COL866: Foundations of Data Science

Ragesh Jaiswal, IITD

Algorithms for Massive Data Problems: Streaming algorithm

Problem

Design a streaming algorithm for computing the number of distinct a_i's in the sequence $a_1, ..., a_n$.

Algorithm

- Let $M > m$ be a prime number
- Let $H = \{h_{a,b} | a, b \in \{0, 1, ..., M-1\}\}$

`Distinct`$(a_1, ..., a_n)$

- Pick a random h from H
- Initialise $min = h(a_1)$
- For $i > 1$: update min to $h(a_i)$ iff $h(a_i) < min$
- return$(\frac{M}{min})$

Theorem

Let d be the number of distinct elements. With probability at least $(2/3 - d/M)$, we have $\frac{d}{6} \leq \frac{M}{min} \leq 6d$ where M and min are as defined in the algorithm.

Algorithm

- Let $M > m$ be a prime number
- Let $H = \{h_{a,b} | a, b \in \{0, 1, ..., M-1\}\}$

Distinct$(a_1, ..., a_n)$

- Pick a random h from H
- Initialise $min = h(a_1)$
- For $i > 1$: update min to $h(a_i)$ iff $h(a_i) < min$
- return$(\frac{M}{min})$

Theorem

Let d be the number of distinct elements. With probability at least $(2/3 - d/M)$, we have $\frac{d}{6} \leq \frac{M}{min} \leq 6d$ where M and min are as defined in the algorithm.

Proof sketch

- Let $b_1, ..., b_d$ be the distinct values that appear in the input.
- Let $S = \{h(b_1), ..., h(b_d)\}$ and $min = \min(S)$.
- Claim 1: $\mathbf{Pr}[\frac{M}{min} > 6d] < \frac{1}{6} + \frac{d}{M}$.

Algorithm

- Let $M > m$ be a prime number
- Let $H = \{h_{a,b} | a, b \in \{0, 1, ..., M-1\}\}$

`Distinct`$(a_1, ..., a_n)$
- Pick a random h from H
- Initialise $min = h(a_1)$
- For $i > 1$: update min to $h(a_i)$ iff $h(a_i) < min$
- return$(\frac{M}{min})$

Theorem

Let d be the number of distinct elements. With probability at least $(2/3 - d/M)$, we have $\frac{d}{6} \leq \frac{M}{min} \leq 6d$ where M and min are as defined in the algorithm.

Proof sketch

- Let $b_1, ..., b_d$ be the distinct values that appear in the input.
- Let $S = \{h(b_1), ..., h(b_d)\}$ and $min = \min(S)$.
- Claim 1: $\mathbf{Pr}[\frac{M}{min} > 6d] < \frac{1}{6} + \frac{d}{M}$.
- Claim 2: $\mathbf{Pr}[[\frac{M}{min} < \frac{d}{6}] < \frac{1}{6}$.
- The theorem follows from the above two claims. □

Problem

Design an algorithm for counting the number of occurrences of a given element in the stream.

- This can clearly be done using a deterministic algorithm that uses $O(\log n)$ space.
- Question: Can a deterministic algorithm do any better?

Problem

Design an algorithm for counting the number of occurrences of a given element in the stream.

- This can clearly be done using a deterministic algorithm that uses $O(\log n)$ space.
- Question: Can a deterministic algorithm do any better?
- Question: Suppose you are allowed some slack with respect to the answer (say constant factor) and allowed randomness. Can you do better?
- Here is a streaming algorithm:
  - Start with $k = 0$.
  - On every occurrence of the given element increment the counter with probability $1/2^k$.
    (*This is to keep the value of* k *so that* 2^k *is approximately the count.*)
  - At the end of the stream, output $2^k - 1$.

Problem

Design an algorithm for counting the number of occurrences of a given element in the stream.

- This can clearly be done using a deterministic algorithm that uses $O(\log n)$ space.
- Question: Can a deterministic algorithm do any better?
- Question: Suppose you are allowed some slack with respect to the answer (say constant factor) and allowed randomness. Can you do better?
- Here is a streaming algorithm:
  - Start with $k = 0$.
  - On every occurrence of the given element increment the counter with probability $1/2^k$.
    (*This is to keep the value of k so that 2^k is approximately the count.*)
  - At the end of the stream, output $2^k - 1$.
- Claim: The above streaming algorithm uses $O(\log \log n)$ space and in expectation outputs an answer within a factor 2 of the correct count.

Problem

Design an algorithm for finding the majority element (in case there exists one).

- We want a time the element that appears in the stream more than $n/2$ times.
- Claim Any deterministic algorithm requires $\Omega(\min(n, m))$ space.
- We can do better if we relax our requirement in the following manner:
  - In case there is a majority element, then the algorithm should output this element.
  - In case there is no majority element, the algorithm is allowed to output any element.

Problem

Design an algorithm for finding the majority element (in case there exists one).

- Claim Any deterministic algorithm requires $\Omega(\min(n, m))$ space.
- We can do better if we relax our requirement in the following manner:
  - In case there is a majority element, then the algorithm should output this element.
  - In case there is no majority element, the algorithm is allowed to output any element.

Algorithm

Majority($a_1, ..., a_n$)
- $s \leftarrow a_1$ and $ctr \leftarrow 1$
- For $i = 2$ to n
  - if $(a_i = s) ctr \leftarrow ctr + 1$
  - elseif $(ctr \neq 0)$ $ctr \leftarrow ctr - 1$
  - else $\{s \leftarrow a_i; ctr \leftarrow 1\}$
- return(s)

End