3.3 **Balanced Search Trees**

- 2–3 search trees
- red–black BSTs (representation)
- red–black BSTs (operations)
- context

https://algs4.cs.princeton.edu
## Symbol table review

<table>
<thead>
<tr>
<th>implementation</th>
<th>guarantee</th>
<th>ordered ops?</th>
<th>key interface</th>
<th>emoji</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
<td>delete</td>
<td></td>
</tr>
<tr>
<td>sequential search (unordered list)</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>equals()</td>
</tr>
<tr>
<td>binary search (sorted array)</td>
<td>$\log n$</td>
<td>$n$</td>
<td>$n$</td>
<td>✓</td>
</tr>
<tr>
<td>BST</td>
<td><img src="n" alt="circle" /></td>
<td><img src="n" alt="circle" /></td>
<td>$n$</td>
<td>✓</td>
</tr>
<tr>
<td>goal</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Challenge.**  $\Theta(\log n)$ time in worst case.

**This lecture.**  2–3 trees and left-leaning **red-black** BSTs.

---

*Optimized for teaching and coding (introduced in COS 226)*

*Co-invented by Bob Sedgewick in the 1970s*
3.3 **BALANCED SEARCH TREES**

- 2–3 search trees
- red–black BSTs (representation)
- red–black BSTs (operations)
- context
2–3 tree

Allow 1 or 2 keys per node.
- 2-node: one key, two children.
- 3-node: two keys, three children.

Symmetric order. Inorder traversal yields keys in ascending order.
Perfect balance. Every path from the root to a null link has the same length.
2–3 tree demo

Search.

- Compare search key against key(s) in node.
- Find interval containing search key.
- Follow associated link (recursively).

search for H
2–3 tree: insertion

Insertion into a 2–node at bottom.
- Add new key to 2–node to create a 3–node.
2–3 tree: insertion

Insertion into a 3-node at bottom.

- Add new key to 3-node to create temporary 4-node.
- Move middle key in 4-node into parent.
- Repeat up the tree, as necessary.
- If you reach the root and it’s a 4-node, split it into three 2-nodes.

Insert $Z$
Suppose that you insert P into the following 2–3 tree. What will be the root of the resulting 2–3 tree?

A. E
B. ER
C. M
D. P
E. R
Balanced search trees: quiz 2

What is the **maximum** height of a 2–3 tree containing \( n \) keys?

A. \( \sim \log_3 n \)

B. \( \sim \log_2 n \)

C. \( \sim 2 \log_2 n \)

D. \( \sim n \)
2–3 tree: performance

Perfect balance. Every path from the root to a null link has the same length.

Key property. The height of a 2–3 tree containing $n$ keys is $\Theta(\log n)$.

- Min: $\sim \log_3 n \approx 0.631 \log_2 n$. [all 3-nodes]
- Max: $\sim \log_2 n$. [all 2-nodes]
- Between 18 and 30 for a billion keys.

Bottom line. Search and insert take $\Theta(\log n)$ time in the worst case.
## ST implementations: summary

<table>
<thead>
<tr>
<th>implementation</th>
<th>guarantee</th>
<th>ordered ops?</th>
<th>key interface</th>
<th>emoji</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
<td>delete</td>
<td></td>
</tr>
<tr>
<td>sequential search</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>equals()</td>
</tr>
<tr>
<td>(unordered list)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binary search</td>
<td>log $n$</td>
<td>$n$</td>
<td>$n$</td>
<td>✓</td>
</tr>
<tr>
<td>(sorted array)</td>
<td></td>
<td></td>
<td></td>
<td>compareTo()</td>
</tr>
<tr>
<td>BST</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>compareTo()</td>
</tr>
<tr>
<td>2-3 trees</td>
<td>log $n$</td>
<td>log $n$</td>
<td>log $n$</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>compareTo()</td>
</tr>
</tbody>
</table>

but hidden constant $c$ is large (depends upon implementation)
2–3 tree: implementation?

