Script generated by TTT

Title: FDS (21.07.2017)

Date: Fri Jul 21 08:32:54 CEST 2017

Duration: 88:10 min

Pages: 104

16 Amortized Complexity

17 Skew Heap

18 Splay Tree

19 Pairing Heap

16 Amortized Complexity

17 Skew Heap

18 Splay Tree

19 Pairing Heap

191

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:

$merge\ (h_1 = \langle l_1,\ a,\ r_1\rangle)\ (h_2 = \langle l_2,\ b,\ r_2\rangle) =$

(if $a \leq b$ then $\langle merge\ h_2\ r_1,\ a,\ l_1\rangle$

else $\langle merge\ h_1\ r_2,\ b,\ l_2\rangle$)

merge

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

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

Swap subtrees when descending:

$merge\ (h_1 = \langle l_1,\ a,\ r_1\rangle)\ (h_2 = \langle l_2,\ b,\ r_2\rangle) =$

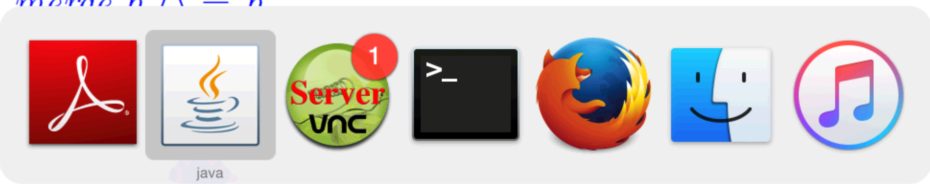
(if $a \leq b$ then $\langle merge\ h_2\ r_1,\ a,\ l_1\rangle$

else $\langle merge\ h_1\ r_2,\ b,\ l_2\rangle$)

Function *merge* terminates because . . .

