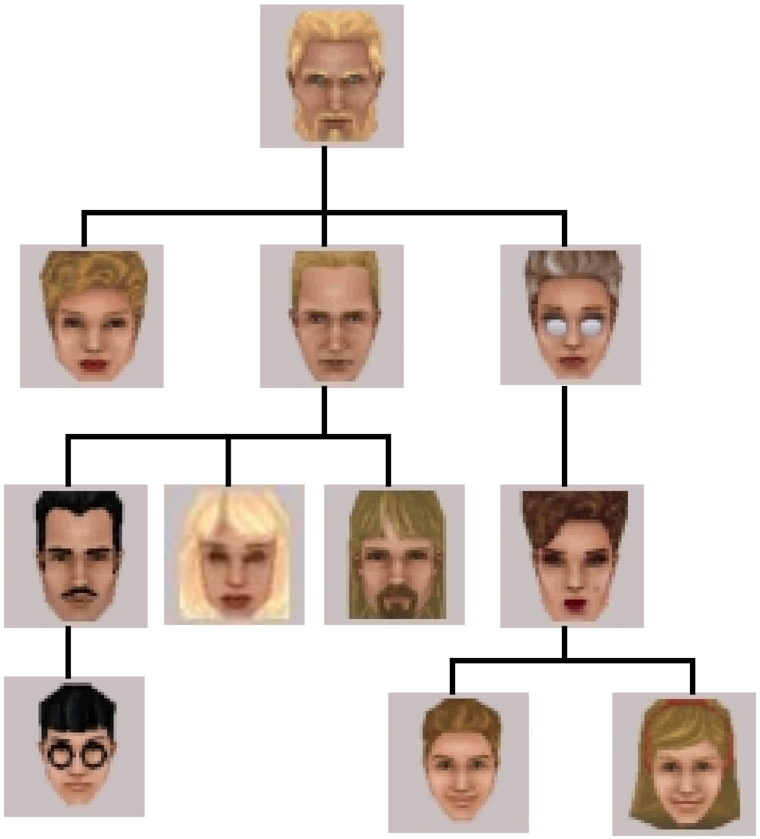
CHAPTER 4

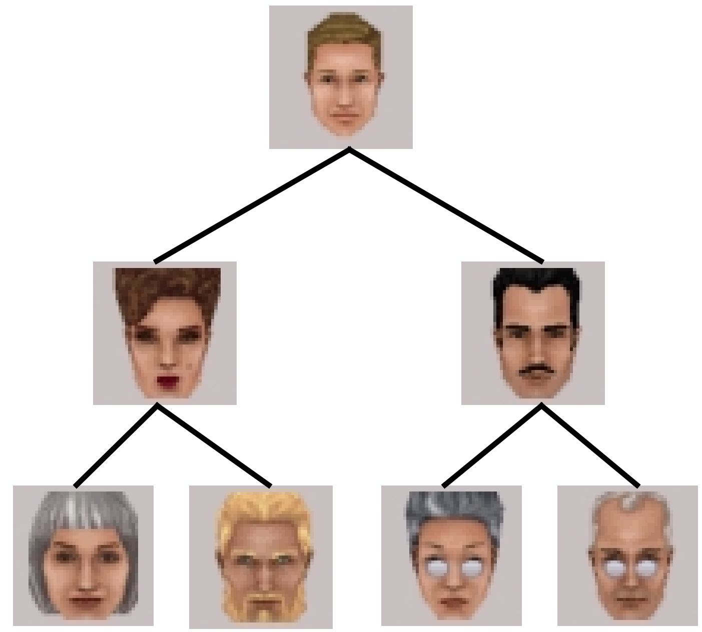
TREES

§ 1 Preliminaries

1. Terminology



Linear Tree



Pedigree Tree
(binary tree)

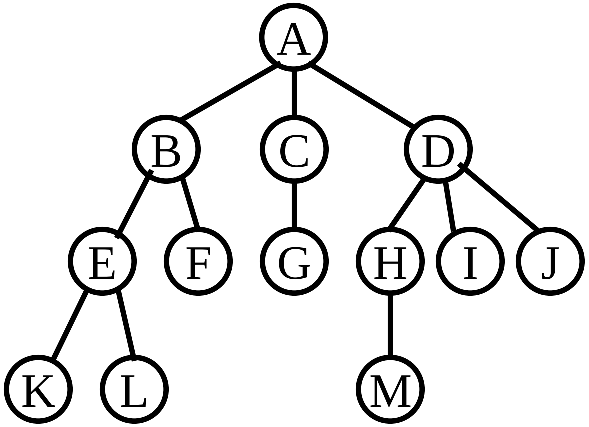
【Definition】A tree is a collection of nodes. The collection can be empty; otherwise, a tree consists of

(1) a distinguished node r, called the root;

(2) and zero or more nonempty (sub)trees $T_1, \cdots, T_k$, each of whose roots are connected by a directed edge from r.

Note:

- Subtrees must not connect together. Therefore every node in the tree is the root of some subtree.
- There are $N - 1$ edges in a tree with N nodes.
- Normally the root is drawn at the top.



- degree of a node ::= number of subtrees of the node. For example, degree(A) = 3, degree(F) = 0.

- degree of a tree ::= $\max_{node \in tree} \{ degree(node) \}$
  For example, degree of this tree = 3.

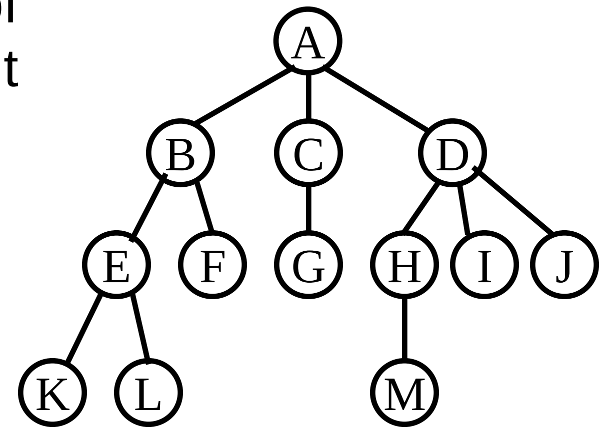
- parent ::= a node that has subtrees.

- children ::= the roots of the subtrees of a parent.

- siblings ::= children of the same parent.

- leaf (terminal node) ::= a node with degree 0 (no children).

- level of a node ::= defined by letting the root be at level one. If a node is at level l the its child nodes are at level l+1.



- path from n_1 to n_k ::= a (unique) sequence of nodes $n_1, n_2, \ldots, n_k$ such that n_i is the parent of n_{i+1} for $1 \le i < k$.

- length of path ::= number of edges on the path.

- depth of n_i ::= length of the unique path from the root to n_i. Depth(root) = 1.

- height of n_i ::= length of the longest path from n_i to a leaf.

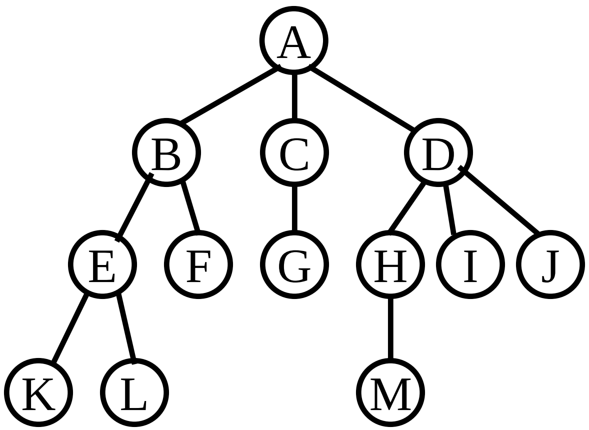
- height (depth) of a tree ::= height(root) = depth(deepest leaf).

- ancestors of a node ::= all the nodes along the path from the node up to the root.

