Special case of most frequent item in a stream

Suppose to have a stream of n items, so that one of them occurs $> n/2$ times in the stream. Also, the main memory is limited to keeping just two items and their counters, plus the knowledge of the value of n beforehand. Show how to find deterministically the most frequent item in this scenario. [Hint: since the problem cannot be solved deterministically if the most frequent item occurs $\leq n/2$ times, the fact that the frequency is $> n/2$ should be exploited.]

SOLUTION

This solution works keeping in memory only one counter c and a stream alphabet value v.
Consider the stream as an array S of elements. The algorihm will work this way for each element in S:

1. set $i = 1$, $c = 1$, $v = s[0]$
2. if $c > 0 \wedge S[i] = v$ we increase the counter c
3. if $c > 0 \wedge S[i] \neq v$ we decrease the counter c
4. if $c = 0$ then set $c = 1$ and change the value $v = S[i]$
5. if there is more stream to analyze increase i and repeat from step 2. else return v as the most frequent item

obs1 The counter increases each time we find an element v' in the stream such that $v = v'$ and decreases each time we find $v \neq v'$.

So when the counter $c = 0$ the algorithm has found a portion of stream where there are as many $v' \neq v$ as those such that $v' = v$. The portion starts from the position where we changed the value of v and ends where the counter was set to 0. Every time the counter goes to 0 we can define such portion.
Let's call P_i the i-th parition created and l_i its length. Defining $f_i(v_i)$ as the frequency of (the number of occurence of) the element v_i from **obs1** we can say that

$$f_i(v_i) = \sum_{x \in P_i, x \neq v_i} f_i(x) \tag{1}$$

that implies that for every element $x \in P_i$ it will be

$$f_i(x) \leq f_i(v_i) \tag{2}$$

Also, from (1), we notice that $f_i(v_i) = l/2$ and so for each element $x \in P_i$

$$f_i(x) \leq \frac{l_i}{2} \tag{3}$$

i.e. all the element in a portion will occur at most half the length of that portion.

obs2 When the algorithm stops, $c > 0$. Infact, since the frequency of an element x is $f(x) = \sum_{i=1}^{k} f_i(x)$, if the algorithm would stop with $c = 0$ it would mean that for all $x \in S$

$$f(x) = \sum_{i=1}^{|P|} f_i(x) \leq \sum_{i=1}^{|P|} \frac{l_i}{2} = \frac{n}{2}$$

where $|P|$ is the number of portions created, and that is not compatible with the problem hypothesis that there is one element that appears more than $n/2$.

obs3 At the end of the execution we can then identify another portion, say P_k, that begins the last time the value v changed and finishes at the last element of the stream. In this portion the element v_k is such that it occurs *more* than the sum of all other:

$$f_k(v_k) > \sum_{x \in P_k} f_k(x) \tag{4}$$

and so the algorithm will split the stream in k portions such that $k-1$ ends with counter set to 0 and the k-th with $c > 0$.

From all this observation we can now prove that, since the algorithm will stop with $c > 0$, the value $v = v_k$ returned from it will be the most frequent element. Infact from all the observation above we know that if $x \neq v_k$

$$\begin{aligned} f(x) = \sum_{i=1}^{k} f_i(x) &\leq \\ \leq \sum_{i=1}^{k} f_i(v_i) &\leq && \texttt{from } \textit{(2)} \\ \leq \sum_{i=1}^{k} \frac{l_i}{2} &= \frac{n}{2} && \texttt{from } \textit{(3)} \end{aligned}$$

Since there can be only one element x^* such that $f(x^*) > n/2$ it will necessary be v_k and then the algorithm is correct.