- 10. Regular Expression Matching
- 97. Interleaving String
- Paint House
  - 931. Minimum Falling Path Sum
  - 265. Paint House II
  - 63. Unique Paths II
- Matrix多维度DP
  - 221. Maximal Square
  - 1277. Count Square Submatrices with All Ones
  - 1292. Maximum Side Length of a Square with Sum Less than or Equal to Threshold
  - 562. Longest Line of Consecutive One in Matrix

BackTracking
- 17. Letter Combinations of a Phone Number
- 784. Letter Case Permutation
- 131. Palindrome Partitioning
- 51. N-Queens
- 301. Remove Invalid Parentheses
- 320. Generalized Abbreviation
- 1087. Brace Expansion
- 489. Robot Room Cleaner

Tree
- Binary Tree
  - 250. Count Univalue Subtrees
  - 437. Path Sum III
  - 124. Binary Tree Maximum Path Sum
  - 99. Recover Binary Search Tree
  - 1110. Delete Nodes And Return Forest
- Binary Search Tree
  - 450. Delete Node in a BST

Binary Search
- 33. Search in Rotated Sorted Array
- 81. Search in Rotated Sorted Array II
- 162. Find Peak Element
- 287. Find the Duplicate Number
- 658. Find K Closest Elements
- 702. Search in a Sorted Array of Unknown Size

LinkedList

- 209. Minimum Size Subarray Sum
- 76. Minimum Window Substring

Monotonic Queue/Stack
- 239. Sliding Window Maximum
- 739. Daily Temperatures
- 862. Shortest Subarray with Sum at Least K
- 84. Largest Rectangle in Histogram

Greedy
- 45. Jump Game II
- 53. Maximum Subarray
- 134. Gas Station
- 517. Super washing machines

Two Pointers
- 475. Heaters
- 42. Trapping Rain Water
- 11. Container With Most Water
- 15. 3Sum
- Cyclic Sort swap

Divide and Conquer
- 241. Different Ways to Add Parentheses
- 53. Maximum Subarray

Graph
- DFS
- BFS
- 200. Number of Islands
- 286. Walls and Gate
- 130. Surrounded Regions
- 269. Alien Dictionary
- 994. Rotting Oranges
- 417. Pacific Atlantic Water Flow
- 127. Word Ladder
- 126. Word Ladder II

Topological Sort
- 207. Course Schedule
- 210. Course Schedule II
- 269. Alien Dictionary

- UnionFind
  - 130. Surrounded Regions
- Subarray Sum
  - 325. Maximum Size Subarray Sum Equals k
  - 560. Subarray Sum Equals K
  - 523. Continuous Subarray Sum
  - 974. Subarray Sums Divisible by K
- Trie
  - 212. Word Search II
  - 642. Design Search Autocomplete System
  - 1032. Stream of Characters
- KMP
  - 214. Shortest Palindrome
- Design
  - 146. LRU Cache
  - 155. Min Stack
  - 158. Read N Characters Given Read4 II - Call multiple times
  - 173. Binary Search Tree Iterator
  - 297. Serialize and Deserialize Binary Tree
  - 449. Serialize and Deserialize BST
  - 981. Time Based Key-Value Store
- 实现Trick
  - Brute Force
    - 681. Next Closest Time
    - 939. Minimum Area Rectangle
    - 963. Minimum Area Rectangle II
  - Comparator
    - 179. Largest Number
    - 937. Reorder Data in Log Files
    - 354. Russian Doll Envelopes
  - Using HashMap/HashSet
    - 652. Find Duplicate Subtrees
    - 138. Copy List with Random Pointer
    - 133. Clone Graph
  - Bitwise Operator
  - String indexOf, replace, replaceAll methods

929. Unique Email Addresses

String, array多重循环

228. Summary Ranges

163. Missing Ranges

1291. Sequential Digits

482. License Key Formatting

DP

Knapsack

0/1 Knapsack

要求从集合里选子集(每个item只能用一次)，满足一个限制条件 (<=capacity, ==sum)
问题一般3种：
有没有满足条件的子集，....dp[] || dp[]....
有多少个满足条件的子集， .....dp[] + dp[]..... 初始化一般是1不是0
在满足条件的子集里，求min, max， Math.max(....dp[]，.....dp[])

遍历 array 的循环 放在外层比较安全（对于化简过的 1 维）
未化简的 2 维哪个 index 放内层都可以，但是Combination Sum IV这种求解的数量的除外，遍历 array 的循环放在内层

Dp是2 dimension, 可以化简成1 dimension
For loop is always 2层

集合对应2个array：weight[], profits[]
限制total weight <= capacity, 求各种子集组合能得到的 max profit

思路：
从第一个item有几种选择入手，helper(weight[], profits[], capacity,index)代表从第index个item到最后，在capacity的限制条件下能得到的max profit。最终结果就是helper(weight[], profits[], capacity, 0)
在helper method里，每个item只有choose，skip两种选择，用profit1,profit2表示，最终结果就是Max(profit1, profit2)，profit1, profit2分别可用helper的递归表示

Bottom up思想，dp[i][c]表示 for first i items, under constraints capacity, what is the max profit
最终结果就是dp[n- 1][capacity]

For each item, 只有choose，skip两种选择
dp[i][c] = max(dp[i - 1][c], profit[i] + dp[i - 1][c - weight[i]])
注意 if (weights[i] <= c) 不是 <= capacity

```
int[][] dp = new int[n][capacity + 1];
初始化第一行和第一列
for (int i = 1; i < n; i++) {
    for (int c = 1; c <= capacity; c++) {
      int profit1 = 0, profit2 = 0;
         if (weights[i] <= c) {
            profit1 = profits[i] + dp[i - 1][c - weights[i]];
            profit2 = dp[i - 1][c];
            dp[i][c] = Math.max(profit1, profit2);
         } else {
            dp[i][c] = dp[i - 1][c];
         }
```

2纬dp化简到1纬dp，可简化initation步骤
dp[c] = max(dp[c], profit[i] + dp[c - weight[i]),
C的循环从右到左

```
int[] dp = new int[capacity + 1];
初始化dp[0], c必须从右向左循环
for(int i = 0; i < n; i++) {
    for(int c = capacity; c > 0; c--) {
       if(weights[i] <= c)
          dp[c] = Math.max(dp[c], profits[i] + dp[c-weights[i]]);
```

Lintcode 92. Backpack

Given n items with size Ai, an integer m denotes the size of a backpack. How full can you fill this backpack?
Example
Example 1:
Input: [3,4,8,5], backpack size=10
Output: 9

Example 2:

Input: [2,3,5,7], backpack size=12

Output: 12

必须用 boolean[][] dp, 不能用 int[][] dp

dp 可以化简为 1 维

```java
public int backPack(int m, int[] A) {
    int len = A.length;
    // dp[i][j] for the first i elements, can we make sum of j
    boolean[] dp = new boolean[m + 1];
    dp[0] = true;

    for (int i = 1; i <= len; i++) {
        for (int sum = m; sum > 0; sum--) {
            if (A[i - 1] <= sum)
                dp[sum] = dp[sum - A[i - 1]] || dp[sum];
        }
    }

    for (int i = m; i >= 0; i--)
        if (dp[i]) return i;

    return -1;
}
```

416. Partition Equal Subset Sum

Given a non-empty array containing only positive integers, find if the array can be partitioned into two subsets such that the sum of elements in both subsets is equal.

boolean canPartition(int[] nums)

普通DP

```java
public boolean canPartition(int[] nums) {
    int sum = 0, len = nums.length;
    for (int n : nums) sum += n;
    if (sum % 2 != 0) return false;
    sum = sum / 2;

    boolean[][] dp = new boolean[len + 1][sum + 1];
    dp[0][0] = true;

    for (int i = 1; i <= len; i++) {
        for (int s = 0; s <= sum; s++) {
            if (nums[i - 1] <= s) dp[i][s] = dp[i - 1][s] || dp[i - 1][s - nums[i - 1]];
            else dp[i][s] = dp[i - 1][s];
        }
    }

    return dp[len][sum];
}
```

space 化简 DP

```java
public boolean canPartition2(int[] nums) {
    int sum = 0, len = nums.length;
    for (int n : nums) sum += n;
    if (sum % 2 != 0) return false;
    sum = sum / 2;

    boolean[] dp = new boolean[sum + 1];
    dp[0] = true;

    for (int i = 1; i <= len; i++) {
        for (int s = sum; s >= 0; s--) {
            if (nums[i - 1] <= s) dp[s] = dp[s] || dp[s - nums[i - 1]];
        }
    }

    return dp[sum];
}
```

494. Count of Subset Sum

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols + and -. For each integer, you should choose one from + and - as its new symbol.
Find out how many ways to assign symbols to make sum of integers equal to target S.
int findTargetSumWays(int[] nums, int sum)

求有多少个解的题，dp[0] 初始化成1

而且s的循环必须包括0

```
int[] dp = new int[sum + 1];
dp[0] = 1;
for (int i = 0; i < n; i++) {
    for (int s = sum; s >= 0; s--) {
        if (num[i] <= s) {
            dp[s] = dp[s] + dp[s - num[i]];
```

K sum

Given n distinct positive integers, integer k (k <= n) and a number target.

Find k numbers where sum is target. Calculate how many solutions there are?

Example

Example 1

Input:
List = [1,2,3,4]
k = 2
target = 5
Output: 2
Explanation: 1 + 4 = 2 + 3 = 5

Example 2

Input:
List = [1,2,3,4,5]
k = 3
target = 6
Output: 1
Explanation: There is only one method. 1 + 2 + 3 = 6

我们关注的维度有三个：

- 前 i 个元素，因为一个元素只能选一次；
- 选了 j 个元素，因为我们要求选 k 个数；
- 总和 sum，用于每次试图添加新元素时用来查询。

时间：O(len * k * target)，三重循环

```
public int kSum(int A[], int k, int target) {
    // int[i][j][s] : number of ways to get sum of s by picking
    //                j numbers from first i numbers
    int len = A.length;
    int[][][] dp = new int[len + 1][k + 1][target + 1];
    for (int i = 0; i <= len; i++) dp[i][0][0] = 1;

    for (int i = 1; i <= len; i++) {
        for (int j = 1; j <= k; j++) {
            for (int v = 0; v <= target; v++) {
                if (A[i - 1] <= v) {
                    dp[i][j][v] = dp[i - 1][j][v] + dp[i - 1][j - 1][v - A[i - 1]];
                } else {
                    dp[i][j][v] = dp[i - 1][j][v];
                }
            }
        }
    }

    return dp[len][k][target];
}
```

Unbounded Knapsack

跟0/1一样，(每个item能用多次)

```
int[][] dp = new int[n][capacity + 1];
// 初始化第一行和第一列
for (int i = 1; i < n; i++) {
    for (int c = 1; c <= capacity; c++) {
      int profit1 = 0, profit2 = 0;
         if (weights[i] <= c) {
            profit1 = profits[i] + dp[i][c - weights[i]];
```

```
            profit2 = dp[i - 1][c];
            dp[i][c] = Math.max(profit1, profit2);
        } else {
            dp[i][c] = dp[i - 1][c];
        }
```

化简1纬

```
int[] dp = new int[capacity + 1];
//初始化dp[0], c从左向右
for(int i = 0; i < n; i++) {
    for(int c = 0; c <= capacity; c++) {
       if(weights[i] <= c)
          dp[c] = Math.max(dp[c], profits[i] + dp[c-weights[i]]);
```

322. Coin Change

fewest number of coins that you need to make up that amount

```
public int coinChange(int[] coins, int amount) {
    int[] dp = new int[amount + 1];
    Arrays.fill(dp, amount + 1);
    dp[0] = 0;

    for (int coin : coins) {
        for (int sum = coin; sum <= amount; sum++) {
            dp[sum] = Math.min(dp[sum], dp[sum - coin] + 1);
        }
    }

    return dp[amount] == amount + 1 ? -1 : dp[amount];
}
```

518. Coin Change 2

compute the number of combinations that make up that amount
求combination的情况，用0个硬币组成sum=0也算是一种combination

注意：两个 for loop 不能换位置
因为相同的数不同的顺序，算同样的解。

```java
public int change(int amount, int[] coins) {
    int[] dp = new int[amount + 1];
    dp[0] = 1;

    for (int coin : coins) {
        for (int sum = coin; sum <= amount; sum++) {
            dp[sum] += dp[sum - coin];
        }
    }

    return dp[amount];
}
```

279. Perfect Squares

Given a positive integer n, find the least number of perfect square numbers (for example, 1, 4, 9, 16, ...) which sum to n.
int numSquares(int n)

```java
public int numSquares(int n) {
    int[] dp = new int[n + 1];
    Arrays.fill(dp, n + 1);
    dp[0] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j*j <= i; j++) {
            dp[i] = Math.min(dp[i], dp[i - j*j] + 1);
        }
    }

    return dp[n];
}
```

1155. Number of Dice Rolls With Target Sum

给定 d 个筛子，每个筛子面值1-f，问有多少种筛子组合 sum=target
dp[i][j]: how many ways can i dices sum up to j

```java
public int numRollsToTarget(int d, int f, int target) {
    int MOD = 1000000007;
    int[][] dp = new int[d + 1][target + 1];
    //how many ways can i dices sum up to j;
    dp[0][0] = 1;

    for (int i = 1; i <= d; i++) {
        for (int j = 1; j <= target; j++) {
            //if (j <= i * f) {
            for (int k = 1; k <= f; k++) {
                if (j - k >= 0) dp[i][j] = (dp[i][j] + dp[i - 1][j - k]) % MOD;
            }
            //}
        }
    }
    return dp[d][target];
}
```

377. Combination Sum IV

Given an integer array with all positive numbers and no duplicates, find the number of possible combinations that add up to a positive integer target.

