摩尔投票算法及其变体

Ynjxsjmh

<2019-03-16 Sat>

目录

1 波义尔摩尔投票算法（Boyer-Moore majority vote algorithm）

Boyer-Moore majority vote algorithm 是用来查找一个长度为 n 的序列中出现次数大于 $\lfloor \frac{n}{2} \rfloor$ 中的数。该算法的时间复杂度是 O(n)，空间复杂度是 O(1)。该算法有一个前提是必须存在这样一个数，否则算法不起作用。

```
int majorityElement(vector<int>& nums) {
    int candidate = 0;
    int count = 0;
    int n = nums.size();
    for(int i = 0; i < n; i++) {
        if (count == 0) {
            candidate = nums[i];
            count = 1;
```

```
        } else if (candidate == nums[i]) {
            count++;
        } else {
            count--;
        }
    }
    return candidate;
}
```

2 要求寻找出现次数大于 $\lfloor \frac{n}{3} \rfloor$ 的数

当要求寻找出现次数大于 $\lfloor \frac{n}{3} \rfloor$ 的数时，满足该条件的数最多有两个。

$$n > (\frac{1}{3}n + 1) * 2 \; n >= 6$$

```
vector<int> majorityElement(vector<int>& nums) {
    int count1 = 0, count2 = 0;
    int candidate1 = 0, candidate2 = 0;
    int n = nums.size();

    for (int i = 0; i < n; i++) {
        if (nums[i] == candidate1) {
            count1++;
        } else if (nums[i] == candidate2) {
            count2++;
        } else if (count1 == 0) {
            count1 = 1;
            candidate1 = nums[i];
        } else if (count2 == 0) {
            count2 = 1;
            candidate2 = nums[i];
```

```
        } else {
            count1--;
            count2--;
        }
    }

    vector<int> result;

    if (count(nums.begin(), nums.end(), candidate1) > n/3) {
        result.push_back(candidate1);
    }

    if (candidate1 == candidate2) {
        return result;
    }

    if (count(nums.begin(), nums.end(), candidate2) > n/3) {
        result.push_back(candidate2);
    }

    return result;
}
```

3 $\lfloor \frac{n}{4} \rfloor$ $\lfloor \frac{n}{k} \rfloor$

Find all elements that appear more than n/4 times in linear time
Find if there is an element repeating itself n/k times