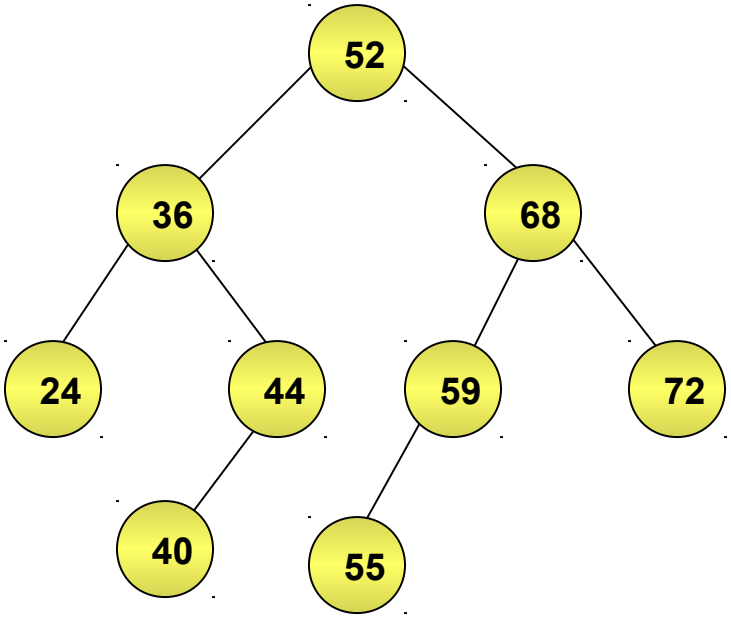
Binary Search Trees

- In this session, you will learn o:
  - Binary Search tree (BST)
  - Operations on a BST
    - Searching
    - Insertion
    - Deletion

Binary Search Tree

- Binary search tree is a binary tree in which every node satisfies the following conditions:
  - All values in the left subtree of a node are less than the value of the node.
  - All values in the right subtree of a node are greater than the value of the node.

- The following is an example of a binary search tree.

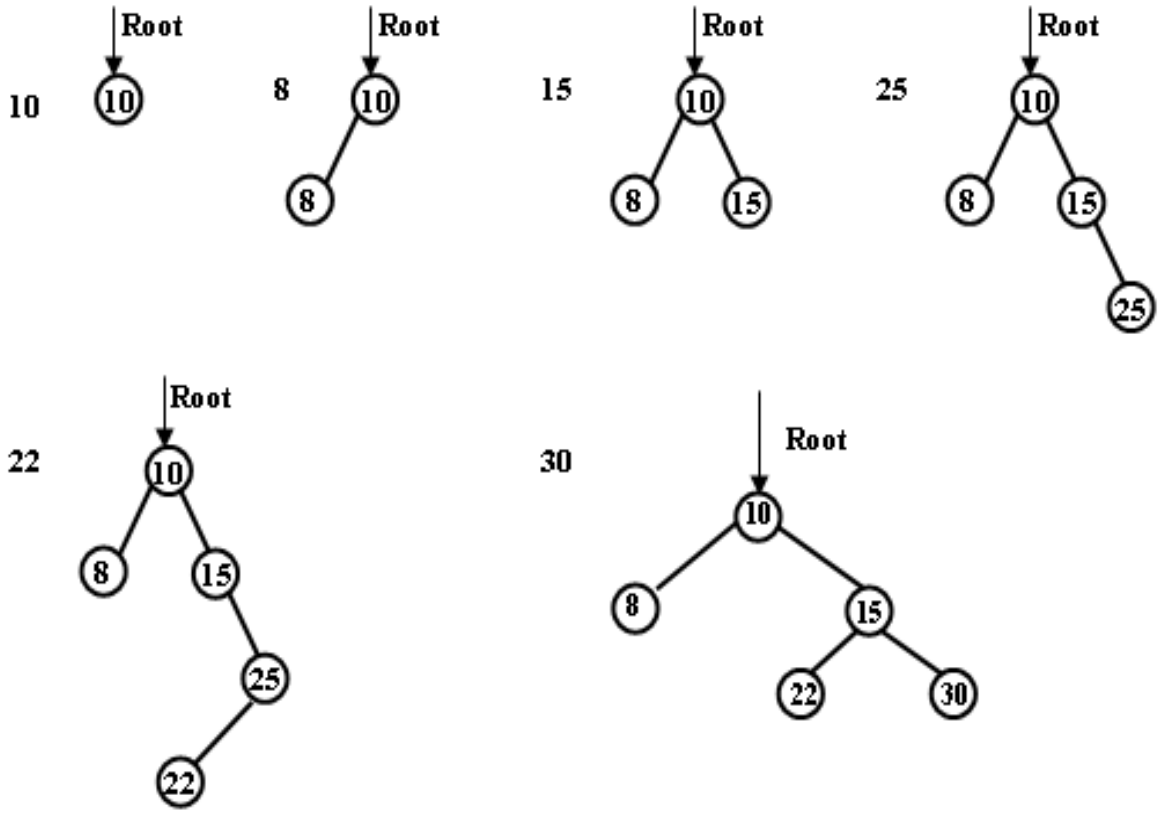


Construction of Binary Search Tree

Ex1: Creating a binary search tree with the following values.

10 8 15 25 22 30

Step by step creation of a binary search tree.



Defining a Binary Search Tree (Contd.)

- You can implement various operations on a binary search tree:
  - Traversal
  - Search
  - Insert
  - Delete

Construction of Binary Search Tree

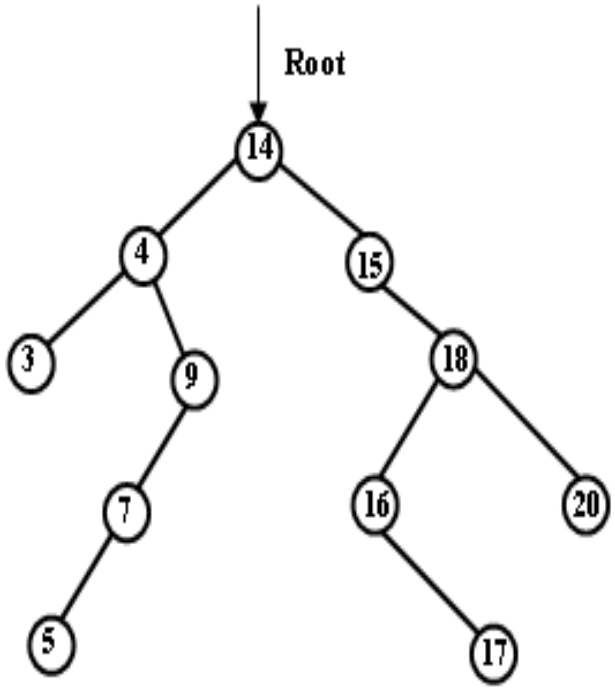
Ex2 : Creating a binary search tree with the following values.

14 4 15 3 9 18 16 20 7 5 17

Construction of Binary Tree

Ex2 : Creating a binary search tree with the following values.

14 4 15 3 9 18 16 20 7 5 17



Searching a Node in a Binary Search Tree

- Search operation refers to the process of searching for a specified value in the tree.
- To search for a specific value, you need to perform the following steps:
  1. Make currentNode point to the root node
  2. If currentNode is null:
     a. Display “Not Found”
     b. Exit
  3. Compare the value to be searched with the value of currentNode. Depending on the result of the comparison, there can be three possibilities:
     a. If the value is equal to the value of currentNode:
        i. Display “Found”
        ii. Exit
     b. If the value is less than the value of currentNode:
        i. Make currentNode point to its left child
        ii. Go to step 2
     c. If the value is greater than the value of currentNode:
        i. Make currentNode point to its right child
        ii. Go to step 2

Searching for a key in a BST

Algorithm:- SearchBST (info, left, right, root, key, LOC, PAR)

{

key is the value to be searched. This procedure find the location LOC of key and also the location PAR of the parent of the key.

1. If (root = NULL) Then
1.1 Print (“Tree does not exist”)
1.2 LOC = NULL and PAR = NULL
1.2 exit
2. PAR = NULL, LOC = NULL
3. ptr = root
4. While (ptr != NULL)
4.1 if (key = ptr -> info) then
4.1.1 LOC = ptr
4.1.2 print PAR AND LOC
4.1.3 exit
4.1 else if (key < ptr -> info) then
4.1.1 PAR = ptr
4.1.2 ptr = ptr -> left
4.1 else
4.1.1 PAR = ptr
4.1.2 ptr = ptr -> right
5. If (LOC = NULL) then
5.1 Print (“Key not found”)

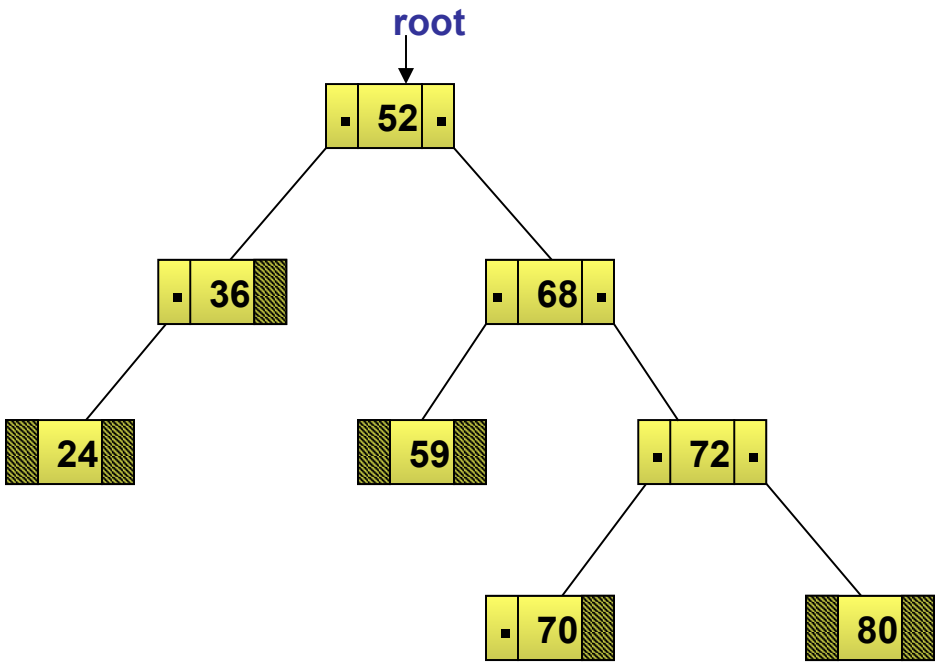
}

Inserting Nodes in a Binary Search Tree (Contd.)

- Write an algorithm to insert a node in a binary search tree.

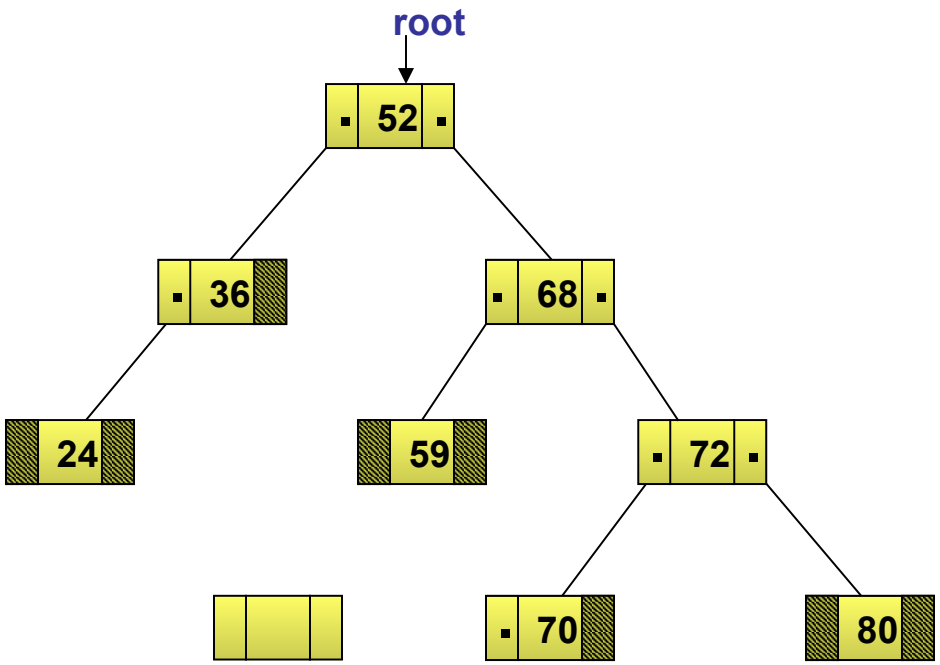
Inserting Nodes in a Binary Search Tree (Contd.)

