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 $S = \{1, 2, \ldots, n\}$ for concreteness
  - must maintain a dynamic BST with elements in $S$
    - can be changed through rotations
  - only operation: search($x$), where $x \in S$
    - Cost = Running Time = # nodes touched by search + # rotations performed

Goals:
- given a sequence $\bar{x} = (x_1, \ldots, x_m)$, find $OPT(\bar{x})$
  - optimal cost to run search($x_1$), search($x_2$), ..., search($x_m$) in BST model
  - Examples:
    - $OPT(1, 2, \ldots, n) = \Theta(n)$
  - $E[OPT(\bar{x})] \leq 2 (m \lg n)$ for random $\bar{x} \in [n]^m$
  - Wanted: $d$-approximation algorithm

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 where we understand the worst-case well.

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

Results:
- $O(lg \lg n)$-competitive = trivial
- Tango trees: $O(lg \lg n)$ competitive, $O(lg lg n)$ approximation
  - today's topic
- Conjecture: splay trees $O(1)$-competitive [Sleator, Tarjan '83]
- Conjecture: a greedy algorithm is $O(1)$-approx [Lucas '88, Munro '00]
  - $\text{if so } \Rightarrow \text{ online } O(1)$-competitive [DHJKP '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 [Wilber '86]

- Plot sequence \( X \) on time/value diagram
- Build a tree over values → "lower bound tree" (arbitrary, static)
- "Interleave thru a node" = switching from left subtree to right subtree

Lower bound = \( \sum \) all nodes (number of interleave)

Proof: Later

Tango trees

Cost = (Wilber I) \( \times \) \( O(\log \log n) \)

Each node has a preferred 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 \( \leq \log n \) nodes \( \Rightarrow \) \( O(\log \log n) \) time/operation
   - # preferred paths visited = Wilber I

Implementation details:

- Preferred path

  \[ \text{we want to break this link in preferred decompo} \]

  \[ \begin{align*}
  1. & \text{split after successor(x)} \\
  2. & \text{split before parent successor(x)} \\
  3. & \text{insert preferred path before successor(x)}
  \end{align*} \quad (\text{split; concatenate; concatenate}) \]

- Successor found by search(x)

- Build a BST over these elements
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 $A[1..n]$
- update $(i, \Delta)$: change $A[i] \leftarrow A[i] + \Delta$
- query $(i)$: return $\sum_{j=1}^{i} A[j]$

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

Given $X = (x_1, \ldots, x_m)$ execute: update $(x_1, \Delta_1)$, query $(x_1)$; update $(x_2, \Delta_2)$, query $(x_2)$, ...
- BSTs support these $2m$ operations in $O(\text{OPT}(X))$ cost (additions)

Given $X = (x_1, \ldots, x_m)$, the answer to query at $x_5$ is $\Delta_1 + \Delta_2 + \Delta_3 + \Delta_5$

Signatures:
- Each memory location looks like $\text{Mem}[a] = \Delta_1 + \Delta_2 + \cdots$
- Signature at $\text{Mem}[a]$: $(\Delta_1, x_1, \Delta_2, x_2, \ldots)$

Wilber I: for each node $v \in$ lowerbound tree, let
- $A(v) = \{ \text{all additions "} M_1 + M_2 \text{" with } \text{Signature}(m_1) \in [a,b] \times [1,m] \}$
- $\text{Signature}(m_2) \in [b,c] \times [1,m]$

$A(v)$ for all $v$ is a partition of all additions executed by code $\implies$ cost $= \sum_{v} |A(v)|$

Conclusion: $\text{Cost} \geq \sum_{v} \# \text{interleaves through } v$

Proof by picture: Fig 2
Fig. 1

"X" mark access sequence \((X_1, \ldots, X_m)\).

There are 3 interleaves through \(U\); 2 interleaves through \(V\).

Fig. 2

Say there are 4 interleaves.

There must be 3 elements in those boxes.

To get the query answers right, must add \(\mu M_1 + M_2\).

\(\text{While} \Rightarrow \text{signature}(M_1) \in \text{curly box}\).

\(\text{signature}(M_2) \in \{b, c, \text{step}\}\).

This is trivial once you understand that signatures can only move right and down through additions.