Lecture Notes for $O(\lg\lg n)$-Competitive BSTs

Recap: • Binary Search Tree (BST) model

→ consider a set S of $|S|=n$ ordered elements

let $S=\{1,\dots,n\}$ for concreteness

→ must maintain a dynamic BST with elements in S

↳ can be changed thru rotations

→ only operation: search(x), where $x \in S$

Cost = Running Time = # nodes touched by search + # rotations performed

• Goals: → given a sequence $\bar{x}=(x_1,\dots,x_m)$, find $OPT(\bar{x})$ = = optimal cost to run search(x_1),..., search(x_m) in BST model

Examples: $OPT(1,2,\dots,n)=\Theta(n)$

$\mathbb{E}[OPT(\bar{x})]=\Omega(m\lg n)$ for random $\bar{x}\in[n]^m$

Wanted: α-approximation algorithm

→ sequence $\bar{x}$ revealed online (x_1, then x_2, then x_3...)

Wanted: α-competitive algorithm (compared to offline $OPT(\bar{x})$)

Why? "All Pairs Shortest Paths should take $O(n^3)$, but on my company's graphs my optimized algorithm runs much faster"

"This problem is NP-complete, but this optimized backtracking solves it very well on my practical instances"

We'll pick a very simple problem (BSTs) where we understand the worst-case well

Can we understand the "difficulty" of each particular instance?

Results:

- $O(\lg n)$-competitive = trivial
- Tango-trees (today's topic): $O(\lg\lg n)$ competitive, $O(\lg\lg n)$ approximation (best known even offline)
- Conjecture: splay trees $O(1)$-competitive [Sleator Tarjan '83]
- Conjecture: a greedy algorithm is $O(1)$-approx [Lucas '88][Munro '00]

If so $\Rightarrow \exists$ online $O(1)$-competitive [DHIKP '09]

How to prove approximation / competitive ratio:
compare upper bound with lower bound
↳ often value of linear program, etc
Here: lower bound = lower bound on computation

Wilber I
[Wilber '86]

- plot sequence $\bar{x}$ on time/value diagram
- build a tree over values ⇝ "lower bound tree" (arbitrary, static)
- "interleave thru a node" = switching from left subtree → right subtree

Lower bound = $\sum_{\text{all nodes}}$ (#interleaves)

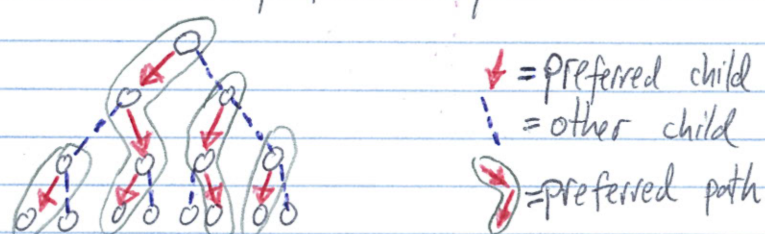
See Fig 1 next page

Proof: later

Tango trees:

Cost = (Wilber I) × $O(\lg\lg n)$

Each node has a preferred ~~path~~ child (where most recent access happened)

