Trees. Binary trees

DS 2018/2019

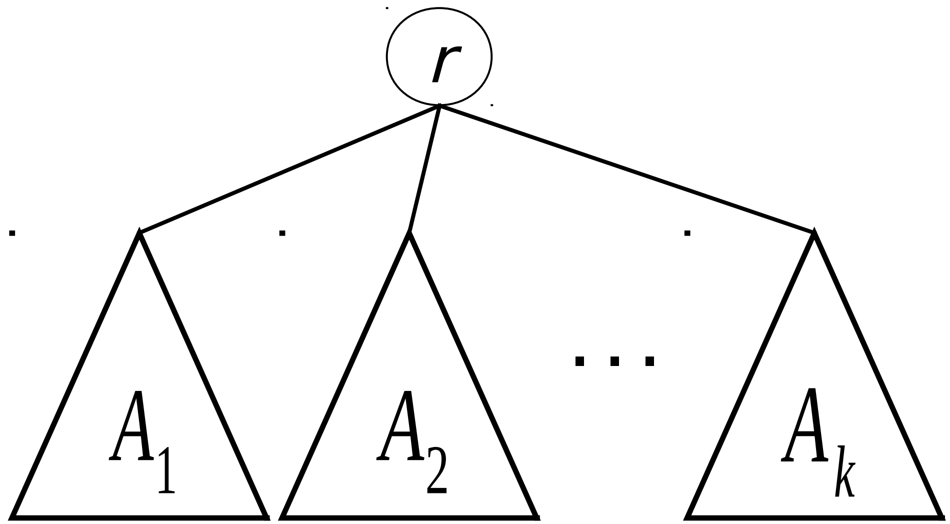
Content

- Trees
- Binary Trees (**`BinTree`**)
- Application: arithmetic expression reprezentation

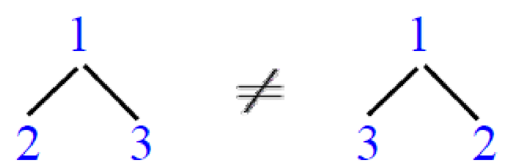
Trees: recursive definition

$$A=\left\{\begin{array}{l} \Lambda,\ \ \text{empty tree}, \\ \left(r,\left\{A_1,\ldots,A_k\right\}\right), r\ \ \text{element}, A_1,\ldots,A_k\ \ \text{trees} \end{array}\right\}$$

$A=\Lambda$ or



If *A is* ordered (planar), then



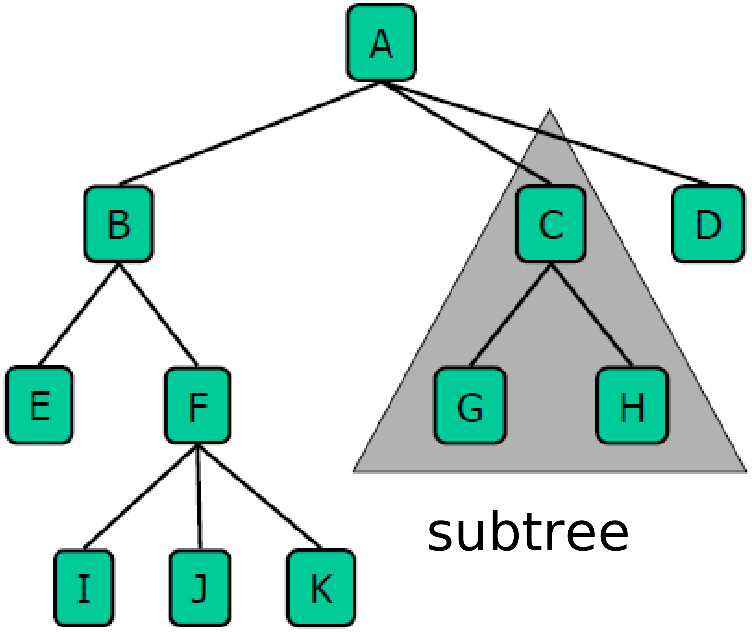
Trees: terminology

- Root: node without parent.

- Intern node: has at least one child.
- External node (leaf): node with no children.

- Descendants of a node: children, grand children, etc.
- Brothers of a node: all other nodes having the same parent.

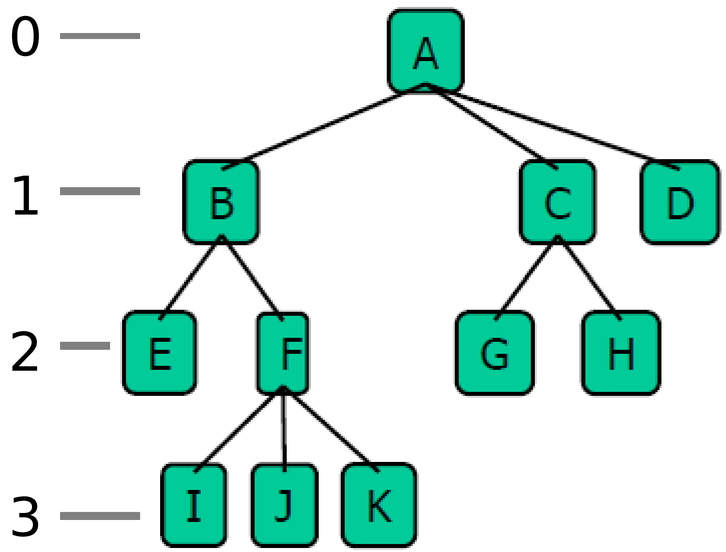
- Subtree: some node and all its descendants.



Trees: terminology

- Depth of some node *x*: number of nodes from the root to *x* (except *x*).

$$\operatorname{depth}(x)=\begin{cases} 0, & \text{if } x \text{ is the root} \\ 1+\operatorname{depth}(\operatorname{father}(x)), & \text{otherwise} \end{cases}$$



- Tree height: maximum depth of tree nodes.

