CSE221

Lecture 8: Trees

Hyungon Moon

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided by Prof. Won-Ki Jeong.

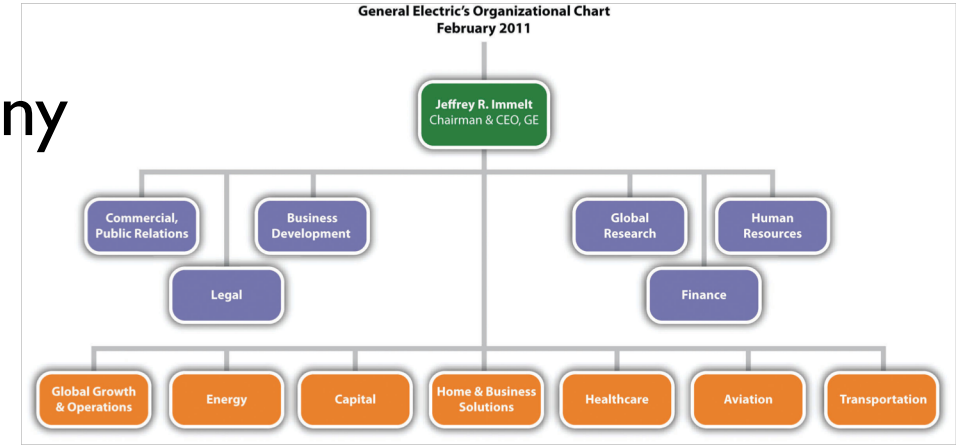
UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Outline

- Tree representation
- Binary trees

Linear Lists and Trees

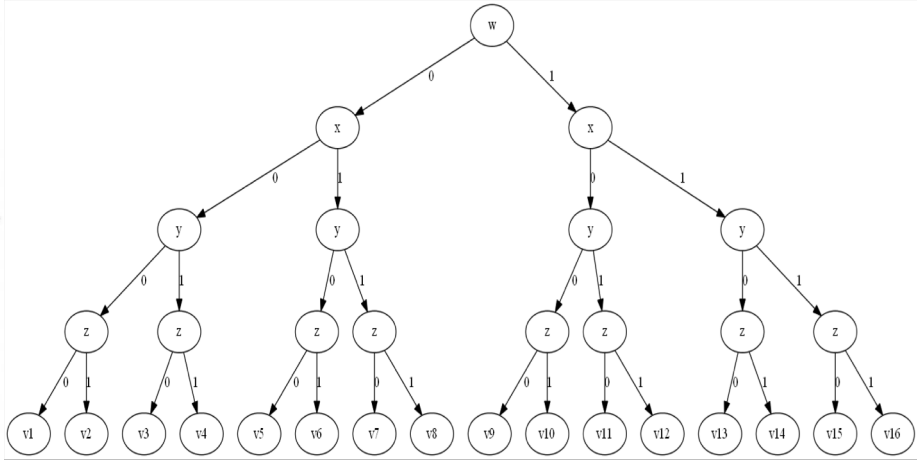
- Linear lists are useful for serially ordered data
  - (a,b,c,d,e)
  - Days of week
  - Students in a class
- Trees are useful for nonlinear (hierarchically) ordered data
  - Structure of a company
  - Pedigree



Trees

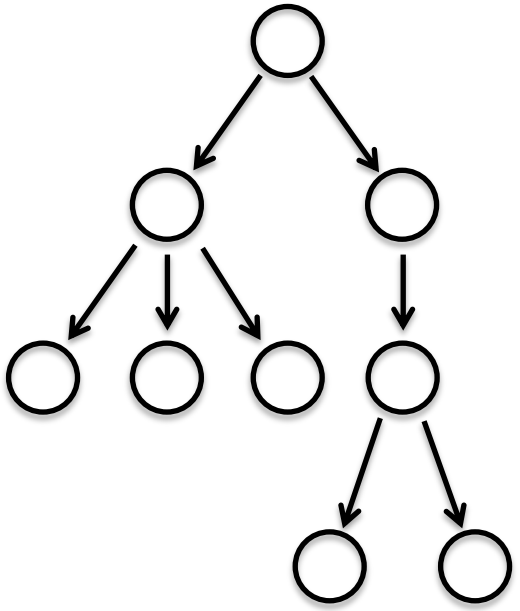
- Definition
  - Finite set T of nodes storing elements in a parent-child relationship
    - If T is nonempty, there is a specially designated node, the root, that has no parent
    - Each node v in T has a unique parent w; every node with parent w is a child of w
  - Either empty or consists of a node r, called the root of T, and a (possibly empty) set of trees whose roots are the children of r (recursive def.)
- By definition, a tree can be empty

Trees


w
x
x
y
y
y
y
z
z
z
z
z
z
z
z
v1
v2
v3
v4
v5
v6
v7
v8
v9
v10
v11
v12
v13
v14
v15
v16

Terminology

- Node
- Degree of a node X
  - # of children of a node X
- Internal nodes
  - Node with at least one child
  - Degree > 0
- Leaf (external) nodes
  - Node with no child
  - Degree 0 nodes



Terminology

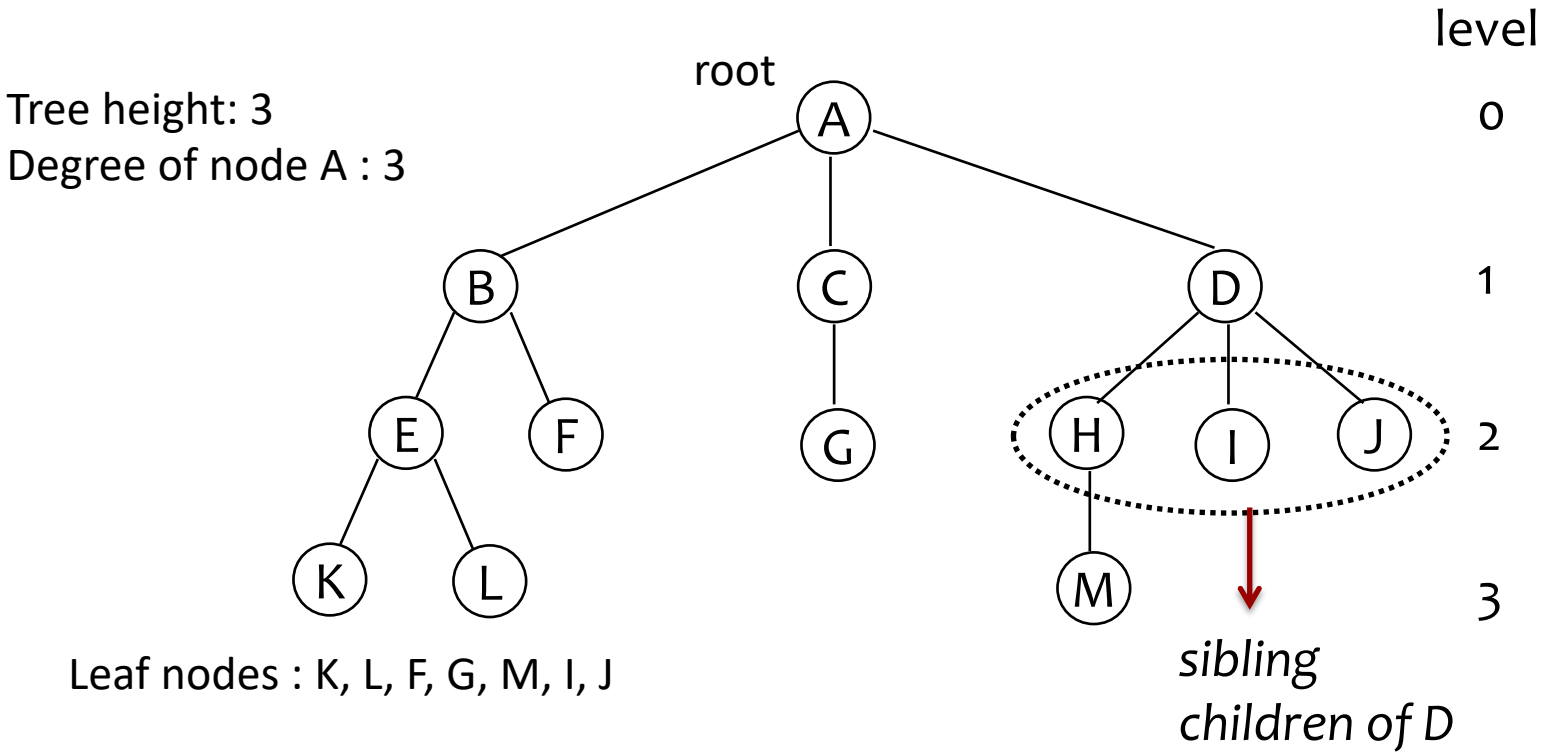
- Level (depth) of a node
  - Root node : 0
  - Child of a node whose level is n : n+1
- Degree of a tree
  - Max degree of nodes in the tree
- Height of a tree
  - Maximum level of nodes in the tree

