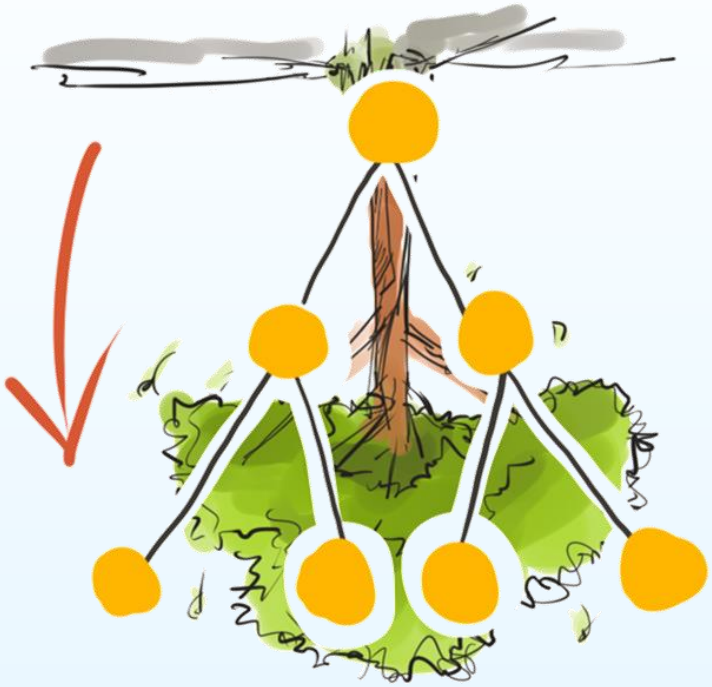


Lecture 8: Binary Trees

01204212 Abstract Data Types and Problem Solving

Department of Computer Engineering
Faculty of Engineering, Kasetsart University
Bangkok, Thailand.

Department of Computer Engineering Kasetsart University

MIKE LABORATORY

Motivation

A general tree is appropriate for storing hierarchical orders, where the relationship is between the parent and the children. There are many cases, however, where the tree data structure is more useful if there is a fixed number of identifiable children. This topic looks at binary trees and their families.

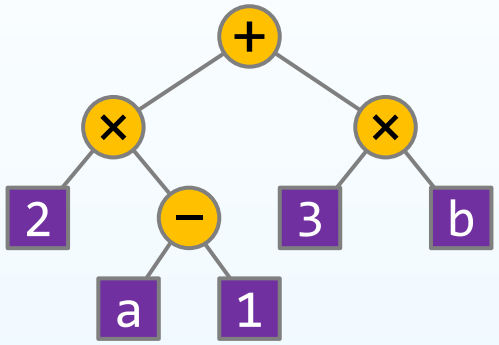
Outline

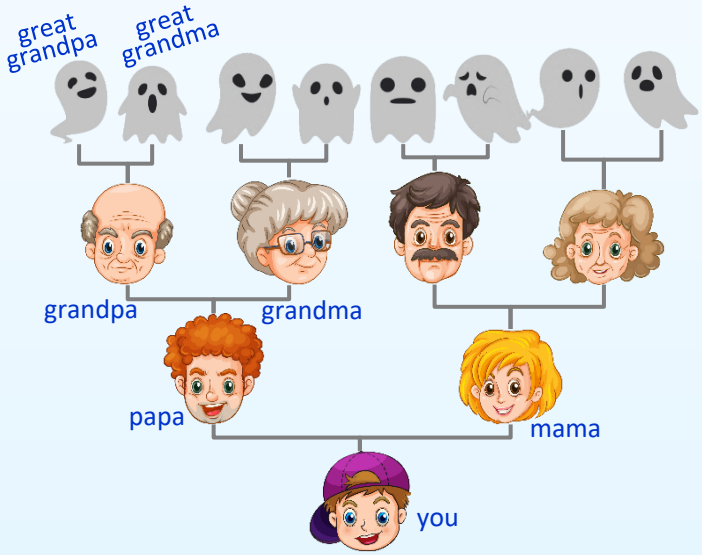
- Binary Trees
- Perfect Binary Trees
- Complete Binary Trees
- N-ary Trees
- Binomial Trees
- Left-child Right-sibling Binary Trees

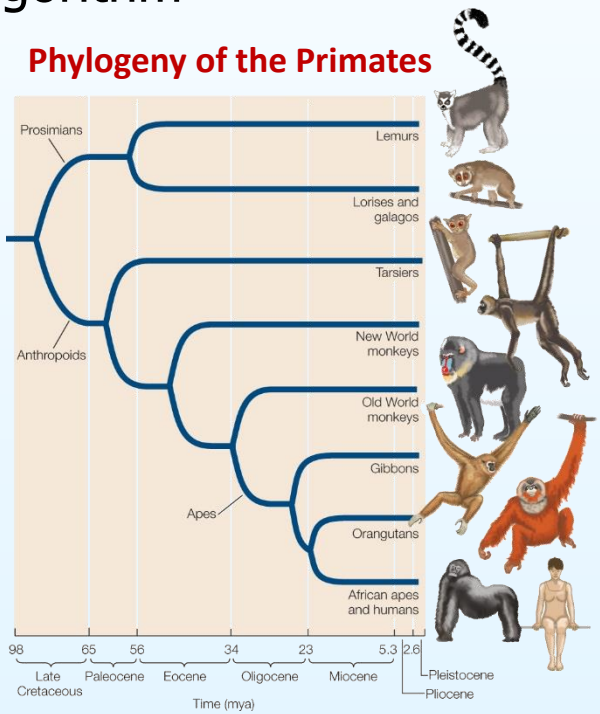
Binary Trees

- Many real-life trees are restricted to two branches:
  - Expression trees using binary operators
  - An ancestral tree of an individual
  - Phylogenetic trees
  - Lossless encoding algorithm

(2×(a−1)+(3×b))

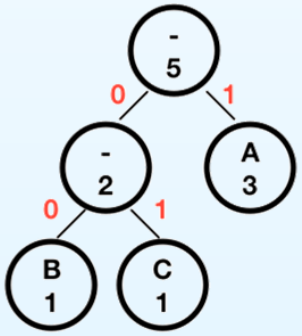






https://www.macmillanhighered.com

String to be encoded: ABACA



Prefix Codes
A - 1
B - 00
C - 01

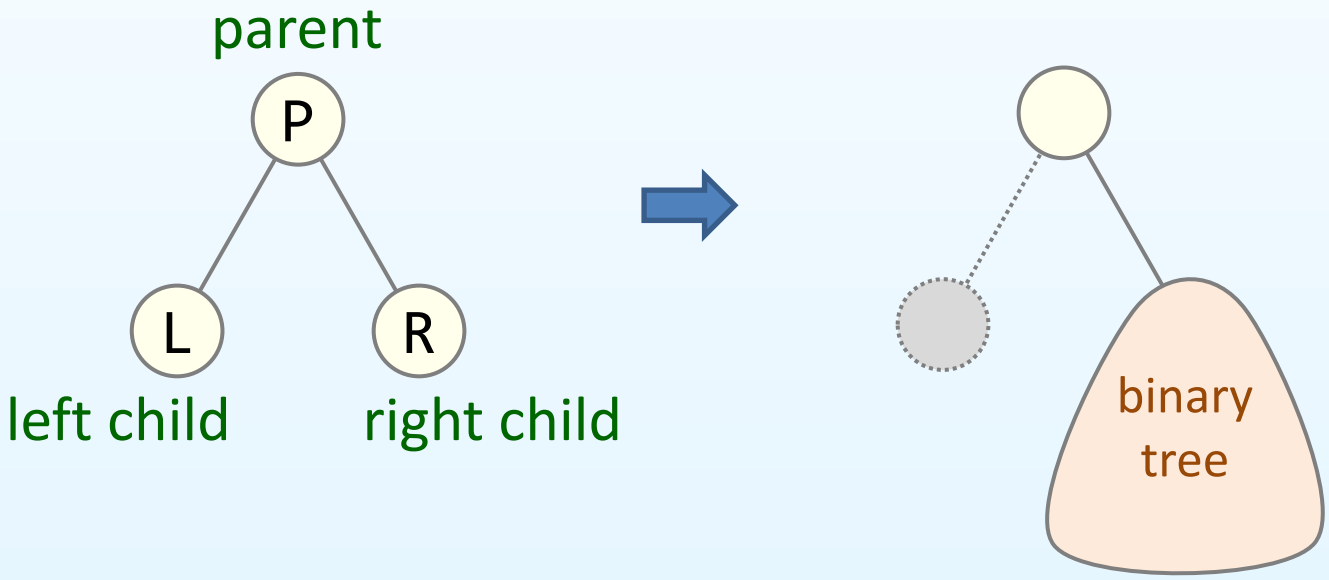
Huffman encoding

https://www.journaldev.com

Definition

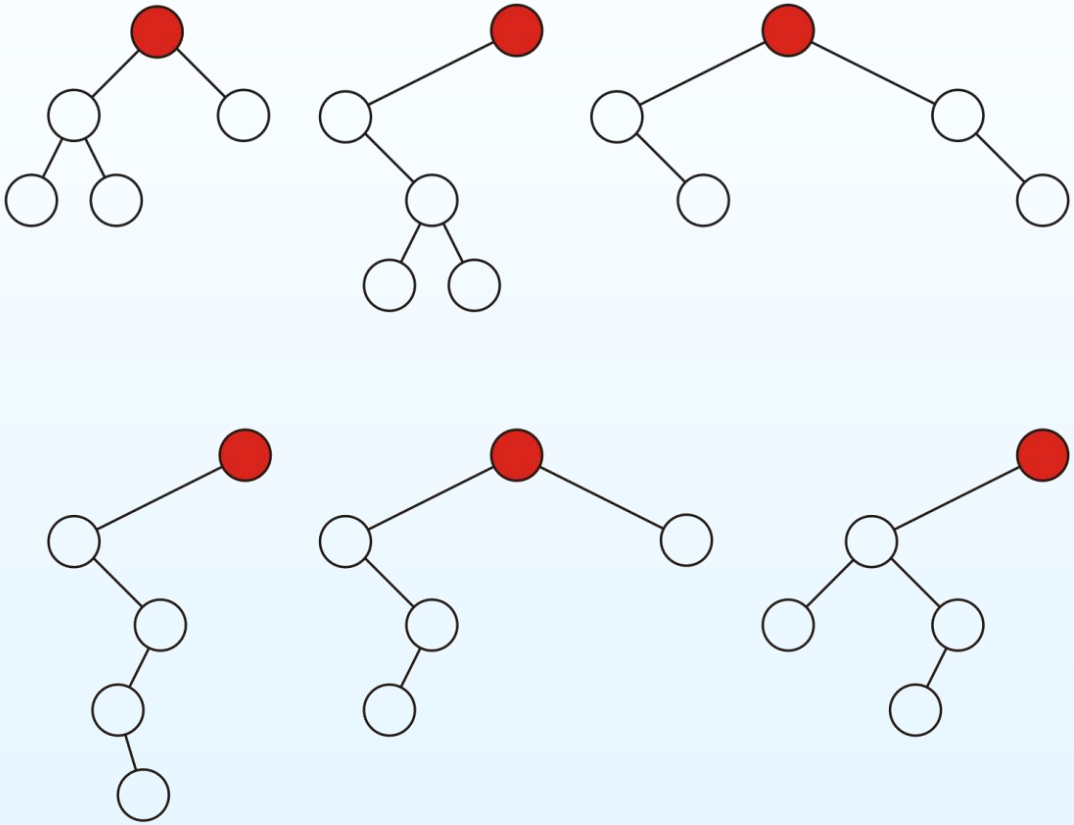
A binary tree is a restriction where each node has exactly two children:

- Each child is either empty or another binary tree
- This restriction allows us to label the children as *left* and *right* subtrees



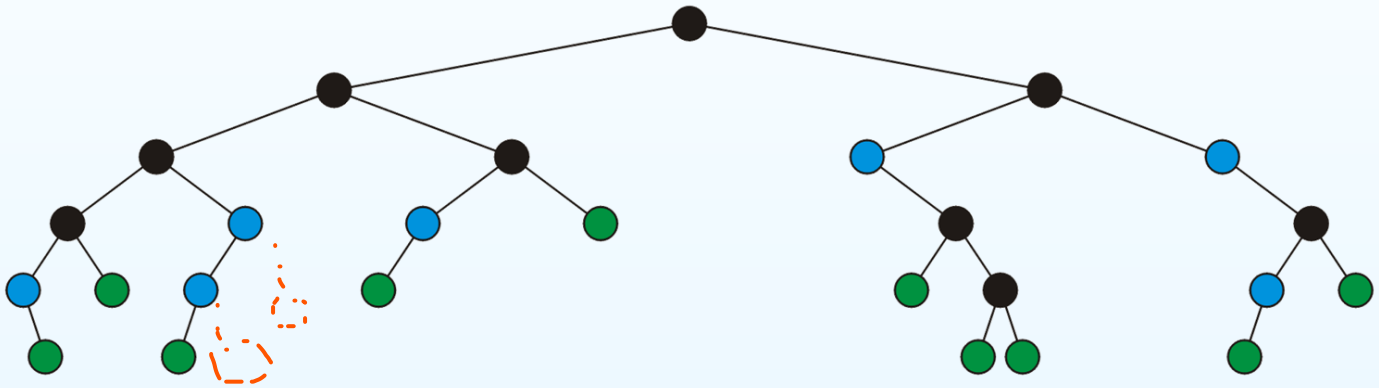
Example of Binary Trees

Sample variations on binary trees with five nodes:



Definition

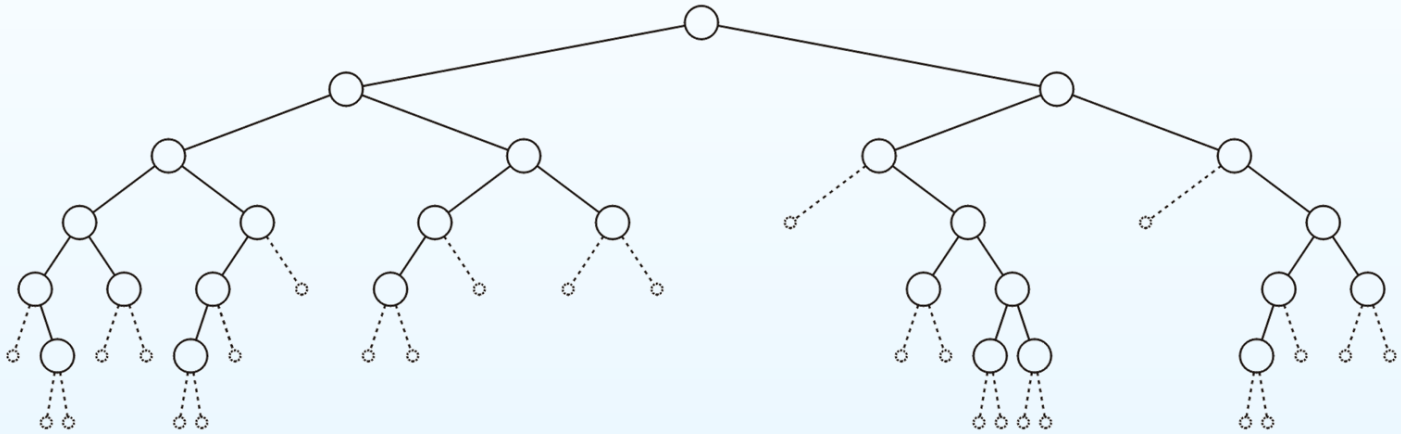
A full node is a node where both the left and right sub-trees are non-empty



