VE281

Data Structures and Algorithms

Tree; Binary Tree Traversal

Announcement

- Midterm exam time
    - Oct. 25 (Wed.), in class

Outline

- Trees
- Binary Trees
- Binary Tree Traversal

Trees

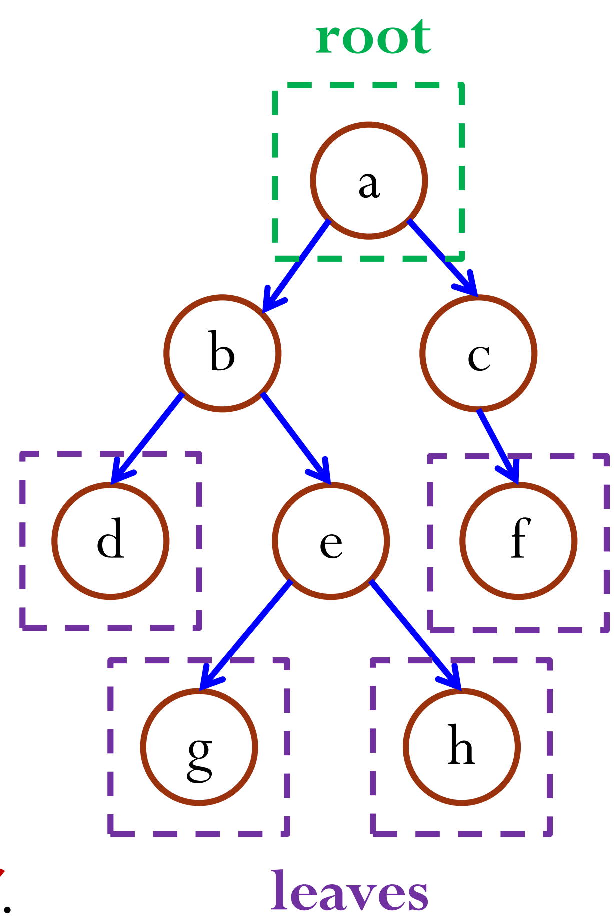
- Tree is an extension of linked list data structure:
  - Each node connects to **multiple** nodes.

- A tree is a "natural" way to represent hierarchical structure and organization.

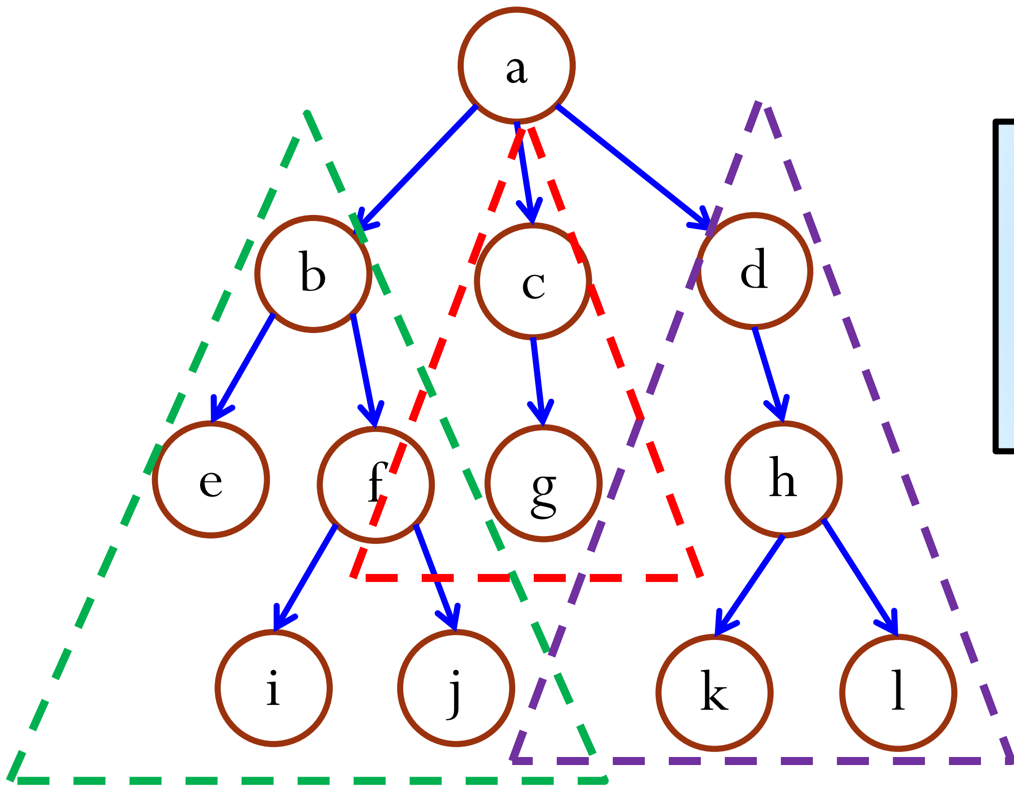
- Many problems in computer science can be solved by breaking it down into smaller pieces and arranging the pieces in some form of hierarchical structure.
  - For example: merge sort.

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 node has **<u>exactly one</u>** parent.
- A node with no children is called a **leaf**.