不同顺序也算不同解
一般考虑用backtracking，这里只能用dp

这题特殊在相同的数不同的顺序也算不同解。
每次i都能取所有数，所以i放在内层for loop

```java
public int combinationSum4(int[] nums, int target) {
    // dp[sum] = number of ways to get sum
    int[] dp = new int[target + 1];
    // initialize, one way to get 0 sum with 0 coins
    dp[0] = 1;

    for (int sum = 1; sum <= target; sum++) {
        for (int num : nums) {
            if (num <= sum)
                dp[sum] += dp[sum - num];
        }
    }

    return dp[target];
}
```

Fibonacci

限制条件与顺序有关，每个位置跟之前几步有关系，状态方程就涉及之前几项
For loop可能是1层，2层(每项与之前几项有关系不定的情况)

198. House Robber

two adjacent houses cannot be broken together
determine the maximum amount of money you can rob
int rob(int[] nums)

Dp is 1 dimension, dp[i] is related to dp[i - 1] and dp[i - 2]
必须初始化 dp[0], dp[1]

```
for (int i = 2; i < len; i++) {
     dp[i] = Math.max(nums[i] + dp[i - 2], dp[i - 1]);
  }
```

space化简成 O(1),不需要初始化

```
public int rob1(int[] nums) {
    int cur = 0, pre = 0;
    for (int n : nums) {
        int tmp = cur;
        cur = Math.max(n + pre, cur);
        pre = tmp;
    }
    return cur;
}
```

213. House Robber II

首尾也算相邻，不能都rob
拆成2个 198 问题解

用 l , r 替代 0, len - 1

```java
public int rob2(int[] nums) {
    int len = nums.length;
    if (len == 0) return 0;
    if (len == 1) return nums[0];
    if (len == 2) return Math.max(nums[0], nums[1]);

    return Math.max(helper(nums, 0, len - 2), helper(nums, 1, len - 1));
}

private int helper(int[] nums, int l, int r) {
    int pre = 0, cur = 0;
    for (int i = l; i <= r; i++) {
        int temp = cur;
        cur = Math.max(nums[i] + pre, cur);
        pre = temp;
    }
    return cur;
}
```

Stock Problems

2个限制条件，允许的交易次数j，当前有没有股票
3纬dp
每天的choice有2个，根据当前有没有股票，选择rest, buy, sell之中的2个
Space可以化简

```java
int[][][] dp = new int[len][k + 1][2];
for (int j = 1; j <= k; j++) {
    dp[0][j][0] = 0;
    dp[0][j][1] = -prices[0];
}

for (int i = 1; i < len; i++) {
    for (int j = 1; j <= k; j++) {
        dp[i][j][0] = Math.max(dp[i - 1][j][0], dp[i - 1][j][1] + prices[i]);
        dp[i][j][1] = Math.max(dp[i - 1][j][1], dp[i - 1][j - 1][0] - prices[i]);
    }
}

return dp[len - 1][k][0];
```

45. Jump Game II

Each number represents your maximum jump length at that position. Return minimum number of jumps to get to end.
int jump(int[] nums)

Start, end: end是从start开始跳，能到达的index
Dp[i]: 从开头0跳到i，需要的最小步数
初始化为Infinity, dp[0] = 0
每个end index有k个选择，从k个可能的start跳过来，for loop求min
Dp[] is 1 dimension, dp[i] is related to dp[i- 1], dp[i-2], …. , dp[i - k], use inner loop

考虑往后跳的思路，内层for算完，dp[j] 只刷新了一边，外层for算完，刷新了n遍，每个dp都是最终结果

考虑从前面来的思路，内层for算完，dp[j] 就算完了
效率稍慢

```
public int jump45(int[] nums) {
    int len = nums.length;
    int[] dp = new int[len];
    Arrays.fill(dp, len + 1);
    dp[0] = 0;

    for (int i = 0; i < len; i++) {
        int start = i;
        int end = Math.min(i + nums[i], len - 1);
        for (int j = start; j <= end; j++) {
            dp[j] = Math.min(dp[j], dp[start] + 1);
        }
    }

    return dp[len - 1] == len + 1 ? -1 : dp[len - 1];
}
```

```java
public int jump(int[] nums) {
    int len = nums.length;
    int[] dp = new int[len];
    dp[0] = 0;
    for (int i = 1; i < len; i++) dp[i] = len;

    // 固定前面，往后跳
    for (int i = 0; i < len; i++) {
        for (int j = i + 1; j < len; j++) {
            if (i + nums[i] >= j) {
                dp[j] = Math.min(dp[j], dp[i] + 1); // 一样
            }
        }
    }

    // 固定后面，从前面来，效率稍慢
    for (int j = 0; j < len; j++) {
        for (int i = 0; i < j; i++) {
            if (i + nums[i] >= j) {
                dp[j] = Math.min(dp[j], dp[i] + 1); // 一样
            }
        }
    }

    return dp[len - 1];
}
```

1326. Minimum Number of Taps to Open to Water a Garden

给定int[] ranges，下标从 0 到 n。range[i]为 >= 0的 int，表示这个 index 的 tap 可以浇灌的长度
问最少需要开几个 tap 可以把 0 ~ n所有的土地(注意不是所有 index)都浇灌了，不可能返 回-1

DP：int[] dp = new int[len]; dp[i] 表示浇灌区间 [0,i] 所需要 的最小taps
dp 初始化为Integer.MAX_VALUE
遍历 taps，找到每个 tap 浇灌的范围 [start, end]，用 dp[start]（如果效）+ 1更新[start, end]里每个 index

```java
public int minTaps3(int[] ranges) {
    int len = ranges.length;
    int[] dp = new int[len]; // dp[i] 表示为了覆盖区间 [0,i] 所需要 的最小taps
    Arrays.fill(dp, len + 1); // 只有 len 个 taps，最多开 len 个，初始化成一个不可能的大数 len+1
    dp[0] = 0;

    for (int i = 0; i < len; i++) {
        int start = Math.max(i - ranges[i], 0);
        int end = Math.min(i + ranges[i], len - 1);
        for (int j = start; j <= end; j++) {
            dp[j] = Math.min(dp[j], dp[start] + 1);
        }
    }

    return dp[len - 1] == len + 1 ? -1 : dp[len - 1];
}
```

1024. Video Stitching

给定int[][] clips和 T，clip 是 int[]表示一个区间，问最少需要clips 里选几个区间能覆盖整个[0, T]

```java
public int videoStitching3(int[][] clips, int T) {
    int[] dp = new int[T + 1];
    Arrays.fill(dp, T + 2);
    dp[0] = 0;

    for (int i = 1; i <= T; i++) {
        for (int[] clip : clips) {
            int start = clip[0];
            int end = clip[1];
            if (i >= start && i <= end)
                dp[i] = Math.min(dp[i], dp[start] + 1);
        }
    }

    return dp[T] == T + 2 ? -1 : dp[T];
}
```

139. Word Break

给定String s, List<String> wordDict，问s能不能被分割开，分开的每个词都是wordDict里的，wordDict里的词没有 duplicate，一个词可以用多次

带 memo 的递归解法比较直观，base case 是 if (set.contains(s))return true;

(1)用 index i 把 s 分割成 firstHalf, secondHalf，如果 firstHalf 在 wordDic 里，secondHalf 递归返回 true，最终返回 true。

```
public boolean wordBreak(String s, List<String> wordDict) {
    return helper(s, new HashSet<>(wordDict), new HashMap<>());
}

private boolean helper(String s, Set<String> wordDict, Map<String, Boolean> memo)
    if (wordDict.contains(s)) return true;
    if (memo.containsKey(s)) return memo.get(s);

    // 用 substring 分割 String 时，index 最好从 1 开始
    for (int i = 1; i <= s.length(); i++) {
        String firstHalf = s.substring(0, i);
        if (wordDict.contains(firstHalf)) {
            String secondHalf = s.substring(i);
            if (helper(secondHalf, wordDict, memo)) {
                memo.put(s, true);
                return true;
            }
        }
    }

    memo.put(s, false);
    return false;
}
```

(2)遍历 wordDict，看 word 是不是 s 开头的某个substring，用s.startsWith(word判断。这样s 被分割成了 word 和 rest， 对 rest 调递归。

```java
public boolean wordBreak(String s, List<String> wordDict) {
    return helper(s, new HashSet<>(wordDict), new HashMap<>());
}

private boolean helper(String s, Set<String> wordDict, Map<String, Boolean> memo)
    if (wordDict.contains(s)) return true;
    if (memo.containsKey(s)) return memo.get(s);

    for (String word : wordDict) {
        if (s.startsWith(word)) {
            String rest = s.substring(word.length());
            if (helper(rest, wordDict, memo)) {
                memo.put(s, true);
                return true;
            }
        }
    }

    memo.put(s, false);
    return false;
}
```

140. Word Break II

给定String s, List<String> wordDict，在 s 里插入空格，变成一个 sentence，要求 sentence 的每个词都在 wordDict 里，wordDict 里的词可以用多次。
要求以List<String>返回 all such possible sentences.

```java
public List<String> wordBreak(String s, List<String> wordDict) {
    return helper(s, new HashSet<>(wordDict), new HashMap<>());
}

private List<String> helper(String s, Set<String> wordDict, Map<String, List<String>> memo)
    if (memo.containsKey(s)) return memo.get(s);

    List<String> res = new ArrayList<>();

    for (String word : wordDict) {
        if (s.equals(word)) res.add(word);
        else if (s.startsWith(word)) {
            String rest = s.substring(word.length());
            List<String> subList = helper(rest, wordDict, memo);
            for (String sub : subList) res.add(word + " " + sub);
        }
    }

    memo.put(s, res);
    return res;
}
```

300. Longest Increasing Subsequence

Given an unsorted array of integers, find the length of longest increasing subsequence.
Example:
Input: [10,9,2,5,3,7,101,18]
Output: 4
Explanation: The longest increasing subsequence is [2,3,7,101], therefore the length is 4.
int lengthOfLIS(int[] nums)

遍历所有 j > i pairs, if nums[j] > nums[i], 有可能会延长longest increasing subsequence.
dp[j] 表示 以j结尾的longest increasing subsequence.
考虑从前面来的思路
外层for loop是j，内层for 是 i，对于固定的j，i遍历所有前面的index

```java
public int lengthOfLIS(int[] nums) {
    int len = nums.length;
    if (len == 0) return 0;
    int[] dp = new int[len];
    Arrays.fill(dp, 1);
    int maxLength = 1;

    // 固定前面，向后跳
    for (int i = 0; i < len; i++) {
        for (int j = i + 1; j < len; j++) {
            if (nums[j] > nums[i]) {
                dp[j] = Math.max(dp[j], dp[i] + 1);
                maxLength = Math.max(maxLength, dp[j]);
            }
        }
    }

    // 固定后面，从前面来
    for (int j = 0; j < len; j++) {
        for (int i = 0; i < j; i++) {
            if (nums[j] > nums[i]) {
                dp[j] = Math.max(dp[j], dp[i] + 1);
                maxLength = Math.max(maxLength, dp[j]);
            }
        }
    }

    return maxLength;
```

上面的时间复杂度 O(n^2)

更快的binarySearch nlog(n) 解法
可以用系统 binarySearch，也可以手写
size 比现在的 sortedArray的长度 大1
Arrays.binarySearch 是在 [0, size)的范围内 找 n
sortedArray找到了 n 就覆盖了，所以sortedArray里不可能有重复

```java
public int lengthOfLIS_300(int[] nums) {
    List<Integer> sorted = new ArrayList<>();
    for (int n : nums) {
        if (sorted.size() == 0 || n > sorted.get(sorted.size() - 1)) sorted.add(n);
        else {
            int index = Collections.binarySearch(sorted, n);
            if (index < 0) {
                index = -index - 1;
                sorted.set(index, n);
            } // if index >= 0, n is found in sorted, do nothing
        }
    }
    return sorted.size();
}
```

132. (MinCut)Palindrome Partitioning II

Return the minimum cuts needed for a palindrome partitioning of s
int minCut(String st)

1. use Palindrome pattern to get a isPalindrome[][] table
2. Use Fibonacci pattern to get cut[n - 1]
固定前面，考虑向后跳

```java
int[] cuts = new int[n];

Arrays.fill(cuts, n);
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        if (isPalindrome[i][j]) {
            if (i == 0) {
                cuts[j] = 0;
            } else {
                cuts[j] = Math.min(cuts[j], cuts[i - 1] + 1);
            }
        }
    }
}

return cuts[n - 1];
```

53. Maximum Subarray

有负数，求subarray的最大sum
public int maxSubArray(int[] nums)

Dp[i] means the max sum for all subarray which ends at i
用dp[i - 1]做if判断条件
Dp[i] 只与dp[i-1]有关，可以化简space 到 O(1)

```java
dp[0] = nums[0];
int res = nums[0];
for (int i = 1; i < len; i++) {
    if (dp[i - 1] < 0) dp[i] = nums[i];
    else dp[i] = dp[i - 1] + nums[i];
    res = Math.max(res, dp[i]);
}
return res;
```

32. Longest Valid Parentheses

括号题，考虑left, right的count

Given a string containing just the characters '(' and ')', find the length of the longest valid (well-formed) parentheses substring.
int longestValidParentheses(String s)

用到dp 2次，分别是fibonacci和knapsack pattern

```
for (int i = 0; i < n; i++) {
    if (s.charAt(i) == '(') {
        left++;
    } else if (right < left) {
        dp[i] = dp[i - 1] + 2;
        if (i - dp[i] >= 0) dp[i] += dp[i - dp[i]];
        res = Math.max(res, dp[i]);
        right++;
    } else {
        left = 0;
        right = 0;
    }
}
return res;
```

Palindrome

给1个String，找这个string内部满足的longest palindrome ….
I, j start from each end, move to middle,
Need to 初始化对角线 dp[i][i]
Result in dp[0][n-1]