- full nodes
- neither
- leaf nodes

Definition

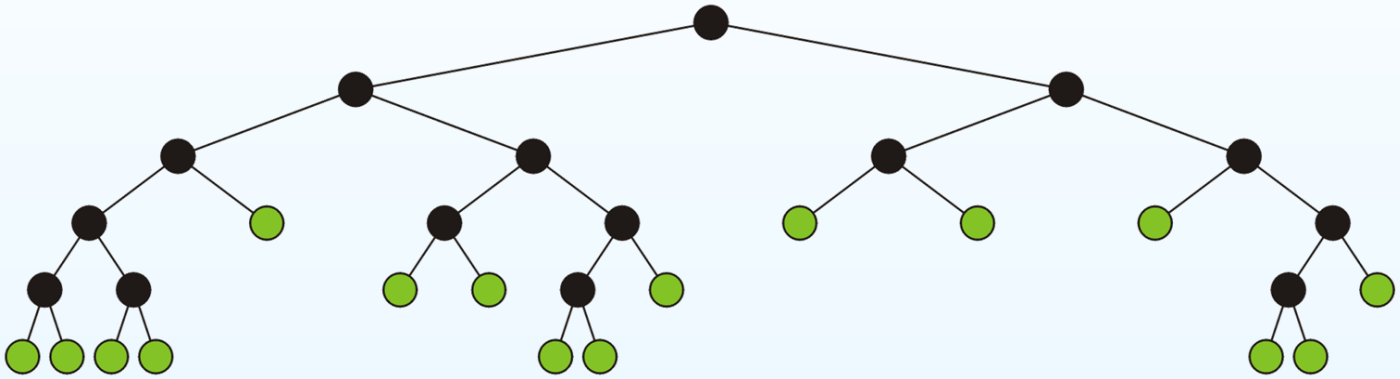
An empty node or a null subtree is any location where a new leaf node could be appended



Definition

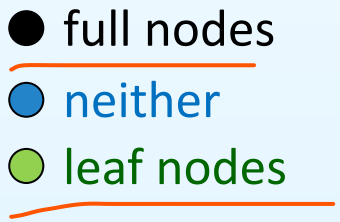
A full binary tree is where each node is

- A full node, or
- A leaf node



Applications:

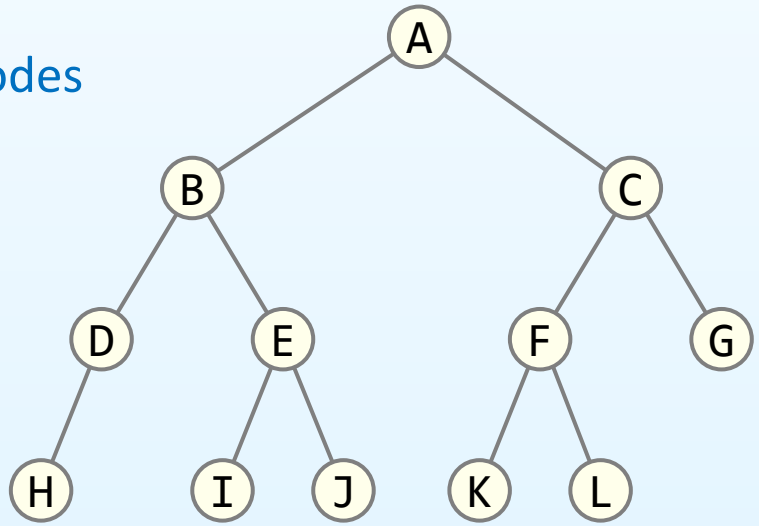
- Expression trees
- Huffman encoding



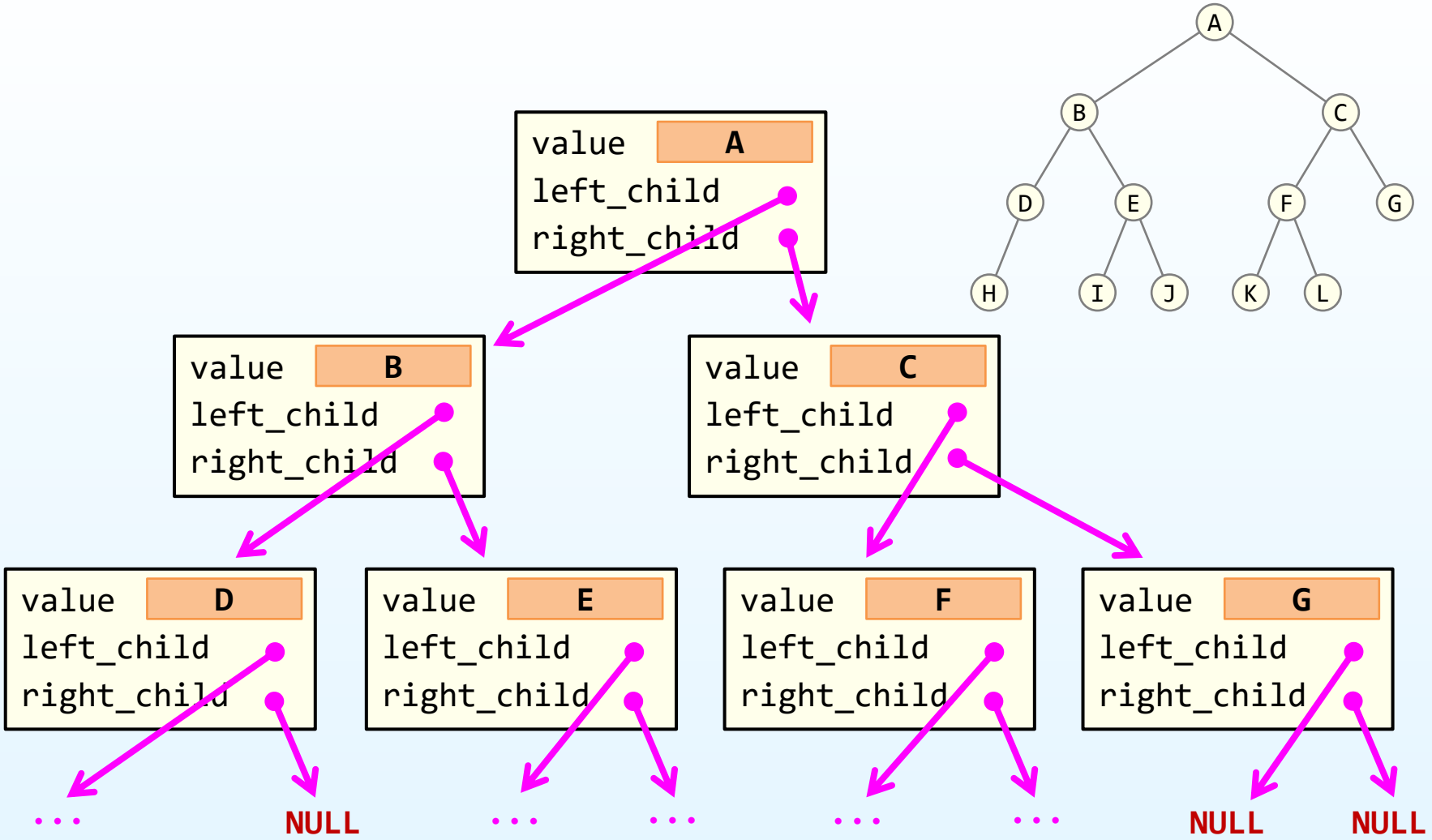
Abstract Binary Trees

- A tree data structure with a most two children

- Operations may depend on some specific properties:
  - `insert(`*tree*`, `*node*`)`
  - `delete(`*tree*`, `*subtree*`)`
  - `search(`*tree*`, `*node*`)`
  - `is_leaf(`*tree*`, `*node*`)`
  - `size(`*tree*`)` // total number of nodes
  - `height(`*tree*`)`



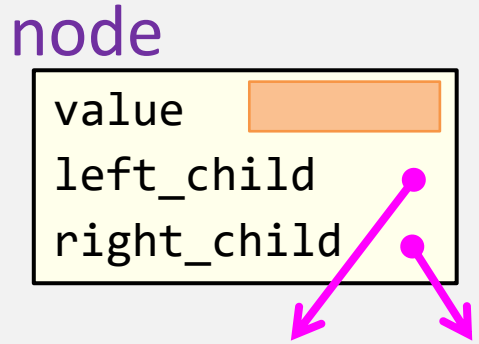
Binary Tree Implementation



Binary Tree Implementation

Assume that all data are distinct positive integers

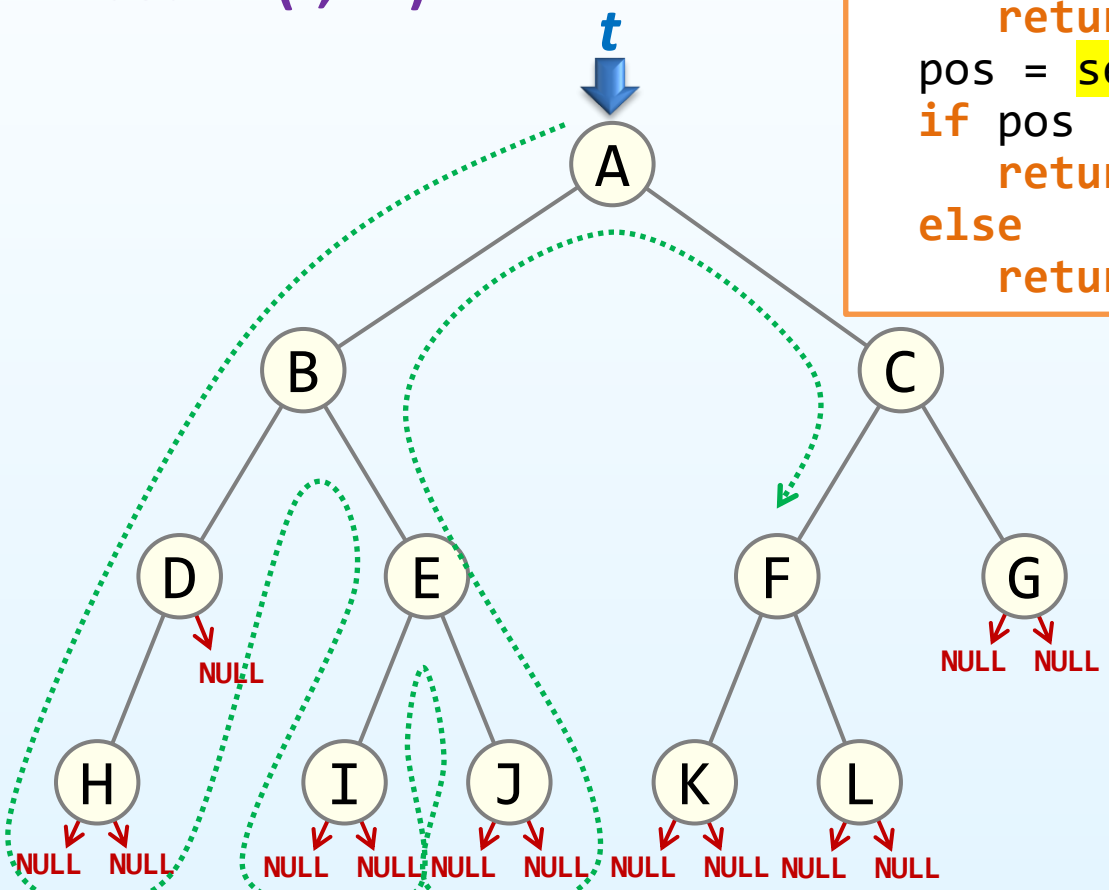
```
1:  #include <stdio.h>
2:  #include <stdlib.h>
3:
4:  typedef struct node {
5:    int value;
6:    struct node *left_child;
7:    struct node *right_child;
8:  } node_t;
9:
10: typedef node_t tree_t;
11:
12: int main(void) {
13:   tree_t *t = NULL;
14:   return 0;
15: }
```



The search() Operation

Return position of node *v* in tree *t* if found, otherwise NULL

search(*t*, 'F')



Algorithm: search(t, v)

```
if t == NULL || t->value == v
    return t
pos = search(t->left_child, v)
if pos != NULL
    return pos
else
    return search(t->right_child, v)
```

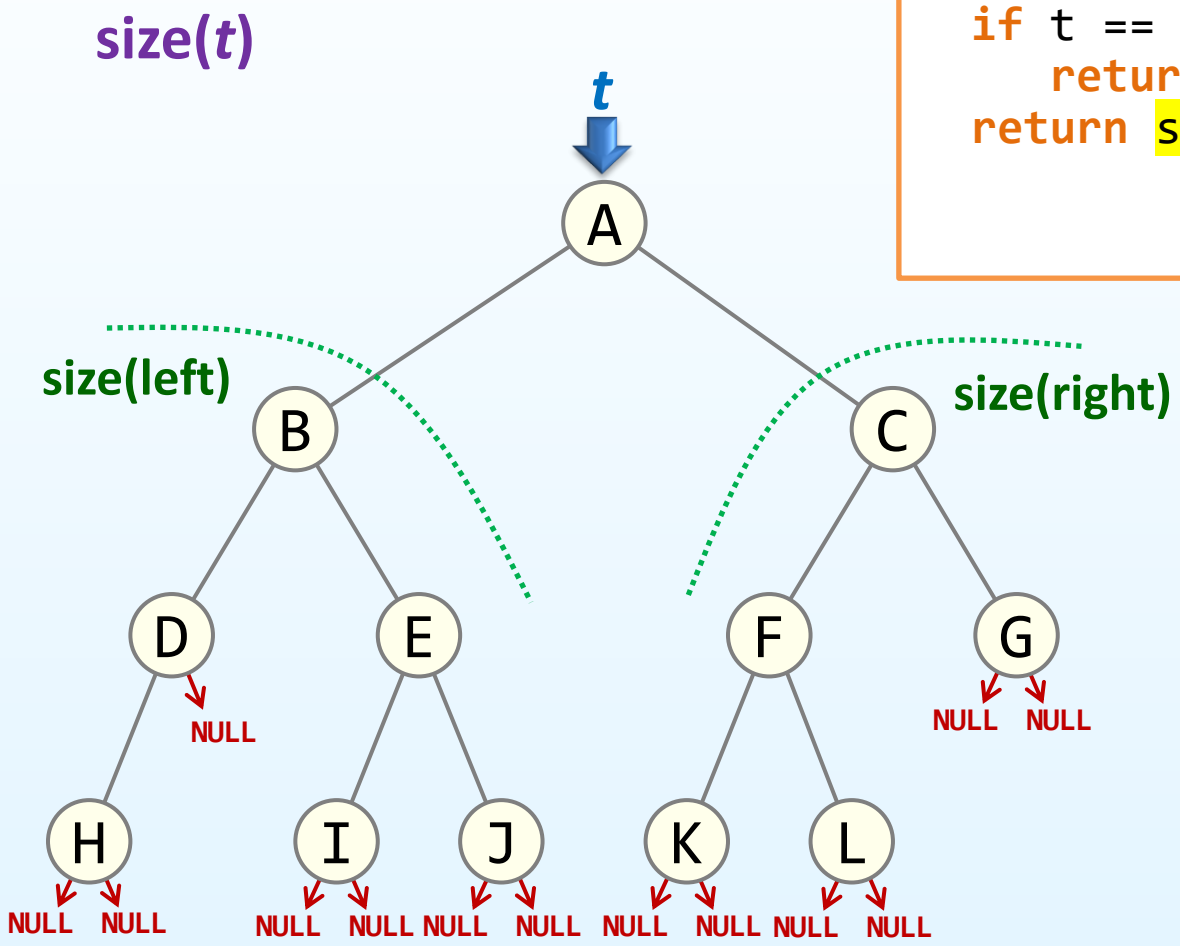
Running time
$= O(n)$

The `size()` Operation

Count total number of nodes in tree *t*

Algorithm: `size(t)`

```
if t == NULL
    return 0
return size(t->left_child)
         + size(t->right_child)
         + 1;
```



