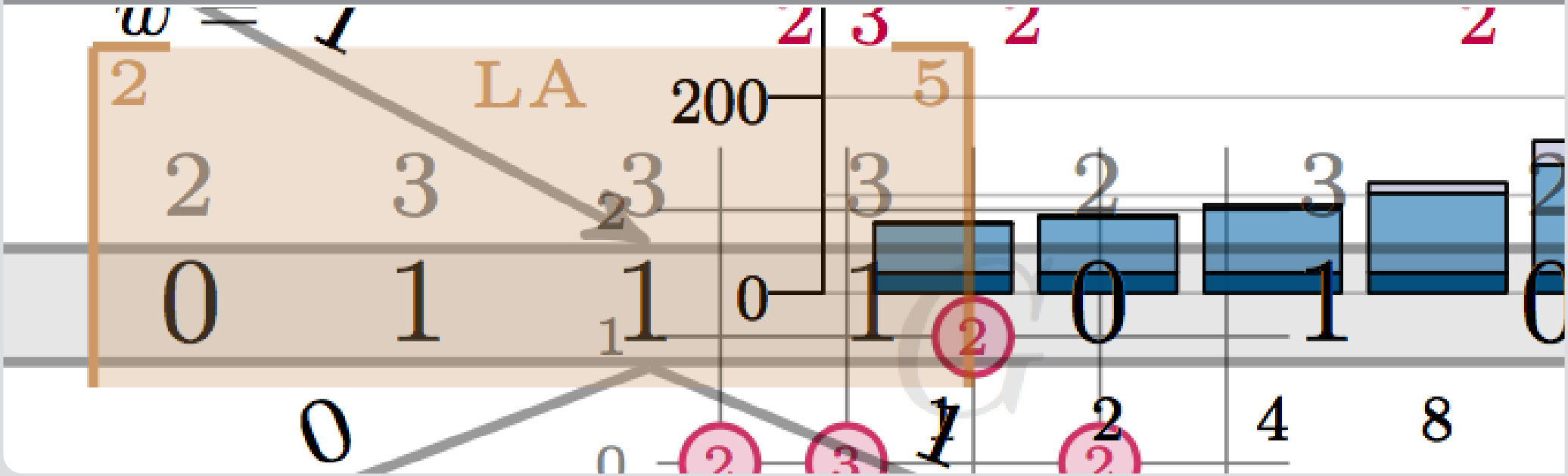
Karlsruhe Institute of Technology

Algorithmen II

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



Range minimum queries (RMQs)

Karlsruhe Institute of Technology

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

$A =$

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

Range minimum queries (RMQs)

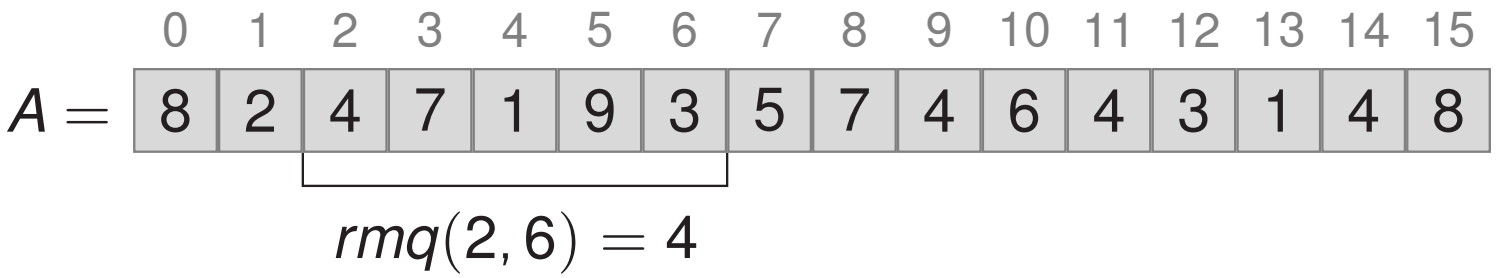
Karlsruhe Institute of Technology

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



Range minimum queries (RMQs)

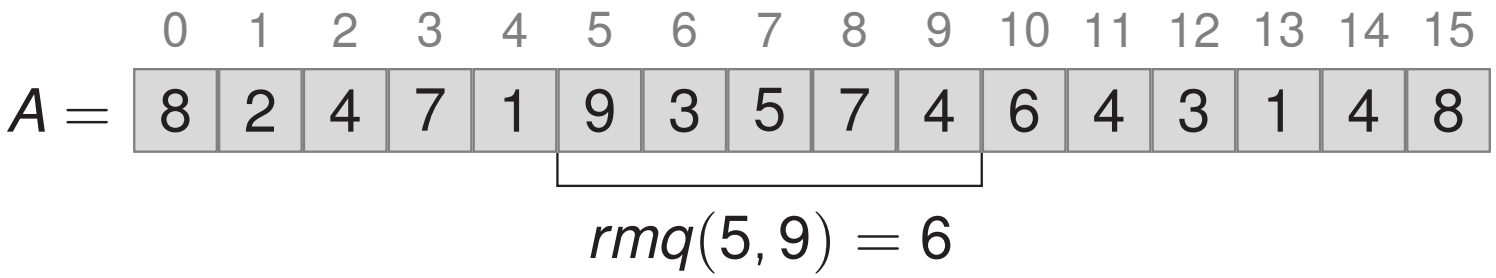
Karlsruhe Institute of Technology

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



Range minimum queries (RMQs)

Karlsruhe Institute of Technology

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

$A =$

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

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

Overview

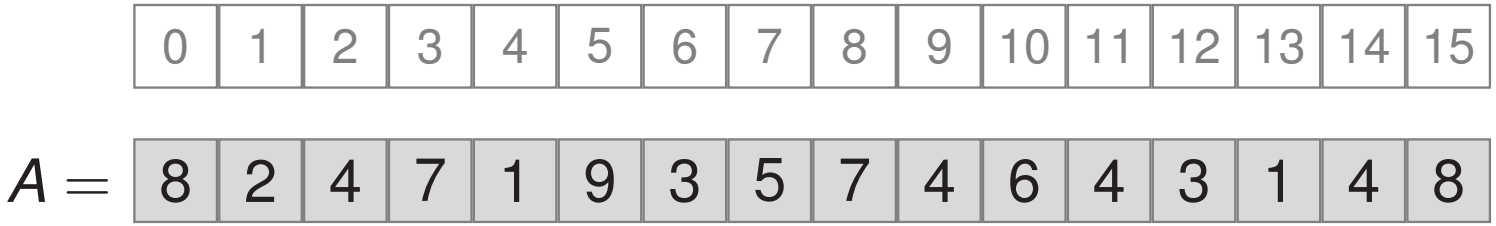
- Notation: Complexity of an algorithm is denoted with $\langle f(n), g(n) \rangle$, where $f(n)$ is preprocessing time and $g(n)$ query time.
- Different solutions:
  - naïve approach 1: $\langle O(n^2), O(1) \rangle$ using $O(n^2)$ words of space
  - naïve approach 2: $\langle O(1), O(n) \rangle$
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

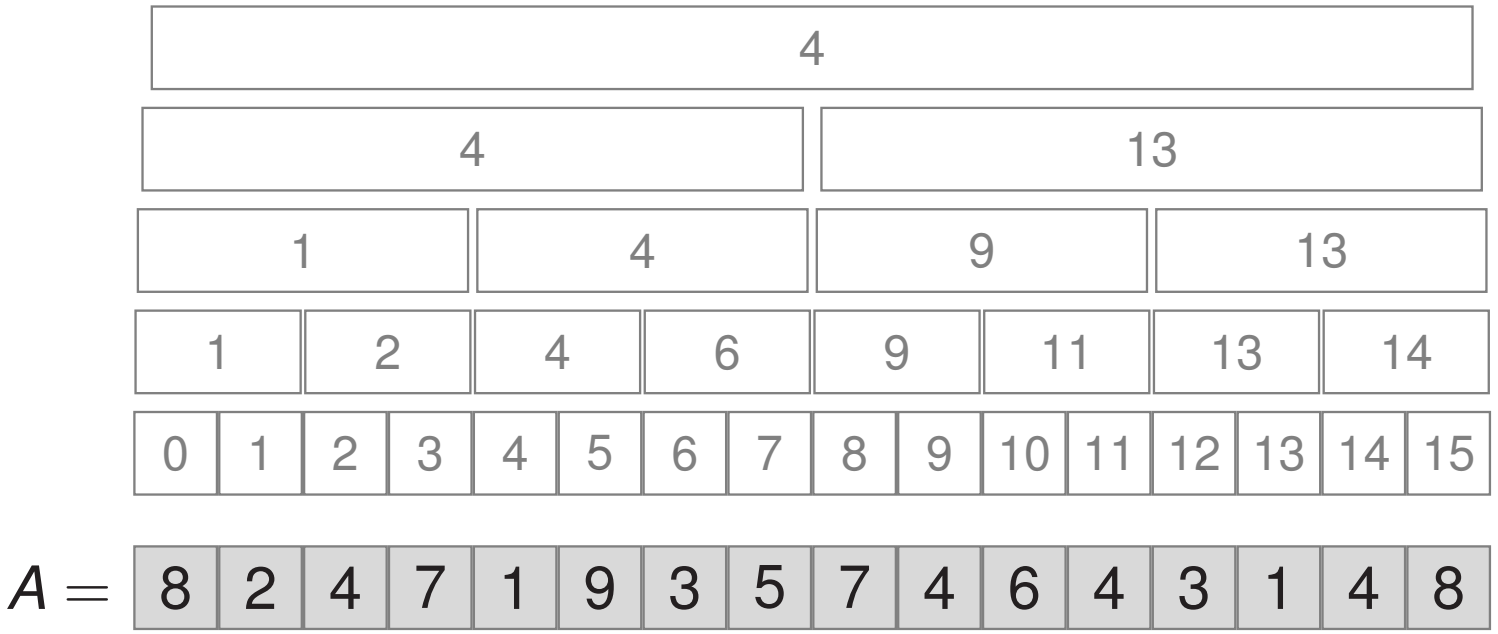
The last two solutions do not require access to the original array A.

$\langle O(n), O(\log n) \rangle$ – solution #1

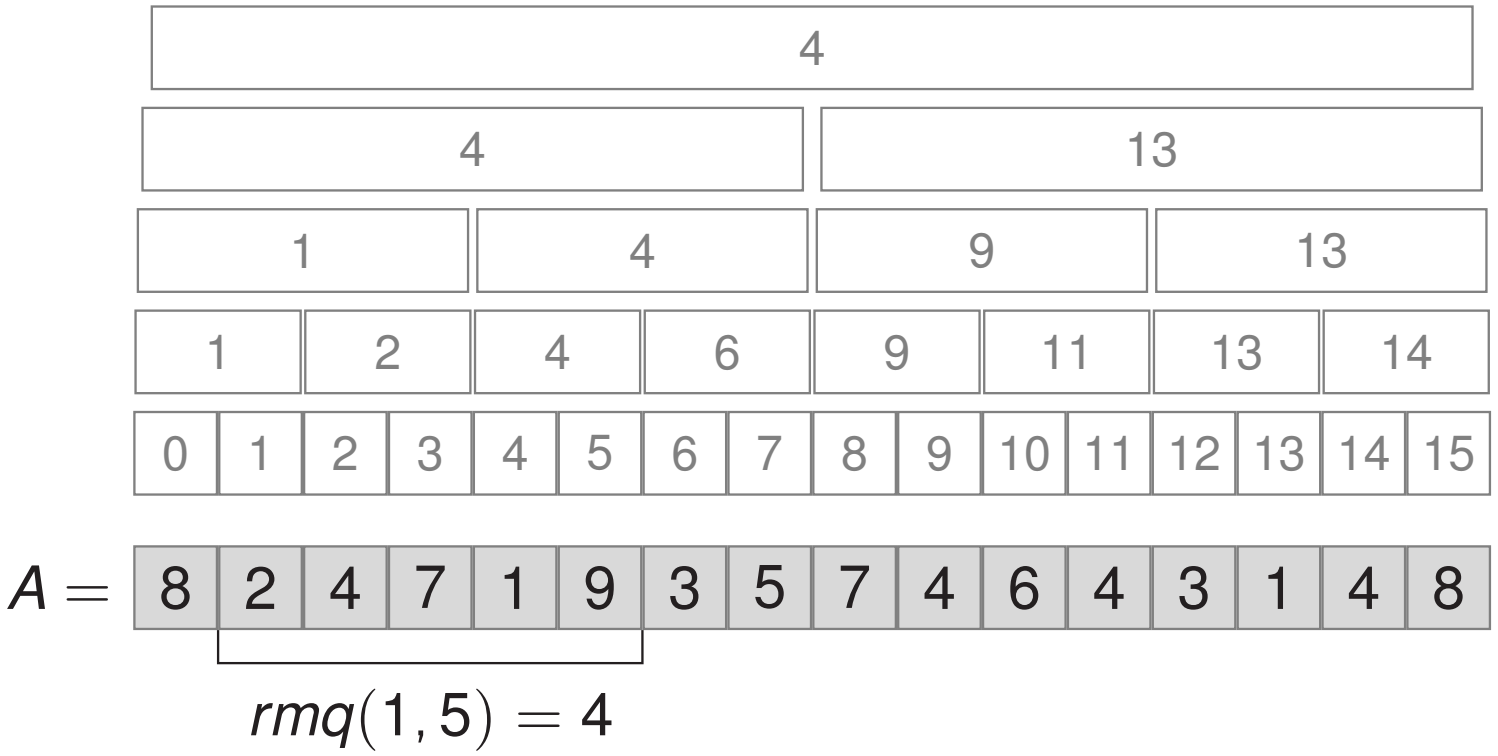
Karlsruhe Institute of Technology



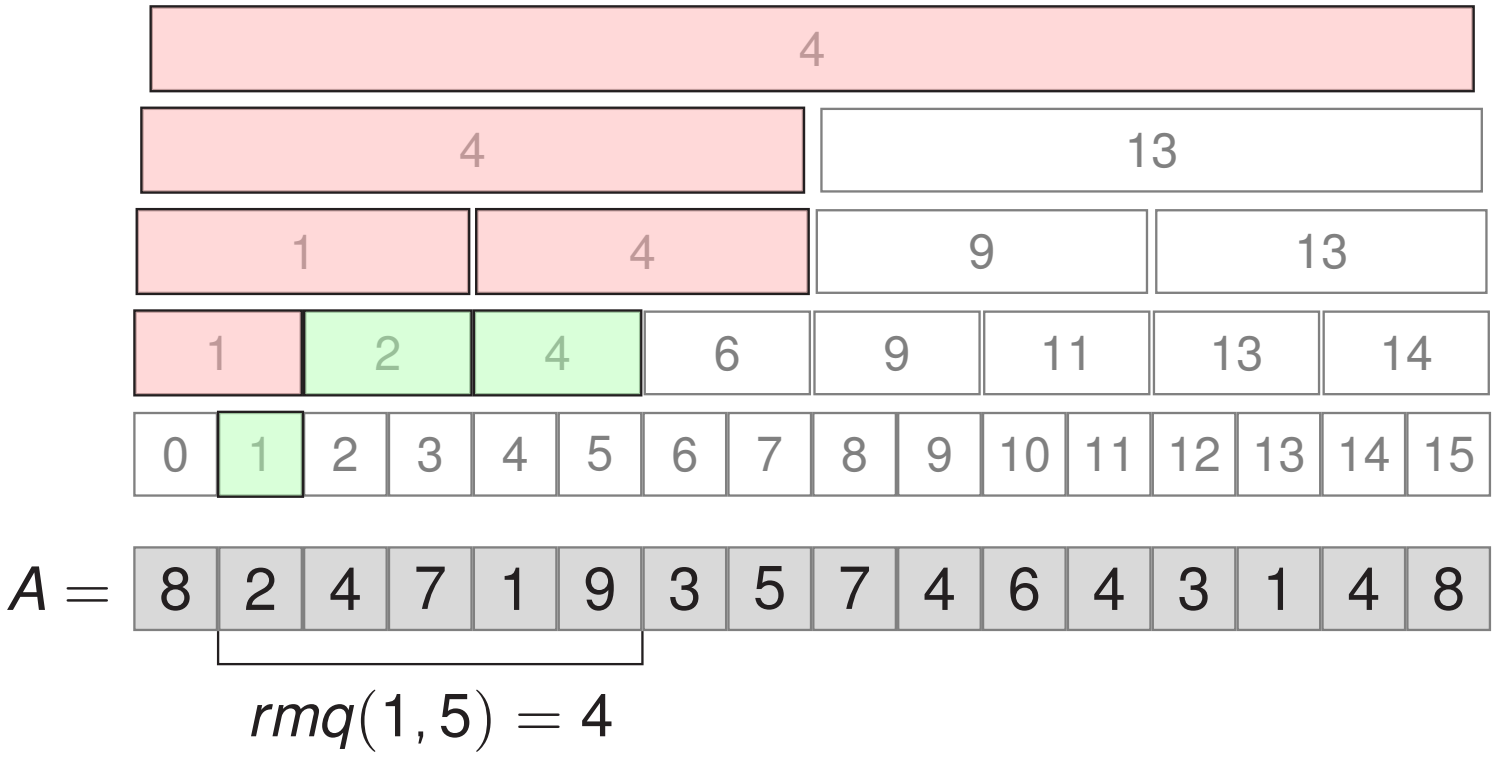
$\langle O(n), O(\log n) \rangle$ – solution #1



$\langle O(n), O(\log n) \rangle$ – solution #1



$\langle O(n), O(\log n) \rangle$ – solution #1



$\langle O(n), O(\log n) \rangle$ – solution #1

Karlsruhe Institute of Technology

- 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.

$\langle O(n \log n), O(1) \rangle$ – solution #2

- For each item $A[i]$ store an array $M_i[0, \log n]$.
- $M_i[j] = rmq_A(i, i + 2^j - 1)$
- 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

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

Question: How can k be determined in constant time?

$\langle O(n \log\log n), O(1) \rangle$ solution

- Split A into $t = \frac{n}{\log n}$ blocks $B_0, ..., B_{t-1}$. B spans $O(\log n)$ items of A.
- Create an array $S[0, t-1]$ with $S[i] = min\{x \in B_i\}$
- Build rmq structure #2 for S
- For each block B_i of $O(\log n)$ elements build rmq structure #2
- Total space: $O(n) + O(n \log\log n)$

Querying

- Determine blocks $B_{\ell'}$, $B_{r'}$ which contain ℓ and r
- Calculate $m = rmq_S(\ell' + 1, r' - 1)$
- Let k_0, k_1, k_2 be the results of the RMQs in blocks ℓ', r', and m relative to A
- Return $\arg\min_{k_i} A[k_i]$ for $0 \leq k \leq 3$

$\langle O(n), O(1) \rangle$ solution

Karlsruhe Institute of Technology

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

$\langle O(n), O(1) \rangle$ solution

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 $\frac{1}{s+1}\binom{2s}{s}$ 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 & $\pm 1 RMQ$

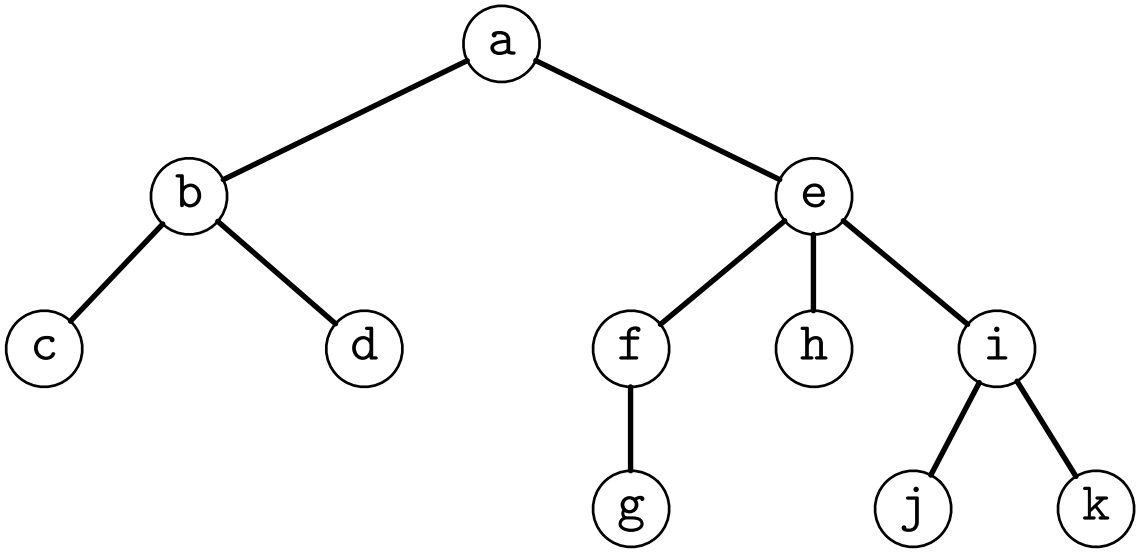
LCA (Lowest Common Ancestor)

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 & $\pm 1 RMQ$

LCA (Lowest Common Ancestor)

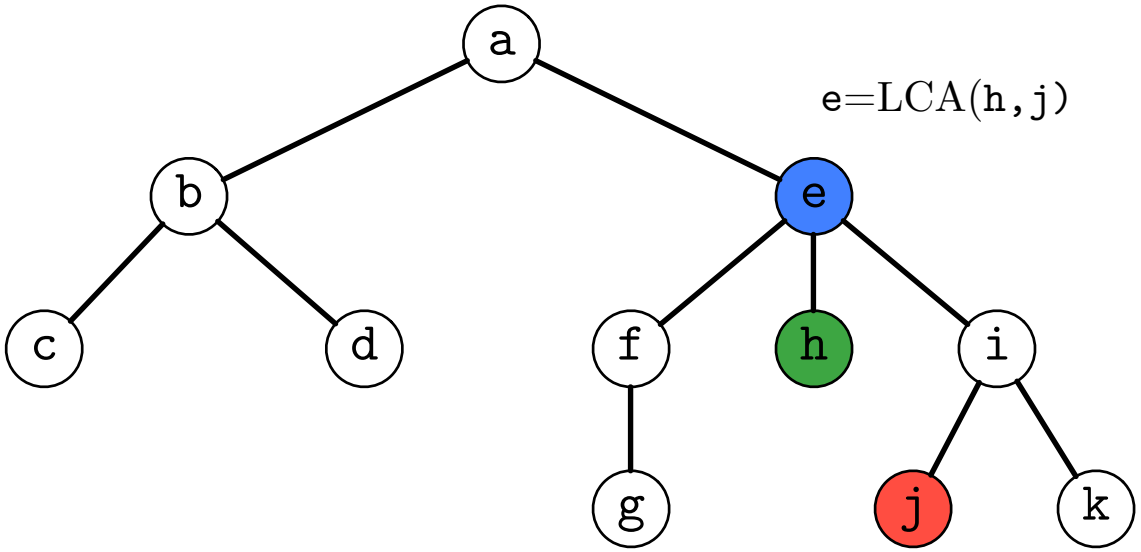
Given a rooted tree T of n nodes. For nodes v and w of T the query $LCA_T(v, w)$ returns the *lowest common ancestor* of u and v in T. I.e. the node which is (1) ancestor of v and w and (2) maximizes the distance to the root.



LCA & $\pm 1 RMQ$

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 & $\pm 1 RMQ$

Karlsruhe Institute of Technology

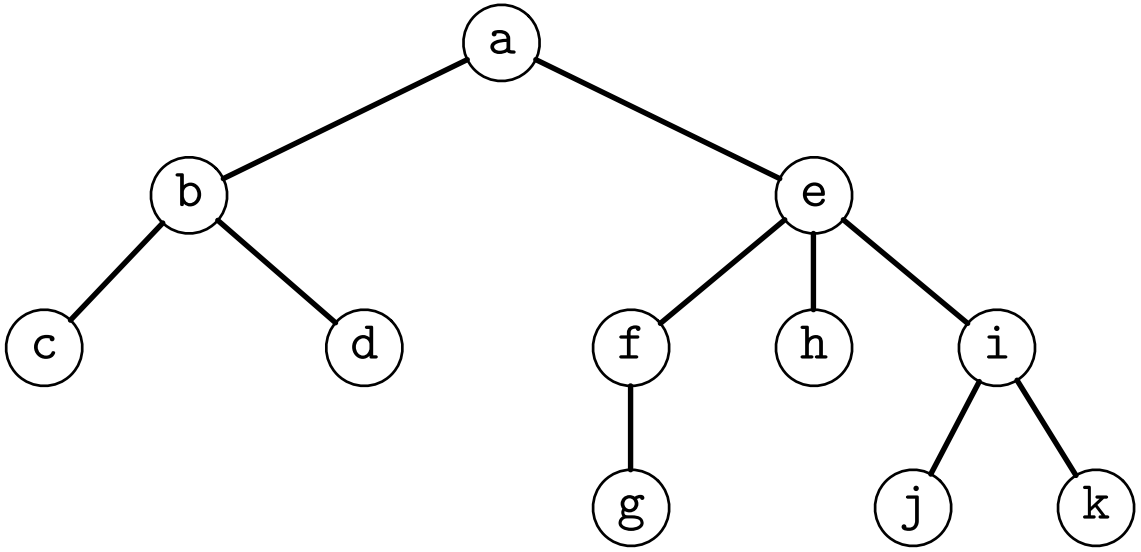
Lemma

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




```
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

    a b c d e f g  h  i  j  k
R = 0 1 2 4 7 8 9 12 14 15 17
```

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

    a b c d e f g  h  i  j  k
R = 0 1 2 4 7 8 9 12 14 15 17
```

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

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

    a b c d e f g  h  i  j  k
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 ± 1 restriction. This is called $\pm 1 RMQ$.

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

Karlsruhe Institute of Technology

