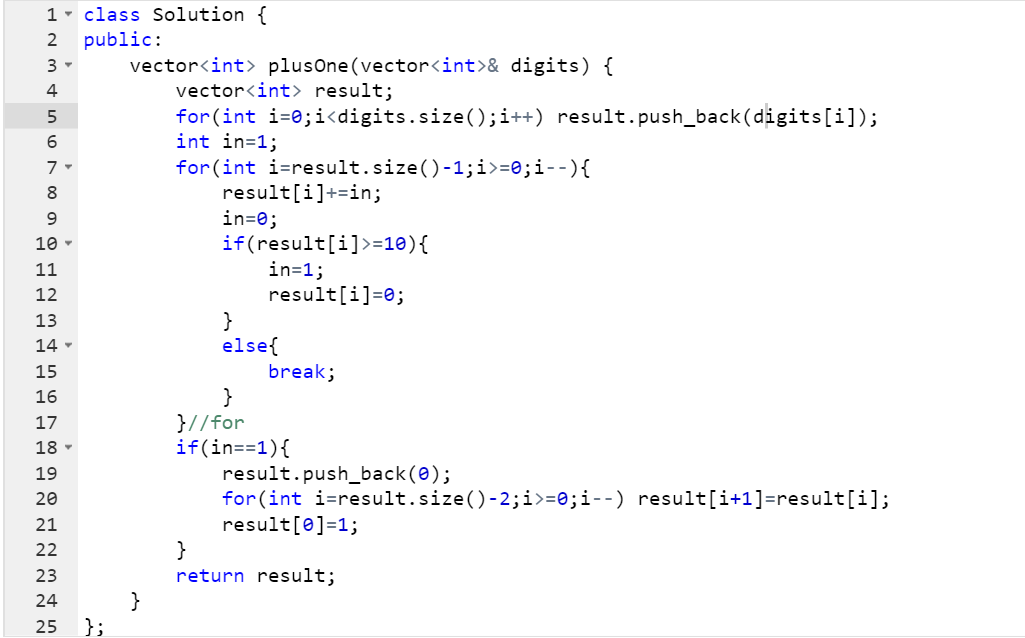
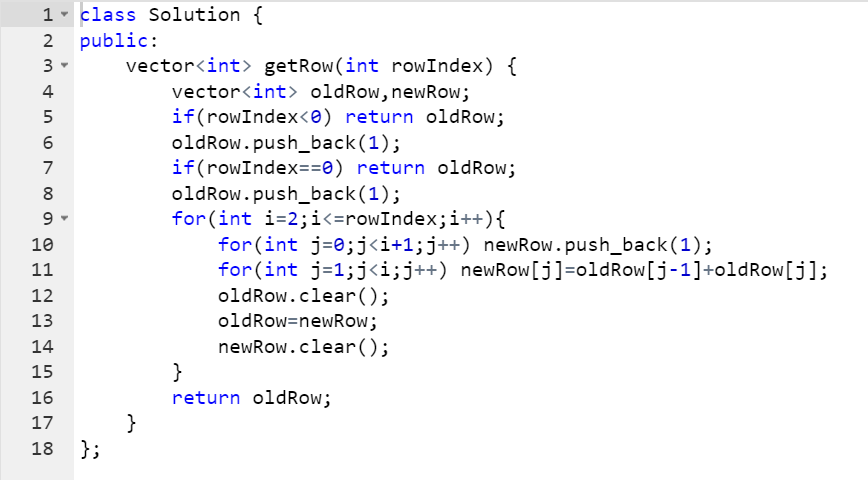
## 1 Problem: Plus One