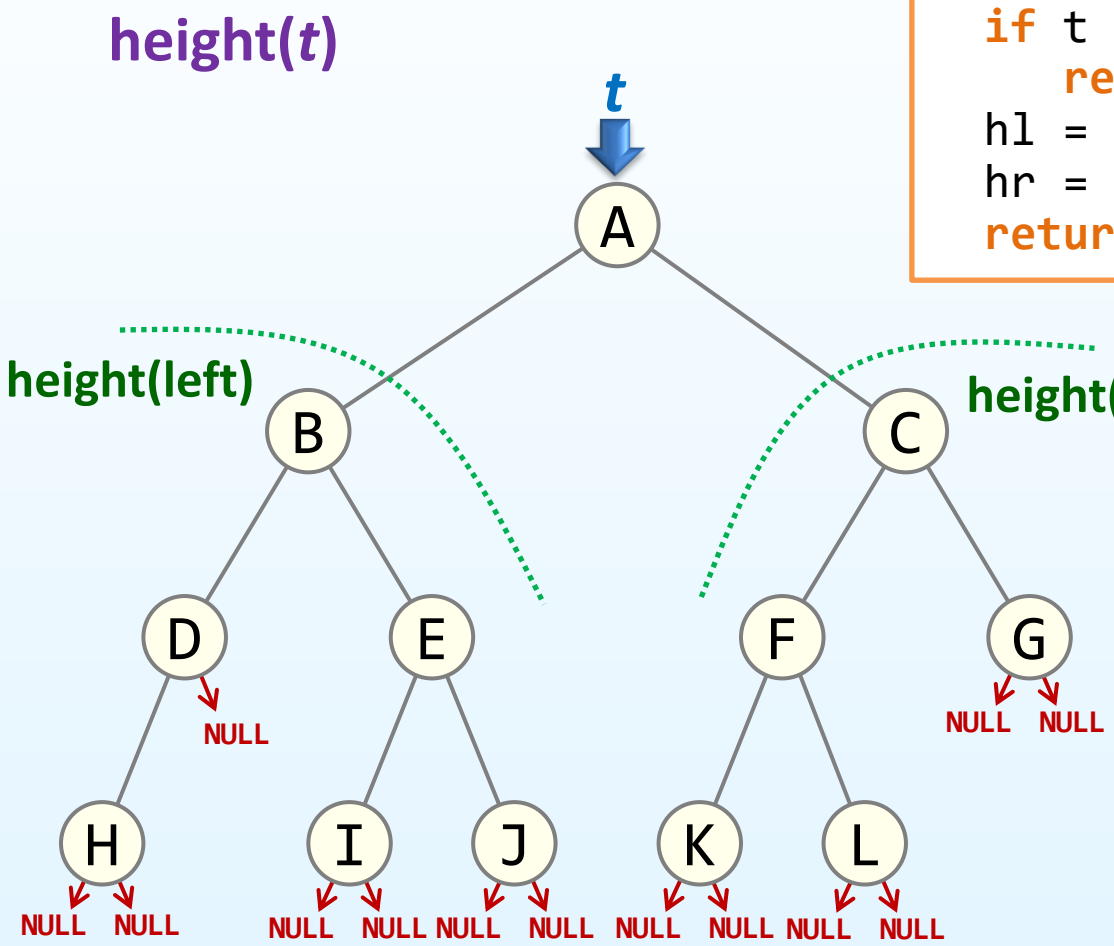
Running time = $\Theta(n)$

The height() Operation

Find the height of tree t

Algorithm: height(t)

```
if t == NULL
    return -1
hl = height(t->left_child)
hr = height(t->right_child)
return (hl>hr)? hl+1 : hr+1;
```

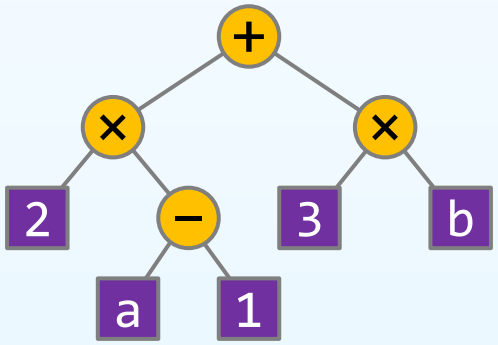


Running time = $\Theta(n)$

Application: Expression Trees

Any basic mathematical expression containing binary operators may be represented using a binary tree

For example, `(2×(a−1)+(3×b))`



Observations:

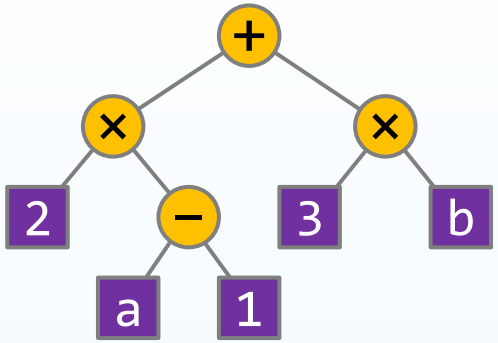
- The tree is full binary tree
- Internal nodes store operators
- Leaf nodes store literals or variables
- No parenthesis need

Depth-First Traversals

- Prefix notation (pre-order)
  - **Print:** node → left child → right child

    + × 2 − a 1 × 3 b

dfs
แบบเดินแล้วปริ้น



- Infix notation (in-order)
  - **Print:** left child → node → right child

    2 × a − 1 + 3 × b

dfs
return print

- Postfix notation (post-order)
  - **Print:** left child → right child → node

    2 a 1 − × 3 b × +

dfs

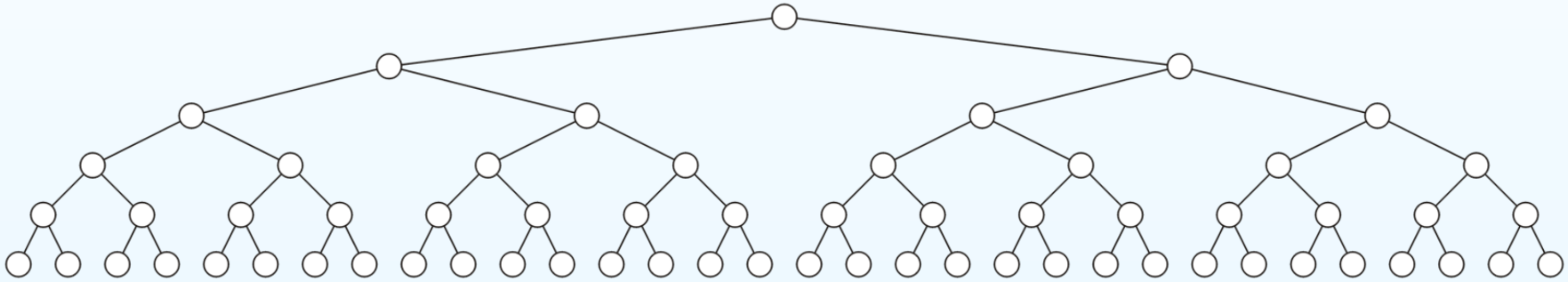
Outline

- Binary Trees
- Perfect Binary Trees
- Complete Binary Trees
- N-ary Trees
- Binomial Trees
- Left-child Right-sibling Binary Trees

Definition

A perfect binary tree of height h is a binary tree where

- All leaf nodes have the same depth h
- All other nodes are full

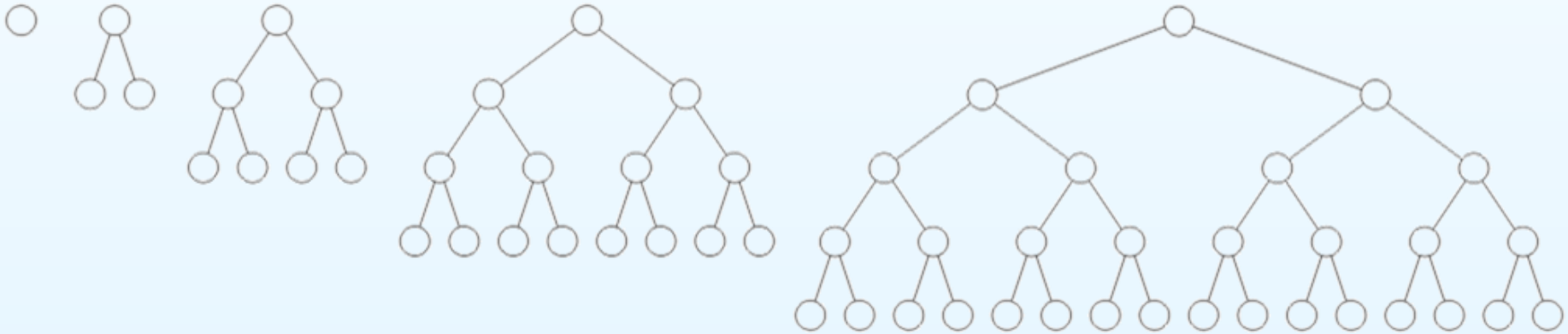


Definition (2)

Recursive definition:

- A binary tree of height $h = 0$ is perfect
- A binary tree with height $h > 0$ is a perfect if both subtrees are perfect binary tree of height $h - 1$

Examples: Perfect binary trees of height $h = 0, 1, 2, 3$ and 4



Theorems

We will look at four theorems that describe the properties of perfect binary trees:

- A perfect binary tree has $2^{h+1} - 1$ nodes
- The height is $\Theta(\log n)$
- There are 2^h leaf nodes
- The average depth of a node is $\Theta(\log n)$

The results of these theorems will allow us to determine the optimal runtime properties of operations on binary trees

1) $2^{h+1} - 1$ Nodes

Theorem:

A perfect binary tree of height h has $2^{h+1} - 1$ nodes

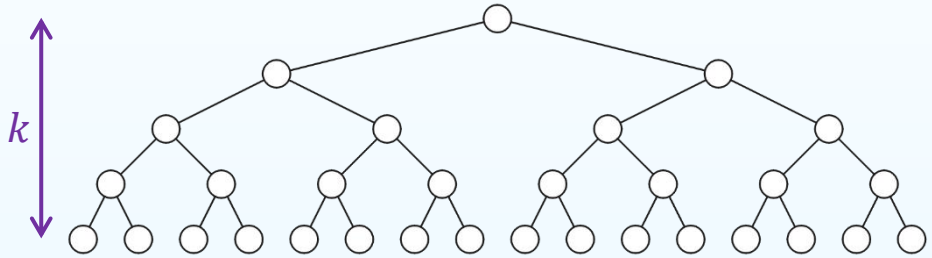
Proof:

We will use mathematical induction

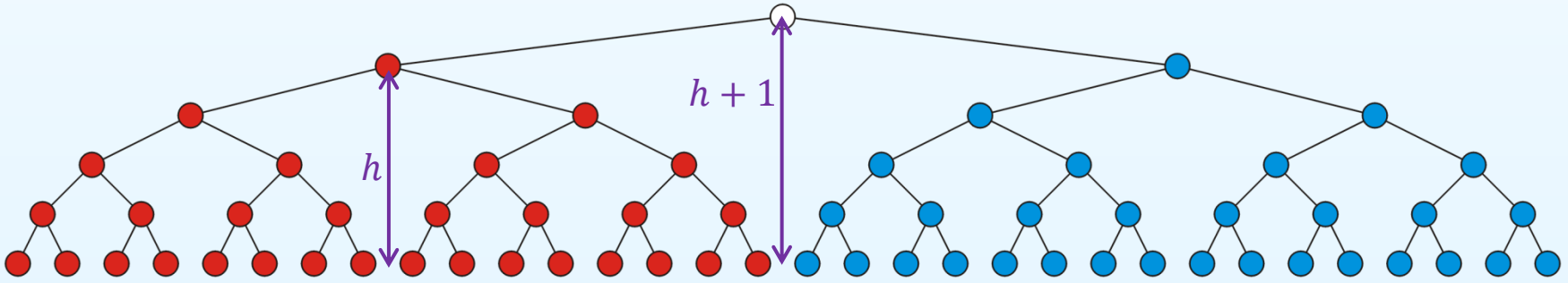
1. Show that it is true for $h = 0$
2. Assume it is true for an arbitrary h
3. Show that the truth for h implies the truth for $h + 1$

1) $2^{h+1} - 1$ Nodes

- The base case:
  - For $h = 0$, the tree has $2^{0+1} - 1 = 1$ node ○
- Inductive step:
  - By hypothesis, for any $0 < k \leq h$, the tree has $2^{k+1} - 1$ nodes



  - We must show that a tree of height $h + 1$ has $2^{(h+1)+1} - 1$ nodes



$$height(red) + height(blue) + 1 = \left(2^{h+1} - 1\right) + \left(2^{h+1} - 1\right) + 1$$
$$= 2\left(2^{h+1}\right) - 1 = 2^{(h+1)+1} - 1$$

2) Logarithmic Height

Theorem:

A perfect binary tree with n nodes has height $\log(n+1)-1$

Proof:

From the previous theorem, we have $n = 2^{h+1} - 1$

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

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

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

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

$$h = \Theta(\log n)$$

3) 2^h Leaf Nodes

Theorem:

A perfect binary tree with height h has 2^h leaf nodes

Proof: (by induction)

- When $h = 0$, there is $2^0 = 1$ leaf node
- For any $0 < k \leq h$, assume that a perfect binary tree of height k has 2^k leaf nodes
- We know that each leaf of a perfect binary tree must have two children to become the tree with higher height
  - A perfect binary tree with height $h + 1$ has $2(2^h) = 2^{h+1}$ leaf nodes

