Text Indexing: Lecture 6

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

Applications in Text Indexes

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

We discuss work by Mäkinen & Navarro [1] (LATIN 2006) and Navarro & Russo [2] (ISAAC 2011)
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 $O(\log n + occ \log n)$ time using the same index. $occ$ is the number of points in $R$. 
Range Counting using Wavelet Trees

range_count(wt, \([x_0, x_1], [y_0, y_1]\))

return range_count(wt, wt.root(), \([x_0, x_1], [y_0, y_1]\))

range_count(wt, v, \([x'_0, x'_1], [y_0, y_1]\))

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

if \([y_0, y_1] \cap y\_range(v) = \emptyset\) then return 0

if \(y\_range(v) \subseteq [y_0, y_1]\) then return \(x'_1 - x'_0 + 1\)

\(<v^l, v^r> \leftarrow wt\_expand(v)\)

\(<[x^l_0, x^l_1], [x^r_0, x^r_1]> \leftarrow wt\_expand(v, [x'_0, x'_1])\)

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

range_count(wt, v^r, \([x^r_0, x^r_1], [y_0, y_1]\))
Range Counting using Wavelet Trees

- 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 ($\text{wt}_{\text{int}}$ 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 $level(v)$.
- Given node $v$ the representation of the children $v_\ell$ and $v_r$ can be calculated as follows:
  
  - $level(v_\ell) = level(v_r) = level(v) + 1$
  - Let $z = rank(level(v) \cdot n + rb(v), 0, B) - rank(level(v) \cdot n + lb(v), 0, B)$
  - $[lb(v_\ell), rb(v_\ell)] = [lb(v), lb(v) + z - 1]$
  - $[rb(v_r), lb(v_r)] = [lb(v) + z, rb(v)]$
WT implementation for large alphabets

0

\[ n \]

2n

3n

\[ lb(v) \]

\[ rb(v) \]

\[ vl \]

\[ vr \]

Side note: Wavelet Matrix for large alphabets
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 \textit{range\_count} method by

03 \hspace{1cm} \textbf{if} \hspace{0.5cm} y\_range(v) \subseteq [y_0, y_1] \hspace{0.5cm} \textbf{then return} \hspace{0.5cm} \text{range\_report}(wt, v, [x'_{0}, x'_{1}])
Range reporting using Wavelet Trees

```python
00 range_report(wt, v, [x'_0, x'_1])
 01 if x'_1 < x'_0 then return 0
 02 if is_leaf(v) then
 03 y ← y_range(v)[0]
 04 for x' ← x'_0 to x'_1 do
 05    output (wt.select(x' + 1, y), y)
 06    return x'_1 - x'_0 + 1
 07 else
 08    ⟨v^l, v^r⟩ ← wt.expand(v)
 09    ⟨[x^l_0, x^l_1], [x^r_0, x^r_1]⟩ ← wt.expand(v, [x'_0, x'_1])
 10 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: $O(m \cdot t_{LF} + \log n + \text{occ} \log n)$
- Space: $n \log n + o(n \log n) + |\text{CSA}|$ 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 $q_0^* k q_1$ for a fixed $k \geq 0$. $q_0$ and $q_1$ are pattern over the alphabet $\Sigma$ and $^*$ 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$da
abraca$da
```
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$$abra$</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>dabra$$abrac$</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

The prefix which ends at position $x$ in $\overrightarrow{T}$ starts as suffix $x' = n - x - 2$ in $\overleftarrow{T}$ 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 $\overleftarrow{T}$.

⇒ Mark points $(i, \overleftarrow{ISA}[n - \overrightarrow{SA}[i] + k - 1])$ in a grid $G$.

Build WT $wt_g$ over $G$
Applications

Pattern matching with a fixed length gap
Applications

Pattern matching with a fixed length gap

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

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

Pattern matching with a fixed length gap

Algorithm:
- Get SA-interval $[l_1, r_1]$ of $\overrightarrow{p}_1$ in $\overrightarrow{SA}$
- Get SA-interval $[l_0, r_0]$ of $\overleftarrow{p}_0$ in $\overleftarrow{SA}$
- range_count($wt\_grid$, $[l_1, r_1]$, $[l_0, r_0]$) corresponds to the number of matches
- Question: How do we get the occurrences?
Range Querying using Wavelet Trees

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 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.