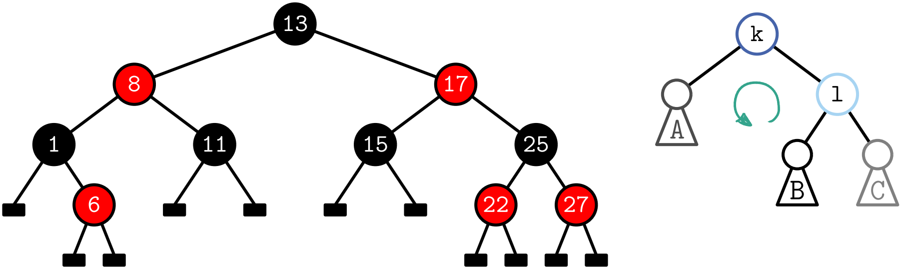
Karlsruhe Institute of Technology

KASTEL

Verification of Red-Black Trees in KeY

A Case Study in Deductive Java Verification

Johanna Stuber | August 10, 2023



Red-Black Trees

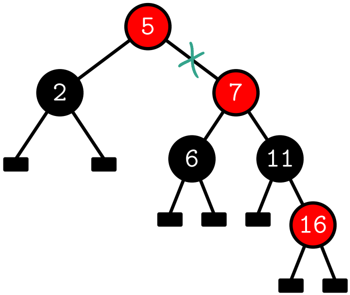
- self-balancing binary search trees
- nodes are either red or black
- JDK implementation: `java.util.TreeMap`
- used internally in `java.util.HashMap`

Why verify red-black trees with KeY?

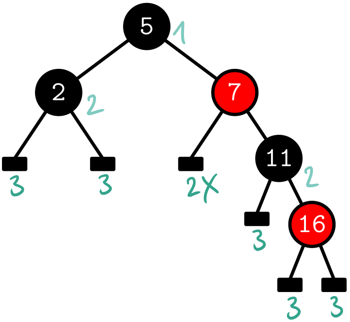
1. towards verification of real-world JDK code
2. contribution to a fully verified algorithmic “Basic Tool Box”
3. insights about framing of tree stuctures in KeY

Red-Black Trees
Properties

- *no double red*



- *black balanced*

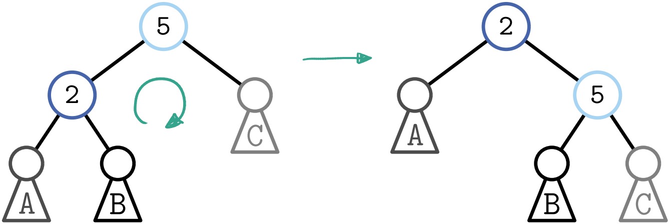


- consequence: tree height in $O(\log n)$ for n elements in the tree

Institute of Information Security and Dependability (KASTEL)

Red-Black Trees
Insertion

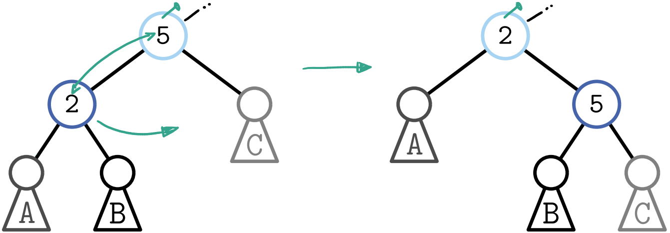
1. insert a red node with the new key “naively”
2. restore red-black properties through
   a. recolouring
   b. tree rotations



- still in $O(\log n)$ for n elements in the tree

Design Decisions

- rotations preserve root node
- no reference to parent node
- recursive implementation of `add()`



Institute of Information Security and Dependability (KASTEL)

Existing Work

- VerCors specification and verification (Armborst and Huisman 2021)
  - ✓ adaption of specifications regarding red black properties
- VerifyThis 2012: Deletion in a Tree (Bruns, Mostowski, and Ulbrich 2015)
  - ✓ starting point for representation of basic tree structure
- attempt in KeY (Bruns 2011)
  - ✗ completely different implementation decisions

Usage of Model Methods
Tree Structure

- ```
  \locset footprint() {
      return this.*
          ∪ left == null ? ∅ : left.footprint()
          ∪ right == null ? ∅ : right.footprint();
  }
  ```
- ```
  \intset treeSet() {
      return this.key
          ∪ left == null ? ∅ : left.treeSet()
          ∪ right == null ? ∅ : right.treeSet();
  }
  ```

Usage of Model Methods

Red Black Properties

- ```
  boolean blackBalanced() {
      return blackHeight(left) == blackHeight(right)
          && (left != null ==> left.blackBalanced())
          && (right != null ==> right.blackBalanced());
  }
  ```
- `static int blackHeight(nullable Tree t)`
- `boolean noDoubleRed()`
- ```
  boolean doubleRedTop()
  boolean doubleRedLeft()
  boolean doubleRedRight()
  ```

