Table of Contents

leetcode讲解

leetcode是一个 Online Judge 网站，在线练习编程，尤其是练习数据结构和算法相关的题。

刷OJ对于锻炼自己的计算机思维，算法能力，以及coding能力都是非常有帮助的。

这份gitbook电子书，将提供leetcode上 所有免费题目的答案，并提供 详细的解答思路，所使用的编程语言有：**Python**、**C++**、**Java**。

几点小建议：

1. 按照AC（accepted）率从高到低刷题，这样就会从易到难，提供一个进步的缓冲空间，不至于一上来就被打击到。
2. 按照算法和数据结构体系，逐个模块的掌握，leetcode会提示哪些题目是类似的，另外网上也有不少按算法和数据结构总结的解答电子书可以参照。
3. 做不出来的时候可以看leetcode网站上的 Discuss 和 Solutions ，有不少大神的解题方法和精简代码。当然也可以在互联网上搜索，有很多讲解leetcode的博客。

本教程打算分两步走：

1. 第一步把每道题的讲解和代码贴出来
2. 第二步，做个总结，按照算法体系把有价值的题归纳总结、举一反三，得出一些有价值的思考和结论。

本gitbook的github仓库：leetcode讲解

本gitbook的地址：leetcode讲解

关于作者：

- liuqinh2s的个人主页

参考(实际上参考了很多网络上的资料，如果您觉得其中有您的工作成果，请私信我并指明出处)：

- 书影博客
- 细语呢喃- 技术改变生活
- leetcode自带的答案讨论区

中文版

第一遍，按AC率刷题

94. Binary Tree Inorder Traversal

Given a binary tree, return the inorder traversal of its nodes' values.

For example:
Given binary tree [1,null,2,3],
1
\
2
/
3
return [1,3,2].

Note: Recursive solution is trivial, could you do it iteratively?

题目地址

嗯，经典的题目，中序遍历二叉树。题目说递归做起来太简单，请用迭代。我觉得还是先用递归做一下吧，也许我递归都不会呢？谁知道呢，是吧。然后发现，嗯，果然很简单，我好像已经不是从前那个菜逼了。

代码如下：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
private:
        vector<int> vec;
public:
    vector<int> inorderTraversal(TreeNode* root) {//中序遍历
        if(root){
            inorderTraversal(root->left);
            vec.push_back(root->val);
            inorderTraversal(root->right);
        }
        return vec;
    }
    vector<int> preorderTraversal(TreeNode* root){//先序遍历
      if(root){
        vec.push_back(root->val);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
      }
      return vec;
    }
    vector<int> postorderTraversal(TreeNode* root){//后序遍历
      if(root){
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        vec.push_back(root->val);
      }
      return vec;
    }
};
```

是不是简单的不要不要的，递归就是这么美。

好了，现在想想怎么用迭代中序遍历呢。别想了，用栈吧，不用栈你做给我看试试。

> 层次遍历树相当于图的广度优先遍历(BFS,breadth first search)，先序、中序、后序遍历二叉树都相当于图的深度优先遍历(DFS,depth first search)。要用迭代可以啊，广度优先请用队列，深度优先请用栈。

借助栈深度优先遍历时，首先我们压栈第一个结点（入口结点），然后又pop栈顶结点，我们把pop出来的结点的所有子节点都压进栈中，然后又是pop栈顶，继续进行如上操作。也许你发现了一个问题：没有说哪个孩子先压进栈，哪个后压栈，但有一点很明显，先压栈的会靠后访问，后压栈的会先被访问。

上代码：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
private:
    vector<int> vec;
    stack<TreeNode*> s;
    stack<TreeNode*> s1;
public:
    vector<int> inorderTraversal(TreeNode* root) {//中序遍历
        TreeNode* current=root;
        while(!s.empty() || current!=NULL){
            while(current){//如果结点存在，则入栈，然后判断左孩子
                s.push(current);//越靠近root的越先压栈哦
                current = current->left;
            }
            //往左走到了尽头，然后就是pop栈顶了
            current = s.top();
            s.pop();
            vec.push_back(current->val);
            //接着是判断右孩子
            current = current->right;
        }
        return vec;
    }
    vector<int> preorderTraversal(TreeNode* root){//先序遍历
```

```
        TreeNode* current=root;
        while(!s.empty() || current!=NULL){
          while(current){
            vec.push_back(current->val);//遇到一个就先访问一个呗
            s.push(current->right);//右孩子压栈，越靠近root的越先压栈哦
            current = current->left;
          }
          current=s.top();
          s.pop();
        }
        return vec;
    }
    //后序遍历，后序遍历就有点复杂了，需要两个栈，蛋疼。。。，
    //什么你问我为啥多了一个栈，因为根节点要存起来啊，它非得要最后访问，就把它一脚踢进栈里。
    vector<int> postorderTraversal(TreeNode* root){
        TreeNode* current=root;
        s.push(current);
        while(!s.empty()){
            current = s.top();
            s.pop();
            s1.push(current);//s1就是用来放后序遍历中的根结点的，root被压到最底下了，哈哈。s1放的是逆序打印顺序哦
            if(current->left)
              s.push(current->left);//左孩子先压栈，先压栈的后访问呐，左孩子不是应该先访问吗，拜托，后面要压入s1，顺序又反了。。
            if(current->right)
              s.push(current->right);//右孩子后压栈
        }
        while(!s1.empty()){//全部释放出来
          vec.push_back(s1.top()->val);
          s1.pop();
        }
        return vec;
    }
};
```

