Red Black Trees
(Guibas Sedgewick 78)

Goal

Keep sorted lists subject to the following operations:

find(x,L)

insert(x,L)

delete(x,L)

catenate(L1,L2) : Assumes that all items in L2 are greater than all items in L1.

split(x,L) : returns two lists one with all item less than or equal to x and the other with all items greater than x.

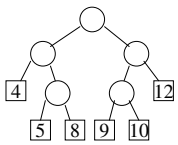
Binary search trees

Binary search tree:

•All internal nodes have degree = 2

•Items are at the leaves.

•All items in left subtree are smaller than all items in right subtree (symmetric order)



Red Black trees - definition

Binary search tree

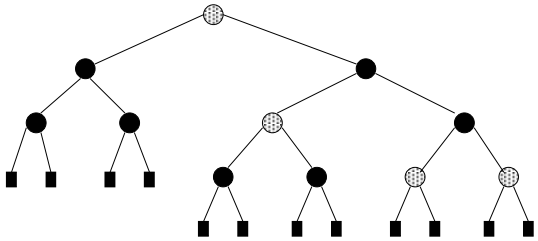
each node is colored red or **black** such that

1) Leaves are **black**

2) All paths from the root to an external node contain the same number of **black** nodes **(black rule)**

3) Any red node, if has a parent, has a **black** parent (red rule)

Red Black trees - example



5

Red Black Trees -properties

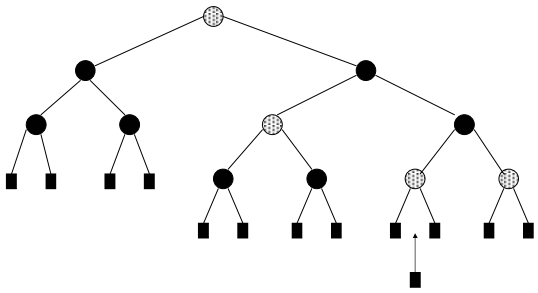
The depth is O(log n) -- prove as an excercise

==> find takes O(log n) time

How do we do the other operations and keep the trees Red-Black ?

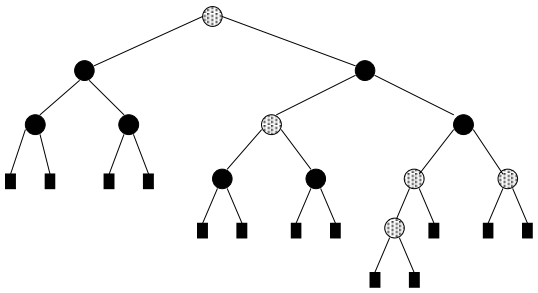
6

Insert



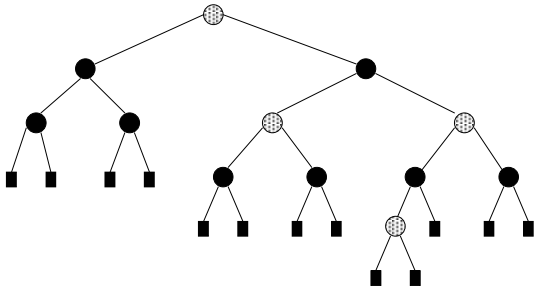
7

Insert (cont)



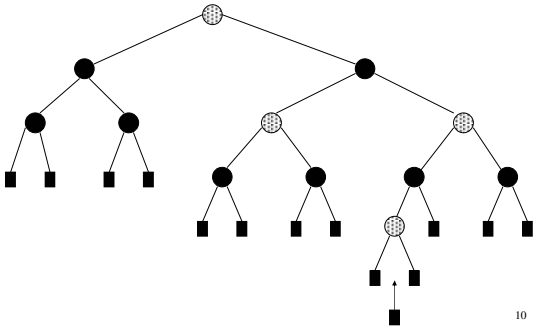
8

Insert (cont)



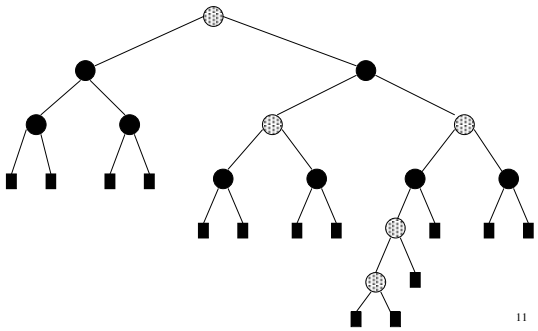
9

Insert (cont)