516. Longest Palindromic Subsequence

find the longest palindromic subsequence's length in s
int longestPalindromeSubseq(String s)

Dp is longest palindrome length

```
for (int i = 0; i < n; i++)
    dp[i][i] = 1;

for (int i = n - 1; i >= 0; i--) {
    for (int j = i + 1; j < n; j++) {
        if (s.charAt(i) == s.charAt(j)) {
            dp[i][j] = 2 + dp[i + 1][j - 1];
        } else {
            dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);
        }
    }
}
return dp[0][n - 1];
```

5. Longest Palindromic Substring

find the longest palindromic substring in s
String longestPalindrome(String s)

Dp is boolean for palindrome substring problem

```
String res = s.substring(0, 1);

for (int i = 0; i < n; i++)
    dp[i][i] = true;

for (int i = n - 1; i >= 0; i--) {
    for (int j = i + 1; j < n; j++) {
        if (s.charAt(i) == s.charAt(j) && (j - i == 1 || dp[i + 1][j - 1])) {
            dp[i][j] = true;
            if (j - i + 1 > res.length()) {
                res = s.substring(i, j + 1);
            }
        }
    }
}
return res;
```

Two Strings (Longest Common subxxxx)

给2个string，找之前的关系，longest common/repeating subxxxxx

Dp[i][j] of size m+1, n+1
Meaning result of Longest subxxxxx between substring s1[0, i] and s2[0, j]

初始化第一行和第一列

Longest Subsequence: else之后可以不max，直接返回dp[m][n]
Longest Substring: else之后必须再max, 返回maxLength
Shortest Supersequence: else之后不能再取min, 必须返回dp[m][n]

1143. Longest Common Subsequence

Given two strings s1 and s2, return the length of their longest common subsequence.
Dp[i][j] means s1[0, i] and s2[0, j] 之中longest common subsequence的长度。

```
int[][] dp = new int[m + 1][n + 1];
int maxLength = 0;
for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
            dp[i][j] = 1 + dp[i - 1][j - 1];
        } else {
            dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
        }
        maxLength = Math.max(maxLength, dp[i][j]);
    }
}
return maxLength;
```

可以不用 maxLength，最终结果在dp[len1][len2]

Longest Common Substring

Same as Longest common subsequence, remove the else part in for loop

1062. Longest Repeating Substring

Given a string S, find out the length of the longest repeating substring
int longestRepeatingSubstring(String s)

跟longest common subxxxxx 一样，只有一个string, 变成自己跟自己比
注意比较charAt时，i，j不能相同

```java
public int longestRepeatingSubstring(String s) {
    int m = s.length();
    int[][] dp = new int[m + 1][m + 1];
    int maxLength = 0;
    for (int i = 1; i <= m; i++) {
        for (int j = i + 1; j <= m; j++) {
            if (s.charAt(i - 1) == s.charAt(j - 1)) {
                dp[i][j] = 1 + dp[i - 1][j - 1];
            }
            maxLength = Math.max(maxLength, dp[i][j]);
        }
    }
    return maxLength;
}
```

Shortest Common Supersequence

Given two sequences ‘s1’ and ‘s2’, write a method to find the length of the shortest sequence
S1, s2都是这个superstring的 subsequence

Dp[i][j] means s1[0, i] and s2[0, j] 的 shortest supersequence的长度
与1143相比，else之后不能再做一次min,
因为subxxx的max可能在中间，而superxxxx的min必须包括整个s1,s2, 结果必须在dp[m][n]

```
for (int i = 0; i <= m; i++) dp[i][0] = i;
for (int j = 0; j <= n; j++) dp[0][j] = j;

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (s1.charAt(i - 1) == s2.charAt(j - 1))
            dp[i][j] = 1 + dp[i - 1][j - 1];
        else
            dp[i][j] = 1 + Math.min(dp[i - 1][j], dp[i][j - 1]);
    }
}
return dp[m][n];
```

115. Distinct Subsequences

string S and a string T, count the number of distinct subsequences of S which equals T
int numDistinct(String s, String t)

Dp[i][[j] : means number of subsequences of substring s[0,i] that equals substring t[0, j]

```
for (int i = 0; i < m; i++) {
    dp[i][0] = 1;
}

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (i < j) continue;
        if (s.charAt(i - 1) == t.charAt(j - 1)) {
                        // keep s[i]     + delete s[i]
            dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];
        } else {
                    //delete s[i]
            dp[i][j] = dp[i - 1][j];
        }
    }
}
return dp[m][n];
```

72. Edit Distance

Given two words word1 and word2, find the minimum number of operations required to convert word1 to word2. Operations include delete, insert, replace
int minDistance(String s1, String s2)

注意初始化

```
// if s2 is empty, we can remove all of s1 to make it empty too
for (int i = 0; i <= m; i++) dp[i][0] = i;
// if s1 is empty, we can remove all of s2
for (int j = 0; j <= n; j++) dp[0][j] = j;

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (s1.charAt(i - 1) == s2.charAt(j - 1))
            dp[i][j] = dp[i - 1][j - 1];
        else
            //delete last char of s1
            dp[i][j] = 1 + Math.min(dp[i - 1][j],
                    //insert last at s1 or delete last of s2
                    Math.min(dp[i][j - 1],
                            //replace last of s1 or s2
                            dp[i - 1][j - 1])); } }
return dp[m][n];
```

10. Regular Expression Matching

'.' Matches any single character.
'*' Matches zero or more of the preceding element.
s could be empty and contains only lowercase letters a-z.
p could be empty and contains only lowercase letters a-z, and characters like . or *.
matching should cover the entire input string (not partial).
boolean isMatch(String s, String p)

特殊的初始化，状态转移方程 make this problem hard

```
boolean[][] dp = new boolean[m + 1][n + 1];
dp[0][0] = true;
for (int j = 2; j <= n; j += 2)
    if (p.charAt(j - 1) == '*' && dp[0][j - 2])
        dp[0][j] = true;

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        char curS = s.charAt(i - 1);
        char curP = p.charAt(j - 1);
        if (curS == curP || curP == '.')
            dp[i][j] = dp[i - 1][j - 1];

        if (curP == '*') {
            char preCurP = p.charAt(j - 2);
            if (preCurP != '.' && preCurP != curS) {
                // * have to mean 0 here
                dp[i][j] = dp[i][j - 2];
            } else {
                // * means       0               1                 more
                dp[i][j] = dp[i][j - 2] || dp[i - 1][j - 2] || dp[i - 1][j];
            } } } }

return dp[m][n];
```

97. Interleaving String

Given s1, s2, s3, find whether s3 is formed by the interleaving of s1 and s2.

3 个 String，i，j，k 3 个 index，意味着 3 维 DP。
因为 i + j == k，第 3 个维度 k 可以省去。
dp[i][j] is true if the first i chars from s1 and first j chars from s2 can interleave to produce first i + j chars of s3

```java
public boolean isInterleave(String s1, String s2, String s3) {
    int len1 = s1.length(), len2 = s2.length(), len3 = s3.length();
    if (len3 != len1 + len2) return false;
    if (len1 == 0) return s2.equals(s3);
    if (len2 == 0) return s1.equals(s3);

    // dp[i][j] is true if the first i chars from s1 and first j chars from s2
    // can interleave to produce first i + j chars of s3
    boolean[][] dp = new boolean[len1 + 1][len2 + 1];

    for (int i = 0; i <= len1; i++) {
        for (int j = 0; j <= len2; j++) {
            if (i == 0 && j == 0) dp[0][0] = true;
            else if (i == 0) {
                if (s2.charAt(j - 1) == s3.charAt(j - 1) && dp[0][j - 1])
                    dp[0][j] = true;
            } else if (j == 0) {
                if (s1.charAt(i - 1) == s3.charAt(i - 1) && dp[i - 1][0])
                    dp[i][0] = true;
            } else {
                char char1 = s1.charAt(i - 1);
                char char2 = s2.charAt(j - 1);
                char char3 = s3.charAt(i + j - 1);
                if (char1 == char3 && dp[i - 1][j])
                    dp[i][j] = true;
                if (char2 == char3 && dp[i][j - 1])
                    dp[i][j] = true;
            }
        }
    }

    return dp[len1][len2];
```

Paint House

每个item有多个选择，必须选择一个（这是跟Knapsack问题的区别），限制条件是相邻的item
输入参数是2纬array
最终结果是最后一行的min，而knapsack的最终结果在右下角

931. Minimum Falling Path Sum

The next row's choice must be in a column that is different from the previous row's column by at most one.
int minFallingPathSum(int[][] A)

最终结果是dp最后一行的min

```
int[][] dp = new int[m][n];
for (int[] arr : dp) Arrays.fill(arr, Integer.MAX_VALUE);

for (int j = 0; j < n; j++) dp[0][j] = A[0][j];

for (int i = 1; i < m; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = -1; k <= 1; k++) {
            if (j + k >= 0 && j + k < n) dp[i][j] = Math.min(dp[i][j], dp[i - 1][j + k]);
        }
        dp[i][j] = dp[i][j] + A[i][j];
    }
}
```

265. Paint House II

a row of n houses, each house can be painted with one of the k colors.
no two adjacent houses have the same color.
Find the minimum cost to paint all houses.
int minCostII(int[][] costs)

标准 DP

```java
public int minCostII2(int[][] costs) {
    int n = costs.length;
    if (n == 0) return 0;
    int k = costs[0].length;
    int[][] dp = new int[n][k];

    for (int i = 0; i < n; i++) {
        if (i == 0) {
            for (int j = 0; j < k; j++) dp[i][j] = costs[i][j];
        } else {
            for (int j = 0; j < k; j++) {
                int min = Integer.MAX_VALUE;
                for (int l = 0; l < k; l++) {
                    if (l != j) min = Math.min(min, dp[i - 1][l]);
                }
                dp[i][j] = costs[i][j] + min;
            }
        }
    }

    int res = Integer.MAX_VALUE;
    for (int j = 0; j < k; j++) res = Math.min(res, dp[n - 1][j]);
    return res;
}
```

计算每行的最小和次小 on the fly

```java
int min1 = 0, min2 = 0, index1 = -1;

for (int i = 0; i < m; i++) {
    int m1 = Integer.MAX_VALUE, m2 = Integer.MAX_VALUE, idx1 = -1;
    for (int j = 0; j < n; j++) {
        int cost = costs[i][j] + (j == index1 ? min2 : min1);

        if (cost < m1) {  // cost < m1 < m2
            m1 = cost;
            m2 = m1;
            idx1 = j;
        } else if (cost < m2) // m1 < cost < m2
            m2 = cost;
    }

    min1 = m1;
    min2 = m2;
    index1 = idx1;
}
return min1;
```

63. Unique Paths II

The robot can only move either down or right at any point in time. The robot is trying to reach the bottom-right corner of the grid
An obstacle and empty space is marked as 1 and 0 respectively in the grid.
How many unique paths would there be?

int uniquePathsWithObstacles(int[][] obstacleGrid)

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

标准 DP

```java
public int uniquePathsWithObstacles(int[][] obstacleGrid) {
    int m = obstacleGrid.length, n = obstacleGrid[0].length;
    int[][] dp = new int[m][n];

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (obstacleGrid[i][j] == 1) dp[i][j] = 0;
            else if (i == 0 && j == 0) dp[i][j] = 1;
            else if (i == 0) dp[i][j] = dp[i][j - 1];
            else if (j == 0) dp[i][j] = dp[i - 1][j];
            else dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
        }
    }

    return dp[m - 1][n - 1];
}
```

save space 写法，因为dp[i][j]只跟上一行 j 和这一行 j-1 有关，可以直接省去一个维度

如果 dp[i][j]只跟上一行 j 和上一行 j-1 有关，也可以省去一个维度，但是行的遍历要从右到左

```java
public int uniquePathsWithObstacles(int[][] obstacleGrid) {
    int m = obstacleGrid.length, n = obstacleGrid[0].length;
    int[] dp = new int[n];

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (obstacleGrid[i][j] == 1) dp[j] = 0;
            else if (i == 0 && j == 0) dp[j] = 1;
            else if (i == 0) dp[j] = dp[j - 1];
            else if (j == 0) dp[j] = dp[j];
            else dp[j] = dp[j] + dp[j - 1];
        }
    }

    return dp[n - 1];
}
```

Matrix多维度DP

221. Maximal Square

Given a 2D binary matrix filled with 0's and 1's, find the largest square containing only 1's and return its area.

int maximalSquare(char[][] matrix)

dp[i][j] 表示 以i,j为右下角的最大1 square的边长

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

```
for (int i = 0; i < m; i++) {
    if(matrix[i][0] == '1')  {
        dp[i][0] = 1;
        res = 1;
    }
}

for (int j = 1; j < n; j++) {
    if(matrix[0][j] == '1')  {
        dp[0][j] = 1;
        res = 1;
    }
}

for (int i = 1; i < m; i++) {
    for (int j = 1; j < n; j++) {
        if (matrix[i][j] == '1') {
            dp[i][j] = 1 + Math.min(dp[i - 1][j - 1], Math.min(dp[i][j - 1], dp[i - 1][j]));
            res = Math.max(res, dp[i][j]);
        }
    }
}
return res * res;
```

因为每个 dp 只跟本行和上一行有关，取 mod save space

```java
public int maximalSquare2(char[][] matrix) {
    if (matrix.length == 0) return 0;
    int m = matrix.length, n = matrix[0].length, res = 0;
    int[][] dp = new int[2][n];

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (i == 0) {
                if (matrix[i][j] == '1') {
                    dp[i][j] = 1;
                    res = Math.max(res, 1);
                } else dp[i][j] = 0;
            }

            else if (j == 0) {
                if (matrix[i][j] == '1') {
                    dp[i % 2][j] = 1;
                    res = Math.max(res, 1);
                } else dp[i % 2][j] = 0;
            }

            else {
                if (matrix[i][j] == '1') {
                    dp[i % 2][j] = 1 + Math.min(dp[(i - 1) % 2][j - 1],
                            Math.min(dp[i % 2][j - 1], dp[(i - 1) % 2][j]));
                    res = Math.max(res, dp[i % 2][j]);
                } else dp[i % 2][j] = 0;
            }
        }
    }
    return res * res;
```

