Lintcode 解题思路

使用“不太明白”标记不会的题目

2.尾部的零

计算阶乘结果最后零的个数，就是考虑乘法因式分解后2\*5的对数，显然2的个数要远远多于5的个数，所以只需要考虑5出现的次数即可，注意处理5的幂时，对于一个数而言提供了多个5，所以在计算时，只需要算出该数除以5，除以25，…，直至除以一个次小于该数的5的幂，然后将除法的结果相加即可。

3.统计数字

统计某数字出现的个数，对于每一个数，都进行分位处理，比较每一位的结果即可，分位时，先通过除10的幂统计出该数的位数，然后从高位低位逐次划分。注意：根据数据的长度采用long long（64），long（32），int（16）。还有注意特殊情况0的处理。

4.丑数Ⅱ

丑数就是因式分解后，只有2，3，5的幂。寻找第n大的丑数，即把丑数一一列出，然后找第n个就行。设置三个量，分别记录2，3，5的幂次。第一个丑数是1，所以三个幂次都是0，然后在第一个丑数的基础上，找乘2，3，5中最小的。故找到2，所以对应记录2的幂次数加一，第二个丑数是2，然后在第二个丑数乘2，第一个丑数乘3，第一个丑数乘5中寻找最小的。依此类推，使得每一个丑数都按照由小到大的次序合理放置。

5.第k大元素

题目要求O（n）的复杂度，参考快速排序的random-partion，使用select算法解决，注意生成随机数的时候选取时间做种子，保证生成的随机数不重复。

6.合并排序数组Ⅱ

合并两个排序好的数据，用两个下标指向数组头，然后比较之后依此放进新数组即可。一个下标到了最右端，就将另一个数组当前下标所在的位置直到数组末端粘贴到合并数组的末尾即可。

7.二叉树的序列化和反序列化

二叉树无论序列化还是反序列化，都采用队列来实现，序列化的操作基本上与层次遍历一致，不同的是注意孩子为空时要保持输出，不为空进队处理。反序列化的操作进行时，关键是通过一个下标来记录序列当前的位置，每一次非空子节点进入队列，或者生成空子节点都会使得下标增加，逐次遍历整个序列。此时队列的目的是为了记录当前的父节点。

**补充：如果从下标从1开始存储，则编号为i的结点的主要关系为：  
双亲：下取整 （i/2）  
左孩子：2i  
右孩子：2i+1  
如果从下标从0开始存储，则编号为i的结点的主要关系为：  
双亲：下取整 （(i-1)/2）  
左孩子：2i+1  
右孩子：2i+2**

8.旋转字符串

该题对于字符串的旋转可有两种解法，要不采用逐次移动（节省空间），要不采用字符串分割然后拼接起来（节省时间）。关键在于对于偏移量，要对字符串长度取模然后进行计算，还有就是对空字符串这种特殊值的处理。

11.二叉查找树中搜索区间

二叉查找树使用中序遍历可以将树中的值由小到大排列，只需更改中序遍历，在判断时候进行区间选择即可。本题没有多余的时间限制，如果有的话，可以采用适当的剪枝处理，如果当前节点的值小于或者等于搜索区间的下界，那么该区间的左子树可以不必再继续向下遍历。如果当前节点的值大于或者等于搜索区间的上界，那么该区间的右子树可以不必再继续向下遍历。

12.带最小值操作的栈

算法导论中有一道用两个栈来实现队列的问题，参考这个思路，采用两个队列来实现栈。push操作时，只需向非空队列中放置即可。pop操作时，将一个队列中的值逐个弹出并压入另一个队列，并把最后一个元素返回。最小值操作类似于pop，相当于每次把一个队列里的值拿出来看一遍再存到另一个队列中去。

15.全排列

c++中有一个库函数可以解决全排列的问题

template <class BidirectionalIterator>

bool next\_permutation (BidirectionalIterator first, BidirectionalIterator last );

从第一个元素起分别与它后面的元素主意交换

自己编写通过递归来实现（但是这种方法生成的全排列不是字典序）。abc全排列可以考虑为a的全排列加上bc的全排列。因而把该问题转化为一个递归的问题。在递归式中，每次选取放在第一个位置值，然后逐次移动这个位置进行递归，直至移动到尾部，每次移动到尾部产生一种排列。特别需要注意的是在每次递归完成之后，要做一次复位操作，回溯到交换之前的状态。

两次交换的原因是，假如当前求123的全排列，1和3先交换，变成了321，然后递归2和1交换，变成了312，然后逐层返回，还是123。接着从2开始向后交换。否则递归就会从上次交换的321这个结果向下递归，会漏掉很多情况。

全排列也可以不用交换，使用深度遍历，每次放一个值即可，但是要注意，每次放之前需要查询这个值是否使用过。

\*带重复元素的全排列

带重复元素的全排列，在交换时要注意，交换i，j时，要保证[i,j）中没有与j位置上重复的元素。

17.子集

这个问题采用二进制的思想，因为对于一个含有n个元素的集合来说，这个集合共有2^n个子集，所以可以和二级制相结合。可以从0到2^n生成2^n个n位的二进制数，然后每一位和集合中的元素一一对应，如果为1就加入到集合，如果为0则不加入，从而产生2^n个子集。

20.骰子求和

这个问题采用动态规划的方法来解决，投掷n枚骰子所得到的结果的可能的取值为n到6n。使用dp[i][j]来表示有i枚骰子进行投掷，产生的点数和为j的次数。递推式可以总结为dp[i][j] = dp[i-1][j-1] + dp[i-1][j-2] + dp[i-1][j-3] + dp[i-1][j-4] + dp[i-1][j-5] + dp[i-1][j-6]。当然，要保证第二个维度j-k的值大于零。根据初始值dp[1][1]=dp[1][2] =dp[1][3]=dp[1][4] =dp[1][5]=dp[1][6]=1可以逐步推出每一种可能的取值的出现次数。再把次数除以6^n就可以得到相应的概率。

22.平面列表

这个问题才用所给的类成员函数使用递归的方式就可以解决。

24.LFU缓存

做这个问题首先要了解LFU的基本原理，对于每一个存储的项，要存储对应的调用次数和调用的时间（用一个全局变量来模拟时间的流逝），get方法较为简单，如果找到就将该块的调用次数加一，并将该块的位置提升到最前。如果找不到就返回-1。set方法中包含一个移除函数，当有新的块加入时，如果该块原来在缓存中有，那么将其调用次数以及调用的时间更新，并放置在最前。如果新的块原先在缓存中未出现，那么如果当前缓存未满，则总是将其放在缓存的最前。如果此时缓存已经满了，那么淘汰一个使用次数最少（第一条件），调用时间最老的块（第二条件）。

28.搜索二维矩阵

矩阵中的元素按递增的顺序排列，则可以先通过第一列判断目标值所在的行数，然后再去对应的行进行搜索即可。（O（n）+ O（m））

32.最小字串覆盖

首先根据目标字串确定在原串中可能的起始位置，然后从每个起始位置出发来寻找包含目标串的字串，并记录其中最短的作为结果输出。

33.N皇后问题

采用回溯法。皇后放置的原则是两个皇后不能放在同一行，同一列或者同一条对角线上（写一个函数判断当前位置是否可以放置皇后）。对于放置的结果矩阵，采用从上到下，从左到右的次序依次进行放置。从第一个位置开始，如果该位置可以放置一个皇后，那么进行判断，如果这是放置的第n个皇后，那么产生了一种可行解，放入解集，并将该放置位置为未放置（回溯，原理可参考全排列）。如果不是第n个那么先判断下一行从第一列开始能否放置。调用完之后要复位方便回溯。然后判断该行的下一列能否放置。如果该位置不能放置，那么判断是否已经放到了这一行的最后一列，如果是，那么不能放置在这一行，如果不是的话，可以选择下一列继续放置。

34.N皇后问题Ⅱ

该问题只统计可以放置的结果，与上一个问题的思路完全相同，只是统计的时候不记录放置结果而只记录放置的次数即可。

35.反转链表

该问题使用两个指针来完成操作，初始化时，一个为空，另一个指向链表的头部，同时声明一个指针记录第二个指针的下一个节点，防止断链，然后改变声明的两个新指针的指向关系，并逐次向后迭代直至链尾。

36.反转链表Ⅱ

该问题要求反转链表中的部分链，关键在于找到反转部分的前一个节点和反转部分的后一个节点，首先新建一个指针指向整个链表的表头，这样即便是从第一个节点开始反转也可以找到反转部分的前一个节点。然后逐次向后寻找，记录反转开始的位置和反转结束的位置。此时声明一个新指针指向反转结束的下一个位置。然后使用反转链表的思想将反转部分反转。接着将链表重新组装。经过反转，原先的头节点变成了尾节点，此时使用反转部分的前一个节点指向反转部分完成后的头节点（反转前的为节点）。反转部分的尾界点指向之前记录的反转结束的下一个节点。从而完成整个反转，链接的过程。

37.反转一个3位整数

使用字符串来处理，反转过后去掉第一个非零数字前的零就可以了。

38.搜索二维矩阵Ⅱ

该题目所给出的二维矩阵每一行，每一列从左至右，从上到下都是有序的，所以在搜索的时候，使用每行，列的第一位来决定是否要搜索该行，列，从而简化搜索的过程。

39.恢复旋转排序数组

题目要求O（n）的复杂度，因为数组旋转之前是按照升序排列的，所以首先找到一个降序的位置，然后把该位置之后的剩余部分换到数组最前端就完成了数组的恢复过程。

40.用栈实现队列

用两个栈来实现队列，主要是pop操作时，先把一个栈中的k-1个元素弹出并压入另一个栈，注意不用再返回操作，如果再次pop直接从另一个栈中取就可以。

41，42.最大子数组1，2

这个问题可以使用一次遍历求解，使用一个值记录当前子数组和的最大值，如果记录和为负数，那么从0开始记录（从第二个值开始循环），直至找到最大值。对于找到两个子数组的和最小，可以新建两个数组，一个从左往右找，一个从右往左找，然后通过数组可能断开的位置进行组合，直至找到最小值。

43.最大子数组3（再斟酌）

这个问题我自己想了一个递归的解法，就是从左向右进行划分，把数组分成段，总共切k-1刀，有点类似于钢条切割，可是通过不了全部测试，还是有bug。

AC的解法是使用两个二维数组进行动态规划，其核心点在于对于一段数组最后一个元素是否取。数组local[i][j]表示到第i个元素，分成j个子数组和的最大值，其中必须包括第j个元素（这一点的含义我还不清楚）。数组global[i][j]则表示到第i个元素，分成j个子数组和的最大值。

状态转移的过程为，如果i<j则数组可以填一个非法值。

其中，local[i][j] = max(local[i-1][j],global[i-1][j-1]) + num[i-1](第i个，数组下标从0开始)。 //是否把j包含到最后一个数组中

如果i==j则两个数组的值相等。

否则global[i][j] = max(local[i][j],global[i-1][j])

44.最小子数组

和最大子数组解法一致，只需要把sum在大于0时清零就可以了。

45.最大子数组差

类似于最大子数组2中找两个和最大的子数组，可以从左到右，从右到左分别找最大和最小值，然后计算差值，选出最大的一个即可。

46.Majority Element

算法导论课程上的思路，采用消去法，从左到右遍历一次，记录当前的值和次数，进行分析就可以得到主元。

47.Majority Element2

我采用了一个时间空间复杂度都是O(n)的算法，遍历一次，通过map进行记录，如果有合适的值就返回。

时间复杂度O(n)，空间复杂度O(1)的算法，也是采用消去法，从左到右遍历一次，与上一个题不同的是采用两个值来记录当前的值和次数，逐次替换，进行分析就可以得到主元。

48.主元素3

采用和47相同的算法，只是判断条件有不同，只要大于n/k就返回结果。

49.字符串大小写排序

一次遍历，将大写字母放置在最后，小写字母放置在头部即可。

50.数组剔除元素后的乘积

遍历数组的每一个位置，求剩下的数字的乘积，可以采用记忆搜索，并且遇到0则不必要往后继续乘的手段来减少运算时间。

51.上一个排列

带有重复元素全排列也可使用15题回溯的方法，但是每次放置时需要做一下检查，比如要将i和j位置的元素进行交换，则需要检查从i+1到j-1是否出现过与在j位置元素相同的元素，如果有就不需要再重复交换了。注意一个元素可以与自身进行交换。

使用algothrim中的next\_permutation(nums.begin, nums.end())以及prev\_ permutation(nums.begin, nums.end())函数来做，注意处理头和尾的特殊情况。

52.下一个排列

与上一题思路一致。

53.翻转字符串

该题目注意题意是反转句子的顺序，句子中的每个单词不变。使用一个spilt函数来进行字符串分割，然后通过除去多余的空格，把单词逐个解析出来并按照逆序拼接起来就可以了。

54.String to Integer

该题目没什么难度，但是要考虑清楚每一种情况，特别是符号位的判断，理解题意，该题目中如果中间有非数字字符将数字分割，那么只取前一段。

55.比较字符串

水题，主要是注意目标串的中的一个字母可能对应母串中的多个字母，只记录一次就行了。

56.两数之和

采用O(nlogn)的解法，对于每一个值，在后边找除去这个值得目标值（注意目标值得变化），采用二分查找，降低复杂度，注意二分查找找不到时的返回条件。可以采用hash映射的方法再次降低复杂度。

57.三数之和

首先对数组进行排序使用三个索引来遍历数组，注意索引的变化，i从头到尾，j从尾开始，k从i+1到j-1，每次计算三个索引所产生的和，根据结果和目标值的差距，k和j的值会相应变化。其中注意去重操作，在i，j，k变化的过程中增加的变量，比如i，k，其需要和下标加一的值做比较，如果重复则不用再次寻找，减少的变量和下标减一的值做比较，如果重复则不用再次寻找。

58.四数之和

思路和三数之和一致，多使用一个下标来记录，注意去重和循环的边界条件即可。

59.最接近的三数之和

这个题基本和三数之和一致，只是记录所计算出来的和与目标值的最小差值即可。

60.搜索插入位置

可以使用二分法。我采用一次遍历求解，注意边界值条件。

61.搜索区间

采用二分法解决，这个问题的二分法有些不同，因为是找一个区间，所以分别使用两次二分法来寻找左右区间，对于每一区间，如果通过中间值找到了目标值，注意不要返回，而是继续再次寻找旁边的值，如果是找右端点，那么再找右边的一个值，如果依然等于目标值，那么需要把此时右边界拓宽再次循环。左边也是一样，知道找到最右或者最左边的目标值。还有就是注意特殊情况的处理，不要造成边界溢出的问题。

62.搜索旋转排序数组

数组中没有重复元素，依然采用二分法来解决，要注意旋转排序数组中有两个左右两个排序数组，注意采用二分法的时候要注意判断条件，首先判断当前这一段数组是否是排好序的，如果有序（两段有序数组的其中一段（start到mid，mid到end）），那么如果要找的目标值在当前start，end所限制的一段数组中，就按照普通的二分法来解决，否则就去另一段寻找。

63.搜索旋转排序数组2

数组中有重复的元素，在上一题思路的基础上，在使用mid和start或者end值进行比较来判断当前子数组是否有序时，增加判断相等的情况，如果相等那么索引相应进行移动，以保证每个搜索区间内，将端点重复的值过滤掉。

64.合并排序数组

通过两个索引分别指向两个数组，然后从头至尾滑动索引，逐个比较即可。

65.两个排序数组的中位数

这个问题可以转化为在两个排序数组中寻找第k大元素的问题。

因为对于数组而言，其长度是奇偶数决定其中位数的位置，通过虚拟补‘#’来使数组长度均为奇数，之前的索引是当前索引除以2。

Eg. 1,2,3 变成#,1,#,2,#,3,#

L1，R1，L2，R2分别代表两个数组划分后分割位置前后的值。C1，C2表示分割的位置。

保证 C1 + C2 = m+n并且L1 < R2 ，L2 < R1即完成了分割对较短的数组进行二分查找，在对L1，L2，R1，R2进行赋值的时候，注意C1,C2到达边界值的情形，并且按照上述虚拟补的方式。有固定关系式：

L1 = (C1 == 0) ? INT\_MIN : nums1[(C1-1)/2]

R1 = (C1==2\*n) ?INT\_MAX: nums1[(C1)/2]

L2 = (C2 == 0) ? INT\_MIN : nums2[(C2-1)/2]

R2 = (C2==2\*n) ?INT\_MAX: num2[(C2)/2]

如果不满足上述大小关系，则说明对数组的划分不均衡，比如如果产生L1 > R2的情况，说明数组一划分的太靠右，需要让划分位向左移动以保证L1 < R2。另一种情况同理。

66.二叉树的前序遍历

递归的方法比较简单，如果节点不为空，那么输出该节点的值，再递归遍历左子树和右子树即可。

非递归的方法用栈来实现，先将根节点压栈，然后弹出栈顶元素，若该元素不为空，取得相应的值后分别将其右孩子和左孩子压栈，重复该操作直至栈为空。

67.二叉树的中序遍历

递归的方法比较简单，如果节点不为空，那么先递归遍历左子树，然后输出该节点的值，最后遍历右子树即可。

非递归的方法用栈来实现，从根节点开始向下寻找，如果当前节点不为空或者栈不为空，将当前节点的左孩子压栈，直至找到最后一个。若栈不为空，那么弹出栈顶的元素并输出，如果当前节点不为空，那么从当前节点的右孩子开始寻找。

68.二叉树的后序遍历

递归的方法比较简单，如果节点不为空，那么先递归遍历左子树，然后遍历右子树，最后输出该节点的值即可。

非递归的方法用栈来实现，首先把根节点入栈，如果当前节点不为空或者栈不为空则继续循环寻找，使用栈顶元素作为当前节点，如果其左孩子或者右孩子都为空或者上一个遍历的节点不为空，并且当前节点等于上一个处理节点的左孩子或者右孩子（说明该节点的左孩子和右孩子均被处理过）那么可以将栈顶元素弹出，并将当前指针保留为上一个处理的节点指针。否则的话如果该节点的右孩子不为空，则压栈，左孩子不为空则压栈。注意为了保证顺序，因而先压右，再压左。

72.中序遍历和后序遍历构造二叉树

这个题目主要在于考分割位置的顺序。使用递归的思想，首先通过后序遍历的最后一位找到树的根节点，然后通过根节点在中序遍历中的位置将这个序列分成两个子树的中序遍历的结果加上根节点，根据两个子树中序遍历的结果再在后序遍历中寻找等长度的后序遍历，然后递归调用生成结果。（注意找到断开位置时，最好使用for来避免位置切割的问题，划分过程中前一个子树中序遍历和后序遍历的标号一致，由于根节点的隔开，后一个子树中序遍历和后序遍历的标号差一，后序遍历小）。

73.前序遍历和中序遍历构造二叉树

与上题思路一致，首先通过前序遍历第一位找到根节点，再在中序遍历中寻找直到完成，由于根节点隔开，第一个子树前序遍历的序号比中序遍历的序号大一位，后边一棵子树相同。

74.第一个错误的代码版本

采用二分法找到第一个错误的值，但是有一个case始终过不了。注意这个题目在进行划分时候的边界。

75.寻找峰值

可以一次遍历求得峰值，也可以通过二分法，因为首尾的元素可以抽象成负无穷，所以可以通过找递增的值来确定二分法的区间划分。

76.最长上升子序列

使用d[i]表示从0到结尾为i的值的最长上升子序列的长度，对于每一个i值，使用0到i-1的最长上升子序列的长度递推得到，如果nums[i] > nums[j]并且d[i] > d[j]+1那么 d[i] = d[j] + 1，从而可以自底向上递推得到最长上升子序列的值。

77.最长公共子序列

经典的动态规划题目，使用d[i][j]表示两个字符串到i,j位置最长公共子序列的长度。递推式为若Xi == Yi d[i][j] = d[i-1][j-1] + 1,否则d[i][j] = max(d[i-1][j],d[i][j-1])。

