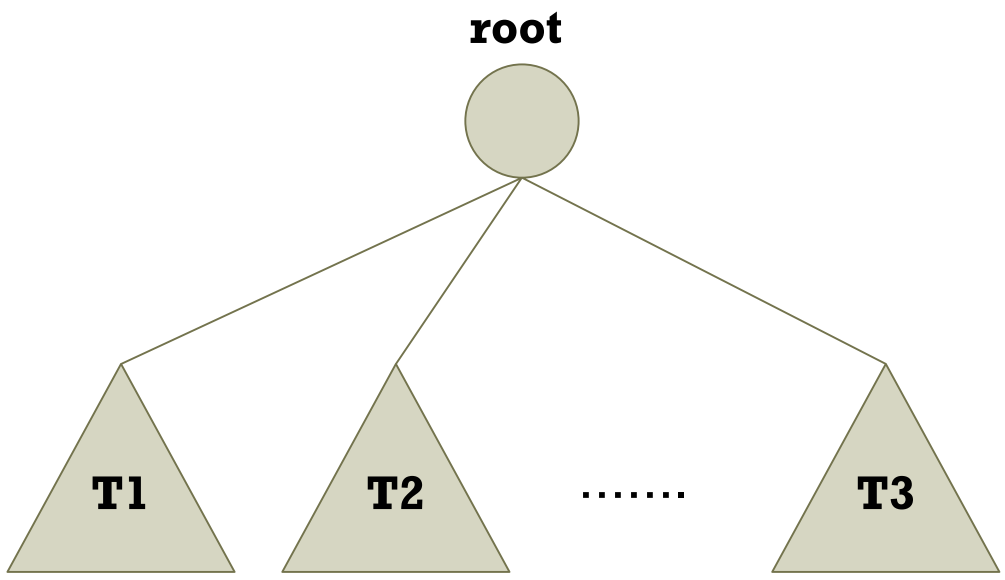
Tree

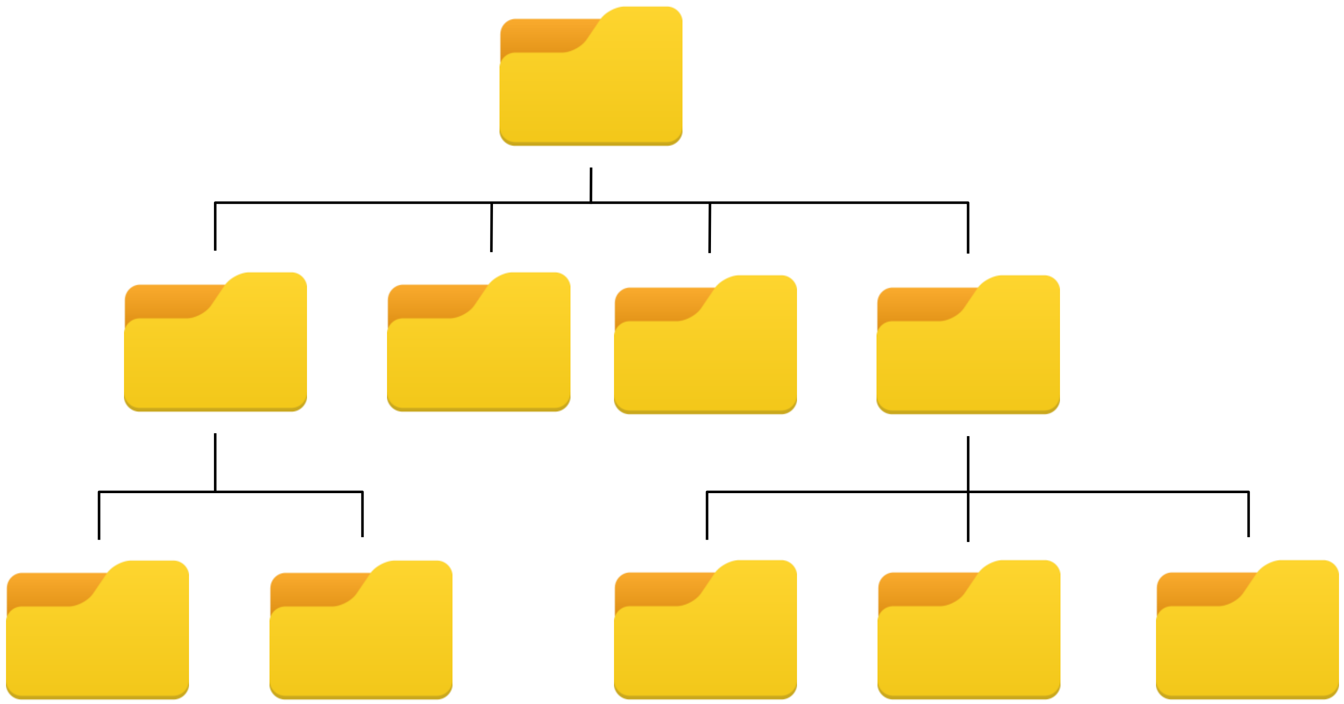
What is Tree?

- Definition
  - A collection of nodes to represent a **hierarchical** relationship
  - Each node has a value with a list of references to nodes.
    - Each node is composed with the **parent-child relationship**.
  - **Acyclic graph**: contain no cycle.



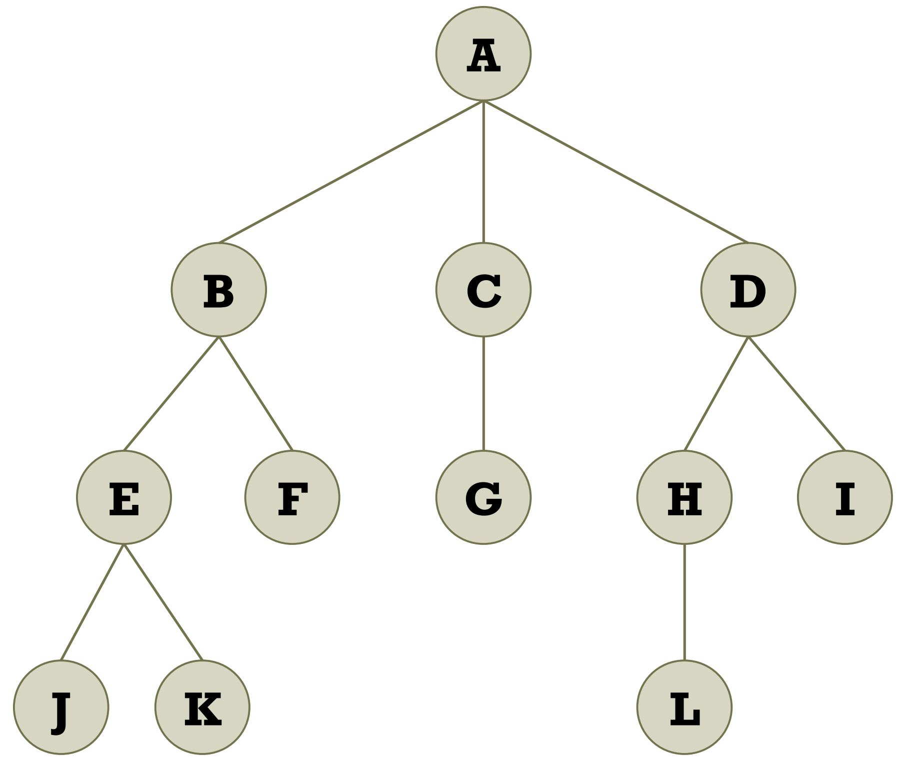
Tree Example

- File directory structure



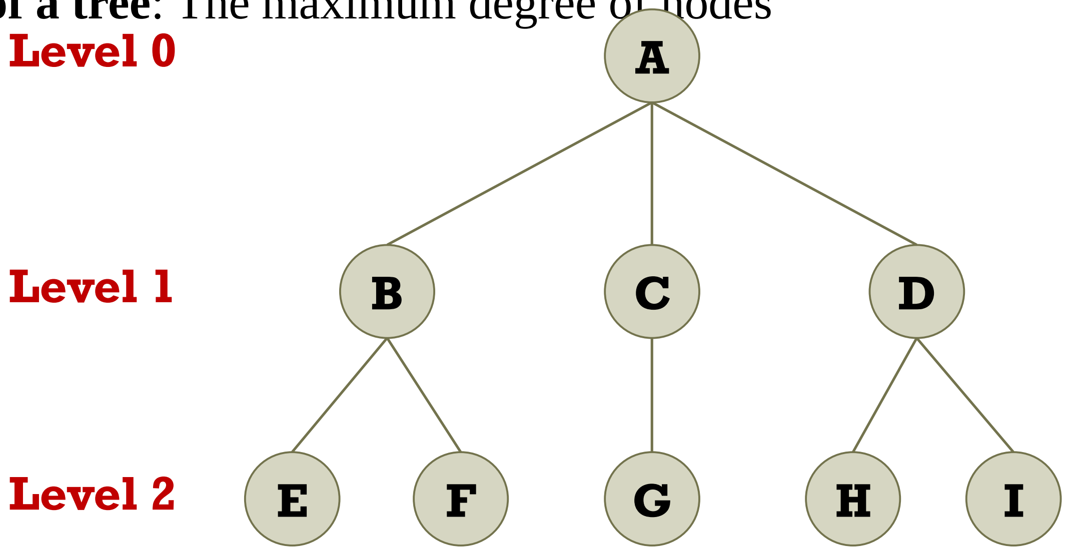
Basic Notations

- Terminology
  - **Node**: a basic component in a tree
  - **Edge**: The connection between one node and another
  - **Root**: the top node in a tree
    - **A**
  - **Internal node** (**non-terminal node**): a node with one or more degrees
    - **A, B, C, D, E, H**
  - **Leaf node** (**terminal node**): a node with degree zero
    - **F, G, I, J, K, L**



