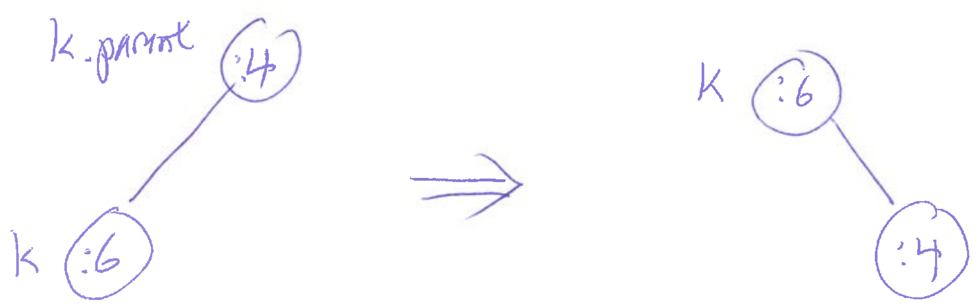
CSCE 750
10/5/2023

Treaps

Inserting into a treap an item k:

1) Insert k as normal for a BST (k is a leaf)
2) [Following path from k to the root:]

while k is not the root & k.priority > k.parent.priority
  do
    rotate to move k into its parent position
end-while



$$\text{Time(treap insertion)} = \Theta(\underbrace{\text{time for BST insertion}}_{\text{step (1)}})$$
$$= \Theta(\text{depth})$$

<u>Randomized</u> treap alg to implement a BST with expected depth $O(\lg n)$.

Starting with an empty treap:

Insert n items (with keys):

1) for each item, give it a priority chosen uniformly at random from, say, the unit interval $[0,1]$.

2) Insert into the treap.

Analysis: Recall: Treap structure is indep of the insertion order — only depends on key:priority ~~comb~~ combinations

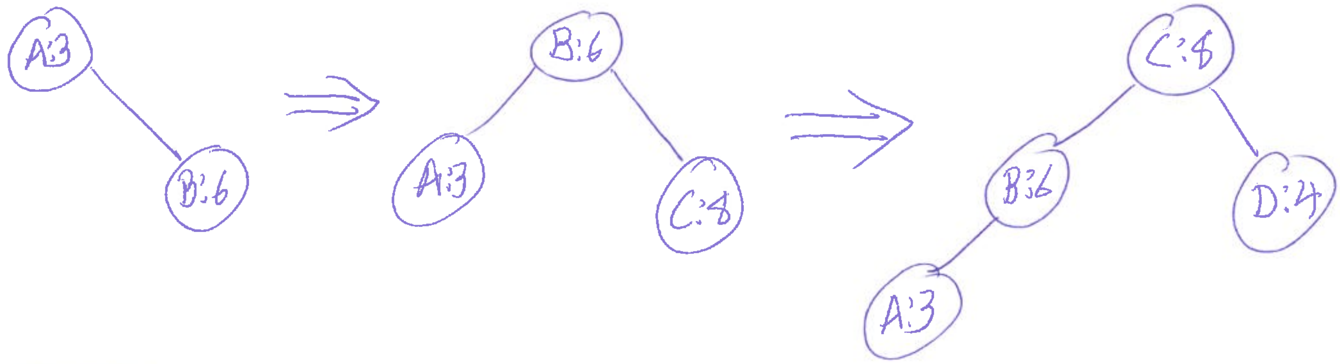
Notice: If items were inserted in order of decreasing priority, then there are no rotations, i.e., treap looks the same as a normal BST would with this insertion order.

Since priorities are random, get a treap identical to a BST whose items were inserted ~~uni in an~~ in uniformly random order.

Can show that

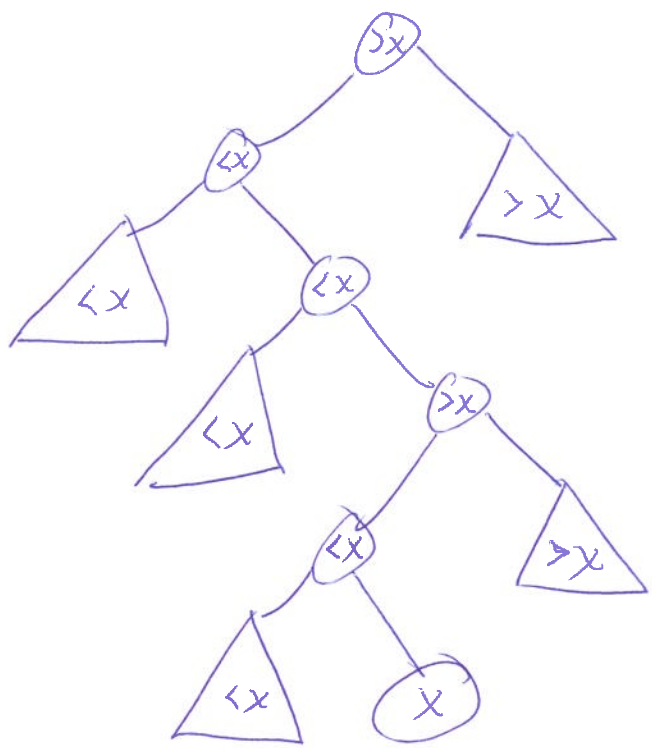
$$E(\text{depth of treap}) = \underset{\substack{\text{random}\\\text{insertion}\\\text{order}}}{E}(\text{depth of a BST}) \overset{\downarrow}{=} \Theta(\lg n)$$

Example:
Insert A:3 B:6 C:8 D:4 in that order



Augmenting data structures

Augment a BST to support selection
(finding the k'th smallest element)



Store at each node the size of the tree rooted at that node
"x.size"

```
Select(T, k)    // k > 0

    if T empty return nil    (T.size = 0)
    if k > T.size return nil

    if T.left.size == k-1
        return T
    if T.left.size >= k
        return Select(T.left, k)
    else // T.left.size < k-1
        return Select(T.right, k-1-T.left.size)
```