



# Tree concepts and Binary Tree

*Data Structures and Algorithms*

**Dept. Computer Science**

*Faculty of Computer Science and Engineering  
Ho Chi Minh University of Technology, VNU-HCM*

# Overview

## ① Basic Tree Concepts

## ② Binary Trees

## ③ Expression Trees

## ④ Binary Search Trees

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees



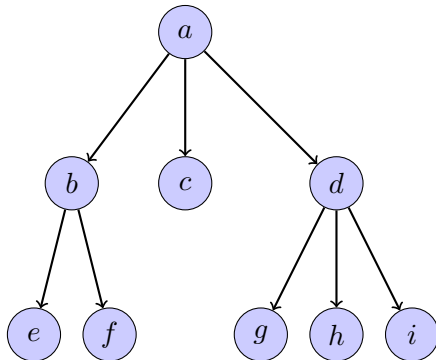
# Basic Tree Concepts

# Basic Tree Concepts



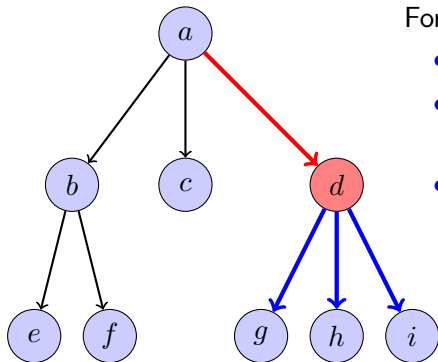
## Definition

A **tree** (cây) consists of a **finite set of elements**, called **nodes** (nút), and a **finite set of directed lines**, called **branches** (nhánh), that connect the nodes.



# Basic Tree Concepts

- **Degree of a node** (Bậc của nút): the number of branches associated with the node.
- **Indegree branch** (Nhánh vào): directed branch toward the node.
- **Outdegree branch** (Nhánh ra): directed branch away from the node.



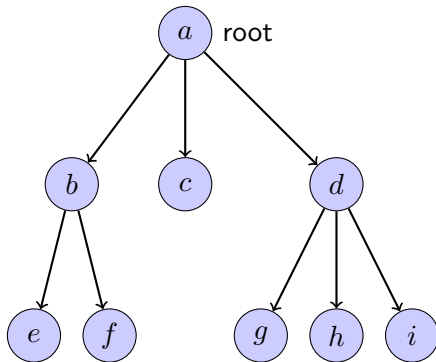
For the node  $d$ :

- **Degree** = 4
- **Indegree branches:**  $ad$   
→ indegree = 1
- **Outdegree branches:**  $dg, dh, di$   
→ outdegree = 3



# Basic Tree Concepts

- The first node is called the **root**.
- indegree of the root = 0
- Except the root, the indegree of a node = 1
- outdegree of a node = 0 or 1 or more.



# Basic Tree Concepts

## Terms

- A **root** (nút gốc) is the first node with an indegree of zero.
- A **leaf** (nút lá) is any node with an outdegree of zero.
- A **internal node** (nút nội) is not a root or a leaf.
- A **parent** (nút cha) has an outdegree greater than zero.
- A **child** (nút con) has an indegree of one.  
→ a internal node is both a parent of a node and a child of another one.
- **Siblings** (nút anh em) are two or more nodes with the same parent.
- For a given node, an **ancestor** is any node in the path from the root to the node.
- For a given node, an **descendent** is any node in the paths from the node to a leaf.





## Terms

- A **path** (đường đi) is a sequence of nodes in which each node is adjacent to the next one.
- The **level** (bậc) of a node is its distance from the root.  
→ Siblings are always at the same level.
- The **height** (độ cao) of a tree is the level of the leaf in the longest path from the root plus 1.
- A **subtree** (cây con) is any connected structure below the root.

