Trees. Binary trees

Mădălina Răschip, Cristian Gațu

Faculty of Computer Science
"Alexandru Ioan Cuza" University of Iași, Romania

DS 2020/2021

Content

Trees

Binary tree (`BinTree`)

Application: integer expression representation as trees

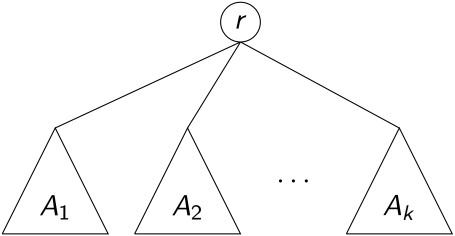
Trees: recursive definition

$$A = \begin{cases} \Lambda - \text{empty tree}, \\ (r, \{A_1, \cdots, A_k\}), \quad r \text{ an element and } A_1, \cdots, A_k \text{ trees.} \end{cases}$$

Trees: recursive definition

$$A = \begin{cases} \Lambda - \text{empty tree}, \\ (r, \{A_1, \cdots, A_k\}), \quad r \text{ an element and } A_1, \cdots, A_k \text{ trees.} \end{cases}$$

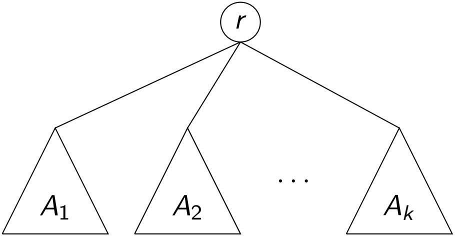
$A = \Lambda$ or



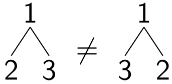
Trees: recursive definition

$$A = \begin{cases} \Lambda - \text{empty tree}, \\ (r, \{A_1, \cdots, A_k\}), \quad r \text{ an element and } A_1, \cdots, A_k \text{ trees.} \end{cases}$$

$A = \Lambda$ or

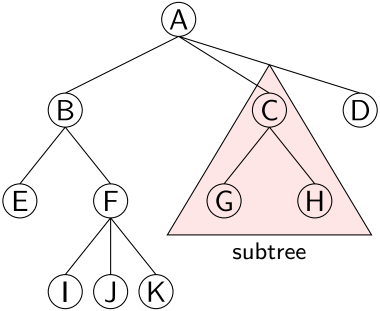


If A is *ordered (planar)*, then



Trees: terminology

- root: node without parent.
- internal node: has at least one child.
- external node (leaf): node with no children.
- descendants of a node: children, grand children, etc.
- siblings: all other nodes having the same parent.
- subtree: some node and all its descendants.

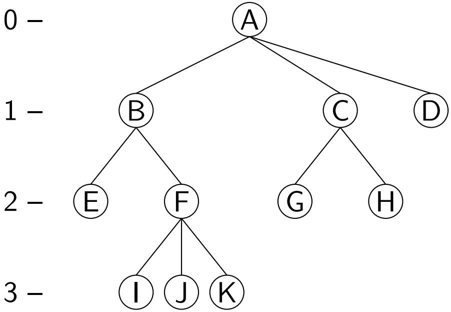


Trees: terminology

- Depth of some node x:

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

- tree height: maximum depth of tree nodes.
- height of node x: distance from x to its most far descendant.



Content

Trees

Binary tree (`BinTree`)

Application: integer expression representation as trees

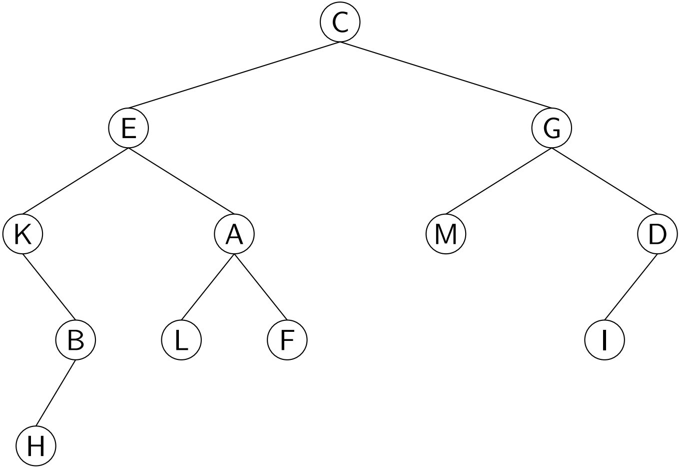
Abstract data type BinTree

OBJECTS: binary trees.

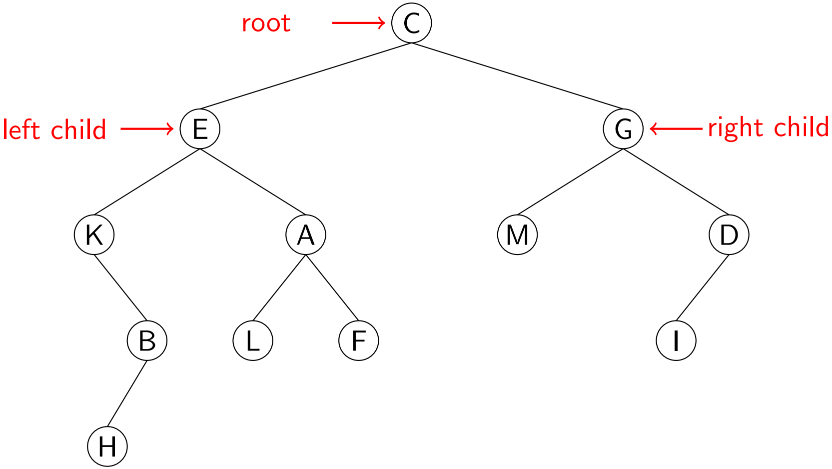
A binary tree is a node collection having the properties:

- any node has 0, 1 or 2 successors (children).
- any node except one — the root — has a single predecessor (parent).
- the root has no predecessors.
- the children are ordered: left child, right child (if a node has single child, it has to be specified which one);
- the nodes without children give the tree frontier.

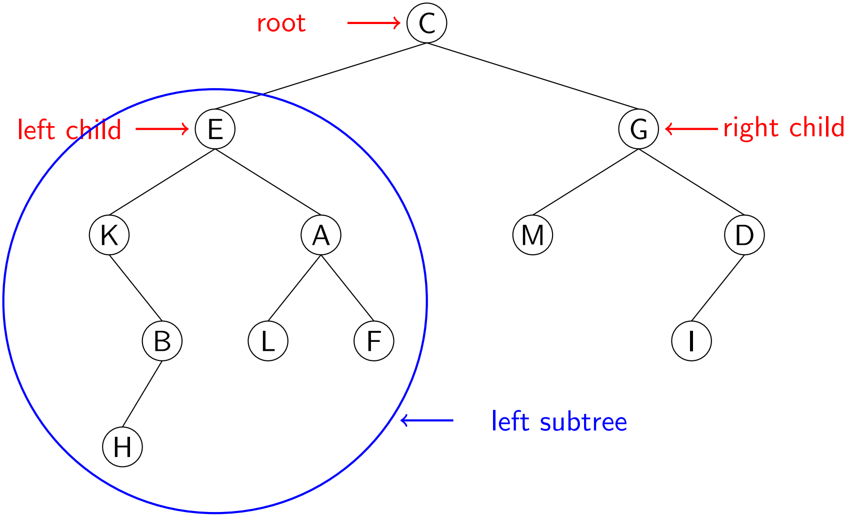
Binary tree: example



Binary tree: example

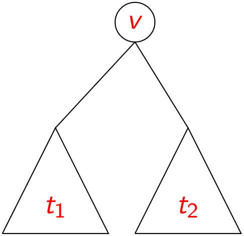


Binary tree: example



Binary tree: recursive definition

- The empty tree is a binary tree.
- If v is a node and t_1 and t_2 are binary trees then the tree having v as root, t_1 the root left subtree and t_2 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 – tree height.

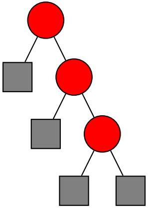
$$h + 1 \le n \le 2^{h+1} - 1; \qquad \log_2(n+1) - 1 \le h \le n - 1$$

$$1 \le n_e \le 2^h; \qquad h \le n_i \le 2^h - 1$$

Binary trees: properties

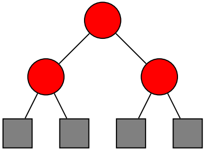
- Proper tree: each internal node has exactly two children.

$$2h + 1 \leq n \leq 2^{h+1} - 1;$$
$$\log_2(n+1) - 1 \leq h \leq (n-1)/2$$
$$h + 1 \leq n_e \leq 2^h;$$
$$h \leq n_i \leq 2^h - 1$$
$$n_e = n_i + 1$$



- Complete tree: proper tree where the leaves have the same depth.

level i has 2^i nodes;
$n = 2^{h+1} - 1 = 2n_e - 1$

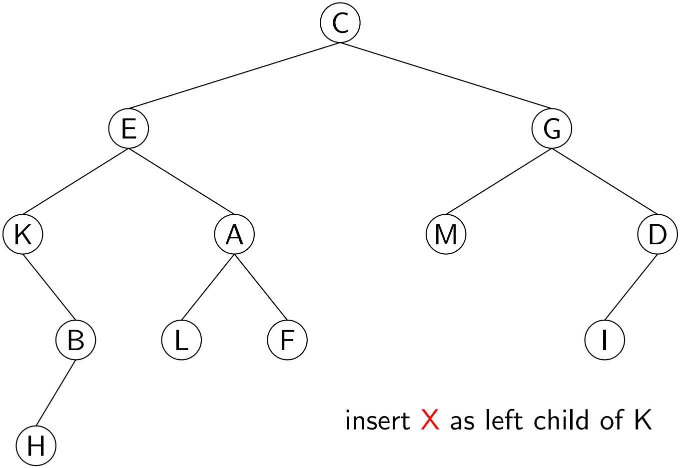


BinTree – operations

`insert()`

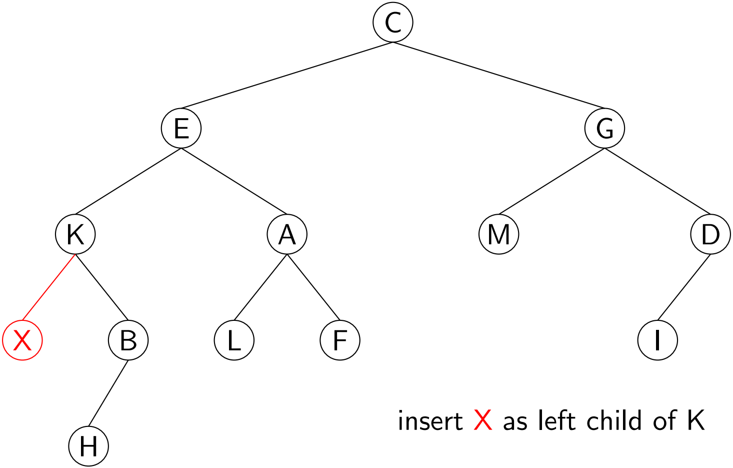
- input:
  - a binary tree `t`;
  - [address of] a node having at most one child (parent on the new node);
  - type of inserted child (left, right);
  - new node information `e`.
- output:
  - tree `t` where a new node that stores `e` has been added;
    the new node has no children.

