CS101 Midterm Review

Linked List, Stack, Queue, Tree, Binary Tree

Midterm

- **Logistics**
  - **Time:** 11/02 Wednesday 8:15 AM ~ 9:55 AM
  - **Exam Classroom:** check your EAMS
- **Content**
  - 10 * True or False + 5 * Single Choice + 4 * Multiple Choices
  - 4 questions covering Complexity, Sort, Divide & Conquer, Heap, AVL…

1.
Array & Linked List

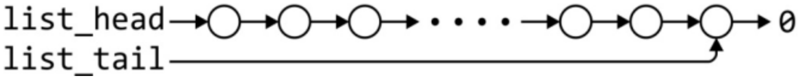
Array/Singly Linked List/Doubly Linked List

Array vs. Linked List

- **Abstract List (List ADT)**
  - **Find k-th** Array *O(1)* vs. Linked List *O(n)*
  - **Insert** Array *O(n)* vs. Linked List *O(1)**
  - **Delete** Array *O(n)* vs. Linked List *O(1)**

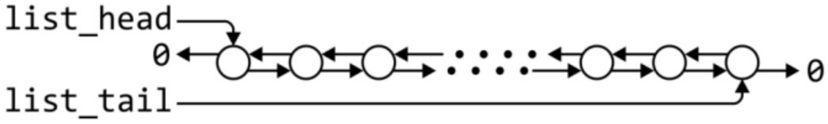
**Assume doubly linked list and the k-th is already accessed*

Singly vs. Doubly Linked List



	Front/1st node	k^{th} node	Back/n^{th} node
Find	$\Theta(1)$	$O(n)$	$\Theta(1)$
Insert Before	$\Theta(1)$	$O(n)$	$\Theta(n)$
Insert After	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Replace	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Erase	$\Theta(1)$	$O(n)$	$\Theta(n)$
Next	$\Theta(1)$	$\Theta(1)^*$	n/a
Previous	n/a	$O(n)$	$\Theta(n)$

		k^{th} node	Back/n^{th} node
		$O(n)$	$\Theta(1)$
		$\Theta(1)^*$	$\Theta(1)$
		$\Theta(1)^*$	$\Theta(1)$
		$\Theta(1)^*$	$\Theta(1)$
Erase	$\Theta(1)$	$\Theta(1)^*$	$\Theta(1)$
Next	$\Theta(1)$	$\Theta(1)^*$	n/a
Previous	n/a	$\Theta(1)^*$	$\Theta(1)$



Questions

(b) (2') Which of the following statements about arrays and linked-lists are true?

A. Gaining access to the k-th element in an array takes constant time.

B. Gaining access to the k-th element in a linked-list takes constant time.

C. Erasing the k-th element in an array takes constant time.

D. With access to the k-th element, inserting an element after the k-th element in a linked-list takes constant time.

(b) **(2 pt)** For a single linked array, erasing the element after the current pointer takes $O(1)$, and erasing the element pointed by the current pointer also takes $O(1)$.

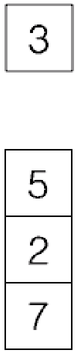
2.
Queue & Stack

Queue/Circular Queue/Stack

Queue vs. Stack

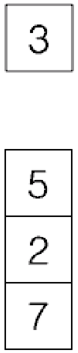
- **Linear Data Structure** push/pop
  - **Queue** First-In-First-Out
  - **Stack** Last-In-First-Out

Stack

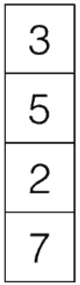


→ Push(3)

Stack



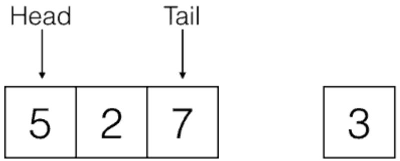
→ Push(3)



→ Pop()

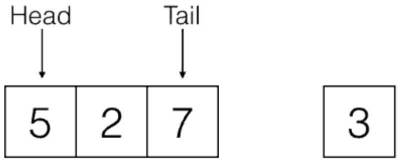
Queue

→ Push(3)

