Advanced Algorithmics

Strategies for Tackling Hard Problems

Sebastian Wild

Markus Nebel

Lecture 14

2017-06-05

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 $S(x)$.

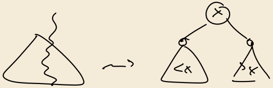
- **Insert:** Insert x as new leaf and let $y_1, \ldots, y_d$ be the nodes on the path from the root. For each y, x should have a $1/S(y)$ chance to replace y as the subtree root.
- **Delete:** After x is gone, one of the remaining $S(x) - 1$ nodes must become subtree root. $\rightsquigarrow$ choose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

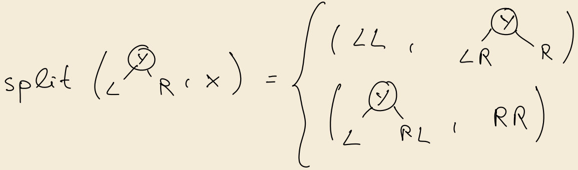
Benefits: Tree occasionally rebuilt, subtree sizes useful for rank-based operation.

Insert in RBSTs

```
Node insert(Node root, int x)          ⟨ base case for null
    n = root.size;
    r = U[0..n] // uniform int between 0 and n
    if (r == n) return insertAtRoot(root,x)      // p = 1/(n+1)
    if (x < root.key)
        root.left = insert(root.left, x)
    else
        root.right = insert(root.right, x)
    return root   // update S(root)

Node insertAtRoot(Node root, int x)
    (smallerRoot,largerRoot) = split(root, x)
    return new Node(x, smallerRoot, largerRoot)
```





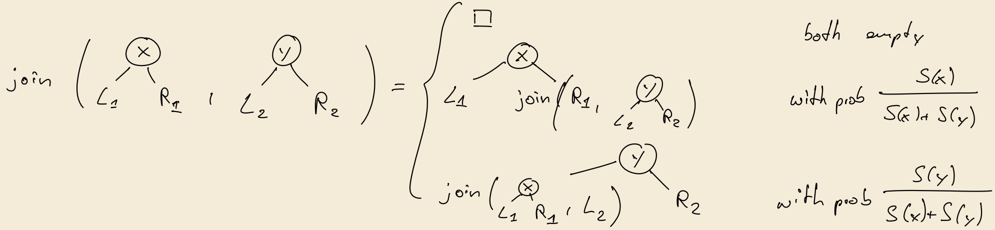
$x < y$, $(LL, LR) = split(L, x)$

$x > y$, $(RL, RR) = split(R, x)$

Delete in RBSTs

```
Node delete(Node root, int x)
    if (root == null) return null
    if (x < root.key)
        root.left = delete(root.left, x)
    else if (x > root.key)
        root.right = delete(root.right, x)
    else // must delete root
        return join(root.left, root.right)
```

update $S(root)$



Theorem 4.41 (Correctness)

Any sequence of insert and delete operations results in a tree whose shape is that of a *random BST*.

Skip detailed proof; inductively show that split, join preserve randomness, then insert & delete

Theorem 4.42 (Operation Costs)

The costs (#visited nodes) of operations in RBSTs on n keys are

- same as in random BSTs for **(un)successful search**, i.e., $\sim 2 \ln n$ in expectation and $\mathcal{O}(\log n)$ w.h.p.;
- **insert** additionally needs $\mathcal{O}(1)$ in expectation during insertAtRoot / split, and
- **delete** needs $\mathcal{O}(1)$ expected cost in join.

