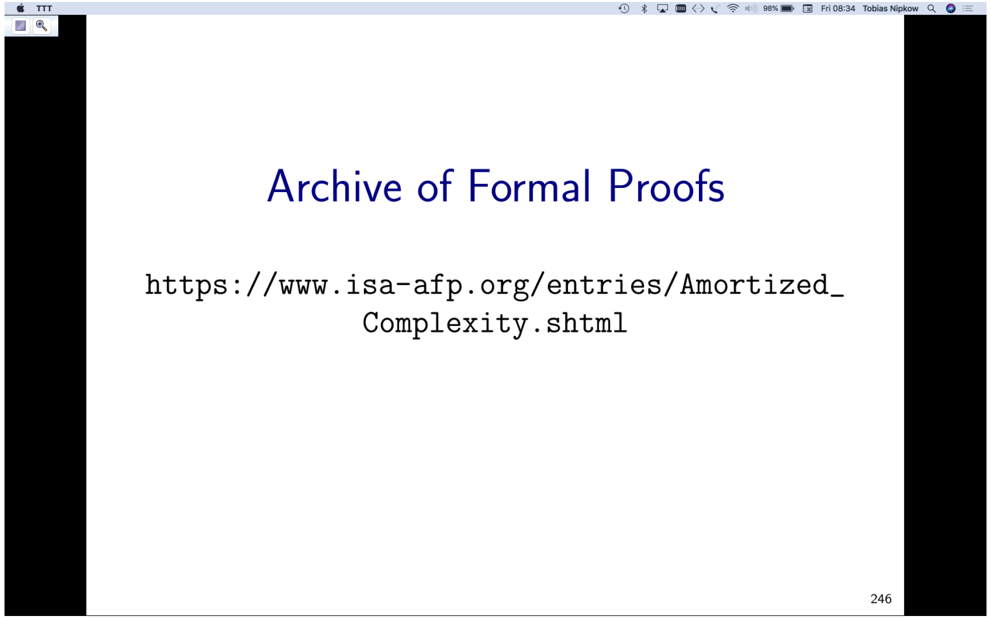
Script generated by TTT

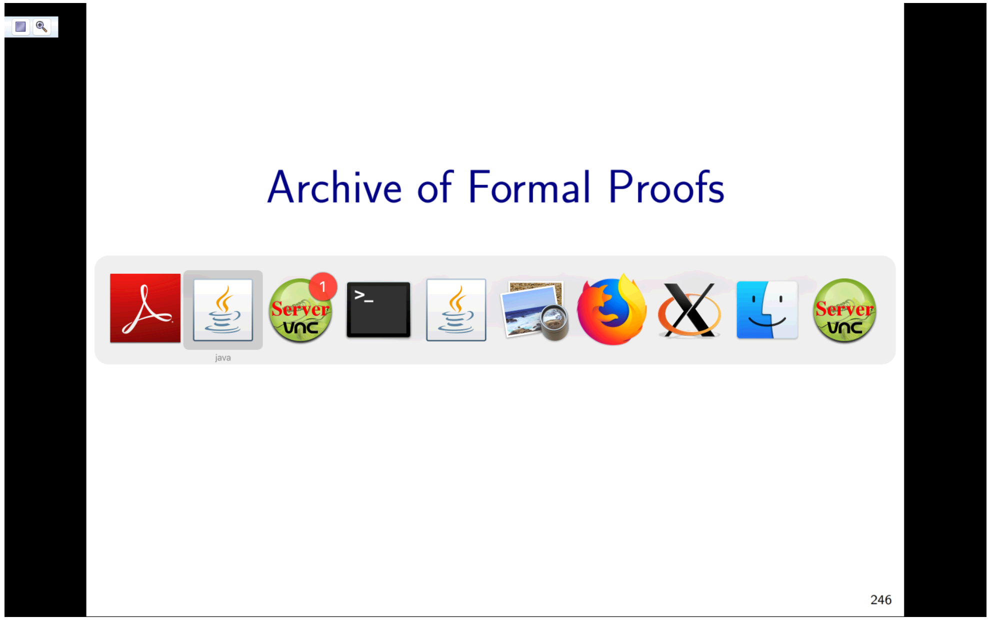
Title: FDS (19.07.2019)

