Advanced Data Structures
Simon Gog – gog@kit.edu
Range minimum queries (RMQs)

Definition
Given an array $A$ of length $n$ containing elements from a totally ordered set. A range minimum query $rmq_A(\ell, r)$ returns the position of the minimal element in the sub-array $A[\ell, r]$:

$$rmq_A(\ell, r) = \arg \min_{\ell \leq k \leq r} A[k]$$

Example

$$A = \begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8
\end{array}$$
Range minimum queries (RMQs)

Definition
Given an array $A$ of length $n$ containing elements from a totally ordered set. A range minimum query $rmq_A(\ell, r)$ returns the position of the minimal element in the sub-array $A[\ell, r]$: 

$$rmq_A(\ell, r) = \arg \min_{\ell \leq k \leq r} A[k]$$

Example

\[
A = \begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\hline
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8 \\
\end{array}
\]

$rmq(2, 6) = 4$
Range minimum queries (RMQs)

Definition
Given an array $A$ of length $n$ containing elements from a totally ordered set. A range minimum query $rmq_A(\ell, r)$ returns the position of the minimal element in the sub-array $A[\ell, r]$:

$$rmq_A(\ell, r) = \arg\min_{\ell \leq k \leq r} A[k]$$

Example

\[
A = \begin{bmatrix}
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8 \\
\end{bmatrix}
\]

$rmq(5, 9) = 6$
Range minimum queries (RMQs)

Definition
Given an array $A$ of length $n$ containing elements from a totally ordered set. A range minimum query $rmq_A(\ell, r)$ returns the position of the minimal element in the sub-array $A[\ell, r]$:

$$rmq_A(\ell, r) = \arg \min_{\ell \leq k \leq r} A[k]$$

Example

$$A = \begin{array}{cccccccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8
\end{array}$$

$$rmq(0, 15) = 4$$
Overview

- Notation: Complexity of an algorithm is denoted with \( \langle f(n), g(n) \rangle \), where \( f(n) \) is preprocessing time and \( g(n) \) query time.
- Different solutions:
  - naïve approach 1: \( \langle O(n^2), O(1) \rangle \) using \( O(n^2) \) words of space
  - naïve approach 2: \( \langle O(1), O(n) \rangle \)
  - \( \langle O(n), O(\log n) \rangle \) using \( O(n) \) words of space
  - \( \langle O(n \log n), O(1) \rangle \) using \( O(n \log n) \) words of space
  - \( \langle O(n \log \log n), O(1) \rangle \) using \( O(n \log \log n) \) words of space
  - \( \langle O(n), O(1) \rangle \) using \( O(n) \) words of space
  - \( \langle O(n), O(1) \rangle \) using \( 4n + o(n) \) bits of space
  - \( \langle O(n), O(1) \rangle \) using \( 2n + o(n) \) bits of space

Note

The last two solutions do not require access to the original array \( A \).
Overview

Literature

- M.A. Bender, M. Farach-Colton: The LCA Problem Revisited. (LATIN 2000)
- K. Sadakane: Compressed Suffix Trees with Full Functionality. (TCS 2007)
- H. Ferrada, G. Navarro: Improved Range Minimum Queries (DCC 2016)
〈O(n), O(log n)〉 – solution #1

\[ A = \begin{bmatrix} 8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8 \end{bmatrix} \]
\[ O(n), O(\log n) \] – solution #1

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{array}
\]

\[
A = \begin{array}{cccccccccccc}
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8 \\
\end{array}
\]
\( \langle O(n), O(\log n) \rangle \) – solution #1

\[ A = \begin{bmatrix}
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8 \\
\end{bmatrix} \]

\[ rmq(1, 5) = 4 \]
\( O(n), O(\log n) \) – solution #1

\[
A = \begin{bmatrix}
8 & 2 & 4 & 7 & 1 & 9 & 3 & 5 & 7 & 4 & 6 & 4 & 3 & 1 & 4 & 8 \\
\end{bmatrix}
\]

\( rmq(1, 5) = 4 \)
Store index of minimum in binary interval tree.
Tree has $O(n)$ nodes.
Follow all nodes which overlap with the query interval but are not fully contained in it (at most 2 per level).
So not more than $2 \log n$ such nodes in total.
Select all children of these nodes which are fully contained in the query interval.
From these nodes select the index with minimal value.
For each item $A[i]$ store an array $M_i[0, \log n]$.

$M_i[j] = \text{rmq}_A(i, i + 2^j - 1)$

Space is $O(n \log n)$ words

How long does pre-computation take?

