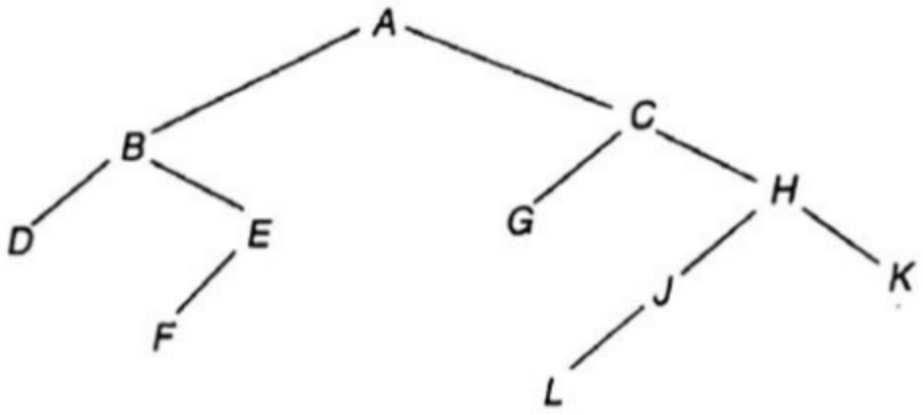
Outline

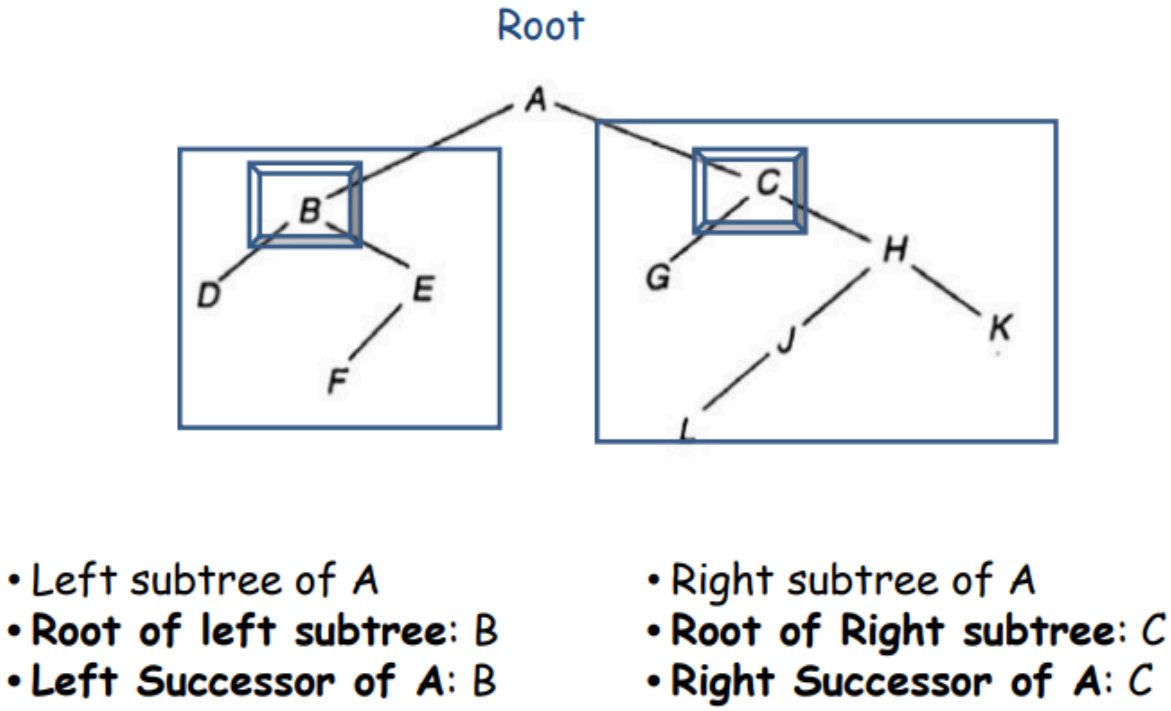
- Binary Tree
- Representing Binary Trees in Memory
- Traversing Binary Trees
- Traversal Algorithm using Stacks
- Header Nodes: Threads
- Binary Search Trees
- Searching and inserting in BST
- Deleting in BST
- AVL Search Trees
- Insertion in an AVL Search Tree
- Deletion in an AVL Search Tree
- m-way Search Trees
- Searching, Insertion and Deletion in an m-way Search Tree
- Heap; Heapsort
- Path Lengths; Huffman's Algorithm
- General Trees

