IBA FCS
Faculty of Computer Science

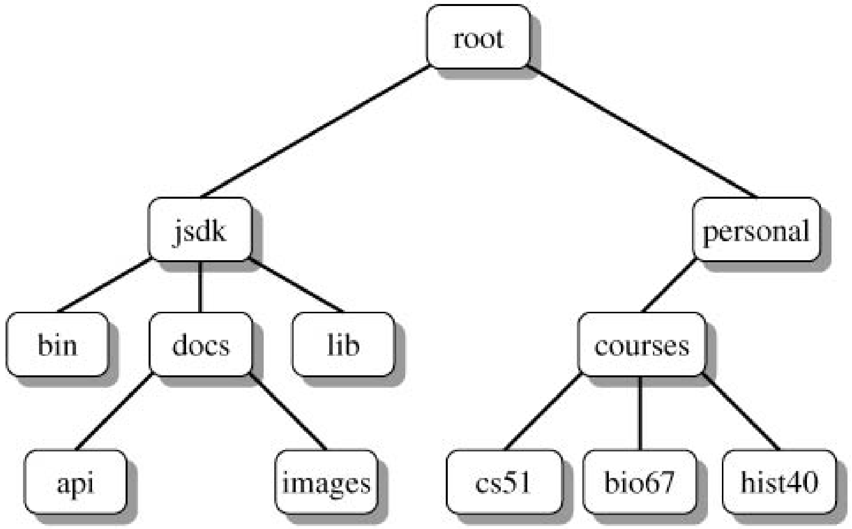
CSE 247
Data Structures

outline

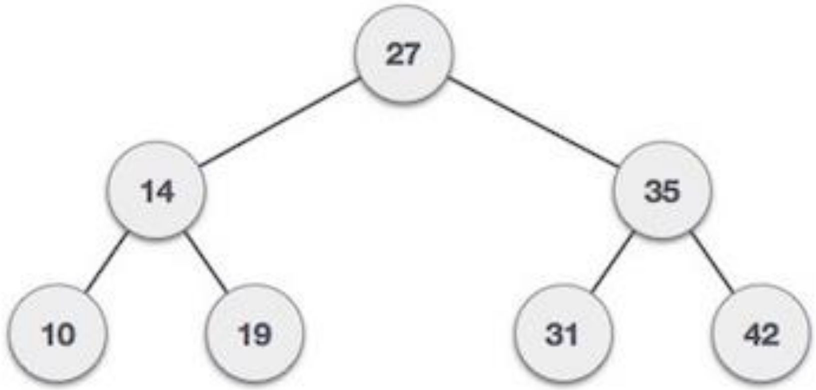
- Learn about binary search trees
- Implementation using a linked list.
  - Learn how to organize data (insert) in the binary search tree
  - Searching data in Binary search tree
  - Binary tree traversal algorithm. Explore recursive and non-recursive algorithms.
  - Binary tree Deletion algorithm
  - Big o analysis of insert, search and traversal, and deletion algorithm
- Applications of binary trees

Binary Trees

- A tree is a natural way to represent hierarchical data where data is arranged in a non-linear fashion.
- E.G.
  - binary tree,
  - binary search trees,
  - multiway trees.

