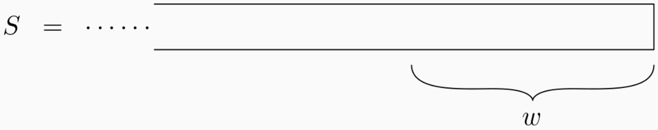
Sliding Window String Indexing in Streams

Philip Bille
Inge Li Gørtz
Johannes Fischer
Max Rishøj Pedersen
Tord Joakim Stordalen

Problem Statement

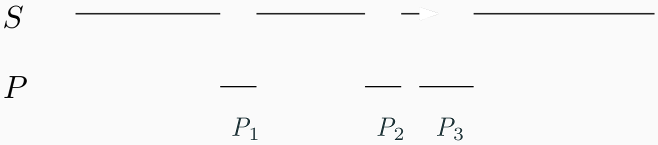
- Maintain an index over the w most recent characters of a stream



- At any point, given a pattern string P, find all occurrences in the window.

Problem Statement

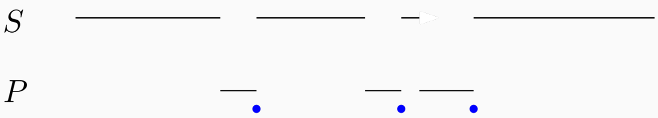
- Both S and patterns P are streamed one character at a time.
- No interleaving:



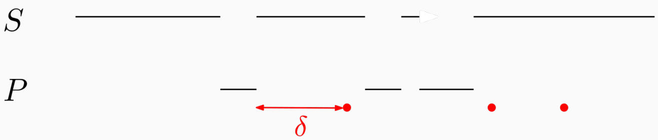
- We do not know the length of each pattern up front.
- The goal is to use the least amount of time ***per character***.

Problem Statement

- The *timely* variant:



- The δ-*delayed* variant:



Previous Work

- Sliding Window Suffix Trees:
  $O(1)$ amortized, worst-case $\Omega(w)$. For *constant-sized alphabets*
  Brodnik & Jekovec 2018.

- Fully Dynamic Suffix Arrays:
  Polylogarithmic time per operation and more general
  Kempa & Kociumaka 2022.

- Online Suffix Tree Construction:
  $O(\log \log n + \log \log |\Sigma|)$ per character
  Kopelowitz 2012.

Our Results

- Timely: $O(w)$ space, $O(\log w)$ time per character whp
- Delayed: $O(w + \delta)$ space, $O(\log(w/\delta))$ time per character whp
- For $\delta = \epsilon w$ we get $O(w)$ space and constant time whp

Agenda

- Simple Solution
- Streaming Patterns
- With Delay
- High-probability Guarantees

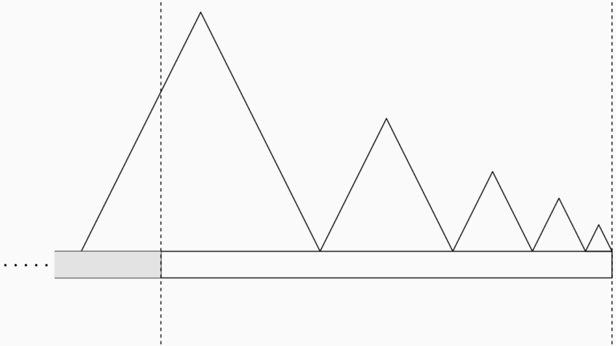
Simple Solution

Simple Solution

- We know the pattern (and its length) upfront
- The timely variant
- Expected running times

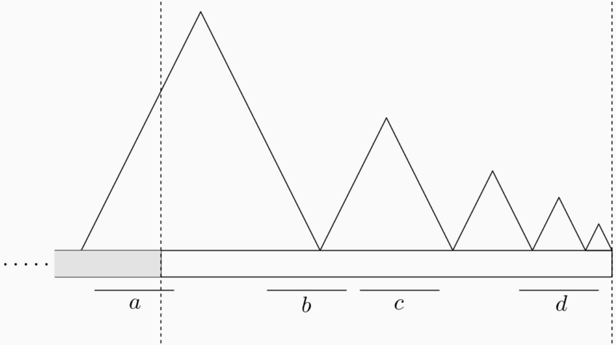
Simple Solution

- We maintain at most $\log w$ suffix trees that cover the window.



