Text Indexing: Lecture 8

Simon Gog – gog@kit.edu
Definition of an LCP-interval ([1])

An interval \([i, j]\), where \(0 \leq i \leq n - 1\) is called LCP-interval of LCP value \(\ell\) (denoted by \(\ell - [i, j]\)) if

- \(LCP[i] < \ell\) or \(i = 0\)
- \(LCP[k] \geq \ell\) for all \(k \in [i + 1, j]\)
- \(LCP[k] = \ell\) for at least one \(k \in [i + 1, j]\)
- \(LCP[j + 1] < \ell\)

Every index \(k\) with \(i < k \leq j\) and \(LCP[k] = \ell\) is called \(\ell\)-index. There are at most \(\sigma - 1\) \(\ell\)-indices in an LCP-interval.

Note: Each LCP-interval corresponds to a node in the suffix tree.
The LCP-Interval Tree – Example

<table>
<thead>
<tr>
<th>$i$</th>
<th>SA</th>
<th>$LCP$</th>
<th>$\mathcal{T}[SA[i], n - 1]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11</td>
<td>0</td>
<td>$$</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>0</td>
<td>i$</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>1</td>
<td>ippi$</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>issippi$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>ississippi$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>mississippi$</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>0</td>
<td>pi$</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>1</td>
<td>ppi$</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>0</td>
<td>sippi$</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2</td>
<td>sissippi$</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>1</td>
<td>ssippi$</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>3</td>
<td>ssissippi$</td>
</tr>
</tbody>
</table>

Singleton intervals $\ell - [i, i]$ are omitted
Properties of the LCP-Interval Tree [4]

Overlapping

Two lcp-intervals \( \ell - [i, j] \neq m - [p, q] \) cannot overlap, i.e. one of the following cases must hold:

- \([i, j]\) is a subinterval of \([p, q]\), i.e. \(p \leq i < j \leq q\).
- \([p, q]\) is a subinterval of \([i, j]\), i.e. \(i \leq p < q \leq j\).
- \([i, j]\) and \([p, q]\) are disjoint, i.e. \(j < p\) or \(q < i\).
Properties of the LCP-Interval Tree [4]

Child interval
An \( m \)-interval \([p, q]\) is said to be embedded in an \( \ell \)-interval \([i, j]\) if it is a subinterval of \([i, j]\) and \( m > \ell \). The \( \ell \)-interval is then called the interval enclosing \([p, q]\). If \([i, j]\) encloses \([p, q]\) and there is no other interval embedded in \([i, j]\) that also encloses \([p, q]\), then \([p, q]\) is called a child interval of \([i, j]\) (vice versa, \([i, j]\) is called parent interval of \([p, q]\)).

Navigation: child operation
Let \([i, j]\) be an \( \ell \)-interval. If \( i_1 < i_2 < \ldots < i_k \) are the \( \ell \)-indices in ascending order, then the child intervals of \([i..j]\) are \([i, i_1 - 1], [i_1, i_2 - 1], \ldots [i_k, j]\).
Previous/Next Smaller Value Queries

Let $A$ be an array of length $n$. For $i \in [1, n - 1]$ the previous smaller value function is defined as:

$$psv(i, A) = \max\{j \mid 0 \leq j < i \land A[j] < A[i]\}$$

Analogously we define the next smaller value function:

$$nsv(i, A) = \min\{j \mid i < j < n \land A[j] < A[i]\}$$

We omit $A$ in $psv / nsv$ if it is clear from the context.
Exercise

With $n \log n$ bits of space the psv or nsv function can be precomputed and answered in constant time. Devise a linear time algorithm to compute the table.
Let $0 < k < n$ and $LCP[k] = \ell$. Then $[psv(k), nsv(k) - 1]$ is an lcp-interval of LCP-value $\ell$.