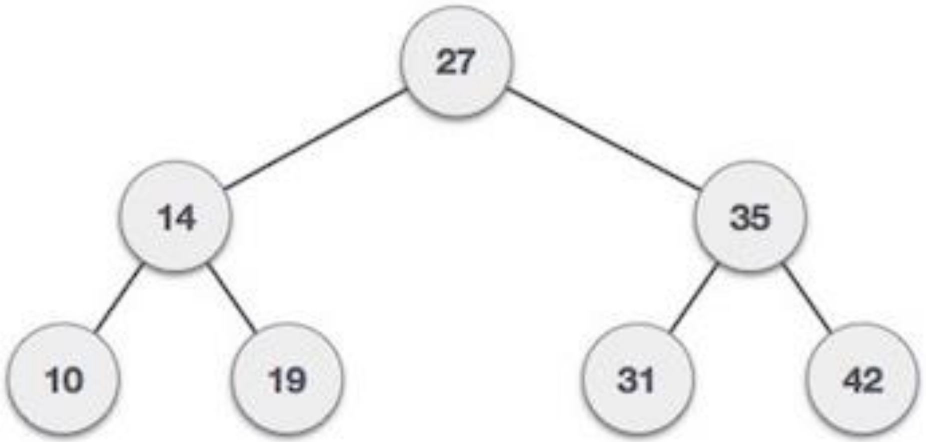


Path name: /jsdk/docs/api



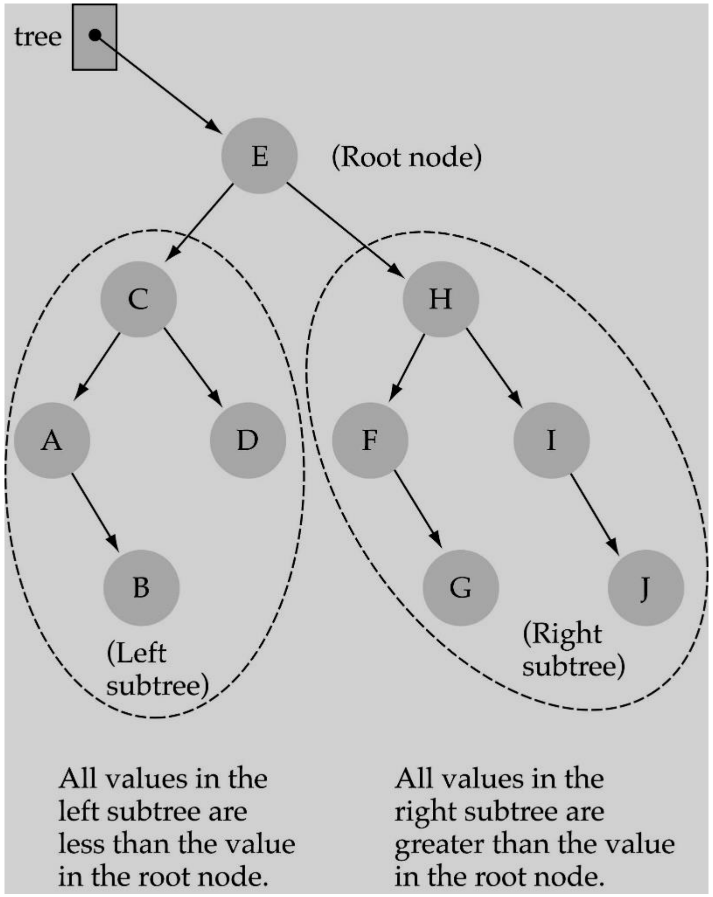
BST properties

- Tree is a non-linear data structure
- Each node has 0, 1 and 2 child
- Each node of a binary tree defines a left and a right subtree. Each subtree is itself a tree.



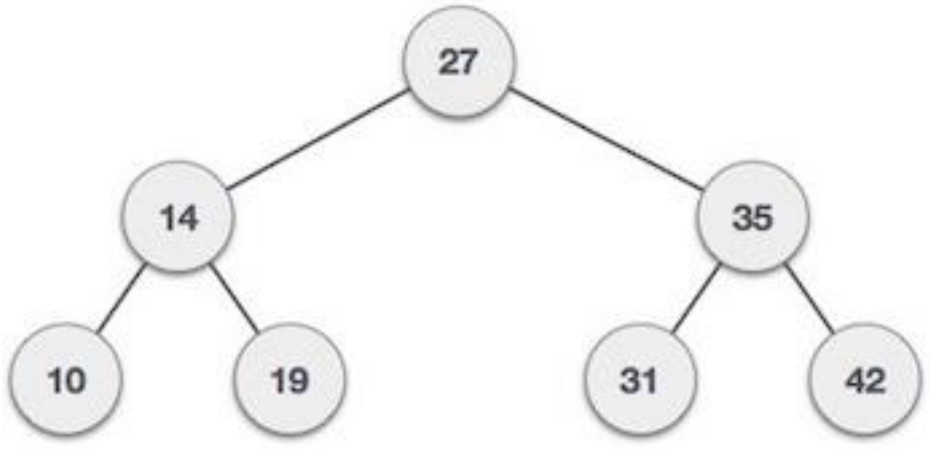
Binary search tree

- For a binary tree to be a binary search tree, the data of all the nodes in the left sub-tree of the root node should be less than the data of the root.
- The data of all the nodes in the right subtree of the root node should be greater than equal to the data of the root.
- As a result, the leaves on the farthest left of the tree have the lowest values, whereas the leaves on the right of the tree have the greatest values.


tree
E
(Root node)
C
H
A
D
F
I
B
G
J
(Left subtree)
(Right subtree)
All values in the left subtree are less than the value in the root node.
All values in the right subtree are greater than the value in the root node.

Binary Tree example

- BST is a collection of nodes arranged in a way where they maintain BST properties.