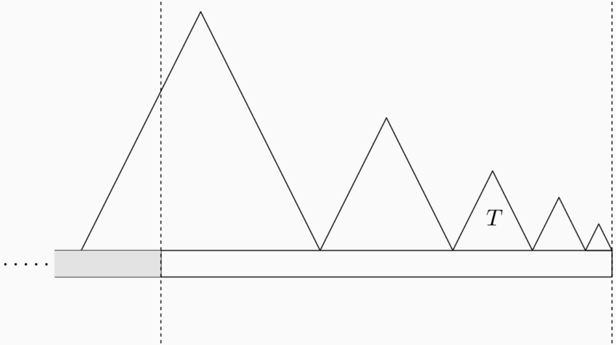
Simple Solution

- We maintain at most $\log w$ suffix trees that cover the window.



Simple Solution

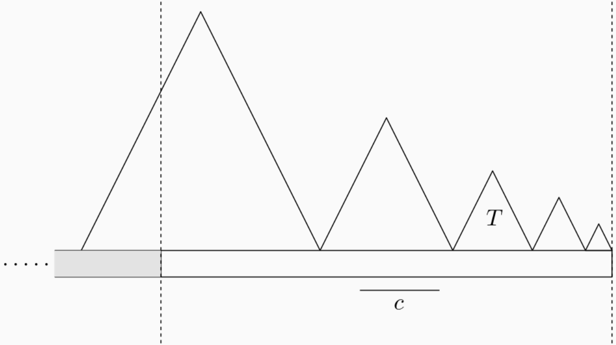
- We maintain at most $\log w$ suffix trees that cover the window.



- T is the smallest tree larger than $m = |P|$

Simple Solution

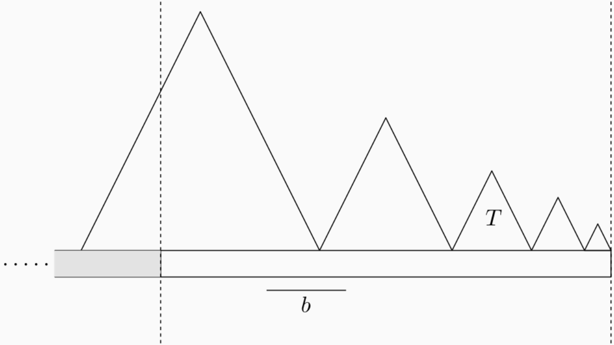
- We maintain at most $\log w$ suffix trees that cover the window.



- T is the smallest tree larger than $m = |P|$
- $O(m \log w)$

Simple Solution

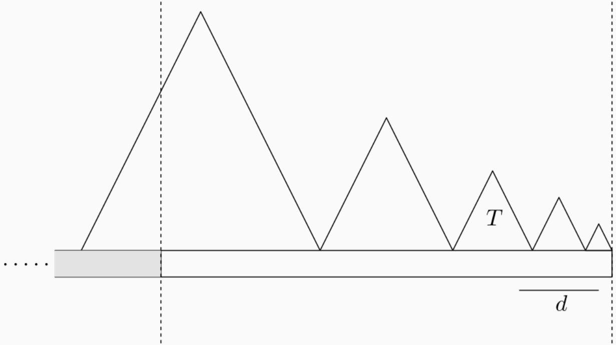
- We maintain at most $\log w$ suffix trees that cover the window.



- T is the smallest tree larger than $m = |P|$
- $O(m \log w) + O(m \log w)$

Simple Solution

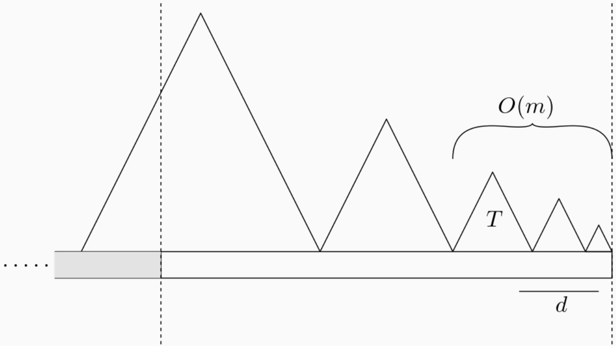
- We maintain at most $\log w$ suffix trees that cover the window.



