# Data Structures and Algorithms

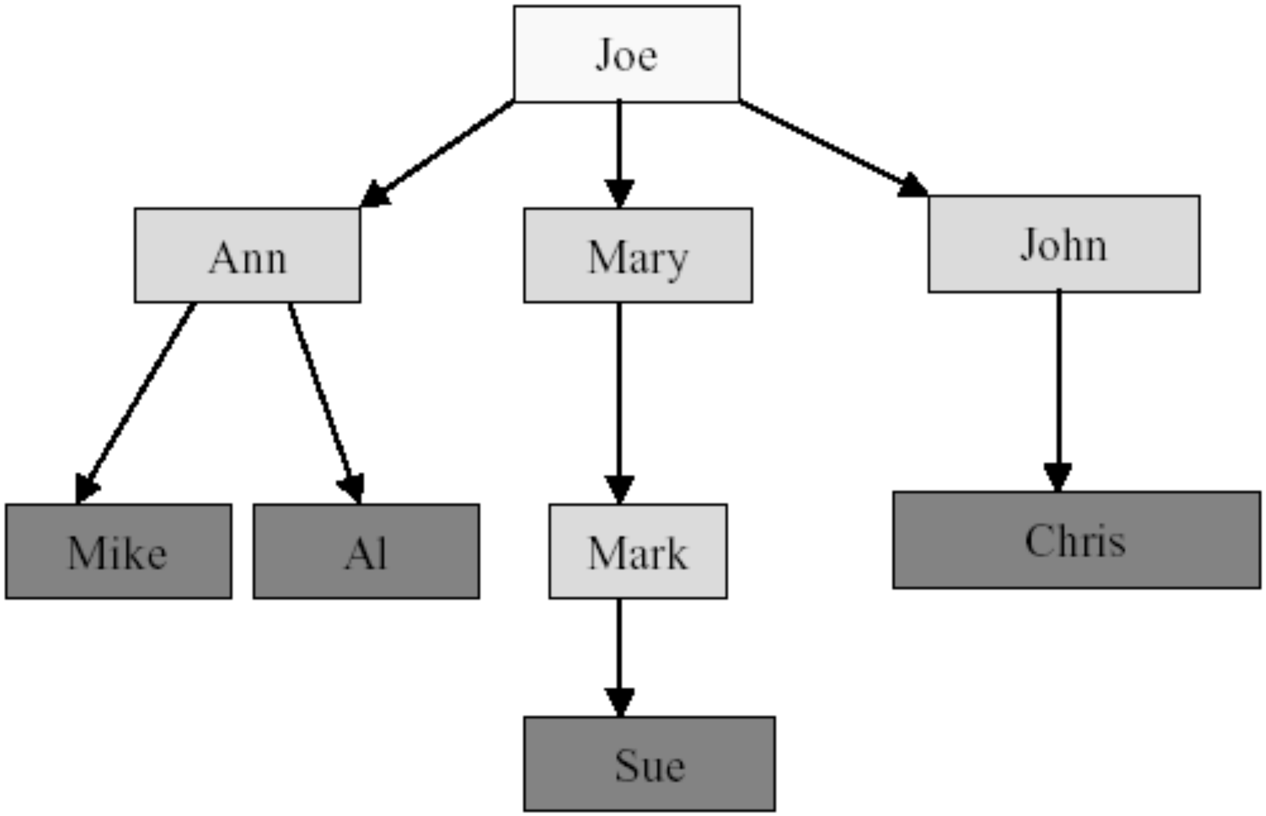
## Lecture 7 – Binary Trees-Basics

Miao Zhang

# Why Do We Need Trees?

- **Lists, Stacks, and Queues are linear relationships (serially ordered data)**
  - $(e_1, e_2, e_3, \ldots, e_n)$
  - Days of week
  - Months in a year
  - Students in a class
- **Information often contains hierarchical relationships (hierarchically ordered data)**
  - Joe's descendants
  - Corporate structure
  - Government Subdivisions
  - Software structure
  - File directories or folders
  - Moves in a game
- **Tree structures permit both efficient access and update to large collections of data**

# Joe’s Descendants




What are other examples of hierarchically ordered data?

Hierarchies in organizations

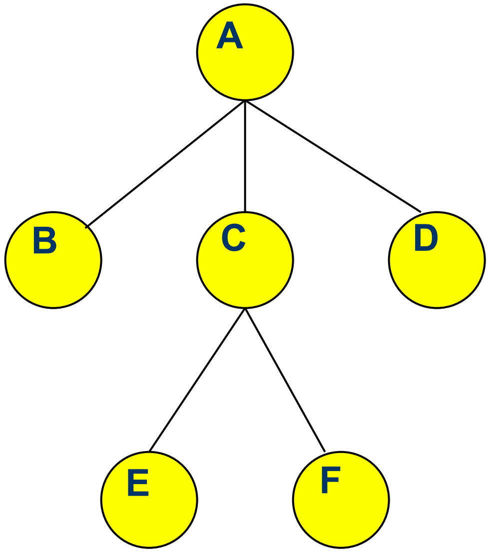
# Tree Terminology/Jargon

- root
- nodes and edges
- leaves

- parent, children, siblings
- ancestors, descendants

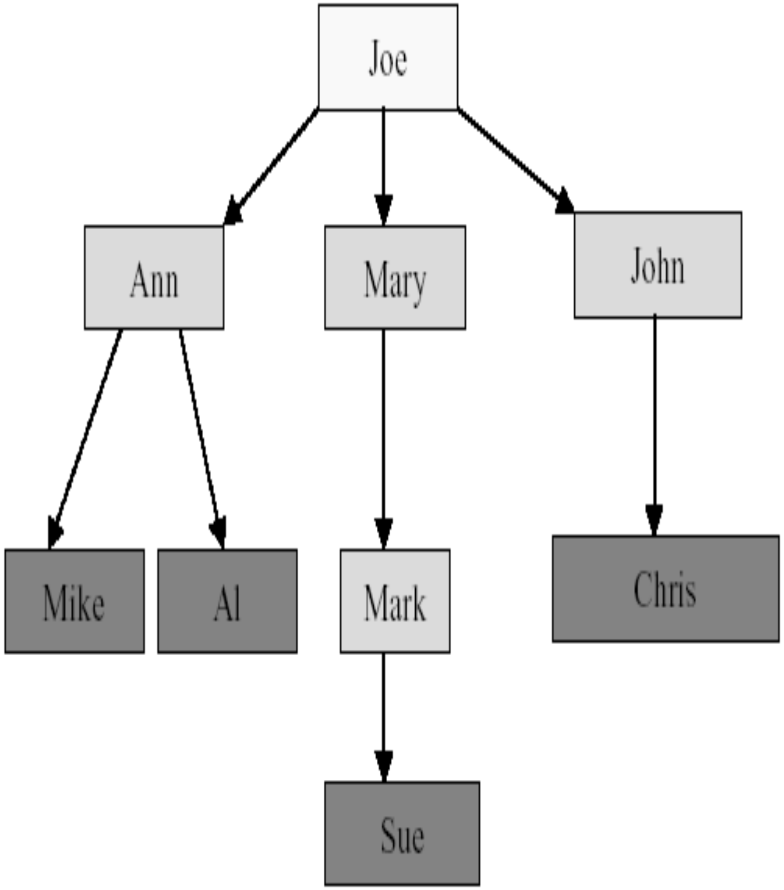
- subtrees

- path, path length
- height, depth



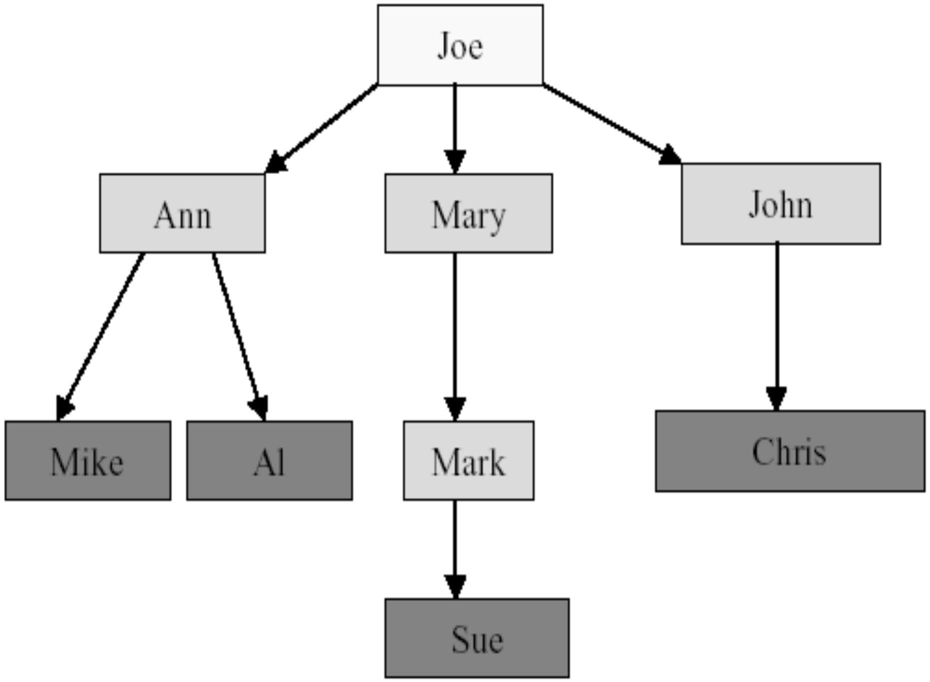
# Tree Terminology/Jargon

- The element at the top of the hierarchy is the **root**.
- Elements next in the hierarchy are the **children** of the root.
- Elements next in the hierarchy are the **grandchildren** of the root and so on.
- Elements that have empty children are **leaves**.



# Tree Terminology/Jargon

- Leaves, Parent, Grandparent, Siblings, Ancestors, Descendents




Leaves = {Mike,Al,Sue,Chris}

Parent(Mary) = Joe

Grandparent(Sue) = Mary

Siblings(Mary) = {Ann,John}

Ancestors(Mike) = {Ann,Joe}

Descendents(Mary)={Mark,Sue}

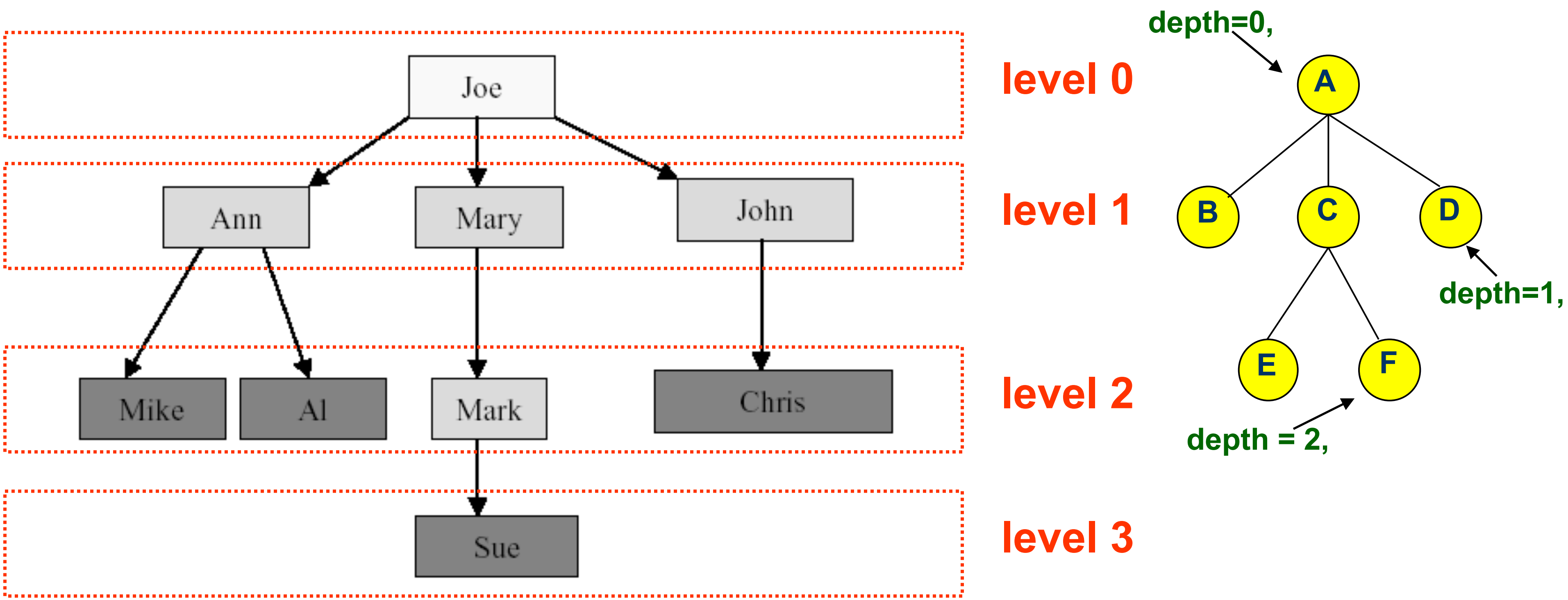
# Tree Terminology/Jargon

**If n1, n2, ..., nk is a sequence of nodes in the tree**

- **Length of a path = number of edges**
- **Depth of a node N = length of path from root to N**
- **Depth of tree = depth of deepest node**
- **Height of tree = one more than the depth of the deepest node in the tree**
- **A** leaf **node is any node that has two empty children.**
- **An** internal **node is any node that has at least one non-empty child.**

