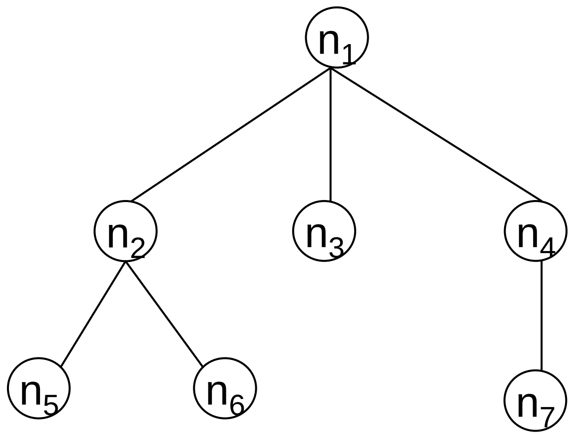
The Tree Data Model

Laura Kovács

Trees (Baumstrukturen) – Definition



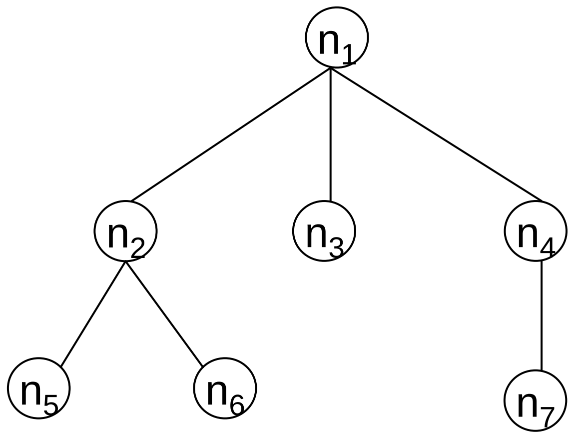
Trees are sets of

- points, called nodes (Knoten)

and

- lines, called edges (Kanten), connecting two distinct nodes,

Trees (Baumstrukturen) – Definition



Trees are sets of

- points, called nodes (Knoten)

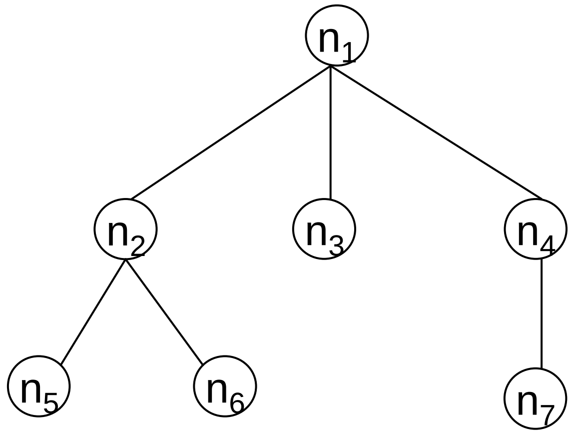
and

- lines, called edges (Kanten), connecting two distinct nodes,

such that:

- there is one special node, called the root (Wurzel); Ex: n_1

Trees (Baumstrukturen) – Definition



Trees are sets of

- points, called nodes (Knoten)

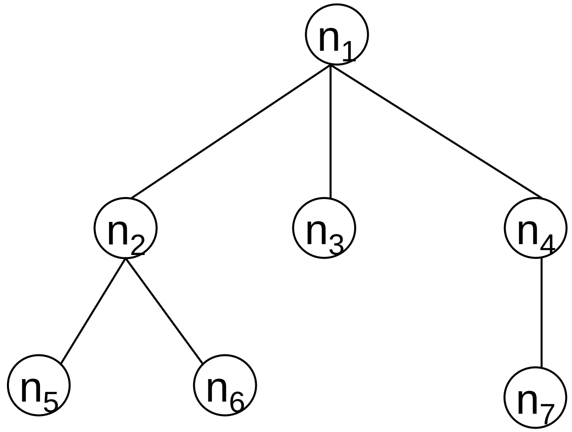
and

- lines, called edges (Kanten), connecting two distinct nodes,

such that:

- there is one special node, called the root (Wurzel); Ex: n_1
- every node c other than the root is connected by an edge to some other node p.
  - Node p is called the parent (Vater) of node c; Ex: n_2 is parent of n_5, n_6
  - Node c is called the child (Sohn) of node p; Ex: n_5, n_6 are children of n_2

Trees (Baumstrukturen) – Definition



Trees are sets of

- points, called nodes (Knoten)

and

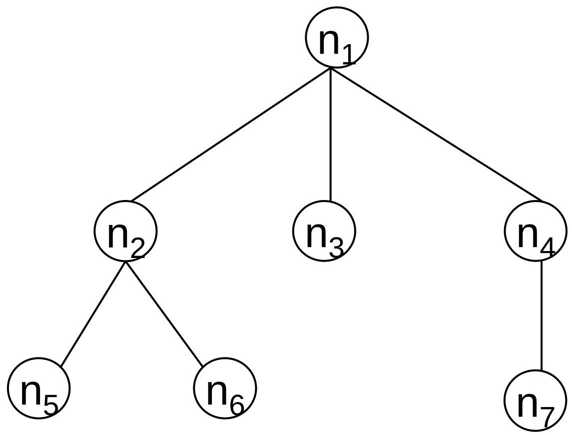
- lines, called edges (Kanten), connecting two distinct nodes,

such that:

- there is one special node, called the root (Wurzel); Ex: n_1
- every node c other than the root is connected by an edge to some other node p.
  - Node p is called the parent (Vater) of node c; Ex: n_2 is parent of n_5, n_6
  - Node c is called the child (Sohn) of node p; Ex: n_5, n_6 are children of n_2
- the tree is connected, that is:

  if we start at any node n different than the root $\rightarrow$ move to the parent of n $\rightarrow$ move to the parent of parent of n $\rightarrow$ ... $\rightarrow$ reach the root of the tree. Ex: $n_7 \rightarrow n_4 \rightarrow n_1$

Trees (Baumstrukturen) – Definition



Trees are sets of

- points, called nodes (Knoten)

and

- lines, called edges (Kanten), connecting two distinct nodes,

such that:

- there is one special node, called the root (Wurzel); Ex: n_1
- every node c other than the root is connected by an edge to some other node p.
  - Node p is called the parent (Vater) of node c; Ex: n_2 is parent of n_5, n_6
  - Node c is called the child (Sohn) of node p; Ex: n_5, n_6 are children of n_2
- the tree is connected, that is:

  if we start at any node n different than the root $\rightarrow$ move to the parent of n $\rightarrow$ move to the parent of parent of n $\rightarrow$ ... $\rightarrow$ reach the root of the tree. Ex: $n_7 \rightarrow n_4 \rightarrow n_1$

A node with no children is called a leaf (Blatt). Ex: n_5, n_6, n_7 are leaves.

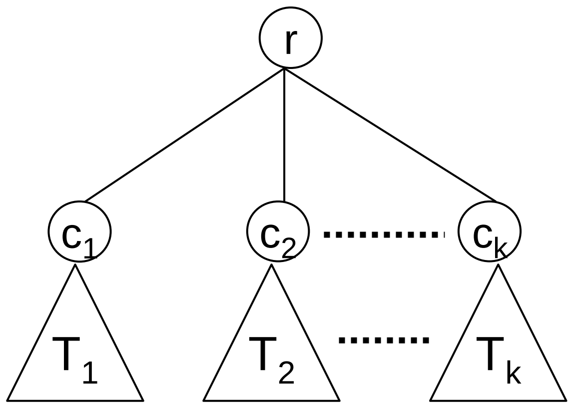
Trees (Baumstrukturen) – Alternative Definition

- A single node n is a tree.
  n is said to be the root of this tree.

Trees (Baumstrukturen) – Alternative Definition

- A single node n is a tree.
  n is said to be the root of this tree.

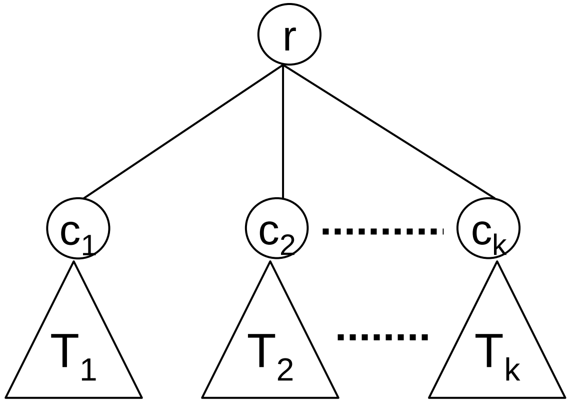
- Let r be a new node, and T_1, …, T_k trees with roots $c_1,\ldots,c_k$.
  Then a new tree T can be formed by
  - make r the root of T;
  - add an edge from r to each c_1, …, c_k.



Trees (Baumstrukturen) – Alternative Definition

- A single node n is a tree.
  n is said to be the root of this tree.

- Let r be a new node, and T_1, …, T_k trees with roots c_1,…,c_k.
  Then a new tree T can be formed by
  - make r the root of T;
  - add an edge from r to each c_1, …, c_k.

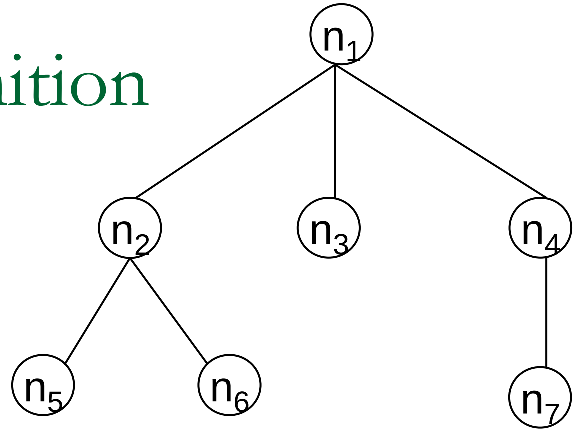


Trees T_1, …, T_k are subtrees (Teilbäume) of r.

Note: T_i contains c_i; the root of T_i is c_i.

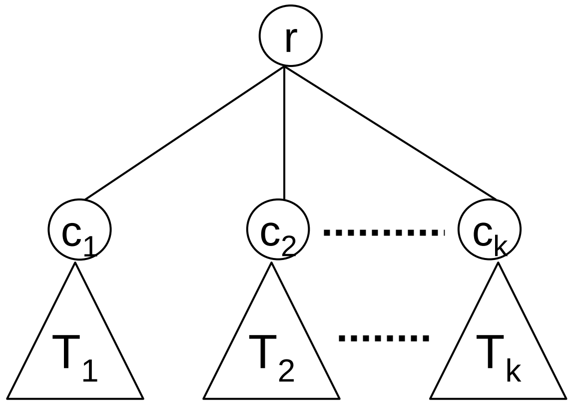
Note: A subtree with root c contains all the children of c, the children of children of c, etc.

