Lower bound on OPT.

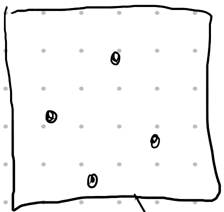
Search sequence $R = r_1, \dots, r_m$

OPT(R) = smallest possible # rotations and pointer moves for serving R

→ "offline" optimum

Recall that we would like to prove
cost of Splay or Greedy is $O(OPT(R))$ for all R.

In geometric view



P

OPT(P): smallest $S \supseteq P$ s.t. S has no empty rectangles

for some Algorithm denote Alg(P): cost of algorithm on P

Want to show $\dfrac{Alg(P)}{OPT(P)} \le$ (something) .

Comparing directly to OPT(P) is difficult, bec. OPT is not well-understood.

(recall in list update MTF vs OPT)
→ # inverted pairs

Idea: compare algorithm against a lower bound on OPT.

A quantity LB(P) s.t.

$$LB(P) \le OPT(P) \le Alg(P)$$

"lower bound" abstract quantity easy to compute

unknown OPT cost

cost of algorithm

$$\frac{Alg(P)}{OPT(P)} \Leftarrow \frac{Alg(P)}{LB(P)} \le \text{(Something)}$$

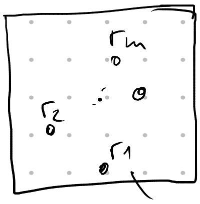
Need a good LB for OPT.

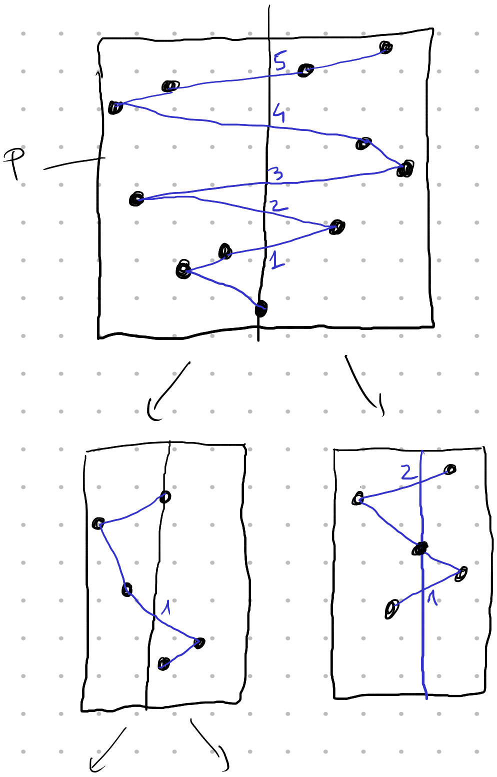
Interleave lower bound (Wilber 1980s)

$R = r_1, \ldots, r_m$

$IL(P)$

$IL(R)$





$IL(P)$:

- split with vertical line through the middle point by x-coordinate
- visit all points in order bottom-to-top
- enumerate L/R crossing
- do the same recursively on L/R side

$IL(P) = |P| + \#$ crossings at all levels

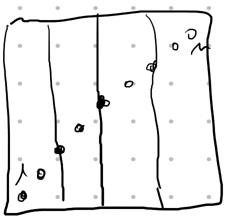
e.g. $IL(P) = 11 + 5 + 1 + 2 = \underline{19}$

Divide and conquer algorithm captured cost of executing search seq. in a fixed balanced tree. ($DC(P)$)

Obs. $IL(P) \le DC(P)$

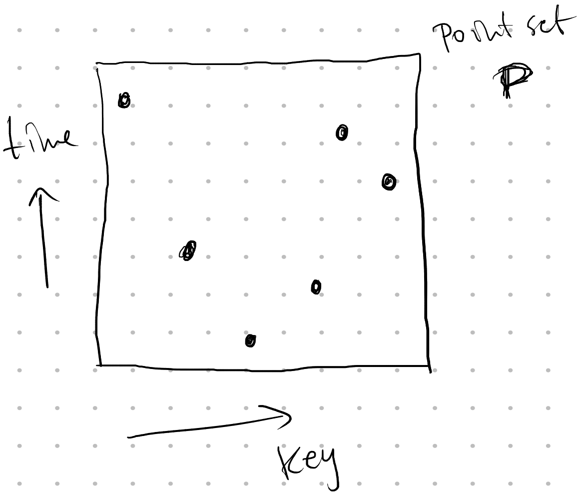
Thm. $\underline{IL(P)} \le \underline{OPT(P)} \le \underline{DC(P)}$

e.g.



