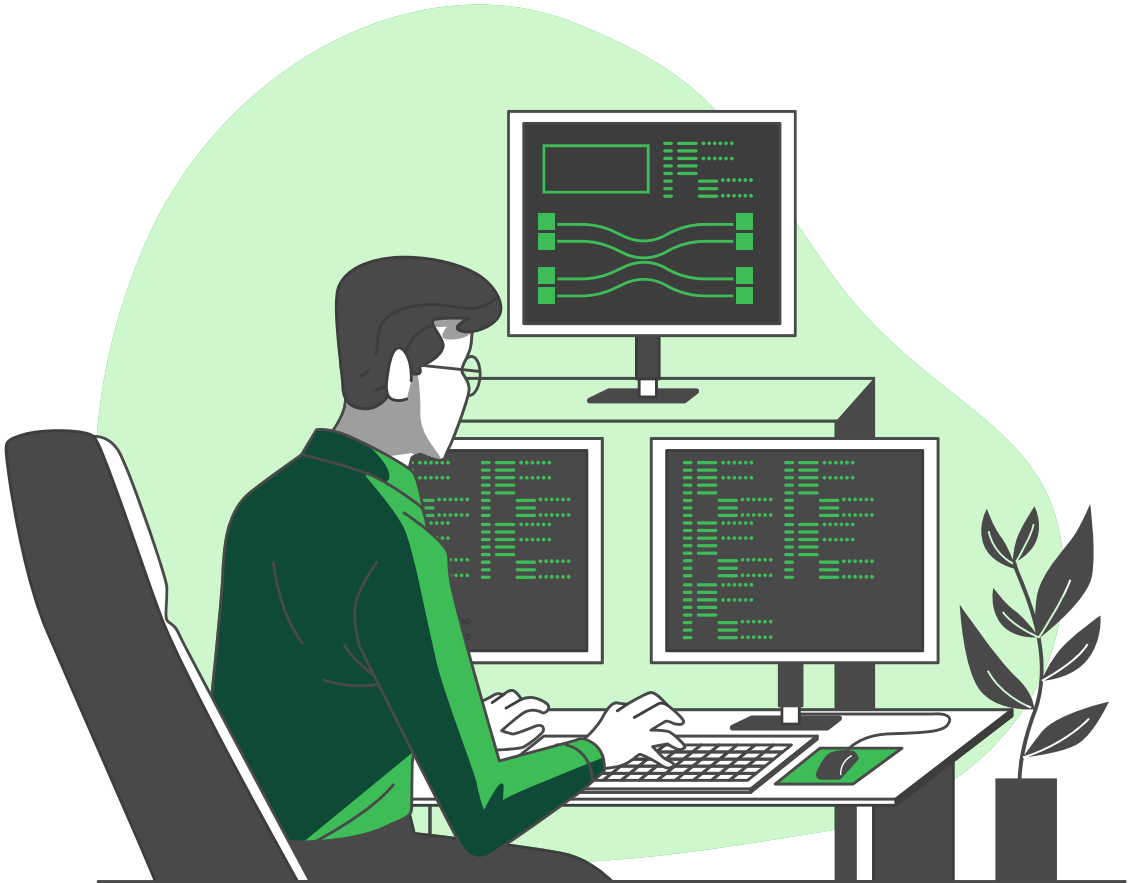


Insertion in Red-Black Trees

CS 251 - Data Structures and Algorithms

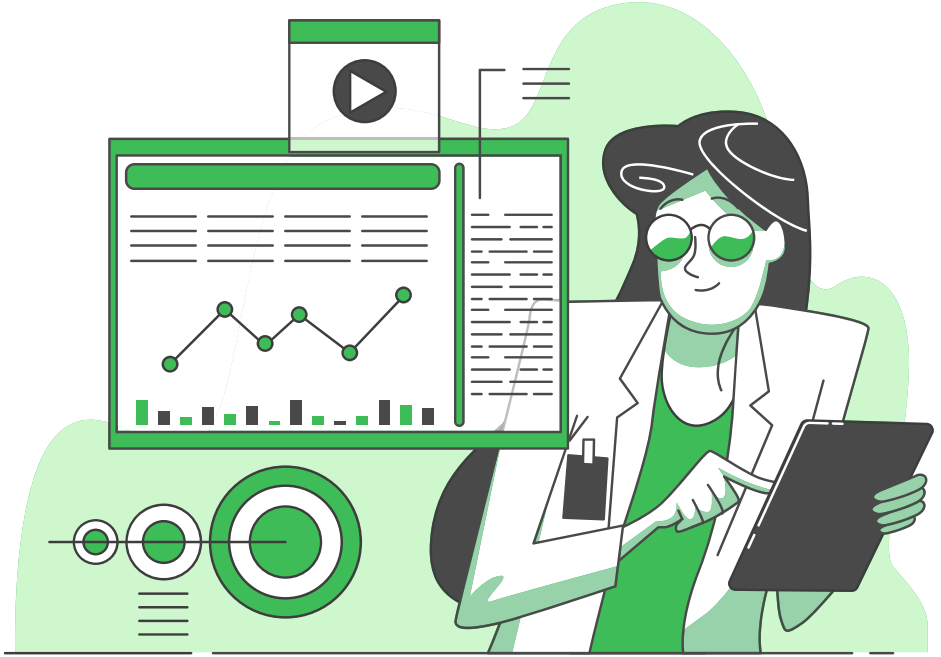
Note:
Slides complement the discussion in class

