Lecture 28-29: Left--leaning Red-Black Trees

- Introduction
- Elementary red-black BST operations
- Insertion
- Mathematical analysis
- Historical context

Some slides adopted from Algorithms 4th Edition or COS226
Left-leaning red-black BSTs correspond 1-1 with 2-3 trees

- Start with standard BSTs which are made up of 2-nodes.
- Add extra information to encode 3-nodes. We will introduce two types of links.
- **Red links**: bind together two 2-nodes to represent a 3-node.
  - Specifically, 3-nodes are represented as two 2-nodes connected by a single red link that leans left (one of the 2-nodes is the left child of the other).
- **Black links**: bind together the 2-3 tree.
- Advantage: Can use BST code with minimal modification.
Left-leaning red-black BSTs correspond 1-1 with 2-3 trees
Definition

- A left-leaning red-black tree is a BST such that:
  - No node has two red links connected to it.
  - Red link leans left.
  - Every path from root to leaves has the same number of black links (perfect black balance).
Search

- Exactly the same as for elementary BSTs (we ignore the color).
- But runs faster because of better balance.

```java
public Value get(Key key) {
    if (key == null) throw new IllegalArgumentException("argument to get() is null");
    return get(root, key);
}
```

```
// value associated with the given key in subtree rooted at x; null if no such key
private Value get(Node x, Key key) {
    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;
}
```

- Operations such as floor, iteration, rank, selection are also identical.
Representation

- Each node is pointed to by one node, its parent. We can use this to encode the color of the links in nodes.
- True if the link from the parent is red and false if it is black. Null links are black.

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

private Node root; // root of the BST

// BST helper node data type
private class Node {
    private Key key; // key
    private Value val; // associated data
    private Node left, right; // links to left and right subtrees
    private boolean color; // color of parent link
    private int size; // subtree count

    private boolean isRed(Node x) {
        if (x == null) return false;
        return x.color == RED;
    }
}
Story so far

- BSTs can get imbalanced and long.
- 2-3 trees are balanced but cumbersome to code.
- Imagine 3-nodes held together by internal glue links shown in red.
- Draw links by giving them red or black color.
- Represent them in memory by storing the color of the link coming from the parent as the color of the child node.
Practice Time

- Which of the following are legal LLRB trees?
Answer

- Which of the following are legal LLRB trees?
- iii and iv
  - i is not balanced and ii is also not in symmetrical order
Lecture 28-29: Left-leaning Red-Black Trees

- Introduction
- Elementary red-black BST operations
- Insertion
- Mathematical analysis
- Historical context
**Left rotation**: Orient a (temporarily) right-leaning red link to lean left.
Right rotation: Orient a left-leaning red link to a (temporarily) lean right
**Color flip:** Recolor to split a (temporary) 4-node
Lecture 28-29: Left-leaning Red-Black Trees

- Introduction
- Elementary red-black BST operations
- Insertion
- Mathematical analysis
- Historical context
Basic strategy: Maintain 1-1 correspondence with 2-3 trees

- During internal operations, maintain:
  - symmetric order.
  - perfect black balance.
- But we might violate color invariants. For example:
  - Right-leaning red link.
  - Two red children (temporary 4-node).
  - Left-left red (temporary 4-node).
  - Left-right red (temporary 4-node).
- To restore color invariant we will be performing rotations and color flips.
Insertion into a LLRB

- Do standard BST insertion and color the new link red.
- Repeat until color invariants restored:
  - Both children red? Flip colors.
  - Right link red? Rotate left.
  - Two left reds in a row? Rotate right.
Red-black BST construction demo

red-black BST
Implementation

- Only three cases:
  - Right child red; left child black: rotate left.
  - Left child red; left-left grandchild red: rotate right.
  - Both children red: flip colors.

```java
private Node put(Node h, Key key, Value val) {
    if (h == null) return new Node(key, val, RED, 1);
    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;

    // fix-up any right-leaning links
    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);
    h.size = size(h.left) + size(h.right) + 1;

    return h;
}
```
Visualization of insertion into a LLRB tree

- 255 insertions in ascending order.
Visualization of insertion into a LLRB tree

- 255 insertions in descending order.
Visualization of insertion into a LLRB tree

- 255 insertions in random order.
Examples
Lecture 28-29: Left-leaning Red-Black Trees

- Introduction
- Elementary red-black BST operations
- Insertion
- Mathematical analysis
- Historical context
Balance in LLRB trees

- Height of LLRB trees is $\leq 2 \log n$ in the worst case.
- Worst case is a 2-3 tree that is all 2-nodes except that the left-most path is made up of 3-nodes.
- All ordered operations (min, max, floor, ceiling) etc. are also $O(\log n)$. 
## Summary for symbol table operations

<table>
<thead>
<tr>
<th></th>
<th>Worst case</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
<td>Search</td>
<td>Insert</td>
<td>Delete</td>
</tr>
<tr>
<td>Sequential search</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>$n/2$</td>
<td>$n$</td>
<td>$n/2$</td>
</tr>
<tr>
<td>(unordered)</td>
<td><strong>Blue</strong></td>
<td></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>$\log n$</td>
<td>$n/2$</td>
<td>$n/2$</td>
</tr>
<tr>
<td>(ordered array)</td>
<td><strong>Blue</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BST</td>
<td>$n$</td>
<td>$n$</td>
<td>$n$</td>
<td>1.39 $\log n$</td>
<td>1.39 $\log n$</td>
<td>?</td>
</tr>
<tr>
<td>2-3 search tree</td>
<td>$c \log n$</td>
<td>$c \log n$</td>
<td>$c \log n$</td>
<td>$c \log n$</td>
<td>$c \log n$</td>
<td>$c \log n$</td>
</tr>
<tr>
<td>Red-black BSTs</td>
<td>$2 \log n$</td>
<td>$2 \log n$</td>
<td>$2 \log n$</td>
<td>$1 \log n$</td>
<td>$1 \log n$</td>
<td>$1 \log n$</td>
</tr>
</tbody>
</table>
Lecture 28-29: Left-leaning Red-Black Trees

- Introduction
- Elementary red-black BST operations
- Insertion
- Mathematical analysis
- Historical context
Red-black trees


- Why red-black? Xerox PARC had a laser printer and red and black had the best contrast...

- Left-leaning red-black trees [Sedgewick, 2008]
  - Inspired by difficulties in proper implementation of RB BSTs.

- RB BSTs have been involved in lawsuit because of improper implementation.
Balanced trees in the wild

- Red-black trees are widely used as system symbol tables.
  - e.g., Java: `java.util.TreeMap` and `java.util.TreeSet`.
- Other balanced BSTs: AVL, splay, randomized.
- 2-3 search trees are a subset of b-trees.
  - See book for more.
- B-trees are widely used for file systems and databases.
Lecture 28-29: Left-leaning Red-Black Trees

- Introduction
- Elementary red-black BST operations
- Insertion
- Mathematical analysis
- Historical context
Readings:

- Textbook: Chapter 3.3 (Pages 432-447)
- Website:
  - [https://algs4.cs.princeton.edu/33balanced/](https://algs4.cs.princeton.edu/33balanced/)

Practice Problems:

- 3.3.9-3.3.22