Text Indexing

Lecture 06: Wavelet Trees

Florian Kurpicz

The slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License ©CreativeCommons: www.creativecommons.org/licenses/by-sa/4.0 | commit 0cd47f0 compiled at 2023-12-04-08:48
Recap: Rank-Queries

- for a bit vector of size $n$
- blocks of size $s = \lfloor \frac{\lg n}{2} \rfloor$
- super blocks of size $s' = s^2 = \Theta(\lg^2 n)$
- information for 0s or 1s enough
  - $\text{rank}_1(i) = i - \text{rank}_0(i)$
- for all $\lfloor \frac{n}{s'} \rfloor$ super blocks, store number of 0s from beginning of bit vector to end of super-block
  - $n/s' \cdot \lg n = O(\frac{n}{\lg n}) = o(n)$ bits of space
- query in $O(1)$ time using three subqueries
  - one in super-block
  - one in block
  - one for remaining bitvector smaller than $s$
Select in $o(n)$ Space and $O(1)$ Time

- $select_0$ in a bit vector of size $n$ that contains $k$ zeros
- naive solutions
  - scan bit vector: $O(n)$ time and no space overhead
  - store $k$ solutions in $S[1..k]$ and $select_0(i) = S[i]$ if $k \in O(n/\lg n)$ this suffice

- better: $k/b$ variable-sized super-blocks $B_i$, such that super-block contains $b = \lg^2 n$ zeros
  - $select_0(i) = \sum_{j=0}^{\lfloor i/b \rfloor - 1} |B_j| + select_0(B_{\lfloor i/b \rfloor}, j - (\lfloor i/b \rfloor b))$

- storing all possible results for the (prefix) sum
  - $O((k \lg n)/b) = o(n)$ bits of space

- select on block depends on size of block
  - $|B_{\lfloor i/b \rfloor}| \geq \lg^4 n$: store answers naively
    - requires $O(b \lg n) = O(\lg^3 n)$ bits of space
    - there are at most $O(n/\lg^4 n)$ such blocks
    - total $O(n/\lg n) = o(n)$ bits of space
  - $|B_{\lfloor i/b \rfloor}| < \lg^4 n$: divide super-block into blocks
    - same idea: variable-sized blocks containing $b' = \sqrt{\lg n}$ zeros
    - (prefix) sum $O((k \lg n)/b') = o(n)$ bits
    - if size $\geq \lg n$ store all answers
    - if size $< \lg n$ store lookup table
Lemma: Binary Rank- and Select-Queries

Given a bit vector of size $n$, there exists data structures that can be computed in time $O(n)$ of size $o(n)$ bits that can answer rank and select queries on the bit vector in $O(1)$ time.
Preliminaries

Definition: Bit Representation
Given a text $T$ over an alphabet of size $\sigma$, each character can be represented using $\lceil \lg \sigma \rceil$ bits.
- the leftmost bit is the most significant bit and
- the rightmost bit is the least significant bit

for simplicity characters are integers
bit representation is integer in binary

Definition: Bit Prefix
A bit prefix of length $k$ are the $k$ MSBs of a character's bit representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

MSB

LSB
Definition: Wavelet Tree

Given a text $T$ of length $n$ over an alphabet $\Sigma = [1, \sigma]$, a wavelet tree is a binary tree, where

- each node represents characters in $[\ell, r] \subseteq [1, \sigma]$,
- if a node represents characters in $[\ell, r]$, then its left and right child
- represent characters in $[\ell, (\ell + r)/2)$ and $[(\ell + r)/2, r]$
- a node is a leaf if $\ell + 2 \geq r$
- characters are represented using a bit vector
- an entry is 1 if the character is represented in the right child and 0 otherwise

Definition: Level-wise Wavelet Tree

A wavelet tree, where all bit vectors on the same depth in the tree are concatenated is called level-wise wavelet tree

- in practice, level-wise wavelet trees have less overhead
- navigation still easy
Wavelet Trees (2/2)

Wavelet Trees (2/2)
The Intervals of a Wavelet Tree

- in each node, all represented characters share a bit prefix
- on depth $\ell$ the longest common bit prefix has length $\ell - 1$
- the bit prefixes form intervals

- finding characters in the wavelet tree requires finding the correct interval
- finding the position of a character requires finding the position in the last interval
### Rank-Queries
- Use rank queries on bit vectors
- At depth $\ell$ as for $\ell$-th MSB
- Follow through tree according to bit
- As seen on a previous slide

**PINGO** What is the query time of rank queries in wavelet trees?

### Select-Queries
- Identify leaf containing character
- Select corresponding occurrence in leaf
- Backtrack position up the tree to the root
- Requires up and down traversal of the wavelet tree
- See example on the board

### Access-Queries
- Follow bits through the wavelet tree
- Return read bits
- Same as rank but returning bit pattern instead of final rank
- See example on the board
Lemma: Query Times Wavelet Tree

Given a text $T$ over an alphabet of size $\sigma$, the wavelet tree of the text can answer rank, select, and access queries in $O(\lg \sigma)$ time.

