

# Topic 5.1

## Tree

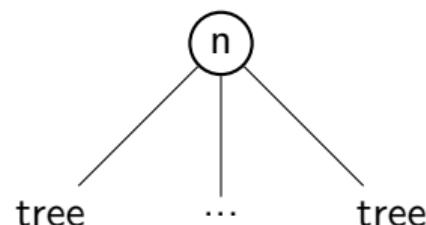
# Tree

## Definition 5.1

A **tree** is either a node



or the following structure consisting of a node and a set of children trees that are **disjoint**.



The above is **our first** recursive definition.

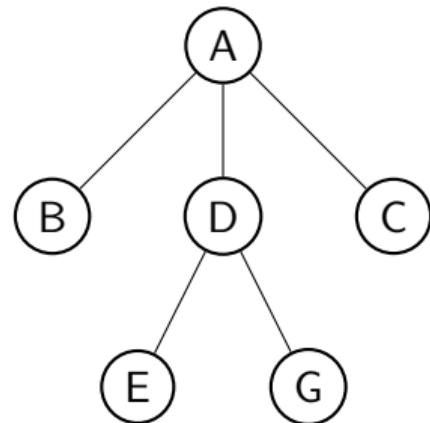
## Exercise 5.1

Does the above definition include **infinite** trees? How would you define an infinite tree?

## Example: tree

### Example 5.1

An instance of tree.



## Some tree terminology(2)

For nodes  $n_1$  and  $n_2$  in a tree  $T$ .

### Definition 5.2

$n_1$  is **child** of  $n_2$  if  $n_1$  is immediately below  $n_2$ . We write  $n_1 \in \text{children}(n_2)$ .

### Definition 5.3

We say  $n_2$  is **parent** of  $n_1$  if  $n_1 \in \text{children}(n_2)$  and write  $\text{parent}(n_1) = n_2$ .

If there is no such  $n_2$ , we write  $\text{parent}(n_1) = \text{Null}$ .

### Definition 5.4

$n_1$  is **ancestor** of  $n_2$  if  $n_1 \in \text{parent}^*(n_2)$ . We write  $n_1 \in \text{ancestors}(n_2)$ .

$n_2$  is **descendant** of  $n_1$  if  $n_1 \in \text{ancestor}(n_2)$ . We write  $n_2 \in \text{descendants}(n_1)$ .

**Commentary:** For a function  $f(x)$ , we define  $f^*(x) = \{y | y = f(\dots f(x))\}$ , i.e., the function is applied 0 or more times (informal definition). What would be a mathematically formal definition?

# Some tree terminology

## Definition 5.5

$n_1$  and  $n_2$  are **siblings** if  $\text{parent}(n_1) = \text{parent}(n_2)$ .

## Definition 5.6

$n_1$  is a **leaf** if  $\text{children}(n_1) = \emptyset$ .

$n_1$  is an **internal node** if  $\text{children}(n_1) \neq \emptyset$ .

## Definition 5.7

$n_1$  is a **root** if  $\text{parent}(n_1) = \text{Null}$ .

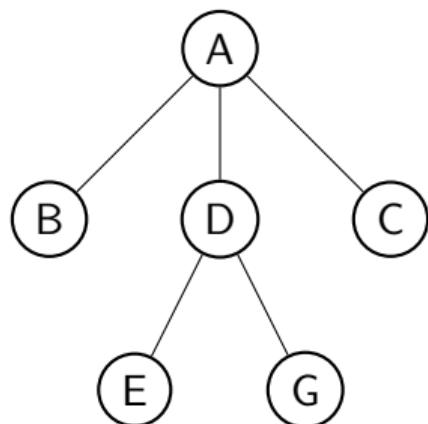
## Exercise 5.2

Can the root be an internal node? Can the root be a leaf?

