5. Красно-черные деревья

150 слайдов

Шустова Л.И.

Красно-черные деревья

Красно-черное (RB – Red-Black) дерево – бинарное дерево поиска с дополнительным полем цвета в узле

Красно-черные деревья

Красно-черное (RB – Red-Black) дерево – бинарное дерево поиска с дополнительным полем цвета в узле

```
typedef struct Node {

    int key;
    struct Node *left, *right;

} Node;
```

Красно-черные деревья

Красно-черное (RB – Red-Black) дерево – бинарное дерево поиска с дополнительным полем цвета в узле

```
typedef struct Node {

   int key;
   struct Node *left, *right;
   struct Node *parent;
} Node;
```

Красно-черные деревья

Красно-черное (RB – Red-Black) дерево – бинарное дерево поиска с дополнительным полем цвета в узле

```
typedef struct Node {
    int color;
    int key;
    struct Node *left, *right;
    struct Node *parent;
} Node;
```

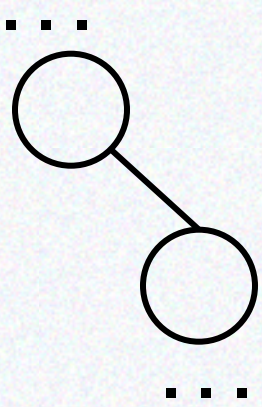
Красно-черные деревья

Вместо ограничения *NULL* используется указатель на внешний узел (лист) – *EList*

Красно-черные деревья

Вместо ограничения *NULL* используется указатель на внешний узел (лист) – *EList*

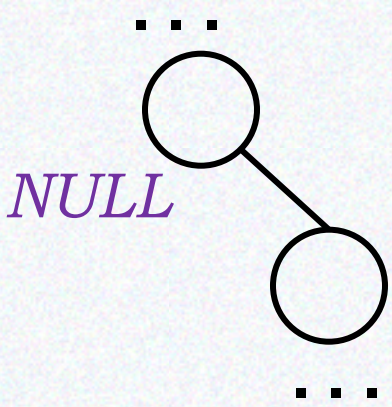
Двоичное дерево



Красно-черные деревья

Вместо ограничения *NULL* используется указатель на внешний узел (лист) – *EList*

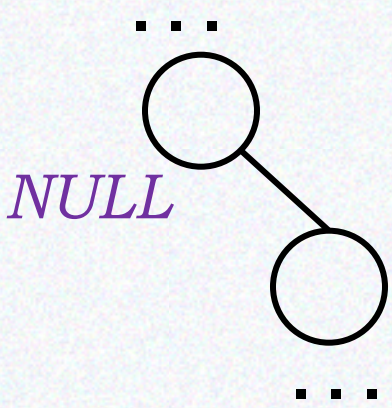
Двоичное дерево



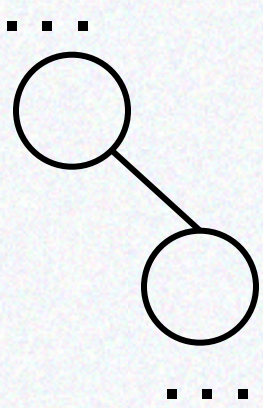
Красно-черные деревья

Вместо ограничения *NULL* используется указатель на внешний узел (лист) – *EList*

Двоичное дерево



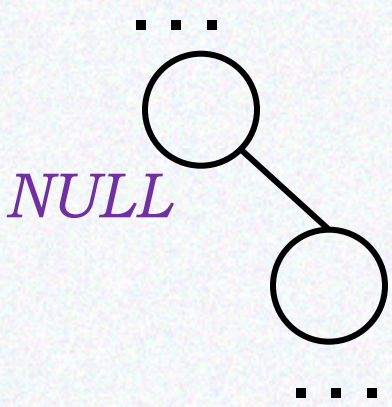
Красно-черное дерево



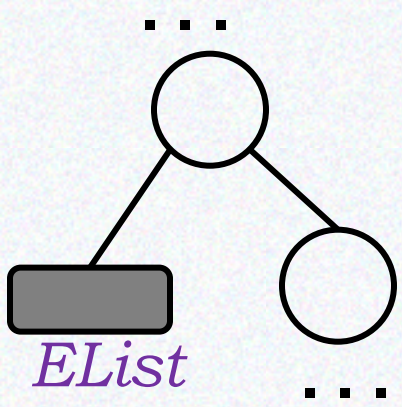
Красно-черные деревья

Вместо ограничения *NULL* используется указатель на внешний узел (лист) – *EList*

Двоичное дерево



Красно-черное дерево



Красно-черные деревья

Красно-черное бинарное дерево поиска удовлетворяет следующим свойствам:

Красно-черные деревья

Красно-черное бинарное дерево поиска удовлетворяет следующим свойствам:

1. Каждый узел является красным или черным

Красно-черные деревья

Красно-черное бинарное дерево поиска удовлетворяет следующим свойствам:

1. Каждый узел является красным или черным
2. Корень дерева является черным

Красно-черные деревья

Красно-черное бинарное дерево поиска удовлетворяет следующим свойствам:

1. Каждый узел является красным или черным
2. Корень дерева является черным
3. Каждый лист дерева (*EList*) является черным

Красно-черные деревья

Красно-черное бинарное дерево поиска удовлетворяет следующим свойствам:

1. Каждый узел является красным или черным
2. Корень дерева является черным
3. Каждый лист дерева (*EList*) является черным
4. Если узел – красный, то оба его дочерних узла (потомка) – черные

Красно-черные деревья

Красно-черное бинарное дерево поиска удовлетворяет следующим свойствам:

1. Каждый узел является красным или черным
2. Корень дерева является черным
3. Каждый лист дерева (*EList*) является черным
4. Если узел – красный, то оба его дочерних узла (потомка) – черные
5. Для каждого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

Красно-черные деревья

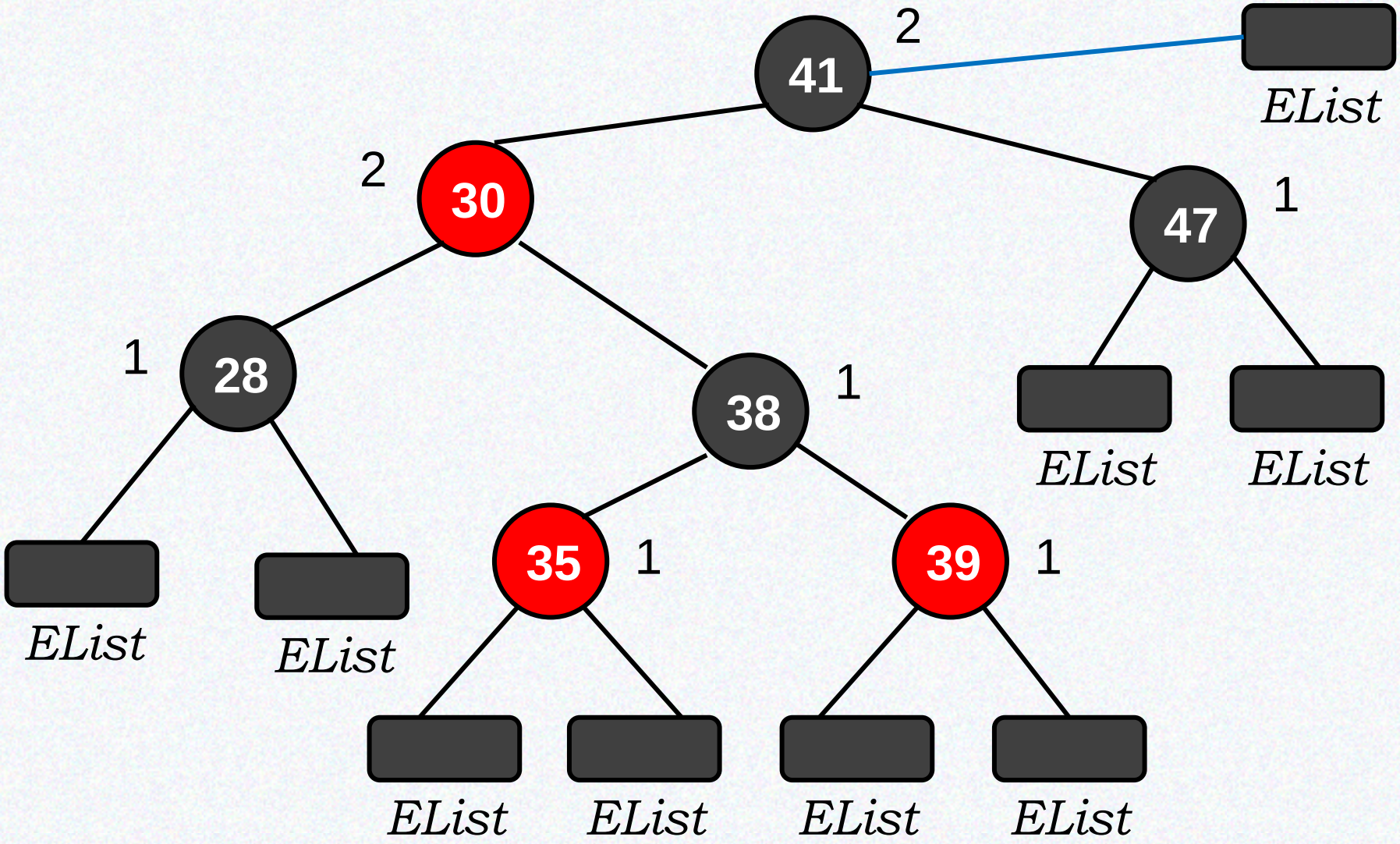
Количество черных узлов на пути от узла x (не считая сам узел) к некоторому (любому) листу называется черной высотой листа $bh(x)$ – *black height*

Красно-черные деревья

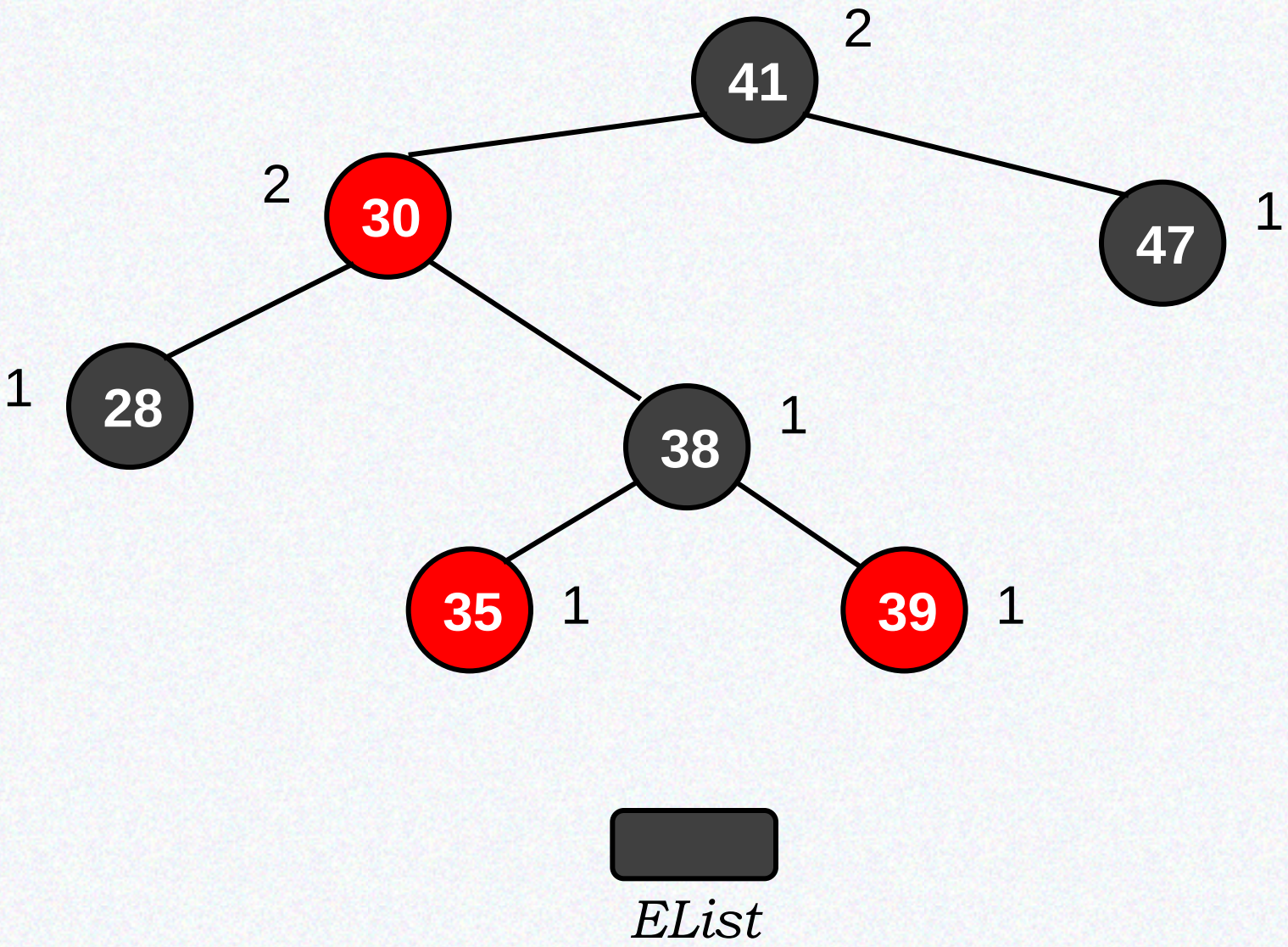
Количество черных узлов на пути от узла x (не считая сам узел) к некоторому (любому) листу называется черной высотой листа $bh(x)$ – *black height*