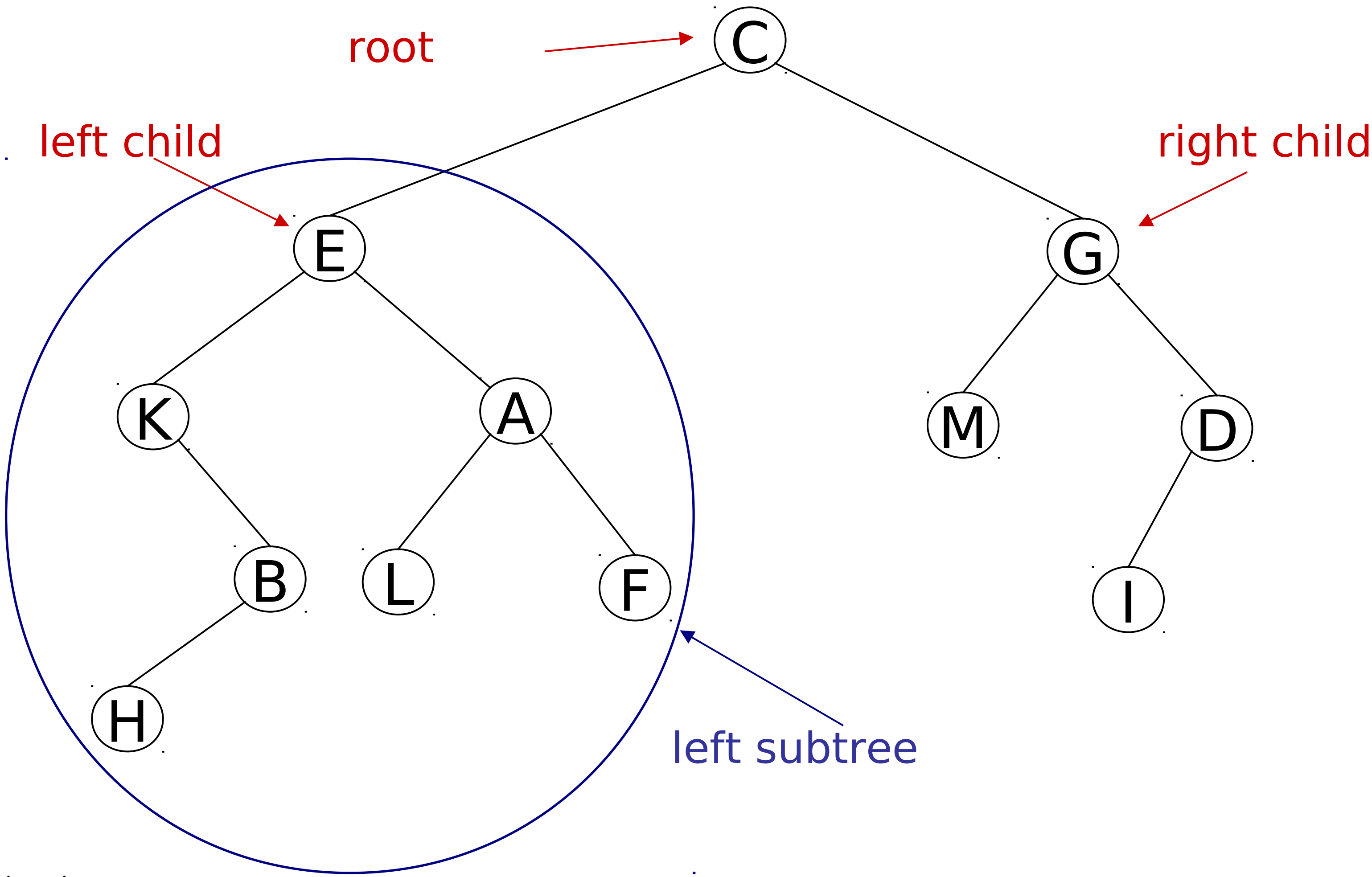
- Height of node *x*: distance from x to its most far descendant.

Abstract data type `BinTree`

- objects : Binary trees.
  - a binary tree is a node collection having the properties:

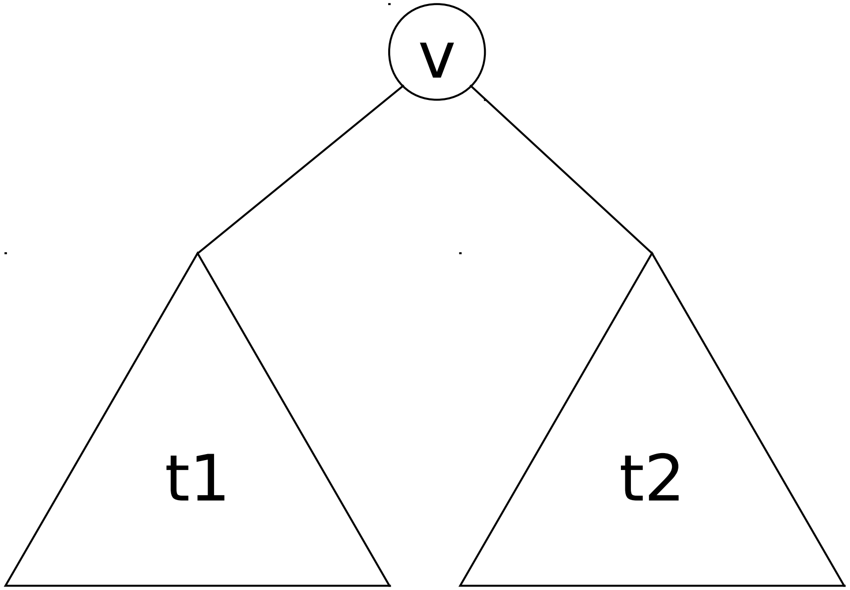
    1. any node has 0, 1 or 2 succesors (**children**);
    2. any node except one – the **root**, has a single predecessor (**parent);**
    3. The root has no predecessors;
    4. the children are ordered: left child, right child (if a node has single child, it has to be specified which one);
    5. the nodes without children gives the tree frontier.

Binary tree: example



Binary tree: recursive definition

- The empty tree is a binary tree.
- If **v** is a node and **t1**, **t2** are binary trees then the tree having **v** as root, **t1** the root left subtree and **t2** the root right subtree, is binary tree.



Binary trees: properties

- Notation
  - n number of nodes
  - n_e number of external nodes
  - n_i number of internal nodes
  - h height