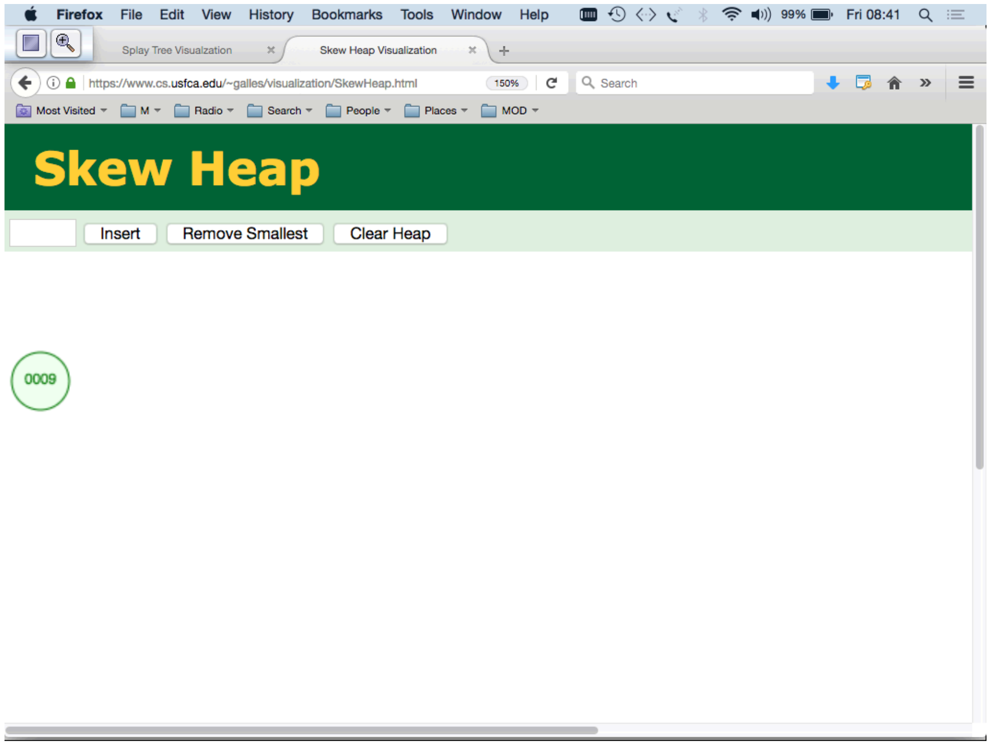
merge

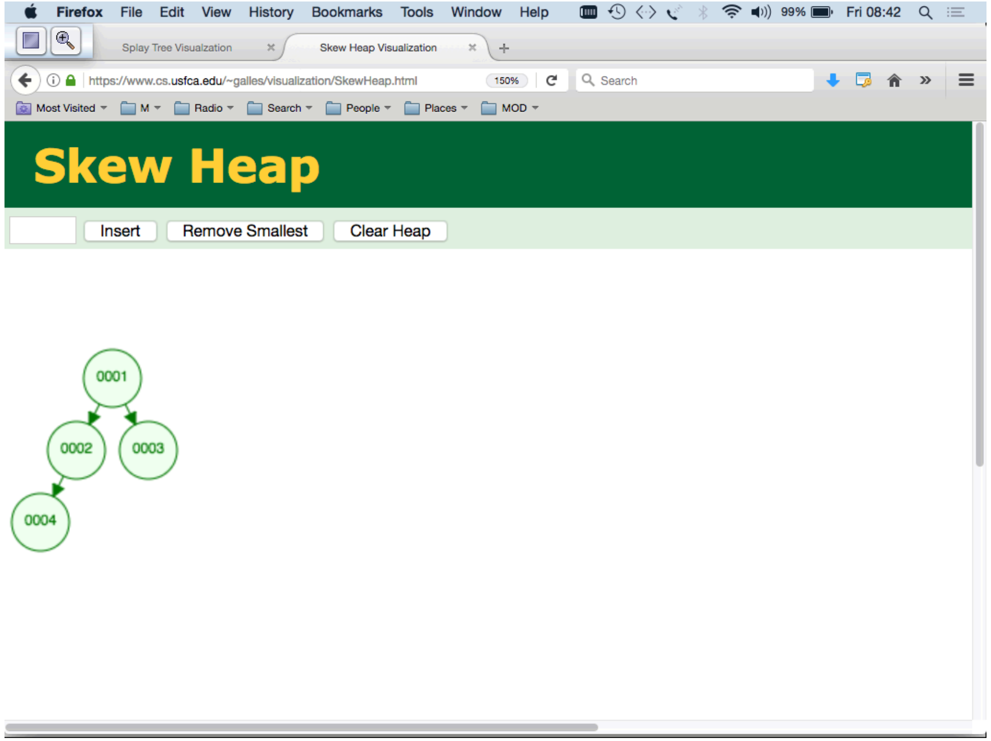
$merge\ \langle\rangle\ h = h$



else $\langle merge\ h_1\ r_2,\ b,\ l_2\rangle$)

Function *merge* terminates because . . .





merge

Very similar to leftist heap but

Towards the proof

$$rheavy\ \langle l,\ \_,\ r\rangle = (|l| < |r|)$$

Towards the proof

$$rheavy\ \langle l,\ \_,\ r\rangle = (|l| < |r|)$$

$$lpath\ \langle\rangle = []$$
$$lpath\ \langle l,\ a,\ r\rangle = \langle l,\ a,\ r\rangle\ \#\ lpath\ l$$

Towards the proof

$rheavy\ \langle l,\ \_,\ r\rangle = (|l| < |r|)$

$lpath\ \langle\rangle = []$
$lpath\ \langle l,\ a,\ r\rangle = \langle l,\ a,\ r\rangle\ \#\ lpath\ l$

$G\ h = length\ (filter\ rheavy\ (lpath\ h))$

Towards the proof

$rheavy\ \langle l,\ \_,\ r\rangle = (|l| < |r|)$

$lpath\ \langle\rangle = []$
$lpath\ \langle l,\ a,\ r\rangle = \langle l,\ a,\ r\rangle\ \#\ lpath\ l$

$G\ h = length\ (filter\ rheavy\ (lpath\ h))$

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

Towards the proof

$rheavy\ \langle l,\ \_,\ r\rangle = (|l| < |r|)$

$lpath\ \langle\rangle = []$
$lpath\ \langle l,\ a,\ r\rangle = \langle l,\ a,\ r\rangle\ \#\ lpath\ l$

$G\ h = length\ (filter\ rheavy\ (lpath\ h))$

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

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

Towards the proof

$rheavy\ \langle l,\ \_,\ r\rangle = (|l| < |r|)$

$rpath\ \langle\rangle = []$
$rpath\ \langle l,\ a,\ r\rangle = \langle l,\ a,\ r\rangle\ \#\ rpath\ r$

$D\ h = length(filter\ (\lambda p.\ \neg\ rheavy\ p)\ (rpath\ h))$

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

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

Potential

The potential is the number of *rheavy* nodes:

Potential

The potential is the number of *rheavy* nodes:

$\Phi\ \langle\rangle = 0$

$\Phi\ \langle l,\ \_,\ r\rangle = \Phi\ l + \Phi\ r + (\text{if } |l| < |r| \text{ then } 1 \text{ else } 0)$

Potential

The potential is the number of *rheavy* nodes:

$\Phi\ \langle\rangle = 0$

$\Phi\ \langle l,\ \_,\ r\rangle = \Phi\ l + \Phi\ r + (\text{if } |l| < |r| \text{ then } 1 \text{ else } 0)$

merge descends on the right $\Longrightarrow$ *rheavy* nodes are bad

Potential

The potential is the number of *rheavy* nodes:

$\Phi\ \langle\rangle = 0$

$\Phi\ \langle l,\ \_,\ r\rangle = \Phi\ l + \Phi\ r + (\text{if } |l| < |r| \text{ then } 1 \text{ else } 0)$

merge descends on the right $\Longrightarrow$ *rheavy* nodes are bad

Lemma

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$
$\leq G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1$

Main proof

$t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2$

Main proof

$$\begin{aligned}
&t_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2\\
&\leq G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1\\
&\leq \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1
\end{aligned}$$

Main proof

$$\begin{aligned}
&t_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2\\
&\leq G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1\\
&\leq \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1\\
&= \log_2\ (|t_1|_1 + |t_2|_1 - 1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1
\end{aligned}$$

Main proof

$$\begin{aligned}
&t_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2\\
&\leq G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1\\
&\leq \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1\\
&= \log_2\ (|t_1|_1 + |t_2|_1 - 1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1\\
&\leq \log_2\ (|t_1|_1 + |t_2|_1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1
\end{aligned}$$

Main proof

$$
\begin{aligned}
& t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2 \\
& \le G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1 \\
& \le \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& = \log_2\ (|t_1|_1 + |t_2|_1 - 1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& \le \log_2\ (|t_1|_1 + |t_2|_1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& \le \log_2\ (|t_1|_1 + |t_2|_1) + 2 * \log_2\ (|t_1|_1 + |t_2|_1) + 1 \\
& \quad \text{because } \log_2\ x + \log_2\ y \le 2 * \log_2\ (x + y) \text{ if } x,y > 0
\end{aligned}
$$