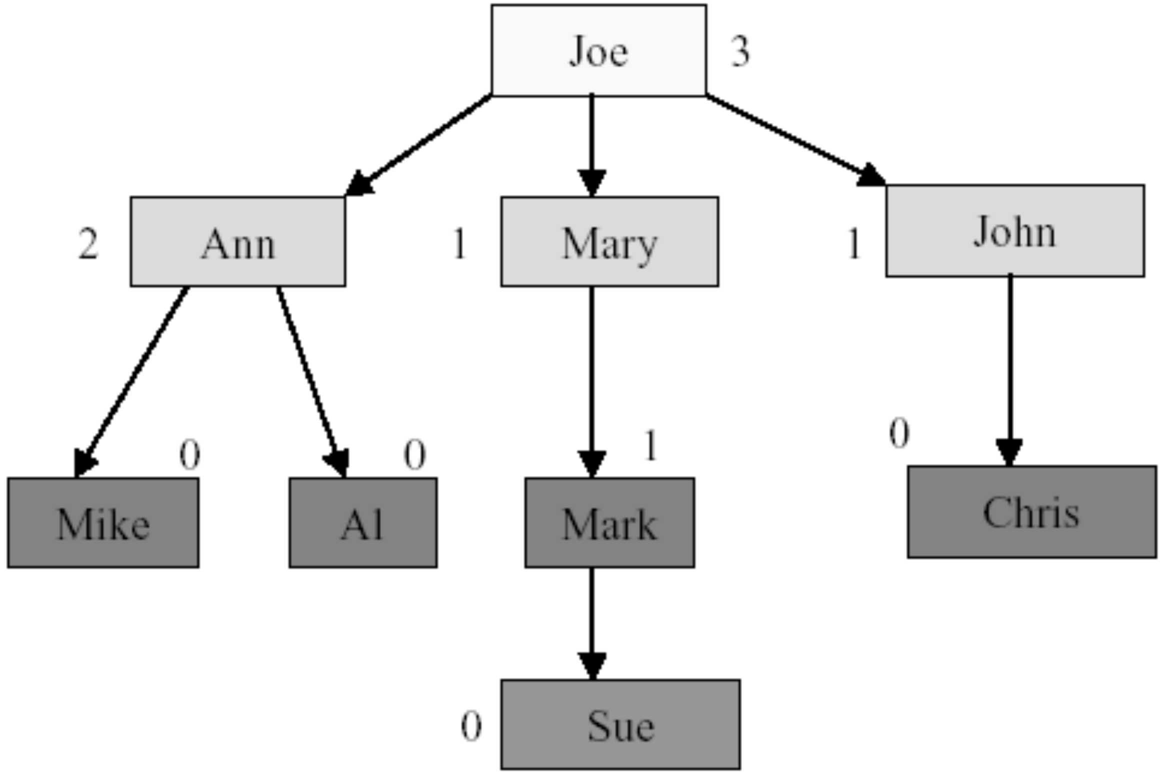
# Levels and Height

- Root is at level 0 and its children are at level 1.
- depth = number of levels



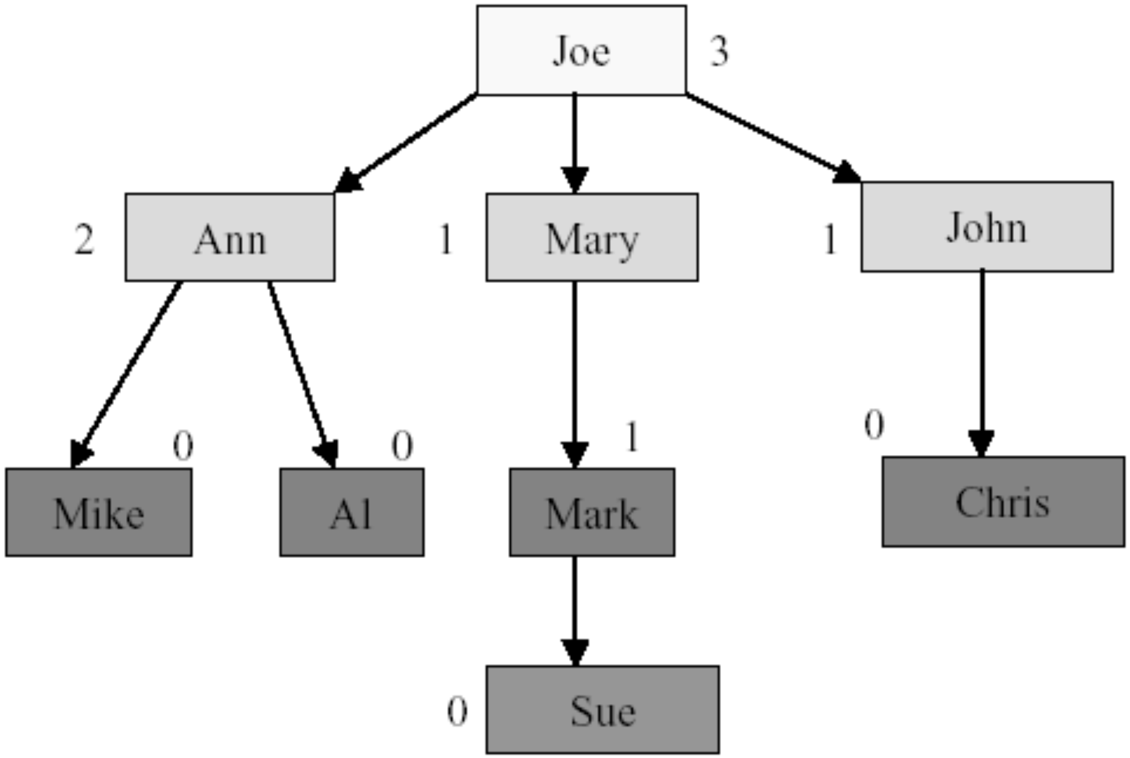
# Node Degree

- **Node degree is the number of children the node has**



# Tree Degree

➢ **Tree degree is the maximum of node degrees**




**tree degree = 3**

# Definition and Tree Trivia

- A tree is a set of **nodes**, i.e., either
  - it's an empty set of nodes, or
  - it has one node called the <u>root</u> from which zero or more **subtrees** descend
- Two nodes in a tree have at most one **path** between them
- Can a non-zero path from node N reach node N again?
  - No! Trees can never have cycles (loops)

# Tree Properties

1. The number of nodes in the tree is one more than the sum of node degrees .

2. If the degree of the tree is m, there are $m^i$ nodes at most on the level i.

3. If the depth of the tree is h-1, the tree with m degree has a maximum of $\frac{m^h-1}{m-1}$ nodes.

4. If the tree with m degree has n nodes, the least height of the tree is $\lceil \log_m(n(m-1)+1 \rceil$

# Tree Properties

1. The number of nodes in the tree is one more than the sum of node degrees .

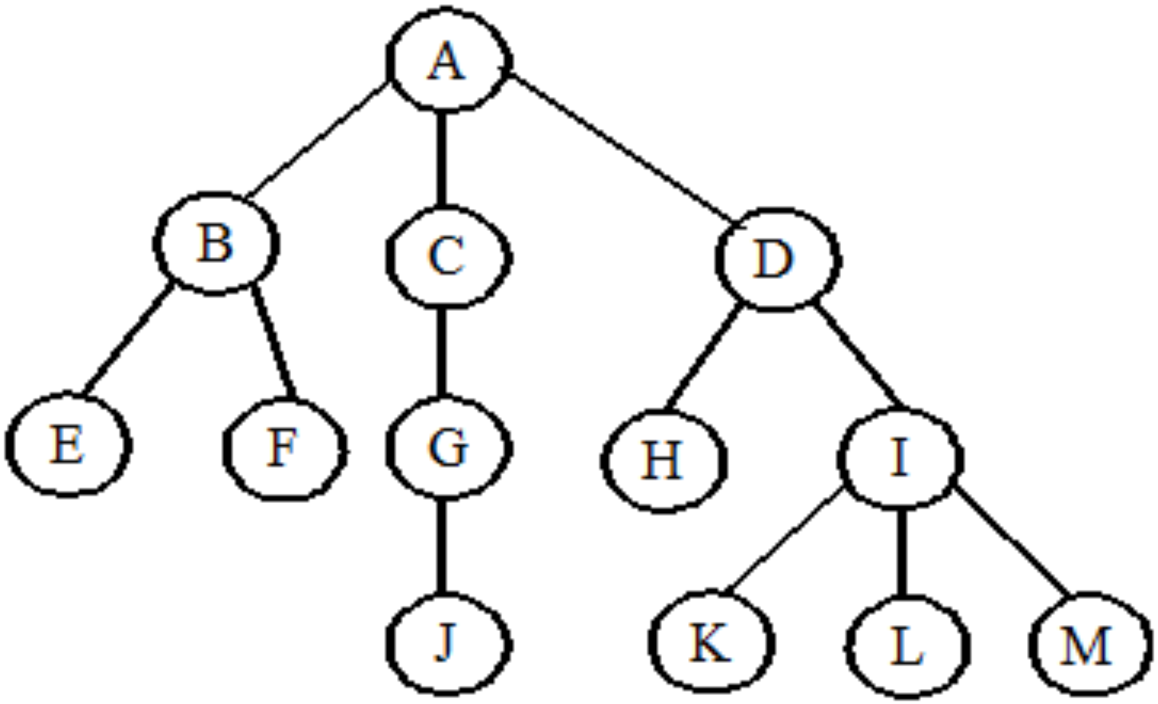
2. If the degree of the tree is m, there are $m^i$ nodes at most on the level i.

3. If the depth of the tree is h-1, the tree with m degree has a maximum of $\frac{m^h-1}{m-1}$ nodes.

4. If the tree with m degree has n nodes, the least height of the tree is $\lceil \log_m(n(m-1)+1 \rceil$

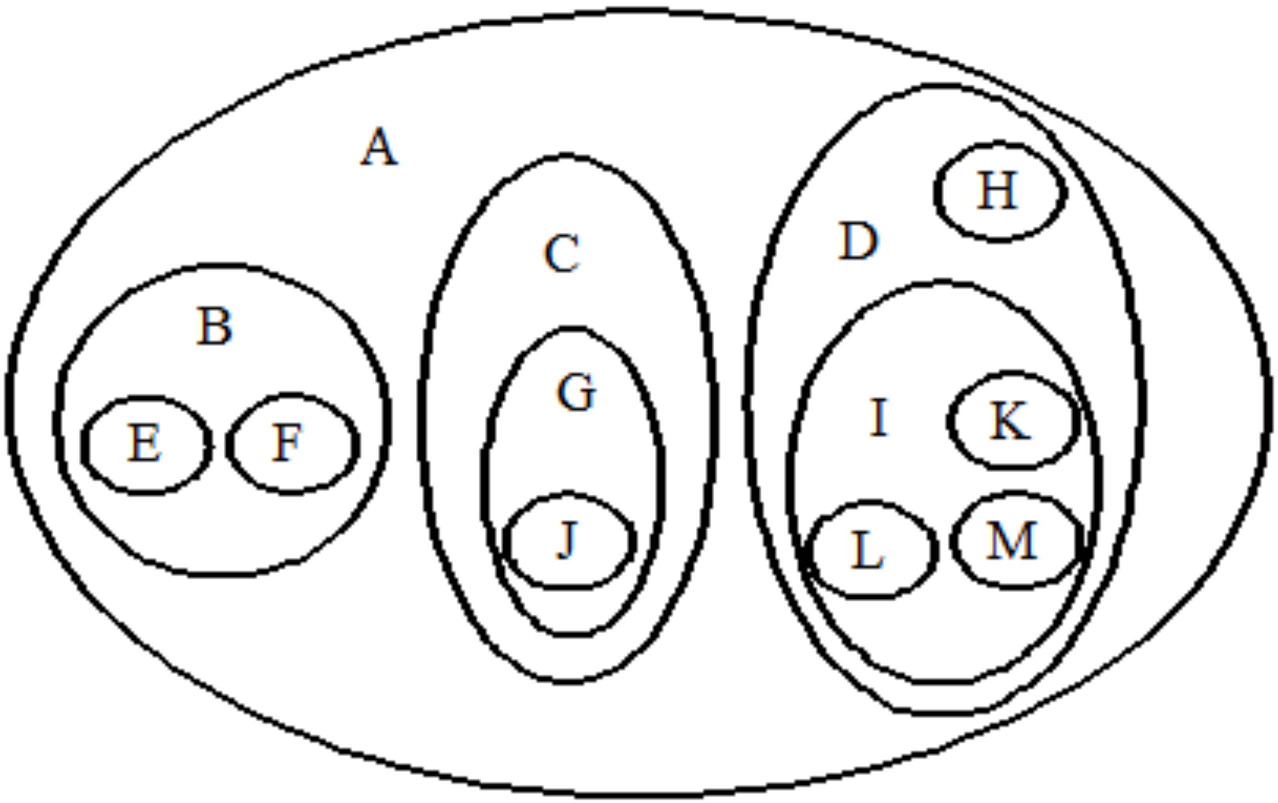
# Logical representation of Tree

- **(1) Tree shaped**



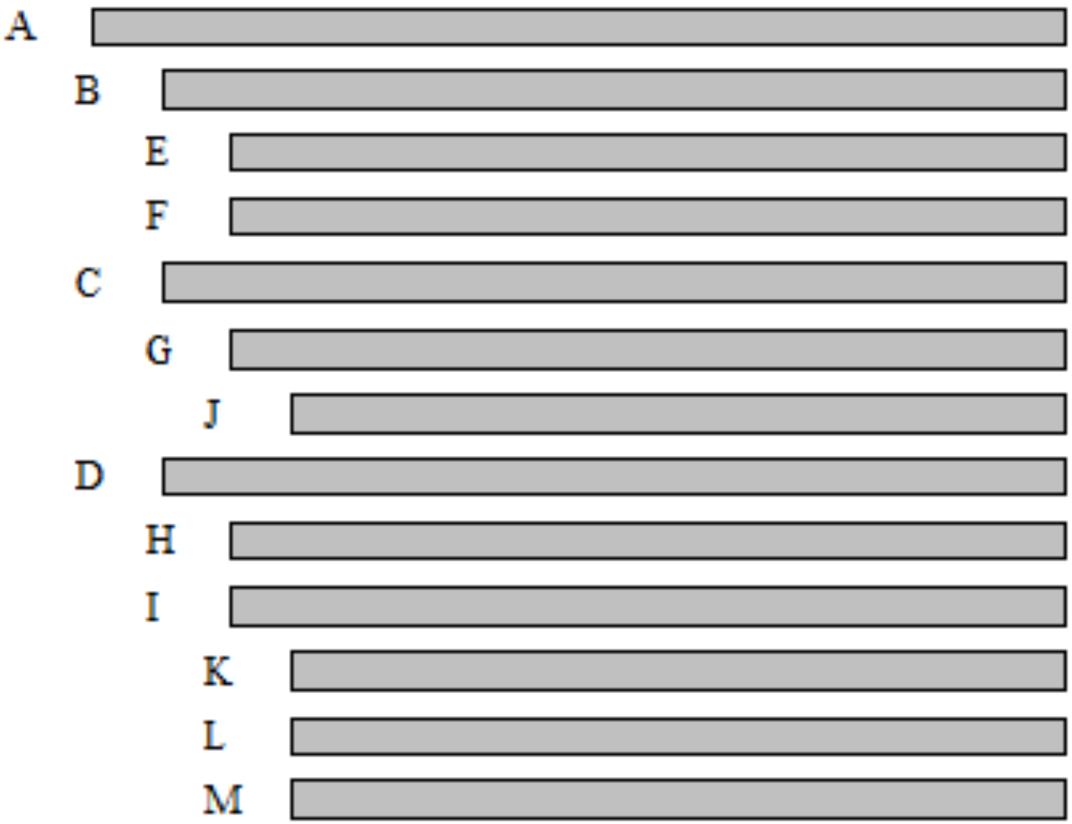
# Logical representation of Tree

- (2) Venn Diagram



# Logical representation of Tree

- (3) Invagination



# Logical representation of Tree

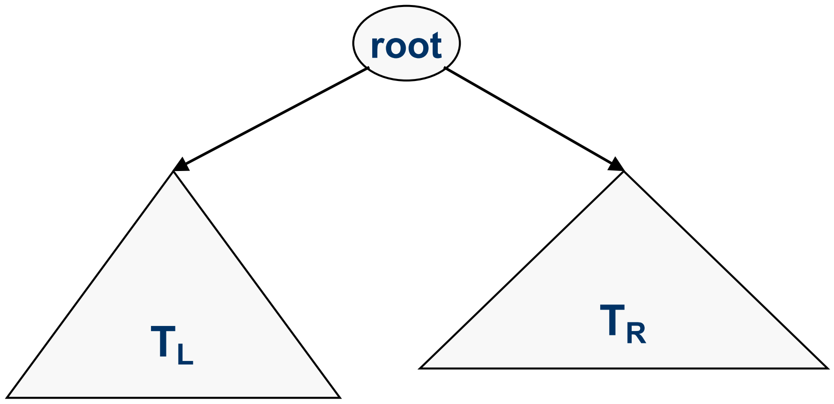
- (4)Generalized list

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

# Binary Trees

# Binary Trees

- A *binary tree* is a tree in which no node can have more than two children




A binary tree consisting of a root and two subtrees $T_L$ and $T_R$, both of which could possibly be empty.

