Splay trees recap

- Dynamic BST
  - ↳ re-arranges tree even after search (via pointer moves and rotations)
- $O(\log n)$ amortized cost per operation
- In some cases stronger guarantees (e.g. if search sequence R structured, biased distribution, locality, ...)
- Conjectured constant-competitive ("almost" optimal on every search sequence R)

  optimal ⇝ best "offline" dynamic BST execution of R
  offline ↳ (knowing R in advance, optimized for R)

→ Dynamic Optimality Conjecture

Some partial results, but not even $o(\log n)$-competitiveness is known.

($O(1)$-comp. conjectured)

Easy claim. Splay is $O(\log n)$-competitive

Proof. $R = r_1, \ldots, r_m$ $m > n$ (n is # of distinct keys)

cost of splay on $R \leq m \cdot O(\log n)$ (Thm 1)

$OPT(R) \geq m$ (each search cost at least 1 — visit the root)

$$\Rightarrow \frac{Splay(R)}{OPT(R)} \leq O(\log n) \quad \square$$

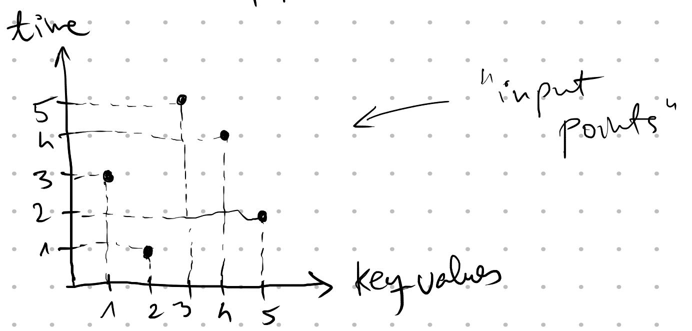
Geometry of Binary Search Trees

Search seq. $R = r_1, \ldots, r_m$

e.g. $R = 2, 5, 1, 4, 3.$ $\left(\begin{matrix} m=5 \\ n=5 \end{matrix}\right)$

Represent R as set of points (r_i, i) for $i = 1, \ldots, m$

e.g.



Serving R in a BST.

(Splay tree or any other BST based on pointer moves and rotations)

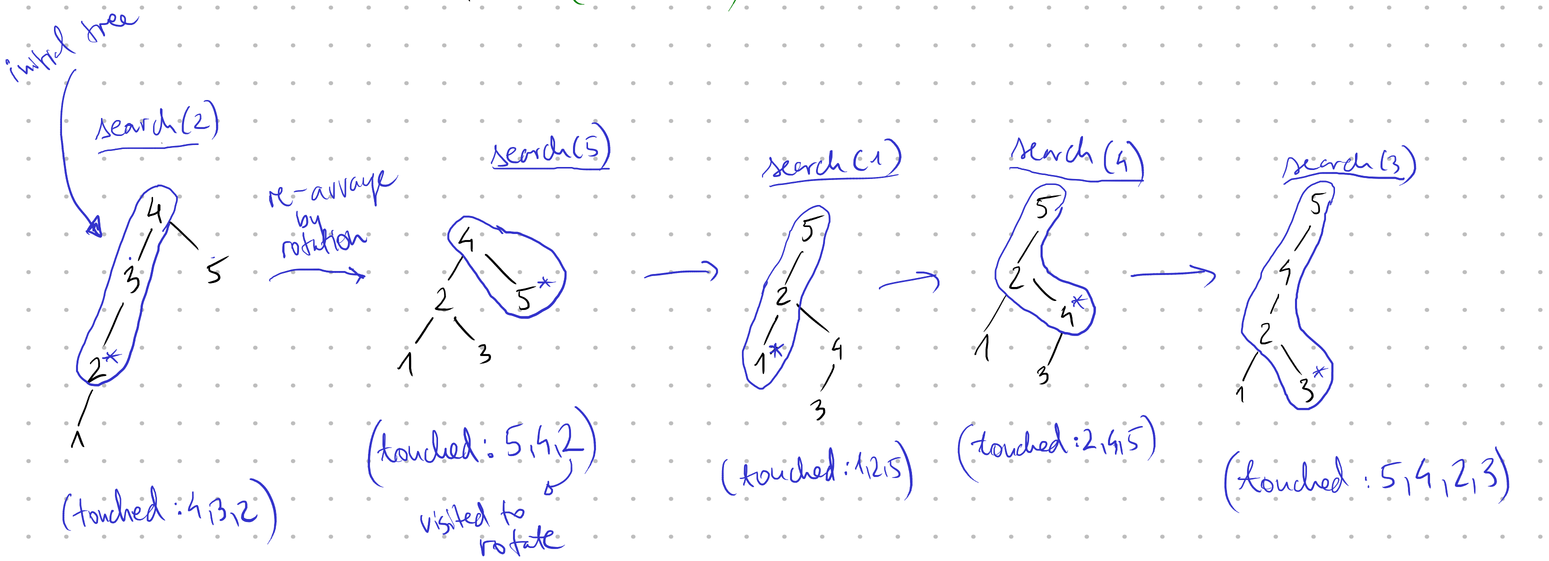
BST model:

- search(x): follow search path root ⇝ x, then re-arrange by doing rotations at the pointer

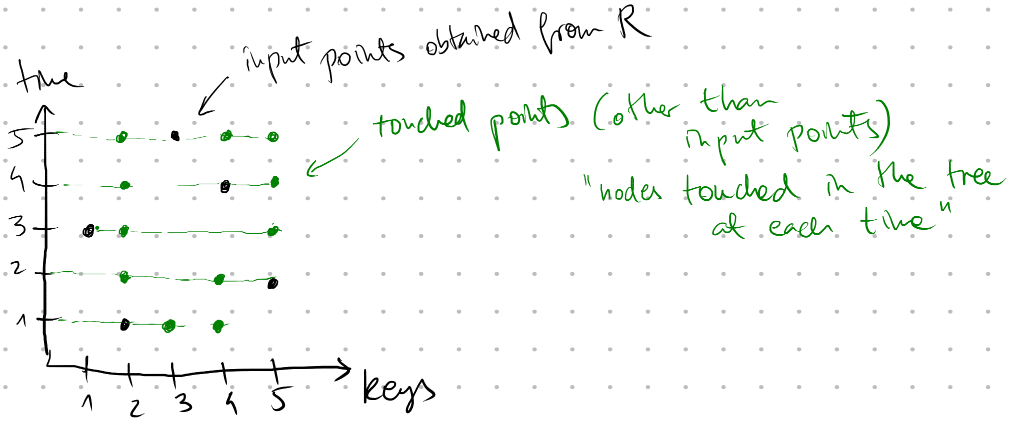
(pointer first moves from root to x, then we can move it anywhere step-by-step)

Cost: # rotations + # pointer moves (incl. search)

e.g. $R = 2,5,1,4,3$ $(m = n = 5)$



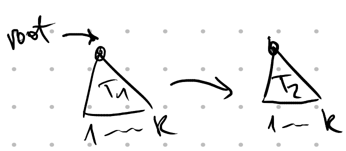
Plot the BST sequence as a set of points (that contains the input points)



Claim: Let T_1, T_2 be two BSTs of size k over same set of nodes.

Then we can transform $T_1 \to T_2$ with $O(k)$ rotations. (was exercise)

[This is true even if we allow rotations only at pointer and also count pointer moves.] (exercise – check this)



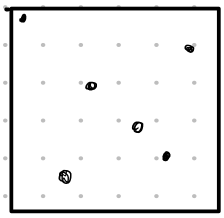
In BST execution we only need to do $O(k)$ rotations + pointer moves, where k is the # touched nodes

$\Downarrow$

$\boxed{\text{Cost of BST execution of } R \overset{*}{=} \#\text{ points in geom. view.}}$

(*small constant factor ignored)

Study BSTs in geometric view.

Given a point set P: 

"input points"
"
search seq.

find a superset of P that describes a valid BST execution and is as small as possible (= small cost)

<u>Theorem</u> A point set S describes a "valid execution" of P (obtained from search seq. R) by a BST

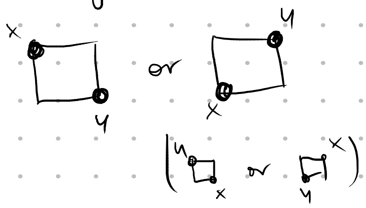
$\Updownarrow$

$P \subseteq S$, and

S has no "<u>empty rectangles</u>" (S is "satisfied")

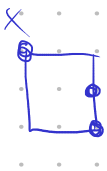
$\forall x, y \in S$: (not on the same horiz. or vertical line)

rectangle with corners x, y

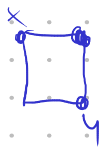


contains some point from $S \setminus \{x, y\}$ (possibly on boundary or other corner)

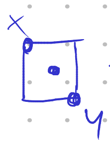
e.g.



→ not empty

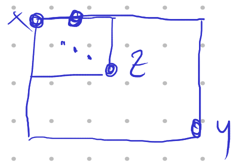


→ not empty



→ not empty

(observe that there must be some point on a rectangle side adjacent to x and to y, as we can apply condition repeatedly)

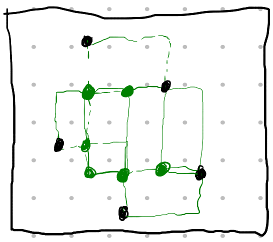


$|S|$ = cost of BST execution

(BST problem is reduced to finding a small satisfied superset of points)

Given P, find smallest $S \supseteq P$ with no empty rectangles

e.g.



