Boyer–Moore majority vote algorithm

The **Boyer-Moore Vote Algorithm** solves the majority vote problem in linear time

$$O(n)$$

and logarithmic space

$$O(\log n)$$

[citation needed]. The majority vote problem is to determine in any given sequence of choices whether there is a choice with more occurrences than half of the total number of choices in the sequence and if so, to determine this choice. Note how this definition contrasts with the mode Mode in which it is not simply the choice with the most occurrences, but the number of occurrences relative to the total length of the sequence.

Mathematically, given a finite sequence (length n) of numbers, the object is to find the majority number defined as the number that appears more than ⌊ n/2 ⌋ times.

Description

The algorithm is carried out in two steps:

1. Eliminate all elements except one.

Iterating through the array of numbers, maintain a current candidate and a counter initialized to 0. With the current element x in iteration, update the counter and (possibly) the candidate:

If the counter is 0, set the current candidate to x and the counter to 1. If the counter is not 0, increment or decrement the counter based on whether x is the current candidate.

2. Determine if the remaining element is a valid majority element.

With the candidate acquired in step 1, iterate through the array of numbers and count its occurrences. Determine if the result is more than half of the sequence's length. If so, the candidate is the majority. Otherwise, the sequence doesn't contain a majority.

Note that the counter can be a maximum of n which requires

$$O(\log n)$$

space. In practice however a constant number of bits should suffice as a 128 bit counter can go upto 2^{128} which is large enough for any practical computation. The time complexity remains

$$O(n)$$

, even considering the amount of time it takes to increment the counter because it can be incremented in constant amortized time.

Implementation in Java

```java
import java.util.*;
public class MajorityVote {
    public int majorityElement(int[] num) {
        int n = num.length;
        int candidate = num[0], counter = 0;
        for (int i : num) {
            if (counter == 0) {
                candidate = i;
                counter = 1;
            } else if (candidate == i) {
                counter++;
            } else {
                counter--;
            }
        }

        counter = 0;
        for (int i : num) {
            if (i == candidate) counter++;
        }
        if (counter < (n + 1) / 2) return -1;
        return candidate;

    }
    public static void main(String[] args) {
        MajorityVote s = new MajorityVote();
        System.out.format("%d\n", s.majorityElement(new int[] {1, 2, 3}));
        System.out.format("%d\n", s.majorityElement(new int[] {2, 2, 3}));
    }
}
```

Implementation in JavaScript and example outputs

```
function majorityVote(nums) {
  var n = nums.length;
  var candidate = nums[0];
  var counter = 0;

  nums.forEach(function(num) {
    if (counter === 0) {
      candidate = num;
      counter = 1;
    } else if (candidate === num) {
      counter++;
    } else {
      counter--;
    }
  });

  // count the number of times the candidate appears
  counter = 0;
  nums.forEach(function(num) {
    if (num === candidate) {
      counter++;
    }
  });
  if (counter < (n + 1) / 2) {
    return -1;
  }
  return candidate;
}

function test(nums) {
  console.log('for', nums, 'result', majorityVote(nums));
}

// equal number of occurrences for all for [1, 2, 3] result -1
test([1,2,3]);

// 2 a clear majority. outputs 2
test([1,2,2,2,2,2,4]);

// here, 2 has the highest number of occurrences (mode) but not the majority
// (aka half the length of the array). outputs -1
test([2,1,1,2,3,3,2,4,4,2,5,5]);
```

External links

- [1]
- [2]