比较常规的题目，主要难点在于是否存在进位，以及进位以后增加一位数的情况，如999.只需要从大到小遍历数组，加一即可，如果最后进位，需要在开始重新加一位。

需要注意的测试用例：空，999

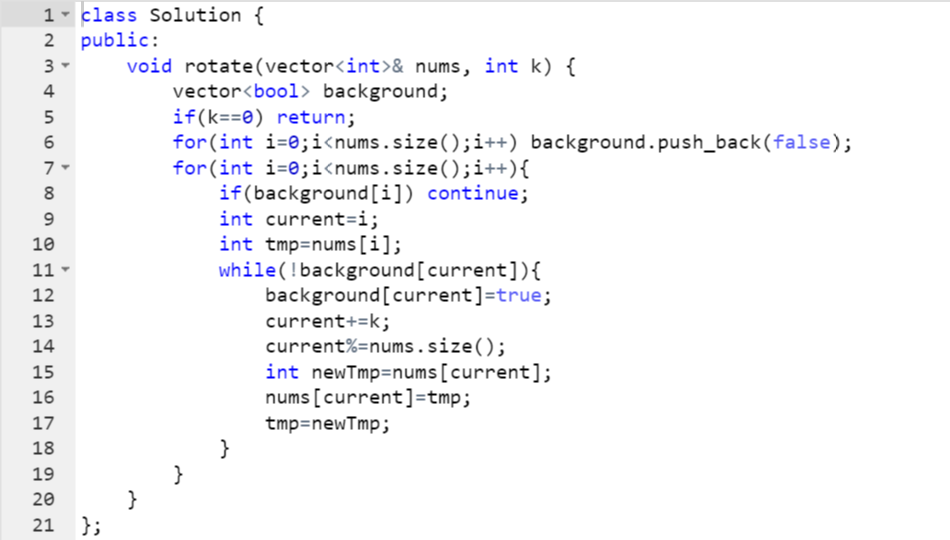
## 2 Problem: Pascal's Triangle II



和PASCAL'S TRIANGLE I 一样，只要和用递推公式就很容易求解。N[I][J]=N[I-1][J]+N[I-1][J-1];

需要注意的测试用例：<0,0,1

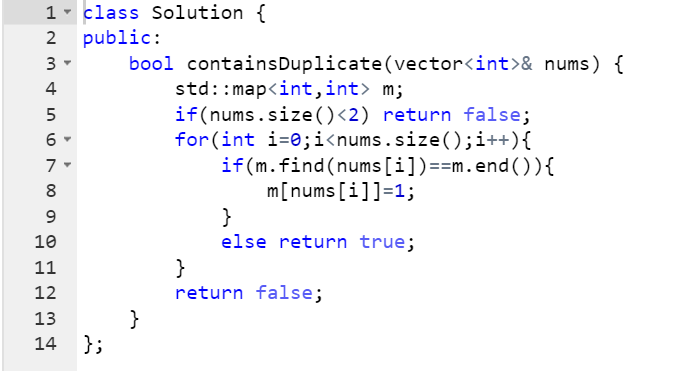
## 3 Problem: Rotate Array



为了节省空间，上面的代码移动的时候是一次将一个数往前移动K，记录需要更改格子的值，再往前移动K，如果超过长度取摩。同时用一个数组记录当前数是否已经移动过，如果是，从头开始让没有移动过的重复上面的过程即可。

