Book

A Simplified Approach to Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

Topics to be discussed

- Algorithm to implement threaded binary tree
- Program to implement Huffman Algorithm
- To find the position of a given element and its parent in a Binary Search Tree.
- To insert a given element into a Binary Search Tree.
- To delete a given element from the Binary Search Tree.
- To find the smallest element in a Binary Search Tree.
- To find the largest element in a Binary Search Tree.
- Expression Trees
- Reconstruction of Binary Trees

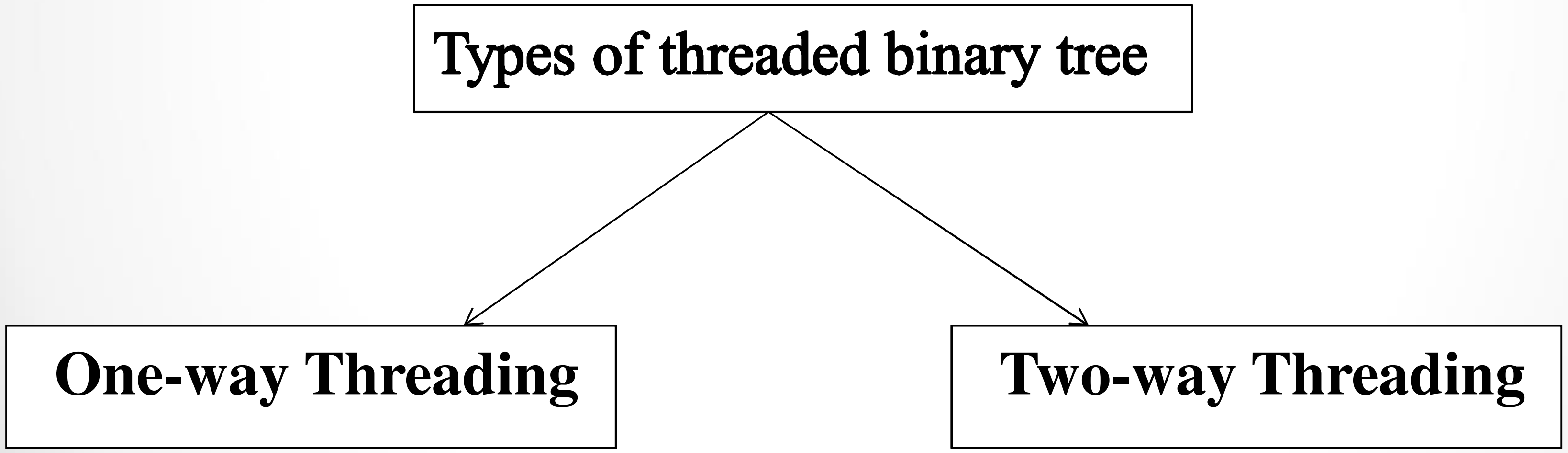
Threaded Binary Tree

A binary tree with thread(s) is known as threaded binary tree. In a threaded binary tree a know may contain pointer to its child node or thread to some higher node in the node. The higher node to which the thread points in the tree is determined according to the tree traversal technique used (i.e. preorder, inorder, postorder).

There are three methods to apply thread binary tree. Each method corresponds to a particular type of tree traversal:
Preorder threading: on this, the threads are applied to the higher node containing the preorder traversal of the tree.
Inorder threading: the threads are applied considering the inorder traversal.
Postorder threading: the thread are applied consider the postorder traversal.

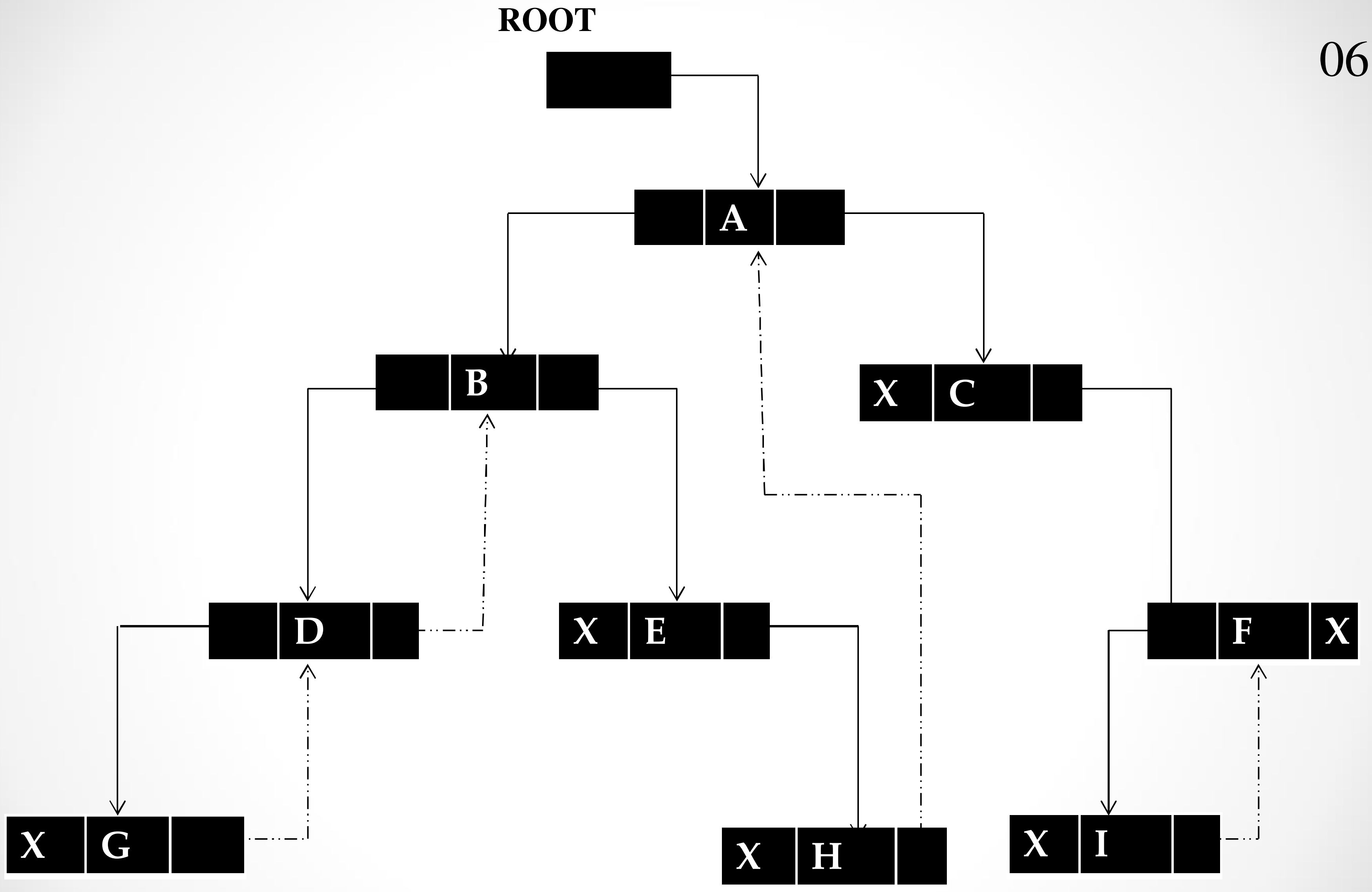
Types of threaded binary tree

Depending on the type of threading, there are two types of threaded binary tree:

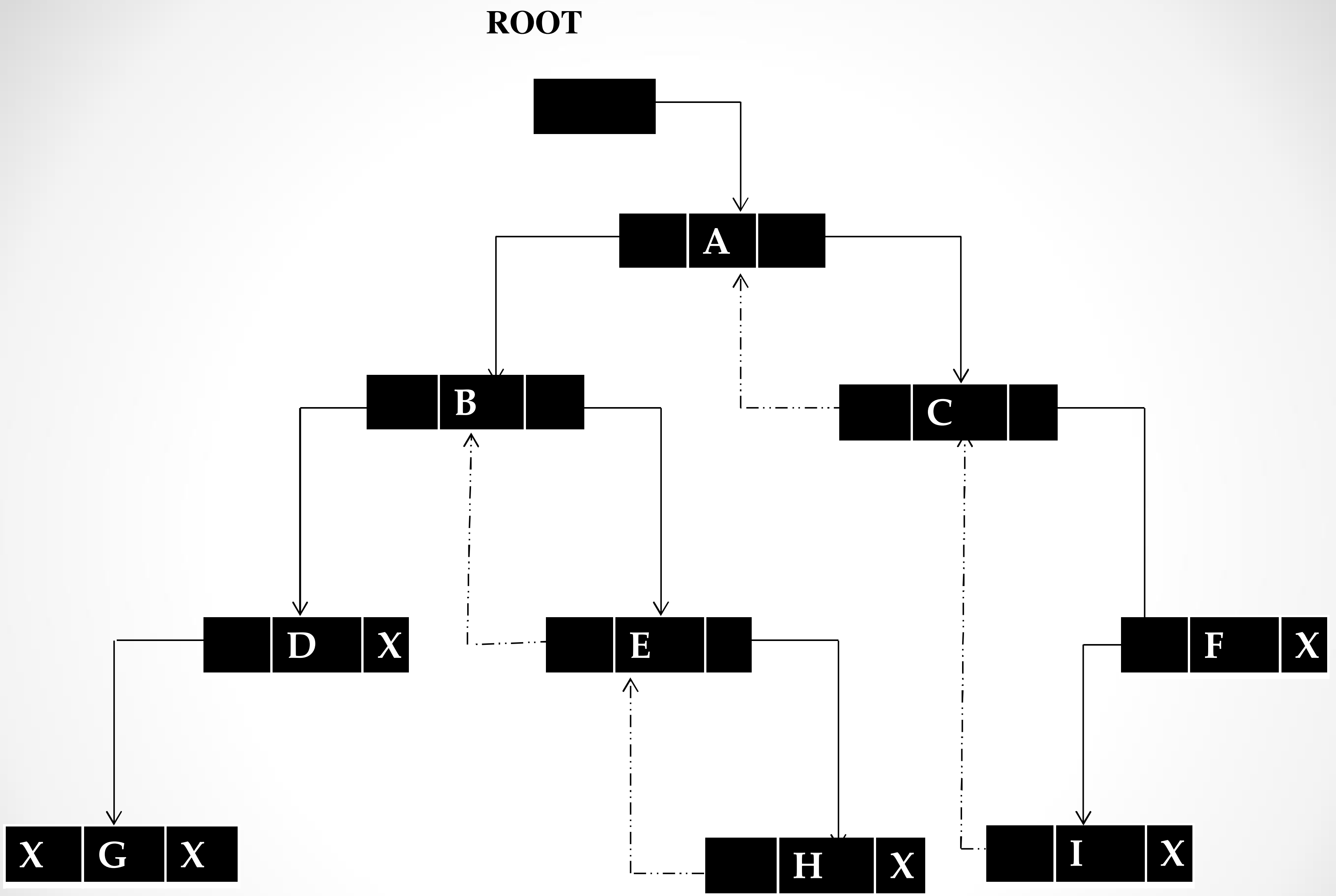


One-way Threading

In this, a thread appears either in the right link field or in the left link field of the node. Is the thread appears in the right link filed of the node then points to the next node in the inorder traversal of the tree i.e. it points to the in order successor of the node. Such tree is known as right threaded binary tree.
If the thread appears in the left link field of the node. It points to the preceding node in the inorder traversal of the tree. Such a tree is known as left threaded binary tree. As, the threading is done according to the inorder traversal, hence the tree is referred as in-threaded binary tree.



