# 一 数组

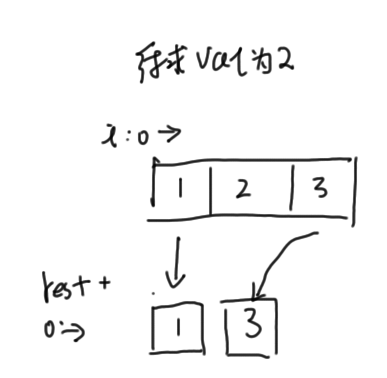
# 27 [[LeetCode] Remove Element 移除元素](https://www.cnblogs.com/grandyang/p/4606700.html)

Given an array and a value, remove all instances of that value in place and return the new length.

The order of elements can be changed. It doesn't matter what you leave beyond the new length.

这道题让我们移除一个数组中和给定值相同的数字，并返回新的数组的长度。是一道比较容易的题，我们只需要一个变量用来计数，然后遍历原数组，如果当前的值和给定值不同，我们就把当前值覆盖计数变量的位置，并将计数变量加1。代码如下：

1. 用后一项代替前面一项.



[复制代码](javascript:void(0);)

class Solution {

public:

int removeElement(vector<int>& nums, int val) {

int res = 0;

for (int i = 0; i < nums.size(); ++i) {

if (nums[i] != val) nums[res++] = nums[i];

}

return res;

}

};

[复制代码](javascript:void(0);)

# 26 [[LeetCode] Remove Duplicates from Sorted Array 有序数组中去除重复项](https://www.cnblogs.com/grandyang/p/4329128.html)

Given a sorted array nums, remove the duplicates [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) such that each element appear only once and return the new length.

Do not allocate extra space for another array, you must do this by modifying the input array [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

Example 1:

Given nums = [1,1,2],

Your function should return length = 2, with the first two elements of *nums* being 1 and 2 respectively.

It doesn't matter what you leave beyond the returned length.

Example 2:

Given nums = [0,0,1,1,1,2,2,3,3,4],

Your function should return length = 5, with the first five elements of *nums* being modified to 0, 1, 2, 3, and 4 respectively.

It doesn't matter what values are set beyond the returned length.

Clarification:

Confused why the returned value is an integer but your answer is an array?

Note that the input array is passed in by reference, which means modification to the input array will be known to the caller as well.

Internally you can think of this:

// nums is passed in by reference. (i.e., without making a copy)

int len = removeDuplicates(nums);

// any modification to nums in your function would be known by the caller.

// using the length returned by your function, it prints the first len elements.

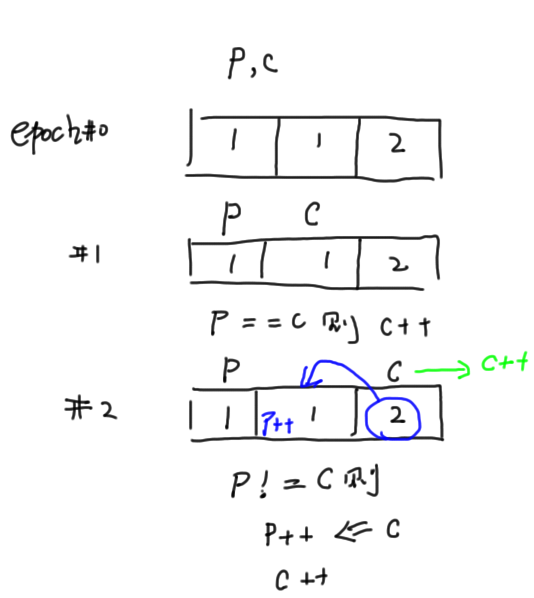
for (int i = 0; i < len; i++) {

    print(nums[i]);

}

这道题要我们从有序数组中去除重复项，和之前那道[Remove Duplicates from Sorted List](http://www.cnblogs.com/grandyang/p/4066453.html) 的题很类似，但是要简单一些，因为毕竟数组的值可以通过下标直接访问，而链表不行。那么这道题的解题思路是，我们使用快慢指针来记录遍历的坐标，最开始时两个指针都指向第一个数字，如果两个指针指的数字相同，则快指针向前走一步，如果不同，则两个指针都向前走一步，这样当快指针走完整个数组后，慢指针当前的坐标加1就是数组中不同数字的个数，代码如下：

1. 刚开始p和c在一个起跑线. 第一个数字两者必定相等.
2. P和c索引代表的值相等时, 更新c.
3. P和c不相等时, 需要先更新p, 然后把c赋值给p.
4. P表示的是**之前的已经”正确的”**(已经满足题目标准的, aka, 不是重复项)
5. 所以**之前的已经”正确”的**p不能被修改.
6. P要先移动到”不确定的”c的值.
7. 随后c的值要为下一次的判断更新位置.本次判断已经”使用”了c的值, 本次的c的值已经没有意义, 需要更新.
8. 题目约束, 输入的数组重复项是挨着的.



解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

int removeDuplicates(vector<int>& nums) {

if (nums.empty()) return 0;

int pre = 0, cur = 0, n = nums.size();

while (cur < n) {

if (nums[pre] == nums[cur]) ++cur;

else nums[++pre] = nums[cur++];

}

return pre + 1;

}

};

[复制代码](javascript:void(0);)

# 80 [[LeetCode] Remove Duplicates from Sorted Array II 有序数组中去除重复项之二](https://www.cnblogs.com/grandyang/p/4329295.html)

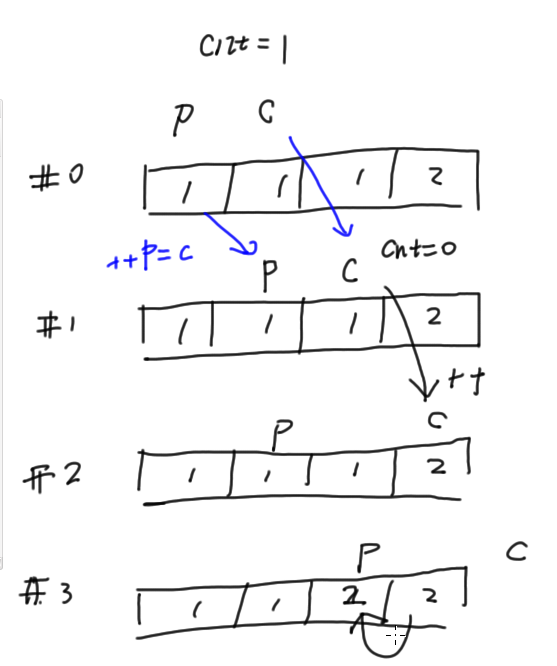
Follow up for "Remove Duplicates":  
What if duplicates are allowed at most twice?