1277. Count Square Submatrices with All Ones

Given a m * n matrix of ones and zeros, return how many square submatrices have all ones.
int countSquares(int[][] matrix)

边长为dp[i][j]的 one square里面含有 dp[i][j]个 one square

```java
for (int i = 1; i < m; i++) {
    for (int j = 1; j < n; j++) {
        if (matrix[i][j] == 1) {
            dp[i][j] = Math.min(dp[i - 1][j - 1], Math.min(dp[i][j - 1], dp[i - 1][j])) + 1;
            res += dp[i][j];
        }
    }
}
```

1292. Maximum Side Length of a Square with Sum Less than or Equal to Threshold

Given a m x n matrix mat and an integer threshold. Return the maximum side-length of a square with a sum less than or equal to threshold or return 0 if there is no such square.
int maxSideLength(int[][] matrix, int threshold)

计算preSum matrix
`preSum[i][j] = matrix[i - 1][j - 1] + preSum[i - 1][j] + preSum[i][j - 1] - preSum[i - 1][j - 1]`
以i, j 为右下角边长为len的square,所有元素和是
`preSum[i][j] - preSum[i - len][j] - preSum[i][j - len] + preSum[i - len][j - len]`

```
for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        preSum[i][j] = matrix[i - 1][j - 1] + preSum[i - 1][j] + preSum[i][j - 1] - preSum[i - 1][j - 1];
        if (i >= len && j >= len &&
                preSum[i][j] - preSum[i - len][j] - preSum[i][j - len] + preSum[i - len][j - len] <= threshold)
            res = len;
            len++;
        }
    }
}
```

562. Longest Line of Consecutive One in Matrix

Given a 01 matrix M, find the longest line of consecutive one in the matrix. The line could be horizontal, vertical, diagonal or anti-diagonal.
int longestLine(int[][] M)

有4种可能的方向取得最大，在原matrix基础上加一个纬度来记录这4个previous result

```java
public int longestLine(int[][] M) {
    if (M.length == 0) return 0;
    int m = M.length, n = M[0].length;
    int[][][] dp = new int[m][n][4];
    int res = 0;

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            if (M[i][j] == 1) {
                dp[i][j][0] = j > 0 ? 1 + dp[i][j - 1][0] : 1;
                dp[i][j][1] = i > 0 ? 1 + dp[i - 1][j][1] : 1;
                dp[i][j][2] = i > 0 && j > 0 ? 1 + dp[i - 1][j - 1][2] : 1;
                dp[i][j][3] = i > 0 && j + 1 < n ? 1 + dp[i - 1][j + 1][3] : 1;
                int a = Math.max(dp[i][j][0], dp[i][j][1]);
                int b = Math.max(dp[i][j][2], dp[i][j][3]);
                res = Math.max(res, Math.max(a, b));
            }
        }
    }

    return res;
```

BackTracking

```java
private void helper(String s, List<List<String>> res, List<String> temp, int start) {
      if (start == s.length()) {
            res.add(new ArrayList<>(temp));
      } else {
            for (int i = start; i < s.length(); i++) {
                  if (xxxxxxxxxxxx)) {
                        temp.add(s.substring(start, i + 1));
                        helper(s, res, temp, i + 1);
                        temp.remove(temp.size() - 1);
```

17. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent.
List<String> letterCombinations(String digits)

用Map先把数字和字母的对应关系存下来
用string做temp可以避免go back,
递归函数参数用digits.substring(1)，可以用digits.length()作为递归base case的判断条件

```
private void backtrack(List<String> res, Map<Character, String> phone, String tmp, String digits) {
    if (digits.length() == 0) res.add(tmp);
    else {
        String letters = phone.get(digits.charAt(0));
        for (char c : letters.toCharArray()) {
            backtrack(res, phone, tmp + c, digits.substring(1));
        }
    }
}
```

784. Letter Case Permutation

Given a string S, we can transform every letter individually to be lowercase or uppercase to create another string. Return a list of all possible strings we could create.
List<String> letterCasePermutation(String S)

递归函数参数用 char[]
Ca作为中间结果，手动覆盖，所以不用go back remove

```
private void backtrack(List<String> res, int i, char[] ca) {
    if (i == ca.length)
        res.add(new String(ca));
    else {
        if (Character.isLetter(ca[i])) {
            ca[i] = Character.toUpperCase(ca[i]);
            backtrack(res, i + 1, ca);
            ca[i] = Character.toLowerCase(ca[i]);
            backtrack(res, i + 1, ca);
        } else
            backtrack(res, i + 1, ca);
    }
}
```

131. Palindrome Partitioning

Return all possible palindrome partitioning of s.
List<List<String>> partition(String s)

```java
private void helper(String s, List<List<String>> res, List<String> temp, int start)
    if (start == s.length()) {
        res.add(new ArrayList<>(temp));
    } else {
        for (int i = start; i < s.length(); i++) {
            if (isPalindrome(s, start, i)) {
                temp.add(s.substring(start, i + 1));
                helper(s, res, temp, i + 1);
                temp.remove(temp.size() - 1);
            }
        }
    }
```

51. N-Queens

Given an integer n, return all distinct solutions to the n-queens puzzle.
List<List<String>> solveNQueens(int n)

```java
private void helper(char[][] board, int row) {
    if (row == board.length) {
        res.add(construct(board));
    } else {
        for (int j = 0; j < board[0].length; j++) {
            if (isValid(board, row, j)) {
                board[row][j] = 'Q';
                helper(board, row + 1);
                board[row][j] = '.';
```

301. Remove Invalid Parentheses

先统计需要删除的left, right括号数
同时需要统计keep下来的左右括号是否balance

```java
private void backTracking(String s, Set<String> res, String temp, int left, int right, int balance, int i) {
    if (left < 0 || right < 0 || balance < 0) return;

    if (i == s.length()) {
        if (balance == 0) res.add(temp);
    } else {
        char c = s.charAt(i);

        if ((c == '(' && left > 0)) { // delete
            backTracking(s, res, temp, left - 1, right, balance, i + 1);
        } else if ((c == ')' && right > 0)) {
            backTracking(s, res, temp, left, right - 1, balance, i + 1);
        }

        if (c != '(' && c != ')') {  // keep
            backTracking(s, res, temp + c, left, right, balance, i + 1);
        } else if (c == '(') {
            backTracking(s, res, temp + c, left, right, balance + 1, i + 1);
        } else {
            backTracking(s, res, temp + c, left, right, balance - 1, i + 1);
        }
    }
}
```

320. Generalized Abbreviation

Write a function to generate the generalized abbreviations of a word.
Input: "word"
Output:
["word", "1ord", "w1rd", "wo1d", "wor1", "2rd", "w2d", "wo2", "1o1d", "1or1", "w1r1", "1o2", "2r1", "3d", "w3", "4"]
List<String> generateAbbreviations(String word)

每一步2个选择，abbrev and no abbrev
Abbrev的情况需要一个parameter count记录till now有多少个abbrev letter

```java
private void helper(String word, List<String> res, String temp, int count, int i) {
    if (i == word.length()) {
        if (count > 0) temp += count;
        res.add(temp);
    } else {
        // Abbreviation with number
        helper(word, res, temp, count + 1, i + 1);

        if (count > 0) temp += count;
        // no Abbreviation, use char
        helper(word, res, temp + word.charAt(i), 0, i + 1);
    }
}
```

1087. Brace Expansion

A string S represents a list of words.
Each letter in the word has 1 or more options. If there is one option, the letter is represented as is. If there is more than one option, then curly braces delimit the options. For example, "{a,b,c}" represents options ["a", "b", "c"].
For example, "{a,b,c}d{e,f}" represents the list ["ade", "adf", "bde", "bdf", "cde", "cdf"].
Return all words that can be formed in this manner, in lexicographical order.
Input: "{a,b}c{d,e}f"
Output: ["acdf","acef","bcdf","bcef"]
String[] expand(String s)

String可以避免 backtracking go back这一步，但是 tep + c 必须作为 parameter传入 helper
用 i 做主循环的pointer, j指向 '}', {}中间用for loop + helper 只选一个数，然后把 j + 1传入 helper 作为新的 i

```java
public String[] expand(String s) {
    List<String> res = new ArrayList<>();
    helper(s, res, "", 0);
    return res.toArray(new String[0]);
}

private void helper(String s, List<String> res, String tmp, int i) {
    if (i == s.length()) {
        res.add(tmp);
        return;
    }
    char c = s.charAt(i);

    if (Character.isLetter(c)) {
        helper(s, res, tmp + c, i + 1);
    } else if (s.charAt(i) == '{') {

        List<Character> charList = new ArrayList<>();
        int j = i + 1;
        while (j < s.length() && s.charAt(j) != '}') {
            if (Character.isLetter(s.charAt(j))) {
                charList.add(s.charAt(j));
            }
            j++;
        }

        Collections.sort(charList);
        for (char d : charList) {
            helper(s, res, tmp + d, j + 1);
        }
    }
}
```

489. Robot Room Cleaner

机器人的行动模拟backtracking抽象的逻辑

```java
private static final int[][] directions = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

public void cleanRoom(Robot robot) {
    clean(robot, 0, 0, 0, new HashSet<>());
}

private void clean(Robot robot, int row, int col, int curDirection, Set<String> visited) {
    visited.add(row + " " + col);
    robot.clean();

    for (int i = 0; i < 4; i++) {
        int nx = row + directions[curDirection][0];
        int ny = col + directions[curDirection][1];
        if (!visited.contains(nx + " " + ny) && robot.move()) {
            clean(robot, nx, ny, curDirection, visited);
            // go back
            robot.turnRight();
            robot.turnRight();
            robot.move();
            robot.turnRight();
            robot.turnRight();
        }
        // change orientation.
        robot.turnRight();
        curDirection++;
        curDirection %= 4;
    }
}
```

Tree

Tree 的问题首先想recursion

Iterative inorder traversal

```
TreeNode p = root;
while (p != null || !stack.isEmpty()) {
      while (p != null) {
            stack.push(p);
            //preorder save
            p = p.left;
      }
```

```
        p = stack.pop();
        //inorder save
        p = p.right;
}
```

```
Deque<TreeNode> queue = new LinkedList<>();
queue.offer(root);

while (!queue.isEmpty()) {
        //new level
        int length = queue.size();
        for (int i = 0; i < length; i++) {
                TreeNode p = queue.poll();
                //save p.val to level
                if (p.left != null) queue.offer(p.left);
                if (p.right != null) queue.offer(p.right);
        }
```

Binary Tree

250. Count Univalue Subtrees

Given a binary tree, count the number of uni-value subtrees.
A Uni-value subtree means all nodes of the subtree have the same value.
int countUnivalSubtrees(TreeNode root)

Post-order遍历，helper method判断是否univalue subtree
主method调 helper method，丢掉返回值
利用helper method的遍历统计count，count设为class field

```java
private int count;

public int countUnivalSubtrees(TreeNode root) {
    isUnival(root);
    return count;
}

private boolean isUnival(TreeNode node) {
    if (node == null) return true;
    boolean left = isUnival(node.left);
    boolean right = isUnival(node.right);
    if (left && right) {
        if (node.left != null && node.val != node.left.val) return false;
        else if (node.right != null && node.val != node.right.val) return false;
        count++;
        return true;
    }
    return false;
}
```

437. Path Sum III

Find the number of paths that sum to a given value.
The path does not need to start or end at the root or a leaf, but it must go downwards
int pathSum(TreeNode root, int sum)

Recursion, Tree problem上来先考虑递归
因为path可以含root(helper method)，可以不含root(主method递归)
Helper method 设为 int pathSumFrom()，这也是个recursion method

```java
int pathSum(TreeNode root, int sum) {
    if (root == null) return 0;
    return pathSumFrom(root, sum) + pathSum(root.left, sum) + pathSum(root.right, sum);
}

private int pathSumFrom(TreeNode node, int sum) {
    if (node == null) return 0;
    return (node.val == sum ? 1 : 0)
            + pathSumFrom(node.left, sum - node.val) + pathSumFrom(node.right, sum - node.val);
}
```

124. Binary Tree Maximum Path Sum

Given a non-empty binary tree, find the maximum path sum.
A path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. 可以先向上再向下
int maxPathSum(TreeNode root)

```java
private int res = Integer.MIN_VALUE;

public int maxPathSum(TreeNode root) {
    maxSumFrom(root);
    return res;
}

private int maxSumFrom(TreeNode node) {
    if (node == null) return 0;
    int left = Math.max(0, maxSumFrom(node.left));
    int right = Math.max(0, maxSumFrom(node.right));
    res = Math.max(res, node.val + left + right);
    return node.val + Math.max(left, right);
}
```

99. Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.
Recover the tree without changing its structure.
void recoverTree(TreeNode root)

pre, curr 2个pointers
中序遍历，出现1次or2次相邻顺序不对，分别对应需要swap的node x and y

```java
TreeNode x = null, y = null, prev = null, curr = root;
while (curr != null || !stack.isEmpty()) {
    while (curr != null) {
        stack.push(curr);
        curr = curr.left;
    }
    curr = stack.pop();
    if (prev != null && curr.val < prev.val) {
        if (x == null) {
            x = prev;
            y = curr;
        } else {
            y = curr;
            break;
        }
    }
    prev = curr;
    curr = curr.right;
}
swap(x, y);
```

