cis112
Tree

Haluk O. Bingol

Faculty of Computer and Information Sciences
Yeditepe University

v2025-04-20

Content

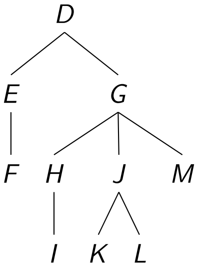
[[1], [2], [3], [4], [5], [6], [7]]

Motivation

Hierarchy

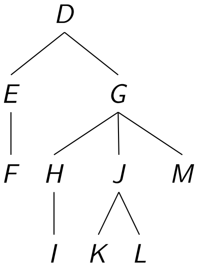
Directory hierarchy

```
MyDisk (D)
|- Private (E)
|  |- Family (F)
|- School (G)
|  |- CIS111 (H)
|  |  |- Project-1 (I)
|  |- CIS112 (J)
|  |  |- Project-1 (K)
|  |  |- Project-2 (L)
|  |- CIS114 (M)
```

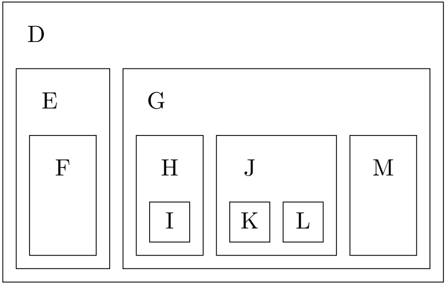


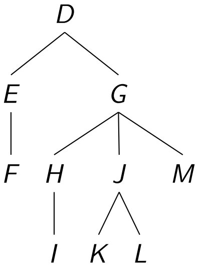
Organizational hierarchy

```
General Manager {
    VP {
        Production {
        }
    }
    VP {
        Human Resources {
            class I {
            }
        }
        Finance {
            Accounting {}
            Treasury  {}
        }
        Sales {}
    }
}
```



Subset hierarchy

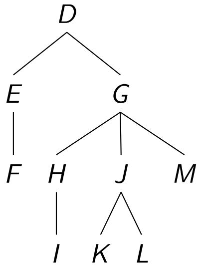




[[1], [3]]

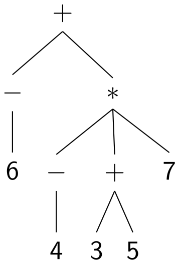
Inner class hierarchy

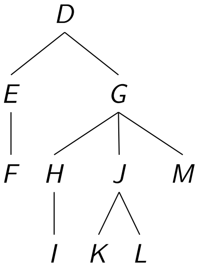
```
class D {
    class E {
        class F {
        }
    }
    class G {
        class H {
            class I {
            }
        }
        class J {
            class K {}
            class L {}
        }
        class M {}
    }
}
```



Expressions

$$(-F) + ((-I) \times (K + L) \times M)$$



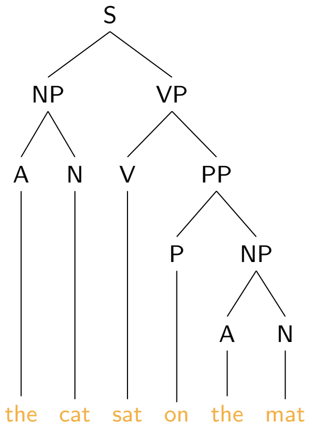


$$(-6) + (-4) \times (3 + 5) \times 7$$

Linguistics

"the cat sat on the mat."

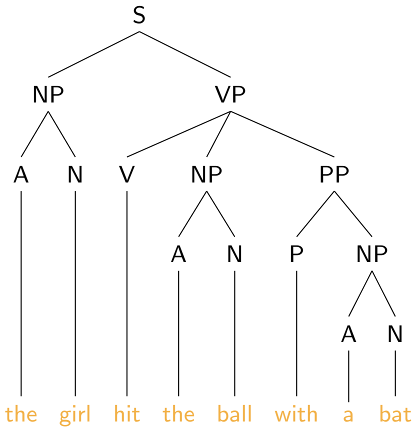
S: Sentence
N: Noun
V: Verb
A: Article
NP: Noun Phrase
PP: Prepositional Phrase
VP: Verb Phrase



Linguistics

"the girl hit the ball with a bat."

