Chapter 3

Tree structures

Definition: A tree is a set of nodes organized from a distinguished node called the root.

Applications : - file organization : unix

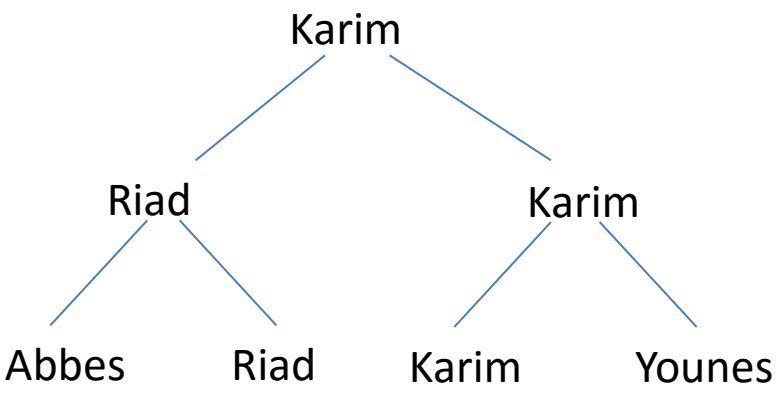
- Representation of pgmes in compilation

Intrinsic property: natural recursion.

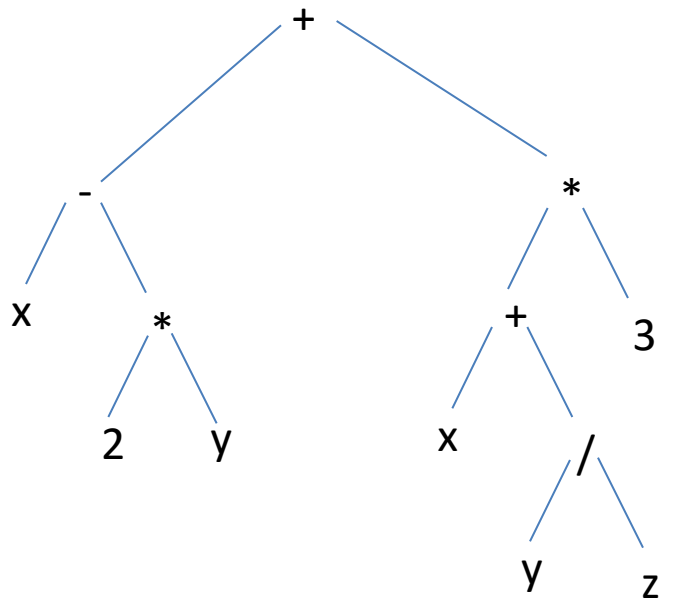
Binairy Trees

Examples :

1) Tennis Tournament



2) Arithmetic expressions

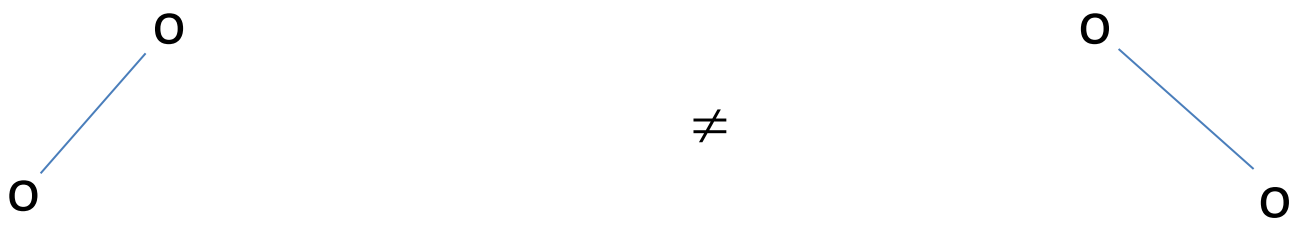


Definition : a binary tree is either empty (ϕ) or B=<o, B_1, B_2>;
where B1, B2 are disjoint binary trees, 'o' is a node called root.

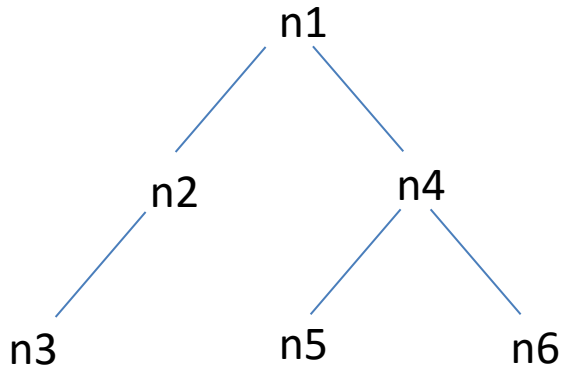
In other words : B=ϕ + <o, B, B>

No left-right symmetry :

<o, <o, ϕ, ϕ>, ϕ> $\neq$ <o, ϕ, <o, ϕ,ϕ>>



To identify the nodes, you need to associate them with different names, as in:



Labeled tree

ADT binary tree

Sort Tree

Uses Node, Element

Operations

empty tree	:		$\rightarrow$ Tree
< _ , _ , _ >	:	Node x Tree x Tree	$\rightarrow$ Tree
root	:	Tree	$\rightarrow$ Node
g	:	Tree	$\rightarrow$ Tree
d	:	Tree	$\rightarrow$ Tree
content	:	Node	$\rightarrow$ Element

Pre-conditions : where B_1, B_2 : Tree; o: Node

root (B_1) is-defin-ioi $B_1 \neq$ empty tree

g(B_1) is-defin-ioi $B_1 \neq$ empty tree

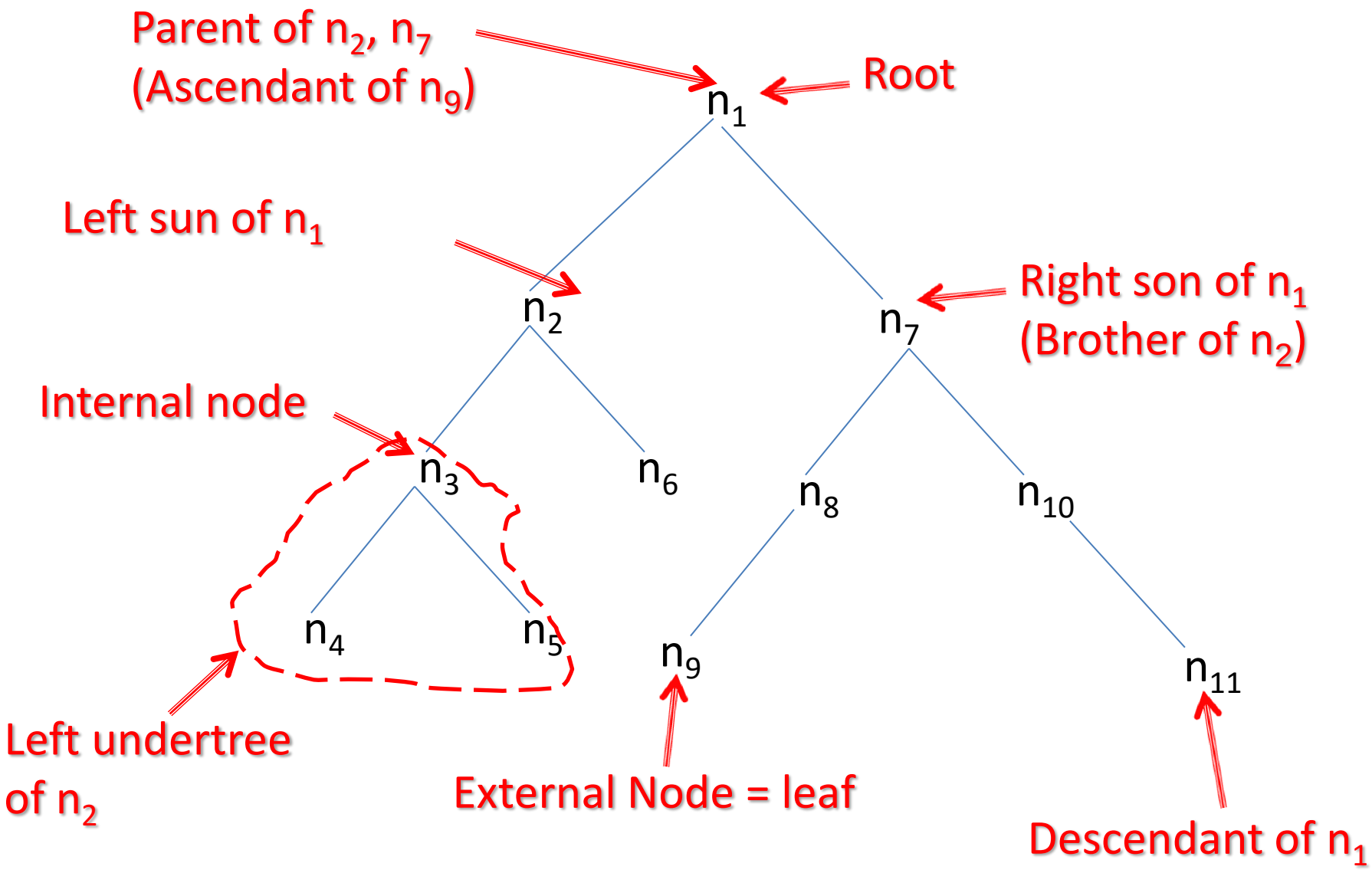
d(B_1) is-defin-ioi $B_1 \neq$ empty tree

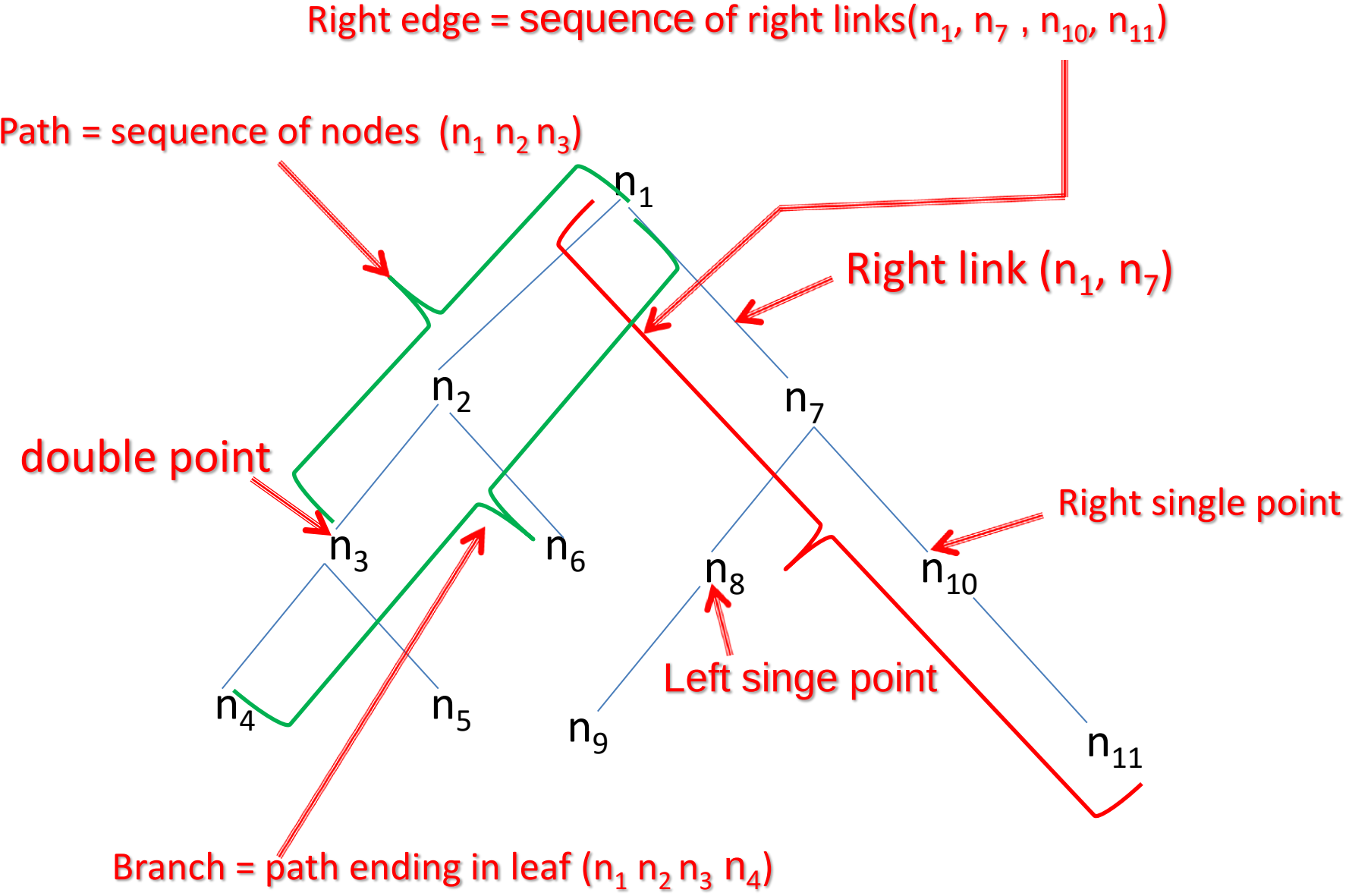
Axiomes : root (<o, B_1, B_2>)=o

g(<o, B_1, B_2>)=B_1

d(<o, B_1, B_2>)=B_2

Vocabulary


Parent of n_2, n_7
(Ascendant of n_9)
Root
n_1
Left sun of n_1
n_2
n_7
Right son of n_1
(Brother of n_2)
Internal node
n_3
n_6
n_8
n_{10}
n_4
n_5
n_9
n_{11}
Left undertree
of n_2
External Node = leaf
Descendant of n_1


Right edge = sequence of right links(n_1, n_7 , n_{10}, n_{11})
Path = sequence of nodes (n_1 n_2 n_3)
n_1
Right link (n_1, n_7)
n_2
n_7
double point
Right single point
n_3
n_6
n_8
n_{10}
Left singe point
n_4
n_5
n_9
n_{11}
Branch = path ending in leaf (n_1 n_2 n_3 n_4)

Measurements on trees

Size = number of nodes

Size (empty tree)=0

Size(<o, B_1, B_2 >)=1+ Size (B_1)+ Size (B_2)

Height=Depth=level of a node x of a tree B

h(x)=0 if x = root (B)

h(x)=1+h(y) if y is parent of x

This is the number of links from the root to x.

Height = Depth of a tree B

h(B)=max {h(x); x node of B}

External path length of a Tree B :

LCE(B)= $\sum$ h(f) where f is a leaf of B.

Internal path length of a tree B :

LCI(B)= $\sum$ h(x) where x is an internal node of B.

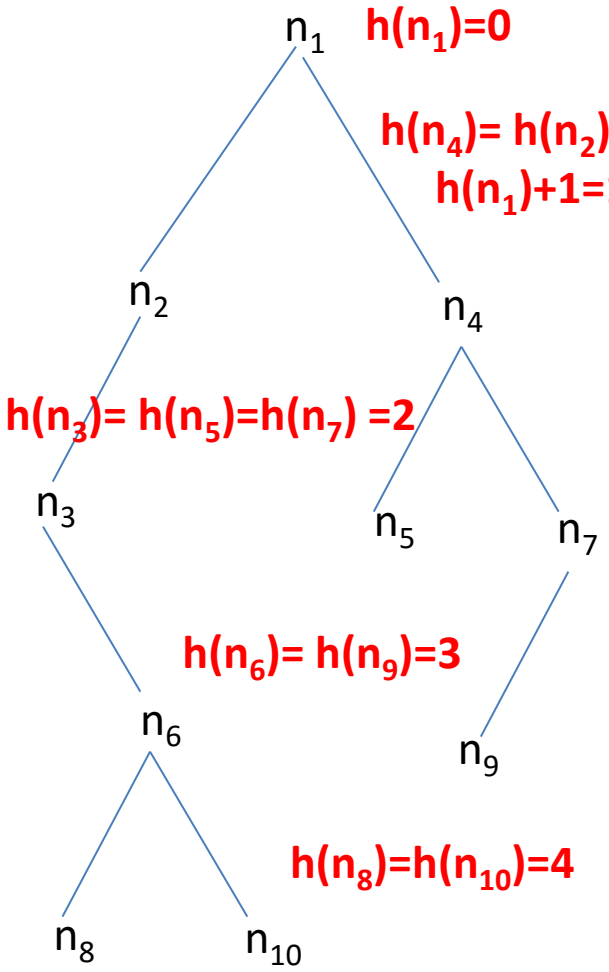
Longueur de cheminement d'un arbre B :

LC(B) = $\sum$ h(x) for all nodes x of B.

