A Proof of the Boyer-Moore Majority Protocol

Jayadev Misra

January 19, 2017

1 The Boyer-Moore Majority Finding Algorithm

Given is a finite bag $B$ of items. An item may occur multiple times in a bag. Each of its occurrences is called an element. A majority item is one that occurs more often in the bag than all other items combined.

It is required to find the majority item of $B$, if one exists, or indicate that there is no such item. The Boyer-Moore algorithm solves this problem by making two passes over the elements of $B$ where each pass runs in linear time. The first pass suggests a candidate item for majority, or that no such candidate exists (so there is no majority item). The second pass determines if the suggested candidate is indeed a majority item. The second pass is straightforward, and we consider only the first pass in this note.

An Abstract Algorithm

The algorithm is based on the next proposition.

Proposition 1 Removing two distinct items from a bag does not alter the majority item, if the bag has one.

Proof: Let the bag have a majority item $m$ and let its number of occurrences be $n$. Let the number of occurrences of all other items in the bag be $n'$. Then $n > n'$. After removal of two distinct items, $n$ is reduced by at most 1 because at most one copy of $m$ could have been removed. And $n'$ is reduced by at least 1 because at least one non-majority item is removed. So, $n > n'$. □

An abstract algorithm, based on this proposition, is as follows: continue removing pairs of distinct elements from bag $B$ as long as it is possible. At termination either the bag is empty — so, there is no majority item — or it has one or more instances of a single item that is a candidate for the majority. A second pass over $B$ determines if the candidate item is a majority. The rest of this note describes refined versions of this abstract algorithm.

A Refined Algorithm

The first refinement uses two auxiliary bags, $b$ and $c$, both subbags of $B$. All the elements in $c$ will be identical; so, to get two distinct elements it is sufficient to find an element of $b$ that differs from one in $c$.

Overload the usual set notation to apply to bags and $\emptyset$ to denote the empty bag. Initially $b, c := B, \emptyset$. Repeat the following step until $b$ becomes empty:
pick an arbitrary element $x$ of $b$ and if $x$ differs from any element $y$ of $c$ remove $x$ and $y$ from their respective bags, otherwise (i.e., $x$ matches every element of $c$) transfer $x$ from $b$ to $c$. Formally, $x$ matches every element of an empty bag.

The algorithm is as follows:

Initially $b, c := B, \emptyset$

while $b \neq \emptyset$
do

pick any $x$ from $b$ \{ $x \in b$ \}

if there exists $y$ in $c$ such that $x \neq y$

then $b, c := b - \{x\}, c - \{y\}$

else \{ for all $y$ in $c$, $x = y$ \}

$b, c := b - \{x\}, c \cup \{x\}$

endo

2 Correctness

We claim that the algorithm terminates, and on termination either $c$ is empty or it consists of a single item, $m$, perhaps multiple times. If $B$ has a majority item then it is $m$. The converse does not hold, i.e., if $c$ includes $m$ then $m$ is not necessarily the majority item because $B$ may have none. So, a second pass determines if indeed $m$ is the majority item. Let $maj(A)$ denote the majority item of any bag $A$ if one exists.

Proposition 2 Invariant: All elements of $c$ are identical.

Proof: The invariant holds initially because $c$ is empty. In each step either an element is removed from $c$, or $x$, which equals every item of $c$, is added to $c$. Therefore, a step preserves the invariant.

Proposition 3 Given $B$ has a majority item $m$, $maj(b \cup c) = m$ is invariant.

Proof: The invariant holds initially because $b \cup c = B$ and $m = maj(B)$. Next, we show that each step preserves the invariant. The step corresponding to the “else” clause does not alter $b \cup c$. For the step corresponding to the “then” part, the required annotation is shown below.

$$\{maj(b \cup c) = m \land x \in b \land y \in c \land x \neq y\}$$

$b, c := b - \{x\}, c - \{y\}$

$$\{maj(b \cup c) = m\}$$

Using the axiom of assignment, we need to show,

$$(maj(b \cup c) = m \land x \in b \land y \in c \land x \neq y) \Rightarrow maj((b - \{x\}) \cup (c - \{y\})) = m.$$

Since $(b - \{x\}) \cup (c - \{y\}) = (b \cup c) - \{x, y\}$ and $x$ and $y$ are distinct, the assertion follows by using $(b \cup c)$ as the bag in Proposition 1.

Proposition 4 The algorithm terminates.

Proof: Each iteration of the loop reduces the size of $b$ by 1; so, the number of iterations is exactly $|B|$. \qed
**Proposition 5** Suppose $B$ has a majority item $m$. Then, at termination $c$ consists of one or more copies of $m$.

Proof: From Proposition 3, $maj(b \cup c) = m$ is invariant. So, at termination, using $b = \phi$, conclude $maj(c) = m$, so $c$ is not empty. From Proposition 2, $c$ holds only copies of $m$. □

3 Algorithm Optimization

Bag $c$ consists of zero or more copies of a single item. Therefore, it may be encoded by two values $m$ and $n$, the item in $c$ and the number of its occurrences (if $n = 0$, $m$ is irrelevant). The algorithm can then be written as follows.

initially $b, m, n := B, -, 0$

while $b \neq \phi$ do

pick any $x$ from $b$ \{$x \in b$\}

if $n > 0 \land x \neq m$ \{c is non-empty, so $m$ has a value\}

then $b, n := b - \{x\}, n - 1$

else \{$n = 0 \lor (n > 0 \land x = m)$\} $b, m, n := b - \{x\}, x, n + 1$

enddo

Note that $m$ assigned to $x$ in the last step of the program because preceding this step if $n = 0$ then $m$ may have an irrelevant value.

References