## Example: Tree terminology

*B, D, and C are children of A.*

*D is the parent of G.*



*A is an ancestor of G and E is a descendant of A.*

*A is an ancestor of A.*

*G and E are siblings.*

*B, E, G, and C are leaves.*

*A and D are internal nodes.*

*A is the root.*

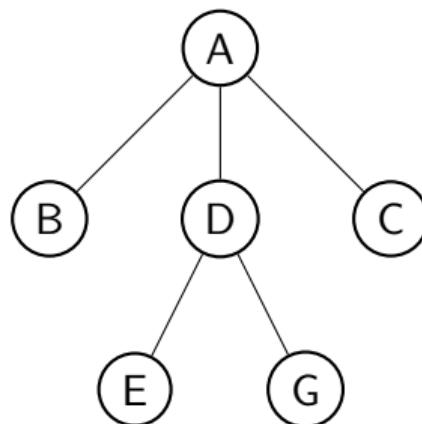
# Degree of nodes

## Definition 5.8

We define the degree of a node  $n$  as follows.

$$\text{degree}(n) = |\text{children}(n)|$$

## Example 5.3



$$\text{degree}(A) = 3$$

$$\text{degree}(B) = 0$$

$$\text{degree}(D) = 2$$

## Label of tree

Usually, we store data on the tree nodes.

We define the *label( $n$ )* of a node  $n$  as the data stored on the node.

## Level/Depth and height of nodes

### Definition 5.9

We define the level/depth of a node  $n$  as follows.

$$\text{level}(n) = \begin{cases} 0 & \text{if } n \text{ is a root} \\ \text{level}(n') + 1 & n' = \text{parent}(n) \end{cases}$$

### Definition 5.10

We define the height of a node  $n$  as follows.

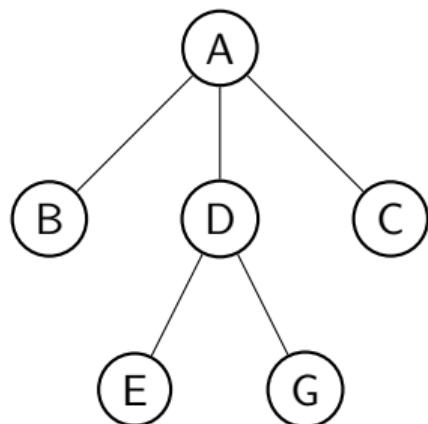
$$\text{height}(n) = \max(\{\text{height}(n') + 1 | n' \in \text{children}(n)\} \cup \{0\})$$

### Exercise 5.3

Why do we need to take a union with 0 in the definition of height?

## Example: Level(Depth) and height of nodes

Example 5.4



$$\text{level}(A) = 0$$

$$\text{level}(B) = 1$$

$$\text{level}(E) = 2$$

$$\text{height}(E) = 0$$

$$\text{height}(D) = 1$$

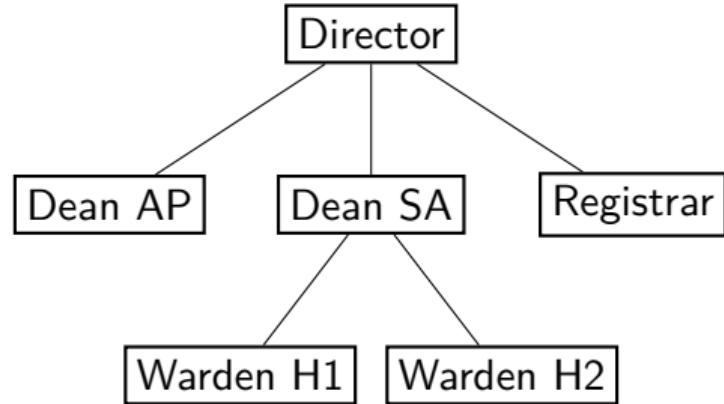
$$\begin{aligned}\text{height}(A) &= \max(\{\text{height}(B) + 1, \\ &\quad \text{height}(D) + 1, \\ &\quad \text{height}(C) + 1\} \cup \{0\}) \\ &= \max(\{1, 2, 1\} \cup \{0\}) = 2\end{aligned}$$

# Why do we need trees?

A tree represents a hierarchy.