需要注意的测试用例：空，<"1"，2>

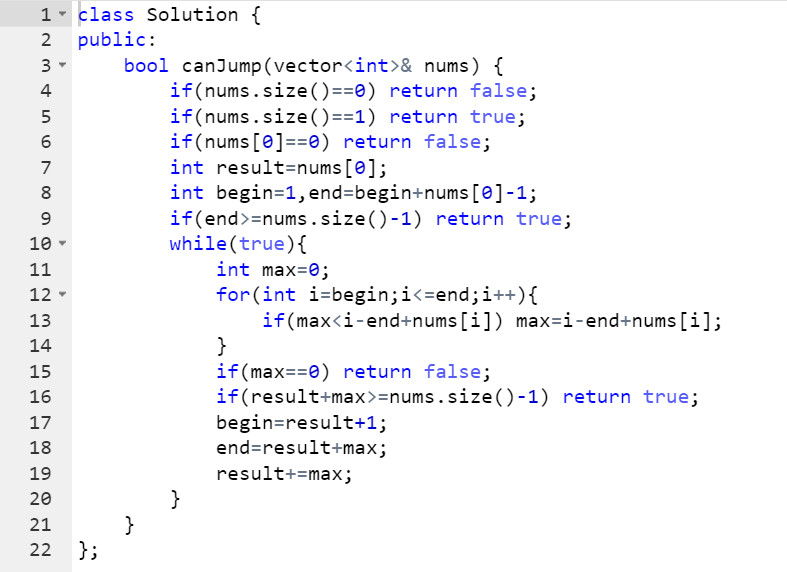
## 4 Problem: Contains Duplicates



很常规的哈希表题目，遍历数组，哈希表没有就加进去，有返回FALSE,遍历完成返回TRUE。

特殊测试用例，空，<1>

## 5 Problem: Jump Game



尝试过很多思路，用一般的搜索肯定超时，用回溯法也超时。所以思路如下，从第一个数开始，计算它当前能到达的范围而不是一个值，在这个范围里面计算从这个范围能到达的最大的下一个范围，以此类推，如果这个范围已经超过了数组大小，则返回TRUE，一旦这个新的范围长度为0，返回FALSE。

需要注意的测试用例：空，<1>，<0>，<0,0,0,0,0>

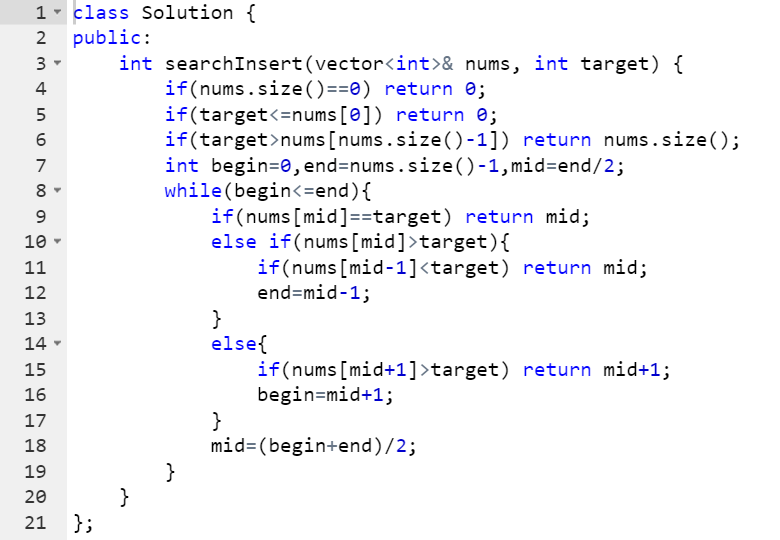
## 6 Problem: Two Sum



用常规办法没有能够求解，于是参考网上以后得知需要用哈希表。用哈希表将让时间复杂度从一般搜索的N平方变成N，实际上是用空间换时间。具体做法是现将所有数放进哈希表然后遍历数组，求第二个数，如果在哈希表里面发现了第二个数，从当前位置开始搜索第二个数（如果当前数在前面则前面搜索的时候已经返回），因为只有一个解，也就是说找到解以后只会执行一次查找便非常快（如果有多个解并且求所有解，那么这个方法可能会比较慢）。因为这个题目要求返回索引，因此不能排序。