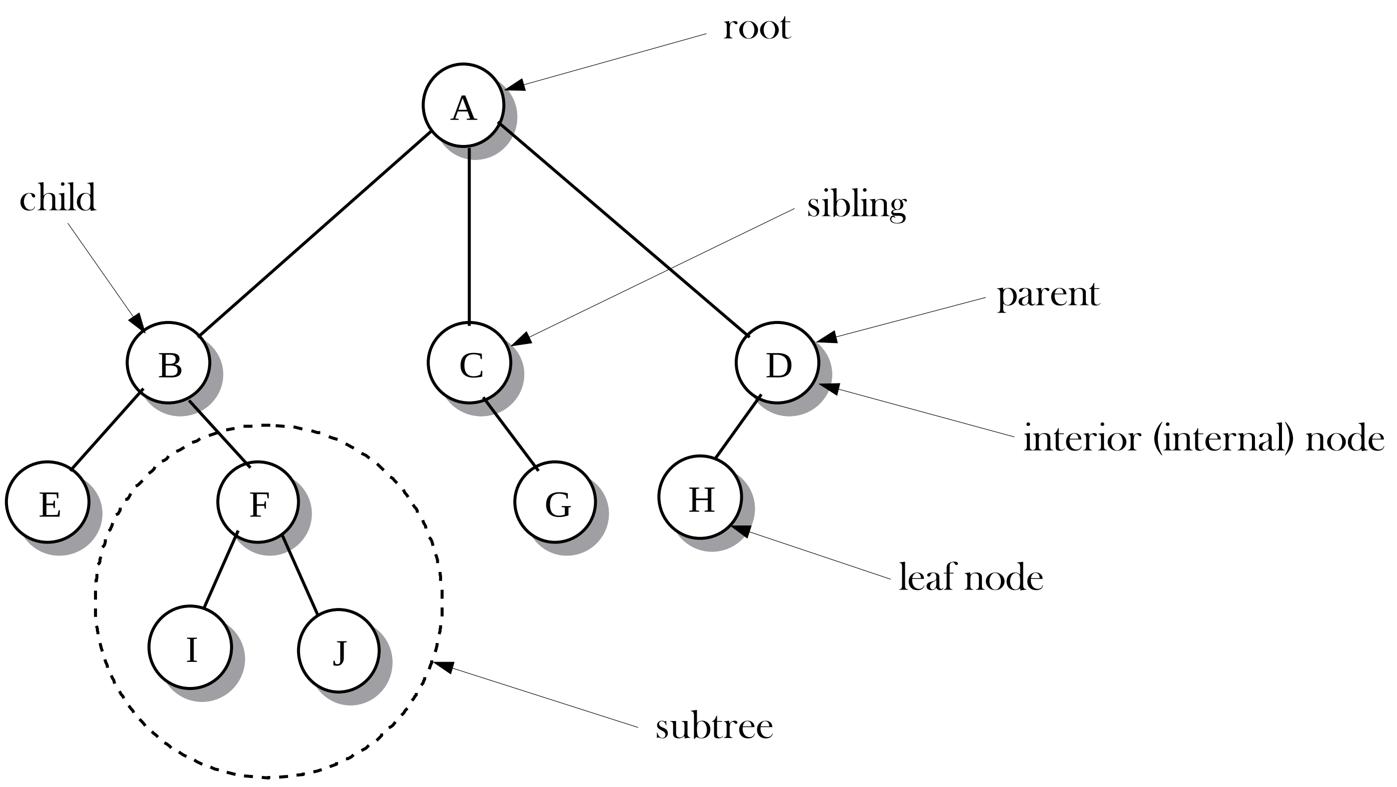


- Binary search tree implementation using
  - Linked list and
  - Arrays.

BST

- BST data structures that can support operations.
    - Search, Minimum, Maximum, Predecessor, Successor, Insert, and Delete.
- Can be used to build
    - Dictionaries.
    - Priority Queues.
- Basic operations take time proportional to the height of the tree – $O(h)$.