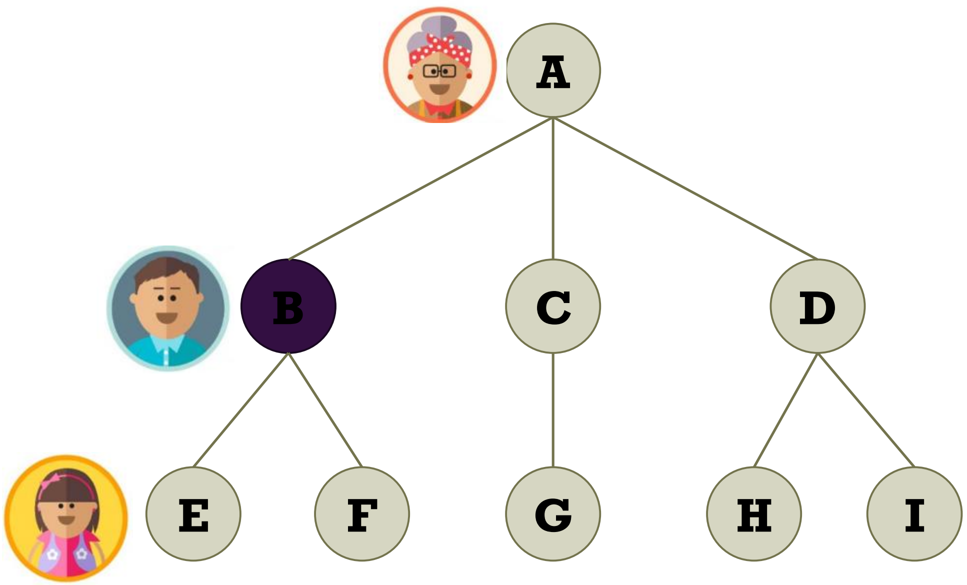
Basic Notations

- Terminology
  - **Level**: The level of a node is its distance from the root.
    - The level of the root node is 0.
  - **Height (depth)**: The longest path (number of nodes) from a root to the farthest leaf
    - **The height of a tree**: The maximum level of nodes plus 1
  - **Degree**: The number of subtrees of a node
    - **Degree of a tree**: The maximum degree of nodes



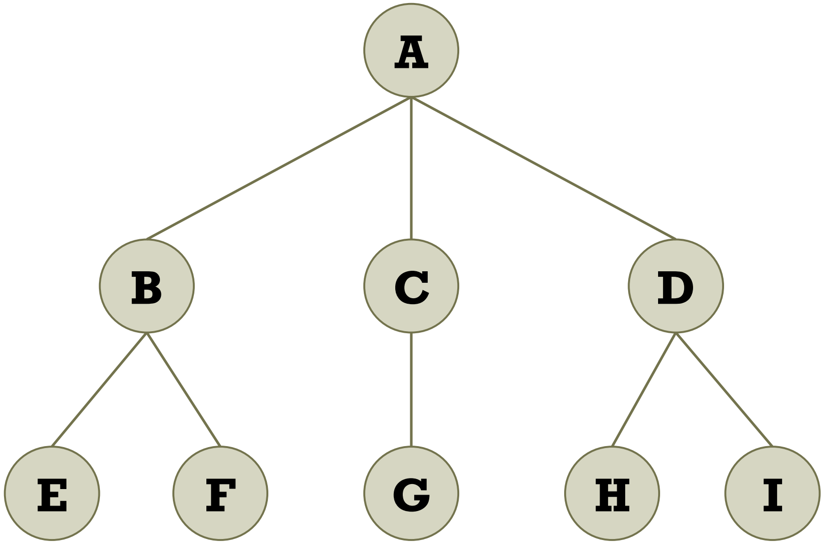
Basic Notations

- Terminology
  - **Parent**: a node that has one or more subtrees.
  - **Child**: a node that is connected from a parent node.
  - **Sibling**: a set of nodes with the same parent
  - **Ancestor**: all nodes along the path from the root to the node
  - **Descendant**: all nodes that are in a subtree



Basic Notations

- Example
  - A is the **root** node.
  - B is the **parent** of nodes E and F.
  - C and D are **children** of node A.
  - B and C are **sibling** nodes.
  - A and D are **ancestors** of node I.
  - H and I are **descents** of node A.

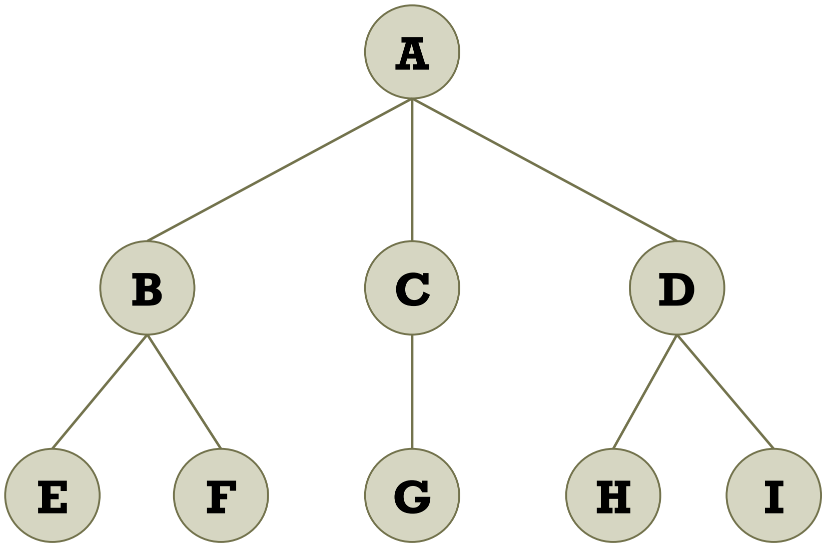


Representation of Tree

- Array representation for child nodes
  - Store the children with an array pointer.

item	*link_1*	*link_2*	...	*link_n*

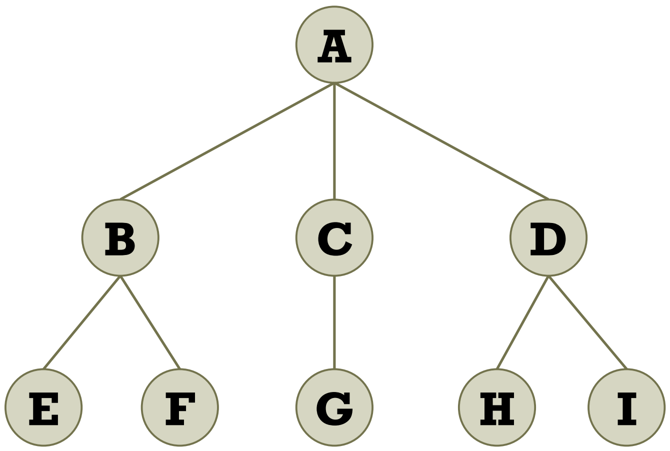
  - *n* is the degree of the tree.

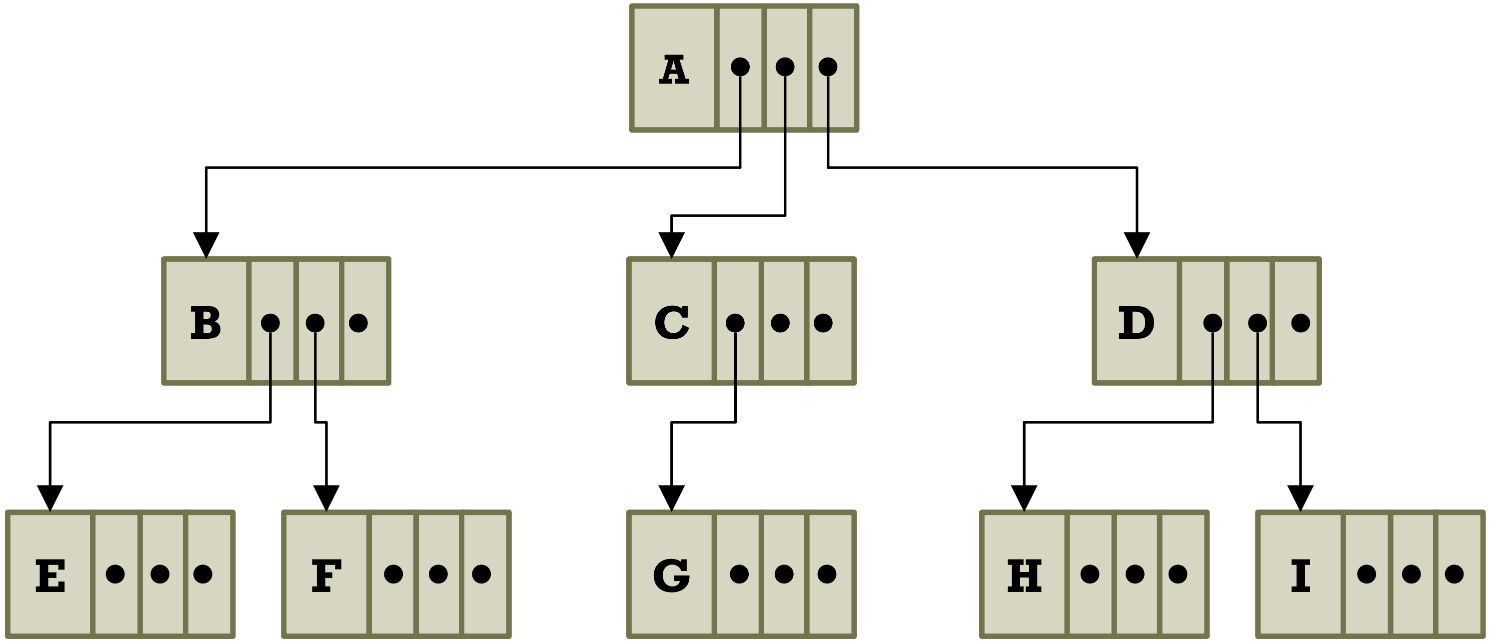


The degree of the tree is 3.