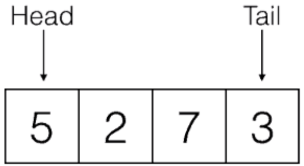


Queue

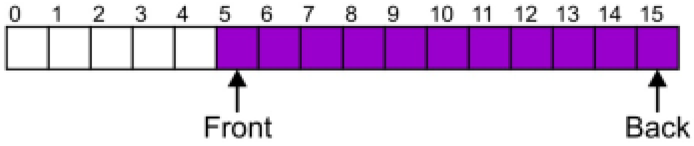
- Push(3)



- Pop()

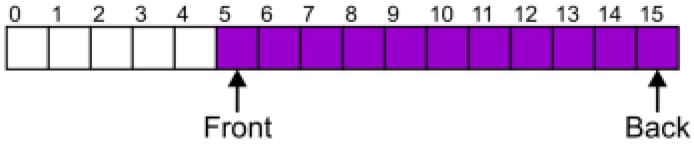


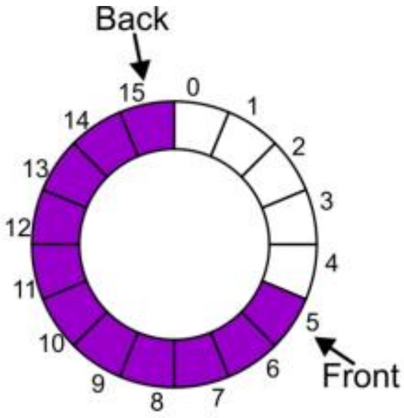
Two-ended vs. Circular Queue



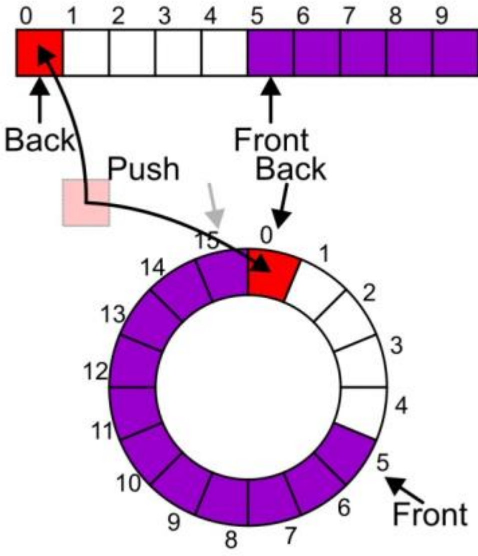
Problem?

Two-ended vs. Circular Queue





Two-ended vs. Circular Queue



Queue&Stack Implementation

- **Queue using Array**
  - maintaining *front* & *back*
  - Two-ended(naïve) / Circular
- **Stack using Array**
  - maintaining *top*
- **Queue/Stack using Linked List**
  - *(See your HW)*

3.
Hash Table

Open Addressing(Probing)/Chaining

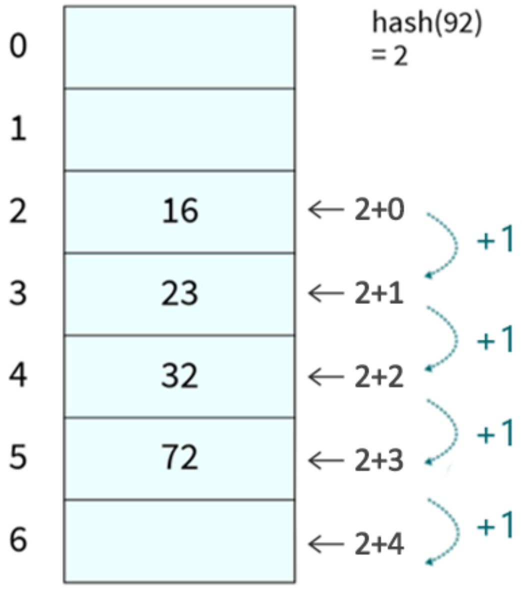
Hash Table

- **Hash Function**
  - $h(key) = \ldots \bmod M$ M: capacity of table
- **Load Factor**
  - $\lambda = n/M$ n: number of elements inserted