- descendants of a node ::= all the nodes in its subtrees.

2. Implementation

❖ List Representation



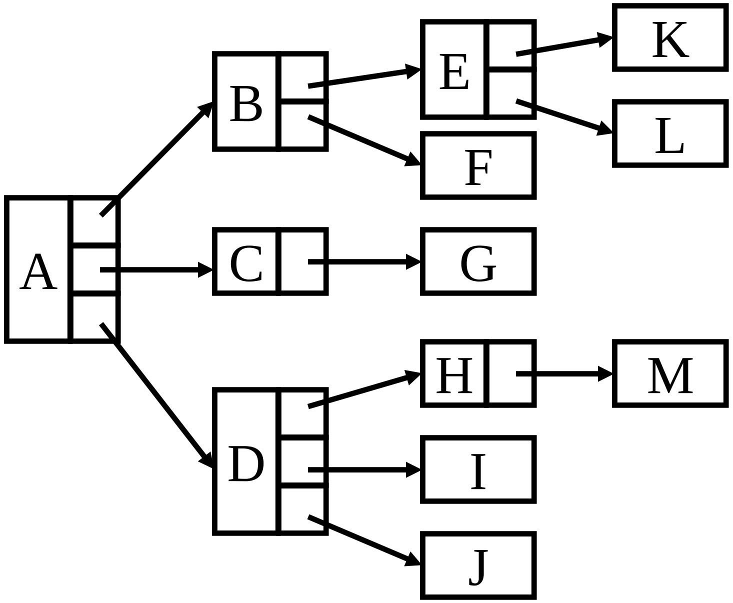
(A)

(A (B, C, D))

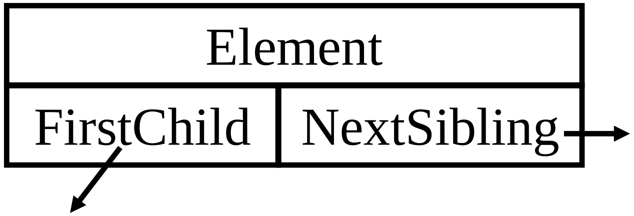
(A (B (E, F), C (G), D (H, I, J)))

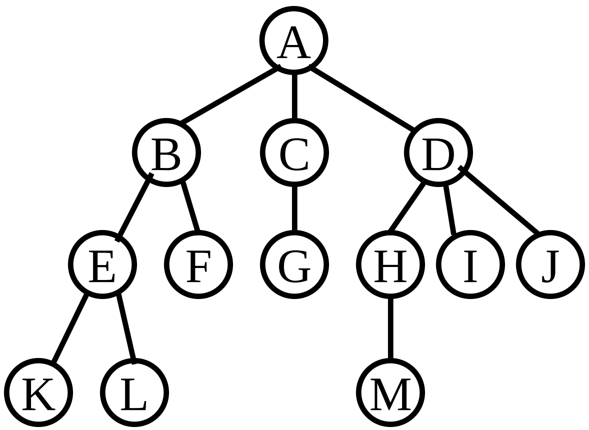
(A (B (E (K, L), F), C (G), D (H (M), I, J)))

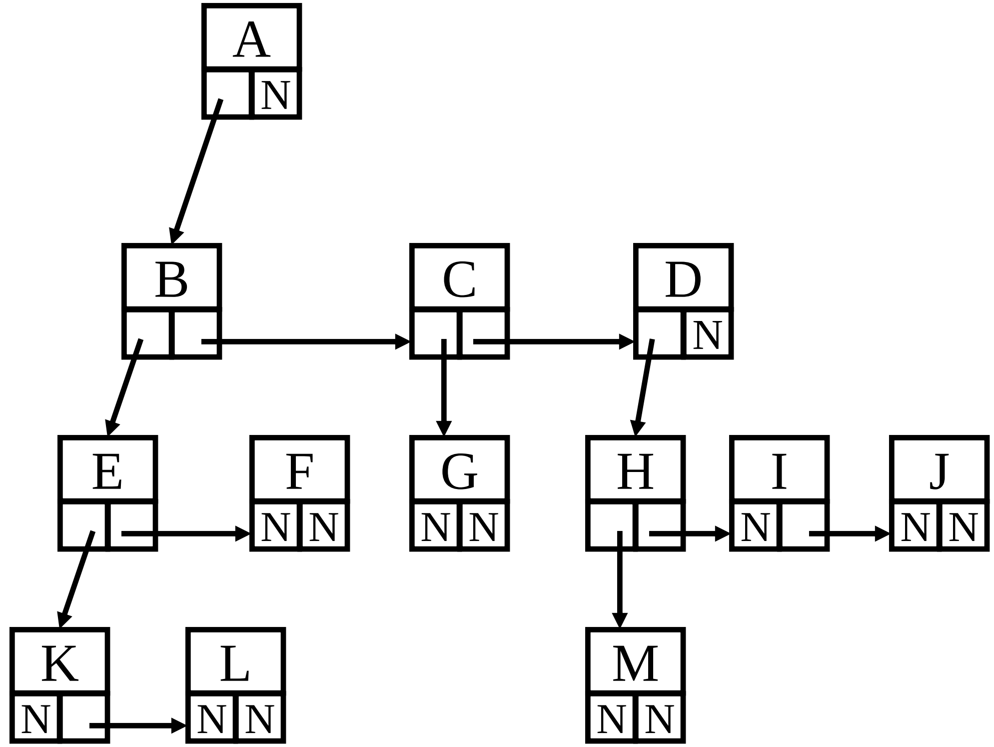
Linked List representation



❖ FirstChild-NextSibling Representation
(Left Child Right Sibling)





