Searching with Dice

A survey on randomized data structures

Conrado Martínez

Univ. Politècnica de Catalunya, Spain

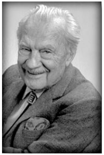
May 11th, 2010

Journée inaugurale de l'équipe

Combinatoire, algorithmique et interactions (CALIN)

Introduction

R. Karp



N. C. Metropolis

M. O. Rabin

The usefulnees of randomization in the design of algorithms has been known for a long time:

- Metropolis' algorithms
- Rabin's primality test
- Rabin-Karp's string search

Introduction

M.N. Wegman

- Hashing is another early success of randomization for the design of data structures.
- Selecting the hash function from a universal class (Carter and Wegman, 1977) guarantees expected performance

Introduction

Randomization yields algorithms:

- Simple and elegant
- Practical
- With guaranteed expected performance
- Without assumptions on the probabilistic distribution of the input

Introduction

- The usual worst-case analysis is not useful for randomized algorithms
- The probabilistic model to use in the analysis is under control; it is not a working hypothesis, but built-in

Introduction

In this talk:

- Skip lists
- Randomized binary search trees

Skip lists

W. Pugh

- Skip lists were invented by William Pugh (C. ACM, 1990) as a simple alternative to balanced trees
- The algorithms to search, insert, delete, etc. are very simple to understand and to implement, and they have very good expected performance—independent of any assumption on the input

Skip lists

W. Pugh

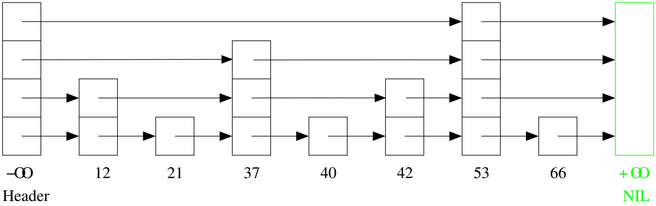
- Skip lists were invented by William Pugh (C. ACM, 1990) as a simple alternative to balanced trees
- The algorithms to search, insert, delete, etc. are very simple to understand and to implement, and they have very good expected performance—independent of any assumption on the input

Skip lists

A skip list S for a set X consists of:

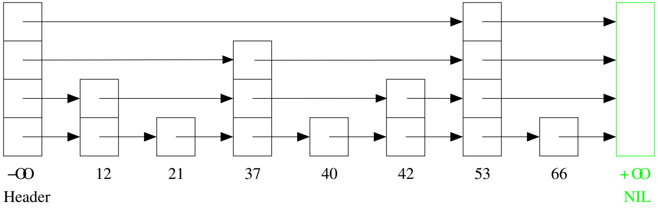
1. A sorted linked list L_1, called level 1, contains all elements of X
2. A collection of non-empty sorted lists $L_2, L_3, \ldots,$ called level 2, level 3, ... such that for all $i \geq 1$, if an element x belongs to L_i then x belongs to L_{i+1} with probability p, for some $0 < p < 1$

Skip lists



To implement this, we store the items of X in a collection of nodes each holding an item and a variable-size array of pointers to the item's successor at each level; an additional dummy node gives access to the first item of each level

Skip lists



To implement this, we store the items of X in a collection of nodes each holding an item and a variable-size array of pointers to the item's successor at each level; an additional dummy node gives access to the first item of each level

Skip lists

- The level or height of a node x, height(x), is the number of lists it belongs to.
- It is given by a geometric r.v. of parameter p:

$$\Pr\{\text{height}(x) = k\} = pq^{k-1}, \qquad q = 1 - p$$

Skip lists

- The level or height of a node x, height(x), is the number of lists it belongs to.
- It is given by a geometric r.v. of parameter p:

$$\Pr\{\text{height}(x) = k\} = pq^{k-1}, \qquad q = 1 - p$$

Skip lists

- The height of the skip list S is the number of non-empty lists,

$$\text{height}(S) = \max_{x \in S}\{\text{height}(x)\}$$

- The random variable H_n giving the height of a random skip list of n is the maximum of n i.i.d. Geom(p)
- Several performance measures of skip lists are expressed in terms of the probabilistic behavior of a sequence of n i.i.d. geometric r.v. of parameter p

Skip lists

- The height of the skip list S is the number of non-empty lists,

$$\text{height}(S) = \max_{x \in S}\{\text{height}(x)\}$$

- The random variable H_n giving the height of a random skip list of n is the maximum of n i.i.d. Geom(p)
- Several performance measures of skip lists are expressed in terms of the probabilistic behavior of a sequence of n i.i.d. geometric r.v. of parameter p

Skip lists

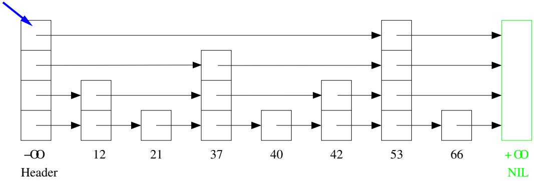
- The height of the skip list S is the number of non-empty lists,

$$\text{height}(S) = \max_{x \in S}\{\text{height}(x)\}$$

- The random variable H_n giving the height of a random skip list of n is the maximum of n i.i.d. Geom(p)
- Several performance measures of skip lists are expressed in terms of the probabilistic behavior of a sequence of n i.i.d. geometric r.v. of parameter p

