ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ
И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Nonlinear dynamic structures

Algorithms and data structures

- lectures -

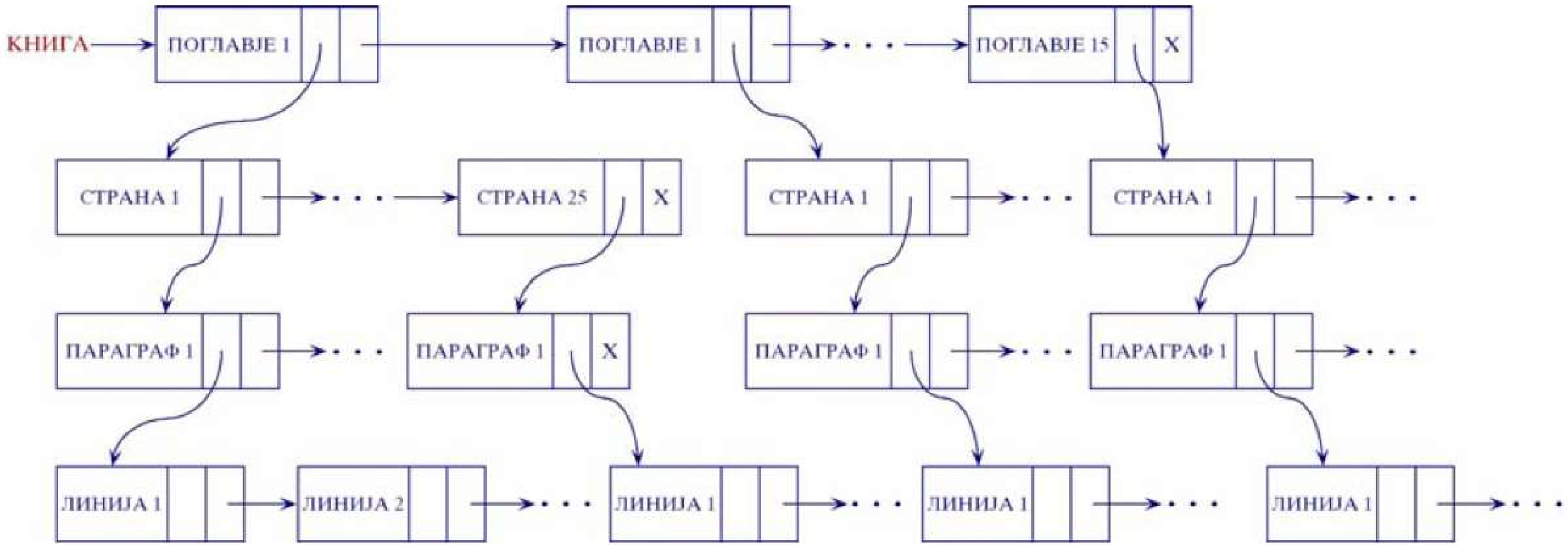
АПС

Complex lists

- Complex dynamic structures
  - lists in which the node points to a new list

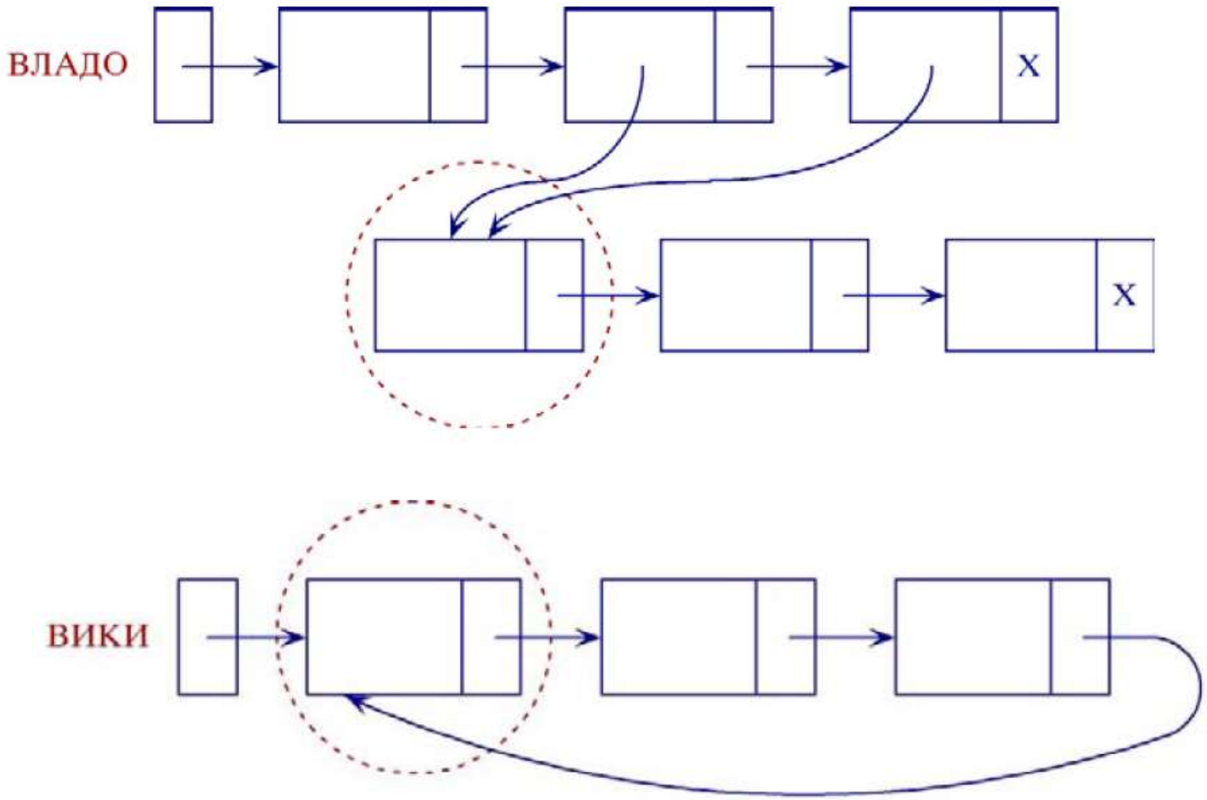
Complex lists

- Complex dynamic structures
  - lists in which the node points to a new list
- **Example**: structure of a book (**hierarchy**)



Complex lists

- Structures which allow more links to point (to share) to one node



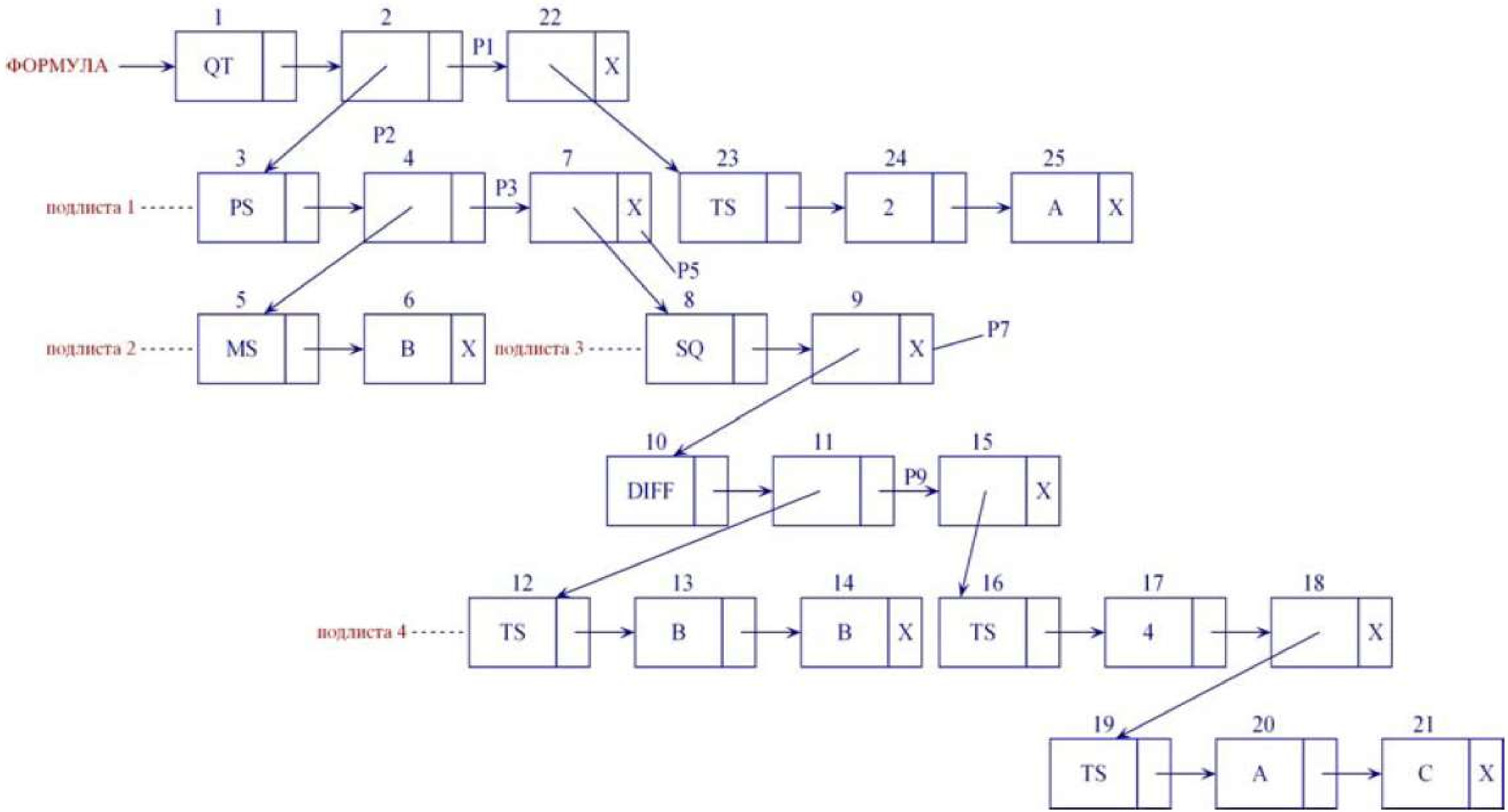
Complex lists

- Operations to work with complex lists:
  - Node insertion
  - Node deletion
  - List traversal

Complex lists

- Traversal of hierarchical complex lists can be described as follows:
  - Access the first node (if it exists)
  - Process the node you accessed
  - If the node is complex, traverse the list(s) it points to.
  - Access the next node (if any)

Complex lists



Trees

- Hierarchical collection of elements
- The tree is:
  - Collection of elements - nodes
  - One node is special - root
  - Relation „is a parent of “
  - Each node has exactly one parent
  - Each node keeps data from any data type

Trees

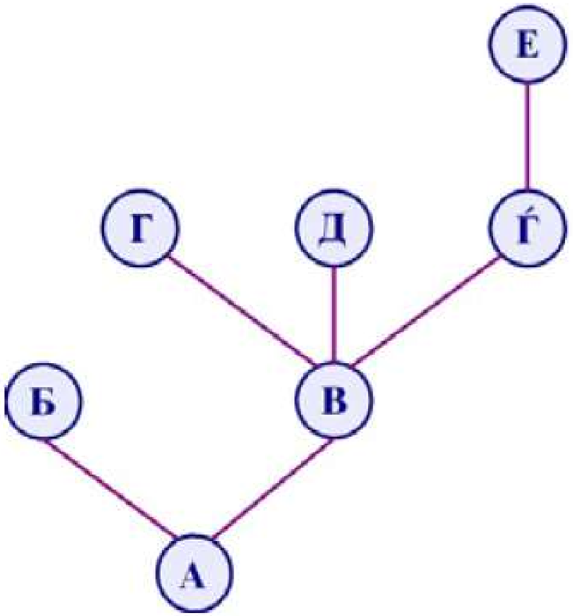
- Formal definition of a tree:
  - A node itself represents a tree. Then, that node is the root of the tree
  - Let n be node and *T1, T2, ..., Tk* be trees with roots *n1, n2, …, nk*, respectively. Then, a tree can be constructed if the node n is made the root of the tree that contains the subtrees *T1, T2, ..., Tk*. The nodes *n1, n2, …, nk* are called children of node *n*.

Trees

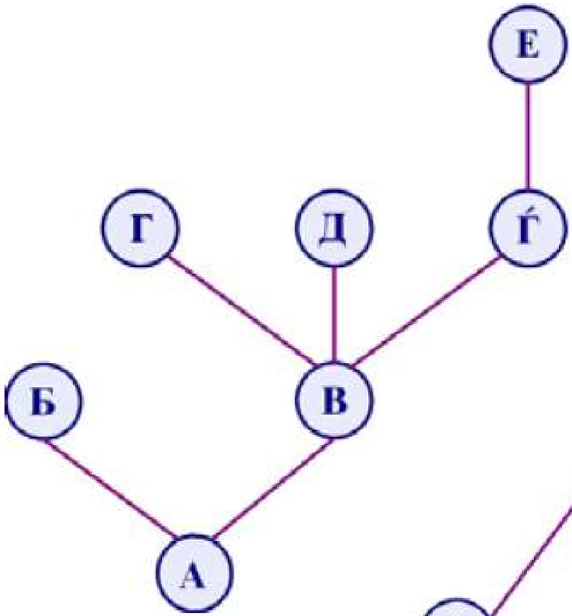
- Recursive definition of a tree:

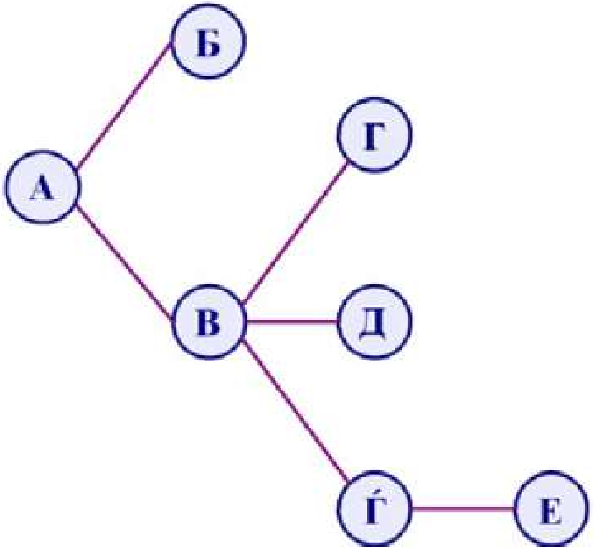
- A tree is a finite set *T* with one or more elements called nodes that satisfies the following rules:
  - There exists one node called the root of the tree
  - The remaining nodes (without the root) are grouped in $k \geq 0$ disjunctive sets *T1, T2, ..., Tk*, which are trees each. These trees are called subtrees of the tree *T*.

