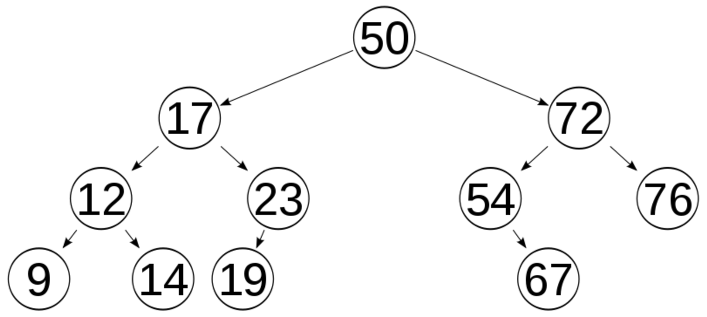
CMSC 132: Object-Oriented Programming II

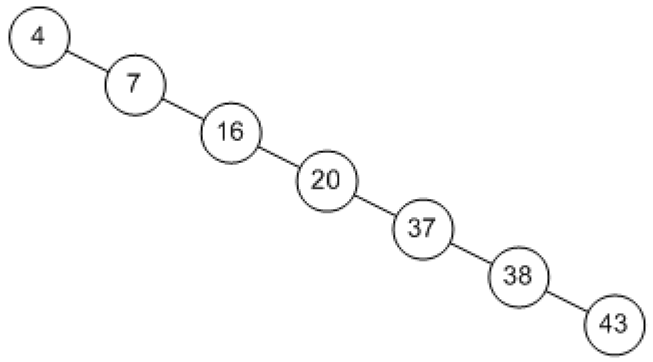
Red & Black Tree

BST



Balanced BST

Search: O(Log n)



Unbalanced BST

Search: O(n)

Quiz 1

What is the worst case time complexity for search, insert and delete operations in a general Binary Search Tree?

A. O(n) for all
B. O(Logn) for all
C. O(Logn) for search and insert, and O(n) for delete
D. O(Logn) for search, and O(n) for insert and delete

Quiz 1

What is the worst case time complexity for search, insert and delete operations in a general Binary Search Tree?

A. O(n) for all
B. O(Logn) for all
C. O(Logn) for search and insert, and O(n) for delete
D. O(Logn) for search, and O(n) for insert and delete

Quiz 2

To delete a node X with 2 non-null children in a BST, we replace the node X with the minimum node Y from X's right subtree. Which of the following is true about the node Y?

A. Y is always a leaf node
B. Y is always either a leaf node or a node with empty left child
C. Y may be an ancestor of the node
D. Y is always either a leaf node or a node with empty right child

Quiz 2

To delete a node X with 2 non-null children in a BST, we replace the node X with the minimum node Y from X's right subtree. Which of the following is true about the node Y?

A. Y is always a leaf node
B. Y is always either a leaf node or a node with empty left child
C. Y may be an ancestor of the node
D. Y is always either a leaf node or a node with empty right child

Quiz 3

We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with the given set so that it becomes a binary search tree?

A. 0
B. 1
C. n!
D. n^2

Quiz 3

We are given a set of n distinct elements and an unlabeled binary tree with n nodes. In how many ways can we populate the tree with the given set so that it becomes a binary search tree?

A. 0
B. 1
C. n!
D. n^2

Quiz 4

Which of the following traversal outputs the data in sorted order in a BST?

A. Preorder
B. Inorder
C. Postorder
D. Level order

Quiz 4

Which of the following traversal outputs the data in sorted order in a BST?

A. Preorder
B. Inorder
C. Postorder
D. Level order

Quiz 5

Numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. What is the in-order traversal sequence of the resultant tree?

A. 7 5 1 0 3 2 4 6 8 9
B. 0 2 4 3 1 6 5 9 8 7
C. 0 1 2 3 4 5 6 7 8 9
D. 9 8 6 4 2 3 0 1 5 7

Quiz 5

Numbers 7, 5, 1, 8, 3, 6, 0, 9, 4, 2 are inserted in that order into an initially empty binary search tree. What is the in-order traversal sequence of the resultant tree?

A. 7 5 1 0 3 2 4 6 8 9
B. 0 2 4 3 1 6 5 9 8 7
C. 0 1 2 3 4 5 6 7 8 9
D. 9 8 6 4 2 3 0 1 5 7

Quiz 6

- The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

A. 2
B. 3
C. 4
D. 6

Quiz 6

- The following numbers are inserted into an empty binary search tree in the given order: 10, 1, 3, 5, 15, 12, 16. What is the height of the binary search tree (the height is the maximum distance of a leaf node from the root)?

A. 2
[✓ B. 3]
C. 4
D. 6

```
  10
/     \
1       15
\      /  \
 3     12   16
  \
   5
```

Quiz 7

The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the postorder traversal sequence of the same tree?

A. 10, 20, 15, 23, 25, 35, 42, 39, 30
B. 15, 10, 25, 23, 20, 42, 35, 39, 30
C. 15, 20, 10, 23, 25, 42, 35, 39, 30
D. 15, 10, 23, 25, 20, 35, 42, 39, 30

Quiz 7

The preorder traversal sequence of a binary search tree is 30, 20, 10, 15, 25, 23, 39, 35, 42. Which one of the following is the postorder traversal sequence of the same tree?

A. 10, 20, 15, 23, 25, 35, 42, 39, 30
B. 15, 10, 25, 23, 20, 42, 35, 39, 30
C. 15, 20, 10, 23, 25, 42, 35, 39, 30
D. 15, 10, 23, 25, 20, 35, 42, 39, 30

```
         30
      /       \
     20         39
    /  \       /  \
  10     25  35    42
   \    /
   15 23
```

Quiz 8

Which of the following traversals is sufficient to construct BST from given traversals 1) Inorder 2) Preorder 3) Postorder

A. Any one of the given three traversals is sufficient
B. Either 2 or 3 is sufficient
C. 2 and 3
D. 1 and 3

Quiz 8

Which of the following traversals is sufficient to construct BST from given traversals 1) Inorder 2) Preorder 3) Postorder

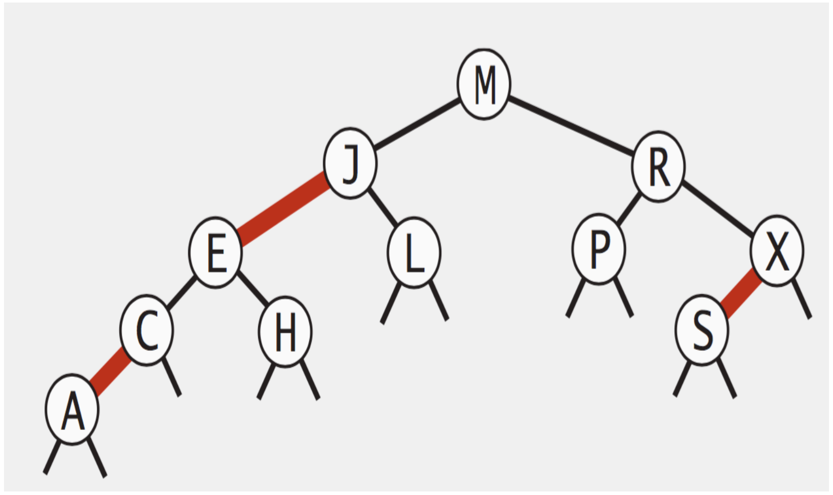
A. Any one of the given three traversals is sufficient
B. Either 2 or 3 is sufficient
C. 2 and 3
D. 1 and 3