78.最长公共前缀

求k个字符串的最长公共前缀，首先找到这些字符串中最短的，长度为L，所以最长公共前缀的长度不会超过L，两层循环，第一层0到L-1第二层0到K-1。从第一位开始逐位寻找，直到找到所有的公共前缀。

79.最长公共字串

这个题目要求的最长公共字串是连续的，所以采用常规思路逐个求解就行，要注意的是，一个字符串中的一个字符可能在另一个字符串中出现多次，所以要注意对于每一位都要从头比较直到尾部，计算出最优值。

80.中位数

这个题目把计算中位数的方法转变为计算数组中中间大的数，使用算法导论中的random-select算法解决，复杂度为O(n)。核心在于partion算法的实现。

81.数据流中位数（vector中push\_heap）

push\_heap(min\_heap.begin(),min\_heap.end(),greater<int>());

push\_heap(max\_heap.begin(),max\_heap.end(),less<int>());

pop\_heap(min\_heap.begin(), min\_heap.end(), greater<int>());

这个题目解法比较新颖，使用一个最小堆，一个最大堆来做。其中最小堆存储数据流的较大的部分，最大堆存储数组较小的部分。注意，两个堆中元素数目的差值不超过一。我采用比较笨的方法，把1，2两个元素当作特殊情况处理，这样两个堆的初始状态都是有值的，不会为空。对于数据流中一个新来的元素，有以下几种情况，如果元素的值在两个堆顶元素之间，那么将其插入到较小的那个堆中（注意堆的变化）。如果元素的值比最小堆堆顶元素大，那么将其插入最小堆中，如果比最大堆堆顶元素小，那么插入最大堆。注意，插入之前判断如果操作后两个堆的元素的差值大于1，则需要将较大的堆的栈顶元素弹出并置入另一个堆，然后再插入。每次有新元素进入求中位数。如果两个堆的大小不相等，则中位数是较大堆的堆顶元素，否则是两个堆顶元素的均值。

82.落单的数

这个题目中，目标数出现一次，其余的数出现两次，所以可以通过异或的方式来做，相同的数异或为0抵消掉了，最后剩下目标数。

第二种方法是把所有的数字转化为二进制，逐位相加并对2取余数，这样就把相同的数对该位的贡献过滤掉了，最后只剩下目标数。注意，对于一个数可以通过向右移动相应的位数（从0开始）并与1按位与就可以得到该数字二进制从低到高的相应的位数，注意按位与运算符的优先级较低，要加括号处理，最后再把得到的二进制转化为十进制数返回即可。

83.落单的数2

该题目可以采用上一题目中的方法，因为除了目标数别的数出现三次，所以需要把逐位相加的结果对3取余数。别的都与上一题一致。

84.落单的数3

该题目我采用较笨的方法，使用map来记录每一个数出现的次数，然后遍历map得到目标值。复杂度为O(2n)

85.在二叉树中插入节点

水题，二叉搜索树满足根节点比左孩子大，比右孩子小，所以递归查找到插入的位置即可。

86.二叉查找树迭代器

参考中序遍历二叉树的非递归算法。使用一个栈来存储当前的遍历情况，如果栈为空，那么就没有下一个值了，注意stack的empty()方法，空返回true非空返回false。

较笨的方法是把中序遍历的结果都存入到栈里。但是这样空间复杂度为O(n)。

采用较为好的方法可以把空间复杂度将为O(h)，h为树的高度。即初始化时，先把当前二叉搜索树一致向左搜索，并逐个压入栈中。调用next方法时，通过pop弹出栈顶元素并输出。考察该元素是否还有右子树，如果有的话从右子树的根节点开始向左遍历，直到节点的左子树为空，并把这些结点压入栈中。其主要的思想和非递归的中序遍历是一致的。

87.删除二叉查找树的节点

使用递归的方式查找，如果要删除的节点大于根节点，那么根节点的右子树等于删除当前根节点的右子树删除目标节点。如果小于根节点，那么根节点的左子树等于当前根节点的左子树删除目标节点。如果等于根节点，那么删除根节点，有三种情况。如果当前节点的左孩子和右孩子均为空，那么之间删除该节点即可。如果左，右孩子一个为空，一个不为空，那么当前节点等于其非空子节点。如果两个孩子均不为空，那么用其右子树中的最小节点代替当前节点，当前节点的右子树等于原来的右子树删除最小的节点。

88.Lowest Common Ancestor of a Binary Tree

寻找二叉树中两个节点的最近公共父节点，其实把这个问题转化为寻找两个点的相对位置关系上，采用先序遍历。如果找到的节点就是两个节点之一，那么返回当前节点（最近公共父节点也随之产生）。否则分别在两个子树中寻找，如果两个子树寻找的结果都不为空，说明两个节点分别处于当前根节点的两个子树中，因而最近父节点就是当前的根节点。如果一边为空，那么返回在另一边的查找结果，两边都为空则说明查询失败。

89.K数之和

这个题目只统计次数，所以采用动态规划的方法，类似于背包问题。dp[i][j]表示i个数字和为j一共有多少种组合。通过对数组中的每一项进行遍历。

dp[i][j] += dp[i-1][j-A[k]]

注意自顶向下开始遍历，初始条件dp[0][0] = 1。

90.K数之和2

这个题目要求求出要求的和式的可能的情况，采用深度遍历的方法

Sum(vector<int> &A, int k, int target, vector<vector<int> > &result, vector<int> temp, int i)

注意深度遍历结束的条件是k==0&&target==0或者i<=A.size() | |target<0。

注意每次试探之后要将temp中的值弹出，再进行新一次的遍历。

temp.push\_back(A[i]);

sum(A,k-1,target-A[i],result,temp,i+1);

temp.pop\_back();

sum(A,k,target,result,temp,i+1);

每个函数都push pop一次，函数结束后pop完就和之前的状态一致。

91.最小调整代价

这个题目要求调整数组中的数，使得数组中相邻的数目的差的绝对值不超过target。采用动态规划的思想，有点类似于背包问题。dp[i][j]表示前i个数字第i个数调整到j所需要的代价。因为数组中的数字都小于等于100，因而j的范围是0到100。首先初始化数组，dp[0][j] = abs(j-A[0])。然后在求dp[i][j]的时候，这一项的调整代价为j-A[i]，可以推出递推式

dp[i][j] = min(dp[i-1][k]+j-A[i],dp[i][j])，其中k的范围是从max(0,j-target)到min(100,j+target)。可以逐步得到数组的内容。构造最优解的过程是从0到100，找dp[n][x]的最小值。

92.背包问题

这个是最简单的01背包问题，使用背包问题的通用解法来做，

dp[i][j]表示容量为j的背包装前i件物品时候的最大重量。

dp[i][j] = max(dp[i-1][j],dp[i-1][j-A[i]] + A[i])

注意这种情况是j >= A[i]，如果装不下，还是要注意状态转移。

dp[i][j] = max(dp[i-1][j],dp[i][j])

最后可以通过dp[n][m]得到最优解。

93.平衡二叉树

首先明确求二叉树的最大深度的递归求解方法，注意要有一个变量来衡量当前左右子树的高度差，如果有子树的高度差大于一，则该树不是平衡二叉树。

1. **int** highOfTree(TreeNode \* root,**int** &flag) {
2. **if** (root == NULL) {
3. **return** 0;
4. }
5. **int** left = highOfTree(root->left,flag) + 1;
6. **int** right = highOfTree(root->right,flag) + 1;
7. **if** (abs(left-right) >= 2) {
8. flag = 0;
9. }
10. **return** (left > right)?left:right;
11. }

94.二叉树中的最大路径和

二叉树中寻找，自然想到递归求解。如果根节点为空则返回0。否则递归求解左子树，和右子树的最大和。用一个值保存循环过程中最大的路径和：

Max = max(max(0,left)+max(0,right)+root->val,Max)

递归函数返回左子树或右子树和根节点和的最大值用来递归，可以求出最后的解。

95.验证二叉查找树

二叉查找树的中序遍历的结果是递增的，使用这个性质来做，先对二叉查找树中序遍历，根据结果是否递增来判断。

96.链表划分

使用两个小链表拼出结果，对原来的链表一次遍历，把大于等于的放在一个链表，小的放在一个链表，然后把两个链表接起来就可以了。

97.二叉树的最大深度

使用递归的方法，如果根节点为空，则返回0，否则递归求解左子树和右子树的最大深度（注意加一操作，带上根节点），返回左右子树深度的较大值，便可以完成求解。

98.链表排序

题目要求O(nlogn)的复杂度，所以使用归并排序来解决这个问题。使用快慢指针来找到链表的中点，然后对两段进行排序，核心在于merge函数，两个指针从头至尾指向两个子数组，然后逐个比较并合并即可，注意处理一个链表的指针到尾，另一个链表的情况。

99.重排链表

这个题目很蛋疼，思路很简单，首先使用快慢指针找到链表的中点，然后把后半段链表反转，接着把后一段链表插入前一段链表中，注意反转链表的操作以及插入的操作细节。

100.删除排序数组中的重复数字

这个简单题目其实有点陷阱，使用两个量来记录当前的长度和当前的值，如果有新的值出现，那么更新长度和值，长度也对应着在数组中的位置，因为重复肯定比单个元素出现的要长，所以直接在原数组进行覆盖即可。注意这个题目只要求返回的长度之前有效，所以后边的不用考虑。

101.删除排序数组中的重复数字2

这个题目要求重复的数字可以出现两次，思路和上一个题目一样，只要在记录一次次数就行。

102.带环链表

使用快慢指针来做，如果到尾部为空，那么一定没有环路。否则快指针会追上慢指针，两者有相等的情况发生。

103.带环链表2

这个题目不仅要求判断链表是否有环，还需要判断环的入口在哪。首先使用快慢指针找到环（注意循环的条件fast != NULL && fast->next != NULL），然后快指针指向链表头，再循环一次。就找到环的入口，注意循环的条件。

1. **while** (slow != fast && fast != NULL &&slow != NULL) {
2. slow = slow->next;
3. fast = fast->next;
4. }
5. **return** slow;

104.Merge K Sorted Lists

这个题目采用一个比较通俗的方法，针对原来的链表数组，再建立一个链表数组存储每一个指针的位置，然后每次循环遍历，取最小的放在结果链表的链尾，直到所有的链表都为空。

105.复制带随机指针的链表

这个题目主要考察深拷贝和浅拷贝的区别。深拷贝是新开辟内存进行拷贝，之后对拷贝的值修改不影响原值，浅拷贝是指针指向原来的区域，对拷贝值修改影响原值。

106.Convert Sorted List to Binary Search Tree

首先明确二分查找树的中序遍历的结果是一个有序序列，所以首先使用快慢指针把链表分成两半，然后递归进行操作，特别需要注意的是，找到的mid

的前一个节点作为根。而是把mid->next作为根，然后通过mid->next = NULL，mid->next->next把链表分成两半。这个题目注意如何找到中点的上一个值：

1. ListNode \* middle(ListNode \* head) {
2. **if** (head == NULL) **return** head;
3. ListNode \* fast = head;
4. ListNode \* slow = head;
5. ListNode \* pre = head;
6. **while** (fast != NULL && fast->next != NULL) {
7. fast = fast->next->next;
8. pre = slow;
9. slow = slow->next;
10. }
11. **return** pre;
12. }

107.单词拆分1

该题目需要考察能否将一个字符串划分为字典中的量，使用动态规划的解法。使用dp[i]表示到第i个字符，前i个字符组成的字符串能够被分割。使用两个循环：

1. **for** (**int** i = 1;i <= s.length();i++) {
2. **int** start = (i>max\_length)?i-max\_length:0;
3. **for** (**int** j = start;j < i;j++) {
4. string temp = spilt(s,j,i);
5. auto it = dict.find(temp);
6. **if** (dp[j] && it != dict.end()) {
7. dp[i] = **true**;
8. **break**;
9. }
10. }
11. }

注意第二个循环不必每次都从0开始，直接从字典中最长的开始做就可以。

108.分割回文串2

该题目动态规划的解法，有点与上一题类似，考察将字符串分割为回文串的最小分割次数，使用dp[i]表示到第i个字符组成的字符串划分成回文串所需要的最小划分次数，可以推出递推式，如果j到i为回文串，dp[i] = min(dp[j] + 1,dp[i])。还有就是要注意使用矩阵表示从i到j是否为回文串，这个矩阵的的构造过程分三种情况。长度为一的字符串一定为回文串，长度为2的串通过两个元素判断是否相等来决定是否为回文串。长度超过2的按照长度递增的顺序循环，矩阵[i][j]限定字符串为回文串的条件是，两端字符相等，[i+1][j-1]为回文串。

109.数字三角形

该题目采用动态规划的解法，使用dp[i][j]表示最后一个元素选择第i行，第j列的元素所产生的最小值。可以得出递推式为。

dp[i][j] = min(dp[i-1][j],dp[i-1][j-1]) + triangle[i][j];

注意对于j的范围限制，j的范围为0到i-1(第i行，上一行的最大下标为i-1)。可以使用技巧来限制取值：

lo = max(j-1,0);

hi = min(j,i-1);

然后递推算出最后一行的值，再在最后一行中选择最小的就可以得到原问题的解。

110.最小路径和

计算矩阵从左上角到右下角的最小路径和，可以使用简单动态规划求解，递推式为：dp[i][j]表示到i行，j列最小路径数目。

dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + value[i][j]

111.爬楼梯

最基础的动态规划，每次可以选择走一步或者走两步：

dp[i] = dp[i-1] + dp[i-2]

112.删除排序链表中的重复元素

通过两个指针，从前向后找，找到不同的再变换第一个指针，第二个指针不断滑动到链尾。

113.删除排序链表中的重复元素2

使用三个指针，如果一个值和前一个和后一个都不同，就可以加在链表中，要注意的是最后一个指针要注意判定条件，因为最后一个值可能存在和上一个值不同，后边尾指针为空，所以也要加入链中，故要注意向后指时要注意判断。

114.不同的路径

也是使用动态规划，dp[i][j]表示到第i行，第j列有多少条不同的路径。

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

115.不同的路径2

也是使用动态规划，dp[i][j]表示到第i行，第j列有多少条不同的路径。

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

如果有障碍物，那么dp[i][j] = 0;

116.跳跃游戏

通过给出的数组来判断是否能到终点，使用动态规划和贪心算法两种方法求解。

动态规划：使用dp[i]表示到第i点是否可达，判断条件是从0到i中，如果dp[j] == 1 && A[j] >= i-j则说明改点可达，从头至尾遍历即可。

贪心算法：维护一个可以到达的最远距离和当前的距离，如果可以到达的最远距离小于当前距离，说明不可达。否则用当前可以到达的最大距离更新这个值，如果判断可以到达则结束判定。

1. **bool** canJump(vector<**int**> &A) {
2. **int** n = A.size();
3. **if** (n == 0) **return** **false**;
4. **if** (n == 1) **return** **true**;
5. **int** max\_gap = 0;
6. **int** cur = 0;
7. **while** (max\_gap >= cur) {
8. max\_gap = max(max\_gap,cur + A[cur]);
9. cur ++;
10. **if** (max\_gap >= A.size()-1)
11. **return** **true**;
12. }
13. **return** **false**;
14. }

117.跳跃游戏2

和上题不同的是，这个问题不仅需要判断是否可达，还需要判断最小的跳跃次数，使用动态规划的方式求解，再使用number[i]来记录到第i个点的最小跳跃次数。每次判断可达，再根据number[j] + 1和number[i]的大小更新当前最小值。

118.不同的子序列

求两个字符串中不同的子序列，使用动态规划的方法来求解。

递归式dp[i][j]表示第一个字符串到i，第二个字符串到j两者的子序列的数目。如果S[i] != T[j]那么dp[i][j] = dp[i-1][j]

否则，子序列的数目可以是S加入第i个字符或者不加入第i个字符(两种子序列)：

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

同时还要注意初始值的设置dp[i][0] = 0

119.编辑距离

经典的动态规划问题，递归式dp[i][j]表示第一个字符串到i，第二个字符串到j需要的最小编辑距离。

如果S[i] == T[j]那么dp[i][j] = dp[i-1][j-1]

否则，可以通过三种不同的变换手法(插入，替换，删除)，来做，比较最小次数

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

同时还要注意初始值的设置dp[i][0] = i，dp[0][i] = i

120.单词接龙

使用一个队列来辅助进行宽度搜索。

121.单词接龙2

122.直方图最大矩形覆盖（不太明白）

使用单调栈（栈中存储的是下标）的方式来做，首先要在高度数组末尾加一个0，保证最后一个块能够被算进去，然后如果栈为空或者当前的高度大于栈顶的高度，可以直接入栈（能够通过栈顶的高度构成矩形）。否则弹出栈顶元素，以栈顶元素作为矩形的高，矩阵的宽度为i(栈为空)或者i-s.top()-1因为第i个直方块是不满足条件的，所以不能计算在内。同时要注意，计算完之后，循环位置要减一，从高度较小的那个直方图重新开始。

1. **int** largestRectangleArea(vector<**int**> &height) {
2. **int** res = 0;
3. stack<**int**> s;
4. height.push\_back(0);
5. **for** (**int** i = 0; i < height.size(); ++i) {
6. **if** (s.empty() || height[s.top()] < height[i]) s.push(i);
7. **else** {
8. **int** cur = s.top();
9. s.pop();
10. res = max(res, height[cur] \* (s.empty() ? i : (i - s.top() - 1)));
11. --i;
12. }
13. }
14. **return** res;
15. }

上述两个题目不太理解，见代码。

123.单词搜索

使用深度遍历搜索，每次搜索可以从上下左右四个方向进行，注意在搜索前要把搜索开始的位置换成一个非字母值，搜索结束之后再换回来，以防止出现重复搜索的情况。注意深度遍历的结束条件。

124.最长连续序列

题目要求使用O(n)的复杂度来解决，使用map，key表示数组中的值，value表示所在其所在的最长连续序列的长度。从头至尾遍历一次数组，每个值的初始长度为1，如果是重复的元素那么不再判断，否则判断该值减一是否在数组中，如果在，根据比当前数小于一的数所记录的最长连续序列找到这个序列的起点，同时根据当前的数找到序列的终点，从而求出加入该值序列的长度，注意更新序列起点和终点所对应的序列长度。按照相似的方法在判断比当前数多一的数是否在序列中，综上方法求出最长连续序列。

127.拓扑排序

使用一个map记录所有点的入度，首先通过遍历邻接链表找到入度为0的点入队列。然后对队列进行循环，弹出队头元素，把该点的邻接点的入度减一，将入度为0的点进入队列。直到队列为空，我们可以通过广度遍历得到图的一个拓扑排序。

128.哈希函数

说是水题，但是要注意在计算时，要对每一次求出的结果取余数，以防止溢出。

129.重哈希

水题，注意链表的插入，以及对负数取余数的方法 a%b = (a%b+b)%b

130.堆化

首先根据堆元素的vector构建一个递归函数heapify，对于vector中的每个元素i，其子节点分别为2\*i+1，2\*i+2。然后根据最大堆或者最小堆的规则判断节点与其孩子节点的位置关系，如果位置需要调整，那么需要从调整到孩子节点的位置开始向下递归。

对于整个过程而言，从vector的n/2位置开始向前逐步遍历到根节点，对于每个位置使用递归函数，自底向上完成堆化。

1. **void** heapify(vector<**int**> &A,**int** i)
2. {
3. **int** ld = 2\*i+1;//数组A下标从0开始 所以i节点的左右孩子节点为2i+1、2i+2
4. **int** rd = 2\*i+2;//
5. **int** minpos = i;
6. **if**(ld < A.size() && A[ld] < A[minpos])
7. minpos = ld;
8. **if**(rd < A.size() && A[rd] < A[minpos])
9. minpos = rd;
10. **if**(minpos != i)
11. {
12. swap(A[i],A[minpos]);
13. heapify(A,minpos);
14. }
15. }

131.The Skyline Problem (超难，不太明白)