=LCE(B) + LCI(B)

Average external depth of a tree B having f leaves:

PE(B)=LCE(B) / f



Donc : h(Arbre B)=4

LCE(B) = h(n_5)+h(n_9)+h(n_{10})+ h(n_8)=13

LCI(B) = h(n_6)+h(n_3)+h(n_7)+ h(n_4)+h(n_2)=9

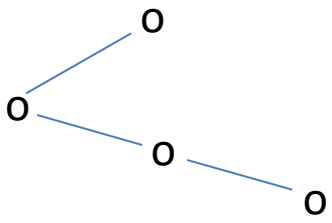
LC(B)=13+9=22

PE(B)=13/4 = 3.25

Special binary trees

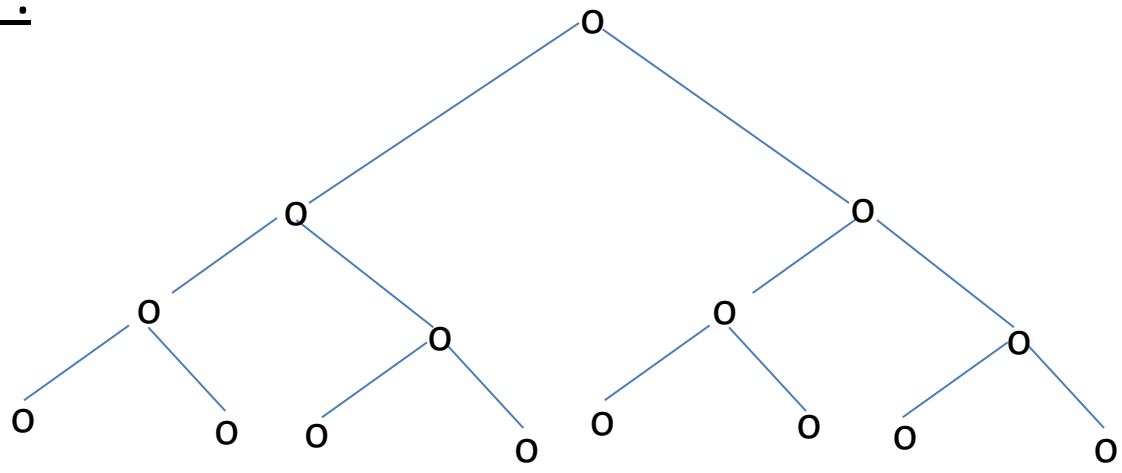
- A degenerate or filiform binary tree is formed only of simple points.

  Example:



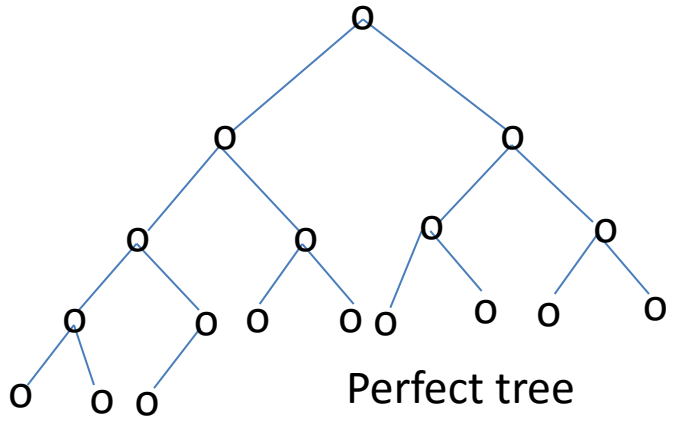
- A binary tree is complete if each level i contains 2^i nodes (i.e. completely filled) and the size of the tree = 2^h+1-1 where h=height of the tree.

  Example :

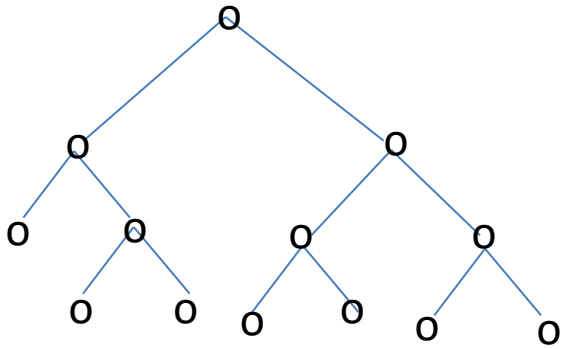


- A binary tree is said to be perfect if all levels are completely filled except, possibly, the last level whose nodes are grouped as far to the left as possible.

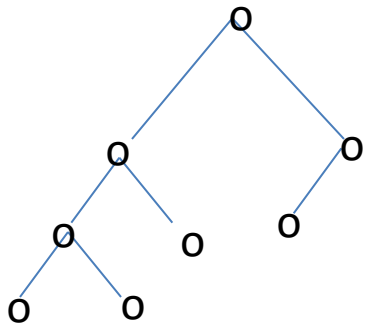
  Examples:



Perfect tree



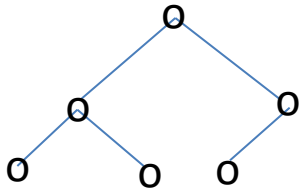
Imperfect tree



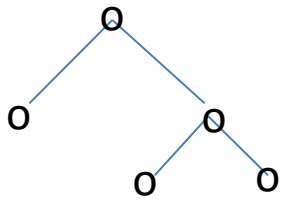
Imperfect tree

- A locally complete binary tree is a non-empty tree that has no single points.

  Examples:



Not locally complete



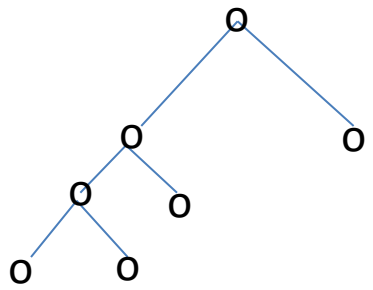
locally complete

Recursively, we define a locally complete tree by :

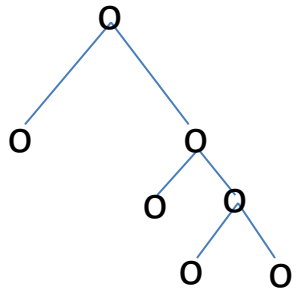
BC = <o, ϕ, ϕ> + <o, BC, BC>

- A left (right) comb is a locally complete binary tree in which every right (left) child is a leaf.

  PG=<o, ϕ, ϕ> + <o, PG, <o, ϕ, ϕ>>



Left comb



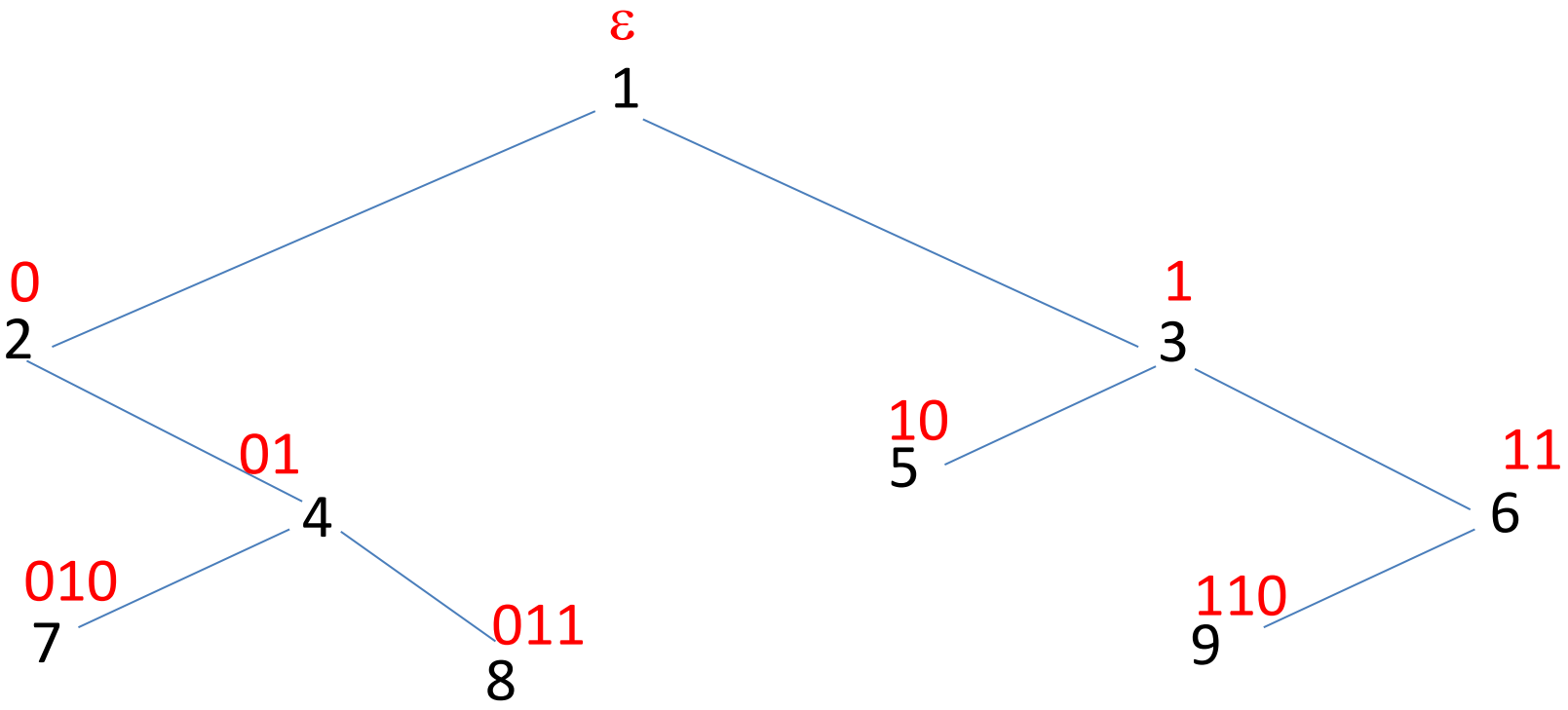
Right comb

Occurrence :

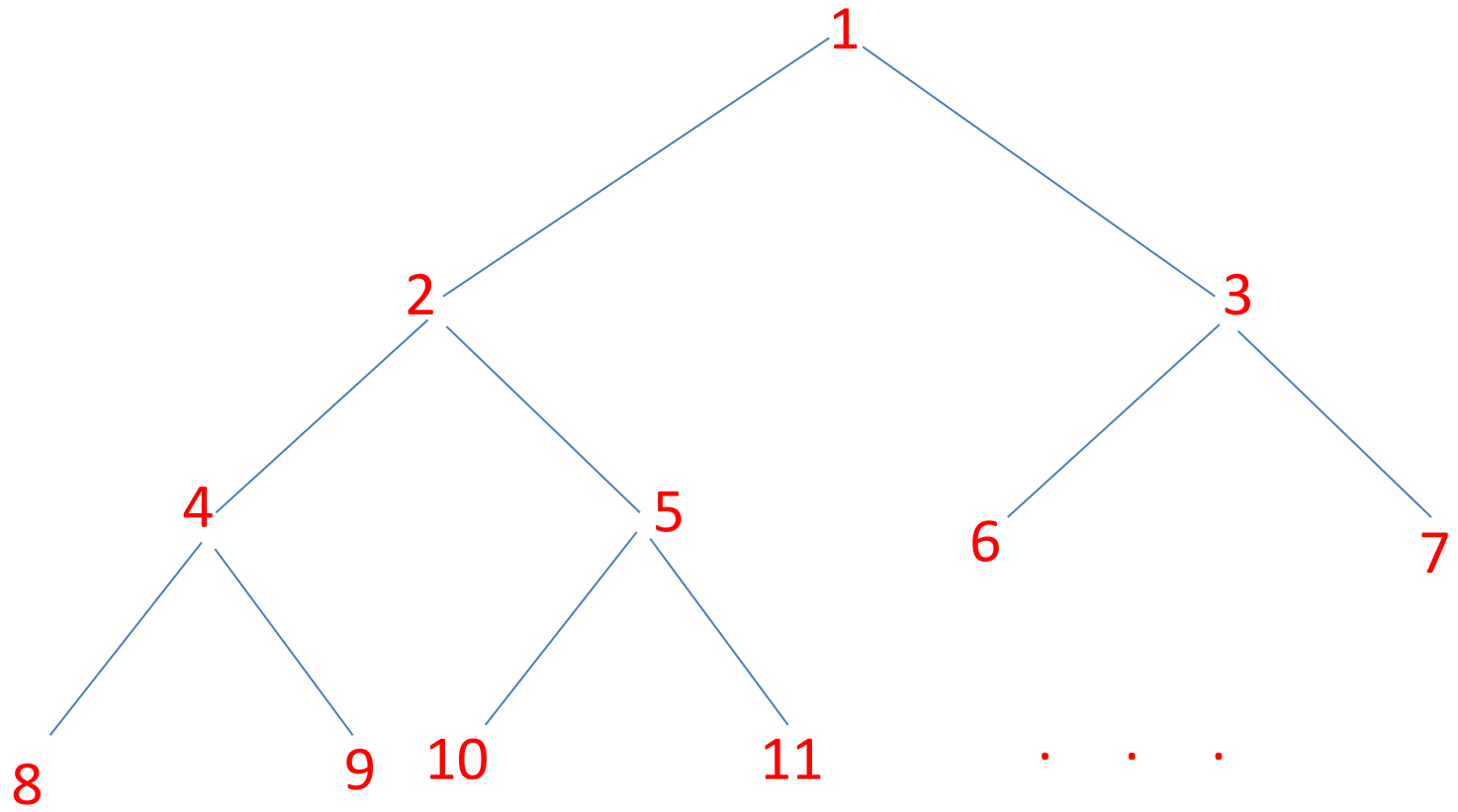
The path from the root to a node is described by a word on $\{0,1\}^*$, called an occurrence such that :

- The occurrence of the root = empty word = ε
- If a node has occurrence μ, then its left child has occurrence $\mu\mathbf{0}$, and its right son has the occurrence $\mu\mathbf{1}$.

Example : Let the following tree B :



Numbering in hierarchical order : Level by level from left to right.



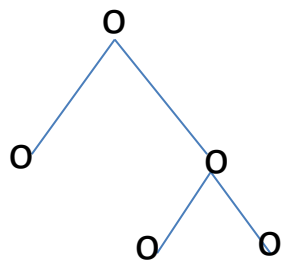
For a node with order i and occurrence μ we have :

$i = 2^{[\log_2 i]} + m$ where m is the integer represented by μ

$(m=(\mu)_{10})$ and [k] = Integer part of k.

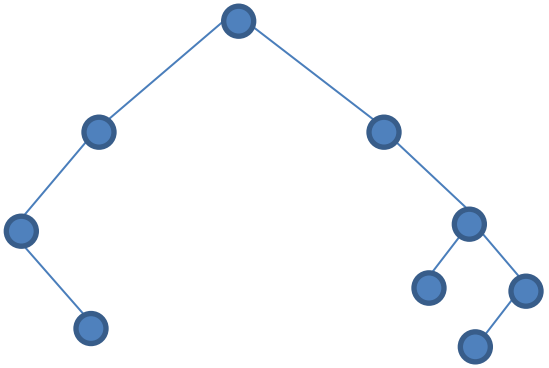
Fundamental properties :

a) A locally complete binary tree B with n internal nodes has (n+1) leaves and we have : LCE (B) =LCI(B) + 2n



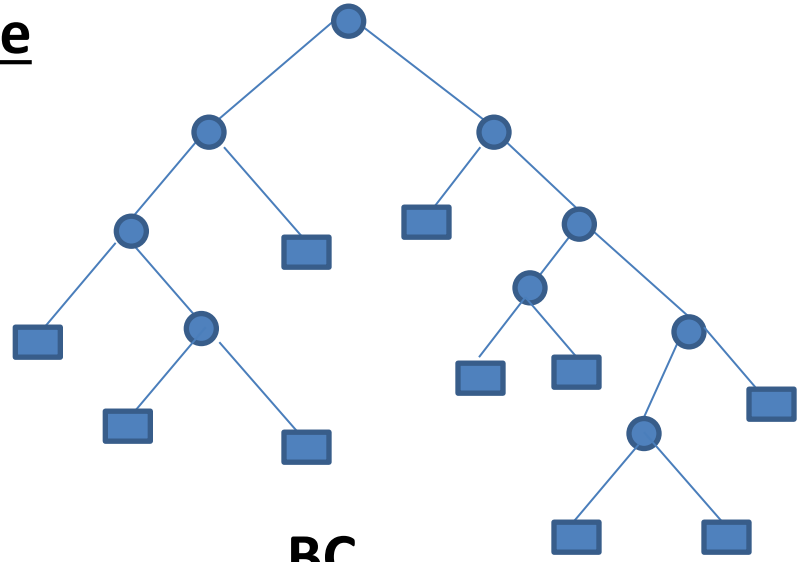
2 internal nodes, 3 leaves,
LCE=5, LCI=1

b) **Local completion of a binary B tree**



B

⇒



BC

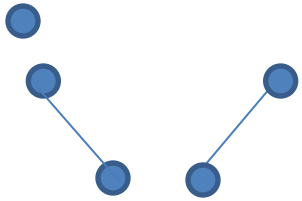
each node of B has two children in BC

c) Bijection between B_n the set of binary trees having n nodes, and the set BC_n of locally complete binary trees having 2n+1 nodes.