$IL(P) = n$

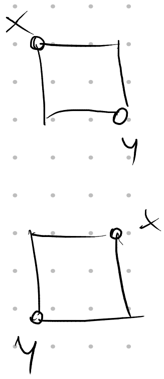
Geometric view of BST (recap)



Given P, find $S \supseteq P$,

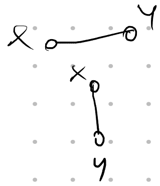
superset of P, as small as possible

such that $\forall x, y \in S$



rectangle with corners x, y contains some point in $S \setminus \{x, y\}$

(unless x, y in same row/column)



$OPT(P)$ ~ size of smallest such set S

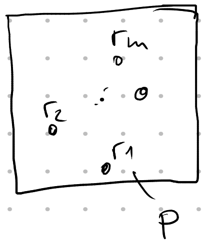
→ captures BST optimum of serving search sequence P

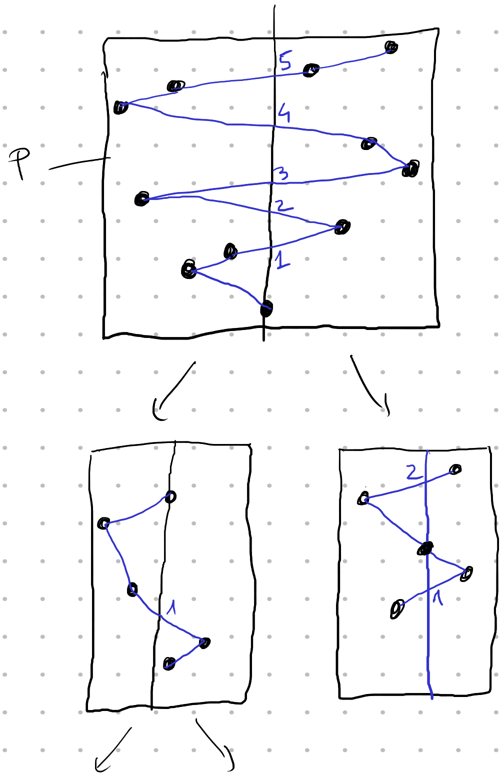
Interleave lower bound (Wilber 1980s)

$R = r_1, \dots, r_m$

IL (P)

IL (R)





IL(P):

- split with vertical line through the middle point by x-coordinate
- visit all points in order bottom-to-top
- enumerate L/R crossing
- do the same recursively on L/R side

IL(P) = |P| + # crossings at all levels

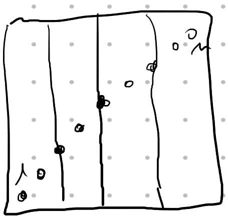
e.g. IL(P) = 11 + 5 + 1 + 1 + 2 = 19

Divide and conquer algorithm: captured cost of executing search seq. in a fixed balanced tree. (DC (P))

Obs. $IL(P) \le DC(P)$

$$\boxed{\text{Thm. } IL(P) \le OPT(P) \le DC(P)}$$

e.g.



