Homework 4 Due Thursday Oct 7

- CLRS 12-4 (number of binary trees)

- CLRS 13.3-6 (rb insert implementation)

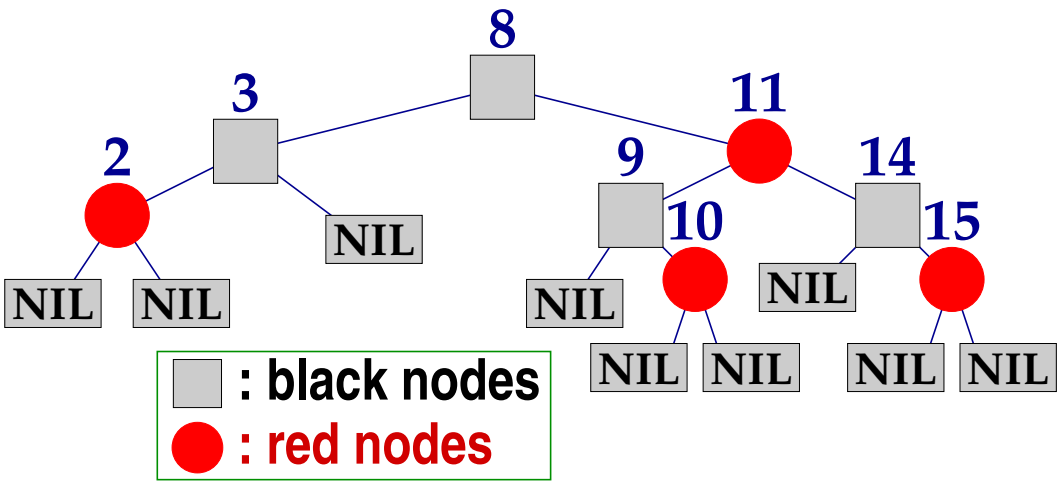
Chapter 13: Red-Black Trees

A **red-black tree** is a node-colored BST. Each node is colored either black or red. The following special rules apply:

1. The root is always black.
2. A **nil** is considered to be black. This means that every non-NIL node has two children.
3. **Black Children Rule:** The children of each red node are black.
4. **Black Height Rule:** For each node v, there exists an integer bh(v) such that each downward path from v to a **nil** has exactly bh(v) black real (i.e. non-**nil**) nodes. Call this quantity the **black height** of v.

We define the black height of an RB tree to be the black height of its root.

An Example


8
3
11
2
9
14
10
15
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
NIL
: black nodes
: red nodes

RB Trees Are Balanced

Lemma A Let T be an RB tree having some $n \geq 1$ nodes. Then the height of T is at most $2 \lg(n+1) - 1$.

Proof Let h be the height of T. Let $v_0, v_1, \ldots, v_{h+1}$ be an arbitrary length h downward path from the root to a **nil**, where v_0 is the root, v_h is the leaf, and v_{h+1} is the **nil**. v_0 is black and v_{h+1} is black. The number of red nodes among $v_1, \ldots, v_h$ is maximized when for all odd i v_i is red. So, the number of red nodes is at most $h/2$. This means that the number of black ones among $v_1, \ldots, v_h$ is at least $h - h/2$. Thus,

$$\mathrm{bh}(T) \geq h/2$$

Proof (cont'd)

If a node has a real black child, then it has another child. This means that the tree contains a complete binary tree of height $\mathsf{bh}(T) - 1$, consisting solely of real black nodes. The number of nodes in the complete binary tree is $2^{\mathsf{bh}(T)} - 1$ black nodes in T. This number is at most n. So, we have

