codility
WE TEST CODERS

Chapter 8

Leader

Let us consider a sequence $a_0, a_1, \ldots, a_{n-1}$. The leader of this sequence is the element whose value occurs more than $\frac{n}{2}$ times.

a_0	a_1	a_2	a_3	a_4	a_5	a_6
6	8	4	6	8	6	6
0	1	2	3	4	5	6

In the picture the leader is highlighted in gray. Notice that the sequence can have at most one leader. If there were two leaders then their total occurrences would be more than $2 \cdot \frac{n}{2} = n$, but we only have n elements.

The leader may be found in many ways. We describe some methods here, starting with trivial, slow ideas and ending with very creative, fast algorithms. The task is to find the value of the leader of the sequence $a_0, a_1, \ldots, a_{n-1}$, such that $0 \leqslant a_i \leqslant 10^9$. If there is no leader, the result should be -1.

8.1. Solution with $O(n^2)$ time complexity

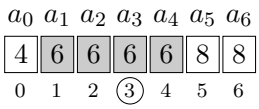
We count the occurrences of every element:

8.1: Leader — $O(n^2)$.

```
def slowLeader(A):
    n = len(A)
    leader = -1
    for k in xrange(n):
        candidate = A[k]
        count = 0
        for i in xrange(n):
            if (A[i] == candidate):
                count += 1
        if (count > n // 2):
            leader = candidate
    return leader
```

8.2. Solution with $O(n \log n)$ time complexity

If the sequence is presented in non-decreasing order, then identical values are adjacent to each other.



Having sorted the sequence, we can easily count slices of the same values and find the leader in a smarter way. Notice that if the leader occurs somewhere in our sequence, then it must occur at index $\frac{n}{2}$ (the central element). This is because, given that the leader occurs in more than half the total values in the sequence, there are more leader values than will fit on either side of the central element in the sequence.

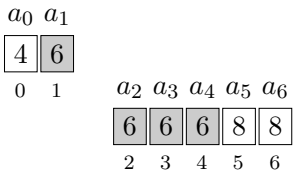
8.2: Leader — $O(n \log n)$.

```
def fastLeader(A):
    n = len(A)
    leader = -1
    A.sort()
    candidate = A[n // 2]
    count = 0
    for i in xrange(n):
        if (A[i] == candidate):
            count += 1
    if (count > n // 2):
        leader = candidate
    return leader
```

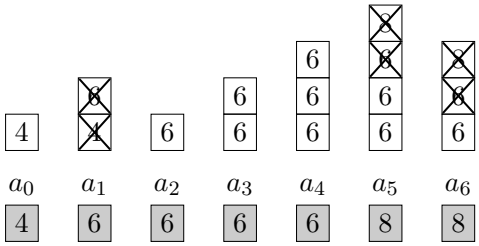
The time complexity of the above algorithm is $O(n \log n)$ due to the sorting time.

8.3. Solution with $O(n)$ time complexity

Notice that if the sequence $a_0, a_1, \ldots, a_{n-1}$ contains a leader, then after removing a pair of elements of different values, the remaining sequence still has the same leader. Indeed, if we remove two different elements then only one of them could be the leader. The leader in the new sequence occurs more than $\frac{n}{2} - 1 = \frac{n-2}{2}$ times. Consequently, it is still the leader of the new sequence of $n - 2$ elements.



Removing pairs of different elements is not trivial. Let's create an empty stack onto which we will be pushing consecutive elements. After each such operation we check whether the two elements at the top of the stack are different. If they are, we remove them from the stack. This is equivalent to removing a pair of different elements from the sequence (in the picture below, different elements being removed are highlighted in gray).



In fact, we don't need to remember all the elements from the stack, because all the values below the top are always equal. It is sufficient to remember only the values of elements and the size of the stack.

8.3: Leader — $O(n)$.

```
1  def goldenLeader(A):
2      n = len(A)
3      size = 0
4      for k in xrange(n):
5          if (size == 0):
6              size += 1
7              value = A[k]
8          else:
9              if (value != A[k]):
10                 size -= 1
11             else:
12                 size += 1
13     candidate = -1
14     if (size > 0):
15         candidate = value
16     leader = -1
17     count = 0
18     for k in xrange(n):
19         if (A[k] == candidate):
20             count += 1
21     if (count > n // 2):
22         leader = candidate
23     return leader
```

At the beginning we notice that if the sequence contains a leader, then after the removal of different elements the leader will not have changed. After removing all pairs of different elements, we end up with a sequence containing all the same values. This value is not necessarily the leader; it is only a candidate for the leader. Finally, we should iterate through all the elements and count the occurrences of the candidate; if it is greater than $\frac{n}{2}$ then we have found the leader; otherwise the sequence does not contain a leader.

The time complexity of this algorithm is $O(n)$ because every element is considered only once. The final counting of occurrences of the candidate value also works in $O(n)$ time.

Every lesson will provide you with programming tasks at `http://codility.com/programmers`.