Main proof

$$
\begin{aligned}
& t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2 \\
& \le G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1 \\
& \le \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& = \log_2\ (|t_1|_1 + |t_2|_1 - 1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& \le \log_2\ (|t_1|_1 + |t_2|_1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& \le \log_2\ (|t_1|_1 + |t_2|_1) + 2 * \log_2\ (|t_1|_1 + |t_2|_1) + 1 \\
& \quad \text{because } \log_2\ x + \log_2\ y \le 2 * \log_2\ (x + y) \text{ if } x,y > 0 \\
& = 3 * \log_2\ (|t_1|_1 + |t_2|_1) + 1
\end{aligned}
$$

Main proof

$$
\begin{aligned}
& t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2 \\
& \le G\ (merge\ t_1\ t_2) + D\ t_1 + D\ t_2 + 1 \\
& \le \log_2\ |merge\ t_1\ t_2|_1 + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& = \log_2\ (|t_1|_1 + |t_2|_1 - 1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1 \\
& \le \log_2\ (|t_1|_1 + |t_2|_1) + \log_2\ |t_1|_1 + \log_2\ |t_2|_1 + 1
\end{aligned}
$$

Main proof

$$
t\_merge\ t_1\ t_2 + \Phi\ (merge\ t_1\ t_2) - \Phi\ t_1 - \Phi\ t_2
$$

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.

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.

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

Functions $isin$, $insert$ and $delete$
have amortized logarithmic complexity.

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.
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* :

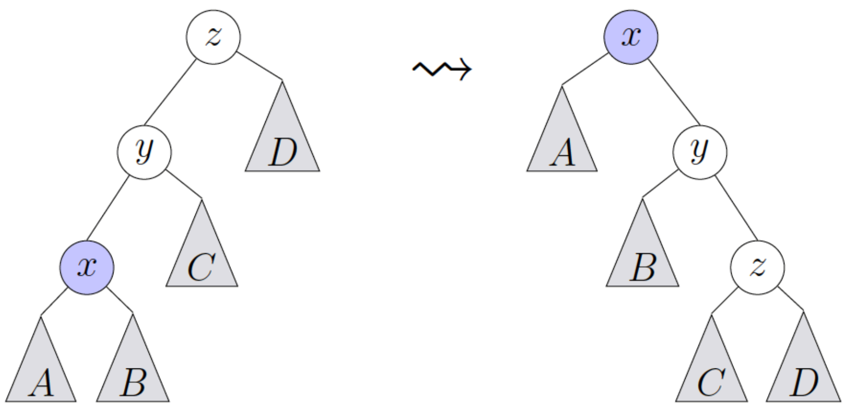
1. *splay a*
2. Perform *isin/insert/delete* action