Balanced Binary Search Tree

- Red & Black Tree
- AVL Tree
- 2-3 Tree
- B-tree: Databases

Red & Black Tree

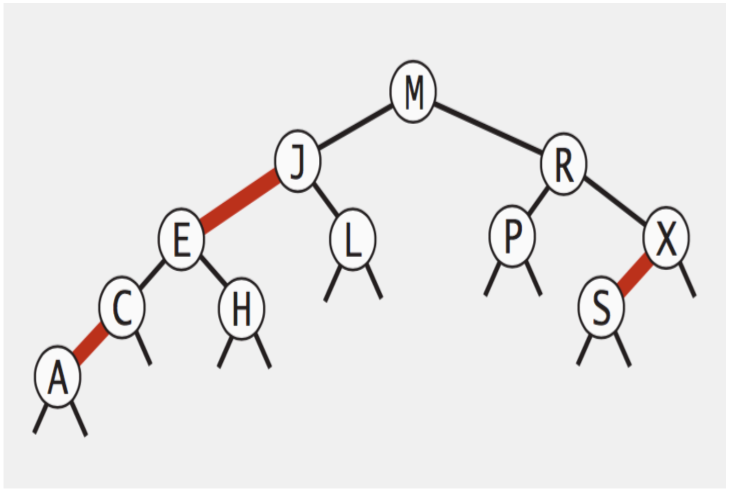
- A BST such that:
  - Tree edges have color: Red or Black
  - No node has two red edges connected to it.
  - Every path from root to null link has the same number of black links.
  - Red links lean left. (LLRB)
  - New node edge is Red



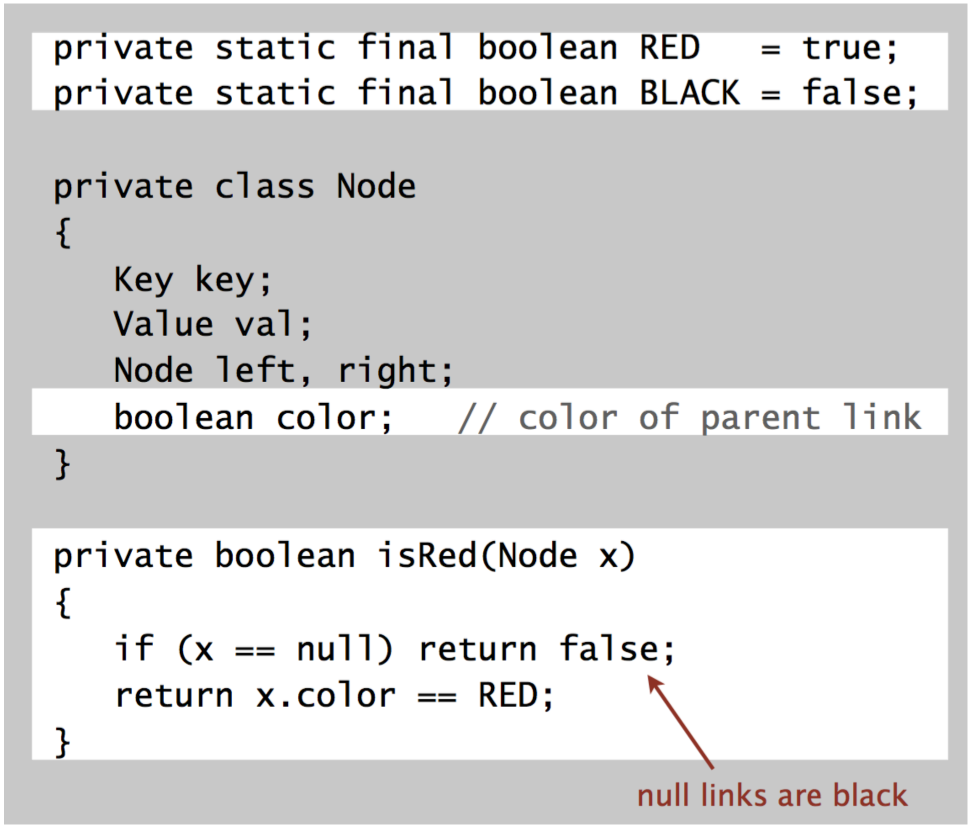
Search: red-black BSTs

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

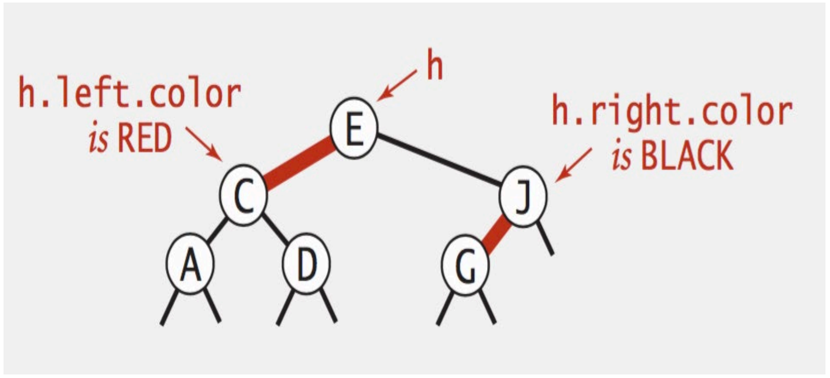
```
public Val get(Key key)
{
   Node x = root;
   while (x != null)
   {
      int cmp = key.compareTo(x.key);
      if      (cmp  < 0) x = x.left;
      else if (cmp  > 0) x = x.right;
      else if (cmp == 0) return x.val;
   }
   return null;
}
```



Red-black BST representation

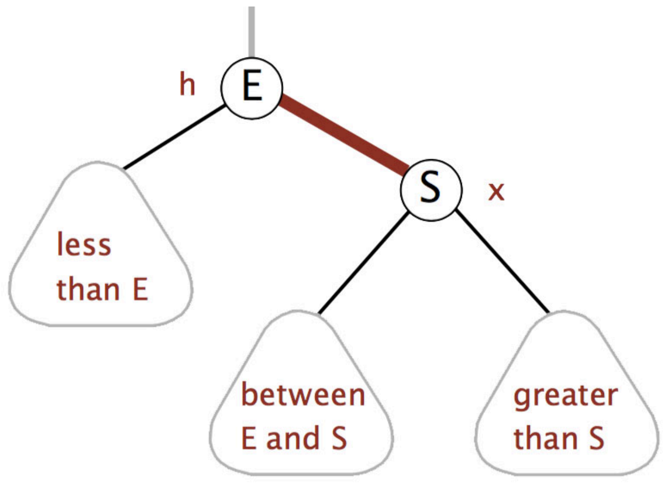


```
private static final boolean RED   = true;
private static final boolean BLACK = false;

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;
}
```

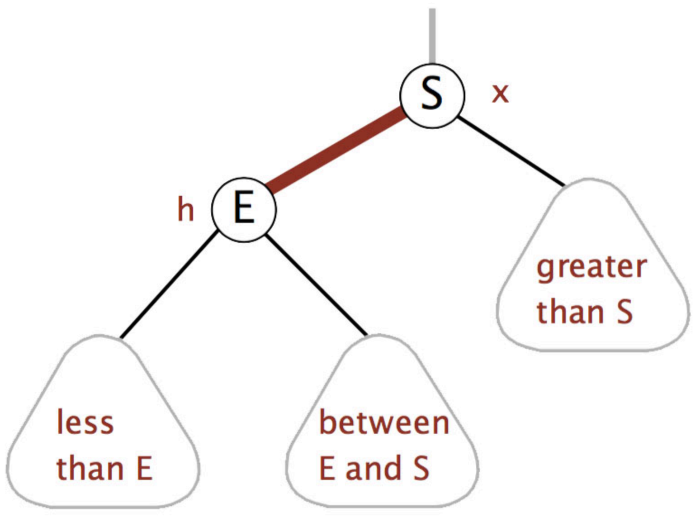


Elementary Operations

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



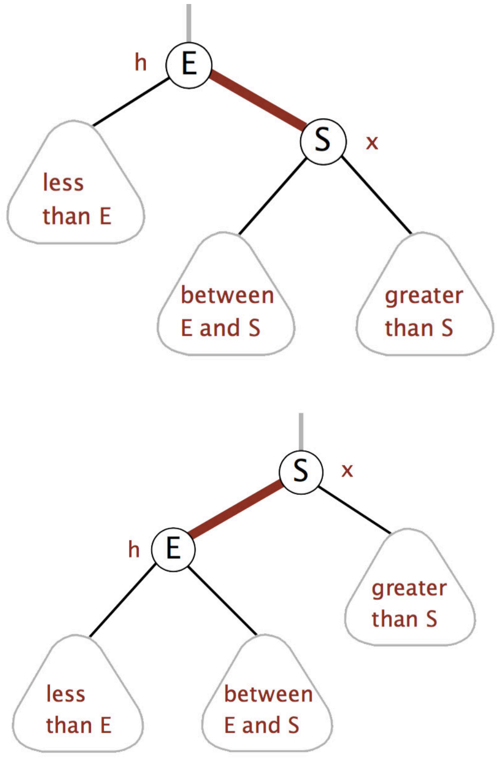
rotate E left (before)



rotate E left (after)

Elementary Operations cont.