## Example 5.5

- ▶ Organization structure of an organization

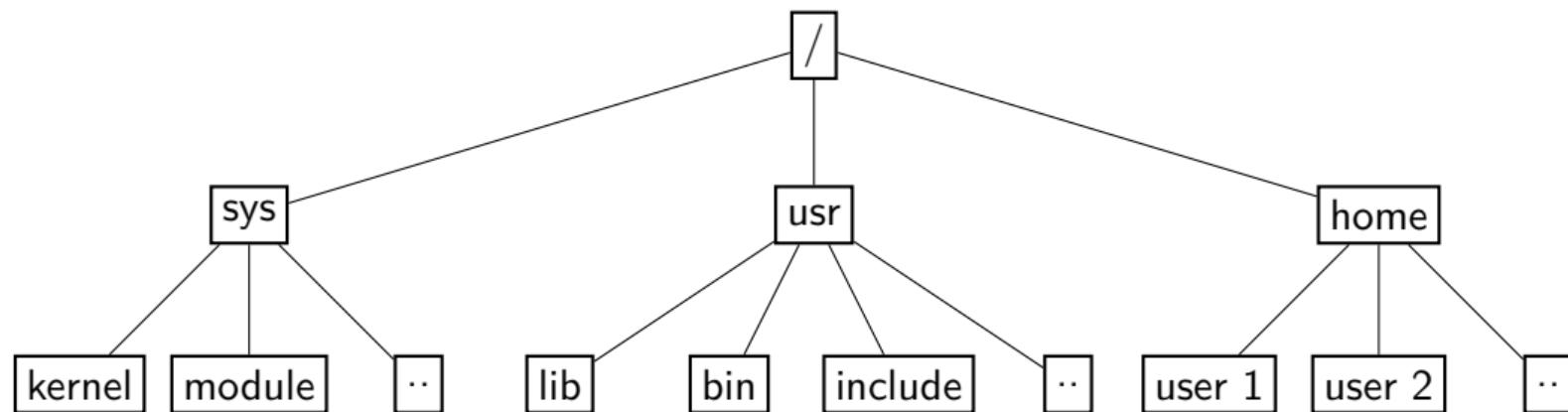


## Example: File system

Files are stored in trees in Linux/Windows.

### Example 5.6

Part of a Linux file system.



## Topic 5.2

### Binary tree

# Ordered tree

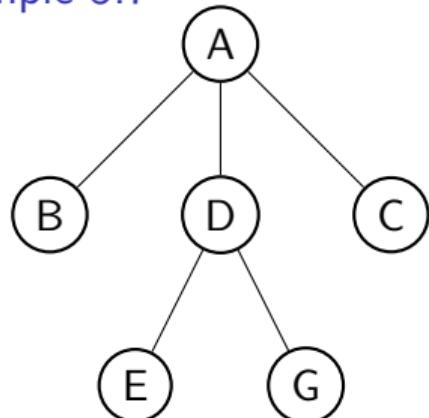
## Definition 5.11

A tree is an **ordered tree** if we assign an order among children.

## Definition 5.12

Let  $n$  be a node. In an ordered tree,  $\text{children}(n)$  is a **list instead of a set**.

## Example 5.7



In a tree, we define the children as follows.

$$\text{children}(A) = \{B, D, C\}$$

In an ordered tree, we define the children as follows.

$$\text{children}(A) = [B, D, C]$$

# Binary tree

## Definition 5.13

An ordered tree  $T$  is a **binary tree** if  $|children(n)| \leq 2$  for each  $n \in T$ .