Черной высотой дерева является черная высота его корня

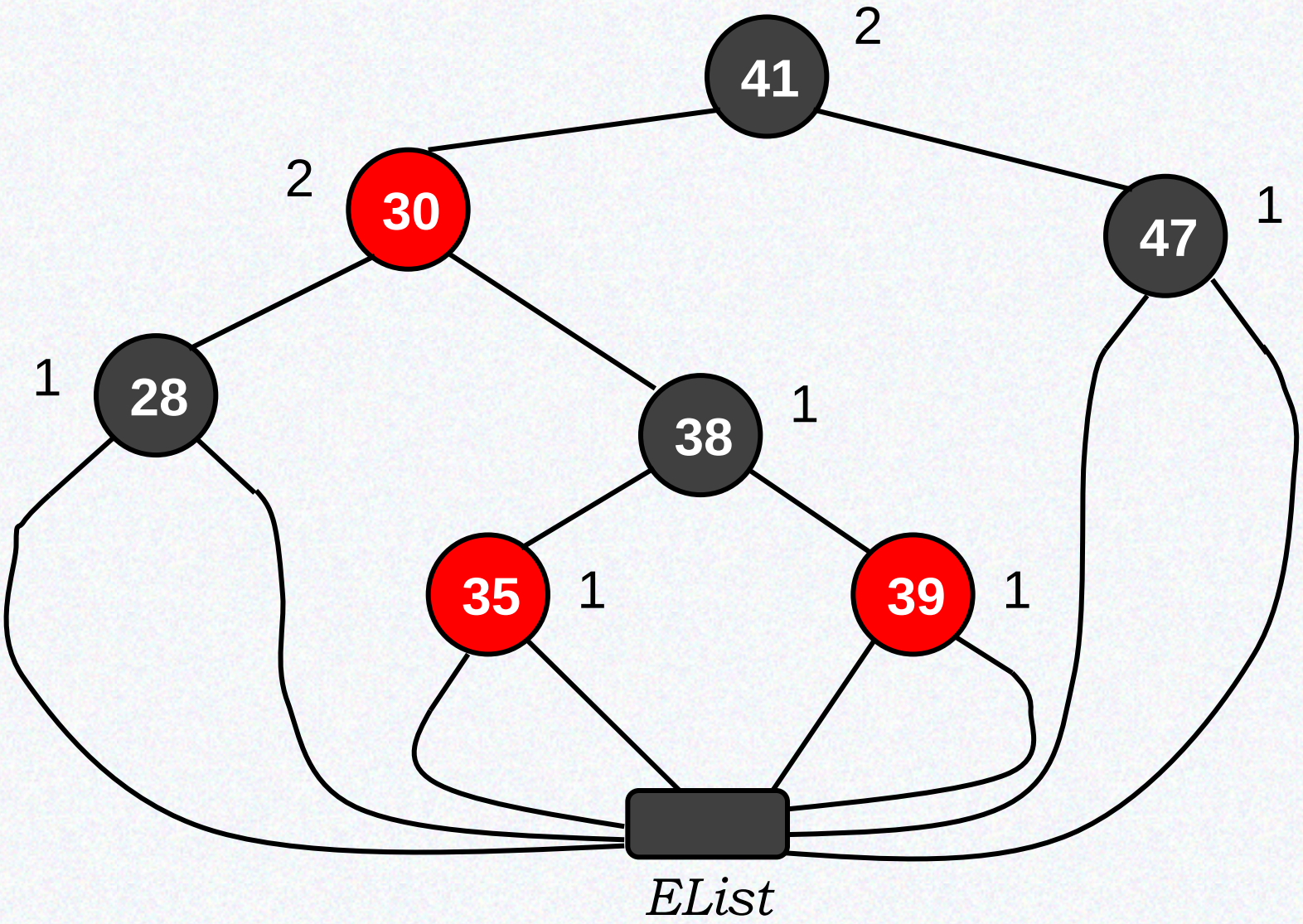
Пример красно-черного дерева



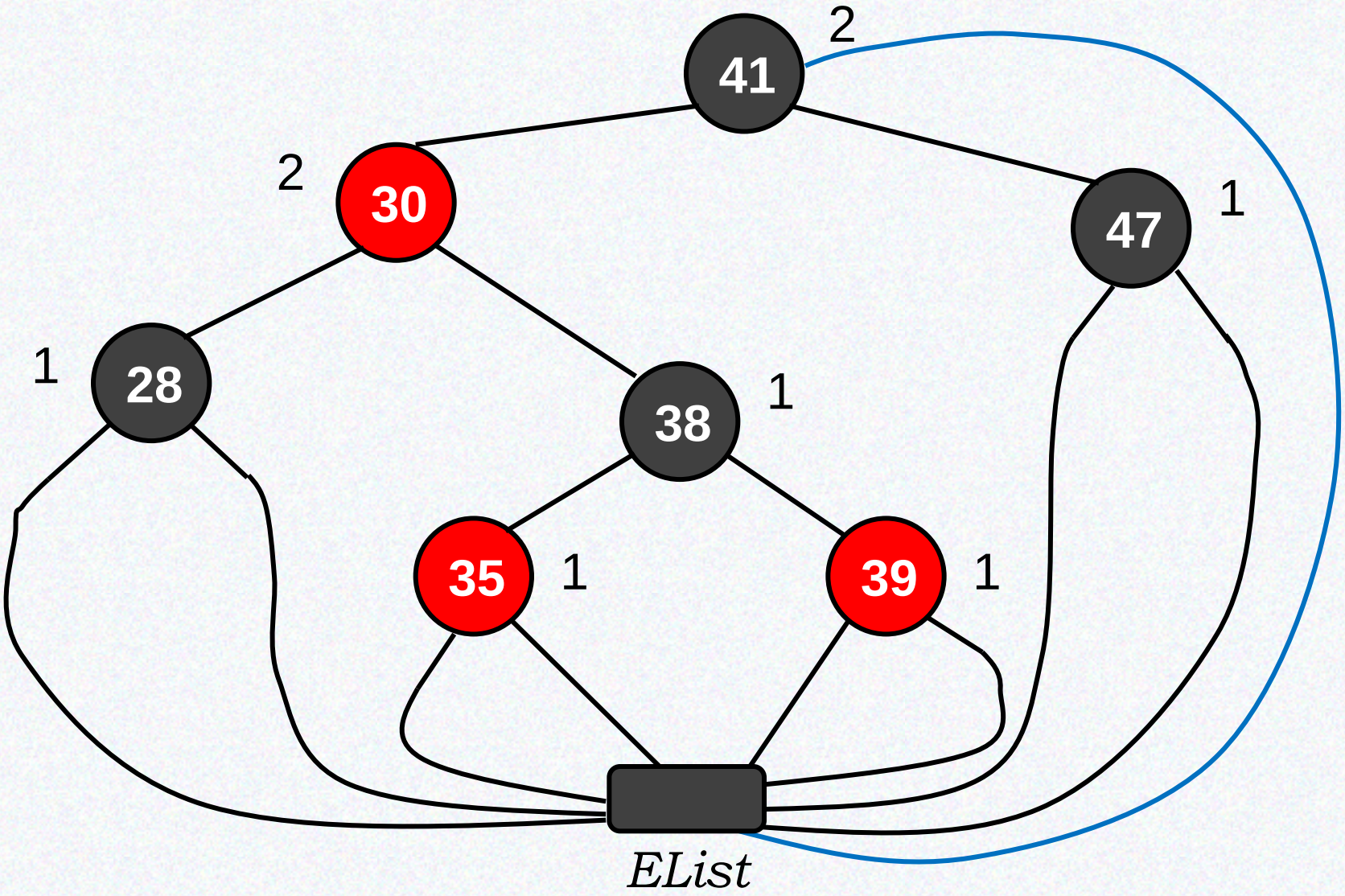
Представление RB- дерева



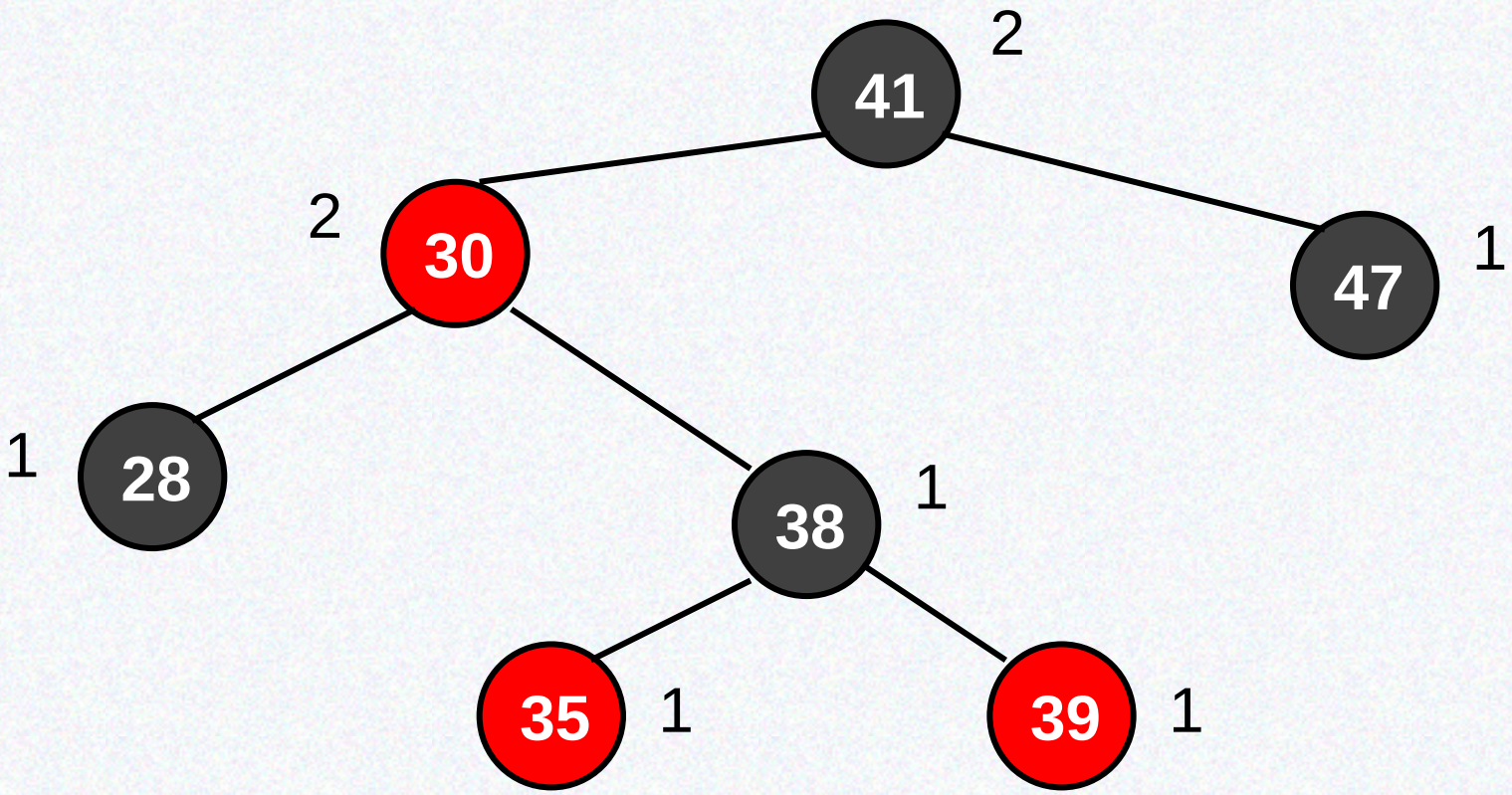
Представление RB- дерева



Представление RB- дерева



Представление RB- дерева



Характеристики RB- дерева

Характеристики RB- дерева

Красно-черное дерево с n внутренними узлами имеет высоту h не более чем

$$2 \log (n + 1)$$

Характеристики RB- дерева

Красно-черное дерево с n внутренними узлами имеет высоту h не более чем

$$2 \log (n + 1)$$

Черная высота корня должна составлять как минимум $h / 2$

Характеристики RB- дерева

Красно-черное дерево с n внутренними узлами имеет высоту h не более чем

$$2 \log (n + 1)$$

Черная высота корня должна составлять как минимум $h / 2$

Операции поиска выполняются за время

$$O(h) = O(\log(n))$$

Характеристики RB- дерева

Красно-черное дерево с n внутренними узлами имеет высоту h не более чем

$$2 \log (n + 1)$$

Черная высота корня должна составлять как минимум $h / 2$

Операции поиска выполняются за время

$$O(h) = O(\log(n))$$

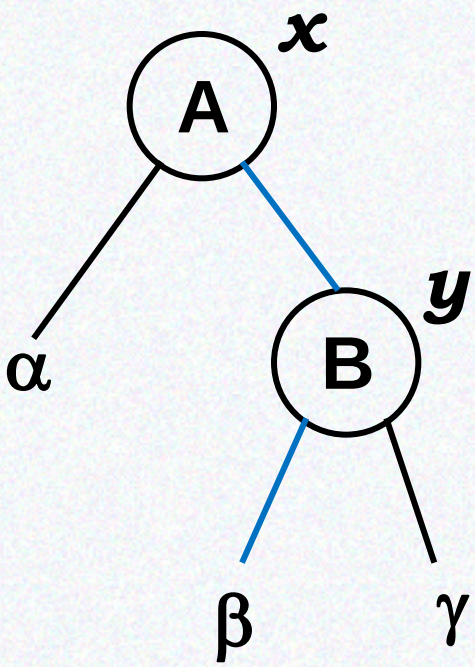
Операции вставки – удаления тоже могут быть выполнены за это время

Повороты

Локальные операции в бинарном дереве поиска, сохраняющие его свойства

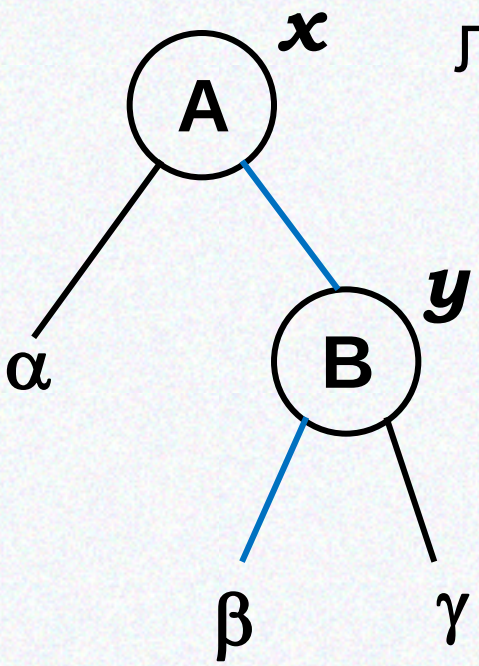
Повороты

Локальные операции в бинарном дереве поиска, сохраняющие его свойства



Повороты

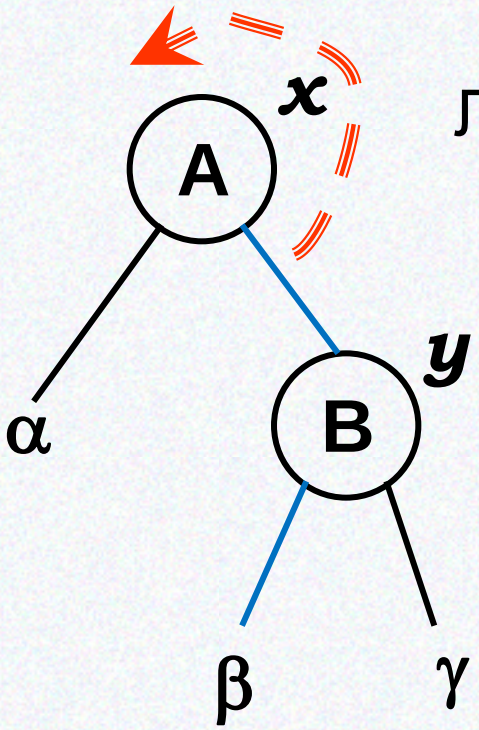
Локальные операции в бинарном дереве поиска, сохраняющие его свойства



Левый поворот – *left_rotate(x)*

Повороты

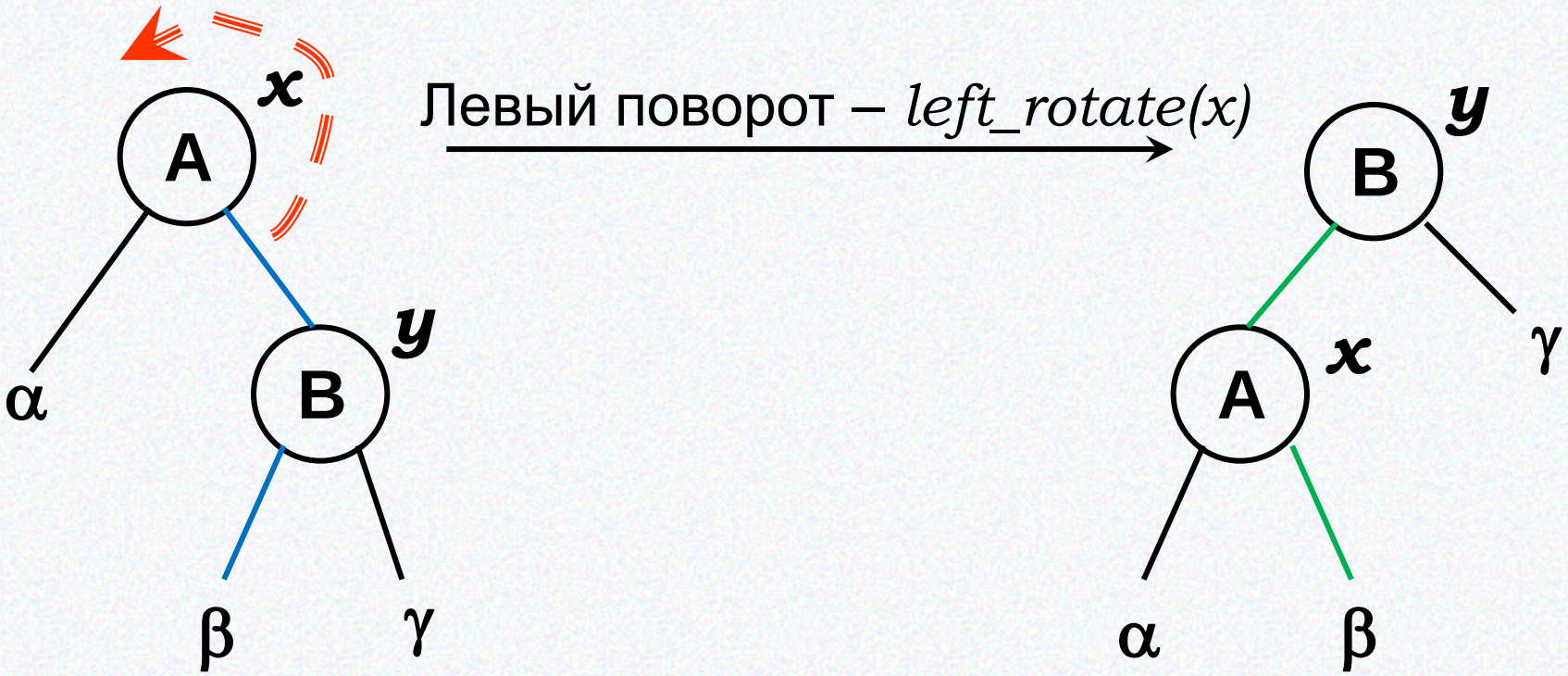
Локальные операции в бинарном дереве поиска, сохраняющие его свойства



Левый поворот – *left_rotate(x)*

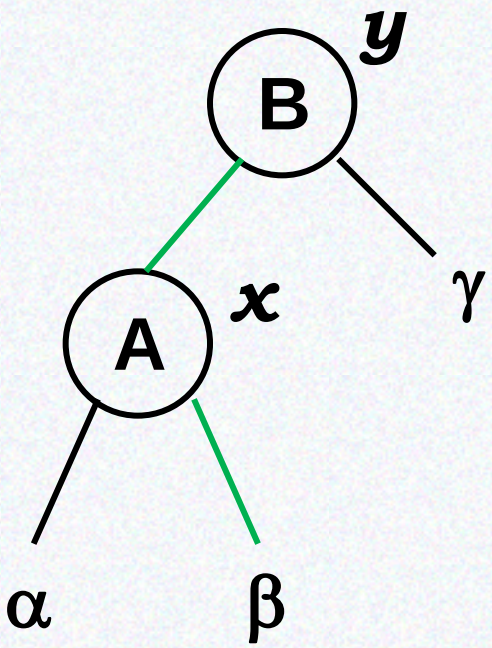
Повороты

Локальные операции в бинарном дереве поиска, сохраняющие его свойства



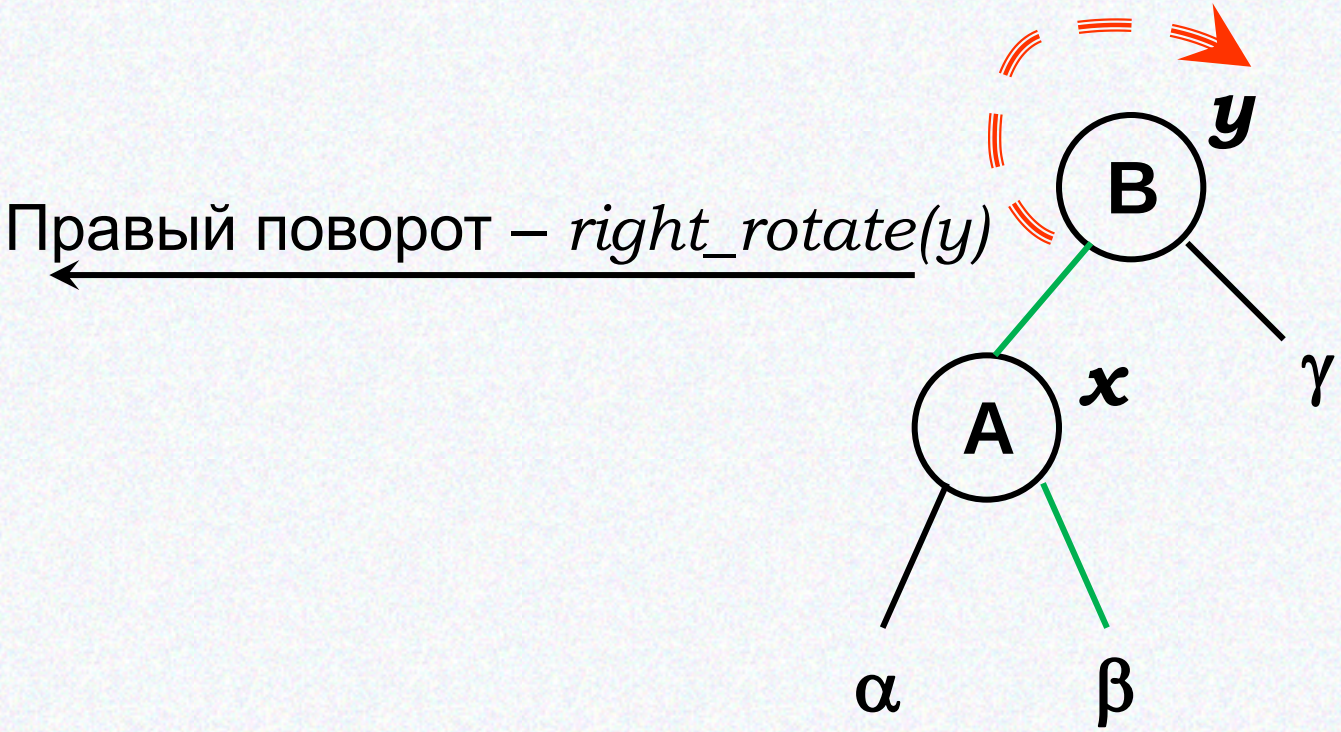
Повороты

Локальные операции в бинарном дереве поиска, сохраняющие его свойства



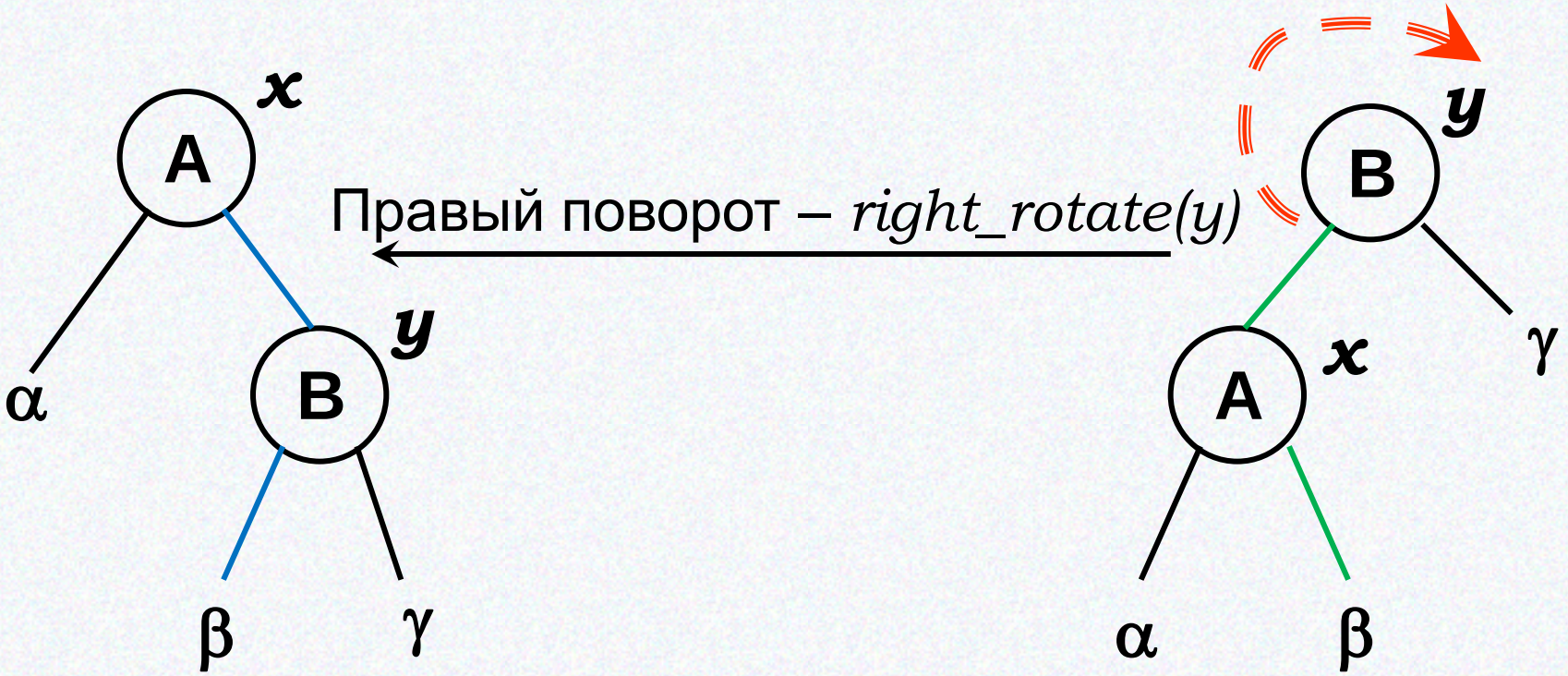
Повороты

Локальные операции в бинарном дереве поиска, сохраняющие его свойства

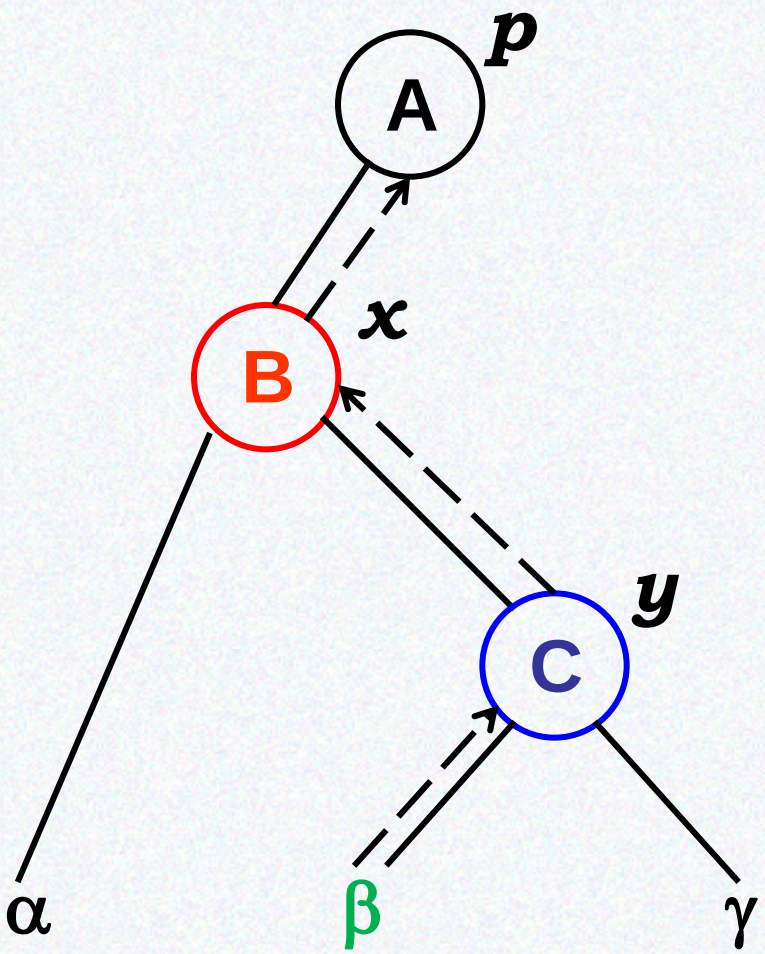


Повороты

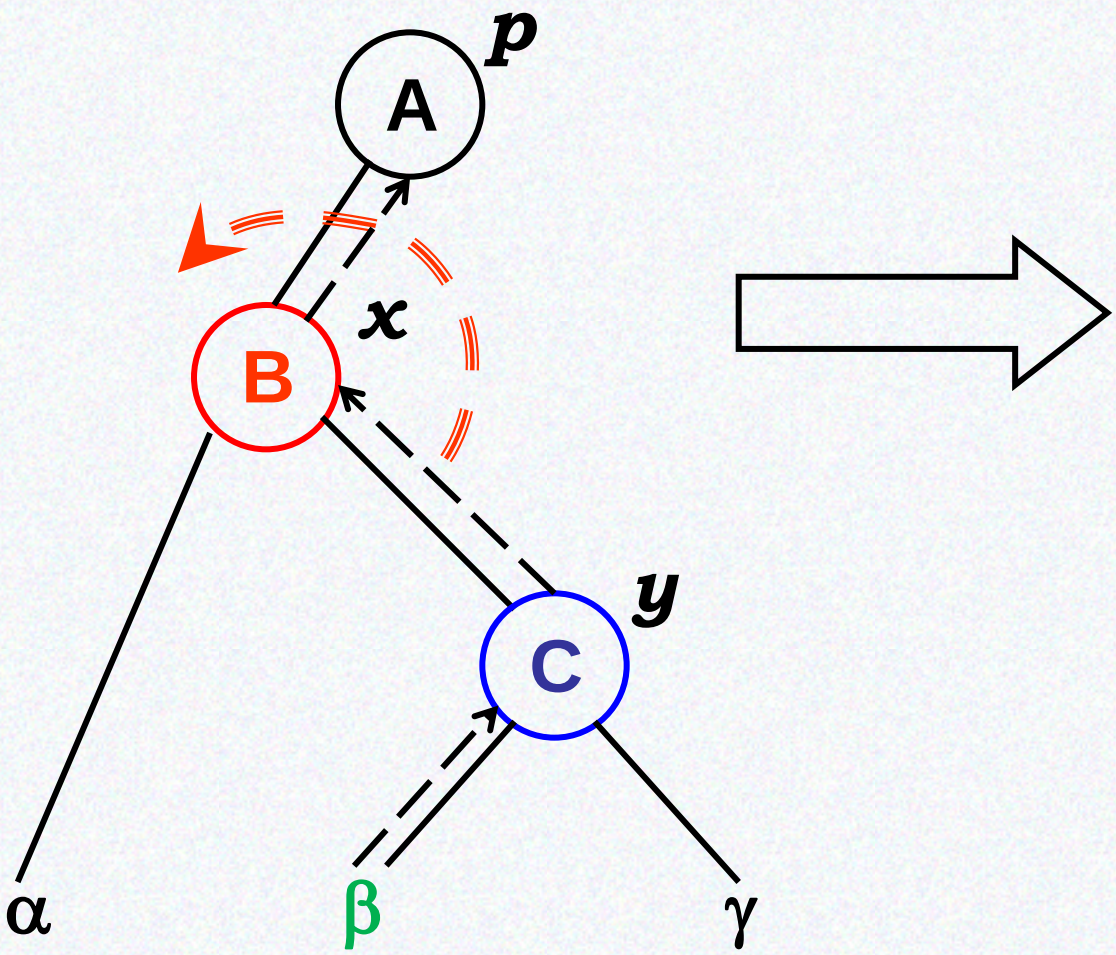
Локальные операции в бинарном дереве поиска, сохраняющие его свойства



