GEMS OF TCS

STREAMING ALGORITHMS

Sasha Golovnev
February 4, 2021
STREAMING ALGORITHMS

• Massively long stream of data
STREAMING ALGORITHMS

- Massively long stream of data
  Instagram, search queries, network packets
STREAMING ALGORITHMS

• Massively long stream of data
  Instagram, search queries, network packets
  \(x_1, x_2, x_3, \ldots, x_n\)
STREAMING ALGORITHMS

• Massively long stream of data
  Instagram, search queries, network packets
  $x_1, x_2, x_3, \ldots, x_n$

• Data has grown: we can’t afford even stoing it
STREAMING ALGORITHMS

- Massively long stream of data
  Instagram, search queries, network packets
  \( x_1, x_2, x_3, \ldots, x_n \)
- Data has grown: we can’t afford even stoing it
- \( n \) inputs, space \( \sqrt{n}; \log^{10} n; \log n \)
Streaming Algorithms

- Massively long stream of data
  Instagram, search queries, network packets
  \( x_1, x_2, x_3, \ldots, x_n \)

- Data has grown: we can’t afford even storing it

- \( n \) inputs, space \( \sqrt{n} \); \( \log_{10} n \); \( \log n \)

- Efficient processing of stream
Streaming Algorithms

- Massively long stream of data
  Instagram, search queries, network packets
  \( x_1, x_2, x_3, \ldots, x_n \)

- Data has grown: we can’t afford even storing it

- \( n \) inputs, space \( \sqrt{n}; \quad \log^{10} n; \quad \log n \)

- Efficient processing of stream

- Mostly randomized algorithms
Missing Number
Missing Number

- Stream contains $n$ distinct numbers in range $\{0, \ldots, n\}$ all but one
**Missing Number**

- Stream contains $n$ distinct numbers in range $\{0, \ldots, n\}$

- Return the only missing number

  Could sort — linear space, go through stream many times
**Missing Number**

- Stream contains $n$ distinct numbers in range \{0, \ldots, n\}

- Return the only missing number

- Efficient algorithm?
Streaming Algorithm

- Compute sum of all elements in stream:

\[ S = x_1 + \ldots x_n \]
Streaming Algorithm

- Compute sum of all elements in stream:
  \[ S = x_1 + \ldots x_n \]

- Sum of all numbers in range \( \{0, \ldots, n\} \) is
  \[ S = \frac{n(n+1)}{2} \]
STREAMING ALGORITHM

- Compute sum of all elements in stream:
  \[ S = x_1 + \ldots + x_n \]

- Sum of all numbers in range \( \{0, \ldots, n\} \) is
  \[ S = \frac{n(n+1)}{2} \]

- Missing number is \( S - s = \frac{n(n+1)}{2} - s \)

See element \( \rightarrow \) process quickly - one addition \( O(\log n) \)
Streaming Algorithm

- Compute sum of all elements in stream:
  \[ S = x_1 + \ldots x_n \]

- Sum of all numbers in range \{0, \ldots, n\} is
  \[ S = \frac{n(n+1)}{2} \]

- Missing number is \( S - s = \frac{n(n+1)}{2} - s \)

- One pass through stream, efficient processing, \( O(\log n) \) space
Two Missing Elements

- Stream contains $n - 1$ distinct numbers in range $\{0, \ldots, n\}$
Two Missing Elements

- Stream contains $n - 1$ distinct numbers in range $\{0, \ldots, n\}$

- Return both missing numbers
Two Missing Elements

- Stream contains \( n - 1 \) distinct numbers in range \( \{0, \ldots, n\} \)

- Return both missing numbers

- Efficient algorithm?

\[
\begin{align*}
S & = x_1 + \ldots + x_{n-1} \\
S & = 0 + 1 + \ldots + n \\
S - s & = a + b \\
\text{Don't want to sort}
\end{align*}
\]
Streaming Algorithm

- Compute **sum and sum of squares** of all elements in stream:

  \[
  s = x_1 + \ldots + x_{n-1} \\
  t = x_1^2 + \ldots + x_{n-1}^2
  \]
Streaming Algorithm

- Compute **sum and sum of squares** of all elements in stream:

\[ s = x_1 + \ldots + x_{n-1} \]
\[ t = x_1^2 + \ldots + x_{n-1}^2 \]

- Sum of all numbers in range \( \{0, \ldots, n\} \) is

\[ S = \frac{n(n+1)}{2} \]

Sum of squares of all numbers in range \( \{0, \ldots, n\} \) is

\[ T = \frac{n(n+1)(2n+1)}{6} \]

\[ T = \sum_{i=0}^{n} i^2 \]
If missing numbers are $a$ and $b$, then

\begin{align*}
    U &= a + b = S - s \\
    V &= a^2 + b^2 = T - t \\

    W &= \frac{U^2 - V}{2} = \frac{(a+b)^2 - a^2 - b^2}{2} = \frac{2ab}{2} = ab
\end{align*}
\[ u = a + b \]
\[ w = a \cdot b \]
\[ a = u - b \]

\[ w = (u - b) \cdot b \]

\[ b^2 - ub + w = 0 \]

\[ \Delta = u^2 - 4w \]

\[ b = \frac{u \pm \sqrt{u^2 - 4w}}{2} \]

Two solutions are the two missing els.
**Streaming Algorithm**

- If missing numbers are $a$ and $b$, then
  
  \[
  S = x_1 + \ldots + x_n \\
  T = x_1^2 + \ldots + x_n^2 \\
  a + b = S - s \\
  a^2 + b^2 = T - t
  \]

- This can be generalized.