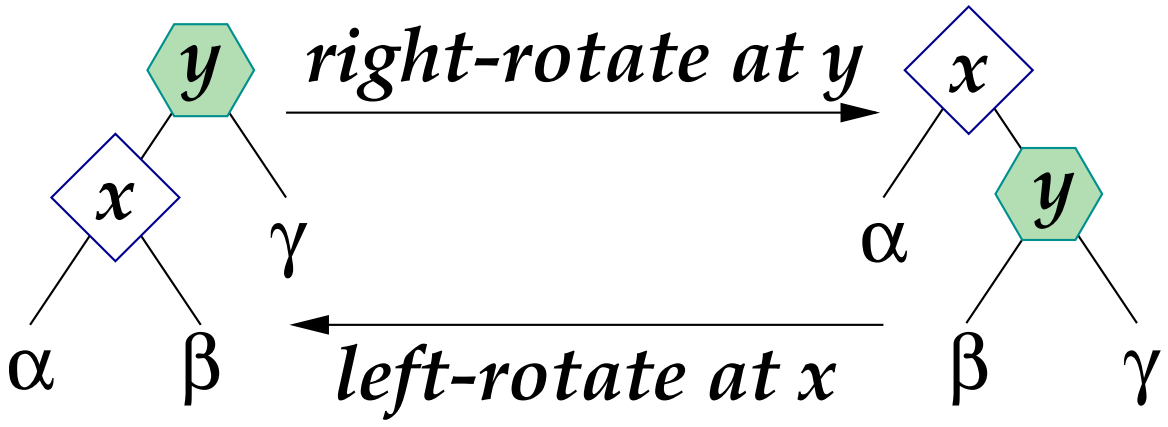
$$\mathsf{lg}(n+1) \geq \mathsf{bh}(T) \geq h/2.$$

By solving this we have $h \leq 2\,\mathsf{lg}(n+1)$. ∎

Operations on RB Trees

We will study two operations, insertion and deletion. The two operations make use of two operations, Left-Rotate and Right-Rotate.

Left-Rotate and Right-Rotate



Rotations do not break the BST-property.

1. Insertion

Suppose that we want to insert a node x into an RB tree T. To do this, we insert x as a red node using the insertion algorithm for BST's and then resolve violation of the coloring rules. The exception is when the tree T is empty. Then we color x black.

Will this operation violate any rule?

Will this operation violate any rule?

Because the node x is red, the Black Children Rule may be violated.

The violation happens when the parent of x is red.

The other rules are not violated.

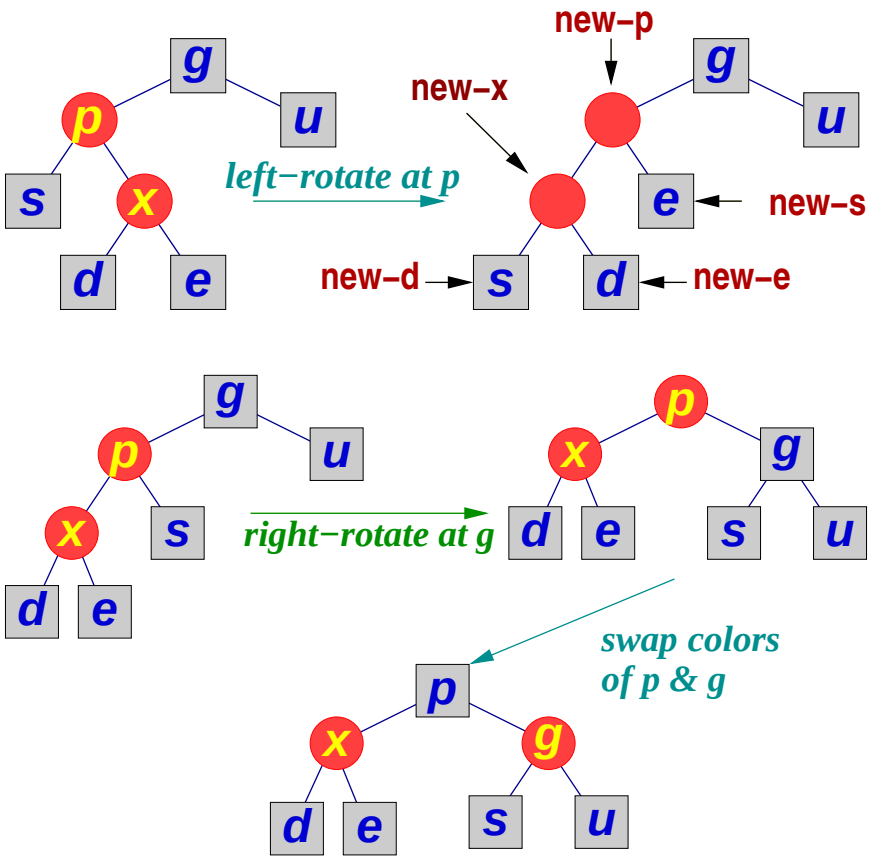
Enforcing the Black Children Rule After Insertion