Subtrees

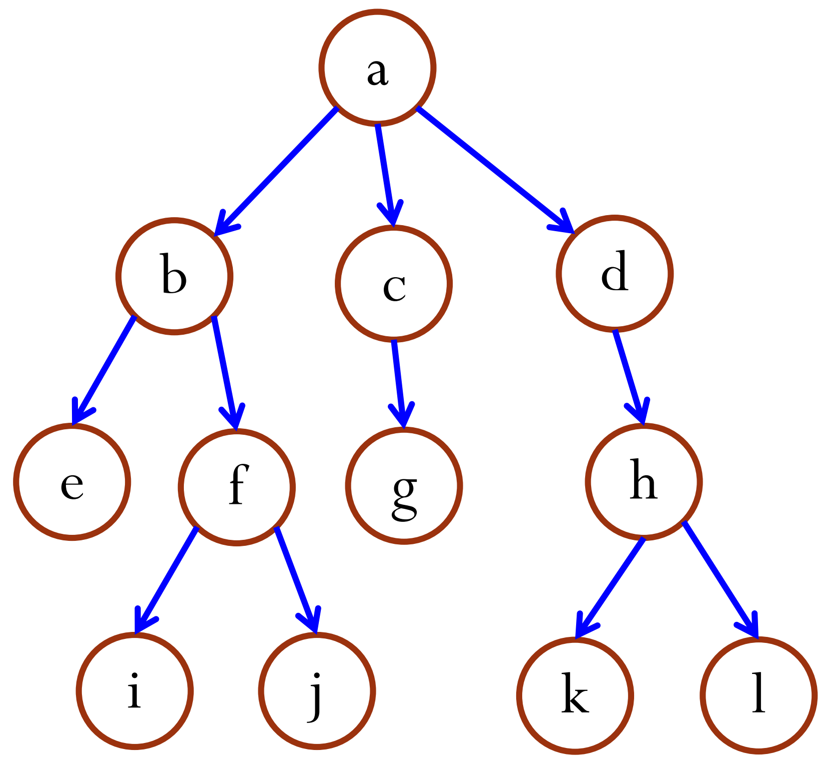


Subtree can be defined for any node in general, not just for the root node.

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.

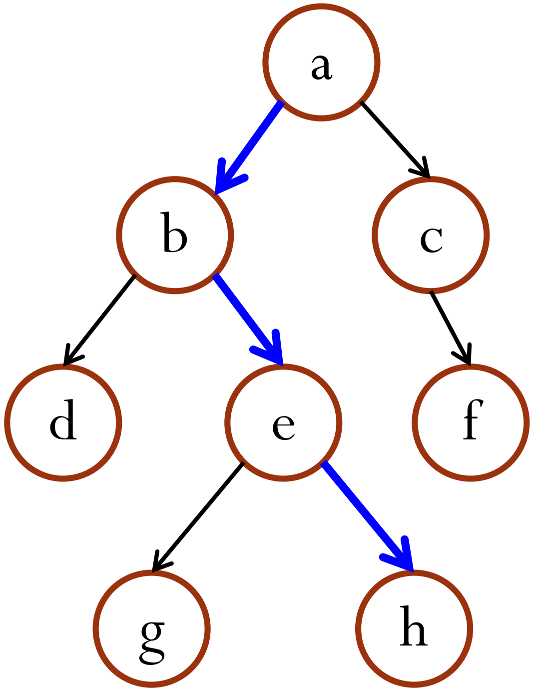


Path

- A **path** is a sequence of nodes such that the next node in the sequence is a child of the previous.
  - E.g., a→b→e→h is a path.
  - The path length is 3.

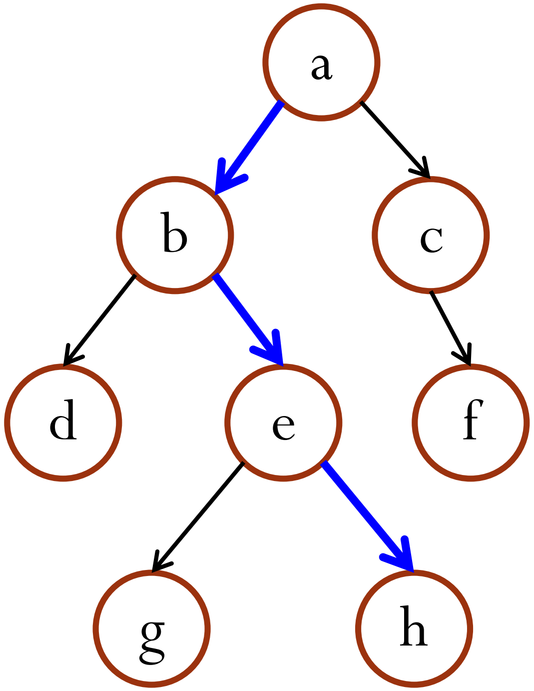
- Path length may be 0, e.g., b going to itself is a path and its length is 0.

- **<u>Claim</u>**: If there exists a path between two nodes, then this path is the **unique** path between these two nodes.