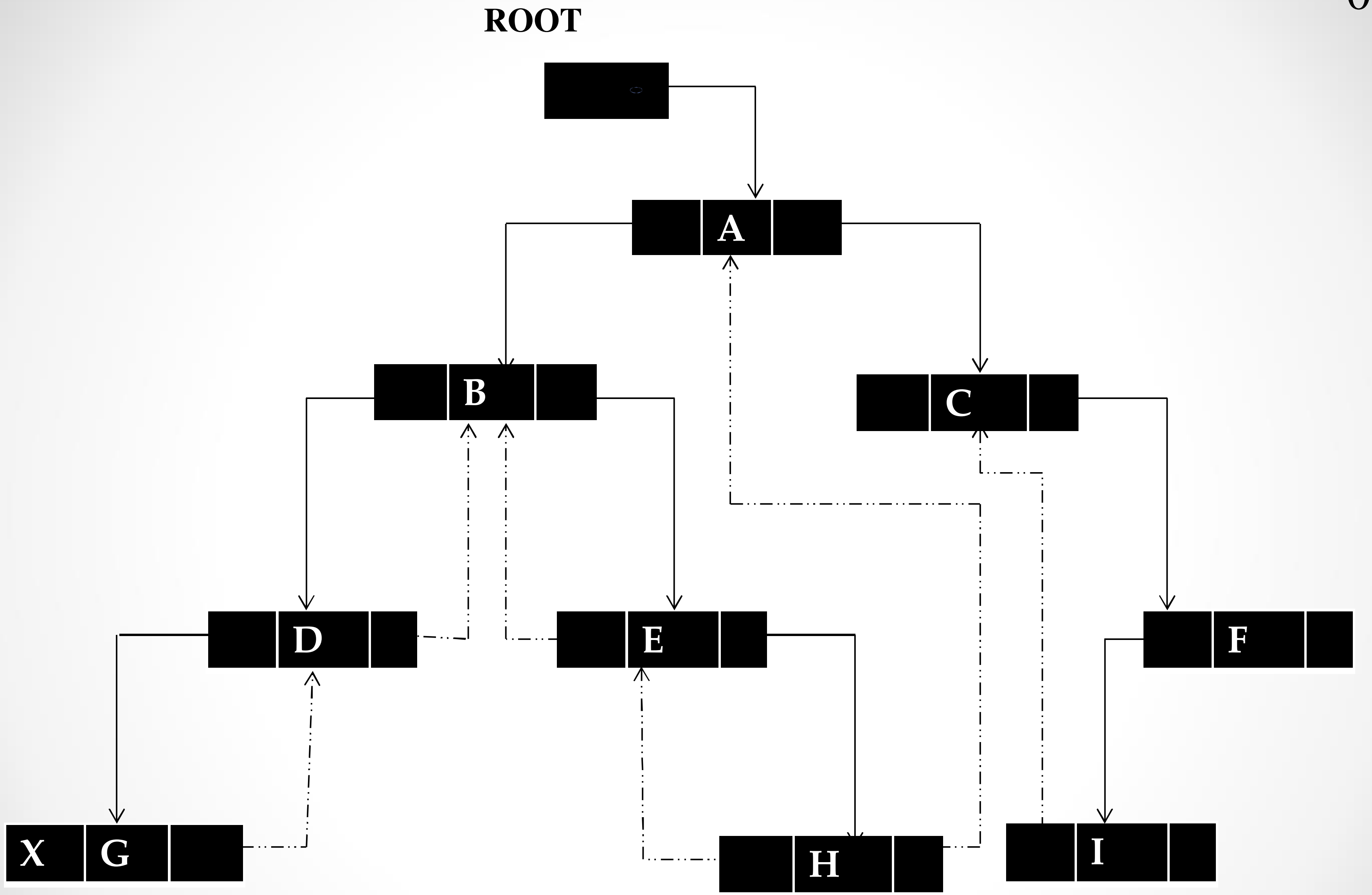
Right threaded binary tree



Left threaded binary tree

Two-way Threading

To obtain a two-way threaded binary tree the right link field containing null is replaced by a thread pointing to the inorder successor of the node and the left link filed containing null is replaced b a thread pointing to the inorder predecessor of the node.
Observe that all the left pointer containing null have been replaced by threads accept the left pointer of G. which is the first node in the inorder traversal of tree T.
Similarly, all the right pointer containing null have been replaced by threads accept the right pointer of F. which is the last node in the inorder traversal of tree T.



Two way threaded binary tree

Huffman Algorithm

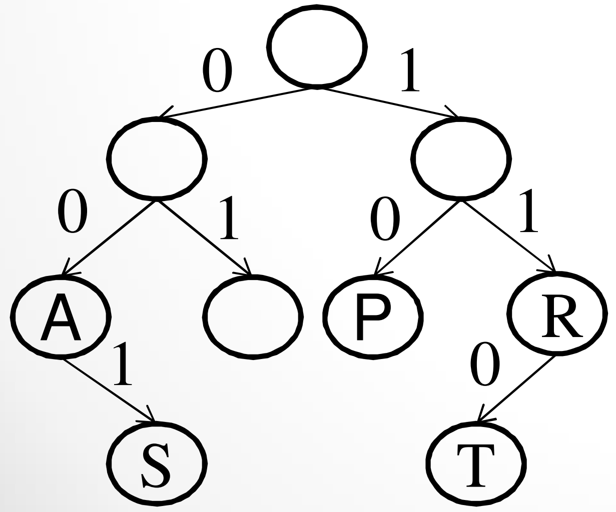
Motivation

- Huffman coding is a method for the construction of minimum redundancy codes.
- Applicable to many forms of data transmission. Example: text files.

Coding : Problem Definition

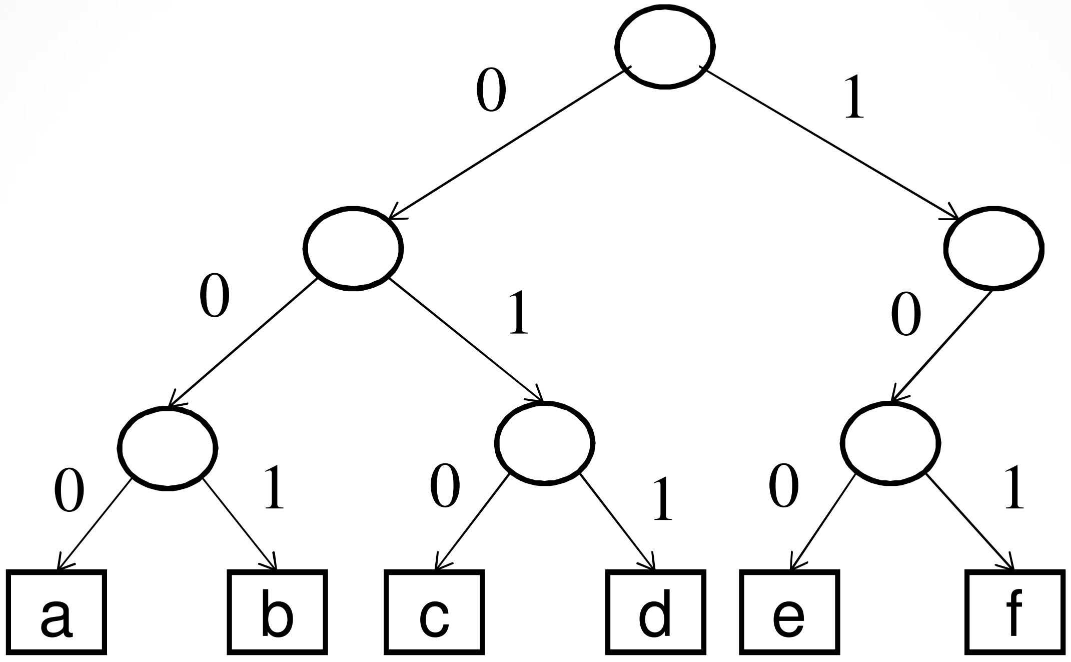
- This problem is that of finding the minimum length bit string which can be used to encode a string of symbols.
- An encoding for each character is found by following the tree from the route to the character in the leaf: the encoding is the string of symbols on each branch followed.

  Example :



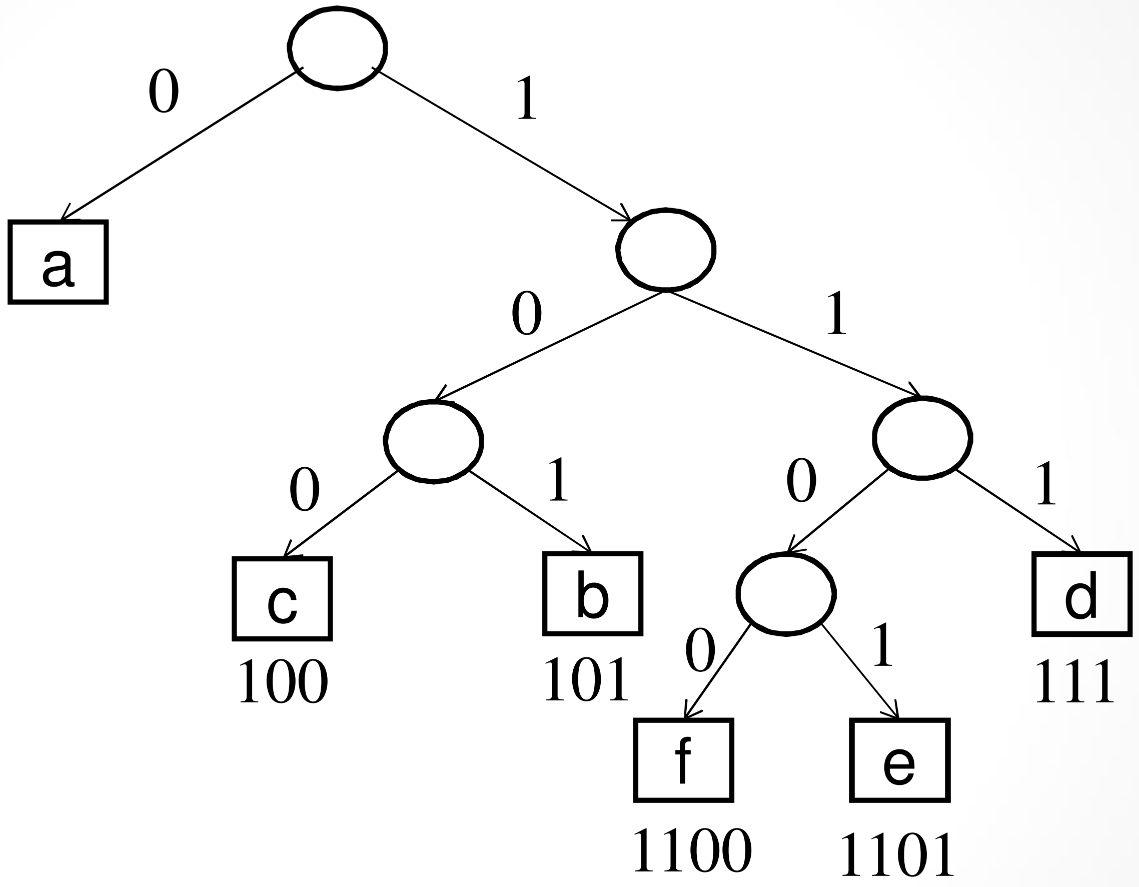
String	Encoding
AST	00 001 110
PRA	10 11 00
PST	10 001 110
RAS	11 00 001

Example: fixed-length prefix code (1)



Message: 000.001.000.011.100.101 abadef

Example: variable-length prefix code (2)



Message: 0.101.0.111.1101.1100

Algorithm to make Huffman Tree

- Scan the message to be encoded, and count the frequency of every character.
- Create a single node tree for each character and its frequency and place into a priority queue (heap) with the lowest frequency at the root.
- Until the forest contains only 1 tree do the following:
  - Remove the two nodes with the minimum frequencies from the priority queue (we now have the 2 least frequent nodes in the tree).
  - Make these 2 nodes children of a new combined node with frequency equal to the sum of the 2 nodes frequencies.
  - Insert this new node into the priority queue.

Huffman Code : Example(1)

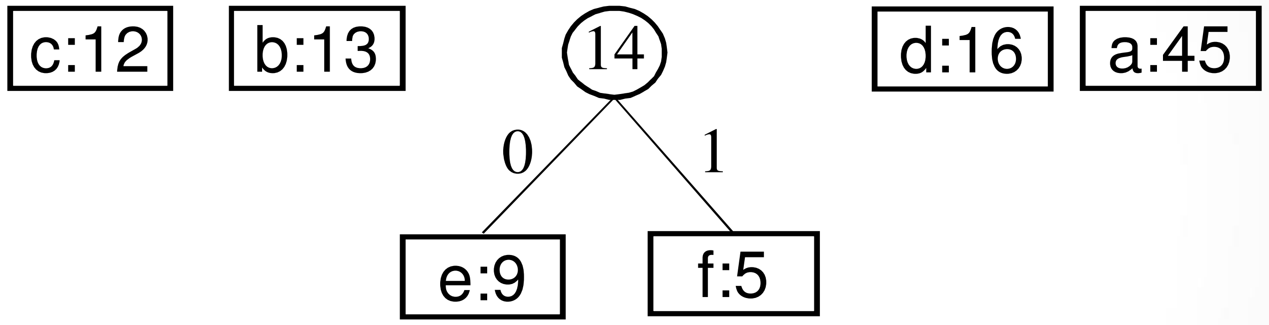
Start :

Put the symbols along with their frequency in increasing / decreasing order as shown below :-

$\boxed{f:5}$ $\boxed{e:9}$ $\boxed{c:12}$ $\boxed{b:13}$ $\boxed{d:16}$ $\boxed{a:45}$

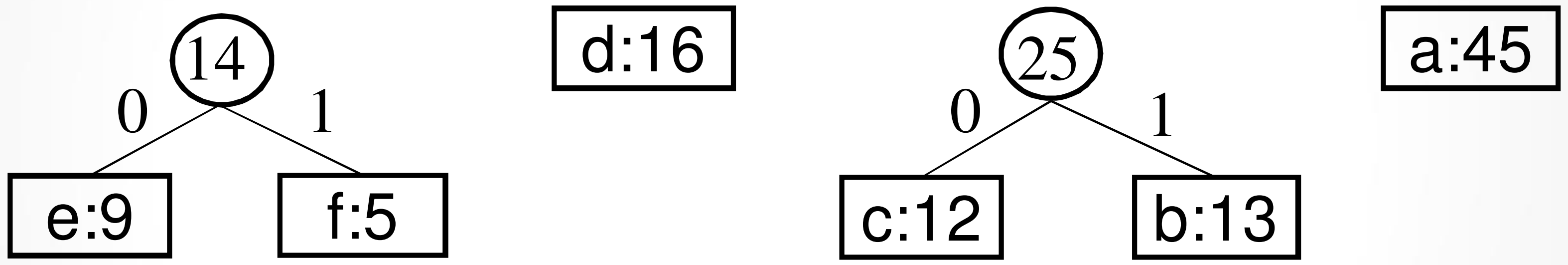
Huffman Code : Example(2)

Step 1 : Add the frequencies e & f and rerank everything so that items are still in sorted order.