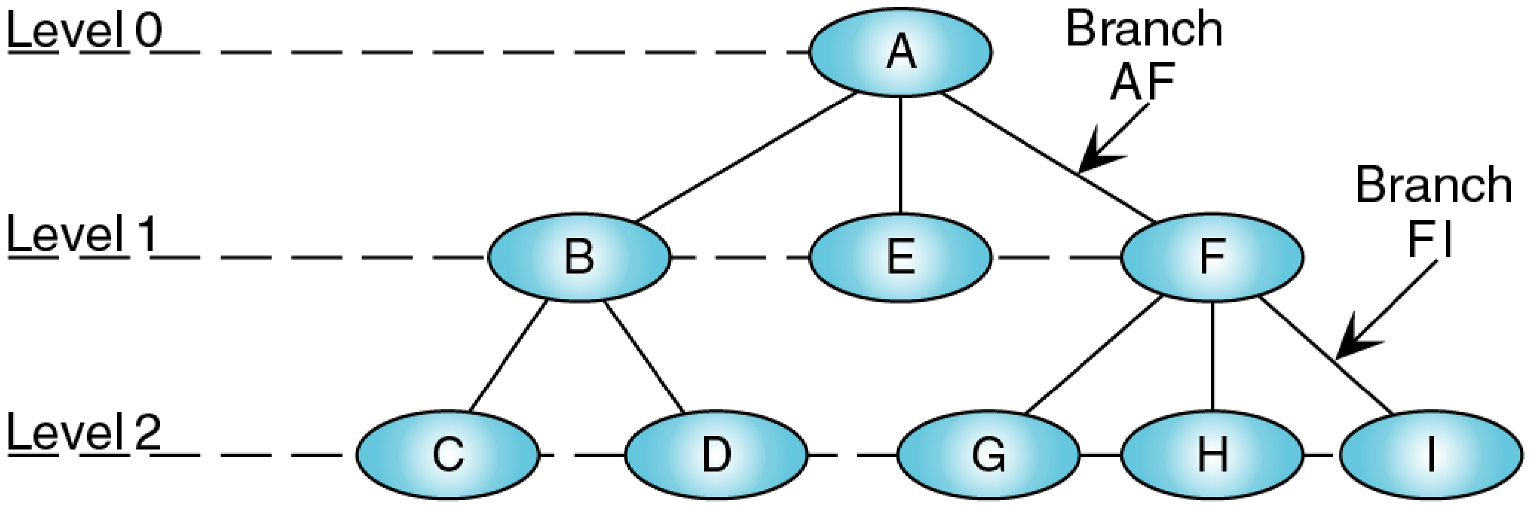
Tree Terminology



The height of a tree is the maximum level in the tree.

Terminology

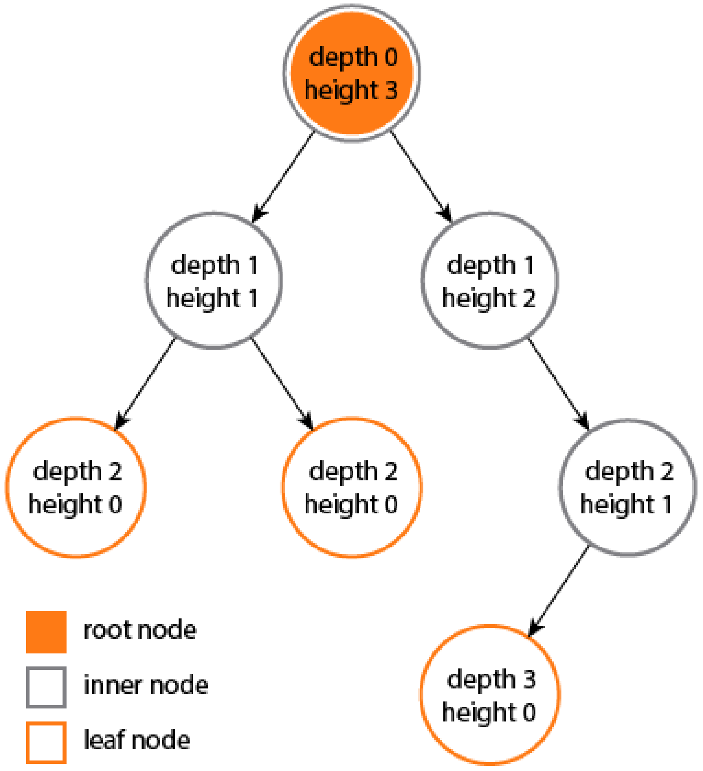
- Node, branch, root, indegree, outdegree, leaf, parent, child, siblings, ancestor, descendent, path, level, height, depth, subtree



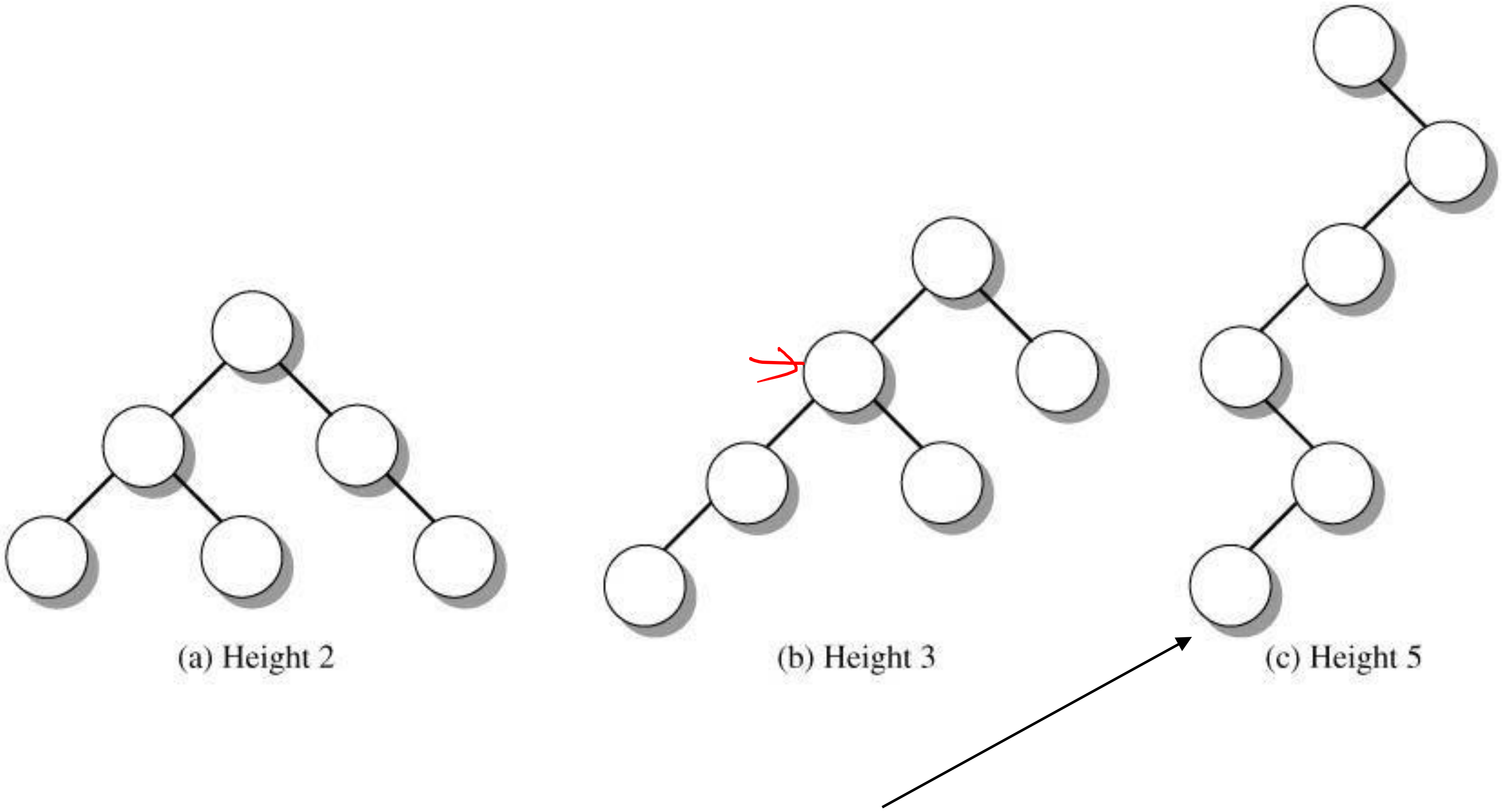
Parents:	A, B, F	Leaves	C, D, E, G, H, I
Children:	B, E, F, C, D, G, H, I	Internal nodes	B, F
Siblings:	{B, E, F}, {C, D}, {G, H, I}		