Basic Tree Concepts

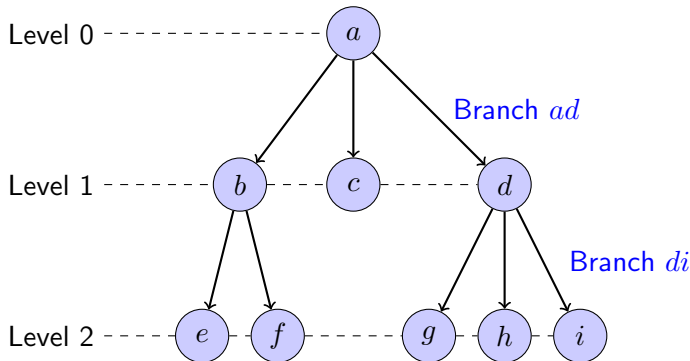
Binary Trees

Expression Trees

Binary Search Trees



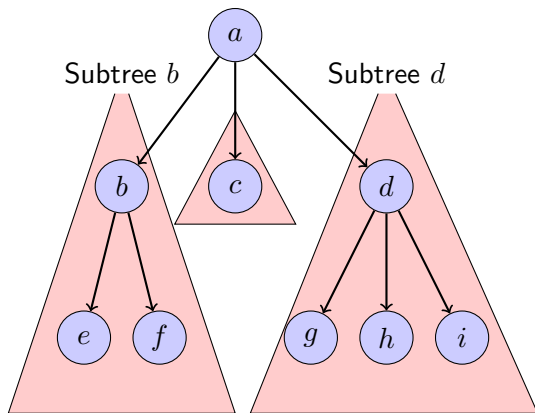
# Basic Tree Concepts



- Parents:  $a, b, d$
- Children:  
 $b, c, d, e, f, g, h, i$
- Leaves:  $c, e, f, g, h, i$
- Internal nodes:  $b, d$
- Siblings:  
 $\{b, c, d\}, \{e, f\}, \{g, h, i\}$
- Height = 3



# Basic Tree Concepts



Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

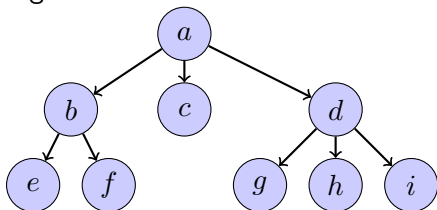
Binary Trees

Expression Trees

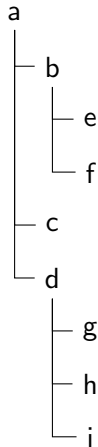
Binary Search Trees

# Tree representation

- organization chart



- parenthetical listing  
 $a (b (e f) c d (g h i))$
- indented list



# Applications of Trees

- Representing hierarchical data
- Storing data in a way that makes it easily searchable (ex: binary search tree)
- Representing sorted lists of data
- Network routing algorithms

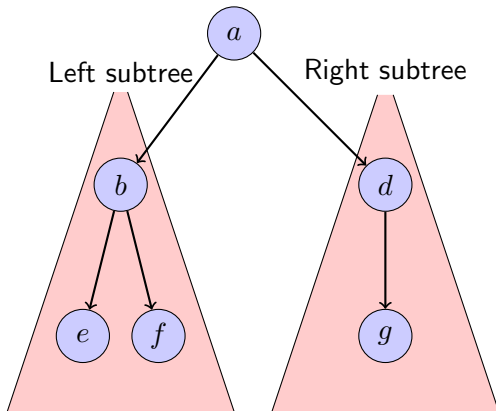




# Binary Trees

# Binary Trees

A binary tree node cannot have more than two subtrees.



## Binary Trees Properties

- To store  $N$  nodes in a binary tree:
  - The minimum height:  $H_{min} = \lfloor \log_2 N \rfloor + 1$  or  $H_{min} = \lceil \log_2(N + 1) \rceil$
  - The maximum height:  $H_{max} = N$
- Given a height of the binary tree,  $H$ :
  - The minimum number of nodes:  $N_{min} = H$
  - The maximum number of nodes:  $N_{max} = 2^H - 1$

### Balance

The **balance factor** of a binary tree is the difference in height between its left and right subtrees.