Representation of Tree

- Array representation for child nodes
  - Each node consist of a value and a list of node pointers.



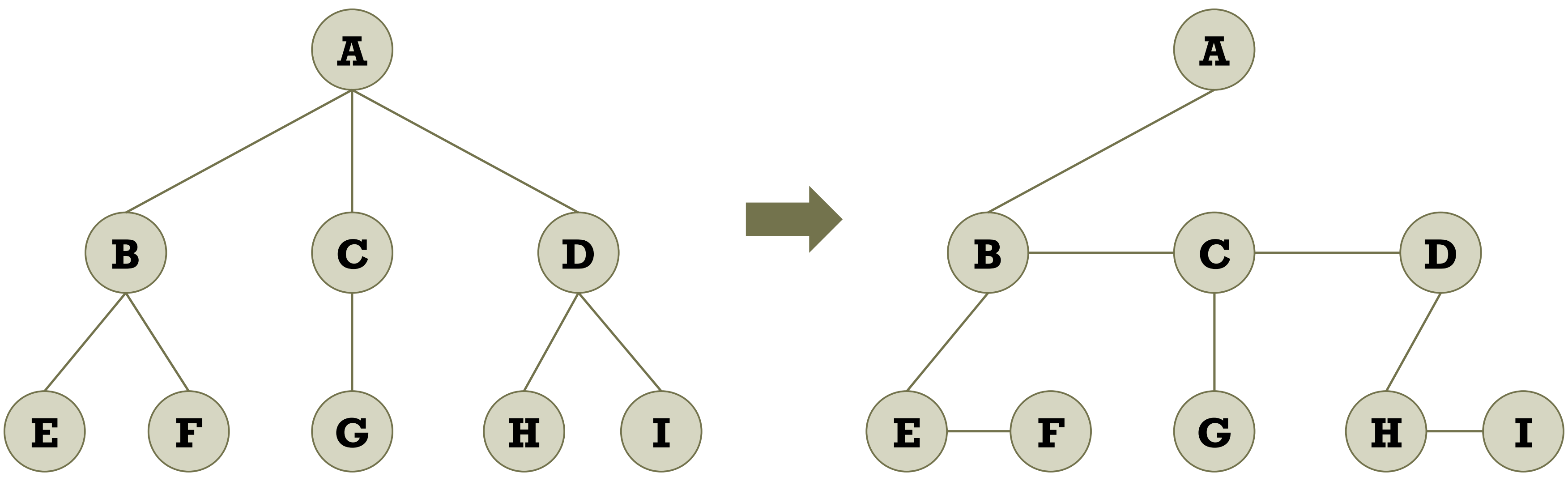


Representation of Trees

- Left-child right-sibling representation
  - Nodes of a fixed size: Two link fields per node

