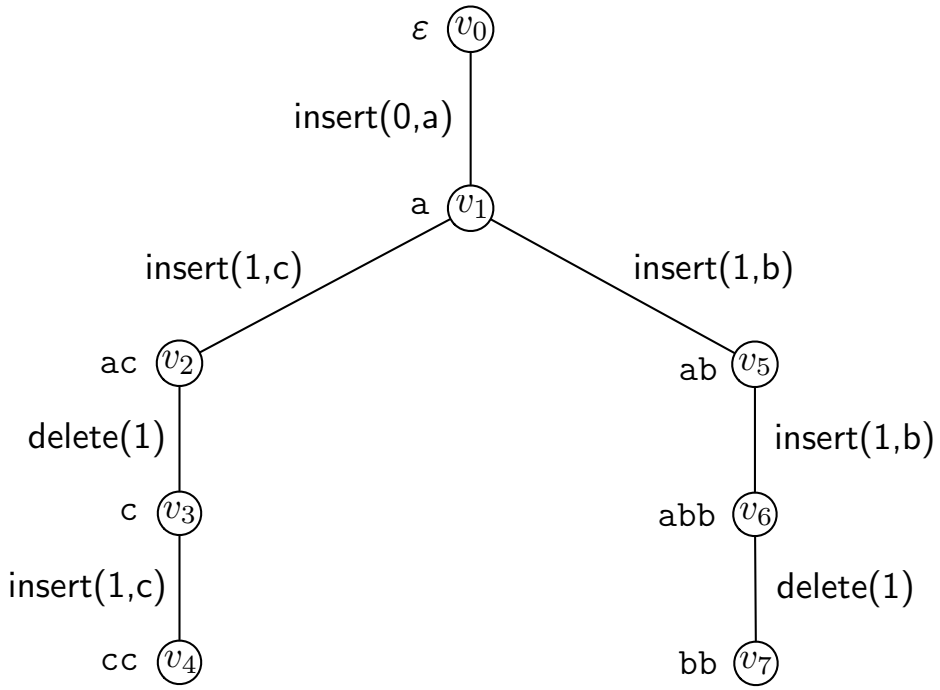
Random Access in Persistent Strings

Philip Bille
Inge Li Gørtz

Random Access in Compressed Data

- What is the ith character?
- What is the substring at [i,j]?

Persistent Strings



N = 13

n = 8

- **Random access in persistent strings problem:** Compactly represent a version tree with n nodes to support
  - access(v, j): return S(v)[j]

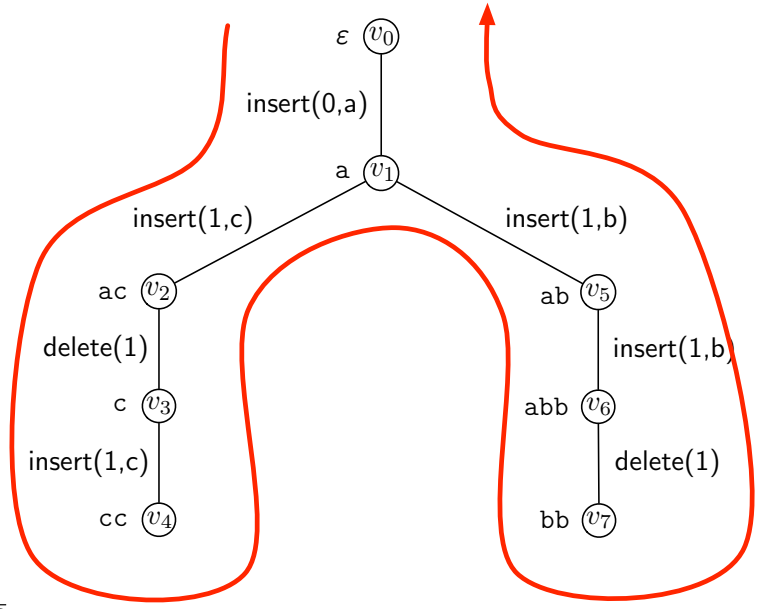
Applications

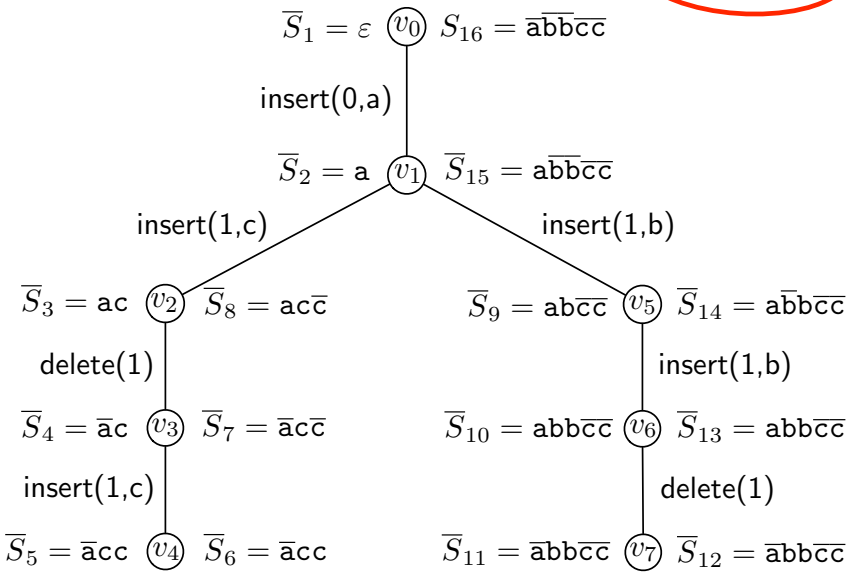
- Version control systems.
- Highly-repetitive collections.

