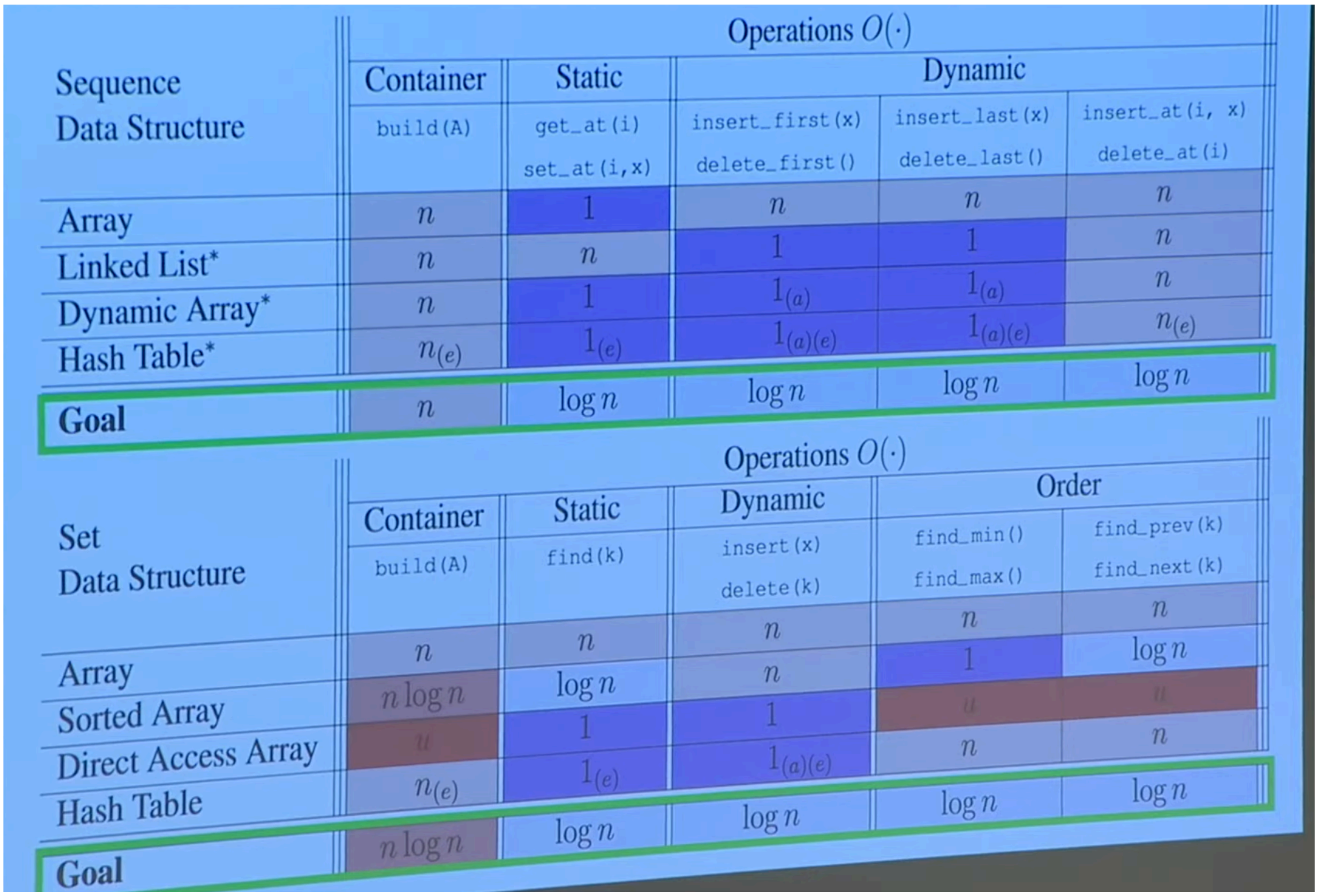
Lec6 Binary Tree 1

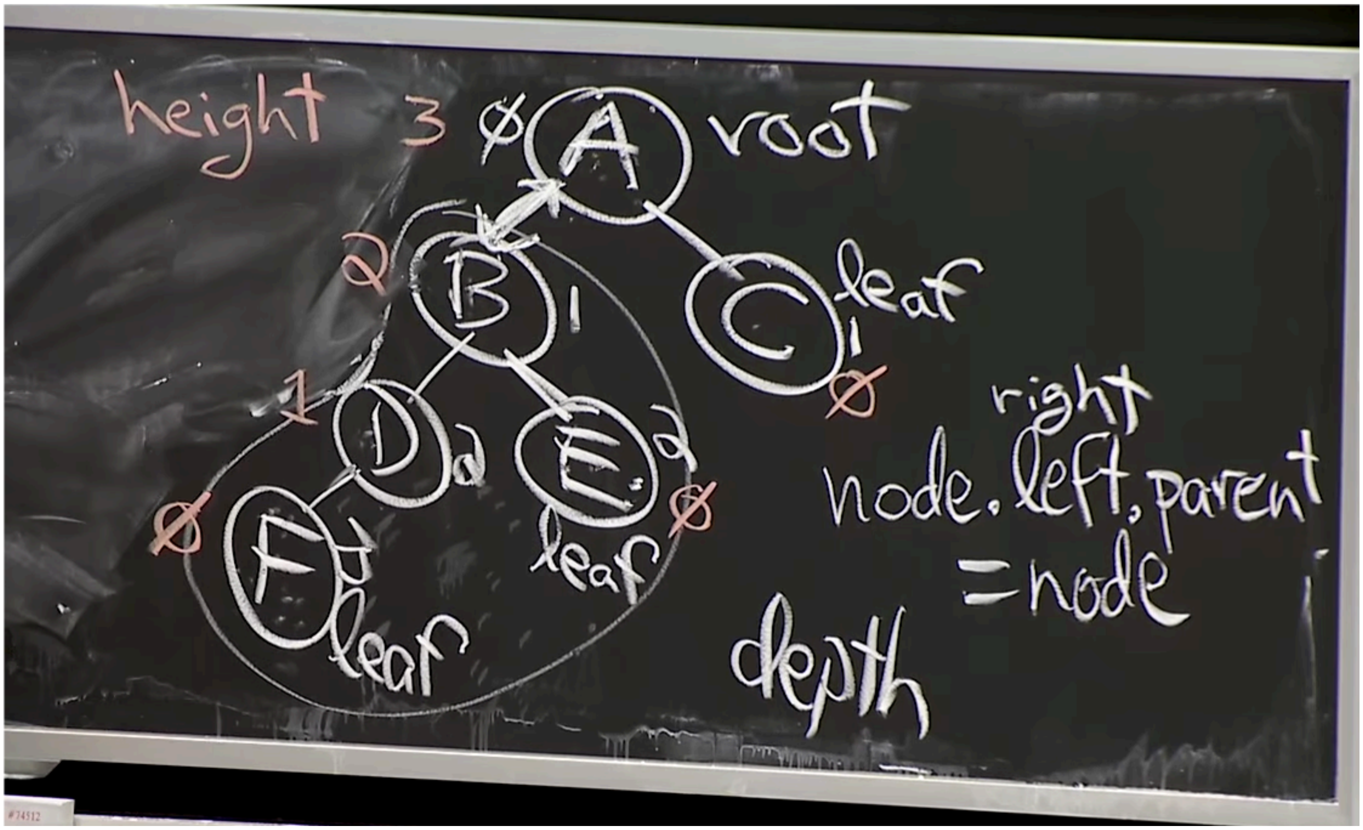
3.21 https://github.com/GUMI-21/MIT6.006_note
This lec should read Recitation meanwhile.



Sequence Data Structure	Operations $O(\cdot)$				
	Container	Static	Dynamic		
	`build(A)`	`get_at(i)` `set_at(i,x)`	`insert_first(x)` `delete_first()`	`insert_last(x)` `delete_last()`	`insert_at(i, x)` `delete_at(i)`
Array	n	1	n	n	n
Linked List*	n	n	1	1	n
Dynamic Array*	n	1	$1_{(a)}$	$1_{(a)}$	n
Hash Table*	$n_{(e)}$	$1_{(e)}$	$1_{(a)(e)}$	$1_{(a)(e)}$	$n_{(e)}$
Goal	n	$\log n$	$\log n$	$\log n$	$\log n$

Set Data Structure	Operations $O(\cdot)$				
	Container	Static	Dynamic	Order	
	`build(A)`	`find(k)`	`insert(x)` `delete(k)`	`find_min()` `find_max()`	`find_prev(k)` `find_next(k)`
Array	n	n	n	n	n
Sorted Array	$n \log n$	$\log n$	n	1	$\log n$
Direct Access Array	u	1	1	u	u
Hash Table	$n_{(e)}$	$1_{(e)}$	$1_{(a)(e)}$	n	n
Goal	$n \log n$	$\log n$	$\log n$	$\log n$	$\log n$

Binary Tree



A node have a parent pointer and left/right pointer and self item.

