DATA STRUCTURES (ITPC-203)

Trees – Part II

Dr. Sanga Chaki

Department of Information Technology

Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Contents

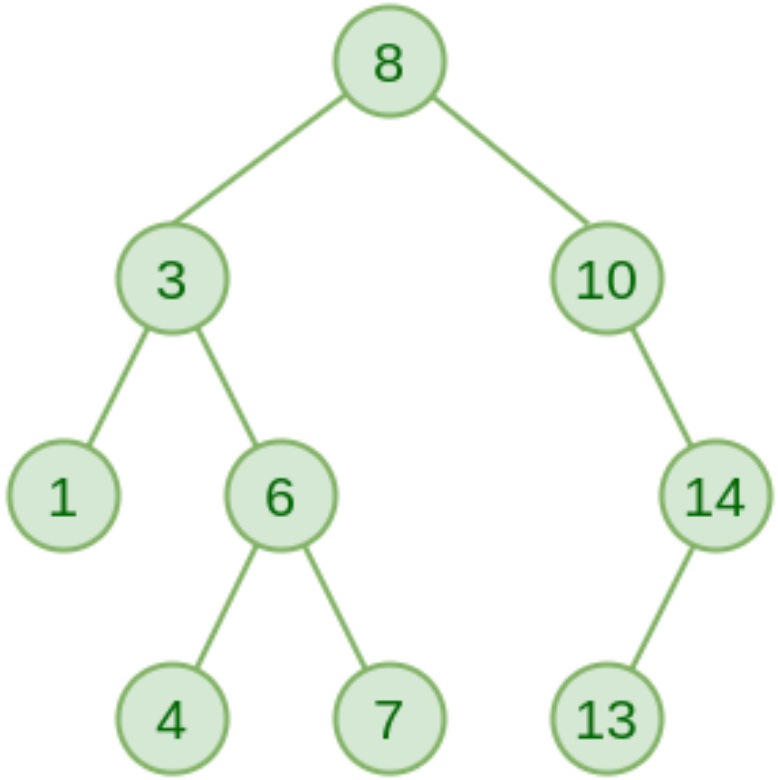
Binary Search Trees

Binary Search Tree

1. Deals with the problem: How should items in a list be stored so that an item can be easily located?
2. Primary goal: To implement a searching algorithm that finds items efficiently when the items are totally ordered.
3. We use Binary Search Trees (BST) for this purpose.
4. BSTs are a variant of Binary Trees

Binary Search Tree

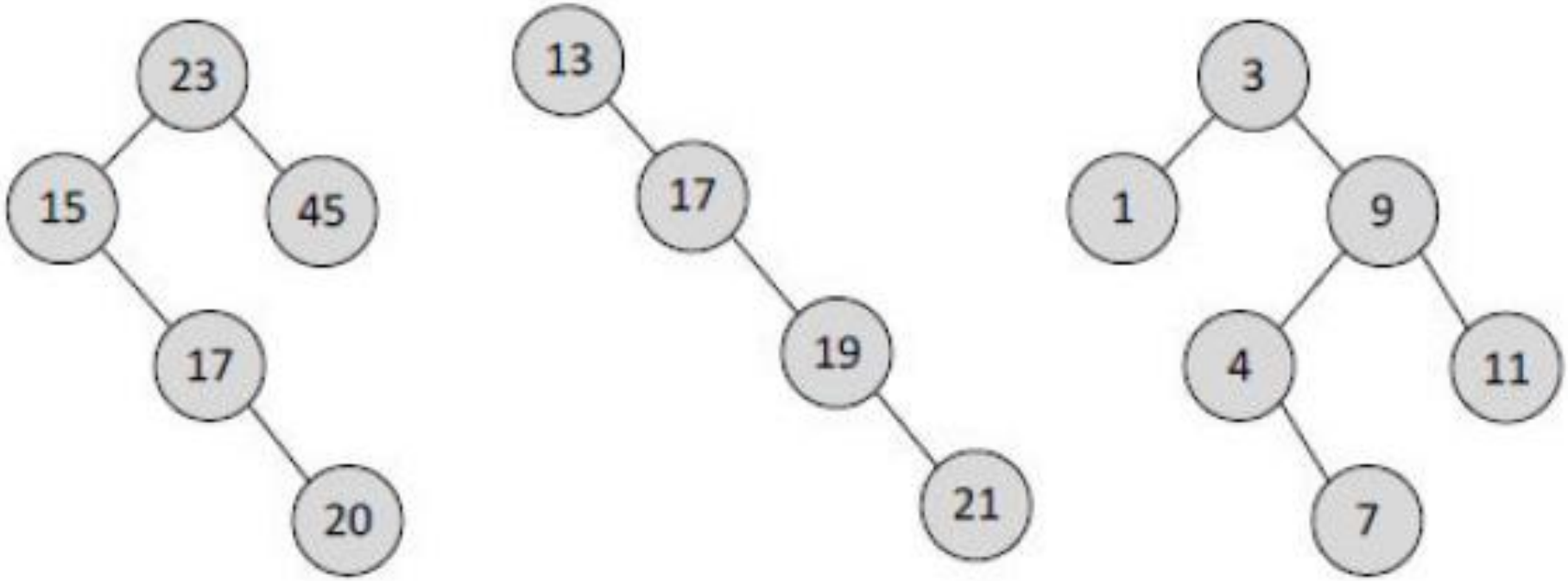
1. A binary search tree (BST) is a binary tree where each node/vertex has a comparable key (associated value) and satisfies the restriction that
   - ➢ the key in any node is larger than the keys in all nodes in that node's left subtree and
   - ➢ the key in any node is smaller than the keys in all nodes in that node's right subtree.
   - ➢ The left and right subtree each must also be a binary search tree.
2. What is the inorder traversal of this tree?



