Advanced Algorithmics

Strategies for Tackling Hard Problems

Sebastian Wild

Markus Nebel

Lecture 13

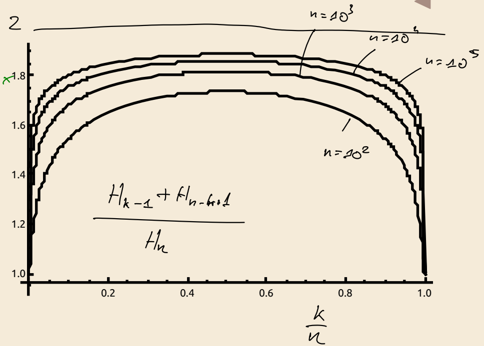
2017-06-01

Theorem 4.25 (Expected depth of kth leaf)

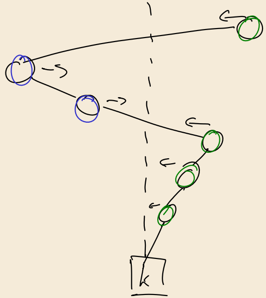
The *expected depth* of the *kth external leaf* (for $k = 1, \dots, n+1$) in a random BST on $n \geq 1$ keys is $H_{k-1} + H_{n-k+1}$.

$1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{k-1}$

depth($\boxed{k}$) = # comps for unsuccessful search for x which terminates in $\boxed{k}$



Proof:

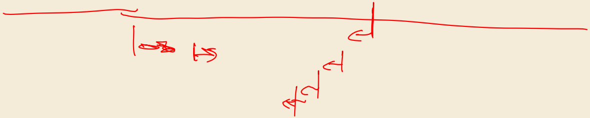


left-to-right minima among keys $> x$
total number $k-1$

left-to-right maxima among keys $< x$
total number $n-k+1$

$\mathbb{E}[\text{depth}(\boxed{k})] = H_{k-1} + H_{n-k+1}$

$\square$



Corollary 4.26 (Depth of typical leaf)

Consider a random BST T_n of n keys.

1. The *expected external path length* of T_n is
   $2(n+1)(H_{n+1}-1) \;=\; 2n\ln n - 2(1-\gamma)n \pm O(\log n)$. ($\gamma \approx 0.5772$ the *Euler-Mascheroni constant*)
2. The depth of the αnth leaf in a random BST of n keys $\sim 2\ln n$ as $n \to \infty$ for any fixed $\alpha \in (0,1)$.

$a_1, \dots, a_n$ permutation of $[n]$ $\cong$ $b_1, \dots, b_n$ $\quad b_i \in [0..n-i]$, b_i = # inversion of form $(\cdot, i)$ = # values left of i that are $> i$

a random permutation $\Leftarrow$ b_i indep., $b_i \overset{D}{=} \mathcal{U}[0..n-i]$

$\mathrm{R2LMax}(a) = \sum_{i=1}^{n} [b_i = n-i]$

Remark 4.27 (Concentration of left-to-right minima)

One can show that the number of left-to-right minima in a permutation of length n is in $O(\log n)$ w.h.p. (using general Chernoff bound).

Hence, the above expected results hold with high probability (up to constant factors). ◀

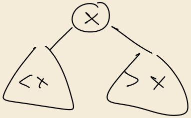
Connection to Quicksort

Previous results sounded familiar?

Recursion trees of Quicksort are also randomly generated BSTs.

- random BSTs: all insertion orders equally likely
- Quicksort trees: value of root uniformly chosen from keys in subtree

Are the shape distributions the same? Yes!



In both cases holds

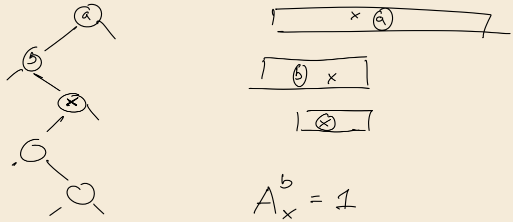
$$\Pr[T_n] = \begin{cases} 1 & n = 0 \\ \frac{1}{n} \cdot \Pr[T_L] \cdot \Pr[T_R] & n \geq 1 \end{cases}$$

i.e., the probability of a tree is computed recursively over the tree structure.

