，分别指向头和尾。如果两个指针所指之和大于target减去第一个数，那么后指针前移，否则前指针后移。找到一组答案则加入结果集。期间还可用一些剪枝。

注意：

还要考虑相同解的情况。在选取第一个数字和选取后面两个数字时都要注意排除相同解。直接使用set会超时。

3Sum Closest

#技巧 #双指针

要点：

和3Sum类似。

4Sum

#技巧 #双指针

要点；

和3Sum类似，只是前两个元素都是遍历获得而已。、

Next Permutation

#技巧 #排列

要点：

从后往前找第一个升序元素（即该元素比后一元素要小）s[a]，同时记录该元素到原序列末尾中比s[a]大的最小元素的位置b。交换这两个元素。然后对a+1到序列末尾的子序列进行重新排序。

Remove Nth Node From End of List

#链表

要点：

使用一个数组记录第i个元素的前驱。

注意：

使用临时头指针。

Valid Parentheses

#栈

要点：

压栈，当发现栈顶元素与期望元素不符时返回false。

Generate Parentheses

#技巧 #深搜

要点：

递归函数参数为左括号剩余数量和右括号剩余数量。当两者剩余数量均为零时，得到的序列作为一个结果加入到结果集中。否则，首先加上一个左括号继续递归。然后当左括号剩余数量小于右括号剩余数量时，加上一个右括号再次递归。

Merge k Sorted Lists

#堆

要点：

每个链表取第一个元素组成最小堆。每次取堆顶元素加入结果链表，并取堆顶元素的后继元素加入最小堆。

Swap Nodes in Pairs

#链表

要点：

使用Reverse Nodes in k-Group的方法解决。

Reverse Nodes in k-Group

#链表

要点：

遍历链表，同时计数。当计数等于k时，对parent到当前链表之间的子链表进行翻转。翻转方法是把最前面一个元素插到当前元素之后。

Remove Duplicates from Sorted Array

#数组 #双指针

要点：

遍历数组，同时使用一个索引记录下一个不同元素的放置位置。如果当前遍历元素和前一遍历元素不同，那么该元素复制到放置位置，放置位置后移一格。

Remove Element

#数组 #双指针

要点：

遍历数组，同时使用一个索引指向放置位置，该索引初始值指向数组尾部。如当前遍历元素为待删除元素，则在放置索引找一个非删除元素与之交换。

Implement strStr()

#字符串

要点：

KMP算法。

Divide Two Integers

#数学 #技巧

要点：

用右移代替除。因为不能用除法，只能用右移代替，因此只能除以2或2的倍数。首先看被除数能够包含几个除数，此处的“几个”只能是1+2+4+……这样的数。然后被除数减去包含除数的部分，剩下部分再次循环，直至被除数比除数小为止。还要考虑复杂的被除数和除数为INT\_MAX或INT\_MIN的情况。

Substring with Concatenation of All words

#双指针

要点：

因为模式串每个单词的长度是一致的。首先把模式串的每个单词放入散列表中，然后遍历匹配串中长度为模式串总长的子串，并记录子串中每个模式串的出现情况。

Longest Valid Parentheses

#动态规划 #技巧 #栈

要点：

使用数组记录每个元素是否属于括号匹配，初始均标记为false。栈中压入的是括号所在下标。当一对括号匹配时，两个括号对应的数组元素标记为true。最后遍历数组，找出最长的连续true长度即可。

还有一种使用动态规划的解决方法。dp[length-1]=0。dp[i]表示以i开头的元素到序列末尾的符合括号匹配的子序列长度。所有’)’对应的dp值都是0。对于对应‘(’的dp元素，如果s[dp[i+1]+i+1]为‘)’，即说明是s[i]和s[dp[i+1]+i+1]括号匹配，因此dp[i]= dp[i+1]+2。同时另外，如果是s[dp[i+1]+i+2]是‘(’的话，dp[i]还要加上dp[dp[i+1]+i+2]，因为这两个部分都是括号匹配的，长度要相加。这种动态规划实在复杂。

Search Insert Position

#数组

要点：

遍历。遇到比target大的元素即终止。

Search for a Range

#数组

要点：

遍历。遇到和target一样的记录start，之后遇到和target不同的元素即终止。

Maximum Gap

#技巧 #数组

要点：

找到最大值最小值，把所有数分在(n-1)个桶里（n为序列长度）。因为一共至多有n-2个数（去除最大最小值），因此必然有一个桶为空。所以只要考虑每个桶里的最大最小值和相邻桶即可。

