CSE221

Trees

Fall 2021

Young-ri Choi

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided in former lectures at UNIST.

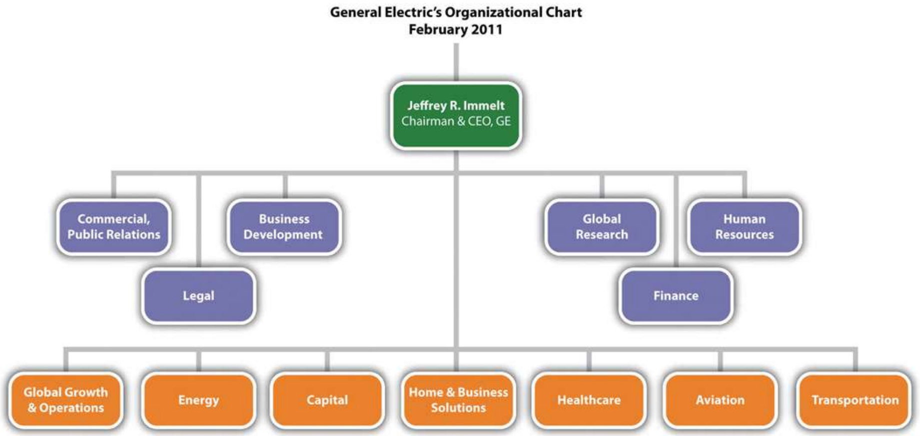
UNIST
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

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
  - Lineage



Trees

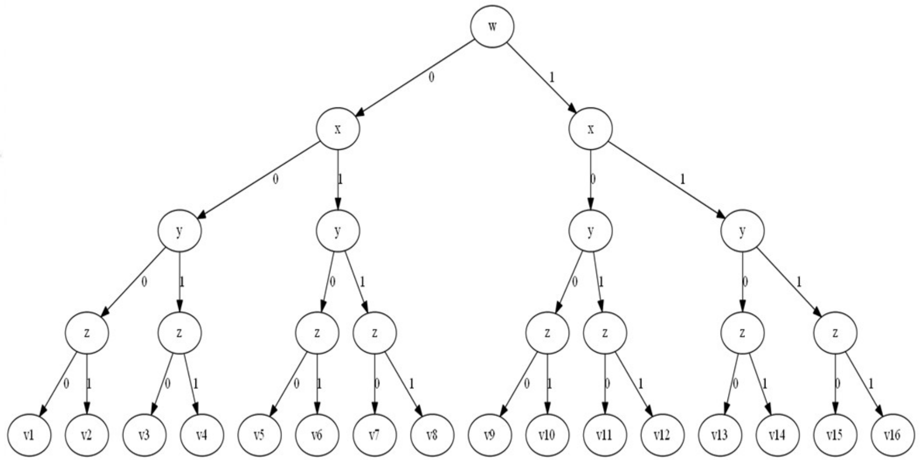
- Definition
  - Finite set T of nodes storing elements in a parent-child relationship
    - If T is nonempty, there is a specially designated node, the root, that has no parent
    - Each node v in T has a <u>unique</u> parent w
    - Every node with parent w is a child of w
- By definition, a tree can be empty

Trees

root

leaf

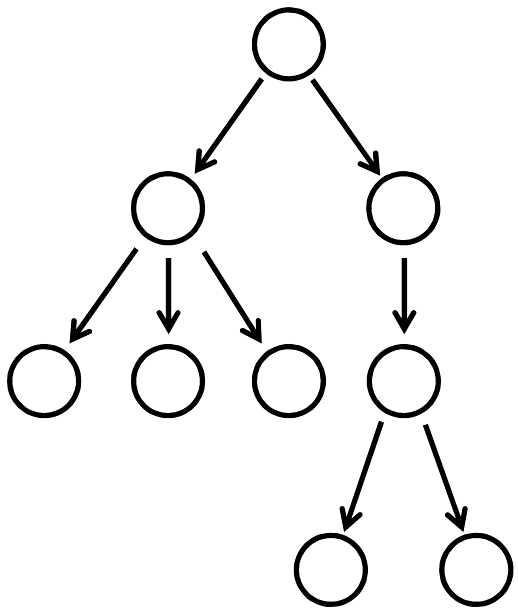
root



leaf

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



ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

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

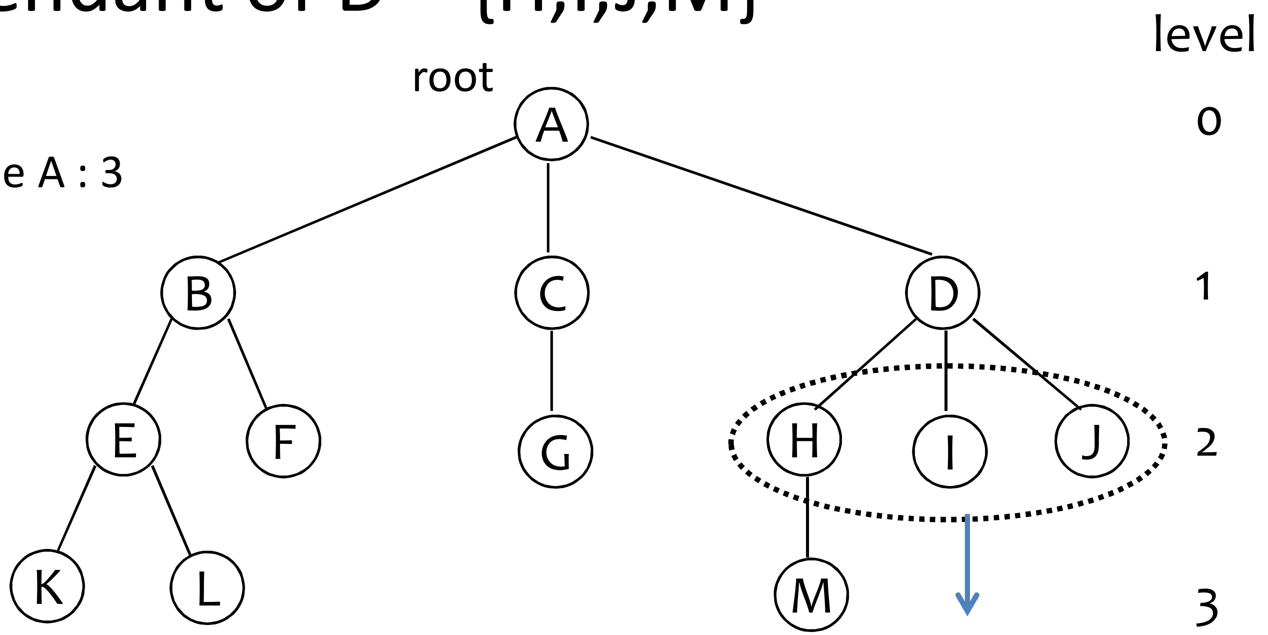
Terminology

- Subtree of X
  - Tree whose root is one of the children of X
- Ancestors 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}

