BITS Pilani
Pilani Campus

Data Structures & Algorithms Design- SS ZG519
Lecture - 8

Dr. Padma Murali

Lecture 8 Topics

- Binary Trees
- Binary Search Trees
- Heaps

Example

Insert the following values into a hash table of size 10 using the hash equation $(x^2 +1) \% 10$ using the linear probing, quadratic probing and separate chaining technique Insert these values in sequential order: 1,2,5,6,8,4,9,3,10,7

Hashing

A very efficient method for implementing a *dictionary,* i.e., a set with the operations:

- find
- insert
- delete

Based on representation-change and space-for-time tradeoff ideas

Important applications:

- symbol tables
- databases (*extendible hashing*)

Hash tables and hash functions

The idea of *hashing* is to map keys of a given file of size n into a table of size m, called the *hash table*, by using a predefined function, called the *hash function*,

$$h: K \rightarrow \text{location (cell) in the hash table}$$

Example: student records, key = SSN. Hash function:
$h(K) = K \bmod m$ where m is some integer (typically, prime)
If $m = 1000$, where is record with SSN= 314159265 stored?

Generally, a hash function should:

– be easy to compute

– distribute keys about evenly throughout the hash table

Collisions

If $h(K_1) = h(K_2)$, there is a *collision*

Good hash functions result in fewer collisions but some collisions should be expected (*birthday paradox*)

Two principal hashing schemes handle collisions differently:

- *Open hashing*
  – each cell is a header of linked list of all keys hashed to it
- *Closed hashing*
  - one key per cell
  - in case of collision, finds another cell by
    - *linear probing:* use next free bucket
    - *double hashing:* use second hash function to compute increment

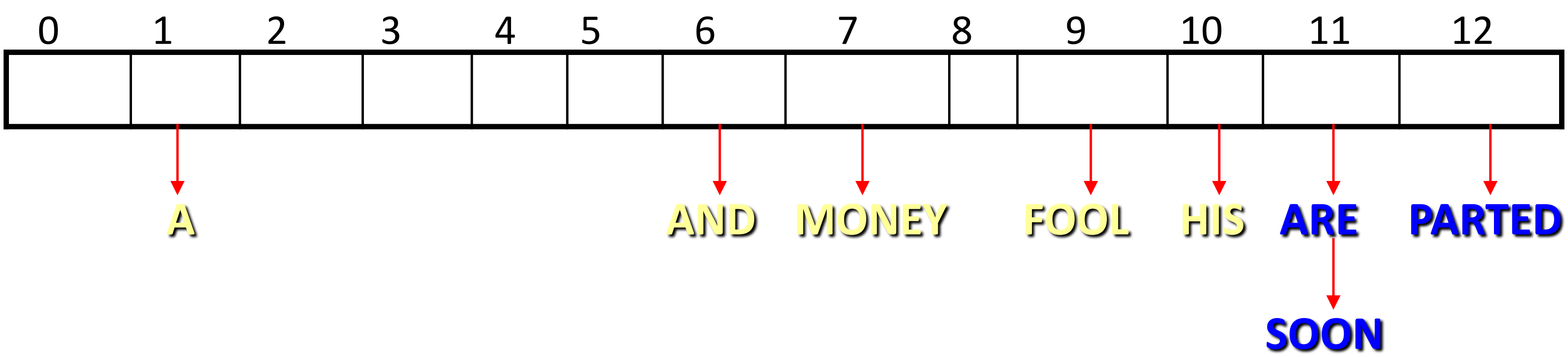
Keys are stored in linked lists outside a hash table whose elements serve as the lists' headers.

Example: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$ = sum of K 's letters' positions in the alphabet MOD 13

Open hashing (Separate chaining)

Key	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
h(*K*)	1	9	6	10	7	11	11	12



Search for KID

Open hashing (cont.)

If hash function distributes keys uniformly, average length of linked list will be $\alpha = n/m$. This ratio is called *load factor*.

Average number of probes in successful, S, and unsuccessful searches, U:

$$S \approx 1+\alpha/2, \qquad U = \alpha$$

Load α is typically kept small (ideally, about 1)

Open hashing still works if $n > m$

Closed hashing (Open addressing)

Keys are stored <u>inside</u> a hash table.

Key	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
$h(K)$	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Closed hashing (cont.)