该题在Leetcode中直接排序然后遍历也能通过。但不是线性时间复杂度。

Find Peak Element

#技巧 #二分查找

要点：

取中点，如果相邻元素均比中点元素小，则返回。否则若左侧元素大于中点元素，往左半段搜索；右侧元素大于中点元素，往右半段搜索。时间复杂度为O(log(n))

Intersection of Two Linked Lists

#链表

要点：

首先得到两个链表的长度。较长的链表先移动指针，使得剩余长度和短链表一致。然后两指针同时遍历，当指针相同时即为汇合点。

Multiply Strings

#数学

要点：

手动模拟乘法。使用两个数组，一个记录乘数某一位和另一个乘数的计算结果，另一个记录乘法当前结果。当某一位和另一个乘数相乘结束后，再和第二个数组进行模拟加法运算，最后清空第一个数组，考虑下一位。

注意：

要考虑结果只有一个0，但模拟时得到了多个0的情况。

Trapping Rain Water

#技巧

要点：

对序列遍历两遍。第一遍从左往右。标记当前最大的值。如果遇到一个值大于标记位置的值，那么计算两者之间的容水量，并把当前位置设为标记位置。第二遍从右往左类似地遍历。

注意：

题目中每个值代表的隔离物也是算空间的，因此在遍历时要记录标记位置和当前位置之间的隔离物体积，在计算容水量时减去隔离物体积。

First Missing Positive

#散列表

要点：

对于序列中的每个正数投入散列表中。然后从1开始对散列表进行查询，返回第一个找不到的值。

另一种方法是利用原序列进行散列。因为是找最小的未出现的正数，所以序列中出现的正数不可能超过序列长度n，因此如果遇到A[i]在1到n范围内，就把它放到序列中第i个位置，即与A[A[i]-1]元素交换。然后遍历原序列，找出A[i]!=i+1时返回i+1即可。

注意：

使用第二种方法时，需要判断A[i]==A[A[i]-1]以防止死循环交换。

Combination Sum

#组合

要点：

对原序列按降序排序，然后深搜。

注意：

深搜有两种方法。一种是取一个元素放入当前结果，然后递归，在当前结果中取出该元素，然后不取该元素，递归。另一种是遍历当前元素至序列末尾元素，取遍历元素放入当前结果，递归，取出当前元素，遍历下一个元素。两种递归方法均可，但是不能混淆！

Combination Sum II

#组合

要点：

为了防止重复结果，即某一条结果中第i个位置只出现1个值为j的元素（不可放多次）。为了达到这个效果，修改上述Combination Sum的解法。当某个元素不取时，为了防止之后相同元素仍出现在同一个位置（该元素在结果中这个位置的情况已在之前递归过），因此跳过所有所有与当前值相同的元素，直接从下一个值不同的元素继续递归。

Count and Say

#技巧

要点：

从2开始模拟。求规模为k的问题解时遍历k-1情况的结果。使用count和last记录同一个数字连续出现的次数（即表示数last出现了count），当出现不同数字时，新的结果序列加入count和last，并更新last，重置count为1。

Repeated DNA Sequences

#哈希

要点：

遍历数组，对于每个子串使用char值对应的int数乘以位数个10（如Hash(AGT)=A\*100+G\*10+T）进行哈希，放入一个set中。如果该哈希值已存在则存入结果set中。

Largest Number

#排序

要点：

其实就是设计一种特定比较函数的快排。比较函数设置为：如果s1+s2（s1和s2是两个int数的string表示）> s2+s1，那么就认为s1>s2。用这个比较函数对数字序列排序，拼接排序后的序列并输出即可。边界情况是数字都是0，只要输出一个0就可以。

Dungeon Game

#动态规划

要点：

本题是要求从左上角到右下角所需要的最小初始血量。和一般动态规划题不同，因为初始状态未知，所以是从右下角往左上角推。对于一个格子的初始血量，肯定是走到其右边或者下边两格中花费血量较少者，即min(dungeon[i+1][j], dungeon[i][j+1])，还要扣除该格子本身的血量，即min(dungeon[i+1][j], dungeon[i][j+1])-dungeon[i][j]。因为初始血量不可能负的，如果当前格子可以补充很多血量的话，初始血量最少为零，即max(min(dungeon[i+1][j], dungeon[i][j+1])-dungeon[i][j], 0)。