- Algorithm to insert a node in a binary search tree.
  - Insert 55



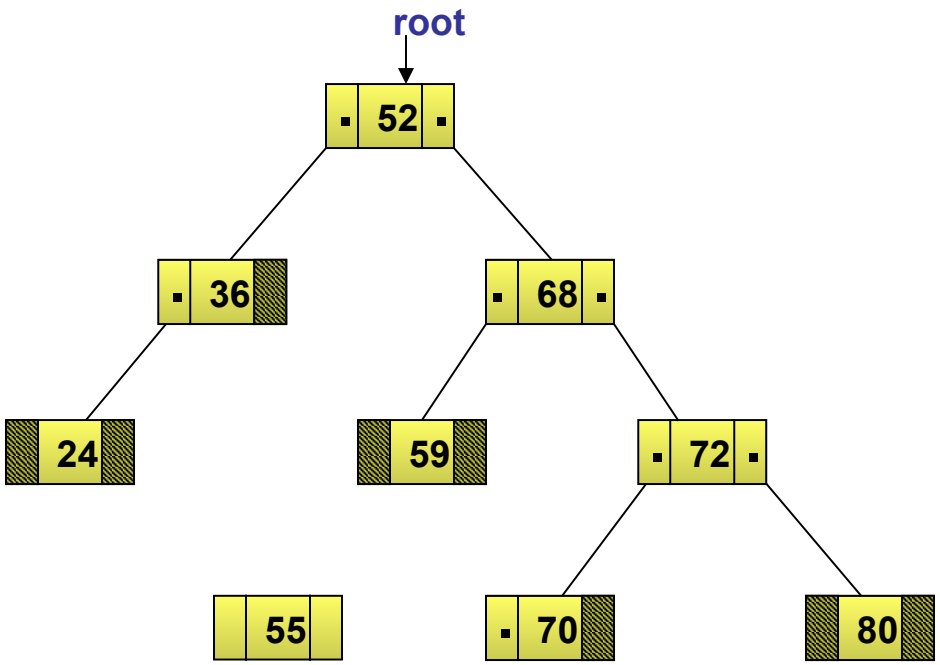
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



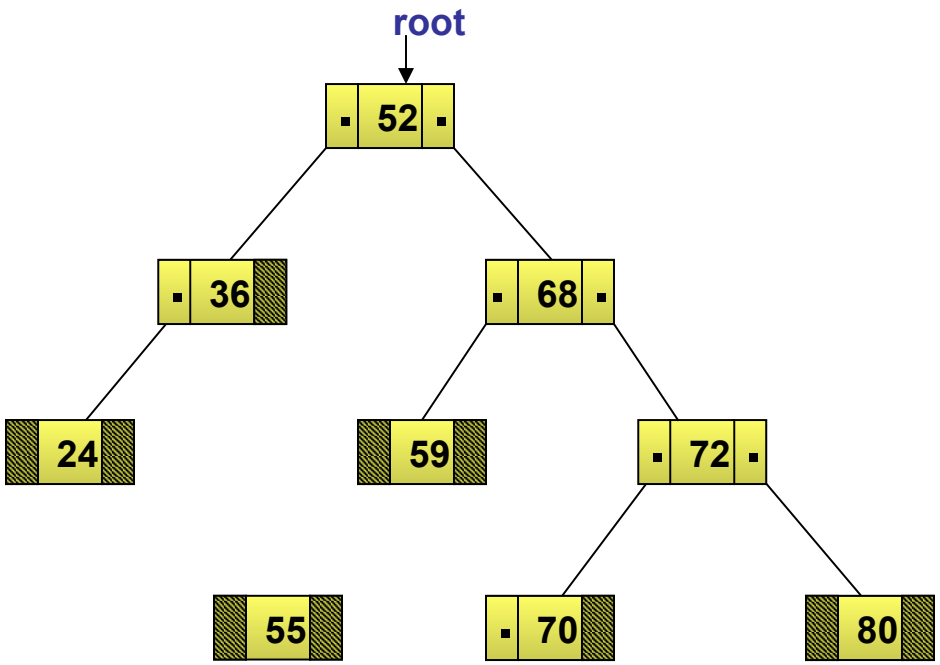
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



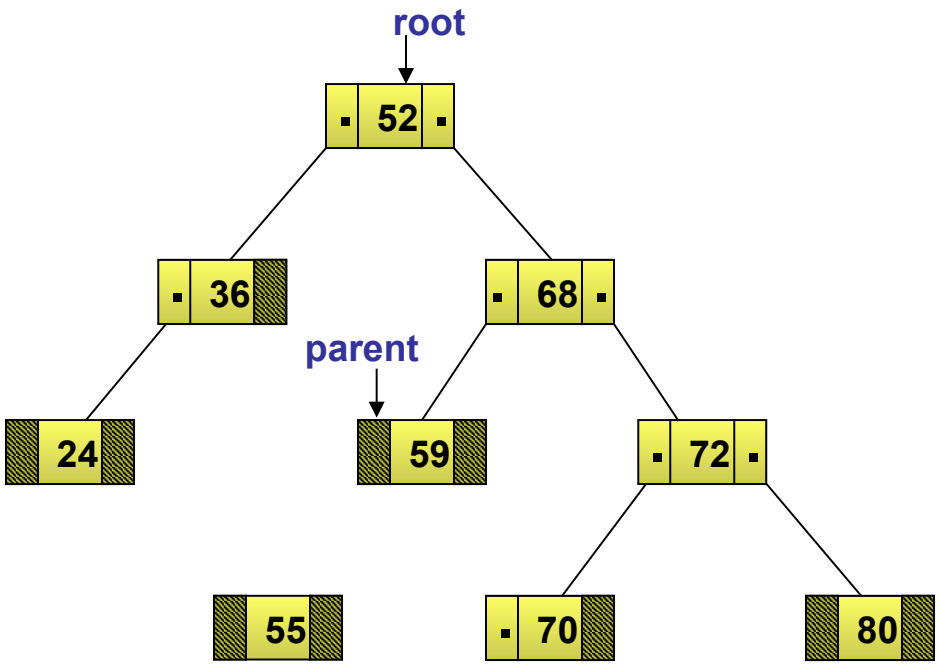
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



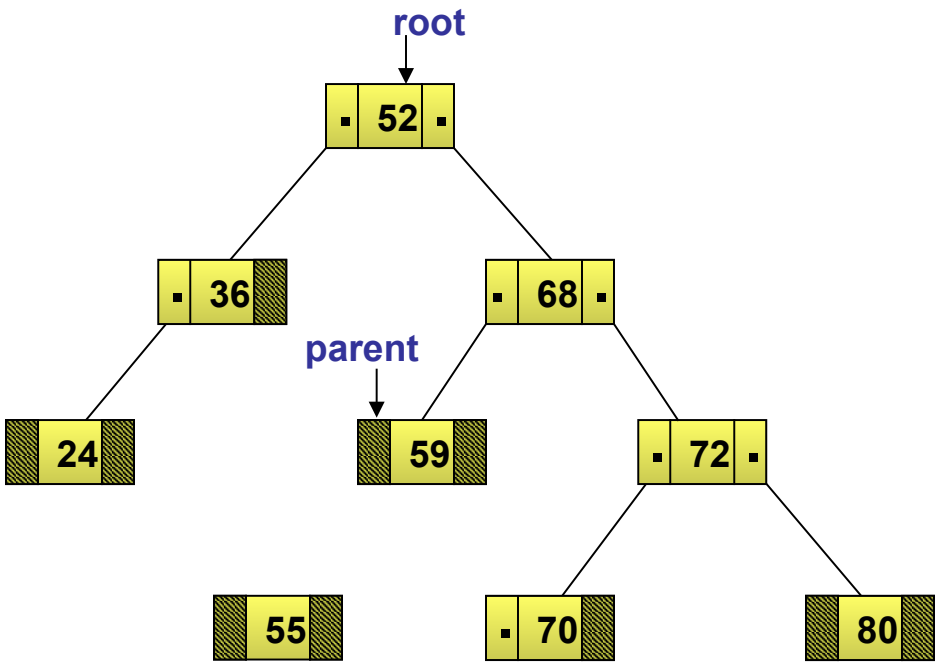
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



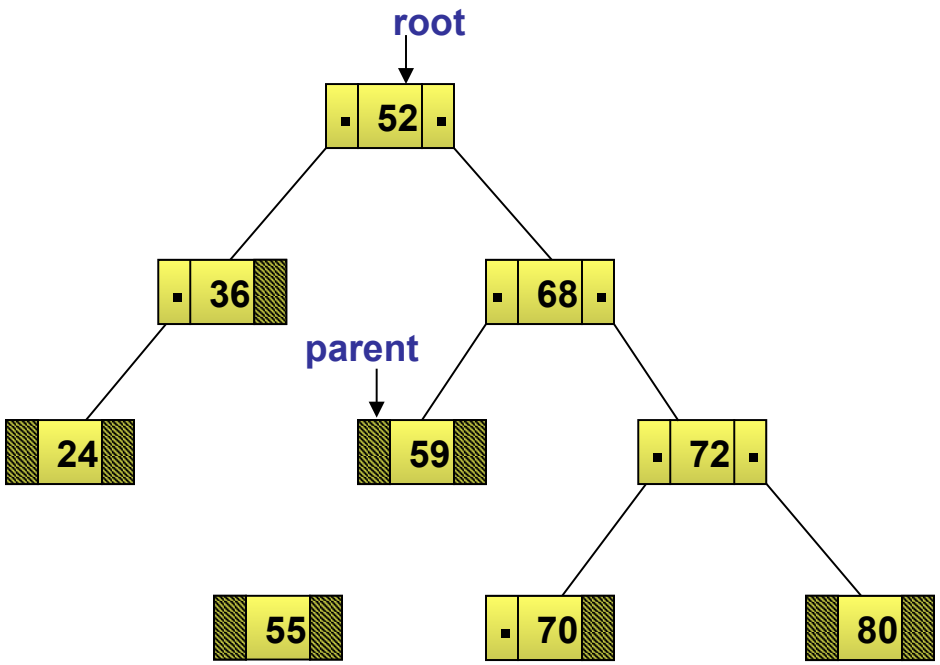
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



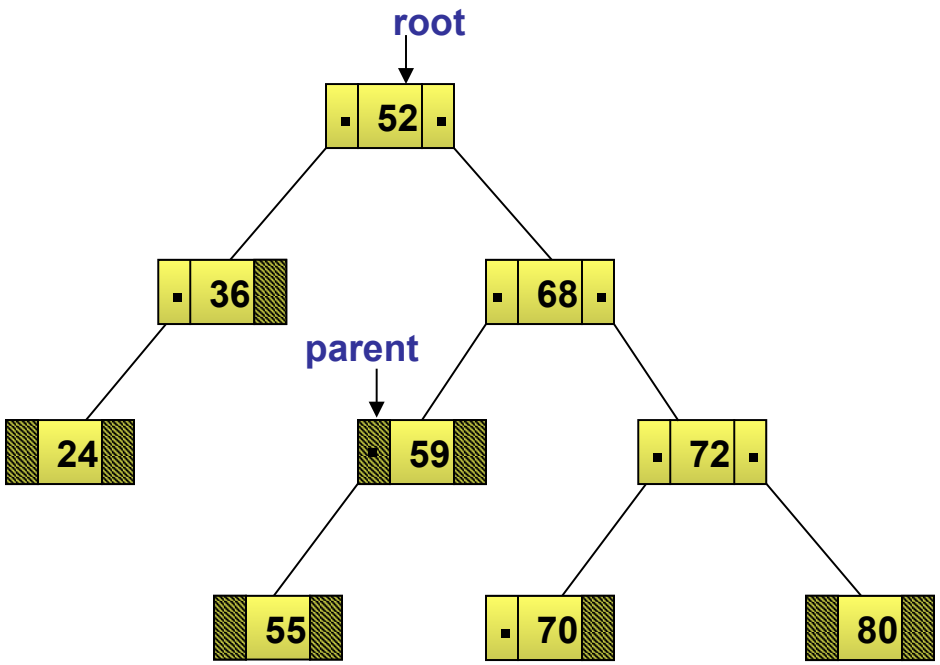
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



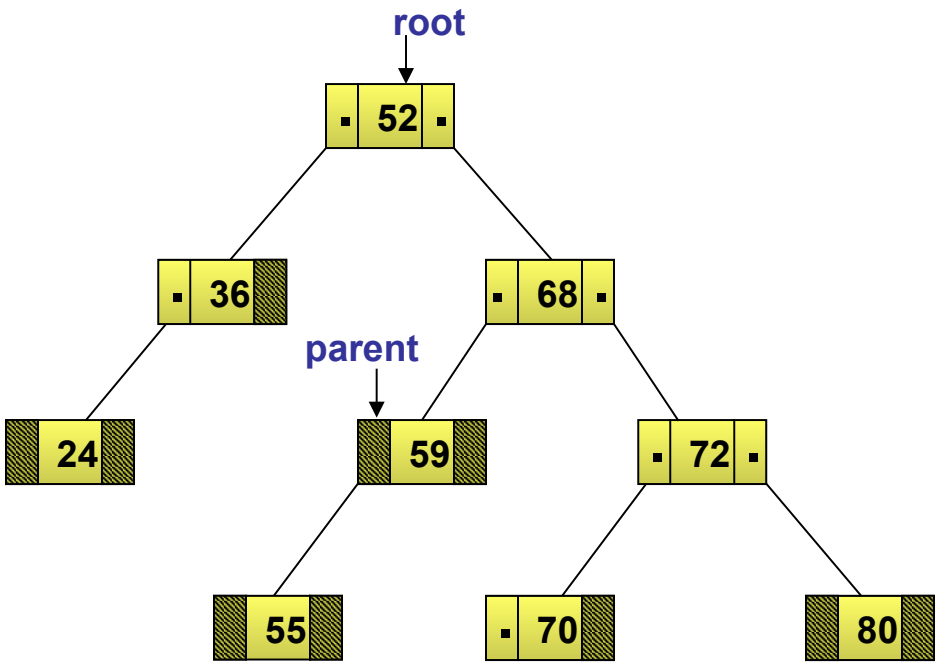
1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Inserting Nodes in a Binary Search Tree (Contd.)