```
                                  1
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
```

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

											1	
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 ($4n + o(n)$ bits)

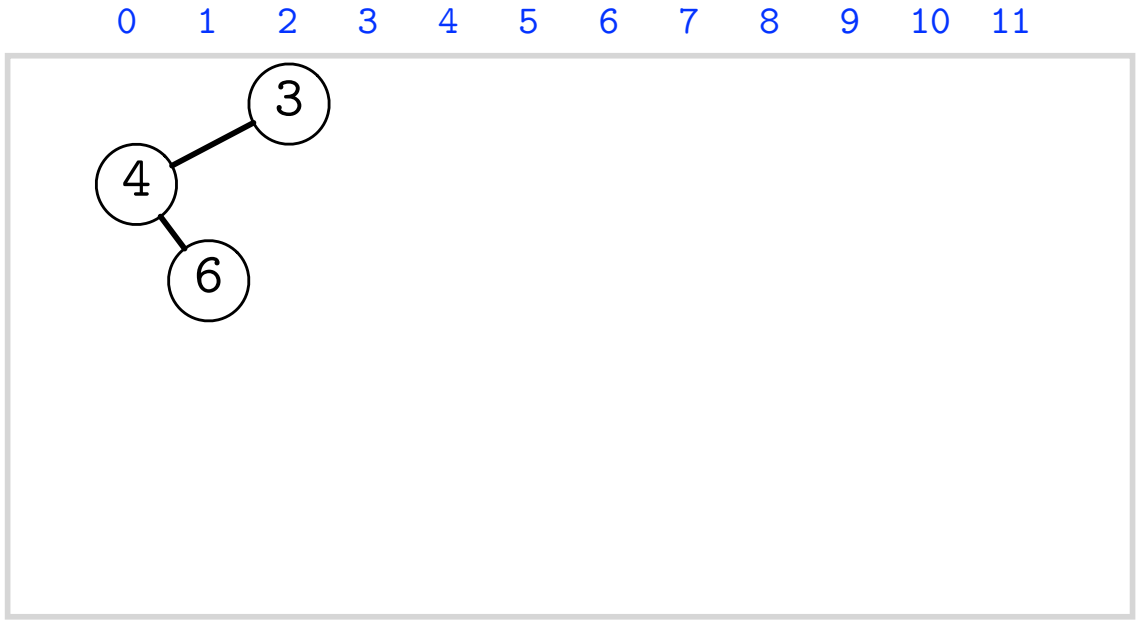
											1	
i	=	0	1	2	3	4	5	6	7	8	9	0	1
A	=	4	6	3	5	1	4	6	4	5	2	6	3

(1) Build Cartesian Tree

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

											1	
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

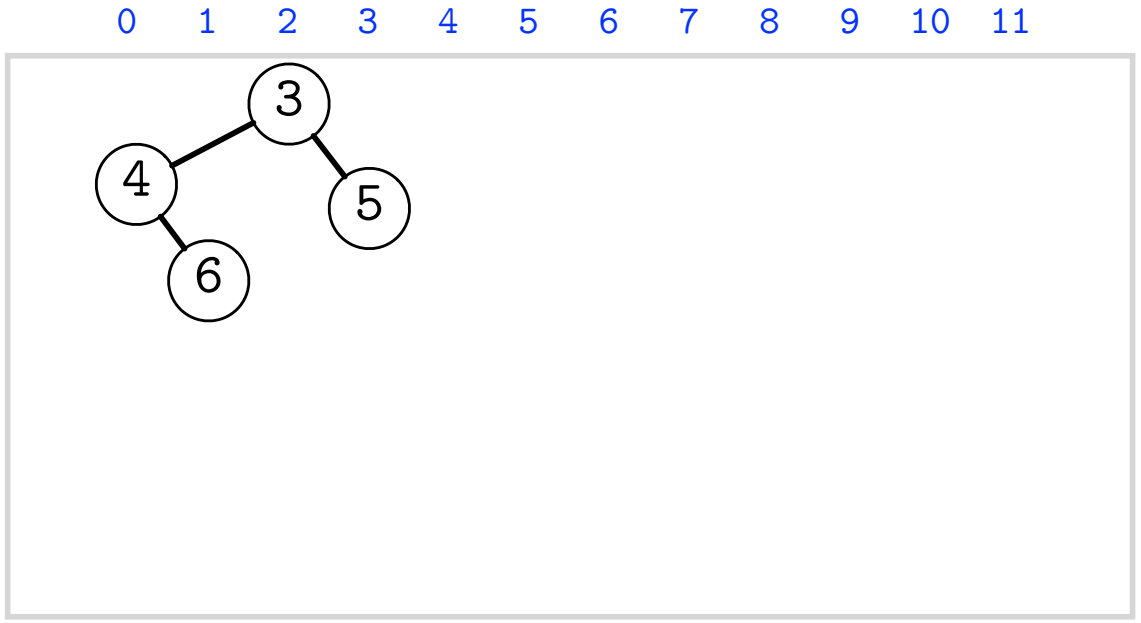
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

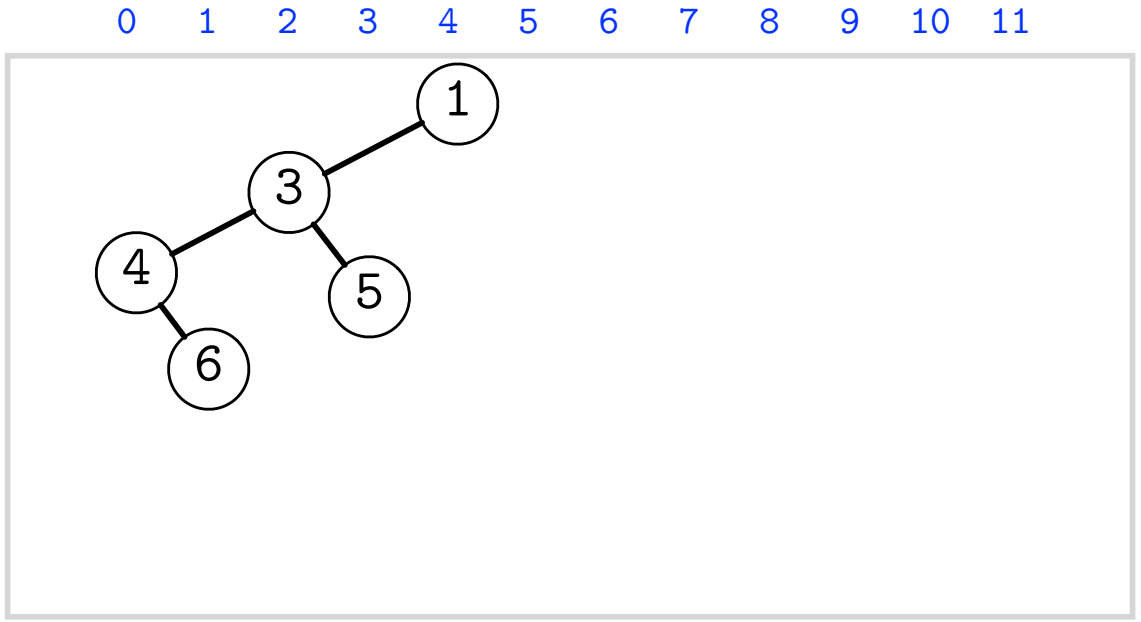
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

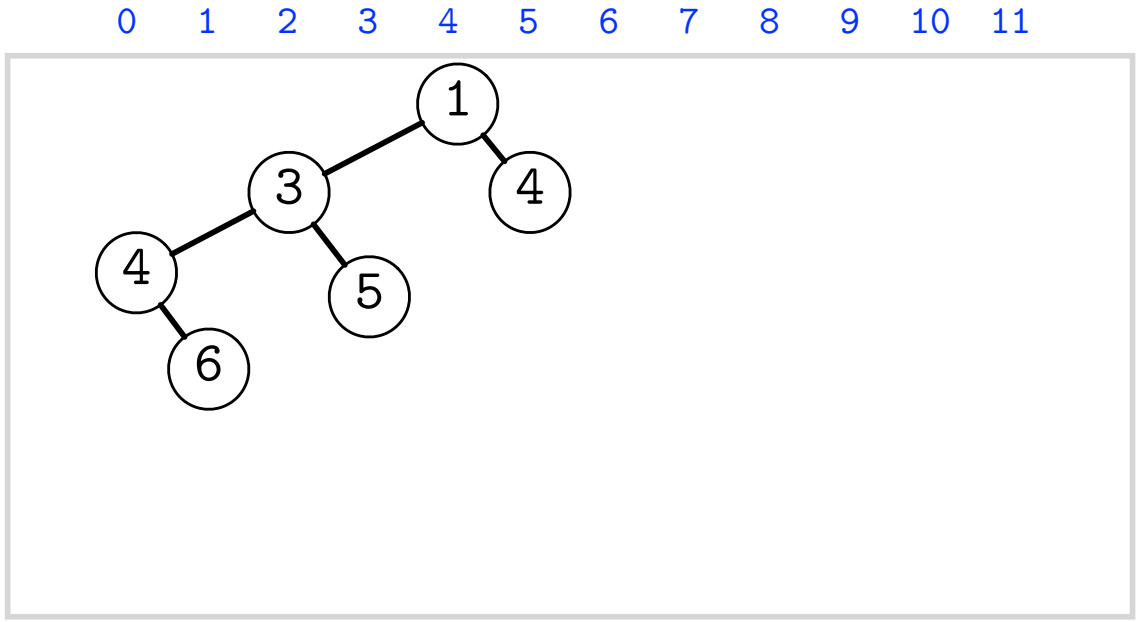
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

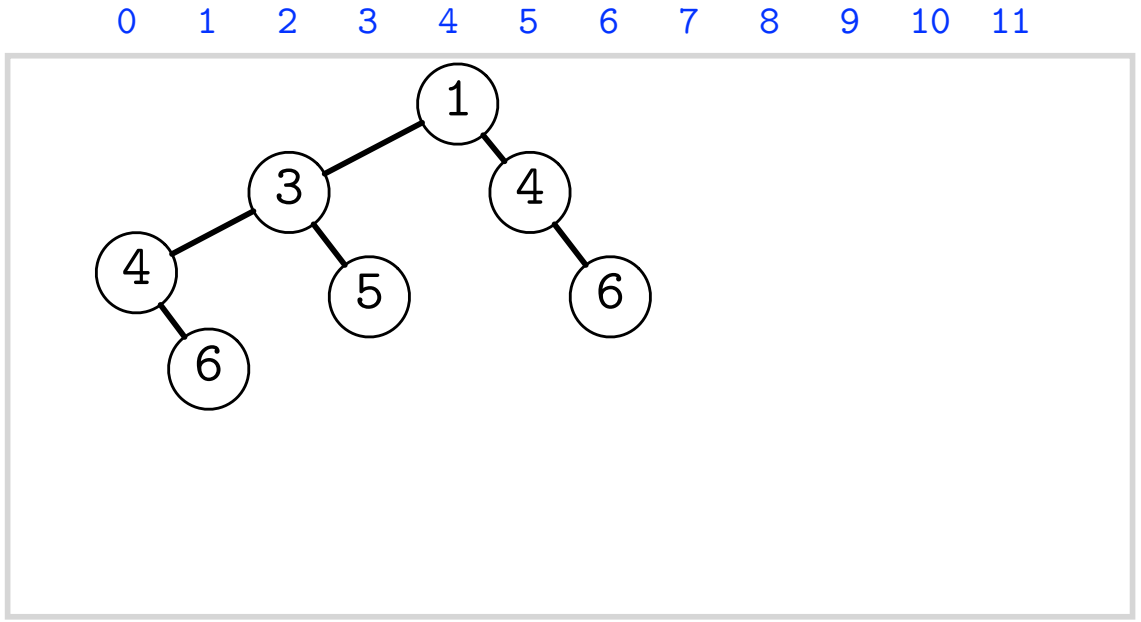
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

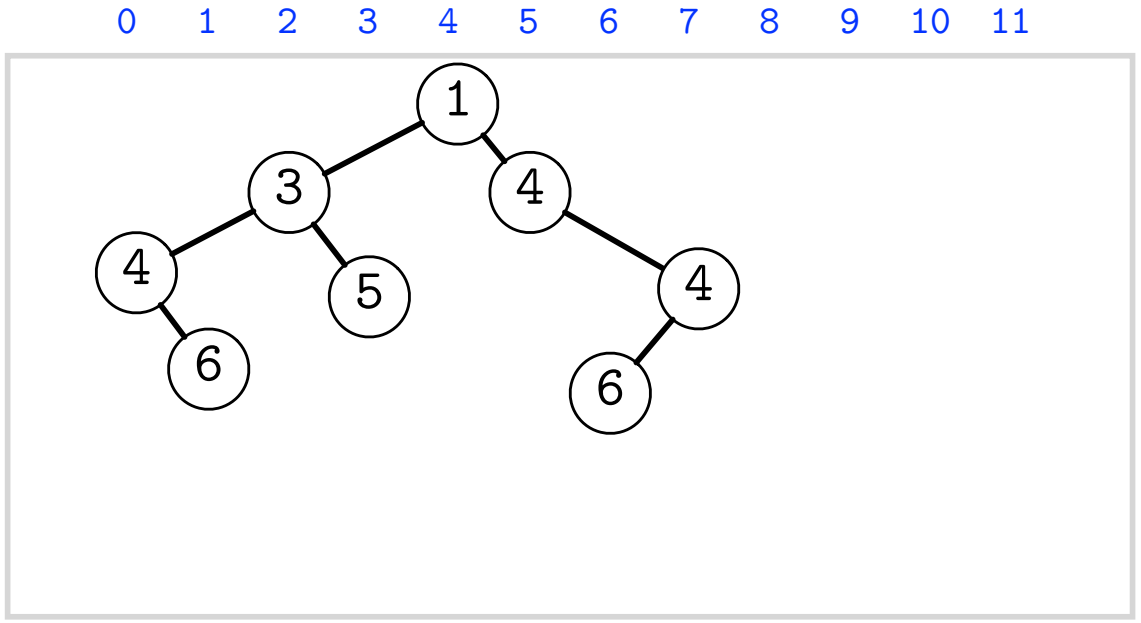
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

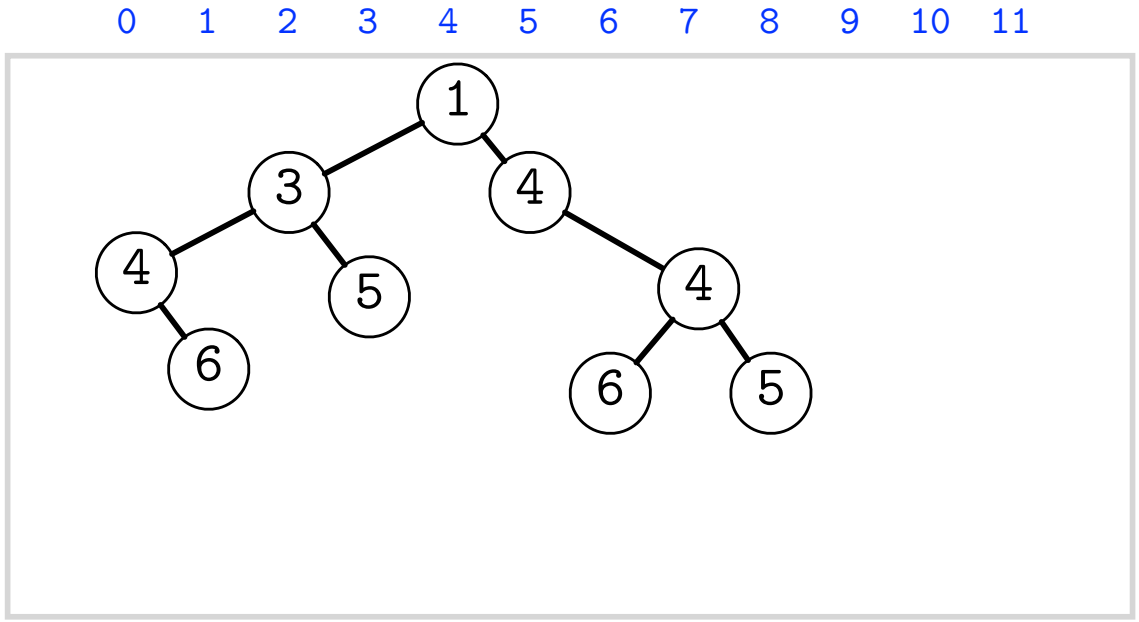
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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

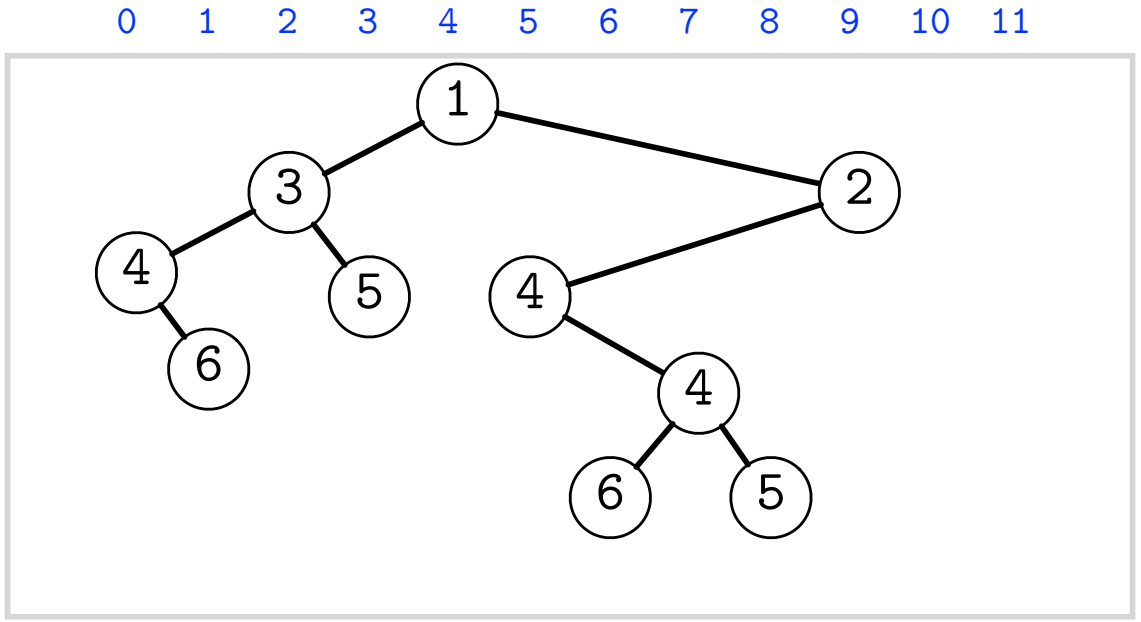


$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

Karlsruhe Institute of Technology

```
                                  1
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
```

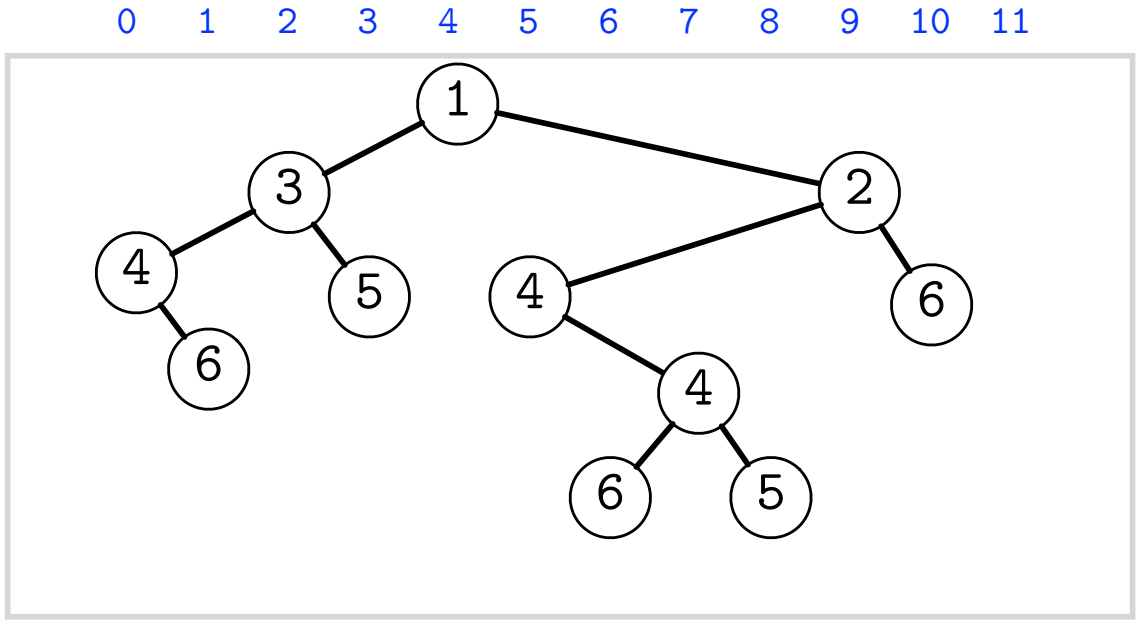
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

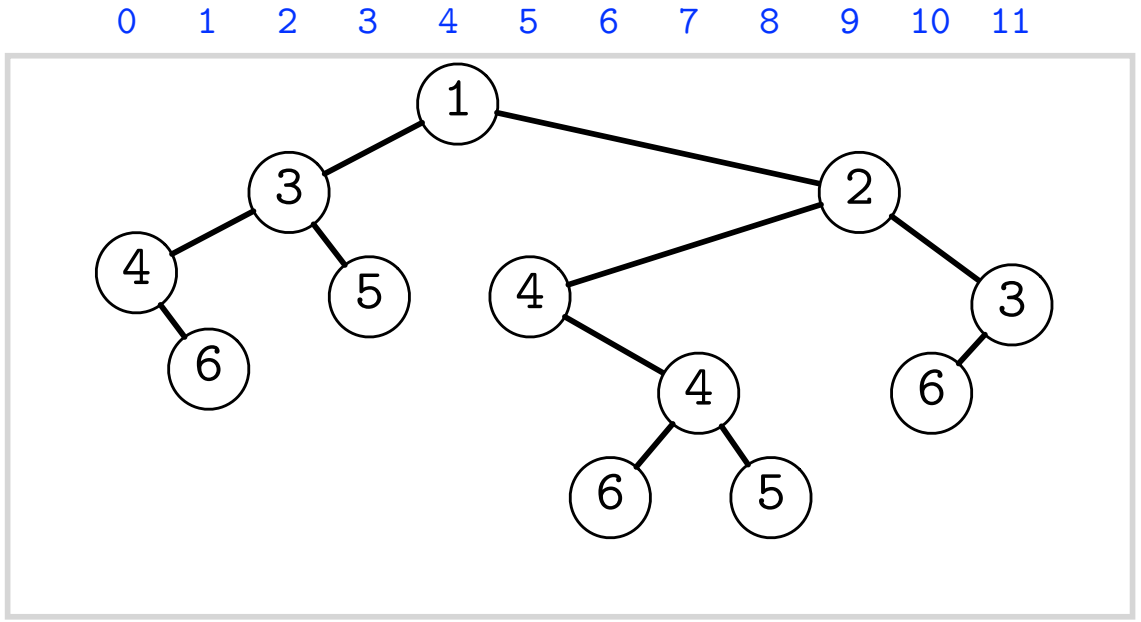
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