Table of Contents

01

Insert in Red-Black Trees

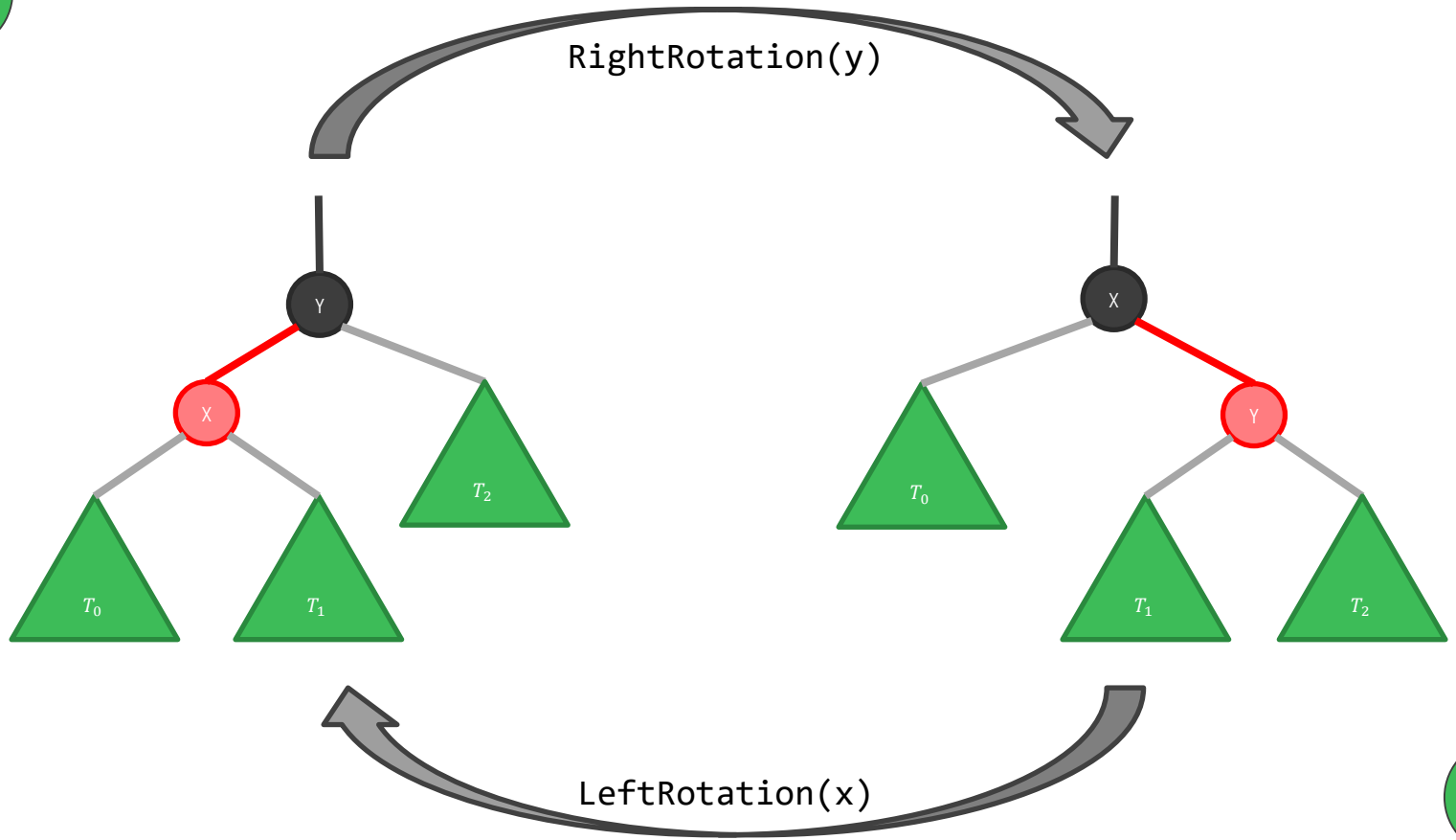
Enforcing “balance” after an insertion



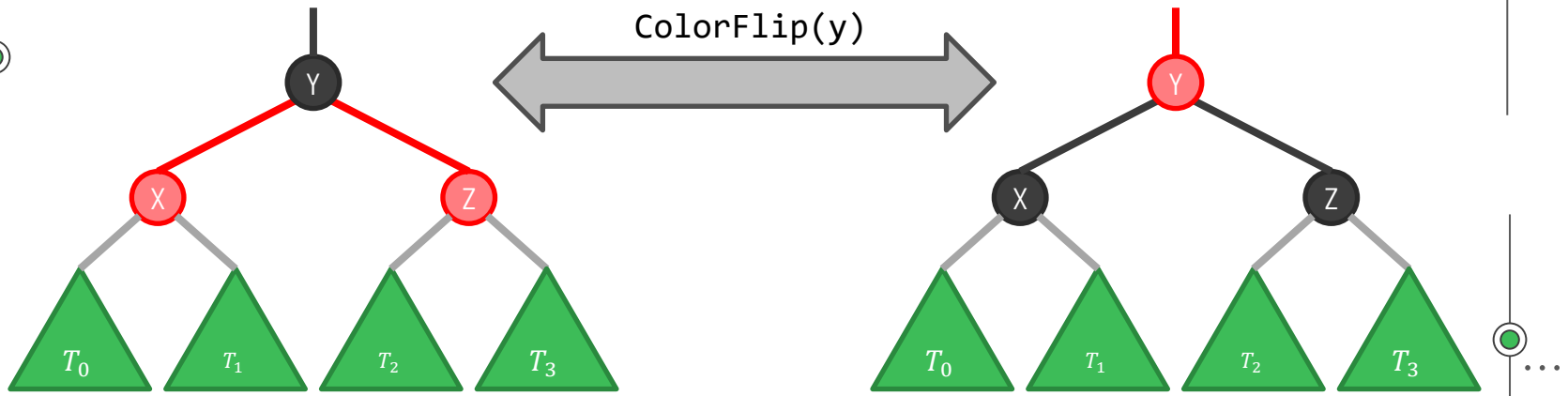
01

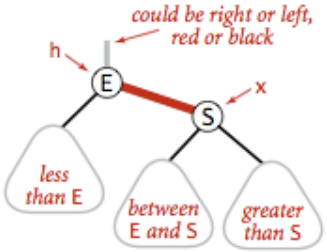
Insert in Red-Black Trees

Enforcing “balance” after an insertion

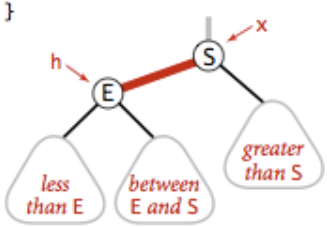


Important: Items rotate, colors don't.

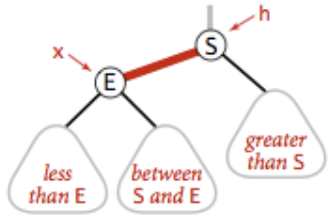

ColorFlip(y)
Y
X
Z
T_0
T_1
T_2
T_3
Y
X
Z
T_0