$$B = H_L - H_R$$

### Balanced tree:

- balance factor is 0, -1, or 1
- subtrees are **balanced**

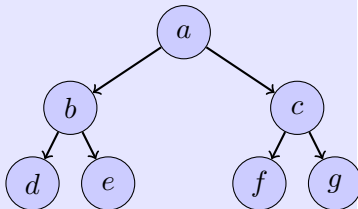


# Binary Trees Properties

## Complete tree

$$N = N_{max} = 2^H - 1$$

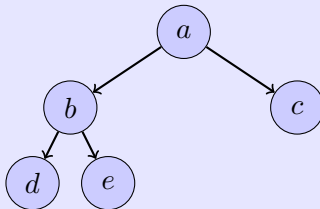
The last level is full.



## Nearly complete tree

$$H = H_{min} = \lfloor \log_2 N \rfloor + 1$$

Nodes in the last level are on the left.

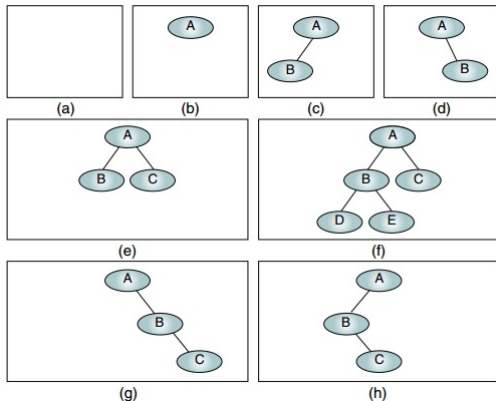




# Binary Tree Structure

## Definition

A **binary tree** is either empty, or it consists of a node called **root** together with two binary trees called the **left** and the **right** subtree of the root.

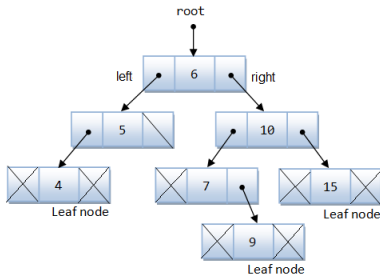


# Binary Tree Structure: Linked implementation

```
node
  data <dataType>
  left <pointer>
  right <pointer>
end node
```

```
// General dataType:
dataType
  key <keyType>
  field1 <...>
  field2 <...>
  ...
  fieldn <...>
end dataType
```

```
binaryTree
  root <pointer>
end binaryTree
```



Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

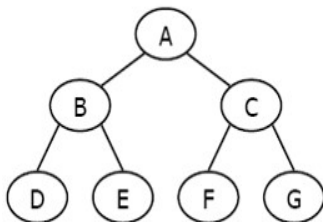
Binary Trees

Expression Trees

Binary Search Trees

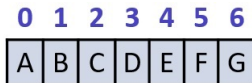
# Binary Tree Structure: Array-based implementation

Suitable for complete tree, nearly complete tree.



**Hình:** Conceptual

```
binaryTree  
  data <array of dataType>  
end binaryTree
```



**Hình:** Physical



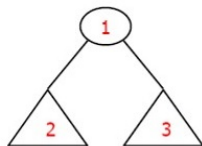
# Binary Tree Traversals

- **Depth-first traversal** (duyệt theo chiều sâu): the processing proceeds along a path from the root through one child to the most distant descendent of that first child before processing a second child, i.e. processes all of the descendents of a child before going on to the next child.
- **Breadth-first traversal** (duyệt theo chiều rộng): the processing proceeds horizontally from the root to all of its children, then to its children's children, i.e. each level is completely processed before the next level is started.

