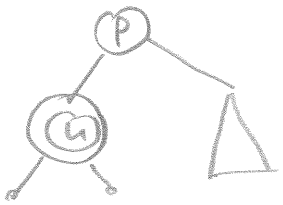
Insertion

- Insert n as in BST

  n.color := red ⇒ Fix up conflicts

- Cases:

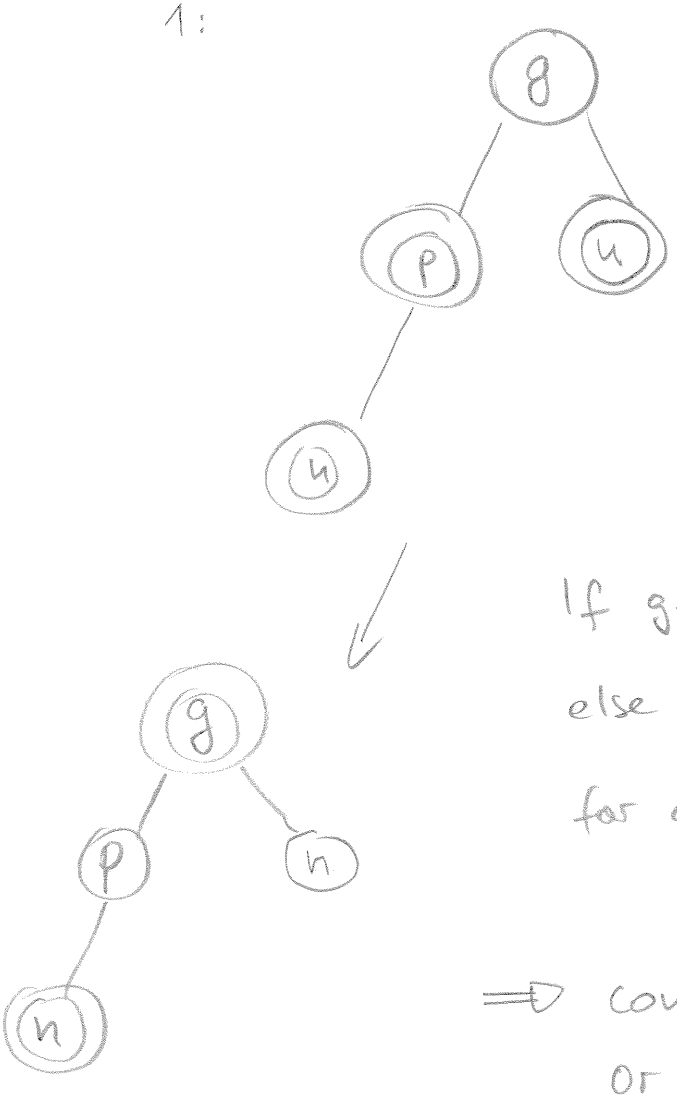
  0:



n.parent is black ✓

Now: n.parent is red ⇒ consider grandparent and uncle

1:



uncle u is red (⇒ g is black)

