Red & Black Tree

Anwar Mamat
BST

Balanced BST

Search: $O(\log n)$

Unbalanced BST

Search: $O(n)$
Balanced Binary Search Tree

• Red & Black Tree
• AVL Tree
• 2-3 Tree
• B-tree
Red & Black Tree

• A BST such that:
  • Tree edges have color: Red or Black
  • No node has two red edges connected to it.
  • Every path from root to null link has the same number of black links.
  • Red links lean left.
  • New node edge is Red
Search: 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;
}
```
Red-black BST representation

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

private class Node
{
    Key key;
    Value val;
    Node left, right;
    boolean color;  // color of parent link
}

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

null links are black
Elementary red-black BST operations

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

rotate E left (before)  
rotate E left (after)
Elementary red-black BST operations cont.

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

```java
private Node rotateLeft(Node h) {
    assert isRed(h.right);
    Node x = h.right;
    h.right = x.left;
    x.left = h;
    x.color = h.color;
    h.color = RED;
    return x;
}
```
Elementary red-black BST operations cont.

- Right rotation: Orient a left-leaning red link to (temporarily) lean right.
Elementary red-black BST operations cont.

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

```java
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 cont.

• Color flip.
Elementary red-black BST operations cont.

- Color flip.

```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;
}
```
Insertion in a LLRB tree

- Right child red, left child black: rotate left.
- Left child, left-left grandchild red: rotate right.
- Both children red: flip colors.
Insertion in a LLRB tree: Java implementation

```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;
}
```
R&B Example: Insertion

Insert: M, D
R&B Example: Insertion

Insert: M, X

Rotate Left
R&B Example: Insertion

Insert: M, X

Rotate Left
R&B Example: Insertion

Insert: M, D, X

Flip Color
R&B Example: Insertion

Insert: M, D, B

Rotate Right

Flip Color
R&B Example: Insertion

Insert: M, D, B

Flip Color
R&B Example: Insertion

M, D, B  Insert H
R&B Example: Insertion

M, D, B, H  Insert S

Color flip
R&B Example: Insertion

M, D, B, H  Insert S

Color flip

Rotate Left
R&B Example: Insertion

M, D, B, H  Insert S

Rotate Left

Diagram shows a binary search tree with nodes M, D, B, H, S. After inserting 'S', there is a need to rotate left to maintain the tree's balance.
R&B Example: Insertion

M, D, B,H,S  Insert E,K
R&B Example: Insertion

M, D, B,H,S  Insert E,K

Color flip

Rotate Left
R&B Example: Insertion

M, D, B,H,S  Insert E,K

Rotate left
R&B Example: Insertion
M, D, B, H, S  Insert E, K

Rotate Right
R&B Example: Insertion

M, D, B, H, S, E, K

Color flip