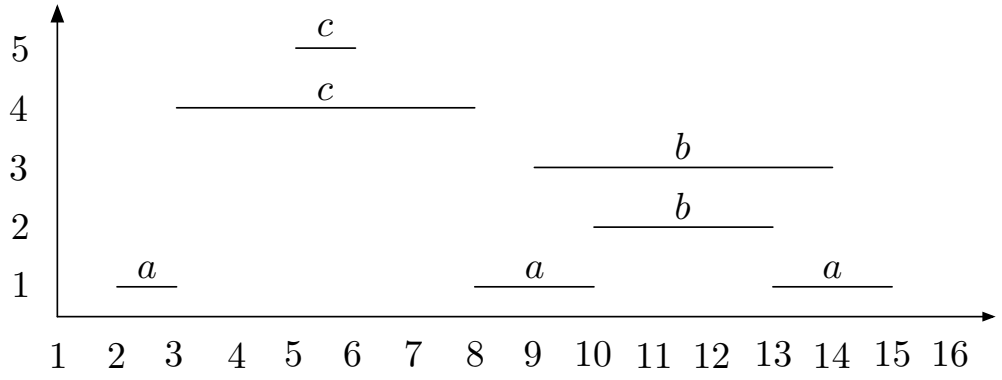
Results

time	space	notes
O(1)	O(N)	Explicit string representation
O(log n)	O(n)	Fully persistent BST
O(log n)	O(n log n)	LZ compression + random access
O(log n/log log n)	$O(n \log^{1+\varepsilon} n)$	LZ compression + random access
O(log n/log log n)	O(n)	**new**
Ω(log n/log log n)	$n \log^{O(1)} n$	**new**