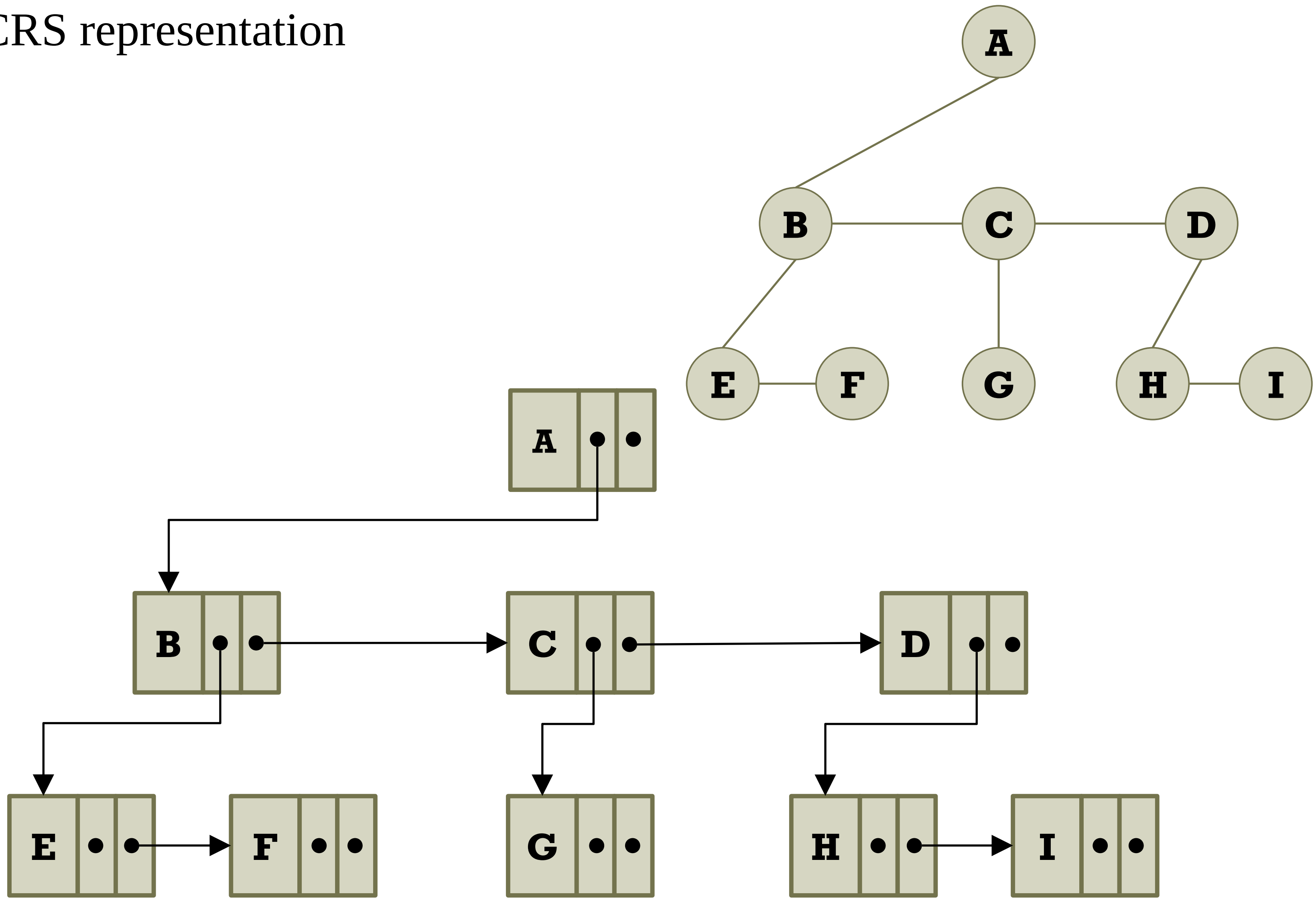
item	*left_child*	*right_sibling*

  - Easy to manage the tree



Representation of Trees

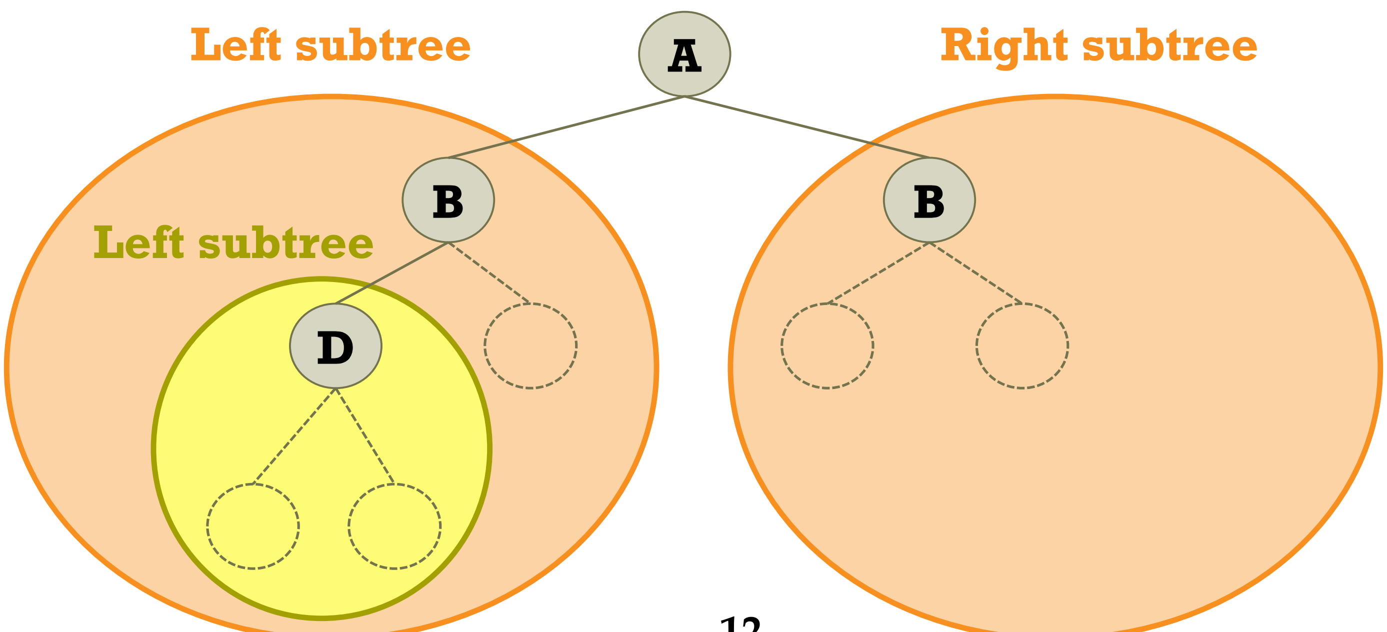
- LCRS representation



What is Binary Tree?

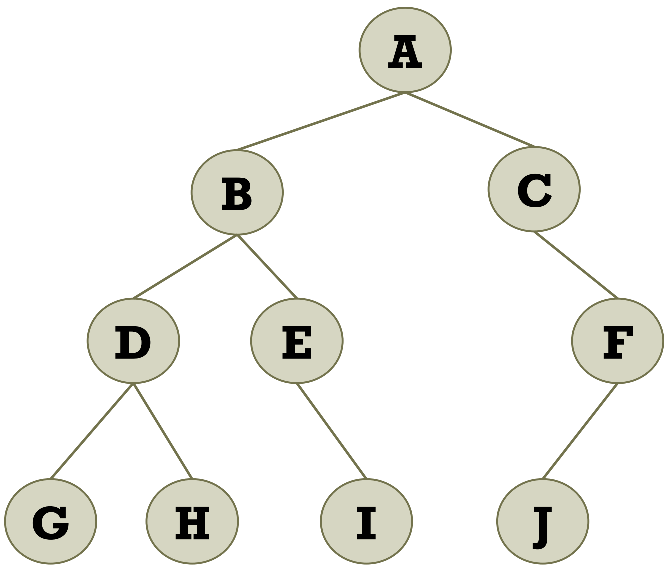
- Definition
  - A binary tree is a finite set of nodes such that
    - 1) empty or
    - 2) consists of root node and **at most two disjoint binary trees**, called left subtree and right subtree

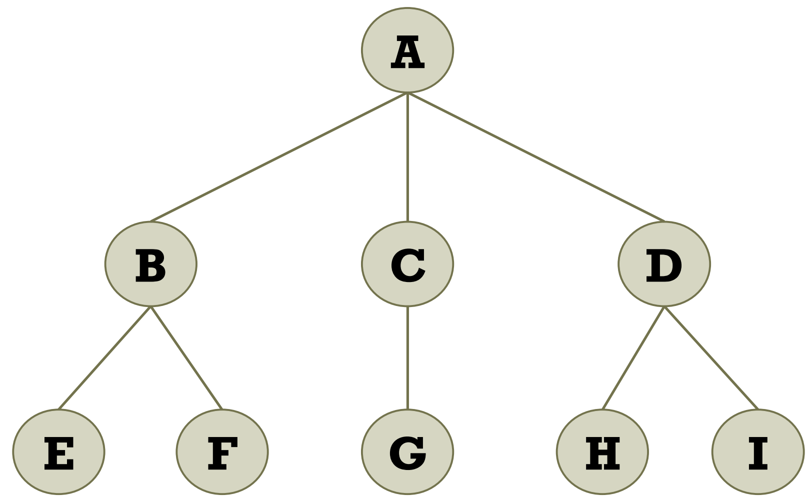
Q: Is it a binary tree?



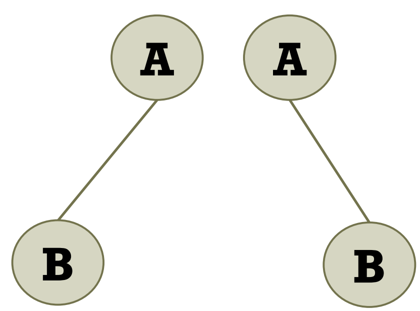
What is Binary Tree?

- Q: Are they binary trees?



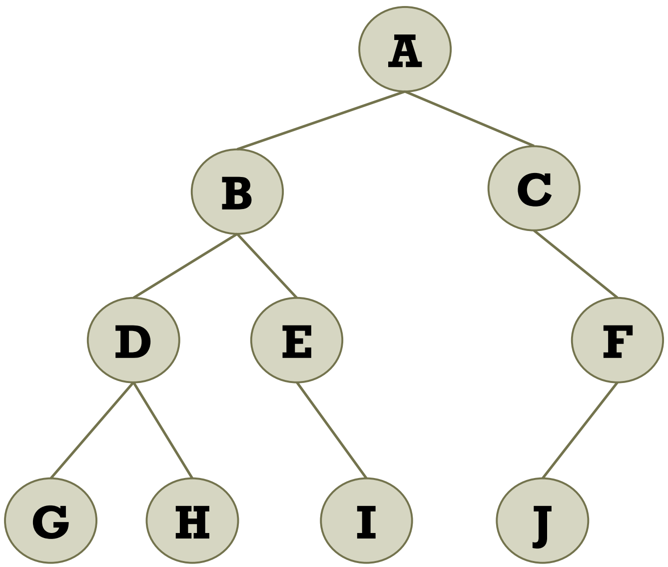


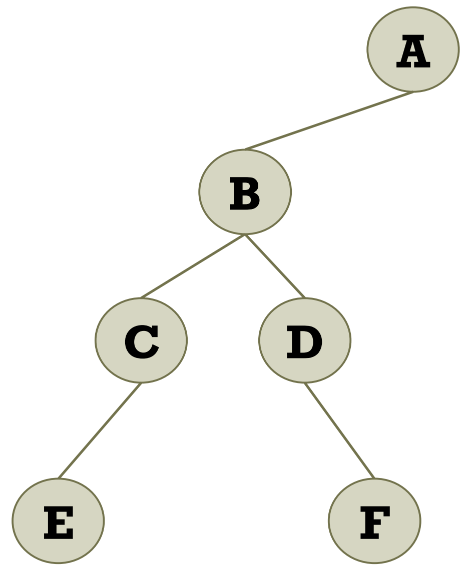
- Q: Are they same?



Properties of Binary Tree

- If a binary tree has $\boldsymbol{n}$ nodes, it has $\boldsymbol{n} - \mathbf{1}$ edges. (Prove it !!)



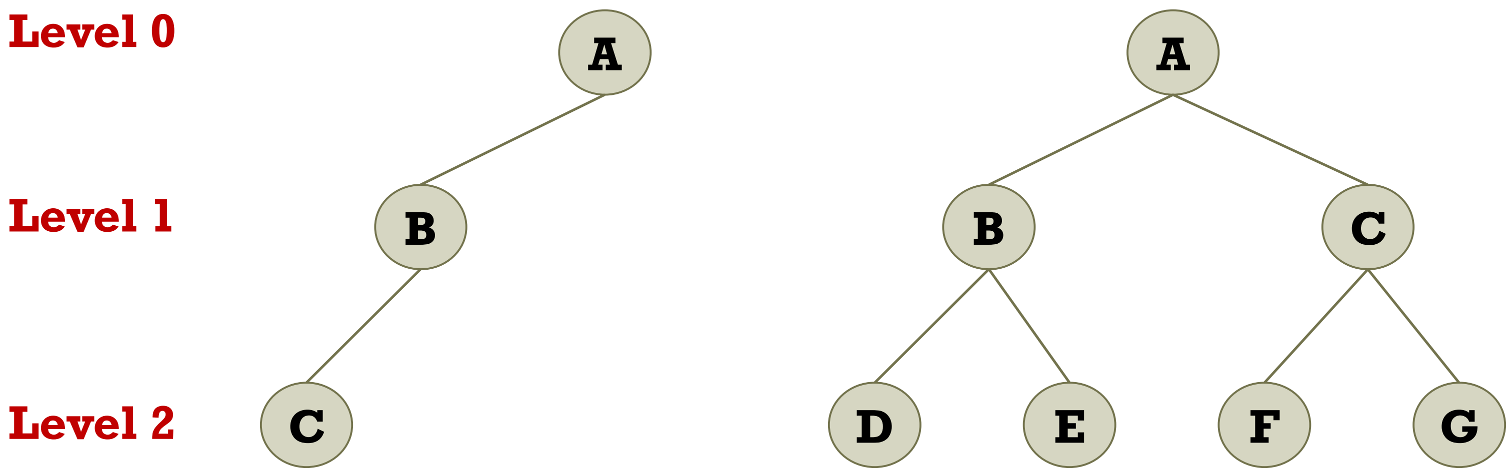


# of nodes: 10, # of edges: 9

# of nodes: 6, # of edges: 5

Properties of Binary Tree

- Suppose that the height of a binary tree is $\boldsymbol{k}$.
  - Minimum number of nodes: $\boldsymbol{k}$
  - Maximum number of nodes: $\mathbf{2^{\boldsymbol{k}} - 1}$