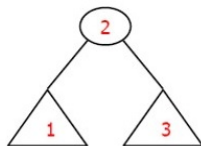


# Depth-first traversal

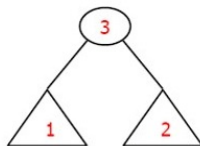
- Preorder traversal
- Inorder traversal
- Postorder traversal



PreOrder  
NLR



InOrder  
LNR

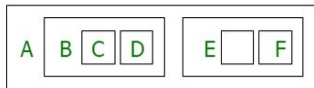
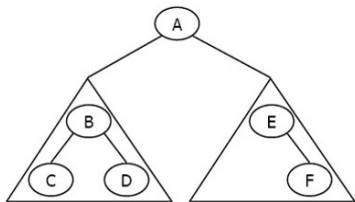


PostOrder  
LRN

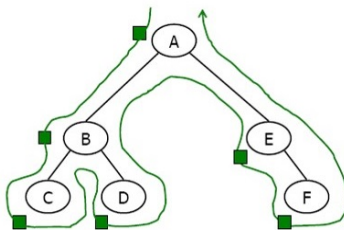


# Preorder traversal (NLR)

In the preorder traversal, the root is processed first, before the left and right subtrees.



Processing order



Walking order

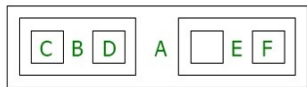
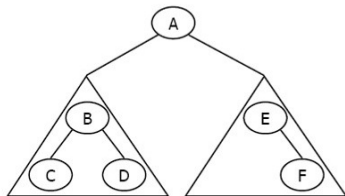
## Preorder traversal (NLR)

```
1 Algorithm preOrder(val root <pointer>)
2 Traverse a binary tree in node-left-right sequence.
3 Pre: root is the entry node of a tree or subtree
4 Post: each node has been processed in order
5 if root is not null then
6     | process(root)
7     | preOrder(root->left)
8     | preOrder(root->right)
9 end
10 Return
11 End preOrder
```

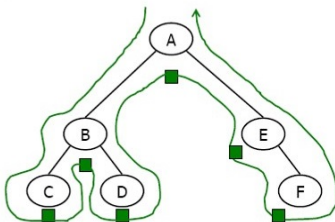


# Inorder traversal (LNR)

In the inorder traversal, the root is processed between its subtrees.



Processing order



Walking order



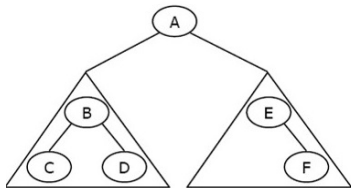
## Inorder traversal (LNR)

```
1 Algorithm inOrder(val root <pointer>)
2 Traverse a binary tree in left-node-right sequence.
3 Pre: root is the entry node of a tree or subtree
4 Post: each node has been processed in order
5 if root is not null then
6     |   inOrder(root->left)
7     |   process(root)
8     |   inOrder(root->right)
9 end
10 Return
11 End inOrder
```

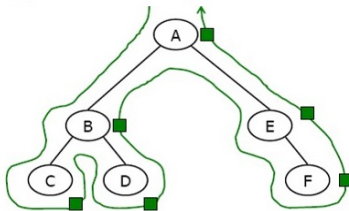


## Postorder traversal (LRN)

In the postorder traversal, the root is processed after its subtrees.



Processing order



Walking order

## Postorder traversal (LRN)

```
1 Algorithm postOrder(val root <pointer>)  
2   Traverse a binary tree in left-right-node sequence.  
3   Pre: root is the entry node of a tree or subtree  
4   Post: each node has been processed in order  
5   if root is not null then  
6     |   postOrder(root->left)  
7     |   postOrder(root->right)  
8     |   process(root)  
9   end  
10  Return  
11 End postOrder
```

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

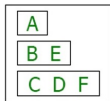
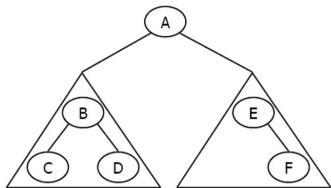
Binary Trees

Expression Trees

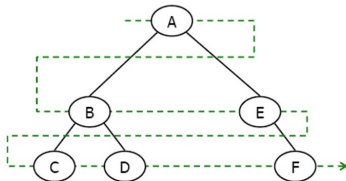
Binary Search Trees

# Breadth-First Traversals

In the breadth-first traversal of a binary tree, we process all of the children of a node before proceeding with the next level.



