

VE281

Data Structures and Algorithms

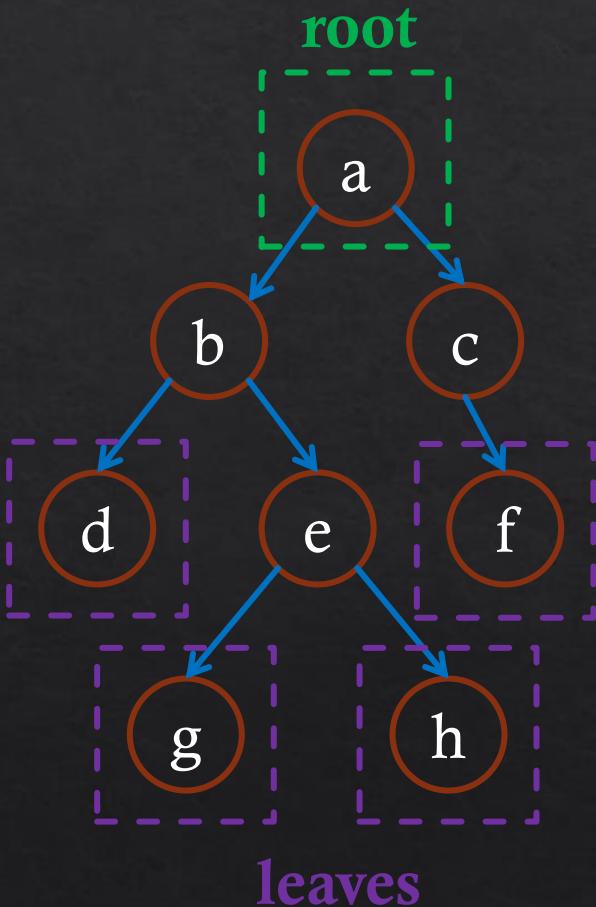
Trees

Learning Objectives:

- Know some basic terminology of trees and binary trees
- Know some basic properties of binary trees
- Know how to represent a binary tree by an array and a linked list