Binary Search Tree

Binary Search Tree – Operations and Examples

1. Possible operations:
   - Construction of a BST by Insertion of nodes
   - Insertion in a previously created BST
   - Deletion of a node
   - Traversal, and Searching



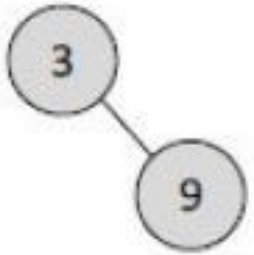
Insertion of a Node in BST

1. Basically, find where the node can be inserted so as to make the resulting tree a BST.
2. While inserting a node in a BST the value being inserted is compared with the root node.
3. A left sub-tree is taken if the value is smaller than the root node
4. A right sub-tree if it is greater or equal to the node.
5. This operation is repeated at each level till a node is found whose left or right sub-tree is empty.
6. Finally, the new node is appropriately made the left or right child of this node.

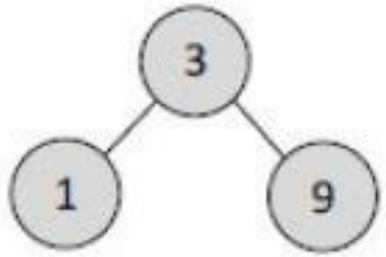
Insertion of a Node in BST

1. Example: Insert these nodes and create a BST stepwise: 3, 9, 1, 4, 7, 11