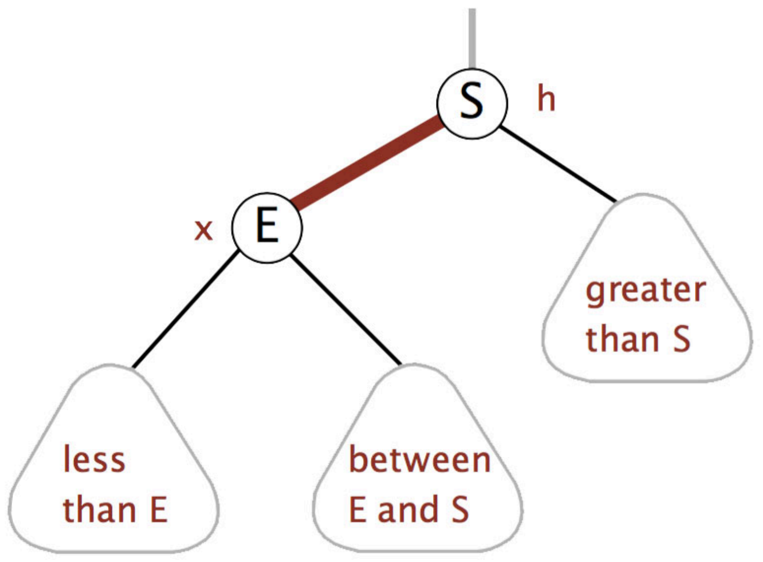
- Left rotation. Orient a (temporarily) right-leaning red link to lean left.



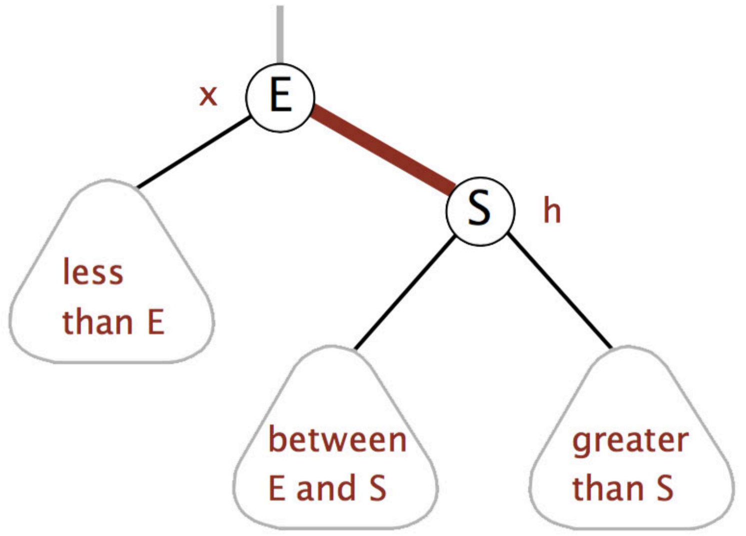
```
private Node rotateLeft(Node h)
{
   assert isRed(h.right);
   Node x = h.right;
   h.right = x.left;
   x.left = h;
   x.color = h.color;
   h.color = RED;
   return x;
}
```

Elementary Operations cont.

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



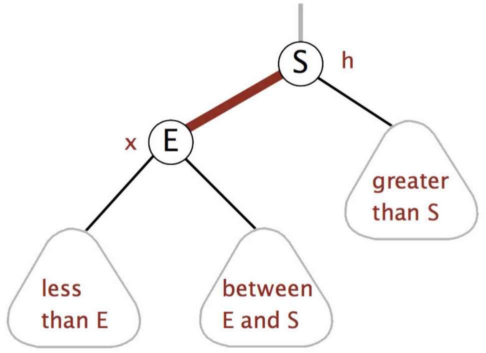
rotate E left (before)

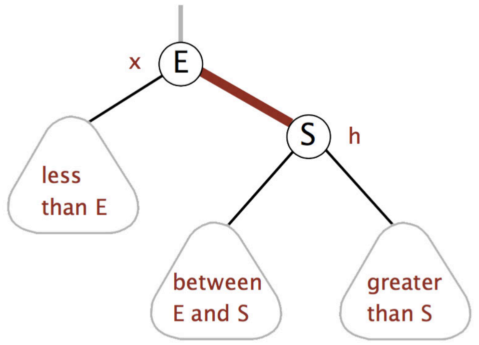


rotate E left (after)

Elementary Operations cont.

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

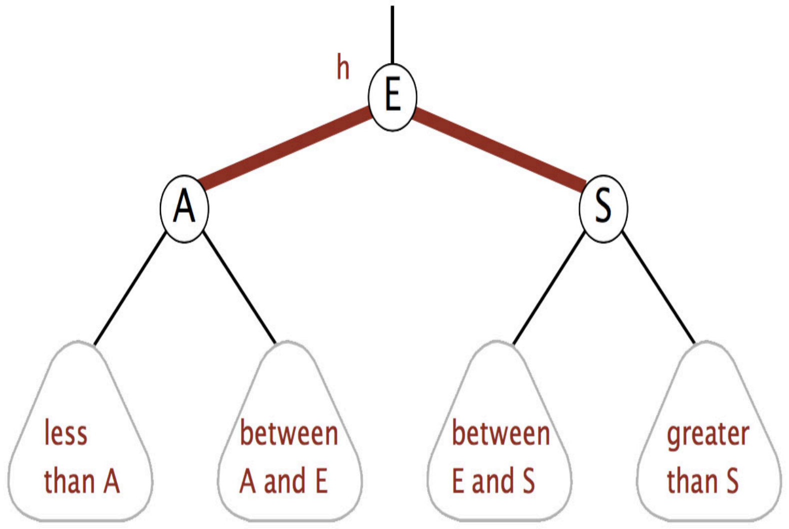


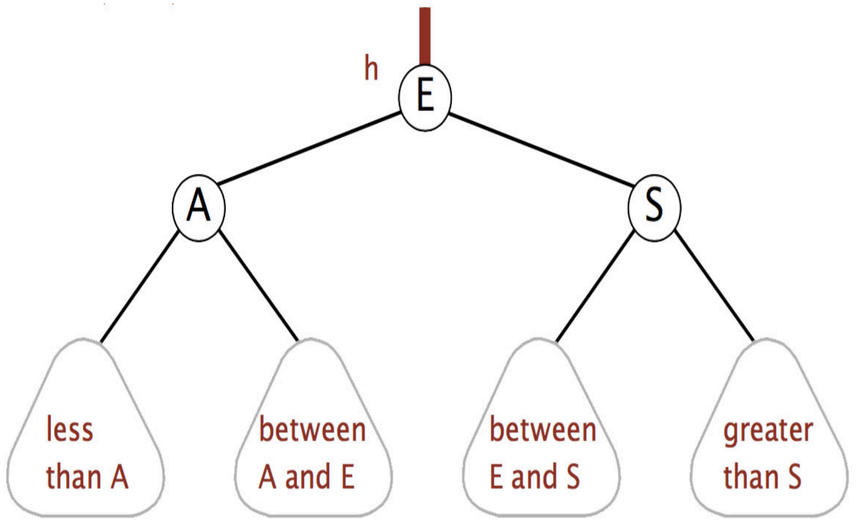


```
private Node rotateRight(Node h)
{
   assert isRed(h.left);
   Node x = h.left;
   h.left = x.right;
   x.right = h;
   x.color = h.color;
   h.color = RED;
   return x;
}
```

Elementary Operations cont.