Querying

Find the largest $k$ with $2^k \leq \ell - r + 1$. Then

$$\text{rmq}_A(\ell, r) = \begin{cases} 
M_i[k] & \text{if } A[M_i[k]] < A[M_{j-2^k+1}[k]] \\
M_{j-2^k+1}[k] & \text{otherwise}
\end{cases}$$

Question: How can $k$ be determined in constant time?
\begin{itemize}
  \item Split $A$ into $t = \frac{n}{\log n}$ blocks $B_0, \ldots, B_{t-1}$. $B$ spans $O(\log n)$ items of $A$.
  \item Create an array $S[0, t-1]$ with $S[i] = \min\{x \in B_i\}$
  \item Build rmq structure #2 for $S$
  \item For each block $B_i$ of $O(\log n)$ elements build rmq structure #2
  \item Total space: $O(n) + O(n \log \log n)$
\end{itemize}

**Querying**

\begin{itemize}
  \item Determine blocks $B_{\ell'}$, $B_{r'}$ which contain $\ell$ and $r$
  \item Calculate $m = \text{rmq}_S(\ell' + 1, r' - 1)$
  \item Let $k_0, k_1, k_2$ be the results of the RMQs in blocks $\ell'$, $r'$, and $m$ relative to $A$
  \item Return $\arg\min_{k_i} A[k_i]$ for $0 \leq k \leq 3$
\end{itemize}
Definition
The Cartesian Tree $C$ of an array is defined as follows:
- The root of $C$ is the (leftmost) minimum element of the array and is labeled with its position
- Removing the root splits the array into two pieces
- The left and right children of the root are recursively constructed Cartesian trees of the left and right subarray
- $C$ can be constructed in linear time
Solution overview:

- Partition the array into blocks of size $s$
- Each block corresponds to a Cartesian Tree of size $s$
- Precompute the $s^2$ answers for all $\frac{1}{n+1}\binom{2n}{n}$ possible Cartesian Trees of size $s$ in a table $P$.
- $P$ requires $O(2^{2s}s^2)$ words of space
- For $s = \frac{\log n}{4}$, $P$ requires $o(n)$ words of space
- Build structure #2 for array $A'$ consisting of the block minima of $A$. This takes $O(n)$ construction time and uses $O(n)$ words of space.
LCA (Lowest Common Ancestor)

Given a rooted tree $T$ of $n$ nodes. For nodes $v$ and $w$ of $T$ the query $LCA_T(v, w)$ returns the *lowest common ancestor* of $u$ and $v$ in $T$. I.e. the node which is (1) ancestor of $v$ and $w$ and (2) maximizes the distance to the root.
LCA (Lowest Common Ancestor)

Given a rooted tree $T$ of $n$ nodes. For nodes $v$ and $w$ of $T$ the query $LCA_T(v, w)$ returns the lowest common ancestor of $u$ and $v$ in $T$. I.e. the node which is (1) ancestor of $v$ and $w$ and (2) maximizes the distance to the root.
LCA (Lowest Common Ancestor)

Given a rooted tree $T$ of $n$ nodes. For nodes $v$ and $w$ of $T$ the query $LCA_T(v, w)$ returns the *lowest common ancestor* of $u$ and $v$ in $T$. I.e. the node which is (1) ancestor of $v$ and $w$ and (2) maximizes the distance to the root.
Lemma
If there is a \( \langle f(n), g(n) \rangle \)-time solution for RMQ, then there is a \( \langle f(2n - 1) + O(n), g(2n - 1) + O(1) \rangle \)-time solution for LCA.

- Let \( T \) be the Cartesian Tree of array \( A \)
- Let array \( E[0, \ldots, 2n - 2] \) store the nodes visited in an DFS Euler Tour of \( T \)
- Let array \( L[0, \ldots, 2n - 2] \) store the corresp. levels of the nodes in \( E \)
- Let \( R[0, \ldots, n - 1] \) be an array which stores a representative \( R[i] = \min\{j \mid E[j] = i\} \) for each node \( i \) of \( T \)
LCA & $\pm 1$ RMQ

Example

```
 E = a b c b d b a e f g f e h e i j i k i e a
 L = 0 1 2 1 2 1 0 1 2 3 2 1 2 1 2 3 2 3 2 1 0
 R = 0 1 2 4 7 8 9 12 14 15 17
```
LCA & ±1 RMQ

