

Data Structures and Algorithms (CS-2001)

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering



Strictly for internal circulation (within KIIT) and reference only. Not for outside circulation without permission

4 Credit

Lecture Note

Chapter Contents



2

Sr #	Major and Detailed Coverage Area	Hrs
5	Trees Introduction, Binary Trees, Binary Tree Traversals, Threaded Binary Trees, Binary Search Trees, AVL Trees, m-way search Trees, B-Trees, B+ Trees, Tree Operation, Forests	8

Introduction



3

There are many basic data structures that can be used to solve application problems. Array is a good data structure that can be accessed randomly and is fairly easy to implement. Linked Lists on the other hand is dynamic and is ideal for application that requires frequent operations such as add, delete, and update. One drawback of linked list is that data access is sequential. Then there are other specialized data structures like, stacks and queues that allows to solve complicated problems using these restricted data structures.

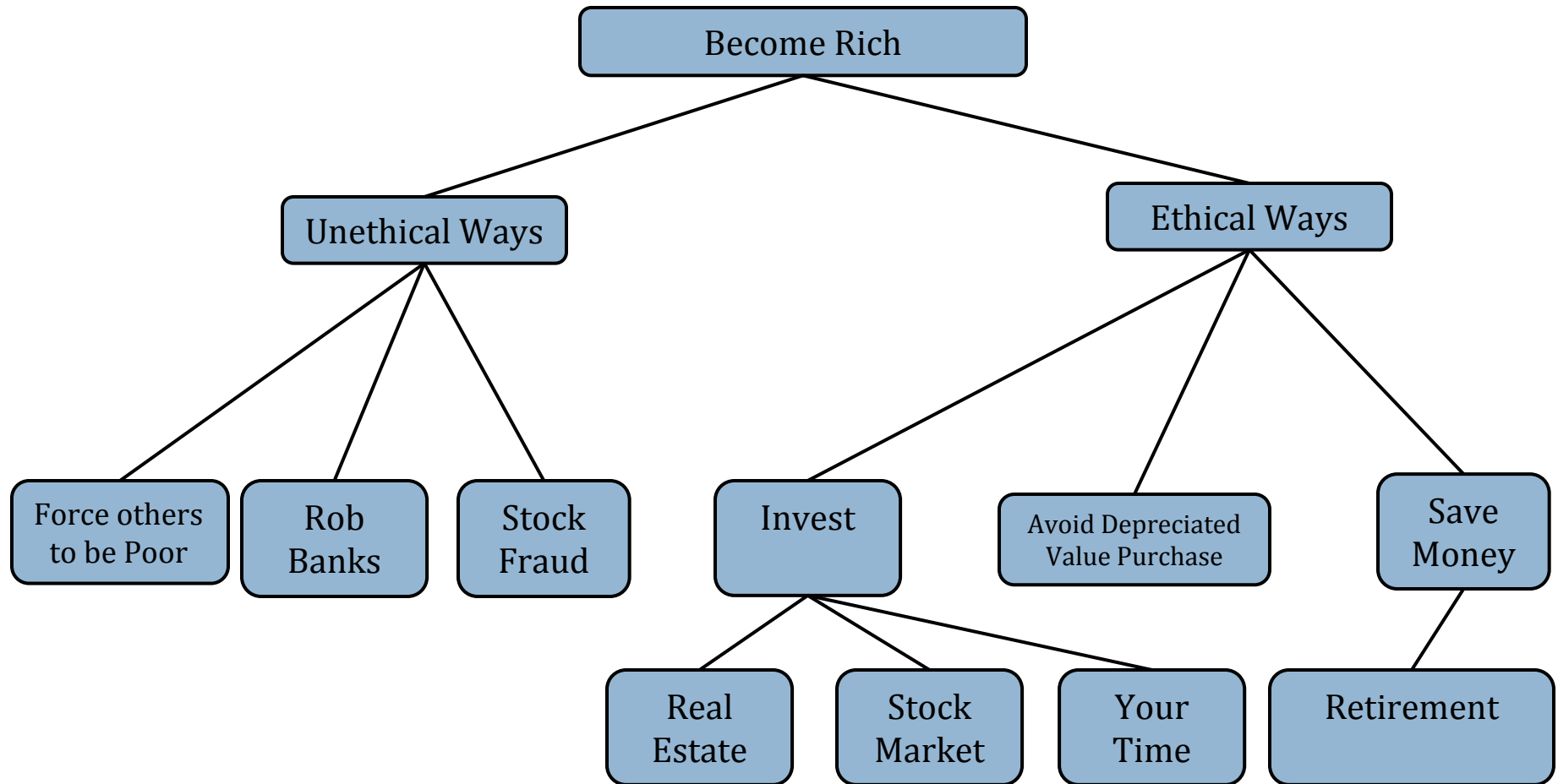
One of the **disadvantages** of using an array or linked list to store data is the time necessary to search for an item. Since both the arrays and Linked Lists are linear structures the time required to search a “linear” list is **proportional** to the size of the data set. For example, if the size of the data set is n , then the number of comparisons needed to find (or not find) an item may be as bad as n . So imagine doing the search on a linked list (or array) with $n = 10^6$ nodes. Even on a machine that can do million comparisons per second, searching for m items will take roughly m seconds. This not acceptable in today’s world where speed at which we complete operations is extremely important. **Time is money**. Therefore it seems that better (more efficient) data structures are needed to store and search data.

In this chapter, we can extend the concept of linked data structure (linked list, stack, queue) to a structure that may have multiple relations among its nodes. Such a structure is called **atree**. A tree is a **nonlinear** data structure, compared to arrays, linked lists, stacks and queues which are linear data structures.

To Become Rich



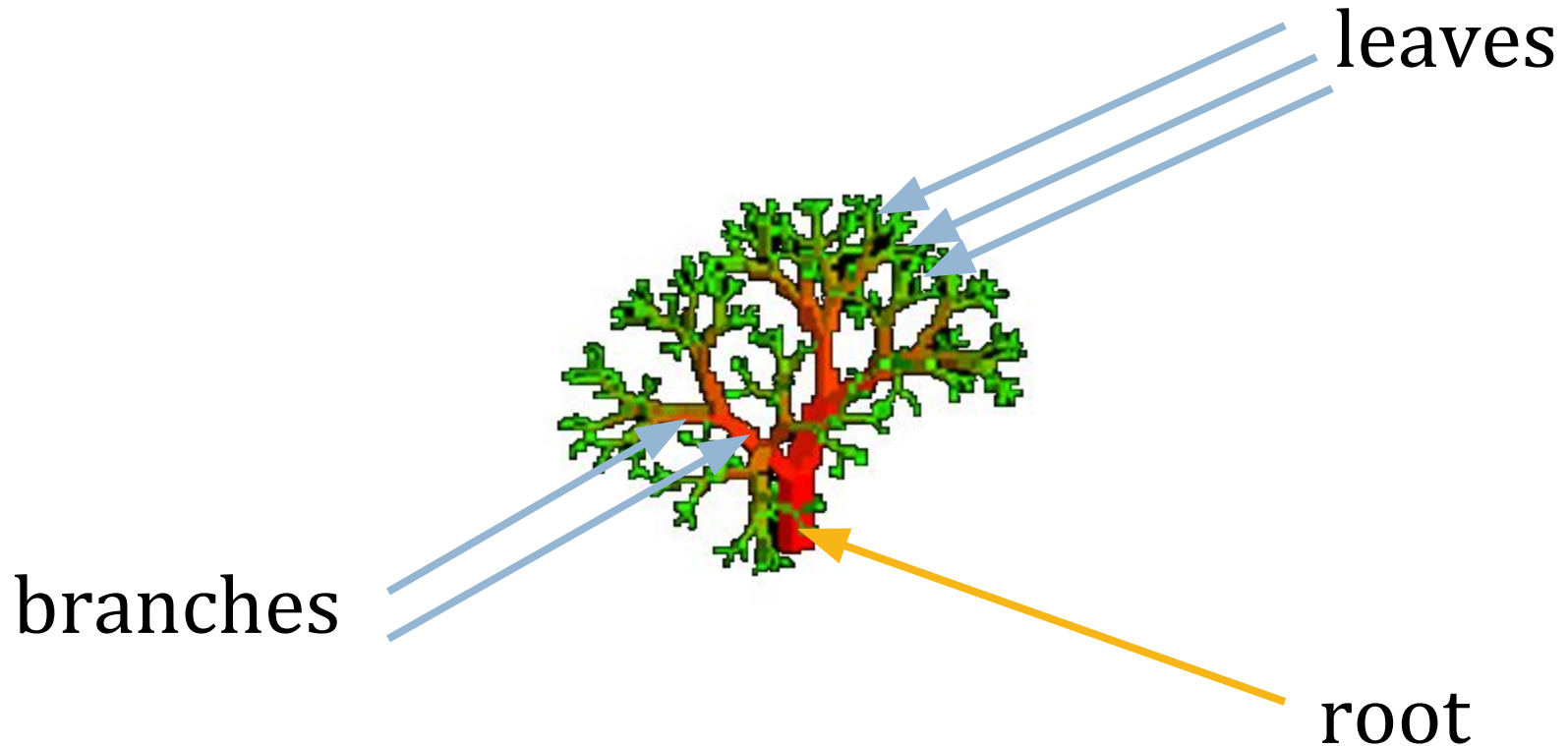
4



Nature View of a Tree



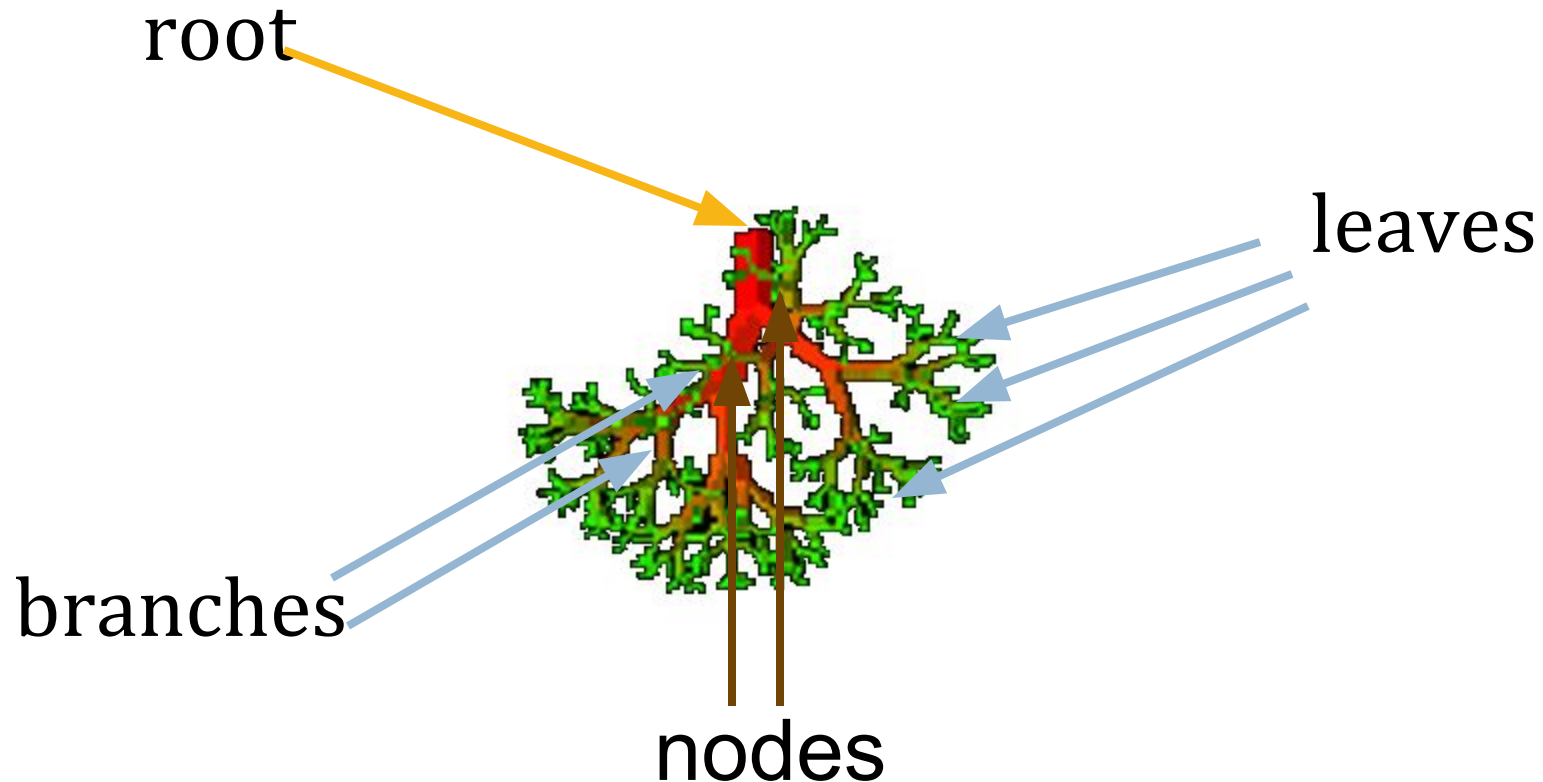
5



Computer View



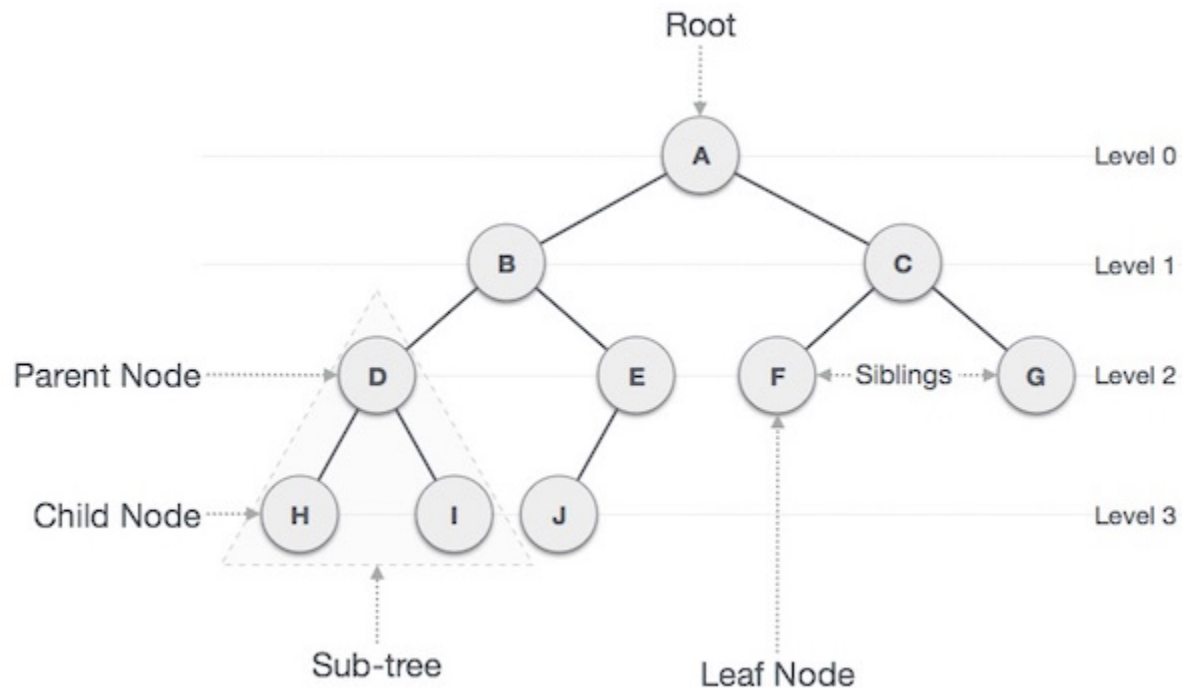
6



What is a Tree?

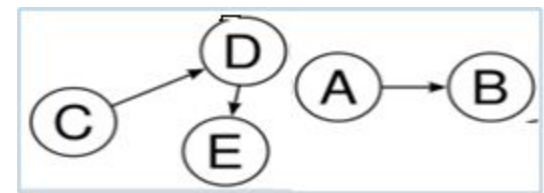
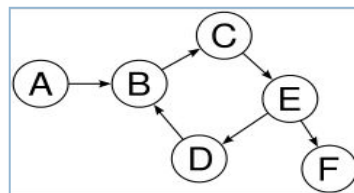
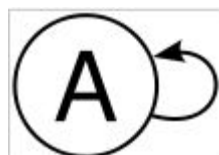
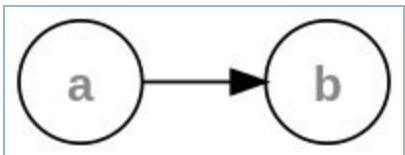


7



- ❑ A tree is a **finite nonempty** set of **nodes** and **edges** without **having any cycle**.
- ❑ It is an abstract model of a hierarchical structure.
- ❑ Nodes represents the parent-child relation.

Which of the following are tree?

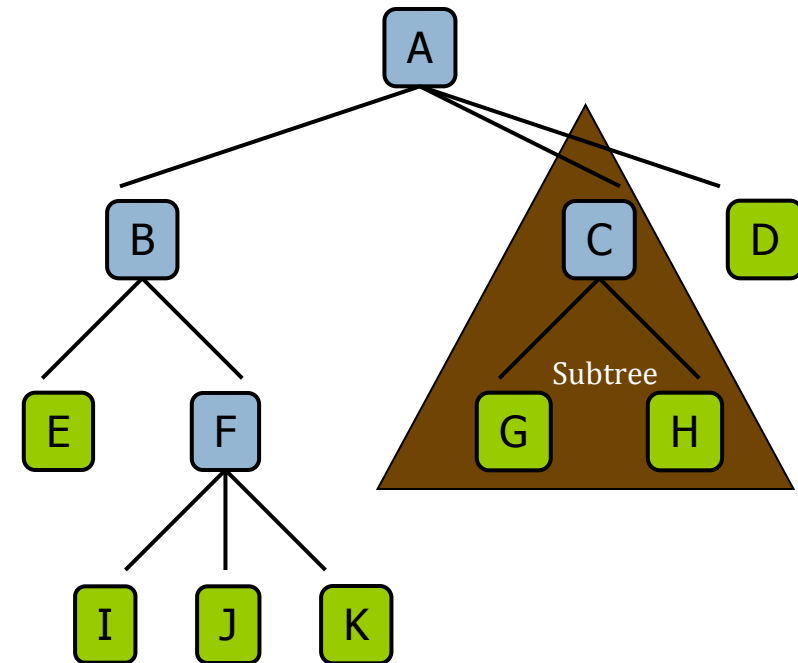


Tree Terminology



8

- ❑ **Path** – Refers to sequence of nodes along the edges of a tree.
- ❑ **Root Node** – Node at the top of the tree is called root. There is **only one root per tree** and **one path from root node to any node**. So node without parent is also called root.
- ❑ **Parent Node** – Any node except root node has one edge upward to a node.
- ❑ **Child Node** – Node below a given node connected by its edge downward.
- ❑ **Leaf Node** - Node which does not have any child node.
- ❑ **Internal Node** – Node with at least one child
- ❑ **Subtree** – Subtree represents descendents of a node.
- ❑ **Visiting** – Refers to checking value of a node when control is on the node.
- ❑ **Traversing** – Passing through nodes in a specific order.
- ❑ **Levels** – Level of a node **represents the generation of a node**. If root node is at level 0, then its next child node is at level 1, its grandchild is at level 2 and so on.
- ❑ **Keys** – Key represents a value of a node based on which a search operation is to be carried out for a node.
- ❑ **Degree** – The degree of a node is the number of its children & the degree of a tree is the maximum degree of any of its node.
- ❑ **Siblings** – nodes share the same parent

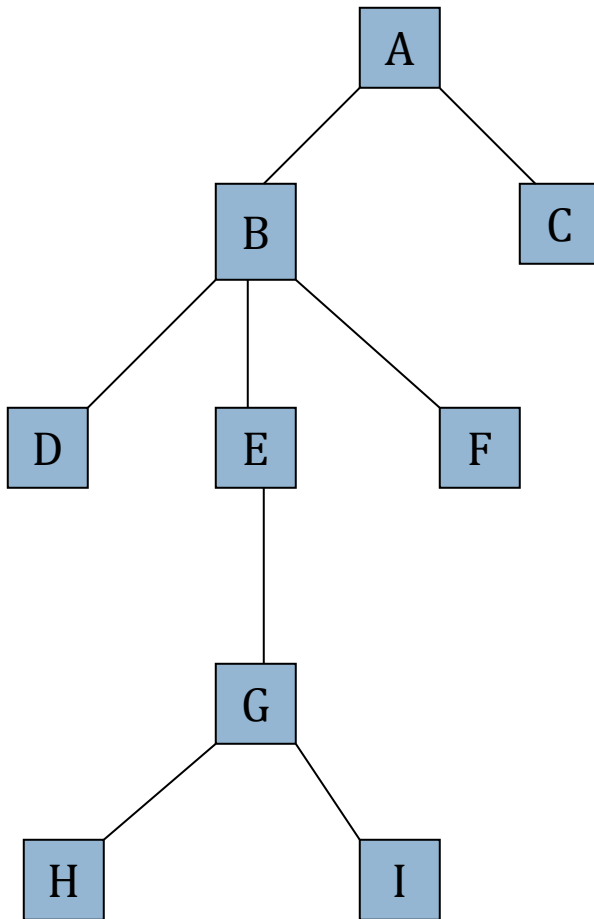


- ❑ **Ancestors of a node:** parent, grandparent, grand-grandparent, etc.
- ❑ **Descendant of a node:** child, grandchild, grand-grandchild, etc.
- ❑ **Depth of a node:** number of ancestors. Depth of tree is 1 more than the maximum depths of the depths of the left and right subtrees of tree
- ❑ **Height of a tree:** maximum depth of any node

Class Work



9



Sr #	Property	Value	
1	Number of nodes of the tree		
2	Height of the tree		
3	Root Node of the tree		
4	Leaves of the tree		
5	Interior nodes of the tree		
6	Ancestors of H node		
7	Descendants of B node		
8	Siblings of E node		
9	Right subtree of A node		
10	Degree of the tree		
11	Parent node of D		
12	Child nodes of G		
13	Level of node G		

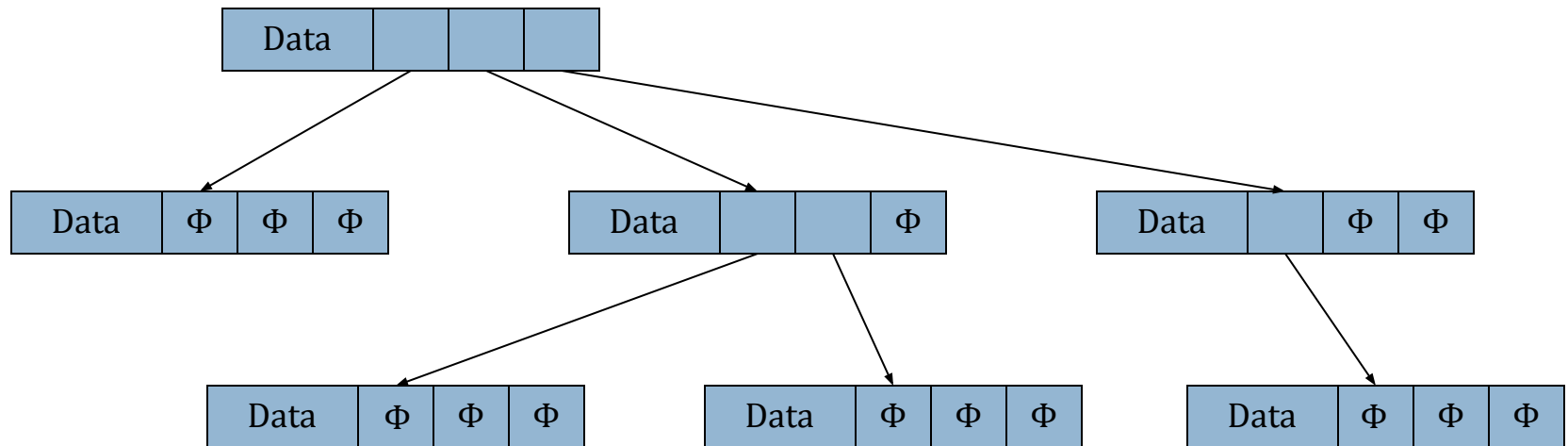
Intuitive Representation



10

Every tree node:

- ❑ INFO – useful information
- ❑ Child – pointers to its children



Trees Application



11

Tree is a fundamentally hierarchical structure. Thus, a tree is appropriate to model any reality that exhibits hierarchy:

- ❑ File system directories.
- ❑ Genealogical trees of all sorts: family relationships among individuals, tribes, languages etc.
- ❑ Classifications systems:
 - ❑ Taxonomic classification of plants and animals.
 - ❑ Dewey decimal (or Library of Congress) classification of books.
- ❑ Breakdown of a manufactured product into subassemblies, each of turn consists of sub-subassemblies etc. down to the smallest components.
- ❑ Structure of a program - main routine is the root, procedures it contains are subtrees, each of which contains nested procedure definitions etc.
- ❑ Information storage and retrieval situations such as symbol tables, even though hierarchy may not be involved.
- ❑ Expression Tress – Compilers and Interpreters as well in symbolic mathematics program to represent arithmetic expressions.
- ❑ Game Tree – Game playing

Binary Trees



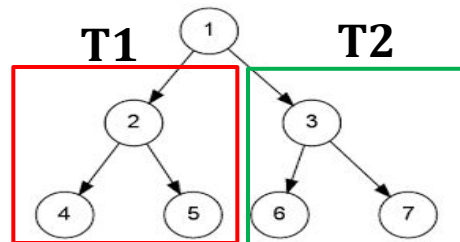
12

A binary tree T is defined as a **finite set of elements** called **node** such that :

- a. T is empty (called the empty tree or null tree) or
- b. T contains a distinguished node R , called the **root node** of T and the remaining nodes of T form an **ordered pair of disjoint** binary trees T_1 and T_2 .

So - a binary tree is a tree data structure in which each node has **at most two children** (i.e. either 0 or 1 or 2), which are referred to as the left child and the right child.

If T contains a root R , then the two trees T_1 and T_2 are called the **left** and **right** subtree of R . If T_1 is **non empty** then its root is called the **left successor of R** . Similarly if T_2 is **not empty** then its root is called the **right successor of R** .



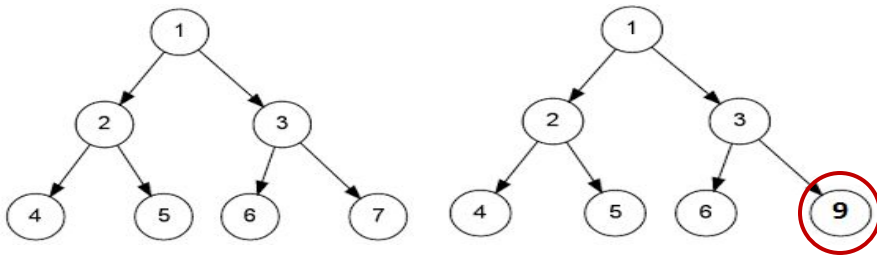
Similar Trees and Tree Copies



13

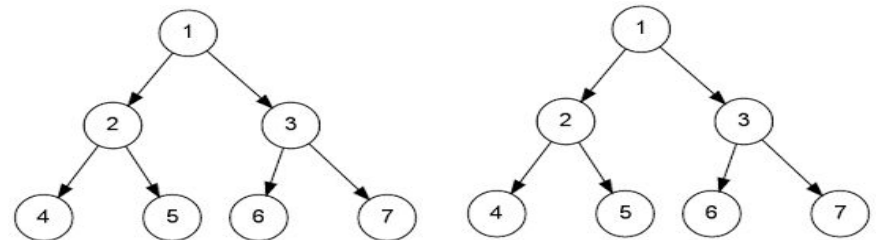
Similar

Binary trees are said to be similar if they have the **same structure** or in other words, if they have the **same shape**.



