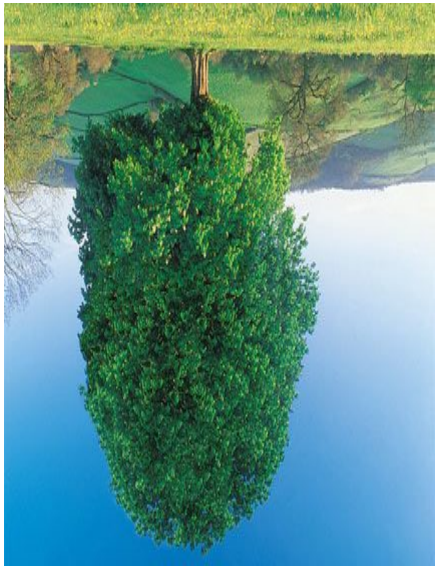
Introduction to Trees

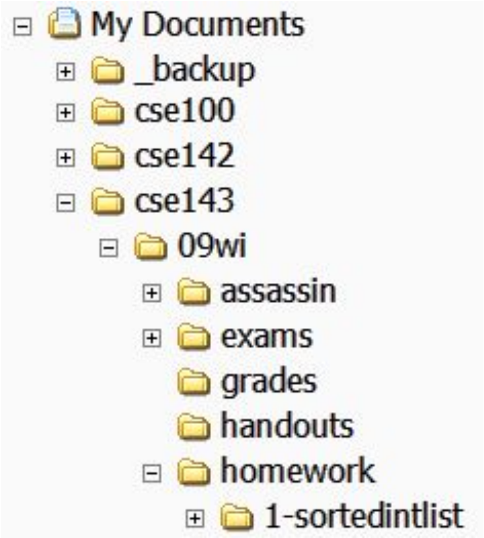
Chapter 8

Trees in computer science

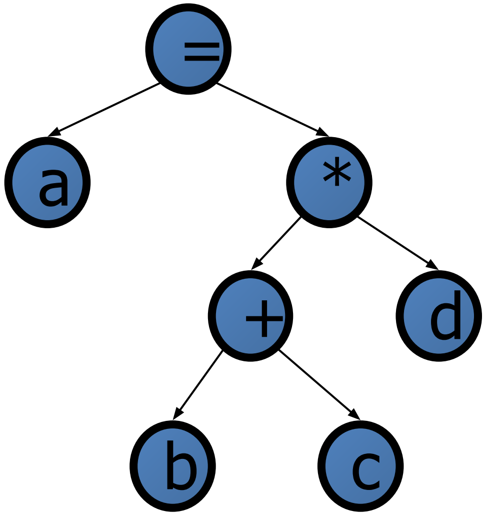
- folders/files on a computer
- family genealogy; organizational charts
- AI: decision trees
- compilers: parse tree

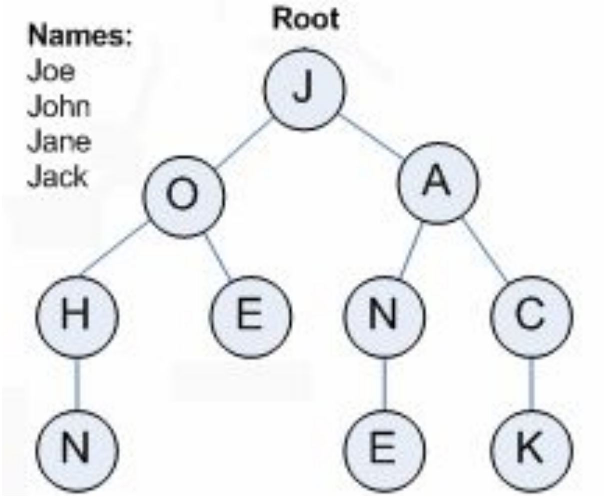
  a = (b + c) * d;
- cell phone T9





My Documents
_backup
cse100
cse142
cse143
09wi
assassin
exams
grades
handouts
homework
1-sortedintlist





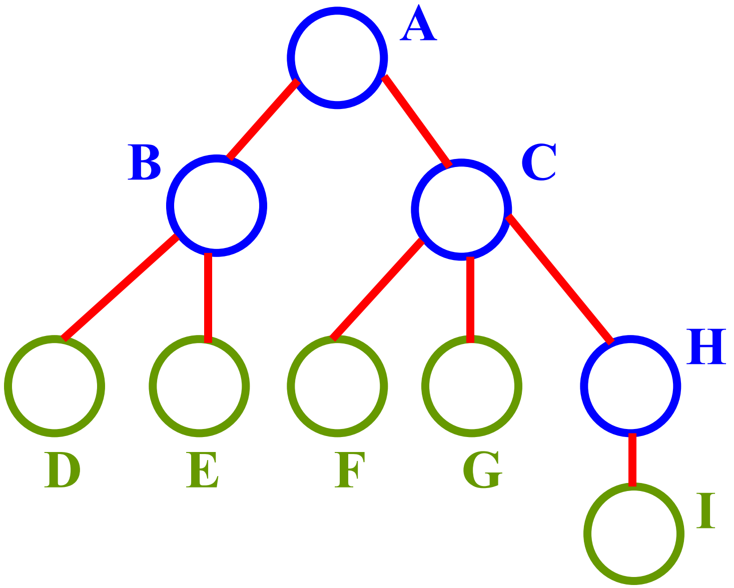
Towards Non-Linear Data Structures

- The data structures we have studied so far are linear; an element is followed by exactly one element
- The data can also be represented in a non-linear fashion
  - An important concept is a family like structure; this structure is called a tree

Tree

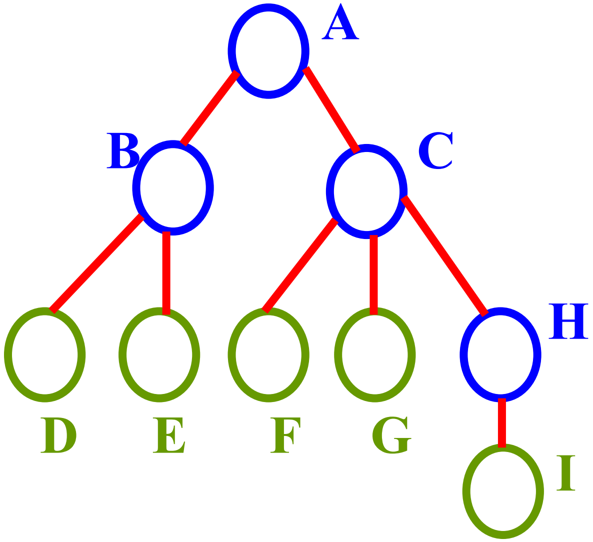
❑ A tree is a hierarchical data structure which consists of a set of nodes connected through edges

❑ Note: A can be followed by B or C.