Processing order



Walking order

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

# Breadth-First Traversals

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

- 1 **Algorithm** breadthFirst(val root <pointer>)
- 2 Process tree using breadth-first traversal.
- 3 **Pre:** root is node to be processed
- 4 **Post:** tree has been processed
- 5 currentNode = root
- 6 bfQueue = createQueue()

## Breadth-First Traversals

```
1 while currentNode not null do
2     process(currentNode)
3     if currentNode->left not null then
4         enqueue(bfQueue, currentNode->left)
5     end
6     if currentNode->right not nul then
7         enqueue(bfQueue, currentNode->right)
8     end
9     if not emptyQueue(bfQueue) then
10         currentNode = dequeue(bfQueue)
11     else
12         currentNode = NULL
13     end
14 end
15 destroyQueue(bfQueue)
16 End breadthFirst
```

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

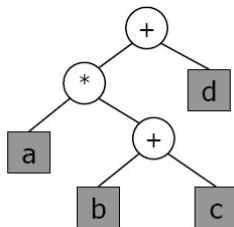
Binary Search Trees



# Expression Trees

# Expression Trees

- Each leaf is an **operand**
- The root and internal nodes are **operators**
- Sub-trees are **sub-expressions**

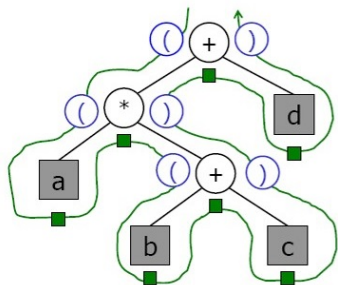


$$a * (b + c) + d$$





# Infix Expression Tree Traversal



$((a * (b + c)) + d)$

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

# Infix Expression Tree Traversal

```
1 Algorithm infix(val tree <pointer>)
2 Print the infix expression for an expression tree.
3 Pre: tree is a pointer to an expression tree
4 Post: the infix expression has been printed
5 if tree not empty then
6     if tree->data is an operand then
7         | print (tree->data)
8     else
9         | print (open parenthesis)
10        | infix (tree->left)
11        | print (tree->data)
12        | infix (tree->right)
13        | print (close parenthesis)
14    end
15 end
16 End infix
```

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

# Postfix Expression Tree Traversal

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

```
1 Algorithm postfix(val tree <pointer>)
2 Print the postfix expression for an expression tree.
3 Pre: tree is a pointer to an expression tree
4 Post: the postfix expression has been printed
5 if tree not empty then
6     | postfix (tree->left)
7     | postfix (tree->right)
8     | print (tree->data)
9 end
10 End postfix
```

# Prefix Expression Tree Traversal

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

```
1 Algorithm prefix(val tree <pointer>)  
2 Print the prefix expression for an expression tree.  
3 Pre: tree is a pointer to an expression tree  
4 Post: the prefix expression has been printed  
5 if tree not empty then  
6 |   print (tree->data)  
7 |   prefix (tree->left)  
8 |   prefix (tree->right)  
9 end  
10 End prefix
```



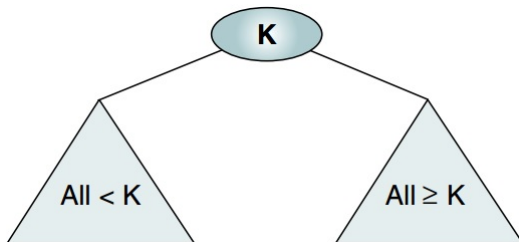
# Binary Search Trees

# Binary Search Trees

## Definition

A **binary search tree** is a binary tree with the following properties:

- 1 All items in the left subtree are less than the root.
- 2 All items in the right subtree are greater than or equal to the root.
- 3 Each subtree is itself a binary search tree.



