Лекция 1.
Красно-чёрные деревья

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.

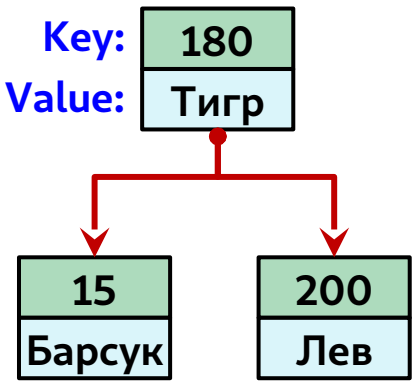
Список основной литературы

- **Кормен Т. Х., Лейзерсон Ч. И., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ — 3-е изд. — М.: Вильямс, 2013**
- Левитин А. В. Алгоритмы: введение в разработку и анализ. — М.: Вильямс, 2006. — 576 с.
- Ахо А. В., Хопкрофт Д., Ульман Д. Д. Структуры данных и алгоритмы. — М.: Вильямс, 2001. — 384 с.
- Курносов М. Г., Берлизов Д. М. Алгоритмы и структуры обработки информации (учебное пособие). — Новосибирск: Параллель, 2019. — 211 с.

Двоичные деревья поиска

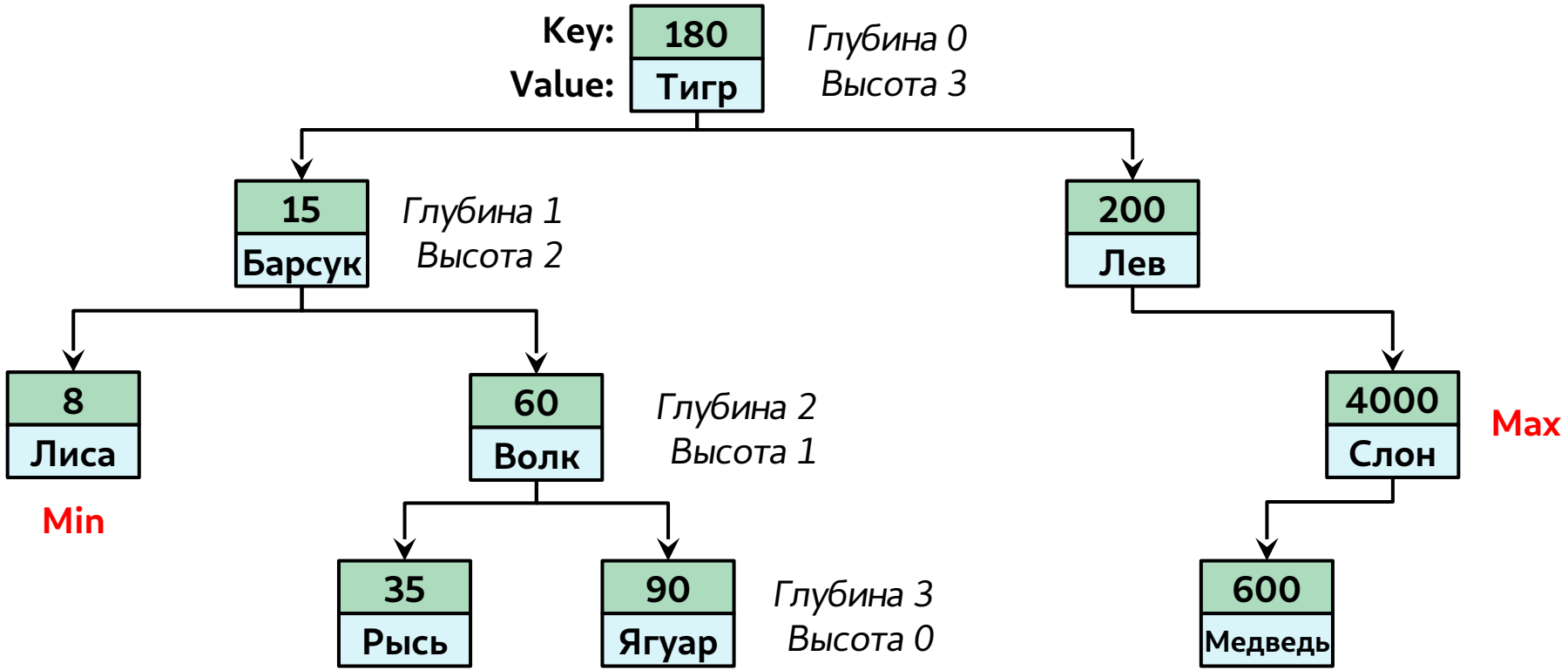
Двоичное дерево поиска (*binary search tree*, *BST*) — это двоичное дерево, в котором:

- каждый узел (*node*) имеет не более двух дочерних узлов (*child nodes*)
- каждый узел содержит ключ (*key*) и значение (*value*)
- ключи всех узлов левого поддерева меньше значения ключа родительского узла
- ключи всех узлов правого поддерева больше значения ключа родительского узла



Двоичные деревья поиска используются для реализации словарей (*map*, *associative array*) и множеств (*set*)

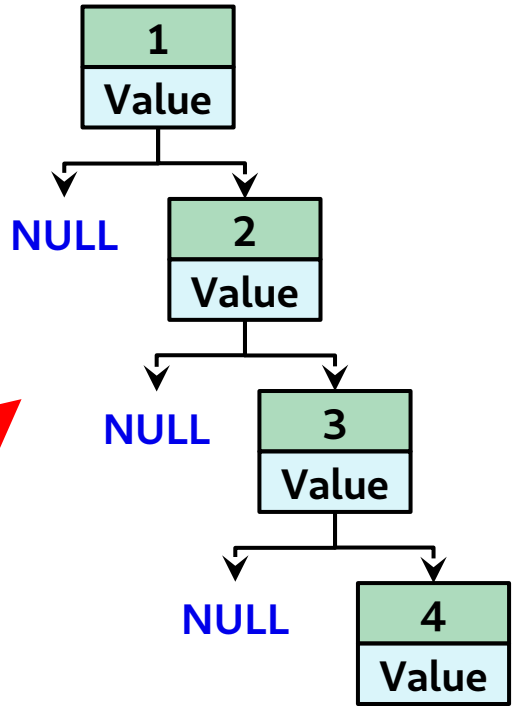
Двоичные деревья поиска



Двоичные деревья поиска

- Операции над двоичным деревом имеют трудоёмкость, пропорциональную высоте ***h*** дерева
- В среднем случае высота дерева ***O*(log*n*)**
- В худшем случае элементы добавляются по возрастанию (убыванию) ключей — дерево вырождается в список длины ***O*(*n*)**

bstree_add(1, value)
bstree_add(2, value)
bstree_add(3, value)
bstree_add(4, value)



Сбалансированные деревья поиска

Сбалансированное дерево поиска (*self-balancing binary search tree*) — дерево поиска,
в котором высота поддеревьев любого узла различается не более, чем на заданную константу *k*

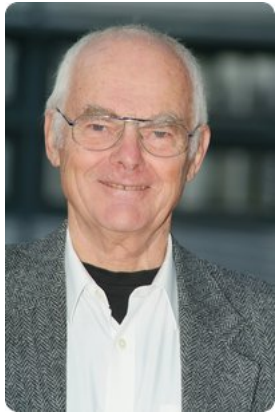
Сбалансированные деревья поиска:

- **Красно-чёрные деревья (*red-black trees*)**
- АВЛ-деревья (*AVL trees*)
- 2-3-деревья (*2-3 trees*)
- B-деревья (*B-trees*)
- AA trees
- ...

Красно-чёрные деревья (*red-black trees*)

Автор: Rudolf Bayer

Technical University of Munich, Germany, 1972



В работе автора дерево названо «symmetric binary B-tree»

В работе Р. Седжвика (1978) дано современное название «red-black tree»

[1] Rudolf Bayer. Symmetric binary B-Trees: Data structure and maintenance algorithms // Acta Informatica. — 1972. — Vol. 1, No. 4 — pp. 290-306.

[2] Guibas L., Sedgewick R. A Dichromatic Framework for Balanced Trees // Proc. of the 19th Annual Symposium on Foundations of Computer Science, 1978. — pp. 8-21.

Красно-чёрные деревья (*red-black trees*)

Операция	Средний случай (average case)	Худший случай (worst case)
Add(*key*, *value*)	*O*(log*n*)	*O*(log*n*)
Lookup(*key*)	*O*(log*n*)	*O*(log*n*)
Remove(*key*)	*O*(log*n*)	*O*(log*n*)
Min	*O*(log*n*)	*O*(log*n*)
Max	*O*(log*n*)	*O*(log*n*)

Сложность по памяти (space complexity): *O*(*n*)

Применение красно-чёрных деревьев

GNU libstdc++ (`/usr/include/c++/bits`)

`std::map, std::multimap,`
`std::set, std::multiset`

LLVM libc++

`std::map, std::set`

Java

`java.util.TreeMap, java.util.TreeSet`

Microsoft .NET 4.5 Framework Class Library

`SortedDictionary, SortedSet`

Ядро Linux

https://www.kernel.org/doc/Documentation/rbtree.txt
https://github.com/torvalds/linux/blob/master/tools/lib/rbtree.c

- CFS scheduler: процессы хранятся в rbtree
- Virtual memory areas (VMAs)
- High-resolution timer (organize outstanding timer requests)
- Ext3 filesystem tracks directory entries
- epoll file descriptors, cryptographic keys, ...

