CS29003 ALGORITHMS LABORATORY
Assignment No: 8
Last Date of Submission: 25–September–2014

Let A be an array with n elements. An element a is called a *majority element* in A if a occurs more than n / 2 times in the array A. Every array need not contain a majority element. But if it does, the majority element is unique. In this assignment, you implement a linear-time algorithm to find the majority element in an array (if it exists).

One possibility to solve this problem is to sort A and search for an index i such that $A[i]$ is the same as $A[i+(n/2)]$. Since the initial sorting takes $O(n \log n)$ time in the worst case, this algorithm has the same worst-case time complexity. A better algorithm is to find the median a in A by a linear-time algorithm, and then check whether a is actually the majority element in A. In this exercise, you implement a practically better and simpler linear-time algorithm.

The boundary cases $n = 0$ and $n = 1$ are easy to handle. So suppose that $n \geq 2$. To start with, assume that n is even. Prepare an array B of size $\leq n/2$ as follows. Arbitrarily pair the n elements of A into n / 2 groups. For example, a common pairing is ($A[0]$, $A[1]$), ($A[2]$, $A[3]$), …, ($A[n–2]$, $A[n–1]$). For each group (a,b) so formed, check whether a equals b. If so, copy this element to the array B. If not, do nothing. After all pairs are considered, recursively compute the majority element in the array B. If the recursive call returns "*no majority element*" (in B), then A too does not contain a majority element. However, if the recursive call returns the majority element a in B, we are not done yet. Now, the returned element a may or may not be the majority element in A. For example, consider the two possibilities for A: a,a,b,a,a,a and a,a,b,c,b,c,a,a. In both these examples, the recursive call is made on the array B equal to a,a, and the recursive call returns a as the majority element in B. In the first case, a is actually the majority element in A, whereas in the second case, A does not contain a majority element. Therefore, after the recursive call on B returns a majority element a, we need to check whether a is actually the majority element in A.

The running time of this algorithm satisfies

$$T(n) \leq T(n/2) + \Theta(n),$$

that is, by the master theorem of divide-and-conquer recurrences, $T(n) = \Theta(n)$. Implement the algorithm mentioned above as a function which returns the majority element in A if it exists, or an invalid character (like –1) otherwise.

You must be able to handle odd values of n. Even if n is even (n may even be a power of two), the recursive call may be on an array B of any size $\leq n$ / 2. We leave it to the students to devise and implement the strategy how to handle odd values of the array size in any of the recursive calls.

Write a *main*() function to do the following.

- Read a string A of length n. Assume that the symbols in A come from $\{a,b,c,d,...\}$.
- Call the function to compute the majority element of A.
- If the majority element exists, print its frequency in A.
- If the majority element does not exist, print the frequencies of all the symbols in A.

Sample output

Here follow two sample runs, each on an input of length 80. In the first case, the majority element exists. In the second case, the majority element does not exist. The array passed to each invocation of the majority-finder function is printed.

```
--- ffeffbfdffaecffffebbbffacfefffbaefafdfceefafffafdfcffffdffffbacfcdbfffcbbbfffcdb
--- fffbffffffbf
--- ffff
--- ff
--- f
Majority element is f with frequency 41

--- ccebcafadcccccfcbadffabfcecccafcdbcccccacdcfeadccffecccaeacbaedbaeecbbdfcdccfdcc
--- cccccccbcc
--- cccc
--- cc
--- c
No majority element exists
+++ Frequency(a) = 11
+++ Frequency(b) = 8
+++ Frequency(c) = 33
+++ Frequency(d) = 9
+++ Frequency(e) = 8
+++ Frequency(f) = 11
```