# Binary Tree Terminology

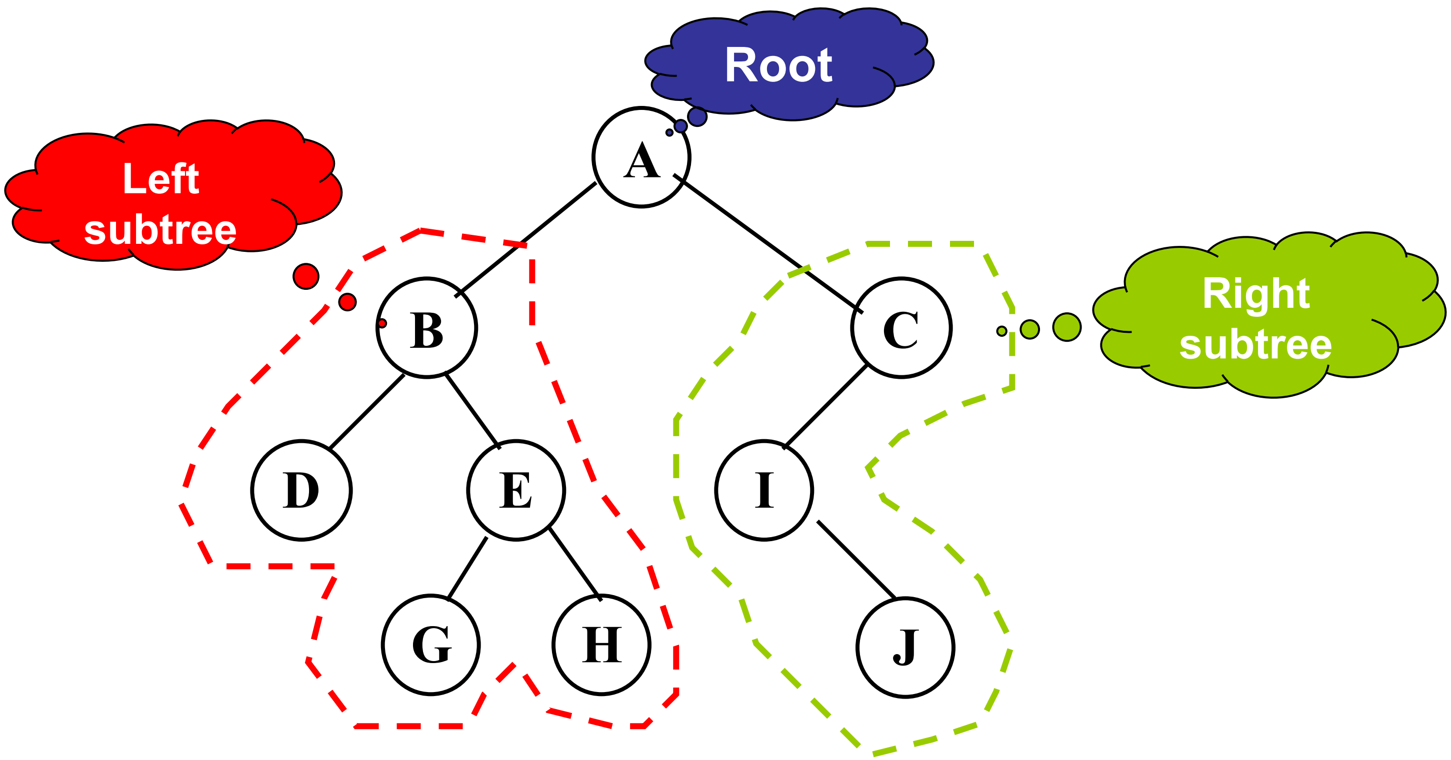
*Left Child* – The left child of node n is a node directly below and to the left of node n in a binary tree.

*Right Child* – The right child of node n is a node directly below and to the right of node n in a binary tree.

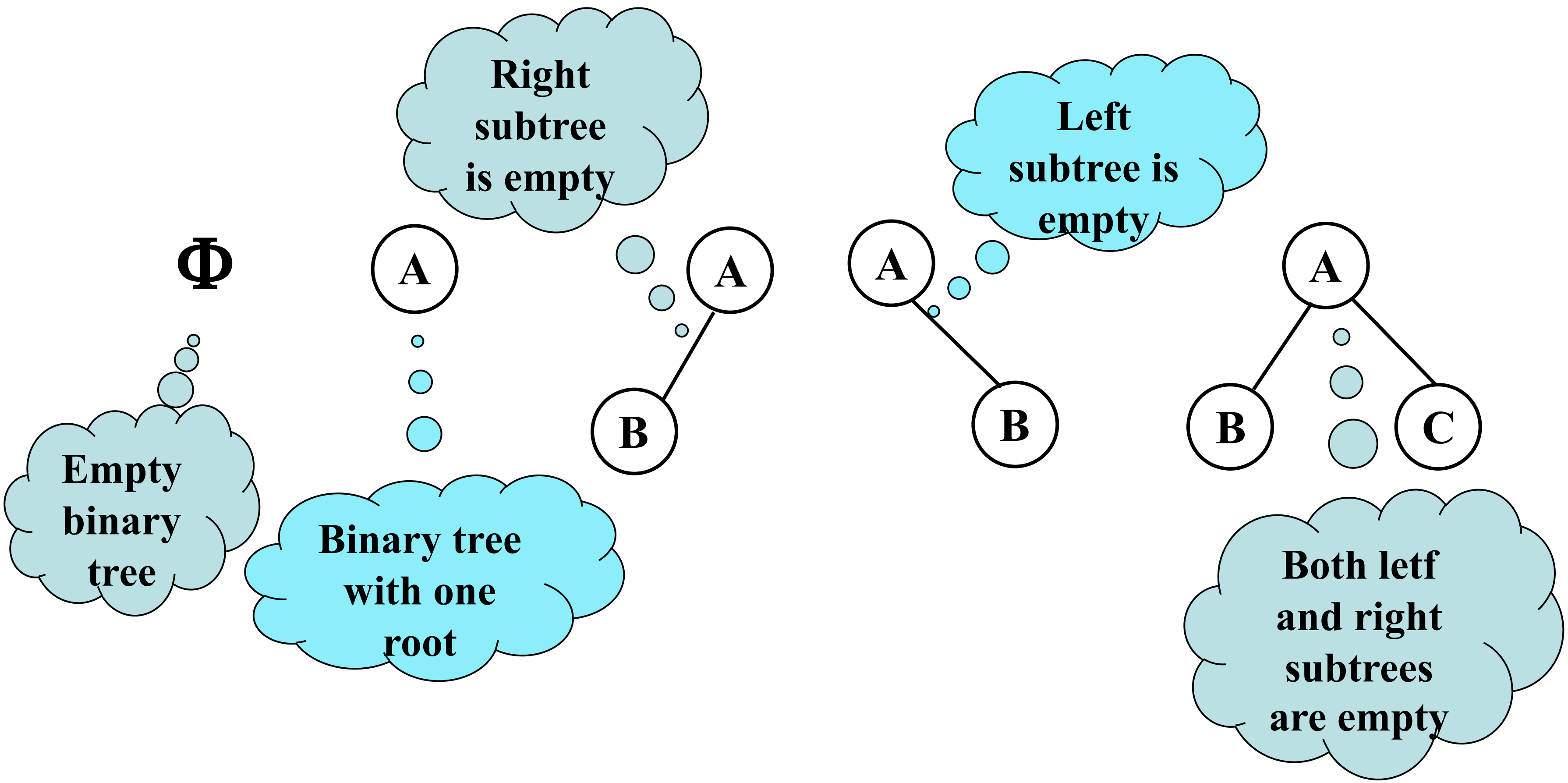
*Left Subtree* – In a binary tree, the left subtree of node n is the left child (if any) of node n plus its descendants.

*Right Subtree* – In a binary tree, the right subtree of node n is the right child (if any) of node n plus its descendants.

# Binary Tree

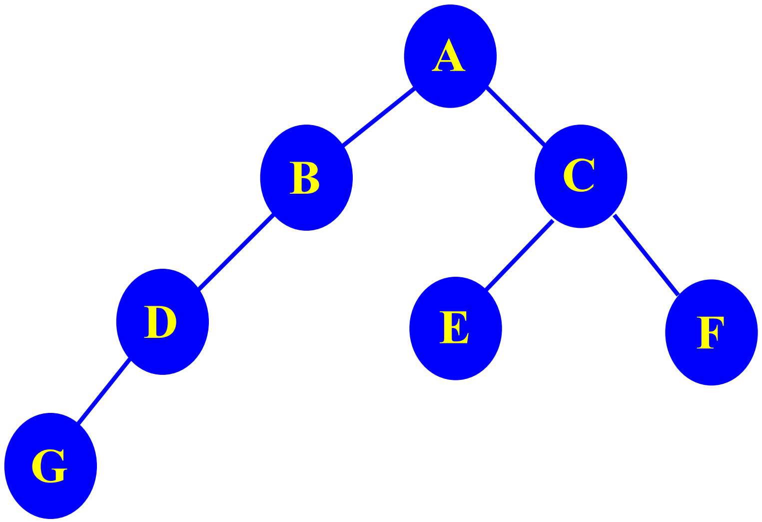


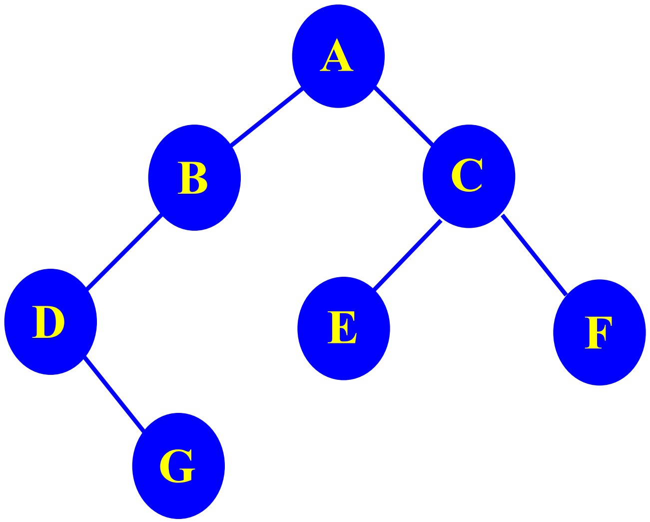
# Binary tree examples



# Binary tree

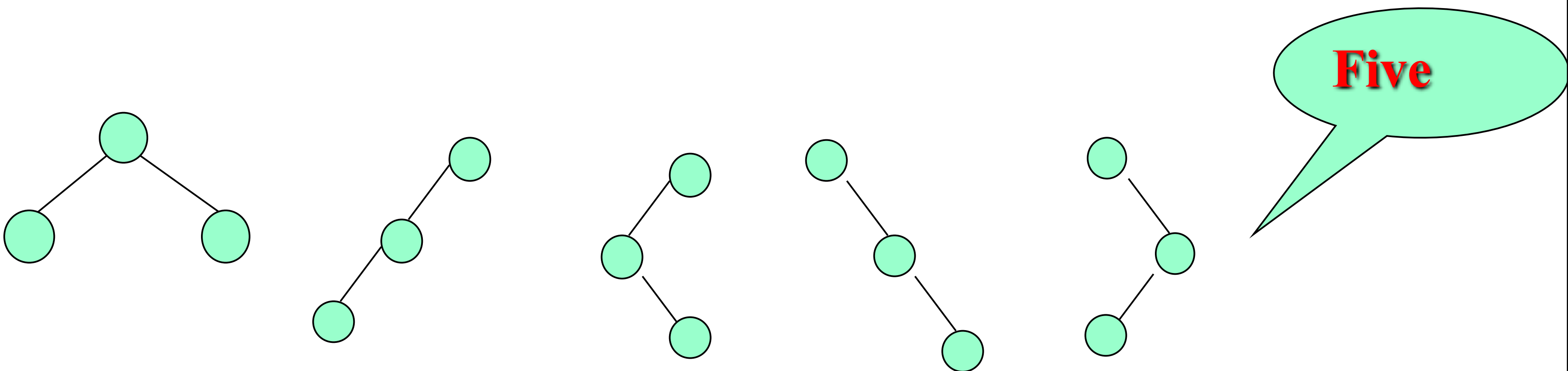
- **Characteristics:**
- **① The node degree is not more than 2.**
- **② The left subtree and right subtree can not be exchanged.(The two trees below are different.)**