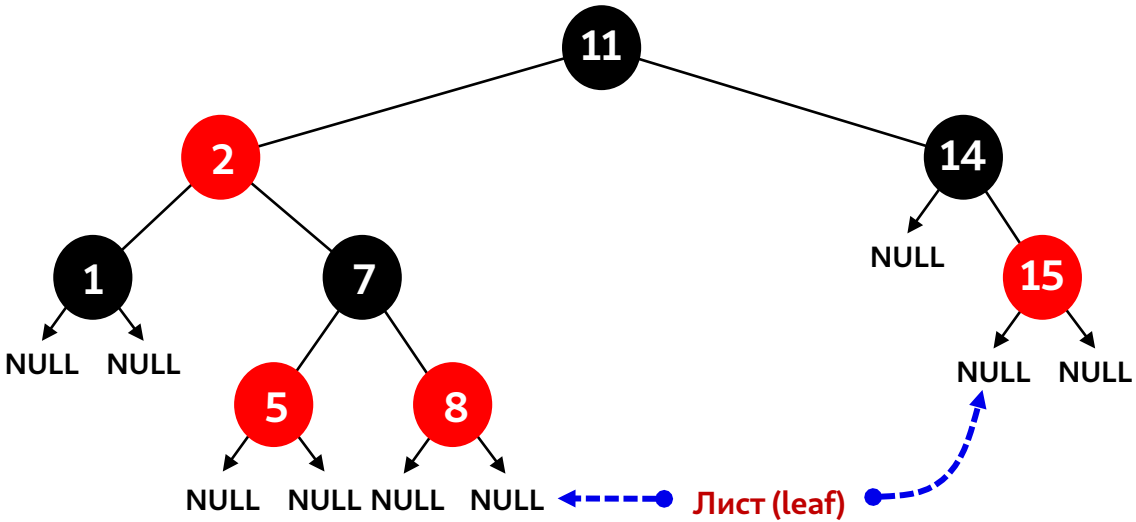
Красно-чёрные деревья

Красно-чёрное дерево (*red-black tree*, *RB-tree*) — это бинарное дерево поиска, для которого выполняются красно-чёрные свойства (red-black properties):

1. Каждый узел дерева является либо красным (red), либо чёрным (black)
2. Корень дерева является чёрным узлом
3. Каждый лист дерева (NULL) является чёрным узлом
4. У красного узла оба дочерних узла — чёрные
5. У любого узла все пути от него до листьев, являющихся его потомками, содержат одинаковое количество чёрных узлов

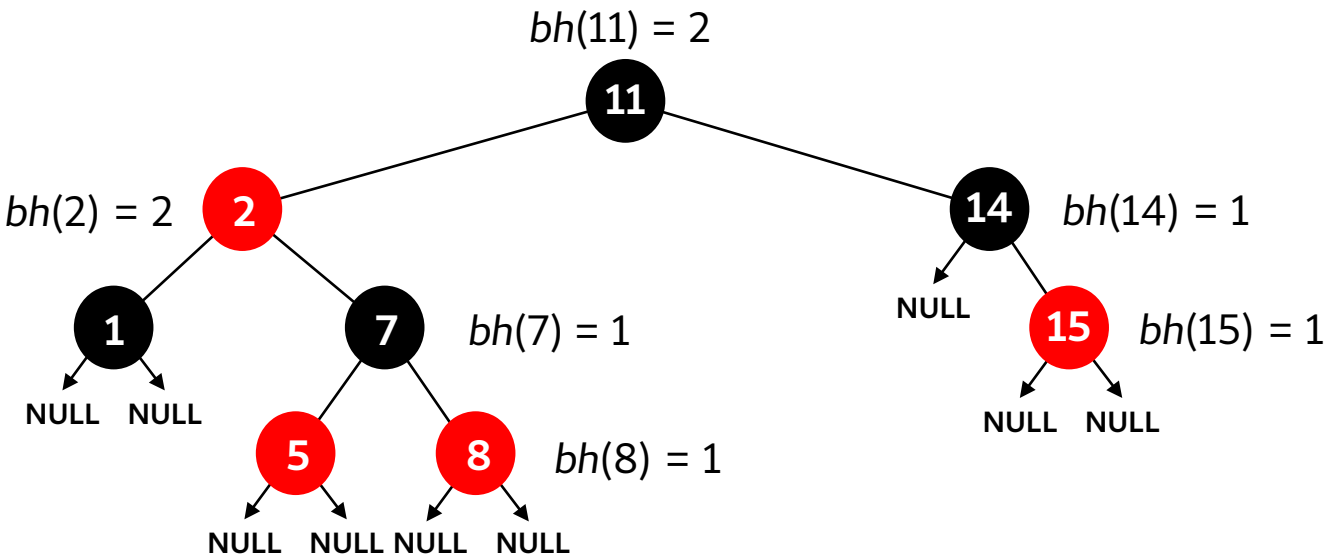
Красно-чёрные деревья

1. Каждый узел дерева является либо красным (red), либо чёрным (black)
2. Корень дерева является чёрным узлом
3. Каждый лист дерева (NULL) является чёрным узлом
4. У красного узла оба дочерних узла — чёрные
5. У любого узла все пути от него до листьев, являющихся его потомками, содержат одинаковое количество чёрных узлов



Красно-чёрные деревья

- Чёрная высота ***bh(x)*** узла (black height) — это количество чёрных узлов на пути от узла x (не считая его) до листа
- Чёрная высота дерева — это чёрная высота его корня



Высота красно-чёрного дерева

Лемма. Красно-чёрное дерево с n внутренними узлами имеет высоту, не превышающую $\mathbf{2\log_2(n+1)}$

Доказательство [CLRS, с. 342]

- Покажем по индукции, что любое поддерево с вершиной в узле x содержит не менее $2^{bh(x)} - 1$ внутренних узлов

Базис индукции

- Если высота h узла x равна 0, то узел x — это лист (NULL), а его поддерево содержит не менее $2^{bh(x)} - 1 = 2^0 - 1 = 0$ внутренних узлов

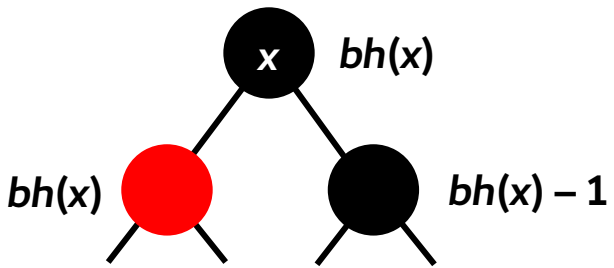
Высота красно-чёрного дерева

Лемма. Красно-чёрное дерево с n внутренними узлами имеет высоту, не превышающую $2\log_2(n+1)$

Доказательство (продолжение)

Шаг индукции

- Рассмотрим узел x, который имеет положительную высоту $h(x)$ — внутренний узел с двумя потомками
- Каждый дочерний узел имеет черную высоту либо $bh(x)$, либо $bh(x) - 1$, в зависимости от его цвета



Высота красно-чёрного дерева

Лемма. Красно-чёрное дерево с n внутренними узлами имеет высоту, не превышающую $2\log_2(n+1)$

Доказательство (продолжение)

- Поскольку высота потомка x меньше высоты узла x, мы можем использовать предположение индукции и сделать вывод о том, что каждый потомок x имеет как минимум $2^{bh(x)-1} - 1$ внутренних узлов
- Тогда все дерево с корнем в узле x содержит не менее

$$(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$$

внутренних узлов.

Высота красно-чёрного дерева

Лемма. Красно-чёрное дерево с n внутренними узлами имеет высоту, не превышающую $\mathbf{2\log_2(n+1)}$

Доказательство (окончание)

- Получили, что в дереве x число n внутренних узлов

$$n \geq 2^{bh(x)} - 1$$

- По *свойству 4* как минимум половина узлов на пути от корня к листу чёрные, тогда

$$bh(x) \geq h(x) / 2$$

- Следовательно:

$$n \geq 2^{h(x)/2} - 1$$

$$\log_2(n+1) \geq h(x) / 2$$

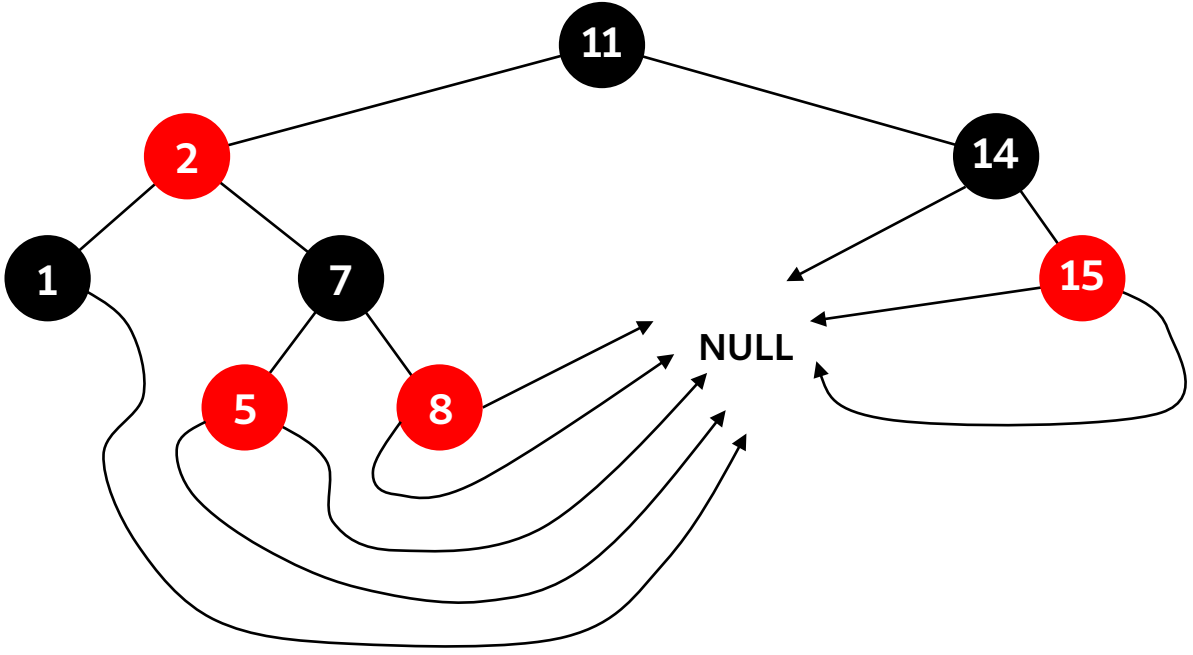
$$\mathbf{h(x) \leq 2\log_2(n+1)}$$

Структура узла красно-чёрного дерева

- ***node.parent*** — указатель на родительский узел
- ***node.left*** — указатель на левый дочерний узел
- ***node.right*** — указатель на правый дочерний узел
- ***node.color*** — цвет узла (RED, BLACK)
- ***node.key*** — ключ
- ***T.root*** — указатель на корень дерева

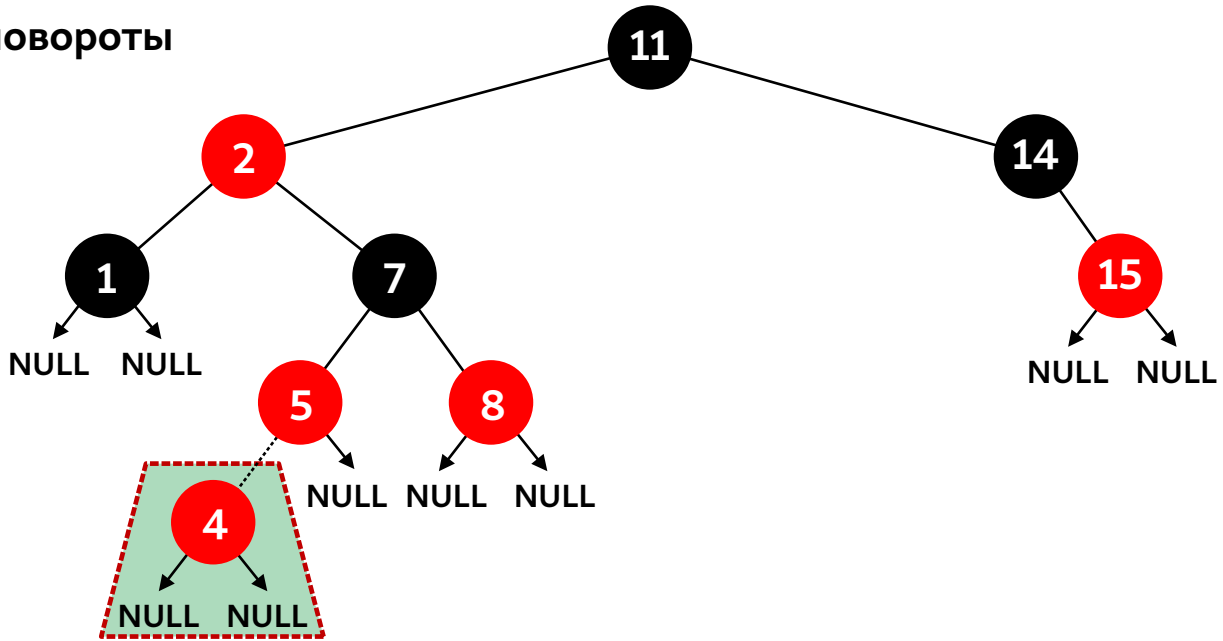
Структура узла красно-чёрного дерева

Для удобства будем считать, что все листья (узлы NULL) — это указатели на один и тот же ограничивающий узел чёрного цвета *T.null* (*sentinel node*)



Добавление узла в красно-чёрное дерево

1. Находим лист для вставки нового элемента
2. Создаем элемент и окрашиваем его в **красный цвет**
3. Перекрашиваем узлы и выполняем **повороты**



Добавили узел 4

Нарушено свойство 4 — красный узел 5 должен иметь два чёрных дочерних узла

Добавление узла в красно-чёрное дерево

```
function RBTree_Add(T, key, value)
    tree = T.root
    while tree != null do
        if key < tree.key then
            tree = tree.left
        else if key > tree.key then
            tree = tree.right
        else
            return  /* Ключ уже присутствует в дереве */
        end if
    end while

    node = CreateNode(key, value)
```

Добавление узла в красно-чёрное дерево

```
    if T.root = NULL then          /* Пустое дерево */
        T.root = node
    else
        if key < tree.parent.key then
            tree.parent.left = node
        else
            tree.parent.right = node
        end if
    end if

    node.color = RED
    RBTree_Add_Fixup(T, node)
end function
```

Нарушение свойств красно-чёрного дерева

Какие свойства красно-чёрного дерева могут быть нарушены при добавлении нового узла?

1. Каждый узел является красным, либо чёрным
2. Корень дерева является чёрным
3. Каждый лист дерева (NULL) является чёрным
4. У красного узла оба дочерних узла являются чёрными
5. У любого узла все пути от него до листьев (его потомков) содержат одинаковое число чёрных узлов

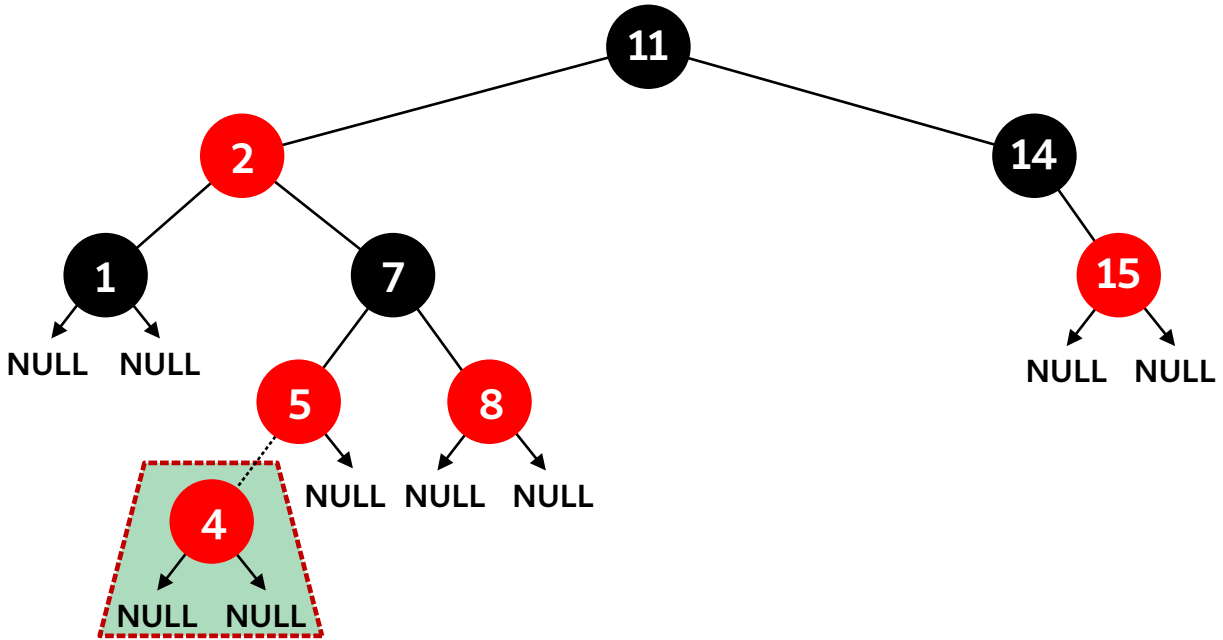
Нарушение свойств красно-чёрного дерева

Какие свойства красно-чёрного дерева могут быть нарушены при добавлении нового узла?