IL(P) = m

Thm

$\forall P$

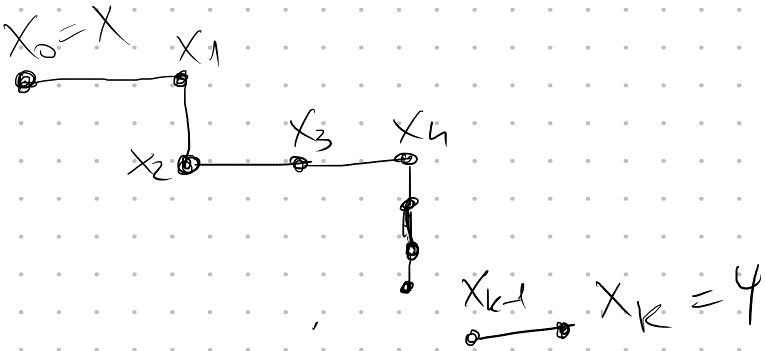
$$IL(P) \leq OPT(P)$$

Proof

$$IL(P) \overset{(2)}{\leq} MN(P) \overset{(1)}{\leq} OPT(P)$$

$MN(P)$ → optimum Manhattan Network of P

Manhattan path from x to y

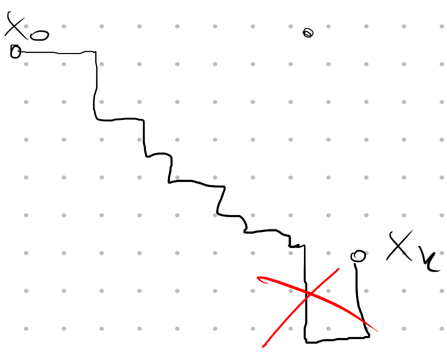


Sequence $x_0, \dots, x_k$ s.t. (1) x_i, x_{i+1} are in the same row or column

$\quad\ \ \shortparallel \qquad\quad \shortparallel$

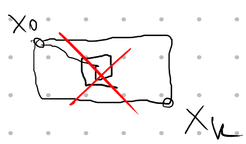
$\quad\ \ x \qquad\quad\ y$

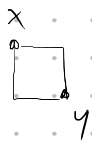
$\forall\, i \in [0, k-1]$



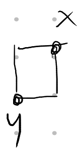
(no step in "wrong direction") ← (2) $d(x_{i+1}, x_k) \leq d(x_i, x_k)$

$\forall\, i \in [0, k-1]$





only allow steps → ↓



only allow ← ↓

(and symmetrical 2 cases)

Obs

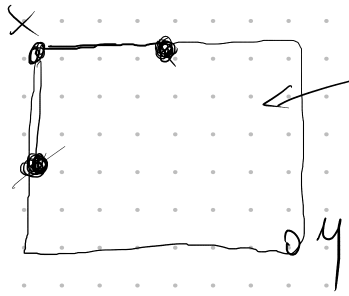
S has no empty rectangles

$\Updownarrow$

$\forall x, y \in S$ there is a Manhattan path $x \rightsquigarrow y$ in S

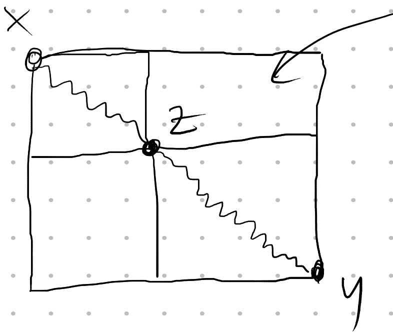
Proof

$\Uparrow$



rect. not empty

$\Downarrow$



must contain some point z

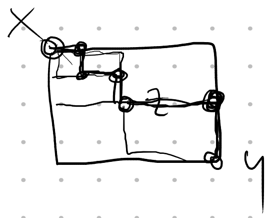
$x \to z$ Manhattan path

$z \to y$ ——"——

merging two paths is

$x \to y$ Manhattan path

(note: this is in general not true, but here:)



BST problem

Given P, find smallest $S \supseteq P$ s.t.

$\forall x, y \in S$ there is a Manhattan path $x \rightsquigarrow y$ in S

cost: $OPT(P)$

Alternative problem

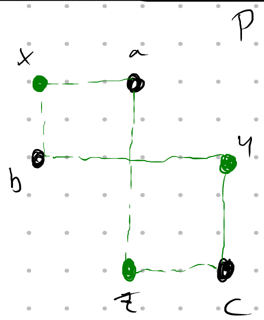
Given P, find smallest $S \supseteq P$ s.t.