Tree height: 3
Degree of node A : 3

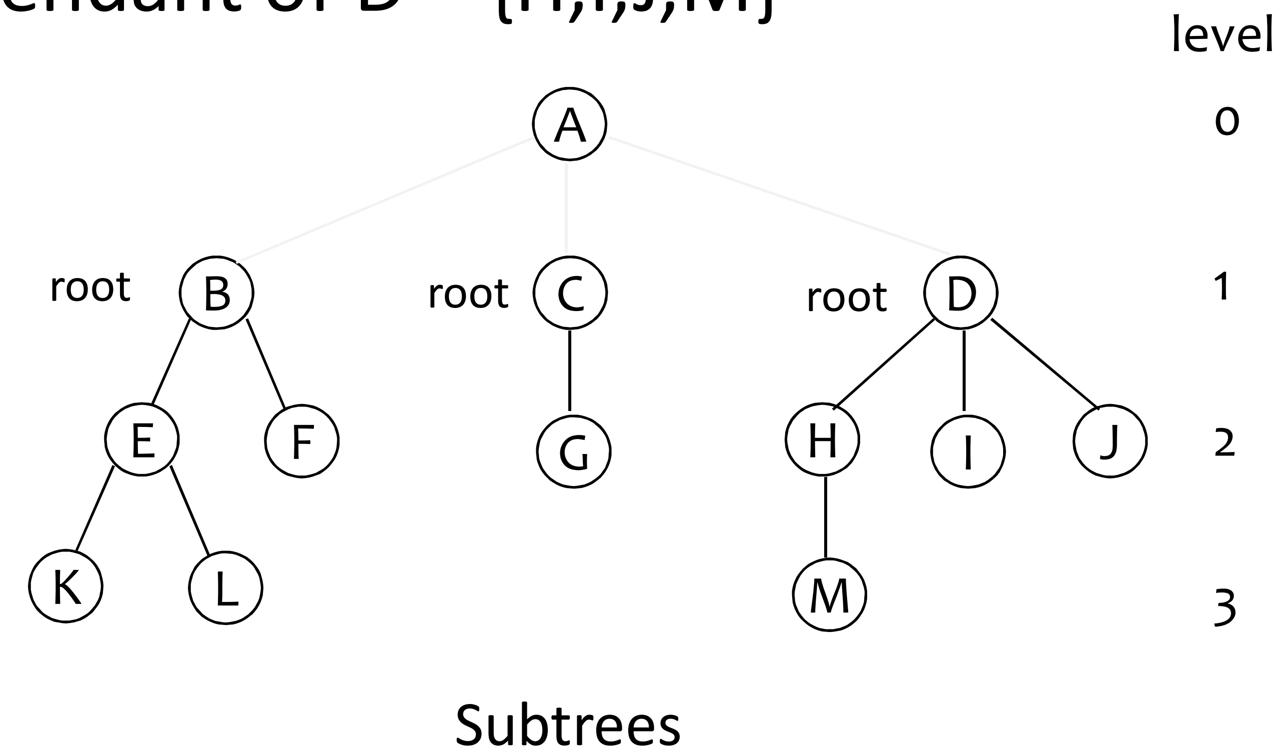


Leaf nodes : K, L, F, G, M, I, J

sibling
children of D

Example: Tree Terminology

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



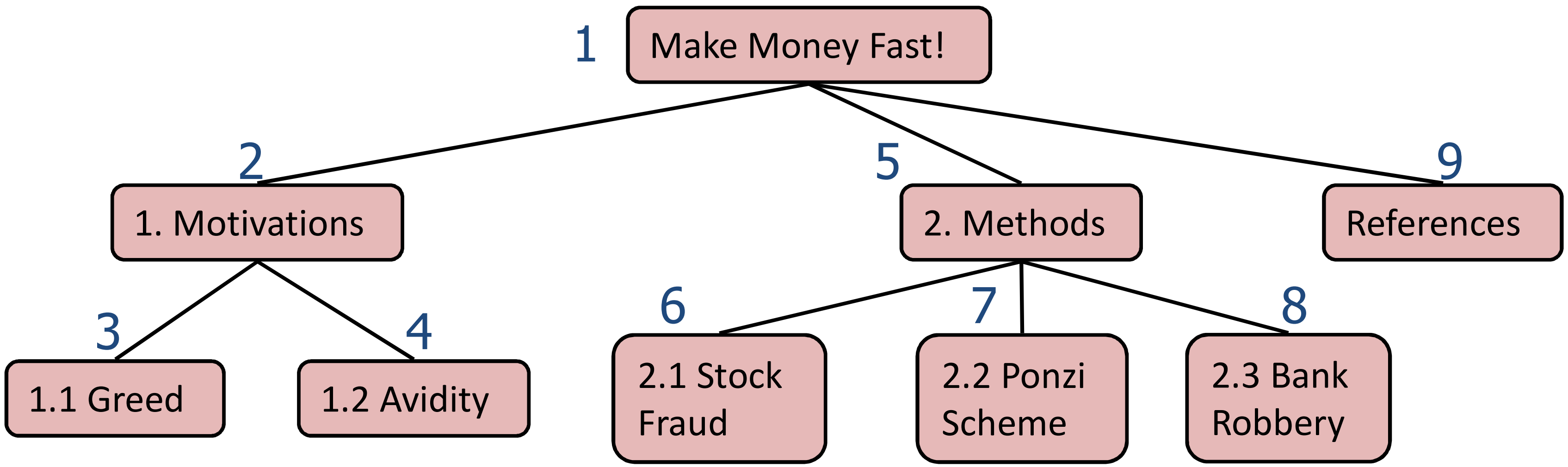
Tree ADT

- 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 isRoot()
  - boolean 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