Date: Fri Jul 19 08:34:10 CEST 2019

Duration: 80:46 min

Pages: 75





A *skew heap* is a self-adjusting heap (priority queue)

A *skew heap* is a self-adjusting heap (priority queue)

Functions $insert$, $merge$ and del_min
have amortized logarithmic complexity.

A *skew heap* is a self-adjusting heap (priority queue)

Functions $insert$, $merge$ and del_min
have amortized logarithmic complexity.

Functions $insert$ and del_min are defined via $merge$

Implementation type

Ordinary binary trees

Invariant: $heap$

merge

$merge\ \langle\rangle\ h = h$
$merge\ h\ \langle\rangle = h$

merge

$merge\ \langle\rangle\ h = h$
$merge\ h\ \langle\rangle = h$

Swap subtrees when descending:

merge

$merge\ \langle\rangle\ h = h$
$merge\ h\ \langle\rangle = h$

Swap subtrees when descending:

$merge\ (\langle l_1,\ a_1,\ r_1\rangle =: h_1)\ (\langle l_2,\ a_2,\ r_2\rangle =: h_2) =$
$(\text{if } a_1 \le a_2 \text{ then } \langle merge\ h_2\ r_1,\ a_1,\ l_1\rangle$
$\ \text{else } \langle merge\ h_1\ r_2,\ a_2,\ l_2\rangle)$

merge

Very similar to leftist heap but

Logarithmic amortized complexity

Theorem

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq 3 * \log_2\ (|t_1|_1 + |t_2|_1) + 1$

Towards the proof

Right heavy:
$rh\ l\ r =$ (if $|l| < |r|$ then 1 else 0)

Towards the proof

Right heavy:
$rh\ l\ r =$ (if $|l| < |r|$ then 1 else 0)

Number of right heavy nodes on left spine:

$lrh\ \langle\rangle = 0$
$lrh\ \langle l,\ \_,\ r\rangle = rh\ l\ r + lrh\ l$

Towards the proof

Right heavy:
$rh\ l\ r =$ (if $|l| < |r|$ then 1 else 0)

Number of right heavy nodes on left spine:

$lrh\ \langle\rangle = 0$
$lrh\ \langle l,\ \_,\ r\rangle = rh\ l\ r + lrh\ l$

Lemma

$2^{lrh\ h} \leq |h| + 1$

Towards the proof

Right heavy:
$rh\ l\ r = (\text{if } |l| < |r| \text{ then } 1 \text{ else } 0)$

Number of right heavy nodes on left spine:
$lrh\ \langle\rangle = 0$
$lrh\ \langle l, \_, r\rangle = rh\ l\ r + lrh\ l$

Lemma
$2^{lrh\ h} \leq |h| + 1$

Corollary
$lrh\ h \leq \log_2\ |h|_1$

Towards the proof

Right heavy:
$rh\ l\ r = (\text{if } |l| < |r| \text{ then } 1 \text{ else } 0)$

Number of not right heavy nodes on right spine:
$rlh\ \langle\rangle = 0$
$rlh\ \langle l, \_, r\rangle = 1 - rh\ l\ r + rlh\ r$

Lemma
$2^{rlh\ h} \leq |h| + 1$

Corollary
$rlh\ h \leq \log_2\ |h|_1$

Potential

The potential is the number of right heavy nodes:
$\Phi\ \langle\rangle = 0$
$\Phi\ \langle l, \_, r\rangle = \Phi\ l + \Phi\ r + rh\ l\ r$
merge descends on the right
$\Longrightarrow$ right heavy nodes are bad