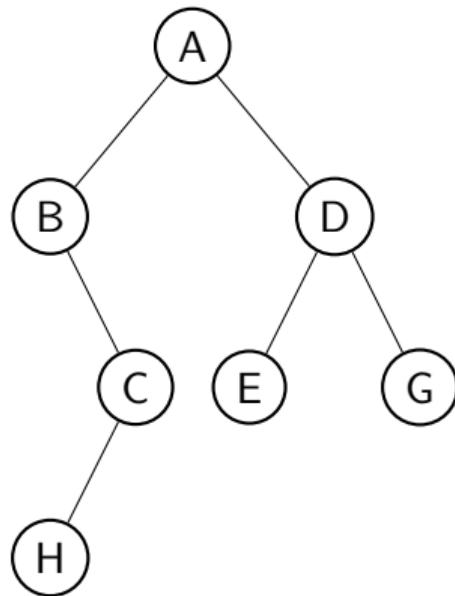
We define the left and right child of  $n$  as follows.

- ▶ if  $children(n) = [n_1, n_2]$ ,
  - ▶  $left(n) = n_1$  and  $right(n) = n_2$ .
- ▶ If  $children(n) = [n_1]$ ,  $n_1$  is either left or right child.
  - ▶  $left(n) = n_1$  and  $right(n) = Null$ , or
  - ▶  $left(n) = Null$  and  $right(n) = n_1$ .
- ▶ If  $children(n) = []$ ,
  - ▶  $left(n) = Null$  and  $right(n) = Null$ .

**Commentary:** For a mathematical nerd, the given definition of left/right child is not satisfactory. How can we interpret  $children(n) = [n_1]$  in two possible ways? In the next slide, we give a more formal definition.

## Example: binary tree

### Example 5.8



*E* is the left and *G* is the right child of *D*. *C* is the right child of *B*. *B* has no left child.

## Binary tree (alternative definition)

There is an alternative, more formal, way to define the binary tree. We may say that there are "Null" nodes, which are the leaves.

### Definition 5.14

An ordered tree  $T$  is a **binary tree** if  $|children(n)| \in \{0, 2\}$  for each  $n \in T$ .

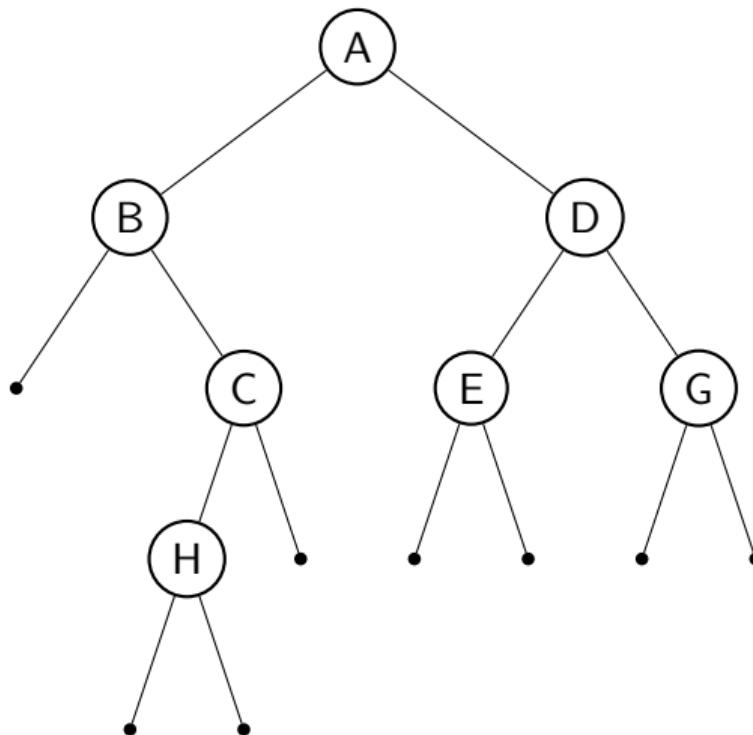
- ▶ If  $|children(n)| = 0$ , the  $n$  is a null node and has no label and we say  $n = Null$  (usually not drawn!).
- ▶ If  $|children(n)| = 2$ , We define the left and right child of  $n$  as follows.  
If  $children(n) = [n_1, n_2]$ ,  $left(n) = n_1$  and  $right(n) = n_2$ .

