Notes on the Boyer-Moore Algorithm

In class we discussed the Boyer-Moore algorithm for finding the majority element in an array $A[1..n]$ (i.e., a value that appears in strictly more than half of the array's entries) if such a majority element exists. Even though the algorithm and the problem it solves are quite simple, it takes a bit of work to see exactly why it is correct. Here is a more detailed version of the argument covered in the lecture.

The first phase identifies a possible candidate *cand* for the majority element. The second phase checks to see if *cand* really is a majority element. Thus, the property we want to ensure after the first phase is that *if* there is a majority element in the array, it is *cand*.

The second phase is straightforward: just count the number of copies of *cand* to see if it occurs in more than half the array locations. The first phase is trickier. The algorithm stores its current candidate in *cand*. It increments a counter *count* when it sees a copy of *cand* and decrements *count* when it sees a copy of a different value. When the counter hits 0, *cand* is changed to the next element of the array.

```
1:  BOYER-MOORE(A[1..n])
2:      ▷ Phase 1: find a candidate for the majority value
3:      count ← 0
4:      for i ← 1..n do
5:          if count = 0 then cand ← A[i]
6:          if A[i] = cand then count++
7:          else count−−
8:      ▷ Phase 2: check if cand really is a majority
9:      count ← 0
10:     for i ← 1..n do
11:         Invariant: count is the number of copies of cand in A[1..i − 1]
12:         if A[i] = cand then count++
13:     if count > n/2 then return cand is a majority
14:     else return there is no majority
```

Why does Phase 1 work to identify the majority element (if there is one)?

To understand how Phase 1 works, we must understand what information is stored in the variables *cand* and *count* after each iteration. Let $cand_i$ and $count_i$ denote the values of the two variables after i iterations of the loop. These values satisfy the following invariant.

Loop Invariant: The elements of $A[1..i]$ can be partitioned into a bunch of pairs of non-equal elements plus $count_i$ copies of *cand*.

We call a pair of non-equal elements a *mismatched pair*. As usual, we carry out the three-step plan for using this loop invariant:

(a) Show that the invariant holds after 0 iterations.

Initially, $count_0 = 0$ and it is true that the first 0 elements of A can be partitioned into 0 mismatched pairs and 0 extra copies of *cand*.

(b) Let $i > 0$. Show that if the invariant holds after $i - 1$ iterations then it holds after i iterations.

Assume that $A[1..i-1]$ can be partitioned into mismatched pairs and $count_{i-1}$ extra copies of $cand_{i-1}$. We consider three cases, depending on what happens in iteration i (the cases correspond to the possible outcomes of the conditions tested by the if statements).

Case 1 ($count_{i-1} = 0$): Then $cand_i = A[i]$ and $count_i = 1$. By the induction hypothesis, $count_{i-1} = 0$ implies that $A[1..i-1]$ consists of a bunch of mismatched pairs, with no extra copies of $cand_{i-1}$. Thus, $A[1..i]$ consists of a bunch of mismatched pairs and one extra copy of $A[i]$, as required.

Case 2 ($count_{i-1} \neq 0$ and $A[i] = cand_{i-1}$): Then, $cand_i = cand_{i-1}$ and $count_i = count_{i-1} + 1$. By the induction hypothesis, $A[1..i-1]$ consists of mismatched pairs plus $count_{i-1}$ copies of $cand_{i-1}$. Since

$A[i] = cand_{i-1}$ too, this means that $A[1..i]$ consists of mismatched pairs plus $count_{i-1} + 1 = count_i$ copies of $cand_{i-1} = cand_i$, as required.

Case 3 ($count_{i-1} \neq 0$ and $A[i] \neq cand_{i-1}$): Then, $cand_i = cand_{i-1}$ and $count_i = count_{i-1} - 1$. By the induction hypothesis, $A[1..i-1]$ consists of mismatched pairs plus $count_{i-1}$ copies of $cand_{i-1}$. Since $A[i] \neq cand_{i-1}$, we can match it with one of the extra copies of $cand_{i-1}$ in $A[1..i-1]$ to form a mismatched pair. This reduces the number of unmatched copies of $cand_{i-1}$ by 1. Thus, $A[1..i]$ consists of some mismatched pairs plus $count_{i-1} - 1 = count_i$ copies of $cand_{i-1} = cand_i$, as reqiured.

(c) Use the loop invariant to show that the postconditions of the loop hold.

In this case, we must show that at the end of Phase 1, if there is a majority element in $A[1..n]$, then it is *cand*.

Suppose there is a majority element m in $A[1..n]$. By the loop invariant (with $i = n$), we know that $A[1..n]$ consists of mismatched pairs plus $count_n$ extra copies of $cand_n$. There are at most $n/2$ mismatched pairs (since there are n elements in total) and each of them contains at most one copy of m. Thus some of the extra copies of $cand_n$ must be m, so $cand_n = m$, as required.

This completes the proof that Phase 1 works. It is straightforward to see why Phase 2 works. If there is a majority element m in the array A, we have shown that *cand* will be m at the end of Phase 1, and then Phase 2 will verify that m occurs more than $n/2$ times and output m. If there is no majority element in A, then no matter what value *cand* has at the end of Phase 1, the check in Phase 2 will not find more than $n/2$ copies of it, so the algorithm will output "there is no majority".