132.单词搜索2

参考123单词搜索1的解法，注意深度遍历前后字母的至位（防止重复搜索）。

133.Longest Word

水题，使用一次遍历，通过一个中间vector记录当前最长的单词组，如果有更长的单词出现则清空之前的vector，再重新开始记录。

134.LRU缓存策略

使用链表实现，查询时，遍历链表，并把查询得到的缓存放在链头。插入时，如果key之前存在，那么改变其value的值，并将其放在链头，如果key不存在则生成一个块放在链头，如果超出了缓存容量，则把链尾的块删除。

135.数字组合

求一个给定vector中，和为目标值的所有可能的组合。采用递归加回溯的算法来实现，注意递归结束的条件为sum == target或者sum > target。回溯的前后注意要回复到之前的状态。

在进行操作之前，vector要进行排序和去重，可以用STL vector中的标准库函数来做。

1. sort(candidates.begin(), candidates.end());
2. candidates.erase(unique(candidates.begin(), candidates.end()), candidates.end());

136.分割回文串

与上一题的解题思路类似，采用深度遍历加回溯的方式来解决，遍历结束的条件是循环到字符串的末尾。注意递归结束的条件和每次分割的取值：

1. **void** dfs(vector<vector<string>> &result, vector<string> &temp, string &str, **int** index) {
2. **if** (index == str.length()) {
3. result.push\_back(temp);
4. }
5. **for** (**int** i = index;i < str.length();i++) {
6. string s = str.substr(index, i - index + 1);
7. **if** (judge(s)) {
8. temp.push\_back(s);
9. dfs(result,temp,str,i+1);
10. temp.pop\_back();
11. }
12. }
13. }

137.克隆图

使用map来建立一个原图到克隆图的映射，同时使用另一个map来记录某个节点是否被访问过。接着对图进行广度优先搜索，使用一个队列来记录遍历的情况。如果某个节点未被访问过，那么建立原图节点和克隆图节点的映射关系，并在克隆图节点那端新建立节点并插入到克隆图中，并将该节点入队列，改变当前节点的访问状态。如果该节点被访问过，直接把该节点的邻接节点放入克隆图中（环），不再进入队列。如此反复直到队列为空。

138.子数组之和

139.最接近零的子数组和

这两个题目都可以用一种方法解决，首先遍历整个数组。建立一个map<int （sum）,int (index)>标识到第i个点的和，然后把这些pair放入vector中，使用sort进行排序（直接使用）。之后遍历排序好的vector找到差值最小的两个位置，然后根据pair中的第一个值来找出起始和中止的位置。

140.快速幂

首先判断特殊情况n = 1或者n = 0，然后使用如下公式递归计算。

(a \* b) % p = (a % p \* b % p) % p

注意如果n为奇数则需要把使用上述计算之后得到的结果temp,再使用上述公式计算一次：

temp = (temp \* a%p)%p

最后的值返回要注意类型的转换。

141.x的平方根

因为要求x2 = n的解，令f(x)=x2-n，相当于求解f(x)=0的解，可以求出递推式如下：

xi+1=xi - (xi2 - n) / (2xi) = xi - xi / 2 + n / (2xi) = xi / 2 + n / 2xi = (xi + n/xi) / 2。

所以使用递推的方式得到，直到xi 和xi+1的值相等（double判断相等只需要小于1e-6即可），最后的值返回注意类型的转换。

142.O(1)时间检测2的幂次

对于一个数，如果其满足n & (n-1)) == 0，则该数为2的幂。通过整个式子来判断就行，还要注意负数和0都不是2的幂。

143.排数字2

由于颜色肯定是正数 1 到 k，所以我们可以用负数比如 colors [i] = -k，表示第 i 种颜色在原来的数组里面出现了 k 次。

首先f遍历一遍原来的数组，如果扫到 colors[i]，首先检查 colors[colors[i]] 是否为正数，如果是把 colors[colors[i]] 移动colors[i] 存放起来，然后把 colors[colors[i]] 记为-1(表示该位置是一个计数器，计1)。 如果 colors[colors[i]] 是负数，那么说明这一个地方曾经已经计数了，那么把 colors[colors[i]] 计数减一，并把color[i] 设置为0 （表示此处已经计算过），然后重复向下遍历下一个数，这样遍历原数组所有的元素过后，数组 colors[i] 里面实际上存储的每种颜色的计数，然后我们**倒着**再输出每种颜色就可以得到我们排序后的数组。

注意：颜色从1开始标号，数组下标从零开始，要注意对应关系。

144.交错正负数

水题，首先根据正负数的多少来判断是正数还是负数放在第一位，然后对每一个数，如果和上一位符号相同，则从该数的位置向后找最近的一个符号不同的数更换位置即可。

145.大小写转换

小写字母的ASCII码比大写字母的ASCII码大32

148.颜色分类

解法与143排数字2的解法一样。要注意这里的颜色是从0开始标号的，还有此时不再是k种颜色。而是只有三种颜色，在倒序输出时需要注意。

149.买卖股票的最佳时机

求一次买卖股票交易的最大利润，只需一次遍历，保存到当前位置的最小值（保存历史最低价），然后根据每个位置与历史最低价的差值可以求出当前位置的最大利润，遍历到最后即可求出最大利润。

150.买卖股票的最佳时机2

可以完成尽量多的交易来计算最大利润，一次遍历，只要针对于每个位置，只要对比上一天有增量就可以进行交易，把结果加起来最后就是最终的结果。

151.买卖股票的最佳时机3

求完成两次交易的最大利润，可以使用枚举法来做。因为用户每次手里只能有一只股票，所以两次交易分开的位置可以是数组中的任一位置。

使用两个数组来记录，left[i]表示到从0到位置i所获得的最大利润（求解的方法与149一致，维护一个历史最低值）。right[i]表示从位置i到位置n-1（n为数组长度）所获得的最大利润（从末尾开始计算，维护一个历史最大值，类似于左边的生成方法）。

然后，一次遍历，计算left[i] + right[i]的最大值作为最后的结果返回。

393.买卖股票的最佳时机4

求完成K次交易可以获得的最大值。首先，如果交易的次数K大于或者等于数组长度的一半，那么该问题退化为150问题，直接求解即可。

否则使用动态规划的方式求解。使用local[i][j]表示前i天，最多进行j笔交易的最大值（局部最优，表示最后一笔交易在最后一天卖出），使用global[i][j]表示前i天，最多进行j笔交易的全局最优解。可以得到递推式如下：[diff表示prices[i] – prices[i-1]，即第i天和第i-1天的差价]

1. local[i][j] = max(global[i-1][j-1] + max(diff,0), local[i-1][j] + diff)
2. global[i][j] = max(global[i-1][j], local[i][j])

其中局部最优值是比较前一天并少交易一次的全局最优加上大于0的差值，和前一天的局部最优加上差值后相比，两者之中取较大值，而全局最优比较局部最优和前一天的全局最优。

152.组合

求n个数的长度为k的全排列，使用传统的回溯+深度遍历的算法即可，注意放置前后的状态变化。

1. **void** create(vector<vector<**int**> > &result, vector<**int**> &temp, **int** cur, **int** n ,**int** k) {
2. **if** (temp.size() == k) {
3. result.push\_back(temp);
4. **return**;
5. }
6. **for** (**int** i = cur;i <= n;i++) {
7. temp.push\_back(i);
8. create(result,temp,i+1,n,k);
9. temp.pop\_back();
10. }
11. }

153.数字组合2

解题思路与上一题一致，都是采用递归加回溯的方式来解决。注意其要求子集中的元素是按照升序排列的，因而需要先排序再操作。对于结果vector result我们使用如下方法去除其中的重复元素。

1. sort(result.begin(), result.end());
2. result.erase(unique(result.begin(), result.end()), result.end());

154.正则表达式匹配（不太理解）

这个问题采用逐字匹配，如果遇到’\*’则采用递归的方法探查每一种情况，如果某一位的下一位为’\*’可以直接跳过（因为可以认为出现0次），其余的逐位进行比较，直到完成匹配。

1. **bool** sub(**int** ls, **int** lp, **int** ps, **int** pp, string &s, string &p)
2. {
3. **while**(ps < ls && pp < lp) {
4. **int** c = p[pp];
5. **if**(c == '\*') { //处理\*
6. **char** pre = p[pp - 1];
7. ++pp;
8. **if**(sub(ls, lp, ps, pp, s, p)) **return** **true**;
9. **for**(; ps < ls; ++ps) {
10. **if**(pre == '.' || pre == s[ps]) {
11. **if**(sub(ls, lp, ps + 1, pp, s, p)) **return** **true**;
12. }**else** {
13. **return** **false**;
14. }
15. //++ps;
16. }
17. **return** **false**;
18. }
19. **else** **if**(pp + 1 >= lp || p[pp + 1] != '\*') { //处理单个字符
20. **if**(c != '.' && c != s[ps]) {
21. **return** **false**;
22. }**else** {
23. ++pp;
24. ++ps;
25. }
26. }
27. **else** { //后面跟\*的单个字符则跳过
28. ++pp;
29. }
30. }
31. **if**(ps == ls && pp == lp) {
32. **return** **true**;
33. }**else** {
34. **return** **false**;
35. }
36. }

155.二叉树的最小深度

参考第97题，二叉树的最大深度，与求最大深度直接递归不太相同的是，求最小深度需要判断左右子树有空出现的情况，有空则使用另一个子树的高度代替。

1. **int** minDepth(TreeNode \* root) {
2. **if** (root == NULL) **return** 0;
3. **if** (root->left == NULL ) {
4. **return** minDepth(root->right)+1;
5. }
6. **else** **if** (root->right == NULL) {
7. **return** minDepth(root->left)+1;
8. }
9. **int** leftDepth = minDepth(root->left)+1;
10. **int** rightDepth = minDepth(root->right)+1;
11. **return** leftDepth < rightDepth ? (leftDepth) : (rightDepth);
12. }

156.合并区间

水题，首先按照区间的start域进行排序，而后逐个合并就可以（注意区间互斥的情况，灵活使用vector的push\_back()和pop\_back()方法）。

1. **bool** cmp(**const** Interval &a, **const** Interval &b) {
2. **return** a.start < b.start;
3. }
4. sort(intervals.begin(),intervals.end(),cmp);

157.判断字符串是否没有重复字符

水题，使用map来做字典

158.两个字符串是变位词

水题，为两个字符串分别建立字典，然后比较这两个字典是否相同。

159，160．寻找旋转排序数组中的最小值（1，2）

这两个题竟然一样，因为是排过序的，所以直接找数组中第一个不增的位置（如果没有则为0），然后就是最后的结果。

161.旋转图像

把一个矩形向右旋转90度，可以先沿着竖直方向对换，再沿着从左上沿到右下角的对角线对换即可。注意循环的边界条件：

沿竖直方向：

1. **for** (**int** i = 0;i < n/2;i++) {
2. **int** j = n - 1 - i;
3. **for** (**int** k = 0;k < n;k++) {
4. swap(matrix[i][k],matrix[j][k]);
5. }
6. }

沿对角线方向：

1. **for** (**int** i = 0;i < n;i++) {
2. **for** (**int** j = 0;j < i;j++) {
3. swap(matrix[i][j],matrix[j][i]);
4. }
5. }

162.矩阵归零

水题，使用存储pair的vector来标记0的位置，然后逐个把行列变为零即可

163.不同的二叉查找树

使用卡特兰数，结合记忆化搜索可以求得。

卡特兰数的递推公式为：

f(n) = f(n-1)f(0) + f(n-2)f(1) + f(n-3)f(2) + ... + f(1)f(n-2) + f(n-1)f(0)

164.不同的二叉查找树2

针对于上一题使用卡特兰数求数目不同，该题目需要构造出所有的二叉查找树，使用二叉查找树中的最小值和最大值作为参数来进行递归。如果最小值大于最大值，说明二叉查找树不合法，则插入空指针。如果相等，说明二叉查找树只有一个节点，直接生成。

否则的话可以将任意一个节点作为根节点，然后从该节点分开，分别生成左子树和右子树。然后左右子树又分别有不同的可能，对于某个节点使用两重循环递归遍历，分别插入左右子树最后生成解集。

165.合并两个排序链表

两个指针分别指向两个排序链表的表头，然后逐个向后遍历，把当前较小的值放在新的链表中，直到链尾。

166.链表倒数第n个节点

使用快慢指针，快指针比慢指针提前n个节点，快指针到链尾，慢指针为倒数第n个节点。

167.链表求和

解法类似于排序链表，注意进位的问题，以及计算完毕后，要注意根据进位位在链头插入一。

168.吹气球

不太明白“既然可以首先吹爆任意一个气球k，那么也可以选择最后吹爆任意一个气球k”这句话。[nums[i-1] \* nums[k] \* nums[j+1]对应这句话]

使用动态规划，为了方面计算，在给定数组的开头和末尾插入一个“1”，使用dp[i][j]标识吹爆下标i到下标j的气球所获得的最大分数。因而可以得到递推公式：

dp[i][j] = max(dp[i][k-1] + nums[i-1] \* nums[k] \* nums[j+1] + dp[k+1][j])

然后再递推的过程中，使用类似矩阵连乘的方法，第一层循环是不同的序列的长度l（从1到n），第二层循环i从1（第0位置插入了一个1）开始，每次计算不同长度的序列的最大分数，循环到n-l+1结束（防止越界），使用j = i+l-1来记录变量值。第三层循环记录计算每一段的最大值，循环变量k从上一层的i开始，到i+l-1结束，使用递推式计算dp[i][j]的最大值，直到最后计算出结果dp[1][n]。

1. **for** (**int** l = 1;l <= n;l++) {
2. **for** (**int** i = 1;i <= n - l + 1;i++) {
3. **int** j = i + l - 1;
4. **for** (**int** k = i;k <= j;k++) {
5. dp[i][j] = max(dp[i][j], dp[i][k-1] + nums[i-1] \* nums[k] \* nums[j+1]+ dp[k+1][j]);
6. }
7. }
8. }

169.汉诺塔

汉诺塔问题是一个经典的回溯问题，解决该问题的方法一共分为三步：

1. 把n-1个盘子移动到缓冲区
2. 把剩下的最大盘子移动到终点
3. 把缓冲区的盘子移动到终点

注意回溯初始状态的保持：

1. **void** solve(vector<string> &result, **int** n, string from, string buffer, string to) {
2. **if** (n == 1) {
3. result.push\_back("from " + from + " to " + to);
4. **return**;
5. }
6. solve(result,n-1,from,to,buffer);
7. result.push\_back("from " + from + " to " + to);
8. solve(result,n-1,buffer,from,to);
9. }

170.旋转链表

首先根据旋转多少找到链尾第k个元素（注意取余），然后从该点断开将链表重新链接即可。注意还要保留链尾第k个元素的前一个元素，用于接链。

171.乱序字符串

判断两个字符串是否乱序，可以用sort函数，如果结果一致则说明这两个字符串是乱序的。首先遍历一次，将排序的结果作为key出现的次数作为value存在map中。再遍历一次，将排序结果在map中对应的value的值大于1（出现了不止一次）的字符串放入结果vector中返回即可。

172.删除元素

水题，但是要注意在循环过程中使用erase的方法，因为使用erase方法后，索引会指向下一元素，所以在循环中要i--:

1. **int** removeElement(vector<**int**> &A, **int** elem) {
2. **int** result = A.size();
3. **for** (**int** i = 0;i < A.size();i++) {
4. **if** (A[i] == elem) {
5. result--;
6. A.erase(A.begin()+i);
7. i--;
8. }
9. }
10. **return** result;
11. }

173.链表插入排序

水题，注意在空间声明上的位置，不要造成内存溢出。

174.删除链表的倒数第n个节点

首先通过快慢指针找到要删除的节点以及该节点的上一个节点。然后根据该节点是头节点或者是中间节点分类处理，还要注意链表长度为一的情况。

175.翻转二叉树

使用递归，将左右子树互换即可。

176.图中两个点之间的路线

使用队列辅助的宽度遍历搜索即可（使用一个map来记录节点是否被访问过）

177．把排序数组转换为高度最小的二叉搜索树

使用递归，每次使用排序数组的中间元素的值生成当前根节点，然后对递归生成左边的子树和右边的子树。（初始化时，第一个根节点手动生成，然后递归生成左右节点）。

178.图是否是树

一个图中每个节点都有边相连，并且图中不存在环，那么改图为树。首先判断特殊情况，一个单独的点为树，有孤立点存在则不是树（根据节点的度是否为0来判断）。

接下来的主要问题是判断图中是否有环，首先计算各个节点的度，然后删除一个度小于或者等于一的节点，接着把跟该点相关的点的度减一，如此往复，如果能删除全部节点则改图中无环存在，如果找不到入度小于等于一的节点则说明图中有环存在。

179.更新二进制位

这个题目看似简单，使用两个数组进行操作，转化即可。但是由于负数的存在导致复杂度提高，负数的二进制使用补码来表示。int使用一个长度为32的vector来存储，最高位是符号位，0代表正数，1代表负数。

原码转为补码，符号位不变，数据位按位取反，**末尾加一**。

补码转为原码，符号位不变，数据位按位取反，**末尾加一**。

操作过程中需要注意要把填充数的位数补齐，同时不同的码转换时注意加一带来的进位操作。

180.二进制表示

思路类比于179题，整数部分的转化方法于上题思路一致，小数部分的转化方法采用乘二取整法来进行。特别注意小数部分在存储时为了防止越界要使用long long来存，还有就是字符串处理类型题目繁杂的细节。包括对于“0.0”，“1.0”，“0.1”这种类型的处理。

181.将整数A转化为B

思路和上两题一致，首先把A和B都转化为32位的vector（其中存储的是该数的二进制表示），然后直接对比两个vector即可得到最后的结果。

182.删除数字

每次选取删除之后所得结果最小的那个数删去。在比较删除结果时，不需要把string转为int单独比较，只需要两个string逐位比较得到大小关系就可以（使用int会造成数位溢出）。返回删除后的结果之前，一定要判断数字前是否有0，要把多余的0删除。

183.木材加工

给定一个数组和要切的段数，求所得段的最大长度。首先，如果每一段长度为1仍然达不到要求的段数，则返回0。否则首先找到最长段，然后逐次递减，直到保证可以切出k段即达到效果（可以使用二分查找来优化程序）。所得的段数大于等于k则返回即可。

184.最大数

首先把vector<int>通过sstream转化为vector<string>，然后使用自定义compare函数在sort中使用对vector<string>进行排序（注意在类中，sort函数中的compare函数需要是static的）。

sort(vector<string>.begin(), vector<string>.end(), **compare**) [注意比较函数不加括号]

然后把排序后的vector<string>拼接起来即可，其中关键是在写比较函数时，要注意从高位向低位比较，如果高位的数大，该数就大，否则该数就是小的。例如：8 > 23, 23 > 21, 2 >12,99=9,0=0。

185.矩形的之字形遍历

这道题首先要把只有一行或者只有一列的数组作为特殊情况单独处理，然后在结果数组中放入第一行的前两个元素。接着首先如果可以向左下方遍历，则向向下循环放入结果数组。到了界限之后，如果能够往下就往下一格子，否则往右一格接着再向右上方遍历，到了界限之后如果能往右一格就向右，否则向下移动一格。如此往复直到遍历完整个矩形。

186.最多有多少个点在一条直线上

两次循环，在第一重循环中建立一个新的map<double,int>，存储对应一个斜率有多少个点。同时，在计算斜率时要注意分母不能为0。所以在统计时，要把重复的点统计上，把横坐标相同的点单独统计。对于第二重循环的每一统计，如果对应斜率存在，则map[k] ++，否则map[k] = 2。在第二重循环结束前，需要判断map[k]+number\_of\_same\_point的最大值。最后判断垂直于x轴的点的个数。从而获得最多的点数。