By the above definition, all internal nodes will have two children. We will write either  $children(n) = [Null, n_1]$  or  $children(n) = [n_1, Null]$  to express single child. Hence, we will have a clean definition of left and right child. For a leaf  $n$ , we will have  $children(n) = [Null, Null]$ .

**Commentary:** Please note that if we consider null-nodes then the height of the tree increases by one. There the subsequent discussion may cause confusion when we talk about the height of the binary trees.

## Example: binary tree with null nodes

Example 5.9

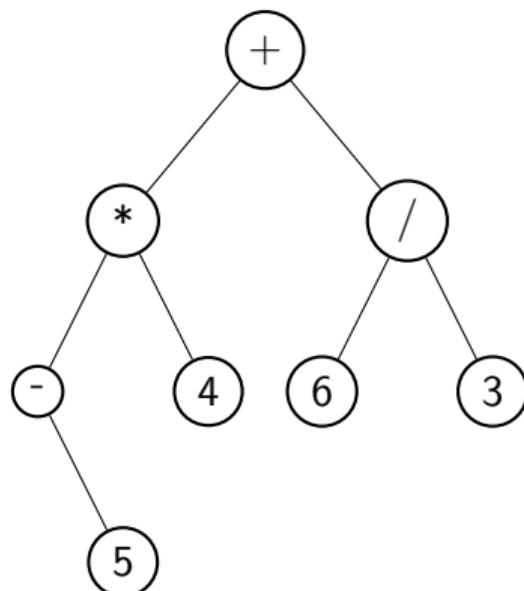


This issue will come up again in Red-Black tree. Meanwhile, we assume there are no null nodes.

## Usage of binary tree: representing expressions

### Example 5.10

Representing mathematical expressions



### Exercise 5.4

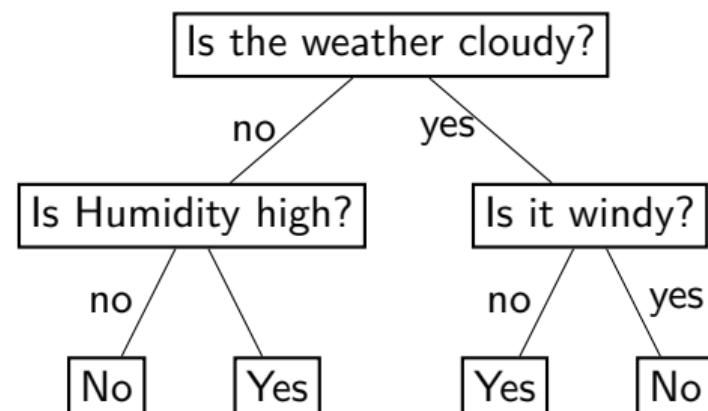
- Why do we need an ordered tree?
- How would you evaluate a mathematical expression given as a binary tree?

# Usage of binary tree: decision trees in AI

## Example 5.11

Do you want to play outside given the weather?

Given the behavior, we may learn the following tree.



# Complete binary tree (aka perfect binary tree)

**Commentary:** Some sources define the complete trees differently, where they allow last level to be not filled and all nodes are as left as possible. They also define full and balanced trees. Do an internet search.

## Definition 5.15

A binary tree is **complete** if the height of the root is  $h$  and every level  $i \leq h$  has  $2^i$  nodes.