Color flip:



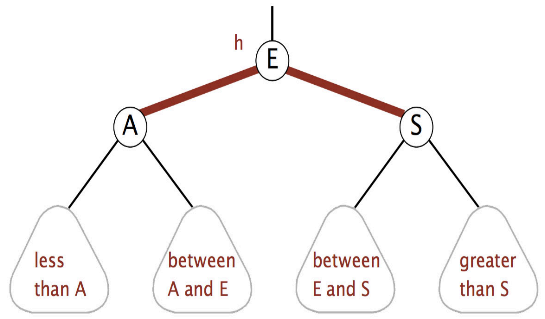


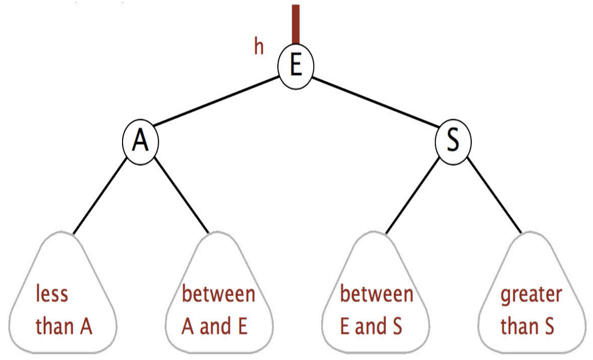
Color flip(before)

Color flip (after)

Elementary Operations cont.

- Color flip.

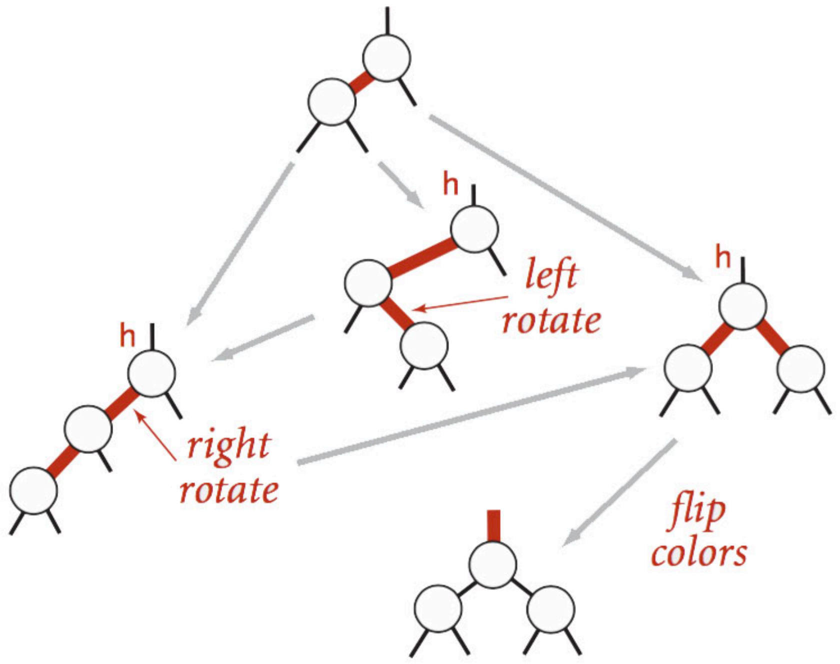




```java
private void flipColors(Node h)
{
   assert !isRed(h);
   assert isRed(h.left);
   assert isRed(h.right);
   h.color = RED;
   h.left.color = BLACK;
   h.right.color = BLACK;
}
```

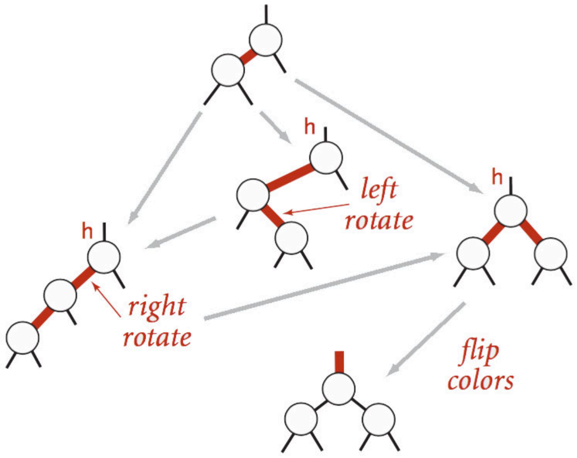
Insertion in a LLRB tree

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



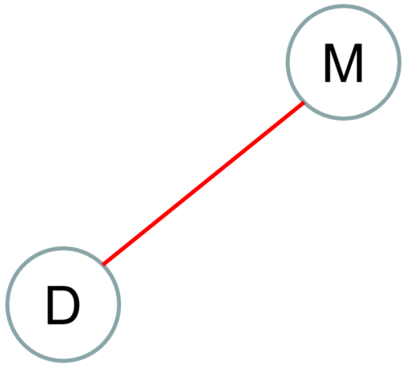
Insertion

```
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);
  return h;
}
```