Let p be the parent of x. Assume p is red; otherwise there is no violation. Since p is red, it cannot be the root. So, let g be the grand parent of x. Let u be the sibling of p. Let s be the sibling of x, which can be **nil**. Since there was no violation before insertion of x, g is black and s is black. However, u can be either red or black. We assume that p is the left child of g. The treatment in the case when p is the right child is similar.

We consider two cases, u is black and u is red.

(Case 1) $color[u] = \mathbf{black}$

First, if x is not the left child of p, then left-rotate at p and swap the role of x and that of p. This preserves bh at g's position. Next, right-rotate at g, then swap the color of p and that of g. Also, at the end the node at g's position is black. So, there is no violation any more.

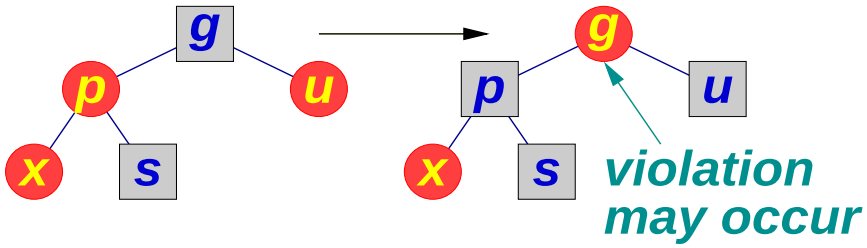


violation resolved

(Case 2) $color[u] = \mathbf{red}$

In this case, we color both p and u black and color g red. This eliminates the violation of the Black Children Rule between p and x, but may introduce violation, which is between g and its parent. So, we may be back to square one, but the location of the violation, if introduced, is two levels closer to the root.

Thus, the bad situation does not repeat more than the height of the tree.


g
p
u
x
s
g
p
u
x
s
violation
may occur

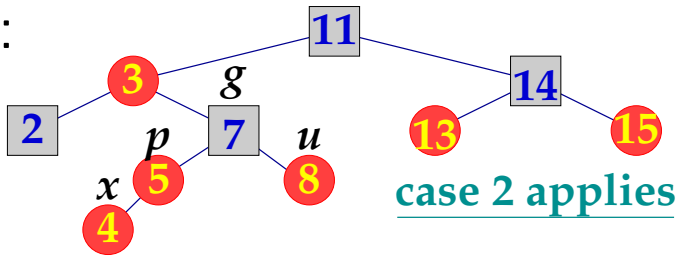
What will happen at the end?

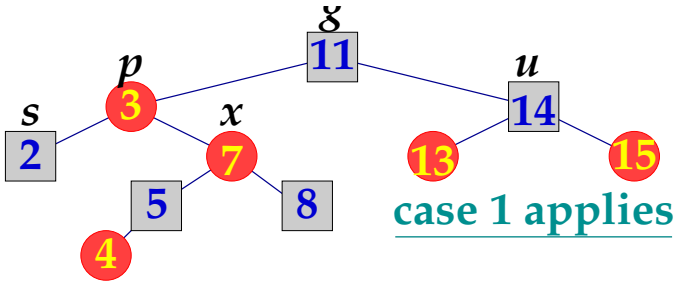
Either Case 1 holds or the resolution for Case 2 does not introduce violation.