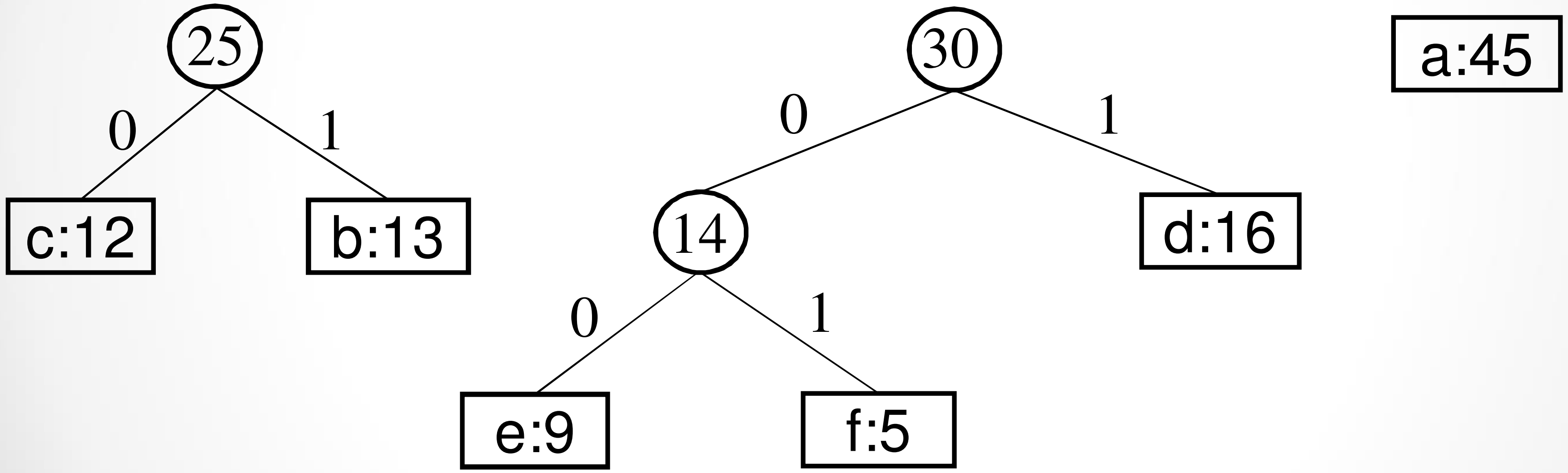
Huffman Code : Example(3)

Step 2 : Add the frequencies c & b and rerank everything to have it in a sorted order.



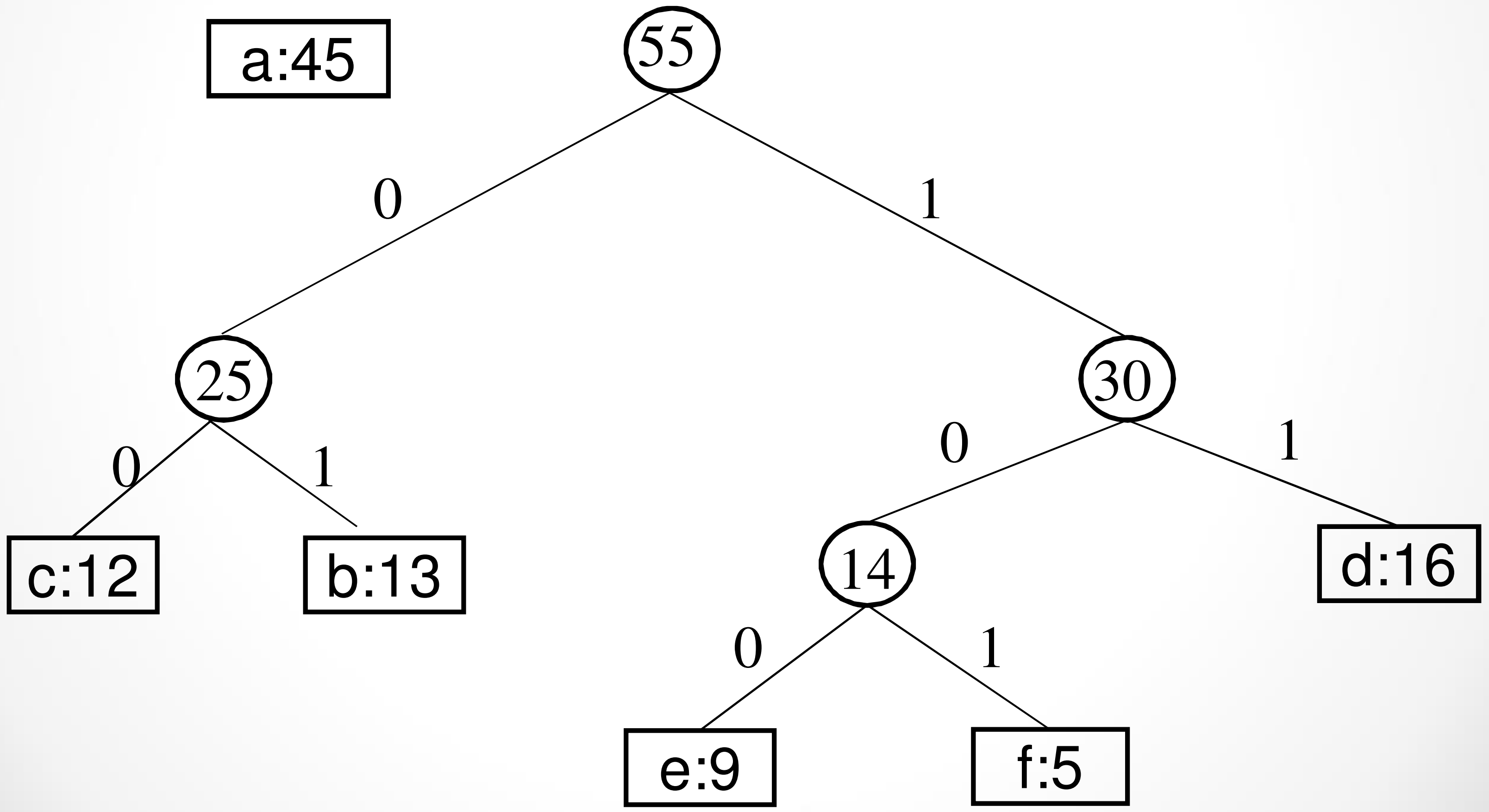
Huffman Code : Example(4)

Step 3: Add the tree of e & f with d and rearrange them in a sorted order.



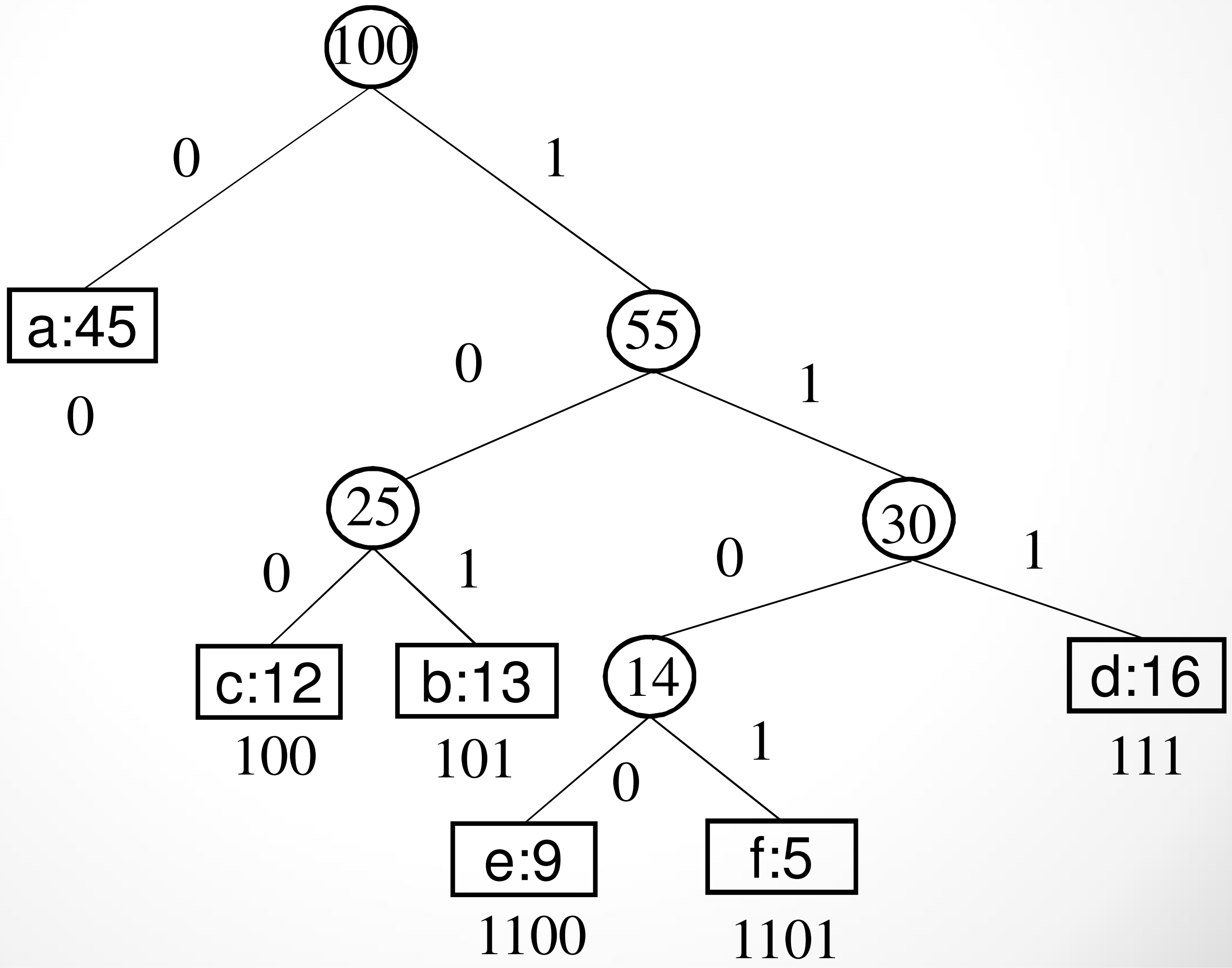
Huffman Code : Example(5)

Step 4 : Now, add the tree of c & b with the tree of e, f & d. Re-rank them in a sorted order.



Huffman Code : Example(6)

Step 5 : At last, combine the tree formed with a. And hence the Huffman tree is formed.



Huffman Code : Example(7)

Result : Codes for the variables :-

a :	0
b :	100
c :	101
d :	111
e :	1100
f :	1101

Hence, no code is the prefix of another code.

Binary Search Trees

- Operations on binary search tree:
  - Searching a particular element in BST
  - Insertion of an element
  - Deletion of an element
  - Finding the smallest element
  - Finding the largest element

Binary Search Tree

A **BST** is a Binary Tree in which

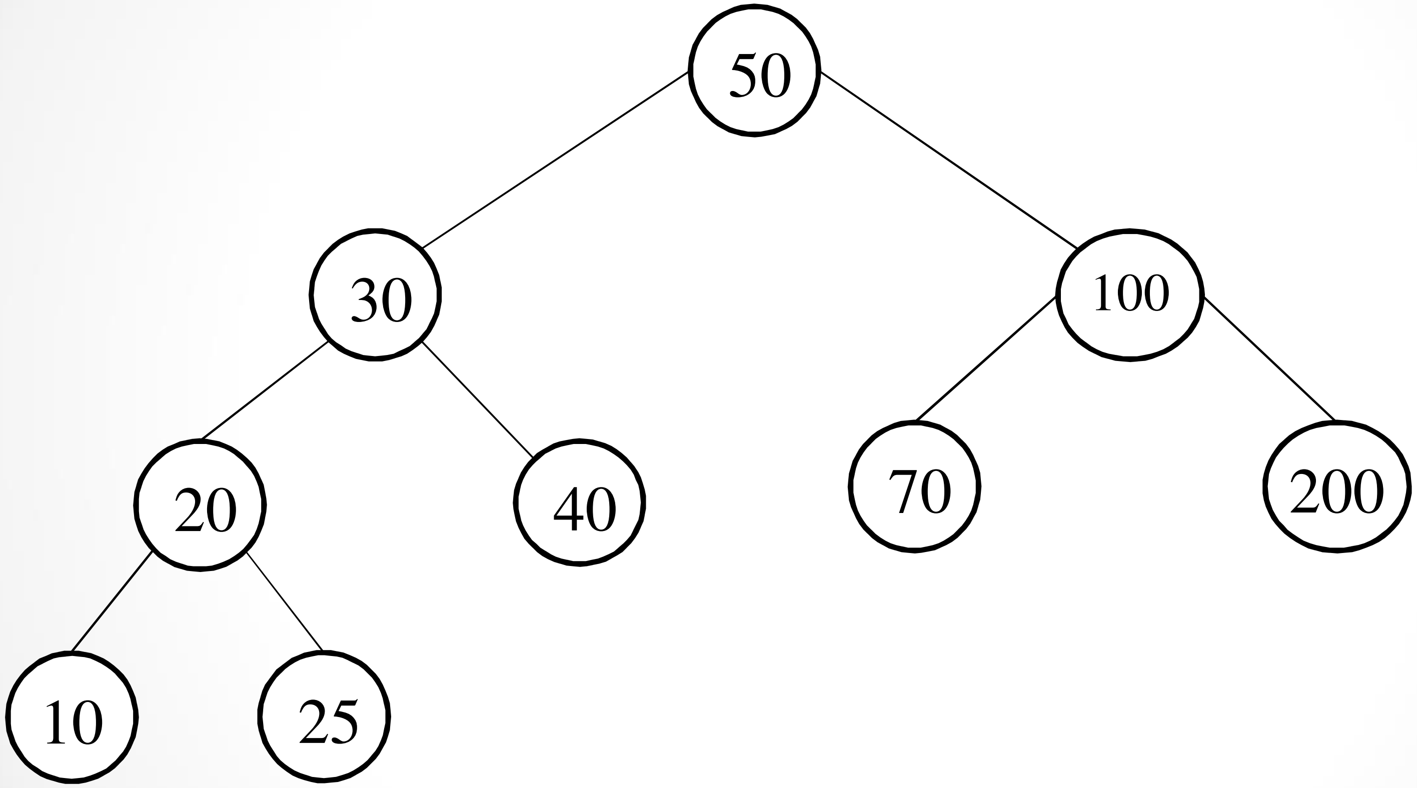
•data is managed in a logical way.