Outline

- ❖ Trees
- ❖ Binary Trees

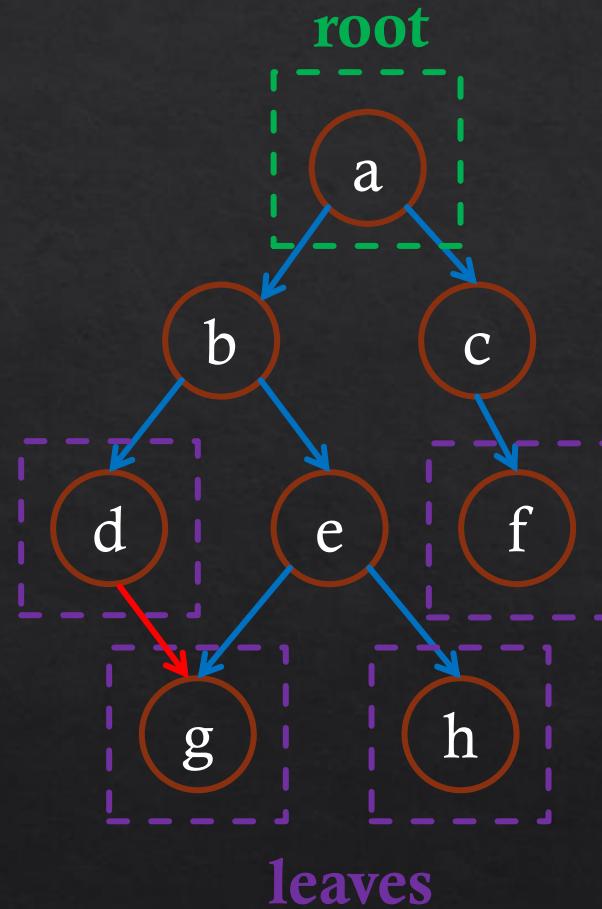


Trees

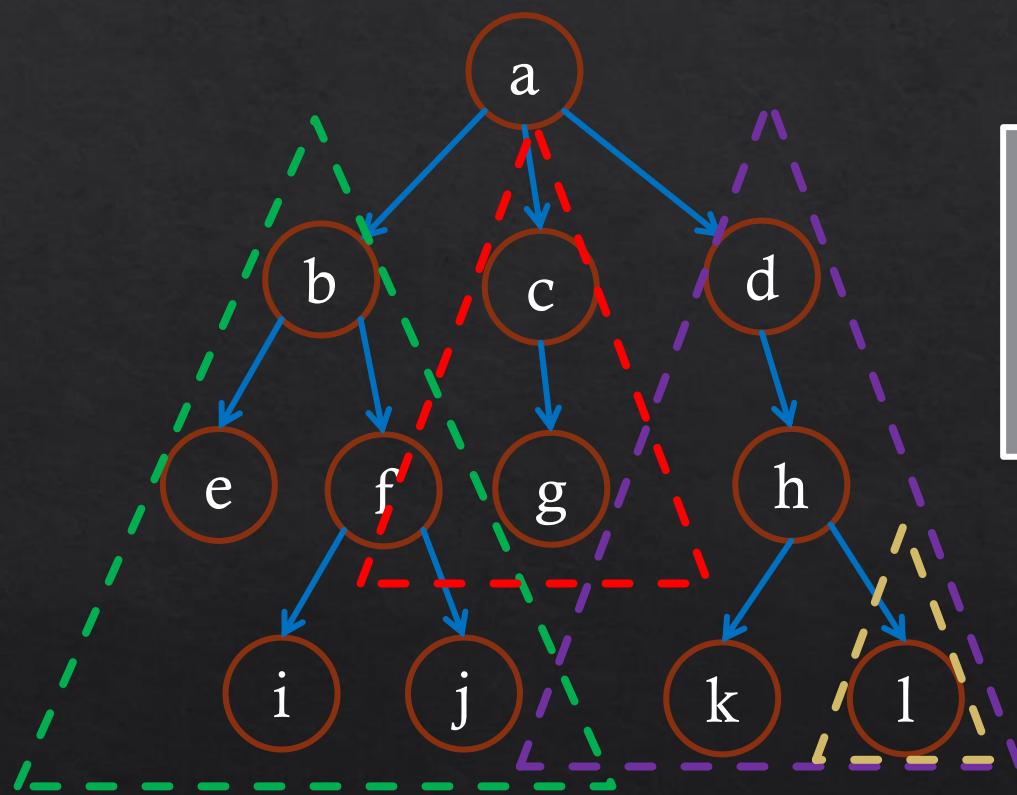
- ❖ Tree is an extension of linked list data structure:
 - ❖ Each node connects to **multiple** nodes.
- ❖ A tree is a “natural” way to represent hierarchical structure and organization.
- ❖ Many problems in computer science can be solved by breaking it down into smaller pieces and arranging the pieces in some form of hierarchical structure.
 - ❖ For example: merge sort.

Tree Terminology

- ❖ Just like lists, trees are collections of nodes.
- ❖ The node at the top of the hierarchy is the **root**.
- ❖ Nodes are connected by **edges**.
- ❖ Edges define **parent-child** relationship.
 - ❖ Root has no parent.
 - ❖ All other nodes have **exactly one** parent.
- ❖ A node with no children is called a **leaf**.



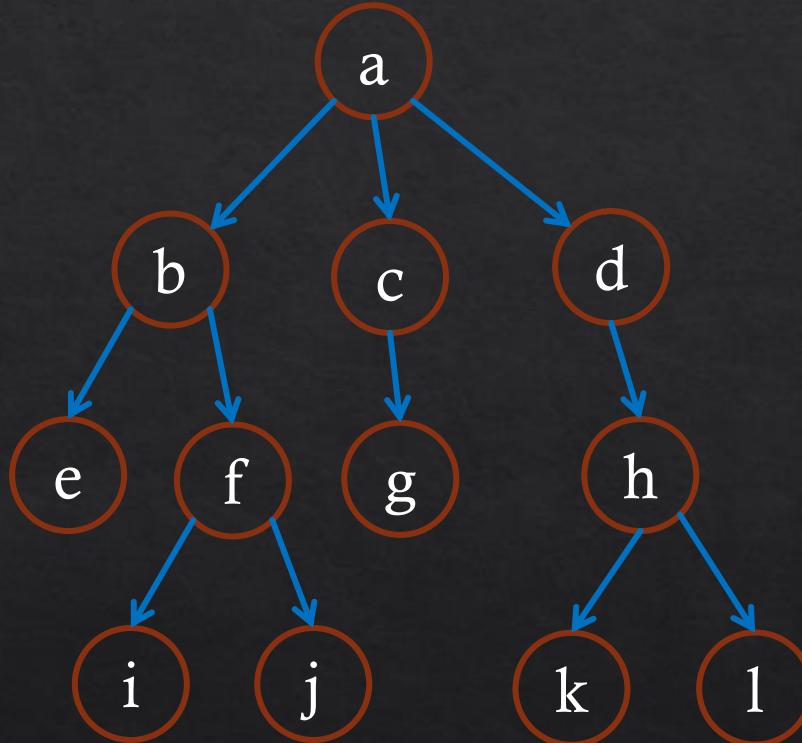
Subtrees



Subtree can be defined for any node in general, not just for the root node.