Proof (Sketch)

All queries require
- just a constant number of rank and select queries on the bit vectors and
- at most one traversals from the root of the tree to a leaf and
- one traversal from a leaf to the root of the tree.
Given a bit representation of a character $\alpha$, $\text{reverse}(\alpha)$ reverses the bits, and the MSB becomes the least significant bit.

**Definition: Bit-Reversal Permutation**

The *bit-reversal permutation* $\rho_k$ is a permutation of the numbers $[0, 2^k)$ with

$$\rho_k(i) = \text{reverse}(i)$$

for $i \in [0, 2^k)$.

- $\rho_2 = (0, 2, 1, 3) = ((00)_2, (10)_2, (01)_2, (11)_2)$
- $\rho_{k+1} = (2\rho_k(0), \ldots, 2\rho_k(2^k - 1), 2\rho_k(0) + 1, \ldots, 2\rho_k(2^k - 1) + 1)$

- Same intervals as a wavelet tree
- Used in the wavelet matrix
Alternative Representation

- alternative representation of wavelet trees
- removing tree structure
- only two areas per level
- the intervals discussed before still exist

- better suited for large alphabets
- seemingly less structure
- retaining all important properties

Definition: Wavelet Matrix [CNP15]

Given a text $T$ of length $n$ over an alphabet of size $\sigma$, a wavelet matrix consists of:

- bit vectors $BV_\ell$ for $\ell \in [1, \lceil \log \sigma \rceil]$ of size $n$ and
- an array $Z[1..\lceil \log \sigma \rceil]$

Such that:

- $Z[\ell]$ contains the number of zero bits in $BV_\ell$
- $BV_1$ contains all MSBs in text order
- $BV_\ell$ contains the $\ell$-th MSB the character at position $i$ in $BV_{\ell-1}$ at position
  - $\text{rank}_0(i)$ if $BV_{\ell-1} = 0$ and
  - $Z[\ell - 1] + \text{rank}_1(i)$ if $BV_{\ell-1} = 1$
Intervals of a Wavelet Matrix

- A wavelet matrix has the same intervals a wavelet tree has.
- Intervals not bounded by parent no tree structure.

PINGO is answering queries with a wavelet matrix as simple as with a wavelet tree?
Example Wavelet Tree and Wavelet Matrix

- queries on the wavelet matrix work similar
- example on the board

```
BV_0
0 1 3 7 1 5 4 2 6 3
0 0 0 1 0 1 1 0 1 0
0 1 3 1 2 3 7 5 4 6

BV_1
0 0 1 0 1 1 0 0 1
0 1 1 3 2 3 5 4 7 6

BV_2
0 1 1 1 0 1 1 0 1 0
```

```
BV_0
0 1 3 7 1 5 4 2 6 3
0 0 0 1 0 1 1 0 1 0
0 1 3 1 2 3 7 5 4 6

BV_1
0 0 1 0 1 1 1 1 0 0 1
0 1 1 5 4 3 2 3 7 6

BV_2
0 1 1 1 0 1 0 0 1 1 0
```

Naive Wavelet Tree and Wavelet Matrix Construction (1/2)

Wavelet Tree
- first level are MSBs of characters of text
- for each level $\ell > 1$
  - stably sort text using Radix sort by bit prefixes of length $\ell - 1$
  - take $\ell$-th MSB of sorted sequence
  - sorted sequence is new text

Wavelet Matrix
- first level are MSBs of characters of text
- for each level $\ell > 1$
  - stably sort text by $\ell - 1$ MSB
  - take $\ell$-th MSB of sorted sequence
  - sorted sequence is new text

\[
\begin{array}{c|cccccccc}
   & 0 & 1 & 3 & 7 & 1 & 5 & 4 & 2 & 6 & 3 \\
BV_0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
BV_1 & 0 & 1 & 3 & 1 & 2 & 3 & 7 & 5 & 4 & 6 \\
BV_2 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
   & 0 & 1 & 1 & 3 & 2 & 3 & 5 & 4 & 7 & 6 \\
\end{array}
\]

\[
\begin{array}{c|cccccccc}
   & 0 & 1 & 3 & 7 & 1 & 5 & 4 & 2 & 6 & 3 \\
BV_0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
BV_1 & 0 & 1 & 3 & 1 & 2 & 3 & 7 & 5 & 4 & 6 \\
BV_1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\
BV_2 & 0 & 1 & 1 & 5 & 4 & 3 & 2 & 3 & 7 & 6 \\
   & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\
\end{array}
\]

\[Z[0] = 6 \quad Z[1] = 5 \quad Z[2] = 4\]
Wavelet Tree and Wavelet Matrix Construction (2/2)