Object Invariants

Karlsruhe Institute of Technology

- `instance invariant`[1]

```
0 < var && left.var < var && right.var < var
&& left != right
&& \disjoint(this.*, right.footprint())
&& \disjoint(this.*, left.footprint())
&& \disjoint(left.footprint(), right.footprint())
&& ∀ k. k < key ==> k ∉ right.treeSet()
&& ∀ k. k > key ==> k ∉ left.treeSet()
&& \invariant_for(left) && \invariant_for(right)
```

- `validRBSubtree()`, `validRBSubtreeExceptRedTop()`

[1] all checks for `left == null` and `right == null` omitted

Simplified Contracts

Karlsruhe Institute of Technology

- ```
  /*@ normal_behaviour
    @ ensures key ∈ treeSet() <==> \result == true;
    @*/
  boolean contains(int key)
  ```

- ```
  /*@ normal_behaviour
    @ requires validRBTree();
    @ ensures validRBTree();
    @ ensures treeSet() == \old(treeSet()) ∪ {key};
    @*/
  void add(int key)
  ```

Assertions

Karlsruhe Institute of Technology

- framing: show that "untouched" parts of the tree haven't changed

```
right.add(key);
//@ assert left.footprint() == \old(left.footprint());
//@ assert left.treeSet() == \old(left.treeSet());
//@ assert left.blackBalanced() == \old(left.blackBalanced());
//@ assert ...
```

- additional assertions

```
//@ assert treeSet() == \old(treeSet()) ∪ {key};
//@ assert validRBSubtreeExceptRedTop();
//@ assert ...
```

Verification Challenges

Karlsruhe Institute of Technology

- prove “simple” statements over sets

- expand the right definition at the right time
- use the right proof strategy settings
- remember the above for later iterations of the proof
- prove some goals “analogously” to others

JML Scripts

```
private void rightRotate() {
    right.left = left.right; ...
    /*@ assert right_inv:   \invariant_for(right) \by {
            rule "recall_right_not_null";
            expand on="self.<inv>";
            assert "self.right.footprint() != empty" \by { ... }
            auto classAxioms=false steps=5000; } @*/
}
```

- written directly after assertion in the code
- help coping with the problems mentioned above
- assertion labels provide access to previous assertions

Statistics

Lines of . . .

	code	spec	#asserts	script	JML
model methods etc.	-	155	-	-	155
`contains()`	11	11	6	2	17
`add()`	9	11	1	2	16
`addRight()`	21	13	52	222	373
`addLeft()` (estimated)	21	13	52	222	373
`rightRotate()`	15	17	48	324	450
`leftRotate()` (estimated)	15	17	48	324	450
`recolour()`	5	13	29	120	204
`setHeight()`	0	38	21	0	65
other	7	17	0	0	17
total (estimated)	108	305	157	670	1,374

Statistics
Proof Statistics

	rule applications	manual	scripted
`contains()`	8,432	0	completely
`addRight()`	83,506	362	partly (most of framing)
`rightRotate()`	33,819	57	all but a few goals 8 min execution
`recolour()`	12,148	31	all but "preparations"
`setHeight()`	23,540	40	nothing

Statistics
Framing vs. Red-Black Trees

Karlsruhe Institute of Technology

- lines of assertions + scripts by purpose:

	framing	rb trees
`rightRotate()`	248	188
`recolour()`	137	54
`setHeight()` (#asserts)	20	1

- gut feeling: ¾ framing, ¼ red-black trees

Desirable Features for KeY

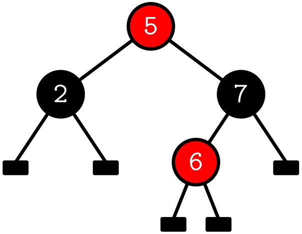
Karlsruhe Institute of Technology

- JML scripts + assertion labels
  - script generator
- better handling of sets
- more transparent proof strategy settings
- interactive proof loader
- (better) proof caching

- enhanced approach to framing – dynamic separation logic?

Conclusion

- successful proof of contains and add methods for red-black trees
- dynamic frames + tree structures are a lot of work
- extensively used:
  - model methods
  - assertions + JML scripts
  - proof strategy settings



References I

- Lukas Armborst and Marieke Huisman. “Permission-Based Verification of Red-Black Trees and Their Merging”. In: *2021 IEEE/ACM 9th International Conference on Formal Methods in Software Engineering (FormaliSE)*. IEEE. 2021, pp. 111–123.
- Daniel Bruns. “Specification of red-black trees: Showcasing dynamic frames, model fields and sequences”. In: *10th KeY Symposium, Nijmegen, the Netherlands*. 2011, p. 296.
- Daniel Bruns, Wojciech Mostowski, and Mattias Ulbrich. “Implementation-level verification of algorithms with KeY”. In: *International journal on software tools for technology transfer* 17 (2015), pp. 729–744.