面试题08.01 三步问题（简单）

1. 题目描述

   三步问题。有个小孩正在上楼梯，楼梯有n阶台阶，小孩一次可以上1阶、2阶或3阶。实现一种方法，计算小孩有多少种上楼梯的方式。结果可能很大，你需要对结果模1000000007。

   ```
   示例1:
    输入: n = 3
    输出: 4
    说明: 有四种走法

   示例2:
    输入: n = 5
    输出: 13
   ```

   提示:n范围在[1, 1000000]之间

2. 简单实现

```cpp
class Solution {
public:
    int waysToStep(int n) {
        int MOD = 1e9+7;
        if(n == 1) return 1;
        else if(n == 2) return 2;
        else if(n == 3) return 4;
        vector<long> dp(n, 0);
        dp[0] = 1;
        dp[1] = 2;
        dp[2] = 4;
        for(int i = 3; i < n; i++)
            dp[i] = (dp[i-1] + dp[i-2] + dp[i-3]) % MOD;
        return dp[n-1];
    }
};
```

面试题08.02 迷路的机器人（中等）

1. 题目描述

   设想有个机器人坐在一个网格的左上角，网格 r 行 c 列。机器人只能向下或向右移动，但不能走到一些被禁止的网格（有障碍物）。设计一种算法，寻找机器人从左上角移动到右下角的路径。

网格中的障碍物和空位置分别用 1 和 0 来表示。

返回一条可行的路径，路径由经过的网格的行号和列号组成。左上角为 0 行 0 列。

```
示例 1:
输入:
[
  [0,0,0],
  [0,1,0],
  [0,0,0]
]
输出: [[0,0],[0,1],[0,2],[1,2],[2,2]]
解释:
输入中标粗的位置即为输出表示的路径，即
0行0列（左上角） -> 0行1列 -> 0行2列 -> 1行2列 -> 2行2列（右下角）
```

说明：r 和 c 的值均不超过 100。

2. 简单实现

要注意很多细节才能不超时通过

```
class Solution {
public:
    int m,n;
    vector<vector<int>> ans;
    bool f = false;
    vector<vector<int>> dir = {{0,1}, {1,0}};
    void dfs(vector<vector<int>>& obstacleGrid, int x, int y, vector<vector<int>>&
temp){
        if(f) return;//提前返回
        temp.push_back({x, y});
        if(x == m-1 && y == n-1){
            ans = temp;
            f = true;
            return;
        }
        for(int i = 0; i < 2; i++){
            int xx = x + dir[i][0];
            int yy = y + dir[i][1];
            if(xx < m && yy < n && obstacleGrid[xx][yy] == 0){
                obstacleGrid[xx][yy] = 1;//必须标记，防止重复dfs
                dfs(obstacleGrid, xx, yy, temp);
```

```
                if(f) return;
                else temp.pop_back();
            }
        }
    }
    vector<vector<int>> pathWithObstacles(vector<vector<int>>& obstacleGrid) {
        m = obstacleGrid.size();
        if(m == 0) return ans;
        n = obstacleGrid[0].size();
        if(n == 0) return ans;
        if(obstacleGrid[0][0] != 0 || obstacleGrid[m-1][n-1] != 0) return ans;//巨
坑
        vector<vector<int>> temp;
        dfs(obstacleGrid, 0, 0, temp);
        return ans;
    }
};
```

面试题08.03 魔术索引（简单）

1. 题目描述

   魔术索引。 在数组A[0...n-1]中，有所谓的魔术索引，满足条件A[i] = i。给定一个有序整数数组，编写一种方法找出魔术索引，若有的话，在数组A中找出一个魔术索引，如果没有，则返回-1。若有多个魔术索引，返回索引值最小的一个。

```
示例1:
 输入: nums = [0, 2, 3, 4, 5]
 输出: 0
 说明: 0下标的元素为0

示例2:
 输入: nums = [1, 1, 1]
 输出: 1
```

   提示:nums长度在[1, 1000000]之间

2. 简单实现

```
class Solution {
public:
    int findMagicIndex(vector<int>& nums) {
        int idx = lower_bound(nums.begin(), nums.end(), 0) - nums.begin();//找到第一
个非负数
        while(idx >= 0 && idx < nums.size()){
            if(nums[idx] == idx)
                return idx;
            else if(nums[idx] > idx)
                idx = nums[idx];//可以直接往后跳
            else
                idx++;
        }
```

```
        return -1;
    }
};
```

3. 二分法

```
class Solution {
public:
    int findMagicIndex(vector<int>& nums) {
        return findMagicIndex_helper(nums,0,nums.size()-1);
    }
    int findMagicIndex_helper(vector<int>& nums,int left,int right) {
        if(left==right) {//base情形
            if(nums[left]==left)return left;
            else return -1;
        }
        int mid=(left+right)/2;
        if(nums[mid]==mid) {//只需在前半部分寻找更小的下标
            int t=findMagicIndex_helper(nums,left,mid);
            return (t==-1)?mid:t;
        }
        else {//先在左边找，没找到再去右边找
            int t=findMagicIndex_helper(nums,left,mid);
            if(t!=-1) return t;
            else return findMagicIndex_helper(nums,mid+1,right);
        }
    }
};
```

面试题08.04 幂集（中等）

1. 题目描述

   幂集。编写一种方法，返回某集合的所有子集。集合中不包含重复的元素。

   说明：解集不能包含重复的子集。

```
示例:
 输入:  nums = [1,2,3]
 输出:
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

2. 简单实现

- 以题目示例为例，可以用产生000,001,...,111共八个模式串，对应自然数0~7，模式串可以作为mask对nums进行子集提取，模式串中对应位为0表示不取该元素，为1表示取该元素
- bitset初始化时必须静态初始化，因此假定nums的长度范围为0~32，再进行计算（直接像方法二那样一位一位判断也行，但是只要用的int型存储n，实际上也是隐含假设了该条件）

```
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int len = nums.size();//模式串长度
        if(len == 0) return {{}};
        if(len > 32) return {{-1}};//长度超过32时无法处理
        int n = pow(2, len);//共个子集
        vector<vector<int>> ans;
        while(--n >= 0){//1..1 -> 0..0
            bitset<32> b(n);
            vector<int> cur;
            for(int i = 0; i < len; i++){
                if(b[i] == 1)
                    cur.push_back(nums[len-1-i]);
            }
            ans.push_back(cur);
        }
        return ans;
    }
};

//法二
class Solution {
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        int len = nums.size();//模式串长度
        if(len == 0) return {{}};
        if(len > 32) return {{-1}};//长度超过32时无法处理
        int n = pow(2, len);//共个子集
        vector<vector<int>> ans;
        while(--n >= 0){//1..1 -> 0..0
            int temp = n;
            int idx = 0;
            vector<int> cur;
            while(temp){//隐含了num小于32位，否则int会溢出
                if(temp & 1 == 1)
                    cur.push_back(nums[len-1-idx]);
                idx++;
                temp = temp >> 1;
            }
            ans.push_back(cur);
        }
        return ans;
    }
};
```

3. 类似BFS

先在result列表中将入一个空子集
然后遍历nums每一个元素，将元素与result中的每个元素结合加到result中
直到nums遍历完

代码

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        result = []
        result.append([])
        for i in nums:
            l = len(result)
            j = 0
            while j < l:
                result.append(result[j]+[i])
                j += 1
        return result
```

面试题08.05 递归乘法（中等）

1. 题目描述

   递归乘法。 写一个递归函数，不使用 * 运算符， 实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。

   ```
   示例1:
    输入: A = 1, B = 10
    输出: 10

   示例2:
    输入: A = 3, B = 4
    输出: 12
   ```

   提示:保证乘法范围不会溢出

2. 简单实现

   - 从二进制的角度看，如果B是一个2的幂，如二进制的100，则A乘B等价于A左移两位
   - 如果B=101010，则可以拆成101000 + 000010，第二部分可以直接位移求得，第一部分递归调用即可

```
class Solution {
public:
```

```
    int multiply(int A, int B) {
        if(A == 0 || B == 0) return 0;
        int cur = 1;
        int cnt = 0;
        while(!cur & B == 1){//找到B种最右侧的1开头的串
            cur = cur << 1;
            cnt++;
        }
        int ans = A << cnt;//第二部分，A左移
        B = B - cur;
        if(B > 0)//递归调用
            return ans + multiply(A, B);
        else//乘完了
            return ans;
    }
};
```

面试题08.06 汉诺塔问题（简单）

1. 题目描述

   在经典汉诺塔问题中，有 3 根柱子及 N 个不同大小的穿孔圆盘，盘子可以滑入任意一根柱子。一开始，所有盘子自上而下按升序依次套在第一根柱子上(即每一个盘子只能放在更大的盘子上面)。移动圆盘时受到以下限制: (1) 每次只能移动一个盘子; (2) 盘子只能从柱子顶端滑出移到下一根柱子; (3) 盘子只能叠在比它大的盘子上。

   请编写程序，用栈将所有盘子从第一根柱子移到最后一根柱子。你需要原地修改栈。

```
示例1:
 输入: A = [2, 1, 0], B = [], C = []
 输出: C = [2, 1, 0]

示例2:
 输入: A = [1, 0], B = [], C = []
 输出: C = [1, 0]
```

   提示:A中盘子的数目不大于14个。

2. 简单实现

```
class Solution {
public:
    void move(int n, vector<int>& A, vector<int>& B, vector<int>& C){
        if(n == 1){
            C.push_back(A.back());
            A.pop_back();
        }
        else{
            move(n-1, A, C, B);
            move(1, A, B, C);
            move(n-1, B, A, C);
        }
```

```
    }
    void hanota(vector<int>& A, vector<int>& B, vector<int>& C) {
        int n = A.size();
        move(n, A, B, C);
    }
};
```

面试题08.07 无重复字符串的排列组合（中等）

1. 题目描述

   无重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合，字符串每个字符均不相同。

```
示例1:
 输入: S = "qwe"
 输出: ["qwe", "qew", "wqe", "weq", "ewq", "eqw"]

示例2:
 输入: S = "ab"
 输出: ["ab", "ba"]
```

   提示:

   - 字符都是英文字母。
   - 字符串长度在[1, 9]之间。

2. 简单实现——BFS（DFS也可）

```
class Solution {
public:
    vector<string> permutation(string S) {
        int len = S.size();
        if(len == 0) return {};
        else if(len == 1) return {S};
        vector<string> ans;
        queue<pair<string, string>> q;//pair<已排列字符，剩下的字符>
        q.push(make_pair("", S));
        while(--len){
            int size = q.size();
            for(int i = 0; i < size; i++){
                string cur = q.front().first;
                string temp = q.front().second;
                q.pop();
                for(int j = 0; j < temp.size(); j++){
                    string newstr = temp;
                    newstr.erase(newstr.begin()+j);
                    q.push(make_pair(cur+temp[j], newstr));
                }
            }
        }
        while(!q.empty()){
            ans.push_back(q.front().first + q.front().second);
            q.pop();
```

```
            }
            return ans;
        }
    };
```