Is this optimal?

S has no empty rectangle. Important: must consider all pairs

● vs ●

● vs ●

● vs ●

Note

In general, no efficient algorithm is known to solve this problem.

(It is also not known to be NP-hard)

Proof of Thm

I) S valid BST execution

$\Downarrow$

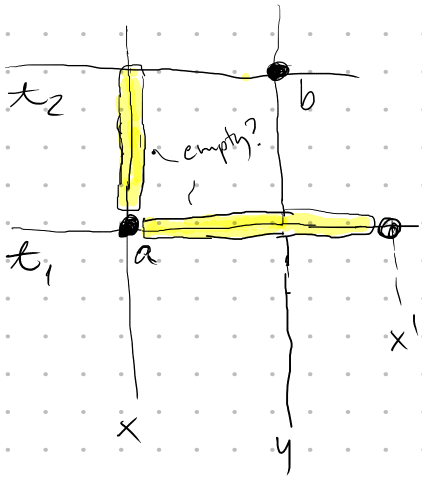
S has no empty rectangles (S is satisfied)

Consider any two points $a, b \in S$

$a = (x, t_1)$ $b = (y, t_2)$

w.l.o.g. $x < y$

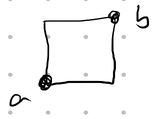
$t_1 < t_2$



BST touched x at time t_1, → a

touched y at time t_2. → b

want to prove



not empty

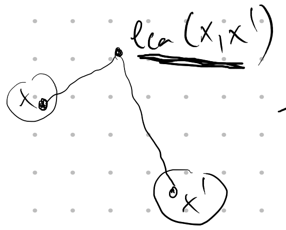
1) if there is a point $(z, t_1) \in S$ s.t. $z \in [x+1, y]$, then DONE.

2) ow. Let $x' > x$ be the smallest value s.t. point $(x', t_1) \in S$

(if no such point, set $x' = n+1$)

(we know $x' > y$, as otherwise we are in case 1)

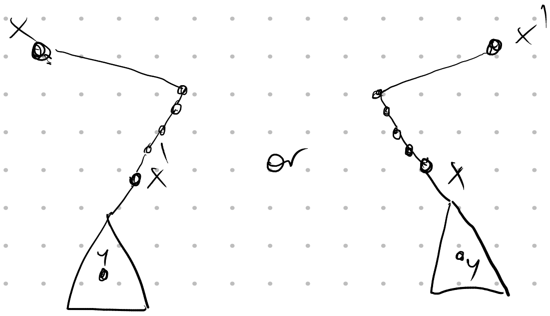
tree after time t_1:



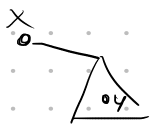
⟶ cannot happen: x and x' were touched but $lca(x,x')$ not (if $x < lca(x,x') < x'$)

(we assume nothing between x, x' is touched)

So it must be that one of x, x' is ancestor of the other.



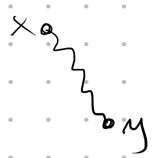
or if x' didn't exist at all:



as y was not touched, and $x < y < x'$, y must be in a subtree hanging below x

$\Longrightarrow$ after time t_1, y is descendant of x

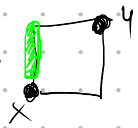
If x is not touched in time interval $[t_1+1, t_2]$, then y remains descendant of x.



$\Rightarrow$ y cannot be touched at time t_2.

So x must have been touched in this time interval, so rectangle not empty

$\Rightarrow$



not empty.

II) S describes a valid BST execution

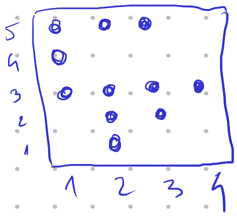
$\Updownarrow$

S has no empty rectangles

→ we need to show how to recover a BST execution from a point set S.
(may be not unique)

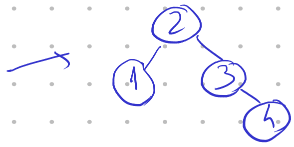
Initial tree T_0: treap of nodes, where priority is given by earliest touch time (with key values $1, \dots, m$) of each key value. (breaking ties arbitrarily)

e.g.



key	priority
1 →	3
2 →	1
3 →	2
4 →	3

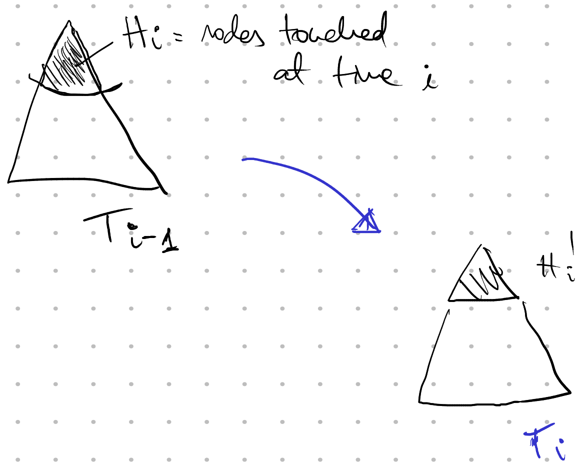
→



Denote T_i: BST after i-th search.