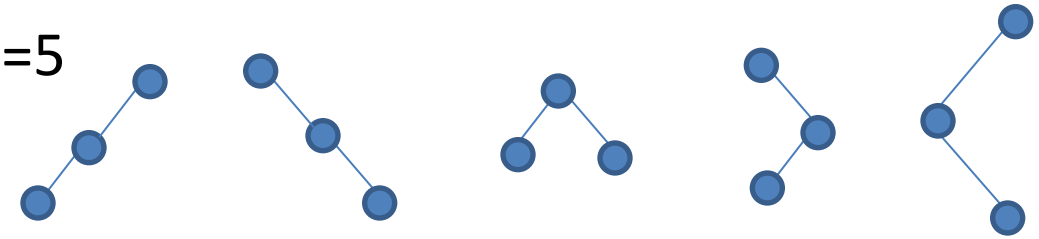
d) The number of binary trees of size n is : $b_n = \frac{1}{n+1} C_{2n}^n$

For n=1 ; b_1 =1

For n=2 ; b_2 =2

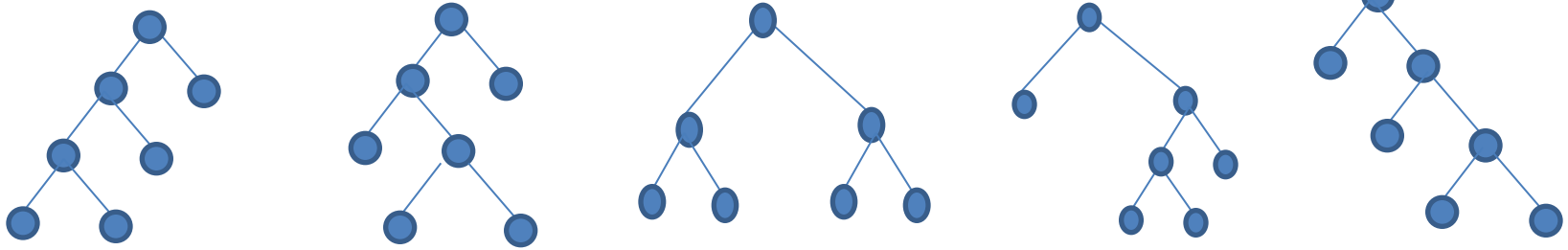


For n=3 ; b_3=5



e) The number of locally complete binary trees having (2n+1) nodes is $bc_n = \frac{1}{n+1} C_{2n}^n$

For n=3 ; bc_3 =5



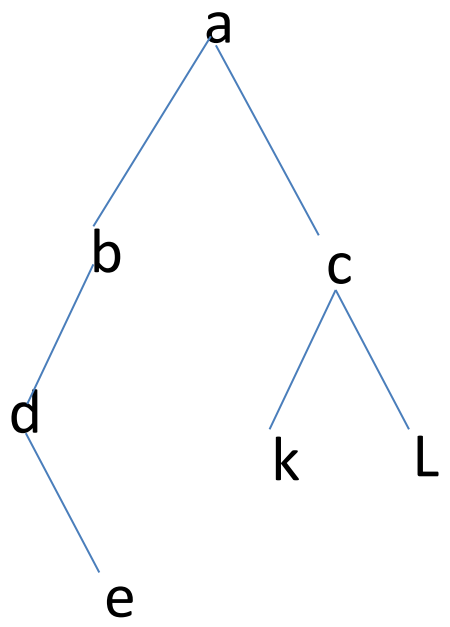
Representation of binary trees

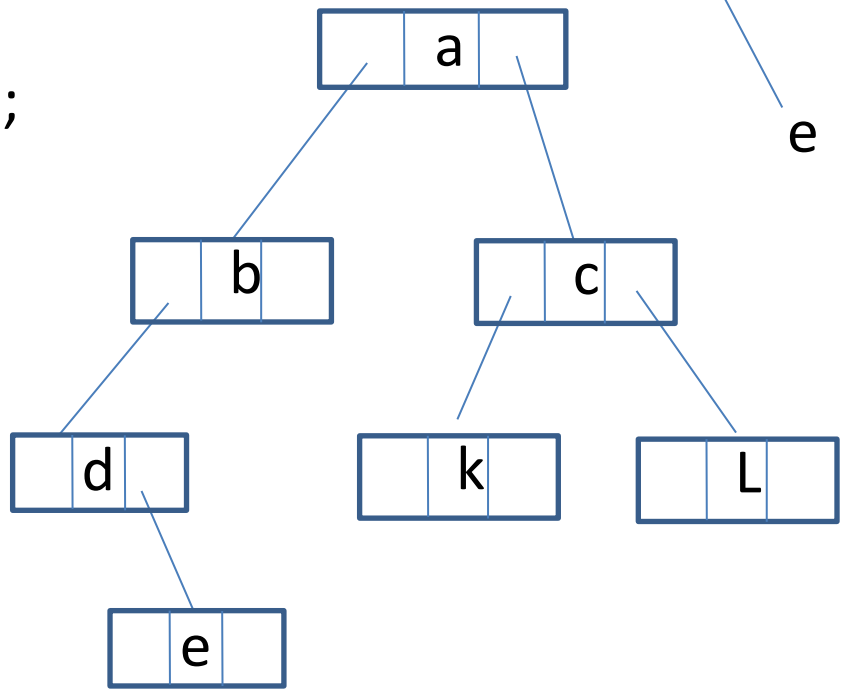
a) Using Pointers : Each node has two pointers: to the left subtree and to the right subtree.

The tree is determined by the address of its root.

```
struct Node {
        Element val;
        Node * g;
        Node *d};

typedef Node * Tree;
```

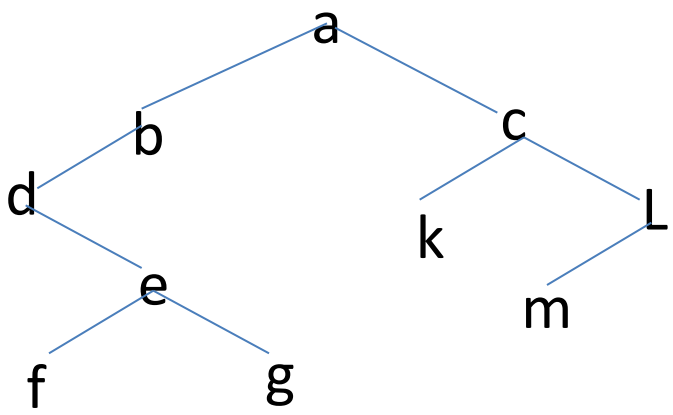




Operations :

A=empty-tree()	A=NULL
Content(Root(A))	A->val
g(A)	A->g
d(a)	A->d

b) Using arrays :



Root

3

tab

	V	G	D
1			
2	d	0	10
3	a	5	6
4	g	0	0
5	b	2	0
6	c	13	11
7			
8	f	0	0
9	m	0	0
10	e	8	4
11	L	9	0
12			
13	K	0	0

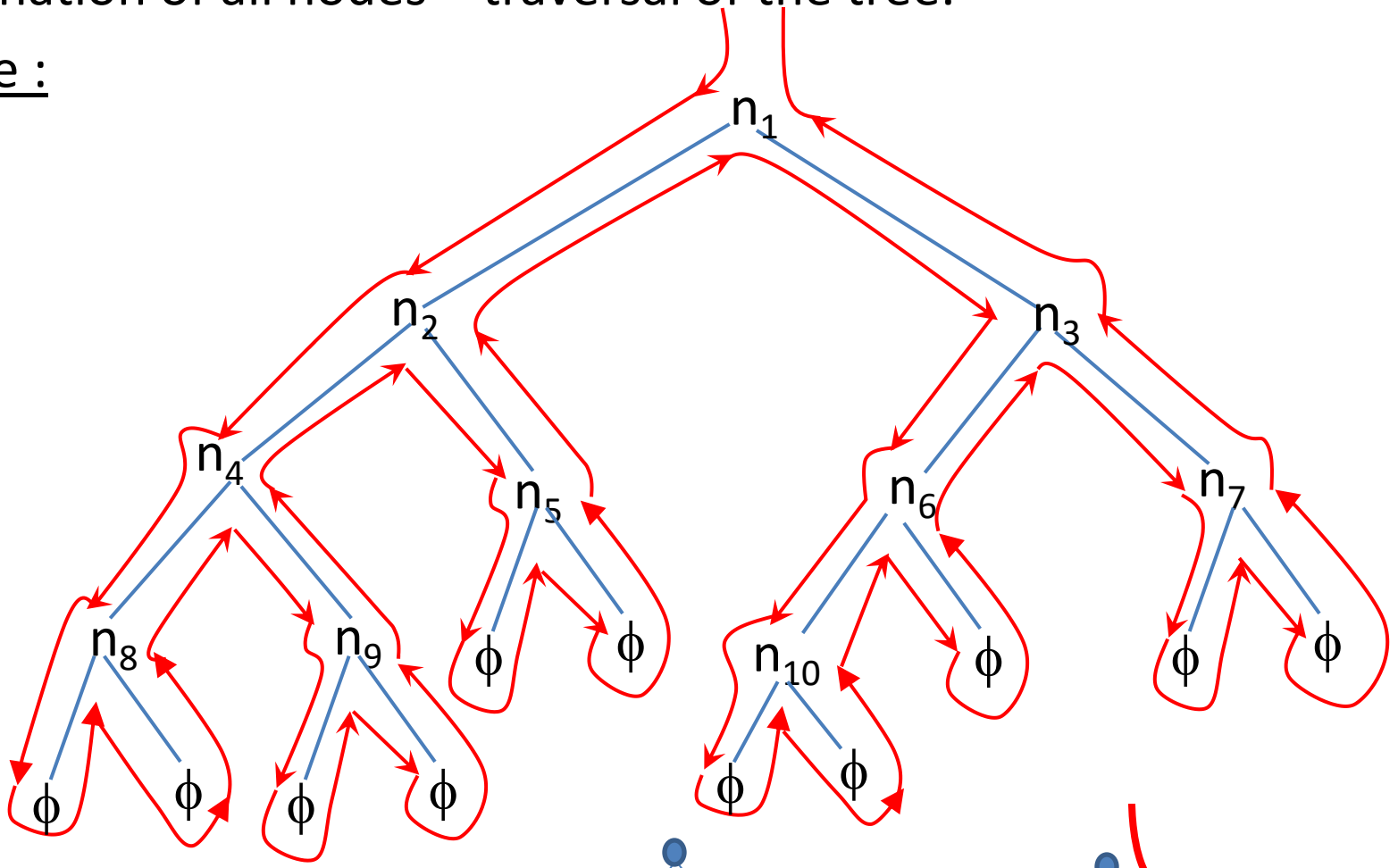
Translation of operations :

A=empty-tree ()	A.Root=0
Root(A)	A.Root
Content(Raoot(A))	A.tab[A.Root]
g(A)	A.tab[A.Root].g
d(A)	A.tab[A.Root].d

Note: Free cels can be chained together in a stack or in a queue.

Depth-first traversal of a binary tree : Systematic ordered examination of all nodes = traversal of the tree.

For a tree :



For a node :

Recursive algorithm for traversing a binary tree

```
void Parcours (Arbre A)
{  if (A == Arbre_vide) Term;
                              else   {        T1 ;
                                              Parcours (g(A));
                                              T2 ;
                                              Parcours (d(A));
                                              T3 ;
                                     }
}
```

Special cases: Classic exploration orders :

1) Prefix order or pre-order : Not $T_2 + T_3$; only T_1 in prefix order (on descent) + Term.

   On the example : $n_1, n_2, n_4, n_8, n_9, n_5, n_3, n_6, n_{10}, n_7$

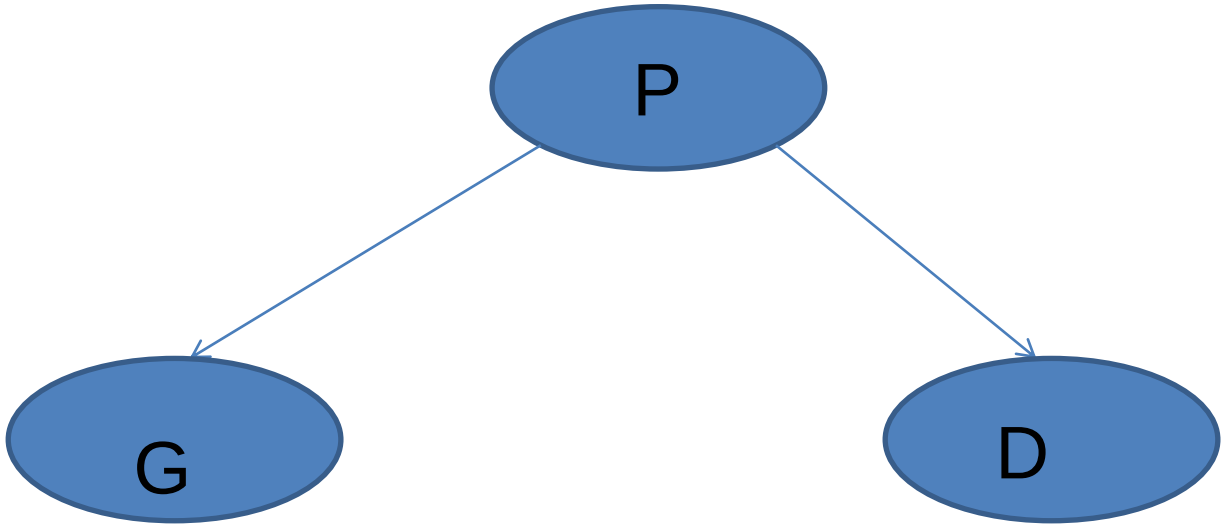
2) Infix or symmetrical order : Not $T_1 + T_3$; only T_2 in infix order (on the left rise) + Term.

   On the example : $n_8, n_4, n_9, n_2, n_5, n_1, n_{10}, n_6, n_3, n_7$

3) Suffix order or post-order : Not $T_1 + T_2$; only T_3 in suffix order (on the right rise) + Term.

   On the example : $n_8, n_9, n_4, n_5, n_2, n_{10}, n_6, n_7, n_3, n_1$

Note : the hierarchical order $\Leftrightarrow$ level-based course.



Try this traversal simplification

Prefix : P-G-D

Infix : G-P-D

Suffix : G-D-P

General Planar Trees or General Trees

The number of a node's sons can be > 2.

ADT Tree :

A=<o, A_1, A_2, . . . , A_p > where o : root and A_1, A_2, . . . , A_p : disjoint trees.

<A_1, A_2, . . . , A_p > is called forest.

So : A = <o, F> where F = ϕ + A + <A, A> + <A, A, A> + . . .

A: set of trees. F= set of forests. ϕ= empty forest.

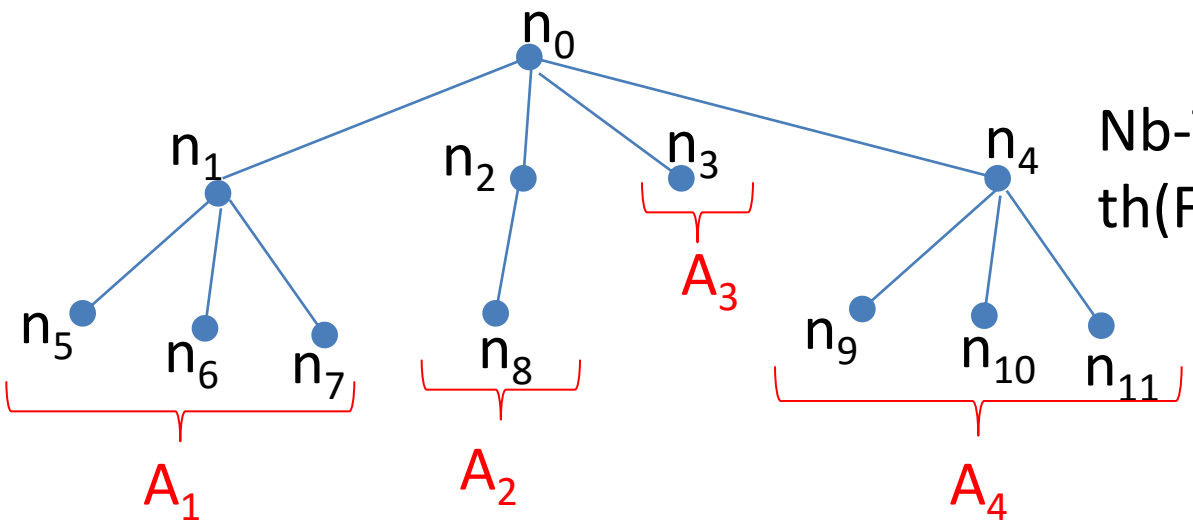
Example : Let the following tree A be :

Root (A)=n_0

Trees-List (A)= <A_1, A_2, A_3, A_4>
= F

Nb-Trees (F) = 4

th(F, 2)= A_2



Signature of the TAD Tree, Forest

Sort GenTree, Forest

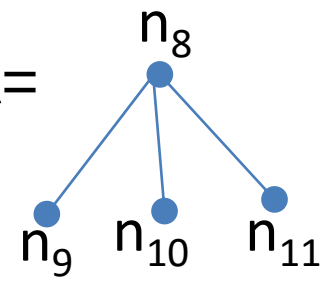
Uses Node, int

Operations

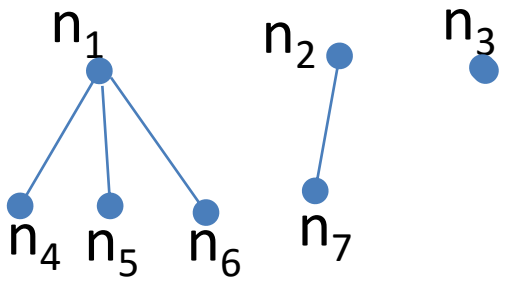
cons	: Node x Forest	$\rightarrow$ GenTree
root	: GenTree	$\rightarrow$ Node
Tree-list	: GenTree	$\rightarrow$ Forest
empty-forest	:	$\rightarrow$ Forest
ith	: Forest x int	$\rightarrow$ GenTree
Tree-nb	: Forest	$\rightarrow$ int
insert	: Forest x int x GenTree	$\rightarrow$ Forest

- Possibility of extension by other operations: delete a tree in a forest, content of a node . . . etc.
- No left-right concept
- Sons ordered from left to right
- A tree is never empty.