3. 动态规划

将每个字母插入到前面字符串所有可能的情况：
例如："qwe"
第一轮：q
第二轮：将w插入q,有两种情况：wq, qw
第三轮：将e插入wq, qw中去 即：ewq,weq,wqe;eqw,qew,qwe六种情况

```
class Solution {
public:
    vector<string> permutation(string S) {
        int len = S.length();
        vector<string> res,result;
        if(len <=0 )
            return res;
        string tmp = S.substr(0,1);
        res.push_back(tmp);
        for(int i=1;i<len;i++){
            int count = res.size();
            for(int j=0;j<count;j++){
                string origin = res[j];
                int length = origin.length();
                for(int k=0;k<length;k++){
                    string t=origin.substr(0,k)+S[i]+origin.substr(k);
                    res.push_back(t);
                }
                res.push_back(origin+S[i]);
            }
            res.erase(res.begin(),res.begin()+count);
        }
        return res;
    }
};
```

面试题08.08 有重复字符串的排列组合（中等）

1. 题目描述

有重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合。

```
示例1:
 输入: S = "qqe"
 输出: ["eqq","qeq","qqe"]

示例2:
 输入: S = "ab"
 输出: ["ab", "ba"]
```

提示:

- 字符都是英文字母。
- 字符串长度在[1, 9]之间。

2. 正确解法

学会利用swap构造排列组合，其实也是DFS的一种写法

```
class Solution {
public:
    vector<string>rs;
    void permutation_(string &S,int index) {
        if(index == S.size() - 1){
            rs.push_back(S);
            return;
        }
        for (int i = index; i < S.size(); i++) {
            if(i == index){
                permutation_(S, index+1);
                continue;
            }
            //去重
            //只要判断前面的有没有  和 自己相等的元素即可
            int k = i-1;
            for (; k >= index; k--) {
                if(S[k] == S[i]) break;
            }
            if(k != index - 1) continue;
            //去掉以上去重这部分代码就是上一题的答案
            swap(S[index], S[i]);
            permutation_(S, index+1);
            swap(S[index], S[i]);
        }
    }
    vector<string> permutation(string S) {
        permutation_(S, 0);
        return rs;
    }
};
```

面试题08.09 括号（中等）

1. 题目描述

括号。设计一种算法，打印n对括号的所有合法的（例如，开闭一一对应）组合。

说明：解集不能包含重复的子集。

```
例如，给出 n = 3，生成结果为：
[
  "((()))",
  "(()())",
  "(())()",
  "()(())",
  "()()()"
]
```

2. 简单实现

```
class Solution {
public:
    vector<string> ans;
    void generate(int l, int r, int n, string temp){
        if(l == r && r == n)
            ans.push_back(temp);
        else{
            if(l == r)
                generate(l+1, r, n, temp+'(');
            else{
                if(l < n)
                    generate(l+1, r, n, temp+'(');
                generate(l, r+1, n, temp+')');
            }
        }
    }
    vector<string> generateParenthesis(int n) {
        generate(0, 0, n, "");
        return ans;
    }
};
```

面试题08.10 颜色填充（简单）

1. 题目描述

颜色填充。编写函数，实现许多图片编辑软件都支持的“颜色填充”功能。给定一个屏幕（以二维数组表示，元素为颜色值）、一个点和一个新的颜色值，将新颜色值填入这个点的周围区域，直到原来的颜色值全都改变。

```
示例1:

 输入:
image = [[1,1,1],[1,1,0],[1,0,1]]
sr = 1, sc = 1, newColor = 2
 输出: [[2,2,2],[2,2,0],[2,0,1]]
 解释:
在图像的正中间，(坐标(sr,sc)=(1,1))，
在路径上所有符合条件的像素点的颜色都被更改成2。
注意，右下角的像素没有更改为2，
因为它不是在上下左右四个方向上与初始点相连的像素点。
```

说明:

- image 和 image[0] 的长度在范围 [1, 50] 内。
- 给出的初始点将满足 0 <= sr < image.length 和 0 <= sc < image[0].length。
- image[i][j] 和 newColor 表示的颜色值在范围 [0, 65535]内。

2. 简单实现

```
class Solution {
public:
    int m,n;
    vector<vector<int>> dir = {{-1,0}, {1,0}, {0,-1}, {0,1}};
    void fill(vector<vector<int>>& image, int x, int y, int oldColor, int newColor)
{
        image[x][y] = newColor;
        for(int i = 0; i < 4; i++){
            int xx = x + dir[i][0];
            int yy = y + dir[i][1];
            if(xx >= 0 && xx < m && yy >= 0 && yy < n && image[xx][yy] == oldColor)
                fill(image, xx, yy, oldColor, newColor);
        }
    }
    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int
newColor) {
        if(image[sr][sc] == newColor)
            return image;//如果不加判断直接fill，会陷入无限的递归种
        m = image.size();
        n = image[0].size();
        fill(image, sr, sc, image[sr][sc], newColor);
        return image;
    }
};
```

面试题08.11 硬币（中等）

1. 题目描述

给定数量不限的硬币，币值为25分、10分、5分和1分，编写代码计算n分有几种表示法。(结果可能会很大，你需要将结果模上1000000007)

```
示例1:
 输入: n = 5
 输出: 2
 解释: 有两种方式可以凑成总金额:
5=5
5=1+1+1+1+1

示例2:
 输入: n = 10
 输出: 4
 解释: 有四种方式可以凑成总金额:
10=10
10=5+5
10=5+1+1+1+1+1
10=1+1+1+1+1+1+1+1+1+1
```

注意: 你可以假设：0 <= n (总金额) <= 1000000

2. 正确解法——二维动态规划
   - 对于以前做过的那种类似于青蛙跳台阶的**排列问题**，只需要简单的一维动态规划即可
   - 对于本题这种**组合问题**，需要二维动态规划——典型完全背包问题

【核心思想】

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

【数据结构】

- 二维数组

【思路】

1. 令 `dp[i][j]` 为遍历到当下这个硬币时，组成金额 j 的方法数目
2. 有两种可能性（1）当前这个硬币没有取，`dp[i][j]=dp[i-1][j]`；（2）当前这个硬币取了，`dp[i][j]=dp[i][j-coins[i]]`。最后的结果是两者的和
3. 将状态转移方程翻译成代码，并处理边界条件

```
class Solution {
public:
    int waysToChange(int n) {
        vector<vector<int>> dp(4, vector<int>(n+1));
        vector<int> coins = {1,5,10,25};
        for(int i=0; i<=n; i++)//仅使用一分的硬币，组成方法唯一
            dp[0][i] = 1;
        for(int i=0; i<4; i++)//组成0元的方法唯一
            dp[i][0] = 1;
        for(int i=1; i<4; i++){
            for(int j=1; j<=n; j++){
                if(j >= coins[i])
                    dp[i][j] = (dp[i-1][j] + dp[i][j-coins[i]])%1000000007;
                else
                    dp[i][j] = dp[i-1][j];
            }
```

```
        }
        return dp[3][n];
    }
};
```

面试题08.12 八皇后（困难）

1. 题目描述

   设计一种算法，打印 N 皇后在 N × N 棋盘上的各种摆法，其中每个皇后都不同行、不同列，也不在对角线上。这里的“对角线”指的是所有的对角线，不只是平分整个棋盘的那两条对角线。

   注意：本题相对原题做了扩展

```
示例:
 输入: 4
 输出: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
 解释: 4 皇后问题存在如下两个不同的解法。
[
 [".Q..",  // 解法 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // 解法 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

2. 简单实现——BFS

   从第一行开始放，一直放到第N行

```
class Solution {
public:
    bool check(vector<string>& cur, int i, int j){//检查能否在cur的(i,j)位置放置皇后
        for(int idx = 1; idx <= i; idx++){
            if(cur[i-idx][j] == 'Q')//同列
                return false;
            if(j-idx >= 0 && cur[i-idx][j-idx] == 'Q')//左上对角线
                return false;
            if(j+idx < cur[0].size() && cur[i-idx][j+idx] == 'Q')//右上对角线
                return false;
        }
        return true;
    }
    vector<vector<string>> solveNQueens(int n) {
        if(n == 1)//再忘了特殊情况你就进不了Google了嘤嘤嘤!
            return {{"Q"}};
        vector<vector<string>> ans;
        queue<vector<string>> q;
```

```
        //初始化，第一行所有位置都可以放
        string str(n, '.');//stirng支持与vector相同的操作
        vector<string> cur(n, str);
        for(int i = 0; i < n; i++){
            cur[0][i] = 'Q';
            q.push(cur);
            cur[0][i] = '.';
        }
        for(int i = 1; i < n; i++){//放置剩余的n-1行
            int size = q.size();
            while(size--){
                vector<string> cur = q.front();
                q.pop();
                for(int j = 0; j < n; j++){//依次检查第i列能不能放
                    if(check(cur, i, j)){
                        cur[i][j] = 'Q';
                        if(i == n-1)//放到最后一行了，直接加入答案中
                            ans.push_back(cur);
                        else
                            q.push(cur);
                        cur[i][j] = '.';
                    }
                }
            }
        }
        return ans;
    }
};
```

面试题08.13 堆箱子（困难）

1. 题目描述

   堆箱子。给你一堆n个箱子，箱子宽 wi、高hi、深di。箱子不能翻转，将箱子堆起来时，下面箱子的宽度、高度和深度必须大于上面的箱子。实现一种方法，搭出最高的一堆箱子。箱堆的高度为每个箱子高度的总和。

   输入使用数组[wi, di, hi]表示每个箱子。

```
示例1:
 输入: box = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]
 输出: 6

示例2:
 输入: box = [[1, 1, 1], [2, 3, 4], [2, 6, 7], [3, 4, 5]]
 输出: 10
```

   提示: 箱子的数目不大于3000个。

2. 简单实现

   最朴素的暴力动态规划就能过，只是时间效率不高

```
class Solution {
```

```
public:
    static bool cmp(vector<int>& a, vector<int>& b){//按宽度升序排序
        if(a[0] < b[0])
            return true;
        else
            return false;
    }
    int pileBox(vector<vector<int>>& box) {
        sort(box.begin(), box.end(), cmp);//排序
        vector<int> dp(box.size());//dp[i]表示以box[i]为底的最高的高度
        int ans = 0;
        for(int i = 0; i < box.size(); i++){
            dp[i] = box[i][2];
            for(int j = i-1; j >= 0; j--){//可能可以摞在当前box上面的box都在前面
                if(box[j][0] < box[i][0] && box[j][1] < box[i][1] && box[j][2] <
box[i][2])
                    dp[i] = max(dp[i], dp[j] + box[i][2]);
            }
            ans = max(ans, dp[i]);
        }
        return ans;
    }
};
```

3. 改进

参考第一的写法，将前面已经遍历过的box高度升序排序，就可以提前终止向前的遍历