$$h+1 \leq n \leq 2^{h+1}-1 \qquad 1 \leq n_e \leq 2^h$$

$$\log_2(n+1)-1 \leq h \leq n-1 \qquad h \leq n_i \leq 2^h-1$$

Binary trees: properties

- <u>Proper tree</u>: each internal node has exactly two children

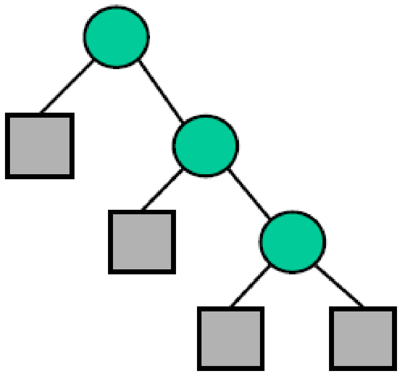
$$2h+1 \le n \le 2^{h+1}-1$$

$$\log_2(n+1)-1 \le h \le (n-1)/2$$

$$h+1 \le n_e \le 2^h$$

$$h \le n_i \le 2^h-1$$

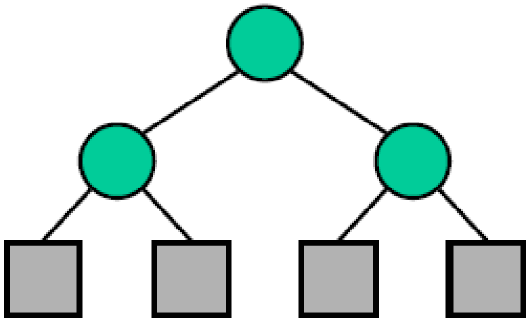
$$n_e = n_i + 1$$



- <u>Complet tree</u>: proper tree where the leaves have the same depth

$$\text{level } i \text{ has } 2^i \text{ nodes}$$

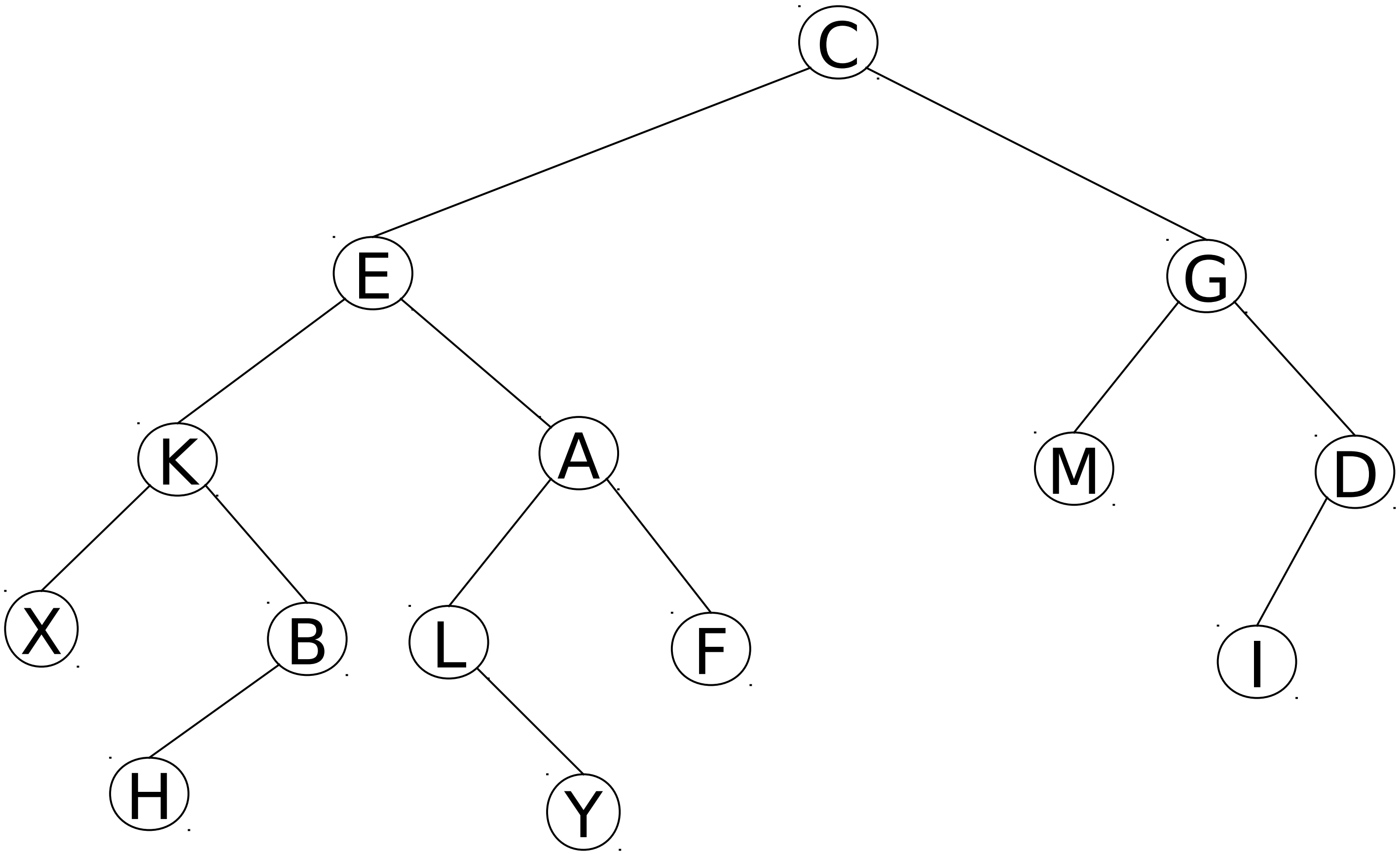
$$n = 2^{h+1}-1 = 2n_e - 1$$



`BinTree`: operations

- insert()
  - input:
    - a binary tree **t**
    - address f a node having at most one child (parent on the new node)
    - type of inserted child (left, right)
    - new node information **e**
  - output
    - tree where a new node that stores **e** has been added; the new node has no children