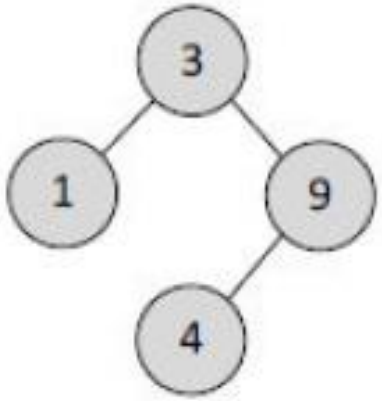
Step 1



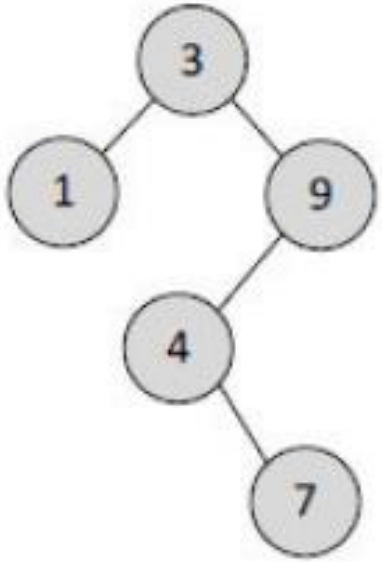
Step 2



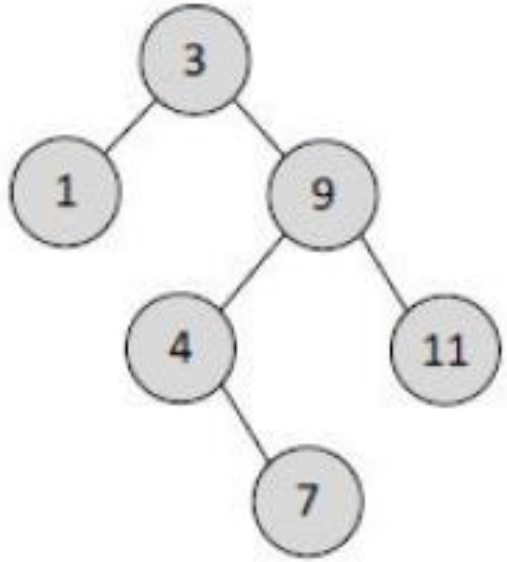
Step 3



Step 4



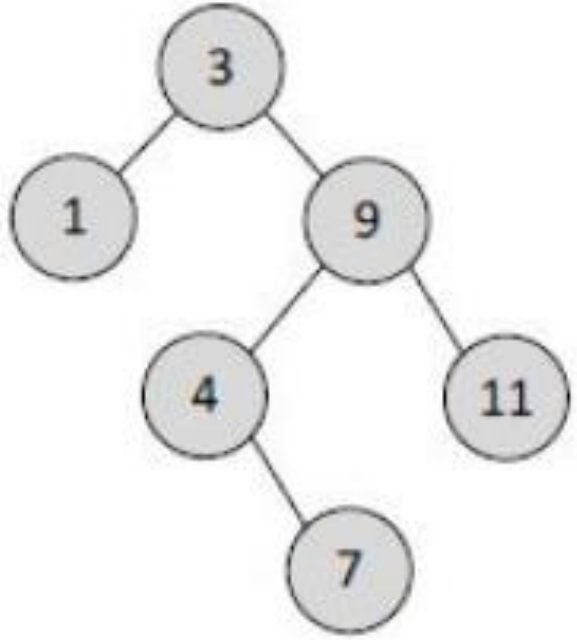
Step 5



Step 6

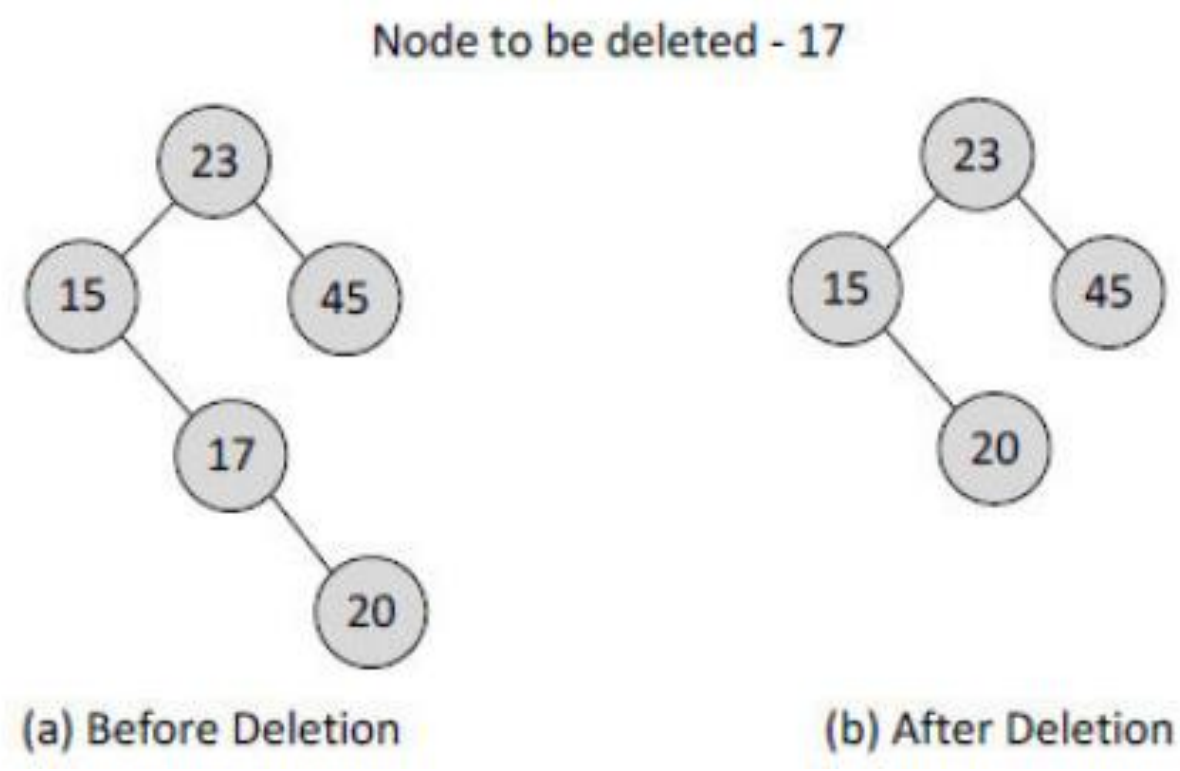
Deletion of a Node in BST

1. While deleting a node from a BST there are four possible cases that we need to consider.
2. **Case (a): Node to be deleted is absent**: If on traversing the BST the node is not found then we merely need to display the message that the node is absent. Eg: Delete node 100 in given BST
3. **Case (b): Node to be deleted has no children**: In this case since the node to be deleted has no children the memory occupied by it should be freed and either the left link or the right link of the parent of this node should be set to NULL depending upon whether the node being deleted is a left child or a right child of its parent. Eg: Delete node 7 in given BST.