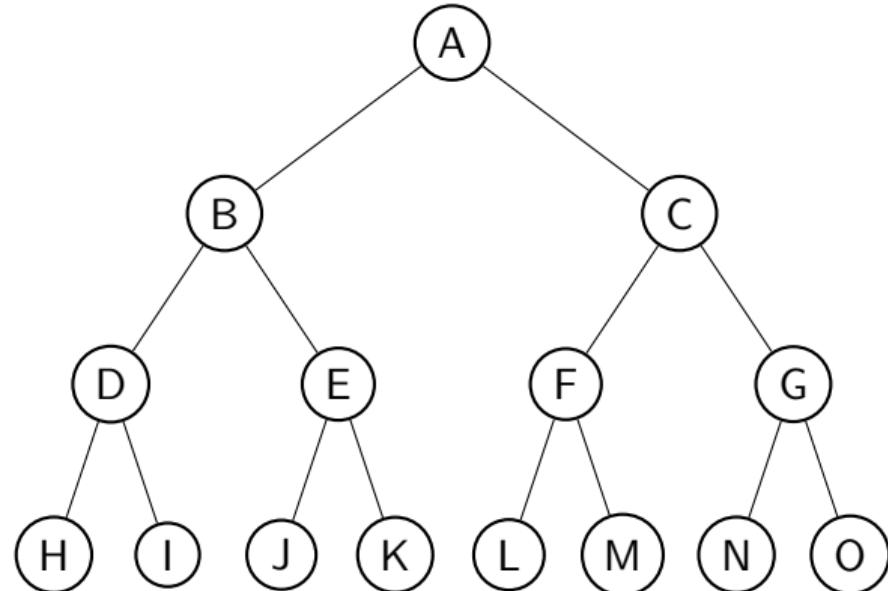
Leaves are only at level  $h$ .

The number of leaves =  $2^h$ .

$$\begin{aligned}\text{Number of internal nodes} &= 1 + 2 + \dots + 2^{h-1} \\ &= 2^h - 1.\end{aligned}$$

The total number of nodes is  $2^{h+1} - 1$ .

## Example 5.12



## Exercise 5.5

- Prove/Disprove: if no node in the binary tree has a single child, the binary tree is complete.
- What fraction of nodes are leaves in a complete binary tree?

# Maximum and minimum height of a binary tree

## Exercise 5.6

Let us suppose there are  $n$  nodes in a binary tree.

- ▶ What is the minimum height of the tree?
- ▶ What is the maximum height of the tree?

**Commentary:** For a given height  $h$ , a complete binary tree has  $2^{h+1} - 1$  nodes. All other binary trees with the height  $h$  have fewer nodes. Therefore,  $n \leq 2^{h+1} - 1$ .  
Therefore,  $\log_2 \frac{n+1}{2} \leq h$ . The maximum possible height for  $n$  nodes is  $n - 1$ . Therefore,  $\log_2 \frac{n+1}{2} \leq h \leq n - 1$ .

# Leaves of binary tree

## Theorem 5.1

For a binary tree,  $|\text{leaves}| \leq 1 + |\text{internal nodes}|$ .

## Proof.

We will prove the theorem by induction over the structure of a tree (Recall the recursive definition of a tree).

### Base case:



We have a single node.

$|\text{leaves}| = 1$  and  $|\text{internal nodes}| = 0$ . Case holds.

...

Commentary:  $|A|$  indicates the size of set A.

## Leaves of binary tree(2)

Proof(continued).

### Induction step:

We have two cases in the induction step: Root has one child or two children.

#### Case 1:

Let tree  $T$  be constructed as follows.



For  $T_1$ , let  $|\text{leaves}| = \ell_1$  and  $|\text{internal nodes}| = i_1$ .  
 $T$  has  $\ell_1$  leaves and  $i_1 + 1$  internal nodes.  
By the induction hypothesis,  $\ell_1 \leq 1 + i_1$ .  
Therefore,  $\ell_1 \leq 1 + i_1 + 1$ .  
Therefore,  $\ell_1 \leq 1 + (i_1 + 1)$ . Case holds.