Direct implementation is complicated, because:

- Maintaining multiple node types is cumbersome.
- Might need two compares to move one level down tree.
- Need to move back up the tree to split 4-nodes.
- Large number of cases for splitting.

```
public void put(Key key, Value val)
{
    Node x = root;
    while (x.getTheCorrectChild(key) != null)
    {
        x = x.getTheCorrectChildKey();
        if (x.is4Node()) x.split();
    }
    if (x.is2Node()) x.make3Node(key, val);
    else if (x.is3Node()) x.make4Node(key, val);
}
```

Bottom line. Could do it (see COS 326!), but there’s a better way.
3.3 **Balanced Search Trees**

- 2–3 search trees
- *red–black BSTs (representation)*
- *red–black BSTs (operations)*
- context
How to implement 2–3 trees as binary search trees?

**Challenge.** How to represent a 3 node?

**Approach 1.** Two BST nodes.
- No way to tell a 3–node from two 2–nodes.
- Can’t (uniquely) map from BST back to 2–3 tree.

**Approach 2.** Two BST nodes, plus red “glue” node.
- Wastes space for extra node.
- Messy code.

**Approach 3.** Two BST nodes, with red “glue” link.
- Widely used in practice.
- Arbitrary restriction: red links lean left.
1. Represent 2–3 tree as a BST.
2. Use “internal” left–leaning red links as “glue” for 3–nodes.
Key property. 1–1 correspondence between 2–3 trees and LLRB trees.
Which LLRB tree corresponds to the following 2–3 tree?

C. Both A and B.

D. Neither A nor B.
An equivalent definition of LLRB trees (without reference to 2–3 trees)

**Def.** A red–black BST is a BST such that:

- No node has two red links connected to it.
- Red links lean left.
- Every path from root to null link has the same number of black links.
Red–black BST representation

Each node is pointed to by precisely one link (from its parent) ⇒ can encode color of links in child nodes.

```java
private static final boolean RED   = true;
private static final boolean BLACK = false;

private class Node
{
    private Key key;
    private Value val;
    private Node left, right;
    private boolean color;
}

private boolean isRed(Node x)
{
    if (x == null) return false;
    return x.color == RED;
}
```
Review: the road to LLRB trees

BSTs (can get imbalanced)

2–3 trees (balanced but awkward to implement)

3-nodes “glued” together with red links

how we draw LLRB trees (color in links)

how we implement LLRB trees (color in nodes)
3.3 Balanced Search Trees

- 2–3 search trees
- red–black BSTs (representation)
- red–black BSTs (operations)
- context

https://algs4.cs.princeton.edu
Search in a red–black BST

**Observation.** Red–black BSTs are BSTs $\Rightarrow$ search is the same as for BSTs (ignore color).

```
public Value get(Key key) {
    Node x = root;
    while (x != null) {
        int cmp = key.compareTo(x.key);
        if (cmp < 0) x = x.left;
        else if (cmp > 0) x = x.right;
        else return x.val;
    }
    return null;
}
```

but runs faster (because of better balance)

**Remark.** Many other operations (iteration, floor, rank, selection) are also identical.
Basic strategy. Maintain 1–1 correspondence with 2–3 trees.

During internal operations, maintain:

- Symmetric order.
- Perfect black balance.
- [ but not necessarily color invariants ]

Example violations of color invariants:

- Right-leaning red link
- Two red children (a temporary 4-node)
- Left-left red (a temporary 4-node)
- Left-right red (a temporary 4-node)

To restore color invariants: perform color flips and rotations.
Elementary red–black BST operations

**Color flip.** Recolor to split a (temporary) 4–node.

flip colors (before)

![Diagram of a tree with nodes A, E, S, and colors]

```java
private void flipColors(Node h) {
    assert !isRed(h);
    assert isRed(h.left);
    assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```

**Invariants.** Maintains symmetric order and perfect black balance.
Elementary red–black BST operations

**Color flip.** Recolor to split a (temporary) 4-node.

Invariants. Maintains symmetric order and perfect black balance.