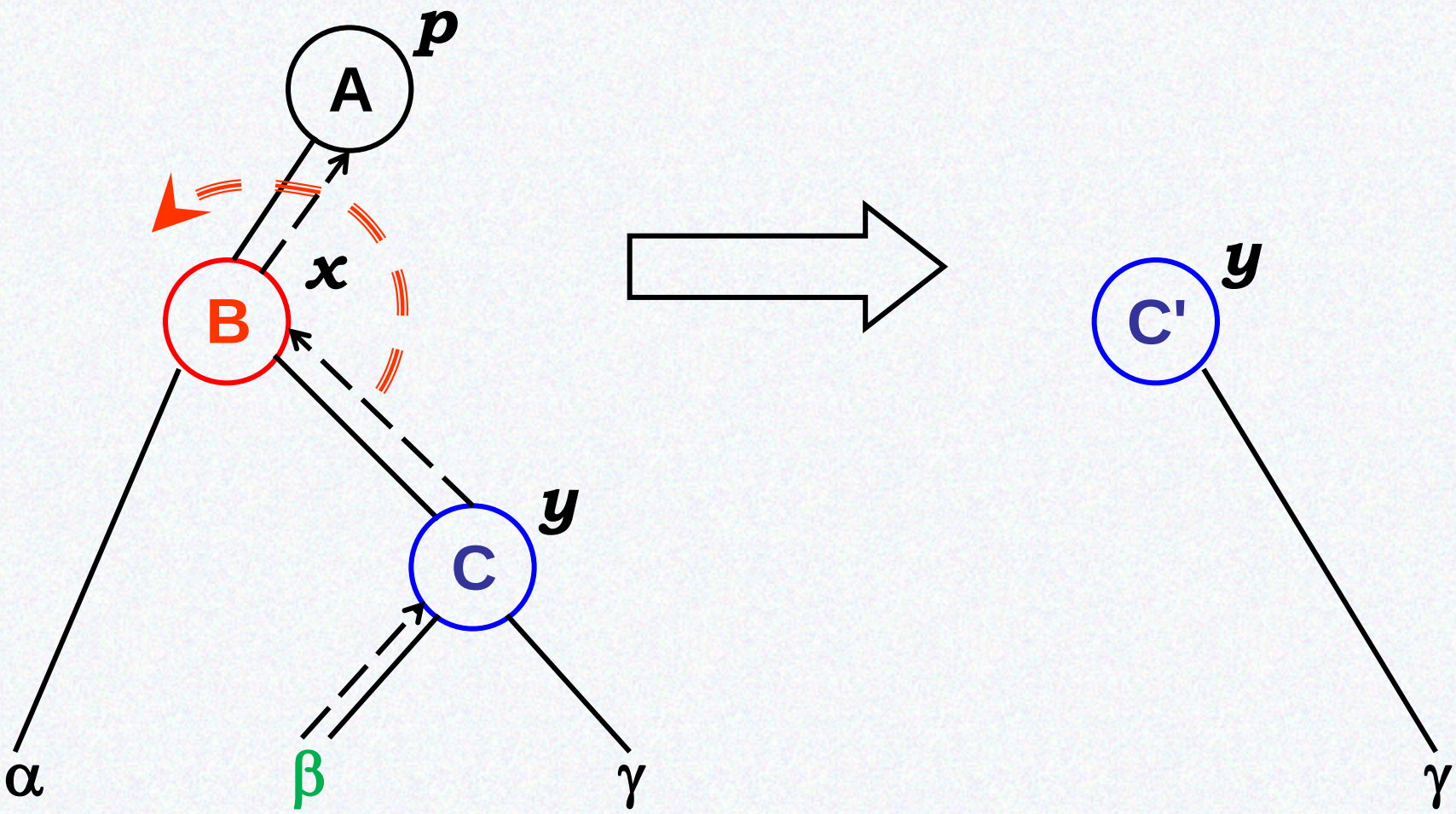
Левый поворот



Левый поворот

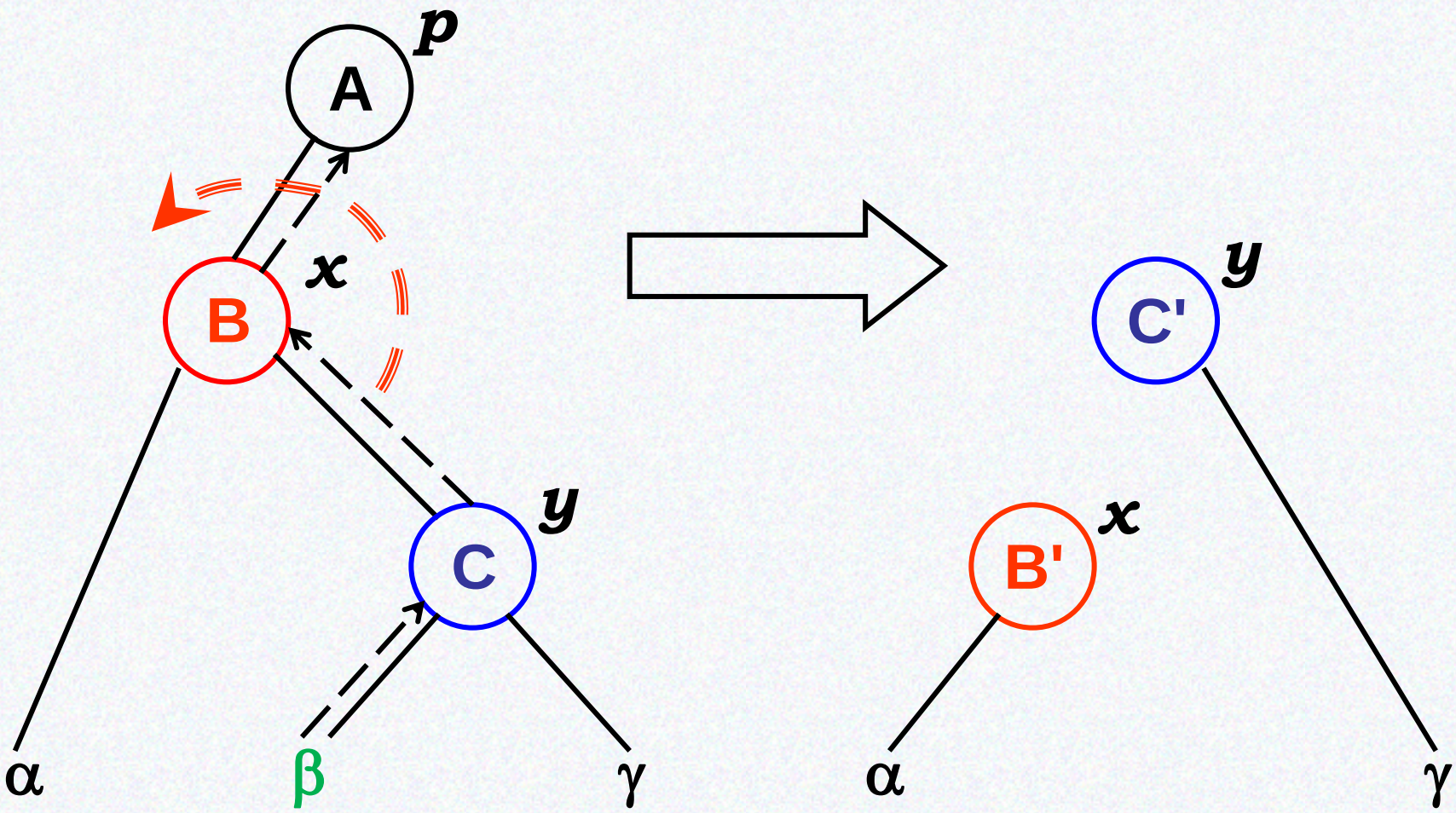


Левый поворот



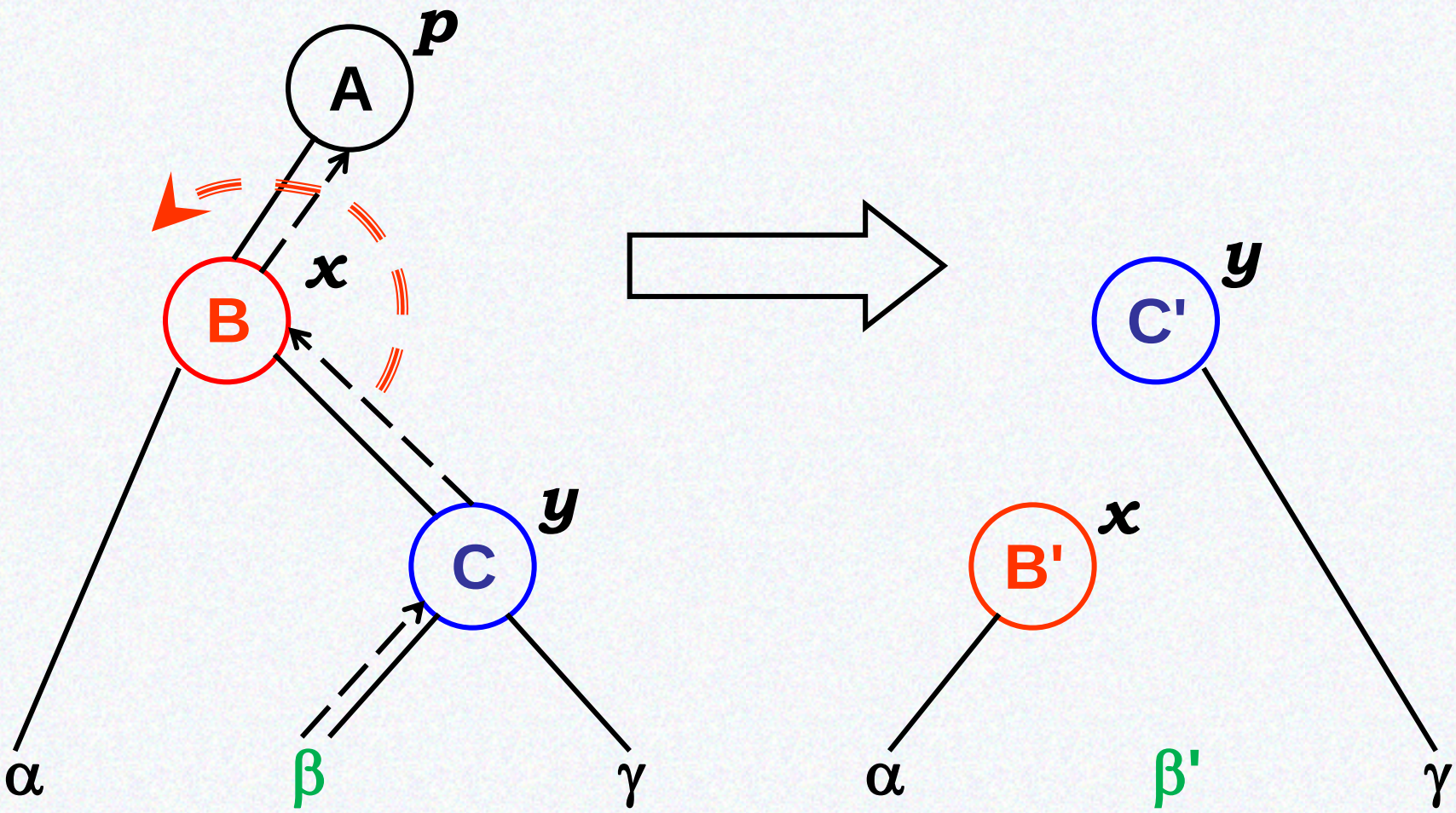
Шустова Л.И.

Левый поворот



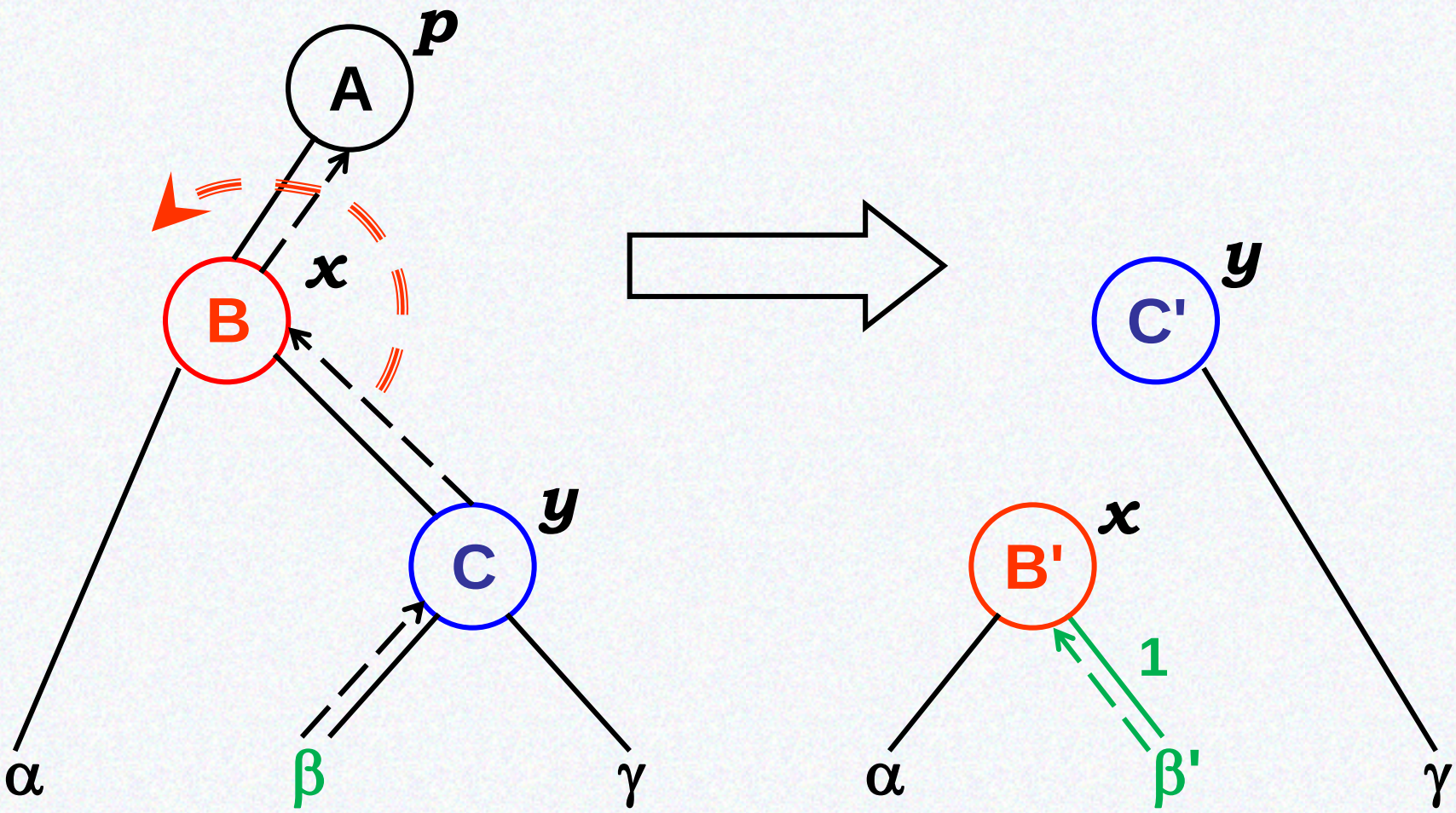
Шустова Л.И.

Левый поворот

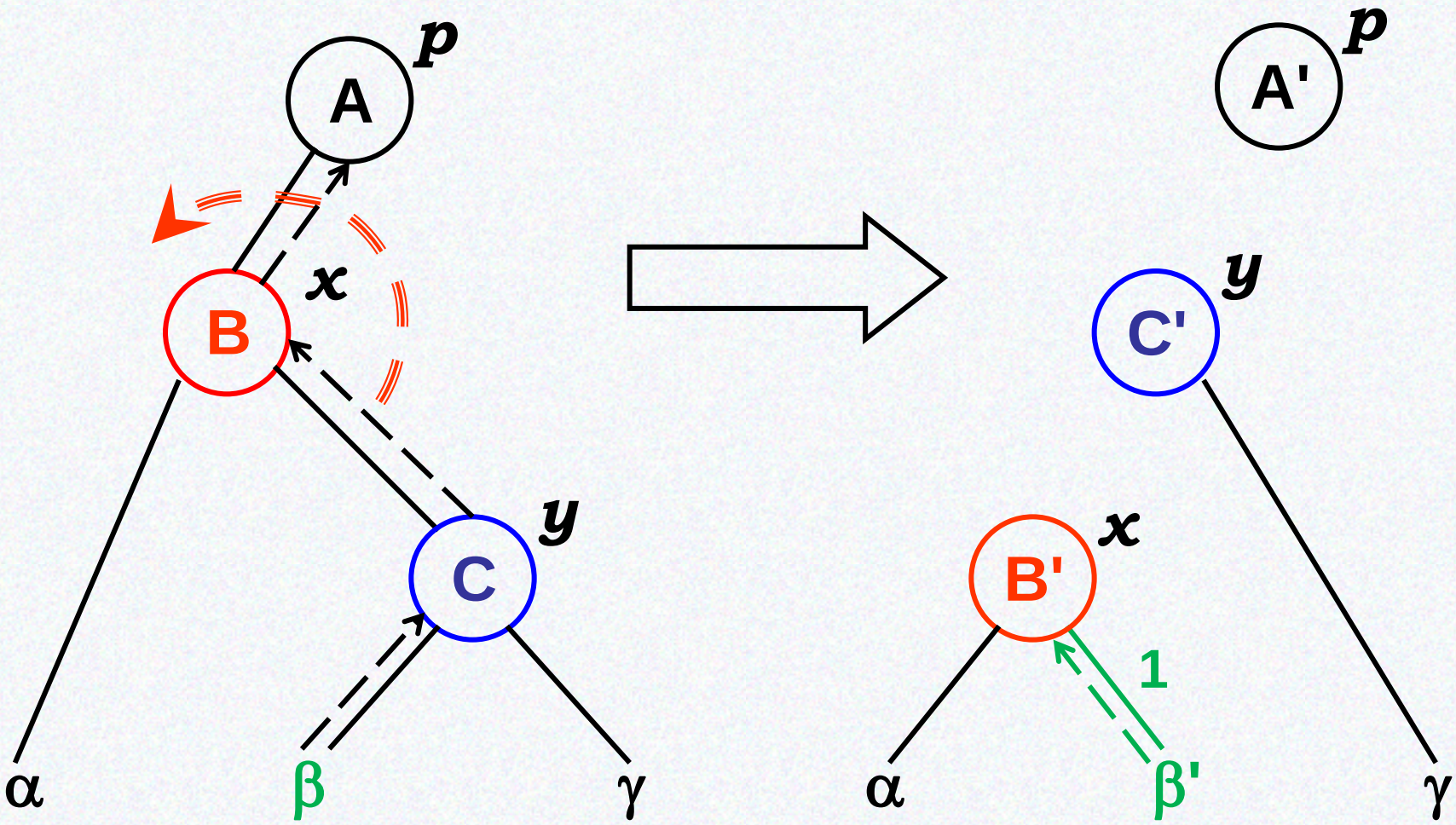


Шустова Л.И.