Tree

The trees are said to be copies if they are **similar** and if they have **same contents at corresponding nodes**

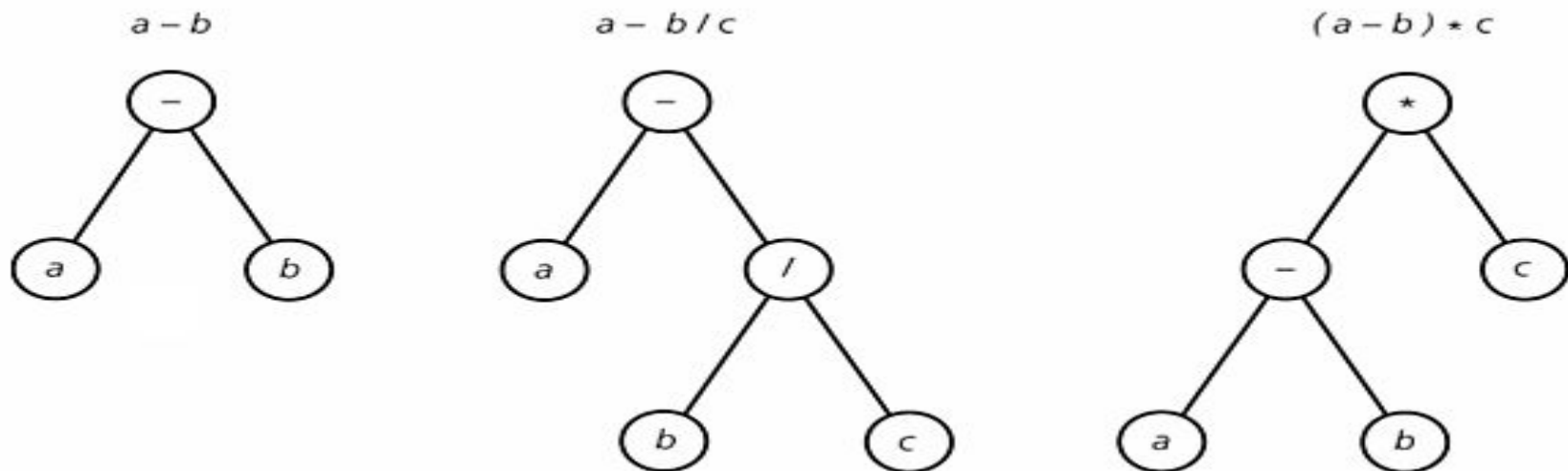


Binary Expression Trees



14

A binary expression tree is a specific kind of a binary tree used to represent expressions. Each node of a binary tree, and hence of a binary expression tree, has zero, one, or two children. This restricted structure simplifies the processing of expression trees. The leaves of a binary expression tree are operands, such as constants or variable names, and the other nodes contain operators i.e. each internal node corresponds to operator and each leaf node corresponds to operand.

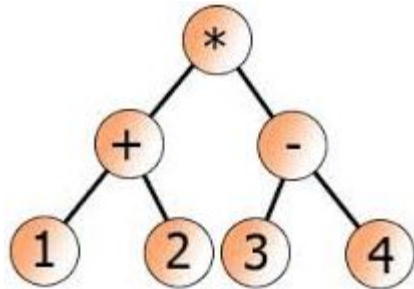


Algebraic Expressions Representation

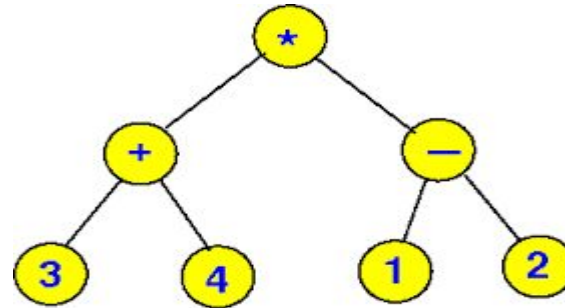


15

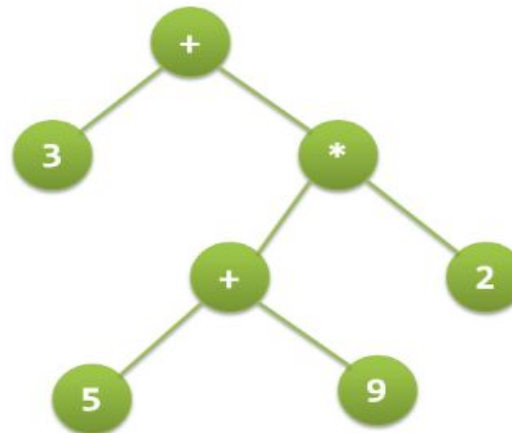
$$((1+2)*(3-4))$$



$$((3+4)*(1-2))$$



$$3 + ((5+9)*2)$$



Expression Tree Class Work



16

Draw the expression tree corresponding to the following:

- ☐ $E = (a/(b + d)) + ((a^2 - d^2)/2)$
- ☐ $F = ((X + Y)^2) - ((X - 4)/12)$
- ☐ $A = 10/4 + 2*(5+1)$
- ☐ $X = ((5*A+B)*(3*C+D))^3$
- ☐ $Y = (35 - 3*(3+2)) / 4$



Full and Complete Binary Trees



17

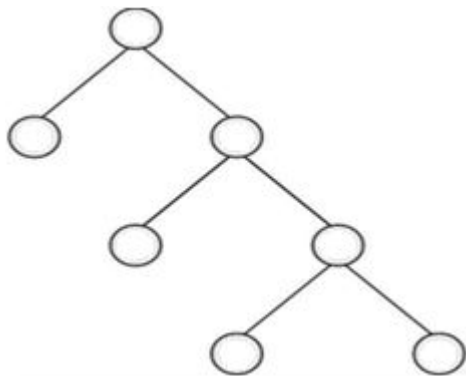
Full Binary

A binary tree T is full if each node is either a leaf or possesses exactly two child nodes.

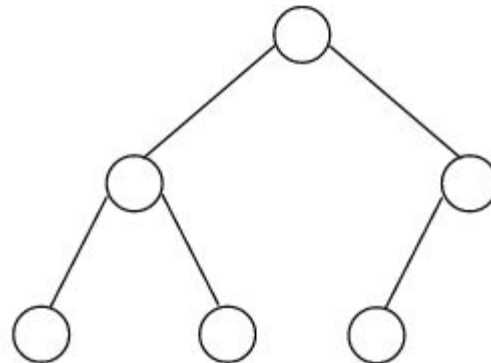
Complete Binary

A binary tree T with n levels is complete if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.

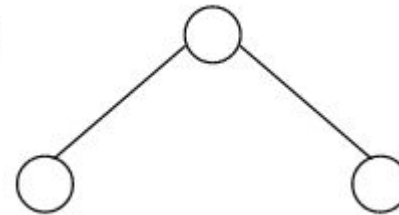
Class Work



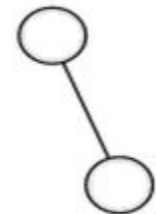
Full but not Complete



Complete but not Full



Full and Complete



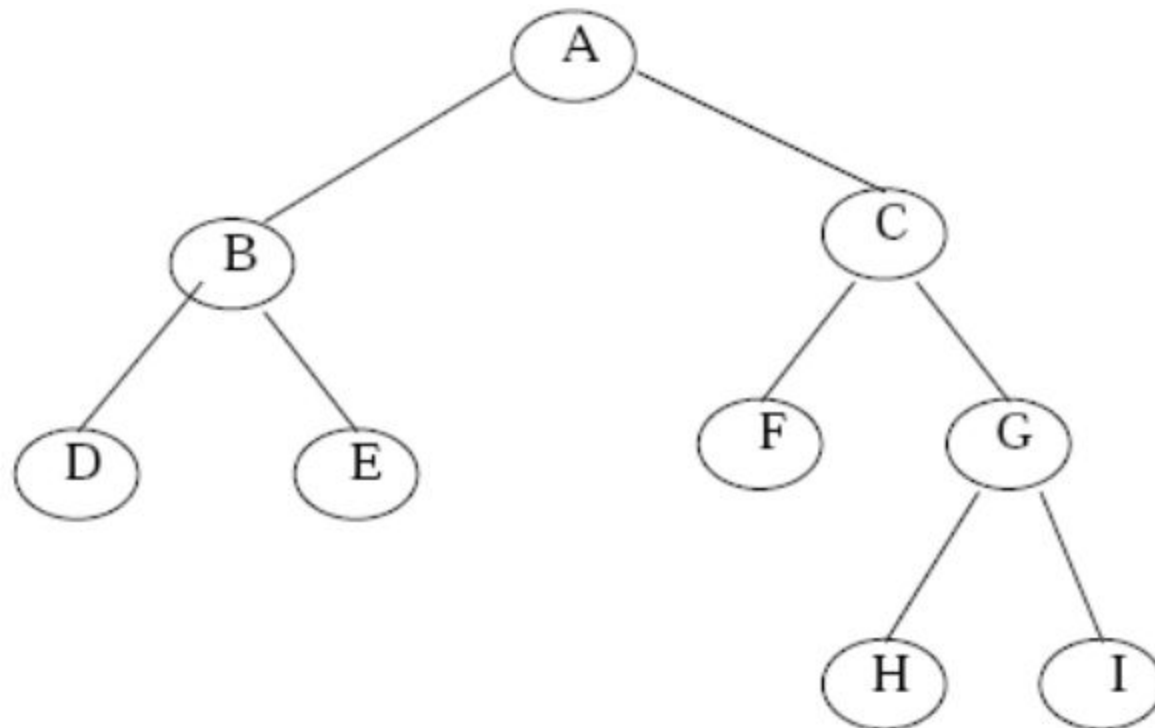
Neither Full nor Complete

Extended Binary Trees : 2-Trees



18

The tree is said to be **strictly** or **extended** or **2-trees** binary tree, if **every non-leaf node in a binary tree has non-empty left and right sub trees**. A strictly binary tree with n leaves always contains $2*n-1$ nodes.

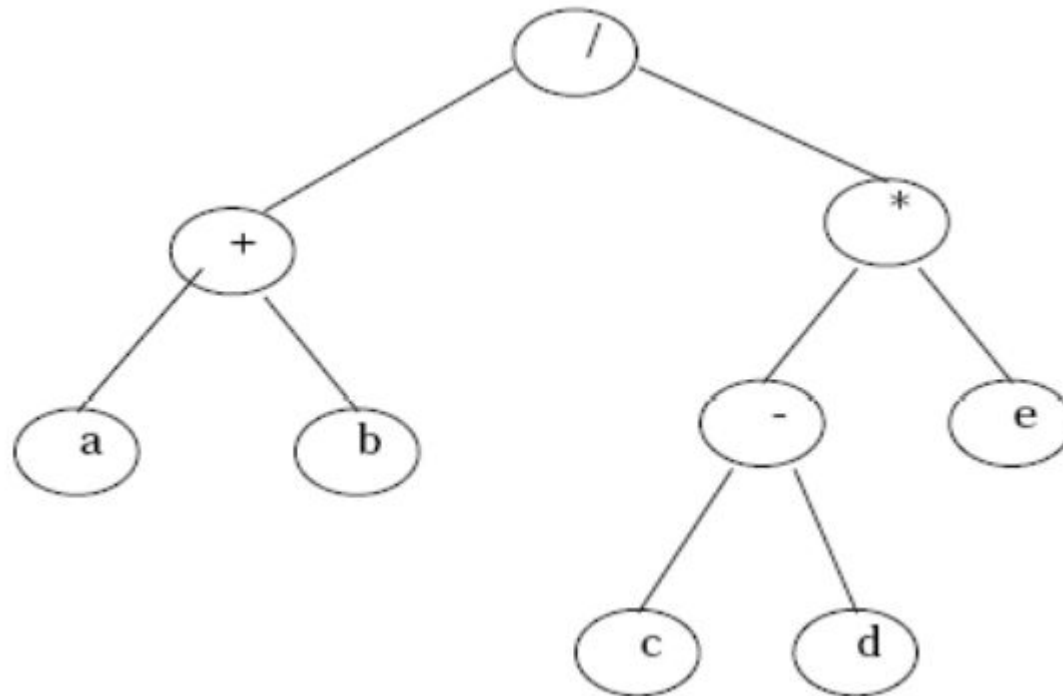


Extended Binary Trees cont...



19

The main application of a 2-tree is to represent and compute any algebraic expression using binary operation. For example, consider an algebraic expression E , where $E = (a + b) / ((c - d) * e)$



Representing Binary Trees in Memory



20

Let T be the binary tree. There are 2 ways of representing binary tree in memory. The first and usual way is called the **link representation of T** and the second way which uses a **single array** called the **sequential representation of T** .

Sequential

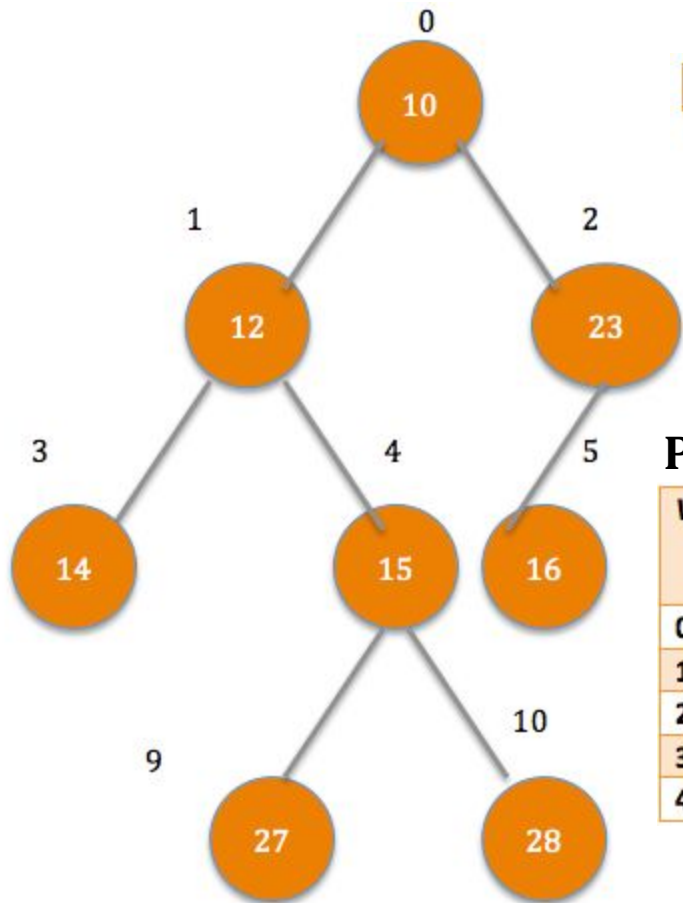
Binary Trees can be represented using 1-D array in memory. The rule to store binary tree in array are :

- ❑ The values of binary tree is stored in an array called as **tree**.
- ❑ The root of the tree is stored at first location i.e. at $\text{tree}[0]$ (or at 0^{th} index).
- ❑ The left child of tree is stored at $2i + 1$ and right child is stored at $2i + 2$ location.
- ❑ The maximum size of the array tree is given as $2^{h+1} - 1$, where h is the height of the tree.
- ❑ An empty tree or sub-tree is specified using NULL. If $\text{tree}[0] = \text{NULL}$, then it means that tree is empty.

Sequential Representation of Binary Tree



21



tree

10	12	23	14	15	16				27	28
0	1	2	3	4	5	6	7	8	9	10

Position of left and right child

Value of i	Position of left child ($2i+1$)	Value of left node	Position of right child ($2i+2$)	Value of right node
0	1	12	2	23
1	3	14	4	15
2	5	16	6	---
3	7	---	8	---
4	9	27	10	28



Linked Representation of Binary Tree

22

Approach 1 - Parallel

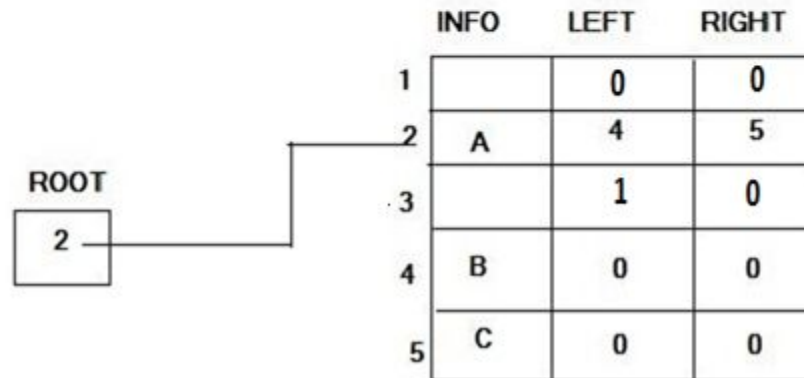
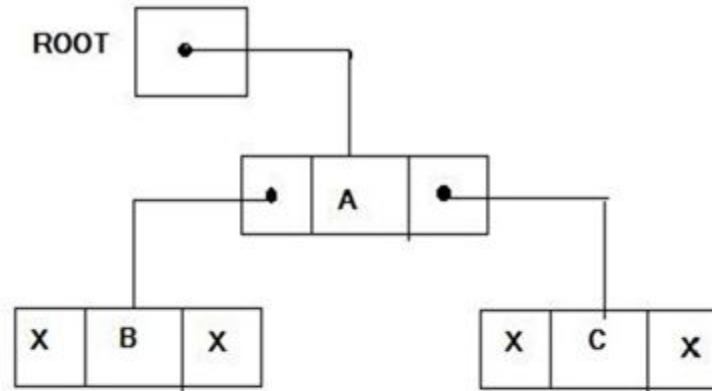
Consider a binary tree T . The linked representation uses three parallel arrays i.e. INFO, LEFT & RIGHT & a pointer variable ROOT. Each node N of T will correspond to a location K such that:

- ❑ INFO[K] contains the data at node N .
- ❑ LEFT[K] contains the location of left child of node N .
- ❑ RIGHT[K] contains the location of right child of node N .
- ❑ ROOT will contain location of root R of T .

If any subtree is empty, then the corresponding pointer will contain NULL values;
if the tree T itself is empty, then ROOT will contain NULL

Linked Representation – Parallel Arrays

23



Linked Representation of Binary Tree



24

Approach 2 - Nodes

Binary trees can be represented by links where each node contains the address of the left child and the right child. If any node has its left or right child empty then it will have in its respective link field, a null value. A leaf node has null value in both of its links.

The structure defining a node of binary tree in C is as follows.

```
struct node
{
    struct node *leftChild ; /* points to the left child */
    int data; /* data field */
    struct node *rightChild; /* points to the right child */
}
```


Linked Representation – Nodes

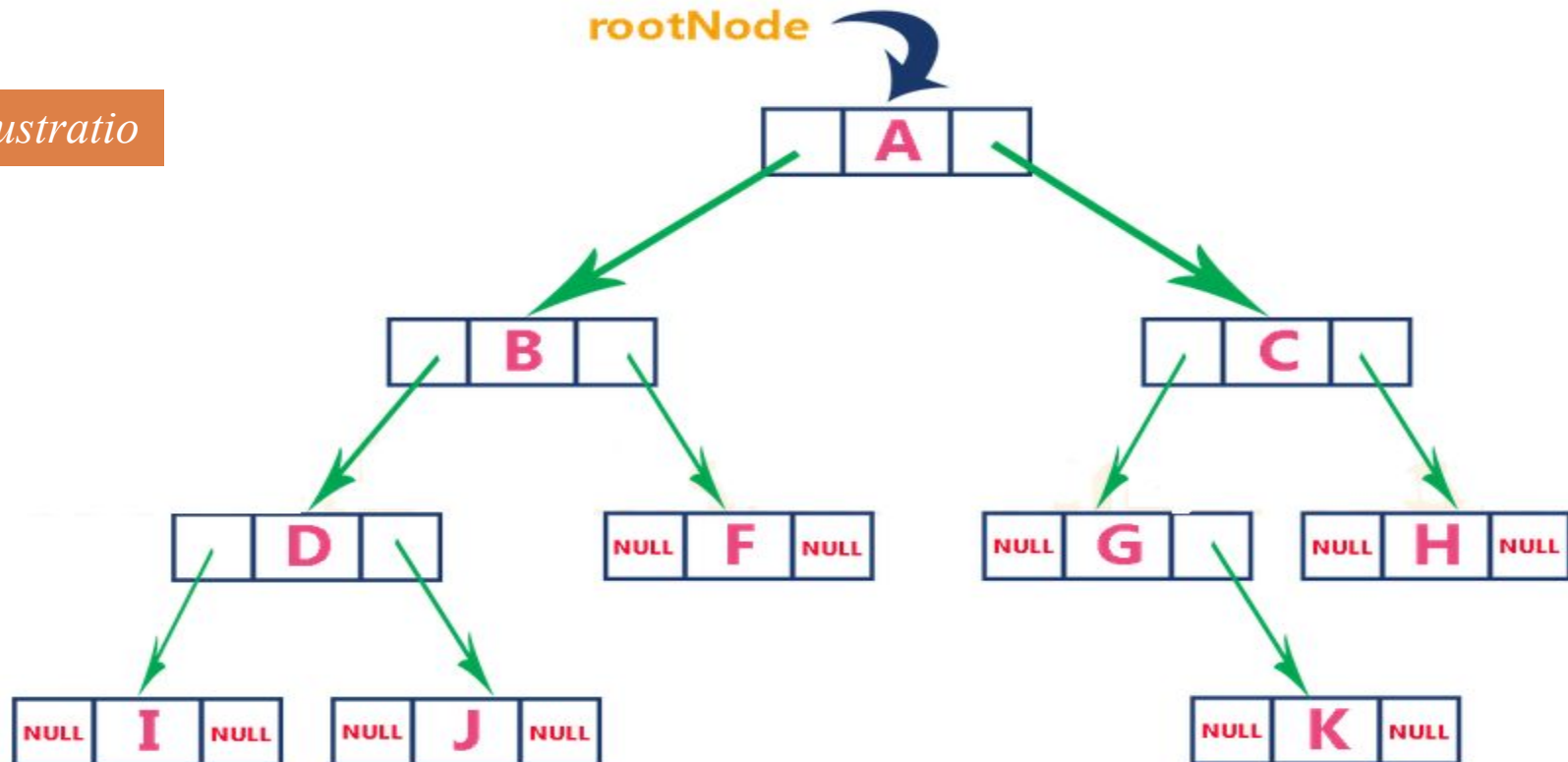


25

Node



Illustratio



Binary Tree Node Structure & Basic Operation



26

A tree node should look like the below structure. It has data part and references to its left and right child nodes.

```
struct node
{
    struct node *leftChild;
    int data;
    struct node *rightChild;
} *root;
```

Basic operations that can be performed on binary search tree data structure, are –

- ❑ **Insert** – insert a node in a tree / create a tree.
- ❑ **Search** – search an element in a tree.
- ❑ **Delete** – Delete the tree or delete a node in the tree.
- ❑ **Preorder Traversal** – traverse a tree in a preorder manner.
- ❑ **Inorder Traversal** – traverse a tree in an inorder manner.
- ❑ **Postorder Traversal** – traverse a tree in a postorder manner.

Binary Tree Traversal



27

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges, traverse always start from the root node and one cannot random access a node in tree. There are three ways to traverse a tree –

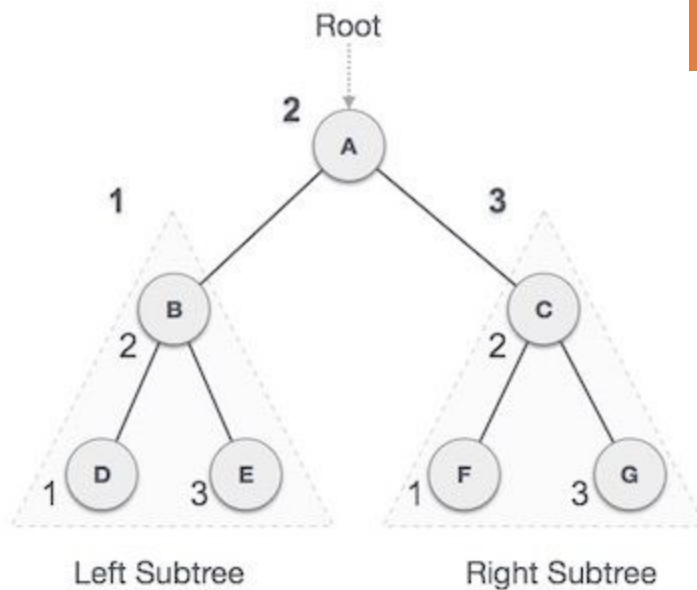
- ☐ In-order Traversal
- ☐ Pre-order Traversal
- ☐ Post-order Traversal

In-order Traversal



28

In this traversal method, the left-subtree is visited first, then root and then the right sub-tree.



Recursive Algorithm

Step 1 – Start

Step 2 – Repeat steps 3, 4 and 5 until all nodes are traversed

Step 3 – Recursively traverse left subtree

Step 4 – Visit root node

Step 5 – Recursively traverse right subtree

Step 6 – Stop

Output

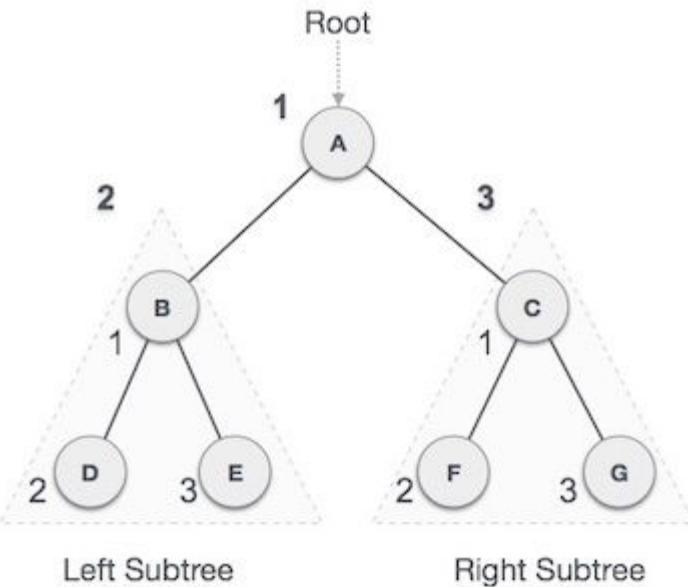
Traversal start from A, and following in-order traversal, we move to its left subtree B. B is also traversed in-ordered and the process goes on until all the nodes are visited. The output of in-order traversal of this tree is **D → B → E → A → F → C → G**

Pre-order Traversal



29

In this traversal method, the root node is visited first, then left subtree and finally right sub-tree.



Recursive Algorithm

Step 1 – Start

Step 2 – Repeat steps 3, 4 and 5 until all nodes are traversed

Step 3 – Visit root node

Step 4 – Recursively traverse left subtree

Step 5 – Recursively traverse right subtree

Step 6 – Stop

Output

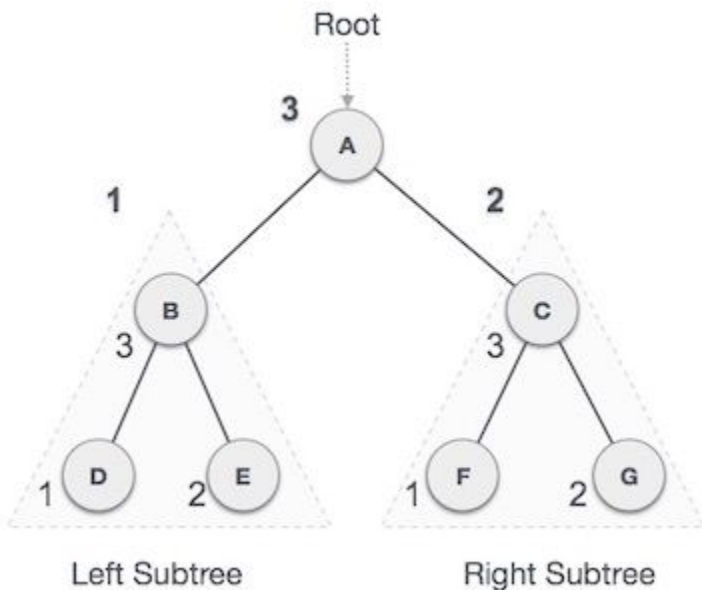
Traversal start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-ordered. And the process goes on until all the nodes are visited. The output of pre-order traversal of this tree is $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C \rightarrow F \rightarrow G$

Post-order Traversal



30

In this traversal method, the root node is visited last, hence the name. First the left subtree is traversed, then right subtree and finally root.