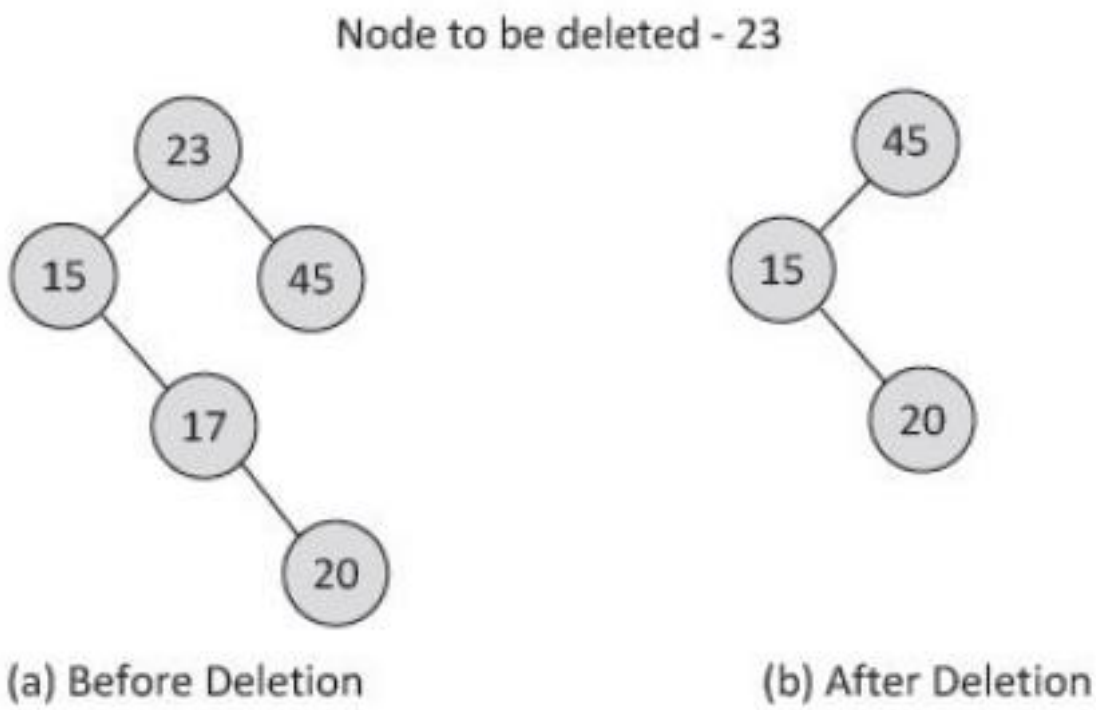
Deletion of a Node in BST

1. **Case (c): Node to be deleted has one child:** Adjust the pointer of the parent of the node to be deleted such that after deletion it points to the child of the node being deleted.



Deletion of a Node in BST

1. **Case (d): Node to be deleted has two children:** Consider node 23 to be deleted.
2. Find its **inorder successor**. The in-order successor of the node 23 is node 45.
3. The in-order successor should now be copied into the node to be deleted and a pointer should be set up pointing to the inorder successor (node 45).
4. **The in-order successor would always have one or zero child. This in-order successor should then be deleted using the same procedure as for deleting a one child or a zero child node**.

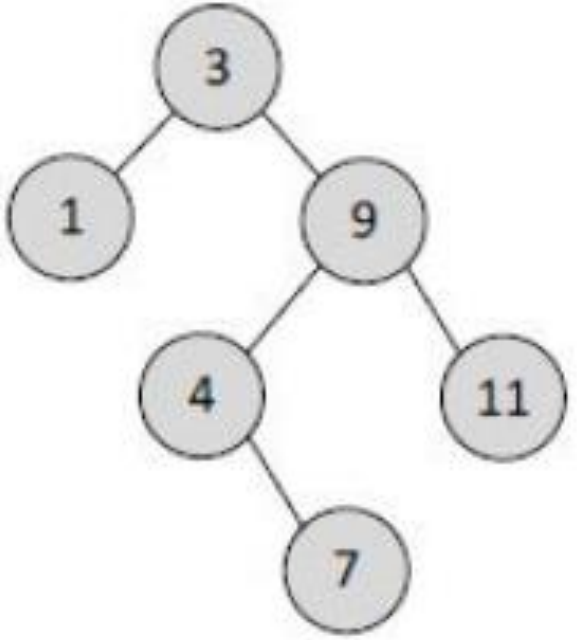


Searching of a Node in BST

1. To search any node in a binary tree, initially the value to be searched (key) is compared with the root node.
2. If they match then the search is successful.
3. If the value is greater than the root node then searching process proceeds in the right sub-tree of the root node,
4. Otherwise, it proceeds in the left sub-tree of the root node.