嗯，总的来说用迭代遍历确实烧脑，没什么性能上的特殊需求还是用递归写法吧，对程序员友好哦。

169. Majority Element

Given an array of size n, find the majority element. The majority element is the element that appears more than ⌊ n/2 ⌋ times.

You may assume that the array is non-empty and the majority element always exist in the array.

题目地址

唉，看到这题就想起了，当年，青葱岁月，菜鸡的我做阿里的笔试题，对的，就是这题。当时题目改成了小明收红包，找出现次数超过一般的那个红包，要求线性时间复杂度，也就是说不能用排序（排序算法最优情况是：O(nlogn)）。

出现次数超过一半的那个数，我们怎么在O(n)时间复杂度内把它揪出来，我教你：大浪淘沙法，是金子总会发光法，我要打十个法（意思就是，这一个数就能干翻全场剩下的所有数，是不是？因为它超过一半啊）。我们来想象一下：我们遍历这个数组的时候，定义一个count用来计数，这个超过一半的数，它遇到自己就给count加1，遇到不是自己的数，就给count减1，最后会怎样呢，count肯定大于0呐，因为这个数的个数超过一半。好，进一步的，我们先随便找个数当这个老大（个数超过一半），如果它的个数不超过一半，就会在相消中时count为0，那么就把它换掉，最后剩下的那个就是，个数超过一半的那个数了。

代码如下：

```
class Solution(object):
    def majorityElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        count = 1
        n = nums[0]
        for x in range(1, len(nums)):
            if count == 0:
                n = nums[x]
                count += 1
            else:
                if nums[x] == n:
                    count += 1
                else:
                    count -= 1
        return n
```

超过一半呐，多么重的一条信息，稍微想想就知道，用相消法都能消出那个数，跟那个用异或消出只出现一次的数（其它的数都是成对的，就是都恰好有两个）是不是异曲同工呐，嗯还是扯得有点远。

另外这个题在leetcode上的难度是easy哦，好伤心啊，当初的我连这题都没想出解法，眞是够年轻啊。

217. Contains Duplicate

Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.

so easy的一道题。我直接上代码吧：

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        unordered_map<int, int> hash;
        for(auto i:nums){
            hash[i]++;
        }
        for(auto i:hash){
            if(i.second>=2)
                return true;
        }
        return false;
    }
};
```

343. Integer Break

题目地址

好吧我还是把题目贴出来，反正不长：

Given a positive integer n, break it into the sum of at least two positive integers and maximize the product of those integers. Return the maximum product you can get.

For example, given n = 2, return 1 (2 = 1 + 1); given n = 10, return 36 (10 = 3 + 3 + 4).

Note: You may assume that n is not less than 2 and not larger than 58.

又是一个要 靠罗列来发现规律 的例子：

规模**n**	最大值
2	`1*1`
3	`2*1`
4	`2*2`
5	`3*2`
6	`3*3`
7	`3*2*2`
8	`3*3*2`
9	`3*3*3`
10	`3*3*2*2`
11	`3*3*3*2`
12	`3*3*3*3`

> 好，我们定义一个最大值数组 `dp[n]` (dynamic programming)， `dp[5]` 即规模为5时候的最大值。

从上面的表中，你发现了什么规律？

1. 尽可能避免拆成1
2. 规模为4的时候，恰好 `dp[4]=4` ，之后 `dp[5]` 就开始>5了，而之前的 `dp[2]、dp[3]` 都是小于自身规模（n=2,n=3）的。
3. 如果我们把2到6看成基本集合，那么后面从7开始是可以归因到2到6的，首先归因到6肯定不对，我们不欢迎1的拆分除非迫不得已（2和3只能那样拆才能拆成2个数的乘积啊），比如7可以写成 `2*dp[5]=2*3*2` ，也可以写成 `3*dp[4]=3*2*2` ，但是没必要写成 `4*dp[3]` 了，因为 `dp[3]` 已经小于3了，也就是说，我们每次归因都到 `dp[4]` 打止。但这还是不对，我们看9，9可以归因为 `2*dp[7],3*dp[6]，4*dp[5],5*dp[4]` （这可不行啊，5已经小于 `dp[5]` 了，所以完全不能用，之后的什么6啊，7啊更加不能用，所以其实也是

到4打止了，不再往前搜），所以实际上我们最多往前归因4位。其实这还是不对，聪明的你应该已经发现，往前归因4位，其实跟归因2位两次是一样的。所以我们只需从 `2*dp[n-2],3*dp[n-3]` 中选出大者就行。

4. 我们得到动态规划公式（动态规划就是归因（多元因）啊，而递归则是单一因）：`dp[n]=max(2*dp[n-2],3*dp[n-3])` ，且n>=7（因为我们不越过 `dp[4]` 啊）。

代码如下：