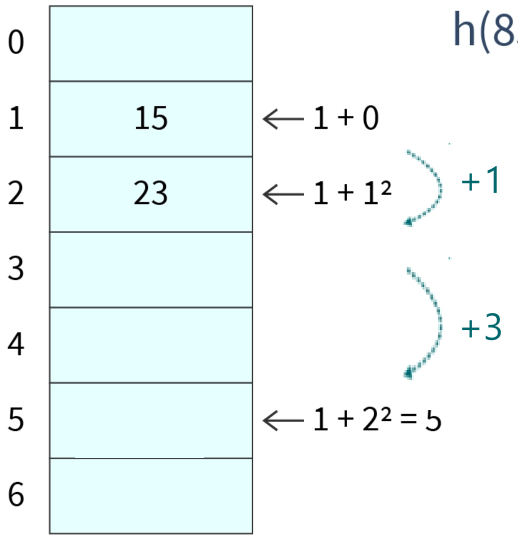
Resolving Conflicts in Hash Table

- **Open Addressing**
  - Linear/Quadratic Probing
  - Double Hashing/… *(See your HW)*
- **Chaining**
  - Using Linked Lists

Linear Probing

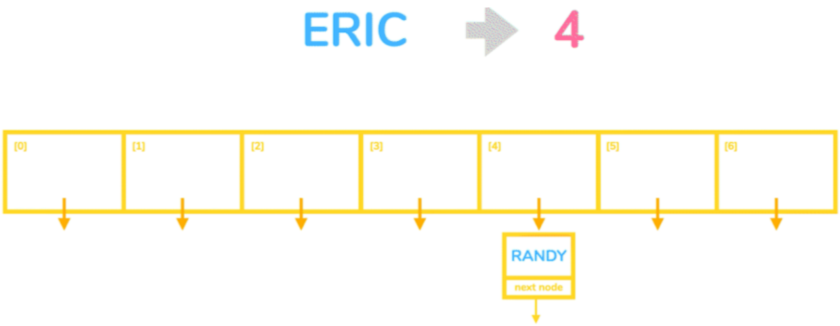


Quadratic Probing



$h(85) = 1$

Chaining



Questions

(b) **(3 pt)** You are given an open-addressing hash table with m slots and we are using linear probing, the probability that the first two slots of the table are filled after the first two insertions is:

(A) $\frac{1}{m^2}$

(B) $\frac{2}{m^2}$

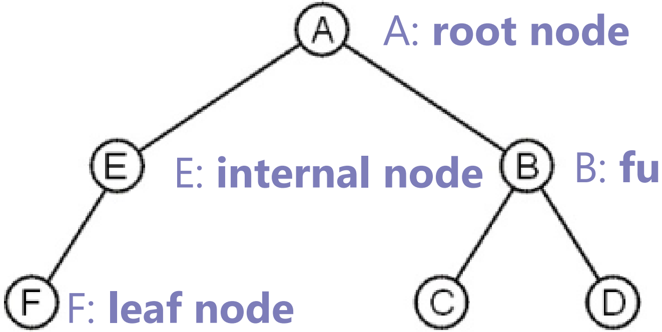
(C) $\frac{3}{m^2}$

(D) $\frac{4}{m^2}$

3.
Tree & Binary Tree

Tree concepts/Binary trees/Tree Traversal

Tree **Concepts**



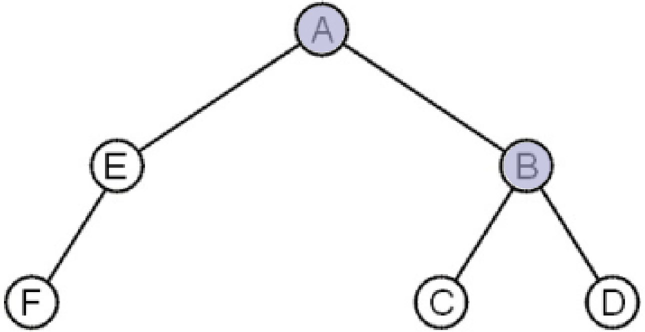
A: **root node** **depth**(A)=0

E: **internal node** B: **full node** depth(E)=depth(B)=1

F: **leaf node** depth(F)=depth(C)=depth(D)=2

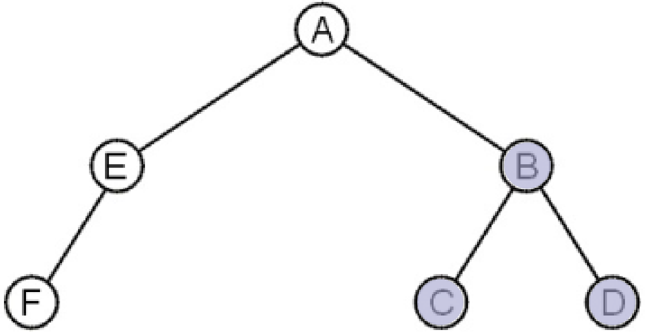
height of tree = max{depth} = 2

Tree **Concepts**



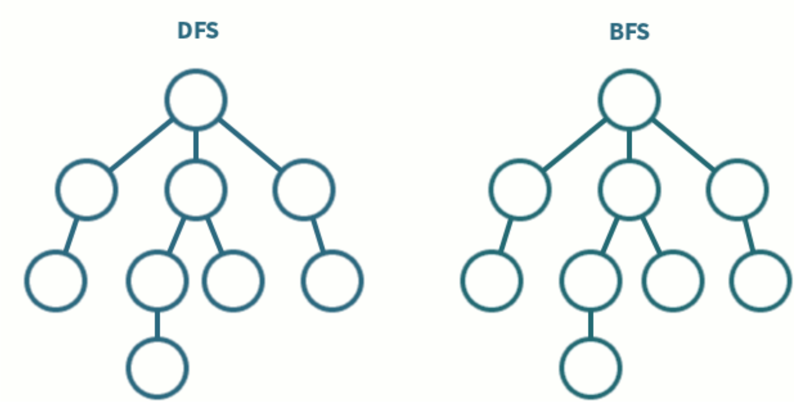
parent of B: A
ancestors of B: A B
strict ancestors of B: A

Tree Concepts



children of B: C D
descendants of B: B C D
strict descendants of B: C D

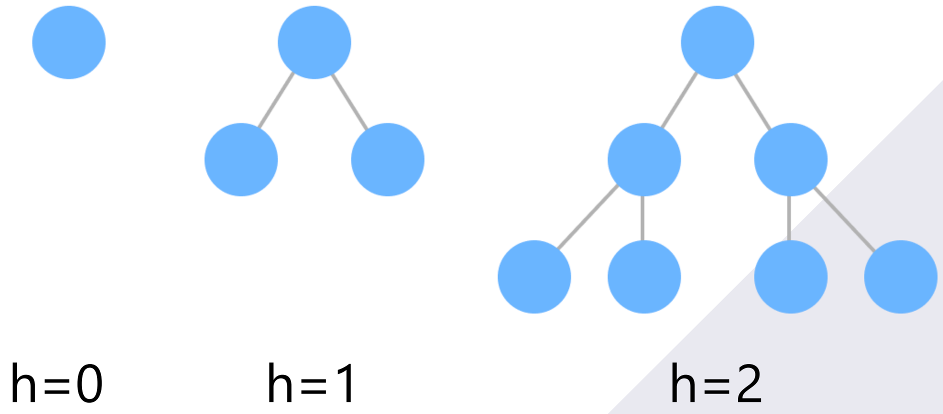
Depth-First vs. Breath-First



- Using stack
- Using queue

Binary Tree Concepts

→ **Perfect Binary Tree** $2^{h+1}-1$ Nodes



Binary Tree Concepts

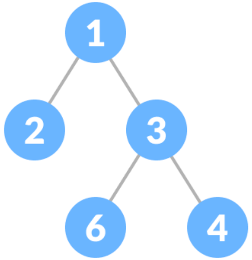
- **Perfect vs. Complete Binary Tree**
  - **Perfect:** Only exists for $N = 2^{h+1} - 1$ for every h = 0, 1, ...
  - **Complete:** exists for every N = 0, 1, ...

Binary Tree Concepts

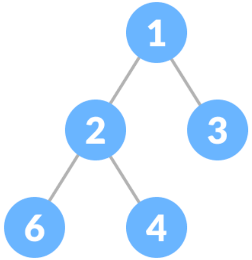
- **Full** vs. **Complete Binary Tree**
  - **Full:** node is either **full (2-children)** or **leaf (no child)**
  - **Complete:** filled at each depth from left to right
    - **Def1:** deepest: L to R; swallower: perfect
    - **Def2:** L complete + R perfect or L perfect + R complete
    - **Height:** $O(\log N)$ → heap