Key ideas

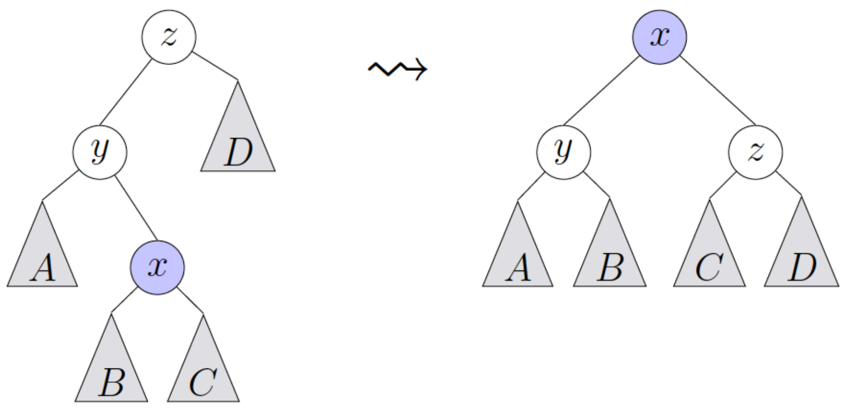
Move to root

Double rotations

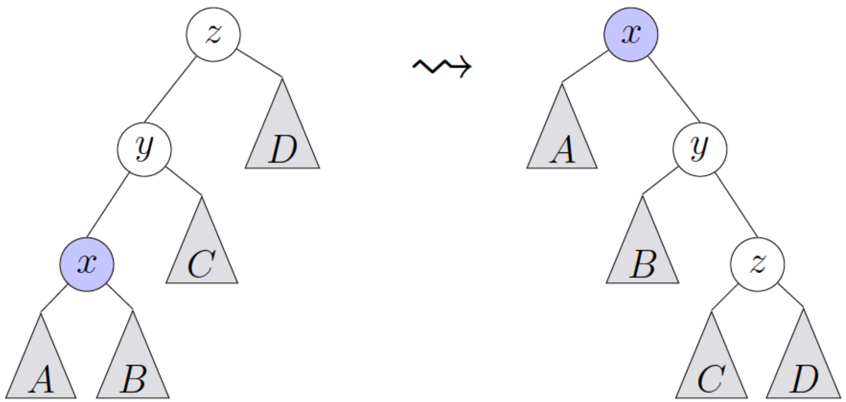
Zig-zig



Zig-zag



Zig-zig



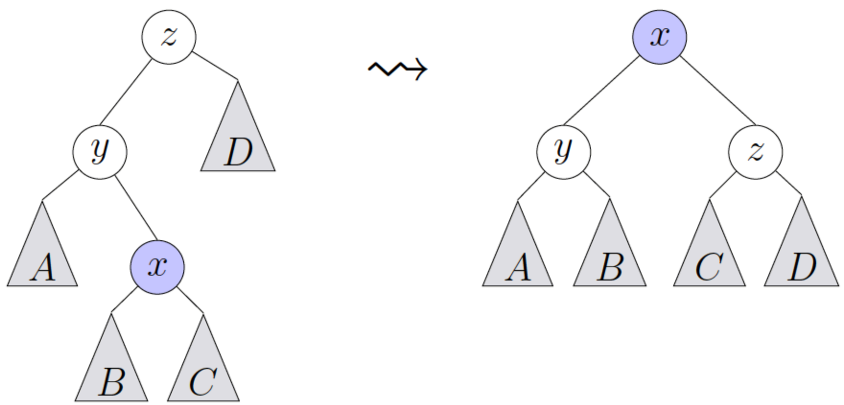
Functional definition

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

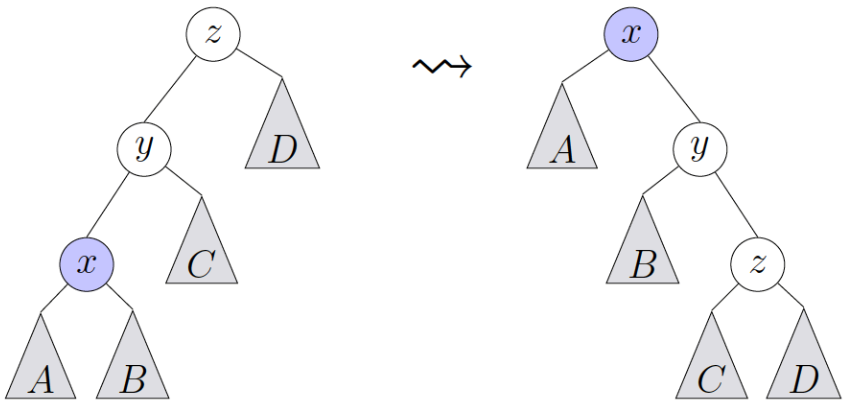
Zig-zig and zig-zag

$$\begin{array}{l}
[\![a < y;\ a < z;\ T \neq \langle\rangle]\!] \\
\Longrightarrow splay\ a\ \langle\langle T,\ y,\ C\rangle,\ z,\ D\rangle = \\
\quad (\text{case } splay\ a\ T \text{ of} \\
\qquad \langle A,\ x,\ B\rangle \Rightarrow \langle A,\ x,\ \langle B,\ y,\ \langle C,\ z,\ D\rangle\rangle\rangle)
\end{array}$$

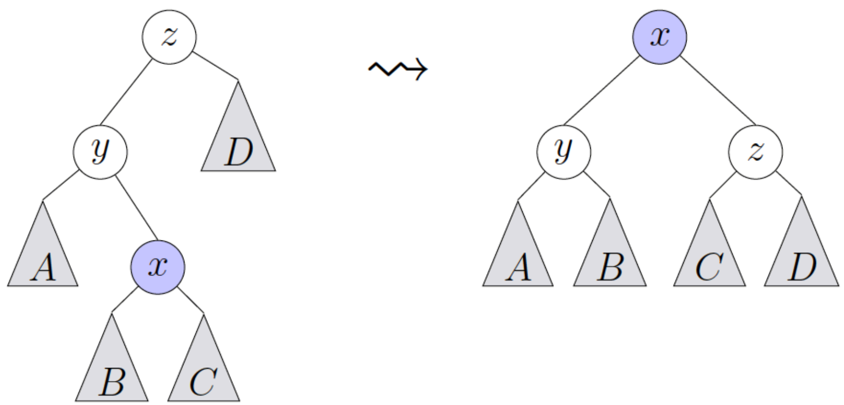
Zig-zag



Zig-zig



Zig-zag



Zig-zig and zig-zag

$[\![a < y;\ a < z;\ T \neq \langle\rangle]\!]$
$\Longrightarrow \mathit{splay}\ a\ \langle\langle T,\ y,\ C\rangle,\ z,\ D\rangle =$
(case $\mathit{splay}\ a\ T$ of
$\langle A,\ x,\ B\rangle \Rightarrow \langle A,\ x,\ \langle B,\ y,\ \langle C,\ z,\ D\rangle\rangle\rangle$)

Zig-zig and zig-zag

$[\![a < y;\ a < z;\ T \neq \langle\rangle]\!]$
$\Longrightarrow \mathit{splay}\ a\ \langle\langle T,\ y,\ C\rangle,\ z,\ D\rangle =$
(case $\mathit{splay}\ a\ T$ of
$\langle A,\ x,\ B\rangle \Rightarrow \langle A,\ x,\ \langle B,\ y,\ \langle C,\ z,\ D\rangle\rangle\rangle$)

216

Some base cases

$a < y \Longrightarrow \textit{splay}\ a\ \langle\langle A,\ a,\ B\rangle,\ y,\ C\rangle =$

Some base cases

$a < y \Longrightarrow \textit{splay}\ a\ \langle\langle A,\ a,\ B\rangle,\ y,\ C\rangle = \langle A,\ a,\ \langle B,\ y,\ C\rangle\rangle$

Some base cases

$a < y \Longrightarrow \textit{splay}\ a\ \langle\langle A,\ a,\ B\rangle,\ y,\ C\rangle = \langle A,\ a,\ \langle B,\ y,\ C\rangle\rangle$

$a < x \Longrightarrow$
$\textit{splay}\ a\ \langle\langle\langle\rangle,\ x,\ A\rangle,\ y,\ B\rangle =$

Some base cases

$a < y \Longrightarrow \textit{splay}\ a\ \langle\langle A,\ a,\ B\rangle,\ y,\ C\rangle = \langle A,\ a,\ \langle B,\ y,\ C\rangle\rangle$

$a < x \Longrightarrow$
$\textit{splay}\ a\ \langle\langle\langle\rangle,\ x,\ A\rangle,\ y,\ B\rangle = \langle\langle\rangle,\ x,\ \langle A,\ y,\ B\rangle\rangle$