$\forall x, y \in P$ there is a Manhattan path $x \rightsquigarrow y$ in S

input points

this is an easier problem

cost: $MN(P)$



Manhattan Network, i.e. there is path $a \leftrightarrow b$, $b \leftrightarrow c$, $a \leftrightarrow c$

but not a valid satisfied superset (BST): b, z form empty rectangle

Consequence:

(1) $\boxed{MN(P) \leq OPT(P)}$

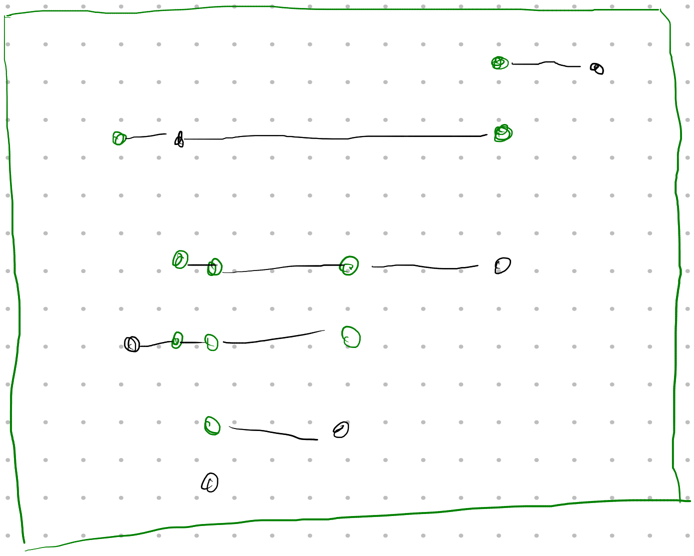
Remark

→ It is conjectured that $MN(P) = \Theta(OPT(P))$

→ Optimum Manhattan Network can be efficiently 2-approximated ($MN(P)$)

Remarks to show

$$IL(P) \overset{(2)}{\le} MN(P)$$

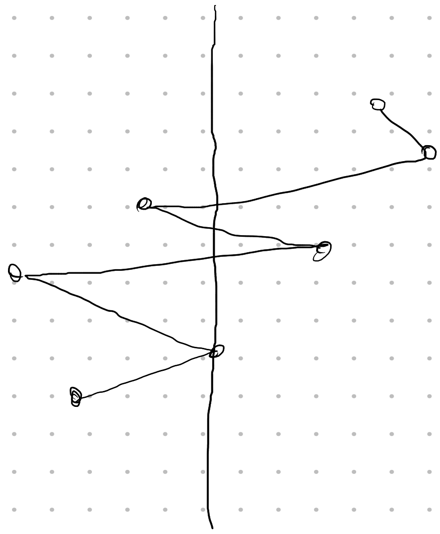


Optimal Manhattan-Netw.