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

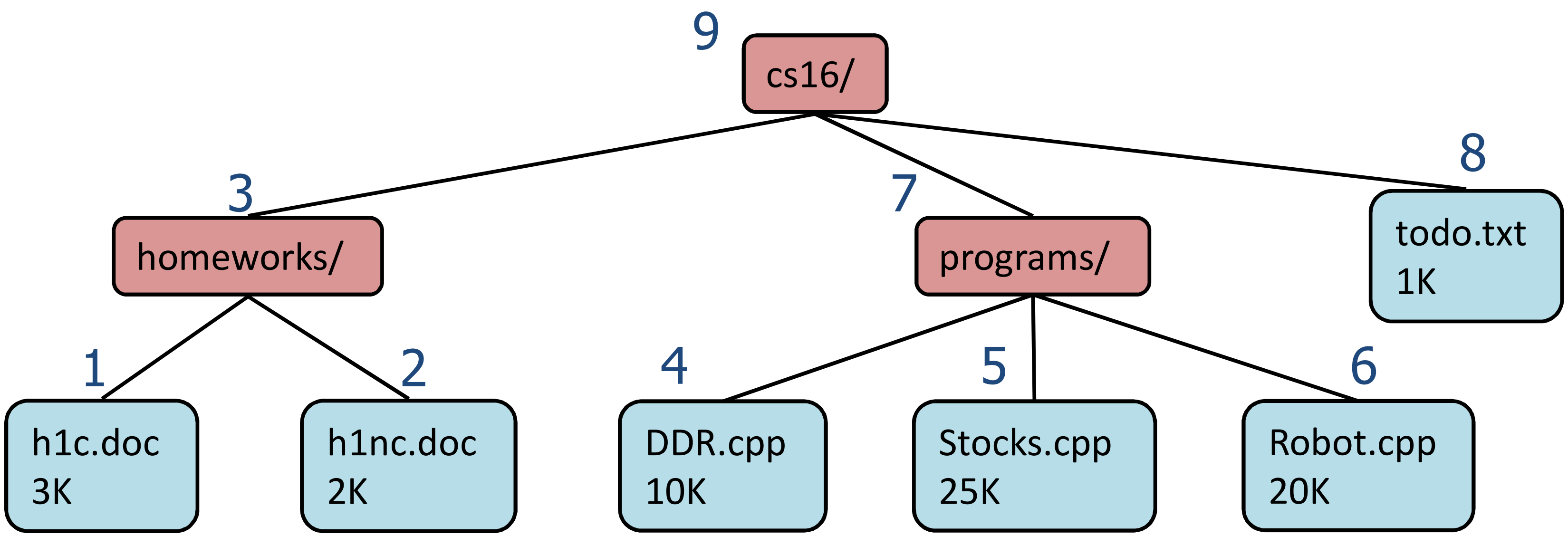


ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Postorder Traversal

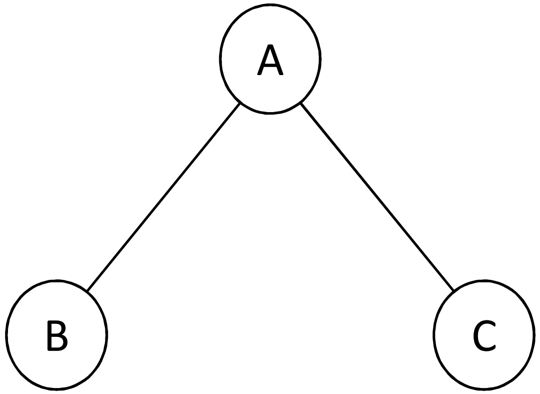
- In a postorder traversal, a node is visited after its descendants