For example,  
Given sorted array A = [1,1,1,2,2,3],

Your function should return length = 5, and A is now [1,1,2,2,3].

这道题是之前那道[Remove Duplicates from Sorted Array 有序数组中去除重复项](http://www.cnblogs.com/grandyang/p/4329128.html) 的延续，这里**允许最多重复的次数是两次**，那么我们就需要用一个变量count来记录还允许有几次重复，count初始化为1，如果出现过一次重复，则count递减1，那么下次再出现重复，快指针直接前进一步，如果这时候不是重复的，则count恢复1，由于整个数组是有序的，所以一旦出现不重复的数，则一定比这个数大，此数之后不会再有重复项。理清了上面的思路，则代码很好写了：

1. 设置一个counter,表示允许重复的个数.
2. Counter为0是临界. Counter从1开始倒数, 所以counter可以有2个值{aka, 0,1}, 表示可以允许的个数.
3. 只有在p=c并且cnt为0时,表示触及最大允许了, p和c对应的值不需要”覆盖”. 判断完c之后, c的值就需要更新了.



[复制代码](javascript:void(0);)

class Solution {

public:

int removeDuplicates(int A[], int n) {

if (n <= 2) return n;

int pre = 0, **cur = 1**, count = 1;

while (cur < n) {

if (A[pre] == A[cur] && count == 0) ++cur;

else {

if (A[pre] == A[cur]) --count;

else count = 1;

A[++pre] = A[cur++];

}

}

return pre + 1;

}

};

[复制代码](javascript:void(0);)

# 277 [[LeetCode] Find the Celebrity 寻找名人](https://www.cnblogs.com/grandyang/p/5310649.html)

Suppose you are at a party with n people (labeled from 0 to n - 1) and among them, there may exist one celebrity. The definition of a celebrity is that all the other n - 1people know him/her but he/she does not know any of them.

Now you want to find out who the celebrity is or verify that there is not one. The only thing you are allowed to do is to ask questions like: "Hi, A. Do you know B?" to get information of whether A knows B. You need to find out the celebrity (or verify there is not one) by asking as few questions as possible (in the asymptotic sense).

You are given a helper function bool knows(a, b) which tells you whether A knows B. Implement a function int findCelebrity(n), your function should minimize the number of calls to knows.

Note: There will be exactly one celebrity if he/she is in the party. Return the celebrity's label if there is a celebrity in the party. If there is no celebrity, return -1.

这道题让我们在一群人中寻找名人，**所谓名人就是每个人都认识他，他却不认识任何人**，限定了只有1个或0个名人，给定了一个API函数，输入a和b，用来判断a是否认识b，让我们尽可能少的调用这个函数，来找出人群中的名人。我最先想的方法是建立个一维数组用来标记每个人的名人候选状态，开始均初始化为true，表示每个人都是名人候选人，然后我们一个人一个人的验证其是否为名人，对于候选者i，我们遍历所有其他人j，如果i认识j，或者j不认识i，说明i不可能是名人，那么我们标记其为false，然后验证下一个候选者，反之如果i不认识j，或者j认识i，说明j不可能是名人，标记之。对于每个候选者i，如果遍历了一圈而其候选者状态仍为true，说明i就是名人，返回即可，如果遍历完所有人没有找到名人，返回-1，参见代码如下：

1. Knows函数是系统提供,用以判断a是否b认识.
2. Candidate是bool元素(成员)的vector, 默认每个人都是名人.
3. 两两遍历.候选人为i. j为测试人.
4. 名人是不认识别人,但是别人认识自己.
5. 题目限定,有0或1个名人.

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

int findCelebrity(int n) {

vector<bool> candidate(n, true);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

if (candidate[i] && i != j) {

**if (knows(i, j) || !knows(j, i))** {

candidate[i] = false;

break;

} else {

candidate[j] = false;

}

}

}

if (candidate[i]) return i;

}

return -1;

}

};

[复制代码](javascript:void(0);)

我们其实可以不用一维数组来标记每个人的状态，我们对于不是名人的i，直接break，继续检查下一个，但是由于我们没有标记后面的候选人的状态，所以有可能会重复调用一些knows函数，所以下面这种方法虽然省了空间，但是调用knows函数的次数可能会比上面的方法次数要多，参见代码如下：

# 189 [[LeetCode] Rotate Array 旋转数组](https://www.cnblogs.com/grandyang/p/4298711.html)

Given an array, rotate the array to the right by k steps, where k is non-negative.

Example 1:

Input: [1,2,3,4,5,6,7] and k = 3

Output: [5,6,7,1,2,3,4]

Explanation:

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

Example 2:

Input: [-1,-100,3,99] and k = 2

Output: [3,99,-1,-100]

Explanation:

rotate 1 steps to the right: [99,-1,-100,3]

rotate 2 steps to the right: [3,99,-1,-100]

Note:

* Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
* Could you do it in-place with O(1) extra space?

Credits:  
Special thanks to [@Freezen](https://oj.leetcode.com/discuss/user/Freezen) for adding this problem and creating all test cases.

新题抢先刷，这道题标为Easy，应该不是很难，我们先来看一种O(n)的空间复杂度的方法，我们复制一个和nums一样的数组，然后利用映射关系i -> (i+k)%n来交换数字。代码如下：

1. 先把之前的赋值一份为t
2. 利用公式更新.

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

void rotate(vector<int>& nums, int k) {

vector<int> t = nums;

for (int i = 0; i < nums.size(); ++i) {

**nums[(i + k) % nums.size()] = t[i]**;

}

}

};

[复制代码](javascript:void(0);)

根据热心网友[waruzhi](http://www.cnblogs.com/waruzhi/)的留言，这道题其实还有种类似翻转字符的方法，思路是先把前n-k个数字翻转一下，再把后k个数字翻转一下，最后再把整个数组翻转一下：

1 2 3 4 5 6 7   
4 3 2 1 5 6 7   
4 3 2 1 7 6 5  
5 6 7 1 2 3 4

解法三：

[复制代码](javascript:void(0);)

class Solution {

public:

void rotate(vector<int>& nums, int k) {

if (nums.empty() || (k %= nums.size()) == 0) return;

int n = nums.size();

reverse(nums.begin(), nums.begin() + n - k);

reverse(nums.begin() + n - k, nums.end());

reverse(nums.begin(), nums.end());

}

};



# 41 [[LeetCode] First Missing Positive 首个缺失的正数](https://www.cnblogs.com/grandyang/p/4395963.html)

Given an unsorted integer array, find the smallest missing positive integer.