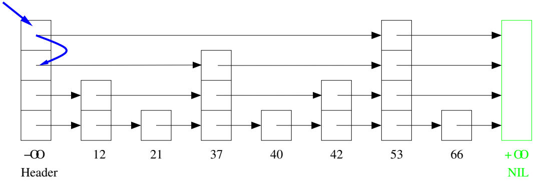
Searching in a skip list

Searching for an item x, $42 < x \leq 53$



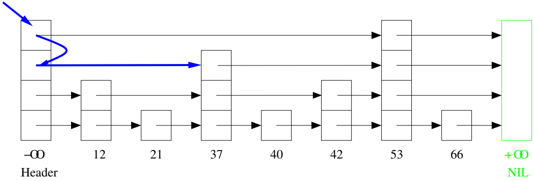
Searching in a skip list

Searching for an item x, $42 < x \leq 53$



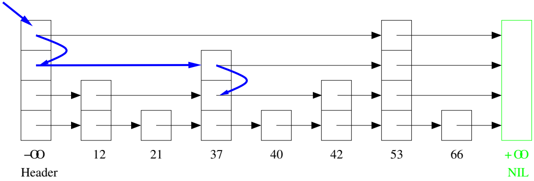
Searching in a skip list

Searching for an item x, $42 < x \leq 53$



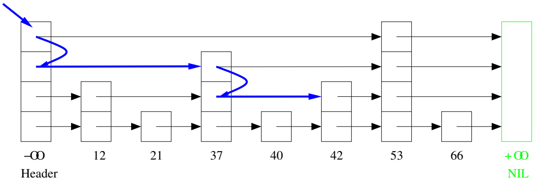
Searching in a skip list

Searching for an item x, $42 < x \leq 53$



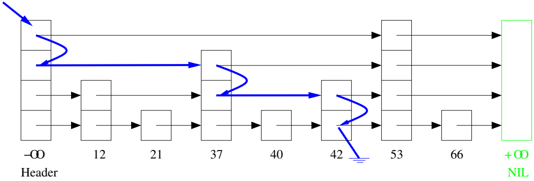
Searching in a skip list

Searching for an item x, $42 < x \leq 53$



Searching in a skip list

Searching for an item x, $42 < x \leq 53$

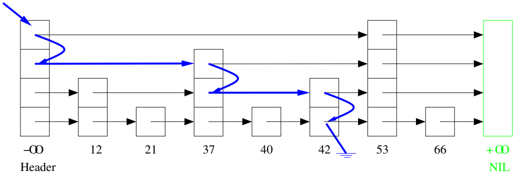


Searching in a skip list

```
procedure Search(S, x)
    p ← S.header
    ℓ ← S.height
    while ℓ ≠ 0 do
        if p.item < x then
            p ← p.next[ℓ]
        else
            ℓ ← ℓ − 1
```

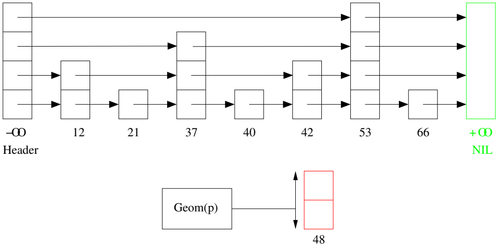
Insertion in a skip list

Inserting an item $x = 48$



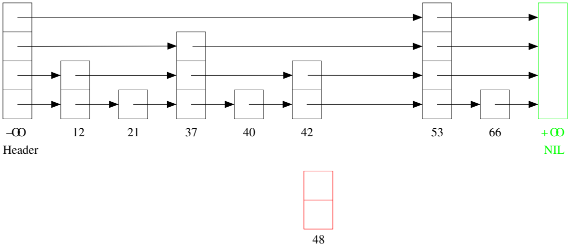
Insertion in a skip list

Inserting an item $x = 48$



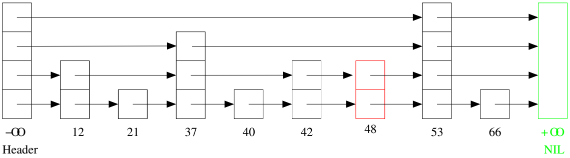
Insertion in a skip list

Inserting an item $x = 48$



Insertion in a skip list

Inserting an item $x = 48$



Performance of skip lists

- The cost of insertions, deletions and searches is essentially that of searching, with

$$\text{Cost of search} = \#\text{ of forward steps} + \text{height}(S)$$

- More formally, with $X = \{x_1, x_2, \dots, x_n\}$, $x_0 = -\infty < x_1 < \cdots < x_n < x_{n+1} = +\infty$, for $0 \le k \le n$,

$$C_{n,k} = F_{n,k} + H_n \qquad \text{cost of searching a key in } (x_k, x_{k+1}]$$

$$F_{n,k} = \#\text{ of forward steps to } (x_k, x_{k+1}]$$

$$H_n = \text{height of the skip list}$$

Performance of skip lists

- The cost of insertions, deletions and searches is essentially that of searching, with

$$\text{Cost of search} = \#\text{ of forward steps} + \text{height}(S)$$

- More formally, with $X = \{x_1, x_2, \ldots, x_n\}$, $x_0 = -\infty < x_1 < \cdots < x_n < x_{n+1} = +\infty$, for $0 \leq k \leq n$,

$$C_{n,k} = F_{n,k} + H_n \qquad \text{cost of searching a key in } (x_k, x_{k+1}]$$

$$F_{n,k} = \#\text{ of forward steps to } (x_k, x_{k+1}]$$

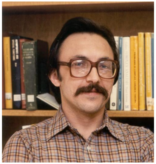
$$H_n = \text{height of the skip list}$$