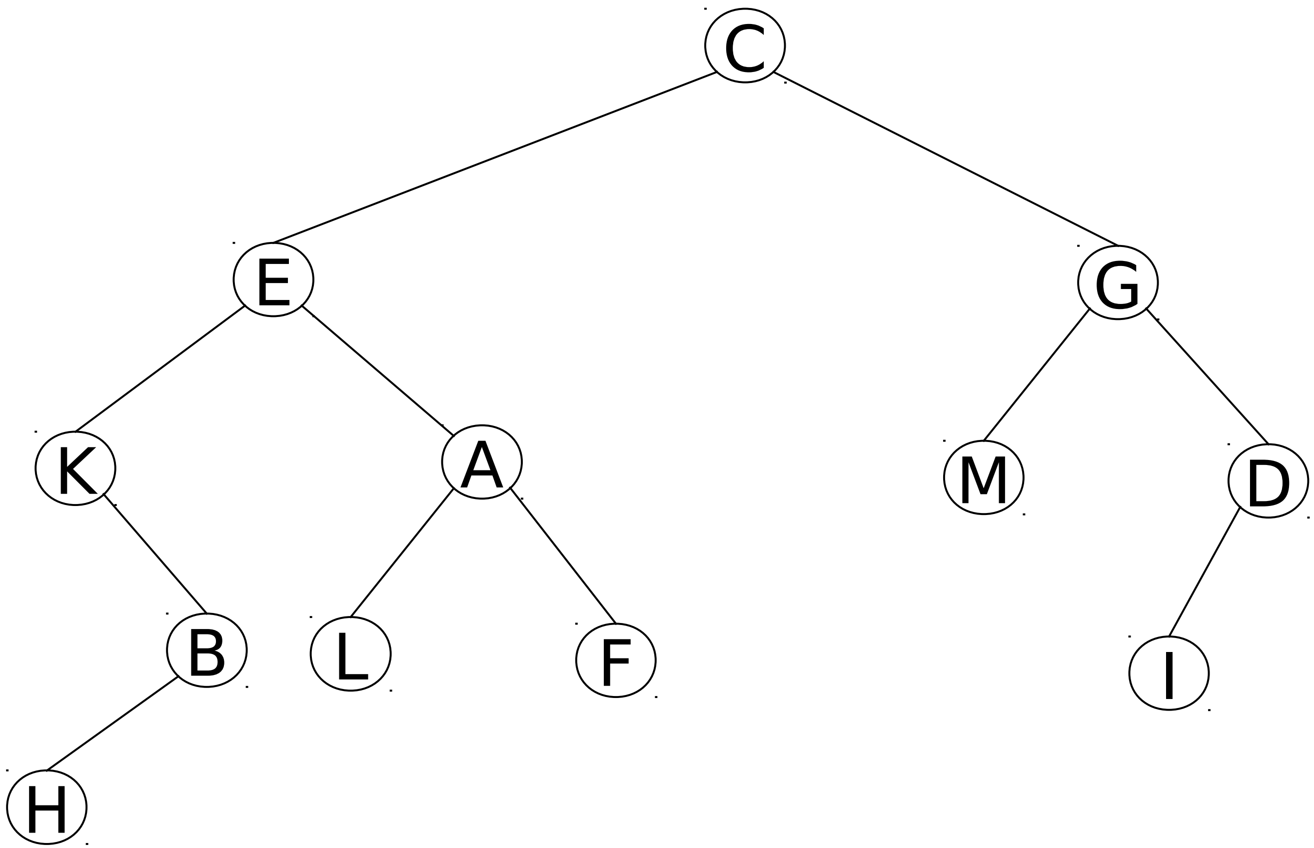
`BinTree`: insert



`BinTree`: delete

- delete()
  - input:
    - a binary tree **t**
    - Address of a leaf node and the address of its parent
  - output
    - Tree from which the given leaf node has been deleted

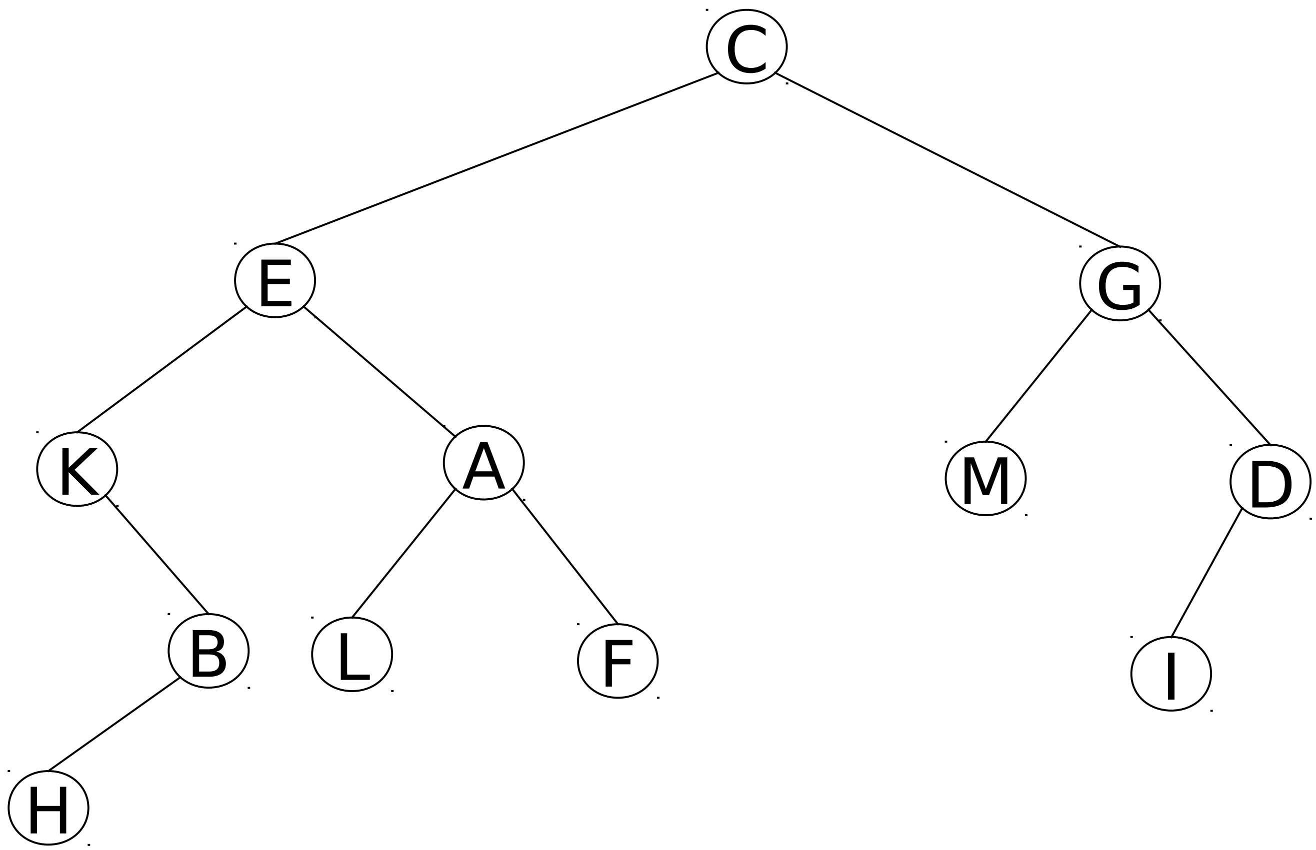
BinTree: delete



`BinTree`: preorder traversal

- preorder()
  - input
    - a binary tree **`t`**
    - a procedure **`visit`**( )
  - output
    - Binary tree **`t`** with ne nodes processed by **`visit()`** in the following order
      - Root (R)
      - Left subtree (S)
      - Right subtree (D)

