CSCI-GA.1170-001/002 Fundamental Algorithms October 14, 2015

Solutions to Problem 5 of Homework 5 (15 points)

Name: GOWTHAM GOLI (N17656180) *Due: Tuesday, October 13*

Let us say that a number x is *c-major* for an n-element array A, if more than n/c elements of A are equal to x.

(a) (6 pts) Give $O(n)$-time algorithm to find all 2-major elements of A. How many could there be?

Solution:

The key idea here is that the 2-majority element remains preserved when a pair of distinct elements are canceled out from the array.

Let α is the 2-majority element in the array. Now if two distinct elements β and γ are discarded, the array length now becomes $n-2$. Therefore α is a 2-majority element. If two distinct elements α and β are discarded the array length becomes $n-2$ and there are $> n/2-1$ occurrences of α. Therefore α is a 2-majority element.

Let there be x number of 2 majority elements in A. Therefore, there are at least $xn/2$ elements in A

$\therefore xn/2 < n \implies x < 2 \implies$ There can be only one 2-majority element in A

Psuedocode

```
Algorithm: Find-2-Majority(A)
count ← 0
for i ← 0 to n-1 do
    if count is 0 then
        x ← A[i]
        count ← 1
    else if x ≠ A[i] then
        count ← count − 1
    Else  count ← count + 1
end
num ← Count-Occurrences(x, A)
if num > n/2 then
    Return x
else
    Return NotFound
```

Algorithm 3: Algorithm to find 2-majority element of A in $O(n)$ time

In the above psuedocode, during i^{th} iteration, we compare $A[i-1]$ with x and cancel both if they are different, and increment count otherwise.

So if $count = 0$, then all elements upto $A[i-1]$ would have been eliminated through distinct-elements pair formations. If $count > 0$, then $\{x, \ldots \text{ count times } \ldots, x, A[i], \ldots A[n-1]\}$ elements would have still survived at the end of the i^{th} iteration. Therefore we are effectively canceling out the distinct elements. Thus at the end of the *for* loop, if there is a 2-majority element it survives. We are doing a linear scan of the array in the for loop and in the last step we find the number of occurrences of x in A which also takes linear time. Therefore running time of the algorithm is $O(n)$ □

(b) (9 pts) Give $O(cn)$-time algorithm to find all c-major elements of A. How many could there be?

Solution: Following the same idea, that if c distinct elements of the array are canceled out the c majority element still remains preserved in the array.

In the below psuedocode, we maintain two arrays *temp* and *count* of length k. We keep iterating over the n elements of the given array and if it matches with any of the k elements in *temp*, we increase the *count* of that element. If none of the elements of *temp* matches, we decrease the count of every element (i.e we are canceling out the distinct elements). If there is an empty slot in *temp* (i.e *count* of that element is 0) then we place the element at that position and set it's *count* to 1. At then end, we individually check for each element of *temp*, if it is a c-majority element.

Let there be x number of c majority elements in A. Therefore, there are at least xn/c elements in A

$\therefore xn/c < n \implies x < c \implies$ There can be at most $c-1$, c-majority elements in A

Psuedocode

```
Algorithm: Find-c-Majority(A)
temp, count ← NewArray(c)
Initialize count to 0
for i ← 0 to n-1 do
    for j ← 0 to c − 1 do
        if temp[j] is A[i] then
            count[j] ← count[j] + 1
            break
        end
        if j is c-1 then
            for p ← 0 to c − 1 do
                if count[p] is 0 then
                    temp[p] ← A[p]
                    count[p] ← 1
                    break
                end
            end
            if p is c-1 then
                for p ← 0 to c − 1 do
                    count[p] ← count[p] − 1
                end
            end
        end
    end
end
for i ← 0 to c − 1 do
    num ← NumberOfOccurrences(temp[i], A)
    if num > n/c then
        temp[i] is a c-major element
    end
end
```

Algorithm 4: Algorithm to find c-majority element of A in $O(nk)$ time

Time Complexity

The first outer *for* loop runs for n times and all the nested *for* loops run for c times. Therefore, the first phase takes $O(nc)$

The second *for* loop runs for c times and in each iteration we find the number of occurrences of the element of *temp* in A. It takes $O(n)$ time. Therefore, the second phase takes $O(nc)$ time. Hence, the total running time of the algorithm is $O(nc)$ $\square$