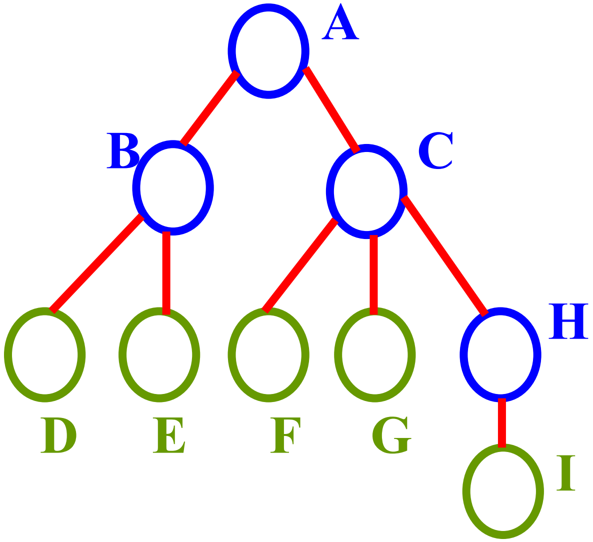
Terminology (1)

- ❑ Node: is a structure which normally contains a value, e.g. round boxes labeled as D,E, etc.
- ❑ Root: the top most node in the tree, e.g. A is root node
- ❑ Child Node: the roots of the subtrees of a node X are the children of X. e.g. B and C are children of A – A is parent of B and C



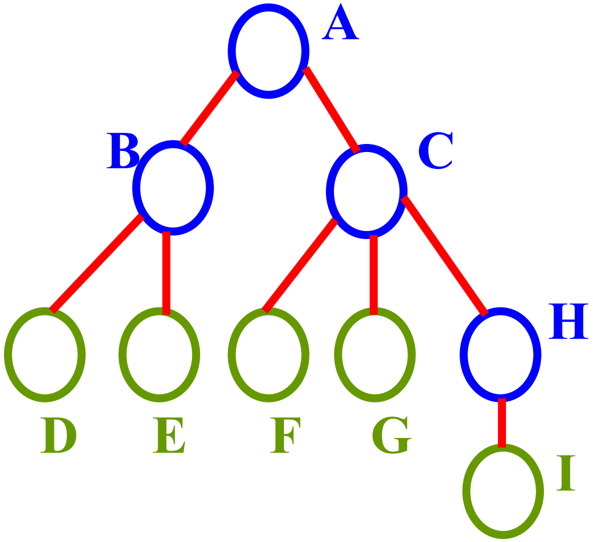
Terminology (2)

- ❑ Terminal nodes (leaf/external): nodes that have degree zero. OR nodes with no children. E.g. D, E
- ❑ Nonterminal/internal nodes: nodes that don't belong to terminal nodes. E.g. B, C



Terminology (3)

- ❑ Siblings: children of the same parent are said to be siblings. E.g. B and C are siblings, so is F and G.
- ❑ Ancestors of a node: all the nodes along the path from the root to that node. e.g. ancestors of I are I, H, C and A



Tree Traversal (1)

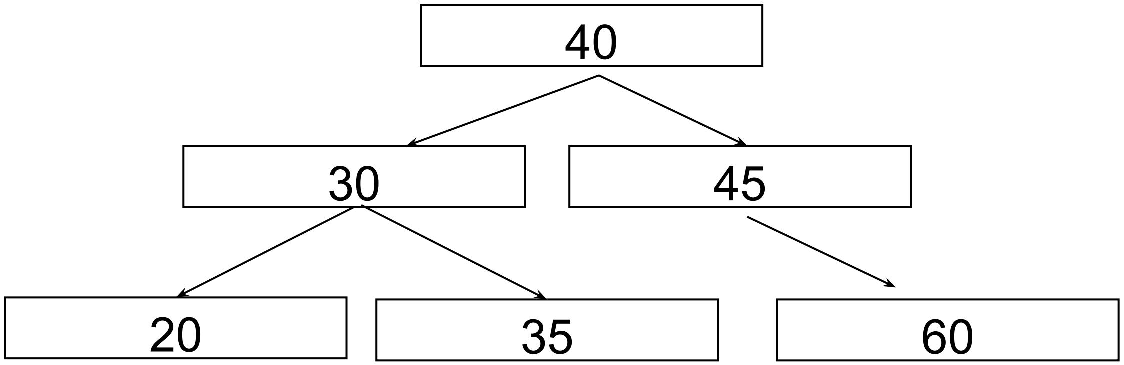
- What is traversal?
  - Traversal is the facility to move through a structure, visiting **each** of the nodes **exactly** once
- Which of the following is not traversal?
  1. Bisha □ Abaha □ Jeddah □ Riadh
  2. Bisha □ Abaha □ Jeddah □ Bisha □ Riadh (A repeated visit to Bisha – not allowed)

Tree Traversal (2)

- Pre-order Traversal
- Post-order Traversal
- In-order Traversal
- Notion
  - P: Visit the parent node
  - L: Visit the left subtree
  - R: Visit the right subtree

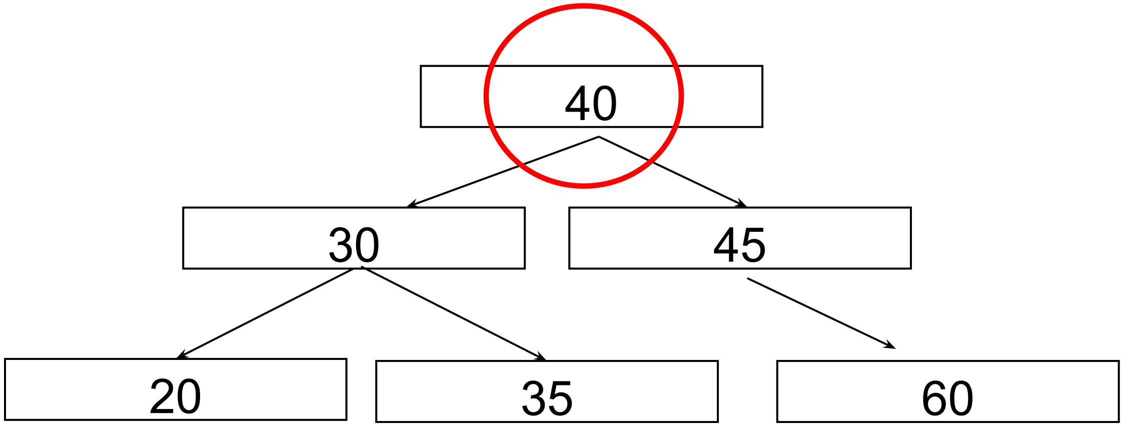
Pre-order Traversal (1)

- PLR, i.e.,
  - First, visit the parent node
  - Then, visit the left subtree (in pre-order)
  - Then, visit the right subtree (in pre-order)



Pre-order Traversal (2)

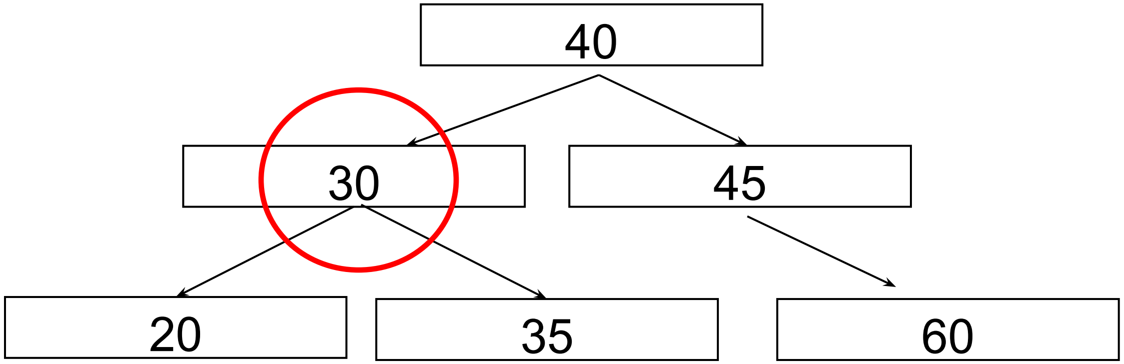
Step 1: root = 40, so display it, then traverse its left subtree (root = 40) and then right subtree (root = 45)



