Script generated by TTT

Title: FDS (08.06.2018)

Date: Fri Jun 08 08:32:13 CEST 2018

Duration: 93:00 min

Pages: 89

`HOL/Data_Structures/RBT_Set.thy`

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;

Relationship to 2-3-4 trees

Idea: encode 2-3-4 trees as binary trees;
use color to express grouping

$$\langle\rangle \approx \langle\rangle$$

Color

Structural invariants

$invh :: {'a}\ rbt \Rightarrow bool$

$invh\ \langle\rangle = True$

$invh\ \langle\_,\ l,\ \_,\ r\rangle = (invh\ l \wedge invh\ r \wedge bh(l) = bh(r))$

Logarithmic height

Lemma

$rbt\ t \Longrightarrow h(t) \leq 2 * \log_2 |t|_1$

Insertion

$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

Insertion

$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$insert\ x\ t = paint\ Black\ (ins\ x\ t)$

Insertion

$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$insert\ x\ t = paint\ Black\ (ins\ x\ t)$

$ins :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$ins\ x\ \langle\rangle = R\ \langle\rangle\ x\ \langle\rangle$

Insertion

$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$insert\ x\ t = paint\ Black\ (ins\ x\ t)$

$ins :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$ins\ x\ \langle\rangle = R\ \langle\rangle\ x\ \langle\rangle$

$$
\begin{aligned}
ins\ x\ (B\ l\ a\ r) = (&\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\ \ LT \Rightarrow baliL\ (ins\ x\ l)\ a\ r \\
&|\ EQ \Rightarrow B\ l\ a\ r \\
&|\ GT \Rightarrow baliR\ l\ a\ (ins\ x\ r))
\end{aligned}
$$

Insertion

$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$insert\ x\ t = paint\ Black\ (ins\ x\ t)$

$ins :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$$
\begin{aligned}
&ins\ x\ \langle\rangle = R\ \langle\rangle\ x\ \langle\rangle \\
&ins\ x\ (B\ l\ a\ r) = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad LT \Rightarrow baliL\ (ins\ x\ l)\ a\ r \\
&\qquad |\ EQ \Rightarrow B\ l\ a\ r \\
&\qquad |\ GT \Rightarrow baliR\ l\ a\ (ins\ x\ r)) \\
&ins\ x\ (R\ l\ a\ r) = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad LT \Rightarrow R\ (ins\ x\ l)\ a\ r \\
&\qquad |\ EQ \Rightarrow R\ l\ a\ r \\
&\qquad |\ GT \Rightarrow R\ l\ a\ (ins\ x\ r))
\end{aligned}
$$

Adjusting colors

$baliL,\ baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

Insertion

$insert :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$insert\ x\ t = paint\ Black\ (ins\ x\ t)$

$ins :: 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

$$
\begin{aligned}
&ins\ x\ \langle\rangle = R\ \langle\rangle\ x\ \langle\rangle \\
&ins\ x\ (B\ l\ a\ r) = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad LT \Rightarrow baliL\ (ins\ x\ l)\ a\ r \\
&\qquad |\ EQ \Rightarrow B\ l\ a\ r \\
&\qquad |\ GT \Rightarrow baliR\ l\ a\ (ins\ x\ r)) \\
&ins\ x\ (R\ l\ a\ r) = (\textsf{case}\ cmp\ x\ a\ \textsf{of} \\
&\qquad LT \Rightarrow R\ (ins\ x\ l)\ a\ r \\
&\qquad |\ EQ \Rightarrow R\ l\ a\ r \\
&\qquad |\ GT \Rightarrow R\ l\ a\ (ins\ x\ r))
\end{aligned}
$$

Adjusting colors

$baliL,\ baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $= R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $= R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $= R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $= R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $= R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
- Principle: replace Red-Red by Red-Black

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
- Principle: replace Red-Red by Red-Black
- Final equation:
  $baliL\ l\ a\ r = B\ l\ a\ r$

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
- Principle: replace Red-Red by Red-Black