Proof:

- $LCP[psv(k)] < \ell$ (by definition of $psv(k)$)
- $LCP[m] \geq \ell$ for all $m \in [psv(k) + 1, nsv(k) - 1]$
- $LCP[k] = \ell$ (note that $psv(k) + 1 \leq k \leq nsv(k) - 1$)
- $LCP[nsv(k)] < \ell$ (by definition of $nsv(k)$)
Let \([i, j] \neq [0, n-1]\) be an lcp-interval with \(LCP[i] = p\) and \(LCP[j + 1] = q\)

- **case** \(p = q\): \(p-[psv(i), nsv(j) - 1]\) is parent of \([i, j]\)
- **case** \(p > q\): \(p-[psv(i), j]\) is parent of \([i, j]\)
- **case** \(p < q\): \(q-[i, nsv(i)]\) is parent of \([i, j]\)
Overview

We have already seen how to represent the CSA and LCP part of a suffix tree (ST) space-efficiently. Now we present different solutions to represent the tree topology/ navigational part of the ST. We will concentrate on two representations:

- Balanced Parentheses Sequence (BPS) of the ST
- BPS of the Super-Cartesian Tree of the LCP Array

The suffix tree of a text $T$ of length $n$ consists of at most $2n - 1$ nodes, with $n$ leaves. Pointer representation would take $O(n \log n)$ bits.
BPS Representation of ST

- Given a traversable tree representation
- Traverse tree in depth first order
- Initialize empty sequence $\text{BPS}_{dfs}$
- Append opening parenthesis to $\text{BPS}_{dfs}$ when visiting a node the first time
- Append closing parenthesis to $\text{BPS}_{dfs}$ when all nodes of the node’s subtree were visited
- Identify each node with the position of its opening parenthesis in $\text{BPS}_{dfs}$
BPS Representation of ST

\[
\text{BPS}_{\text{dfs}} = (())(())(())(()(()))(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())()}
BPS Representation of ST

$BPS_{dfs} = (())((()))(()(()()))((()))(()((()))))$
BPS Representation of ST

BPS_{dfs} = ((()())(()))(()(()))((()))((())()(())))
BPS Representation of ST

BPS_{dfs} = ((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))((()))(}
BPS Representation of ST

$\text{BPS}_{dfs} = (())(()(()(())((()(()))))(((()(()(()))))))$
BPS Representation of ST

BPS\textsubscript{dfs} = (())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())(()())()}
BPS Representation of ST

BPS_{dfs} = (0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(0)(
BPS Representation of ST

\[ \text{BPS}_{dfs} = (())(()()())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(())(.)
BPS Representation of ST

\[ BPS_{dfs} = (())()()()()(())()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()0 \]
BPS Representation of ST

BPS_{dfs} = (())(()())(()(()))((())())(()(()))(()(()))()
BPS Representation of ST

Space usage: at most $4n$ bits

BPS\textsubscript{dfs} = (())(()(()))(()(()())())(()(()))(()(()))()(()(()))()
Represent BPS_{dfs} as bitvector:
- Opening parenthesis represented as „1”
- Closing parenthesis represented as „0”
- Leaves are represented by bitpattern „10”

We can support rank/select on bitpatterns „0”, „1”, „10” by adding o(n) bits. Exercise: Implement the following operations in constant time:
- Get root of tree.
- Select the i-th leaf (numbered from left to right).
- Test if a node ν is a leaf.
- Left bound of node ν’s interval.
Support more complex tree operation
Given nodes $v, w$.

- Get size (=number of leaves) of subtree rooted at $v$.
- Get right bound of $v$’s interval ($rb(v)$).
- Get parent of $v$ ($parent(v)$).
- Lowest common ancestor of $v$ and $w$ ($lca(v, w)$).
- (Right) Sibling of $v$ ($siblings(v)$).
Support the following basic operations on a bitvector $b$

\[excess(i) = \text{# of 1 bits minus # of 0-bits in } b[0..i]\]

\[find\_close(i) = \min\{j \mid j > i \land excess(j) = excess(i) - 1\}\]

\[find\_open(i) = \max\{j \mid j < i \land excess(j) = excess(i) + 1 \land B[j] = 1\}\]

\[enclose(i) = \max\{j \mid j < i \land find\_close(j) > find\_close(i)\}\]

\[double\_enclose(i, j) = \max\{k \mid k < i \land find\_close(k) > find\_close(j)\}\]

\[rr\_enclose(i, j) = \min\{k \mid k \in [find\_close(i) + 1, j - 1] \land \exists m \leq k \land \forall n > k \land n \in [i, j] : find\_close(n) > find\_close(m)\}\]

Operation $find\_close(i)$, $enclose(i)$ can be used to solve $parent(v)$, operation $double\_enclose(i, j)$ to solve $lca(v, w)$. 
For a balanced parentheses sequence of length $n$ the presented operations can be supported in constant time and $o(n)$ additional space.