Example 1:

Input: [1,2,0]

Output: 3

Example 2:

Input: [3,4,-1,1]

Output: 2

Example 3:

Input: [7,8,9,11,12]

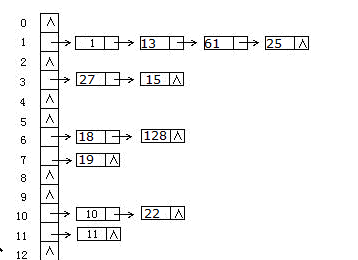
Output: 1

Note:

Your algorithm should run in O(n) time and uses constant extra space.

这道题让我们找缺失的首个正数，由于限定了O(n)的时间，所以一般的排序方法都不能用，最开始我没有看到还限制了空间复杂度，所以想到了用HashSet来解，这个思路很简单，第一遍遍历数组把所有的数都存入HashSet中，并且找出数组的最大值，下次循环从1开始递增找数字，哪个数字找不到就返回哪个数字，如果一直找到了最大的数字，则返回最大值+1，代码如下：

1. 找的是从1开始到该数组中最大正整数之间缺少的第一个正整数.
2. 首先, 找到数组所有正整数以及最大的正整数.
3. 然后从1开始到最大的正整数, 看第一个缺少的是哪个.
4. 使用了unordered\_set是基于哈希表.
5. 当用冲突发生时把具有同一关键码的数据组成一个链表



解法一：

[复制代码](javascript:void(0);)

// NOT constant space

class Solution {

public:

int firstMissingPositive(vector<int>& nums) {

int mx = 0;

unordered\_set<int> s;

for (int num : nums) {

if (num <= 0) continue;

s.insert(num);

mx = max(mx, num);

}

for (int i = 1; i <= mx; ++i) {

if (!s.count(i)) return i;

}

return mx + 1;

}

};

[复制代码](javascript:void(0);)

# 299 [[LeetCode] Bulls and Cows 公母牛游戏](https://www.cnblogs.com/grandyang/p/4929139.html)

You are playing the following Bulls and Cows game with your friend: You write a 4-digit secret number and ask your friend to guess it, each time your friend guesses a number, you give a hint, the hint tells your friend how many digits are in the correct positions (called "bulls") and how many digits are in the wrong positions (called "cows"), your friend will use those hints to find out the secret number.

For example:

Secret number: 1807

Friend's guess: 7810

Hint: 1 bull and 3 cows. (The bull is 8, the cows are 0, 1 and 7.)

According to [Wikipedia](https://en.wikipedia.org/wiki/Bulls_and_Cows): "Bulls and Cows (also known as Cows and Bulls or Pigs and Bulls or Bulls and Cleots) is an old code-breaking mind or paper and pencil game for two or more players, predating the similar commercially marketed board game Mastermind. The numerical version of the game is usually played with 4 digits, but can also be played with 3 or any other number of digits."

Write a function to return a hint according to the secret number and friend's guess, use A to indicate the bulls and B to indicate the cows, in the above example, your function should return 1A3B.

You may assume that the secret number and your friend's guess only contain digits, and their lengths are always equal.

Credits:  
Special thanks to [@jeantimex](https://leetcode.com/discuss/user/jeantimex) for adding this problem and creating all test cases.

[Subscribe](https://leetcode.com/subscribe/) to see which companies asked this question

这道题提出了一个叫公牛母牛的游戏，其实就是之前文曲星上有的猜数字的游戏，有一个四位数字，你猜一个结果，然后根据你猜的结果和真实结果做对比，**提示有多少个数字和位置都正确的叫做bulls，还提示有多少数字正确但位置不对的叫做cows**，根据这些信息来引导我们继续猜测正确的数字。这道题并没有让我们实现整个游戏，而只用实现一次比较即可。给出两个字符串，让我们找出分别几个bulls和cows。这题需要用哈希表，来建立数字和其出现次数的映射。我最开始想的方法是用两次遍历，第一次遍历找出所有位置相同且值相同的数字，即bulls，并且记录secret中不是bulls的数字出现的次数。然后第二次遍历我们针对guess中不是bulls的位置，如果在哈希表中存在，cows自增1，然后映射值减1，参见如下代码：

1. 数字和位置都正确叫做bull
2. 数字正确但是位置不正确的是cow.
3. 哈希表, 数字和次数映射.
4. M保存的是secret的非bull的元素的个数.
5. M[guess[i]]的含义是: guess[i]是输入guess的, guess的i字符在表示secret的m中个数是多少, 意味着secret也有guess的值.
6. 有一个就增加一个cows.

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

string getHint(string secret, string guess) {

int m[256] = {0}, bulls = 0, cows = 0;

for (int i = 0; i < secret.size(); ++i) {

if (secret[i] == guess[i]) ++bulls;

else ++m[secret[i]];

}

for (int i = 0; i < secret.size(); ++i) {

if (secret[i] != guess[i] && m[guess[i]]) {

++cows;

--m[guess[i]];

}

}

return to\_string(bulls) + "A" + to\_string(cows) + "B";

}

};

[复制代码](javascript:void(0);)

# 134 [[LeetCode] Gas Station 加油站问题](https://www.cnblogs.com/grandyang/p/4266812.html)

There are N gas stations along a circular route, where the amount of gas at station i is gas[i].

You have a car with an unlimited gas tank and it costs cost[i] of gas to travel from station i to its next station (i+1). You begin the journey with an empty tank at one of the gas stations.

Return the starting gas station's index if you can travel around the circuit once, otherwise return -1.

Note:  
The solution is guaranteed to be unique.

这道转圈加油问题不算很难，只要想通其中的原理就很简单。我们首先要知道能走完整个环的前提是gas的总量要大于cost的总量，这样才会有起点的存在。假设开始设置起点start = 0, 并从这里出发，如果当前的gas值大于cost值，就可以继续前进，此时到下一个站点，剩余的gas加上当前的gas再减去cost，看是否大于0，若大于0，则继续前进。当到达某一站点时，若这个值小于0了，则说明从起点到这个点中间的任何一个点都不能作为起点，则把起点设为下一个点，继续遍历。当遍历完整个环时，当前保存的起点即为所求。代码如下：

1. Total描述了这个gas图是否能够跑通.
2. 如果总共的消耗比加油值还多,那无论从哪一站起步都不会跑完一圈.
3. 如果total值大于零, 则说明肯定至少有一站开始可以跑完.
4. 肯定有情况,是起步都起步不了的.
5. 也有情况,起步跑几站就跑不了了.
6. 必然有一站加的油足够多, 能够跑完.
7. 从i=0开始遍历.
8. 只要有一站的加油量(累计油量)不足以跑完i站和i+1站之间的距离, 就把七点设置为i+1站.同时累计sum归零.