Invariant: T_i is a treap according to earliest touched time of keys in the interval $[i+1, \dots, m]$

Operation at time i:



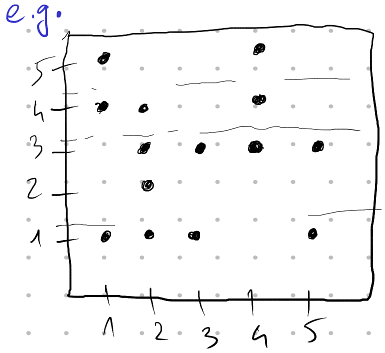
transform $H_i \longrightarrow H_i'$ s.t. it is a treap according to earliest touch time in future

T_i is T_{i-1}, with the top part H_i transformed into treap H_i' (rest of tree unchanged)

(need to prove that invariant is maintained)

→ from this it follows that the set of touched nodes H_i is indeed at the top of the tree.

Since these are the "earliest touched", they must have smallest priority, so they are on top by treap-property e.g.



S is satisfied

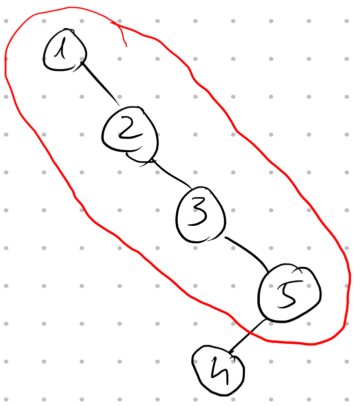
(break ties arbitrarily)

priorities

$1 \to 1$
$2 \to 1$
$3 \to 1$
$5 \to 1$

$4 \to 3$

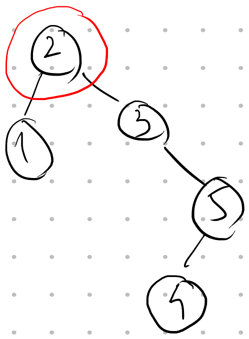
T_0



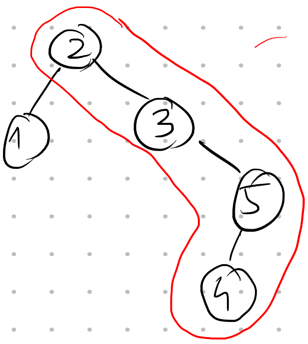
priorities

$1 \to 4$
$2 \to 2$
$3 \to 3$
$5 \to 3$

T_1

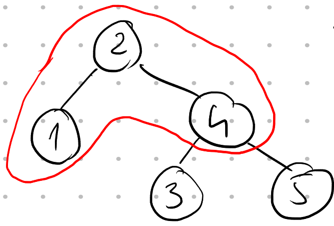


T_2



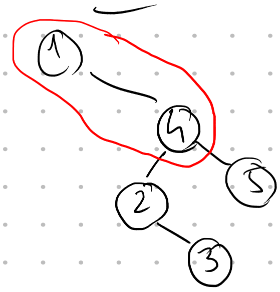
$2 \to 4$
$3 \to \infty$
$5 \to \infty$
$4 \to 4$

T_3



$1 \to 5$
$2 \to \infty$
$4 \to 5$

T_4

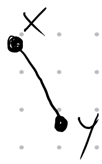


Proof that invariant is maintained:

(reminder:
invariant: T_i is a treap acc. to earliest future touch times)

Induction: true for $T_0, \dots, T_{i-1}$.

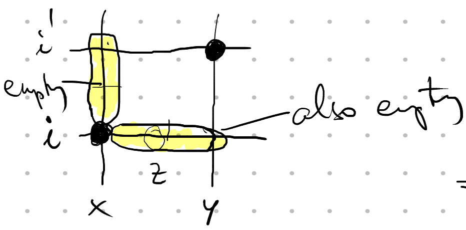
Suppose T_i not a treap. (some edge x, y must witness this)



x is above y, even though y will be touched earlier than x

T_i

- x, y cannot be both in H_i' (we just made H_i' a correct treap)
- x, y cannot be both in $T_i \setminus H_i'$ (T_{i-1} was a correct treap) and we didn't disturb this part



Suppose some point z touched at time i ($x \le z < y$)

$\Rightarrow$ y cannot be child of x

Bar above (x, i) empty since y will be touched first.