Analysis of the height

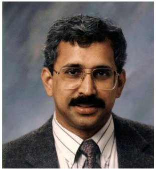
$$\begin{aligned}
a_i &= \mathsf{height}(x_i) \sim Geom(p) \\
H_n &= \mathsf{height}(S) = \max\{a_1, \ldots, a_n\} \\
\mathbb{E}[H_n] &= \sum_{k>0} \Pr\{H_n > k\} = \sum_{k>0} (1 - \Pr\{H_n \leq k\}) \\
&= \sum_{k>0} \left(1 - \prod_{1\leq i\leq n} \Pr\{a_i \leq k\}\right) = \sum_{k>0} (1 - (\Pr\{a_i \leq k\})^n) \\
&= \sum_{k>0} \left(1 - \left(1 - q^k\right)^n\right)
\end{aligned}$$

with $q := 1 - p$.

Analysis of the height



W. Szpankowski



V. Rego

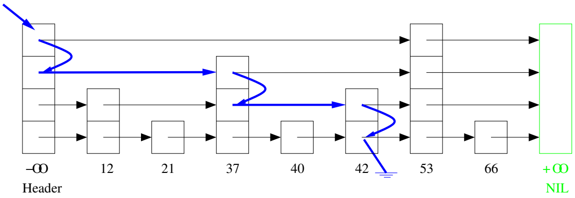
Theorem (Szpankowski and Rego,1990)

$$\mathbb{E}[H_n] = \log_Q n + \frac{\gamma}{L} - \frac{1}{2} + \chi(\log_Q n) + O(1/n)$$

with $Q := 1/q$, $L := \ln Q$, $\chi(t)$ *a fluctuation of period 1, mean 0 and small amplitude.*

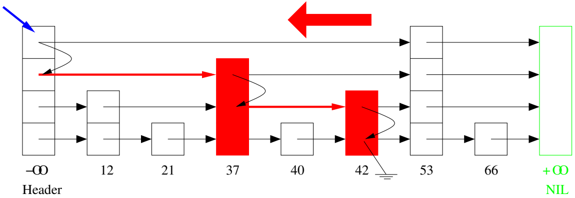
Analysis of the forward cost

The number of forward steps $F_{n,k}$ is the number of weak left-to-right maxima in $a_k, a_{k-1}, \ldots, a_1$, with $a_i = \text{height}(x_i)$



Analysis of the forward cost

The number of forward steps $F_{n,k}$ is the number of weak left-to-right maxima in $a_k, a_{k-1}, \ldots, a_1$, with $a_i = \text{height}(x_i)$



Analysis of the forward cost

- Total unsuccessful search cost

$$C_n = \sum_{0 \le k \le n} C_{n,k} = nH_n + F_n$$

- Total forward cost

$$F_n = \sum_{0 \le k \le n} F_{n,k}$$

Analysis of the forward cost

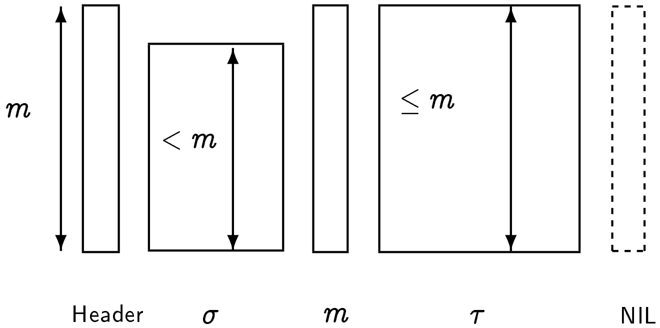
- Total unsuccessful search cost

$$C_n = \sum_{0 \le k \le n} C_{n,k} = nH_n + F_n$$

- Total forward cost

$$F_n = \sum_{0 \le k \le n} F_{n,k}$$

Analysis of the forward cost



A recursive decomposition of the skip list S

Analysis of the forward cost

- $F(S)$ = total forward cost of the skip list S
- The recursive decomposition $S = \langle \sigma, m, \tau \rangle$ gives

$$F(S) = F(\sigma) + F(\tau) + |\tau| + 1$$

- Let $\mathcal{S}^{[\text{cond}]}$ denote the set of all skip lists whose height satisfies the condition *cond*

$$F^{[\text{cond}]}(z, u) = \sum_{S \in \mathcal{S}^{[\text{cond}]}} z^{|S|} u^{F(S)} \Pr(S),$$

with

$$\Pr(S) = \Pr(\sigma) \cdot pq^{m-1} \cdot \Pr(\tau)$$

Analysis of the forward cost

- $F(S)$ = total forward cost of the skip list S
- The recursive decomposition $S = \langle \sigma, m, \tau \rangle$ gives

$$F(S) = F(\sigma) + F(\tau) + |\tau| + 1$$

- Let $\mathcal{S}^{[\text{cond}]}$ denote the set of all skip lists whose height satisfies the condition *cond*

$$F^{[\text{cond}]}(z, u) = \sum_{S \in \mathcal{S}^{[\text{cond}]}} z^{|S|} u^{F(S)} \Pr(S),$$

with

$$\Pr(S) = \Pr(\sigma) \cdot pq^{m-1} \cdot \Pr(\tau)$$

Analysis of the forward cost

- $F(S)$ = total forward cost of the skip list S
- The recursive decomposition $S = \langle \sigma, m, \tau \rangle$ gives