Preorder traversal - example

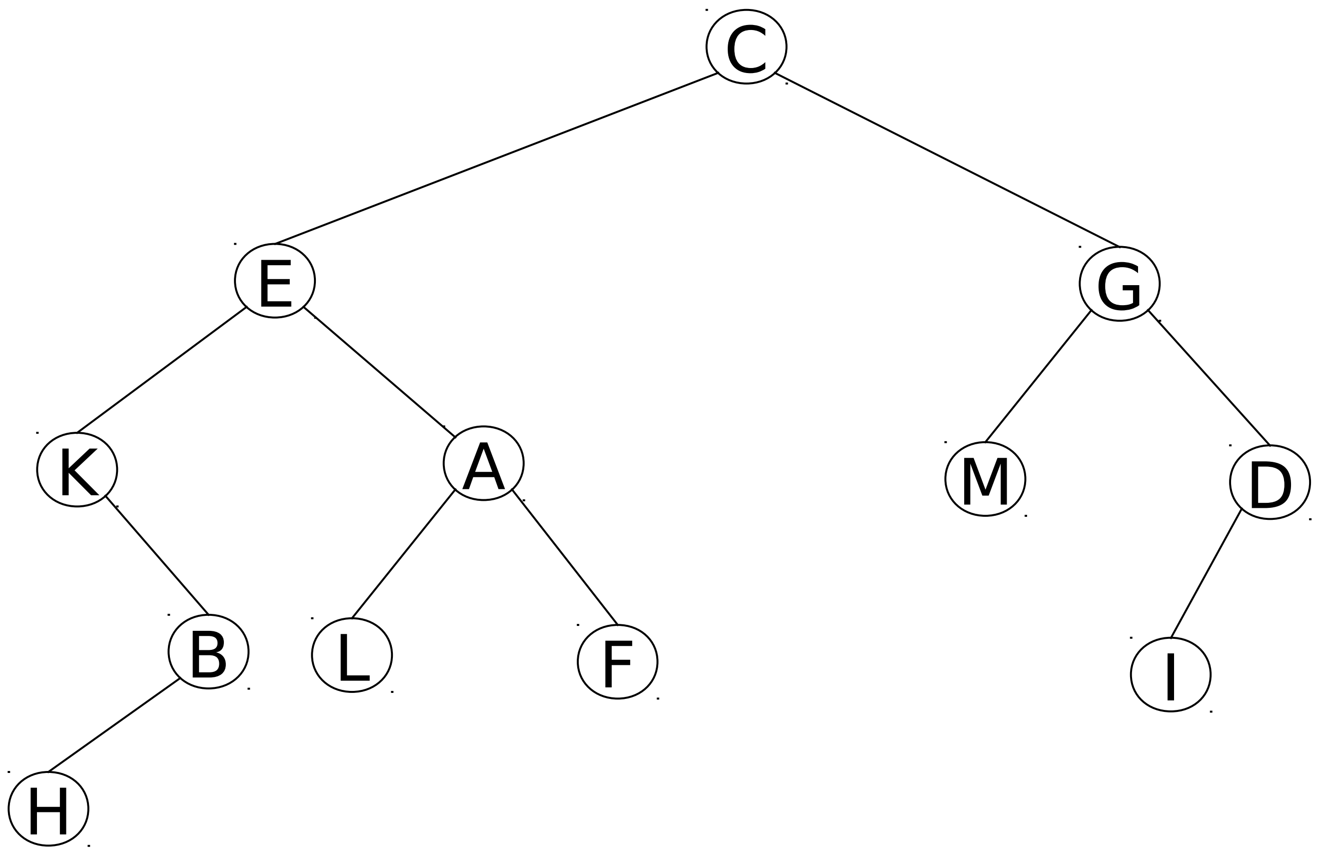


C, E, K, B, H, A, L, F, G, M, D, I

`BinTree:` inorder traversal

- inorder()
  - input
    - a binary tree **`t`**
    - a procedure **`visit()`**
  - output
    - binary tree **`t`** with nodes processed by **`visit()`** in order S R D

Inorder traversal - example

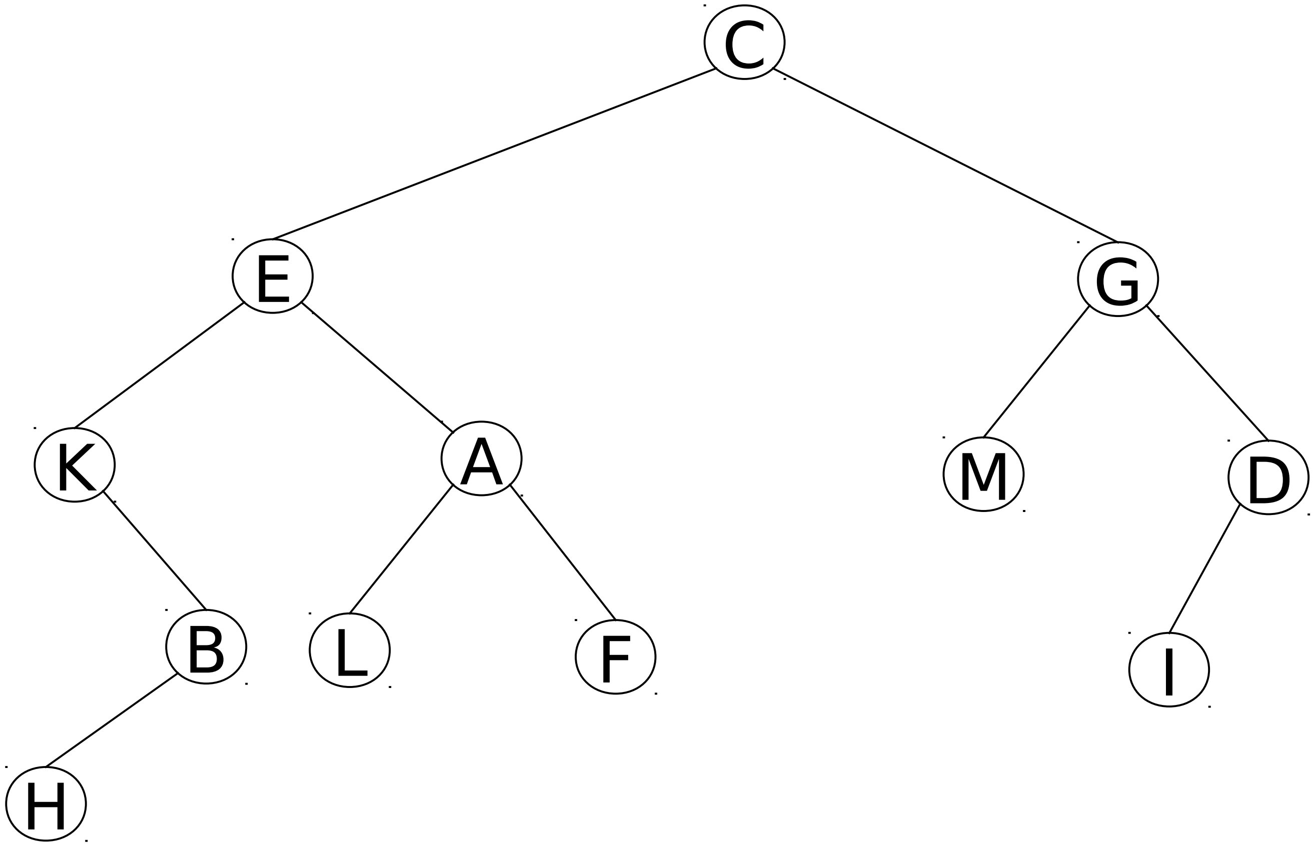


K, H, B, E, L, A, F, C, M, G, I, D

`BinTree`: postorder traversal

- postorder()
  - input
    - a binary tree **`t`**
    - a procedure **`visit()`**
  - output
    - binary tree **`t`** with nodes processed by **`visit()`**in order S D R