Binary Tree

- The **depth** of a node is the *number of edges* from the root node to the leaf node. A root node will have a depth of 0.
- The **height** of a node is the number of edges on the *longest path* from the leaf node to root. A leaf node will have a height of 0.
- The depth and height of a tree is same.
- But the depth and height of each node is different.

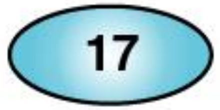


Height of a Binary Tree (concluded)

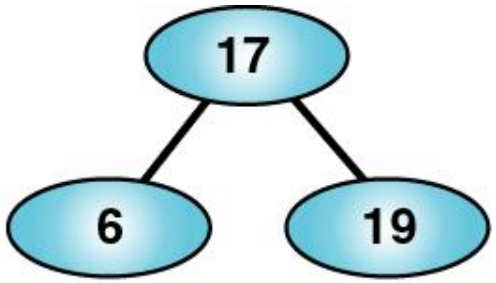


Degenerate binary tree

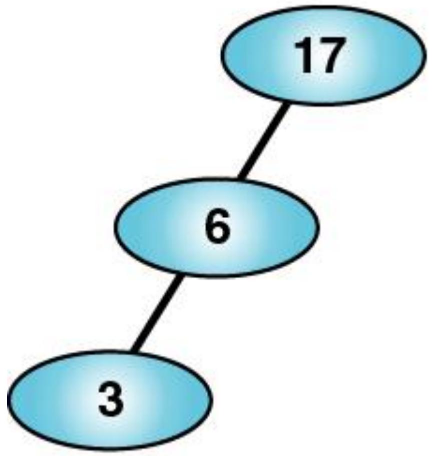
Some valid Binary Search Trees



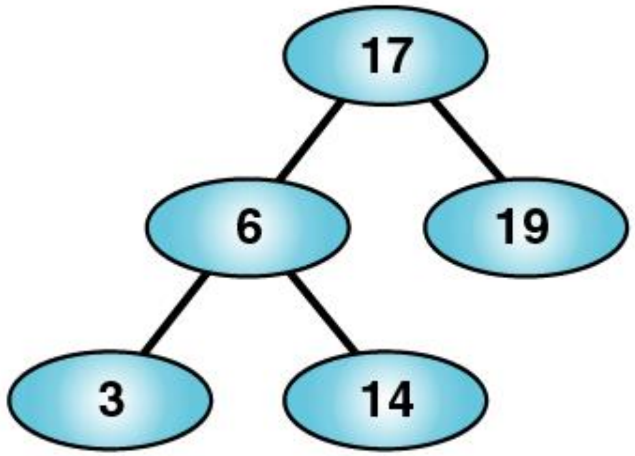
(a)