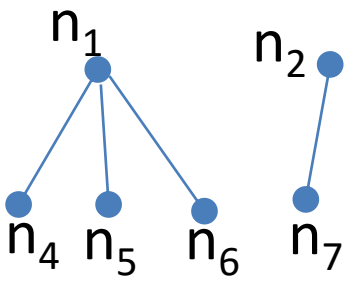
A=

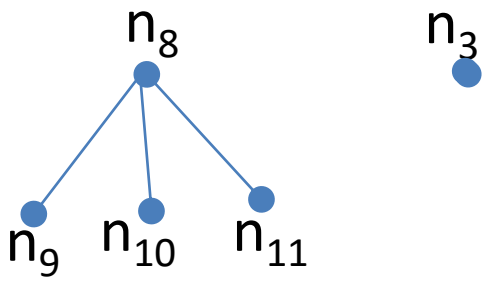


F=



F'=
insert(F, i=3, A)





ith (F', k=1 / 2)= ith(F, k)

ith (F', k=3) = A

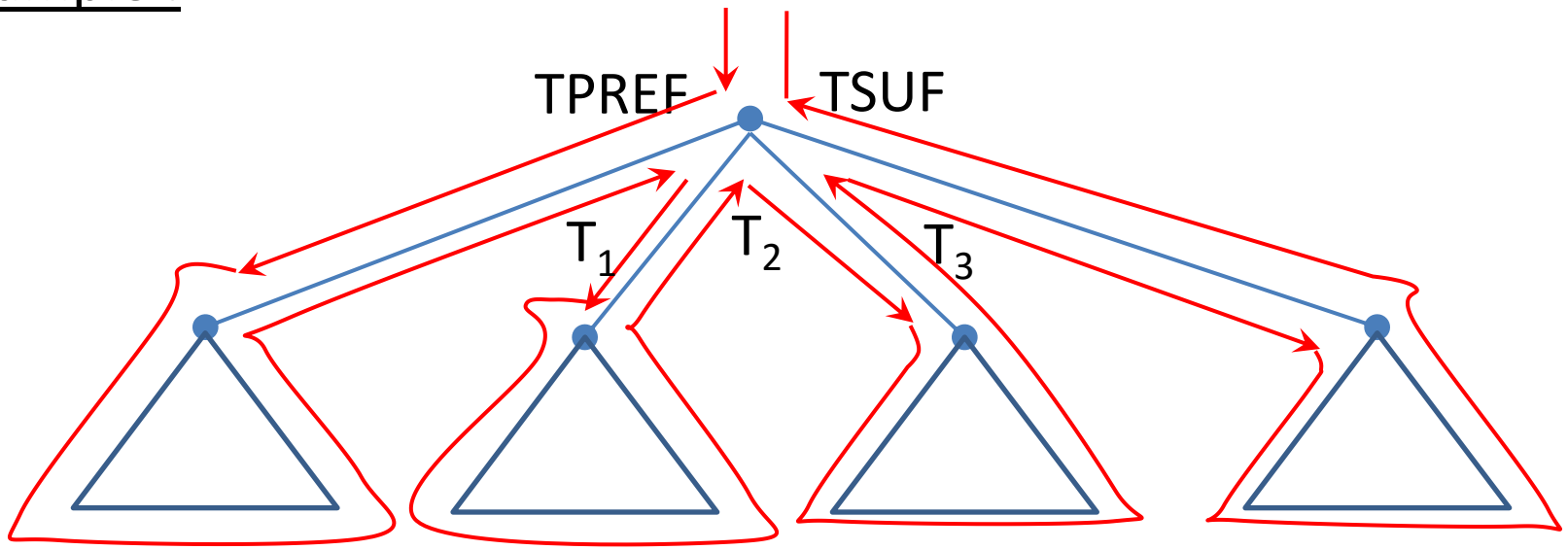
ith (F', k=4)= ith(F, k-1)

Traversal of a general tree

Extension of the traversal of binary trees. Thus, a node having n children is traversed (n+1) times.

- on the first pass, apply the TPREF treatment
- at the (i+1) th passage, the treatment T(i)
- at the last pass, the TSUF treatment
- at wireless nodes, TERM treatment

Example :



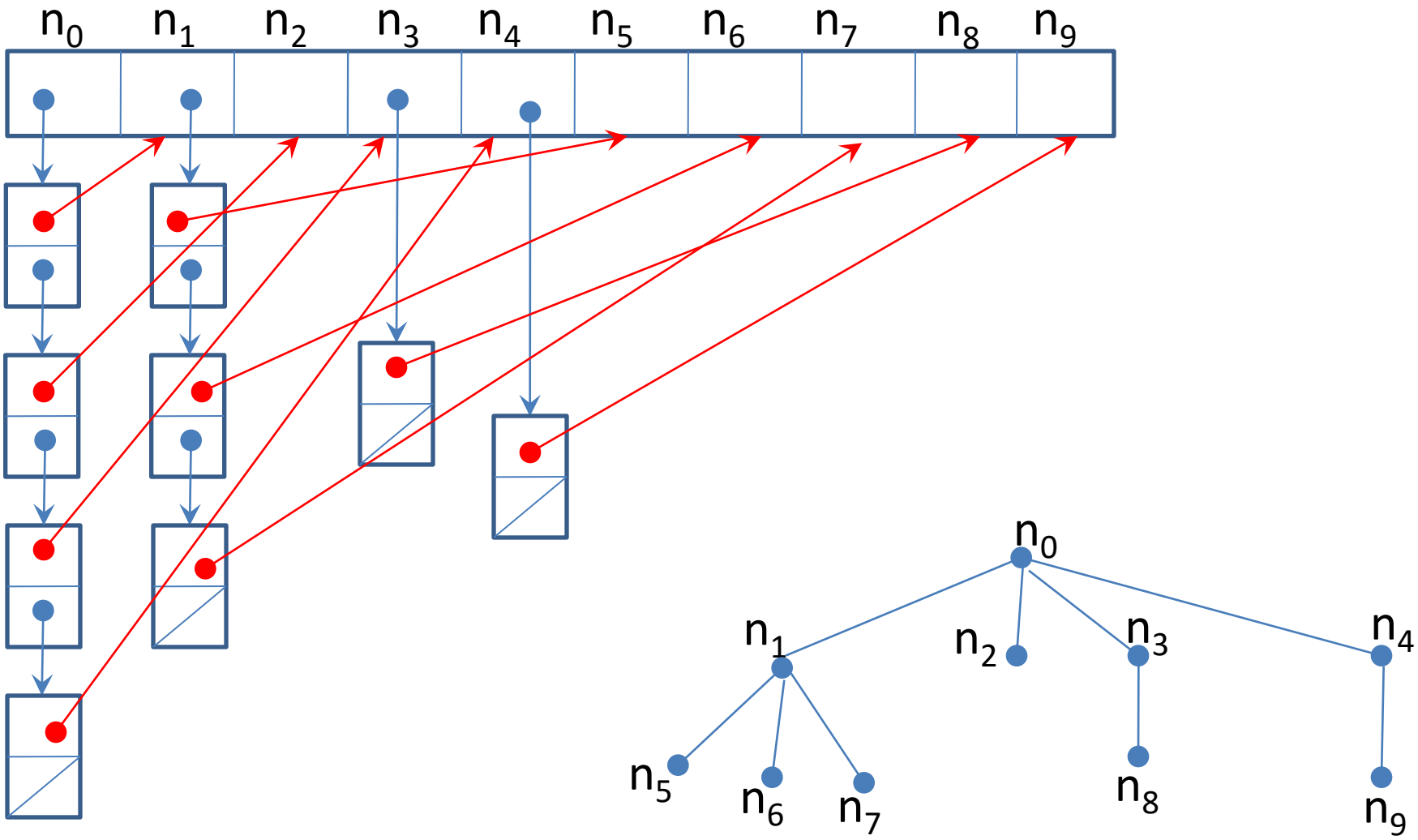
Let the operation leaf: GenTree → Boolean such that :

leaf (A) ← (Tree-list (A) =empty-forest)

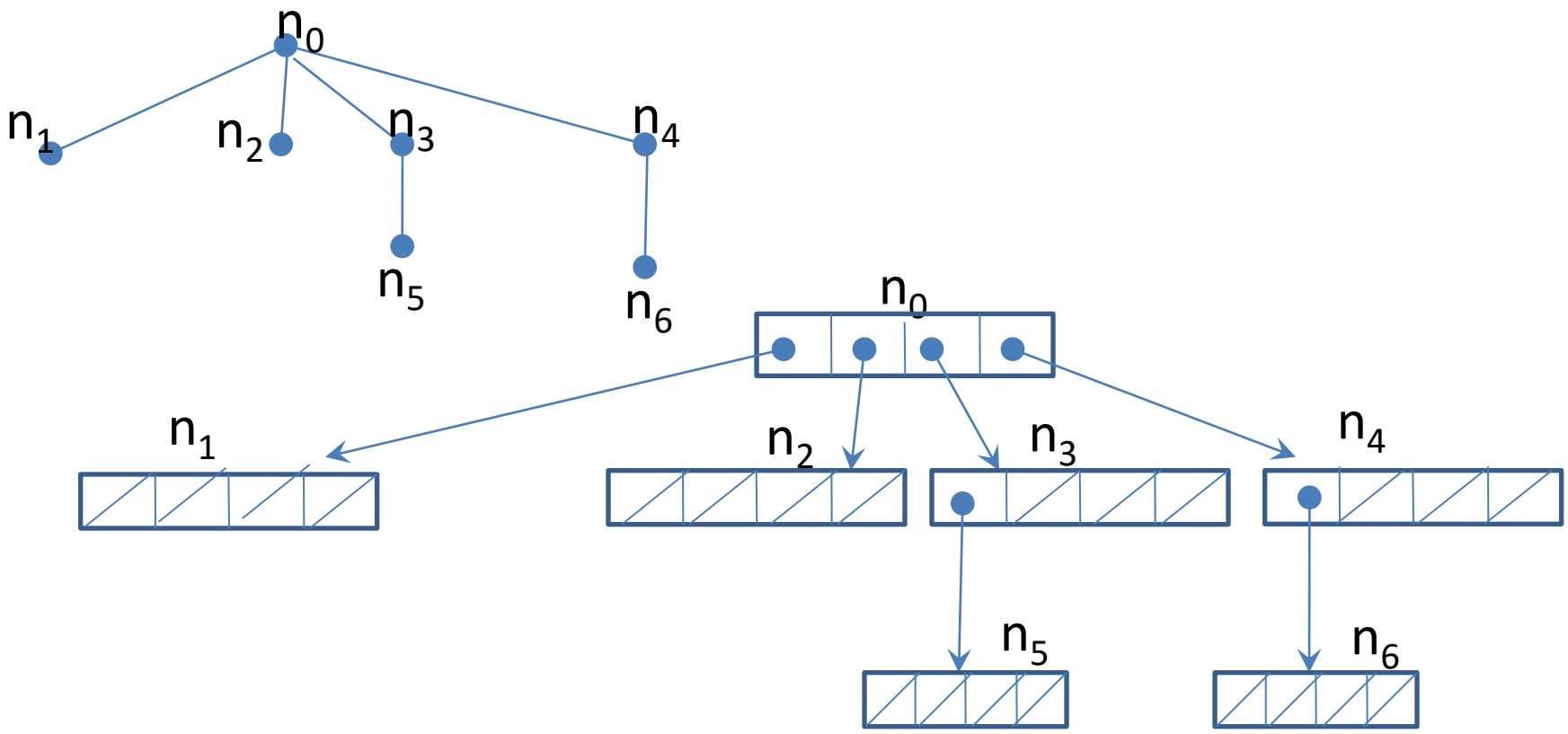
```
Void parcrs (GenTree A) {
    int i, nb;
    nb=Tree-nb (Tree-list(A));
    if (leaf (A)) TERM;
                else   {     TPREF; {treatment before seeing his sons}
                             for (int i= 1; i<nb; i++)
                                    {parcrs (ith (Tree-list (A),i));
                                    T(i); }
                                Parcrs (ith (Tree-list (A),nb));
                                TSUF;  {treatment after seeing his sons}
                        }
}
```

Representation of general trees

a) ***Representation by list of children :*** This representation is not well-suited to dynamic management because of the node array.



b) ***Representation by a pointer to each subtree :*** This representation requires too much memory space.



c) ***Representation using binary trees :***

The size of a general tree is the total number of its nodes.

size (empty-forest)=0

size (insert (F, i, A)) = size (F) + size (A)

size (cons (o, F)) = 1+ size (F)

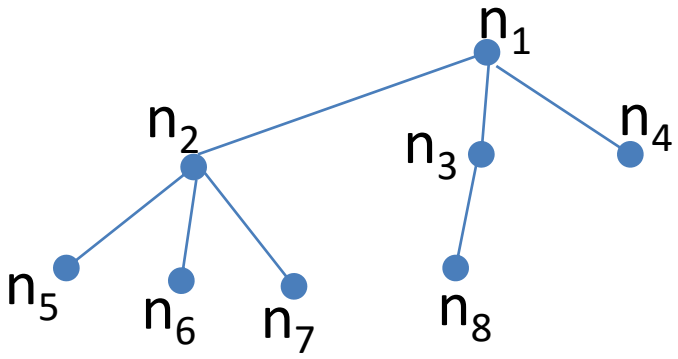
Lemma: There exists a bijection between general trees of size n+1 and forests of size n.

Proof :

We associate with the tree A= <o, A_1, A_2, . . . , A_p> the forest F=<A_1, A_2, . . . , A_p>

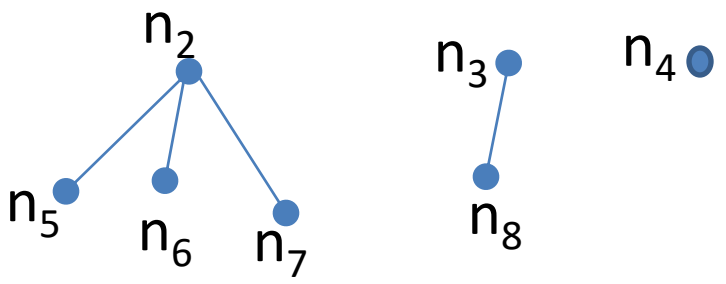
Example :

A



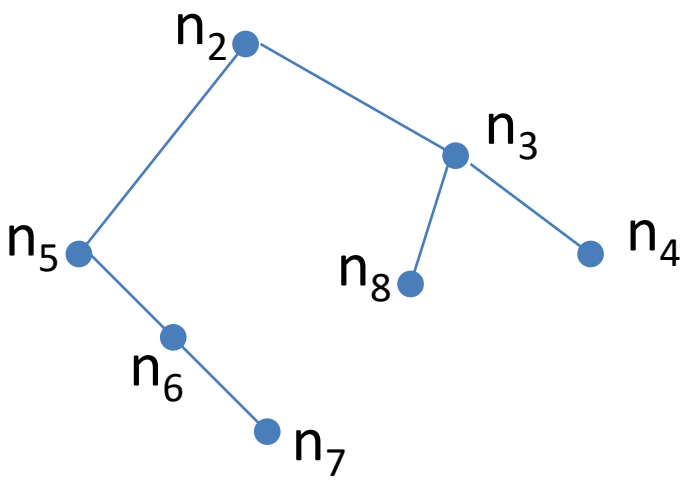
⇒

F



Bijection eldest son – right brother:

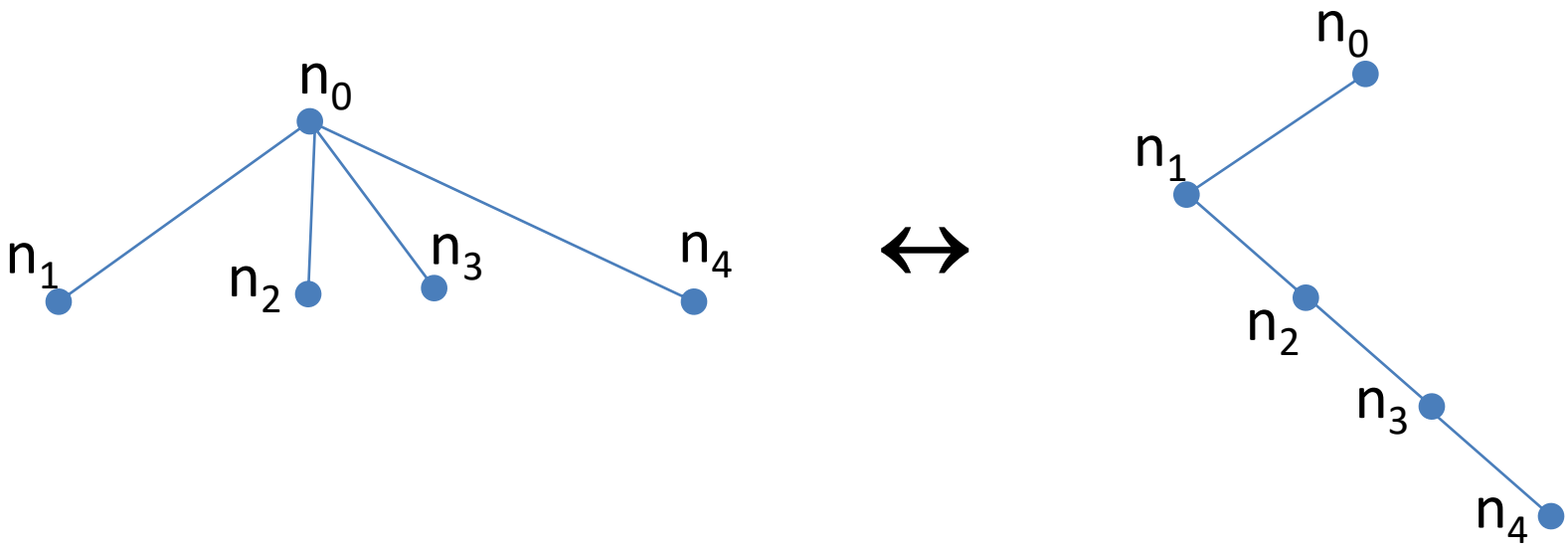
F ⇒



Proposition 1 : There exists a bijection between forests of size n and binary trees of size n.