187.加油站

从某一点出发，假设加入当前节点的所有汽油，然后看是否能够到达下一个加油站，然后再加上下一个站点的所有油，如此往复，如果有一站到达不了则返回错误。

189.丢失的第一个正整数

可以对数组进行排序，然后找到丢失的正整数。该题注意一些特殊情况，首先是要返回正整数，然后要注意数组中的数字相等的特殊情况。

如果只用一次循环，一次遍历，把当前的数放到该放的位置即可，如1应该放到第0个位置，2应该放到第一个位置。

然后最后使用一次遍历找到存放位置不对的数。

1. **for**(**int** i=0;i<n;)
2. {
3. **if**(A[i]==i+1)
4. i++;
5. **else**
6. {
7. **if**(A[i]>=1&&A[i]<=n&& A[A[i]-1]!=A[i])
8. swap(A[i],A[A[i]-1]);
9. **else**
10. i++;
11. }
12. }

190.下一个排列

可以使用C++的全排列函数，使用next\_permutation(nums.begin(),nums.end())来生成下一个全排列。

生成下一个全排列的算法：

从后向前寻找第一对（i，j），其中i < j，使得nums[i] < nums[j]，然后交换数组中i，j元素的位置，然后从vector.begin()+i+1到vector.end()排序就是下一个排列。

如果数组是逆序则直接返回排序后的数组即可。

191.乘积最大子序列

使用max\_value[i]以及min\_value[i]记录乘上第i个数的最大和最小值，因为有负数存在，所以计算过程中还需要存最小值。比较某一位时，有三种可能，代码如下：

1. max\_value[i] = max(max\_value[i-1]\*nums[i], max(min\_value[i-1]\*nums[i], nums[i]));
2. min\_value[i] = min(min\_value[i-1]\*nums[i], min(max\_value[i-1]\*nums[i], nums[i]));
3. result = max(result,max\_value[i]);

192.通配符匹配

使用动态规划求解，注意dp[i][j]数组下标从1开始，表示s到i，p到j的匹配情况。

首先初始化dp数组，dp[0][0] = 1。dp[i][0] 在（dp[i-1][0] == 1 && s[i-1] == ‘\*’）时为1否则为0，（注意s[i-1]是第i个字符）同理可以dp[0][j]在（dp[0][j-1] == 1 && p[j-1] == ‘\*’）时为1否则为0。

接着进行循环对dp求解，递推式如下所示：

1. **if** (s[i-1] == '?' || p[j-1] == '?') {
2. dp[i][j] = dp[i-1][j-1];
3. }
4. **else** **if** (**if** (s[i-1] == '\*' || p[j-1] == '\*') {
5. dp[i][j] = dp[i-1][j] || dp[i][j-1];
6. }
7. **else** {
8. **if** (s[i-1] == p[j-1]) {
9. dp[i][j] = dp[i-1][j-1];
10. }
11. **else** {
12. dp[i][j] = 0;
13. }
14. }

最后返回dp[n][m]为最后的结果。

196.Missing Number

解题思路与189.丢失的第一个正整数一致，把数组中的数放置在它该出现的位置，最后没在规定位置上的数返回该位置的下标就是丢失的数。

197.排列序号

该题需要求解某一序列对应全排列对应的字典序的序号。无需求出所有的全排列进行比较。对于数组vector<int> num其对应全排列序号的计算方法是：

Sequence\_number = sum(count\*fac(n-i+1))

其中，count表示从位置i+1到数组末尾比num[i]大的数的个数，每一个位数对应着一个权重为（n-i+1）！。

198.排列序号2

该题与上一题的思路基本一致，不同的是对于每一个位置i需要记录一个map<int,int>，对应从位置i到数组末尾重复出现的数字以及对应的个数。计算公式更新如下：

Sequence\_number = sum(count\*fac(n-i+1))/dup

其中dup代表从位置i（包含i）到数组末尾，重复元素的阶乘的积。

200.最长回文子串

水题，首先写一个判断回文的函数，然后二重循环，使用substr函数来截取并判断。注意可以通过维护当前回文子串的最大长度以及简单头尾字符的判断来减少循环的次数。

其中substr（a,b）函数的两个参数，a代表要截取第一个位置，b代表要**截取的长度**。

201.线段树的构造

这个题目采用递归的思想，根据start和end的大小关系来进行不同的操作。首先如果start > end则直接返回空指针。否则根据start和end的值新建一个线段树节点，接着如果start != end，那么计算mid的值，递归生成当前节点的左子树和右子树，否则直接返回。

202.线段树的查询

在上一个题目线段树的基础上，每个节点还维护了当前区间的最大值。该题目仍然采用递归的思路，根据root和查询区间（start, end）的大小关系来做不同的处理。

首先判断非法情况，然后如果查询区间包括了root的区间，那么之间返回root的最大值。否则的话计算出mid值递归查询，这里注意在递归操作的时候要注意比较，放置线段树的查询区间越界。如果节点的区间中点小于查询的左边界，那么只需要查询较大的右子树，如果区间的中的大于查询的右边界，那么只需要查询较小的左子树。

1. **int** query(SegmentTreeNode \* root, **int** start, **int** end) {
2. **int** mid = (root->start + root->end) / 2;
3. **if** (start <= root->start && end >= root->end) {
4. **return** root->max;
5. }
6. **else** **if** (start > mid) {
7. **return** query(root->right, start, end);
8. }
9. **else** **if** (end < mid + 1) {
10. **return** query(root->left, start, end);
11. }
12. **else** {
13. **return** max(query(root->left, start, mid), query(root->right, mid + 1, end));
14. }
15. }

203.线段树的修改

给出要修改的点和要修改的值，返回修改后的结果。这个题目对线段树的修改包含两部分，首先是要把对应的值改了。递归操作，如果start == end && start == index则修改当前节点的值，否则根据index和mid的大小关系决定递归修改哪一棵子树。在修改之后，需要维护root的值（每次修改之后，更新父节点的值）:

1. root->max = max(root->left->max, root->right->max);

204.单例（static的特点和用法，函数，变量）

首先声明一个静态的指向该类指针（使用指针是保持内容的持久化），如果该指针非空，则新建一个对象使其初始化，否则返回该指针。

1. **static** Solution\* getInstance() {
2. **static** Solution\* one;
3. **if** (one == NULL) {
4. one = **new** Solution();
5. }
6. **return** one;
7. }

205.区间最小值（不太明白）

这个题目依托于线段树的构造，查询，在构造时，与之前的构造题目不同，此处需要根据一个存储节点的值的数组来构造线段树。在构造时，根据start和end的大小关系来进行不同的操作。如果start > end则返回空指针。否则根据start和end建立一个线段树节点，然后如果start < end，那么递归建立该节点的左右子树，并且当前节点的最小值等于左右子节点中的最小值。如果start == end那么用数组中下标为start的元素来初始化这个节点的最小值（到达树的叶子节点）。

查询的过程我不太明白，与之前的查询函数不太一样。其根据mid和start以及end的对应关系来查询，代码如下所示：（start > mid查较大的一边）

1. **int** query(SegmentTreeNode \* root, **int** start, **int** end) {
2. **int** mid = (root->start + root->end) / 2;
3. **if** (start <= root->start && end >= root->end) {
4. **return** root->max;
5. }
6. **else** **if** (mid < start) {
7. **return** query(root->right, start, end);
8. }
9. **else** **if** (mid + 1 > end) {
10. **return** query(root->left, start, end);
11. }
12. **else** {
13. **return** min(query(root->left, start, mid), query(root->right, mid + 1, end));
14. }
15. }

206.区间求和1

该题目和上述几题的思路一致，都是先建立线段树，再查询，不同的是，这个题目在建立线段树的时候要累计子节点的和。查询函数还是一致的。

207.区间求和2

该题目和上述题目的思路一致，都是先建立求和线段树，再查询。查询函数还是一致的。这个题目唯一不同的的是增加了一个修改函数，修改函数和原来的也是一致的，但是需要特别注意的是，在判断mid和index的大小关系的时候。

当index >= mid+1 搜索右子树，index <= mid时搜索左子树。

208.赋值运算符重载（考察深拷贝和运算符重载）

完成A = B = C的操作，首先要注意深拷贝和浅拷贝的区别，深拷贝是拷贝的结果变化，被拷贝的那个内容也会随之而变，而浅拷贝则不会变化。另外注意只有C++才具有运算符重载。首先判断二者是否相等，如果不想等的话首先把this中的数据delete然后为其开辟新的空间存储新的内容。

1. Solution operator=(**const** Solution &object) {
2. **if** (**this** != &object) {
3. **delete**[] **this**->m\_pData;
4. **if** (object.m\_pData != NULL) {
5. **this**->m\_pData = **new** **char**[strlen(object.m\_pData) + 1];
6. strcpy(**this**->m\_pData, object.m\_pData);
7. }
8. }
9. **return** \***this**;
10. }

209.第一次只出现一次的字符

两次遍历，第一次使用一个map记录所有的字符以及出现的次数，第二次遍历这个map找出第一个出现次数为1的字符。

211.字符串置换

这个题目使用两个map记录出现的所有字符以及次数，然后对比两个map，如果这两个map相等那么这个这两个字符串的组成是一致的，可以通过置换得到。

212.空格替换

这个题目有点意思，首先题解函数的参数传递是”char srting[]”给出了一个不定长的char数组，首先通过一次遍历找出空格的个数，然后计算出新字符串的长度。接着使用一次循环（注意：一定是从原字符串的尾端开始循环，因为这样方便字符移动，相当于提前把前面的空格插入新字符的位置提前空出来），如此一来，如果遇到空格，就插入“02%”同时注意新数组中的下标移动，如果是非空格就直接将字符复制。

213.字符串压缩

一次遍历，记录当前的字符以及出现的次数，如果下一个字符和当前字符不一致则把当前字符和出现的次数（注意转化为string）放入结果字符串。注意最后一个字符串要单独处理，因为没有不同的字符来触发。然后判断，如果压缩后新字符比原来短则返回新字符，否则返回未压缩的结果。

221.链表求和2

参考链表求和1的思路，首先使用两个指针进行操作将链表反转，然后把反转的链表求和（注意进位，特别是最后一位有没有1）最后把求和的结果再次反转成正序即可。

223.回文链表

这个题目有点像上一个题目，有两种解法，第一种是一次遍历，把链表存在一个数组里，然后判断是否回文。第二种是拷贝一份链表（深拷贝），然后计算该链表的反转链表，看二者是否相等来判断是否回文。

227.用栈模拟汉诺塔问题

这个题目考察类中的相关操作，然后就是明确汉诺塔这个经典回溯问题的解法。

1. 把第一个柱子的前n-1个盘子移到缓冲区（此时将目的柱子作为缓冲）；
2. 然后把第n个盘子移到目的柱子；
3. 接着把缓冲区上的n-1个盘子移动到目的柱子上（第一个柱子作为缓冲区）。

245.子树

使用递归来进行判断一个子树是否是另一个树的子树，首先写一个判断函数，从根节点开始判断两个树是否是相同的。

接着进行查找操作，首先如果两棵树均为空则返回true，否则如果较大的树为空则返回false。然后如果查到两个节点的值相等，那么调用检查函数，如果检查成功则返回true。否则的，在较大的那棵树的左，右子树分别递归查找。最后均查找不成功则返回false。

247.线段树查询2。

线段树中存在控制，只要对root和start以及end进行正确性检查，跟之前的线段树查询方法一致，当然，网上还有一个考虑边界的较为复杂的解法。给出链接：

<https://blog.csdn.net/sunday0904/article/details/69371940> 其主要思想就是边界的划分，因为不是每个位置都有节点，所以既要判断空，也要重新划分边界，所以在递归的时候，start取两者最大，end取两者最小。

248.统计比给定整数小的数的个数

采用循环加二分法来查找，要注意此处二分法的特殊情况，是在排序数组中找第一个不大于给定值数（返回的是上边界）。代码如下：

1. **int** find\_num(vector<**int**> &A, **int** target) {
2. **int** start = 0;
3. **int** end = A.size() - 1;
4. **while** (start < end) {
5. **int** mid = (start + end)/2;
6. **if** (target > A[mid]) {
7. start = mid + 1;
8. }
9. **else** {
10. end = mid;
11. }
12. }
13. **return** start;
14. }

249.统计前面比自己小的数的个数

采用循环结合二分法会超时。构造线段树来求解，节点中包含元素出现的次数，所有count > 0的叶子节点表示元素存在。依次更新结点计数，所以后面的元素不影响前面的元素计数。

需要写出build, query, modify三个函数。注意query函数需要用边界限制法来求（不连续的子树需要用这个方法吗？不太明白），在modify函数中，因为右子树不一定存在，所以在最后回溯时需要判断，右子树不为空才能加上其对应的值。

1. **int** query(SegmentTreeNode \* root, **int** start, **int** end) {
2. **if**(root == NULL) {**return** 0;}
3. **if**(start > end || root->start > end || root->end < start) { **return** 0; }
4. **if**(start <= root->start && end >= root->end) {**return** root->count; }
5. **int** mid = (root->start + root->end) >> 1;
6. **int** leftCount = query(root->left, start, min(mid, end));
7. **int** rightCount = query(root->right, max(mid, start), end);
8. **return** leftCount + rightCount;
9. }

360.滑动窗口的中位数

首先使用一个临时数组记录当前滑动窗口中的数，保证该数组中的数有序，每次使用二分法（注意边界条件是start <= end）来查找出滑出去的数，然后把该数删去，接着把新加入的数找到插入的位置，保证滑动窗口还是有序。然后输出每个时段的中位数（位置是（k-1）/2），其中k为窗口的大小。

362.滑动窗口的最大值

使用C++ STL中的双端队列（deque）来做，队列从左至右存储当前维护的值自大到小顺序的下标。每次新插入一个值时，把队列尾端小于该数的值弹出，然后插入新的值（维持队首最大）。

1. **while**(!q.empty()&&nums[q.back()] <= nums[i])
2. q.pop\_back();
3. q.push\_back(i);

然后在判断的时候，还要维护一个量记录当前滑动窗口的左边范围，在队首元素的下标大于或者等于左边范围时，才表明队首的数在当前窗口内，才能够使用。

363.接雨水

这个题目要能从题目中抽象出解决的办法，形象的来说，两个较高的值中间部分可以用来接雨水，所以解决办法如下：

首先一次遍历，找到所有值中的最高点，记录最高点的值和最大值出现的位置，然后从左右两端分别向中间遍历，首先用端点的值来作为局部最高点，然后如果当前点比局部最高点低，那么其比局部最高点低的部分可以用来接雨水，反正用当前的点替换局部最高点。左边同理（可以考虑为都使用中间最高点作为接雨水的一端，然后局部求解）。

364.接雨水2

首先，思考之前的一维那个题目，要想知道其中的一个位置的盛水量，我们只需求出这个位置的左右两边最高的两个柱子中，小的那个柱子的高度，就可以推算出（这个就是木桶原理哈）。现在看看这个题目，要想知道每个位置的盛水量，那么这个位置的盛水量一定和这个位置的四周的盛水量有关，而且里面的位置的盛水量一定和外围的盛水量有关。按照这个思路，可以从边界一层一层向里推导，相当于广度优先搜索，最外层边界先入队，然后出队，每次出队一个位置，就以这个位置为中心，更新这个位置四周的位置盛水量，并判断入队。以此类推直到队列为空，就可以获得每一个位置的盛水高度，加之，之前的位置高度，就可以得出总的盛水量了。如下图，类似于一层一层向里推算，出的队，红色方块，更新四周位置的盛水量（边界不考虑了）。

这道题的思路我不太明白：

首先建立对应高度的二维数组，存储每个位置选择的高度。由外向内逐层遍历，初始化时把初始矩阵的四条边的值放入选择的高度数组，并且放入一个队列，其余的部分初始化为INT\_MAX。

如果队列不为空，取出队头元素，查看该元素上下左右四个位置，对于每一个位置，比较其和中心位置选择数组中的高度，保留较大的高度，如果该限制值比当前试探位置所存储的选择值大，那么更新选择值数组，并将其放入队列。

最后遍历整个数组，选择高度和初始高度的差值之和就是可以接的总量。

365.二进制中有多少个1

这个题目可以采用一个巧妙的方法：

n &= (n – 1)能清除最右边的1，因为从二进制的角度讲，n相当于在n - 1的最低位加上1。举个例子，8（1000）= 7（0111）+ 1（0001），所以8 & 7 = （1000）&（0111）= 0（0000），清除了8最右边的1（其实就是最高位的1，因为8的二进制中只有一个1）。再比如7（0111）= 6（0110）+ 1（0001），所以7 & 6 = （0111）&（0110）= 6（0110），清除了7的二进制表示中最右边的1（也就是最低位的1）。

1. **int** countOnes(**int** num) {
2. **int** result = 0;
3. **while** (num) {
4. num = num & (num - 1);
5. result ++;
6. }
7. **return** result;
8. }

366.斐波那契数列

水题，但是单纯的使用递归会超时，可以使用数组暂存中间结果以减少递归过程中的重复计算。

367.表达树构造（不太明白）

首先把中缀表达式换成前缀或者后缀表达式，然后用计算前缀或者后缀表达式的思路继续构造：碰到数字，压入堆栈（有一个函数专门计算优先级）。碰到运算符：新建一个节点，左右分别指向出堆栈的两个节点，再将节点压入堆栈。最后返回堆栈顶部的元素即为表达树的根节点。

368.表达式求值

首先表达式按照第367题的思路转化为表达树，然后采用后序遍历的方法逐步计算（注意每个节点要用当前计算的值更新自己所储存的结果）。还要注意处理特殊情况：如果一个表达式中没有数字，则直接返回0。

370.将表达式转化为逆波兰表达式

首先表达式按照第367题的思路转化为表达树，然后采用后序遍历的方法遍历即可得到结果。

371.用递归打印数字

可以使用一个数组来记录要打印的数字，用数组的长度限制递归的条件（注意参数传递要使用引用，保证数组的变化被记录下来），最后输出结果。

372. Delete Node in a Linked List

曾经折戟的一道题，给出一个节点，将其从链表中删去，只需要把后边的复制到这个节点上来（整体往前移动一个节点），然后删除链尾的节点即可。

373.奇偶分割数组

这道题要求把奇数放在前面，偶数放在后边，并且在原地完成。两重循环，如果找到偶数那么从这个数下一个位置开始寻找奇数，然后不断换位置，把奇数都换到这个数的前面。循环完成后则完成数组的分割。

374.螺旋矩阵

用一个整数记录当前遍历的方向，先向右边，然后向下，接着向左，最后向上，同时用一个数组来记录位置是否被访问过。注意，每次访问到不能访问的位置时，要将下一次访问的节点的位置计算好。

375.克隆二叉树

使用递归的方式进行树的复制类似于二叉树的先序遍历。

376.二叉树的路径和

使用深度遍历的方法来寻找路径和，其中如果满足条件就放入解集，否则进行深度递归，最后要保证回到递归之前的状态。

378.将二叉查找树转换成双链表

二叉查找树中序遍历就变成了有序数组，然后根据数组生成双链表即可。

379.将数组重新排序以构造最小值

将数组中的数构造出一个最小数，关键点在于排序算法的compare函数。对于比较的两个数，逐位进行比较，如果有较小的位，则比较结束，否则指针向后移动，直到比较到两个数的最低位。把比较值较大的放在后边，组合出的就是最小的数。（比较部分的代码如下所示）

1. **bool** compare (string a, string b) {
2. **if** (a == b) {
3. **return** **false**;
4. }
5. **int** i = 0;
6. **int** j = 0;
7. **while** (i < a.size() && j < b.size()) {
8. **if** (a[i] > b[j]) {
9. **return** **true**;
10. }
11. **else** **if** (a[i] < b[j]) {
12. **return** **false**;
13. }
14. **else** {
15. **if** (i + 1 == a.size() && j + 1 == b.size()) {
16. **return** **false**;
17. }
18. **if** (i + 1 < a.size()) {
19. i ++;
20. }
21. **if** (j + 1 < b.size()) {
22. j ++;
23. }
24. }
25. }
26. }