•node containing the data has the following constraints:

- Each data element in the left subtree is less than its root element.
- Each data element in the right subtree is greater than or equal to its root element.
- Both the left and right subtree of the root will be again a BST.

Binary Search Tree(continued)

The binary tree shown is binary search tree.



When this BST is traversed in **in-order** manner, it produces a sorted list of data elements. The list is given as:

10 20 25 30 40 50 70 100 200

Operations on Binary Search Tree

Various operations that can be performed on binary search tree are:

- Searching a particular key element.
- Inserting an element.
- Deletion of an element.
- Finding the smallest element.
- Finding the largest element.

Searching of a particular key value in BST

- Compare the element to be searched with the value of root.
- If both are same , stop the search.
- If its value is less than root, follow the left subtree and
- If its value is larger than the root, follow the right subtree.
- Repeat this procedure recursively until we find the desired element.
- If not found then conclude that element is not present in the binary search tree.

Algorithm to search a particular value in BST

```
BSTSearch(Root,Item,Position,Parent)
Step1:  If Root=Null Then
                set Position = null
                 set Parent =  null
                 Return
           [End If]
Step 2:  Pointer=Root And Pointer P = Null
Step 3:  Repeat Step 4 While Pointer != Null
Step 4:  If Item = Pointer→Info Then
               set Position = Pointer
                 set Parent = PointerP
                 Return
Step 5:  Else If Item<Pointer→Info Then
```

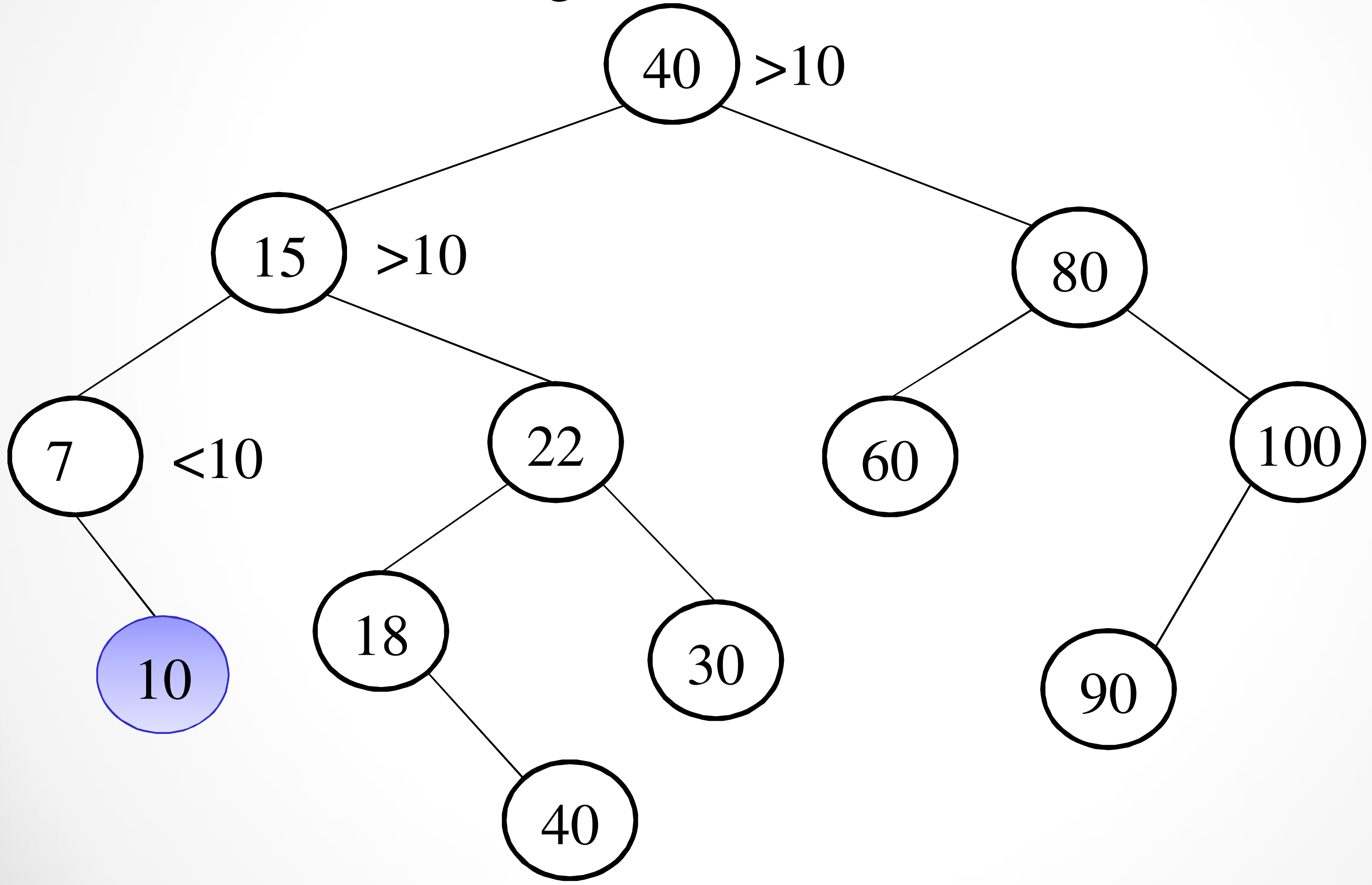
•   .

Algorithm (continued)

```
            set PointerP = Pointer
            set Pointer = Pointer→Left
     Else
            set PointerP = Pointer
            set Pointer = Pointer→Right
      [End If]
  [End Loop]
Step 5: Set Position =Null And Parent = Null
Step 6: Return
```

Insertion of a particular key value in BST

Consider the following BST and insert the element 10 into it.



Algorithm to insert a given element

```
Step 1: If Free = Null Then
    Print: "No space is available for the node to insert"
    Exit
Else
     Allocate memory to new node for insertion
      (New = Free And Free = Free→ Right)
      Set New→Info= Item
     Set New→left= Null And New→Right= Null
      [End if]
Step 2: If Root= Null Then Set Root= New
      Exit
[End If]
Step 3: If Item >= Root → Info Then
      Set Pointer=Root→ Right
        Set PionterP= Root
```

Algorithm to insert a given element(continued)

```
   Else
  Set Pointer=Root→ Left
     Set PionterP= Root
  [End If]
Step 4: Repeat step 5 while Pointer !=Null
Step 5:   If Item >=Pointer→ Info Then
             Set PionterP=Pointer
              Set Pointer=Pointer→ Right
  Else
             Set PionterP=Pointer
              Set Pointer=Pointer→ Left
    [End If]
[End Loop]
```

```
Step 6: If Item< PointerP→Info Then
                Set PointerP→Left=New
            Else
            Set PointerP→Right= New
    [End If]
Step 7: Exit
```

Complexity of Insertion Process

- Complexity of Insertion Process = O(h) where h is the height of BST.

- If BST is complete binary tree or almost complete binary tree then,
- Complexity of the Insertion Process = $O(\log_2 n)$.

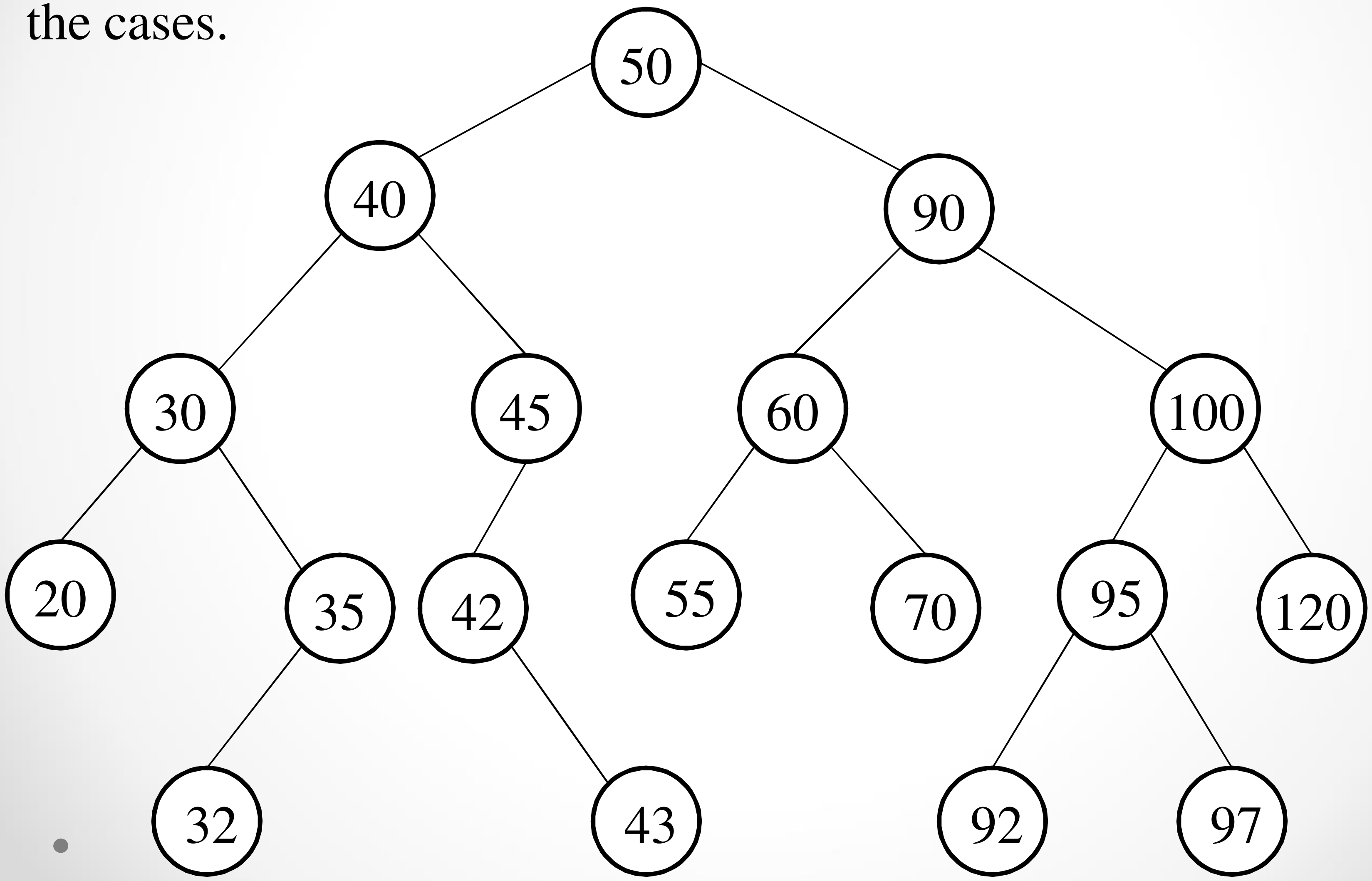
Deletion of a node from binary search tree

- Firstly locate the node containing the element to be deleted and also locate its parent node.

- There are three cases of deletion :
  Case 1. where the node to be deleted is a leaf node.
  Case 2.where it has one child.
  Case3. where it has 2 children.

Deletion of a node from binary search tree

Consider the following BST as an example to understand all the cases.