Trees (Baumstrukturen) – Alternative Definition

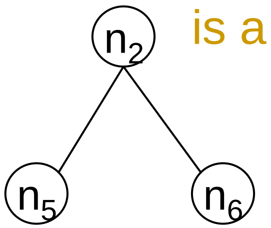


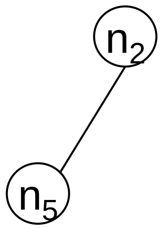
- A single node n is a tree.
  n is said to be the root of this tree.

- Let r be a new node, and T_1, …, T_k trees with roots c_1,…,c_k.
  Then a new tree T can be formed by
  - make r the root of T;
  - add an edge from r to each c_1, …, c_k.



Trees T_1, …, T_k are subtrees (Teilbäume) of r.

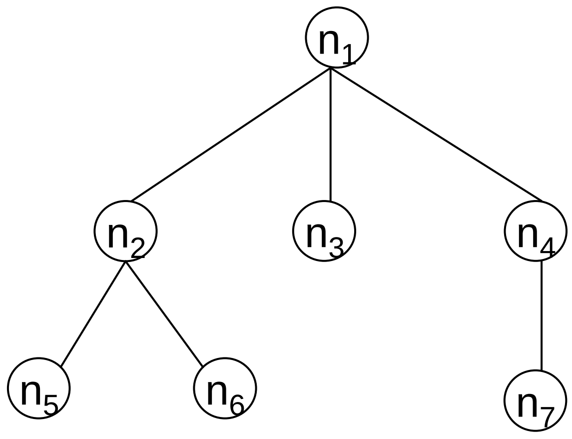
Ex: 

is a subtree. 

is not a subtree.

Note: A subtree with root c contains all the children of c, the children of children of c, etc.

Trees – Path



A path (Pfad) in a tree is a sequence of nodes

$$m_1,m_2,m_3,\ldots,m_k$$

such that:

- m_2 is the parent of m_1,
- m_3 is the parent of m_2,

⋮

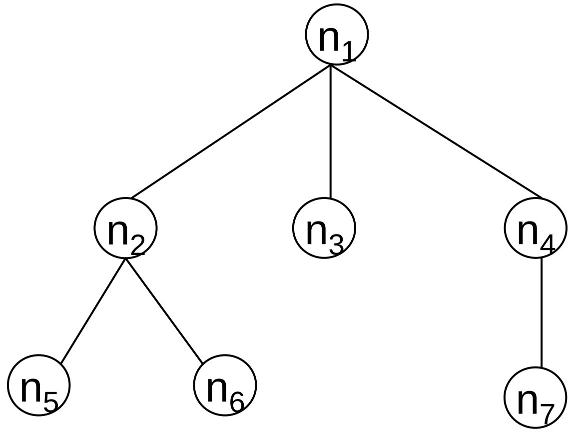
- m_{k-1} is the parent of m_k.

Ex: n_1, n_2, n_6 is a path

Note: (m_1,m_2), (m_2,m_3),…,(m_{k-1},m_k) are edges of the tree.

Between arbitrary two nodes there is <u>exactly one</u> path.

Trees – Path



A path (Pfad) in a tree is a sequence of nodes

$$m_1, m_2, m_3, \ldots, m_k$$

such that:

- m_2 is the parent of m_1,
- m_3 is the parent of m_2,

⋮

- m_{k-1} is the parent of m_k.

Ex: n_1, n_2, n_6 is a path of length 2.

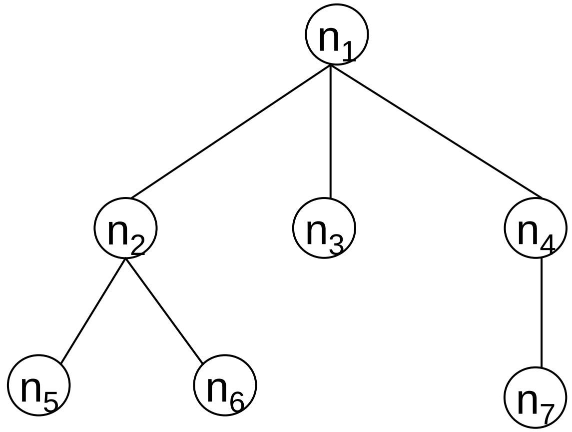
Ex: n_1 is a path of length 0.

Note: (m_1,m_2), (m_2,m_3),…,(m_{k-1},m_k) are edges of the tree.

Between arbitrary two nodes there is exactly one path.

The length (Länge) of the path is k-1.

Trees – Path



A path (Pfad) in a tree is a sequence of nodes

$$m_1, m_2, m_3, \ldots, m_k$$

such that:

- m_2 is the parent of m_1,
- m_3 is the parent of m_2,

⋮

- m_{k-1} is the parent of m_k.

Ex: n_1, n_2, n_6 is a path of length 2.

Ex: n_1 is a path of length 0.

Note: (m_1,m_2), (m_2,m_3),…,(m_{k-1},m_k) are edges of the tree.

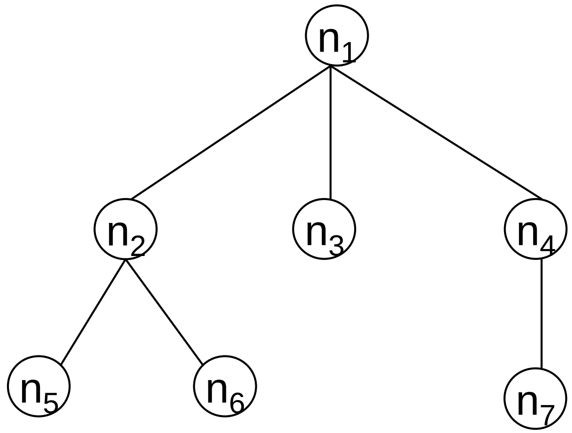
Between arbitrary two nodes there is exactly one path.

The length (Länge) of the path is k-1.

m_1 is called an ancestor (Vorgänger) of m_k; m_k is a descendant (Nachfolger) of m_1.

Ex: n_1 is an ancestor of n_2, n_6; n_6, n_2 are descendants of n_1.

Trees – Height, Depth, Degree



- The height (Höhe) of node m is the length of the longest path from m to a leaf.

  Ex: Height of n_1 is 2, height of n_2 is 1, leaf n_5 has height 0.

- The height of a tree is the height of the root.

  Ex: Height of the tree is 2.

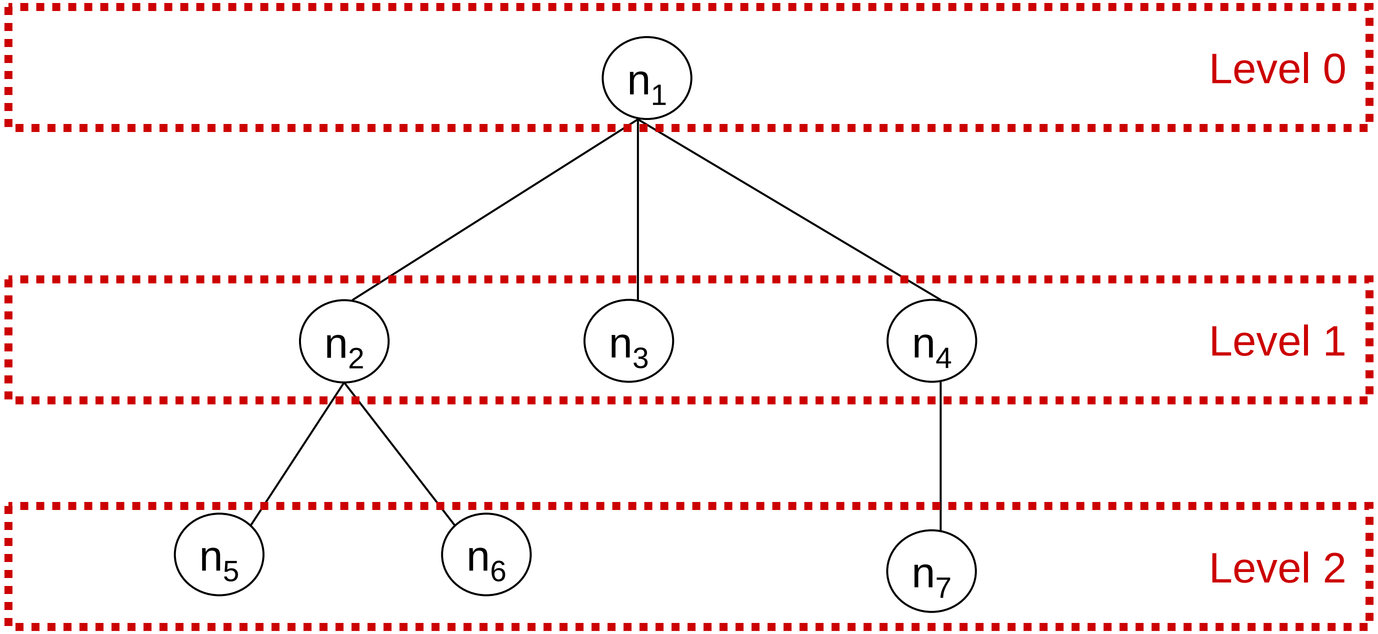
- The depth/level (level) of node m is the length of the path from the root to m.

  Ex: Depth of n_1 is 0, depth of n_2 is 1, leaf n_5 has depth 2.

- The degree (Ordnung) of a tree is the maximum of the number of subtrees of nodes.

  Ex: Degree of the tree is 3.

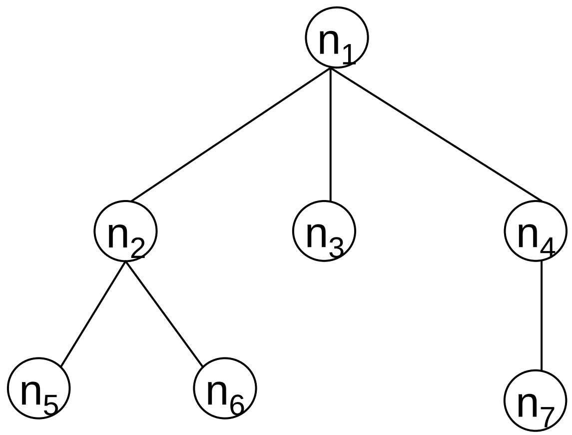
Trees – Height, Depth, Degree



Height of the tree is 2.

Degree of the tree is 3.

Trees – Ordered Trees (geordnete Baum)



An ordered tree (geordneten Baum) is a tree where an order is assigned to the children of any node.

Example: Assign a *left-to-right order* to the children of any node. Then, among the children of n_1:

n_2 is the leftmost child of n_1, then n_3, then n_4.