Recursive Algorithm

Step 1 – Start

Step 2 – Repeat steps 3, 4 and 5 until all nodes are traversed

Step 3 – Recursively traverse left subtree

Step 4 – Recursively traverse right subtree

Step 5 – Visit root node

Step 6 – Stop

Output

Traversal start from A, and following pre-order traversal, we first visit left subtree B. B is also traversed post-ordered. And the process goes on until all the nodes are visited. The output of post-order traversal of this tree is **D → E → B → F → G → C → A**

Binary Tree C Implementation



31



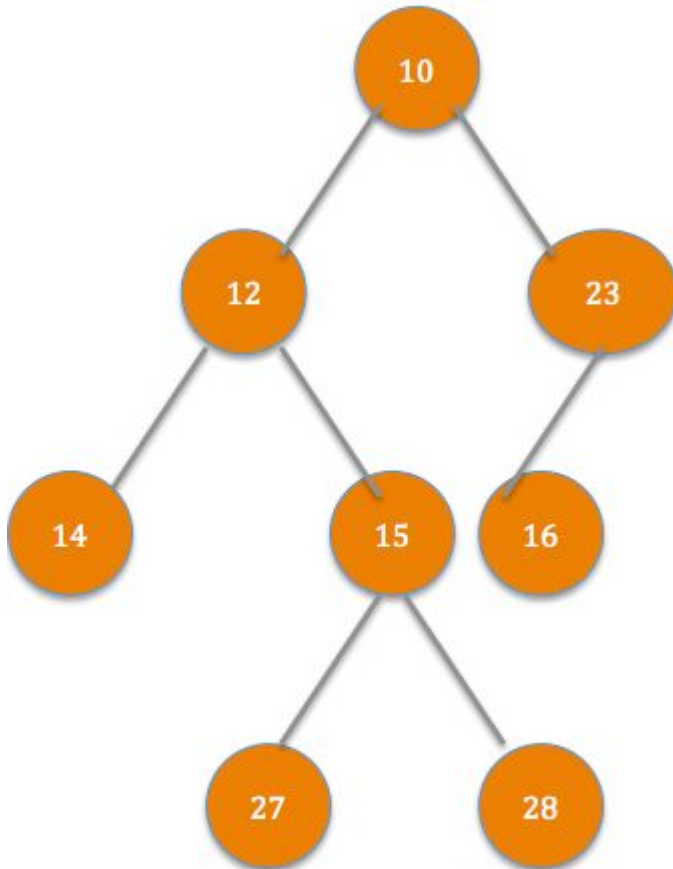
Tree

Level-order Traversal



32

In this traversal method, visit the nodes level by level from left to right



Recursive Algorithm

Step 1 – Start

Step 2 – Visit the root node (Level 0)

Step 3 – Visit the root's left child followed by right child (Level 1)

Step 4 – Recursively visit the next levels from left most child to right most child (Level 2 and onwards) until all levels are covered

Step 5 – Stop

Output

10 → 12 → 23 → 14 → 15 → 16 → 27 →

28

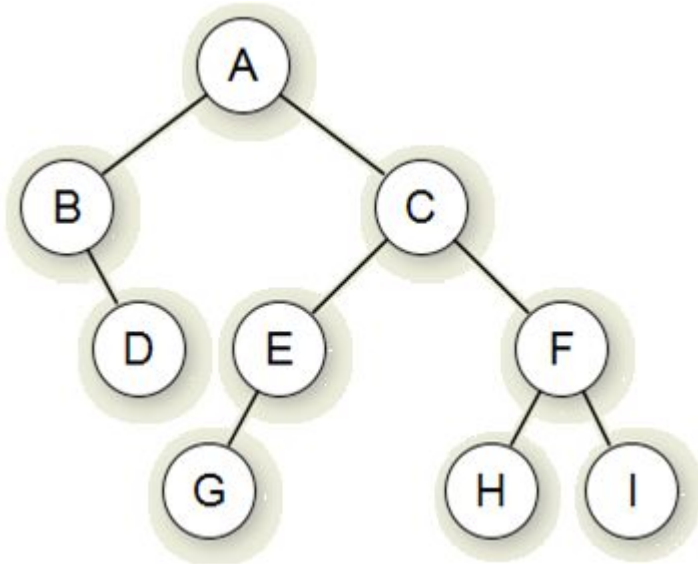
School of Computer Engineering

Class Work



33

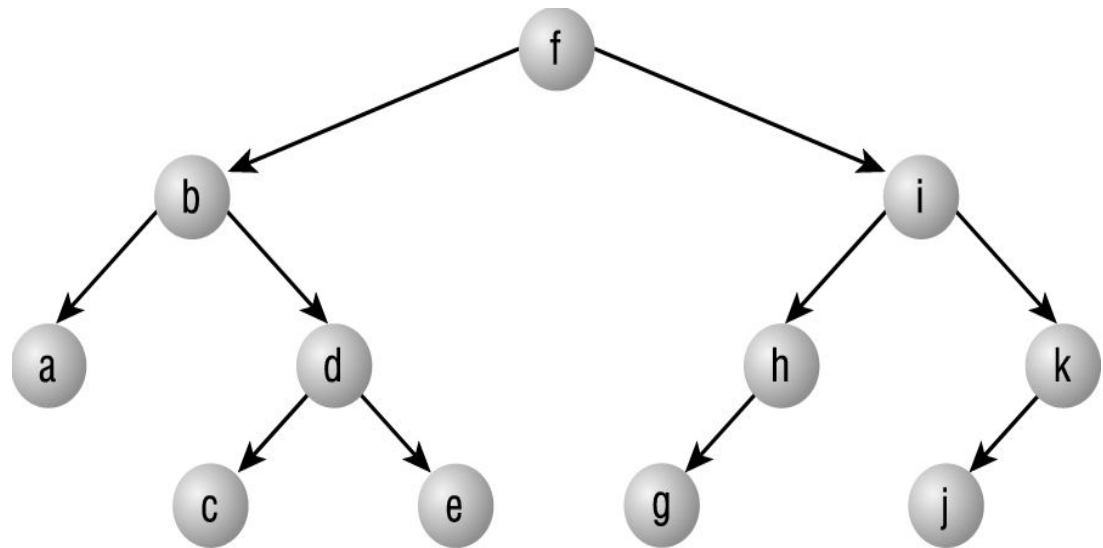
What are the In Order, Pre Order and Post Order traversal sequence of following trees?



In Order : $B \rightarrow D \rightarrow A \rightarrow G \rightarrow E \rightarrow C \rightarrow H \rightarrow F \rightarrow I$

Pre Order : $A \rightarrow B \rightarrow D \rightarrow C \rightarrow E \rightarrow G \rightarrow F \rightarrow H \rightarrow I$

Post Order : $D \rightarrow B \rightarrow G \rightarrow E \rightarrow H \rightarrow I \rightarrow F \rightarrow C$



In Order : $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h \rightarrow i \rightarrow j \rightarrow k$

Pre Order : $f \rightarrow b \rightarrow a \rightarrow d \rightarrow c \rightarrow e \rightarrow i \rightarrow h \rightarrow g \rightarrow k \rightarrow j$

Post Order : $a \rightarrow c \rightarrow e \rightarrow d \rightarrow b \rightarrow g \rightarrow h \rightarrow j \rightarrow k \rightarrow i \rightarrow f$

Note

In Order : LPR

Pre Order : PLR

Post Order : LRP

P: Parent, L: Left and R: Right

In-Order Traversal Non-Recursive Algorithm



34

1. Create a stack.
2. Push the root into the stack and set the root = root.left and continue till it hits NULL.
3. If root is null and stack is empty then
 return // we are done.
 Else
 pop the top node from the stack and set it as, root = popped_node.
 print the root and go right, root = root.right.
 Go to step 2.
4. End If

Animation

<http://algorithms.tutorialhorizon.com/files/2015/12/Inorder1.gif>

Class Work

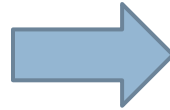
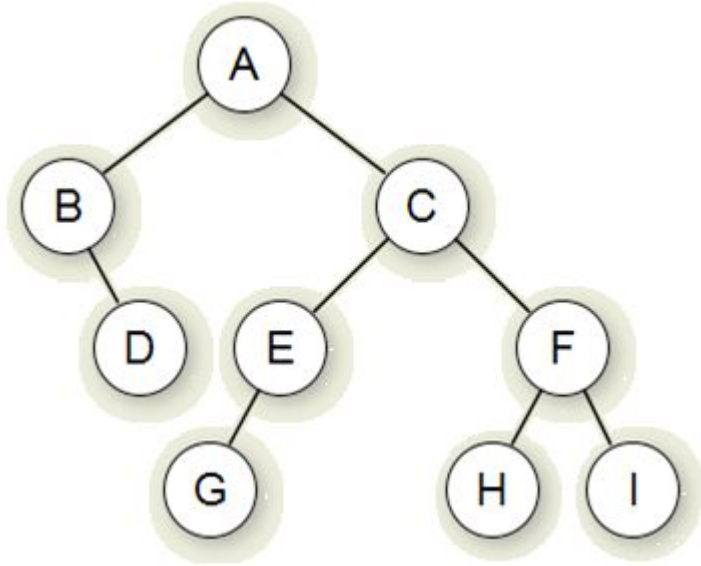
- ☐ Design non-recursive algorithm for Pre-order Traversal
- ☐ Design non-recursive algorithm for Post-order Traversal

Class Work – Construction of Binary Tree



35

In Order : $B \rightarrow D \rightarrow A \rightarrow G \rightarrow E \rightarrow C \rightarrow H \rightarrow F \rightarrow I$



Is the tree different? If so – don't worry

With one sequence, it is not possible to estimate the left/right child of the tree. So a given In-Order sequence, will not generate exact binary tree. Similarly given Pre-Order or Post-order sequence will not generate exact binary tree. So to generate the exact binary tree, more than 2 traversal sequence is mandate.



If you are given two traversal sequences, can you construct the binary tree?

It depends on what traversals are given. If one of the traversal methods is In-order then the tree can be constructed, otherwise not. Therefore, following combination can uniquely identify a tree.

- ☐ In-order and Pre-order
- ☐ In-order and Post-order

Construction of Binary Tree from In Order and Pre Order Traversal



36

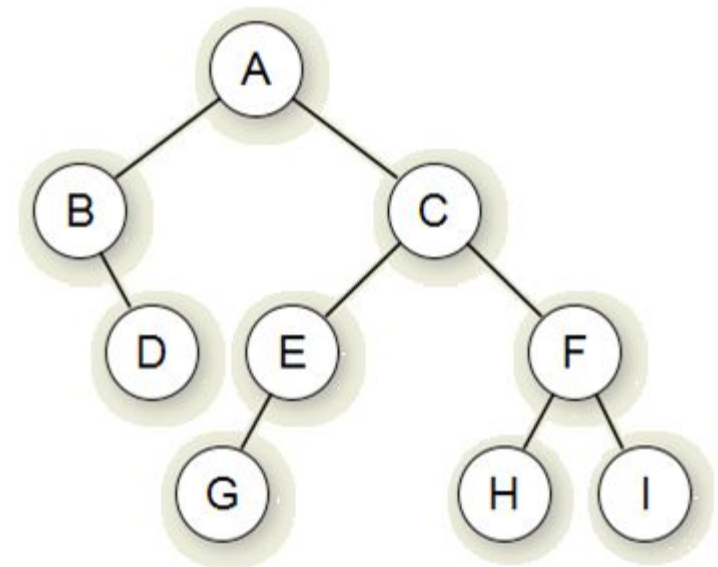
In Order : LPR and Pre Order : PLR

In Order : $B \rightarrow D \rightarrow A \rightarrow G \rightarrow E \rightarrow C \rightarrow H \rightarrow F \rightarrow I$

Pre Order : $A \rightarrow B \rightarrow D \rightarrow C \rightarrow E \rightarrow G \rightarrow F \rightarrow H \rightarrow I$

Steps to be followed

- ❑ First element in the Pre Order is the root of the tree, hence it is **A**
- ❑ Now search element A in “**In Order**”, say you find it at position i, once you find it, make note of elements which are left to i (this will construct the left subtree) and elements which are right to i (this will construct the right subtree).
- ❑ Recursively construct left subtree and link it root left and recursively construct right subtree and link it root right.



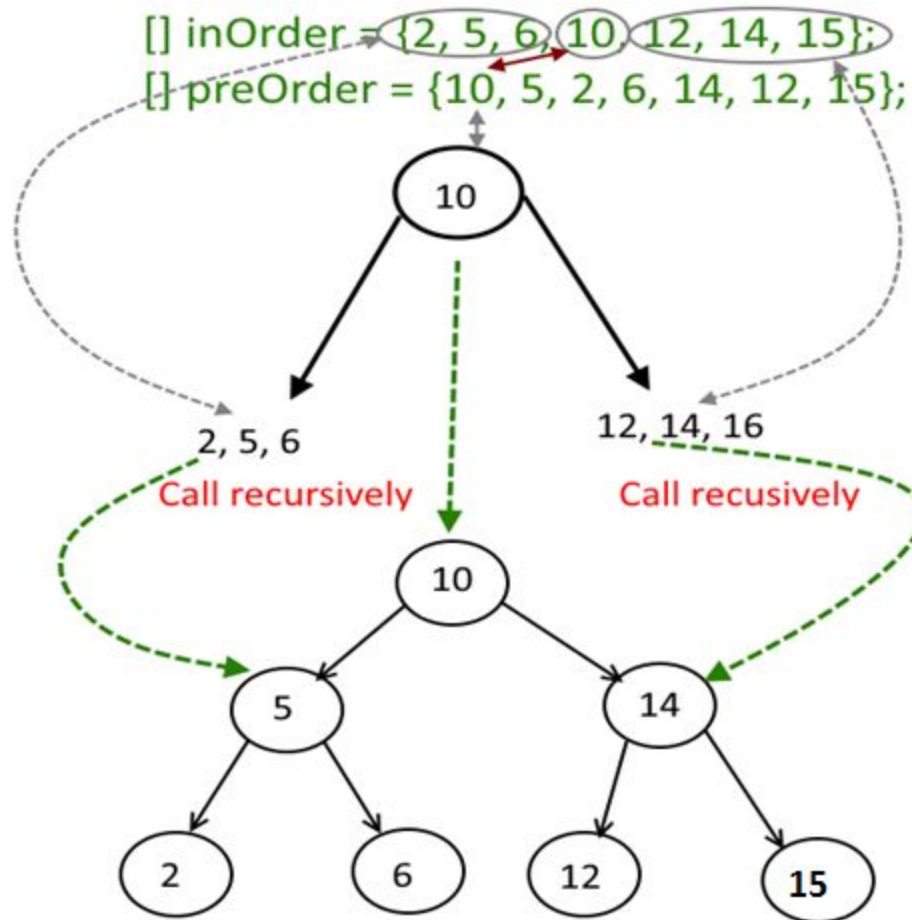
P: Parent, L: Left and R: Right

Construction of Binary Tree from In Order and Pre Order Traversal



37

In Order : {2,5,6,10,12,14,15} and **Pre Order :** {10,5,2,6,14,12,15}



Construction of Binary Tree from In Order and Post Order Traversal



38

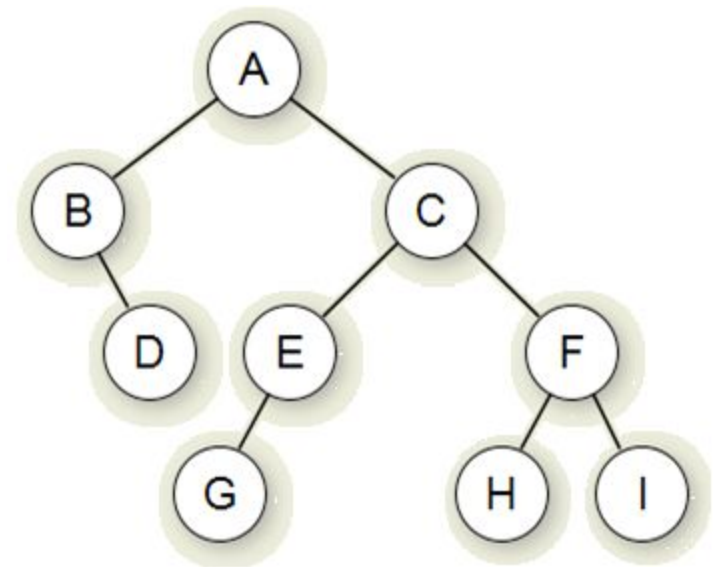
In Order : LPR and Post Order : LRP

In Order : $B \rightarrow D \rightarrow A \rightarrow G \rightarrow E \rightarrow C \rightarrow H \rightarrow F \rightarrow I$

Post Order : $D \rightarrow B \rightarrow G \rightarrow E \rightarrow H \rightarrow I \rightarrow F \rightarrow C \rightarrow A$

Steps to be followed

- ❑ Last element in the Post Order is the root of the tree, hence it is **A**
- ❑ Now search element **A** in “**In Order**”, say you find it at position i , once you find it, make note of elements which are left to i (this will construct the left subtree) and elements which are right to i (this will construct the right subtree).
- ❑ Recursively construct left subtree and link it root left and recursively construct right subtree and link it root right.



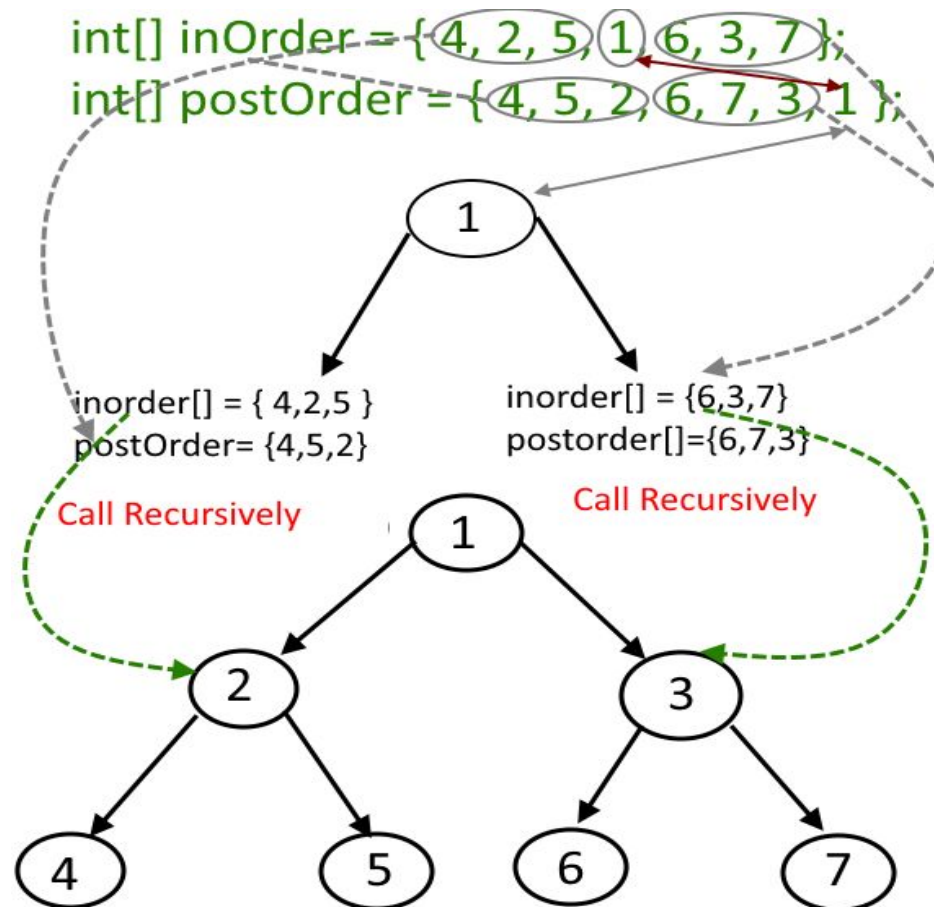
P: Parent, L: Left and R: Right

Construction of Binary Tree from In Order and Post Order Traversal



39

In Order : {4, 2, 5, 1, 6, 3, 7} and **Post Order :** {4, 5, 2, 6, 7, 3, 1}



Construction of Binary Tree from Pre Order and Post Order Traversal



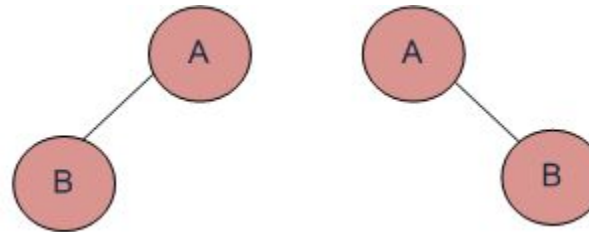
40

Pre Order : PLR and Post Order : LRP

Pre Order : $A \rightarrow B$

Post Order : $B \rightarrow A$

Conclusion



- ❑ One cannot create an exact binary tree from preorder and postorder only as you would never be able to estimate the left/right child of the tree.
- ❑ You need the inorder traversal along with the any of the above

P: Parent, L: Left and R: Right

Threaded Binary Tree



41

A threaded binary tree is a binary tree in which the **nodes that don't have a right child, have a thread to their inorder successor**. The idea of threaded binary trees is to make inorder traversal faster and do it **without stack and without recursion**.

Types of threaded binary trees:

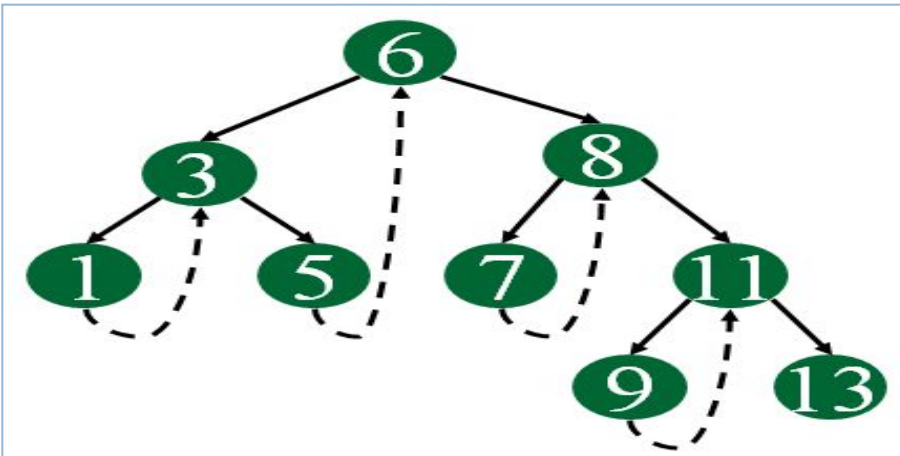
- ❑ **Single Threaded:** each node is threaded towards either the in-order predecessor or successor (left or right) means all right null pointers will point to inorder successor OR all left null pointers will point to inorder predecessor.
- ❑ **Double threaded:** each node is threaded towards both the in-order predecessor and successor (left and right) means all right null pointers will point to inorder successor AND all left null pointers will point to inorder predecessor.

Single & Double Threaded Binary Tree

42

In Order :- 1, 3, 5, 6, 7, 8, 9, 11, 13

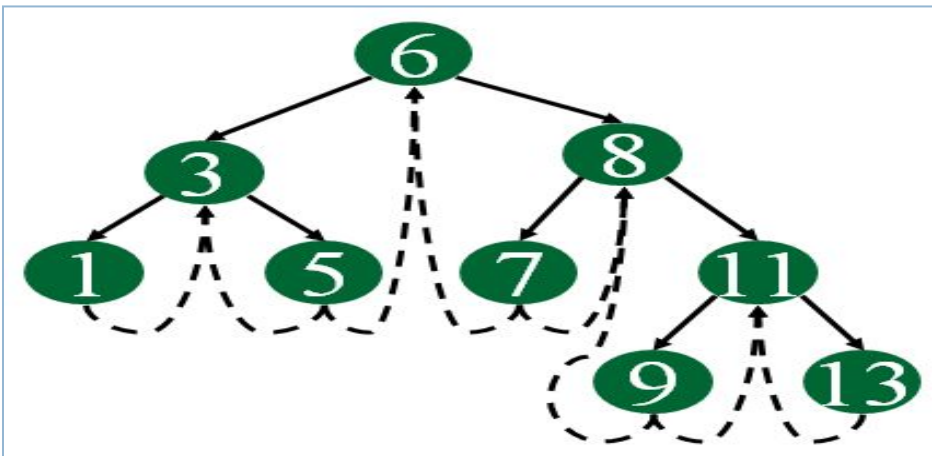
Single Threaded Binary Tree



Node Structure :-

```
struct node
{
    int data;
    struct node *leftChild;
    struct node *rightChild;
    struct node *thread;
};
```

Double Threaded Binary Tree



Node Structure :-

```
struct node
{
    int data;
    struct node *leftChild;
    struct node *rightChild;
    struct node *rthread, *lthread;
};
```

Binary Search Tree



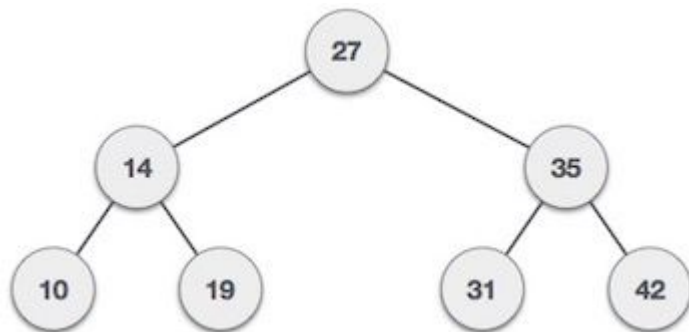
43

A binary search tree (BST) is a tree in which all nodes follows the below mentioned properties –

- ❑ The left sub-tree of a node has data less than to its parent node's data.
- ❑ The right sub-tree of a node has data greater than to its parent node's data.

Thus, a BST divides all its sub-trees into two segments; left sub-tree and right sub-tree and can be defined as - **left_subtree (data) < node (data) < right_subtree (data)**

Example



Quick Questions?

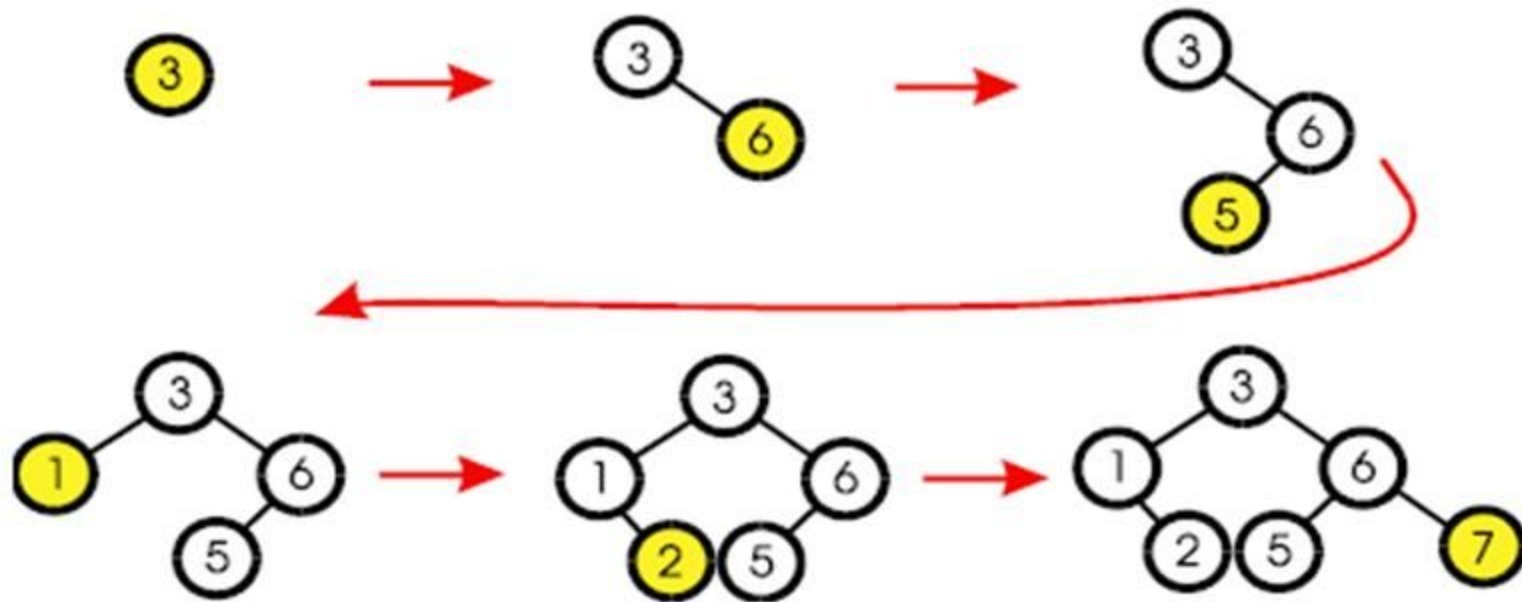
Question - Where is the smallest & largest element in binary search tree resides?

