September 6, 2023 10:46 AM

HW1 available soon

Balanced Search Trees

AVL trees
2-3 trees
weight-balanced trees
⋮

Method 5: Splay Trees (Sleator–Tarjan '85)

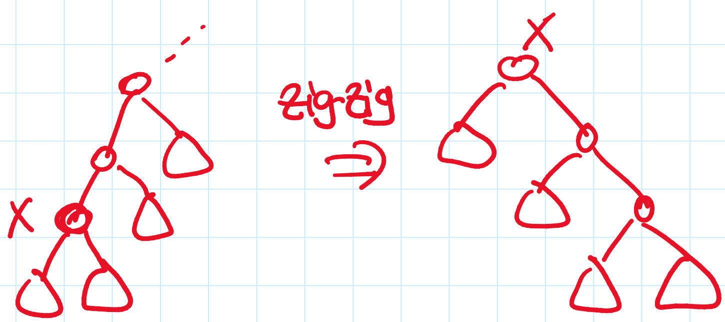
BST but without any color/size/height invariants!

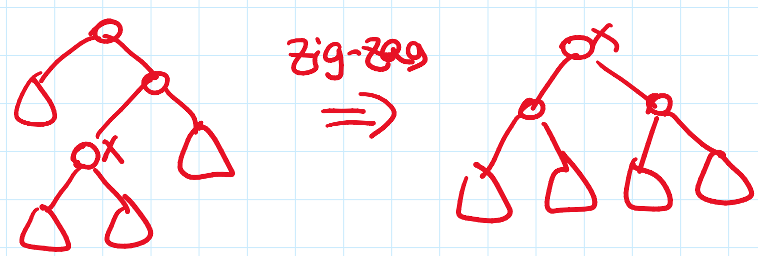
"self-adjusting"

after inserting x,
do rotations along path from x to root

after deleting x,
similar

after query, similar ** ← querying changes DS!





Known results:

- $O(\log n)$ amortized time for insert/delete/query
- in static case, for any sequence of queries:
  splay tree is optimal (up to const factor)
  compared to any other BST methods

Open: "Dynamic Optimality Conjecture"

is splay tree optimal for any sequence of queries & updates??

is there any method that is within $O(1)$ factor of optimal?

Known: $O(\log\log n)$ Demaine et al.'04 tango tree

Problem (Priority Queues)

build data structure for set S of n numbers
to support

key

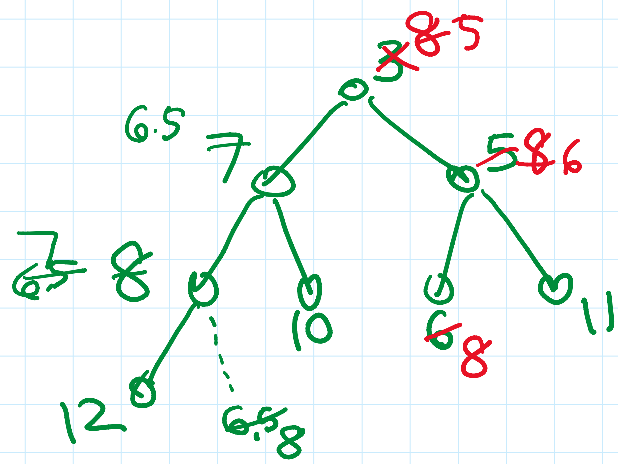
- find-min
- insert
- delete-min
- change-key { increase-key, decrease-key }

Method 0 balanced search trees

Method 0 - balanced search trees

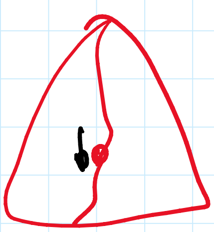
$O(\log n)$ time for all ops.

Method 1 - Standard binary heap



heap property: $\forall$ node v, $key(parent(v)) \le key(v)$.

perfectly balanced
(no ptrs, $i \to 2i, 2i+1$)



find-min: $O(1)$
insert: $O(\log n)$ (up a path)
delete-min: $O(\log n)$ (down a path)
increase-key: $O(\log n)$ (up a path)
decrease-key: $O(\log n)$ ~~(down a path)~~

preprocess: $O(n)$ ←
(build-heap)

increase-key/find-min

Lower bd: n inserts/delete-mins require $\Omega(n \log n)$ time
in comparison model
by reduction from sorting.

Q: faster insert? decrease-key?

binomial heaps (Vuillemin '78):
find-min $O(1)$

Binomial heaps (Vuillemin '78):

find-min $O(1)$
insert $O(1)$ amortized
delete-min $O(\log n)$

Fibonacci heaps (Fredman-Tarjan '85):

decrease-key $O(1)$ amortized.

[appl: Dijkstra's alg'm   n delete-mins
m decrease-keys
$\Rightarrow O(n\log n + m)$]

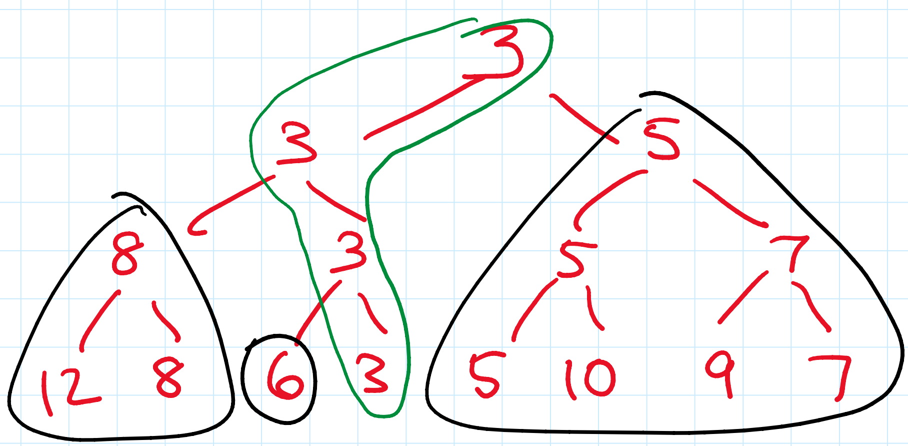
Other alternatives:

- Takaoka '03: "2-3 heaps"
- C '09: "quake heaps"
- Hansen-Kaplan-Tarjan-Zwick '15: "hollow heaps"
- Brodal '96: worst case
- pairing heaps: simplified Fibonacci heaps but poorer guarantees

⋮

O(1) amort. decrease-key

Tournament Tree Approach:



allow multiple tournament trees (forest)

insert(x): // be lazy!

just create a new tree for $\{x\}$

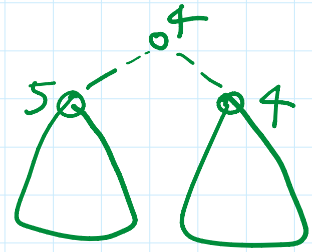
delete-min():

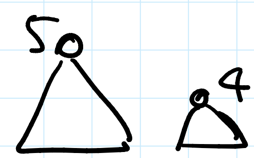
x = min of all the roots

remove path of x's nodes

whenever $\exists$ 2 trees of same height

link them





at end, $\leq \log n$ trees