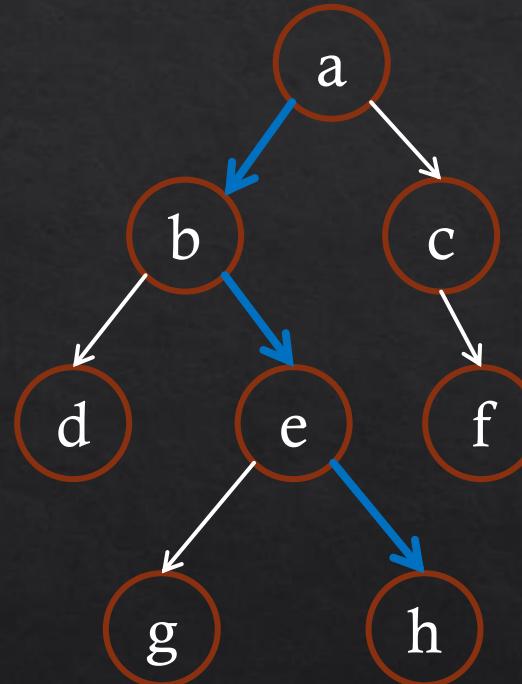
More Tree Terminology

- ◊ f is the **child** of b.
- ◊ b is the **parent** of f.
- ◊ Nodes that share the same parent are **siblings**.
 - ◊ b and c are the **siblings** of d.
 - ◊ e is the **sibling** of f.



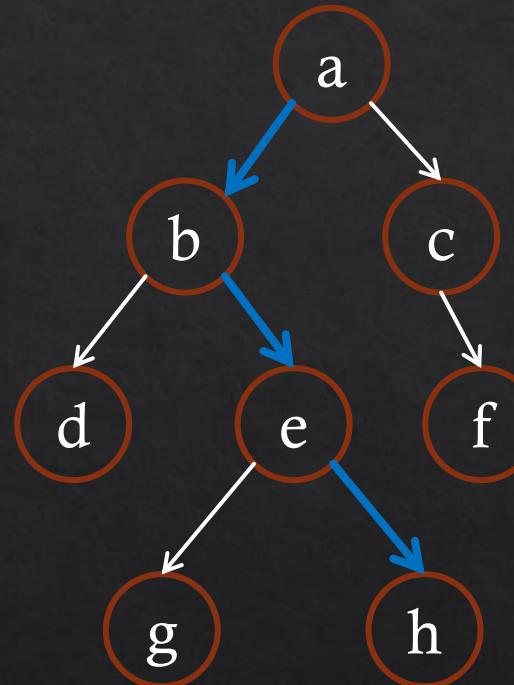
Path

- ◊ A **path** is a sequence of nodes such that the next node in the sequence is a child of the previous.
 - ◊ E.g., $a \rightarrow b \rightarrow e \rightarrow h$ is a path.
 - ◊ The path length is 3.
- ◊ Path length may be 0, e.g., b going to itself is a path and its length is 0.
- ◊ **Claim:** If there exists a path between two nodes, then this path is the **unique** path between these two nodes.



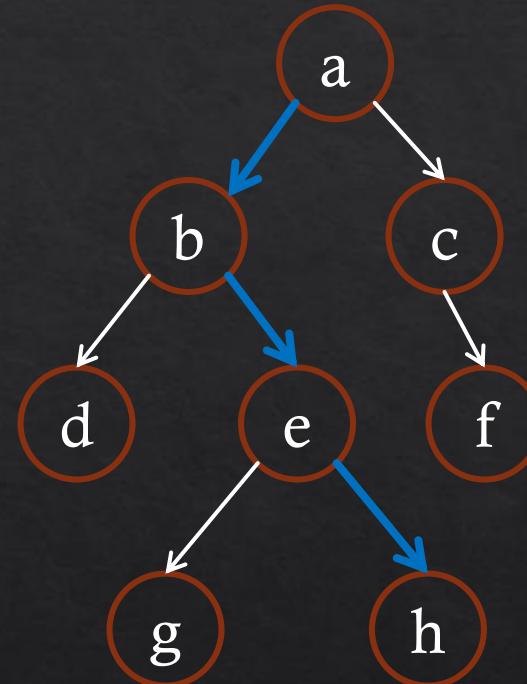
Ancestors and Descendants

- ◊ If there exists a path from a node A to a node B, then A is an **ancestor** of B and B is a **descendant** of A.
 - ◊ E.g., a is an ancestor of h and h is a descendant of a.



