P50

2 的 N 次方的计算

递归方法:

将 2^n=2^(n/2)*2^(n/2)*2^(n%2)

int temp=2^(n/2);

int 2^n=temp*temp*2^(n%2);

特殊情况：x=0,1,-1;

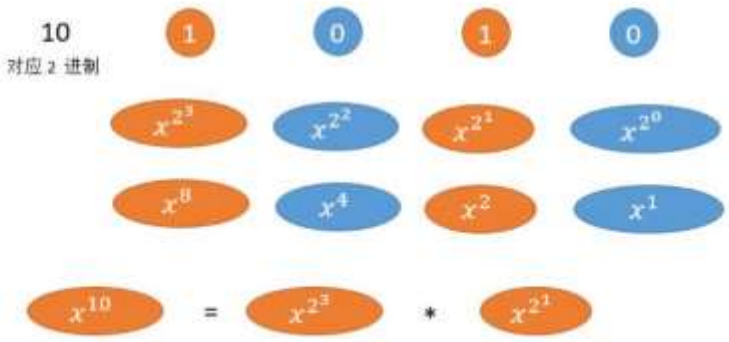
单独考虑 n=-2147483648 的情况

```
if(n==-2147483648){
    if(x<1&&x>-1){
    return Double.POSITIVE_INFINITY;
    }
    else
    return 0;
}
```

以 x 的 10 次方举例。10 的 2 进制是 1010，然后用 2 进制转 10 进制的方法把它展成 2 的幂次的和。

$$x^{10} = x^{(1010)_2} = x^{1*2^3+0*2^2+1*2^1+0*2^0} = x^{1*2^3} * x^{0*2^2} x^{1*2^1} * x^{0*2^0}$$

这样话，再看一下下边的图。它们之间的对应关系就出来了。



迭代:

```
private double getPower(double x,int n){
double ans=1;
while(n>0){
    if ((n & 1) == 1) {
    ans*=x;
    }
    x=x*x;
    n=n>>1;
    }
    return ans;
}
```

P49

49. Group Anagrams

Medium 9285 316 Add to List Share

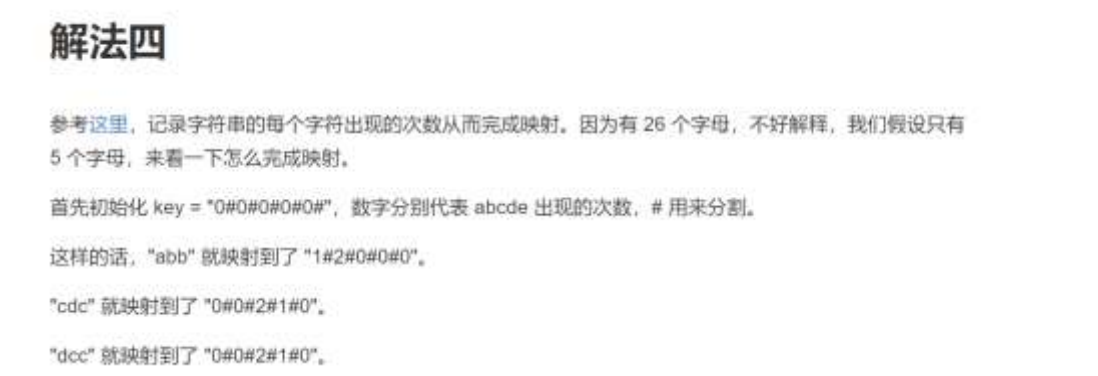
Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

映射的方法。



解法四

参考这里，记录字符串的每个字符出现的次数从而完成映射。因为有 26 个字母，不好解释，我们假设只有 5 个字母，来看一下怎么完成映射。

首先初始化 key = "0#0#0#0#0#"，数字分别代表 abcde 出现的次数，# 用来分割。

这样的话，"abb" 就映射到了 "1#2#0#0#0"。

"cdc" 就映射到了 "0#0#2#1#0"。

"dcc" 就映射到了 "0#0#2#1#0"。

Key 的确定。（1）字母排序，（2）字母出现的次数生成 key

字符数组转成字符串（1） String.valueOf(charArray) （2）String s1=new String(charArray)