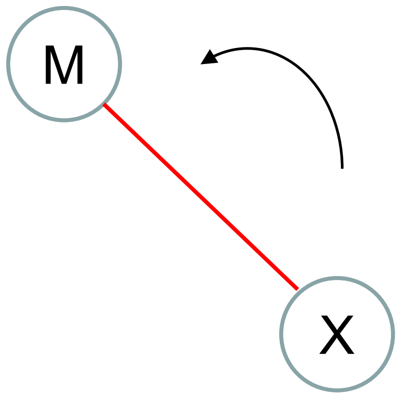
R&B Example: Insertion

Insert: M, D



R&B Example: Insertion

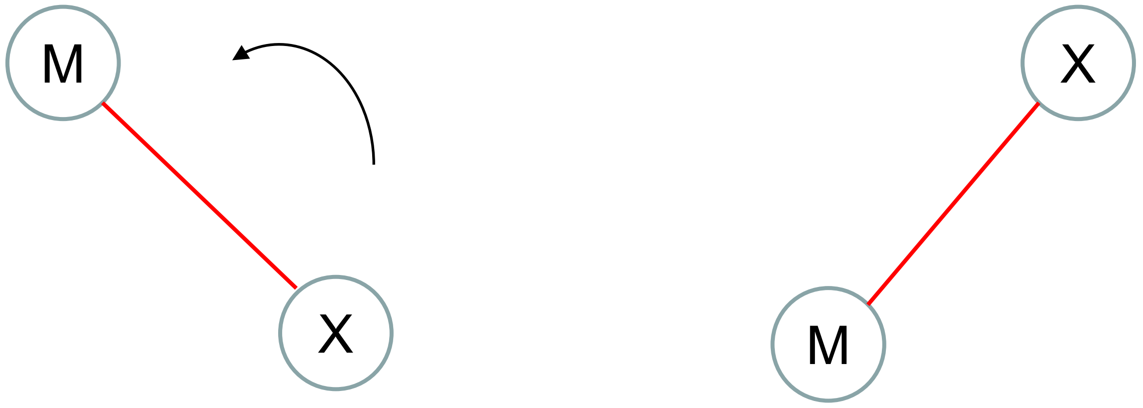
Insert: M, X



Rotate Left

R&B Example: Insertion

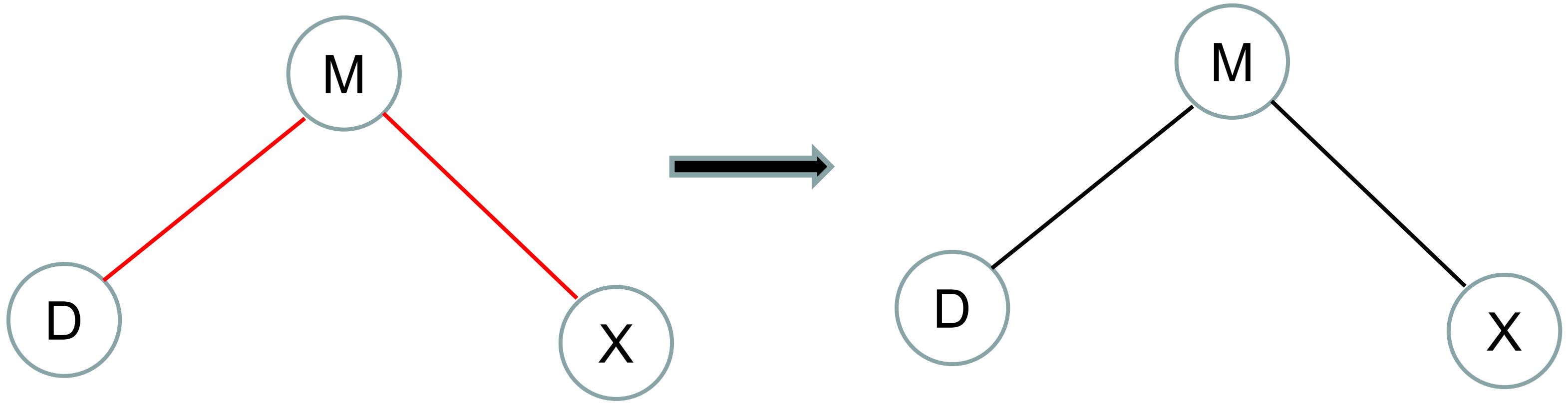
Insert: M, X



Rotate Left

R&B Example: Insertion

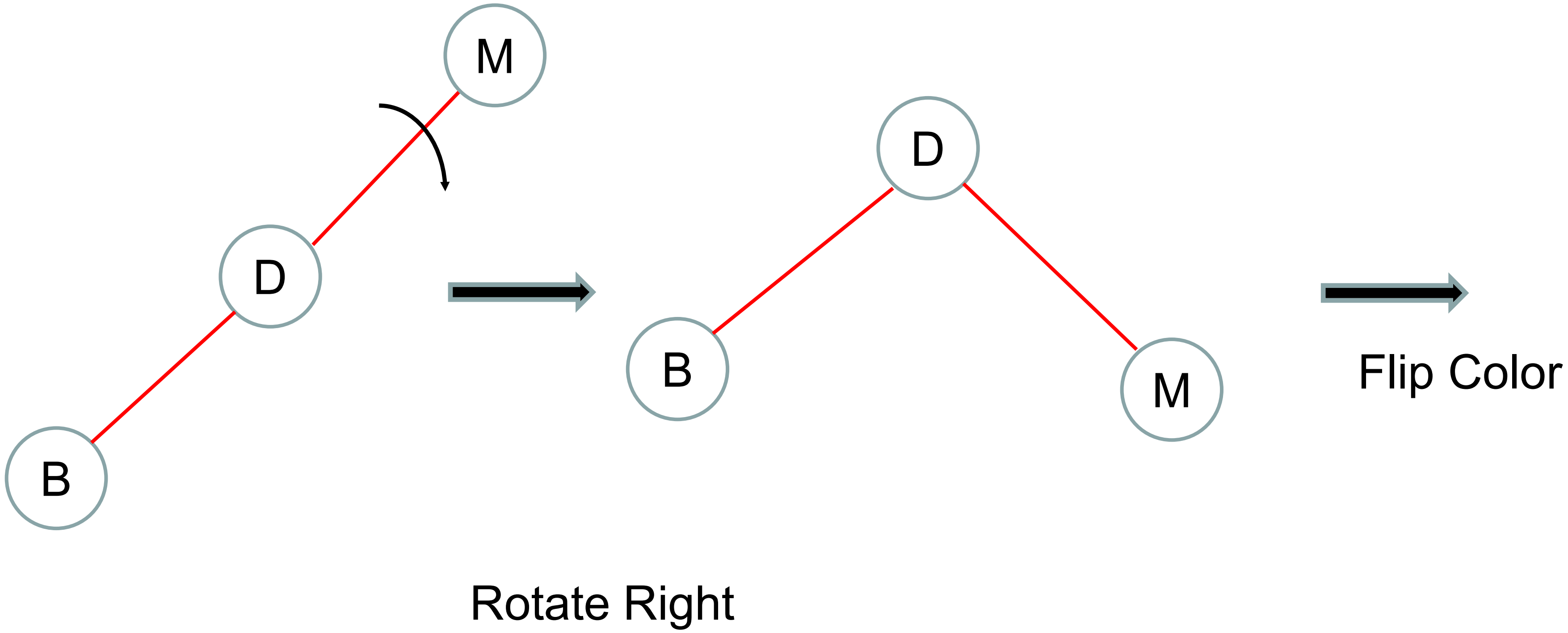
Insert: M, D, X



Flip Color

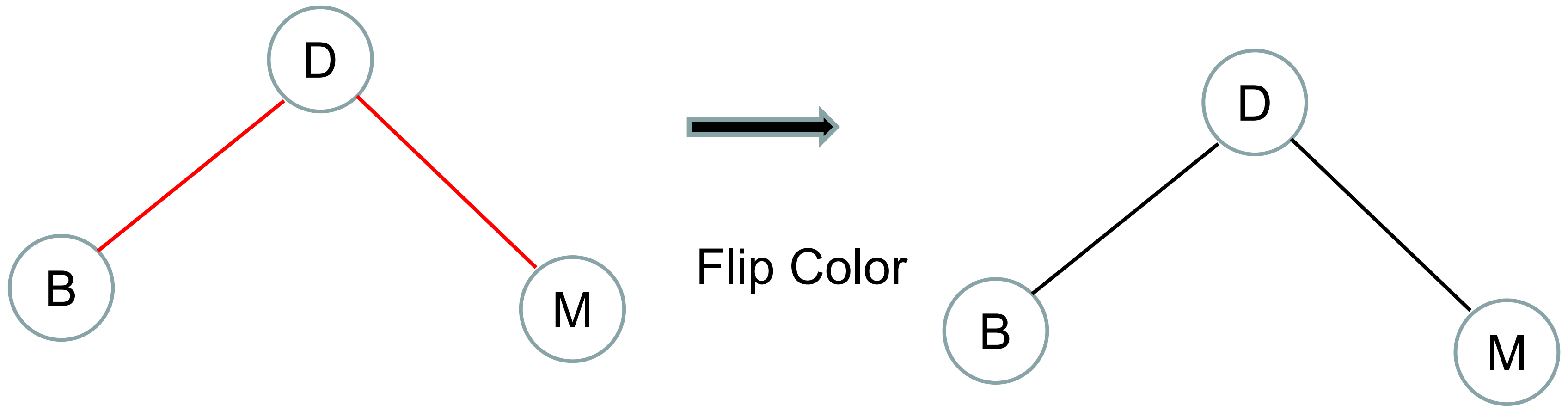
R&B Example: Insertion

Insert: M, D, B