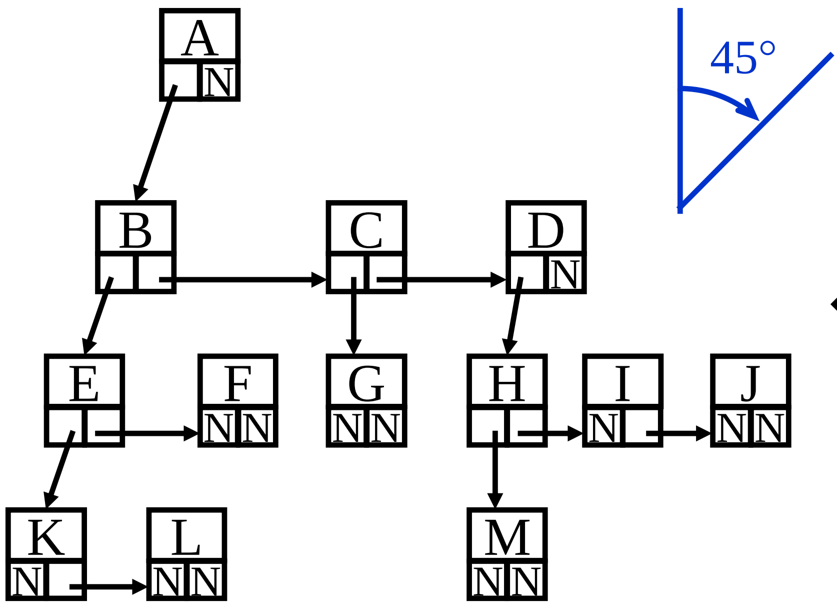


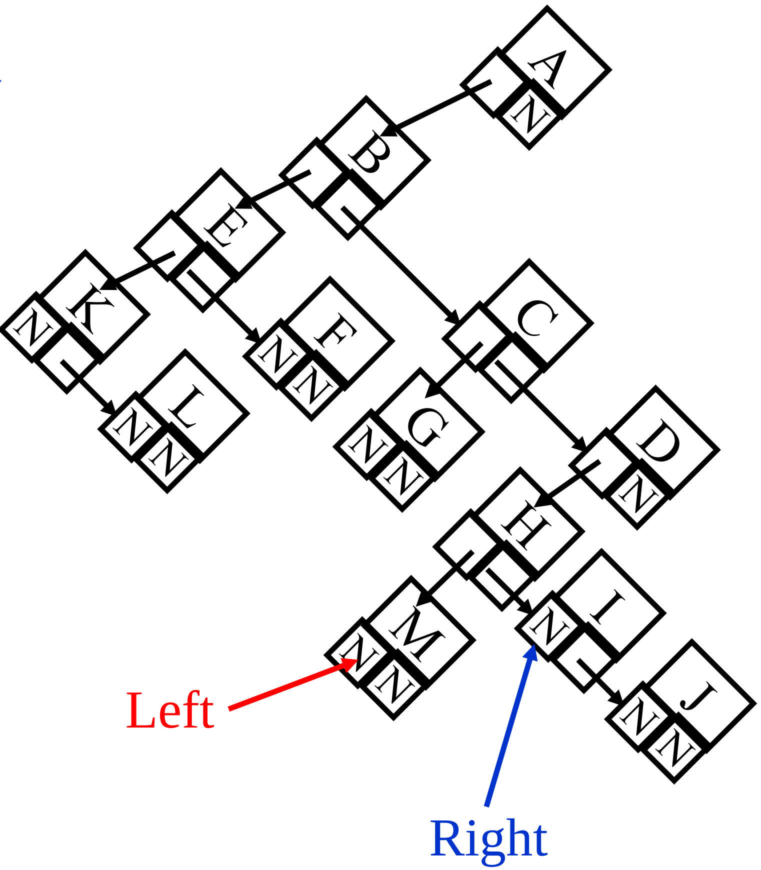
Note: The representation is not unique since the children in a tree can be of any order.

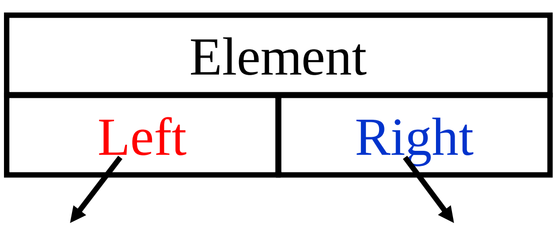
§ 2 Binary Trees

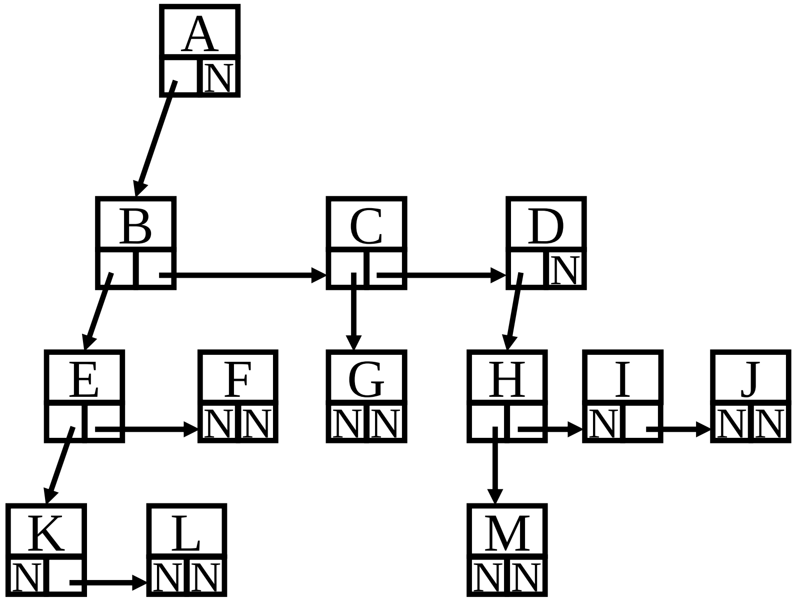
【Definition】A binary tree is a tree in which no node can have more than two children.

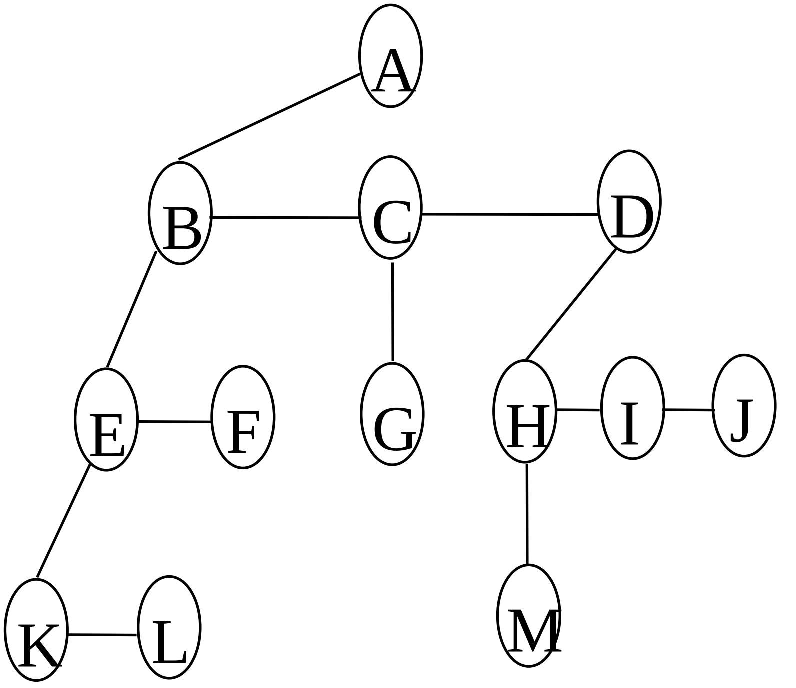
Rotate the FirstChild-NextSibling tree clockwise by 45°.