1. Каждый узел является красным, либо чёрным — **выполняется**
2. **Корень дерева является чёрным — может быть нарушено**
3. Каждый лист дерева (NULL) является чёрным — **выполняется**
4. **У красного узла оба дочерних узла являются чёрными — может быть нарушено**
5. У любого узла все пути от него до листьев (его потомков) содержат одинаковое число чёрных узлов — **выполняется (узел замещает чёрный NULL, но сам имеет два чёрных дочерних NULL)**

Восстановление свойств красно-чёрного дерева

- Возможны **6 случаев** нарушения свойства красно-чёрного дерева (3 из них симметричны другим)
- Восстановление свойств начинаем с нового элемента и продвигаемся вверх к корню дерева

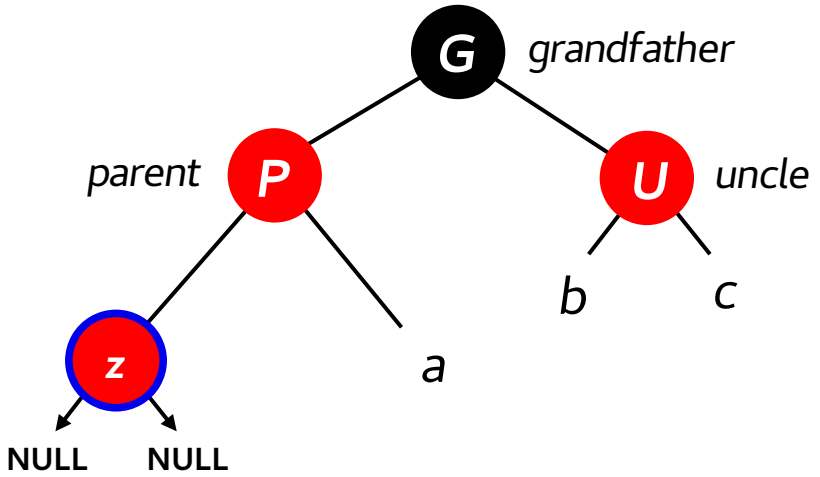


Восстановление свойств красно-чёрного дерева

Случай 1: «дядя» *U* (*uncle*) нового узла *z* — красный

- Узел *z* красный
- Дядя *U* узла *z* — красный
- Узел *P*, корень левого поддерева своего родителя *G* — красный

Нарушено свойство 4 у узла *P*



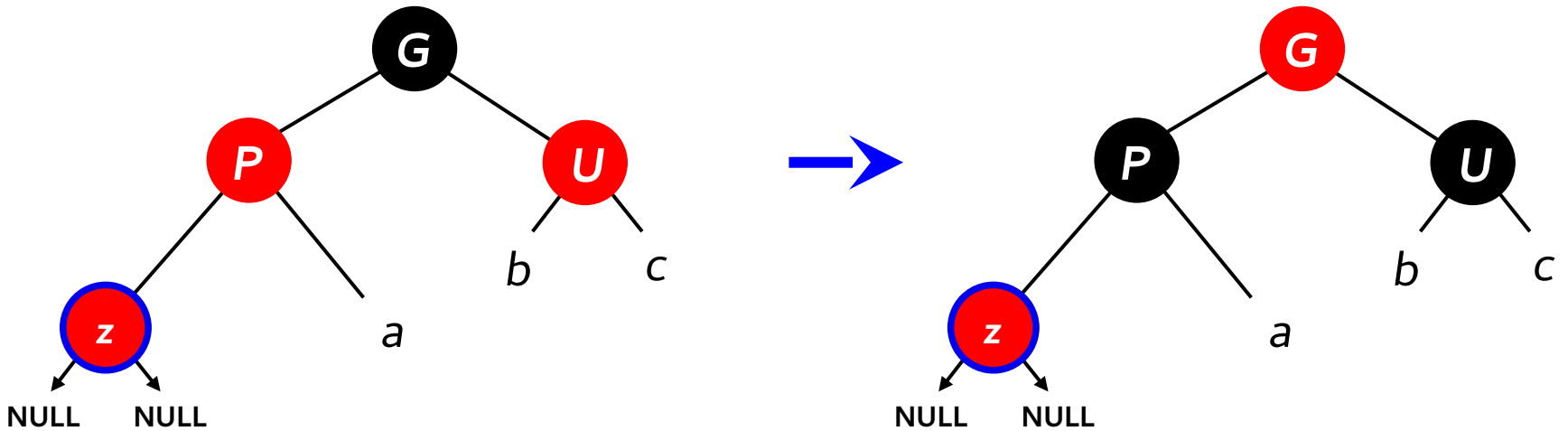
Восстановление свойств красно-чёрного дерева

Случай 1: «дядя» *U* (*uncle*) нового узла *z* — красный

- Узел *z* красный
- Дядя *U* узла *z* — красный
- Узел *P*, корень левого поддерева своего родителя *G* — красный

Перекрашиваем узлы:

- *P* — чёрный (*z.p*)
- *U* — чёрный (*z.p.p.right*)
- *G* — красный (*z.p.p*)

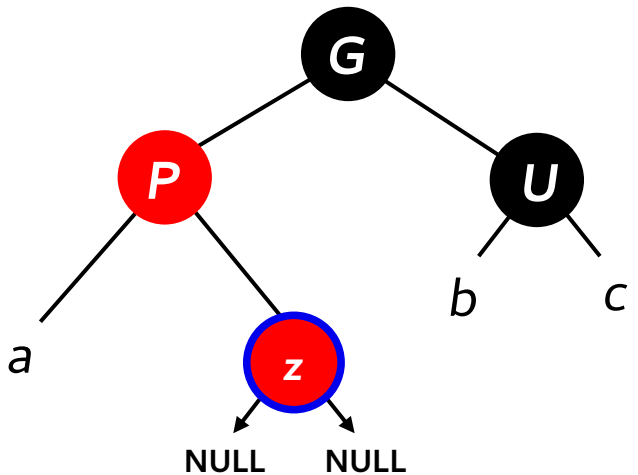


Восстановление свойств красно-чёрного дерева

Случай 2:

- Дядя *U* узла *z* — чёрный
- Узел *z*, правый потомок *P* — красный
- Родительский узел *P* узла *z* — красный
- Узел *P* — корень левого поддерева своего родителя *G*

Нарушено свойство 4 у узла *P*

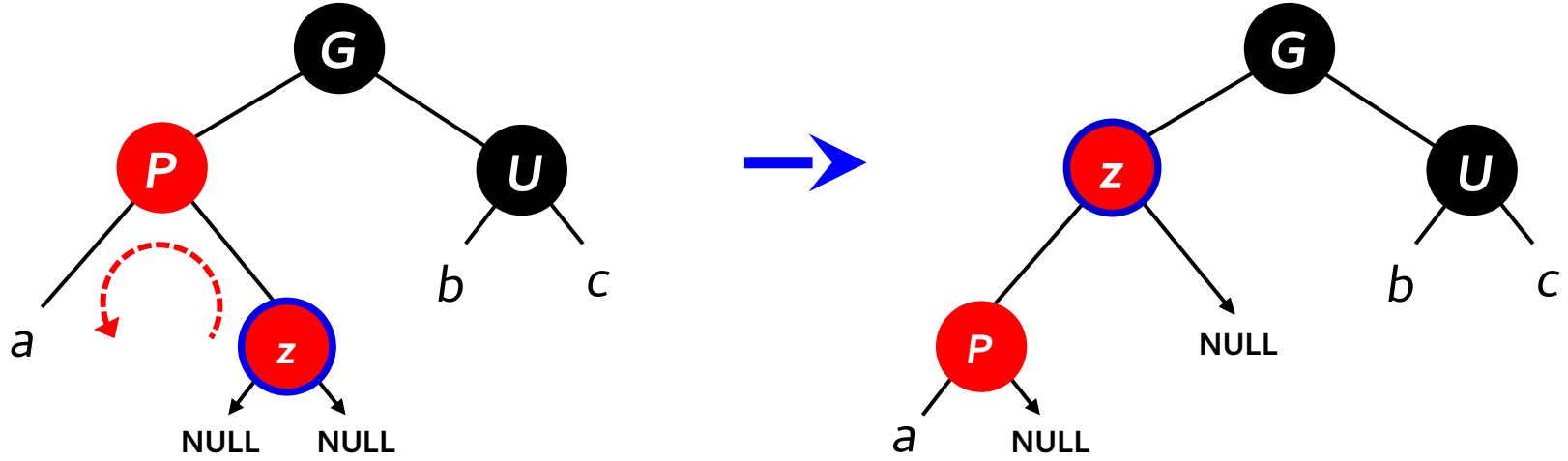


Восстановление свойств красно-чёрного дерева

Случай 2:

- Дядя *U* узла *z* — чёрный
- Узел *z*, правый потомок *P* — красный
- Родительский узел *P* узла *z* — красный
- Узел *P* — корень левого поддерева своего родителя *G*

Путём поворота дерева *P* влево переходим к **случаю 3**



Повороты красно-чёрного дерева

$$T_{LeftRotate} = T_{RightRotate} = O(1)$$

Повороты красно-чёрного дерева

```
function LeftRotate(x)
    y = x.right
    x.right = y.left
    if y.left != NULL then
        y.left.parent = x
    y.parent = x.parent
    if x = x.parent.left then
        x.parent.left = y
    else
        x.parent.right = y
    y.left = x
    x.parent = y
end function
```

```
function RightRotate(x)
    y = x.left
    x.left = y.right
    if y.right != NULL then
        y.right.parent = x
    y.parent = x.parent
    if x = x.parent.left then
        x.parent.left = y
    else
        x.parent.right = y
    y.right = x
    x.parent = y
end function
```