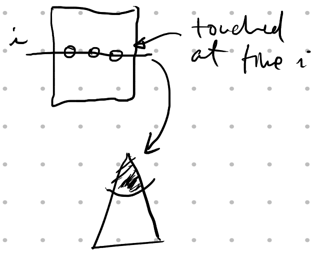
But then both sides of rectangle empty $\Rightarrow$ S not satisfied

Summary:

Finding a satisfied superset of P $\longleftrightarrow$ Finding a BST execution of R

(P: geom. view of R)

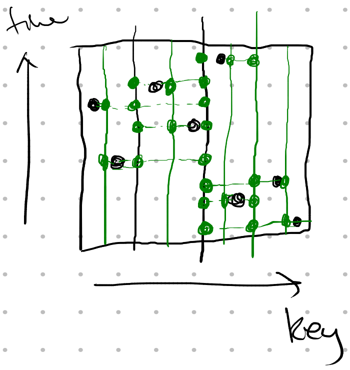
BST execution $\rightarrow$ point set : plot touched elements at each time



point set $\rightarrow$ BST execution: at every time i, touched elements form the top part of current tree, transform into a treap with future touch time is the priority.

(need to look at point set "globally")

Thm is called "offline equivalence"



- find satisfied superset
- no efficient algorithm is known

Two heuristics

1) Divide and Conquer

n: # input points

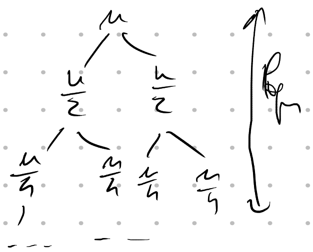
- split point set with vertical line in middle
- project each point to middle line

  Obs. no empty rectangles that cross middle line
- recurse on left/right side

Cost : size of point set

added "touch" points

$$T(n) = n + 2T\left(\frac{n}{2}\right)$$

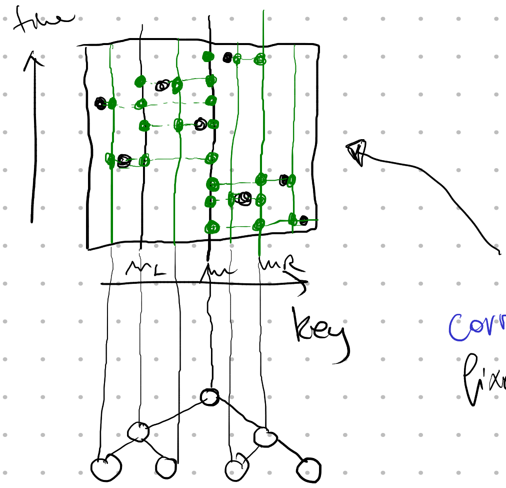


$$T(n) = O(n \log n)$$

$$T(1) = 1$$

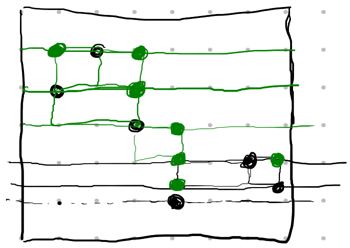
$\rightarrow$ BST execution that serves n searches in time $O(n \log n)$

BST with $O(\log n)$ cost per search



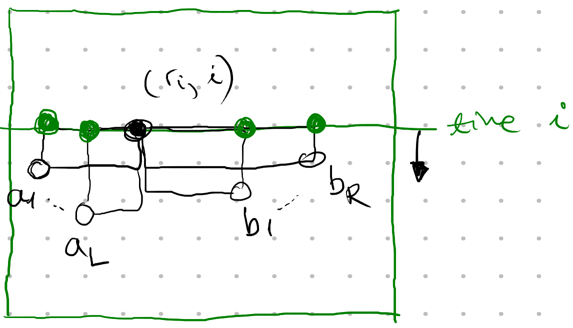
Corresponds to
fixed balanced BST (without rotations)

2) A different geometric algorithm:



↳ point set is satisfied

- geometric sweepline algorithm
- move through the input with horiz. line, step-by-step.
- add new points only on the line.
- invariant: below the line there are no empty rectangles

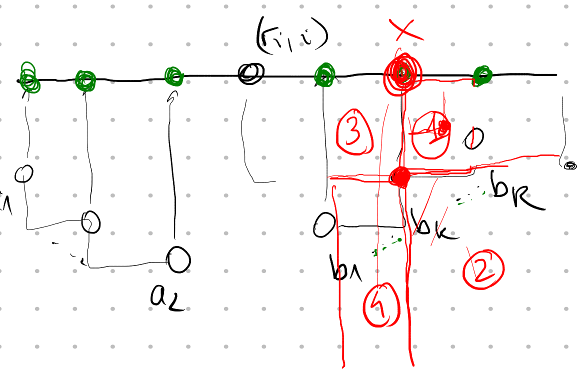


at time i, current search point (r_i, i) forms empty rectangles with points
$a_1, \dots, a_L$, $b_1, \dots, b_R$ } below line i

→ project $a_1, \dots, a_L$, $b_1, \dots, b_R$ } to line i

Claim. Alg. maintains invariant.