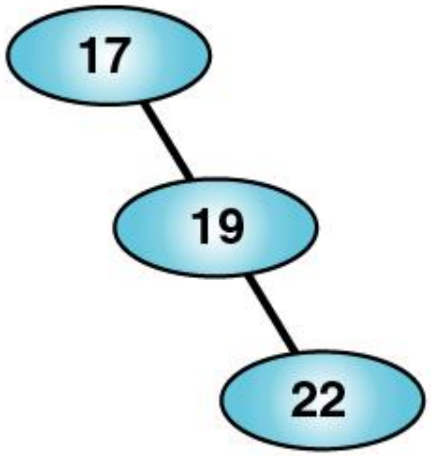
(b)



(c)

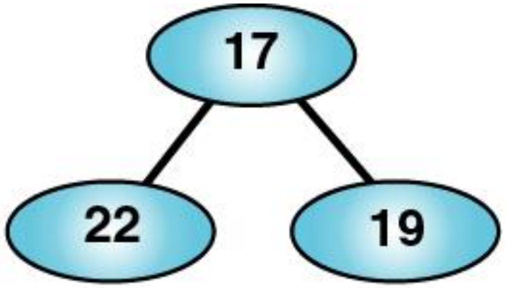


(d)

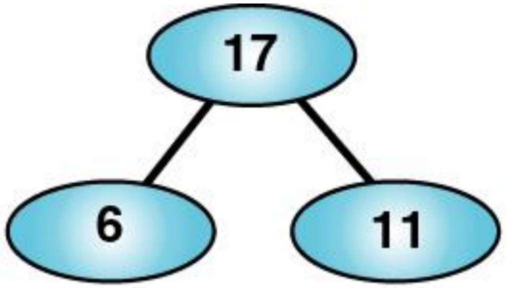


(e)

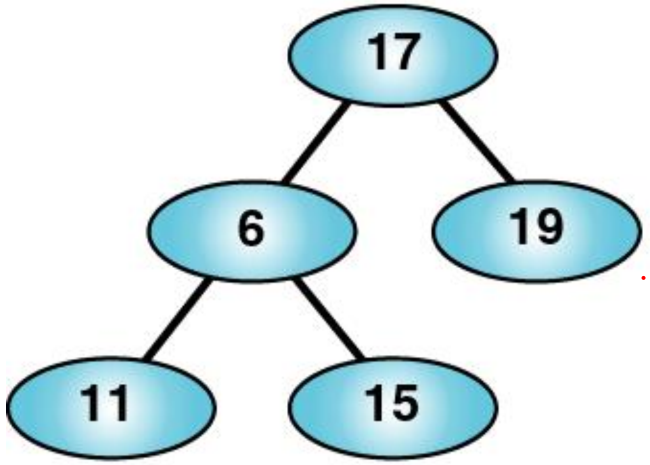
Some invalid Binary Search Trees



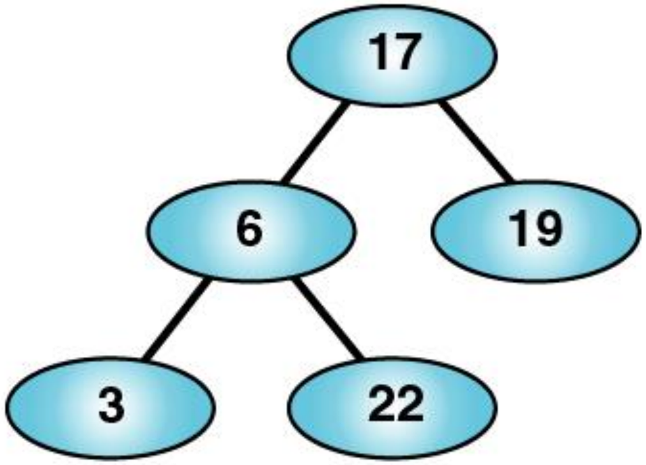
(a)



(b)



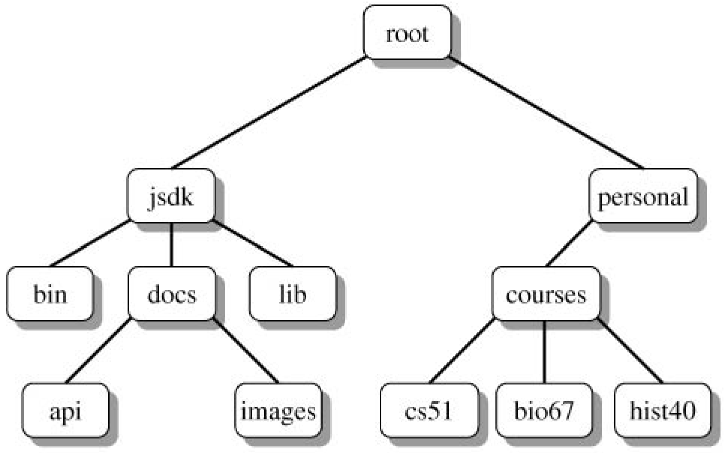
(c)



(d)

Application of Binary Trees

- For two-way decisions at each point in a process. Then the number of comparison could be reduced.
- to path finding, connected components.
- Application of Sorting
- Application of searching
- Expression tree
- Decision tree
- Workflow for visual effects
- File structure in operating system
- Router algorithm
- Data compression



