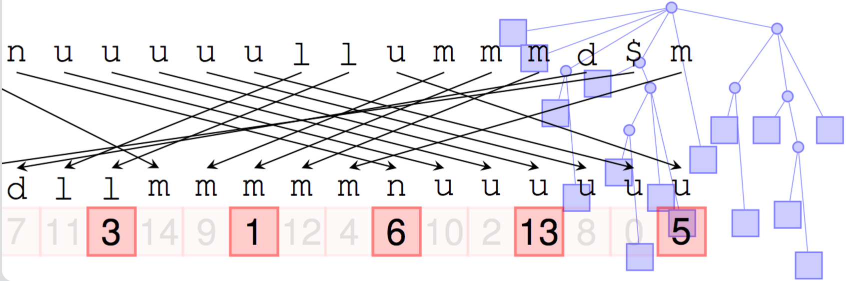
Karlsruhe Institute of Technology

Text Indexing: Lecture 6

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



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)

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 $O(\log n + occ \log n)$ time using the same index. *occ* is the number of points in R.

Range Counting using Wavelet Trees

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

$\text{range\_count}(wt, v, [x'_0, x'_1], [y_0, y_1])$
$\quad \textbf{if}\ x'_1 < x'_0\ \textbf{then return}\ 0$
$\quad \textbf{if}\ [y_0, y_1] \cap y\_range(v) = \emptyset\ \textbf{then return}\ 0$
$\quad \textbf{if}\ y\_range(v) \subseteq [y_0, y_1]\ \textbf{then return}\ x'_1 - x'_0 + 1$
$\quad \left\langle v^{\ell}, v^{r} \right\rangle \leftarrow wt.expand(v)$
$\quad \left\langle [x^{\ell}_0, x^{\ell}_1], [x^{r}_0, x^{r}_1] \right\rangle \leftarrow wt.expand(v, [x'_0, x'_1])$
$\quad \textbf{return}\ \text{range\_count}(wt, v^{\ell}, [x^{\ell}_0, x^{\ell}_1], [y_0, y_1]) +$
$\qquad\qquad \text{range\_count}(wt, v^{r}, [x^{r}_0, x^{r}_1], [y_0, y_1])$

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 (`wt_int` in SDSL) as discussed next

WT implementation for large alphabets

Problem

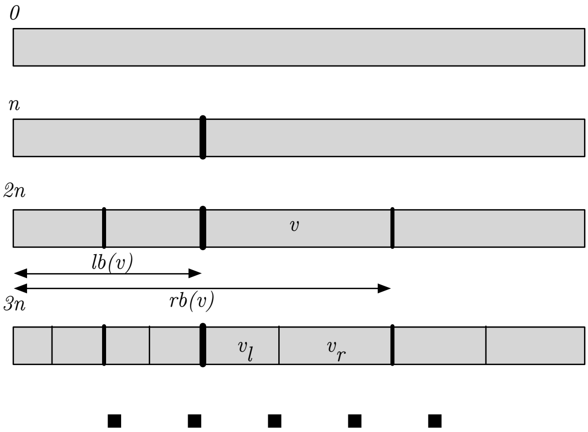
Storing the topology of a WT for large alphabets (e.g. $\sigma = n$) using pointers generated considerable overhead: $O(n \cdot \log n)$ bits.

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 ℓ 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_ℓ 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



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

03 $\quad$ **if** $y\_range(v) \subseteq [y_0, y_1]$ **then return** $\text{range\_report}(wt, v, [x'_0, x'_1])$

Range reporting using Wavelet Trees

range_report$(wt, v, [x'_0, x'_1])$
  if $x'_1 < x'_0$ **then return** 0
  if $is\_leaf(v)$ **then**
    $y \leftarrow y\_range(v)[0]$
    for $x' \leftarrow x'_0$ **to** x'_1 **do**
      output $(wt.select(x'+1, y), y)$
    return $x'_1 - x'_0 + 1$
  else
    $\left\langle v^\ell, v^r \right\rangle \leftarrow wt.expand(v)$
    $\left\langle [x^\ell_0, x^\ell_1], [x^r_0, x^r_1] \right\rangle \leftarrow wt.expand(v, [x'_0, x'_1])$
    return range_report$(wt, v^\ell, [x^\ell_0, x^\ell_1], [y_0, y_1])+$
        range_report$(wt, v^r, [x^r_0, x^r_1], [y_0, y_1])$

Applications

Karlsruhe Institute of Technology

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 $\mathit{rank}(i, q, T)$ asks for the number of occurrences of q in $T[0, i-1]$.
- a *substring select* query $\mathit{select}(i, q, T)$ asks for the i-th occurrences of q in T.