$$F(S) = F(\sigma) + F(\tau) + |\tau| + 1$$

- Let $\mathcal{S}^{[\text{cond}]}$ denote the set of all skip lists whose height satisfies the condition *cond*

$$F^{[\text{cond}]}(z,u) = \sum_{S \in \mathcal{S}^{[\text{cond}]}} z^{|S|} u^{F(S)} \Pr(S),$$

with

$$\Pr(S) = \Pr(\sigma) \cdot pq^{m-1} \cdot \Pr(\tau)$$

Analysis of the forward cost

- The recursion translates to

$$F^{=m}(z,u) = pq^{m-1}zu^2 F^{\le m-1}(z,u)F^{\le m}(z,u), \qquad m > 0$$
$$F^{=0}(z,u) = 1$$

- Taking derivatives w.r.t. u and setting $u = 1$, we obtain a recurrence for the GF of expectations:

$$f^{=m}(z) = \frac{2pq^{m-1}z}{[\![m-1]\!][\![m]\!]} + \frac{f^{\le m-1}(z)}{[\![m]\!]} + \frac{f^{\le m}(z)}{[\![m-1]\!]},$$

with $[\![m]\!] := 1 - z(1-q^m)$

Analysis of the forward cost

- The recursion translates to

$$F^{=m}(z,u) = pq^{m-1}zu^2F^{\leq m-1}(z,u)F^{\leq m}(z,u), \qquad m > 0$$
$$F^{=0}(z,u) = 1$$

- Taking derivatives w.r.t. u and setting $u = 1$, we obtain a recurrence for the GF of expectations:

$$f^{=m}(z) = \frac{2pq^{m-1}z}{[\![m-1]\!][\![m]\!]} + \frac{f^{\leq m-1}(z)}{[\![m]\!]} + \frac{f^{\leq m}(z)}{[\![m-1]\!]},$$

with $[\![m]\!] := 1 - z(1-q^m)$

Analysis of the forward cost

- We solve the recurrence by iteration, with $f^{=m} = f^{\leq m} - f^{\leq m-1}$ and finally take the limit $f(z) := \lim_{m\to\infty} f^{\leq m}(z)$

$$f(z) = \frac{z^2}{(1-z)^2} \sum_{i\geq 1} \frac{pq^{i-1}(1-q^i)}{[\![i]\!]}$$

- Using Euler transform we can easily extract the nth coefficient of $f(z)$, $[z^n]f(z) = \mathbb{E}[F_n]$

$$\mathbb{E}[F_n] = \frac{p}{q} \sum_{k=2}^{n} (-1)^k \frac{1}{Q^{k-1} - 1},$$

$q := 1 - p, Q := 1/q$

Analysis of the forward cost

- We solve the recurrence by iteration, with $f^{=m} = f^{\leq m} - f^{\leq m-1}$ and finally take the limit $f(z) := \lim_{m\to\infty} f^{\leq m}(z)$

$$f(z) = \frac{z^2}{(1-z)^2} \sum_{i\geq 1} \frac{pq^{i-1}(1-q^i)}{[\![i]\!]}$$

- Using Euler transform we can easily extract the nth coefficient of $f(z)$, $[z^n]f(z) = \mathbb{E}[F_n]$

$$\mathbb{E}[F_n] = \frac{p}{q} \sum_{k=2}^{n} (-1)^k \frac{1}{Q^{k-1} - 1},$$

$$q := 1 - p, Q := 1/q$$

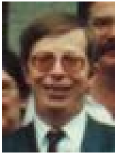
Analysis of the forward cost

The asymptotic behavior of F_n (and other quantities that arise in the analysis of skip lists) can be analyzed using Mellin transforms or Rice's method

$$\sum_{k=a}^{n} \binom{n}{k} (-1)^k f(k) = -\frac{1}{2\pi \mathrm{i}} \int_{\mathcal{C}} \frac{\Gamma(n+1)\Gamma(-z)}{\Gamma(n+1-z)} f(z)\, dz$$

with $\mathcal{C}$ a positively oriented curve enclosing a, $a+1$, ..., n, and $f(z)$ an analytic continuation of $f(k)$

Analysis of the forward cost



P. Kirschenhofer

H. Prodinger

Theorem (Kirschehofer, Prodinger, 1994)

The expected forward cost in a random skip list of size n is

$$\mathbb{E}[F_n] = (Q-1)n\left(\log_Q n + \frac{\gamma - 1}{L} - \frac{1}{2} + \frac{1}{L}\chi(\log_Q n)\right) + O(\log n),$$

with $Q := 1/q$, $L = \ln Q$ and χ a periodic fluctuation of period 1, mean 0 and small amplitude.

To learn more

L. Devroye.
A limit theory for random skip lists.
The Annals of Applied Probability, 2(3):597–609, 1992.

P. Kirschenhofer and H. Prodinger.
The path length of random skip lists.
Acta Informatica, 31(8):775–792, 1994.

P. Kirschenhofer, C. Martínez and H. Prodinger.
Analysis of an Optimized Search Algorithm for Skip Lists.
Theoretical Computer Science, 144:199–220, 1995.

To learn more (2)

T. Papadakis, J. I. Munro, and P. V. Poblete.
Average search and update costs in skip lists.
BIT, 32:316–332, 1992.

H. Prodinger.
Combinatorics of geometrically distributed random variables: Left-to-right maxima.
Discrete Mathematics, 153:253–270, 1996.