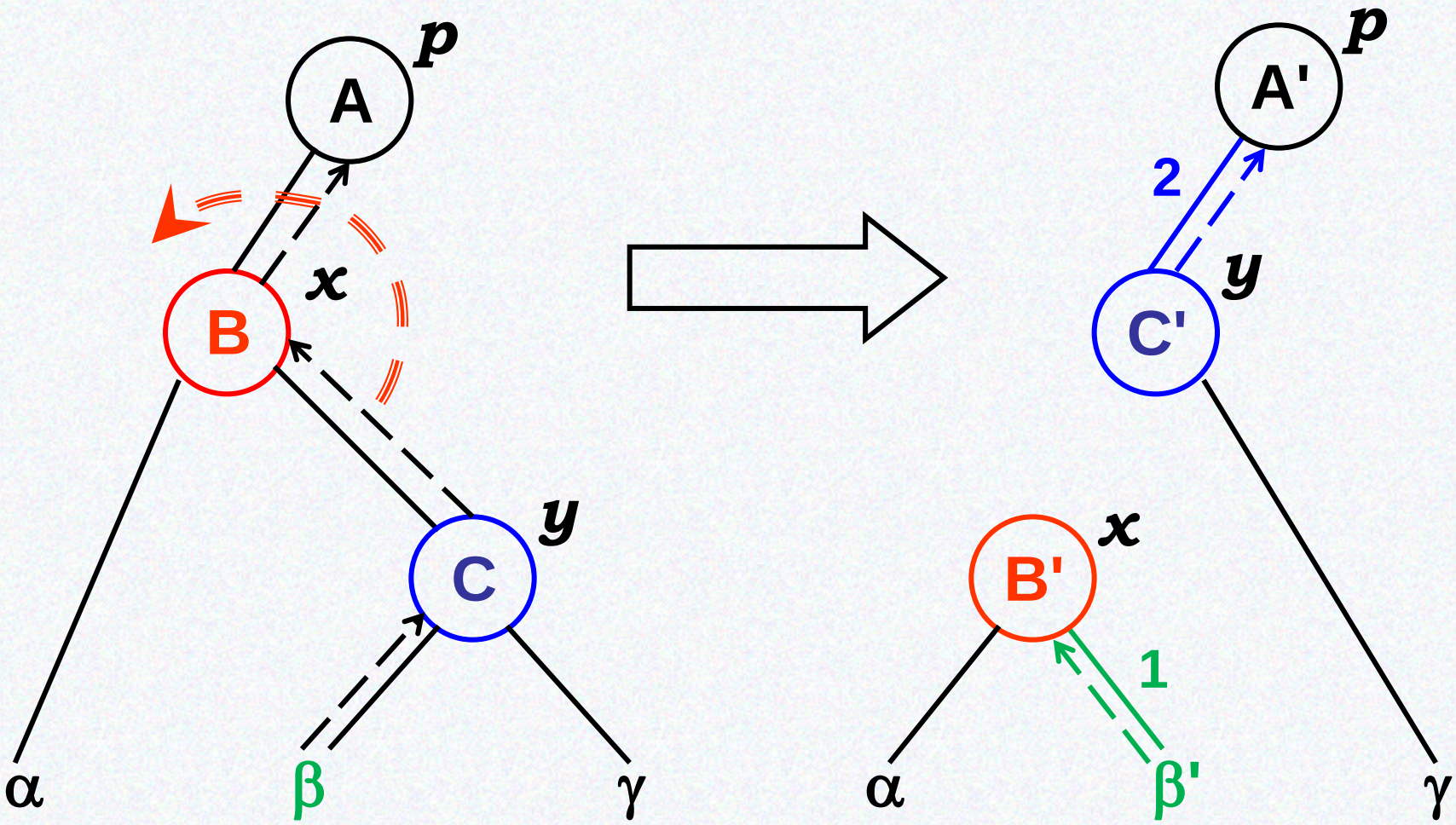
Левый поворот



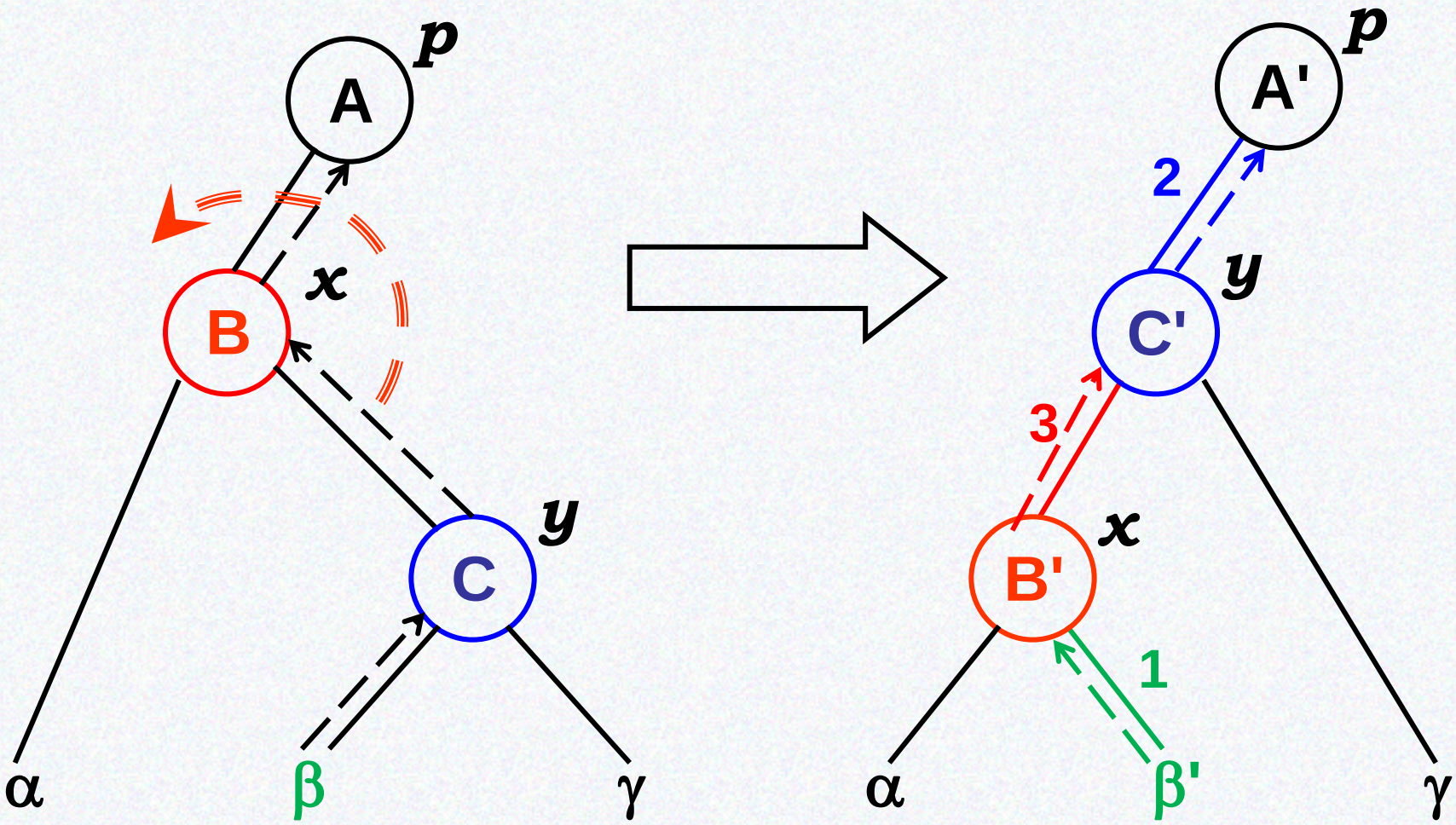
Левый поворот



Левый поворот



Левый поворот



Алгоритм *Left_rotate(x)*

x – заданный узел

$y = x->right$ – правое поддерево узла x

$p = x->parent$ – родительский узел узла x

Алгоритм *Left_rotate(x)*

x – заданный узел

$y = x->right$ – правое поддерево узла x

$p = x->parent$ – родительский узел узла x

1. Формирование связи x – левое поддерево y:

Алгоритм *Left_rotate(x)*

x – заданный узел

y = x->right – правое поддерево узла *x*

p = x->parent – родительский узел узла *x*

1. Формирование связи *x* – левое поддерево *y*:

Переустановить правое поддерево *x* (1а):

x->right = y->left

Алгоритм *Left_rotate(x)*

x – заданный узел

y = x->right – правое поддерево узла *x*

p = x->parent – родительский узел узла *x*

1. Формирование связи *x* – левое поддерево *y*:

Переустановить правое поддерево *x* (1а):

x->right = y->left

Переустановить родительский узел левого поддерева *y* (если он есть) (1б):

if y->left ≠ EList

y->left->parent = x

Алгоритм *Left_rotate(x)*

2. Формирование связи $p - y$:

Алгоритм *Left_rotate(x)*

2. Формирование связи $p - y$:

Переустановить родительский узел y (2а):

$y->parent = p$

Алгоритм *Left_rotate(x)*

2. Формирование связи $p - y$:

Переустановить родительский узел y (2а):

$y->parent = p$

Возможно, x был корнем дерева:

if $p = EList$

$root = y$ – новый корень дерева

Алгоритм *Left_rotate(x)*

2. Формирование связи $p - y$:

Переустановить родительский узел y (2а):

$y->parent = p$

Возможно, x был корнем дерева:

if $p = EList$

$root = y$ – новый корень дерева

else {

Переустановить левое или правое поддерево родительского узла x (2б):

Алгоритм *Left_rotate(x)*

```
    if p->left == x
        p->left = y
    else
        p->right = y
}
```

Алгоритм *Left_rotate(x)*

```
    if p->left == x
        p->left = y
    else
        p->right = y
}
```

3. Формирование связи $y - x$:

Алгоритм *Left_rotate(x)*

```
    if p->left == x
        p->left = y
    else
        p->right = y
}
```

3. Формирование связи $y - x$:

Переустановить левое поддерево y (3а):

```
    y->left = x
```

Алгоритм *Left_rotate(x)*

```
    if p->left == x
        p->left = y
    else
        p->right = y
}
```

3. Формирование связи $y - x$:

Переустановить левое поддерево y (3а):

```
y->left = x
```

Переустановить родительский узел для x (3б):

```
x->parent = y
```

Алгоритм *Left_rotate(x)*

```
    if p->left == x
        p->left = y
    else
        p->right = y
}
```

3. Формирование связи $y - x$:

Переустановить левое поддерево y (3а):

```
y->left = x
```

Переустановить родительский узел для x (3б):

```
x->parent = y
```

Время выполнения – *O(1)*

Вставка в RB- дерево

Алгоритм *RB_Insert(x)*:

Вставка в RB- дерево

Алгоритм *RB_Insert(x)*:

1. Вставить узел x как в обычное бинарное дерево поиска

Вставка в RB- дерево

Алгоритм *RB_Insert(x)*:

1. Вставить узел *x* как в обычное бинарное дерево поиска
2. Полям *left* и *right* нового узла присвоить значение указателя на узлы-листы *EList*

Вставка в RB- дерево

Алгоритм *RB_Insert(x)*:

1. Вставить узел *x* как в обычное бинарное дерево поиска
2. Полям *left* и *right* нового узла присвоить значение указателя на узлы-листы *EList*
3. Окрасить новый узел в красный цвет

Вставка в RB- дерево

Алгоритм *RB_Insert(x)*:

1. Вставить узел *x* как в обычное бинарное дерево поиска
2. Полям *left* и *right* нового узла присвоить значение указателя на узлы-листы *EList*
3. Окрасить новый узел в красный цвет
4. Перекрасить узлы и выполнить повороты

Алгоритм *RB_Insert(x)*

1. Вставка узла в бинарное дерево

prev = EList – родительский узел

ptr = root – текущий узел

while дерево не пусто: *ptr ≠ EList* {

  prev = ptr – родительский узел

  if x->key < ptr->key – выбор поддерева

    ptr = ptr->left – левое поддерево

  else

    ptr = ptr->right – правое поддерево

}

Алгоритм *RB_Insert(x)*

x->parent = prev – родительский узел

if вставка в пустое дерево: *prev == EList*

root = x – новый корень дерева

else

выбрать поддерево:

if x->key < prev->key

вставить в левое поддерево: *prev->left = x*

else

вставить в правое поддерево: *prev->right = x*

Алгоритм *RB_Insert(x)*

2. Установить поля *left* и *right* нового узла дерева:

x->left = EList

x->right = EList

Алгоритм *RB_Insert(x)*

2. Установить поля *left* и *right* нового узла дерева:

x->left = EList

x->right = EList

3. Покрасить узел: *x->color = RED*

Алгоритм *RB_Insert(x)*

2. Установить поля *left* и *right* нового узла дерева:

x->left = EList

x->right = EList

3. Покрасить узел: *x->color = RED*

4. Выполнить перекраску узлов и повороты дерева: *RB_Insert_Fixup(x)*

Нарушение свойств дерева

Нарушение свойств дерева

1. Каждый узел является красным или черным

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

2. Корень дерева является черным

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

2. Корень дерева является черным

   может быть нарушено

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

2. Корень дерева является черным

   может быть нарушено

3. Каждый лист дерева является черным

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

2. Корень дерева является черным

   может быть нарушено

3. Каждый лист дерева является черным

   не нарушается

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

2. Корень дерева является черным

   может быть нарушено

3. Каждый лист дерева является черным

   не нарушается

4. Если узел – красный, то оба его дочерних узла – черные

Нарушение свойств дерева

1. Каждый узел является красным или черным

   не нарушается

2. Корень дерева является черным

   может быть нарушено

3. Каждый лист дерева является черным

   не нарушается

4. Если узел – красный, то оба его дочерних узла – черные

   может быть нарушено

Нарушение свойств дерева

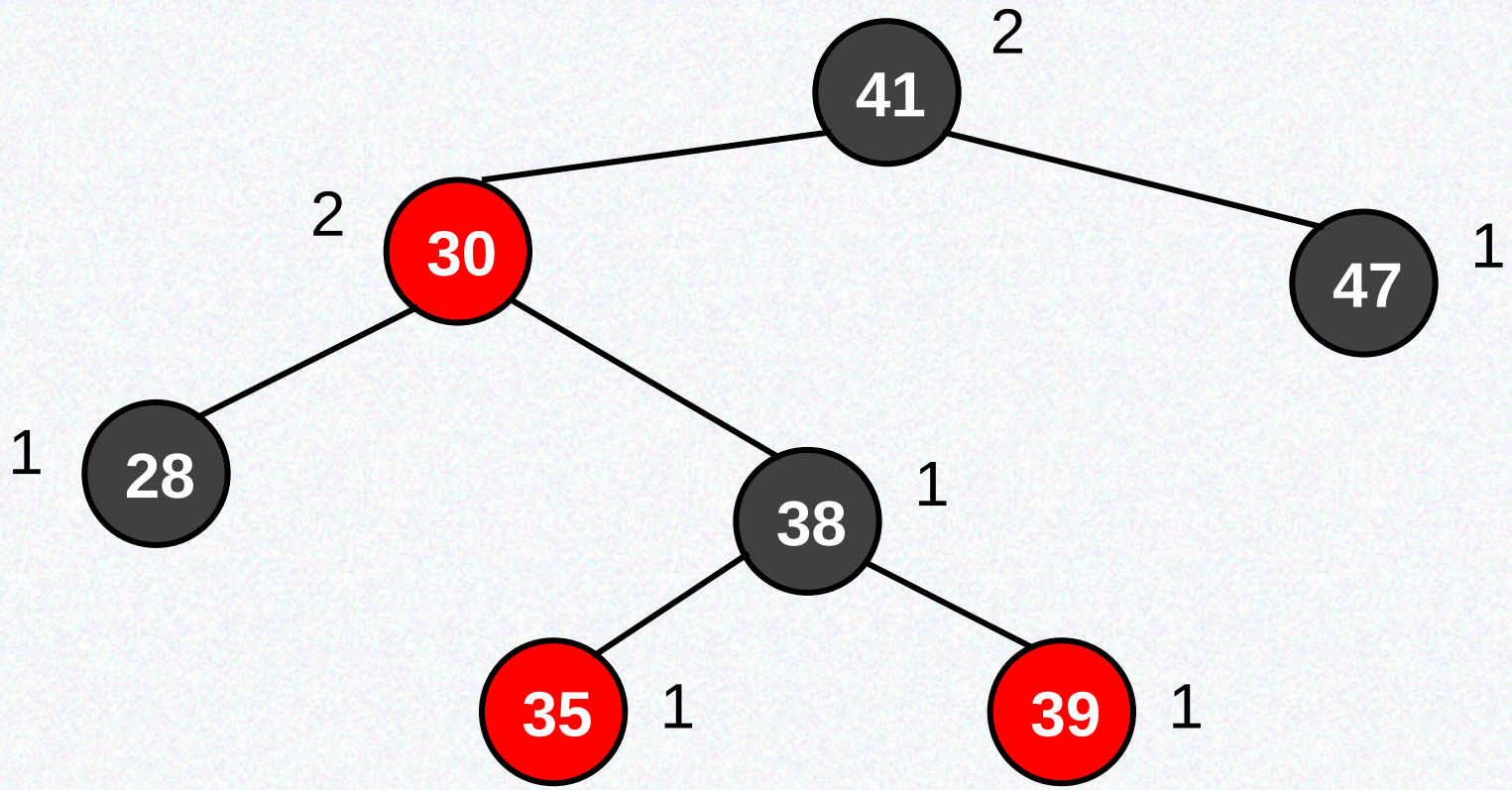
5. Для каждого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

Нарушение свойств дерева

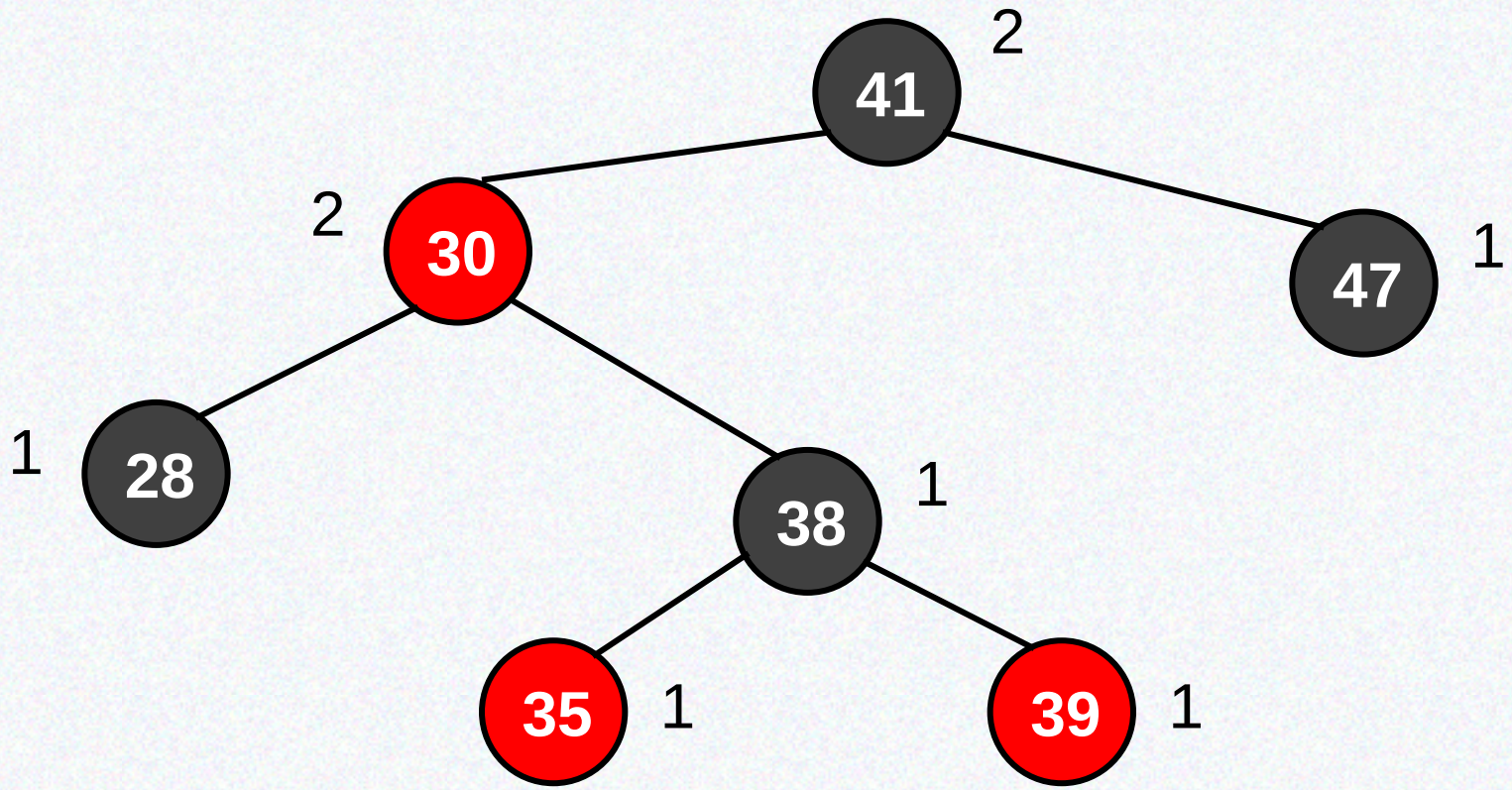
5. Для каждого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