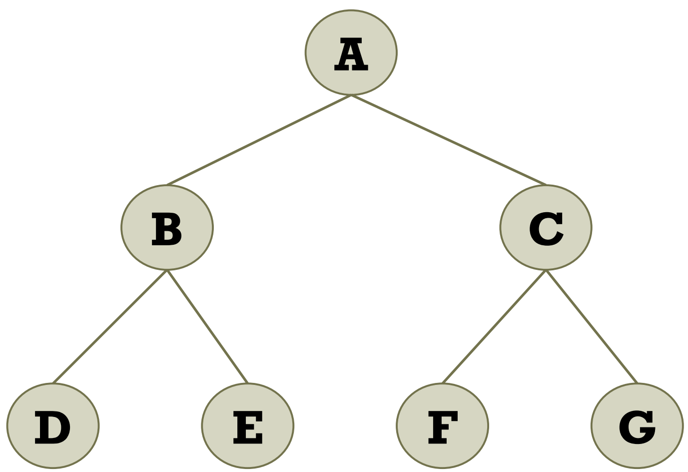


number of nodes: 1 + 1+ 1 = 3

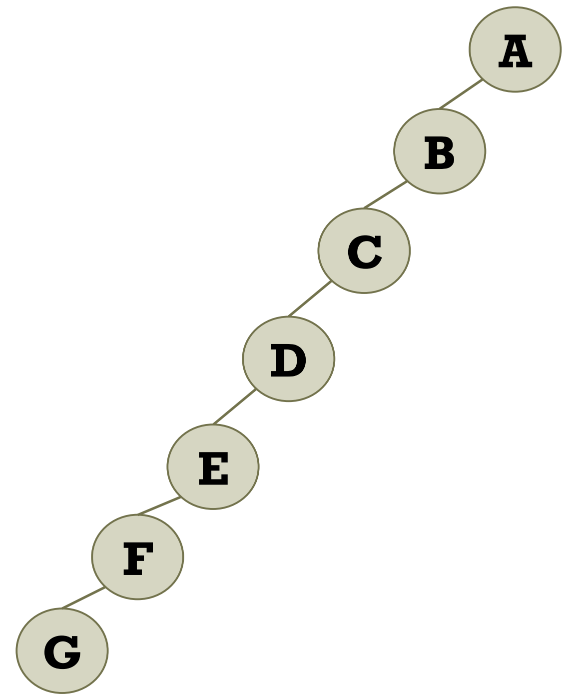
number of nodes: 1 + 2 + 4 = 7

Properties of Binary Tree

- Suppose that the number of nodes in a tree is $\boldsymbol{n}$.
  - Minimum height of the tree: $\lceil \boldsymbol{log_2(n+1)} \rceil$
  - Maximum height of the tree: $\boldsymbol{n}$



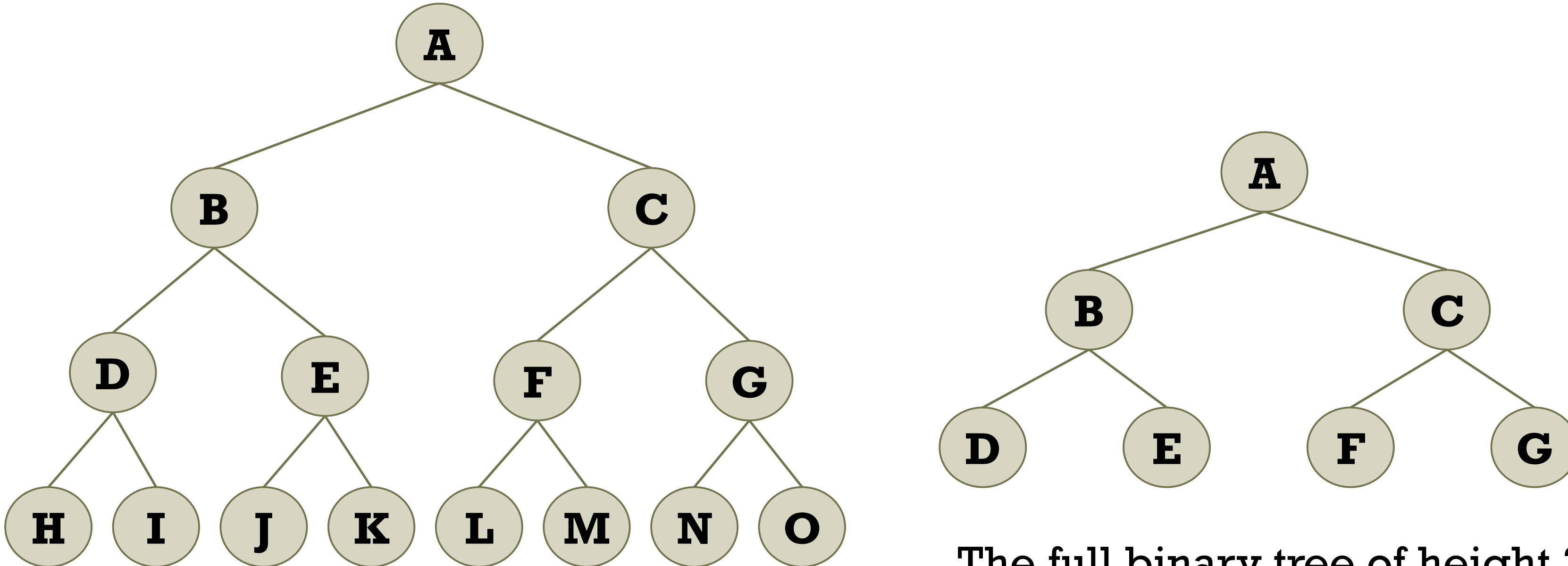
Height of the tree: 3



Height of the tree: 7

Perfect? Full Binary Tree

- The full binary tree of height $\boldsymbol{k}$ has $\mathbf{2^k - 1}$ nodes.

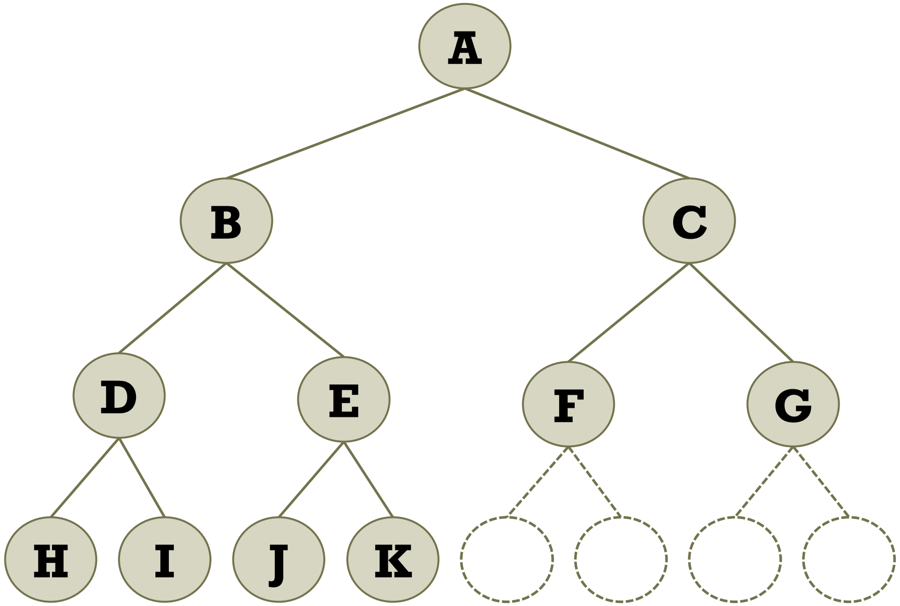


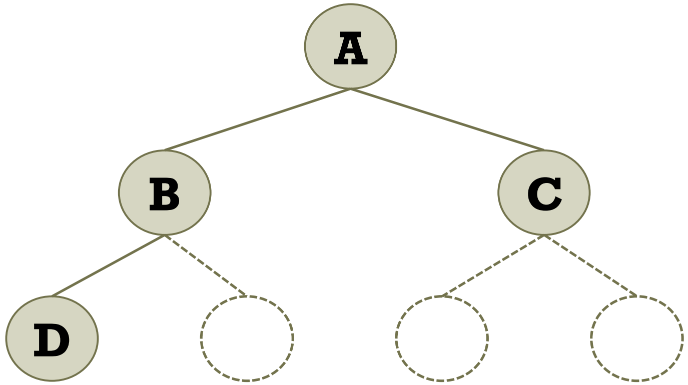
The full binary tree of height 4 has 15 nodes.

The full binary tree of height 3 has 7 nodes.

