University of Pisa

Department of computer science

Algorithm design

Seventh hands-on: Deterministic data streaming

Domenico Erriquez

1. PROBLEM

Consider a stream of n items, where items can appear more than once. The problem is to find the most frequently appearing item in the stream (where ties are broken arbitrarily if more than one item satisfies the latter). For any fixed integer $k \geq 1$, suppose that only k items and their counters can be stored, one item per memory word; namely, only O(k) memory words of space are allowed.

Show that the problem cannot be solved deterministically under the following rules: any algorithm can only use these O(k) memory words, 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 these O(k) memory words: you cannot change these words and the past items, namely, the items already read, but you can change the future, namely, the next item to be read. Since any algorithm must be correct for any input, you can use any amount of streams and as many distinct items as you want.

Hints:

1. This is "classical" adversarial argument based on the fact that any deterministic algorithm A using O(k) memory words gives the wrong answer for a suitable stream chosen by the adversary.
2. The stream to choose as an adversary is taken from a candidate set sufficiently large: given O(k) memory works, let f(k) denote the maximum number of possible situations that algorithm A can discriminate. Create a set of C candidate streams, where C > f(k): in this way there are two streams S1 and S2 that A cannot distinguish, by the pigeon principle.

2. SOLUTION

To begin, let's define a function $most\_frequent$ which takes as input a stream and returns the exact most frequent item of the stream. The goal is to show that for any deterministic algorithm A using $O(k)$ space, we can generate two streams S_1, S_2 such that $most\_frequent(S_1) \neq most\_frequent(S_2)$ but for the algorithm A we have that the two strings are the same, so $A(S_1) = A(S_2)$. Since the algorithm A uses $O(k)$ space, we indicate with $f(k)$ the finite number of situations the algorithm can discriminate, so we must generate a set of different streams with cardinality greater than $f(K)$.

To do so, let's take a universe U and its subsets $\sum_i \subseteq U$ where their cardinality is $|\sum_i| = \left\lfloor \frac{|U|}{2} \right\rfloor + 1$ and $\forall\, i, j$ with $i \neq j$ they satisfy the following properties:

1. $|\Sigma_i \cap \Sigma_j| \geq 1$: they have at least one element in common
2. $|\Sigma_i \setminus \Sigma_j| \geq 1$: they have at least one element not in common

All the possible subsets are:

$$\binom{|U|}{\left\lfloor \frac{|U|}{2} \right\rfloor + 1.} \approx 2^{|U|}$$

Now for all Σ_i we define the corresponding stream:

$$S_i \stackrel{\text{def}}{=} s_1 s_2 \dots s_{\left\lfloor \frac{|U|}{2} \right\rfloor + 1}$$

Where S_i contains all the elements of Σ_i repeated only once.

Let's define the set S of all the streams S_i defined as above. Since we build for each subset a stream, we have that the cardinality of S is $|S| = 2^{|U|}$.
For our goal we need that $|S| > f(k)$, so:

$$|S| > f(k) \leftrightarrow 2^{|U|} > f(k) \leftrightarrow |U| > \log_2 f(k)$$

If we take a universe U with cardinality greater than $\log_2 f(k)$, the set of streams S will have a cardinality greater than $f(k)$ and then, for the pigeonhole principle there exists two different streams $S_i, S_j \in S$ indistinguishable for the algorithm A, which means that the two streams are the same for the algorithm A.

Therefore, if we pick $x \in \Sigma_i \setminus \Sigma_j$ and $y \in \Sigma_j \setminus \Sigma_i$(in other words $x \in \Sigma_i, x \notin \Sigma_j \; and \; y \in \Sigma_j, y \notin \Sigma_i$) and we concatenate them to streams S_i, S_j obtaining $S_i xy, S_j xy$, we will have that $most\_frequent(S_i xy) = x$ because x was already present in the stream S_i(which contains elements repeated only one), so now x is repeated two times. For the same reason $most\_frequent(S_j xy) = y$. Thus $most\_frequent(S_i xy) \neq most\_frequent(S_j xy)$.

But because the streams S_i, S_j are indistinguishable for the algorithm A, they are also $S_i xy, S_j xy$, then we can conclude that $A(S_i xy) = A(S_j xy)$, so the algorithm A gives a wrong answer