```
class Solution {
public:
    static bool cmp(vector<int>& a, vector<int>& b){//按宽度升序排序
        if(a[0] < b[0])
            return true;
        else
            return false;
    }
    int pileBox(vector<vector<int>>& box) {
        sort(box.begin(), box.end(), cmp);//排序
        vector<int> dp(box.size());//dp[i]表示以box[i]为底的最高的高度
        for(int i = 0; i < box.size(); i++){
            dp[i] = box[i][2];
            for(int j = i-1; j >= 0; j--){//可能可以摞在当前box上面的box都在前面
                if(box[j][0] < box[i][0] && box[j][1] < box[i][1] && box[j][2] <
box[i][2]){
                    dp[i] = max(dp[i], dp[j] + box[i][2]);
                    break;
                }
            }
            //插入排序，使dp升序
            int idx = i;
            while(idx >= 1 && dp[idx] < dp[idx-1]){
                swap(dp[idx], dp[idx-1]);
                swap(box[idx], box[idx-1]);
```

```
                idx--;
            }
        }
        return dp.back();
    }
};
```

面试题08.14 布尔运算（中等）

1. 题目描述

   给定一个布尔表达式和一个期望的布尔结果 result，布尔表达式由 0 (false)、1 (true)、& (AND)、 | (OR) 和 ^ (XOR) 符号组成。实现一个函数，算出有几种可使该表达式得出 result 值的括号方法。

```
示例 1:
输入: s = "1^0|0|1", result = 0
输出: 2
解释: 两种可能的括号方法是
1^(0|(0|1))
1^((0|0)|1)

示例 2:
输入: s = "0&0&0&1^1|0", result = 1
输出: 10
```

   提示：运算符的数量不超过 19 个

2. 正确解法——区间DP

   `dp[i][j][0/1]`表示第i..j个布尔值组合结果为0/1的组合个数，转移方法为：枚举区间分割点，讨论左右区间计算结果，方案数增量为左右方案数相乘。

```
class Solution {
public:
    int countEval(string s, int result) {
        int len = s.size();
        int n = (len+1) / 2;//布尔值个数
        vector<vector<vector<int>>> dp(n, vector<vector<int>>(n, vector<int>(2,
0)));
        for(int i = 0; i < n; i++){
            dp[i][i][0] = (s[2*i] == '0');
            dp[i][i][1] = (s[2*i] == '1');
        }
        for(int l = 2; l <= n; l++){//区间长度逐渐变大
            for(int i = 0; i+l-1 < n; i++){//区间左起点
                int j = i+l-1;//区间右终点
                for(int k = i; k < j; k++){//遍历区间内各个可分裂点，以该点为左半段结尾
                    char op = s[2*k+1];//连接左右半段的运算符
                    if(op == '&'){
                        dp[i][j][0] += dp[i][k][0] * dp[k+1][j][0]
                                     + dp[i][k][0] * dp[k+1][j][1]
                                     + dp[i][k][1] * dp[k+1][j][0];
```

```
                        dp[i][j][1] += dp[i][k][1] * dp[k+1][j][1];
                    }
                    else if(op == '|'){
                        dp[i][j][0] += dp[i][k][0] * dp[k+1][j][0];
                        dp[i][j][1] += dp[i][k][1] * dp[k+1][j][1]
                                     + dp[i][k][0] * dp[k+1][j][1]
                                     + dp[i][k][1] * dp[k+1][j][0];
                    }
                    else{
                        dp[i][j][0] += dp[i][k][0] * dp[k+1][j][0]
                                     + dp[i][k][1] * dp[k+1][j][1];
                        dp[i][j][1] += dp[i][k][0] * dp[k+1][j][1]
                                     + dp[i][k][1] * dp[k+1][j][0];
                    }
                }
            }
        }
        return dp[0][n-1][result];
    }
};
```

面试题10.01 合并排序的数组（简单）

1. 题目描述

   给定两个排序后的数组 A 和 B，其中 A 的末端有足够的缓冲空间容纳 B。 编写一个方法，将 B 合并入 A 并排序。

   初始化 A 和 B 的元素数量分别为 m 和 n。

```
示例:
输入:
A = [1,2,3,0,0,0], m = 3
B = [2,5,6],       n = 3
输出: [1,2,2,3,5,6]
```

2. 简单实现

```
class Solution {
public:
    void merge(vector<int>& A, int m, vector<int>& B, int n) {
        int i = m-1, j = n-1;
        int idx = m+n-1;
        while(i >= 0 && j >= 0){
            if(A[i] <= B[j])
                A[idx--] = B[j--];
            else
                A[idx--] = A[i--];
        }
        while(j >= 0)
            A[idx--] = B[j--];
    }
```

```
};
```

面试题10.02 变位词组（中等）

1. 题目描述

   编写一种方法，对字符串数组进行排序，将所有变位词组合在一起。变位词是指字母相同，但排列不同的字符串。

   注意：本题相对原题稍作修改

   ```
   示例:
   输入: ["eat", "tea", "tan", "ate", "nat", "bat"],
   输出:
   [
     ["ate","eat","tea"],
     ["nat","tan"],
     ["bat"]
   ]
   ```

   说明：

   - 所有输入均为小写字母。
   - 不考虑答案输出的顺序。

2. 简单实现

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> m;//记录字符串内字符排序后结果相同的字符串
        for(int i = 0; i < strs.size(); i++){
            string cur = strs[i];
            sort(cur.begin(), cur.end());
            m[cur].push_back(strs[i]);
        }
        vector<vector<string>> ans;
        for(auto it = m.begin(); it != m.end(); ++it)
            ans.push_back(it->second);
        return ans;
    }
};
```

面试题10.03 搜素旋转数组（中等）

1. 题目描述

   搜索旋转数组。给定一个排序后的数组，包含n个整数，但这个数组已被旋转过很多次了，次数不详。请编写代码找出数组中的某个元素，假设数组元素原先是按升序排列的。若有多个相同元素，返回索引值最小的一个。

```
示例1:
 输入: arr = [15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14], target = 5
 输出: 8 (元素5在该数组中的索引)

示例2:
 输入: arr = [15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14], target = 11
 输出: -1  (没有找到)
```

提示:arr 长度范围在[1, 1000000]之间

2. 简单实现

出题描述让人醉了， 可能他的意思是一次只旋转一个数吧，总之整个组数会分为两段升序区间

先二分找旋转点，再二分搜索，这样不容易写崩

```
class Solution {
    int find(const vector<int>& nums, int lo, int hi, int target) {//单调掉区间内的二分查找
        int m = -1;
        while (lo < hi) {
            m = lo + (hi-lo)/2;
            if (target <= nums[m]) hi = m;
            else lo = m+1;
        }
        // 若有多个相同元素，返回索引值最小的一个
        if(nums[lo] == target)
            return lo;
        else
            return -1;
    }
public:
    int search(vector<int>& arr, int target) {
        if (arr.size() == 0) return -1;
        int lo = 0;
        int hi = arr.size()-1;
        // 找旋转点
        while (lo < hi) {
            int m = lo + (hi-lo)/2;
            if (arr[m] > arr[hi]) lo = m+1;
            else if (arr[m] < arr[hi]) hi = m;
            else --hi;
        }
        if (target > arr[0]) // 在前半段
            return find(arr, 0, hi-1, target);
        else if (target < arr[0]) // 在后半段
            return find(arr, hi, arr.size()-1, target);
        else return 0;
    }
};
```

面试题10.05 稀疏数组搜索（简单）

1. 题目描述

稀疏数组搜索。有个排好序的字符串数组，其中散布着一些空字符串，编写一种方法，找出给定字符串的位置。

```
示例1:
 输入: words = ["at", "", "", "", "ball", "", "", "car", "", "","dad", "", ""], s = "ta"
 输出: -1
 说明: 不存在返回-1。

示例2:
 输入: words = ["at", "", "", "", "ball", "", "", "car", "", "","dad", "", ""], s = "ball"
 输出: 4
```

提示: words的长度在[1, 1000000]之间

2. 简单实现

```
class Solution {
public:
    int findString(vector<string>& words, string s) {
        int len = words.size();
        int l = 0, r = len-1;
        while(l <= r){
            int mid = l + (r - l) / 2;
            if(words[mid] != ""){
                if(words[mid] == s)
                    return mid;
                else if(words[mid] < s)
                    l = mid + 1;
                else
                    r = mid - 1;
            }
            else{
                //向左找第一个非空字符串
                int idx = mid;
                while(idx >= 0 && words[idx] == "") idx--;
                if(idx >= 0){
                    if(words[idx] == s)
                        return idx;
                    else if(words[idx] > s){
                        r = mid - 1;
                        continue;
                    }
                }
                //向右找第一个非空字符串
                idx = mid;
                while(idx < len && words[idx] == "") idx++;
                if(idx < len){
                    if(words[idx] == s)
                        return idx;
```

```
                    else if(words[idx] < s){
                        l = mid+1;
                        continue;
                    }
                }
                return -1;//大于左边的，小于右边的，说明不存在该字符串
            }
        }
        return -1;
    }
};
```

面试题10.09 排序矩阵查找（中等）

1. 题目描述

   给定M×N矩阵，每一行、每一列都按升序排列，请编写代码找出某元素。

```
示例:
现有矩阵 matrix 如下:
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
给定 target = 5，返回 true。
给定 target = 20，返回 false。
```