Example

\begin{align*}
\text{i} & = 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 \\
\text{E} & = a b c b d b a e f g f e h e i j i k i e a \\
\text{L} & = 0 1 2 1 2 1 0 1 2 3 2 1 2 1 2 3 2 3 2 1 0 \\
\text{R} & = 0 1 2 4 7 8 9 12 14 15 17 \\

LCA_T(v, w) & = E[RMQ_L(min(R[v], R[w]), max(R[v], R[w]))]
\end{align*}
LCA & $\pm 1$RMQ

Example

\[
i = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0
\]
\[
E = a \ b \ c \ b \ d \ b \ a \ e \ f \ g \ f \ e \ h \ e \ i \ j \ i \ k \ i \ e \ a
\]
\[
L = 0 \ 1 \ 2 \ 1 \ 2 \ 1 \ 0 \ 1 \ 2 \ 3 \ 2 \ 1 \ 2 \ 1 \ 2 \ 3 \ 2 \ 3 \ 2 \ 1 \ 0
\]
\[
\begin{array}{ccccccc}
\text{a} & \text{b} & \text{c} & \text{d} & \text{e} & \text{f} & \text{g} \\
\text{h} & \text{i} & \text{j} & \text{k}
\end{array}
\]
\[
R = 0 \ 1 \ 2 \ 4 \ 7 \ 8 \ 9 \ 12 \ 14 \ 15 \ 17
\]

\[
LCA_T(v, w) = E[RMQ_L(min(R[v], R[w]), max(R[v], R[w])))]
\]

Note: $(L[i] - L[i + 1]) \in \{-1, +1\}$. So we only need to solve RMQs over arrays with this $\pm 1$ restriction. This is called $\pm 1$RMQ.
\( O(n), O(1) \) solution \((4n + o(n) \text{ bits})\)
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

\[
\begin{align*}
i & = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \\
A & = 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3
\end{align*}
\]

(1) Build Cartesian Tree
\( \langle O(n), O(1) \rangle \) solution \( (4n + o(n) \text{ bits}) \)

\[
i = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \\
A = 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3
\]

(1) Build Cartesian Tree
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

\[
\begin{align*}
  i &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \\
  A &= 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3
\end{align*}
\]

(1) Build Cartesian Tree
\[ \langle O(n), O(1) \rangle \text{ solution (} 4n + o(n) \text{ bits) } \]

\[
\begin{align*}
\text{i} &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \\
A &= 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3
\end{align*}
\]

(1) Build Cartesian Tree
\( \langle O(n), O(1) \rangle \) \textbf{solution} \((4n + o(n) \text{ bits})\)

\[ i = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \]
\[ A = 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3 \]

(1) Build Cartesian Tree
\[ O(n), O(1) \] solution \((4n + o(n) \text{ bits})\)

\[ i = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \]
\[ A = 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3 \]

(1) Build Cartesian Tree
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

\[
\begin{align*}
  i &= 0 \hspace{0.5cm} 1 \hspace{0.5cm} 2 \hspace{0.5cm} 3 \hspace{0.5cm} 4 \hspace{0.5cm} 5 \hspace{0.5cm} 6 \hspace{0.5cm} 7 \hspace{0.5cm} 8 \hspace{0.5cm} 9 \hspace{0.5cm} 0 \hspace{0.5cm} 1 \\
  A &= 4 \hspace{0.5cm} 6 \hspace{0.5cm} 3 \hspace{0.5cm} 5 \hspace{0.5cm} 1 \hspace{0.5cm} 4 \hspace{0.5cm} 6 \hspace{0.5cm} 4 \hspace{0.5cm} 5 \hspace{0.5cm} 2 \hspace{0.5cm} 6 \hspace{0.5cm} 3
\end{align*}
\]

(1) Build Cartesian Tree

![Cartesian Tree Diagram]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

\[
\begin{align*}
  i &= 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 0 \quad 1 \\
  A &= 4 \quad 6 \quad 3 \quad 5 \quad 1 \quad 4 \quad 6 \quad 4 \quad 5 \quad 2 \quad 6 \quad 3
\end{align*}
\]

(1) Build Cartesian Tree
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

\[ i = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \]
\[ A = 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3 \]

(1) Build Cartesian Tree
\langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits})

i = 0 1 2 3 4 5 6 7 8 9 0 1
A = 4 6 3 5 1 4 6 4 5 2 6 3

(1) Build Cartesian Tree
\[ \langle O(n), O(1) \rangle \] solution \((4n + o(n) \text{ bits})\)

\[
\begin{align*}
\text{i} & = 0 1 2 3 4 5 6 7 8 9 0 1 \\
A & = 4 6 3 5 1 4 6 4 5 2 6 3
\end{align*}
\]