Proof :

- The binary tree is constructed by the following process: given a node in a forest, we create a left link to its first child (eldest child) and a right link to its brother located on the right.
- The forest is built from the binary tree by preserving left links and removing right links while putting them at the same level as the eldest child.

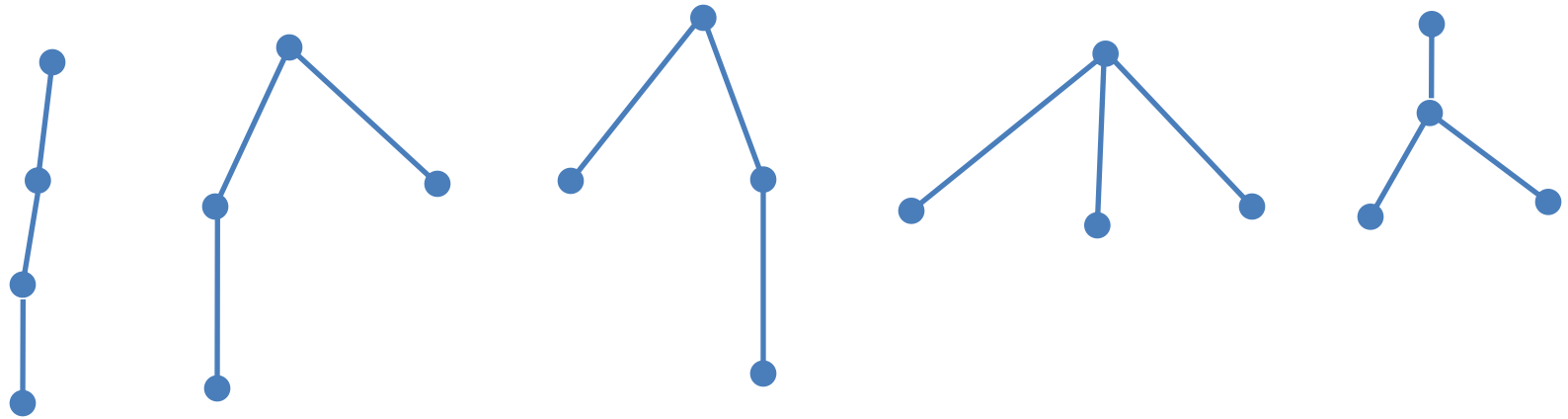


Proposition 2 : The number a_{n+1} of planar trees with size n+1 is :

$$a_{n+1} = \frac{1}{n+1} C_{2n}^{n}$$

Example : for n=3; $a_4 = \frac{1}{4} C_6^3 = 5$

With 4 nodes, we can build 5 different trees:



Exercises of DT N° 3

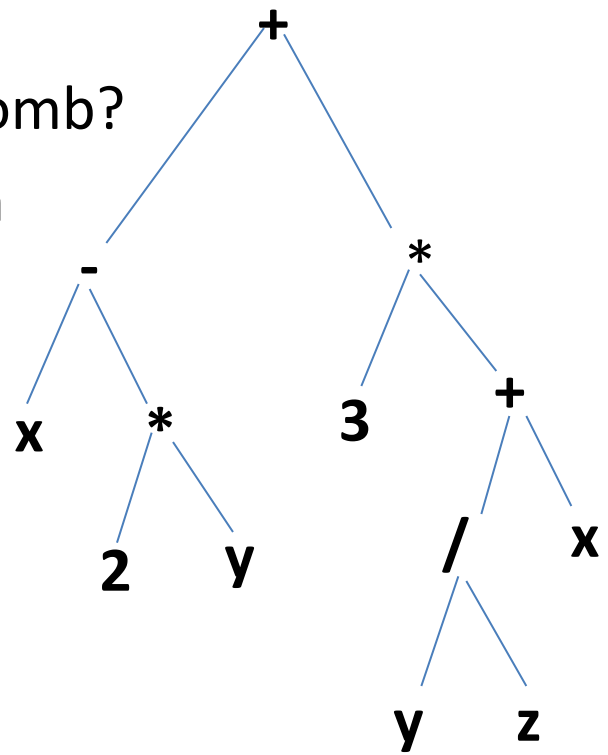
Exercise 01 : An arithmetic expression can be represented by a binary tree. If it only uses binary operators (+, -, *, /), the tree is locally complete.

Let the arithmetic expression represented by the tree A be:

1) Is A degenerate? Complete? Perfect?
2) Does it contain a left comb subtree? Right comb?
3) Give the expressions represented by A when

A is read in prefix, infix, and postfix order.

Conclude.



Exercises of DT N° 3

Exercise 02 :

a) Give the number of locally complete binary trees of size 2n+1

b) Deduce all locally complete binary trees whose symmetric traversal gives the expression: 2+3*4+5

c) Give a procedure for correctly parenthetizing an expression representing a tree.

Exercises of DT N° 03

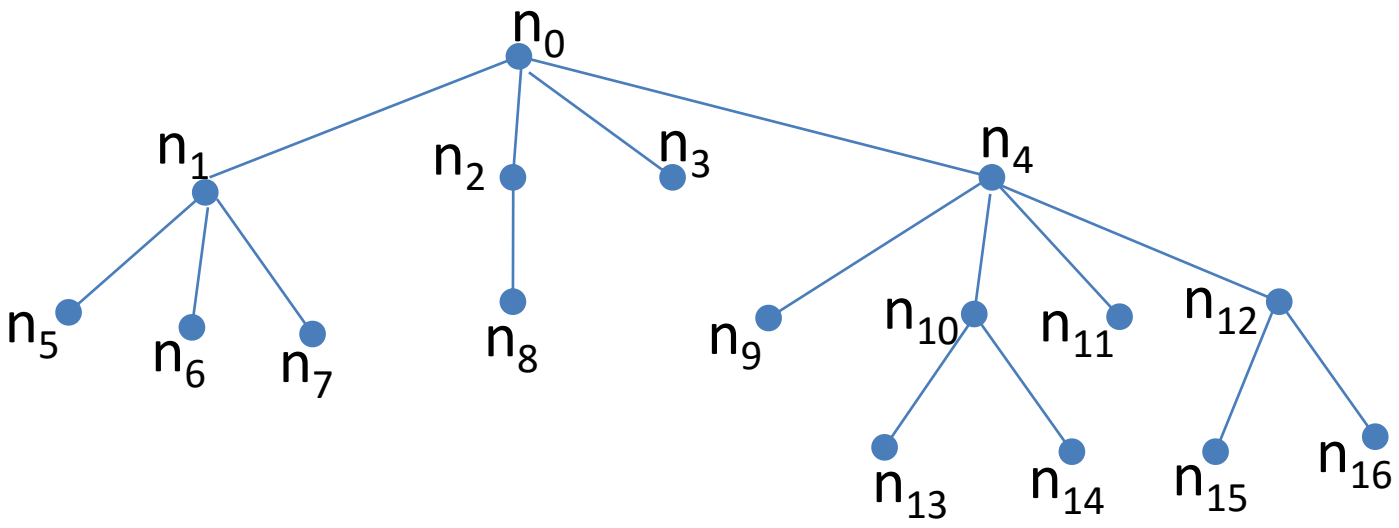
Exercise 03 : write a program for the construction operation :

< _ , _ , _ > : Node x GenTree x GenTree $\rightarrow$ GenTree

for chained representation.

Exercises of DT N° 03

Exercise 04: Let the following general tree be :



1) Give a procedure (based on the one in the course) that allows you to traverse the tree as shown in the expression :

$(n_0(n_1(n_5)(n_6)(n_7))(n_2\ (n_8))(n_3)(n_4(n_9)(n_{10}(n_{13})(n_{14}))(n_{11})(n_{12}(n_{15})(n_{16}))))$

2) Transform this tree into a binary tree.

Exercises of DT N° 03

Exercise 05 : Construct a BST for the ordered set of values E={25, 60, 35, 10, 5, 20, 65, 45, 70, 40, 15, 43, 30, 37}

a) By addition to the leaves

b) By addition in root

Exercise 06: Function for adding to the leaves

Exercise 07: Function for addition to root

Exercise 08 : Program the deletion operation in SBT

Exercises of DT N° 03

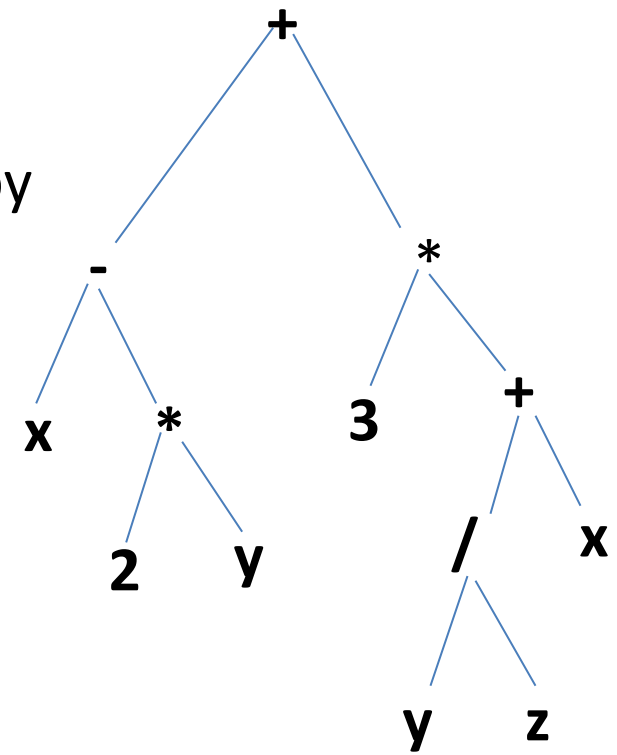
Exercise 09 : personal work

Program the eldest son – right brother bijection to transform a general tree into another binary one.

Continuation of DT N° 03

Exercise 10 : personal work

Let the arithmetic expression represented by the locally complete binary tree A:



Give the expressions represented by A

In the right-hand depth-first search, prefix,

infix, and postfix. Conclude

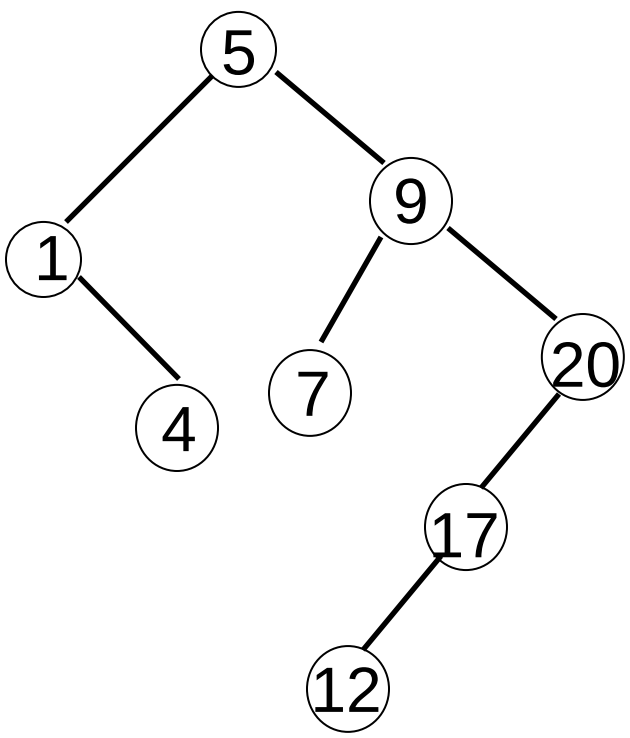
Continuation of DT N° 03

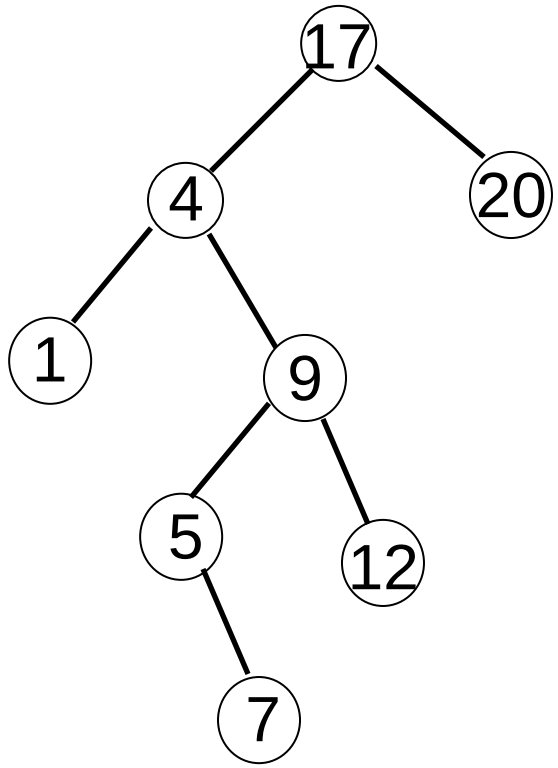
Exercise 11 : personal work

Give the declarations to represent a general tree by list of children.

Binary search trees

Examples of binary search trees



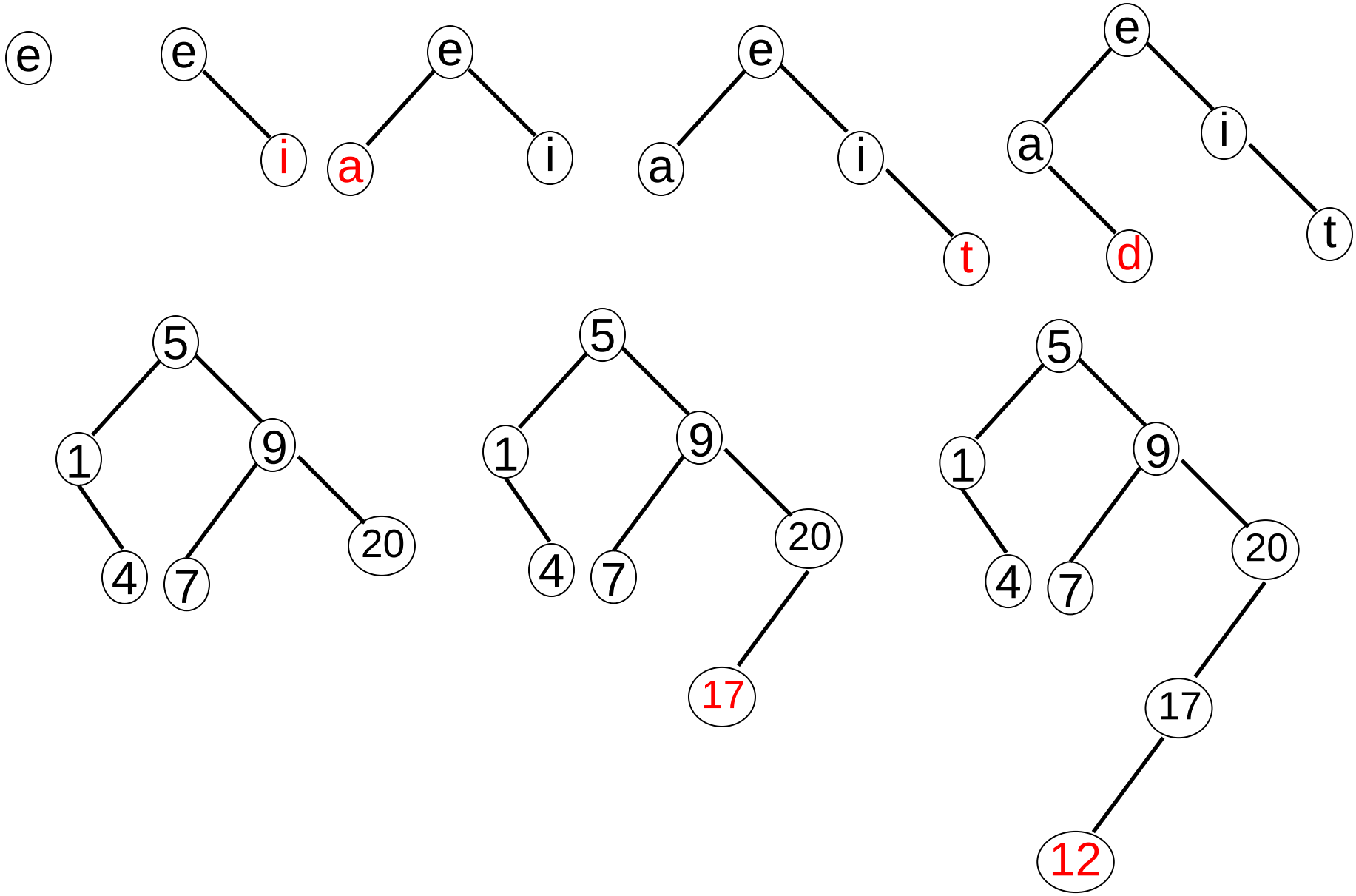


Definition : A binary search tree is a labeled binary tree such that for any node **v** in the tree:

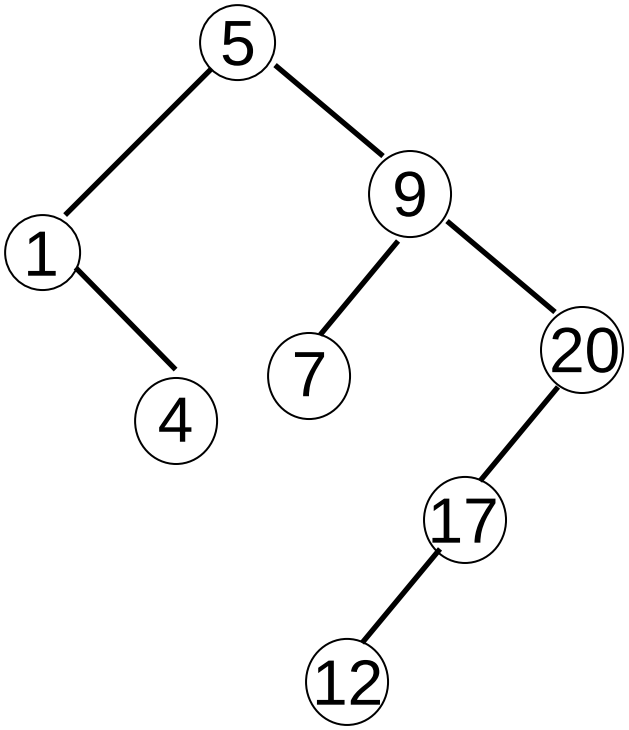
- The elements of all nodes in the left subtree of **v** are less than or equal to element **v**.
- The elements of all nodes in the right subtree of **v** are greater than element **v**.