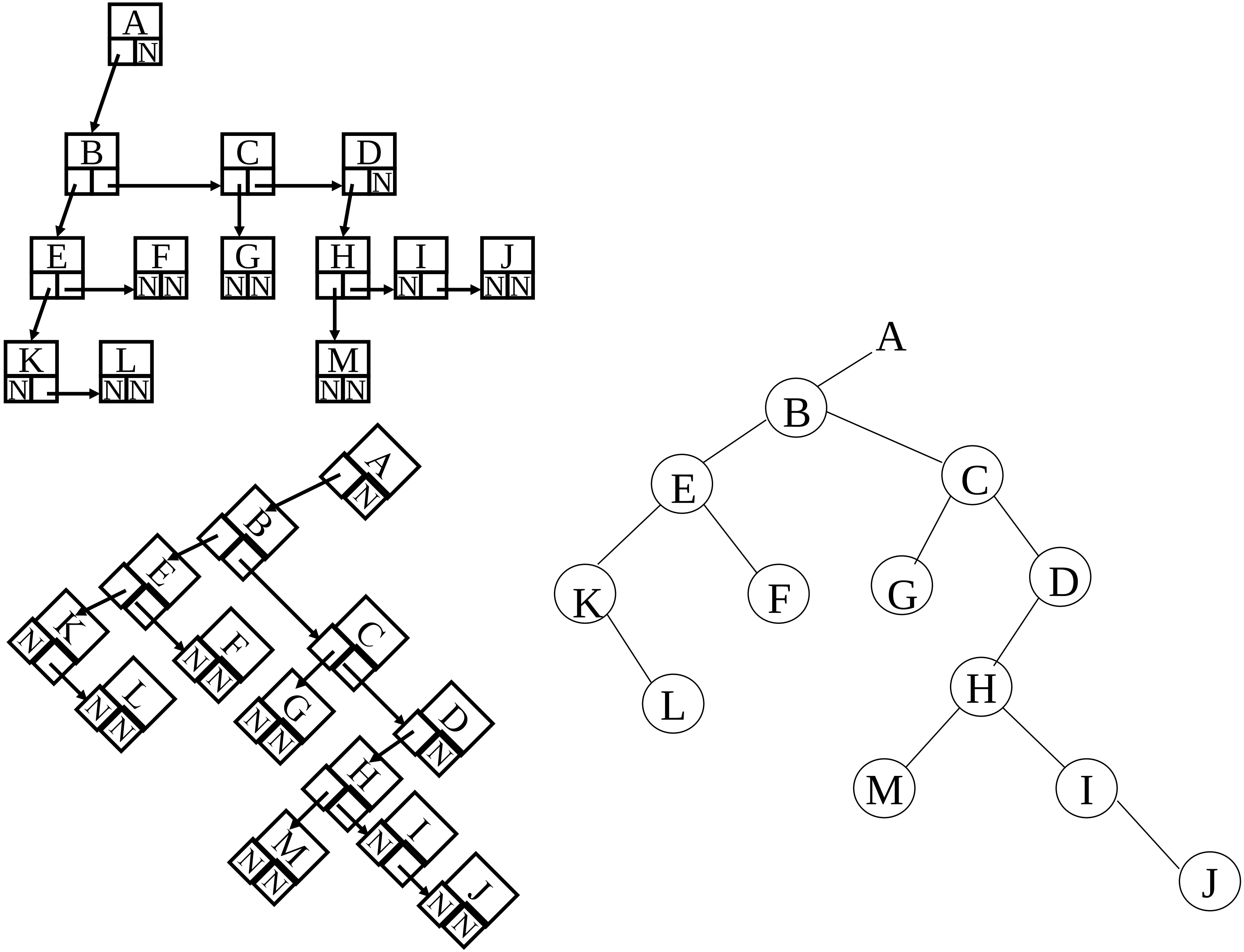







A
N
B
C
D
N
E
F
N N
G
N N
H
I
N
J
N N
K
N
L
N N
M
N N


A
B
C
D
E
F
G
H
I
J
K
L
M


A
B
C
D
E
F
G
H
I
J
K
L
M

Maximum Number of Nodes in BT

- The maximum number of nodes on level i of a binary tree is 2^{i-1}, $i>=1$.
- The maximum number of nodes in a binary tree of depth k is 2^k-1, $k>=1$.

Prove by induction.

$$\sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

Relations between Number of Leaf Nodes and Nodes of Degree 2

For any nonempty binary tree, T, if n_0 is the number of leaf nodes and n_2 the number of nodes of degree 2, then $n_0=n_2+1$

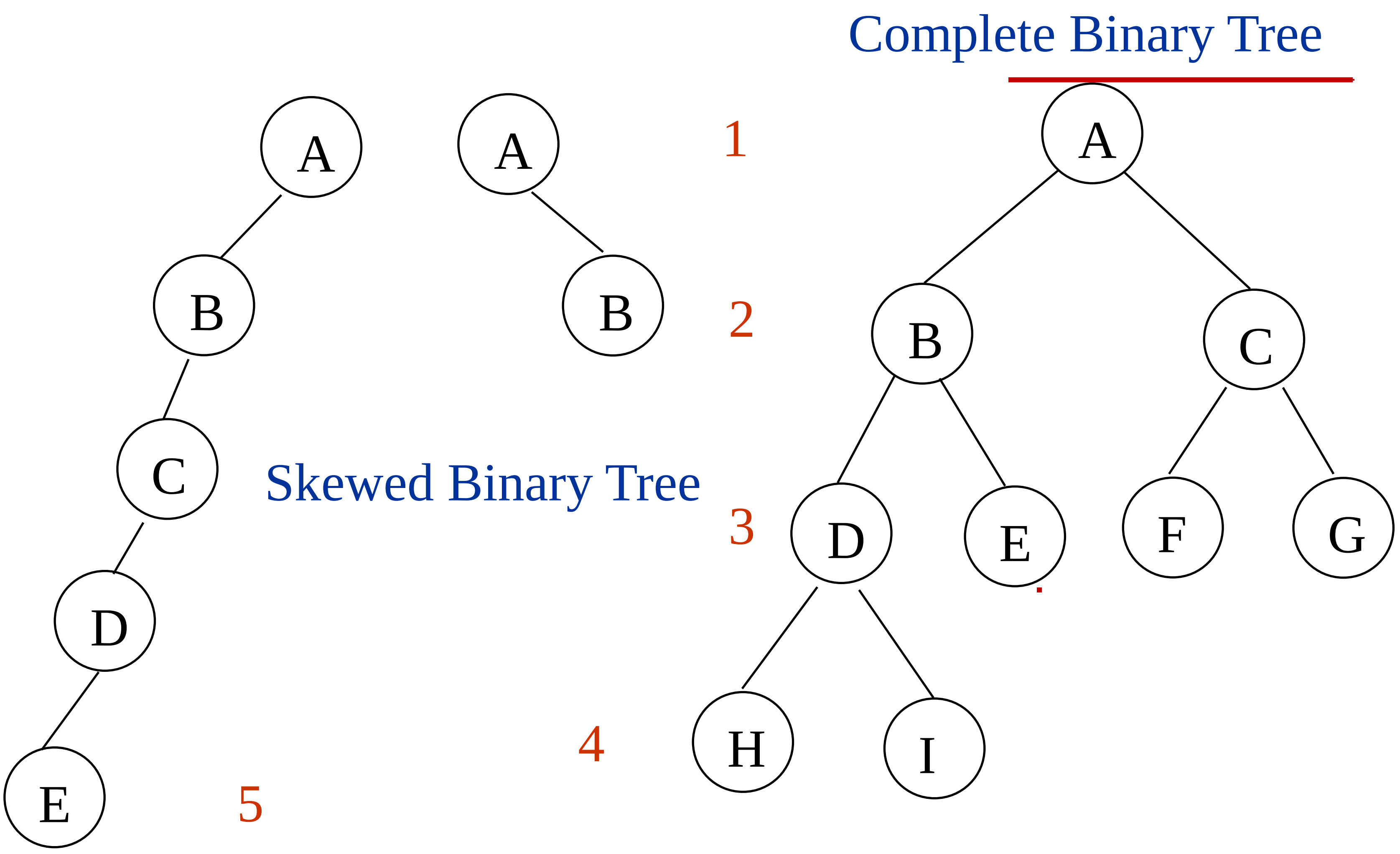
proof:

Let n and B denote the total number of nodes & branches in T.

Let n_0, n_1, n_2 represent the nodes with no children, single child, and two children respectively.

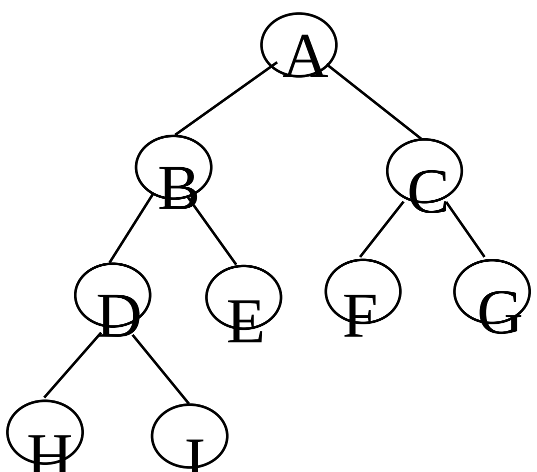
$n= n_0+n_1+n_2$, $B+1=n$, $B=n_1+2n_2$ ==> $n_1+2n_2+1= n$,
$n_1+2n_2+1= n_0+n_1+n_2$ ==> $n_0=n_2+1$

Samples of Trees

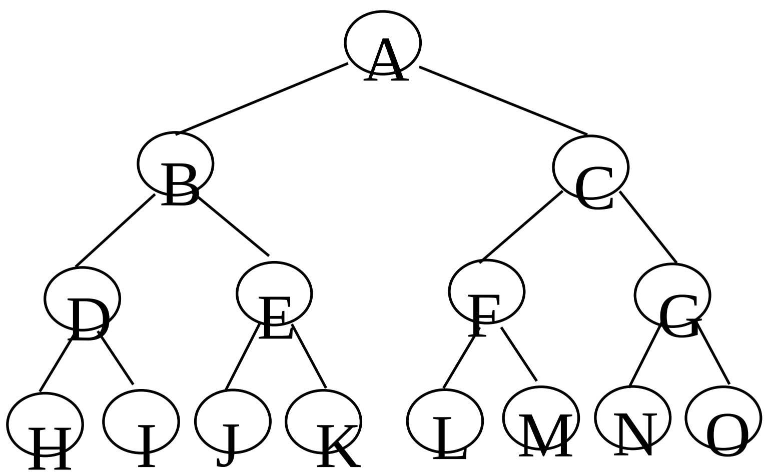


Full BT VS Complete BT

- A full binary tree of depth *k* is a binary tree of depth *k* having $2^k - 1$ nodes, $k>=0$.
- A binary tree with *n* nodes and depth *k* is complete *iff* all of its nodes are filled except possibly the last level, where in the last level the nodes are filled from left to right .