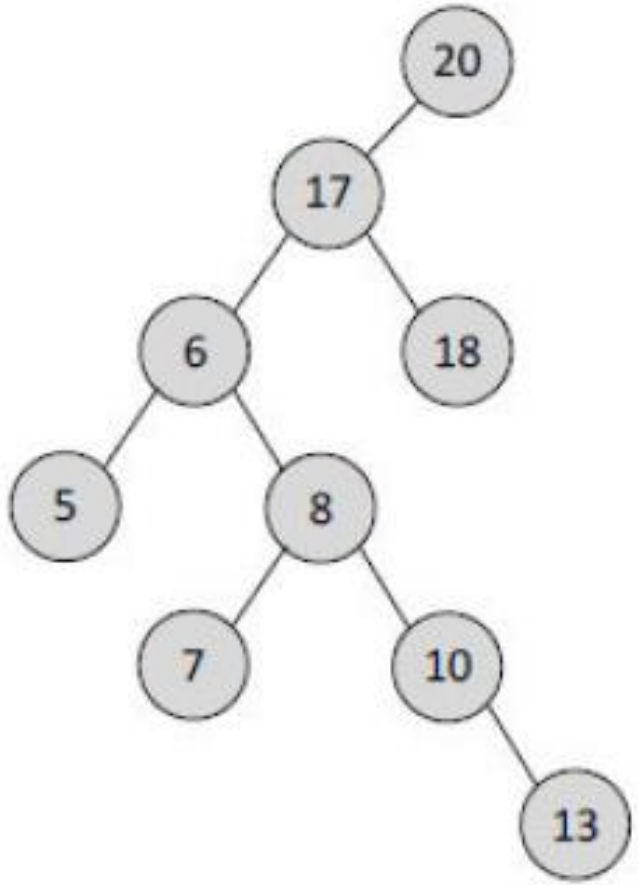
Searching of a Node in BST

1. Example: Search for 7 in the given BST
2. **Efficient because at each step, we eliminate half of the tree to be searched**.
3. We know that we have to scan only the right sub-tree of node 3, since 7 is greater than 3.
4. Likewise, when we descend down the tree and reach 9 we have to search only its left sub-tree as 7 is less than 9.
5. And so on.
6. What should be the complexity?
7. Is it better than linear and binary search of arrays?



Traversal of a BST

1. The traversal of a BST is to visit each node in the tree exactly once.
2. There are three popular methods of BST traversal—
   - inorder traversal:
     - (1) Traverse the left sub-tree in in-order
     - (2) Visit the root
     - (3) Traverse the right sub-tree in in-order
   - pre-order traversal:
     - (1) Visit the root
     - (2) Traverse the left sub-tree in pre-order
     - (3) Traverse the right sub-tree in pre-order
   - post-order traversal:
     - (1) Traverse the left sub-tree in post-order
     - (2) Traverse the right sub-tree in post-order
     - (3) Visit the root



Inorder : 5, 6, 7, 8, 10, 13, 17, 18, 20

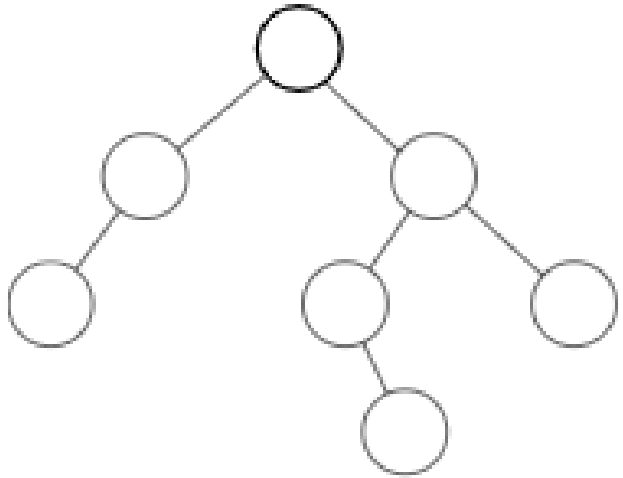
Prorder : 20, 17, 6, 5, 8, 7, 10, 13, 18

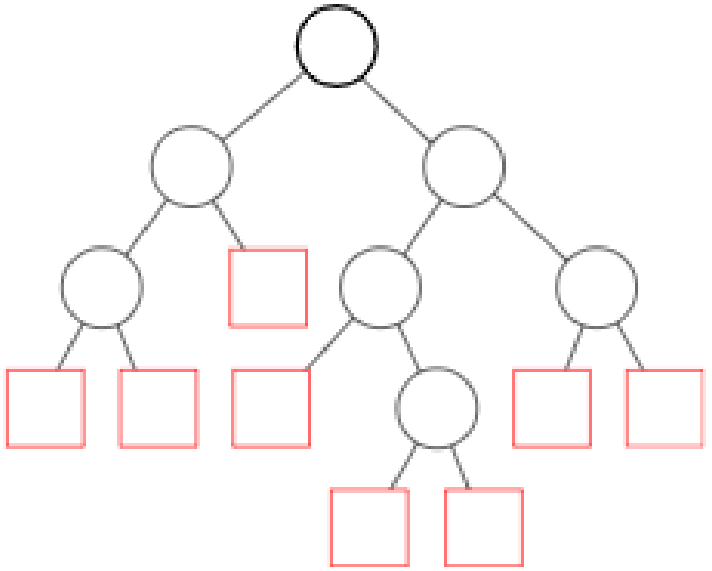
Postorder : 5, 7, 13, 10, 8, 6, 18, 17, 20