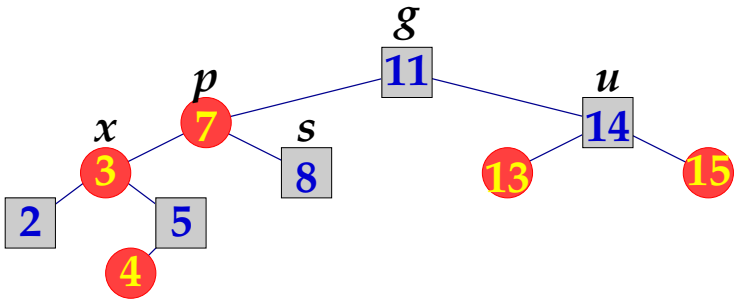
What other condition might be violated?

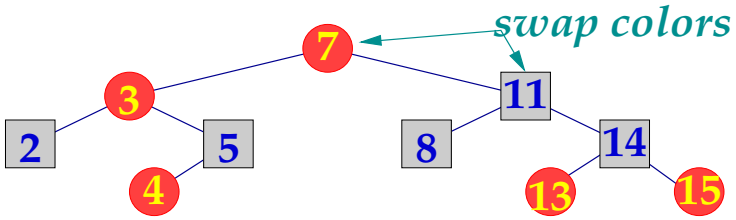
It is when the resolution for Case 2 eliminates violation of the Black Children Rule but turns the root red.

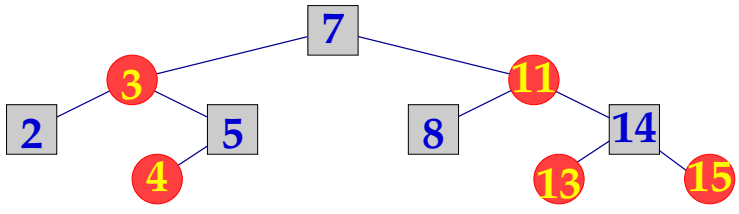
Example 1:


11
3
g
14
2
p
7
u
13
15
x
5
8
4
case 2 applies

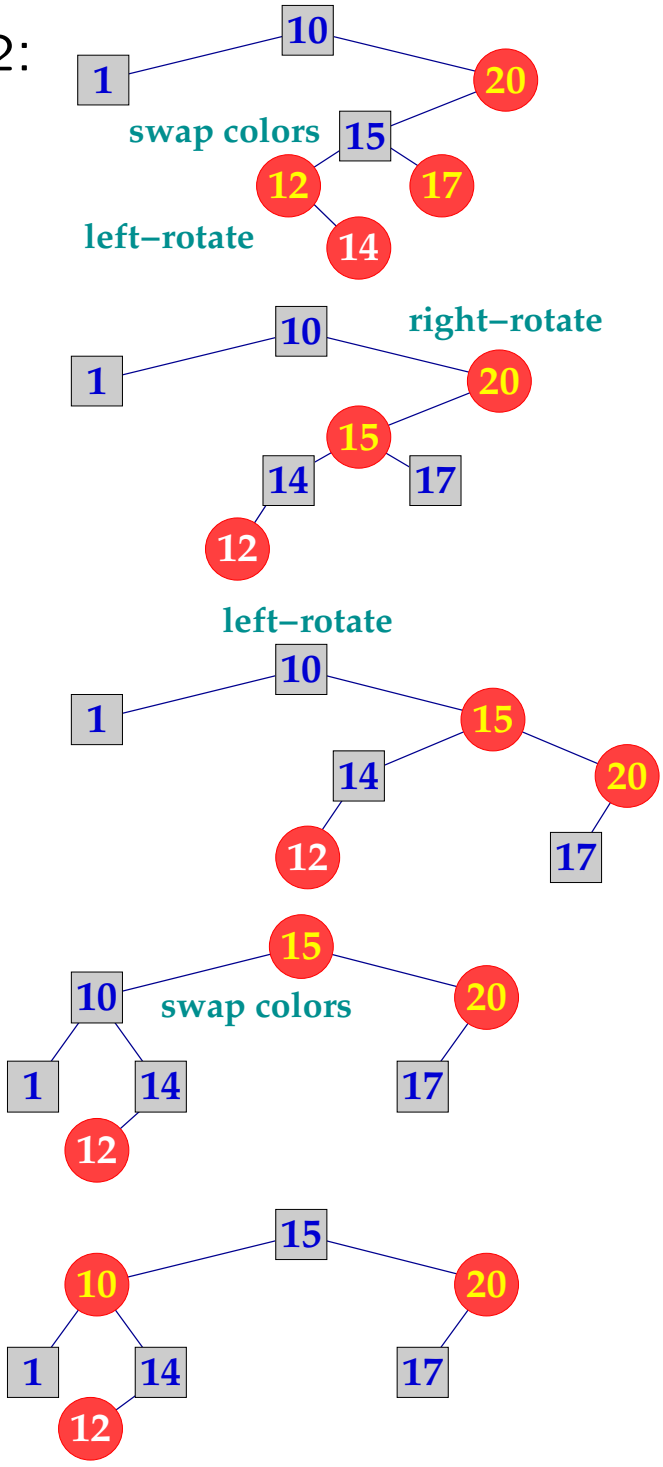

g
11
p
3
u
14
s
x
2
7
13
15
5
8
4
case 1 applies


g
11
p
7
u
14
x
3
s
8
13
15
2
5
4


swap colors
7
3
11
2
5
8
14
4
13
15


7
3
11
2
5
8
14
4
13
15

Example 2:


10
1
20
swap colors
15
12
17
left–rotate
14
10
right–rotate
1
20
15
14
17
12
left–rotate
10
1
15
14
20
12
17
15
10
swap colors
20
1
14
17
12
15
10
20
1
14
17
12

2. Deletion of a node z

We first apply $\mathbf{BST\text{-}Delete}$. In the case when a node is copied to z (the successor of z comes to z's position), color the new one by the color of z.

The deletion routine gives back the pointer, x, to a node where the actual elimination took places. There are two possibilities: (1) There was a leaf at x's position and x is a $\mathbf{nil}$ (2) The node who was at x's position had a unique child and now this unique child is at x's position.

In the latter case, the unique child is red.

Why?

In the latter case, the unique child is red.

Why?

Assume otherwise.

Then the black height of the other subtree, which is a **nil**, is one, while the black height of the unique subtree is at least two.

That means that the Black Height Rule is already violated.

So, the latter case will not create violation.

Thus, we will consider only the case when there was a leaf at x is position.

Furthermore, if the leaf that has been eliminated is red, then the elimination does not introduce violation. So, we assume that the leaf is black.

Resolving the Black Height Rule Violation

Let w be a node. We say that **Few(w)** holds if the following conditions are satisfied:

- The Black Height Rule is violated in the red-black tree.
- T_w, the subtree rooted at w, is a red-black tree without coloring rule violation, except the rule about the color of the root.
- If bh(w) were bh(w) + 1 then the Black-Height Rule would be satisfied for all the nodes outside T_w

Some Properties of Few($\cdot$)

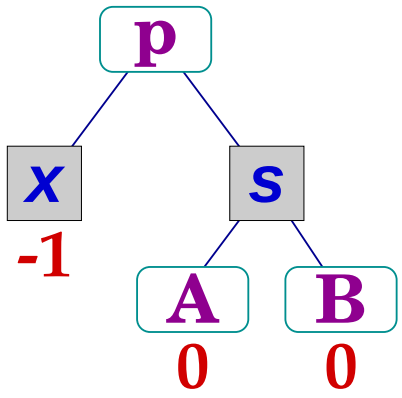
- The condition Few initially holds at the **nil** who's replacing x.
- If Few(w) holds and w is red, then coloring w black resolves violation. So, we will consider the case when w is black.
- If Few(w) holds then w cannot be the root.

Assume that x is the left child of its parent p. The case in which x is the right child can be solved similarly. Let s be x's sibling.

Let A and B be the left child and the right child of s, respectively. These two nodes exist since we've eliminated a real black node.