2. 简单实现

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size();
        if(m <= 0) return false;
        int n = matrix[0].size();
        if(n <= 0) return false;
        int r = 0, c = n-1;
        while(r < m && c >= 0){
            if(matrix[r][c] == target)
                return true;
            else if(matrix[r][c] < target)
                r++;
            else
                c--;
        }
        return false;
    }
};
```

面试题10.10 数字流的秩（中等）

1. 题目描述

   假设你正在读取一串整数。每隔一段时间，你希望能找出数字 x 的秩(小于或等于 x 的值的个数)。请实现数据结构和算法来支持这些操作，也就是说：

   - 实现 track(int x) 方法，每读入一个数字都会调用该方法；
   - 实现 getRankOfNumber(int x) 方法，返回小于或等于 x 的值的个数。

   注意：本题相对原题稍作改动

   ```
   示例:
   输入: "StreamRank", "getRankOfNumber", "track", "getRankOfNumber", [1], [0], [0]]
   输出: [null,0,null,1]
   ```

   提示：

   - x <= 50000
   - track 和 getRankOfNumber 方法的调用次数均不超过 2000 次

2. 简单实现

   ```
   class StreamRank {
   public:
       vector<int> data;//把输入过的数字升序存储
       StreamRank() {
       }
       void track(int x) {//二分法插入
           auto it = upper_bound(data.begin(), data.end(), x);
           data.insert(it, x);
       }
       int getRankOfNumber(int x) {//二分法查找
           int idx = upper_bound(data.begin(), data.end(), x) - data.begin();
           return idx;
       }
   };
   ```

   也可以用带子数计数值的二叉搜索树来做，但最坏复杂度会达到O(n)，不过本题中插入时也有可能达到O(n)

面试题10.11 峰与谷（中等）

1. 题目描述

   在一个整数数组中，“峰”是大于或等于相邻整数的元素，相应地，“谷”是小于或等于相邻整数的元素。例如，在数组{5, 8, 6, 2, 3, 4, 6}中，{8, 6}是峰， {5, 2}是谷。现在给定一个整数数组，将该数组按峰与谷的交替顺序排序。

   ```
   示例:
   输入: [5, 3, 1, 2, 3]
   输出: [5, 1, 3, 2, 3]
   ```

提示: nums.length <= 10000

2. 简单实现

顺序遍历按要求交换即可

```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        int len = nums.size();
        if(len <= 1) return;
        bool up = 1;//上升
        for(int i = 1; i < len; i++){
            if(up && nums[i] < nums[i-1])
                swap(nums[i-1], nums[i]);
            else if(!up && nums[i] > nums[i-1])
                swap(nums[i-1], nums[i]);
            up = !up;
        }
    }
};
```

也可以排序后前后穿插

面试题16.01 交换数字（中等）

1. 题目描述

编写一个函数，不用临时变量，直接交换numbers = [a, b]中a与b的值。

```
示例:
输入: numbers = [1,2]
输出: [2,1]
```

提示：numbers.length == 2

2. 简单实现

```
class Solution {
public:
    vector<int> swapNumbers(vector<int>& numbers) {
        numbers[0] = numbers[0] ^ numbers[1];
        numbers[1] = numbers[0] ^ numbers[1];
        numbers[0] = numbers[0] ^ numbers[1];
        return numbers;
    }
};
```

面试题16.02 单词频率（中等）

1. 题目描述

设计一个方法，找出任意指定单词在一本书中的出现频率。你的实现应该支持如下操作：

- WordsFrequency(book)构造函数，参数为字符串数组构成的一本书
- get(word)查询指定单词在数中出现的频率

```
示例：
WordsFrequency wordsFrequency = new WordsFrequency({"i", "have", "an", "apple",
"he", "have", "a", "pen"});
wordsFrequency.get("you"); //返回0，"you"没有出现过
wordsFrequency.get("have"); //返回2，"have"出现2次
wordsFrequency.get("an"); //返回1
wordsFrequency.get("apple"); //返回1
wordsFrequency.get("pen"); //返回1
```

提示：

- book[i]中只包含小写字母
- 1 <= book.length <= 100000
- 1 <= book[i].length <= 10
- get函数的调用次数不会超过100000

2. 简单实现

```
class WordsFrequency {
public:
    unordered_map<string, int> cnt;
    WordsFrequency(vector<string>& book) {
        for(int i = 0; i < book.size(); i++)
            cnt[book[i]]++;
    }
    int get(string word) {
        return cnt[word];
    }
};
```

面试题16.03 交点（困难）

1. 题目描述

给定两条线段（表示为起点start = {X1, Y1}和终点end = {X2, Y2}），如果它们有交点，请计算其交点，没有交点则返回空值。

要求浮点型误差不超过10^-6。若有多个交点（线段重叠）则返回X值最小的点，X坐标相同则返回Y值最小的点。

```
示例 1:
输入:
line1 = {0, 0}, {1, 0}
line2 = {1, 1}, {0, -1}
输出:  {0.5, 0}

示例 2:
输入:
```

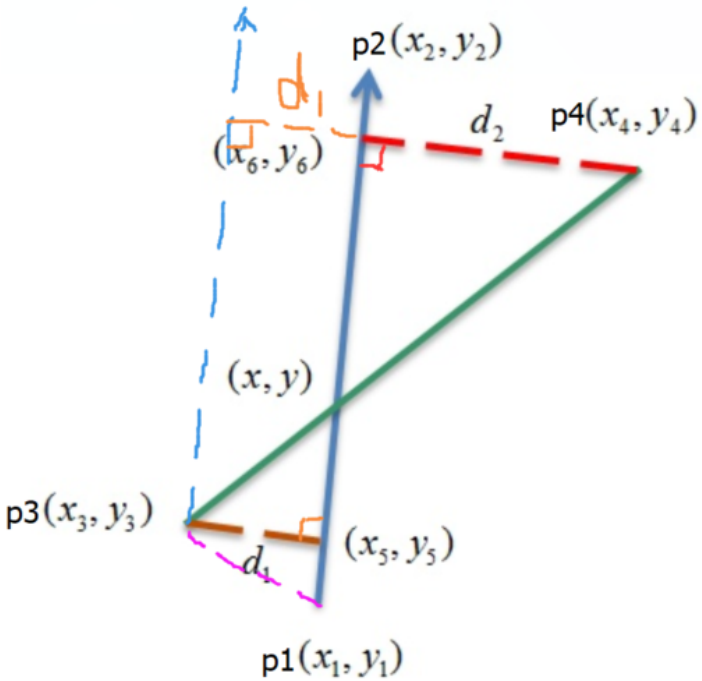
```
line1 = {0, 0}, {3, 3}
line2 = {1, 1}, {2, 2}
输出:  {1, 1}

示例 3:
输入:
line1 = {0, 0}, {1, 1}
line2 = {1, 0}, {2, 1}
输出:  {}, 两条线段没有交点
```

提示:

- 坐标绝对值不会超过2^7
- 输入的坐标均是有效的二维坐标

2. 正确解法——数学题



如上图所示，根据相似三角形可以得到以下关系式:

相似+凸优化里线段上某点坐标的求法

$$(x,y)=\frac{d1}{d1+d2}(x_4,y_4)+\frac{d2}{d1+d2}(x_3,y_3)$$

那如何求得d1/(d1+d2)，也就是比例ratio的值呢？可以利用叉乘:

见辅助线，几何意义为：两个底都是p1p2的平行四边形的面积(左)的比就是其高(右)的比

$$\frac{p1p3\times p1p2}{p3p4\times p1p2}=\frac{|p1p3||p1p2|sin\theta_1}{|p3p4||p1p2|sin\theta_2}=\frac{|p1p3|sin\theta_1}{|p3p4|sin\theta_2}=\frac{d1}{d1+d2}$$

```
static const double EPSILON = 1e-7;//浮点运算精度
struct vector2 {//定义二维坐标点
    double x, y;
```

```
    vector2(double a, double b) :x(a), y(b) {}
    vector2(const vector2& v2) :x(v2.x), y(v2.y) {}
    vector2& operator=(const vector2& v2) {
        x = v2.x;
        y = v2.y;
        return *this;
    }
    vector2 operator+(const vector2& v2) { return vector2(x + v2.x, y + v2.y); }
    vector2 operator-(const vector2& v2) { return vector2(x - v2.x, y - v2.y); }
    double operator*(const vector2& v2) { return x * v2.x + y * v2.y; }//点积
    vector2 operator*(double n) { return vector2(x * n, y * n); }
    double operator^(const vector2& v2) { return x * v2.y - y * v2.x; }//叉乘
    bool operator==(const vector2& v2)
        { return fabs(x - v2.x) < EPSILON && fabs(y - v2.y) < EPSILON; }
    bool operator < (const vector2& v2) {
        if (x < v2.x) return true;
        if (fabs(x - v2.x)<EPSILON) return y < v2.y;
        return false;
    }
    bool operator <= (const vector2& v2) {
        if ((*this < v2) || (*this == v2)) return true;
        else return false;
    }
};
class Solution {
public:
    vector<double> intersection(vector<int>& start1, vector<int>& end1,
                                vector<int>& start2, vector<int>& end2) {
        vector2 p1(start1[0], start1[1]), p2(end1[0], end1[1]);
        vector2 p3(start2[0], start2[1]), p4(end2[0], end2[1]);
        if (p2 < p1)swap(p1, p2);
        if (p4 < p3)swap(p3, p4);
        vector2 p1p2 = p2 - p1, p3p4 = p4 - p3, p3p1 = p1 - p3, p4p1 = p1 - p4;

        if (fabs(p1p2 ^ p3p4) < EPSILON) {//所在直线平行或重合
            if (fabs(p4p1 ^ p3p1) < EPSILON) {//所在直线重合
                if (p4 < p1 || p2 < p3) return {};//线段不重合
                else {//线段有重合
                    if (p1 < p3) return { p3.x,p3.y };
                    else return { p1.x, p1.y };
                }
            }
            else return {};//所在直线平行，无交点
        }
        else {//所在直线有一个交点
            double ratio = (p3p1 ^ p1p2) / (p3p4 ^ p1p2);
            vector2 ans = p3 * (1 - ratio) + p4 * ratio;
            if (p1 <= ans && ans <= p2 && p3 <= ans && ans <= p4)//在线段上
                return { ans.x,ans.y };
            else return {};
        }
    }
};
```

面试题16.04 井字游戏（中等）

1. 题目描述

   设计一个算法，判断玩家是否赢了井字游戏。输入是一个 N x N 的数组棋盘，由字符" "，"X"和"O"组成，其中字符" "代表一个空位。以下是井字游戏的规则：

   - 玩家轮流将字符放入空位（" "）中。
   - 第一个玩家总是放字符"O"，且第二个玩家总是放字符"X"。
   - "X"和"O"只允许放置在空位中，不允许对已放有字符的位置进行填充。
   - 当有N个相同（且非空）的字符填充任何行、列或对角线时，游戏结束，对应该字符的玩家获胜。
   - 当所有位置非空时，也算为游戏结束。
   - 如果游戏结束，玩家不允许再放置字符。

   如果游戏存在获胜者，就返回该游戏的获胜者使用的字符（"X"或"O"）；如果游戏以平局结束，则返回 "Draw"；如果仍会有行动（游戏未结束），则返回 "Pending"

   ```
   示例 1:
   输入:  board = ["O X"," XO","X O"]
   输出:  "X"

   示例 2:
   输入:  board = ["OOX","XXO","OXO"]
   输出:  "Draw"
   解释:  没有玩家获胜且不存在空位

   示例 3:
   输入:  board = ["OOX","XXO","OX "]
   输出:  "Pending"
   解释:  没有玩家获胜且仍存在空位
   ```

   提示：

   - 1 <= board.length == board[i].length <= 100
   - 输入一定遵循井字棋规则

2. 简单实现

