题目：

给定一个无序整型数组arr，找到数组中未出现的最小正整数。要求**时间复杂度为O(N)空间复杂度为O(1)**。

例如：

arr=[-1,2,3,4]。返回1。

arr=[1,2,3,4]。返回5。

=========================================================

分析：

这道题要理解最小正整数的意思，最小的正整数就是1，所以考察的方法就是在数组中找1，然后找2，依次找下去...。直到第一个没有找到的数，这个数就是未出现的最小的正整数。但是这样的时间复杂度很大，达到了O(n2)。

先看一个时空复杂度均为O(n)的方案，思路如下：

|  |
| --- |
| 新建一个和原数组大小一致的新数组，通过遍历原数组将其中每个元素e(忽略掉小于1或大于数组长度的元素)填充到新数组中[e-1]位置上。之后遍历新数组就可找到目标，这个遍历可能会遇到两种情况，**一般情况下**，上一步的操作总有被忽略的元素，每忽略一个数，新数组中就会少填充一个正整数，如{-1,1,2,5,6}>>{1,2,0,0,5}，这种情况要找的数就是**第一个值为0的元素的下标+1**；**极端情况下**，上一步的操作没有被忽略的元素，如{3,2,1,5,4}>>{1,2,3,4,5}，这种情况要找的数就是length+1；  为什么开辟的新数组大小要和原数组大小一致？这是为了确保在极端情况下能够容纳下由原数组中元素组成的从1开始的最长连续整数序列。  为什么要忽略掉大于数组长度的元素？这是因为如果存在这样的数X，剩下的小于length个元素不可能组成1~length的连续整数序列，则X更不可能在连续序列中，就没必要维护它了。 |

相应代码的实现如下：

|  |
| --- |
| @org.junit.Test  **public** **void** test() {  System.***out***.println("结果：" + func1(**new** **int**[] { -1, 5, 1, 6, 2 }));  System.***out***.println("结果：" + func1(**new** **int**[] { 3, 2, 1, 5, 4 }));  }/\* out:  \* [-1, 5, 1, 6, 2] >>  \* [1, 2, 0, 0, 5]  \* 结果：3  \* [3, 2, 1, 5, 4] >>  \* [1, 2, 3, 4, 5]  \* 结果：6  \*/  **public** **int** func1(**int**[] arr) {  **int**[] newArr = **new** **int**[arr.length];  **for** (**int** e : arr) {  **if** (e < 1 || e > arr.length) {  **continue**;  }  newArr[e - 1] = e;  }  System.***out***.println(Arrays.*toString*(arr) + " >> ");  System.***out***.println(Arrays.*toString*(newArr));  **for** (**int** i = 0; i < newArr.length; i++) {  **if** (newArr[i] == 0) {  **return** i + 1;  }  }  **return** arr.length + 1;  } |

再看改进方案，减小空间复杂度为O(1)，代码如下：

|  |
| --- |
| @org.junit.Test  **public** **void** test2() {  System.***out***.println(funcFinal(**new** **int**[]{-1,5,1,6,2}));  System.***out***.println(funcFinal(**new** **int**[]{3,2,1,5,4}));  }/\* out:  \* 原数组:[-1, 5, 1, 6, 2]  \* 处理后:[1, 2, 1, 6, 2]  \* 结果：3  \* 原数组:[3, 2, 1, 5, 4]  \* 处理后:[1, 2, 3, 4, 5]  \* 结果：6  \*/    **public** **int** funcFinal(**int**[] arr) {  System.***out***.println("原数组:" + Arrays.*toString*(arr));  /\*  \* right是一个边界值，表示用数组中元素组成的从1开始的连续整数序列中可能的最大值（初始等于数组长度）。  \* 处理数组过程中如果遇到比right大的数，就表示该数不合法，应该被丢掉（代码中还处理了其它表示数不合法的情况）。  \* >> 随着数组元素被处理，每遇到一个不合法的元素，就应将right减1。  \*/  **int** right = arr.length;  /\*  \* 索引left（初始为0），left将数组分成两部分。  \* [0,left)是处理完成的部分，其中每个元素都满足a[i]=i+1;  \* [left,right]是待处理部分。  \* >> 随着数组元素被处理，left会逐渐向右移动。  \*/  **int** left = 0;    **while** (left + 1 <= right) { // 正在处理的元素的值(left+1) <= 边界值  // 分支1、arr[left]在理想的位置  // 则处理完成部分长度加1，然后继续处理未完成部分的下一个待处理元素  **if** (arr[left] == left + 1) {  left++;  }  // 分支2、arr[left]是不合法的数据  // 则先将right减1，然后丢掉不合法的数并将待处理部分最后一个元素填充到left位置继续处理  **else** **if** (arr[left] < left + 1 || arr[left] > right) {  right--;  arr[left] = arr[right];  }  // 分支3、arr[left]合法，但是没有在理想的位置上  // 则需要交换arr[left]与其理想位置上元素，然后继续处理交换后left位置处的元素  // 求理想位置p的索引：p+1 = arr[left] >> p = arr[left]-1  **else** {  // 如果要交换的两个元素相同，也算当前处理的元素arr[left]不合法，进行与分支2一样的处理  **if**(arr[left] == arr[arr[left] - 1]) {  right--;  arr[left] = arr[right];  } **else** {  swap(arr, left, arr[left] - 1);  }  }  }  System.***out***.println("处理后:" + Arrays.*toString*(arr));  **return** left + 1;  }  **private** **void** swap(**int**[] a, **int** i, **int** j) {  **int** temp = a[i];  a[i] = a[j];  a[j] = temp;  } |