  - One pass through stream, efficient processing, $O(\log n)$ space

- Keys are missing.
Majority Element
MAJORITY ELEMENT

\( n \) - length of stream

- Stream has element occurring > \( n/2 \) times
MAJORITY ELEMENT

- Stream has element occurring $> n/2$ times

- Find it!
  
  Sort, Median
  We can't afford storing insert
Streaming Algorithm

- $\text{count} \leftarrow 0$; $m \leftarrow \perp \text{Null}$
Streaming Algorithm

- count $\leftarrow 0$; $m \leftarrow \perp$

- For each element $x_i$ of Stream:
**STREAMING ALGORITHM**

- $\text{count} \leftarrow 0; \quad m \leftarrow \perp$

- For each element $x_i$ of Stream:
  - If $\text{count} = 0$, then $m \leftarrow x_i$ and $\text{count} \leftarrow 1$
Streaming Algorithm

- count ← 0; m ← ⊥

- For each element $x_i$ of Stream:
  - If count = 0, then m ← $x_i$ and count ← 1
  - Elself $x_i = m$, then count ++
Streaming Algorithm

- count ← 0; m ← ⊥

- For each element $x_i$ of Stream:
  - If count = 0, then m ← $x_i$ and $\text{count} < 1$
  - Elseif $x_i = m$, then count ++
  - Else count -- $x_i \neq m$
STREAMING ALGORITHM

count ← 0; m ← ⊥

For each element $x_i$ of Stream:
  - If count = 0, then $m \leftarrow x_i$ and
  - Elself $x_i = m$, then count ++
  - Else count ← $x_i + m$

Return m
Example

\[ n = 7 \quad MoS = 2 \]

\[
\begin{array}{c}
m \leftarrow X \\
\text{count} \leftarrow 0 \\
1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \\
\end{array}
\]

Return \ 2

\[
\begin{array}{c}
\text{count} \\
\end{array}
\]

\[
\begin{array}{c}
m = 1 \\
m = 2 \\
m = 3 \\
m = 2 \\
m = 3 \\
m = 2 \\
m = 2 \\
\end{array}
\]

\[
\begin{array}{c}
\text{count} \geq 0 \\
\end{array}
\]
\( \text{PROOF} \)

\( \text{variable count}_1 \)

\( \text{this is only for analysis not for algorithm} \)

\( \text{if m = Maj} \quad \text{time Maj elt} \)

\[ \text{count}_1 = \begin{cases} \text{count} & \text{if m = Maj} \\ \text{count} & \text{if m \neq Maj} \\ -\text{count} & \text{if m = Maj} \\ -\text{count} & \text{if m \neq Maj} \end{cases} \]

\( \text{When I see Maj, increment count}_1 \)

\( \text{Proof:} \)

\( \text{if m = Maj} \Rightarrow \text{count}_1++, \quad \text{count}_1++ \)

\( \text{if m \neq Maj} \Rightarrow \text{count}_1--; \quad \text{count}_1++ \)

\( \text{See Maj > } \frac{n}{2} \text{ times} \Rightarrow \text{count}_1 \) is incremented > \( \frac{n}{2} \)

\( \Rightarrow \text{count}_1 \) is decremented < \( \frac{n}{2} \)

\( \text{In the end,} \)

\( \sqrt{\text{count}_1 > 0} \Rightarrow \text{count}_1 = \text{count}_1 = \text{count} \Rightarrow m = \text{Maj} \Rightarrow \text{output Maj} \)
- Pains up distinct els
- Kills all these pains
- Remaining els are Maj

\[ \frac{n}{2} + 1 \quad \text{and} \quad n - 1 \]

Majority els remain
Assume there is Maj in stream ($\geq \frac{n}{2}$ occurrences),

Without this assumption, we'll make two passes through input stream.

1. Candidate $m$ =

2. Count how many times $m$ appears in stream.

For KEN, $k$-Heavy Hitters: Find all els that appear $\geq \frac{n}{k}$ in the stream ($\leq k$ such els)

$\text{Maj} = \text{case } k=2$
**Misra-Gries Algorithm**

- \( \text{count}_1, \ldots, \text{count}_k \leftarrow 0; \ m_1, \ldots, m_k \leftarrow \perp \)

- For each element \( x_i \) of Stream:
  - If \( x_i = m_j \), then \( \text{count}_j \leftarrow + \)
  - Else
    - Let \( \text{count}_j \) be min in \( \text{count}_1, \ldots, \text{count}_k \)
    - If \( \text{count}_j = 0 \), then \( m_j = x_i; \ \text{count}_j = 1 \)
    - Else \( \text{count}_1 \leftarrow -, \ldots, \text{count}_k \leftarrow - \)