Ancestors and Descendants

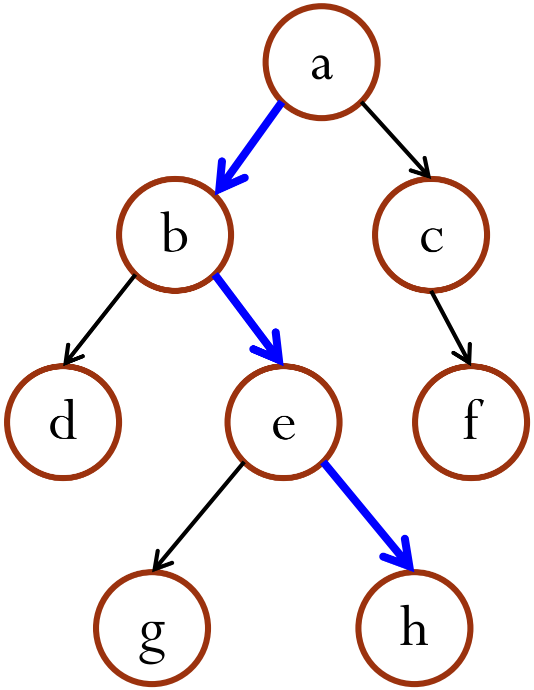
- If there exists a path from a node A to a node B, then A is an **ancestor** of B and B is a **descendant** of A.
  - E.g., a is an ancestor of h and h is a descendant of a.



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., depth(b)=1, depth(a)=0.

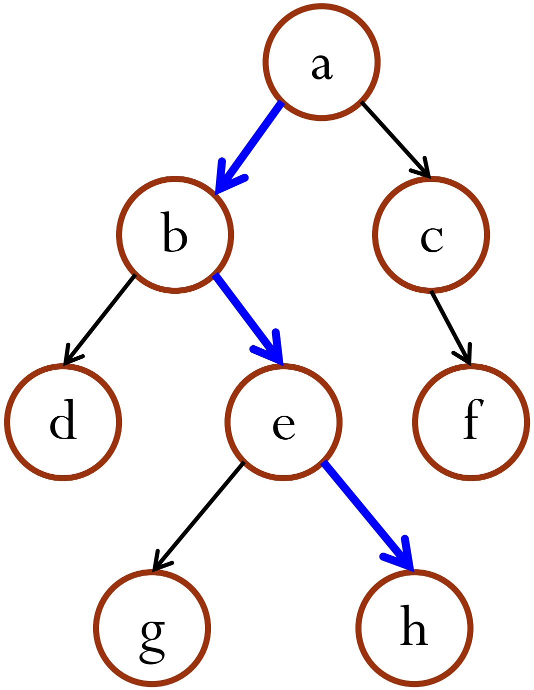
- The **height of a node** is the length of the **longest** path from the node to a **leaf**.
  - E.g., height(b)=2, height(a)=3.
  - All leaves have height zero.



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.

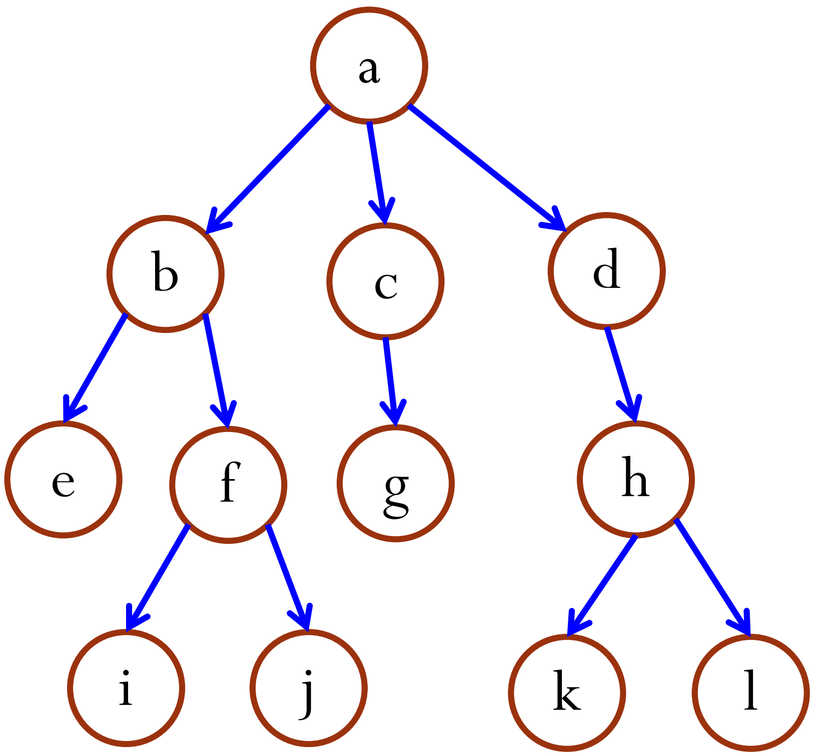
- The **number of levels of a tree** is the height of the tree **plus one**.
  - The number of levels of the tree on the right is 4.



Degree

- The **degree of a node** is the number of children of a node.
  - E.g., degree(a) = 3, degree(c) = 1.