1110. Delete Nodes And Return Forest

Given the root of a binary tree, each node in the tree has a distinct value.
After deleting all nodes with a value in to_delete, we are left with a forest (a disjoint union of trees).
Return the roots of the trees in the remaining forest.
List<TreeNode> delNodes(TreeNode root, int[] to_delete)

Post-order traversal, 如果一个node被delete, 他的parent的左右link要改变，也就是把node自己设为null。同时这个node的children如果不为null，可以加入res
如果一个node不被delete, parent的link不用变，也就是这个node还是自己。这个node的children不用加入res

Parent link是否要改变(node自己是否设为null)这个信息需要往上传递，用后序遍历的返回值

```
    if (!deleteSet.contains(root.val)) res.add(root);
    dfs(root, deleteSet, res);
    return res;
}

private TreeNode dfs(TreeNode node, Set<Integer> deleteSet, List<TreeNode> res) {
    if (node == null) return null;
    node.left = dfs(node.left, deleteSet, res);
    node.right = dfs(node.right, deleteSet, res);

    if (deleteSet.contains(node.val)) {
        if (node.left != null) res.add(node.left);
        if (node.right != null) res.add(node.right);
        node = null;
    }
    return node;
```

Binary Search Tree

450. Delete Node in a BST

Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return the root node reference (possibly updated) of the BST.
Basically, the deletion can be divided into two stages:

1. Search for a node to remove.
2. If the node is found, delete the node.

Note: Time complexity should be O(height of tree).
TreeNode deleteNode(TreeNode root, int key)

先找到要delete的node
1. 如果node是leaf, 直接delete
2, 如果只有一个child, 用这个child replace 自己
3. 有2个children, 用rightSmallest(也就是successor) 代替自己, 然后递归root.right删掉 rightSmallest

```java
public TreeNode deleteNode(TreeNode root, int key) {
    if (root == null) return null;

    if (key < root.val) {
        root.left = deleteNode(root.left, key);
    } else if (key > root.val) {
        root.right = deleteNode(root.right, key);
    } else {
        if (root.left == null && root.right == null) return null;
        else if (root.left == null) {
            return root.right;
        } else if (root.right == null) {
            return root.left;
        } else {
            TreeNode rightSmallest = root.right;
            while (rightSmallest.left != null) rightSmallest = rightSmallest.left;
            root.val = rightSmallest.val;
            root.right = deleteNode(root.right, rightSmallest.val);
        }
    }
    return root;
}
```

Binary Search

```java
int binarySearch(int[] nums, int key) {
    int lo = 0, hi = nums.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key > nums[mid]) lo = mid + 1;
        else hi = mid - 1;
    }
    return lo;
}
```

上面的模板返回的是 >= key 里最小的num 的 index

1. key在 nums里
   返回值范围 [0, nums.length - 1]
   有重复num的情况，返回第一个 key = 2, (1, [2], 2, 2, 2, 5, 8, ...)
2. key不在 nums里
   返回值范围 [0, nums.length]

返回第一个＞key的数的index，也就是key应该插入的index

```
int binarySearch(int[] nums, int key) {
    int lo = 0, hi = nums.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (key >= nums[mid]) lo = mid + 1;
        else hi = mid - 1;
    }
    return lo;
}
```

上面的模板返回的是 > key 里最小的num 的 index

33. Search in Rotated Sorted Array

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.
(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).
You are given a target value to search. If found in the array return its index, otherwise return -1.
You may assume no duplicate exists in the array.
int search(int[] nums, int target)

Rotate之后，mid左右一边有序一边无序
如果target在有序的一边，排除无序的一边，否则排除有序的一边

```
int lo = 0, hi = nums.length - 1;
while (lo <= hi) {
    int mid = lo + (hi - lo) / 2;
    if (nums[mid] == target) return mid;

    if (nums[mid] < nums[hi]) { // [4 5 1 2 3]
        if (target > nums[mid] && target <= nums[hi]) lo = mid + 1;
        else hi = mid - 1;
    } else {                    // [3 4 5 1 2]
        if (target < nums[mid] && target >= nums[lo]) hi = mid - 1;
        else lo = mid + 1;
    }
}
return -1;
```

81. Search in Rotated Sorted Array II

You are given a target value to search. If found in the array return true, otherwise return false.
数有重复

最后的else hi--;
2种case：

1. lo == mid == hi, 如果此时还没找到解，直接返回false
2. mid != hi, 但是nums[mid] == nums[hi], hi--不会丢失解，mid还在search space

```
while (lo <= hi) {
    int mid = lo + (hi - lo) / 2;
    if (target == nums[mid]) return true;

    if (nums[mid] < nums[hi]) {          // 后半部有序
        if (target > nums[mid] && target <= nums[hi]) lo = mid + 1;
        else hi = mid - 1;
    } else if (nums[mid] > nums[hi]) {   // 前半部有序
        if (target < nums[mid] && target >= nums[lo]) hi = mid - 1;
        else lo = mid + 1;
    } else if (mid == hi) return false;
    else hi--;
}

return false;
```

162. Find Peak Element

A peak element is an element that is greater than its neighbors.
Given an input array nums, where nums[i] ≠ nums[i+1], find a peak element and return its index.
The array may contain multiple peaks, in that case returning the index to any one of the peaks is fine.
You may imagine that nums[-1] = nums[n] = -∞. 没有重复
int findPeakElement(int[] nums)

Lo < hi, 保证nums[mid + 1]不会越界
Nums[mid + 1] > nums[mid], mid不可能是解，mid + 1到右端至少有一个解
Nums[mid + 1] < nums[mid], mid可能是解，左端到mid至少有一个解

```
int lo = 0, hi = nums.length - 1;

while (lo < hi) {
    int mid = lo + (hi - lo) / 2;
    if (nums[mid + 1] > nums[mid]) lo = mid + 1;
    else hi = mid;
}
return lo;
```

287. Find the Duplicate Number

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Example 1: Input: [1,3,4,2,2] Output: 2

Example 2: Input: [3,1,3,4,2] Output: 3

nlog(n), 根据值的范围二分。

658. Find K Closest Elements

Given a sorted array, two integers k and x, find the k closest elements to x in the array. The result should also be sorted in ascending order. If there is a tie, the smaller elements are always preferred.
List<Integer> findClosestElements(int[] nums, int k, int x)

Assume we are taking A[i] ~ A[i + k -1].
We can binary research i
We compare the distance between x - A[mid] and A[mid + k] - x
If x - A[mid] > A[mid + k] - x,
it means A[mid + 1] ~ A[mid + k] is better than A[mid] ~ A[mid + k - 1],

```java
int lo = 0, hi = nums.length - k;

while (lo < hi) {
    int mid = (lo + hi) / 2;
    if (x - nums[mid] > nums[mid + k] - x)
        lo = mid + 1;
    else
        hi = mid;
}

while (res.size() < k) res.add(nums[lo++]);
return res;
```

702. Search in a Sorted Array of Unknown Size

确定hi, 指数增长

```
int hi = 1;
while (reader.get(hi) < target) {
      hi = hi * 2;
}
```

LinkedList

LinkedList问题首先想 recursion

In-place reverse (iteration)

遍历node cur, cur = cur.next, 用了temp做中间缓存
把每个cur接到newHead的尾部

```
        ListNode newHead = null, cur = head;
        while (cur != null) {
            ListNode tmp = cur.next;
            cur.next = newHead;
            newHead = cur;
            cur = tmp;
        }
```

```
        return newHead;
```

24. Swap Nodes in Pairs

递归

```
if (head == null || head.next == null) return head;
ListNode res = head.next;
head.next = swapPairs(head.next.next);
res.next = head;
return res;
```

Iterative

在原list上做改动，还是用原来的head来返回List的情况，（Iterative解法）一般要用dummy node

```
ListNode dummy = new ListNode(-1);
dummy.next = head;
ListNode prev = dummy;

while (head != null && head.next != null) {
    ListNode firstNode = head;
    ListNode secondNode = head.next;

    firstNode.next = secondNode.next;
    secondNode.next = firstNode;
    prev.next = secondNode;

    // prepare for next swap
    prev = firstNode;
    head = firstNode.next;
}

return dummy.next;
```

25. Reverse Nodes in k-Group

Example:
Given this linked list: 1->2->3->4->5
For k = 2, you should return: 2->1->4->3->5
For k = 3, you should return: 3->2->1->4->5
ListNode reverseKGroup(ListNode head, int k)

递归解法

先检查list长度有没有k，如果没有k长，作为recursion base case

如果有k长，递归，reverse前面k个，用template
template 有2个改动， newHead 初始化不是null，是后面要接的
while不是 ！= null, 是正好reverse k个

```
ListNode node = head;
int count = 0;
while (node != null && count != k) {
    node = node.next;
    count++;
}

if (count != k) return head;
else {
    ListNode cur = head;
    ListNode newHead = reverseKGroup(node, k);
    while (count-- > 0) {
        // 下面4行是reverse 模板
        ListNode tmp = cur.next;
        cur.next = newHead;
        newHead = cur;
        cur = tmp;
    }
    return newHead;
}
```

143. Reorder List

Given 1->2->3->4->5, reorder it to 1->5->2->4->3.
这个题很经典，寻找中间值、翻转和创建 list 都有了
思路: 第一步:寻找中间节点 第二步：翻转后半部分。非常精炼的模板，记牢！ 第三步：链接头结点和第二部分的头结点。 困难在于如何连接前半部分和后半部分，要思路清晰保留那些节点。 需要保留的节点包括每个节点的 next 以便下一步前进

```java
public void reorderList(ListNode head) {
    if(head == null) return;
    ListNode slow = head; // 以下寻找中间节点
    ListNode fast = head;
    while(fast.next != null && fast.next.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }
    ListNode pre = null; // 以下翻转后半部分
    ListNode curr = slow;
    while(curr != null){
        ListNode next = curr.next;
        curr.next = pre;
        pre = curr;
        curr = next;
    }
    ListNode walker = head;// 以下链接节点
    while(walker != null && pre != null){
        ListNode next = walker.next;
        ListNode preNext = pre.next;
        walker.next = pre;
        pre.next = next;
        walker = next;
        pre = preNext;
    }
```

Fast & Slow pointers

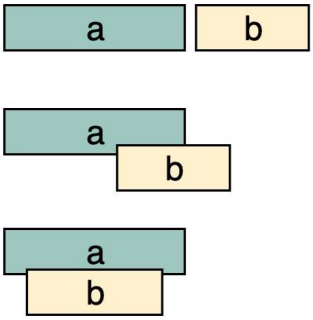
234. Palindrome Linked List

O(n) O(n)解法，把linked list转成 arraylist, 再用two pointers两遍夹
O(n) O(1)解法，fast slow pointer找到中点，reverse 后半，比较，再恢复后半
fast slow方法返回中点，奇数是中点，偶数是中间偏左
返回boolean后要继续回复后半段，所以不能马上return false， 而是while (result&& p1 != null)

```java
ListNode firstHalfEnd = endOfFirstHalf(head);
ListNode secondHalfStart = reverseList(firstHalfEnd.next);
ListNode p1 = head;
ListNode p2 = secondHalfStart;
boolean result = true;

while (result && p1 != null) {
    if (p1.val != p2.val) result = false;
    p1 = p1.next;
    p2 = p2.next;
}

firstHalfEnd.next = reverseList(secondHalfStart);
return result;
```

Intervals

2个intervals [a0, a1], [b0, b1]，一共6种情况
如果sort starting time了(a0 < b0)，有3种情况，其中overlapped有2种情况



56. Merge Intervals

Given a collection of intervals, merge all overlapping intervals.
public int[][] merge(int[][] intervals)

先sort, 一共3种情况
遍历intervals，跟merged结果的最后一个interval比较
第1种情况, interval加到 merged
2，3都是更新merged结果的最后一个interval

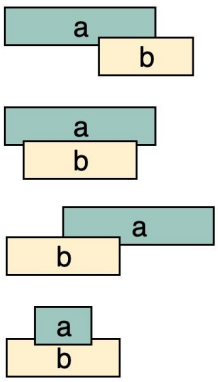
更新的是merged的最后一项

```java
public int[][] merge(int[][] intervals) {
    int[][] res = new int[][]{};
    if (intervals.length == 0) return res;
    Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

    ArrayList<int[]> merged = new ArrayList<>();

    for (int i = 0; i < intervals.length; i++) {
        if (merged.isEmpty() || intervals[i][0] > merged.get(merged.size() - 1)[1])
            merged.add(intervals[i]);
        else {
            int[] last = merged.get(merged.size() - 1);
            last[1] = Math.max(last[1], intervals[i][1]);
        }
    }

    return merged.toArray(new int[0][]);
}
```

57. Insert Interval

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).
You may assume that the intervals were initially sorted according to their start times.
int[][] insert(int[][] intervals, int[] newInterval)

遍历intervals，更新的是 toAdd
1. [toAdd] [inter[i]...]: toAdd设为interval[i]
2. [inter[i]...] [toAdd]：直接加入interval[i]
3-6.



```java
List<int[]> res = new ArrayList<>();
int[] toAdd = newInterval;

for (int i = 0; i < intervals.length; i++) {
    // case 1
    if (toAdd[1] < intervals[i][0]) {
        res.add(toAdd);
        toAdd = intervals[i];
    }
    // case 3-6
    else if (toAdd[0] <= intervals[i][1])
        toAdd = new int[]{Math.min(intervals[i][0], toAdd[0]),
                Math.max(intervals[i][1], toAdd[1])};
    // case 2
    else res.add(intervals[i]);
}
res.add(toAdd);

return res.toArray(new int[res.size()][]);
```

986. Interval List Intersections

Given two lists of closed intervals, each list of intervals is pairwise disjoint and in sorted order. Return the intersection of these two interval lists.
int[][] intervalIntersection(int[][] arr1, int[][] arr2)

两个intervals overlap的判断标准
left = max(a0, b0)
right = min(a1, b1)
left <= right : Overlap
left > right : NO overlap

双指针 i, j
如果overlap, 把overlap加入res, else do nothing
arr1[i], arr2[j] 谁先结束，谁向后++

```java
int m = arr1.length, n = arr2.length;
int i = 0, j = 0;

while (i < m && j < n) {
    int left = Math.max(arr1[i][0], arr2[j][0]);
    int right = Math.min(arr1[i][1], arr2[j][1]);
    if (left <= right) {
        res.add(new int[]{left, right});
    }

    if (arr1[i][1] < arr2[j][1]) i++;
    else j++;
}

return res.toArray(new int[0][]);
```

253. Meeting Rooms II

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.
int minMeetingRooms(int[][] intervals)

用一个MinPQ存结果，pq里放ending time
遍历intervals, 如果intervals[i] start和pq里最早结束不overlap，则不需要新的room，(pq.poll() pq.offer())
如果 overlap, 需要新的room (pq.offer)
最终min room就是 pq.size(), size只增不减

```java
PriorityQueue<Integer> rooms = new PriorityQueue<>();
rooms.offer(intervals[0][1]);

for (int i = 1; i < intervals.length; i++) {
    if (intervals[i][0] >= rooms.peek()) {
        rooms.poll();
    }
    rooms.offer(intervals[i][1]);
}

return rooms.size();
```

Heap

Two Heap: Find Median

295. Find Median from Data Stream

Design a data structure that supports the following two operations:

- void addNum(int num) - Add a integer number from the data stream to the data structure.
- double findMedian() - Return the median of all elements so far.