- EX
  node A B C D E F
  item A B C D E F
  parent / A A B B D
  left B D ...
  right C E ...
  node.left.parent = node
  leaf

height of tree

- $subtree(\otimes)$
  x & its descendants (x root)
- $depth(x)$
  number of ancestores = number of edges in path from x *up* to root
- $height(\otimes)$
  number of edges in langest downward path from x.= max depth in subtree(x)
  h = height(root)=height(tree)
  Today: O(h) opearations

Traversal order of nodes/items

trav. order of example

F-D-B-E-*A*-C

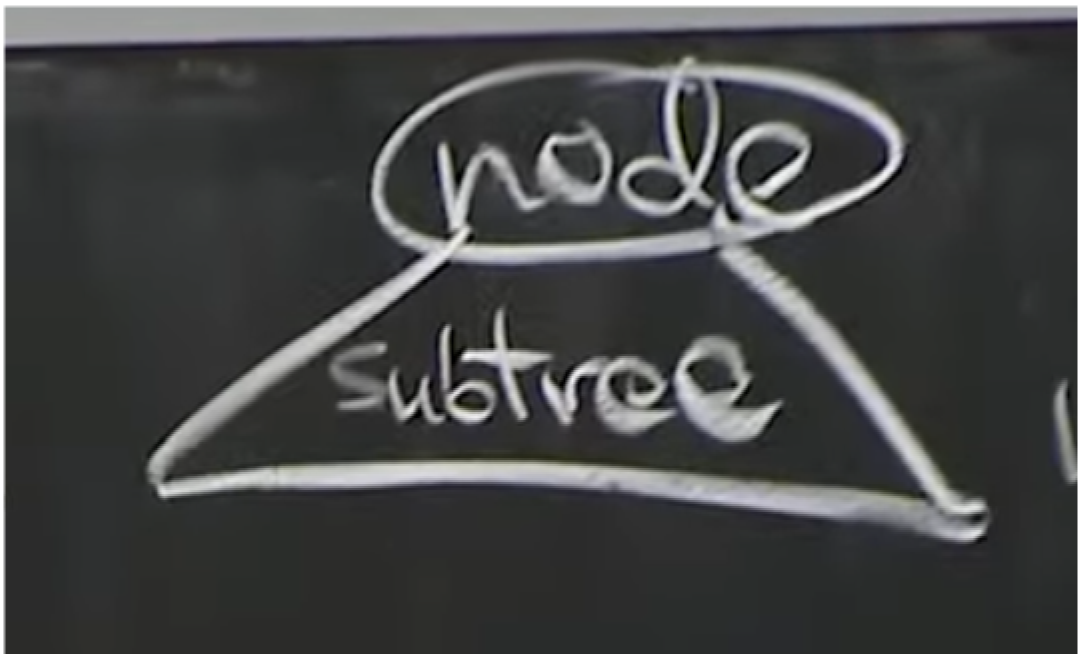
for every node x, -nodes in x.left befer x, x.right after x

*also callde **in-order traversal** order*

```
iter(x):
    iter(x.left)
    output(x)
    iter(x.right)
```

Traversal ops

- subtree_first(node):

  which comes first in traversal order within subtree.

  follow nodes of in-order traversal steps

```
1. from given x go left (node = node.left) until would fall off tree (node = Noe)
2. return node
3. find successor(node): next after node in tree's traversal order
-if node.right: return subtree_first(node.right)
-else: walk up tree (node = node.parent)
     until go up a left branch (node == node.parent.left)
     -return node
```

runtime: O(h)

subtree_insert_agter(node.new):

in the traversal order.
means: ....node ^(new) .....

- *algorithm*
  -if no node.right: put new there.
  -if node.right: put new as successor(node)left *successor fined from subtree_first()*
- in upper example:
  insert G before E: G.left = E, E.parent = G
  Insert H after A: put in C.left
- Runtime: O(h)

subtree_delete(node):

- *algorithm*
  -if node is leaf: detach from parent
  -else: if node.left: swap node.tem - predecessor(node).item, subtree_delete(predecessor).
  if node.right:
- in upper example
  delete F: derict erase
  delete A: presuccessor,

Sequence

travsal order = Sequence order
next tiem

Set BST* binary set tree

travsal order = increasing item.key

- find(k)/find-prev/find-next