```cpp
class Solution {
public:
    string tictactoe(vector<string>& board) {
        int n = board.size();
        string ans = "";
        bool full = true;//记录棋盘是否已满
        for(int i = 0; i < n; i++){//各行
            char cur = board[i][0];
            int j = 0;
            for(j = 0; j < n; j++){
                if(board[i][j] == ' '){
                    full = false;//未满
                    break;
                }
                if(board[i][j] != cur) break;//出现不同元素
```

```
                }
                if(j == n){
                    ans += cur;
                    return ans;
                }
            }
            for(int j = 0; j < n; j++){//各列
                char cur = board[0][j];
                int i = 0;
                for(i = 0; i < n; i++)
                    if(board[i][j] == ' ' || board[i][j] != cur) break;
                if(i == n) {
                    ans += cur;
                    return ans;
                }
            }
            //右下对角线
            char cur = board[0][0];
            int i = 0;
            for(i = 0; i < n; i++)
                if(board[i][i] == ' ' || board[i][i] != cur) break;
            if(i == n){
                ans += cur;
                return ans;
            }
            //右上对角线
            cur = board[0][n-1];
            for(i = 0; i < n; i++)
                if(board[i][n-i-1] == ' ' || board[i][n-i-1] != cur) break;
            if(i == n){
                ans += cur;
                return ans;
            }
            if(full)
                return "Draw";
            else
                return "Pending";
        }
    };
```

面试题16.05 阶乘尾数（简单）

1. 题目描述

   设计一个算法，算出 n 阶乘有多少个尾随零。

```
示例 1:
输入: 3
输出: 0
解释: 3! = 6, 尾数中没有零。

示例 2:
输入: 5
输出: 1
解释: 5! = 120, 尾数中有 1 个零.
说明: 你算法的时间复杂度应为 O(log n) 。
```

2. 简单实现

```
class Solution {
public:
    int trailingZeroes(int n) {
        int ans = 0;
        while(n = n/5)
            ans += n;
        return ans;
    }
};
```

面试题16.06 最小差（中等）

1. 题目描述

给定两个整数数组a和b，计算具有最小差绝对值的一对数值（每个数组中取一个值），并返回该对数值的差

```
示例:
输入: {1, 3, 15, 11, 2}, {23, 127, 235, 19, 8}
输出:  3, 即数值对(11, 8)
```

提示:

- 1 <= a.length, b.length <= 100000
- -2147483648 <= a[i], b[i] <= 2147483647
- 正确结果在区间[-2147483648, 2147483647]内

2. 简单实现

格外注意一下溢出问题就好，需要加强对溢出的敏感

```
class Solution {
public:
    int smallestDifference(vector<int>& a, vector<int>& b) {
        sort(a.begin(), a.end());
        sort(b.begin(), b.end());
        int idx_a = 0, idx_b = 0;
        long ans = 2147483647;
        while(idx_a < a.size() && idx_b < b.size()){//每次把小的那个数所在数组的索引后移
            ans = min(ans, long(abs(long(a[idx_a]) - long(b[idx_b]))));
```

```
            if(a[idx_a] < b[idx_b])
                idx_a++;
            else if(a[idx_a] > b[idx_b])
                idx_b++;
            else//相等，已经是最小了
                return 0;
        }
        //根据索引后移条件，即使有剩余也不可能产生更小的差的绝对值
        return ans;
    }
};
```

3. 改进——使用lower_bound更新索引

```
class Solution {
public:
    int smallestDifference(vector<int>& a, vector<int>& b) {
        sort(a.begin(), a.end());
        sort(b.begin(), b.end());
        int idx_a = 0, idx_b = 0;
        long ans = 2147483647;
        while(idx_a < a.size() && idx_b < b.size()){//每次把小的那个数所在数组的索引后移
            if(a[idx_a] < b[idx_b]){
                idx_a = lower_bound(a.begin(), a.end(), b[idx_b]) - a.begin() -
1;//最后一个小于b的
                ans = min(ans, long(abs(long(a[idx_a]) - long(b[idx_b]))));
                idx_a++;//下一个大于b
            }
            else if(a[idx_a] > b[idx_b]){
                idx_b = lower_bound(b.begin(), b.end(), a[idx_a]) - b.begin() - 1;
                ans = min(ans, long(abs(long(a[idx_a]) - long(b[idx_b]))));
                idx_b++;
            }
            else//相等，已经是最小了
                return 0;
        }
        //根据索引后移条件，即使有剩余也不可能产生更小的差的绝对值
        return ans;
    }
};
```

面试题16.07 最大数值（简单）

1. 题目描述

编写一个方法，找出两个数字 `a` 和 `b` 中最大的那一个。不得使用if-else或其他比较运算符。

示例：

```
输入：  a = 1, b = 2
输出：  2
```

2. 解法一

- 本质是**平均值法**：`max(a, b) = ((a + b) + abs(a - b)) / 2`
- 但是abs内部也可能调用了比较符号，所以用位运算实现绝对值运算

绝对值的位运算

> 为了回避`abs`，利用位运算实现绝对值功能。
> 以`int8_t`为例：分析运算：`(var ^ (var >> 7)) - (var >> 7)`
>
> - **var >= 0**：`var >> 7 => 0x00`，即：`(var ^ 0x00) - 0x00`，异或结果为`var`
> - **var < 0**：`var >> 7 => 0xFF`，即：`(var ^ 0xFF) - 0xFF`，`var ^ 0xFF`是在对var的全部位取反，`-0xFF <=> +1`，对`signed int`取反加一就是取其**相反数**。

举个栗子🌰：`var = -3 <=> 0xFD`，`(var ^ 0xFF) - 0xFF= 0x02 - 0xff= 0x03`

> 基于上述分析：
>
类型	绝对值位运算
int8_t	(var ^ (var >> 7)) - (var >> 7)
int16_t	(var ^ (var >> 15)) - (var >> 15)
int32_t	(var ^ (var >> 31)) - (var >> 31)
int64_t	(var ^ (var >> 63)) - (var >> 63)

代码中`(_diff ^ (_diff >> 63)) - (_diff >> 63)`就是在求取`long (int64_t)`的绝对值。

```
class Solution {
public:
    int maximum(int a, int b) {
        long sum = long(a) + long(b);
        long diff = long(a) - long(b);
        long abs_diff = (diff ^ (diff >> 63)) - (diff >> 63);
        return (sum + abs_diff) / 2;
    }
};
```

3. 解法二

```
class Solution {
public:
    int maximum(int a, int b) {
        long k = (((long)a - (long)b) >> 63) & 1;//求a-b的符号
        return b * k + a * (k ^ 1);//k为0则a>b，否则a<b
    }
};
```

面试题16.08 整数的英语表示（困难）

1. 题目描述

给定一个整数，打印该整数的英文描述。

```
示例 1:
输入: 123
输出: "One Hundred Twenty Three"

示例 2:
输入: 12345
输出: "Twelve Thousand Three Hundred Forty Five"

示例 3:
输入: 1234567
输出: "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

示例 4:
输入: 1234567891
输出: "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven Thousand
Eight Hundred Ninety One"
```

2. 简单实现

三个数三个数地处理

```
class Solution {
public:
    vector<string> dic1 = {"Zero", "One", "Two", "Three", "Four", "Five", "Six",
"Seven", "Eight", "Nine", "Ten", "Eleven", "Twelve",  "Thirteen", "Fourteen",
"Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"};
    vector<string> dic2 = {"","","Twenty", "Thirty", "Forty", "Fifty", "Sixty",
"Seventy", "Eighty", "Ninety"};
    string get3len(int num){//三位数以内的数字翻译
        string ans = "";
        if(num < 20)
            return dic1[num];
        else if(num < 100){
            ans += dic2[num/10];
            num %= 10;
            if(num > 0)
                ans += " " + get3len(num);
        }
        else{
            ans += dic1[num/100] + " Hundred";
            num %= 100;
            if(num > 0)
                ans += " " + get3len(num);
        }
        return ans;
    }
    string numberToWords(int num) {
        string ans = "";
        if(num < 1000)
            return get3len(num);
        else if(num < 1000000){
```

```
            ans += get3len(num / 1000) + " Thousand";
            num %= 1000;
            if(num > 0)
                ans += " " + numberToWords(num);
        }
        else if(num < 1000000000){
            ans += get3len(num / 1000000) + " Million";
            num %= 1000000;
            if(num > 0)
                ans += " " + numberToWords(num);
        }
        else{
            ans += get3len(num / 1000000000) + " Billion";
            num %= 1000000000;
            if(num > 0)
                ans += " " + numberToWords(num);
        }
        return ans;
    }
};
```

面试题16.10 生存人数（中等）

1. 题目描述

   给定N个人的出生年份和死亡年份，第i个人的出生年份为birth[i]，死亡年份为death[i]，实现一个方法以计算生存人数最多的年份。

   你可以假设所有人都出生于1900年至2000年（含1900和2000）之间。如果一个人在某一年的任意时期都处于生存状态，那么他们应该被纳入那一年的统计中。例如，生于1908年、死于1909年的人应当被列入1908年和1909年的计数。

   如果有多个年份生存人数相同且均为最大值，输出其中最小的年份。

   ```
   示例：
   输入：
   birth = {1900, 1901, 1950}
   death = {1948, 1951, 2000}
   输出：  1901
   ```

   提示：

   - 0 < birth.length == death.length <= 10000
   - birth[i] <= death[i]

2. 简单实现

   `第i年生存人数=第1900~i年出生的人数-第1900~i-1年死亡的人数`，因此可以动态存储遍历（专业说法是前缀和？）

   ```
   class Solution {
   public:
       int maxAliveYear(vector<int>& birth, vector<int>& death) {
           vector<int> b(101, 0);//各年出生的人数
   ```

```
        vector<int> d(101, 0);//各年死亡的人数
        for(int i = 0; i < birth.size(); i++){
            b[birth[i]-1900]++;
            d[death[i]-1900]++;
        }
        int ans = 1900;
        int max_alive = birth[0];
        int cur_alive = max_alive;
        for(int i = 1; i <= 100; i++){
            cur_alive += b[i] - d[i-1];
            if(cur_alive > max_alive){
                max_alive = cur_alive;
                ans = 1900 + i;
            }
        }
        return ans;
    }
};
```

改进点：直接存储每年的人数变化，可能省下100个int的存储空间

面试题16.11 跳水版（简单）

1. 题目描述

   你正在使用一堆木板建造跳水板。有两种类型的木板，其中长度较短的木板长度为shorter，长度较长的木板长度为longer。你必须正好使用k块木板。编写一个方法，生成跳水板所有可能的长度。

   返回的长度需要从小到大排列。

   ```
   示例
   输入：
   shorter = 1
   longer = 2
   k = 3
   输出：  {3,4,5,6}
   ```

   提示：

   - 0 < shorter <= longer
   - 0 <= k <= 100000

2. 简单实现