Does not work if $n > m$

Avoids pointers

Deletions are *not* straightforward

Number of probes to find/insert/delete a key depends on load factor $\alpha = n/m$ (hash table density) and collision resolution strategy. For linear probing:

$$S = (½)\,(1+ 1/(1- \alpha)) \text{ and } U = (½)\,(1+ 1/(1- \alpha)^2)$$

As the table gets filled (α approaches 1), number of probes in linear probing increases dramatically:

Closed hashing

α	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

Lecture 8 Topics

- Binary Trees
- Binary Tree Traversals
- Heap Sort

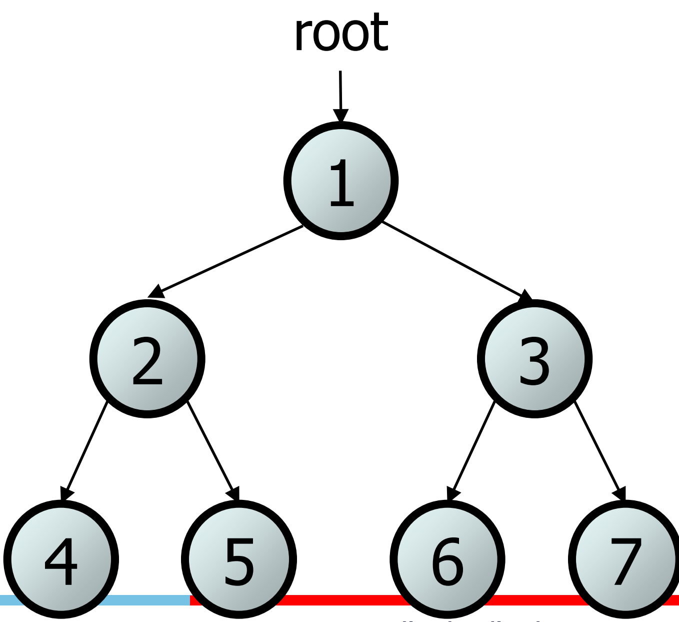
Binary Tree ADT

Tree: A directed, acyclic structure of linked nodes.

- *directed* : Has one-way links between nodes.
- *acyclic* : No path wraps back around to the same node twice.

Binary tree: One where each node has at most two children.

- *Recursive definition:* A tree is either:
  - empty (`null`), or
  - a **root** node that contains:
    - **data**,
    - a **left** subtree, and
    - a **right** subtree.
  - (The left and/or right subtree could be empty.)



Binary Tree ADT

Applications of Binary Tree

- Expression Tree (Parse Tree in Compilers)
- Decision trees (AI)
- Huffman Coding Tree

Types of Binary Tree

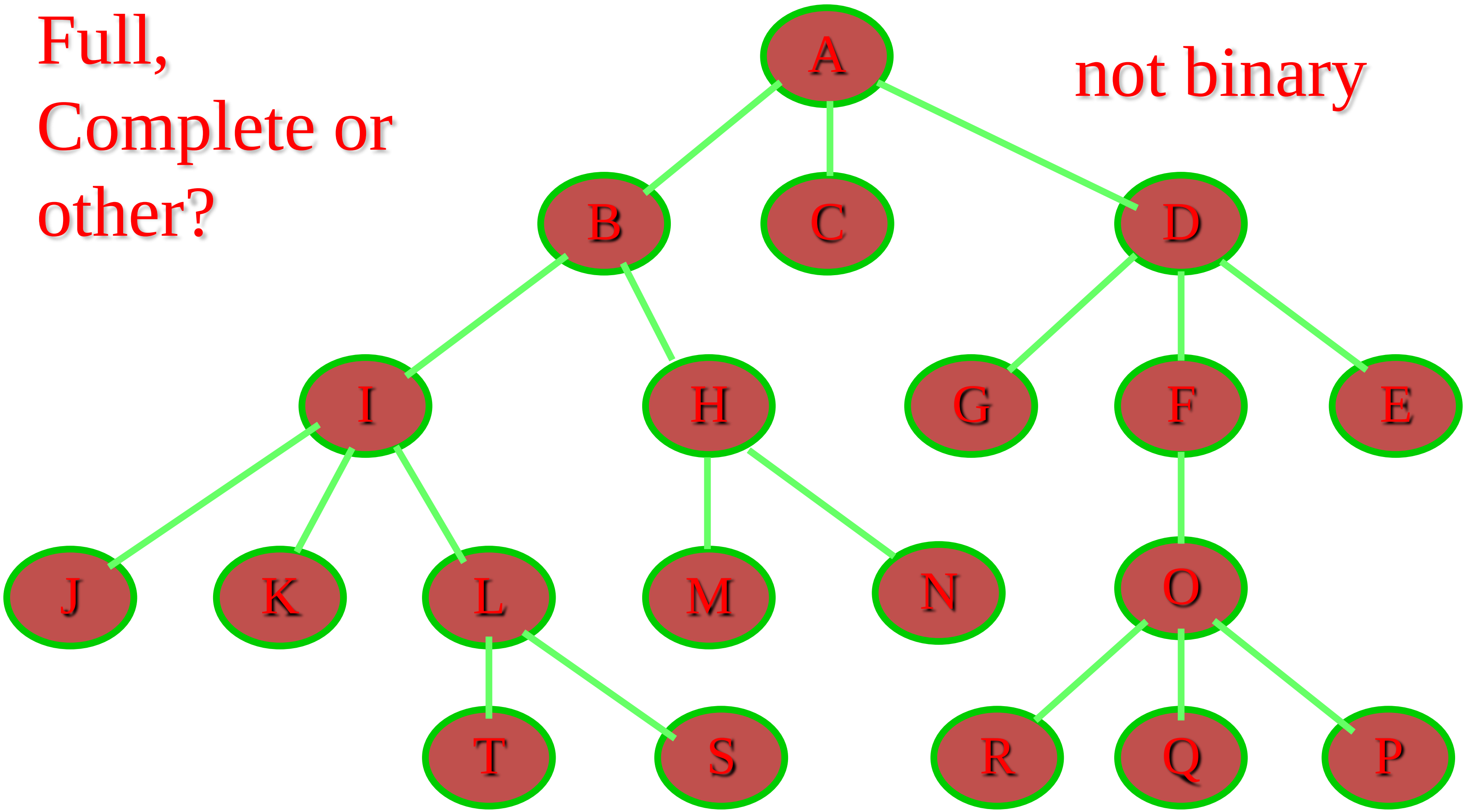
Full - Every node has exactly two children in all levels, except the last level. Nodes in last level have 0 children