(1) Build Cartesian Tree
\langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits})

\[
\begin{align*}
    i &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \\
    A &= 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3
\end{align*}
\]

(1) Build Cartesian Tree
\(\langle O(n), O(1) \rangle\) solution \((4n + o(n)\) bits\)

\[
\begin{align*}
i &= 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 0 \ 1 \\
A &= 4 \ 6 \ 3 \ 5 \ 1 \ 4 \ 6 \ 4 \ 5 \ 2 \ 6 \ 3
\end{align*}
\]

(2) Add a leaf to each node
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = (1\)
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

\(\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}\)

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = ( ( \]
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = ( ( ( \) ) ) \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = ( ) ( ) ( ) \]
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = ( ((()) ) ) \]
\langle O(n), O(1) \rangle \textbf{ solution } (4n + o(n) \text{ bits})

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = (((()())(}
\]

\[
1 \ 3 \ 4 \ 6
\]

14 Simon Gog:
Advanced Data Structures

Institute of Theoretical Informatics
Algorithmics
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( ((())()) ( \\
0
) )
\]
\[<O(n), O(1)> \text{ solution } (4n + o(n) \text{ bits})\]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[BP_{ext} = ( (( ( ) ) ) ( ) \]

\[0 \ 1 \ 3 \ 4 \ 6\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ((()))()
\]

0 1 2 3 4 5 6 7 8 9 10 11

0 1 2 3 4 5 6 7 8 9 10 11

6
3
5
1
4
6
4
5
2
6
3
\( \langle O(n), O(1) \rangle \) solution \( (4n + o(n) \text{ bits}) \)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ((()()()()()) \) \]

0 1 2 3 4 5 6 7 8 9 10 11
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( ( ( ) ( ) ) ) ( 0 1 )
\]
\( O(n), O(1) \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ( ((()())())())() \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = (( ( ) ( ( ) ) ) ( ) ) \]

0 1 2 3 4 5 6 7 8 9 10 11
\( \langle O(n), O(1) \rangle \) solution \( (4n + o(n) \text{ bits}) \)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = ( ((())())())()() \]

0 1 2 3 4 5 6 7 8 9 10 11
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = ( ((()((())())))())()()
\]

0 1 2 3 4 5 6 7 8 9 10 11
\[O(n), O(1)\] solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = ((((())))())()() \\
0 1 2 3
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( (( ( ) ( ( ) ) ) ( ) ( ( ) ) )
\]

\[
0 1 2 3 4 5 6 7 8 9 10 11
\]

Simon Gog:
Advanced Data Structures

Institute of Theoretical Informatics
Algorithmics
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = (( ( ( ) ( ) ) ) ( ) ( ( ) ) ) ( \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = (((((()())())(()))())()) \]

\[ 0 \ 1 \ 2 \ 3 \ 4 \]
\(\langle O(n), O(1) \rangle\) solution \((4n + o(n)\) bits\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = ( ((()) ()()) ()()() ) ( \\
0 1 2 3 4
\]

Simon Gog: 
Advanced Data Structures

Institute of Theoretical Informatics 
Algorithmics
\(\langle O(n), O(1) \rangle\) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ( ( ( ( ( ( ) ) ) ) ) ( ) ( ( ) ) ) ( ) ( ) \]

0 1 2 3 4 5 6 7 8 9 10 11
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

```
0 1 2 3 4 5 6 7 8 9 10 11
```

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = ( ((())())())()(()())()()()\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = \left( ((())(())) (()) (()) (()) \right)
\]
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ( (( () ( () )) ( ) ( () ) ) ( ) ( () ) ( ) ( ) ( ) ( ) ( ) ) \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n)\) bits\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = \left( \left( \left( (())(()) \right)(())(()) \right)(())(()) \right)
\]

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
8 & 9 & 10 & 11 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 3 & 4 & 6 & 6 & 4 & 5 & 5 \\
3 & 2 & 4 & 4 & 6 & \end{array}
\]
\(O(n), O(1)\) solution \((4n + o(n)\) bits\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[BP_{\text{ext}} = ( ( ( ( ) ( ( ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ) ( ( ) ) ( ) )\]
\langle O(n), O(1) \rangle \textbf{ solution} \ (4n + o(n) \text{ bits})

