Yanwen Gong
CS 261, Spring 2020
Homework 4
Due Monday, April 27

1. Suppose that we want to maintain a random sample of a cash-register-model data stream whose size depends on the number of stream elements seen so far: after seeing n elements, we should have a sample of size $\lceil \log_2 n \rceil$, chosen uniformly at random among all possible samples of that size. However, we have no advance knowledge of how big n is going to be.

Explain why, in this scenario, it is not possible to maintain a sketch that allows us to generate samples of the desired size (with the exact probability distribution specified above) using a sublinear amount of space.

To select $log_2 n$ samples uniformly from n samples, we need to keep all n samples in memory to achieve the uniformity. If we only keep a fraction of the n samples, the left out samples don't have equal chance to be selected as the samples in memory. Thus, it's not possible to maintain a sublinear memory.

■

2. Suppose we have two Boyer–Moore majority sketches (m_A, c_A) and (m_B, c_B) for input sequences A and B. Explain how to use them to compute a single sketch for the concatenation of sequences AB, again consisting of a pair of numbers (m, c). Your combined sketch does not have to be equal to the sketch that the majority algorithm would produce for AB, but it should provide an estimate for the number $\text{count}(x)$ of occurrences of each element x that is bounded between $\text{count}(x) - |AB|/2$ and $\text{count}(x)$, just like the majority algorithm would. (In particular, this implies that if AB has a majority element, your combination will choose that element as its value of m). Explain why your combination has this property.

If AB has a majority m_{ab}, its count $|m_{ab}| > \frac{1}{2}|AB|$. Suppose A and B both have majority (m_a, c_a) and (m_b, c_b), then $|m_a| + |m_b| > \frac{1}{2}|AB|$. This leads to $|m_a| + |m_b| + |m_{ab}| > AB$, which is impossible. Thus $|m_{ab}| = |m_a| or |m_b|$

Prove that if $C_a > C_b$ we should choose $m_{ab} = m_b$:
if $C_a > C_b$ and $m_{ab} = m_b$
then $|m_b| > \frac{1}{2}|m_{ab}|$
we have $C_a >= |m_a| - (|A| - |m_a|)$
$C_a >= 2|m_a| - |A|$
$similarly C_b >= 2|m_b| - |B|$
$m_b <= (C_b + |B|)/2$ Thus $\frac{1}{2}|AB| < (C_b + |B|)/2$ $|AB| < C_b + |B|$
$|A| < C_b$
since we set $C_a > C_b$, we have $C_a > |A|$ which is impossible.

Thus, if $m_a = m_b$, we choose m_a. Else if $C_a > C_b$ we should choose m_a; Else if $C_b > C_a$ we should choose m_b

■

3. Suppose that we are maintaining a MinHash sketch of size k for a cash-register-model data stream, and that we additionally store one more piece of information, the *largest* hash value among the k values already stored in the stream.

(a) What is the probability that the nth item in the stream has a hash value smaller than this largest value? You can assume that all items in the stream have distinct hash values and that the hash function is uniformly random.

(b) Suppose that the algorithm for maintaining the sketch, when each new value x is processed, does the following steps. It first checks in constant time whether x has a smaller hash value than the stored largest hash value. If x does have a smaller hash value, the algorithm updates the sketch in $O(k)$ time, but if not the algorithm discards x in constant time. What is the expected total time for this procedure to process a sequence of n values? State your answer using O-notation, as a function of k and n.

(Note: faster algorithms for updating the sketch are possible. Please answer this question using the stated time for an update, rather than using these faster algorithms.)

(a) The probability is equivalent to the probability that the nth element are among the k elements with the smallest hash value. Thus it's $\frac{k}{n}$

(b) O(kn) ∎

4. The lecture notes sketch a method for estimating the number of set elements in any query interval $[\ell, r]$, in the turnstile model. The method uses logarithmically many count-min sketches, for sets S_i generated from the given data set S by rounding each element of S down to a multiple of 2^i.

Provide more detailed pseudocode for a subroutine used in this method that takes as input the pair (ℓ, r) and produces as output a sequence of pairs (i, x) of elements to query in the count-min sketch for S_i. Your subroutine's output should have the property that for each number y in the range $[\ell, r]$, there should be exactly one pair (i, x) in the output such that rounding y down to a multiple of 2^i produces x.

For instance, for the range $[3, 11]$ your output should be the set of three pairs $(0, 3)$, $(2, 4)$, and $(2, 8)$ (in any order). Every number in the range $[3, 11]$ rounds to one of these pairs: 3 rounded to a multiple of 2^0 gives 3, matching the pair $(0, 3)$; 4, 5, 6, and 7 rounded to a multiple of 2^2 give 4, matching the pair $(2, 4)$, and 8, 9, 10, and 11 rounded to a multiple of 2^2 give 8, matching the pair $(2, 8)$.

Given interval [l,r]
let short = l, results = []
while *start* <= *r* **do**
  find the biggest multiple of 2^i that is smaller than start
  results.append((i, start))
  start = 2^i + start
end while
return results

∎