Lecture 10: Compressed Suffix Trees

Johannes Fischer
What we already have...

- Text + BWT + WT and backwards search
  \[ \mathcal{O}(m \lg \sigma) \] counting queries for \( P[1,m] \)
  - space \( \mathcal{O}(n \lg \sigma) \) bits (text size!)
- + sampled suffix array values
  \[ \mathcal{O}(k \lg n) \] for enumerating \( k \) occurrences
  - can be improved to \( \mathcal{O}(k \lg^\varepsilon n) \) for \( \varepsilon < 1 \)
Suffix Tree Functionality

• often, more functionality is desired
  ▶ repeat recognition (e.g. $T=\alpha\rho\beta\rho\gamma$)
  ▶ tandem repeats (e.g. $T=\alpha\rho\rho\beta$)
  ▶ longest common substrings
  ▶ matching statistics
  ▶ suffix-prefix matches
  ▶ etc.

• want suffix tree functionality!

compress?
Suffix Tree

$T = aababaa$'

$A = 8,7,6,1,4,2,5,3$
Some real numbers

- **guess**: how much bigger than text is ST?
  - A: <10
  - B: 10-20
  - C: >20

- Suffix Tree
  - 20-40 times text size !!!

- Text+BWT+WT (incl. rank/select):
  - \( \approx 3 \) times text size

- **goal**: drop suffix tree and simulate operations using suffix- and LCP array
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROOT()</strong></td>
<td>return root</td>
</tr>
<tr>
<td><strong>COUNT(v)</strong></td>
<td>count leaves below v</td>
</tr>
<tr>
<td><strong>ISANCESTOR(v,w)</strong></td>
<td>true if v is an ancestor of w</td>
</tr>
<tr>
<td><strong>ISLEAF (v)</strong></td>
<td>true if v is a leaf</td>
</tr>
<tr>
<td><strong>LEAFLABEL(v)</strong></td>
<td>suffix number represented by leaf v</td>
</tr>
<tr>
<td><strong>SDEPTH(v)</strong></td>
<td>string depth of v</td>
</tr>
<tr>
<td><strong>PARENT(v)</strong></td>
<td>parent node of v</td>
</tr>
<tr>
<td><strong>FIRSTCHILD(v)</strong></td>
<td>first (alphabetically smallest) child of v</td>
</tr>
<tr>
<td><strong>NEXTSIBLING(v)</strong></td>
<td>next sibling of v</td>
</tr>
<tr>
<td><strong>EDGELABEL(v,i)</strong></td>
<td>i’th letter on the edge leading to v</td>
</tr>
<tr>
<td><strong>LCA(v,w)</strong></td>
<td>lowest common ancestor of v and w</td>
</tr>
</tbody>
</table>
ROOT()
\textbf{COUNT}(v) = 5
\textsc{IsAncestor}(v, w) = \text{True}
\textsc{IsAncestor}(v,w) = \textsc{False}
ISLEAF(\(v\))

ISLEAF(\(v\)) = TRUE
ISLEAF(v)

ISLEAF(v) = FALSE
\textbf{LEAFLABEL}(v) = 4
$\text{SD}epth(v) = 3$
\textbf{PARENT}(v) = w
\textbf{FIRST\textsc{Child}(v)}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{tree.pdf}
\caption{FIRST\textsc{Child}(v) = w}
\end{figure}
$\text{NextSibling}(v) = w$
\textsc{EdgeLabel}(v,i)

\textsc{EdgeLabel}(v,2)=a
LCA\((v, w)\)

\[ \text{EDGELABEL}(v, w) = u \]
Goal

drop suffix tree and simulate operations using suffix- and LCP array
Represent Nodes by Intervals

A = 8, 7, 6, 1, 4, 2, 5, 3

H = -1, 0, 1, 2, 1, 3, 0, 2
Represent Nodes by Intervals

A = 8, 7, 6, 1, 4, 2, 5, 3

H = -1, 0, 1, 2, 1, 3, 0, 2
Intervals $[v_l, v_r]$ in $H$

1. $H[i] \geq \text{SDPETH}(v)$
   $\forall \ v_l < i \leq v_r$

2. $H[v_l] < \text{SDPETH}(v)$
   $H[v_r + 1] < \text{SDPETH}(v)$

3. $\exists \ v_l < i \leq v_r$ with
   $H[i] = \text{SDPETH}(v)$

$A = 8, 7, 6, 1, 4, 2, 5, 3$

$H = -1, 0, 1, 2, 1, 3, 0, 2$
Consequences

1. \( H[i] \geq S\text{DEPTH}(v) \)
   \[ \forall v_l < i \leq v_r \]