Answer - Leftmost & Rightmost Positions

BST- Insertion



44



BST- Insertion Algorithm



45

```
if tree is empty
    create a root node with the new key
else
    compare key with the top node
    if key = node key
        replace the node with the new value
    else if key > node key
        compare key with the right subtree:
        if subtree is empty create a leaf node
        else add key in right subtree
    else key < node key
        compare key with the left subtree:
        if the subtree is empty create a leaf node
        else add key to the left subtree
```

BST- Search

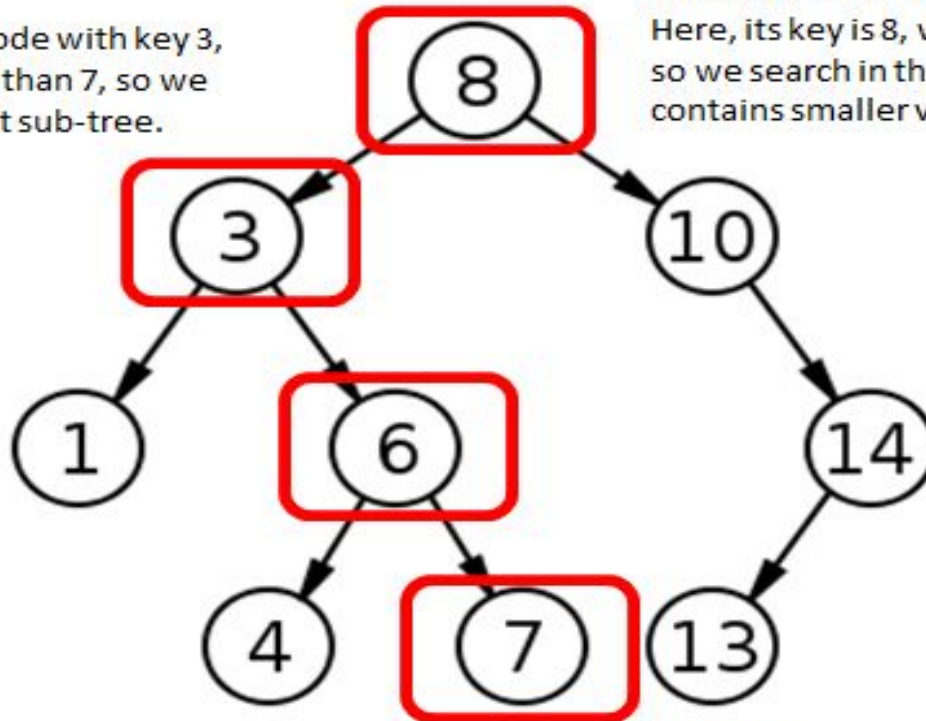


46

Node with key value 7 to be searched

Step 2:

We arrive at a node with key 3, which is smaller than 7, so we search in its right sub-tree.



Step 1:

We start at the “root” node. Here, its key is 8, which is larger than 7, so we search in the left sub-tree, which contains smaller values than 8.

Step 3:

We arrive at a node with key 6, which is again smaller than 7, so we again search in its right sub-tree.

Step 4:

By comparing each node’s key to the key we are looking for, we eventually arrive at the right node with key 7.

BST– Search Algorithm



47

if the tree is empty
 return NULL

else if the key value in the node(root) equals the target
 return the node value

else if the key value in the node is greater than the target
 return the result of searching the left subtree

else if the key value in the node is smaller than the target
 return the result of searching the right subtree

BST– Search - C Function



48

```
/* return a pointer to the node that contains key. If there is no such node, return NULL */
node* search(node * root, int key)
{
    if (!root)
    {
        return NULL;
    }

    if (key == root->key)
    {
        return root;
    }

    if (key < root->key)
    {
        return search(root->left, key);
    }

    return search(root->right, key);
}
```


BST – Search Minimum and Maximum Key C Function



49

Minimum

```
node* searchMinimum(node * root)
{
    if (!root)
    {
        return NULL;
    }

    while (root -> left != NULL)
    {
        root = root -> left;
    }

    return root;
}
```

Maximum

```
node* searchMaximum(node * root)
{
    if (!root)
    {
        return NULL;
    }

    while (root -> right != NULL)
    {
        root = root -> right;
    }

    return root;
}
```

BST - Deletion



50

- ❑ Removing a node from a BST is a bit more complex, since any “holes” should not be created in the tree. The intention is to remove the specified item from the BST and adjust the tree.
- ❑ The binary search algorithm is used to locate the target item: starting at the root, it probes down the tree till it finds the target or reaches a leaf node.

Experimenting the

- ❑ **if** the tree is empty return false
- ❑ **else** attempt to locate the node containing the target using the binary search algorithm:
 - ❑ if the target is not found return false
 - ❑ else the target is found, then remove its node & return true. Now while removing the node four cases may happen
 - ❑ **Case 1** - if the node has 2 empty subtrees
 - ❑ **Case 2** - if the node has a left and a right subtree
 - ❑ **Case 3** - if the node has no left child
 - ❑ **Case 4** - if the node has no right child

BST – Deletion – Case 1

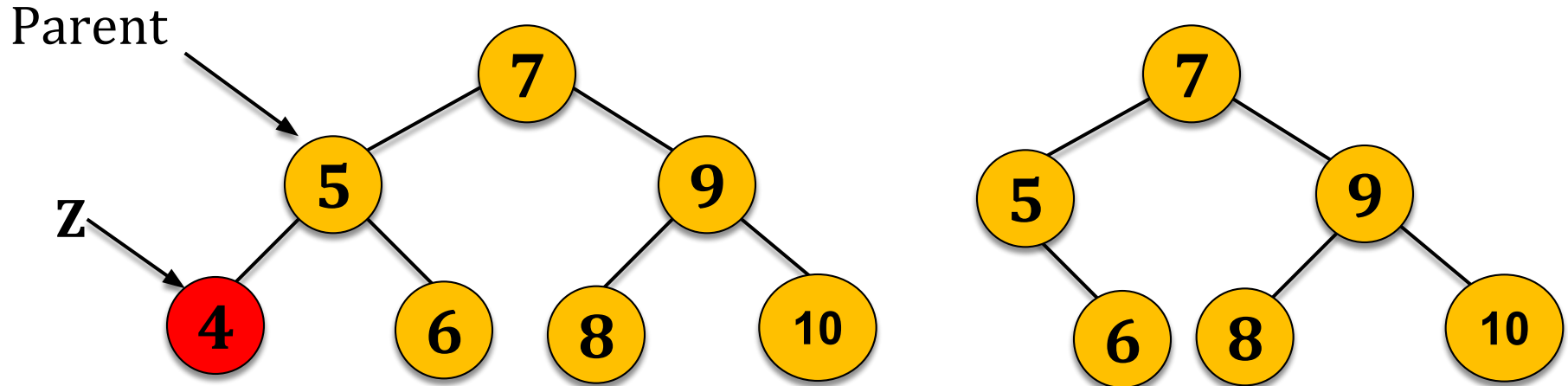


51

Case 1: removing a node with 2 empty subtrees

Action: replace the link in the parent with null and delete the node

Removing 4



Z points to the node which is to be deleted from the tree

BST – Deletion – Case 2

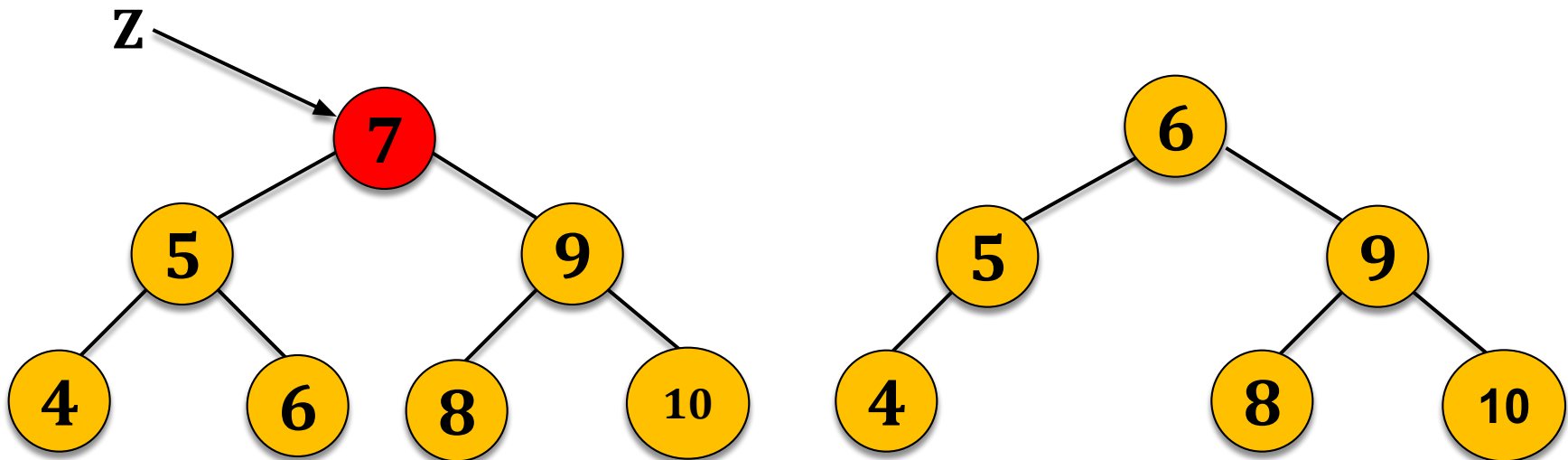


52

Case 2: removing a node with 2 subtrees

Action: - replace the node's value with the max value in the left subtree and delete the max node in the left subtree

Removing 7



Z points to the node which is to be deleted from the tree

BST – Deletion – Case 3

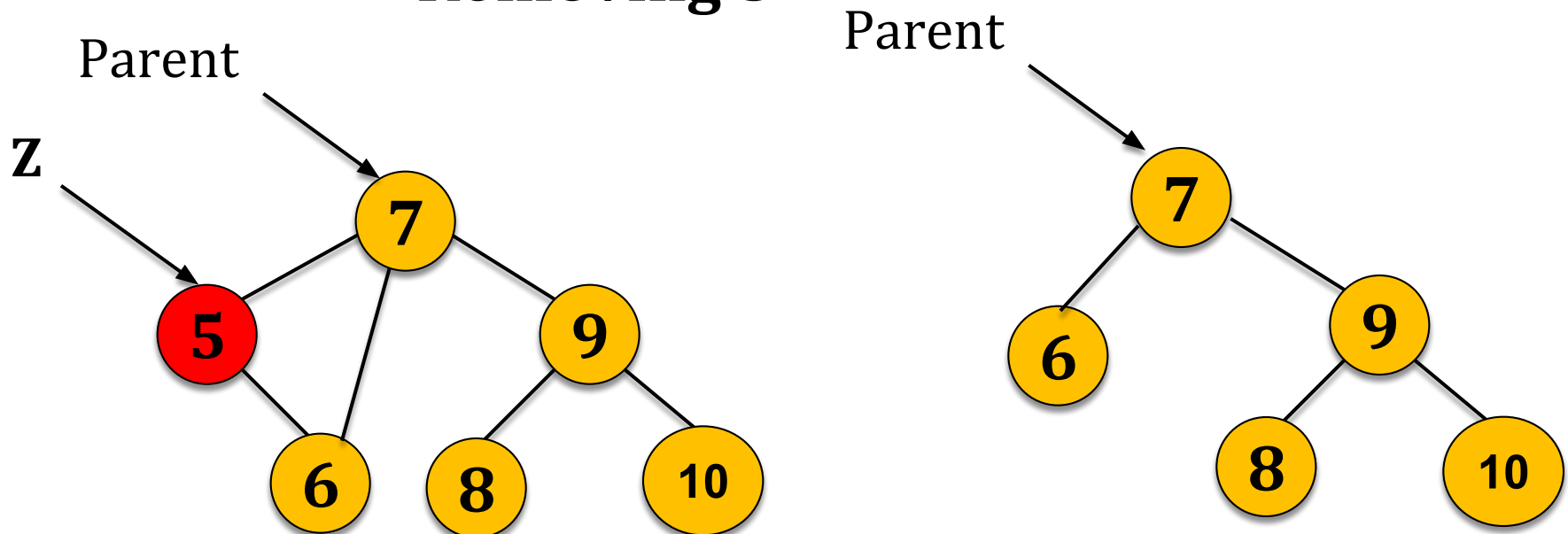


53

Case 3: if the node has no left child

Action: link the parent of the node to the right node of non-empty subtree and delete the node

Removing 5



Z points to the node which is to be deleted from the tree

BST – Deletion – Case 4

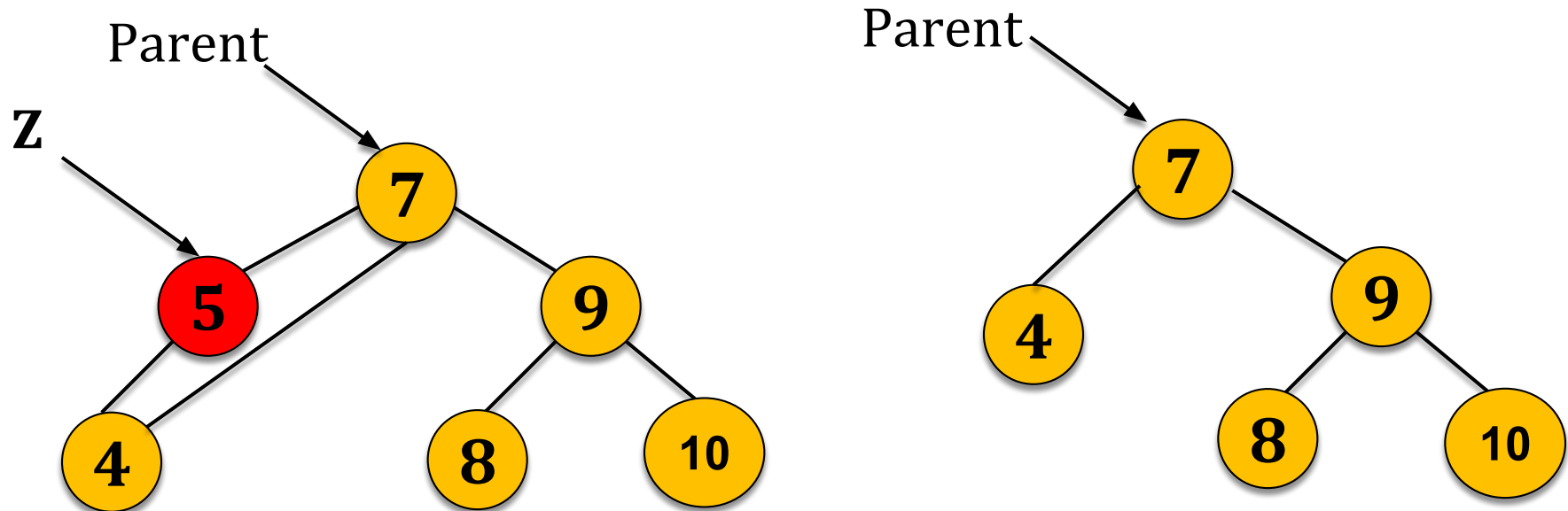


54

Case 4: if the node has no right child

Action: link the parent of the node to the left node of non-empty subtree and delete the node

Removing 5



Z points to the node which is to be deleted from the tree

BST Deletion Pseudo code



55

```
void delete(node *root, key)
{
  IF (root == NULL) THEN
    exit
  END IF
```

```
  IF (root->data > key) THEN
    delete(root->left, key)
  ELSE IF (root->data < key) THEN
    delete(root->right, key)
  ELSE
    deletenode(root)
  ENDIF
}
```

```
void deleteNode(node *root)
{
  IF (root->left == NULL AND root->right == NULL) //Case - 1
    free(root) [free is the C function to release the memory allotted to the node]
    exit
  ENDIF

  IF (root->left != NULL) THEN //Case - 4 and Case - 2
    struct node *temp = root->left;
    WHILE(temp->right != NULL)
      temp = temp->right;
    ENDWHILE
    root->data = temp->data;
    free(temp)
  ELSE //Case 3
    struct node *temp = root->right;
    WHILE(temp->left != NULL)
      temp = temp->left;
    ENDWHILE
    root->data = temp->data;
    free(temp);
  ENDIF
}
```

BST C Implementation



56



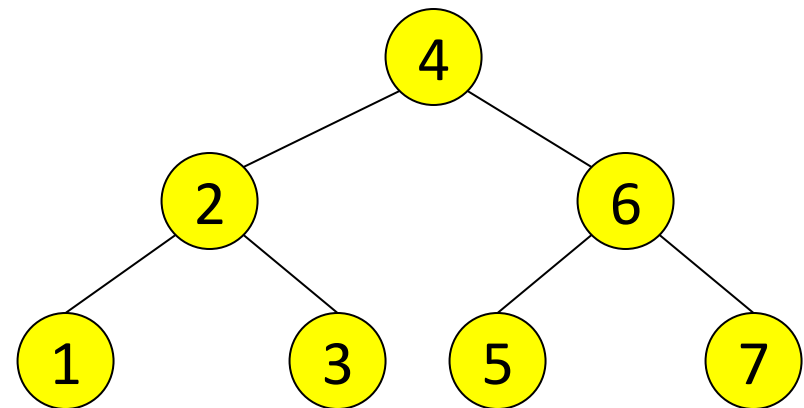
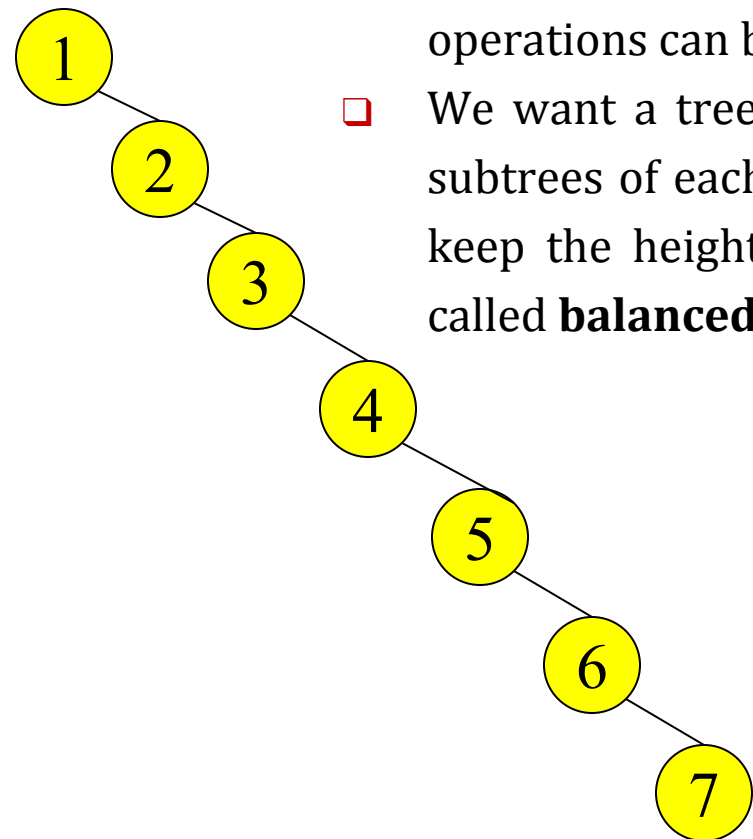
BST

Balanced Binary Tree



57

- ❑ The disadvantage of a binary search tree is that its height can be as large as $N-1$ where N is the number of nodes. This means that the time needed to perform insertion and deletion and many other operations can be $O(N)$ in the worst case.
- ❑ We want a tree with small height and this can be attained if both subtrees of each node have **roughly the same height**. So goal is to keep the height of a binary search tree $O(\log N)$. Such trees are called **balanced binary trees**



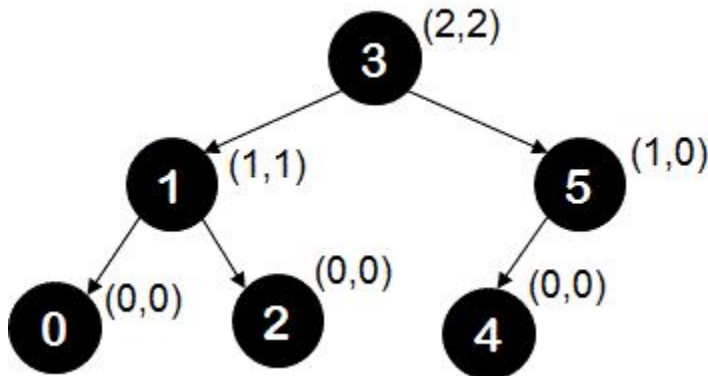
How to check a Binary Tree is balanced or not?

58

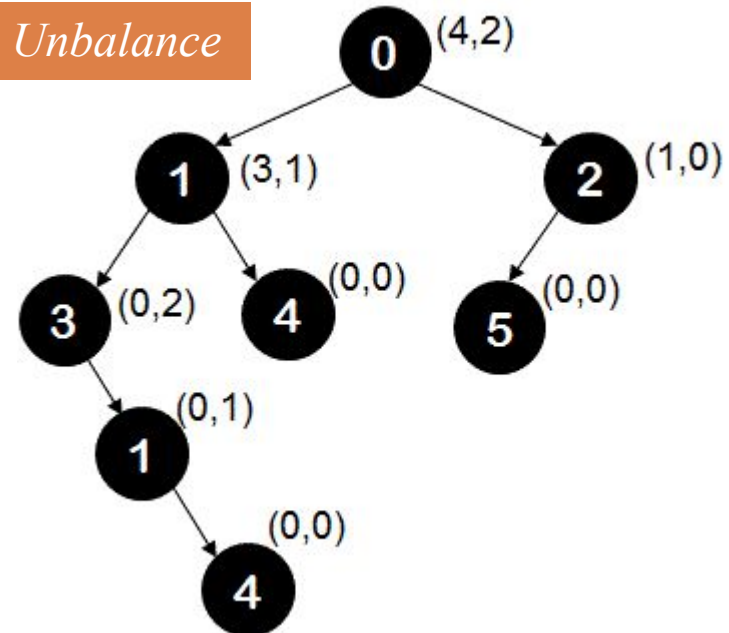
A binary tree is called a balanced binary tree if it satisfies following conditions -

- If at any given node, absolute difference of height of left sub-tree and height of right sub-tree (called as **balance factor**) is not greater than 1 (i.e. should be -1 or 1 or 0)

Balance



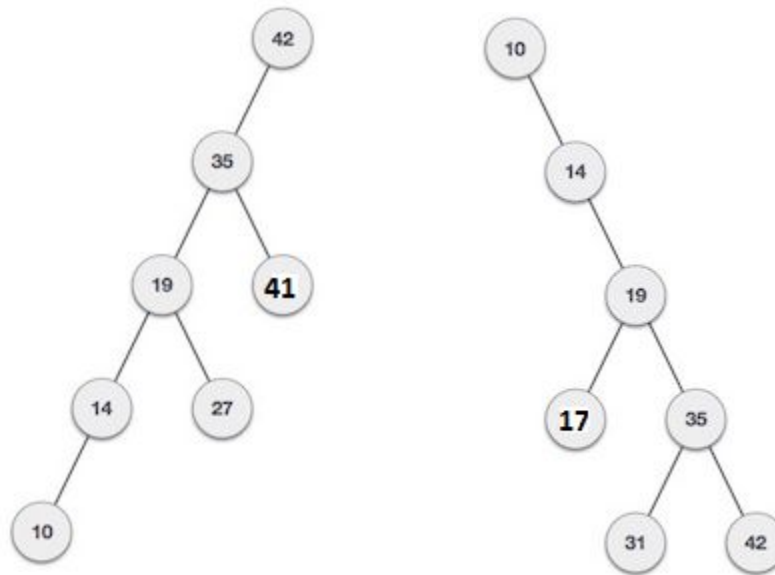
Unbalance



AVL Tree

59

What if the input to binary search tree comes in sorted (ascending or descending) manner? It will then look like this –



It is observed that BST's worst-case performance closes to linear search algorithms, that is $O(n)$. In real time data we cannot predict data pattern and their frequencies. So a need arises to balance out existing BST. Named after their inventor **Adelson, Velski & Landis**, AVL trees are **height balancing binary search tree**. AVL tree checks the height of left and right sub-trees and assures that the difference is not more than 1. This difference is called Balance Factor. **Balance Factor = height(left-subtree) – height(right-subtree)**



AVL Rotation Technique

60

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using **rotation techniques**.

To make itself balanced, an AVL tree may perform four kinds of rotations –

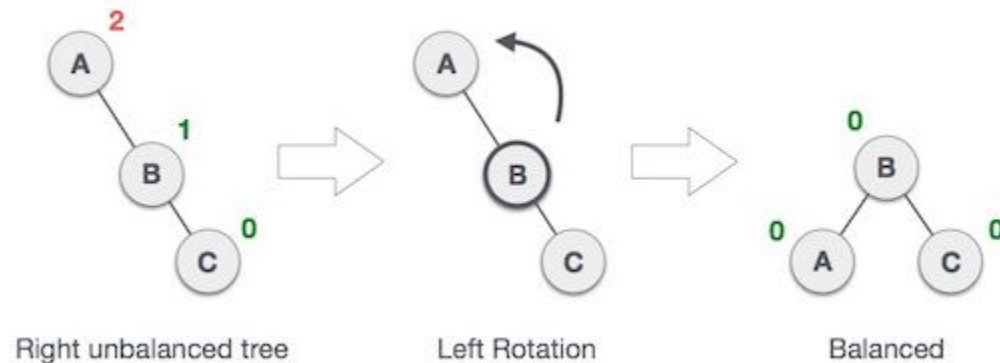
- ☐ Left rotation
- ☐ Right rotation
- ☐ Left-Right rotation
- ☐ Right-Left rotation

First two rotations are **single rotations** and next two rotations are **double rotations**.

AVL Tree Left Rotation

61

If a tree become unbalanced, when a node is inserted into the right subtree of right subtree, then we perform single left rotation –

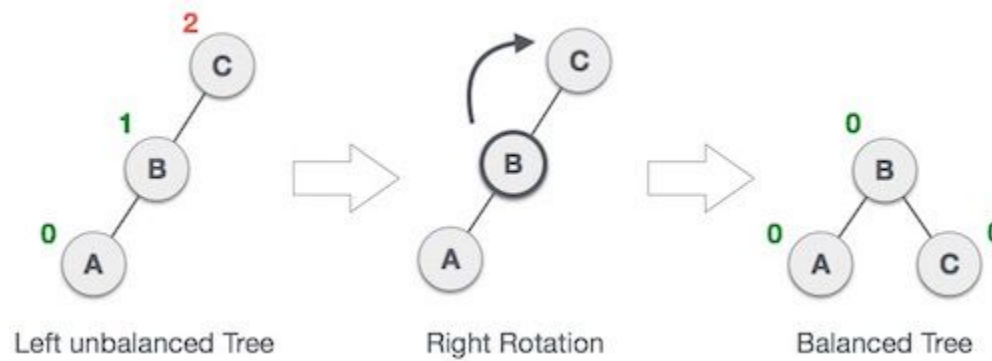


In the above example, node A has become unbalanced as a node is inserted in right subtree of A's right subtree. We perform left rotation by making A left-subtree of B.

AVL Tree Right Rotation

62

AVL tree may become unbalanced if a node is inserted in the left subtree of left subtree. The tree then needs a right rotation.

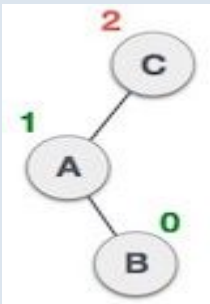
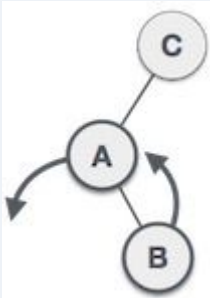


As depicted, the unbalanced node becomes right child of its left child by performing a right rotation.