Insert operation complete

1. Allocate memory for the new node.
2. Assign value to the data field of new node.
3. Make the left and right child of the new node point to NULL.
4. Locate the node which will be the parent of the node to be inserted. Mark it as parent.
5. If parent is NULL (Tree is empty):
   a. Mark the new node as ROOT
   b. Exit
6. If the value in the data field of new node is less than that of parent:
   a. Make the left child of parent point to the new node
   b. Exit
7. If the value in the data field of the new node is greater than that of the parent:
   a. Make the right child of parent point to the new node
   b. Exit

Insertion of a key in a BST

Algorithm:- InsertBST (info, left, right, root, key, LOC)

{

key is the value to be inserted.

1. call SearchBST (info, left, right, root, key, LOC , PAR) // Find the parent of the new node
2. If (LOC != NULL)

2.1 Print " Node alredy exist"

2.2 Exit

3. create a node [new1 = (struct node*) malloc (sizeof(struct node))]
4. new1 -> info = key
5. new1 -> left = NULL , new1 -> right = NULL
6. If (PAR = NULL) Then

6.1 root = new1

6.2 exit

elseif (new1 -> info < PAR -> info)

6.1 PAR -> left = new1

6.2 exit

else

6.1 PAR -> right = new1

6.2 exit

}

Deleting Nodes from a Binary Search Tree

- Write an algorithm to locate the position of the node to deleted from a binary search tree.

- Delete operation in a binary search tree refers to the process of deleting the specified node from the tree.
- Before implementing a delete operation, you first need to locate the position of the node to be deleted and its parent.
- To locate the position of the node to be deleted and its parent, you need to implement a search operation.

Deleting Nodes from a Binary Search Tree (Contd.)

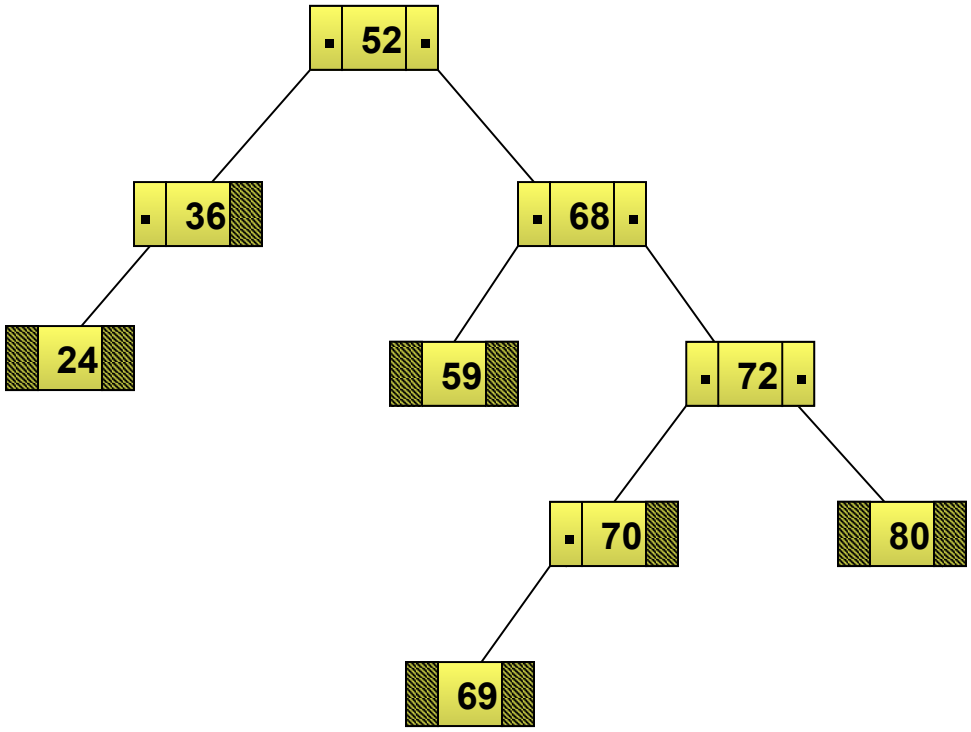
- Once the nodes are located, there can be three cases:
  - **Case I**: Node to be deleted is the leaf node
  - **Case II**: Node to be deleted has one child (left or right)
  - **Case III**: Node to be deleted has two children

Deleting Nodes from a Binary Search Tree (Contd.)

- Write an algorithm to delete a leaf node from a binary search tree.

Deleting Nodes from a Binary Search Tree (Contd.)

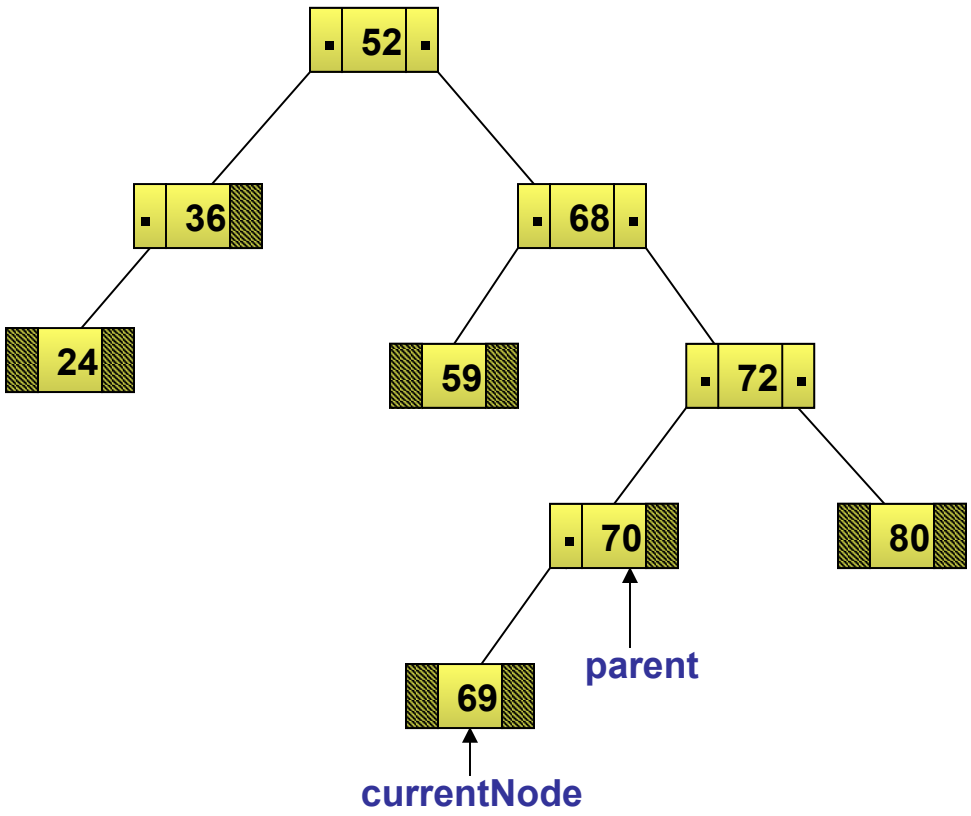
- Algorithm to delete a leaf node from the binary tree.
- Delete node 69



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: **// If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

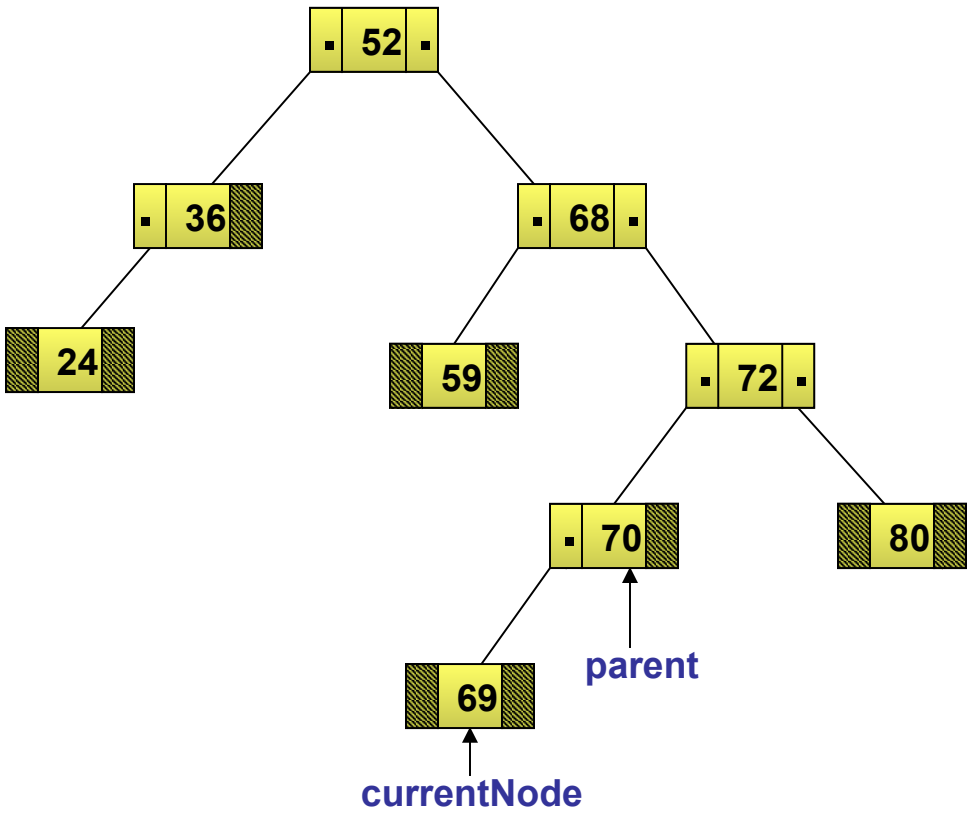
- Delete node 69



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: // **If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

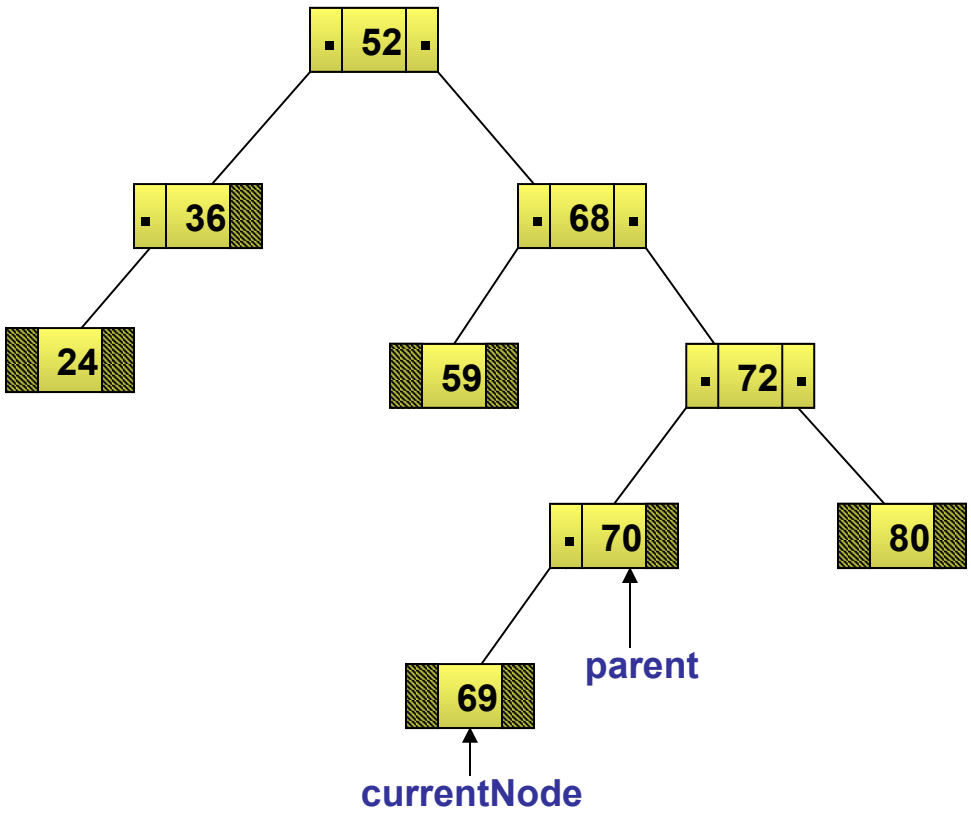
- Delete node 69



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: **// If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