Potential

The potential is the number of right heavy nodes:
$\Phi\ \langle\rangle = 0$
$\Phi\ \langle l, \_, r\rangle = \Phi\ l + \Phi\ r + rh\ l\ r$
merge descends on the right
$\Longrightarrow$ right heavy nodes are bad

Lemma
$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq lrh\ (merge\ t_1\ t_2) + rlh\ t_1 + rlh\ t_2 + 1$

Potential

The potential is the number of right heavy nodes:

$\Phi\ \langle\rangle = 0$

$\Phi\ \langle l,\ \_,\ r\rangle = \Phi\ l + \Phi\ r + rh\ l\ r$

merge descends on the right

$\Longrightarrow$ right heavy nodes are bad

Lemma

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\le lrh\ (merge\ t_1\ t_2) + rlh\ t_1 + rlh\ t_2 + 1$

```
by(induction t1 t2 rule: merge.induct)(auto)
```


Node-Node case

Let $t_1 = \langle l_1,\ a_1,\ r_1\rangle$, $t_2 = \langle l_2,\ a_2,\ r_2\rangle$.

Node-Node case

Let $t_1 = \langle l_1,\ a_1,\ r_1\rangle$, $t_2 = \langle l_2,\ a_2,\ r_2\rangle$.
Case $a_1 \le a_2$. Let $m = merge\ t_2\ r_1$

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$= t\_merge\ t_2\ r_1 + 1 + \Phi\ m + \Phi\ l_1 + rh\ m\ l_1$
$\quad - \Phi\ t_1 - \Phi\ t_2$
$= t\_merge\ t_2\ r_1 + 1 + \Phi\ m + rh\ m\ l_1$
$\quad - \Phi\ r_1 - rh\ l_1\ r_1 - \Phi\ t_2$
$\le lrh\ m + rlh\ t_2 + rlh\ r_1 + rh\ m\ l_1 + 2 - rh\ l_1\ r_1$
by IH
$= lrh\ m + rlh\ t_2 + rlh\ t_1 + rh\ m\ l_1 + 1$

Node-Node case

Let $t_1 = \langle l_1,\ a_1,\ r_1\rangle$, $t_2 = \langle l_2,\ a_2,\ r_2\rangle$.

Main proof

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq lrh\ (merge\ t_1\ t_2) + rlh\ t_1 + rlh\ t_2 + 1$
$\leq \log_2 |merge\ t_1\ t_2|_1 + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$= \log_2 (|t_1|_1 + |t_2|_1 - 1) + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$

Main proof

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq lrh\ (merge\ t_1\ t_2) + rlh\ t_1 + rlh\ t_2 + 1$
$\leq \log_2 |merge\ t_1\ t_2|_1 + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$= \log_2 (|t_1|_1 + |t_2|_1 - 1) + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$\leq \log_2 (|t_1|_1 + |t_2|_1) + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$\leq \log_2 (|t_1|_1 + |t_2|_1) + 2 * \log_2 (|t_1|_1 + |t_2|_1) + 1$
because $\log_2 x + \log_2 y \leq 2 * \log_2 (x + y)$ if $x,y > 0$

Main proof

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq lrh\ (merge\ t_1\ t_2) + rlh\ t_1 + rlh\ t_2 + 1$
$\leq \log_2 |merge\ t_1\ t_2|_1 + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$= \log_2 (|t_1|_1 + |t_2|_1 - 1) + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$\leq \log_2 (|t_1|_1 + |t_2|_1) + \log_2 |t_1|_1 + \log_2 |t_2|_1 + 1$
$\leq \log_2 (|t_1|_1 + |t_2|_1) + 2 * \log_2 (|t_1|_1 + |t_2|_1) + 1$
because $\log_2 x + \log_2 y \leq 2 * \log_2 (x + y)$ if $x,y > 0$
$= 3 * \log_2 (|t_1|_1 + |t_2|_1) + 1$

insert and *del_min*

Easy consequences:

Lemma
$t\_insert\ a\ h + \Phi\ (insert\ a\ h) - \Phi\ h$
$\leq 3 * \log_2 (|h|_1 + 2) + 2$

Lemma
$t\_del\_min\ h + \Phi\ (del\_min\ h) - \Phi\ h$
$\leq 3 * \log_2 (|h|_1 + 2) + 2$

Sources

The inventors of skew heaps:

Daniel Sleator and Robert Tarjan.
Self-adjusting Heaps.
SIAM J. Computing, 1986.