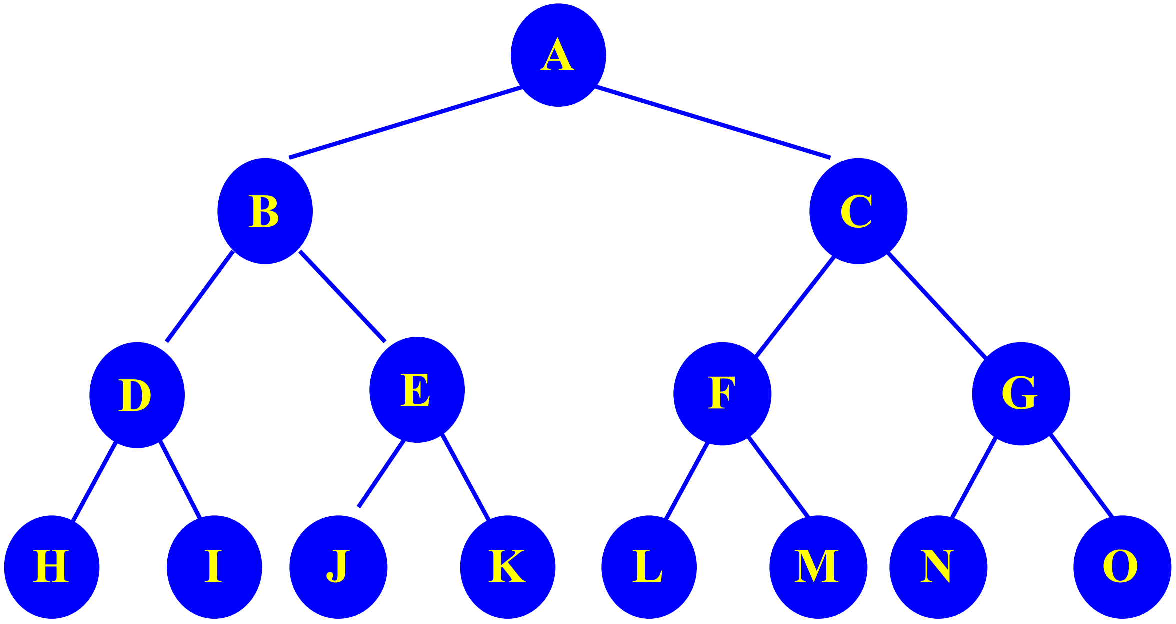
# Binary tree

**Ex.: How many shapes of binary tree with 3 nodes?**



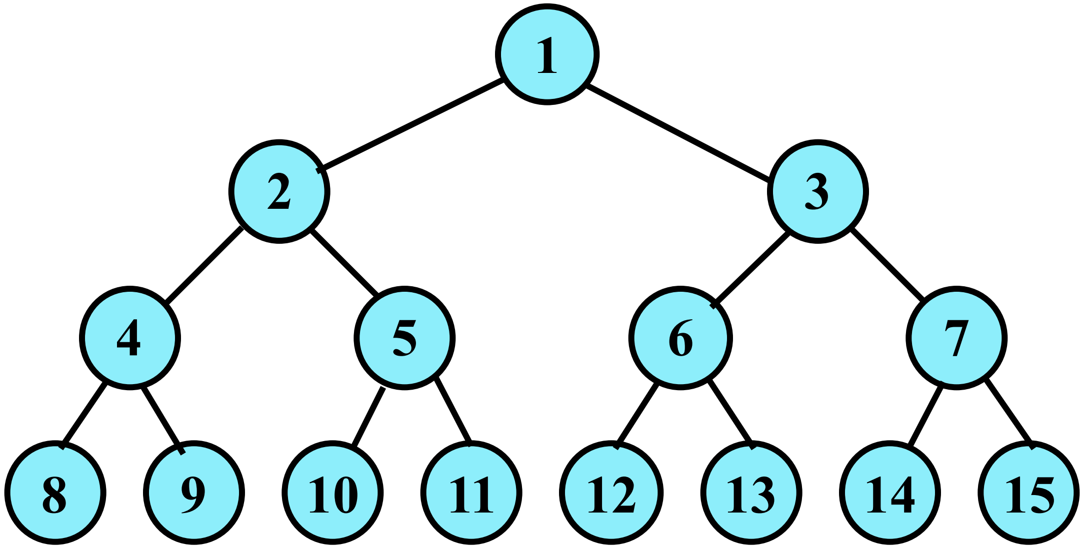
# Full binary tree

- Full binary tree: Any internal node has exactly two non-empty children, and all leaf nodes are on the same level.



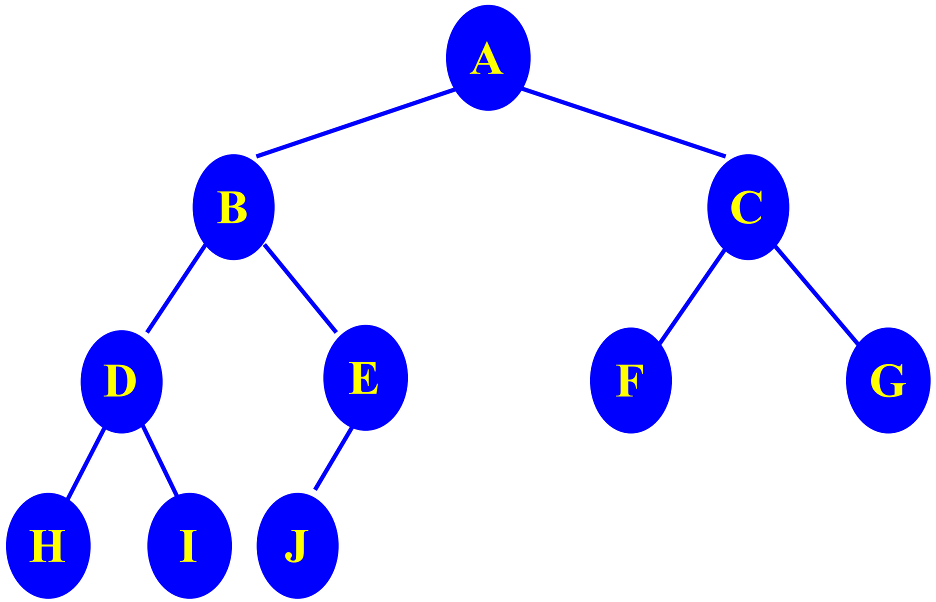
# Characteristics of full binary trees

- The full binary tree with height k has $2^k$-1 nodes.
  - Each level has the maximum number of nodes;
  - The degree of each internal node is two;
  - Leaf nodes are on the same level.

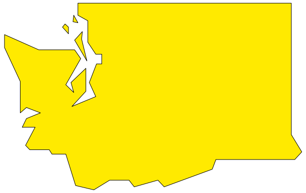


# Complete binary trees

**Complete binary tree:** A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.



# Complete Binary Trees

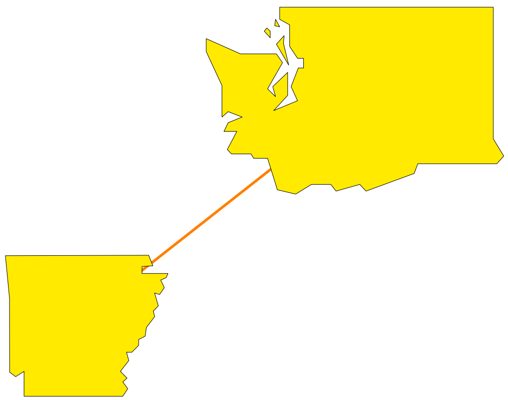


**A complete binary tree is a special kind of binary tree which will be useful to us.**

When a complete binary tree is built, its first node must be the root.

# Complete Binary Trees

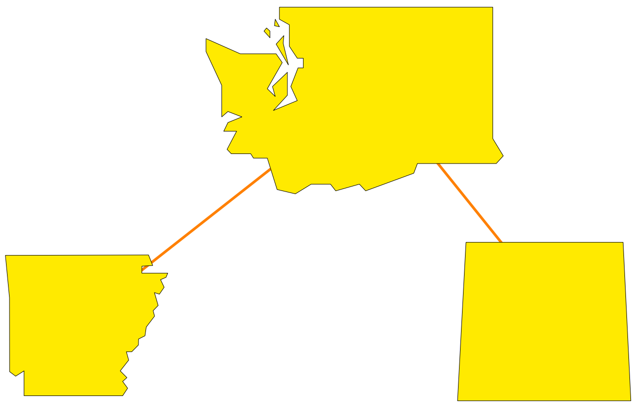
The second node of a complete binary tree is always the left child of the root...



# Complete Binary Trees

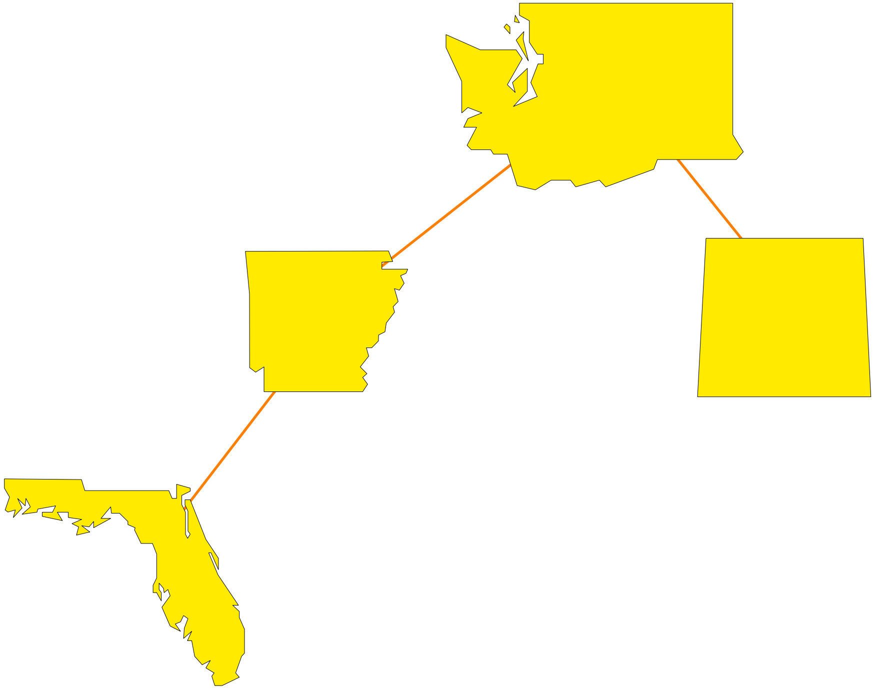
The second node of a complete binary tree is always the left child of the root...

... and the third node is always the right child of the root.



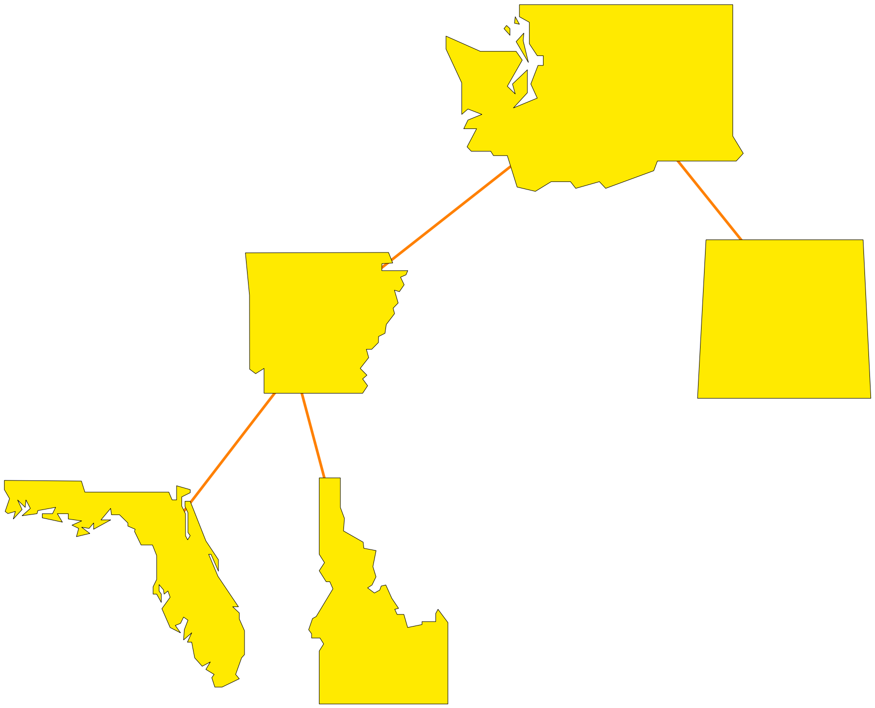
# Complete Binary Trees

The next nodes must always fill the next level from <u>left to right</u>.



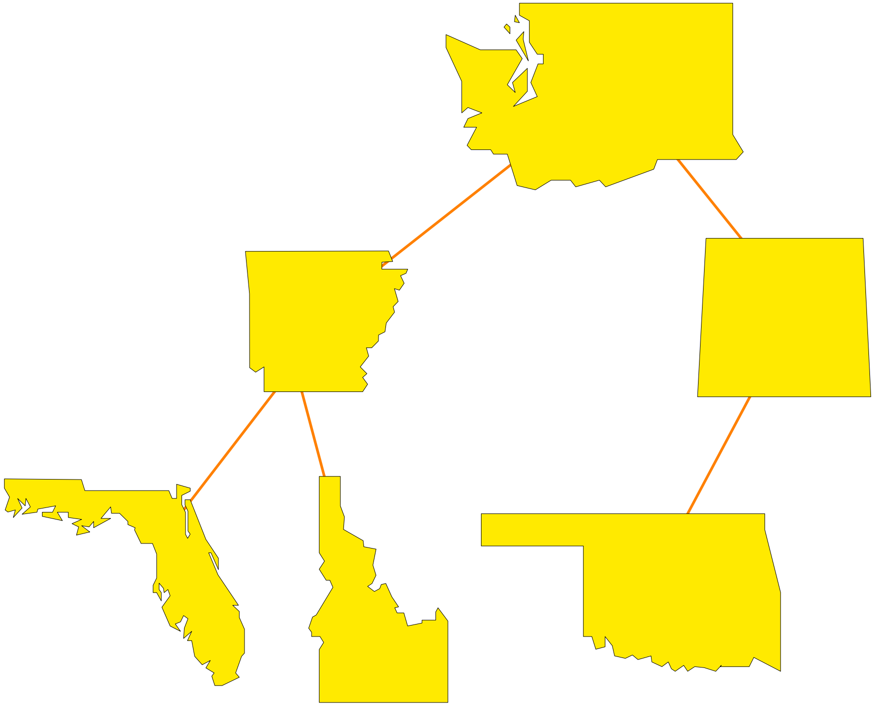
# Complete Binary Trees

The next nodes must always fill the next level from <u>left to right</u>.