Consequence: Over half all nodes are leaf nodes

leaf — 2^h

node all — $2^{h+1} - 1$

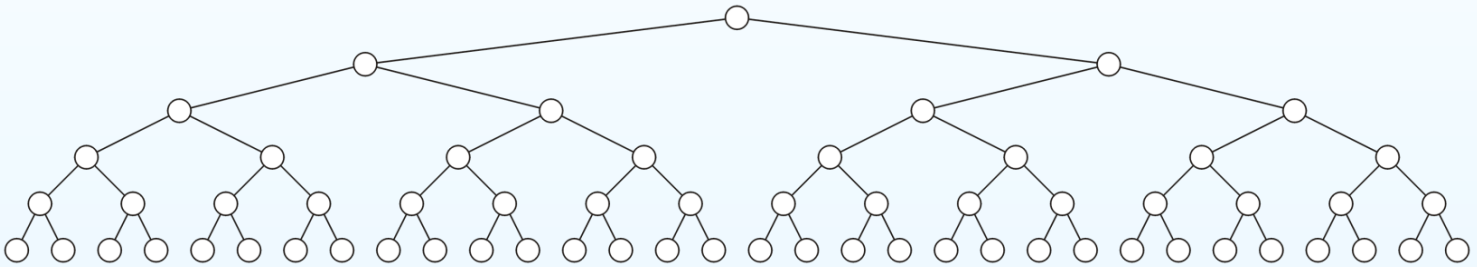
$$\frac{2^h}{2^{h+1} - 1} > \frac{1}{2}$$

4) The Average Depth of a Node

Theorem:

The average depth of a node is $\Theta(\log n)$

Proof:



Depth	#Nodes
0	1
1	2
2	4
3	8
4	16
5	32

$$avg.depth = \frac{\sum_{k=0}^{h} k2^k}{2^{h+1}-1} = \frac{h2^{h+1} - 2^{h+1} + 2}{2^{h+1}-1} = \frac{h\left(2^{h+1}-1\right) - \left(2^{h+1}-1\right) + 1 + h}{2^{h+1}-1}$$

$$= h - 1 + \frac{h+1}{2^{h+1}-1} \approx h - 1 = \Theta(\log n)$$

Consequence

- The height and average depth are both $\Theta(\log n)$

- Perfect binary trees are considered to be the ideal case
- So that we attempt to find trees which are as close as possible to perfect binary trees

Outline

- Binary Trees
- Perfect Binary Trees
- Complete Binary Trees
- N-ary Trees
- Binomial Trees
- Left-child Right-sibling Binary Trees

Motivation

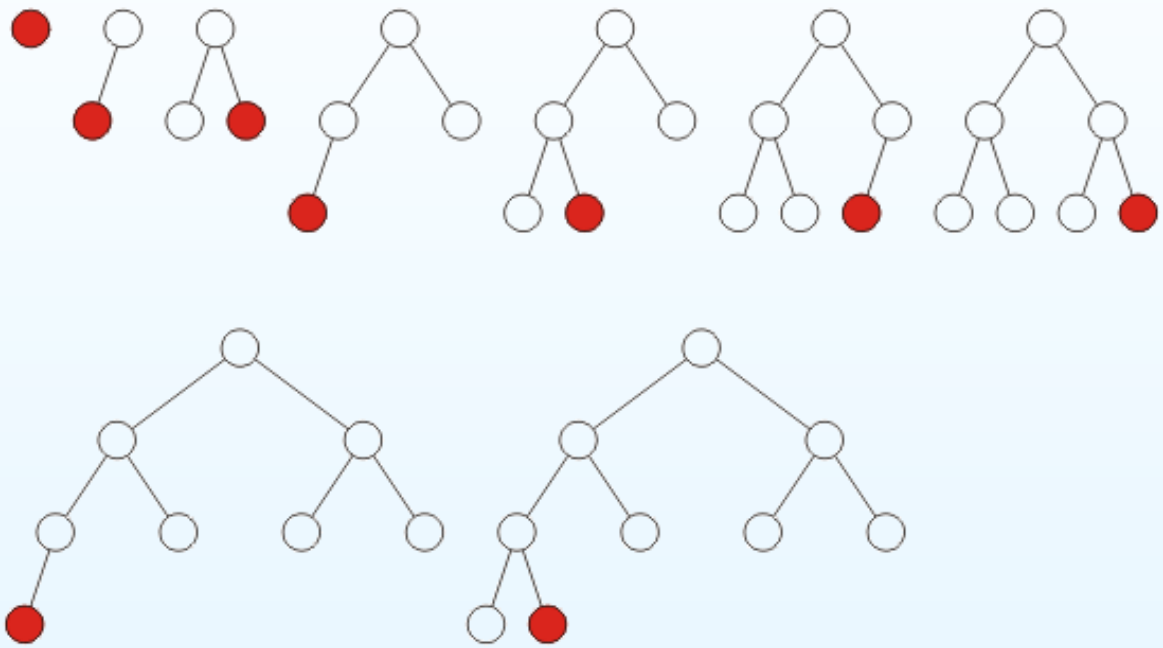
- A perfect binary tree has ideal properties but restricted in the number of nodes: $n = 2^{h+1} - 1$ for $h = 0,1,2, \ldots$
  - i.e., 1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, …

- We require binary trees which are
  - Similar to perfect binary trees, but
  - Defined for all n

Definition

A complete binary tree filled at each depth from left to right

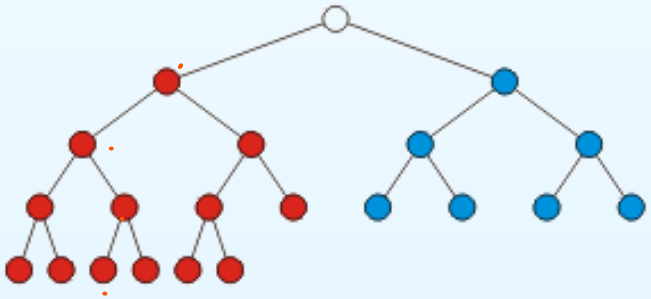
- The order is identical to that of a breadth-first traversal

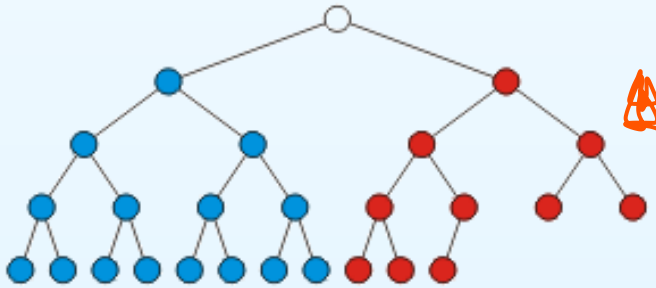


Definition (2)

Recursive definition:

- A binary tree of height $h = 0$ (a single node) is complete
- A binary tree with height $h > 0$ is a complete where either:
  - The left subtree is a complete tree of height $h - 1$ and the right subtree is a perfect tree of height $h - 2$, or
  - The left subtree is a perfect tree of height $h - 1$ and the right subtree is a complete tree of height $h - 1$





Height

Theorem:

A complete binary tree with n nodes has height $h = \lfloor \log n \rfloor$

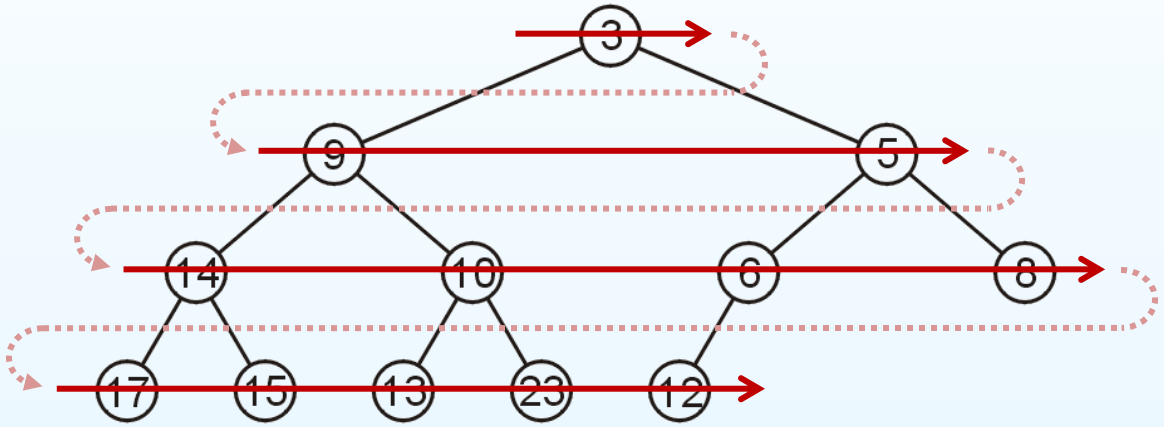
Proof: (by induction)

- When $n = 1$ then $\lfloor \log 1 \rfloor = 0$, a tree with one node is a complete binary tree with height $h = 0$
- Assume that a complete binary tree with n nodes has height $\lfloor \log n \rfloor$
- Must show that $\lfloor \log(n+1) \rfloor$ gives the height of a complete binary tree with $n + 1$ nodes

Array Storage

We are able to store a complete binary tree as an array

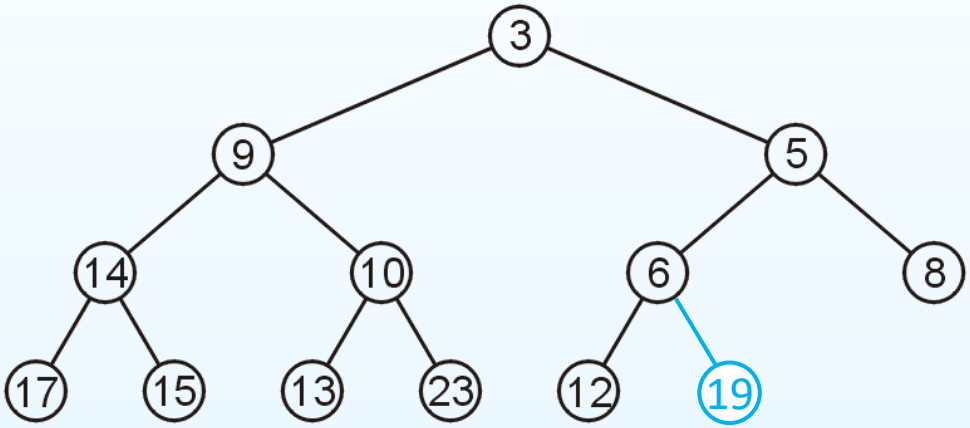
- Traverse the tree in breadth-first order, placing the entries into the array



	3	9	5	14	10	6	8	17	15	13	23	12							

Array Storage: Insertion

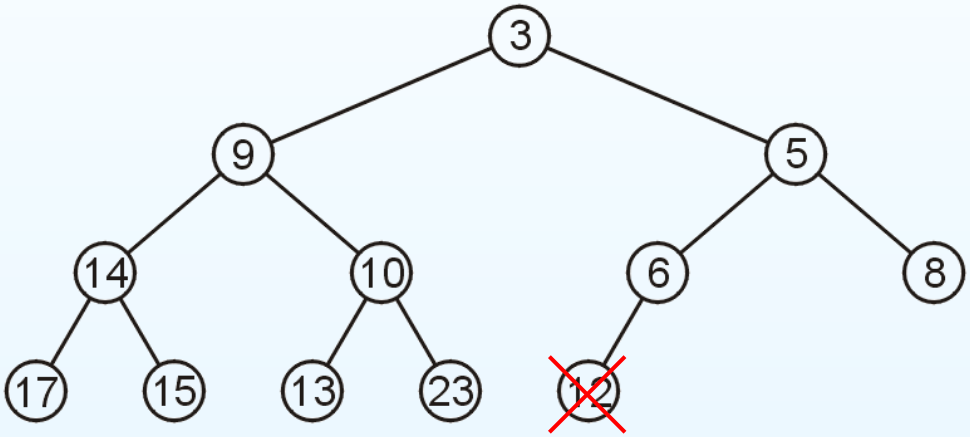
To insert another node while maintaining the complete binary tree structure, we must insert into the next array location



	3	9	5	14	10	6	8	17	15	13	23	12	19						

Array Storage: Deletion

To remove a node while maintaining the complete binary tree structure, we must remove the last element in the array



	3	9	5	14	10	6	8	17	15	13	23	~~12~~							

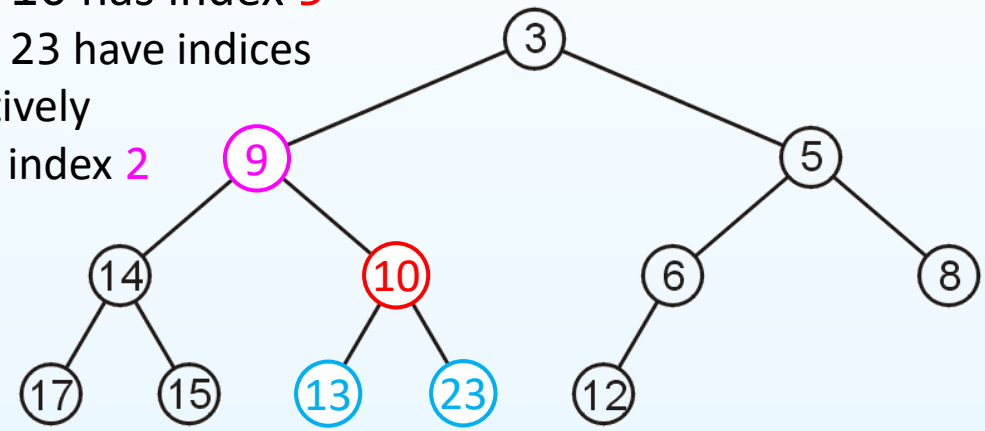
Array Storage

Leaving the first entry blank yields a bonus:

- The children of node with index k are in $2k$ and $2k + 1$
- The parent of node with index k is in $\lfloor k/2 \rfloor$

For example, node 10 has index 5

- Its children 13 and 23 have indices 10 and 11, respectively
- Its parent is 9 with index 2



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	3	9	5	14	10	6	8	17	15	13	23	12							

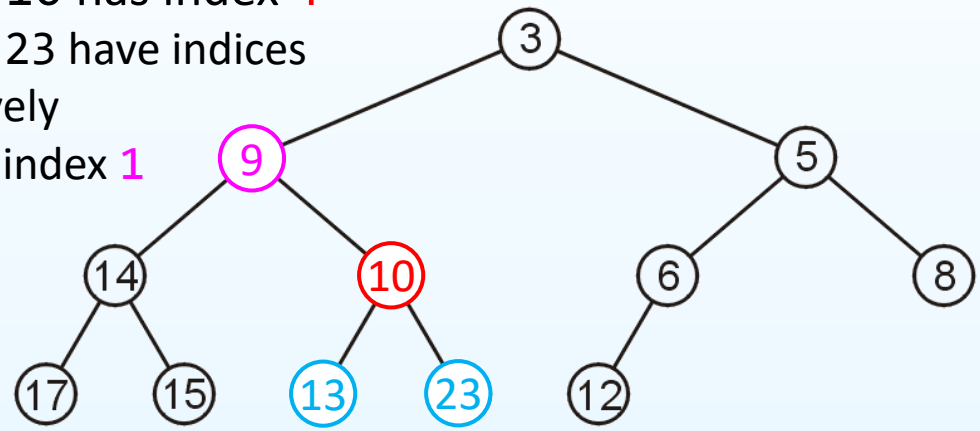
Array Storage: Alternative Method

However, we can also start with the index 0:

- The children of node with index k are in $2k + 1$ and $2k + 2$
- The parent of node with index k is in $\lfloor (k - 1)/2 \rfloor$

For example, node 10 has index 4

- Its children 13 and 23 have indices 9 and 10, respectively
- Its parent is 9 with index 1



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
3	9	5	14	10	6	8	17	15	13	23	12								

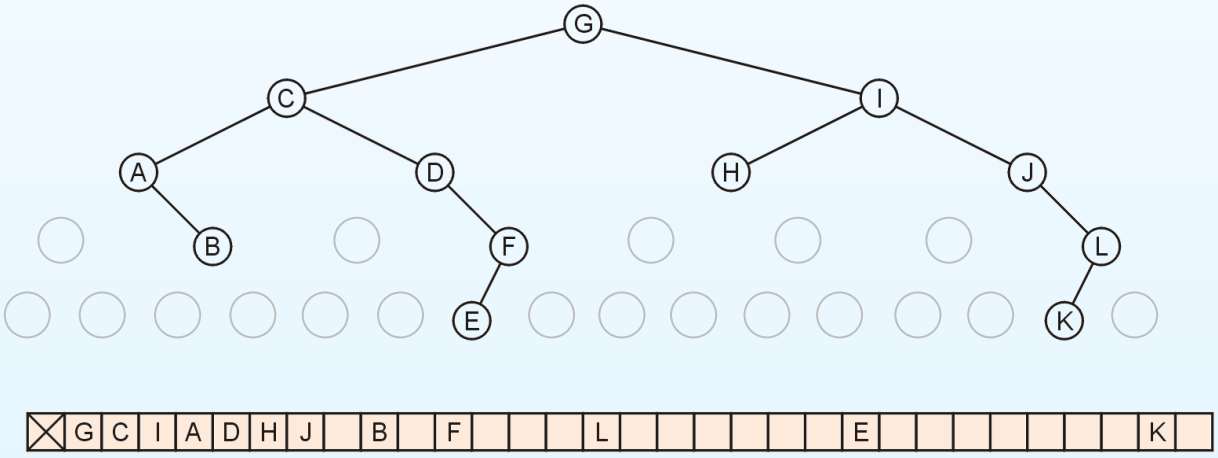
Array Storage

Question: Why don't we store any tree as an array using breadth-first traversals?

Answer: There is a significant potential for a lot of wasted memory

Consider a binary tree with 12 nodes of height 4, it would require an array of size 32

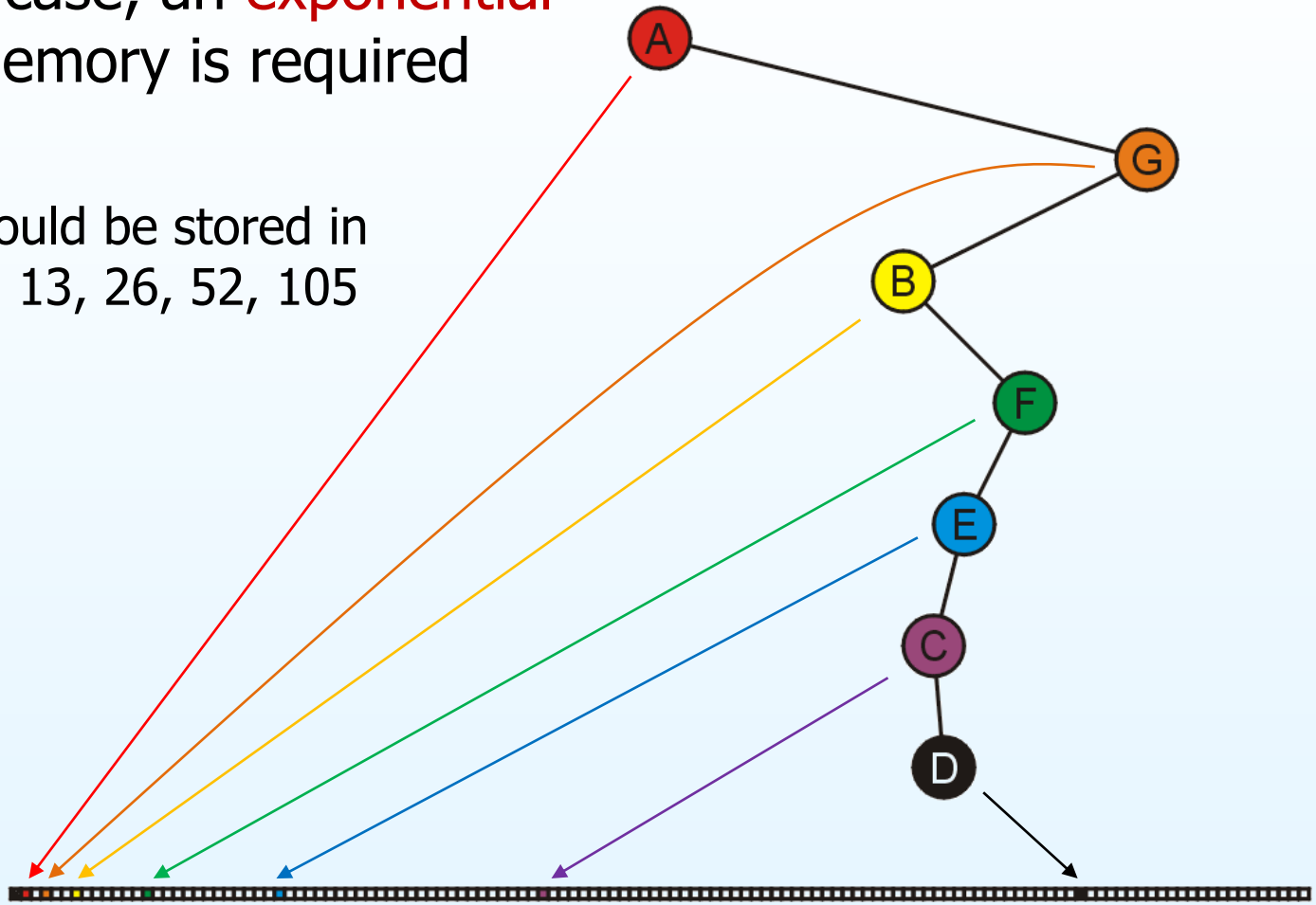
- Adding a child to node K doubles the required memory



Array Storage

In the worst case, an exponential amount of memory is required

These nodes would be stored in entries: 1, 3, 6, 13, 26, 52, 105



Outline

- Binary Trees
- Perfect Binary Trees
- Complete Binary Trees
- N-ary Trees
- Binomial Trees
- Left-child Right-sibling Binary Trees

N-ary Trees

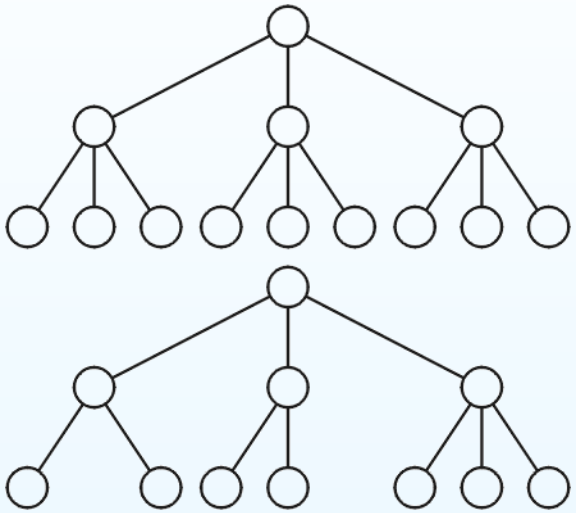
One generalization of binary trees are a class of trees termed N-ary trees

N-ary tree is a tree where each node has N subtrees, any of which may be empty trees

- i.e., each node has at most N subtrees

Ternary Trees

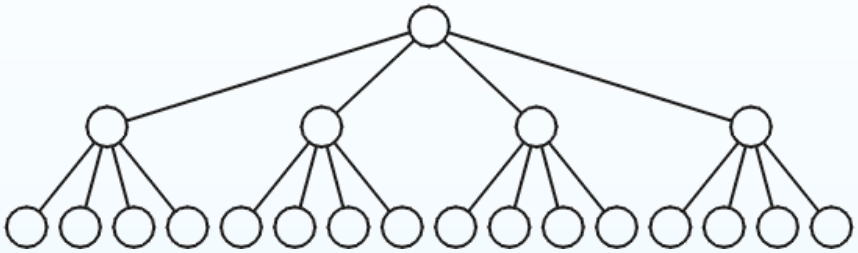
Examples of ternary (3-ary) trees



Note: we don't usually indicate empty subtrees

Quaternary Trees

Example of a perfect quaternary (4-ary) tree



Perfect N-ary Trees

- Each node has N children
- The number of nodes in a perfect N-ary tree of height h is

$$1 + N + N^2 + N^3 + \cdots + N^h$$

This is a geometric sum, and therefore, the number of nodes is

$$n = \sum_{k=0}^{h} N^k = \frac{N^{h+1} - 1}{N - 1}$$

Perfect N-ary Trees

- Solving the equation for h, a perfect N-ary tree with n nodes has a height given by

$$h = \log_N(n(N-1)+1) - 1$$

- We note that $\log_N n = \log_N n + \log_N N - 1 = \log_N(nN) - 1$

Hence,

$$\lim_{n\to\infty} \frac{\log_N(n(N-1)+1)-1}{\log_N n} = \lim_{n\to\infty} \frac{\frac{N-1}{(n(N-1)+1)\ln N}}{\frac{1}{n\ln N}}$$

$$= \lim_{n\to\infty} \frac{n(N-1)}{nN-n+1} = \lim_{n\to\infty} \frac{N-1}{N-1} = 1$$

โดยเทียบ

- The height can be approximated to $\Theta(\log_N n)$

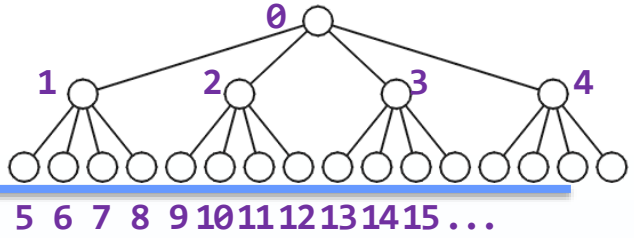
Perfect N-ary Trees

- Recall that the height for perfect binary tree is $\log_2 n$
- If the height of an N-ary tree is $\log_N n$, then the ratio of heights is:

$$\frac{\log_2 n}{\log_N n} = \frac{\log_2 n}{\frac{\log_2 n}{\log_2 N}} = \log_2 N$$

- Therefore,
  - The height of a perfect quaternary tree is approximately ½ the height of a perfect binary tree with the same number of nodes
  - The height of a perfect 16-ary tree is approximately ¼ the height of a perfect binary tree with the same number of nodes

Complete N-ary Trees

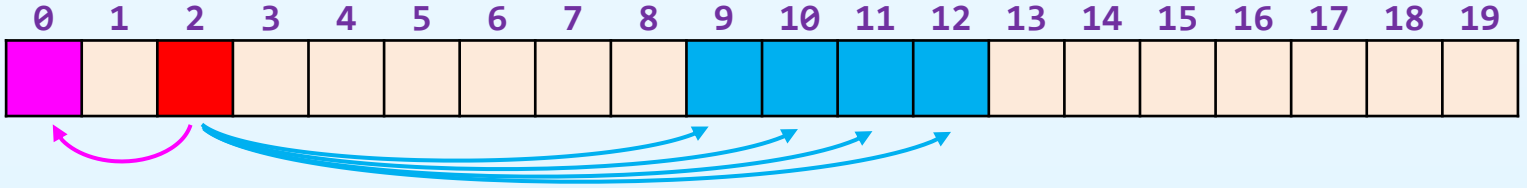


- A complete N-ary tree has height

$$h = \lfloor \log_N((N-1)n) \rfloor$$

- Like complete binary trees, complete N-ary trees can also be stored efficiently using an array:
  - The root is at index 0
  - The parent of a node with index k is at location $\left\lfloor \frac{k-1}{N} \right\rfloor$
  - The children of a node with index k are at locations $kN + j$ for $j = 1, 2, 3, \cdots, N$

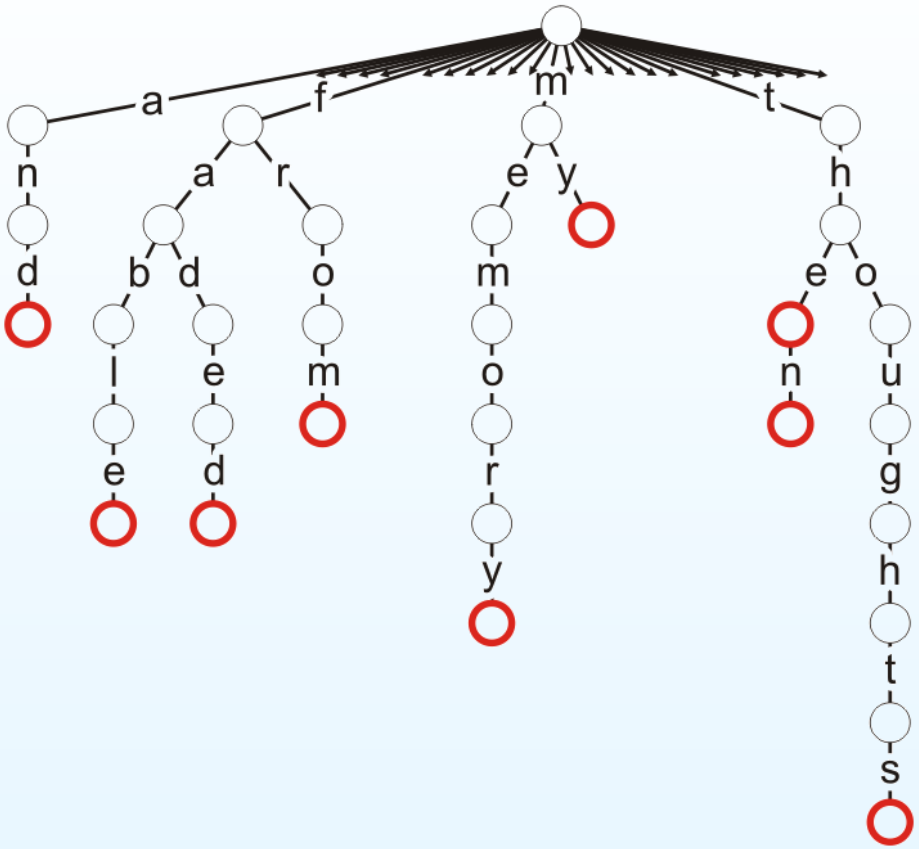
for an 4-ary tree,

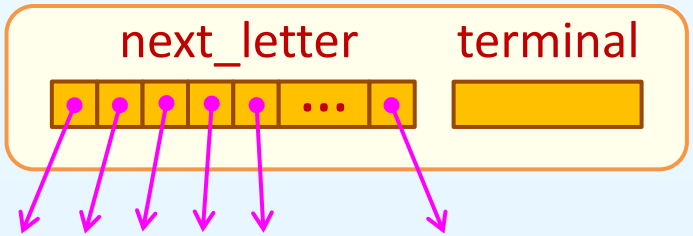


Application: Trie

A trie is an 26-ary tree, where

- Root represents the start of each valid word, and different subtrees represent next letters
- All nodes have 26 subtrees
- Some nodes are marked as terminal indicating the end of a valid word