Binary Search Tree Iterator

#二叉树

要点：

使用一个栈保存从根结点开始的所有左子树结点。调用next()函数时候弹出当前栈顶元素并压入栈顶元素的右子树节点及之后的所有左子树节点。

Factorial Trailing Zeroes

#数学

要点：

该题本质上是考察给定数有几个5（因为所有0都是由5和2相乘得到的；10也可以理解为5\*2，而有5必有2），即对给定数做因数分解时，有几个5。因此使用辗转相除法，记录5的个数即可。

Majority Element

#数组

要点：

因为存在一个数超过n/2，所以可以采取两两删除的方法，剩下的必然是要求的数。选择第一个数为候选者，候选计数设为1，然后从第二个数开始遍历数列。如果当前遍历的数和候选数不一致，则候选计数减1。当候选计数为0时，选择当前遍历数为新的候选者。

注意：

本题可以扩展为求数列中至少存在超过n/k个的数，本题是k=2的特例。解决n/k问题时，[有一个时间复杂度为O(nk)的方法](http://www.geeksforgeeks.org/given-an-array-of-of-size-n-finds-all-the-elements-that-appear-more-than-nk-times/)。这个方法和k=2的方法本质是类似的，即首先选择数列中前k个不一样的数作为候选解，然后遍历之后的数。对于当前遍历的数，如果候选解中有和它一样的数，就对该数的候选计数加1；若没有一个候选解和该数一样，就对每个候选解的候选计数减1。最终剩下的候选解都是答案。

Fraction to Recurring Decimal

#数学

要点：

首先得到整数部分，然后处理小数部分。小数部分每位的得到方法和竖式类似，就是余数乘10再除以除数。使用一个map保存每次的被除数和所得到的小数在结果字符串的位置。当某个被除数在map中出现过时，说明小数部分出现循环。此时从map中取出该被除数对应的结果字符串位置，在该位置处插入左括号。

注意：

首先是int转long long，避免溢出问题（long和int是一样的，所以不行）。还有一个就是当整数部分为0时，负号会因为整数相除而遗漏（如-1/2=0，其实应该是-0）。因此需要先判断除数和被除数的正负，决定答案字符串中是否要“-”，然后对除数和被除数取绝对值。

Find Minimum in Rotated Sorted Array

#数组 #二分

要点：

典型的二分。如果中点元素大于左边，则遍历左侧；否则遍历右侧；当只有两个元素时判断大小并返回。

Find Minimum in Rotated Sorted Array II

#数组 #二分

要点：

比较恶心。

递归方法：

只有两个元素时，返回较小者。

当中点元素大于左边或中点元素等于左边且左边大于右边时，遍历左侧；

当中点元素小于右边或中点元素等于右边且左边大于右边时，遍历右侧；

当中点元素、左边和右边元素均相等时，需要分别遍历左右侧，然后返回两边返回值的较小者。

非递归方法：

本质和递归方法类似。见代码。

Maximum Product Subarray

#数组 #技巧

要点：

本题的关键在于负数的处理。当出现奇数个负数时，乘积会变成负数，而当出现偶数个负数时，乘积又会变成很大的正数。当然还要考虑0对乘积的影响。

因此需要额外保存一个乘到当前位置时的最小值，如果之前出现奇数个负数，而之后出现第偶数个负数，那么该值就会变成乘积最大值。因此遍历过程中的核心式子为maxs[i]=max(maxs[i-1]\*A[i],mins[i-1]\*A[i],A[i])和mins[i]=min(maxs[i-1]\*A[i],mins[i-1]\*A[i],A[i])，还要考虑A[i]就是防止之前的乘积为0。最后遍历maxs数组即可。

Reverse Words in a String

#字符串

要点：

主要考虑空格带来的边界情况。

Evaluate Reverse Polish Notation

#数学 #堆栈

要点：

使用栈保存输入的数字，遇到运算符就出栈运算，把结果压入栈。

Max Points on a Line

#技巧 #哈希表

要点：

该题要考虑在同一斜线上的情况，因此不能用对X轴和Y轴分别排序的方法做。

比较两两点之间的斜率是否存在，相同的斜率作为键保存。其中要考虑两点重叠的情况；还要考虑没有斜率的情况（即x不同y相同）。

Sort List

#链表 #归并排序

要点：

给出的序列是链表形式的，只能使用归并排序，否则会超时。

Insertion Sort List

#链表 #插入排序

要点：

对链表实现插入排序，在头指针前插一个临时头指针方便编写代码。

Binary Tree Postorder Traversal

#二叉树 #栈

要点：

后根遍历。需要一个栈保存根结点，而且需要一个计数。当第一次出栈时，如果有右孩子，需要再次入栈并且将计数加一。以右孩子为根的左孩子再入栈。如果没有右孩子或已是第二次出栈，则访问该节点元素即可。