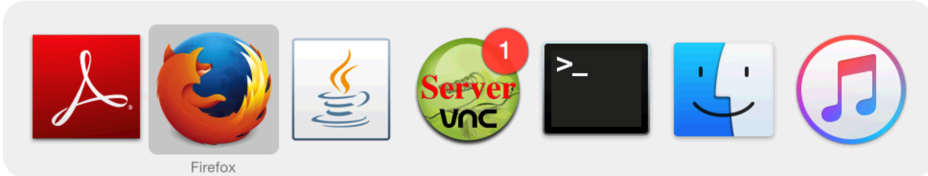
Functional correctness proofs

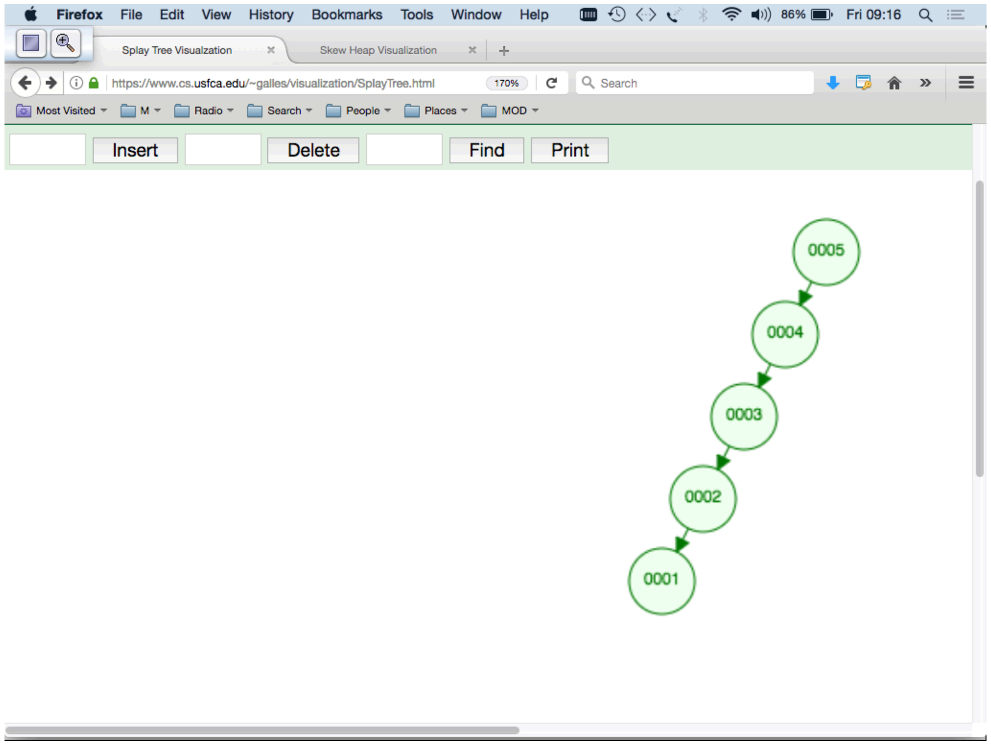
including preservation of bst

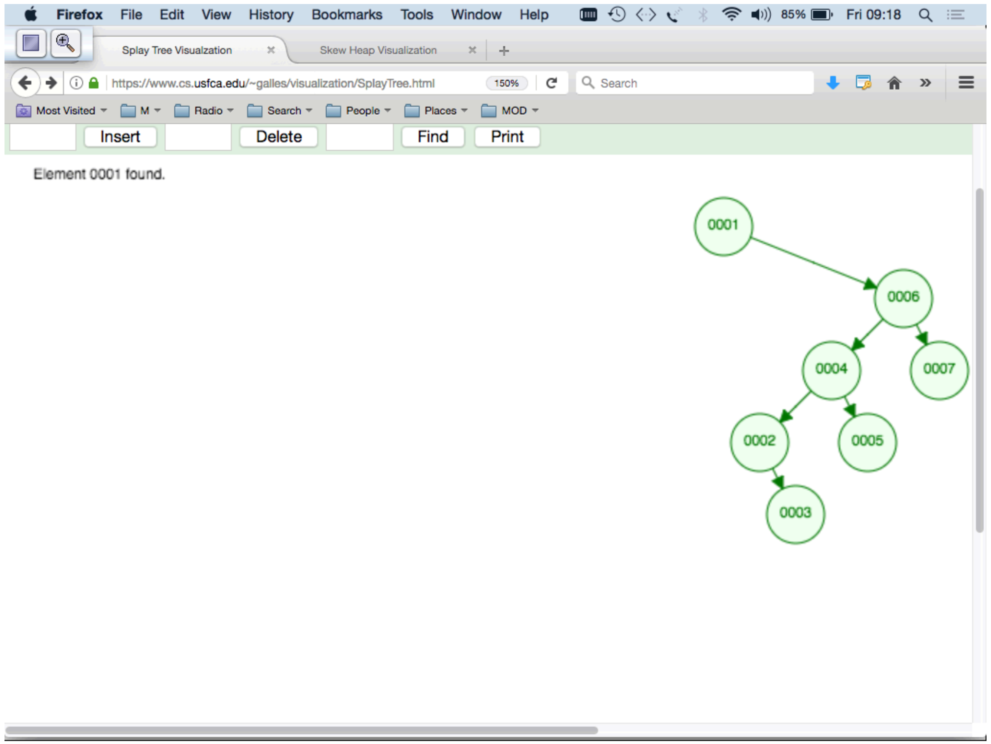
Straightforward

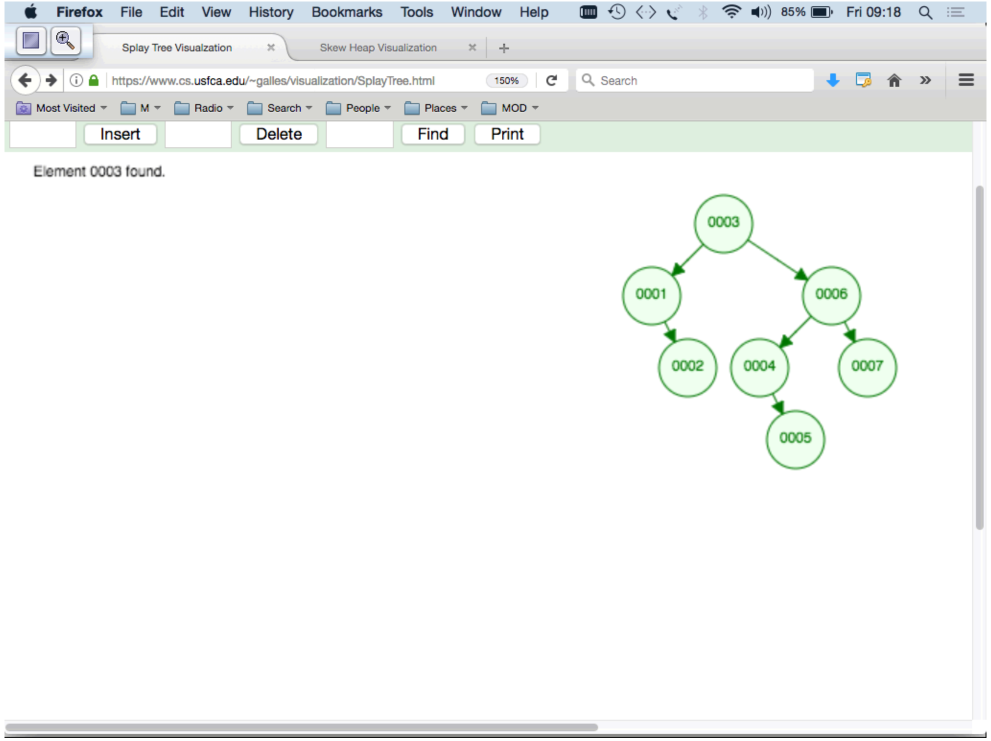
Functional correctness proofs

including preservation of bst









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

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

Analysis of *splay*

Theorem

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

Corollary

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

Analysis of *splay*

Theorem

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

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

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

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

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

Corollary

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

Corollary

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

Lemma

$[\![t \neq \langle\rangle;\ bst\ t]\!]$
$\Longrightarrow \exists\, a' \in set\_tree\ t.$
$\quad splay\ a'\ t = splay\ a\ t \wedge t\_splay\ a'\ t = t\_splay\ a\ t$

insert

Definition

$insert\ x\ t =$
(if $t = \langle\rangle$ then $\langle\langle\rangle,\ x,\ \langle\rangle\rangle$
else case $splay\ x\ t$ of
$\langle l,\ a,\ r\rangle \Rightarrow$
if $x = a$ then $\langle l,\ a,\ r\rangle$
else if $x < a$ then $\langle l,\ x,\ \langle\langle\rangle,\ a,\ r\rangle\rangle$
else $\langle\langle l,\ a,\ \langle\rangle\rangle,\ x,\ r\rangle$)

insert

Definition

$insert\ x\ t =$
(if $t = \langle\rangle$ then $\langle\langle\rangle,\ x,\ \langle\rangle\rangle$
else case $splay\ x\ t$ of
$\langle l,\ a,\ r\rangle \Rightarrow$
if $x = a$ then $\langle l,\ a,\ r\rangle$