S: Sentence
N: Noun
V: Verb
A: Article
NP: Noun Phrase
PP: Prepositional Phrase
VP: Verb Phrase



Definitions

[[1], [2], [3], [4], [5], [6], [7]]

Tree and Subtree

Definition (Mathematical)

A tree T is a finite, non-empty set of nodes

$$T = \{r\} \cup T_1 \cup T_2 \cup \cdots \cup T_n,$$

with the following properties:

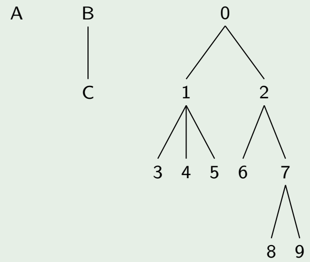
1. A designated node of the set, r, is called the root of the tree; and
2. The remaining nodes are partitioned into $n \geq 0$ subsets, T_1, T_2, ..., T_n, each of which is a tree. T_i is called a subtree.

Q. What is the minimum number of vertices in a tree?

Notation.

- $T = \{r, T_1, T_2, \ldots, T_n\}$ denotes the tree T.
- V is the set of vertices (nodes) in T.
- $v \in V$ is a vertex (node) of T.

Example



A is a tree with 1 node.
B is a tree with 2 nodes.
0 is a tree with 9 nodes.
1 is a tree with 4 nodes.
2 is a tree with 5 nodes.
4 is a tree with 1 node.

Degree and Leaf

Definition

Let $T = \{r, T_1, T_2, \ldots, T_n\}$ be a tree.
The degree of a node is the number of subtrees associated with that node.

Example

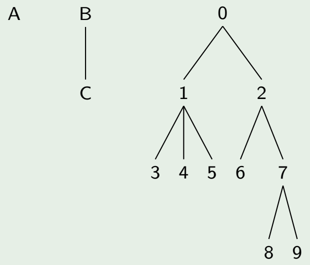
The degree of r in tree $T = \{r, T_1, T_2, \ldots, T_n\}$ is n.

Definition

A node of degree 0 is called a leaf.

Q. Is it possible to have a negative degree?

Example



Nodes A, C, 3 are leafs.
Degree of node B is 1.
Degree of node 0 is 2.
Degree of node 1 is 3.

Parent, Child and Siblings

Definition

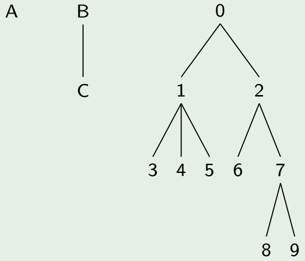
Let $T = \{r, T_1, T_2, \ldots, T_n\}$, $n \geq 0$ be a tree T.
Let r_i be the root of subtree T_i . Then

- r is called the parent of r_i .
- r_i is called a child of r
- Roots r_i and r_j of distinct subtrees T_i and T_j of tree T are called siblings.

Q. What is the minimum and maximum number of

- parents of a node a?
- children of a node a?
- siblings of a node a?

Example



Node B is parent of node C.
Node C is a child of node B.
Nodes 3 and 4 are siblings.

Path and Path Length

Definition

Let $T = \{r, T_1, T_2, \ldots, T_n\}$ be a tree T. Let V be the set of nodes in T.

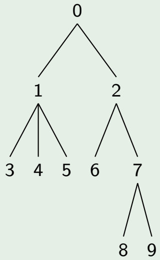
- A path is a non-empty sequence of nodes $P = (v_1, v_2, \ldots, v_k)$ where, $v_i \in V$ for $1 \leq i \leq k$, such that v_i is parent of v_{i+1}.
- The length of path P is $k - 1$.

Remark.

- Direction of a path is from root to leaf.
- There is a unique path from root to any node in the tree.

Q. Is (v) a path?

Example



$(2, 7, 9)$ is a path.
$(0, 1)$ is a path.
Unique path to node 8 is the path $(0, 2, 7, 8)$

Ancestor and Descendant

Definition

Let T be a tree with the set of nodes V. Suppose there exists a path P from v_i to v_j for $v_i, v_j \in V$, Then

- the vertex v_i is an ancestor of v_j;
- the vertex v_j is a descendant of v_i.

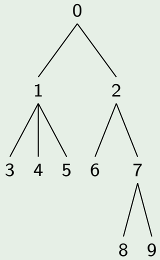
If the length of P is non-zero, then

- the vertex v_i is a proper ancestor of v_j;
- the vertex v_j is a proper descendant of v_i.

Remark.

- A path is from ancestor to descendant.
- Vertex v is ancestor of itself.
- Vertex v is descendant of itself.