- Return \( m_1, \ldots, m_k \) contain all els of stream that appear \( \geq k \times \text{times} \)
Approximate Counting
• Router receives stream of network packages
• Router receives stream of network packages

• Want to count number of packages from IP “1.2.3.4”

EQ: \[ \leq n \text{ of them in the stream} \]

\[ \text{output length of the stream} \]
• Router receives stream of network packages

• Want to count number of packages from IP “1.2.3.4”

• Efficient algorithm?

\[
\text{count} = 0
\]

See input: \( \text{count} = \text{count} + 1 \)

In the end, \( \text{count} = \text{stream length} \)

\( \log_2(n+1) \) bits
Can we use fewer than login bits?

Input length \( \leq n \)

outputs \( \in \{0, 1, 2, \ldots, n-1, n^2\} \)

2 bits

\[
\begin{array}{c|c}
00 & 01 \\
10 & 11 \\
\end{array}
\]

\( \leq 4 \) distinct answers

IF \( \log_2 \) login bits => different answers

\[ \leq 2^{\log_2 n} = n \]

\( \log_2 (n+1) \) bits is optimal
• Router receives stream of network packages

• Want to count number of packages from IP “1.2.3.4”

• Efficient algorithm?

• Efficient approximate algorithm?

\[ \log \log n < \text{exponentially better than previous sol} \]
n

Trivial alg stones n

\[ n = 147 \quad 10010011 \] \[ \log n \]

What if instead of stoning \( n \) in binary, I'm stoning the length of "n in binary"? Instead of stoning 147, I'd stone numbers

Instead of stoning 147, I'd stone numbers

\[ 1000 \]

Instead of \( \log \) facts to write \( f_1, \ldots, n \)
stone log\( \log_2 \) bits to write \( f_1, \ldots, \log n \)

If length = 4
\[ \begin{align*}
8 & \leq n \leq 15 \\
2^{\text{length}-1} & \leq n < 2^{\text{length}}
\end{align*} \]
MORRIS ALGORITHM

\[ C = \text{length of } n \text{ in binary} \]

\[ n \approx 2^C \]

When should I want to increment \( C \)?

I want to increment \( C \) often seeing \( 2^C \) new els

Now see new el

w. p. \( \frac{1}{2^C} \)  \( C++ \)

w. p. \( (1 - \frac{1}{2^C}) \) don't update \( C \)

After seeing \( 2^C \) els, I expect to increment \( C \) once
MORRIS ALGORITHM

\[
\begin{align*}
\text{n} &= 0 \\
2^c - 1 &\approx n
\end{align*}
\]

- \( c \leftarrow 0 \)
MORRIS ALGORITHM

• $c \gets 0$

• When see next element:
  • with probability $\frac{1}{2^c}$ increment $c$
  • with probability $1 - \frac{1}{2^c}$ do nothing
**MORRIS ALGORITHM**

\[ n \approx 2^c - 1 \]

- \( c \leftarrow 0 \)
- When see next element:
  - with probability \( \frac{1}{2^c} \) increment \( c \)
  - with probability \( 1 - \frac{1}{2^c} \) do nothing
- Return \( 2^c - 1 \)
**Probability of Success**

- **Thm**
  \[ \mathbb{E}[\text{output}] = n \]

- **Markov's**
  \[ \mathbb{E}[\text{output} \geq 5n] < \frac{1}{5} \]

  1. **Markov's ineq:** \( \Pr[\text{output} \notin [n-0(n), n+0(n)]] < 0.9 \)

  \[ \implies \]

  2. **Amplify prob. of success by repeating this alg several times:**

     \[ \Pr[\text{output} \notin \left[ \frac{n}{2}, 2n \right]] < 0.01 \]
PROBABILITY OF SUCCESS

- $\mathbb{E}[\text{output}] = n$

- By Markov’s, $\Pr[\text{output} \geq 2n] \leq 1/2$
PROBABILITY OF SUCCESS

• $\mathbb{E}[\text{output}] = n$

• By Markov’s, $\Pr[\text{output} \geq 2n] \leq 1/2$

• Similar inequalities show that $\Pr[\text{output} \in [n - O(n), n + O(n)] \geq 0.9$
PROBABILITY OF SUCCESS

• \( \mathbb{E}[\text{output}] = n \)

• By Markov’s, \( \Pr[\text{output} \geq 2n] \leq 1/2 \)

• Similar inequalities show that
  \( \Pr[\text{output} \in [n - O(n), n + O(n)] \geq 0.9 \)

• Again, repeating Algorithm several times significantly amplifies probability of success
SUMMARY

- One pass through stream may be sufficient
SUMMARY

• One pass through stream may be sufficient

• Use Randomness and Approximation
SUMMARY

• One pass through stream may be sufficient

• Use Randomness and Approximation

• Markov’s inequality: from Expectation to Probability
SUMMARY

• One pass through stream may be sufficient

• Use Randomness and Approximation

• Markov’s inequality: from Expectation to Probability

• Amplify probability by Repetitions