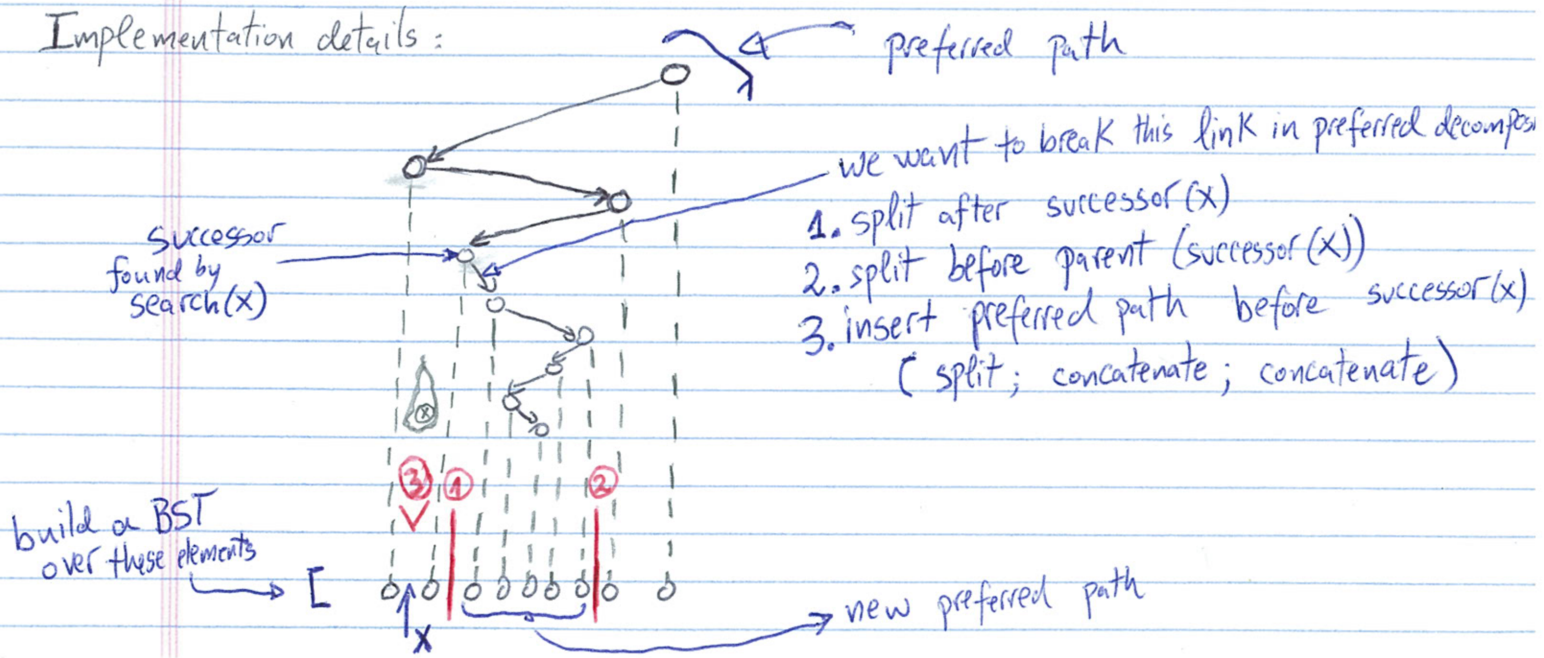


"Preferred path decomposition" of the lower-bound tree

1. Lower bound tree = perfect binary tree
2. Maintain each preferred path as dynamic BST (red-black tree / AVL / splay tree / etc)

{ Preferred path has $\leq \lg n$ nodes ⇒ $O(\lg\lg n)$ time / operation
{ # preferred paths visited = Wilber I

Implementation details:



Proof of Wilber I ⟿ a lower bound on computation

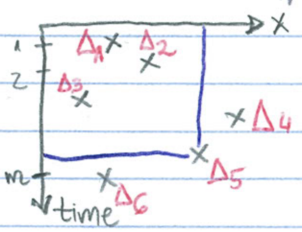
Conceptual shift: what are BSTs useful for?

→ searching for elements? maybe, but not really optimal (think hashing, etc)
→ augmented BSTs maintain aggregates!

Partial-sums problem: given array $A[1..n]$ $\begin{cases} \bullet\ \text{update}(i,\Delta): \text{change } A[i]=\Delta \\ \bullet\ \text{query}(i): \text{return } \sum_{j=1}^{i} A[j] \end{cases}$

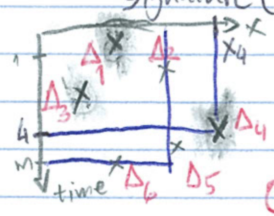
Computational model: let S = arbitrary semigroup
each $A[i] \in S$; machine memory stores $Mem[i] \in S$
Cost = # semigroup additions performed

Given $\bar{x}=(x_1,\dots,x_m)$ execute: update(x_1,Δ_1), query(x_1); update(x_2,Δ_2), query(x_2); ..
BSTs support these $2m$ operations in $O(OPT(\bar{x}))$ cost (additions)



the answer to query(x_5) is $\Delta_1+\Delta_2+\Delta_3+\Delta_5$

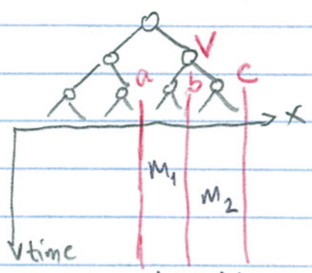
Signatures: each memory location looks like $Mem[a] = \Delta_{i_1}+\Delta_{i_2}+\dots$
Signature$(Mem[a])$ is $(t,x) = (\max\{i_1,i_2,\dots\}, \max\{x_{i_1},x_{i_2},\dots\})$



if $Mem[17] = \Delta_1+\Delta_3+\Delta_4 \Rightarrow$ Signature$(Mem[17]) = (4, x_4)$
better example: $Mem[13] = \Delta_1+\Delta_6+\Delta_2 \Rightarrow$ signature$(Mem[13]) = (6, x_2)$