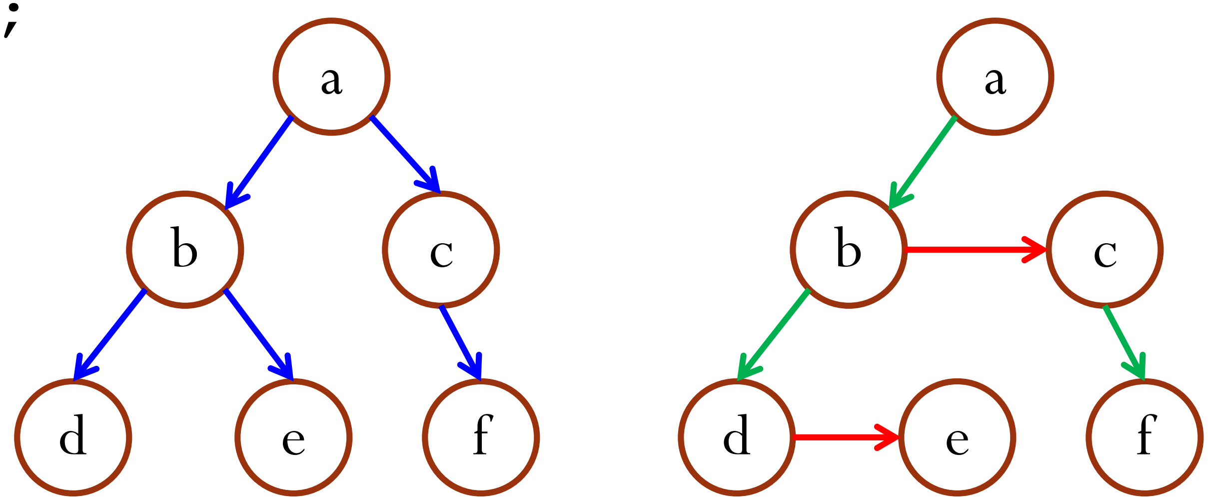
- The **degree of a tree** is the **maximum** degree of a node in the tree.
  - The degree of the tree on the right is 3.



A Simple Implementation of Tree

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

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



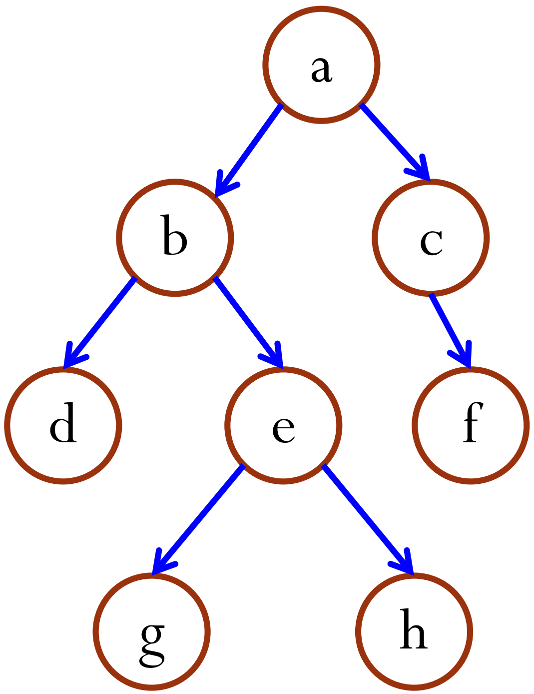
Outline

- Trees
- Binary Trees
- Binary Tree Traversal

Binary Tree

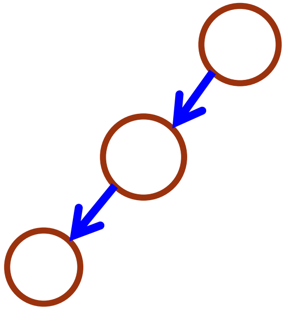
- Every node can only have **at most two** children.

- An empty tree is a special binary tree.



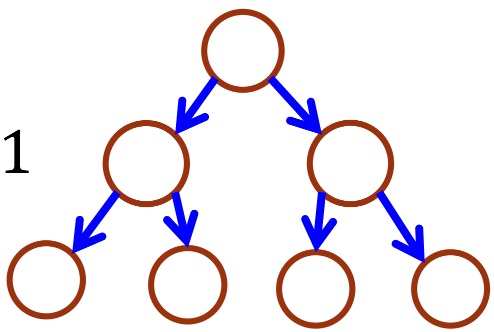
Binary Tree Properties

- 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

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



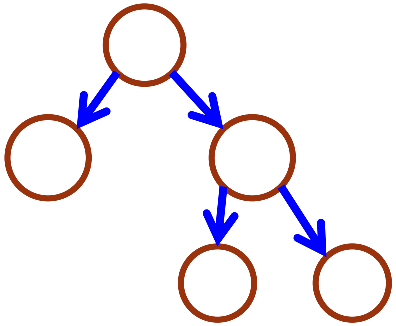
Number Of Nodes and Height

- **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?
  - $\log_2(n + 1) - 1 \leq h \leq n - 1$

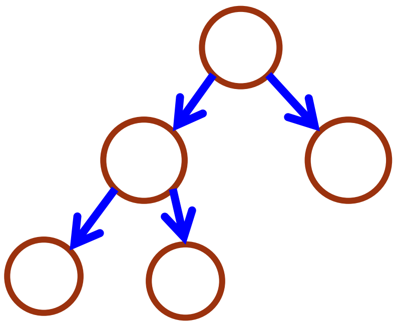
Types of Binary Trees

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

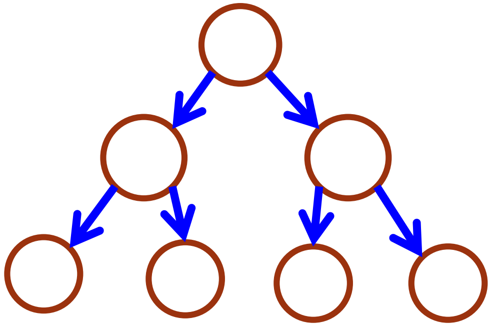


- A binary tree is **complete** if:

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

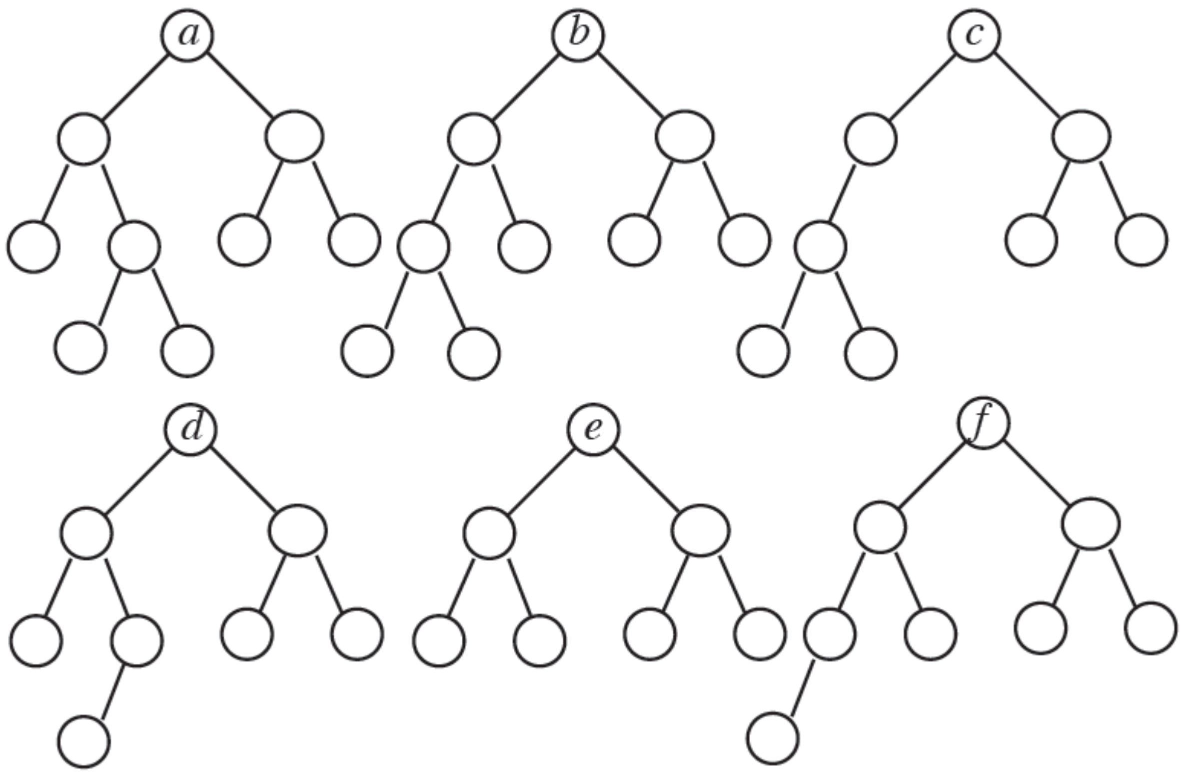


- A binary tree is **perfect** if **every level** is fully populated.



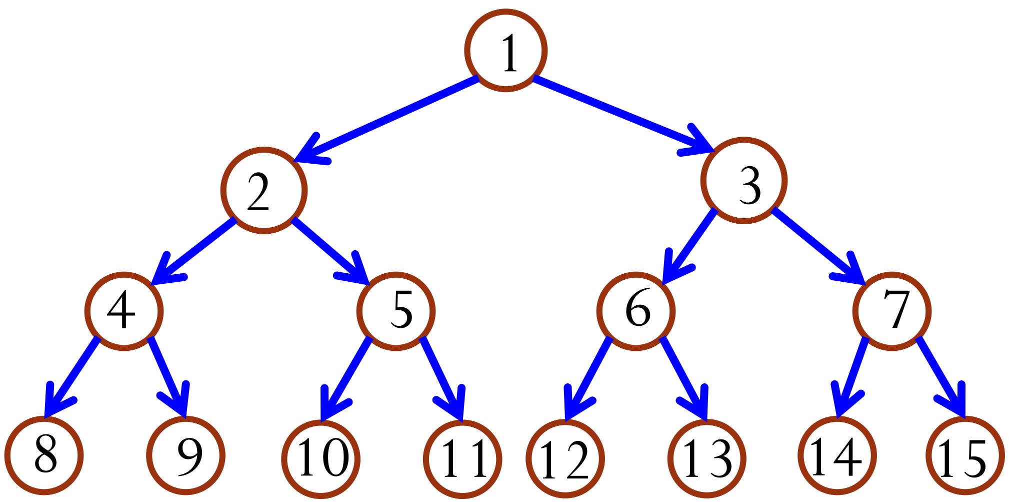
Exercises

- Identify any **proper**, **complete**, and **perfect** binary trees below:

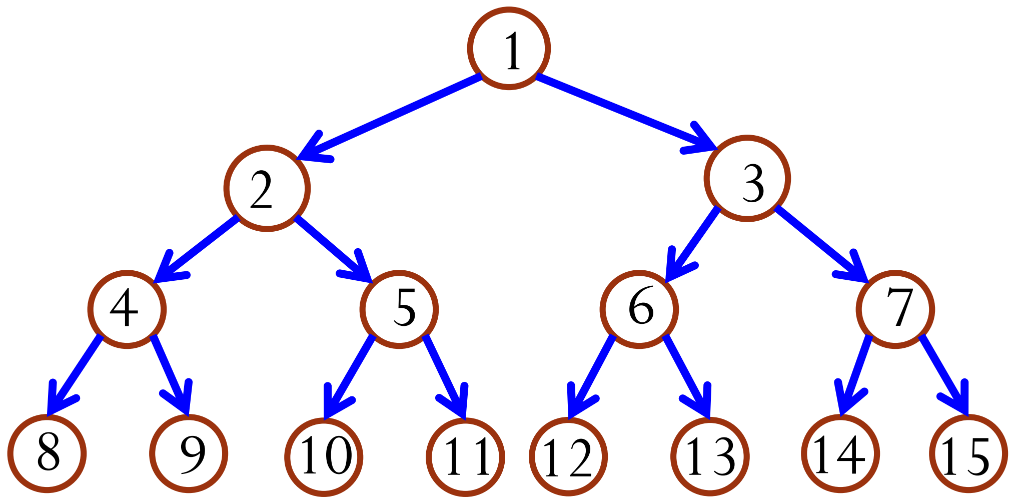


Numbering Nodes In a Perfect Binary Tree

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



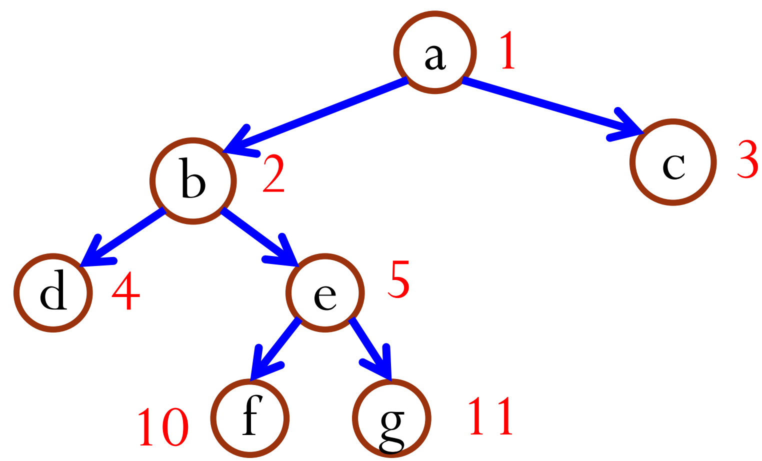
Numbering Nodes In a Perfect Binary Tree



- 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.

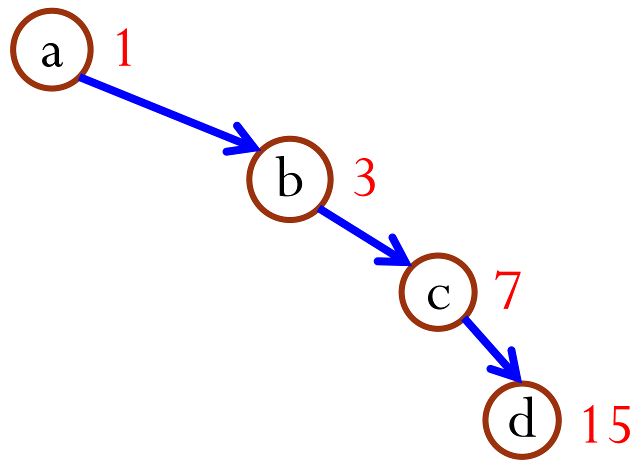


–	a	b	c	d	e	–	–	–	–	f	g
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]

Representing Binary Tree Using Array

Space Efficiency

- How would you represent a **right-skewed** binary tree?



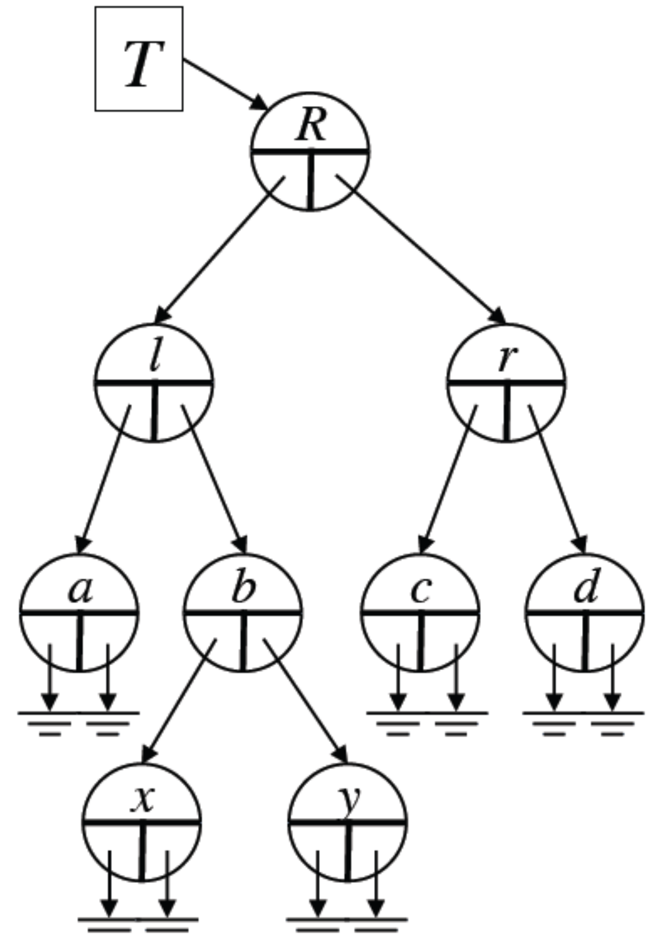
–	a	–	b	–	–	–	c	–	–	–	–	–	–	–	d
[0]	[1]		[3]		[5]		[7]		[9]		[11]		[13]		[15]

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

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.