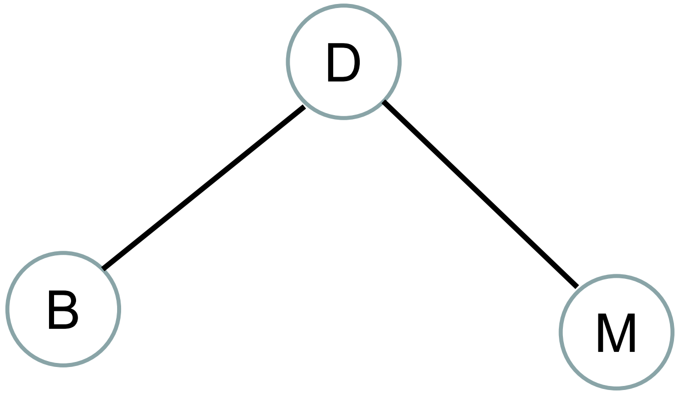
R&B Example: Insertion

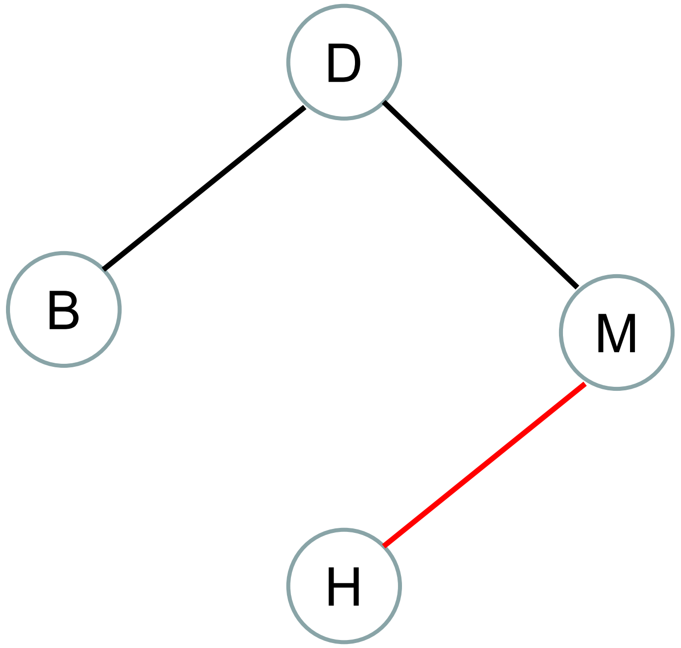
Insert: M, D, B



R&B Example: Insertion

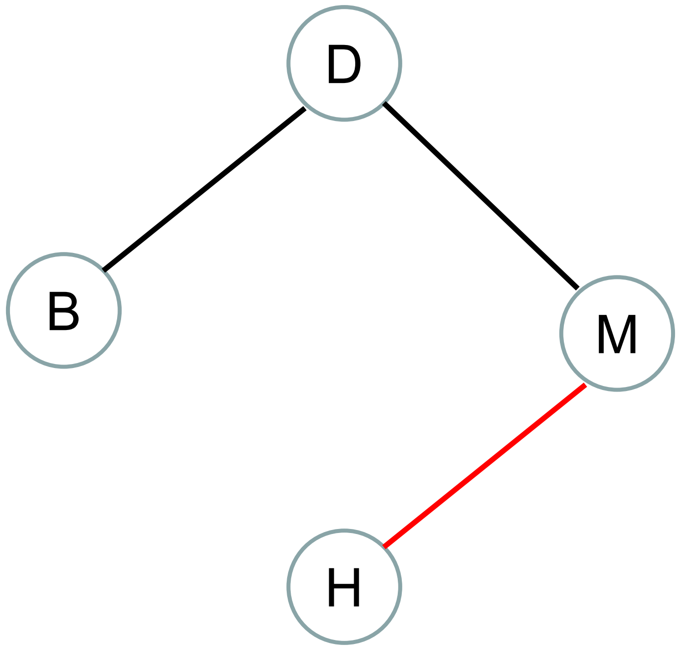
M, D, B   Insert H

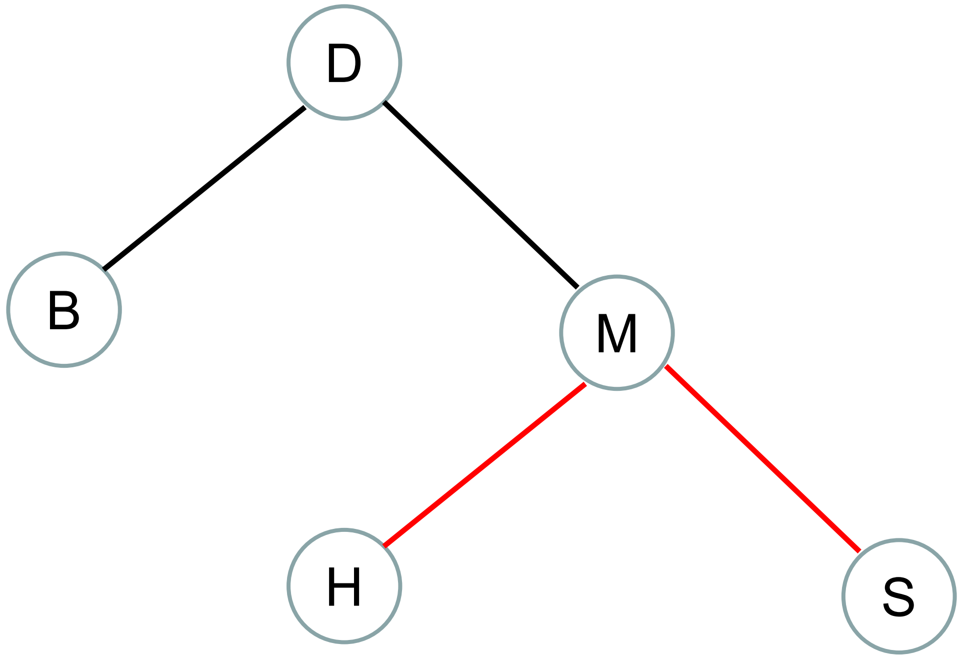




R&B Example: Insertion

M, D, B,H   Insert S

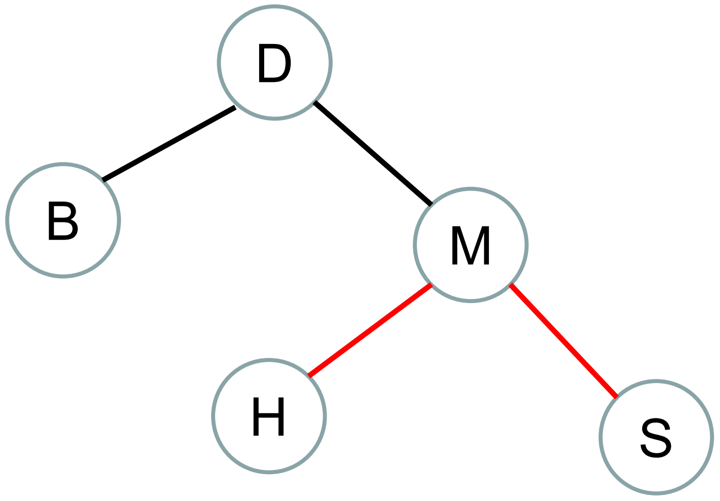




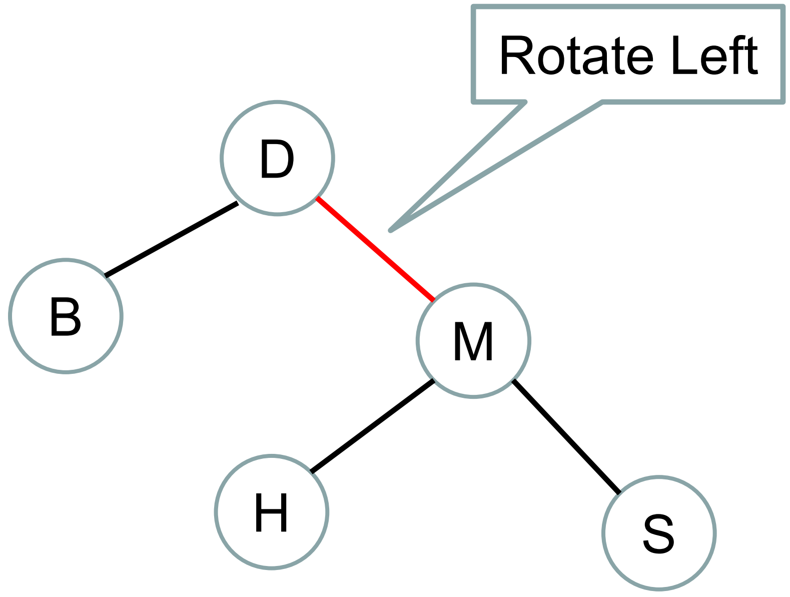
Color flip

R&B Example: Insertion

M, D, B,H   Insert S

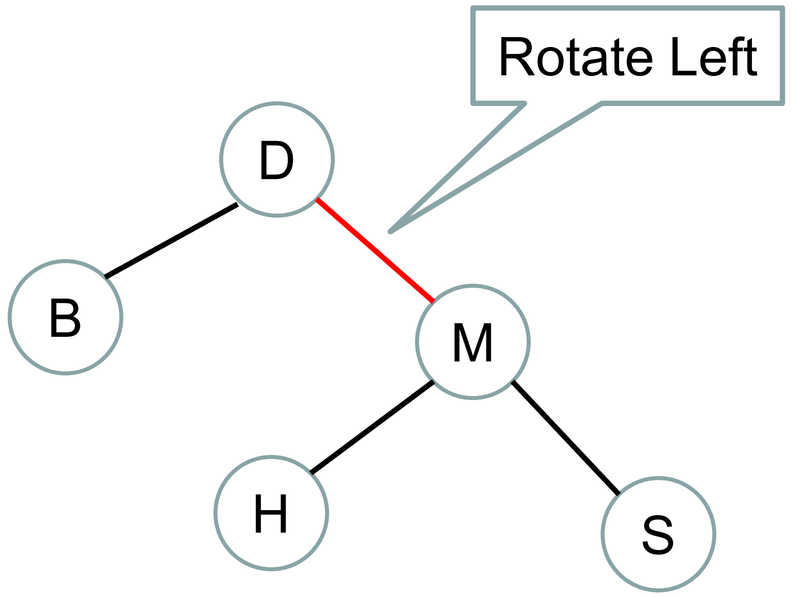