380.两个链表的交叉 （这个问题只考虑尾对齐）

首先计算得到两个链表的长度，然后把较长的链表向尾部移动，直到两个链表的长度一致，然后逐个节点进行比较直到找到两个链表交叉的第一个节点。

381.螺旋矩阵2

明确矩阵生成的四个方向，然后逐个方向进行求解即可，注意每个方向变化的边界情况。

382.三角形计数

这个题目比较巧妙，首先将所有的边按照升序进行排列，然后进行两次循环，在确定了首尾之后，中间使用二分法找到比以首尾作为三角形两边的，比这两条边的差大的位置，然后可以记录从这个位置到尾部的长度就是当前可以组成的三角形个数。

383.装最多水的容器

从两边向中间逼近，选择过程中最大的点。在逐步向中间移动的过程中，如果边较小则向中间移动，保留较长的边。直至找到最终的最大值。

384.最长无重复字符的子串

使用字典<字符，位置>来存储每个字符的位置，用一个length来记录当前字符串的长度。i 是当前字符位置，i - length是当前字符串起始位置。出现重复字符时，先判断字符上一次出现位置是否在当前字符串内，如果不在，则Map中更新位置并且length++；如果在，则length截断，从 上一次出现+1 的位置开始计算。

1. **for** (**int** i = 0;i < s.length();i++) {
2. map<**char**, **int**>::iterator it = record.find(s[i]);
3. **if** (it == record.end()) {
4. record[s[i]] = i;
5. length ++;
6. }
7. **else** {
8. **if** (record[s[i]] + length < i) {
9. length ++;
10. }
11. **else** {
12. max\_length = max(max\_length, length);
13. length = i - record[s[i]];
14. }
15. record[s[i]] = i;
16. }
17. }
18. max\_length = max(max\_length, length);

386.最多有k个不同字符的最长字符串

使用map作为辅助存储来做，使用map<char, int>来记录某个字符出现的最后的位置。使用一个变量left记录最多有k个不同字符的字符子串的起始位置。当map中多于k个不同的字符时，如果left就是当前s[left]目前出现的最大位置，则直接从map中删除s[left]（过滤了那些只出现一次的元素），否则将left向后移动。

1. **for** (**int** i = 0; i < s.size(); ++i)
2. {
3. m[s[i]] = i;
4. **while** (m.size()>k)
5. {
6. **if** (m[s[left]] == left)
7. m.erase(s[left]);
8. ++left;
9. }
10. res = max(res, i - left + 1);
11. }

387.最小差

找到两个数组中两个数的最小差值，首先将两个数组进行排序，然后把两个排序数组合并为一个有序的数组。合并的同时要使用一个辅助的数组来记录在合并之前这个数是来自哪个数组，之后依次求合并数组中两数之差（注意求得时候要求两个数来自不同的数组），然后得到最小差值。

388.第k个排列

求出按照字典序的第k个排列，需要知道对于某一个字符串的下一个按照字典序的排列。只需按照如下步骤求解。

首先从右到左找到第一个比右边的数小的数，然后从这个数向后查，找到比这个数大的最小值。然后这两个数交换位置。最后把第一次找到的位置后边的字符倒序即可。

例如839647521是数字1～9的一个排列。从它生成下一个排列的步骤如下：

1.自右至左找出排列中第一个比右边数字小的数字4

2.在该数字后的数字中找出比4大的数中最小的一个5

3.将5与4交换 839657421

4.将7421倒转 839651247

所以839647521的下一个排列是839651247。

389.判断数独是否合法

使用三个循环分别判断每行，每列，每一个小的三乘三的格子中是否有重复的数字即可。

391.数飞机

使用数组dp[i]记录到第i个时刻，天上的飞机数目。主要是记录边界情况，使用另一个数组record[i] 记录飞机的i时刻的起降情况，如果此时有飞机降落，记为1，在下次累加时，对于record[i]为1的点不累加飞机数目，然后将其还原成普通值（一架i时刻降落的飞机只影响一次计数）。

392.打劫房屋

使用两个变量，分别记录到第i间屋子，是否打劫的最大值。然后反复迭代即可。

value\_yes = value\_no + values[i]

value\_no = min (value\_yes, value\_no)

result = max (value\_yes,value\_no)

394.硬币排成线

这个问题使用数学推理的方式来计算。当B开始拿有三个硬币时，可以保证A最后拿完，所以只要硬币的数目为3n+1或者3n+2既可以保证先手必胜。

395.硬币排成线2

计算拿硬币的最大价值，使用动态规划。dp[i]表示从i到len所拿的硬币的最大值。

dp[len] = 0

当剩下一个或者两个硬币时，一定都拿价值多

dp[len-1] = values[n-1]

dp[len-2] = values[n-1] + values[n-2]

当剩下三个硬币时，一定拿前两个价值多（对手要不肯定拿后两个）

dp[len-3] = values[n-2] + values[n-3]

其余情况，可以选择拿一个或者两个，

如果拿一个，对手一定选择拿后边较大的：

dp[i] = values[i] + min(dp[i+2],dp[i+3])

如果拿两个对手同样选择拿较大的：

dp[i] = values[i] + values[i+1] + min(dp[i+3],dp[i+4])

综上：

dp[i] = max (values[i] + min(dp[i+2],dp[i+3])，values[i] + values[i+1] + min(dp[i+3],dp[i+4]))

最后推出dp[0]即为最后的结果

400.最大间距

使用O（n）的算法进行排序，使用桶排序，对于n个元素，安排n+1个桶，首先计算出数组中元素的最大和最小值，然后使用n/(max-min)算出步长（特别注意处理分母为0的情况），然后使用（num[i]-min）\* interval来计算每个元素所在的桶的位置，只需保存每个桶中的最小值和最大值，因为最大间距一定出现在桶间，然后逐个桶进行比较即可（bucket[i+1].low – bucket[i].high）。

401.排序矩阵中的从小到大第k个数

与剑指offer中从右下角找在不在这个矩阵中的数不同，这个题找第k大。维护一个列数组，存储当前寻找的每一行所对应的列数。首先找到第一列中最小的数，然后有最小数的那一行的列数加一，接着在包括那一列以及前一次寻找的剩下的列中找第二大的数，并把第二大的数所在的行的列数加一。直到找到第k大的数字。

402.连续子数组求和

与求子数组最大的和的方法一样，不同的时需要记录起点和终点的位置，每次子数组的和小于零时更新起点的位置。终点的位置随着指针的移动逐步更新。

405.和为零的子矩阵

这个题目比较巧妙，首先计算出一个和矩阵，sum[i][j]表示从matrix[0][0]到matrix[i][j]的和（第一行和第一列按照特殊情况处理，其余的按照下式递推），然后点（i，j）到点（p，q）所对应的矩阵的为

sum[p][q] – sum[i-1][q] – sum[p][j-1] + sum[i-1][j-1]

注意判断边界情况，如果越界则那一项的值为0。

406.和大于S的最小子数组

使用两根指针从左往右一次遍历，走在前面的指针算出sum>=s后，第二根指针逐渐逼近，找到最小的子数组长度，直到前面的指针走到数组尾部，可以最终计算出最小子数组的长度。

407.加一

水题，直接使用加操作计算就可以，注意进位操作，同时插入的过程使用vector的insert函数，每次都放在前面。

408.二进制求和

注意求和问题，这个题目采用三个部分来做，首先，把两个数相同长度的位数相加，注意保留进位，然后把较长的那个数的剩下部分补全到前面（主要是进位的加法操作），最后计算完毕之后要根据进位是否在最前面加一。

411.格雷编码

这个问题采用递归的方式来解决，对于一个n位的格雷编码，其是通过n-1位的格雷编码分别在前面加上0，1组合而成，所以在生成时，首先生成位数为（n-1）的格雷码，然后**倒着**遍历（保持顺序），每一位加上2^(n-1)即可。

412.分糖果 （不太明白）

这个问题的思路是比较明确的，一次遍历，同时声明几个变量来记录当前的情况，要记录上一个节点的分配的值cur\_valut，进入递减序列之前分配的值pre\_value，以及当前的递减序列的长度length。

首先把第一个节点的值初始化为一，然后从第二个点向后计算，如果

ratings[i] < ratings[i-1]

则递减序列长度length++，并且如果进入递减序列之前的pre\_value小于或者等于length，那么结果需要进行修正加一，然后结果需要加上当前递减序列的长度。同时把上一个节点分配的值置为1。

否则当前节点的值等于上一个节点的值加一，（后边节点优先级小或者和前面相等从一开始），并更新cur\_value，length=0，pre\_value = cur\_value;

413.反转整数

使用字符流实现int和string的相互转化，然后把整数变成字符串处理即可。注意使用long long来保存，注意处理整数溢出的情况。

414.两个整数相除

使用位操作来做，使用被除数减去除数，然后除数左移一位，接着减，注意保存中间的值，直到被除数小于除数则完成出发。

1. **while** (d1 >= d2) {
2. **long** **long** temp = d2;
3. **long** **long** cnt = 1;
4. **while** (d1 >= temp) {
5. d1 -= temp;
6. result += cnt;
7. cnt = cnt << 1;
8. temp = temp << 1;
9. }
10. }

415.有效回文串

使用两根指针分别指向字符串的首和尾，然后找的合法的字符进行比较，判断是否回文。

416.有效数字

首先使用两根指针，去除首尾多余的空格，正负号，以及小数点，然后判断中间数字部分是否合法，注意要判断整个字符串是否含有数字。

418.整数转罗马数字

建立一个4\*10的字符数组，分别存储对应千，百，十，个位的罗马数字，然后整数求出每一个位数对应的罗马数字组合起来即可。

419.罗马数字转整数

首先建立一个罗马数字对应十进制数字的map，同时使用一个临时变量记录上一位罗马数字的值，从后向前依次遍历罗马数，如果当前的值大于或者等于上一个值，则加当前值，否则减去当前值，遍历结束即求得罗马数字的值。

420.报数 （题目含义不太明白）

处理末尾前的数字从第一个位置开始判断，和后边的数字一样count++，否则就加入字符串temp，直到末尾数字的判断，最后返回res。

421.简化路径 （使用栈来处理）

使用一个字符串cur来保存之前的字符串，如果当前遍历的位置对应的字符串为”/”，则需要验证之前的字符串cur是否是”..”，如果是，则需要把栈清空，如果之前的字符串cur是”.”则无需任何操作，否则如果之前字符串cur不为空，进行压栈操作，然后将当前字符串cur置为空。

如果当前遍历的位置对应的字符串不为”/”，则cur+=path[i]，然后继续进行循环。

最后进行出栈（逆序）操作即可，注意在每个栈元素之间需要加”/”，如果栈为空，则直接返回”/”。

423.有效的括号序列

使用栈来模拟可以轻松解决

424.逆波兰表达式求值

求解思路和上一题一致

425.电话号码的字母组合

首先建立每个按键对于字母组合的字典，然后使用深度遍历求解即可（注意回溯）。

426.恢复ip地址

直接枚举三个点的位置即可，注意判断三个点出现的合理情况，首先每一个分割的子阈值在0到255之间，其次，要保证剩下的点可以合理分完余下的字符串，每个位置至多有3个选择，所以总共四重循环也只有81次，因而可以直接使用暴搜解决。

427.生成括号

使用递归树来做，当左括号大于右括号的时候，增加左右括号均可，如果左括号小于右括号则匹配失败，因而直接采用类似树遍历的递归方式求解即可。

428.x的n次幂

不用逐次乘，而是采用使用之前结果的平方继续运算（指数变为之前的两倍）。注意边界值的处理，当指数为INT\_MIN时，要注意取绝对值的时候注意溢出的情况。

430.攀爬字符串

使用递归的思想，如果字符串的长度为一或者二，则按照特殊情况进行处理，否则将字符串从任意位置分开，判断两个字符串的左右子串是否相等，同时还要兼顾两个字符串交换左右部分的情况下是否相等，进行递归处理。

1. string left1 = s1.substr(0,i);
2. string left2 = s2.substr(0,i);
3. string right1 = s1.substr(i,n);
4. string right2 = s2.substr(i,n);
5. string x = s2.substr(0,n - i);
6. string y = s2.substr(n - i,n);
7. **if** ((isScramble(left1,left2))&&(isScramble(right1,right2)) || (isScramble(x,right1))&&(isScramble(y,left1))) {
8. **return** **true**;

433.岛屿的个数

使用深度遍历的思想，如果找到一个位置为1，则把这个位置置为0，然后向上下左右四个方向进行深度遍历。

434.岛屿的个数2（不太明白）

使用一个一维数组来记录每个位置所位于的岛屿，当nums[pos] = pos时，这个点为当前岛屿的最高点。遍历每个点，如果这个点与上下左右的岛屿不属于同一个岛屿，需要将岛屿的数目减1，更新nums中的值。

436.最大正方形

每次找到从一个位置出发的横竖两条边长的最小值，然后判断从这个点开始，以这个边长能否组成一个正方形。

437.书籍复印

使用二分查找的方式，边界为0到所有书籍的页数之和。找到最小边界，使用一个函数来判断当前每个人印制n页，能否能够使用k个人印刷完成。

判断方法是使用一个中间值sum进行记录，如果sum+pages[i]<n则说明当前这个人还可以继续印刷，否则人数要加一，同时把sum置为pages[i]（注意判断单个的pages[i]不能大于count），最后因为第一个人没有加，所以要在sum不为0的情况下将人数加一，完成人数小于k则说明合法，知道找到最小值。

439.线段树构造2

采用自底向上的方式进行构造，当start < end时，使用递归的方式构造，start == end时，更新当前节点的最大值为A[start]。

442.实现Trie（前缀树）

首先明确前缀树的定于（根节点没有值），然后要建立前缀树节点，其中包括节点本身的值以及其后边的一个长度为26的前缀树节点数组，还有一个bool值标识当前节点是否为字母结尾。

插入时，每次插入一个字母，如果当前字母存在，则继续向后插入，否则新生成一个节点再继续。

search和startwith的逻辑基本一致，都是根据查询字符逐个向后查找，但要注意search最后要根据是否是字母结尾来判断是否找到，而startwith直接判断就可以，不一定是字母的末尾。

445.余弦相似度

水题，注意判断分母不为0的情况即可。

448.二叉查找树的中序后继

这个题目分两种情况进行查询，首先如果查询节点存在右子树，则判断其右子树是否有左子树，如果有，则其后继是左子树的最左的一个节点，如果没有左子树，其后继就是右子树。

如果查询的节点没有右子树，就在树中找到第一个比当前节点大的节点就是其中序后继。

450.K组翻转链表

首先明确翻转链表的操作方式：从头开始，两个指针向后操作，注意需要记录后一个指针的下一个节点，防止断链。然后使用两个指针分别保存每K个翻转链的前一个指针和后一个指针，保证翻转之后链表不断，然后注意细节操作即可。还有注意第一次翻转后，要更新head的位置。

451.两两交换链表中的节点

可以理解为K组翻转链表的一种情况，还有一个比较直接的方法，直接取奇数个数节点与其后一个节点（不为空的话）的值交换即可。

452.删除链表中的元素

删除过程中，分了两种情况处理，头节点和中间节点，特别注意因为有可能出现连续相等的值，因为在删除时要使用while进行判断，保证重复的值都被删去。

453.将二叉树拆成链表（不太明白）

这个问题可以使用递归的方式解决，首先根据根节点的左右指针分别生成，如果左指针生成的链不为空，则左链的下一个节点为根节点的右子树，同时根节点的右子树保存成左子树并把左子树置空。

然后左链或者右链不为空则返回，否则返回根节点。

1. TreeNode \* ConvertToLink(TreeNode \* root) {
2. **if** (root == NULL) {
3. **return** root;
4. }
5. TreeNode \* leftLinkTail = ConvertToLink(root->left);
6. TreeNode \* rightLinkTail = ConvertToLink(root->right);
7. **if** (leftLinkTail != NULL) {
8. leftLinkTail->right = root->right;
9. root->right = root->left;
10. }
12. root->left = NULL;
13. **if** (rightLinkTail != NULL) {
14. **return** rightLinkTail;
15. }
16. **else** **if** (leftLinkTail != NULL) {
17. **return** leftLinkTail;
18. }
19. **else**
20. **return** root;
21. }

454．矩阵面积

水题

457.经典二分查找问题

水题，直接按照最基本的二分查找的范式来写。

460.在排序数组中找最接近的K个数

首先使用二分法或者暴力在排序数组中找到最接近目标的那个数，然后使用两个指针分别向左右查找，知道找到最接近的K个数字

463.整数排序

水题，注意冒泡排序的写法。

464.整数排序2

注意快速排序的写法！

466.链表节点计数

水中水。

469.等价二叉树

从根节点的值开始比较，然后递归向下比较。注意判断子树为空的情况（都为空，一个空另一个不空）。

471.最高频的K个单词

首先使用map存储单词和出现次数的映射，然后使用结构体数组存储map中的各个pair之后自定义compare函数使用sort就可以找到最高频的K个单词。

473.单词的添加与查找

首先想到使用前缀树来做，但是在search部分，没有很好的解决“.”的匹配问题。

使用vector和map做，用map保持没有重复节点的插入，可以刚好擦边通过。

474.最近公共祖先2

类比第88题，那道题是节点没有指向父节点的指针，所以可以使用递归来进行查找（分别找根节点左右节点的公共祖先，如果不为空，则说明公共祖先为当前节点，如果左为空则说明公共祖先为右边节点，否则为左边节点）。

这个题目因为有指向父节点的指针，所以可以从两个节点分别产生两条到根节点的路径，然后可以把这个问题转化为两条链表的第一个公共节点问题。可以让长的链表先走k步，然后和短链表一同遍历，直到找到第一个节点。

476.石子归并

刚开始使用贪心的算法结果不能得到左右解，其实这个问题和矩阵连乘是一类问题，都是区间动态规划。状态转移方程为：

dp[i][j] = dp[i][k] + dp[k+1][j] + weight(i,j) i <= k < j

dp[i][j]表示从i到j的最小合并代价。weight(i,j)表示从i到j的重量之和。

初始化时dp[i][i] = 0;因为一堆合并没有代价。

然后通过区间动态规划的范式来做，三重循环，第一重是区间长度从2到n。第二重是i从0到i+l-1<n。对应的，令j=i+l-1作为区间末尾。对于每一对i，j，有i <= k < j作为第三重循环，最后递推出最后结果dp[0][n-1]。

1. **for** (**int** l = 2;l <= n;l++) {
2. **for** (**int** i = 0;i+l-1 < n;i++) {
3. **int** j = i+l-1;
4. dp[i][j] = INT\_MAX;
5. **int** w = weight(A,i,j);
6. **for** (**int** k = i;k < j;k++) {
7. dp[i][j] = min(dp[i][j], dp[i][k] + dp[k+1][j] + w);
8. }
9. }
10. }

477.被围绕的区域

这个题目和岛屿的个数那个题目思路一致，采用深度遍历的思想来做。由于在边缘的区域不能计算在内，所以这个问题的解法分两步：

* 1. 首先遍历矩阵四个边，把边上的“O”替换成“\*”，并且从这些点出发开始进行深度遍历，把相连的“O”替换成“\*”。
  2. 遍历矩阵，先把“O”换成“X”，再把“\*”换成“O”。

479.数组第二大数

水题，使用两个变量来保存最大值和次大值，一次遍历，如果当前值大于最大值，更新最大值，并把原来的最大值赋给次大值，如果当前值大于次大值，则直接更新次大值。

480.二叉树的所有路径

采用递归的思想来做，首先，把字符串初始化为根节点的值。如果，当前节点左右子树均为空，则把当前路径放入结果数组。否则分别初始化两个空串用来记录左右路径。并逐步向下遍历，直到找到所有路径。

1. **void** find\_paths(vector<string> &result, string &path, TreeNode \*root) {
2. **if** (root->left == NULL && root->right == NULL) {
3. result.push\_back(path);
4. }
5. **else** {
6. string left = "";
7. string right = "";
8. **if** (root->left != NULL) {
9. left = path + "->" + IntToString(root->left->val);
10. find\_paths(result,left,root->left);
11. }
12. **if** (root->right != NULL) {
13. right = path + "->" + IntToString(root->right->val);
14. find\_paths(result,right,root->right);
15. }
16. }
17. }