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



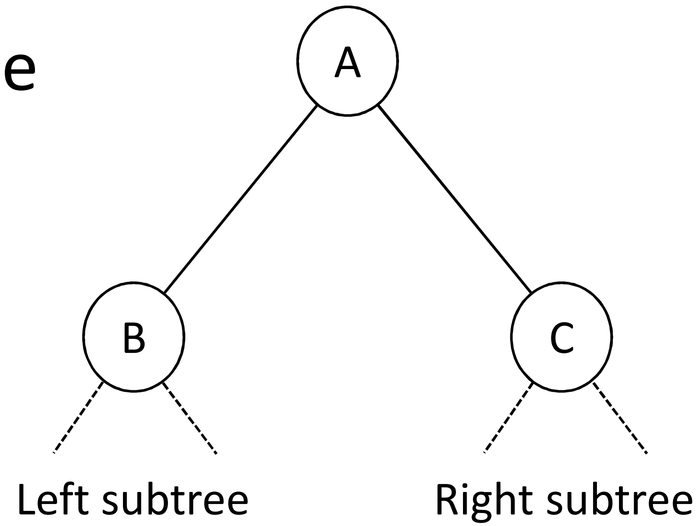
Tree Representation

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



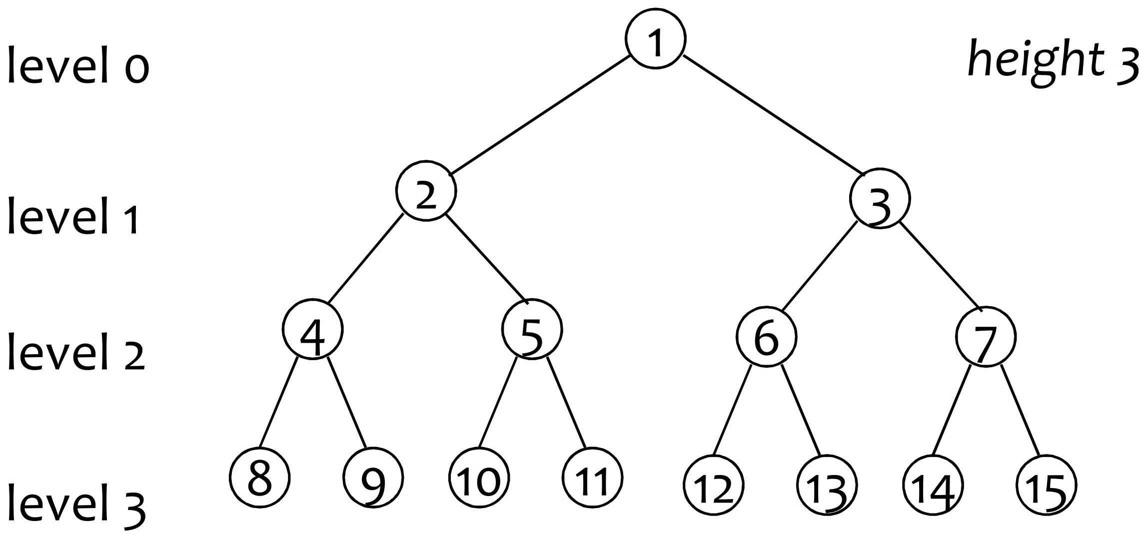
Binary Trees

- Definition
  - Finite set of nodes that consist of a root and two disjoint binary trees called the left subtree and the right subtree
  - Or empty tree



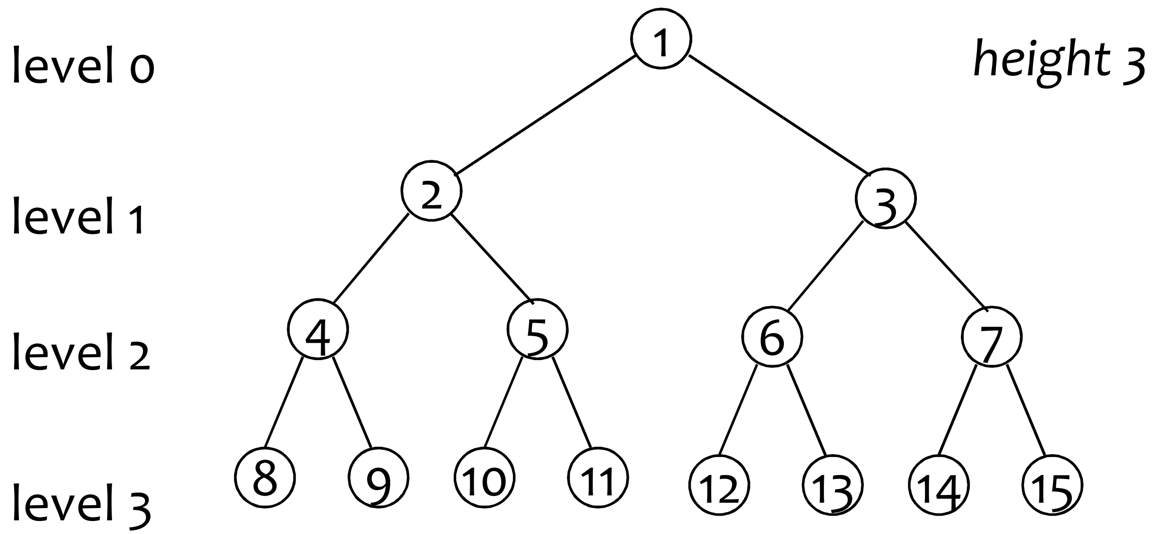
Properties of Binary Trees

- Maximum # of nodes on level i
  - 2^i, i>=0
- Maximum # of nodes in a binary tree of height k
  - $1 + 2 + 2^2 + 2^3 \ldots + 2^k = 2^{k+1}-1$



Full Binary Tree

- For height k, the tree has 2^{k+1}-1 nodes
  - Example: a full binary tree for height 3



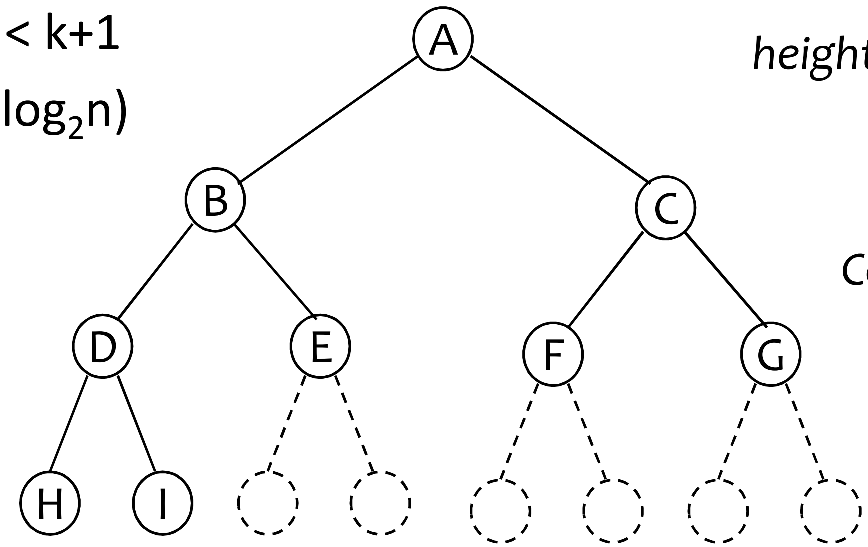
Complete Binary Tree

- All levels except the last level are fully filled
- All leaf nodes are filled from left to right
- Height of a complete binary tree (n nodes)