Binary Tree

A binary tree T is defined as a finite set of elements, called nodes, such that:

- T is empty (called the null tree or empty tree) or
- T contains a distinguished node R, called the root of T, and the remaining nodes of T form an ordered pair of disjoint binary tree T_1 and T_2





- Left subtree of A
- **Root of left subtree**: B
- **Left Successor of A**: B

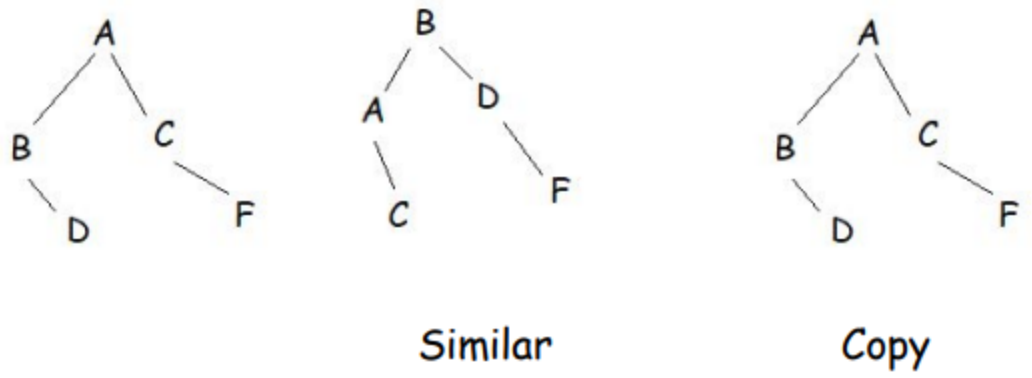
- Right subtree of A
- **Root of Right subtree**: C
- **Right Successor of A**: C

A, B, C, H, have two successors
E and J have one successor
D, F, G, L and K have no successor

The nodes with no successor are called terminal nodes or leaf nodes.

Binary Tree T and T' are said to be **similar** if they have the same structure.

The trees are said to be **copies**, if they are **similar** and they have the same contents at corresponding nodes.

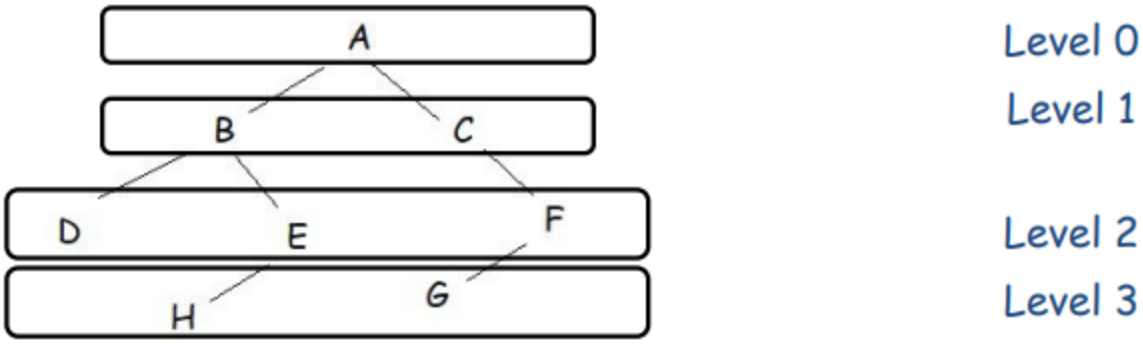


Descendant is the opposite of ancestor. A node might have multiple descendant and/or ancestor.

The line connecting two nodes are called edge.