```
class Solution {
public:
    vector<int> divingBoard(int shorter, int longer, int k) {
        if(k == 0) return {};//特殊情况的判断
        else if( shorter == longer) return {shorter*k};//特殊情况的判断，注意读题
        vector<int> ans(k+1);
        for(int cnt = 0; cnt <= k; cnt++)
            ans[cnt] = longer*cnt + shorter*(k-cnt);
        return ans;
    }
};
```

面试题16.13 平分正方形（中等）

1. 题目描述

   给定两个正方形及一个二维平面。请找出将这两个正方形分割成两半的一条直线。假设正方形顶边和底边与 x 轴平行。

   每个正方形的数据square包含3个数值，正方形的左下顶点坐标[X,Y] = [square[0],square[1]]，以及正方形的边长square[2]。所求直线穿过两个正方形会形成4个交点，请返回4个交点形成线段的两端点坐标（两个端点即为4个交点中距离最远的2个点，这2个点所连成的线段一定会穿过另外2个交点）。2个端点坐标[X1,Y1]和[X2,Y2]的返回格式为{X1,Y1,X2,Y2}，要求若X1 != X2，需保证X1 < X2，否则需保证Y1 <= Y2。

   若同时有多条直线满足要求，则选择斜率最大的一条计算并返回（与Y轴平行的直线视为斜率无穷大）。

   ```
   示例:
   输入:
   square1 = {-1, -1, 2}
   square2 = {0, -1, 2}
   输出:  {-1,0,2,0}
   解释:  直线 y = 0 能将两个正方形同时分为等面积的两部分，返回的两线段端点为[-1,0]和[2,0]
   ```

   提示：

   - square.length == 3
   - square[2] > 0

2. 正确解法

   又是恶心的数学题，关键在于要发现：能平分两个正方形的直线要同时通过两个正方形的中心，然后根据斜率判断截取情况即可

   ```
   class Solution {
   public:
       vector<double> cutSquares(vector<int>& square1, vector<int>& square2) {
           double c1_x = square1[0] + double(square1[2]) / 2;
           double c1_y = square1[1] + double(square1[2]) / 2;
           double c2_x = square2[0] + double(square2[2]) / 2;
           double c2_y = square2[1] + double(square2[2]) / 2;
           if(abs(c1_x - c2_x) < 1e-7)//竖直
               return {c1_x, min(square1[1],square2[1]),
                       c1_x, max(square1[1]+square1[2], square2[1]+square2[2])};
   ```

```
        else if(abs(c1_y - c2_y) < 1e-7)//水平
            return {min(square1[0],square2[0]),
                    c1_y, max(square1[0]+square1[2], square2[0]+square2[2]), c1_y};
        double k = (c2_y - c1_y) / (c2_x - c1_x);//斜率
        if(abs(k) >= 1){//截取正方形为左右两部分，截取点分布在上下两条边
            //正方形之间可能存在嵌套关系，因此边界点的判断需要讨论
            double up_x, up_y;
            if(square1[1]+square1[2] > square2[1]+square2[2]){//正方形1的上边更高
                up_x = c1_x + square1[2] / k / 2.0;//画图可算
                up_y = square1[1]+square1[2];
            }
            else{//正方形2的上边更高
                up_x = c2_x + square2[2] / k / 2;
                up_y = square2[1]+square2[2];
            }
            double down_x, down_y;
            if(square1[1] < square2[1]){//正方形1的下边更低
                down_x = c1_x - square1[2] / k / 2;
                down_y = square1[1];
            }
            else{//正方形2的下边更低
                down_x = c2_x - square2[2] / k / 2;
                down_y = square2[1];
            }
            if(down_x < up_x)
                return {down_x, down_y, up_x, up_y};
            else
                return {up_x, up_y, down_x, down_y};
        }
        else{//截取正方形为上下两部分，截取点分布在左右两条边
            double left_x, left_y;
            if(square1[0] < square2[0]){
                left_x = square1[0];
                left_y = c1_y - square1[2] * k / 2;
            }
            else{
                left_x = square2[0];
                left_y = c2_y - square2[2] * k / 2;
            }
            double right_x, right_y;
            if(square1[0]+square1[2] > square2[0]+square2[2]){
                right_x = square1[0]+square1[2];
                right_y = c1_y + square1[2] * k / 2;
            }
            else{
                right_x = square2[0]+square2[2];
                right_y = c2_y + square2[2] * k / 2;
            }
            return {left_x, left_y, right_x, right_y};
        }
        return {};
    }
};
```

面试题16.14 最佳直线（中等）

1. 题目描述

   给定一个二维平面及平面上的 N 个点列表Points，其中第i个点的坐标为Points[i]=[Xi,Yi]。请找出一条直线，其通过的点的数目最多。

   设穿过最多点的直线所穿过的全部点编号从小到大排序的列表为S，你仅需返回[S[0],S[1]]作为答案，若有多条直线穿过了相同数量的点，则选择S[0]值较小的直线返回，S[0]相同则选择S[1]值较小的直线返回。

   ```
   示例:
   输入:  [[0,0],[1,1],[1,0],[2,0]]
   输出:  [0,2]
   解释:  所求直线穿过的3个点的编号为[0,2,3]
   ```

   提示：

   - 2 <= len(Points) <= 300
   - len(Points[i]) = 2

2. 简单实现

   直接暴力求解，每个两点组成的直线，看有多少点落在上面；用乘法的方式防止对水平和竖直的额外判断

   ```
   class Solution {
   public:
       vector<int> bestLine(vector<vector<int>>& points) {
           vector<int> ans;
           int max_num = 0;
           for(int i = 0; i+1 < points.size(); ++i){
               for(int j = i + 1; j+1 < points.size(); ++j){
                   int cur_num = 2;//目前两点在直线上
                   long dx = points[j][0] - points[i][0];//用long防止之后乘法溢出
                   long dy = points[j][1] - points[i][1];
                   for(int k = j + 1; k < points.size(); k++){//只需要判断j后面的点即可
                       if(dx*(points[k][1]-points[j][1]) == dy*(points[k][0]-points[j]
   [0]))
                           cur_num++;
                   }
                   if(cur_num > max_num){
                       max_num = cur_num;
                       ans = {i, j};
                   }
               }
           }
           return ans;
       }
   };
   ```

3. 其他解法

   根据斜率排序，再数斜率相同的直线数目，这样子时间复杂度其实差不远，是`n^2*log_2_n`

```cpp
class Solution {
    const double MIN=1e-5;//这是避免double精度问题，用来进行比较的常数
    int n;
    vector<int> Ans;//这就是存储答案的vector
    double slope[310][310];//存储斜率的二维表
    double abs(double x){//手写的绝对值函数
        return (x>0)?x:-x;
    }
    double Calc(vector<int> a,vector<int> b){//计算两个点直线的斜率
        double dx=a[0]-b[0];
        double dy=a[1]-b[1];
        if(abs(dx)<MIN)
        return -37;//这里的-37是我随便取的特殊值，用来处理斜率无穷大的情况
        return dy/dx;
    }
public:
    vector<int> bestLine(vector<vector<int>>& points) {
        n=points.size();
        memset(slope,-1,sizeof(slope));

        for(int i=0;i<n;i++)
            for(int j=i+1;j<n;j++)
                slope[i][j]=Calc(points[i],points[j]);//先预处理每两个点的斜率
        int ans=0;
        for(int i=0;i<n;i++){
            sort(&slope[i][i+1],&slope[i][n]);//对于每个点points[i]，都进行排序，便于查找斜率相同的点
            slope[i][n]=-31.4;//这里的-31.4也只是个特殊值
            int len=2;
            for(int j=i+2;j<=n;j++)
                if(abs(slope[i][j]-slope[i][j-1])<MIN)
                len++;
                else{
                    if(len>ans){
                        ans=len;
                        for(int k=i+1;k<n;k++){//由于排序打乱了原有的顺序，所以需要这样扫一遍找到对应斜率的点
                            double sp=Calc(points[i],points[k]);
                            if(abs(sp-slope[i][j-1])<MIN){
                                Ans.clear();
                                Ans.push_back(i);
                                Ans.push_back(k);
                                break;
                            }
                        }
                    }
                    len=2;
                }
        }
        return Ans;
    }
};
```

面试题16.15 珠玑妙算（简单）

1. 题目描述

   珠玑妙算游戏（the game of master mind）的玩法如下。

   计算机有4个槽，每个槽放一个球，颜色可能是红色（R）、黄色（Y）、绿色（G）或蓝色（B）。例如，计算机可能有RGGB 4种（槽1为红色，槽2、3为绿色，槽4为蓝色）。作为用户，你试图猜出颜色组合。打个比方，你可能会猜YRGB。要是猜对某个槽的颜色，则算一次“猜中”；要是只猜对颜色但槽位猜错了，则算一次“伪猜中”。注意，“猜中”不能算入“伪猜中”。

   给定一种颜色组合solution和一个猜测guess，编写一个方法，返回猜中和伪猜中的次数answer，其中answer[0]为猜中的次数，answer[1]为伪猜中的次数。

   ```
   示例:
   输入:  solution="RGBY",guess="GGRR"
   输出:  [1,1]
   解释:  猜中1次，伪猜中1次。
   ```

   提示：

   - len(solution) = len(guess) = 4
   - solution和guess仅包含"R","G","B","Y"这4种字符

2. 简单实现

```
class Solution {
public:
    vector<int> masterMind(string solution, string guess) {
        vector<int> re;
        int arr1[27] = {0}, arr2[27] = {0}, count = 0;
        for(int i = 0; i < 4; i++){
            if(solution[i] == guess[i]) count++;
            else{
                arr1[solution[i]-65]++;
                arr2[guess[i]-65]++;
            }
        }
        re.push_back(count);
        count = 0;
        for(int i = 0; i < 27; i++)
            if(arr1[i]!=0 && arr2[i]!=0)
                count += min(arr1[i], arr2[i]);
        re.push_back(count);
        return re;
    }
};
```

面试题16.16 部分排序（中等）

1. 题目描述

给定一个整数数组，编写一个函数，找出索引m和n，只要将索引区间[m,n]的元素排好序，整个数组就是有序的。注意：n-m尽量最小，也就是说，找出符合条件的最短序列。函数返回值为[m,n]，若不存在这样的m和n（例如整个数组是有序的），请返回[-1,-1]。

```
示例:
输入:  [1,2,4,7,10,11,7,12,6,7,16,18,19]
输出:  [3,9]
```

提示：0 <= len(array) <= 1000000

2. 简单实现

排序前后个位置数比较，不一样的地方就是需要调整的地方