484.交换数组两个元素

水题，直接交换即可，注意要判断是否越界。

486.合并K个排序数组

这个题目有两个解法，第一是直接把所有的数组放入到一个大的数组中，然后把大数组直接进行排序即可。

第二种方法是采用堆来做，首先使用每个数组的第一个元素维护一个大小为K的大根堆，然后取出堆顶最小的元素作为排序后的第一个元素。然后把当前最小值所在数组的下一个元素放入堆中，如此往复直到所有数字都被排序。

488.快乐数

这个题目直接使用递归求解即可，使用一个值来比较递归前后值的大小，防止无限循环的出现。同时使用一个常量来保存递归的次数，防止堆栈溢出。

491.回文数

水题，为了防止回文之后数字溢出，我们使用一个数组来存储数字的每一位的值，然后进行计算。使用如下结构可以得到数字的每一位。

1. **while**(num) {
2. record.push\_back(num%10);
3. num = num/10;
4. }

495.实现栈

水题，使用一个队列就可以实现。

496.玩具工厂

注意子类对于基类纯虚函数的实现，要注意const的位置。const放在函数前是用来修饰函数的返回值，说明函数的返回值是一个const常量。其放在函数后是说明这个函数是常成员函数，该函数只能读取成员变量，不能修改。

499.单词计数（Map Reduce版本）[不太明白]

题目本省很简单，使用map把输入的链表的value按照键值对的方式output。

Reducer记录输入链表中的值的出现次数，然后使用output对应的键值对。

501.迷你推特

一个编程练习，使用一个vector来存储发的推特，然后使用一个map来记录每个用户的好友关系，剩下的实现工作都比较直观，要注意两点。首先是在返回推特时要注意保持顺序。第二点是在用迭代器删除时，要按照如下格式，因为erase后，索引会指向下一个。

1. **for** (vector<**int**>::iterator iter = friend\_list[from\_user\_id].begin();iter != friend\_list[from\_user\_id].end();) {
2. **if** (\*iter == to\_user\_id) {
3. iter = friend\_list[from\_user\_id].erase(iter);
4. }
5. **else** {
6. iter ++;
7. }
8. }

504.倒排索引（Map Reduce版本）

与499题一致，采用map reduce的编程格式来做即可，基本都是这个套路，根据题目要求调整输出信息即可。

1. **void** Map(Input<Document>\* input) {
2. **while** (!input->done()) {
3. stringstream ss;
4. string word;
5. ss << input->value().content;
6. **while** (ss >> word)
7. output(word,input->value().id);
8. input->next();
9. }
10. }
12. **void** Reduce(string &key, Input<**int**>\* input) {
13. vector<**int**> idList;
14. **while** (!input->done()) {
15. **if** (idList.size() == 0 || input->value() != idList.back()) {
16. idList.push\_back(input->value());
17. }
18. input->next();
19. }
20. output(key,idList);
21. }

508.摆动排序

使数组满足nums[0] <= nums[1] >= nums[2] <= nums[3] …

首先将数组进行排序，然后如果数组的长度为偶数，那么把倒数第二个元素和正数第二个元素进行交换，步长为2，直到两个指针不相等。

如果数组的长度为奇数，那么把倒数第一个元素和正数第二个元素进行交换。步长为2，直到两个指针不相等。

507.摆动排序2

使数组满足nums[0] < nums[1] > nums[2] < nums[3] …

首先将数组进行排序，然后找到数组的中间一个元素，（size+1）/2（属于后半部分）。然后从前半部分拿出倒数第一个元素，从后半部分拿出倒数第一个元素，组成新数组的第一第二个元素，以此类推，直到所有元素都安放合理。

510.最大矩形

这个题目和122求直方图的最大覆盖面积的解法一样，转化为以矩形中的每一行为底的直方图的最大覆盖面积。首先要计算一个矩阵，即每一行对应的直方图的高度。然后逐行按照求直方图的最大覆盖面积的方法来求解。（求解思路可见122题，注意使用单调栈时，为了计算最后一个块，要在高度向量最后补充一项0）。

511.交换链表当中的两个节点

为了防止删除头指针的特殊情况出现，我们为链表加上一个新的头指针，这样可以把删除的情况都简化为删除链表中的元素。然后找到待交换的两个节点的前一个节点，之后分情况讨论。

按照交换的两个节点是否相邻分两种情况讨论。

1. **if** (abs(flag\_q - flag\_p) == 1) {
2. ListNode \* temp = q->next->next;
3. q->next->next = p->next;
4. p->next = q->next;
5. q->next = temp;
6. **return** new\_head->next;
7. }
8. **else** {
9. ListNode \*temp\_1 = q->next->next;
10. q->next->next = p->next->next;
11. p->next->next = temp\_1;
12. ListNode \* temp\_2 = q->next;
13. q->next = p->next;
14. p->next = temp\_2;
15. **return** new\_head->next;
16. }

512.解码方法

这个题目与上楼梯问题很像，使用动态规划来解决。首先声明dp[n+1]表示到第i个字母有多少种解法。

首先进行初始化，dp[0] = 1; dp[1]根据第一个字符来初始化，如果为0则为0，否则为1。

接着进行动态规划，如果str[i-1]不为0，那么dp[i] = dp[i-1](至少有这么多解法)。

然后判断str[i-2]和str[i-1]新组成的两位数是否在10到26这个区间内，如果是，dp[i] += dp[i-2]。如此迭代直到找到最后的解dp[n]。注意str中的第i-1个字符对应的是dp中第i个位置。

513.完美平方

求一个数最少有几个平方数之和，典型的动态规划问题，使用dp[n]表示数字n的最小平方数之和。初始化dp[0]=0,dp[1] = 1。然后i从1到n开始循环，如果i是一个平方数（(int)sqrt(i)\*(int)sqrt(i) == i），则dp[i] = 1。否则j从1到j\*j < i进行循环，找到dp[i] = min(dp[i]，dp[j\*j] + dp[i-j\*j])。最后可以递推出问题的解。

514.栅栏染色

这个问题与上楼梯问题很类似，要求是不能出现三个连续的颜色一样的柱子。使用dp[i]表示到第i个柱子的染色方案。初始化dp[0] = k，dp[1] = k\*k。然后根据递推式dp[i] = dp[i-2]\*(k-1) + dp[i-1]\*(k-1)[解释为最后两根柱子涂一样的颜色，或者涂不一样的颜色]。最后递推出问题的解。

515.房屋染色

这个问题和上一题类似，也是典型的动态规划问题，使用dp[i][j]表示到第i个房子染j色的最小代价。首先是初始化操作dp[0][j] = costs[0][j]。在迭代过程中，dp[i][j] = min(dp[i][j], dp[i-1][k] + costs[i][j])其中（k!=j）。最后在dp[n][i]中选择一个最小的值即为问题的解。

516.房屋染色2

这个问题的解和上一题完全一样，题目要求O（nk）的解法，解析说染到i的不同颜色的最小值只是基于i-1的最小值和i-1的第二小值。所以只需要用fi,si两个下标分别记录染到i-1时的最小颜色以及第二小颜色即可。（不太明白）

517.丑数

只含有2，3，5因子的数，判断时，%2==0则一直/2。%3==0则一直/3。%5==0则一直/5。最后如果剩下的数为1则说明是丑数，否则不是。

518.超级丑数

把生成的每个丑数放在一个vector a中。初始化时，把1放入a。第二个丑数是a中的所有数与丑数因子的乘积中的比当前a中最大的数还大的第一个数。如此递推可以得到最后的解。

1. **while** (record.size() < n) {
2. **long** **long** max\_value = record[record.size()-1];
3. **long** **long** value = INT\_MAX;
4. **for** (**int** i = 0;i < primes.size();i++) {
5. **for** (**int** j = 0;j < record.size();j++) {
6. **long** **long** temp = primes[i]\*record[j];
7. **if** (temp > max\_value) {
8. value = min(temp,value);
9. }
10. }
11. }
12. record.push\_back(value);
13. }

524.左填充

水题，注意把题目中的函数声明成static即可。

526.负载均衡器

感觉是水题，题解也能符合要求，但是运行到50%会网站卡死，不知道原因，不 太明白。

528.摊平嵌套的列表  
 使用队列来记录，注意使用递归处理嵌套的结构即可（如果是单个的整数则直接放入队列中，否则按照列表来递归调用操作过程）。

532.逆序对

这个题目可以使用归并排序来解决，在一次归并排序的合并过程中，使用两个指针分别指向要合并的两个数组头，如果前面的值大于后边，那么说明i到mid这mid-i+1个数都会和后边的数字组成逆序对，如此统计可以求得所有的逆序对。

534.打劫房屋2

打劫房屋这个题目可以使用动态规划求解。dp[i]表示到第i个房屋能够获得的最大利润。dp[i] = max(dp[i-2] + nums[i],dp[i-1])。

针对房屋环的问题，我们把上述动态规划函数写成带有区间约束的，即可以限制动态规划的起始位置。然后这个问题可以根据是否打劫第一个房屋求0到n-2和求1到n-1的两个区间的最大值。

535.打劫房屋3

这个题目和树相结合，使用递归的方法，主要思想是使用一个具有两个元素的向量res来记录每个节点对应的结果。

res[0]表示不取这个点的值的最大值（可以取左右子树），res[1]则表示取这个点的值的最大值（不能取左右子树）。

每次递归，可以分别求得左右子树的res然后分情况讨论。

1. vector<**int**> record(TreeNode \* root) {
2. **if** (root == NULL) {
3. **return** {0,0};
4. }
5. vector<**int**> left = record(root->left);
6. vector<**int**> right = record(root->right);
7. vector<**int**> res = {0,0};
8. res[0] = max(left[0],left[1]) + max(right[0],right[1]);
9. res[1] = root->val + left[0] + right[0];
10. **return** res;
11. }

539.移动零

水题，从头到尾一次遍历，找到值为0的点之后删除该点，然后把这个值放在尾部即可。

540.左旋右旋迭代器

水题，根据奇偶数交替遍历两个数组即可（注意最后把没遍历完的添在结果后边）。

541.左旋右旋迭代器

使用一个数组记录每一行当前数组的长度。

然后对这个二维数组进行交替循环，如果当前这一行记录的长度不为0则说明当前行还有元素可以进行输出，index为原始长度-记录数组中的长度。然后把记录数组中对应行数的长度减一。最终完成所有元素的遍历。

544.前K大数

可以使用最小堆来做，依次遍历，堆中只存放K个最大的元素。如果元素比当前堆中最小的元素还大，则把队列头的元素弹出加入新的元素。

其中最大，最小堆可以使用STL 中的priority queue实现：

1. priority\_queue<**int**, vector<**int**>, greater<**int**> > q;
2. priority\_queue<**int**, vector<**int**>, less<**int**> > q;

使用greater队列中的元素按照升序排列，队头元素是最小的。

使用less队列中的元素按照降序排列，队头元素是最大的。

545.前K大数2

思路和上题一致，注意要考虑元素个数不足K个时的情况。

547.两数组的交集

把两个数组排序，对于数组1中的每个数，使用二分法在数组2中查找，把查找结果放入set中（不会有重复元素）。最后把集合中的元素放入vector即可，要注意的是set的遍历要使用迭代器来做，不能使用数组下标的方式直接做。

548.两数组的交集2

这个题目和上题不同的是交集中可以包含重复的元素。所以先把两个数组排序，然后使用两个指针从头向后遍历，如果nums1[i] == nums2[j]则把值放入结果数组，如果nums1[i] > nums2[j]则数值较小的那个数组（nums2[j]）游标向后移动，否则nums1[i]对应的游标向后移动。

550.最常用的k个单词

首先建立struct来存储单词以及出现的次数。

使用一个vector<struct word>来存储前k个单词，注意其中排序的方式是先按照出现的次数，然后按照字典序。

使用一个map来记录所有单词出现的次数。

在添加一个单词时，如果vector中有，则直接将对应的单词的频数加一并排序。否则看这个单词和vector中最小的那个哪个比较大。如果新单词大则要改变vector中的的单词及其次序（注意如果vector中不足k个直接添加）。

要注意本题超时的原因是在vector中寻找，但是虽然按出现的次数排序去没能够很好地使用二分法解决。不太明白。

551.嵌套列表的加权和

使用递归即可完成求解，注意参数的传递和使用方式。

552.创建最大数

这个问题要求使用两个数组来创建一个最大数，并且保持数字在原来的数组中顺序不变。因而这个问题可以转化为第一个数组出i位最大数，第二个数组出i-k位最大数，组合的数最大。

需要两个辅助函数来做，第一是求两个数的合并的最大数。

比较简单，直接使用两个游标，向生成的数组中放入当前较大的数即可。

第二个辅助函数是求数组中的k位最大数。

求k位最大数，那么在生成过程中，整个数组最多舍弃size – k位元素。使用一个k位的向量来记录结果，对于每个数，把它放在当前最大的位置上（结果数组中比这个数小的要弹出），最后输出结果即可。

1. vector<**int**> find\_vec(vector<**int**> num, **int** k) {
2. vector<**int**> res;
3. **int** m = num.size() - k;
4. **for** (**int** i = 0;i < num.size();i++) {
5. **while** (m > 0&&res.size() > 0&&res.back() < num[i]) {
6. res.pop\_back();
7. m--;
8. }
9. res.push\_back(num[i]);
10. }
11. **return** res;
12. }

553.炸弹袭击

直接用暴力法从每个位置向四个方向寻找会超时。

可以采用递推式来做，考虑四个方向上的不同炸弹，以考虑上部分节点为例，使用动态规划。如果当前节点为’E’那么使用炸弹可以击杀一个敌人，如果为’0’则dp[i][j] = dp[i-1][j]，如果是墙’W’的话，则dp[i][j] = 0。

dp[i][j] 的递推分三种情况，将其分四个方向记录。最后某一点的结果为四个方向的和。经过一轮比较就可以得到最优解。

555.计数型布隆过滤器

水题，使用map来记录出现的字符串和对应的出现次数，当进行删除操作时，要对出现的次数进行减操作，当减到零时把改项删去。

556.标准型布隆过滤器

水题，和上题的思路一样，只是不再统计次数，删除的时候直接进行删除即可。

564.组合总和4

典型的动态规划，使用dp[i]表示数字i共有多少种解决方案。可以分析求得递推式为（num为原始数组集合）：

dp[i] += dp[i-num[i]]，即逐次把num中的每个元素当作一个总体，然后分解剩下的数，最后可以递推出问题的解。

569.各位相加

水题，直接按位分解然后递归求解即可。

570.寻找丢失的数2

对问题分析后，每次取数的时候可以取一位，也可以取两位。所以采用回溯法来做。使用一个vector来记录已经解析除了哪些数，如果最后只剩下一个数没有解析，则得到最后的结果。

在回溯的过程中，同样使用一个vector记录那些数已经被访问，回溯的函数定义格式如下：

1. **int** partition\_vec(vector<**int**>& visited, **int** index, **int** n, string &str) {
2. **if** (index == str.length()) {
3. vector<**int**> res;
4. **for** (**int** i = 1;i < visited.size();i++) {
5. **if** (visited[i] == 0) {
6. res.push\_back(i);
7. }
8. }
9. **if** (res.size() == 1) {
10. **return** res[0];
11. }
12. **else** {
13. **return** -1;
14. }
15. }
16. **if** (str[index] == '0') {
17. **return** -1;
18. }
19. **for** (**int** i = 1;i < 3;i++) {
20. **int** temp = strToint(str.substr(index,i));
21. **if** (temp >= 1 && temp <= n && visited[temp] == 0) {
22. visited[temp] = 1;
23. **int** val = partition\_vec(visited,index+i,n,str);
24. **if** (val != -1) {
25. **return** val;
26. }
27. visited[temp] = 0;
28. }
29. }
30. **return** -1;
31. }

其中，index用来记录当前开始遍历的位置，如果当前位置为‘0’或者当前走一步或者两步获得的数字大于n，则都是不合法的，要返回-1。在回溯之前，要判断结果是否为-1。如果不为-1说明已经完成了成功的划分，直接返回即可。

573.邮局的建立2

这个题目使用宽度遍历的思想来做，首先对整个二维数组进行遍历，记录下所有房子节点的位置，然后从每个房子节点开始进行宽度遍历。使用visited[i][j]来记录每个空地可以访问到的房子节点个数，使用path[i][j]来记录每个空地访问其所能访问的房子的路径和。判断的时候找到能够访问所有房子的最小路径空地位置即可。

主要难度是在于如何完成宽度遍历，使用队列进行辅助，每个点可以向上下左右四个方向进行探索（使用两个大小为4的一维数组提高编程效率）。队列每次出头节点的时候要把步长加1，同时使用一个二维数组记录当前节点是否被访问过。只有下一个节点满足**没有越界，是空地并且没被访问过**这三个条件才能够走这个节点。最后完成搜索，代码如下所示：

1. **void** bfs(vector<vector<**int**> > &grid,vector<vector<**int**> > &visited,vector<vector<**int**> > &path,point p) {
2. **int** dx[4] = {-1,1,0,0};
3. **int** dy[4] = {0,0,-1,1};
4. **int** n = grid.size();
5. **int** m = grid[0].size();
6. **int** record[n][m];
7. **for** (**int** i = 0;i < n;i++) {
8. **for** (**int** j = 0;j < m;j++) {
9. record[i][j] = 0;
10. }
11. }
12. queue<point> q;
13. q.push(p);
14. record[p.x][p.y] = 1;
15. **int** steps = 0;
16. **while** (!q.empty()) {
17. **int** length = q.size();
18. **for** (**int** i = 0;i < length;i++) {
19. point top = q.front();
20. q.pop();
21. visited[top.x][top.y] ++;
22. path[top.x][top.y] += steps;
23. **for** (**int** j = 0;j < 4;j++) {
24. point next(top.x + dx[j], top.y + dy[j]);
25. **if** (isVaild(grid,next.x,next.y) && record[next.x][next.y] == 0 && grid[next.x][next.y] == EMPTY) {
26. q.push(next);
27. record[next.x][next.y] = 1;
28. }
29. }
30. }
31. steps ++;
32. }
33. }

575.字符串解码

递归问题，写一个函数来匹配括号（找到当前括号匹配的括号），每次遇到数字时，先把数字解析出来（注意分析数字的位数，同时单独处理数字为0的情况），然后把数字后括号内的字符串进行递归处理，最后把解析出的字符串重复相应的次数即可。

577.合并K个排序间隔列表

写两个函数，一个判断间隔是否能合并，另一个判断间隔合并的结果。然后把间隔放入一个vector中，以间隔的起点为关键字进行排序（升序），然后每次拿出前两个元素，判断是否能合并，如果能，则把合并后的结果放入，继续判断，如果不能则向后判断，直到遍历整个vector。

578.最近公共祖先3

这个题目和第88题一致，就是多了一步，首先判断两个节点是否在树上（类似遍历，递归查找即可）。然后按照之前的思路，先判断这两个节点是否跟根节点相等，是的话直接返回，否则在左右子树递归查找，返回左右子树不为空的查找结果，若都为空则返回根节点。

582.单词拆分2

这个题目首先使用dp[i](1-n)记录到第i个字符能否被划分，使用循环先计算这个数组，如果dp[n]为真，说明可以划分。

1. **bool** isBreak(string &s, unordered\_set<string> &wordDict)
2. {
3. **int** len = s.size();
4. vector<**bool**> dp(len + 1, **false**);
5. dp[0] = **true**;
6. **for** (**int** i = 1; i <= len; ++i)
7. {
8. **for** (**int** j = 0; j < i; ++j)
9. {
10. **if** (dp[j] && (wordDict.find(s.substr(j, i-j)) != wordDict.end()))
11. {
12. dp[i] = **true**;
13. **break**;
14. }
15. }
16. }
17. **return** dp[len];
18. }