保持small, large两个 Heap,
small存所有小数，是一个MaxHeap
large存所有大数，是一个MinHeap
保证每次addNum后， small和large size一样大，或者small比large多一个
1. addNum之前size一样，为了add之后small多一个，往对方也就是large里offer,然后再从large里poll出来offer到small。这样才能保证small里的数都 <= large里的数
2. addNum之前small多一个，为了add之后size一样，往small里offer, 然后再从small里poll出来offer到large

```
private PriorityQueue<Integer> small = new PriorityQueue<>(Collections.reverseOrder());
private PriorityQueue<Integer> large = new PriorityQueue<>();

public double findMedian() {
    if (small.size() == large.size())
        return (small.peek() + large.peek()) / 2.0;
    else
        return small.peek();
}

public void addNum(int num) {
    if (small.size() == large.size()) {
        large.offer(num);
        small.offer(large.poll());
    } else {
        small.offer(num);
        large.offer(small.poll());
    }
}
```

480. Sliding Window Median

给定int[] nums, int k，表示在 nums 里有一个长度为 k 的 window，这个window 依次往右移动。要求以double[]的形式返回每个 window 的 median

two Heaps 来解 median 问题，一个maxHeap 存所有较小的数，一个 minHeap存所有较大的数
每次要 add 新数，先跟 maxHeap.peek()比，加完后rebalanceHeaps()
每次要 delete 数，要找这个数在哪个 heap，也是跟maxHeap.peek()比，delete完之后 rebalanceHeaps()

```java
public double[] medianSlidingWindow(int[] nums, int k) {
    PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
    PriorityQueue<Integer> minHeap = new PriorityQueue<>();
    double[] result = new double[nums.length - k + 1];

    for (int i = 0; i < nums.length; i++) {
        if (maxHeap.size() == 0 || nums[i] <= maxHeap.peek()) {
            maxHeap.add(nums[i]);
        } else {
            minHeap.add(nums[i]);
        }
        balanceHeaps(maxHeap, minHeap);

        if (i >= k - 1) {
            if (k % 2 == 0) {
                result[i - k + 1] = maxHeap.peek() / 2.0 + minHeap.peek() / 2.0;
            } else {
                result[i - k + 1] = maxHeap.peek();
            }

            int elementToBeRemoved = nums[i - k + 1];
            if (elementToBeRemoved <= maxHeap.peek()) maxHeap.remove(elementToBeRemoved);
            else minHeap.remove(elementToBeRemoved);
            balanceHeaps(maxHeap, minHeap);
        }
    }
    return result;
}
```

Top K elements

求largest k用 minHeap, 求smallest k用 maxHeap
time complexity 一般从 brute force的Nlog(N) 优化到 Nlog(K)

```
for (int n : nums) {
    pq.offer(n);
    if (pq.size() > k) pq.poll();
}
```

215. Kth Largest Element in an Array

Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.
int findKthLargest(int[] nums, int k)

```
public int findKthLargest(int[] nums, int k) {
    PriorityQueue<Integer> pq = new PriorityQueue<>();
    for (int n : nums) {
        pq.offer(n);
        if(pq.size() > k) pq.poll();
    }
    return pq.peek();
}
```

347. Top K Frequent Elements

Given a non-empty array of integers, return the k most frequent elements.
Example 1:
Input: nums = [1,1,1,2,2,3,3], k = 2
Output: [1,2]
List<Integer> topKFrequent(int[] nums, int k)

自定义Comparator, 按frequency升序排，也就是minHeap的效果

```java
public List<Integer> topKFrequent(int[] nums, int k) {
    Map<Integer, Integer> count = new HashMap<>();
    for (int n : nums) count.put(n, count.getOrDefault(n, 0) + 1);

    PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> count.get(a) - count.get(b));

    for (int n : count.keySet()) {
        pq.offer(n);
        if (pq.size() > k) pq.poll();
    }

    return new ArrayList<>(pq);
}
```

K-way merge

time complexity 一般从 brute force的Nlog(N) 优化到 Nlog(K)

23. Merge k Sorted Lists

Merge *k* sorted linked lists and return it as one sorted list. Analyze and describe its complexity.
ListNode class was defined
ListNode mergeKLists(ListNode[] lists)

用minHeap，初始化为size = k，包含所有heads
while (!pq.isEmpty()) poll一个，offer一个(如果还有)
注意resHead, resTail，add to the end of res的 trick

```java
public ListNode mergeKLists(ListNode[] lists) {
    PriorityQueue<ListNode> pq = new PriorityQueue<>((n1, n2) -> n1.val - n2.val);

    for (ListNode root : lists) {
        if (root != null) pq.add(root);
    }

    ListNode resHead = null, resTail = null;
    while (!pq.isEmpty()) {
        ListNode node = pq.poll();
        if (resHead == null) {
            resHead = resTail = node;
        } else {
            // add node to the end of res
            resTail.next = node;
            resTail = resTail.next;
        }

        if (node.next != null) pq.add(node.next);
    }

    return resHead;
```

Merge k Sorted Arrays

要用private class ArrayNode记录每个element 从那个arrayId来的
用 iterator 取得下一个array 的下一个 element

```java
private static class ArrayNode {
    public Integer value;
    public Integer arrayId;

    public ArrayNode(Integer value, Integer arrayId) {
        this.value = value;
        this.arrayId = arrayId;
    }
}
```

```java
for (List<Integer> array : sortedArrays) {
    iters.add(array.iterator());
}

for (int i = 0; i < sortedArrays.size(); i++) {
    if (iters.get(i).hasNext()) {
        pq.add(new ArrayNode(iters.get(i).next(), i));
    }
}

List<Integer> result = new ArrayList<>();
while (!pq.isEmpty()) {
    ArrayNode node = pq.poll();
    result.add(node.value);
    if (iters.get(node.arrayId).hasNext()) {
        pq.add(new ArrayNode(iters.get(node.arrayId).next(), node.arrayId));
    }
}

return result;
```

Stack

394. Decode String

Given an encoded string, return its decoded string.
s = "3[a]2[bc]", return "aaabcbc".
s = "3[a2[c]]", return "accaccacc".
s = "2[abc]3[cd]ef", return "abcabccdcdcdef".
String decodeString(String s)

带nested的括号，用 stack 或者 recursion
recursion更直观，普通的recursion要涉及 index
如果用一个queue来recursion可以避免涉及 index

先把所有char 加入 queue, 然后遍历依次pop queue
遇到左括号, 递归
遇到右括号, break

```java
public String decodeString(String s) {
    Deque<Character> queue = new LinkedList<>();
    for (char c : s.toCharArray()) queue.offer(c);
    return helper(queue);
}

private String helper(Deque<Character> queue) {
    StringBuilder sb = new StringBuilder();
    int num = 0;

    while (!queue.isEmpty()) {
        char c = queue.poll();

        if (Character.isDigit(c)) {
            num = num * 10 + c - '0';
        } else if (c == '[') {
            String sub = helper(queue);
            while (num-- > 0) sb.append(sub);
            num = 0;
        } else if (c == ']') {
            break;
        } else { // c is letter
            sb.append(c);
        }
    }
    return sb.toString();
}
```

普通递归，手动找匹配的反括号index

```java
public String decodeString(String s) {
    StringBuilder sb = new StringBuilder();
    int num = 0;

    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);

        if (Character.isDigit(c)) {
            num = num * 10 + c - '0';
        } else if (c == '[') {
            int j = findCloseParentheseIdx(s, i);
            String sub = decodeString(s.substring(i + 1, j));
            while (num-- > 0) sb.append(sub);
            num = 0;
            i = j; // i is at ']', i++ in for will skip ']'
        } else {  // c is letter
            sb.append(c);
        }
    }

    return sb.toString();
}
```

手动找匹配的反括号index的 helper method，后面的题也会用

```java
private int findCloseParentheseIdx(String s, int i) {
    int count = 0;
    int j;
    for (j = i; j < s.length(); j++) {
        if (s.charAt(j) == '[')
            count++;
        else if (s.charAt(j) == ']')
            count--;
        if (count == 0)
            break;
    }
    return j;
}
```

316. Remove Duplicate Letters

388. Longest Absolute File Path

split(), String拆成array

.lastIndexOf()
Stack pop if 当前 level维持不变或者减小

```
for (String s : arr) {
    int numOfTabs = s.lastIndexOf("\t") + 1;
    int level = numOfTabs + 1;
    int len = s.length() - numOfTabs;

    while (level < stack.size()) stack.poll();

    int curLen = stack.peek() + len + 1;
    stack.push(curLen);
    if (s.contains(".")) maxLen = Math.max(maxLen, curLen - 1);
}
```

224. Basic Calculator

Implement a basic calculator to evaluate a simple expression string.
The expression string may contain open (and closing parentheses), the plus + or minus sign -, non-negative integers and empty spaces .
只有加减 (), 计算结果

手动找到匹配的 close parenthsis, 对中间的substring递归
为了不丢失最后一个数， 开始时 在 s 后面加一个 +

```java
public int calculate(String s) {
    if (s.length() == 0) return 0;
    s = s + '+';
    int res = 0, num = 0;
    char lastOp = '+';

    for (int i = 0; i < s.length(); i++) {
        char c = s.charAt(i);
        if (Character.isDigit(c))
            num = num * 10 + c - '0';
        else if (c == '(') {
            int j = findCloseParentheseIdx(s, i);
            num = calculate(s.substring(i + 1, j));
            i = j;
        } else if (c == '+' || c == '-') {
            switch (lastOp) {
                case '+':
                    res += num;
                    break;
                case '-':
                    res -= num;
                    break;
            }
            num = 0;
            lastOp = c;
        }
    }

    return res;
}
```

227. Basic Calculator II

加减乘除，没有括号，计算结果

乘除有优先级，不存在nested问题，用stack解
遇到 lastOp, 处理之前的num, 更新 lastOp
为了不丢失最后一个数，开始时 在 s 后面加一个 +

```java
s = s + "+";
Deque<Integer> stack = new LinkedList<>();
int res = 0, num = 0;
char lastOp = '+';

for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    if (Character.isDigit(c)) {
        num = num * 10 + c - '0';
    } else if (c == '+' || c == '-' || c == '*' || c == '/') {
        switch (lastOp) {
            case '+':
                stack.push(num);
                break;
            case '-':
                stack.push(-num);
                break;
            case '*':
                stack.push(stack.pop() * num);
                break;
            case '/':
                stack.push(stack.pop() / num);
                break;
        }
        num = 0;
        lastOp = c;
    }
}

for (int n : stack) res += n;
return res;
```

772. Basic Calculator III

加减乘除（ ），计算结果
int calculate(String s)

前2题的结合，必须用 stack + recursion

括号用递归处理
乘除优先级高，用stack处理，计算完了再入栈

lastOp num c
因为是遇到下一个c是oper才处理num
为了不丢失最后一个数，开始时 在 s 后面加一个 +

```java
int res = 0, num = 0;
char lastOp = '+';

for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    if (Character.isDigit(c))
        num = num * 10 + c - '0';
    else if (c == '(') {
        int j = findCloseParentheseIdx(s, i);
        num = calculate(s.substring(i + 1, j));
        i = j;
    } else if (c == '+' || c == '-' || c == '*' || c == '/') {
        switch (lastOp) {
            case '+':
                stack.push(num);
                break;
            case '-':
                stack.push(-num);
                break;
            case '*':
                stack.push(stack.pop() * num);
                break;
            case '/':
                stack.push(stack.pop() / num);
                break;
        }
        num = 0;
        lastOp = c;
    }
}

while (!stack.isEmpty()) res += stack.pop();
return res;
```

Sliding Window

Fix Size

```
int left = 0, right;

for (right = 0; right < s.length(); right++) {
      count(right)++;
      if (right >= K - 1) {
            SAVE Result
            count(left)--;
            left++;
            }
}
```