- to make both fully functional bit vectors are augmented with binary rank and select support

**Lemma: Running Time and Memory Requirements Wavelet Tree and Wavelet Matrix**

Given a text $T$ over an alphabet of size $\sigma$, the wavelet tree and wavelet matrix require $(1 + o(1))n\lceil \lg \sigma \rceil$ bits of space and can be constructed in $O(n \lg \sigma)$ time

PINGO is there a asymptotically faster construction method?
Better Wavelet Tree Construction \[\text{[Bab+15; MNV16]}\]

- using requires broadword programming
- every \(\tau\)-th level is a big level
- big levels contain enough information to compute small levels below
- small levels computed by splitting big levels
- \(O(b/\lg n)\) characters at a time with \(b = o(\lg n)\)
- sketch on board 🎨

**Lemma: Better Wavelet Tree Construction**

Given a text \(T\) over an alphabet of size \(\sigma\), the wavelet tree and wavelet matrix require \((1 + o(1))n\lceil\lg \sigma\rceil\) bits of space and can be constructed in \(O(n \lg \sigma / \sqrt{\lg n})\) time.

- can be implemented using AVX/SSE instructions \([\text{Din+23; Kan18}]\)
wavelet trees can be compressed
more precise: the text can be compressed
use Huffman codes
wavelet trees cannot handle holes
use canonical Huffman codes

Canonical Huffman Codes (Recap)
- start with Huffman codes, code word 0, and length 1
- to get canonical code for current length, then add 1 to code word
- to update length add 1 and append required amount of zeros to code word

Huffman Codes (Recap)
- idea is to create a binary tree
- each character $\alpha$ is a leaf and has weight $\text{Hist}[\alpha]$ 
- create node for two nodes without parent with smallest weight
- give new node total weight of children
- repeat until only one node without parent remains
- label edges:
  - left edge: 0
  - right edge: 1
- path to children gives code for character
Huffman-shaped Wavelet Trees

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>$hc(\alpha)$</th>
<th>$chc(\alpha)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(11)$_2$</td>
<td>(11)$_2$</td>
</tr>
<tr>
<td>3</td>
<td>(01)$_2$</td>
<td>(10)$_2$</td>
</tr>
<tr>
<td>6</td>
<td>(100)$_2$</td>
<td>(011)$_2$</td>
</tr>
<tr>
<td>7</td>
<td>(101)$_2$</td>
<td>(010)$_2$</td>
</tr>
<tr>
<td>0</td>
<td>(0000)$_2$</td>
<td>(0011)$_2$</td>
</tr>
<tr>
<td>2</td>
<td>(0001)$_2$</td>
<td>(0010)$_2$</td>
</tr>
<tr>
<td>4</td>
<td>(0010)$_2$</td>
<td>(0001)$_2$</td>
</tr>
<tr>
<td>5</td>
<td>(0011)$_2$</td>
<td>(0000)$_2$</td>
</tr>
</tbody>
</table>

- Huffman codes (hc)
- canonical Huffman codes ( chc) that are bit-wise negated

- intervals are only missing to the right (white space)
- no holes allow for easy querying
Bottom-Up Construction [FKL18]

- scan the text and create histogram
- while scanning compute first level
- use histogram to compute borders of intervals
- scan text again and fill bit vectors

example on the next slide
Experimental Setup

- 64 GB RAM
- Two Intel Xeon E5-2640v4 CPUs (10 cores at 2.4 GHz base frequency, 3.4 GHz maximum turbo frequency, and cache sizes: 32 KB L1D and L1I, 256 KB L2, 25.6 MB L3)
- Same texts as in chapter 04
- Results are average of 5 runs
Experiments: Sequential Wavelet Tree Construction