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

Adjusting colors

$baliL, baliR :: 'a\ rbt \Rightarrow 'a \Rightarrow 'a\ rbt \Rightarrow 'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
- Principle: replace Red-Red by Red-Black
- Final equation:

Adjusting colors

$baliL,\ baliR :: {}'a\ rbt \Rightarrow {}'a \Rightarrow {}'a\ rbt \Rightarrow {}'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
- Principle: replace Red-Red by Red-Black
- Final equation:
  $baliL\ l\ a\ r = B\ l\ a\ r$

Logarithmic height

Lemma
$rbt\ t \Longrightarrow h(t) \leq 2 * \log_2 |t|_1$

Adjusting colors

$baliL,\ baliR :: {}'a\ rbt \Rightarrow {}'a \Rightarrow {}'a\ rbt \Rightarrow {}'a\ rbt$

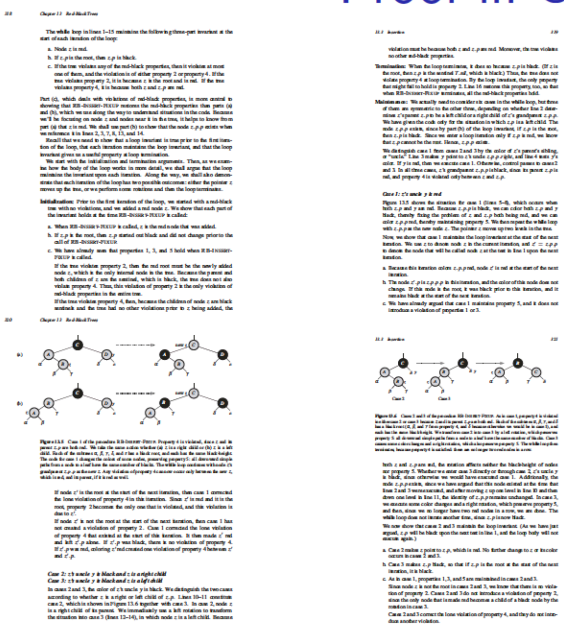
- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$

Adjusting colors

$baliL,\ baliR :: {}'a\ rbt \Rightarrow {}'a \Rightarrow {}'a\ rbt \Rightarrow {}'a\ rbt$

- Combine arguments $l\ a\ r$ into tree, ideally $\langle l,\ a,\ r\rangle$
- Treat invariant violation Red-Red in l/r
  $baliL\ (R\ (R\ t_1\ a_1\ t_2)\ a_2\ t_3)\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
  $baliL\ (R\ t_1\ a_1\ (R\ t_2\ a_2\ t_3))\ a_3\ t_4$
  $\quad = R\ (B\ t_1\ a_1\ t_2)\ a_2\ (B\ t_3\ a_3\ t_4)$
- Principle: replace Red-Red by Red-Black
- Final equation:
  $baliL\ l\ a\ r = B\ l\ a\ r$

Proof in CLRS



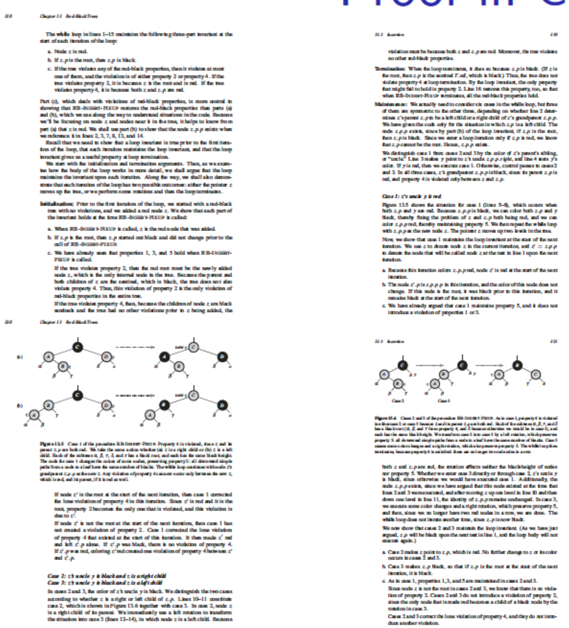
Preservation of invariant

After 14 simple lemmas:

Theorem

$rbt\ t \Longrightarrow rbt\ (insert\ x\ t)$

Proof in CLRS



Deletion

$delete\ x\ t = paint\ Black\ (del\ x\ t)$

Deletion

$delete\ x\ t = paint\ Black\ (del\ x\ t)$

$del\ \_\ \langle\rangle = \langle\rangle$
$del\ x\ \langle\_,\ l,\ a,\ r\rangle =$
$(\text{case}\ cmp\ x\ a\ \text{of}$
$\quad LT \Rightarrow$
$\quad\quad \text{if}\ l \neq \langle\rangle \wedge color\ l = Black$
$\quad\quad \text{then}\ baldL\ (del\ x\ l)\ a\ r\ \text{else}\ R\ (del\ x\ l)\ a\ r$
$\mid EQ \Rightarrow combine\ l\ r$
$\mid GT \Rightarrow$
$\quad\quad \text{if}\ r \neq \langle\rangle \wedge color\ r = Black$
$\quad\quad \text{then}\ baldR\ l\ a\ (del\ x\ r)\ \text{else}\ R\ l\ a\ (del\ x\ r))$

Deletion

Tricky functions: $baldL,\ baldR,\ combine$

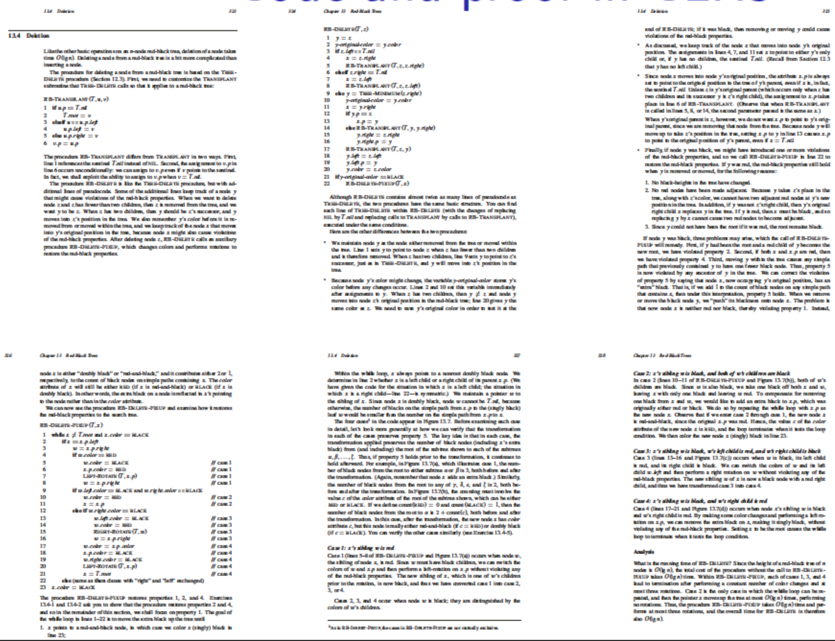
Deletion

Tricky functions: $baldL,\ baldR,\ combine$

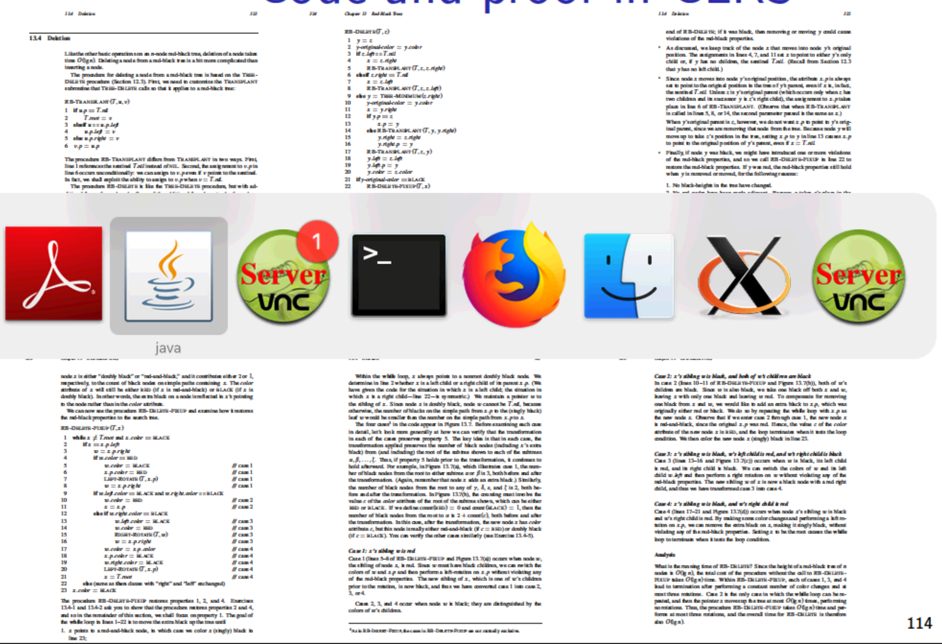
12 short but tricky to find invariant lemmas with short proofs. The worst:

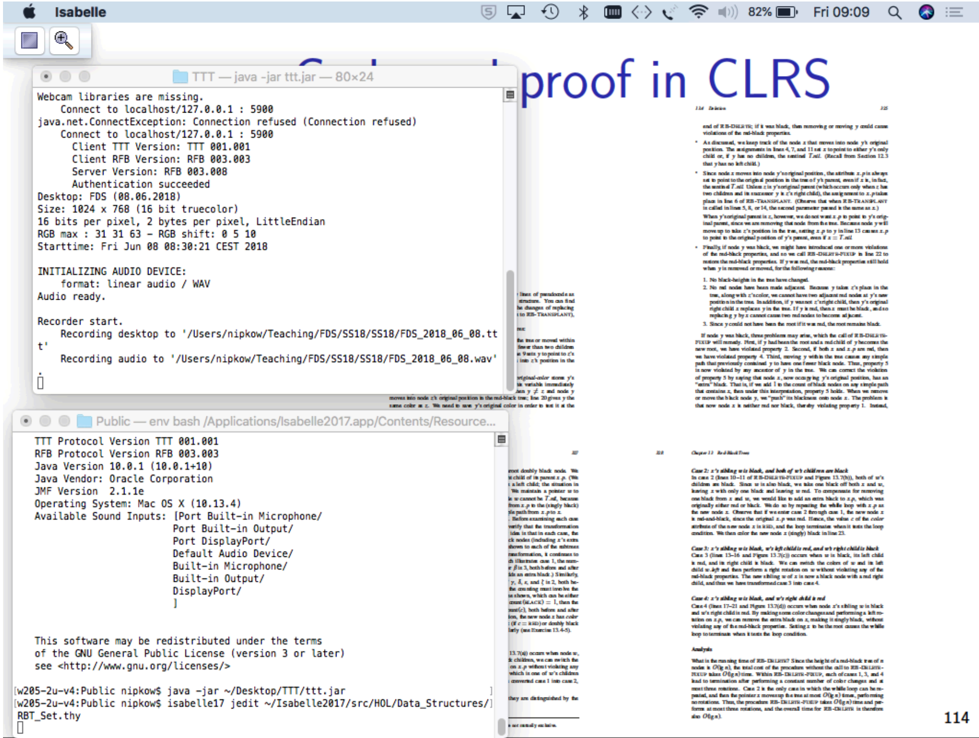
$[\![invh\ t;\ invc\ t]\!]$
$\Longrightarrow invh\ (del\ x\ t) \wedge$
$\quad (color\ t = Red \wedge$
$\quad\ bh(del\ x\ t) = bh(t) \wedge invc\ (del\ x\ t) \vee$
$\quad\ color\ t = Black \wedge$
$\quad\ bh(del\ x\ t) = bh(t) - 1 \wedge invc2\ (del\ x\ t))$