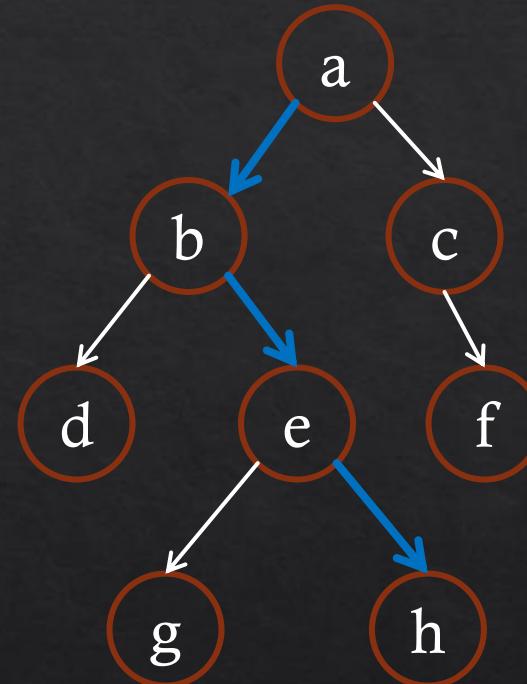
Depth, Level, and Height of a Node

- ◊ The **depth or level of a node** is the length of the unique path from the root to the node.
 - ◊ E.g., $\text{depth}(b)=1$, $\text{depth}(a)=0$.
- ◊ The **height of a node** is the length of the longest path from the node to a leaf.
 - ◊ E.g., $\text{height}(b)=2$, $\text{height}(a)=3$.
 - ◊ All leaves have height zero.



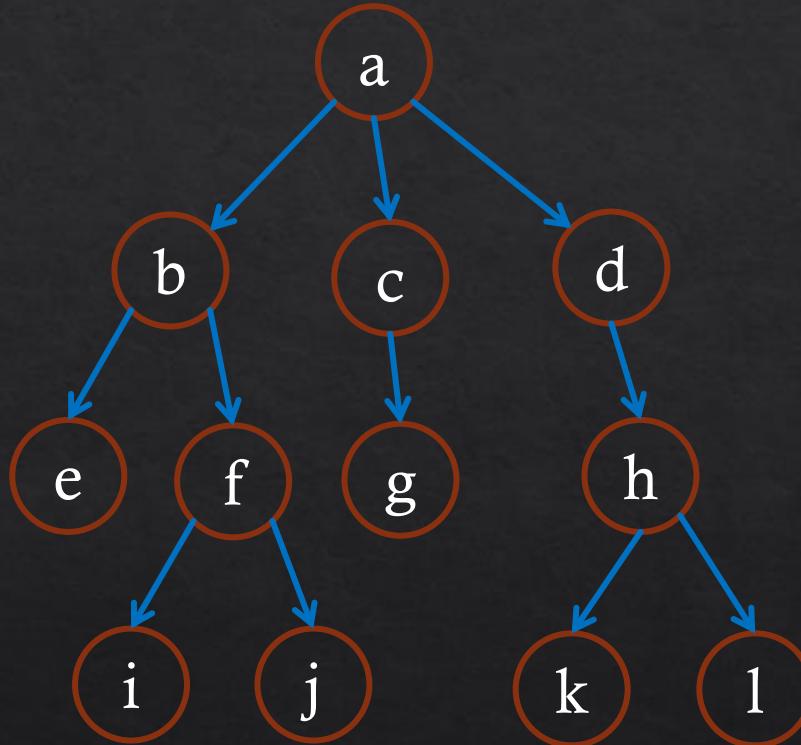
Depth, Level, and Height of a Tree

- ◊ The **height of a tree** is the height of its root.
 - ◊ This is also known as the **depth of a tree**.
 - ◊ The depth of the tree on the right is 3.
- ◊ The **number of levels of a tree** is the height of the tree **plus one**.
 - ◊ The number of levels of the tree on the right is 4.