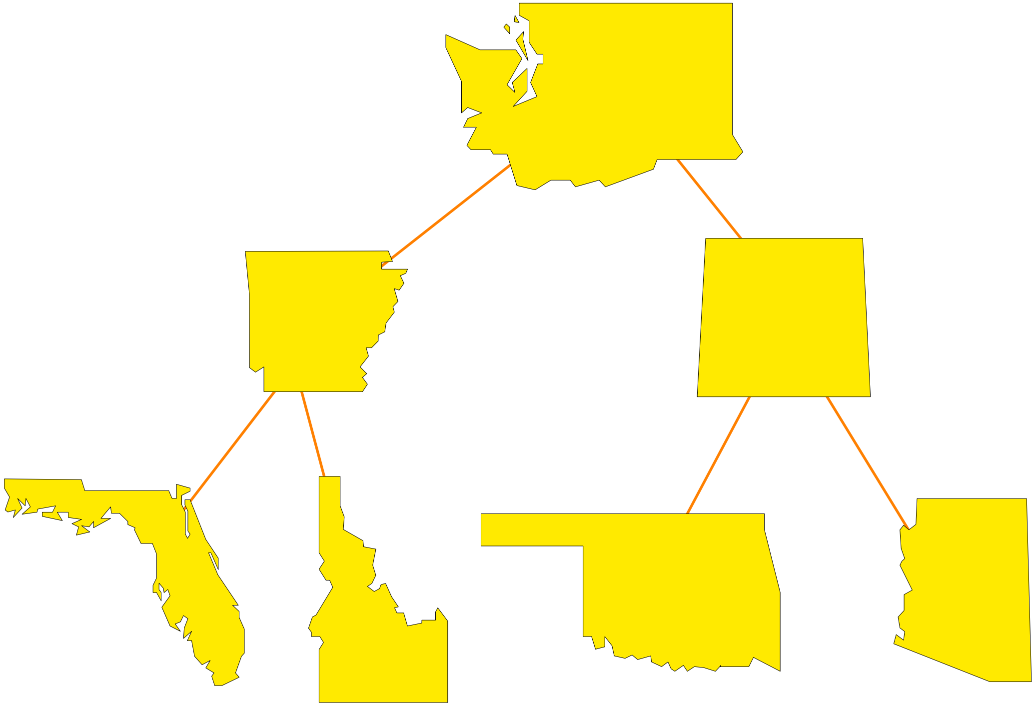
# Complete Binary Trees

**The next nodes must always fill the next level from left to right.**



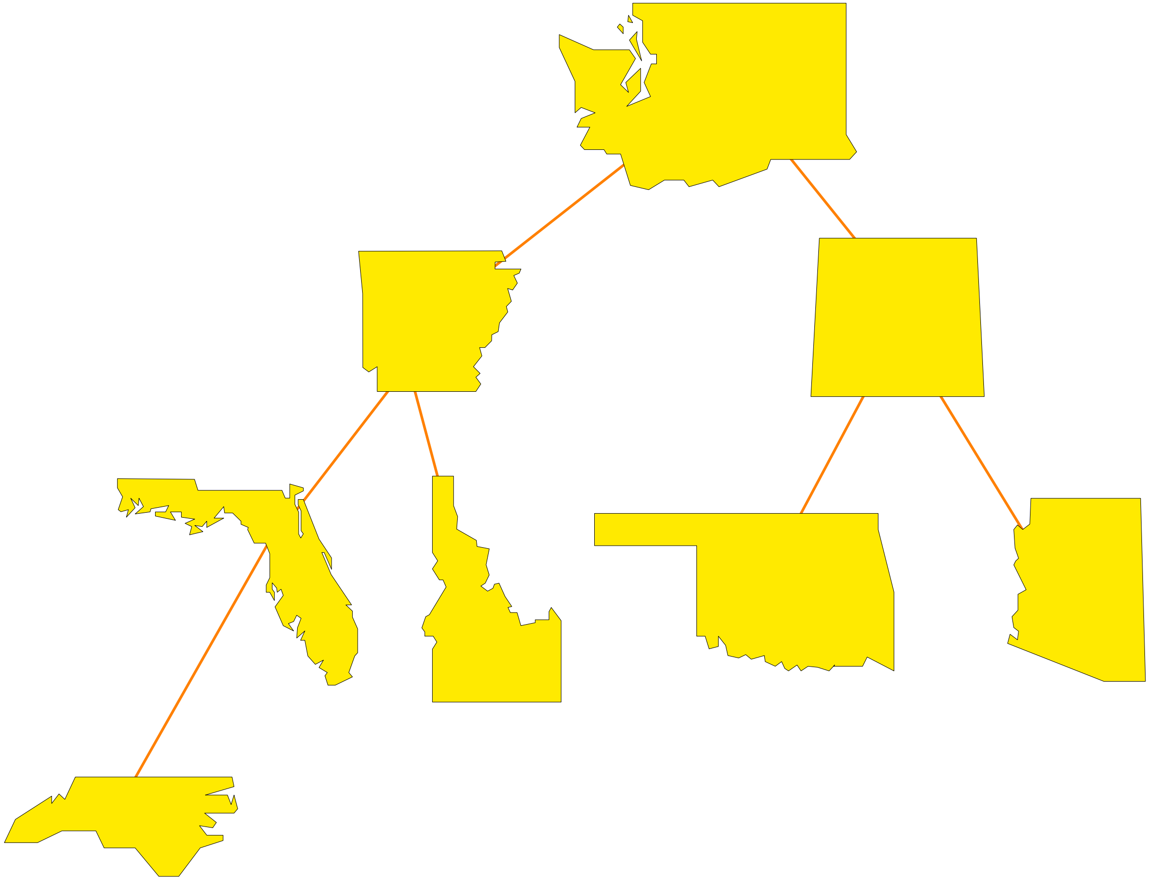
# Complete Binary Trees

**The next nodes must always fill the next level from <u>left to right</u>.**



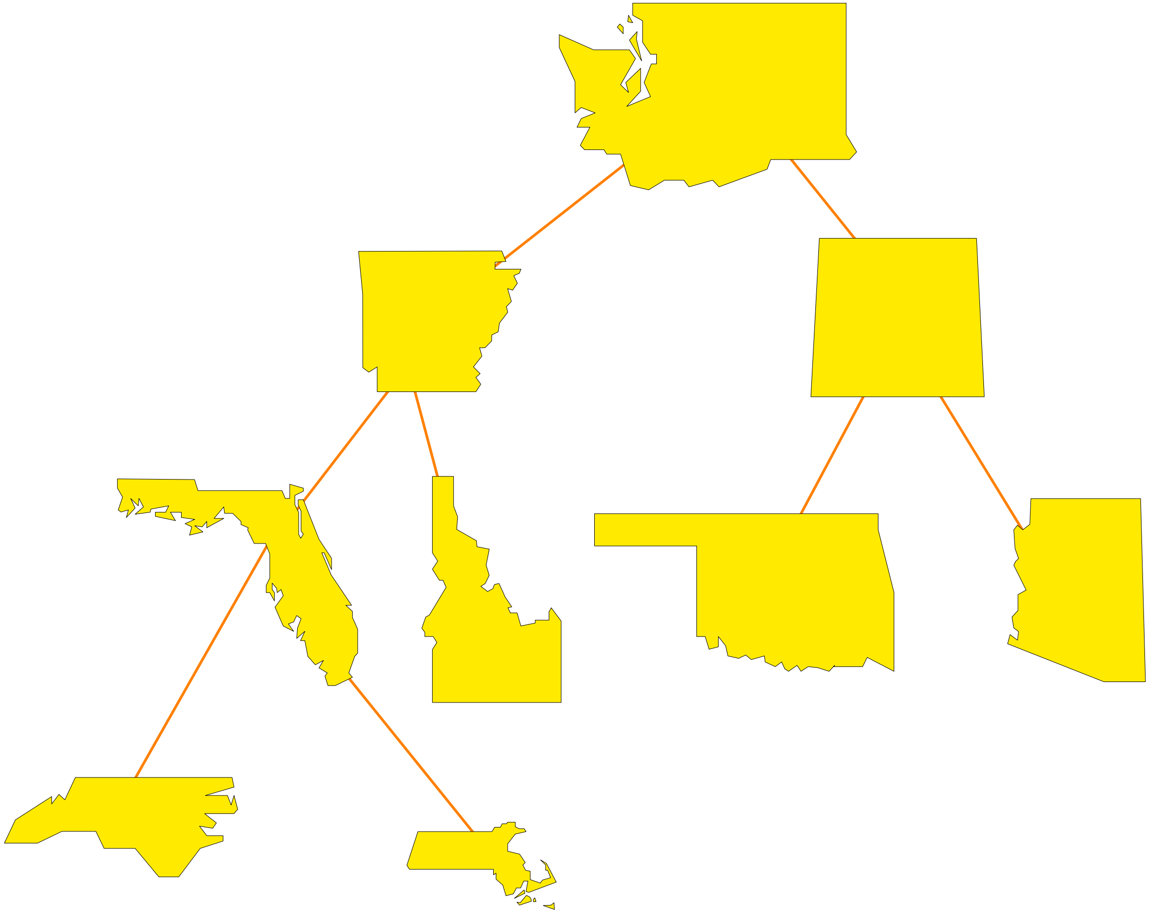
# Complete Binary Trees

**The next nodes must always fill the next level from <u>left to right</u>.**

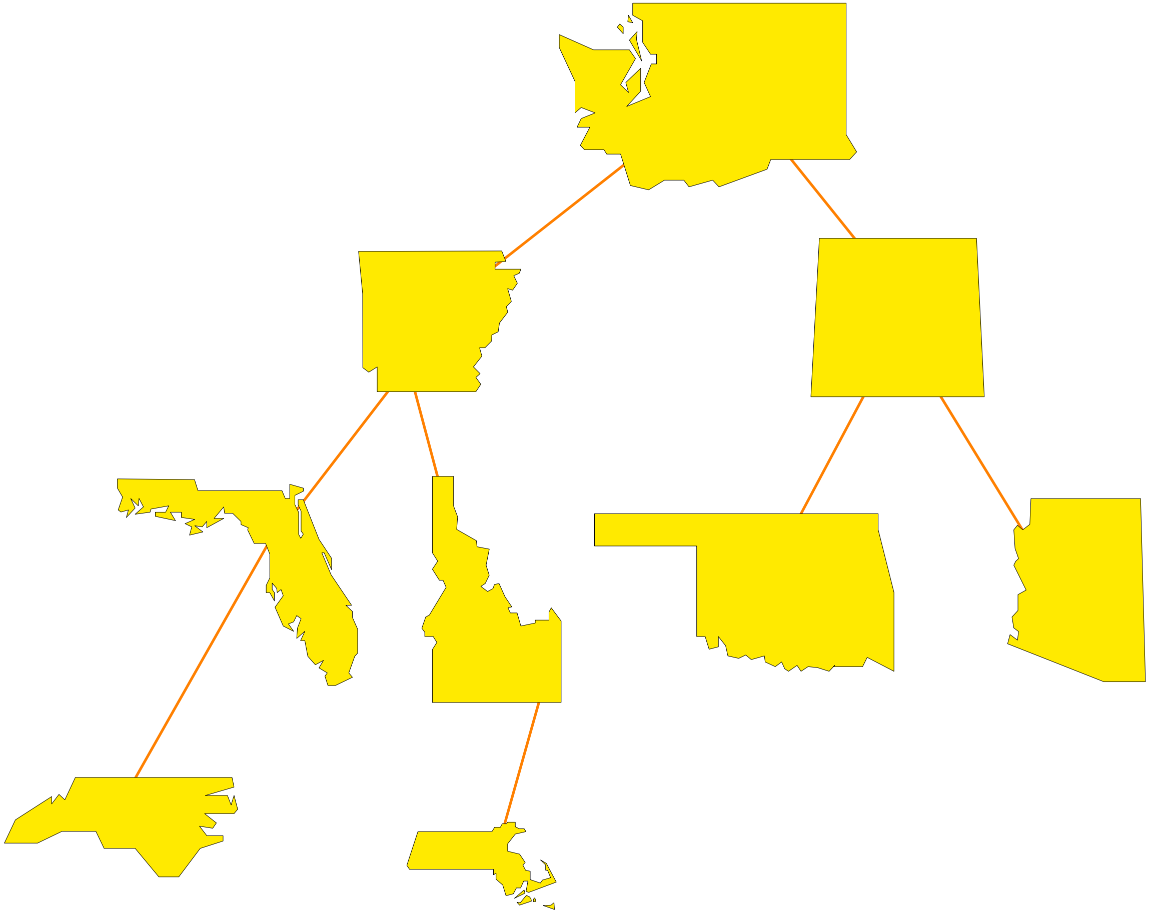


# Complete Binary Trees

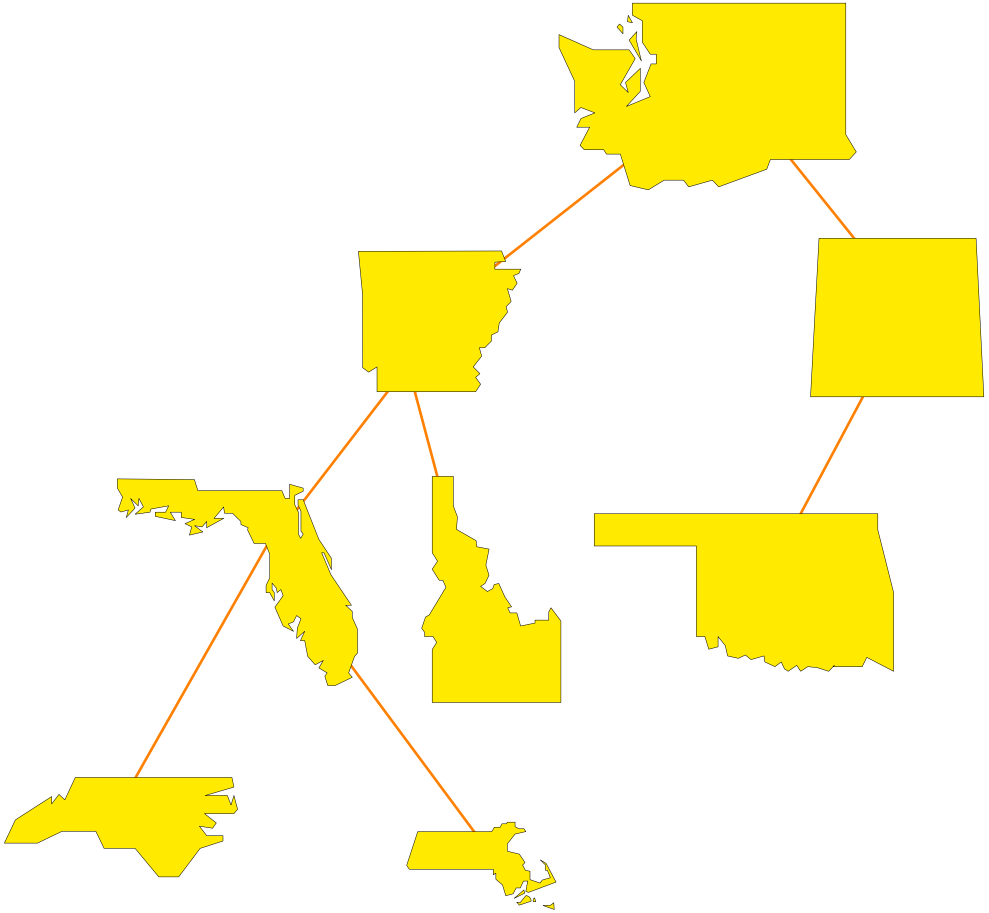
The next nodes must always fill the next level from <u>left to right</u>.



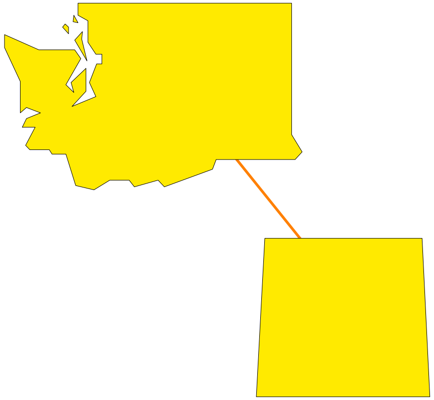
# Is This Complete?