Solution

Exercise

Applications

Pattern matching with a fixed length gap

For a text T of length n over alphabet Σ 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 Σ and $*$ is a wildcard $\notin \Sigma$ which matches any character in Σ. 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 $L_0[i] + |q_0| + k = L[j]$

Applications

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
0 1 2 3 4 5 6 7 8 9 0 1
a b r a c a d a b r a $
        ←— * ——→
```

Applications

Pattern matching with a fixed length gap

i	$\overrightarrow{SA}$	$\overrightarrow{T}$
0	11	`$abracadabra`
1	10	`a$abracadabr`
2	7	`abra$abracad`
3	0	`abracadabra$`
4	3	`acadabra$abr`
5	5	`adabra$abrac`
6	8	`bra$abracada`
7	1	`bracadabra$a`
8	4	`cadabra$abra`
9	6	`dabra$abraca`
10	9	`ra$abracadab`
11	2	`racadabra$ab`

Applications

Karlsruhe Institute of Technology

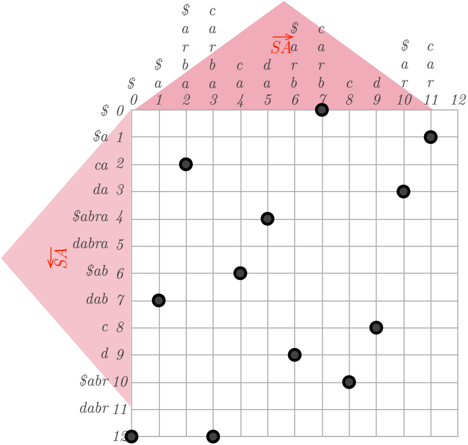
Pattern matching with a fixed length gap

i	$\overleftarrow{SA}$	$\overleftarrow{ISA}$	$\overleftarrow{T}$
0	11	5	`$arbadacarba`
1	10	11	`a$arbadacarb`
2	5	7	`acarba$arbad`
3	3	3	`adacarba$arb`
4	7	9	`arba$arbadac`
5	0	2	`arbadacarba$`
6	9	8	`ba$arbadacar`
7	2	4	`badacarba$ar`
8	6	10	`carba$arbada`
9	4	6	`dacarba$arba`
10	8	1	`rba$arbadaca`
11	1	0	`rbadacarba$a`

- 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 $\overleftarrow{T}$
- $\Rightarrow$ Mark points $(i, \overleftarrow{ISA}[n - \overrightarrow{SA}[i] + k - 1])$ in a grid G
- Build WT *wt_g* over G

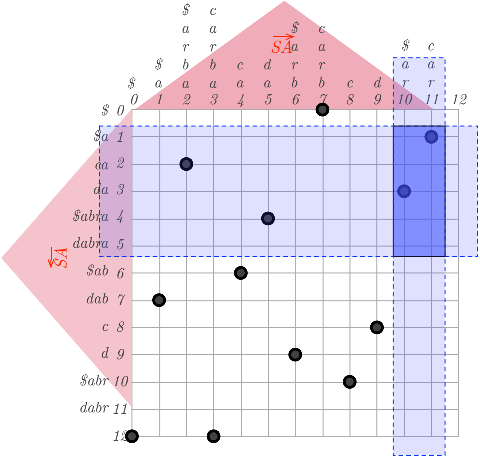
Applications

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. $q = a * ra$
- 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 $\overrightarrow{p}_1$ in $\overrightarrow{SA}$
- Get SA-interval $[\ell_0, r_0]$ of $\overleftarrow{p}_0$ in $\overleftarrow{SA}$
- 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?

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.

Bibliography

[1] Veli Mäkinen and Gonzalo Navarro. Position-restricted substring searching. In *Proc. LATIN*, pages 703–714, 2006.

[2] Gonzalo Navarro and Luís Russo. Space-efficient data-analysis queries on grids. In *Proc. ISAAC*, pages 323–332, 2011.