"The fable then faded from my thoughts and memory"

Outline

- Binary Trees
- Perfect Binary Trees
- Complete Binary Trees
- N-ary Trees
- Binomial Trees
- Left-child Right-sibling Binary Trees

Binomial Trees

A binomial tree is an alternate optimal tree to a perfect binary tree

Recursive definition:

- A single node is a binomial tree of order $h = 0$
- A binomial tree B of order h is two binomial trees B_1 and B_2 of order $h - 1$ where
  - The root of B_1 is the root of B, and
  - B_2 is a subtree of B_1

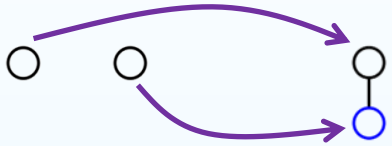
Recursive Definition

Thus, we start with a binomial tree of order 0

Recursive Definition

A binomial tree of order $h = 1$ is

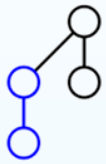
- A binomial tree of order 0 with a subtree (binomial tree) of order 0



Recursive Definition

A binomial tree of order $h = 2$ is

- A binomial tree of order 1 with a subtree of order 1
- Here, we have 4 nodes

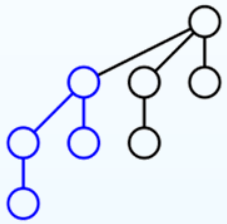


Like perfect trees, binomial trees are not defined for some number of nodes such as three nodes

Recursive Definition

A binomial tree of order $h = 3$ is

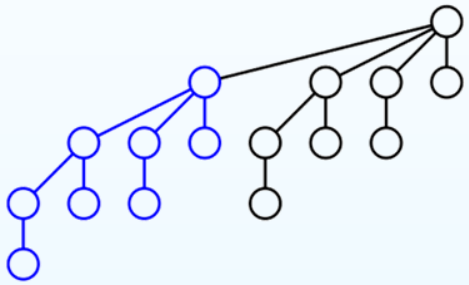
- A binomial tree of order 2 with a subtree of order 2
- Here, we have 8 nodes



Recursive Definition

A binomial tree of order $h = 4$ is

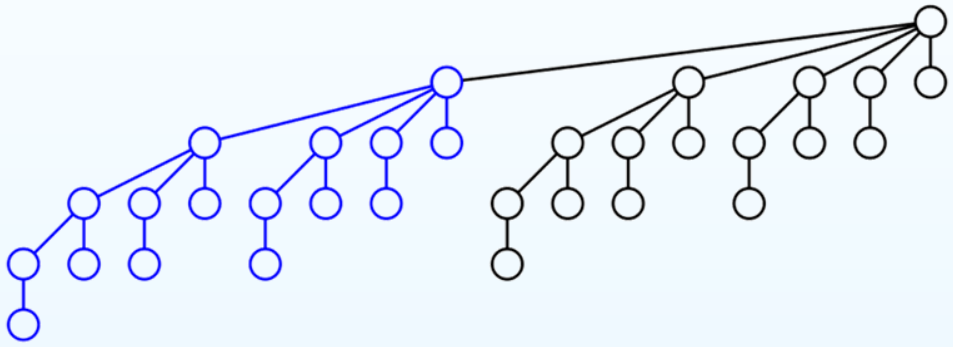
- A binomial tree of order 3 with a subtree of order 3
- Here, we have 16 nodes



Recursive Definition

A binomial tree of order $h = 5$ is

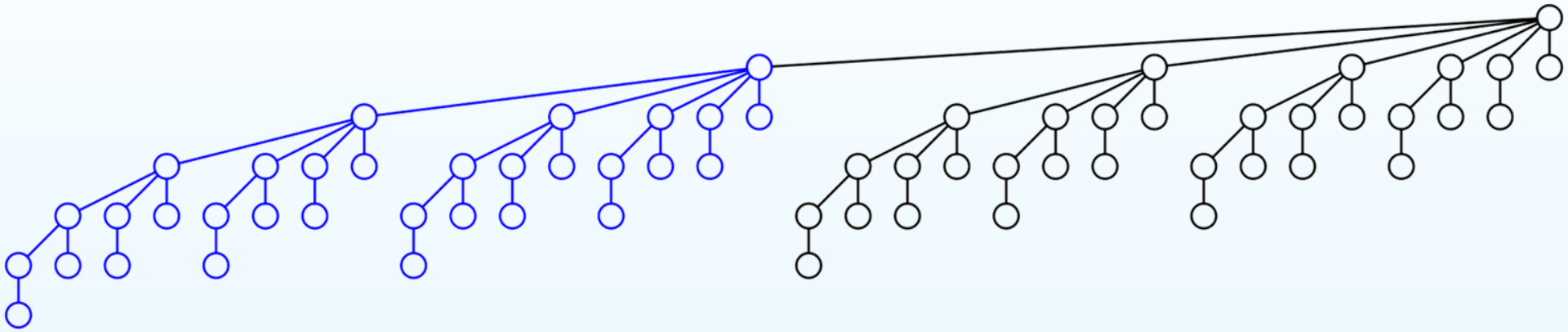
- A binomial tree of order 4 with a subtree of order 4
- Here, we have 32 nodes



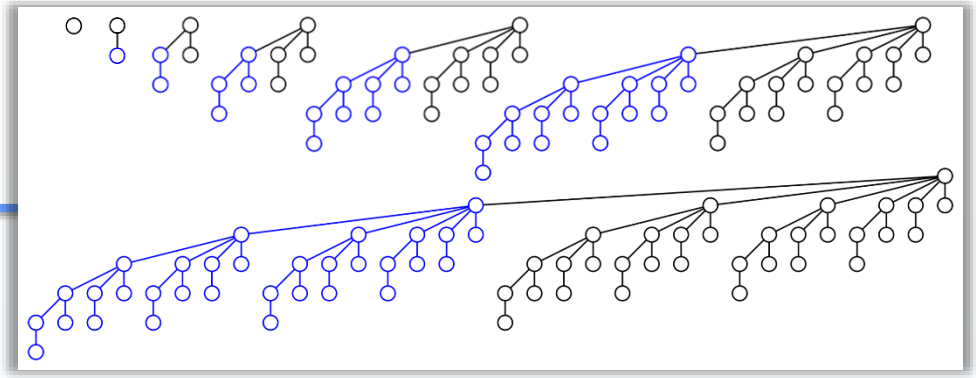
Recursive Definition

As a last example, a binomial tree of order $h = 6$ is

- A binomial tree of order 5 with a subtree of order 5
- Here, we have 64 nodes



Various Theorems



1. A binomial tree of order h has height h
2. A binomial tree of height h has 2^h nodes: $1, 2, 4, 8, 16, 32, \ldots$

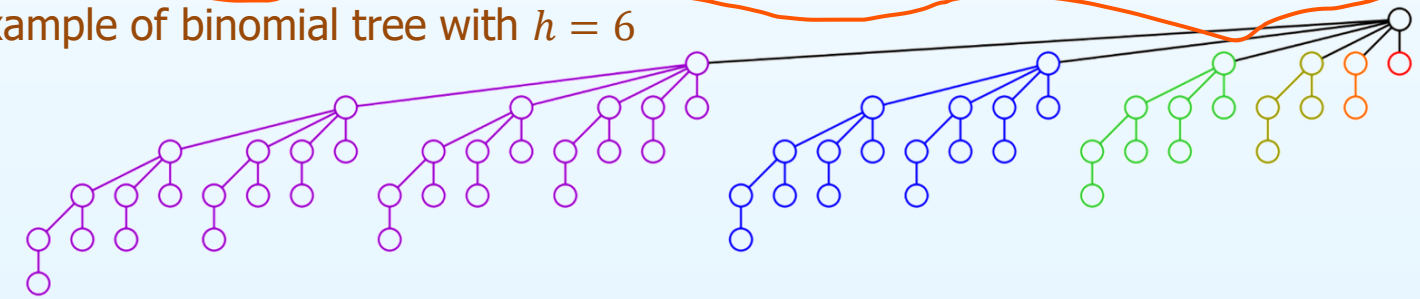
   Corollary: A binomial tree with n nodes has a height of $\log n$

3. A binomial tree of height h has $\binom{h}{k}$ nodes at depth k

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}
$$

4. (Alternative definition) a binomial tree of height h is a root node and has h subtrees which are binomial trees of heights $0, 1, 2, \ldots, h-1$

   Example of binomial tree with $h = 6$



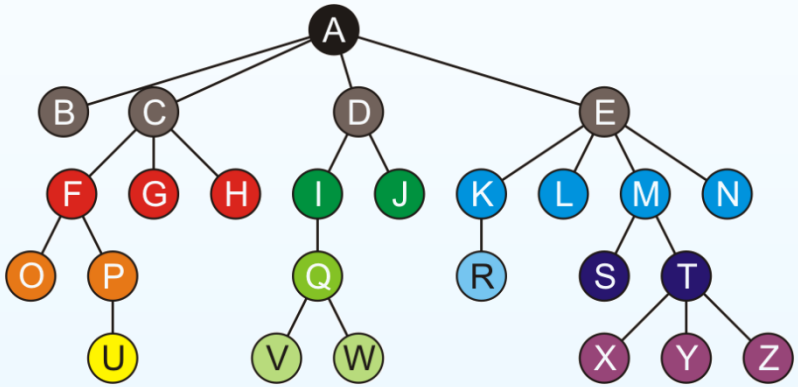
5. The degree of a root of a binomial tree of height h is h (or $\log n$)
6. The root has the maximum degree of any node in a binomial tree

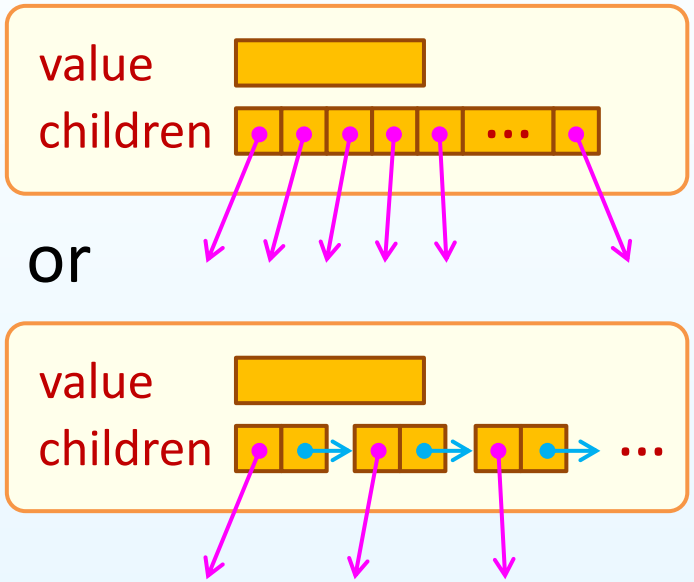
Outline

- Binary Trees
- Perfect Binary Trees
- Complete Binary Trees
- N-ary Trees
- Binomial Trees
- Left-child Right-sibling Binary Trees

Motivation

For any general trees, a simple tree data structure is node-based where children are stored as a linked list





Is it possible to store a general tree as a binary tree?

➡ Yes! You have seen this implementation already

The Idea

Consider the following

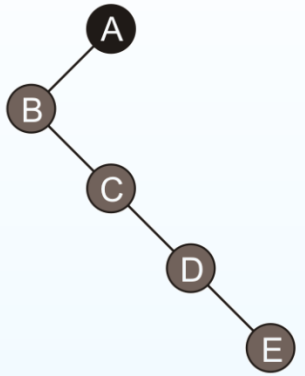
- The first child of each node is its left subtree
- The next sibling of each node is in its right subtree

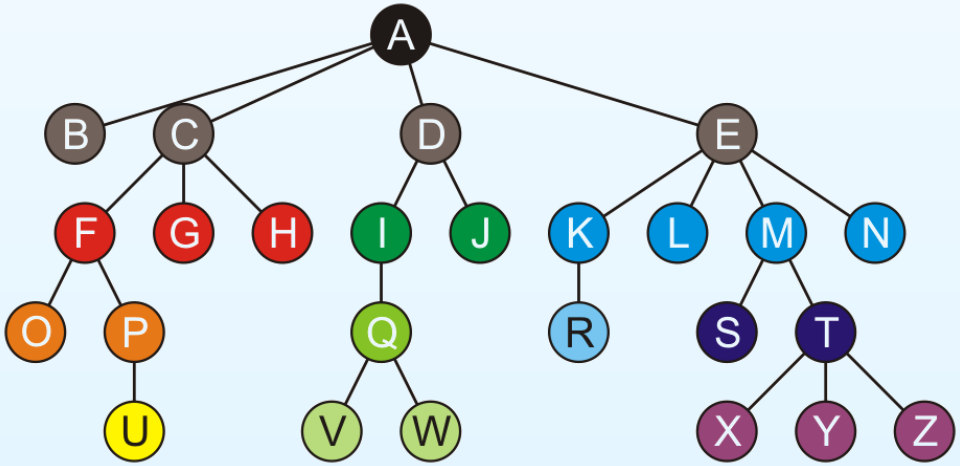
This is called a left-child right-sibling binary tree

Example

Consider this general tree

- B is the first child of A, so it is the left child
- For the three siblings C, D, and E:
  - C is the right subtree of B
  - D is the right subtree of C
  - E is the right subtree of D