```
class Solution(object):
    def integerBreak(self, n):
        """
        :type n: int
        :rtype: int
        """
        dp = [0]*(n+7)
        dp[2]=1
        dp[3]=2
        dp[4]=4
        dp[5]=6
        dp[6]=9
        if(n>=7):
            for x in range(7, n + 1):
                dp[x] = max(3 * dp[x - 3], 2 * dp[x - 2])
        return dp[n]
```

我自己感到这代码有着深深的蛋疼（手动赋了一大堆值），因为我看到别人是这样写的：

```
class Solution(object):
    def integerBreak(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n<4:
            return n-1
        dp = [0]*(n+1)
        dp[2], dp[3] = 2, 3
        for x in range(4, n + 1):
            dp[x] = max(3 * dp[x - 3], 2 * dp[x - 2])
        return dp[n]
```

好吧，他的代码其实也差不多。

下面真正吊炸天的来了。

聪明的你肯定已经发现，这完全就是个2跟3的游戏，是的，事实就是如此。

- n%3==0时，就全部用3
- n%3==2时，勉为其难用个2吧
- n%3==1时，就再勉为其难用两个2吧

当n%3==3，哈哈别傻了，

当然你还得看到，最开始的2个里面是有1的，可不要忘了。

所以，得到牛逼的专门解题代码：

```
class Solution(object):
    def integerBreak(self, n):
        """
        :type n: int
        :rtype: int
        """
        if n < 4:
            return n - 1
        if n % 3 == 0:
            return 3 ** (n / 3)
        if n % 3 == 1:
            return 3 ** ((n - 3) / 3) * 4
        if n % 3 == 2:
            return 3 ** ((n - 2) / 3) * 2
```

这种题也就娱乐娱乐，程序员还是要以工程代码为主业啊。

350. Intersection of Two Arrays II

题目地址

Given two arrays, write a function to compute their intersection.

Example: Given nums1 = [1, 2, 2, 1], nums2 = [2, 2], return [2, 2].

Note: Each element in the result should appear as many times as it shows in both arrays. The result can be in any order. Follow up: What if the given array is already sorted? How would you optimize your algorithm? What if nums1's size is small compared to nums2's size? Which algorithm is better? What if elements of nums2 are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

翻译：

给两个数组，写一个函数来计算它们的交集。

例子：

给定 nums1 = [1, 2, 2, 1], nums2 = [2, 2], 返回 [2, 2].

注意：

结果中的每个元素出现的次数都应该与它们出现在这两个数组中的次数一样多。

结果中的元素顺序随意。

接下来：

如果给的数组已经排好了序怎么办？你怎么优化你的算法？ 如果 nums1 的长度比 nums2 长怎么办？哪种算法更好？ 如果 nums2 的元素存储在磁盘上，并且内存有限以至于你无法一次性把所有元素都加载到内存，怎么办？

先上代码：

```
class Solution(object):
    def intersect(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: List[int]
        """
        ans=[]
        c = collections.Counter(nums1)
        for x in nums2:
            if c[x]>0:
                ans += x,
                c[x] -= 1
        return ans
```

很简单吧，遍历两个数组即可（先把其中一个用哈希表，即 hashtable 存起来，然后遍历另一个，一边遍历一边匹配，匹配到了就加进结果集，然后哈希表相应项减一）。没有任何难度。

再来看题目给出的附加问题：

跟排没排好序没有半毛钱关系，因为你至少要遍历一遍吧，O(n)的时间总是要的。

nums1 和 nums2 哪个长，也没有半毛钱关系（你把长的用哈希表存起来，再去匹配短的，和把短的用哈希表存起来，再去匹配唱的，不是一样的嘛）。

如果 nums2 无法一次性拿出来，那就分几次拿呗，同样，没有半毛钱关系。

这附加问题有点古怪啊，我没看出来有什么意义啊，求看懂了的同学教教我。

392. Is Subsequence

Given a string s and a string t, check if s is subsequence of t.

You may assume that there is only lower case English letters in both s and t. t is potentially a very long (length ~= 500,000) string, and s is a short string (<=100).

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ace" is a subsequence of "abcde" while "aec" is not).

Example 1: s = "abc", t = "ahbgdc"

Return true.

Example 2: s = "axc", t = "ahbgdc"

Return false.

Follow up: If there are lots of incoming S, say S1, S2, ... , Sk where k >= 1B, and you want to check one by one to see if T has its subsequence. In this scenario, how would you change your code?

题目地址

这题我觉得很简单啊，不知道为何评级是medium。

就是遍历两个字符串就行了。直接上代码，看不懂可以在评论里面写上，我会针对性讲解。