Trees

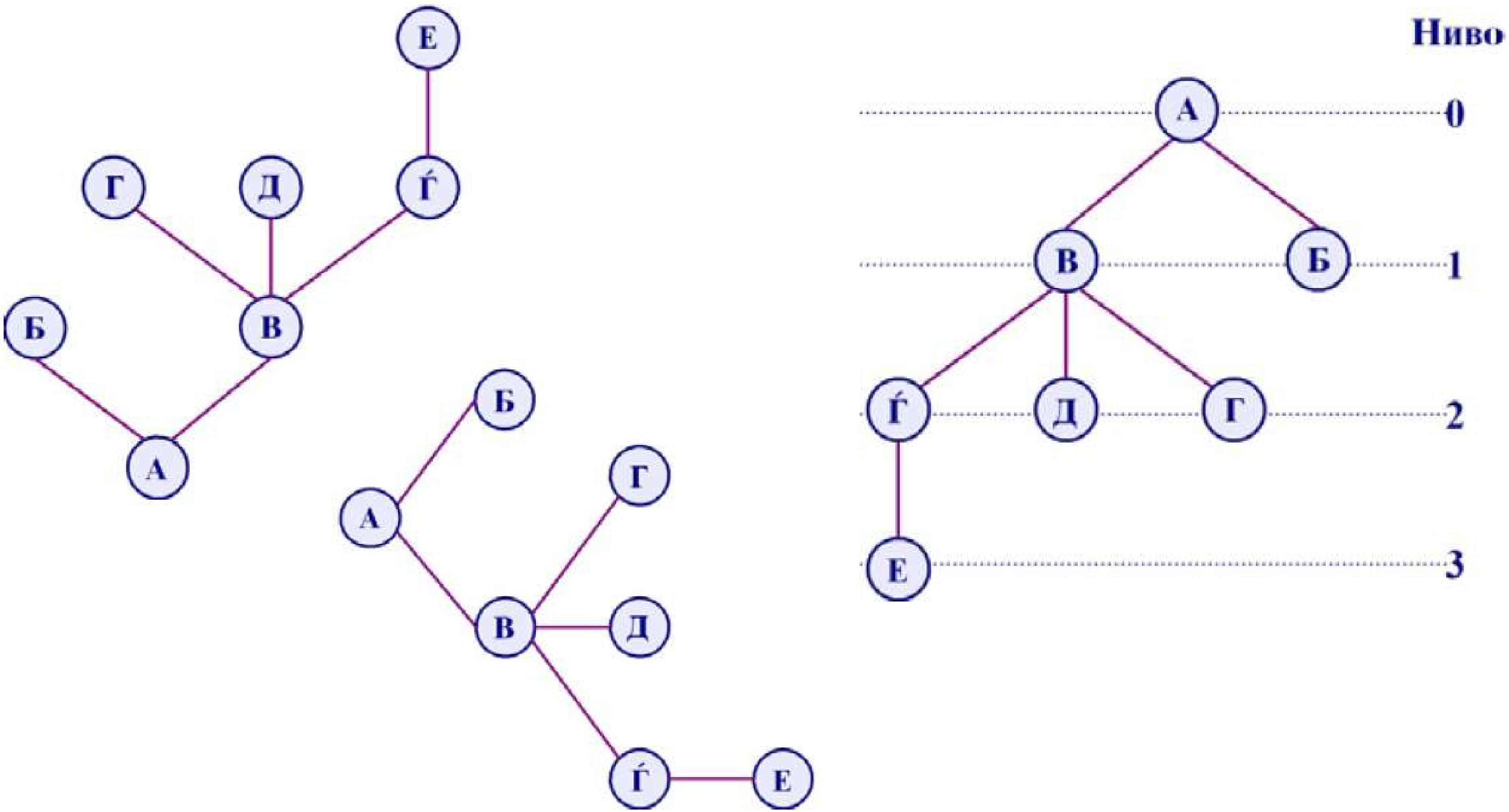


Trees





Trees



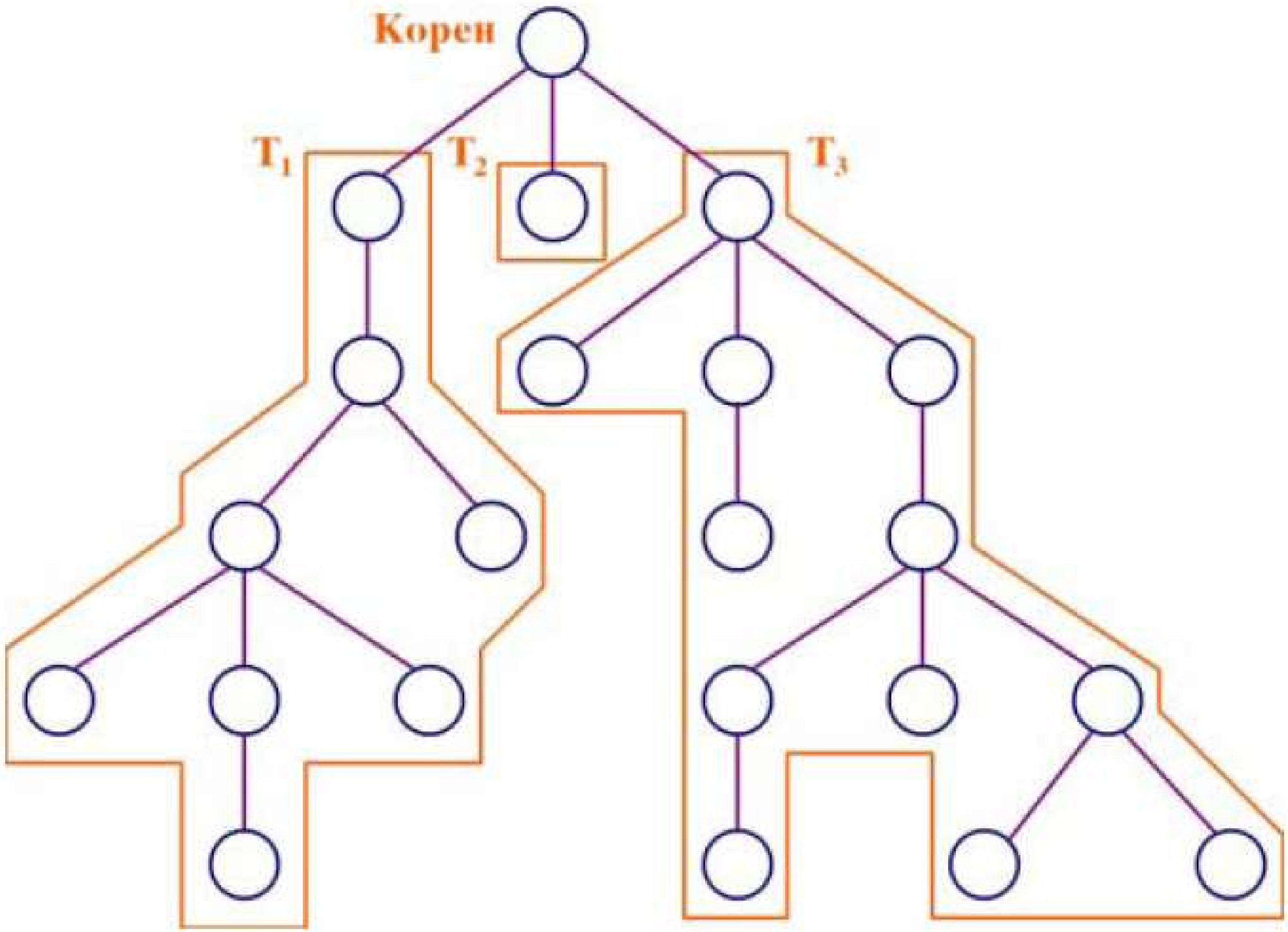
Trees

- every internal node in the tree is the root of a subtree
- the number of subtrees of a node is called the degree of the node
  - When this number is 0, the node is called a terminal node or leaf
- all nodes (except the root) have their own parent

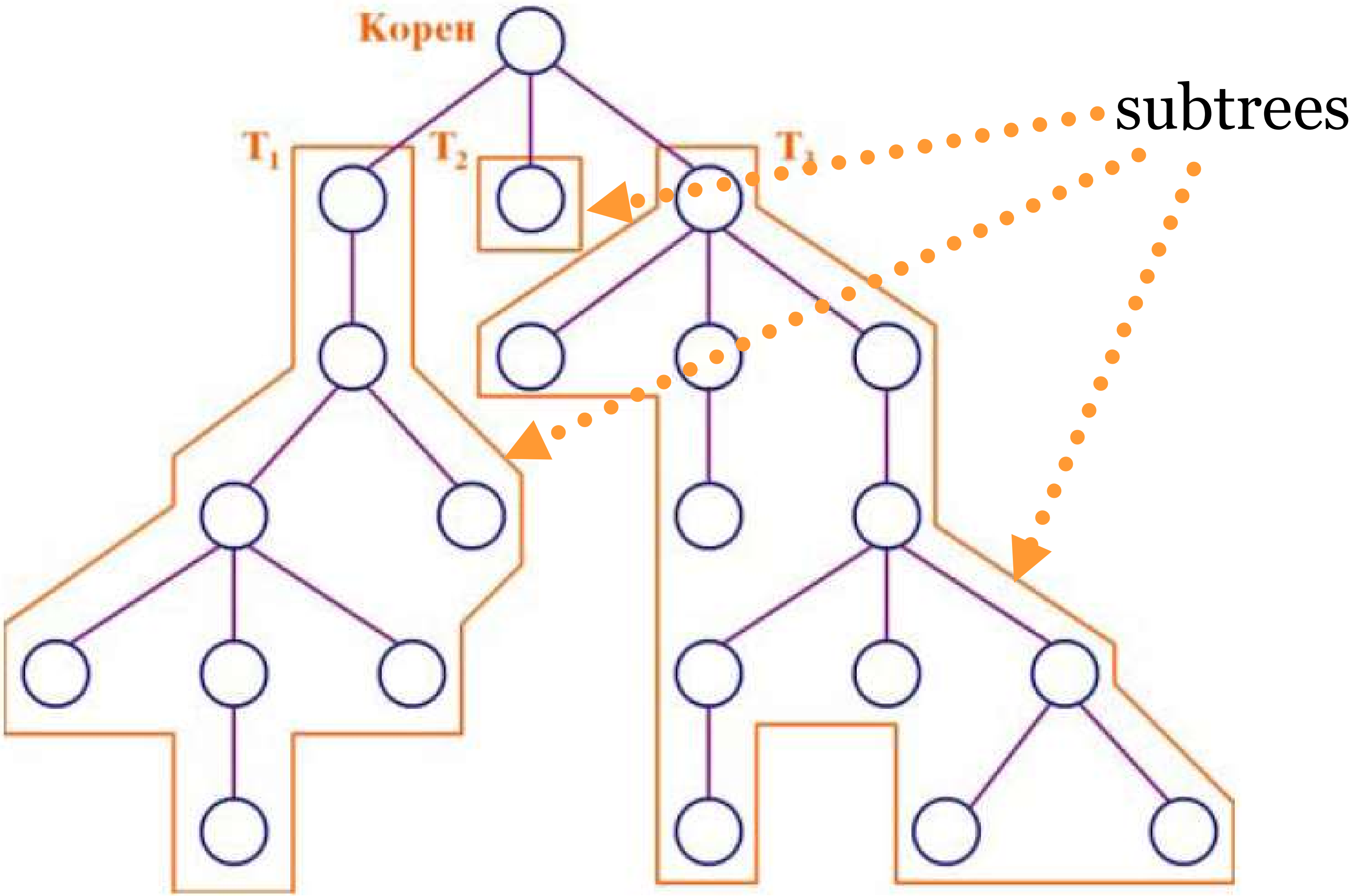
Trees

- **Similar trees** are trees that have the same structure, i.e. whose nodes and links are corresponding (if the node in one tree has two children, and the corresponding node in the other tree has two children, and the number of their children is the same)
- **Equivalent trees** are trees that are similar, but that carry the same information in each node.

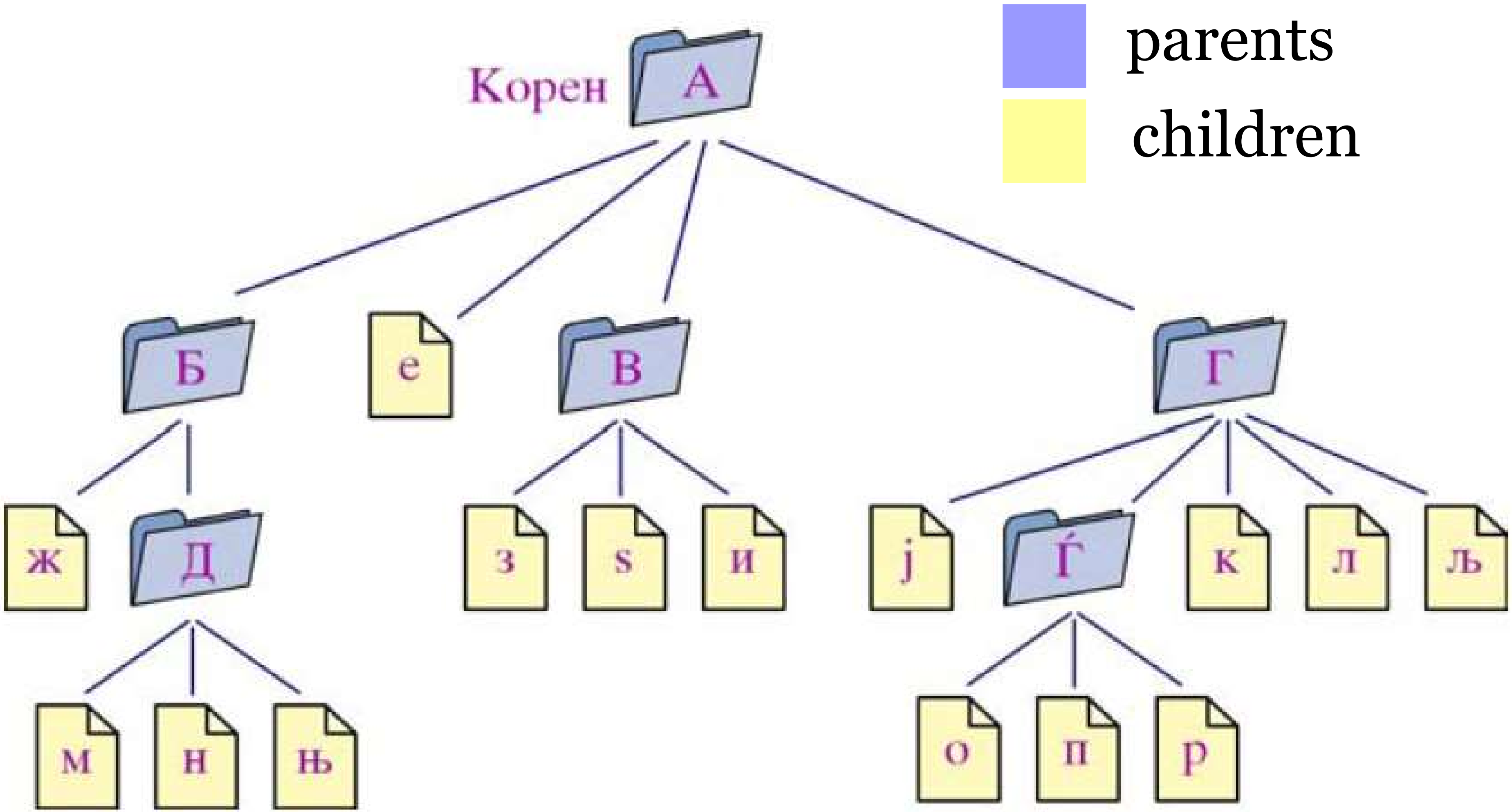
Trees



Trees



Trees



Forest of trees

- A set (usually ordered) of different (disjoint) trees is called a **forest**
- If we remove the root from a tree, we get a forest
- If we add only one node to a forest and connect it to the roots of the trees, we get one tree from the forest

