Randomized Data Structures

Treaps I

History:

1989: Seidel + Aragon
[Explosion of randomized algorithms]

Later discovered this was already known: Priority Search Trees from different context (geometry) McCreight 1980

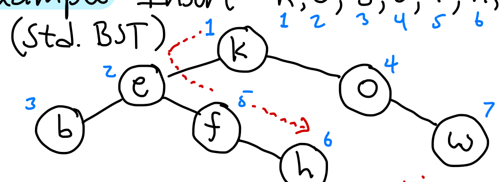
Intuition:

- Random insertion into BSTs $\Rightarrow O(\log n)$ expected height
- Worst case can be very bad $O(n)$ height
- Treap: A tree that behaves as if keys are inserted in random order

Example: Insert: k, e, b, o, f, h, w

1 2 3 4 5 6 7

(Std. BST)



Along any path - Insertion times increase

- Use a random number generator
- Running in expectation over all random choices
- Often simpler than deterministic

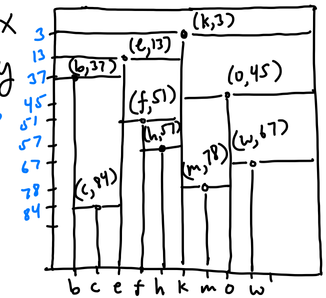
Obs: In a standard BST, keys are by inorder + insert times are in heap order (parent < child)

Geometric Interpretation:

key → x

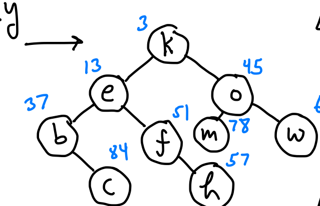
priority → y

McCreight's Priority Search Tree



Example:

Key	Priority
b	37
c	84
e	13
f	51
h	57
k	3
m	78
o	45
w	67



Treap: Each node stores a key + a random priority. Keys are in inorder. Priorities are in heap order

? Is it always possible to do both?

Yes: Just consider the corresponding BST

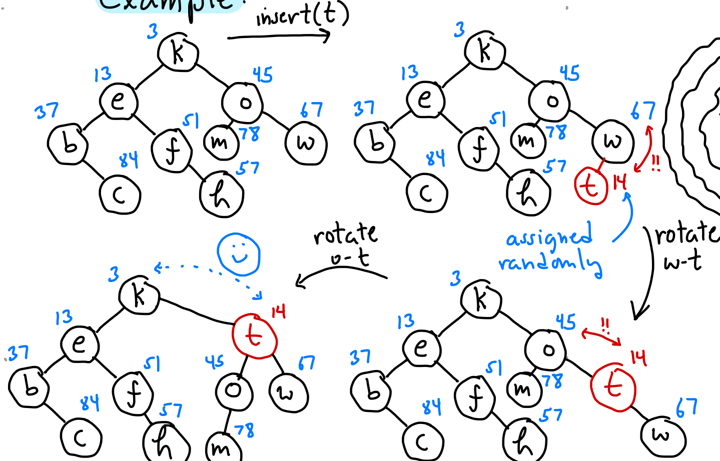
Treaps II

Insertion: As usual, find the leaf + create a new leaf node.
- Assign random priority
- On backing out - check heap order + rotate to fix.

Theorem: A treap containing n entries has height O(log n) in expectation (averaged over all assignments of random priorities)

Proof: Follows directly from BST analysis

Example:



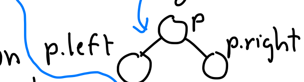
Deletion: (Cute solution) Find node to delete. Set its priority to +∞. Rotate it down to leaf level + unlink.

Implementation: (See pdf notes)

Node: Stores priority + usual...

Helpers:

lowest priority(p) returns node of lowest priority among:



restructure: performs rotation (if needed) to put lowest priority node at p.

Example: