Text Indexing: Lecture 5

Simon Gog – gog@kit.edu
Range Querying using Wavelet Trees

Applications in Text Indexes

- Top-\(k\) retrieval
- Position-restricted substring search
- Pattern matching with a fixed length gap
- LZ index
- Geometric BWT
- ...

We discuss work by Mäkinen & Navarro [1] (LATIN 2006) and Navarro & Russo [2] (ISAAC 2011)
Range Querying using Wavelet Trees

Problem 1
Given a \([0, n - 1] \times [0, n - 1]\) grid \(G\) and a set \(P\) of \(n\) points \((i, S[i])\) for \(0 \leq i < n\). For a pair of points \((x_0, y_0)\) and \((x_1, y_1)\) with \(x_0 \leq x_1\) and \(y_0 \leq y_1\) we define the following two queries:

- A range report query asks for all points \((x, y)\) \(\in P\) such that \(x \in [x_0, x_1]\) and \(y \in [y_0, y_1]\). Let \(R\) be the resulting set.
- A count query asks for the size \(|R|\) of \(R\).

Results
Count queries can be answered in \(O(\log n)\) time using an index of \(n \log n + o(n \log n)\) bits. Report queries in \(O(\log n + \text{occ log } n)\) time using the same index. \(\text{occ}\) is the number of points in \(R\).
range_count\( (wt, [x_0, x_1], [y_0, y_1]) \)

\[ \text{return } \text{range\_count}(wt, wt.\text{root}(), [x_0, x_1], [y_0, y_1]) \]

\[ \text{range\_count}(wt, v, [x'_0, x'_1], [y_0, y_1]) \]

\[ \text{if } x'_1 < x'_0 \text{ then return } 0 \]
\[ \text{if } [y_0, y_1] \cap y\_range(v) = \emptyset \text{ then return } 0 \]
\[ \text{if } y\_range(v) \subseteq [y_0, y_1] \text{ then return } x'_1 - x'_0 + 1 \]
\[ \langle v^\ell, v^r \rangle \leftarrow \text{wt.expand}(v) \]
\[ \langle [x^\ell_0, x^\ell_1], [x^r_0, x^r_1] \rangle \leftarrow \text{wt.expand}(v, [x'_0, x'_1]) \]
\[ \text{return } \text{range\_count}(wt, v^\ell, [x^\ell_0, x^\ell_1], [y_0, y_1]) + \]
\[ \text{range\_count}(wt, v^r, [x^r_0, x^r_1], [y_0, y_1]) \]
The range_count algorithm finds all $O(\log n)$ maximal WT nodes whose $y$-range covers $[y_0, y_1]$ (Line 3) and contains points from the original range $[x_0, x_1]$ (Line 1) all such points form an interval $[x'_0, y'_0]$ in node $v$. $x$-ranges of child nodes are mapped in constant time via rank operations in the expand call (Line 5) Children of WT nodes can be calculated in constant time (Line 4) also on pointerless WT (wt_int in SDSL) as discussed next
WT implementation for large alphabets

Problem
Storing the topology of a WT for large alphabets (e.g. $\sigma = n$) using
pointers generated considerable overhead: $O(n \cdot \log n)$ bits.

Solution: Implicit topology representation

Given a WT for a sequence $X$ of length $n$ and depth $L$.

- Store one bitvector $B$ of length $n \cdot L$.
- $B[0, n - 1]$ represents the first level, $B[n, 2n - 1]$ the second, and $B[\ell \cdot n, (\ell + 1) \cdot n - 1]$ level $\ell$ for $\ell < L$.
- A node $v$ is represented by a range $[lb(v), rb(v)]$ and its level $\text{level}(v)$.
- Given node $v$ the representation of the children $v_\ell$ and $v_r$ can be calculated as follows:
  - $\text{level}(v_\ell) = \text{level}(v_r) = \text{level}(v) + 1$
  - Let $z = \text{rank}(\text{level}(v) \cdot n + rb(v), 0, B) - \text{rank}(\text{level}(v) \cdot n + lb(v), 0, B)$
  - $[lb(v_\ell), rb(v_\ell)] = [lb(v), lb(v) + z - 1]$
  - $[rb(v_r), lb(v_r)] = [lb(v) + z, rb(v)]$
WT implementation for large alphabets

Side note: Wavelet Matrix for large alphabets
Range reporting using Wavelet Trees

- With range counting we have found the $O(\log n)$ nodes which cover the $y$-range of the query and contain points which lie in the $x$-range of the query.
- For reporting we have to visit all leaves of the $O(\log n)$ which meet the $x$ and $y$ constraints.
- Replace Line 3 in the `range_count` method by

```python
03 if y_range(v) ⊆ [y₀, y₁] then return range_report(wt, v, [x₀', x₁'])
```
range_report(\(wt, v, [x'_0, x'_1]\))

if \(x'_1 < x'_0\) then return 0

if is_leaf(\(v\)) then

\[y \leftarrow y_{\text{range}}(v)[0]\]

for \(x' \leftarrow x'_0\) to \(x'_1\) do

output \((wt.select(x' + 1, y), y)\)

return \(x'_1 - x'_0 + 1\)

else

\(\langle v^l, v^r \rangle \leftarrow wt.\text{expand}(v)\)

\(\langle [x^l_0, x^l_1], [x^r_0, x^r_1]\rangle \leftarrow wt.\text{expand}(v, [x'_0, x'_1])\)

return range_report(\(wt, v^l, [x^l_0, x^l_1], [y_0, y_1]\)) +

range_report(\(wt, v^r, [x^r_0, x^r_1], [y_0, y_1]\))
Applications

Position restricted substring searching

For a text $T$ of length $n$ and a query $q$ of length $m$
- a count query asks for the number of occurrences of $q$ in $T[\ell, r]$.
- a locate query asks for the list of all occurrences of $q$ in $T[\ell, r]$.