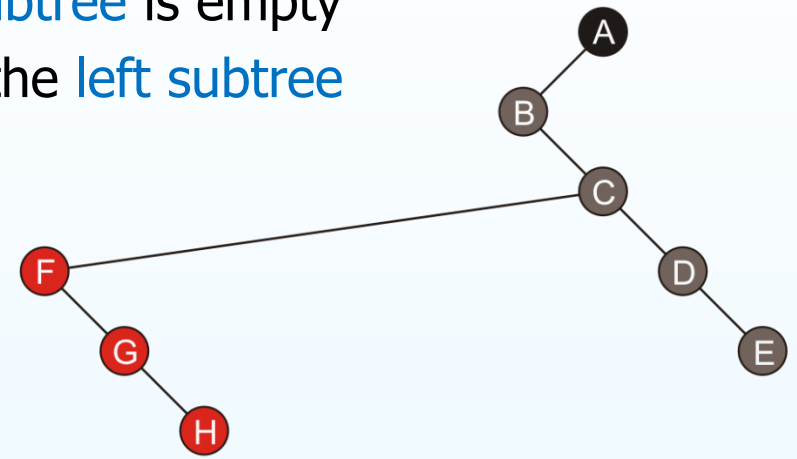


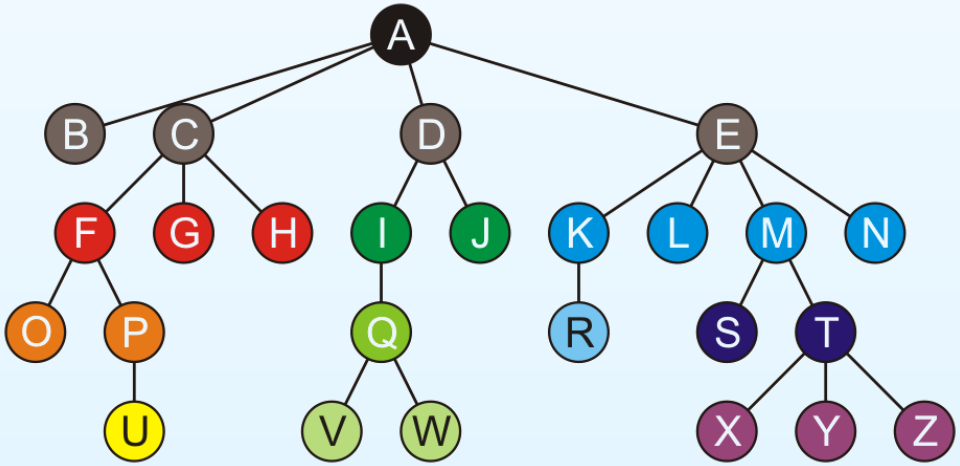


Example

Consider this general tree

- B has no child, so its the left subtree is empty
- F is the first child of C, so it is the left subtree
- For the next two siblings:
  - G is the right subtree of F
  - H is the right subtree of G

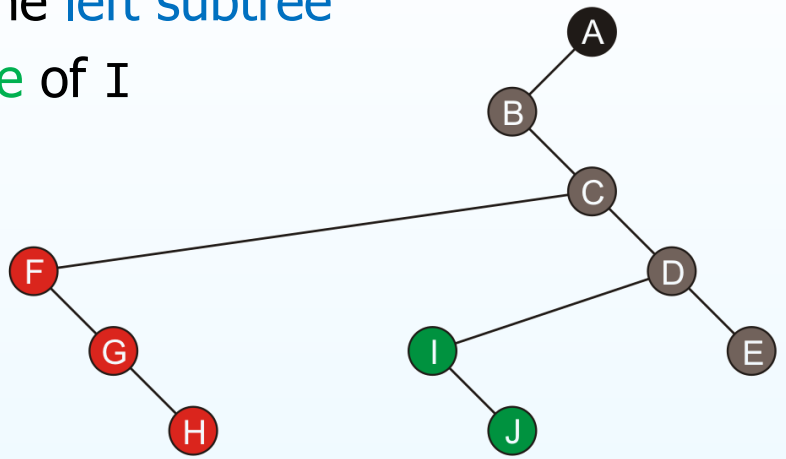


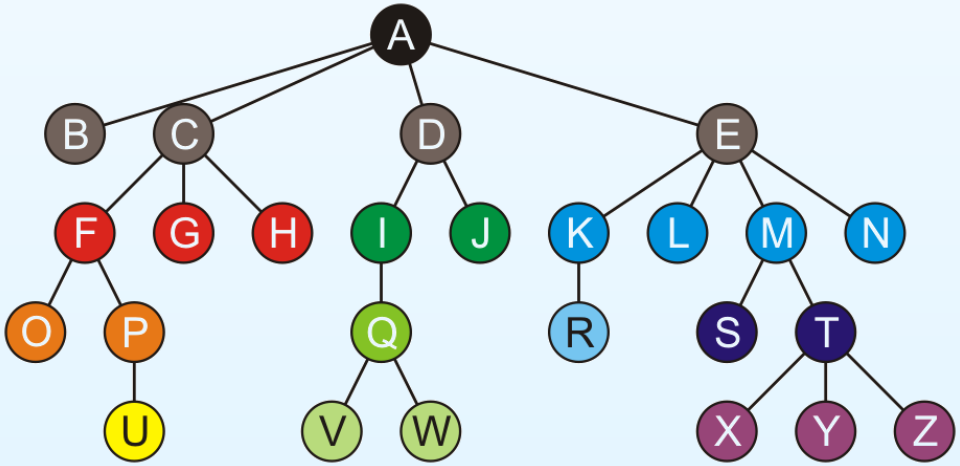


Example

Consider this general tree

- `I` is the first child of `C`, so it is the left subtree
- Its sibling, `J`, is the right subtree of `I`

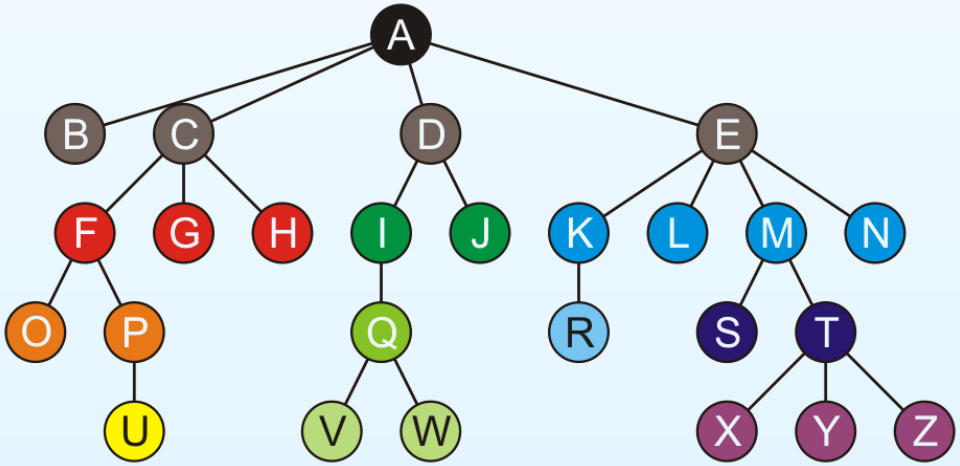


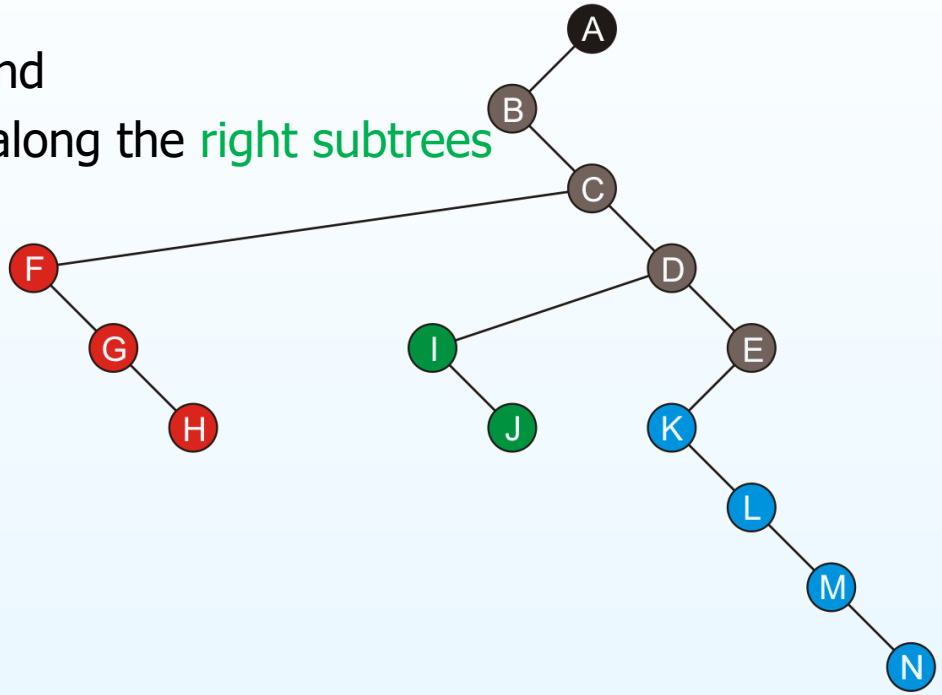


Example

Consider this general tree

- Similarly, the four children of `E`:
  - `K` forms the left subtree of `E`, and
  - Its three siblings form a chain along the right subtrees