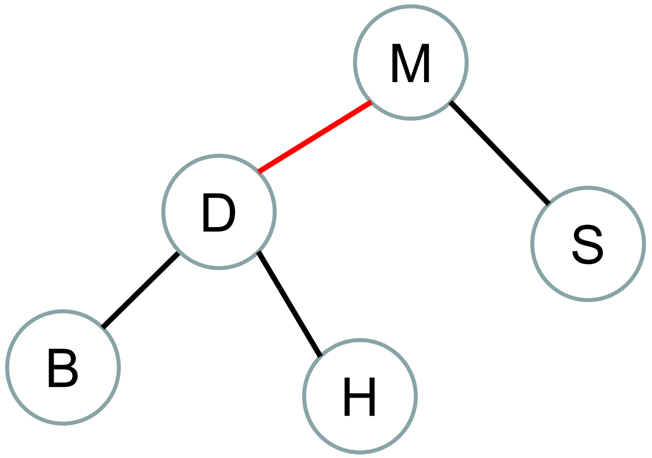
Color flip



R&B Example: Insertion

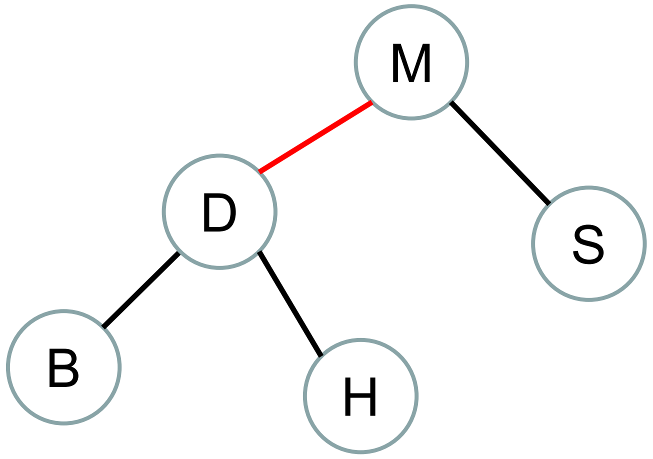
M, D, B,H   Insert S

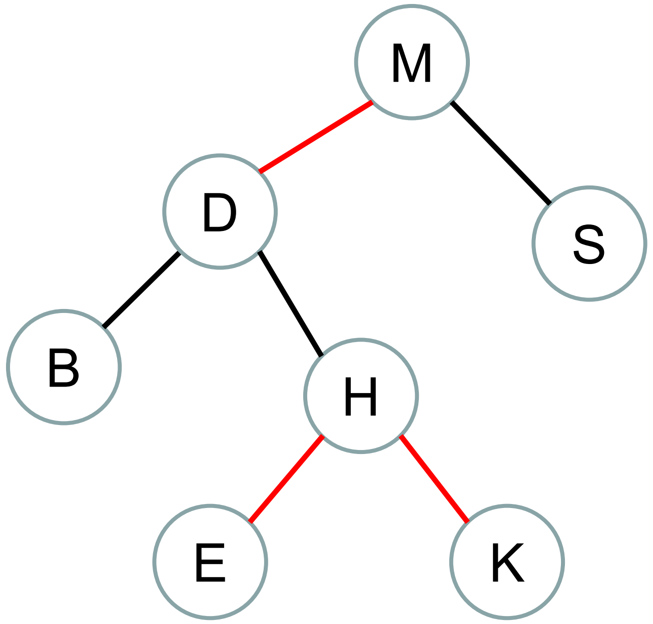




R&B Example: Insertion

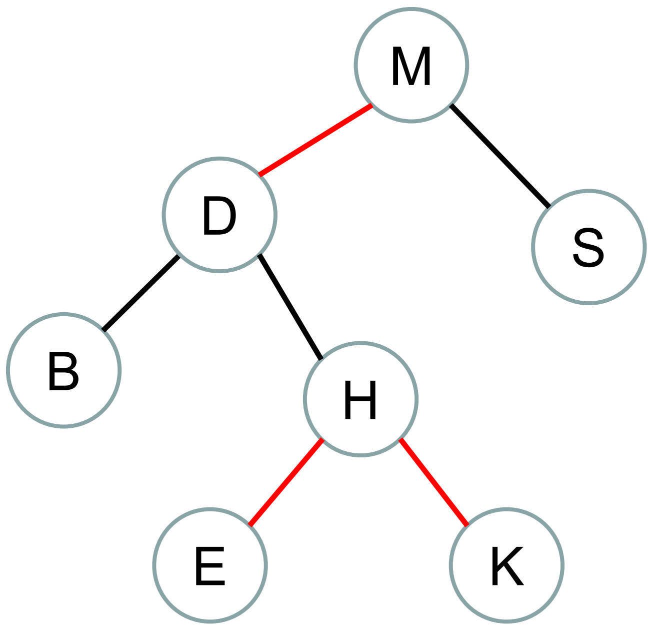
M, D, B,H,S   Insert E,K

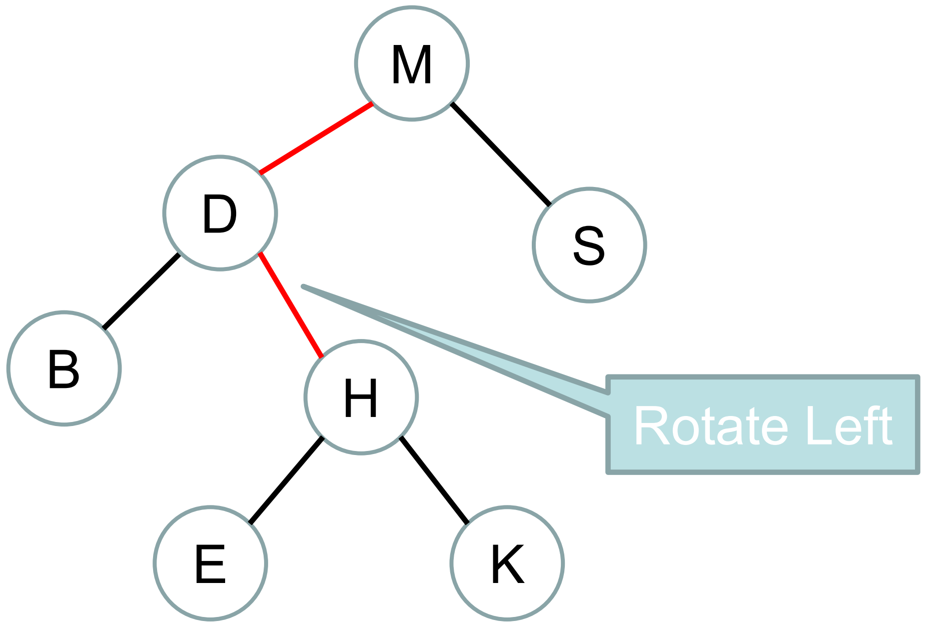




R&B Example: Insertion

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

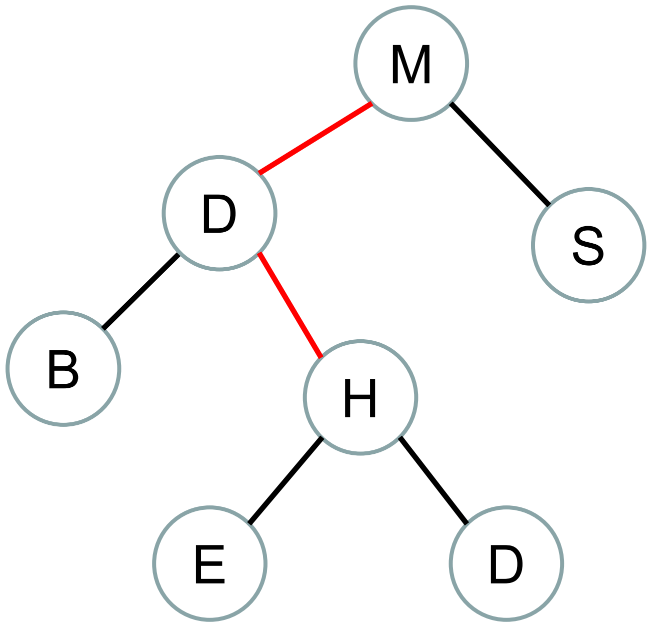