Solution

- Build a WT over the suffix array (SA).
- Get the SA-interval $[x_0, x_1]$ of $q$.
- Perform a locate (resp. count) query for the range $[x_0, x_1] \times [\ell, r]$ on the WT.
- Time complexity: $O(m \cdot t_{LF} + \log n + occ \log n)$
- Space: $n \log n + o(n \log n) + |CSA|$ bits
Applications

Substring rank and select
For a text $T$ of length $n$ and a query $q$ of length $m$
- a substring rank query $\text{rank}(i, q, T)$ asks for the number of occurrences of $q$ in $T[0, i – 1]$.
- a substring select query $\text{select}(i, q, T)$ asks for the $i$-th occurrences of $q$ in $T$.

Solution
Exercise
Pattern matching with a fixed length gap

For a text $T$ of length $n$ over alphabet $\Sigma$ and a query $q$ of the form $q_0 \ast^k q_1$ for a fixed $k \geq 0$. $q_0$ and $q_1$ are pattern over the alphabet $\Sigma$ and $\ast$ is a wildcard $\not\in \Sigma$ which matches any character in $\Sigma$. We define the following two queries:

- A count query asks for the number of occurrences of $q$ in $T$.
- A locate query asks for the list of all occurrences of $q$ in $T$.

Obvious solution:

- Use a SA to get a list of all occurrences of $q_0$ (called $L_0$) and $q_1$ (called $L_1$).
- Make one pass through both lists and filter out all pairs $(L_0[i], L_1[j])$ such that $L_0[i] + |q_0| + k = L[j]$
Pattern matching with a fixed length gap

Problems with obvious solution:

- Time of locate queries depends on size of subpattern list which might be much larger than the size of the occurrences of the whole pattern
- Count query not faster than locate query

Also: Not easy to create index, since the filtering condition depends on $|q_0|$.

Idea

Search bidirectional starting at the gap to be independent of $|q_0|$.

```
0 1 2 3 4 5 6 7 8 9 0 1
abraca da b r a $

\leftarrow \star \rightarrow
```
Pattern matching with a fixed length gap

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\overrightarrow{SA}$</th>
<th>$\overrightarrow{T}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>$$abracadabra$</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>a$$abracadabr$</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>abra$$abracad$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>abracadabra$$</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>acadabra$$abr$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>adabra$$abrac$</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>bra$$abracada$</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>bracadabra$$a$</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>cadabra$$abra$</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>dabra$$abraca$</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>ra$$abracadab$</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>racadabra$$ab$</td>
</tr>
</tbody>
</table>
Pattern matching with a fixed length gap

<table>
<thead>
<tr>
<th>i</th>
<th>$\mathsf{SA}$</th>
<th>$\mathsf{ISA}$</th>
<th>$\mathsf{T}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>5</td>
<td>$$arbadacarba$</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>11</td>
<td>a$$arbadacarba$</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>7</td>
<td>acarba$$arbad$</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>adacarba$$arb$</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>9</td>
<td>arba$$arbadac$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
<td>arbadacarba$$</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>8</td>
<td>ba$$arbadacar$</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>badacarba$$ar$</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>10</td>
<td>carba$$arbada$</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>6</td>
<td>dacarba$$arba$</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>1</td>
<td>rba$$arbadaca$</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>rbadacarba$$a$</td>
</tr>
</tbody>
</table>

- The prefix which ends at position $x$ in $\mathsf{T}$ starts as suffix $x' = n - x - 2$ in $\mathsf{T}$ for $x \in [0, n-2]$.
- For suffix $x$ and a gap $k$ we are interested in the suffix $x' = n - (x - k - 1) - 2$ in $\mathsf{T}$.
- ⇒ Mark points $(i, \mathsf{ISA}[n - \mathsf{SA}[i] + k - 1])$ in a grid $G$.
- Build WT $\mathsf{wt}_g$ over $G$. 

\[ \]
Applications

Pattern matching with a fixed length gap
Pattern matching with a fixed length gap

- $q_0 = a$
- $q_1 = ra$
- $k = 1$
- i.e. $q = a \ast ra$

Note: if $\overrightarrow{SA}[i] < k$ or $\overrightarrow{SA}[i] = n - 1$ we set a non-reachable point $(i, n)$
Applications

Pattern matching with a fixed length gap

Algorithm:
- Get SA-interval $[\ell_1, r_1]$ of $p_1$ in $SA$
- Get SA-interval $[\ell_0, r_0]$ of $p_0$ in $SA$
- `range_count(wt_grid, [\ell_1, r_1], [\ell_0, r_0])` corresponds to the number of matches
- Question: How do we get the occurrences?
Problem 2
Given a \([0, n - 1] \times [0, n - 1]\) grid \(G\) and a set \(P\) of \(n\) points \((i, S[i])\) with weight \(w[i]\) for \(0 \leq i < n\). For a pair of points \((x_0, y_0)\) and \((x_1, y_1)\) with \(x_0 \leq x_1\) and \(y_0 \leq y_1\) we define the top-\(k\) range query:

- A top-\(k\) range report query asks for the \(k\) points \((x, y) \in P\) such that \(x \in [x_0, x_1]\) and \(y \in [y_0, y_1]\) with maximum weight sorted in decreasing order of weight.

Results
Top-\(k\) range queries can be answered in \(O(\log^2 n + k \cdot \log n)\) time using an index of size \(3n \log n + o(n \log n) + |weights|\) bits, where \(|weights|\) is the space required to store the weights associated with the \(n\) points.