Show that no new empty rectangle is created at or below line i

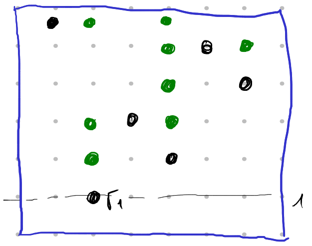


obs $b_1, \dots, b_R$ increasing by y-coordinate LTR

Suppose that some newly added point x created an empty rect.
Other corner is x'
Split plane in quadrants ①, ②, ③, ④
x' cannot exist ∄ (argue case-by-case)

This is called the Greedy algorithm.

e.g.

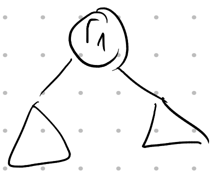


Algorithm finds satisfied superset.
What is it in tree-view?

<u>Claim</u>: Nodes touched by Greedy at time i (points in row i)
are exactly the search path of the tree.

<u>Proof</u>:

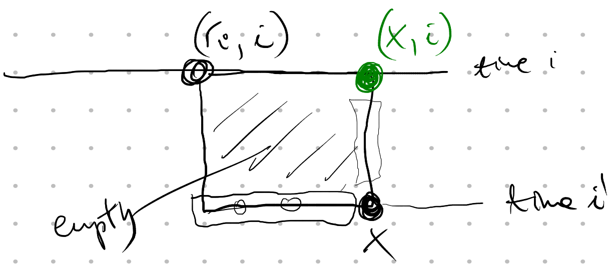
To: treap according to earliest touch time



for $i=1$ claim is true

Suppose it is true for $1, \dots, i-1$

at time i:

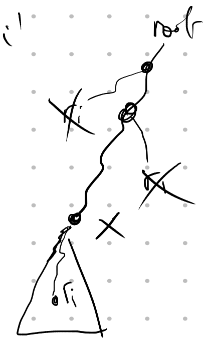


Need to show: x is on search path at time i.

by induction hypothesis x was on search path at time i',

no element in $\underline{[r_i, x)}$ was on search path at time i'.

at time i'



- r_i must have been a descendant of x at time i'.
- since x was not touched in the interval (i', i), r_i is still descendant of x when we search r_i.

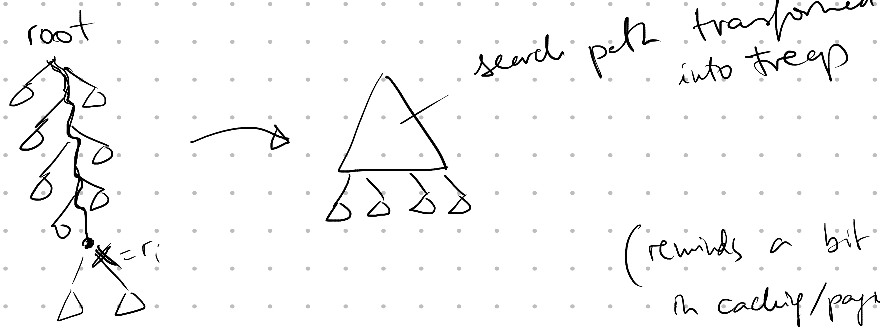
$\Rightarrow$ x is on search path of r_i at time i.

Greedy corresponds to a BST algorithm called GreedyFuture:

- search in BST,
  transform search path into a treap, using earliest future search as priority.

$R = r_1, \ldots, r_m$

search (r_i)



(reminds a bit of LFD in caching/paging)

Recall that Splay is an online algorithm.

GreedyFuture is an offline algorithm. [We will see how to make GreedyFuture online]

How good is GreedyFuture?

Conjectured $O(OPT)$ (constant-competitive)
which is better Splay / Greedyfuture?
OPEN

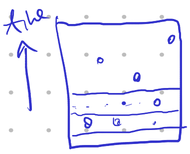
GreedyFuture satisfies Thm 1 – 6 (like Splay) and more.

Summary

tree-view	geom-view
offline algorithm ⟷	offline algorithm
GreedyFuture ⟷	Greedy
online: Splay, Move-to-root, Static tree, ? OnlineGreedy ⟷	Online: Greedy

online BST algorithm in tree-view: serves search sequence $R = r_1, \dots, r_m$ one-by-one, without knowing the future requests

online BST algorithm in geometric view: processes input points one-by-one (row-by-row) bottom-to-top, adds new points only in current row, without knowing input points above current row.



satisfied superset problem

Obs. Greedy is an online algorithm.