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {

int total = 0, sum = 0, start = 0;

for (int i = 0; i < gas.size(); ++i) {

total += gas[i] - cost[i];

sum += gas[i] - cost[i];

if (sum < 0) {

start = i + 1;

sum = 0;

}

}

return (total < 0) ? -1 : start;

}

};

[复制代码](javascript:void(0);)

# 118 [[LeetCode] Pascal's Triangle 杨辉三角](https://www.cnblogs.com/grandyang/p/4032449.html)

Given a non-negative integer numRows, generate the first numRows of Pascal's triangle.

  
In Pascal's triangle, each number is the sum of the two numbers directly above it.

Example:

Input: 5

Output:

[

[1],

[1,1],

[1,2,1],

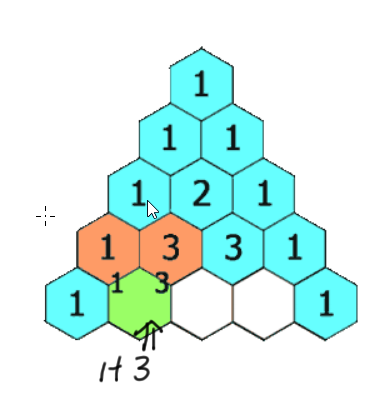
[1,3,3,1],

[1,4,6,4,1]

]

杨辉三角是二项式系数的一种写法，如果熟悉杨辉三角的五个性质，那么很好生成，可参见我的上一篇博文[Pascal's Triangle II](http://www.cnblogs.com/grandyang/p/4031536.html)。具体生成算是：**每一行的首个和结尾一个数字都是1，从第三行开始，中间的每个数字都是上一行的左右两个数字之和**。代码如下：

1. 每行收尾两个值为1.
2. 第三行开始, 值为上一行对应位置和上一行对应位置的左边的值之和.
3. 二重vector<vector<int>>
4. Vector的resize能够为每行vector<int>配置空间, 这里是i+1空间, 初始值为1.
5. 考虑到第0行有1个元素.
6. 第1行有2个
7. 第2行有3个.等等…



[复制代码](javascript:void(0);)

class Solution {

public:

**vector<vector<int>>** generate(int numRows) {

vector<vector<int>> res(numRows, vector<int>());

for (int i = 0; i < numRows; ++i) {

res[i].resize(i + 1, 1);

for (int j = 1; j < i; ++j) {

res[i][j] = res[i - 1][j - 1] + res[i - 1][j];

}

}

return res;

}

};

[复制代码](javascript:void(0);)

# 119 [[LeetCode] Pascal's Triangle II 杨辉三角之二](https://www.cnblogs.com/grandyang/p/4031536.html)

Given a non-negative index k where k ≤ 33, return the kth index row of the Pascal's triangle.

Note that the row index starts from 0.

  
In Pascal's triangle, each number is the sum of the two numbers directly above it.

Example:

Input: 3

Output: [1,3,3,1]

Follow up:

Could you optimize your algorithm to use only O(k) extra space?

杨辉三角想必大家并不陌生，应该最早出现在初高中的数学中，其实就是二项式系数的一种写法。

　　　　　　　　１

　　　　　　　１　１

　　　　　　１　２　１

　　　　　１　３　３　１

　　　　１　４　６　４　１

　　　１　５　10　10　５　１

　　１　６　15　20　15　６　１

　１　７　21　35　35　21　７　１

１　８　28　56　70　56　28　８　１

杨辉三角形第n层（顶层称第0层，第1行，第n层即第n+1行，此处n为包含0在内的自然数）正好对应于二项式\left(a+b\right)^{n}展开的系数。例如第二层1 2 1是幂指数为2的二项式\left(a+b\right)^{2}展开形式a^{2}+2ab+b^{2}的系数。

**杨辉三角主要有下列五条性质：**

1. **杨辉三角以正整数构成，数字左右对称，每行由1开始逐渐变大，然后变小，回到1。**
2. **第n行的数字个数为n个。**
3. **第n行的第k个数字为组合数C_{n-1}^{k-1}。**
4. **第n行数字和为2^{n-1}。**
5. **除每行最左侧与最右侧的数字以外，每个数字等于它的左上方与右上方两个数字之和（也就是说，第n行第k个数字等于第n-1行的第k-1个数字与第k个数字的和）。这是因为有组合恒等式：C_{n}^{i}=C_{n-1}^{i-1}+C_{n-1}^{i}。可用此性质写出整个杨辉三角形。**

由于题目有额外限制条件，程序只能使用O(k)的额外空间，那么这样就不能把每行都算出来，而是要用其他的方法, 我最先考虑用的是第三条性质，算出每个组合数来生成第n行系数，代码参见评论区一楼。本地调试输出前十行，没啥问题，拿到OJ上测试，程序在第18行跪了，中间有个系数不正确。那么问题出在哪了呢，仔细找找，原来出在计算组合数那里，由于算组合数时需要算连乘，而整形数int的数值范围只有-32768到32768之间，那么一旦n值过大，连乘肯定无法计算。而丧心病狂的OJ肯定会测试到成百上千行，所以这个方法不行。那么我们再来考虑利用第五条性质，除了第一个和最后一个数字之外，其他的数字都是上一行左右两个值之和。那么我们只需要两个for循环，除了第一个数为1之外，后面的数都是上一次循环的数值加上它前面位置的数值之和，不停地更新每一个位置的值，便可以得到第n行的数字，具体实现代码如下：

[复制代码](javascript:void(0);)

class Solution {

public:

vector<int> getRow(int rowIndex) {

vector<int> res(rowIndex + 1);

res[0] = 1;

for (int i = 1; i <= rowIndex; ++i) {

for (int j = i; j >= 1; --j) {

res[j] += res[j - 1];

}

}

return res;

}

};

[复制代码](javascript:void(0);)

# 169 [[LeetCode] Majority Element 求大多数](https://www.cnblogs.com/grandyang/p/4233501.html)

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.