Extended Binary Trees

Extended Binary Trees or 2-tree

1. A binary tree in which special nodes are added wherever a null subtree was present in the original tree
   - so that each node in the original tree (except the root node) has degree three.
2. So, each node now either has 2 or 0 children
3. Nodes of original tree = Internal nodes (circles)
4. New nodes added to create extended binary tree = External nodes (squares)
5. It is useful for representation of algebraic expressions.





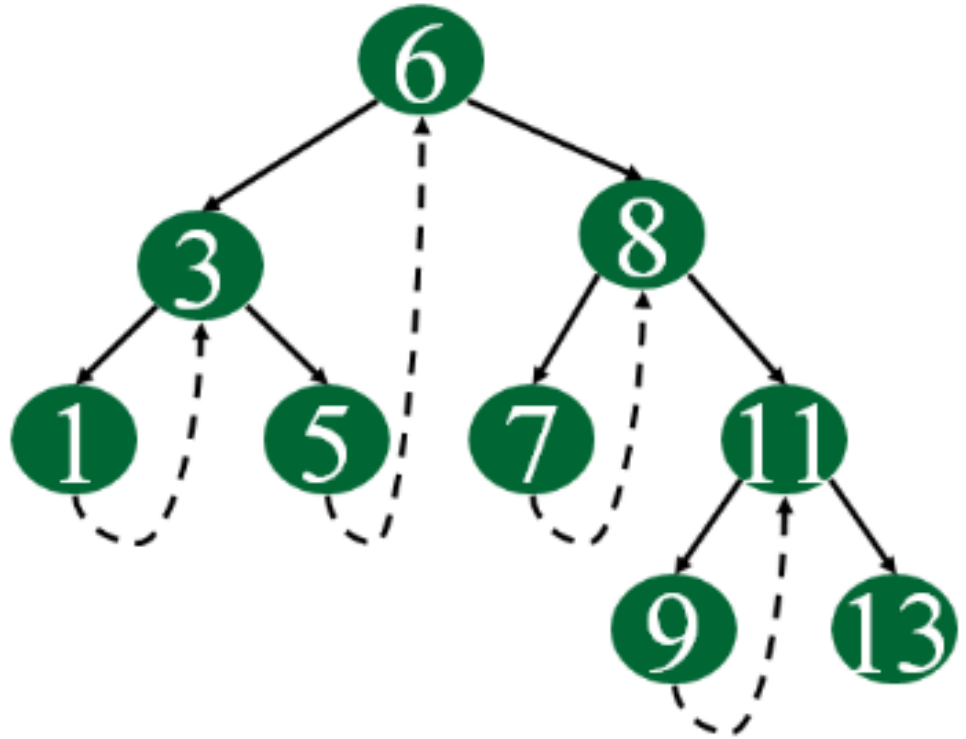
Some Properties of Binary Trees

1. If a tree has n nodes, the number of branches it has = n-1
2. Except the root node, every node in a tree has exactly 1 parent
3. Any two nodes of a tree are connected by only once single path
4. For a binary tree of height h, the maximum number of nodes can be $2^{h+1} -1$
5. Any binary tree with n internal nodes has n+1 external nodes.