Binary Tree Preorder Traversal

#二叉树 #栈

要点：

先根遍历。需要一个栈保存根结点。访问栈顶元素时先把右孩子入栈，再把左孩子入栈。

Reorder List

#链表 #栈

要点：

题目要求后半段元素交叉，因此用一个栈保存前半段元素，在遍历后半段元素时，从栈中弹出元素，顺次插入即可。临时头指针便利代码编写。

Linked List Cycle II

#链表

要点：

使用快慢指针判环。如果出现环，快指针指向头指针，然后两个指针每次跳一格继续运行，俩指针相遇之处即为环开始。

注意：

[寻找环入口的证明](http://blog.csdn.net/wencheng2998/article/details/5935771)：

在快指针和慢指针相遇的时候，慢指针走的步数为n步，n=p+c，c是环入口到相遇点的距离；快指针走的步数为2n，2n=p+c+k\*L，L是环的长度。那么也就是说，p+c=n=k\*L。

只要保证从相遇点开始，再走n步的话，仍然可以回到c点；

从头指针开始走n步，也可以走到c点。

在上述两种情况中，只有前p步不同，而后c步是一样的，所以从相遇点和从头开始一起走，新的相遇点就是环的入口。

Word Break

#字符串 #动态规划

要点：

因为要考虑所有切词组合，并且前面的切词对后面有影响（决定后面能不能组成一个或几个词），因此想到动态规划。

动态规划方法：问题规模从小到大，因此字符串长度从1到s.size()。

规模i的字符串能不能被分隔成几个单词的组合用valids[i]表示，只要其中有一个分割可行，valids[i]就为true。因此，对于规模i的字符串遍历每个分割位置，从0到i。

对于任一个分隔位置j，是否是一个有效的分隔位置由两方面决定：

1. 该位置以前的子字符串是否为一个由单词组成的字符串，即valids[j]是否为true；

2. 该位置以后的字符串是否是一个完整单词，即s.substr(j,i-j)是否是一个单词。

由上所述，动态规划的核心语句为valids[i]=valids[i]||(valids[j]&& s.substr(j,i-j) is a word)

Word Break II

#字符串 #回溯

要点：

枚举回溯，无论是从前往后枚举和从后往前枚举总有一个会超时。测试集去除了会导致从后往前枚举超时的变态例子，所以网上的解法在遇到代码中的测试样例2时仍然会超时。

Copy List with Random Pointer

#链表 #技巧

要点：

第一次遍历链表，为每个表元后面插一个该表元的替身表元，插在原表元后后续表元之间；

第二次遍历链表，让每个表元的替身表元的random指针指向原来random指针指向的表元的替身表元。

第三次遍历链表，把替身表元从原链表中抽出来。

Single Number

#技巧 #位运算

要点：

遍历数组。两两求异或。因为异或运算：两个元素相同返回0，不同返回1。因此所有正好出现两次的数字所得到的异或结果均为0，而0和唯一一个出现一次的数进行异或得到的就是原数，即为答案。

Single Number II

#技巧 #位运算

要点：

Candy

#数组 #技巧

要点：

本题的关键就在于当遇到一个下降序列时候，如何保证分给最低一个rating的糖果数为1。因为最后只要所需糖果的总数，所以关键就是给那些最低rating之前的连续下降序列补足足够的糖果数即可（因为再之前是上升序列，所以不用管）。因此当序列上升时，用最新的糖果数更新preMax变量；当遇到下降序列时候，用descendingLength记录连续下降数，当descendingLength>=preMax时，说明当前rating的糖果数到0了，那么之前preMax个（因为下降序列的最高点是preMax，现在降到0，那么说明序列长度正好是preMax！）糖果数每个加1，即总数加preMax即可。

注意：

本题的另外一个trick就是当相邻两个rating一样时，第二个糖果数为1（因为题目里是“rating比邻居高的糖果多”）

Gas Station

#技巧 #数组

要点：

从第一个站开始走。如果能走的话，就正常走；如果回到起点，就返回起点位置；如果不能走（油量不足以抵消耗油），就把起点设为当前起点后一个点，并且补偿转起点变化带来的状态（退掉起点的油量，补回起点到下个点的耗油）。注意如果当前点已经在原起点后面的话，需要直接设当前点为原起点（因为这种情况下，起点—>当前点的路段是没有遍历过的）。