AVL Tree Left-Right Rotation

63

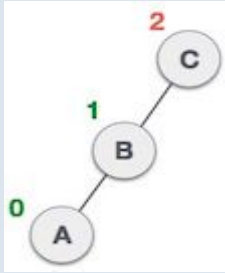
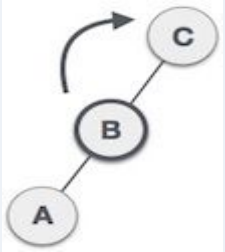
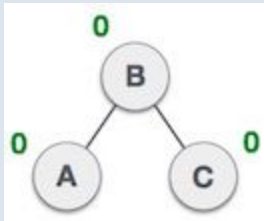
Double rotations are slightly complex version of already explained versions of rotations. A left-right rotation is combination of left rotation followed by right rotation.

State	Action
	<p>A node has been inserted into right subtree of left subtree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.</p>
	<p>We first perform left rotation on left subtree of C. This makes A, left subtree of B.</p>

AVL Tree Left-Right Rotation cont...



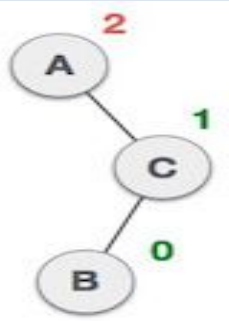
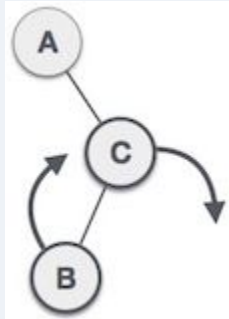
64

State	Action
	Node C is still unbalanced but now, it is because of left-subtree of left-subtree.
	We shall now right-rotate the tree making B new root node of this subtree. C now becomes right subtree of its own left subtree.
	The tree is now balanced.

AVL Tree Right-Left Rotation

65

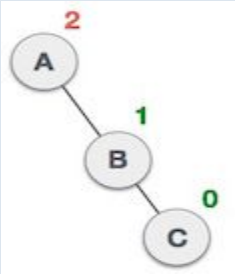
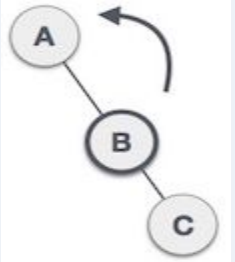
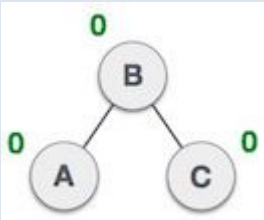
A right-left rotation is combination of right rotation followed by left rotation.

State	Action
	<p>A node has been inserted into left subtree of right subtree. This makes A an unbalanced node, with balance factor 2.</p>
	<p>First, we perform right rotation along C node, making C the right subtree of its own left subtree B. Now, B becomes right subtree of A.</p>

AVL Tree Left-Right Rotation cont...



66

State	Action
	Node A is still unbalanced because of right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making B the new root node of the subtree. A becomes left subtree of its right subtree B.
	The tree is now balanced.



AVL Tree Insertion and Deletion

67

Insertio

- ❑ After inserting a node, check each of the node's ancestors for consistency with the AVL rules.
- ❑ For each node checked, if the balance factor remains 1, 0, or -1 then no rotations are necessary. Otherwise, it's unbalanced.
- ❑ After each insertion, at most two tree rotations are needed to restore the entire tree.

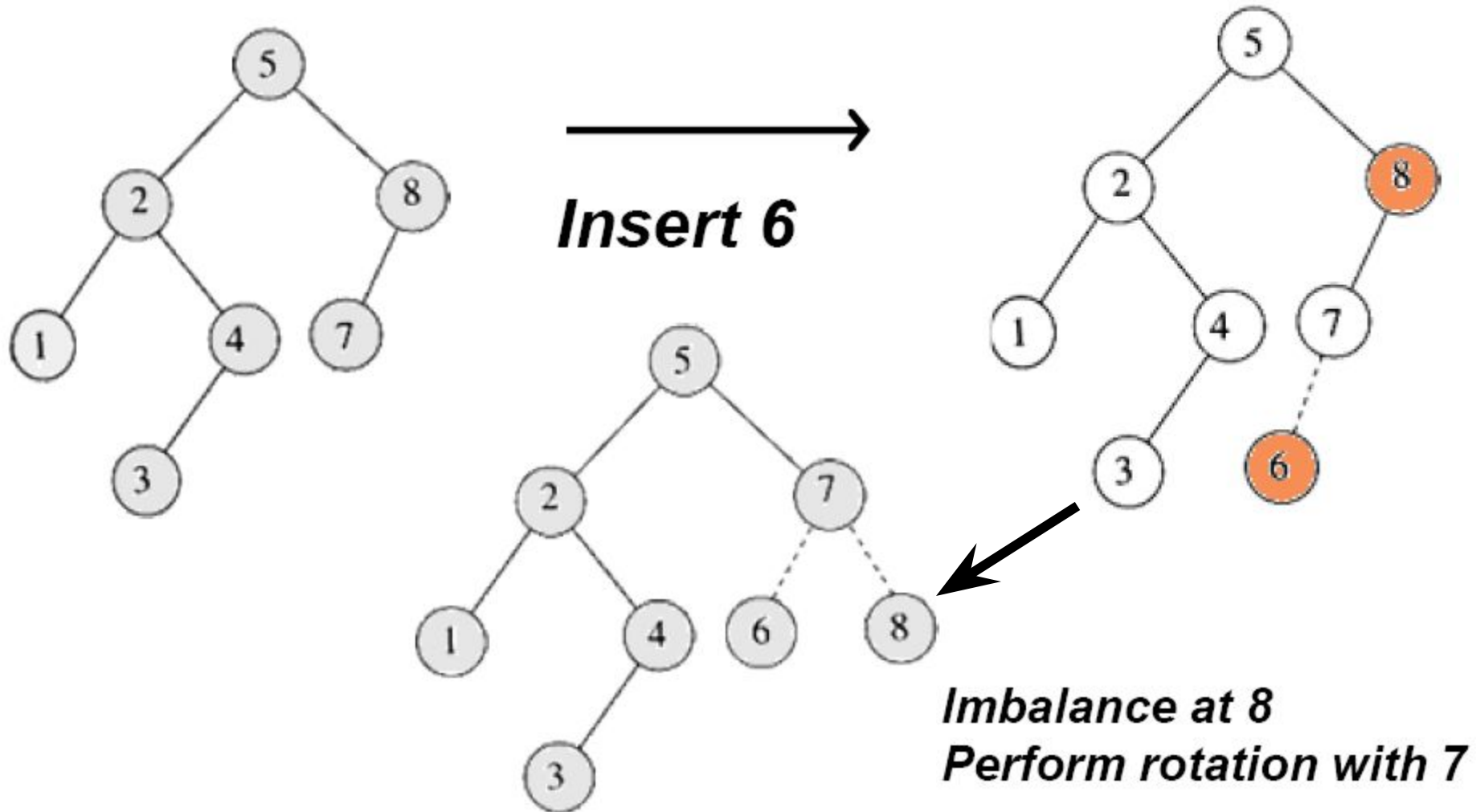
Deletio

- ❑ If a node is a leaf, remove it.
- ❑ If the node is not a leaf, replace it with either the largest in its left subtree (i.e. the rightmost) or the smallest in its right subtree (i.e. the leftmost), and remove that node.
- ❑ After deletion, retrace the path from parent of the replacement to the root, adjusting the balance factors as needed.

AVL Tree Insertion Example

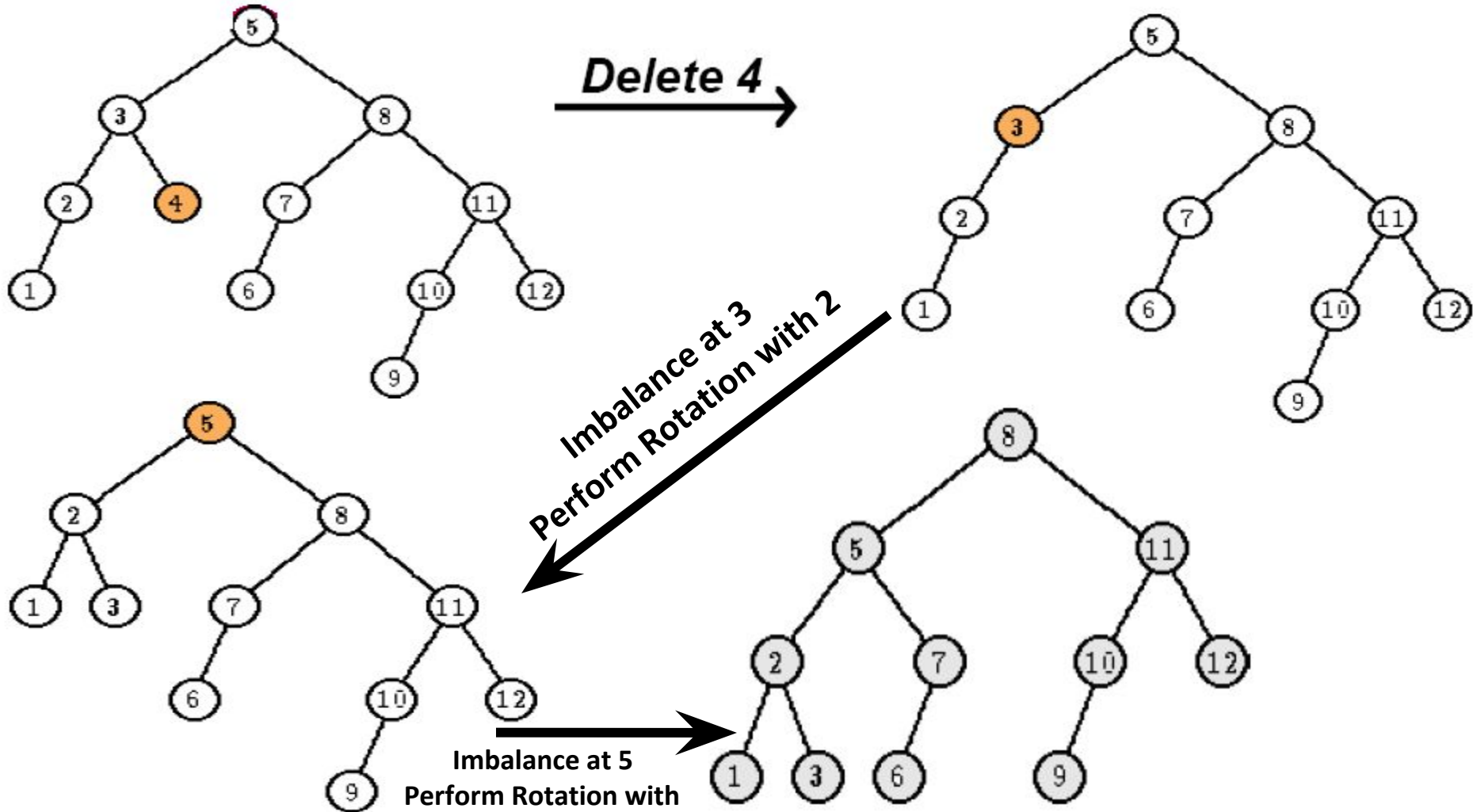


68



AVL Tree Deletion Example

69





AVL Tree Insertion and Deletion Simulation

70

[https://www.cs.usfca.edu/~galles/visualization/AVLtree.htm](https://www.cs.usfca.edu/~galles/visualization/AVLtree.html)
l

AVL Tree C Implementation



71



AVL Tree

Multi-way Search Tree



72

- ❑ A binary search tree has one value in each node and two subtrees. This notion easily generalizes to an M -way search tree, which has $(M-1)$ values per node and M subtrees. M is called the degree of the tree. A binary search tree, therefore, has degree 2.
- ❑ In fact, it is not necessary for every node to contain exactly $(M-1)$ values and have exactly M subtrees. In an **M -way** subtree a node can have anywhere from 1 to $(M-1)$ values and the number of (non-empty) subtrees can range from 0 (for a leaf) to $1 + (\text{the number of values})$. M is thus a fixed upper limit on how much data can be stored in a node.
- ❑ The values/keys in a node are stored in ascending order, $K_1 < K_2 < \dots < K_k$ ($k \leq M-1$) and all the values in K_1 's left subtree are less than K_1 ; all the keys in K_k 's subtree are greater than K_k ; and all the values in the subtree between $K_{(i)}$ and $K_{(i+1)}$ are greater than $K_{(i)}$ and less than $K_{(i+1)}$.



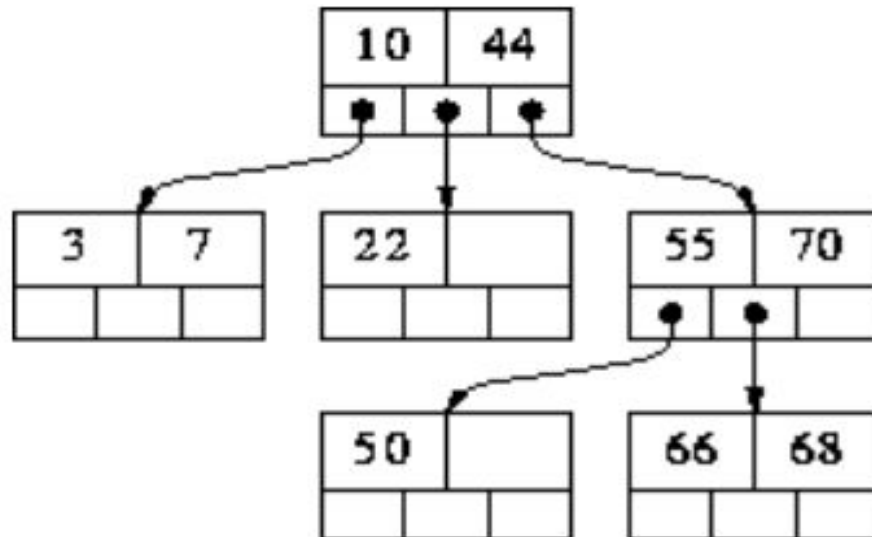
Keys are ordered such that:

$$k_1 < k_2 < \dots < k_{M-1}$$

3-way Search Tree Example



73



Search for X in m -way search tree



74

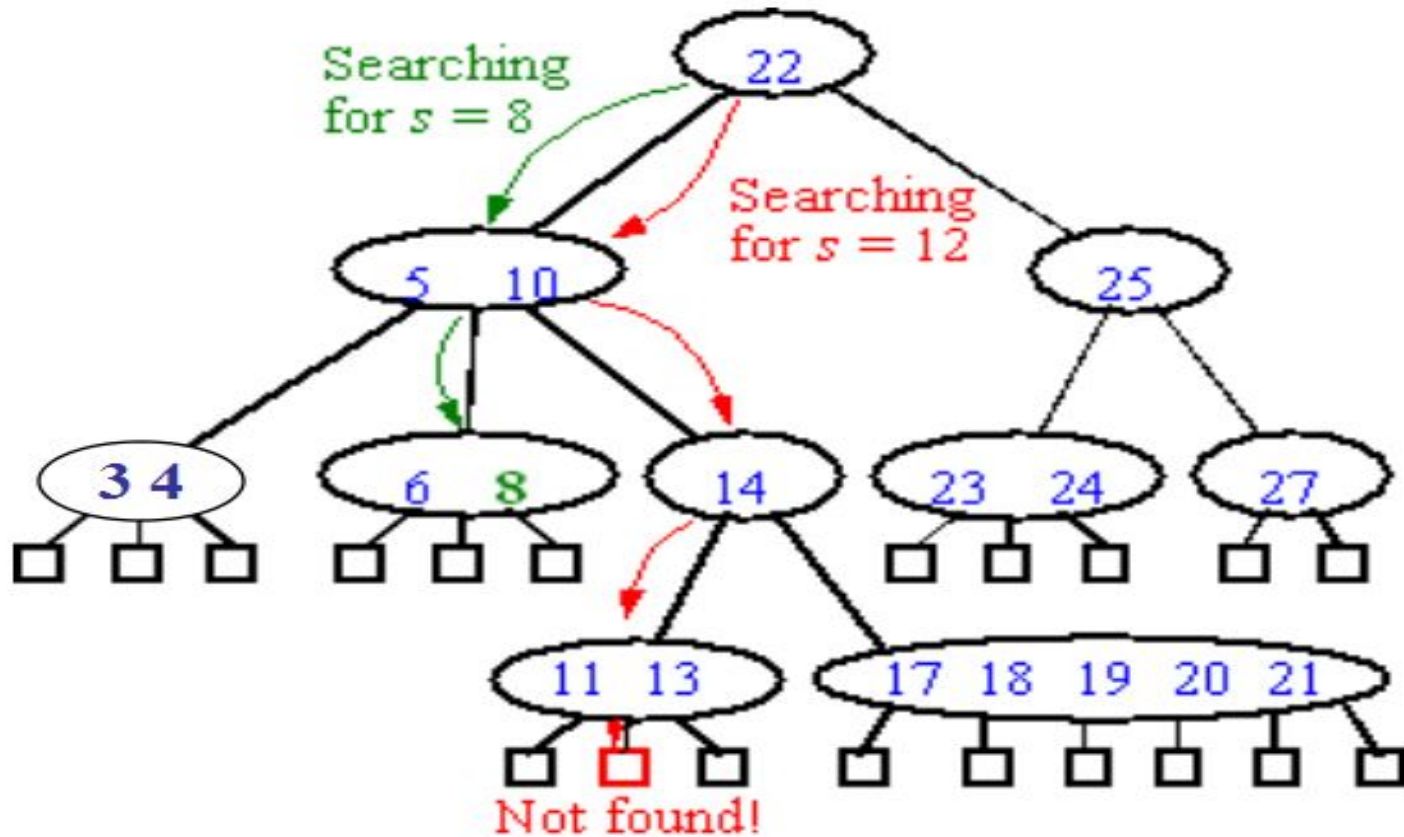
At a node consisting of values $V_1 \dots V_k$, there are four possible cases:

- If $X < V_1$, recursively search for X in the subtree that is left of V_1
- If $X > V_k$, recursively search for X in the subtree that is right of V_k
- If $X = V_i$ for some i , then we are done (X has been found)
- Else, for some i , $V_i < X < V_{i+1}$. In this case recursively search for X in the subtree that is between V_i and V_{i+1}

Searching an m-way Search Tree



75



Searching for a key in an m-way search tree is similar to that of binary search tree

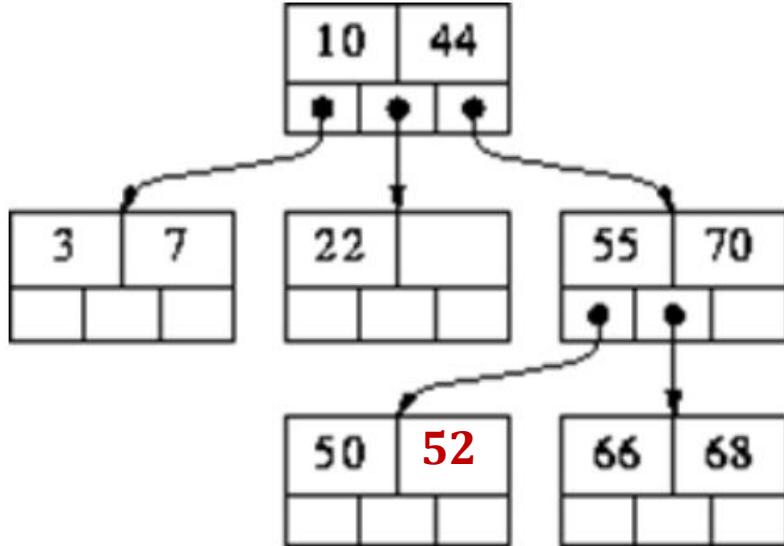
Insert into m-way search tree



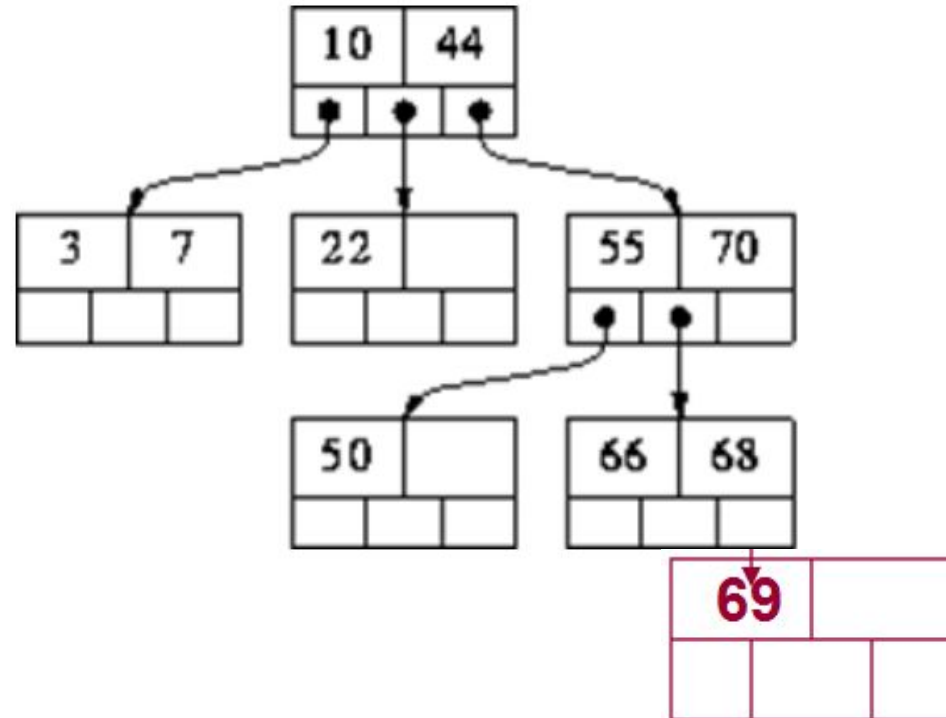
76

- ❑ The algorithm for binary search tree can be generalized
- ❑ Follow the search path and perform one of the following
 - ❑ Add new key or value into the last leaf
 - ❑ Add a new leaf if the last leaf is fully occupied

Add 52



Add 69



Deletion from m-way search tree

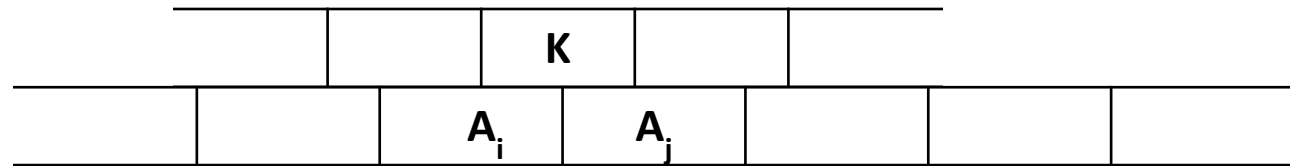


77

Let K be the key to be deleted from the m-way search tree.

K : Key

A_i, A_j : Pointers to sub-tree



Deletion Cases

- [1] If $(A_i = A_j = \text{NULL})$ then delete K
- [2] If $(A_i \neq \text{NULL}, A_j = \text{NULL})$ then choose the largest of the key elements K' in the child node pointed to by A_i and replace K by K' .
- [3] If $(A_i = \text{NULL}, A_j \neq \text{NULL})$ then choose the smallest of the key element K'' from the subtree pointed to by A_j , delete K'' and replace K by K'' .
- [4] If $(A_i \neq \text{NULL}, A_j \neq \text{NULL})$ then choose the largest of the key elements K' in the subtree pointed to by A_i or the smallest of the key element K'' from the subtree pointed to by A_j to replace K .