438. Find All Anagrams in a String

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.
p: "abc" s: "cbaebabacd"
Output:
[0, 6]

```java
public List<Integer> findAnagrams(String s, String p) {
    List<Integer> res = new ArrayList<>();
    int[] countS = new int[128], countP = new int[128];

    for (char c : p.toCharArray()) countP[c]++;

    int left = 0;
    for (int right = 0; right < s.length(); right++) {
        countS[s.charAt(right)]++;
        if (right >= p.length() - 1) {
            if (Arrays.equals(countS, countP)) res.add(left);
            countS[s.charAt(left)]--;
            left++;
        }
    }

    return res;
}
```

Longest Window

应用条件，不能有负数，限制条件为<= or >=, (==一般不work)

```
Map<xxxxx, Integer> map = new HashMap<>();
int left = 0;
for (int right = 0; right < s.length(); right++) {
      count.put(xxxRight, count.getOrDefault(xxxRight, 0) + 1);
      while (condition invalid) {
            count.put(xxxLeft, count.get(xxxLeft) - 1);
            count.remove(xxxLeft, 0);
            left++;
            }
      SAVE Result
}
```

159. Longest Substring with At Most Two Distinct Characters

```
for (right = 0; right < s.length(); right++) {
    char charR = s.charAt(right);
    count.put(charR, count.getOrDefault(charR, 0) + 1);

    while (count.size() > 2) {  // condition invalid
        char charL = s.charAt(left);
        count.put(charL, count.get(charL) - 1);
        count.remove(charL, 0);
        left++;
    }
    res = Math.max(res, right - left + 1);
}
return res;
```

3. Longest Substring Without Repeating Characters

```
while (count.get(charR) >= 2)
```

340. Longest Substring with At Most K Distinct Characters

```
while (count.size() > k)
```

713. Subarray Product Less Than K

没有负数
有多少个Subarray
因为all integers are positive, 可以用sliding window做到 O(n)

Shortest Window

```
Map<xxxxx, Integer> map = new HashMap<>();
int left = 0;
for (int right = 0; right < s.length(); right++) {
      count.put(xxxRight, count.getOrDefault(xxxRight, 0) + 1);
      while (condition Valid) {
            SAVE Result
            count.put(xxxLeft, count.get(xxxLeft) - 1);
            count.remove(xxxLeft, 0);
            left++;
            }
}
```

209. Minimum Size Subarray Sum

没有负数
an array of positive integers and a positive integer s
find the minimal length of a contiguous subarray of which the sum ≥ s

76. Minimum Window Substring

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).
String minWindow(String s, String t)

Input: S = "ADOBECODEBANC", T = "ABC"
Output: "BANC"
Input: S = "ADOBECODEBANC", T = "ABCC"
Output: "CODEBANC"

2点需要注意，marks this problem hard
1. T里的char可以重复，相应substring就需要有同样数目的该char
2. substring可以有T里没有的char，不影响结果。

因为substring是连续的，用一个sliding window(left, right 两个pointer)来表示substring
分别用两个 HashMap 统计 T 和 substring 里 char的个数
substring 是否valid: T里的每个char, substring 必须有相同数量（或者更多的）该char。
所以用 matches == countT.size() 作为 valid判断条件
    window增长时，char的数量相等，matches++, 数量超过matches不++
    window缩短时，char的数量小于，matches--

```java
int left = 0, minLen = Integer.MAX_VALUE, matches = 0;
String res = "";

for (int right = 0; right < s.length(); right++) {
    char charR = s.charAt(right);
    if (countT.containsKey(charR)) {
        countWindow.put(charR, countWindow.getOrDefault(charR, 0) + 1);
        if (countWindow.get(charR).equals(countT.get(charR))) matches++;
    }

    while (matches == countT.size()) {
        char charL = s.charAt(left);
        if (right - left + 1 < minLen) {
            minLen = right - left + 1;
            res = s.substring(left, right + 1);
        }
        if (countT.containsKey(charL)) {
            countWindow.put(charL, countWindow.get(charL) - 1);
            if (countWindow.get(charL) < countT.get(charL)) matches--;
        }
        left++;
    }
}

return res;
```

Monotonic Queue/Stack

239. Sliding Window Maximum

有负数
PQ, treeMap, Monotonic queue

PQ解法，维持一个大小为k的max pq　39ms
remove(object) time complexity is O(k)

```java
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());
for (int i = 0; i < k; i++) pq.add(nums[i]);
res[0] = pq.peek();

for (int i = k; i < n; i++) {
    pq.remove(nums[i - k]);
    pq.add(nums[i]);
    res[i - k + 1] = pq.peek();
}
```

TreeMap解法，　33ms
TreeMap<nums[i], i>
Remove time complexity is O(logk)
因为TreeMap不允许duplicate key, 相同的nums[i]，index会被updated

```java
TreeMap<Integer, Integer> map = new TreeMap<>();
for (int i = 0; i < k; i++) map.put(nums[i], i);
res[0] = map.lastKey();

for (int i = k; i < n; i++) {
    if (map.get(nums[i - k]) == i - k) map.remove(nums[i - k]);
    // map.remove(nums[i - k], i - k);
    map.put(nums[i], i);
    res[i - k + 1] = map.lastKey();
}
```

Monotonic Queue　8ms
维持一个decreasing queue
考虑：如果将要进queue的数 > peekLast()，队尾小的数不可能是结果，可以delete
所以最大数永远在peakFirst();
如果即将被移出window的数是当前最大数，需要removeFirst()

```
    Deque<Integer> queue = new LinkedList<>();
    for (int i = 0; i < k; i++) addToQueue(queue, nums[i]);
    res[0] = queue.peekFirst();

    for (int i = k; i < n; i++) {
        if (nums[i - k] == queue.peekFirst()) queue.removeFirst();
        addToQueue(queue, nums[i]);
        res[i - k + 1] = queue.peekFirst();
    }
    return res;
}

private void addToQueue(Deque<Integer> queue, int n) {
    while (!queue.isEmpty() && n > queue.peekLast()) queue.removeLast();
    queue.addLast(n);
}
```

739. Daily Temperatures

Given a list of daily temperatures T, return a list such that, for each day in the input, tells you how many days you would have to wait until a warmer temperature. If there is no future day for which this is possible, put 0 instead.

Next Greater Element的问题, 联想到 Monotonic Stack
Stack里一直维持 温度decreasing 的天数的index
如果将要进stack的数 i > peek()，pop出index，那么这一天的next greaterelement就找到了，继续用i 和 peek() 比较

```
Deque<Integer> stack = new LinkedList<>();
int[] res = new int[n];

for (int i = 0; i < n; i++) {
    while (!stack.isEmpty() && temperatures[i] > temperatures[stack.peek()]) {
        int idx = stack.pop();
        res[idx] = i - idx;
    }
    stack.push(i);
}
return res;
```

862. Shortest Subarray with Sum at Least K

有负数
Only difference from 209 is that this problem have negative numbers.

计算preSum[]
找nearest element from left less than preSum[i] - K
考虑：如果将要进queue的数 < peekLast()，队尾小的数不可能是最小区间的头，可以delete
i 进queue后 考虑以i 为区间尾的最短区间

```
int n = A.length, res = n + 1;
int[] preSum = new int[n + 1];
for (int i = 0; i < n; i++) preSum[i + 1] = preSum[i] + A[i];

Deque<Integer> queue = new LinkedList<>();
for (int i = 0; i < n + 1; i++) {
    while (!queue.isEmpty() && preSum[i] < preSum[queue.peekLast()]) queue.removeLast();
    queue.addLast(i);
    while (!queue.isEmpty() && preSum[i] - preSum[queue.peekFirst()] >= K)
        res = Math.min(res, i - queue.removeFirst());
}
return res == n + 1 ? -1 : res;
```

84. Largest Rectangle in Histogram

Given n non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.
int largestRectangleArea(int[] heights)

找两头的 nearest less than index
Increasing stack
如果将要进stack的数 i < peek(), pop()出的index的右边 nearest less than index就找到了，就是i
又因为stack维持了increasing，pop()出的index的左边 nearest lessthan index就是上一个index，
也就是现在的stack.peek()

```java
int maxArea = 0, currArea;
Deque<Integer> stack = new LinkedList<>();

for (int i = 0; i <= heights.length; i++) {
    while (!stack.isEmpty() && (i == heights.length || heights[i] < heights[stack.peek()])) {
        currArea = heights[stack.pop()] * (stack.isEmpty() ? i : (i - stack.peek() - 1));
        maxArea = Math.max(maxArea, currArea);
    }
    stack.push(i);
}
return maxArea;
```

Greedy

适用条件：If you make a choice that seems the best at the moment and solve the remaining sub-problems later, you still reach an optimal solution.
You will never have to reconsider your earlier choices.
Pick the locally optimal move at each step, and that will lead to the globally optimal solution.

45. Jump Game II

```
for (int i = 0; i < len; i++) {
reachable = Math.max(reachable, i + nums[i]);
if (i == lastReachable) {
    step++;
    lastReachable = reachable;
    if (lastReachable >= len - 1) return step;
}
```

53. Maximum Subarray

greedy就是化简的DP

134. Gas Station

517. Super washing machines

Two Pointers

475. Heaters

you are given positions of houses and heaters on a horizontal line, find out the minimum radius of heaters so that all houses could be covered by those heaters.
int findRadius(int[] houses, int[] heaters)

分别sort houses heaters
每个house被离他最近的heater cover，找到每个house最近的heater, 取所有最近距离的max

```java
public int findRadius(int[] houses, int[] heaters) {
    Arrays.sort(houses);
    Arrays.sort(heaters);

    int i = 0, res = 0;
    for (int house : houses) {
        while (i + 1 < heaters.length && Math.abs(heaters[i + 1] - house) <= Math.abs(heaters[i] - house)) {
            i++;
        }
        res = Math.max(res, Math.abs(heaters[i] - house));
    }
    return res;
}
```

42. Trapping Rain Water

11. Container With Most Water

15. 3Sum

Cyclic Sort swap

Divide and Conquer

241. Different Ways to Add Parentheses

53. Maximum Subarray

Graph

DFS

```
public static final int[][] dirs = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
           if (……)  dfs(board, i, j);
            }

private void dfs(char[][] board, int i, int j) {
       Mark(i, j)
       for (int[] dir : dirs) {
           int p = i + dir[0], q = j + dir[1];
           if (isValidBound(p, q) && isNotMarked(p, q)) {
               dfs(board, p, q);
           }
       }
```

BFS

```
public static final int[][] dirs = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
            if (……)  bfs(board, i, j);
   }

private void bfs(char[][] board, int i, int j) {
      Deque<int[]> queue = new LinkedList<>();
      Mark(i, j)
      queue.offer(new int[]{i, j});
      while (!queue.isEmpty()) {
            int[] x = queue.poll();
            for (int[] dir : dirs) {
            int p = x[0] + dir[0], q = x[1] + dir[1];
            if (isValidBound(p, q) && isNotMarked(p, q)) {
                  Mark(p, q)
                  queue.offer(new int[]{p, q});
            }
```

200. Number of Islands

286. Walls and Gate

130. Surrounded Regions

269. Alien Dictionary

Dfs, visit array set to -1,0,1,2 to represent not exist, exist, visiting, finished

994. Rotting Oranges

417. Pacific Atlantic Water Flow

127. Word Ladder

126. Word Ladder II

Topological Sort

207. Course Schedule

210. Course Schedule II

269. Alien Dictionary

UnionFind

模板

```java
class UnionFind {
    int[] parent;
    int[] rank;
    int count;

    public UnionFind(int n) {
        parent = new int[n];
        rank = new int[n];
        count = n;
        for (int i = 0; i < n; ++i) {
            parent[i] = i;
            rank[i] = 0;
        }
    }

    private int find(int p) { // path compression
        if (parent[p] != p)
            parent[p] = find(parent[p]);
        return parent[p];
    }

    private void union(int p, int q) { // union with rank
        int i = find(p);
        int j = find(q);
        if (i != j) {
            if (rank[i] < rank[j]) parent[i] = j;
            else if (rank[i] > rank[j]) parent[j] = i;
            else {
                parent[j] = i;
                rank[i]++;
            }
            count--;
```

最关键的是find，分为普通和 path compression
普通：while (p != parent[p])
p = parent[p]
return parent[p]

path compression:
if (p != parent[p])
parent[p] = find(parent[p]);
return parent[p]

130. Surrounded Regions

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X'.
A region is captured by flipping all 'O's into 'X's in that surrounding region. 'O's on border are not flipped.
Example:
X X X X
X O O X
X X O X
X O X X

After running your function, the board should be:
X X X X
X X X X
X X X X
X O X X

给每个position标上 site：0 ~ m*n-1
一个dummy site是 m*n，总site数是 m*n+1

建立int[] parent, int[] rank
parent初始化成 0~m*n
rank初始化0
rank在这里的意义就是 height - 1

Union by rank能大大提高效率
path compression提升不明显

```java
public void union(int p, int q) {
    int i = find(p), j = find(q);

    if (i != j) {
        parent[i] = j;
        count--;
    }
}

public void union_rank(int p, int q) {
    int i = find(p), j = find(q);
    if (i != j) {
        if (rank[i] < rank[j]) parent[i] = j;
        else if (rank[i] > rank[j]) parent[j] = i;
        else {
            parent[j] = i;
            rank[i]++;
        }
        count--;
    }
}
```

```java
public int find(int p) {
    while (p != parent[p]) p = parent[p];
    return p;
}

public int find_pathCompression(int p) {
    if (parent[p] == p) return p;
    else {
        parent[p] = find_pathCompression(parent[p]);
        return parent[p];
    }
}
```

Subarray Sum

Start, end两次循环，内层循环重置sum, 然后累加
HashMap记录sum出现的频次
最优解可以做到 O(n)

325. Maximum Size Subarray Sum Equals k

有负数
HashMap <curSum, index>

560. Subarray Sum Equals K

有负数，有多少个Subarray sum == k
HashMap <curSum, count>
put(0, 1)