⇒ color p, u black, color g red

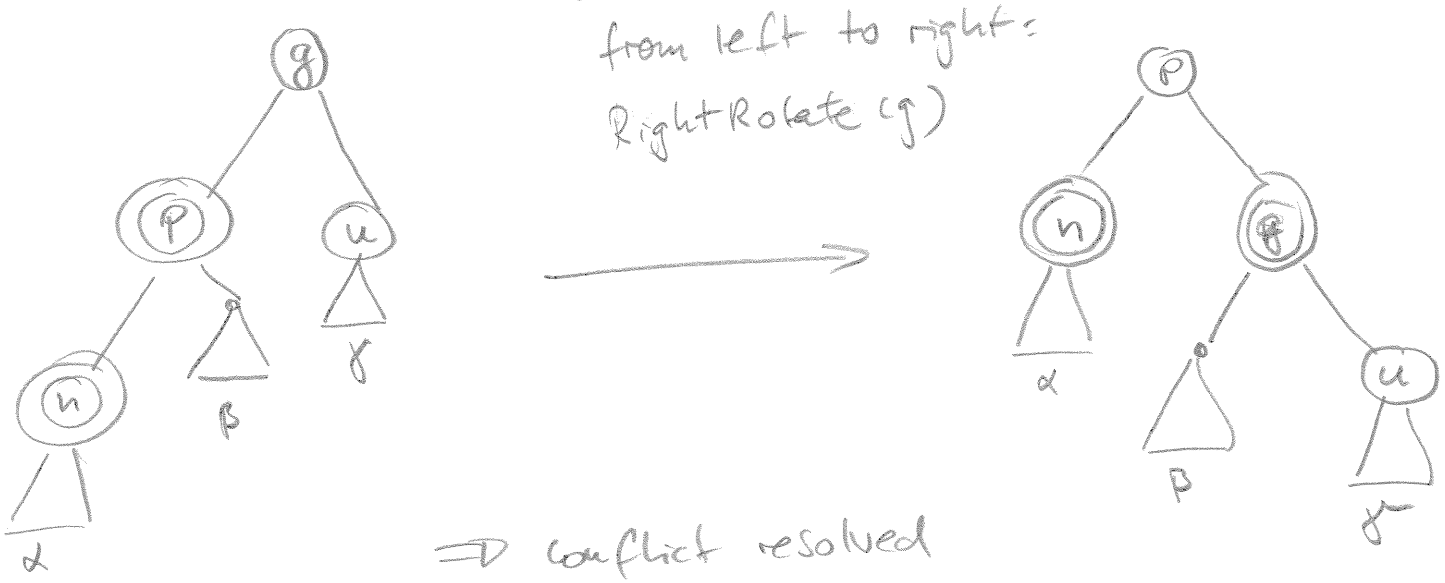
If g.parent is red, then ✓

else fix up conflict for g, g.parent

⇒ conflict resolved or propagated

2: uncle u is black (g is black)

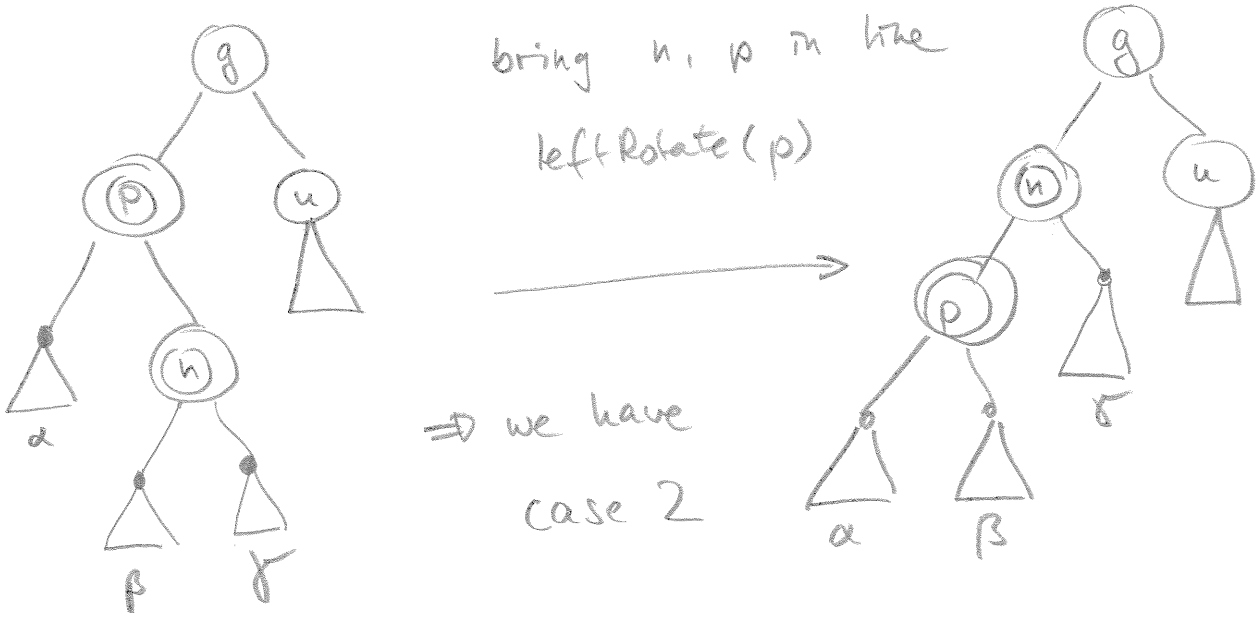
n is left child



⇒ conflict resolved
by restructuring

3: uncle u is black (g is black)