需要注意的测试用例：<0，0>,0,<-1，1，-1>,0

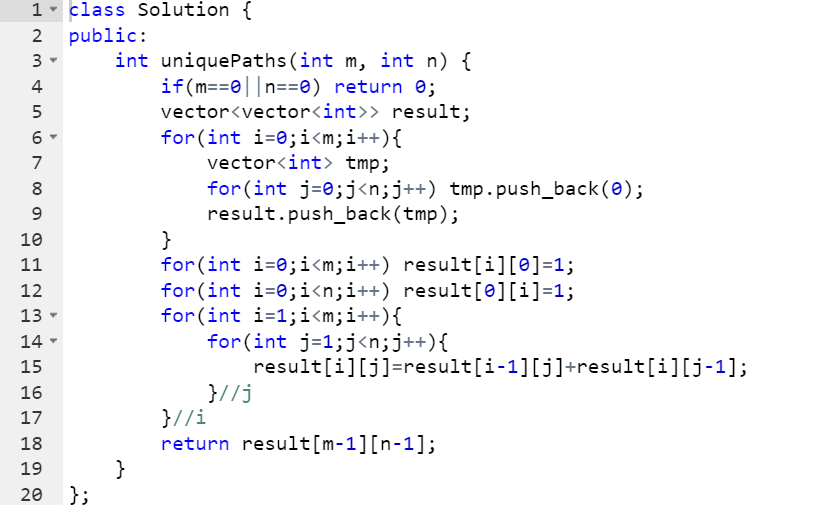
## 7 Problem: Search Insert Position



用二分查找的典型题目，如果用线性搜索非常简单但时间慢。首先看是否为空，为空直接放第一个，否则看时候小于第一个或者大于最后一个。如果都不是，则开始二分查找，查找时注意如果当前的数比TARGET小，TARGET应该在当前数右边，此时要看当前数的后一个数是否比TARGET大，如果是就放这里，如果不是才将头指针往前移动。因为我们开始就测试了收尾的情况，所以不存在当前数的后面一个数数组越界的情况，这种方法在排好序的二分搜索中都应该是适用的。

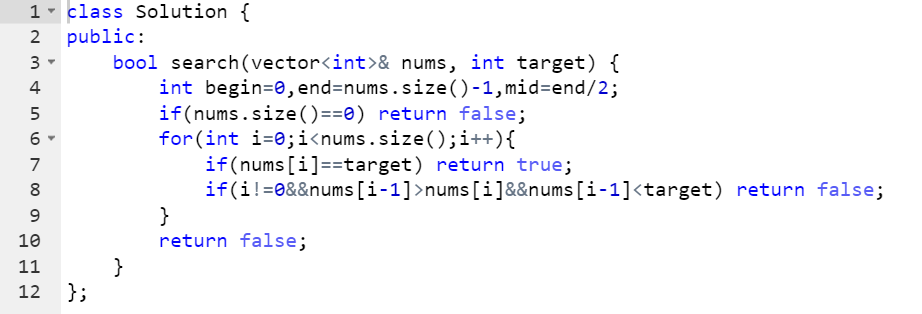
需要注意的测试用例：空，<"0",-1>,<"0",1>

## 8 Problem: Unique Paths



和上周的MINIMUM PATH SUM如出一辙，都是加法原理，或者简单的动态规划。可以参考MINIMUM PATH SUM。

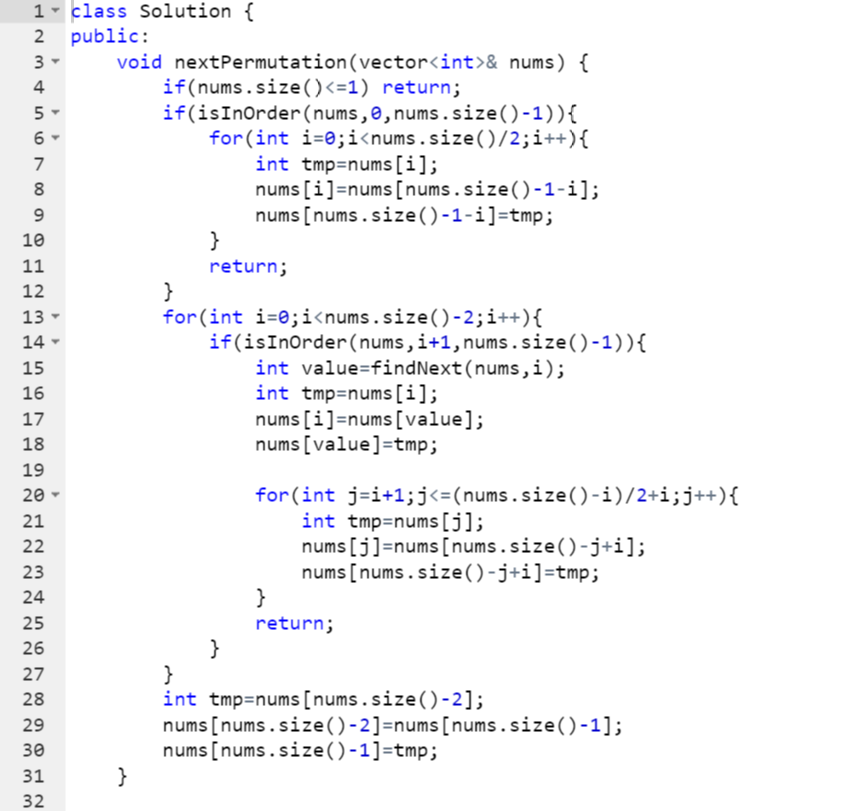
## 9 Problem: Search in Rotated Sorted Array II