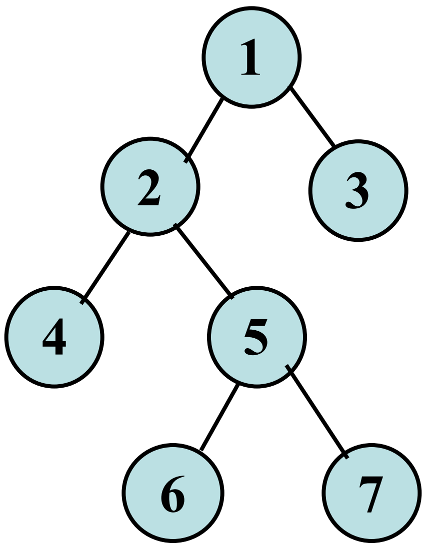
# Is This Complete?

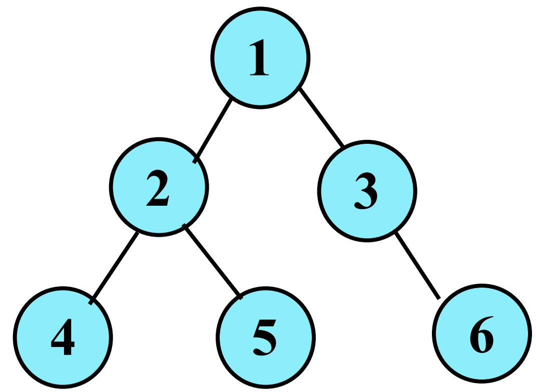


# Is This Complete?



# Is this a complete binary tree?



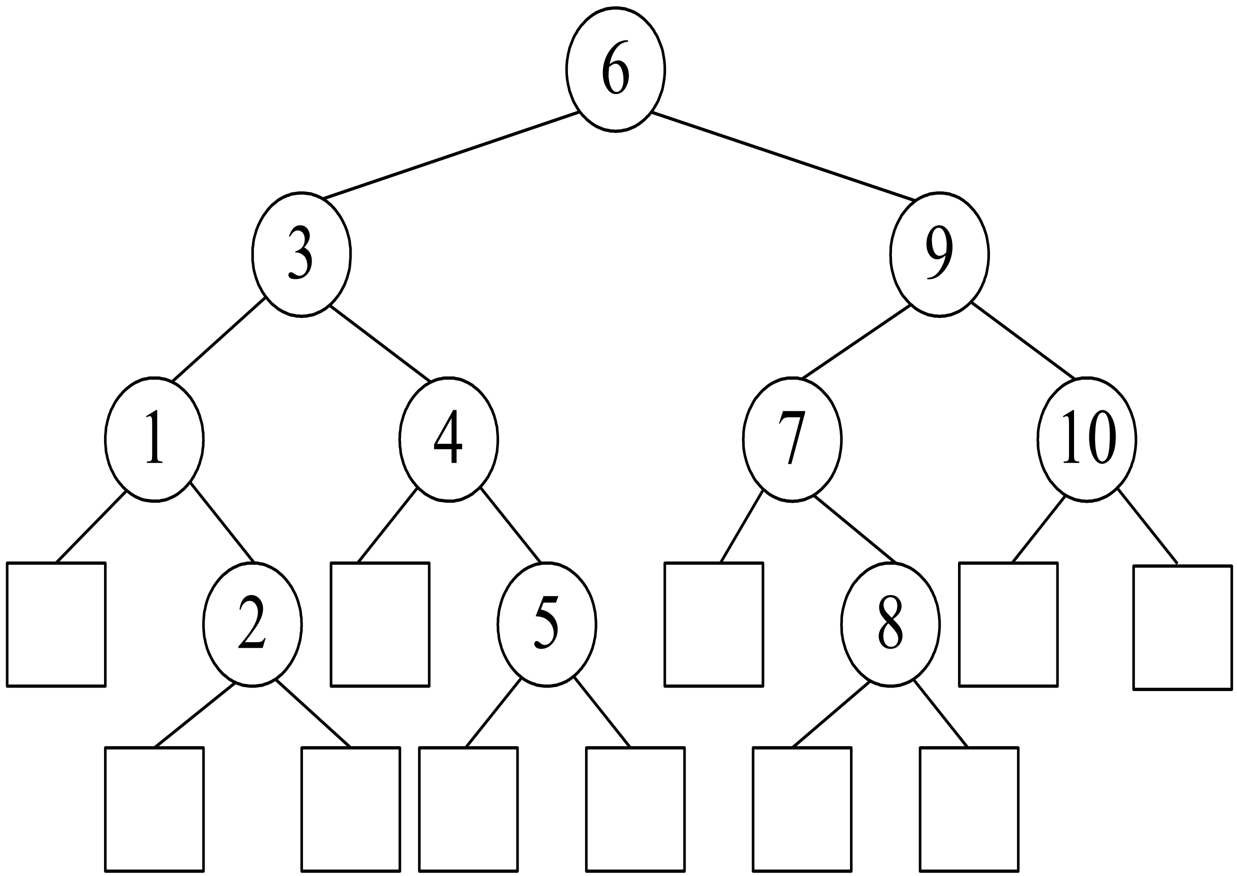




**Q：What is the relationship between full binary tree and complete binary tree?**

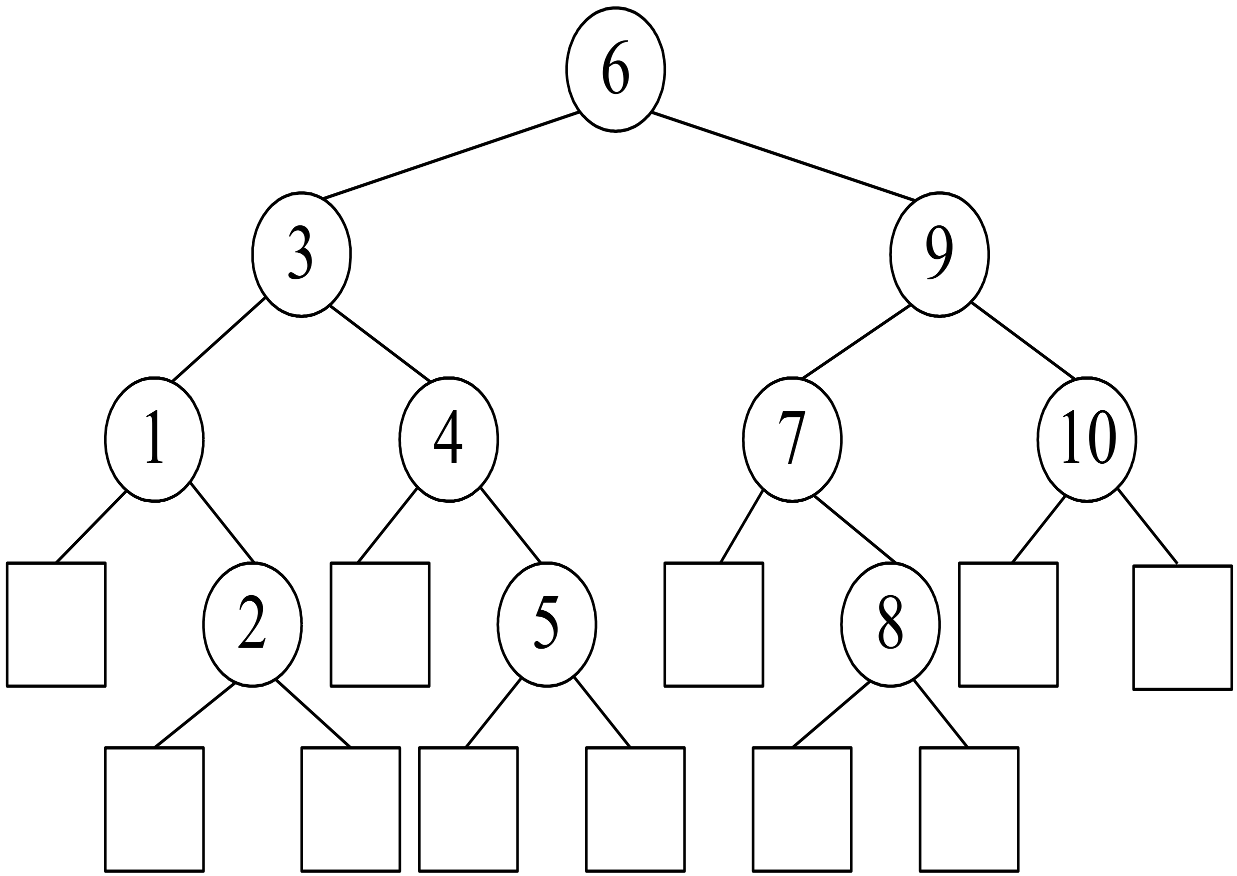
# Extended binary tree

- If the binary tree has empty subtree, add new special nodes (empty leaves)
  - For the nodes with degree 1, add one empty leaf in its subtree;
  - For the leaves, add two empty leaves for each original leaf.

# Extended binary tree



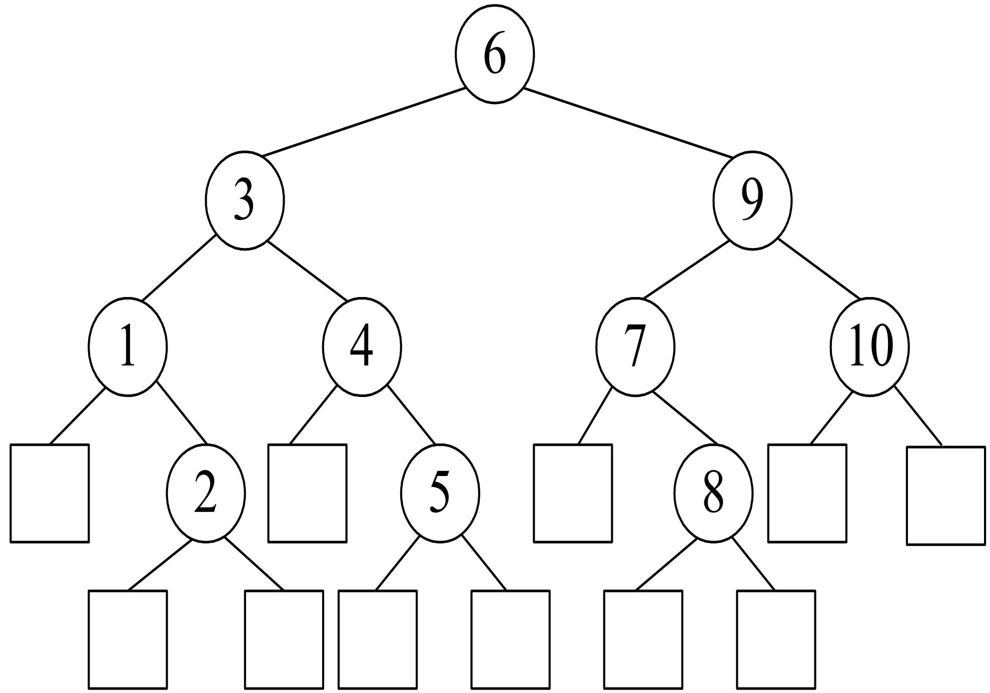
# Extended binary tree



# Extended binary tree

- **Length of External path E** The sum of the path length from the root to every external node.
- **Length of Internal path I** The sum of the path length from the root to every internal node.
- **E=I+2n  n is the number of internal nodes**

# Extended binary tree




- 10 internal nodes
- E = 3 + 4 + 4 + 3 + 4 + 4 + 3 + 4 + 4 + 3 + 3= 39
- I = 0 + 1 + 2 + 3 + 2 + 3 + 1 + 2 + 3 + 2 = 19
- E = I + 2n=19+2*10=39

# Binary Tree Theorem

Six important properties for binary trees

# Binary Tree Theorem

**Theorem: The number of leaves in a non-empty binary tree is one more than the number of internal nodes with degree 2.**

$$n_0=n_2+1$$

# Binary Tree Theorem (Con't)

- **The number of the leaves in a non-empty full binary tree is one more than the number of internal nodes.**

# Binary Tree Theorem (Con't)

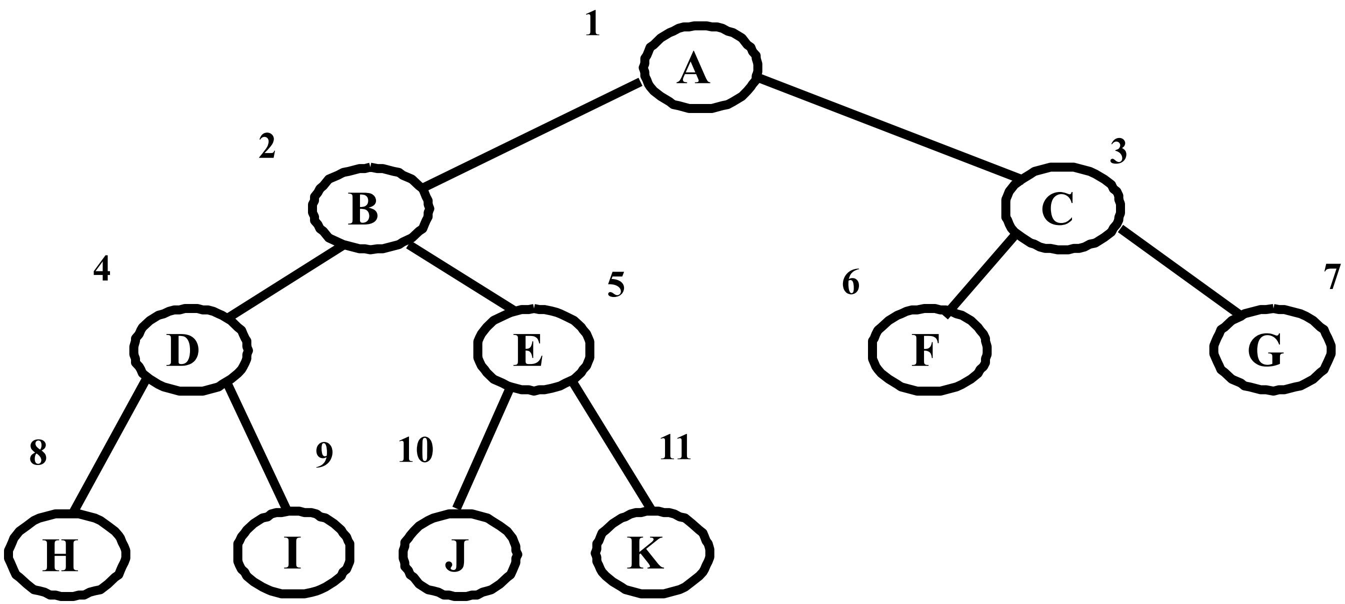
- **There are $2^i$ nodes($i \geq 0$) at most on the level i.**