T_1
T_2
T_3



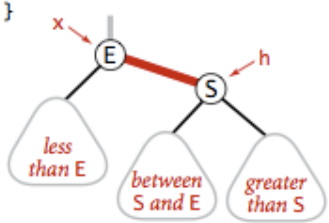

```
Node rotateLeft(Node h)
{
   Node x = h.right;
   h.right = x.left;
   x.left = h;
   x.color = h.color;
   h.color = RED;
   x.N = h.N;
   h.N = 1 + size(h.left)
           + size(h.right);
   return x;
}
```



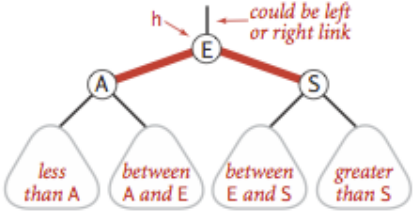
Left rotate (right link of h)



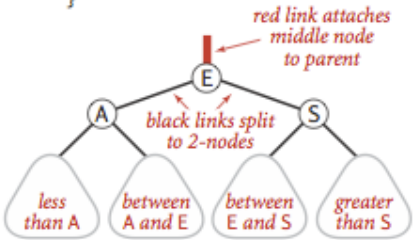

```
Node rotateRight(Node h)
{
   Node x = h.left;
   h.left = x.right;