- **Commoncrawl**
- **DNA**
- **Proteins**
- **Wikipedia**

**Variables:**
- **Input size**: $\lg n$ (B)
- **Throughput**: (Mbit/s)

**Methods:**
- naive
- pc
- pc.ss
- ps
- sdsl.pc
- serialWT

**Experiments:** Sequential Wavelet Tree Construction
## Experiments: Vectorized Wavelet Tree Construction [Din+23]

<table>
<thead>
<tr>
<th>File</th>
<th>lut</th>
<th>ext</th>
<th>shuf64</th>
<th>shuf128</th>
<th>shuf256</th>
<th>shuf512</th>
<th>pc</th>
<th>pc-ss</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp.xml</td>
<td>433.44</td>
<td>722.21</td>
<td>614.24</td>
<td>834.92</td>
<td>1 197.80</td>
<td>1 477.77</td>
<td>608.43</td>
<td>752.48</td>
</tr>
<tr>
<td>dna</td>
<td>529.32</td>
<td>883.00</td>
<td>563.11</td>
<td>668.93</td>
<td>862.49</td>
<td>1 011.45</td>
<td>594.02</td>
<td>745.68</td>
</tr>
<tr>
<td>english</td>
<td>456.91</td>
<td>770.55</td>
<td>677.96</td>
<td>906.42</td>
<td>1 304.80</td>
<td>1 642.69</td>
<td>623.08</td>
<td>704.90</td>
</tr>
<tr>
<td>pitches</td>
<td>448.02</td>
<td>749.24</td>
<td>686.88</td>
<td>886.62</td>
<td>1 276.36</td>
<td>1 584.19</td>
<td>578.70</td>
<td>328.47</td>
</tr>
<tr>
<td>proteins</td>
<td>375.73</td>
<td>575.99</td>
<td>565.63</td>
<td>707.23</td>
<td>985.35</td>
<td>1 178.02</td>
<td>633.58</td>
<td>761.41</td>
</tr>
<tr>
<td>sources</td>
<td>451.24</td>
<td>757.75</td>
<td>650.22</td>
<td>882.45</td>
<td>1 296.80</td>
<td>1 632.85</td>
<td>594.22</td>
<td>754.72</td>
</tr>
<tr>
<td>cc.16gib</td>
<td>453.97</td>
<td>729.58</td>
<td>653.25</td>
<td>875.61</td>
<td>1 265.27</td>
<td>1 604.84</td>
<td>628.46</td>
<td>752.97</td>
</tr>
<tr>
<td>dna.16gib</td>
<td>436.89</td>
<td>644.08</td>
<td>483.45</td>
<td>451.33</td>
<td>537.36</td>
<td>593.96</td>
<td>669.70</td>
<td>650.33</td>
</tr>
<tr>
<td>wiki.16gib</td>
<td>447.95</td>
<td>714.42</td>
<td>634.91</td>
<td>871.14</td>
<td>1 267.69</td>
<td>1 604.39</td>
<td>591.01</td>
<td>753.05</td>
</tr>
<tr>
<td>ru.8gib</td>
<td>317.20</td>
<td>642.51</td>
<td>506.04</td>
<td>660.23</td>
<td>938.68</td>
<td>1 121.03</td>
<td>346.96</td>
<td>170.44</td>
</tr>
</tbody>
</table>
Domain Decomposition [Fue+17]
- create wavelet tree in parallel using $p$ PEs
- each PE gets a consecutive slice of text
- each PE builds partial wavelet tree for its text
- merge partial wavelet trees in parallel

- can utilize any sequential algorithm
- very fast in practice
- $O(n \lg \sigma / \sqrt{\lg n})$ work and $O(\sigma + \lg n)$ time
  [Shu20]
Parallel merge
Experiments: Parallel Wavelet Tree Construction

![Graph showing Commoncrawl throughput (Gbit/s) vs. number of PEs (1 to 48) for 256 MiB, 512 MiB, and 1024 MiB per PE.](image)

- **256 MiB per PE**
  - ddWT
  - dd.ps
  - levelWT
  - ppc.ss
  - recWT
  - dd.pc
  - dd.pc.ss
  - ppc
  - pps
  - sortWT

- **512 MiB per PE**
  - ddWT
  - dd.ps
  - levelWT
  - ppc.ss
  - recWT
  - dd.pc
  - dd.pc.ss
  - ppc
  - pps
  - sortWT

- **1024 MiB per PE**
  - ddWT
  - dd.ps
  - levelWT
  - ppc.ss
  - recWT
  - dd.pc
  - dd.pc.ss
  - ppc
  - pps
  - sortWT
Conclusion and Outlook

This Lecture
- wavelet tree and wavelet matrix
- Huffman-shaped wavelet trees
- select on bit vectors
- practical algorithms for wavelet tree construction

Next Lecture
- FM-index
- r-Index

Linear Time Construction

- ST
- SA
- WT
- LZ
- LCP
- BWT
Bibliography I


[Din+23] Patrick Dinklage, Johannes Fischer, Florian Kurpicz, and Jan-Philipp Tarnowski. “Bit-Parallel (Compressed) Wavelet Tree Construction”. In: DCC. IEEE, 2023, pages 81–90. DOI: 10.1109/DCC55655.2023.00016.

Bibliography II