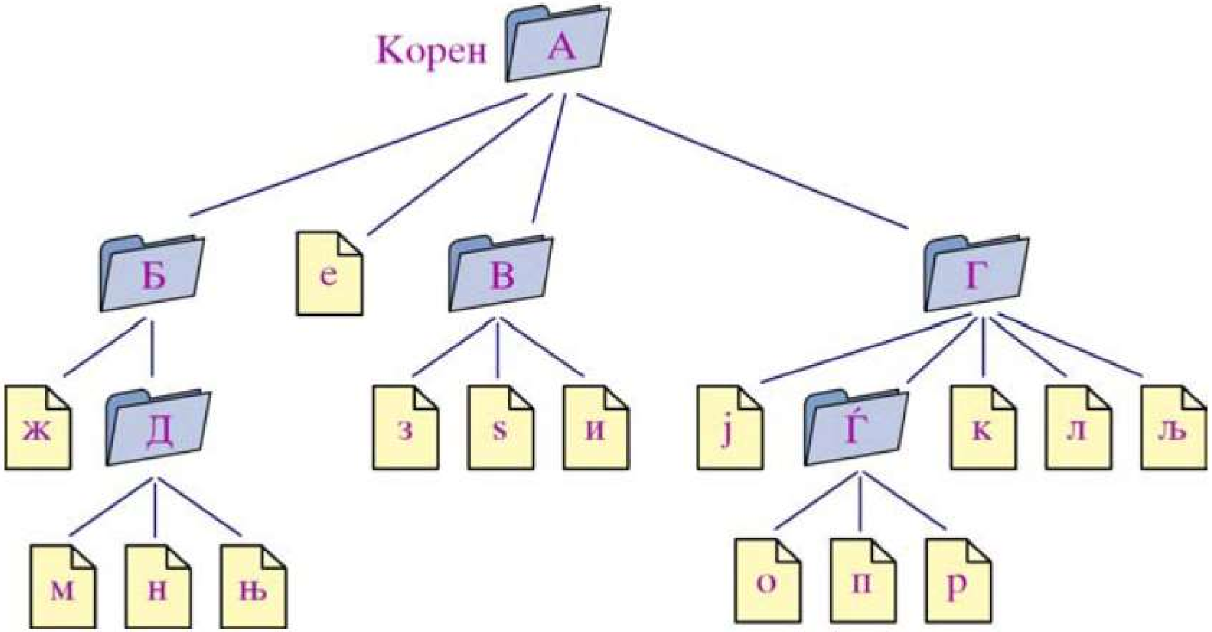
Path in a tree

- If $n_1, n_2, \ldots, n_k$ is a sequence of nodes in a tree such that n_i is the parent of n_{i+1}, $1 \le i < n$, the the sequence is called a **path** from node n_1 to n_k
- **Path length** represents the number of connections between two nodes, i.e. it is one less than the number of nodes in the path
- **Ancestor** and **descendant** of node

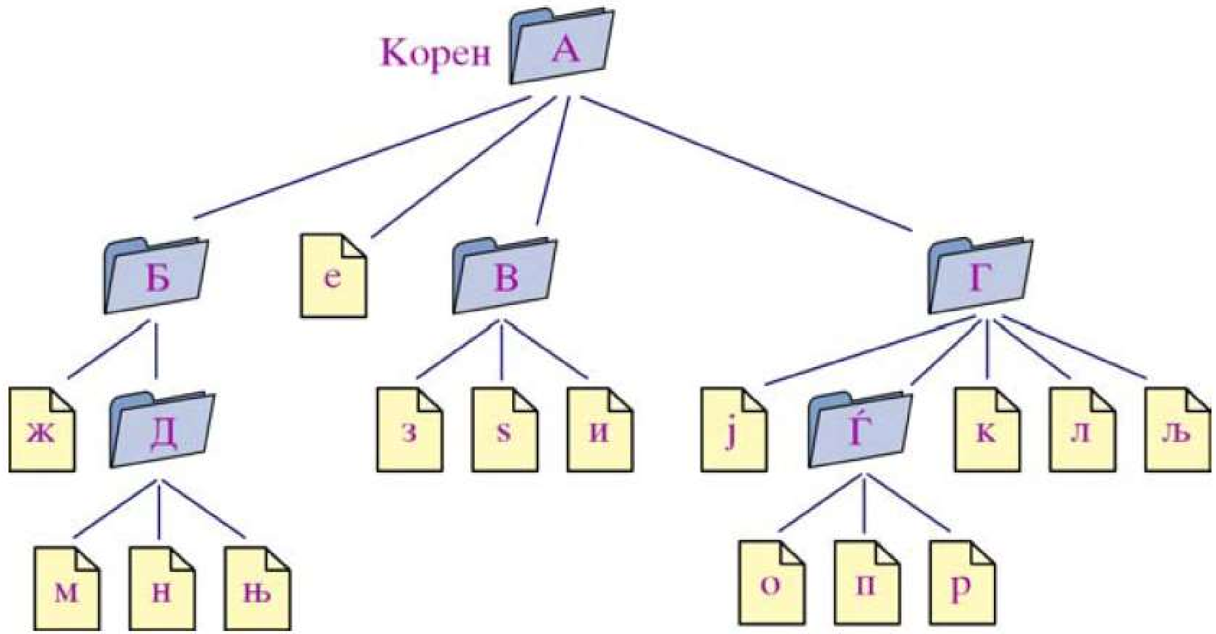
Trees

- A **subtree** of a given node in a tree is the child node with all its descendants
- The number of subtrees of a node is called the **degree** of the node
- **Node height** in a tree is the length of the longest path from the node to the leaves
- The **depth** of a node is the length of the unique path from the root to the node