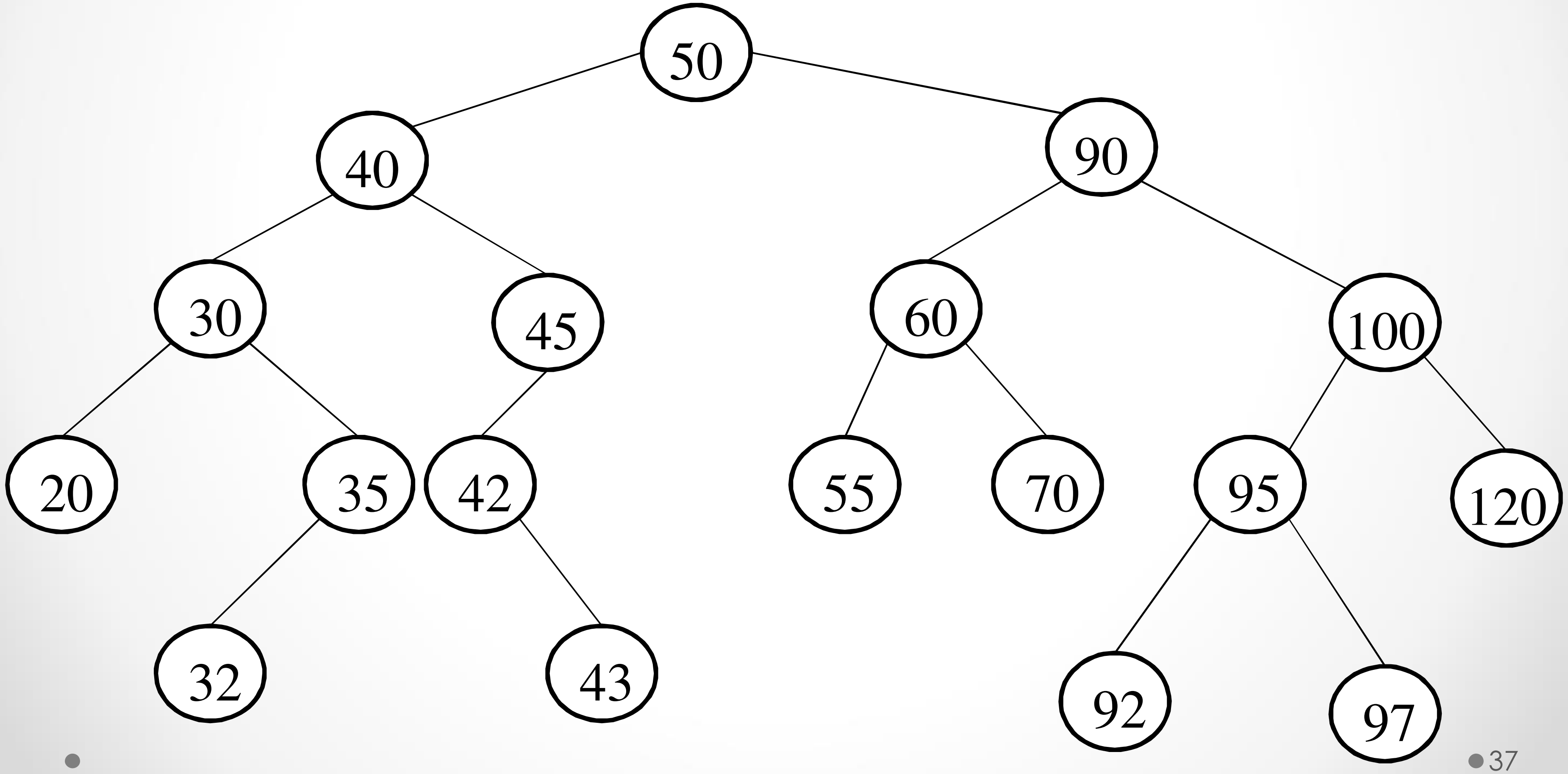
Deletion of a node from binary search tree

CASE 1: Node to be deleted is a leaf node.

Here , in the binary search tree shown, the leaf nodes are **20, 32, 43, 55, 70, 92, 97** and **120,**
And it is very simple to delete them by changing the respective pointer in their parent node to Null.

Deletion of a node from binary search tree

CASE 2: It has one child.
For example in this BST Delete the element 45 which has one child. After deletion tree is shown as:



Deletion of a node from binary search tree

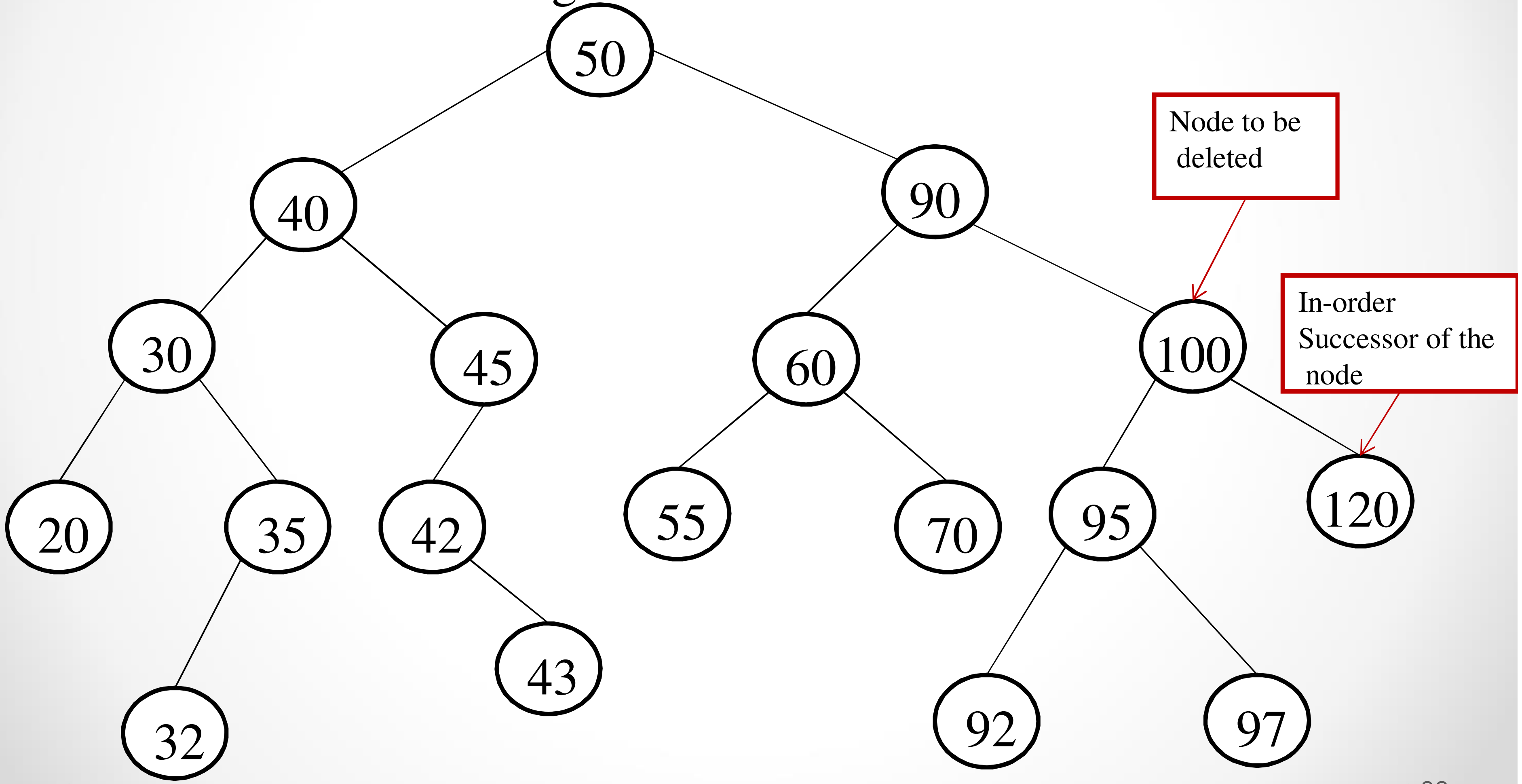
CASE 3: It has two children.

Here we have to find the in-order successor of the node to be deleted.
Because In-Order successor of any node having two children can have one or zero child and cannot have any left subtree.

Deletion of a node from binary search tree

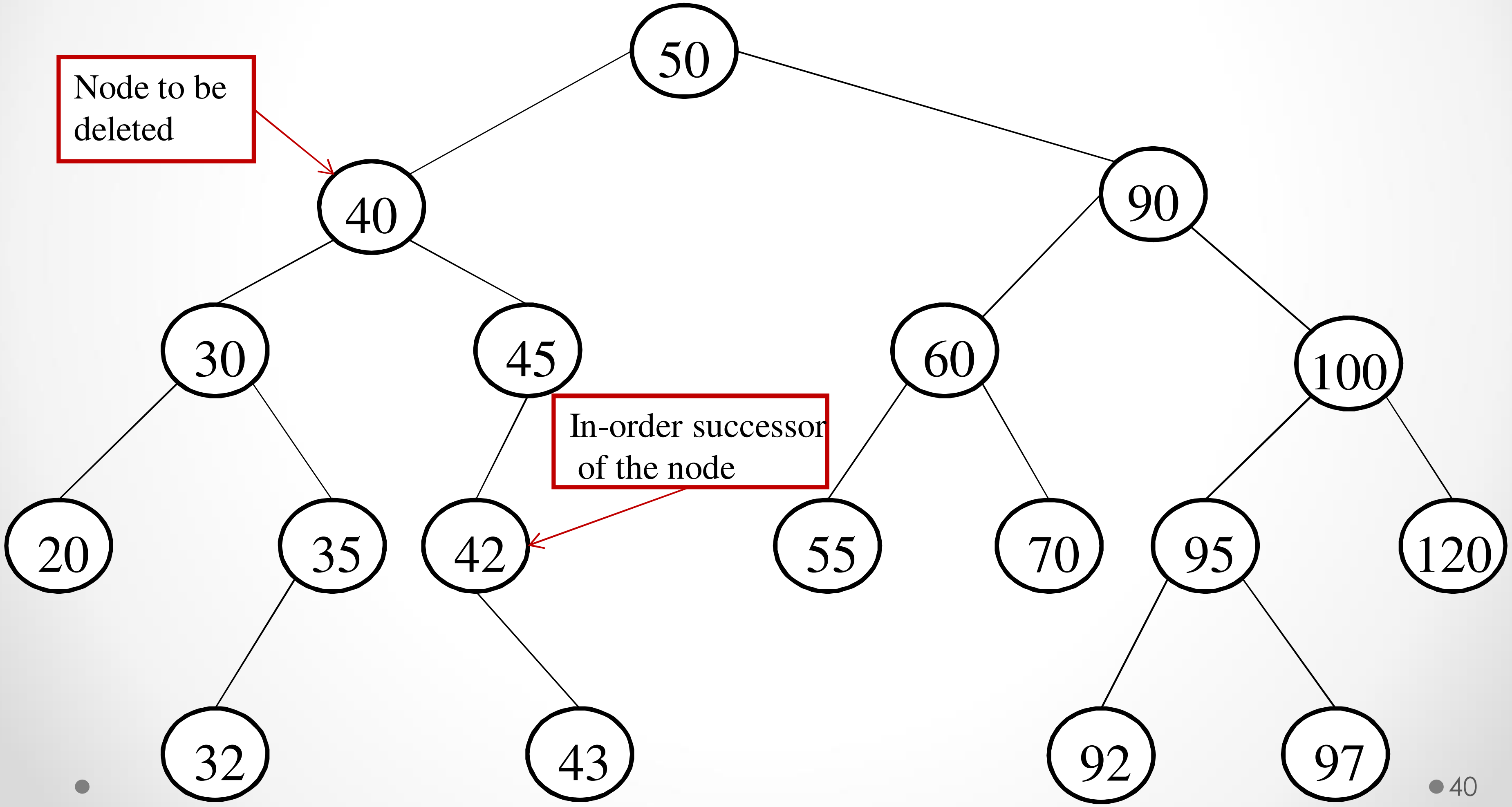
Consider the following example:

Delete the node containing the element 100 which has 2 children.



Deletion of a node from binary search tree

Consider an another example having two children:
Now delete the node containing the element 40.



Algorithm to delete a given element

```
DeleteItem(Root, Item)
 Step1: Call BSTSearch (Root, Item, Position, Parent)
 Step2: If Position=Null Then
                 Print:"Item not found in the tree"
             Exit
         [End if]
 Step 3: If Position →Left != Null And  Position → Right
!=Null
                 Then
              Call Delete2(Root, Position, Parent)
         Else
              Call Delete1(Root, Position ,Parent)
       [End If]
```

Algorithm to delete a given element

```
Step 4: Deallocate memory held by node Position
        (Set Position→Right = Free And Free=Position)
Step 5: Exit
```

BSTSearch() algorithm has already been explained
Refer this sub algorithm from there.