Example 1:

Input: [3,2,3]

Output: 3

Example 2:

Input: [2,2,1,1,1,2,2]

Output: 2

这是到求大多数的问题，有很多种解法，其中我感觉比较好的有两种，一种是用哈希表，这种方法需要 O(n) 的时间和空间，另一种是用一种叫摩尔投票法 Moore Voting，需要 O(n) 的时间和 O(1) 的空间，比前一种方法更好。这种投票法先将第一个数字假设为过半数，然后把计数器设为1，比较下一个数和此数是否相等，若相等则计数器加一，反之减一。然后看此时计数器的值，若为零，则将下一个值设为候选过半数。以此类推直到遍历完整个数组，当前候选过半数即为该数组的过半数。不仔细弄懂摩尔投票法的精髓的话，过一阵子还是会忘记的，首先要明确的是这个叼炸天的方法是有前提的，就是数组中一定要有过半数的存在才能使用，下面我们来看本算法的思路，这是一种先假设候选者，然后再进行验证的算法。我们现将数组中的第一个数假设为过半数，然后进行统计其出现的次数，如果遇到同样的数，则计数器自增1，否则计数器自减1，如果计数器减到了0，则更换下一个数字为候选者。这是一个很巧妙的设定，也是本算法的精髓所在，为啥遇到不同的要计数器减1呢，为啥减到0了又要更换候选者呢？首先是有那个强大的前提存在，一定会有一个出现超过半数的数字存在，那么如果计数器减到0了话，说明目前不是候选者数字的个数已经跟候选者的出现个数相同了，那么这个候选者已经很weak，不一定能出现超过半数，我们选择更换当前的候选者。那有可能你会有疑问，那万一后面又大量的出现了之前的候选者怎么办，不需要担心，如果之前的候选者在后面大量出现的话，其又会重新变为候选者，直到最终验证成为正确的过半数，佩服算法的提出者啊，代码如下：

C++ 解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

int majorityElement(vector<int>& nums) {

int res = 0, cnt = 0;

for (int num : nums) {

if (cnt == 0) {res = num; ++cnt;}

else (num == res) ? ++cnt : --cnt;

}

return res;

}

};

[复制代码](javascript:void(0);)

# 229 [[LeetCode] Majority Element II 求大多数之二](https://www.cnblogs.com/grandyang/p/4606822.html)

Given an integer array of size n, find all elements that appear **more than ⌊ n/3 ⌋ times**.

Note: The algorithm should run in linear time and in O(1) space.

Example 1:

Input: [3,2,3]

Output: [3]

Example 2:

Input: [1,1,1,3,3,2,2,2]

Output: [1,2]

这道题让我们**求出现次数大于 n/3 的数字**，而且限定了时间和空间复杂度，那么就不能排序，也不能使用 HashMap，这么苛刻的限制条件只有一种方法能解了，那就是**摩尔投票法 Moore Voting**，这种方法在之前那道题 [Majority Element](http://www.cnblogs.com/grandyang/p/4233501.html) 中也使用了。题目中给了一条很重要的提示，让我们先考虑可能会有多少个这样的数字，经过举了很多例子分析得出，**任意一个数组出现次数大于 n/3 的数最多有两个**，具体的证明我就不会了，我也不是数学专业的（热心网友[用手走路](https://www.cnblogs.com/grandyang/p/4606822.html#3277653)提供了证明：如果有超过两个，也就是至少三个数字满足“出现的次数大于 n/3”，那么就意味着数组里总共有超过 3\*(n/3) = n 个数字，这与已知的数组大小矛盾，所以，只可能有两个或者更少）。那么有了这个信息，我们使用投票法的核心是找出两个候选数进行投票，需要两遍遍历，第一遍历找出两个候选数，第二遍遍历重新投票验证这两个候选数是否为符合题意的数即可，选候选数方法和前面那篇 [Majority Element](http://www.cnblogs.com/grandyang/p/4233501.html) 一样，由于之前那题题目中限定了一定会有大多数存在，故而省略了验证候选众数的步骤，这道题却没有这种限定，即满足要求的大多数可能不存在，所以要有验证。代码如下：

1. **出现大于n/3次数的数字, 最多有2个.**
2. **相同数字是连续摆放的.**
3. **使用a和b来描述大于n/3的数字. a和b地位是相等的. 所以下面的处理步骤也是相同的.**
4. **a, b的初值为0. Cnt1和cnt2初值为0.**
5. **对于初值的更新处理是相同的.**
6. **在num和a,b值不同情况下,Cnt1和cnt2为0表示需要更新初值.**
7. **Cnt1和cnt2非0, 说明已经有两个候选a和b.**
8. **仅仅是候选哦, 并不一定说此时的a和b就是期望值(大于n/3次数的值)了**
9. **参考[1,1,1,3,3,2,2,2], 遍历中间的两个3的时候,a值记录的是1, 而b值记录的是3, 但是实际上b应该为2.**
10. **在a,b都有候选值(cnt1和cnt2非0)情况下, 遍历一个num和a,b都不等,此时类似上面的”3”的情况, 对a和b得cnt都减一(a和b地位等价).**
11. **a,b必然有个cnt值小. 它先减到0. 然后它重新记录新的值.**
12. **遍历nums. 如果num和a或b相等, 增加a和b对应的计数.**
13. **最后再检查下是否满足n/3的条件(a和b).**

[复制代码](javascript:void(0);)

class Solution {

public:

vector<int> majorityElement(vector<int>& nums) {

vector<int> res;

int a = 0, b = 0, cnt1 = 0, cnt2 = 0, n = nums.size();

for (int num : nums) {

if (num == a) ++cnt1;

else if (num == b) ++cnt2;

else if (cnt1 == 0) { a = num; cnt1 = 1; }

else if (cnt2 == 0) { b = num; cnt2 = 1; }

else { --cnt1; --cnt2; }

}

cnt1 = cnt2 = 0;

for (int num : nums) {

if (num == a) ++cnt1;

else if (num == b) ++cnt2;

}

if (cnt1 > n / 3) res.push\_back(a);

if (cnt2 > n / 3) res.push\_back(b);

return res;

}

};

[复制代码](javascript:void(0);)

# 274 [[LeetCode] H-Index 求H指数](https://www.cnblogs.com/grandyang/p/4781203.html)

Given an array of citations (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the [definition of h-index on Wikipedia](https://en.wikipedia.org/wiki/H-index): "A scientist has index h if h of his/her N papers have at leasth citations each, and the other N − h papers have no more than h citations each."