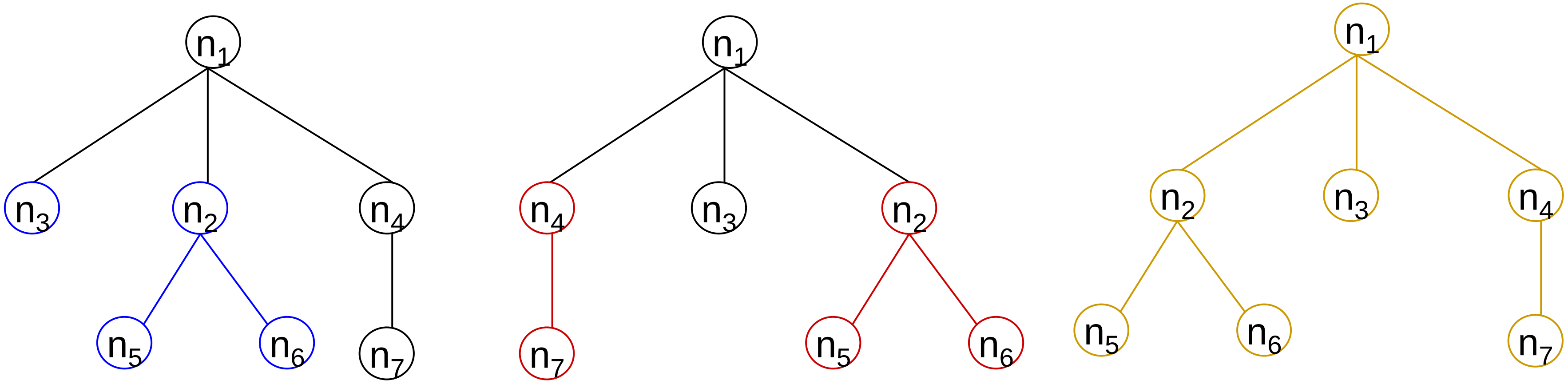
- n_4 is the rightmost child of n_1.
- n_3 is to the left of n_4.

In an ordered tree (geordneten Baum) the order of the subtrees is relevant.

Trees – Isomorphic Trees

Trees who differ only by the order of their subtrees are isomorphic.

Example of isomorphic trees:



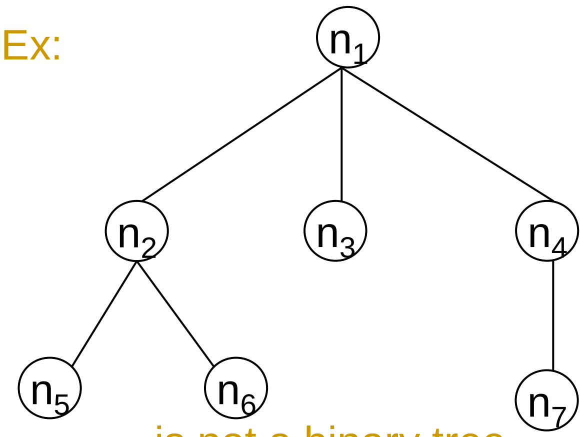
Trees – Binary Trees (binärer Baum)

- A binary tree is a tree such that each node has maximum two subtrees.

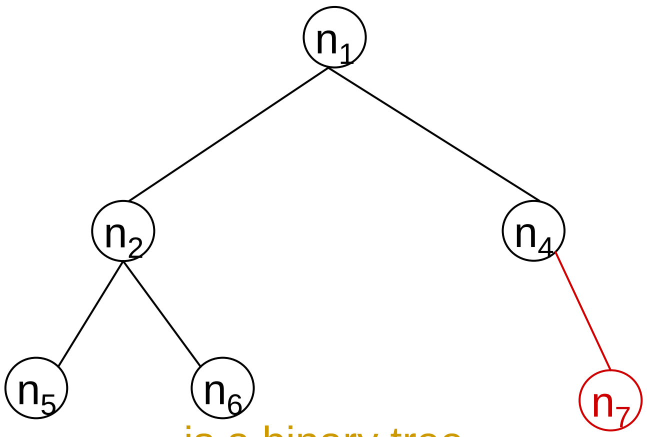
Special binary tree: empty tree (no nodes, no edges).

Note: The degree of a binary tree is maximum 2.

Ex:



is not a binary tree



is a binary tree

Binary trees have left (linken) and right (rechten) subtrees.

Difference between a Tree and a Binary Trees

BINARY TREE

- A binary tree may be empty.
- No node in a binary tree may have more than 2 subtrees.
- Degree of a binary tree is maximum 2.
- Subtrees of a binary tree are ordered.

TREE

- A tree cannot be empty.
- No limit on the number of subtrees of a node in a tree.
- No limit on the degree of a tree.
- Subtrees of a tree are not ordered.

Difference between a Tree and a Binary Trees

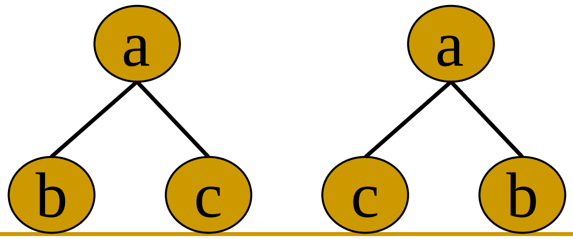
BINARY TREE

- A binary tree may be empty.
- No node in a binary tree may have more than 2 subtrees.
- Degree of a binary trees is maximum 2.
- The subtrees of a binary tree are ordered.

TREE

- A tree cannot be empty.
- No limit on the number of subtrees of a node in a tree.
- No limit on the degree of a tree.
- Subtrees of a tree are not ordered.