A sequence of consecutive edges is called a path
A terminal node is called a leaf
A path ending in a leaf is called a branch

Level Number:



The depth (or height) of a tree T is the maximum number of nodes in a branch of T

Complete Binary Tryy

The tree T is said to be complete

- If all its levels, except possibly the last, have the maximum number of possible nodes, and
- If all the nodes at the last level appear as far left as possible.
  Level r of T can have at most 2^r nodes

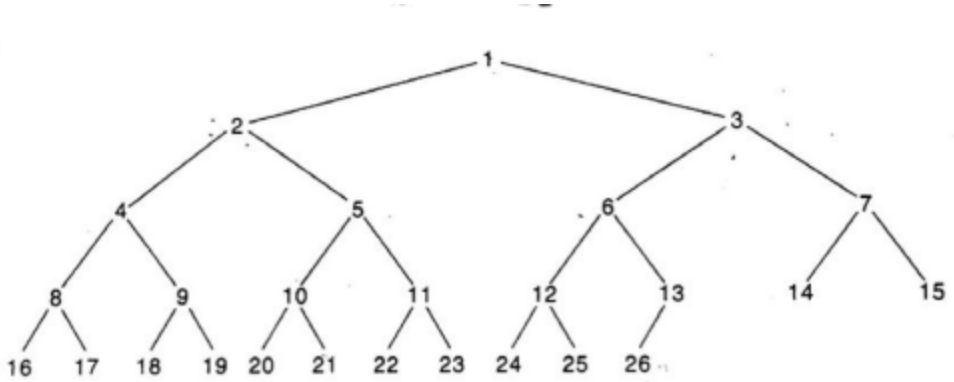


Fig. 7.4 Complete Tree T_{26}

- Left child K node is $2K$
- Right child K node is $2K + 1$
- the depth d_n of the complete tree T_n with n nodes is given by,

$$D_n = \lfloor \log_2 n + 1 \rfloor$$

Extended Binary Tree: 2-Trees

A binary tree T is said to be a 2-tree or an extended binary tree if each node N has either 0 or 2 children.

- The nodes with 2 children are called internal nodes
- The nodes with 0 children are called external nodes.

Representing Binary Trees in Memory

Linked Representation

Uses three parallel arrays: $INFO$, $LEFT$, $RIGHT$
A pointer variable $ROOT$ as follows - (each node N of T will correspond to location K such that)

1. $INFO[K]$ contains the data at the node N
2. $LEFT[K]$ contains the location of the left child of node N
3. $RIGHT[K]$ contains the location of the right child of node N

Sequential Representation

Uses only a single linear array $TREE$ as follows:

- The root R of T is stored in $TREE[1]$

- If a node N occupies $TREE[K]$, then its left child is stored in $TREE[2K]$ and its right child is stored in $TREE[2K+1]$

Traversing Binary Trees

Preorder

- Process the root R
- Traverse the left subtree of R in preorder
- Traverse the right subtree of R in preorder

Inorder

- Traverse the left subtree of R in inorder
- Process the root R
- Traverse the right subtree of R in inorder

Postorder

- Traverse the left subtree of R in postorder
- Traverse the right subtree of R in postorder
- Process the root R

Preorder Traversal using stacks
$PREORD(INFO, LET, RIGHT, ROOT)$
A binary tree T is in memory. The algorithm does a preorder traversal of T, applying an operation $PROCESS$ to each of its nodes. An array $STACK$ is used to temporarily hold the address of nodes.

```
1. [Initially push NULL onto STACK, and initialize PTR.]
   Set TOP := 1, STACK[1] := NULL and PTR := ROOT
2. Repeat Steps 3 to 5 while PTR != NULL:
3.     Apply PROCESS to INFO[PTR].
4.     [Right child?]
       If RIGHT[PTR] != NULL, then: [Push on STACK.]
           Set TOP := TOP + 1, and STACK[TOP] := RIGHT[PTR]
       [End of If structure.]
4.     [Left child?]
       If LEFT[PTR] != NULL, then:
           Set PTR := LEFT[PTR]
       Else: [Pop from STACK.]
           Set PTR := STACK[TOP] and TOP := TOP - 1
       [End of If structure.]
```

```
    [End of Step 2 loop.]
5. Exit.
```

Header Nodes: Threads

Approximately half of the entries in the pointer fields $LEFT$ and $RIGHT$ will contain null element.

We will replace certain null entries by special pointers which point to nodes higher in the tree

These special pointers are called threads

Binary tree with such pointers are called threaded trees.

There are many ways to thread a binary tree T.