Example:

Input: citations = [3,0,6,1,5]

Output: 3

Explanation: [3,0,6,1,5] means the researcher has 5 papers in total and each of them had

received 3, 0, 6, 1, 5 citations respectively.

  Since the researcher has 3 papers with at least 3 citations each and the remaining

  two with no more than 3 citations each, her h-index is 3.

Note: If there are several possible values for h, the maximum one is taken as the h-index.

这道题让我们求[H指数](https://zh.wikipedia.org/wiki/H%E6%8C%87%E6%95%B0)，这个质数是用来衡量研究人员的学术水平的质数，定义为一个人的学术文章有n篇分别被引用了n次，那么H指数就是n。而且wiki上直接给出了算法，可以按照如下方法**确定某人的H指数**：

**1、将其发表的所有SCI论文按被引次数从高到低排序；**

**2、从前往后查找排序后的列表，直到某篇论文的序号大于该论文被引次数。所得序号减一即为H指数。**

我也就没多想，直接按照上面的方法写出了代码：

[复制代码](javascript:void(0);)

class Solution {

public:

int hIndex(vector<int>& citations) {

sort(citations.begin(), citations.end(), greater<int>());

for (int i = 0; i < citations.size(); ++i) {

if (i >= citations[i]) return i;

}

return citations.size();

}

};

[复制代码](javascript:void(0);)

# 275 [[LeetCode] H-Index II 求H指数之二](https://www.cnblogs.com/grandyang/p/4782695.html)

Given an array of citations sorted in ascending order (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the [definition of h-index on Wikipedia](https://en.wikipedia.org/wiki/H-index): "A scientist has index h if h of his/her N papers have at least h citations each, and the other N − h papers have no more than h citations each."

Example:

Input: citations = [0,1,3,5,6]

Output: 3

Explanation: [0,1,3,5,6] means the researcher has 5 papers in total and each of them had

received 0, 1, 3, 5, 6 citations respectively.

  Since the researcher has 3 papers with at least 3 citations each and the remaining

  two with no more than 3 citations each, her h-index is 3.

Note:

If there are several possible values for h, the maximum one is taken as the h-index.

Follow up:

* This is a follow up problem to [H-Index](https://leetcode.com/problems/h-index/description/), where citations is now guaranteed to be sorted in ascending order.
* Could you solve it in logarithmic time complexity?

这题是之前那道 [H-Index](http://www.cnblogs.com/grandyang/p/4781203.html) 的拓展，输入数组**是有序的**(**从高到低的**)，让我们在 O(log n) 的时间内完成计算，看到这个时间复杂度，而且数组又是有序的，应该有很敏锐的意识应该用**二分查找法**，属于博主之前的总结帖 [LeetCode Binary Search Summary 二分搜索法小结](https://www.cnblogs.com/grandyang/p/6854825.html) 中的第五类，目标值 target 会随着 mid 值的变化而变化，这里的 right 的初始值和 while 循环条件是否加等号是需要注意的问题，一般来说，博主的习惯是把 right 初始化为数组的长度，然后循环条件中不加等号，但是这种 right 的初始化对于这种目标值不固定的情况下不好使，需要初始化为长度减1（目前博主还没有遇到反例，有的话请务必告知博主）。那么此时循环条件中是否要加等号，这个其实很玄学，在 [Find Peak Element](https://www.cnblogs.com/grandyang/p/4217175.html#4111463) 中，right 也是初始化为数组长度减1，但是循环条件却不能加等号。这道题却一定需要加等号，否则会跪在 [0] 这个 test case，有些时候固有的规律并不好使，可能只能代一些 corner case 来进行检验，比如 [], [0], [1,2] 这种最简便的例子。

基于上面的分析，我们最先初始化 left 和 right 为0和 len-1，然后取中间值 mid，比较 citations[mid] 和 len-mid 做比较，如果前者大，则 right 移到 mid 之前，反之 right 移到 mid 之后，循环条件是 left<=right，最后返回 len-left 即可，参见代码如下：

1. 变化为一个二分查找问题. 引用值从高到低排序.
2. 设置r,l和mid.
3. Mid是索引,取l和r的中间值.
4. Citation[mid]是这个索引对应的citation值(引用次数).
5. 找到Hindex的标志是: citation[mid]==len-mid.

[复制代码](javascript:void(0);)

class Solution {

public:

int hIndex(vector<int>& citations) {

int len = citations.size(), left = 0, right = len - 1;

while (left <= right) {

int mid = 0.5 \* (left + right);

if (citations[mid] == len - mid) return len - mid;

else if (citations[mid] > len - mid) right = mid - 1;

else left = mid + 1;

}

return len - left;

}

};

[复制代码](javascript:void(0);)

# 243 [[LeetCode] Shortest Word Distance 最短单词距离](https://www.cnblogs.com/grandyang/p/5187041.html)

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

For example,  
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = “coding”, word2 = “practice”, return 3.  
Given word1 = "makes", word2 = "coding", return 1.

Note:  
You may assume that word1 does not equal to word2, and word1 and word2 are both in the list.

这道题让我们**给了我们一个单词数组，又给定了两个单词**，让我们求这两个单词之间的最小距离，限定了两个单词不同，而且都在数组中。我最先想到的方法比较笨，我首先想的是要用哈希表来做，建立每个单词和其所有出现位置数组的映射，但是后来想想，反正建立映射也要遍历一遍数组，我们还不如直接遍历一遍数组，直接把两个给定单词所有出现的位置分别存到两个数组里，然后我们在对两个数组进行两两比较更新结果，参见代码如下：

1. 两个单词是不同的.
2. Push\_back是把元素push到vector的方法(放到向量的末尾). 会自己管理(增加)内存.

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

int shortestDistance(vector<string>& words, string word1, string word2) {

vector<int> idx1, idx2;

int res = INT\_MAX;

for (int i = 0; i < words.size(); ++i) {

if (words[i] == word1) idx1.push\_back(i);

else if (words[i] == word2) idx2.push\_back(i);

}

for (int i = 0; i < idx1.size(); ++i) {

for (int j = 0; j < idx2.size(); ++j) {

res = min(res, abs(idx1[i] - idx2[j]));

}

}

return res;

}

};

[复制代码](javascript:void(0);)

# 244 [[LeetCode] Shortest Word Distance II 最短单词距离之二](https://www.cnblogs.com/grandyang/p/5187640.html)

This is a follow up of [Shortest Word Distance](https://leetcode.com/problems/shortest-word-distance). The only difference is now you are given the list of words and your method will be called repeatedly many times with different parameters. How would you optimize it?