Complete binary tree

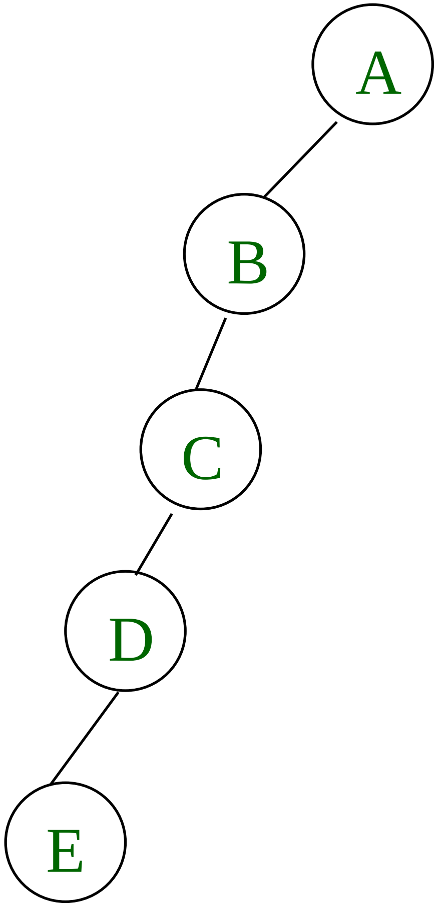


Full binary tree of depth 4

Binary Tree Representations

- If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1<=i<=n$, we have:
  – *parent*(i) is at $i/2$ if $i!=1$. If $i=1$, i is at the root and has no parent.
  – *left_child*(i) is at $2i$ if $2i<=n$. If $2i>n$, then i has no left child.
  – *right_child*(i) is at $2i+1$ if $2i+1<=n$. If $2i+1>n$, then i has no right child.

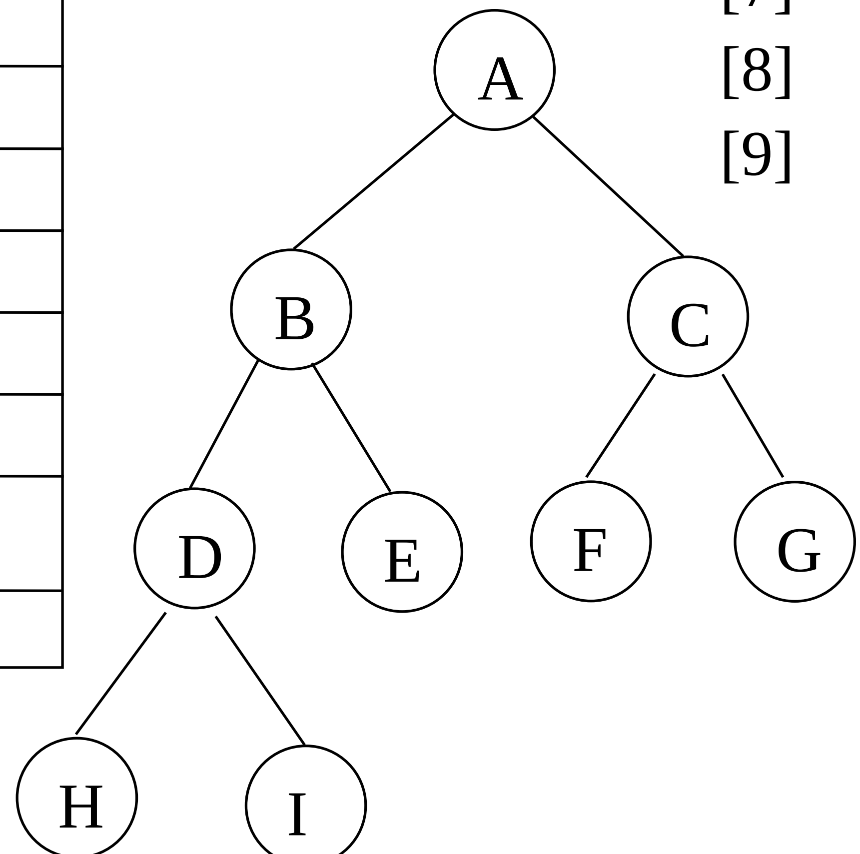
Sequential Representation



[1]	A
[2]	B
[3]	--
[4]	C
[5]	--
[6]	--
[7]	--
[8]	D
[9]	--
.	.
[16]	E

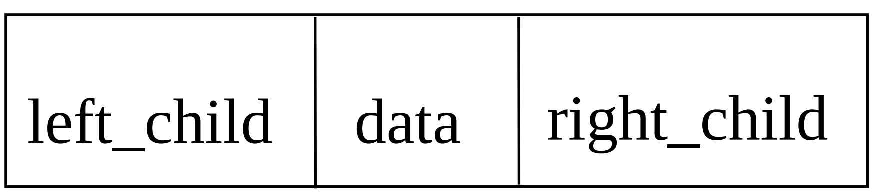
(1) waste space
(2) insertion/deletion
problem

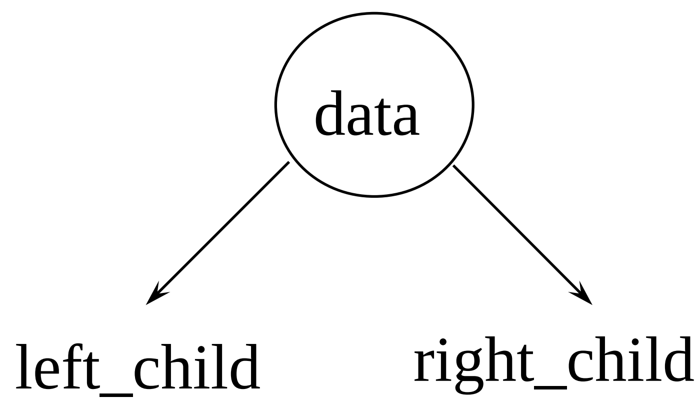
[1]	A
[2]	B
[3]	C
[4]	D
[5]	E
[6]	F
[7]	G
[8]	H
[9]	I



Linked Representation

```
class node {
 int data;
 node *left_child, *right_child;
};
```