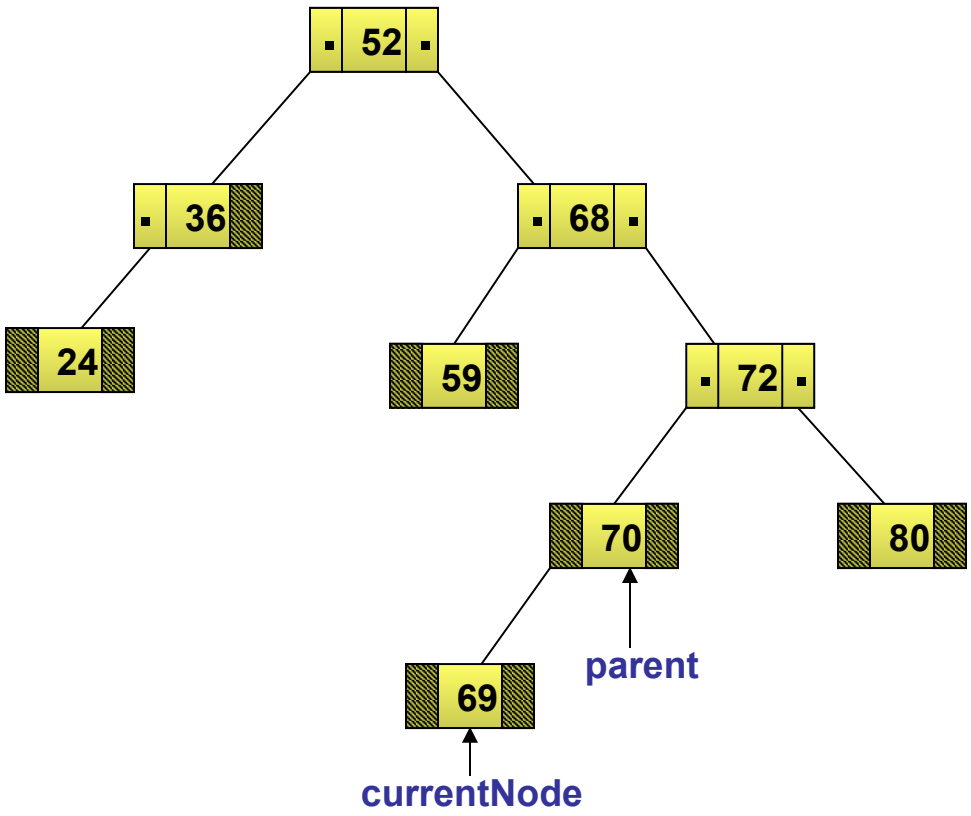
- Delete node 69



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: **// If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

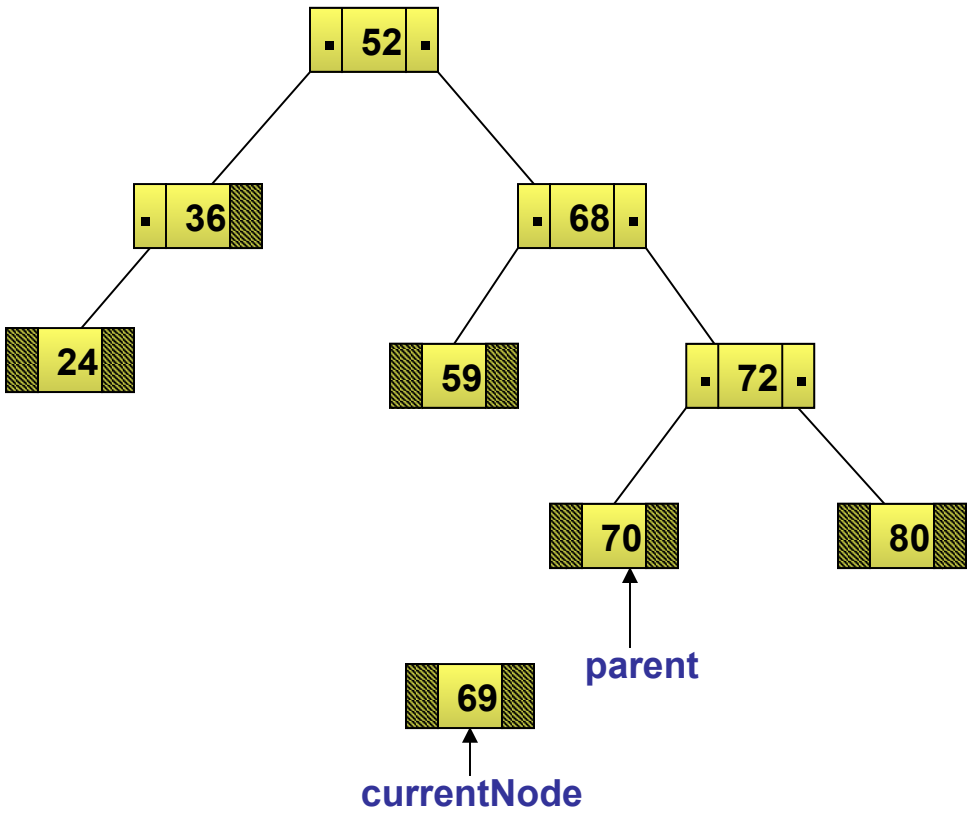
- Delete node 69



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: **// If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

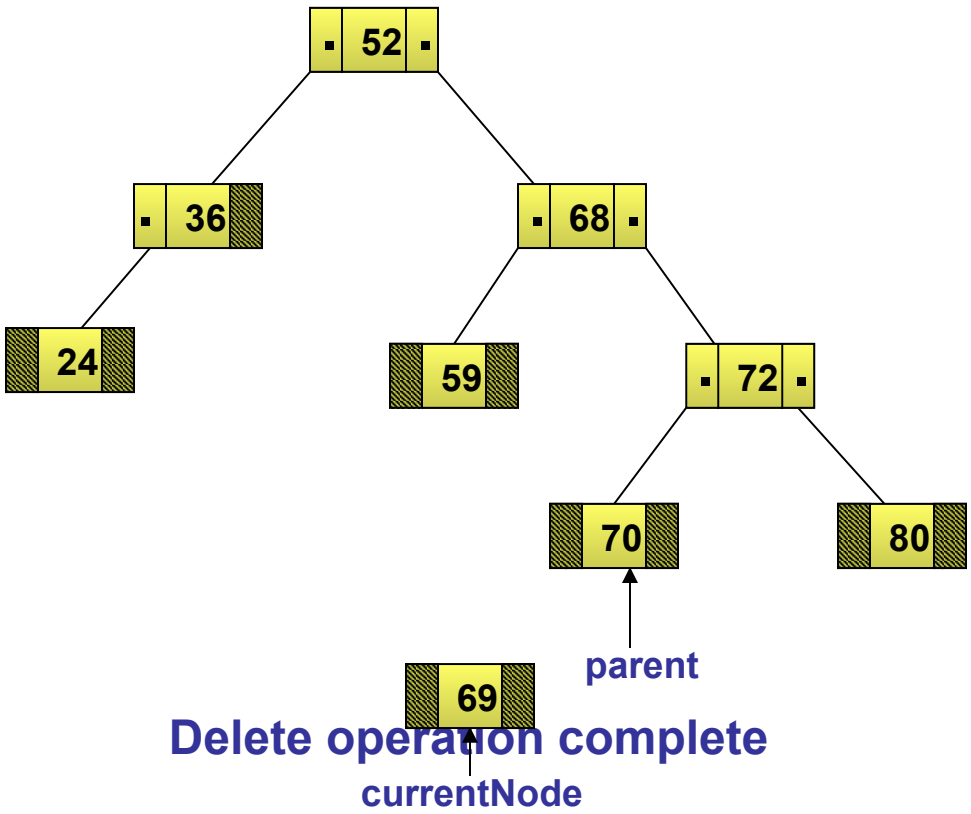
- Delete node 69



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: **// If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

- Delete node 69



Delete operation complete

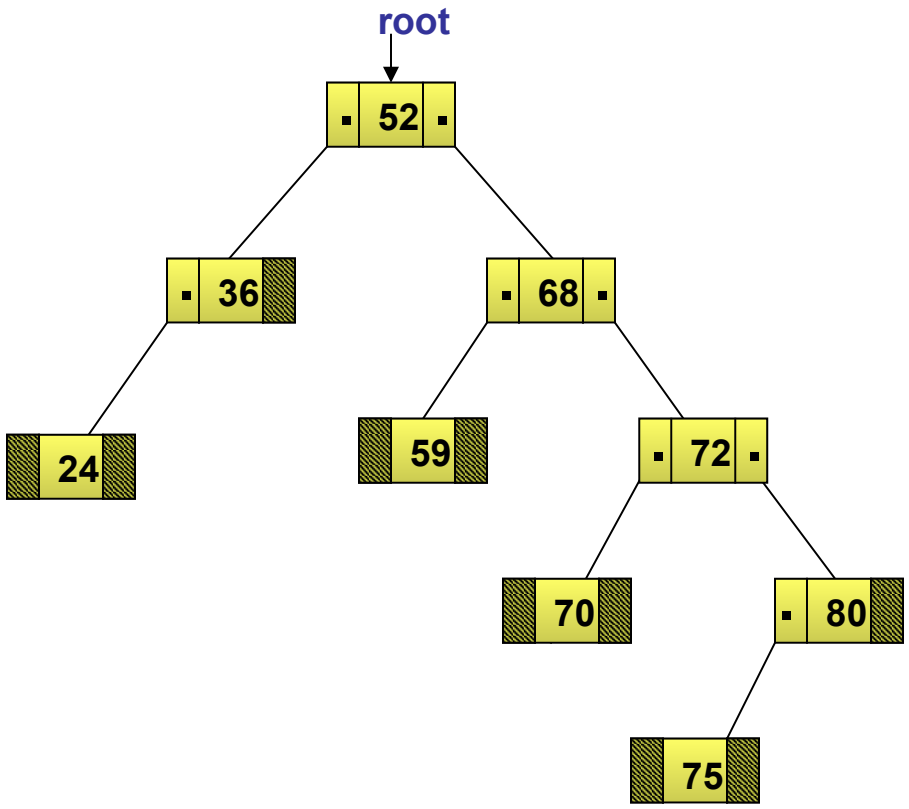
1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode is the root node: **// If parent is // NULL**
   a. Make ROOT point to NULL.
   b. Go to step 5.
3. If currentNode is left child of parent:
   a. Make left child field of parent point to NULL.
   b. Go to step 5.
4. If currentNode is right child of parent:
   a. Make right child field of parent point to NULL.
   b. Go to step 5.
5. Release the memory for currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

- Write an algorithm to delete a node, which has one child from a binary search tree.

Deleting Nodes from a Binary Search Tree (Contd.)

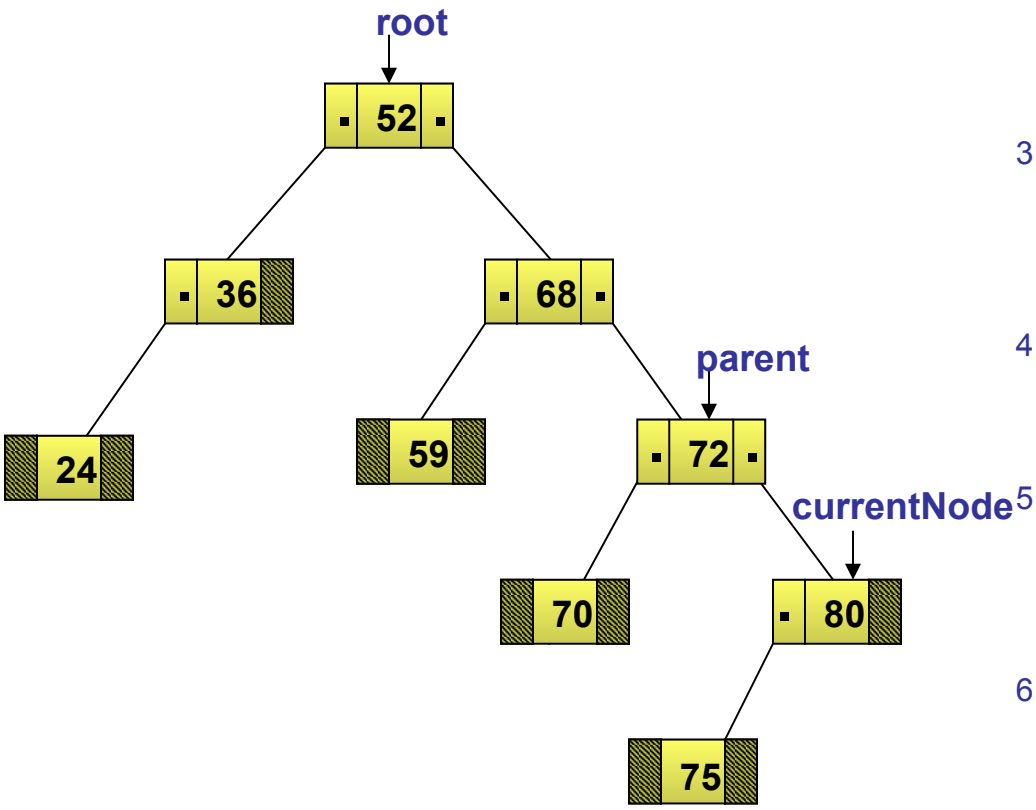
- Algorithm to delete a node with one child.
  - Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