Design a class which receives a list of words in the constructor, and implements a method that takes two words word1 and word2 and return the shortest distance between these two words in the list.

For example,  
Assume that words = ["practice", "makes", "perfect", "coding", "makes"].

Given word1 = “coding”, word2 = “practice”, return 3.  
Given word1 = "makes", word2 = "coding", return 1.

Note:  
You may assume that word1 does not equal to word2, and word1 and word2 are both in the list.

这道题是之前那道[Shortest Word Distance](http://www.cnblogs.com/grandyang/p/5187041.html)的拓展，不同的是这次我们**需要多次调用求最短单词距离的函数**，那么用之前那道题的解法二和三就非常不高效，而当时我们摒弃的解法一的思路却可以用到这里，**我们用哈希表来建立每个单词和其所有出现的位置的映射**，然后在找最短单词距离时，我们只需要取出该单词在哈希表中映射的位置数组进行两两比较即可，参见代码如下：

解法一：

[复制代码](javascript:void(0);)

class WordDistance {

public:

WordDistance(vector<string>& words) {

for (int i = 0; i < words.size(); ++i) {

m[words[i]].push\_back(i);

}

}

int shortest(string word1, string word2) {

int res = INT\_MAX;

for (int i = 0; i < m[word1].size(); ++i) {

for (int j = 0; j < m[word2].size(); ++j) {

res = min(res, abs(m[word1][i] - m[word2][j]));

}

}

return res;

}

private:

unordered\_map<string, vector<int> > m;

};

[复制代码](javascript:void(0);)

# 245 [[LeetCode] Shortest Word Distance III 最短单词距离之三](https://www.cnblogs.com/grandyang/p/5192426.html)

This is a follow up of Shortest Word Distance. The only difference is now word1 could be the same as word2.

Given a list of words and two words word1 and word2, return the shortest distance between these two words in the list.

word1 and word2 may be the same and they represent two individual words in the list.

For example,  
Assume that words = ["practice", **"makes"**, "perfect", "coding", **"makes"**].

Given word1 = “makes”, word2 = “coding”, return 1.  
**Given word1 = "makes", word2 = "makes", return 3.**

Note:  
You may assume word1 and word2 are both in the list.

这道题还是让我们求最短单词距离，有了之前两道题[Shortest Word Distance II](http://www.cnblogs.com/grandyang/p/5187640.html)和[Shortest Word Distance](http://www.cnblogs.com/grandyang/p/5187041.html)的基础，就大大降低了题目本身的难度。**这道题增加了一个条件，就是说两个单词可能会相同**，

上述代码其实可以优化一下，我们并不需要变量t来记录上一个位置，**我们将p1初始化为数组长度，p2初始化为数组长度的相反数**，然后当word1和word2相等的情况，我们用**p1来保存p2的结果，p2赋为当前的位置i**，这样我们就可以更新结果了，如果word1和word2不相等，则还跟原来的做法一样，这种思路真是挺巧妙的，参见代码如下：

1. **当word1和word2不相同时, 退化为找到word1时,记录其位置, 找到word2时记录其位置.然后位置abs差.**
2. **当兼容word1和word2相同(但是其对应的index不同)时, 用p1和p2表示word1和word2. 找到word1时(因为相等,也就找到了word2), 这时只把p2的值更新, p1的值跟新为p2的上一次的值.**
3. **因为有上一次的含义, 这样在找到相同的第二个时, 还是更新p2为最新的这个位置, p1更新为p2上一个值(也就是第一个的位置)**

解法二：

[复制代码](javascript:void(0);)

class Solution {

public:

int shortestWordDistance(vector<string>& words, string word1, string word2) {

int p1 = words.size(), p2 = -words.size(), res = INT\_MAX;

for (int i = 0; i < words.size(); ++i) {

if (words[i] == word1) p1 = word1 == word2 ? p2 : i;

if (words[i] == word2) p2 = i;

res = min(res, abs(p1 - p2));

}

return res;

}

};

# 217 [[LeetCode] Contains Duplicate 包含重复值](https://www.cnblogs.com/grandyang/p/4537029.html)

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.

这道题不算难题，就是使用一个哈希表，遍历整个数组，如果哈希表里存在，返回false，如果不存在，则将其放入哈希表中，代码如下：

 解法一

[复制代码](javascript:void(0);)

class Solution {

public:

bool containsDuplicate(vector<int>& nums) {

unordered\_map<int, int> m;

for (int i = 0; i < nums.size(); ++i) {

if (m.find(nums[i]) != m.end()) return true;

++m[nums[i]];

}

return false;

}

};

[复制代码](javascript:void(0);)

# 219 [[LeetCode] Contains Duplicate II 包含重复值之二](https://www.cnblogs.com/grandyang/p/4539680.html)

Given an array of integers and an integer k, find out whether there are two distinct indices i and j in the array such that **nums[i] = nums[j]** and the difference between i and j is at most k. (New Version)

 这道题是之前那道[Contains Duplicate 包含重复值](http://www.cnblogs.com/grandyang/p/4537029.html)的延伸，不同之处在于那道题只要我们判断下数组中是否有重复值，而这道题限制了数组中只许有一组重复的数字，而且他们坐标差不能超过k。

坑爹啊，题目要求变了，那么就没啥歧义了，正确解法如下：

1. 还是用unordered\_map保存出现的值及其**坐标**
2. 217是保存num值及其出现**次数.**
3. **这里是坐标(或index值)**
4. 判断true的条件加上重复值偏差不超过k.

[复制代码](javascript:void(0);)

class Solution {

public:

bool containsNearbyDuplicate(vector<int>& nums, int k) {

unordered\_map<int, int> m;

for (int i = 0; i < nums.size(); ++i) {

if (m.find(nums[i]) != m.end() && i - m[nums[i]] <= k) return true;

else m[nums[i]] = i;

}

return false;

}

};

# 220 [[LeetCode] Contains Duplicate III 包含重复值之三](https://www.cnblogs.com/grandyang/p/4545261.html)

Given an array of integers, find out whether there are two distinct indices i and j in the array such that the difference between **nums[i]** and **nums[j]** is at most t and the difference between i and j is at most k.