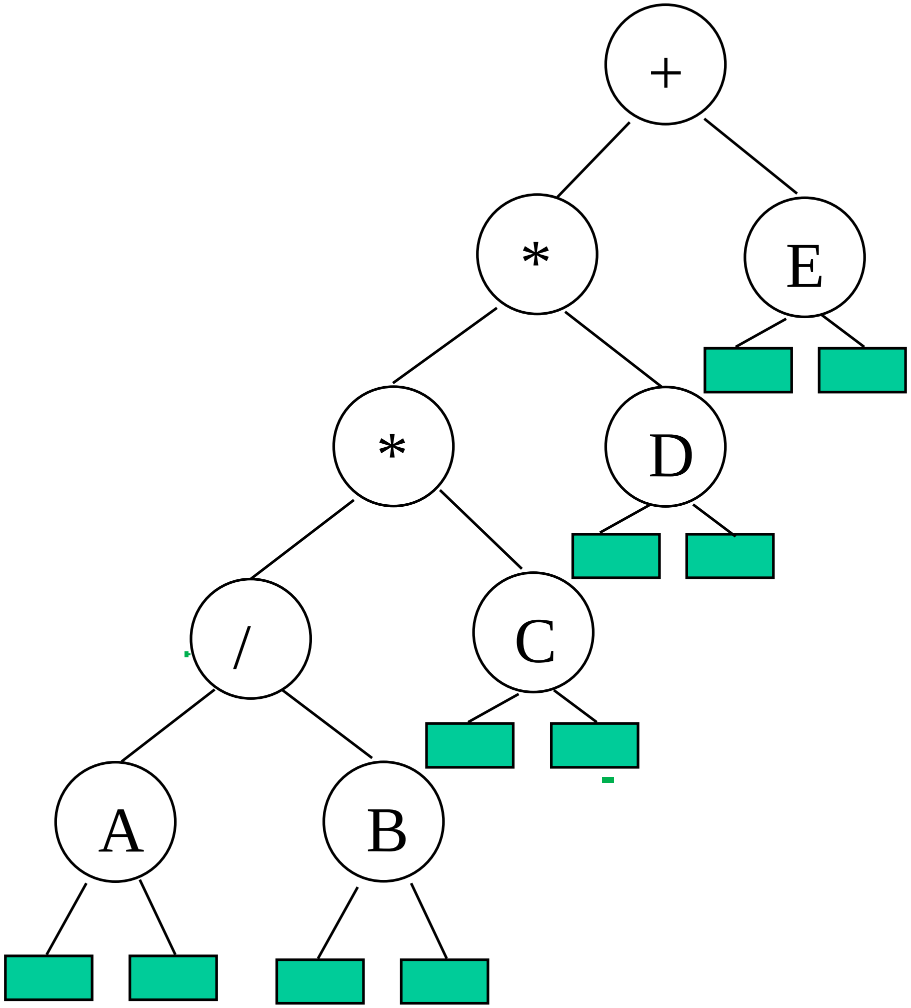




Binary Tree Traversals

- Let L, N, and R stand for moving left, visiting the node, and moving right.
- There are six possible combinations of traversal
  – LNR, LRN, NLR, NRL, RNL, RLN
- Adopt convention that we traverse left before right, only 3 traversals remain
  – LNR, LRN, NLR
  – inorder, postorder, preorder

Arithmetic Expression Using BT



inorder traversal

A / B * C * D + E

infix expression

preorder traversal

+ * * / A B C D E

prefix expression

postorder traversal

A B / C * D * E +

postfix expression

level order traversal

+ * E * D / C A B

Inorder Traversal (recursive version)

```
void inorder(struct node *root)
/* inorder tree traversal */
{
    if (root) //if(root!=NULL)
{
        inorder(root->left);
        printf(“%d”,root->data);
        indorder(root->right);
    }
}
```

Preorder Traversal (recursive version)

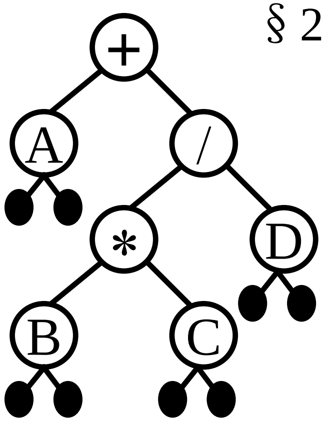
```
void preorder(struct node *root)
/* preorder tree traversal */
{
    if (root) //if(root!=NULL)
    {
        printf("%d",root->data);
        preorder(root->left);
        predorder(root->right);
    }
}
```

Postorder Traversal (recursive version)

```
void postorder(struct node *root)
/* postorder tree traversal */
{
     if (root)  //if(root!=NULL)
     {
          postorder(root->left);
          postdorder(root->right);
          printf(“%d”,root->data);
     }
}
```

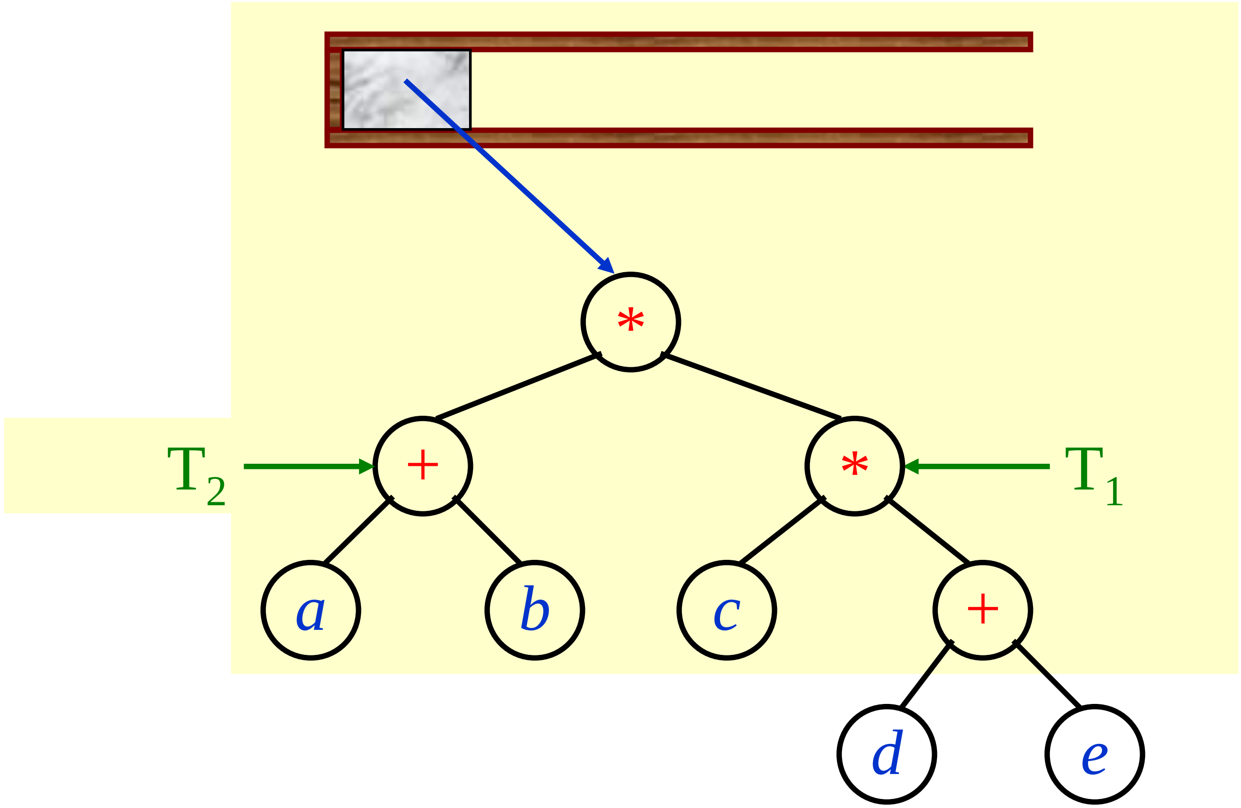
❖ Expression Trees (syntax trees)

〖Example〗 Given an infix expression:
$A + B * C / D$



☞ Constructing an Expression Tree
(from postfix expression)