Degree

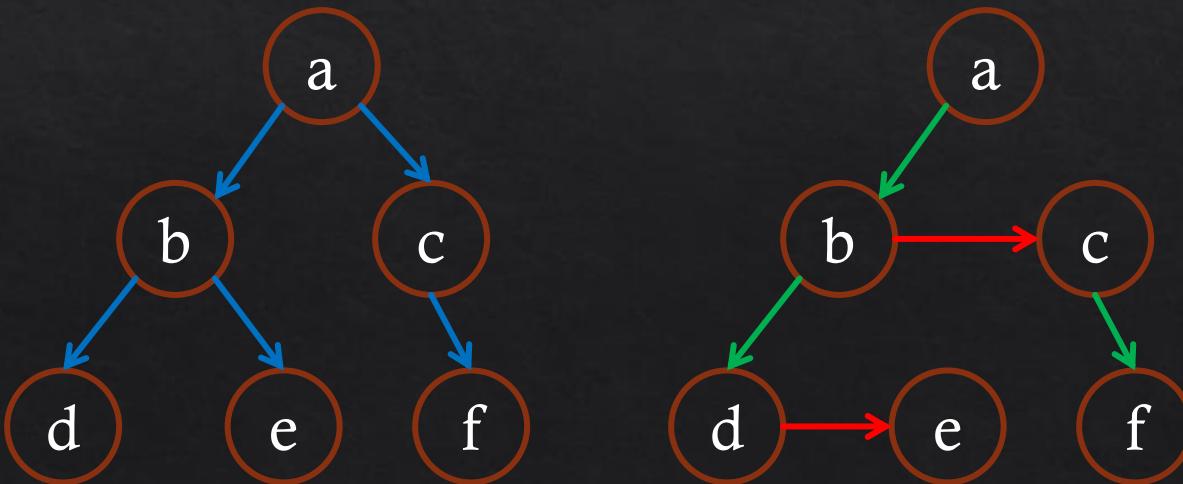
- ◊ The **degree of a node** is the number of children of a node.
 - ◊ E.g., $\text{degree}(a) = 3$,
 $\text{degree}(c) = 1$.
- ◊ The **degree of a tree** is the maximum degree of a node in the tree.
 - ◊ The degree of the tree on the right is 3.



A Simple Implementation of Tree

- ❖ Each node is part of a **linked list** of siblings.
- ❖ Additionally, each node stores a pointer to its **first child**.

```
struct node {  
    Item item;  
    node *firstChild;  
    node *nextSibling;  
};
```

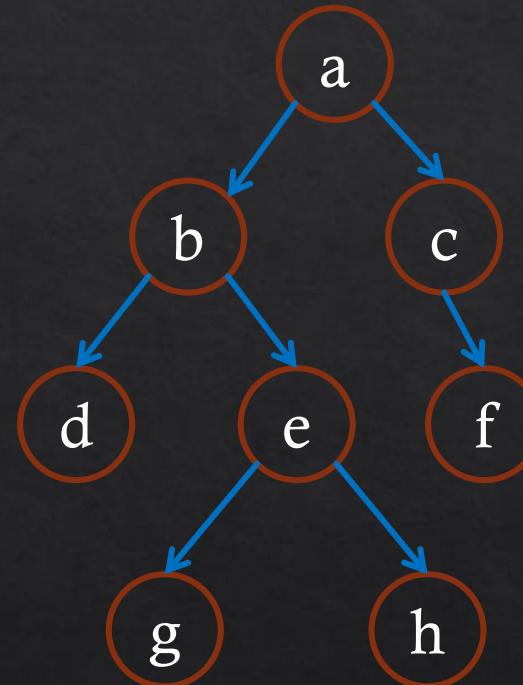


Outline

- ❖ Trees
- ❖ Binary Trees

Binary Tree

- ◊ Every node can only have **at most two** children.
- ◊ An empty tree is a special binary tree.

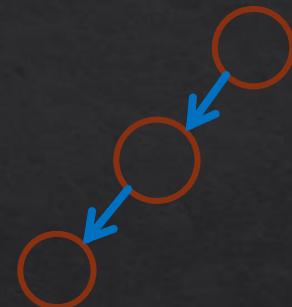


Binary Tree Properties

◊ What is the **minimum** number of nodes in a binary tree of height h (i.e., has $h + 1$ levels)?

◊ Answer: **At least** one node at each level.

◊ $h + 1$ levels means at least $h + 1$ nodes.

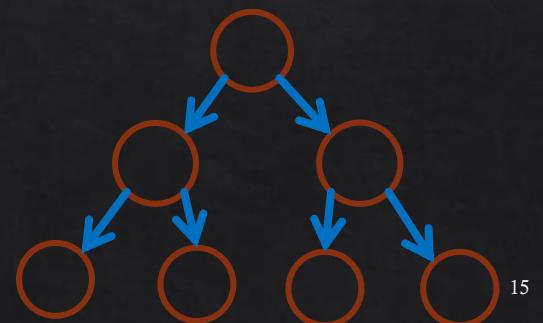


◊ What is the **maximum** number of nodes in a binary tree of height h (i.e., has $h + 1$ levels)?