```java
private void flipColors(Node h) {
    assert !isRed(h);
    assert isRed(h.left);
    assert isRed(h.right);
    h.color = RED;
    h.left.color = BLACK;
    h.right.color = BLACK;
}
```
Elementary red–black BST operations

**Left rotation.** Orient a (temporarily) right-leaning red link to lean left.

![Diagram of left rotation]

**Invariants.** Maintains symmetric order and perfect black balance.
**Elementary red–black BST operations**

**Left rotation.** Orient a (temporarily) right–leaning red link to lean left.

```
private Node rotateLeft(Node h) {
    assert !isRed(h.left);
    assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

**Invariants.** Maintains symmetric order and perfect black balance.
Elementary red–black BST operations

**Right rotation.** Orient a left-leaning red link to (temporarily) lean right.

```
private Node rotateRight(Node h)
{
    assert isRed(h.left);
    assert !isRed(h.right);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

**Invariants.** Maintains symmetric order and perfect black balance.
**Elementary red–black BST operations**

**Right rotation.** Orient a left-leaning red link to (temporarily) lean right.

```java
private Node rotateRight(Node h) {
    assert isRed(h.left);
    assert !isRed(h.right);
    Node x = h.left;
    h.left = x.right;
    x.right = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```

**Invariants.** Maintains symmetric order and perfect black balance.
Which sequence of elementary operations transforms the red–black BST at left to the one at right?

A. Color flip E; left rotate R.
B. Color flip R; left rotate E.
C. Color flip R; left rotate R.
D. Color flip R; right rotate E.
Insertion into a LLRB tree

- Do standard BST insert and color new link red.
- Repeat up the tree until color invariants restored:
  - two left red links in a row? ⇒ rotate right
  - left and right links both red? ⇒ color flip
  - only right link red? ⇒ rotate left

To preserve symmetric order and perfect black balance.
Insertion into a LLRB tree

- Do standard BST insert and color new link red.
- Repeat up the tree until color invariants restored:
  - two left red links in a row? \(\Rightarrow\) rotate right
  - left and right links both red? \(\Rightarrow\) color flip
  - only right link red? \(\Rightarrow\) rotate left

---

**Diagram**

- Inserting \(P\)
- Passing a red link up the tree
  - two lefts in a row so rotate right
  - both children red so flip colors
- Right link red so rotate left
- Two lefts in a row so rotate right
- Both children red so flip colors
- Red-black BST
insert S E A R C H X M P L
Insertion into a LLRB tree: Java implementation

- Do standard BST insert and color new link red.
- Repeat up the tree until color invariants restored:
  - only right link red? ⇒ rotate left
  - two left red links in a row? ⇒ rotate right
  - left and right links both red? ⇒ color flip

```java
private Node put(Node h, Key key, Value val) {
    if (h == null) return new Node(key, val, RED);

    int cmp = key.compareTo(h.key);
    if (cmp < 0) h.left = put(h.left, key, val);
    else if (cmp > 0) h.right = put(h.right, key, val);
    else h.val = val;

    if (isRed(h.right) && !isRed(h.left)) h = rotateLeft(h);
    if (isRed(h.left) && isRed(h.left.left)) h = rotateRight(h);
    if (isRed(h.left) && isRed(h.right)) flipColors(h);

    return h;
}
```

only a few extra lines of code provides near-perfect balance
Insertion into a LLRB tree: visualization

n = 255
height = 9
average depth = 6.3

255 insertions in random order
Insertion into a LLRB tree: visualization

\[
\begin{align*}
n &= 255 \\
\text{height} &= 7 \\
\text{average depth} &= 6.0
\end{align*}
\]

255 insertions in ascending order
Insertion into a LLRB tree: visualization

n = 254
height = 13
average depth = 6.5

254 insertions in descending order
Proposition. Height of LLRB tree is $\leq 2\log_2 n$.

Pf.

- Black height $=$ height of corresponding 2–3 tree $\leq \log_2 n$.
- Never two red links in a row.
  $\Rightarrow$ height of LLRB tree $\leq (2 \times \text{black height}) + 1$
  $\leq 2\log_2 n + 1$.