Binary Tree Concepts

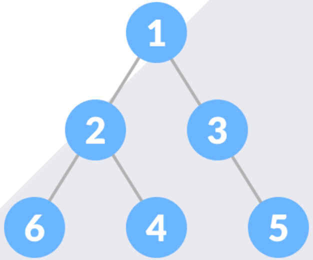
→ Full vs. Complete Binary Tree



✔ Full Binary Tree
✘ Complete Binary Tree

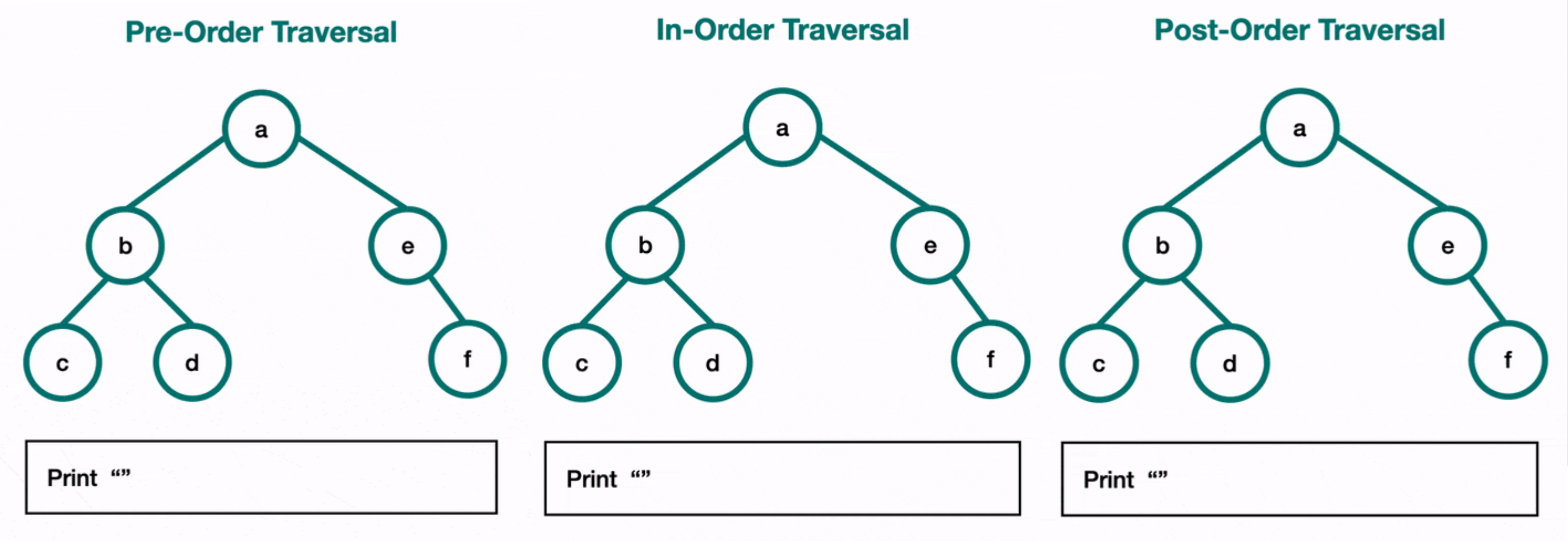


✔ Full Binary Tree
✔ Complete Binary Tree



✘ Full Binary Tree
✘ Complete Binary Tree

Pre-/In-/Post-order Traversal

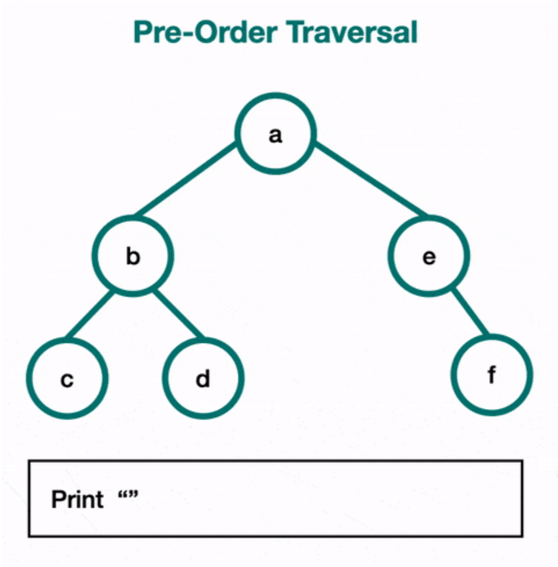


→ Root-L-R

→ L-Root-R

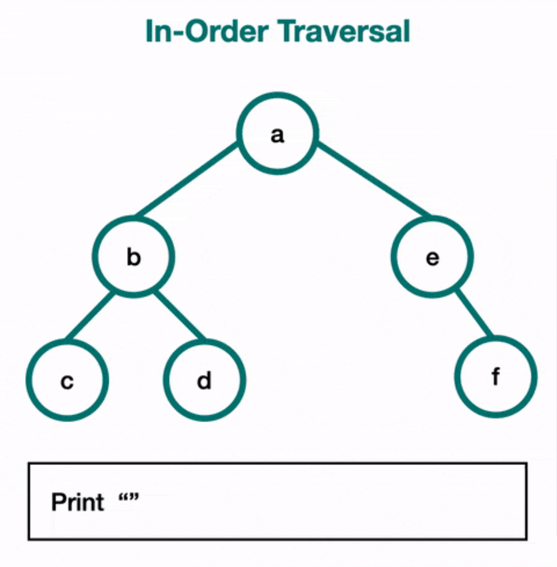
→ L-R-Root

Pre-/In-/Post-order Traversal



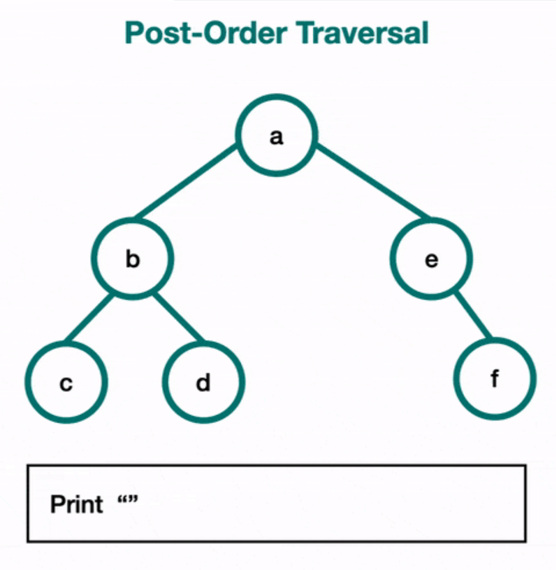
→ Root-L-R

Pre-/In-/Post-order Traversal



→ L-Root-R

Pre-/In-/Post-order Traversal



→ L-R-Root

Questions

(c) (1') If the pre-order traversal and in-order traversal of two binary trees are equal respectively, then the two binary trees are exactly the same. ✓

(h) (1') If the pre-order traversal and post-order traversal of two binary trees are equal respectively, then the two binary trees are exactly the same. ✗

- Pre-order: Root-L-R
- In-order: L-Root-R ✓
- Post-order: L-R-Root

Can't determine tree structure!

Questions

(a) (2') A full binary tree with n leaf nodes contains $2n - 1$ total nodes.

B. A rooted binary tree has the property that the number of leaf nodes equals to the number of full nodes plus 1. ✓