〖Example〗 $(a + b) * (c * (d + e)) = a\ b + c\ d\ e + * *$

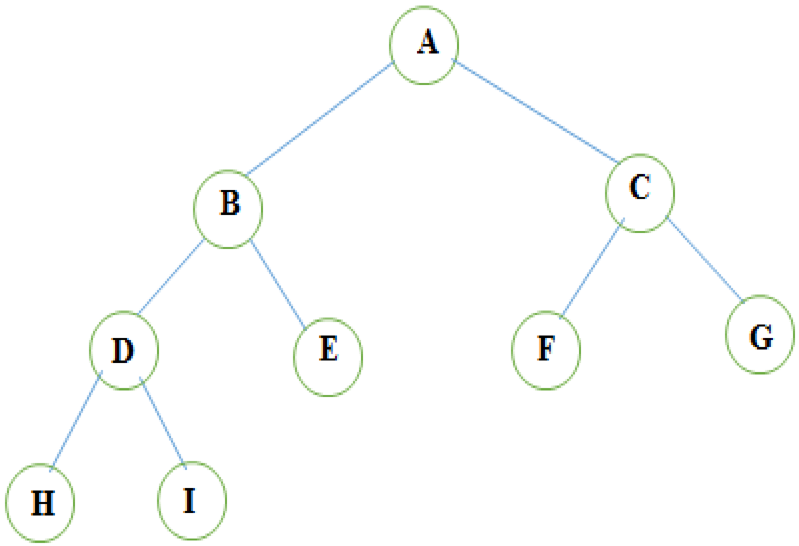


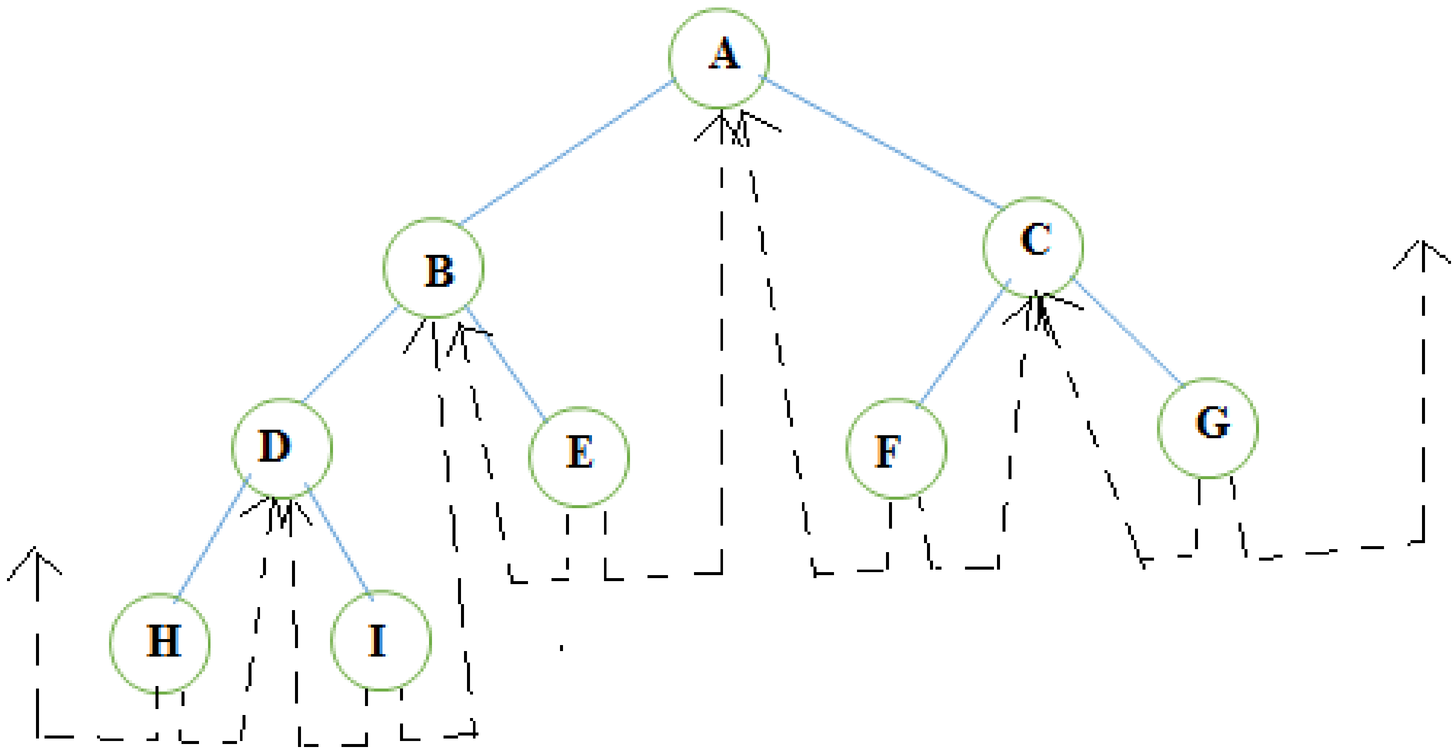
Rule 1: If root->lcl is null, replace it with a pointer to the inorder predecessor of Tree.

Rule 2: If root->rcl is null, replace it with a pointer to the inorder successor of Tree.

Rule 3: There must not be any loose threads. Therefore a threaded binary tree must have a head node of which the left child points to the first node.

```
struct  ThreadedTreeNode {
      int                             LeftThread;   /* if it is TRUE, then Left */
      ThreadedTree                    Left;      /* is a thread, not a child ptr.   */
      ElementType  Element;
      int                             RightThread; /* if it is TRUE, then Right */
      ThreadedTree                    Right;    /* is a thread, not a child ptr.   */
};
```


A
B
C
D
E
F
G
H
I


A
B
C
D
E
F
G
H
I

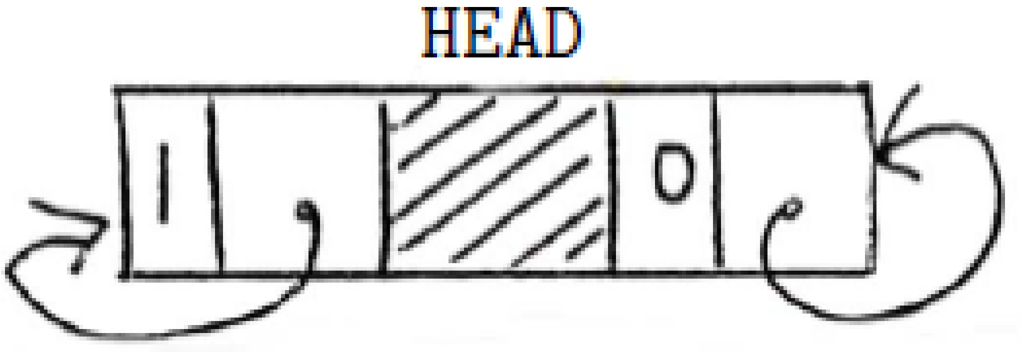
- **Assume that ptr is an arbitrary node in a threaded binary tree, then the following constraints hold:**

❑ **If** ptr->leftThread = TRUE or 1**, then** ptr->lcl **contains thread.**

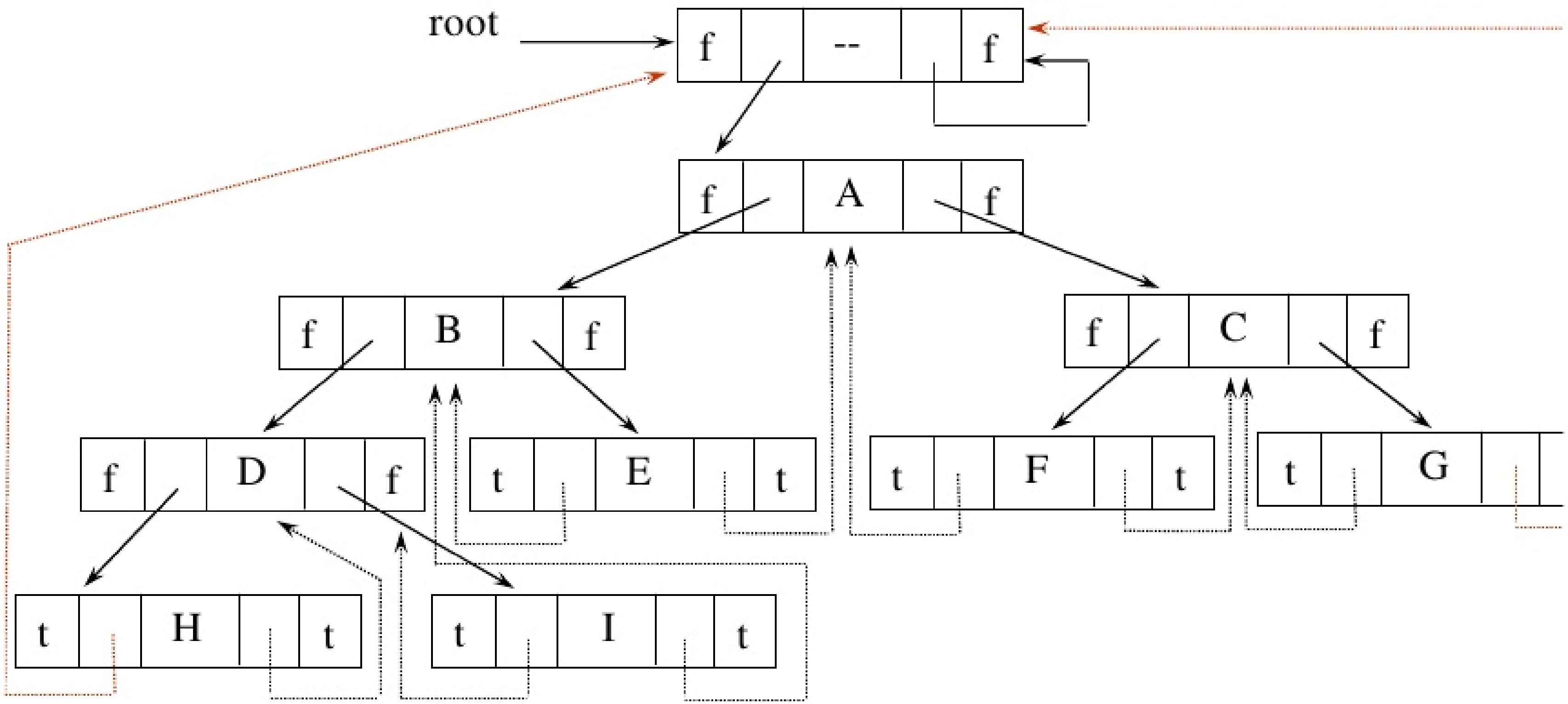
❑ **If** ptr->rightThread = TRUE or 1**, then** ptr->rcl **contains thread.**

- **Traditionally,** root->rlink = root **and** root->rightThread = 0 **for any threaded binary tree.**
- **The root points to the header node of the tree, while** root->llink **points to the start of the first node of the actual tree.**
- **The loose thread from the right most node and the left most node is handled by having them pointed to the header node.**

Empty Threaded BT

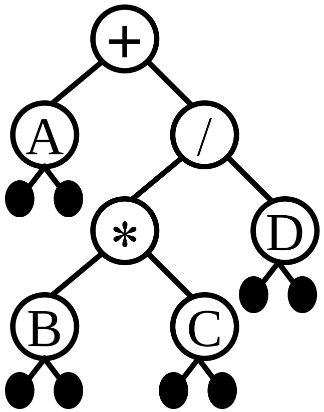


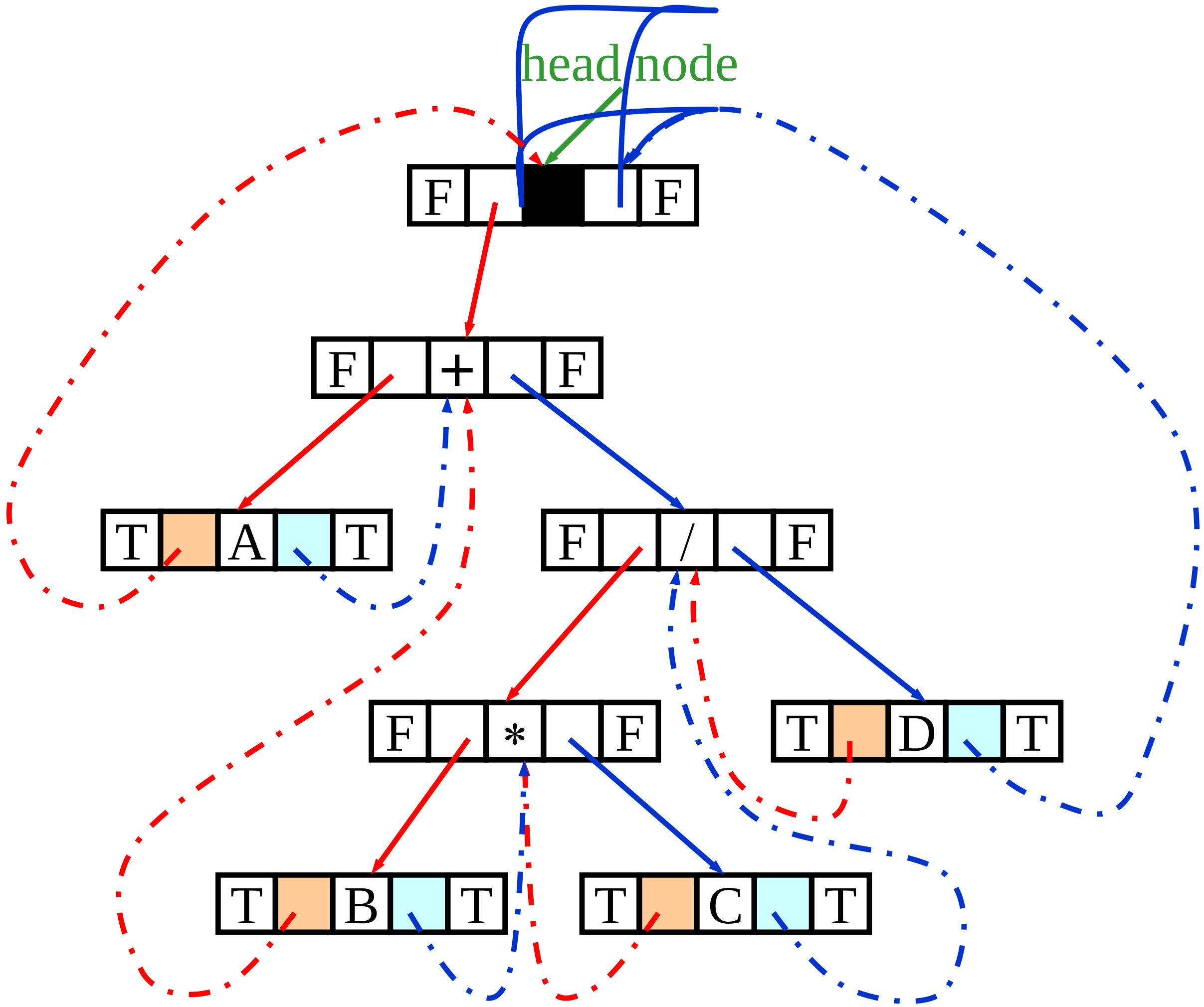
Memory Representation of A Threaded BT



〖Example〗 Given the syntax tree of an expression (infix)

$$A + B * C / D$$





```
void tinorder(node *root)
{
 node *temp=root;
 for(;;)
 {
  temp=in_suc(temp);
  if(temp==root)
      break;
printf(" %d "temp->info);
 }
}
```

```
node *in_suc(node *root)
{
 node *temp;
 temp=root->rlink;
 if(!root->rthread)
 {
  while(!temp->lthread)
      temp=temp->llink;
 }
return temp;
}
```