Example



2 is ancestor of 7.
9 is descendant of 2.

Level (Depth) and Height

Definition

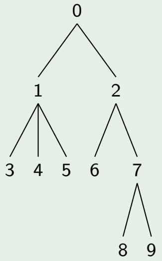
Let T be a tree with the set of nodes V.

- The level (depth) of a node $v \in V$ in a tree T is the length of the unique path in T from its root r to the node v.
- The height of a node $v \in V$ in a tree T is the length of the longest path from node v to a leaf.
- The height of a tree T is the height of its root r.

Remark.

- The root r of T is at level-0.
- The roots of the subtrees of r are at level-1.
- The leaves are at height 0.

Example



Node 0 is at level-0.
Node 1 is at level-1.
Node 4 is at level-2.
Node 9 is at level-3.
Height of node 7 is 1.
Height of node 2 is 2.
Height of node 0 is 3.
Height of tree T is 3.

Tree as a Data Structure

N-ary Trees

Definition (Data Structure)

An N-ary tree T is a finite set of nodes with the following properties:

1. Either the set is empty, $T = \emptyset$; or
2. The set consists of a root, r, and exactly N distinct N-ary trees, T_i.

   I.e., the remaining nodes are partitioned into $N \geq 0$ subsets, T_0, T_1, ..., T_{N-1}, each of which is an N-ary tree such that $T = \{r, T_0, T_1, \ldots, T_{N-1}\}$.

Note that

- The degree of each node of an N-ary tree is either zero or N
- The empty tree, $T = \emptyset$, is a tree. That is, it is an object of the same type as a non-empty tree

Binary Trees

Definition (Data Structure)

A binary tree T is a finite set of nodes with the following properties:

1. Either the set is empty, $T = \emptyset$; or
2. The set consists of a root, r, and exactly two distinct binary trees T_L and T_R, $T = \{r, T_L, T_R\}$.

The tree T_L is called the left subtree of T, and the tree T_R is called the right subtree of T.

Note that

- The degree of each node of an binary tree is either 0, 1 or 2.
- A binary tree of height $h \geq 0$ has at most $2^{h+1} - 1$ nodes
- Therefore the height of a binary tree with n nodes is at least $\lceil \log_2 n + 1 \rceil - 1$.

Ordered Trees

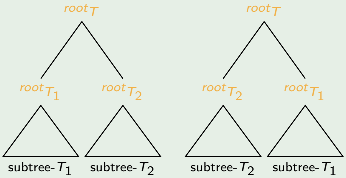
Definition

An ordered tree is a tree in which the order of the subtrees matters.

Warning. Unless stated otherwise, all trees we deal with are ordered trees.

Example

Tree $T_{12} = \{r, T_1, T_2\}$ is different than tree $T_{21} = \{r, T_2, T_1\}$.



Balanced Trees

Balanced Trees

Definition

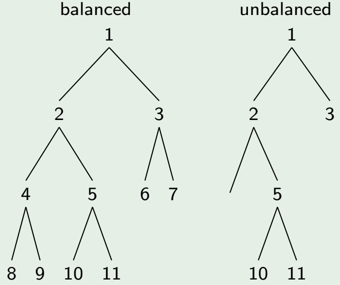
A tree $T = \{r, T_1, T_2, \ldots, T_n\}$ is balanced iff

- T is empty or
- $T_1, T_2, \ldots, T_n$ have "almost the same height".

Remark.

- Consider "almost the same height" as difference of 1.
- How to keep a tree balanced is an important issue that we will deal with

Example



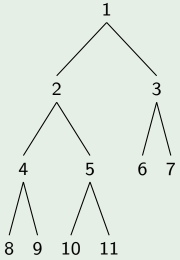
Complete Binary Tree

Definition

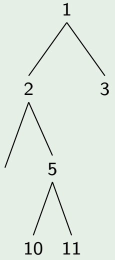
A binary tree $T = \{r, T_L, T_R\}$ is called complete binary tree iff each node has either 0 or 2 degrees.

Example

complete binary tree



not



due to node 2.

Full Binary Tree

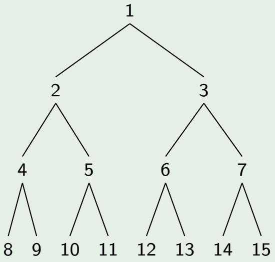
Definition

A binary tree T is called full binary tree iff all the leafs in T are at level h. [[7]]

Remark.