BinTree: insert - example



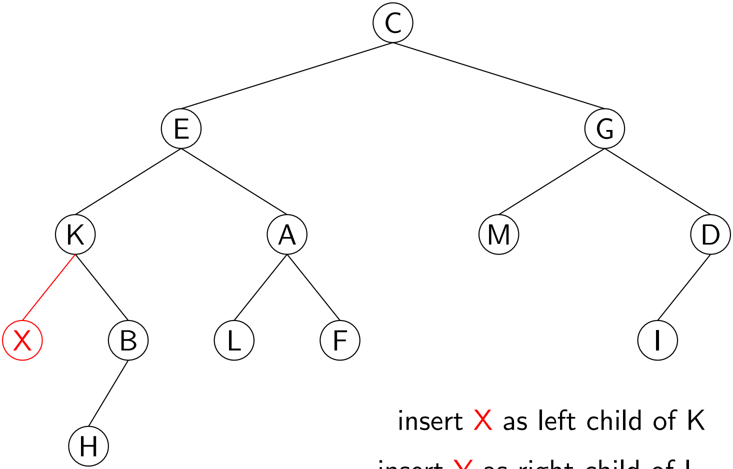
insert X as left child of K

BinTree: insert - example



insert X as left child of K

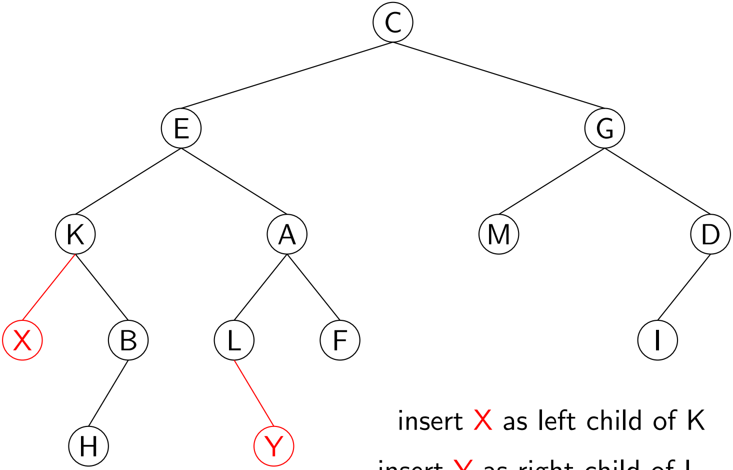
BinTree: insert - example



insert X as left child of K

insert Y as right child of L

BinTree: insert - example



insert X as left child of K

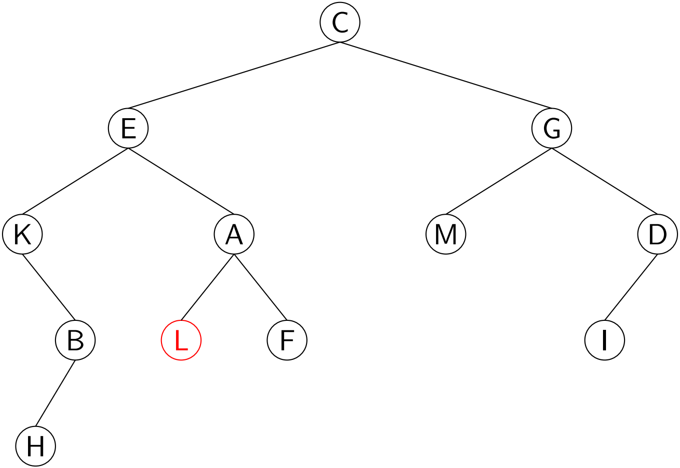
insert Y as right child of L

BinTree – operation

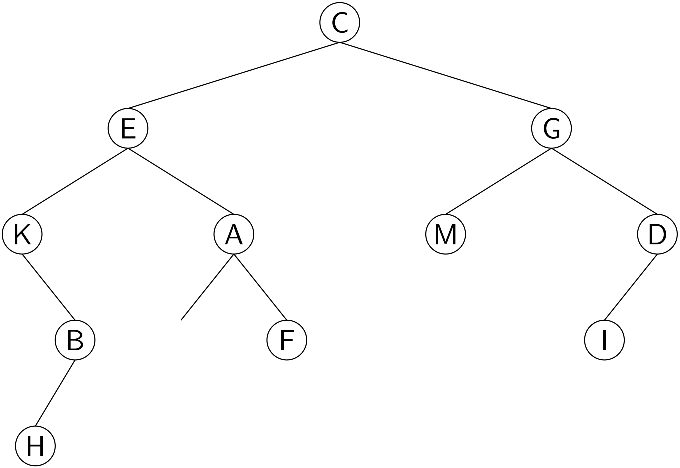
`delete()`

- input:
  - a binary tree `t`;
  - [address of] a leaf node and [address of] its parent.
- output:
  - tree `t` from which the given leaf node has been deleted (from the frontier).

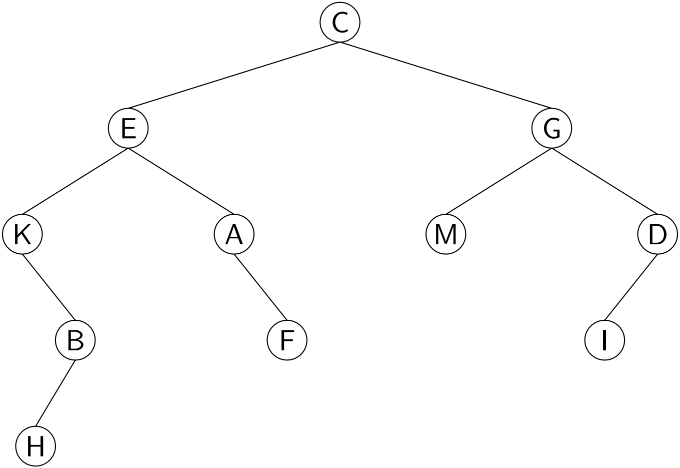
BinTree: delete - example



BinTree: delete - example



BinTree: delete - example



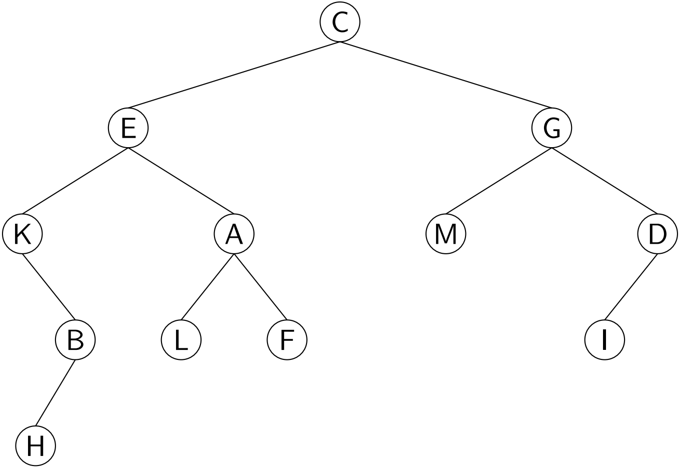
BinTree – preorder traversal

`preorder()`

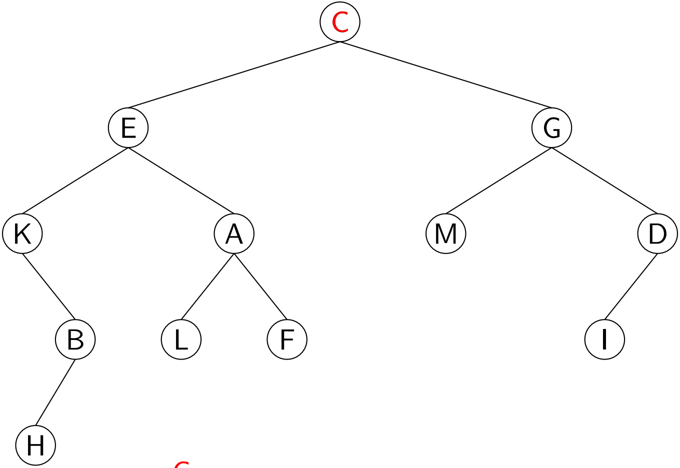
- input:
  - a binary tree `t`;
  - a procedure `visit()`.

- output:
  - binary tree `t`, with the nodes processed by `visit()` in the following order
    - (R) – root
    - (S) – left subtree
    - (D) – right subtree

