Deterministic data streaming

Consider a stream of n items, where items can appear more than once in the stream. The problem is to find the most frequently appearing item in the stream (where ties broken arbitrarily if more than one item satisfies the latter). Suppose that only k items can be stored, one item per memory cell, where the available storage is $k + O(1)$ memory cells. Show that the problem cannot be solved deterministically under the following rules: the algorithm can access only $O(log^c n)$ bits for each of the k items that it can store, and can read the next item of the stream; you, the adversary, have access to all the stream, and the content of the k items stored by the algorithm, and can decide what is the next item that the algorithm reads (please note that you cannot change the past, namely, the items already read by the algorithm) . Hint: it is an adversarial argument based on the k items chosen by the hypothetical deterministic streaming algorithm, and the fact that there can be a tie on $> k$ items till the last minute.

Show that the problem cannot be solved deterministically under the following rules: the algorithm can only use b bits, and read the next item of the stream, one item at a time. You, the adversary, have access to all the stream, and the content of the b bits stored by the algorithm: you cannot change those b bits and the past, namely, the items already read by the algorithm, but you can change the future, namely, the next item to be read. Since the algorithm must be correct for any input, you can use any amount of streams to be fed to the algorithm and as many distinct items as you want. [Hint: it is an adversarial argument based on the fact that, for many streams, there can be a tie on the items.]

SOLUTION

Any deterministic algorithm can be described by a deterministic finite state machine, in which every state given an input move deterministically to another state. Suppose to have a streaming of n element, k memory cell to store the frequencies, and θ are the bit necessary to store the algorithm state. Therefore we can have 2^θ possible states.

Let's suppose to input $2^\theta + 1$ times the same element x to the algorithm. Then, it must be the case that the algorithm is going to be in the same state in two different moments (notice that the possible state are 2^θ). Let's call A that particular state, let's use A_i and A_j to denote the two different moments in which we arrive in state A during the execution of the algorithm (without loss of generality we can assume $i < j - 1$, we will later handle the $i = j - 1$ case).

Now let's input to the algorithm an element $y \neq x$ for $(i + 1)$-times, starting from both A_i and A_j. Then let's proof that the algorithm replies in two different ways, depending on whether it starts from A_i or A_j:

Case 1: when we start from A_i, we will end up in this situation:
(for simplicity, we'll use a JSON-like syntax for representing the counters)
Counters: $\{x : i, y : i + 1\} \implies$ the algorithm outputs y as the most frequent element.

Case 2: when we start from A_i, we will end up in this situation:
Counters: $\{x : j, y : i + 1\} \implies$ since $j > i + 1$, the algorithm outputs x.

We've seen how, starting from the same state A and with the same input, the algorithm produces different outputs; therefore it cannot be deterministic.

case $i = j - 1$

If i=j-1, then the algorithm contains the transition $(A_i, x) \to A_j = A_{i+1}$. This means that the algorithms remains in state A after receving x. In particular, $A_{i+2} = \delta(\delta(A_i, x), x) = \delta(A_i, x) = A_i$, and thus we can just take $j = i + 2$ and we fall in the previous case $i < j + 1$.

ALTERNATIVE SOLUTION (well, same solution with different proof)

Let Λ be the alphabet of the stream, we consider the set of streams S to be the set of all possible concatenations of arbitrary length of word on the alphabet (i.e. $S = \Lambda^*$).

Any deterministic algorithm can be described by a deterministic finite state machine, in which every state $\sigma \in \Sigma$ given an input move deterministically to another state. The machine is deterministic, hence the relation between the pairs of possible states and streams with the states the machine will transit starting from that given state and seeing that particular stream is a function. Also the relation between the pair of state and streams with the output is a function.

$update\_state : \Sigma \times S \longrightarrow S$

$update\_state(\sigma, s) = \sigma'$ if when the machine is in state σ and sees the stream s go to state σ'

$output : \Sigma \times S \longrightarrow \Lambda$

$output(\sigma, s) = a$ if when the machine is in state σ and sees the stream s resturn in output a

Given that the number of bits for representing the state of the machine is finite (say k), the possible different states cannot be more than 2^k (i.e. $|\Sigma| \leq 2^k$).

Take one element a from the alphabet of the stream and consider the set of streams $\{a\}^*$[1]. Let σ_0 be the initial state of the machine (the state of the machine before it sees any character). The set of states in which the machine can be after seeing one element from $\{a\}^*$, starting from σ_0 will be $update\_state(\sigma_0, \{a\}^*)$ defined in the usual way.

$$update\_state(\sigma_0, \{a\}^*) = \{\sigma | \exists s \in \{a\}^* update\_state(\sigma_0, s) = \sigma\}$$

Since the streams in the set $\{a\}^*$ are infinite, and $update\_state(\sigma_0, \{a\}^*)$ is a subset of Σ and hence a finite set, the function cannot be injective. There will be infinite different stream such that they lead to the same state of the machine. Let a^i and a^j be two different streams such that they lead to the same state $\bar{\sigma}$ of the machine and without lost of generality assume that $i > j + 1$.

$$update\_state(\sigma_0, a^i) = update\_state(\sigma_0, a^j) = \bar{\sigma} \qquad \text{with } i > j + 1$$

Now take a new stream composed by $j + 1$ word $b \in \Lambda$ different from a. The machine in state $\bar{\sigma}$, when sees the stream b^{j+1} will gives as output $output(\bar{\sigma}, b^{j+1})$.

Let's assume that the machine gives always the correct answer, we will prove it must be not deterministic. First case, the machine is in state $\bar{\sigma}$ because of the stream a^j, then to give the correct answer the algorithm must return b because in the stream $a^j b^{j+1}$ the most frequent item is b. Second case, the machine is in state $\bar{\sigma}$ because of the stream a^i, then to give the correct answer the algorithm must return a because in the stream $a^i b^{j+1}$ the most frequent item is a (since $i > j + 1$).

Hence if the algorithm must be correct the relation between the pair of state and stream with the output cannot be a function and the algorithm cannot be deterministic.

$$\begin{aligned} output(\bar{\sigma}, b^{i+1}) &= b && \text{sometimes} \\ output(\bar{\sigma}, b^{i+1}) &= a && \text{some other times} \end{aligned}$$

And by composition of state and output functions

$$\begin{aligned} output(\sigma_0, a^i b^{i+1}) &= b && \text{sometimes} \\ output(\sigma_0, a^i b^{i+1}) &= a && \text{some other times} \end{aligned}$$

[1]This is the set $\{a^i | i \in \mathbb{N}\}$ where we mean a^i to be the concatenation of i times the word a, you can see it like the language indicated by the regular expression a^*.