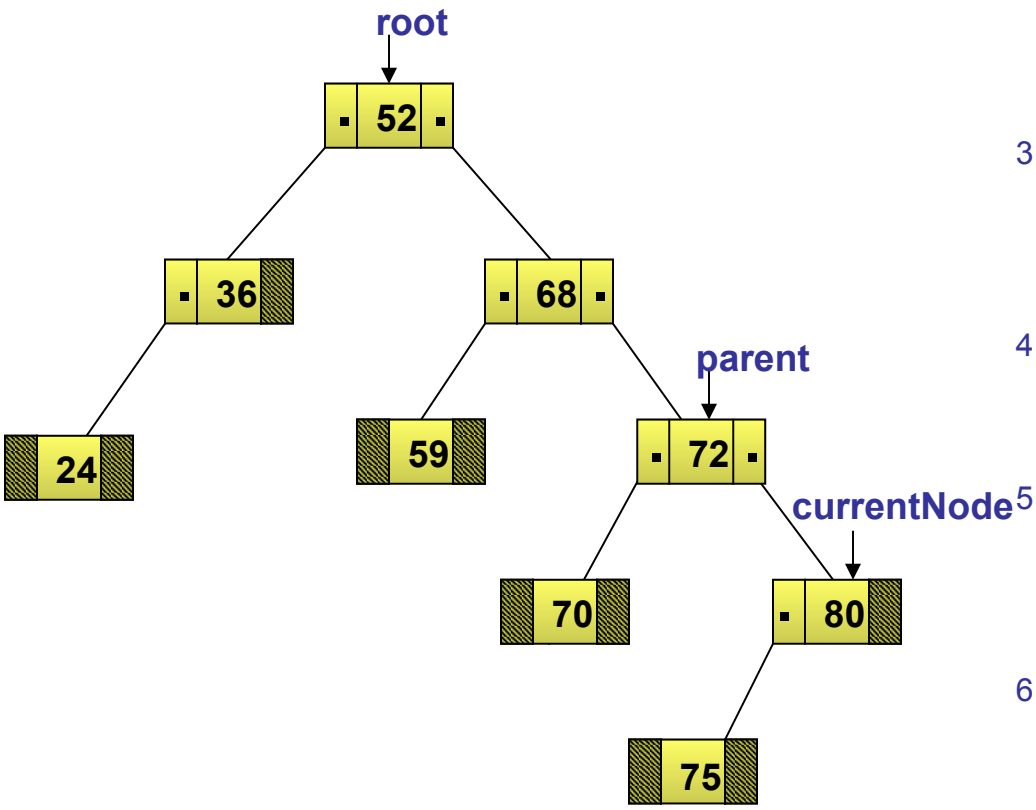
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

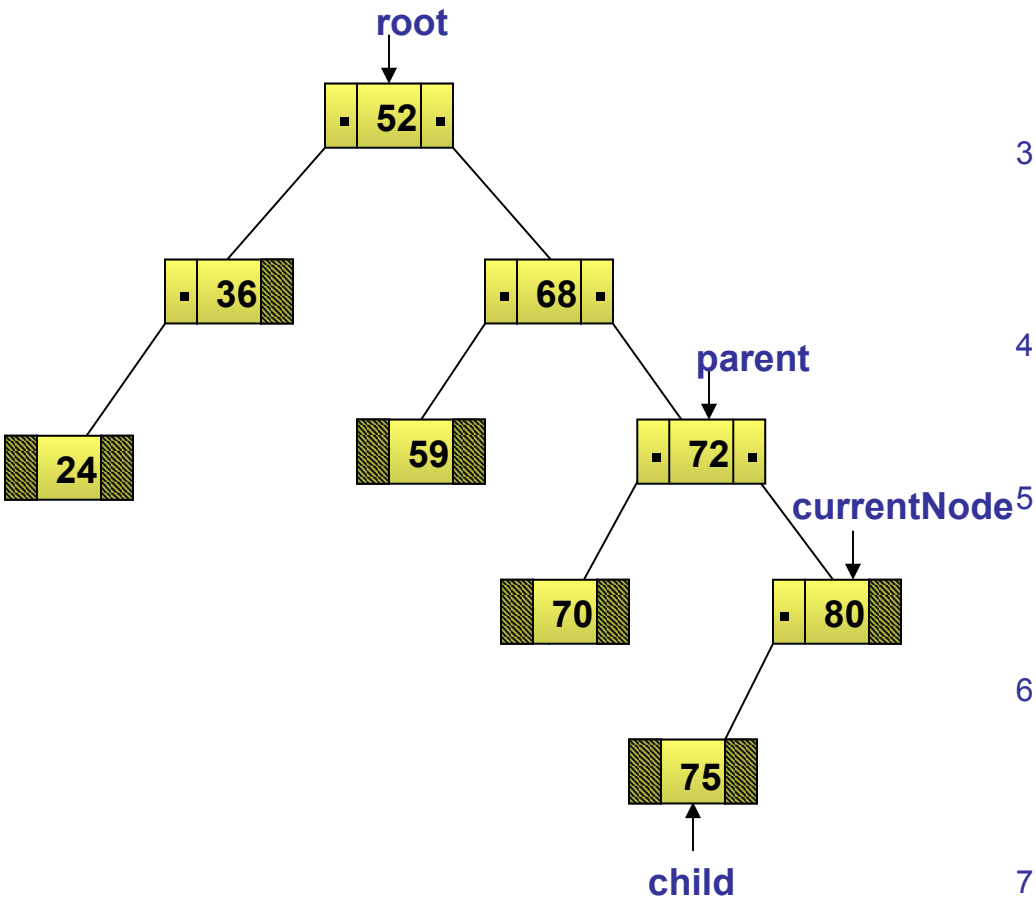
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

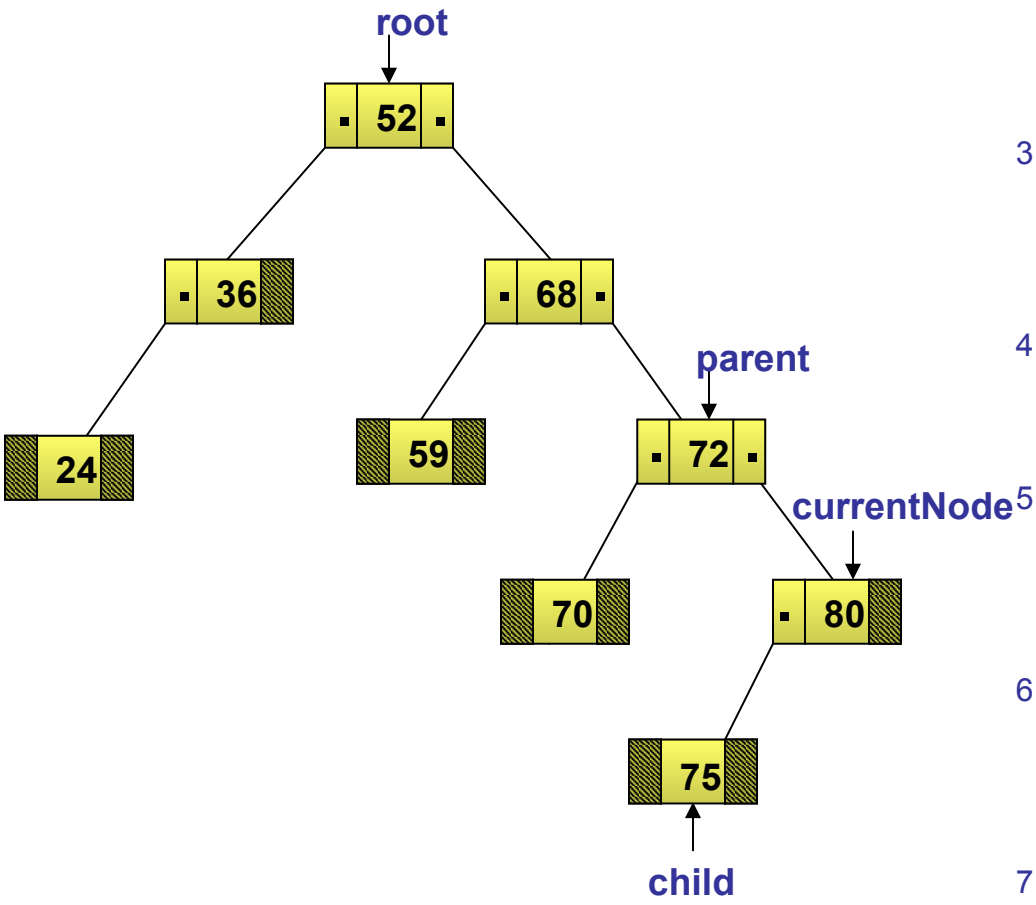
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

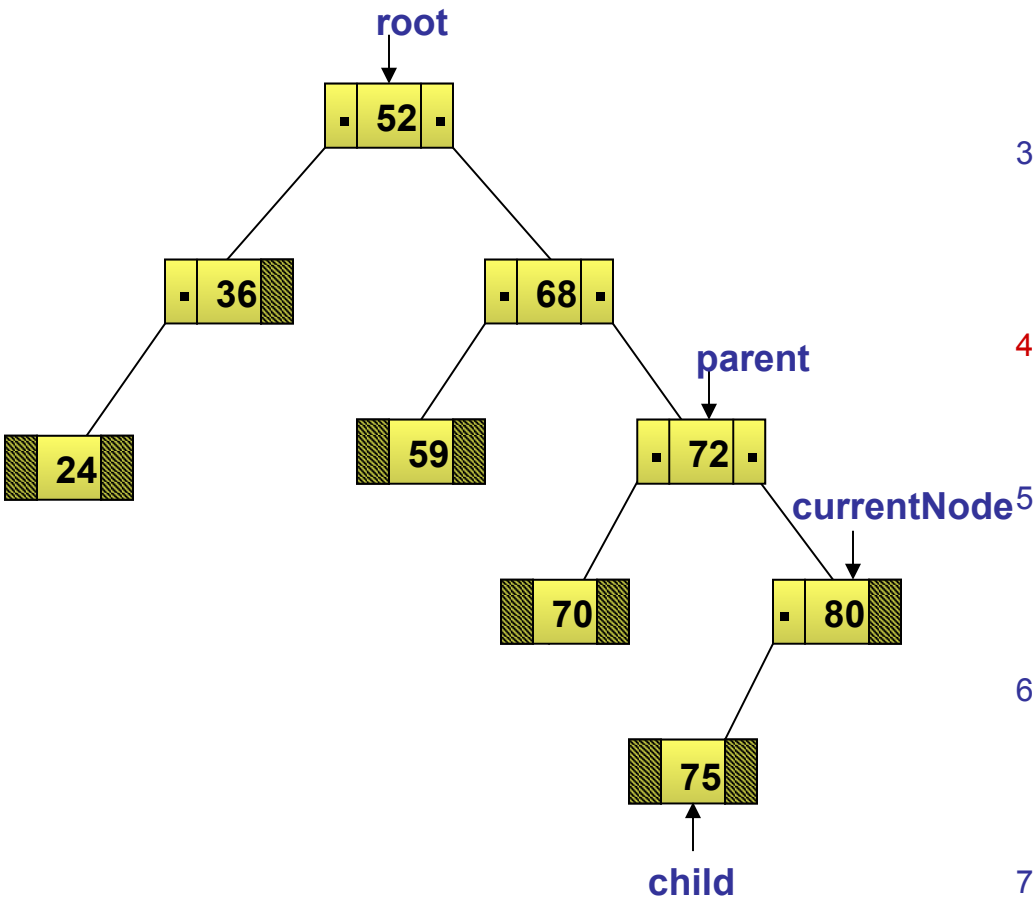
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

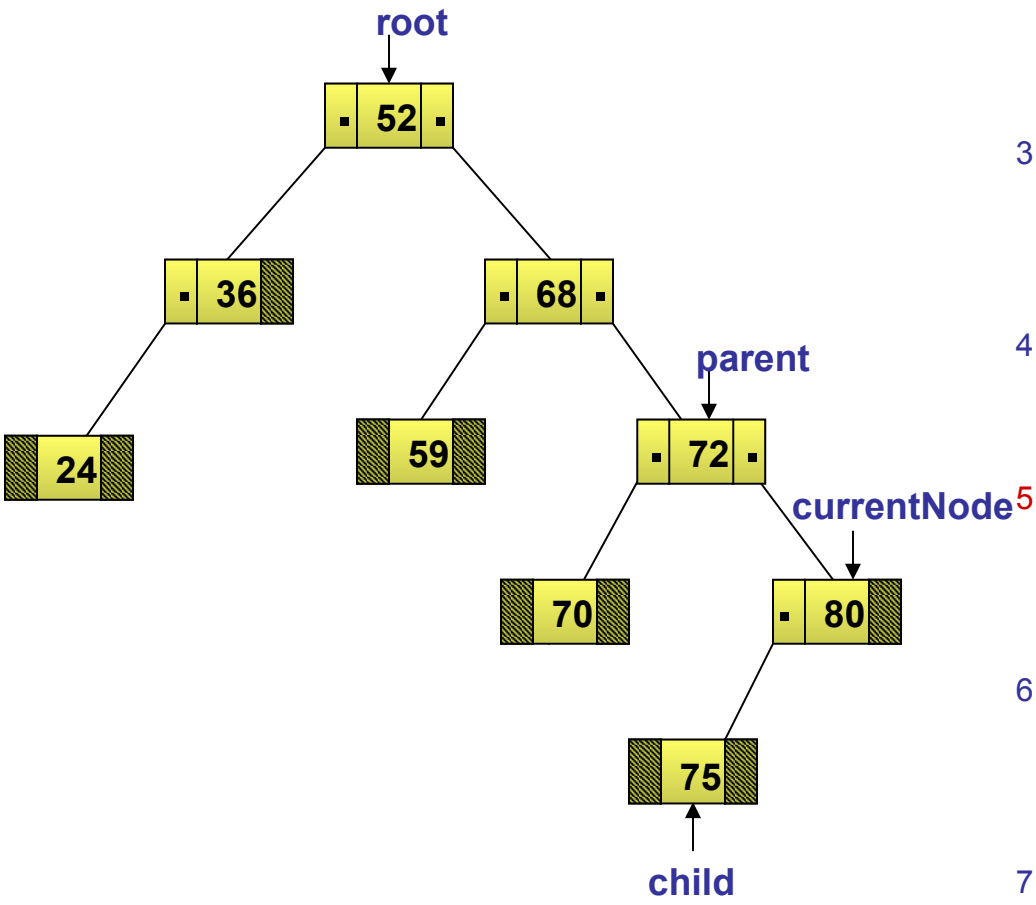
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