Map 遍历 key，value。

```java
    // 使用map接口默认方法
    private static void method_3(HashMap<String, Integer> map) {
        map.forEach((Key, Value)->{
            System.out.println(Key+"---"+Value);
        });
    }

    // 通过迭代器的方式获取
    public static void method_1( HashMap<String, Integer> mp){
        Set<Map.Entry<String, Integer>> set = mp.entrySet();
        Iterator<Map.Entry<String, Integer>> iterator = set.iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Integer> next = iterator.next();
            System.out.println(next.getKey()+"---"+next.getValue());
        }
    }

    // 通过get方式获取
    public static void method_2( HashMap<String, Integer> mp) {
        Set<String> set = mp.keySet();
        for (String s : set) {
            Integer val = mp.get(s);
            System.out.println(s+"----"+val);
        }
    }
    public static void method( HashMap<String, Integer> mp){
        Set<String> set = mp.keySet();
        for (String s : set) {
            System.out.println(s);
        }
        Collection<Integer> values = mp.values();
        for (Integer value : values) {
            System.out.println(value);
        }

    }
```

P48

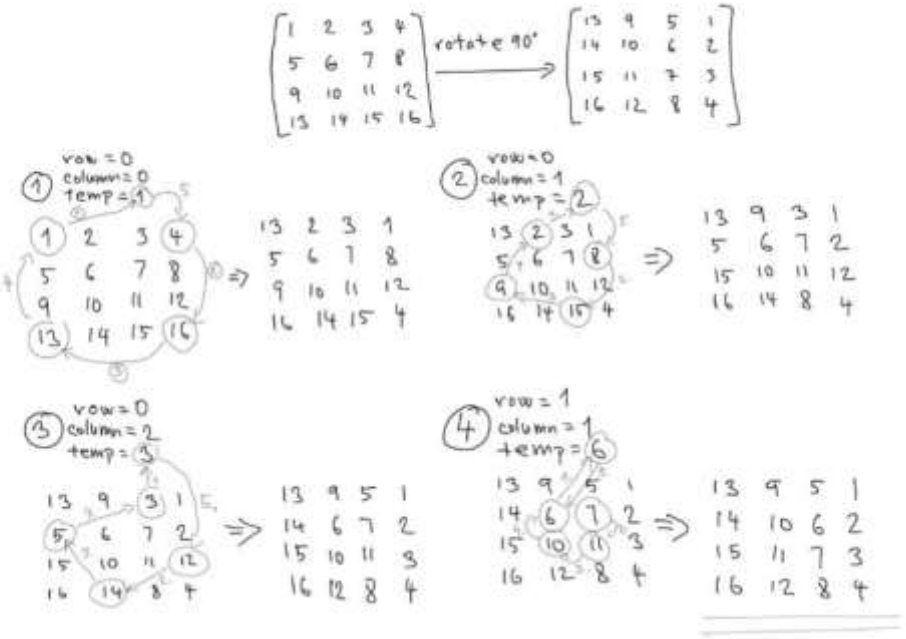
You are given an `n x n` 2D `matrix` representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

90 度旋转一个数组：

方法：（1）先将数组对角线交换（2）对称列交换；

挨个旋转：



一圈一圈的循环交换，很妙！

```
public void rotate(int[][] matrix) {
    int n=matrix.length;
    for (int i=0; i<n/2; i++)
        for (int j=i; j<n-i-1; j++) {
            int tmp=matrix[i][j];
            matrix[i][j]=matrix[n-j-1][i];
            matrix[n-j-1][i]=matrix[n-i-1][n-j-1];
            matrix[n-i-1][n-j-1]=matrix[j][n-i-1];
            matrix[j][n-i-1]=tmp;
        }
}
```

Copy

P47

47. Permutations II

Medium 5634 99 Add to List Share

Given a collection of numbers, `nums`, that might contain duplicates, return *all possible unique permutations* ***in any order***.