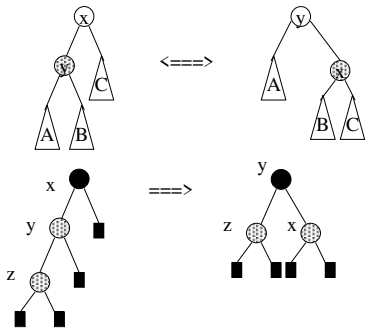
10

Insert (cont)



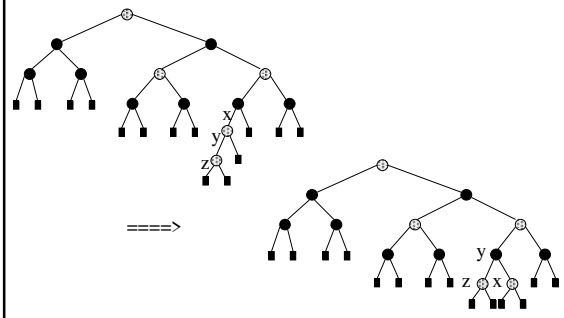
11

Use rotations



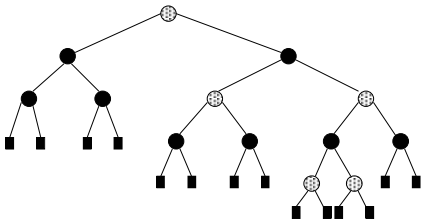
12

Insert (cont)



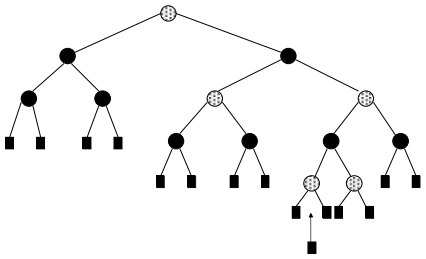
13

Insert (cont)



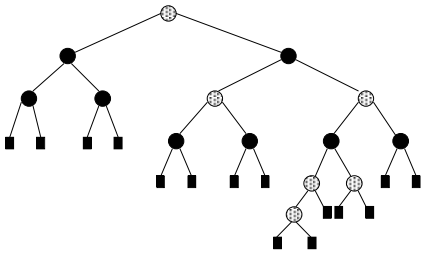
14

Insert (cont)



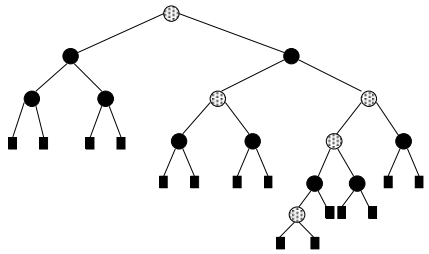
15

Insert (cont)



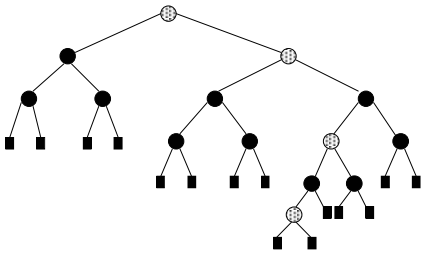
16

Insert (cont)



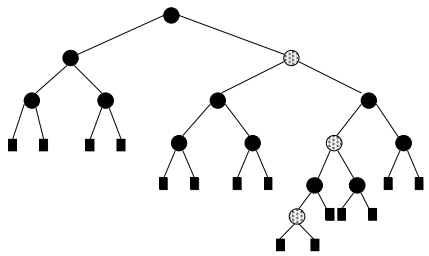
17

Insert (cont)



18

Insert (cont)



19

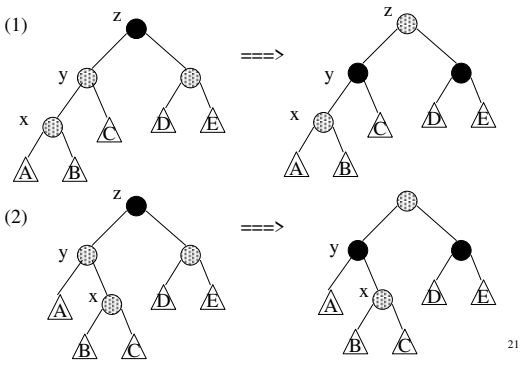
Insert -- definition

Convert a leaf to a red internal node with two leaves.