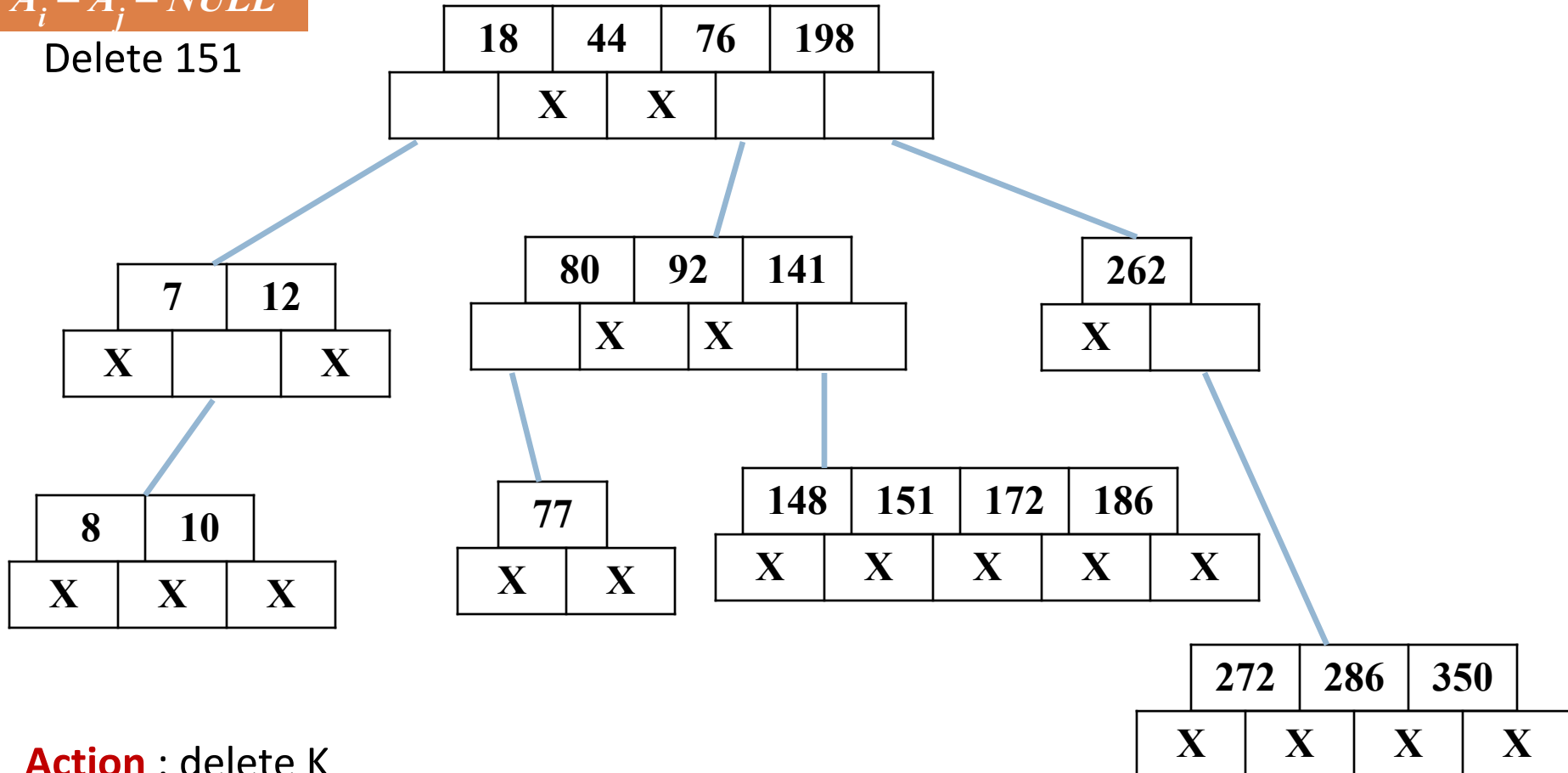
Deletion Case - 1



78

$A_i = A_j = \text{NULL}$

Delete 151



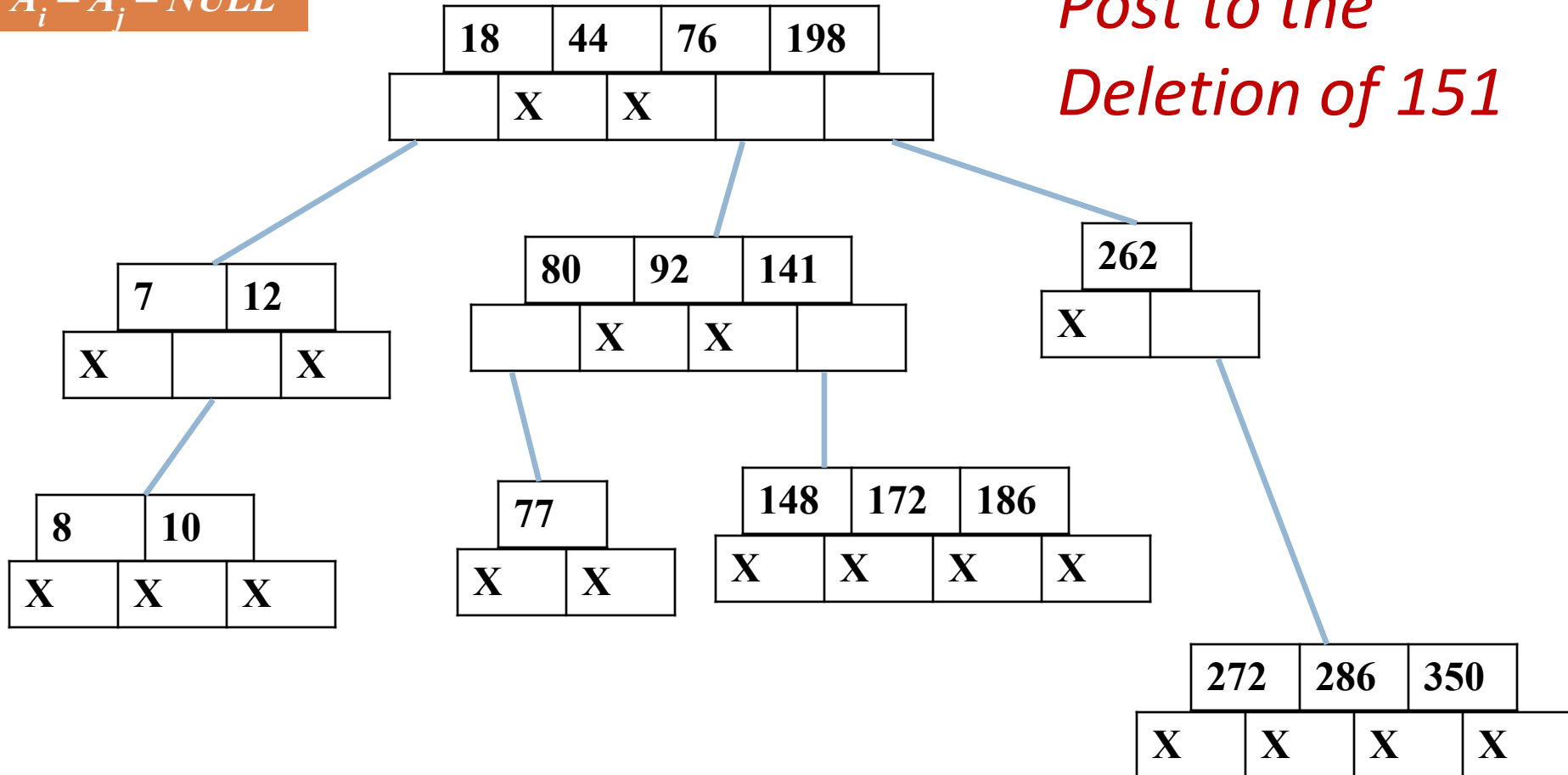
Deletion Case - 1 Result



79

$A_i = A_j = \text{NULL}$

*Post to the
Deletion of 151*



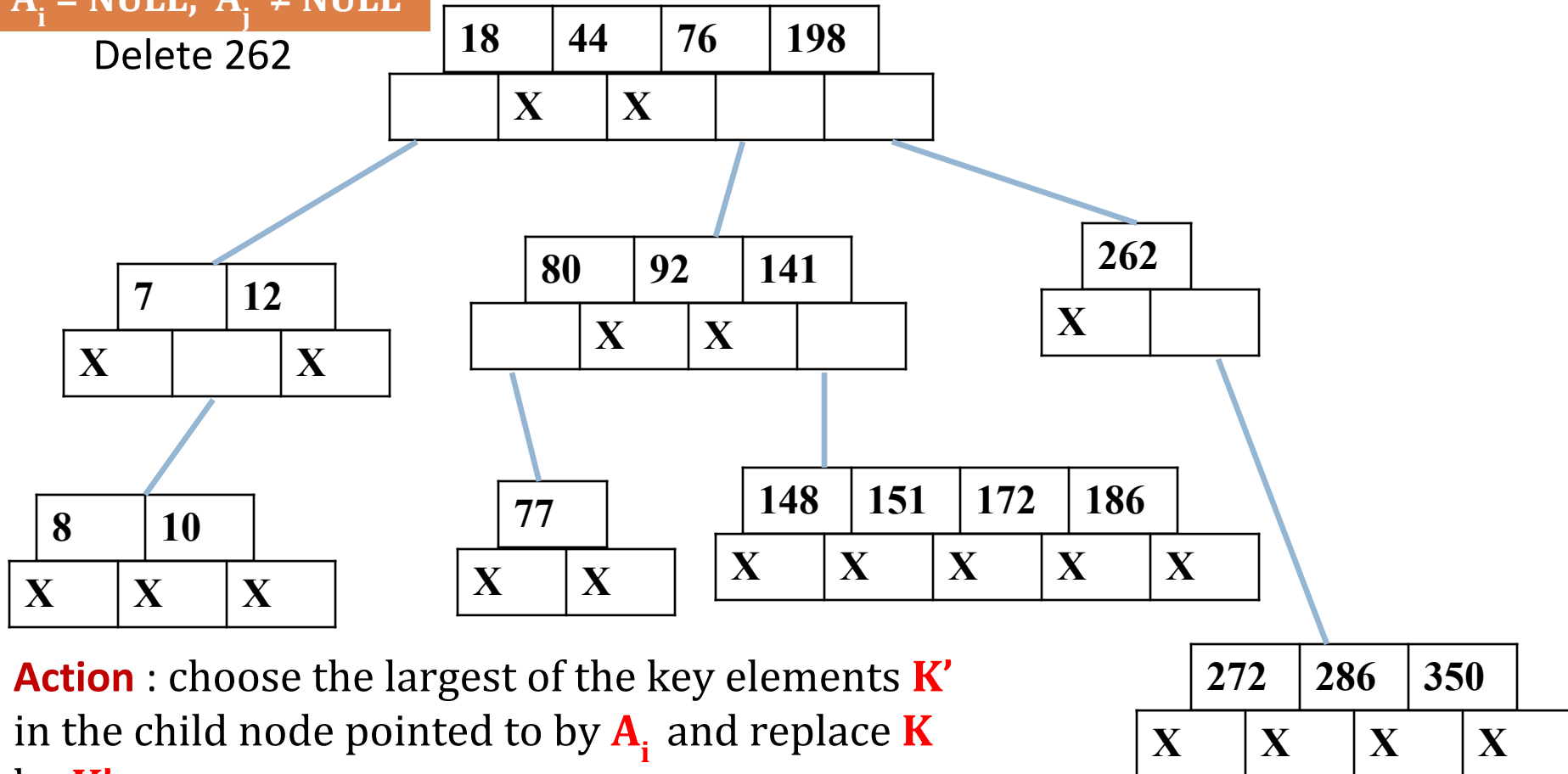
Deletion Case - 2



80

$A_i = \text{NULL}, A_j \neq \text{NULL}$

Delete 262



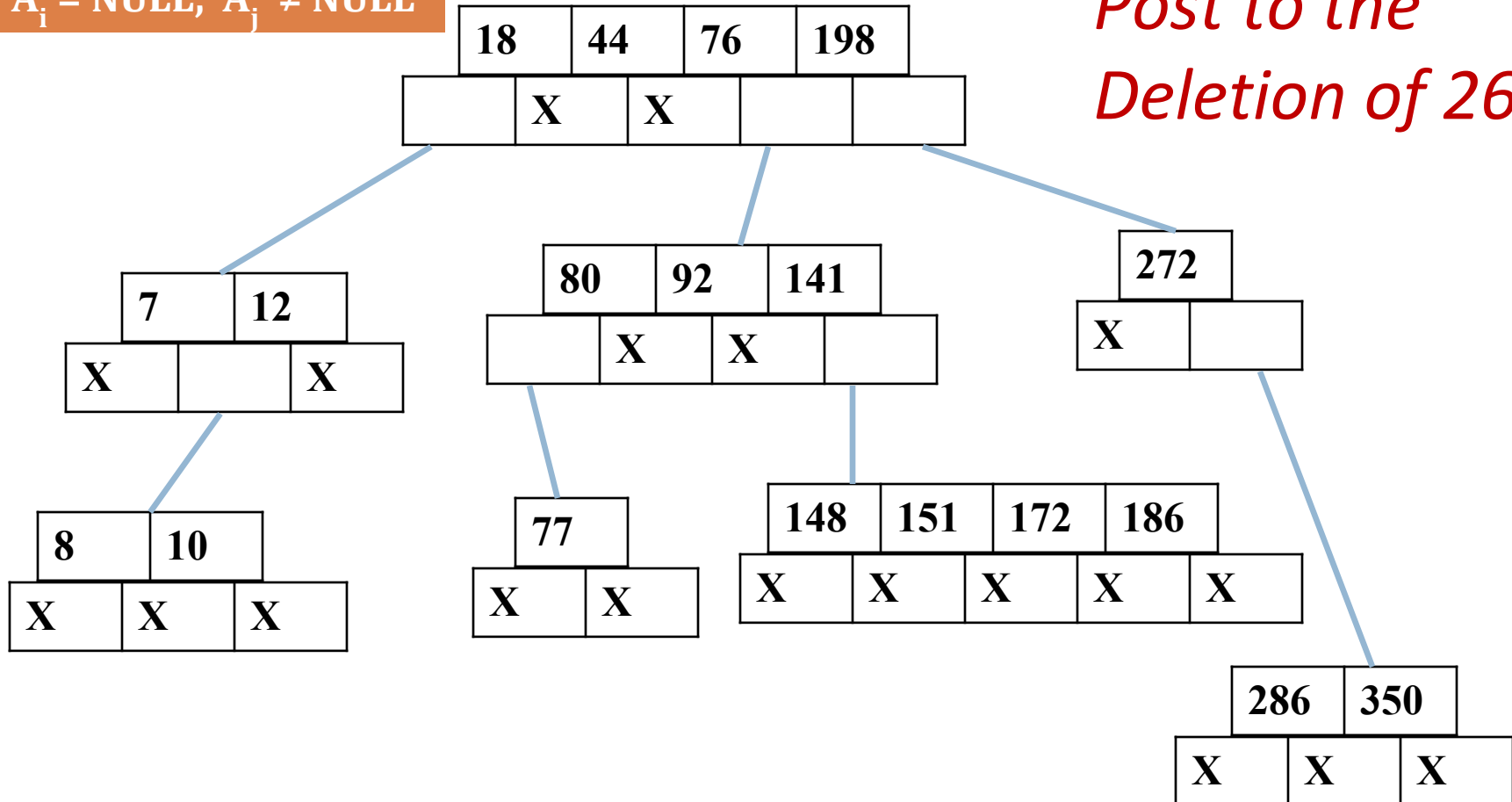
Deletion Case - 2 Result



81

$A_i = \text{NULL}, A_j \neq \text{NULL}$

*Post to the
Deletion of 262*



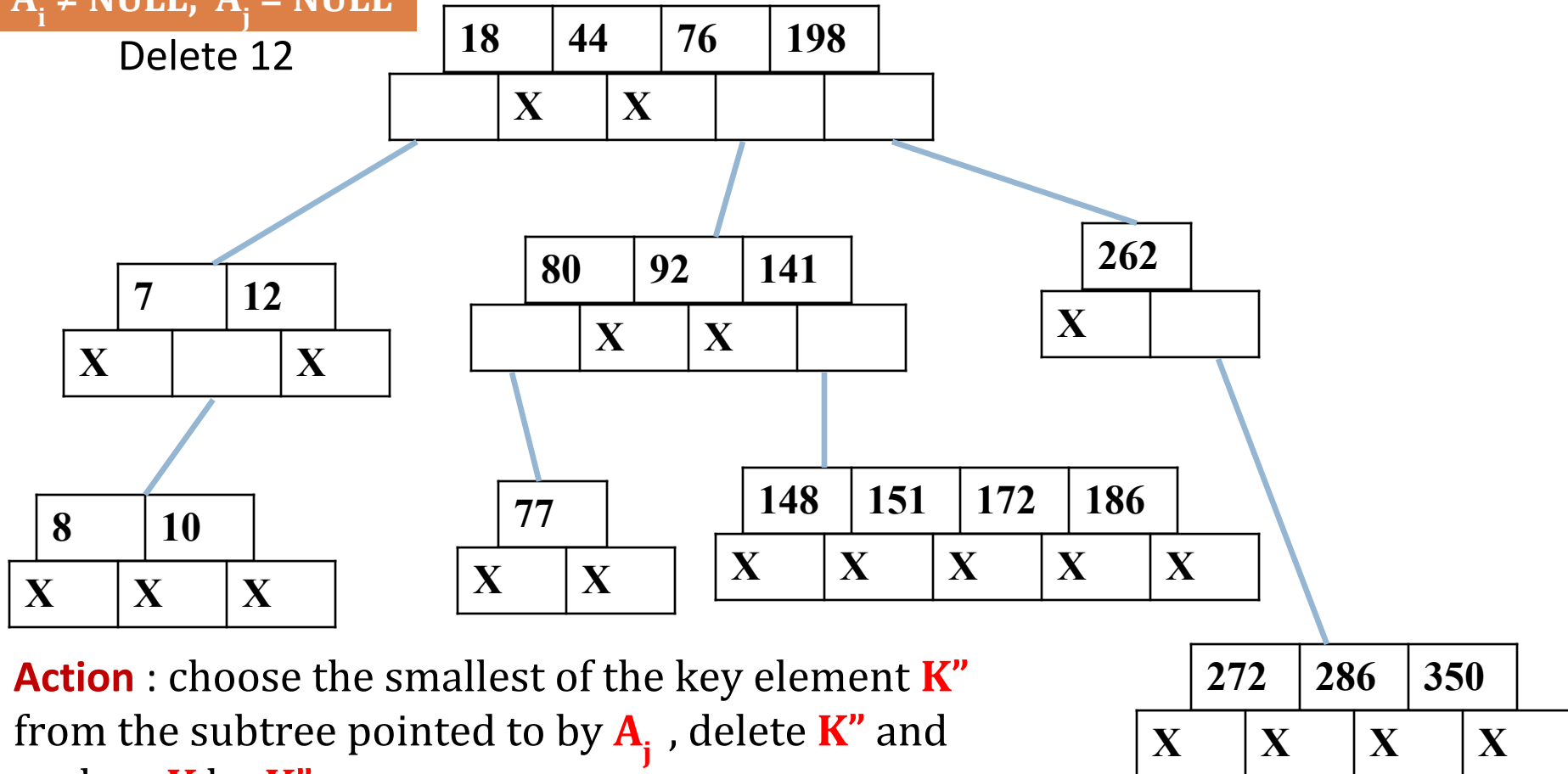
Deletion Case - 3



82

$A_i \neq \text{NULL}, A_j = \text{NULL}$

Delete 12



Action : choose the smallest of the key element **K''** from the subtree pointed to by A_j , delete **K''** and replace **K** by **K''**

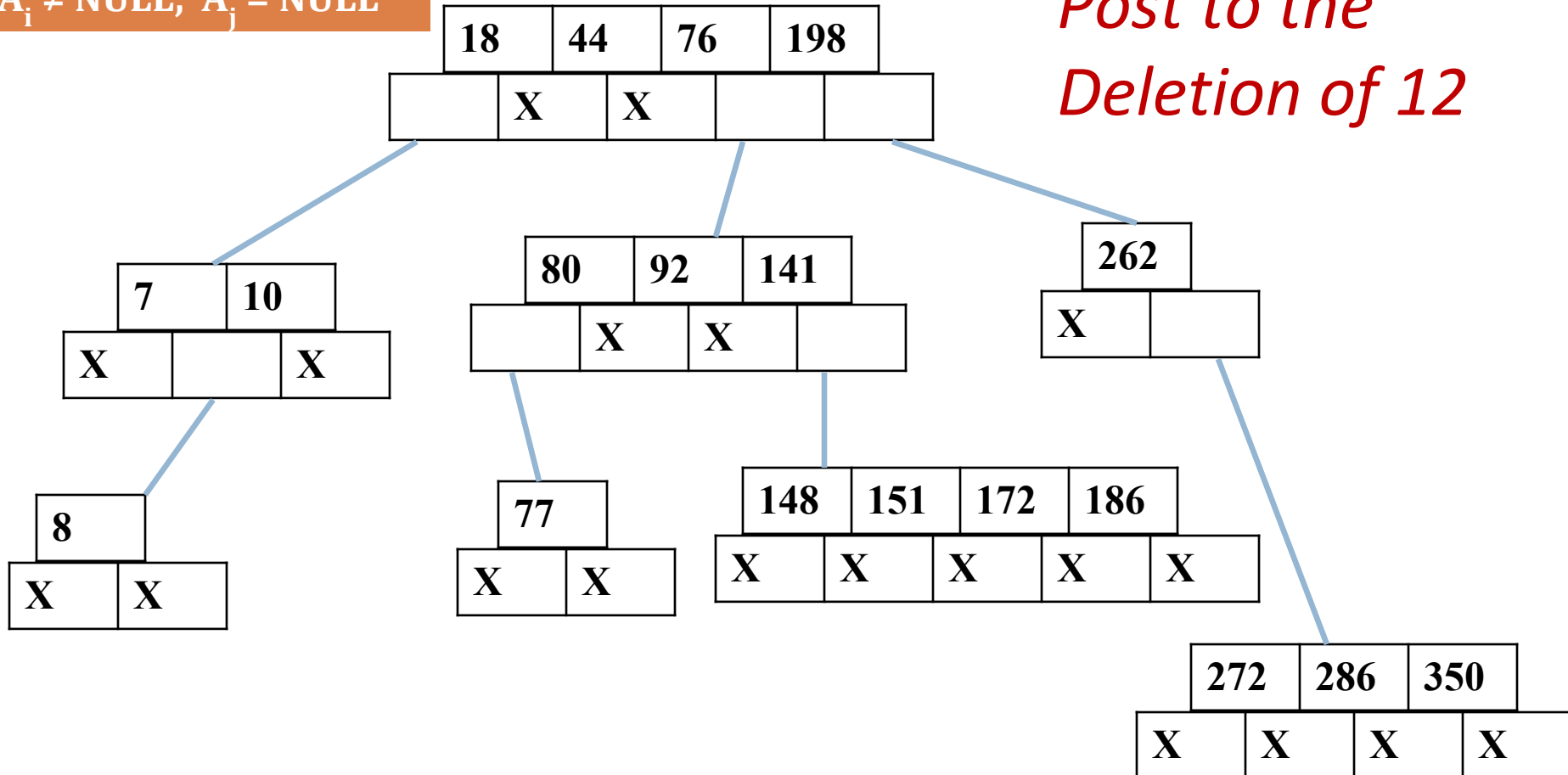
Deletion Case - 3 Result



83

$A_i \neq \text{NULL}, A_j = \text{NULL}$

*Post to the
Deletion of 12*



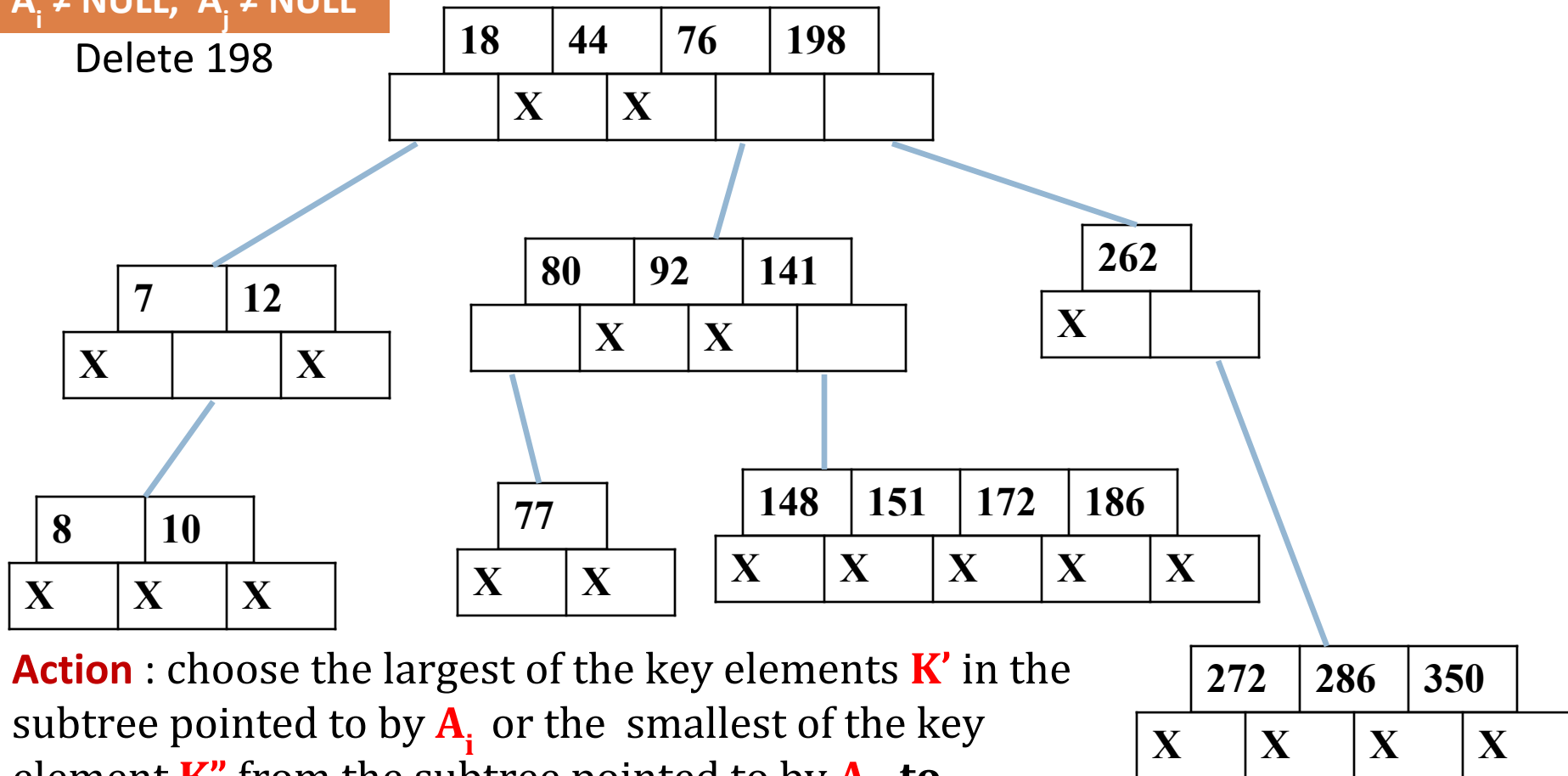
Deletion Case - 4



84

$A_i \neq \text{NULL}, A_j \neq \text{NULL}$

Delete 198



Action : choose the largest of the key elements **K'** in the subtree pointed to by A_i or the smallest of the key element **K''** from the subtree pointed to by A_j to replace **K**.

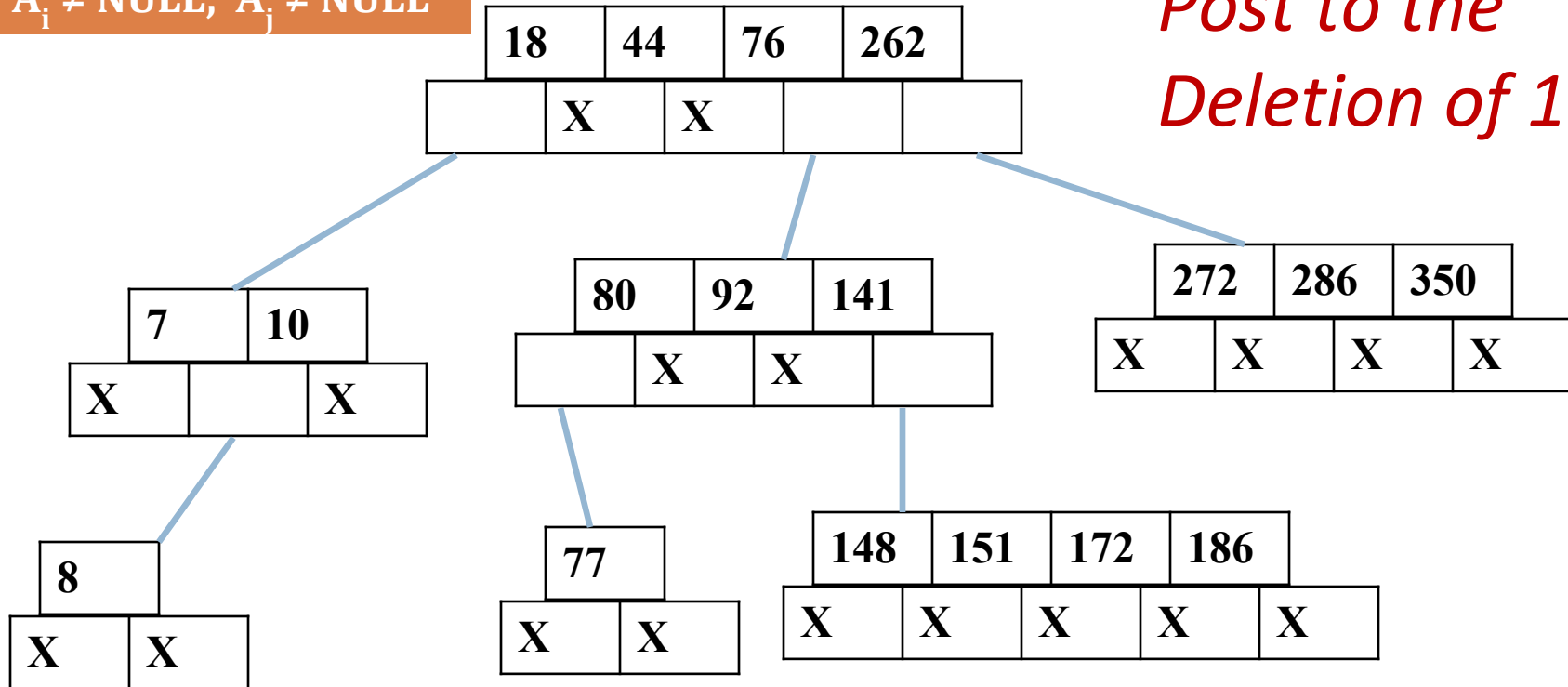
Deletion Case - 4 Result



85

$A_i \neq \text{NULL}, A_j \neq \text{NULL}$

*Post to the
Deletion of 198*



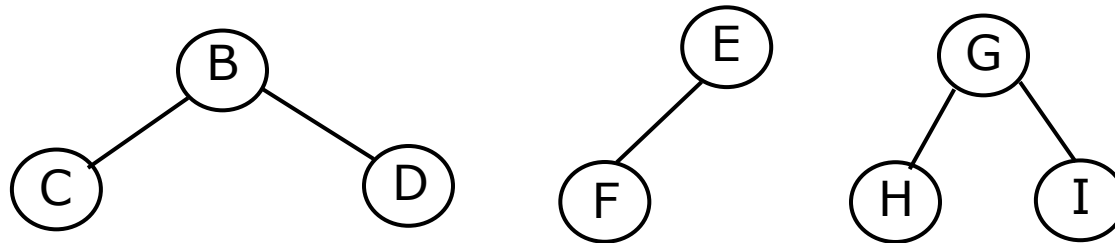
Forest



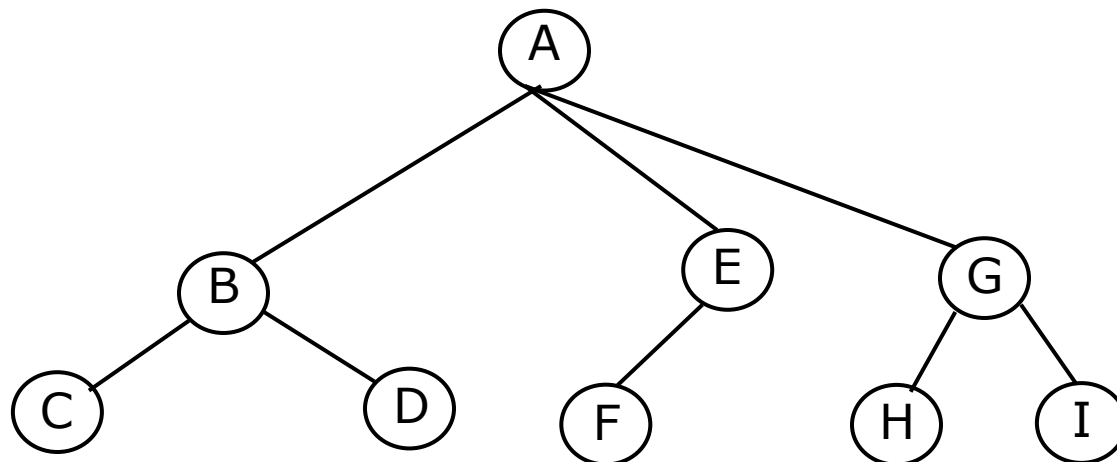
86

A **Forest F** defined to be an ordered collection of zero or more distinct **general** trees.