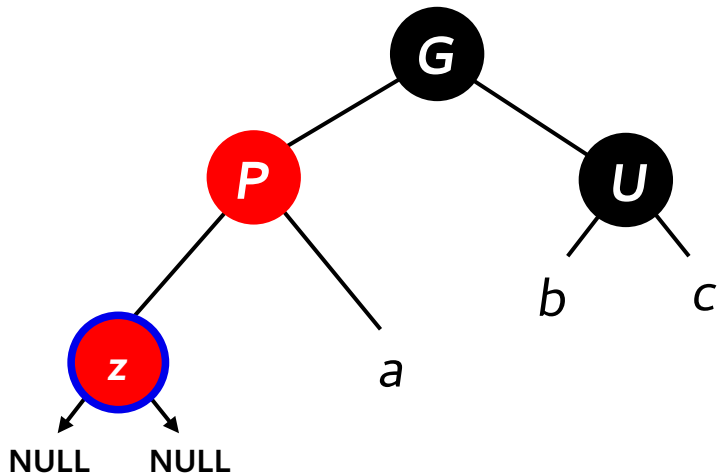
$$T_{LeftRotate} = T_{RightRotate} = O(1)$$

Восстановление свойств красно-чёрного дерева

Случай 3:

- Дядя *U* узла *z* — чёрный
- Узел *z*, левый потомок *P* — красный
- Родительский узел *P* узла *z* — красный
- Узел *P* — корень левого поддерева своего родителя *G*

Нарушено свойство 4 у узла *P*



Восстановление свойств красно-чёрного дерева

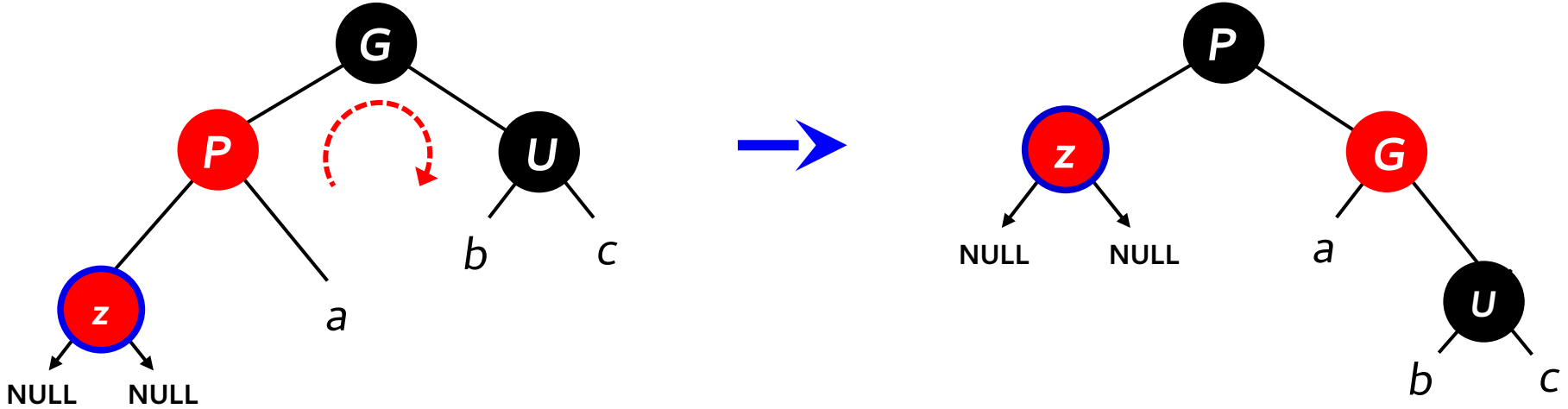
Случай 3:

- Дядя *U* узла *z* — чёрный
- Узел *z*, левый потомок *P* — красный
- Родительский узел *P* узла *z* — красный
- Узел *P* — корень левого поддерева своего родителя *G*

1. Перекрашиваем вершины:

- *P* — чёрный
- *G* — красный

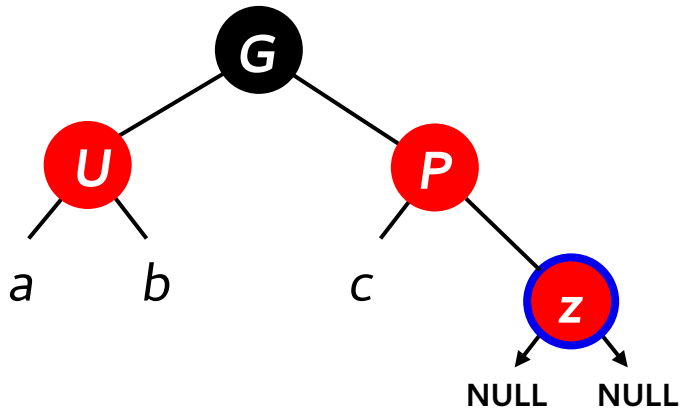
2. Поворачиваем дерево *G* вправо



Восстановление свойств красно-чёрного дерева

Случаи 4, 5 и 6 симметричны случаям 1, 2 и 3

- Узел *P* — это корень *правого* поддерева своего родителя *G*
- Узел *z* красный
- Родительский узел *P* узла *z* красный
- Узел *U* *чёрный* или *красный*
- Узел *z* — *левый* или *правый* дочерний элемент *P*



Восстановление свойств красно-чёрного дерева

```
function RBTree_Add_Fixup(T, z)
    while z.parent.color = RED do
        if z.parent = z.parent.parent.left then
            /* z belongs to left subtree of G */
            y = z.parent.parent.right;   /* Uncle */
            if y.color = RED then
                /* Case 1 */
                z.parent.color = BLACK
                y.color = BLACK
                z.parent.parent.color = RED
                z = z.parent.parent
            else
                if z = z.parent.right then
                    /* Case 2 --> Case 3 */
                    z = z.parent
                    RBTree_RotateLeft(T, z)
                end if
```

Восстановление свойств красно-чёрного дерева

```
                /* Case 3 */
                z.parent.color = BLACK
                z.parent.parent.color = RED
                RBTree_RotateRight(T, z.parent.parent)
            end if
        else
            /* z belongs to right subtree of G */
            /* ... */
        end if
    end while
    T.root.color = BLACK
end function
```

$T_{AddFixup} = O(\log n)$

- Цикл **while** повторно выполняется только в случае 1, указатель *z* перемещается вверх по дереву на 2 уровня. Количество итераций цикла в худшем случае — $O(\log n)$.
- Никогда не выполняется больше двух поворотов (в случаях 2 и 3 цикл **while** завершает работу)

Дальнейшее чтение

1. Изучить алгоритм удаления узлов из красно-чёрного дерева **[CLRS 3ed., с. 356]**

2. Познакомиться с устройством **АВЛ-деревьев**:
   - Левитин А. В. Алгоритмы: введение в разработку и анализ. — М.: Вильямс, 2006. — 576 с. **(С. 267-271)**
   - Вирт Н. Алгоритмы и структуры данных. — М.: Мир, 1989. — 360 с. **(С. 272-286)**

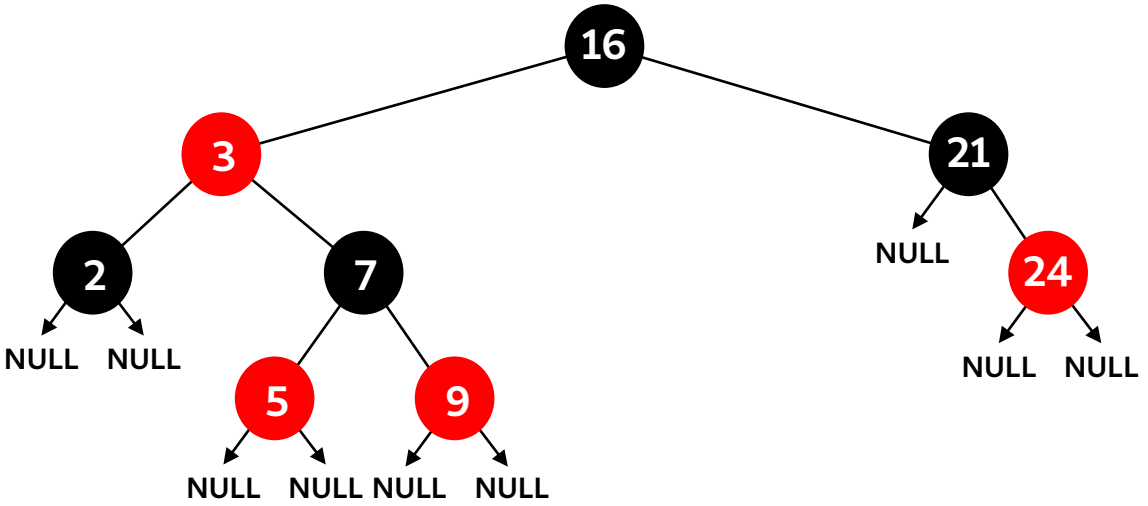
ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Осенний семестр, 2021 г.