523. Continuous Subarray Sum

没有负数
A list of non-negative numbers and a target integer k, write a function to check if the array has a continuous subarray of size at least 2 that sums up to a multiple of k
有没有解
boolean checkSubarraySum(int[] nums, int k)

HashMap 做到 O(n)

974. Subarray Sums Divisible by K

有负数
有多少个Subarray
HashMap记录Mod, frequency做到O(n)

Trie

Trie 需要一个 TrieNode class

```java
class TrieNode {
    private TrieNode[] next;
    private boolean isEnd;

    public TrieNode() {
        next = new TrieNode[128];
    }
}
```

对比 TreeNode

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
}
```

Trie class有1个 class field

```
private TrieNode root
```

3个 methods

都有类似的pattern

```
TrieNode node = root;
for (char c : word.toCharArray()) {
      if (node.next[c - 'a'] == null) {
            xxxxxxxxxxx;
      }
      node = node.next[c - 'a'];
}
xxxxxxxxxxx
```

void insert(s) 方法，遇到 null 新建TrieNode, for 循环外标记 word 结束（node.val = word，可以用 node.isEnd = true）

boolean search(s), boolean startWith(s)方法，遇到null 返回 false，for 循环外再return

```java
class Trie {
    private TrieNode root;

    public Trie() {
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.next[c] == null) node.next[c] = new TrieNode();
            node = node.next[c];
        }
        node.isEnd = true;
    }

    public boolean search(String word) {
        TrieNode node = root;
        for (char c : word.toCharArray()) {
            if (node.next[c] == null) return false;
            node = node.next[c];
        }
        return node.isEnd;
    }

    public boolean startsWith(String prefix) {
        TrieNode node = root;
        for (char c : prefix.toCharArray()) {
            if (node.next[c] == null) return false;
            node = node.next[c];
        }
        return true;
```

212. Word Search II

Given a 2D board and a list of words from the dictionary, find all words in the board.
Each word must be constructed from letters of sequentially adjacent cell, where "adjacent" cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

Example:
Input:
board = [

['o','a','a','n'],
['e','t','a','e'],
['i','h','k','r'],
['i','f','l','v']
]
words = ["oath","pea","eat","rain"]

Output: ["eat","oath"]

(1)可以对每个 word 都遍历一遍 board，调用if (exist(board, word))res.add(word);
(2)只遍历一遍 board，从 i，j 出发建立 String s，如果 words 里没有以 s 开头的 string，返回，如果 words 里恰好有 s, s 加到结果。中间调递归。把所有 words 都放到一个 Trie 里。
注意可能多个 i, j 出发都找打同一个 word，所以用 Set 保存 s，再放到 List 里

```java
public List<String> findWords(char[][] board, String[] words) {
    int m = board.length, n = board[0].length;
    Trie trie = new Trie();
    Set<String> res = new HashSet<>();
    boolean[][] visited = new boolean[m][n];
    for (String word : words) {
        trie.insert(word);
    }

    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            dfs(board, visited, "", i, j, trie, res);
        }
    }
    return new ArrayList<>(res);
}

private void dfs(char[][] board,boolean[][] visited, String s, int i, int j, Trie trie, Set<String> res)
    s += board[i][j];
    if (!trie.startsWith(s)) return;
    if (trie.search(s)) res.add(s);

    visited[i][j] = true;
    for (int[] dir : dirs) {
        int p = i + dir[0];
        int q = j + dir[1];
        if (inBound(board, p, q) && !visited[p][q]) {
            dfs(board, visited, s, p, q, trie, res);
        }
    }
    visited[i][j] = false;
}
```

642. Design Search Autocomplete System

"i love you" : 5 times
"island" : 3 times
"ironman" : 2 times
"i love leetcode" : 2 times

Operation: input('i')
Output: ["i love you", "island","i love leetcode"]

Operation: input(' ')
Output: ["i love you","i love leetcode"]

Operation: input('a')
Output: []

Operation: input('#')
Output: []

---

class TrieNode
TrieNode[128] 因为有空格，不在 26 个小写字母范围内
Map<String, Integer> count，记录每个string出现的次数，这些 String 都包含了从 root 到这 TrieNode 的 char，具体实现是 insert这个 string到 count 时，沿途的TrieNode 的 count 里都加上<string, count>

```
class TrieNode {
    TrieNode[] next = new TrieNode[128];
    Map<String, Integer> count = new HashMap<>();
}
```

class AutocompleteSystem 有 2个instance variables

```
private TrieNode root;
private String buffer;
```

Tire 的模板：

```
TrieNode node = root;
for (char c : word.toCharArray()) {
if (node.next[c - 'a'] == null) {
xxxxxxxxxxx;
}
node = node.next[c - 'a'];
}
```

void insert(s) 方法，遇到 null 新建TrieNode, for 循环外标记 word 结束（node.val = word，可以用 node.isEnd = true）
boolean search(s), boolean startWith(s)方法，遇到null 返回 false，for 循环外再return

这题的变化在于：
void insert(String s, int count) 在 for 循环里面，每个 node的 countMap 都加上 <s, count>
List<String> input(char input_c) 相当于模板里的 search，null 返回空 List，for 循环外 return get_top_3(node.countMap)

List<String> input(char input_c)方法相当于search，沿途如果出现 null，返回空 list，直到最后的 node，取这个 node 的 count 的前 3 个，写get_top_3

input method相当于 Trie 里的 search method
get_top_3 method保存结果，用到了自定义comparator Heap

```java
class TrieNode {
    TrieNode[] next = new TrieNode[128];
    Map<String, Integer> countMap = new HashMap<>();
}
```

```java
class AutocompleteSystem {
    private TrieNode root = new TrieNode();
    private String buffer = "";

    public AutocompleteSystem(String[] sentences, int[] times) {
        for (int i = 0; i < sentences.length; i++) {
            insert(sentences[i], times[i]);
        }
    }

    public List<String> input(char input_c) {
        if (input_c == '#') {
            insert(buffer, 1);
            buffer = "";
            return new ArrayList<>();
        }
        buffer = buffer + input_c;
        TrieNode node = root;
        for (char c : buffer.toCharArray()) {
            if (node.next[c] == null) return new ArrayList<>();
            node = node.next[c];
        }
        return get_top_3(node.countMap);
    }
```

```
private void insert(String s, int count) {
    TrieNode node = root;
    for (char c : s.toCharArray()) {
        if (node.next[c] == null) {
            node.next[c] = new TrieNode();
        }
        node = node.next[c];
        node.countMap.put(s, node.countMap.getOrDefault(s, 0) + count);
    }
}

private List<String> get_top_3(Map<String, Integer> counts) {
    PriorityQueue<String> pq = new PriorityQueue<>(
            (a, b) -> counts.get(a).equals(counts.get(b)) ? a.compareTo(b) :
                    counts.get(b) - counts.get(a));
    pq.addAll(counts.keySet());

    List<String> res = new ArrayList<>();
    int k = 3;
    while (k-- > 0 && !pq.isEmpty()) {
        res.add(pq.poll());
    }
    return res;
}
```

1032. Stream of Characters

设计StreamChecker class，给定String[] words。
boolean query(char letter)一个个给 char，组成一个 String s 要求返回能否找到s从后往前的 substring 等于 words 里的某个 string

TrieNode里用boolean isWord，比 String val 要好用。

insert 就是标准 Trie的模板，注意把很多 String 倒过来的 trick：new StringBuilder(word).reverse().toString()

boolean query(char letter)是 boolean seach(String word)的变形，search 里 word 走途中可以确定 false，但是一定要走完才能确定 true。query 里sb 走途中可以确定 false，但是在途中也可以确定 true(只要当前 sb 在 trie 里了，用node.isWord判断)。sb 倒着遍历，可以用 sb.insert(0, xxx)，但是直接 append然后 for 倒着循环更快。

```java
class StreamChecker {

    private TrieNode root = new TrieNode();
    private StringBuilder sb = new StringBuilder();

    public StreamChecker(String[] words) {
        for (String word : words) {
            insert(new StringBuilder(word).reverse().toString());
        }
    }

    public boolean query(char letter) {
        TrieNode node = root;
        sb.append(letter);

        for (int i = sb.length() - 1; i >= 0; i--) {
            char c = sb.charAt(i);
            if (node.next[c - 'a'] == null) return false;
            else {
                node = node.next[c - 'a'];
                if (node.isWord) return true;
            }
        }
        return false;
    }
```

KMP

214. Shortest Palindrome

convert it to a palindrome by adding characters in front of it
return the shortest palindrome you can find by performing this transformation.
String shortestPalindrome(String s)

```java
public String shortestPalindrome(String s) {
    String tmp = s + "#" + new StringBuilder(s).reverse().toString();
    int[] next = getTable(tmp);
    return new StringBuilder(s.substring(next[next.length - 1])).reverse().toString() + s;
}


private int[] getTable(String tmp) {
    int[] next = new int[tmp.length() + 1];
    int k = 0, j = 2;

    while (j < next.length) {
        if (tmp.charAt(j - 1) == tmp.charAt(k)) {
            k++;
            next[j] = k;
            j++;
        } else if (k > 0) {
            k = next[k];
        } else {
            j++;
        }
    }
    return next;
}
```

Design

146. LRU Cache

155. Min Stack

158. Read N Characters Given Read4 II - Call multiple times

173. Binary Search Tree Iterator

297. Serialize and Deserialize Binary Tree

449. Serialize and Deserialize BST

981. Time Based Key-Value Store

实现Trick

Brute Force

681. Next Closest Time

939. Minimum Area Rectangle

Given a set of points in the xy-plane, determine the minimum area of a rectangle formed from these points, with sides parallel to the x and y axes.
If there isn't any rectangle, return 0.
int minAreaRect(int[][] points)

963. Minimum Area Rectangle II

Given a set of points in the xy-plane, determine the minimum area of any rectangle formed from these points, with sides not necessarily parallel to the x and y axes.
If there isn't any rectangle, return 0.
double minAreaFreeRect(int[][] points)

Comparator

179. Largest Number

Given a list of non negative integers, arrange them such that they form the largest number.
String largestNumber(int[] nums)

因为从大到小排序，s2.compareTo(s1)

```
Arrays.sort(strs, new Comparator<String>() {
    @Override
    public int compare(String a, String b) {
        String s1 = a + b;
        String s2 = b + a;
        return s2.compareTo(s1);
    }
});
```

937. Reorder Data in Log Files

自定义comparator, 匿名函数

```
Arrays.sort(logs, new Comparator<String>() {
    @Override
    public int compare(String s1, String s2) {
        int s1Space = s1.indexOf(' ');
        int s2Space = s2.indexOf(' ');
        char s1fc = s1.charAt(s1Space + 1);
        char s2fc = s2.charAt(s2Space + 1);

        if (s1fc <= '9') {
            if (s2fc <= '9') return 0;
            else return 1;
        } else {
            if (s2fc <= '9') return -1;
            else {
                int comp = s1.substring(s1Space + 1).compareTo(s2.substring(s2Space + 1));
                if (comp == 0) return s1.substring(0, s1Space).compareTo(s2.substring(0, s2Space));
                return comp;
            }
        }
    }
});
return logs;
```

354. Russian Doll Envelopes

You have a number of envelopes with widths and heights given as a pair of integers (w, h). One envelope can fit into another if and only if both the width and height of one envelope is greater than the width and height of the other envelope.
What is the maximum number of envelopes can you Russian doll? (put one inside other)
Input: [[5,4],[6,4],[6,7],[2,3]]
Output: 3
Explanation: The maximum number of envelopes you can Russian doll is 3 ([2,3] => [5,4] => [6,7]).
int maxEnvelopes(int[][] envelopes)

[width, height]
对width ascending sort, if width相等，对height descendingsort
然后对 int[] height求 300. Longest Increasing Subsequence

```java
Arrays.sort(envelopes, new Comparator<int[]>(){
    public int compare(int[] arr1, int[] arr2){
        if(arr1[0] == arr2[0])
            return arr2[1] - arr1[1];
        else
            return arr1[0] - arr2[0];
    }
});
```

Using HashMap/HashSet

652. Find Duplicate Subtrees

Given a binary tree, return all duplicate subtrees. For each kind of duplicate subtrees, you only need to return the root node of any one of them.
Two trees are duplicate if they have the same structure with same node values.
List<TreeNode> findDuplicateSubtrees(TreeNode root)

postorder traversal, bottom-up
每个subtree保存成String, 如果String一样，subtree就一样
HashMap保存每个subtree出现的次数

```java
public List<TreeNode> findDuplicateSubtrees(TreeNode root) {
    List<TreeNode> res = new LinkedList<>();
    helper(root, new HashMap<>(), res);
    return res;
}

private String helper(TreeNode cur, Map<String, Integer> map, List<TreeNode> res) {
    if (cur == null) return "#";
    String serial = cur.val + "#" + helper(cur.left, map, res) + "#" + helper(cur.right, map, res);
    if (map.getOrDefault(serial, 0) == 1) res.add(cur);
    map.put(serial, map.getOrDefault(serial, 0) + 1);
    return serial;
}
```

138. Copy List with Random Pointer

133. Clone Graph

Bitwise Operator

a ^ b ^ b = a

String indexOf, replace, replaceAll methods

indexOf : String
replace: String
replaceAll(String regex, String replacement)
split(String regex)

929. Unique Email Addresses

```
Set<String> seen = new HashSet<>();
for (String email : emails) {
    int i = email.indexOf("@");
    String local = email.substring(0, i);
    String rest = email.substring(i);
    if (local.contains("+")) {
        local = local.substring(0, local.indexOf("+"));
    }
    local = local.replace(".", "");
    //local = local.replaceAll("\\.", "");
    seen.add(local + rest);
}

return seen.size();
```

String, array多重循环

228. Summary Ranges

163. Missing Ranges

1291. Sequential Digits

482. License Key Formatting