# Valid Binary Search Trees

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

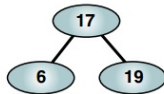
Binary Trees

Expression Trees

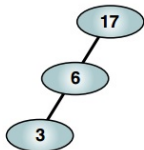
Binary Search Trees



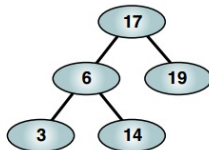
(a)



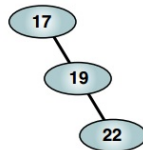
(b)



(c)



(d)



(e)

# Invalid Binary Search Trees

Tree concepts

Dept. Computer  
Science

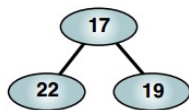


Basic Tree Concepts

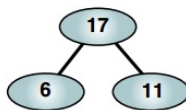
Binary Trees

Expression Trees

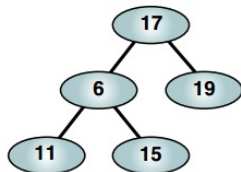
Binary Search Trees



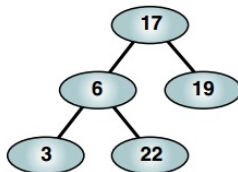
(a)



(b)



(c)



(d)



# Binary Search Tree (BST)

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

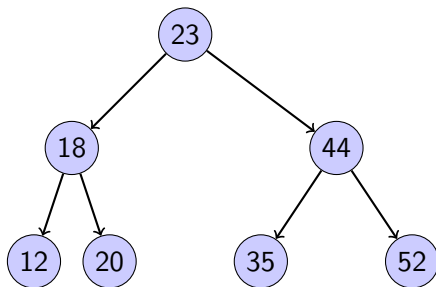
Binary Trees

Expression Trees

Binary Search Trees

- BST is one of implementations for ordered list.
- In BST we can search quickly (as with **binary search** on a contiguous list).
- In BST we can make **insertions and deletions quickly** (as with a linked list).

# Binary Search Tree Traversals



- Preorder traversal: 23, 18, 12, 20, 44, 35, 52
- Postorder traversal: 12, 20, 18, 35, 52, 44, 23
- Inorder traversal: **12, 18, 20, 23, 35, 44, 52**

The **inorder traversal** of a binary search tree produces an ordered list.



# Binary Search Tree Search

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Find Smallest Node

```
1 Algorithm findSmallestBST(val root <pointer>)  
2 This algorithm finds the smallest node in a BST.  
3 Pre: root is a pointer to a nonempty BST or  
   subtree  
4 Return address of smallest node  
5 if root->left null then  
6   |   return root  
7 end  
8 return findSmallestBST(root->left)  
9 End findSmallestBST
```

# Binary Search Tree Search

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Find Largest Node

```
1 Algorithm findLargestBST(val root <pointer>)
2 This algorithm finds the largest node in a BST.
3 Pre: root is a pointer to a nonempty BST or
   subtree
4 Return address of largest node returned
5 if root->right is null then
6   |   return root
7 end
8 return findLargestBST(root->right)
9 End findLargestBST
```



## Recursive Search

- 1 **Algorithm** searchBST(val root <pointer>, val  
target <keyType>)
- 2 Search a binary search tree for a given value.
- 3 **Pre:** root is the root to a binary tree or subtree
- 4 target is the key value requested
- 5 **Return** the node address if the value is found
- 6 null if the node is not in the tree

# Binary Search

## Recursive Search

```
1 if root is null then  
2   |   return null  
3 end  
  
4 if target < root->data.key then  
5   |   return searchBST(root->left, target)  
6 else if target > root->data.key then  
7   |   return searchBST(root->right, target)  
8 else  
9   |   return root  
10 end  
  
11 End searchBST
```

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees



## Iterative Search

- 1 **Algorithm** iterativeSearchBST(val root  
    <pointer>, val target <keyType>)
- 2 Search a binary search tree for a given value using  
    a loop.
- 3 **Pre:** root is the root to a binary tree or subtree
- 4 target is the key value requested
- 5 **Return** the node address if the value is found
- 6 null if the node is not in the tree