- We present the solution for operation $\text{find\_close}(i)$
- First, techniques from Jacobson’s $O(n)$ extra space solution [3]
- Then Geary et al.’s improvement to $o(n)$ extra space [2]
BPS Representation of ST
Efficient Navigation

excess:
parentheses:
pioneer bitmap:
block numbers of matching parantheses of pioneers:

block 0
block 1
block b−2
block b−1

matches for far parentheses

1 0 0 0 1 0 0 0 0 0 0 1 0 1 1

1 0 0 0 1 0 0 0 1 0 0 1 1 0 1 0 0 1

b−1
1
0
2

b−2
0

Simon Gog:
Text Indexing: Lecture 8
Institute of Theoretical Informatics
Algorithmics
Given a BPS of size \( n \).

Partition BPS into \( N \) blocks of size \( L = \frac{1}{2} \log n \)

Let \( \mu(i) = \begin{cases} \text{find\_close}(i) & \text{if } i \text{ represents an opening parenthesis} \\ \text{find\_open}(i) & \text{otherwise} \end{cases} \)

Let \( \beta(i) = \frac{i}{L} \) be the block ID of the \( i \)-th parenthesis

We call a parenthesis a **far parenthesis** if \( \beta(i) \neq \beta(\mu(i)) \)

A far parenthesis \( i \) is called a **pioneer** if there is no other parenthesis \( j < i \) with \( \beta(j) = \beta(i) \) and \( \beta(\mu(j)) = \beta(\mu(i)) \). \( \mu(i) \) is also called pioneer.

Upper bound for pioneers is \( 4N - 6 = \frac{8n}{\log n} - 6 = \mathcal{O}\left(\frac{n}{\log n}\right) \)

Blocks are nodes in the pioneer graph \( G \). Each pioneer \( i \) adds an edge \( (\beta(i), \beta(\mu(i))) \).

\( G \) is outerplanar. Maximal edges in outerplanar graph: \( 2N - 3 \).
BPS Representation of ST

Efficient Navigation - $\mathcal{O}(n)$ space structure for \texttt{find_close}

Components:

- Bitvector $PB$ which marks the pioneers + rank + select. Space: $n + o(n)$ bits.
- For each pioneer store the matching block in an array $M[0, N - 1]$. Space: $\mathcal{O}(\frac{n}{\log n} \log(n)) = \mathcal{O}(n)$.
- Rank structure for BPS (to get excess values).
- Precomputed table $P$ for in-block queries. Space: $\mathcal{O}(\sqrt{n} \log^2 n)$.
**BPS Representation of ST**

Efficient Navigation - $O(n)$ space structure for *find_close*

---

*find_close(i)* in constant time

- Let $i$ be (the position of) an opening parenthesis.
- If $i$ is not a far parenthesis: Use $P$ to get result.
- Use a lookup table to get the largest pioneer $j$ with $\beta(j) = \beta(i)$ and $j \leq i$.
- Go to block $x = M[\text{rank}(j, 1, PB)]$.
- Determine the first position $k$ in block $x$ such that $\text{excess}(k) = \text{excess}(i) - 1$.
- Return $k$.

Note: *find_open(i)* is symmetric.
PB is a sparse uniform bitvector. There is a $O(n \frac{\log \log n}{\log n}) = o(n)$ representation which also supports rank in constant time.

The subsequence of pioneers of the original BPS forms again a BPS called BPS’.

Instead of storing $M$ for the original BPS, we build the linear space findclose structure on BPS’.

This takes $O(\frac{n}{\log n})$ bits space.