Preorder traversal - example

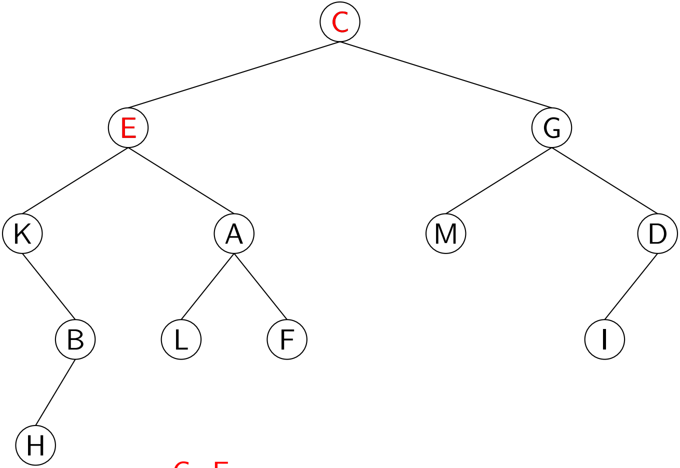


Preorder traversal - example



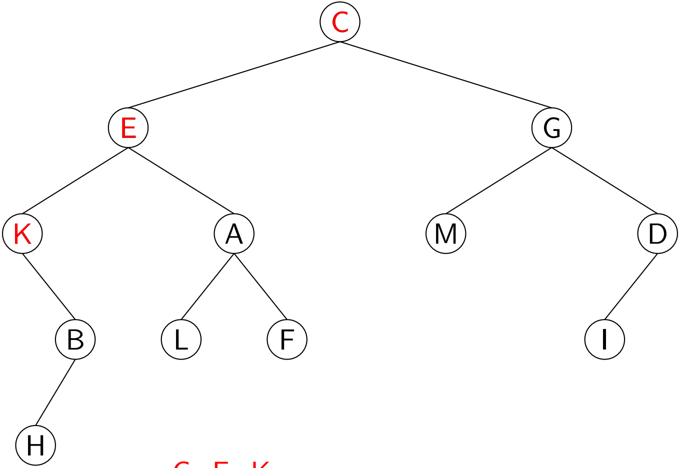
C

Preorder traversal - example



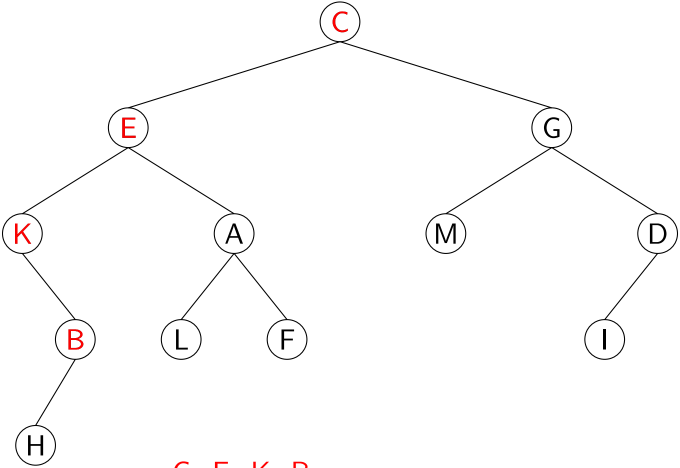
C E

Preorder traversal - example



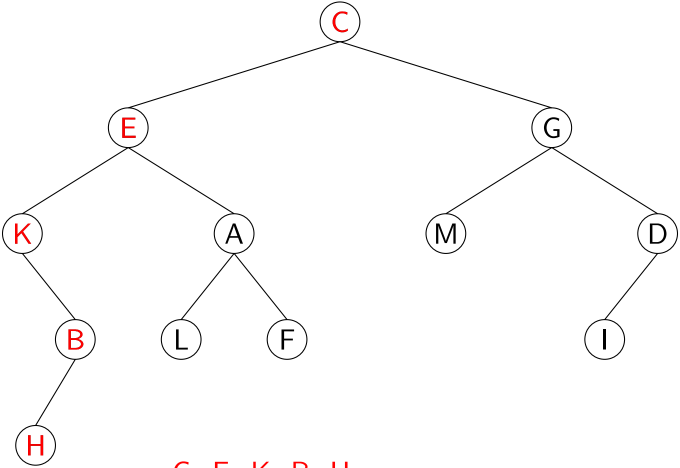
C E K

Preorder traversal - example



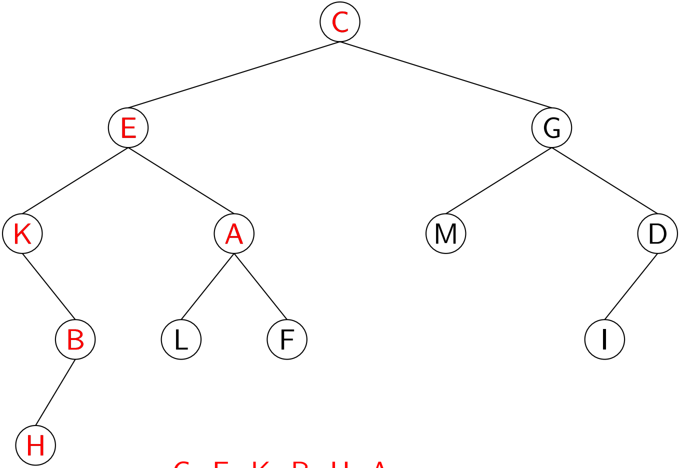
C E K B

Preorder traversal - example



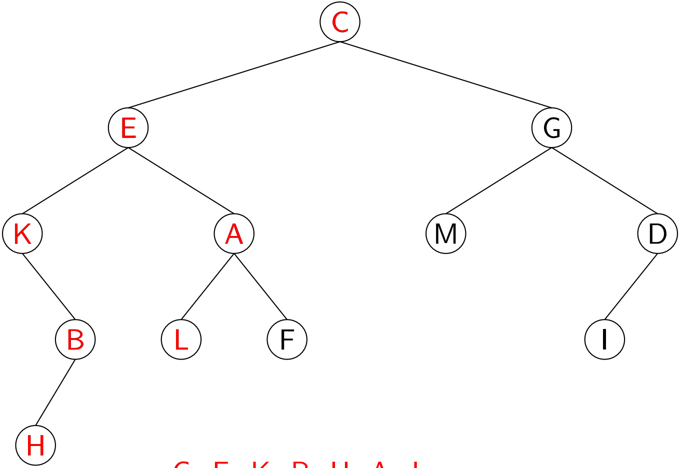
C E K B H

Preorder traversal - example



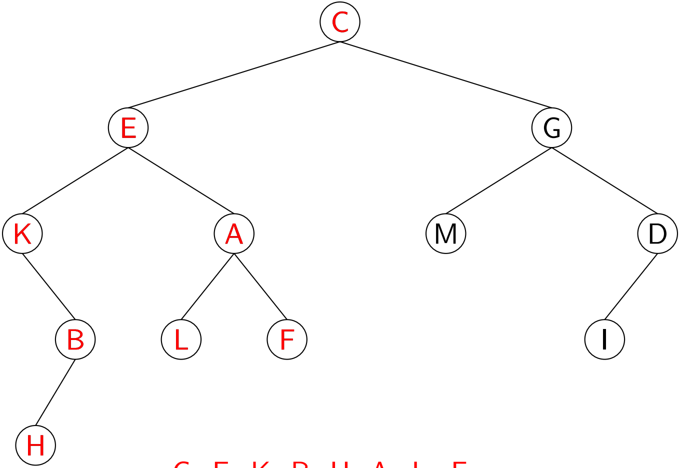
C E K B H A

Preorder traversal - example



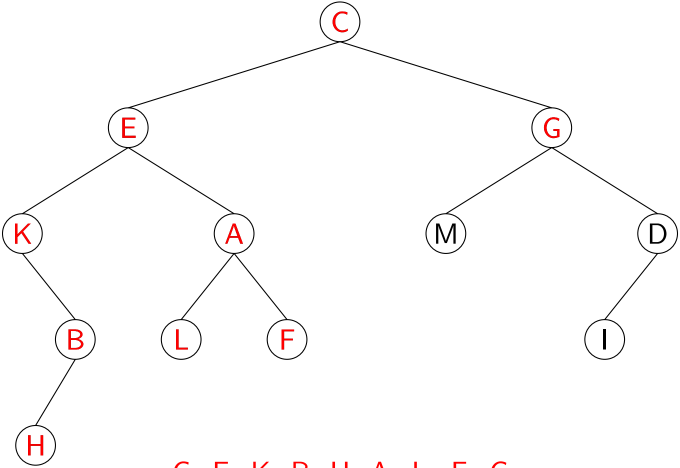
C E K B H A L

Preorder traversal - example



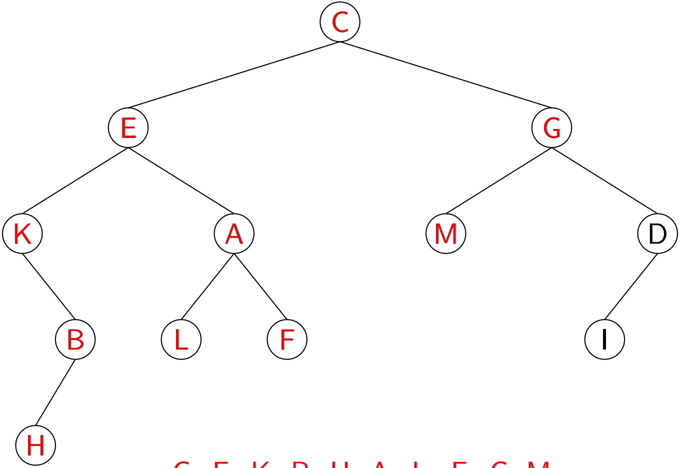