# Binary Search



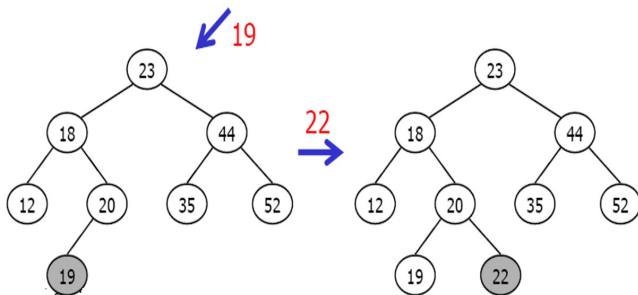
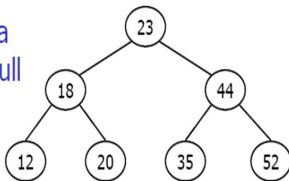
## Iterative Search

```
1 while (root is not NULL) AND (root->data.key !=  
   target) do  
2   |   if target < root->data.key then  
3   |       root = root->left  
4   |   else  
5   |       root = root->right  
6   |   end  
7 end  
8 return root  
9 End iterativeSearchBST
```



# Insert Node into BST

Taking place at a  
node having a null  
branch



All BST insertions take place at a leaf or a leaflike node (a node that has only one null branch).

## Insert Node into BST: Iterative Insert

- 1 **Algorithm** `iterativeInsertBST(ref root <pointer>, val new <pointer>)`
- 2 Insert node containing new data into BST using iteration.
- 3 **Pre:** root is address of first node in a BST
- 4 new is address of node containing data to be inserted
- 5 **Post:** new node inserted into the tree

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Insert Node into BST: Iterative Insert

```
1  if root is null then
2      |   root = new
3  else
4      |   pWalk = root
5      |   while pWalk not null do
6          |       parent = pWalk
7          |       if new->data.key < pWalk->data.key then
8              |       pWalk = pWalk->left
9          |       else
10             |       pWalk = pWalk->right
11             end
12         end
13         if new->data.key < parent->data.key then
14             |       parent->left = new
15         else
16             |       parent->right = new
17         end
18 end
```

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Insert Node into BST: Recursive Insert

- 1 **Algorithm** recursiveInsertBST(ref root <pointer>, val new <pointer>)
- 2 Insert node containing new data into BST using recursion.
- 3 **Pre:** root is address of current node in a BST
- 4 new is address of node containing data to be inserted
- 5 **Post:** new node inserted into the tree

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Insert Node into BST: Recursive Insert

```
1  if root is null then  
2    |   root = new  
3  else  
4    |   if new->data.key < root->data.key then  
5    |     recursiveInsertBST(root->left, new)  
6    |   else  
7    |     recursiveInsertBST(root->right, new)  
8    |   end  
9  end  
10 Return  
11 End recursiveInsertBST
```