```
class Solution {
public:
    bool isSubsequence(string s, string t) {
        int i=0,j=0;
        while(i<s.size() && j<t.size()){
            if(s[i]==t[j]){
                i++;
                j++;
            }else{
                j++;
            }
        }
        if(i!=s.size())
            return false;
        return true;
    }
};
```

409. Longest Palindrome

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

This is case sensitive, for example "Aa" is not considered a palindrome here.

Note: Assume the length of given string will not exceed 1,010.

Example:

Input: "abccccdd"

Output: 7

Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.

题目地址

这题有点小儿科了，很简单，你看有多少个对子（打过牌吗，就是看有多少对子），然后看有没有一个单（注意：只需要一个单，放中间就行了）。

```
class Solution {
public:
    int longestPalindrome(string s) {
        unordered_map<char, int> hash;
        int count=0;
        int flag=0;//看有没有单
        for(auto i:s){
            hash[i]++;
        }
        for(auto i:hash){
            if(i.second%2 != 0)
                flag=1;
            count += (i.second/2)*2;//把对子全都加上
        }
        return count+flag;
    }
};
```

13. Roman to Integer

题目地址

Given a roman numeral, convert it to an integer.

Input is guaranteed to be within the range from 1 to 3999.

翻译

给定一个罗马数，把它转化成整数。

输入保证在1到3999的范围内。

分析

罗马数字有：{'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}，（这里是用键值对表示）

罗马数字的组合规则：左边是减，右边是加。如：IV=4，VI=6。

那么我们很容易想到，先统一加起来，然后看到相邻的逆序（小的在左，大的在右，为逆序），就减2倍这个数（因为原来多加了一次，所以减2倍）。

代码：

```
class Solution(object):
    def romanToInt(self, s):
        """
        :type s: str
        :rtype: int
        """
        map={'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500,
'M':1000}
        sum=0
        for i in range(len(s)):
            sum += map[s[i]]
            if i>0 and map[s[i-1]]<map[s[i]]:
                sum = sum - 2*map[s[i-1]]
        return sum
```

268. Missing Number

题目地址

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the one that is missing from the array.

For example, Given nums = [0, 1, 3] return 2.

Note: Your algorithm should run in linear runtime complexity. Could you implement it using only constant extra space complexity?

翻译

给定一个数组包含 0, 1, 2, ..., n， n个不相同的数字，从数组中找出这个缺失的数。

例如，

给定 nums = [0, 1, 3] 返回 2。

注意：

你的算法应该运行在线性的时间复杂度内。你能只使用常数空间复杂度来实现它吗？

分析

这题极其简单啊，用加减法就能找出那个数啊。

代码：

```
class Solution(object):
    def missingNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        count1=0
        count2=0
        for i in range(len(nums)+1):
            count1 += i
        for i in nums:
            count2 += i
        return count1-count2
```

401. Binary Watch

题目地址

A binary watch has 4 LEDs on the top which represent the hours (0-11), and the 6 LEDs on the bottom represent the minutes (0-59).

Each LED represents a zero or one, with the least significant bit on the right.

For example, the above binary watch reads "3:25".

Given a non-negative integer n which represents the number of LEDs that are currently on, return all possible times the watch could represent.

Example:

Input: n = 1

Return: ["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16", "0:32"]

Note:

The order of output does not matter.

The hour must not contain a leading zero, for example "01:00" is not valid, it should be "1:00".

The minute must be consist of two digits and may contain a leading zero, for example "10:2" is not valid, it should be "10:02".

翻译

一个二进制手表有4个LED灯在表盘的上部，代表（0-11）个小时，且有6个LED灯在表盘的底部，代表（0-59）分钟。

每个LED灯代表0或者1，权重最小的比特位在右边。

例如，上面的二进制表读作："3：25"

给定一个非负整数 n ，n 代表亮着的LED灯的个数，返回手表所有可能表示的时间。

例子：

输入： n = 1

返回：["1:00", "2:00", "4:00", "8:00", "0:01", "0:02", "0:04", "0:08", "0:16", "0:32"]

注意：

输出顺序任意。

小时不能以0开头，例如："01:00"是无效的，而应该是"1:00"。

分钟必须包含两个数字且可能包含开头是0的情况，例如："10:2"是无效的，而应该是"10:02"。

分析

位运算

总共有10个位，遍历0~1024，当1的个数等于num时，利用位运算抽取小时，分钟的读数，然后符合条件（小时在0-11之间，分钟在0-59之间）就加进结果集里。

代码：