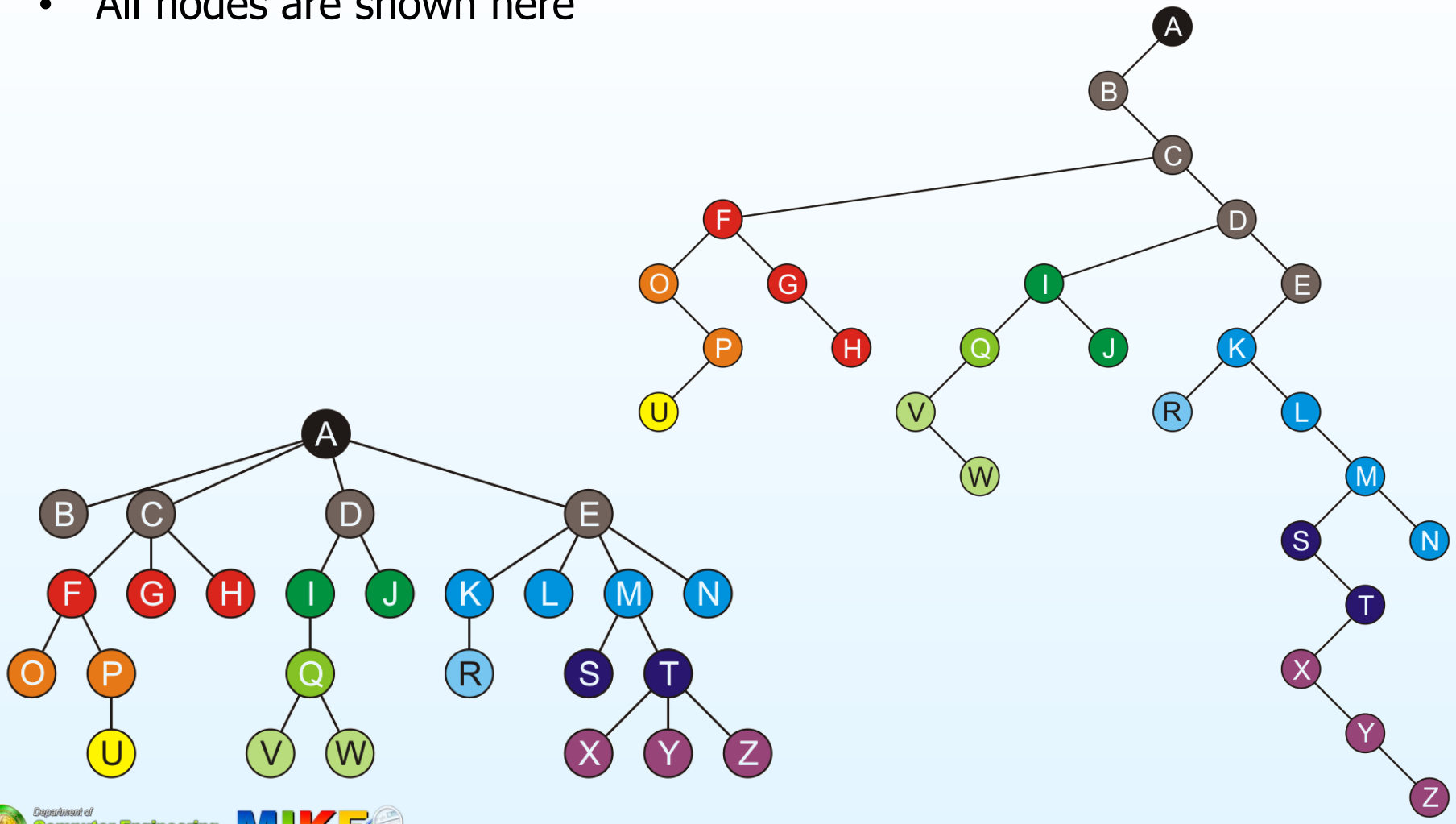




Example

Consider this general tree

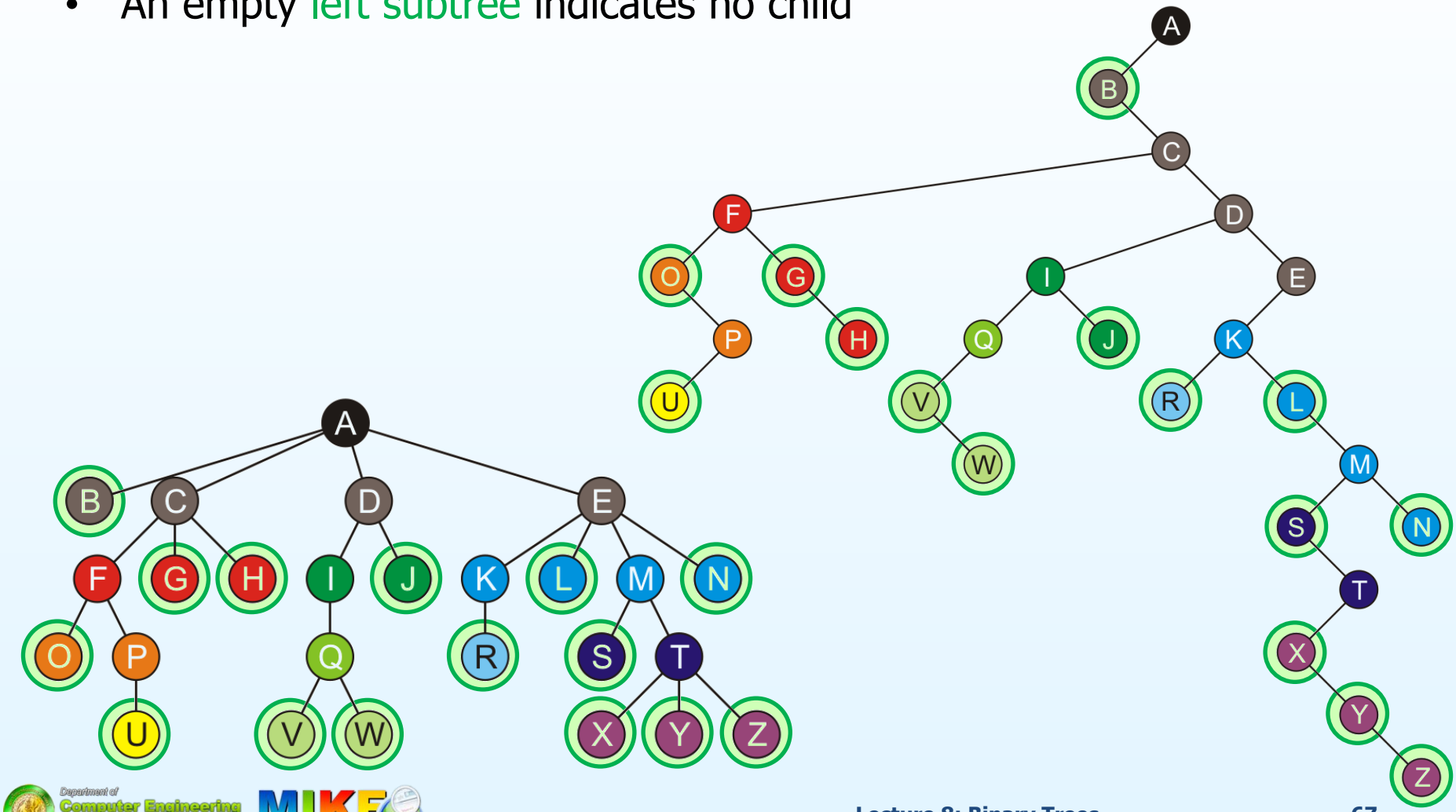
- All nodes are shown here



Example

Notice that

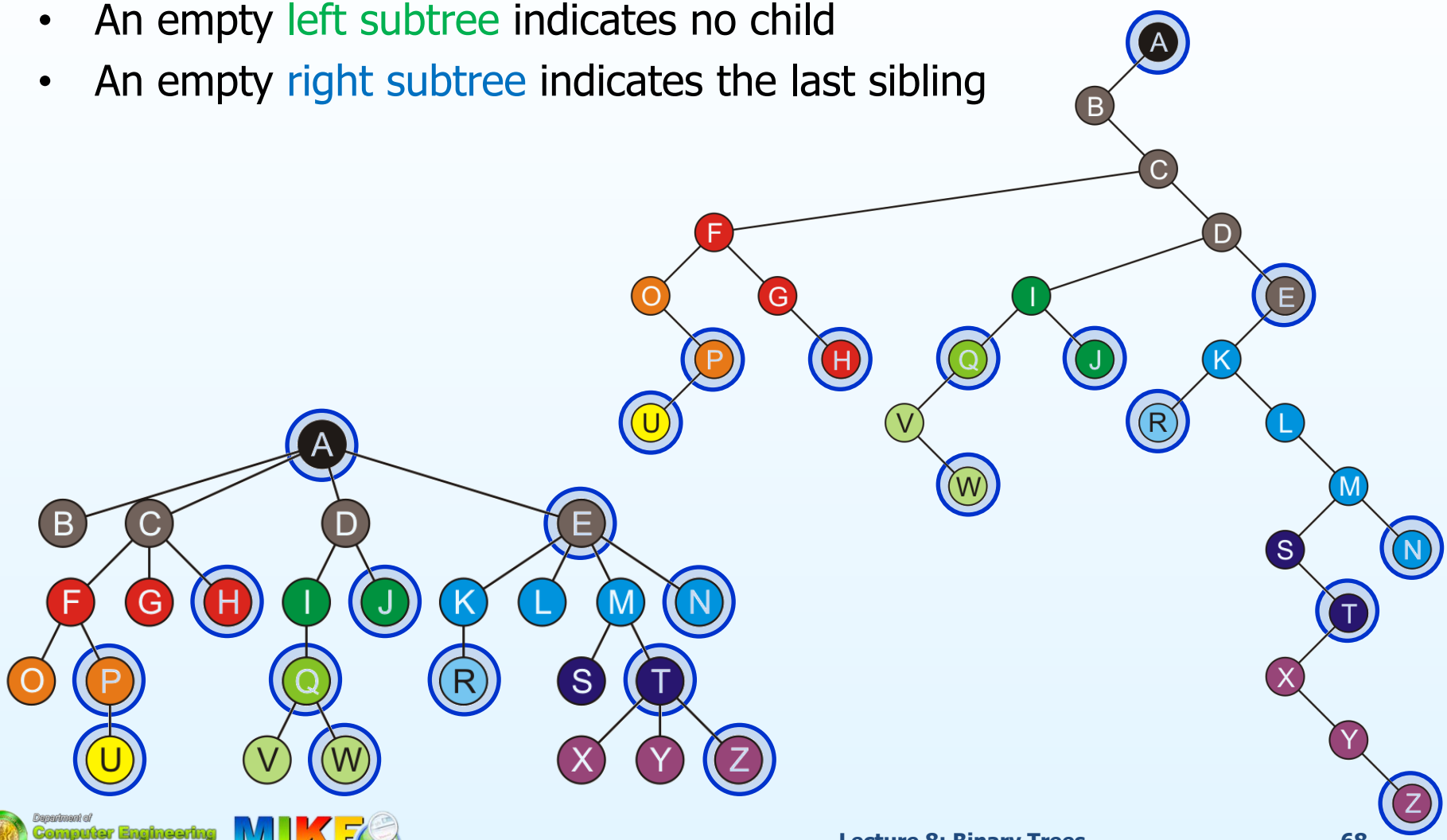
- An empty left subtree indicates no child



Example

Notice that

- An empty left subtree indicates no child
- An empty right subtree indicates the last sibling



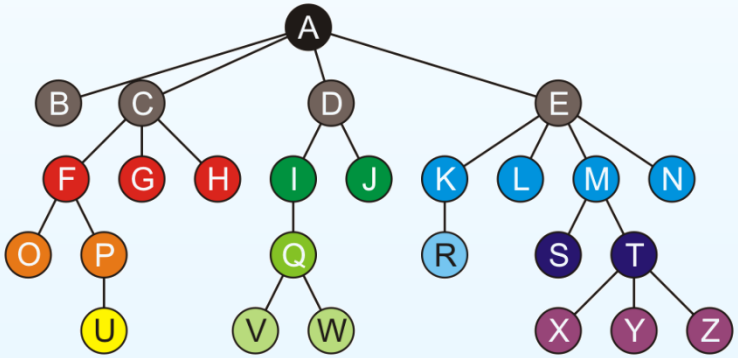
Transformation

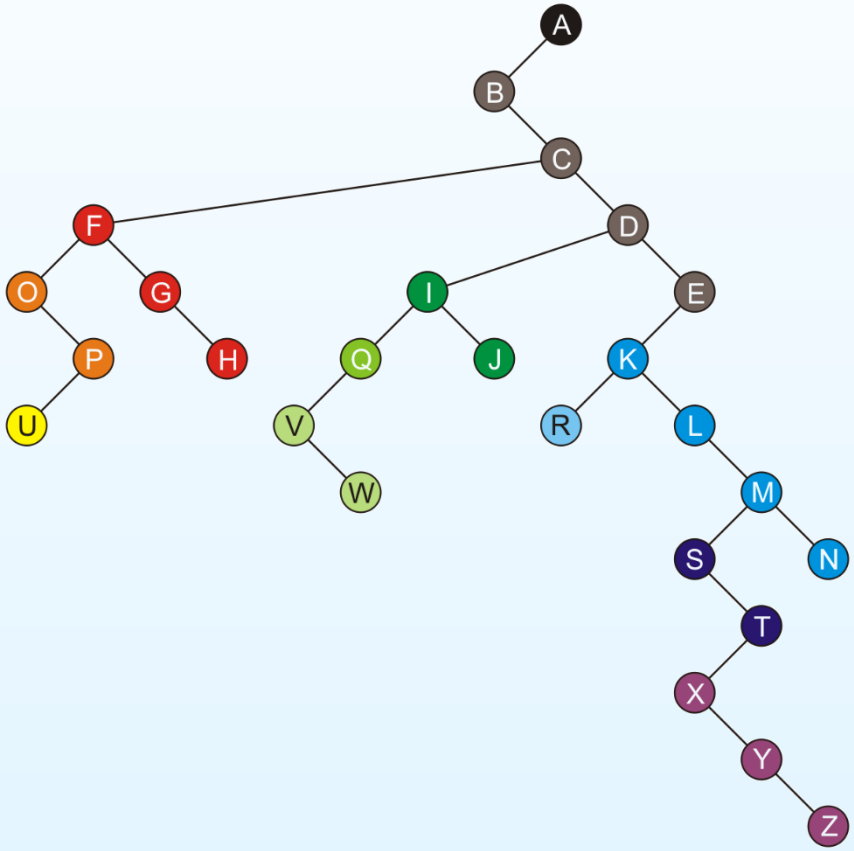
The transformation of a general tree into a left-child right-sibling binary tree has been called Knuth transform

Traversals

A pre-order traversal on both the original tree and the Knuth transform is identical

A B C F O P U G H D I Q V W J E K R L M S T X Y Z N

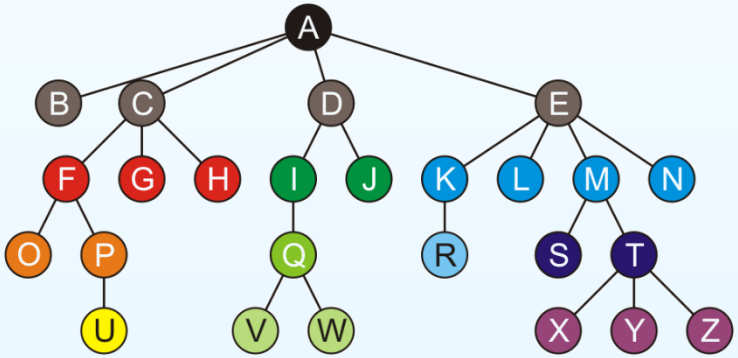


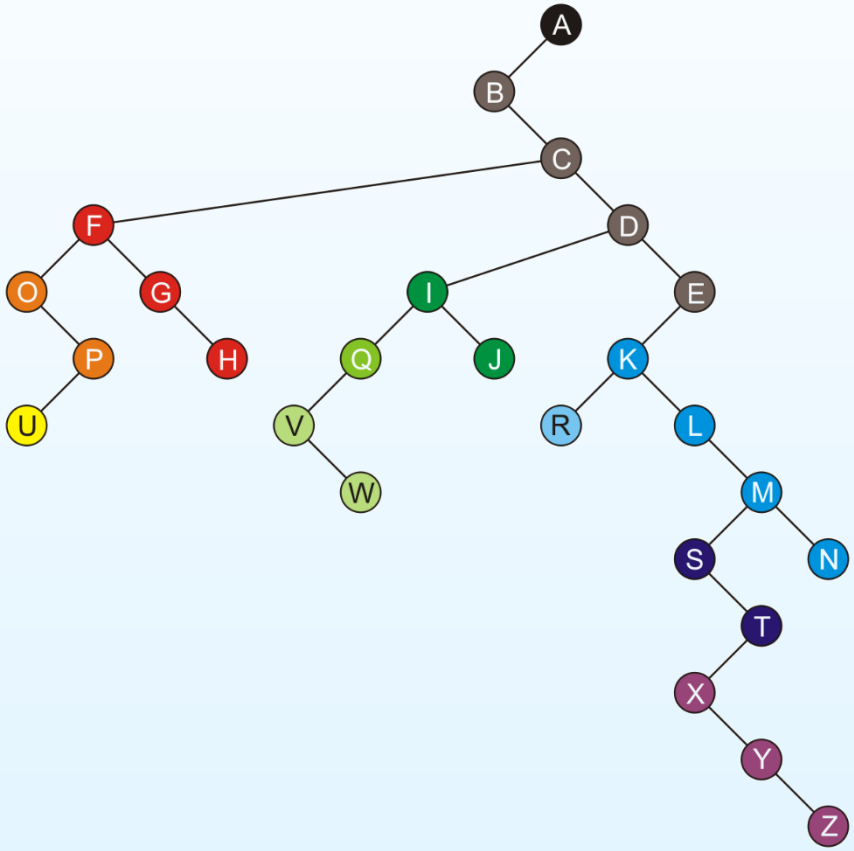


Traversals

A post-order traversal on the original tree can be achieved by an in-order traversal on the Knuth transform

B O U P F G H C V W Q I J D R K L S X Y Z T M N E A



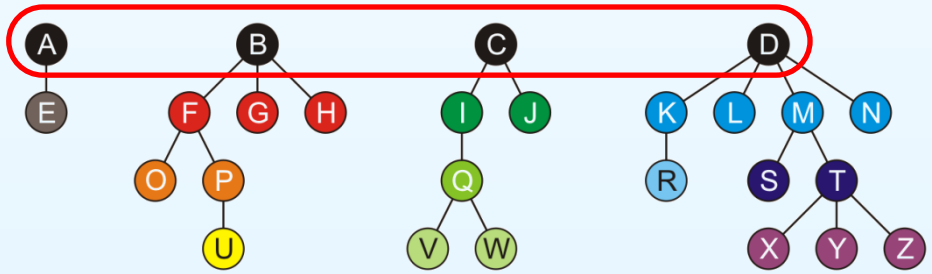


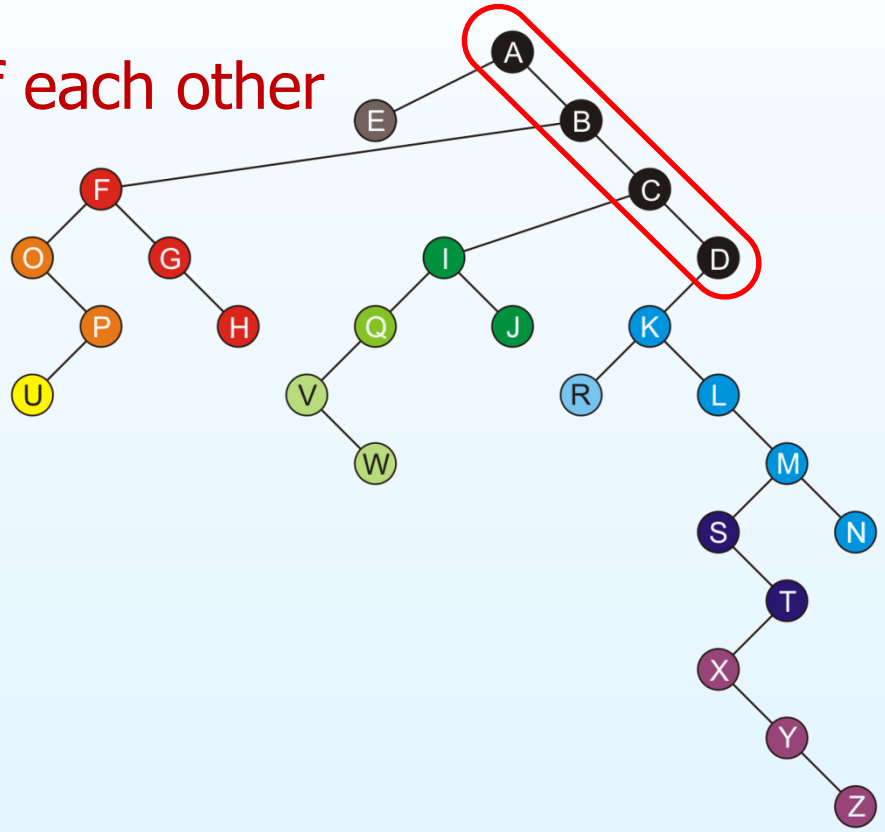
Forests

A forest can be stored in this representation as follows:

- Choose one of the roots of the trees as the root of the binary tree
- Let each subsequent root of a tree be a right child of the previous root

Think of the roots as siblings of each other





Any Question?