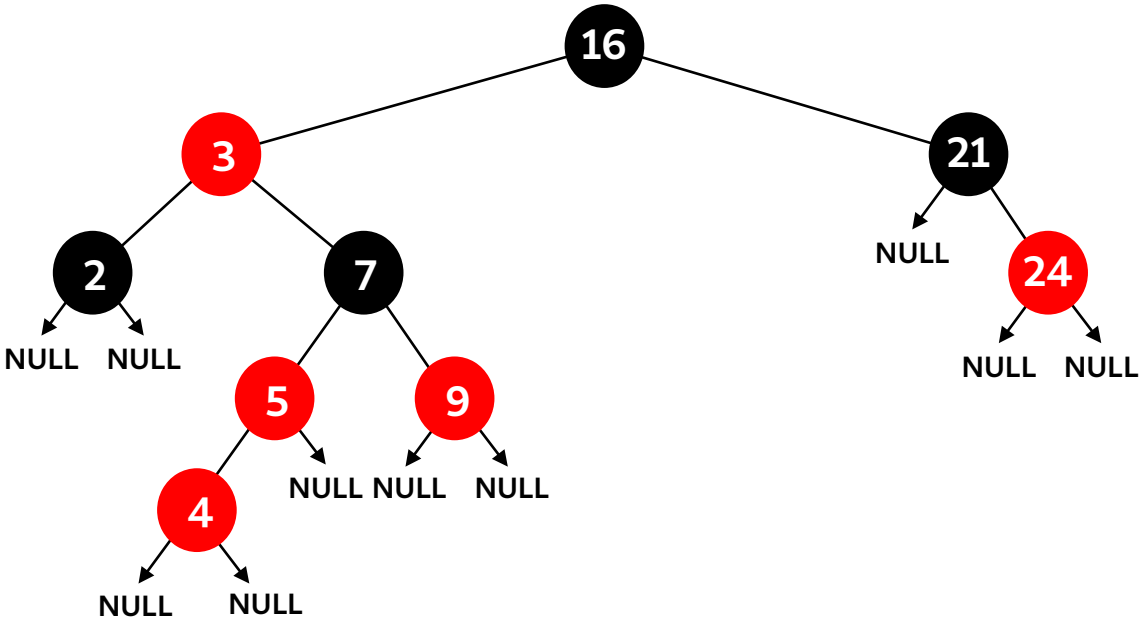
Восстановление красно-чёрных свойств, пример

```
bstree_add(4, value)
```

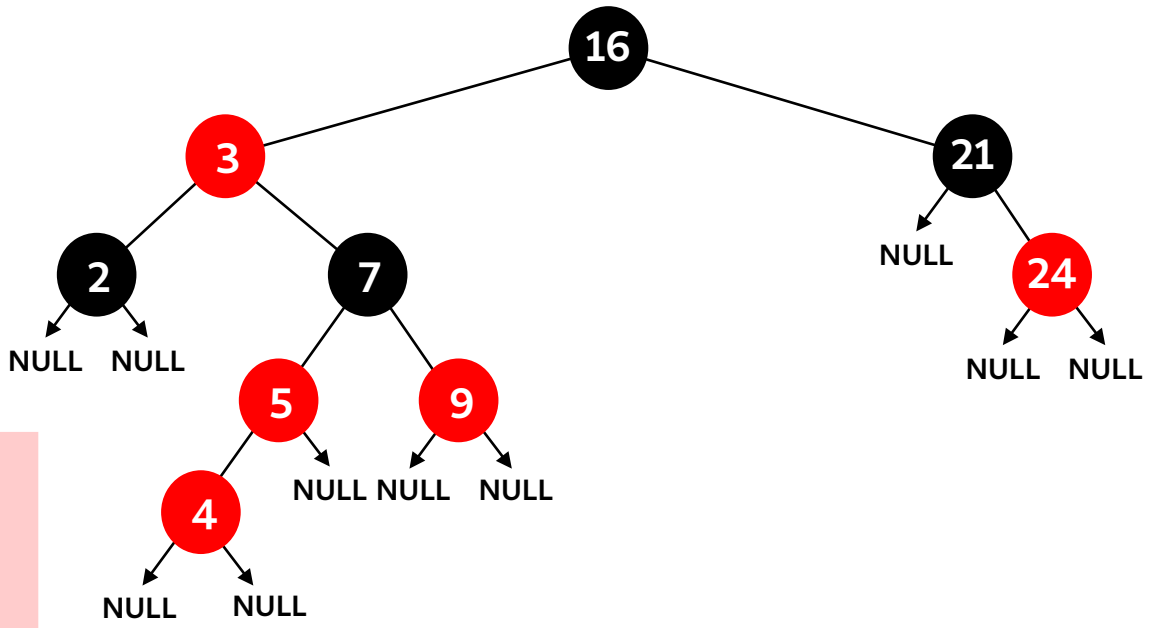


Восстановление красно-чёрных свойств, пример

```
bstree_add(4, value)
```



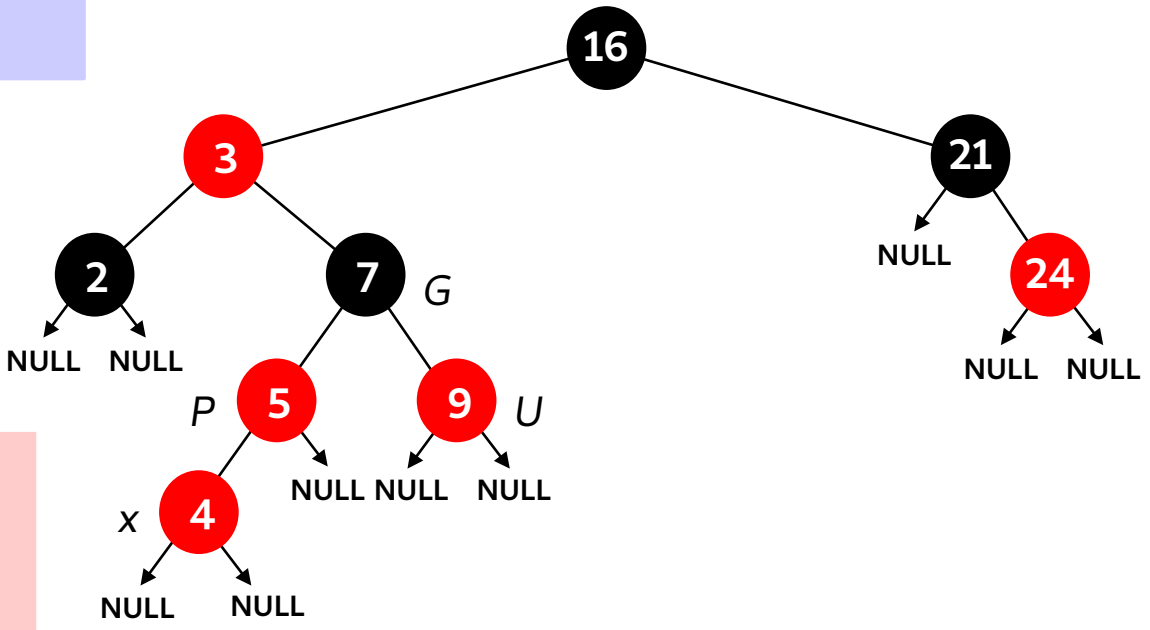
Восстановление красно-чёрных свойств, пример