$2^k \leq n < 2^{k+1}$

$\rightarrow k \leq \log_2 n < k+1$

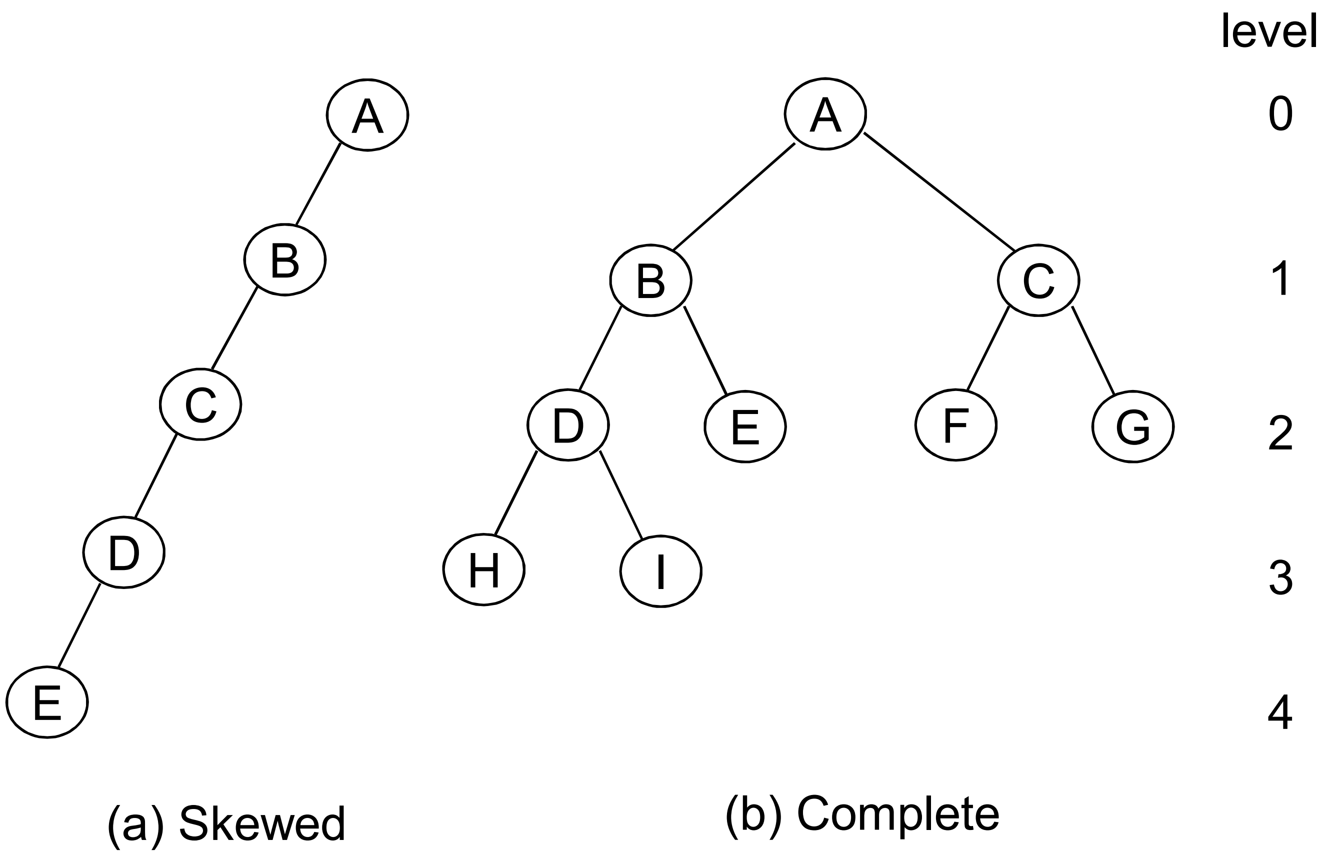
$\rightarrow k = \mathrm{floor}(\log_2 n)$



height 3

Complete Binary tree

Binary Trees



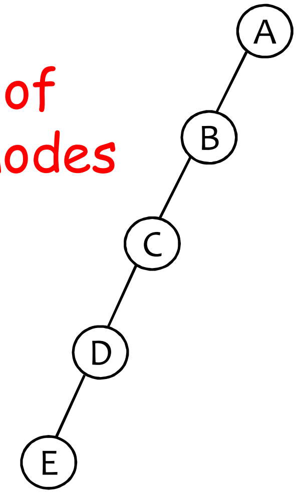
Tree Representation

- Linked-list based node structure for binary tree

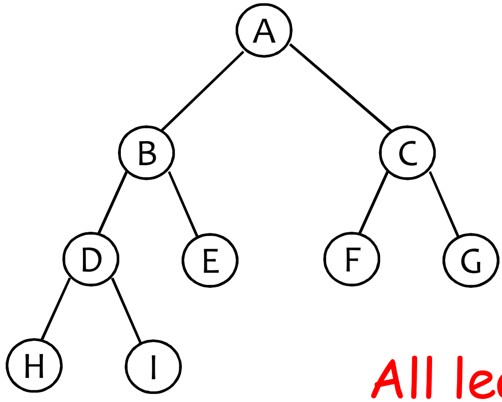
data	$CHILD_1$	$CHILD_2$

- All fields are efficiently used?

Right childs of
all internal nodes
are empty



(a)



All leaf nodes
have empty childs

(b)

Tree Representation

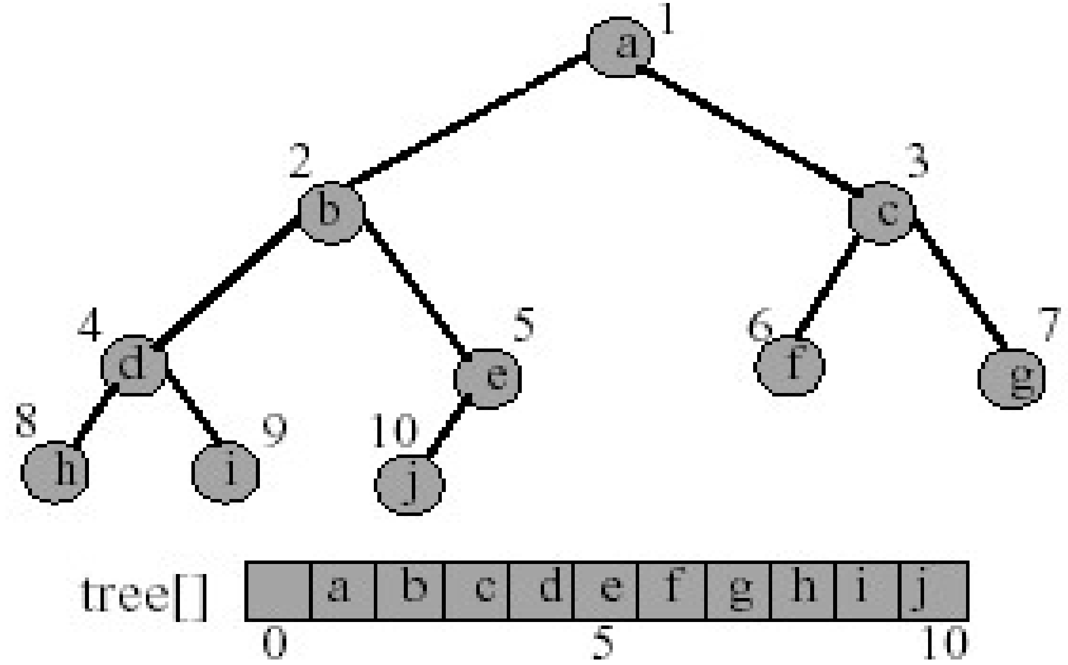
- Linked-list based node structure for a tree of degree k