```
class Solution(object):
    def readBinaryWatch(self, num):
        """
        :type num: int
        :rtype: List[str]
        """
        ans = []
        for i in range(0,1024):
            if bin(i).count('1') == num:
                h, m = i>>6, i&0x3F
                if h<12 and m<60:
                    ans += '%d:%02d' %(h, m),
        return ans
```

枚举

枚举小时和分钟（共12*60种），读取1的个数，如果和num相等就加进结果集。

代码：

```
class Solution(object):
    def readBinaryWatch(self, num):
        """
        :type num: int
        :rtype: List[str]
        """
        ans = []
        for h in range(12):
            for m in range(60):
                if bin(h).count('1')+bin(m).count('1')==num:
                    ans.append("%d:%02d" %(h,m))
        return ans
```

206. Reverse Linked List

题目地址

Reverse a singly linked list.

翻译：

翻转一个单链表。

好经典的题目，翻转单链表是数据结构中单链表操作的基本操作。操作方法有：

1. 使用头插法（新建一个新的头结点，然后不断的从旧结点中取出结点，查到新的头结点后面）。
2. 双指针法（一个指针用来遍历链表，一个指针用来当新的头指针）。

另外可以用迭代和递归两种做法。

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.next = None

# 迭代+头插法
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        list = ListNode(0)
        while head:
            next = head.next
            head.next = list.next
            list.next = head
            head = next
        return list.next
# 递归+头插法
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
```

```
        """
        newHead = ListNode(0)
        return self.doReverse(head, newHead)
    def doReverse(self, head, newHead):
        if head:
            next = head.next
            head.next = newHead.next
            newHead.next = head
            return self.doReverse(next, newHead)
        else:
            return newHead.next
# 迭代+双指针法
class Solution(object):
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        newHead = None
        while head:
            next = head.next
            head.next = newHead
            newHead = head
            head = next
        return newHead
# 递归+双指针法
class Solution(object):
    def doReverse(self, nowNode, newHead):
        if nowNode:
            next = nowNode.next
            nowNode.next = newHead
            return self.doReverse(next, nowNode)
        else:
            return newHead
    def reverseList(self, head):
        """
        :type head: ListNode
        :rtype: ListNode
        """
        return self.doReverse(head, None)
```

378. Kth Smallest Element in a Sorted Matrix

Given a n x n matrix where each of the rows and columns are sorted in ascending order, find the kth smallest element in the matrix.

Note that it is the kth smallest element in the sorted order, not the kth distinct element.

Example:

```
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,

return 13.
```

Note:

You may assume k is always valid, 1 ≤ k ≤ n^2.

翻译

给定一个 n x n 的矩阵，它的行和列都是按升序排列的，找到矩阵中第k小的元素。

注意：是按排序第k小的元素，不是第k个不同的元素。

例子：

```
matrix = [
   [ 1,  5,  9],
   [10, 11, 13],
   [12, 13, 15]
],
k = 8,

return 13.
```

注意：

你应该假设k总是有效的，1 ≤ k ≤ n^2。

分析

算法题嘛，让我们先从复杂度高的算法开始逐步进化到复杂度低的吧。

利用数据结构：堆

首先我们很容易想到，可以利用小根堆来存放所有元素，然后弹出到第k个，那就是第k小的啦。遍历一遍矩阵，复杂度：O(n^2)，弹出堆顶以及维护堆，k*log(n^2)。复杂度是有点高。

```
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        q = []
        for i in range(0,len(matrix)):
            for j in range(0,len(matrix[0])):
                heapq.heappush(q,matrix[i][j])
        for x in range(1,k):
            heapq.heappop(q)
        return heapq.heappop(q)
# 下面的代码用了一个库函数，哈哈
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        q = []
        for i in range(0,len(matrix)):
            for j in range(0,len(matrix[0])):
                heapq.heappush(q,matrix[i][j])

        return heapq.nsmallest(k, q)[k-1]
```

这里我们漏了一个条件就是：行和列都是升序排列的，所以没有必要遍历一遍整个矩阵。

算法：

首先将矩阵的左上角（下标0,0）元素加入堆

然后执行k次循环：

弹出堆顶元素top，记其下标为i, j

将其下方元素matrix[i + 1][j]，与右方元素matrix[i][j + 1]加入堆（若它们没有加入过堆）

我们还是维护一个小根堆，不过这次我们一边加元素进堆（每次都找堆顶元素在矩阵中的下面和右边的元素入堆），一边弹出堆顶（弹k次即可哦）。复杂度：入堆是2log(n^2)，弹出堆顶并维护堆是log(n^2)，所以时间复杂度是O(log(n^2))。

```
class Solution(object):
    def kthSmallest(self, matrix, k):
        """
        :type matrix: List[List[int]]
        :type k: int
        :rtype: int
        """
        q = [(matrix[0][0], 0, 0)]
        n = len(matrix)
        visited = [[False]*n for i in range(n)]
        for x in range(k):
            ans, i, j = heapq.heappop(q)
            if i<n-1 and visited[i+1][j]==False:
                visited[i+1][j]=True
                heapq.heappush(q, [matrix[i+1][j], i+1, j])
            if j<n-1 and visited[i][j+1]==False:
                visited[i][j+1]=True
                heapq.heappush(q, [matrix[i][j+1], i, j+1])

        return ans
```