- Some books, such as [[2]], calls it *complete binary tree*.
- A full binary tree is a special form with the property that maximum possible number of nodes in a minimum possible height.

Example



Relations in Number of Nodes and Height

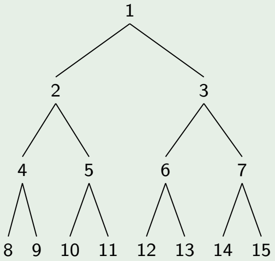
In a full binary tree

- at level-0, there is only $2^0 = 1$: node: 1
- at level-1, there are $2^1 = 2$: nodes: 2, 3
- at level-2, there are $2^2 = 4$: nodes: 4, 5, 6, 7
- at level-3, there are $2^3 = 8$: nodes: 8, 9, 10, 11, 12, 13, 14, 15

Generalization.

- In a full binary tree
  - at level-ℓ, there are 2^ℓ nodes.
  - height $h \implies n = 2^{h+1} - 1$.
  - number of nodes $n \implies h = \log_2(n+1) - 1$.
- In any binary tree, $n \leq 2^{h+1} - 1$.

Example



A full binary tree. Levels: 0, 1, 2, and 3. The number of nodes: $n = 15$ and height: $h = 3$. Then $n = 2^{h+1} - 1$ and $h = \log_2(n+1) - 1$.

Theorem.

$$(1 + x^1 + \ldots + x^n)(x - 1) = x^{n+1} - 1$$

Relations in Number of Nodes and Height

Height of a tree is very important

- In search, number of steps is directly related to the height
- The relation between n and h is logarithmic, i.e., $h = \lceil \log_2(n+1) - 1 \rceil$. It is due to nonlinearity of tree data structure
- Therefore, searching in trees is $O(\log n)$ rather than $O(n)$, which is a big improvement for big n values.
- Because of these nice properties, trees are very frequently used in CS.

In any binary tree,

- Given h, $n \leq 2^{h+1} - 1$
- Given n, $h \geq \lceil \log_2(n+1) - 1 \rceil$

n	$\log_2(n+1) - 1$	h
1,000	8.97	9
1,000,000	18.93	19
10,000,000	22.25	23
100,000,000	25.58	26
1,000,000,000	28.90	29

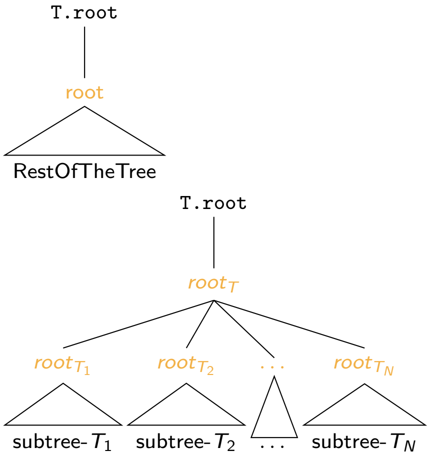
Implementation

Tree

Tree T is a data structure

- with a $T.root$ pointing to the root node of the tree
- $T.root.parent = \text{NIL}$
- Let x be a node of T
  - If $x.parent = \text{NIL}$, then x is the $root$, i.e., root is the only node with parent is NIL
  - If $x.subtree_i = \text{NIL}$, then x has no subtree T_i i.e., the subtree T_i is empty

[[1], [2], [3], [4], [5], [6], [7]]

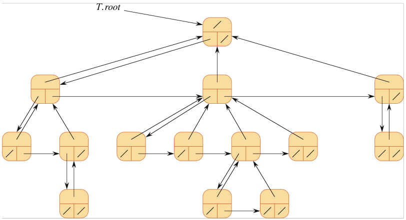


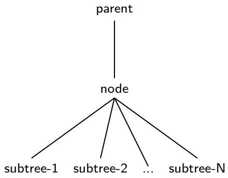
N-ary Tree

Algorithm 1: Node for N-ary tree:
General case

```
begin
    data
    parent            // reference to the parent
    child1            // reference to the first subtree
    child2            // reference to the second subtree
    ...
    childN            // reference to the N'th subtree
```

Remark. In the most general case of a tree, each node may have different number N of subtrees. This is a problem.