Outline

- Trees
- Binary Trees
- Binary Tree Traversal

Binary Tree Traversal

- Many binary tree operations are done by performing a **traversal** of the binary tree.

- In a traversal, each node of the binary tree is visited **exactly once**.

- During the visit of a node, all actions (making a clone, displaying, evaluating the operator, etc.) with respect to this node are taken.

Binary Tree Traversal Methods

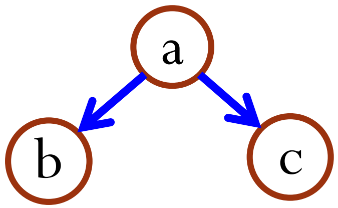
- Depth-first traversal
  - Pre-order
  - Post-order
  - In-order

- Level order traversal

Pre-Order Depth-First Traversal

Procedure

- Visit the node
- Visit its left subtree
- Visit its right subtree



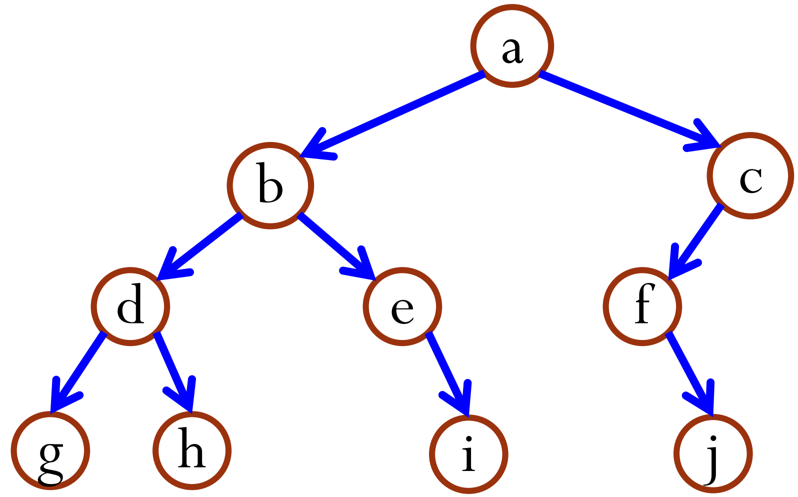
a→b→c

```
void preOrder(node *n) {
  if(!n) return;
  visit(n);
  preOrder(n->left);
  preOrder(n->right);
}
```

Pre-Order Depth-First Traversal

Example

a
b
d
g
h
e
i
c
f
j

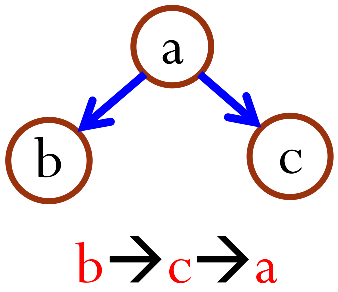


a→b→d→g→h→e→i→c→f→j

Post-Order Depth-First Traversal

Procedure

- Visit the left subtree
- Visit the right subtree
- Visit the node

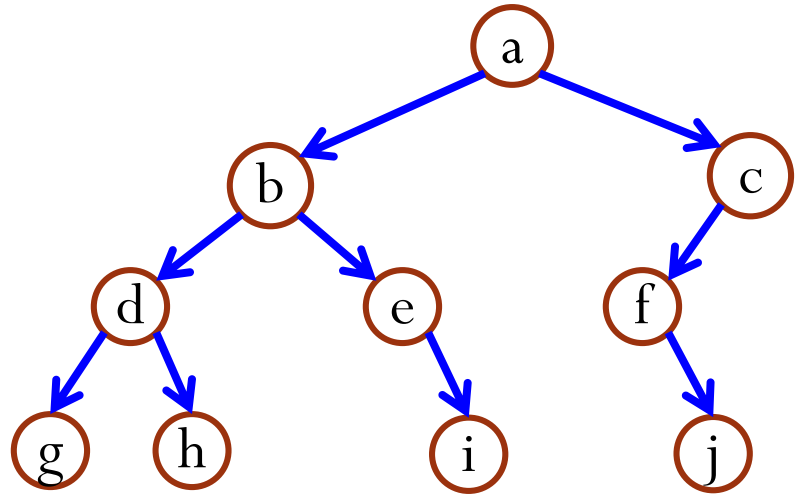



```
void postOrder(node *n) {
  if(!n) return;
  postOrder(n->left);
  postOrder(n->right);
  visit(n);
}
```