W. Pugh.
Skip lists: a probabilistic alternative to balanced trees.
Comm. ACM, 33(6):668–676, 1990.

Randomized binary search trees

C. Aragon

R. Seidel

Two incarnations

- Randomized treaps (tree+heap) invented by Aragon and Seidel (FOCS 1989, Algorithmica 1996) use random priorities and bottom-up balancing
- Randomized binary search trees (RBSTs) invented by Martínez and Roura (ESA 1996, JACM 1998) use subtree sizes and top-down balancing

Randomized binary search trees

C. Aragon

R. Seidel

S. Roura

Two incarnations

- Randomized treaps (tree+heap) invented by Aragon and Seidel (FOCS 1989, Algorithmica 1996) use random priorities and bottom-up balancing
- Randomized binary search trees (RBSTs) invented by Martínez and Roura (ESA 1996, JACM 1998) use subtree sizes and top-down balancing

Randomized binary search trees

- In a random binary search tree (built using random insertions) any of its n elements is the root with probability $1/n$
- Idea: to insert a new item, insert it at the root with probability $1/(n+1)$, otherwise proceed recursively
- The random priorities of treaps "simulate" random timestamps (cif. Vuillemin's Cartesian trees 1980); rotations are used to maintain the BST invariant w.r.t. keys and the heap invariant w.r.t. priorities

Randomized binary search trees

- In a random binary search tree (built using random insertions) any of its n elements is the root with probability $1/n$
- Idea: to insert a new item, insert it at the root with probability $1/(n+1)$, otherwise proceed recursively
- The random priorities of treaps "simulate" random timestamps (cif. Vuillemin's Cartesian trees 1980); rotations are used to maintain the BST invariant w.r.t. keys and the heap invariant w.r.t. priorities

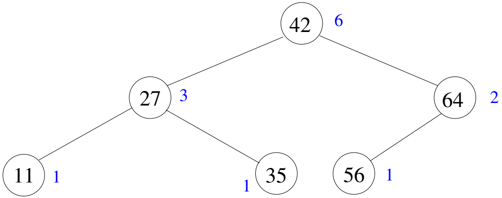
Randomized binary search trees

J. Vuillemin

- In a random binary search tree (built using random insertions) any of its n elements is the root with probability $1/n$
- Idea: to insert a new item, insert it at the root with probability $1/(n+1)$, otherwise proceed recursively
- The random priorities of treaps "simulate" random timestamps (cif. Vuillemin's Cartesian trees 1980); rotations are used to maintain the BST invariant w.r.t. keys and the heap invariant w.r.t. priorities

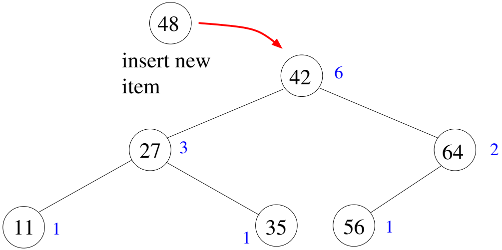
Insertion in a RBST

Inserting an item $x = 48$



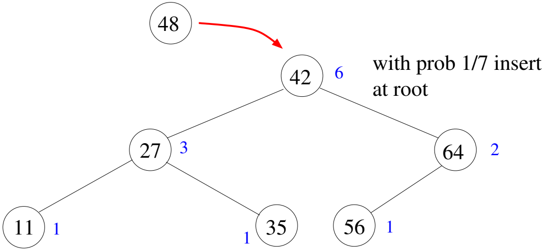
Insertion in a RBST

Inserting an item $x = 48$



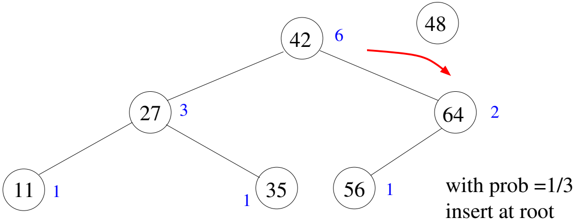
Insertion in a RBST

Inserting an item $x = 48$



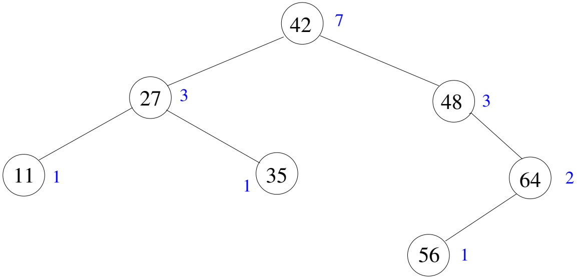
Insertion in a RBST

Inserting an item $x = 48$



Insertion in a RBST

Inserting an item $x = 48$



Insertion in a RBST

procedure Insert(T, x)
    $n \leftarrow T$.size $\triangleright$ $n = 0$ if $T = \square$
    if Uniform$(0, n) = 0$ **then**
        return Insert-at-Root(T, x)
    if $x < T$.item **then**
        T.left $\leftarrow$ Insert$(T$.left$, x)$
    else
        T.right $\leftarrow$ Insert$(T$.right$, x)$
    Update T.size
    return T

Insertion in a RBST

- To insert a new item x at the root of T, we use the algorithm Split that returns two RBSTs T^- and T^+ with element smaller and larger than x, resp.

