2022 年 4 月 1 日

BOYER MOORE 多数投票算法

尼基尔·库马尔·辛格

弗里希克

期间

9分钟

类别

竞争性编程

搜索

标签

C++

分享

学到更多

TOPCODER 蓬勃发展

Boyer-Moore 投票法是最常用的优化算法之一，用于确定出现次数超过 N/2 的元素中的多数元素。这对于查找多数元素非常有效，这需要对提供的项目进行两次遍历，并且是 O(N) 时间和 O(1) 空间复杂度。

这种方法经常在与谷歌、Facebook、**微软**、Salesforce **和亚马逊**等公司的采访中被问到。

寻找多数元素问题

在这个任务中，我们必须在提供的数组中发现频率超过数组长度一半的数字，即 N/2 次。数组的长度用 N 表示。

示例 1：
输入：{4,7,4,4,3,4,6,4}
输出：4
解释：数组的长度为 8，2 的频率为 5 > 8/2

示例 2：
输入：{4,7,0,5,3,4,6,4}
输出：-1
解释：没有多数元素

方法一：蛮力

使用嵌套的 for 循环来解决多数元素问题，计算每个元素，如果每个元素的 count>n/2。但是，这种方法效率低下，因为它需要 O(n2) 时间。

代码：

```
int findMajorityElement(vector < int > & nums) {

 int n = nums.size(), count;
 for (int i = 0; n && i <= n / 2; i++) {

  count = 1;
  for (int j = i + 1; j < n; j++)
   if (nums[j] == nums[i])
    count++;

  if (count > n / 2)
   return nums[i];
 }

 return -1;
}
```

时间复杂度： O(n2)，嵌套循环
空间复杂度： O(1) 只有两个变量

方法#2：地图方法

我们不检查数组中每个数字的频率，而是将每个数字的频率存储在映射中，然后检查是否有任何数字具有所需的频率。

代码：

```
int findMajorityElement(vector < int > & nums) {

 unordered_map < int, int > mp;

 for (auto x: nums)
  mp[x]++;

 for (auto x: mp)
  if (x.second > n / 2)
   return x.first;

 return -1;
}
```

Time complexity: O(n), we make only two iterations through the array.
Space complexity: O(n), sorting frequency of each number.

APPROACH #3: BOYER-MOORE VOTING ALGORITHM

In its most basic form, the algorithm seeks out a majority element if one exists. A majority element is one that appears more than half of the time in the input elements. However, if there is no majority, the algorithm will not recognize this and will continue to output one of the items.

The algorithm is divided into two parts. A first pass identifies an element as a majority, and a second pass confirms that the element identified in the first pass is indeed a majority.

The method will not identify the majority element if it does not exist, and will thus return an arbitrary element.

ALGORITHM:

1. Begin by initializing two variables, num and a counter count, both of which are set to 0.
2. Now we'll begin iterating over the number and for each element x.
3. We first check to see if the count is 0, and then we assign num to x.
4. Then we check to see if the current num is equal to x, if not, we decrease the count by one, else we increment it by one.
5. Reset the counter to zero.
6. Find the frequency of the num variable by looping through the nums array.
7. Now, if the count is larger than N/2, we return num; otherwise we return -1.

ILLUSTRATION:

Input: { 1,7,2,7,8,7,7}

Index 0: num = 1, count =1
Index 1: num = 1, count =0 (7 not equal to 1)
Index 2: num = 2 (as the count is 0 we initalise num to current), count =1
Index 3: num = 2, count =0 (7 not equal to 2)
Index 4: num = 8 (as the count is 0 we initalise num to current), count =1
Index 5: num = 8, count =0 (7 not equal to 8)
Index 6: num = 7 (as the count is 0 we initalise num to current), count =1

Now we can check for the frequency of 7, i.e, 4 which is > 7/2.

Code:

```
int findMajority(vector < int > nums) {

 int n = nums.size();
 int num, count = 0;
 for (auto x: nums) {
   if (count == 0)
     num = x;

   count += (x == num ? 1 : -1);
 }

 count = 0;
 for (auto x: nums)
   if (x == num)
     count++;

 if (count > n / 2)
   return num;

 return -1;
}
```

Time complexity: O(n), we make only two iterations through the array.

Space complexity: O(1), only two variables.

RECOMMENDED FOR YOU