не нарушается

Примеры вставки

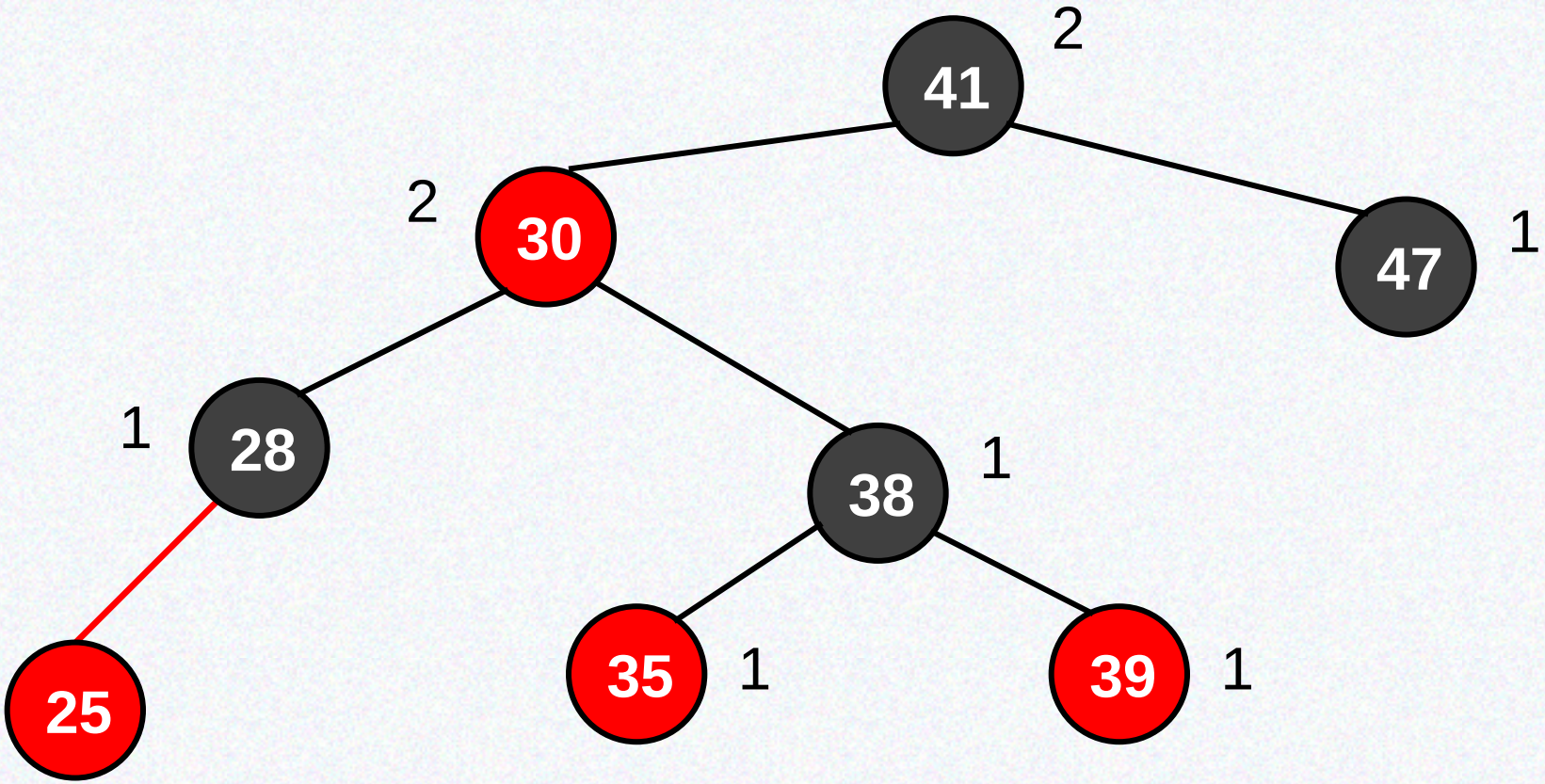


Примеры вставки



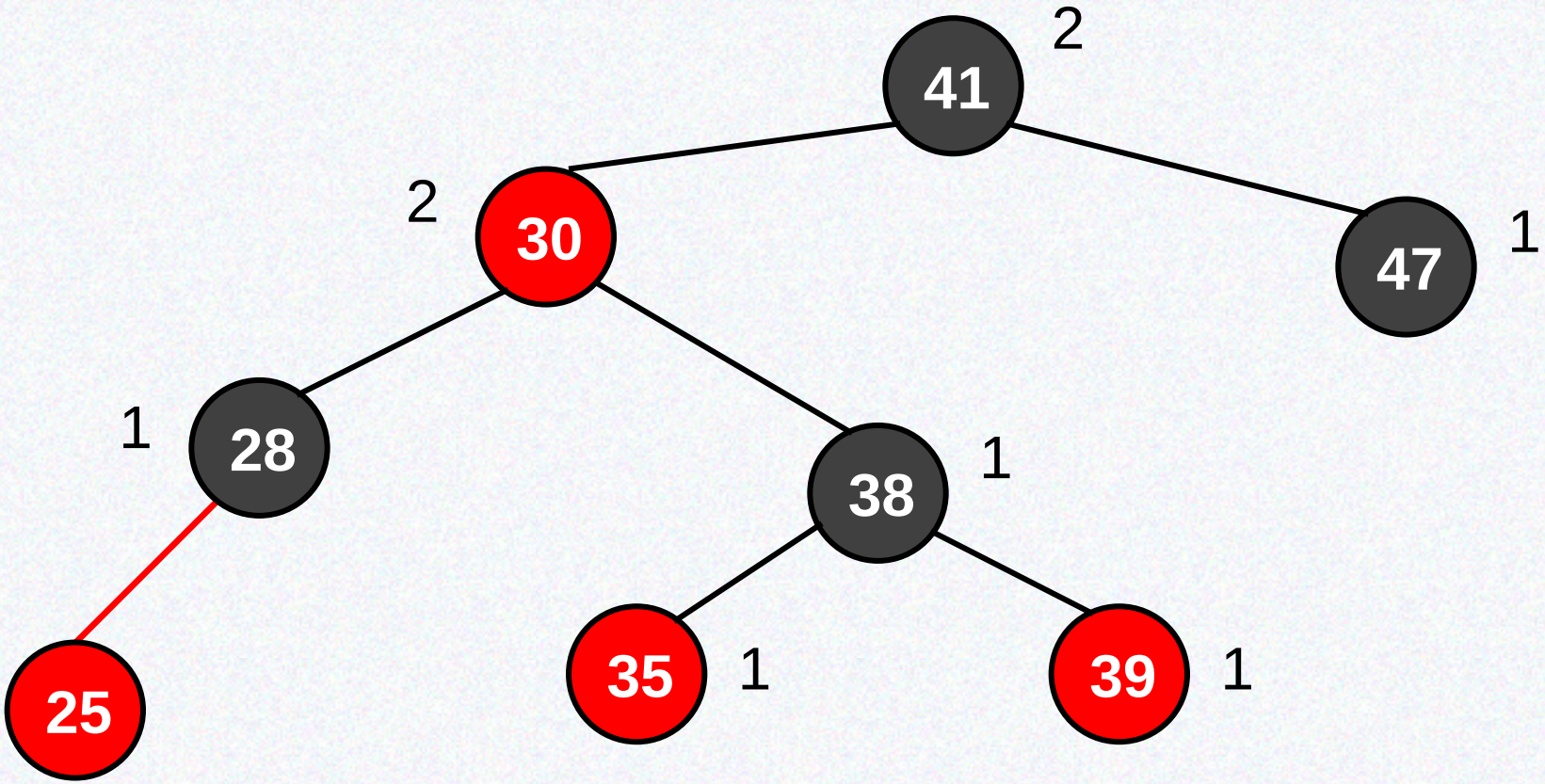
Новый элемент 25

Примеры вставки



Новый элемент 25

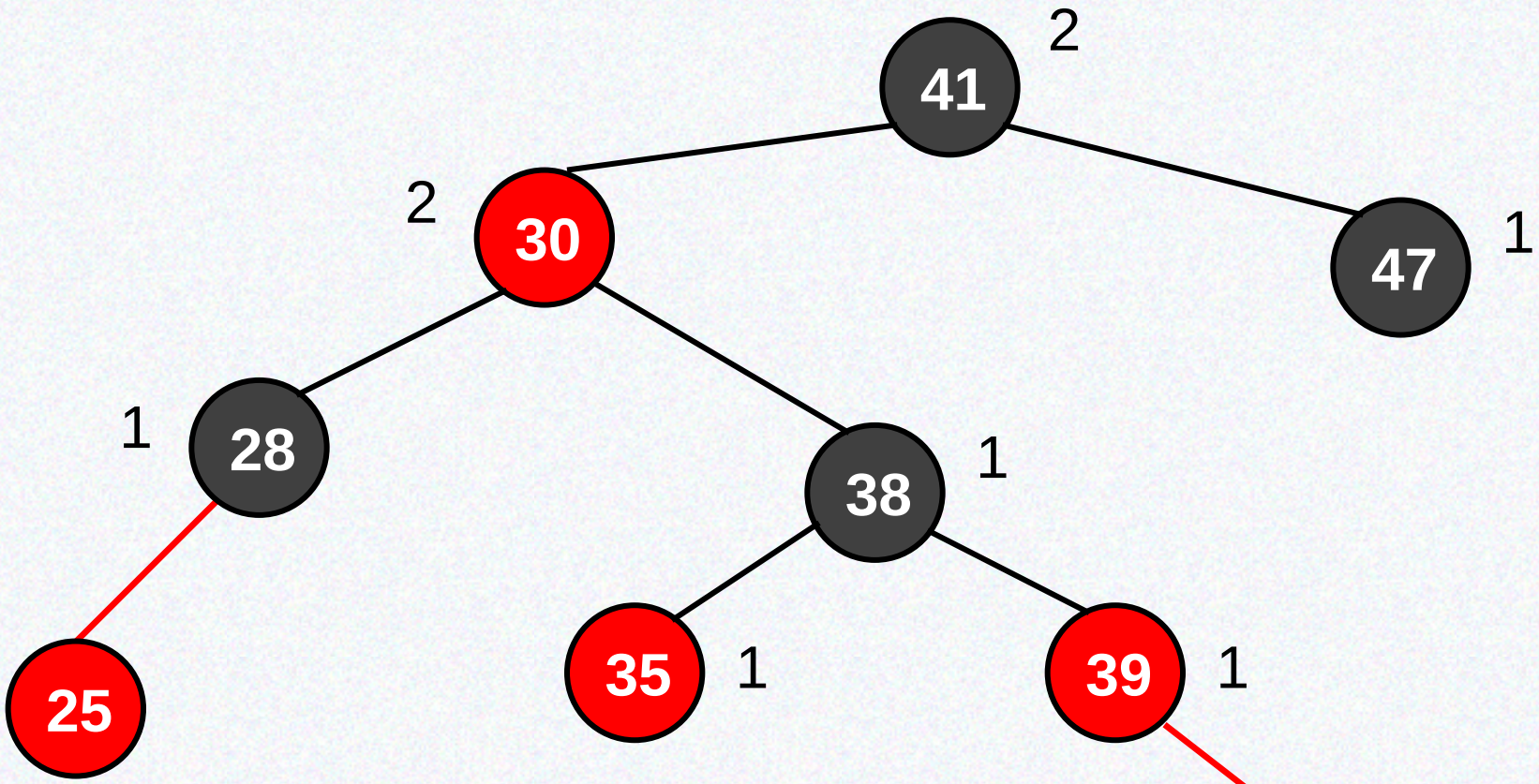
Примеры вставки



Новый элемент 25

Новый элемент 40

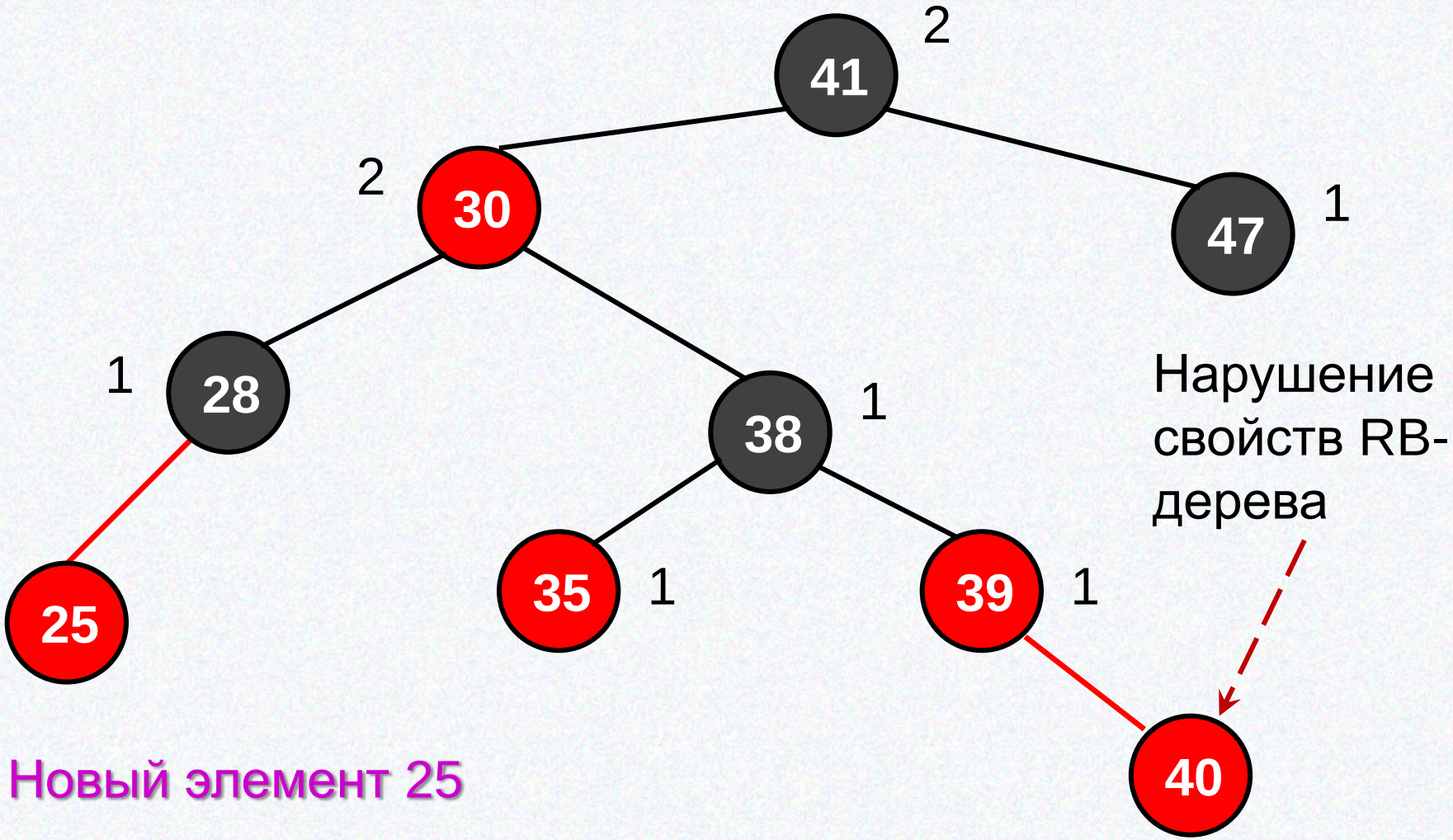
Примеры вставки



Новый элемент 25

Новый элемент 40

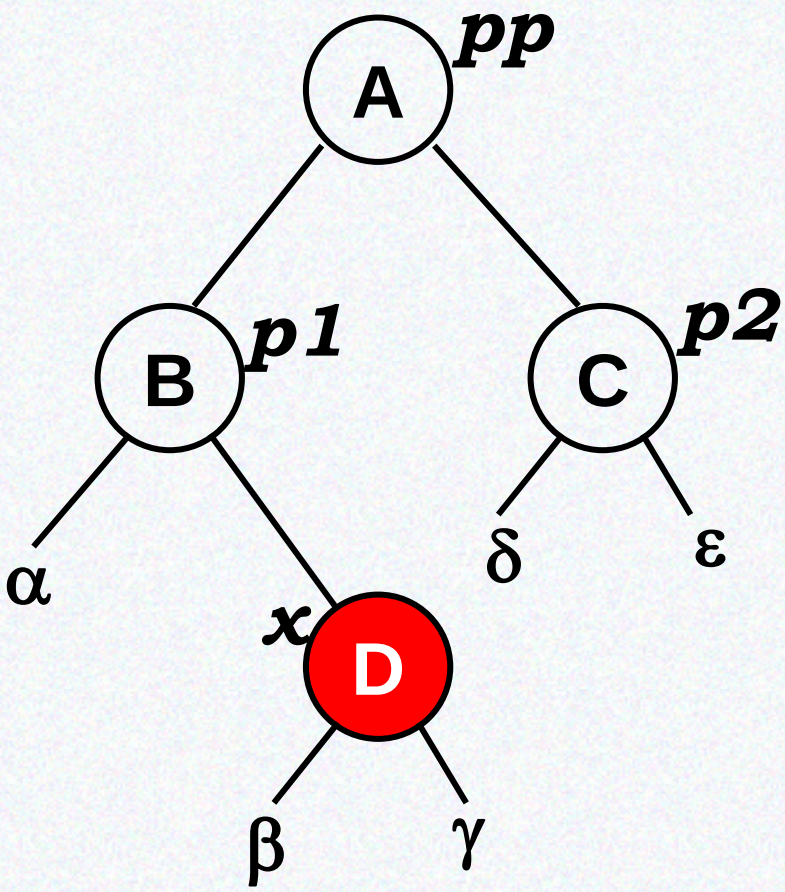
Примеры вставки



Новый элемент 25

Новый элемент 40

Восстановление свойств дерева

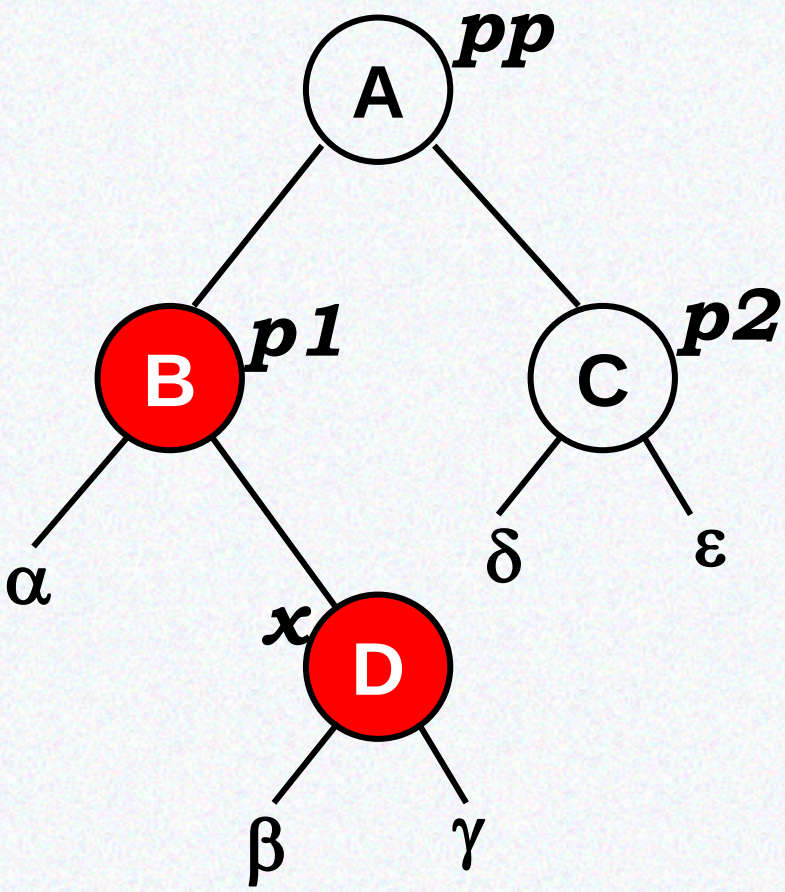


Новый узел – x

$p1 = x->parent$

узел $p1$ – красный

Восстановление свойств дерева

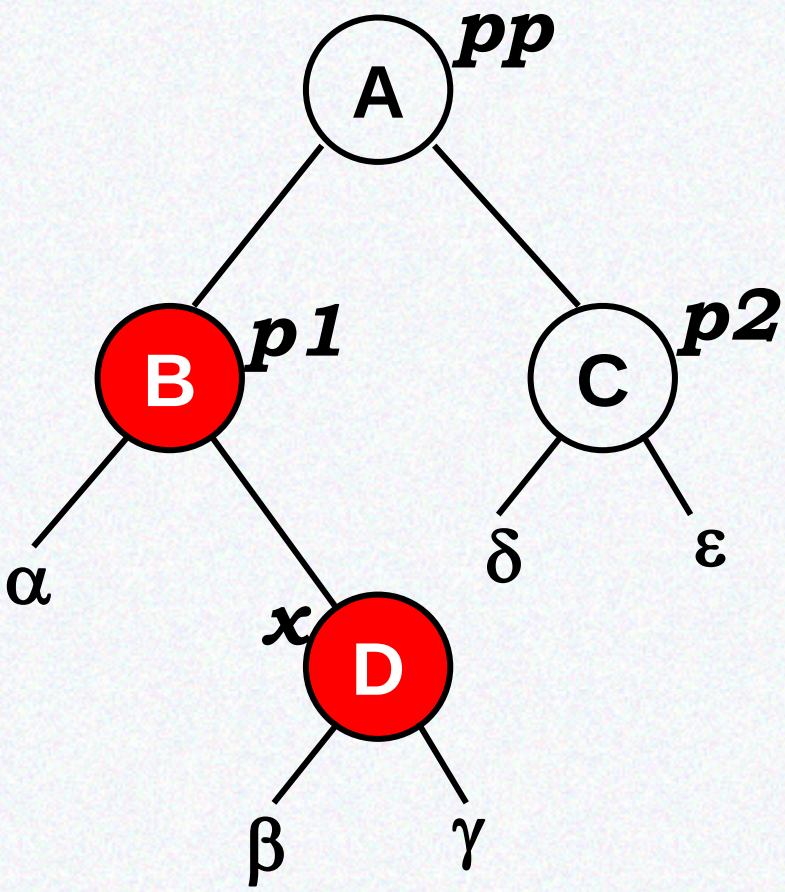


Новый узел – x

$p1 = x->parent$

узел $p1$ – красный

Восстановление свойств дерева



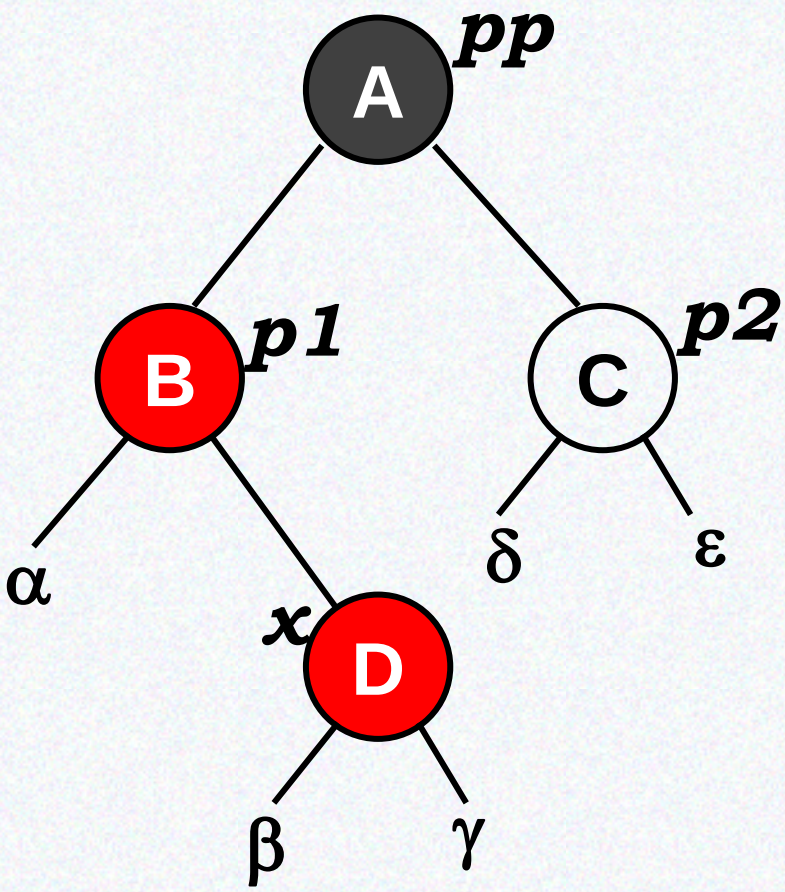
Новый узел – x

$p1 = x->parent$

узел $p1$ – красный

$pp = p1->parent$ – черный

Восстановление свойств дерева



Новый узел – *x*

p1 = x->parent

узел *p1* – красный

pp = p1->parent – черный

p2 = pp->right

(или *p2 = pp->left*)

