lecture 6 – September 16

Last time. recurrence relation

given:
$$\boxed{\begin{array}{l} T(1) = 1 \\ T(N) = 2T\left(\frac{N}{2}\right) + N \end{array}}$$

after trying $N = 2^0$, $N = 2^1$, $N = 2^2$, $N = 2^3$,

we found a pattern and claimed that

claim:
$$\boxed{T(2^k) = 2^k + k \cdot 2^k \quad \text{for all } k \in \mathbb{N}.}$$

Can we prove this claim rather than just saying that it seems to be correct based on our observations for $k = 0, 1, 2, 3$?

Formal proof. induction on k

induction base: $k = 0$

$$T(2^k) = T(2^0) = T(1) = 1 = 2^0 + 0 \cdot 2^0 = 2^k + k \cdot 2^k$$

induction hypothesis: assume $T(2^k) = 2^k + k \cdot 2^k$ for some fixed k.

induction step: $k \rightsquigarrow k+1$.

$$T(2^{k+1}) = 2 \cdot T\left(\frac{2^{k+1}}{2}\right) + 2^{k+1} = 2 \cdot \underbrace{T(2^k)}_{\substack{2^k + k \cdot 2^k \\ \text{by ind. hyp.}}} + 2^{k+1}$$

$$= 2 \cdot (2^k + k \cdot 2^k) + 2^{k+1}$$

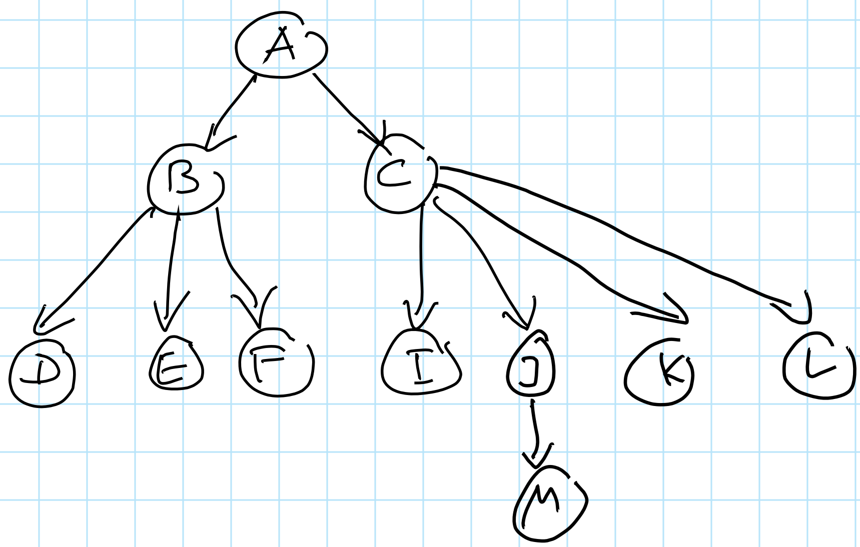
$$= 2^{k+1} + k \cdot 2^{k+1} + 2^{k+1} = 2^{k+1} + (k+1)2^{k+1}.$$

Excursion (Reminder: trees (Book: Chapter 4)

linked list operations → $\Theta(N)$

trees are often more efficient data structure

EXAMPLE 9.



root: A

children of A: B, C

children of B: D, E, F

parent of J: C

leaves: D, E, F, I, M, K, L

(C, J, M) is a path of length 2

depth of node J: 2

height of the tree: 3

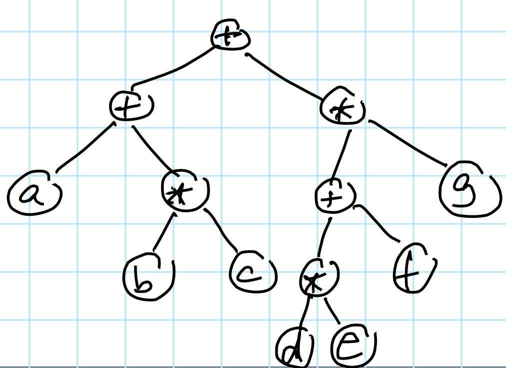
(directory listing)

preorder traversal: A, B, D, E, F, C, I, J, M, K, L

postorder -"- : D, E, F, B, I, M, J, K, L, C, A

level-order (breadth-first) -"- : A, B, C, D, E, F, I, J, K, L, M

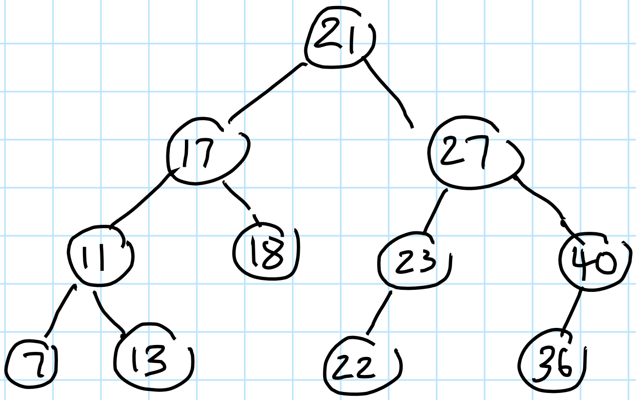
EXAMPLE 10. Binary tree: each node has at most 2 children



in-order traversal:

$[a + (b*c)] + [((d*e)+f)*g]$

EXAMPLE 11. BST binary search tree



- binary tree
- key value in node n is larger than any key value in left subtree of n and smaller than any key value in right subtree of n

searching for a key takes at most h steps, where h is the height of the tree.

recall how to search, insert, delete, find maximum, find minimum. (recursion!)

for these BST operations:

$$T_{avg}(N) = O(d_e(N))$$

where $d_e(N)$ is the expected depth of a node in a BST with N nodes.

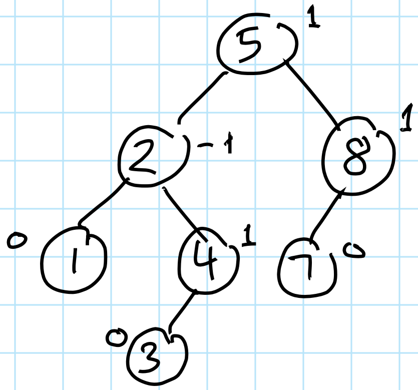
Theorem 3. If all insertion sequences are equally likely and no deletions occur, then the expected depth of a node in a BST with N nodes is $O(\log(N))$

$\leadsto$ the BST ops listed above would, under the conditions of Theorem 3, have an average running time of $O(\log(N))$.

difficulty with deletions: they make some tree shapes more likely than others

to ensure $O(\log(N))$ time, we need to balance trees?

EXAMPLE 12.



AVL tree : BST with additional property: height of left of subtree of node n differs by at most 1 from the height of right subtree of n.

balance factor of node n:
height of left subtree of n
minus " right -"-

⇝ AVL trees: each node has balance factor in $\{-1, 0, +1\}$

Theorem 4. The height of an AVL tree with N nodes is at most $\approx 1.44 \log(N+2) - 0.328 = O(\log(N))$

(In practice, slightly above $\log(N)$)

Corollary 2. The operations search, findMin, findMax, for AVL trees with N nodes have a worst-case running time of $O(\log(N))$

Same for insertion, deletion, since rebalancing works in time $O(\log(N))$.