Terminology

- Subtree of X
  - Tree whose root is one of the children of X
- Ancestor of node X
  - All nodes in the paths from X to the root
- Descendants of node X
  - All nodes in the subtrees of node X

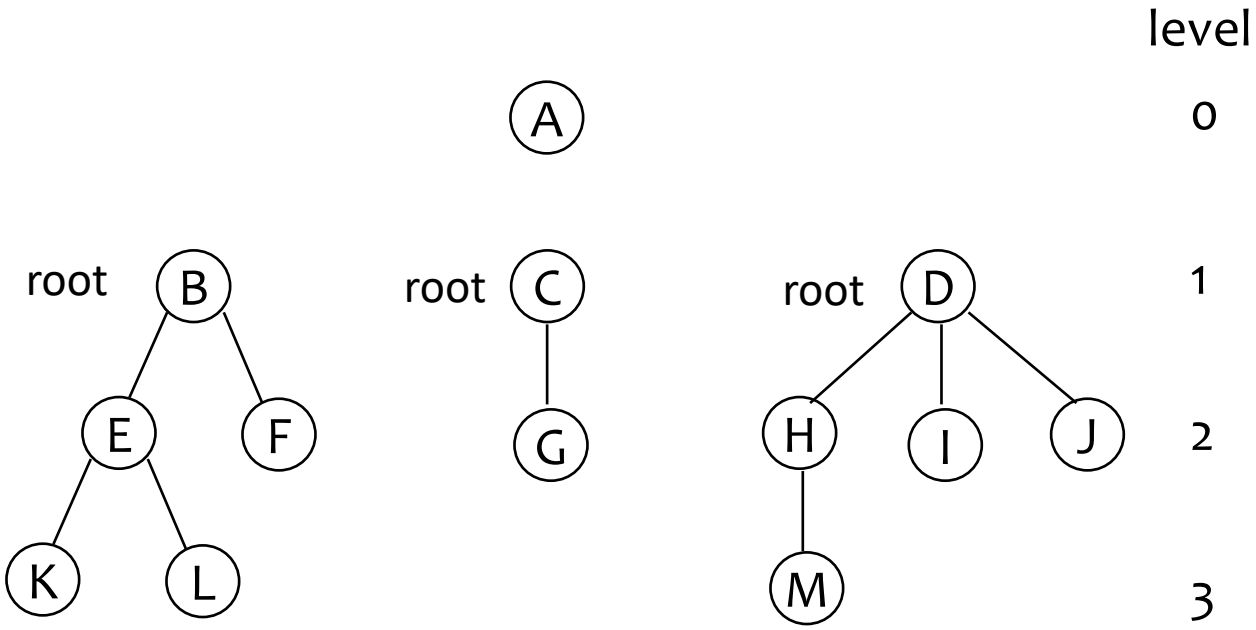
Example: Tree Terminology

- Ancestor of M = {H,D,A}
- Descendant of D = {H,I,J,M}



Example: Tree Terminology

- Ancestor of M = {H,D,A}
- Descendant of D = {H,I,J,M}



Tree ADT

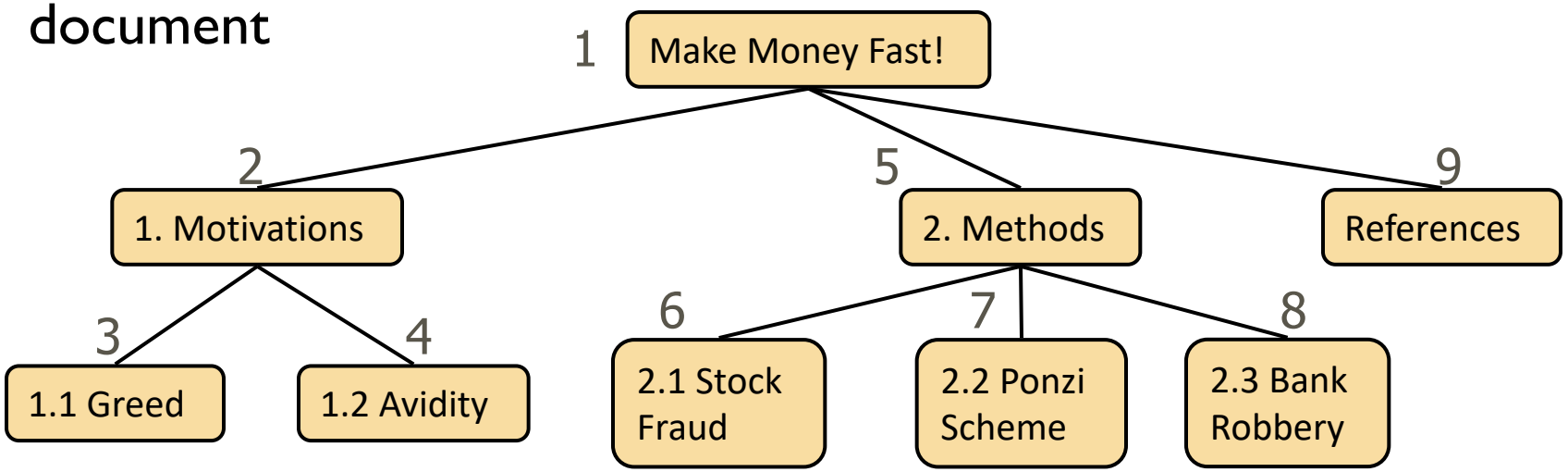
- We use positions to abstract nodes
- Generic methods:
  - integer size()
  - boolean empty()
- Accessor methods:
  - position root()
  - list<position> positions()
- Position-based methods:
  - position p.parent()
  - list<position> p.children()

- Query methods:
  - boolean p.isRoot()
  - boolean p.isExternal()
- Additional update methods may be defined by data structures implementing the Tree ADT

Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

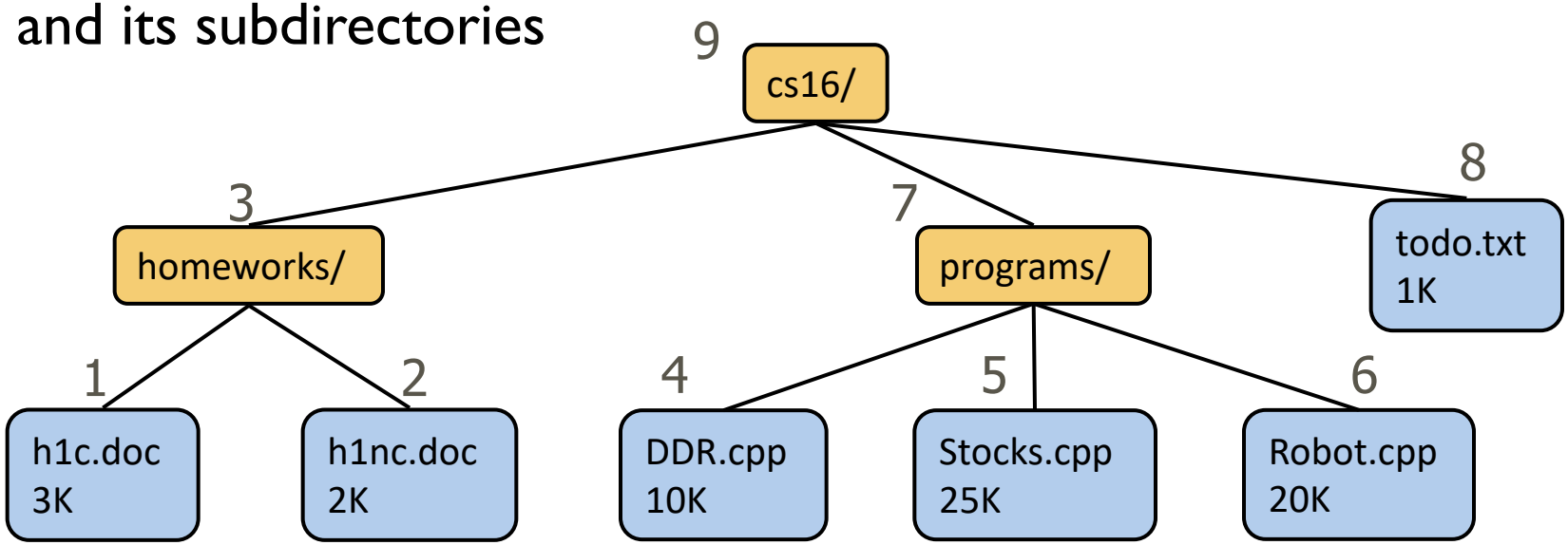
Algorithm *preOrder(v)*
    visit(v)
    for each child *w* of *v*
        preorder (w)



Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm *postOrder(v)*
  for each child *w* of *v*
    postOrder (w)
  visit(v)



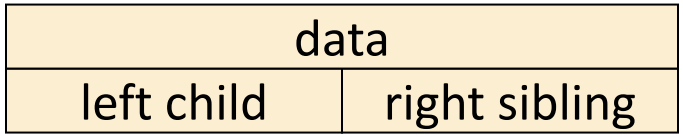
Tree Representation

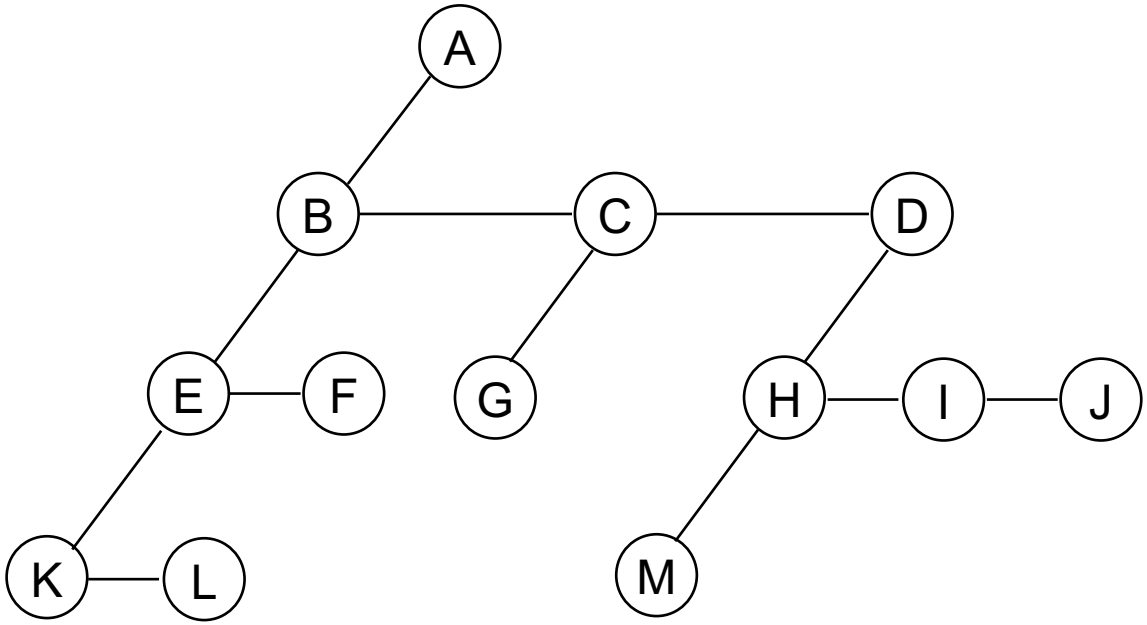
- Node structure for a tree of degree k

- Total # of zero child pointers : n(k-1)+1
  - Total # of nodes : n
  - Total # of child pointers : nk
  - Total # of child pointers used : n-1 (root cannot be pointed by a child pointer)
  - Total # of zero child pointers : nk–(n-1) = n(k-1)+1

Tree Representation

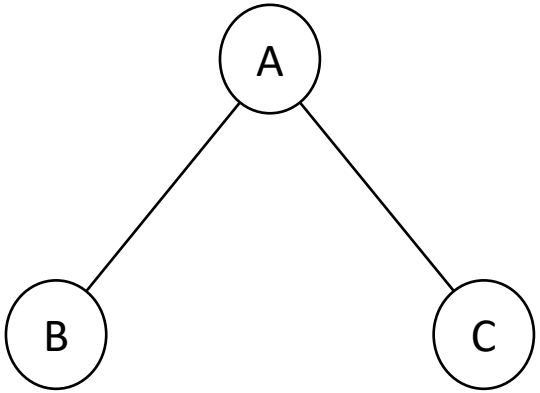
- Left child - right sibling representation





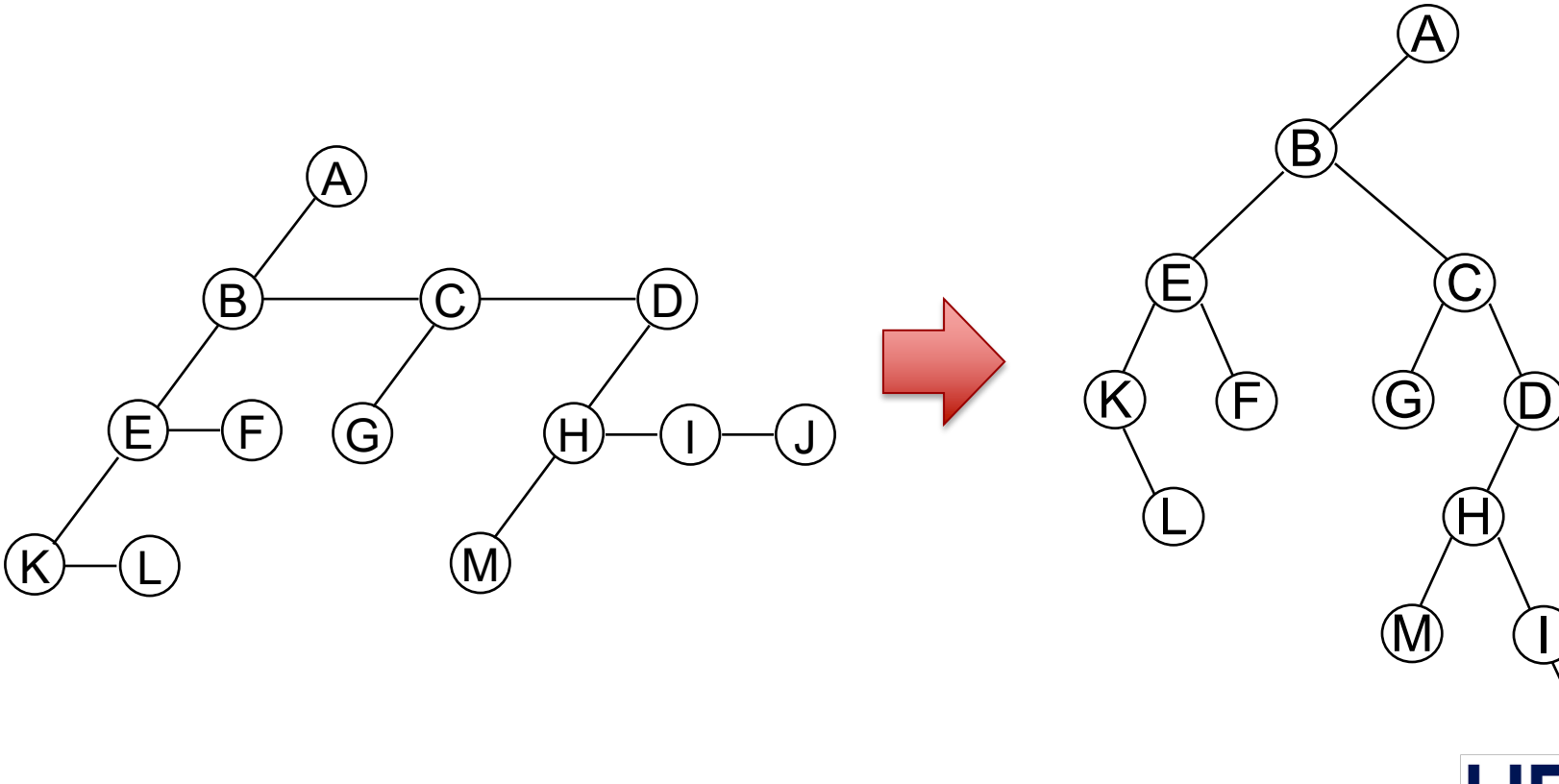
Tree Representation

- Binary tree
  - k=2
  - Left child – right child



Tree Representation

- Convert left child-right sibling to binary tree
  - Rotate right sibling clockwise by 45 degree