Trees

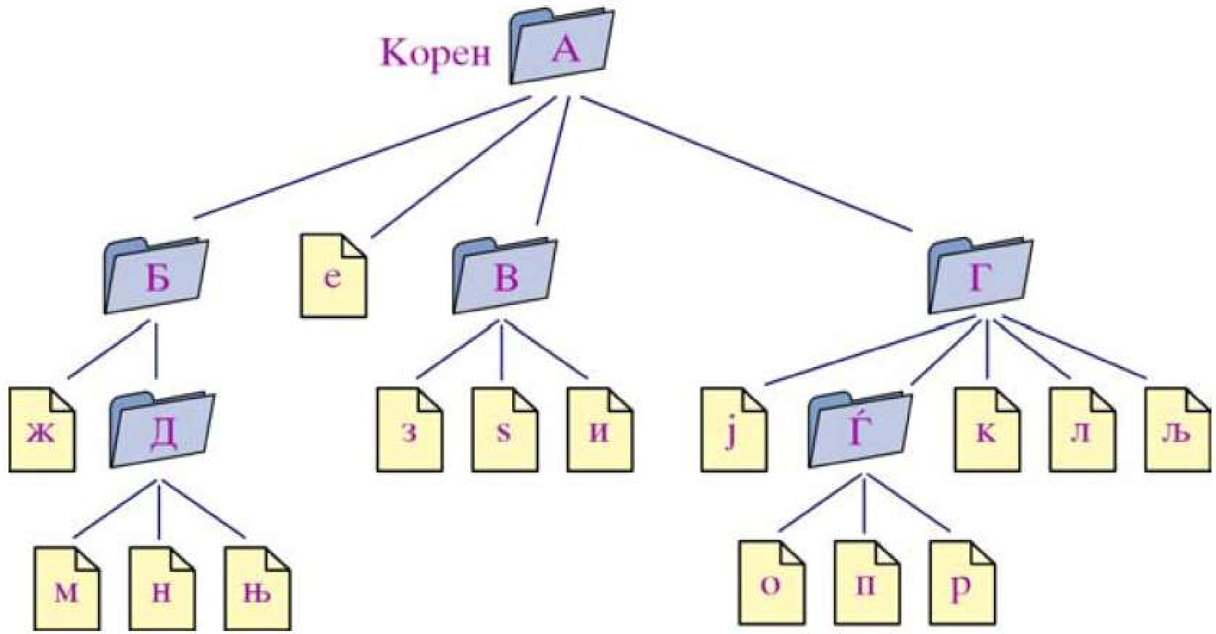


Trees



Path from A to м: А Б Д м

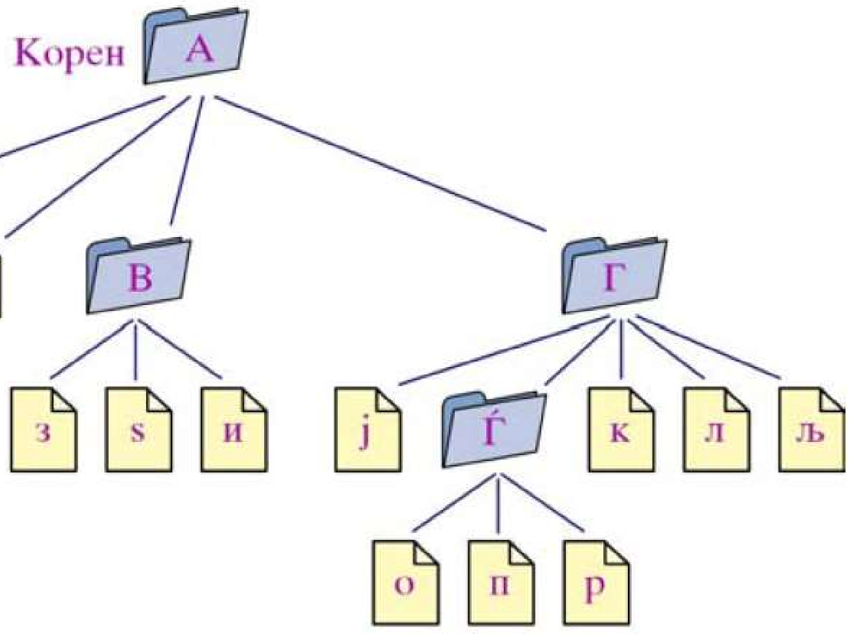
Trees



Path from A to м: А Б Д м

Descendants of Б: Д ж м н њ

Trees

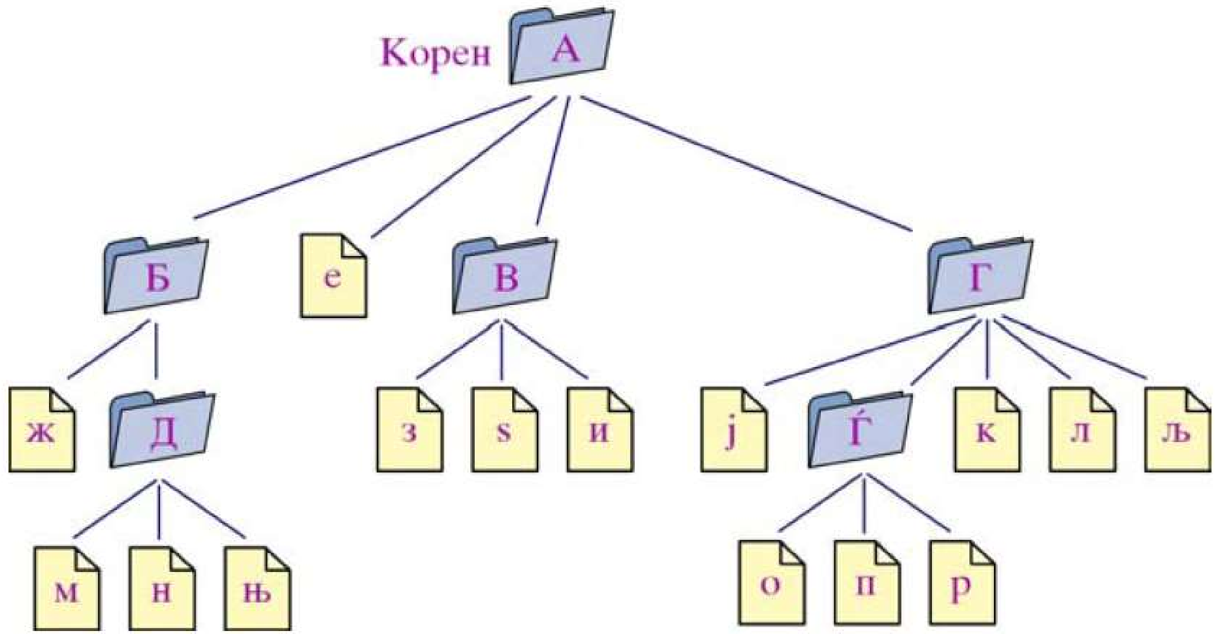


Path from A to м: А Б Д м

Descendants of Б: Д ж м н њ

Ancestors of м: Д Б А

Trees



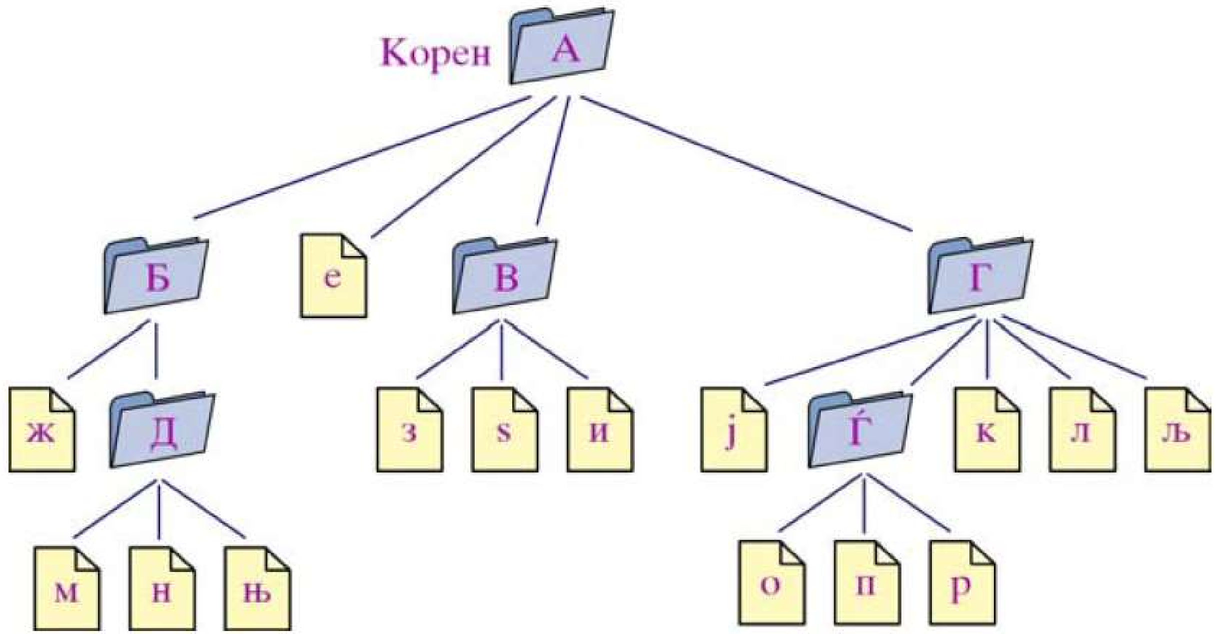
Path from A to м: А Б Д м

Degree of A: 4

Descendants of Б: Д ж м н њ

Ancestors of м: Д Б А

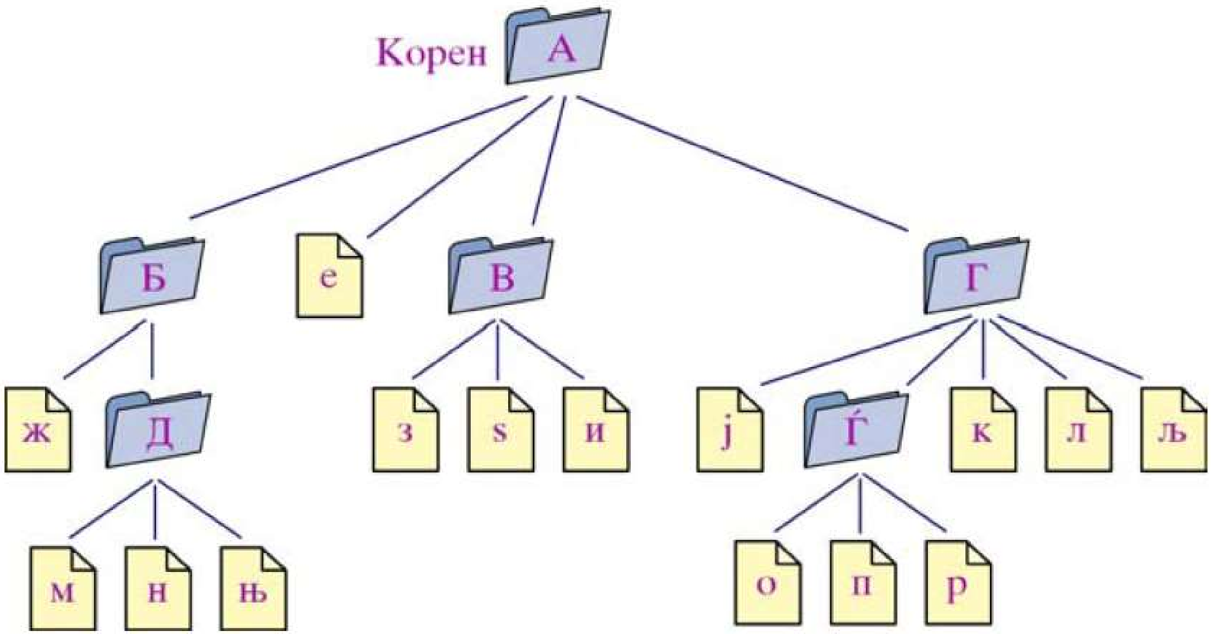
Trees



Path from A to м:	А Б Д м
Descendants of Б:	Д ж м н њ
Ancestors of м:	Д Б А

Degree of A:	4
Degree of Г:	5

Trees



Path from A to м: А Б Д м

Descendants of Б: Д ж м н њ

Ancestors of м: Д Б А

Degree of A: 4

Degree of Г: 5

Degree of the tree: 5

Tree representation

- The parent node contains pointers to its child nodes

Tree representation

- The parent node contains pointers to its child nodes

Problem: Each root of a subtree in a tree can have an arbitrary number of children!

Tree representation

- The parent node contains pointers to its child nodes

Problem: Each root of a subtree in a tree can have an arbitrary number of children!

Solution1: Predictable number of pointers in each node of the tree!

Tree representation

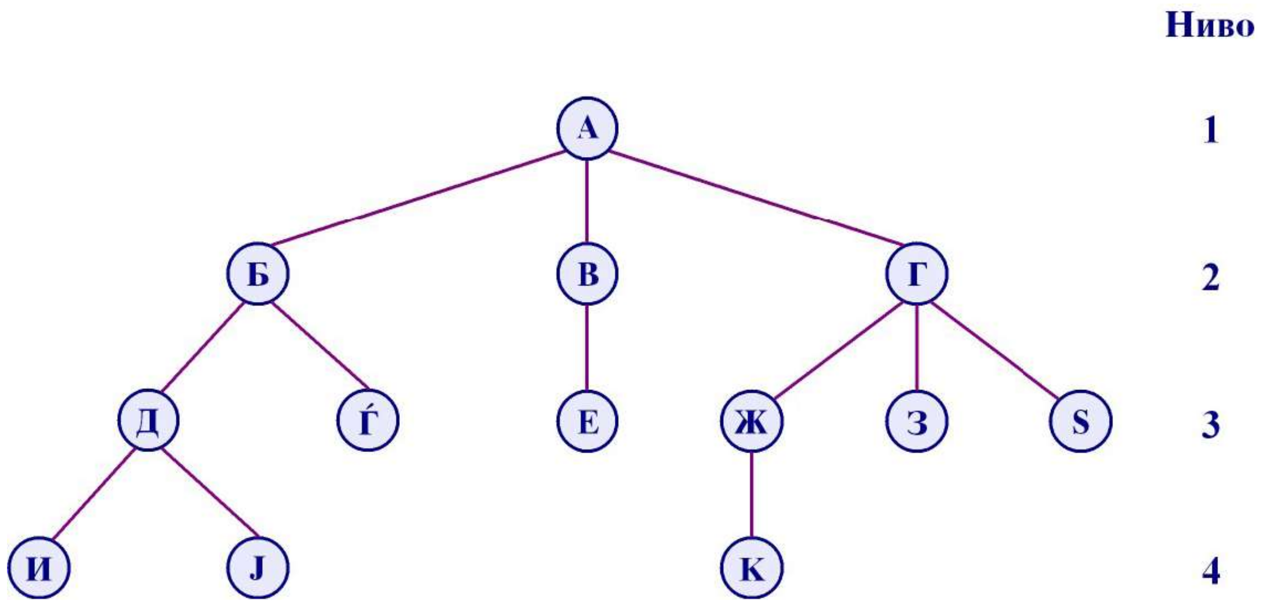
- The parent node contains pointers to its child nodes

Problem: Each root of a subtree in a tree can have an arbitrary number of children!

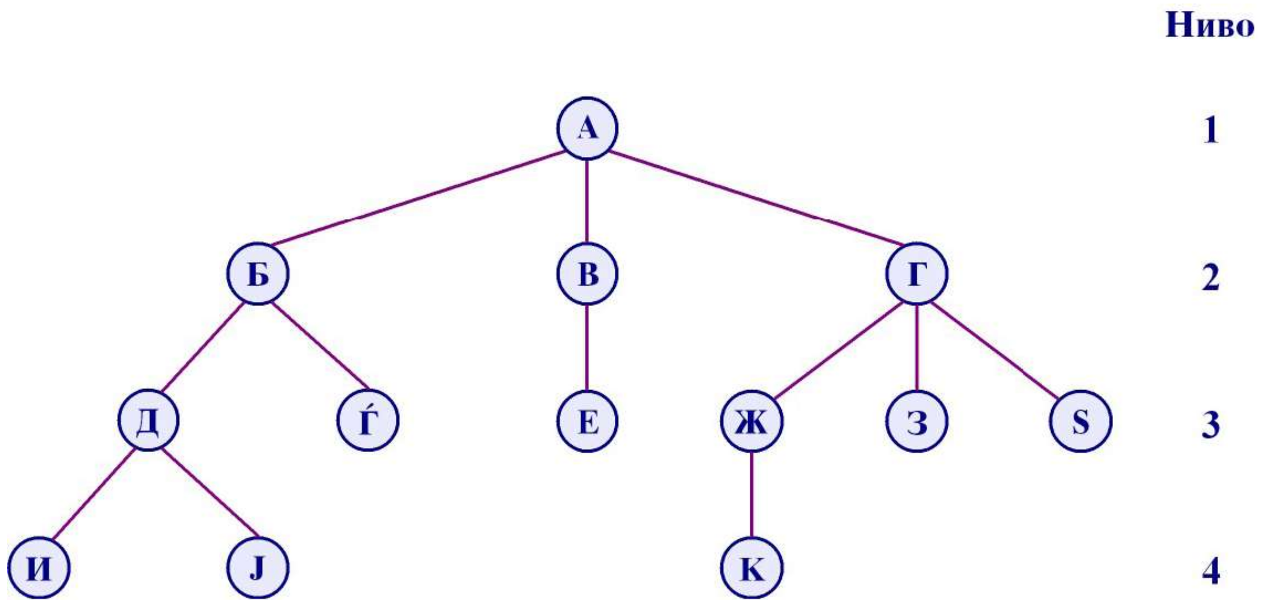
Solution1: Predictable number of pointers in each node of the tree!

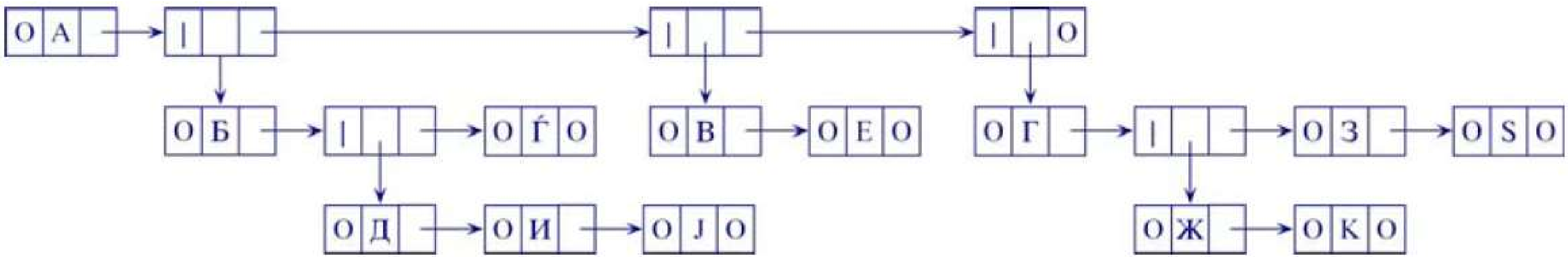
Solution2: A structure where the number of pointers is at most two!

Tree representation



Tree representation



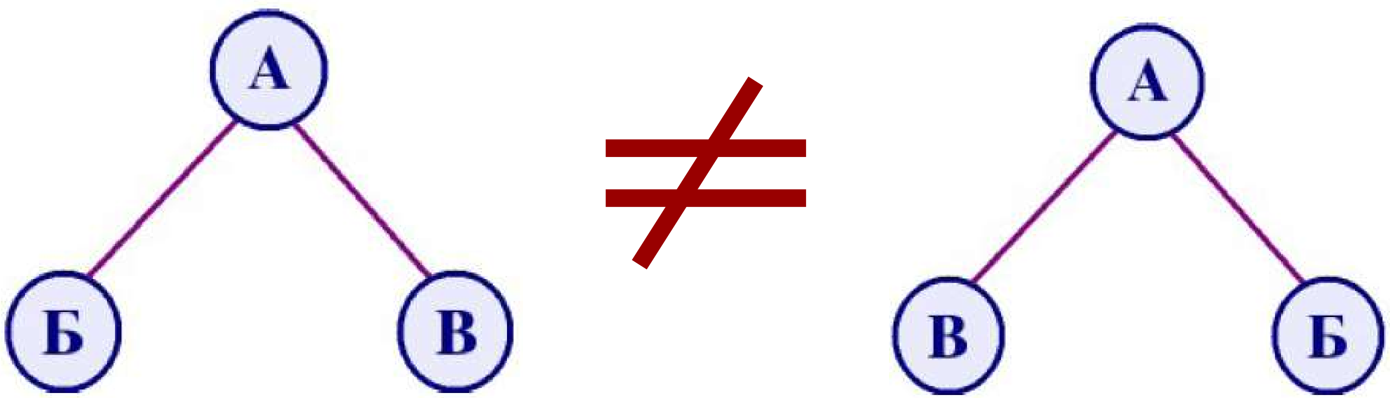


Ordered tree

- When the order of subtrees in a tree is important, the tree is called an **ordered tree**

Ordered tree

- When the order of subtrees in a tree is important, the tree is called an **ordered tree**



Binary tree

- each node can have a maximum of two subtrees
  - left subtree
  - right subtree
- the degree of the tree is two
- nodes in a binary tree can:
  - have no children
  - have one or
  - maximum of two children

Binary tree

- If we consider that the root of a binary tree is at level 1, and at each subsequent level the number of nodes is twice as large, then the maximum number of nodes at level *i* in binary trees is 2^{i-1}, $i>=1$
- The maximum total number *n* of nodes in the binary tree (number of nodes for a maximally filled tree) with depth *d* is 2^d-1, $d>=1$. It is obtained from the equation:

$$n = \sum_{i=1}^{d} 2^{i-1} = 2^d - 1$$

Binary tree

- From the previous equation it follows that:

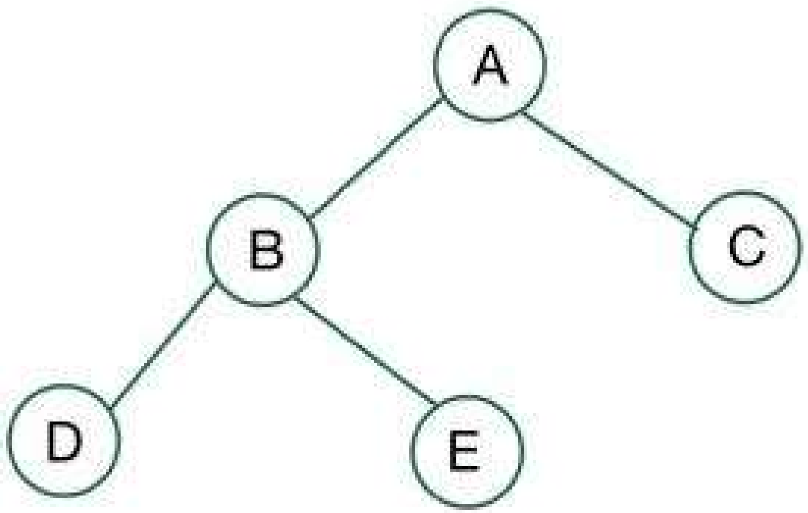
$$n \le 2^d - 1$$

- Thus, if we know the total number of nodes, the depth of the tree is obtained as: $d \ge \log_2(n+1)$.
- When the tree is maximally filled, it has the smallest depth of all possible binary trees with total number of nodes n, and that depth is obtained as:

$$d = \lceil \log_2(n+1) \rceil$$

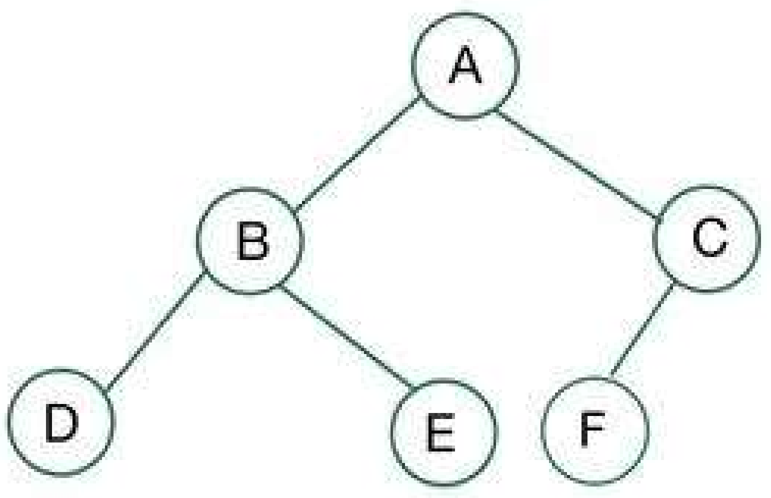
Full binary tree

- A full binary tree is a binary tree in which all nodes have either 0 or 2 children.
- A full binary tree is a binary tree in which all nodes, except leaf nodes, have two children.

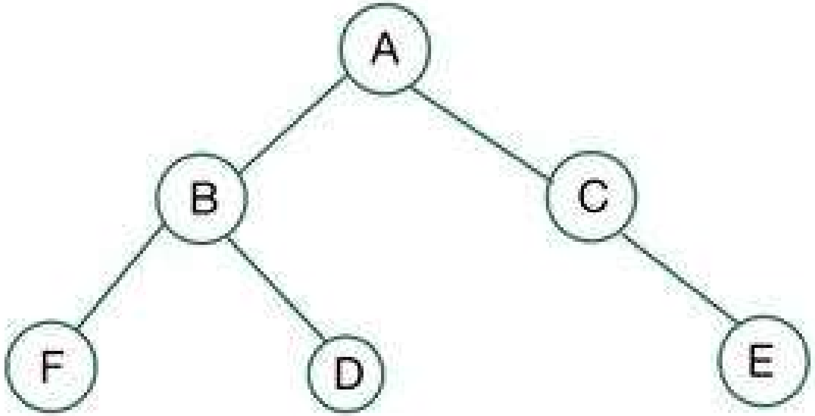


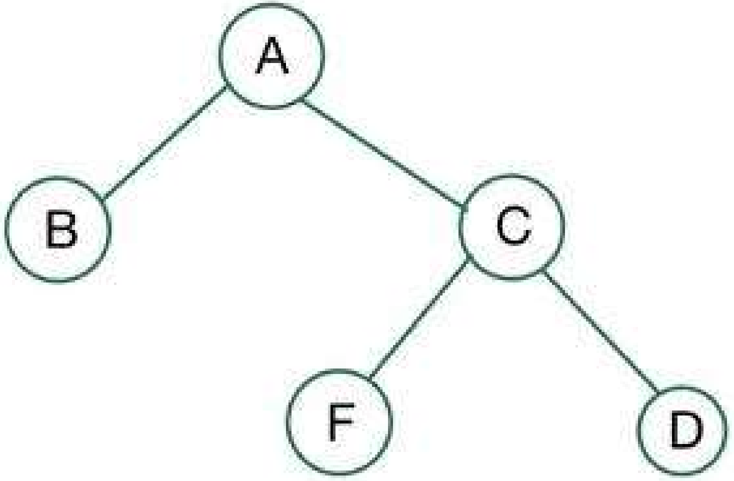
Complete binary tree

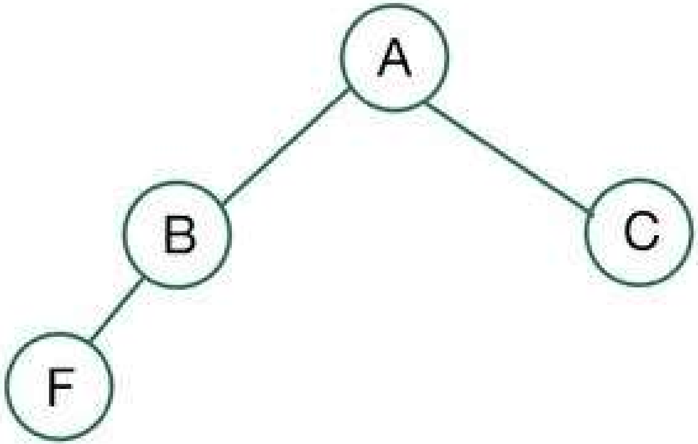
- When all levels of a binary tree are completely filled except for the last level, which may contain 1 or 2 children and is filled from the left, it is said to be a complete binary tree.

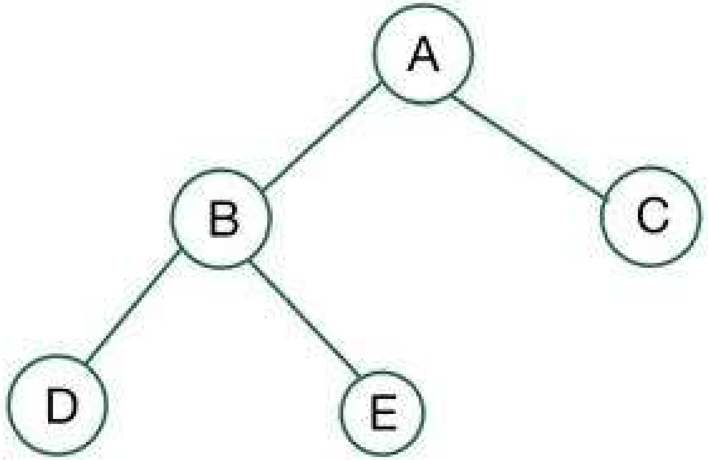


What type are the following binary trees?

