# COL106 - Data Structures and Algorithms

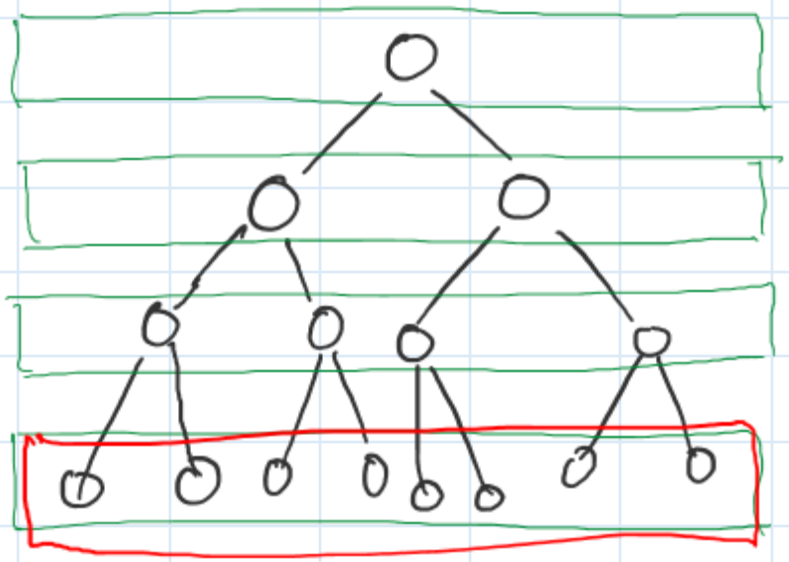
# BINARY TREES

- ORDERED TREES
- EACH NODE HAS **AT MOST** TWO CHILDREN : **LEFT, RIGHT**
- LEFT PRECEDES RIGHT.

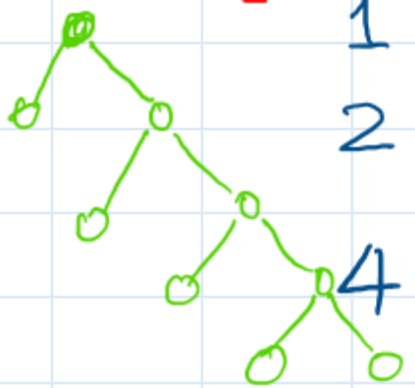
A PROPER BINARY TREE IS THE ONE WHERE EVERY NODE HAS **0 OR 2** CHILDREN

BINARY TREES ARE VERY POPULAR -

# PROPERTIES OF BINARY TREES




MAX # OF NODES




$h \leq \frac{n-1}{2}$

level 2

h upperbounded by n-1

$n$ = # of nodes

$e$ = # of leaf nodes

$i$ = # of internal -"-

$h$ = the height of tree

If T is proper

(l) $h+1 \leq n \leq 2^{h+1}-1$

$2h+1 \leq n \leq 2^{h+1}-1$

$1 \leq e \leq 2^h$

$h \leq i \leq 2^h - 1$

$e = i + 1$

# $e = i + 1$ IN A PROPER BINARY TREE

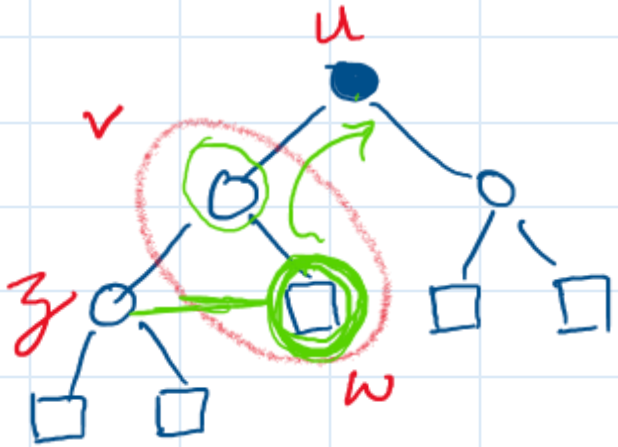
SPLIT TREE INTO TWO PILES
INTERNAL & EXTERNAL