Нарушено свойство 4 в узле 4

Восстановление красно-чёрных свойств, пример

Случай 1: «дядя» нового узла — красный

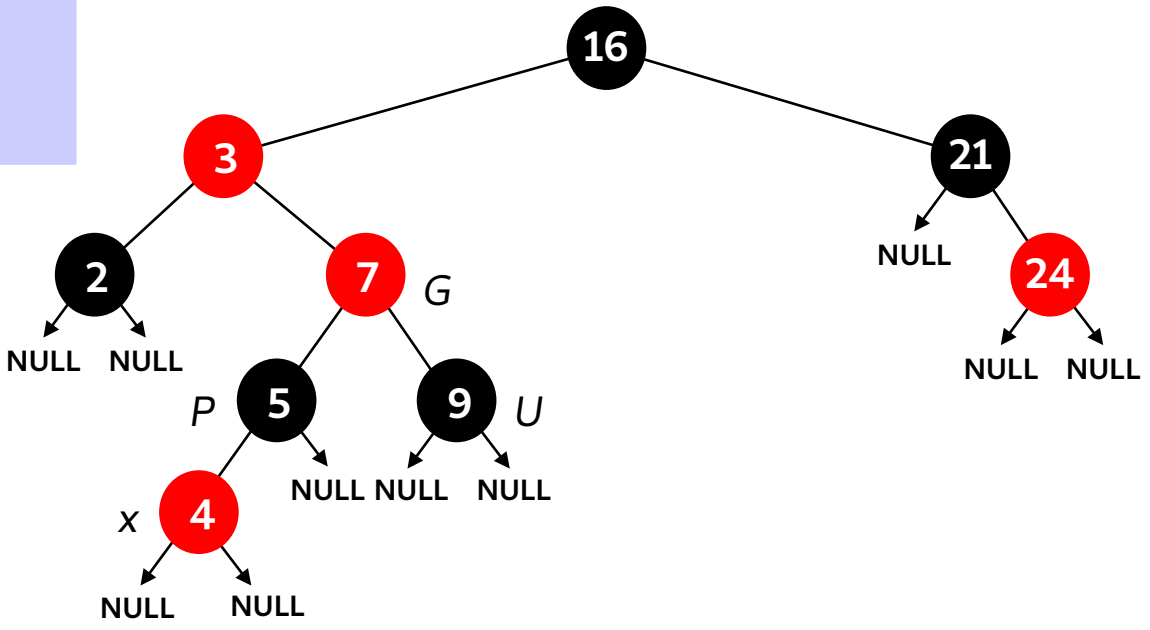


Нарушено свойство 4 в узле 4

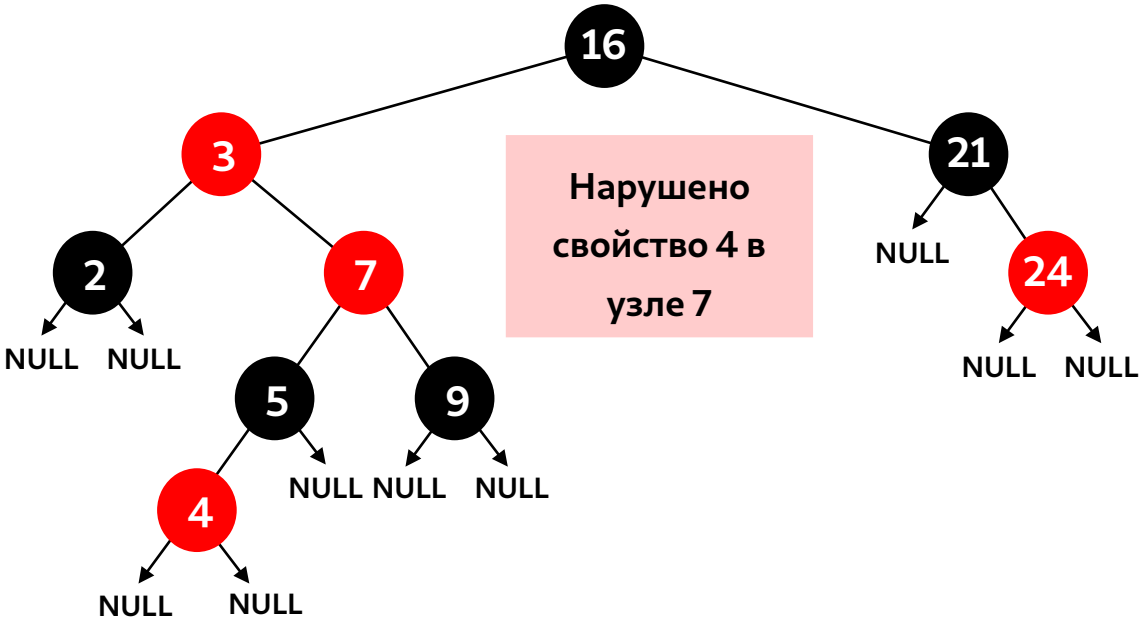
Восстановление красно-чёрных свойств, пример

Перекрашиваем узлы:

- *P* — чёрный
- *U* — чёрный
- *G* — красный

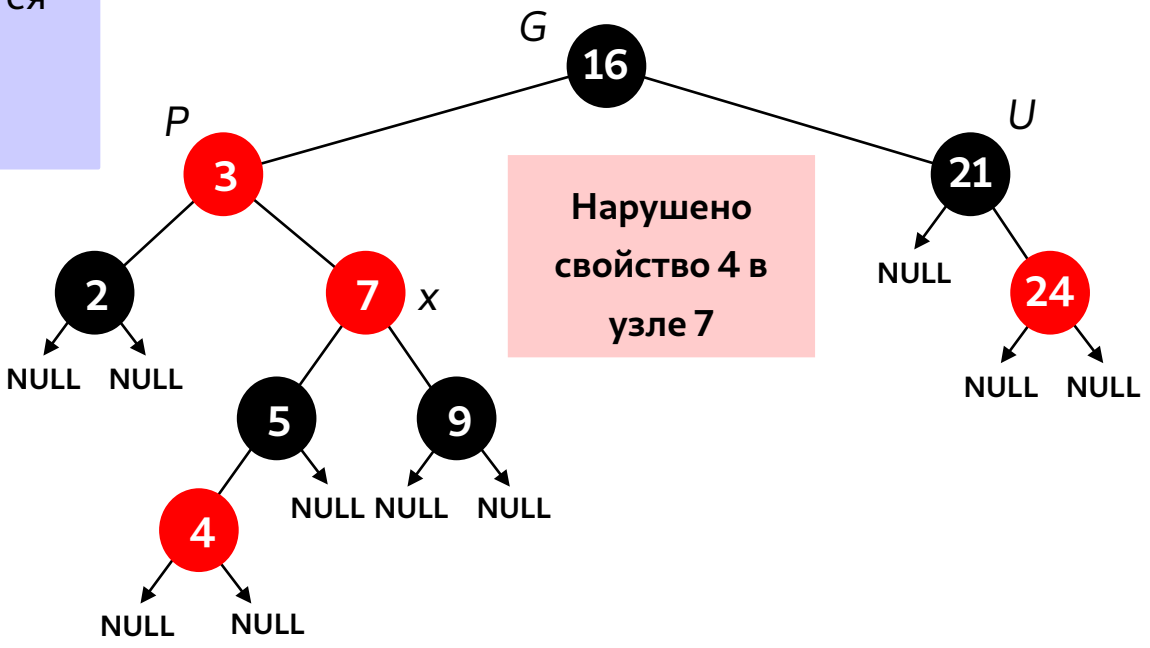


Восстановление красно-чёрных свойств, пример



Восстановление красно-чёрных свойств, пример

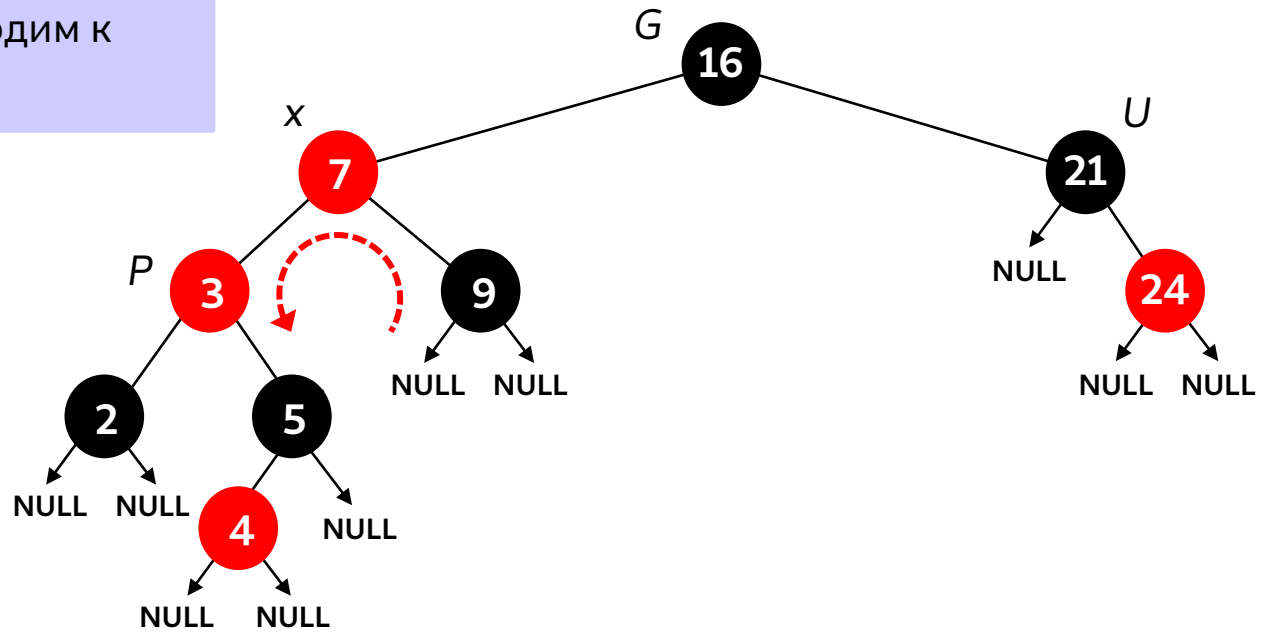
Случай 2: «дядя» узла x — чёрный, x является правым потомком своего родителя



Нарушено свойство 4 в узле 7

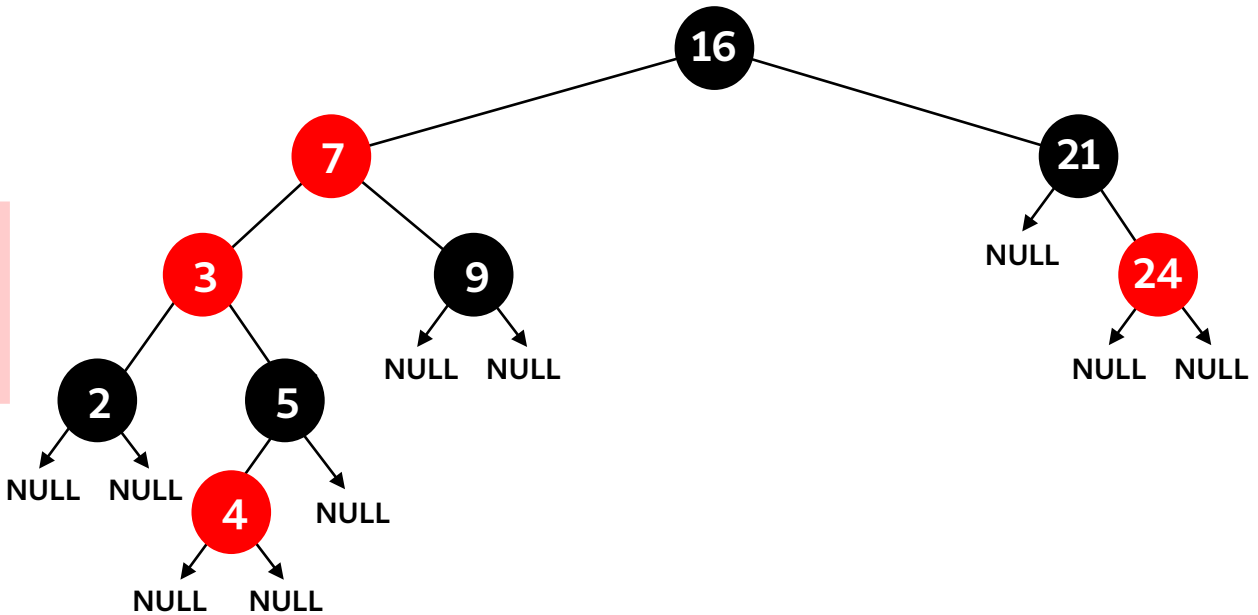
Восстановление красно-чёрных свойств, пример