$$\begin{aligned} \langle T^-, T^+ \rangle &= \text{Split}(T, x) \\ T^- &= \text{BST for } \{y \in T \,|\, y < x\} \\ T^+ &= \text{BST for } \{y \in T \,|\, x < y\} \end{aligned}$$

- Split is like partition in Quicksort
- Insertion at root was invented by Stephenson in 1976

Insertion in a RBST

- To insert a new item x at the root of T, we use the algorithm Split that returns two RBSTs T^- and T^+ with element smaller and larger than x, resp.

$$\begin{aligned} \langle T^-, T^+\rangle &= \text{Split}(T, x) \\ T^- &= \text{BST for } \{y \in T \,|\, y < x\} \\ T^+ &= \text{BST for } \{y \in T \,|\, x < y\} \end{aligned}$$

- Split is like partition in Quicksort
- Insertion at root was invented by Stephenson in 1976

Insertion in a RBST

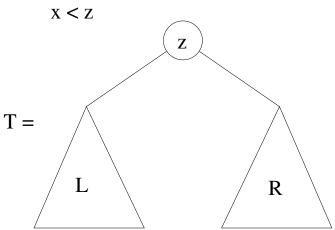
- To insert a new item x at the root of T, we use the algorithm Split that returns two RBSTs T^- and T^+ with element smaller and larger than x, resp.

$$\begin{aligned} \langle T^-, T^+\rangle &= \text{Split}(T, x) \\ T^- &= \text{BST for } \{y \in T \,|\, y < x\} \\ T^+ &= \text{BST for } \{y \in T \,|\, x < y\} \end{aligned}$$

- Split is like partition in Quicksort
- Insertion at root was invented by Stephenson in 1976

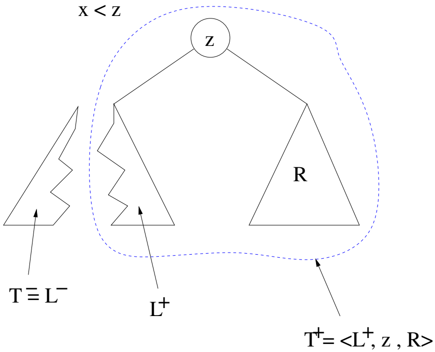
Splitting a RBST

To split a RBST T around x, we need just to follow the path from the root of T to the leaf where x falls



Splitting a RBST

To split a RBST T around x, we need just to follow the path from the root of T to the leaf where x falls



Splitting a RBST

- The cost of the insertion at root (measured # of visited nodes) is exactly the same as the cost of the standard insertion
- For a random(ized) BST this is the depth $L_{n,i}$ of the ith leaf plus 1 (see, e.g., Knuth's volume 3)

$$\begin{aligned}\mathbb{E}[L_{n,i}] &= H_{i-1} + H_{n+1-i} \\ &\sim 2\log n + O(1), \quad i = \alpha \cdot n + o(n), 0 < \alpha < 1\end{aligned}$$

Splitting a RBST

- The cost of the insertion at root (measured # of visited nodes) is exactly the same as the cost of the standard insertion
- For a random(ized) BST this is the depth $L_{n,i}$ of the ith leaf plus 1 (see, e.g., Knuth's volume 3)

$$\begin{aligned}\mathbb{E}[L_{n,i}] &= H_{i-1} + H_{n+1-i} \\ &\sim 2\log n + O(1), \quad i = \alpha \cdot n + o(n), 0 < \alpha < 1\end{aligned}$$

Splitting a RBST

Lemma

Let T^- *and* T^+ *be the BSTs produced by* Split(T, x)*. If* T *is a random BST containing the set of keys* K*, then* T^- *and* T^+ *are independent random BSTs containing the sets of keys* $K^- = \{y \in T \mid y < x\}$ *and* $K^+ = \{y \in T \mid y > x\}$*, respectively.*

Insertion in RBSTs

Theorem

If T is a random BST that contains the set of keys K and x is any key not in K, then $\text{Insert}(T, x)$ *produces a random BST containing the set of keys $K \cup \{x\}$.*

Deletions in RBSTs

```
procedure Delete(T, x)
    if T = □ then
        return T
    if x = T.item then
        return Delete-Root(T)
    if x < T.item then
        T.left ← Delete(T.left, x)
    else
        T.right ← Delete(T.right, x)
    Update T.size
    return T
```

Deletions in RBSTs

- The fundamental problem is how to remove the root node of a BST, in particular, when both subtrees are not empty
- The original deletion algorithm by Hibbard was assumed to preserve randomness
- In 1975, G. Knott discovered that Hibbard's deletion preserves randomness of shape, but an insertion following a deletion would destroy randomness (Knott's paradox)

Deletions in RBSTs

- The fundamental problem is how to remove the root node of a BST, in particular, when both subtrees are not empty
- The original deletion algorithm by Hibbard was assumed to preserve randomness
- In 1975, G. Knott discovered that Hibbard's deletion preserves randomness of shape, but an insertion following a deletion would destroy randomness (Knott's paradox)

Deletions in RBSTs

- The fundamental problem is how to remove the root node of a BST, in particular, when both subtrees are not empty
- The original deletion algorithm by Hibbard was assumed to preserve randomness
- In 1975, G. Knott discovered that Hibbard's deletion preserves randomness of shape, but an insertion following a deletion would destroy randomness (Knott's paradox)