Postorder traversal - example

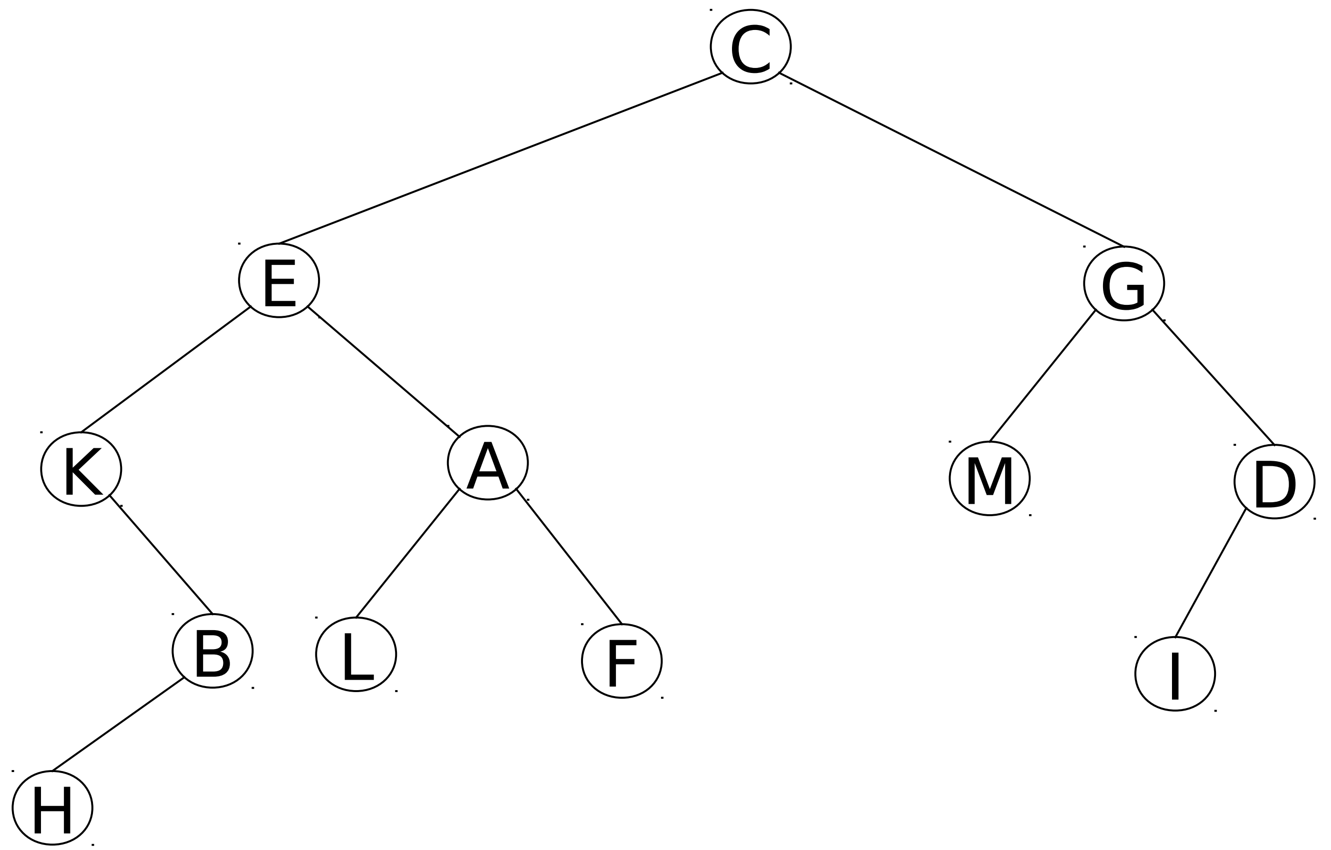


H, B, K, L, F, A, E, M, I, D, G, C

`BinTree`: BFS traversal

- BFS() - Breadth-First Search
  - input
    - a binary tree **`t`**
    - a procedure **`visit()`**
  - output
    - binary tree **`t`** with nodes processed by **`visit()`** in BFS order (level by level)

