Binary Search Tree (BST)

Data structures for dynamic set operations → INSERT, DELETE, SEARCH

Time of these ops is proportional to height h of tree O(h)

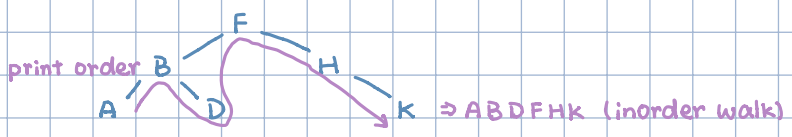
Each node contains these fields:

- key: value stored
- left: pointer to left child
- right: pointer to right child
- p : pointer to parent (p[root[T]] = NULL)

we also have root[T] pointing to root of BST T

Property

1. if y is in left subtree of x then key[y] ≤ key[x]
2. if y is in right subtree of x then key[y] ≥ key[x]



⇒ A B D F H K (inorder walk)

1) Inorder tree walk (recursive) (left, root, right)
   a. make sure x is not null.
   b. recursively print keys in x's left subtree
   c. print x's keys
   d. recursively print x's right subtree

2) preorder walk (root, left, right)

3). postorder walk (left, right, root)

Time for all three: Θ(n)

Minimum: Go to left most node: O(h)

Maximum: Go to right most node: O(h)

Successor of key x: next element after x in in-order walk

Predecessor of key x: previous element before x in in-order walk

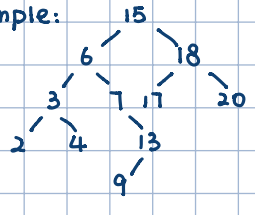
Successor time: O(h)

1. If node x has an non-empty right subtree then return the minimum in x's right subtree
2. If node x has an empty right subtree. Then x's successor is y where y is the predecessor of x (x is the max in y's left subtree)

Predecessor time: O(h)

Symmetric to successor

Example:



a) Successor of 15 : 17

b). successor of 6 : 7

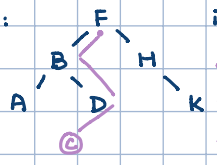
c). successor of 4 : 6

d). predecessor of 6 : 4

SEARCH / INSERT / DELETE

- **Search key k**: start at root and go left / right by comparing k with nodes if find k then return node
  else if reach leaf and not found, return NULL
  time: O(h)
- **Insert key k**: searches for key k (ignore duplicates) add new node with value k where search is done.
  time: O(h)

  Example:



  insert k
  search not found.
  reach leaf D
  insert at left of D.

- **Delete key k**: i.e: delete node z

  time: O(h)

  1). z has no children: delete z. parent of z points to null.

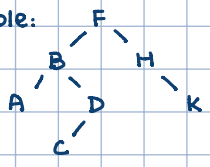
  2). z has one child: delete z. parent of z points to z's child.

  3). z has two children: a). find successor of z (say y)

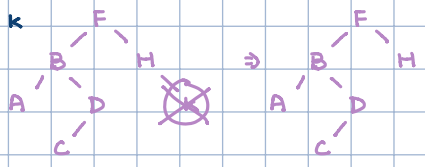
  b). delete y from the tree → has to be case 1 or 2 from successor's definition

  c). replace z's key with y's key

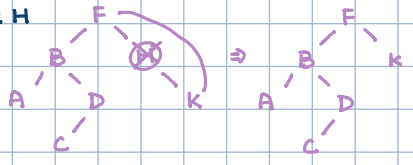
Example:



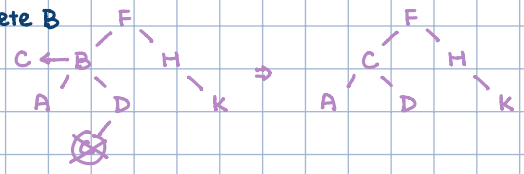
1. Delete k



2. Delete H



3. Delete B



1. Find successor : C
2. Delete C
3. replace key