然后使用回溯法使用深度遍历的方式可以得到结果。

1. **void** findWord(string &s, unordered\_set<string> &wordDict,**int** len,string &str,vector<string> &res)
2. {
3. **if** (s.size()==0)
4. {
5. res.push\_back(str);
6. }
7. **else**
8. {
9. **for** (**int** i = len; i <= s.size(); ++i)
10. {
11. **if** (wordDict.find(s.substr(0, i)) != wordDict.end())
12. {
13. string nstr = s.substr(i);
14. string mstr = str;
15. str += (s.substr(0, i) + " ");
16. findWord(nstr, wordDict, 1, str,res);
17. str = mstr;
18. }
19. }
20. }
21. }

585.山脉序列中的最大值

一次遍历，找到递减序列的第一个元素（或者是递增序列的最后一个元素）即为问题的答案（注意要判断数组长度为1的情况）。

588.划分和相等的子集

首先对数组进行排序，然后计算整个数组的和，同时使用dp[i]记录到第i个数字的和。接着找到位置i，使得dp[i] > sum/2。然后使用一个指针j从前向后（0~i）进行寻找，如果满足dp[i] – dp[j] 或者dp[i] – nums[j]的值为sum/2则返回真，否则返回假。

591.连接图3

使用并查集的思想来做，并查集的寻找格式如下所示（类似于寻找领导的领导）：

1. **int** unionSearch(**int** target, vector<**int**>& node\_record) {
2. **while** (target != node\_record[target]) {
3. node\_record[target] = node\_record[node\_record[target]];
4. target = node\_record[target];
5. }
6. **return** target;
7. }

594.字符串查找2

直接可以用暴力法过，每次判断可以使用首尾字符串限制一下，能减少一些判断。

也可以使用Rabin Karp算法来做，使用hash计算source和target的hash code进行比较。核心是计算source的hash code，依次计算每target长度的hash code。

hashcode(cd) = (hashcode(bc + d) - hashcode(b)\*2 ) % BASE如果hashcode值为负，则 hashcode += BASE。（不太明白<https://blog.csdn.net/roufoo/article/details/82955646>）

595.二叉树的最长连续序列

题目要求最长序列要遵循父子关系，因而在查询的时候，只需要考虑根到孩子一个方向。对左右两个子树分开查询（首先判断是否为空，再继续递归查询），并使用一个map记录到达某一点后当前路径的长度。每次判断时，如果路径连续，则把当前的路径数目加一，如果路径不连续则把当前的路径数置为一。

最后遍历map找到对应的最长路径即可。

599.向循环有序链表中插入节点

本题目的关键是如何找到插入的位置，找到之后因为链表是循环的，可以很方便的找出插入节点的前一个节点和后一个节点来执行插入操作。经过分析，链表的插入位置只有两种情况，要么能够在链表中找到一个合适的位置（大于等于前一个值，小于等于后一个值），要么这个插入值比链表中的所有值都小（或者大），在这种情况下都是插入链尾和头之间。

600.包裹黑色像素点的最小矩形

由题目可知，黑色像素点都是连在一起的（题目给的示例二没练在一起），所以给出黑色区域的一个点，使用深度遍历的方法向上下左右四个方向进行探索，找到所有黑色的像素点（传统方法，使用一个矩形来判断点是否被访问过）。在遍历的过程中记录黑色矩形点的横，纵坐标的最大和最小值，通过（y\_max – y\_min）\*（x\_max - x\_min）可以求出包裹黑色像素点的最小矩形。

601.摊平二维向量

水题，只需要先使用一个一维的向量把二维向量中的元素记录下来，然后维护一个下标记录当前遍历的位置，如果下摆到了一维向量的末尾，则hasnext()方法返回false，否则可以返回一个数组中的元素，并把下标加一。

602.俄罗斯最大套娃信封

这个题目有两种解法：

1. 使用传统的动态规划，使用一个一维数组来存储中间的结果即可，递推式如下：

dp[i] = max(dp[j] + 1, dp[i])，其中j的范围是i-1到0，如果第i个信封比第j个信封大，则使用该递推式求解。注意，最后的递推结果不是dp[n-1]，因为最后一个信封可能没有被取走，所以要在中间过程中维护一个变量记录结果的最大值。

(为了降低复杂度，可以使用二分法来做，二分法的代码不太明白)

1. ///二分法
2. **static** **bool** cmp(pair<**int**,**int**> &a,pair<**int**,**int**>& b){
3. **if**(a.first==b.first) **return** a.second>b.second;
4. **else** **return** a.first<b.first;
5. }
6. **int** maxEnvelopes(vector<pair<**int**, **int**>>& envelopes) {
7. **int** n=envelopes.size(),len=0;
8. sort(envelopes.begin(),envelopes.end(),cmp);
9. vector<**int**> vec(n,0);
10. **for**(**int** i=0;i<n;i++){
11. **int** m=envelopes[i].second;
12. auto it=lower\_bound(vec.begin(),vec.begin()+len,m);
13. **if**(it==vec.begin()+len) vec[len++]=m;
14. **else** \*it=m;
15. }
16. **return** len;
17. }
18. 首先把信封的宽按照升序排列（宽相同按照高的降序排列）。然后把问题转化成寻找最长上升子序列的问题。

603.最大整除子集

首先把集合中的元素按升序排列，并使用map<i,vector<int>>记录当集合中的最大元素为i时的最大整除子集。两层循环，当一个元素j能够整除i时，并且当前记录的j的最大整除子集的规模大于i的最大整除子集的规模时，可以将i的最大整除子集的个数加一(包括了j)，最后遍历map找到最大整除子集即可。核心代码如下所示：

1. **for** (**int** i = 0;i < n;i++) {
2. temp.clear();
3. **for** (**int** j = i-1;j >= 0;j--) {
4. **if** (nums[i]%nums[j] == 0 && record[nums[j]].size() > temp.size()) {
5. temp = record[nums[j]];
6. }
7. }
8. temp.push\_back(nums[i]);
9. record[nums[i]] = temp;
10. }

605.序列重构

拓扑排序问题，使用map<int,set<int>>这个结构来记录邻接链表，使用map<int，int>来记录每个节点的入度。拓扑排序使用队列作为辅助数据结构来做，如果根据结果得出拓扑排序的序列是唯一的（根重构序列比较，必须每个位置都相同），那么可以重构序列，否则不可以。

607.两数之和III-数据结构设计

水题，使用一个vector来记录插入的元素，并使用一个标志位来记录数组是否有序（避免过多的重排序）。在寻找两数之和时，首先确保数组按照升序排列，然后使用两根指针从头向尾，从尾向头来查找，如果得出的和大于目标值，则尾部指针移动，否则头部指针移动，直到找到两数之和等于目标值（两根指针相遇则寻找失败）。

608.两数和II-输入已排序的数组

水题，按照上题的方法来寻找和为目标值的两个数的下标（加一）即可。

612.K个最近的点

声明一个类记录每个节点的信息以及该节点距离目的节点的距离，根据题目中给出的节点，生成一个保存新的类的数组，重构compare方法，按照题目中的要求进行排列。然后使用sort函数进行排序，最后找出前K小个节点即可。

614.二叉树的最长连续子序列II

这个题目与595题的不同是节点不用满足父子关系，因而可以从左右两个方向生长，使用up,down两个变量记录从根节点到左右两棵子树的路径，当前的最长路径是up+down-1，然后比较出最长的路径即可。

1. **class** ReturnInfo {
2. **public**:
3. **int** longest\_up;
4. **int** longest\_down;
5. ReturnInfo(**int** up, **int** down):longest\_up(up), longest\_down(down) {};
6. };
8. **int** longest = 0;
10. ReturnInfo longestInfo(TreeNode \* root) {
11. **if** (root == NULL) **return** ReturnInfo(0,0);
12. **int** up = 1;
13. **int** down = 1;
14. ReturnInfo tempL = longestInfo(root->left);
15. ReturnInfo tempR = longestInfo(root->right);
17. **if** (root->left) {
18. **if** (root->left->val == root->val - 1)
19. up = max(up, tempL.longest\_up+1);
20. **if** (root->left->val == root->val + 1)
21. down = max(down, tempL.longest\_down+1);
22. }
23. **if** (root->right) {
24. **if** (root->right->val == root->val - 1)
25. up = max(up, tempR.longest\_up+1);
26. **if** (root->right->val == root->val + 1)
27. down = max(down, tempR.longest\_down+1);
28. }
29. longest = max(longest, up+down-1);
30. **return** ReturnInfo(up,down);
31. }

615.课程表

使用拓扑排序解决即可。

616.安排课程

和上一题的思路一致，使用拓扑排序解决，输出拓扑排序的中间结果。

617.最大平均子数组II

首先想到采用递推的方式求解，使用dp[i][j]表示从i开始长度为j的子数组的和。

能够通过79%的用例，因为后来数组太大，出现溢出。

网上采用二分法的解法**不太明白**：

首先找到数组的最大值与最小值，确定平均值的范围；

不断缩小范围，在r-l的值足够小的时候，得到结果

令mid=(r+l)/2，将原数组的每个元素减去mid，如果可以找到至少k个相邻元素的和>0，说明最终结果一定比现在的mid要大，在[l,mid]范围内寻找，否则，在[mid,r]范围内寻找。

接下来判断是否存在有至少k个相邻元素的>0，利用数组sum[i]存储sum[i]-mid到sum[i-1]-mid的和，当i>k时，用一个min\_pre的变量来保存sum[0]~sum[i-k]之间的最小值，并且每次随着i变大不断更新该数字。这样sum[i]-min\_pre得到的是sum[k+1]~sum[i-1]，它所记录的总和个数也就是到num[i]为止能够找到的最大平均值子数组的长度。

622.青蛙跳

这个问题使用递归求解，首先从后向前判断，如果一个点大于它前一个点的两倍，则肯定无法到达，首先筛除这种情况。然后使用一个函数判断从当前节点走step-1，step，step+1步能够到达的点。接着从当找到的点出发，递归寻找能否到达终点。

1. **bool** CanCross(vector<**int**> &stones, **int** cur, **int** lastStep) {
2. **if** (cur == stones.size() - 1) {
3. **return** **true**;
4. }
5. **int** next = 0;
6. **if** ((next = find\_next(stones, cur, lastStep + 1)) != -1
7. && CanCross(stones, next, lastStep + 1)) {
8. **return** **true**;
9. }
10. **if** ((next = find\_next(stones, cur, lastStep)) != -1
11. && CanCross(stones, next, lastStep)) {
12. **return** **true**;
13. }
14. **if** ((next = find\_next(stones, cur, lastStep - 1)) != -1
15. && CanCross(stones, next, lastStep - 1)) {
16. **return** **true**;
17. }
18. **return** **false**;
19. }

623.K步编辑

解法参照第119题，使用动态规划求解，在计算单词距离之前，可以先通过判断两个单词长度差是否小于等于k，然后决定是否进行下一步判断。

626.矩阵重叠

通过判断两个矩阵最远点组成的最大长和宽是否能够大于两个单独矩阵的长和宽来判断两个矩阵是否重叠。

627.最长回文串

建立一个map，统计给出的字符串中单词出现的次数，然后遍历整个map，如果一个字母出现了偶数次，那么可以直接添加到回文串中，如果是奇数次，要维护一个值记录奇数次出现最多的单词，这个最大次数可以累加到结果中，然后别的出现奇数次的单词可以贡献的长度是出现的次数减一。

632.二叉树的最大节点

水题，全局变量 + 递归求解。

633.寻找重复的数

使用表来记录可以很快求解。

为了把辅助空间限制在O(1)。可以使用映射法来解决这个问题，把数字下标作为映射，重复出现的数导致映射关系中有环出现，相当于使用快慢指针找环的入口。代码如下：

1. **int** findDuplicate(vector<**int**> &nums) {
2. **int** n = nums.size();
3. **int** slow = 0;
4. **int** fast = 0;
5. **do** {
6. slow = nums[slow];
7. fast = nums[nums[fast]];
8. } **while**(slow != fast);
9. fast = 0;
10. **while** (fast != slow) {
11. slow = nums[slow];
12. fast = nums[fast];
13. }
14. **return** fast;
15. }

634.单词矩阵

使用map建立一个前缀表，即前缀长度从0到n每个前缀所对应的单词。

然后根据条件逐个放置即可。

1. vector<vector<string>> wordSquares(vector<string>& words) {
2. vector<vector<string>> res;
3. unordered\_map<string, set<string>> m;
4. **int** n = words[0].size();
5. **for** (string word : words) {
6. **for** (**int** i = 0; i < n; ++i) {
7. string key = word.substr(0, i);
8. m[key].insert(word);
9. }
10. }
11. vector<vector<**char**>> mat(n, vector<**char**>(n));
12. helper(0, n, mat, m, res);
13. **return** res;
14. }
16. **void** helper(**int** i, **int** n, vector<vector<**char**>>& mat, unordered\_map<string, set<string>>& m, vector<vector<string>>& res) {
17. **if** (i == n) {
18. vector<string> out;
19. **for** (**int** j = 0; j < n; ++j) out.push\_back(string(mat[j].begin(), mat[j].end()));
20. res.push\_back(out);
21. **return**;
22. }
23. string key = string(mat[i].begin(), mat[i].begin() + i);
24. **for** (string str : m[key]) {
25. mat[i][i] = str[i];
26. **int** j = i + 1;
27. **for** (; j < n; ++j) {
28. mat[i][j] = str[j];
29. mat[j][i] = str[j];
30. **if** (!m.count(string(mat[j].begin(), mat[j].begin() + i + 1))) **break**;
31. }
32. **if** (j == n) helper(i + 1, n, mat, m, res);
33. }
34. }

636.132模式

题目要求中间的数最大，右边的数小于左边的数。首先使用一个数组记录到下标i的最小值。然后把数组从后向前遍历并逐个入栈，如果有数大于当前下标对应的最小值，那么把当前元素和栈顶元素进行比较，如果当前元素比栈顶元素大，则出栈，并记录出栈的元素，最后进行比较，如果记录的右边的最大值大于左边的最小值即找到了132模式。

1. **bool** find132pattern(vector<**int**> &nums) {
2. **int** n = nums.size();
3. **if** (n < 3) {
4. **return** **false**;
5. }
6. **int** min\_r\_l[n] = {INT\_MAX};
7. min\_r\_l[0] = nums[0];
8. stack<**int**> max\_s;
9. **for** (**int** i = 1;i < n;i++) {
10. **if** (nums[i] < min\_r\_l[i-1]) {
11. min\_r\_l[i] = nums[i];
12. }
13. **else** {
14. min\_r\_l[i] = min\_r\_l[i-1];
15. }
16. }
17. **for** (**int** i = n-1;i >= 0;i--) {
18. **if** (nums[i] > min\_r\_l[i]) {
19. **int** max\_value = INT\_MIN;
20. **while** (!max\_s.empty()&&nums[i] > max\_s.top()) {
21. max\_value = max\_s.top();
22. max\_s.pop();
23. }
24. **if** (max\_value > min\_r\_l[i]) {
25. **return** **true**;
26. }
27. }
28. max\_s.push(nums[i]);
29. }
30. **return** **false**;
31. }

637.检查缩写字

字符串配对问题同时涉及字符串转整数。使用两根指针分别指向两个字符串，从头到尾遍历，如果两个指针对应的字符串相同或者是缩写字符串中的数字为0，那么指针同时向后移动。如果缩写字符串中的数字在1~9之间，那么从缩写字符串的当前位置向后寻找，找到当前对应的数字，第二个指针移动到非数字位置，第一个指针向后移动数字代表的值。如果两个指针对应的都是字符且不相等则返回false。如果最后两个指针都分别走到字符串的末尾返回true，否则返回false。

638.字符同构

这个题有个陷阱，需要使用两个map分别记录a->b的映射和b->a的映射。

求映射的过程是一致的，两个字符串逐个匹配，如果出现了匹配过的字母对应不同的字符则匹配失败。只有两个方向都成功匹配才能说明字符同构。

639.单词缩写

这个题目需要根据一个源字符串数组构建一个前缀长度数据，分别存储每个单词当前存储的前缀的长度（从一开始）。同时需要写一个函数来生成一个单词前缀长度为n时，对应的缩写情况（前缀长度对应的字符+缩写的字母数+结尾。同时注意，缩写过后不能缩小长度则不缩写）。然后对于所有前缀长度为1的缩写情况，分别判断他们之间是否有冲突（相等），如果有，则需要把冲突双方放入一个集合中，然后把集合中的对应的元素的前缀长度加一，知道没有冲突出现。最后得到所有的缩写元素。

640.一次编辑的距离

这个题目和单词编辑的思路完全一致，使用动态规划找到两个单词的最短编辑距离就行了。

641.丢失的间隔

水题，直接按照题目要求求解即可，特别需要注意的是对于整数边界情况的处理，不要造成溢出。

642.数据流滑动窗口的平均值

这个题目维持一个队列，同时记录当前的和即可。如果窗口溢出则使用当前的和减去溢出的数字同时加上新数字求解即可（注意和要用long long来存储）。

643.最长绝对文件路径（这道题本地编译结果和提交的不一致，不太明白）

使用一个map记录到某一层次所对应的文件路径的最长长度。首先要数初始化map[0] = 0。然后从头至尾遍历字符串，根据字符’\n’和’\t’进行分割。对于分割出的文件名，查找其中是否包含’.’来判断其是否是文件名，如果是的话就更新结果，否则更新map。同时在查找过程中，使用字符’\t’来记录当前遍历的深度。

1. **int** lengthLongestPath(string &input) {
2. **int** res = 0;
3. **int** level = 0;
4. **int** n = input.length();
5. map<**int**, **int**> record;
6. record[0] = 0;
7. **for** (**int** i = 0;i < n;i++) {
8. **int** start = i;
9. **while** (i < n && input[i] != '\n' && input[i] != '\t') {
10. i ++;
11. }
12. **if** (i >= n || input[i] == '\n') {
13. string s = input.substr(start,i - start);
14. **if** (find\_point(s)) {
15. res = max(res,record[level] + (**int**)s.length());
16. }
17. **else** {
18. level ++;
19. record[level] = record[level-1] + (**int**)s.length() + 1;
20. }
21. level = 0;
22. }
23. **else** {
24. level ++;
25. }
26. }
27. **return** res;
28. }

644.镜像数字

首先使用一个map记录每个0到9每个数字旋转180度后的映射结果（如果翻转之后不是一个合法的数字则使用‘＃’代替），然后把原数字按照这个map转化成旋转之后的数字。最后比较这两个数字是否相等即可。

645.识别名人

把改题目抽象成图问题，A认识B可以抽象成A到B有一条有向边。使用邻接矩阵来做，根据题目的定义可以得知，名人的定义可以转化为入度为n-1（邻接矩阵的列之和），出度为0（邻接矩阵对应的出度之和）。

根据条件构造矩阵，然后依次判断每个点是否是名人即可。

646.第一个独特字符位置

找到第一个出现了一次的字符，首先使用map记录每个字符出现的次数，然后再从头至尾一次遍历，找到第一个出现了一次的字符，没有的话则返回-1。

647.子串字谜

如果两个单词是同构的，那么其对应的map<char,int>[每个字符对应出现次数]应该是相等的，对于要判断的单词，我们首先建立对应的map再判断是否和源单词生成的map相等即可。

648.单词缩写集

使用map<string,set<string>>结构来解决这个问题，存储每个缩写所对应的单词。判断是否缩写是否唯一时只要判断集合中的对应的元素个数即可。

649.二叉树翻转（不太明白）

这个题题目使用递归求解，经过分析可知，如果当前根节点的左子树不为空，翻转之后，当前根节点的左节点变为新的根节点，当前根节点的左节点的左节点是当前根节点的右节点，当前根节点的左节点的右节点是根节点。

1. TreeNode \* upsideDownBinaryTree(TreeNode \* root) {
2. **if** (root == NULL || root->left == NULL) {
3. **return** root;
4. }
5. TreeNode \* new\_root = upsideDownBinaryTree(root->left);
6. root->left->left = root->right;
7. root->left->right = root;
8. root->left = NULL;
9. root->right = NULL;
10. **return** new\_root;
11. }