Ex:



- different when viewed as a binary tree

- same when viewed as a tree

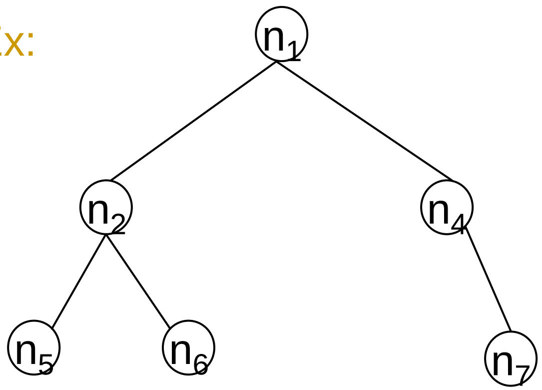
Full (Perfect/Complete) Binary Trees (perfekter/voll binärer Baum)

- A binary tree is full / complete / perfect when
  - the left subtree

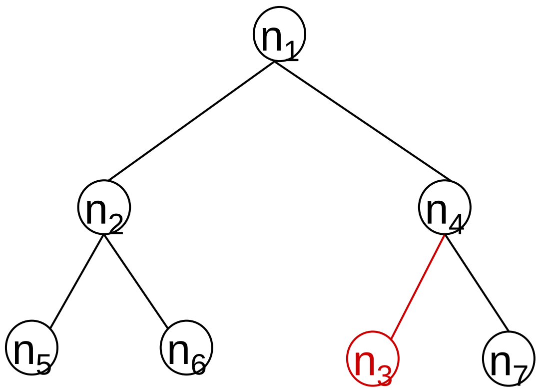
  and
  - the right subtree

  of each node contains the same number of nodes.

Ex:



is not a full binary tree



is a full binary tree

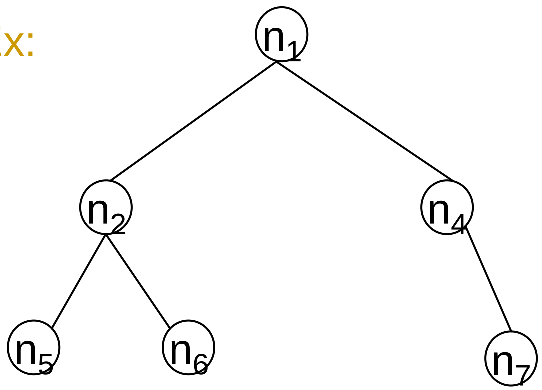
Full (Perfect/Complete) Binary Trees (perfekter/voll binärer Baum)

- A binary tree is full / complete / perfect when
  - the left subtree

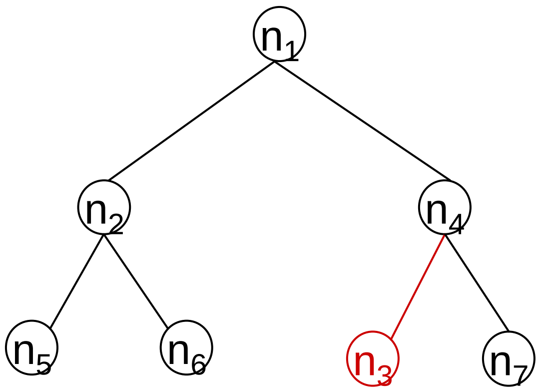
  and
  - the right subtree

  of each node contains the same number of nodes.

Ex:



is not a full binary tree



is a full binary tree

In a full binary tree each node

- is either a leaf;

- or has exactly two non-empty subtrees.

Full Binary Trees

- In a full binary tree with N nodes and height h:

$$N = 2^{h+1} - 1$$

and

$$h = ld(N+1) - 1$$

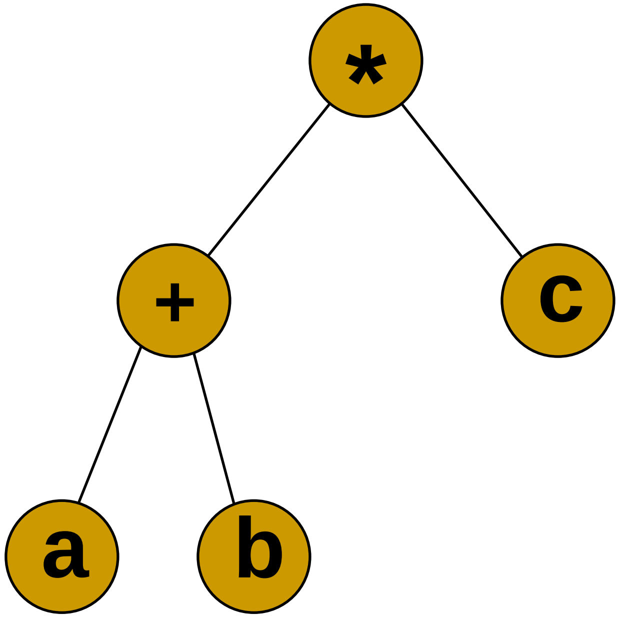
- A full binary tree with height h has exactly 2^h leaves.

Binary Trees - Syntax Trees (Syntaxbaum)

Syntax tree (expression tree) is a binary tree of an arithmetic expression.

- Nodes: arithmetic operators (+,-,*,…) and numbers/variables
  - Leafs: numbers/variables
- Edges:
  - parent-child relation between nodes is defined by the precedence of operators (indicated by parentheses).