   x.right = h;
   x.color = h.color;
   h.color = RED;
   x.N = h.N;
   h.N = 1 + size(h.left)
           + size(h.right);
   return x;
}
```



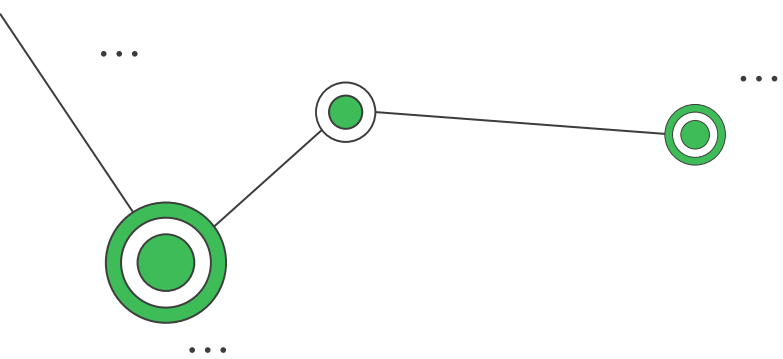
Right rotate (left link of h)




```
void flipColors(Node h)
{
   h.color = RED;
   h.left.color = BLACK;
   h.right.color = BLACK;
}
```



Flipping colors to split a 4-node

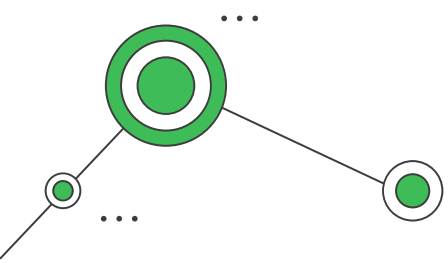


Insert(k)

Insert the key as usual in BSTs.

