【DS】Day19

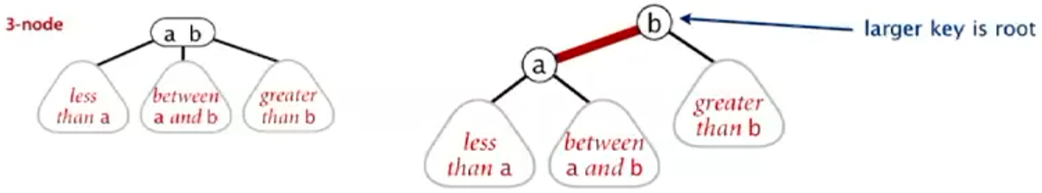
☰ Tags	
Date	@June 15, 2022
Summary	Red-Black Tree Insertion

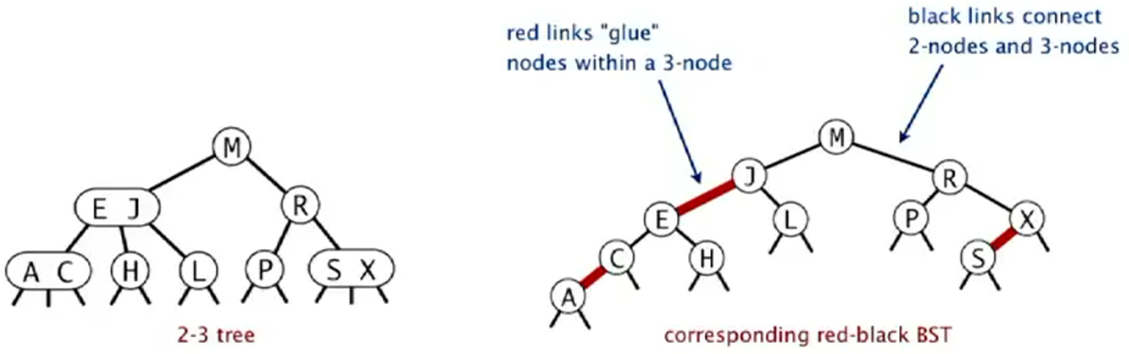
【Week 5】Balanced Search Tree

5.2 Red-Black Tree

Left-leaning red-black BSTs

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



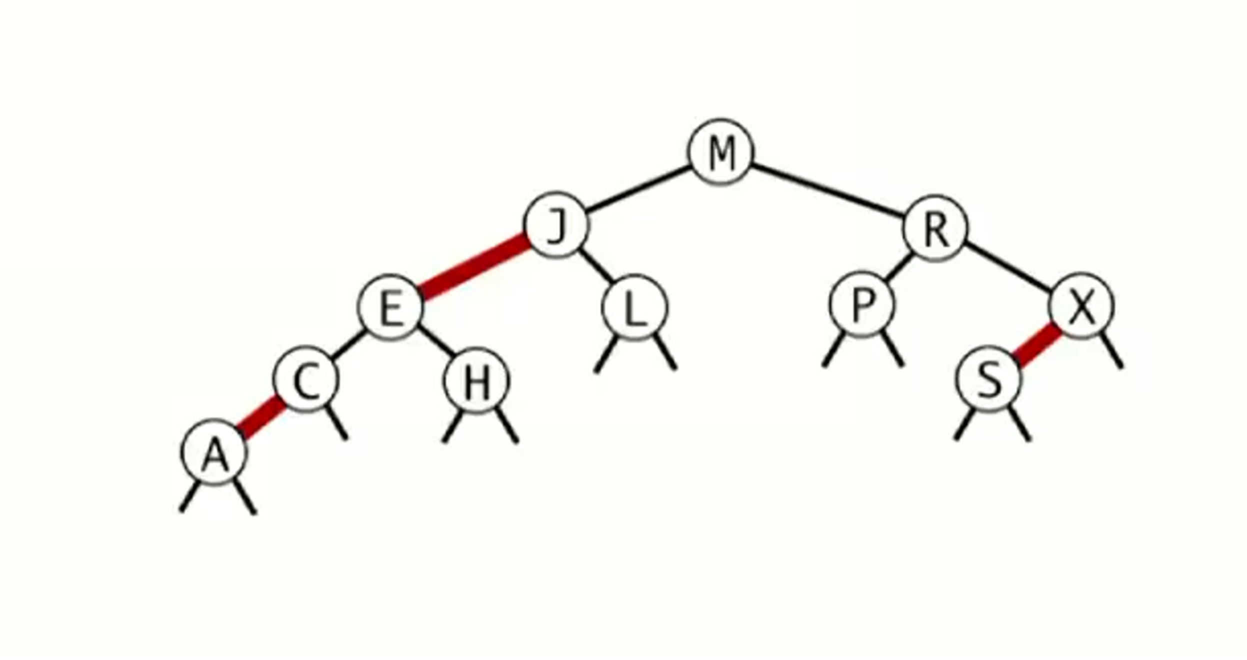

M
E J
R
A C
H
L
P
S X
2-3 tree
red links "glue" nodes within a 3-node
black links connect 2-nodes and 3-nodes
M
J
R
E
L
P
X
C
H
S
A
corresponding red-black BST

An equivalent definition would be:

A Binary Search Tree such that

- No nodes has two red links connected to it

- Every path from root to null link has the same number of black links
- Red links lean left.