注意：

1. 本题的关键在于：如果第i个站到第j个站是可以走通的，那么从第i+1个站、i+2个站……到第j个站必然都是可以走通的。
2. 网上有一些更简洁的解法，本质上是考虑了gas[i]-cost[i]这样一个每个加油站的耗油剩余的情况。因为如果存在一种能够走完全程的解法，那么所有加油站的耗油剩余和必然要是非负；另外，如果存在走完全程的走法，那么必然存在一个起点，是的耗油剩余相加过程中每一个中间结果都为非负。那么就可以使用sum和total，total记录从0到n的和，sum也是记录和，但是当sum<0时，就设置startIndex为当前遍历的位置且sum清零，如果最终total为非负，那么startIndexbiao+1就是最大子序列和的开始（参考注意点1，startIndex+1的耗油剩余必然是正的，而startIndex的耗油剩余必然是负的（除非startIndex为-1））

Clone Graph

#广搜 #哈希表

要点：

使用一个map记录原图结点和新图结点的映射关系，原图结点作为key，新图结点作为value。随后对map进行遍历，根据原图中每个结点的邻接关系，给对应的新图结点的邻接表添上这些关系。

Palindrome Partitioning

#回溯 #回文

要点：

首先使用一个二维数组dp，dp[i][j]表示字符串s.substr(i, j-i+1)是不是回文字符串。然后根据这个二维数组从字符串头开始深搜，遇到dp[start][i]为1的时候就压栈迭代。

注意：

dp数组的填充本身就是一个动态规划问题。dp[i][j]=(s[i]==s[j])&&(j-i>2||dp[i+1][j-1])

Palindrome Partitioning II

#回文 #动态规划

要点：

这道题很难。解法参考：。求最优解，考虑动态规划。使用D[i, j]表示区间[I, j]之间的最小cut数，那么D[i, n]=min{D[i, j]+D[j+1, n]+1}。因为对于每一个i，该式都是独立的，因此可以把二维变为一维。即对于每一个i（从第i个字符开始到末尾的子字符串），D[i]表示该子字符串的最小cut数。

D[i]最大值是子字符串元素个数减一（相当于没有任何回文子串），若第i个元素到第j个元素之间是回文子串，那么一种方案就是在第j个元素和第j+1个元素之间切一刀，那么整个子串的最小cut数就是D[j+1]（第j+1个字符到末尾的子串的最小cut数）+1（第j个元素和第j+1个元素中划一刀），对于j>=i都进行一次遍历，因此是D[i]=min(D[i], D[j+1]+1)。当i遍历到0时，就是整个字符串的最小cut数。这就是最关键的D[i]迭代代码的意义。

二维数组P，即为记录[I, j]是否为回文串的记忆化数组。

注意：

有一种只需要O(n)空间的[解法](https://oj.leetcode.com/discuss/9476/solution-does-not-need-table-palindrome-right-uses-only-space)，是一种分治法的思想，解法比较玄妙。

Surrounded Regions

#广搜

要点：

先从边界开始找O，找到一个后以此为入口进行广搜，所有找到的O标记为E。对四个边界都遍历过之后遍历全图，剩下的O变为X，再把E变回O即可。

Sum Root to Leaf Numbers

#深搜

要点：

对树进行先根序的深搜，遇到一个树节点就对当前数字乘10并加树节点代表的数字。如果是叶子结点就把当前数字加上结果。

Longest Consecutive Sequence

#哈希

要点：

把数组哈希进一个set中，然后枚举set中的一个元素，如果该元素的邻近左右元素存在，则迭代，结果长度增加；若不存在，则枚举另一个元素，当哈希表元素数小于等于当前最长长度时终止。

Word Ladder

#广搜

要点：

对于每一个从当前单词变换一个词所得到的、且在字典中的新词，把它加入到队列中去，并且在字典中删去这个词（因为广搜得到的总是到该词的最短路径，不可能存在从第一个词到该词比当前跳数更短的）。

注意：

“#“是用来标记layer层数的。遇到一个#、且队列还不为空，则layer加一。

Word Ladder II

#广搜

要点：

使用一个map结构pres保存能够到达每一个单词的单词列表。在和上题类似的迭代过程中，填充pres。当找到目标单词后，对pres从目标单词开始，进行深度优先搜索。当找到一个单词对应的列表为空时，说明该单词即为起点单词，输出深度搜索序列即为一条答案。