else if $x < a$ then $\langle l,\ x,\ \langle\langle\rangle,\ a,\ r\rangle\rangle$
else $\langle\langle l,\ a,\ \langle\rangle\rangle,\ x,\ r\rangle$)

Counting only the cost of $splay$:

Lemma

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

insert

Definition

$insert\ x\ t =$
(if $t = \langle\rangle$ then $\langle\langle\rangle,\ x,\ \langle\rangle\rangle$
else case $splay\ x\ t$ of
$\langle l,\ a,\ r\rangle \Rightarrow$
if $x = a$ then $\langle l,\ a,\ r\rangle$
else if $x < a$ then $\langle l,\ x,\ \langle\langle\rangle,\ a,\ r\rangle\rangle$
else $\langle\langle l,\ a,\ \langle\rangle\rangle,\ x,\ r\rangle$)

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
$\langle l,\ a,\ r\rangle \Rightarrow$
if $x = a$
then if $l = \langle\rangle$ then r
else case $splay\_max\ l$ of
$\langle l',\ m,\ r'\rangle \Rightarrow \langle l',\ m,\ r\rangle$
else $\langle l,\ a,\ r\rangle$)

delete

Definition

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

Lemma

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

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;
$\_ = incr\ bad;$
$\_ = incr\ bad;$
$\_ = incr\ bad;$
$\vdots$

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

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

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:

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

Solution 2:

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

Client uses $splay$ before calling is_root:

Definition

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

Solution 2:

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

Client uses $splay$ before calling is_root:

Definition

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

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.