二分查找

排序的东西，

486. Predict the Winner

Given an array of scores that are non-negative integers. Player 1 picks one of the numbers from either end of the array followed by the player 2 and then player 1 and so on. Each time a player picks a number, that number will not be available for the next player. This continues until all the scores have been chosen. The player with the maximum score wins.

Given an array of scores, predict whether player 1 is the winner. You can assume each player plays to maximize his score.

Example 1:

Input: [1, 5, 2]

Output: False

Explanation: Initially, player 1 can choose between 1 and 2.

If he chooses 2 (or 1), then player 2 can choose from 1 (or 2) and 5. If player 2 chooses 5, then player 1 will be left with 1 (or 2).

So, final score of player 1 is 1 + 2 = 3, and player 2 is 5.

Hence, player 1 will never be the winner and you need to return False.

Example 2:

Input: [1, 5, 233, 7]

Output: True

Explanation: Player 1 first chooses 1. Then player 2 have to choose between 5 and 7. No matter which number player 2 choose, player 1 can choose 233.

Finally, player 1 has more score (234) than player 2 (12), so you need to return True representing player1 can win.

Note:

1 <= length of the array <= 20.

Any scores in the given array are non-negative integers and will not exceed 10,000,000.

If the scores of both players are equal, then player 1 is still the winner.

翻译

给定一个非负的整形的数组（数组中装的数称为分数）。玩家1从数组的任意一端挑走一个数，接着是玩家2，接着又是玩家1。每次一个玩家挑走一个数，这个数就不能被下一个玩家挑了。一直这样直到所有分数都被选走。拥有分数总和多的玩家获胜。

给定一个包含分数的数组，预言玩家1是否会赢。你应该假设每个玩家都想要把自己的分数最大化。

例子1：

输入：[1, 5, 2]

输出: False

解释: 初始条件下, 玩家1可以在 1 和 2 之间选择。

如果他选择 2 (或 1), 然后玩家2能从 1（或者 2）和 5 之间选择. 如果玩家2选择5, 然后玩家1只剩下 1 或者 2 可以选择。

所以最后玩家1的分数是 3 ，玩家2的分数是 5。

因此，玩家1不可能赢，你应该返回 False。

例子2：

输入: [1, 5, 233, 7]

输出: True

解释: 玩家1首先选择1. 玩家2必须在 5 和 7 之间选择. 不管玩家2选择哪个数, 玩家1总能选到233。

最后玩家1（234）比玩家2（12）拥有更多的分数, 所以你应该返回True，代表玩家1会赢。

注意：

1 <= 数组的长度 <= 20。

给定的数组中的每个分数都是非负的并且不会超过10,000,000。

如果每个玩家的分数都是相等的，那么玩家1就是赢家。

分析

这是一个典型的Minimax算法的案例，两个玩家都要使自己的利益最大化，使对手的利益最小化。Minimax是一个零和博弈中的算法。而Alpha-beta剪枝则是一个基于Minimax的剪枝算法（当算法评估出某策略的后续走法比之前策略的还差时，就会停止计算该策略的后续发展）。

Minimax算法伪代码：

```
function minimax(node, depth)
    if node is a terminal node or depth = 0
        return the heuristic value of node
    if the adversary is to play at node
        let α := +∞
        foreach child of node
            α := min(α, minimax(child, depth-1))
    else {we are to play at node}
        let α := -∞
        foreach child of node
            α := max(α, minimax(child, depth-1))
    return α
```

当你看上面的伪代码，有没有注意到，在对手回合时，我们这个算法默认对手的收益是无穷大，而在自己回合时，我们这个算法默认我们的收益是无穷小。这是一种 选择总比不选择好的算法，也就是说我们默认必须要从子节点中选出一个最佳的，而不能什么都不选（默认什么都不选时情况最差）。换过来思考，如果我们默认：对手的收益是无穷小，自己收益无穷大。那我们什么都不用做了，因为任何选择都好不过使对手收益无穷小，自己收益无穷大，也就是说无论什么情况不选择任何一个子节点就行了（不选择比选择要好）。