◊ Answer: At most 2^k nodes at level k .

◊ Maximum number of nodes is

$$1 + 2 + 2^2 + \dots + 2^h = 2^{h+1} - 1$$



Number Of Nodes and Height

- ❖ **Claim** (from the previous slide): Let n be the number of nodes in a binary tree whose height is h (i.e., has $h + 1$ levels).
 - ❖ We have $h + 1 \leq n \leq 2^{h+1} - 1$.
- ❖ **Question:** given n nodes, what is the height h of the tree?
 - ❖ $\log_2(n + 1) - 1 \leq h \leq n - 1$

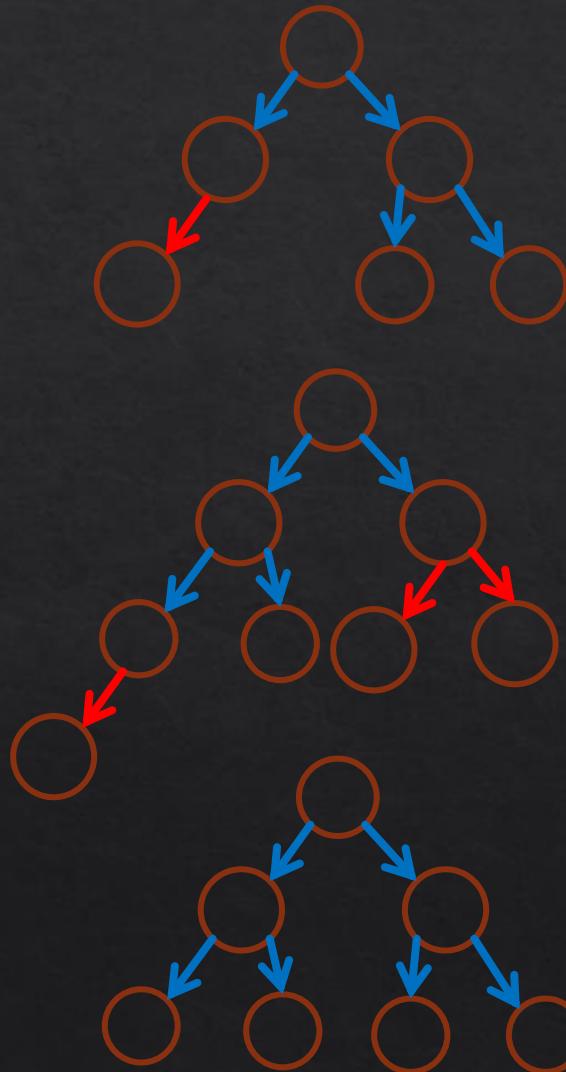
Types of Binary Trees

◊ A binary tree is **proper** if every node has 0 or 2 children.

◊ A binary tree is **complete** if:

1. every level **except** the lowest is fully populated, and
2. the lowest level is populated **from left to right**.

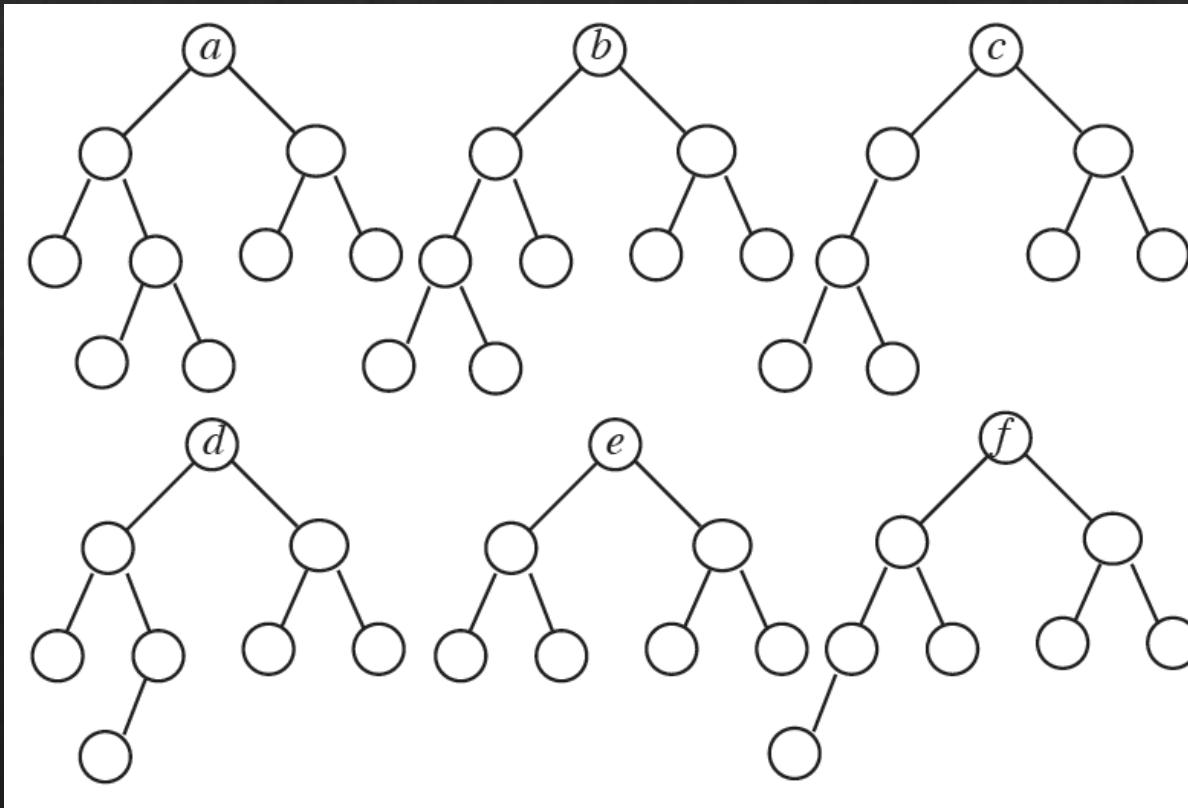
◊ A binary tree is **perfect** if **every level** is fully populated.