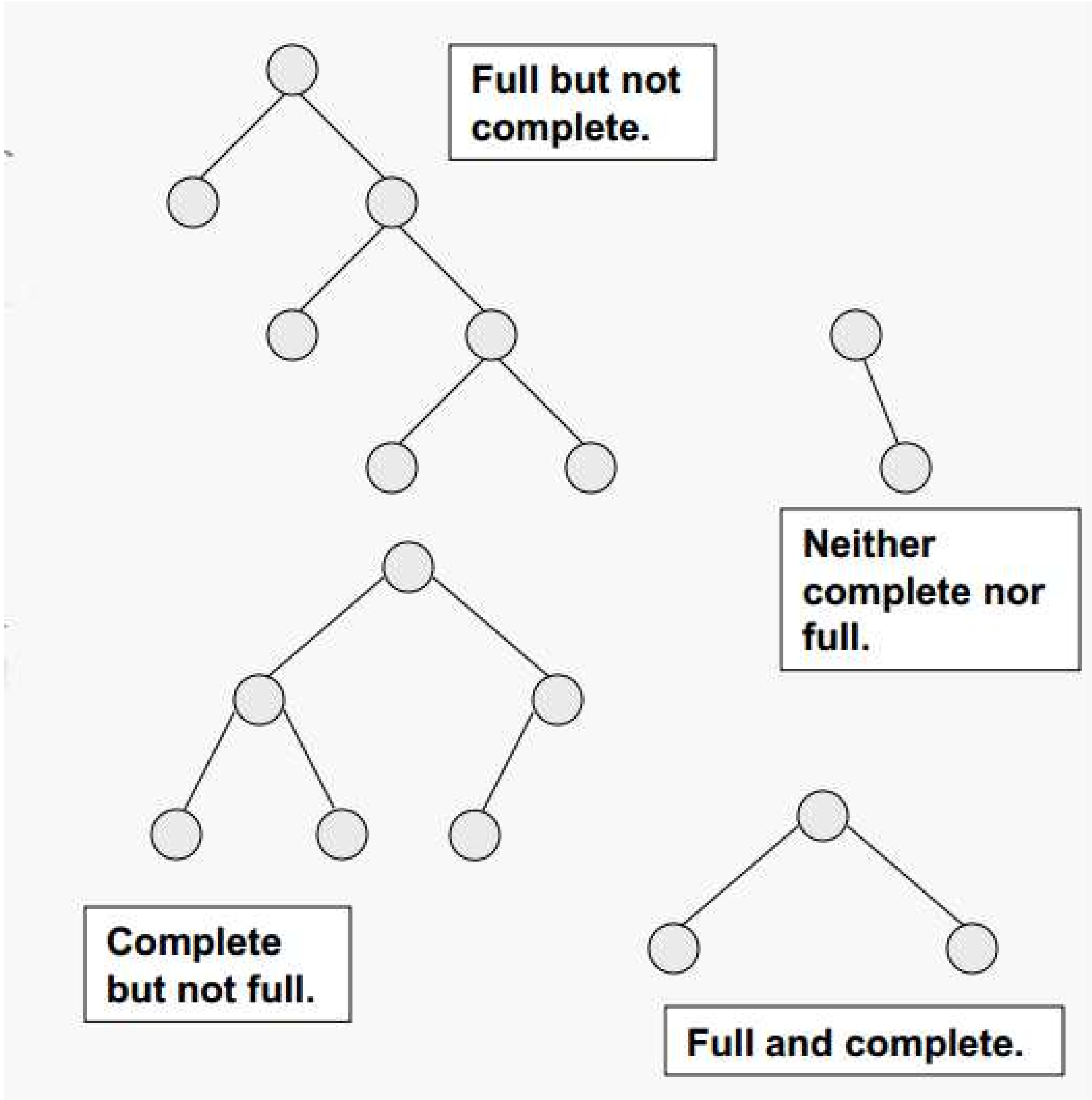






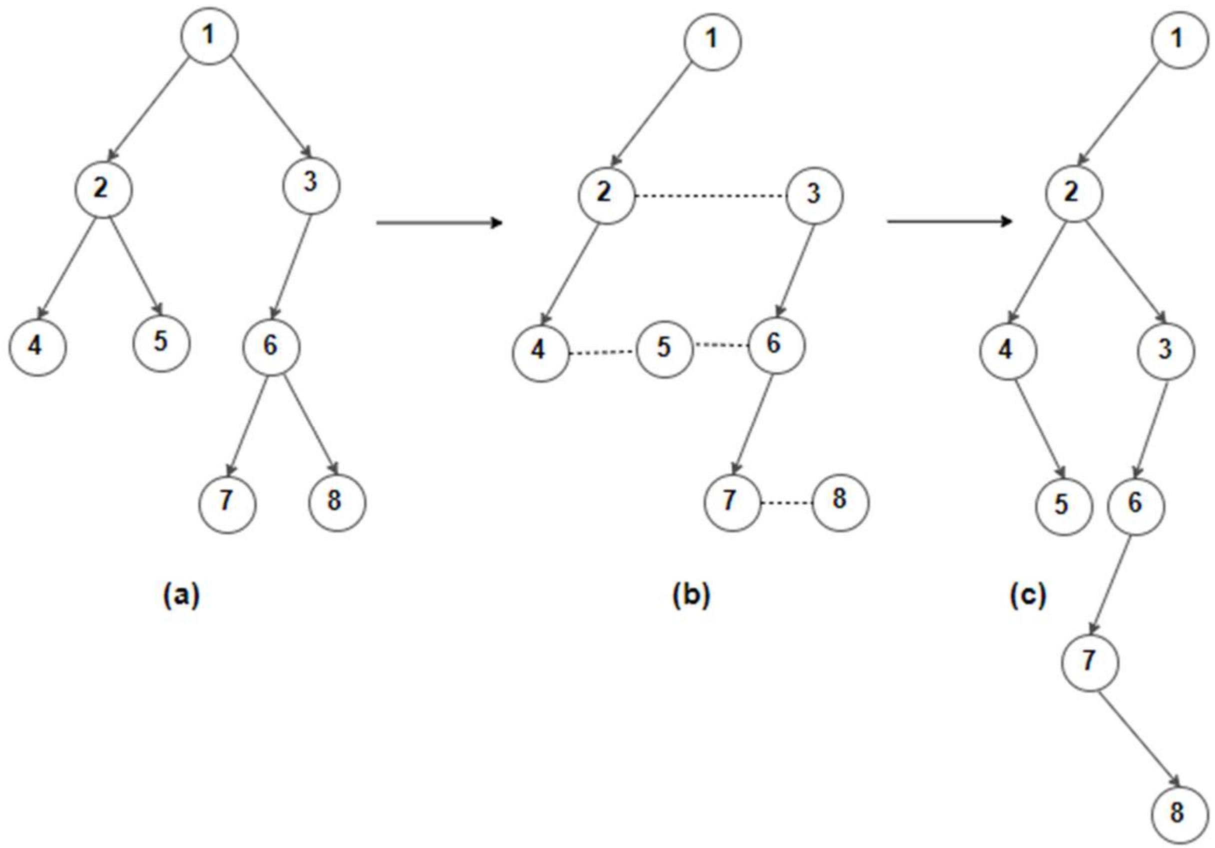


Examples

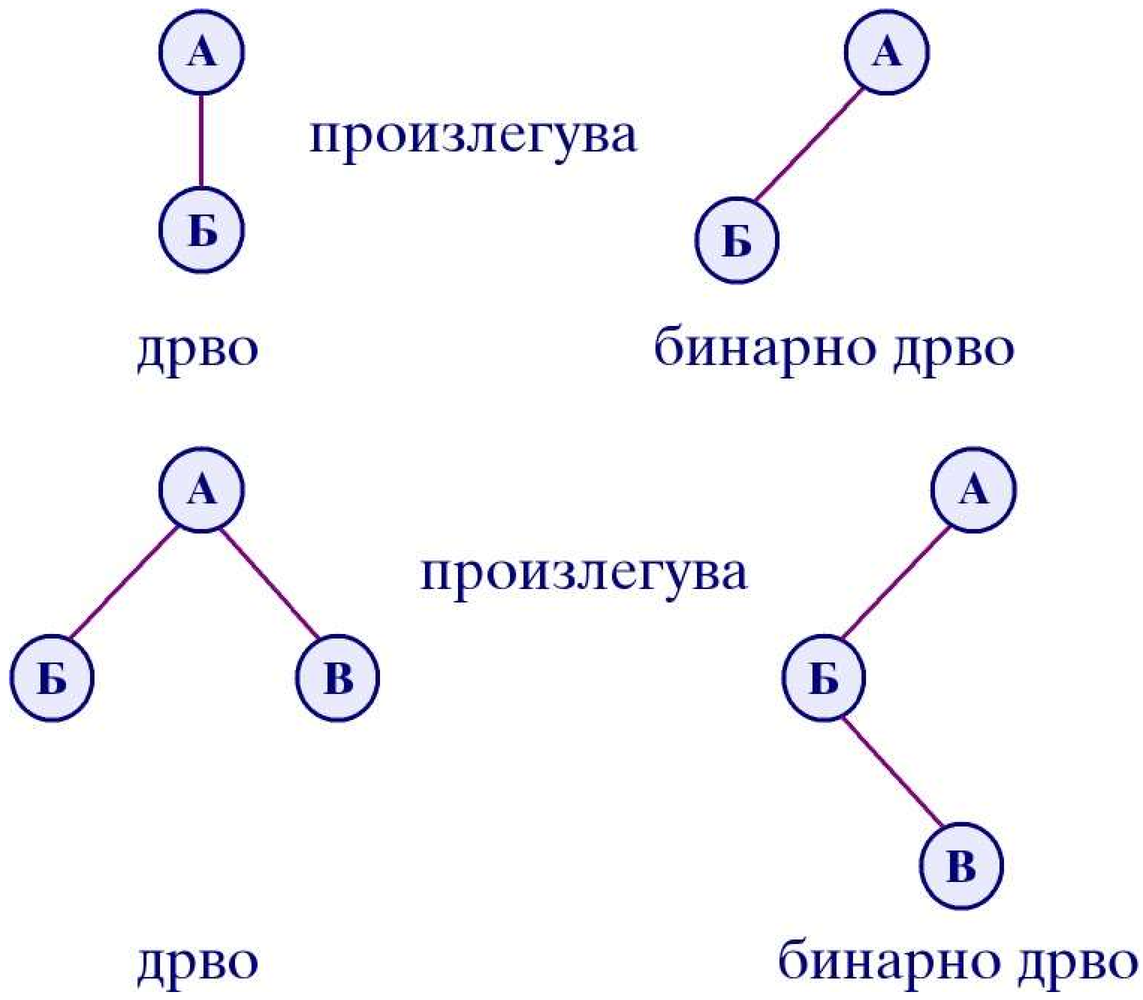

Full but not complete.
Neither complete nor full.
Complete but not full.
Full and complete.

Tree transformation in a binary tree

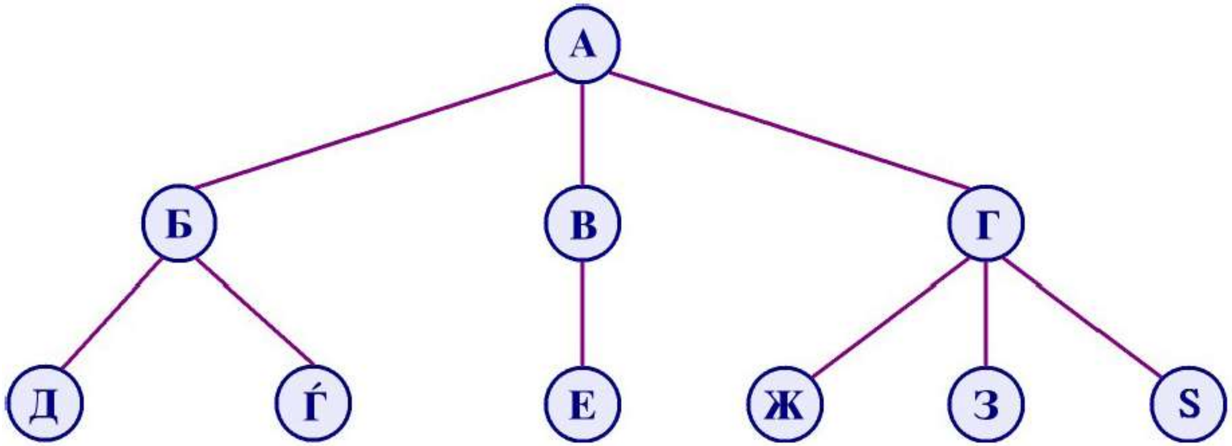
- Any tree can be transformed into a binary tree, so that one of the nodes at the same level becomes the parent of all the others.
- "right brother - left child" transformation



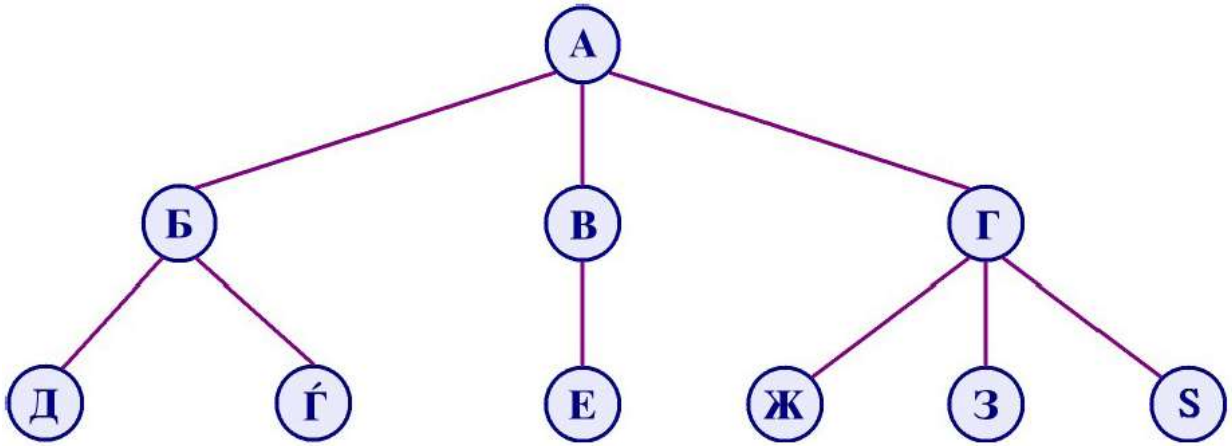
Simple trees representation with a binary tree

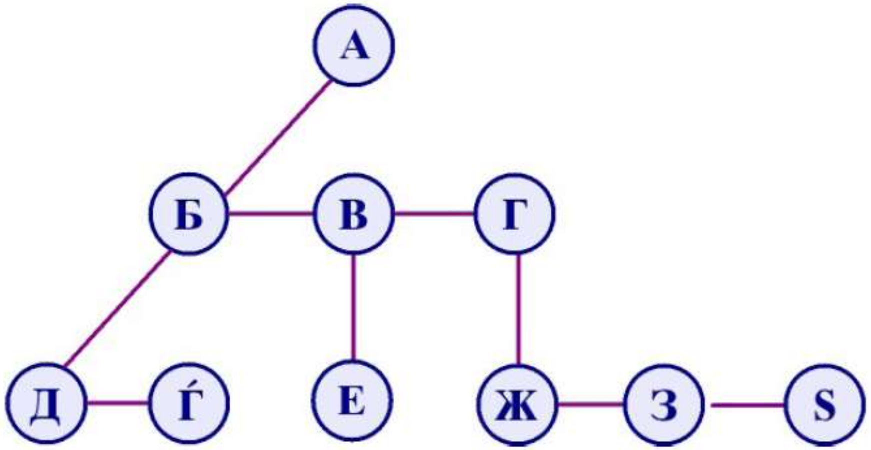


Tree transformation in a binary tree - example

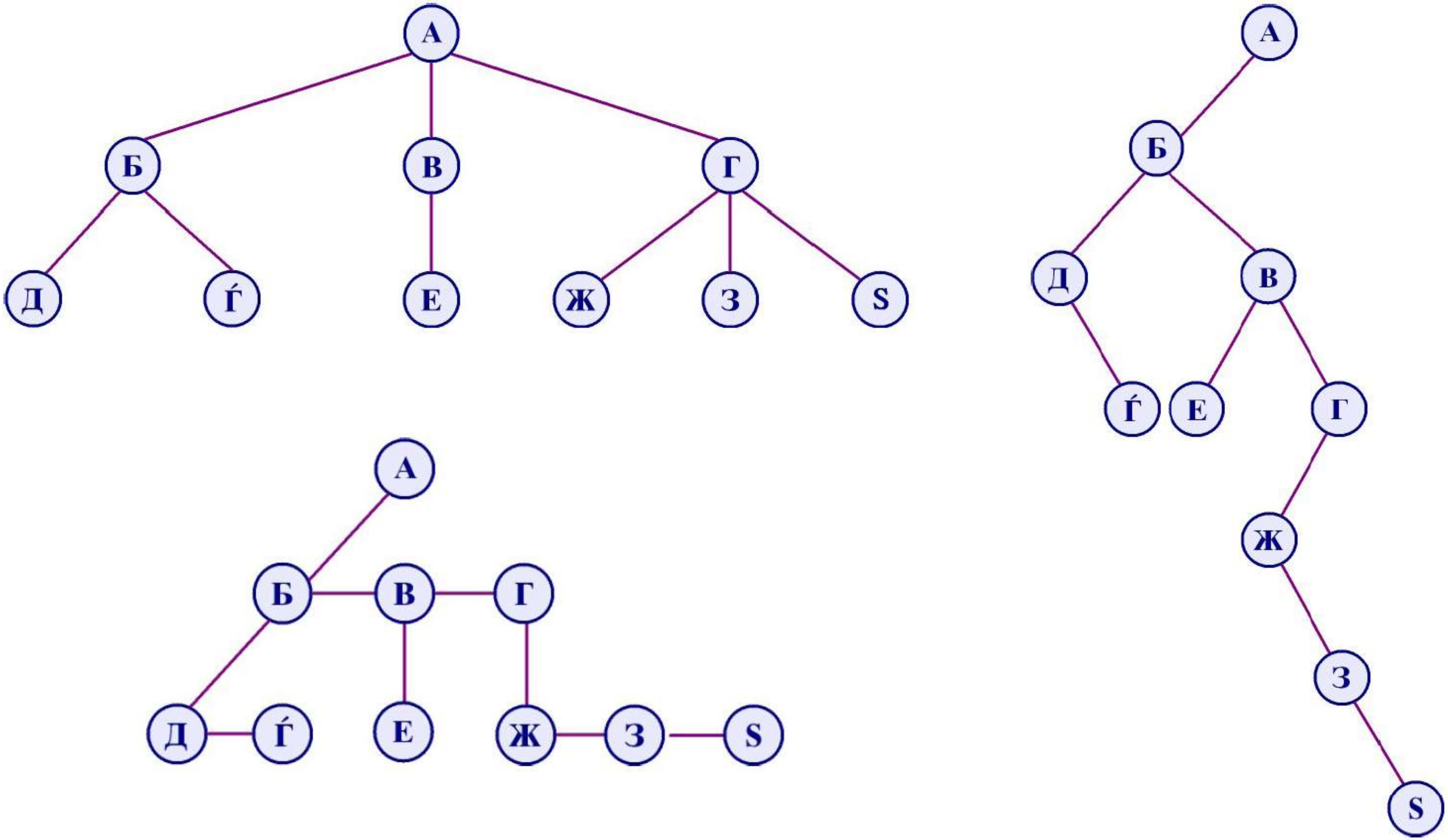


Tree transformation in a binary tree - example





Tree transformation in a binary tree - example

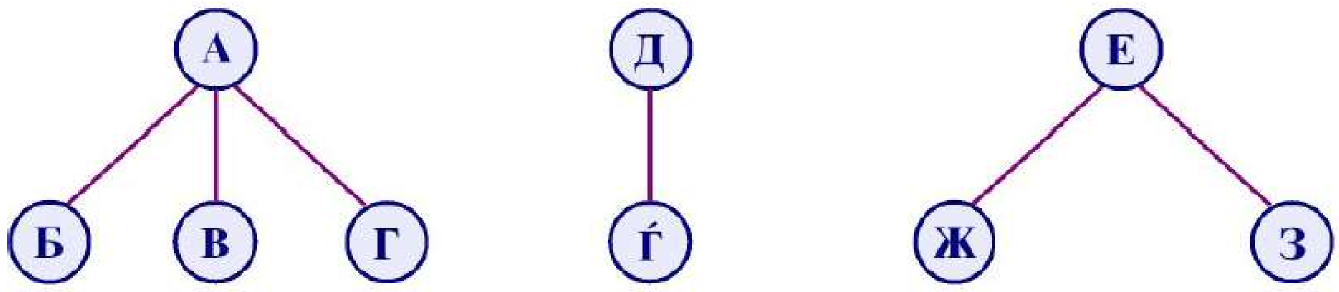


Transformation of a forest of trees in a binary tree

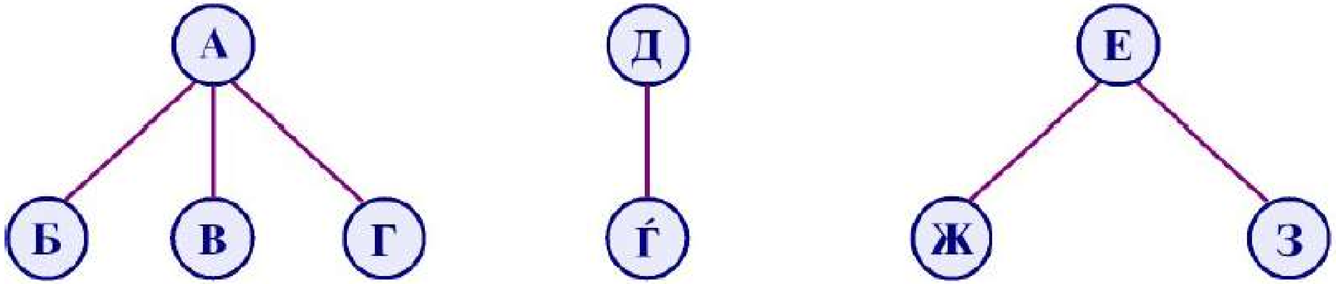
- Let $T_1, ..., T_n$ be a forest of trees, then the binary tree that is obtained with the transformation of this forest can be denoted with $B(T_1, ..., T_n)$ and for it holds:

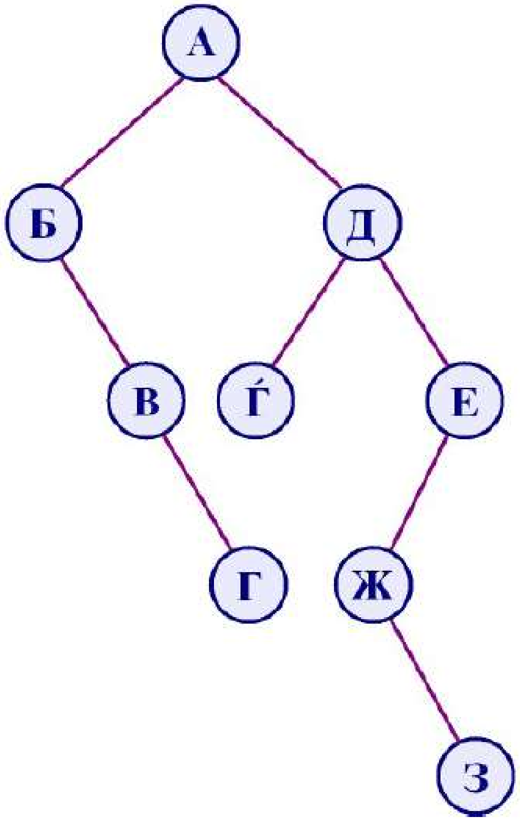
- $B(T_1, ..., T_n)$
  - is empty if $n = 0$;
  - has a root equal to the root of (T_1);
  - has left subtree $B(T_{11}, T_{12}, ..., T_{1m})$ where $T_{11}, ..., T_{1m}$ are the subtrees of the root (T_1);
  - has right subtree $B(T_2, ..., T_n)$

Transformation of a forest of trees in a binary tree



Transformation of a forest of trees in a binary tree



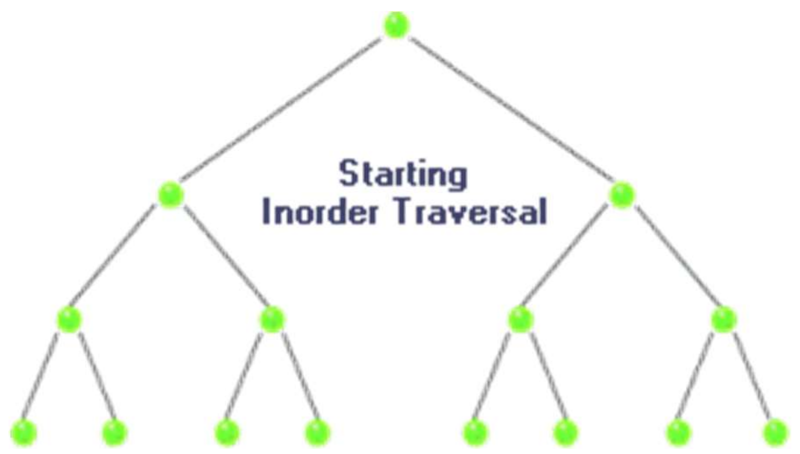


Binary tree traversal

- There are three main ways in which all nodes in a binary tree can be traversed:
  - inorder
  - preorder
  - postorder

Inorder binary tree traversal

- Traverse the left subtree, i.e. call Inorder(left child-> subtree)
- Visit the root
- Traverse the right subtree, i.e. call Inorder(right child-> subtree)



Preorder binary tree traversal

- Visit the root
- Traverse the left subtree, i.e. call Preorder(left child-> subtree)
- Traverse the right subtree, i.e. call Preorder(right child-> subtree)

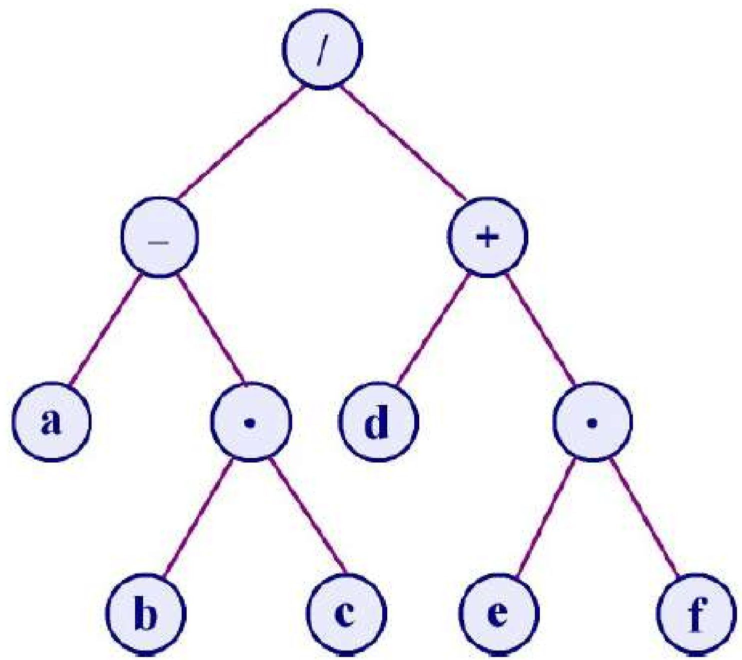
Postorder binary tree traversal

- Traverse the left subtree, i.e. call Postorder(left child-> subtree)
- Traverse the right subtree, i.e. call Postorder(right child-> subtree)
- Visit the root

Binary tree traversal

- There are three main ways in which all nodes in a binary tree can be traversed:
  - inorder
  - preorder
  - postorder

Example: Binary tree for arithmetic expressions representation and ccomputation

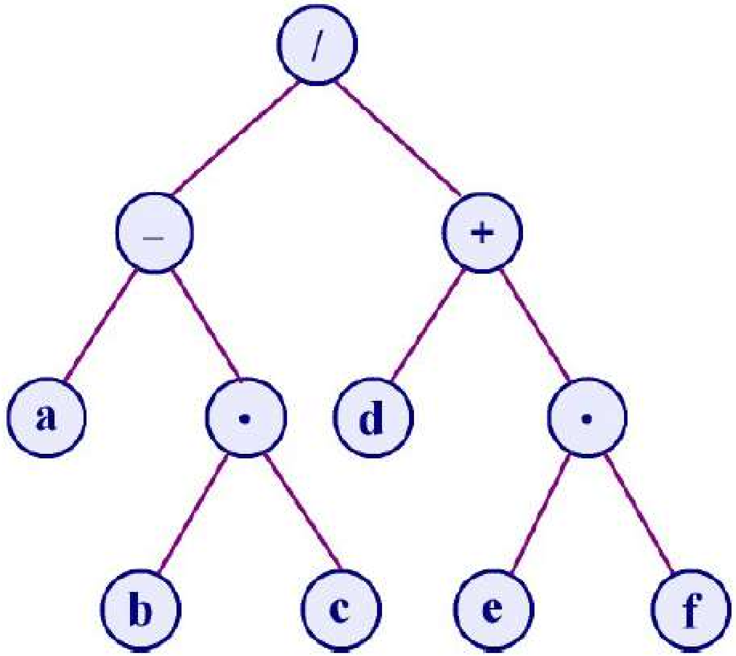


Binary tree traversal

- There are three main ways in which all nodes in a binary tree can be traversed:
  - inorder
  - preorder
  - postorder

a - b * c / d + e * f

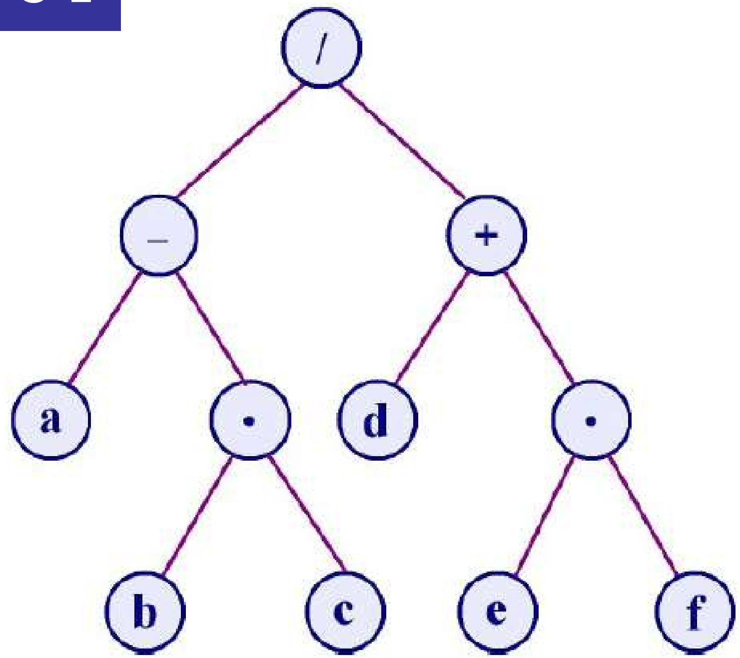
Example: Binary tree for arithmetic expressions representation and ccomputation



Binary tree traversal

- There are three main ways in which all nodes in a binary tree can be traversed:
  - inorder a - b * c / d + e * f
  - preorder / - a * b c + d * e f
  - postorder

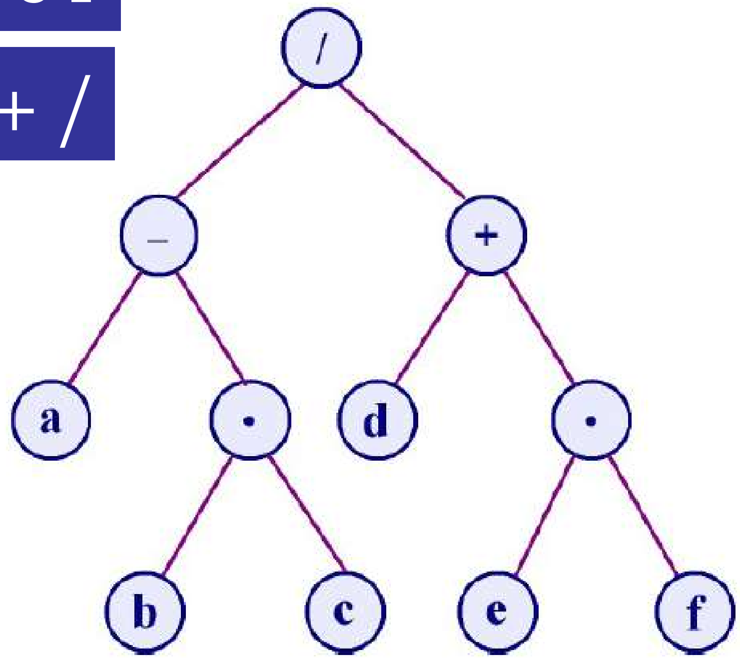
Example: Binary tree for arithmetic expressions representation and ccomputation



Binary tree traversal

- There are three main ways in which all nodes in a binary tree can be traversed:
  - inorder   a - b * c / d + e * f
  - preorder   / - a * b c + d * e f
  - postorder   a b c * - d e f * + /

Example: Binary tree for arithmetic expressions representation and ccomputation



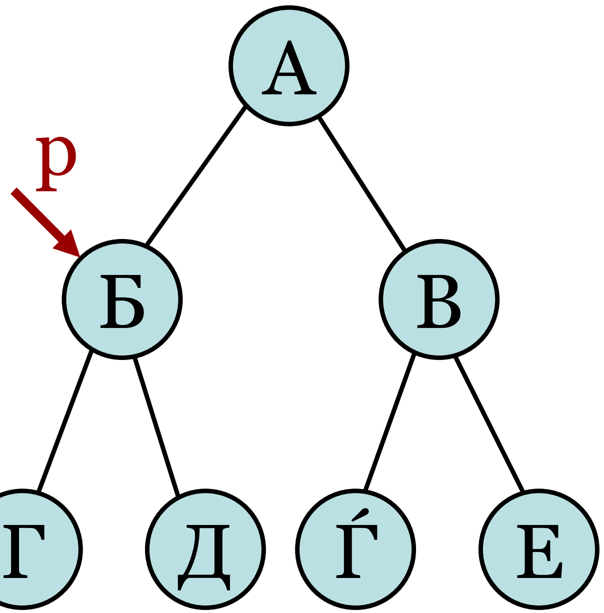
Binary tree traversal

Recursive realizations of these traversals are trivial!!