Случай 1

Узел *p1* – красный

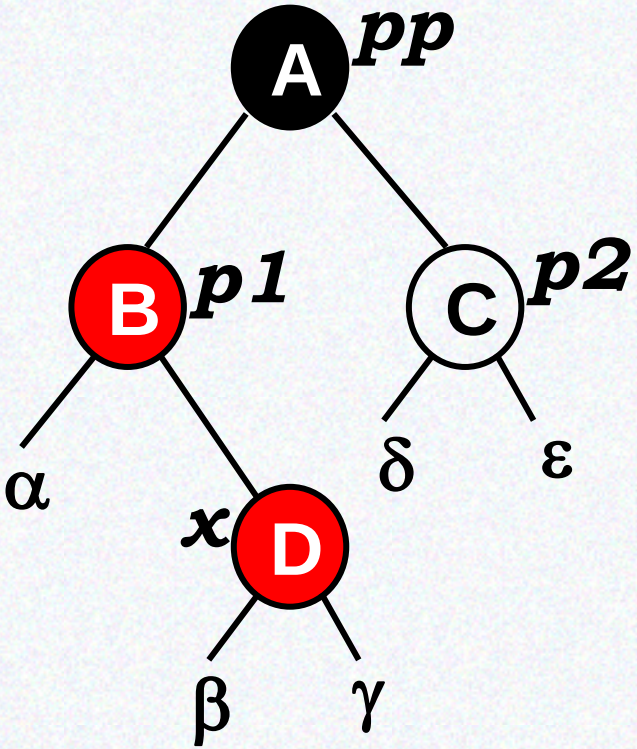
Узел *p1* – в левом поддереве

Узел *pp* – черный

Узлы α, β, γ, δ, ε – черные

Узел *p2* – в правом поддереве

Узел *p2* – красный



Случай 1

Узел *p1* – красный

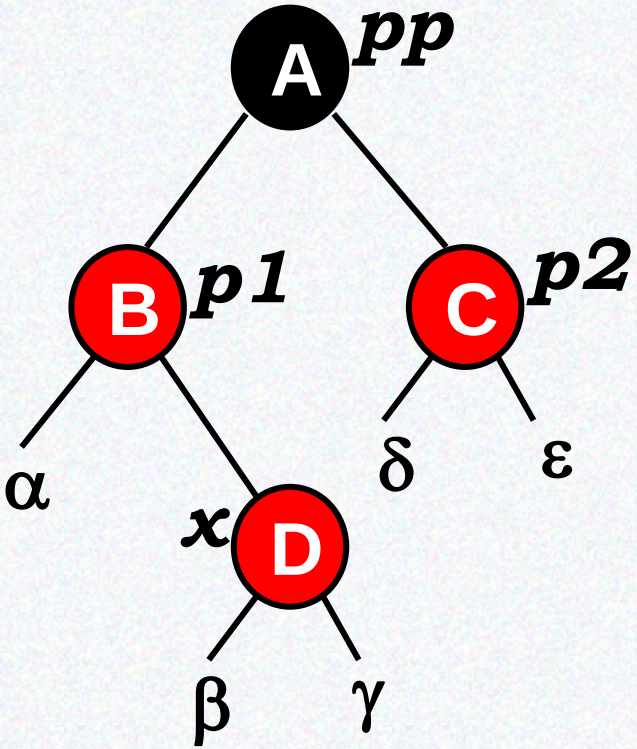
Узел *p1* – в левом поддереве

Узел *pp* – черный

Узлы α, β, γ, δ, ε – черные

Узел *p2* – в правом поддереве

Узел *p2* – красный



Случай 1

Узел *p1* – красный

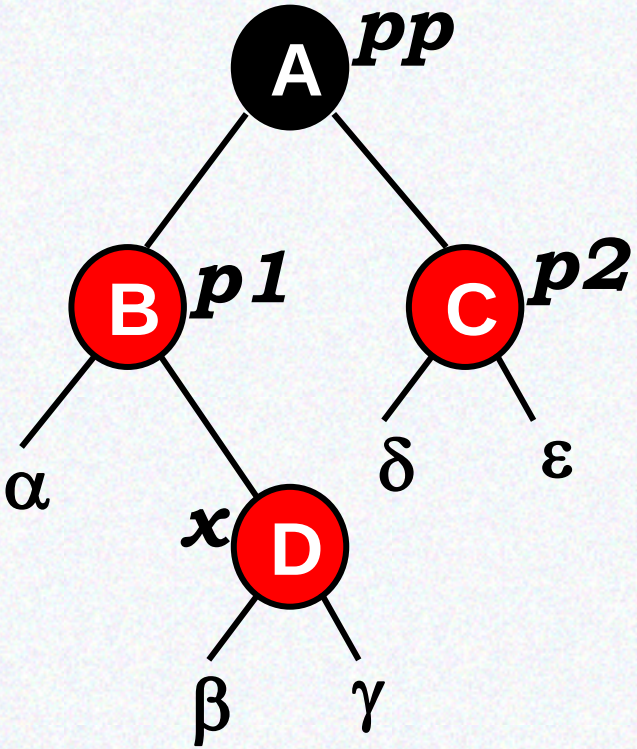
Узел *p1* – в левом поддереве

Узел *pp* – черный

Узлы α, β, γ, δ, ε – черные

Узел *p2* – в правом поддереве

Узел *p2* – красный



Коррекция:

перекрасить узлы:

p1 и *p2* в черный цвет,

pp – в красный

Случай 1

Узел *p1* – красный

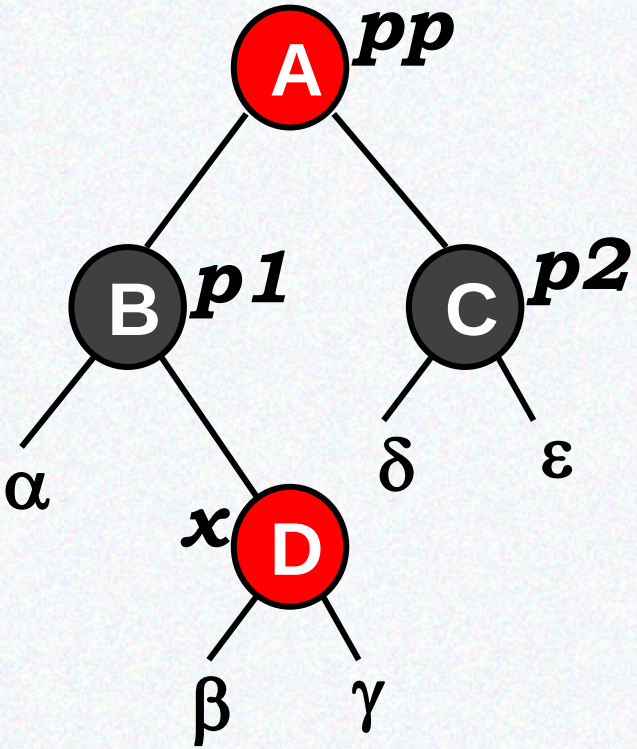
Узел *p1* – в левом поддереве

Узел *pp* – черный

Узлы α, β, γ, δ, ε – черные

Узел *p2* – в правом поддереве

Узел *p2* – красный



Коррекция:

перекрасить узлы:

p1 и *p2* в черный цвет,

pp – в красный

Случай 1

Узел $p1$ – красный

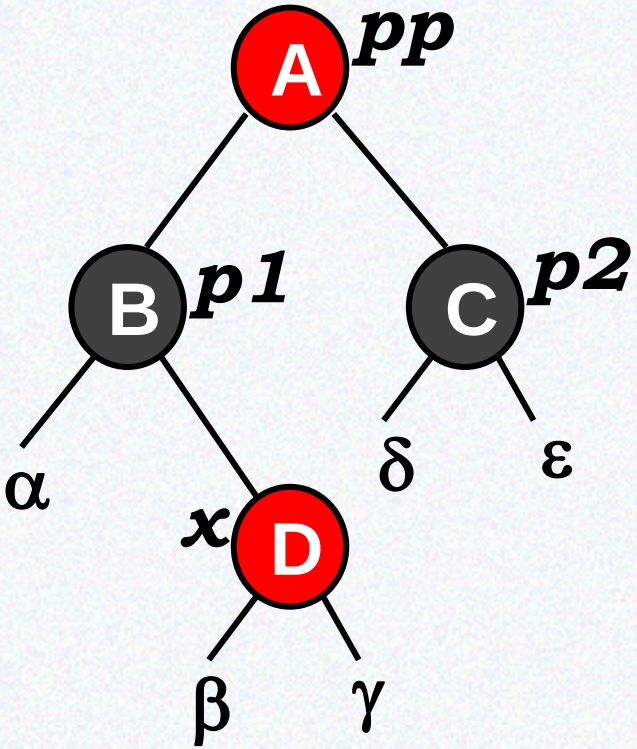
Узел $p1$ – в левом поддереве

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел $p2$ – в правом поддереве

Узел $p2$ – красный



Коррекция:

перекрасить узлы:

$p1$ и $p2$ в черный цвет,

pp – в красный

Переустановить узел x

Случай 1

Узел $p1$ – красный

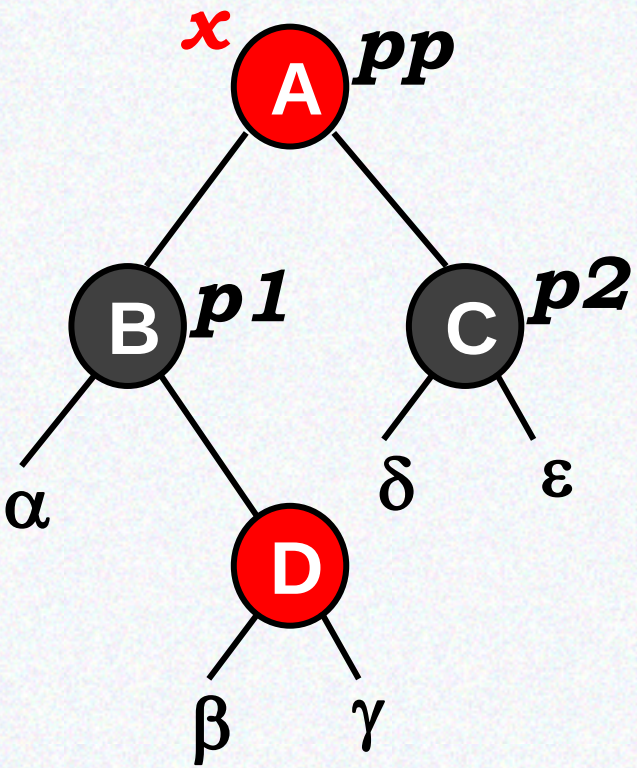
Узел $p1$ – в левом поддереве

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел $p2$ – в правом поддереве

Узел $p2$ – красный



Коррекция:

перекрасить узлы:

$p1$ и $p2$ в черный цвет,

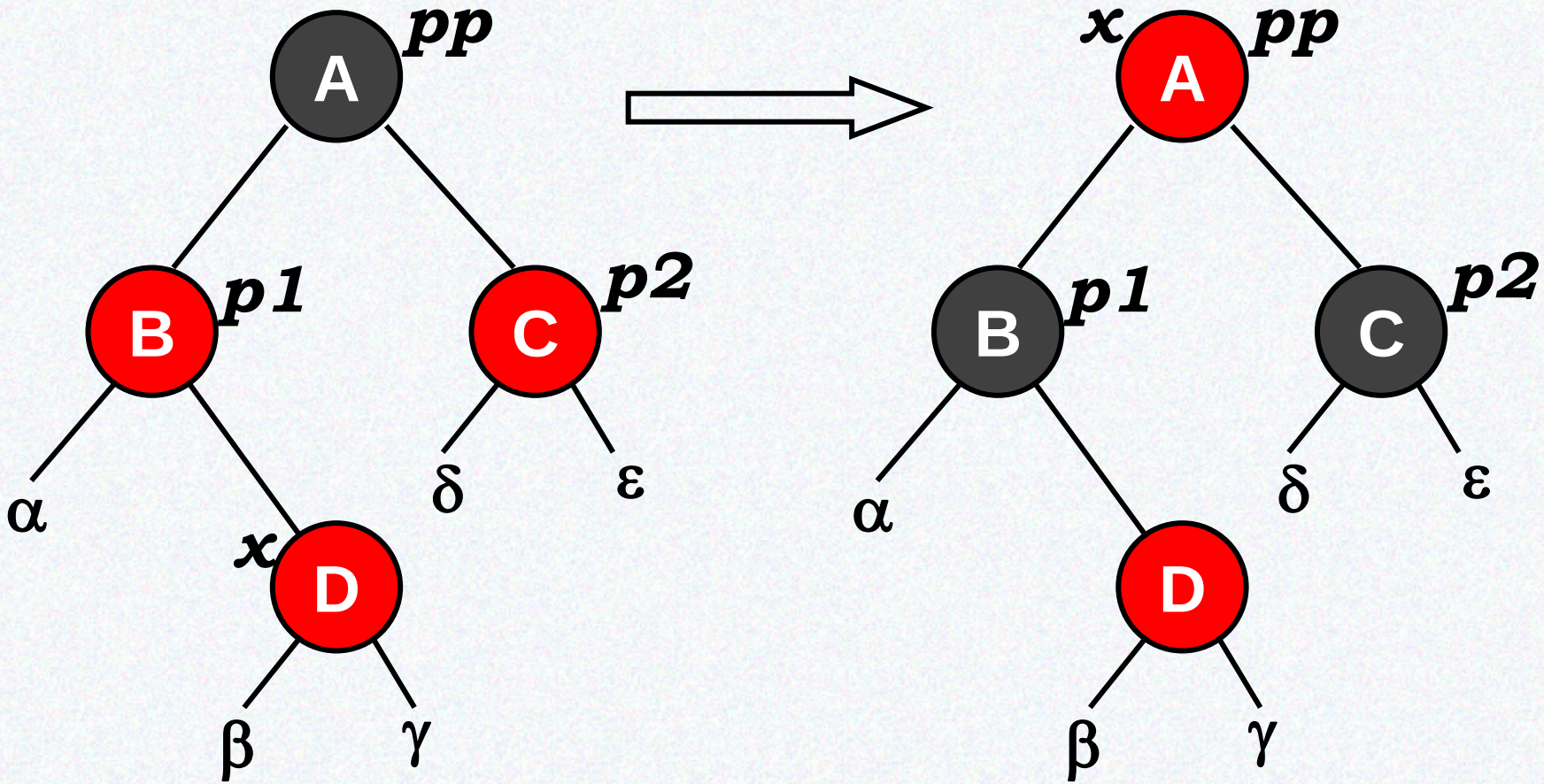
pp – в красный

Переустановить узел x

Продолжить коррекцию дерева

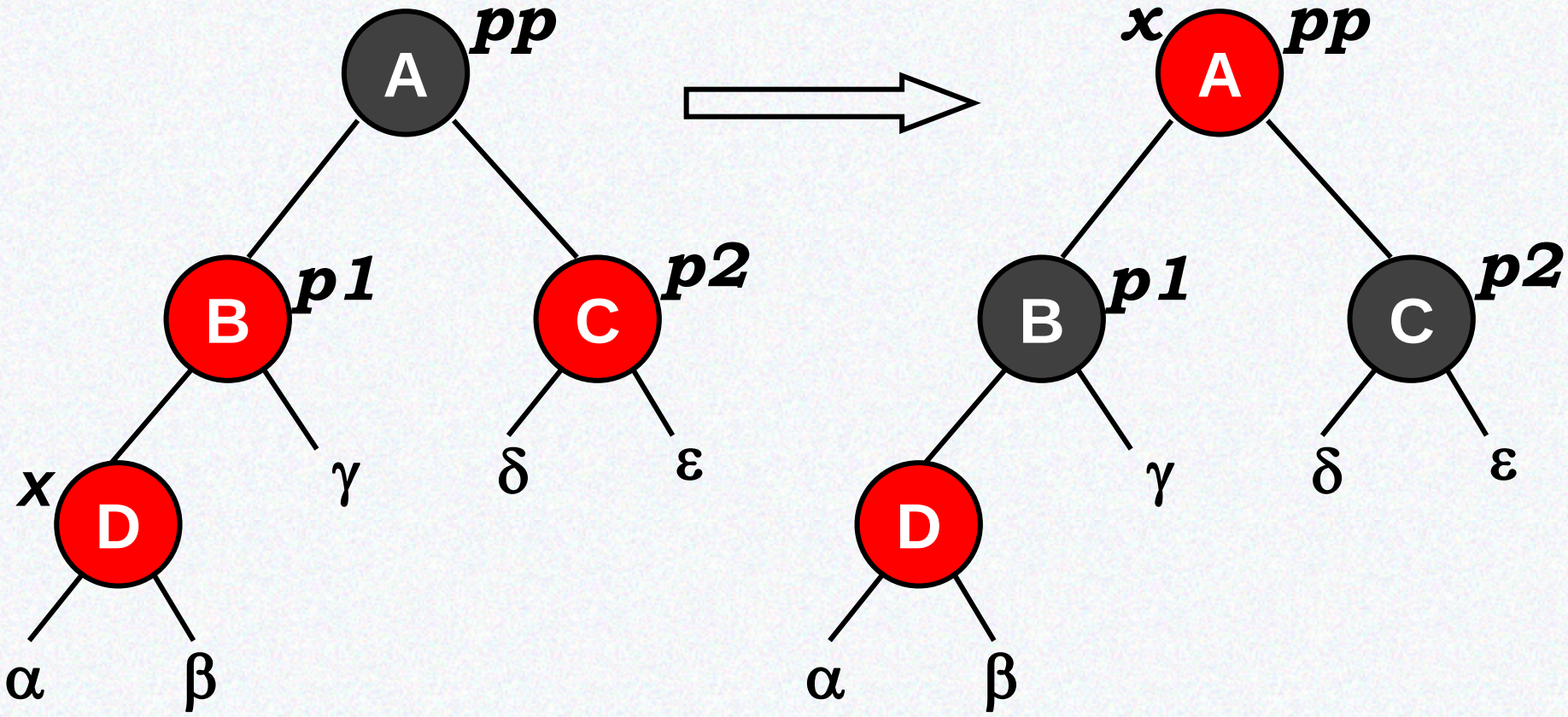
Случай 1

Узел *p1* – в левом поддереве

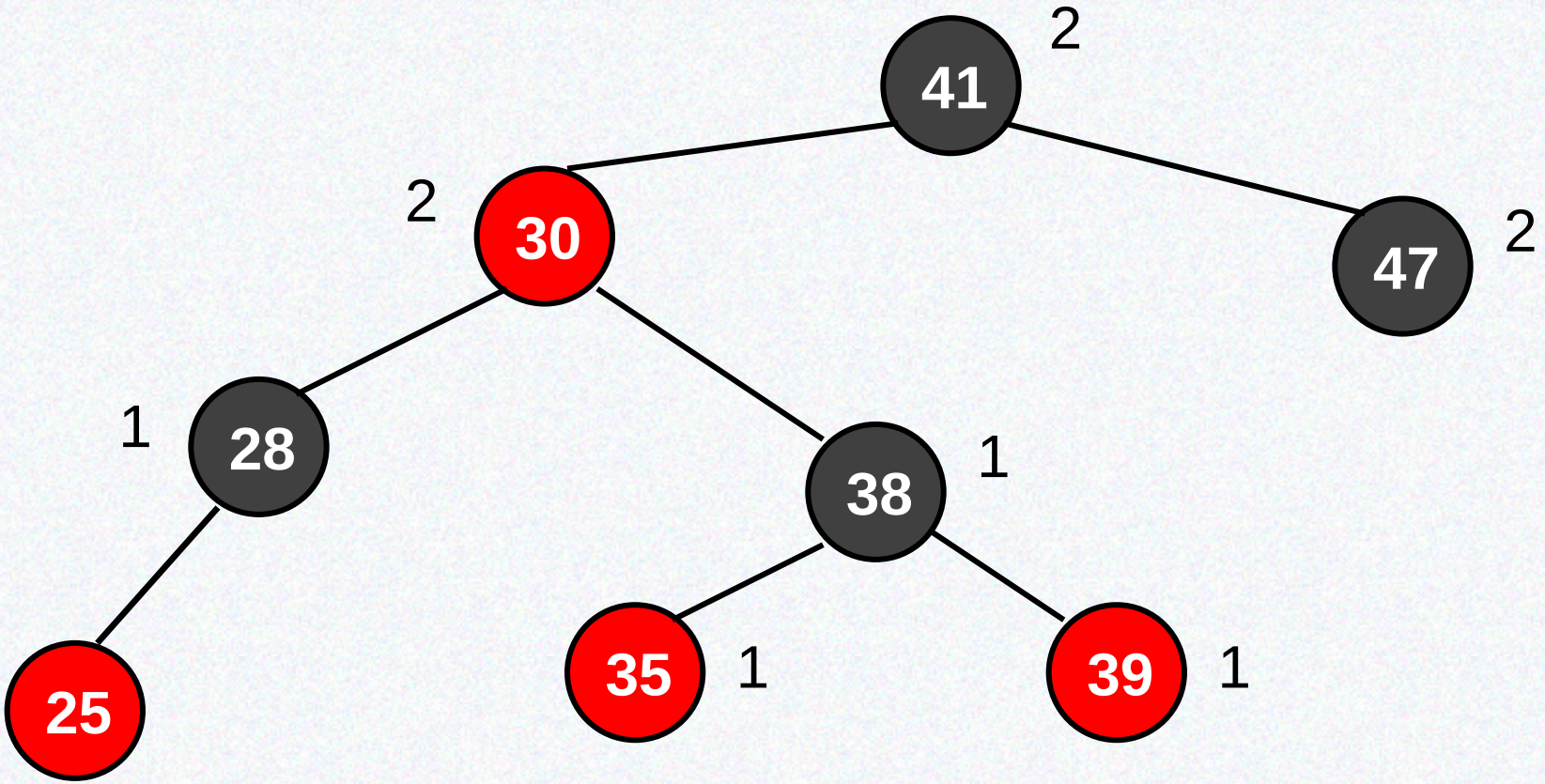


Случай 1

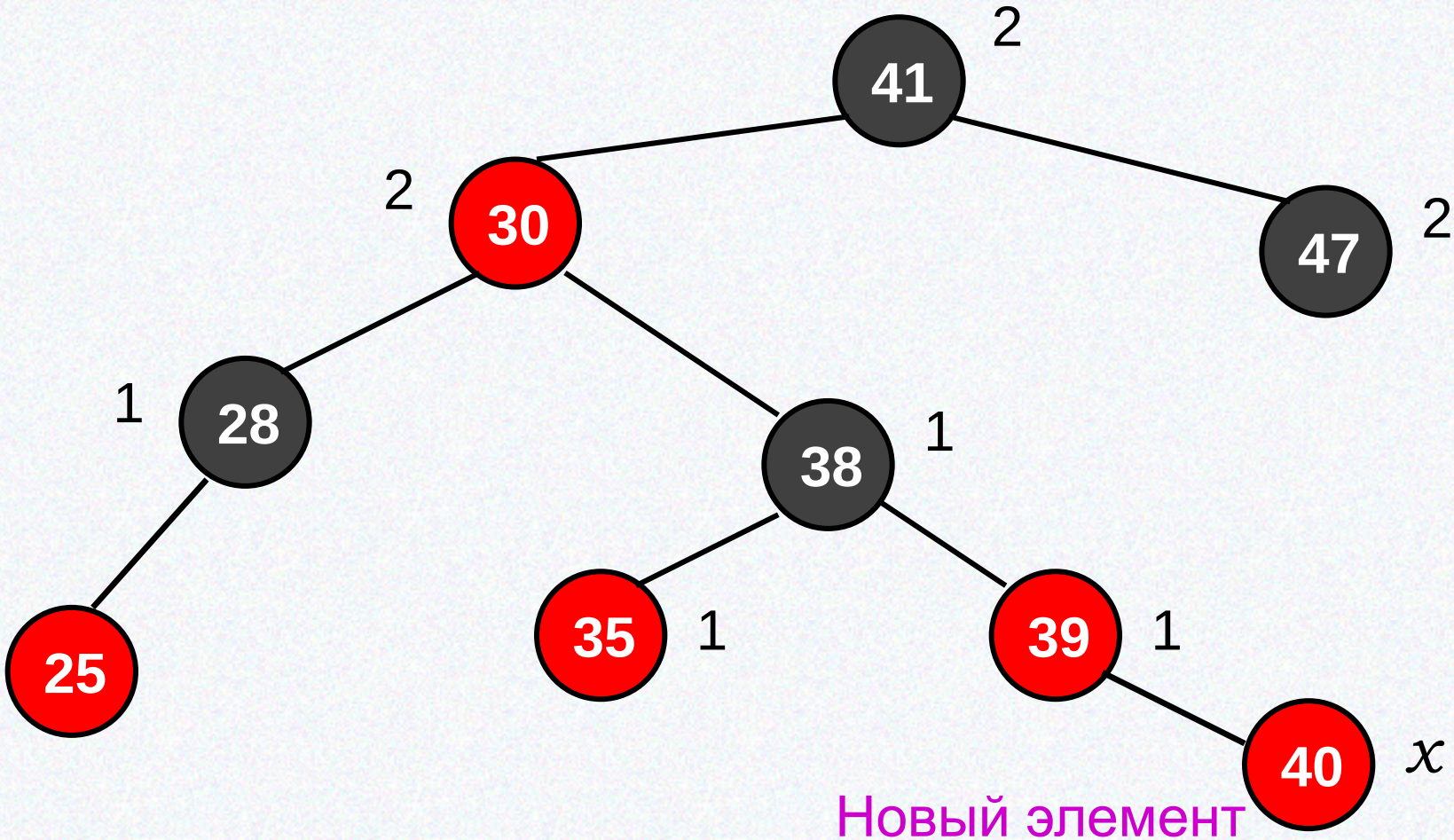
Узел *p1* – в левом поддереве



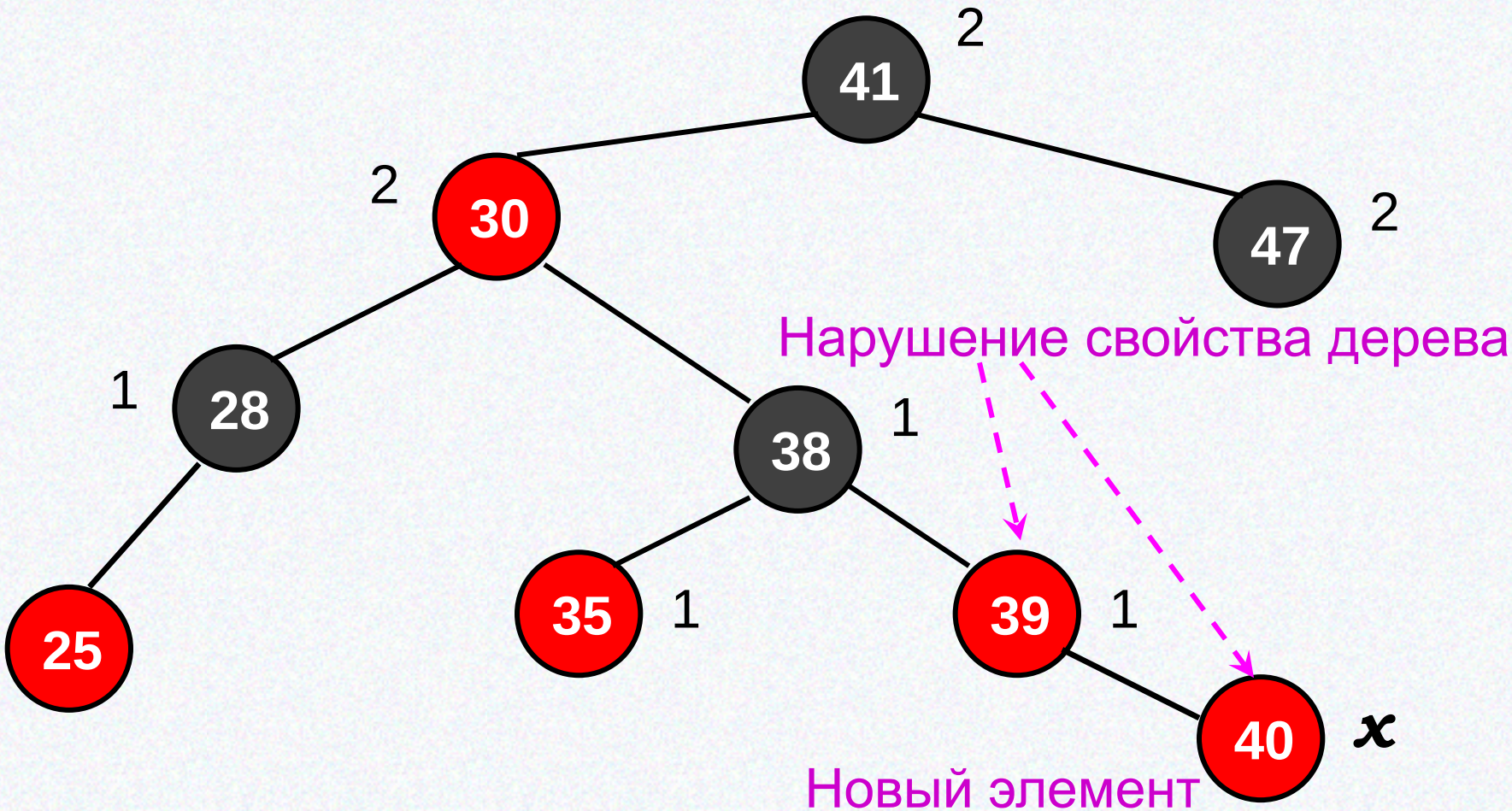
Вставка узла 40



Вставка узла 40

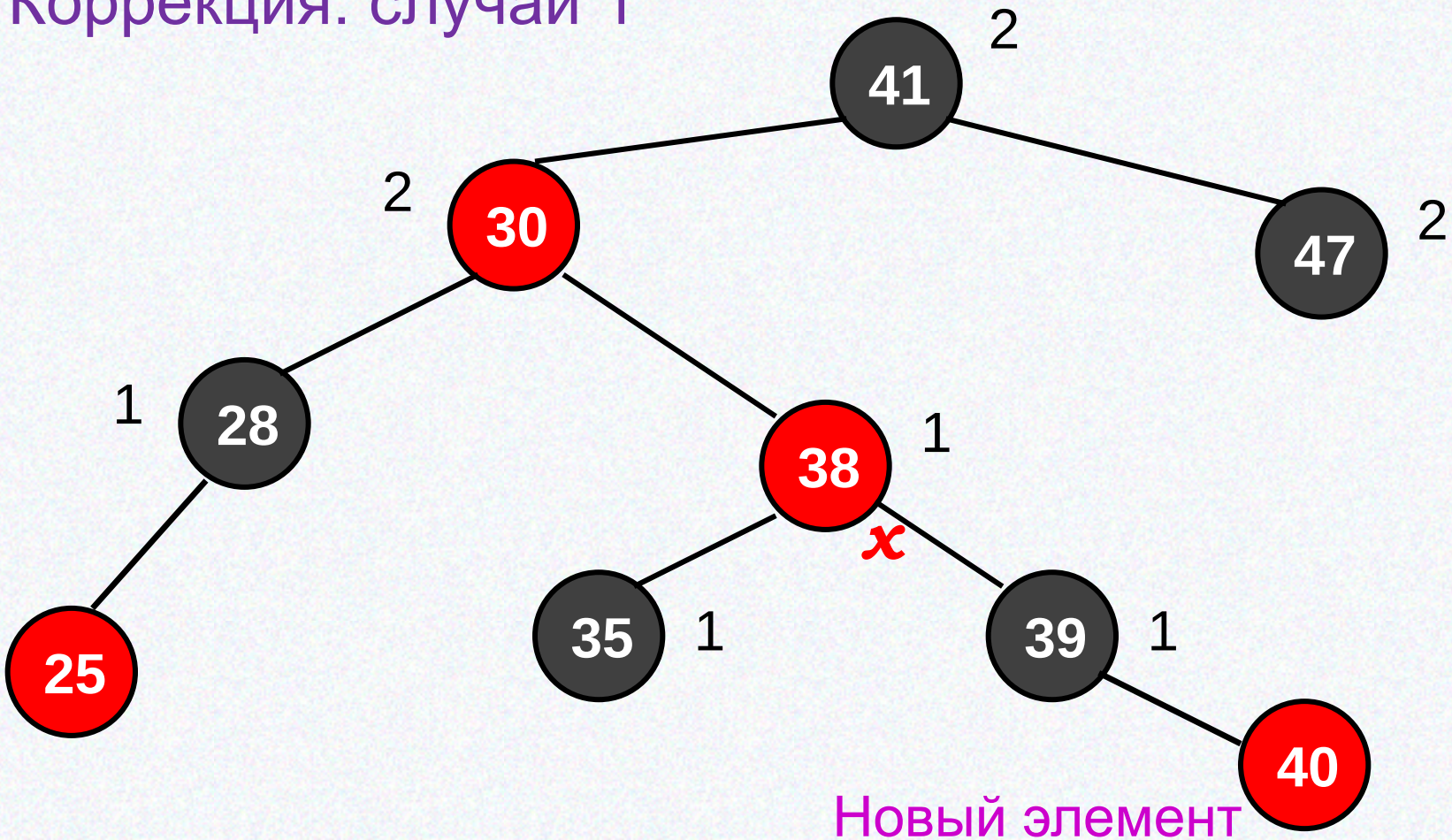


Вставка узла 40



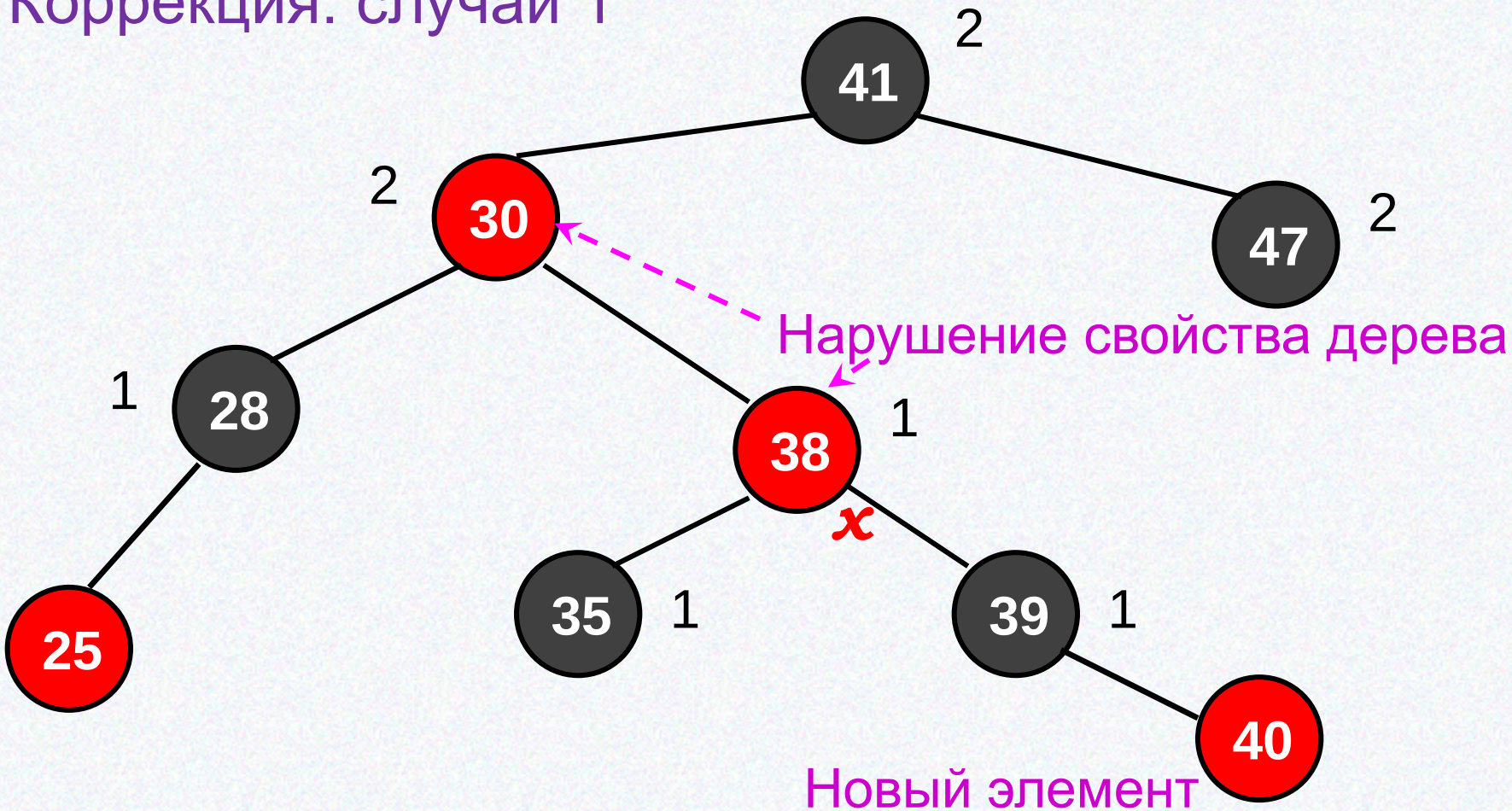
Вставка узла 40

Коррекция: случай 1



Вставка узла 40

Коррекция: случай 1



Случай 2

Узел $p1$ – красный

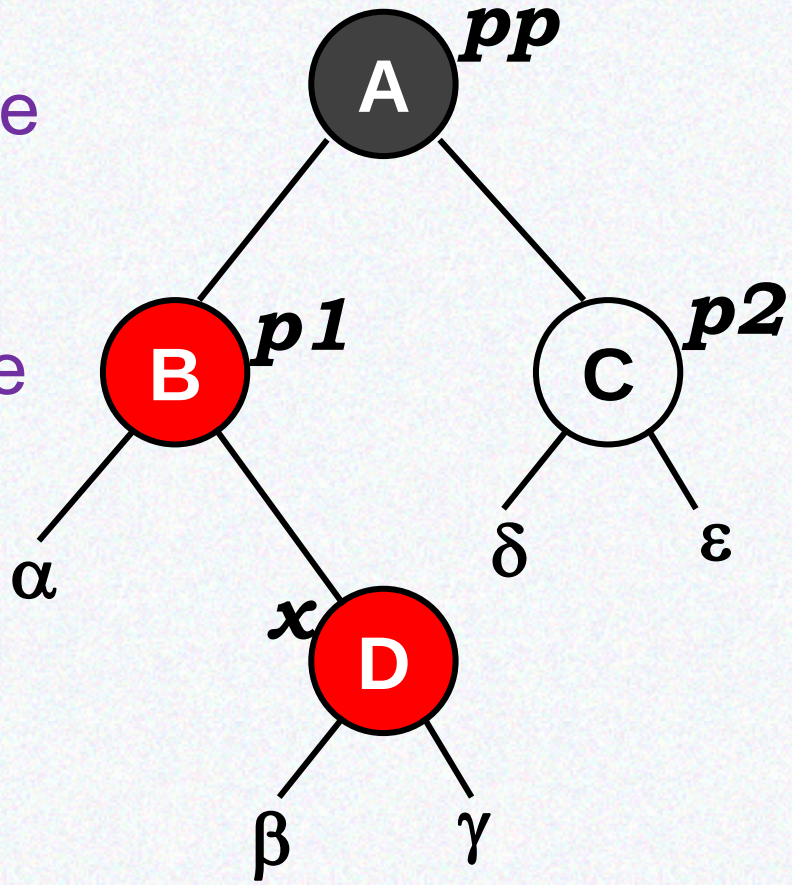
Узел $p1$ – в левом поддереве

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел x – в правом поддереве узла $p1$

Узел $p2$ – черный



Случай 2

Узел $p1$ – красный

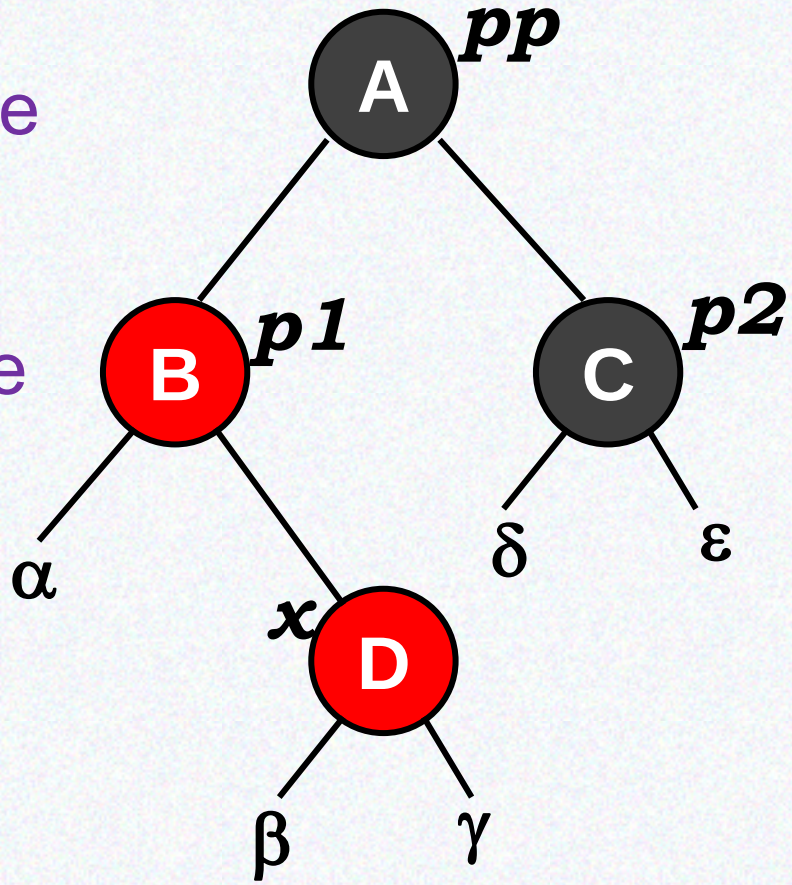
Узел $p1$ – в левом поддереве

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел x – в правом поддереве узла $p1$

Узел $p2$ – черный



Случай 2

Узел $p1$ – красный

Узел $p1$ – в левом поддереве

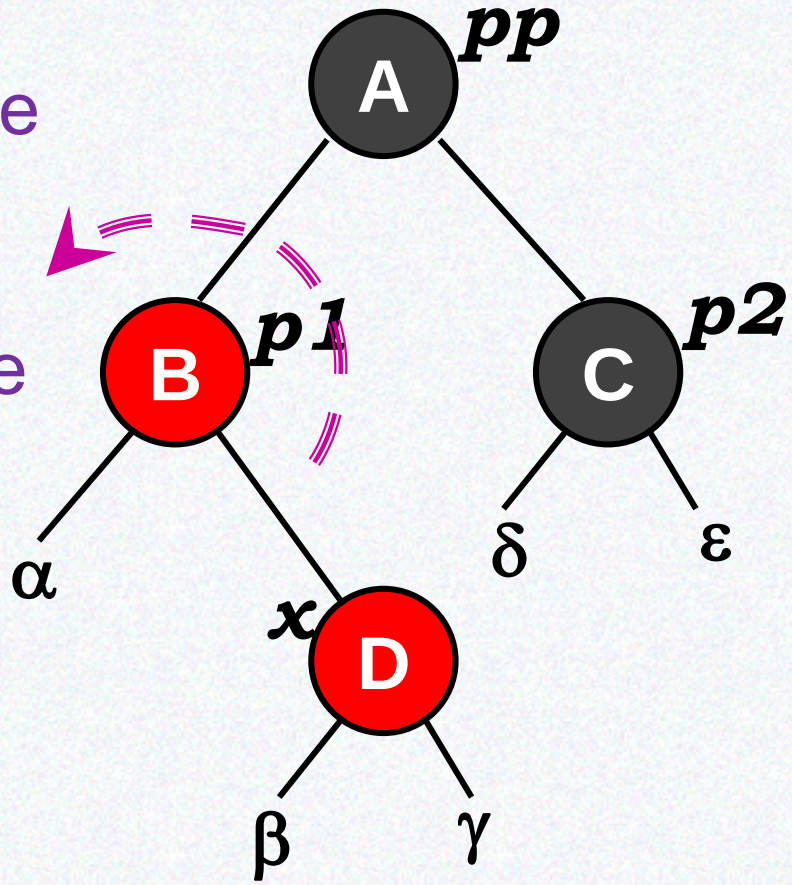
Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел x – в правом поддереве узла $p1$