Example: (a+b)*c



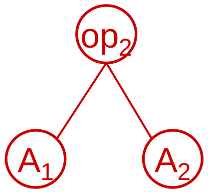
Binary Trees - Syntax Trees (Syntaxbaum)

1. The syntax tree of operand a is a *single-node tree* with root labeled by a.

2. If T_1 is the syntax tree of arithmetic expression A_1, and T_2 is the syntax tree of the arithmetic expression A_2, then:

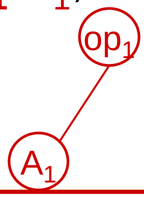
2.1. the expression tree of A_1 **op_2** A_2, where op_2 is a binary operator (+,*,-, …), is:

Binary operator = operator with 2 arguments



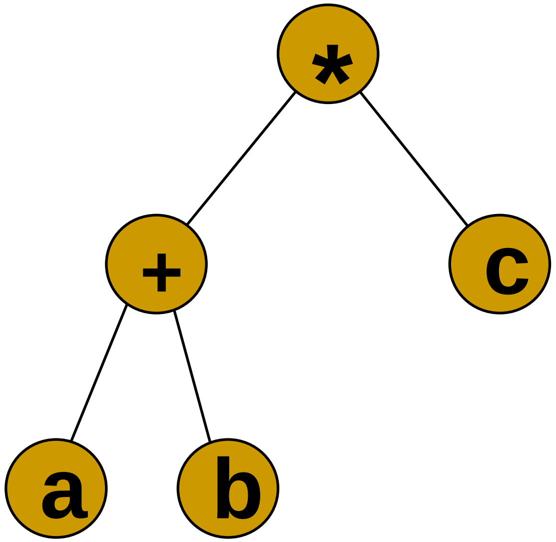
2.2. the expression tree of **op_1** A_1, where op_1 is a unary operator (!, Id, …), is:

Unary operator = operator with 1 arguments

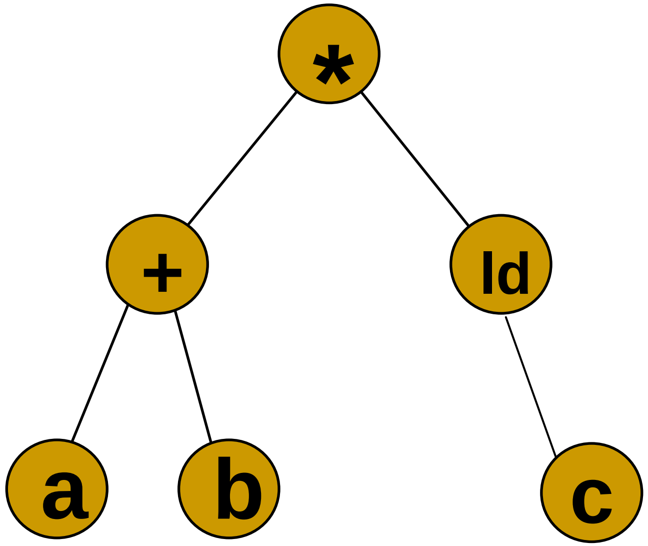


Binary Trees - Syntax Trees (Syntaxbaum)

Example: (a+b)*c



Example: (a+b)*ld(c)



Binary Trees – Traversal of Binary Trees

- Prefix traversal
- Infix traversal
- Postfix

Binary Trees – Prefix Traversal

PREFIX / PREORDER Traversal → *Prefix / Preorder notation* *(polish notation):*

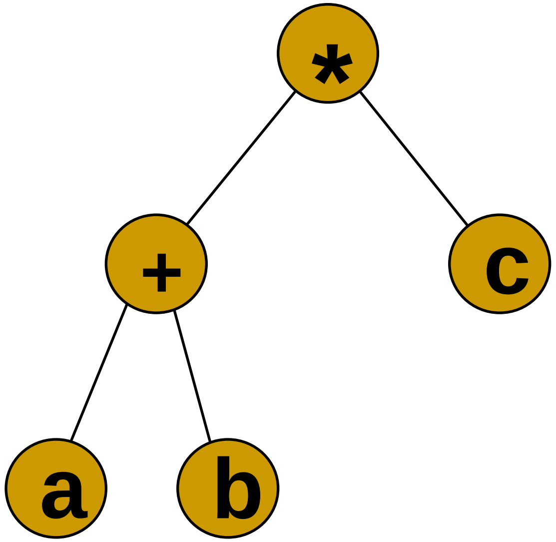
Recursively perform the following operations:

- Visit the node;
- Traverse left subtree;
- Traverse right subtree.

(Also called: depth-first traversal.)

```
preorder(root)
{
print root.value;
if NotEmpty(root.left) then preorder(root.left);
if NotEmpty(root.right) then preorder(root.right)
}
```

Example: (a+b)*c



Prefix/Preorder notation: *+abc

For a node n, let n.value denote its value, n.left its left subtree, n.right its right subtree.

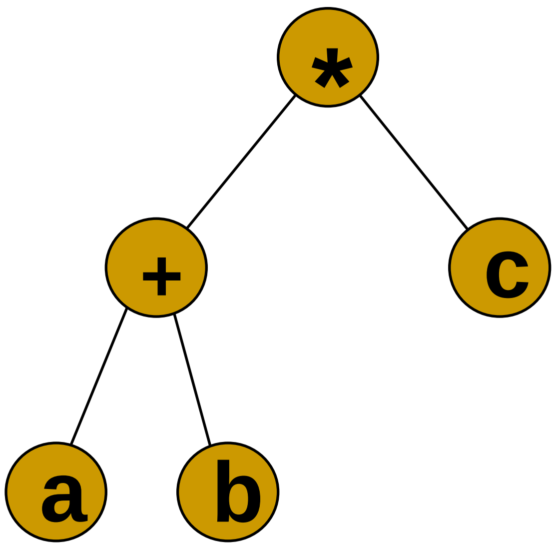
Binary Trees – Infix Traversal

INFIX / INORDER Traversal → *Infix / Inorder notation:*

Recursively perform the following operations:

- Traverse the left subtree;
- Visit the node;
- Traverse the right subtree.