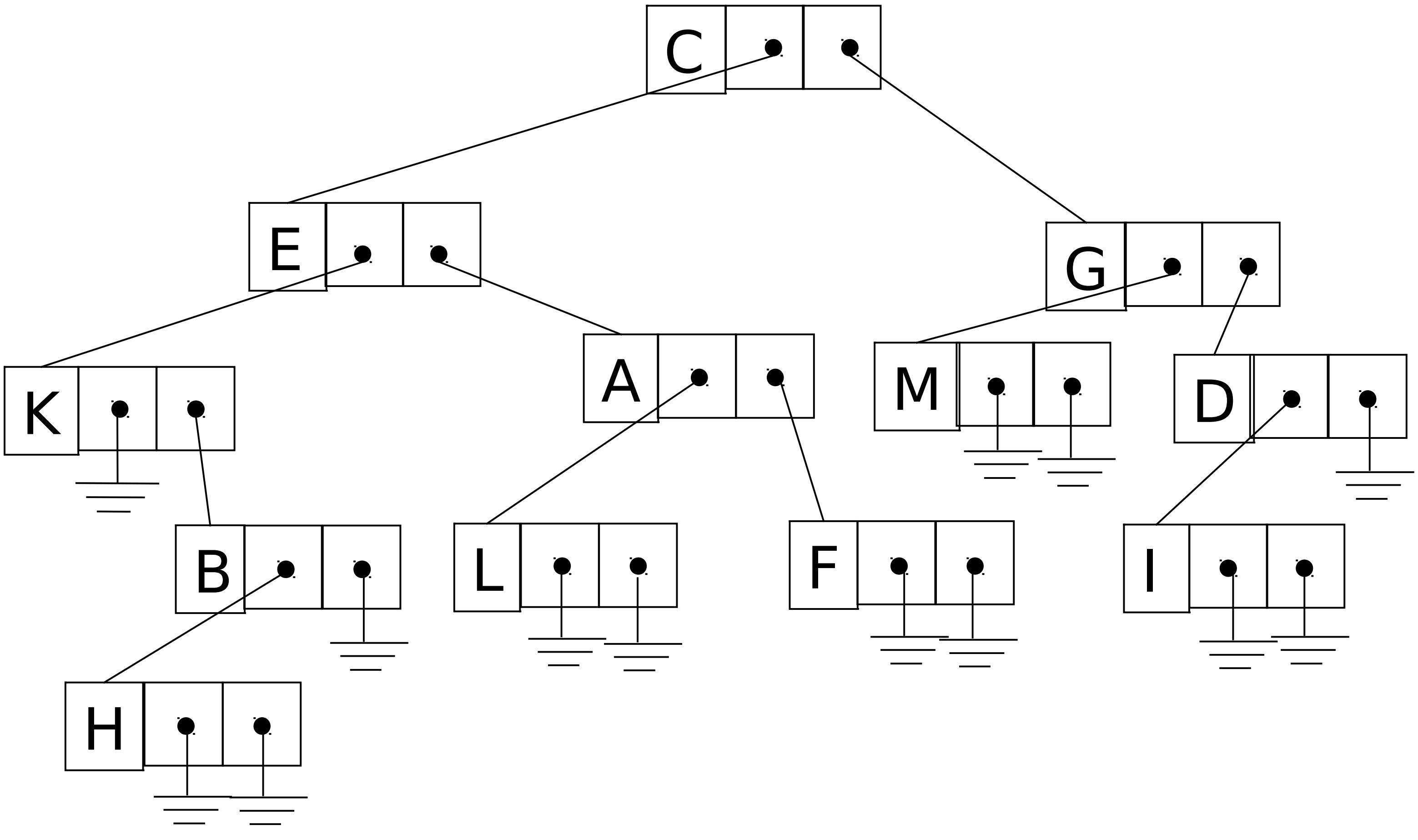
BFS traversal - example



C, E, G, K, A, M, D, B, L, F, I, H

BinTree: linked list implementation

- Object reprezentation



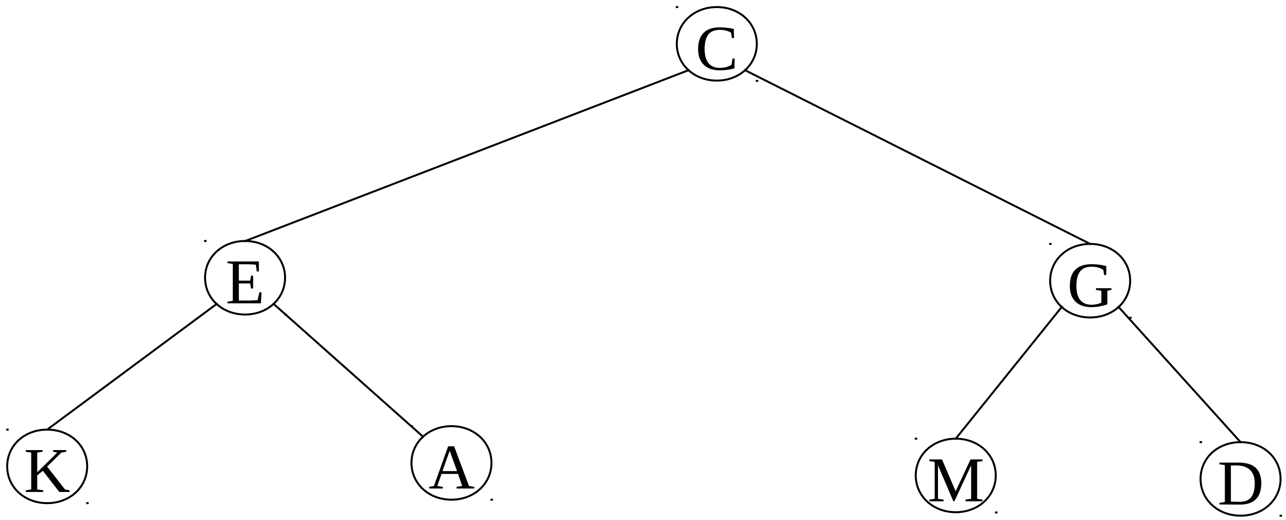
`BinTree`: node structure

- a node **`v`** (stored at address v) is a structure with three fields:
  - **`v->inf`** /*information stored in node*/
  - **`v->left`** /*left child address*/
  - **`v->right`** /*right child address*/

BinTree: preorder()

```
procedure preorder(v, visit)
begin
  if (v == NULL)
   then return
  else
   visit(v)
   preorder(v->left, visit)
   preorder(v->right, visit)
end
```

BFS traversal implementation



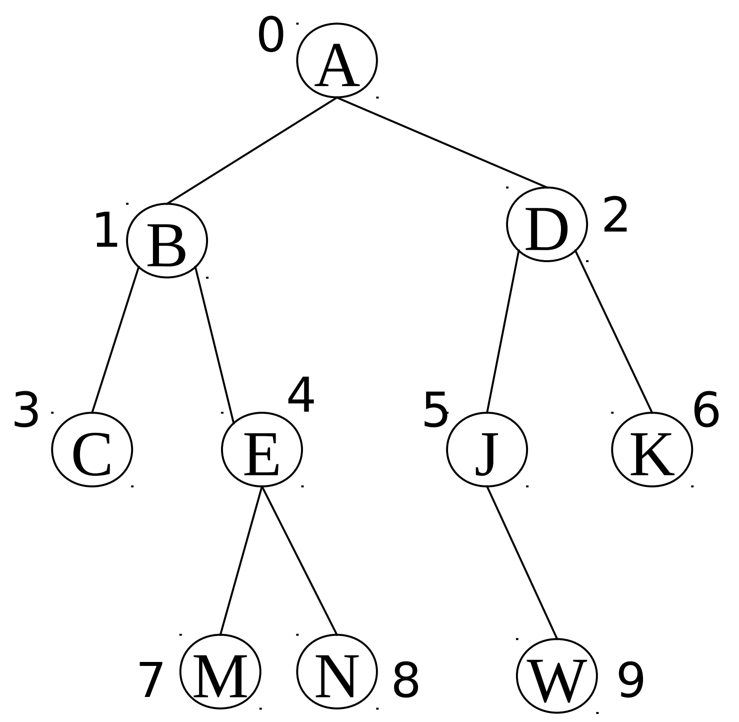
Queue = (C E G K A M D)

BFS traversal implementation

```
procedure BFS(t, visit)
begin
   if (t == NULL) then return
   else
      Queue ← emptyQueue()
      insert(Queue, t)
      while (not isEmpty(Queue)) do
         read(Queue, v); visit(v)
         if (v->left != NULL)
            then insert(Queue, v->left)
         if (v->right != NULL)
            then insert(Queue, v->right)
         delete(Queue)
end
```

