

Boyer-Moore Majority Voting Algorithm

The Boyer-Moore voting algorithm is one of the popular optimal algorithms which is used to find the majority element among the given elements that have more than $N/2$ occurrences.

This algorithm takes 2 traversals over the given elements.

-- works in $O(N)$ time complexity and $O(1)$ space complexity.

This algorithm works on the fact that if an element occurs more than $N/2$ times, it means that the remaining elements other than this would definitely be less than $N/2$.

We will have at max 1 majority element as $n\%2$ can be 0 or 1.

$$\Delta = \text{COUNT}(\text{ME}) - \sum \text{COUNT}(\text{NON-ME})$$

This delta will always be greater than ZERO.

SPECIAL CASE : Single element is majority element in itself.

5

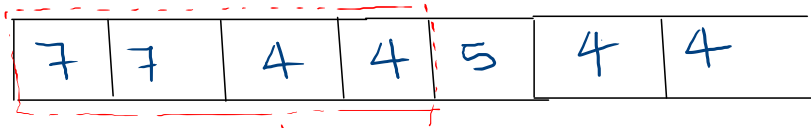
$$n/2 = \frac{1}{2} = 0$$
$$\text{count of ME} > n/2$$
$$1 > 0$$

STRATEGY OF ALGORITHM TO FIND MAJORITY CANDIDATE ELEMENT:

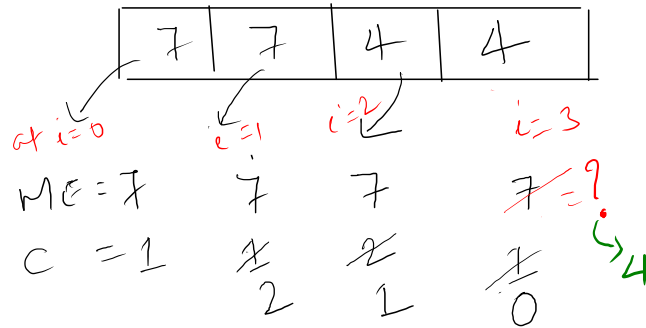
First, choose a **candidate** from the given set of elements and check the next element if it is the same as the candidate element, increase the **vote_count**. Otherwise, decrease the **vote_count** if votes become 0, select another new element as the new **candidate**.

In other words we can infer that we need to **Gather 2 distinct elements and cancel their votes. and Repeat the process for remaining element.**

NOTE: Algorithm can be applied on parts of array independently. See below with the help of example what do we mean by applying on parts independently ?



apply the voting algorithm on this selected part



If we assume that only 0 till current_index part of array exists and assume that rest part does not exist then understanding the algorithm is easy

so, till 0th index: ME is 7, as it is the single element in 1-sized array.

till 1th index : ME is 7, as its vote count is 2 in 2-sized array.

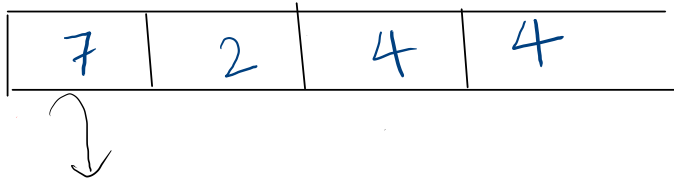
till 2nd index: ME is 7, as the vote diff = $\text{COUNT}(\text{ME}) - \text{SUM_OF_COUNT}(\text{NON_ME}) = 1$

till 3rd index : ME is indetrminate as vote diff = $\text{COUNT}(\text{ME}) - \text{SUM_OF_COUNT}(\text{NON_ME}) = 0$

So, till $i=3$, this is very much clear that 7 is not the majority element as its vote count is cancelled out by the presence of some different element whose value is not equal to 7. Moreover, the values which cancel down the vote of 7 to ZERO also cannot be the majority element till $i=3$ because of the same voting count reason.

So, now we need to pick the new ME and check the candidature in remaining part of array independently.

Question : post finding the majority candidate element why do we need to verify it ?



For this array if we apply the algorithm to find the majority candidate element it will give us 4 as answer. Now if we don't verify 4 then the final answer is wrong.

Implementation strategy to get the majority candidate element:

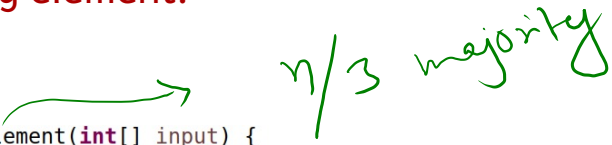
We will have at max 1 majority element as $n\%2$ can be 0 or 1. So we will Gather 2 distinct elements and cancel their votes. and repeat the process for remaining elements.

Boyer-Moore "n/k" Majority Voting Algorithm

We will have at max $(k-1)$ majority element as $n\%k$ can be any one of 0 to $k-1$.
For example: for "n/3 majority" we will have 0 or 1 or 2 as majority elements because $n\%3$ can be 0 or 1 or 2.

ALGO STRATEGY TO GET THE MAJORITY CANDIDATE ELEMENT:

1. Gather K distinct elements and cancel their votes.
2. Repeat the process for remaining element.



```
private List<Integer> getNby3majorityCandidateElement(int[] input) {  
    int majCandidate1 = 0;  
    int majVoteCount1 = 0;  
    int majCandidate2 = 0;  
    int majVoteCount2 = 0;  
    for (int i = 0; i < input.length; i++) {  
        // if-else-if for majVoteCount1 should be together, otherwise majCandidate2 and  
        // majCandidate1 may have same value.  
        if (majVoteCount1 == 0) {  
            majCandidate1 = input[i];  
            majVoteCount1 = 1;  
        } else if (majCandidate1 == input[i]) {  
            majVoteCount1++;  
        } else if (majVoteCount2 == 0) {  
            majCandidate2 = input[i];  
            majVoteCount2 = 1;  
        } else if (majCandidate2 == input[i]) {  
            majVoteCount2++;  
        } else {  
            // this is the place where all the three majorityCandidate1,  
            // majorityCandidate2 and current_element are distinct, so we need to  
            // cancel their votes  
            majVoteCount1--;  
            majVoteCount2--;  
        }  
    }  
}
```