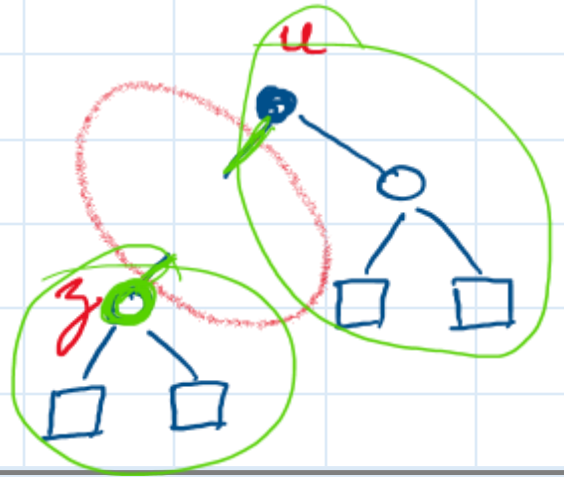
**CASE 1:**

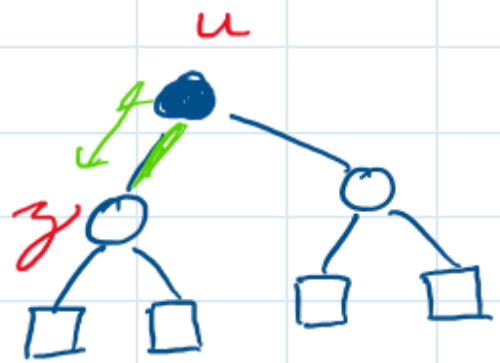
If tree T has only one node V
then put it in external pile

**CASE 2:**

Repeat the process of deleting a leaf node and its parent, reconnecting to get a full binary tree. At the end you are left with ...?



Any tree with $n$ nodes, has exactly $n-1$ edges

(*) For $n = 1$, the statement holds

(*) For some $k$, say it holds. ($n =$)

(*) Now consider a tree with $k+1$ nodes ($n =$)

There has to be a leaf node.
remove the leaf node and the edge connecting

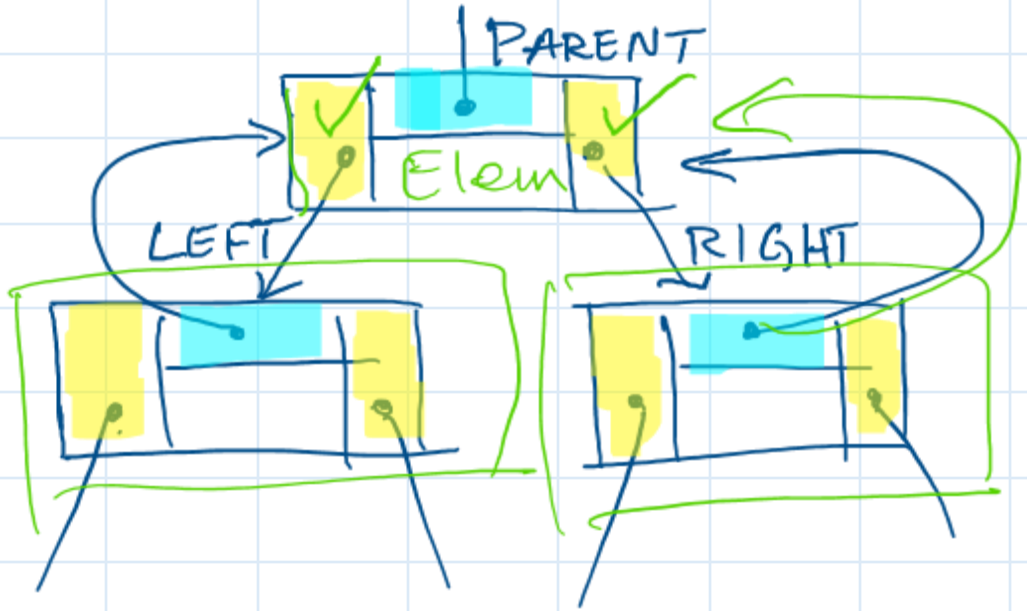
(k-1) edge

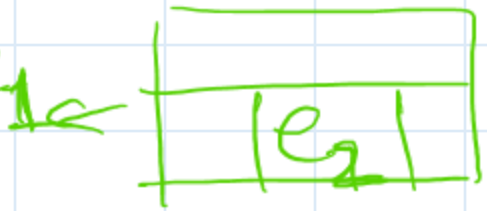
$\Rightarrow$ now we have $k$ nodes and one less edge.

adding back $= (k-1) + 1 = k$ edges

$k+1$ nodes

# IMPLEMENTING BINARY TREES.




TREE ADT had only accessor, query, iterator methods

- ✓ addRoot(e)
- addLeft(p, e)
- addRight(p, e)
  - → set(p, e) left
- ⇒ attach(p, $T_1$, $T_2$)
  - remove(p) → right

methods that modify the binary tree

Show that each of these methods can be implemented to run in O(1).