```
class Solution {
public:
    vector<int> subSort(vector<int>& array) {
        vector<int> back = array;
        sort(array.begin(), array.end());
        int l = 0;
        while(l < array.size() && array[l] == back[l]) l++;
        if(l == array.size()) return {-1, -1};
        int r = array.size() - 1;
        while(array[r] == back[r]) r--;
        return {l, r};
    }
};
```

3. 自我改进

```
class Solution {
public:
    vector<int> subSort(vector<int>& array) {
        int l = 0, r = array.size() - 1;
        while(l < r && array[l] <= array[l+1]) l++;//从左向右找到第一个不符合上升趋势的点
        if(l >= r) return {-1,-1};//全部升序
        while(array[r] >= array[r-1]) r--;//从右向左找到第一个不符合下降趋势的点
        //l与r之间是一定会被改的，找到这之间的最小值和最大值，则最小值在左半部的插入点就是最短序列的左端点，最大值在右半部的插入点就是最短序列的右端点
        int cur = l;
        int max_num = array[l];
        int min_num = array[l];
        while(cur <= r){
            max_num = max(max_num, array[cur]);
            min_num = min(min_num, array[cur]);
            cur++;
        }
        //左端点，找upper_bound
        int l_idx = upper_bound(array.begin(), array.begin()+l+1, min_num) - array.begin();
        //右端点，找lower_bound
        int r_idx = lower_bound(array.begin()+r, array.end(), max_num) - array.begin();
```

```
        return {l_idx, r_idx-1};//右端点要-1, lower_bound的位置是不需要修改的
    }
};
```

面试题16.17 连续数列（简单）

1. 题目描述

   给定一个整数数组（有正数有负数），找出总和最大的连续数列，并返回总和。

```
输入:  [-2,1,-3,4,-1,2,1,-5,4]
输出:  6
解释:  连续子数组 [4,-1,2,1] 的和最大，为 6。
```

   进阶：如果你已经实现复杂度为 O(n) 的解法，尝试使用更为精妙的分治法求解。

2. 简单实现——O(n)算法

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int ans = INT_MIN;
        int cur = 0;
        for(int i = 0; i < nums.size(); i++){
            cur += nums[i];
            ans = max(ans, cur);//先加和判断可以处理全为负的情况
            if(cur < 0)
                cur = 0;
        }
        return ans;
    }
};
```

面试题16.18 模式匹配（中等）

1. 题目描述

   你有两个字符串，即pattern和value。 pattern字符串由字母"a"和"b"组成，用于描述字符串中的模式。例如，字符串"catcatgocatgo"匹配模式"aabab"（其中"cat"是"a"，"go"是"b"），该字符串也匹配像"a"、"ab"和"b"这样的模式。但需注意"a"和"b"不能同时表示相同的字符串。编写一个方法判断value字符串是否匹配pattern字符串。

```
示例 1:
输入:  pattern = "abba", value = "dogcatcatdog"
输出:  true

示例 2:
输入:  pattern = "abba", value = "dogcatcatfish"
输出:  false
```

```
示例 3:
输入:  pattern = "aaaa", value = "dogcatcatdog"
输出:  false

示例 4:
输入:  pattern = "abba", value = "dogdogdogdog"
输出:  true
解释:  "a"="dogdog",b="",反之也符合规则
```

提示:

- 0 <= len(pattern) <= 1000
- 0 <= len(value) <= 1000
- 你可以假设pattern只包含字母"a"和"b",value仅包含小写字母。

2. 简单实现

```
class Solution {
public:
    bool patternMatching(string pattern, string value) {
        int len = value.size();
        int a_cnt = 0, b_cnt = 0;//模式串中a和b的个数
        for(int i = 0; i < pattern.size(); ++i)
            if(pattern[i] == 'a') a_cnt++;
            else b_cnt++;
        if(a_cnt == 0 && b_cnt == 0) return len==0;//pattern为空时,value必须是空的
        else if(len == 0) return a_cnt==0 || b_cnt == 0;//value为空时,a和b只能存在一
种
        if(a_cnt == 0){//只有b
            if(len % b_cnt != 0) return false;//不能整除,无法划分
            string b = value.substr(0, len/b_cnt);//b对应的字符串
            string cur = "";
            while(b_cnt--) cur += b;
            return cur == value;
        }
        if(b_cnt == 0){//与只有b同理
            if(len % a_cnt != 0) return false;
            string a = value.substr(0, len/a_cnt);
            string cur = "";
            while(a_cnt--) cur += a;
            return cur == value;
        }
        //a,b都有,需要满足a_len*a_cnt + b_len*b_cnt = len的方案才有可能成立
        for(int a_len = 0; a_len <= len/a_cnt; ++a_len){//a字符串可能的长度
            if((len - a_len*a_cnt) % b_cnt== 0){//对应的b字符串的长度解为整数
                int b_len = (len - a_len*a_cnt) / b_cnt;
                //有了a,b的长度,可以得到a,b对应的字符串
                int i = 0;
                while(pattern[i] != 'a') i++;
                string a = value.substr(i*b_len, a_len);
                i = 0;
                while(pattern[i] != 'b') i++;
                string b = value.substr(i*a_len, b_len);
                string cur = "";
```

```
                for(int i = 0; i < pattern.size(); i++)//根据模式串指示构造当前方案形成
的字符串
                    if(pattern[i] == 'a') cur += a;
                    else cur += b;
                if(cur == value) return true;
            }
        }
        return false;
    }
};
```

面试题16.19 水域大小（中等）

1. 题目描述

   你有一个用于表示一片土地的整数矩阵land，该矩阵中每个点的值代表对应地点的海拔高度。若值为0则表示水域。由垂直、水平或对角连接的水域为池塘。池塘的大小是指相连接的水域的个数。编写一个方法来计算矩阵中所有池塘的大小，返回值需要从小到大排序。

   ```
   示例：
   输入：
   [
     [0,2,1,0],
     [0,1,0,1],
     [1,1,0,1],
     [0,1,0,1]
   ]
   输出：  [1,2,4]
   ```

   提示：

   - 0 < len(land) <= 1000
   - 0 < len(land[i]) <= 1000

2. 简单实现

```
class Solution {
public:
    vector<vector<int>> dir = {{-1,0}, {1,0}, {0,-1}, {0,1}, {-1,1}, {1,1},
{-1,-1}, {1,-1}};
    vector<int> pondSizes(vector<vector<int>>& land) {
        vector<int> ans;
        int m = land.size();
        if(m == 0) return {};
        int n = land[0].size();
        if(n == 0) return {};
        for(int i = 0; i < m; i++)
            for(int j = 0; j < n; j++){
                if(land[i][j] == 0){
                    queue<vector<int>> q;
                    q.push({i,j});
                    land[i][j] = 1;
```

```
                    int tmp_num = 0;
                    while(!q.empty()){
                        tmp_num++;
                        int x = q.front()[0];
                        int y = q.front()[1];
                        q.pop();
                        for(int k = 0; k < dir.size(); k++){
                            int xx = x + dir[k][0];
                            int yy = y + dir[k][1];
                            if(xx >= 0 && xx < m && yy >= 0 && yy < n && land[xx]
[yy] == 0){
                                land[xx][yy] = 1;
                                q.push({xx,yy});
                            }
                        }
                    }
                    ans.push_back(tmp_num);
                }
            }
        sort(ans.begin(), ans.end());
        return ans;
    }
};
```

面试题16.20 T9键盘（中等）

1. 题目描述

   在老式手机上，用户通过数字键盘输入，手机将提供与这些数字相匹配的单词列表。每个数字映射到0至4个字母。给定一个数字序列，实现一个算法来返回匹配单词的列表。你会得到一张含有有效单词的列表。映射如下图所示：

```
示例 1:
输入: num = "8733", words = ["tree", "used"]
输出: ["tree", "used"]

示例 2:
输入: num = "2", words = ["a", "b", "c", "d"]
输出: ["a", "b", "c"]
```

   提示：

   - num.length <= 1000
   - words.length <= 500
   - words[i].length == num.length
   - num中不会出现 0, 1 这两个数字

2. 简单实现

```
class Solution {
public:
    vector<string> getValidT9Words(string num, vector<string>& words) {
```

```
        unordered_map<char, char> m;
        m['a'] = m['b'] = m['c'] = '2';
        m['d'] = m['e'] = m['f'] = '3';
        m['g'] = m['h'] = m['i'] = '4';
        m['j'] = m['k'] = m['l'] = '5';
        m['m'] = m['n'] = m['o'] = '6';
        m['p'] = m['q'] = m['r'] = m['s'] = '7';
        m['t'] = m['u'] = m['v'] = '8';
        m['w'] = m['x'] = m['y'] = m['z'] = '9';
        vector<string> ans;
        int len = num.size();
        for(int i = 0; i < words.size(); i++){
            string cur = "";
            if(words[i].size() != len) continue;
            int j;
            for(j = 0; j < len; j++)
                if(m[words[i][j]] != num[j])
                    break;
            if(j == len)
                ans.push_back(words[i]);
        }
        return ans;
    }
};
```

面试题16.21 交换和（中等）

1. 题目描述

   给定两个整数数组，请交换一对数值（每个数组中取一个数值），使得两个数组所有元素的和相等。

   返回一个数组，第一个元素是第一个数组中要交换的元素，第二个元素是第二个数组中要交换的元素。若有多个答案，返回任意一个均可。若无满足条件的数值，返回空数组。

```
示例:
输入: array1 = [4, 1, 2, 1, 1, 2], array2 = [3, 6, 3, 3]
输出: [1, 3]

示例:
输入: array1 = [1, 2, 3], array2 = [4, 5, 6]
输出: []
```

   提示：1 <= array1.length, array2.length <= 100000

2. 简单实现

   注意题意中，要返回的是交换的数字，而不是它们的下标

```
class Solution {
public:
    vector<int> findSwapValues(vector<int>& array1, vector<int>& array2) {
        unordered_set<int> nums;//记录array2中有的数字
        int sum1 = 0, sum2 = 0;
```

```
        for(int i = 0; i < array1.size(); i++)
            sum1 += array1[i];
        for(int i = 0; i < array2.size(); i++){
            sum2 += array2[i];
            nums.insert(array2[i]);
        }
        if(abs(sum1-sum2) % 2 != 0) return {};//差值必须为偶数才有可能交换成功
        int diff = (sum2-sum1) / 2;
        for(int i = 0; i < array1.size(); i++){
            if(nums.count(array1[i] + diff) > 0)
                return {array1[i], array1[i]+diff};
        }
        return {};
    }
};
```