It follows immediately from this definition that the symmetrical traversal of a binary search tree produces the sequence of elements sorted in ascending order.

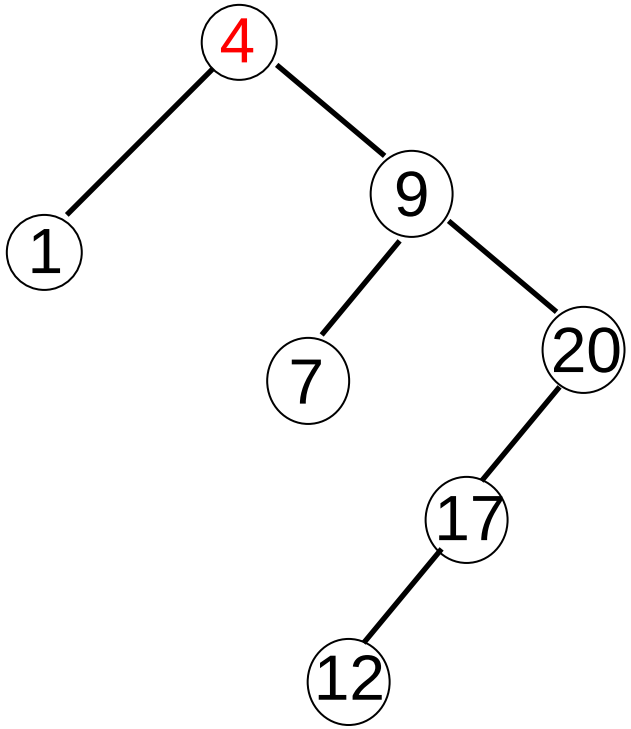
Construction by successive additions to the leaves