Valid Palindrome

#栈

要点：

把字符串前半段加入栈，后半段与栈顶元素比较即可。注意空格、无意义字符和大小写。

Binary Tree Maximum Path Sum

#深搜

要点：

对于每个非叶子节点，首先计算左右子树的和，然后比较左右子树和+本结点、左子树和+本结点、右子树和+本结点，以及本结点的大小，取四者中较大者更新结果，然后取后三者中大者作为返回值。

Best Time to Buy and Sell Stock

#数组 #动态规划

要点：

本题和求“最长子串和”一体本质上是一样的。因此，使用dp数组记录到当前元素为止的最大获利。prices[i]-prices[i-1]表示昨日到今天的获利；如果总获利为负，则选择不买入（即当前获利为0）。迭代式：dp[i]=max(0,dp[i-1]+prices[i]-prices[i-1])。另外使用一个变量记录curMax总的最大获利。动态规划过程结束后返回curMax即可。

Best Time to Buy and Sell Stock II

#数组 #动态规划

要点：

和上题类似。只是因为可以多次买卖，因此是求所有上升子串的和。因此迭代式变为dp[i]=max(dp[i-1],dp[i-1]+prices[i]-prices[i-1])，即只有当天卖出获利更多（即为上升子串）时才更新获利值。

Best Time to Buy and Sell Stock III

#数组 #动态规划

要点：

因为是最多可以买卖两次，因此暴力解法是枚举每个元素作为买卖分割点，左边获利加上右边获利，和最大。但这样时间复杂度是O(n^2)。顺着这个思路可以想到，前一题中从左往右找，找的是卖出获利最多的日子，那么从右往左找，找的就是买入日获利最多的日子，正好符合枚举法中右边的需求！因此使用两个dp数组，分别记为dp1和dp2。dp1表示一次交易到当日（或之前）卖出获利最大值；dp2表示当日（或之后）买入获利最大值。最后对两个数组的对应元素求和，返回和的最大值。

Triangle

#动态规划

要点：

一道典型的动态规划题。使用一个一维数组记录从上一层转移下来的数值，然后遍历该数组寻找最小值即可。

Pascal's Triangle

#数组 #技巧

要点：

按杨辉三角形定义逐层输出即可。

Pascal's Triangle II

#数组 #技巧

要点：

题目其实只要求O(n)的空间复杂度（而不是时间复杂度），因此还是从第一行开始迭代，但只是从后往前更新数组即可；答案中的数学解法看不懂……

Populating Next Right Pointers in Each Node II

#广搜

要点：

其实就是广搜，因为只能用固定大小的空间，因此不能用队列。对于当前遍历层的上一层而言，之前已经用next指针串起来了，因此只需要使用next指针，把上一层的左右孩子依次串起来，然后该层最左边一个结点作为递归参数传入下一个递归即可。

Distinct Subsequences

#动态规划

要点：

又一道很典型的动态规划。根据字符串长度设置二维dp数组：t\*s。dp数组中每个元素代表字符串s[j]中有几次目标串t[i]出现。首先考虑第一行，相当于目标串为t[0]。那么如果s[j]==t[0], dp[0][j]=dp[0][j-1]+1，否则dp[0][j]=dp[0][j-1]；然后考虑下面几行，发现二维数组是个上半阵，即只有当j>=i时元素才存在。那么对于s[i]==t[i]来说，dp[i][i]=dp[i-1][i-1]；对于一般情况s[j]==t[i]来说，首先其基础肯定是dp[i-1][j-1]（即小规模的字符串和小规模的目标串），增量部分应该是同等规模的目标串和小规模的字符串，因为这个才是当前问题的小规模问题（切记：当前问题是字符串s[j]中有几个目标串t[i]，t[i]是不动条件，小规模问题是s[j-1]）。因此，若s[j]==t[i]，dp[i][j]=dp[i-1][j-1]+dp[i][j-1]，否则dp[i][j]=dp[i][j-1]。