Alpha-beta剪枝算法伪代码：

```
function alphabeta(node, depth, α, β, maximizingPlayer) // node
= 节点，depth = 深度，maximizingPlayer = 大分玩家
      if depth = 0 or node是终端节点
          return 节点的启发值
      if maximizingPlayer
          v := -∞
          for 每个子节点
              v := max(v, alphabeta(child, depth - 1, α, β,
FALSE)) // child = 子节点
              α := max(α, v)
              if β ≤ α
                  break // β裁剪
          return v
      else
          v := ∞
          for each 每个子节点
              v := min(v, alphabeta(child, depth - 1, α, β,
TRUE))
              β := min(β, v)
              if β ≤ α
                  break // α裁剪
          return v
//初始调用
alphabeta(origin, depth, -∞, +∞, TRUE) // origin = 初始节点
```

先讲Minimax算法：

首先，Max就是要最大化利益，Max函数会选择一个利益最大的子分支，Min就是要最小化利益，Min函数会选择一个利益最小的子分支，Max和Min交替执行（因为两个玩家一人走一步嘛），当我玩的时候我就调用Max（让自己赚的盆满钵满），到了对手的回合，他就调用Min（让我得不到好处，这就叫零和博弈，你死我活，没有合作）。那么Max和Min分别都是怎么计算出利益大小的呢？递归！不知道利益大小你就递归地往下问呗，问道最后再递归地回答上来，利益就计算出来了。

再讲讲Alpha-beta剪枝算法：

这是youtube上的一个视频，讲的非常细，听完之后肯定懂。

Step by Step: Alpha Beta Pruning

理解这个算法的关键点就在于：min 和 max 各自自身的性质，

- 如果当前结点是min，min只要选出一个β, β ≤ α, 由于 min 会选最小的，所以min<=β，或者说min不会比β大，无疑。而α代表父节点（max）已经找到的最大价值子节点，所以父

节点会果断抛弃当前分支（因为当前分支<= β ≤ α），这就叫α剪枝。

- 类似的道理，如果当前结点是max，max只要选出一个α，β ≤ α，由于max会选最大的，所以 max >= α，而β 代表父节点（min）已经找到的最小价值子节点，所以父节点会果断抛弃当前分支（因为当前分支>= α >= β）。

很显然，这个Alpha-beta剪枝算法带来一个好处，就是发现当前结点的值的范围不符合父节点的要求时，立马break回到父节点，节省了很多不必要的分支计算。

代码

我们用一个solve(nums)函数来计算当前玩家从nums中可以获得的最大收益，当收益>=0时，玩家1获胜。
solve(nums) = max(nums[0] - solve(nums[1:]), nums[-1] - solve(nums[:-1]))
为什么是nums[0]-solve(nums[1:])而不是+呢？因为玩家拿完一个数之后就轮到别人了，
此时solve函数是别人的收益，
我们用累计自己收益，并扣除对手收益的方式来表示自己的收益最后是否比对手多（如果结果大于0就表示比对手多）。

另外我们发现，在用Minimax算法的时候可能会有重复的计算，比如输入如果是[1, 5, 233, 7]的话，solve([5, 233])就会被重复计算。所以我们可以用记忆化。

下面这段代码用的是 Minimax+记忆化 算法

```
class Solution(object):
    def PredictTheWinner(self, nums):
        """
        :type nums: List[int]
        :rtype: bool
        """
        cache = dict()
        def solve(nums):
            if len(nums)<=1:
                return sum(nums)
            tnums = tuple(nums)
            if tnums in cache:
                return cache[tnums]
            cache[tnums] = max(nums[0]-solve(nums[1:]),
nums[-1]-solve(nums[:-1]))
            return cache[tnums]
        return solve(nums)>=0
```

用Python就是很爽啊，有现成的dict（相当于其他语言中的hashmap），可以函数中定义函数（这样就少传一些参数啦，就像这里cache可以直接使用，而不必作为一个参数传递给solve函数，顺便说下，这种环境变量（cache）+函数的捆绑叫：闭包）。

如果不用记忆化，Python写出来的代码会超时。

下面这段Java代码其实还是用的同样的算法（除去了记忆化）：

```
import java.util.HashMap;

public class Solution {
    public boolean PredictTheWinner(int[] nums) {
        return helper(nums, 0, nums.length-1)>=0;
    }
    private int helper(int[] nums, int s, int e){
        return s==e ? nums[e] : Math.max(nums[e] - helper(nums,
s, e-1), nums[s] - helper(nums, s+1, e));
    }
}
```

Java代码一出来，C++还不是分分钟的事？

```
class Solution {
public:
    bool PredictTheWinner(vector<int>& nums) {
        return helper(nums, 0, nums.size()-1)>=0;
    }
private:
    int helper(vector<int>& nums, int s, int e){
        return s==e ? nums[e] : max(nums[e] - helper(nums, s, e-
1), nums[s] - helper(nums, s+1, e));
    }
};
```

那怎么把Alpha-beta剪枝算法用在这一题呢？

447. Number of Boomerangs

Given n points in the plane that are all pairwise distinct, a "boomerang" is a tuple of points (i, j, k) such that the distance between i and j equals the distance between i and k (the order of the tuple matters).