C E K B H A L F

Preorder traversal - example



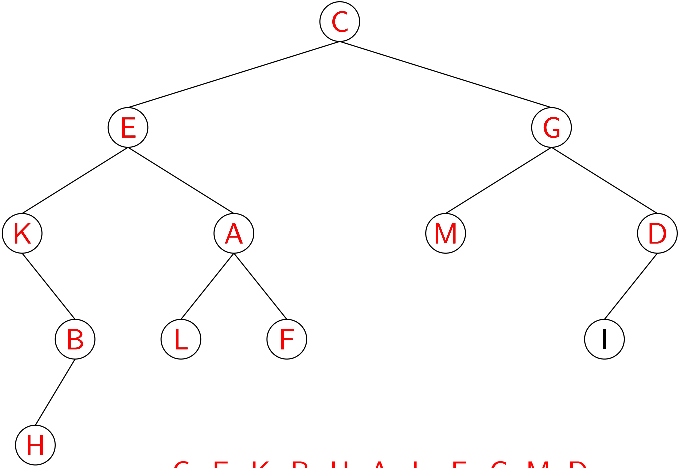
C E K B H A L F G

Preorder traversal - example



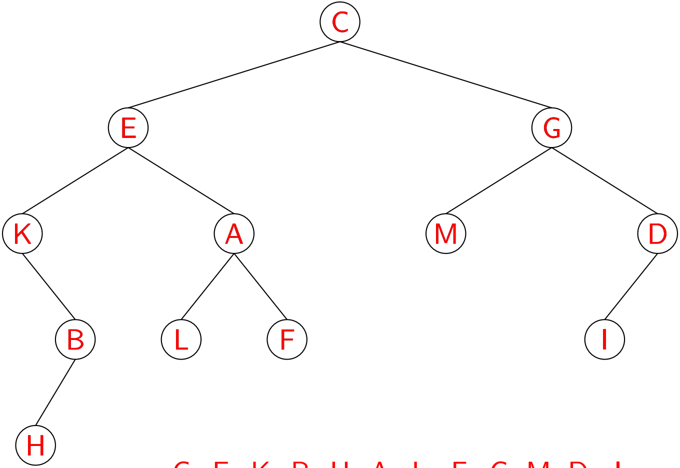
C E K B H A L F G M

Preorder traversal - example



C E K B H A L F G M D

Preorder traversal - example



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

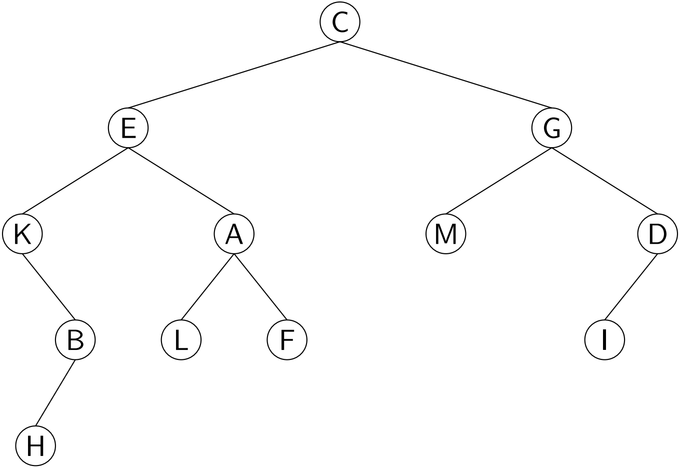
BinTree – inorder traversal

`inorder()`

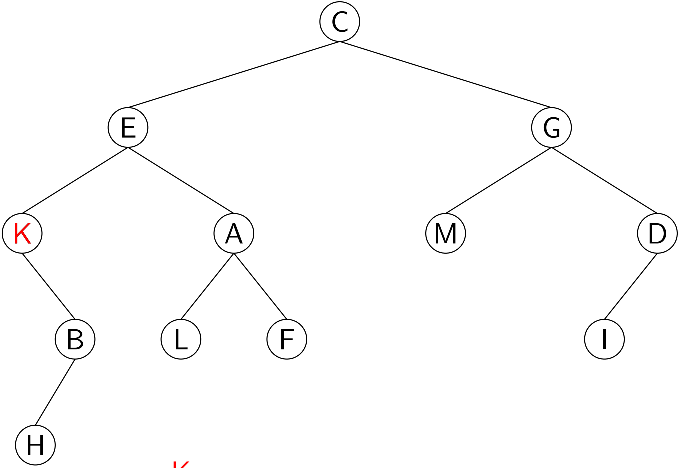
- input:
  - a binary tree `t`;
  - a procedure `visit()`.

- output:
  - binary tree `t` with nodes processed by `viziteaza()` in the following order
    - (S) – left subtree
    - (R) – root
    - (D) – right subtree