Complete - Full up to second last level and last level is filled from left to right

Other - not full or complete

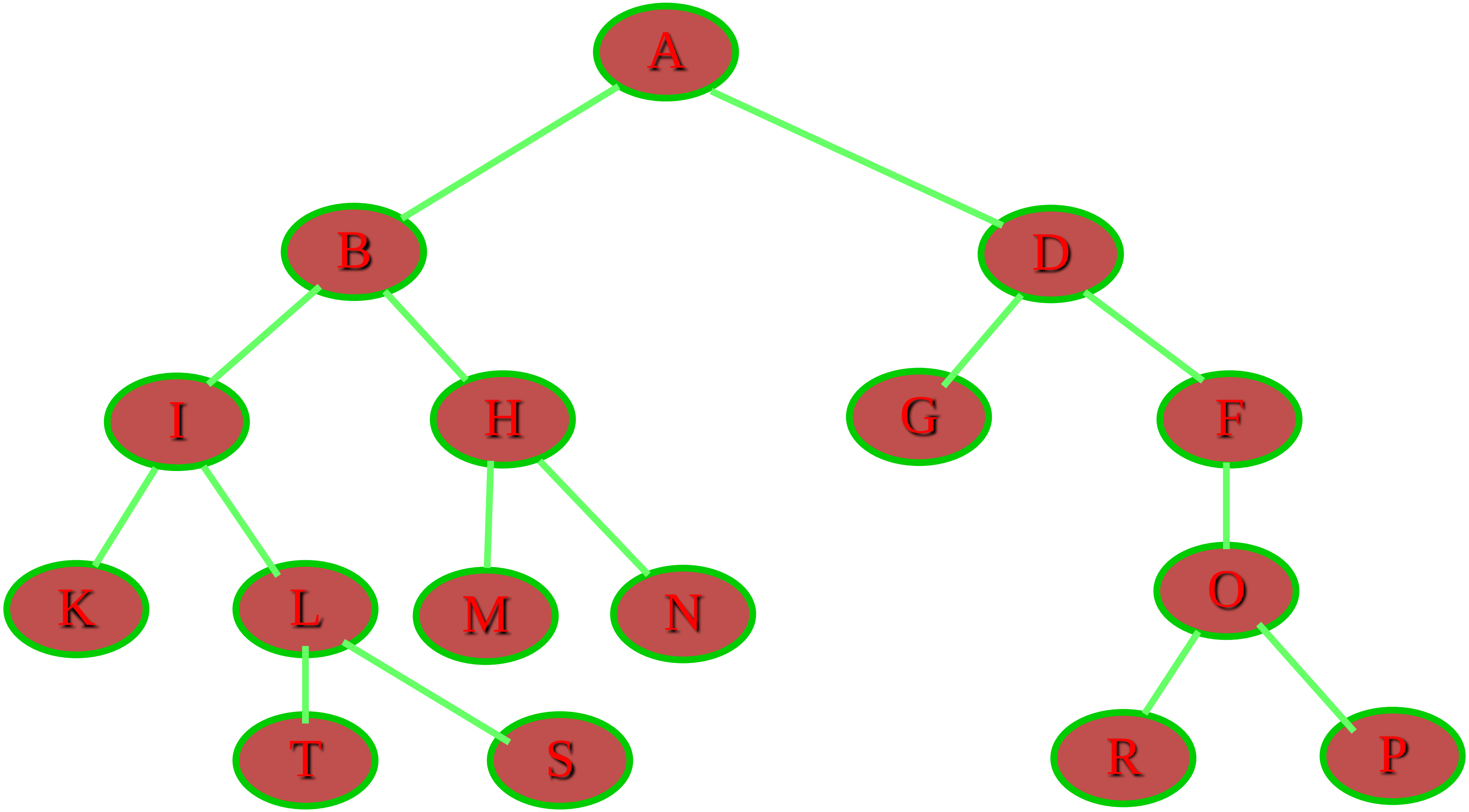
Types of Binary Tree

Full, Complete or other?

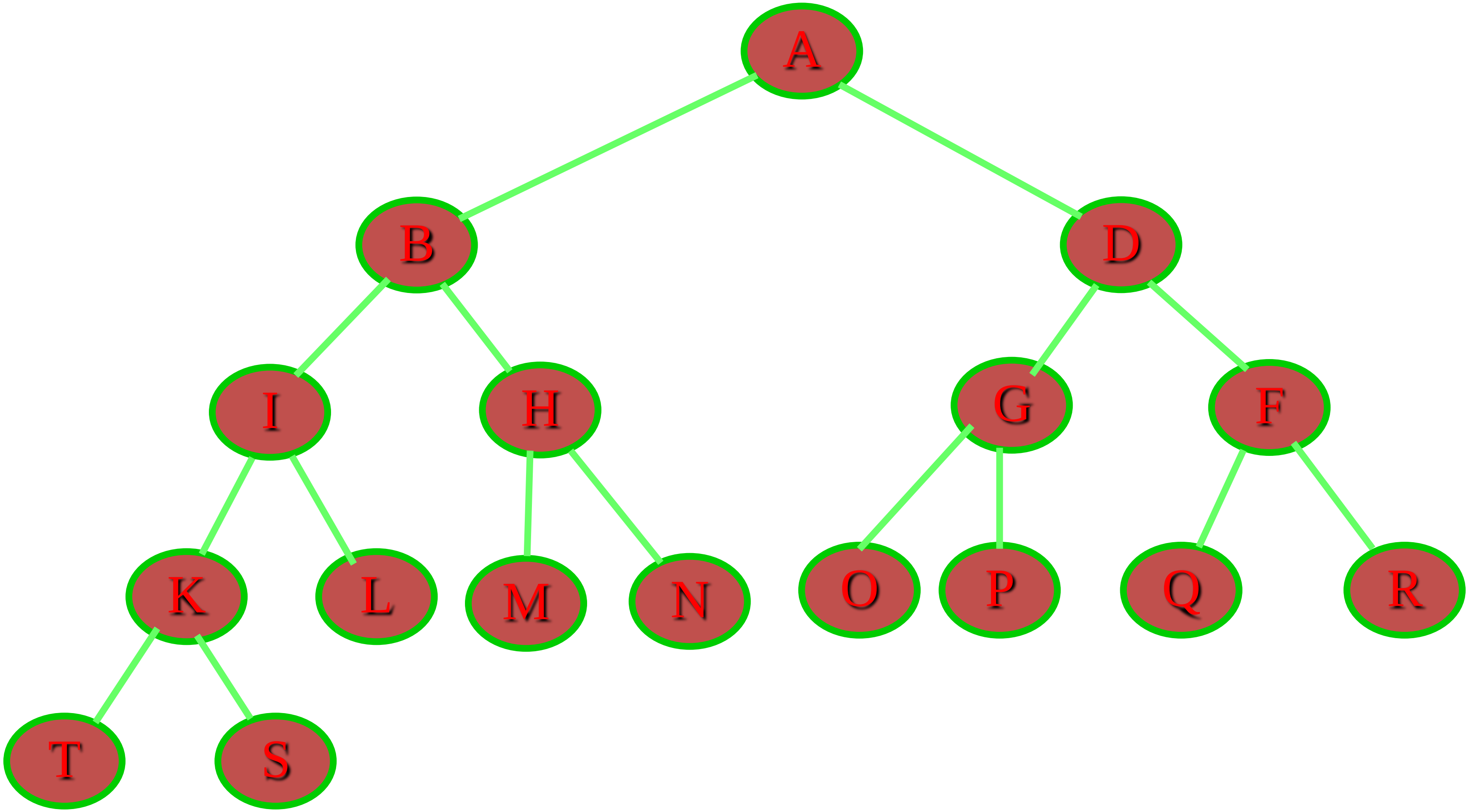


not binary

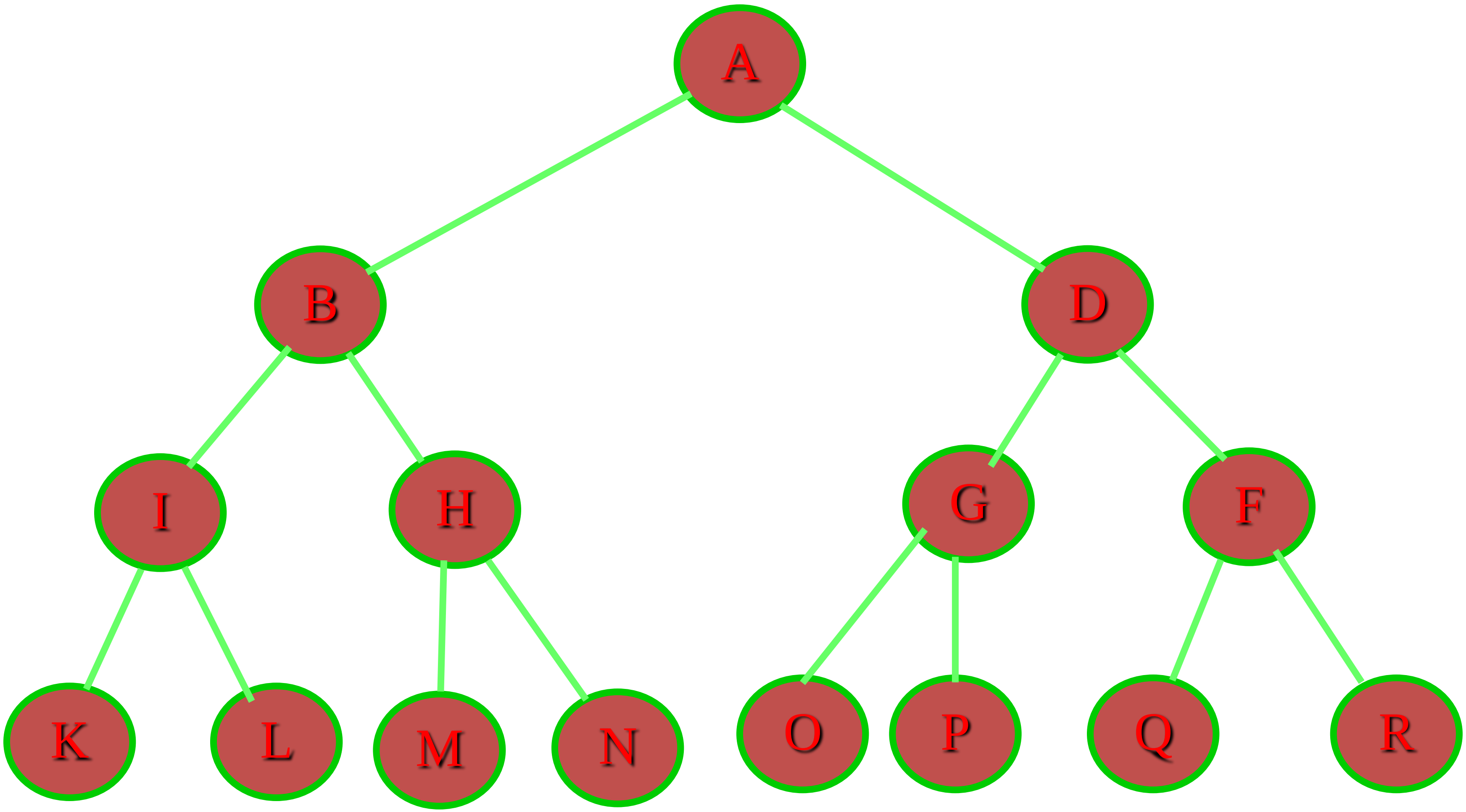
Types of Binary Tree



Types of Binary Tree

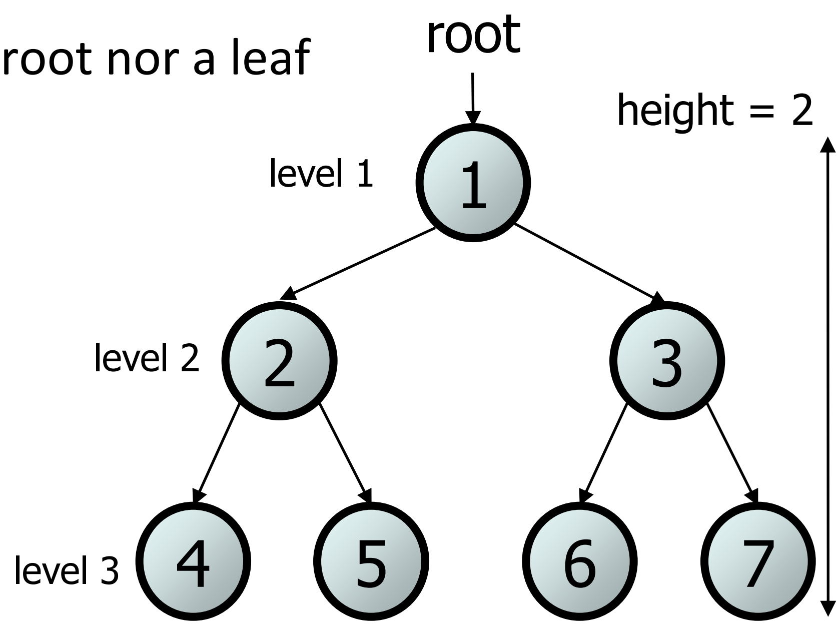


Types of Binary Tree



Terminology

- **node**: an object containing a data value and left/right children
    - **root**: topmost node of a tree
    - **leaf**: a node that has no children