`BinTree`: list implementation

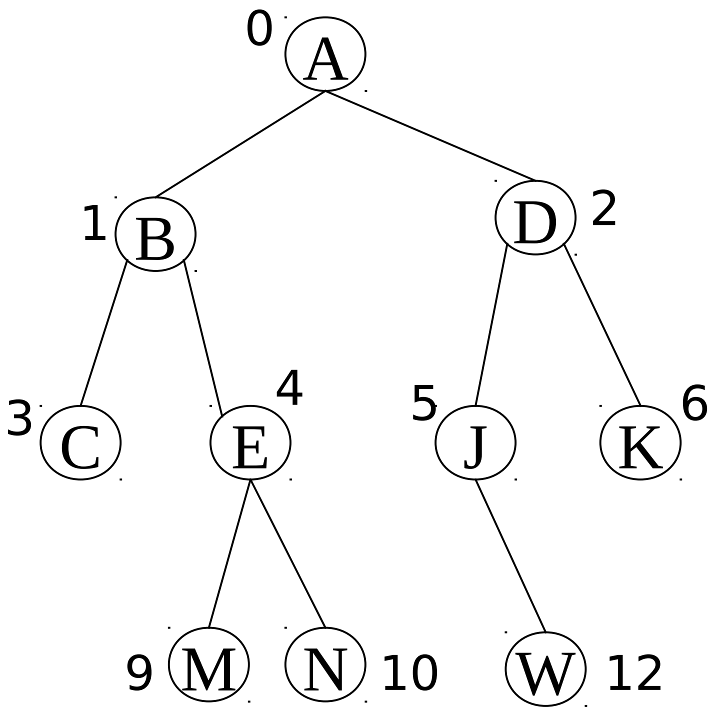
- "parent" relation reprezentation: <u>parent array</u>
- **Advantages**:
  - Simplicity;
  - Easy access from any node to the root;
  - Memory saving.
- **Disadvantages**:
  - Non-easy access from the root to some node.



-1	**0**	**0**	**1**	**1**	**2**	**2**	**4**	**4**	**5**
0	1	2	3	4	5	6	7	8	9

`BinTree`: array implementation

- Nodes are stored in an array.
- Node index:
  - index(root) = 0
  - index(x) = 2*index(parent(x))+1, if x is left child
  - index(x) = 2*index(parent(x))+2, if x is right child



A	B	D	C	E	J	K			M	N		w		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Application: integer expression

- Integer expressions
  - definition;
  - examples.

- Tree representation of expressions:
  - definition similarities;
  - expression associated tree;
  - prefix, infix and postfix notation and tree traversal.

Integer expression definition

<int> ::= ... –2 | -1 | 0 | 1 | 2 ...

<bin_op> ::= + | - | * | / | %

<int_exp> ::= <int>

| <int_exp> <bin_op> <int_exp>

| (<int_exp>)

- priorities

  `12-5*2` is **`(12-5)*2`** or **`12-(5*2)?`**

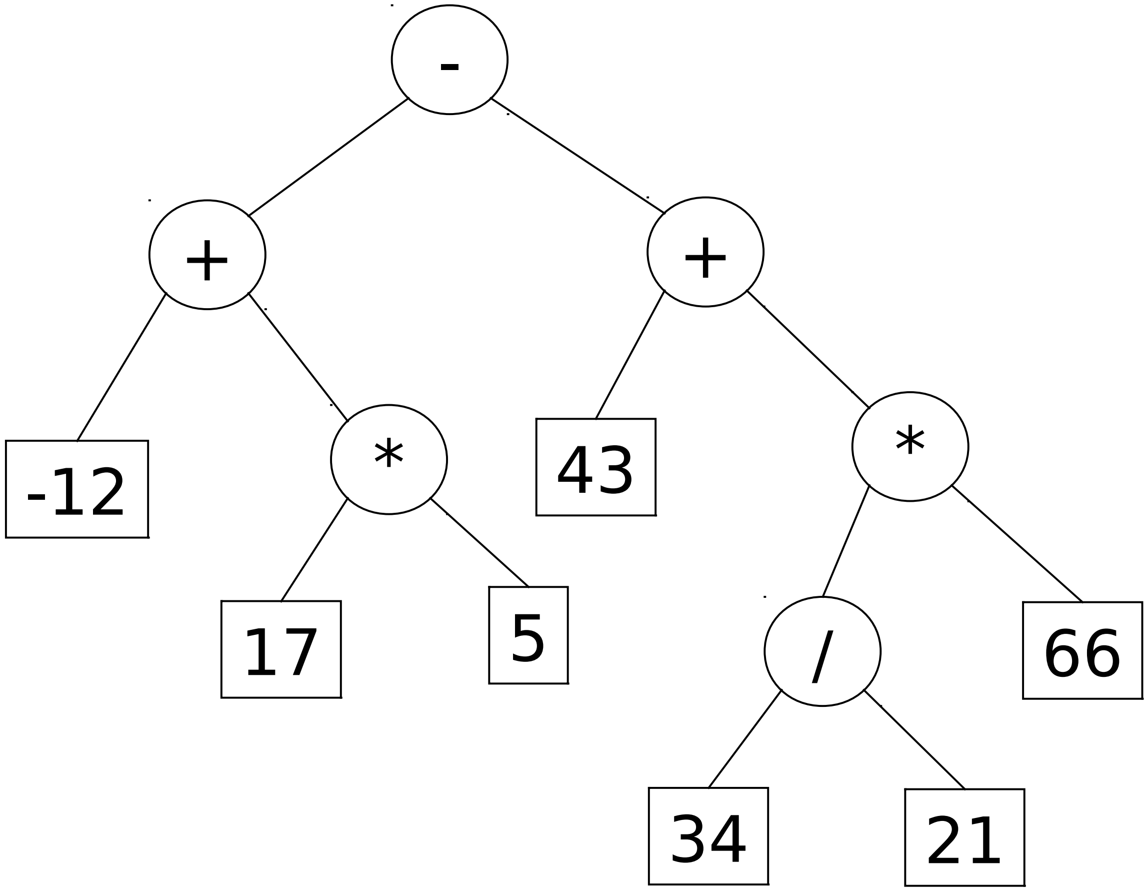
- association rules

  `15/4/2` is **`(15/4)/2`** or **`15/(4/2)?`**

  `15/4*2` is **`(15/4)*2`** or **`15/(4*2)?`**

Expressions as trees

`-12 + 17 * 5 – (43 + 34 / 21 * 66)`



Postfix and prefix notations

- postfix notation is given by the postorder traversal

  -12, 17, 5, *, +, 43, 34, 21, /, 66, *, +, -

- Prefix notation is givenby the preorder traversal

  -, +, -12, *, 17, 5, +, 43, *, /, 34, 21, 66