Sources

The inventors of skew heaps:

Daniel Sleator and Robert Tarjan.
Self-adjusting Heaps.
SIAM J. Computing, 1986.

Sources

The inventors of skew heaps:

Daniel Sleator and Robert Tarjan.
Self-adjusting Heaps.
SIAM J. Computing, 1986.

The formalization is based on

Anne Kaldewaij and Berry Schoenmakers.
The Derivation of a Tighter Bound for Top-down Skew Heaps. *Information Processing Letters*, 1991.

A *splay tree* is a self-adjusting binary search tree.

Definition (splay)

Become wider or more separated.

Definition (splay)

Become wider or more separated.

Example

The river splayed out into a delta.

Splay tree

Implementation type = binary tree

Key operation *splay* a:

1. Search for a ending up at x where $x = a$ or x is a leaf node.

Splay tree

Implementation type = binary tree

Key operation *splay* a:

1. Search for a ending up at x
   where $x = a$ or x is a leaf node.
2. Move x to the root of the tree by rotations.

Splay tree

Implementation type = binary tree

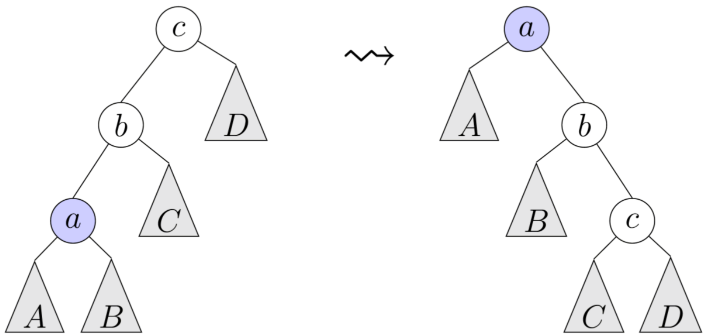
Key operation *splay* a:

1. Search for a ending up at x
   where $x = a$ or x is a leaf node.
2. Move x to the root of the tree by rotations.

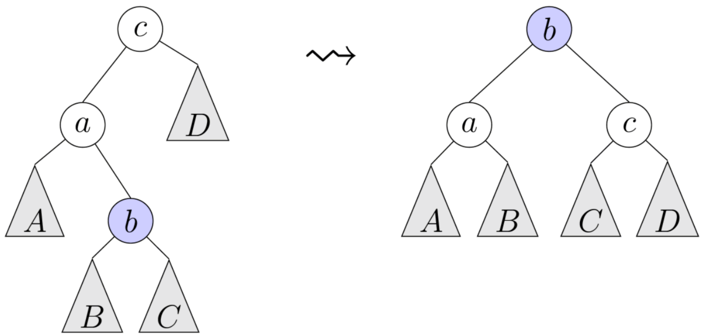
Derived operations $isin/insert/delete\ a$:

1. $splay\ a$
2. Perform $isin/insert/delete$ action

Zig-zig



Zig-zag

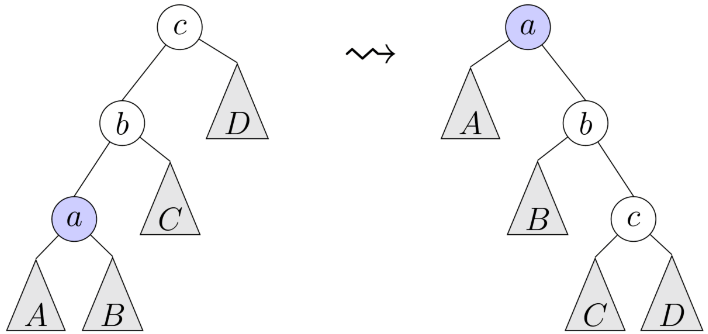


Zig-zig and zig-zag

Zig-zig $\neq$ two single rotations

Zig-zag $=$ two single rotations

Zig-zig



Functional definition

$splay :: 'a \Rightarrow 'a\ tree \Rightarrow 'a\ tree$

Zig-zig and zig-zag

$[\![x < b;\ x < c;\ AB \neq \langle\rangle]\!]$
$\Longrightarrow splay\ x\ \langle\langle AB,\ b,\ C\rangle,\ c,\ D\rangle =$
(case $splay\ x\ AB$ of
$\langle A,\ a,\ B\rangle \Rightarrow \langle A,\ a,\ \langle B,\ b,\ \langle C,\ c,\ D\rangle\rangle\rangle)$

Zig-zig and zig-zag

$[\![x < b;\ x < c;\ AB \neq \langle\rangle]\!]$
$\Longrightarrow$ *splay* $x\ \langle\langle AB,\ b,\ C\rangle,\ c,\ D\rangle =$
(case *splay* $x\ AB$ of
$\langle A,\ a,\ B\rangle \Rightarrow \langle A,\ a,\ \langle B,\ b,\ \langle C,\ c,\ D\rangle\rangle\rangle)$

$[\![x < c;\ c < a;\ BC \neq \langle\rangle]\!]$
$\Longrightarrow$ *splay* $c\ \langle\langle A,\ x,\ BC\rangle,\ a,\ D\rangle =$
(case *splay* $c\ BC$ of
$\langle B,\ b,\ C\rangle \Rightarrow \langle\langle A,\ x,\ B\rangle,\ b,\ \langle C,\ a,\ D\rangle\rangle)$

Some base cases

$x < b \Longrightarrow$ *splay* $x\ \langle\langle A,\ x,\ B\rangle,\ b,\ C\rangle =$

Some base cases

$x < b \Longrightarrow$ *splay* $x\ \langle\langle A,\ x,\ B\rangle,\ b,\ C\rangle = \langle A,\ x,\ \langle B,\ b,\ C\rangle\rangle$

Some base cases

$x < b \Longrightarrow$ *splay* $x\ \langle\langle A,\ x,\ B\rangle,\ b,\ C\rangle = \langle A,\ x,\ \langle B,\ b,\ C\rangle\rangle$

$x < a \Longrightarrow$
splay $x\ \langle\langle\langle\rangle,\ a,\ A\rangle,\ b,\ B\rangle =$

Functional correctness proofs

Automatic

Potential

Sum of logarithms of the size of all nodes:

Potential

Sum of logarithms of the size of all nodes:

$\Phi\ \langle\rangle = 0$

$\Phi\ \langle l,\ a,\ r\rangle = \Phi\ l + \Phi\ r + \varphi\ \langle l,\ a,\ r\rangle$

where $\varphi\ t = \log_2\ (|t| + 1)$

Potential

Sum of logarithms of the size of all nodes:

$\Phi\ \langle\rangle = 0$

$\Phi\ \langle l,\ a,\ r\rangle = \Phi\ l + \Phi\ r + \varphi\ \langle l,\ a,\ r\rangle$

where $\varphi\ t = \log_2\ (|t| + 1)$

Amortized complexity of *splay*:

$$a\_splay\ a\ t = t\_splay\ a\ t + \Phi\ (splay\ a\ t) - \Phi\ t$$

Analysis of *splay*

Theorem

$[\![bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t]\!]$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Analysis of *splay*

Theorem

$[\![bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t]\!]$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Analysis of *splay*

Theorem

$[\![bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t]\!]$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Corollary

$[\![bst\ t;\ a \in set\_tree\ t]\!]$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$

Analysis of *splay*

Theorem

$[\![bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t]\!]$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Corollary

$[\![bst\ t;\ a \in set\_tree\ t]\!]$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$