u is right child of p



⇒ we have
case 2

Example

REDSOX CUBS

Deletion

Delete(u)

u has ≤ 1 child $\Rightarrow$ remove u

u has $= 2$ children $\Rightarrow$ v := Succ(u);
copy v.content to
u.content;
remove v

Thus: we always remove a node, say u,
with ≤ 1 children

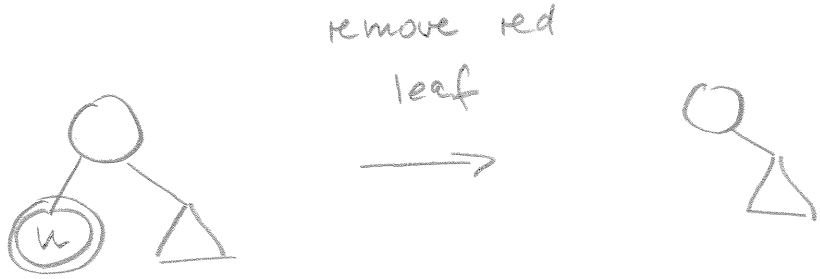
If u.color = red $\Rightarrow$ ✓

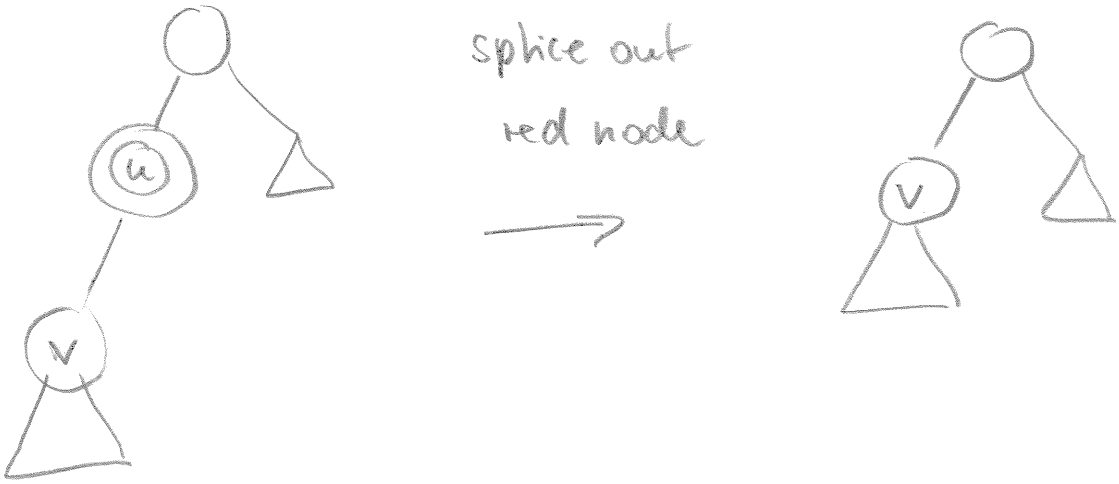
Problem if u.color = black (why?)

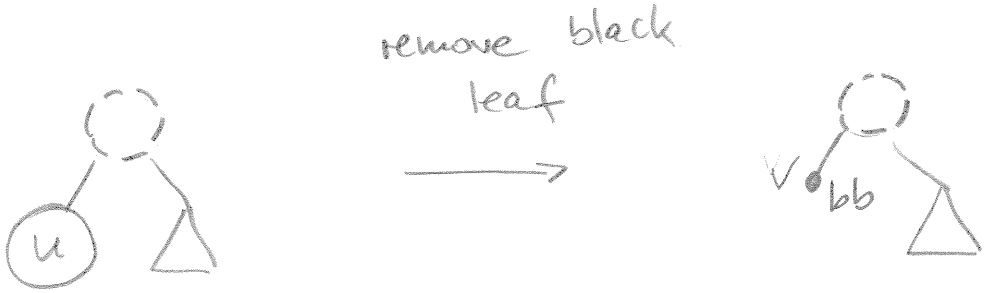
Idea: Let v := replacement of u (possibly leaf)