Exercise
Describe how the \textit{enclose} operation can be solved in constant time with $o(n)$ additional space.
Definition
Let $A[l, r]$ be an array of integers. The Super-Cartesian Tree $C^{sup}(A[l, r])$ of $A[l, r]$ is recursively constructed as follows:
- $C^{sup}(A[l, r])$ is empty, if $l > r$
- otherwise, let $p_0 < p_1 < \ldots < p_{k-1}$ the minima in $A[l, r]$. Create $k$ nodes $v_0, v_1, \ldots, v_{k-1}$ and label each $v_j$ with $p_j$. For each $j$ with $0 < j < k$, node $v_j$ is the right sibling of node $v_{j-1}$. Recursively construct $C_0 = C^{sup}(A[l, p_0 - 1]), C_1 = C^{sup}(A[p_0, p_1 - 1]), \ldots, C_{k-1} = C^{sup}(A[p_{k-1}, p_1 - r])$. For each $j$ with $0 \leq j < k$ the left child of $v_j$ is the root of $C_j$. The right child of $v_{k-1}$ is the root of $C_k$.

Blackboard: Example for array: 0,0,0,3,0,1,5,2,2,0,0,4,1,2,6,1.
BPS of the Super-Cartesian Tree of the LCP Array

BPS\textsubscript{sc} = ( ( \\
LCP = 0 0 0 3 0 1 5 2 2 0 0 4 1 2 6 1)
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = ( ( ) ) \]

\[ \text{LCP} = \begin{array}{cccccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = ( ( ( ) ) ) \]

\[ \text{LCP} = \begin{array}{ccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1 \\
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = ( ( ( ( ( ) ) ) ) \]

\[ \text{LCP} = \begin{array}{ccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1 & & \\
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = ( ( ( ( ( ) ) ) ) \]

LCP = 0 0 0 3 0 1 5 2 2 0 0 4 1 2 6 1
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = ( ( ( ( ) ) ) ( ) } \]

\[ \text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1 \\
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

$$BPS_{sct} = (\ (\ (\ )) (\ ))$$

$$LCP = 0\ 0\ 0\ 3\ 0\ 1\ 5\ 2\ 2\ 0\ 0\ 4\ 1\ 2\ 6\ 1$$
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{\text{sct}} = ( ( ( ( ) ) ) ( ( ) ) )
\]

\[
\text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sc} = ( ( ( ( ( )) (())) } \]

\[ \text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2
\end{array} \begin{array}{cccccccc}
0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = \begin{pmatrix} \end{pmatrix} \]

\[ \text{LCP} = \begin{pmatrix} 0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1 \end{pmatrix} \]
BPS of the Super-Cartesian Tree of the LCP Array

$$\text{BPS}_{sc} = ( ( ( ( ) ( ( ) ( ) ) ) )$$

$$\text{LCP} = \begin{array}{cccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}$$
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = ( ( ( ( ) ) ( ( ) ( ) ) ) ) \]

\[ \text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{sct} = ( ( ( ( ( ( ( ( ( ( )))))))
\]

\[
\text{LCP} = \begin{array}{ccccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{scf} = ( ( ( ( ( ( ( ( ( ( ) ) ) ) ) ) ) ) ) \]

\[ \text{LCP} = 0 \ 0 \ 0 \ 3 \ 0 \ 1 \ 5 \ 2 \ 2 \ 0 \ 0 \ 4 \ 1 \ 2 \ 6 \ 1 \]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{sct} = ( ( ( ) ) ( ( ) ( ) ))) ) ( \\
\text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sc} = ( ( ( ( ) ) ( ( ) ( ( ))) ( ( ) ( ( )) ) ) } \]

\[ \text{LCP} = [0, 0, 0, 3, 0, 1, 5, 2, 2, 0, 0, 4, 1, 2, 6, 1] \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sc}= ( ( ( ( ) ) ( ( ) ( ( ) ) ) ) ) \]

\[ \text{LCP}= \begin{array}{cccccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\begin{align*}
\text{BPS}_{sct} & = & \left( \begin{array}{c}
\left( \begin{array}{c}
\left( \begin{array}{c}
\left( \begin{array}{c}
0
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{array}\right)
\end{align*}
\]

\[
\text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1 & & \\
\end{array}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{sct} = \begin{pmatrix}
( & ( & ( & ( & ) & ( & ( & ) & ( & ) & ) & ) & ) & ) & ( & ) & ) & )
\end{pmatrix}
\]