data	$CHILD_1$	$CHILD_2$	…	$CHILD_k$

- Inherent space waste by zero child pointers
  - 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

Binary Tree Representation using Array

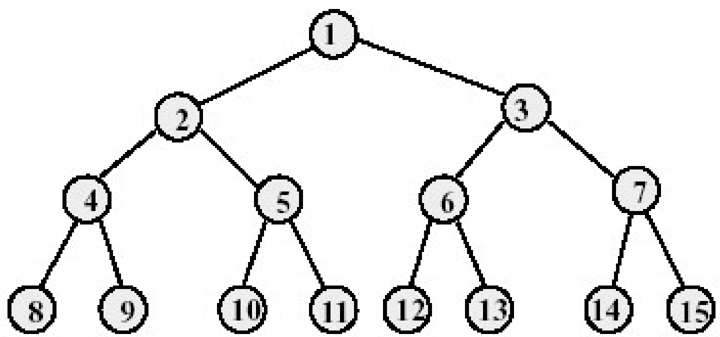
- Array representation
  - Each node is stored at the array position corresponding to the number assigned to it



ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

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 is 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) = $\lfloor 3.5 \rfloor = 3$

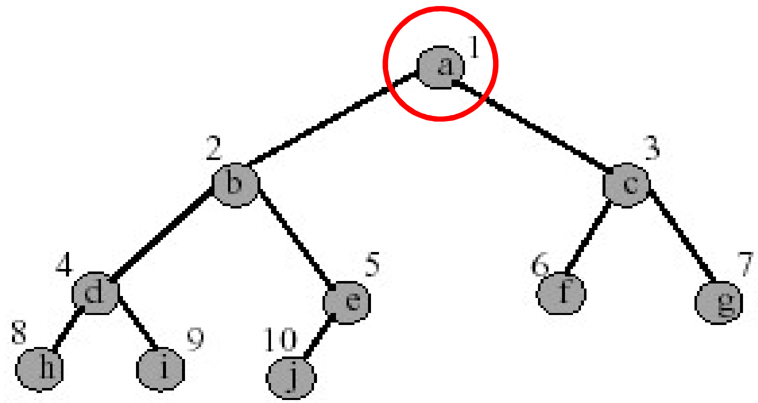


Binary Tree Representation using Array

- Traverse array

a : T(1)