Example –



A forest can be converted to a **general tree** (or simply tree i.e. number of subtrees for any node is not required to be 0, 1, or 2) by adding a single node to serve as the root node of the tree in which each of the original trees are sub-tree. Example –

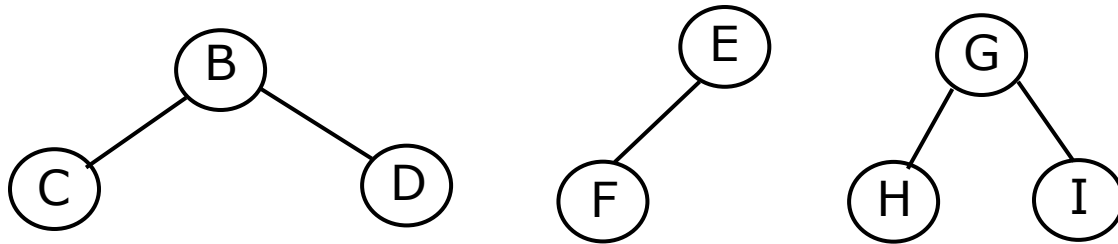


Forest cont...



87

Conversely, deleting the root from a tree leaves behind a forest consisting of its subtrees. (Obviously, this is how we got our forest from our original tree.)



Can we provide or say, nodes with 1000 pointer fields in order to be able to represent general trees where nodes might have up to 1000 children? In any given case, most of the fields of all the node records will be unused, so a great deal of memory is required - and wasted. And what if we want to introduce a node with 1001 children? We have to reconfigure the implementation to accommodate this one "**anomalous**" node. Surely, this is not satisfactory. Remarkably, a forest can be represented by a binary tree. Note that the resulting tree is a **binary tree** but **not a binary search tree**. Obviously, an empty forest can be represented by the empty binary tree.

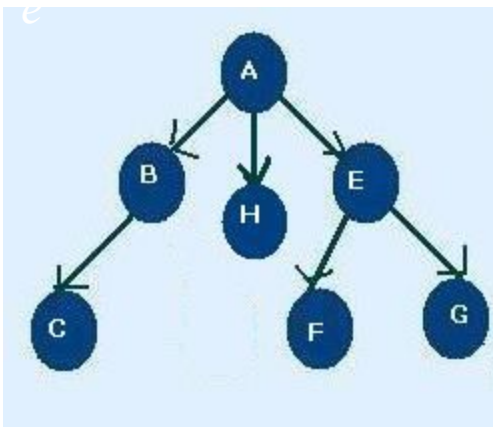
Conversion of General Tree to Binary Tree



88

1. Root node of general tree becomes root node of Binary Tree.
2. Now consider $N_1, N_2, N_3 \dots N_n$ are child nodes of the root node in general tree. The left most child (N_1) of the root node in general tree becomes left most child of root node in the binary tree. Now Node N_2 becomes right child of Node N_1 , Node N_3 becomes right child of Node N_2 and so on in binary tree.
3. The same procedure of step 2 is repeated for each leftmost node in the general tree.

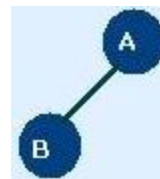
Exempl



Step 1: Root Node of General tree becomes the Root node of binary tree.



Step 2: Now Root Node (A) has three child Nodes (B, H, E) in general tree. The leftmost node (B) of the root node (A) in the general tree becomes the left most node of the root node (A) in binary tree.

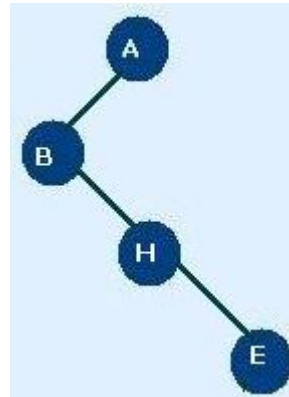


Conversion of General Tree to Binary Tree Example

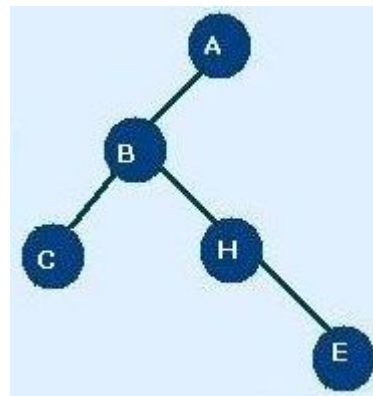


89

Step 3: Now Node H becomes the right node of B and Node E becomes the right node of H.



Step 4: Now Node B has only one left child node, which is C in general tree. So Node C becomes left child of Node B in binary tree.

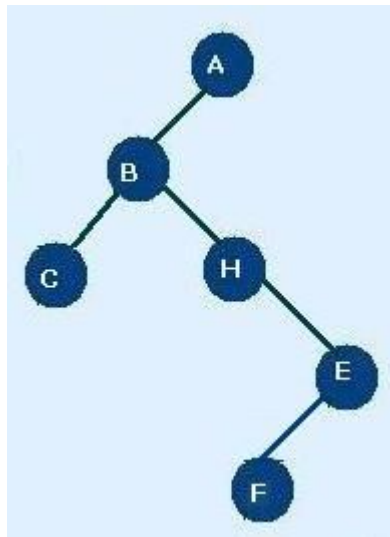


Conversion of General Tree to Binary Tree Example



90

Step 5: Now Node E has two child nodes (F, G). The leftmost node (F) of the node (E) in the general tree becomes the left most node of the node E in the binary tree.

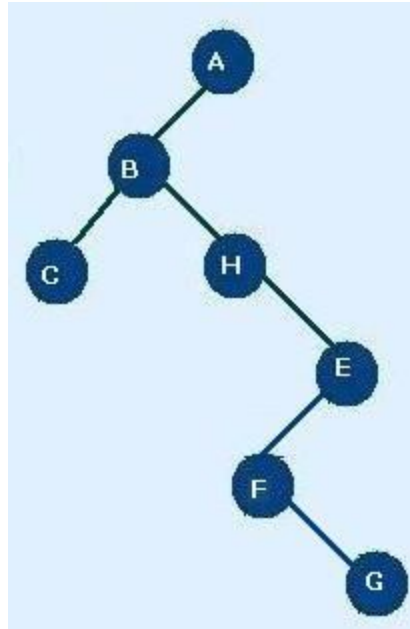


Conversion of General Tree to Binary Tree Example



91

Step 6: Now Node G becomes right node of Node F in binary tree.

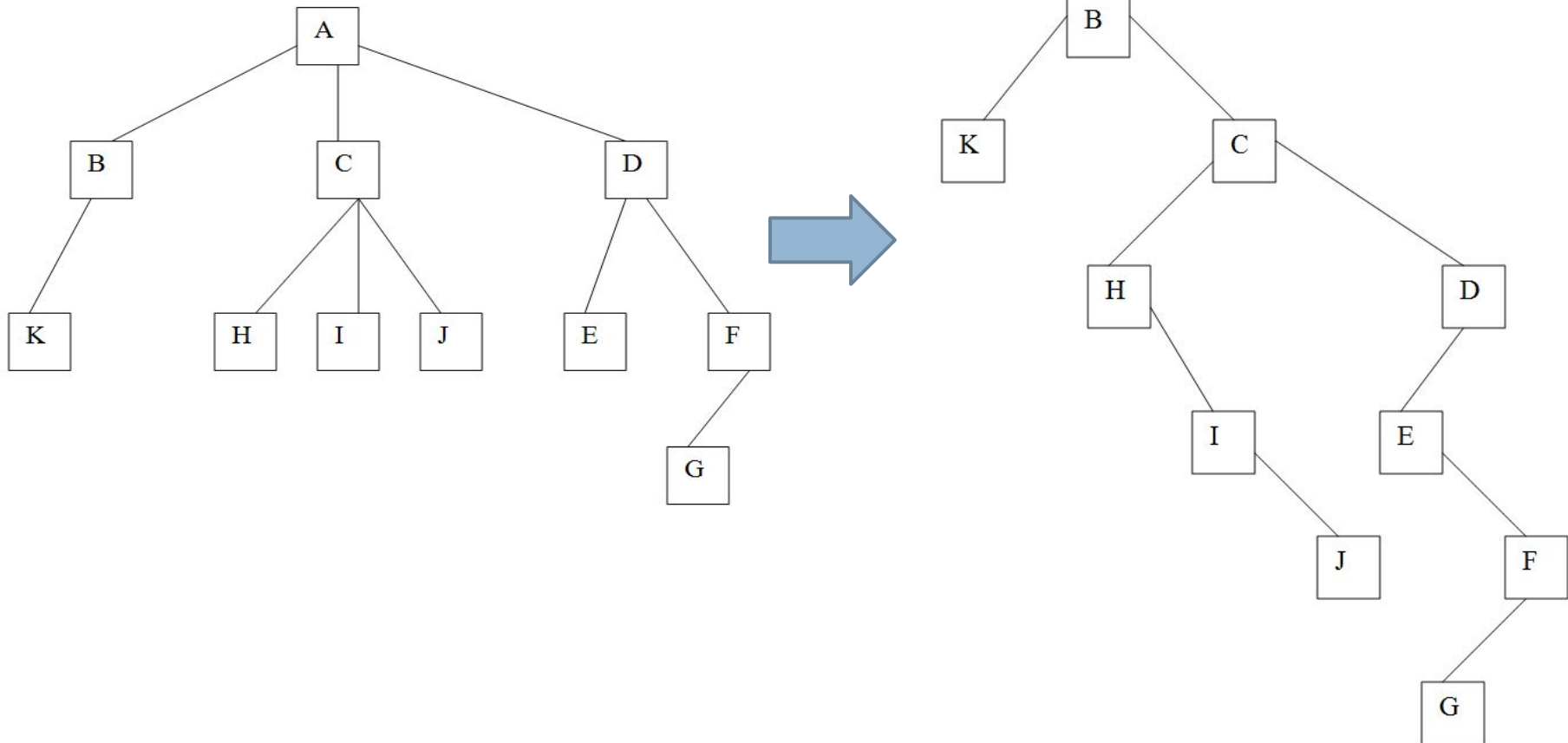


Class Work



92

Convert the following general tree to Binary Tree



B-Tree



93

M-way binary search trees have the advantage of having a node storing multiple values. However it is essential that the **height of the tree be kept as low as possible** and therefore there arises the need to maintain balanced m-way search trees. Such a **balanced m-way search tree is called B-tree**.

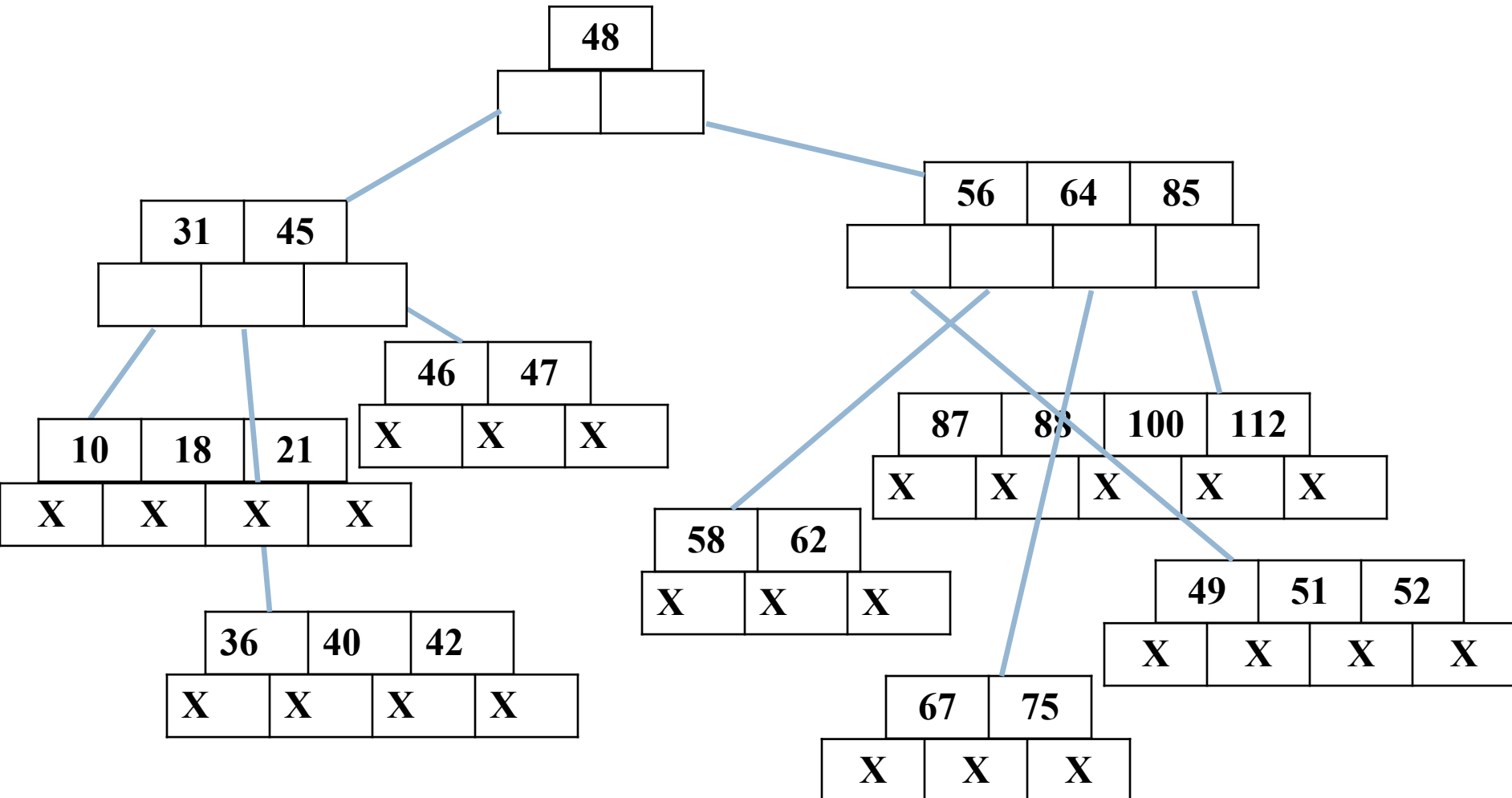
A B-tree of order m, if non empty is an m-way search tree in which

- ❑ The root has at least two child nodes and at most m child nodes
- ❑ Internal nodes except the root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes
- ❑ The number of keys in each internal node is one less than the number of child nodes and these keys partition the keys in the subtrees of nodes in a manner similar to that of m-way search trees
- ❑ All leaf nodes are on the same level

Example - B-Tree of order 5



94



B-Tree Search



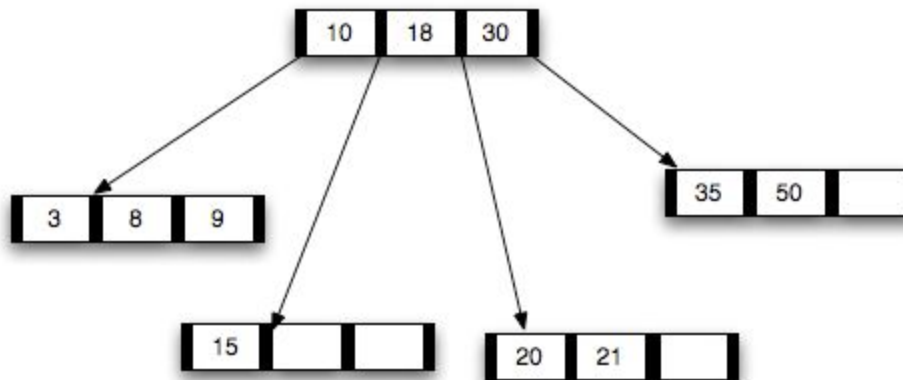
95

Searching for a key in a B-tree is similar to the one on an m-way search key. The number of accesses depends on the height h of the B-Tree.

Class Work

Draw the path to search for the key:

- ☐ 9
- ☐ 20
- ☐ 53
- ☐ 21



B-Tree Insertion



96

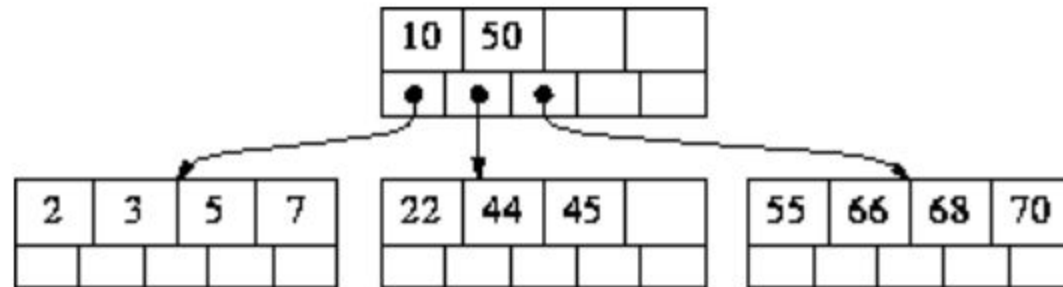
1. B-tree starts with a single root node (which is also a leaf node) at level 0.
2. Once the root node is full with $m - 1$ search key values and when attempt is made to insert another entry in the tree, the root node splits into two nodes at level 1.
3. Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes. i.e. the first $(m-1)/2$ values goes to left, the last $(m-1)/2$ values goes to right.
4. When a non-root node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes.
5. If the parent node is full, it is also split.
6. Splitting can propagate all the way to the root node, creating a new level if the root is split.

B-Tree Insertion Example

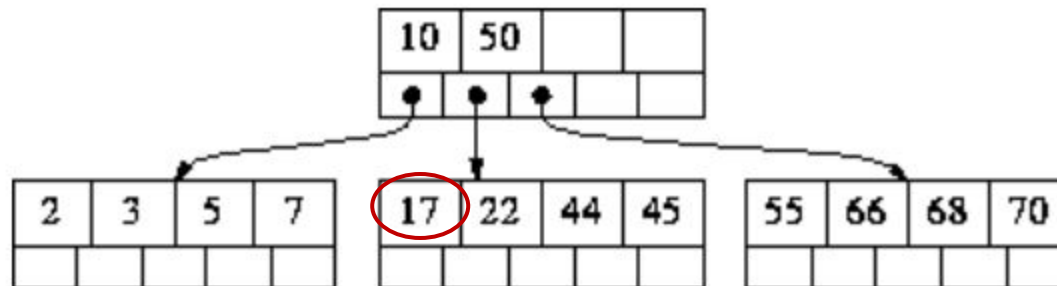


97

B-Tree where $m = 5$



Insert 17

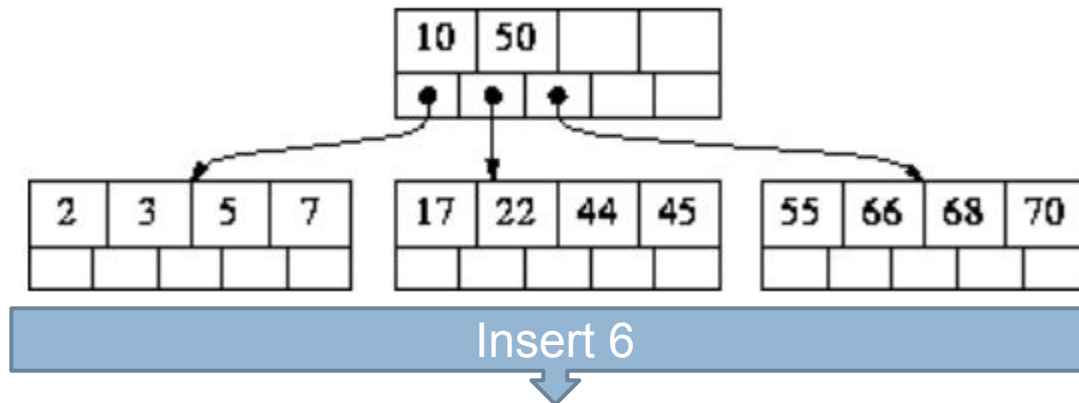


Add it to the middle leaf. No overflow, so we're done.

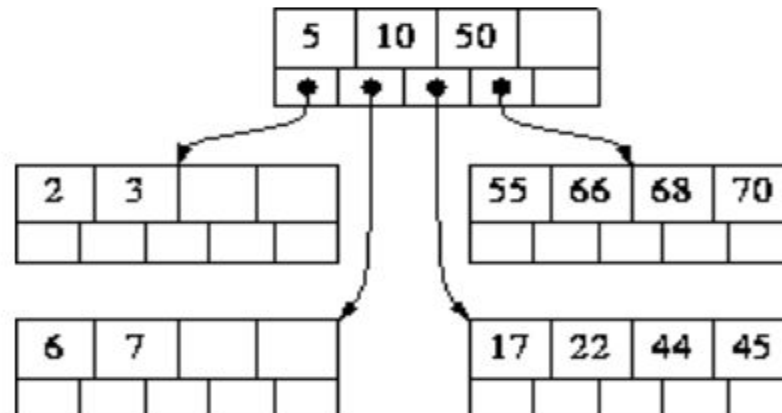
B-Tree Insertion Example



98



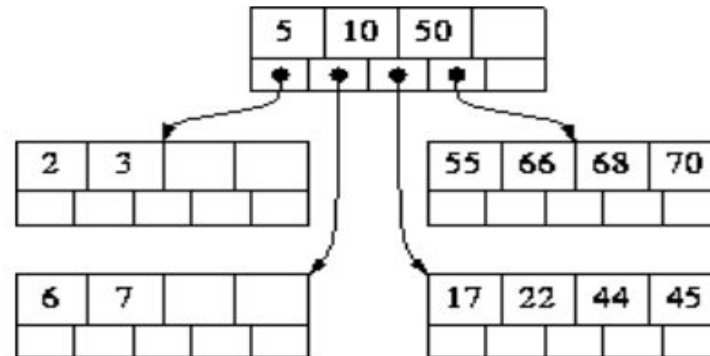
Add it to the leftmost leaf. That overflows, so we split it: Left = [2 3], Middle = 5, Right = [6 7]
Left and Right become nodes; Middle is added to the node above with Left and Right as its children. The node above (the root in this small example) does not overflow, so we are done.



B-Tree Insertion Example

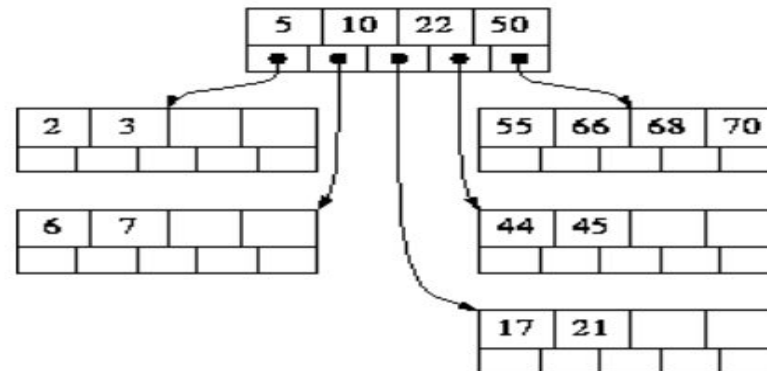


99



Insert 21

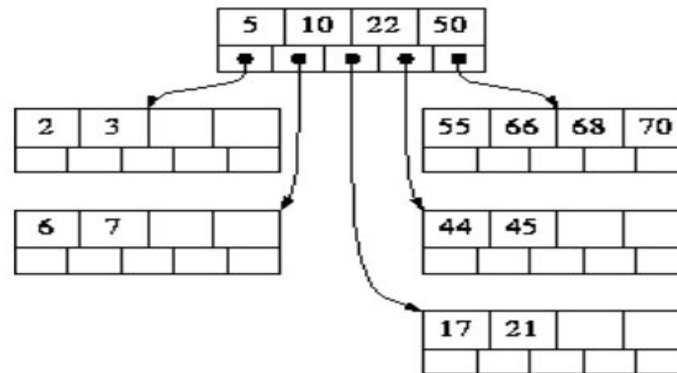
Add it to the middle leaf. That overflows, so we split it: Left = [17 21], Middle = 22, Right = [44 45]
Left and Right become nodes; Middle is added to the node above with Left and Right as its children. The node above (the root node in this small example) does not overflow, so we are done.



B-Tree Insertion Example

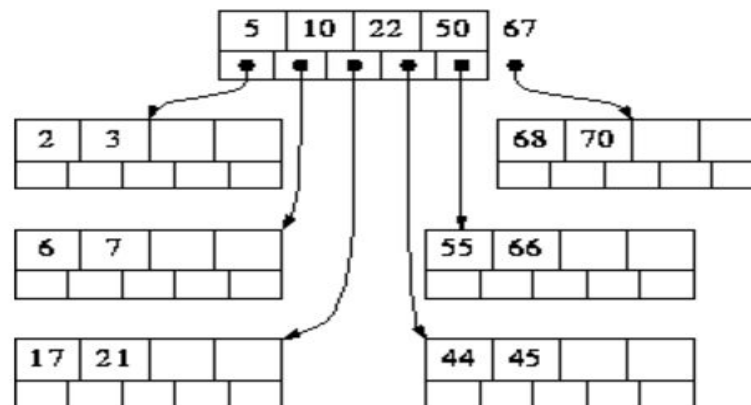


100



Insert 67

Add it to the rightmost tree. That overflows, so we split it: Left = [55 66], Middle = 67, Right = [68 70]
Left and Right become nodes; Middle is added to the node above with Left and Right as its children.



B-Tree Insertion Example

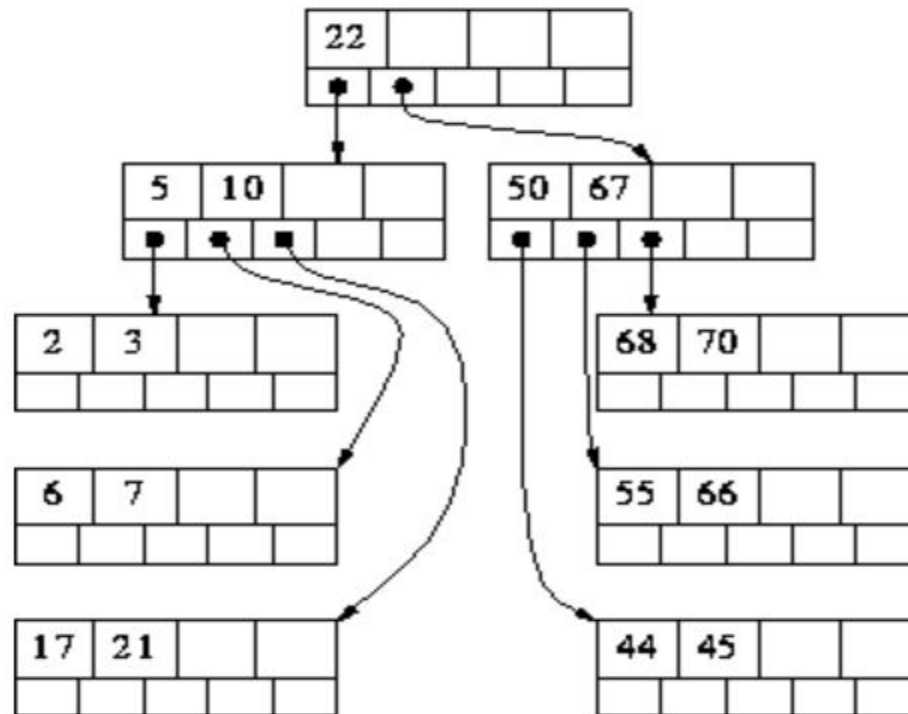


101

But now the node above does overflow. So it is split in exactly the same manner: so we split it:

Left = [5 10], Middle = 22, Right = [50 67]

Left and Right become nodes; , the children of Middle. If this were not the root, Middle would be added to the node above and the process repeated. If there is no node above, as in this example, a new root is created with Middle as its only value.