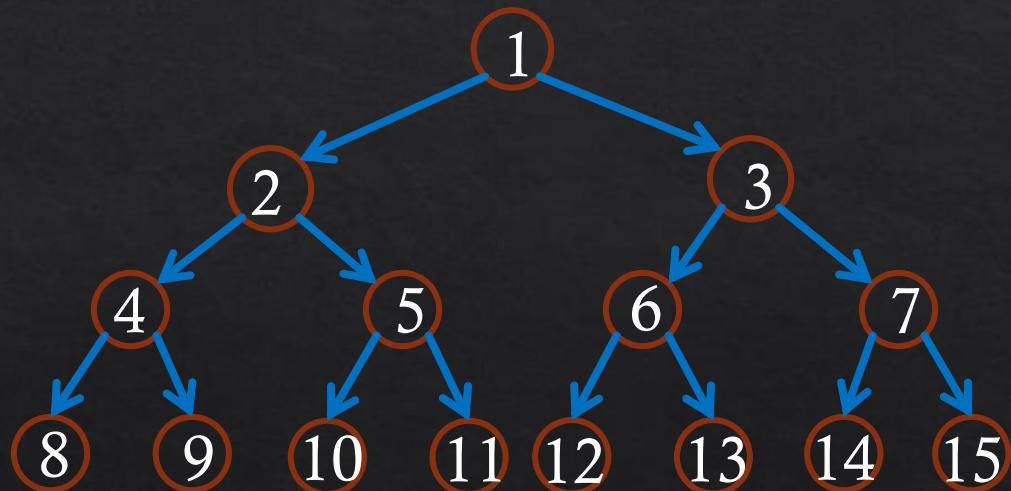
Which Statements Are Correct?

- A. Trees a and d are proper.
- B. Tree c is complete.
- C. Trees b and f are complete.
- D. Tree e is perfect.

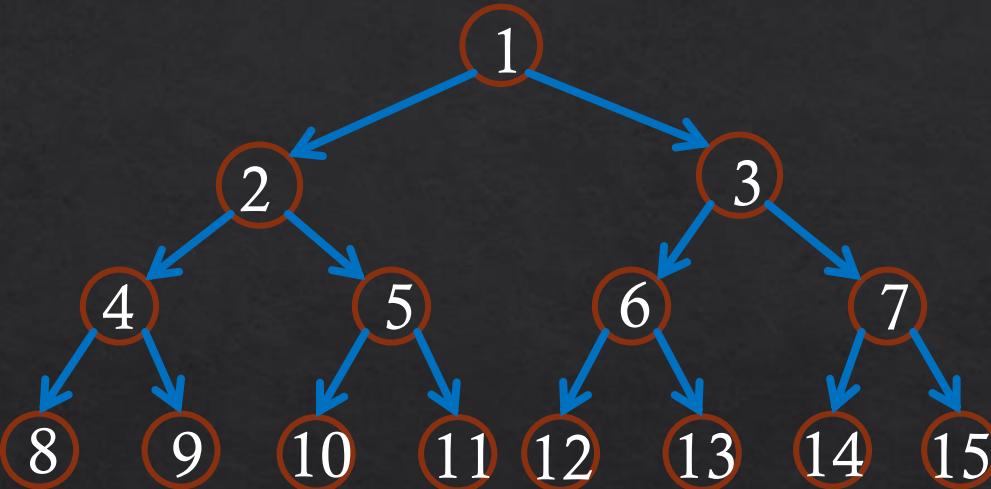


Numbering Nodes In a Perfect Binary Tree

- ❖ Numbering nodes from 1 to $2^{h+1} - 1$.
- ❖ Numbering **from top to bottom** level.
- ❖ Within a level, numbering **from left to right**.