```
                                  1
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
```

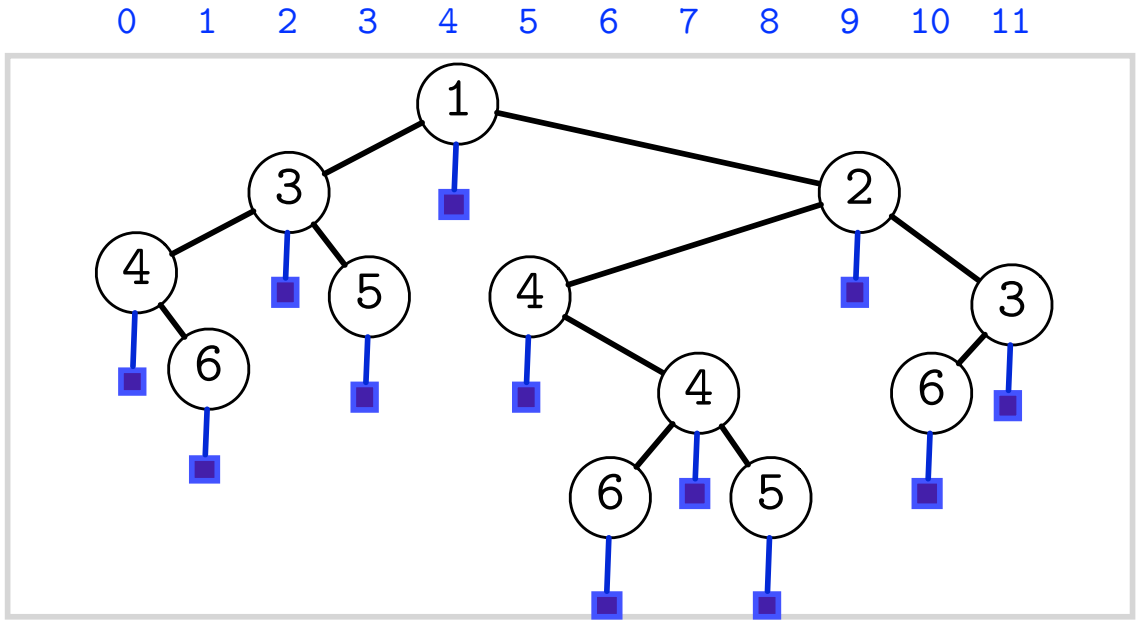
(1) Build Cartesian Tree



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

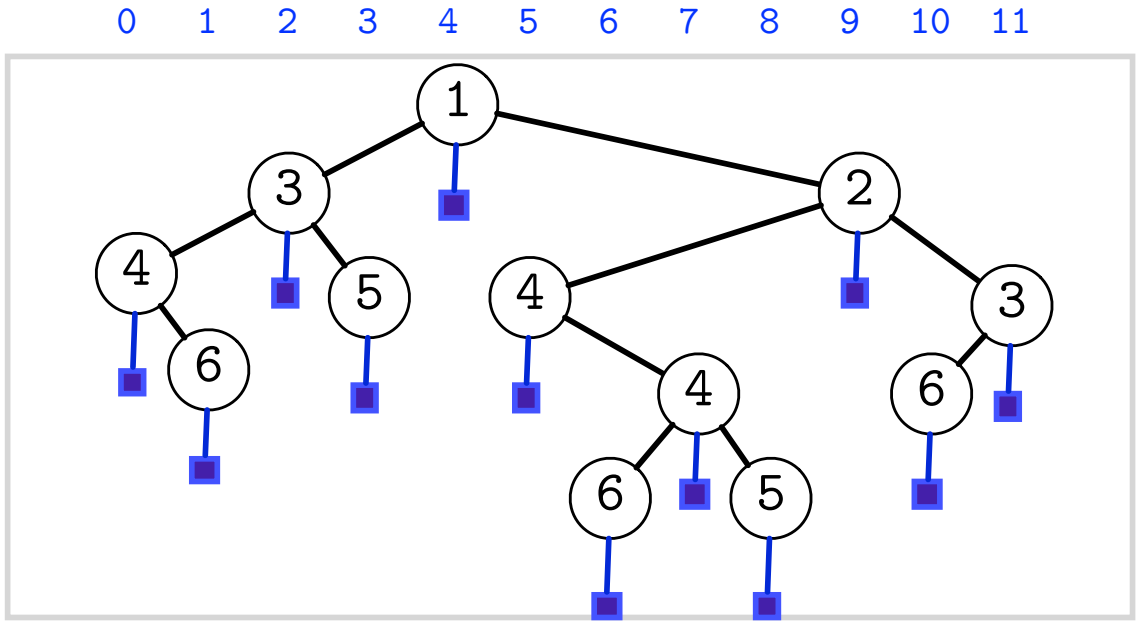
											1	
i	=	0	1	2	3	4	5	6	7	8	9	0	1
A	=	4	6	3	5	1	4	6	4	5	2	6	3

(2) Add a leaf to each node



$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

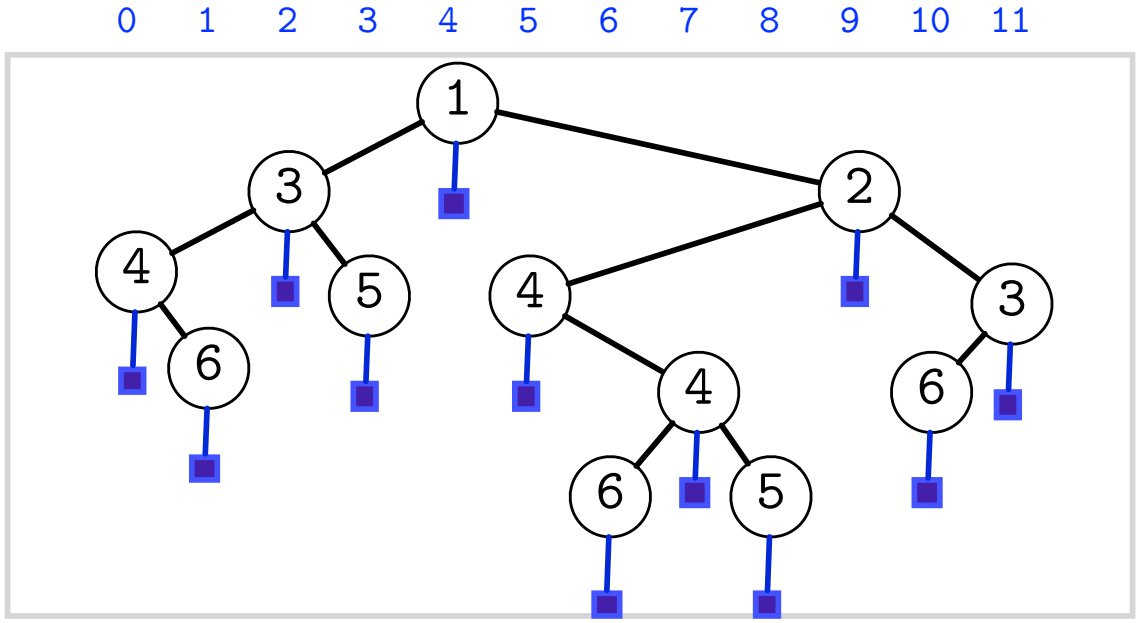


(3) DFS traversal to construct balanced parentheses sequence

$BP_{ext} =$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

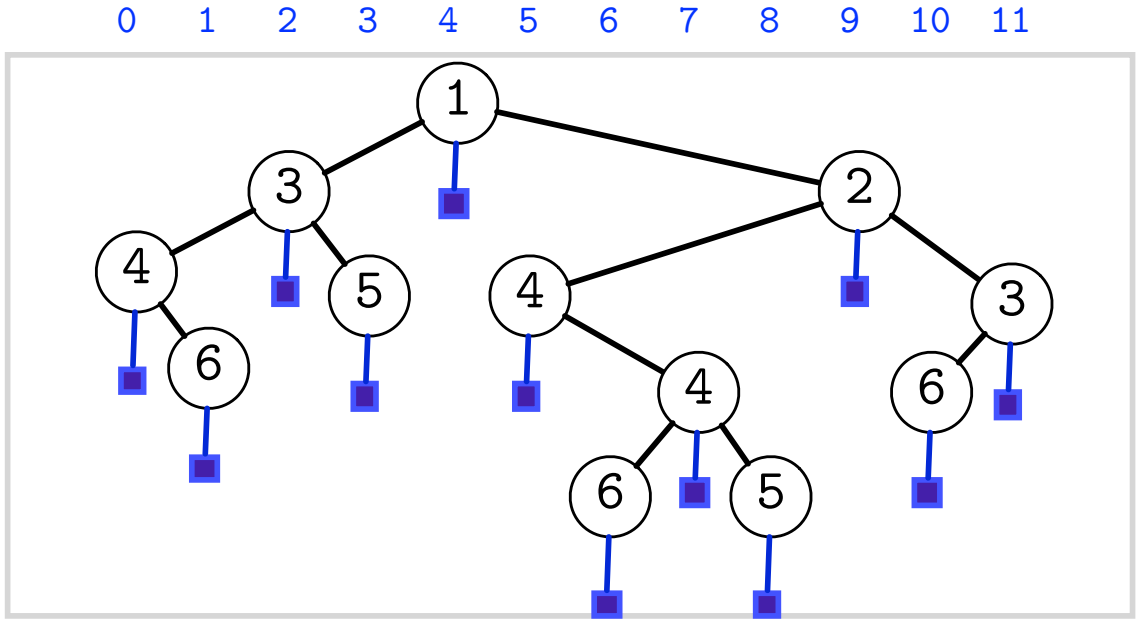


(3) DFS traversal to construct balanced parentheses sequence

$BP_{ext} = \overset{1}{(}$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

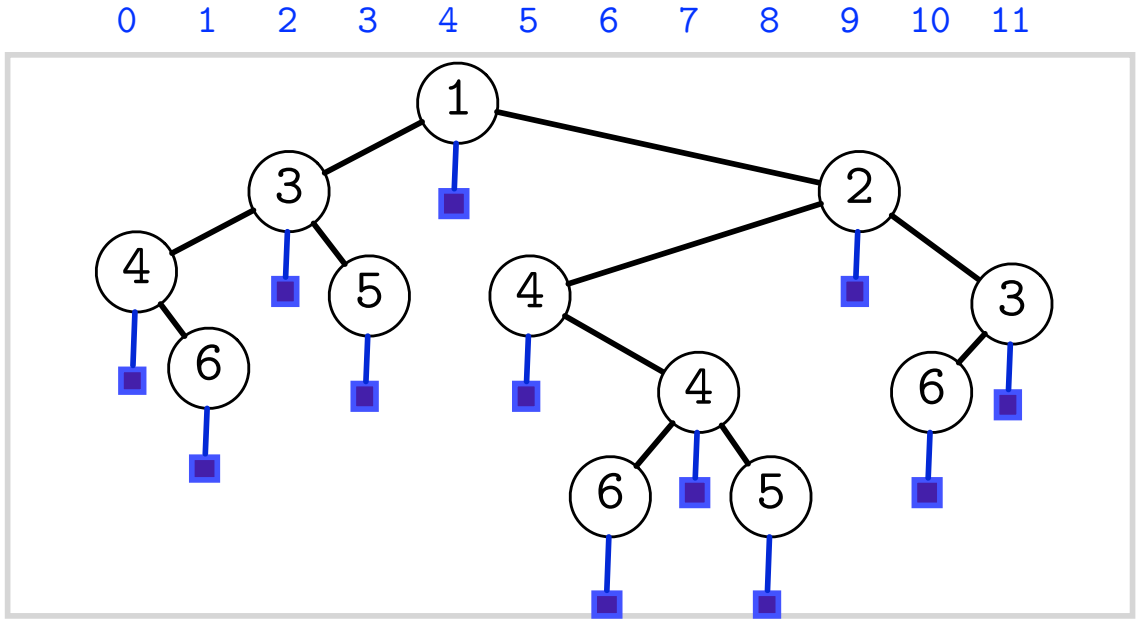


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1}{(}\overset{3}{(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

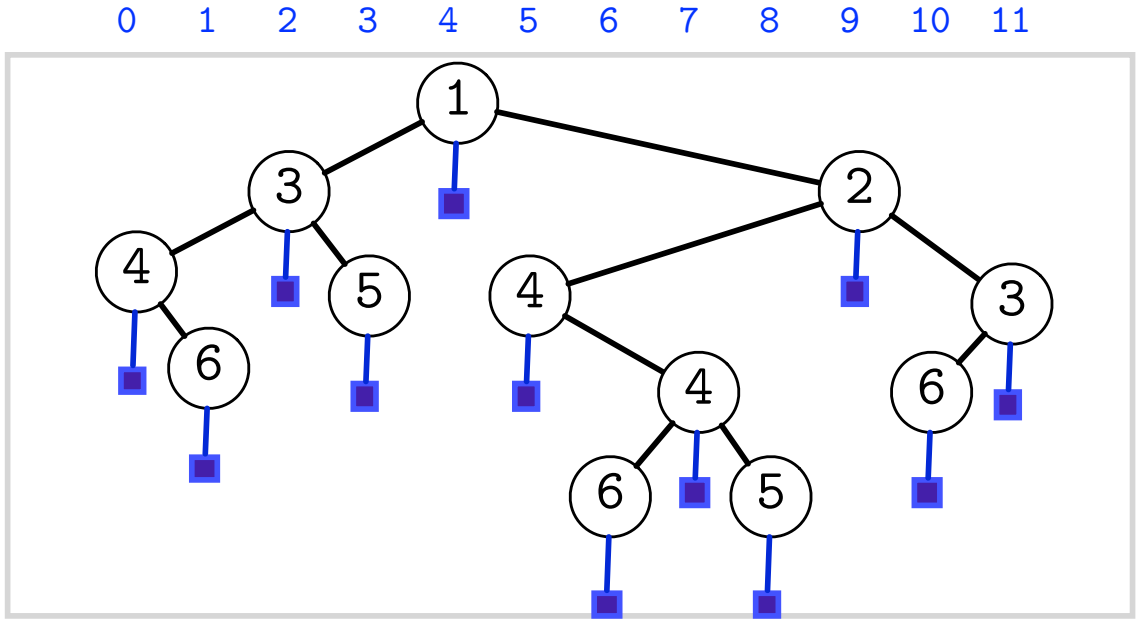


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(\,(\,(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

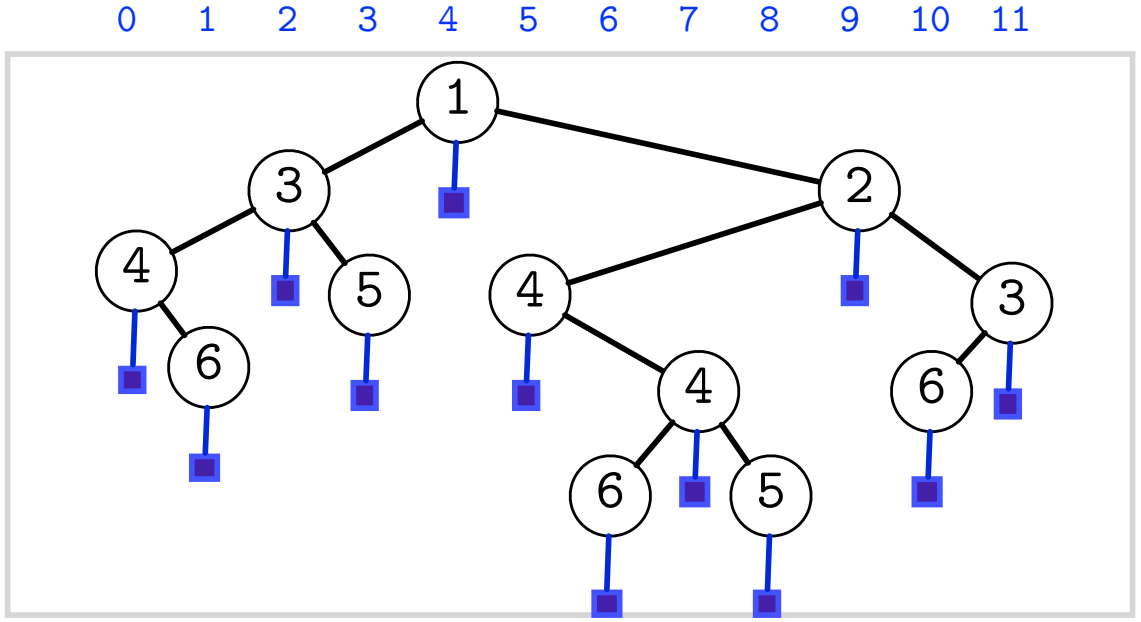


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(\,(\,(\,(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



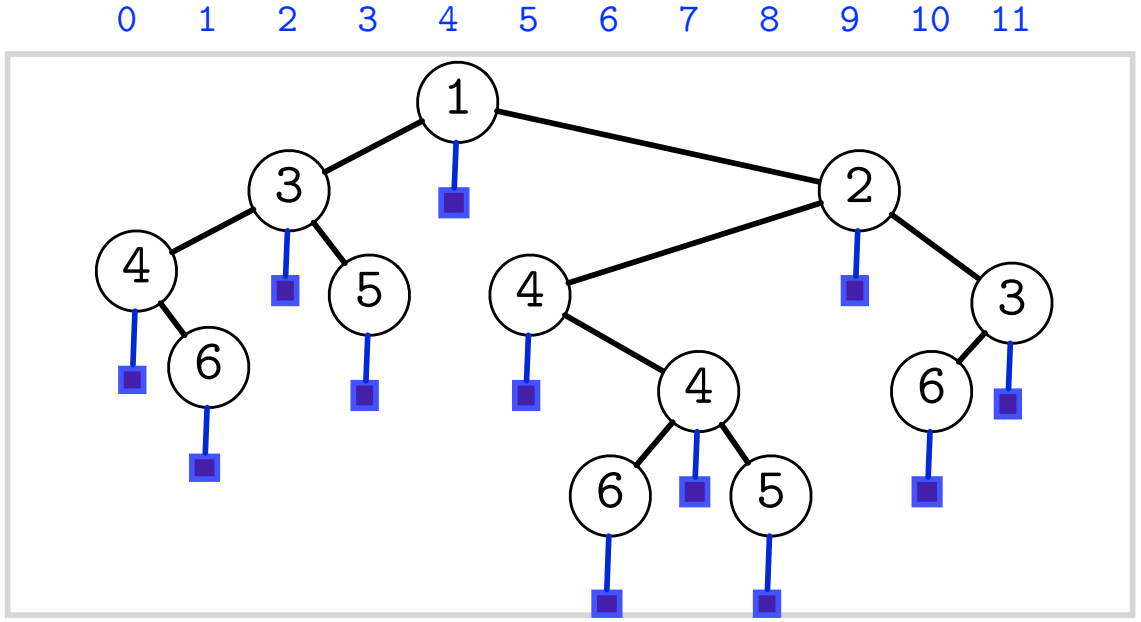
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(\,(\,(\,(\,)\,)}$$

0

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



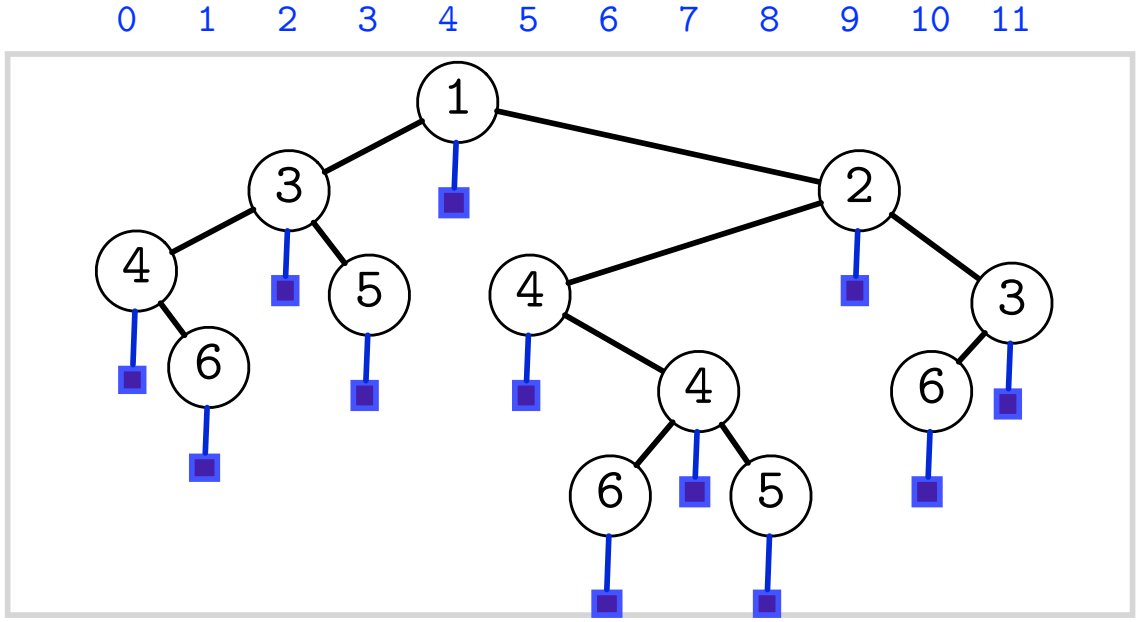
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,(\,(\,(\,)\,(\,\overset{6}{}$$

0

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



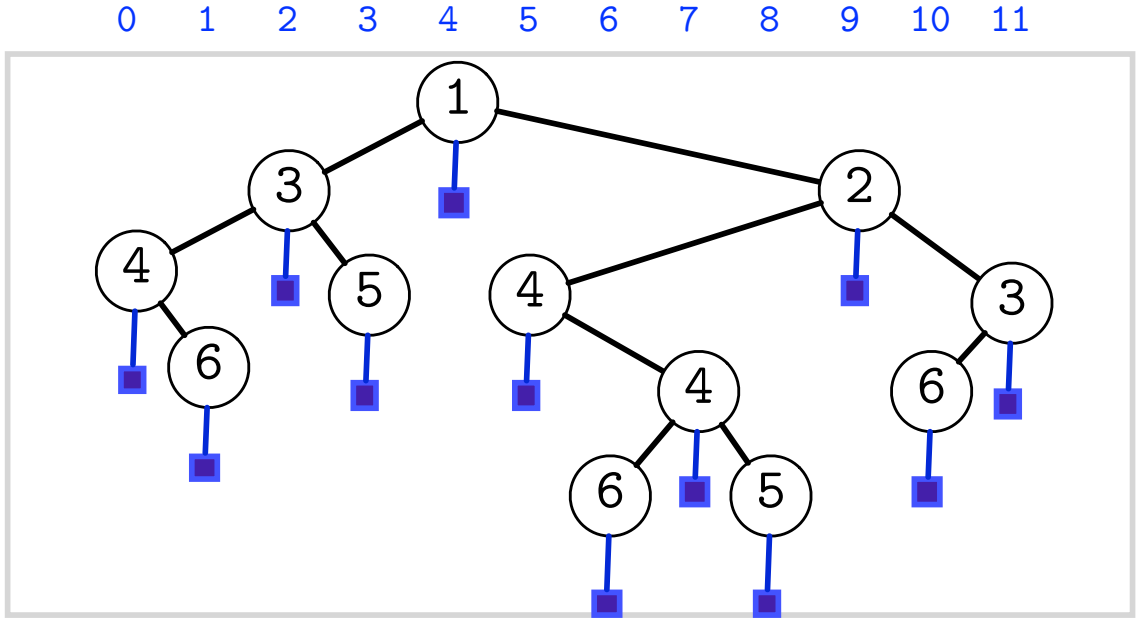
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,(\,(\,(\,)\,(\,\overset{6}{(}$$

0

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



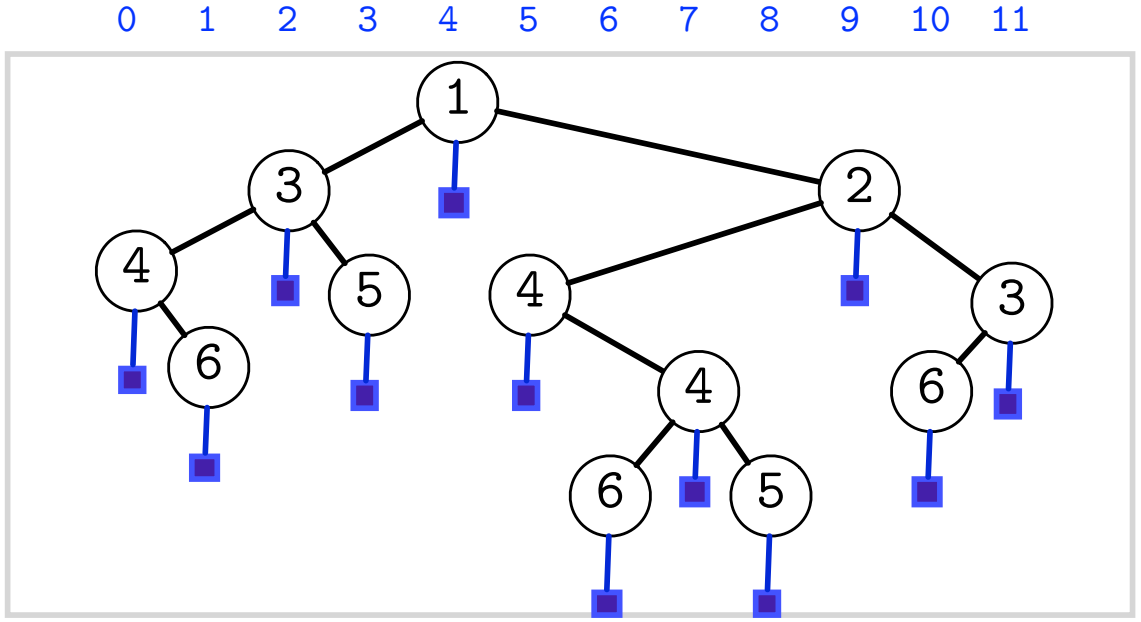
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,(\,(\,(\,)\,(\,\overset{6}{(}\,)$$

0 1

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

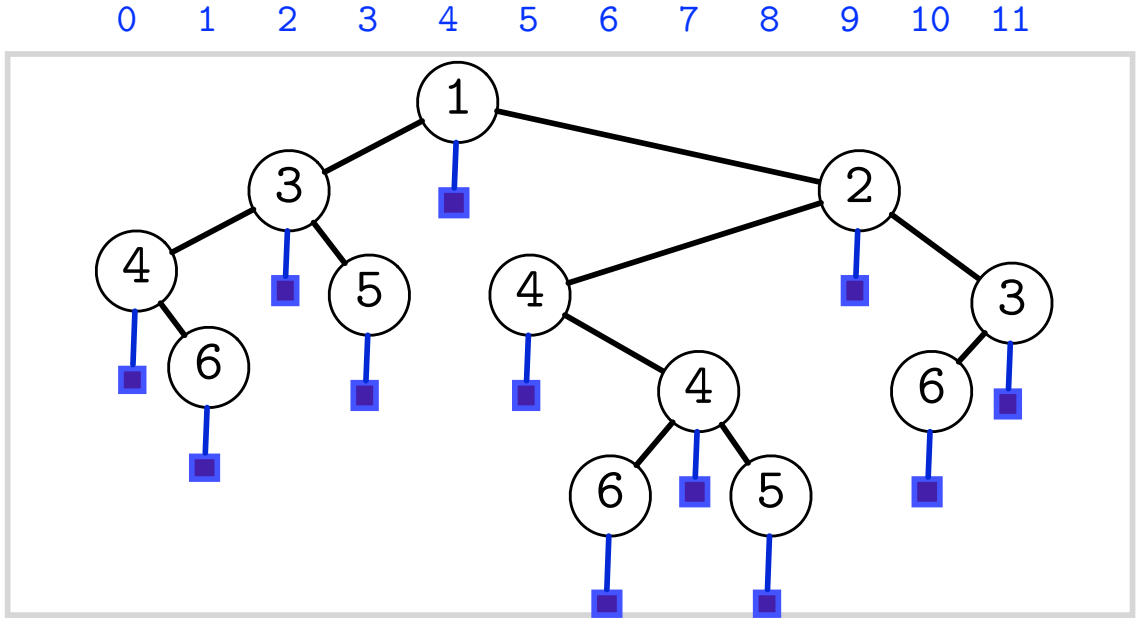


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,\overset{}{(}\,\overset{}{(}\,\overset{0}{(\,)}\,\overset{6}{(}\,\overset{1}{(\,)}\,\overset{6}{)}\,)$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

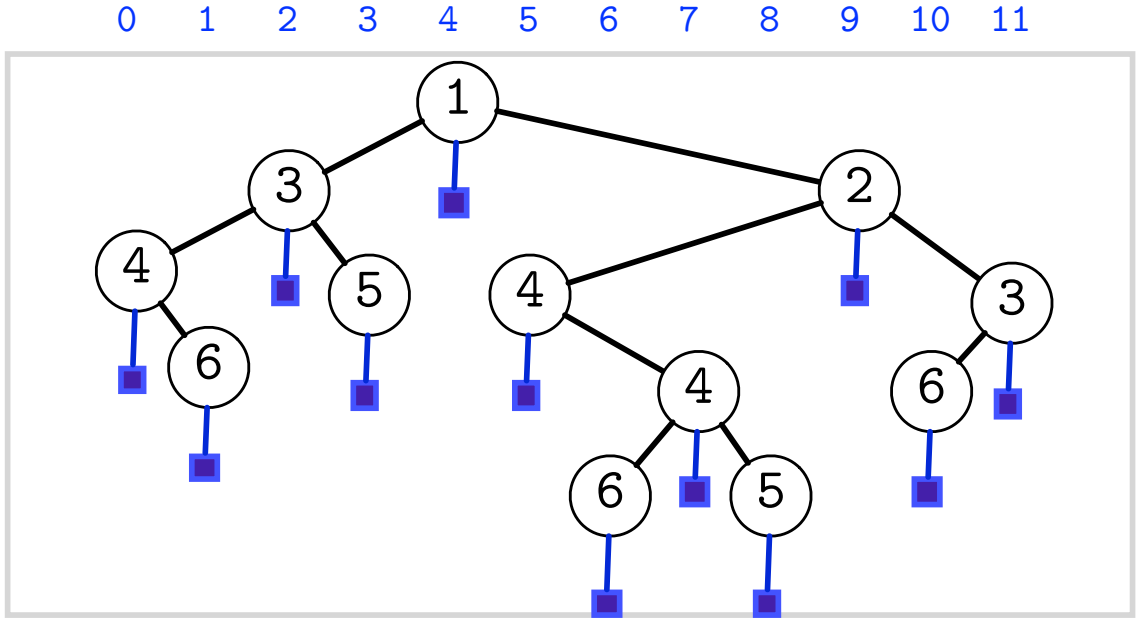


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,\overset{}{(}\,\overset{}{(}\,\overset{0}{(\,)}\,\overset{6}{(}\,\overset{1}{(\,)}\,\overset{6\ 4}{)}\,)\,)$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



(3) DFS traversal to construct balanced parentheses sequence

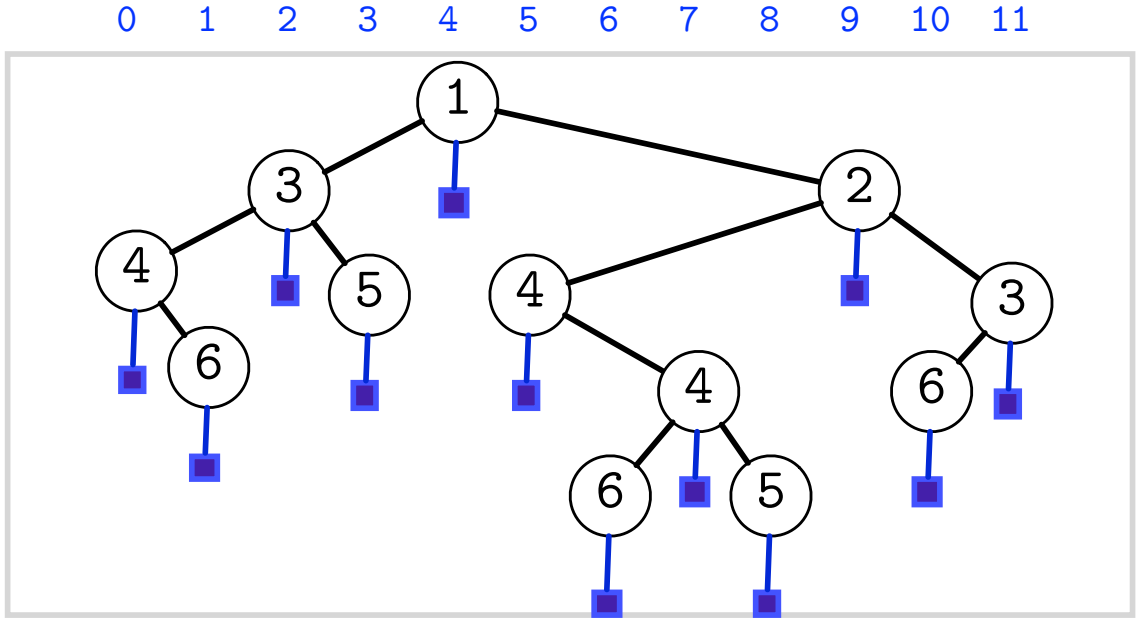
$$BP_{ext} = (((()(())))($$

1 3 4 6 6 4

0 1

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

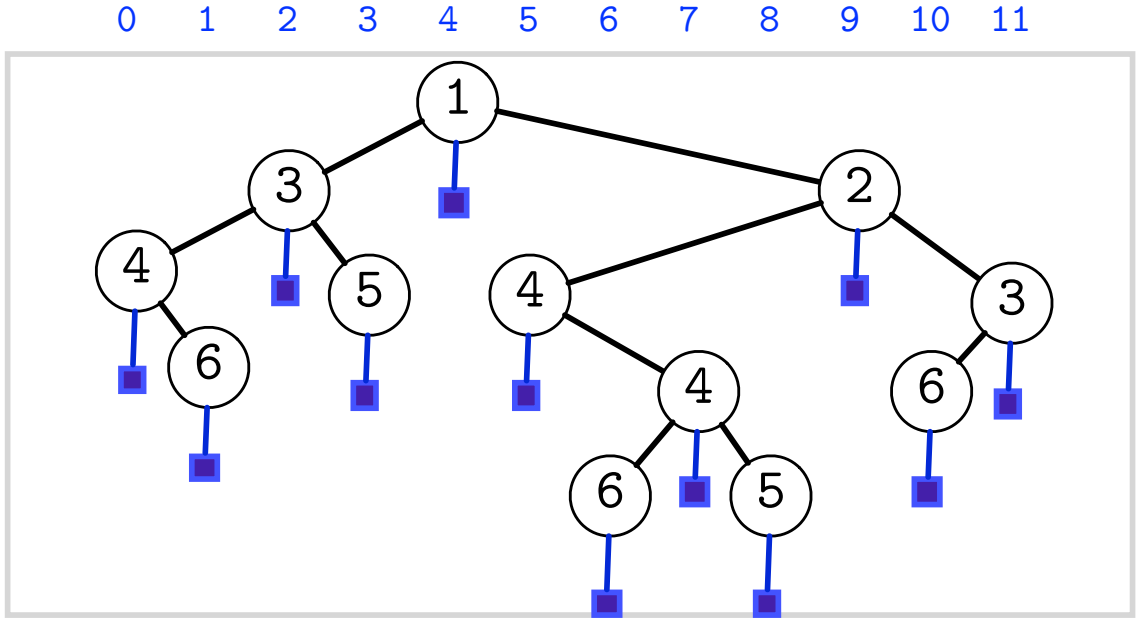


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,\overset{}{(}\,\overset{}{(}\,\overset{0}{(\,)}\,\overset{6}{(}\,\overset{1}{(\,)}\,\overset{}{)}\,\overset{6\ 4}{)}\,\overset{}{)}\,\overset{2}{(\,)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



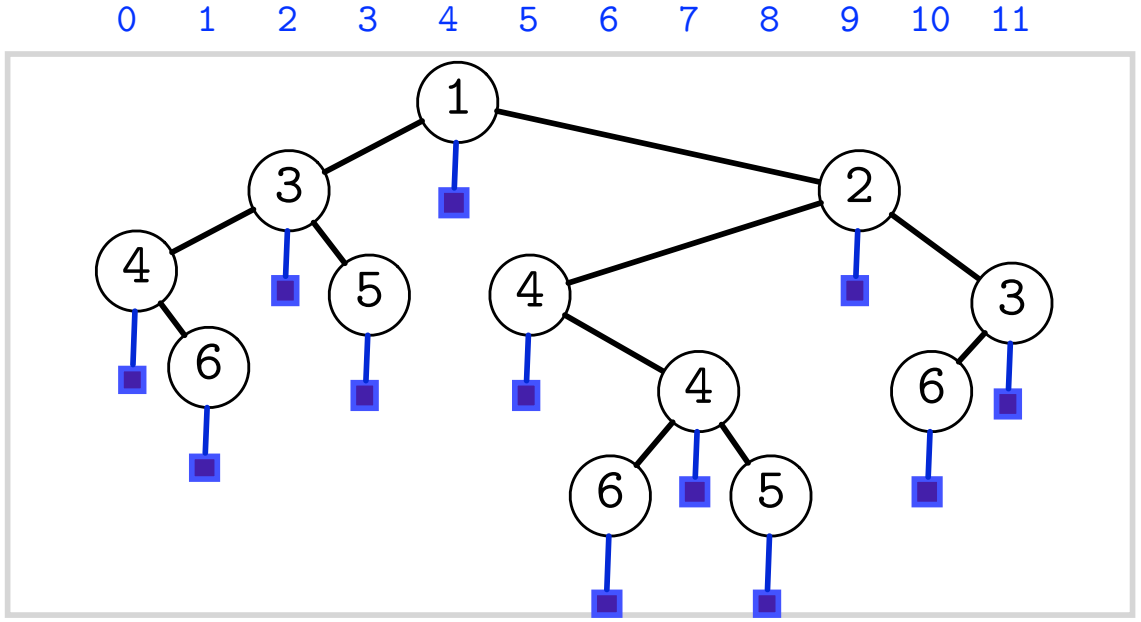
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,(\,(\,\overset{}{(}\,)\,\overset{6}{(}\,\overset{}{(}\,)\,)\,)\,\overset{6\ 4}{(}\,)\,\overset{5}{(}$$

0 1 2

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



(3) DFS traversal to construct balanced parentheses sequence

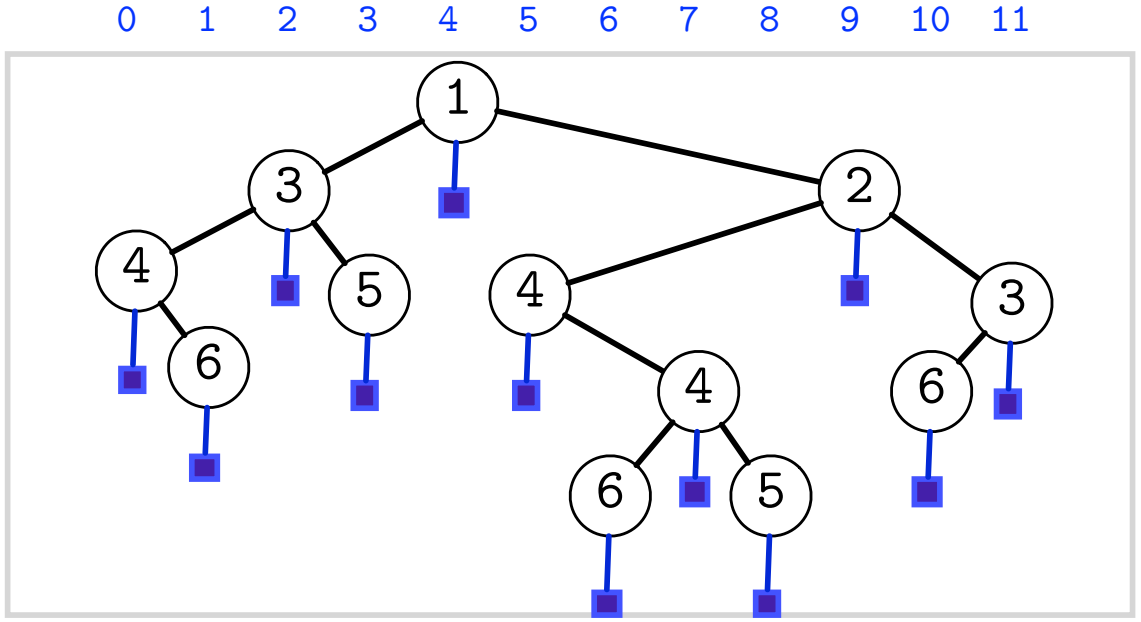
$$BP_{ext} = (\,(\,(\,(\,)\,(\,)\,)\,)\,(\,)\,(\,($$

1 3 4 6 6 4 5

0 1 2

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

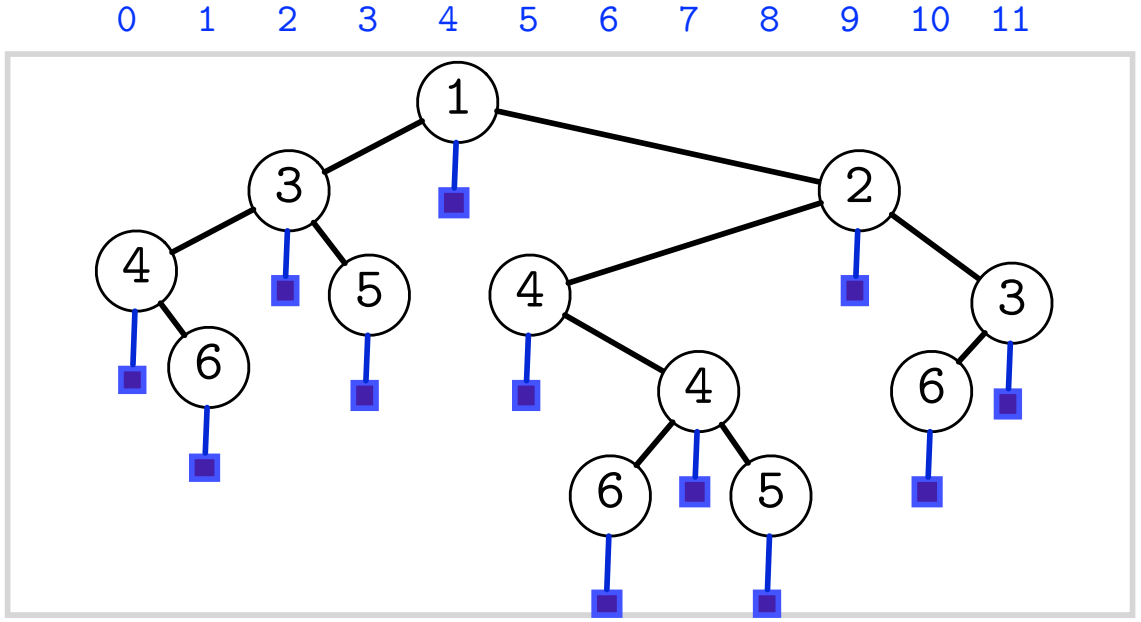


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,\overset{}{(}\,\overset{}{(}\,\overset{}{(}\,\overset{0}{)}\,\overset{}{(}\,\overset{1}{)}\,\overset{6}{)}\,\overset{6\ 4}{)}\,\overset{2}{(}\,\overset{}{)}\,\overset{5}{(}\,\overset{3}{(}\,\overset{}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

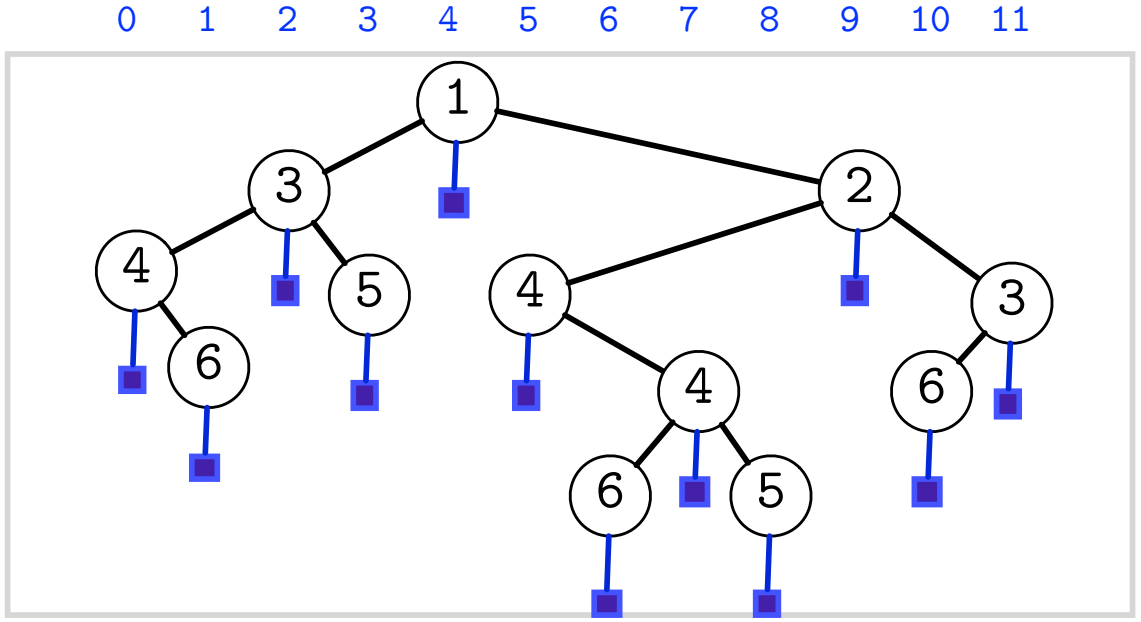


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{0}{()}\overset{6}{(}\overset{1}{()}\overset{6\ 4}{)}\overset{}{)}\overset{2}{()}\overset{5}{(}\overset{3}{()}\overset{5}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

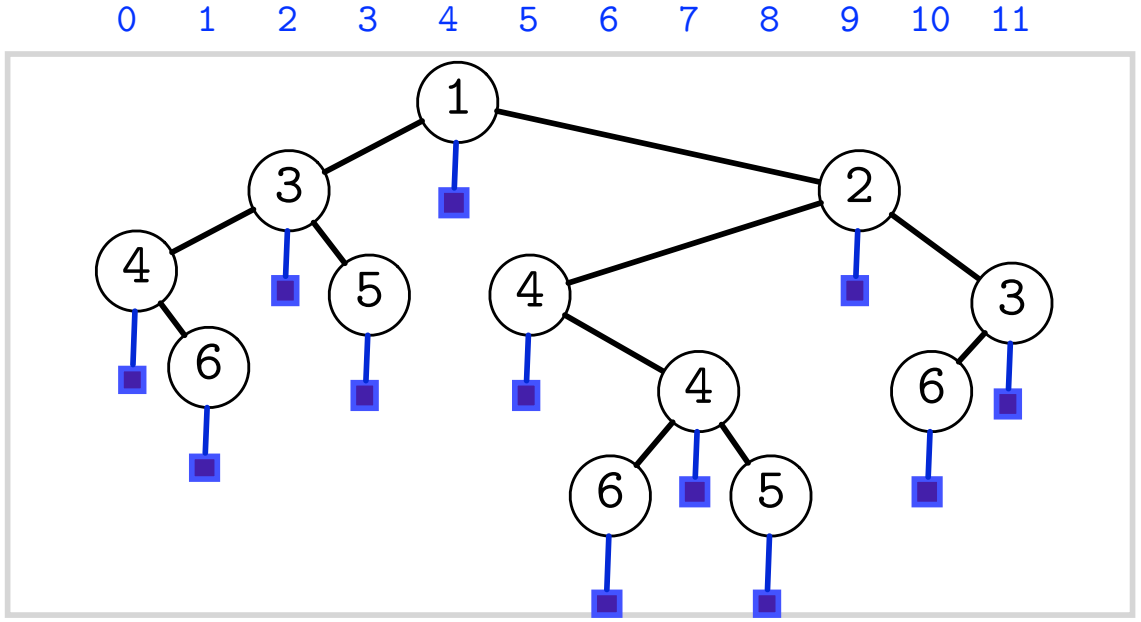


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{(}\overset{0}{)}\overset{}{(}\overset{6}{(}\overset{1}{)}\overset{}{)}\overset{6\ 4}{)}\overset{2}{(}\overset{}{)}\overset{5}{(}\overset{}{(}\overset{3}{)}\overset{}{)}\overset{5\ 3}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



(3) DFS traversal to construct balanced parentheses sequence

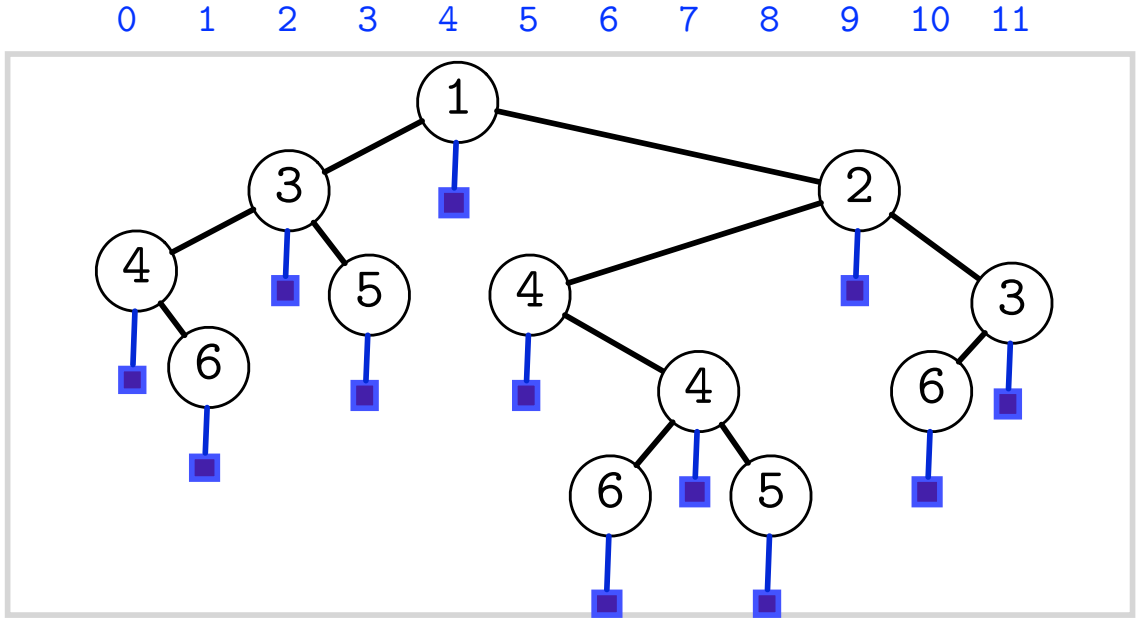
$$BP_{ext} = (((()(()))()(()))($$

1 3 4 6 6 4 5 5 3

0 1 2 3

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

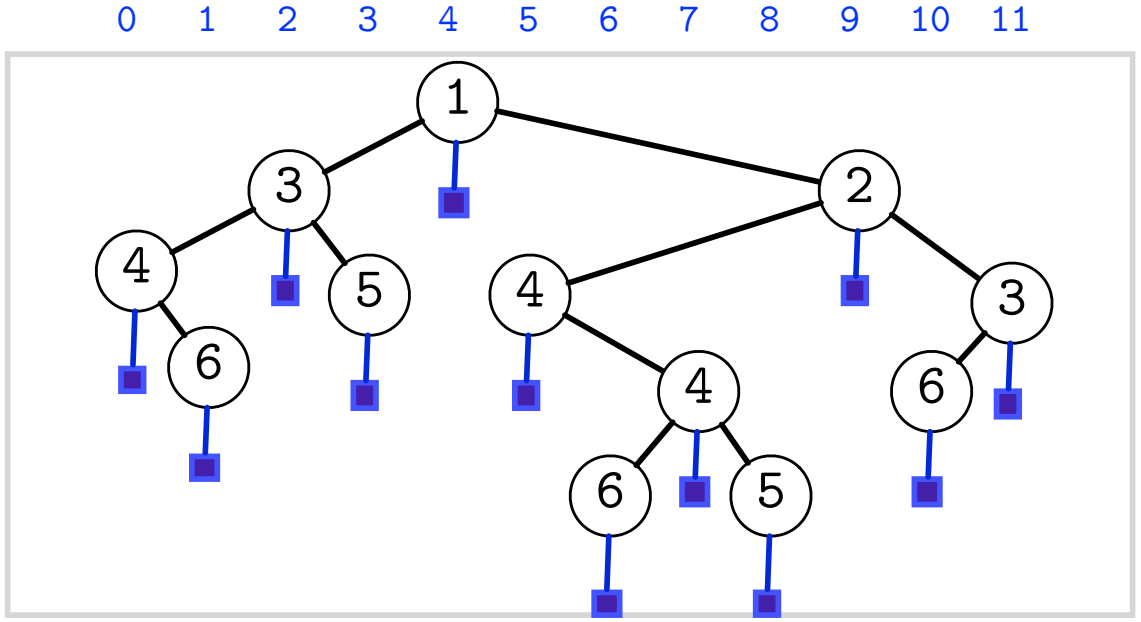


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\,\overset{}{(}\,\overset{}{(}\,\overset{0}{(\,)}\,\overset{6}{(}\,\overset{1}{(\,)}\,\overset{6\ 4}{)}\,\overset{2}{(\,)}\,\overset{5}{(}\,\overset{3}{(\,)}\,\overset{5\ 3}{)}\,\overset{4}{(\,)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



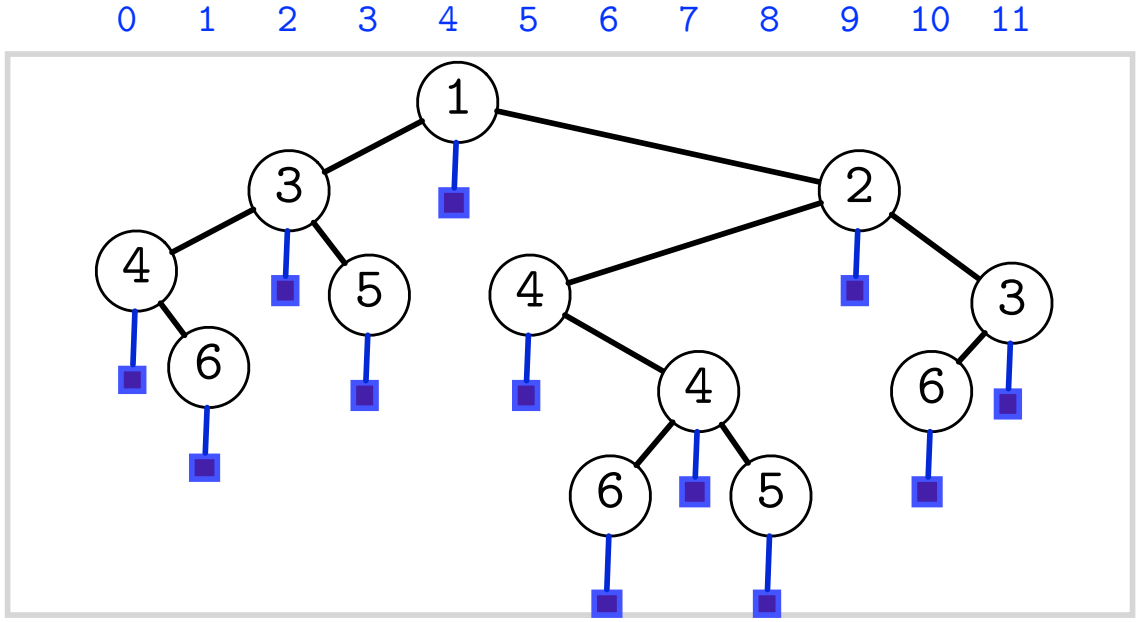
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1}{(}\overset{3}{(}\overset{4}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{(}\overset{}{)}\overset{6}{)}\overset{4}{)}\overset{}{(}\overset{}{)}\overset{5}{(}\overset{}{(}\overset{}{)}\overset{5}{)}\overset{3}{)}\overset{}{(}\overset{}{)}\overset{2}{(}$$

$$\qquad\qquad\quad 0 \quad 1 \quad\quad 2 \quad 3 \quad\quad 4$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



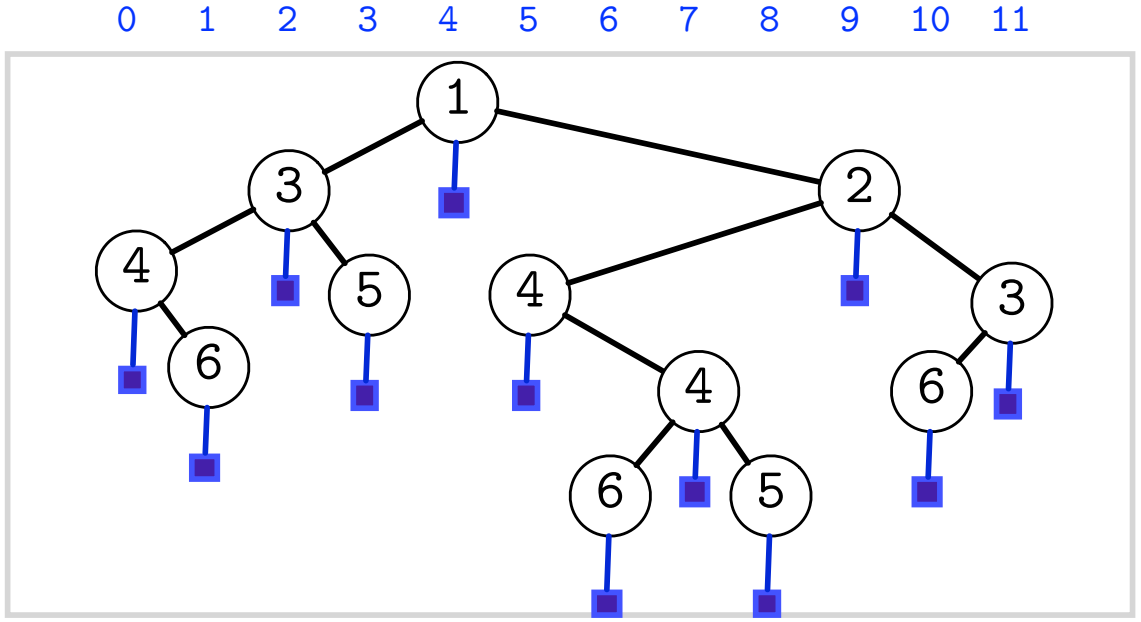
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{}{)}\overset{5}{(}\overset{}{(}\overset{5\ 3}{)}\overset{}{)}\overset{2\ 4}{(}\overset{}{)}\overset{}{(}\overset{}{(}$$

$$\phantom{BP_{ext} = (((}\ 0\ \ 1\ \ \ 2\ \ 3\ \ \ 4$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



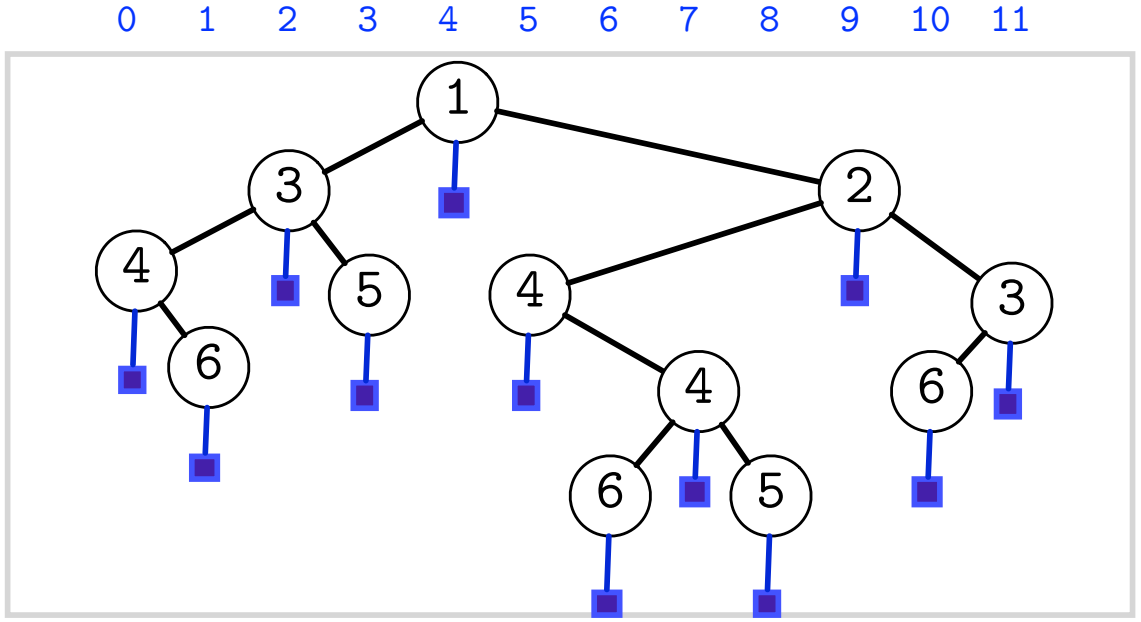
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{}{)}\overset{5}{(}\overset{}{(}\overset{5\ 3}{)}\overset{}{)}\overset{}{)}\overset{2\ 4}{(}\overset{}{(}\overset{}{(}$$

$$0 \quad 1 \quad 2 \quad 3 \quad 4$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

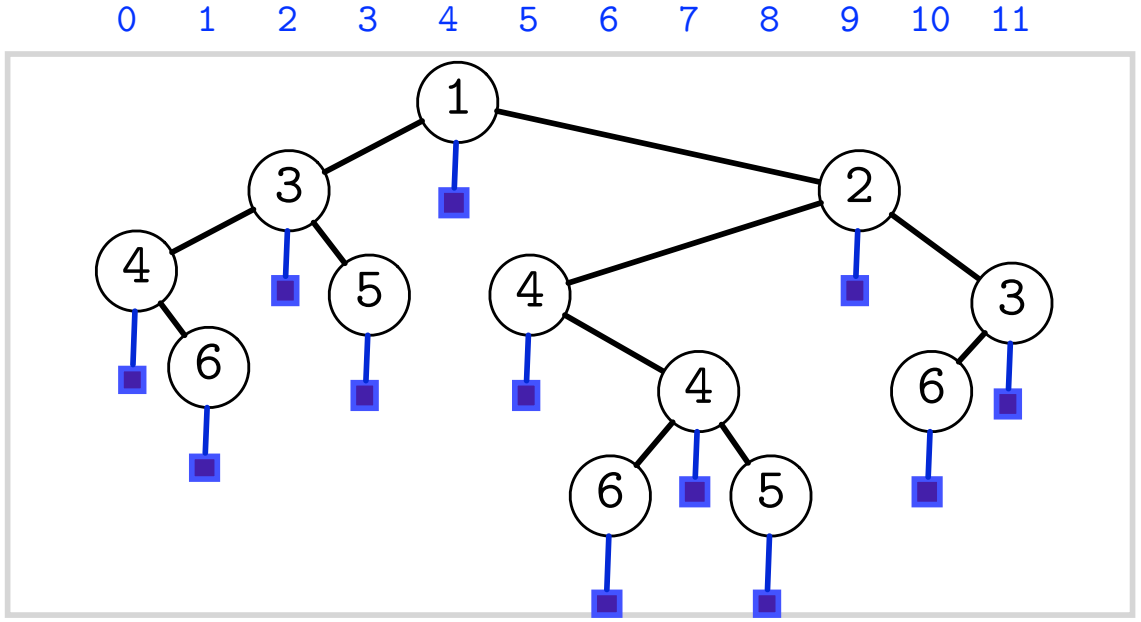


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{0}{()}\overset{6}{(}\overset{1}{()}\overset{6\ 4}{)}\overset{2}{()}\overset{5}{(}\overset{3}{()}\overset{5\ 3}{)}\overset{4}{()}\overset{2\ 4}{(}\overset{}{(}\overset{5}{()}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



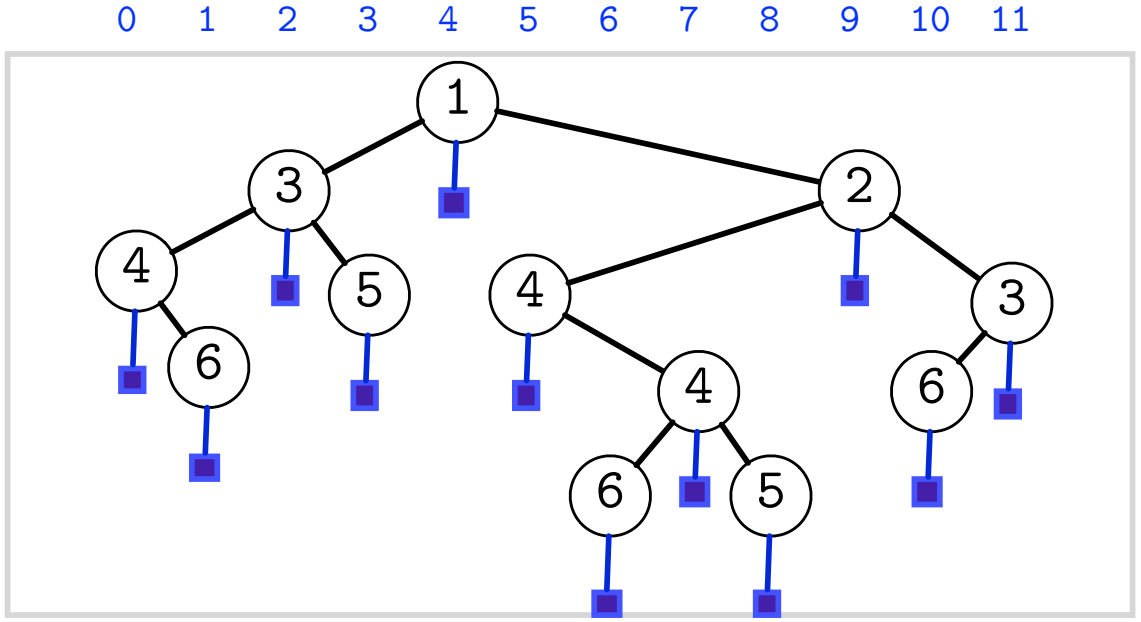
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{}{)}\overset{5}{(}\overset{}{(}\overset{5\ 3}{)}\overset{}{)}\overset{}{)}\overset{2\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{4}{(}$$

$$\qquad\qquad\quad 0 \quad 1 \quad\quad 2 \quad 3 \quad\quad 4 \quad\quad 5$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



(3) DFS traversal to construct balanced parentheses sequence

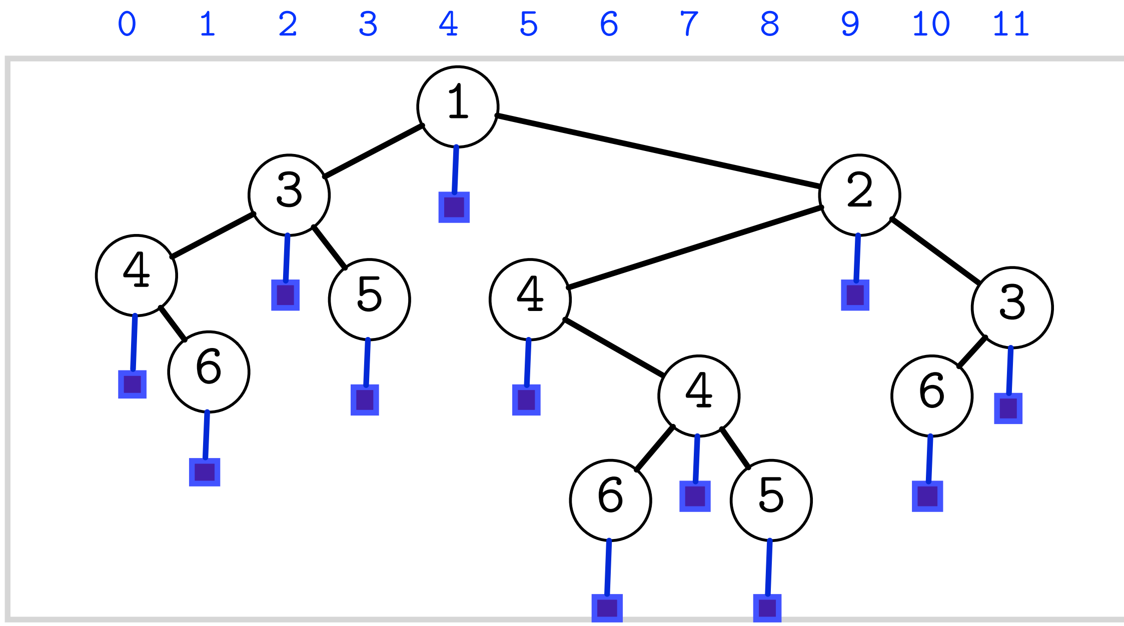
$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{(}\overset{6}{)}\overset{}{(}\overset{}{(}\overset{6\ 4}{)}\overset{}{)}\overset{}{(}\overset{5}{)}\overset{}{(}\overset{}{(}\overset{5\ 3}{)}\overset{}{)}\overset{}{)}\overset{2\ 4}{(}\overset{}{(}\overset{}{(}\overset{4\ 6}{)}\overset{}{(}\overset{}{(}$$

$$\qquad\qquad\qquad\quad 0 \quad 1 \quad\quad 2 \quad 3 \quad\quad 4 \quad\quad 5$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

Karlsruhe Institute of Technology

(2) Add a leaf to each node



(3) DFS traversal to construct balanced parentheses sequence

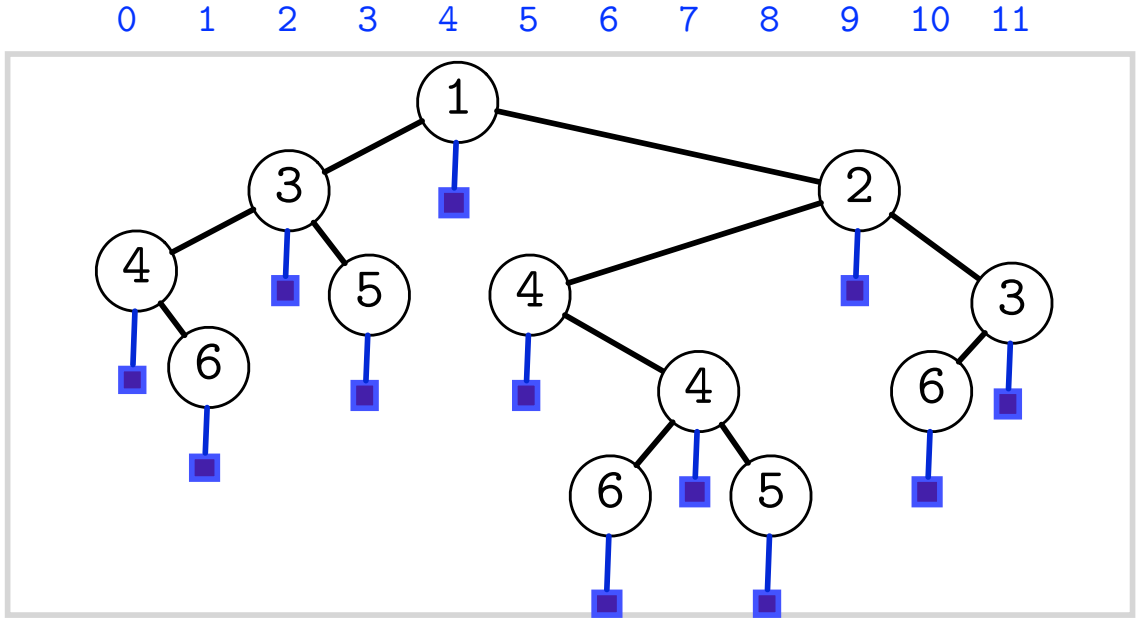
1 3 4 6 6 4 5 5 3 2 4 4 6

$BP_{ext} =$ ((()(()))()(()))()((()(((

0 1 2 3 4 5

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



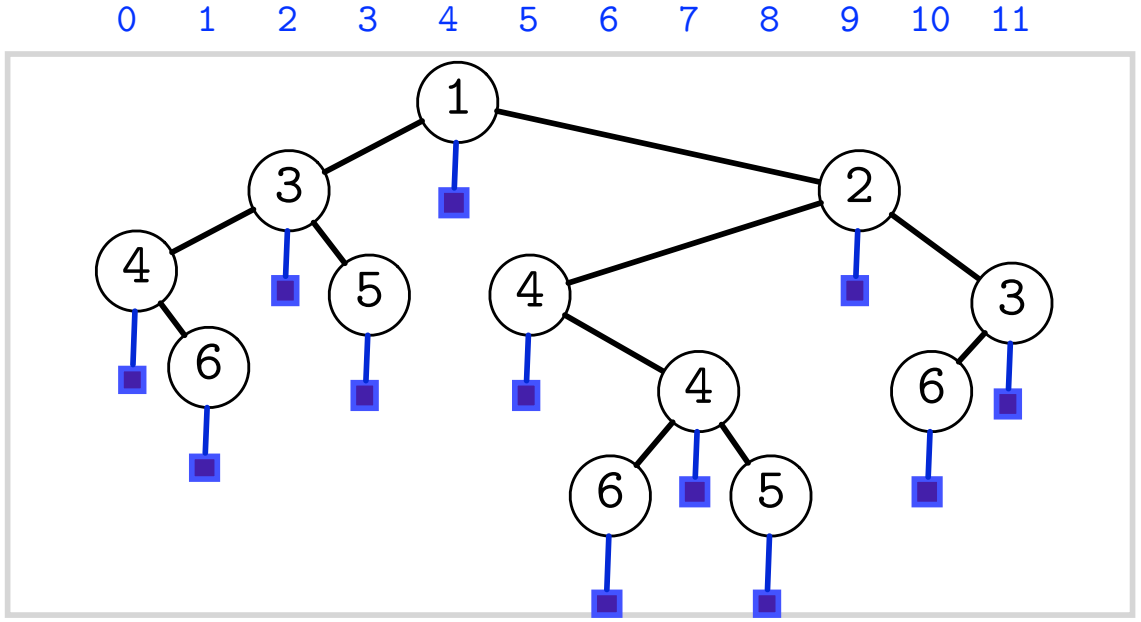
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{}{)}\overset{}{(}\overset{5}{(}\overset{}{)}\overset{5\ 3}{)}\overset{}{)}\overset{}{(}\overset{2\ 4}{(}\overset{}{(}\overset{}{)}\overset{4\ 6}{(}\overset{}{(}\overset{}{)}$$

$$\qquad\qquad\quad 0 \quad 1 \quad\quad 2 \quad 3 \quad\quad 4 \quad\quad 5 \quad\quad 6$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



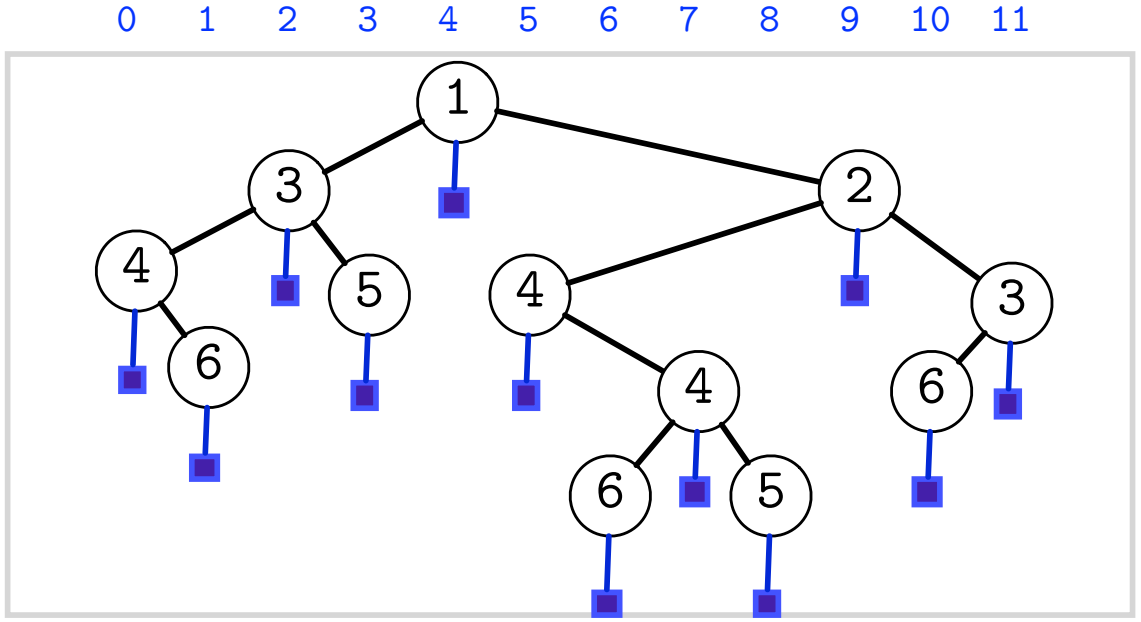
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1}{(}\overset{3}{(}\overset{4}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{(}\overset{}{)}\overset{6}{)}\overset{4}{)}\overset{}{(}\overset{5}{)}\overset{}{(}\overset{}{(}\overset{}{)}\overset{5}{)}\overset{3}{)}\overset{}{(}\overset{2}{)}\overset{4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{4}{(}\overset{6}{(}\overset{}{(}\overset{}{)}\overset{6}{)}$$

$$\qquad\qquad\qquad\qquad\qquad\quad 0 \quad 1 \quad\quad 2 \quad 3 \quad\quad 4 \quad\quad 5 \quad\quad 6$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



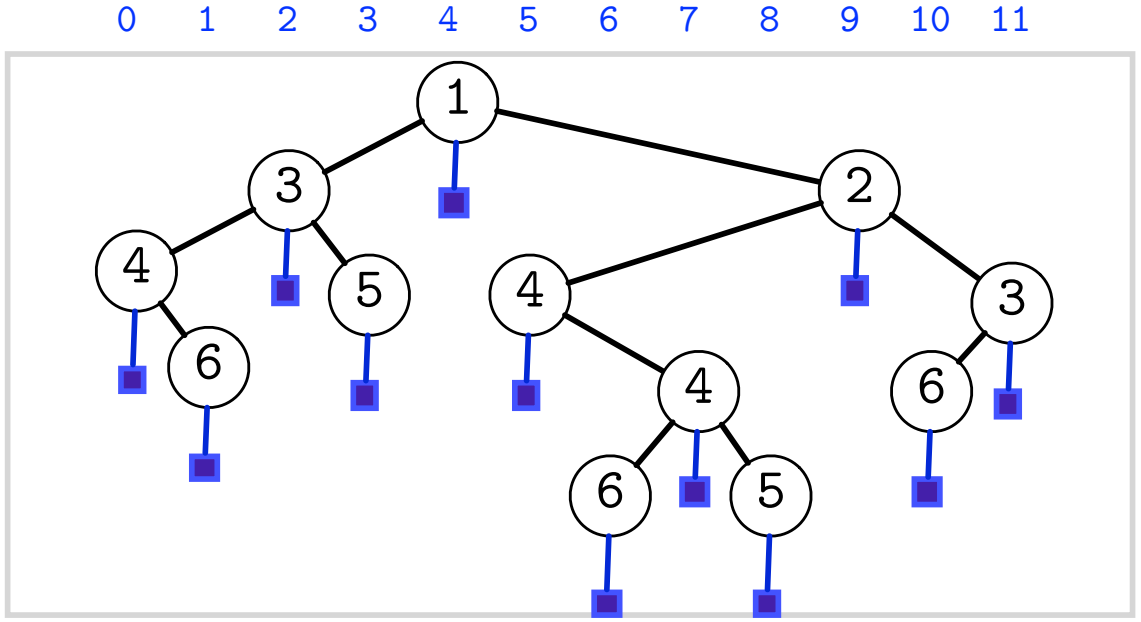
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{5}{)}\overset{}{(}\overset{}{(}\overset{5\ 3}{)}\overset{}{)}\overset{2\ 4}{(}\overset{}{(}\overset{4\ 6}{)}\overset{6}{(}\overset{}{(}\overset{}{)}\overset{}{)}\overset{}{(}$$

$$\phantom{BP_{ext} = (}\ 0\ \ 1\ \ \ 2\ \ 3\ \ \ 4\ \ \ 5\ \ \ 6$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



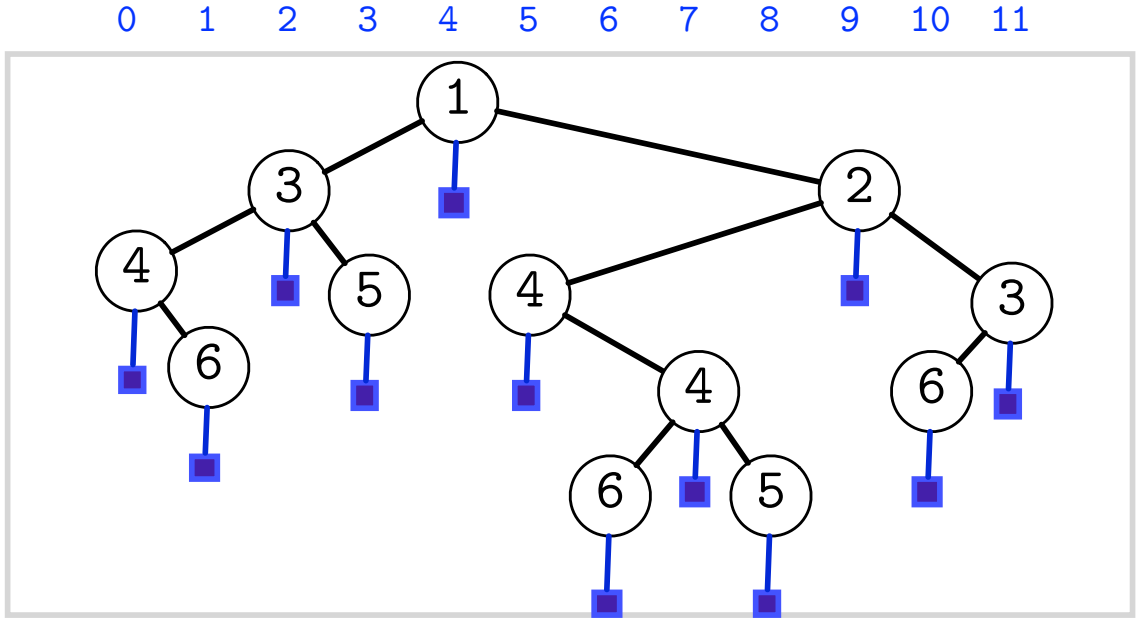
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{}{)}\overset{}{(}\overset{5}{(}\overset{}{)}\overset{5\ 3}{)}\overset{}{)}\overset{2\ 4}{(}\overset{}{(}\overset{}{)}\overset{4\ 6}{(}\overset{}{(}\overset{}{)}\overset{6}{)}\overset{}{)}\overset{}{(}\overset{}{)}$$

$$\qquad\qquad\quad 0\quad 1\quad\ 2\quad 3\quad\ 4\quad\ 5\quad\ 6\quad 7$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



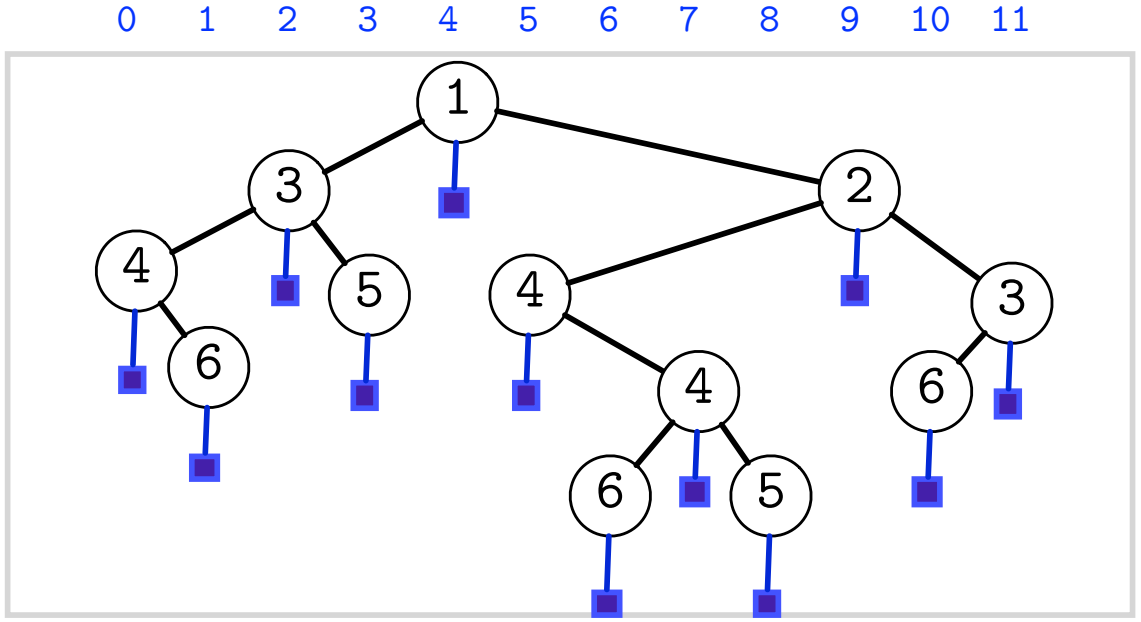
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1\ 3\ 4}{(}\overset{}{(}\overset{}{(}\overset{}{)}\overset{6}{(}\overset{}{)}\overset{6\ 4}{)}\overset{5}{)}\overset{}{(}\overset{5\ 3}{(}\overset{}{)}\overset{}{)}\overset{2\ 4}{)}\overset{}{(}\overset{}{(}\overset{}{)}\overset{4\ 6}{(}\overset{}{(}\overset{}{)}\overset{6}{)}\overset{}{(}\overset{5}{)}\overset{}{(}$$

$$\qquad\qquad\qquad\quad 0\quad 1\qquad 2\quad 3\qquad 4\qquad 5\qquad 6\quad 7$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

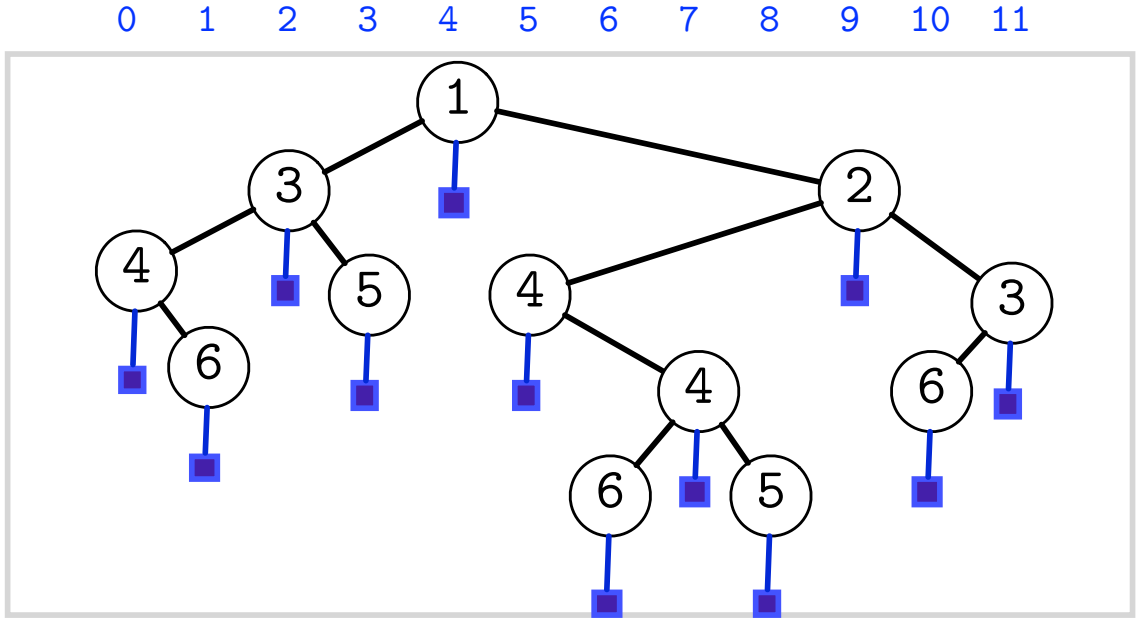


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(} \underset{\substack{1 \\ 1}}{(} \underset{\substack{6 \\ 2}}{(} \underset{\substack{6\ 4 \\ 3}}{(} \underset{\substack{5 \\ 4}}{)} \underset{\substack{5\ 3 \\ 5}}{(} \underset{\substack{2\ 4 \\ 6}}{)} \underset{\substack{4\ 6 \\ 7}}{)} \underset{\substack{6 \\ 8}}{)} \underset{\substack{5 \\ 9}}{)} \underset{\substack{ \\ 10}}{)} \underset{\substack{ \\ 11}}{(} \underset{\substack{ \\ 12}}{(} \underset{\substack{ \\ 13}}{(} \underset{\substack{ \\ 14}}{)} \underset{\substack{ \\ 15}}{(} \underset{\substack{ \\ 16}}{(} \underset{\substack{ \\ 17}}{)} \underset{\substack{ \\ 18}}{)} \underset{\substack{ \\ 19}}{(} \underset{\substack{ \\ 20}}{(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

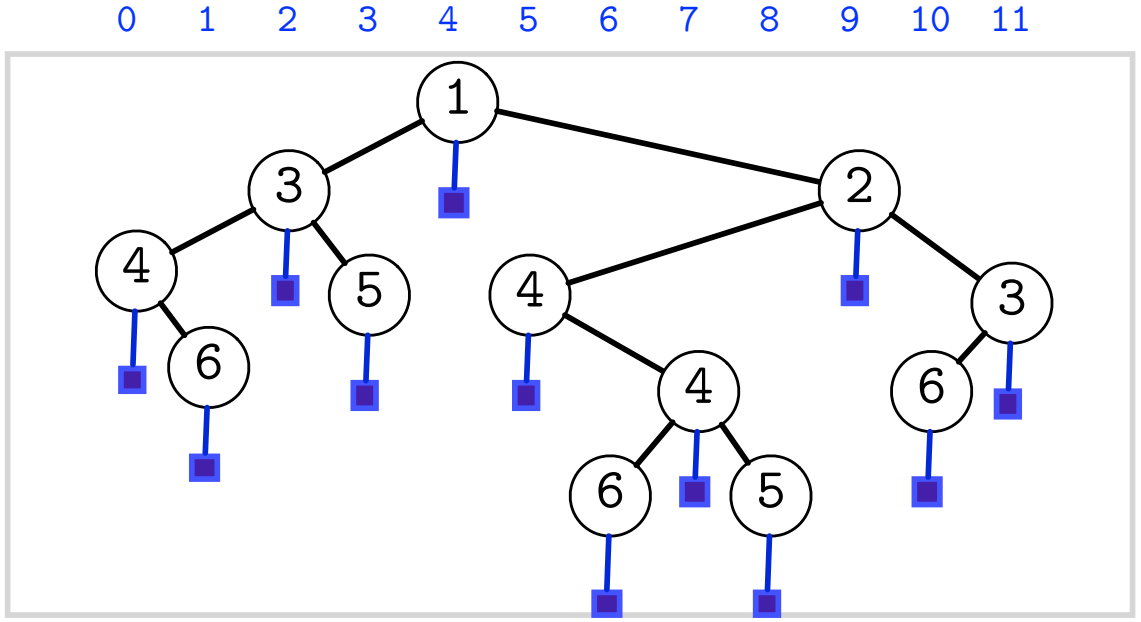


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \overset{1}{(}\overset{3}{(}\overset{4}{(}\underset{0}{()}\overset{6}{(}\underset{1}{()}\overset{6}{)}\overset{4}{)}\underset{2}{()}\overset{5}{(}\underset{3}{()}\overset{5}{)}\overset{3}{)}\underset{4}{()}\overset{2}{(}\overset{4}{(}\underset{5}{()}\overset{4}{(}\overset{6}{(}\underset{6}{()}\overset{6}{)}\underset{7}{()}\overset{5}{)}\overset{4}{(}\underset{8}{()}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



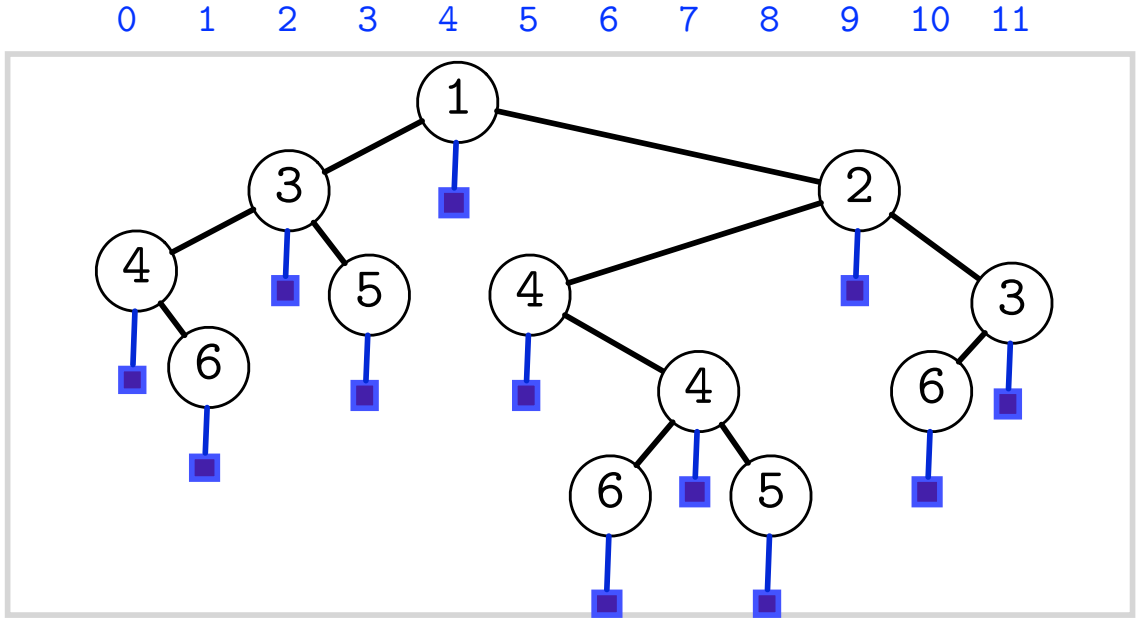
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{1\ 3\ 4}{(}\underset{6}{(}\underset{6\ 4}{(}\underset{5}{(}\underset{5\ 3}{)}\underset{2\ 4}{)}\underset{4\ 6}{(}\underset{6}{(}\underset{5}{(}\underset{5}{)}$$

$$\underset{0}{(}\underset{1}{(}\underset{2}{)}\underset{3}{(}\underset{4}{)}\underset{5}{(}\underset{6}{(}\underset{7}{)}\underset{8}{(}\underset{}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



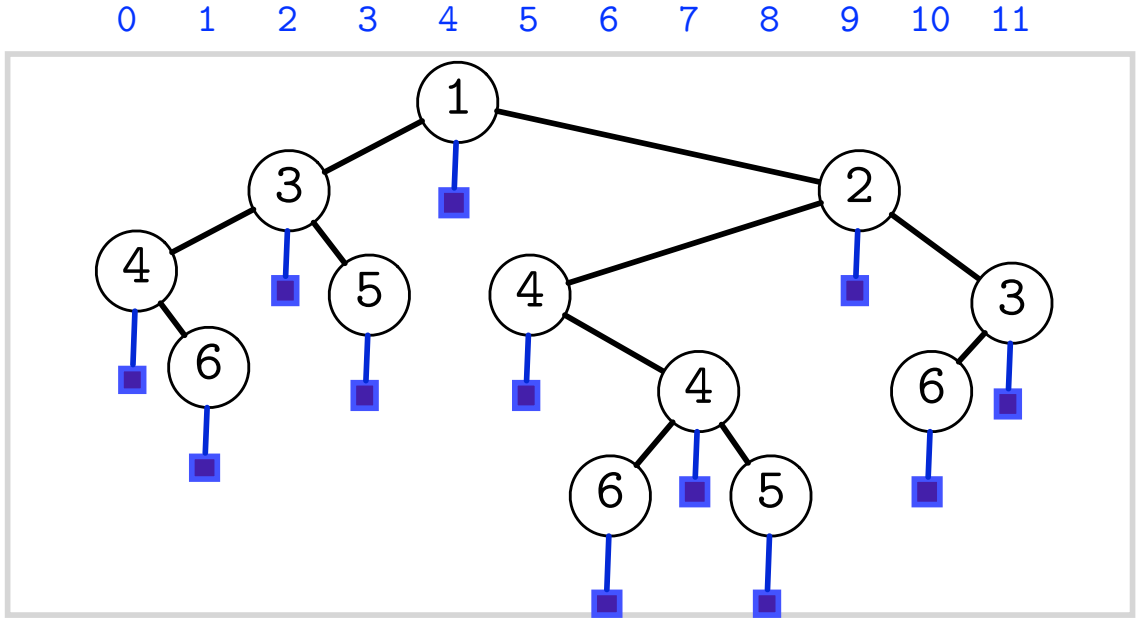
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{1\ 3\ 4}{(}\underset{6}{(}\underset{6\ 4}{(}\underset{5}{(}\underset{5\ 3}{)}\underset{2\ 4}{)}\underset{4\ 6}{(}\underset{6}{(}\underset{5}{(}\underset{5\ 4}{)}\underset{}{)}\underset{}{)}\underset{}{)}\underset{}{)}\underset{}{)}\underset{}{)}\underset{}{)}\underset{}{)}$$

$$\underset{0}{(}\underset{1}{(}\underset{2}{)}\underset{3}{)}\underset{4}{(}\underset{5}{(}\underset{6}{)}\underset{7}{)}\underset{8}{(}\underset{}{)}\underset{}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



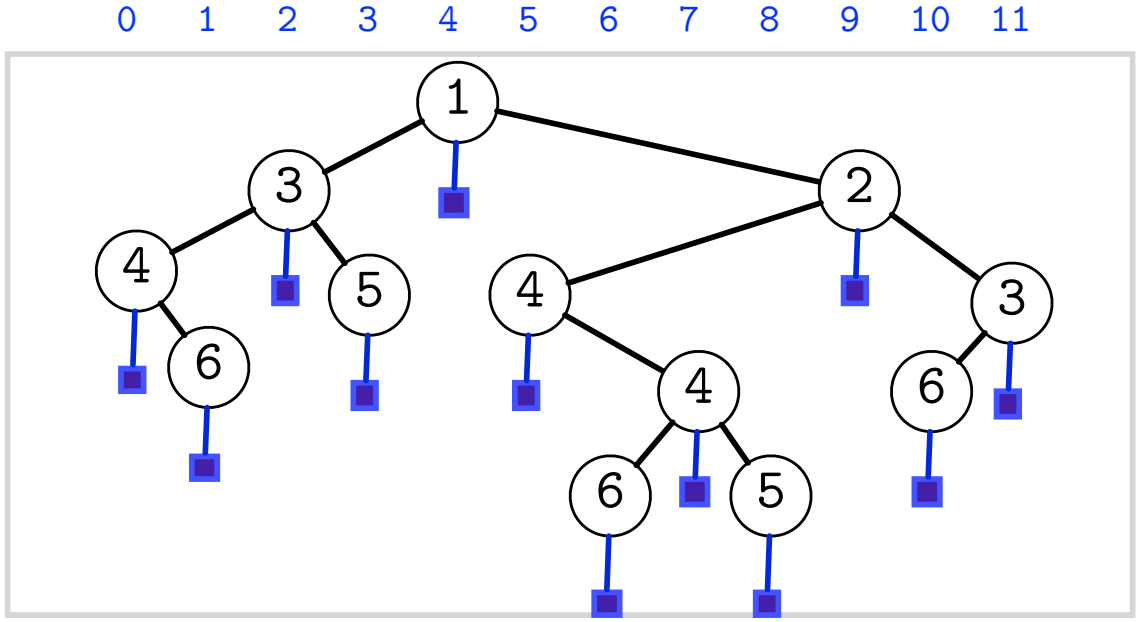
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{1\ 3\ 4}{(}\underset{6}{(}\underset{6\ 4}{(}\underset{5}{(}\underset{5\ 3}{)}\underset{2\ 4}{)}\underset{4\ 6}{(}\underset{6}{(}\underset{5}{)}\underset{5\ 4\ 4}{)}))$$

$$\underset{0}{(}\underset{1}{(}\underset{2}{)}\underset{3}{(}\underset{4}{)}\underset{5}{(}\underset{6}{(}\underset{7}{)}\underset{8}{(}))$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

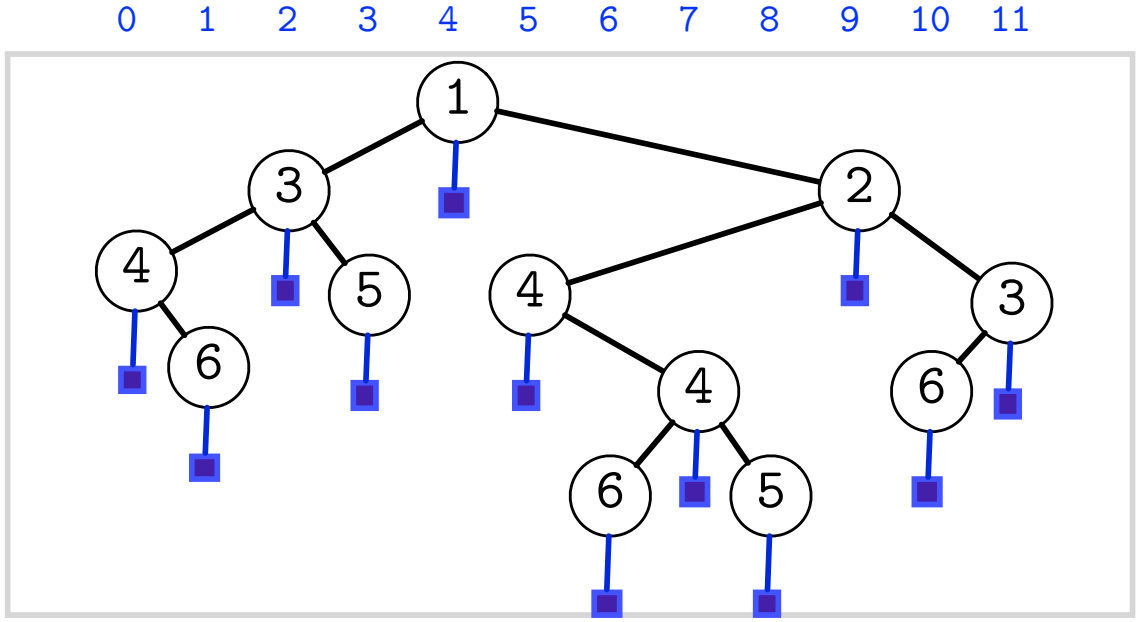


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = (((()(()))()(()))()((()((()()(()))(
\quad 0\ 1\ \ 2\ 3\ \ 4\ \ 5\ \ 6\ 7\ 8$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



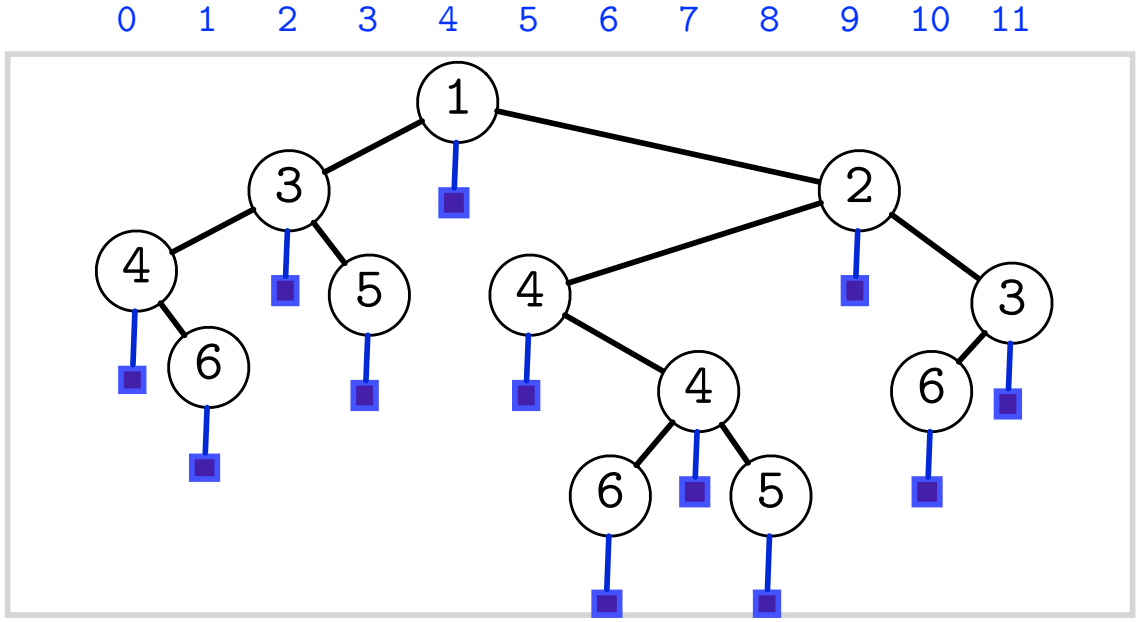
(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{1\ 3\ 4}{(}\underset{6}{(}\underset{6\ 4}{(}\underset{5}{(}\underset{5\ 3}{)}\underset{2\ 4}{)}\underset{4\ 6}{(}\underset{6}{(}\underset{5}{)}\underset{5\ 4\ 4}{)}\underset{}{)}$$

$$\underset{0}{(}\underset{1}{)}\underset{2}{(}\underset{3}{)}\underset{4}{(}\underset{5}{(}\underset{6}{)}\underset{7}{(}\underset{8}{)}\underset{9}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

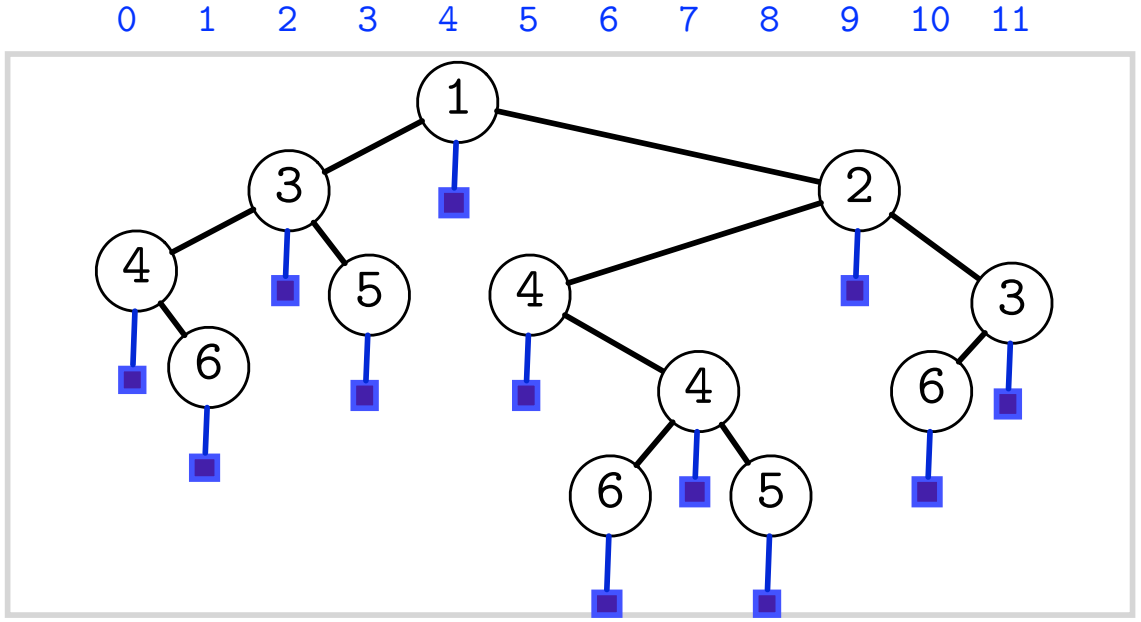


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1 \\ 1}}{(}\underset{\substack{6 \\ 2}}{(}\underset{\substack{ \\ 3}}{)}\underset{\substack{6\ 4 \\ 4}}{(}\underset{\substack{ \\ 5}}{)}\underset{\substack{ \\ 6}}{)}\underset{\substack{5 \\ 7}}{)}\underset{\substack{5\ 3 \\ 8}}{(}\underset{\substack{ \\ 9}}{)}\underset{\substack{2\ 4 \\ 10}}{)}\underset{\substack{4\ 6 \\ 11}}{(}\underset{\substack{ \\ 12}}{(}\underset{\substack{6 \\ 13}}{)}\underset{\substack{ \\ 14}}{(}\underset{\substack{5 \\ 15}}{(}\underset{\substack{ \\ 16}}{)}\underset{\substack{5\ 4\ 4 \\ 17}}{)}\underset{\substack{3 \\ 18}}{)}\underset{\substack{ \\ 19}}{)}\underset{\substack{ \\ 20}}{(}\underset{\substack{ \\ 21}}{)}\underset{\substack{ \\ 22}}{(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

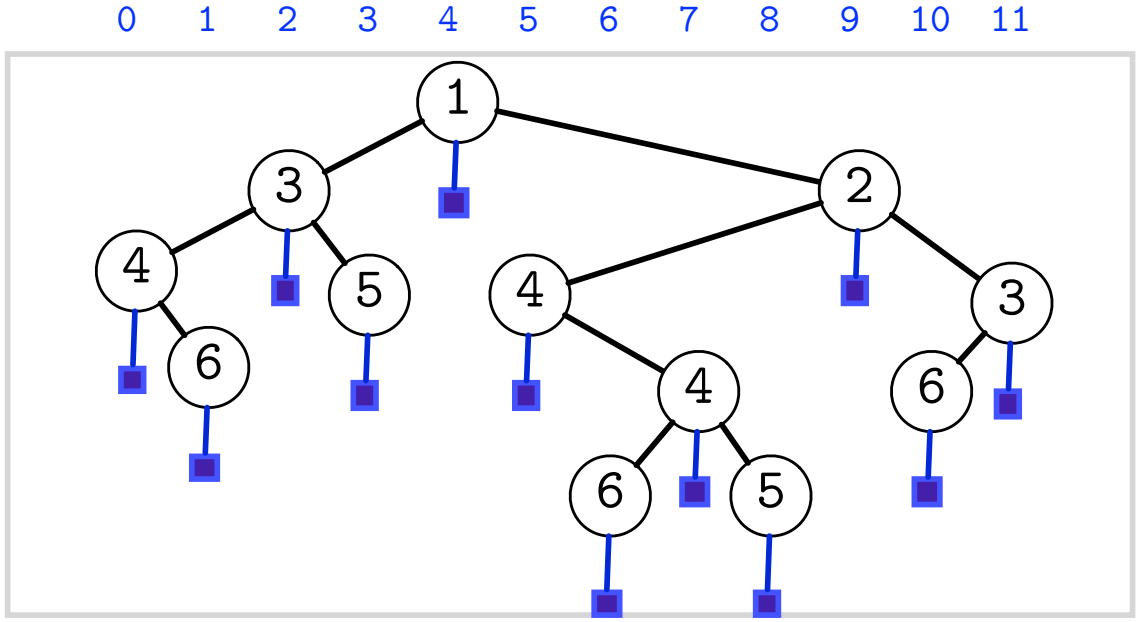


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1 \\ 1}}{(}\underset{\substack{6 \\ 2}}{(}\underset{\substack{6\ 4 \\ 3}}{(}\underset{\substack{5 \\ 4}}{)}\underset{\substack{5\ 3 \\ 5}}{)}\underset{\substack{2\ 4 \\ 6}}{)}\underset{\substack{4\ 6 \\ 7}}{(}\underset{\substack{6 \\ 8}}{(}\underset{\substack{5 \\ 9}}{)}\underset{\substack{5\ 4\ 4 \\ 10}}{)}\underset{\substack{3\ 6 \\ 11}}{)}\underset{\substack{ \\ 12}}{(}\underset{\substack{ \\ 13}}{(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

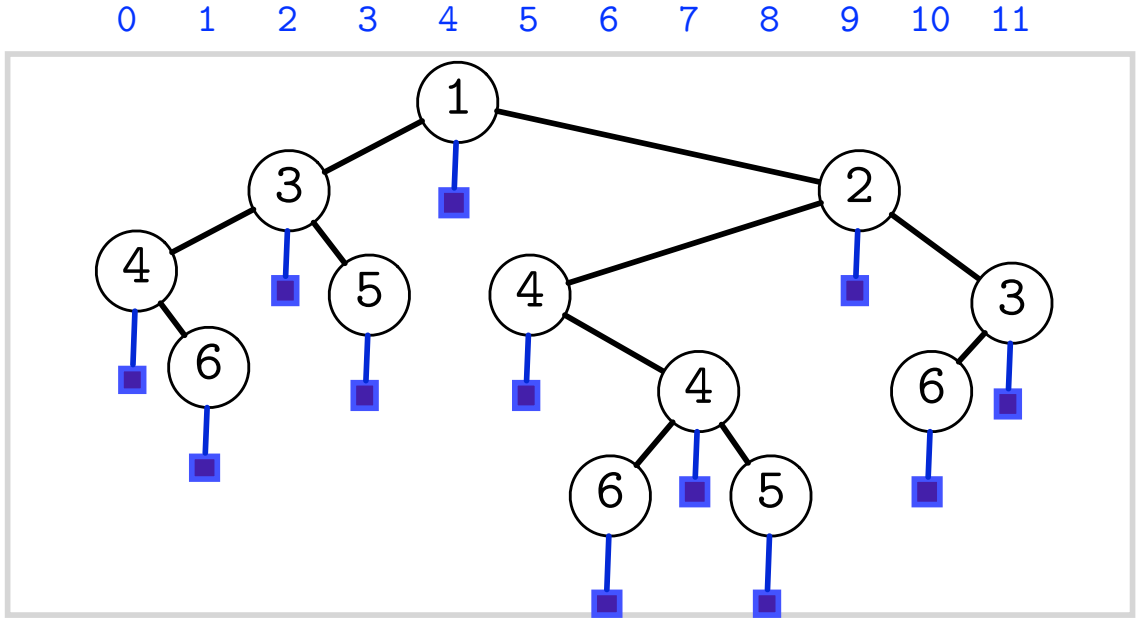


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{1\ 3\ 4}{(}\underset{6}{(}\underset{6\ 4}{(}\underset{5}{(}\underset{5\ 3}{)}\underset{2\ 4}{)}\underset{4\ 6}{(}\underset{6}{(}\underset{5}{)}\underset{5\ 4\ 4}{)}\underset{3\ 6}{)}(((\underset{0}{)}\underset{1}{(}\underset{2}{)}\underset{3}{(}\underset{4}{)}\underset{5}{)}\underset{6}{(}\underset{7}{(}\underset{8}{)}\underset{9}{(}))$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

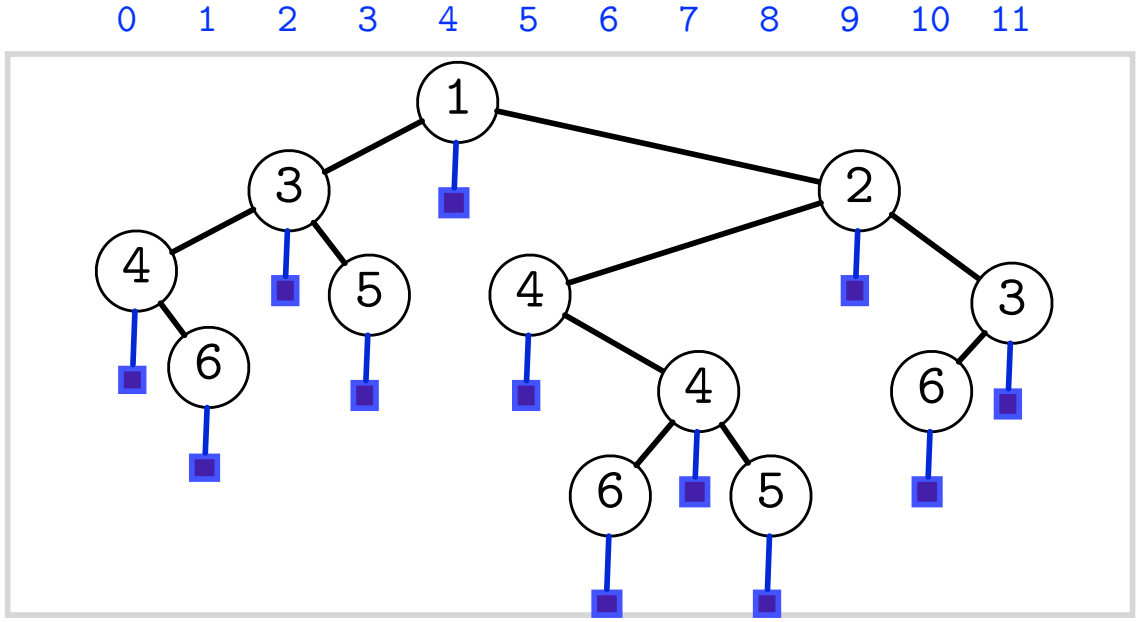


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1\ 3\ 4 \\ 1}}{(}\underset{\substack{1\ 3\ 4 \\ 2}}{(}\underset{\substack{6 \\ 3}}{(}\underset{\substack{6 \\ 4}}{)}\underset{\substack{1\ 3\ 4 \\ 5}}{)}\underset{\substack{6\ 4 \\ 6}}{(}\underset{\substack{6\ 4 \\ 7}}{)}\underset{\substack{1\ 3\ 4 \\ 8}}{)}\underset{\substack{5 \\ 9}}{(}\underset{\substack{5 \\ 10}}{)}\underset{\substack{5\ 3 \\ 11}}{)}\underset{\substack{2\ 4 \\ 12}}{(}\underset{\substack{2\ 4 \\ 13}}{(}\underset{\substack{4\ 6 \\ 14}}{(}\underset{\substack{4\ 6 \\ 15}}{)}\underset{\substack{6 \\ 16}}{(}\underset{\substack{6 \\ 17}}{(}\underset{\substack{6 \\ 18}}{)}\underset{\substack{5 \\ 19}}{(}\underset{\substack{5 \\ 20}}{)}\underset{\substack{5\ 4\ 4 \\ 21}}{)}\underset{\substack{5\ 4\ 4 \\ 22}}{)}\underset{\substack{5\ 4\ 4 \\ 23}}{)}\underset{\substack{3\ 6 \\ 24}}{(}\underset{\substack{3\ 6 \\ 25}}{(}\underset{\substack{3\ 6 \\ 26}}{(}\underset{\substack{3\ 6 \\ 27}}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

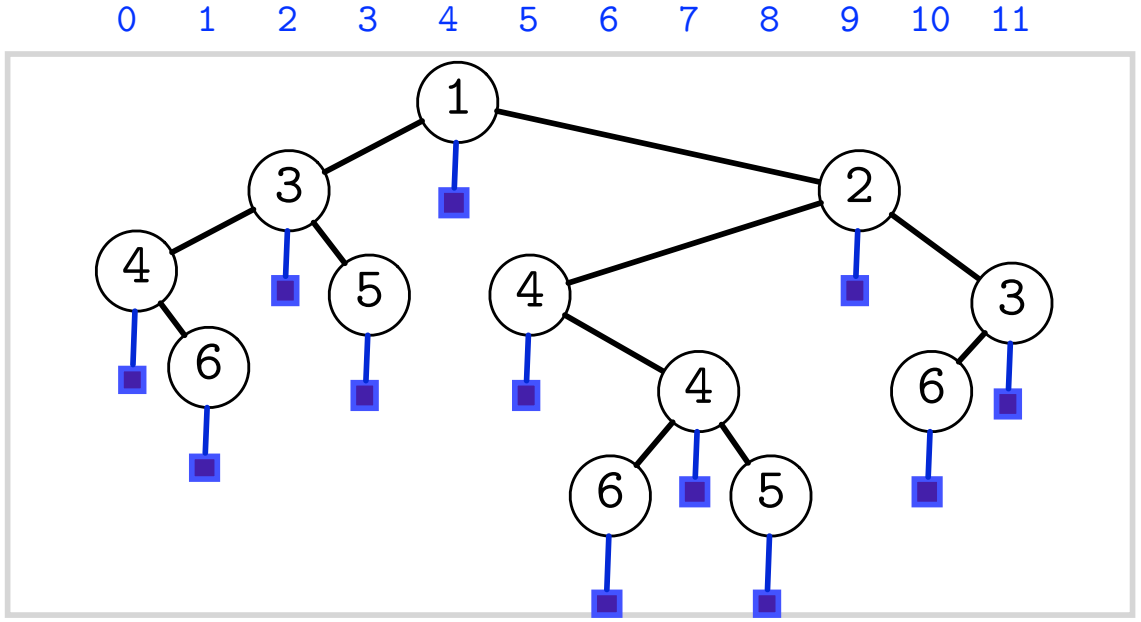


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1\ 3\ 4 \\ 1}}{(}\underset{\substack{1\ 3\ 4 \\ 2}}{(}\underset{\substack{6 \\ 3}}{(}\underset{\substack{6 \\ 4}}{)}\underset{\substack{6\ 4 \\ 5}}{(}\underset{\substack{6\ 4 \\ 6}}{)}\underset{\substack{1\ 3\ 4 \\ 7}}{)}\underset{\substack{5 \\ 8}}{(}\underset{\substack{5 \\ 9}}{)}\underset{\substack{5\ 3 \\ 10}}{(}\underset{\substack{5\ 3 \\ 11}}{(}\underset{\substack{2\ 4 \\ 12}}{)}\underset{\substack{2\ 4 \\ 13}}{(}\underset{\substack{4\ 6 \\ 14}}{(}\underset{\substack{6 \\ 15}}{(}\underset{\substack{6 \\ 16}}{)}\underset{\substack{4\ 6 \\ 17}}{(}\underset{\substack{6 \\ 18}}{)}\underset{\substack{5 \\ 19}}{(}\underset{\substack{5 \\ 20}}{)}\underset{\substack{5\ 4\ 4 \\ 21}}{)}\underset{\substack{5\ 4\ 4 \\ 22}}{)}\underset{\substack{3\ 6 \\ 23}}{)}\underset{\substack{3\ 6 \\ 24}}{(}\underset{\substack{6 \\ 25}}{(}\underset{\substack{6 \\ 26}}{)}\underset{\substack{6 \\ 27}}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

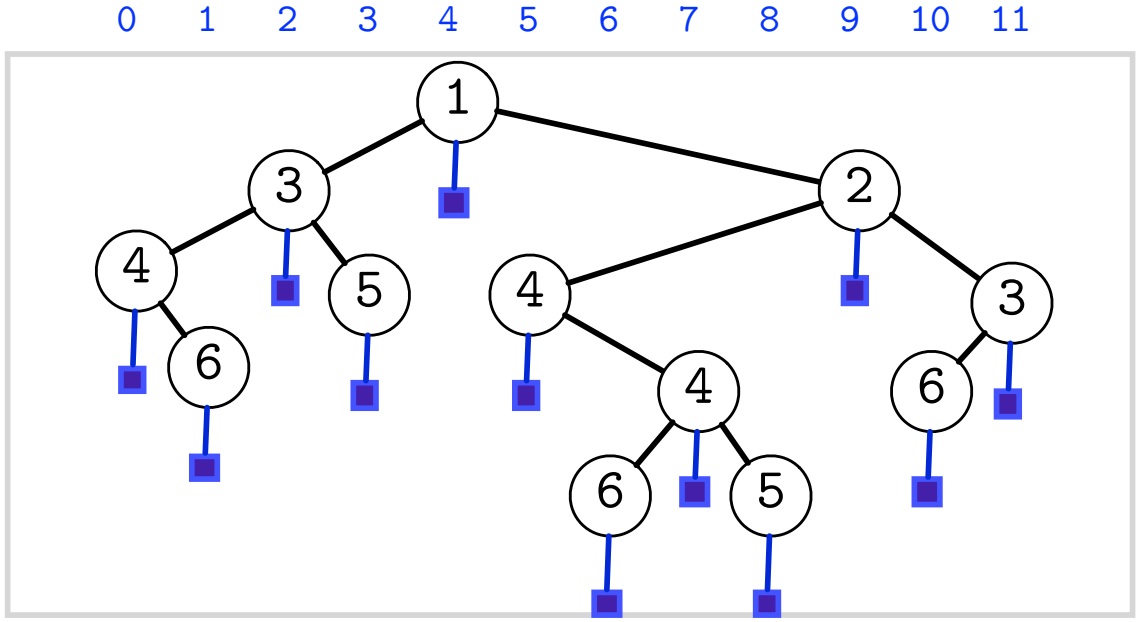


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1 \\ 1}}{(}\underset{\substack{6 \\ 2}}{(}\underset{\substack{ \\ 3}}{)}\underset{\substack{6\ 4 \\ 4}}{(}\underset{\substack{ \\ 5}}{)}\underset{\substack{ \\ 6}}{)}\underset{\substack{5 \\ 7}}{)}\underset{\substack{ \\ 8}}{(}\underset{\substack{5\ 3 \\ 9}}{(}\underset{\substack{ \\ 10}}{)}\underset{\substack{ \\ 11}}{)}\underset{\substack{2\ 4 \\ 12}}{(}\underset{\substack{ \\ 13}}{(}\underset{\substack{4\ 6 \\ 14}}{)}\underset{\substack{6 \\ 15}}{(}\underset{\substack{ \\ 16}}{(}\underset{\substack{ \\ 17}}{)}\underset{\substack{5 \\ 18}}{(}\underset{\substack{ \\ 19}}{)}\underset{\substack{5\ 4\ 4 \\ 20}}{(}\underset{\substack{ \\ 21}}{)}\underset{\substack{ \\ 22}}{)}\underset{\substack{3\ 6 \\ 23}}{)}\underset{\substack{ \\ 24}}{(}\underset{\substack{6 \\ 25}}{(}\underset{\substack{ \\ 26}}{)}\underset{\substack{ \\ 27}}{)}\underset{\substack{ \\ 28}}{(}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

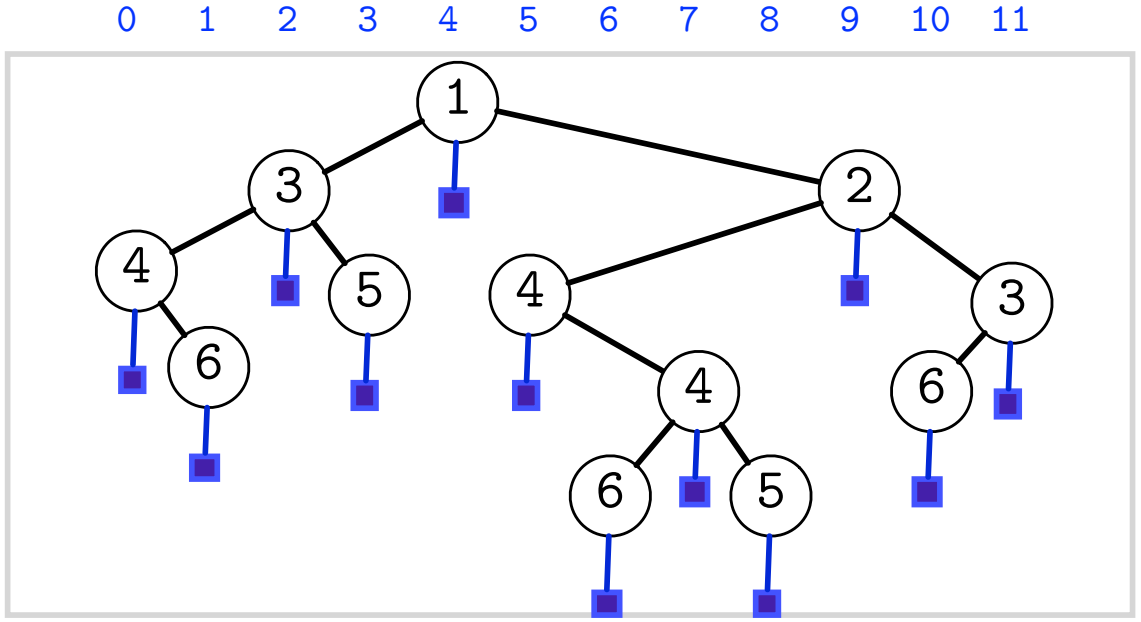


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(} \underset{\substack{1\ 3\ 4 \\ 1}}{(} \underset{\substack{1\ 3\ 4 \\ 2}}{(} \underset{\substack{6 \\ 3}}{(} \underset{\substack{6 \\ 4}}{)} \underset{\substack{6\ 4 \\ 5}}{(} \underset{\substack{6 \\ 6}}{)} \underset{\substack{6\ 4 \\ 7}}{)} \underset{\substack{5 \\ 8}}{)} \underset{\substack{5\ 3 \\ 9}}{(} \underset{\substack{5 \\ 10}}{)} \underset{\substack{5\ 3 \\ 11}}{)} \underset{\substack{2\ 4 \\ 12}}{(} \underset{\substack{2\ 4 \\ 13}}{(} \underset{\substack{4\ 6 \\ 14}}{(} \underset{\substack{6 \\ 15}}{(} \underset{\substack{6 \\ 16}}{)} \underset{\substack{4\ 6 \\ 17}}{(} \underset{\substack{6 \\ 18}}{)} \underset{\substack{4\ 6 \\ 19}}{)} \underset{\substack{5 \\ 20}}{)} \underset{\substack{5\ 4 \\ 21}}{(} \underset{\substack{5 \\ 22}}{)} \underset{\substack{5\ 4 \\ 23}}{)} \underset{\substack{4 \\ 24}}{)} \underset{\substack{4 \\ 25}}{)} \underset{\substack{3\ 6 \\ 26}}{(} \underset{\substack{3\ 6 \\ 27}}{(} \underset{\substack{6 \\ 28}}{(} \underset{\substack{6 \\ 29}}{)} \underset{\substack{6 \\ 30}}{)} \underset{\substack{6 \\ 31}}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

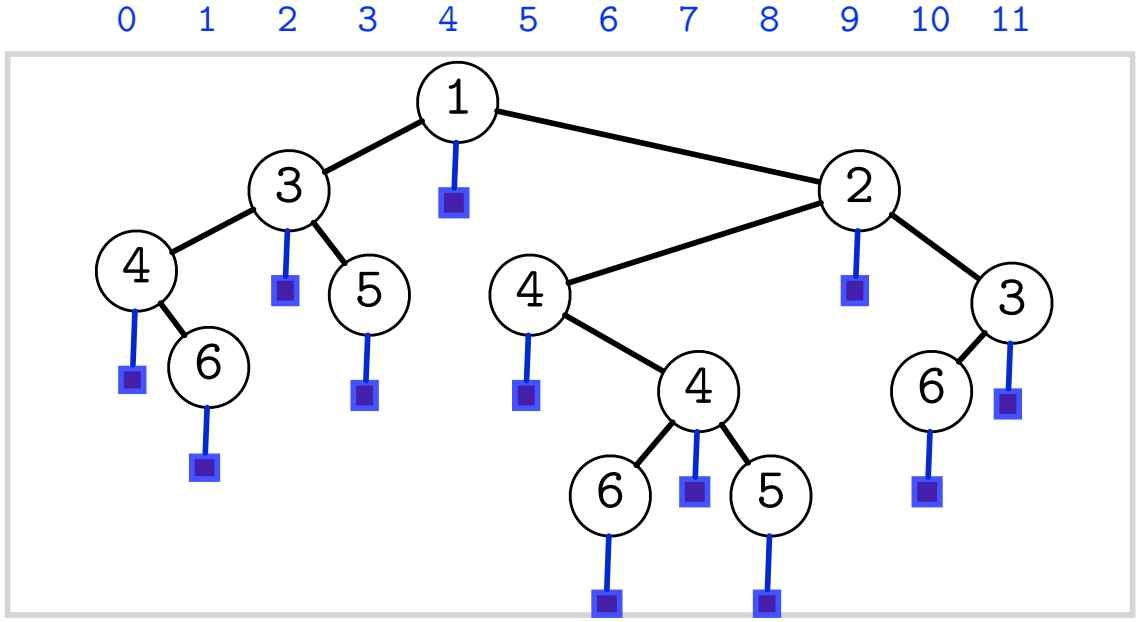


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1\ 3\ 4 \\ 1}}{(}\underset{\substack{1\ 3\ 4 \\ 2}}{(}\underset{\substack{6 \\ 3}}{(}\underset{\substack{6 \\ 4}}{)}\underset{\substack{1\ 3\ 4 \\ 5}}{)}\underset{\substack{6\ 4 \\ 6}}{(}\underset{\substack{6\ 4 \\ 7}}{)}\underset{\substack{1\ 3\ 4 \\ 8}}{)}\underset{\substack{5 \\ 9}}{(}\underset{\substack{5 \\ 10}}{)}\underset{\substack{5\ 3 \\ 11}}{)}\underset{\substack{2\ 4 \\ 12}}{(}\underset{\substack{2\ 4 \\ 13}}{(}\underset{\substack{4\ 6 \\ 14}}{(}\underset{\substack{4\ 6 \\ 15}}{)}\underset{\substack{6 \\ 16}}{(}\underset{\substack{6 \\ 17}}{(}\underset{\substack{6 \\ 18}}{)}\underset{\substack{5 \\ 19}}{(}\underset{\substack{5 \\ 20}}{)}\underset{\substack{5\ 4\ 4 \\ 21}}{)}\underset{\substack{5\ 4\ 4 \\ 22}}{)}\underset{\substack{1\ 3\ 4 \\ 23}}{)}\underset{\substack{3\ 6 \\ 24}}{(}\underset{\substack{3\ 6 \\ 25}}{(}\underset{\substack{6 \\ 26}}{(}\underset{\substack{6 \\ 27}}{)}\underset{\substack{3 \\ 28}}{)}\underset{\substack{3 \\ 29}}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node

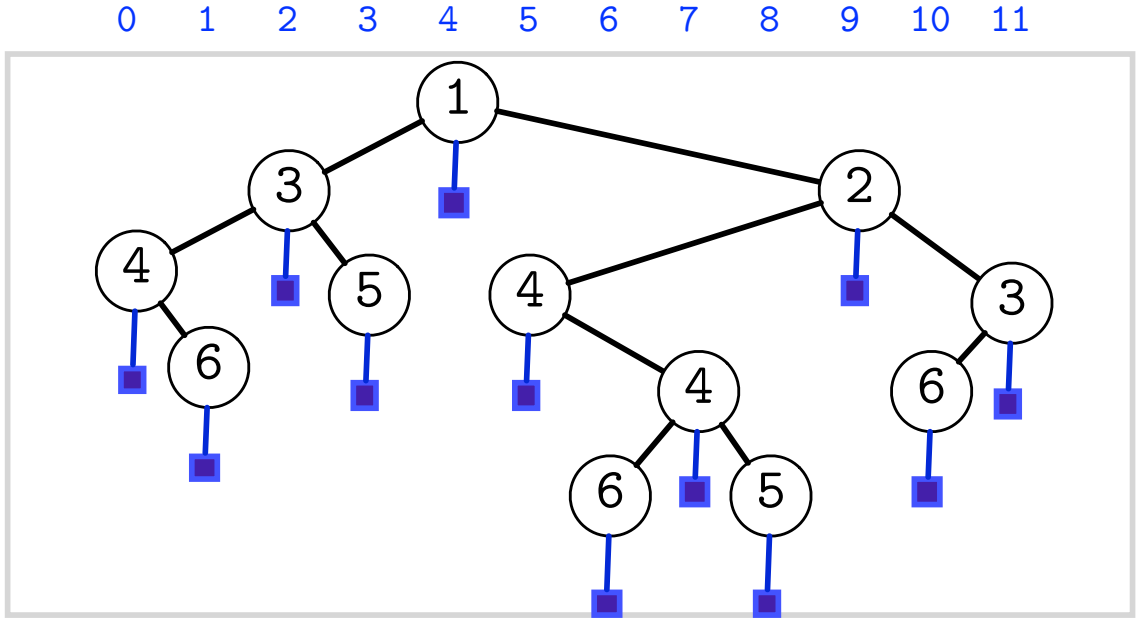


(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = \underset{\substack{1\ 3\ 4 \\ 0}}{(}\underset{\substack{1\ 3\ 4 \\ 1}}{(}\underset{\substack{1\ 3\ 4 \\ 2}}{(}\underset{\substack{6 \\ 3}}{(}\underset{\substack{6 \\ 4}}{)}\underset{\substack{6\ 4 \\ 5}}{(}\underset{\substack{6\ 4 \\ 6}}{)}\underset{\substack{6\ 4 \\ 7}}{)}\underset{\substack{5 \\ 8}}{)}\underset{\substack{5\ 3 \\ 9}}{(}\underset{\substack{5\ 3 \\ 10}}{)}\underset{\substack{5\ 3 \\ 11}}{)}\underset{\substack{2\ 4 \\ 12}}{)}\underset{\substack{2\ 4 \\ 13}}{(}\underset{\substack{4\ 6 \\ 14}}{(}\underset{\substack{4\ 6 \\ 15}}{)}\underset{\substack{6 \\ 16}}{(}\underset{\substack{6 \\ 17}}{(}\underset{\substack{6 \\ 18}}{)}\underset{\substack{5 \\ 19}}{(}\underset{\substack{5 \\ 20}}{)}\underset{\substack{5 \\ 21}}{)}\underset{\substack{5\ 4\ 4 \\ 22}}{)}\underset{\substack{5\ 4\ 4 \\ 23}}{)}\underset{\substack{3\ 6 \\ 24}}{(}\underset{\substack{3\ 6 \\ 25}}{(}\underset{\substack{6 \\ 26}}{(}\underset{\substack{6 \\ 27}}{)}\underset{\substack{3\ 6 \\ 28}}{)}\underset{\substack{3\ 2 \\ 29}}{(}\underset{\substack{3\ 2 \\ 30}}{)}\underset{\substack{3\ 2 \\ 31}}{)}$$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

(2) Add a leaf to each node



(3) DFS traversal to construct balanced parentheses sequence

$$BP_{ext} = (((()(()))()(()))()((()((()()(()))))()(((()())))))$$

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

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

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ 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 $excess(i) = rank(i+1, 1, BP_{ext}) - rank(i+1, 0, BP_{ext})$

For $0 \leq i \leq j < n$ we get:

00 **rmq**$_A(i,j)$
01 $ipos \leftarrow select(i+1, (), BP_{ext})$
02 $jpos \leftarrow select(j+1, (), BP_{ext})$
03 **return** $rank(rmq^{\pm 1}_{excess}(ipos, jpos+1), (), BP_{ext})$

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ bits)

- Added leaves are used to navigate to inorder index nodes
- Select on a pattern „`()`" 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
- $rmq^{\pm 1}_{excess}(ipos, jpos + 1)$ is the position of closing parenthesis of the leaf node z added to $LCA(v, w)$
- In-order number of $LCA(v, w)$ corresponds to index of minimum in $A[i, j]$ and can be determined by a rank operation on pattern „`()`"

Next: $o(n)$ data structure to support $rmq^{\pm 1}_{excess}$ queries on BP_{ext}

$\langle O(n), O(1) \rangle$ solution ($4n + o(n)$ 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 $o(n)$

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

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 ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

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 ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

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

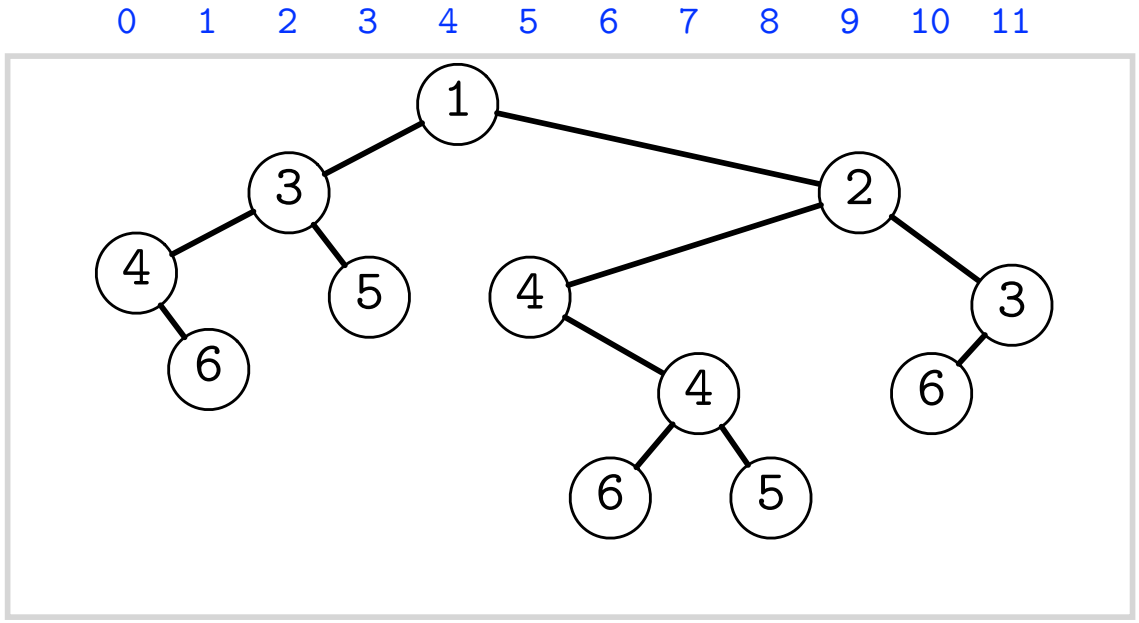
$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

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



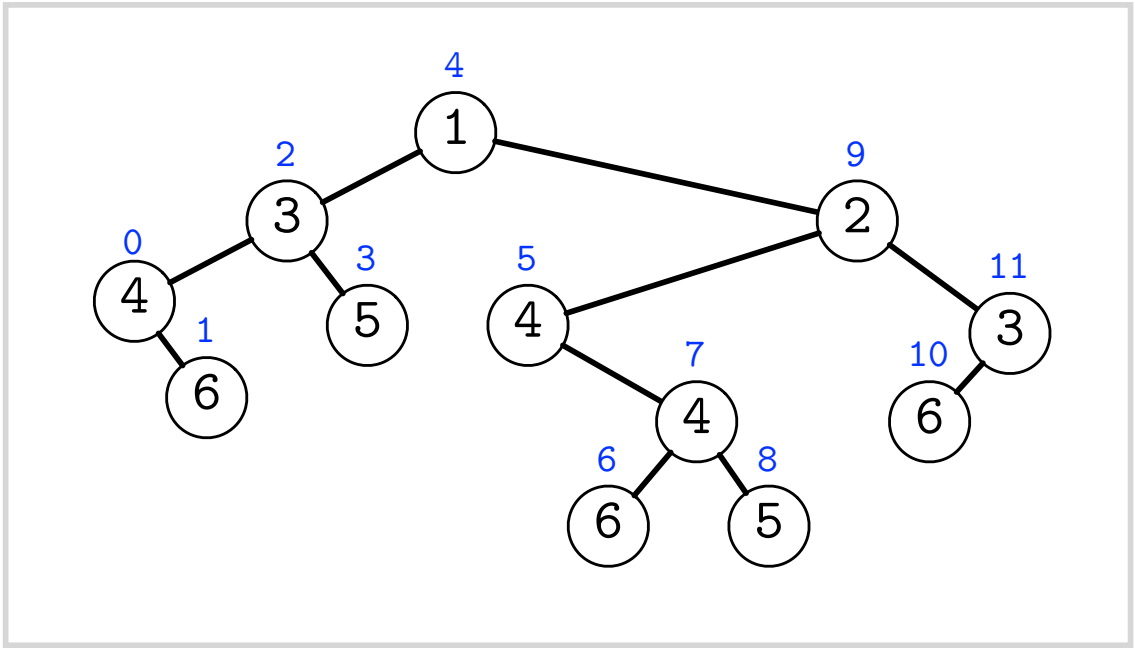
$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Observation

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

Next: Approach which does not require additional nodes.

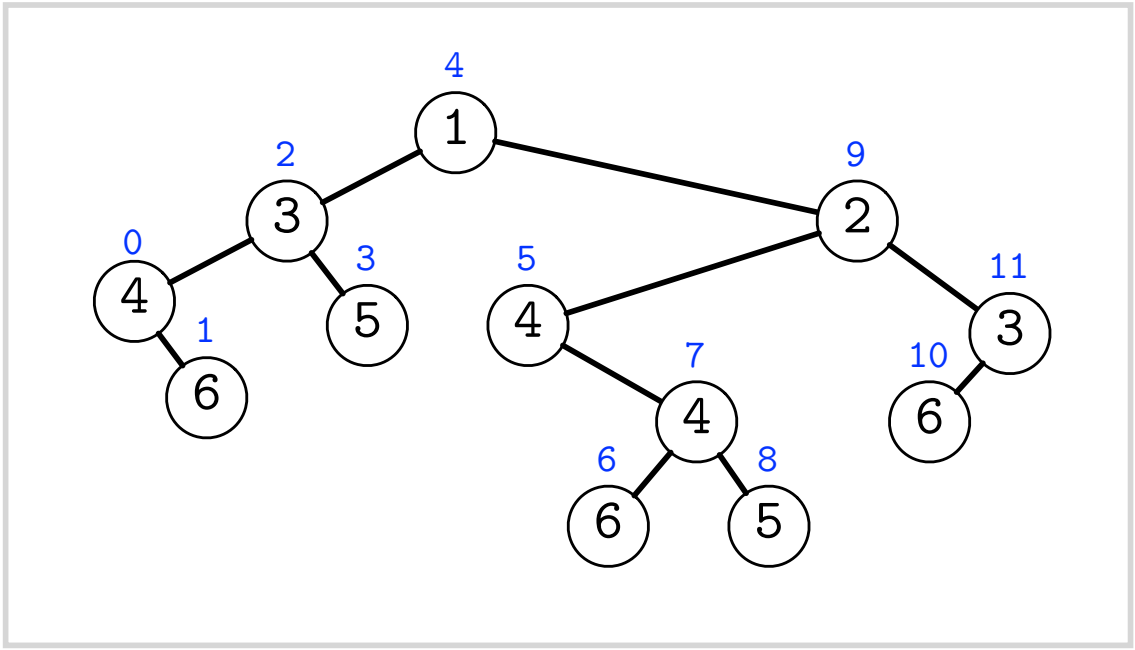
- Cartesian Tree (CT) is a binary tree of n nodes
- Transform CT into general tree of $n + 1$ nodes



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

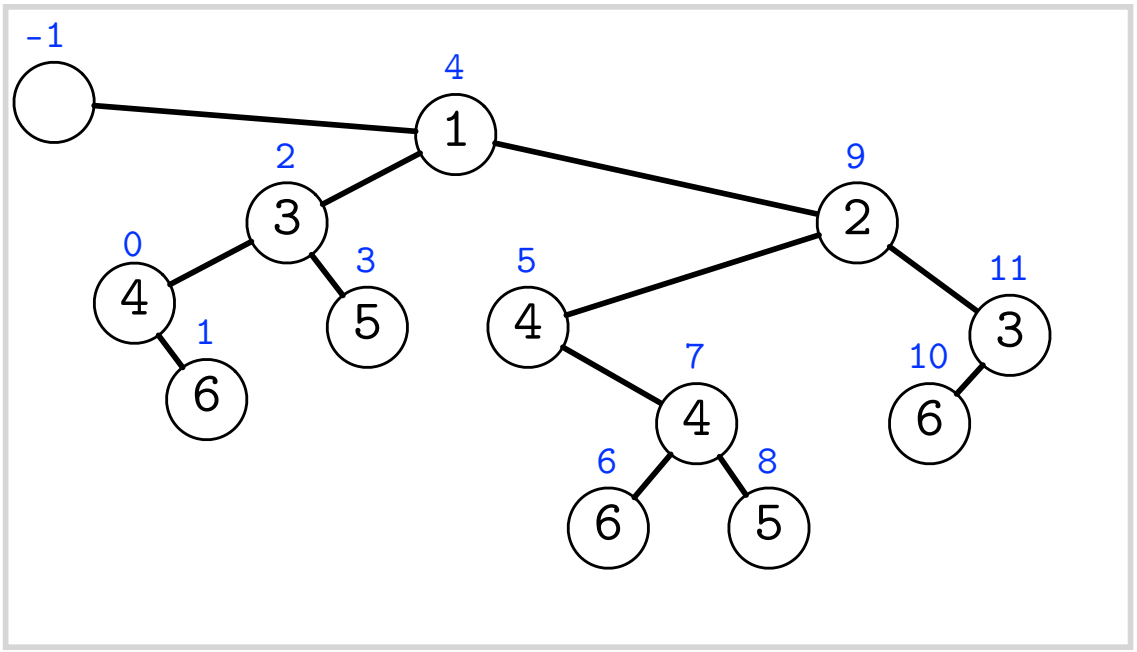
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

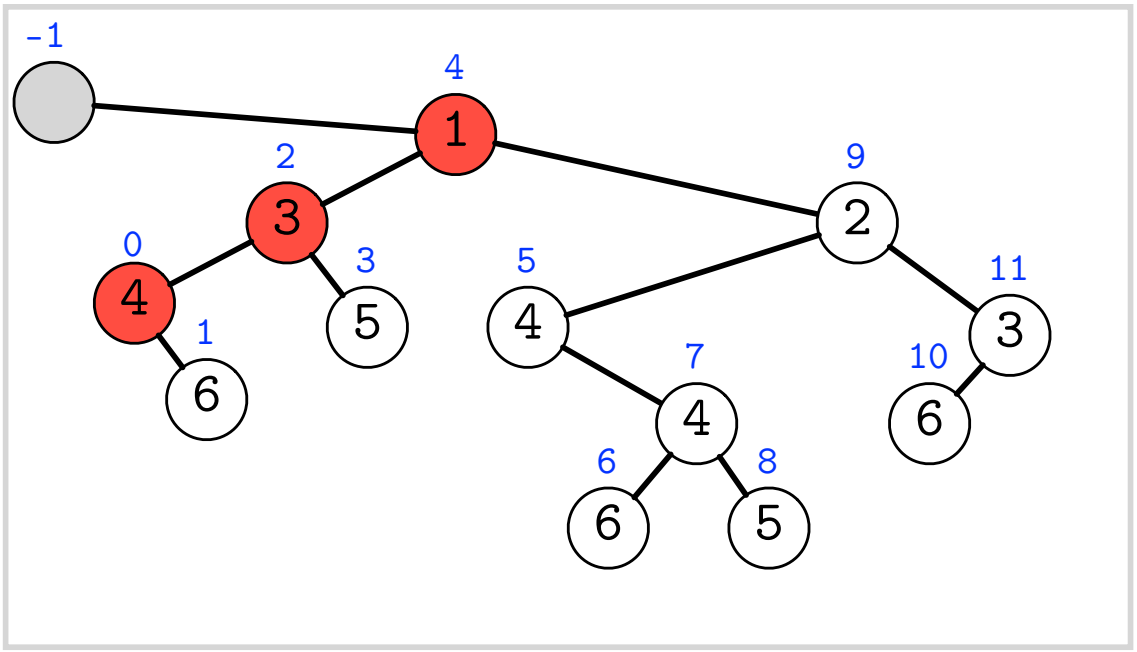
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

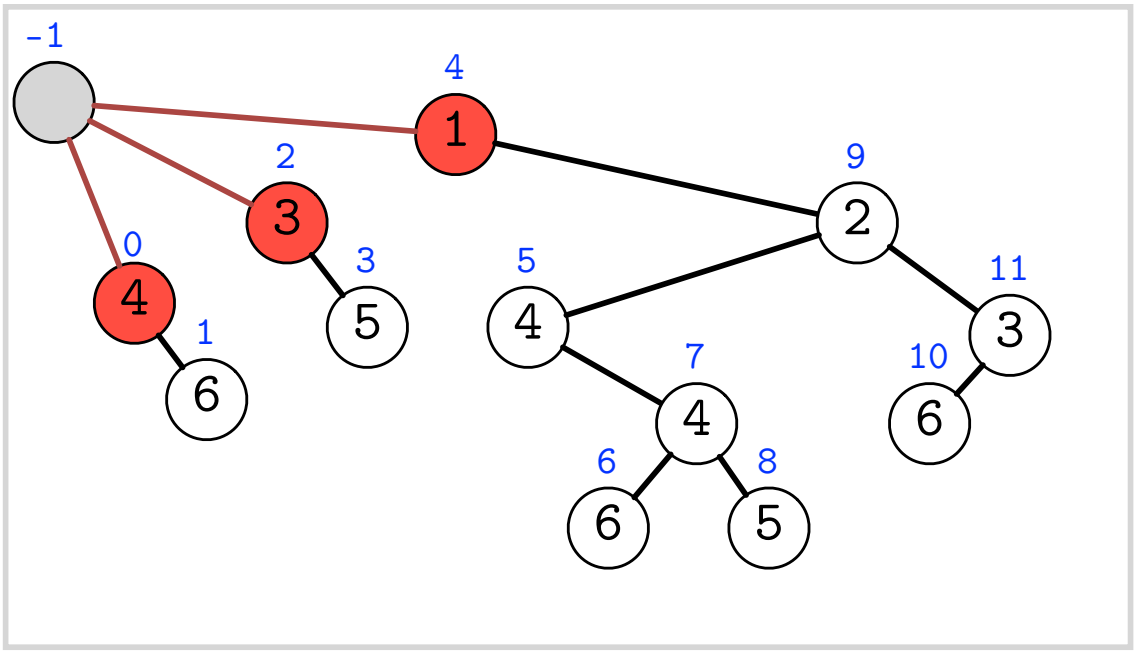
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

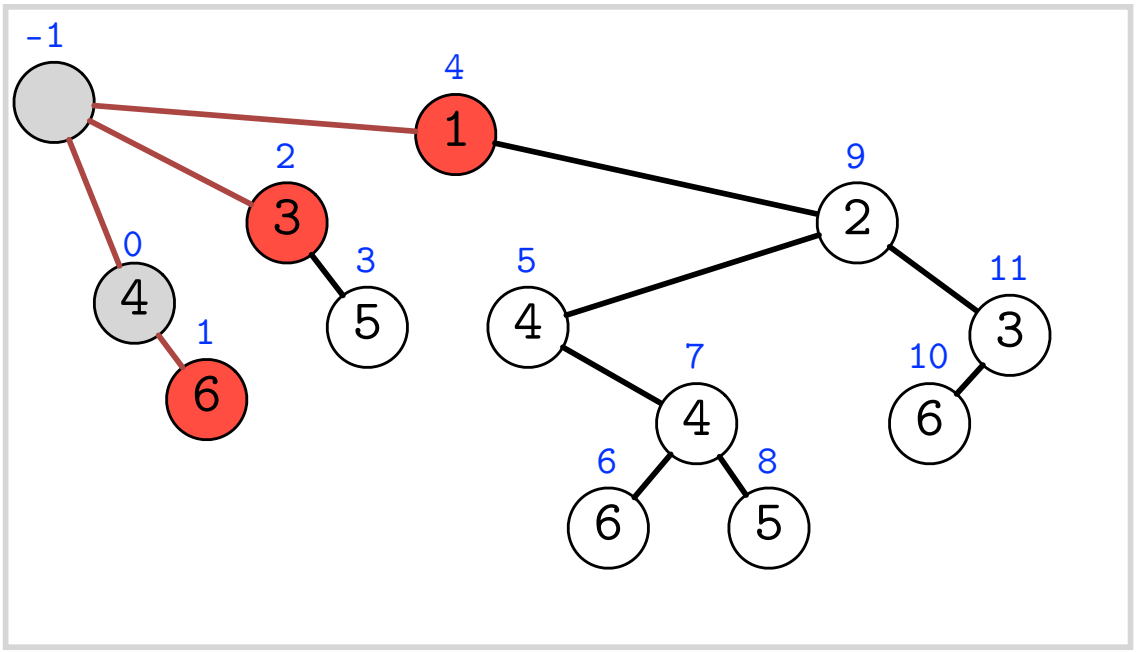
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

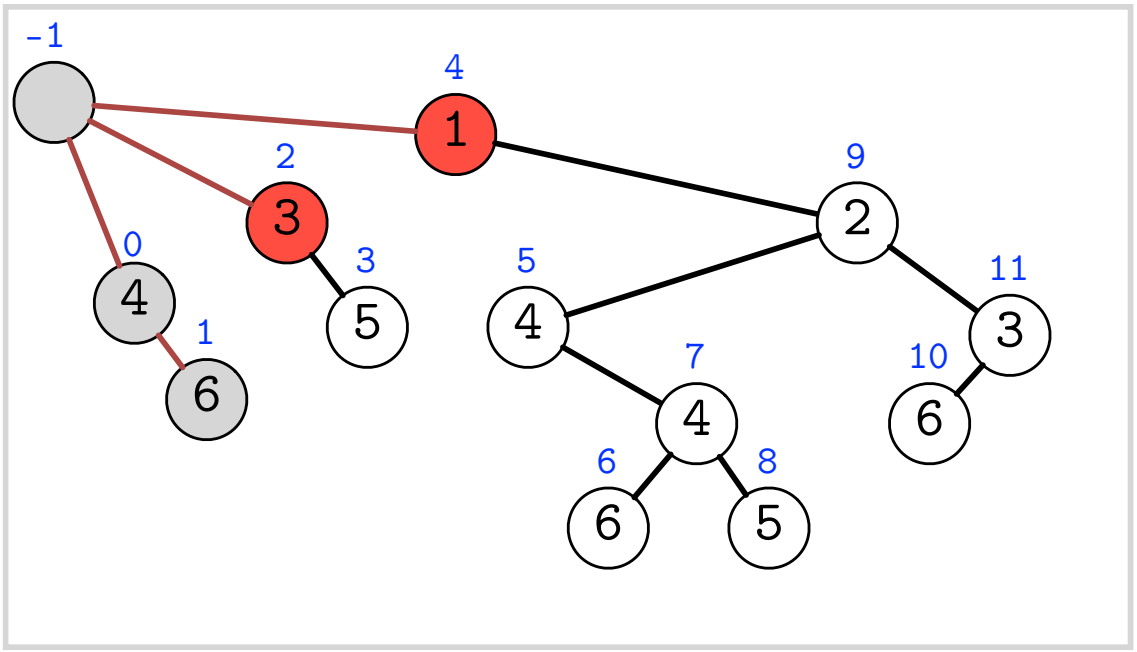
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

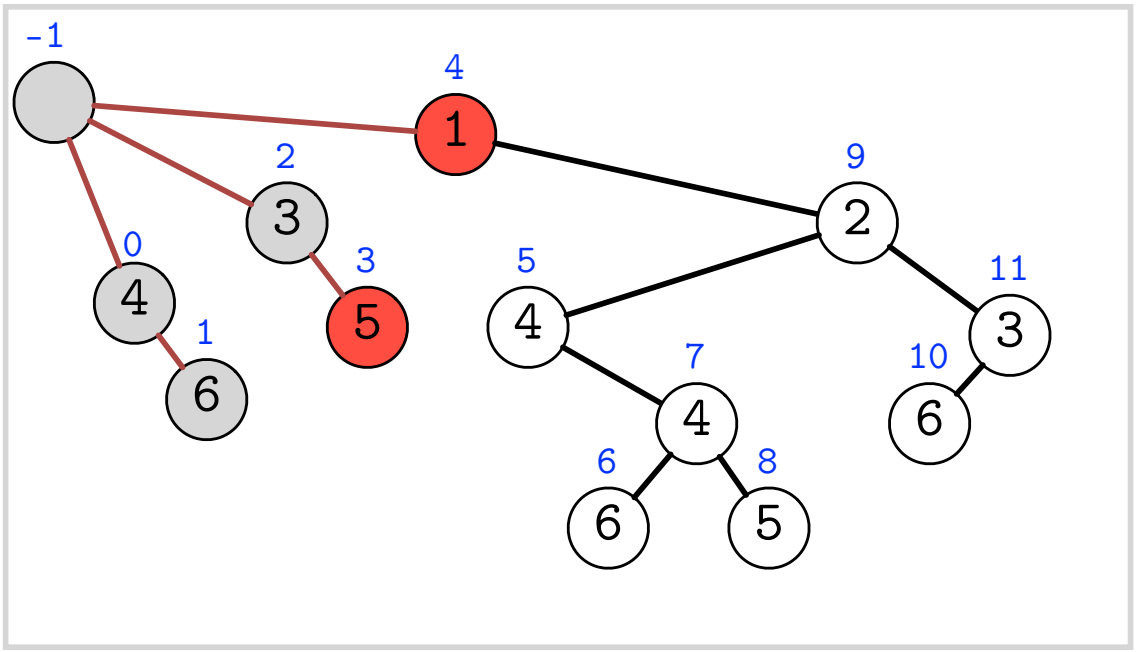
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

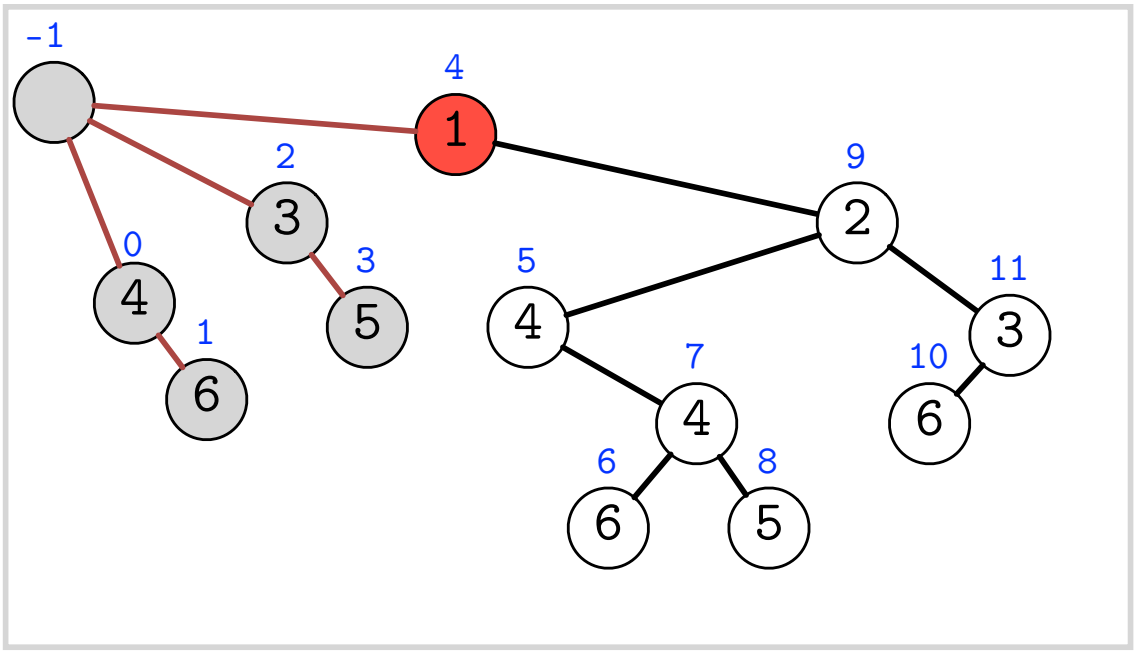
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

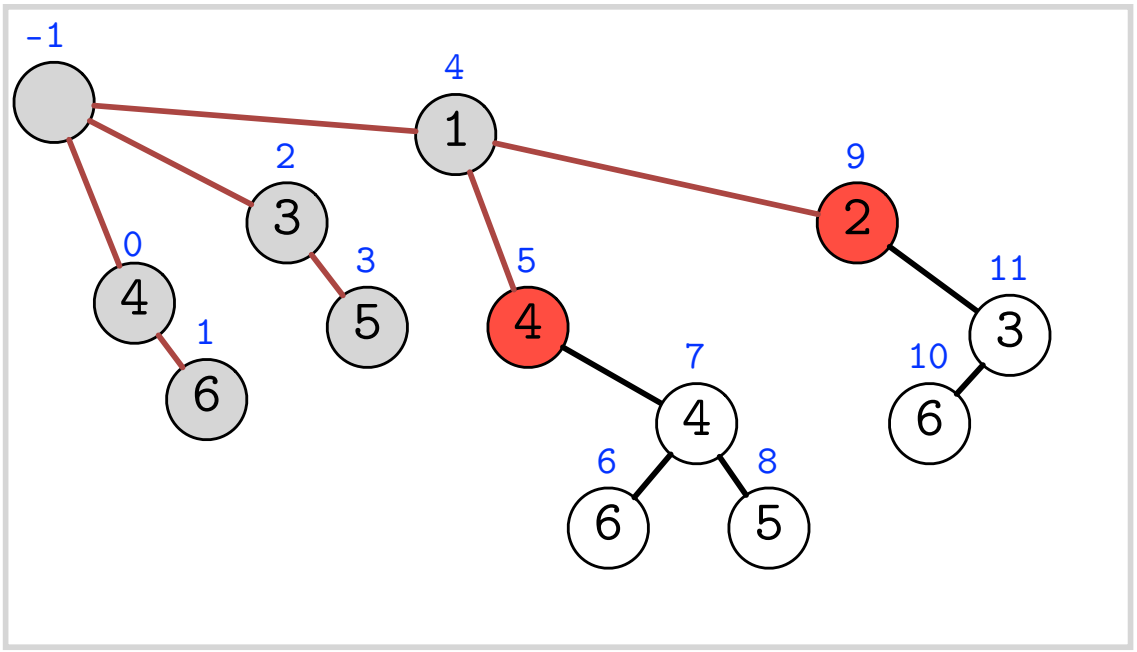
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

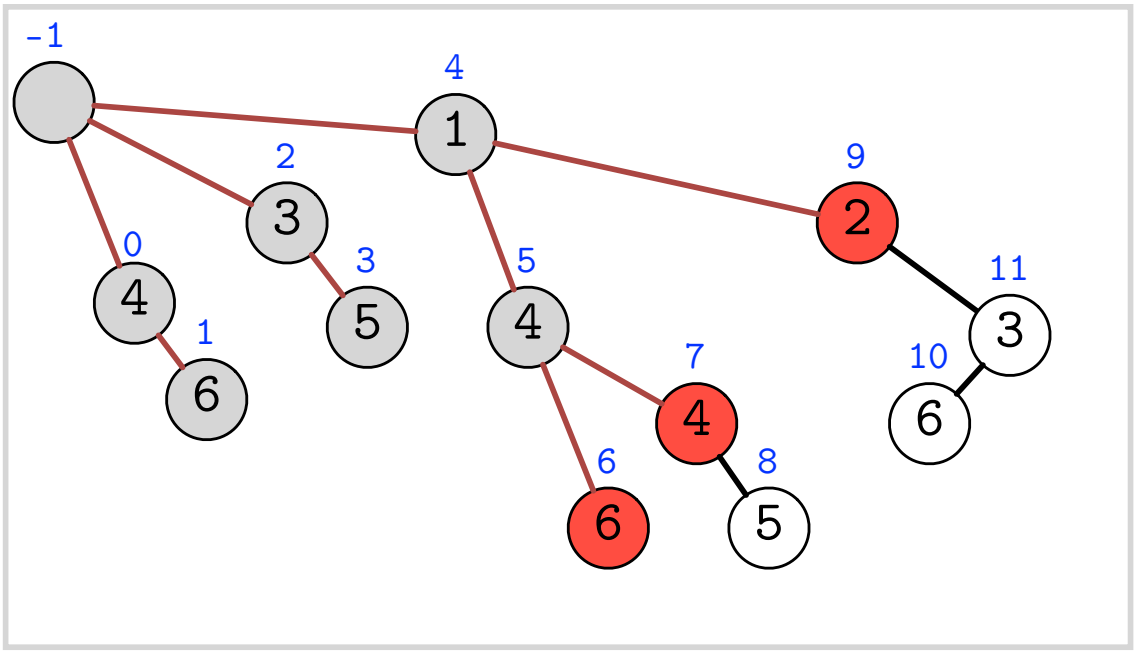
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

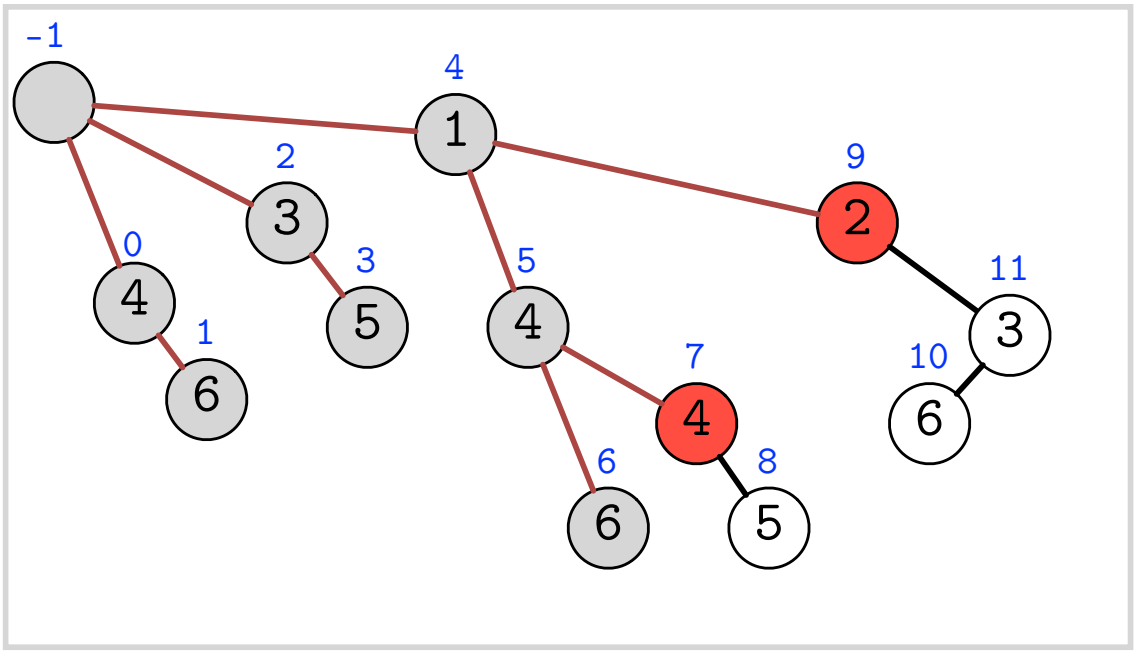
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

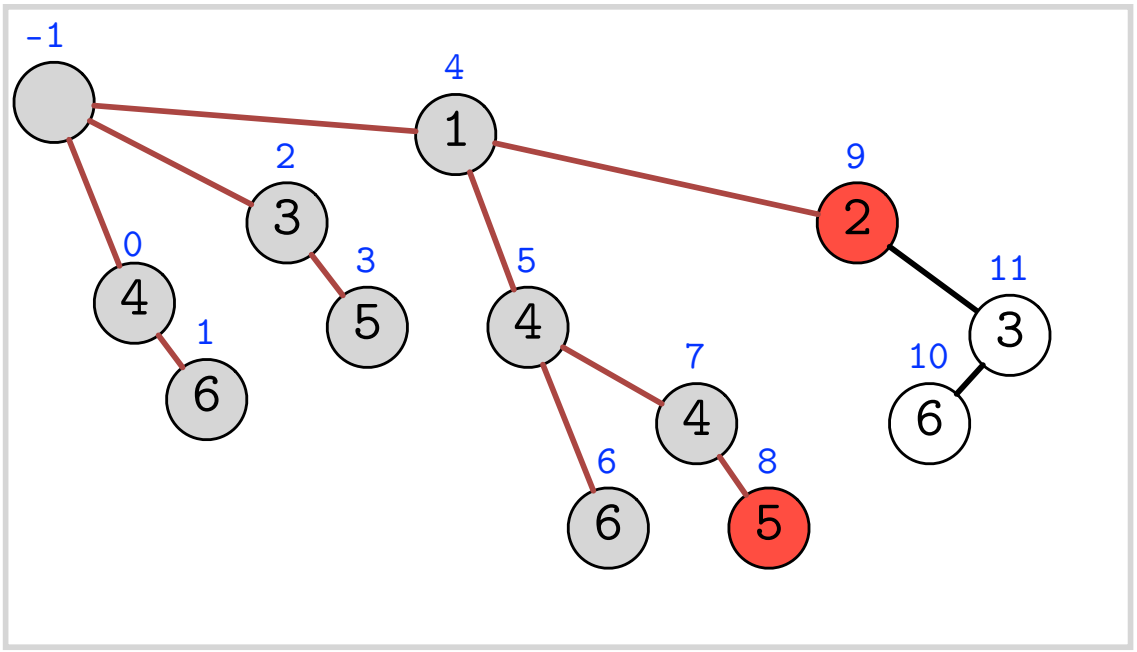
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

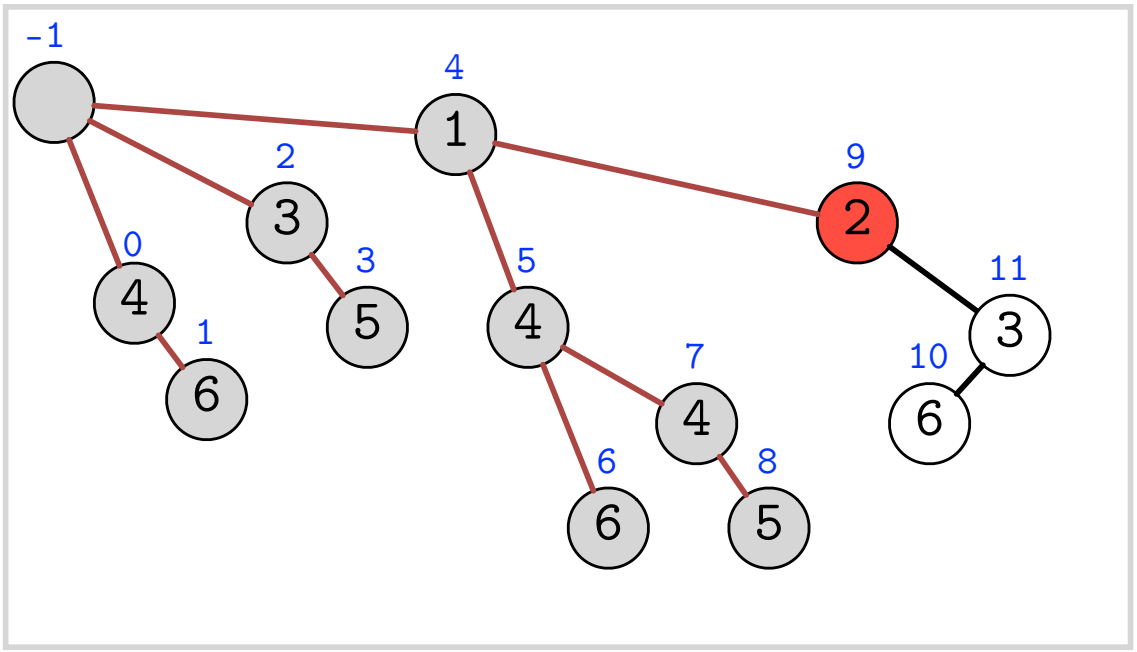
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

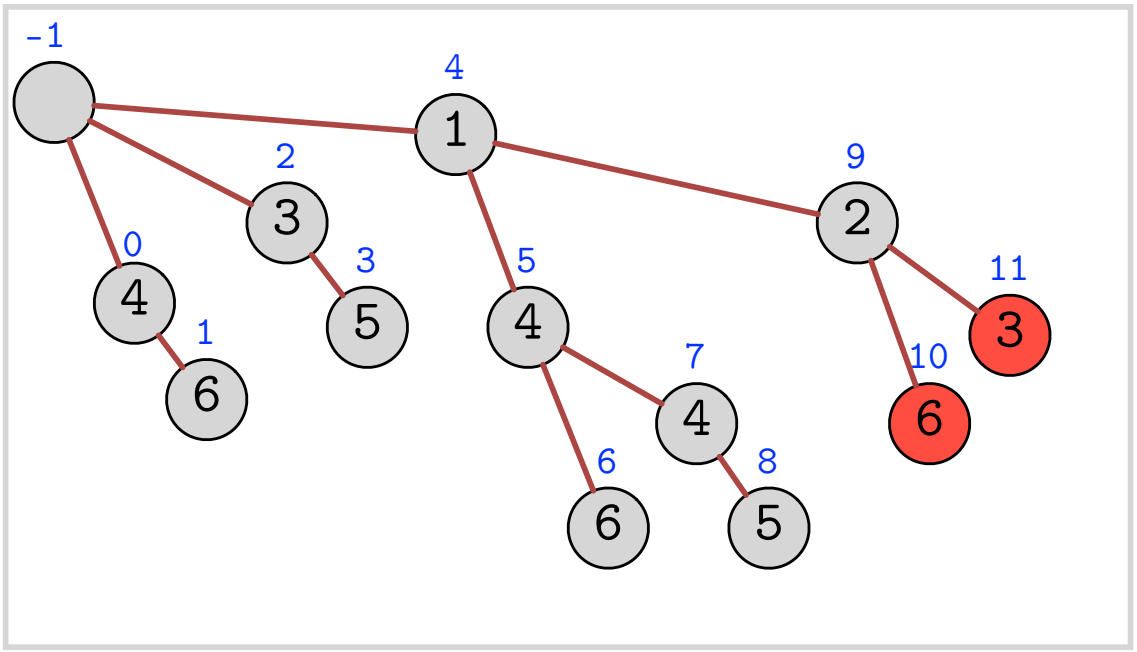
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

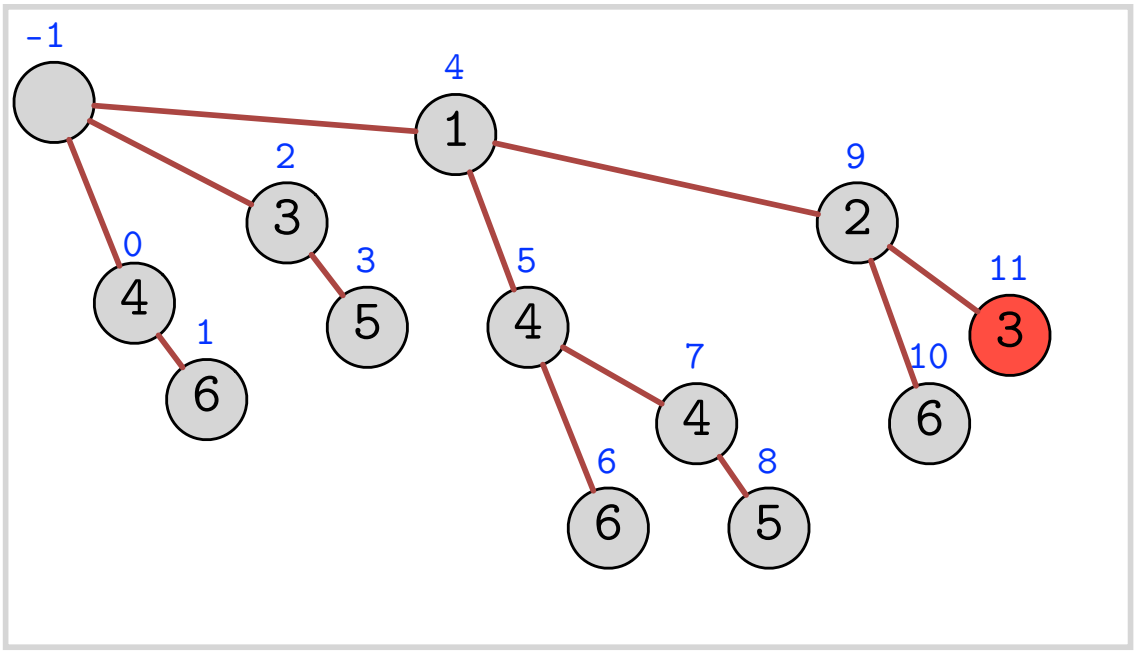
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Transformation

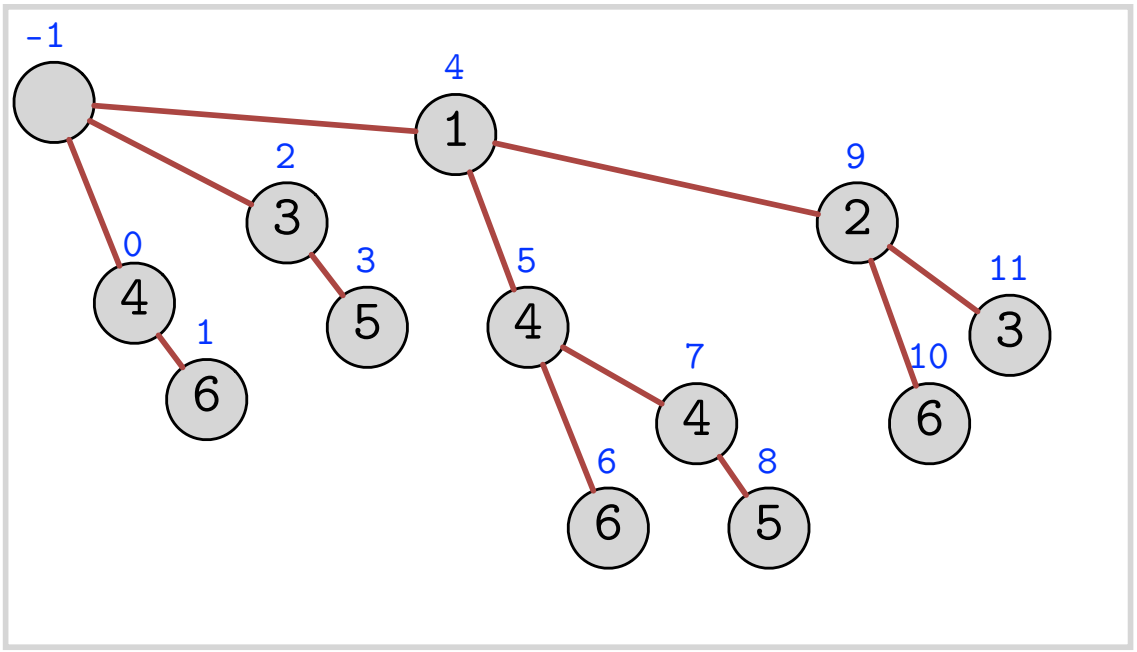
- Add a new root node to the leaf of the original root
- For each node v (starting at the root) take the right child w, and add the nodes on the leftmost path from w as children of v



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ 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 ($2n + o(n)$ 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 parenthes sequence of the general tree

For $0 \leq i \leq j < n$ we get:

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

Where $rmq^{\pm 1}_{excess}$ returns the rightmost position of the minimal value.

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ 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 parenthes sequence of the general tree

For $0 \leq i \leq j < n$ we get:

```
00  rmq_A(i,j)
01      ipos ← select(i + 2, (, BP)
02      jpos ← select(j + 2, (, BP)
03      return rank(rmq^{±1}_{excess}(ipos − 1, jpos), (, BP) − 1
```

Where $rmq^{\pm 1}_{excess}$ returns the rightmost position of the minimal value.

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ 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 parenthes sequence of the general tree

For $0 \leq i \leq j < n$ we get:

```
00  rmq_A(i,j)
01      ipos ← select(i + 2, (, BP)
02      jpos ← select(j + 2, (, BP)
03      return rank(rmq^{±1}_excess(ipos − 1, jpos), (, BP) − 1
```

Where $rmq^{\pm 1}_{excess}$ returns the rightmost position of the minimal value.

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ 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 parenthes sequence of the general tree

For $0 \leq i \leq j < n$ we get:

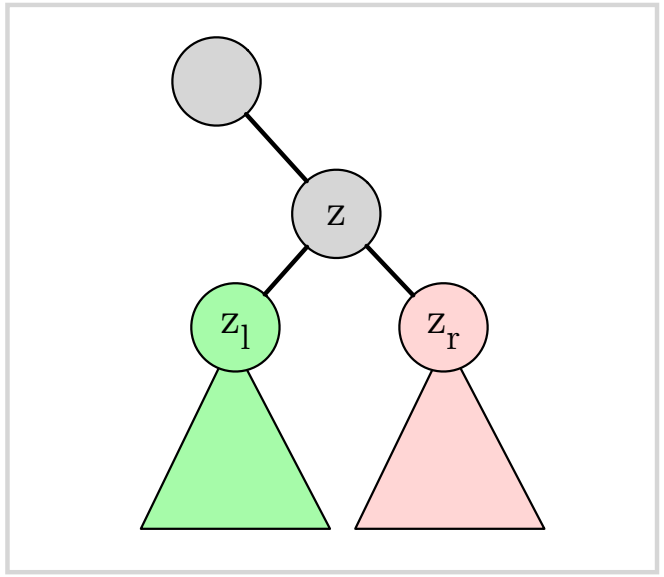
00 **rmq**$_A(i,j)$
01 $ipos \leftarrow select(i + 2, (, BP)$
02 $jpos \leftarrow select(j + 2, (, BP)$
03 **return** $rank(rmq^{\pm 1}_{excess}(ipos - 1, jpos), (, BP) - 1$

Where $rmq^{\pm 1}_{excess}$ returns the rightmost position of the minimal value.

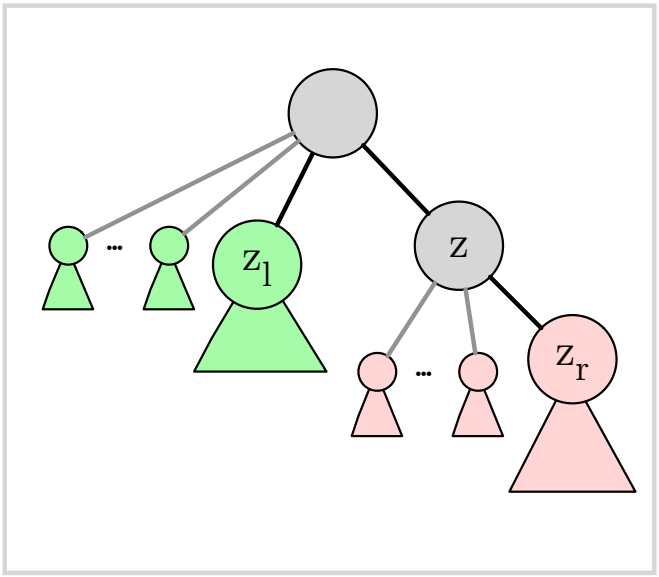
$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

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.

Situation in binary tree



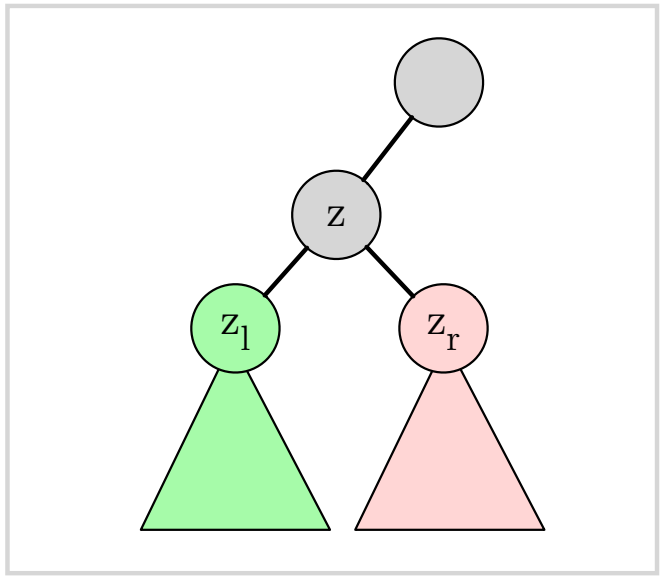
Situation in general tree



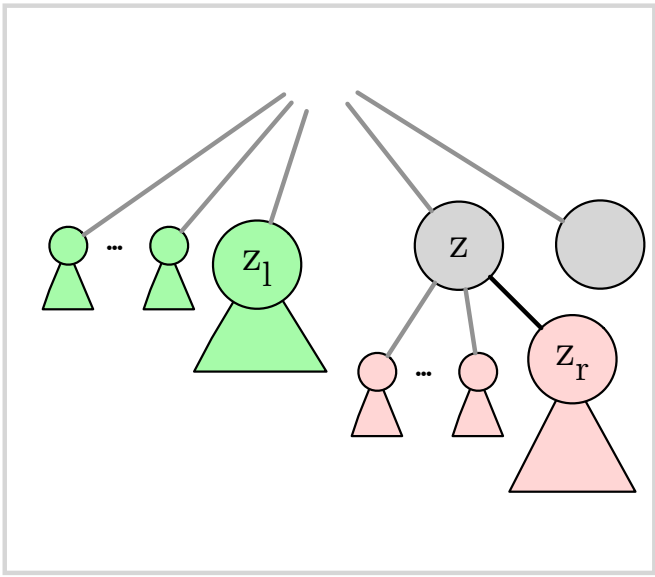
$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

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.

Situation in binary tree



Situation in general tree



$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

First case: $v \neq z$ and $w \neq z$

- v's opening parenthesis is the green area
- w's opening parenthesis is the the red area
- the (righmost) $\pm 1 RMQ$ will return the position p of the closing parenthesis of z_l
- 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 $A[i, j]$

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

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 (righmost) $\pm 1 RMQ$ will return the position p of the closing parenthesis of z_l
- 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 $A[i, j]$

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

Third case: $v = z$ and $w \neq z$:

- v's opening parenthesis is z's opening parenthesis now
- w's opening parenthesis is the the red area
- the (righmost) $\pm 1 RMQ$ will return the position p of the closing parenthesis of z_l
- 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 $A[i, j]$

$\langle O(n), O(1) \rangle$ solution ($2n + o(n)$ bits)

Karlsruhe Institute of Technology

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 (righmost) $\pm 1 RMQ$ will return the position p of the closing parenthesis of z_l
- 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 $A[i, j]$

Conclusion

Range Minimum Queries (RMQ)s over an array A can be answered in constant time after prepocessing a $2n + o(n)$ space data structure in linear time.
Applications:

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

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)