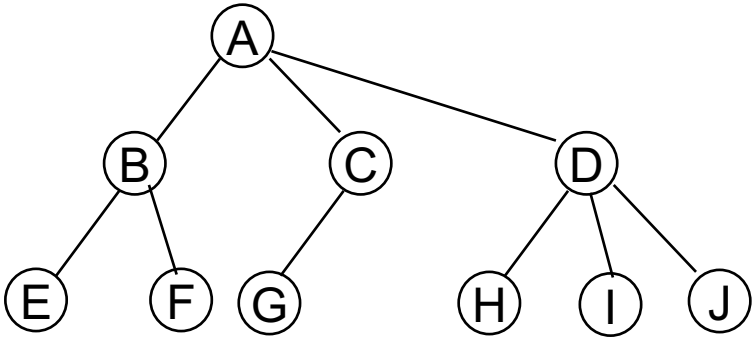


Convert Arbitrary Tree to Binary Tree

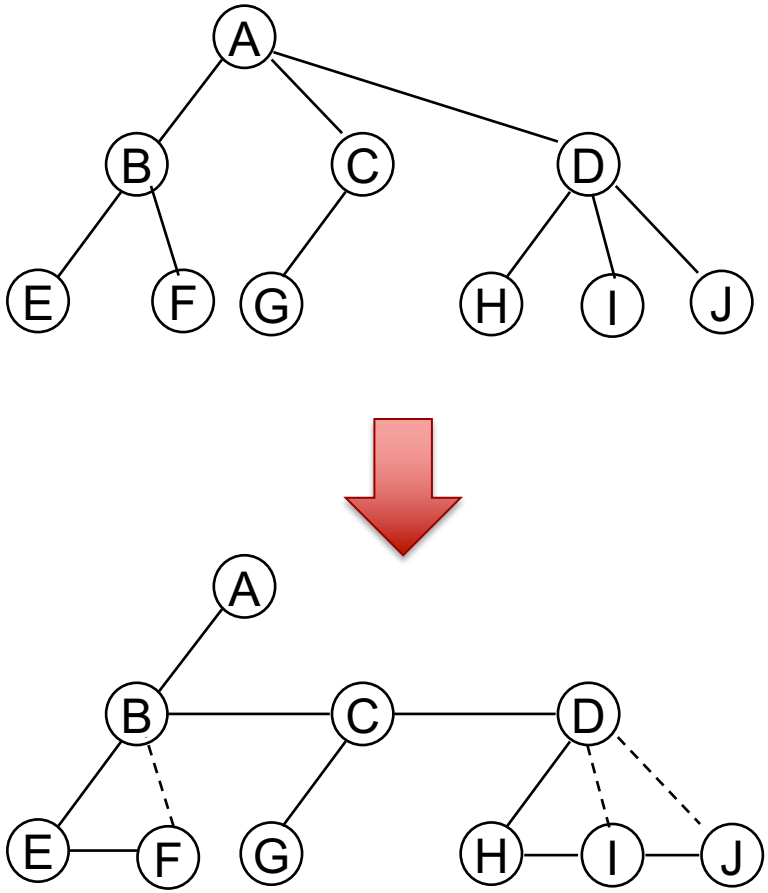
- Connect siblings
- Disconnect siblings and their parent except left-most node
  - Convert k-way tree to left child – right sibling representation
- Rotate 45° clockwise

We can represent any tree as a binary tree!

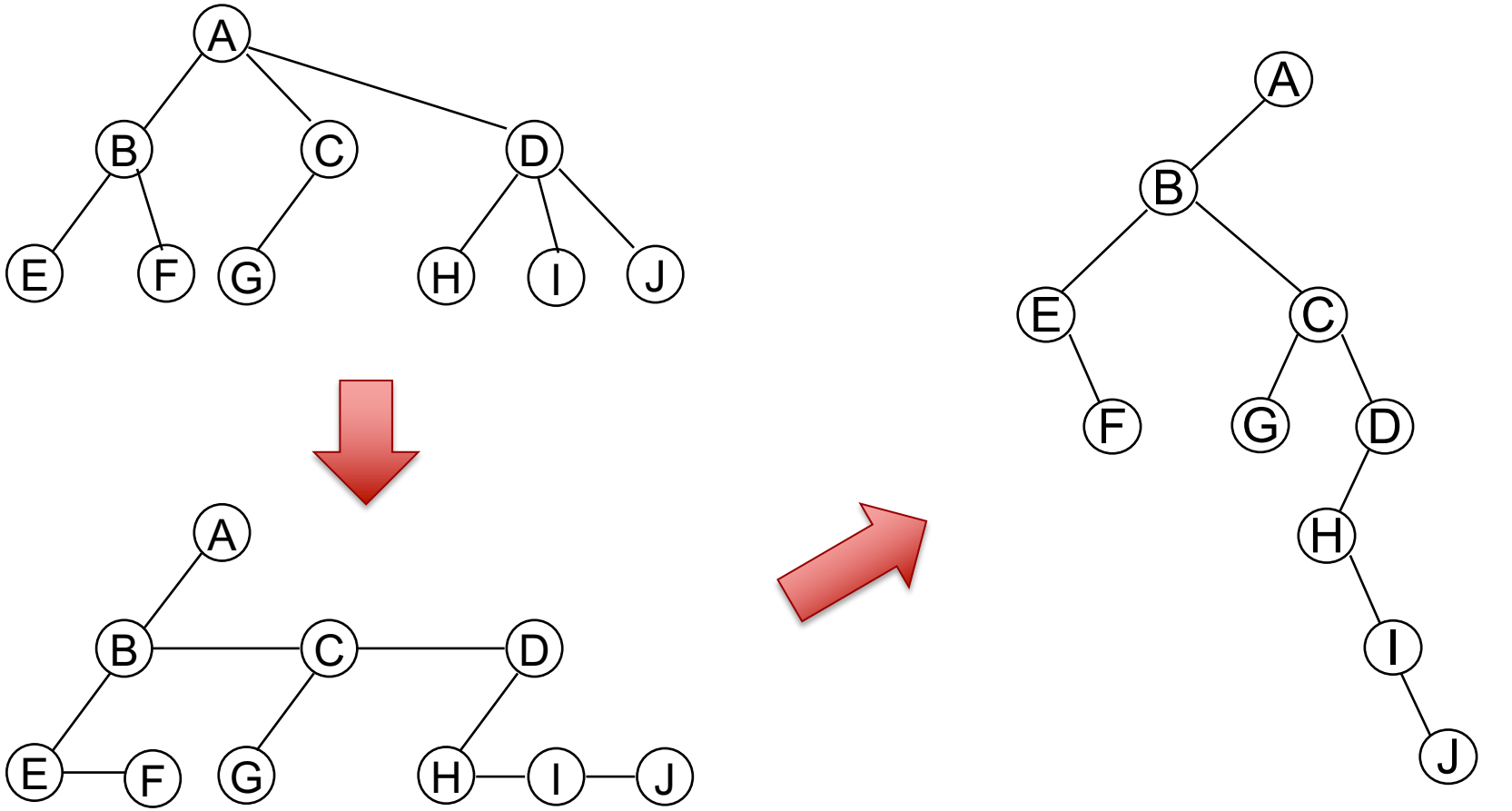
Example



Example

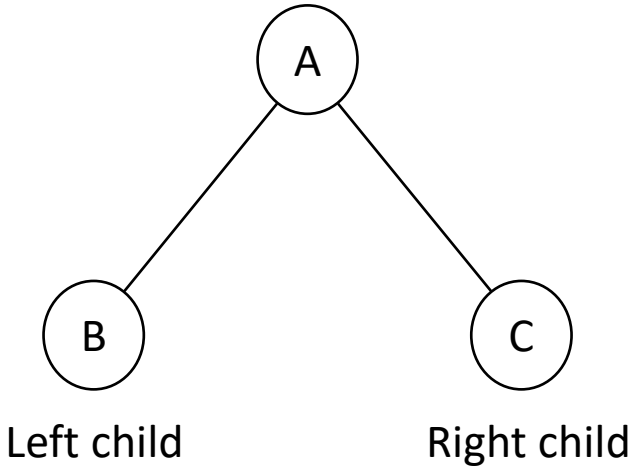


Example



Binary Trees

- Definition
  - Finite set of nodes that either empty or consists of a root and two disjoint binary trees called the left subtree and the right subtree.