Implementation

Search is the same for elementary BST(ignore color)

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

Better performance because of better balance in the tree.

Red-Black BST representation

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

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

private boolean isRed(Node node) {
  // Null links are black
  if (node == null) return BLACK;
  return node.color == RED;
}
```

Left-orientation: Orient a (temporarily) right-leaning red link to lean left

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

Right-orientation

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

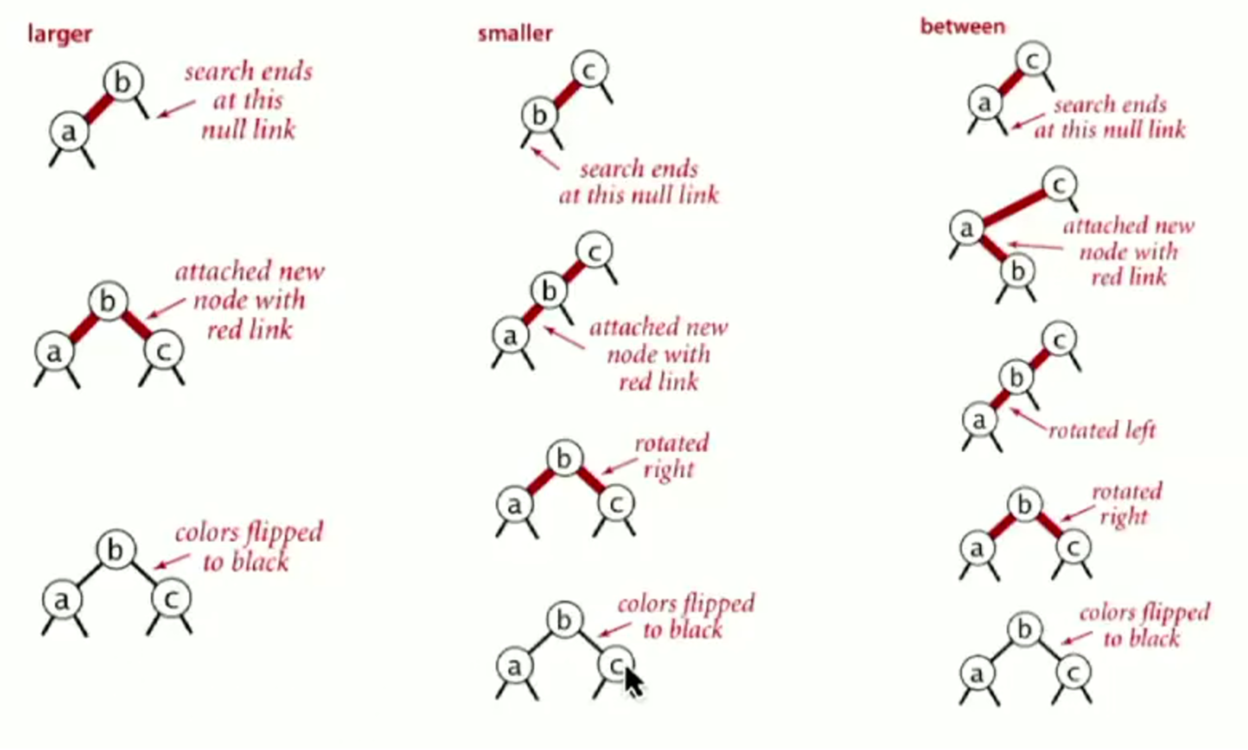
Color flip: Recolor to split a (temporary) 4-node

```
private void flipColors(Node node) {
  assert !isRed(node);
  assert isRed(node.right);
  assert isRed(node.left);