- **The binary tree with height h(depth is h-1) has a maximum of $2^h-1$ nodes.**

- **The height of a complete binary tree with n nodes (n>0) is $\lceil \log_2(n+1) \rceil$.**

# Binary Tree Theorem (Con't)

- **In the complete binary tree, if the index of a node is i (1≤i≤n, n≥1,n is the total number of the nodes):**




**Complete binary tree**

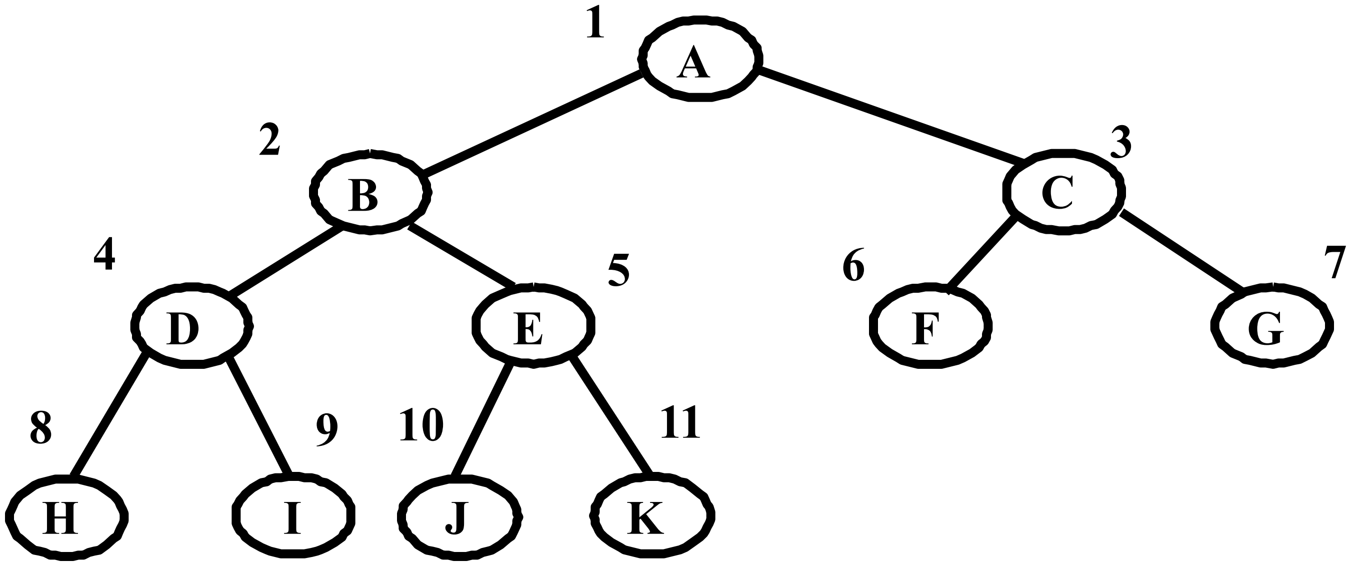
# Binary Tree Theorem (Con't)

（1）If i=1,the node i is root, it has no parents.

（2）If i>1,its parent index is $\lfloor i/2 \rfloor$.

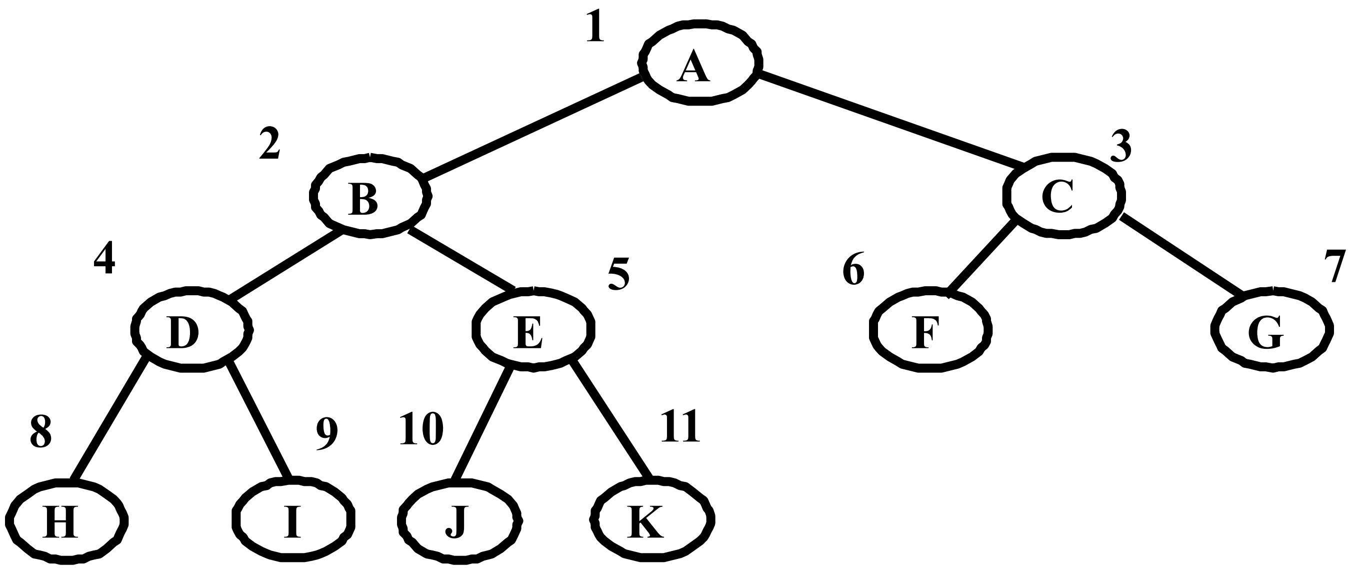
If i is even number, its parent index is i/2, it is left child node.

If i is odd number, its parent index is （i-1）/2, it is right child.



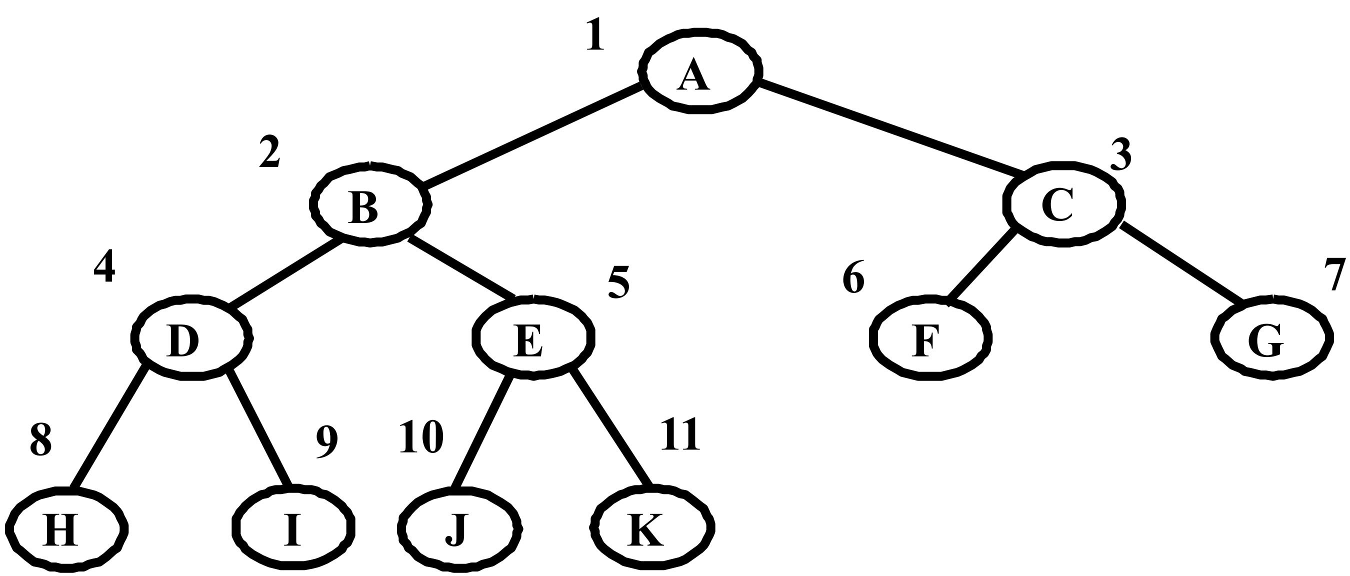
# Binary Tree Theorem (Con't)

**(3) If node i has left child, the index number of its left child is 2i; If node i has right child, the index number of its right child is (2i+1).**



# Binary Tree Theorem (Con't)

**(4) If n is odd number, each internal node has both left child and right child; If n is even number, the internal node with biggest index number only has left child, others have both left and right children.**



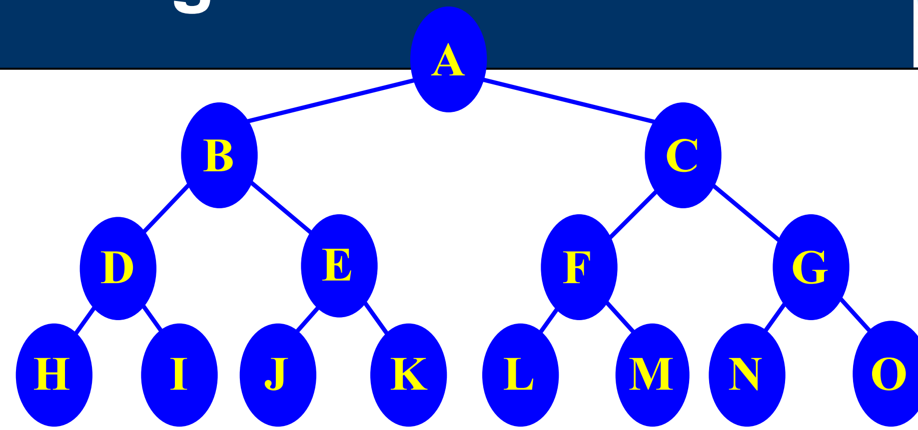
# The storage structure of binary tree

- Sequential Storage Structure

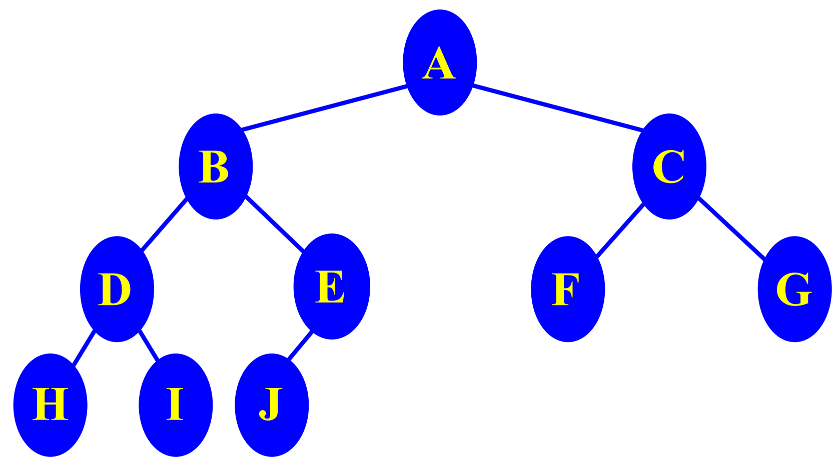
- Linked Storage structure

# Sequential Storage Structure

**Complete binary tree**

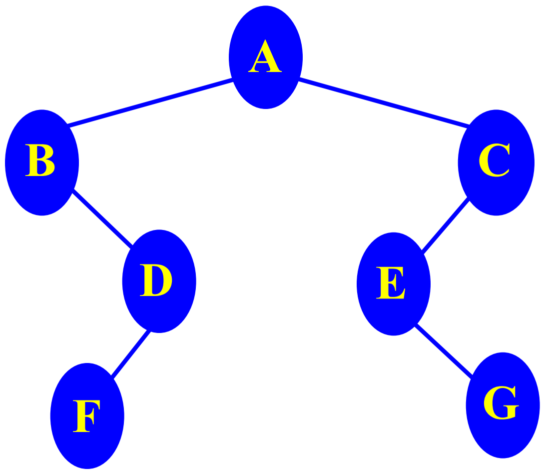



| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |

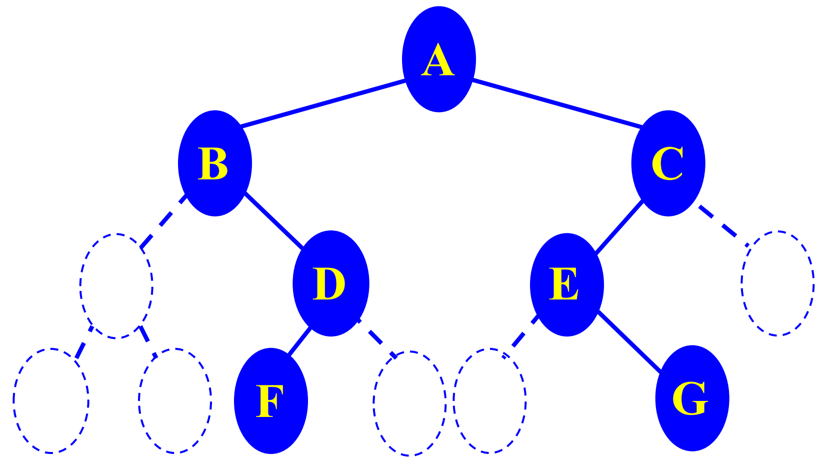



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G | H | I | J |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

