Max Consecutive Ones II

Given a binary array, find the maximum number of consecutive 1s in this array if you can flip at most one 0.

Example 1:

```
Input: [1,0,1,1,0]
Output: 4
Explanation: Flip the first zero will get the the maximum number of consecutive 1s.
    After flipping, the maximum number of consecutive 1s is 4.
```

Note:

- The input array will only contain `0` and `1`.
- The length of input array is a positive integer and will not exceed 10,000

Follow up:

What if the input numbers come in one by one as an **infinite stream**? In other words, you can't store all numbers coming from the stream as it's too large to hold in memory. Could you solve it efficiently?

Solution 1

The idea is to keep a window `[l, h]` that contains at most `k` zero

The following solution does not handle follow-up, because `nums[l]` will need to access previous input stream
`Time: O(n) Space: O(1)`

```java
    public int findMaxConsecutiveOnes(int[] nums) {
        int max = 0, zero = 0, k = 1; // flip at most k zero
        for (int l = 0, h = 0; h < nums.length; h++) {
            if (nums[h] == 0)
                zero++;
            while (zero > k)
                if (nums[l++] == 0)
                    zero--;
            max = Math.max(max, h - l + 1);
        }
        return max;
    }
```

Now let's deal with follow-up, we need to store up to `k` indexes of zero within the window `[l, h]` so that we know where to move `l` next when the window contains more than `k` zero. If the input stream is infinite, then the output could be extremely large because there could be super long consecutive ones. In that case we can use `BigInteger` for all indexes. For simplicity, here we will use `int`
`Time: O(n) Space: O(k)`

```java
    public int findMaxConsecutiveOnes(int[] nums) {
        int max = 0, k = 1; // flip at most k zero
        Queue<Integer> zeroIndex = new LinkedList<>();
        for (int l = 0, h = 0; h < nums.length; h++) {
            if (nums[h] == 0)
                zeroIndex.offer(h);
            if (zeroIndex.size() > k)
                l = zeroIndex.poll() + 1;
            max = Math.max(max, h - l + 1);
        }
        return max;
    }
```

Note that setting `k = 0` will give a solution to the earlier version Max Consecutive Ones

For `k = 1` we can apply the same idea to simplify the solution. Here `q` stores the index of zero within the window `[l, h]` so its role is similar to `Queue` in the above solution

```java
public int findMaxConsecutiveOnes(int[] nums) {
    int max = 0, q = -1;
    for (int l = 0, h = 0; h < nums.length; h++) {
        if (nums[h] == 0) {
            l = q + 1;
            q = h;
        }
        max = Math.max(max, h - l + 1);
    }
    return max;
}
```

written by yuxiangmusic original link here

Solution 2

The solution for the previous problem "Max Consecutive Ones" is as simple as this

```
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        nums.append(0)
        lo = 0
        ret = 0
        for hi,n in enumerate(nums):
            if n==0:
                ret = max(ret, hi-lo)
                lo = hi+1
        return ret
```

Based on the above solution, for "Max Consecutive Ones II", we simply keep track of 2 low pointers:

```
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        nums.append(0)
        lo1, lo2 = 0, 0
        ret = 0
        for hi,n in enumerate(nums):
            if n==0:
                ret = max(ret, hi-lo1)
                lo1, lo2 = lo2, hi+1
        return ret
```

A code golfing version just for fun:

```
class Solution(object):
    def findMaxConsecutiveOnes(self, nums):
        return reduce(lambda l,e:(max(l[0],e[0]-l[1]),[l[2],l[1]][e[1]>0],[e[0]+1
,l[2]][e[1]>0]),enumerate(nums+[0]),[0,0,0])[0]
```

written by o_sharp original link here

Solution 3

```
public int findMaxConsecutiveOnes(int[] nums) {
     int maxConsecutive = 0, zeroLeft = 0, zeroRight = 0;
     for (int i=0;i<nums.length;i++) {
         zeroRight++;
         if (nums[i] == 0) {
             zeroLeft = zeroRight;
             zeroRight = 0;
         }
         maxConsecutive = Math.max(maxConsecutive, zeroLeft+zeroRight);
     }
     return maxConsecutive;
}
```

written by compton_scatter original link here