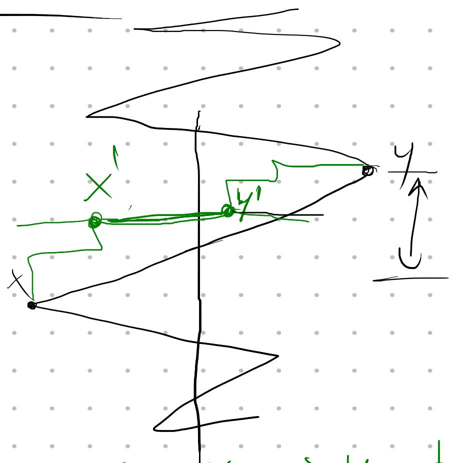
size $MN(P)$

segments in MN

$$\boxed{\#\text{segments} = MN(P) - |P|}$$

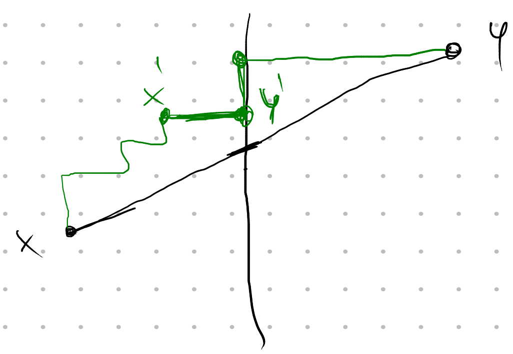


IL lower bound



charge crossing $x \to y$ to horizontal segment x', y' of Manhattan path (in OPT MN) $x \rightsquigarrow y$ that crosses separating line.

Obs. each horizontal segment of MN is charged at most once.



$\Rightarrow$ # crossings in IL $\le$ # segments in MN

$$IL(P) = \#\text{crossings} + |P| \le \#\text{segments in } MN + |P| = MN(P) \cancel{+|P|} \cancel{-|P|}$$

$$\Rightarrow IL(P) \overset{(2)}{\leq} MN(P)$$

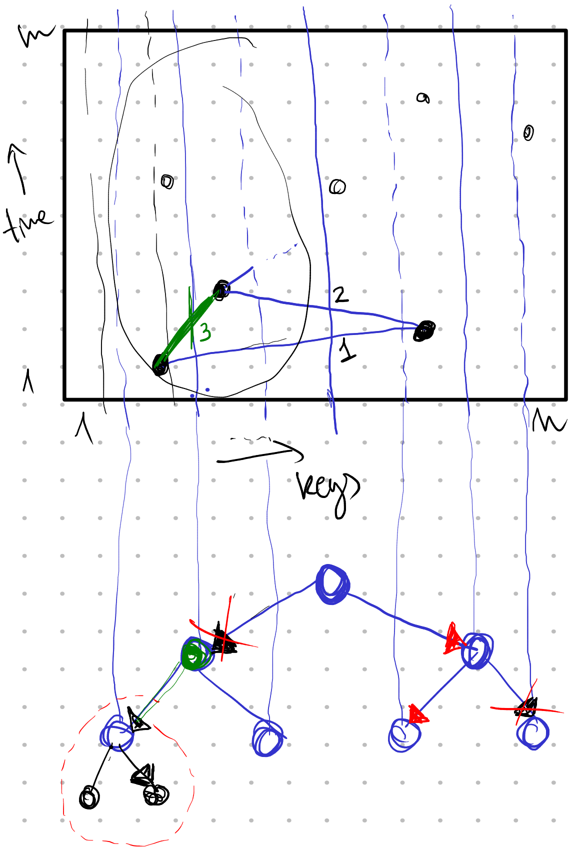
$$\Rightarrow IL(P) \leq OPT(P)$$

Next: Tango tree

Cost of Tango tree for R is $\leq O(\log\log m) \cdot IL(R)$

$\leq O(\log\log m) \cdot OPT(R)$

Start with IL lower bound



Point set P obtained from search sequence R

$R = r_1, \dots, r_m$

View recursive separating lines as nodes of a balanced BST

Each node has a "preferred child"

on which side we searched previously (most recently)

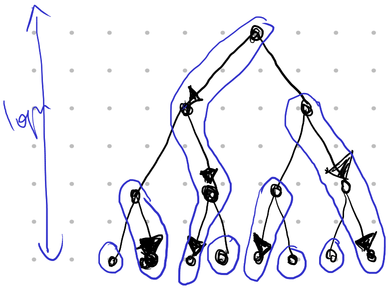
not yet a DS, just a "visualization".

Initialize arrows (preferred children) arbitrarily

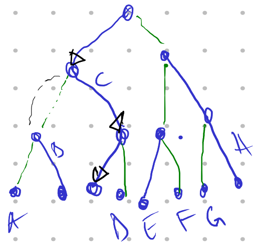
Update arrows after each search.