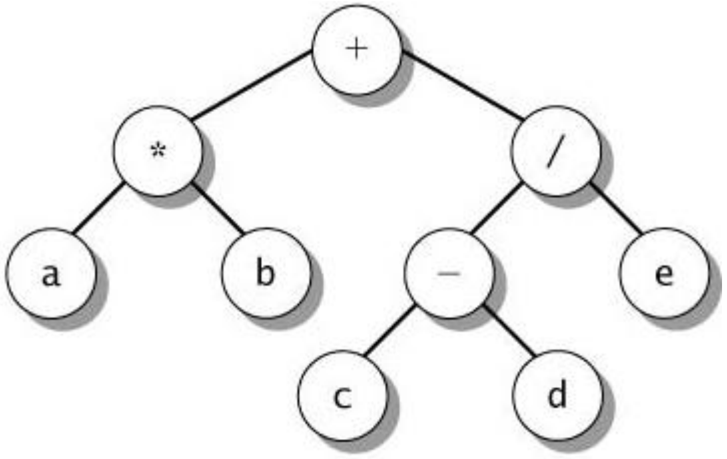
Path name: /jsdk/docs/api

Application cont.

- Efficiently locate elements to delete or insert (minimum or maximum)
- It can traverse a list in order without applying sorting algorithms.
- efficiently maintain a dynamically changing dataset in sorted order
- To maintain large data set such as dictionary, indexes, etc.
- To decide based on some condition, can be store in tree such kind of the tree is decision trees

Application Cont.

- In a binary tree each node has at most two successors. A compiler builds binary trees while parsing expressions in a program's source code.



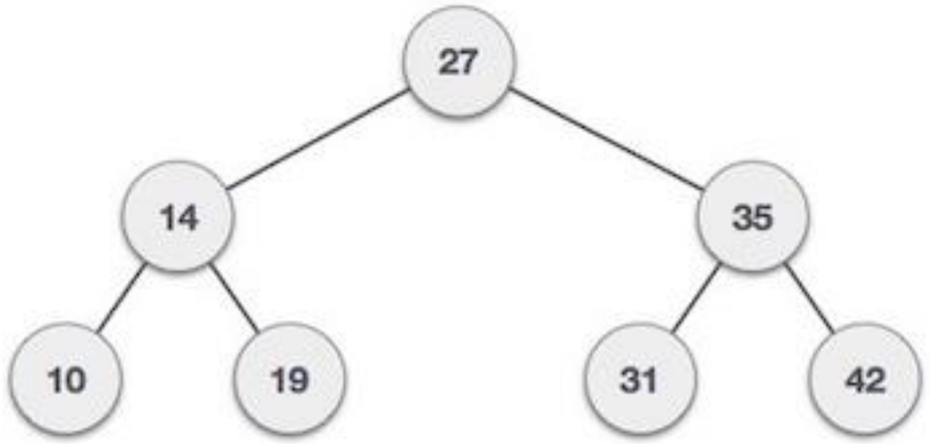
Expression: a* b + (c − d)/e

Implement Binary Search Trees

- Operations typically performed on a binary tree
  - Determine if the binary tree is empty
  - Find a particular item from the binary search tree
  - Insert an item in the binary search tree
  - Traverse the binary search tree
  - Delete a node from the binary search tree

Binary Tree example

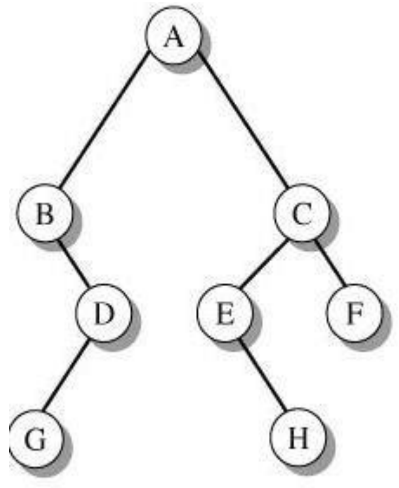
- BST is a collection of nodes arranged in a way where they maintain BST properties.