B-Tree Deletion



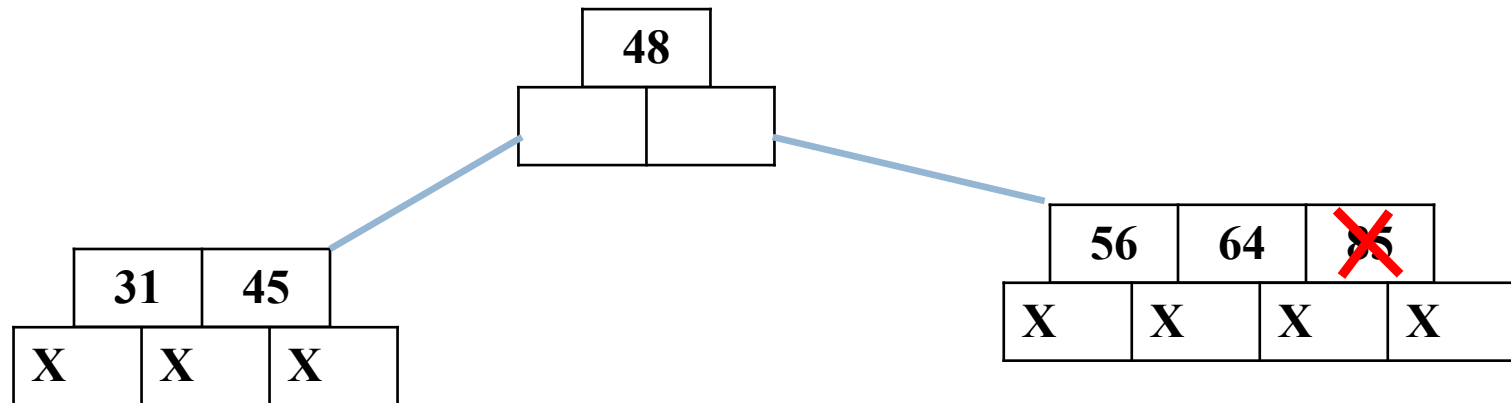
102

Case #	Instance	Action
1	Delete a leaf key with no underflow	Delete
2	Delete a non-leaf key with no underflow	Delete and promote
3	Delete a leaf key with underflow and rich sibling	Delete and borrow
4	Delete a leaf key with underflow and poor sibling	Merge

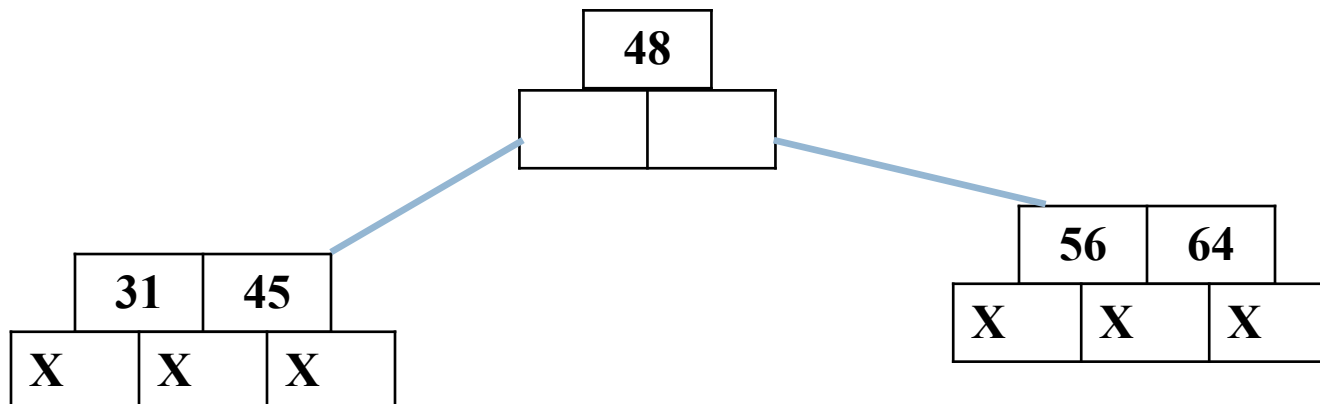
Deletion of leaf key with no overflow



103



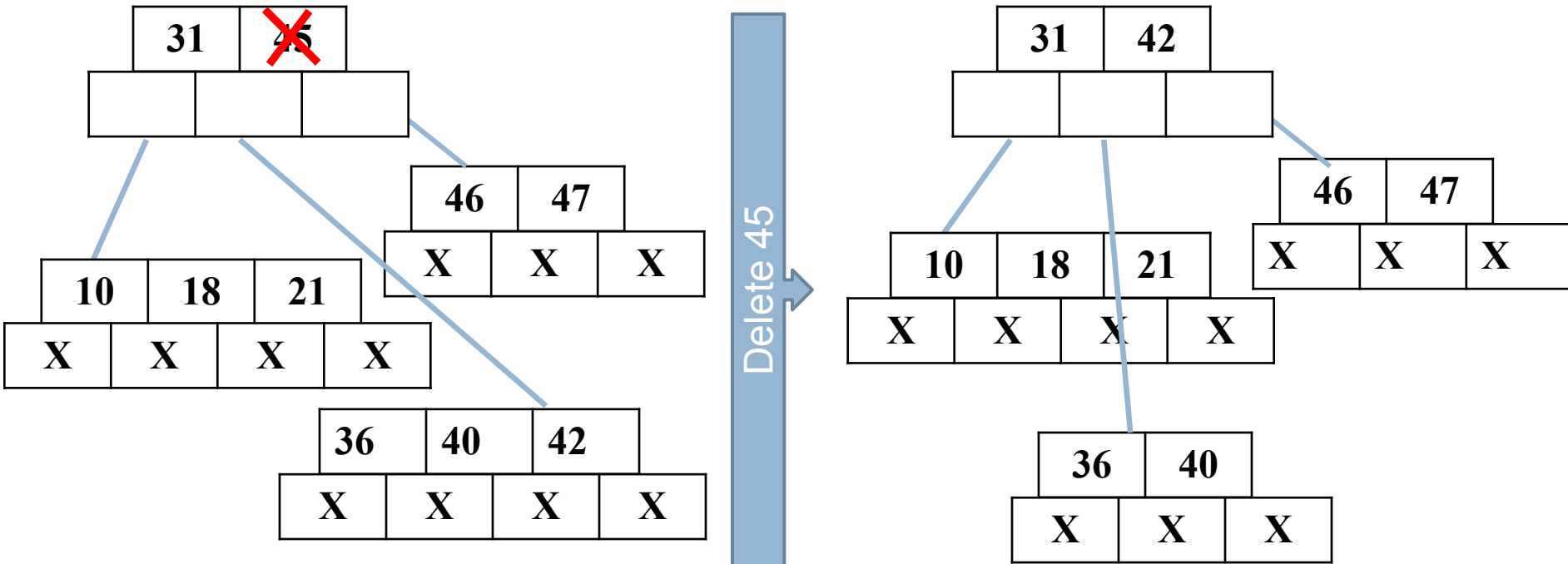
Delete 85



Deletion of non-leaf key with no overflow



104



Delete and Promote

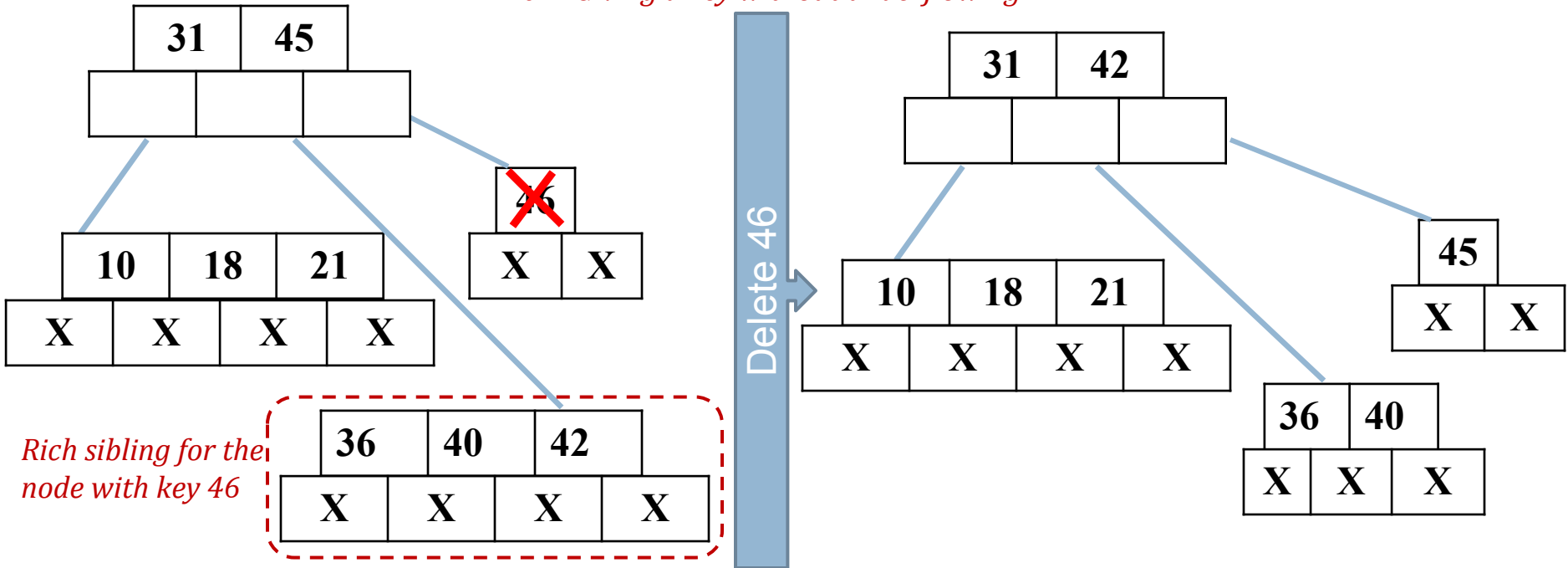
- ❑ Pick the largest key i.e. 42 from the left subtree of the key 45
- ❑ Delete the key 45 from the root node
- ❑ Promote 42 to the parent node of 45

Deletion of leaf key with overflow and rich sibling



105

Rich- Giving a key without underflowing



Rich sibling for the node with key 46

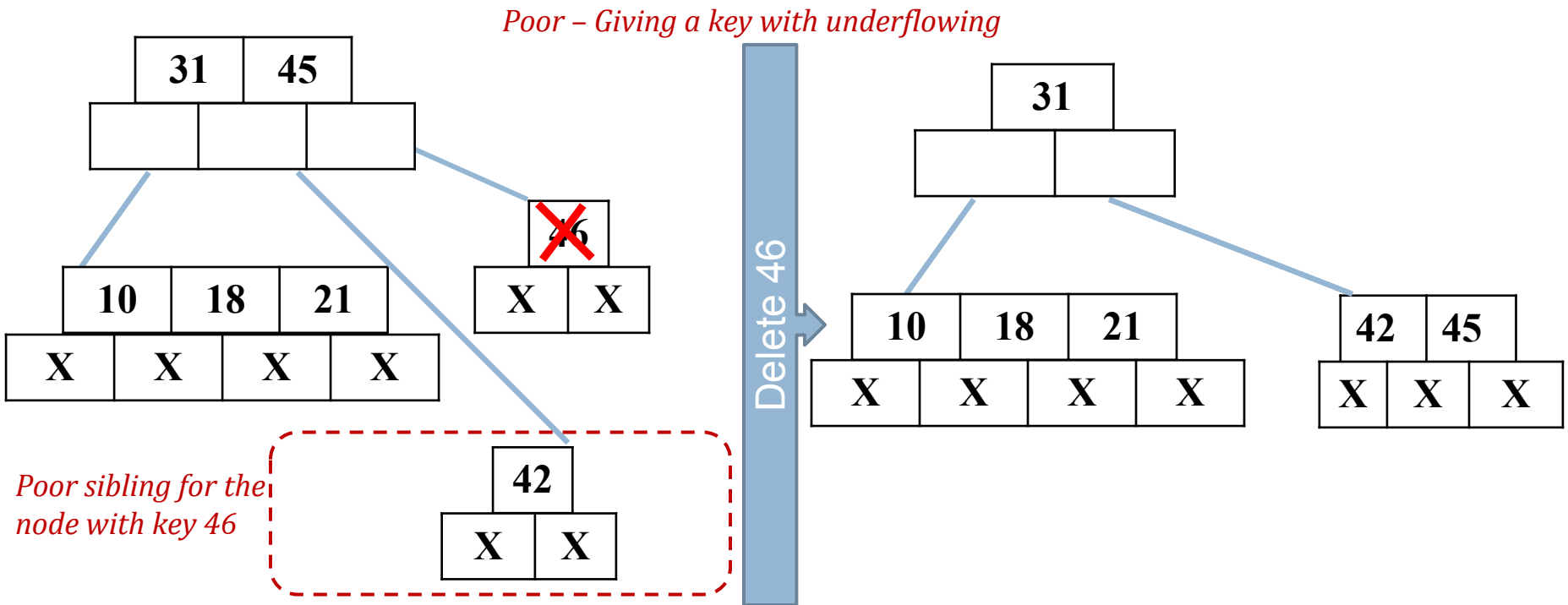
Delete and Borrow

- ❑ **Delete** key 46
- ❑ **Borrow** the largest key i.e. **42** from the rich sibling through parent and replace the appropriate keys.

Deletion of leaf key with overflow and poor sibling



106



Delete and Merge

- ❑ **Delete** key 46
- ❑ **Merge** by pulling the key from parent with the poor sibling

B-Tree C Implementation



107



B-Tree

B+ Tree



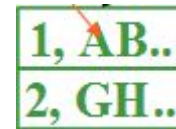
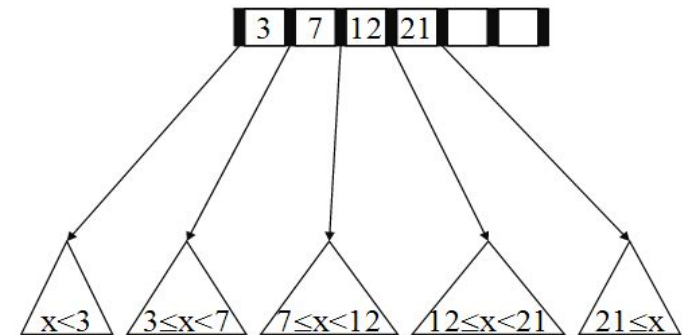
108

In a B+ tree, the internal nodes have no data – only the leaves do!

- ❑ Each internal node still has up-to M-1 keys
- ❑ Subtree between two keys x and y contain leaves with values v such that $x \leq v < y$
- ❑ Leaf nodes have up-to L (i.e. data items in leaf) sorted keys
- ❑ From the root to the leaf all paths should be of same length

Properties:

- ❑ Root (Special Case)
 - ❑ Has between 2 and M children
- ❑ Internal Nodes
 - ❑ Store up-to M-1 keys
 - ❑ Have between $\lceil M/2 \rceil$ and M children
- ❑ Leaf Nodes
 - ❑ Where data is stored
 - ❑ All at the same depth
 - ❑ Contain between $\lceil L/2 \rceil$ and L data items



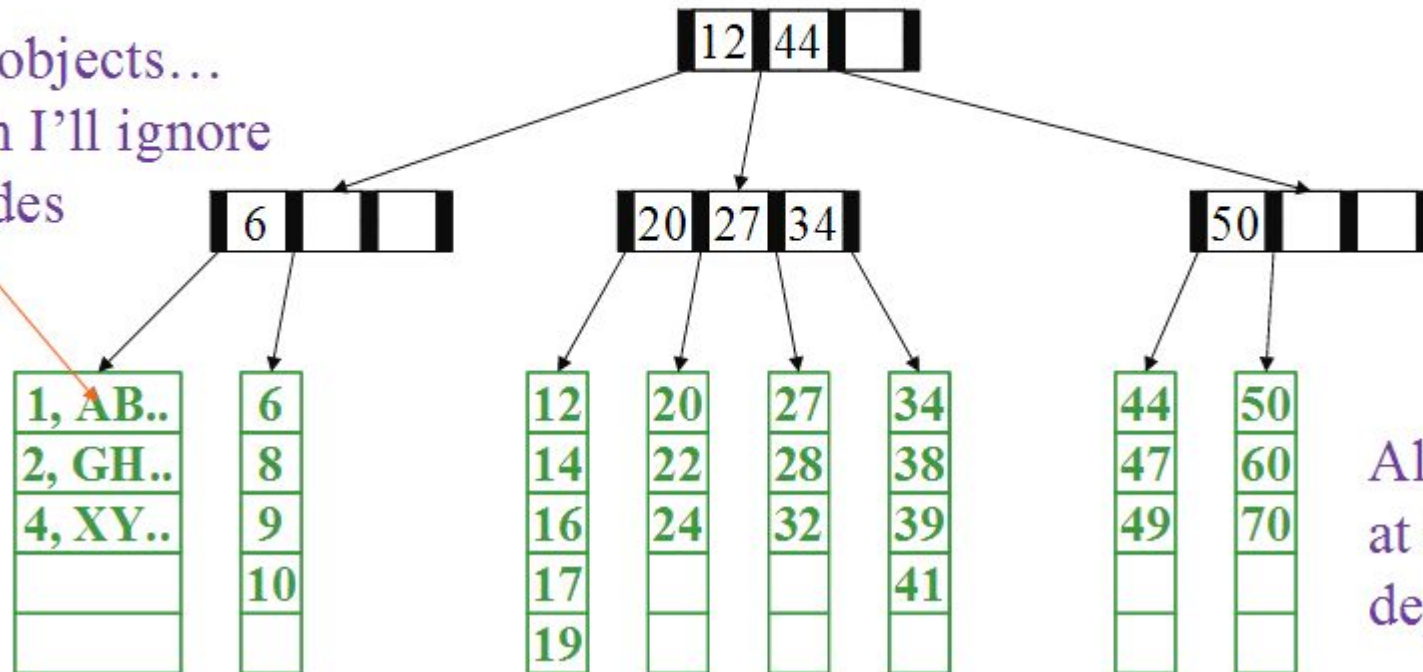
Red arrow represents data elements
Green rectangle represents the node with key and data elements

B+ Tree Example

109

B+ Tree with $M = 4$ (# pointers in internal node)
and $L = 5$ (# data items in leaf)

Data objects...
which I'll ignore
in slides



All leaves
at the same
depth

Assignments



110

- ☐ A binary tree has 9 nodes. The inorder and preorder traversal of T yields the following sequence of node:
Inorder: E, A, C, K, F, H, D, B, G and Preorder: F, A, E, K, C, D, H, G, B
Draw the tree T
- ☐ Suppose T is a binary tree. Write a recursive & non-recursive algorithm to find the number of leaf nodes, number of nodes and depth in T. Depth of T is 1 more than the maximum depths of the depths of the left and right subtrees of T.
- ☐ Suppose T is a binary tree. Write a recursive & non-recursive algorithm to find the height of T.
- ☐ Insert the following keys in the order to form the binary search tree
J, R, D, G, T, E, M, H, P, A, F, Q
- ☐ Insert the following keys in the order to form the binary search tree
50, 33, 44, 22, 77, 35, 60, 40
- ☐ Suppose a binary tree T is in memory. Write the pseudo code to delete all terminals or leaf nodes in T

Assignments cont...



111

- ☐ Write a C function to copy the binary tree T to T'
- ☐ Suppose ROOTA points to a binary tree T1. Write the pseudo code which makes a copy T2 of the tree T1 using ROOTB as pointer
- ☐ Write algorithm to insert and delete the nodes in B tree.
- ☐ Write a program to check whether the binary tree is a binary search tree
- ☐ Write the non-recursive function for inorder traversal, preorder traversal, and postorder traversal of a binary tree.
- ☐ Write a function to display all the paths from root to leaf nodes in a binary tree.
- ☐ Write a programme to insert a node into BST both in recursive and non-recursive manner.
- ☐ Write a programme to display the nodes of a tree in level wise (first we need to display 0th level node, then 1th level nodes and so on).
- ☐ Write a program to check whether a binary tree is an AVL tree.

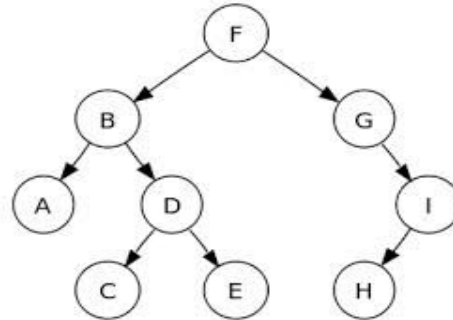
**THANK
YOU!**

Home Work



113

- Find out the inorder, preorder and postorder traversal of the following tree.



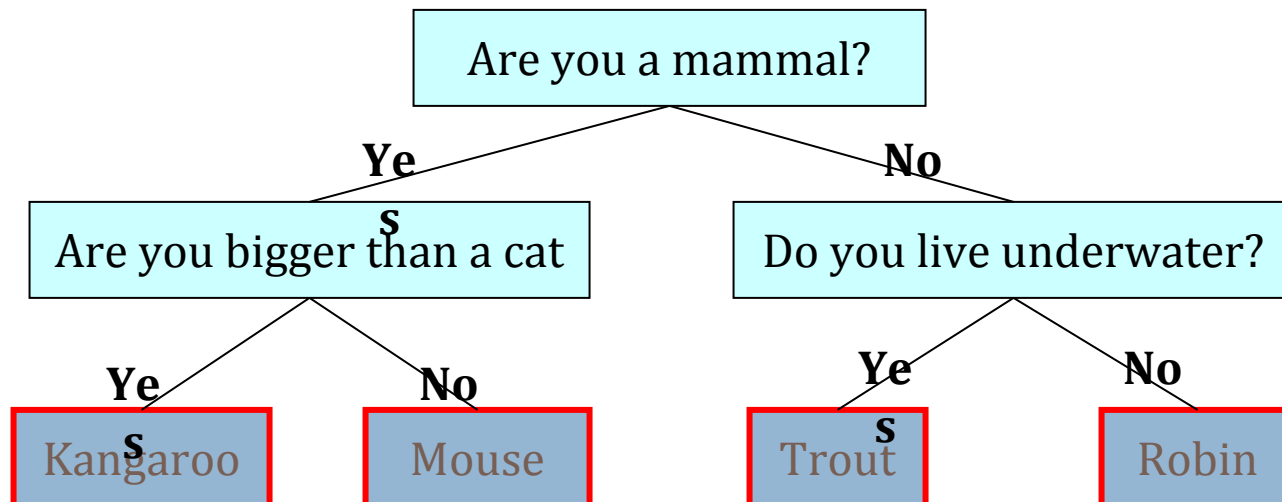
- Draw the internal memory representation using sequential and linked representation
- The inorder and postorder traversals of a binary tree T yield the following sequence of nodes: Inorder : ABCDEFGHI and Postorder: ACEDBHIGF. Find out its Pre-order traversal sequence of the binary tree drawn.
- Design a linked data structure to represent the node of an m-way search tree. Write functions SEARCH_MWAY, INSERT_MWAY, DELETE_MWAY to perform search, insert and delete operations on an m-way search tree.
- Implement the function DELETE_AVL which delete a given element ITEM from the AVL Search tree T.

Home Work cont...



114

- ❑ Draw a complete binary tree with exactly six nodes. Put a different value in each node. Then draw an array with six components and show where each of the six node values would be placed in the array (using the usual array representation of a complete binary tree).
- ❑ Draw a binary taxonomy tree that can be used for these four animals: Rabbit, Horse, Whale, Snake. A binary taxonomy tree is also called as binary decision tree that allows to going down the tree with simple test.



Home Work cont...



115

- ☐ Draw the binary tree for expression: $A * B - (C + D) * (P / Q)$
- ☐ For the following sequence, draw a B-tree of order 3 by inserting the data 91, 24, 6, 7, 11, 8, 21, 4, 5, 16, 19, 20, 76
- ☐ How many different trees are possible with n nodes?
- ☐ Draw a full binary tree with at least 6 nodes.
- ☐ Suppose that a binary search tree contains the number 42 at a node with two children. Write two or three clear sentences to describe the process required to delete the 42 from the tree.
- ☐ Suppose that we want to create a binary search tree where each node contains information of some data type. What additional factor is required for the data type?
- ☐ The maximum number of nodes on level i of a binary tree is ?
- ☐ The maximum number of nodes in a binary tree of depth d is ?
- ☐ Write a C function swapTree that takes a binary tree root node and swaps the left and right children of every node.

Home Work cont...



116

- ☐ Write a function that traverses a threaded binary tree in postorder. What are the time and space requirement of your method?
- ☐ Write a function that traverses a threaded binary tree in preorder. What are the time and space requirement of your method?
- ☐ Write pseudo code to delete the node with key k from a binary search tree.
- ☐ Assume that a binary search tree is represented as a threaded binary search tree. Write functions to search, insert and delete
- ☐ Write an algorithm to construct the binary tree with given preorder and inorder.
- ☐ Write the pseudo code to construct the binary tree with given postorder and inorder.
- ☐ Prove that every binary tree is uniquely by its preorder and inorder sequences.
- ☐ Please provide constructive suggestions to this course in terms of lectures, home works, or any other aspects

Supplementary Reading



117

- ❑ <http://www.geeksforgeeks.org/binary-tree-data-structure/>
- ❑ <http://www.geeksforgeeks.org/binary-search-tree-data-structure/>
- ❑ https://www.tutorialspoint.com/data_structures_algorithms/tree_data_structure.htm
- ❑ http://btechsmartclass.com/DS/U3_T1.html
- ❑ http://btechsmartclass.com/DS/U3_T3.html
- ❑ <https://www.hackerrank.com/domains/data-structures/trees>
- ❑ <https://www.youtube.com/watch?v=qH6yxkw0u78>
- ❑ <http://nptel.ac.in/courses/106102064/6>



Describe Trees with an example.

Tree is a structure that is similar to linked list. A tree will have two nodes that point to the left part of the tree and the right part of the tree. These two nodes must be of the similar type.

The following code snippet describes the declaration of trees. The advantage of trees is that the data is placed in nodes in sorted order.

```
struct TreeNode
{
    int item; // The data in this node.
    TreeNode *left; // Pointer to the left subtree.
    TreeNode *right; // Pointer to the right subtree.
}
```

What is binary tree?

A binary tree is a tree structure, in which each node has only two child nodes. The first node is known as root node. The parent has two nodes namely left child and right child.

Uses of binary tree:

- ☐ To create sorting routine.
- ☐ Persisting data items for the purpose of fast lookup later.
- ☐ For inserting a new item faster

What is a B tree?

A B-tree of order m (the maximum number of children for each node) is a tree which satisfies the following properties :

1. Every node has $\leq m$ children.
2. Every node (except root and leaves) has $\geq m/2$ children.
3. The root has at least 2 children.
4. All leaves appear in the same level, and carry no information.
5. A non-leaf node with k children contains $k - 1$ keys

What are threaded binary trees?

In a threaded binary tree, if a node 'A' has a right child 'B' then B's left pointer must be either a child, or a thread back to A.

In the case of a left child, that left child must also have a left child or a thread back to A, and so we can follow B's left children until we find a thread, pointing back to A.

This data structure is useful when stack memory is less and using this tree the traversal around the tree becomes faster.

What are Infix, prefix, Postfix notations?

- ❑ **Infix notation:** $X + Y$ – Operators are written in-between their operands. This is the usual way we write expressions. An expression such as $A * (B + C) / D$
- ❑ **Postfix notation** (also known as “Reverse Polish notation”): $X Y +$ Operators are written after their operands. The infix expression given above is equivalent to $A B C + * D /$
- ❑ **Prefix notation** (also known as “Polish notation”): $+ X Y$ Operators are written before their operands. The expressions given above are equivalent to $/ * A + B C D$

What is a B+ tree?

It is a dynamic, multilevel index, with maximum and minimum bounds on the number of keys in each index segment. all records are stored at the lowest level of the tree; only keys are stored in interior blocks.