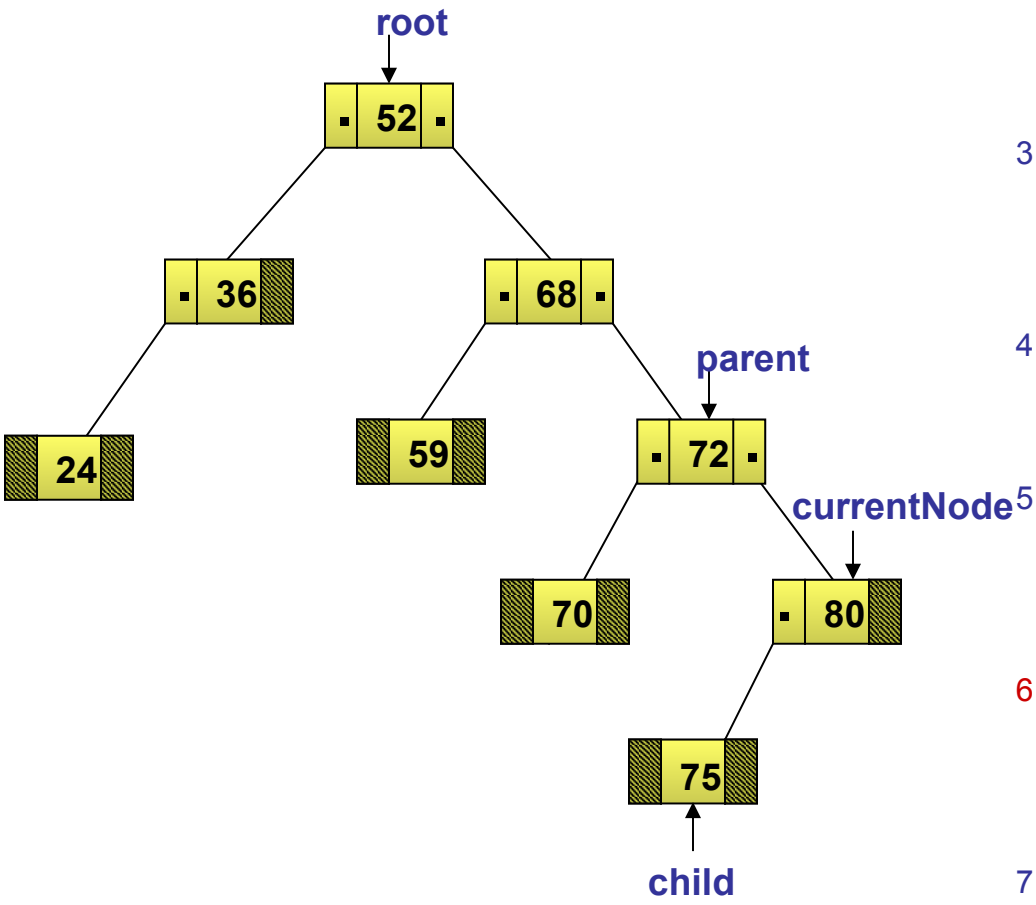
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

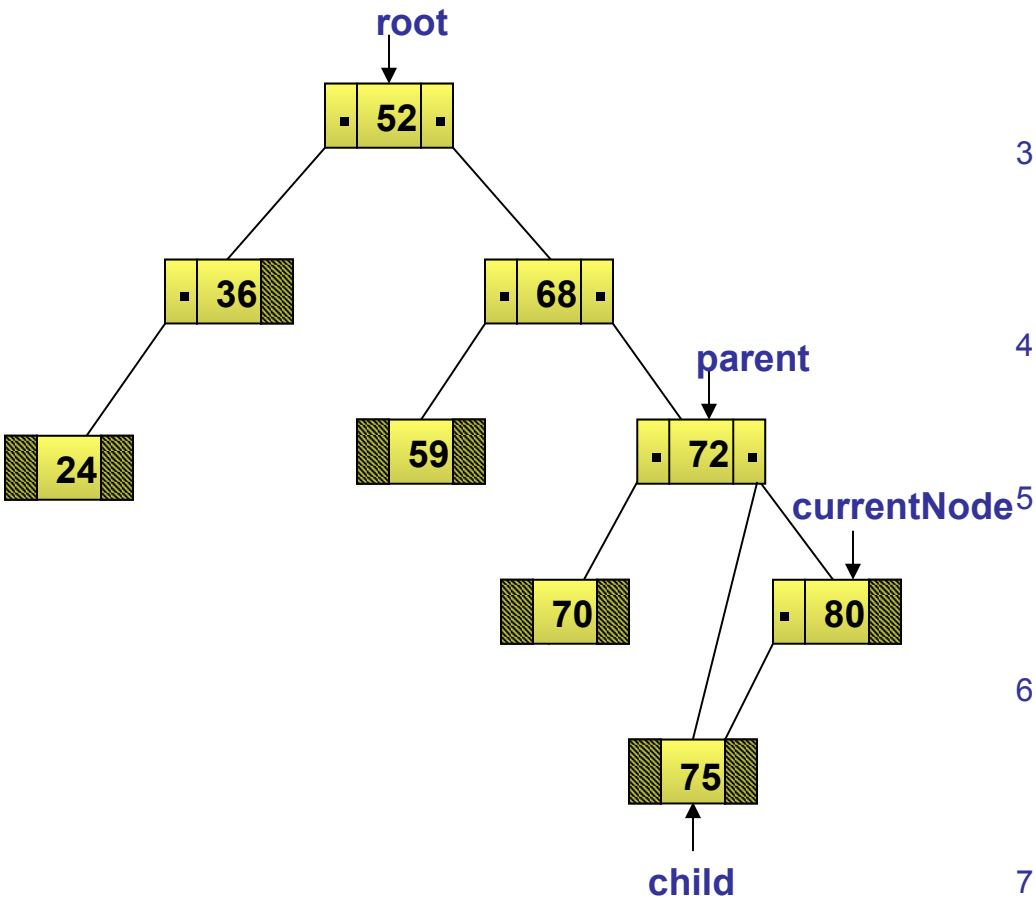
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

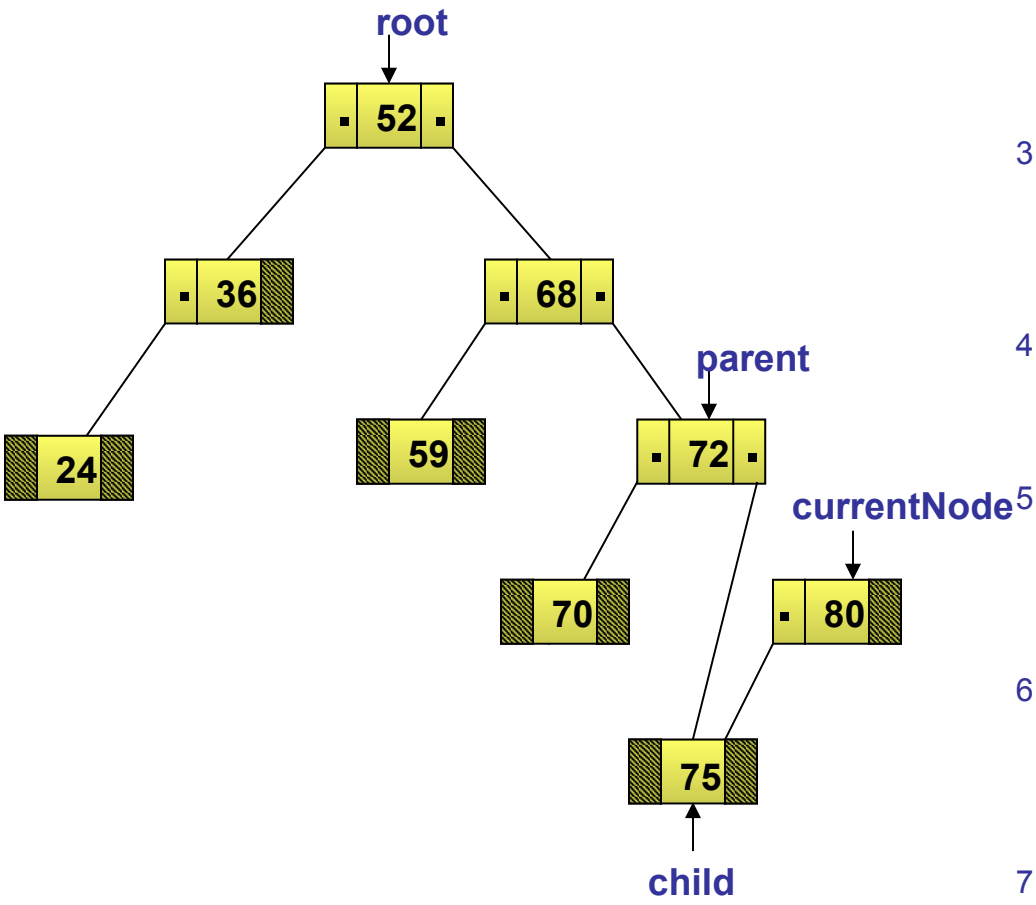
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

