The Wavelet Trie: Maintaining an Indexed Sequence of Strings in Compressed Space

CSI 5335 Paper presentation

Roberto Grossi, Giuseppe Ottaviano

presented by: Petr Praus

April 26th, 2012

Use Case

- A lot of things are string sequences.
- Column databases store and index string sequences.
- Great example: access logs

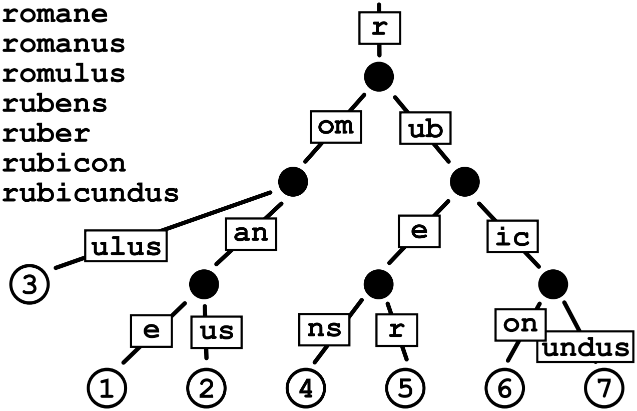
```
188.26.52.117 - - [24/Apr/2013:03:35:48 -0500] "GET /img/welcome/corner.png
188.26.52.117 - - [24/Apr/2013:03:35:49 -0500] "GET /img/welcome/arrowDown.gif
188.26.52.117 - - [24/Apr/2013:03:35:49 -0500] "GET /img/welcome/regionals.jpg
```

- Pretty similar, huh?
- I heard indexes make stuff faster → **indexed sequence of strings**
- Rank query: Number of requests for `/img/welcome/corner.png`?
- Select query: Position of *i*-th occurrence of `/img/welcome/corner.png`
- We can do prefix operations too

Patricia Trie

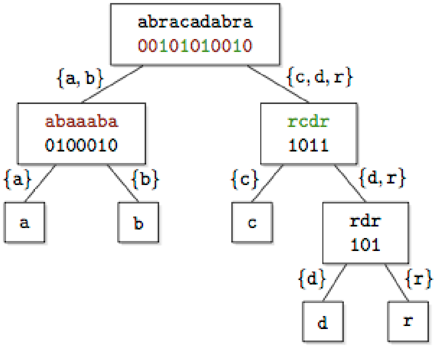
- **Trie** is ordered tree data structure used to store a *dynamic set*
- Space-efficient trie.
- Node always has at least two children.

1	romane
2	romanus
3	romulus
4	rubens
5	ruber
6	rubicon
7	rubicundus



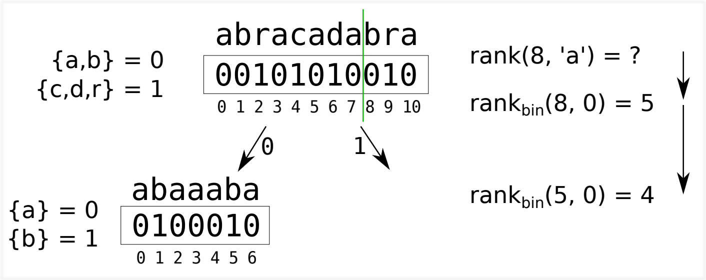
Wavelet Tree

- Organizes a string into a balanced binary tree of bit vectors.
- Alphabet $\sum = \{a, b, c, d, r\} \rightarrow \{0, 0, 1, 1, 1\}$
- At root, we have *ambiguity*, reducing ambiguity towards leaves
- Only the binary vectors are stored!



Efficient Computation of Rank in Wavelet Tree

- $rank(8, a)$ = how many a's before position 8
- $rank_{bin}(pos, s)$ binary rank, # of occurences of s before pos



E.g.: # of requests to `/img/welcome/corner.png` before April 10th.

Mutable & Compressed Indexed Sequences

- Sequences can change over time:
  $\mathit{Insert}(s, \mathit{pos})$, $\mathit{Append}(s)$, $\mathit{Delete}(\mathit{pos})$
- Alphabet not always known in advance

- Traditional approach, store explicitly (e.g. array), make an index
- Space-inefficient, we want to query the compressed representation
- **Succint data structure** – uses space close to lower information-theoretic bound

Wavelet Trie to rescue!