650.二叉树叶子节点遍历

使用“剥洋葱”的算法，逐层遍历二叉树的叶子节点，对于已经遍历过的叶子节点，将其置为控制，然后不断循环，直到根节点为空即可完成所有节点的遍历。

1. TreeNode \* help (vector<**int**> &temp, TreeNode \* root) {
2. **if** (root == NULL)
3. **return** NULL;
4. **if** (root->left == NULL && root->right == NULL) {
5. temp.push\_back(root->val);
6. **return** NULL;
7. }
8. root->left = help(temp,root->left);
9. root->right = help(temp,root->right);
10. }

651.二叉树的垂直遍历

使用层次遍历，对于每个遍历的节点使用一个map<int,vector<int>>记录每个对应的宽度有哪些节点（使用层次遍历是为了保障节点有序），然后使用一个vector<int>记录都出现过哪些宽度，最后对记录宽度的vector排序，再逐个输出map中对应的序列即可。要注意使用一个队列记录父节点的宽度值，左孩子的宽度值等于父节点宽度值减一，右孩子的宽度值等于父节点的宽度值加一。

653.添加运算符

典型的深度搜索题目，使用辅助函数划分字符串，逐个求解即可。注意保存当前的值和上一次计算的值，因为如果加了一个乘号，改变了运算的顺序，当前计算的结果等于当前结果减去上一次的值再乘以现在的运算数。同时注意处理现在运算数为0的特殊情况。

1. **void** dfs(vector<string> &res, string temp, **int** target, **int** cur\_val, **int** pre\_val, **int** pos, string num) {
2. **if** (pos == num.size() && cur\_val == target) {
3. res.push\_back(temp);
4. **return**;
5. }
6. **for** (**int** i = pos+1;i <= num.size();i++) {
7. string tmp = num.substr(pos,i-pos);
8. **long** **long** n = stoll(tmp);
9. **if** (tmp[0] == '0' && tmp.size() > 1) {
10. **break**;
11. }
12. **else** {
13. **if** (pos == 0) {
14. dfs(res,tmp,target,n,n,i,num);
15. }
16. **else** {
17. dfs(res,temp + "+" + tmp,target,cur\_val+n,n,i,num);
18. dfs(res,temp + "-" + tmp,target,cur\_val-n,-n,i,num);
19. dfs(res,temp + "\*" + tmp,target,cur\_val - pre\_val + pre\_val\*n,pre\_val\*n,i,num);
20. }
21. }
22. }
23. }

654.稀疏矩阵乘法

水题，对于稀疏矩阵，我们需要维护两个map分别记录第一个矩阵中哪一行全0，第二个矩阵中哪一列全0。在进行计算时，只有第一个矩阵的行不为0，第二个矩阵的列不为0才执行计算，否则直接在结果矩阵中填充0。

655.大整数加法

水题，按照字符串逐位相加即可，要注意进位的情况。

656.大整数乘法

结合上一个题目的加法操作，写一个辅助函数，计算一个数字乘一位数字的乘法函数，类似于加法操作，注意进位时，标志位不一定为1了。然后注意对应一位乘法计算之后要有移位操作。最后把结果相加即可。

657.Insert Delete GetRandom O(1)

使用一个vector和一个map来做，map可以使用函数count(val)来查询一个元素是否出现过。

插入操作时，如果count的结果不为0，则返回false。否则直接在vector的尾部插入，同时建立val和存储位置的映射关系存在map中。

删除操作时，如果count的结果已经为0，则返回false。否则首先通过map找到要删除的元素的位置。然后将其与vector的末尾的元素进行交换，接着删除vetor的末尾元素并把map中对应的项删除即可。

获得一个随机位置直接使用rand()%vector.size()即可。

659.编码和解码字符串

这个题目可以有多种编解码方法，我的方法时对于编码的每个字符串，加上前缀#length#即可。解码时，通过这个结构以及每个字符的长度就可以完成解码，注意子串分割时的边界范围。

660.用Read4从文件中读取N个字符II-多次调用（不太明白）

使用两个变量readPos和writePos来分别记录读取和写的位置，i从0到n开始循环，如果此时读和写的位置一样，那么我们调用read4函数，将结果赋给writePos，并把readPos置零，如果writePos为零则说明buf中额米有东西了，返回当前坐标i即可。然后我们用内置的buf变量的readPos位置覆盖输入字符串的i位置，如果完成遍历，则返回n。

1. **class** Solution {
2. **public**:
3. **int** read(**char** \*buf, **int** n) {
4. **for** (**int** i = 0; i < n; ++i) {
5. **if** (readPos == writePos) {
6. writePos = read4(buff);
7. readPos = 0;
8. **if** (writePos == 0) **return** i;
9. }
10. buf[i] = buff[readPos++];
11. }
12. **return** n;
13. }
14. **private**:
15. **int** readPos = 0, writePos = 0;
16. **char** buff[4];
17. };

661.把二叉搜索树转化为更大的树

直接使用中序遍历（注意这里为了计算累加值，先遍历右子树，后便利左子树）递归求解即可，使用一个全局变量记录累加值，然后更新每个节点的值。

662.猜数游戏

直接使用二分法求解即可。（注意二分法中有start + end 的情况，所以要注意判断溢出）。

663.墙和门

这个题目使用深度遍历来做，对数组中的每个门进行深度遍历操作，如果这个位置没有越界，同时当前数组该位置所存的值大于距离门的距离时，进行深度遍历，首先更新这个点的值，然后向四个方向进行递归遍历。

664.数1

使用x&(x-1)来计算每个数中对应的二进制中1的个数。

665.平面范围求和-不可变矩阵

水题，直接循环求和即可。

666.猜数游戏2

区间动态规划问题，假如区间的长度为1，则在[n,n]内，猜数的代价为0。如果区间的长度为2，在区间[n,n+1]内，猜数的代价为n。如果区间的长度为3，在区间[n,n+1,n+2]内，猜数的代价为n+1。类推到区间[n,n+1,…，n+m]，代价为：

dp[n][n+m] = min(dp[n][n+m],max(dp[n][x-1],dp[x+1][n+m]) + x)

根据递推式，进行初始化后递推即可得到问题的解。

1. **int** getMoneyAmount(**int** n) {
2. vector<vector<**int**> > dp(n+1,vector<**int**>(n+1,INT\_MAX));
3. **for** (**int** i = 1;i < n;i++) {
4. dp[i][i] = 0;
5. dp[i][i+1] = i;
6. }
7. dp[n][n] = 0;
8. **for** (**int** range = 2;range < n;range++) {
9. **for** (**int** i = 1;i < n;i++) {
10. **if** (i + range <= n) {
11. **for** (**int** j = i+1;j < i+range;j++) {
12. dp[i][i+range] = min(dp[i][i+range],max(dp[i][j-1], dp[j+1][i+range]) + j);
13. }
14. }
15. }
16. }
17. **return** dp[1][n];
18. }

667.最长回文序列

使用动态规划的方式求解，dp[i][j]表示从i到j的回文序列的最长长度。递推式如下所示：

dp[i][j] = dp[i+1][j-1] + 2 (s[i] == s[j]) / max(dp[i][j-1],dp[i+1][j]) (s[i] != s[j] )

具体写法如下：

1. **for** (**int** i = 0;i < n;i++) {
2. record[i][i] = 1;
3. **for** (**int** j = i-1;j >= 0;j--) {
4. **if** (s[i] == s[j]) {
5. record[j][i] = record[j+1][i-1] + 2;
6. }
7. **else** {
8. record[j][i] = max(record[j][i-1],record[j+1][i]);
9. }
10. }
11. }

668.一和零

这个问题可以理解成是装有两种物品的背包问题，使用动态规划求解。dp[i][j]表示还剩下i个0和j个1能够表示的最大字符串的个数。

首先要统计每个字符中0和1的个数，然后使用递推式求解：

dp[i][j] = max(dp[i][j],dp[i-number\_0][j-number\_1]+1)。

1. **for** (**int** i = 0;i < len;i++) {
2. **int** number\_0 = 0;
3. **int** number\_1 = 0;
4. **for** (**int** j = 0;j < strs[i].length();j++) {
5. **if** (strs[i][j] == '0') {
6. number\_0 ++;
7. }
8. **else** {
9. number\_1 ++;
10. }
11. }
12. **if** (number\_0 > m || number\_1 > n) {
13. **continue**;
14. }
15. **for** (**int** l = m;l >= number\_0;l--) {
16. **for** (**int** k = n;k >= number\_1;k--) {
17. dp[l][k] = max(dp[l][k],dp[l-number\_0][k-number\_1] + 1);
18. }
19. }
20. }

669.换硬币

这个问题使用贪心算法不行，因为可能有些数不能被整除。

所以使用动态规划的方式来求解：

dp[i] = min(dp[i-coins[0]] + 1, dp[i-coins[1]] + 1, dp[i-coins[2]] + 1,…, dp[i-coins[n]] + 1)

初始条件时dp[0] = [0]

对于i-coins[n]小于0的情况，则保持正无穷状态，表示不能拼出i，从而不做更新。

1. **for** (**int** i = 1;i <= amount;i++) {
2. dp[i] = INT\_MAX;
3. **for** (**int** j = 0;j < n;j++) {
4. **if** (i >= coins[j] && dp[i-coins[j]] != INT\_MAX &&
5. dp[i-coins[j]] +1 < dp[i]) {
6. dp[i] = dp[i-coins[j]] + 1;
7. }
8. }
9. }

670.预测能否胜利

使用动态规划来求解（区间动态规划），使用dp[i][j]表示从第i个位置到第j个位置，先取的用户比后取的用户多取的数目。假如A和B来取，如果A取了nums[i]，那么B需要在i+1 ~ j位置取，按照定义可以得到B比A多取了dp[i+1][j]，此时A也获取了价值nums[i]，所以A比B多取了nums[i] – dp[i+1][j]。同理可以分析得到，如果A从尾端开始取，A将比B多取nums[j] – dp[i][j-1]。综上，可以得到递推式：

dp[i][j] = max(nums[i] – dp[i+1][j], nums[j] – dp[i][j-1])

使用区间动态规划，区间的长度从1到len(j = i + len):

1. **for** (**int** len = 1;len < n;len ++) {
2. **for** (**int** i = 0;i + len < n;i++) {
3. **int** j = i + len;
4. dp[i][j] = max(nums[i] - dp[i+1][j], nums[j] - dp[i][j-1]);
5. }
6. }

671.循环单词

这个题目关键点在于判断两个单词是否是循环单词，一个巧妙的方法是，把其中一个单词复制一次，组成一个新的长单词（重复出现两次）。然后在这个长单词中寻找另一个单词，如果能找到，说明是循环单词，否则不是。（寻找可以用如下方式）

temp.find(target) != string::npos;

然后逐个比较即可判断，注意使用一个数组记录已经比较成功过的单词的位置，避免再次计算（出现的类别初始化为数组长度，每次比较成功一次就减一）。

676.解码方式II

比较复杂的字符串问题，要分多种情况进行讨论。

1. 初始化dp[0]=1，但是如果第一个字符是\*，dp[1] = 9，如果s[0] > 0 && s[0] < 9，dp[1] = 1。
2. 遍历字符串，如果当前字符是数字，就判断当前字符的前一个位置，如果俩个组合小于26，就让十位等于1。如果前一个位置是\*，则判断当前位置如果小于等于6 则second = 2，否则等于1。如果当前位置是字符\* ，first= 9，则判断前一个位置.最后运算。

677.大岛的数量

解法类似于433.岛屿的个数，不同的是使用一个全局变量记录当前岛屿的面积即可。

678.最短回文串

首先从原字符串第一个字符开始向后寻找，找到最长的回文串，然后把这个串后面的字符串倒序加到原字符串前面就可以得到最短回文串。

681.缺失的第一个素数

首先，如果最小的素数不是2，则缺失的第一个素数为2。然后分两种情况处理：

1. 从3到数组中的最大值循环，找到第一个不能被数组中所有的数整除的数，就是缺失的第一个素数。
2. 如果素数不在数组范围内，则从数组最大值开始，找到第一个不能被数组中所有的数整除的数。

683.单词拆分III

使用传统的回溯法只能过80%的测试用例，方法就是传统的逐步放置的方法。

684.缺少的字符串

对两个字符串使用空格进行字符串分割。然后把较短的字符串作为字典，对较长的字符串进行查询，找到未出现的字符串返回即可。

688.单词表示数字

把1~10，11~19，20~90作为字典存储，然后对三位做一个解析，之后把整个数字每三位单独处理，然后加上合适的单位即可。

689.两数之和-BST版本

不能使用额外的存储空间，使用中序遍历，在遍历过程中，在树中寻找目标值与根节点的差值，如果能够找到则饭回相应的结果。

690.阶乘

使用一个数组来记录阶乘过程中的结果，每次乘法只需要计算最后一位和当前乘数的执行结果，对于当前的结果，需要把结果按位分割，然后根据进位更新不同的位数。（使用字符串进行大数乘法依然会超时）。

691.恢复二叉搜索树

题目要求不能使用额外的空间，因为在当前的二叉搜索树中只有一次节点交换，因而会产生两个逆序对（逆序对中的元素不能重复计算）。通过分析可知，在当前情况下，恢复这个二叉搜索树只需要把第一个逆序对的第一个元素和第二个逆序对的第二个元素进行交换即可完成二叉搜索树的恢复。

在实现过程中，通过中序遍历来做，记录遍历当前节点的前一个节点，记录第一个逆序对的第一个元素和第二个逆序对的第二个元素，最后实现交换即可。

1. **void** inOrder(TreeNode \* root) {
2. **if** (!root) {
3. **return**;
4. }
5. inOrder(root->left);
6. **if** (pre != NULL && pre->val > root->val) {
7. **if** (p1 == NULL) {
8. p1 = pre;
9. }
10. p2 = root;
11. }
12. pre = root;
13. inOrder(root->right);
14. }

692.滑动窗口内唯一元素数目

初始化时，使用map记录第一个滑动窗口中对应的元素和元素出现的次数，统计只出现一次的元素数目。

在滑动过程中，更新map的信息，同时更新元素出现的数目，从而减少了不必要的重复计算，最终统计出结果。

696.课程表III

首先按照课程的截至时间的升序排序（越早结束的课程越要提前上），之后使用一个变量time来记录当前的时间。使用最小堆来放置已经安排的课程时间（时间越长的课程越容易被淘汰）。

对于每一个课程，如果当前时间加上课程时长未超过课程的截至时间，则更新当前的时间，并把课程时长放入优先队列中。

如果加上当前的课程视场超过了课程的截至时间，同时此课程时长小于堆顶的课程时长，那么将堆顶元素推出，放入此课程时长，同时更新当前的时间。

最终优先级队列的长度就是能进行的课程数目。

697.判断是否为平方数之和

暴力搜索的方式会超时。

首先使用tmp记录目标数的平方根，如果该数为完全平方数，则返回true。

在此基础上，我们使用map记录从1~tmp中的完全平方数。

对于i从1~tmp，如果在map中，可以找到（目标数n-i\*i），则说明可以表示成平方数之和。

698.数组中最大的差值

对整个数组从前向后遍历，对于每一子数组，记录子数组中对应的最小值min\_value和最大值max\_value，同时使用两个变量记录此数组之前的最小值pre\_min和最大值pre\_max。最大差值为：max(abs(max\_value – pre\_min)，abs(pre\_max – min\_value))。

699.判断K个素数之和

利用哥德巴赫猜想得出，任何大于2的偶数可以分解为两个质数的和。

由于最小的素数为2，当n<2k时，不可能实现，返回false；

当n为偶数，

若k也为偶数（k=2q），n>=4q，n可以分成q个大于等于4的数，根据猜想，每个数又可以表示为两个质数的和，故可以表示为k（即2q）个质数的和，返回true。

若k为奇数，为了保证n可以是k个数的和，必存在素数2，返回isSumOfKPrimes(n - 2, k - 1)。

当n为奇数，

若k为偶数，n不可能由偶数个奇数组成，必存在素数2，返回isSumOfKPrimes(n - 2, k - 1)。

若k为奇数，必然存在至少一个奇数，返回isSumOfKPrimes(n - 3, k - 1)

1. **bool** isPrime(**int** num) {
2. **if** (num < 2) {
3. **return** **false**;
4. }
5. **for** (**int** i = 2;i \* i < num;i++) {
6. **if** (num % i == 0) {
7. **return** **false**;
8. }
9. }
10. **return** **true**;
11. }
13. **bool** isSumOfKPrimes(**int** n, **int** k) {
14. **if** (n < 2\*k || k <= 0) {
15. **return** **false**;
16. }
17. **if** (k == 1) {
18. **return** isPrime(n);
19. }
20. **if** (n % 2 == 0) {
21. **if** (k % 2 == 0) {
22. **return** **true**;
23. }
24. **else** {
25. **return** isSumOfKPrimes(n - 2, k - 1);
26. }
27. }
28. **else** {
29. **if** (k % 2 == 0) {
30. **return** isSumOfKPrimes(n - 2, k - 1);
31. }
32. **else** {
33. **return** isSumOfKPrimes(n - 3, k - 1);
34. }
35. }
36. }

700.杆子分割

经典的动态规划问题，使用r[i]记录划分长度为i的竿子的最大值。

1. **int** cutting(vector<**int**> &prices, **int** n) {
2. **int** m = prices.size();
3. **if** (n <= 0 || m == 0) {
4. **return** 0;
5. }
6. vector<**int**> record(n+1, 0);
7. record[0] = 0;
8. **for** (**int** i = 1;i <= n;i++) {
9. **int** temp = INT\_MIN;
10. **for** (**int** j = 1; j <= i; j++) {
11. temp = max(temp, prices[j-1] + record[i - j]);
12. }
13. record[i] = temp;
14. }
15. **return** record[n];
16. }

701．修剪二叉搜索树

根据给定的范围修剪二叉搜索树，采用递归的方式，如果当前节点为空则直接返回，否则对于当前节点的值进行判断。如果当前节点的值小于区间右端点则直接修剪左子树，如果当前节点的值大于区间左端点，则修剪右子树。如果值在区间之中，则递归修剪（做右子树等于修改的结果）。

702．连接两个字符串中的不同字符

使用两个map生成两个单词的字典，然后找到不重复的字符拼接即可。

703. 折叠数组

每次折叠，生成一个新的二维数组，行数多一倍，列数变成原来的一半，根据这个特点，把从左到右和从右到左两个操作分开，对数组进行处理，最后拼装即可。

704. 电灯切换II

使用sting0,1来模拟开关的操作（可以使用二进制），然后对于每一步都有四种不同的操作。我使用的递归的方式，同时对于相同步数的操作，如果之前出现过的结果就不再向下递归，进行剪枝。最后一个测试用例空间超限。

可以用数学方法直接推导得出。

1. **int** flipLights (**int** n, **int** m) {
2. **if**(n==0){
3. **return** 0;
4. }
5. **if**(m==0){
6. **return** 1;
7. }
8. **if**(n==1){
9. **return** 2;
10. }
11. **if**(n==2){
12. **if**(m==1){
13. **return** 3;
14. }
15. **return** 4;
16. }
17. **if**(m==1){
18. **return** 4;
19. }
20. **if**(m==2){
21. **return** 7;
22. }
23. **return** 8;
24. }

706.二进制手表

首先把可能出现的数字0~60每个数有几个1使用dict存下来。计算每个数中1的个数的方法不要忘记！然后循环找出满足要求的即可。（就是两个数二进制表示1的个数）。

717.二叉树具有相同值的最长路径

使用一个辅助数组adj[i][j]存储和节点i相连的有哪些边。同时使用一个数组还记录哪些边被访问过，接着使用深度遍历的方法根据adj数组查找左右最长路径，记录左右最长路径和的最大值，同时返回二者之中的最大值。

718.重复子串

分两种情况，如果目标串的长度小于源串，直接判断目标串是否为源串的子串。否则从头至尾寻找，先从头开始找到第一个完整源串，对于前边的部分，判断是否为源串的子串，然后逐次向后寻找，直到末尾，注意判断最后剩下的子串是否是源串的子串。