Complete Binary Tree

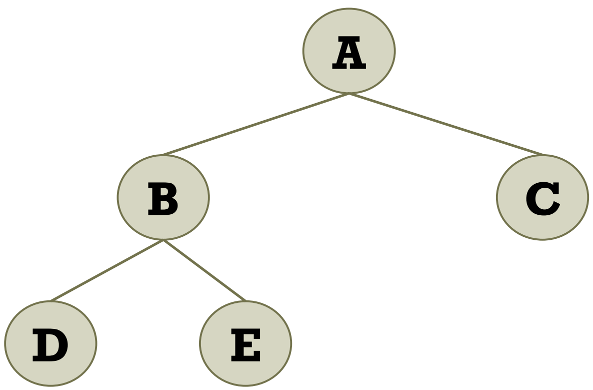
- Complete binary tree
  - A binary tree with n nodes that correspond to the nodes numbered from 1 to n in the full binary tree of height k

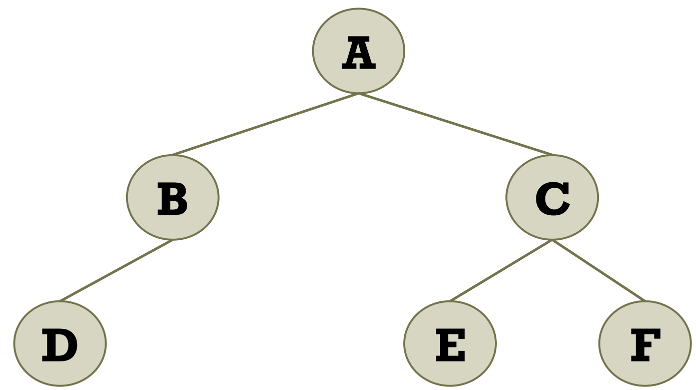




Full vs. Complete Binary Tree

- Q: Is it a full binary tree, a complete binary tree, or none?

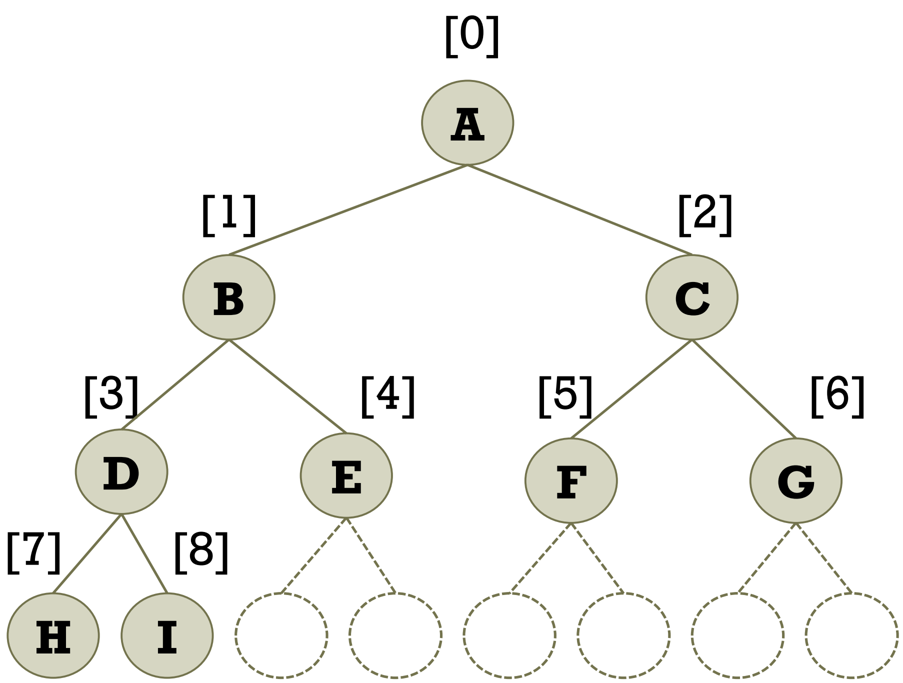




- Q: If a binary tree is the full binary tree, then it is also a complete binary tree. True of False.

Array Representation

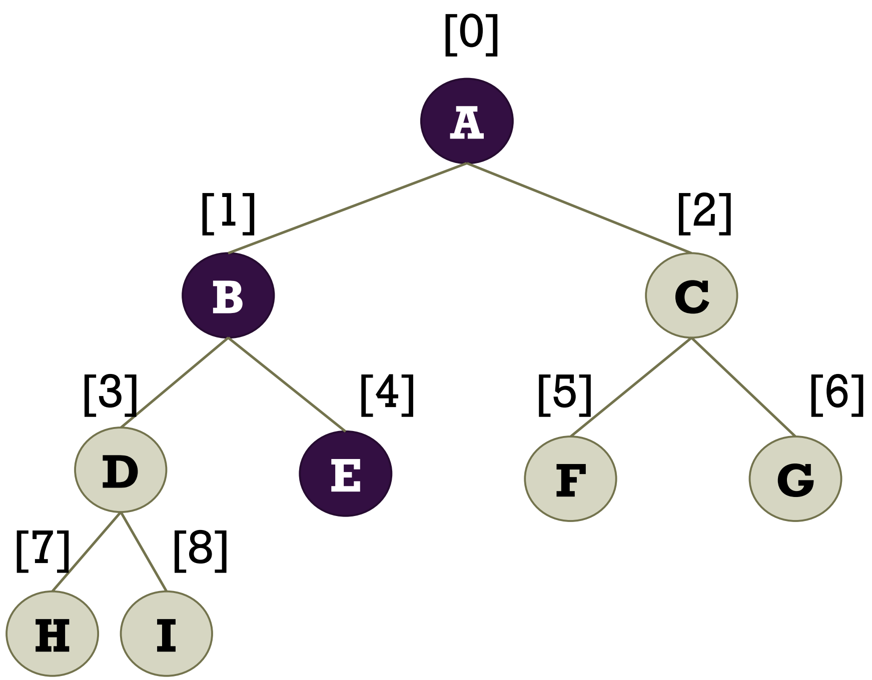
- How to represent a binary tree by an array
  - The mapping for a binary tree corresponds to the nodes numbered from 1 to n in the full binary tree of height k



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

Array Representation

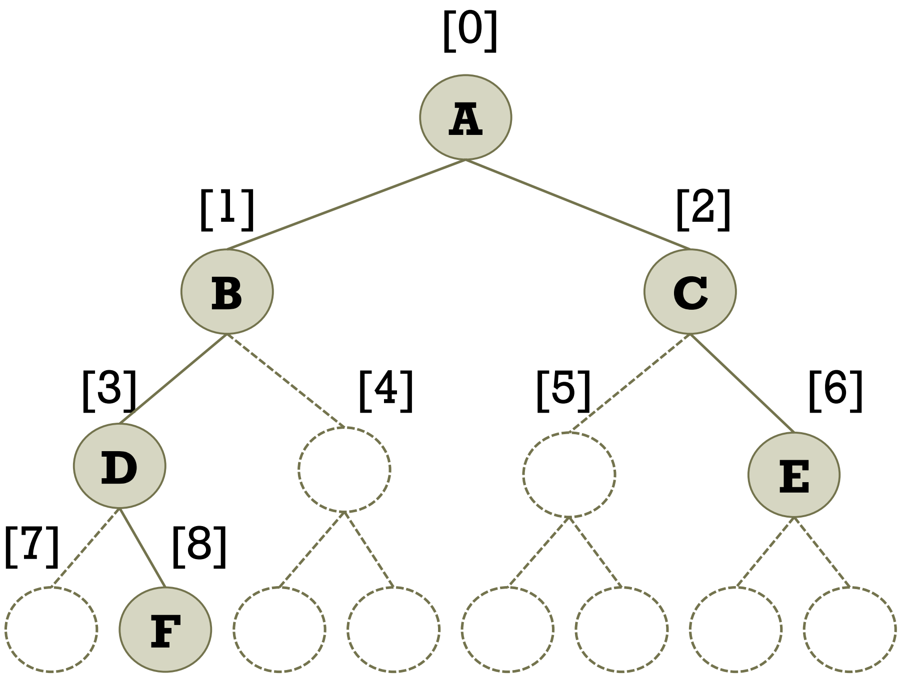
- Rule for array representation
  - The parent node i is at $\lfloor (i-1)/2 \rfloor$.
  - The left child of node i is at $2(i+1)-1$.
  - The right child of node i is at $2(i+1)$.