This may create violation to property 2. To restore it we walk up towards the root applying one of the following cases (each case has a symmetric version)

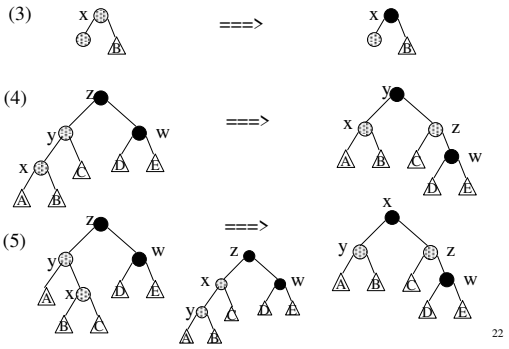
20

Insert -- non terminal cases



21

Insert -- terminal cases



22

Insert - analysis

O(log n) time worst case, since the height is O(log n)

Suppose you start with an empty tree and do m insertions such that the point of insertion is given to you each time, how much time does it take ?

Obviously O(mlog n),

but maybe we can prove it cannot be that bad ?

23

Insert - analysis

Each time we do a color-flip-step the number of red nodes decreases by one.
Φ(tree) = #red nodes

Actual(insert) = O(1) + #color-flips-steps

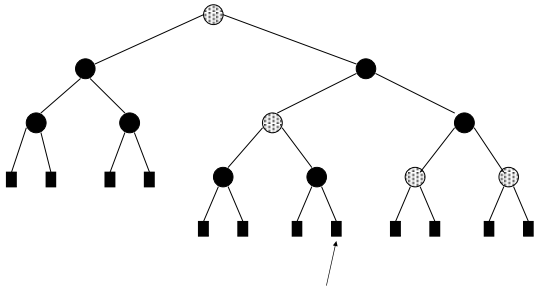
ΔΦ(insert) = O(1) - #color-flips-steps

==> amortized(insert) = O(1)

and the sequence actually takes O(m) time.

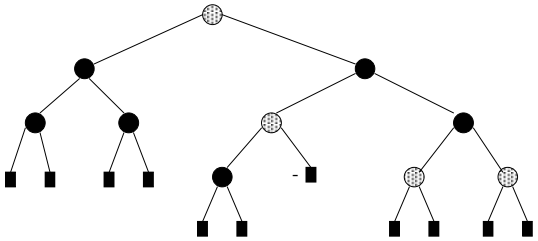
24

Delete -- example



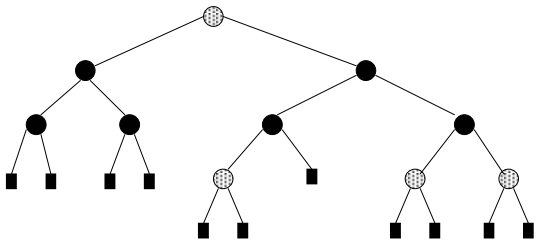
25

Delete -- example (cont)



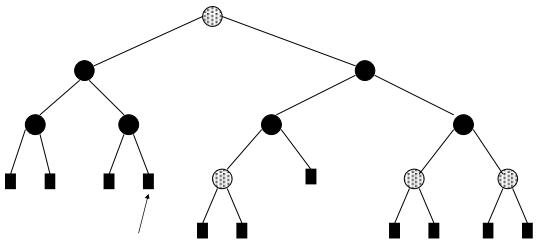
26

Delete -- example (cont)



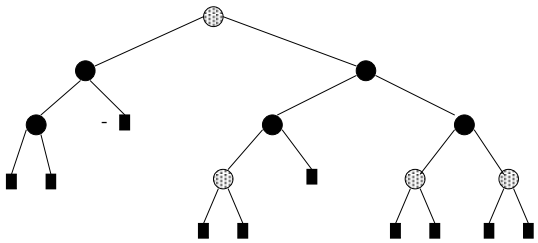
27

Delete -- example2 (cont)