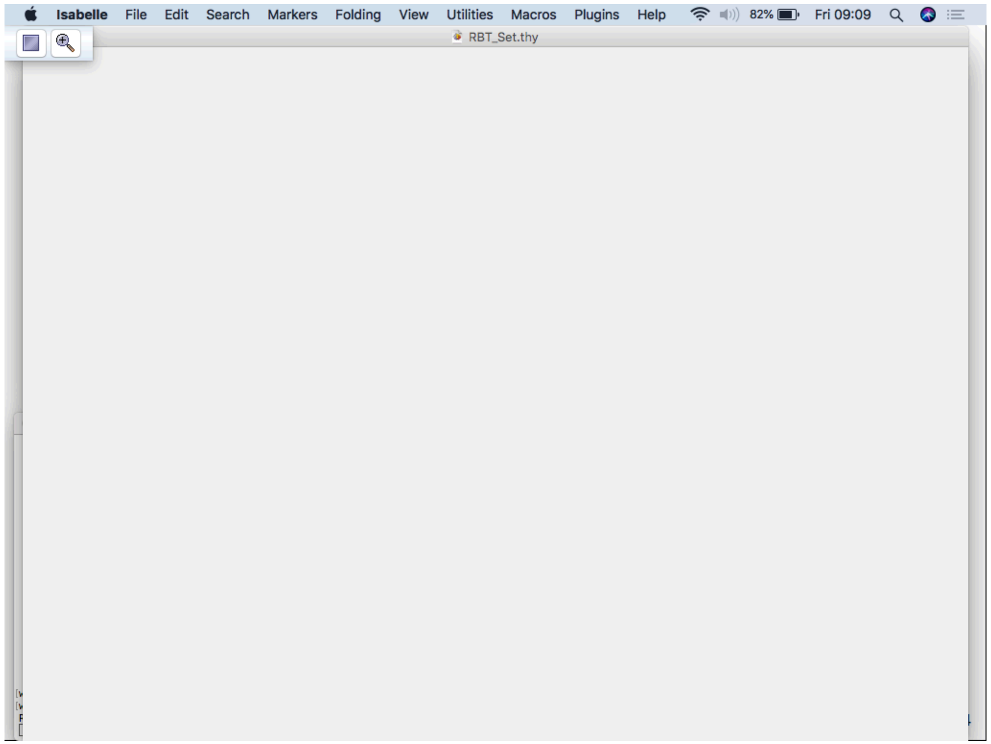
Code and proof in CLRS



Code and proof in CLRS







Source of code

Insertion:

Okasaki's *Purely Functional Data Structures*

Deletion:

Stefan Kahrs. *Red Black Trees with Types.*
J. Functional Programming. 1996.

AVL Trees

[Adelson-Velskii & Landis 62]

AVL Trees

[Adelson-Velskii & Landis 62]

- Every node $\langle l, \_, r \rangle$ must be balanced:
  $|h(l) - h(r)| \leq 1$

Weight-Balanced Trees

[Nievergelt & Reingold 72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$

Weight-Balanced Trees

[Nievergelt & Reingold 72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$
- Every node $\langle l,\_,r \rangle$ must be balanced:
  $\alpha \leq |l|_1/(|l|_1 + |r|_1) \leq 1{-}\alpha$

Weight-Balanced Trees

[Nievergelt & Reingold 72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$
- Every node $\langle l,\_,r \rangle$ must be balanced:
  $\alpha \leq |l|_1/(|l|_1 + |r|_1) \leq 1{-}\alpha$
- Insertion and deletion: single and double rotations depending on subtle numeric conditions

Weight-Balanced Trees

[Nievergelt & Reingold 72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$
- Every node $\langle l,\_,r \rangle$ must be balanced:
  $\alpha \leq |l|_1/(|l|_1 + |r|_1) \leq 1{-}\alpha$
- Insertion and deletion: single and double rotations depending on subtle numeric conditions
- Nievergelt and Reingold incorrect