Example: (a+b)*c




```
inorder(root)
{
if NotEmpty(root.left) then inorder(root.left);
print root.value;
if NotEmpty(root.right) then inorder(root.right)
}
```

Infix/Inorder notation: a+b*c

For a node n, let n.value denote its value, n.left its left subtree, n.right its right subtree.

Binary Trees – Postfix Traversal

POSTFIX / POSTORDER Traversal → *Postfix / Postorder notation* *(reverse polish notation):*

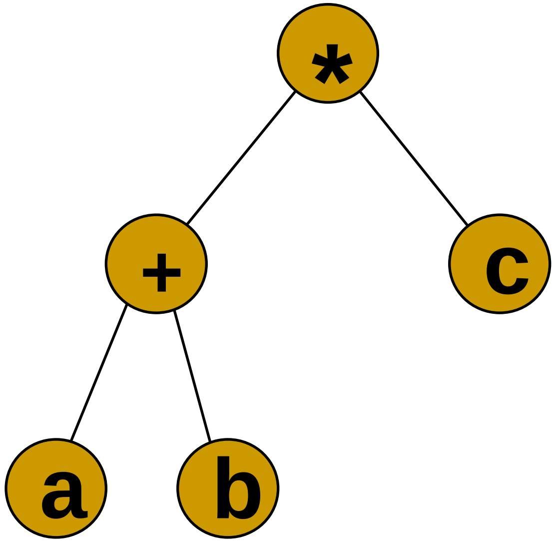
Recursively perform the following operations:

- Traverse the left subtree;
- Traverse the right subtree;
- Visit the node.

(Also called *breadth-first traversal*.)

```
postorder(root)
{
if NotEmpty(root.left) then postorder(root.left);
if NotEmpty(root.right) then postorder(root.right);
print root.value
}
```

Example: (a+b)*c



Postfix/Postorder notation: ab+c*

For a node n, let n.value denote its value, n.left its left subtree, n.right its right subtree.

Binary Trees – Binary Search (Sort) Tree (Sortierbaum)

A binary search tree is a binary tree with:

- The left subtree of a node n contains only nodes with values (keys) less than the value of n;
- The right subtree of a node n contains only nodes with values (keys) greater than the value of n;
- Both the left and right subtrees of n must be also binary search trees.

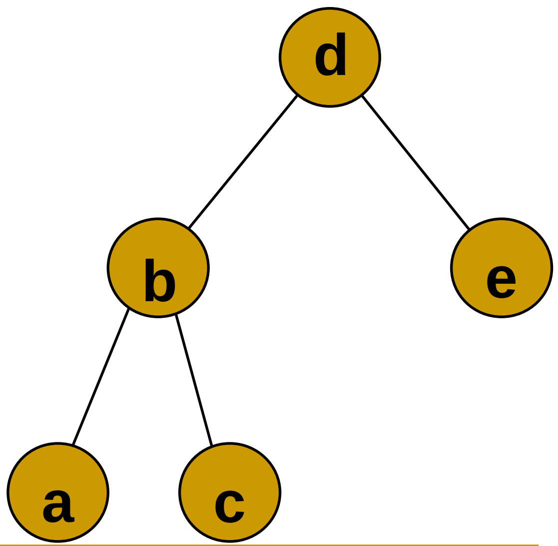
Note: Each node has a distinct value.

Inorder traversal of a binary search tress yields a sorted list of nodes.

Example:

- **Inorder: abcde ← SORTED LIST of NODES**
- **Preorder: dbace**
- **Postorder: acbed**
- ***Levelorder: d be ac***

(listing nodes from left-to-right, level-by-level starting from root)



Binary Trees – Binary Search (Sort) Tree (Sortierbaum)

A binary search tree is a binary tree with:

- The left subtree of a node n contains only nodes with values (keys) less than the value of n;
- The right subtree of a node n contains only nodes with values (keys) greater than the value of n;
- Both the left and right subtrees of n must be also binary search trees.

Note: Each node has a distinct value.

Inorder traversal of a binary search tress yields a sorted list of nodes.

Let T be a binary search tree. Let Nodes(T) denote the set of nodes of T. For a node n of T, let:

- n.left denote its left subtree;
- n.right denote its right subtree;
- n.value denote the value of n.

Then:

$$\forall n: n \in \text{Nodes}(T):$$
$$(\forall n_l: n_l \in \text{Nodes}(n.\text{left}): n_l.\text{value} < n.\text{value}) \wedge (\forall n_r: n_r \in \text{Nodes}(n.\text{right}): n_r.\text{value} > n.\text{value})$$

An alternative:

$\forall n: n \in \text{Nodes}(T): (\forall n_l: n_l \in \text{Nodes}(n.\text{left}): n_l.\text{value} \leq n.\text{value}) \wedge (\forall n_r: n_r \in \text{Nodes}(n.\text{right}): n_r.\text{value} > n.\text{key})$

Binary Trees - Exercises

- Consider the expression **a b + c d - * e f + /** in postfix form.
  - What is its infix form?
  - What is its prefix form?

- Consider the binary tree:
  - Is it a binary search tree?
  - Is it a full binary tree?
  - What is the degree of the tree?
  - What is the height of the tree?
  - What is its prefix form?