Threaded Binary Trees

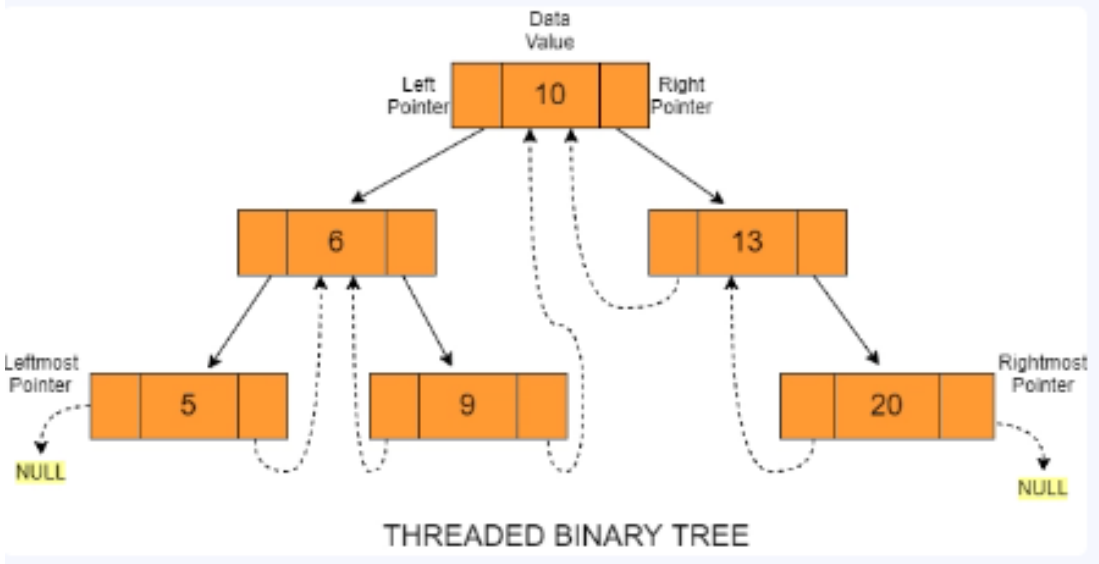
Threaded Binary Trees

1. A threaded binary tree is a type of binary tree data structure where the empty left and right child pointers in a binary tree are replaced with threads that link nodes directly to their in-order predecessor or successor
2. Inorder traversal of a Binary tree can generally be done using recursion or with the use of a auxiliary stack.
3. Threaded binary trees make inorder traversal faster without using stack/recursion, using the threads.
4. Inorder sequence: 1, 3, 5, 6, 7, 8, 9, 11, 13
5. Threaded lines – threads to inorder successor



Threaded Binary Trees

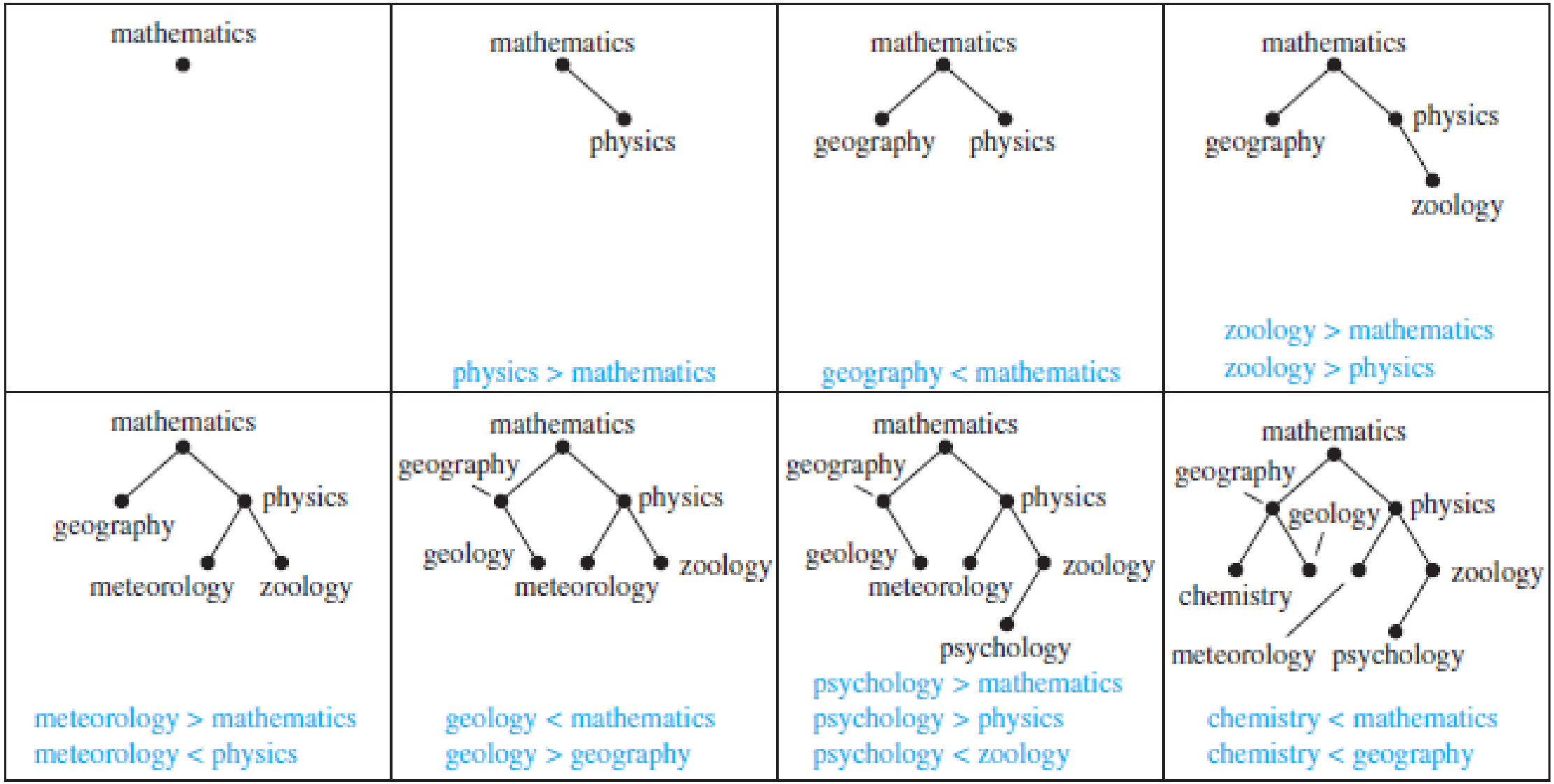
1. There are two types of threaded binary trees.
   a) Single Threaded: Where a NULL right pointers is made to point to the inorder successor (if successor exists)
   b) Double Threaded: Where both left and right NULL pointers are made to point to inorder predecessor and inorder successor respectively.
2. The predecessor threads are useful for reverse inorder traversal and postorder traversal.
3. The threads are also useful for fast accessing ancestors of a node.