Basic operations with nodes

- Insertion of a node in a binary tree

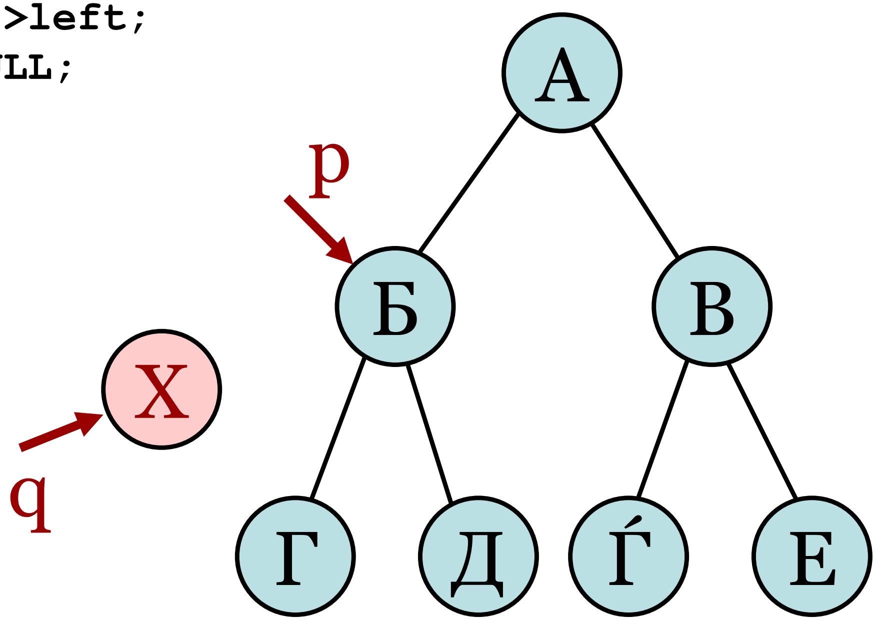
```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```



Basic operations with nodes

- Insertion of a node in a binary tree

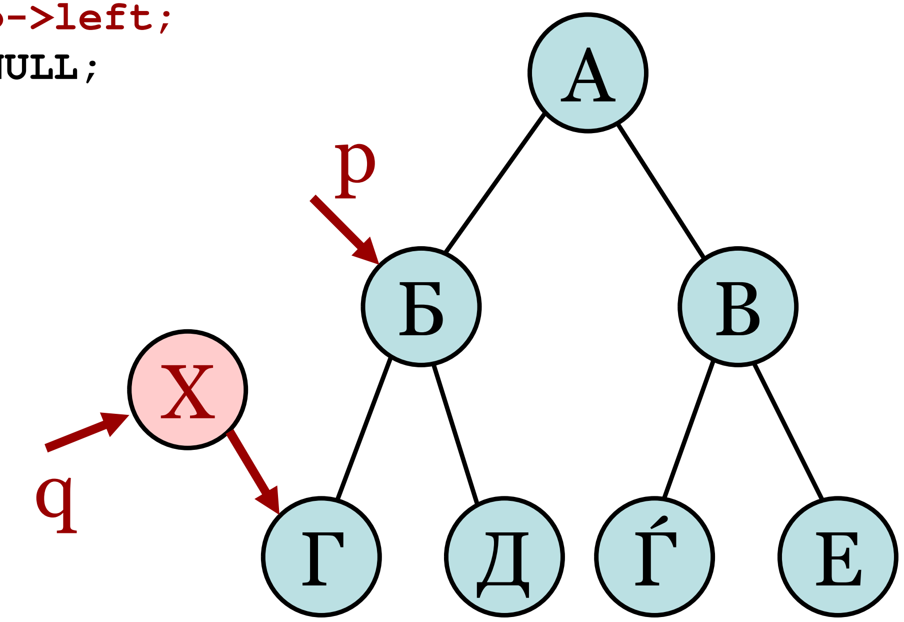
```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```



Basic operations with nodes

- Insertion of a node in a binary tree

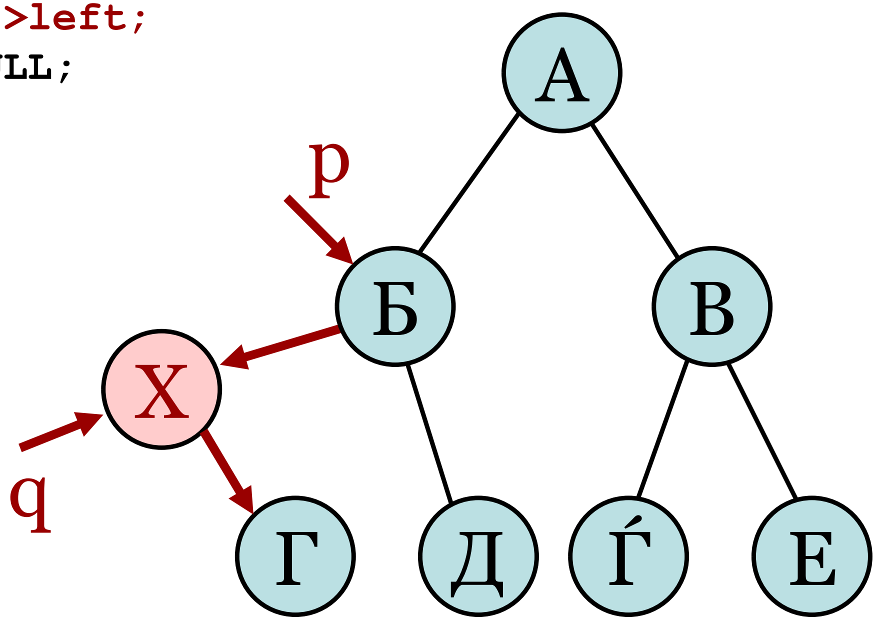
```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```



Basic operations with nodes

- Insertion of a node in a binary tree

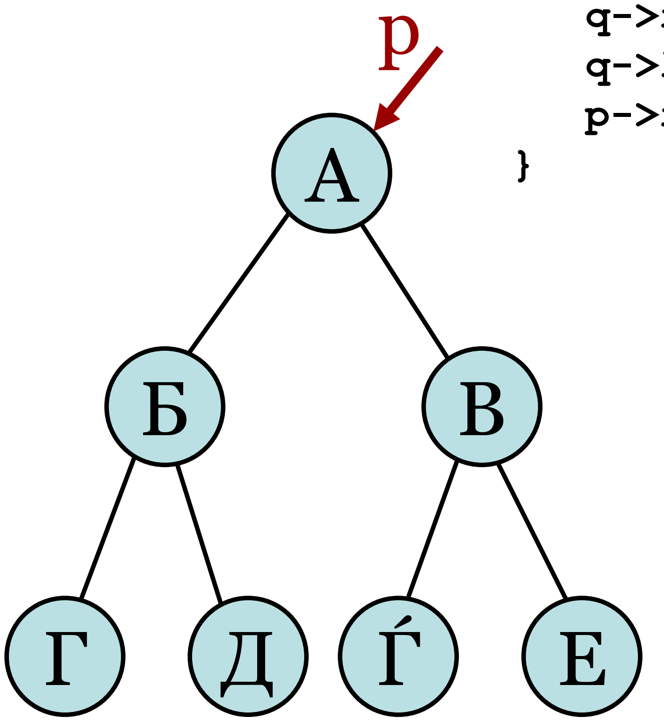
```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```



Basic operations with nodes

- Insertion of a node in a binary tree

```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```



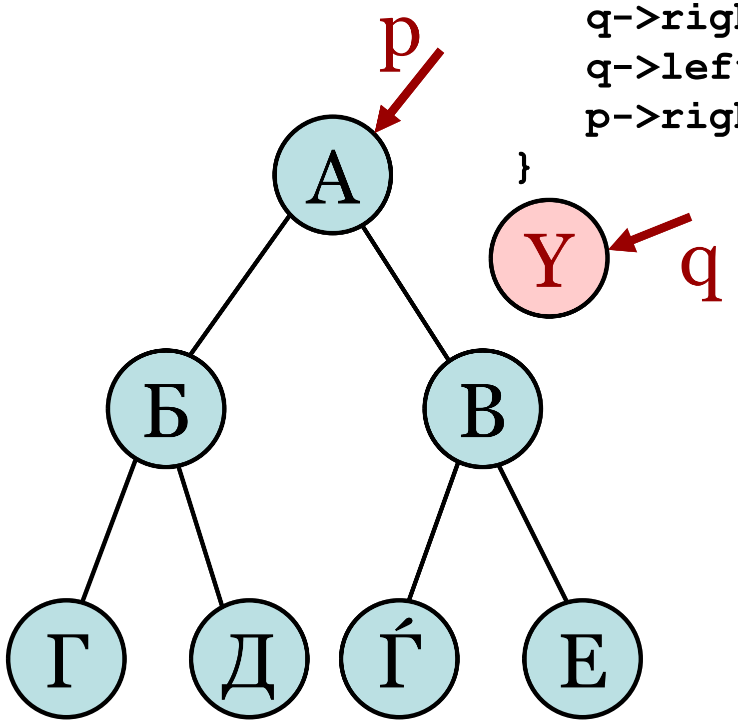

```
void vmetni_desno(nodep p, info_t y)
{
   nodep q=NEW(node);
   q->info=y;
   q->right = p->right;
   q->left=NULL;
   p->right=q;
}
```

Basic operations with nodes

- Insertion of a node in a binary tree

```
void vmetni_levo(nodep p, info_t x)
{
    nodep q=NEW(node);
    q->info=x;
    q->left= p->left;
    q->right=NULL;
    p->left=q;
}
```

```
void vmetni_desno(nodep p, info_t y)
{
    nodep q=NEW(node);
    q->info=y;
    q->right = p->right;
    q->left=NULL;
    p->right=q;
}
```

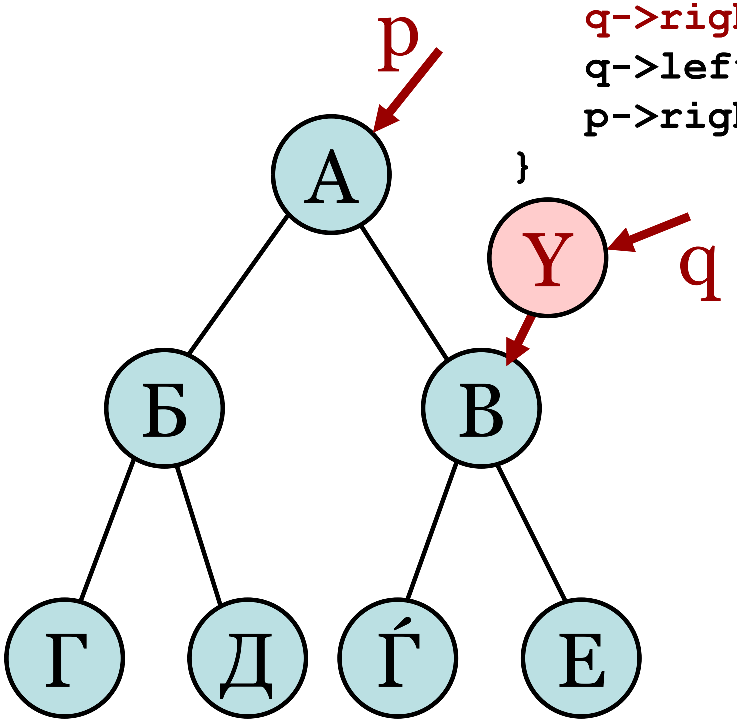


Basic operations with nodes

- Insertion of a node in a binary tree

```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```

```
void vmetni_desno(nodep p, info_t y)
{
   nodep q=NEW(node);
   q->info=y;
   q->right = p->right;
   q->left=NULL;
   p->right=q;
}
```

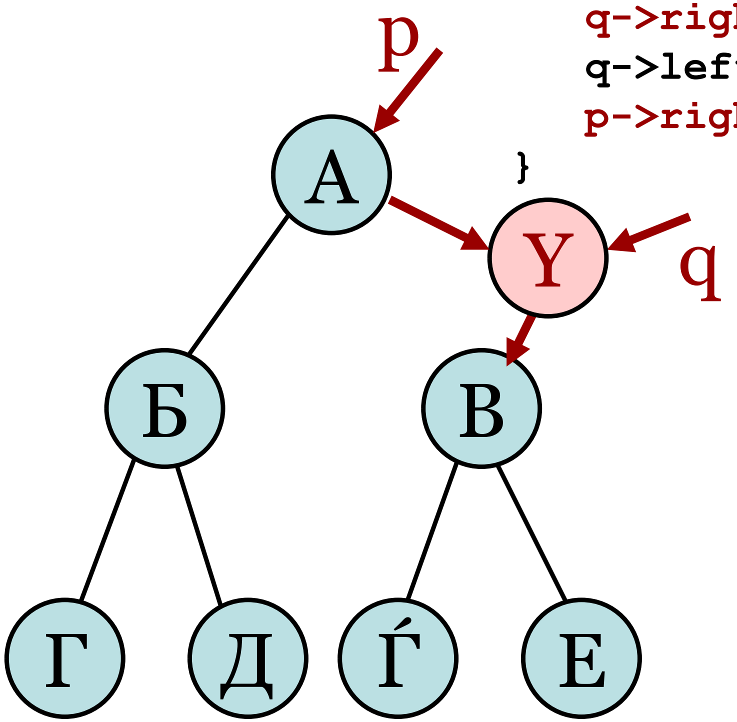


Basic operations with nodes

- Insertion of a node in a binary tree

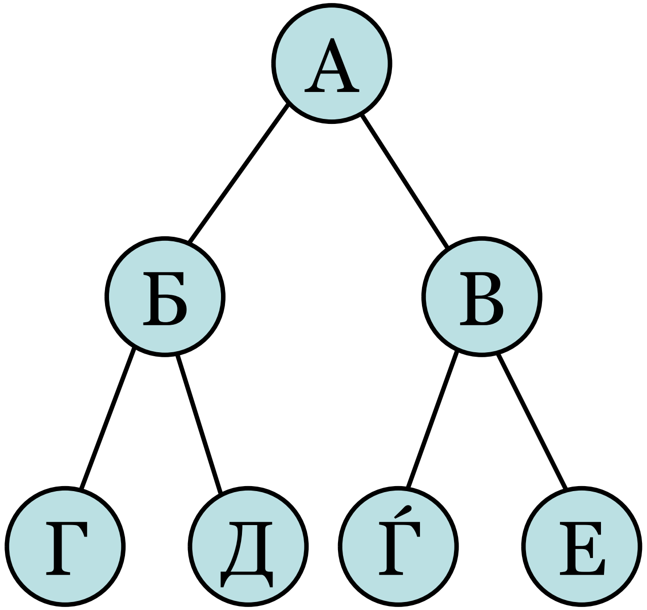
```
void vmetni_levo(nodep p, info_t x)
{
   nodep q=NEW(node);
   q->info=x;
   q->left= p->left;
   q->right=NULL;
   p->left=q;
}
```

```
void vmetni_desno(nodep p, info_t y)
{
   nodep q=NEW(node);
   q->info=y;
   q->right = p->right;
   q->left=NULL;
   p->right=q;
}
```



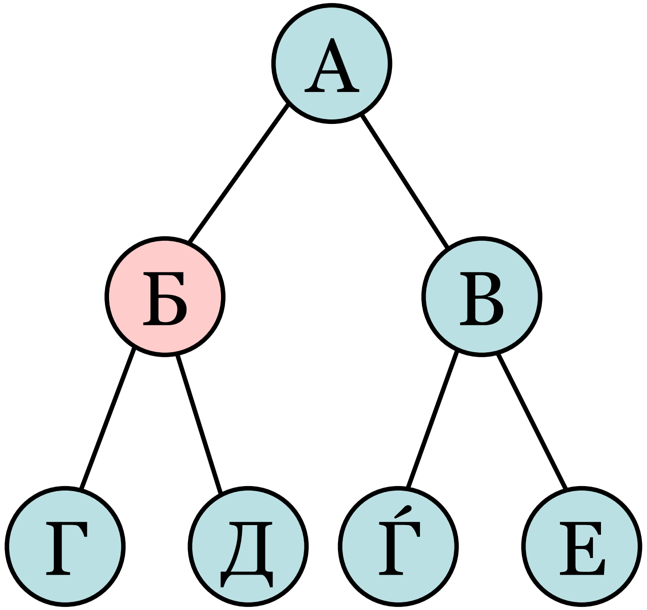
Basic operations with nodes

- Deletion of a node in a binary tree



Basic operations with nodes

- Deletion of a node in a binary tree



Problem: What will replace the deleted node so that we have a binary tree structure again?

Solution: the deleted node should be replaced by another node (usually one from its subtree). When the node being deleted has no children or only one child, the algorithm is trivial.