The formalization is based on

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

Implementation type

datatype $'a\ heap = Empty \mid Hp\ 'a\ ('a\ heap\ list)$

Implementation type

datatype $'a\ heap = Empty \mid Hp\ 'a\ ('a\ heap\ list)$

Heap invariant:

$pheap\ Empty = True$
$pheap\ (Hp\ x\ hs) =$
$(\forall h \in set\ hs.\ (\forall y \in set\_heap\ h.\ x \leq y) \wedge pheap\ h)$

Also: $Empty$ must only occur at the root

$merge$ and $insert$

$merge\ h\ Empty = h$
$merge\ Empty\ h = h$

$merge$ and $insert$

$merge\ h\ Empty = h$
$merge\ Empty\ h = h$

$merge\ (hx = Hp\ x\ hsx)\ (hy = Hp\ y\ hsy) =$
(if $x < y$ then $Hp\ x\ (hy\ \#\ hsx)$ else $Hp\ y\ (hx\ \#\ hsy)$)

$merge\_pairs$ and del_min

$merge\_pairs\ [] = Empty$
$merge\_pairs\ [h] = h$
$merge\_pairs\ (h_1\ \#\ h_2\ \#\ hs) =$
$merge\ (merge\ h_1\ h_2)\ (merge\_pairs\ hs)$

$merge\_pairs$ and del_min

$$merge\_pairs\ [\,] = Empty$$
$$merge\_pairs\ [h] = h$$
$$merge\_pairs\ (h_1\ \#\ h_2\ \#\ hs) =$$
$$merge\ (merge\ h_1\ h_2)\ (merge\_pairs\ hs)$$

$$del\_min\ Empty = Empty$$
$$del\_min\ (Hp\ x\ hs) = merge\_pairs\ hs$$

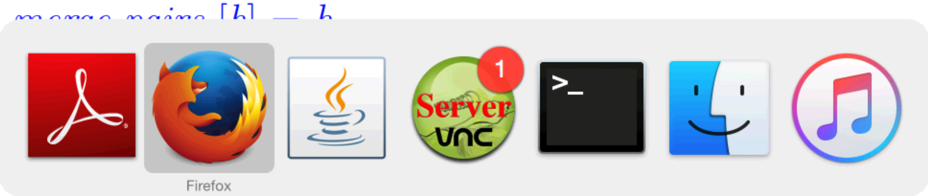
$merge\_pairs$ and del_min

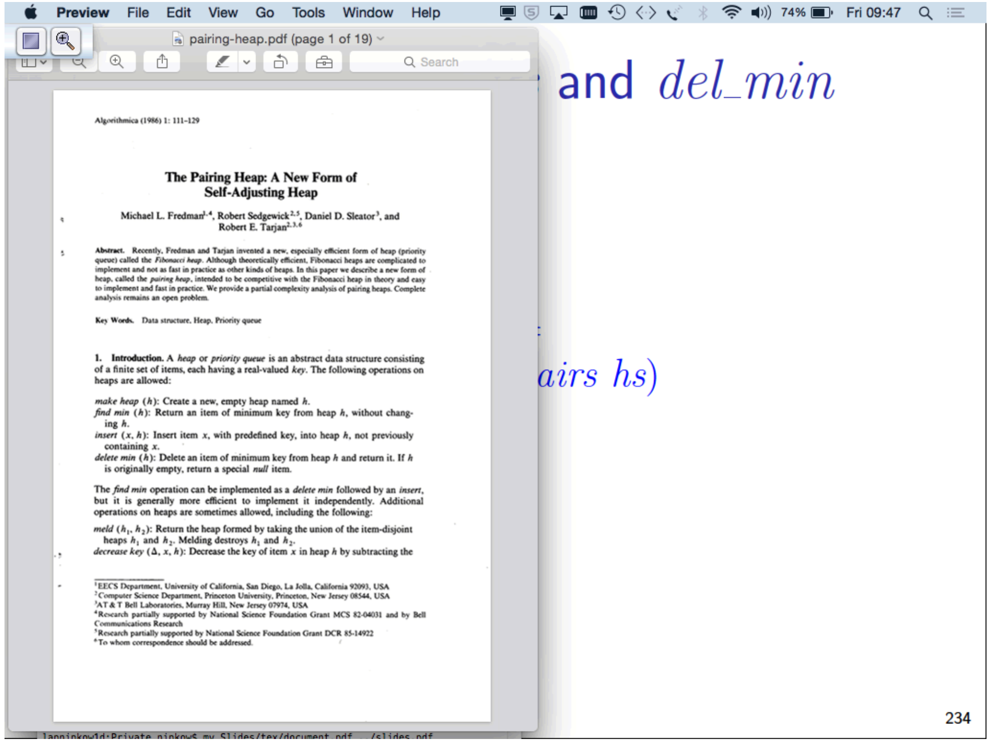
$$merge\_pairs\ [\,] = Empty$$
$$merge\_pairs\ [h] = h$$
$$merge\_pairs\ (h_1\ \#\ h_2\ \#\ hs) =$$
$$merge\ (merge\ h_1\ h_2)\ (merge\_pairs\ hs)$$

234

$merge\_pairs$ and del_min

$$merge\_pairs\ [\,] = Empty$$





Algorithmica (1986) 1: 111–129

The Pairing Heap: A New Form of Self-Adjusting Heap

Michael L. Fredman[1,4], Robert Sedgewick[2,5], Daniel D. Sleator[3], and Robert E. Tarjan[2,3,6]

Abstract. Recently, Fredman and Tarjan invented a new, especially efficient form of heap (priority queue) called the *Fibonacci heap*. Although theoretically efficient, Fibonacci heaps are complicated to implement and not as fast in practice as other kinds of heaps. In this paper we describe a new form of heap, called the *pairing heap*, intended to be competitive with the Fibonacci heap in theory and easy to implement and fast in practice. We provide a partial complexity analysis of pairing heaps. Complete analysis remains an open problem.