	a	b	c	d	e	f	g	h	i	j				
0	1	2	3	4	5	6	7	8	9	10				

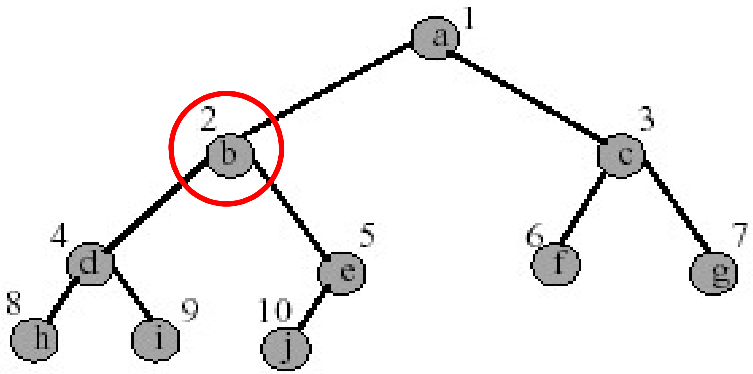


Binary Tree Representation using Array

- Traverse array

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

	a	b	c	d	e	f	g	h	i	j				
0	1	2	3	4	5	6	7	8	9	10				

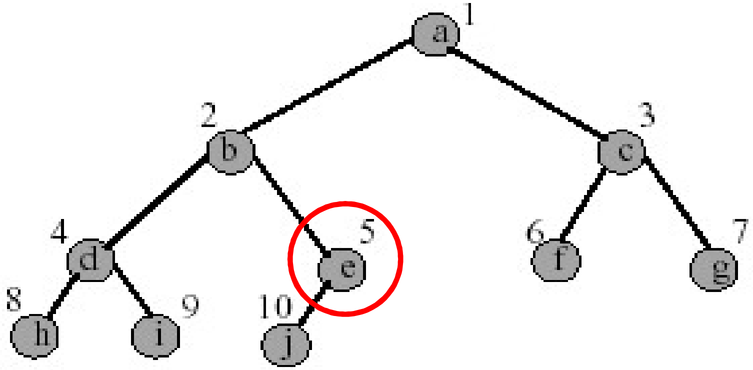


Binary Tree Representation using Array

- Traverse array

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

	a	b	c	d	e	f	g	h	i	j				
0	1	2	3	4	5	6	7	8	9	10				

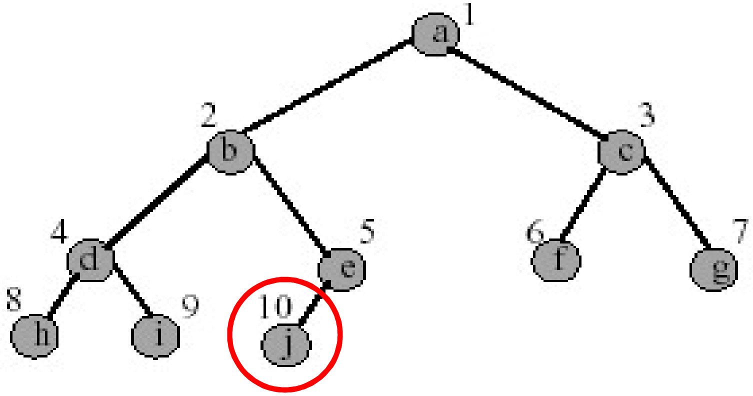


Tree Representation using Array

- Traverse array

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

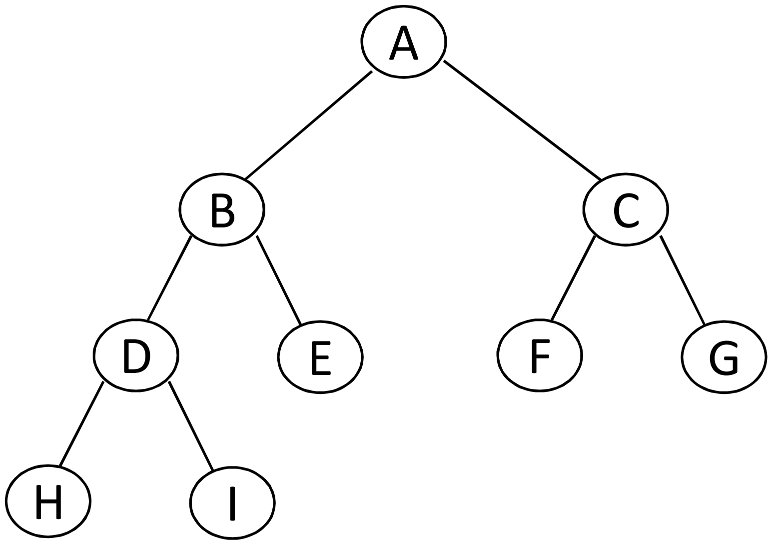
	a	b	c	d	e	f	g	h	i	j				
0	1	2	3	4	5	6	7	8	9	10				



ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Binary Tree Representation using Array

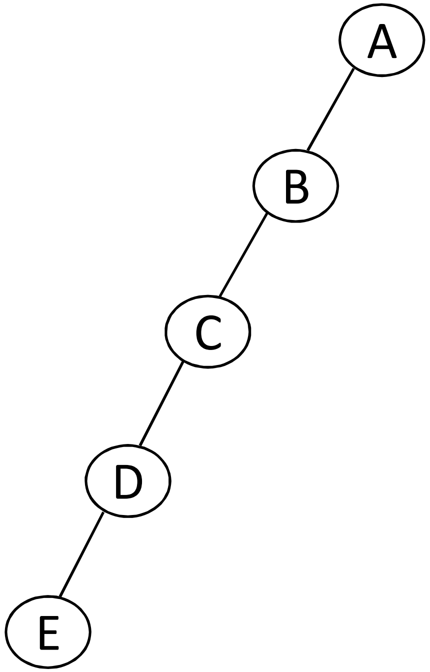
- Space usage
  - Best case : binary tree is complete



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

Binary Tree Representation using Array

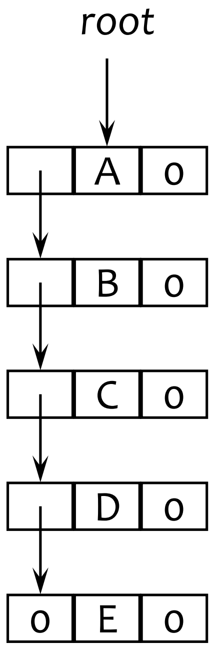
- Space usage
  - Worst case : binary tree is skewed
    - k+1 is used out of 2^{k+1}-1 (height k)



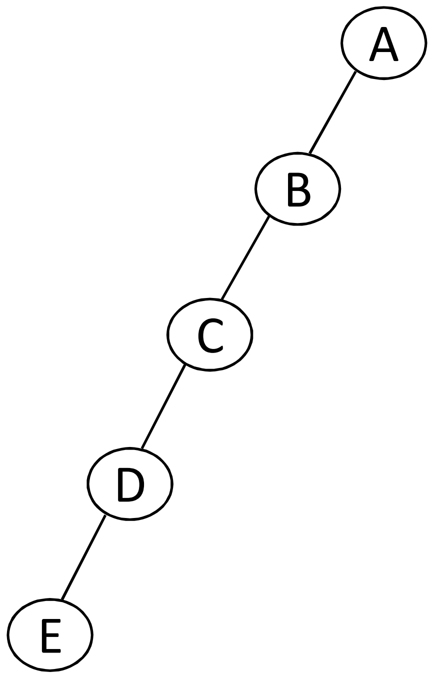
[0]	-
[1]	A
[2]	B
[3]	-
[4]	C
[5]	-
[6]	-
[7]	-
[8]	D
[9]	-
. . .	. . .
[16]	E

Array v.s. Linked List for Binary Trees

- Worst case in array-based binary tree



Linked list based
(space = 15)

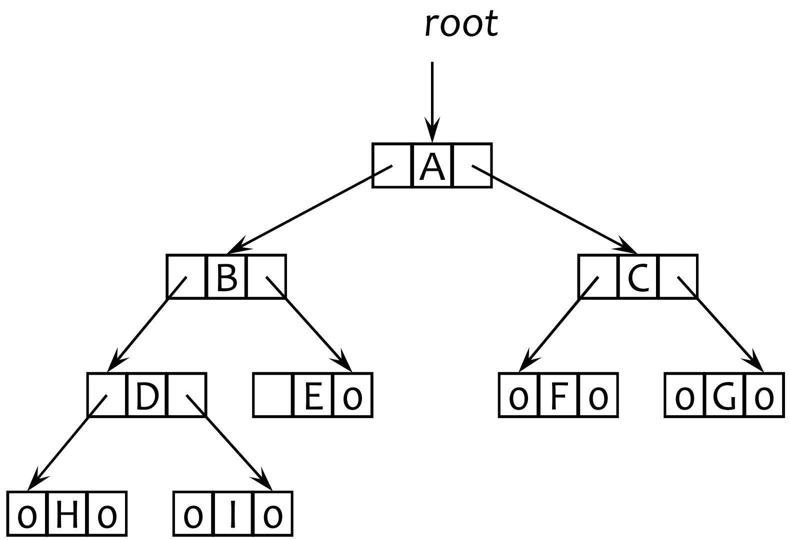


Array based
(space = 32: right skew,
17: left skew)