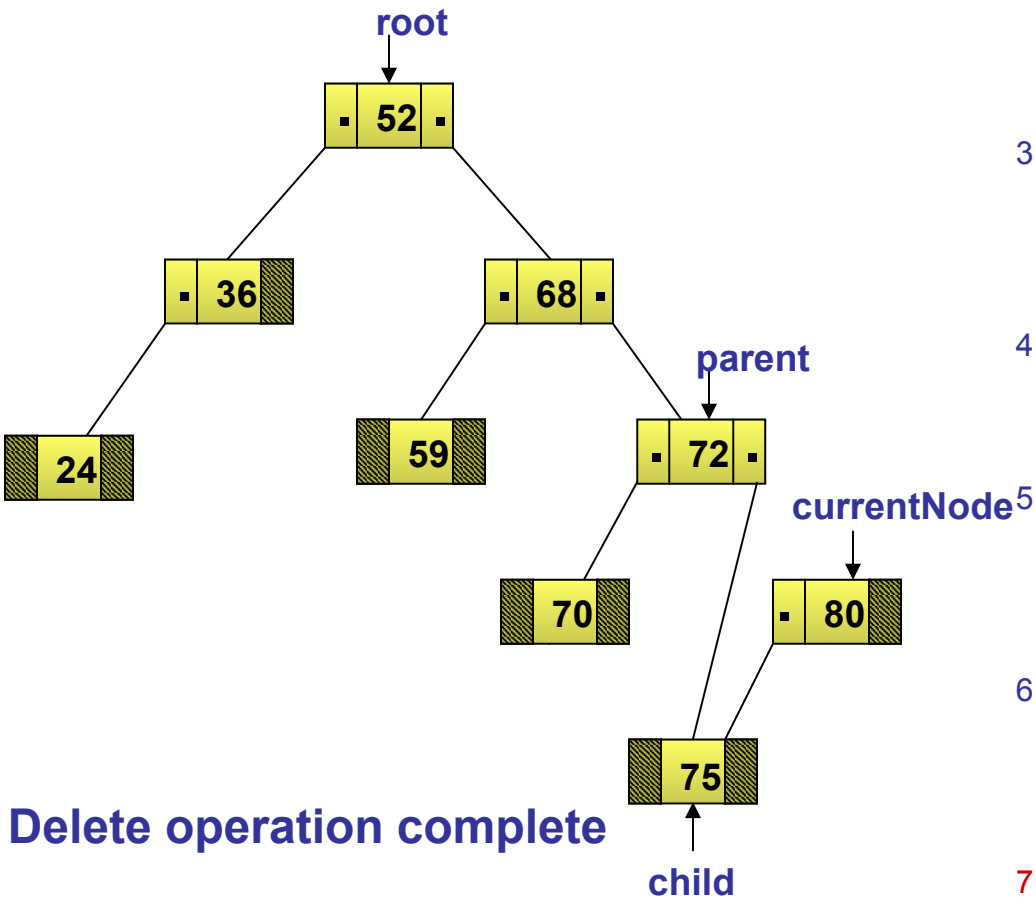
- Delete node 80



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

- Delete node 80



Delete operation complete

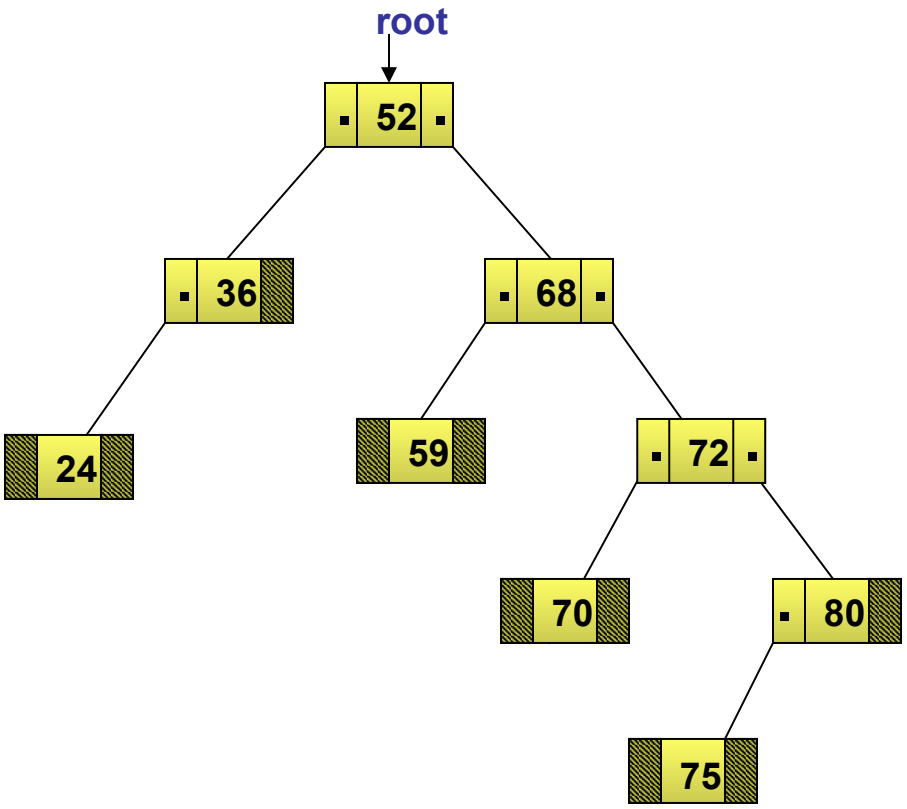
1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. If currentNode has a left child:
   a. Mark the left child of currentNode as child.
   b. Go to step 4.
3. If currentNode has a right child:
   a. Mark the right child of currentNode as child.
   b. Go to step 4.
4. If currentNode is the root node:
   a. Mark child as root.
   b. Go to step 7.
5. If currentNode is the left child of parent:
   a. Make left child field of parent point to child.
   b. Go to step 7.
6. If currentNode is the right child of parent:
   a. Make right child field of parent point to child.
   b. Go to step 7.
7. Release the memory of currentNode.

Deleting Nodes from a Binary Search Tree (Contd.)

- Write an algorithm to delete a node, which has two children from a binary search tree.

Deleting Nodes from a Binary Search Tree (Contd.)

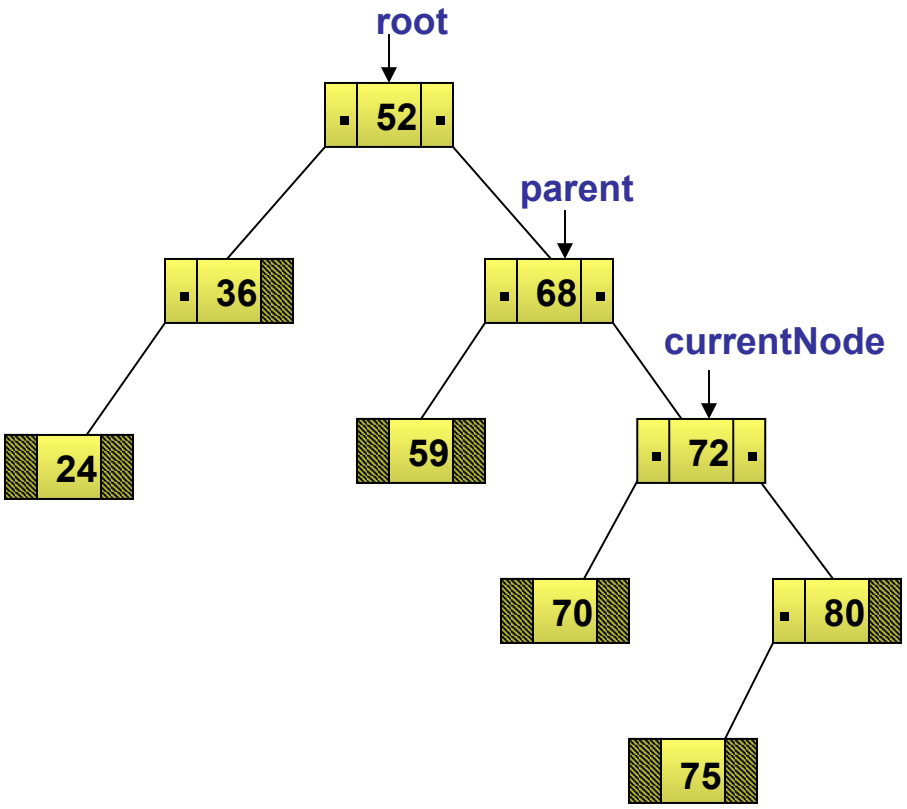
- Algorithm to delete a node with two children.
  - Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.
3. Replace the information held by currentNode with that of Inorder_suc.
4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.
5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.

2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.

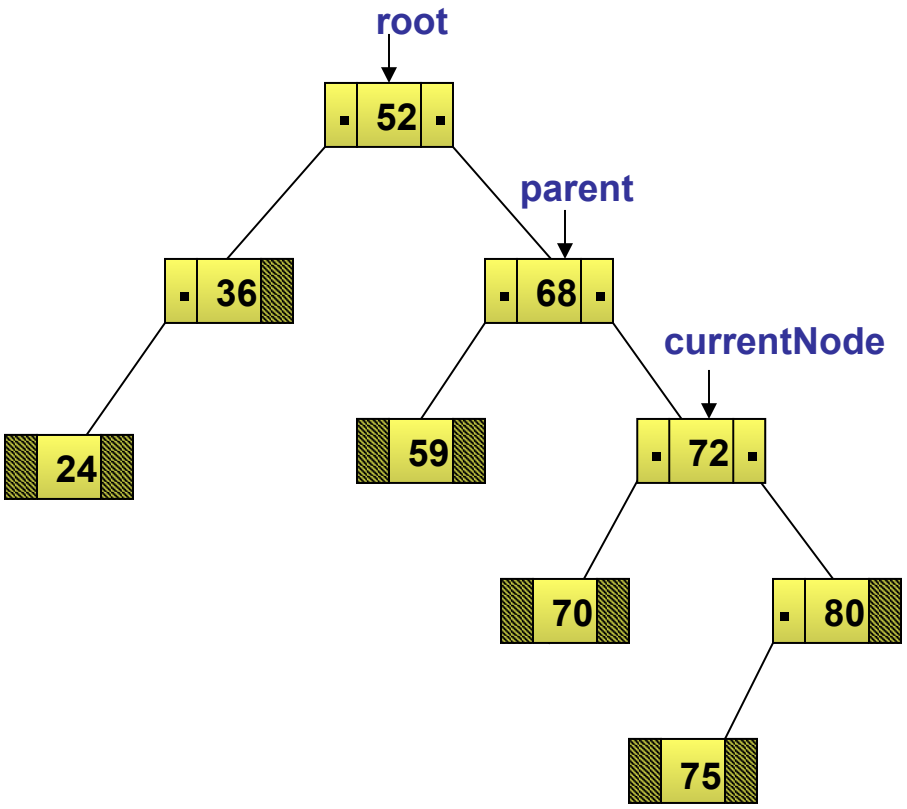
3. Replace the information held by currentNode with that of Inorder_suc.

4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.

5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

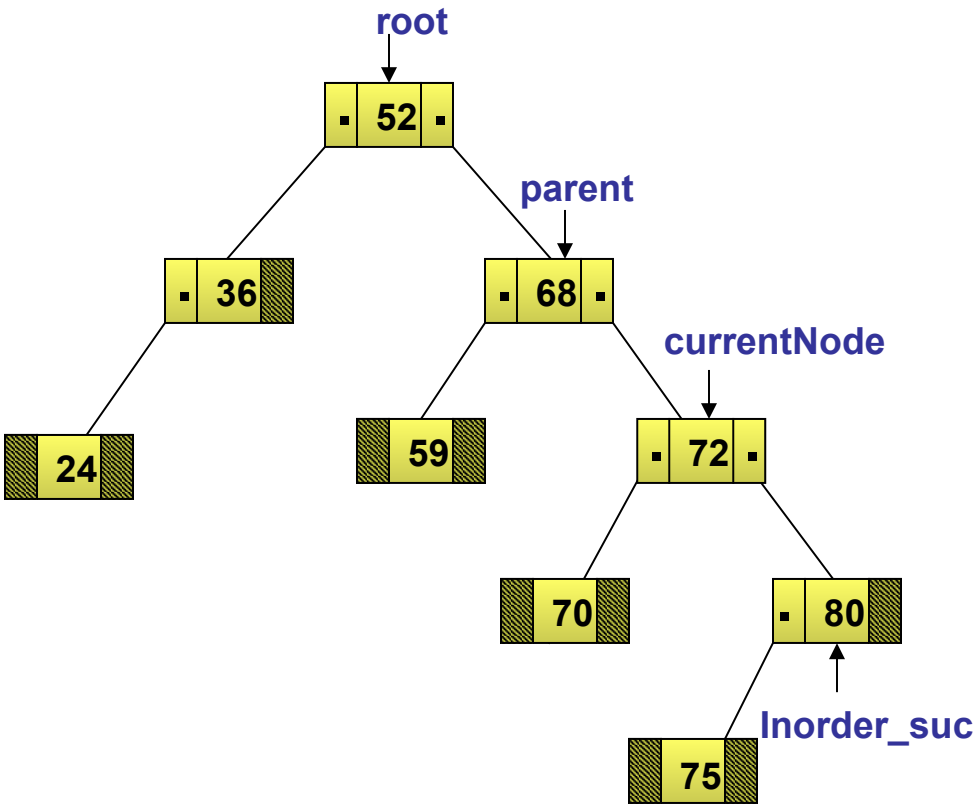
- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.
3. Replace the information held by currentNode with that of Inorder_suc.
4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.
5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

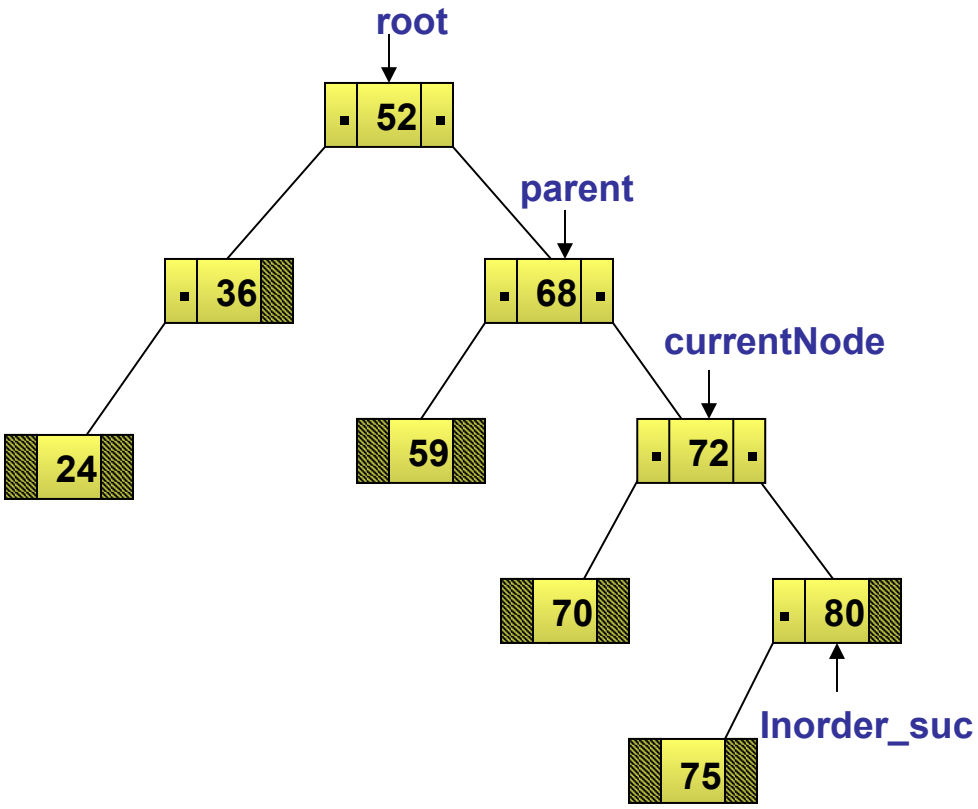
- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.
3. Replace the information held by currentNode with that of Inorder_suc.
4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.
5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