Here we can preprocess the trees so that the following two conditions are satisfied:

1. **the sibling of x is black** and
2. **if the left child of the sibling of x is red, then the right child of the sibling of x is red.**


p
x
s
-1
A
B
0
0

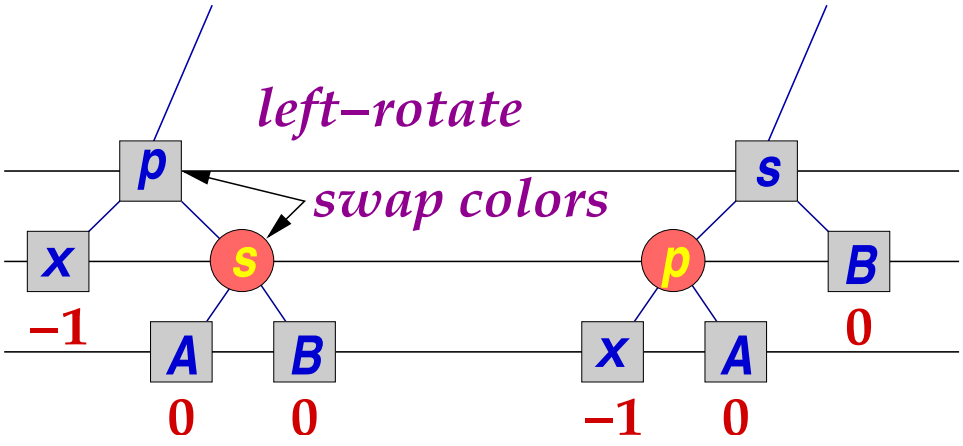
Establishing Property #1

We need to make sure that the sibling of x is black.

Suppose that s is red. Then its parent and its children are all black. That is, p, A, and B are black.

Suppose we left-rotate at p and then swap the colors of p and s. Then,

- the sibling of x is now A and is black;
- the bh-value is unchanged for x, A, and B, so Few(x) still holds;
- the depth of x, i.e. the location of Few, is increased by 1; and
- the parent of x is red.

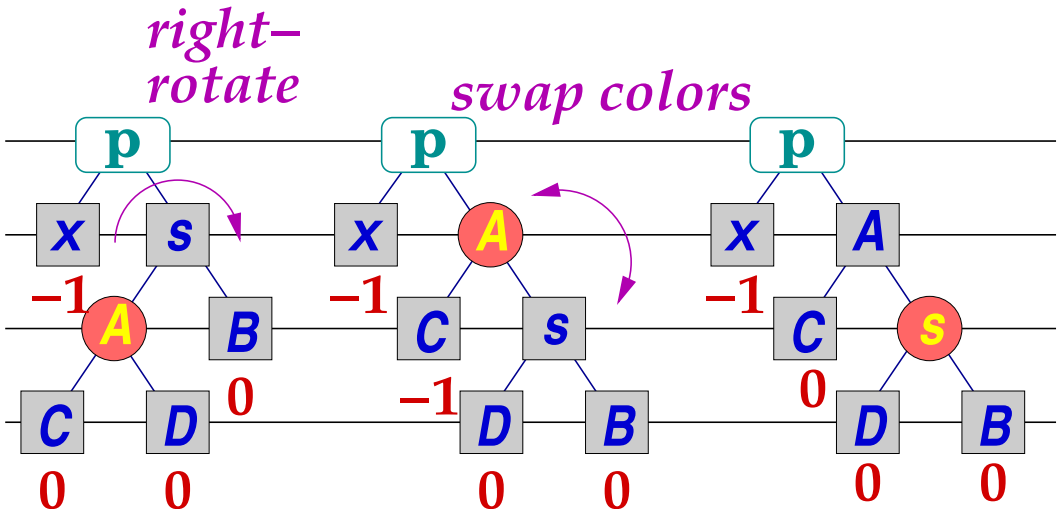

left–rotate
swap colors
p
s
x
s
p
B
–1
A
B
x
A
0
0
0
–1
0

Establishing Property #2

We need to ensure that if the left child of the sibling of x is red, then the right child of the sibling of x is red.

