Lec13-binary-search-tree

Saturday, September 21, 2024 9:20 PM

We often don't want to find just a number, but an object associated with a number

Dictionary abstract data type:

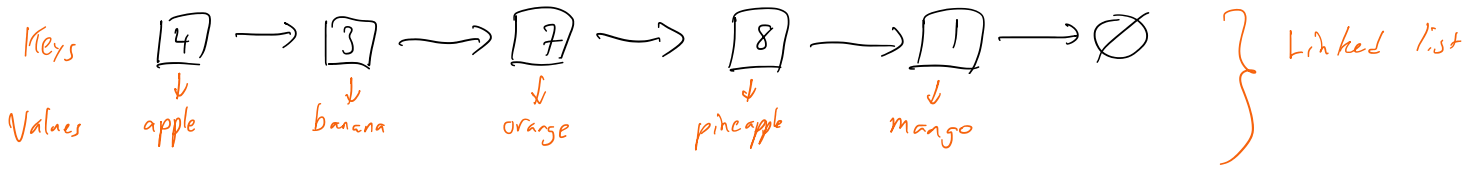
- insert (key, value)
- delete (key)
- value = find (key)

Many different ways to implement

Ex. Linked list with attached values

Ex Hash table " " "

Ex. Binary search tree



Finding value associated with "8" might require traversing list.
(what about "10")?

Binary search tree — attached items to "key"

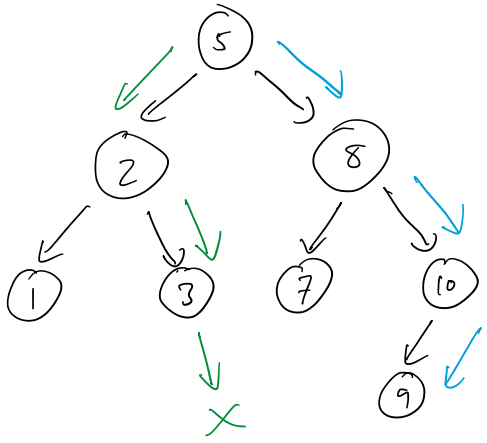
Binary: every node has 2 children

tree: rooted

BST property:

Given a node (k, v), all nodes
in left subtree have keys $< k$
in right subtree have keys $> k$.
(disallow duplicate keys for simplicity)

Ex.



(sorted [1, 2, 3, 5, 7, 8, 9, 10])

Find (4)
L 4<5, L
L 4>2, R
L 4>3, R ×
⇒ not present

Find (9)
L 9>5, right
L 9>8, right
L 9<10, left

Find:
if bigger, right
if smaller, left

Find Min ()
L always walk left

Insert: Same as find, but add a child when you reach an

Insert: Same as find, but add a child when you reach an appropriate empty spot. (otherwise, emit error on duplicate)

Pseudocode:

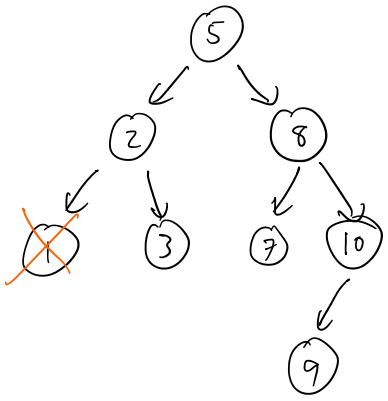
```
insert (T, k):
    q = NULL
    p = root(T)
    while p ≠ nil ∧ p.key ≠ k:        } find
        q = p
        if p.key < k, p = p.right
        else, p = p.left
    if p ≠ nil, error Duplicate
    N = new Node(k)
    if q.key > k, q.left = N
    else, q.right = N
```

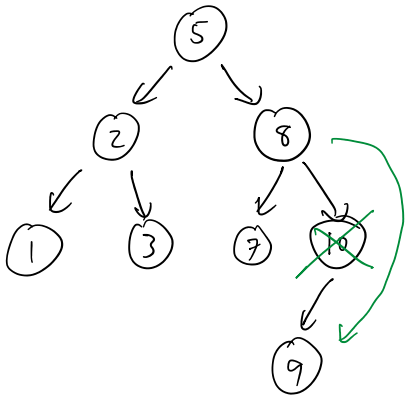
Delete: Find node u with parent p

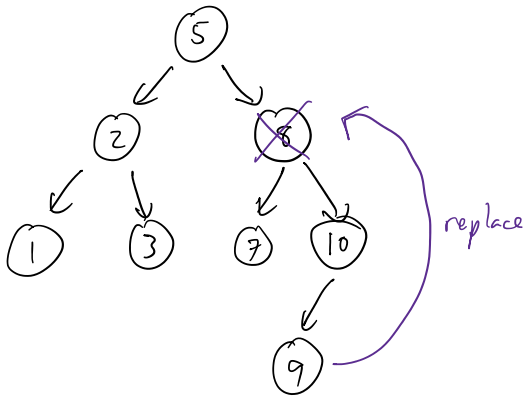
→ If u is a leaf, just delete

→ If u has 1 child c, delete u, and make c a child of p

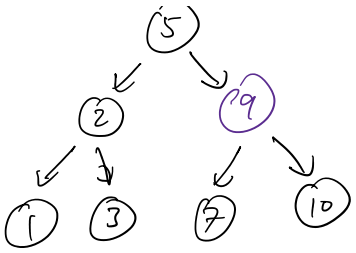
→ If u has 2 children, find smallest node in right subtree of u, delete it, and replace u with it.







⇒



9 still larger than left subtree
and smaller than right subtree

What can go wrong? Linearity

What would be optimal? Balance

How can we achieve balance?

↳ many options, such as AVL trees, red-black trees, splay trees, etc