- T is the smallest tree larger than $m = |P|$
- $O(m \log w) + O(m \log w)$

Simple Solution

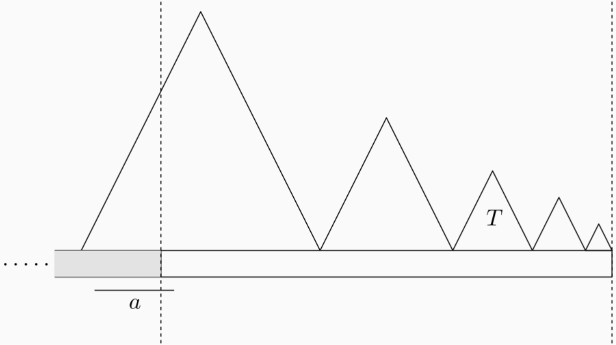
- We maintain at most $\log w$ suffix trees that cover the window.



- T is the smallest tree larger than $m = |P|$
- $O(m \log w) + O(m \log w) + O(m)$

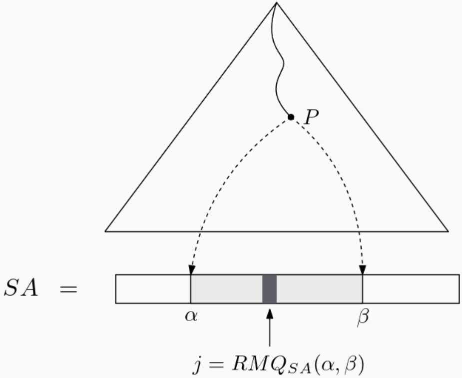
Simple Solution

- We maintain at most $\log w$ suffix trees that cover the window.



- T is the smallest tree larger than $m = |P|$
- $O(m \log w) + O(m \log w) + O(m)$

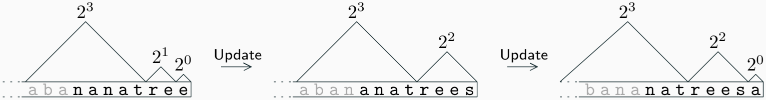
Simple Solution



- Recurse on $[\alpha, j-1]$ and $[j+1, \beta]$
- RMQ in linear space, constant time; reporting in $O(\text{occ})$ time

Maintaining the Data Structure

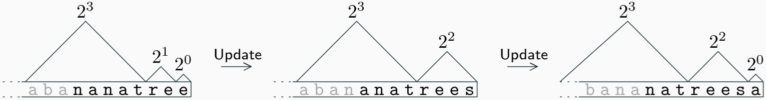
- Log-structured merge!



- Each character is included in at most $\log w$ suffix trees

Maintaining the Data Structure

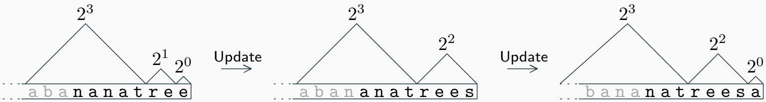
- Log-structured merge!



- Each character is included in at most $\log w$ suffix trees
- Expected linear time construction gives expected amortized $O(\log w)$ time per update

Maintaining the Data Structure

- Log-structured merge!