面试题16.22 兰顿蚂蚁（中等）

1. 题目描述

一只蚂蚁坐在由白色和黑色方格构成的无限网格上。开始时，网格全白，蚂蚁面向右侧。每行走一步，蚂蚁执行以下操作。

(1) 如果在白色方格上，则翻转方格的颜色，向右(顺时针)转 90 度，并向前移动一个单位。 (2) 如果在黑色方格上，则翻转方格的颜色，向左(逆时针方向)转 90 度，并向前移动一个单位。

编写程序来模拟蚂蚁执行的前 K 个动作，并返回最终的网格。

网格由数组表示，每个元素是一个字符串，代表网格中的一行，黑色方格由 'X' 表示，白色方格由 '_' 表示，蚂蚁所在的位置由 'L', 'U', 'R', 'D' 表示，分别表示蚂蚁 左、上、右、下 的朝向。只需要返回能够包含蚂蚁走过的所有方格的最小矩形。

```
示例 1:
输入: 0
输出: ["R"]

示例 2:
输入: 2
输出:
[
  "_X",
  "LX"
]

示例 3:
输入: 5
输出:
[
  "U",
  "X",
  "XX"
]
```

说明：K <= 100000

2. 简单实现

暴力模拟，注意节省内存

```
class Solution {
public:
    vector<vector<int>> dirs = {{0,1}, {1,0}, {0,-1}, {-1,0}};//按顺时针的变化存储各个方向
    vector<char> loc = {'R', 'D', 'L', 'U'};
    vector<string> printKMoves(int K) {
        vector<string> ans = {"_"};//当前地图
        int x = 0, y = 0;//当前坐标
        int dir = 0;//当前方向
        while(K--){
            if(ans[x][y] == '_'){
                ans[x][y] = 'X';
                dir = (dir+1) % dirs.size();//顺时针
                x += dirs[dir][0];
                y += dirs[dir][1];
            }
            else{
                ans[x][y] = '_';
                dir = (dir+3) % dirs.size();//逆时针
                x += dirs[dir][0];
                y += dirs[dir][1];
            }
            //处理出界情况
            if(x < 0){//上界
                // string cur(ans[0].size(), '_'); //这样写再insert会超内存
                ans.insert(ans.begin(), string(ans[0].size(), '_'));
                x++;
            }
            else if(x >= ans.size()){//下界
                // string cur(ans[0].size(), '_');
                ans.push_back(string(ans[0].size(), '_'));
            }
            else if(y < 0){//左界
                for(int i = 0; i < ans.size(); i++)
                    ans[i].insert(ans[i].begin(), '_');// = "_" + ans[i];//insert更省内存
                y++;
            }
            else if(y >= ans[0].size()){//右界
                for(int i = 0; i < ans.size(); i++)
                    ans[i] += '_';
            }
        }
        ans[x][y] = loc[dir];//在蚂蚁当前位置写下当前方向
        return ans;
    }
};
```

3. 最优解法

   也可以用hashmap存储变黑的点和走过的边界，最后统一构造答案

```
class Solution {
public:
    vector<string> printKMoves(int K) {
        set<pair<int,int>> black;
        if(K == 0) return vector<string>{"R"};
        vector<char> direction = {'L', 'U', 'R', 'D'};
        vector<int> hori = {-1, 0, 1, 0};
        vector<int> vert = {0, 1, 0, -1};
        int cur_x = 1, cur_y = 0;
        int cur_direction = 2;
        int top_left = 1, top_right = 0;
        int top_up = 0, top_down = 0;
        for(int i = 0; i < K; i++){
            if(black.find(pair<int,int>{cur_x, cur_y}) == black.end()){
                black.insert(pair<int,int>{cur_x, cur_y});
                cur_direction = (cur_direction+1)%4;
            }else{
                black.erase(pair<int,int>{cur_x, cur_y});
                cur_direction = (cur_direction - 1 + 4) % 4;
            }
            cur_x += hori[cur_direction];
            cur_y += vert[cur_direction];
            top_left = min(top_left, cur_x);
            top_right = max(top_right, cur_x);
            top_down = min(top_down, cur_y);
            top_up = max(top_up, cur_y);
        }
        int rows = top_up - top_down + 1;
        int cols = top_right - top_left + 1;
        string t;
        for(int i = 0; i < cols; i++) t.push_back('_');
        vector<string> ans(rows, t);
        for(auto it=black.begin(); it != black.end(); it++){
            int x = it->first, y = it->second;
            ans[top_up-y][x-top_left] = 'X';
        }
        ans[top_up-cur_y][cur_x-top_left] = direction[cur_direction];
        return ans;
    }
};
```

面试题16.24 数对和（中等）

1. 题目描述

   设计一个算法，找出数组中两数之和为指定值的所有整数对。一个数只能属于一个数对。

```
示例 1:
输入: nums = [5,6,5], target = 11
输出: [[5,6]]

示例 2:
输入: nums = [5,6,5,6], target = 11
输出: [[5,6],[5,6]]
```

提示：nums.length <= 100000

2. 简单实现

```
class Solution {
public:
    vector<vector<int>> pairSums(vector<int>& nums, int target) {
        unordered_map<int, int> cnt;
        vector<vector<int>> ans;
        for(int i = 0; i < nums.size(); i++)
            cnt[nums[i]]++;
        for(auto it = cnt.begin(); it != cnt.end(); it++){
            int aim = target - it->first;
            if(cnt.count(aim) > 0){
                it->second--;
                cnt[aim]--;
                while(it->second >= 0 && cnt[aim] >= 0){//这样的判断防止数对出自同一个数
字的情况
                    ans.push_back({it->first, aim});
                    it->second--;
                    cnt[aim]--;
                }
            }
        }
        return ans;
    }
};
```

3. 最简写法

```
class Solution {
public:
    vector<vector<int>> pairSums(vector<int>& nums, int target) {
        unordered_map<int,int>m;
        vector<vector<int>>res;
        for(auto i : nums){
            if(m[target-i]>0){
                m[target-i]--;
               res.push_back({target-i,i});
            }
            else m[i]++;
        }
        return res;
    }
```

```
};
```

面试题16.25 LRU缓存（中等）

1. 题目描述

   设计和构建一个“最近最少使用”缓存，该缓存会删除最近最少使用的项目。缓存应该从键映射到值(允许你插入和检索特定键对应的值)，并在初始化时指定最大容量。当缓存被填满时，它应该删除最近最少使用的项目。

   它应该支持以下操作： 获取数据 get 和 写入数据 put 。

   获取数据 get(key) - 如果密钥 (key) 存在于缓存中，则获取密钥的值（总是正数），否则返回 -1。 写入数据 put(key, value) - 如果密钥不存在，则写入其数据值。当缓存容量达到上限时，它应该在写入新数据之前删除最近最少使用的数据值，从而为新的数据值留出空间。

```
示例:
LRUCache cache = new LRUCache( 2 /* 缓存容量 */ );
cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // 返回  1
cache.put(3, 3);    // 该操作会使得密钥 2 作废
cache.get(2);       // 返回 -1 (未找到)
cache.put(4, 4);    // 该操作会使得密钥 1 作废
cache.get(1);       // 返回 -1 (未找到)
cache.get(3);       // 返回  3
cache.get(4);       // 返回  4
```

2. 简单实现

```
class LRUCache {
private:
    int cap;
    // 双链表：装着 (key, value) 元组
    list<pair<int, int>> cache;
    // 哈希表：key 映射到 (key, value) 在 cache 中的位置
    unordered_map<int, list<pair<int, int>>::iterator> map;
public:
    LRUCache(int capacity) {
        this->cap = capacity;
    }

    int get(int key) {
        auto it = map.find(key);
        // 访问的 key 不存在
        if (it == map.end()) return -1;
        // key 存在，把 (k, v) 换到队头
        pair<int, int> kv = *map[key];
        cache.erase(map[key]);
        cache.push_front(kv);
        // 更新 (key, value) 在 cache 中的位置
        map[key] = cache.begin();
```

```
        return kv.second; // value
    }

    void put(int key, int value) {

        /* 要先判断 key 是否已经存在 */
        auto it = map.find(key);
        if (it == map.end()) {
            /* key 不存在，判断 cache 是否已满 */
            if (cache.size() == cap) {
                // cache 已满，删除尾部的键值对腾位置
                // cache 和 map 中的数据都要删除
                auto lastPair = cache.back();
                int lastKey = lastPair.first;
                map.erase(lastKey);
                cache.pop_back();
            }
            // cache 没满，可以直接添加
            cache.push_front(make_pair(key, value));
            map[key] = cache.begin();
        } else {
            /* key 存在，更改 value 并换到队头 */
            cache.erase(map[key]);
            cache.push_front(make_pair(key, value));
            map[key] = cache.begin();
        }
    }
};
```

面试题16.26 计算器（中等）

1. 题目描述

给定一个包含正整数、加(+)、减(-)、乘(*)、除(/)的算数表达式(括号除外)，计算其结果。

表达式仅包含非负整数，+， - ，*，/ 四种运算符和空格 。 整数除法仅保留整数部分。

```
示例 1:
输入: "3+2*2"
输出: 7

示例 2:
输入: " 3/2 "
输出: 1

示例 3:
输入: " 3+5 / 2 "
输出: 5
```

说明:

- 你可以假设所给定的表达式都是有效的。
- 请不要使用内置的库函数 eval。

2. 简单实现

恶心的细节

```
class Solution {
public:
    int calculate(string s) {
        deque<string> cache;
        string cur = "";
        for(int i = 0; i < s.size(); i++){
            if(s[i] == ' ') continue;
            else if(s[i] >= '0' && s[i] <= '9') cur += s[i];
            else{
                if(!cache.empty() && (cache.back() == "*" || cache.back() == "/")){
                    //把前面的*和/处理掉
                    string op = cache.back();
                    cache.pop_back();
                    int a = stoi(cache.back());
                    cache.pop_back();
                    int b = stoi(cur);
                    if(op == "*")
                        cache.push_back(to_string(a*b));
                    else
                        cache.push_back(to_string(a/b));
                }
                else
                    cache.push_back(cur);
                cache.push_back(s.substr(i,1));
                cur = "";
            }
        }
        if(!cache.empty() && (cache.back() == "*" || cache.back() == "/")){
            string op = cache.back();
            cache.pop_back();
            int a = stoi(cache.back());
            cache.pop_back();
            int b = stoi(cur);
            if(op == "*")
                cache.push_back(to_string(a*b));
            else
                cache.push_back(to_string(a/b));
        }
        else
            cache.push_back(cur);
        int ans = stoi(cache.front());
        cache.pop_front();
        while(!cache.empty()){
            if(cache.front() == "-"){
                cache.pop_front();
                int a = stoi(cache.front());
                cache.pop_front();
                ans = ans - a;
            }
            else if(cache.front() == "+"){
```

```
                cache.pop_front();
                int a = stoi(cache.front());
                cache.pop_front();
                ans = ans + a;
            }
        }
        return ans;
    }
};
```