Inorder traversal – example

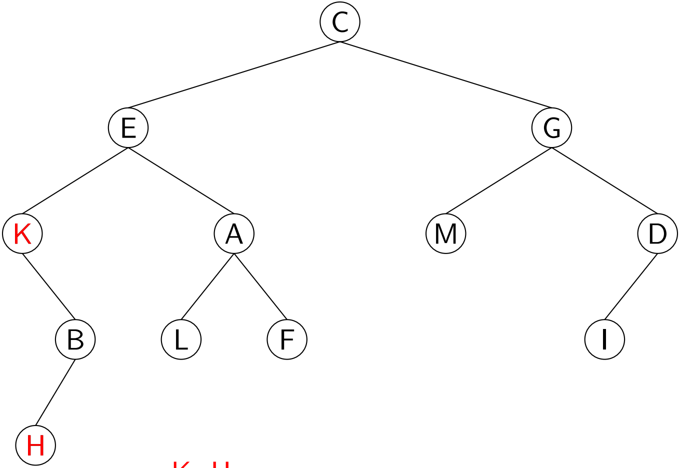


Inorder traversal – example



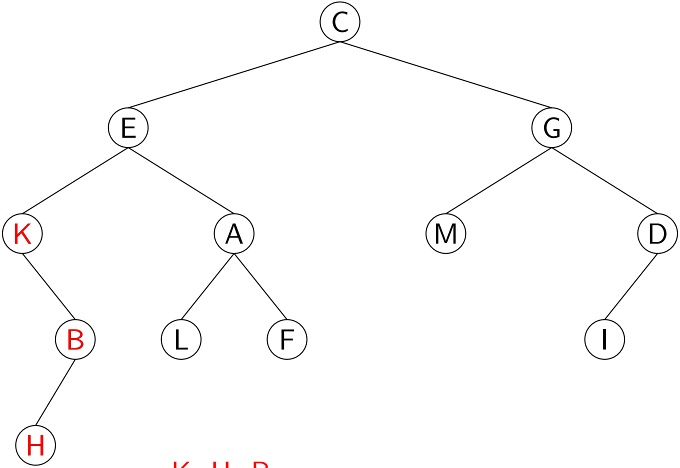
K

Inorder traversal – example



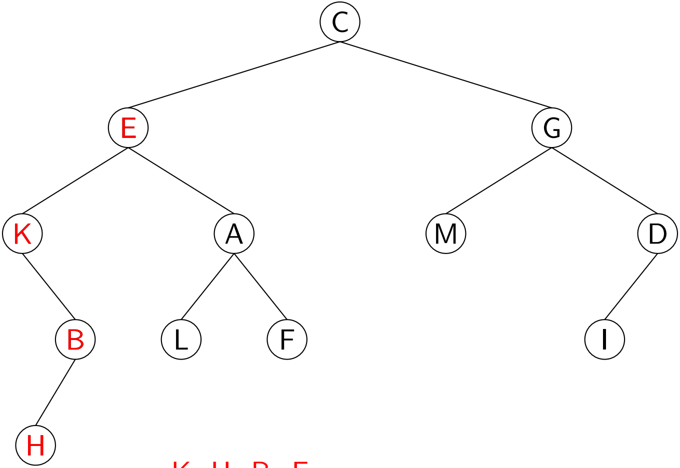
K H

Inorder traversal – example



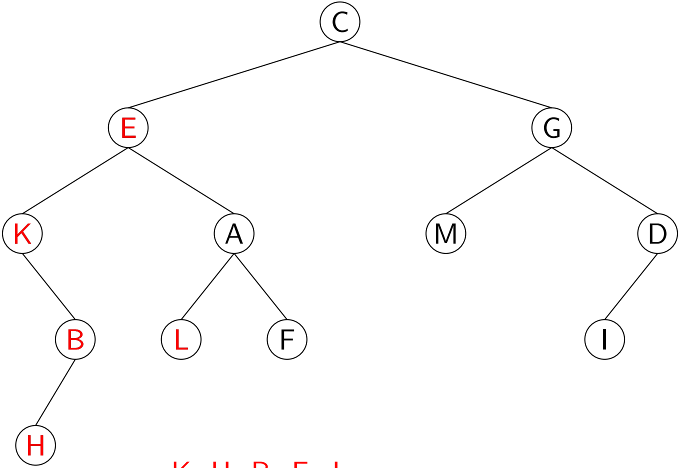
K H B

Inorder traversal – example



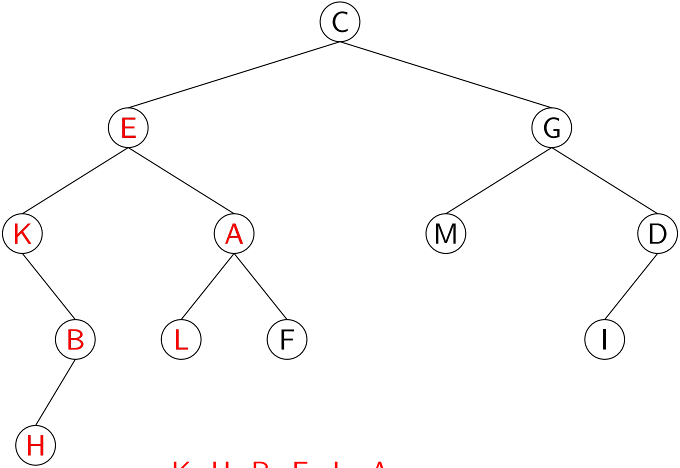
K H B E

Inorder traversal – example



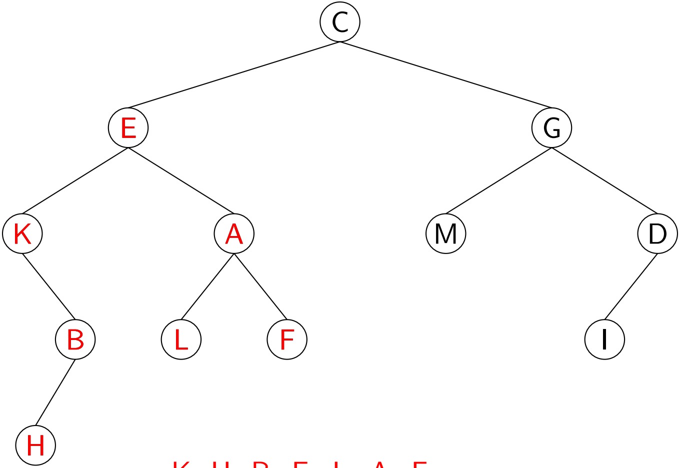
K H B E L

Inorder traversal – example



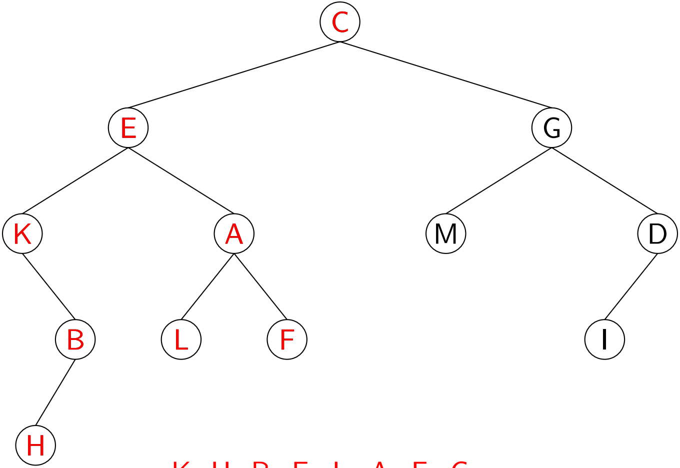
K H B E L A

Inorder traversal – example



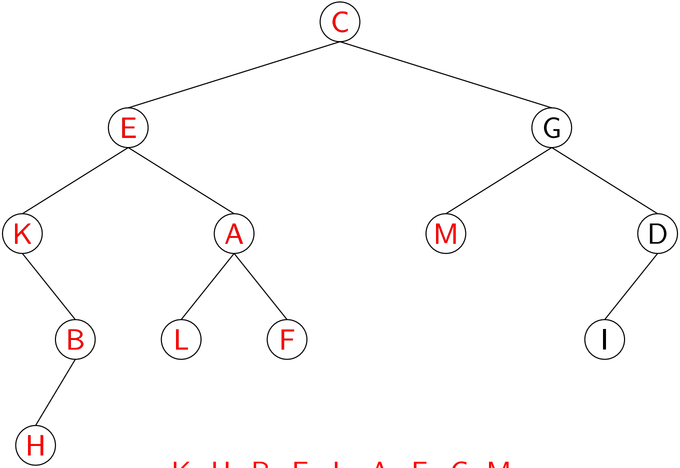
K H B E L A F

Inorder traversal – example



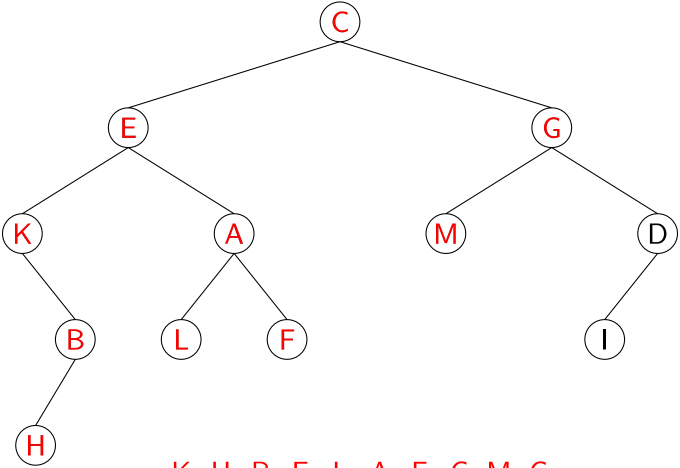
K H B E L A F C

Inorder traversal – example



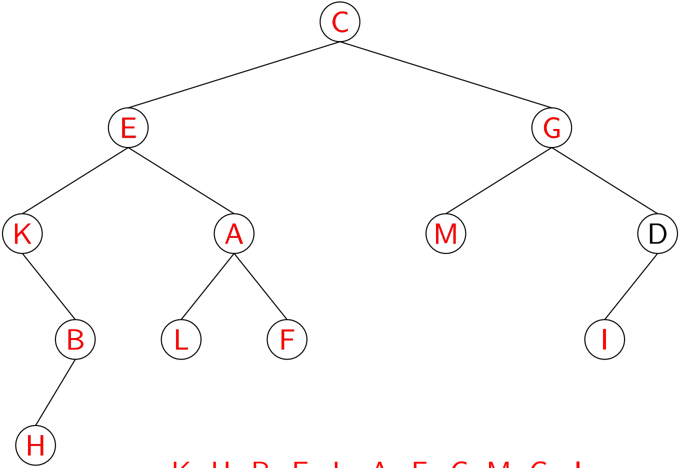
K H B E L A F C M

Inorder traversal – example



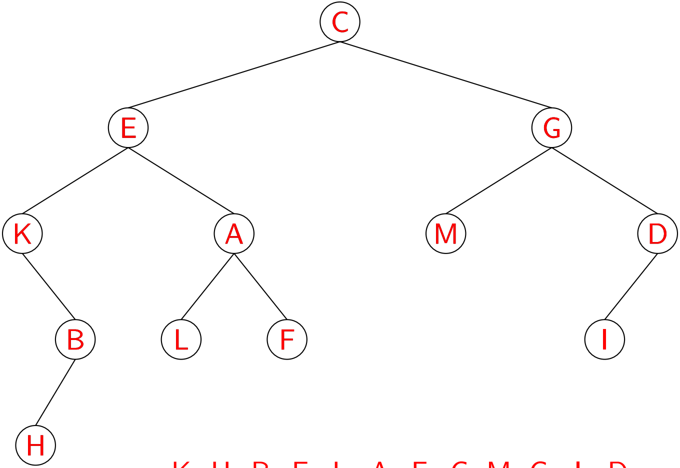
K H B E L A F C M G

Inorder traversal – example



K H B E L A F C M G I

Inorder traversal – example



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

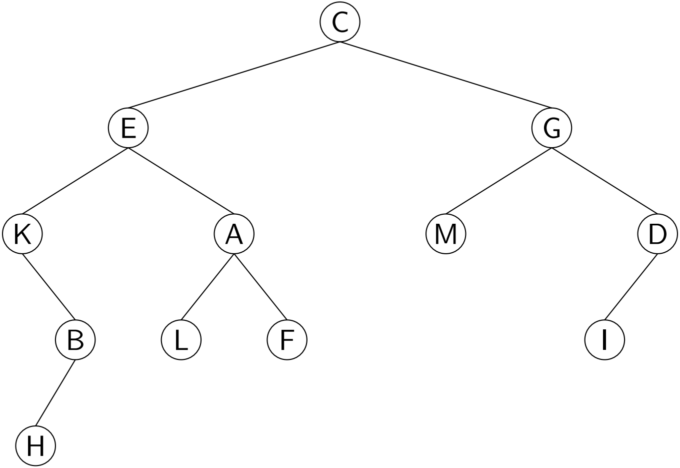
BinTree – postorder traversal

`postorder()`

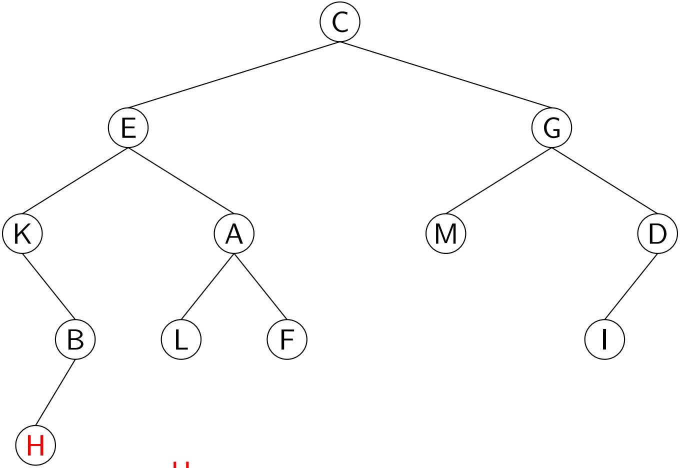
- input:
  - a binary tree `t`;
  - a procedure `visit()`.

- output:
  - binary tree `t` with nodes processed by `visit()` in the following order
    - (S) – left subtree
    - (D) – right subtree
    - (R) – root