  node.color = RED;
  node.left.color = BLACK;
  node.right.color = BLACK;
}
```

Insertion

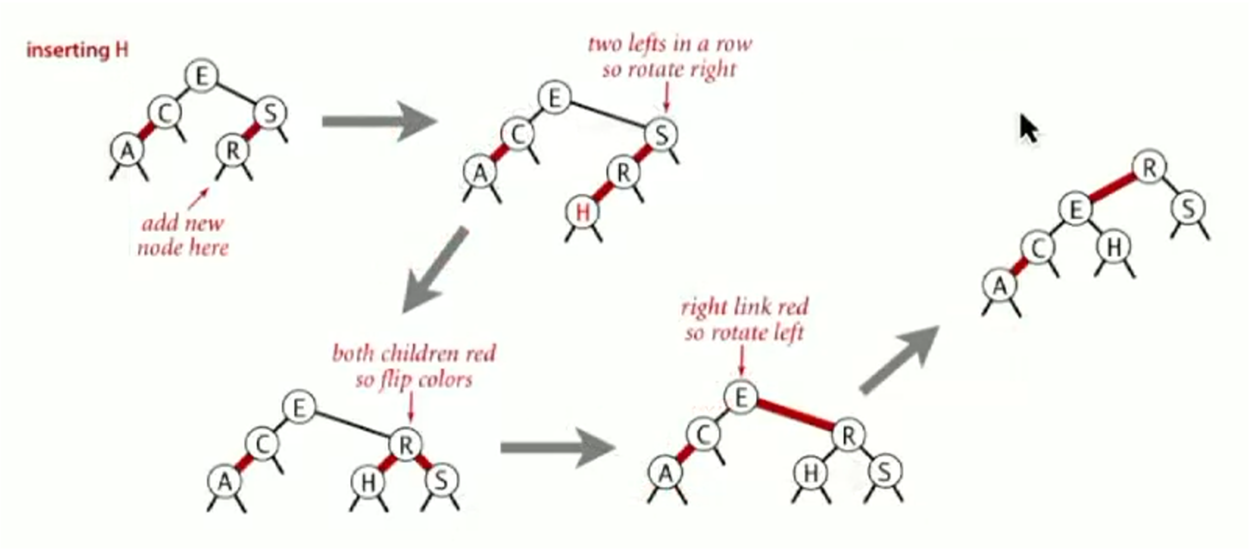
Insert into a tree with exactly 2 nodes


larger
search ends at this null link
attached new node with red link
colors flipped to black
smaller
search ends at this null link
attached new node with red link
rotated right
colors flipped to black
between
search ends at this null link
attached new node with red link
rotated left
rotated right
colors flipped to black

Insert into a 3-node at the bottom:

- Do standard BST insertion; 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)

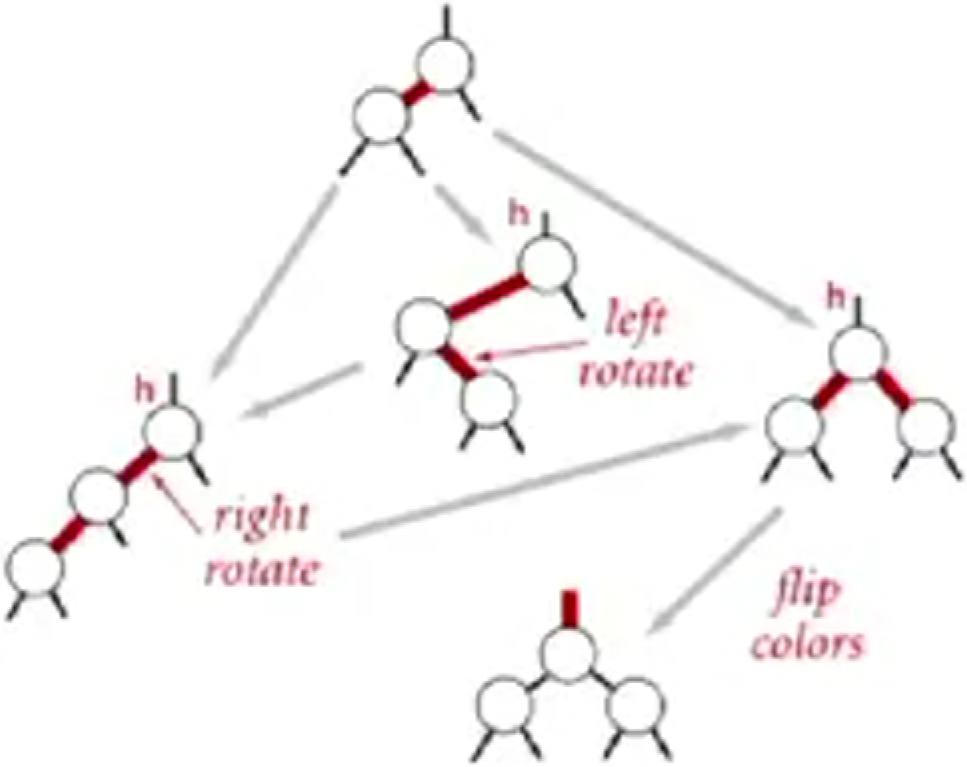

inserting H
E
C
S
A
R
add new node here
two lefts in a row so rotate right
E
C
S
A
R
H
both children red so flip colors
E
C
R
A
H
S
right link red so rotate left
E
C
R
A
H
S
R
E
S
C
H
A

Implementation

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

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

  if (isRed(node.right) && !isRed(node.left)) node = rotateLeft(node);
  if (isRed(node.left) && isRed(node.left.left)) node = rotateRight(node);
  if (isRed(node.left) && isRed(node.right)) flipColors(node);
  return node;
}
```


h
left rotate
h
h
right rotate
flip colors