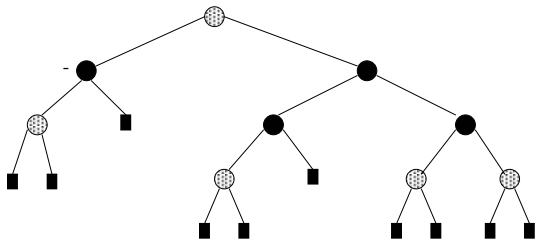
28

Delete -- example2 (cont)



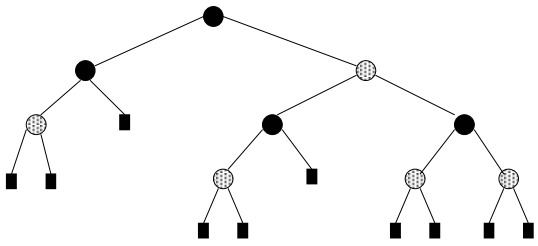
29

Delete -- example2 (cont)



30

Delete -- example2 (cont)



31

Delete -- definition

Replace the parent of the external node containing the item with the sibling subtree of the deleted item

If the parent of the deleted item is black then we create a short node

To restore the black constraint we go bottom up applying one of the following cases.

32

Delete -- fixing a short node

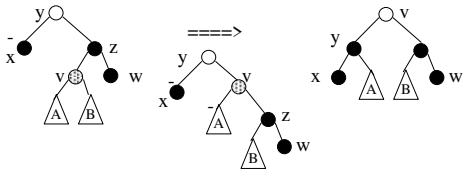
(1)

(2)

Delete -- fixing a short node (cont)

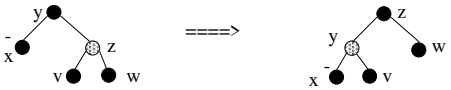
(3)

(4)



Delete -- fixing a short node (cont)

(5)



And apply one of the previous 3 cases.

Delete + insert -- analysis

O(log n) time, since the height is O(log n)

Suppose you start with an empty tree and do m insertions and deletions such that the point of insertion is given to you each time, how much time does it take ?

Obviously O(mlog n),

but maybe we can prove it cannot be that bad ?

Delete + insert - analysis

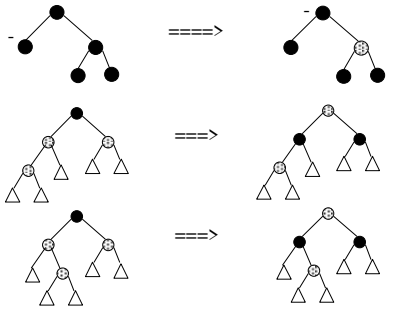
The previous potential won't do the trick

Φ(tree) = #red nodes

Here are the transformation that we want to release potential

37

Delete + insert -- analysis



38

Delete + insert -- analysis

Φ(tree) = #( ) + 2 #( )

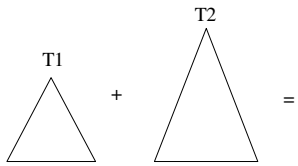
==> amortized(delete) = O(1)

amortized(insert) = O(1)

sequence of m delete and inserts, starting from an empty tree takes O(m) time

39

Concatenation



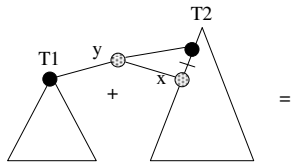
Define the rank of a node v as the number of black nodes from v to a leaf .

Assume T1 has a black root.

Look on the left spine of T2 for a node x of the same rank as the root of T1

40

Concatenation (cont)



Allocate y make the root of T1 and x children of y.

Color y red

Make y a child of p(x)

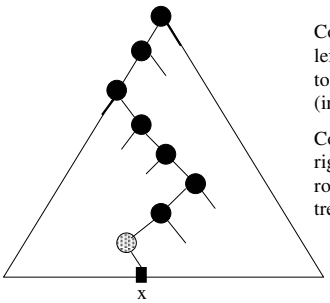
Continue as for insert

Concatenation (analysis)

O(|r1-r2| + 1) = O(log n) worst case.

If the right point on the spine of the taller tree is given then its O(1) amortized

Split



Concatenate all trees to the left of the path from the root to x to from the left tree T1 (including x).

Concatenate all trees to the right of the path from the root to x to from the right tree T2

Split -- analysis.

Can prove that split takes O(log n) time.