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

Array Representation

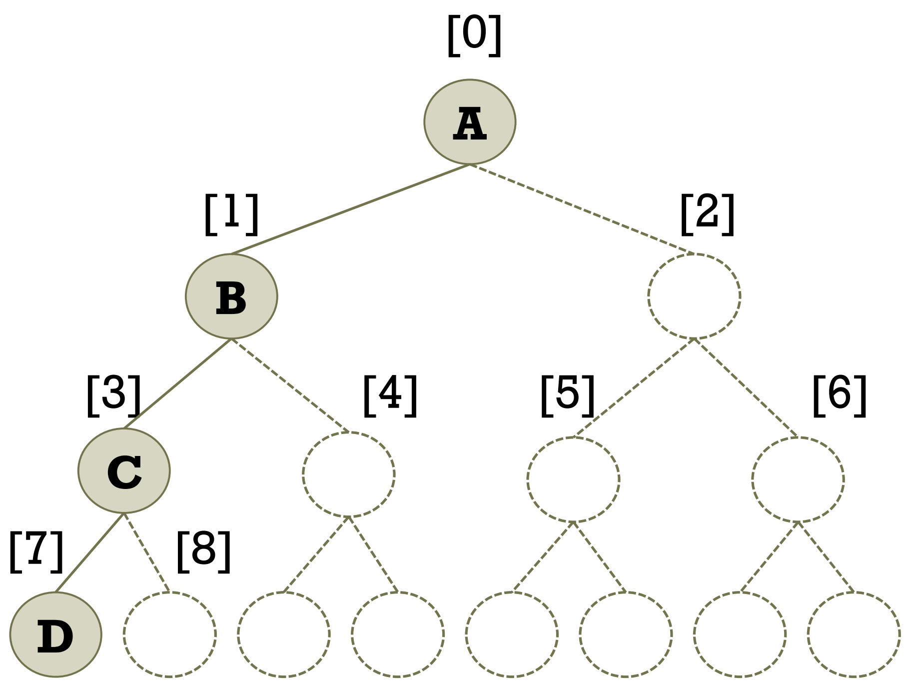
- Example of array representation



Index	Data
[0]	A
[1]	B
[2]	C
[3]	D
[4]	
[5]	
[6]	E
[7]	
[8]	F

Array Representation

- Example of array representation

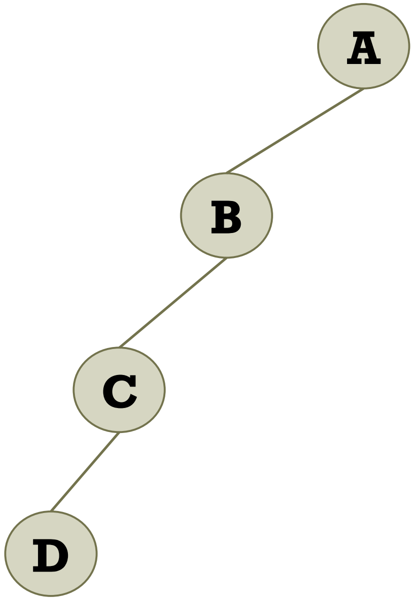


Index	Data
[0]	A
[1]	B
[2]	
[3]	C
[4]	
[5]	
[6]	
[7]	D
[8]	

Cons/Pros of Array Representation

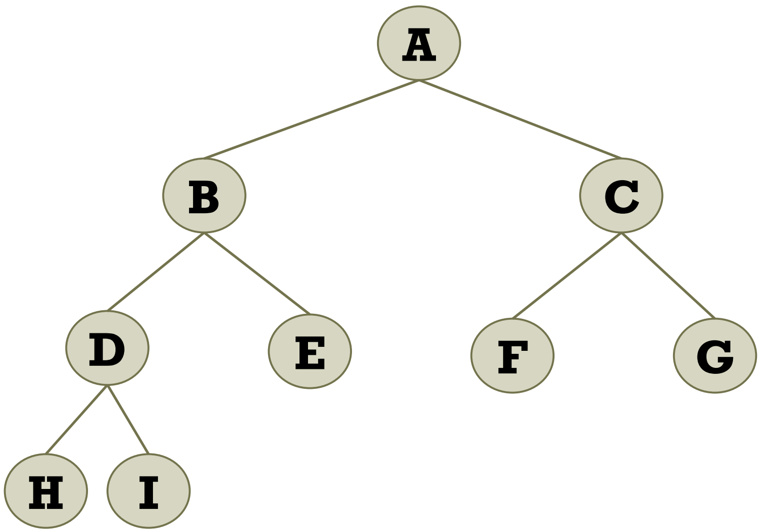
- Pros
  - Easy to implement a binary tree
  - Ideal for representing complete binary tree

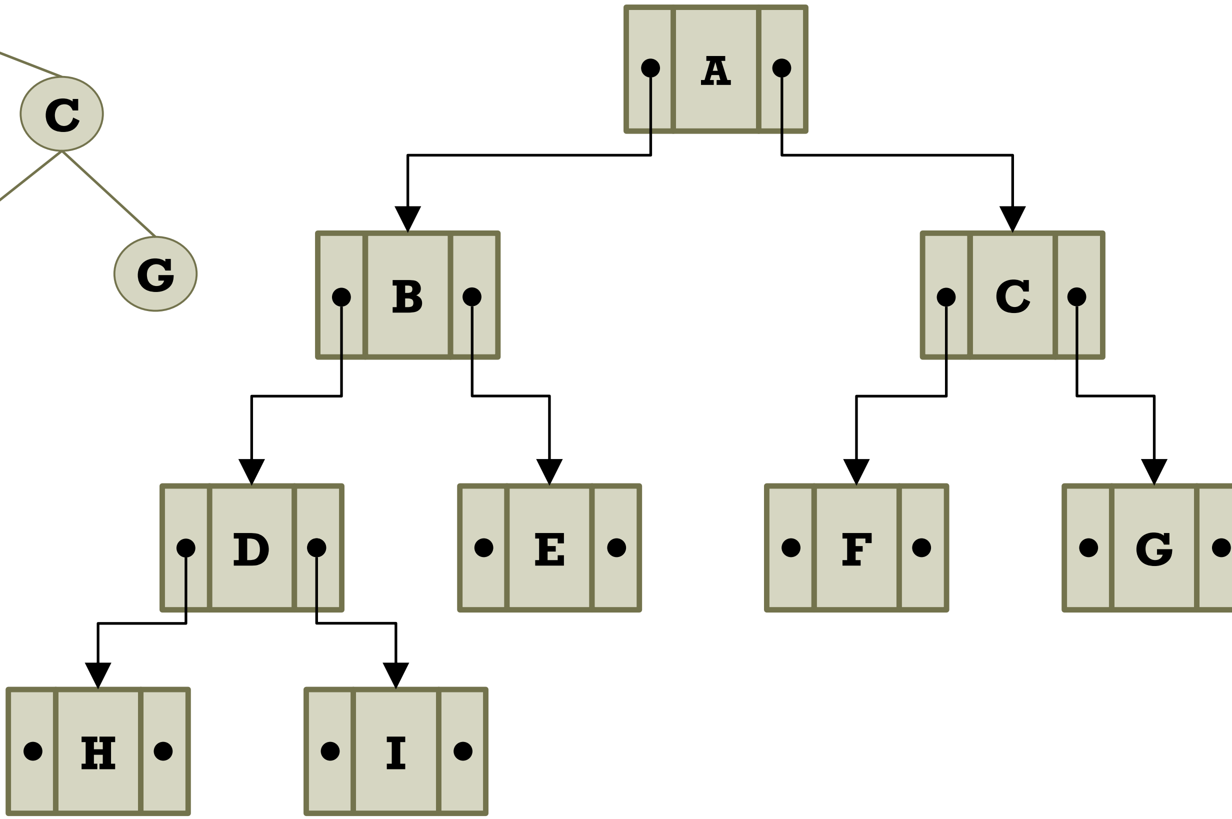
- Cons
  - Inefficient for arbitrary binary tree
    - E.g., skewed binary tree
  - Difficult to update insertions/deletions



Linked List Representation

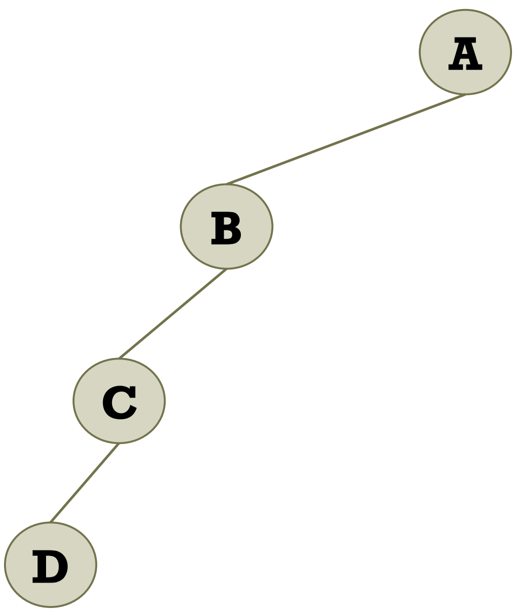
- How to represent a binary tree by linked list

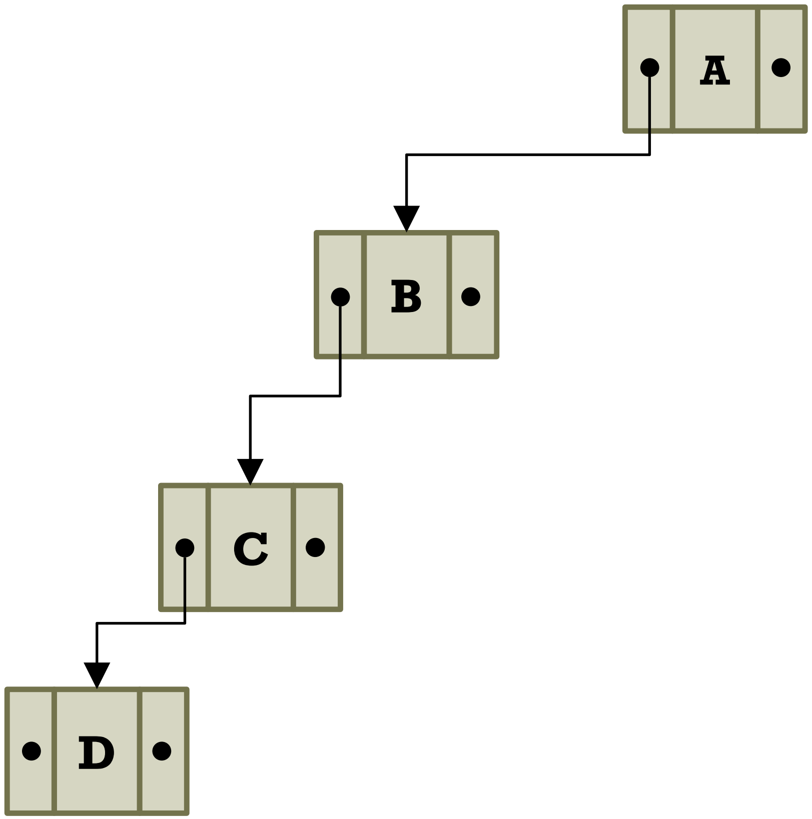




Linked List Representation

- How to represent a binary tree by linked list
  - The skewed binary tree does not incur unnecessary space overhead.