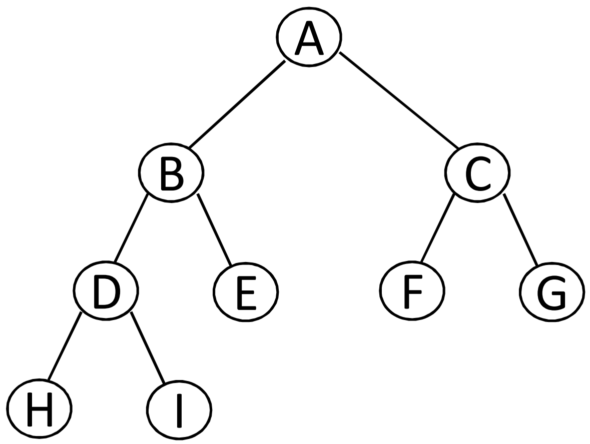
[0]	-
[1]	A
[2]	B
[3]	-
[4]	C
[5]	-
[6]	-
[7]	-
[8]	D
[9]	-
. . .	. . .
[16]	E

Array v.s. Linked List for Binary Trees

- Best case in array-based binary tree



Linked list based
(space = 27)



Array based
(space = 10)

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

Tree Using Linked List

- Pros
  - Efficient memory management in many incomplete binary trees
  - Node insert / delete can be fast
    - Array-based tree requires shifting elements
- Cons
  - Extra memory to store pointers for complete trees
  - Traversing parent needs extra pointer per node

data	PARENT	$CHILD_1$	…	$CHILD_k$

Binary Tree Class

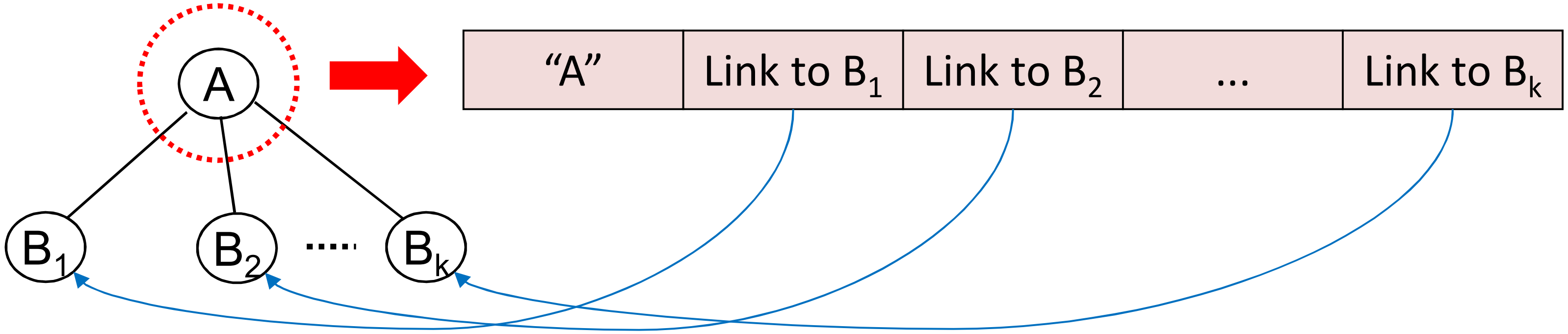
```
class TreeNode{
private:
        TreeNode *LeftChild;
        char data;
        TreeNode *RightChild;
};

class Tree {
public:
        // Tree operations
        ...
private:
        TreeNode *root;
};
```

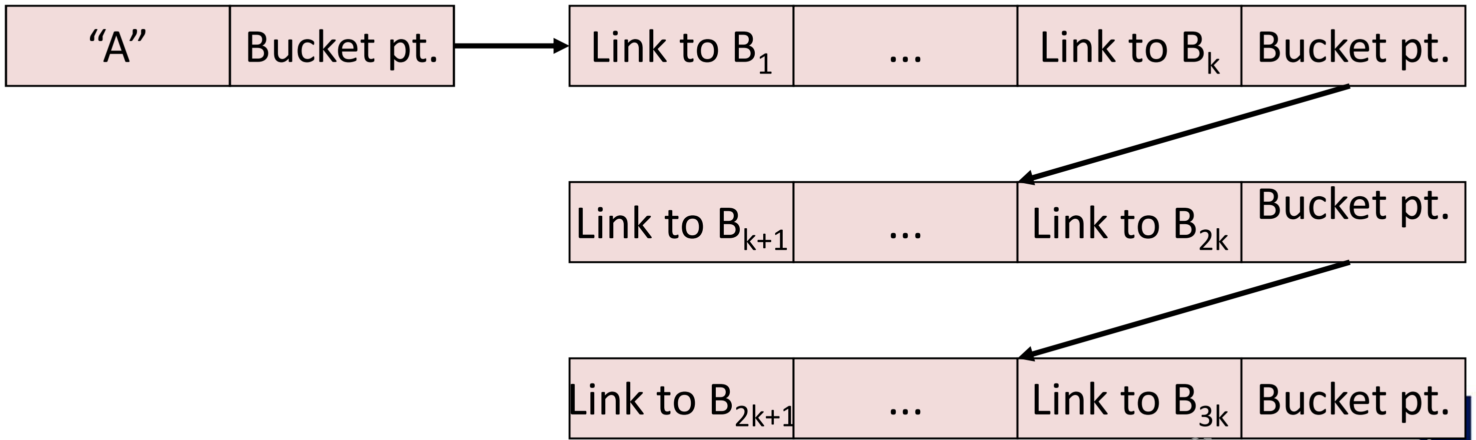
Can We Make Node Size Flexible?

- A lot of engineering trick is possible
  - Bucket size k can be arbitrary, but not 1 (↑overhead)
  - Dynamically add a bucket to have more childs
  - Can merge buckets if there are too many buckets

Increasing Degree Dynamically



If node A has to include any number of children



Questions?