3.3 Balanced Search Trees

- 2-3 search trees
- red-black BSTs
- B-trees

Symbol table review

<table>
<thead>
<tr>
<th>implementation</th>
<th>worst-case cost (after N inserts)</th>
<th>average case (after N random inserts)</th>
<th>ordered iteration?</th>
<th>key interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>N</td>
<td>N</td>
<td>N/2</td>
<td>N/2</td>
</tr>
<tr>
<td>insert</td>
<td>N</td>
<td>N</td>
<td>N/2</td>
<td>N/2</td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search hit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average case</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search hit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ordered iteration?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>key interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


This lecture. 2-3 trees, left-leaning red-black BSTs, B-trees.

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 root to null link has same length.
2-3 tree demo

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

**search for H**

![Search for H diagram]

2-3 tree construction demo

**insert S**

![Insert S diagram]

**2-3 tree construction demo**

**Insertion into a 2-3 tree**

**Insertion into a 2-node at bottom.**
- Add new key to 2-node to create a 3-node.

**insert G**

![Insert G diagram]
Insertion into a 2-3 tree

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.

Insertion into a 2-3 tree

Local transformations in a 2-3 tree

Splitting a 4-node is a local transformation: constant number of operations.

Global properties in a 2-3 tree

Invariants. Maintains symmetric order and perfect balance.
Pf. Each transformation maintains symmetric order and perfect balance.

2-3 tree: performance

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

Tree height.
- Worst case:
- Best case:
2-3 tree: performance

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

**Tree height.**
- **Worst case:** $\lg N$. [all 2-nodes]
- **Best case:** $\log_3 N \approx 0.631 \lg N$. [all 3-nodes]
- Between 12 and 20 for a million nodes.
- Between 18 and 30 for a billion nodes.

Guaranteed **logarithmic** performance for search and insert.

2-3 tree: implementation?

Direct implementation is complicated, because:
- Maintaining multiple node types is cumbersome.
- Need multiple compares to move down tree.
- Need to move back up the tree to split 4-nodes.
- Large number of cases for splitting.

**fantasy code**

```java
public void put(Key key, Value val) {
    Node x = root;
    while (x.getTheCorrectChild(key) != null) {
        x = x.getTheCorrectChild(key);
        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, but there's a better way.

### 3.3 Balanced Search Trees

- 2-3 search trees
- red-black BSTs
- B-trees

---

**ST implementations: summary**

<table>
<thead>
<tr>
<th>Implementation</th>
<th>worst-case cost (after $N$ inserts)</th>
<th>average case (after $N$ random inserts)</th>
<th>ordered iteration?</th>
<th>key interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential search (unordered list)</td>
<td>$N$</td>
<td>$N$</td>
<td>$N/2$</td>
<td>$N$</td>
</tr>
<tr>
<td>binary search (ordered array)</td>
<td>$\lg N$</td>
<td>$N$</td>
<td>$N$</td>
<td>$\lg N$</td>
</tr>
<tr>
<td>BST</td>
<td>$N$</td>
<td>$N$</td>
<td>$N$</td>
<td>$1.39 \lg N$</td>
</tr>
<tr>
<td>2-3 tree</td>
<td>$c \lg N$</td>
<td>$c \lg N$</td>
<td>$c \lg N$</td>
<td>$c \lg N$</td>
</tr>
</tbody>
</table>

Constants depend upon implementation
Left-leaning red-black BSTs (Guibas-Sedgewick 1979 and Sedgewick 2007)

1. Represent 2–3 tree as a BST.
2. Use “internal” left-leaning links as “glue” for 3–nodes.

An equivalent definition

A BST such that:
1. No node has two red links connected to it.
2. Every path from root to null link has the same number of black links.
3. Red links lean left.

Search implementation for red-black BSTs

Observation. Search is the same as for elementary BST (ignore color).

public Val 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 if (cmp == 0) return x.val;
    }
    return null;
}

Remark. Most other ops (e.g., floor, iteration, selection) are also identical.
Red-black BST representation

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

Elementary red-black BST operations

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

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

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.

```
rotate S right
(after)
```

```
E

\[ \text{less than E} \]

\[ \text{between E and S} \]

\[ \text{greater than S} \]

S

\[ 0 \]

h
```

**Invariants.** Maintains symmetric order and perfect black balance.

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

Elementary red-black BST operations

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

```
flip colors
(before)
```

```
E

\[ \text{less than A} \]

\[ \text{between A and E} \]

\[ \text{between E and S} \]

\[ \text{greater than S} \]

A

S

h
```

```
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

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

```
flip colors
(after)
```

```
E

\[ \text{less than A} \]

\[ \text{between A and E} \]

\[ \text{between E and S} \]

\[ \text{greater than S} \]

A

S

h
```

**Invariants.** Maintains symmetric order and perfect black balance.

```
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

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

```
flip colors
(after)
```

```
E

\[ \text{less than A} \]

\[ \text{between A and E} \]

\[ \text{between E and S} \]

\[ \text{greater than S} \]

A

S

h
```

**Invariants.** Maintains symmetric order and perfect black balance.