Weight-Balanced Trees

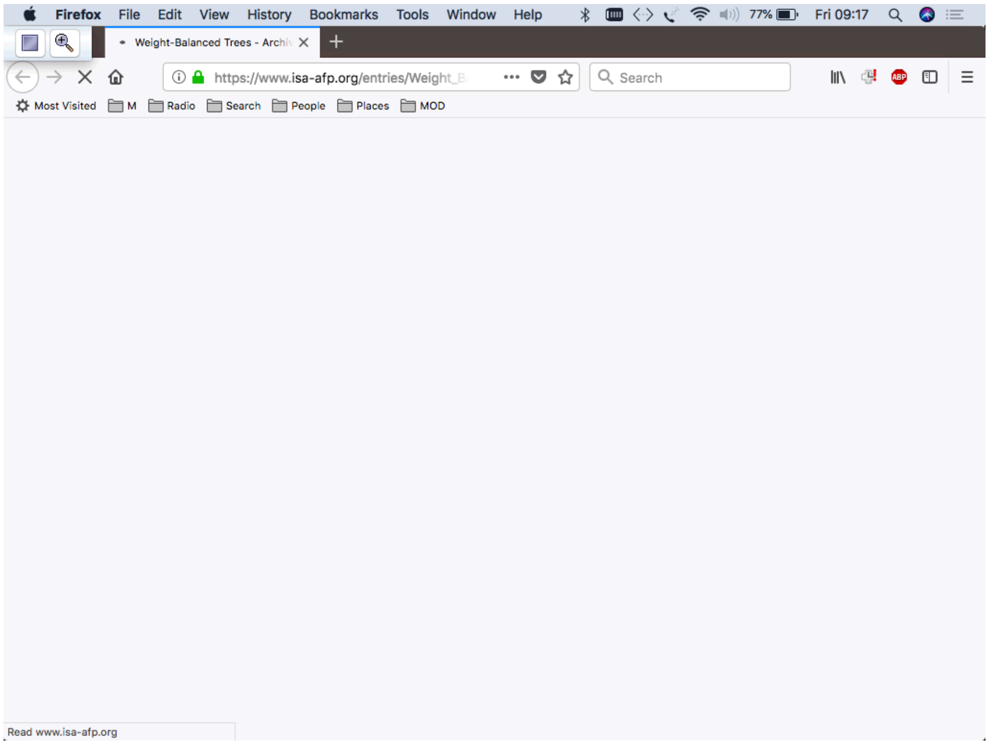
[Nievergelt & Reingold 72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$
- Every node $\langle l, \_, r \rangle$ must be balanced: $\alpha \leq |l|_1/(|l|_1 + |r|_1) \leq 1-\alpha$
- Insertion and deletion: single and double rotations depending on subtle numeric conditions
- Nievergelt and Reingold incorrect
- Mistakes discovered and corrected by [Blum & Mehlhorn 80] and [Hirai & Yamamoto 2011]

Weight-Balanced Trees

[Nievergelt & Reingold 72,73]

- Parameter: balance factor $0 < \alpha \leq 0.5$
- Every node $\langle l, \_, r \rangle$ must be balanced: $\alpha \leq |l|_1/(|l|_1 + |r|_1) \leq 1-\alpha$
- Insertion and deletion: single and double rotations depending on subtle numeric conditions
- Nievergelt and Reingold incorrect
- Mistakes discovered and corrected by [Blum & Mehlhorn 80] and [Hirai & Yamamoto 2011]
- Verified implementation in Isabelle's Archive of Formal Proofs.