Corollary 4.28 (Recycling Quicksort results)

In a random BST holds:

- *Height* is in $\mathcal{O}(\log n)$ *w.h.p.*
  e.g., $\Pr[\text{height} \ge 42 \ln n] \le 2n^{-7.4}$
- *Expected internal path length* (= expected number of comparisons in Quicksort) is
  $2(n+1)H_n - 4n \quad = \quad 2n \ln n - 2(2-\gamma)n \pm \mathcal{O}(\log n)$.



Depth of Internal Nodes

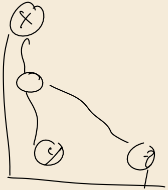
Previous results mostly for *external leaves*; how about *internal nodes*?

Similarly possible based on handy notion:

Lemma 4.29 (Ancestor indicators)

Let T_n be a random BST with keys $[n]$ and denote by $A^x_y = [x \text{ is a proper ancestor of } y]$ for $x, y \in [n]$. (This means $A^x_x = 0$ and for $x \neq y$, $A^x_y = 1$ iff x lies on the path from the root to y.)
Then holds:

1. $A^x_y = 1$ iff x was the *first* among the keys $[x..y] \cup [y..x]$ that was inserted into T_n.
2. $A^x_y = 1$ iff x and y are *directly compared* by randomized Quicksort during a partitioning step using pivot x.
3. $\Pr[A^x_y = 1] \;=\; \Pr[A^y_x = 1] \;=\; \dfrac{1}{|y-x|+1}$ for $x \neq y$.



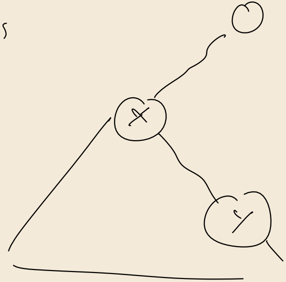
Remark 4.30 (Common ancestor indicators)

Idea generalizes to $C^x_{y,z} = [x \text{ is common ancestor of } y \text{ and } z]$:

$$\Pr[C^x_{y,z} = 1] \;=\; \frac{1}{\max\{x,y,z\} - \min\{x,y,z\} + 1}.$$

Proofs

A_y^x

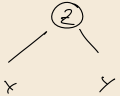


$x < y$ wlog.

which of the keys $x, x+1, \dots, y$ was inserted into tree

- y first $\leadsto$ x cannot be on path to y $\Rightarrow A_y^x = 0$
- $z \in (x+1, \dots, y-1)$ first

  $\Rightarrow$

  $\leadsto A_y^x = 0$
- x first: at x's insertion, $[x+1 .. y]$ correspond to the same gap

  $\leadsto$ all $[x+1 .. y]$ in (x)'s right subtree

  $\Rightarrow A_y^x = \underline{1}$

Theorem 4.31 (Expected depth of kth node)

The *expected depth* of the *kth internal node* (for $k = 1, \ldots, n$) in a random BST on $n \geq 1$ nodes is $H_k + H_{n-k+1} - 2$.

$$\text{depth}(k) = \sum_{x=1}^{n} A_k^x \quad \text{(not indep!)}$$

Recall: $\mathbb{E}[\text{depth of } k\text{th leaf}] = H_{k-1} + H_{n-k+1}$.

depth($\boxed{k}$)

$$\mathbb{E}\{\text{depth}(k)\} = \sum_{x=1}^{n} \mathbb{E}[A_k^x] = \sum_{x=1}^{n} \Pr(A_k^x = 1) = \sum_{x=1}^{k-1} \frac{1}{k-x+1} + \sum_{x=k+1}^{n} \frac{1}{x-k+1}$$

$$= \sum_{i=2}^{k} \frac{1}{i} + \sum_{i=2}^{n-k+1} \frac{1}{i} = H_k + H_{n-k+1} - 2$$

Remark 4.32 (Expected subtree size)

The *expected size* of the *subtree* rooted at the kth internal node is also $H_k + H_{n-k+1} - 2$. $+1$

$$\sum_{x=1}^{n} A_x^k$$

Remark 4.33 (Further Results)

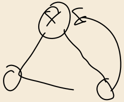
Random BSTs are extremely well-studied. A few more results:

- The **expected height** is $\alpha \ln n - \beta \ln\ln n \pm \mathcal{O}(1)$ with $\alpha \approx 4.311$ and $\beta \approx 1.953$.
- The **height** divided by $\ln n$ **converges in probability** to the constant α.
- The number X_{nk} of **external leaves at depth** k satisfies $\mathbb{E}[X_{nk}] = \frac{2^k}{n!}\left[{n \atop k}\right]$.
- The **depth** of a typical **leaf** divided by $\ln n$ **converges in probability** to 2.
- The standardized **depth** of a random leaf **converges** in distribution to a standard **normal distribution**.
- The same is true for the standardized depth of a random internal node.
- Let D_n be the **depth of the** n**th inserted node**. Then $(D_n - \ln n)/\sqrt{\ln n}$ converges in distribution to a standard **normal distribution**.

$\rightsquigarrow$ *plain BSTs have* ***great*** *performance* ***if*** *insertions come in random order.*

Interesting fact: *no longer true* if there are *deletions*!
After long sequence of random inserts and deletes: expected height $\Theta(\sqrt{n})$, not $\Theta(\log n)$ (!)

Reason: Hibbard's deletion algorithm destroys randomness!



Animations: `http://algs4.cs.princeton.edu/32bst`

Need for Randomization

"Defects" of plain BSTs:

1. linear worst case height
2. many deletions have negative impact

Classic deterministic strategies to avoid **worst case**: *balanced BSTs*

- **height-balanced trees:** AVL-trees, 2-3-trees / B-trees, red-black trees, scapegoat trees, ...
- **weight-balanced trees:** BB(α)-trees, ...
- **self-balancing trees:** splay trees, ...

All use somewhat sophisticated rotation / rebalancing schemes ...
can we achieve **similar performance** using **simpler randomized** data structure?

Treaps

Observation: The *preorder* (sequence of the keys) is a *1:1 characterization* of a given BST since

- each BST has unique preorder, and
- each preorder generates a unique tree by inserting keys in preorder into an initially empty tree.

$\leadsto$ Enforcing the preorder corresponding to a random BST suffices to avoid worst cases.
. . . but we have no control over the set of keys to be inserted.

Idea: Separate *key values* from *rank in preorder* using random **priorities**.

Definition 4.34 (Treaps)

Let $S = \{(k_1, p_1), \ldots, (k_n, p_n)\}$ be a set of *key-priority pairs* where $k_i \in K$ and $p_i \in [0, 1]$ for K some totally ordered universe.
A *treap* for S is a binary tree with n internal nodes labeled by the key-priority pairs so that

1. the search tree property holds w.r.t. the keys, and
2. the heap property holds w.r.t. the priorities.

Theorem 4.35 (Treaps are unique)

Let S be a set of n key-priority pairs where all keys and all priorities are distinct. Then there is *exactly one treap* for S.

Proof: Existence: Insert keys in decreasing-priority order into an initially empty BST ← with keys

→ BST w.r.t. keys

→ heap w.r.t. priorities.

Uniqueness: by induction on n

$n = 1$ ✓

$n \geq 2$ root uniquely determ. as k_i with

$p_i = \max_j p_j$

→ subtrees unique (IH)

□.

Definition 4.36 (Randomized Treaps)

A *randomized treap* is the unique treap that results from given keys $k_1, k_2, \ldots$ where (upon insertion) we assign k_i a priority $p_i \overset{\mathcal{D}}{=} \mathcal{U}(0,1)$ independent of all previous priorities.

Theorem 4.37 (Shape of randomized treaps)

The (random) shape of a randomized treap for n keys has the *same distribution* as random BST with n keys.

$$\Pr[\text{treap shape } T_n] = \begin{cases} 1 & n \le 1 \\ \frac{1}{n} \cdot \Pr[T_\ell] \cdot \Pr[T_r] & n \ge 2 \end{cases}$$

Corollary 4.38 (Search Costs)

All results for random BSTs apply, in particular:

- Expected search costs (#comparisons) $< 2 \ln n + 1$.
- Search costs in $\mathcal{O}(\log n)$ w.h.p.