Find the number of boomerangs. You may assume that n will be at most 500 and coordinates of points are all in the range [-10000, 10000] (inclusive).

Example:

Input:

[[0,0],[1,0],[2,0]]

Output:

2

Explanation:

The two boomerangs are [[1,0],[0,0],[2,0]] and [[1,0],[2,0],[0,0]]

翻译

给定 n 个两两各不相同的平面上的点，一个“回旋镖”是一个元组（tuple）的点（i,j,k），并且 i 和 j 的距离等于 i 和 k之间的距离（考虑顺序）。

找出回旋镖的个数。你可以假设 n 不大于500，点的坐标范围在[-10000, 10000]（包括边界）。

例子：

输入：

[[0,0],[1,0],[2,0]]

输出：

2

解释：

两个回旋镖分别是：[[1,0],[0,0],[2,0]] 和 [[1,0],[2,0],[0,0]]

分析

抓住两组点 (x1,y1)、(x2,y2) 和 (x1,y1)、(x3,y3) 之间的距离相等这个信息：distance = sqrt{(x1-x2)^2+(y1-y2)^2} = sqrt{(x1-x3)^2+(y1-y3)^2}

按照这种相等的距离，我们可以给所有点进行分类，相同距离的这些点（假设n个）可以构成一个排列组合中的排列：n*(n-1)个回旋镖。

代码

Python代码：

```
class Solution(object):
    def numberOfBoomerangs(self, points):
        """
        :type points: List[List[int]]
        :rtype: int
        """
        ans = 0
        for x1,y1 in points:
            dmap = collections.defaultdict(int)
            for x2,y2 in points:
                dmap[(x1-x2)**2+(y1-y2)**2] += 1
            for d in dmap:
                ans += dmap[d]*(dmap[d]-1)
        return ans
```

C++代码：

```
class Solution {
public:
    int numberOfBoomerangs(vector<pair<int, int>>& points) {
        int ans = 0;
        unordered_map<int, int> hash;
        for(int i=0;i<points.size();i++){
            for(int j=0;j<points.size();j++){
                hash[pow(points[i].first-
points[j].first,2)+pow(points[i].second-points[j].second,2)] +=
1;
            }
            for(auto d:hash){
                ans += d.second*(d.second-1);
            }
            hash.clear();
        }
        return ans;
    }
};
```

Java代码：

```
public class Solution {
    public int numberOfBoomerangs(int[][] points) {
        int ans = 0;
        HashMap<Integer,Integer> map = new
HashMap<Integer,Integer>();
        for(int i=0;i<points.length;i++){
            for(int j=0;j<points.length;j++){
                int d = (int)(Math.pow(points[i][0]-points[j]
[0],2) + Math.pow(points[i][1]-points[j][1],2));
                map.put(d, map.getOrDefault(d,0)+1);
            }
            for(int d:map.values()){
                ans += d*(d-1);
            }
            map.clear();
        }
        return ans;
    }
}
```

12. Integer to Roman

Given an integer, convert it to a roman numeral.

Input is guaranteed to be within the range from 1 to 3999.

翻译

给定一个整数，把它转化成罗马数字形式。

输入确保在 1 到 3999 的范围内。

分析

首先要了解罗马数字的构成规则：

- I: 1
- V: 5
- X: 10
- L: 50
- C: 100
- D: 500
- M: 1000

字母可以重复，但不超过三次（左减右加）：

- VI : 6
- IV : 4
- VII : 7
- VIII : 8
- IX : 9

代码

Python代码：

```
class Solution(object):
    def intToRoman(self, num):
        """
        :type num: int
        :rtype: str
        """
        values = [1000,900,500,400,100,90,50,40,10,9,5,4,1]
        strings =
["M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"]
        ans = ""
        for i in range(len(values)):
            while num>=values[i]:
                num -= values[i]
                ans += strings[i]
        return ans
```

C++代码：

```
class Solution {
public:
    string intToRoman(int num) {
        vector<string> strings =
{"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};
        int val[] = {1000,900,500,400,100,90,50,40,10,9,5,4,1};
        string ans="";
        for(int i=0; i<strings.size(); i++) {
            while(num>=val[i]) {
                num -= val[i];
                ans += strings[i];
            }
        }
        return ans;
    }
};
```

Java代码：

```java
public class Solution {
    public String intToRoman(int num) {
        String[] s =
{"M","CM","D","CD","C","XC","L","XL","X","IX","V","IV","I"};
        int[] v = {1000,900,500,400,100,90,50,40,10,9,5,4,1};
        String ans = "";
        for(int i=0;i<s.length;i++){
            while(num>=v[i]){
                num -= v[i];
                ans += s[i];
            }
        }
        return ans;
    }
}
```

第二遍，按算法体系总结

English Version

First，according to acception rate

Second, according to algorithm