Postorder traversal – example

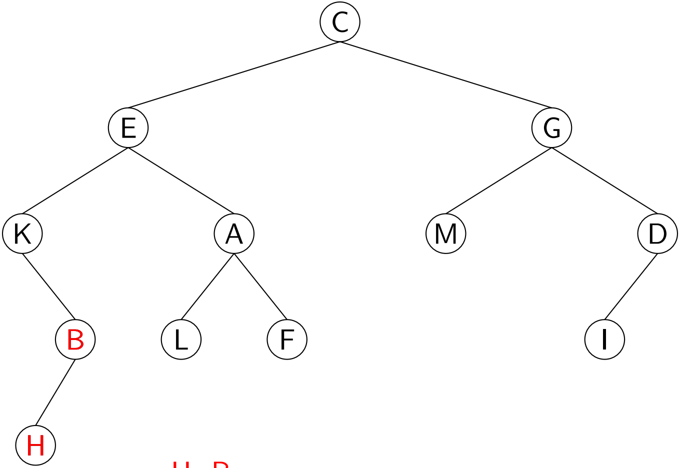


Postorder traversal – example



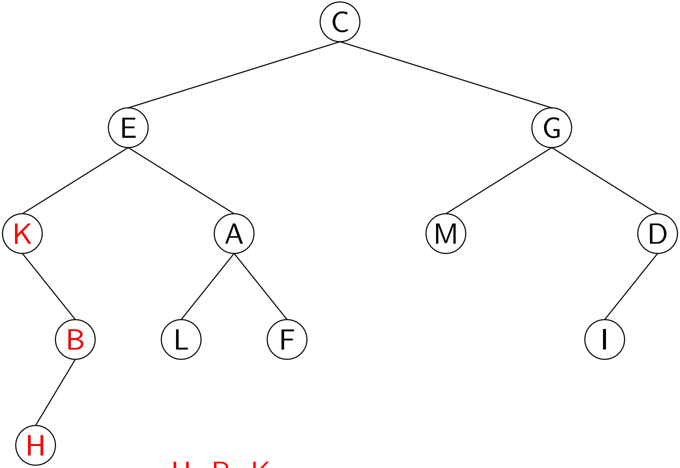
H

Postorder traversal – example



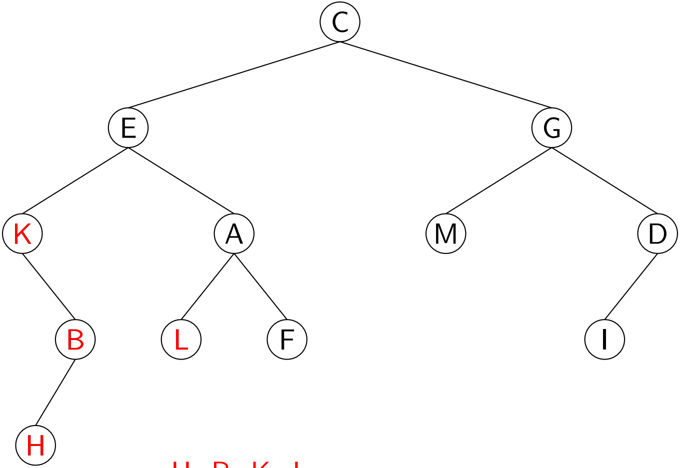
H B

Postorder traversal – example



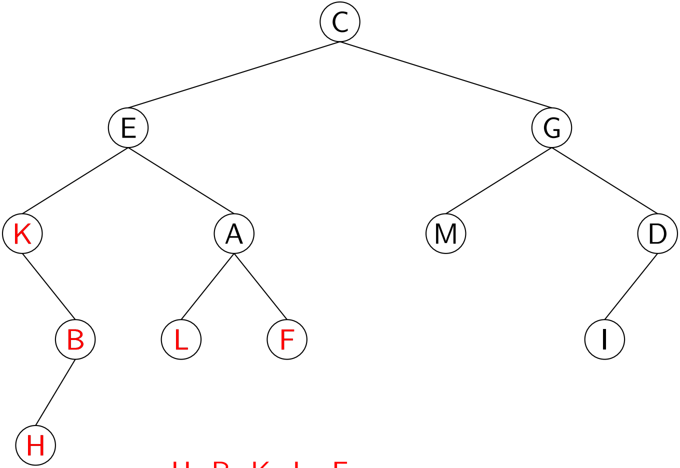
H B K

Postorder traversal – example



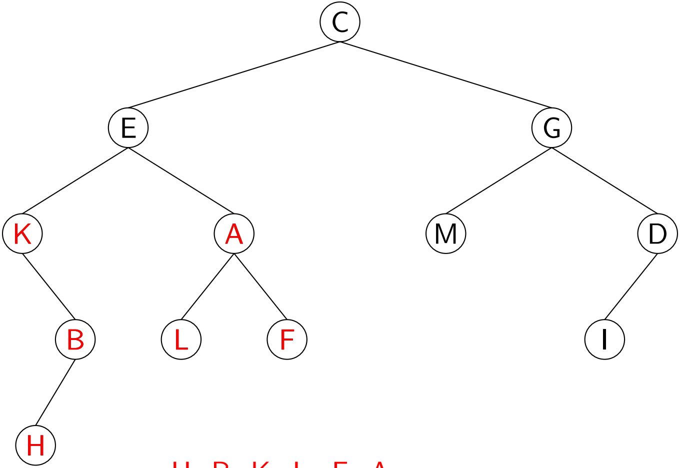
H B K L

Postorder traversal – example



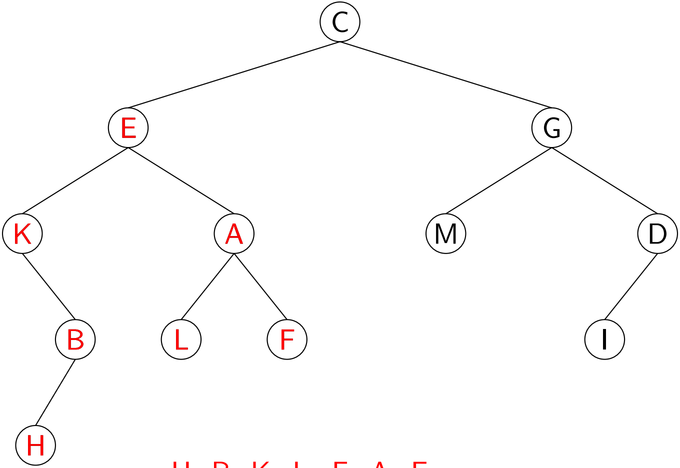
H B K L F

Postorder traversal – example



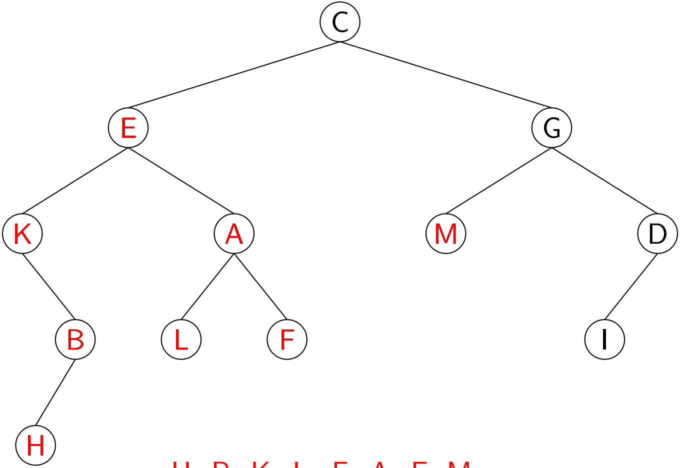
H B K L F A

Postorder traversal – example



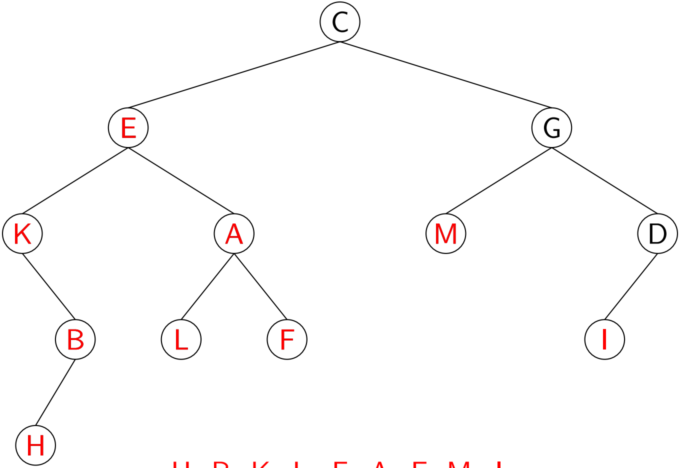
H B K L F A E

Postorder traversal – example



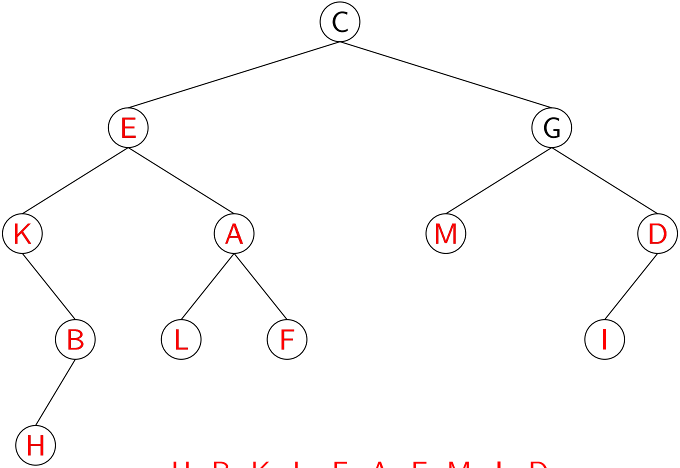
H B K L F A E M

Postorder traversal – example



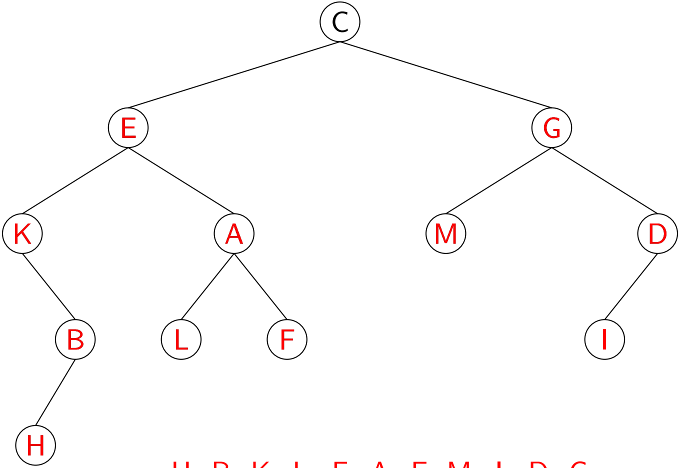
H B K L F A E M I

Postorder traversal – example



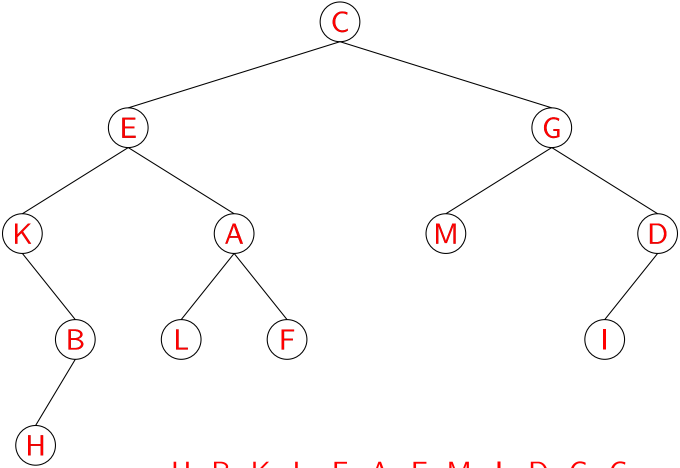
H B K L F A E M I D

Postorder traversal – example



H B K L F A E M I D G

Postorder traversal – example



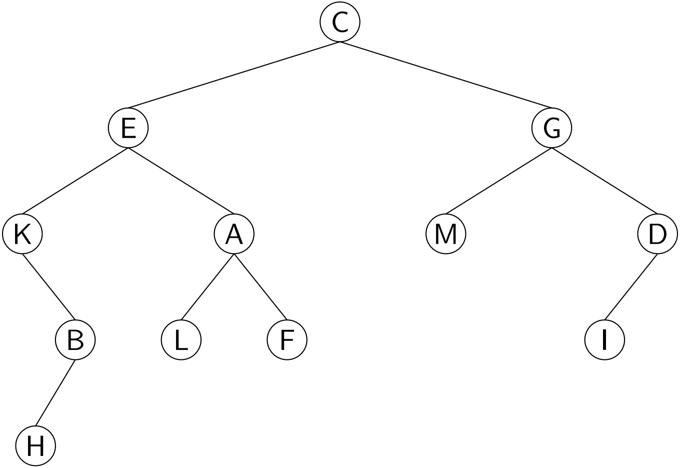
H B K L F A E M I D G C

BinTree – BFS traversal

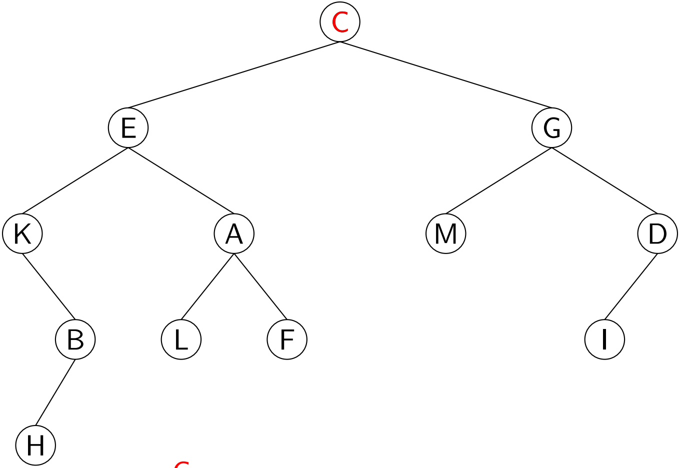
BFS() – Breadth-First Search

- input:
  - a binary tree t;
  - a procedure visit().
- output:
  - binary tree t with the nodes processed by visit() in BFS order (level by level).

