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。