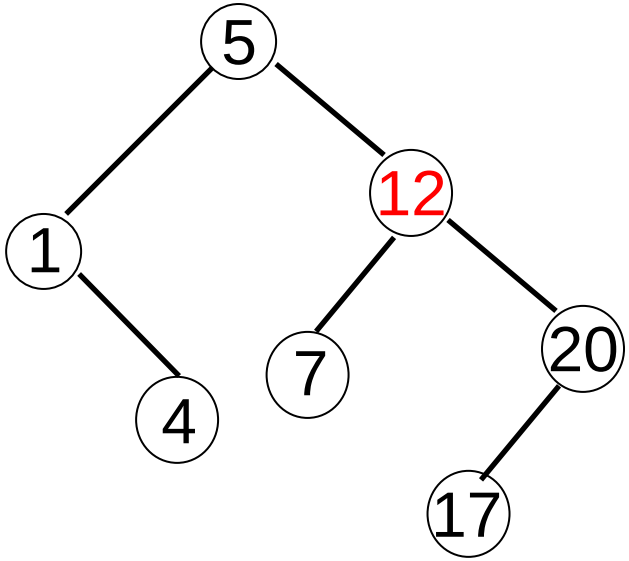
Deleting an item

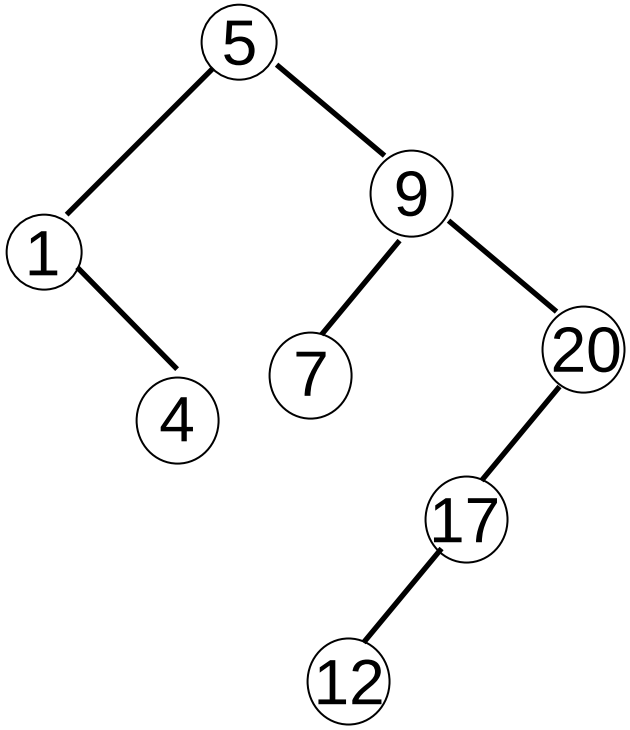


Deleting 5

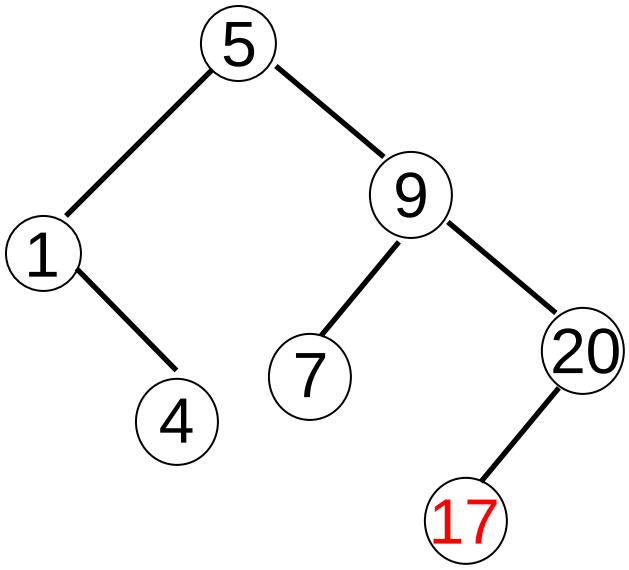


Deleting 9

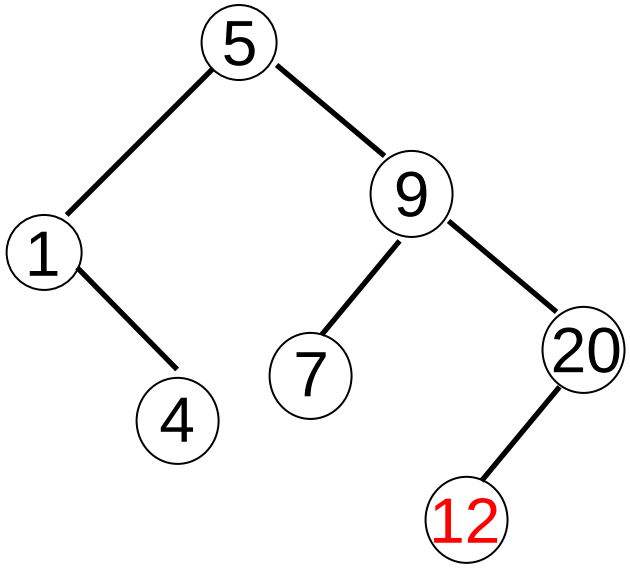




Deleting 12

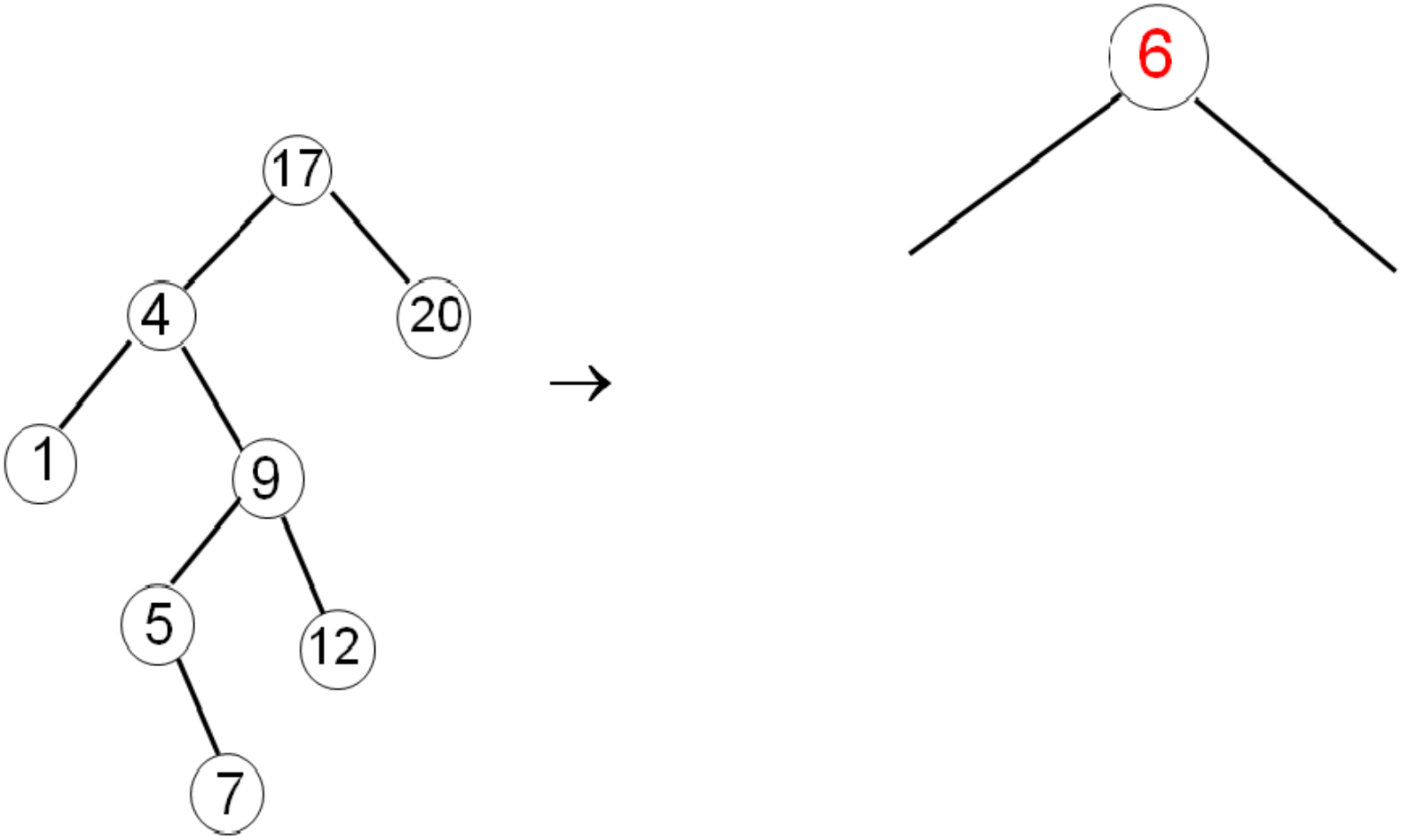


Deleting 17



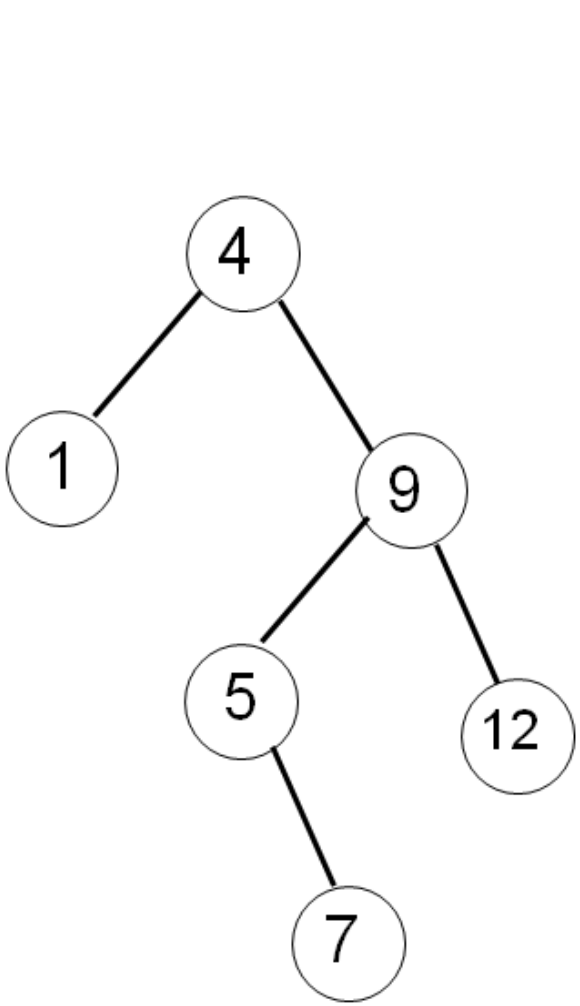
Addition to the root and cutting

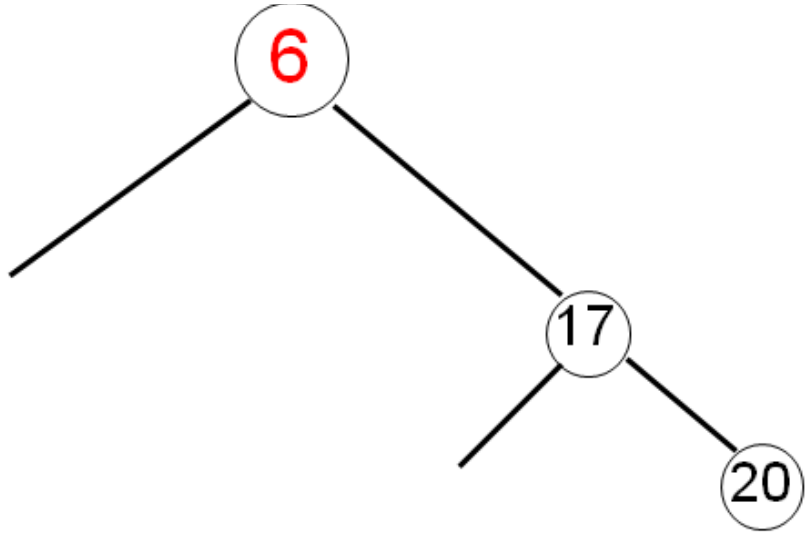
Cut according to element 6



Addition to the root and cutting

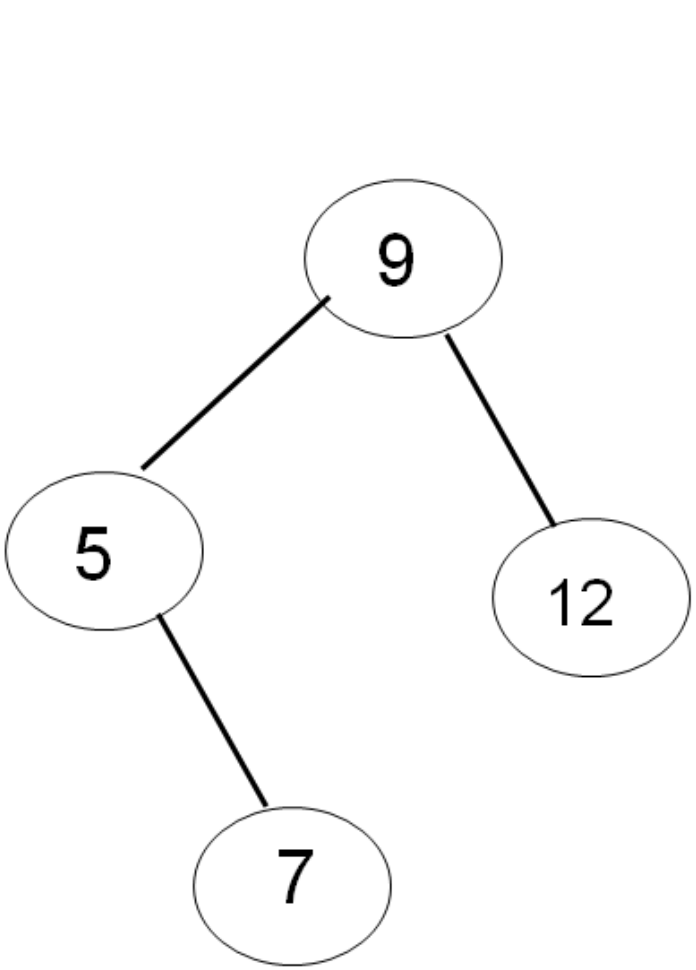
Cut according to element 6

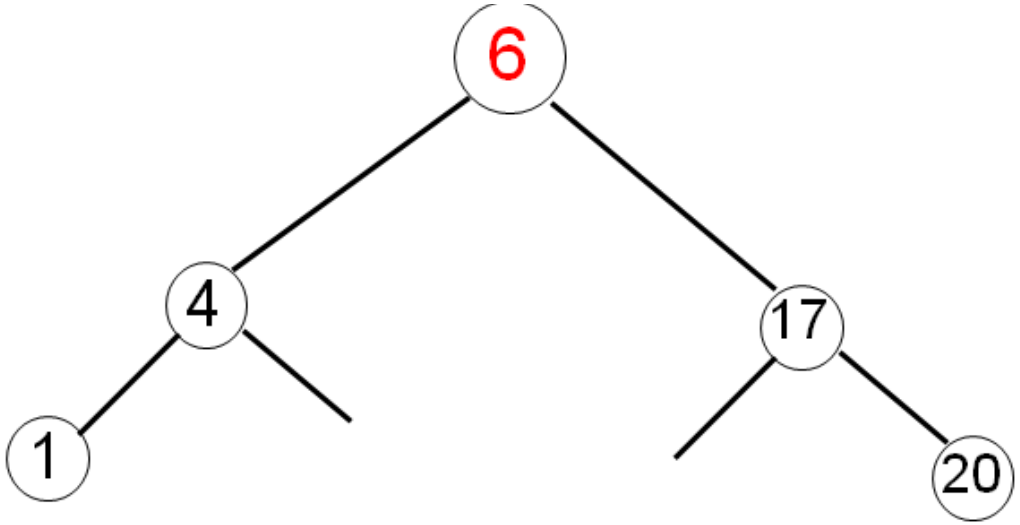




Addition to the root and cutting

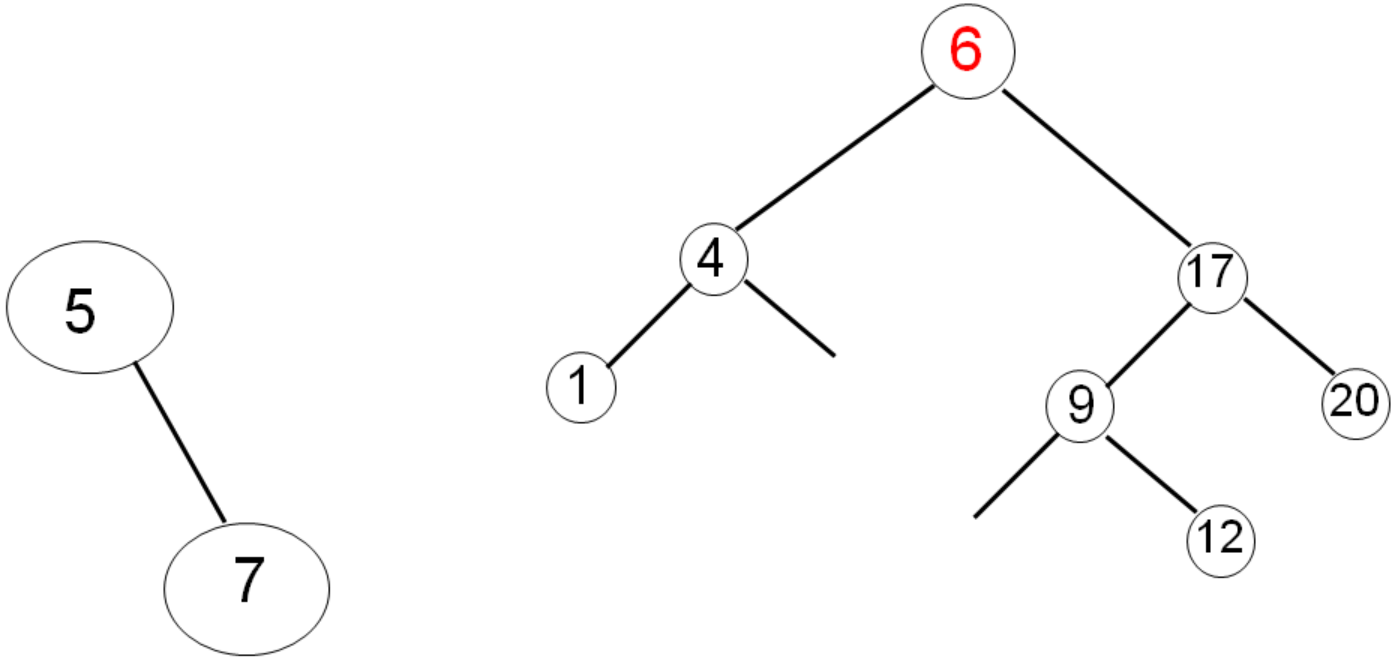
Cut according to element 6





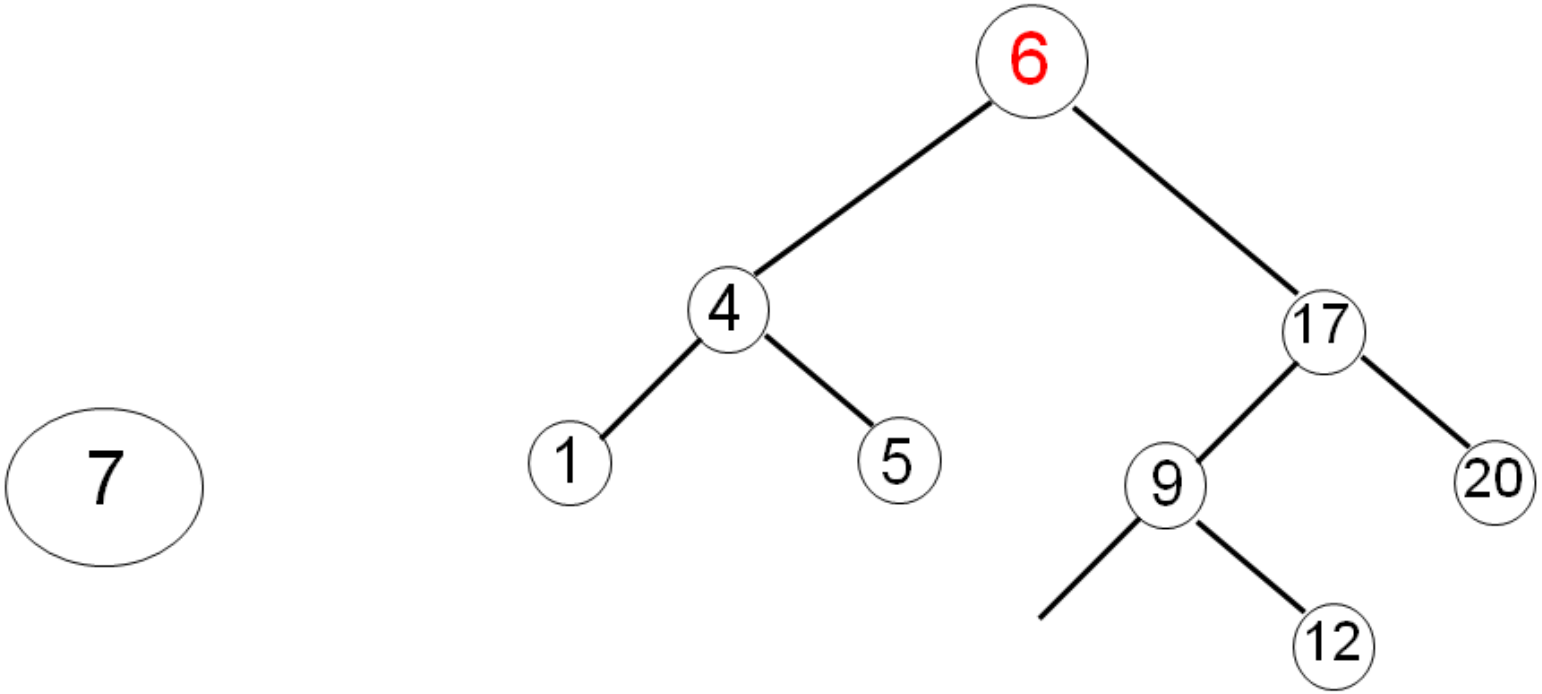
Addition to the root and cutting

Cut according to element 6



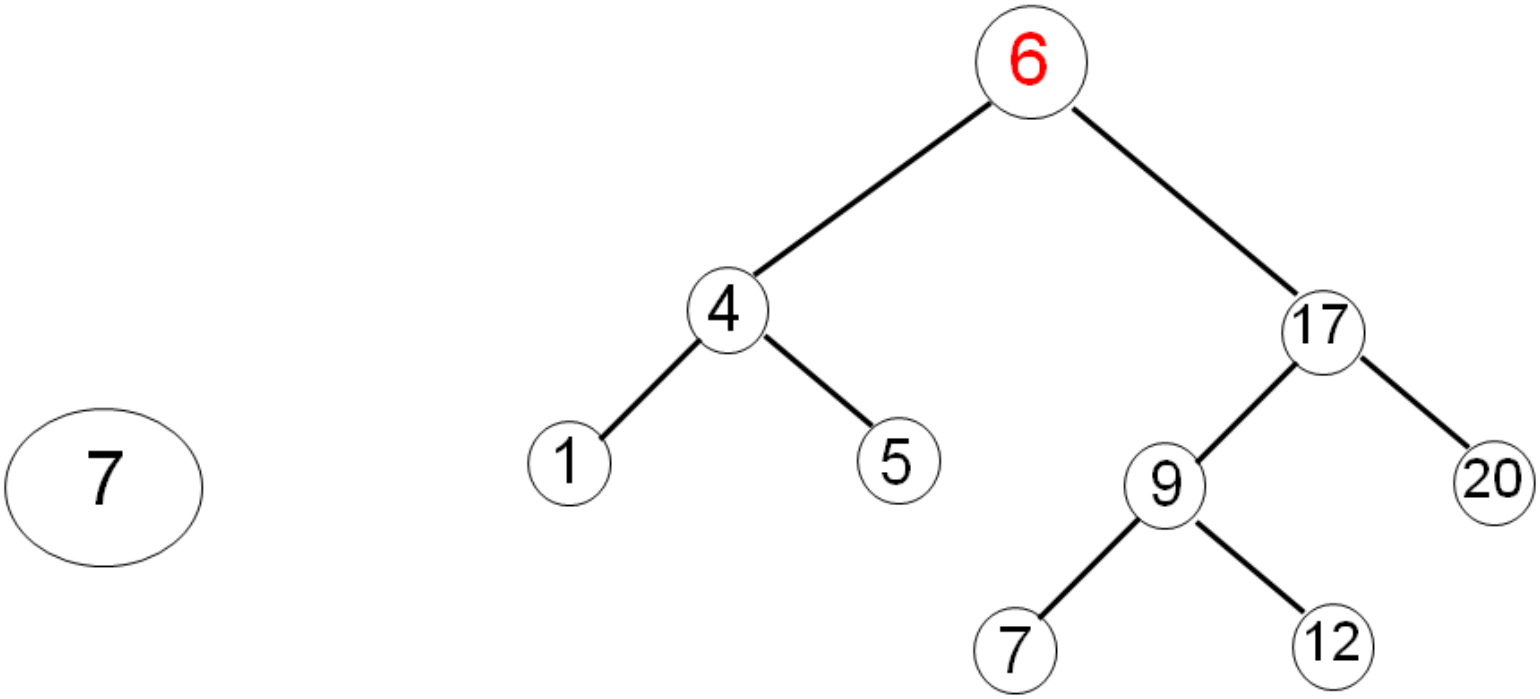
Addition to the root and cutting

Cut according to element 6



Addition to the root and cutting

Cut according to element 6



TREE Structure :

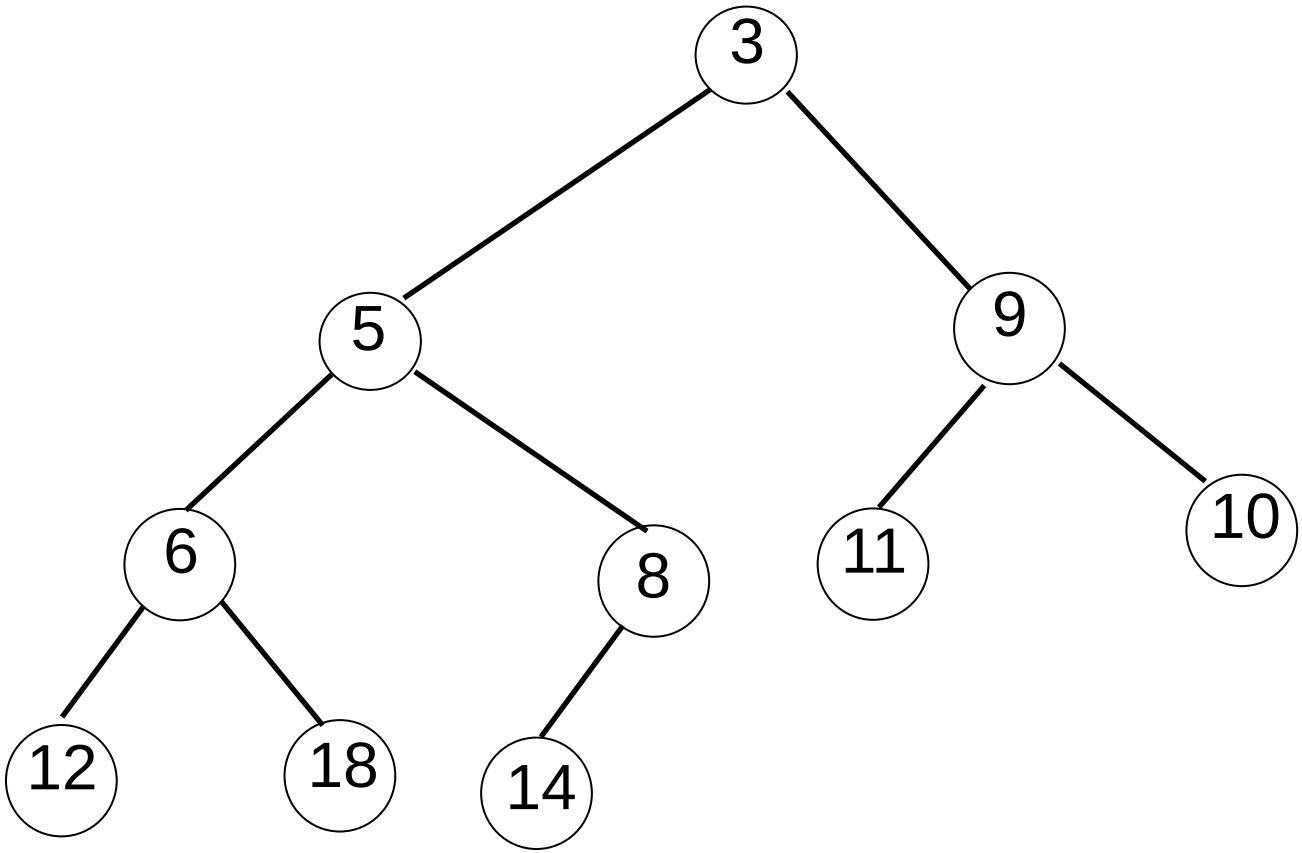
Heap

Definition of the heap structure

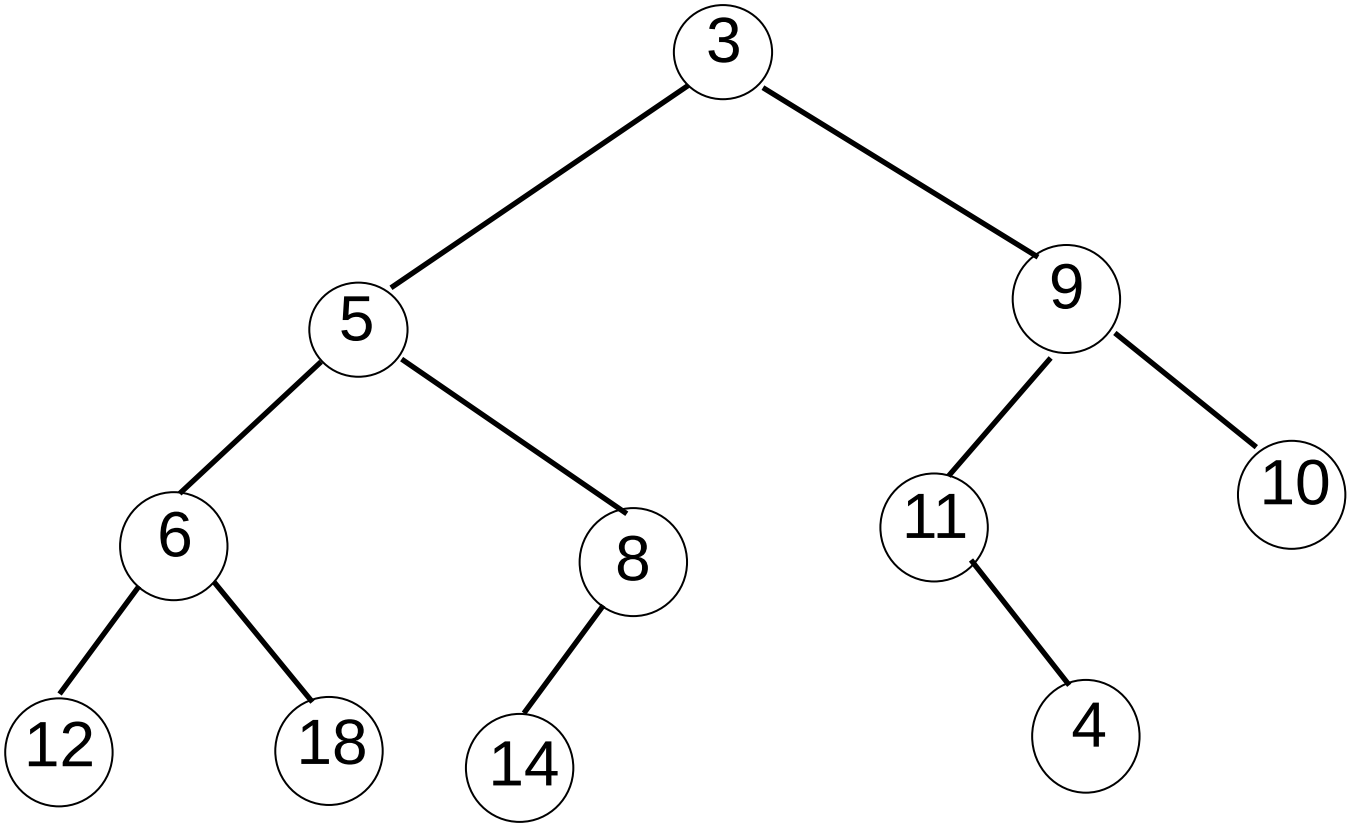
A heap is an array representing **a partially ordered perfect binary tree.**

Perfect tree

It is a binary tree whose all levels are filled except possibly the last level whose nodes must be grouped from the left of the tree.