Display: 40

Pre-order Traversal (3)

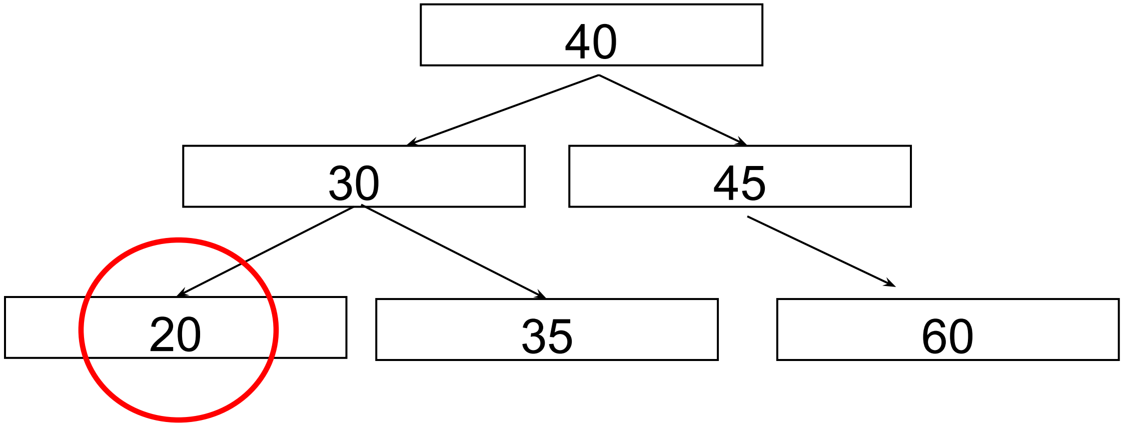
Step 2: root = 30, so display it, then traverse its left subtree (root = 20) and then right subtree (root = 35)



Display: 40 30

Pre-order Traversal (4)

Step 3: root = 20, so display it, then traverse its left subtree (root = null) and then right subtree (root = null)

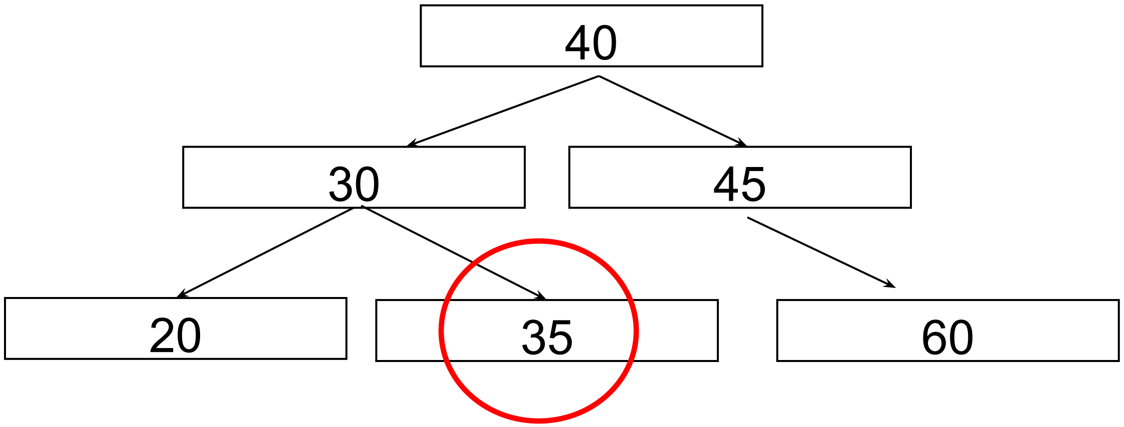


Display: 40 30 20

Since node with value 20 is a leaf node, we finished traversing this subtree (root = 20), which is a left subtree of node with value 30. So, in the next step we'll traverse the right subtree of 30.

Pre-order Traversal (5)

Step 4: root = 35, so display it, then traverse its left subtree (root = null) and then right subtree (root = null)

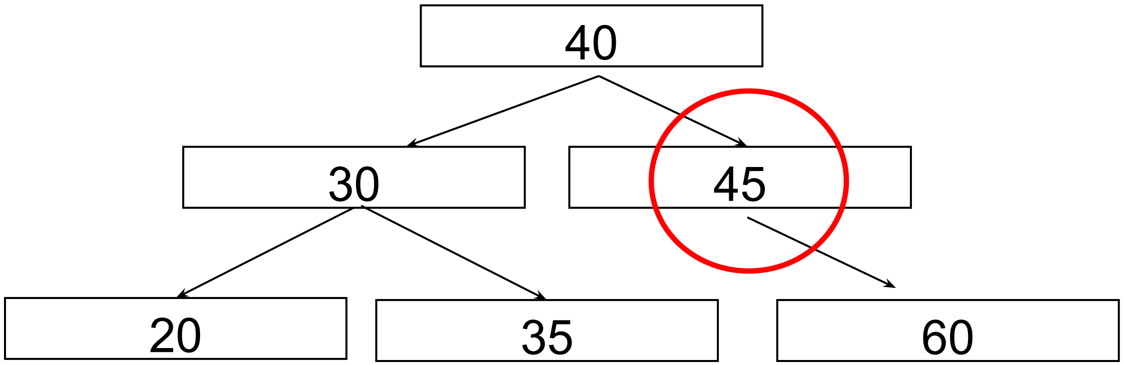


Display: 40 30 20 35

Since node with value 35 is a leaf node, we finished traversing this subtree (root = 35), which is a right subtree of node with value 30. So, in the next step we'll traverse the right subtree of 40.

Pre-order Traversal (6)

Step 5: root = 45, so display it, then traverse its left subtree (root = null) and then right subtree (root = 60)

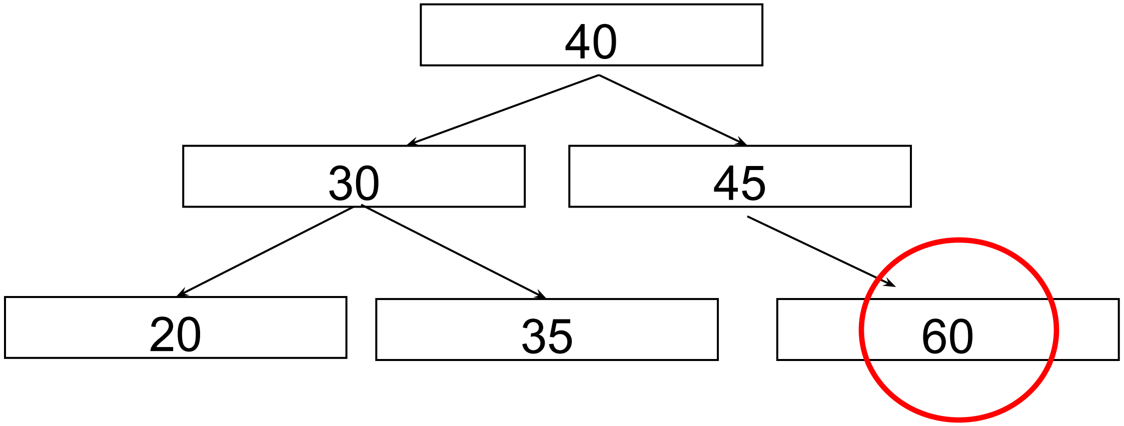


Display: 40 30 20 35 45

Since node with value 45 has no left subtree but a right subtree (root = 60), in the next step we'll traverse this subtree (root = 60).