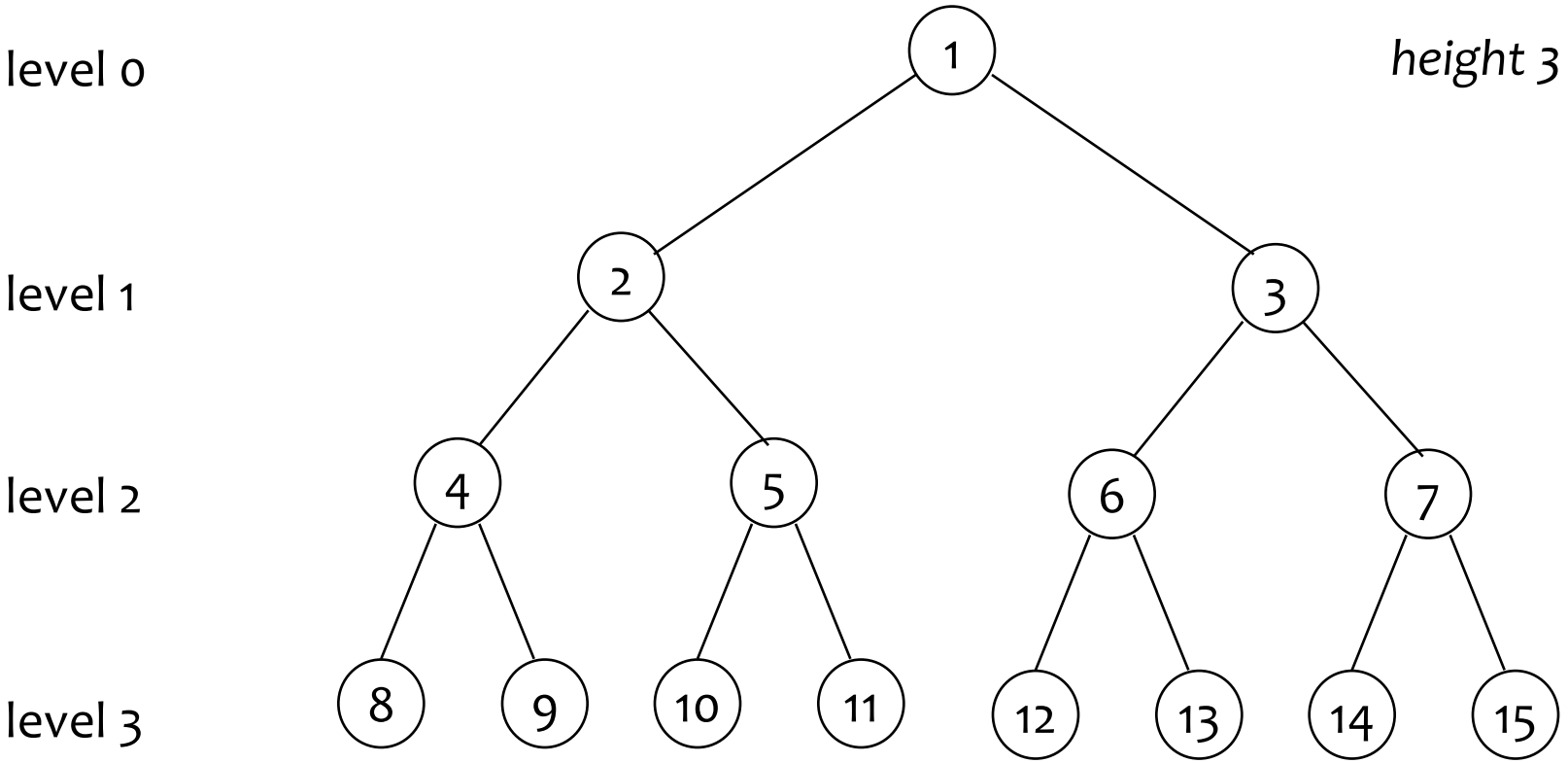


Properties of Binary Trees

- Maximum # of nodes on level i
  - 2^i, i>=0
- Maximum # of nodes in a binary tree of height k
  - $2^{k+1}-1$
- For nonempty binary tree, n_0 is the # of leaf nodes and n_2 is the number of degree 2
  - $n_0=n_2+1$

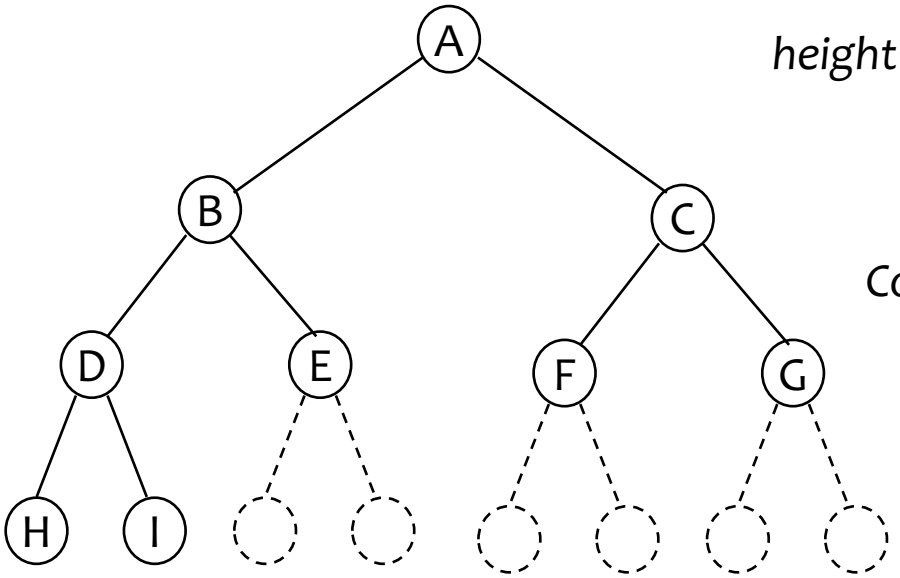
Full Binary Tree

- For height k, the tree has $2^{k+1}-1$ nodes
  - Numbering: left to right, top to bottom