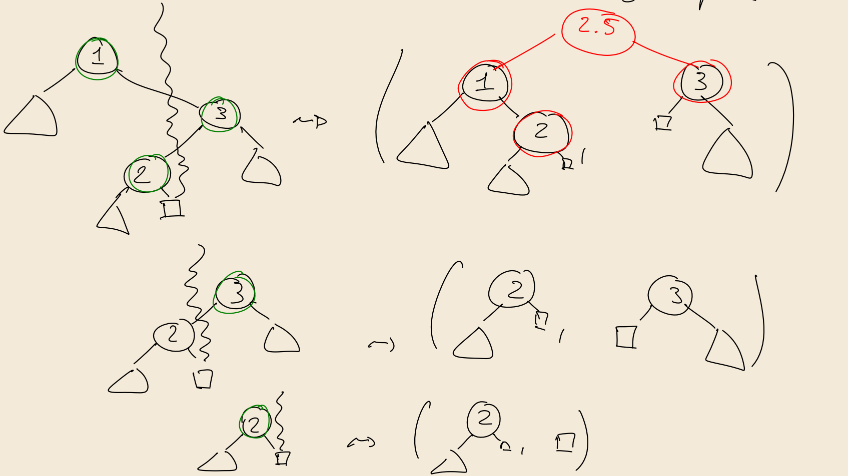
Remark 4.43

Split and join are also helpful operations in their own right.

Proof: Thm 4.42

Search: follows from Thm 4.41 (and results for random BSTs)

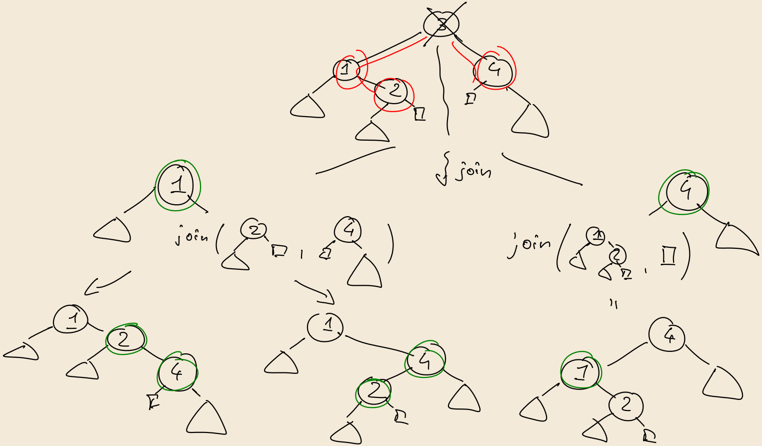
Insert: Split in the resulting tree, touched nodes are exactly nodes on left & right spine



↝ Insert touches exactly nodes on spine after insert

≤ 2 in expectation

<u>Delete / Join</u> In <u>original</u> tree, touched nodes are at most the nodes on the spines



□

Abundance of Witnesses: Primality Testing

non determinism NP

with many certificates $\leadsto$ so just guess

PRIMES $\in$ co-NP

$\in$ NP

$\in$ P

When can we hope to guess a certificate

(1) certificates must be easy to check "vce"

(2) the set of all candidates must contain many certificates

How about $\mathrm{Cand}(n) = \{2, \ldots, n-1\}$ $\qquad$ n given in binary

$\mathrm{Wit}(n) = \{a : a \mid n\}$

$\rightsquigarrow$ Guess a, check $a \mid n$

$n = p \cdot q \quad \rightsquigarrow \quad 2$ witnesses $\quad \frac{2}{n-2}$ prob.

$\rightsquigarrow \quad \Omega(n)$ samples $\qquad$ (pseudopolynomial)

Theorem 4.44 (Fermat's Little Theorem)
For p a prime and $a \in [1..p-1]$ holds

$$a^{p-1} \equiv 1 \pmod{p} \qquad (*)$$

Pick random a, if $a^{p-1} \bmod p \neq 1 \rightsquigarrow p$ not prime

Problem: Carmichael numbers also fulfill $(*)$, but are not prime
↳ ∞ many!

Theorem 4.45 (Euler's Criterion)

Let $p > 2$ an odd number.

$[1 .. p-1]$ // $-1 \pmod p$ ///

$$p \text{ prime} \iff \forall a \in \mathbb{Z}_p \setminus \{0\} : a^{\frac{p-1}{2}} \bmod p \in \{1, p-1\}$$

Theorem 4.46 (Number of Witnesses)

For every odd $n \in \mathbb{N}$, $(n-1)/2$ odd, we have:

1. If n is prime then $a^{(n-1)/2} \bmod n \in \{1, n-1\}$, for all $a \in \{1, \ldots, n-1\}$. ✓
2. If n is not prime then $a^{(n-1)/2} \bmod n \notin \{1, n-1\}$ for *at least half* of the elements in $\{1, \ldots, n-1\}$.