Suppose that A is red and B is black. Let C and D be the children of A. Then these two are black. Suppose we right-rotate at s and swap the colors of A and s. Then,

- A becomes the sibling of x;
- C and s become the left child and the right child of A, respectively;
- the bh-value is preserved for x, C, D, B, and the sibling of x, so Few(x) still holds;
- the left child of x's sibling is now black.

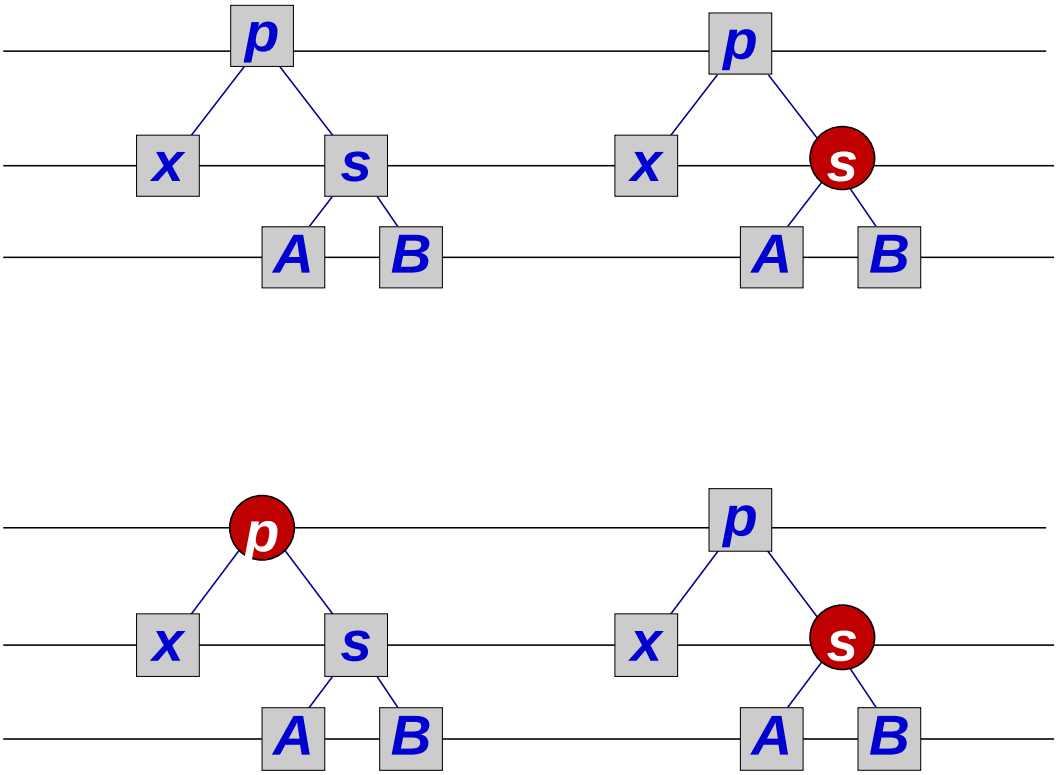

right–
rotate
swap colors
p
x
s
A
B
C
D
–1
0
–1
0
–1
0
0
0
0
0
0
0

Resolution After Enforcing the Conditions

In the case when A and B are both black, we do the following.

If p is black, then we color s red. Then, the downward paths going through s lose one black node. Thus, the location of the Few condition moves one level up to p.