New nodes are always **red** (it doesn't upset the black height).

Use transformations to **rebalance the black height** of the RBT.



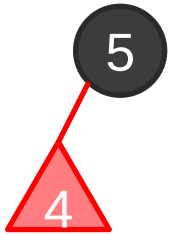
Insert 5, 4, 3, 2, 1 in a Red-Black Tree:

Insert 5:

Root Color Flip

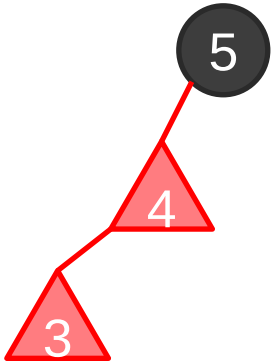
Height = 0
Black height = 1

Insert 4:

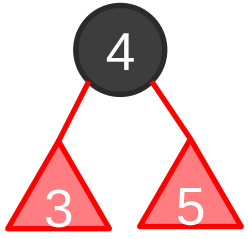


Height = 1
Black height = 1

Insert 3:

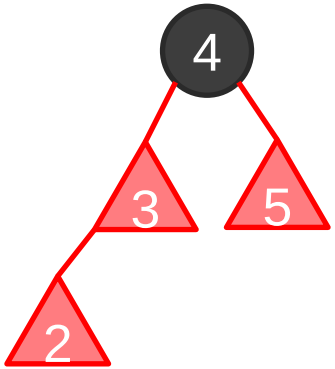


rightrotation(5)

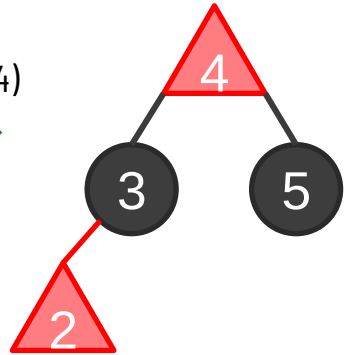


Height = 1
Black height = 1

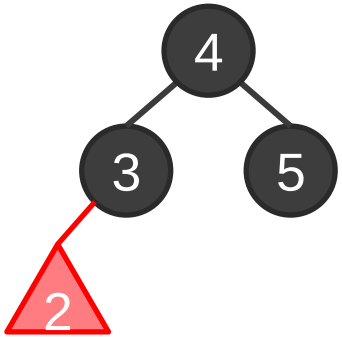
Insert 2:



colorflip(4)

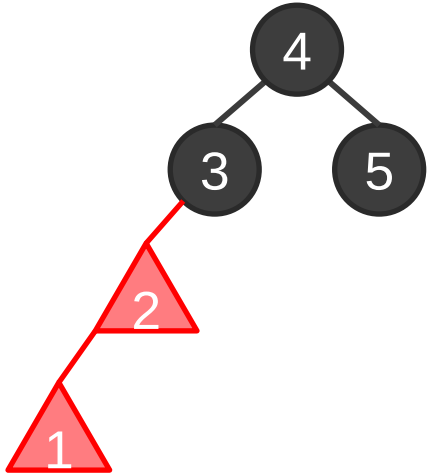


rootcolorflip()