\[
\text{LCP} = \begin{pmatrix}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{pmatrix}
\]
BPS of the Super-Cartesian Tree of the LCP Array

$$\text{BPS}_{sct} = ( ( ( ( ( ) ) ( ( ) ) ( ( ) ) ) ) ( ( ) ) ( ( ) ) )$$

$$\text{LCP} = 0 \ 0 \ 0 \ 3 \ 0 \ 1 \ 5 \ 2 \ 2 \ 0 \ 0 \ 4 \ 1 \ 2 \ 6 \ 1$$
BPS of the Super-Cartesian Tree of the LCP Array

$$\text{BPS}_{sct} = ( ( ( ( ) ) ( ( ) ) ( ( ) ) ) ( ( ) ) ( ( ) ) )$$

$$\text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}$$
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{\text{sc}t} = ( ( ( ( ) ) ( ( ( ) ) ( ( ))) ( ( ( )) ( ( ))) ) \]

\[ \text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sc} = ( ( ( ( ) ) ( ( ) ) ( ( ) ) ) ) ( ( ( ) ) ( ( ) ) ) \]

\[ \text{LCP} = \begin{bmatrix} 0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1 \end{bmatrix} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sc} = ( ( ( ( ) ( ( ( ())))(( ( ( ())))( ))) ) \]

\[ \text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1 \\
\end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{sc} = \begin{pmatrix}
( & ( & ( & ( & ) & ( & ( & ) & ( & ) & ) & ) & ( & ( & ) & ) & ) & ( & ( & ) & ) & ) & ( & ( & ) & ) & ) & \end{pmatrix}
\]

\[
\text{LCP} = \begin{pmatrix}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{pmatrix}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sct} = \begin{pmatrix} ( & ( & ( & ( & ) & ) & ( & ) & ( & ( & ( & ( & ) & ) & ) & ) & ) & ) \end{pmatrix} \]

\[ \text{LCP} = \begin{pmatrix} 0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1 \end{pmatrix} \]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\begin{align*}
\text{BPS}_{scT} &= ( ( ( ( ) ( ( ( ) ( ( ) ) ( ( ) ) ( ( ) ) ( ( ) ) ) ) ) ) \\
\text{LCP} &= \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}
\end{align*}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[ \text{BPS}_{sc} = ( ( ( ( ) ) ( ( ) ( ) ) ) ( ( ( ) ( ) ) ) ) \]

\[ \text{LCP} = 0 \quad 0 \quad 0 \quad 3 \quad 0 \quad 1 \quad 5 \quad 2 \quad 2 \quad 0 \quad 0 \quad 4 \quad 1 \quad 2 \quad 6 \quad 1 \]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{sct} = ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( ( ())))))) ))) ))) )))
\]

\[
\text{LCP} = \begin{array}{cccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1 \\
\end{array}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[
\text{BPS}_{\text{sct}} = ( ( ( ( ( ) ) ( ( ) ) ) ) ( ( ) ) ( ( ) ) ( ( ) ( ) ) ( ) )
\]

\[
\text{LCP} = \begin{array}{cccccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 & 0 & 0 & 4 & 1 & 2 & 6 & 1
\end{array}
\]
BPS of the Super-Cartesian Tree of the LCP Array

\[ BPS_{sct} = ( ( ( ( ) ) ( ( ( ) ) ) ( ( ( ) ) ) ) ) \]

\[ \text{LCP} = \begin{array}{ccccccccc}
0 & 0 & 0 & 3 & 0 & 1 & 5 & 2 & 2 \\
0 & 0 & 4 & 1 & 2 & 6 & 1 & \end{array} \]
BPS of the Super-Cartesian Tree of the LCP Array

Blackboard: Linear time construction algorithm
BPS of the Super-Cartesian Tree of the LCP Array

Operations:
- Next smaller value ($nsv(i)$): ?
- Previous smaller or equal value : ?
- Previous smaller value ($psv(i)$): ? (add additional bitvector)
- parent operation in the lcp-interval tree can be solved with $nsv(i)$ and $psv(i)$
- Find $\ell$-indices: ?
- RMQ: ?

See Chapter 6.3 in [4].