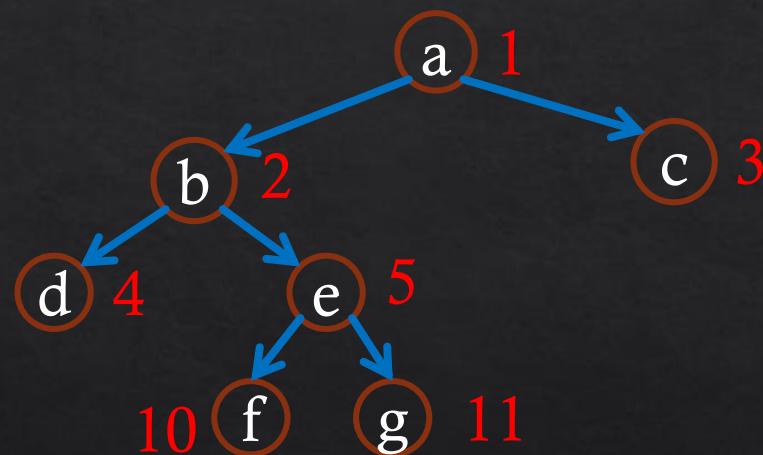
Numbering Nodes In a Perfect Binary Tree



- ❖ What is the parent of node i ?
 - ❖ For $i \neq 1$, it is $\lfloor i/2 \rfloor$. For node 1, it has no parent.
- ❖ What is the left child of node i ? Let n be the number of nodes.
 - ❖ If $2i \leq n$, it is $2i$; If $2i > n$, no left child.
- ❖ What is the right child of node i ?
 - ❖ If $2i + 1 \leq n$, it is $2i + 1$; If $2i + 1 > n$, no right child.

Representing Binary Tree Using Array

- ◊ Based on the numbering scheme for a **perfect** binary tree.
- ◊ If the number of the node **in a perfect binary tree** is i, then the node is put at index i of the array.



a	b	c	d	e	-	-	-	-	f	g
---	---	---	---	---	---	---	---	---	---	---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]

How Would You Represent a **Right-skewed** Binary Tree?

❖ Assume array index starts from 1.

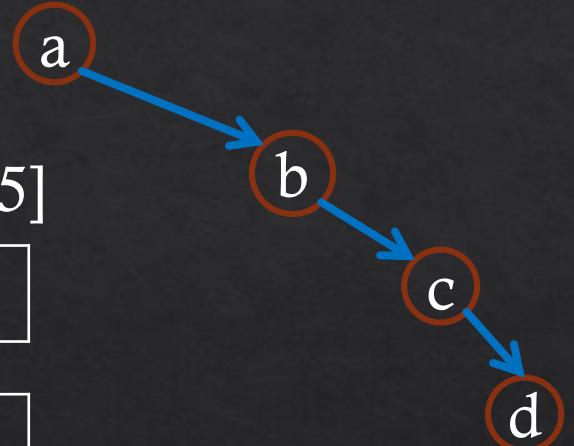
- A.

[1]	[3]	[5]	[7]	[9]	[11]	[13]	[15]
a	b	-	c	-	-	-	d
- B.

[1]	[3]	[5]	[7]	[9]	[11]	[13]	[15]
a	b	c	d	-	-	-	-
- C.

[1]	[3]	[5]	[7]	[9]	[11]	[13]	[15]
a	-	b	-	-	-	-	d
- D.

[1]	[3]	[5]	[7]	[9]	[11]	[13]	[15]
a	-	b	-	c	-	d	-



An n node binary tree needs an array whose length is between n and $2^n - 1$.

Representing Binary Tree Using Linked Structure

```
struct node {  
    Item item;  
    node *left;  
    node *right;  
};
```

- ❖ **left/right** points to a left/right **subtree**.
 - ❖ If the subtree is an empty one, the pointer points to **NULL**.
 - ❖ For a leaf node, both its **left** and **right** pointers are **NULL**.