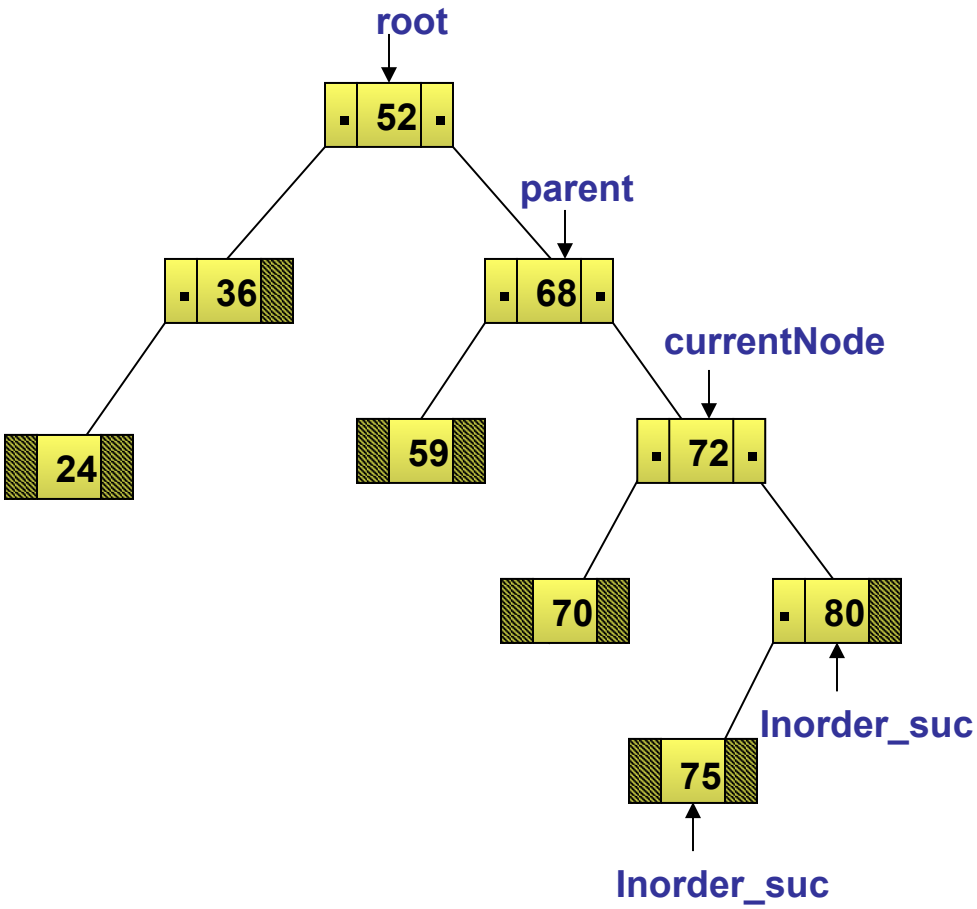
- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.
3. Replace the information held by currentNode with that of Inorder_suc.
4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.
5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

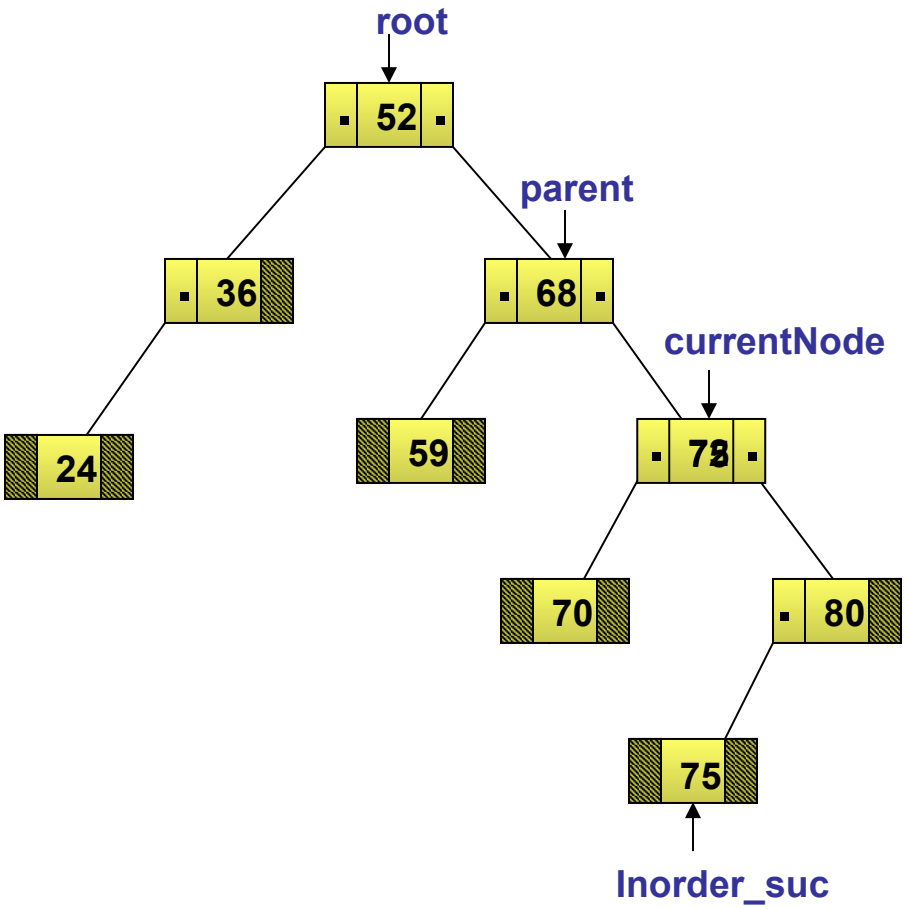
- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.
3. Replace the information held by currentNode with that of Inorder_suc.
4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.
5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.

2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.

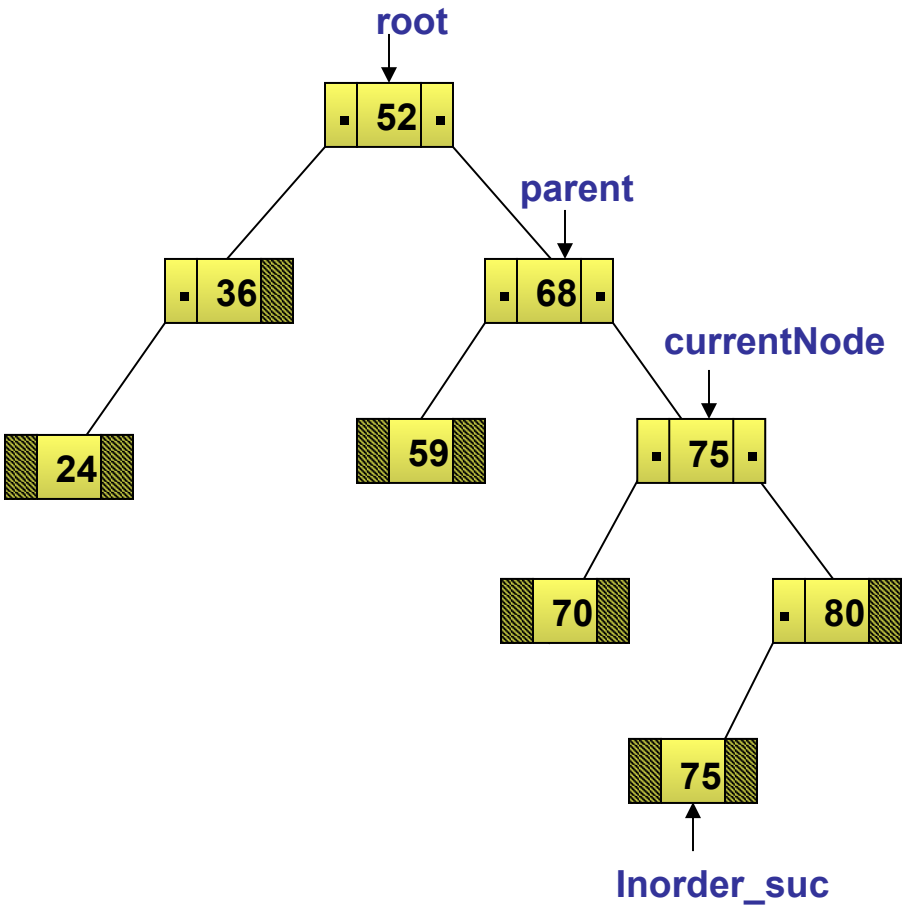
3. Replace the information held by currentNode with that of Inorder_suc.

4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.

5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

- Delete node 72



1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.

2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.

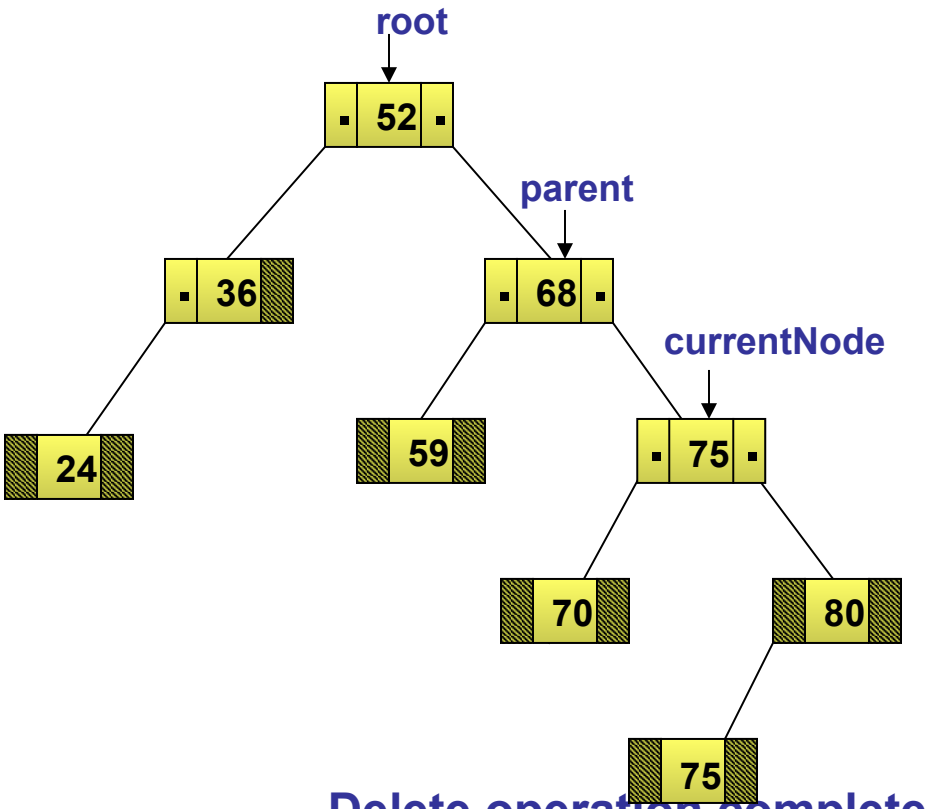
3. Replace the information held by currentNode with that of Inorder_suc.

4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.

5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deleting Nodes from a Binary Search Tree (Contd.)

- Delete node 72:



Delete operation complete

Inorder_suc

1. Locate the node to be deleted. Mark it as currentNode and its parent as parent.
2. Locate the inorder successor of currentNode. Mark it as Inorder_suc. Execute the following steps to locate Inorder_suc:
   a. Mark the right child of currentNode as Inorder_suc.
   b. Repeat until the left child of Inorder_suc becomes NULL:
      i. Make Inorder_suc point to its left child.
3. Replace the information held by currentNode with that of Inorder_suc.
4. If the node marked Inorder_suc is a leaf node:
   a. Delete the node marked Inorder_suc by using the algorithm for Case I.
5. If the node marked Inorder_suc has one child:
   a. Delete the node marked Inorder_suc by using the algorithm for Case II.

Deletion of a key from a BST

```
Algorithm:- Delete1BST (info, left, right, root, LOC, PAR)
            // When leaf node has no child or only one child
{
   1.  if ( ( LOC -> left = NULL)  and  ( LOC -> right = NULL))
           1.1 Child = NULL
       elseIf ( LOC -> left != NULL)
           1.1 Child = LOC -> left
       else
           1.1 Child = LOC -> right
   2. if ( PAR != NULL)
           2.1   if ( LOC = PAR -> left )
                       2.1.1 PAR -> left = Child
           2.1   else
                       2.1.1 PAR -> right = Child
       else
           2.1 root = Child
 }
```

Deletion of a key from a BST

```
Algorithm:- Delete2BST (info, left, right, root, LOC, PAR)
                // When leaf node has both child
{
   1. ptr1 = LOC
   2. ptr2 = LOC -> right
   3. While ( ptr2 -> left != NULL )
                3.1 ptr1 = ptr2
                3.2 ptr2 = ptr2 -> left
   4. call Delete1BST (info, left, right, root, ptr2, ptr1)
   5. If ( PAR != NULL)
                5.1   If LOC = PAR -> left
                              5.1.1 PAR -> left = ptr2
                5.1 else
                                        5.1.1 PAR -> right = ptr2
        else
                5.1 root = ptr2
   6. ptr2 -> left = LOC -> left
   7. ptr2 -> right = LOC -> right
}
```

Deletion of a key from a BST

Algorithm:- DeleteBST (info, left, right, root, key)

{

key is the value to be deleted.

1. call SearchBST (info, left, right, root, key, LOC, PAR)
   // To find the location LOC and parent PAR of the node to be deleted.
2. If (LOC = NULL) Then
   2.1 Print " Node does not exist"
   2.2 exit
3. if ((LOC -> left != NULL) and (LOC -> right != NULL))
   // when the node to be deleted has both child
   3.1 call Delete2BST (info, left, right, root, LOC, PAR)

   else
   3.1 call Delete1BST (info, left, right, root, LOC, PAR)

}

Summary

- In this session, you learned that:
  - A tree is a nonlinear data structure that represents a hierarchical relationship among the various data elements.
  - A binary tree is a specific type of tree in which each node can have a maximum of two children.
  - Binary trees can be implemented by using arrays as well as linked lists, depending upon requirement.
  - Traversal of a tree is the process of visiting all the nodes of the tree once. There are three types of traversals, namely inorder, preorder, and postorder traversal.
  - Binary search tree is a binary tree in which the value of the left child of a node is always less than the value of the node, and the value of the right child of a node is greater than the value of the node.