    - **branch**: any internal node; neither the root nor a leaf
    - **parent**: a node that refers to this one
    - **child**: a node that this node refers to
    - **sibling**: a node with a common parent
- **subtree**: the smaller tree of nodes on the left or right of the current node
- **height**: length of the longest path from the root to any node
- **level** or **depth**: length of the path from a root to a given node



Binary Tree Traversal

- **Traversal:** An examination of the elements of a tree.
  - A pattern used in many tree algorithms and methods

1. **Preorder Traversal**

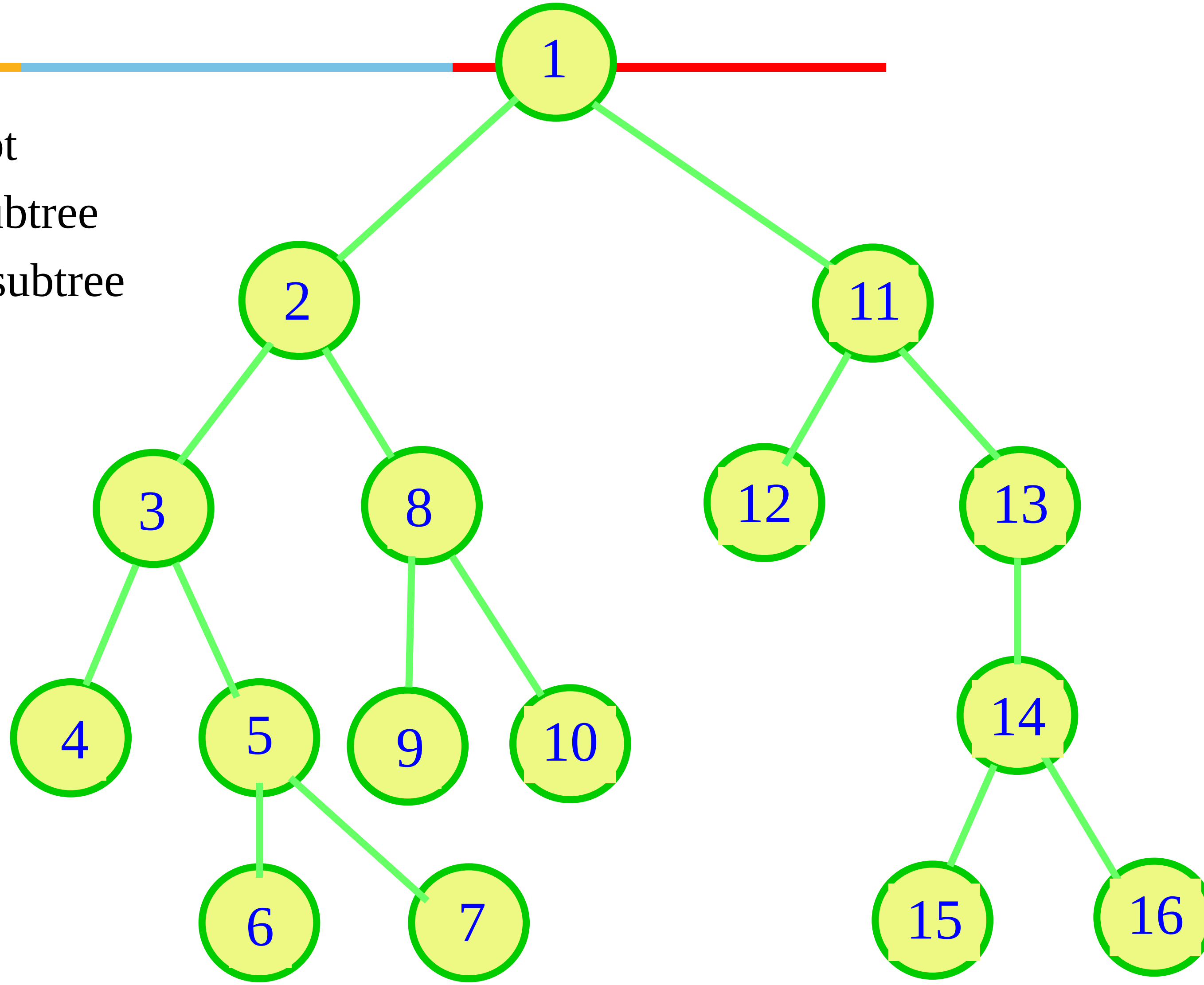
   Each node is processed before any node in either of its subtrees
2. **Inorder Traversal**