Key Words. Data structure, Heap, Priority queue

1. Introduction. A *heap* or *priority queue* is an abstract data structure consisting of a finite set of items, each having a real-valued *key*. The following operations on heaps are allowed:

make heap (h): Create a new, empty heap named h.
find min (h): Return an item of minimum key from heap h, without changing h.
insert (x, h): Insert item x, with predefined key, into heap h, not previously containing x.
delete min (h): Delete an item of minimum key from heap h and return it. If h is originally empty, return a special *null* item.

The *find min* operation can be implemented as a *delete min* followed by an *insert*, but it is generally more efficient to implement it independently. Additional operations on heaps are sometimes allowed, including the following:

meld (h_1, h_2): Return the heap formed by taking the union of the item-disjoint heaps h_1 and h_2. Melding destroys h_1 and h_2.
decrease key (Δ, x, h): Decrease the key of item x in heap h by subtracting the

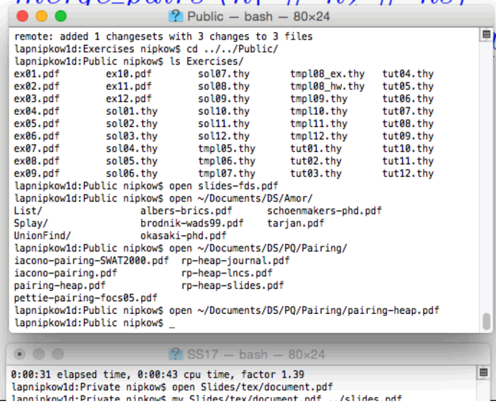
[1] EECS Department, University of California, San Diego, La Jolla, California 92093, USA
[2] Computer Science Department, Princeton University, Princeton, New Jersey 08544, USA
[3] AT & T Bell Laboratories, Murray Hill, New Jersey 07974, USA
[4] Research partially supported by National Science Foundation Grant MCS 82-04031 and by Bell Communications Research
[5] Research partially supported by National Science Foundation Grant DCR 85-14922
[6] To whom correspondence should be addressed.

and del_min

$airs\ hs)$

merge_pairs and *del_min*

$merge\_pairs\ [\,] = Empty$
$merge\_pairs\ [h] = h$
$merge\_pairs\ (h_1\ \#\ h_2\ \#\ hs) =$
$\ldots pairs\ hs)$



Analysis

Analysis easier (more uniform) if a pairing heap is viewed as a binary tree:

$homs :: 'a\ heap\ list \Rightarrow 'a\ tree$
$homs\ [\,] = \langle\rangle$
$homs\ (Hp\ x\ hs_1\ \#\ hs_2) = \langle homs\ hs_1,\ x,\ homs\ hs_2\rangle$

Analysis

Analysis easier (more uniform) if a pairing heap is viewed as a binary tree:

$homs :: 'a\ heap\ list \Rightarrow 'a\ tree$
$homs\ [\,] = \langle\rangle$
$homs\ (Hp\ x\ hs_1\ \#\ hs_2) = \langle homs\ hs_1,\ x,\ homs\ hs_2\rangle$

$hom :: 'a\ heap \Rightarrow 'a\ tree$
$hom\ Empty = \langle\rangle$
$hom\ (Hp\ x\ hs) = \langle homs\ hs,\ x,\ \langle\rangle\rangle$

Analysis

Analysis easier (more uniform) if a pairing heap is viewed as a binary tree:

$homs :: 'a\ heap\ list \Rightarrow 'a\ tree$
$homs\ [\,] = \langle\rangle$
$homs\ (Hp\ x\ hs_1\ \#\ hs_2) = \langle homs\ hs_1,\ x,\ homs\ hs_2\rangle$

$hom :: 'a\ heap \Rightarrow 'a\ tree$
$hom\ Empty = \langle\rangle$
$hom\ (Hp\ x\ hs) = \langle homs\ hs,\ x,\ \langle\rangle\rangle$

Potential function same as for splay trees

Verified:

The functions $insert$, del_min and $merge$ all have $O(\log_2 n)$ amortized complexity.

Verified:

The functions $insert$, del_min and $merge$ all have $O(\log_2 n)$ amortized complexity.

These bounds are not tight.
Better amortized bounds in the literature:

$insert \in O(1)$, $del\_min \in O(\log_2 n)$, $merge \in O(1)$

Verified:

The functions $insert$, del_min and $merge$ all have $O(\log_2 n)$ amortized complexity.

These bounds are not tight.
Better amortized bounds in the literature:

$insert \in O(1)$, $del\_min \in O(\log_2 n)$, $merge \in O(1)$

The exact complexity is still open.

Sources

The inventors of the pairing heap:

M. Fredman, R. Sedgewick, D. Sleator and R. Tarjan.
The Pairing Heap: A New Form of Self-Adjusting Heap.
Algorithmica, 1986.

Sources

The inventors of the pairing heap:

M. Fredman, R. Sedgewick, D. Sleator and R. Tarjan.
The Pairing Heap: A New Form of Self-Adjusting Heap.
Algorithmica, 1986.

The functional version:

Chris Okasaki. *Purely Functional Data Structures.*
Cambridge University Press, 1998.

Verified:

> The functions $insert$, del_min and $merge$ all have $O(\log_2 n)$ amortized complexity.

These bounds are not tight.
Better amortized bounds in the literature:

$insert \in O(1)$, $del\_min \in O(\log_2 n)$, $merge \in O(1)$

Sources

The inventors of the pairing heap:

M. Fredman, R. Sedgewick, D. Sleator and R. Tarjan.
The Pairing Heap: A New Form of Self-Adjusting Heap.
Algorithmica, 1986.

Verified:

> The functions $insert$, del_min and $merge$ all have $O(\log_2 n)$ amortized complexity.

`Thys/Pairing_Heap.thy`