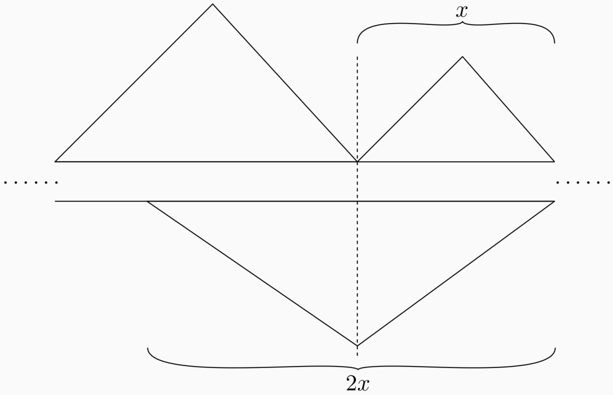


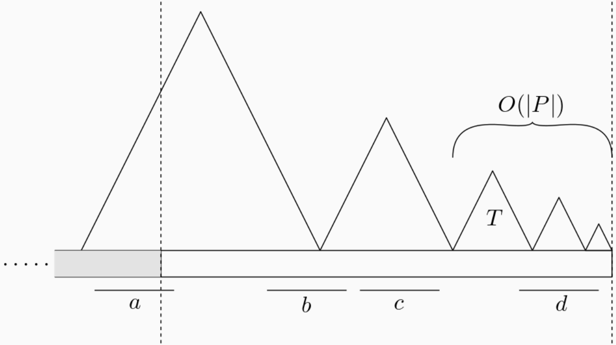
- Each character is included in at most $\log w$ suffix trees
- Expected linear time construction gives expected amortized $O(\log w)$ time per update
- Deamortize by keeping both trees and merging in the background

Streaming the Pattern

Streaming the Pattern

- We cannot use KMP since we do not know the pattern upfront
- We instead add suffix trees across the boundaries

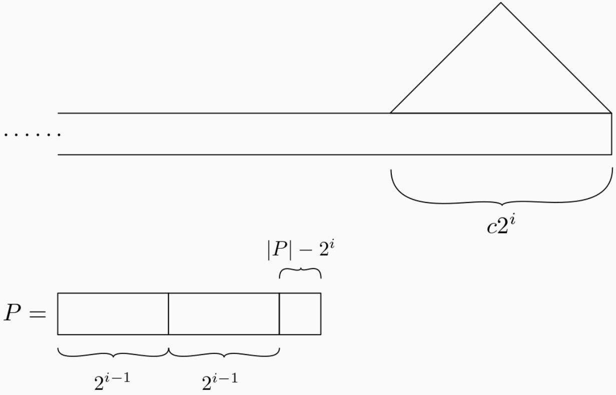




- Boundary trees do not work to the right of T
- We grow a suffix tree at query time!

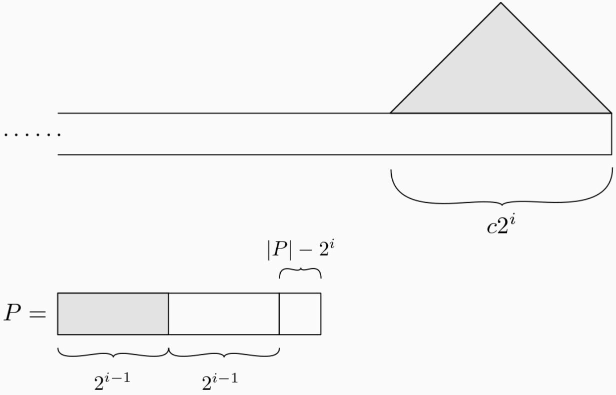
Streaming the Pattern

- Suppose that $2^i \leq |P| < 2^{i+1}$.



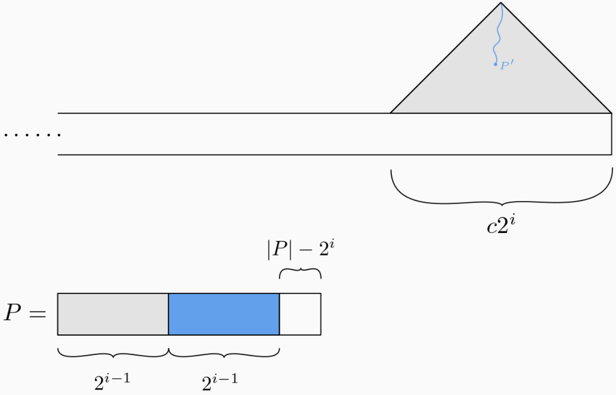
Streaming the Pattern

- Suppose that $2^i \leq |P| < 2^{i+1}$.