Wavelet Trie

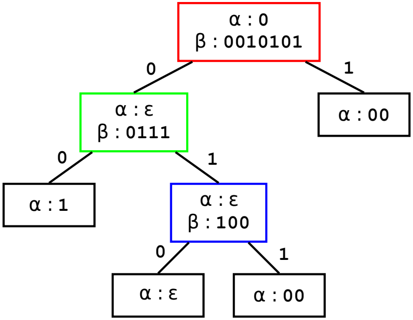
- Wavelet Tree + Patricia Trie
- Compressed data structure
- Can support *Insert*, *Append*, *Delete*, and dynamic alphabet
- Variants: Static, Append-only, Fully-dynamic
- Note $O(|s| + h_s)$ is very fast for balanced trees

	Query	Append	Insert	Delete	Space
Static	$O(\|s\| + h_s)$	–	–	–	$LB + o(\tilde{h}n)$
Append-only	$O(\|s\| + h_s)$	$O(\|s\| + h_s)$	–	–	$LB + PT + o(\tilde{h}n)$
Fully-dynamic	$O(\|s\| + h_s \log n)$	$O(\|s\| + h_s \log n)$	$O(\|s\| + h_s \log n)$	$O(\|s\| + h_s \log n)$	$LB + PT + O(nH_0)$

h_s – number of nodes traversed while searching for s is Patricia Tree
$\tilde{h}$ – average height of the Wavelet Trie
$|s|$ – length of query string
LB – lower bound on storing the unique elements of a sequence

Wavelet Trie - Example (Static)

- $\Sigma = \{0001, 0011, 0100, 00100\}$, α common prefix, β bit-vector
- Splitting based on common prefix like in Patricia Trie, not halving an alphabet like in Wavelet Tree!

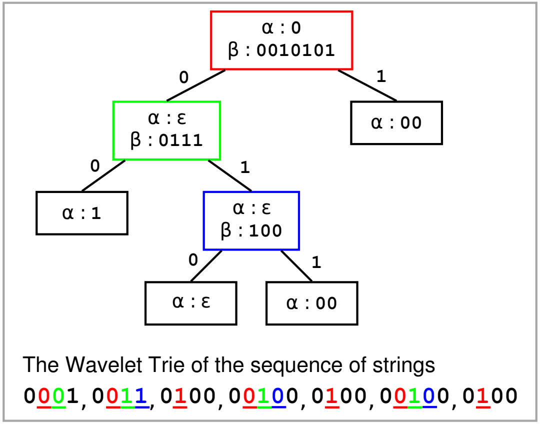


The Wavelet Trie of the sequence of strings

0001, 0011, 0100, 00100, 0100, 00100, 0100

Wavelet Trie - Example (Static)

- $Rank(6, 0\mathbf{1}00) \rightarrow Rank(6, 1) = 2$
- Notice how ambiguity decreases towards the leaves

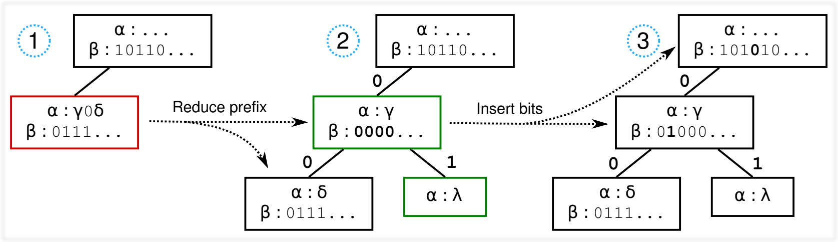


The Wavelet Trie of the sequence of strings

0001, 0011, 0100, 00100, 0100, 00100, 0100

Dynamic Wavelet Tries

- Bitvectors must support insertion and deletion
- We also want dynamic alphabet, e.g. we want to add "0111"



Inserting new $s = \ldots \gamma 1 \lambda$ at $pos = 3$; γ – common prefix, λ – suffix

1. Original Wavelet Trie
2. After splitting red node, adding internal node and leaf node (green)
3. Inserting bits in the root-to-leaf path nodes

Summary

- Wavelet Trie = Wavelet Tree + Patricia Trie
- Compressed sequences of strings
- Very space efficient (no need for data+index)
- Applications in column databases, access log aggregation, ...
- Quite fast, querying in $O(|s| + h_s)$
- Can support prefix searches

Thank you.

Resources

- Grossi, Roberto, and Giuseppe Ottaviano. “The wavelet trie: Maintaining an indexed sequence of strings in compressed space.” In Proceedings of the 31st symposium on Principles of Database Systems, pp. 203-214. ACM, 2012.
- `http://alexbowe.com/wavelet-trees/`
- `http://en.wikipedia.org/wiki/Wavelet_Tree`
- `http://siganakis.com/challenge-design-a-data-structure-thats-small`