Extras Examples

Extra: Creating a Binary Search Tree

1. Example: Form a binary search tree for the words mathematics, physics, geography, zoology, meteorology, geology, psychology, and chemistry (using alphabetical order).



Extra: Binary Search Tree

ALGORITHM 1 Locating an Item in or Adding an Item to a Binary Search Tree.

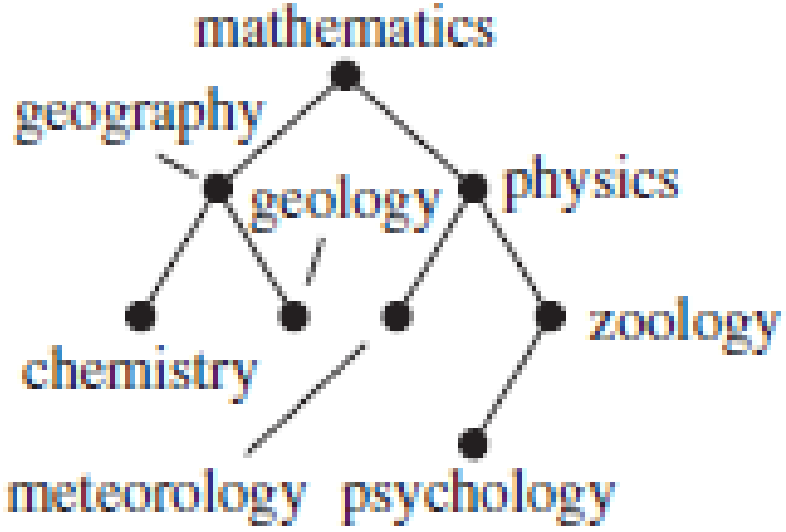
procedure *insertion*(T: binary search tree, x: item)
$v :=$ root of T
{a vertex not present in T has the value *null*}
while $v \neq$ *null* and *label*(v) $\neq x$
    if $x <$ *label*(v) **then**
        if left child of $v \neq$ *null* **then** $v :=$ left child of v
        else add *new vertex* as a left child of v and set $v :=$ *null*
    else
        if right child of $v \neq$ *null* **then** $v :=$ right child of v
        else add *new vertex* as a right child of v and set $v :=$ *null*
if root of $T =$ *null* **then** add a vertex v to the tree and label it with x
else if v is null or *label*(v) $\neq x$ **then** label *new vertex* with x and let v be this new vertex
return v {$v =$ location of x}

Extra: Algorithm – Stepwise Explanation

1. Algorithm 1 is a procedure that locates an item x in a binary search tree if it is present, and adds a new vertex with x as its key if x is not present.
2. In the pseudocode, v is the vertex currently under examination and label(v) represents the key of this vertex.
3. The algorithm begins by examining the root.
4. If x equals the key of v, then the algorithm has found the location of x and terminates;
5. if x is less than the key of v, we move to the left child of v and repeat the procedure;
6. if x is greater than the key of v, we move to the right child of v and repeat the procedure.
7. If at any step we attempt to move to a child that is not present, we know that x is not present in the tree, and we add a new vertex as this child with x as its key.

Extra: Binary Search Tree

1. **Example: Use Algorithm 1 to insert the word oceanography into the binary search tree in the previous example**
2. Algorithm 1 begins with v, the vertex under examination, equal to the root of T , so label(v) = mathematics.
3. Because v ≠ null and label(v) = mathematics < oceanography, we next examine the right child of the root.
4. This right child exists, so we set v, the vertex under examination, to be this right child.
5. At this step we have v ≠ null and label(v) = physics > oceanography, so we examine the left child of v.



Extra: Binary Search Tree

1. This left child exists, so we set v, the vertex under examination, to this left child (which at this point is the vertex with the key meteorology).
2. At this step, we also have v ≠ null and label(v) = meteorology < oceanography, so we try to examine the right child of v.
3. However, this right child does not exist, so we add a new vertex as the right child of v and we set v := null.
4. We now exit the while loop because v = null.
5. Because the root of T is not null and v = null, we use the else if statement at the end of the algorithm to label our new vertex with the key oceanography.