AA trees

[Arne Andersson 93, Ragde 14]

AA trees

[Arne Andersson 93, Ragde 14]

- Simulation of 2-3 trees by binary trees
  $\langle t_1, a, t_2, b, t_3 \rangle \;\leadsto\; \langle t_1, a, \langle t_2, b, t_3 \rangle \rangle$

AA trees

[Arne Andersson 93, Ragde 14]

- Simulation of 2-3 trees by binary trees
  $\langle t_1, a, t_2, b, t_3 \rangle \;\leadsto\; \langle t_1, a, \langle t_2, b, t_3 \rangle \rangle$
- Height field (or single bit) to distinguish single from double node

AA trees

[Arne Andersson 93, Ragde 14]

- Simulation of 2-3 trees by binary trees
  $\langle t_1, a, t_2, b, t_3 \rangle \;\leadsto\; \langle t_1, a, \langle t_2, b, t_3 \rangle \rangle$
- Height field (or single bit) to distinguish single from double node
- Code short but opaque

AA trees

[Arne Andersson 93, Ragde 14]

- Simulation of 2-3 trees by binary trees
  $\langle t_1, a, t_2, b, t_3 \rangle \rightsquigarrow \langle t_1, a, \langle t_2, b, t_3 \rangle \rangle$
- Height field (or single bit) to distinguish single from double node
- Code short but opaque
- 4 bugs in *delete* in [Ragde 14]:
  non-linear pattern;

AA trees

[Arne Andersson 93, Ragde 14]

- Simulation of 2-3 trees by binary trees
  $\langle t_1, a, t_2, b, t_3 \rangle \rightsquigarrow \langle t_1, a, \langle t_2, b, t_3 \rangle \rangle$
- Height field (or single bit) to distinguish single from double node
- Code short but opaque
- 4 bugs in *delete* in [Ragde 14]:
  non-linear pattern; going down wrong subtree; missing function call; off by 1

AA trees

[Arne Andersson 93, Ragde 14]

After corrections, the proofs:

- Code relies on tricky pre- and post-conditions that need to be found

AA trees

[Arne Andersson 93, Ragde 14]

After corrections, the proofs:

- Code relies on tricky pre- and post-conditions that need to be found
- Structural invariant preservation requires most of the work

12 More Search Trees

Scapegoat trees

[Anderson 89, Igal & Rivest 93]

Scapegoat trees

[Anderson 89, Igal & Rivest 93]

Central idea:

Don't rebalance every time,
Rebuild when the tree gets "too unbalanced"

Scapegoat trees

[Anderson 89, Igal & Rivest 93]

Central idea:

Don't rebalance every time,
Rebuild when the tree gets "too unbalanced"

- Tricky: amortized logarithmic complexity analysis

One by one (Union)

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

$c(x)$ = cost of adding element to set of size x

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

$c(x)$ = cost of adding element to set of size x

- Smaller (m elements) into bigger (n elements):

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

$c(x)$ = cost of adding element to set of size x

- Smaller (m elements) into bigger (n elements):
  Cost = $c(n) + \cdots + c(n+m-1)$

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

$c(x)$ = cost of adding element to set of size x

- Smaller (m elements) into bigger (n elements):
  Cost = $c(n) + \cdots + c(n+m-1)$
- Bigger into smaller:
  Cost = $c(m) + \cdots + c(m+n-1)$

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

$c(x)$ = cost of adding element to set of size x

- Smaller (m elements) into bigger (n elements):
  Cost = $c(n) + \cdots + c(n+m-1)$
- Bigger into smaller:
  Cost = $c(m) + \cdots + c(m+n-1)$
- $c(x) = \log_2 x \implies$
  Cost = $O(m * \log_2(n+m))$

One by one (Union)

What is better:
Adding smaller set to bigger or bigger to smaller?