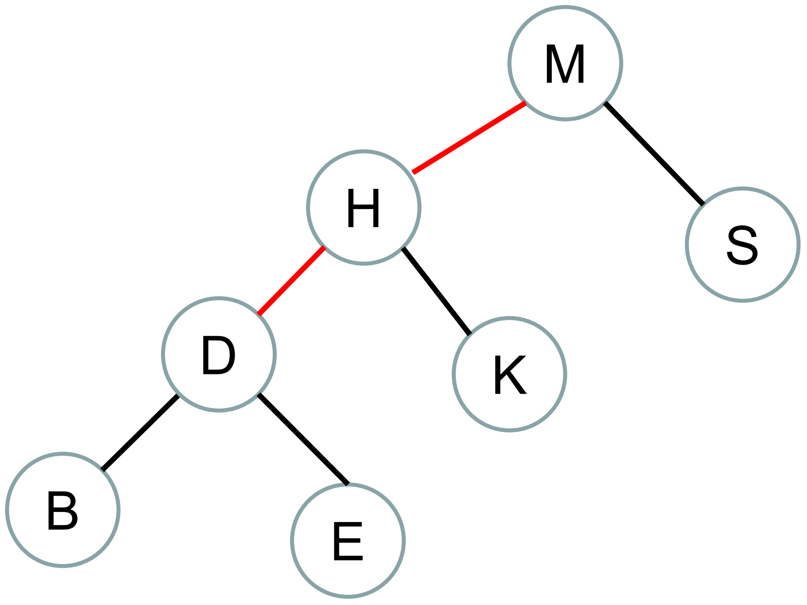
Color flip

R&B Example: Insertion

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

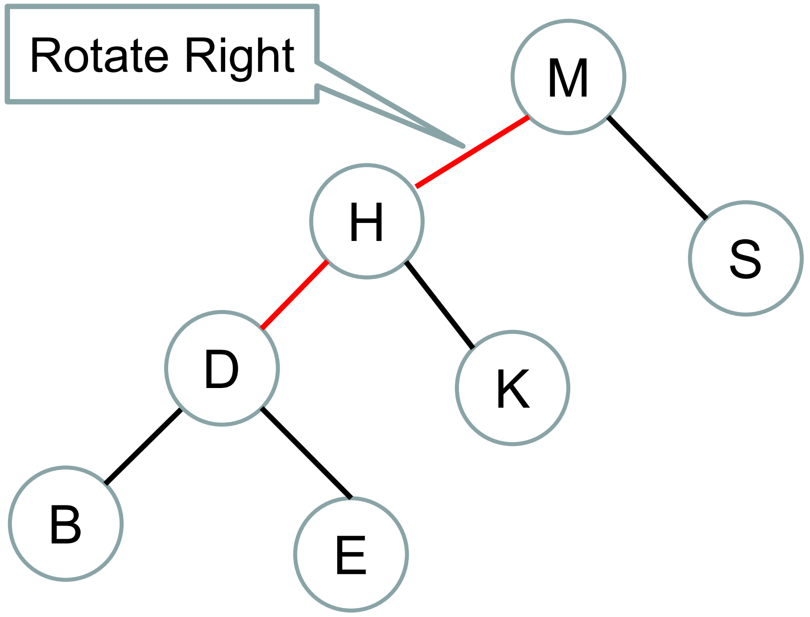


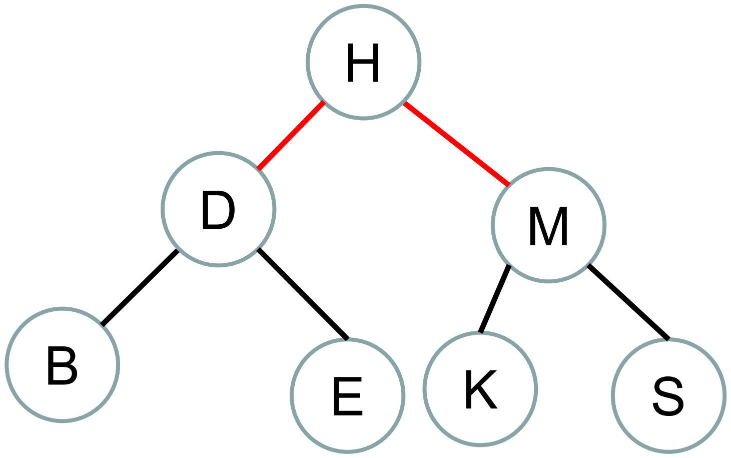
Rotate left



R&B Example: Insertion

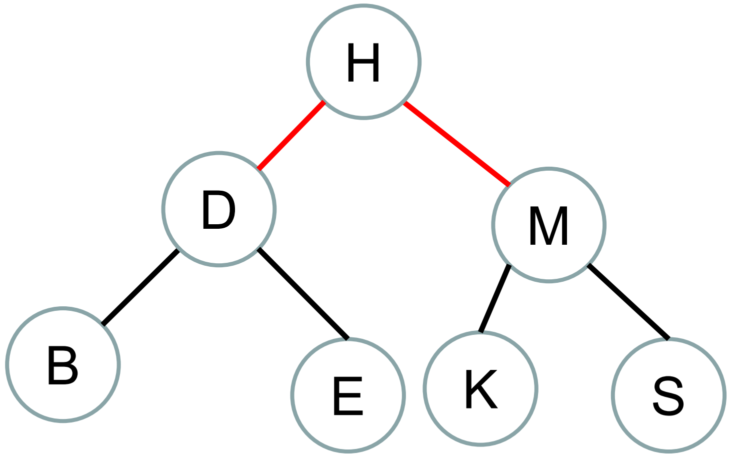
M, D, B,H,S   Insert E,K

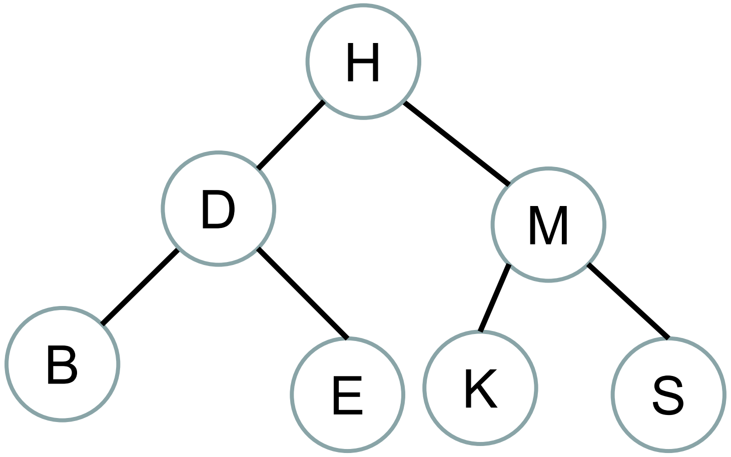




R&B Example: Insertion

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





Color flip

R&B tree vs. BST