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\begin{align*}
BP_{\text{ext}} &= \langle ( ( ( ( ) ( ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ) ( ( ) ) \\
&\quad \langle 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \rangle
\end{align*}
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = ((()())())(()())()(()())(()())
\]

```
0 1 2 3 4 5 6
```

```
1 3 4 6 6 4 5 5 3 2 4 4 6 6
```

```
6 5 6
```
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ([(()()())()()()()()()()()] ) \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = ( ( ( ( ( ) ) ) ) ( ) ( ( ) ) ) ( ) ( ( ( ) ( ( ) ) ) ( )
\]

\[
0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ( ( ( ( ( ( ) ) ) ) ( ) ( ( ) ) ) ( ( ) ) ( ( ) ) ( ( ) ) ( ) ) \]

[Diagram of a tree with nodes labeled 0 to 11, and leaf nodes added to each node.]
\langle O(n), O(1) \rangle \textbf{ solution} (4n + o(n) \text{ bits})

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = \left( ( ( ( ( ) ) ) ( ( ) ) ) ( ( ) ) ( ( ) ) ( ( ) ) ( ( ) ) ( ( ) ) \right)
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ( ((()())()) ()(())) () ((())((())()())()() ) \]
〈O(n), O(1)〉 solution (4n + o(n) bits)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

$BP_{ext} =$

```
(((()(()))))(()(()))(()(())(()())())((())())()
```

0 1 2 3 4 5 6 7 8 9 10 11
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = \langle ( ( ( ) ( ( ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ( ( ) ) ( ) ( ( ) ) ) \rangle \]
\langle O(n), O(1) \rangle \textbf{ solution } (4n + o(n) \text{ bits})

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{ext} = \left( ( ( ( ) ( ( ) ) ) ( ( ) ) ) ( ( ) ) ( ( ) ) ( ( ) ) ( ( ) ) ( ( ) ) \right) \]
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = ( ( ( ( ) ( ( ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ( ( ( ) ) ( ) ( ( ) ) ) ) ) ( \\
0 1 2 3 4 5 6 7 8 9 10 11
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11
\end{array}
\]

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( ( ( ( ( ( ) ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ( ( ( ) ) ( ) ( ( ) ) ) ) ) ( )
\]

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{array}
\]
\[\langle O(n), O(1) \rangle \textbf{ solution } (4n + o(n) \text{ bits})\]

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[BP_{\text{ext}} = ( ( ( ( ) ( ( ) ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ( ( ( ) ( ) ( ) ) ) ) ) ( ) ( \]

0 1 2 3 4 5 6 7 8 9 10 11
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\( BP_{\text{ext}} = ( ( ( ( ) ( ( ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ( ( ) ) ( ) ( ( ) ) ) ) ( ) ( ) \)
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = \left( ((())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())())(}
\]

Simon Gog: Advanced Data Structures

Institute of Theoretical Informatics
Algorithmics
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( (((((()))())())())())(())(((())()(()))())())(((())())())())\)
\[
( 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

\[
\begin{array}{ccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11
\end{array}
\]

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = \left( ( ( ( ) ( ( ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ( ( ) ) ( ) ( ( ) ) ) ) ( ) ( ( ) ) \right)
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( ( ( ( ) ) ) ) ( ) ( ( ) ) ) ( ) ) (( ( ( ) ) ) ( ) ( ( ) ) ) ) ( ) ( ( ) ) ) ( ) ( ( ) ) ( )
\]
\(\langle O(n), O(1) \rangle\) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = (( ( ( ( ) ) ) ) ( ) ( ) ) ( ) ((( ( ( ) ) ) ( ) ) ( ) ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) \]

0 1 2 3 4 5 6 7 8 9 10 11
\( \langle O(n), O(1) \rangle \) \textbf{solution} \( (4n + o(n) \text{ bits}) \)

(2) Add a leaf to each node

![Diagram showing a tree with leaves added]

(3) DFS traversal to construct balanced parentheses sequence

\[ BP_{\text{ext}} = ( ( ( ( ( ( ) ) ) ) ( ) ( ( ) ) ) ) ( ( ( ) (( )) ( ) ( ( ) ) ) ) ( ( ( ( ) ) ) ( ) ) \]

\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

\[
\begin{array}{cccccccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{ext} = ( ((()(()))(()))((())((()))(())))(())((()))(()))((())(()))
\]
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

(2) Add a leaf to each node

(3) DFS traversal to construct balanced parentheses sequence

\[
BP_{\text{ext}} = (((())())(())) () ((())((())(()()))) () ((())())())
\]

0 1 2 3 4 5 6 7 8 9 10 11
\[ \langle O(n), O(1) \rangle \textbf{ solution } (4n + o(n) \text{ bits}) \]