Pre-order Traversal (7)

Step 6: root = 60, so display it, then traverse its left subtree (root = null) and then right subtree (root = null)

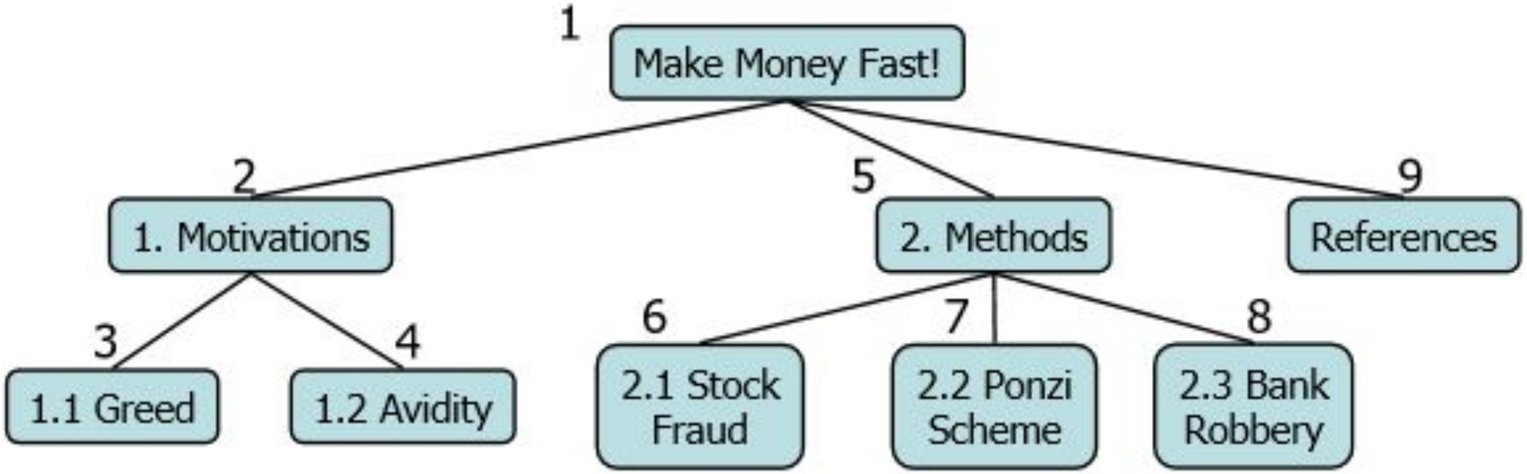


Display: 40 30 20 35 45 60

Finished!

Pre-order Traversal (8)

- In a preorder traversal, a node is visited before its descendants
- **Application**: print a structured document

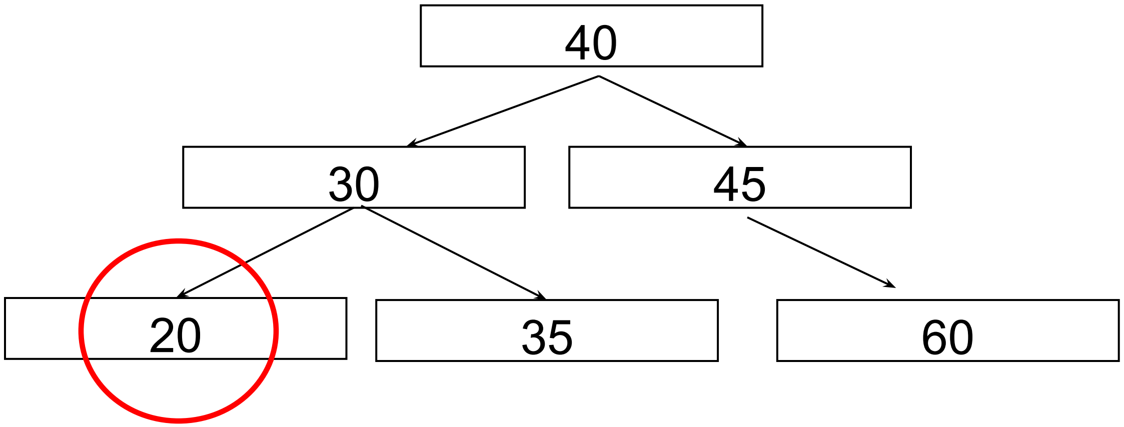


Post-order Traversal (1)

- LRP, i.e.,
  - First, visit the left subtree (in post-order)
  - Then, visit the right subtree (in post-order)
  - Then, visit the parent

Post-order Traversal (2)

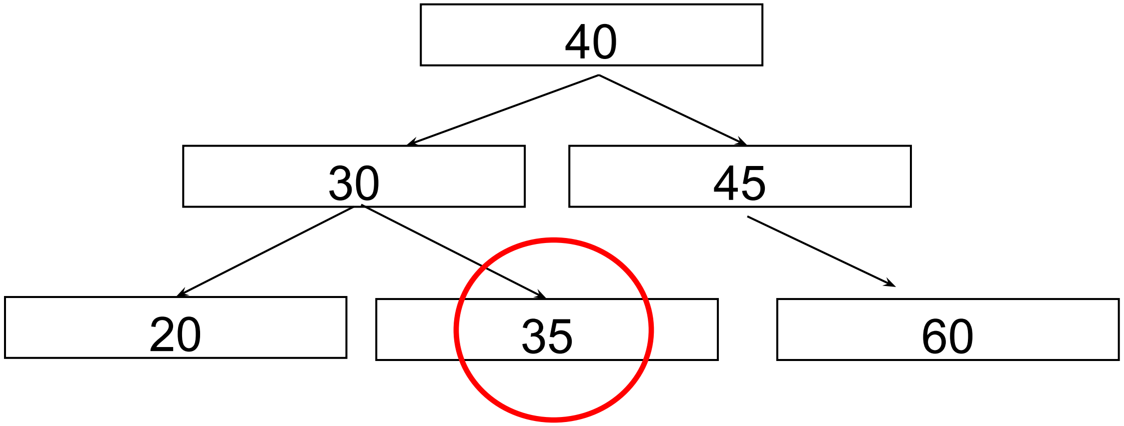
Step 1:



Display: 20

Post-order Traversal (3)

