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。

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]

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。

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

#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。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。