BFS traversal – example

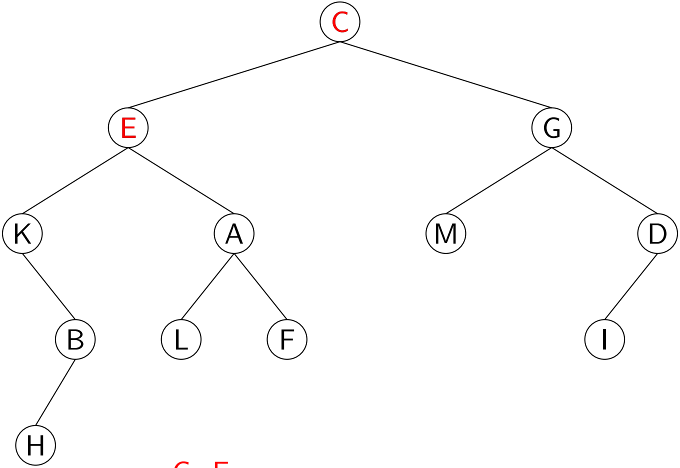


BFS traversal – example



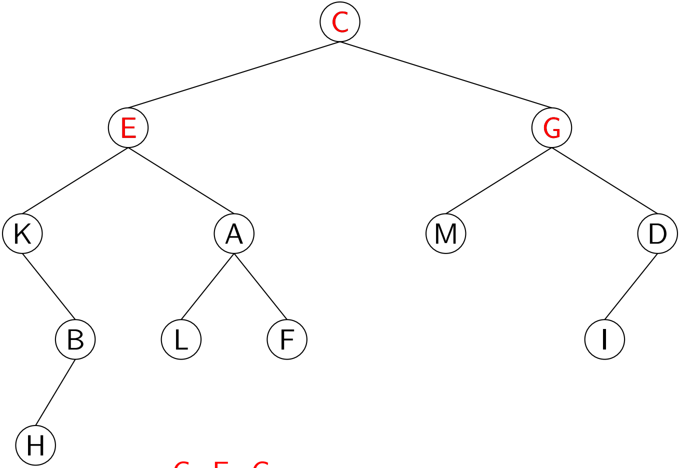
C

BFS traversal – example



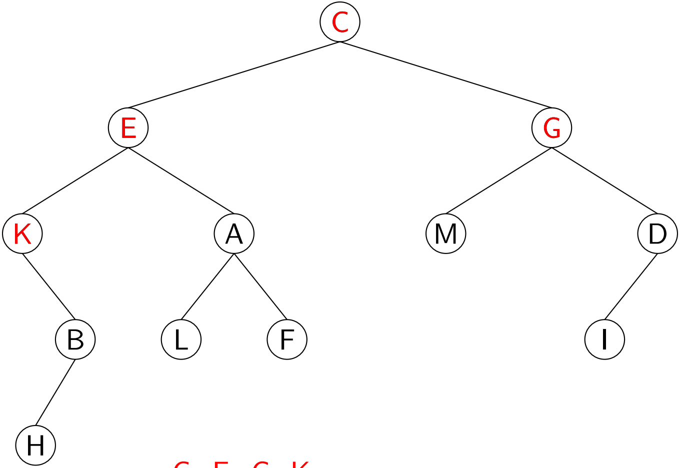
C E

BFS traversal – example



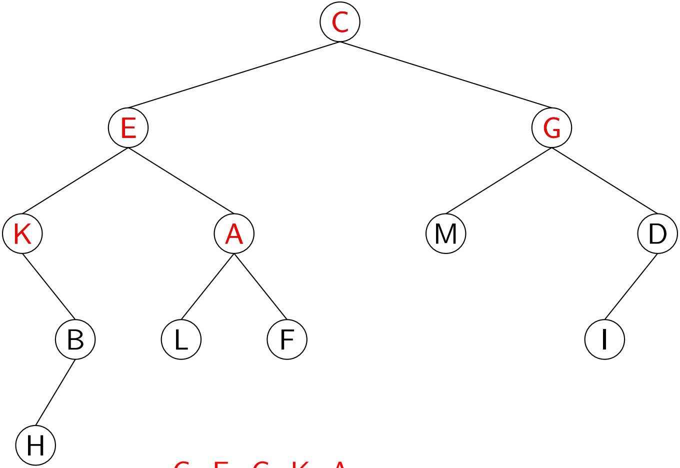
C E G

BFS traversal – example



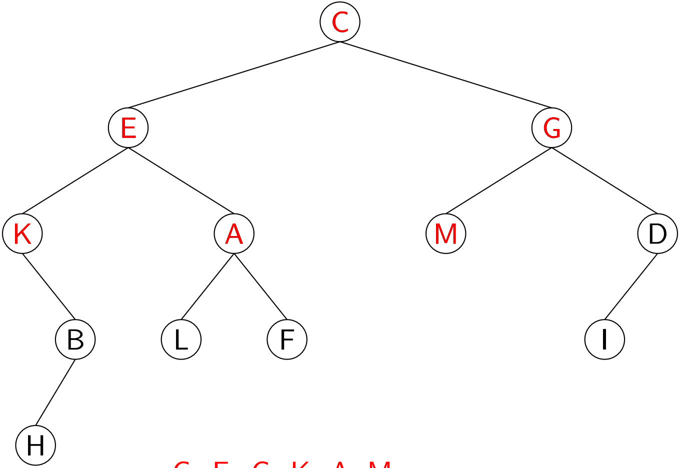
C E G K

BFS traversal – example



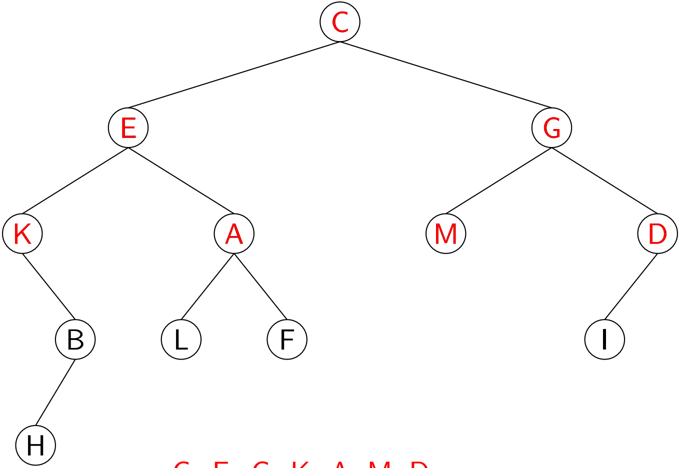
C E G K A

BFS traversal – example



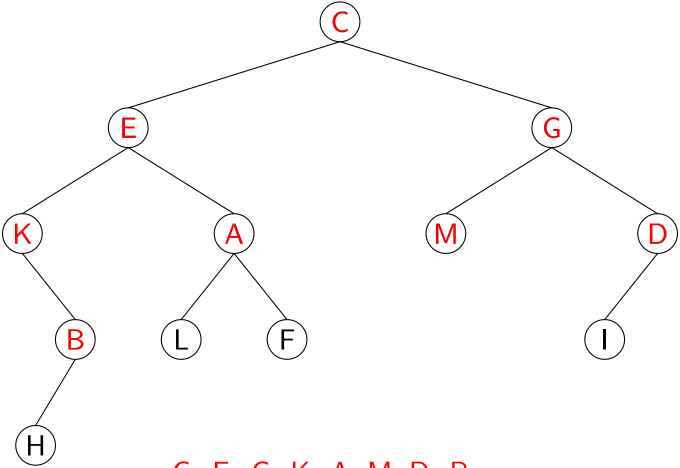
C E G K A M

BFS traversal – example



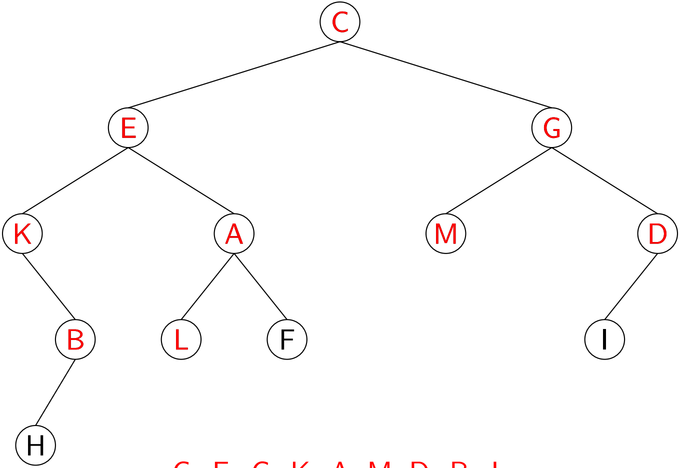
C E G K A M D

BFS traversal – example



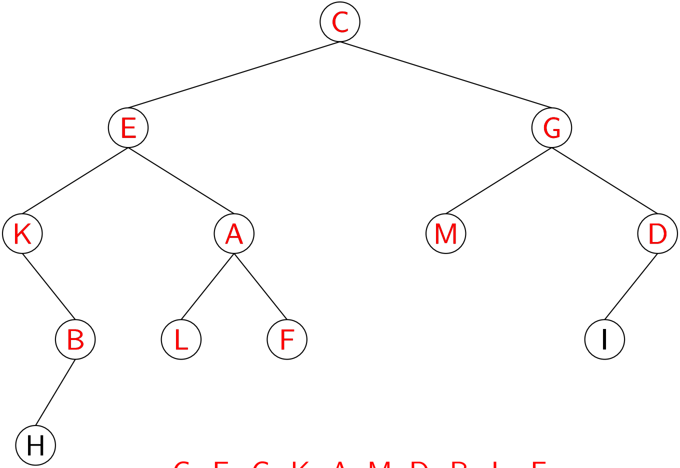
C E G K A M D B

BFS traversal – example



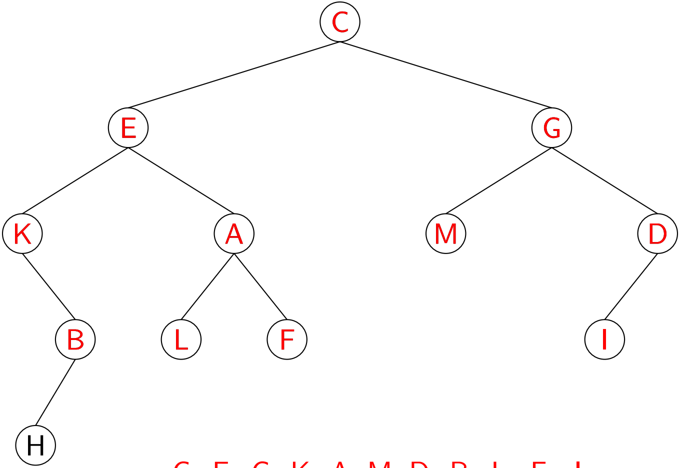
C E G K A M D B L

BFS traversal – example



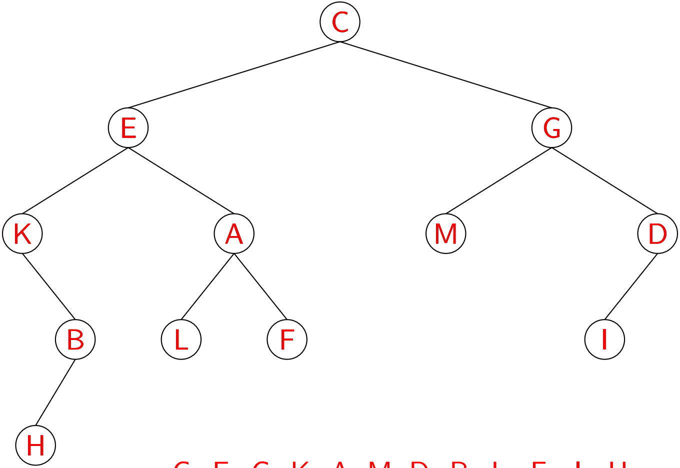
C E G K A M D B L F

BFS traversal – example



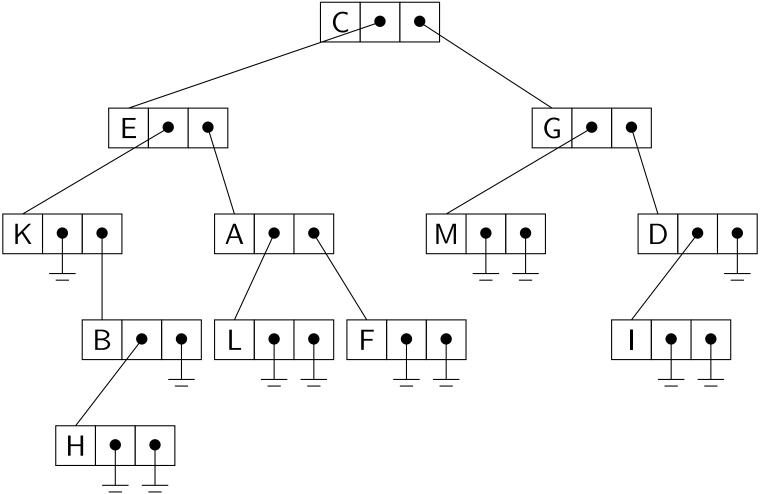
C E G K A M D B L F I

BFS traversal – example



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

BinTree: Linked structures implementation



BinTree: node structure

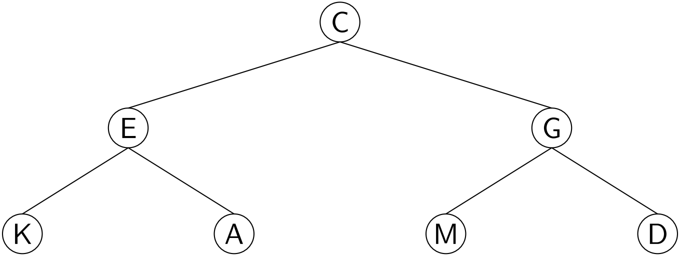
A node `v` (stored at memory address `v`) is a structure with three fields:

- `v->inf` – information stored in the node;
- `v->left` – left child address;
- `v->right` – right child address.

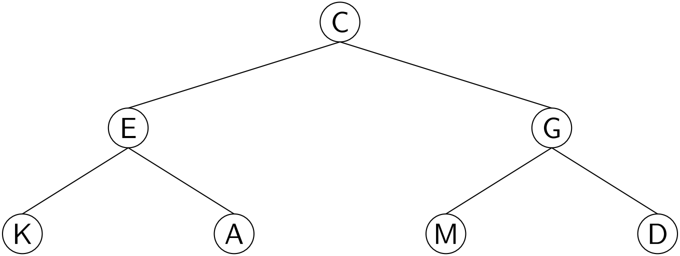
BinTree: preorder()

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

BFS traversal implementation



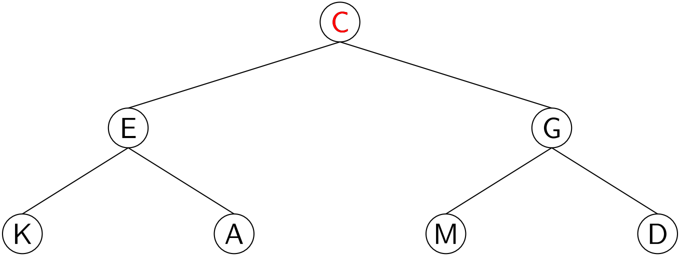
BFS traversal implementation



BFS =

Queue = (   )

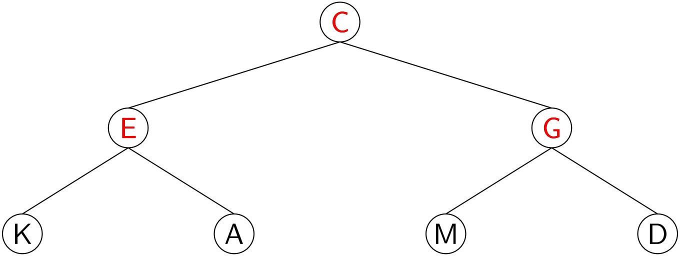
BFS traversal implementation



BFS =

Queue = (C)

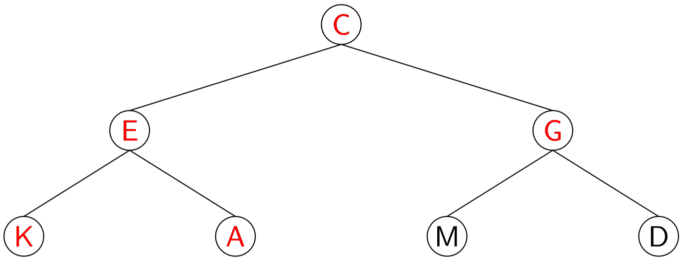
BFS traversal implementation



BFS = C

Queue = (E G)

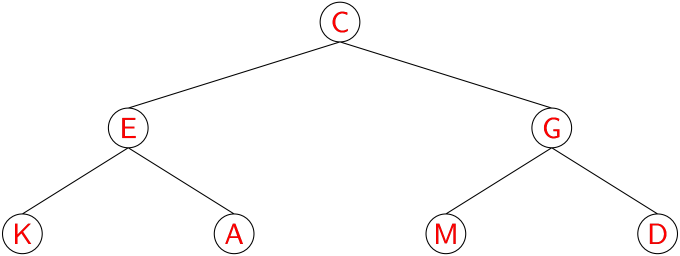
BFS traversal implementation



BFS = C E