Поворачивая дерево *P* влево, переходим к случаю 3



Восстановление красно-чёрных свойств, пример

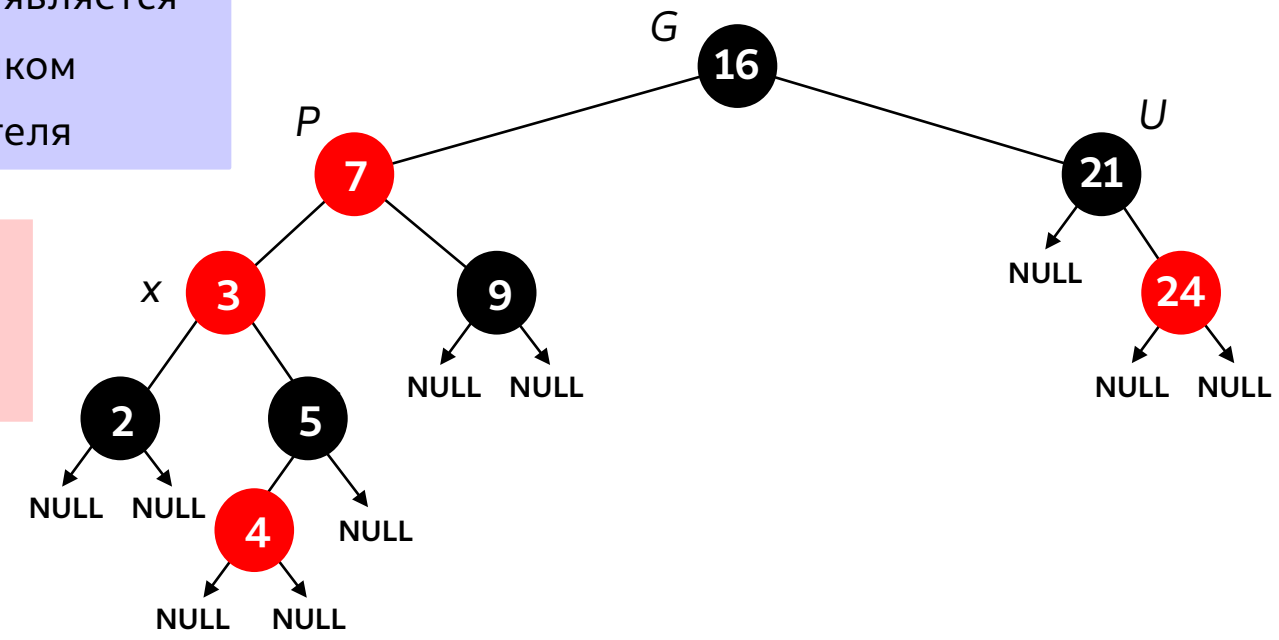
Нарушено свойство 4 в узле 3



Восстановление красно-чёрных свойств, пример

Случай 3: «дядя» узла *x* — чёрный, *x* является левым потомком своего родителя

Нарушено свойство 4 в узле 3



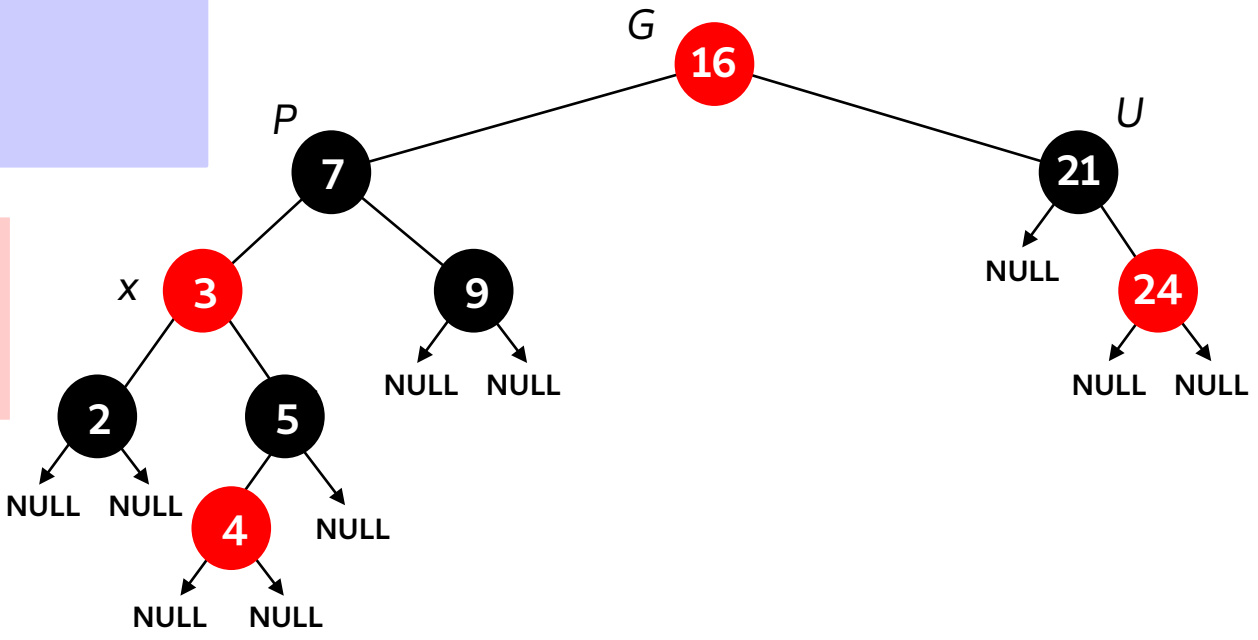
Восстановление красно-чёрных свойств, пример

1. Перекрашиваем вершины:

P — чёрный

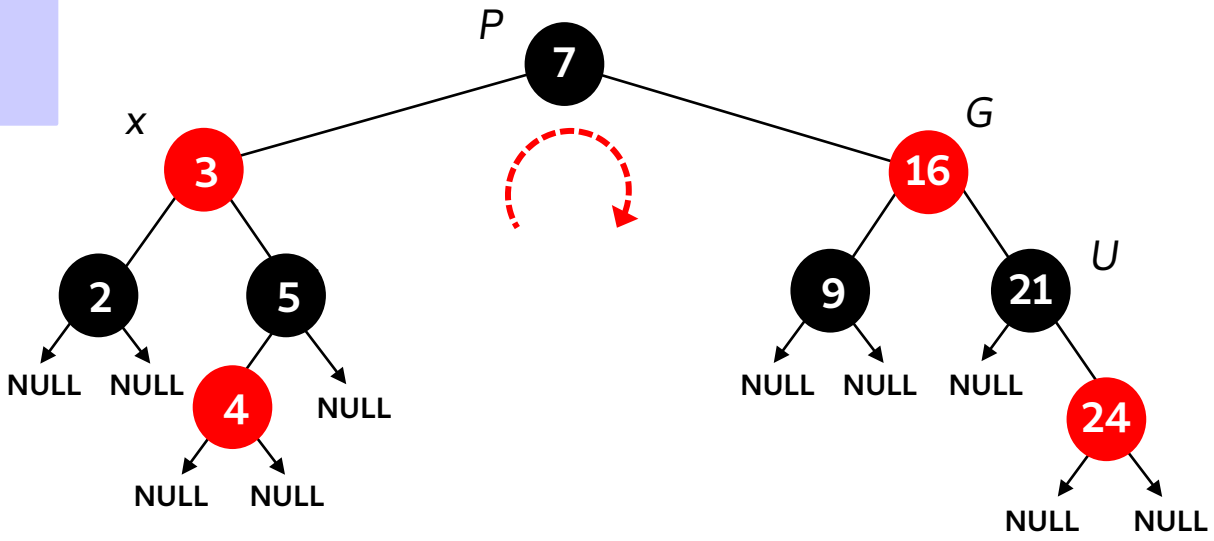
G — красный

Нарушено свойство 4 в узле 3



Восстановление красно-чёрных свойств, пример

2. Выполняем правый поворот дерева *G*



Восстановление красно-чёрных свойств, пример

Дерево сбалансировано