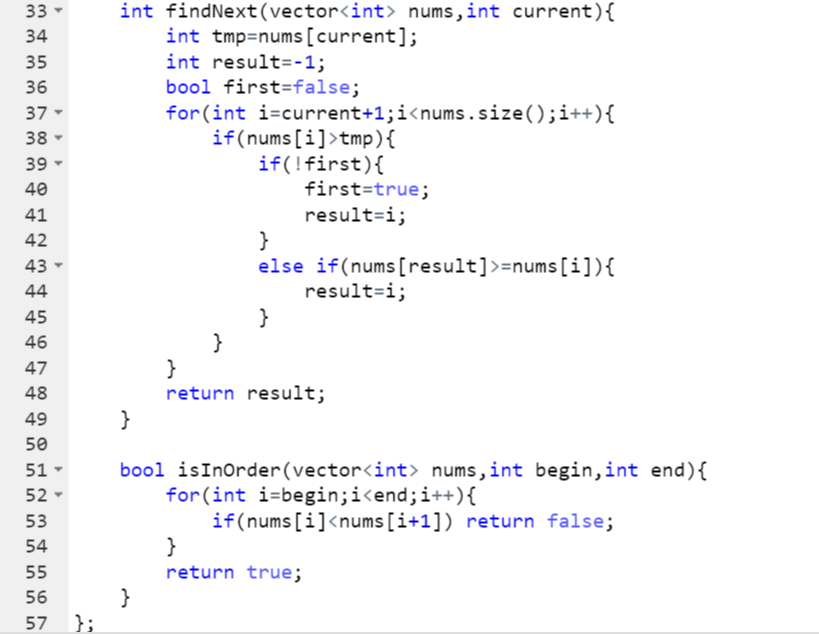


这个题目因为有了重复的数，情况变得很复杂。我用一般的线性查找虽然过了但是查找的时间比较慢，其实不是很理想（毕竟如果用这个方法这个题目就失去了意义）。在网上看了解法，其实和一般的ROTATED SORTED ARRAY差不多，但是因为可以相同，所以通过判断首尾大小已经不一定能缩小查找范围，如果两个数（首尾）相等，需要移动首或者尾（往中间）到不相等为止进行判断。

需要注意的测试用例：空，<1，2，3，"1">,<2，3，1，"2">,<3，3，1，2，3，"2">

## 10 Problem: Next Permutation





首先定义一种概念叫顺序，也就是INORDER这个函数的意义，定义从BEGIN到END（BEGIN<END）的数按照从大到小排列（可以相等），然后我们发现如果一个子序列满足顺序，则这个子序列一定为被重排（最大的被更换），更换过程如下：将BEGIN前一个数和BEGIN到END里面比BEGIN前一个大（不能相等，否则出现重复情况）的最小的一个交换（这种有多个这种最小，取INDEX最大的，见最后一个测试用例），再将BEGIN到END顺序倒过来（321-123）。需要注意的是：1BEGIN可能没有前一个，也就是说出现了类似321的情况，则在开始直接判断并在开始将顺序直接倒过来；2如果不满足顺序，但是BEGIN+1=END，则直接交换返回，类似312-321。

需要注意的测试用例：空，<1>，<3,2,1>,<3,1,2>,<2,3,2,2,1>,<1,2,2,2,1>