One-way Threading

- A thread will appear in the right field of a node and will point to the next node in the inorder traversal of T

Two-way Threading

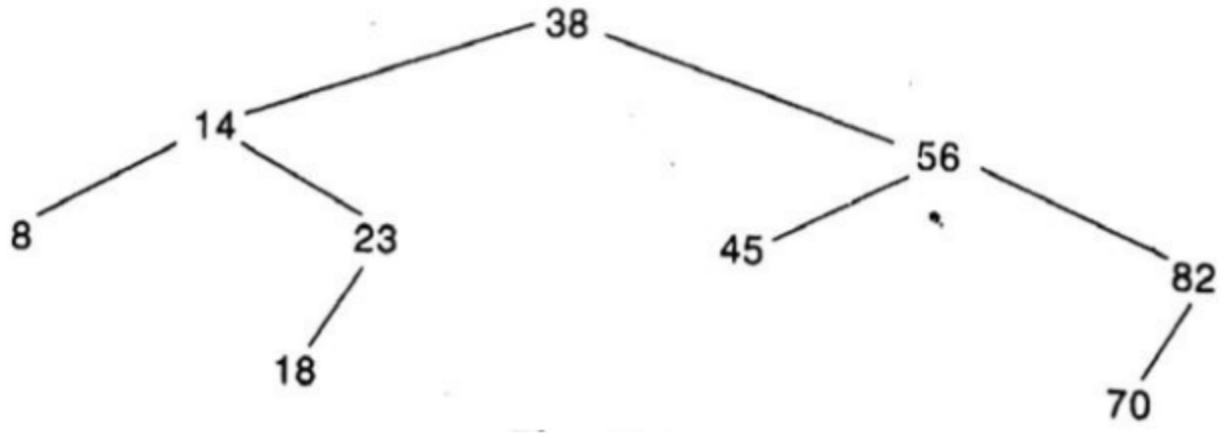
- A thread also appear in the $LEFT$ field of a node and will point to the preceding node in the inorder traversal of T

Binary Search Tree

A tree T is called a binary search tree (or binary sorted tree) if each node N of T has the following property

- The value at N is great than every value in the left subtree of N
- The value at N is less than or equal to every value in the right subtree of N

To find a sorted list of the elements of T, we need to traverse a BST by inorder.



AVL Search Tree

Disadvantage of a binary search tree is that the worst case time complexity of a search can be up to $O(n)$, due to bad building of the BST. The solution is to maintain the BST to be of balanced height. That's where AVL Search Tree comes in

- An empty BST is an AVL Tree
- A non empty binary tree T is an AVL tree if and only if:
  - Given T_L and T_R is the left and right subtrees of T
  - $h(T_L)$ and $h(T_R)$ are the heights of subtree T_L and T_R respectively
  - T_L and T_R are AVL trees and $|h(T_L) - h(T_R)| \leq 1$

Balance factor of an AVL tree is given by

$$h(T_L) - h(T_R)$$

Searching in AVL search tree is same as BST

Insertion in an AVL Search Tree

First Phase: Insert an element into AVL Search tree
Second Phase; The balance factor of any node in the tree is affected, we resort to techniques called rotation to resort the balance of the search tree

To perform rotation, it is necessary to identify node A whose $BF(A)$ is either 0, 1, -1 and which is the nearest ancestor to the inserted node on the path from the inserted node to the root

The rebalancing rotations are classified as LL, LR, RR and RL based on the position of the inserted node with reference to A

m-way Search Trees

An m-way search tree T may be an empty tree.

If T is non empty, it satisfied the following properties:

- Each node has at most m (order of the tree) child nodes, a node may be represented as A_0, (K_1, A_1), (K_2, A_2), $\ldots$, (K_{m-1}, A_{m-1}) where, K_i, are keys and A_i are the pointers to the subtrees of T
- If a node has k child where $k \leq m$, then the node have only $(k-1)$ keys, $K_1, K_2, \ldots, K_{k-1}$ contained nodes such that $K_i < k_{i+1}$ and each of the keys partitions all the keys in the subtree into k subsets.

B Trees

A B-tree of order m, if non empty, is an m-way search tree in which:

- The root has at least two child nodes and at most m child nodes
- The internal nodes except root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes
- The number of keys in each internal node is one less than the number of child nodes and these keys partition the keys in the subtrees of the node in a manner similar to that of m-way search trees.
- All leaf nodes are on the same level

Heap

H is a complete binary tree with n element

H is called heap or a maxheap, if each node N of H has the following property

- The value at N is greater than or equal to the value at each of the children of N.
- For a minheap; the value of N is less than or equal to the value at each of the children of N