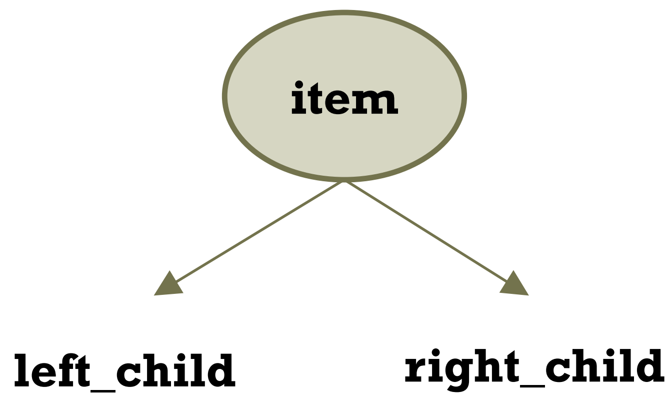
Linked List Representation

- Node representation in a binary tree

left_child	***item***	***right_child***

```
typedef int BData;

typedef struct _bTreeNode
{
    BData item;
    struct _bTreeNode * left_child;
    struct _bTreeNode * right_child;
} BTreeNode;
```



How to Build a Binary Tree

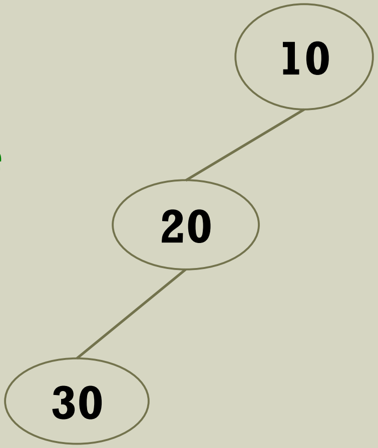
```c
int main()
{
    BTreeNode * n1 = (BTreeNode *)malloc(sizeof(BTreeNode));
    BTreeNode * n2 = (BTreeNode *)malloc(sizeof(BTreeNode));
    BTreeNode * n3 = (BTreeNode *)malloc(sizeof(BTreeNode));

    n1->item = 10;     // Setting the first node
    n1->left_child = n2;
    n1->right_child = NULL;

    n2->item = 20;     // Setting the second node
    n2->left_child = n3;
    n2->right_child = NULL;

    n3->item = 30;     // Setting the third node
    n3->left_child = NULL;
    n3->right_child = NULL;

    free(n1), free(n2), free(n3);
    return 0;
}
```



Binary Tree Implementation

- Operations

```
// Create a new node.
BTreeNode * CreateNode(BData item);
// Destroy a node.
void DestroyNode(BTreeNode * node);

// Conect the root to a left-side node.
void CreateLeftSubtree(BTreeNode* root, BTreeNode * left);
// Conect the root to a right-side node.
void CreateRightSubtree(BTreeNode* root, BTreeNode * right);

// Traverse a tree.
void Inorder(BTreeNode* root);
void Preorder(BTreeNode* root);
void Postorder(BTreeNode* root);
void Levelorder(BTreeNode* root);
```

Binary Tree Implementation

- CreateNode and DestroyNode operations

```
// Create a new node.
BTreeNode * CreateNode(BData item)
{
    BTreeNode * node = (BTreeNode*)malloc(sizeof(BTreeNode));
    node->item = item;
    node->left_child = NULL;
    node->right_child = NULL;

    return node;
}

// Destroy a node.
void DestroyNode(BTreeNode * node)
{
    free(node);
}
```

Binary Tree Implementation

- CreateLeftSubtree and CreaterightSubtree operations

```
// Conect the root to a left-side node.
void CreateLeftSubtree(BTreeNode* root, BTreeNode * left)
{
    if (root->left_child != NULL)
        exit(1);

    root->left_child = left;
}

// Conect the root to a right-side node.
void CreateRightSubtree(BTreeNode* root, BTreeNode * right)
{
    if (root->right_child != NULL)
        exit(1);

    root->right_child = right;
}
```

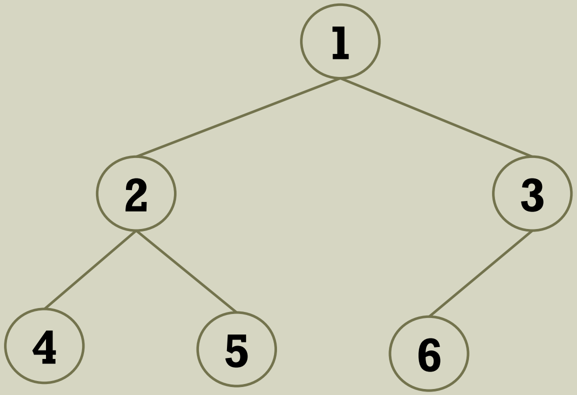
Binary Tree Implementation

```
int main()
{
     BTreeNode * node1 = CreateNode(1);
     BTreeNode * node2 = CreateNode(2);
     BTreeNode * node3 = CreateNode(3);
     BTreeNode * node4 = CreateNode(4);
     BTreeNode * node5 = CreateNode(5);
     BTreeNode * node6 = CreateNode(6);

     CreateLeftSubtree(node1, node2);
     CreateRightSubtree(node1, node3);

     CreateLeftSubtree(node2, node4);
     CreateRightSubtree(node2, node5);
     CreateLeftSubtree(node3, node6);

     DestroyNode(node1);
     DestroyNode(node2);
     DestroyNode(node3);
     DestroyNode(node4);
     DestroyNode(node5);
     DestroyNode(node6);
     return 0;
}
```



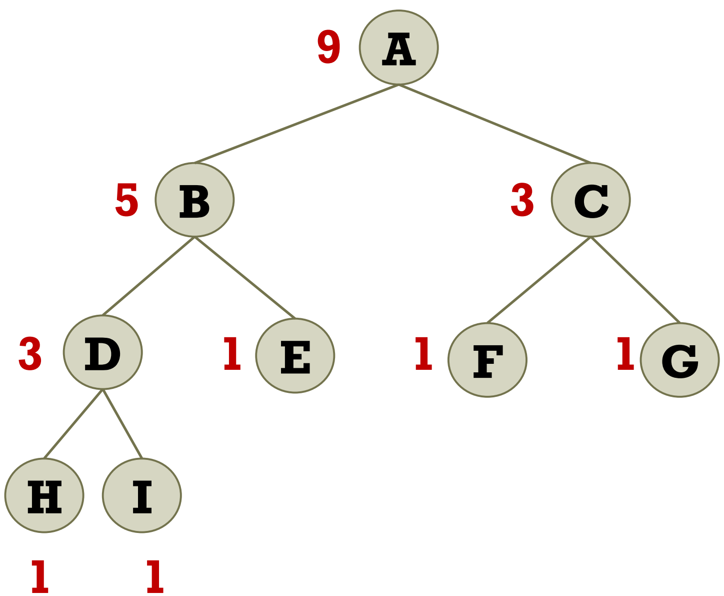
Total Number of Nodes

- How to count the number of nodes of a binary tree
  - Recursive definition

$$N(T) = \begin{cases} 1 & \mathit{if\ T\ is\ a\ leaf\ node} \\ N(T.right), +N(T.left)) + 1 & \mathit{Otherwise} \end{cases}$$

```
int Nodes(BTreeNode *node)
{
    int r = 0, l = 0;
    if (node->right_child != NULL)
        r = Nodes(node->right_child);
    if (node->left_child != NULL)
        l = Nodes(node->left_child);

    return 1 + r + l;
}
```



The Height of Binary Tree

- How to calculate the height of a binary tree
  - Recursive definition

$$H(T) = \begin{cases} 1 & \textit{if } T \textit{ is a leaf node} \\ \max(H(T.right), H(T.left)) + 1 & \textit{Otherwise} \end{cases}$$

```
int Height(BTreeNode *node)
{
    int r = 0, l = 0;
    if (node->right_child != NULL)
        r = Height(node->right_child);
    if (node->left_child != NULL)
        l = Height(node->left_child);

    return 1 + max(r, l);
}
```