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

J. Culberson

J.L. Eppinger

D.E. Knuth

- Several theoretical and experimental work aimed at understanding what was the effect of deletions, e.g.,
  - Jonassen & Knuth's *An Algorithm whose Analysis Isn't* (JCSS, 1978)
  - Knuth's *Deletions that Preserve Randomness* (IEEE Trans. Soft. Eng., 1977)
  - Eppinger's experiments (CACM, 1983)
  - Culberson's paper on deletions of the left spine (STOC, 1985)
  - ...
- These studies showed that deletions degraded performance in the long run

Deletions in RBSTs

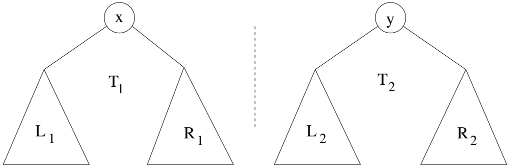
We delete the root using a procedure Join(T_1, T_2). Given two BSTs such that for all $x \in T_1$ and all $y \in T_2$, $x \le y$, it returns a new BST that contains all the keys in T_1 and T_2. Then

$$\text{Delete-Root}(T) \equiv \text{Join}(T.\text{left}, T.\text{right})$$

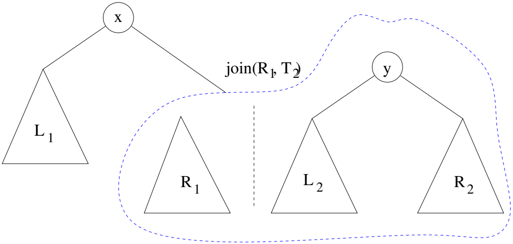
with

$$\begin{aligned} \text{Join}(\square, \square) &= \square \\ \text{Join}(T, \square) &= \text{Join}(\square, T) = T \\ \text{Join}(T_1, T_2) &= ?, \qquad T_1 \neq \square, T_2 \neq \square \end{aligned}$$

Joining two BSTs



Joining two BSTs



Joining two BSTs

- If we systematically choose the root of T_1 as the root of Join(T_1, T_2), or the other way around, we will introduce an undesirable bias
- Suppose both T_1 and T_2 are random. Let m and n denote their sizes. Then x is the root of T_1 with probability $1/m$ and y is the root of T_2 with probability $1/n$
- Choose x as the common root with probability $m/(m+n)$, choose y with probability $n/(m+n)$

$$\frac{1}{m} \times \frac{m}{m+n} = \frac{1}{m+n}$$
$$\frac{1}{n} \times \frac{n}{m+n} = \frac{1}{m+n}$$

Joining two BSTs

- If we systematically choose the root of T_1 as the root of Join(T_1, T_2), or the other way around, we will introduce an undesirable bias
- Suppose both T_1 and T_2 are random. Let m and n denote their sizes. Then x is the root of T_1 with probability $1/m$ and y is the root of T_2 with probability $1/n$
- Choose x as the common root with probability $m/(m+n)$, choose y with probability $n/(m+n)$

$$\frac{1}{m} \times \frac{m}{m+n} = \frac{1}{m+n}$$

$$\frac{1}{n} \times \frac{n}{m+n} = \frac{1}{m+n}$$

Joining two BSTs

- If we systematically choose the root of T_1 as the root of Join(T_1, T_2), or the other way around, we will introduce an undesirable bias
- Suppose both T_1 and T_2 are random. Let m and n denote their sizes. Then x is the root of T_1 with probability $1/m$ and y is the root of T_2 with probability $1/n$
- Choose x as the common root with probability $m/(m+n)$, choose y with probability $n/(m+n)$

$$\frac{1}{m} \times \frac{m}{m+n} = \frac{1}{m+n}$$
$$\frac{1}{n} \times \frac{n}{m+n} = \frac{1}{m+n}$$

Joining two RBSTs

Lemma

Let L and R be two independent random BSTs, such that the keys in L are strictly smaller than the keys in R. Let K_L and K_R denote the sets of keys in L and R, respectively. Then $T = \mathrm{Join}(L, R)$ is a random BST that contains the set of keys $K = K_L \cup K_R$.

Joining two RBSTs

- The recursion for JoinT_1, T_2 traverses the rightmost branch (right spine) of T_1 and the leftmost branch (left spine) of T_2
- The trees to be joined are the left and right subtrees L and R of the ith item in a RBST of size n; then

$$\text{length of left spine of } L = \text{path length to } i\text{th leaf}$$
$$\text{length of right spine of } R = \text{path length to } (i+1)\text{th leaf}$$

- The cost of the joining phase is the sum of the path lengths to the leaves minus twice the depth of the ith item; the expected cost follows from well-known results

$$\left(2 - \frac{1}{i} - \frac{1}{n+1-i}\right) = O(1)$$

Joining two RBSTs

- The recursion for JoinT_1, T_2 traverses the rightmost branch (right spine) of T_1 and the leftmost branch (left spine) of T_2
- The trees to be joined are the left and right subtrees L and R of the ith item in a RBST of size n; then

$$\text{length of left spine of } L = \text{path length to } i\text{th leaf}$$
$$\text{length of right spine of } R = \text{path length to } (i+1)\text{th leaf}$$

- The cost of the joining phase is the sum of the path lengths to the leaves minus twice the depth of the ith item; the expected cost follows from well-known results

$$\left(2 - \frac{1}{i} - \frac{1}{n+1-i}\right) = O(1)$$

Joining two RBSTs

- The recursion for JoinT_1, T_2 traverses the rightmost branch (right spine) of T_1 and the leftmost branch (left spine) of T_2
- The trees to be joined are the left and right subtrees L and R of the ith item in a RBST of size n; then

$$\text{length of left spine of } L = \text{path length to } i\text{th leaf}$$
$$\text{length of right spine of } R = \text{path length to } (i+1)\text{th leaf}$$

- The cost of the joining phase is the sum of the path lengths to the leaves minus twice the depth of the ith item; the expected cost follows from well-known results

$$\left(2 - \frac{1}{i} - \frac{1}{n+1-i}\right) = O(1)$$

Deletions in RBSTs

Theorem

If T is a random BST that contains the set of keys K, then Delete(T, x) *produces a random BST containing the set of keys* $K \setminus \{x\}$.