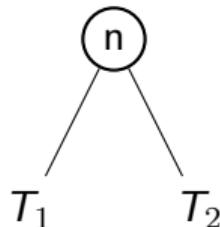
...

## Leaves of binary tree(3)

Proof(continued).

### Case 2:

Let tree  $T$  be constructed as follows.



For  $T_1$ , let  $|\text{leaves}| = \ell_1$  and  $|\text{internal nodes}| = i_1$ .

For  $T_2$ , let  $|\text{leaves}| = \ell_2$  and  $|\text{internal nodes}| = i_2$ .

$T$  has  $\ell_1 + \ell_2$  leaves and  $i_1 + i_2 + 1$  internal nodes.

By induction hypothesis,  $\ell_1 \leq 1 + i_1$  and  $\ell_2 \leq 1 + i_2$ .

Therefore, we have  $\ell_1 + \ell_2 \leq 2 + i_1 + i_2$ .

Therefore,  $\ell_1 + \ell_2 \leq 1 + (i_1 + i_2 + 1)$ . Case holds.



### Exercise 5.7

Prove/Disprove: If no node in the binary tree has a single child,  $|\text{leaves}| = 1 + |\text{internal nodes}|$ .  
(Quiz 2023)

## Maximum and minimum number of leaves

Let  $n$  be the number of nodes in a binary tree  $T$ .

Due to the previous theorem, we know  $|\text{leaves}| \leq 1 + |\text{internal nodes}|$ .

Since  $|\text{leaves}| + |\text{internal nodes}| = n$ ,  $|\text{leaves}| \leq 1 + n - |\text{leaves}|$ .

$$|\text{leaves}| \leq \frac{(n+1)}{2}.$$

### Exercise 5.8

- When do  $|\text{leaves}|$  meet the inequality?
- When is the number of leaves minimum?

**Commentary:** If  $T$  is complete, the number of leaves is  $\frac{(n+1)}{2}$ .

## Topic 5.3

### Representing Tree

## Container for tree

There is no C++ container for the tree.

Trees are the backbone of many abstract data structures.

For some reason, it is not explicitly there.

### Exercise 5.9

Why is there no tree container in C++ STL? (Let us ask ChatGPT)

**Commentary:** I guess that we rarely explicitly need trees in our programming. We usually have higher goals such as stack, queue, set, and map, which may need a tree as an internal data structure, but users need not be exposed. However, there are applications where there is a clear need for trees. For example, the representation of arithmetic expressions. In my programming, whenever I needed a tree, I have implemented it myself.

# Representation of a binary tree on a computer

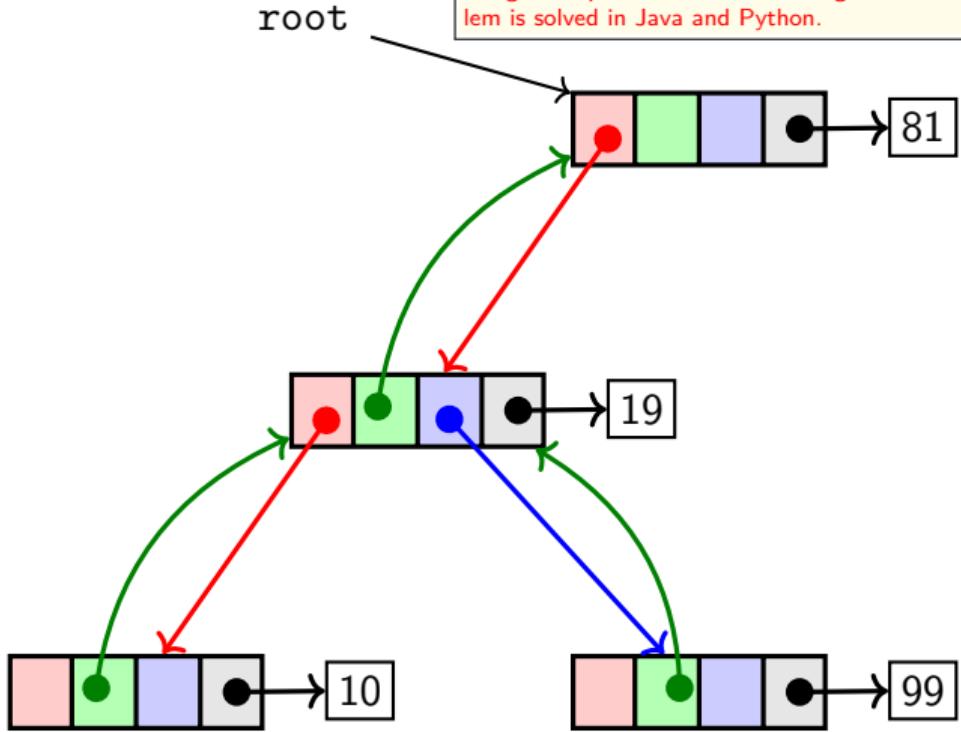
Commentary: parent pointer causes cycle of pointers, which makes the garbage collection difficult in the languages like Java. In C++, we may break the cycle by using weak pointers. Please investigate how the problem is solved in Java and Python.