N-ary Trees: Left-child, right-sibling representation

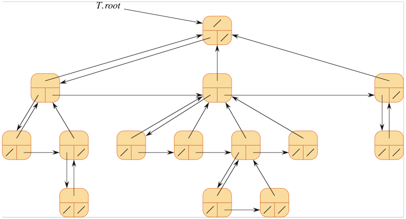
Algorithm 2: Node for Left-child, right-sibling

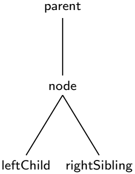
```
begin
    data
    parent              // reference to the parent
    leftChild           // reference to the left child
    rightSibling        // reference to the right sibling
```

Use node with three pointers:

- parent: points the parent
- leftChild: points the left child
- rightSibling: points the right sibling

Remark. Now node becomes uniform, i.e., independent of degree.





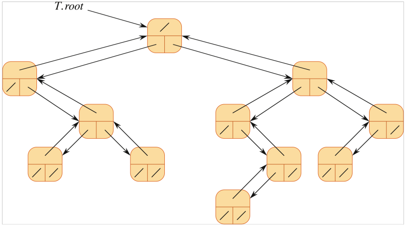
Binary Tree

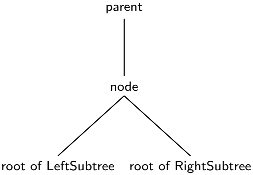
Algorithm 3: Node for binary tree

```
begin
    data
    parent          // reference to the parent
    left            // reference to the left child
    right           // reference to the right child
```

In binary tree, each node has 0, 1 or 2 subtrees.
Use node with three pointers:

- parent: points the parent
- left: points the left child
- right: points the right child





MyNode in Java

Algorithm 4: Node for binary tree

```
begin
    data
    parent          // reference to the parent
    left            // reference to the left child
    right           // reference to the right child
```

MyNode implements interface NodeBinaryI.

```java
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

```java
public class MyNode<T> implements NodeBinaryI<T>{

  public T data;
  public MyNode<T> left;
  public MyNode<T> right;

  public MyNode() {
    this(null);
  }

  public MyNode(T data) {
    this.data = data;
    left = null;
    right = null;
  }

  ...

  @Override
  public String toString() {
    return "[MyNode: data=" + data + "]";
  }

}
```

MyTree in Java: Constructors

```java
public class MyBinaryTree<T> {

  private MyNode<T> root;

  public MyBinaryTree() {
    root = null;
  }

  public MyBinaryTree(T data) {
    root = new MyNode<>(data);
  }

  public MyBinaryTree(
      T data
      , MyBinaryTree<T> left
      , MyBinaryTree<T> right
    ) {
    root = new MyNode<>(data);
    if (left == null) {
      root.left = null;
    } else {
      root.left = left.root;
    }
    if (right == null) {
      root.right = null;
    } else {
      root.right = right.root;
    }
  }
  ...
}
```

```java
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

```java
public class MyNode<T> implements NodeBinaryI<T>{

  public T data;
  public MyNode<T> left;
  public MyNode<T> right;

  public MyNode() {
    this(null);
  }

  public MyNode(T data) {
    this.data = data;
    left = null;
    right = null;
  }

  ...

  @Override
  public String toString() {
    return "[MyNode: data=" + data + "]";
  }

}
```

Implementation

Navigation in Trees

Navigation

```java
public class MyBinaryTree<T> {

  private MyNode<T> root;
  ...
  public String traverseInOrder() {
    return LibTree.traverseInOrder(root);
  }

  public String traversePreOrder() {
    return LibTree.traversePreOrder(root);
  }

  public String traversePostOrder() {
    return LibTree.traversePostOrder(root);
  }
  ...
}
```

```java
public class LibTree<T> {
  public static <T> String traverseInOrder(
    NodeBinaryInterface<T> node
  ) {
    if (node == null) {
      return "";
    }
    String s = "";
    s += traverseInOrder(node.left());
    s += node.toString();
    s += traverseInOrder(node.right());
    return s;
  }
}
```

```java
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

```java
public class MyNode<T> implements NodeBinaryI<T>{

  public T data;
  public MyNode<T> left;
  public MyNode<T> right;

  @Override
  public NodeBinary left() {
    return left;
  }

  @Override
  public NodeBinary right() {
    return right;
  }

  @Override
  public T data() {
    return data;
  }

}
```

Navigation: InOrder

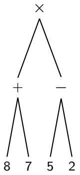
```java
public class MyBinaryTree<T> {

  private MyNode<T> root;
  ...
  public String traverseInOrder() {
    return LibTree.traverseInOrder(root);
  }

  public String traversePreOrder() {
    return LibTree.traversePreOrder(root);
  }

  public String traversePostOrder() {
    return LibTree.traversePostOrder(root);
  }
  ...
}
```