Insertions and Deletions in Randomized Treaps

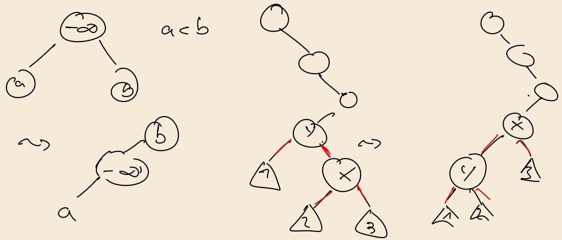
Up to now: *static* view on treaps.

But can we efficiently turn a randomized treap for $k_1, \ldots, k_n$ into one for $k_1, \ldots, k_{n+1}$?
And vice versa?

Yes!

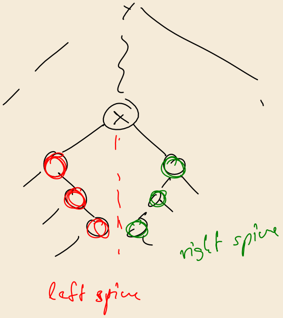
- **Insert:** Start as in plain BST, then *rotate up* until heap property holds.
- **Delete:** Rotate node down (as if priority was $-\infty$) until it is a leaf, then remove it.

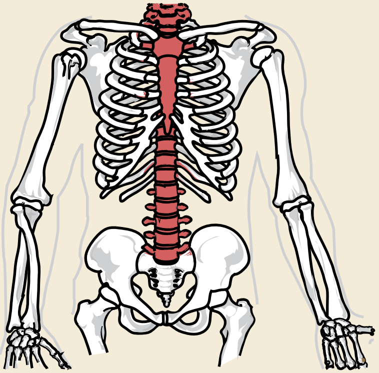
Conceptually very simple!



$\rightsquigarrow$ *all operations in* $\mathcal{O}(\log n)$ *time w.h.p.!*

Spines of Trees





Lemma 4.39 (Bound on Rotations)

The number of *rotations* to insert or delete a node x in a randomized treap is at most $LS(x) + RS(x)$, where $LS(x)$ and $RS(x)$ are the *lengths of the left resp. right spine* of (the subtree of) x in the treap (after insertion resp. before deletion).

Lemma 4.40 (Expected Spine Lengths)

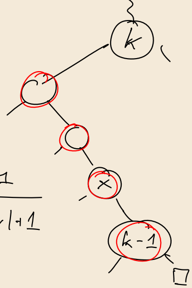
The expected length of the left and right spine of (the subtree of) the kth internal node (for $k = 1, \ldots, n$) in random BST of n keys are given by

$$\mathbb{E}[LS(k)] = 1 - \frac{1}{k} \qquad \mathbb{E}[RS(k)] = 1 - \frac{1}{n-k+1}$$

$$LS(k) = \sum_{x=1}^{k-1} \left(A^{x}_{k-1} - C^{x}_{k-1,k} \right)$$

$$A^{k}_{k} = 1$$

$$\Pr[A^{x}_{y} = 1] = \frac{1}{|x-y|+1}$$



$$\mathbb{E}[LS(k)] = \sum_{x=1}^{k-1} \frac{1}{k-x} - \frac{1}{k-x+1}$$

$$= \sum_{i=1}^{k-1} \frac{1}{i} - \frac{1}{i+1}$$

$$= 1 - \frac{1}{k}$$

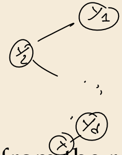
□

Randomized BSTs

Weaknesses of treaps:

- priorities *fixed once and for all* $\rightsquigarrow$ never recovers from bad luck
- have to store *priorities* (at least in a direct implementation), but these are *not helpful* algorithmically.

Recall: Key property in random BSTs is that in every subtree of size m, each key value is the root of the subtree with probability $1/m$.



Idea of RBSTs: enforce this property *anew* after *each* insertion / deletion!
Store in each node x the size of its subtree $S(x)$.

- **Insert:** Insert x as new leaf and let $y_1, \ldots, y_d$ be the nodes on the path from the root. For each y, x should have a $1/S(y)$ chance to replace y as the subtree root.
- **Delete:** After x is gone, one of the remaining $S(x) - 1$ nodes must become subtree root. $\rightsquigarrow$ choose one of x's children y and z with probabilities $\frac{S(y)}{S(y)+S(z)}$ resp. $\frac{S(z)}{S(y)+S(z)}$.

Benefits: Tree occasionally rebuilt, subtree sizes useful for rank-based operation.