- The extended Cartesian Tree contains \(2n\) nodes
- The balanced parentheses sequence consists of \(4n\) bits
- The position of each leaf corresponds to the inorder number of its parent in the Cartesian tree
- The inorder number corresponds to the array index of the element
- Let \(\text{excess}(i) = \text{rank}(i + 1, 1, BP_{ext}) - \text{rank}(i + 1, 0, BP_{ext})\)

For \(0 \leq i \leq j < n\) we get:

00 \( \text{rmq}_A(i, j) \)
01 \( \text{ipos} \leftarrow \text{select}(i + 1, (\cdot), BP_{ext}) \)
02 \( \text{jpos} \leftarrow \text{select}(j + 1, (\cdot), BP_{ext}) \)
03 \( \text{return } \text{rank}(\text{rmq}_{\text{excess}}^{\pm 1}(\text{ipos}, \text{jpos} + 1), (\cdot), BP_{ext}) \)
\( \langle O(n), O(1) \rangle \) solution \((4n + o(n) \text{ bits})\)

- Added leaves are used to navigate to inorder index nodes
- Select on a pattern \(\cdot\) of fixed size can be done in constant time after precomputing a \(o(n)\)-space structure
- Let \(v\) be the \((i + 1)\)th leaf node
- Let \(w\) be the \((j + 1)\)th leaf node
- \(\text{rmq}^{\pm 1}_{\text{excess}}(ipos, jpos + 1)\) is the position of closing parenthesis of the leaf node \(z\) added to \(\text{LCA}(v, w)\)
- In-order number of \(\text{LCA}(v, w)\) corresponds to index of minimum in \(A[i, j]\) and can be determined by a rank operation on pattern \(\cdot\)

Next: \(o(n)\) data structure to support \(\text{rmq}^{\pm 1}_{\text{excess}}\) queries on \(BP_{\text{ext}}\)
\[ \langle O(n), O(1) \rangle \text{ solution } (4n + o(n) \text{ bits}) \]

- Divide the (conceptional) array excess in blocks of size \( \log^3 n \)
- \( S[0, n/ \log^3 n] \) stores the minima of the blocks
- Solution #2 for \( S \) requires \( O(\frac{n}{\log^3 n} \cdot \log^2 n) = o(n) \) bits
- Divide each block in subblocks of size \( \frac{1}{2} \log n \)
- Apply again solution #2 on subblocks \( (n' = \log^2 n) \). I.e. total space \( O(\frac{n}{\log n} \log n' \cdot \log n') = O(\frac{n}{\log n} \log^2 \log n) = o(n) \)
- Lookup table for blocks of size \( \frac{1}{2} \log n \) is also in \( o(n) \)
\( O(n), O(1) \) solution \((2n + o(n) \text{ bits})\)

Observation
Adding the leaf nodes enables inorder indexing of the nodes but doubles the space.

Next: Approach which does not require additional nodes.
- Cartesian Tree (CT) is a binary tree of \( n \) nodes
- Transform CT into general tree of \( n + 1 \) nodes
\(\langle O(n), O(1) \rangle\) solution \((2n + o(n) \text{ bits})\)

Observation

Adding the leaf nodes enables inorder indexing of the nodes but doubles the space.

Next: Approach which does not require additional nodes.

- Cartesian Tree (CT) is a binary tree of \(n\) nodes
- Transform CT into general tree of \(n + 1\) nodes
\langle O(n), O(1) \rangle \textbf{solution} \ (2n + o(n) \text{ bits})

Observation
Adding the leaf nodes enables inorder indexing of the nodes but doubles the space.

Next: Approach which does not require additional nodes.
- Cartesian Tree (CT) is a binary tree of \( n \) nodes
- Transform CT into general tree of \( n + 1 \) nodes
\( O(n), O(1) \) solution \((2n + o(n) \text{ bits})\)

**Observation**
Adding the leaf nodes enables inorder indexing of the nodes but doubles the space.

Next: Approach which does not require additional nodes.
- Cartesian Tree (CT) is a binary tree of \( n \) nodes
- Transform CT into general tree of \( n + 1 \) nodes
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Observation

Adding the leaf nodes enables inorder indexing of the nodes but doubles the space.

Next: Approach which does not require additional nodes.

- Cartesian Tree (CT) is a binary tree of \( n \) nodes
- Transform CT into general tree of \( n + 1 \) nodes
\(O(n), O(1)\) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root
- For each node \(v\) (starting at the root) take the right child \(w\), and add the nodes on the leftmost path from \(w\) as children of \(v\)
\(\langle O(n), O(1) \rangle\) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root
- For each node \(v\) (starting at the root) take the right child \(w\), and add the nodes on the leftmost path from \(w\) as children of \(v\)
\( O(n), O(1) \) solution \((2n + o(n) \text{ bits})\)

**Transformation**

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root.
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \).
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

**Transformation**

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \( (2n + o(n) \text{ bits}) \)

Transformation

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

**Transformation**

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)

![Diagram showing the transformation process](image)
\[\langle O(n), O(1) \rangle \text{ solution } (2n + o(n) \text{ bits})\]

Transformation

- Add a new root node to the leaf of the original root
- For each node \(v\) (starting at the root) take the right child \(w\), and add the nodes on the leftmost path from \(w\) as children of \(v\)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( O(n), O(1) \) solution \((2n + o(n) \text{ bits})\)

**Transformation**

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Transformation

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\langle O(n), O(1) \rangle \textbf{ solution} \ (2n + o(n) \text{ bits})

Transformation

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

**Transformation**

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

**Transformation**

- Add a new root node to the leaf of the original root
- For each node \( v \) (starting at the root) take the right child \( w \), and add the nodes on the leftmost path from \( w \) as children of \( v \)

![Diagram of tree transformation](image)
\( \langle O(n), O(1) \rangle \) \textbf{solution} \((2n + o(n) \text{ bits})\)

- Node with inorder \(i\) in CT becomes node with preorder \(i + 1\) in the general tree
- So we can identify the node of array element \(i\) by selecting the \((i + 2)\)th opening parenthesis in the balanced parentheses sequence of the general tree (+1 for the added root node, +1 for index shift by one)
- Let \(BP\) be the balanced parentheses sequence of the general tree

For \(0 \leq i \leq j < n\) we get:

00 \( \text{rmq}_A(i,j) \)
01 \( \text{i} \text{pos} \leftarrow \text{select}(i + 2, (, BP) \)
02 \( \text{j} \text{pos} \leftarrow \text{select}(j + 2, (, BP) \)
03 \( \text{return} \ \text{rank}(\text{rmq}^{\pm 1}_{\text{excess}}(\text{i} \text{pos} - 1, \text{j} \text{pos}), (, BP) - 1 \)

Where \(\text{rmq}^{\pm 1}_{\text{excess}}\) returns the rightmost position of the minimal value.
\( O(n), O(1) \) solution \((2n + o(n) \text{ bits})\)

- Node with inorder \(i\) in CT becomes node with preorder \(i + 1\) in the general tree.
- So we can identify the node of array element \(i\) by selecting the \((i + 2)\)th opening parenthesis in the balanced parentheses sequence of the general tree (+1 for the added root node, +1 for index shift by one).
- Let \(BP\) be the balanced parentheses sequence of the general tree.

For \(0 \leq i \leq j < n\) we get:

00 \( \text{rmq}_A(i, j) \)
01 \( \text{ipos} \leftarrow \text{select}(i + 2, (, BP) \}
02 \( \text{jpos} \leftarrow \text{select}(j + 2, (, BP) \}
03 \( \text{return rank}(\text{rmq}_{excess}^\pm(i \text{pos} - 1, j \text{pos}), (, BP) - 1 \}

Where \(\text{rmq}_{excess}^\pm\) returns the rightmost position of the minimal value.
\[ O(n), O(1) \] solution \( (2n + o(n) \text{ bits}) \)

- Node with inorder \( i \) in CT becomes node with preorder \( i + 1 \) in the general tree
- So we can identify the node of array element \( i \) by selecting the \((i + 2)\)th opening parenthesis in the balanced parentheses sequence of the general tree (+1 for the added root node, +1 for index shift by one)
- Let \( BP \) be the balanced parentheses sequence of the general tree

For \( 0 \leq i \leq j < n \) we get:

00 \( \text{rmq}_A(i,j) \)
01 \( \text{ipos} \leftarrow \text{select}(i + 2, (, BP) \)
02 \( \text{jpos} \leftarrow \text{select}(j + 2, (, BP) \)
03 \( \text{return } \text{rank}(\text{rmq}^{\pm 1}_{\text{excess}}(\text{ipos} − 1, \text{jpos}), (, BP) − 1 \)

Where \( \text{rmq}^{\pm 1}_{\text{excess}} \) returns the rightmost position of the minimal value.
\( O(n), O(1) \) solution \( (2n + o(n) \text{ bits}) \)