Post-Order Depth-First Traversal

Example

g
h
d
i
e
b
j
f
c
a

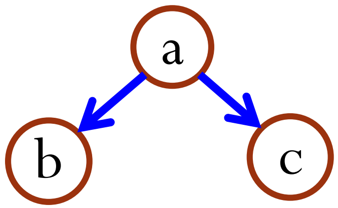


g→h→d→i→e→b→j→f→c→a

In-Order Depth-First Traversal

Procedure

- Visit the left subtree
- Visit the node
- Visit the right subtree

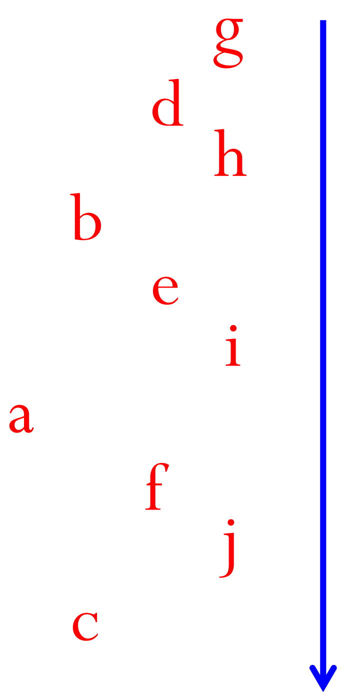


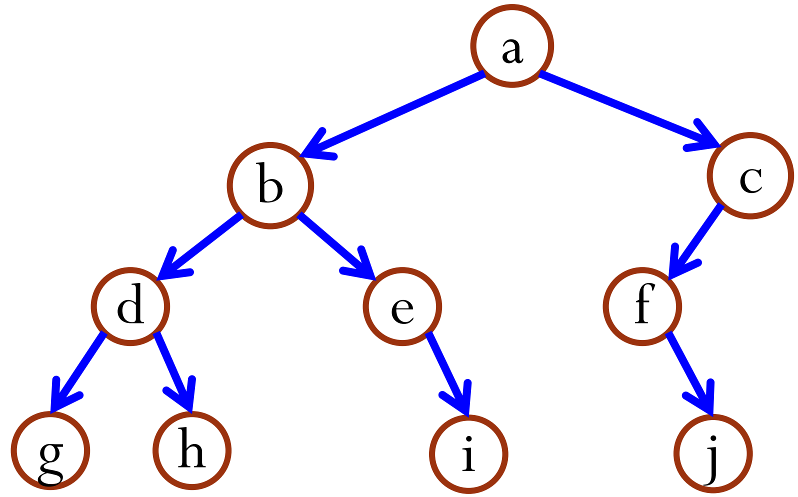
b→a→c

```
void inOrder(node *n) {
  if(!n) return;
  inOrder(n->left);
  visit(n);
  inOrder(n->right);
}
```

In-Order Depth-First Traversal

Example

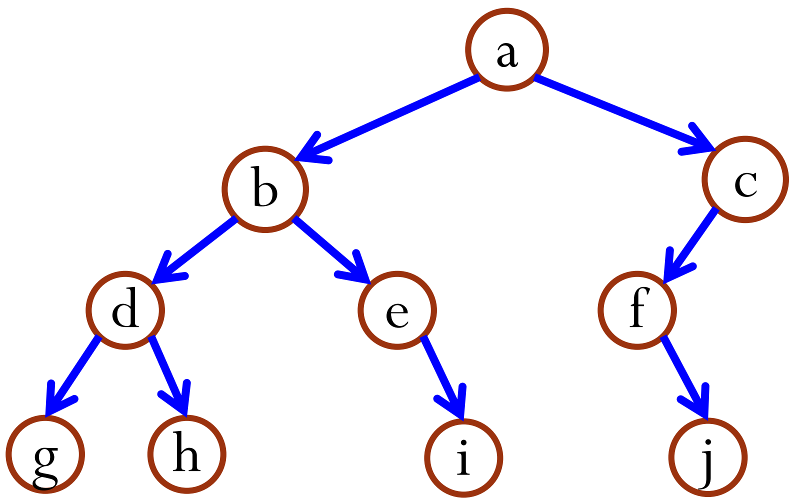




g→d→h→b→e→i→a→f→j→c

Level-Order Traversal

- We want to traverse the tree level by level **from top to bottom**.
- Within each level, traverse **from left to right**.



How can we implement this traversal?

a→b→c→d→e→f→g→h→i→j

Level-Order Traversal

Procedure

- Use a queue!

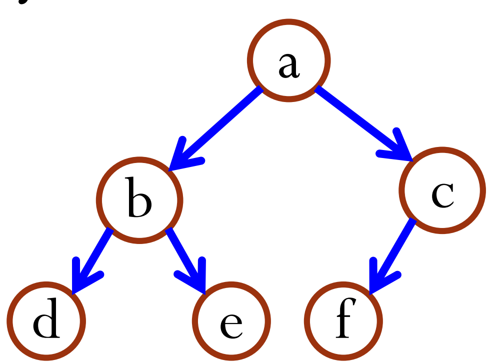
1. Enqueue the root node into an empty queue.
2. While the queue is not empty, dequeue a node from the front of the queue.
   1. Visit the node.
   2. Enqueue its left child (if exists) and right child (if exists) into the queue.

Loop

Level-Order Traversal

Code and Example

```
void levelOrder(node *root) {
  queue q; // Empty queue
  q.enqueue(root);
  while(!q.isEmpty()) {
    node *n = q.dequeue();
    visit(n);
    if(n->left) q.enqueue(n->left);
    if(n->right) q.enqueue(n->right);
  }
}
```



Queue: a b c d e f

Output: a b c d e f

Binary Tree Traversal

Application

- The expression $a/b + (c - d)e$ has been encoded as a tree T.
  - The leaves are **operands**.
  - The internal nodes are **operators**.
- How would you traverse the tree T to print out the expression (ignoring parentheses)?
  - In-order depth-first traversal.
- What is the expression printed out by post-order depth-first traversal?
  - $ab/cd - e * +$
  - **Reverse Polish Notation**