Complete Binary Tree

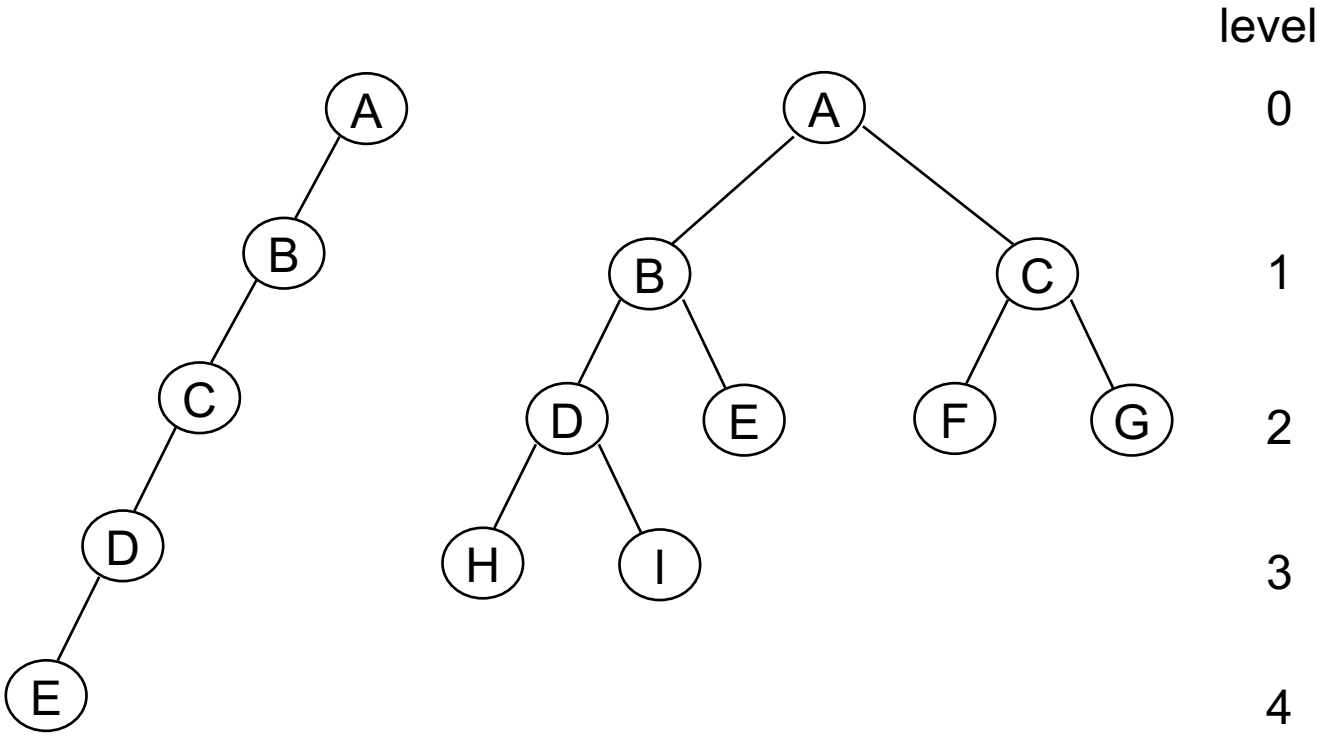
- Nodes corresponds to 1~n nodes in the full binary tree of depth k
- Height of a complete binary tree (n nodes)
  - $floor(\log_2(n))$



height 3

Complete Binary tree

Binary Trees

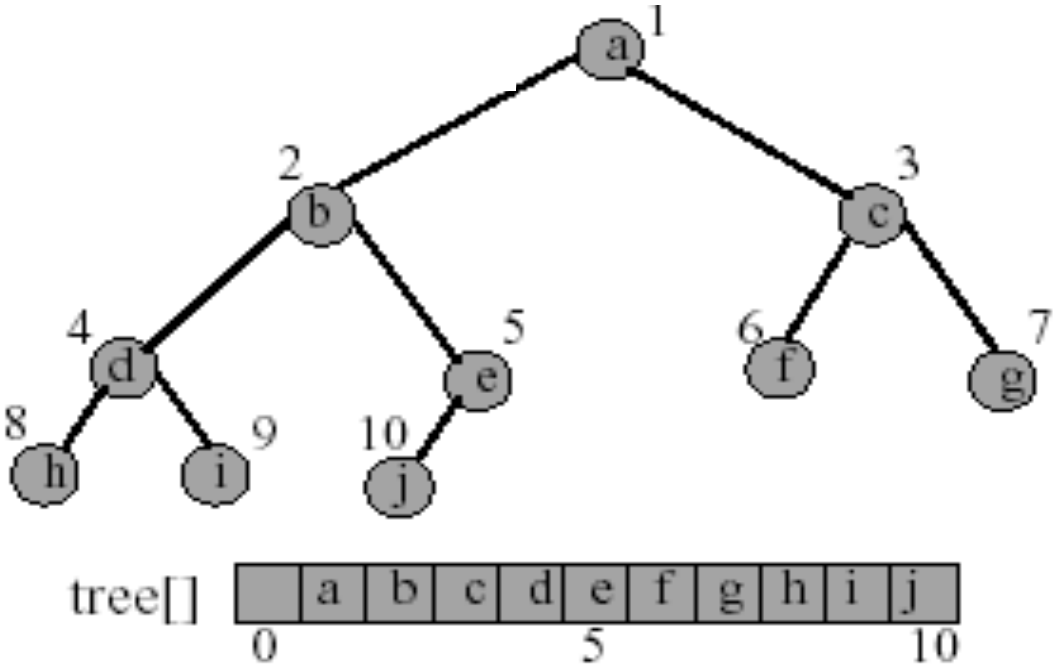


(a) Skewed

(b) Complete

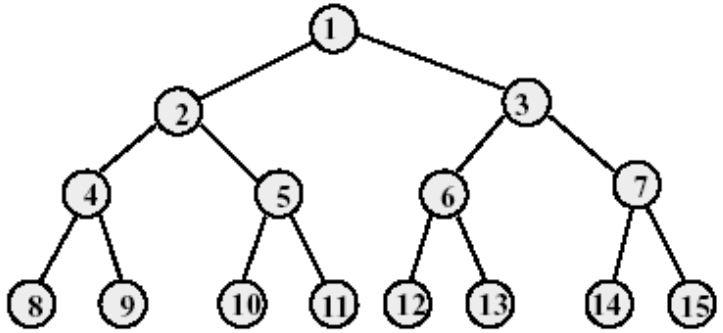
Binary Tree Representation

- Array representation
  - Each node is stored at the array position corresponding to the number assigned to it



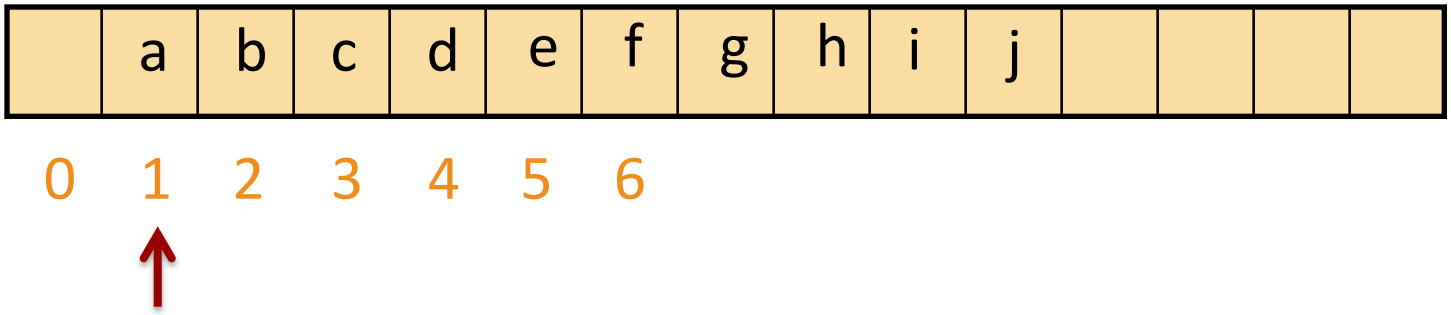
Binary Tree Representation using Array

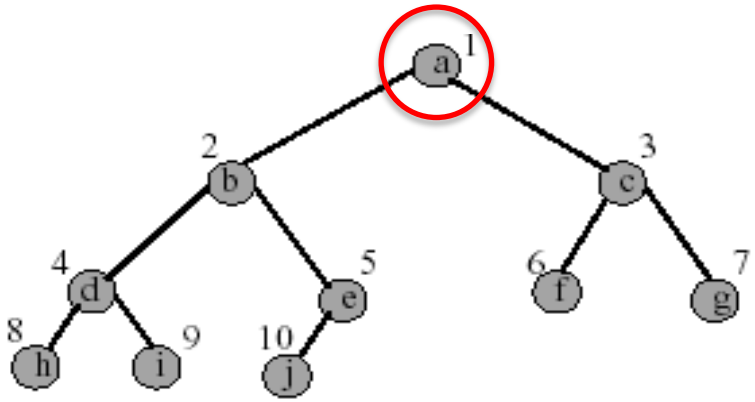
- If a complete binary tree with n nodes is represented sequentially
  - parent(i) is at $\lfloor i/2 \rfloor$ if i != 1 (i=1 then root and no parent exists)
  - leftChild(i) is 2i if 2i <= n
  - rightChild(i) is 2i+1 if 2i+1 <= n
- Example
  - leftChild(5) = 10
  - parent(7) = $\lfloor 3.5 \rfloor = 3$