Obs. Changes in preferred child = Crossings in IL

lower bound tree

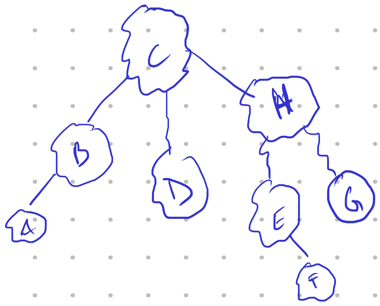


- Convert arrows into paths
- "preferred path decomposition" of tree

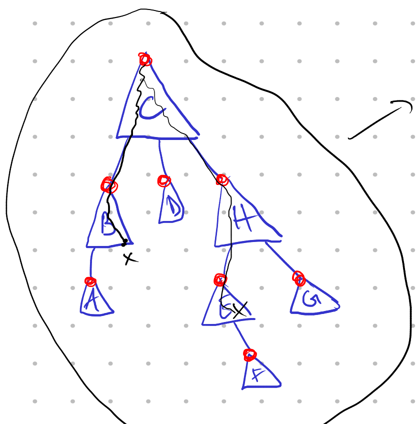


Obs each path is of length at most $\log n$.

Idea: maintain preferred-path decomposition of lower bound tree as a data structure.



→ each preferred path is maintained as an auxiliary balanced BST (e.g. AVL, red-black, splay)



→ a BST → Tango tree DS.

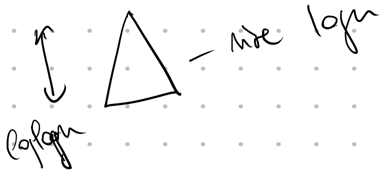
Search in Tango tree?

- as in a normal BST, from root until element is found

Obs. Cost of search:

as long as we stay inside an auxiliary tree (blob), cost is only $O(\log\log n)$

(size of each auxiliary tree = size of preferred path $\leq \log n$)



Problem: search may pass from one auxiliary tree to another.

→ taking a non-preferred child in LB tree → crossing in IL

In summary: whenever we cross from one aux-tree to another, we charge it to a crossing in IL

$\Rightarrow$ for every crossing in IL we spend $O(\log\log n)$ cost

$$\text{Cost} \le O(\log\log n) \cdot IL$$

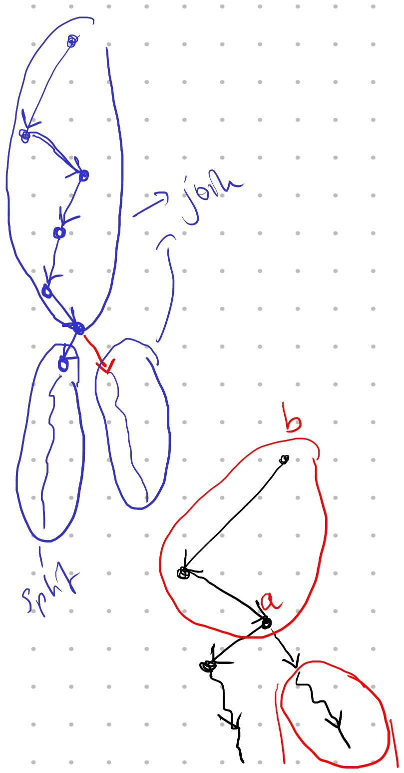
Note: we initialized arrows arbitrarily, so there are $|P|$ times when we cannot charge to a real crossing in $IL(P)$. But we defined $IL(P) = |P| + \#\text{crossings}$, so we can charge these to $|P|$ term

A detail still missing:

preferred child decomposition of LB tree must be updated after each search.