- Node with inorder \( i \) in CT becomes node with preorder \( i + 1 \) in the general tree
- So we can identify the node of array element \( i \) by selecting the \( (i + 2) \text{th} \) opening parenthesis in the balanced parentheses sequence of the general tree (+1 for the added root node, +1 for index shift by one)
- Let \( BP \) be the balanced parentheses sequence of the general tree

For \( 0 \leq i \leq j < n \) we get:

00 \hspace{1em} \textbf{rmq}_{A}(i,j)
01 \hspace{1em} i\text{pos} \leftarrow \text{select}(i + 2, (, BP)
02 \hspace{1em} j\text{pos} \leftarrow \text{select}(j + 2, (, BP)
03 \hspace{1em} \textbf{return} \hspace{1em} \text{rank}(\text{rmq}_{\text{excess}}^{\pm 1}(\text{ipos} - 1, \text{jpos}), (, BP) - 1

Where \( \text{rmq}_{\text{excess}}^{\pm 1} \) returns the rightmost position of the minimal value.
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Proof sketch: Let \( v \) and \( w \) be the nodes of \( A[i] \) and \( A[j] \) and \( z \) be the LCA of \( v \) and \( w \) in the Cartesian Tree. Node \( v \) is in the left subtree and \( w \) in the right subtree of \( z \).
\( \langle O(n), O(1) \rangle \) solution \((2n + o(n) \text{ bits})\)

Proof sketch: Let \( v \) and \( w \) be the nodes of \( A[i] \) and \( A[j] \) and \( z \) be the LCA of \( v \) and \( w \) in the Cartesian Tree. Node \( v \) is in the left subtree and \( w \) in the right subtree of \( z \).
\[ O(n), O(1) \] solution \((2n + o(n) \text{ bits})\)

First case: \(v \neq z\) and \(w \neq z\)
- \(v\)'s opening parenthesis is the green area
- \(w\)'s opening parenthesis is the red area
- The rightmost \(\pm 1\ RMQ\) will return the position \(p\) of the closing parenthesis of \(z\).
- \(z\)'s opening parenthesis is at position \(p + 1\) by construction
- \(r = rank(p) - 1\) corresponds to the preorder number of \(z\) in the general tree
- Which in turns corresponds to the inorder number in \(CT\)
- Which in turns corresponds to the index of the minimum in \(A[i, j]\)
\langle O(n), O(1) \rangle \textbf{ solution} \ (2n + o(n) \text{ bits})

Second case: \( v \neq z \) and \( w = z \):

- \( v \)'s opening parenthesis is the green area
- \( w \)'s opening parenthesis is \( z \)'s opening parenthesis now
- the (rightmost) \( \pm 1 \text{ RMQ} \) will return the position \( p \) of the closing parenthesis of \( z \)
- \( z \)'s opening parenthesis is at position \( p + 1 \) by construction
- \( r = \text{rank}(p) - 1 \) corresponds to the preorder number of \( z \) in the general tree
- which in turns corresponds to the inorder number in \( \text{CT} \)
- which in turns corresponds to the index of the minimum in \( A[i, j] \)
Third case: $v = z$ and $w \neq z$:

- $v$'s opening parenthesis is $z$'s opening parenthesis now
- $w$'s opening parenthesis is the red area
- The (rightmost) $1 \text{RMQ}$ will return the position $p$ of the closing parenthesis of $z_i$
- $z$'s opening parenthesis is at position $p + 1$ by construction
- $r = \text{rank}(p) - 1$ corresponds to the preorder number of $z$ in the general tree
- which in turns corresponds to the inorder number in CT
- which in turns corresponds to the index of the minimum in $A[i, j]$
Third case: \( v = z \) and \( w = z \):
- \( v \)'s opening parenthesis is \( z \)'s opening parenthesis now
- \( w \)'s opening parenthesis is \( z \)'s opening parenthesis now
- The (rightmost) \( \pm 1 \) RMQ will return the position \( p \) of the closing parenthesis of \( z \).
- \( z \)'s opening parenthesis is at position \( p + 1 \) by construction
- \( r = \text{rank}(p) - 1 \) corresponds to the preorder number of \( z \) in the general tree
- Which in turns corresponds to the inorder number in CT
- Which in turns corresponds to the index of the minimum in \( A[i, j] \)
Range Minimum Queries (RMQ)s over an array $A$ can be answered in constant time after preprocessing a $2n + o(n)$ space data structure in linear time.

Applications:
- Compressed suffix trees
- Document retrieval
- Weighted query completion
- ...