数组 `A` 是 `[0, 1, ..., N - 1]` 的一种排列，`N` 是数组 `A` 的长度。全局倒置指的是 `i,j` 满足 `0 <= i < j < N` 并且 `A[i] > A[j]` ，局部倒置指的是 `i` 满足 `0 <= i < N` 并且 `A[i] > A[i+1]` 。

当数组 `A` 中全局倒置的数量等于局部倒置的数量时，返回 `true` 。

示例 1:

```
输入: A = [1,0,2]
输出: true
解释: 有 1 个全局倒置，和 1 个局部倒置。
```

示例 2:

```
输入: A = [1,2,0]
输出: false
解释: 有 2 个全局倒置，和 1 个局部倒置。
```

注意:

- `A` 是 `[0, 1, ..., A.length - 1]` 的一种排列
- `A` 的长度在 `[1, 5000]`之间
- 这个问题的时间限制已经减少了。

- 题目难度: 中等
- 通过次数: 172
- 提交次数: 491
- 贡献者: LeetCode
- 相关话题

数组数学

这道题给了一个长度为n的数组，里面是0到n-1数字的任意排序。又定义了两种倒置方法，全局倒置和局部倒置。其中全局倒置说的是坐标小的值大，局部倒置说的是相邻的两个数，坐标小的值大。那么我们可以发现，其实局部倒置是全局倒置的一种特殊情况，即局部倒置一定是全局倒置，而全局倒置不一定是局部倒置，这是解这道题的关键点。题目让我们判断该数组的全局倒置和局部倒置的个数是否相同，那么我们想，什么情况下会不相同？如果所有的倒置都是局部倒置，那么由于局部倒置一定是全局倒置，则二者个数一定相等。如果出现某个全局倒置不是局部倒置的情况，那么二者的个数一定不会相等。所以问题的焦点就变成了是否能找出不是局部倒置的全局倒置。所以为了和局部倒置区别开来，我们不能比较相邻的两个，而是至少要隔一个来比较。我们可以从后往前遍历数组，遍历到第三个数字停

止，然后维护一个 [i, n-1] 范围内的最小值，每次和 A[i - 2] 比较，如果小于 A[i - 2]，说明这是个全局的倒置，并且不是局部倒置，那么我们直接返回false即可，参见代码如下：

解法一：

```
class Solution {
public:
    bool isIdealPermutation(vector<int>& A) {
        int n = A.size(), mn = INT_MAX;
        for (int i = n - 1; i >= 2; --i) {
            mn = min(mn, A[i]);
            if (A[i - 2] > mn) return false;
        }
        return true;
    }
};
```

同理，我们可以反其道行之，我们可以从前往后遍历数组，遍历到倒数第三个数字停止，然后维护一个 [0, i] 范围内的最大值，每次和 A[i + 2] 比较，如果大于 A[i + 2]，说明这是个全局的倒置，并且不是局部倒置，那么我们直接返回false即可，参见代码如下：

解法二：

```
class Solution {
public:
    bool isIdealPermutation(vector<int>& A) {
        int n = A.size(), mx = INT_MIN;
        for (int i = 0; i < n - 2; ++i) {
            mx = max(mx, A[i]);
            if (A[i + 2] < mx) return false;
        }
        return true;
    }
};
```

其实这道题最叼最炫酷的解法是下面这种解法，最早由grandyang大神的帖子中提出，嗯？？grandyang大神？？没错，就是博主本人啦，哈哈，秀不秀？！？这个解法也是博主

脑子灵光一现而想出的，由于原数组正常的顺序应该是 [0, 1, 2, 3, 4...] 这种，即数字和其下标是相同的，所以如果我们发现乱序数组中某个数字和其坐标差的绝对值大于1的话，那么一定是有非局部倒置的全局倒置的存在。猛然这么一说，可能你会问为啥啊？因为0到n-1中每个数字都是在数组中存在的，如果当前数字 A[i] 比起坐标 i 大1的话，比如 A[i] = 3, i = 1 的时候，那么数组的第二个数字是3了，前三个数字suppose是 0，1，2 的，但是由于第二个数字是3了，那么一定会有一个小于3的数字被挤出前三个数字，这个小于3的数字最多出现在下标为3的位置上，那么由于数字3出现在了下标为1的位置上，所以non-local的全局倒置就出现了。同理，如果当前数字 A[i] 比其坐标 i 小1的话，比如 A[i] = 1, i = 3 的时候，那么就是后 n-i 个数字中有一个大于 A[i] 的数字被挤到了前面去了，而且其跟 A[i] 的距离最小为2，所以non-local的全局倒置就出现了，大声告诉博主，这个解法精彩不精彩？

解法三：

```
class Solution {
public:
    bool isIdealPermutation(vector<int>& A) {
        for (int i = 0; i < A.size(); ++i) {
            if (abs(A[i] - i) > 1) return false;
        }
        return true;
    }
};
```

总结，想过博主最后一种方法，可以自己想的是数组自己进行操作，而未想到坐标层面，可以记住的东西有INT_MAX,INT_MIN为有符号的int型的最大最小值