GreedyFuture is offline because of the transformation geom. view $\xrightarrow{\text{(Thm)}}$ tree view (used treap built with earliest future access times)

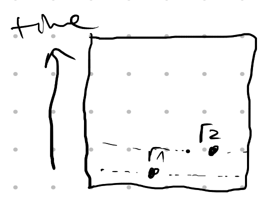
Thm (online equivalence)

online BST algorithm (in tree view) that serves R with cost C ⟷ online geometric algorithm that serves P (obtained from R) with cost $\Theta(C)$

Proof

⟶ (easy)

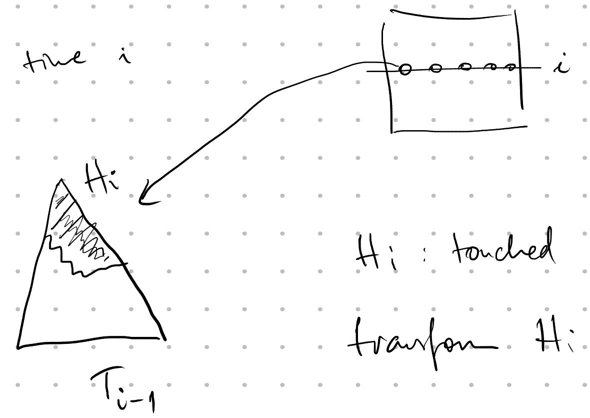
just run BST algorithm in tree-view, at each time plot in geom-view the points that are touched (visited by pointer)



tree-view $\longleftarrow$ geom.

Recall proof of offline equiv.

at time i

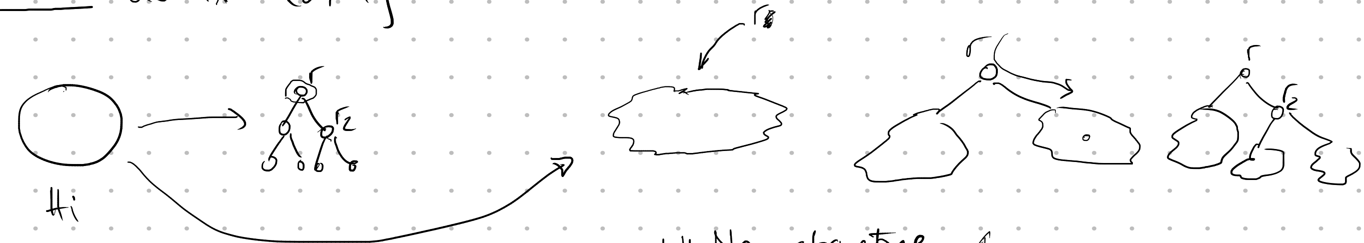


H_i : touched points at time i

transform $H_i \rightarrow H_i'$ treap according to future access times.

<u>Problem</u>: we don't know the future

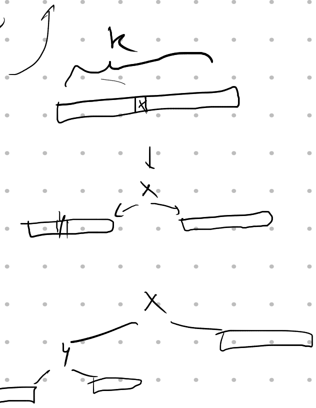
<u>Solution idea</u>: do it lazily.



$|H_i| = k$

- splittable structure
- splittable list

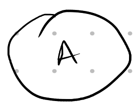
(see doubling search lecture)



amortized cost of splitting is $O(1)$

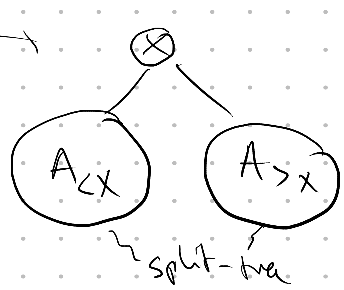
Split-tree data structure

Stores set A of keys



make-split-tree $(A) \longrightarrow O(|A|)$

split (x) $O(1)$ amortized time



Implementation:

(1) - splittable lists

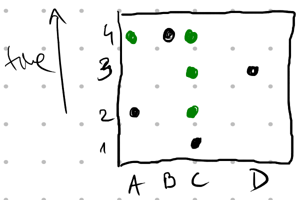
(2) - finger search trees

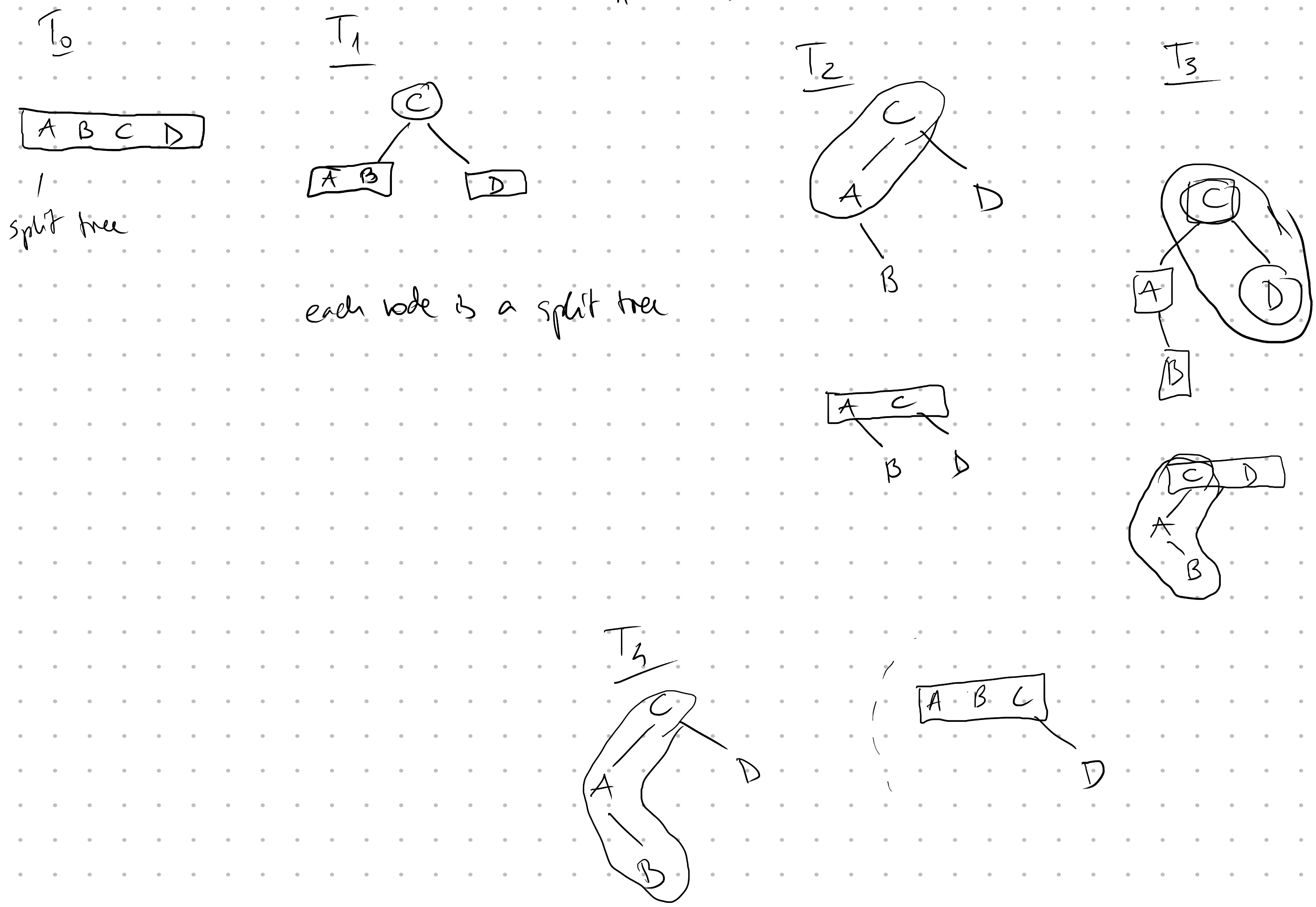
(3) - using AVL / red-black / (2,4) -trees (exercise)

→ need to use such tree-based implementation to remain in BST model.

→ replace treap in proof of thm by split-tree.

e.g. Online Greedy



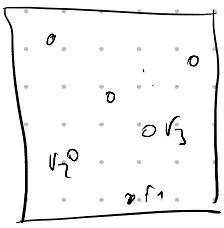


Summary

Geometric view of BST

$R = r_1, \ldots, r_m$

view it as set of points



Task: Find smallest superset of points with no empty rectangles

offline equiv. tree-strategy $\longleftrightarrow$ geometric algorithm.

online equiv. online tree-strategy $\longleftrightarrow$ geometric algorithm row-by-row

- Greedy Future
- Online Greedy $\longleftrightarrow$ Greedy

Conjecture

Greedy is constant-competitive

Like for Splay, conjecture is open.

Which is better? Splay or Greedy?

For both Splay and Greedy we cannot prove $o(\log n)$-competitiveness.

Is there any BST algorithm with $o(\log n)$-competitiveness?

$$\text{cost} \le OPT \cdot o(\log n)$$

YES: Tango-tree is $O(\log \log n)$-competitive (next)

(Best known)