Algorithm to delete a given element

The below Sub-algorithm delete a node having zero or one child from the binary search tree.

```
Delete1( Root, Position, Parent)
Step1: If  Position→Left=Null And Position→Right= Null
            Then
                    Set Temp= Null
             Else If Position →Right!=Null   Then
                   Set Temp = Position→Right
            Else
                 Set Temp =Position→Left
         [End If]
Step 2:  If Parent= Null Then
               Set Root= Temp
            Else If Position= Parent→Left Then
```

Algorithm to delete a given element

```
        Set **Parent→Left=Temp**
      Else
        Set **Parent→Right=Temp**
    [End If]
Step 3:  Return
```

Algorithm to delete a given element

The below sub-algorithm delete a node having two children from the binary search tree.

```
Delete2( Root, Position, Parent)
Step1: Set Pointer = Position →Right And PointerP=Position
Step2 : Repeat while Pointer →Left !=Null
            Set PointerP=Pointer And Pointer= Pointer→Left
          [End Loop]
Step3:   Set Successor =Pointer And PSuccessor=PointerP
Step4:   Call Delete(Root,Successor,PSuccessor)
Step5:   If Parent != Null Then
         If Position =Parent→Then
             Set Parent→Left =Successor
         Else
```

Algorithm to delete a given element

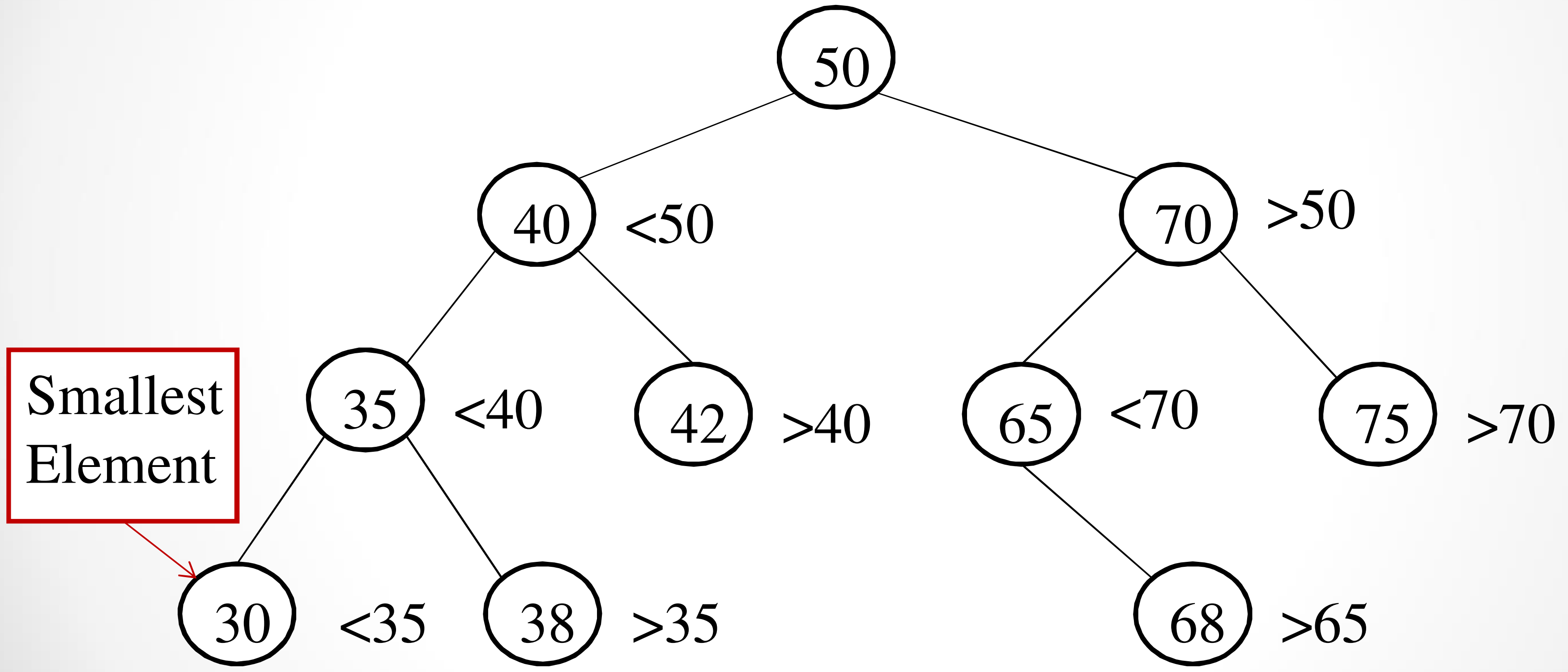
```
Set Parent→Right =Successor [End If]
     Else
           Set Root=Successor
         [End If]
Step6:   Set Successor→Left=Position→Left
Step7:    Set Successor→Right =Position→Right
Step8:      Return
```

Finding the smallest element in BST

- As in BST every left node is smaller than right node in each subtree of BST.
- Therefore to find the **smallest element** in BST we will have to traverse the **left most node of the BST**.

Finding the smallest element in BST

Consider the following BST as an example:



Hence the lefttmost node of the tree contain the smallest element i.e 30

Algorithm to find the smallest element in BST

```
Step1:   If  Root=Null Then
              Print " Tree is Empty"
              Exit
         Else
         Set Pointer= Root
    [end if ]
Step2: Repeat while Pointer → Left= Null
            Set Pointer= Pointer→ Left
        [End Loop]
Step3: Set Min=Pointer→Info
Step4: Print : Min
Step5: Exit
```

Complexity to find the smallest element in BST

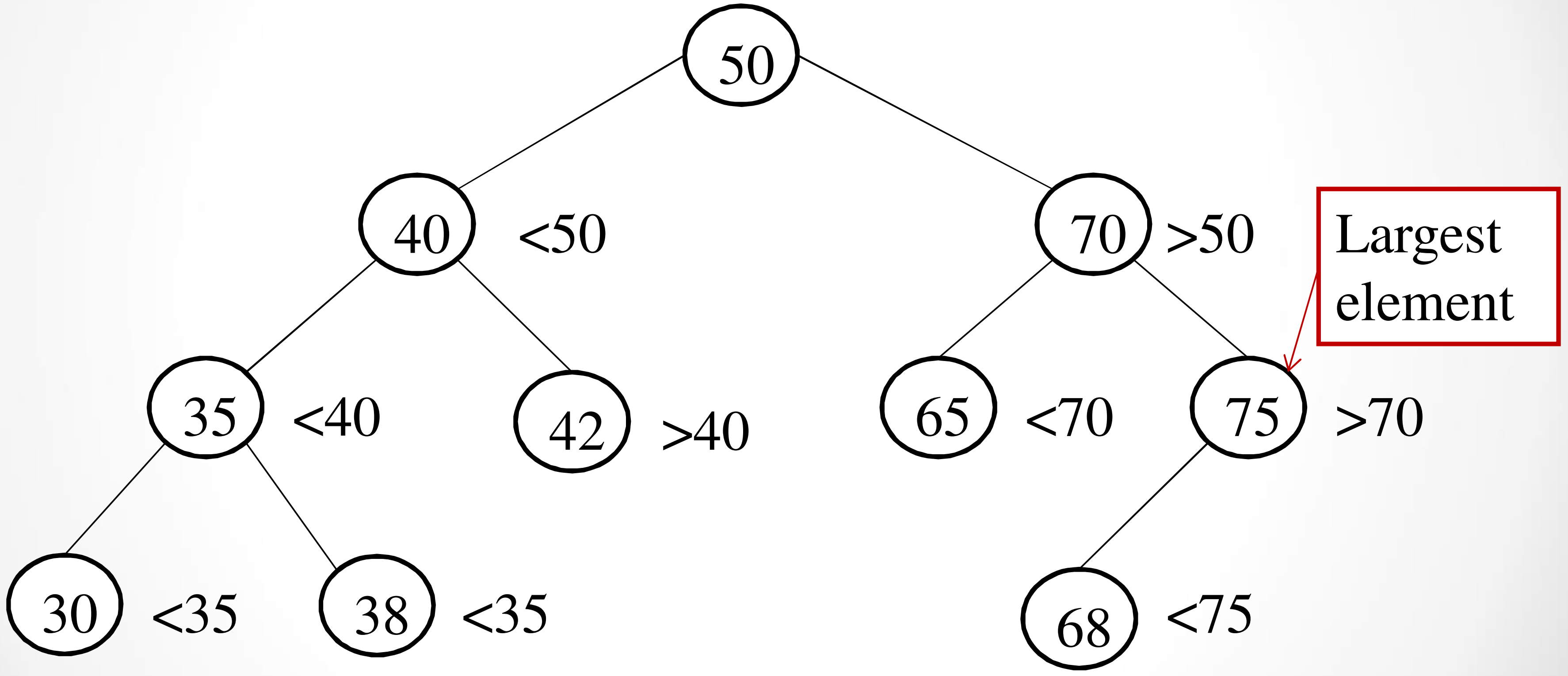
- The complexity of finding the smallest element is dependent upon the height of the binary search tree.
- So, if the height of the left leg of the tree is highest then the worst case complexity will be O(h).
- In case the binary search tree is complete or almost complete binary search tree with n elements, the complexity of finding the smallest element will be $O(\log_2 n)$.

Finding the largest element in BST

- As in binary search tree every right node is smaller than left node in each subtree of BST.
- Therefore to find the **largest** element in BST we will have to traverse the **right most node** of the BST .

Finding the largest element in BST

Consider the following BST as an example:



Hence the rightmost node of the tree contain the largest element i.e 75

Algorithm to find the largest element in BST

```
Step1:  If Root=Null Then
            Print " Tree is Empty"
            Exit
        Else
        Set Pointer= Root
  [end if ]
Step2: Repeat while Pointer → Right= Null
          Set Pointer= Pointer→ Right
      [End Loop]
Step3: Set Max=Pointer→Info
Step4: Print : Max
Step5: Exit
```

Complexity to find the largest element in BST

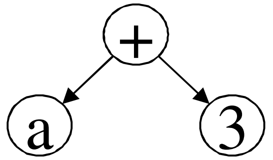
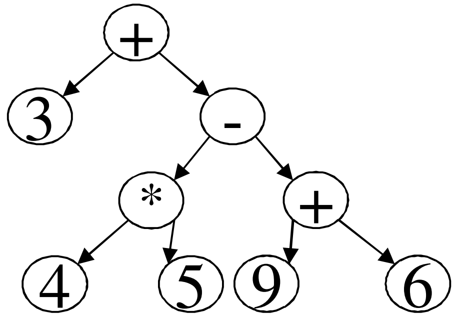
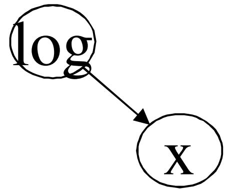
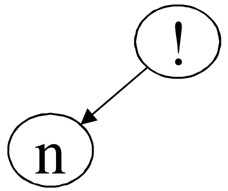
- The complexity of finding the largest element is dependent upon the height of the binary search tree.
- So, if the height of the right leg of the tree is highest then the worst case complexity will be O(h).
- In case the binary search tree is complete or almost complete binary search tree with n elements, the complexity of finding the largest element will be O($\log_2 n$).

Expression Trees

- An expression tree for an arithmetic, relational, or logical expression is a binary tree in which :

- The parentheses in the expression do not appear.
- The leaves are the variables or constants in the expression.
- The non-leaf nodes are the operators in the expression :
  - A node for a binary operator has two non-empty subtrees.
  - A node for a unary operator has one non-empty subtree.

Example of Expression Tree

Expression	Expression Tree	Inorder Traversal Result
(a+3)		a + 3
3+(4*5-(9+6))		3+4*5-9+6
log(x)		log x
n!		n !

Why Expression Trees?

- Expression trees are used to remove ambiguity in expressions.
- Consider the algebraic expression 2 - 3 * 4 + 5.
- Without the use of precedence rules or parentheses, different orders of evaluation are possible :

((2-3)*(4+5)) = -9
((2-(3*4))+5) = -5
(2-((3*4)+5)) = -15
(((2-3)*4)+5) = 1
(2-(3*(4+5))) = -25

- The expression is ambiguous because it uses infix notation : each operator is placed between its operands.

Why Expression trees? (contd.)

- Storing a fully parenthesized expression, such as ((x+2)-(y*(4-z))), is wasteful, since the parentheses in the expression need to be stored to properly evaluate the expression.
- A compiler will read an expression in a language like Java, and transform it into an expression tree.
- Expression trees impose a hierarchy on the operations in the expression. Terms deeper in the tree get evaluated first. This allows the establishment of the correct precedence of operations without using parentheses.
- Expression trees can be very useful for:
    - Evaluation of the expression.
    - Generating correct compiler code to actually compute the expression's value at execution time.
    - Performing symbolic mathematical operations (such as differentiation) on the expression.

Expressing an Expression

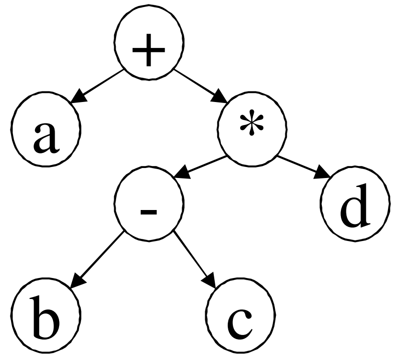
Expressions can be expressed by using three notations. These are :

- Prefix Notation
- Infix Notation
- Postfix Notation

Prefix Notation

- A preorder traversal of an expression tree yields the prefix (or polish) form of the expression.
- In this form, every operator appears before its operand(s).

For Example , Consider the tree :

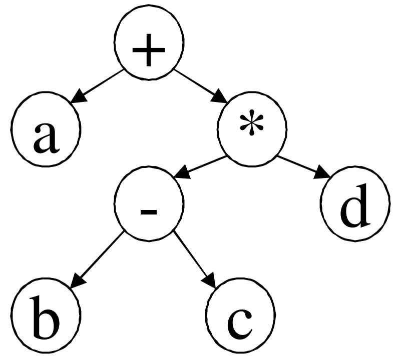


Prefix Notation : + a * - b c d

Infix Notation

- An inorder traversal of an expression tree yields the infix form of the expression.
- In this form, every operator appears between its operand(s).

For Example , Consider the tree :

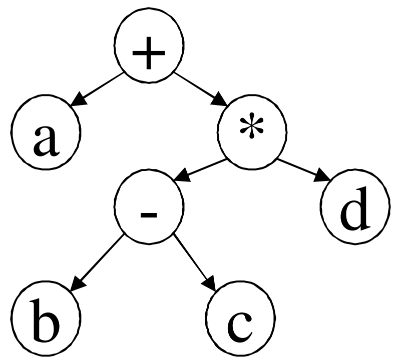


Infix Notation : a + b - c * d

Postfix Notation

- An postorder traversal of an expression tree yields the postfix form of the expression.
- In this form, every operator appears after its operand(s).

For Example , Consider the tree :



Postfix Notation : a b c - d * +

Prefix, Infix, and Postfix Forms (contd.)

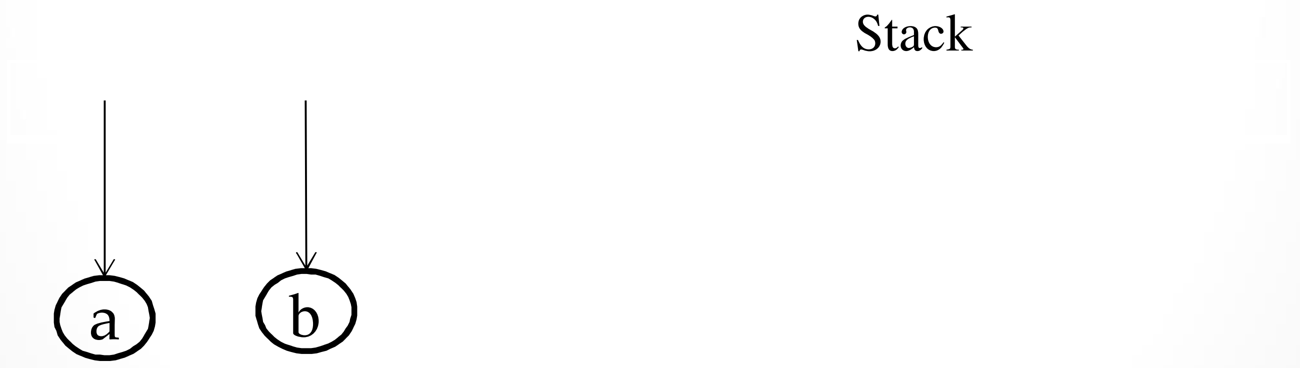
Expression	Prefix forms	Infix forms	Postfix forms
(a + b)	+ a b	a + b	a b +
a - (b * c)	- a * b c	a - b * c	a b c * -
log (x)	log x	log x	x log
n !	! n	n !	n !

Expression Tree Example (1)

Consider the expression (a + b) * c. The postfix expression is:

a b + c *

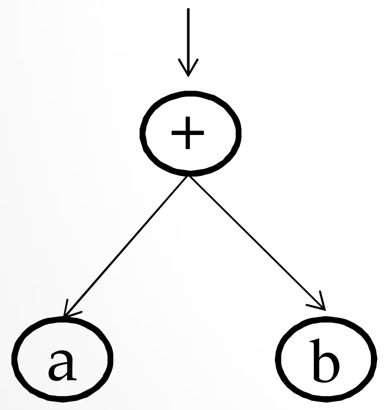
Step 1 : The first tow symbols are operands, so we create one-node trees and push pointers to them onto a stack.



Expression Tree Example (2)

Step 2 : Next, we read '+' so two pointers to trees are popped, a new tree is formed, and a pointer to it is pushed onto the stack.

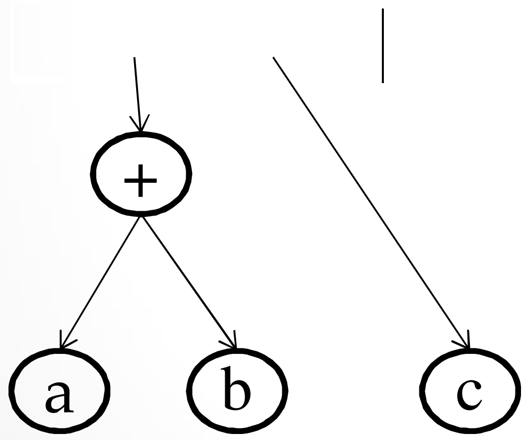
Stack



Expression Tree Example (3)

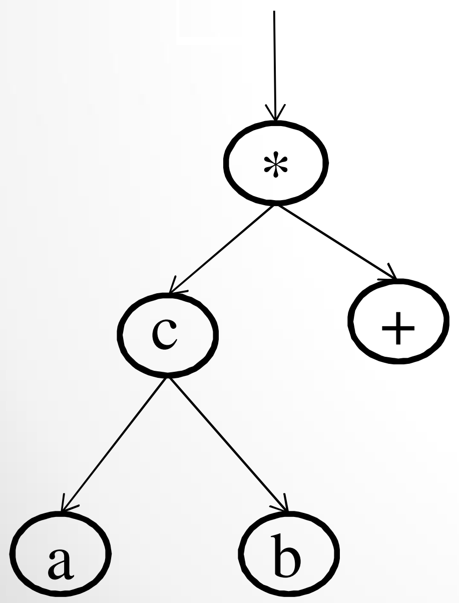
Step 3 : Next, c is read, and a one-node tree is created and a pointer is pushed onto the stack:

Stack



Expression Tree Example (4)

Step 4 : Finally, the last symbol '*' is read, two trees are merged, and a pointer to the final tree is pushed onto the stack.



Hence, the Expression Tree.

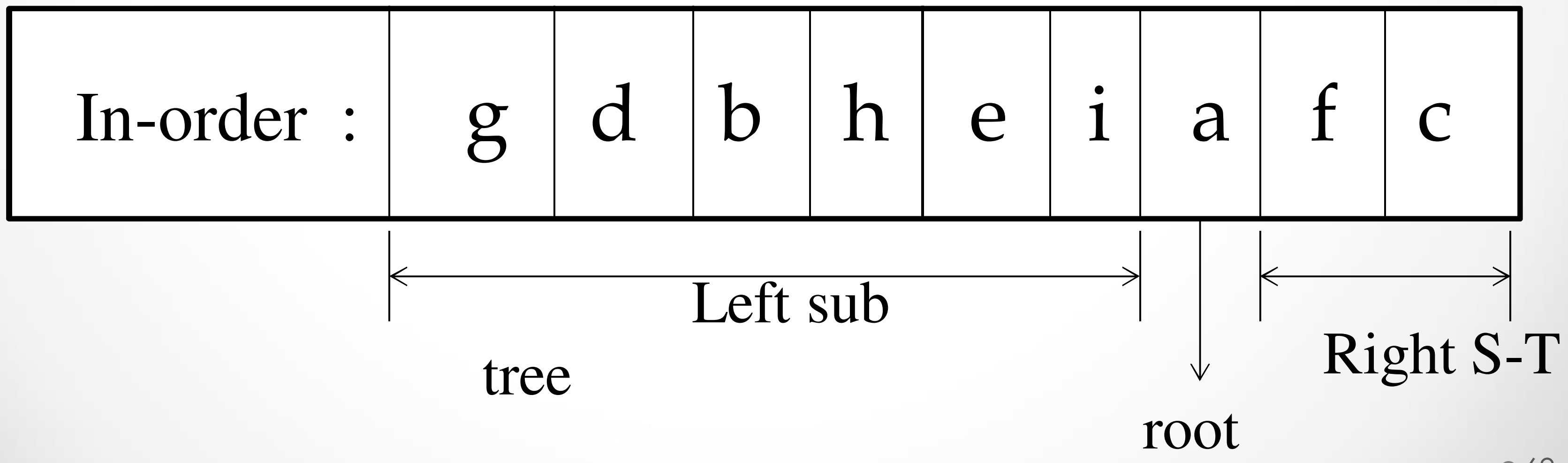
Reconstruction of Binary Trees

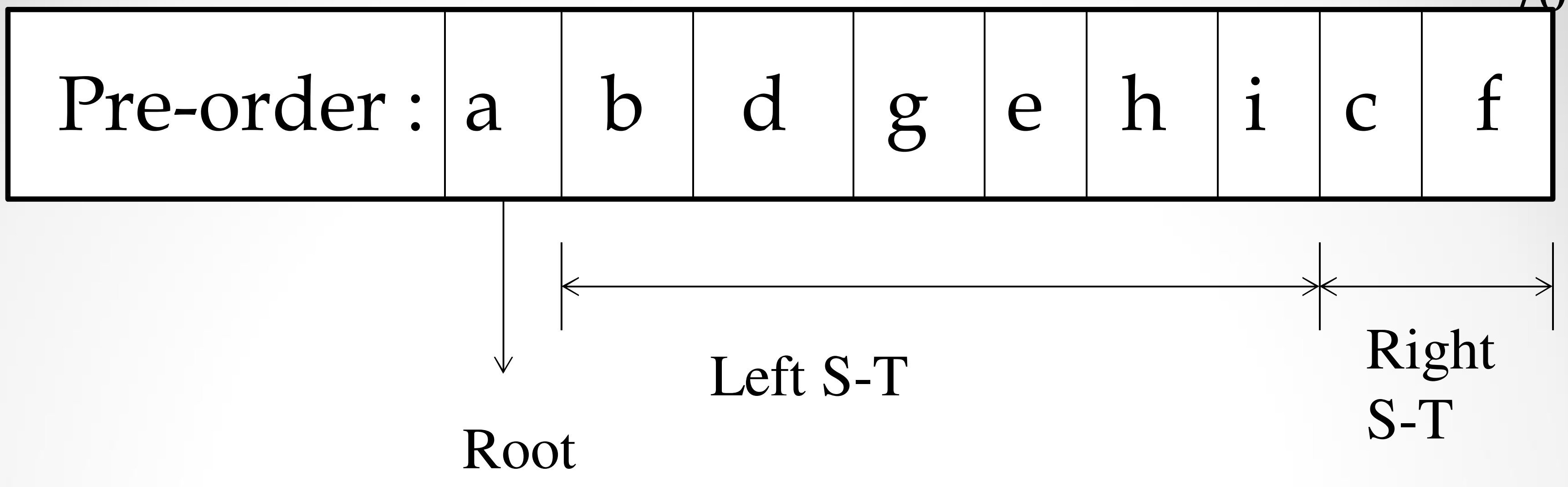
- Consider the pre-order and in-order traversal of binary tree given below:-
- **In-order Traversal :** g d b h e i a f c
- **Pre-order Traversal :** a b d g e h i c f

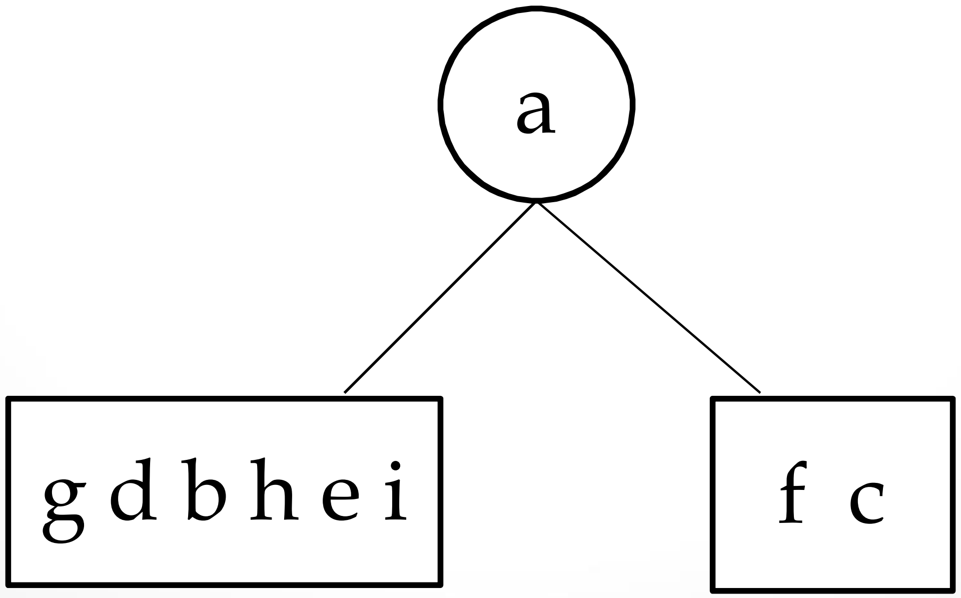
Here, we will construct binary tree from these traversals.

- As in pre-order traversal ,the first element is always the root of the tree . So, the root of the tree is a.
- After determining the root, we have to find the nodes which will form left sub tree and right sub tree of the root.

