Abstract

We study the problem of maintaining sequences of strings under insertion/deletion and indexed queries in compressed space. We introduce a new data structure, the Wavelet Trie, that supports efficient operations in space close to the information-theoretic lower bound.

Rank/Select Sequences

Let $S = \{s_0, \ldots, s_{n-1}\}$ sequence of symbols from alphabet $S_{\text{set}}$.

- Access($i$): access the $i$-th symbol $s_i$
- Rank($s$, pos): count the number of occurrences of $s$ before position pos
- Select($s$, idx): find the position of the idx-th occurrence of $s$

Can use Rank to count the number of occurrences of a symbol in an interval of the sequence, Select to iterate all the occurrences of a symbol.

Example: $S = \text{abracadabra}$, $S_{\text{set}} = \{a, b, c, d, r\}$.

- Access(0) = a
- Rank(a, 3) = 1
- Rank(a, 4) = 2
- Select(a, 2) = 5

“Easy” for the binary case, $S_{\text{set}} \subset \{0, 1\}^\ast$.

Dynamic sequences

A data structure for storing sequences is dynamic if it supports the following operations:

- Insert($s$, pos): insert the symbol $s$ immediately before $s_{\text{pos}}$
- Delete(pos): delete the symbol at position pos

We define Append as a special case of Insert:

- Append($s$): append the symbol $s$ at the end of the sequence

We call a data structure that only supports Append append-only.

Example scenarios: logging, time series, column-oriented databases, ...

Wavelet Trees

Wavelet Trees reduce queries on alphabet $S_{\text{set}}$ to queries on bitvectors. Example for $S_{\text{set}} = \{a, b, c, d, r\}$: $S = \text{abracadabra}$:

- Balanced tree built on $S_{\text{set}}$
- Each node splits the alphabet into two subsets
- At each node, sequence split into two subsequences
- At each node, Os in correspondence with left subset, 1s with right subset
- Support Access and Rank by performing Rank operations top-down on bitvectors, Select by bottom-up Select

By using dynamic bitvectors on the nodes, Insert and Delete can be supported, but the alphabet $S_{\text{set}}$ must be set a priori.

This limitation prevents the use of dynamic Wavelet Trees for large alphabets and database applications.

The Wavelet Trie

We consider the problem of sequences of binary strings, i.e. $S_{\text{set}} \subset \{0, 1\}^\ast$.

No loss of generality: non-binary strings, integers, ... can be binarized.

Example: $S = (0001, 0011, 00100, 0100, 00100, 0100)$.

- Rank(0100, 2) = 0
- Rank(0100, 3) = 1
- Select(0100, 2) = 6

We introduce the Wavelet Trie on $S$:

- Tree structure is the Patricia Trie on $S_{\text{set}}$. Each node corresponds to a subsequence with a common prefix
- $\alpha$ is the longest common prefix of the subsequence
- Each subsequence is partitioned based on the first bit after $\alpha$
- Bitvector $\beta$ discriminates between left and right subsequence
- Same operations as Wavelet Tree

New prefix operations

The Wavelet Trie enables two new operations:

- RankPrefix($p$, pos): count the strings prefixed by $p$ before position pos
- SelectPrefix($p$, idx): find the position of idx-th string prefixed by $p$

Example application: $S$ is a sequence of URLs, find the number of URLs from a given hostname in a given range, or enumerate them.

String set updating

The Patricia trie structure enables updates to the alphabet $S_{\text{set}}$. When an unseen string is inserted, an existing node is split. The new node is given a constant bitvector.

For example, Insert($\ldots \gamma \lambda \alpha$, pos) performs the following operations:

Delete is symmetric. To support efficient bitvector initialization with a constant sequence, we introduce new dynamic compressed bitvector data structures.

This yields the first dynamic compressed sequence data structure that efficiently supports a dynamic alphabet.

Time and space

<table>
<thead>
<tr>
<th>Query</th>
<th>Append</th>
<th>Insert</th>
<th>Delete</th>
<th>Space (in bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>$O(</td>
<td>s</td>
<td>+ h_s)$</td>
<td>=</td>
</tr>
<tr>
<td>Append-only</td>
<td>$O(</td>
<td>s</td>
<td>+ h_s)$</td>
<td>$O(</td>
</tr>
<tr>
<td>Fully-dynamic</td>
<td>$O(</td>
<td>s</td>
<td>+ h_s \log n)$</td>
<td>$O(</td>
</tr>
</tbody>
</table>

- Sequence of $n$ strings $\langle s_0, \ldots, s_{n-1} \rangle$. $h_s$: number of nodes traversed in the trie for string $s$.
- Average height
- LB: information theoretic lower bound $LT + nH_s$, where $LT$ is the lower bound for the set of strings $S_{\text{set}}$
- PT: space for dynamic Patricia trie on the set of strings $S_{\text{set}}$