Binary Tree Representation using Array

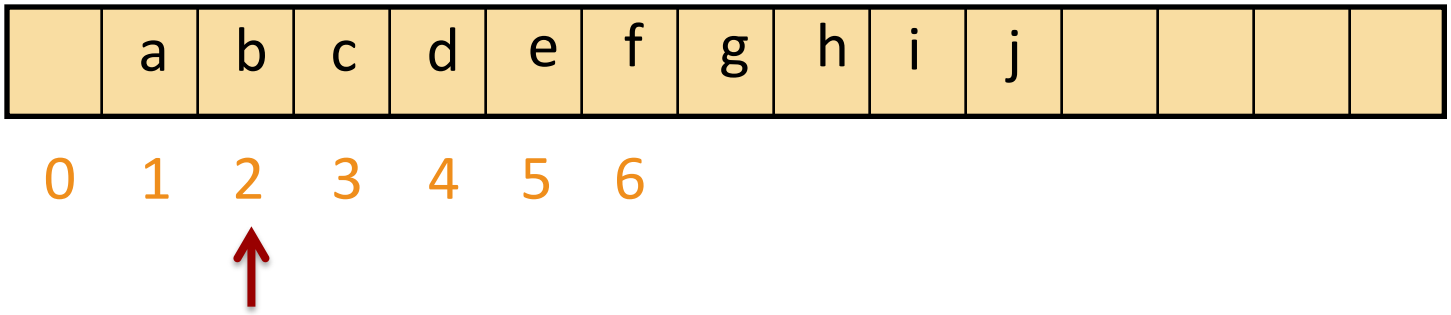
- Traverse array
  - a :T(I)

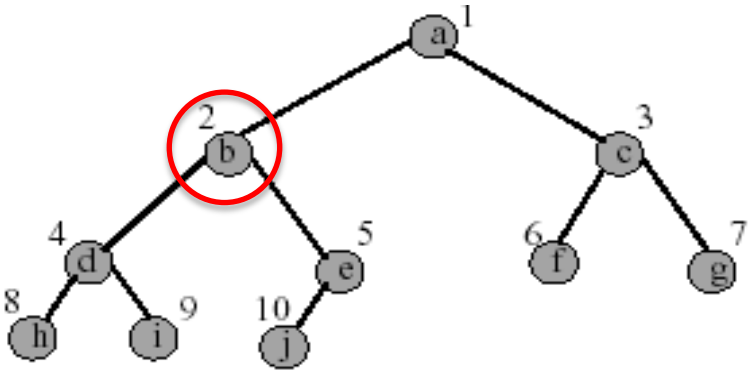




Binary Tree Representation using Array

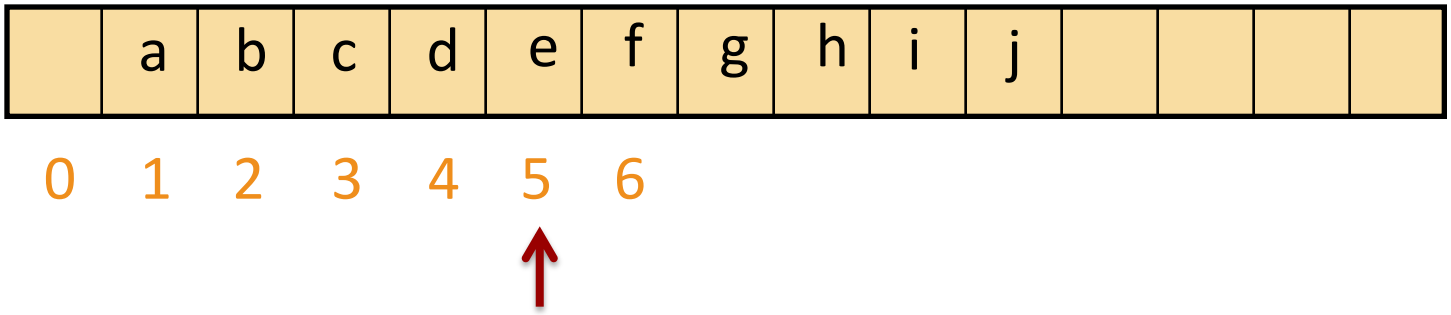
- Traverse array
  - a -> leftChild = T(2*1) = T(2) = b

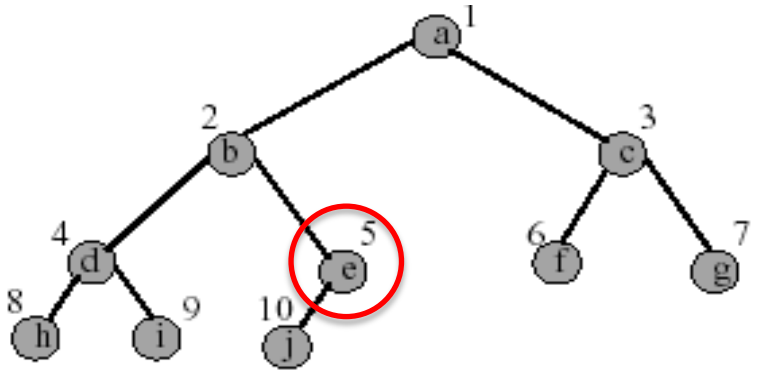




Binary Tree Representation using Array

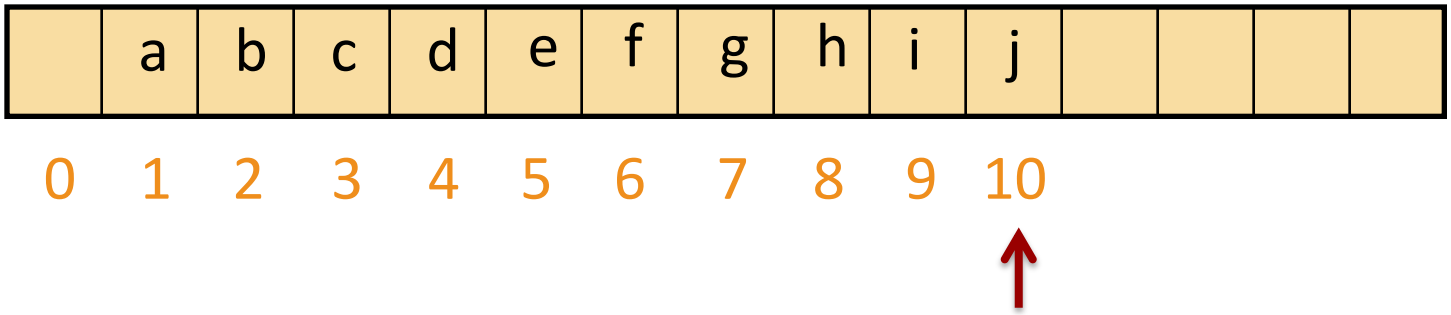
- Traverse array
  - b -> rightChild = T(2*2+1) = T(5) = e

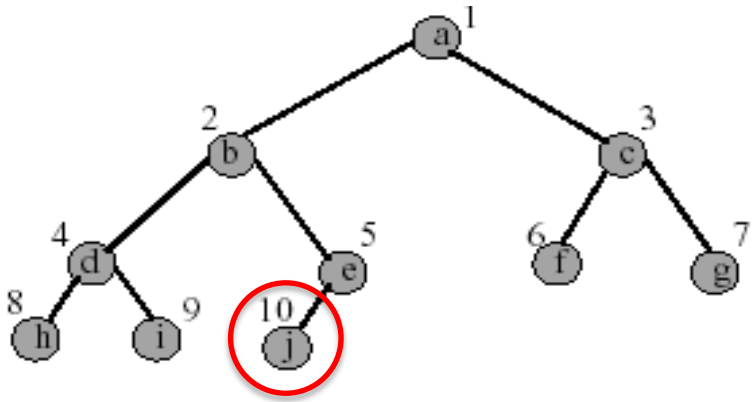




Tree Representation using Array

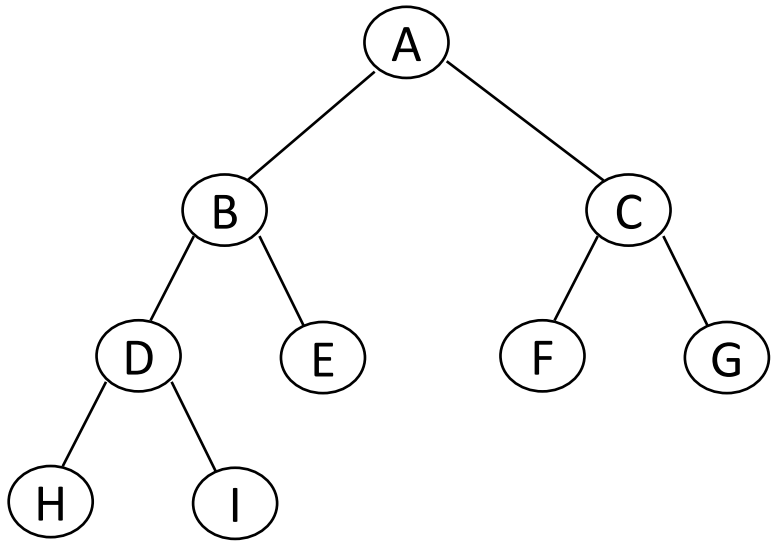
- Traverse array
  - e -> leftChild = T(2*5) = T(10) = j