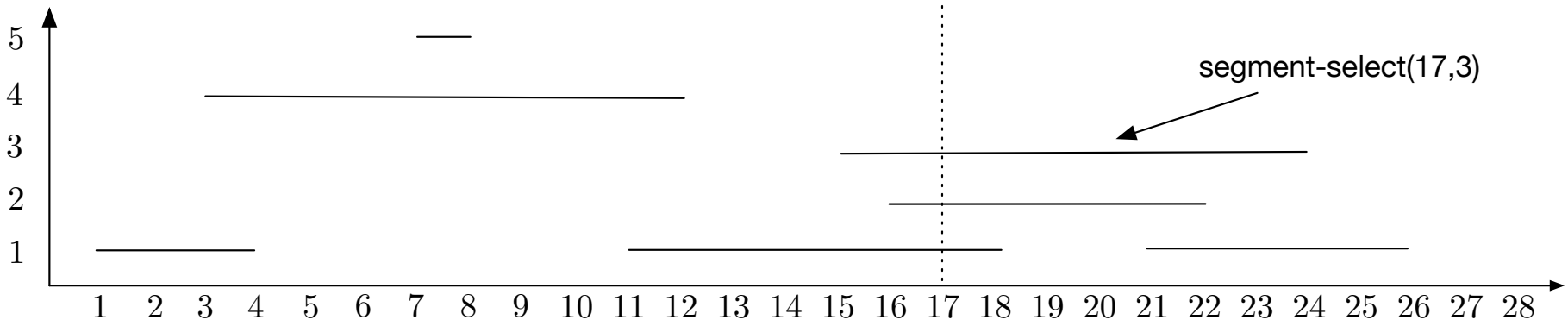
Random Access to Segment Selection





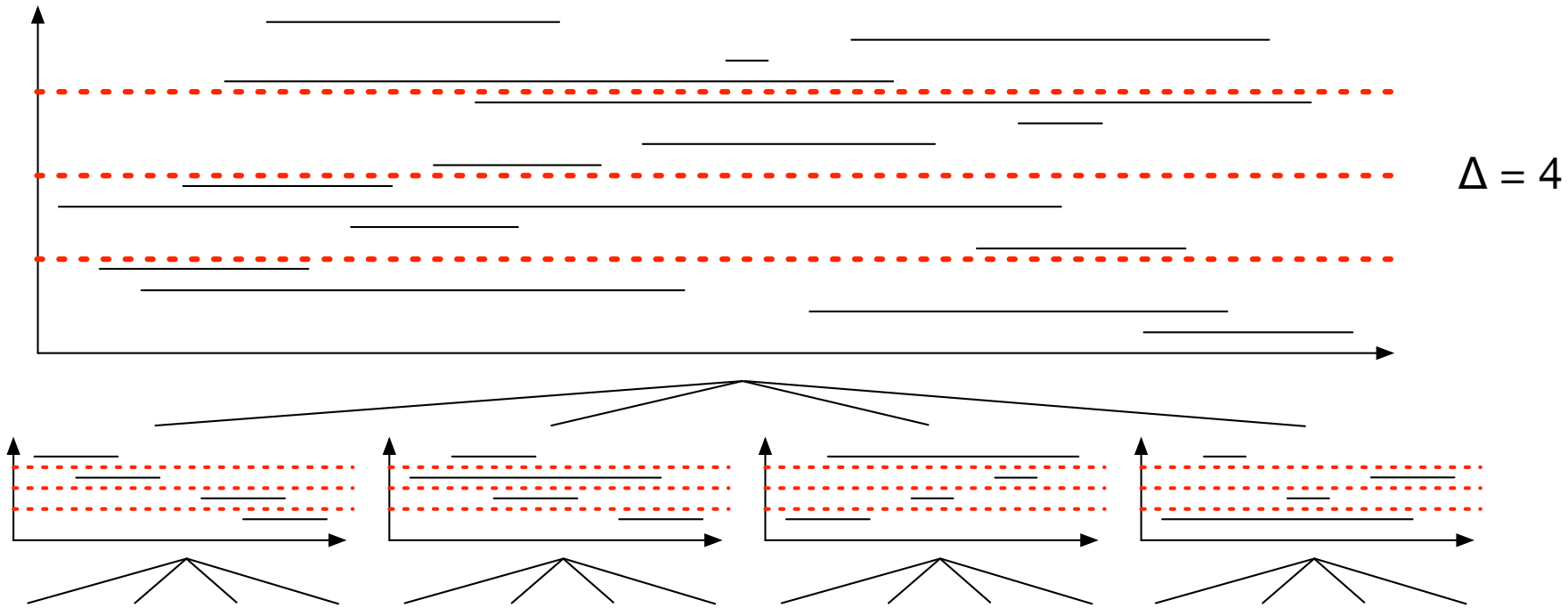


Segment Selection



- **Segment selection problem:** Compactly represent collection of n horizontal line segments to support
  - segment-select(i,j): return the jth smallest segment crossing the vertical line through x-coordinate i.

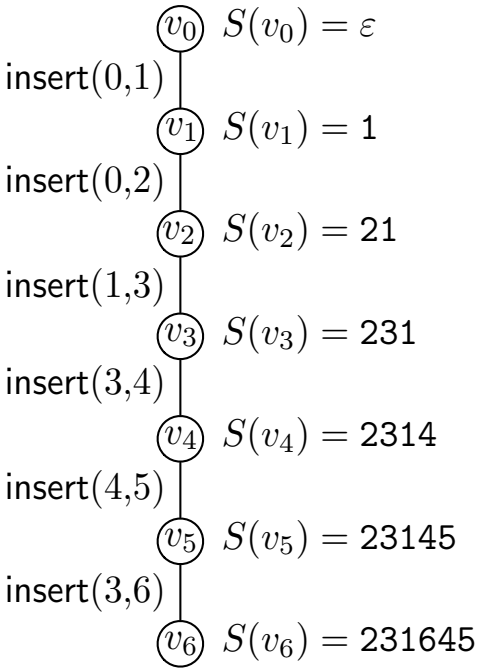
Segment Selection



- Recursively partition segments into $\Delta = \log^{\varepsilon} n$ horizontal slabs with the same number of segments.
- Segment-select query via top-down traversal.
- **Goal:**
  - O(1) time per node $\Longrightarrow$ $O(\log_{\Delta} n) = O(\log n / \log \log n)$
  - O(n log log n) *bits* per node $\Longrightarrow$ $O(n \log_{\Delta} n \log \log n) = O(n \log n)$ bits

Lower Bound

$A = [3, 1, 2, 5, 6, 4]$



- **Prefix selection problem:** Compactly represent an array A of n integers, to support
  - prefix-select(i,j): return the jth smallest integer in subarray A[1,i].
- **Lemma** [JL2011]. Any data structure for prefix selection that uses n $\log^{O(1)}$ n space needs Ω(log n/ log log n) time.
- Construct version path from prefix selection instance such that S(v_i) represents sorted order of A[1..i].

Results

time	space	notes
O(1)	O(N)	Explicit string representation
O(log n)	O(n)	Fully persistent BST
O(log n)	O(n log n)	LZ compression + random access
O(log n/log log n)	$O(n \log^{1+\varepsilon} n)$	LZ compression + random access
O(log n/log log n)	O(n)	**new**
Ω(log n/log log n)	$n \log^{O(1)} n$	**new**

Open Problems

- Can we make solution dynamic?
- Faster preprocessing time?
- Pattern search?