Tree concepts

Dept. Computer  
Science



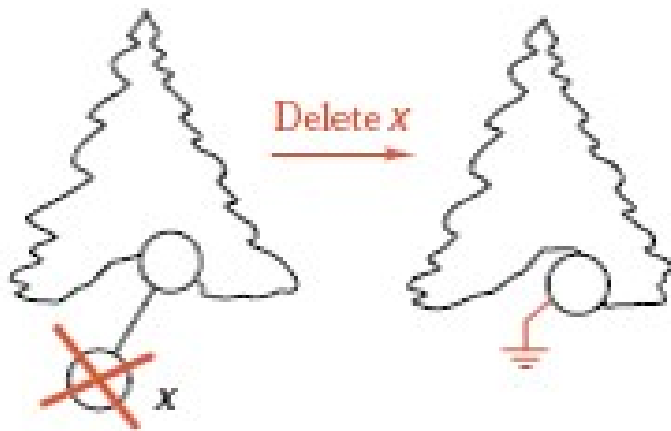
Basic Tree Concepts

Binary Trees

Expression Trees

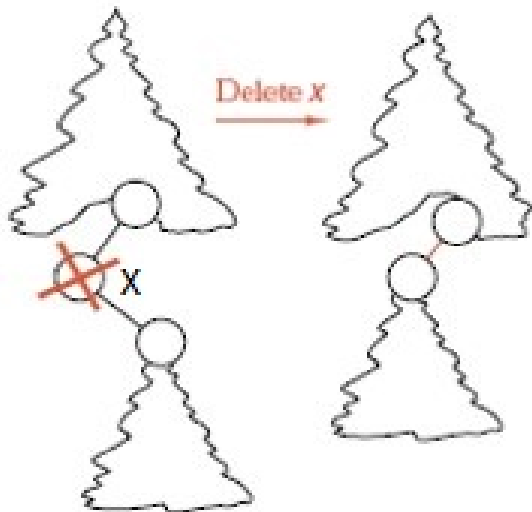
Binary Search Trees

## Delete node from BST



**Deletion of a leaf:** Set the deleted node's parent link to NULL.

## Delete node from BST



Deletion of a node having only right subtree or left subtree:  
Attach the subtree to the deleted node's parent.

# Delete node from BST

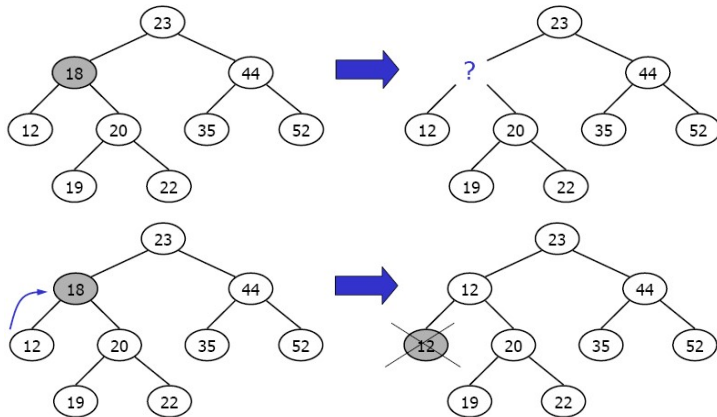
## Delete node having both subtrees:

Replace the deleted node by its predecessor or by its successor, recycle this node instead.





# Delete node from BST



Using largest node in the left subtree

Tree concepts

Dept. Computer  
Science



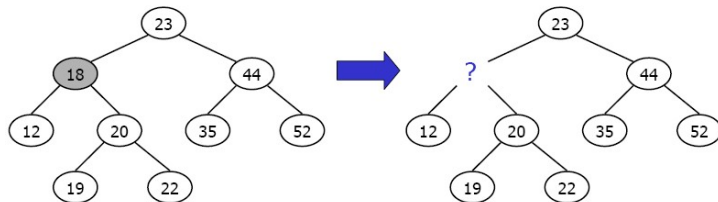
Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

# Delete node from BST



Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Delete node from BST

- 1 **Algorithm** deleteBST(ref root <pointer>, val dltKey <keyType>)
- 2 Deletes a node from a BST.
- 3 **Pre:** root is pointer to tree containing data to be deleted
- 4 dltKey is key of node to be deleted
- 5 **Post:** node deleted and memory recycled
- 6 if dltKey not found, root unchanged
- 7 **Return** true if node deleted, false if not found



# Delete node from BST

```
1 if root is null then
2   |   return false
3 end
4 if dltKey < root->data.key then
5   |   return deleteBST(root->left, dltKey)
6 else if dltKey > root->data.key then
7   |   return deleteBST(root->right, dltKey)
```

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees

## Delete node from BST

```
1  else
2      // Deleted node found – Test for leaf node
3      if root->left is null then
4          dltPtr = root
5          root = root->right
6          recycle(dltPtr)
7          return true
8      else if root->right is null then
9          dltPtr = root
10         root = root->left
11         recycle(dltPtr)
12         return true
```



## Delete node from BST

```
1  else
2      // ...
3      else
4          // Deleted node is not a leaf.
5          // Find largest node on left subtree
6          dltPtr = root->left
7          while dltPtr->right not null do
8              dltPtr = dltPtr->right
9          end
10         // Node found. Move data and delete leaf node
11         root->data = dltPtr->data
12         return deleteBST(root->left, dltPtr->data.key)
13     end
14 end
15 End deleteBST
```



# THANK YOU.

Tree concepts

Dept. Computer  
Science



Basic Tree Concepts

Binary Trees

Expression Trees

Binary Search Trees