Binary Tree Representation using Array

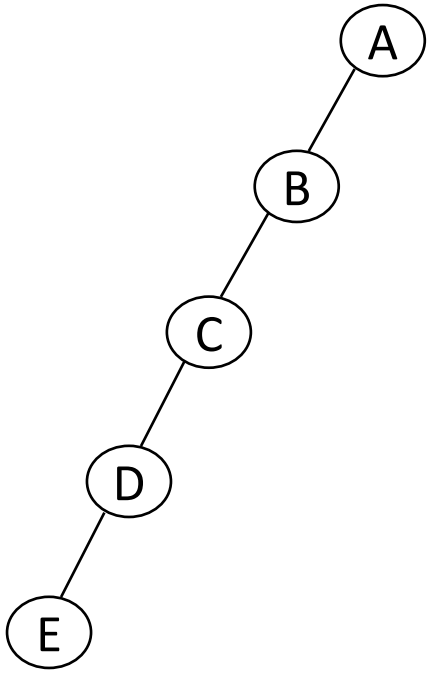
- Space usage
  - Best case : binary tree is complete



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

Binary Tree Representation using Array

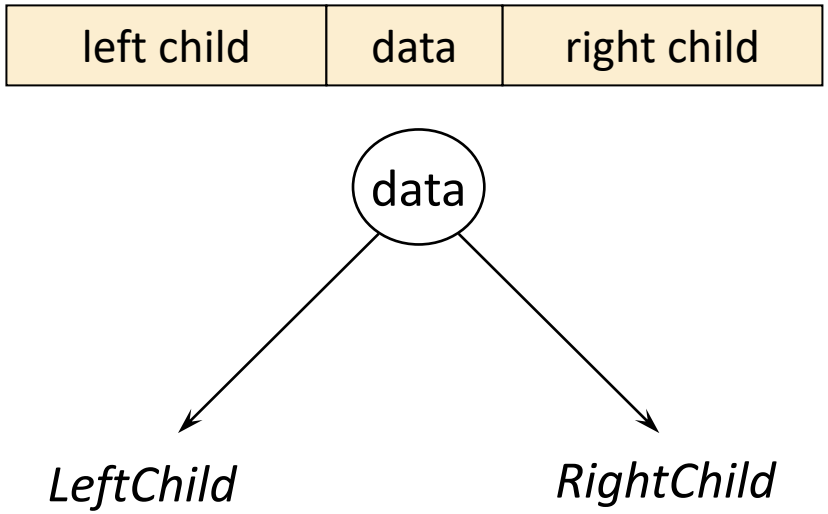
- Space usage
  - Worst case : binary tree is skewed
    - k is used out of 2^k-1 (height k-1)



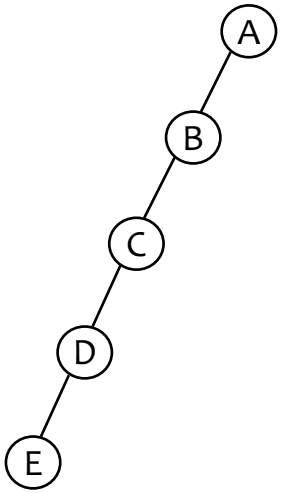
[0]	-
[1]	A
[2]	B
[3]	-
[4]	C
[5]	-
[6]	-
[7]	-
[8]	D
[9]	-
. . .	. . .
[16]	E

Linked Representation of a Tree

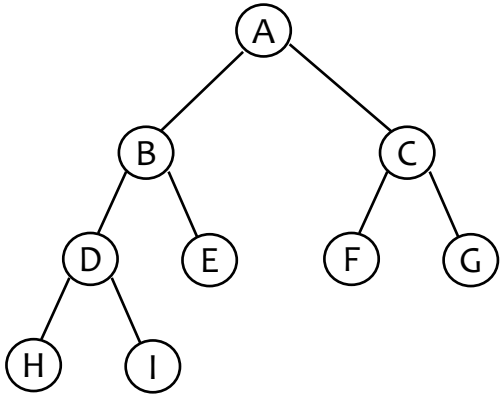
- Problems of array representation
  - Wasting of memory for incomplete binary trees
  - Insert/delete causes shifting elements
- Linked representation



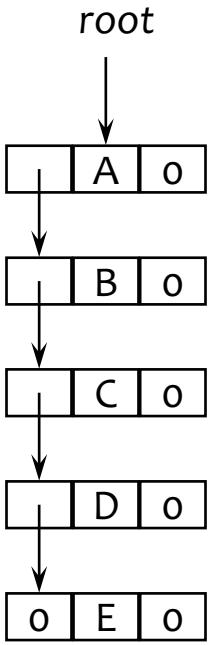

```
class Tree;
class TreeNode{
friend class Tree;
private:
        TreeNode *LeftChild;
        char data;
        TreeNode *RightChild;
};
class Tree {
public:
        // Tree operations
        ...
private:
        TreeNode *root;
};
```

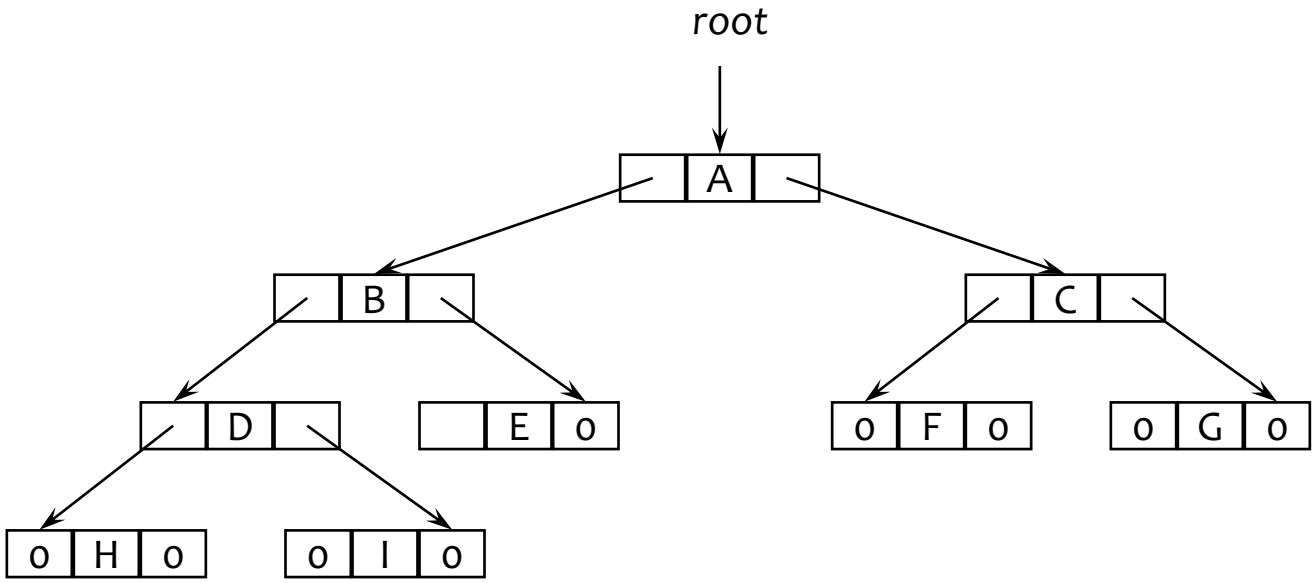


(a)



(b)





Linked Representation of a Tree

- Pros
  - Efficient memory management in most cases
  - Node insert / delete can be fast
- Cons
  - Extra memory usage (storing pointers) for complete binary trees
  - Traversing parent is not possible (need extra pointer per node)

Questions?