注意：

本题是很典型的二维动态规划问题，值得反复揣摩理解！

Flatten Binary Tree to Linked List

#深搜

要点：

从树根节点开始深搜。如果左子树存在，则递归左子树，然后把右子树接到递归左子树返回的结点右孩子指针上，并把左子树转为右子树，然后递归原右子树。

Path Sum II

#深搜

要点：

从根节点开始深搜，使用一个数组记录从根节点到当前结点的各个数字，如果遇到叶子节点且和等于给定值，则把数组中的值放入答案集中。递归返回时把当前结点从数组中弹出。

Minimum Depth of Binary Tree

#深搜

要点：

Balanced Binary Tree

#深搜

要点：

从根结点开始深搜，比较左右子树返回的最深深度差，如果深度差大于1或小于-1则不是平衡树。

Convert Sorted List to Binary Search Tree

#中序遍历

要点：

基本是利用中序遍历的思想。维护遍历start和end，当start>end时返回NULL，否则先构造左子树，然后构造根结点本身，再构造右子树。

注意：

函数中start、end和middle更多地是为了表示当前元素是否应为叶子结点，对于迭代过程没有太多影响。值得注意的是遍历链表的指针curNode是一个引用！这意味着curNode会随着递归过程自动向前移动，如：当左子树构造完成后，curNode指向的元素即为当前根结点，而后当以当前节点为根的树返回是，curNode指向的已是本棵树的上层根结点。这是本代码的一个巧妙之处，忽视这个就无法都懂代码。

Convert Sorted Array to Binary Search Tree

#中序遍历

要点：

也是利用中序遍历思想。先找出中点，然后先构造左子树，再构造右子树。

注意：

可以比较本题和上题，发现两者的思路和代码结构都是高度相似的。虽然数组可以O(1)时间取得任意位置的值，而链表不能随机读取，但是通过使用指针的引用，巧妙地保证了构造每个结点时，函数中的链表指针总是指向需要的元素处。

Binary Tree Level Order Traversal II

#层次遍历

要点：

使用队列对树进行层次遍历。在每一层结束后插入一个特定标记，遇到该标记就在结果集中插入新的数组。最后把结果数组倒置一下即可。

Construct Binary Tree from Inorder and Postorder Traversal

#递归

要点：

本题的难点是如何把已知中根和后根次序序列求树的解法用代码表达出来。因为后根序列的最后一个结点肯定是根结点，因此对后根次序从后往前遍历。对于后根次序中最后一个元素a，首先构造结点，然后在中根次序中寻找该元素所在位置p。然后寻找后根次序中前一个元素b在中根次序中的位置q。如果该位置q在位置p的左侧，那么说明b是a的左孩子，因此先递归a的左子树，再递归a的右子树；如果q在p的右侧，那么说明b是a的右孩子，那么先递归a的右子树，再递归a的左子树。

注意：

从代码上看，无论是先递归右子树还是先递归左子树，传入参数是一模一样的，但这并不表示nextIter（即位置q）的计算是无用的，因为indexOfPostorder是引用，每次递归都会引起该值的变化！

Construct Binary Tree from Preorder and Inorder Traversal

#递归

要点：

和上一题高度类似。只是把从后往前遍历后根次序序列变为从前往后遍历先根次序序列。

Maximum Depth of Binary Tree+

#深搜

要点：

深搜，记录当前深度，大于最大深度时更新。

Binary Tree Zigzag Level Order Traversal

#层次遍历

要点：

和Binary Tree Level Order Traversal等题类似。只是对最终结果中奇数位置的数组进行倒置即可。

Symmetric Tree

#深搜

要点：

使用两个指针同步地对树的左右子树进行深搜。一个传入左孩子、另一个传入右孩子；反之亦然。

Same Tree

#深搜

要点：

使用两个指针同步地对树的左右子树进行深搜。一个传入左孩子、另一个也传入左孩子；反之亦然。

Recover Binary Search Tree

#中根遍历 #技巧

要点：

本题不能用二叉搜索树“左孩子小于根结点，右孩子大于根结点”来做，而应给想到二叉搜索树的中根次序是升序序列，利用这个性质来做。对树进行中根次序遍历，使用pre指针来表示当前遍历结点的前序结点。当发现pre结点大于当前结点时，使用wrong1指针指向较大的一个错误结点（即pre），wrong2指向较小的结点（这也就是为什么对wrong2的更新并不只有一次，因为出错的两节点间可能有一段元素）。最后交换wrong1和wrong2所指结点即可。

Validate Binary Search Tree

#深搜

要点：

用二叉搜索树“左孩子小于根结点，右孩子大于根结点”这一性质来判断。

Interleaving String

#动态规划

要点：

设立二维数组dp，行数和列数分别为字符串s1和s2的长度加1，其中每个元素dp[i][j]表示到s1[i-1]为止的子串和s2[j-1]为止的子串能不能构成到s3[i+j-1]为止的子串。基于以上这个设定，那么迭代式为：if(s3[i+j-1]==s1[i-1]) dp[i][j]=dp[i-1][j]||dp[i][j]; if(s3[i+j-1]==s2[j-1]) dp[i][j]=dp[i][j-1]||dp[i][j];