## Definition 5.16

A binary tree consists of nodes containing four pointer fields.

- ▶ left child
- ▶ parent
- ▶ right child
- ▶ label

An additional root pointer points to the root of the tree.



The pointers that are not pointing anywhere are NULL.

## Exercise 5.10

Do we need the parent pointer?

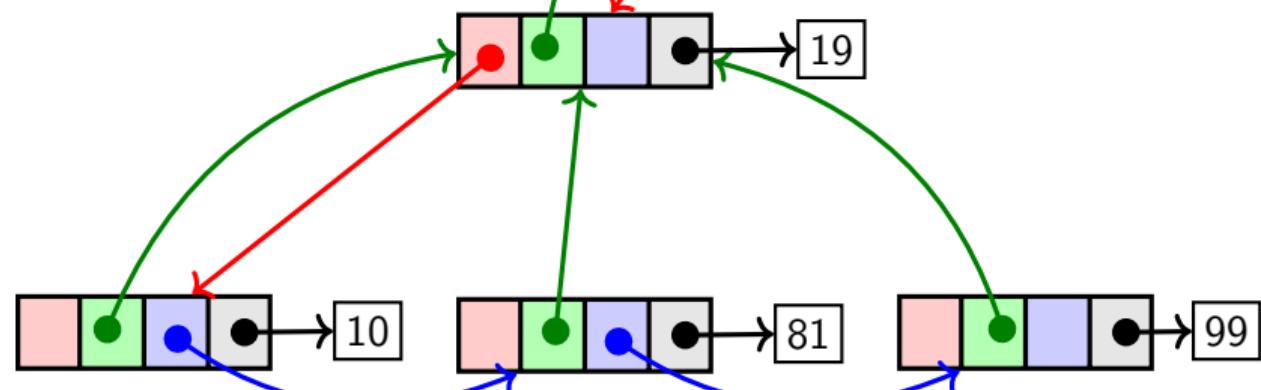
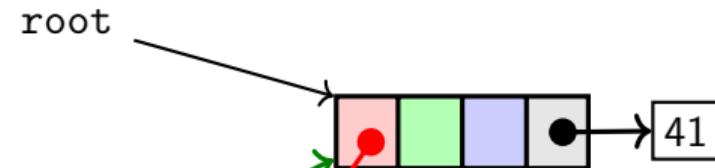
# Representation of a tree on a computer

## Definition 5.17

A tree consists of nodes containing four pointer fields.

- ▶ first child
- ▶ parent
- ▶ next sibling
- ▶ label

An additional root pointer points to the root of the tree.



## Exercise 5.11

Are we representing an ordered tree or an unordered tree?