Majority Element
Rengakrishnan Subramanian
December 08, 2001

Let T [1..n] be an array of n integers. An integer is a *majority* element in T if it appears strictly more than n/2 times in T. This paper is concerned with finding an algorithm that can decide whether an array T [1..n] contains a majority element, and if so, find it. The algorithm runs in linear time in the worst case.
Also, the only comparisons allowed between the elements of T are tests of equality. This means, there does not exist an order relation between the elements of the array.

Input

An input array, unsorted, of the type T [i..j].

Output

The majority element *ele*, if it exists. If a majority element does not exist, the algorithm returns a *null*.

Algorithm

Introduction
The algorithm *findmajority (A [i..j])* approaches the problem in two steps. In the first step, the algorithm calls another algorithm *findelement (A[i..j]* to find out a possible element that could be the majority element in the array A. The algorithm *findelement(A [i..j])* returns an element from the array A. In the second step, the algorithm *findmajority ()* actually finds out whether the element, *ele* returned by *findelement( )* is the majority in A by counting the number of occurrences of *ele* in A.

Working
As described above, in the first step, the algorithm calls the function *findelement (A [i..j]).* This function takes the first element and puts a *mark* and considers it as the *majority* element, *ele*. It then takes this element, *ele* and its mark, *mark* as the basis and traverses through the other elements of the array and does the following checks:

   a. If the next element in the array is equal to *ele*, then *mark* is incremented by 1.
   b. If the next element in the array is not equal to *ele,* then *mark* is decremented by 1.
   c. If at the next element, the *mark* is equal to zero, then *mark* is set to one and this element is considered as *ele*.

Finally, *findelement( )* returns *ele* as the possible majority element.

In the second step, *findmajority (A [i..j])*, takes this element *ele* and counts the number of occurrence of the element in the array A. If the count is greater than n/2, this element is the majority element, otherwise, null is returned.

Algorithm

```
procedure  findmajority (A [i..j])

    element ← findelement (A [i..j])

    for (x = i to j)
            if  A[i] = element
                    count ← count + 1
            endif
    endfor

    if (count > n/2)
            return element
    else
            return null

end procedure


 procedure findelement (A [i..j])

    m ← 0

    for (x = i..j)

            if (m = 0)
                    m ← 1
                    ele ← A [x]

            else if (A [x] = ele)
                    m ← m + 1

            else  m ← m –1
            endif
    enfor

    return ele

endprocedure
```

Proof of Correctness

Proof by Loop Invariant
The loop invariant for the algorithm *findelement(A [i..j])* is:

At any point of time, for the array A [i..x],
1. m = 0, findmajority(A, i, x) = null
2. m > 0, findmajority (A', i, x) = null, where A' = { A – *m* elements of *ele* in A }

We need to show that the loop invariants hold for each iteration of the loop. We show it by ensuring the invariants hold true:

a. At initialization – before the loop starts,
b. During maintenance – during the execution of the loop

Initialization
At initialization, m = 0 and the array has not yet been traversed. At this point, surely, the majority element is null in the array. Condition 1 is thus satisfied. Also, at initialization, m is not greater than zero. So, condition 2 is also satisfied.

Maintenance

```
if (m = 0)
       m ← 1
       ele ← A [x]
```
(a)

```
else if (A [x] = ele)
       m ← m + 1
```
(b)

```
else  m ← m –1
```
(c)
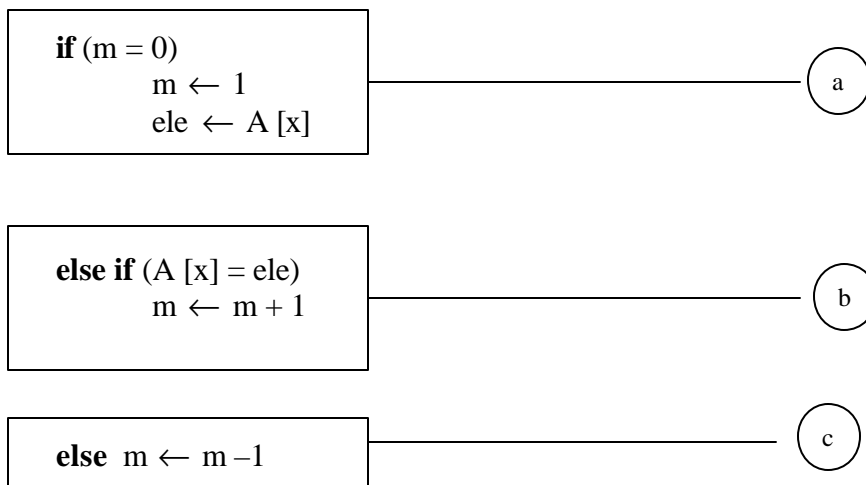
For proving the loop invariant through the iterations of the algorithm, the algorithm is divided into three parts as shown above.

Case a)
In this case, the value of m is zero. This means that either this is the first iteration of the loop or there have been many iterations, where m has been incremented and decremented continuously. The former case is same as initialization phase. In the latter case, the algorithm has paired all elements that are distinct. So, at this point, the current element is

named as the *ele*, thus maintaining the condition 1 of the loop invariant. Since m is not greater than zero, condition 2 is also satisfied.

Case b)
In this case, it is true that value of m is definitely greater than zero. So, it is true that in elements so far traversed by findelement, there is at least one element that cannot be paired with another element distinct from *ele*. Thus, it is true that *ele* is the element that is majority in the array until now and it has occurred at least *m* times in the array. It is to be noted that, if *ele* has occurred for more than *m* times (say *M* times), then all the *M –m* times *ele* has occurred, it has been paired up with other distinct elements. So, in the array that consists of *M-m* occurrences of *ele* and other elements, there is no majority element. Thus, condition 2 is satisfied. It is also to be noted at this point that, since m = 0, condition 1 is also satisfied.

Case c)
This point of algorithm is reached iff m > 0. Thus, it can said that there is at least one element that is distinct from *ele* and can be paired with. After this operation m >= 0. If m = 0, then it is true that there is no *ele* that can be said as majority. So, condition 1 is satisfied. If m > 0, it can be argued in the same way as case b that condition 2 is satisfied.

Time Complexity Analysis
The algorithm majority takes a time in O (n) to find the majority element in an array. The algorithm *findelement (A [i..j])* has only one for loop and takes a time in O (n). The *findmajority (A [i..j])* algorithm also goes through only one for loop and takes a time in O (n). The total time complexity is thus, O (n) + O (n) = **O (n)**.

References

[1] Gilles Brassard and Paul Bratley. *Fundamentals of Algorithmics.* Prentice Hall of India, 1998.