Queue = (G K A)

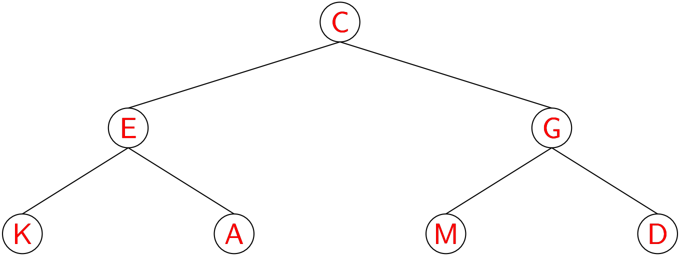
BFS traversal implementation



BFS = C E G

Queue = (K A M D)

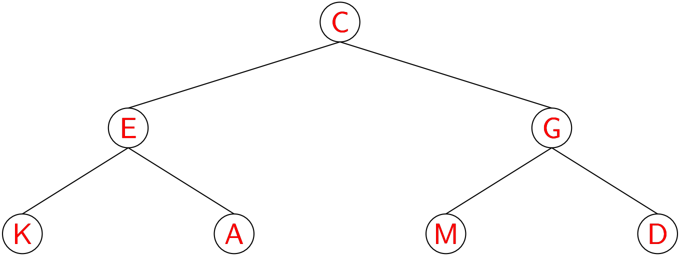
BFS traversal implementation



BFS = C E G K

Queue = (A M D)

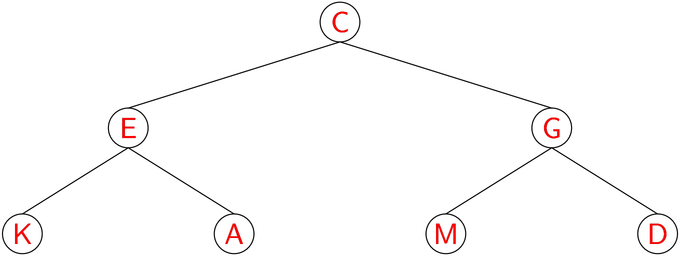
BFS traversal implementation



BFS = C E G K A

Queue = (M D)

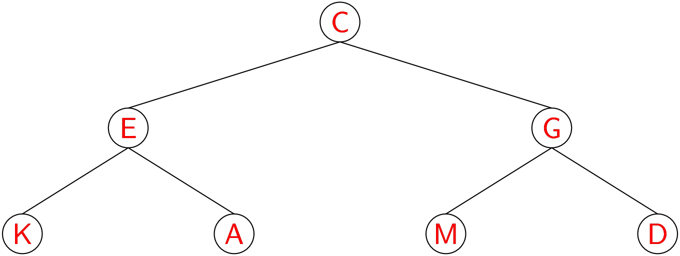
BFS traversal implementation



BFS = C E G K A M

Queue = (D)

BFS traversal implementation



BFS = C E G K A M D

Queue = ( )

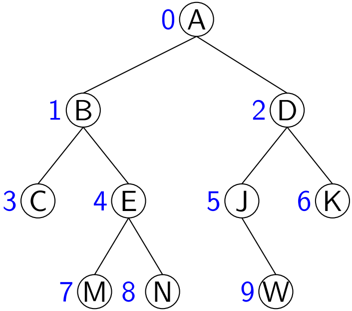
BFS traversal implementation

```
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 array: “parent” relation representation.
- Pro:
  - simplicity;
  - easy access from any node to the root;
  - memory savings.
- Cons:
  - non-easy access from the root to other nodes.

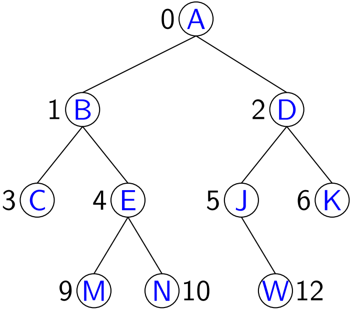
-1	0	0	1	1	2	2	4	4	5
0	1	2	3	4	5	6	7	8	9



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



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

Content

Trees

Binary tree (`BinTree`)

Application: integer expression representation as trees

Application: integer expression

- Integer expressions
  - definition;
  - examples.

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

Integer expression definition

```
<int> ::= ... -2 | -1 | 0 | 1 | 2 ...
<op_bin> ::= + | - | * | / | %
<exr_int> ::= <int>
            | (<exp_int>)
            | <exp_int> <op_bin> <exp_int>
```

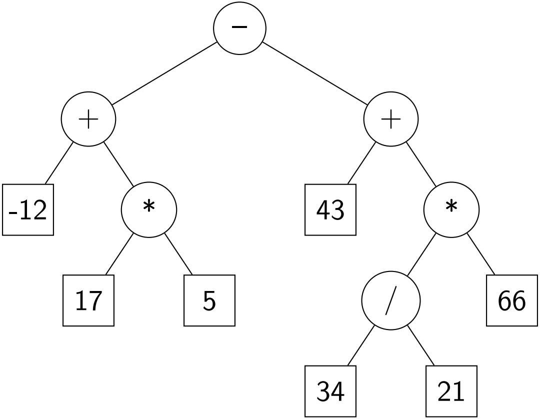
- priorities
  $12 - 5 * 2$ is $(12 - 5) * 2$ or $12 - (5 * 2)$?
- association rules
  $15/4/2$ is $(15/4)/2$ or $15/(4/2)$?
  $15/4 * 2$ is $(15/4) * 2$ or $15/(4 * 2)$?

Integer expressions as trees

$$-12 + 17 * 5 – (43 + 34/21 * 66)$$

Integer 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 given by the preorder traversal
  $-, +, -12, *, 17, 5, +, 43, *, /, 34, 21, 66$