Proof Idea:

$\text{Witness}(n) = \{ a : a^{\frac{n-1}{2}} \bmod n \notin \{1, -1\} \}$

$R(n) = [1 .. n-1] \setminus \text{Witness}(n)$

$\exists$ injective function $h : R \to \text{Witness}$

Simple Solovay-Strassen Primality Test

Input: an odd number n with $(n-1)/2$ odd.

1. Choose a random $a \in \{1, 2, \ldots, n-1\}$.
2. Compute $A := a^{(n-1)/2} \bmod n$.
3. If $A \in \{1, n-1\}$ then output "n probably prime" (reject);
4. otherwise output "n not prime" (accept).

Theorem 4.47 (Correctness)

The simple Solovay-Strassen algorithm is a polynomial OSE-MC algorithm to detect composite numbers n with $n \bmod 4 = 3$. ◀

drop this is possible, $n \bmod 4 = 1$ still fulfills Thm 4.46

Corollary 4.48

For positive integers n with $n \bmod 4 = 3$ the simple Solovay-Strassen algorithm provides a polynomial TSE-MC algorithm to detect prime numbers. ◀

Sampling Primes

RANDOMPRIME(ℓ, k) Input: $\ell, k \in \mathbb{N}$, $\ell \geq 3$.

1. Set X :="not found yet"; $I := 0$;
2. `while` X = "not found yet" and $I < 2\ell^2$ `do`
   - generate random bit string $a_1, a_2, \ldots, a_{\ell-2}$ and
   - compute $n := 2^{\ell-1} + \sum_{i=1}^{\ell-2} a_i \cdot 2^i + 1$

     // This way n becomes a random, odd number of length ℓ
   - Realize k independent runs of Solovay-Strassen-algorithm on n;
   - `if` at least one output says "$n \notin PRIMES$" `then` $I := I + 1$

     `else` X :="PN found"; `output` n;
3. `if` $I = 2 \cdot \ell^2$ `then output` "no PN found".

$\rightsquigarrow$ Thus RandomPrime(ℓ, ℓ) is TSE-MC poly-time for generating primes of length n

Proof Sketch: Prime Number Theorem: $\pi(x) \sim \frac{x}{\ln(x)}$

$\pi(x) = \#\text{primes} \leq x$

Bertrand's Postulate: $\forall n \ \exists p: n \leq p \leq 2n$

We draw uniform from $[2^{\ell-1}, 2^{\ell}-1]$

$$\Pr[\text{prime}] \approx \frac{\pi(x)}{x} = \frac{1}{\ln(x)} \sim \frac{1}{\ln(2^{\ell})} = \frac{1}{\ell}$$

$$\Pr[\text{"Prime"} \mid \text{no prime hit}] \leq \left(\frac{1}{2}\right)^{k}$$

$$\approx 0 \quad \text{for large } k = \ell$$

Number of iteration to obtain prime $\overset{D}{=}$ Geo$(1/\ell)$

$\Rightarrow \mathbb{E}[\text{iterations}] = \ell$

We do $2\ell^2$

$$\Pr\{\#\text{ iterations} > \underbrace{a}_{= 2\ell^2}\} \le \frac{\mathbb{E}\{\#\text{iter}\} = \ell}{a} = \frac{1}{2\ell}$$

Running time $O(\ell^5)$.