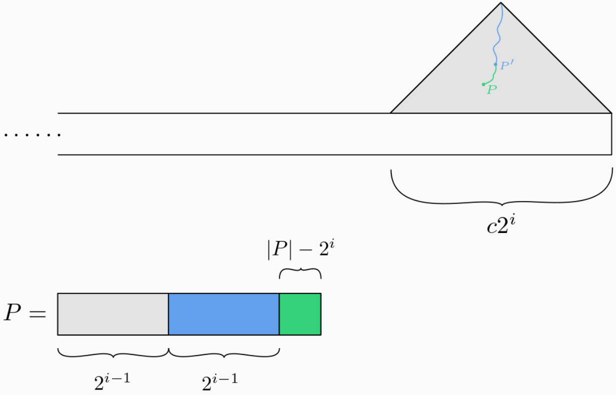
Streaming the Pattern

- Suppose that $2^i \leq |P| < 2^{i+1}$.



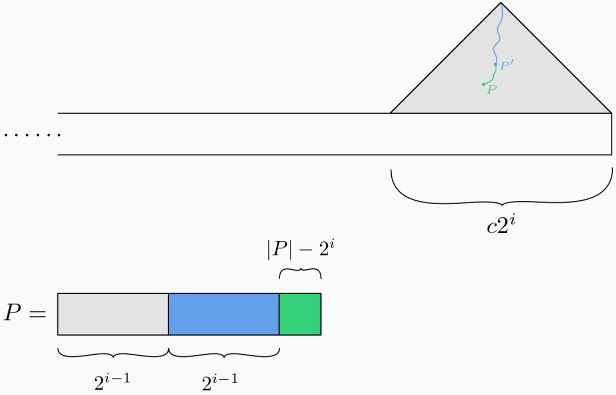
Streaming the Pattern

- Suppose that $2^i \leq |P| < 2^{i+1}$.



Streaming the Pattern

- Suppose that $2^i \leq |P| < 2^{i+1}$.

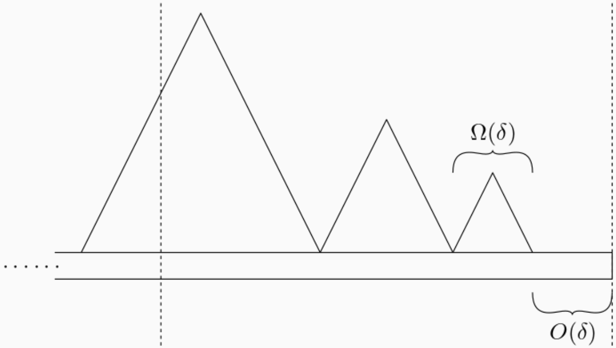


- Run the algorithm for each of the $\log w$ choices of i

Introducing Delay

Data Structure

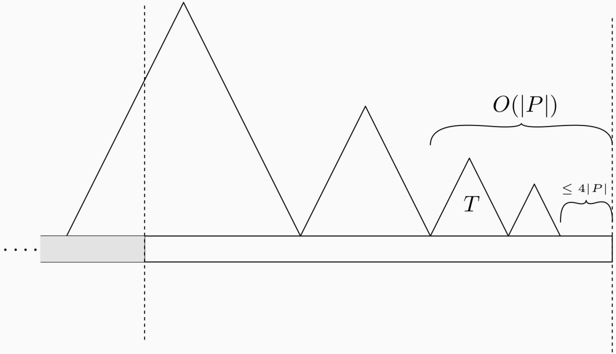
- We leave a suffix of length $O(\delta)$ uncovered by suffix trees



- The smallest tree has size $\Omega(\delta)$. There are $\approx \log w - \log \delta = \log(w/\delta)$ trees

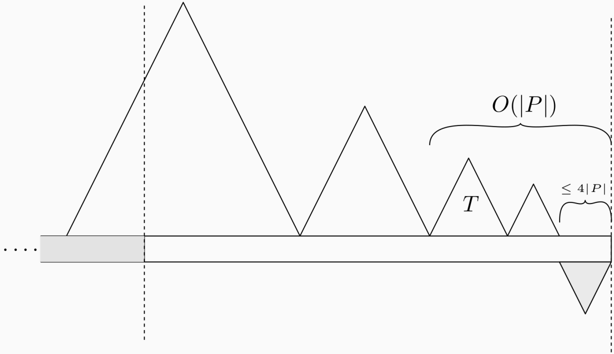
Queries

- Long patterns: $|P| > \delta/4$. Queries remain the same! (no delay)



Queries

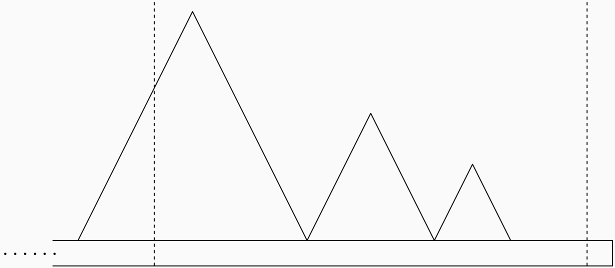
- Long patterns: $|P| > \delta/4$. Queries remain the same! (no delay)



- Short patterns: buffer queries and updates until we can afford to construct suffix tree in $O(\delta)$ time

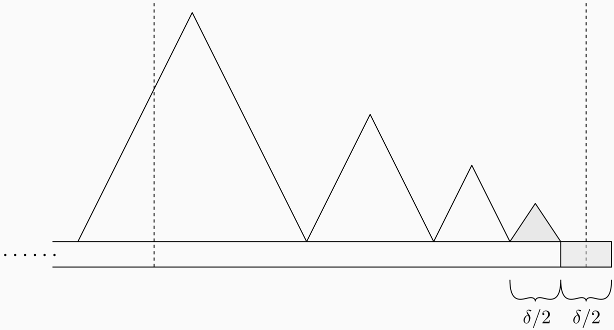
Construction

- Construct trees of size $\delta/2$



Construction

- Construct trees of size $\delta/2$



- Each character included in $O(\log(w/\delta))$ suffix trees
- Deamortized in the same way as before

High-probability Guarantees

High-probability Guarantees

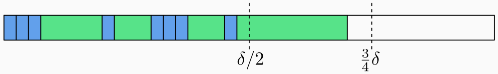
- Given a string of length n over alphabet Σ
- Expected $O(n)$ time: pick a hash function $\Sigma \to [0, n^2]$ + radix sort
- With high probability: pick a hash function $\Sigma \to [0, n^d]$
- We have trees of many sizes, e.g., $\log w$. We allocate arrays of size w for small cases.

Questions?

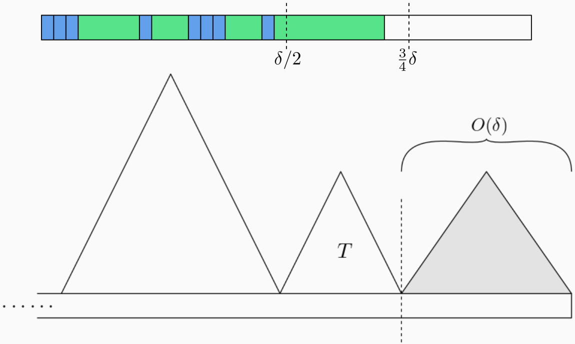
Queries: Short Patterns

- Short patterns: $|P| \leq \delta/4$
- Challenge: the uncovered suffix may be much larger than P
- Solution: buffer queries and updates until we can afford to do $\Omega(\delta)$ work
- Buffer of size δ. Add each update and each pattern to the buffer.
- *Flush* when there are at least $\delta/2$ characters in the buffer, deamortized over $\delta/4$ characters.

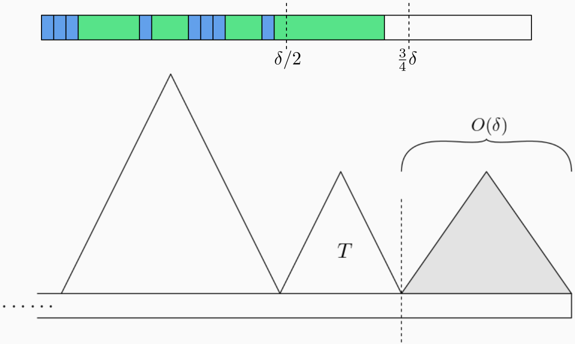
Flushing the Buffer



Flushing the Buffer



Flushing the Buffer



- Match in each (boundary) tree
- Build and match in suffix tree over uncovered suffix
- Use KMP across the boundary: $O(\delta)$ time
- Process updates
- Total: $O(\delta \log(w/\delta))$