Example 1:

```
Input: nums = [1,1,2]
Output:
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

Example 2:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

含有重复元素的 DFS，使用 Stack 记录当前加入的元素的 index，当当前元素已经存入时，continue。关键技术：（1）首先对数组排序，方便跳过重复元素。（2）跳过重复元素。也就是说，重复元素不可以作为新的开始。重复元素作为新的开始时，此时记录 index 的 Stack 为空，也就是说不包含前面的重复元素。

```
    Arrays.sort(nums);
    getPermuteUnique(nums,ans,new ArrayList<>(),new Stack<>());
    return ans;

rivate void getPermuteUnique(int []nums,List<List<Integer>> ans,List<Integer> temp,Stack<Integer> i
    if(temp.size()==nums.length){//达到最大深度
        ans.add(new ArrayList<>(temp));
        return;
    }
    for(int i=0;i<nums.length;i++){
        if(index.contains(i)) continue;//已经添加过该元素
        if(i>0&&nums[i]==nums[i-1]&&!index.contains(i-1)){//重复的元素，只添加
            continue;
        }
        temp.add(nums[i]);
        index.push(i);
        getPermuteUnique(nums,ans,temp,index);
        temp.remove( index: temp.size()-1);
        index.pop();
```

P46

全排列：

46. Permutations

Medium 10442 189 Add to List Share

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Example 2:

```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

深度优先搜索。

每一个数字作为开始，挨个遍历。

```java
private void callBack(List<List<Integer>> ans,List<Integer> temp,int [] nums){
    if(temp.size()==nums.length){
    ans.add(new ArrayList(temp));
    return;
    }
    for(int i=0;i<nums.length;i++){
        if(temp.contains(nums[i])){
        continue;}
        temp.add(nums[i]);
        callBack(ans,temp,nums);
        temp.remove( index: temp.size()-1);
    }
```

插入法。

```
        List<List<Integer>> ans=new ArrayList<>();
        List<Integer> temp=new ArrayList<>();
        temp.add(nums[0]);
        ans.add(temp);
        for(int i=1;i<nums.length;i++){//要添加的数字
            int currSize=ans.size();
            for(int j=0;j<currSize;j++){//可以添加到哪些list当中
                for(int k=0;k<=i;k++){//可以添加的位置
                  temp=new ArrayList<>(ans.get(j));
                    temp.add(k,nums[i]);
                    ans.add(temp);
                }


            }
             for(int j=0;j<currSize;j++){//删除旧的list
                 ans.remove(0);
             }

        }
        return ans;

    }
```

P45

45. Jump Game II

Medium 7996 297 Add to List Share

Given an array of non-negative integers `nums`, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

You can assume that you can always reach the last index.

求出每一步的最大跳跃距离。为下一步的边界。当遍历到该长度时，step+1。更新边界。

```java
class Solution {
    public int jump(int[] nums) {

        int maxJump=0;
        int step=0;
        int end=0;
        for(int i=0;i<nums.length-1;i++){
           maxJump=Math.max(maxJump,i+nums[i]);
            if(i==end){
                end=maxJump;
                step++;
            }
        }
        return step;

        }

}
```

P44

44. Wildcard Matching

Hard 4689 211 Add to List Share

Given an input string (`s`) and a pattern (`p`), implement wildcard pattern matching with support for `'?'` and `'*'` where:

- `'?'` Matches any single character.
- `'*'` Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

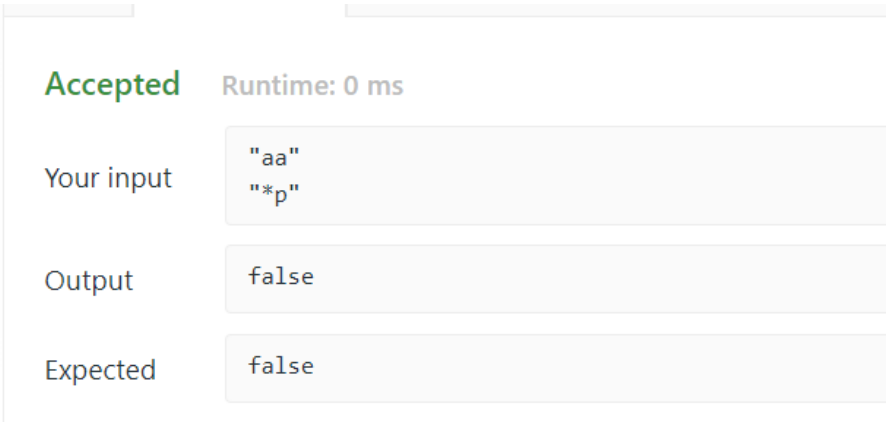
Example 1:

```
Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
```