Corollary

$bst\ t \Longrightarrow a\_splay\ a\ t \leq 3 * \varphi\ t + 1$

Analysis of *splay*

Theorem

$\llbracket bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Corollary

$\llbracket bst\ t;\ a \in set\_tree\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$

Corollary

$bst\ t \Longrightarrow a\_splay\ a\ t \leq 3 * \varphi\ t + 1$

Analysis of *splay*

Theorem

$\llbracket bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Corollary

$\llbracket bst\ t;\ a \in set\_tree\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$

Corollary

$bst\ t \Longrightarrow a\_splay\ a\ t \leq 3 * \varphi\ t + 1$

Lemma

$\llbracket t \neq \langle\rangle;\ bst\ t\rrbracket$
$\Longrightarrow \exists\, a' \in set\_tree\ t.$
$splay\ a'\ t = splay\ a\ t \wedge t\_splay\ a'\ t = t\_splay\ a\ t$

Analysis of *splay*

Theorem

$\llbracket bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Corollary

$\llbracket bst\ t;\ a \in set\_tree\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$

Corollary

$bst\ t \Longrightarrow a\_splay\ a\ t \leq 3 * \varphi\ t + 1$

279

Analysis of *splay*

Theorem

$\llbracket bst\ t;\ \langle l,\ a,\ r\rangle \in subtrees\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - \varphi\ \langle l,\ a,\ r\rangle) + 1$

Corollary

$\llbracket bst\ t;\ a \in set\_tree\ t\rrbracket$
$\Longrightarrow a\_splay\ a\ t \leq 3 * (\varphi\ t - 1) + 1$

Corollary

$bst\ t \Longrightarrow a\_splay\ a\ t \leq 3 * \varphi\ t + 1$

Lemma

$\llbracket t \neq \langle\rangle;\ bst\ t\rrbracket$
$\Longrightarrow \exists\, a' \in set\_tree\ t.$
$splay\ a'\ t = splay\ a\ t \wedge t\_splay\ a'\ t = t\_splay\ a\ t$

279

insert

Definition

$insert\ x\ t =$
(if $t = \langle\rangle$ then $\langle\langle\rangle,\ x,\ \langle\rangle\rangle$
else case $splay\ x\ t$ of
$\quad\langle l,\ a,\ r\rangle \Rightarrow$ case $cmp\ x\ a$ of
$\qquad LT \Rightarrow \langle l,\ x,\ \langle\langle\rangle,\ a,\ r\rangle\rangle$
$\qquad |\ EQ \Rightarrow \langle l,\ a,\ r\rangle$
$\qquad |\ GT \Rightarrow \langle\langle l,\ a,\ \langle\rangle\rangle,\ x,\ r\rangle)$

insert

Definition

$insert\ x\ t =$
(if $t = \langle\rangle$ then $\langle\langle\rangle,\ x,\ \langle\rangle\rangle$
else case $splay\ x\ t$ of
$\quad\langle l,\ a,\ r\rangle \Rightarrow$ case $cmp\ x\ a$ of
$\qquad LT \Rightarrow \langle l,\ x,\ \langle\langle\rangle,\ a,\ r\rangle\rangle$
$\qquad |\ EQ \Rightarrow \langle l,\ a,\ r\rangle$
$\qquad |\ GT \Rightarrow \langle\langle l,\ a,\ \langle\rangle\rangle,\ x,\ r\rangle)$

Counting only the cost of *splay*:

Lemma

$bst\ t \Longrightarrow$
$t\_splay\ a\ t + \Phi\ (insert\ a\ t) - \Phi\ t \leq 4 * \varphi\ t + 2$

delete

Definition

$delete\ x\ t =$
(if $t = \langle\rangle$ then $\langle\rangle$
else case $splay\ x\ t$ of
$\quad\langle l,\ a,\ r\rangle \Rightarrow$
$\qquad$ if $x = a$
$\qquad$ then if $l = \langle\rangle$ then r
$\qquad\quad$ else case $splay\_max\ l$ of
$\qquad\qquad \langle l',\ m,\ r'\rangle \Rightarrow \langle l',\ m,\ r\rangle$
$\qquad$ else $\langle l,\ a,\ r\rangle)$

delete

Definition

$delete\ x\ t =$
(if $t = \langle\rangle$ then $\langle\rangle$
else case $splay\ x\ t$ of
$\quad\langle l,\ a,\ r\rangle \Rightarrow$
$\qquad$ if $x = a$
$\qquad$ then if $l = \langle\rangle$ then r
$\qquad\quad$ else case $splay\_max\ l$ of
$\qquad\qquad \langle l',\ m,\ r'\rangle \Rightarrow \langle l',\ m,\ r\rangle$
$\qquad$ else $\langle l,\ a,\ r\rangle)$

Lemma

$bst\ t \Longrightarrow$
$t\_delete\ a\ t + \Phi\ (delete\ a\ t) - \Phi\ t \leq 6 * \varphi\ t + 2$

Remember

Amortized analysis is only correct for single threaded uses of a data structure.

Otherwise:

let $counter = 0;$
$bad =$ increment $counter$ $2^n - 1$ times;
$\_ = incr\ bad;$
$\_ = incr\ bad;$
$\_ = incr\ bad;$
$\vdots$

$isin :: 'a\ tree \Rightarrow 'a \Rightarrow bool$

$isin :: 'a\ tree \Rightarrow 'a \Rightarrow bool$

Single threaded $\Longrightarrow$ $isin\ t\ a$ eats up t

$isin :: 'a\ tree \Rightarrow 'a \Rightarrow bool$

Single threaded $\Longrightarrow$ $isin\ t\ a$ eats up t

Otherwise:

let $bad =$ build unbalanced splay tree;
$\_ = isin\ bad\ a;$
$\_ = isin\ bad\ a;$
$\_ = isin\ bad\ a;$
$\vdots$

Solution 1:

$$isin :: 'a\ tree \Rightarrow 'a \Rightarrow bool \times 'a\ tree$$

Observer function returns new data structure:

Solution 1:

$$isin :: 'a\ tree \Rightarrow 'a \Rightarrow bool \times 'a\ tree$$

Observer function returns new data structure:

Definition

$isin\ t\ a =$
$(\text{let } t' = splay\ a\ t \text{ in } (\text{case } t' \text{ of}$
$\langle\rangle \Rightarrow False$
$\mid \langle l,\ x,\ r\rangle \Rightarrow a = x,$
$t'))$

Solution 2:

$$isin = splay;\ is\_root$$

Client uses $splay$ before calling is_root:

Definition

$is\_root :: 'a \Rightarrow 'a\ tree \Rightarrow bool$
$is\_root\ x\ t = (\text{case } t \text{ of}$
$\langle\rangle \Rightarrow False$
$\mid \langle l,\ a,\ r\rangle \Rightarrow x = a)$

Solution 2:

$$isin = splay;\ is\_root$$

Client uses $splay$ before calling is_root:

Definition

$is\_root :: 'a \Rightarrow 'a\ tree \Rightarrow bool$
$is\_root\ x\ t = (\text{case } t \text{ of}$
$\langle\rangle \Rightarrow False$
$\mid \langle l,\ a,\ r\rangle \Rightarrow x = a)$

May call $is\_root\ \_\ t$ multiple times (with the same t!) because is_root takes constant time

isin

Splay trees have an imperative flavour and are a bit awkward to use in a purely functional language

Sources

The inventors of splay trees:

Daniel Sleator and Robert Tarjan.
Self-adjusting Binary Search Trees. *J. ACM*, 1985.

Sources

The inventors of splay trees:

Daniel Sleator and Robert Tarjan.
Self-adjusting Binary Search Trees. *J. ACM*, 1985.

The formalization is based on

Berry Schoenmakers. A Systematic Analysis of Splaying.
Information Processing Letters, 1993.