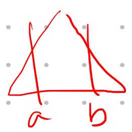
next search is in right subtree of root

updating arrow: we cut out part of B and join it with C

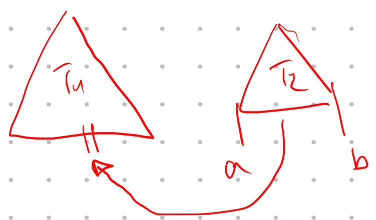


for every change of arrow 1 split + 1 join

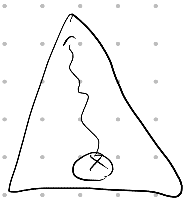
- split subtree contains keys in some interval $[a,b]$



- join is between two trees, one forms a contiguous interval $[a,b]$, other has no key in $[a,b]$

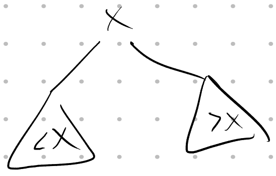


Lemma: We can implement split and join of trees of size $\le k$ in time $O(\log k)$.
↳ AVL / red-black / splay



- split splay tree at x

Splay(x)



Sketch

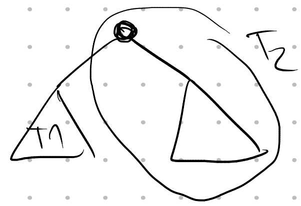
amortized $O(\log n)$ time

using standard split/join in splay trees

- join

splay max of T_2 as root

attach T_1 as left child



split and join of auxiliary trees takes $O(\log \log n)$ time

size $O(\log n)$

happens for each arrow-change crossing in IL

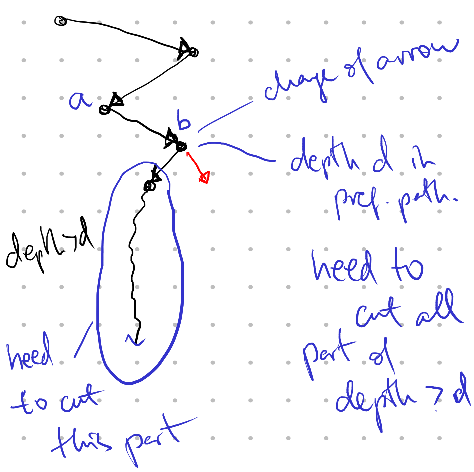
We also charge this to IL lower bound.

$\Rightarrow$ total cost: $O(\log \log n) \cdot IL(R)$

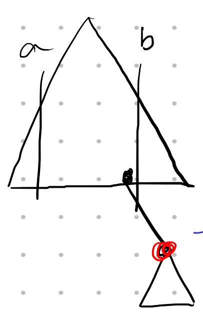
$\leq O(\log \log n) \cdot OPT(R)$

Note What bookkeeping is needed to implement split/join of aux-trees?

pref.-path



in aux-tree



We notice we entered a different aux-tree, so we need to cut/join.

We augment aux-tree to store in each node its original depth in pref. path.

So we know b and depth d.

We need to cut out all nodes with depth > d.

this is depth in pref.-path, not depth in aux-tree

These nodes form an interval in aux-tree [a,b]

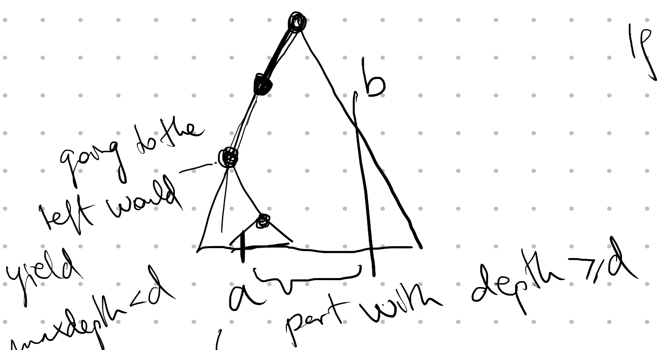
How do we find a?

Need an extra augmentation: each node stores maxdepth of nodes in its subtree.

(this needs to be maintained under rotations in red-black/splay, see lecture on augm. tree)

to find a, go down from root repeatedly, as long as maxdepth ≥ d

If possible, go to left child, o.w. right



walking down from root is loglogn time in red/black (loglogn amortized in splay if we re-arrange tree)

Summary

- we need to cut out nodes with depth > d
- aux-tree is not sorted by depth, but by key.
- nodes with depth ≥ d form interval [a,b] by key
- We can find a,b using augmentation