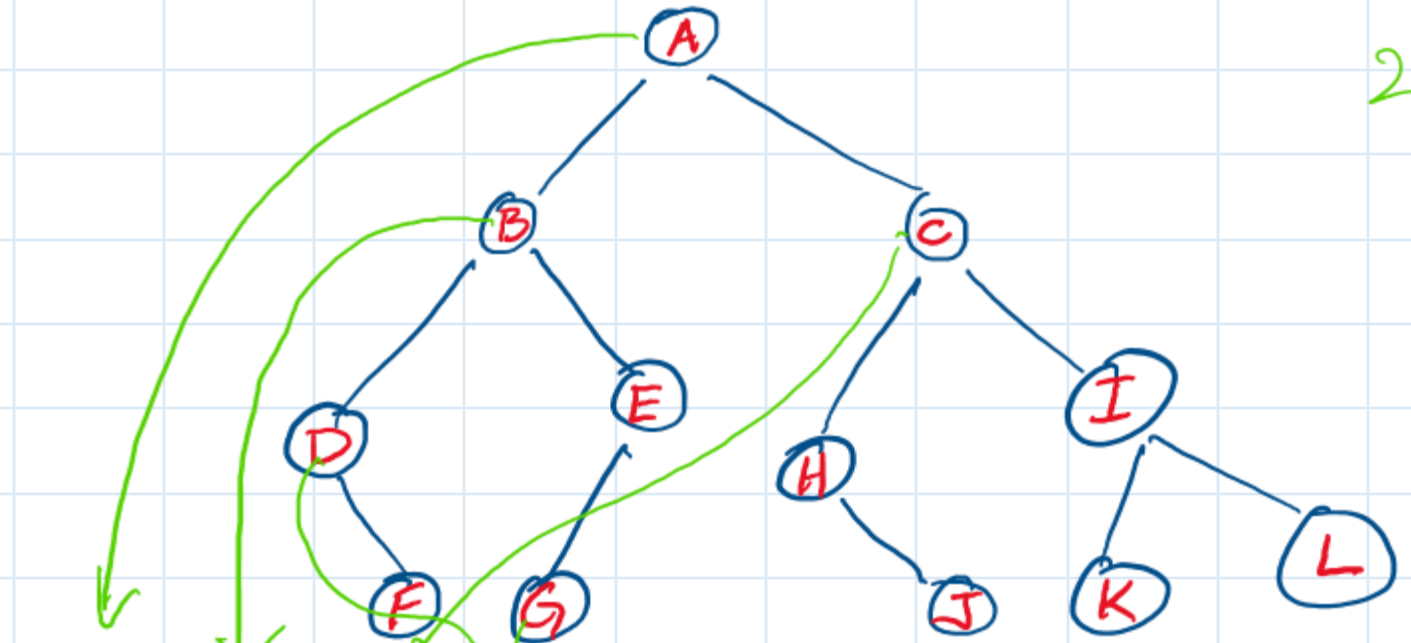
# Implementing Binary Tree with Array

i) If node v is the root of the tree then store it in index $f(v) = 0$.

ii) If node u is the left child of v then store it in index $f(u) = 2f(v) + 1$

iii) If node w is the right child of v then store it in index $f(w) = 2f(v) + 2$

# Implementing Binary Tree with Array




$2f(A)+1$

$0+1$

$f(D) = 2f(B)+1$

$= 2 \times 1 + 1$

$= 3$

$f(F) = 2 \times 3 + 2$

| A | B | C | D | | | | | F | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | . . . |

Establish the complexity of all Binary Tree methods

# Space Usage

Depends on the "shape" of the binary tree

The total array size is $1 + \max(f(p))$ over all nodes in the tree.

⇒ Note that there can be many empty array slots

What is the worst-case size of the array for representing a tree with $n$ nodes?

$2^n - 1$