A perfect binary Tree

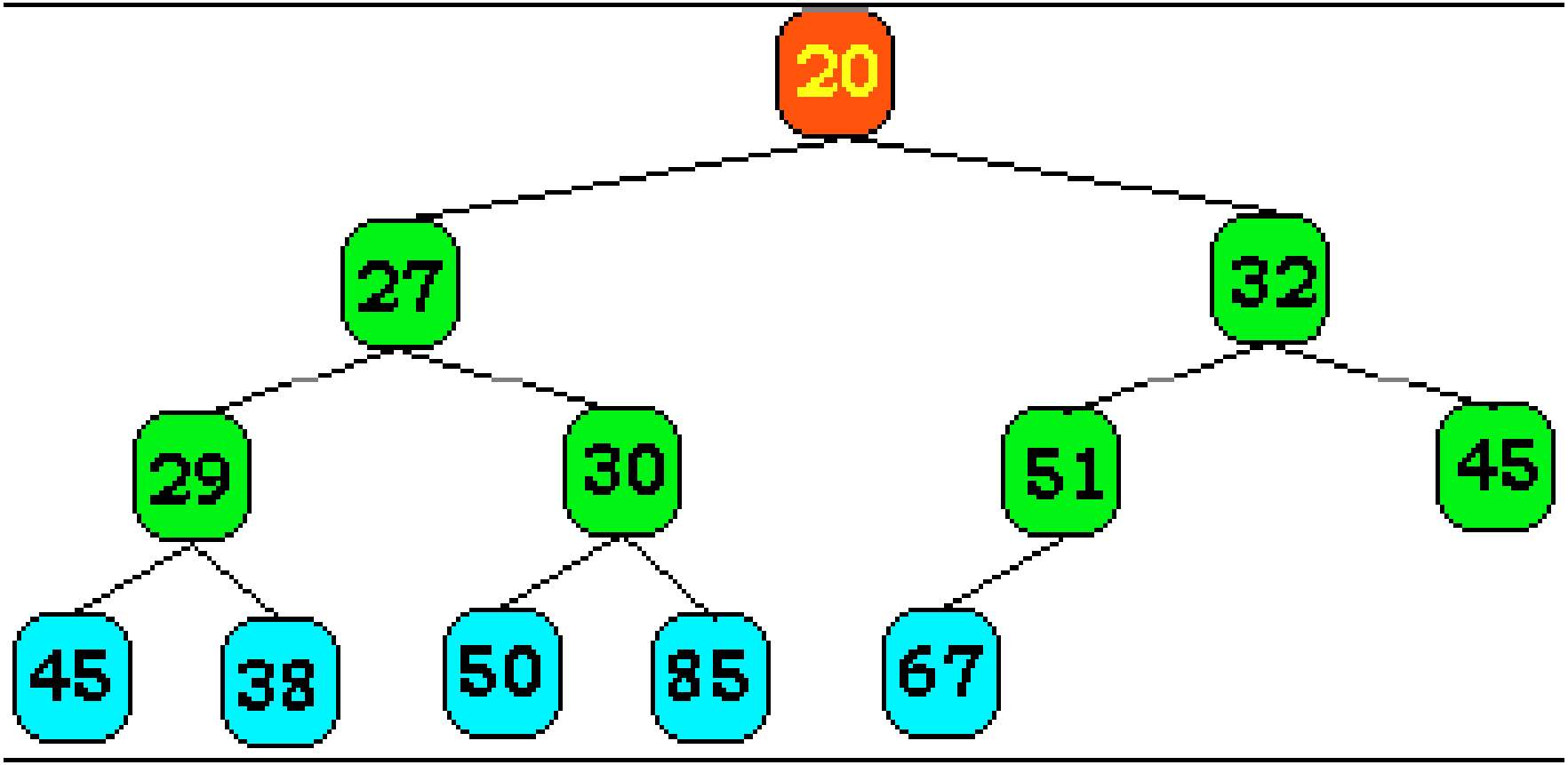


Example of a non-perfect binary tree

Partially ordered binary tree

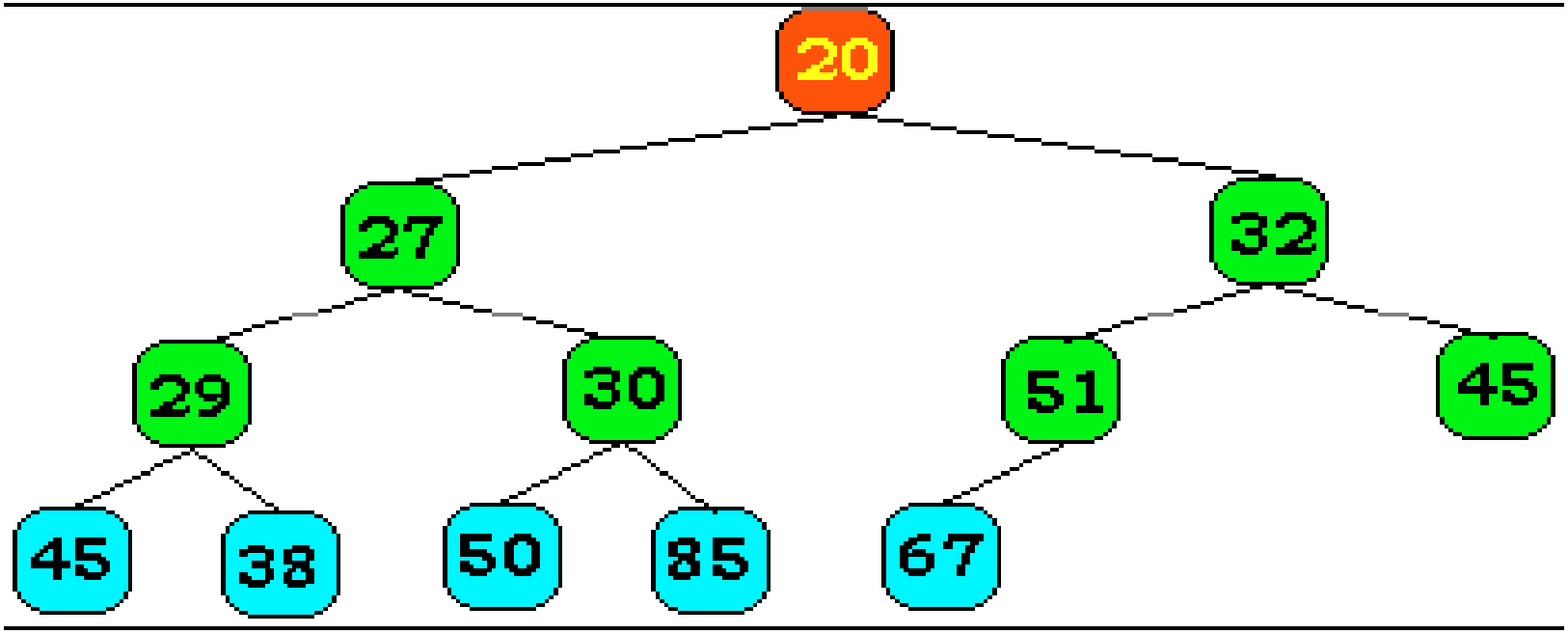
- *It is a labeled tree whose nodes belong to a set provided with a total order relation (Integers, reals etc.) such that any given node has a value less than or equal to those of its children.*
- *If we represent a list or a set of elements by such a tree, we always have a minimum element at the root.*

- *Example of a **partially ordered tree** on the set {20, 27, 29, 30, 32, 38, 45, 45, 50, 51, 67, 85} of natural numbers.*



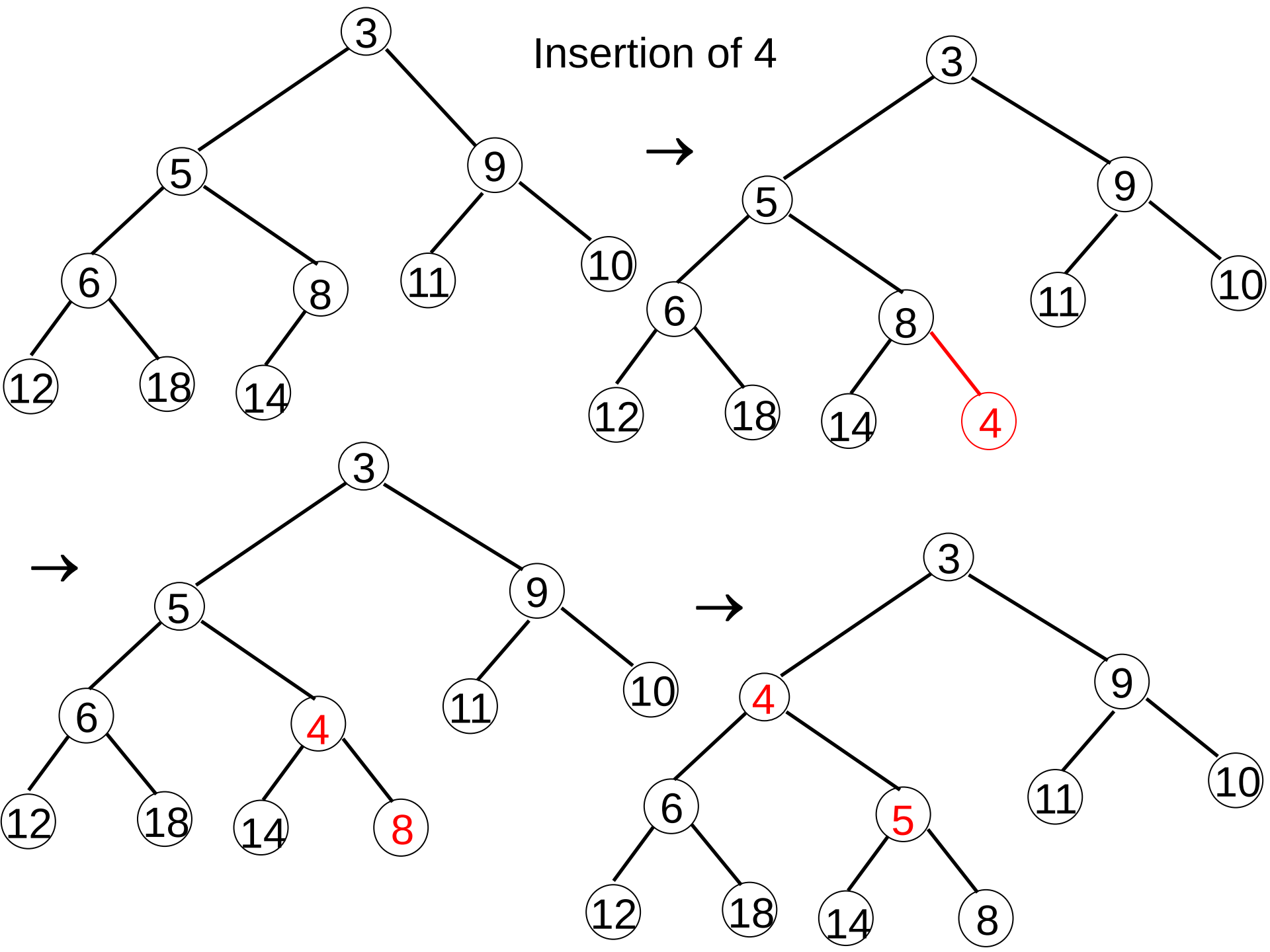
A partially ordered binary tree

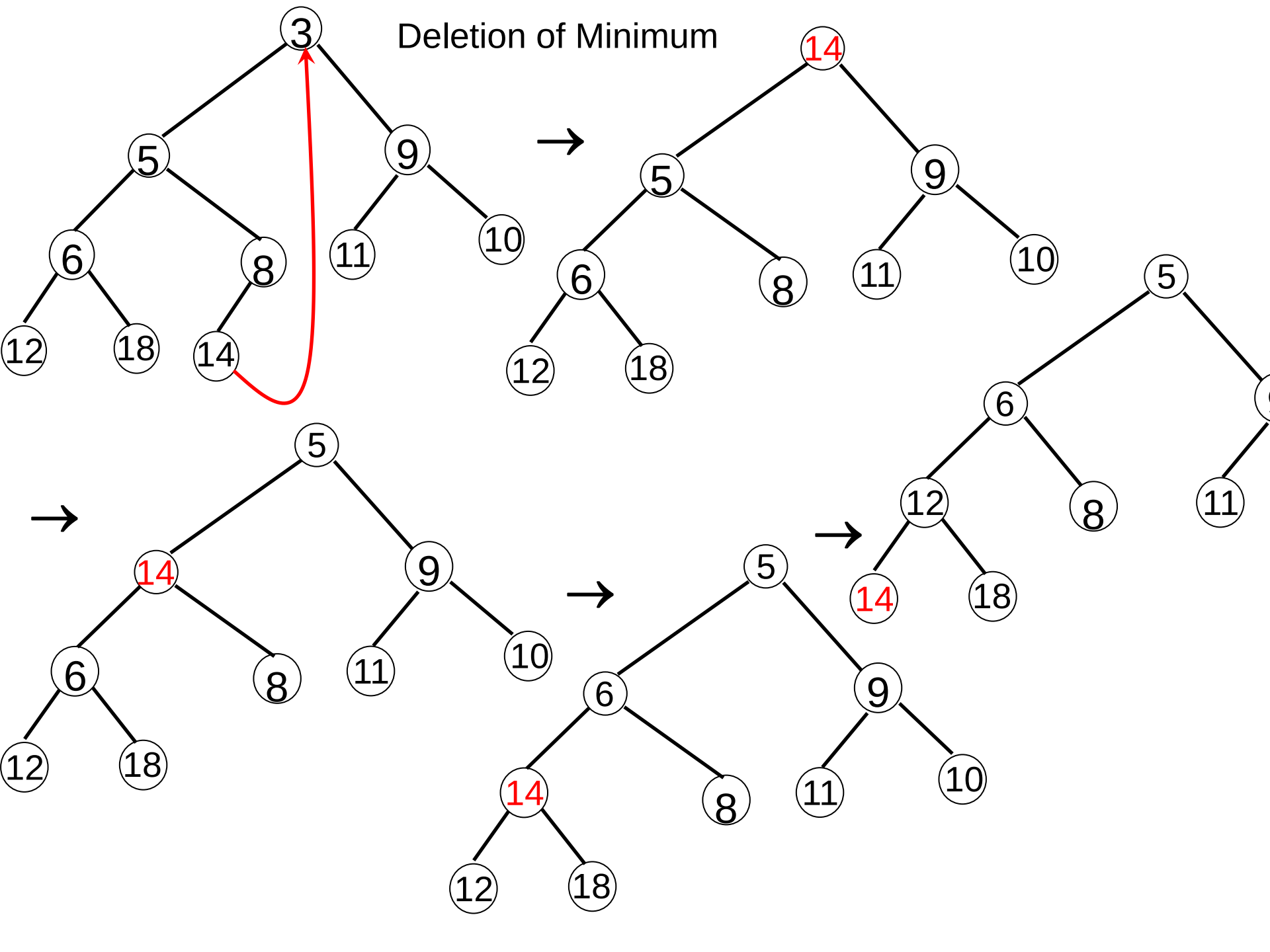
Here is a partially ordered tree



and here is the corresponding **heap**

1	2	3	4	5	6	7	8	9	10	11	12
20	27	32	29	30	51	45	45	38	50	85	67


Insertion of 4
3
5
9
6
8
11
10
12
18
14
4
3
5
9
6
4
11
10
12
18
14
8
3
4
9
6
5
11
10
12
18
14
8


Deletion of Minimum
3
5
9
6
8
11
10
12
18
14
14
5
9
6
8
11
10
12
18
5
14
9
6
8
11
10
12
18
5
6
12
8
11
14
18
5
6
9
14
8
11
10
12
18

Following to DT 03

Exercise 11 : Program insertion and deletion operations in a Heap