# Arbitrary binary tree: add empty nodes




(a)Normal binary tree

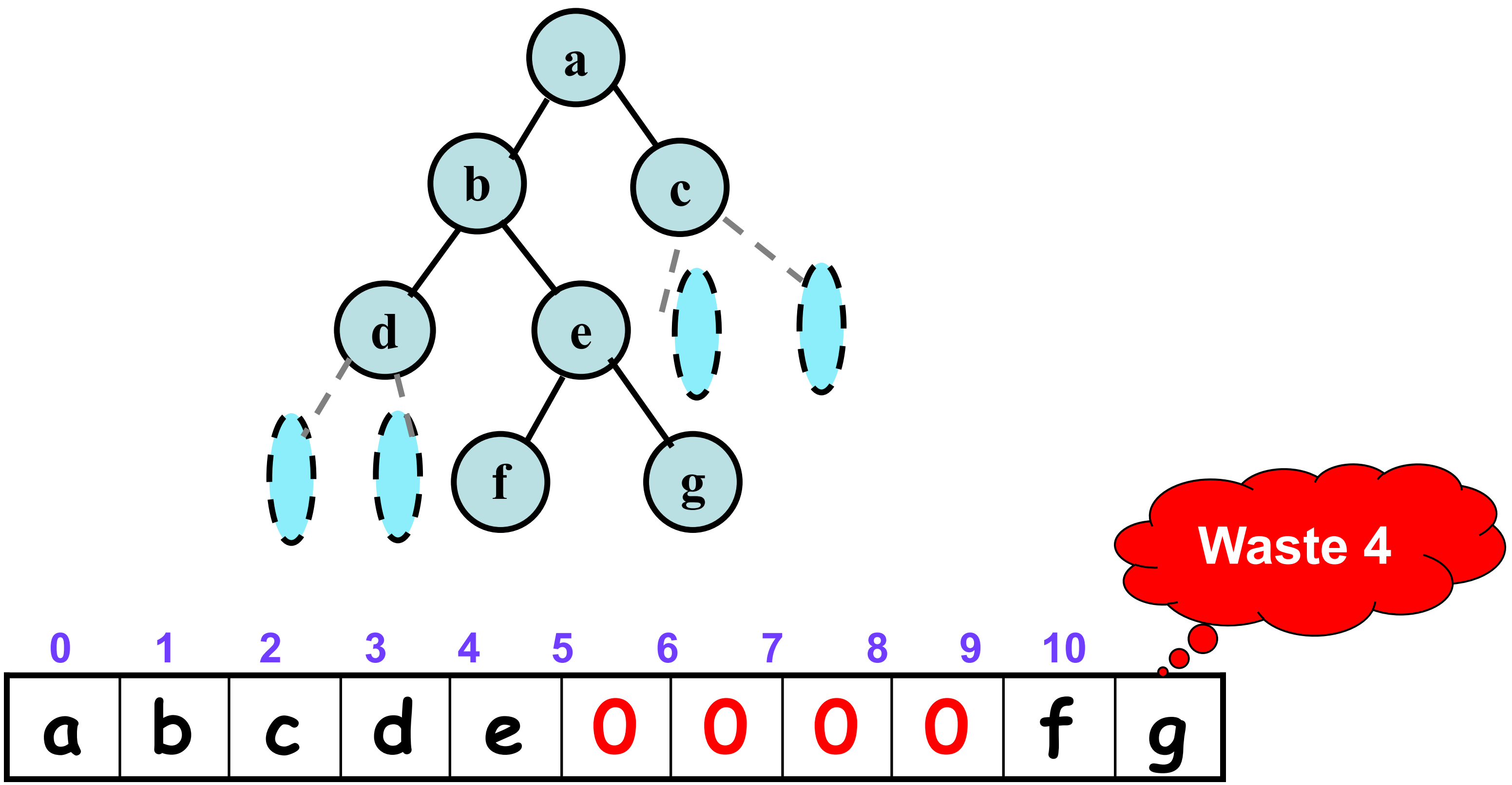



(b)Complete binary tree

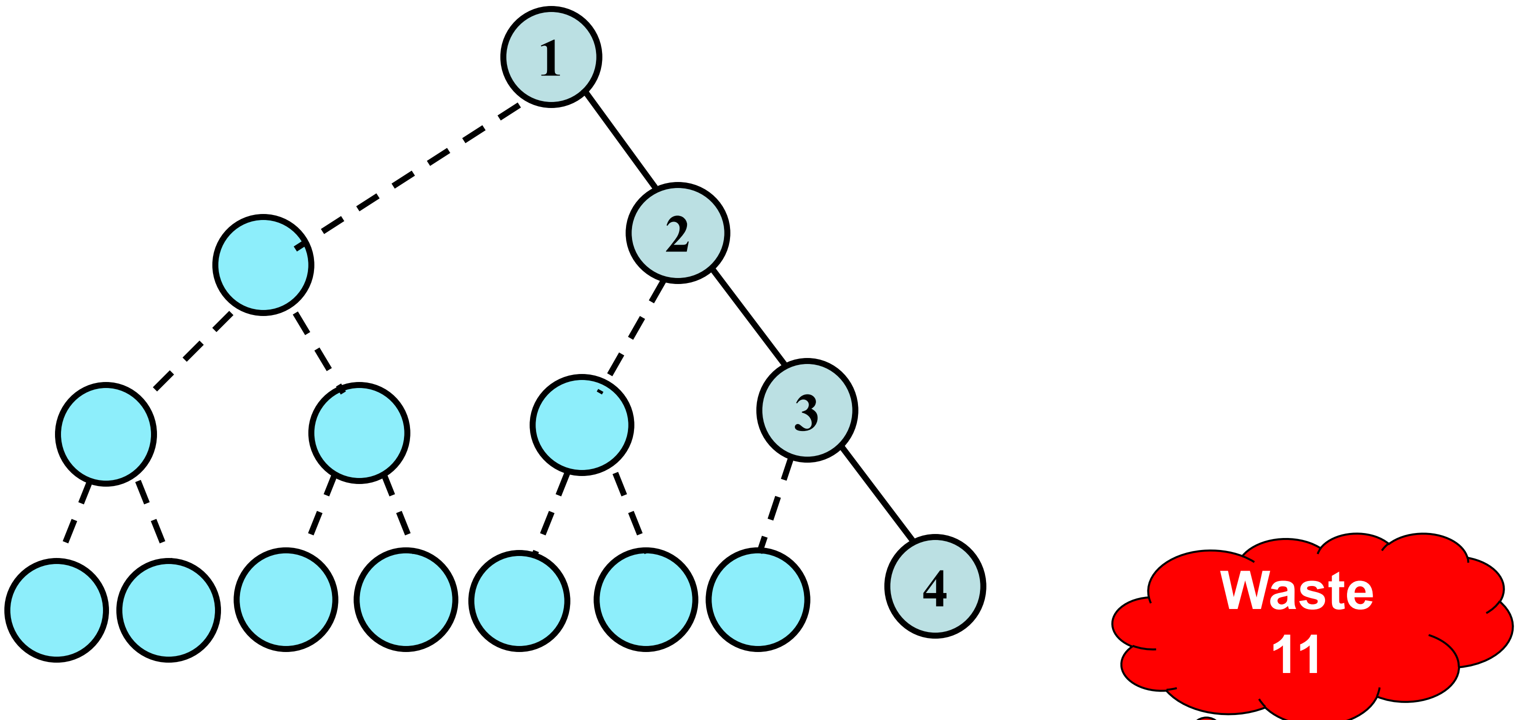
| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| array | A | B | C | ∧ | D | E | ∧ | ∧ | ∧ | F | ∧ | ∧ | G |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |

(c) array-based storage

# Height 4，  7 nodes



# Height 4， only 3 right children




| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 |

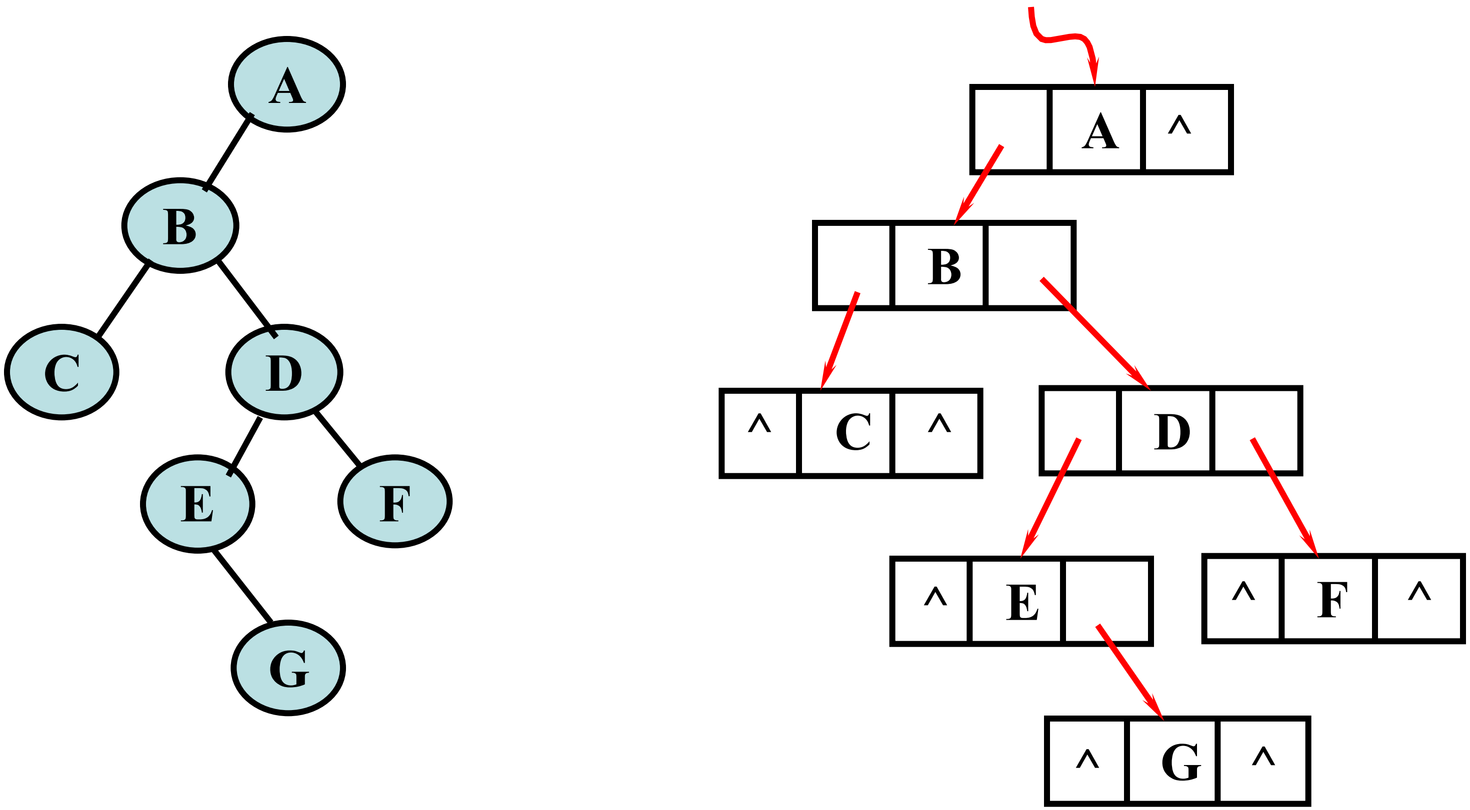
- Sequential Storage Structure is efficient for full binary trees and complete binary trees.

# Linked structure storage

- Three fields： data, left pointer and right pointer

| | | |
|---|---|---|
| leftChild | data | rightChild |

# Linked structure storage



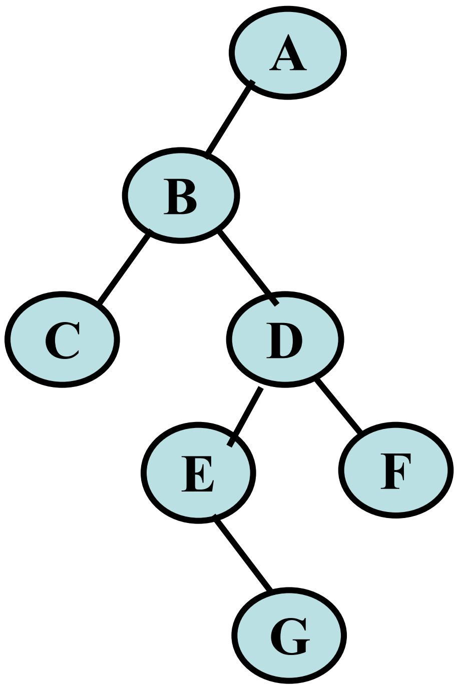

For n nodes, there are n+1 empty pointer field.

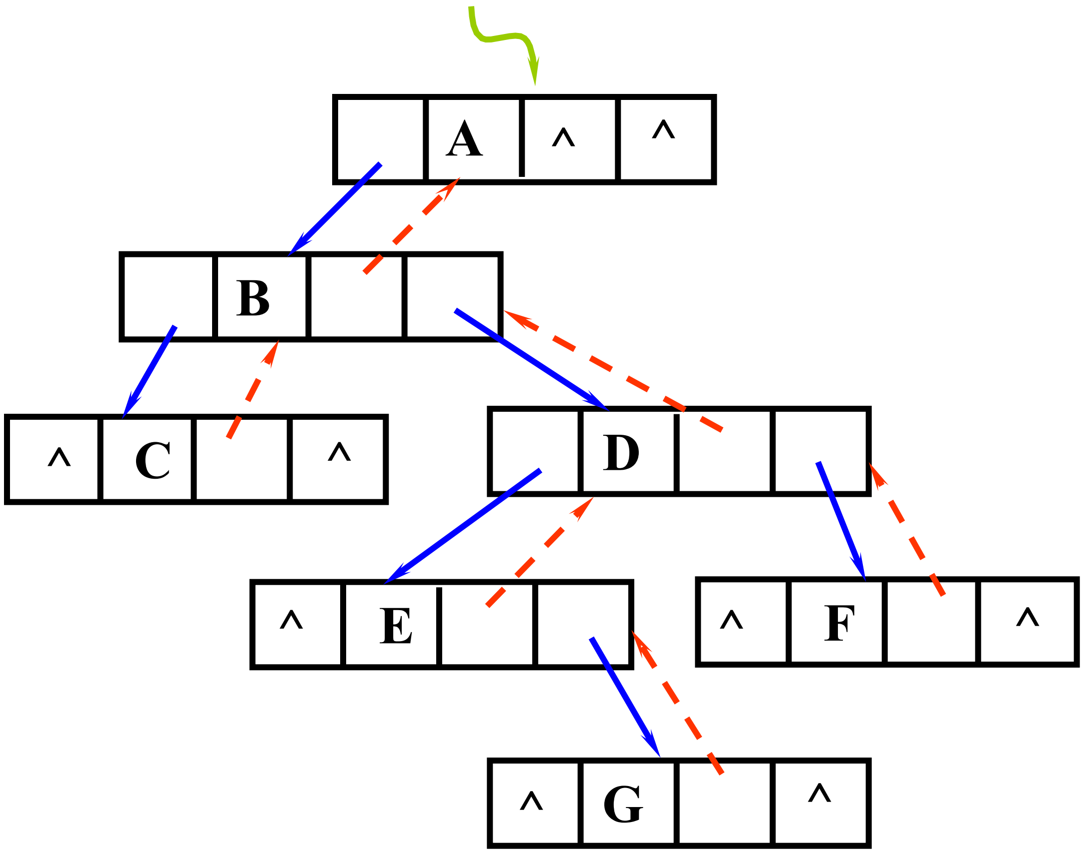
# Linked structure storage

- **Trigeminal linked list**:
- Add parent pointer for searching the parent node.

| leftChild | data | parent | rightChild |
|---|---|---|---|

# Trigeminal linked list for binary tree



# Homework

- Please refer to Icourse，Huawei Cloud.

- Due date for quiz: 23:30  2022/4/12
- Due date for homework: 23:30 2022/4/17
- Due data for online lab assignment: 2022/4/17  23：30
- Due data for offline lab assignment: 2022/4/24  18:00