Deletions in RBSTs

Theorem

If T is a random BST that contains the set of keys K, then Delete(T, x) *produces a random BST containing the set of keys* $K \setminus \{x\}$.

Corollary

The result of any arbitary sequence of insertions and deletions, starting from an initially empty tree is always a random BST.

Additional remarks

- Arbitrary insertions and deletions yield always random BSTs
- A deletion algorithm for BSTs that preserved randomness was a long standing open problem (10-15 yr)
- Properties of random BSTs have been investigated in depth and for a long time
- Treaps only need to generate a single random number per node (with $O(\log n)$ bits)
- RBSTs need $O(\log n)$ calls to the random generator per insertion, and $O(1)$ calls per deletion (on average)

Additional remarks

- Arbitrary insertions and deletions yield always random BSTs
- A deletion algorithm for BSTs that preserved randomness was a long standing open problem (10-15 yr)
- Properties of random BSTs have been investigated in depth and for a long time
- Treaps only need to generate a single random number per node (with $O(\log n)$ bits)
- RBSTs need $O(\log n)$ calls to the random generator per insertion, and $O(1)$ calls per deletion (on average)

Additional remarks

- Arbitrary insertions and deletions yield always random BSTs
- A deletion algorithm for BSTs that preserved randomness was a long standing open problem (10-15 yr)
- Properties of random BSTs have been investigated in depth and for a long time
- Treaps only need to generate a single random number per node (with $O(\log n)$ bits)
- RBSTs need $O(\log n)$ calls to the random generator per insertion, and $O(1)$ calls per deletion (on average)

Additional remarks

- Arbitrary insertions and deletions yield always random BSTs
- A deletion algorithm for BSTs that preserved randomness was a long standing open problem (10-15 yr)
- Properties of random BSTs have been investigated in depth and for a long time
- Treaps only need to generate a single random number per node (with $O(\log n)$ bits)
- RBSTs need $O(\log n)$ calls to the random generator per insertion, and $O(1)$ calls per deletion (on average)

Additional remarks

- Arbitrary insertions and deletions yield always random BSTs
- A deletion algorithm for BSTs that preserved randomness was a long standing open problem (10-15 yr)
- Properties of random BSTs have been investigated in depth and for a long time
- Treaps only need to generate a single random number per node (with $O(\log n)$ bits)
- RBSTs need $O(\log n)$ calls to the random generator per insertion, and $O(1)$ calls per deletion (on average)

Additional remarks

- Storing subtree sizes for balancing is more useful: they can be used to implement search and deletion by rank, e.g., find the ith smallest element in the tree
- Other operations, e.g., union and intersection are also efficiently supported by RBSTs
- Similar ideas have been used to randomize other search trees, namely, K-dimensional binary search trees (Duch and Martínez, 1998) and quadtrees (Duch, 1999)

Additional remarks

- Storing subtree sizes for balancing is more useful: they can be used to implement search and deletion by rank, e.g., find the ith smallest element in the tree
- Other operations, e.g., union and intersection are also efficiently supported by RBSTs
- Similar ideas have been used to randomize other search trees, namely, K-dimensional binary search trees (Duch and Martínez, 1998) and quadtrees (Duch, 1999)

Additional remarks

- Storing subtree sizes for balancing is more useful: they can be used to implement search and deletion by rank, e.g., find the ith smallest element in the tree
- Other operations, e.g., union and intersection are also efficiently supported by RBSTs
- Similar ideas have been used to randomize other search trees, namely, K-dimensional binary search trees (Duch and Martínez, 1998) and quadtrees (Duch, 1999)

To learn more

H. M. Mahmoud.
Evolution of Random Search Trees.
Wiley Interscience, 1992.

C. Martínez and S. Roura.
Randomized binary search trees.
J. Assoc. Comput. Mach., 45(2):288–323, 1998.

R. Seidel and C. Aragon.
Randomized search trees.
Algorithmica, 16:464–497, 1996.

General References

- Ph. Flajolet and R. Sedgewick.
  Analytic Combinatorics.
  Cambridge University Press, 2008.
- D. E. Knuth.
  The Art of Computer Programming: Sorting and Searching, volume 3.
  Addison-Wesley, 2nd edition, 1998.
- C. Pandu Rangan.
  Randomized Data Structures, in *Handbook of Data Structures and Applications*.
  D.P. Mehta and S. Sahni, editors.
  Chapman & Hall, CRC, 2005.

General References (2)

P. Raghavan and R. Motwani.
Randomized Algorithms.
Cambridge University Press, 1995.

R. Sedgewick.
Algorithms in C.
Addison-Wesley, 3rd edition, 1997.

MERCI BEAUCOUP!

Avec mes meilleurs voeux de succès et longue vie
pour CALIN