Observation: Signatures move down-~~left~~ right when adding elements

Wilber I: for each node $v \in$ lowerbound tree, let
$A(v) = \{$ all additions "m_1+m_2" with → signature$(m_1) \in [a,b] \times [1,m]$, → signature$(m_2) \in (b,c] \times [1,m]$ $\}$



$A(v)$ for all v is a partition of all additions executed by code
$\Rightarrow$ cost $= \sum_v |A(v)|$

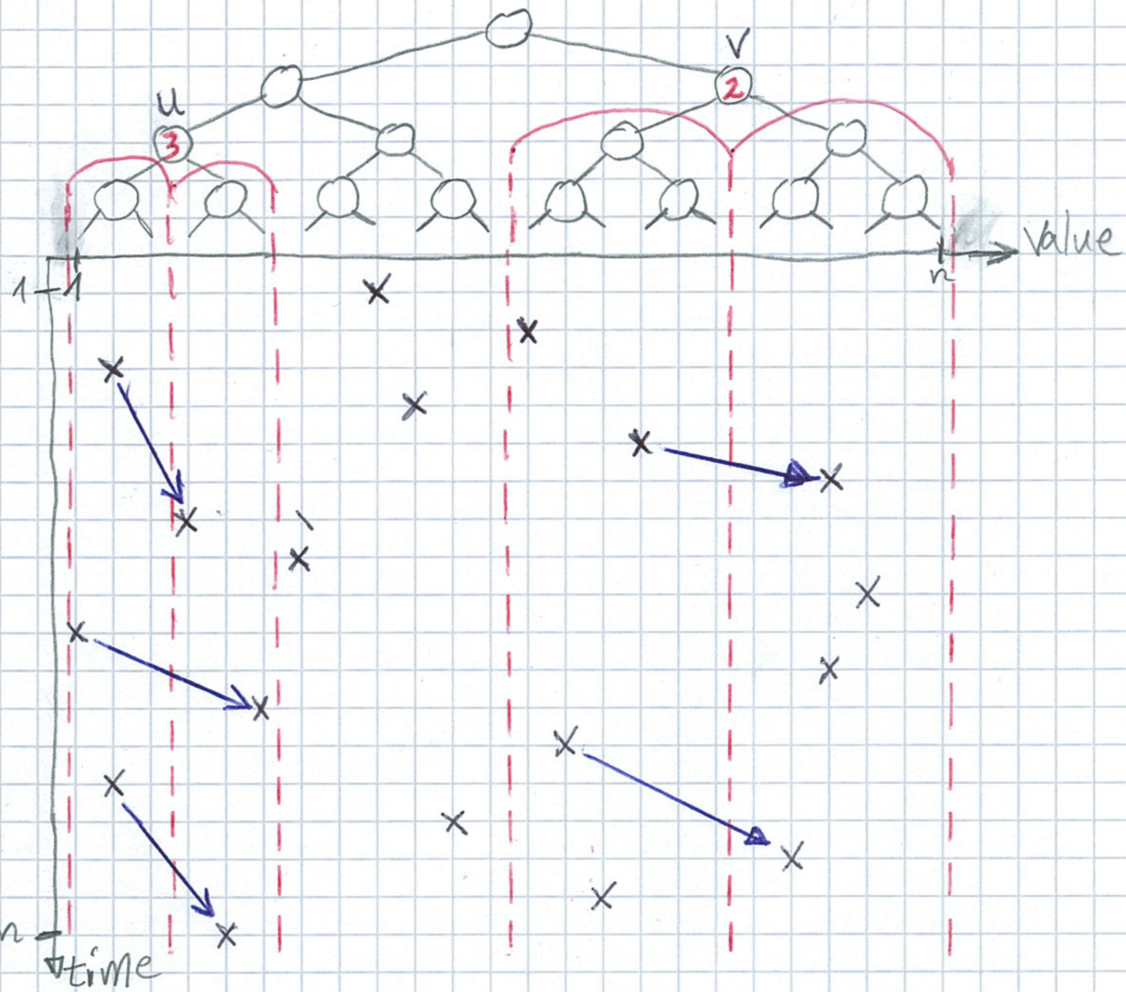
because v = lowest common ances

Lemma: $\forall v$, $|A(v)| \geq$ #interleaves through node v

Proof by picture: Fig 2

Conclusion: Cost $\geq \sum_v$ #interleaves through v

Fig1



"x" mark access sequence $(x_1, \dots, x_m)$

there are 3 interleaves through u ; 2 interleaves through v

Fig 2