- In in-order traversal, root of tree lies between nodes forming left sub tree and right sub tree .
- Therefore, after looking at node a in the in-order traversal, we find that the elements g , d , b , e , i form left sub tree and the elements f , c form right sub tree.
- The separation of elements forming left and right sub trees are shown below :

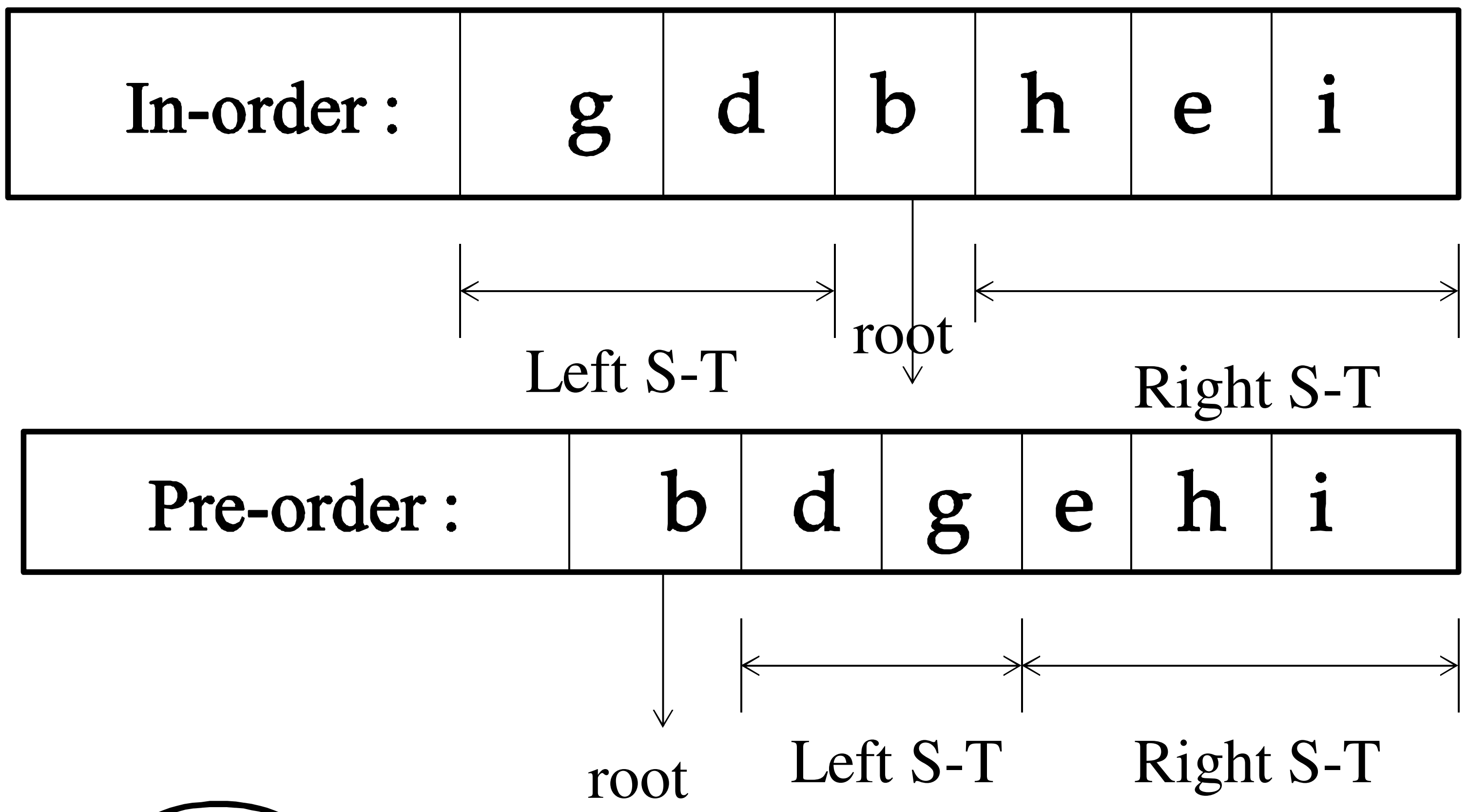



Pre-order : a b d g e h i c f
Root
Left S-T
Right S-T

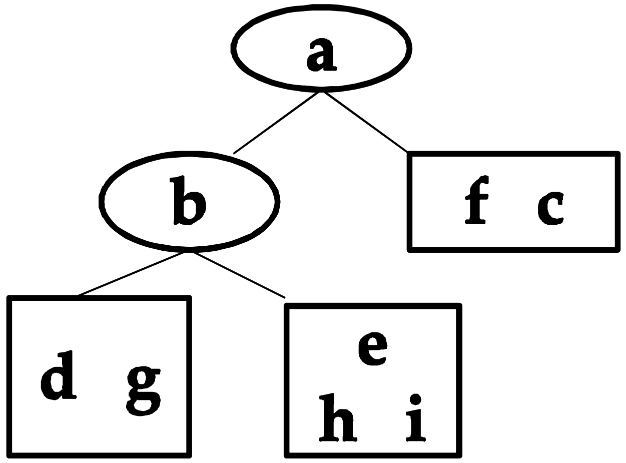

a
g d b h e i
f c

- At this stage , the partial tree can be constructed as shown above;
- In the next stage ,we will construct the left sub tree with the elements of the left sub tree whose in-order and pre-order traversals are given below:
- in-order traversal: g d b h e i
- pre-order traversal: b d g e h i
- By analyzing pre-order , b is the root of left sub tree and after looking for the element b in in-order , we find elements g , d form the left sub tree and the elements h , e , i form the right sub tree of the tree rooted at b as shown below;

•



In-order :
g
d
b
h
e
i
Left S-T
root
Right S-T
Pre-order :
b
d
g
e
h
i
root
Left S-T
Right S-T



a
b
f c
d g
e
h i

- At this stage partial binary tree can be constructed as shown above:
- Now following the same procedure for left sub tree of node b whose in-order and pre-order traversals are:
  - in-order traversal : g d
  - e-order traversal : d g
- From pre-order traversal , d is the root of sub tree and looking for this element d in in-order traversal,
- we find that the element g form the left sub tree and there is no element in the right sub tree of the element d as shown below;

·

In-order :	g	d

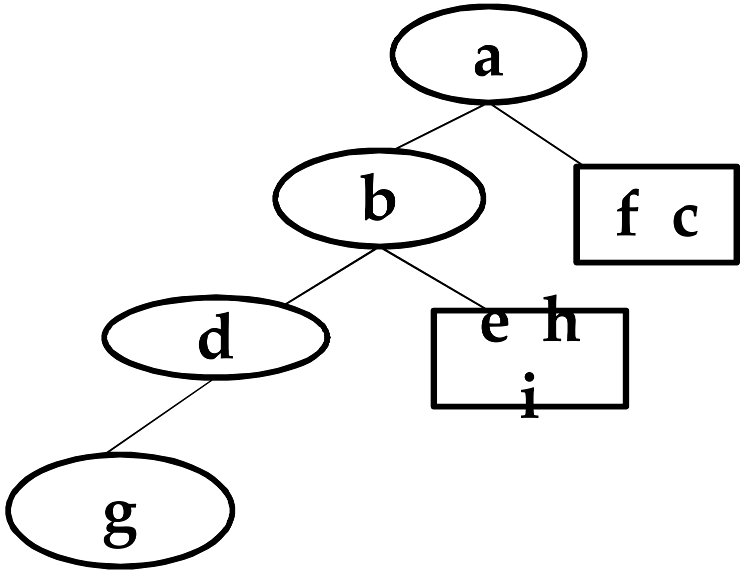
Left S-T

root

Pre-order :	d	g

root

Left S-T



- At this stage the partial binary tree can be constructed as shown above ;
- Now the left sub tree of node **b** has been constructed .

The same procedure applied to elements **e ,h ,i** which belongs to the right sub tree of node **b** and whose in-order and pre-order traversals are :

- **In-order traversal :** h e i
- **Pre-order traversal :**e h i

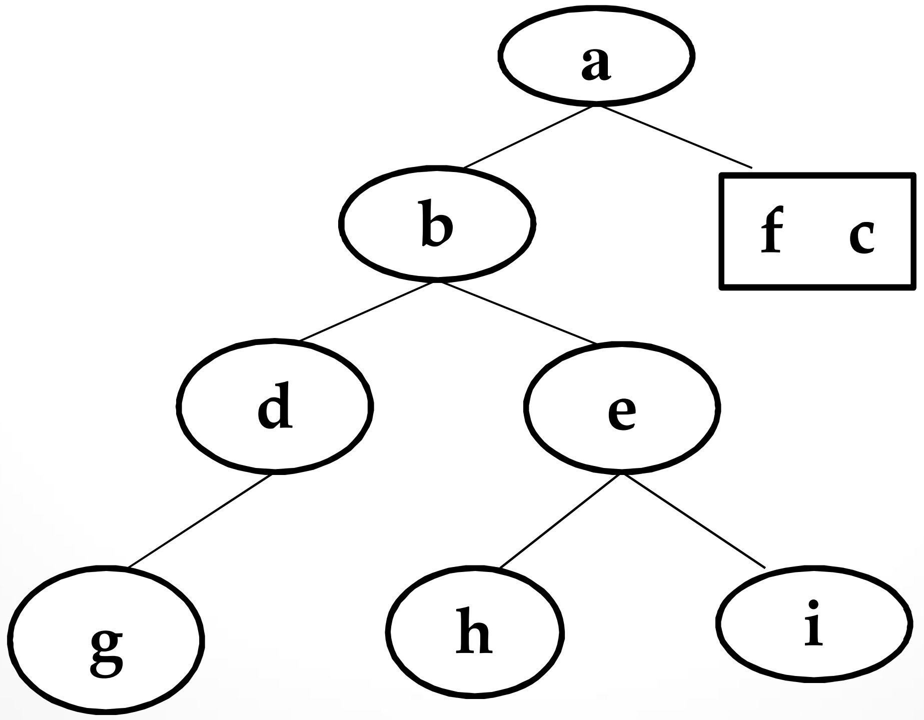
- From the pre-order traversal , **e** is the root of this sub tree and looking for this element in in-order traversal , we find that the element **h** form the left sub tree and the element **i** form the right sub tree as shown below ;

In-order :	h	e	i

Left ST ro ot Right ST

Pre-order :	e	h	i

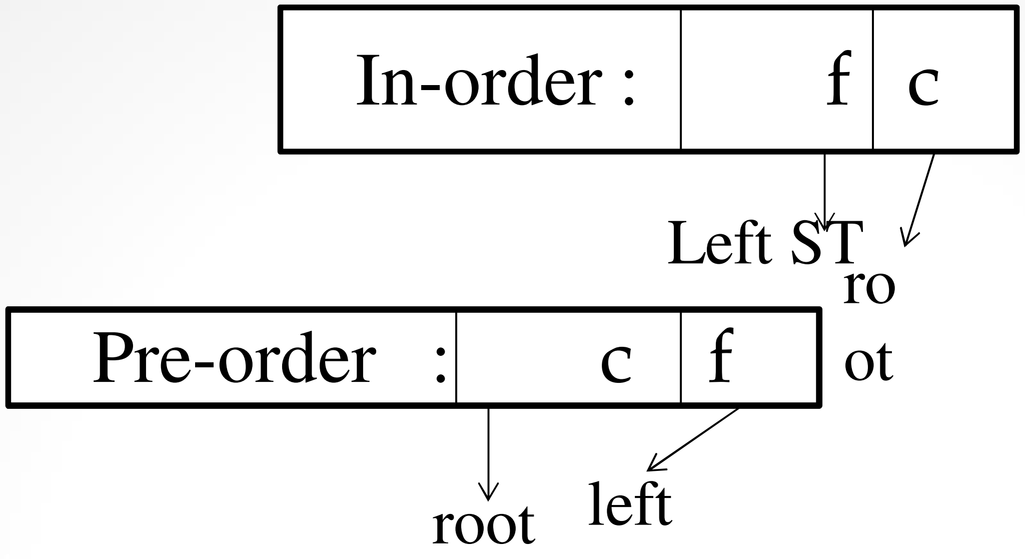
root left right

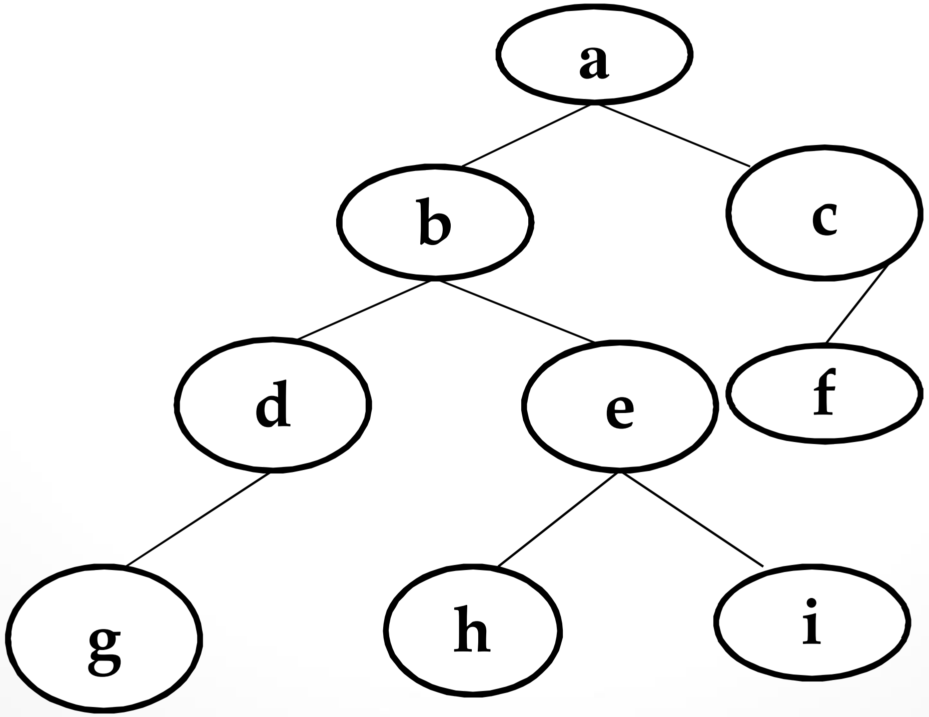


- At this stage the binary tree will look like as shown in figure above ;
- Now only a right sub tree of root element **a** is remaining whose in-order and pre-order traversals are:

  In-order traversal: f c

  Pre-order traversal : c f
- Here ,from pre-order traversal , **c** is the root of this sub tree and looking for this element in in-order traversal , we find that element **f** from the left sub tree and there is no element on the right of element **c** as shown below :


In-order :
f
c
Left ST
ro
ot
Pre-order :
c
f
root
left


a
b
c
d
e
f
g
h
i

- Hence , the required binary tree is constructed from its traversals.
- Similarly we can construct the binary tree from its in-order and post-order traversals.