注意：

这道题还有一个递归解法。使用三个下标分别标记三个字符串的当前比较位置，如果s1[i]==s3[k]，那么递归i+1, j, k+1；如果s2[j]==s3[k]，那么递归i, j+1, k+1（此处的关键是这两种情况分别递归，而不是两种情况选一种递归）。递归方法比较容易相通，想通了递归方法后，上面这种动态规划方法也就不难理解了。

Unique Binary Search Trees

#技巧

要点：

此题的关键就是要想到k个结点的二叉搜索树的可能性是各种右子树\*左子树的可能性的求和。而右子树和左子树都是这个问题的跟小规模的问题。因此，使用一维数组c表示长度为i的二叉搜索树的可能性。c[k]=sigma(c[i]\*c[k-i-1])（减一是因为要有一个元素作为根结点）。

Unique Binary Search Trees II

#递归 技巧

要点：

本题的思想其实和上题是一样的，也是枚举根结点，对左右子树进行进一步地递归。关键是代码怎么写。因为对于同一段左子树/右子树，可能有多种树形结构，因此递归函数返回的是根结点数组，对于左右子树分别返回的两串根结点数组，再使用双层循环枚举所有组合与本函数当前根结点组合起来，把当前根结点放入结果数组中，最后返回结果数组

注意：

本题需要比较清楚的递归思维，值得多次推敲。

Binary Tree Inorder Traversal

#中序遍历 #栈

要点：

要求不能使用递归，那就用栈来代替。因为是中序，因此先把左子树全部压入栈，然后逐个弹出栈顶元素，访问该元素后把该元素的右子树的左子树再全部塞入栈中。

Restore IP Addresses

#贪心 #回溯

要点：

典型的贪心回溯题。要注意开头为0的数字。

Reverse Linked List II

#链表 #栈

要点：

在遍历链表的过程中，使用一个栈来保存要倒置部分的表元。当遍历到倒置部分末尾时，弹出栈中表元完成倒置。使用一个临时头指针便利编写代码。

Subsets II

#贪心 #技巧

要点：

很典型的贪心法。枚举每一个元素，放入结果集中，进一步递归。值得注意的是，每个递归函数开始前的结果集状态，都是一个有效解，因为题目要求的所有子集。

Decode Ways

#动态规划

要点：

一道简单的动态规划题。因为要求所有可能个数，因此想到动态规划。该题子规模的问题比较简单：要么是到前一个数字为止（当前数字单独解码），要么是到前两个数字为止（当前数字与前一个数字一起解码）。因此设一维dp数组，长度为给定编码串长度，dp[i]表示到第i+1个数字的解码可能。由上面易得，dp[i]=dp[i-1]+dp[i-2]（当前数字与前一个数字能够一起解码时才成立，否则只有dp[i]=dp[i-1]。如果当前数字为0，那只能尝试和之前数字一起解码，而不能单独解码）。最后返回dp[s.size()-1]即可。

Gray Code

#技巧

要点：

本题全凭技巧。(二进制码>>1)^二进制码=对应的格雷码

Merge Sorted Array

#技巧

要点：

直觉做法是另开一个数组，把两个数组的元素按大小插进去，再把合并后的数组赋给原数组。更好地方法是对两个数组从后向前进行比较遍历，然后从后往前地插在原数组里，这样空间复杂度是常数。

Scramble String

#递归

要点：

本题理解题意后如果想到二分递归做法其实很简单。就是比较当前递归函数的字符串和原串是否相符，如果不符则枚举区间内每个子段与原串的相同部位的子段进行递归比较，再把该子段与原串中靠后的相同长度的子段进行递归比较。

注意：

为了减少递归次数，当发现当前字符串与原串不完全一致时，可以进一步比较两字符串的每个字符的数量，当字符数量都不一致时，就没有继续二分递归的必要了，这样可以减少递归次数。

Partition List

#链表

要点：

使用两个指针，分别指向比给定值小的表元和比给定值大的表元，最后把两个链表串在一起。使用临时头指针遍历编写代码。

Binary Tree Right Side View

#二叉树 #层次遍历

要点：

层次遍历，每层遍历完成后插入一个标记。当遇到标记时输出前一个数字至结果集。

House Robber

#动态规划

要点：

求不相邻的数的最大和。