Example 2:

```
Input: s = "aa", p = "*"
Output: true
Explanation: '*' matches any sequence.
```

问题分析:



迭代：当前字符相等或者是‘？’的时候移动到下一个字符。当存在’*’时，存在不匹配的情况时，p 回溯到‘*’的位置，s 向后移动。最后检查 p 是不是到达最后的位置，没有到达最后的位置时，判定为不匹配。

写代码要细心，随时检查可能出现的问题，越界。条件是否成立。

P43

```
public boolean isMatch(String str,String Pattern){
    int p=0;//index of String p
    int s=0;//index of String s
    int starIn=0;//index of '*'
    int match=0;//the index of s when p is '*'
    int flag=-1;//if exist '*'
    while(s<str.length()){
        if(p<Pattern.length()&&(str.charAt(s)==Pattern.charAt(p)||Pattern.charAt(p)=='?')){//matched at this position
            p++;
            s++;
        }
        else if(p<Pattern.length()&&Pattern.charAt(p)=='*'){
            //match
            match=s;
            starIn=p;
            flag=p;//change the flag
            p++;
        }
        else if(flag!=-1){
            //not match
            p=starIn+1;//come back the position
            match++;//move back
            s=match;
        }
        else{
            return false;
        }
    }
    while(p<Pattern.length()&&Pattern.charAt(p)=='*'){
        p++;
    }
    return p==Pattern.length();
}
```

P42

```java
/*
DP   get the maxmimum left and right height
*/
class P42 {
    public int trap(int[] height) {
        int sum=0;//the initil trapping water
        int n=height.length;
        int leftmax[]=new int [n];
        int rightmax[]=new int [n];
        //get max left height
        for(int i=1;i<n;i++){
            //the left max can be the leftself and the leftmax of left.
            leftmax[i]=Math.max(height[i-1],leftmax[i-1]);
        }
        for(int i=n-2;i>=0;i--){
            rightmax[i]=Math.max(height[i+1],rightmax[i+1]);

        }
        for(int i=1;i<n;i++){
            int min=Math.min(leftmax[i],rightmax[i]);
            if(min>height[i]){
                sum+=min-height[i];
            }
        }
        return sum;
    }
```

```java
/*
double pointer
问题分析：对于leftmax只用了一次，可以不用数组。对于rightmax因为是从右向左更新的，因此需要用到数组。可以通过判定
i的微分（走势），来确定.if (height[left-1]<height[right+1]) 则i两侧的最小值肯定是从左边更新。同理，则是右边
*/
    public int trap_1(int[] height) {
        int sum=0;
        int n=height.length;
        int leftmax=0;
        int rightmax=0;
        int left=1;
        int right=n-2;
        for(int i=1;i<n;i++){
            if(height[left-1]<height[right+1]){
                leftmax=Math.max(leftmax,height[left-1]);
                int min=leftmax;
                if(min>height[left]){
                    sum+=min-height[left];
                }
                left++;
            }
            else{
                rightmax=Math.max(rightmax,height[right+1]);
                int min=rightmax;
                if(min>height[right]){
                    sum+=min-height[right];
                }
                right--;
            }
        }
        return sum;
    }
```

```
stack
采用栈操作的方法。
    */
    public int trap_2(int[] height) {
        int n=height.length;
        int sum=0;
        int curr=0;
        Stack<Integer> stack=new Stack<>();
        while(curr<n){
            while(!stack.isEmpty()&&height[curr]>=stack.peek()){
                int h=height[stack.peek()];
                stack.pop();
                if(stack.isEmpty()){
                    break;
                }
                int distance=curr-stack.peek()-1;
                int min=Math.min(height[curr],height[stack.peek()]);
                sum+=distance*(min-h);
            }
            stack.push(curr);
            curr++;
        }
        return sum;

    }
```