If v.color = red
then v.color := black ($\Rightarrow$ ✓)
else v.color := double black
propagate the additional black

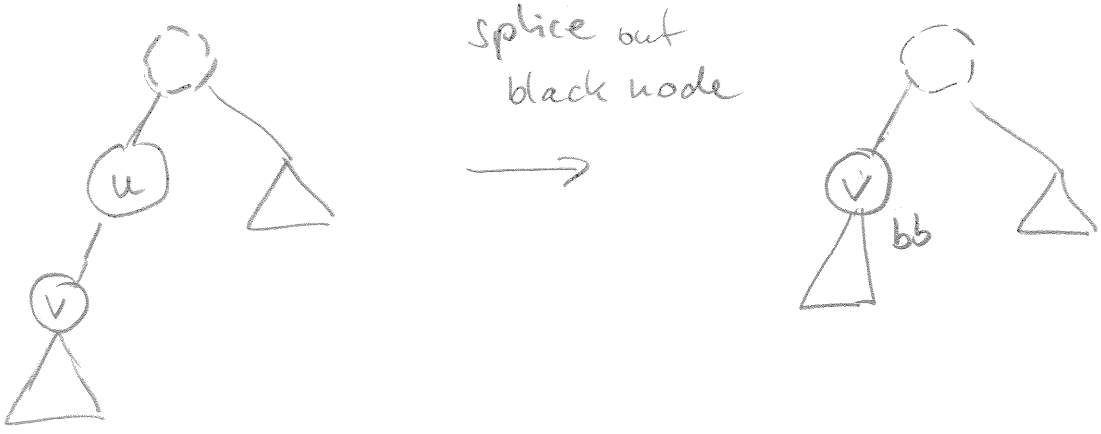
Examples



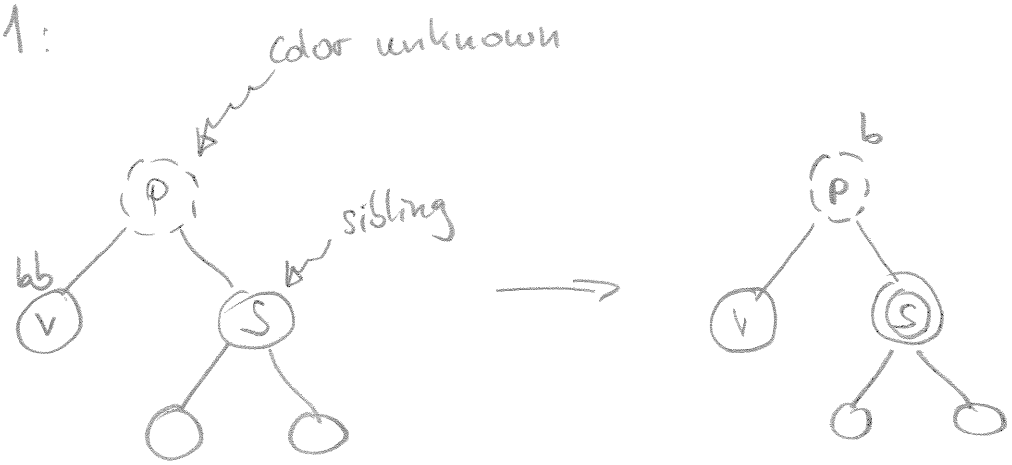




introduce double black nil



Case 1:

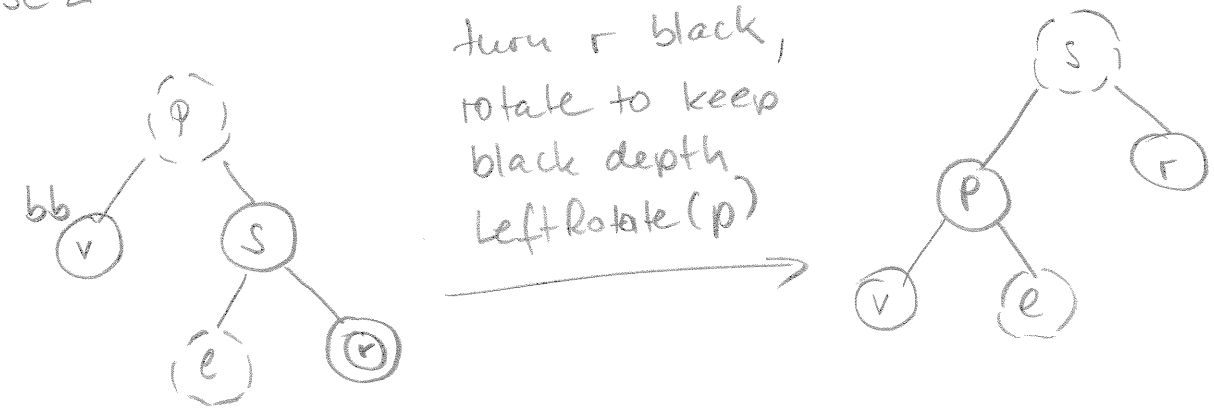


Sibling s is black, both children of s are black

Then: Make parent p "more black", make s red

- p.color = red → p.color = black (⇒ v)
- p.color = black → p.color = double black. (⇒ propagate)

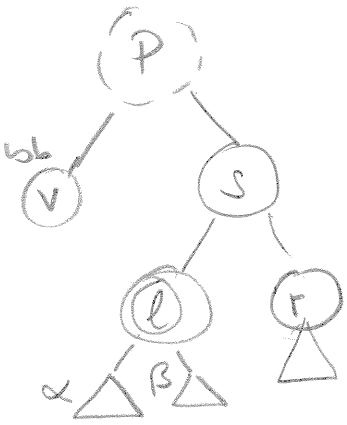
Case 2:



Sibling s is black, right child r of s is red

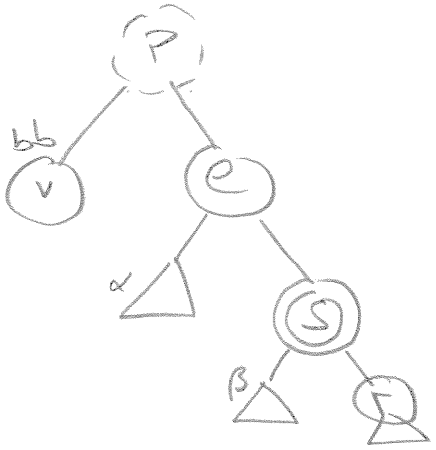
⇒ Conflict resolved by restructuring

Case 3:



Reduce to case 2

RightRotate(s)

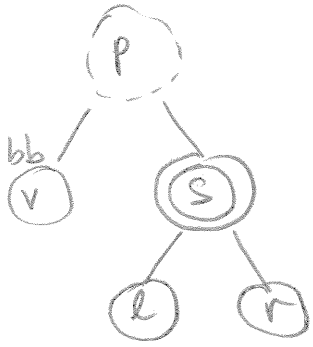


Sibling s is black, left child l is red,

right child r is black

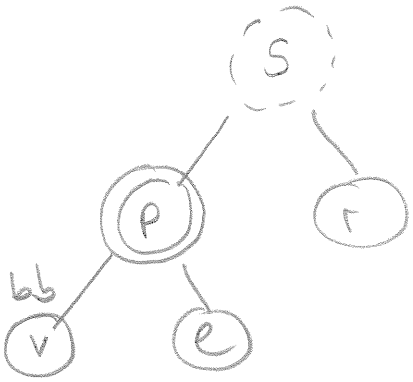
⇒ case 2

Case 4:



Give v a black sibling l

LeftRotate(p)



Sibling s is red ⇒ l, r black

Result: Cas 1, 2, or 3