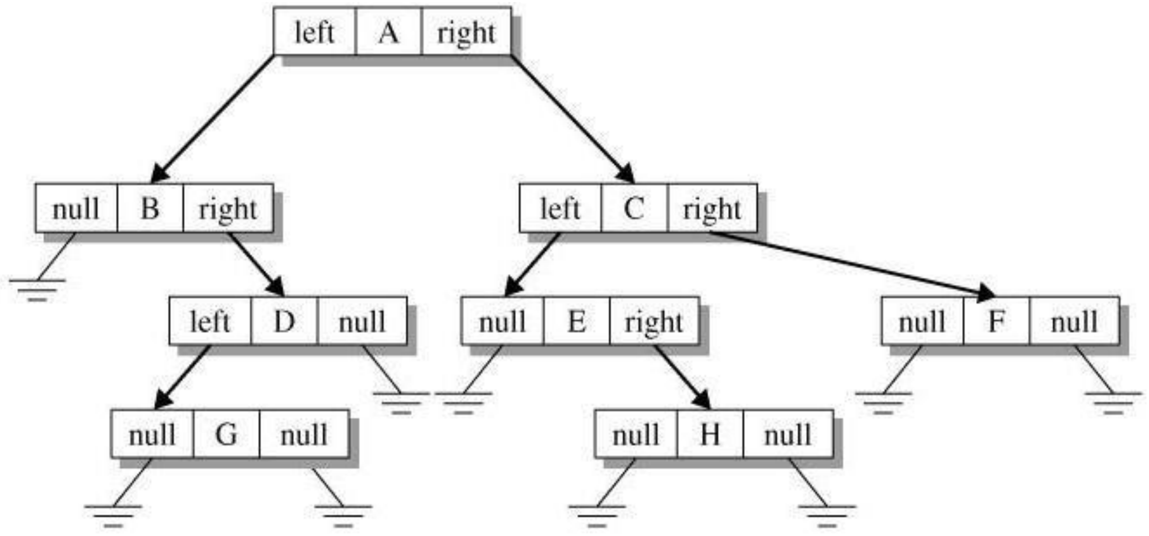
- Binary search tree implementation using
  - Linked list and
  - Arrays.

Binary tree implementation

- Linked list-based implementation



Abstract Tree Model



Tree Node Model

Abstract and tree node model of a binary tree.

Algorithm to Create binary tree

1. Compare VALUE with root node N of tree
   - If VALUE <N, proceed to the left child of N
   - If VALUE >N, proceed to the right child of N
2. Repeat step 1. until N=null and assign it a new node with value VALUE.

Recursive definition of binary tree

T is a binary tree if T

- has no node (T is an empty tree)

  or

- has at most two subtrees.

Binary Tree Traversal

- Must start with the root, and then
- Three different traversals
  - Inorder (LNR)
  - Preorder(NLR)
  - Postorder(LRN)

Binary Tree Traversal

- Inorder traversal (LNR)
  - Traverse the left subtree
  - Visit the node
  - Traverse the right subtree
- Preorder traversal (NLR)
  - Visit the node
  - Traverse the left subtree
  - Traverse the right subtree
- Postorder traversal (LRN)
  - Traverse the left subtree
  - Traverse the right subtree
  - Visit the node

Big oh Analysis

- The tree structure depends on input sequence so in

average case :

  - insert $\rightarrow$ O(logn)
  - Find $\rightarrow$ O(logn)
  - Minimum $\rightarrow$ O(logn)
  - Maximum $\rightarrow$ O(logn)
  - Traverse $\rightarrow$ O(n+m) so, O(n)

- Worst case:
  - insert $\rightarrow$ O(n)
  - Find $\rightarrow$ O(n)
  - Minimum $\rightarrow$ O(n)
  - Maximum $\rightarrow$ O(n)
  - Traverse $\rightarrow$ O(n+m) so, O(n)

Delete node in a Binary Search Trees

- There are three possibilities:
  1. To delete a leaf node (no children): disconnect it.
  2. To delete a node with one child: bypass the node and directly connect to the child.
  3. To delete a node with two children: find the smallest node in its right subtree (or the largest node in its left subtree), copy the minimum value into the info field of the "node to be deleted" and then delete the minimum node instead, which can only have a right child, so the situation becomes one of the above two.

BST deletion

Deleting a node is the most common operation required for binary search trees. You start by finding the node you want to delete, using the same approach as we saw in find() and insert() method. once node is found then there are three possible cases in binary search tree to consider:

1. **The node to be deleted is a leaf (has no children):**

   To delete a leaf node, you simply change the appropriate child field in the node's parent to point to null, instead of to the node.

2. **The node to be deleted has one child:**

   The node has only two connections: to its parent and to it's only a child. You want to cut the node out of this sequence by connecting its parent directly to its child.

3. **The node to be deleted has two children:**

   To delete a node with two children, replace the node with its in-order **successor** and delete the successor node as case 1 or case 2.

BST deletion

- Finding the Successor:

First, the algorithm goes to the node's right child, which must have a value larger than the node. Then it goes to this right child's left child (if it has one), until no further left child in the following down to the path of left child.

The last left child in this path is the successor of the node to deleted.

If the right child of the original node has no left children, this right child is itself the successor.

Algorithm of delete method

```
Delete(Object element){
  node[] ref=find(data)  //find node to delete that return node t and its parent p references

  if(node t has no child)  {   // call deleteNochild(t,p)  }

  if(node t has one child) {  // call deleteOnechild(t,p)   }

else{    //(node t has two children)
        Find node minNode with minimum value on t's  right subtree
        Replace t data with this minNode value
         If(minNode is noChiled node)  // remove minNode
          // call deleteNochild(t,p);
          else
         // call deleteOnechild(t,p)
      }
}
```

Big oh

- Deletion
  - Worst case is O(n)
  - Average case is O(logn)