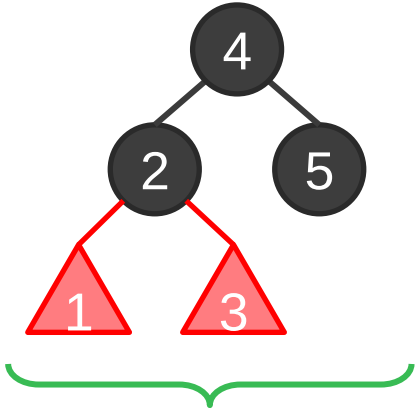


Height = 2
Black height = 2

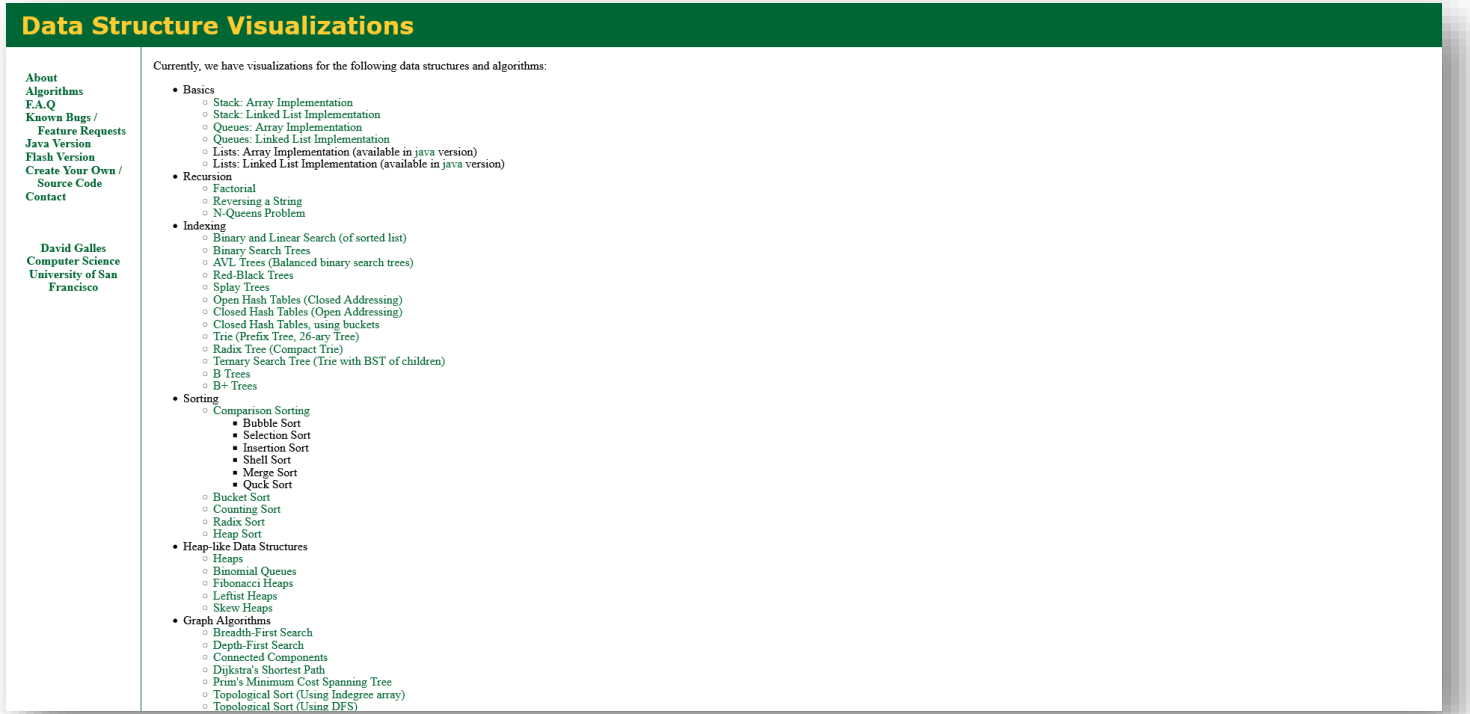
Insert 1:



rightrotation(3)



Height = 2
Black height = 2



Visit https://www.cs.usfca.edu/~galles/visualization/Algorithms.html to practice insertions in Red-Black Trees.

Done!

Do you have any questions?