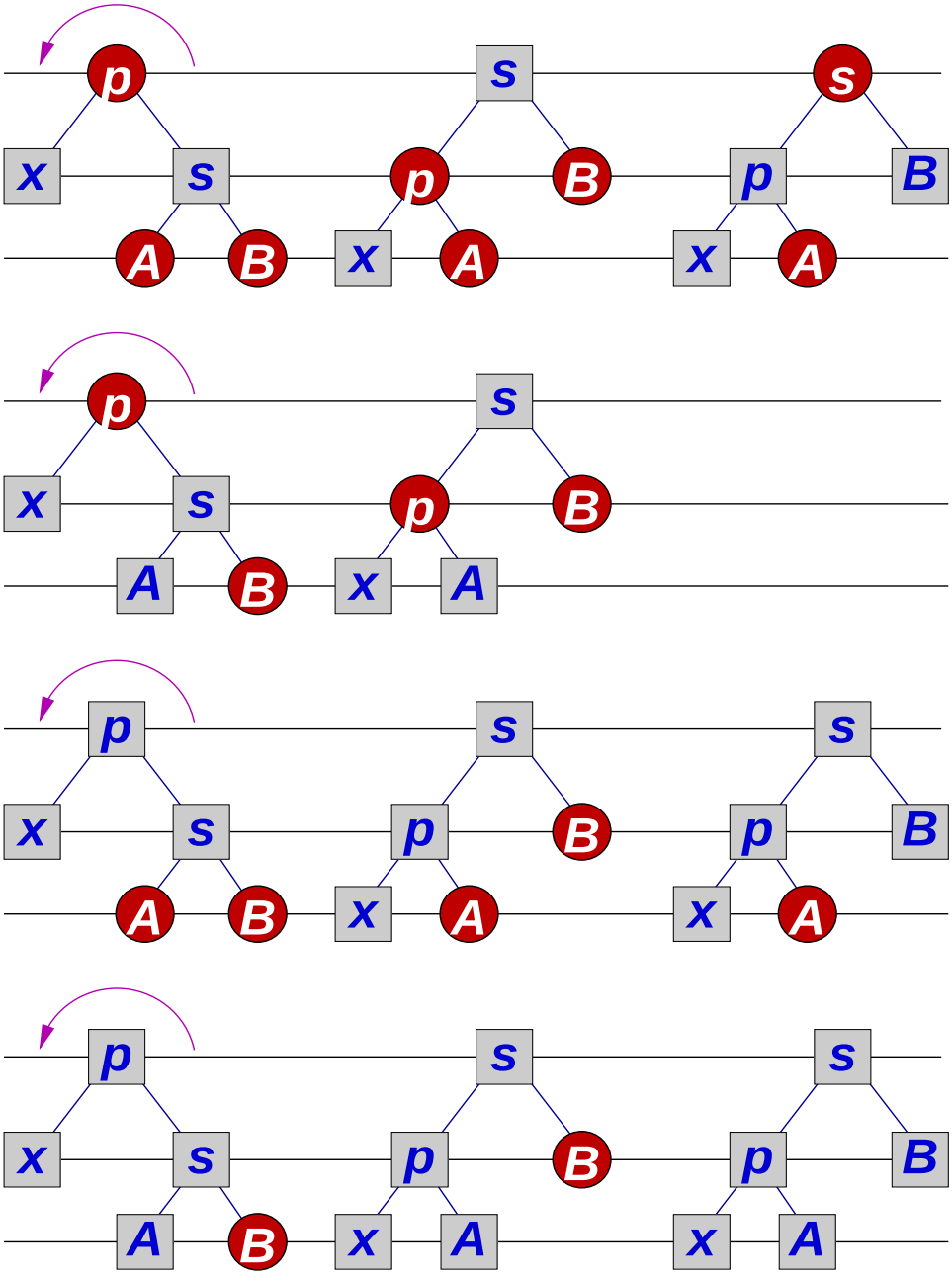
If p is red, we color p black and color s red. Then the Few condition disappears.


p
x
s
A
B
p
x
s
A
B
p
x
s
A
B
p
x
s
A
B

The Remaining Cases

p	A	B	status
B	B	B	done
R	B	B	done
R	R	B	prohibited
B	R	B	prohibited
R	R	R	to be done
R	B	R	to be done
B	R	R	to be done
B	B	R	to be done

We left-rotate at p. Furthermore, in the case of (R,R,R) we color s red and color p and B black and in the case of (B,R,R) and (B,B,R) we color B black.


p
s
s
x
s
p
B
p
B
A
B
x
A
x
A
p
s
x
s
p
B
A
B
x
A
p
s
s
x
s
p
B
p
B
A
B
x
A
x
A
p
s
s
x
s
p
B
p
B
A
B
x
A
x
A