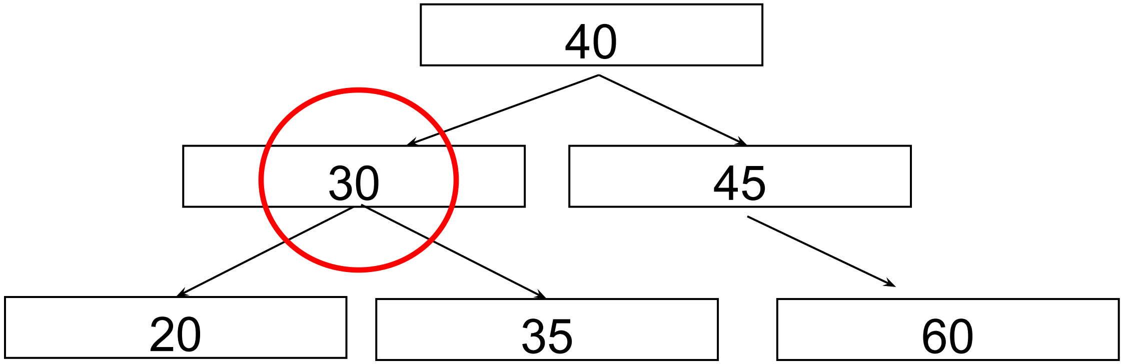
Step 2:



Display: 20 35

Post-order Traversal (4)

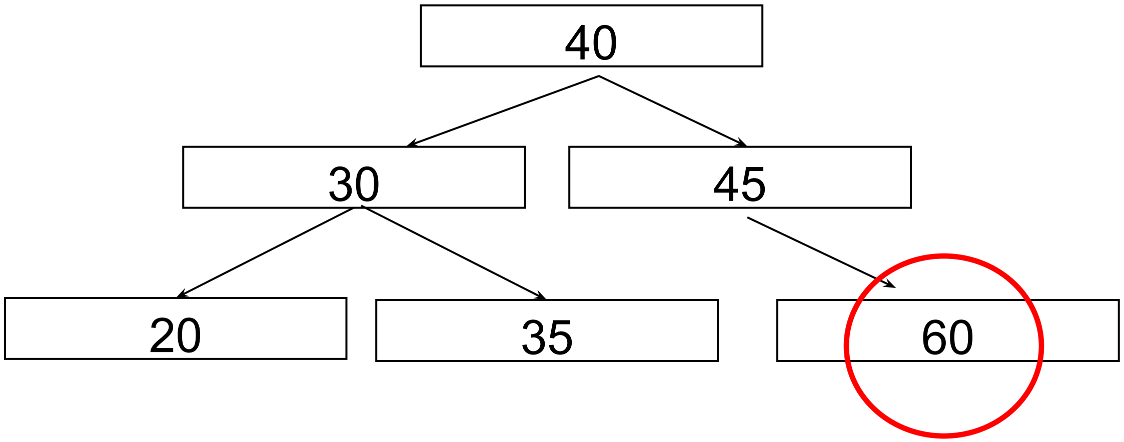
Step 3:



Display: 20 35 30

Post-order Traversal (5)

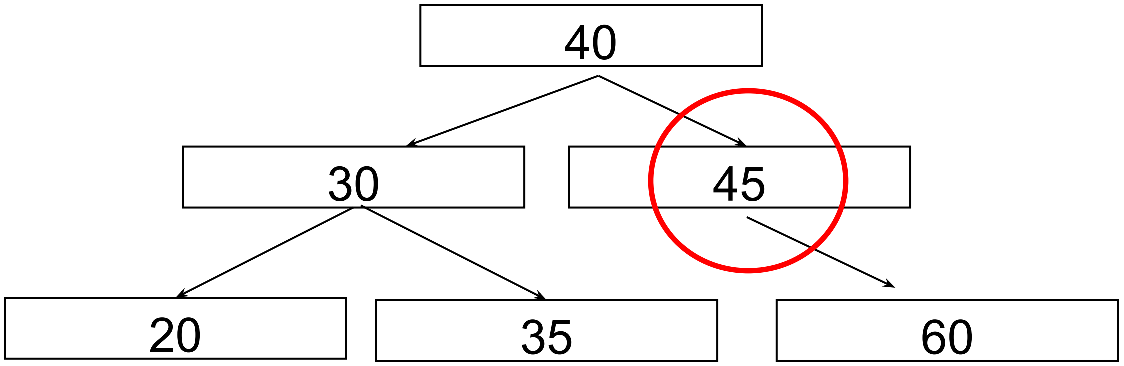
Step 4: Note that the node with value 45 has no left subtree!



Display: 20 35 30 60

Post-order Traversal (6)

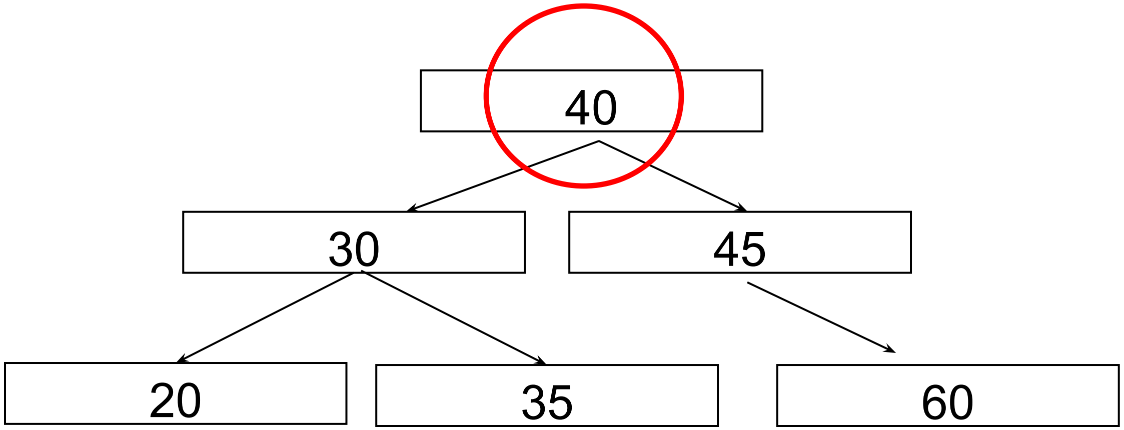
Step 5:



Display: 20 35 30 60 45

Post-order Traversal (7)

Step 6:

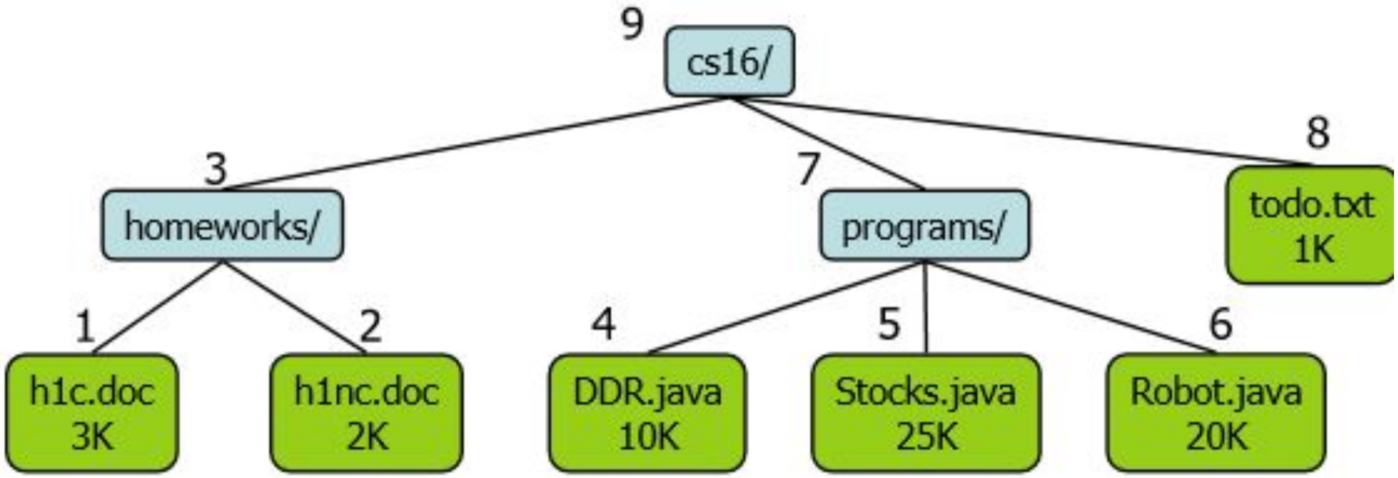


Display: 20 35 30 60 45 40

Finished!

Post-order Traversal (8)

- In a postorder traversal, a node is visited after its descendants
- **Application**: compute space used by files in a directory and its subdirectories

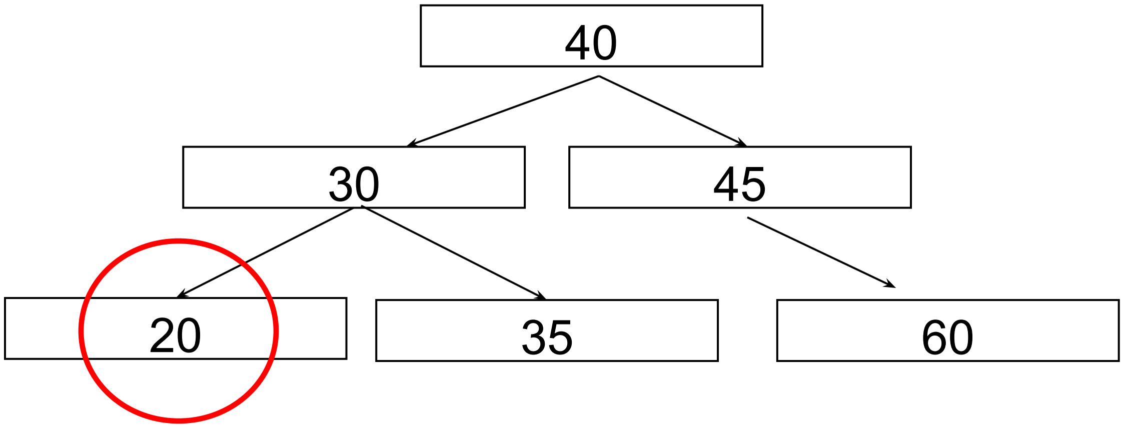


In-order Traversal (1)

- ❑ LPR, i.e.,
    - ▪ First, visit the left subtree (in in-order)
    - ▪ Then, visit the parent
    - ▪ Then, visit the right subtree (in in-order)

In-order Traversal (2)

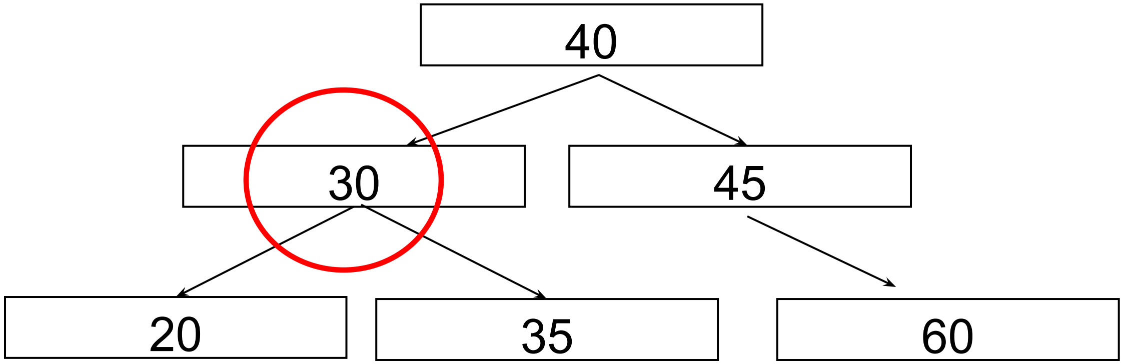
Step 1:



Display: 20

In-order Traversal (3)

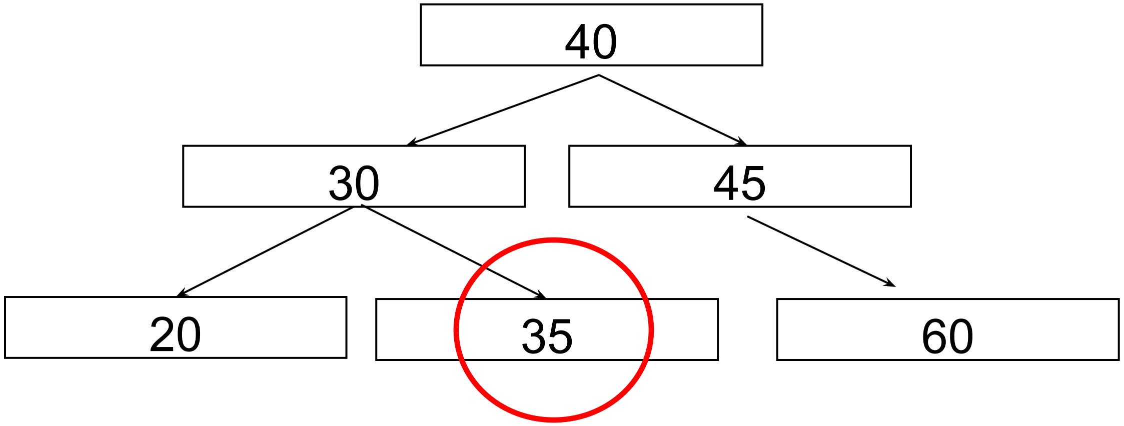
Step 2:



Display: 20 30

In-order Traversal (4)

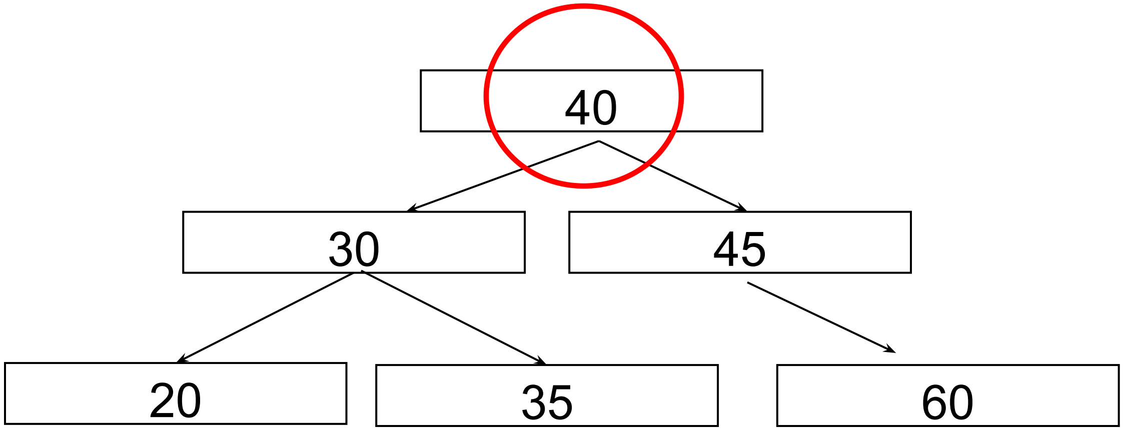
Step 3:



Display: 20 30 35

In-order Traversal (5)

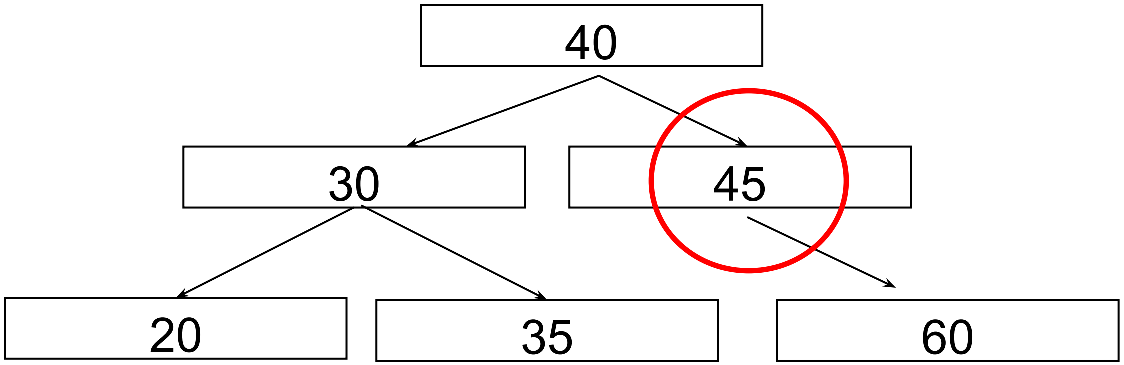
Step 4:



Display: 20 30 35 40

In-order Traversal (6)

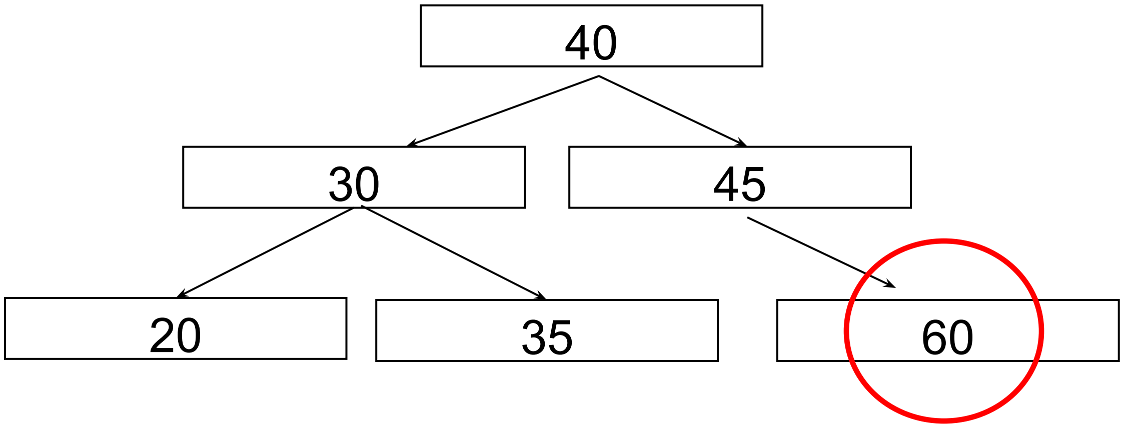
Step 5:



Display: 20 30 35 40 45

In-order Traversal (7)

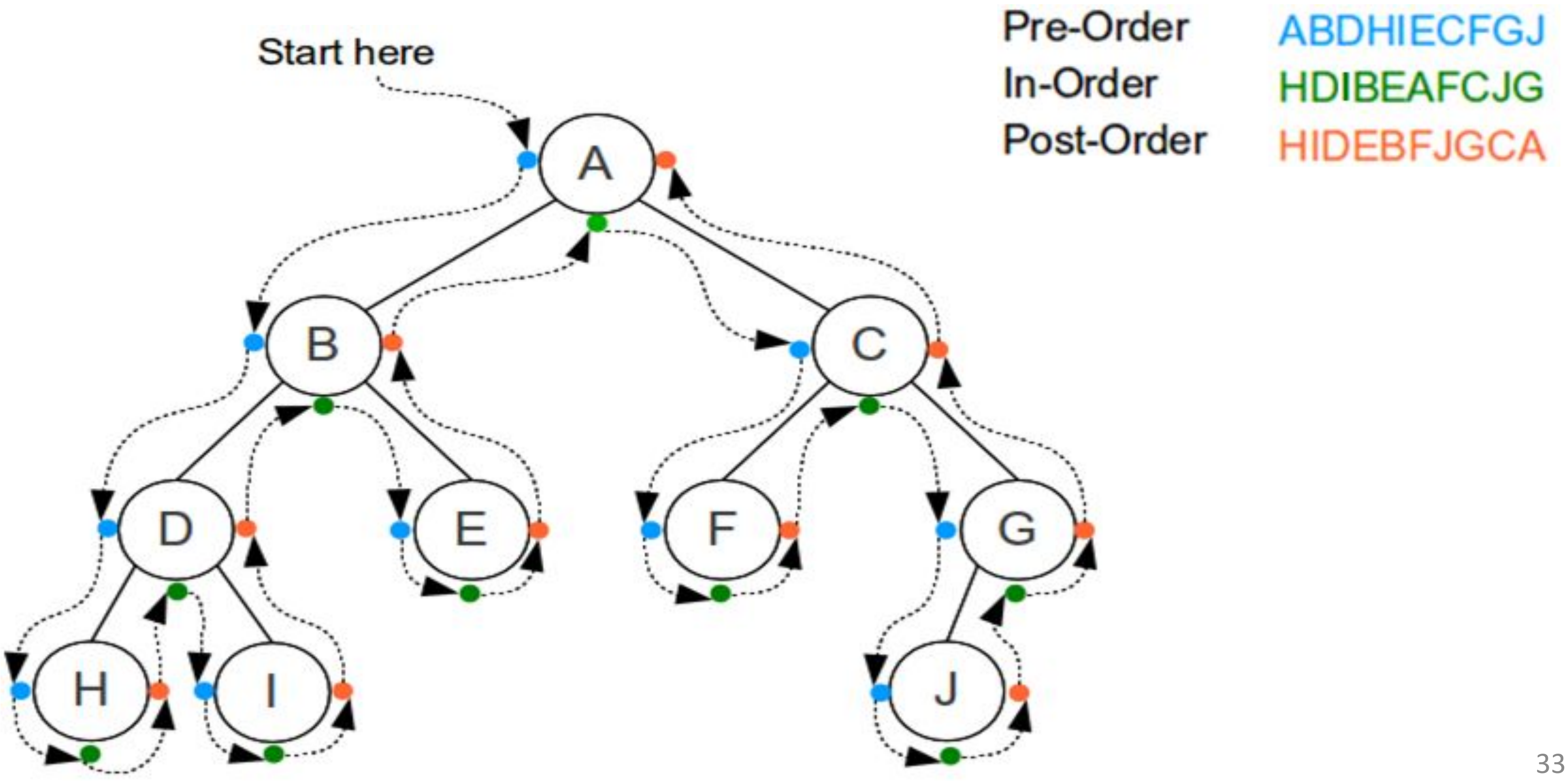
Step 6:



Display: 20 30 35 40 45 60

Example

The order in which the nodes are visited during a tree traversal can be easily determined by imagining there is a “ colored flag” attached to each node, as follows:



Pre-Order	ABDHIECFGJ
In-Order	HDIBEAFCJG
Post-Order	HIDEBFJGCA

Binary tree

- A binary tree is the most common kind of tree
  - Each node in a binary tree has at most two link instance variables
  - A binary tree must satisfy the Binary Search Tree Storage Rule
- The root of the tree serves a purpose similar to that of the instance variable **`head`** in a linked list
  - The node whose reference is in the **`root`** instance variable is called the *root node*
- The nodes at the "end" of the tree are called *leaf nodes*
  - Both of the link instance variables in a leaf node are **`null`**

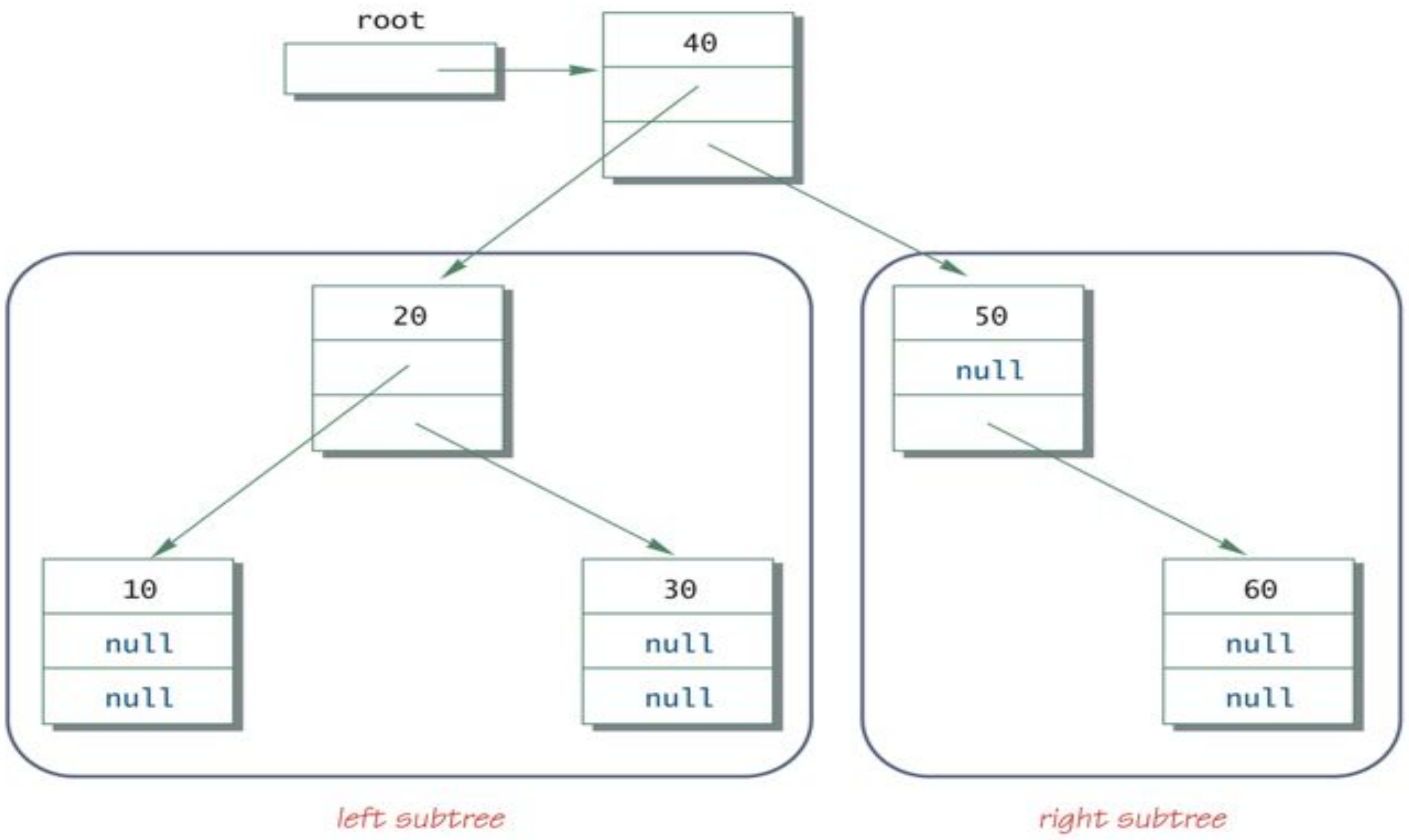
Binary Search Tree Property

- All the values in the left subtree must be less than the value in the root node
- All the values in the right subtree must be greater than or equal to the value in the root node
- This rule is applied recursively to each of the two subtrees

- Stored keys must satisfy the *binary search tree* property.
  - ∀ y in left subtree of x, then $key[y] \le key[x]$.
  - ∀ y in right subtree of x, then $key[y] \ge key[x]$.

Binary tree Example

A Binary Tree



Binary tree coding

```
public class BinaryTree {
 private int value;
 private BinaryTree leftChild;
 private BinaryTree rightChild;

// constructor
public BinaryTree(int x, BinaryTree l, BinaryTree r) {
     value = x;
     leftChild = l;
     rightChild = r;
}

// accessors
public int getValue() {
return(value);
}
```

```
public BinaryTree getLeftSubTree() {
return(leftChild);
}

public BinaryTree getRightSubTree()
{
return(rightChild);
}
……..
}
```

Binary Tree Prorder Traversal

- In preorder, the root is visited *first*
- Here’s a preorder traversal to print out all the elements in the binary tree:

```
public void preorderPrint(BinaryTree bt) {
     if (bt == null) return;
     System.out.println(bt.value);
     preorderPrint(bt.leftChild);
     preorderPrint(bt.rightChild);
}
```

PLR

Binary Tree Inorder Traversal

- In inorder, the root is visited *in the middle*
- Here’s an inorder traversal to print out all the elements in the binary tree:

```
public void inorderPrint(BinaryTree bt) {
     if (bt == null) return;
     inorderPrint(bt.leftChild);
     System.out.println(bt.value);
     inorderPrint(bt.rightChild);
}
```

LPR

Binary Tree Postorder Traversal

- In postorder, the root is visited *last*
- Here’s a postorder traversal to print out all the elements in the binary tree:

```
public void postorderPrint(BinaryTree bt) {
    if (bt == null) return;
    postorderPrint(bt.leftChild);
    postorderPrint(bt.rightChild);
    System.out.println(bt.value);
}
```

LRP