BoyerMoore-Multipass

CSE-595 Fall 23

```
Majority-BoyerMoore-Multipass( A[1 . . . n] )

    Majority-Multipass( A[1 . . . n], −1 )
```

```
Majority-Multipass( A[1 . . . n], tiebreaker )

    Create a dynamic array B ← [ ]

    for i ← 1 to n − 1 increment 2 do
        if A[i] = A[i + 1] then
            B.Add( A[i] )

    isAOdd → (n mod 2 = 1)

    if isAOdd then
        tiebreaker ← A[n]

    if B is empty and !isAOdd then                                          ←#1
        return tiebreaker

    c ← Majority-Multipass( B, tiebreaker )

    if c = −1 then
        return −1

    count ← CountOccurrences( A[1 . . . n], c )

    if count > floor(n/2) or ( !isAOdd and count = floor(n/2) and c = tiebreaker ) then   ←#2
        return c

    return −1
```

Algorithm explanation in simple words:

- The core idea is to keep breaking down Array A recursively until elements cancel each other out. All the child methods return their majority element to their parent methods; the final majority element is returned from the original caller.

- The first step is to create array B out of array A, as shown in the pseudocode above. Now that B is created, check if A is of odd length. If A is of odd length, the last element will obviously be the new tiebreaker. On the other hand, if A was of even length, we would have continued with the same tiebreaker. (The initial tiebreaker is -1).

- Now, suppose that when you created B, it turned out to be empty, i.e., every subsequent pair of elements in A were different and got canceled out. Now, in this case, if A was of even length, then we are sure to have processed all the elements in A, and therefore we can return the same tiebreaker we got from parent back to the parent function for further processing.
  On the other hand, if A was of odd length, we couldn't just return the tiebreaker because this is a new tiebreaker, and we don't want to miss checking if this is the majority or not by executing the countOccurrences method on it.

- Now, when we calculate the countOccurrences value for the element returned from the child method (array B), we need to verify if that element is the majority for my array too (array A). One way to check this is simple: if that element occurs more than floor(n/2) times in my array A.
  There is one more check: in cases where A is of even length, even if the value of count is equal to floor(n/2), the element can still be a majority. Remember, when A was of even length, tiebreaker was not extracted from A; tiebreaker was received from its parent. So in that case, if my tiebreaker turned out to be the same as the element c, whose count was just equal to floor(n/2), then the total count of the element in the current stage becomes 1+floor(n/2), which makes it the majority.

- Also note that when A is of odd length, we extract the tiebreaker from A, so the count value already contains the count of the instance of the tiebreaker, so we do not want to check the second half of the OR condition here. So here, for the element to be a majority, its count has to be greater than floor(n/2).

- Some nice examples to dry run and visualize the algorithm
  [1, 2, 3]
  [1, 1, 2, 2, 3]
  [1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2]
  [2, 2, 2, 3, 4]