$N = N_0 + N_1 + N_2$ N_0:leaf N_2:full

$N - 1 = 0 \cdot N_0 + 1 \cdot N_1 + 2 \cdot N_2$ except root, each node is a child of its parent

$N_1 = 0$ node in full binary tree is either full or leaf

Questions

(a) (2') A full binary tree with n leaf nodes contains $\underline{2n-1}$ total nodes.

B. A rooted binary tree has the property that the number of leaf nodes equals to the number of full nodes plus 1. ✓

$N=N_0+N_1+N_2$ N_0:leaf N_2:full

$N-1=0\cdot N_0+1\cdot N_1+2\cdot N_2$ except root, each node is a child of its parent

$N_1=0$ $N_2=N_0-1$ node in full binary tree is either full or leaf

Questions

(a) (2') A full binary tree with n leaf nodes contains $\underline{2n - 1}$ total nodes.

B. A rooted binary tree has the property that the number of leaf nodes equals to the number of full nodes plus 1. ✓

$N=N_0+N_1+N_2$ N_0:leaf N_2:full

$N-1=0 \cdot N_0+1 \cdot N_1+2 \cdot N_2$ except root, each node is a child of its parent

$N_1=0$ $N_2=N_0-1$ node in full binary tree is either full or leaf

$$\Downarrow$$

$N=2 \cdot N_0-1$

Thanks!

Any questions?

Good Luck!

Contact me via lianyh@shanghaitech.edu.cn if you have any doubt