这道题跟之前两道[Contains Duplicate 包含重复值](http://www.cnblogs.com/grandyang/p/4537029.html)和[Contains Duplicate II 包含重复值之二](http://www.cnblogs.com/grandyang/p/4539680.html)的关联并不是很大，前两道起码跟重复值有关，这道题的焦点**不是在重复值上面**，反而是关注与不同的值之间的关系，**这里有两个限制条件，两个数字的坐标差不能大于k，值差不能大于t。**这道题如果用brute force会超时，所以我们只能另辟蹊径。这里我们使用map数据结构来解,用来记录数字和其下标之间的映射。 这里需要两个指针i和j，刚开始i和j都指向0，然后i开始向右走遍历数组，如果i和j之差大于k，且m中有nums[j]，则删除并j加一。这样保证了m中所有的数的下标之差都不大于k，然后我们用map数据结构的lower\_bound()函数来找一个特定范围，就是大于或等于nums[i] - t的地方，所有小于这个阈值的数和nums[i]的差的绝对值会大于t (可自行带数检验)。然后检测后面的所有的数字，如果数的差的绝对值小于等于t，则返回true。最后遍历完整个数组返回false。代码如下：

1. **不要求重复值.**
2. **关注在I,j坐标差小于k, nums[i]和nums[j]差小于t.**
3. **用map记录nums值及其坐标.**
4. **遍历I, j从0开始.**
5. **删除i-j差大于k的所有map的key-value对.**
6. **然后在map中找满足条件的**



[复制代码](javascript:void(0);)

class Solution {

public:

bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t) {

map<long long, int> m;

int j = 0;

for (int i = 0; i < nums.size(); ++i) {

if (i - j > k) m.erase(nums[j++]);

auto a = m.lower\_bound((long long)nums[i] - t);

if (a != m.end() && abs(**a->first** - nums[i]) <= t) return true;

m[nums[i]] = i;

}

return false;

}

};

[复制代码](javascript:void(0);)

# 55 [[LeetCode] Jump Game 跳跃游戏](https://www.cnblogs.com/grandyang/p/4371526.html)

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

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

Determine if you are able to reach the last index.

Example 1:

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

Output: true

Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: [3,2,1,0,4]

Output: false

Explanation: You will always arrive at index 3 no matter what. Its maximum

  jump length is 0, which makes it impossible to reach the last index.

这道题说的是有**一个非负整数**的数组，**每个数字表示在当前位置的基础上最多可以走的步数，求判断能不能到达最后一个位置**，开始我以为是必须刚好到达最后一个位置，超过了不算，其实是理解题意有误，因为每个位置上的数字表示的是最多可以走的步数而不是像玩大富翁一样摇骰子摇出几一定要走几步。那么我们可以**用动态规划Dynamic Programming来解**，我们维护一个一位数组dp，**其中dp[i]表示达到i位置时剩余的步数**，那么难点就是推导状态转移方程啦。我们想啊，**到达当前位置的剩余步数跟什么有关呢，其实是跟上一个位置的剩余步数和上一个位置的跳力有关**，这里的跳力就是原数组中每个位置的数字，因为其代表了以当前位置为起点能到达的最远位置。所以**当前位置的剩余步数（dp值）和当前位置的跳力中的较大那个数决定了当前能到的最远距离，而下一个位置的剩余步数（dp值）就等于当前的这个较大值减去1**，因为需要花一个跳力到达下一个位置，所以我们就有状态转移方程了：**dp[i] = max(dp[i - 1], nums[i - 1]) - 1**，如果当某一个时刻dp数组的值为负了，说明无法抵达当前位置，则直接返回false，最后循环结束后直接返回true即可，代码如下：

1. **维护dp方程.**
2. **上一次剩余dp[i-1]步数比上一次跳力大, 则这次剩余的跳力肯定是采用上一次大的跳力(aka, 上一次剩余的dp[i-1]), 而到达本次必须消耗一个跳力.所以要减1.**
3. **如果上一次剩余跳力不足(小于)上一次提供的跳力. 肯定也是使用最大跳力.**
4. **如果有一次的dp[i]小于0, 可以提前返回(直接返回)false**

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

bool canJump(vector<int>& nums) {

vector<int> dp(nums.size(), 0);

for (int i = 1; i < nums.size(); ++i) {

dp[i] = max(dp[i - 1], nums[i - 1]) - 1;

if (dp[i] < 0) return false;

}

return true;

}

};

# 45 [[LeetCode] Jump Game II 跳跃游戏之二](https://www.cnblogs.com/grandyang/p/4373533.html)

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

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

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

For example:  
Given array A = [2,3,1,1,4]

The minimum number of jumps to reach the last index is 2. (Jump 1 step from index 0 to 1, then 3 steps to the last index.)

这题是之前那道[Jump Game 跳跃游戏](http://www.cnblogs.com/grandyang/p/4371526.html) 的延伸，**那题是问能不能到达最后一个数字，而此题只让我们求到达最后一个位置的最少跳跃数**，貌似是默认一定能到达最后位置的? 此题的核心方法是利用贪婪算法Greedy的思想来解，想想为什么呢？ 为了较快的跳到末尾，我们想知道每一步能跳的范围，这里贪婪并不是要在能跳的范围中选跳力最远的那个位置，因为这样选下来不一定是最优解，这么一说感觉又有点不像贪婪算法了。我们这里贪的是一个能到达的最远范围，我们遍历当前跳跃能到的所有位置，然后根据该位置上的跳力来预测下一步能跳到的最远距离，贪出一个最远的范围，一旦当这个范围到达末尾时，当前所用的步数一定是最小步数。**我们需要两个变量cur和pre分别来保存当前的能到达的最远位置和之前能到达的最远位置**，只要cur未达到最后一个位置则循环继续，pre先赋值为cur的值，表示上一次循环后能到达的最远位置，如果当前位置i小于等于pre，说明还是在上一跳能到达的范围内，**我们根据当前位置加跳力来更新cur，更新cur的方法是比较当前的cur和i + A[i]之中的较大值**，如果题目中未说明是否能到达末尾，我们还可以判断此时pre和cur是否相等，如果相等说明cur没有更新，即无法到达末尾位置，返回-1，代码如下：

解法一：

[复制代码](javascript:void(0);)

class Solution {

public:

int jump(vector<int>& nums) {

int res = 0, n = nums.size(), i = 0, cur = 0;

while (cur < n - 1) {

++res;

int pre = cur;

for (; i <= pre; ++i) { **// 在上一次能跳的范围下. 遍历上一次能跳的范围. 目的找到下一次能跳的最大的值**.

cur = max(cur, i + nums[i]);**// i+nums[i]是说在第i台阶上(不一定是第i个, 就是编号是i的台阶上)能调到哪个编号的台阶.**

}

if (pre == cur) return -1; // May not need this

}

return res;

}

};