$c(x)$ = cost of adding element to set of size x

- Smaller (m elements) into bigger (n elements):
  Cost $= c(n) + \cdots + c(n+m-1)$
- Bigger into smaller:
  Cost $= c(m) + \cdots + c(m+n-1)$
- $c(x) = \log_2 x \Longrightarrow$
  Cost $= O(m * \log_2(n+m)) = O(m * \log_2 n)$

- We can do better than $O(m * \log_2 n)$

- We can do better than $O(m * \log_2 n)$
- This section:

  A parallel divide and conquer approach

- We can do better than $O(m * \log_2 n)$
- This section:

  A parallel divide and conquer approach
- Cost: $O(m * \log_2(\frac{n}{m} + 1))$

- We can do better than $O(m * \log_2 n)$
- This section:

  A parallel divide and conquer approach
- Cost: $O(m * \log_2(\frac{n}{m} + 1))$
- Works for many kinds of balanced trees

- We can do better than $O(m * \log_2 n)$
- This section:

  A parallel divide and conquer approach
- Cost: $O(m * \log_2(\frac{n}{m} + 1))$
- Works for many kinds of balanced trees
- For ease of presentation: use concrete type *tree*

Uniform *tree* type

Red-Black trees, AVL trees, weight-balanced trees, etc can all be implemented with one more field per node:

datatype $('a,\ 'b)\ \mathit{tree} = \langle\rangle$
$\quad |\ \mathit{Node}\ 'b\ (('a,\ 'b)\ \mathit{tree})\ 'a\ (('a,\ 'b)\ \mathit{tree})$

Uniform *tree* type

Red-Black trees, AVL trees, weight-balanced trees, etc can all be implemented with one more field per node:

datatype $('a,\ 'b)\ \mathit{tree} = \langle\rangle$
$\quad |\ \mathit{Node}\ 'b\ (('a,\ 'b)\ \mathit{tree})\ 'a\ (('a,\ 'b)\ \mathit{tree})$

We work with this type of trees without committing to any particular kind of balancing schema.

Uniform *tree* type

Red-Black trees, AVL trees, weight-balanced trees, etc can all be implemented with one more field per node:

datatype $('a, 'b)$ *tree* $= \langle\rangle$
$\mid$ *Node* $'b\ (('a, 'b)\ tree)\ 'a\ (('a, 'b)\ tree)$

We work with this type of trees without committing to any particular kind of balancing schema.

Syntax:

$$\langle b, l, a, r\rangle \equiv Node\ b\ l\ a\ r$$

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r$

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r \approx Node\ \_\ l\ a\ r$

Just *join*

Can synthesize all BST interface functions from just one function:

$join\ l\ a\ r \approx Node\ \_\ l\ a\ r +$ rebalance

Just *join*

Given $join :: tree \Rightarrow {'a} \Rightarrow tree \Rightarrow tree$
(where $tree$ abbreviates $('a,'b)\ tree$), implement

Just *join*

Given $join :: tree \Rightarrow {'a} \Rightarrow tree \Rightarrow tree$
(where $tree$ abbreviates $('a,'b)\ tree$), implement

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$

Just *join*

Given $join :: tree \Rightarrow {'a} \Rightarrow tree \Rightarrow tree$
(where $tree$ abbreviates $('a,'b)\ tree$), implement

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$
$insert :: {'a} \Rightarrow tree \Rightarrow tree$

Just *join*

Given $join :: tree \Rightarrow {'a} \Rightarrow tree \Rightarrow tree$
(where $tree$ abbreviates $('a,'b)\ tree$), implement

$split :: tree \Rightarrow {'a} \Rightarrow tree \times bool \times tree$
$insert :: {'a} \Rightarrow tree \Rightarrow tree$
$union :: tree \Rightarrow tree \Rightarrow tree$

13 Union, Intersection, Difference on BSTs

Correctness

join for Red-Black Trees

Just *join*

Given $join :: tree \Rightarrow 'a \Rightarrow tree \Rightarrow tree$
(where *tree* abbreviates $('a,'b)\ tree$), implement

$split :: tree \Rightarrow 'a \Rightarrow tree \times bool \times tree$
$insert :: 'a \Rightarrow tree \Rightarrow tree$
$union :: tree \Rightarrow tree \Rightarrow tree$
$join2 :: tree \Rightarrow tree \Rightarrow tree$
$delete :: 'a \Rightarrow tree \Rightarrow tree$