Fingerprinting: Hashing

$n = |S|$

universe U want to store $S \subseteq U$

$|U| \gg |R|$ function $h: U \to R$

$\quad\quad\quad\quad = m$ $\quad\quad\quad\quad = $ set of bins

$\rightsquigarrow$ mathematical is to throw n balls into m bins

1. Random model for worst case for uniform hashing

   assumption

   all hash sequences

   $h(x_i)$ independent of all

   $h(x_1), \ldots, h(x_{i-1})$

2. Universal Hashing

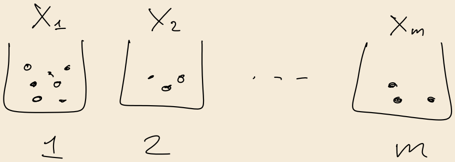
3. Perfect Hashing

Uniform Hashing – Balls into Bins

uniform hashing assumption

hash values are iid uniform

$\rightsquigarrow$ best-possible hash function



m bins

n balls

$X_1 \overset{\mathcal{D}}{=} \cdots \overset{\mathcal{D}}{=} X_m \overset{\mathcal{D}}{=} \mathrm{Bin}(n, 1/m)$

not independent

$\sum X_i = n$

$\Rightarrow \mathbb{E}[X_1] = \ldots = \mathbb{E}[X_m] = \frac{n}{m}$

single X_i concentrated around $\frac{n}{m}$ for large n

Theorem 4.49

$n = m \rightsquigarrow \mathbb{E}[X_i] = 1$

If we throw n balls into n bins, then the *fullest bin* has $O(\log n / \log\log n)$ balls w.h.p.

$\hat{=} \max X_i$

Implications:

1. Hashing never too bad, even for w.c bin, better than BST
2. If worst case is important, typically hardly better BST

Proof: $\Pr[\max X_i \ge M] \underset{\text{union bound}}{\le} n \Pr[X_1 \ge M]$

$$\Pr[X_1 \ge M] = \Pr\Big[\bigcup_{\substack{I \subseteq [n] \\ |I| = M}} \text{balls } i \in I \text{ land in bin } 1\Big]$$

$$\le \binom{n}{M} \Pr\big[|I| \text{ balls land in bin } 1\big]$$

$$\leq \binom{n}{M} \left(\frac{1}{m}\right)^M \qquad n = m$$

$$= \frac{n!}{M!\,(n-M)!} \cdot \frac{1}{n^M} \quad \leq 1$$

$$\frac{n(n-1)\cdots 1}{\underbrace{n \cdot n \cdot n \cdots n}_{M}\,(n-M)(n-M-1)\cdots 1} \leq 1$$

$$\leq \frac{1}{M!}$$

$$M! \geq \left(\frac{M}{e}\right)^M \sqrt{2\pi M}$$

Stirlings formula

$$(*) \qquad \leq \left(\frac{e}{M}\right)^M \frac{1}{\sqrt{2\pi M}} \quad \leq 1$$

to show: $\forall d\ \exists c$ $\quad = O(n^{d})$

$$\Pr\left[\max X_i \geq c \cdot \frac{\ln(n)}{\ln(\ln(n))}\right]$$

$$\boxed{c \geq 3}$$

$$\leq n \cdot \Pr\left[X_1 \geq c\,\frac{\ln n}{\ln \ln n}\right]$$

$$\underset{(*)}{\leq} n \cdot \left(\frac{e}{c} \cdot \frac{\ln \ln n}{\ln n}\right)^{c \frac{\ln n}{\ln \ln n}}$$

$\frac{e}{c} \leq 1$

$$= \exp\left(\ln(n) + c \cdot \frac{\ln n}{\ln\ln n} \ln\left(\frac{\ln\ln n}{\ln n}\right)\right)$$

$$= \exp\left(\ln n + \ln n \frac{c \cdot \ln\ln\ln n}{\ln\ln n} - c \ln n \cdot \frac{\cancel{\ln\ln n}}{\cancel{\ln\ln n}}\right)$$

$$= \exp\left((2-c)\ln(n) - \ln(n) + \ln(n) \frac{c \ln\ln\ln n}{\ln\ln n}\right)$$

$$= n^{2-c} \underbrace{\exp\Big(\ln(n)\Big(\underbrace{\frac{c \ln\ln\ln n}{\ln\ln n}}_{\substack{o(1) \\ \Rightarrow\ \leq 1 \text{ for large } n}} - 1\Big)\Big)}_{\leq 1 \text{ for large } n}$$

$$\leq n^{2-c} \qquad \square.$$

for large n

$n^d \quad \rightsquigarrow \quad c = d+2$