   Each node is processed after all nodes in its left subtree and before any node in its right subtree
3. **Postorder Traversal**

   Each node is processed after all nodes in both of its subtrees

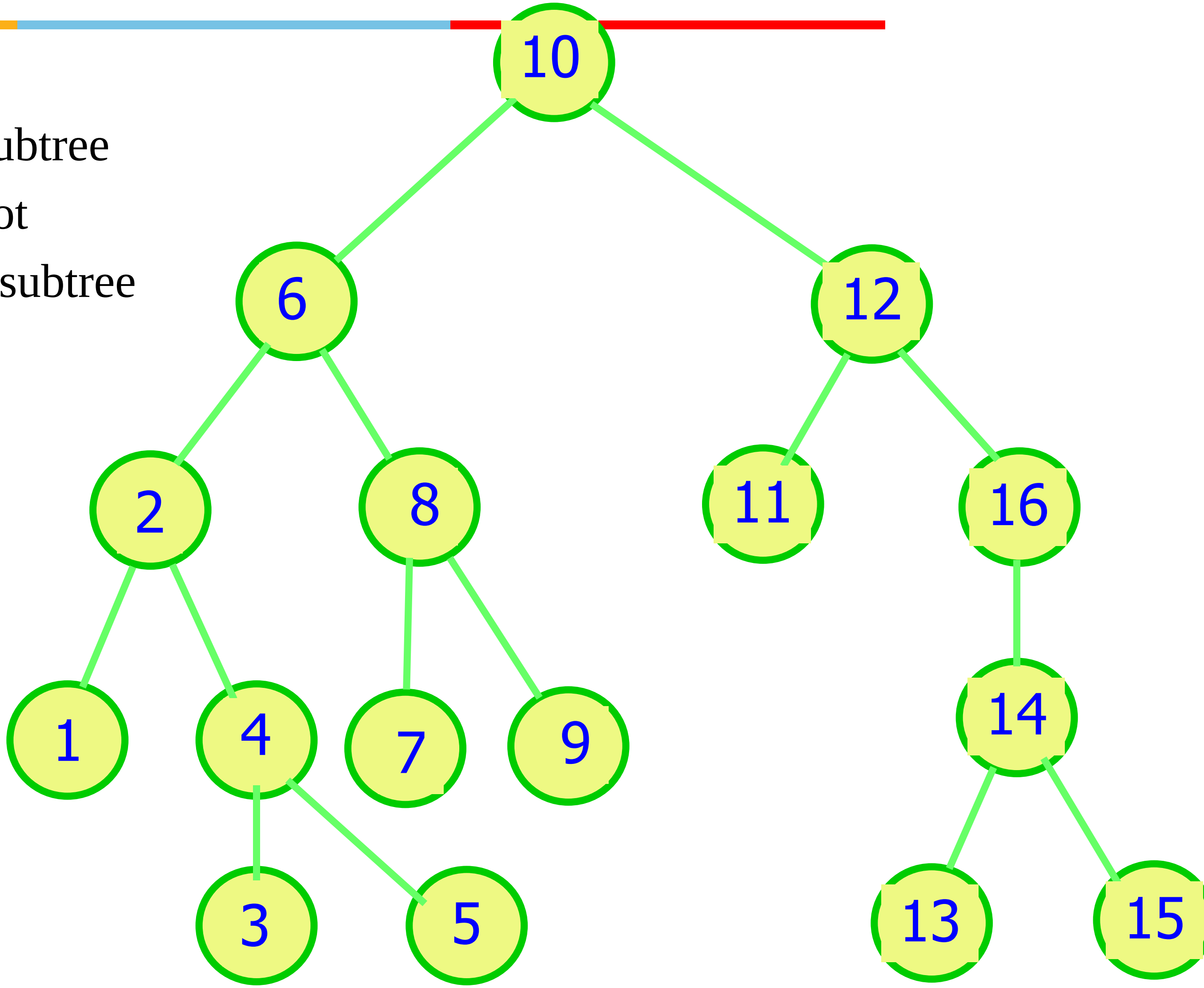
Preorder Traversal

1. Visit the root
2. Visit Left subtree
3. Visit Right subtree



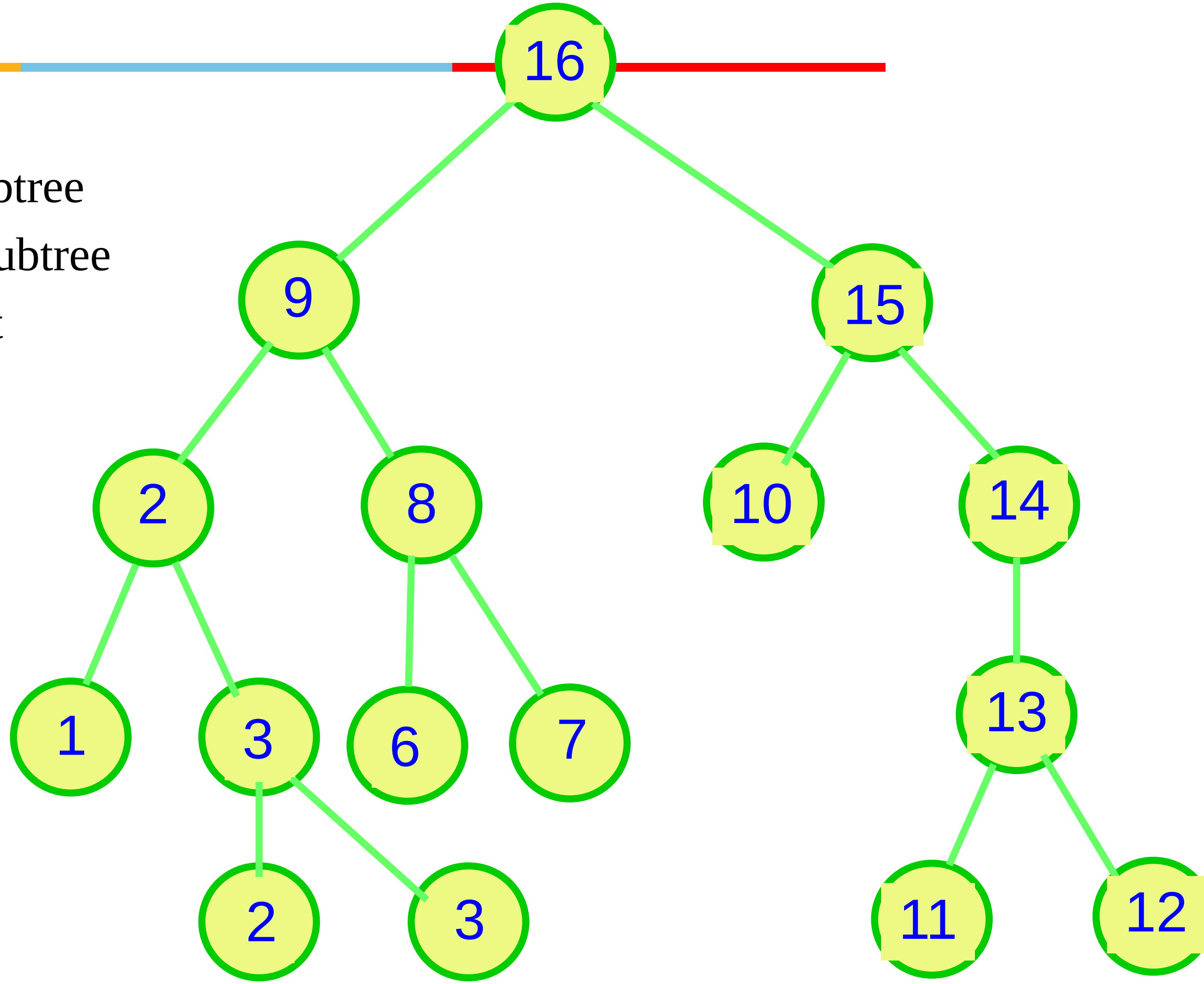
Inorder Traversal

1. Visit Left subtree
2. Visit the root
3. Visit Right subtree



Postorder Traversal

1. Visit Left subtree
2. Visit Right subtree
3. Visit the root



Represent ion of Binary Tree ADT

A binary tree can represented using

- **Linked List**
- **Array**

Note : Array is suitable only for full and complete binary trees

Height and the number of nodes in a Binary Tree

- If the height of a binary tree is *h* then the maximum number of nodes in the tree is 2^{h+1} -1
- If the number of nodes in a complete binary tree is *n*, then

$$2^{h+1} - 1 = n$$

$$2^{h+1} = n + 1$$

$$h+1 = \log(n+1)$$

$h = \log(n+1) – 1$ **----** **O(logn)**