```java
public class LibTree<T> {
  public static <T> String traverseInOrder(
    NodeBinaryInterface<T> node
  ) {
    if (node == null) {
      return "";
    }
    String s = "";
    s += traverseInOrder(node.left());
    s += node.toString();
    s += traverseInOrder(node.right());
    return s;
  }
}
```




```java
public class MyBinaryTreeConstructor {
  public static MyBinaryTree<String>
      constructBT_S_Expression() {
    // ( 8 + 7 ) * ( 5 - 2 )
    MyBinaryTree<String> tree;
    tree = //
        new MyBinaryTree<>("*", //
            new MyBinaryTree<>("+", //
                new MyBinaryTree<>("8"), //
                new MyBinaryTree<>("7") //
            ), //
            new MyBinaryTree<>("-", //
                new MyBinaryTree<>("5"), //
                new MyBinaryTree<>("2") //
            ) //
        );
    // check
    tree.plot();
    System.out.println("\ncanonical:" + tree.
      canonical());
    System.out.println("\ntraverseInOrder:" + tree
      .traverseInOrder());
    return tree;
```

Navigation: InOrder

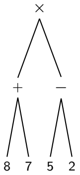
```
public class MyBinaryTree<T> {

  private MyNode<T> root;
  ...
  public String traverseInOrder() {
    return LibTree.traverseInOrder(root);
  }

  public String traversePreOrder() {
    return LibTree.traversePreOrder(root);
  }

  public String traversePostOrder() {
    return LibTree.traversePostOrder(root);
  }
  ...
}
```

```
public class LibTree<T> {
  public static <T> String traverseInOrder(
    NodeBinaryInterface<T> node
  ) {
    if (node == null) {
      return "";
    }
    String s = "";
    s += traverseInOrder(node.left());
    s += node.toString();
    s += traverseInOrder(node.right());
    return s;
  }
}
```




```
   /2
 /—
   \5
>*
   /7
 \+
   \8

canonical:/[8]\[+]/[7]\[*]/[5]\[—]/[2]\

traverseInOrder:
  [MyNode: data=8]
  [MyNode: data=+]
  [MyNode: data=7]
  [MyNode: data=*]
  [MyNode: data=5]
  [MyNode: data=—]
  [MyNode: data=2]
```

```
public class
  public sta
      constructBT_S_Expression() {
    // ( 8 + 7 ) * ( 5 — 2 )
    MyBinaryTree<String> tree;
    tree = //
        new MyBinaryTree<>("*", //
            new MyBinaryTree<>("+", //
                new MyBinaryTree<>("8"), //
                new MyBinaryTree<>("7") //
            ), //
            new MyBinaryTree<>("—", //
                new MyBinaryTree<>("5"), //
                new MyBinaryTree<>("2") //
            ) //
        );
    // check
    tree.plot();
    System.out.println("\ncanonical:" + tree.
      canonical());
    System.out.println("\ntraverseInOrder:" + tree
      .traverseInOrder());
    return tree;
```

Navigation: InOrder, PreOrder, PostOrder

```
public class MyBinaryTree<T> {

  private MyNode<T> root;
  ...
  public String traverseInOrder() {
    return LibTree.traverseInOrder(root);
  }

  public String traversePreOrder() {
    return LibTree.traversePreOrder(root);
  }

  public String traversePostOrder() {
    return LibTree.traversePostOrder(root);
  }
  ...
}
```

```
public class LibTree<T> {
  public static <T> String traverseInOrder(
    NodeBinaryInterface<T> node
  ) {
    if (node == null) {
      return "";
    }
    String s = "";
    s += traverseInOrder(node.left());
    s += node.toString();
    s += traverseInOrder(node.right());
    return s;
  }
}
```

```
public class LibTree<T> {
  ...
  public static <T> String traversePreOrder(
      NodeBinaryInterface<T> node) {
    if (node == null) {
      return "";
    }
    String s = "";
    s += node.toString();
    s += traversePreOrder(node.left());
    s += traversePreOrder(node.right());
    return s;
  }

  public static <T> String traversePostOrder(
      NodeBinaryInterface<T> node) {
    if (node == null) {
      return "";
    }
    String s = "";
    s += traversePostOrder(node.left());
    s += traversePostOrder(node.right());
    s += node.toString();
    return s;
  }
  ...
}
```

Implementation

Other Recursive Methods

Plot

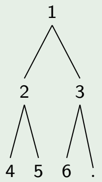
```java
public class MyBinaryTree<T> {
  private MyNode<T> root;
  ...
  public void plot() {
    LibTree.plot(root);
    System.out.println();
  }
}
```