```
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;
}
```

Insertion in a LLRB tree: overview

**Basic strategy.** Maintain 1-1 correspondence with 2-3 trees by applying elementary red-black BST operations.
### Warmup 1. Insert into a tree with exactly 1 node.

**Case 1.** Insert into a 2-node at the bottom.
- Do standard BST insert; color new link red.
- If new red link is a right link, rotate left.

**Case 2.** Insert into a 3-node at the bottom.
- Do standard BST insert; color new link red.
- Flip colors to pass red link up one level.
- Rotate to make lean left (if needed).

---

**Example Insertion in a LLRB Tree**

1. **Insertion in a LLRB tree**

2. **Insertion in a LLRB tree**
**Insertion in a LLRB tree: passing red links up the tree**

**Case 2.** Insert into a 3-node at the bottom.
- Do standard BST insert; color new link red.
- Rotate to balance the 4-node (if needed).
- Flip colors to pass red link up one level.
- Rotate to make lean left (if needed).
- Repeat case 1 or case 2 up the tree (if needed).

**Red-black BST construction demo**

```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 if (cmp == 0) 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;
}
```

**Red-black BST construction demo**

- Right child red, left child black: rotate left.
- Left child, left-left grandchild red: rotate right.
- Both children red: flip colors.
Balance in LLRB trees

**Proposition.** Height of tree is \( \leq 2 \lg N \) in the worst case.

**Pf.**
- Every path from root to null link has same number of black links.
- Never two red links in-a-row.

**Property.** Height of tree is \( \sim 1.00 \lg N \) in typical applications.
### War story: red-black BSTs

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

**Database implementation.**
- Red-black BST search and insert; Hibbard deletion.
- Exceeding height limit of 80 triggered error-recovery process.

**Extended telephone service outage.**
- Main cause = height bounded 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 lg N." — expert witness

---

### 3.3 Balanced Search Trees

- 2-3 search trees
- red-black BSTs
- B-trees

---

### Xerox PARC innovations. [1970s]

- Alto.
- GUI.
- Ethernet.
- Smalltalk.
- InterPress.
- Laser printing.
- Bitmapped display.
- WYSIWYG text editor.
- ...

---

### Algorithims

- 2-3 search trees
- red-black BSTs
- B-trees

---
**File system model**

**Page.** Contiguous block of data (e.g., a file or 4,096-byte chunk).

**Probe.** First access to a page (e.g., from disk to memory).

**Property.** Time required for a probe is much larger than time to access data within a page.

**Cost model.** Number of probes.

**Goal.** Access data using minimum number of probes.

---

**Searching in a B-tree**

- Start at root.
- Find interval for search key and take corresponding link.
- Search terminates in external node.

---

**B-trees (Bayer-McCreight, 1972)**

**B-tree.** Generalize 2-3 trees by allowing up to $M - 1$ key-link pairs per node.

- At least 2 key-link pairs at root.
- At least $M / 2$ key-link pairs in other nodes.
- External nodes contain client keys.
- Internal nodes contain copies of keys to guide search.

---

**Insertion in a B-tree**

- Search for new key.
- Insert at bottom.
- Split nodes with $M$ key-link pairs on the way up the tree.
**Proposition.** A search or an insertion in a B-tree of order $M$ with $N$ keys requires between $\log_{M-1} N$ and $\log_{M/2} N$ probes.

**Pf.** All internal nodes (besides root) have between $M/2$ and $M-1$ links.

In practice. Number of probes is at most 4.

Optimization. Always keep root page in memory.

---

**Building a large B tree**

Building a large B tree full page splits into two half-full pages then a new key is added to one of them.

---

**Balanced trees in the wild**

Red-black trees are widely used as system symbol tables.
- Java: `java.util.TreeMap, java.util.TreeSet`
- C++ STL: `map, multimap, multiset`
- Linux kernel: `completely fair scheduler, linux/rbtree.h`
- Emacs: `conservative stack scanning`

B-tree variants. B+ tree, B*tree, B# tree, ...

B-trees (and variants) are widely used for file systems and databases.
- Windows: `HPFS`
- Mac: `HFS, HFS+`
- Linux: `ReiserFS, XFS, Ext3FS, JFS`
- Databases: `ORACLE, DB2, INGRES, SQL, PostgreSQL`

---

**Red-black BSTs in the wild**

Common sense. Sixth sense. Together they’re the FBI’s newest team.
Antonio is at the computer as Jess explains herself to Nicole and Follow. The conference table is adorned with open reference books, tourist guides, maps and reams of printouts.

**JESS**
It was the red door again.

**FOLLOW**
I thought the red door was the storage container.

**JESS**
But it wasn’t red anymore. It was black.

**ANTONIO**
So red turning to black means... what?

**FOLLOW**
What defines red and black ink?

**NICOLE**
Yes, I’m sure that’s what it is. But maybe we should come up with a couple other options, just in case.

Antonio refers to his computer screen, which is filled with mathematical equations.

**ANTONIO**
It could be an algorithm from a binary search tree. A red-black tree traverse every single path from a node to a descendant leaf with the same number of black nodes.

**JESS**
Does that help you with girls?