- [A slightly more careful argument shows height $\leq 2\log_2 n$.]
## ST implementations: summary

<table>
<thead>
<tr>
<th>implementation</th>
<th>guarantee</th>
<th>ordered ops?</th>
<th>key interface</th>
<th>emoji</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>search</td>
<td>insert</td>
<td>delete</td>
<td></td>
</tr>
<tr>
<td>sequential search (unordered list)</td>
<td>(n)</td>
<td>(n)</td>
<td>(n)</td>
<td>equals()</td>
</tr>
<tr>
<td>binary search (sorted array)</td>
<td>(\log n)</td>
<td>(n)</td>
<td>(n)</td>
<td>✔ compareTo()</td>
</tr>
<tr>
<td>BST</td>
<td>(n)</td>
<td>(n)</td>
<td>(n)</td>
<td>✔ compareTo()</td>
</tr>
<tr>
<td>2–3 trees</td>
<td>(\log n)</td>
<td>(\log n)</td>
<td>(\log n)</td>
<td>✔ compareTo()</td>
</tr>
<tr>
<td>red-black BSTs</td>
<td>(\log n)</td>
<td>(\log n)</td>
<td>(\log n)</td>
<td>✔ compareTo()</td>
</tr>
</tbody>
</table>

Hidden constant \(c\) is small (\(\leq 2 \log_2 n\) compares)
3.3 Balanced Search Trees

- 2–3 search trees
- red–black BSTs (representation)
- red–black BSTs (operations)
- context
Balanced search trees in the wild

Red-black BSTs are widely used as system symbol tables.

- Java: java.util.TreeMap, java.util.TreeSet.
- C++ STL: map, multimap, multiset.
- Linux kernel: CFQ I/O scheduler, VMAs, linux/rbtree.h.

Other balanced BSTs. AVL trees, splay trees, randomized BSTs, rank-balanced BSTs, ....

B-trees (and cousins) are widely used for file systems and databases.
Industry story 1: red–black BSTs

Telephone company contracted with database provider to build a real-time database to store customer information.

Database implementation.
- Red–black BST.
- Exceeding height limit of 80 triggered error–recovery process.

Database crashed.
- Main cause = height bound exceeded!
- Telephone company sues database provider.
- Legal testimony:

  “If implemented properly, the height of a red–black BST with $n$ keys is at most $2 \times \log_2 n$.” — expert witness
Industry story 2: red–black BSTs

Celestine Omin@
cyberomin

I was just asked to balance a Binary Search Tree by JFK's airport immigration. Welcome to America.

8:26 AM · 26 Feb 2017 from Manhattan, NY

8,025 Retweets 7,087 Likes

Celestine Omin@
cyberomin · 26 Feb 2017

I was too tired to even think of a BST solution. I have been travelling for 23hrs. But I was also asked about 10 CS questions.

8 164 244

Celestine Omin@
cyberomin · 26 Feb 2017

Sad thing is, if I didn't give the Wikipedia definition for these questions, it was considered a wrong answer.

19 324 703

Simon Sharwood@s sharwood · 26 Feb 2017

Replying to @cyberomin

Seriously? am reporter for @theregister and would love to know more about your experience

22 171

https://twitter.com/cyberomin/status/835888786462625792
The red–black tree song (by Sean Sandys)

I see a brand new node,
I want to paint it black.
We need a balanced tree,
we've got to paint it black.

No time for AVL trees,
We must paint it black.

I want to
find my key in
log n time—that's all.
Rotating subtrees 'round,
sure can be a ball.

I see a brand new node,
I want to paint it black.
Can't have a lot of red nodes,
we must paint them black.

Unfortunately, coding them
can be a $!#%..
If we had half a brain,
to splay trees we would switch.
And if they're still confusing,
you should have no fear.
Because outside this class,
of them you'll never hear.

I wanna see it,
painted, painted black.
Black is nice.
I wanna see the nodes painted black.
Black is nice.
I wanna see 'em
painted, painted, painted, painted black.

Mm mm mm mm mm mm mm.
Mm mm mm mm mm-mm.
Mm mm mm mm mm mm mm.
Mm mm mm mm mm-mm.

performed by U. Washington CSE Band '02