```java
public class LibTree<T> {
  public static void plot(NodeBinaryI<?> node) {
    plot(node, 0, ">");
  }

  private static void plot(NodeBinaryI<?> node,
      int level, String leftRight) {
    // right subtree
    if (node.right() != null) {
      plot(node.right(), level + 1, "/");
    }
    // print the node
    String indent = " ".repeat(level);
    System.out.println(indent + leftRight + node
      .data());
    // left subtree
    if (node.left() != null) {
      plot(node.left(), level + 1, "\\");
    }
  }
}
```

```java
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

Example



Plot

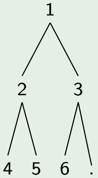
```java
public class MyBinaryTree<T> {
  private MyNode<T> root;
  ...
  public void plot() {
    LibTree.plot(root);
    System.out.println();
  }
}
```

```java
public class LibTree<T> {
  public static void plot(NodeBinaryI<?> node) {
    plot(node, 0, ">");
  }

  private static void plot(NodeBinaryI<?> node,
      int level, String leftRight) {
    // right subtree
    if (node.right() != null) {
      plot(node.right(), level + 1, "/");
    }
    // print the node
    String indent = " ".repeat(level);
    System.out.println(indent + leftRight + node
      .data());
    // left subtree
    if (node.left() != null) {
      plot(node.left(), level + 1, "\\");
    }
  }
}
```

```java
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

Example




```
 /3
  \6
>1
  /5
 \2
  \4
```

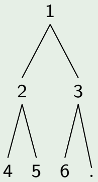
Height

```
public class MyBinaryTree<T> {
  private MyNode<T> root;
  ...
  public int height() {
    return LibTree.height(root);
  }
}
```

```
public class LibTree<T> {
  ...
  public static <T> int height(
      NodeBinaryInterface<T> node) {
    if (node == null) {
      return 0;
    } else if (node.left() == null//
       && node.right() == null) {
      return 0;
    } else {
      int h = 1 + Math.max(//
          height(node.left())//
          , height(node.right())//
      );
      return h;
    }
  }
  ...
}
```

```
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

Example



Call MyBinaryTree_Test.height_constructBT_S_1_6()

MyBinaryTree_Test.height_constructBT_S_1_6()

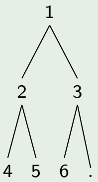
Height

```
public class MyBinaryTree<T> {
  private MyNode<T> root;
  ...
  public int height() {
    return LibTree.height(root);
  }
}
```

```
public class LibTree<T> {
  ...
  public static <T> int height(
      NodeBinaryInterface<T> node) {
    if (node == null) {
      return 0;
    } else if (node.left() == null//
       && node.right() == null) {
      return 0;
    } else {
      int h = 1 + Math.max(//
          height(node.left())//
          , height(node.right())//
      );
      return h;
    }
  }
  ...
}
```

```
public interface NodeBinaryI<T> {
  NodeBinary<T> left();
  NodeBinary<T> right();
  T data();
}
```

Example




```
—constructBT_S_1_6
 /3
  \6
>1
  /5
 \2
  \4

 node:1, height:2
 node:2, height:1
 node:3, height:1
 node:4, height:0
 node:5, height:0
 node:6, height:0
```

Call MyBinaryTree_Test.heigh

MyBinaryTree_Test.height_constructBT_S_1_6()

References I

[1] D. E. Knuth, *The Art of Computer Programming: Fundamental Algorithms, volume 1*, 2nd ed. Addison-Wesley Professional, 1973.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms (CLRS)*, 4th ed. MIT Press, 2022.

[3] B. Preiss, *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. Wiley, 1999.

[4] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[5] E. Horowitz and S. Sahni, *Fundamentals of Data Structures*. Pitman, 1982.

[6] C. S. Horstmann, *Big Java: Early Objects*, 7th ed. John Wiley & Sons, 2019.

[7] R. P. Grimaldi, *Discrete and Combinatorial Mathematics*, 5th ed. Pearson Education India, 2006.