2. \( H[v_l] < S\text{DEPTH}(v) \)
   \[ H[v_r + 1] < S\text{DEPTH}(v) \]

3. \( \exists v_l < i \leq v_r \) with
   \[ H[i] = S\text{DEPTH}(v) \]

(1) given \( v_l \) & \( v_r \): compute \( i \) by \( i \leftarrow \text{RMQ}_H(v_l + 1, v_r) \)

\[ A = 8, 7, 6, 1, 4, 2, 5, 3 \]
\[ H = -1, 0, 1, 2, 1, 3, 0, 2 \]

\[ \text{RMQ} \]
Consequences

1. $H[i] \geq \text{SDPETH}(v)$
   \[ \forall v_l < i \leq v_r \]

2. $H[v_l] < \text{SDPETH}(v)$
   $H[v_r+1] < \text{SDPETH}(v)$

3. \( \exists v_l < i \leq v_r \) with
   $H[i] = \text{SDPETH}(v)$

(1) given $v_l$ & $v_r$: compute $i$ by $i \leftarrow \text{RMQ}_H(v_l+1, v_r)$

(2) given $i$: compute
   $v_l \leftarrow \text{PSV}_H(i)$
   $v_r \leftarrow \text{NSV}_H(i)-1$

\[ A = \{8, 7, 6, 1, 4, 2, 5, 3\} \]
\[ H = \{-1, 0, 1, 2, 1, 3, 0, 2\} \]
3 Components of CST

- A: compressed (sampled) suffix array
- H: compressed LCP-array
- Compressed RMQ & PSV/NSV on LCP

CST

Node $v$ represented by interval $[v_l, v_r]$ in $H$ (or $A$)
**IsAncestor**($v, w$)

IsAncestor($v, w$) = True $\iff v_l \leq w_r \leq v_r$

$A = 8, 7, 6, 1, 4, 2, 5, 3$

$H = -1, 0, 1, 2, 1, 3, 0, 2$
\textbf{PARENT}(v):}

larger of $H[v_l] \& H[v_r+1]$ is SDEPTH of parent!

\[ A = 8, 7, 6, 1, 4, 2, 5, 3 \]

\[ H = -1, 0, 1, 2, 3, 0, 2 \]
<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ROOT()</strong></td>
<td>return root</td>
</tr>
<tr>
<td><strong>COUNT(v)</strong></td>
<td>count leaves below v</td>
</tr>
<tr>
<td><strong>ISANCESTOR(v,w)</strong></td>
<td>true if v is an ancestor of w ✓</td>
</tr>
<tr>
<td><strong>ISLEAF(v)</strong></td>
<td>true if v is a leaf</td>
</tr>
<tr>
<td><strong>LEAFLABEL(v)</strong></td>
<td>suffix number represented by leaf v</td>
</tr>
<tr>
<td><strong>SDEPTH(v)</strong></td>
<td>string depth of v</td>
</tr>
<tr>
<td><strong>PARENT(v)</strong></td>
<td>parent node of v</td>
</tr>
<tr>
<td><strong>FIRSTCHILD(v)</strong></td>
<td>first (alphabetically smallest) child of v</td>
</tr>
<tr>
<td><strong>NEXTSIBLING(v)</strong></td>
<td>next sibling of v</td>
</tr>
<tr>
<td><strong>EDGELABEL(v,i)</strong></td>
<td>i’th letter on the edge leading to v</td>
</tr>
<tr>
<td><strong>LCA(v,w)</strong></td>
<td>lowest common ancestor of v and w</td>
</tr>
</tbody>
</table>
Summary

• Represent ST nodes by *intervals*

• Simulate operations by **RMQs** and **PSV**s/**NSV**s on LCP-array

  $\Rightarrow$ suffix and LCP-array **replace** suffix tree

• for a completely compressed ST:
  - compress LCP-array
  - small-space solutions for RMQ/PSV/NSV