Узел $p2$ – черный

Коррекция:
левый поворот
вокруг узла $p1$



Случай 2

Узел $p1$ – красный

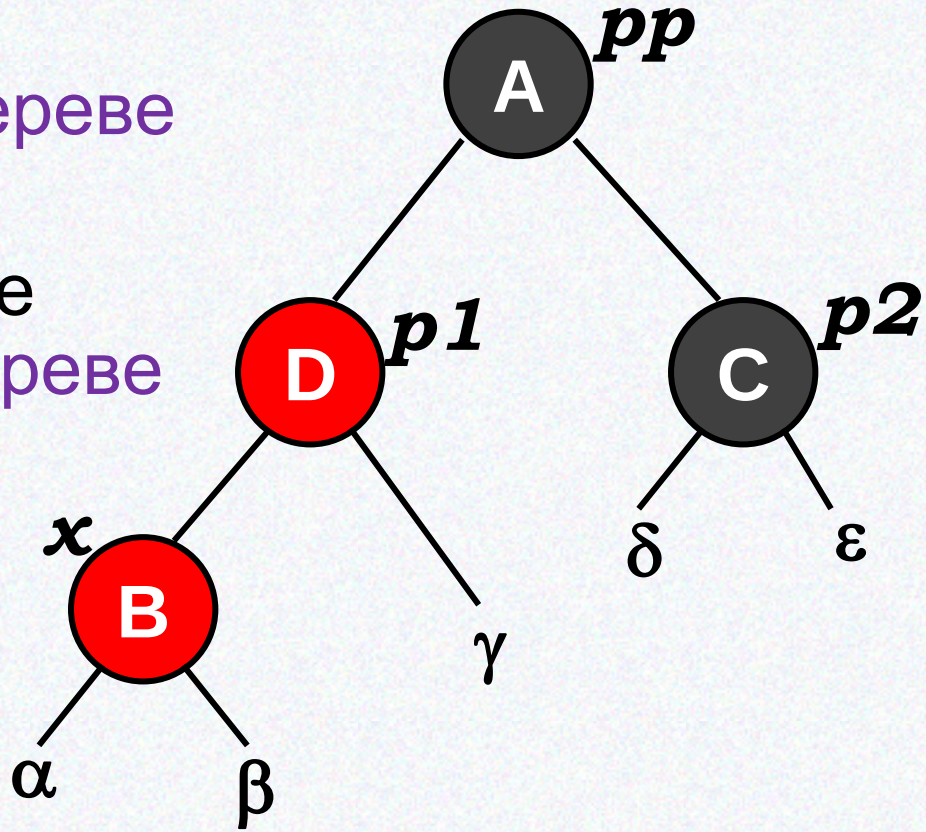
Узел $p1$ – в левом поддереве

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел x – в правом поддереве узла $p1$

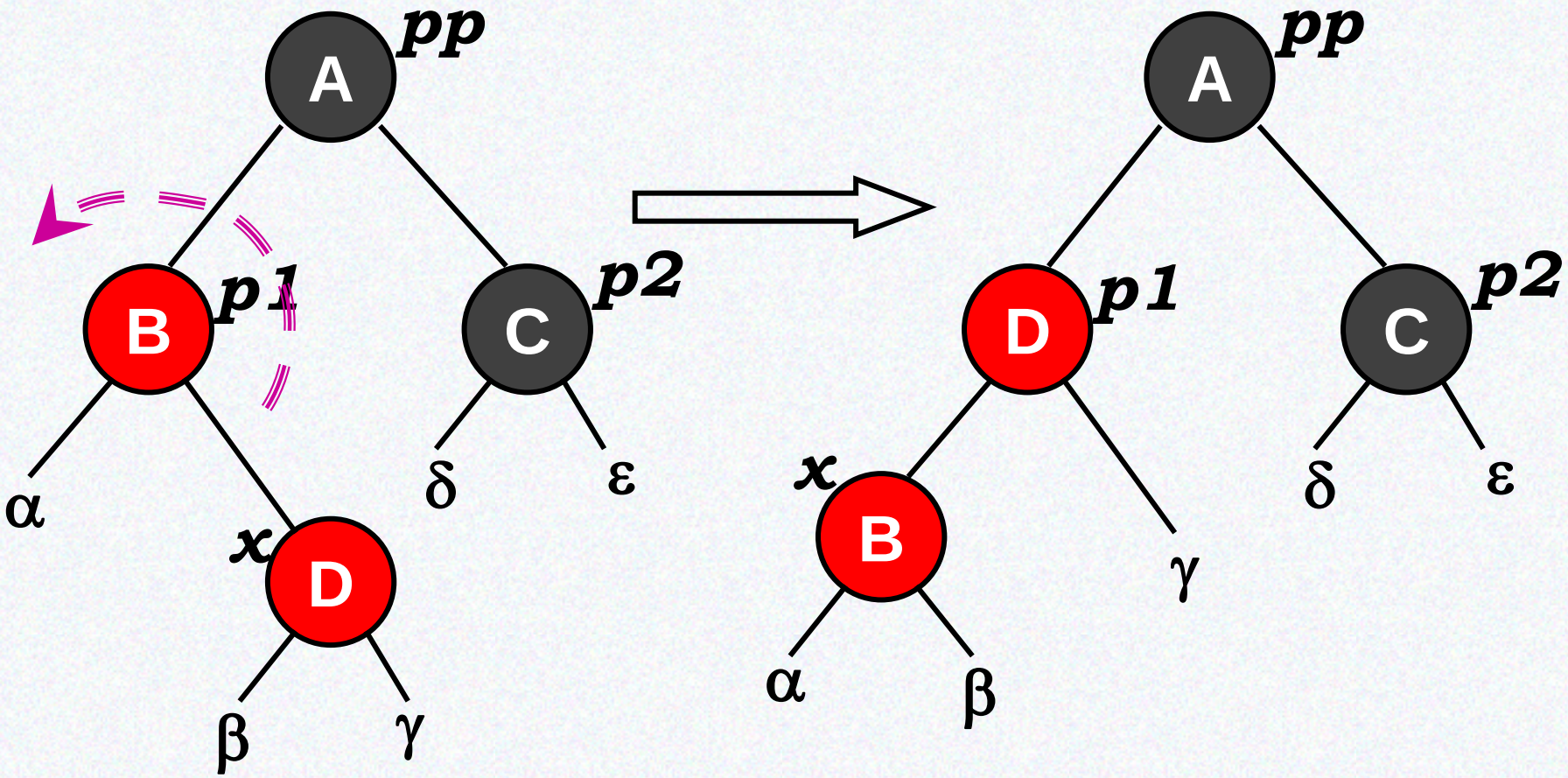
Узел $p2$ – черный



Коррекция:
левый поворот вокруг узла $p1$

Переходим к случаю 3

Случай 2



Случай 2

Узел $p1$ – красный

Узел $p1$ – в правом поддереве

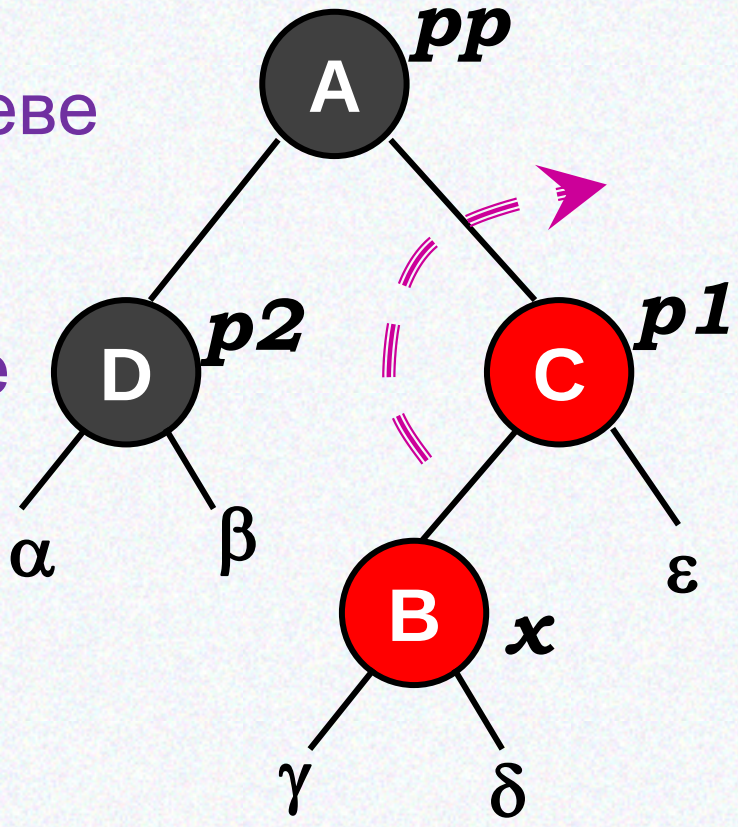
Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел x – в левом поддереве узла $p1$

Узел $p2$ – черный

Коррекция:
правый поворот вокруг узла $p1$



Случай 2

Узел $p1$ – красный

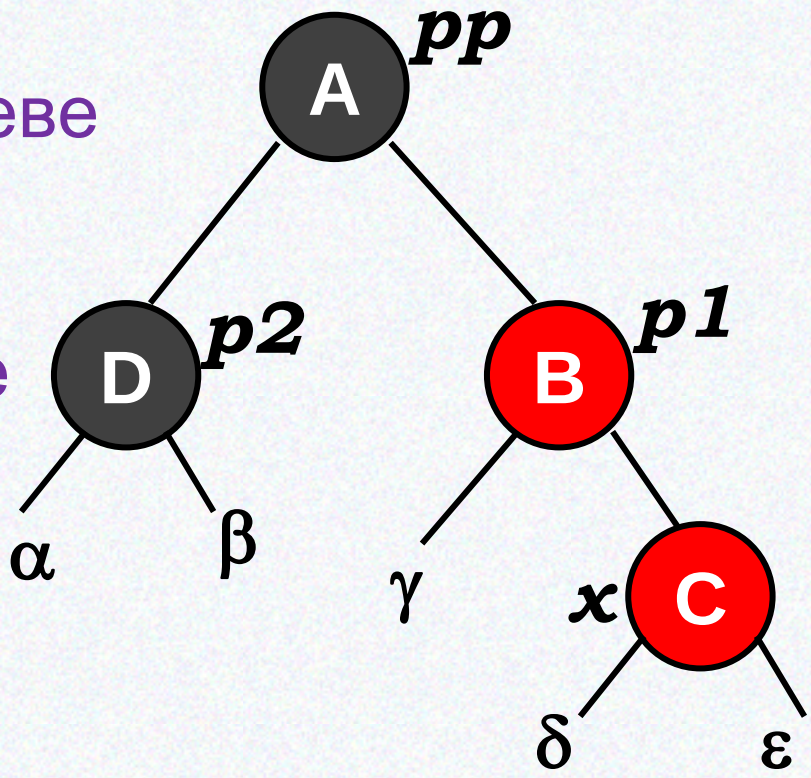
Узел $p1$ – в правом поддереве

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел x – в левом поддереве узла $p1$

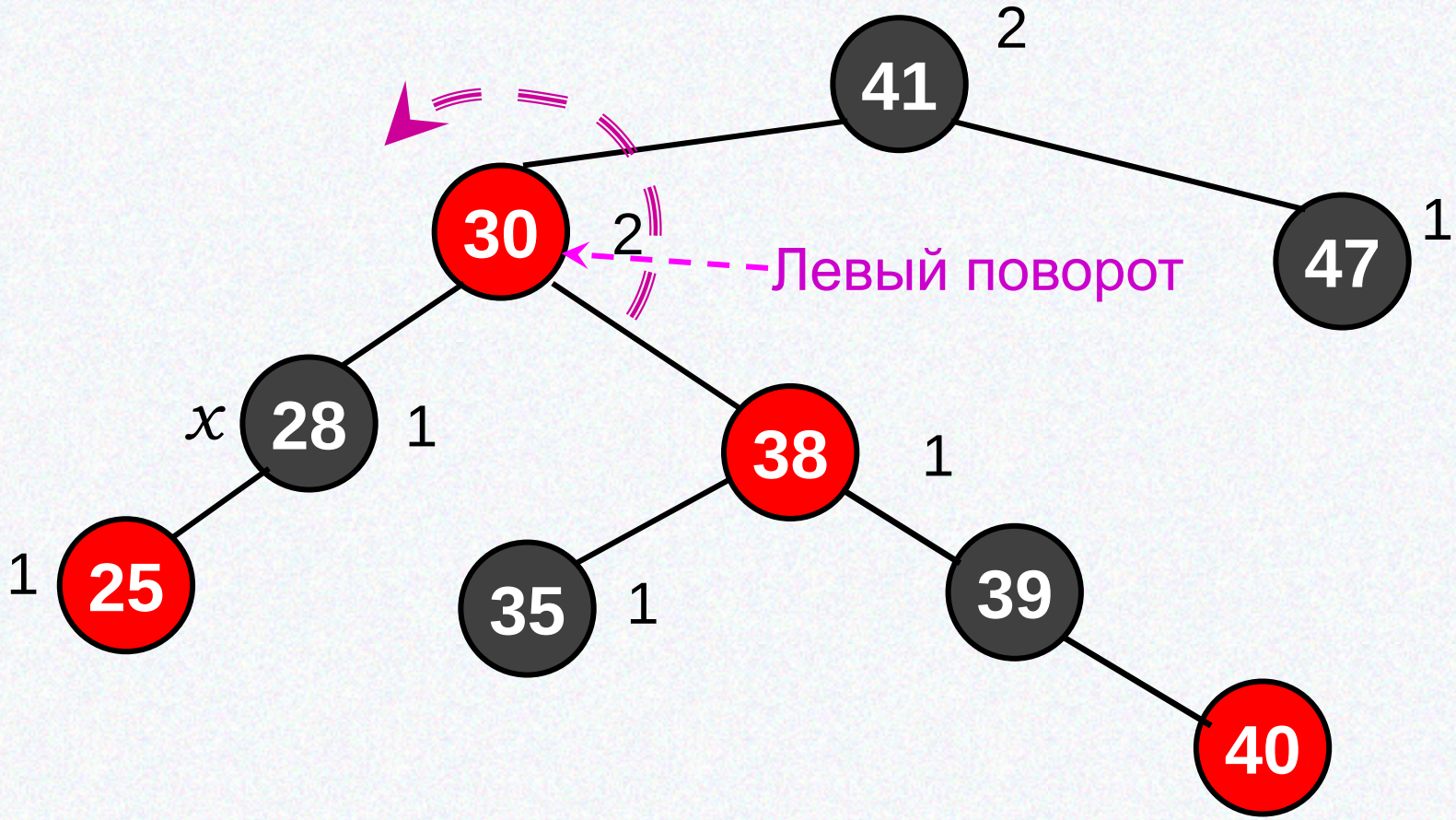
Узел $p2$ – черный



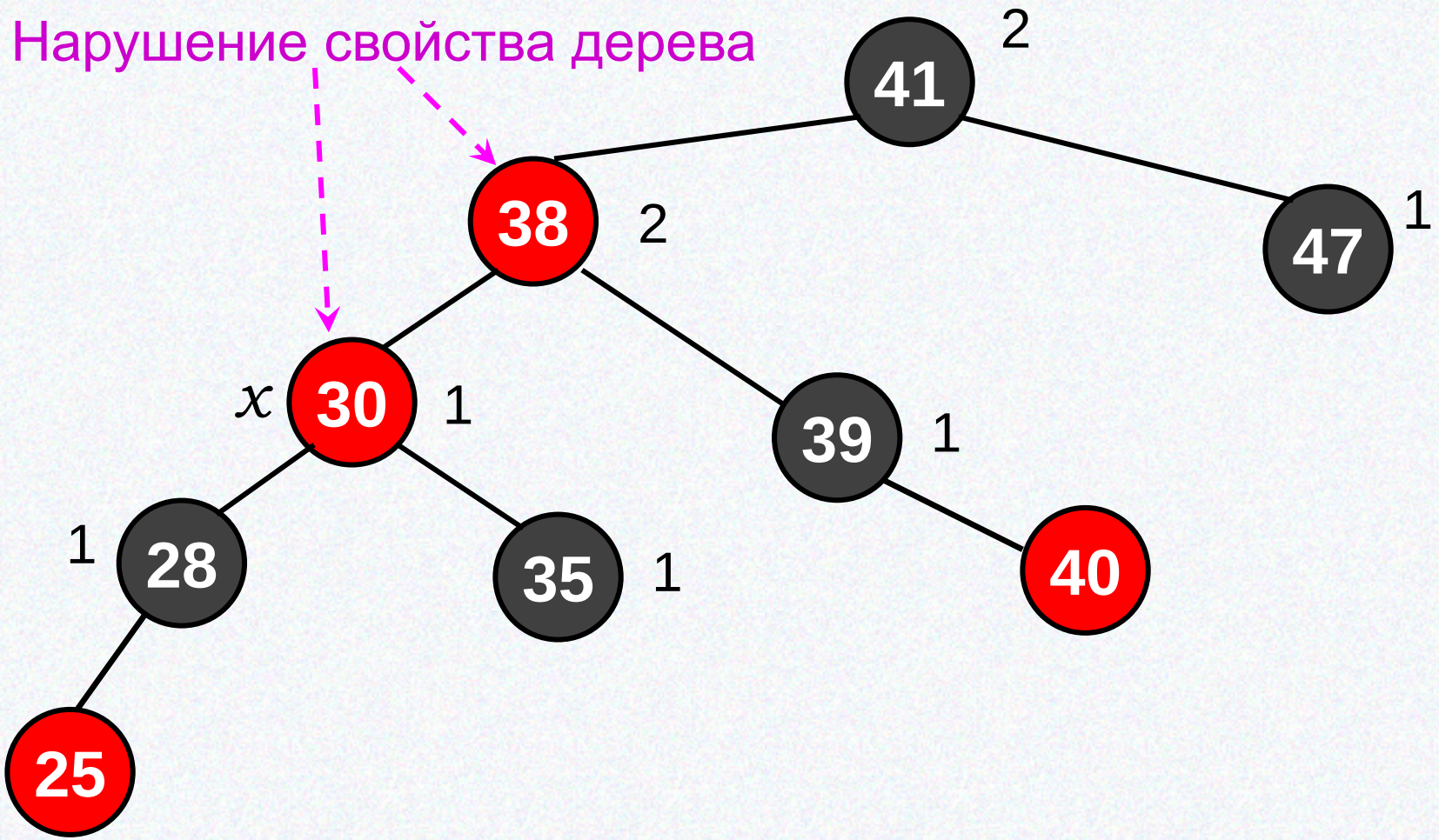
Коррекция:
правый поворот вокруг узла $p1$

Переходим к случаю 3

Вставка узла 40 (продолжение)



Вставка узла 40 (продолжение)



Случай 3

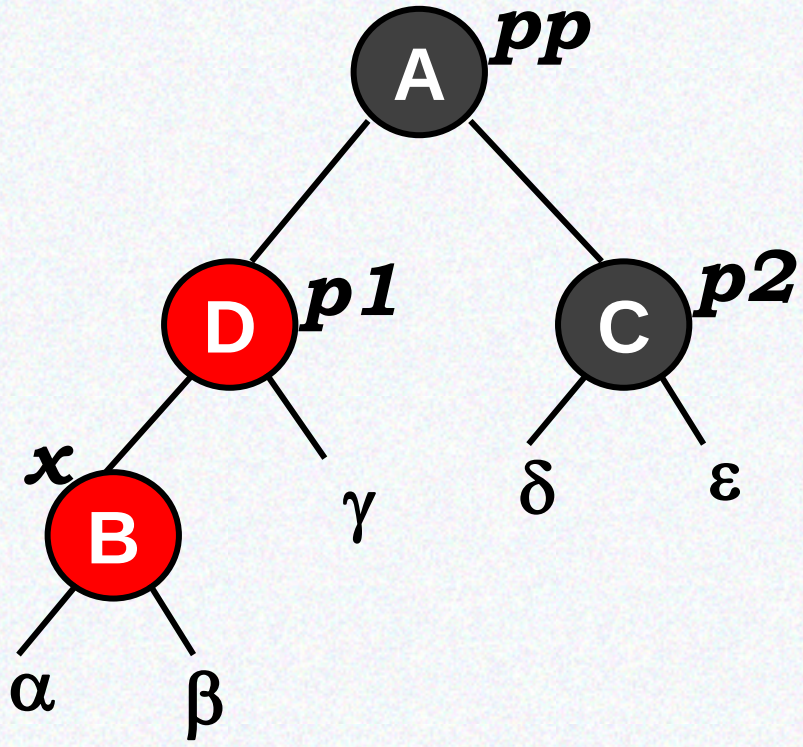
Узел $p1$ – красный

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел $p2$ – черный

Узел x – в левом поддереве узла $p1$



Случай 3

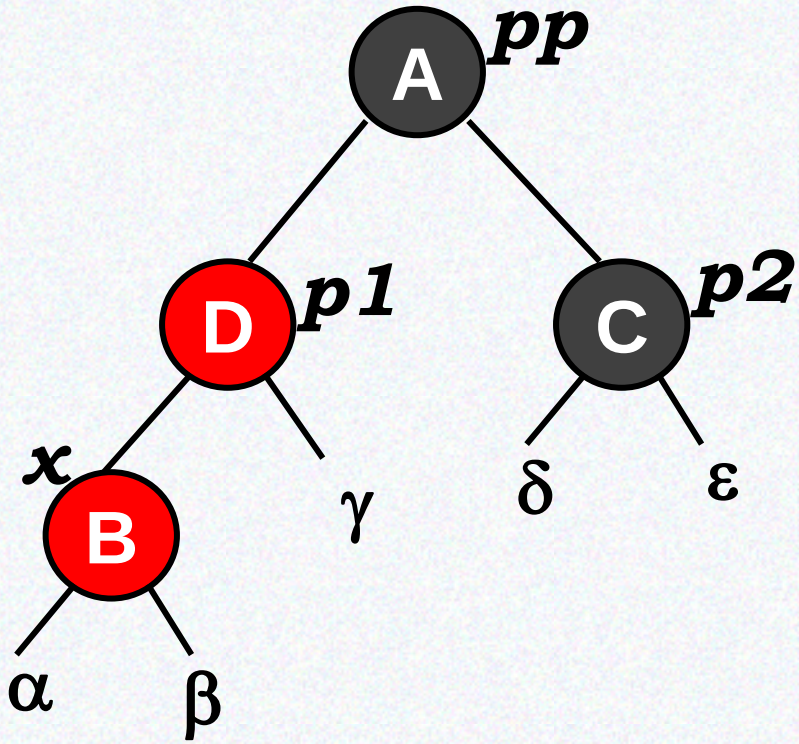
Узел $p1$ – красный

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел $p2$ – черный

Узел x – в левом поддереве узла $p1$



Коррекция:

шаг 1 – перекрасить узел $p1$ в черный цвет,
узел pp – в красный цвет

Случай 3

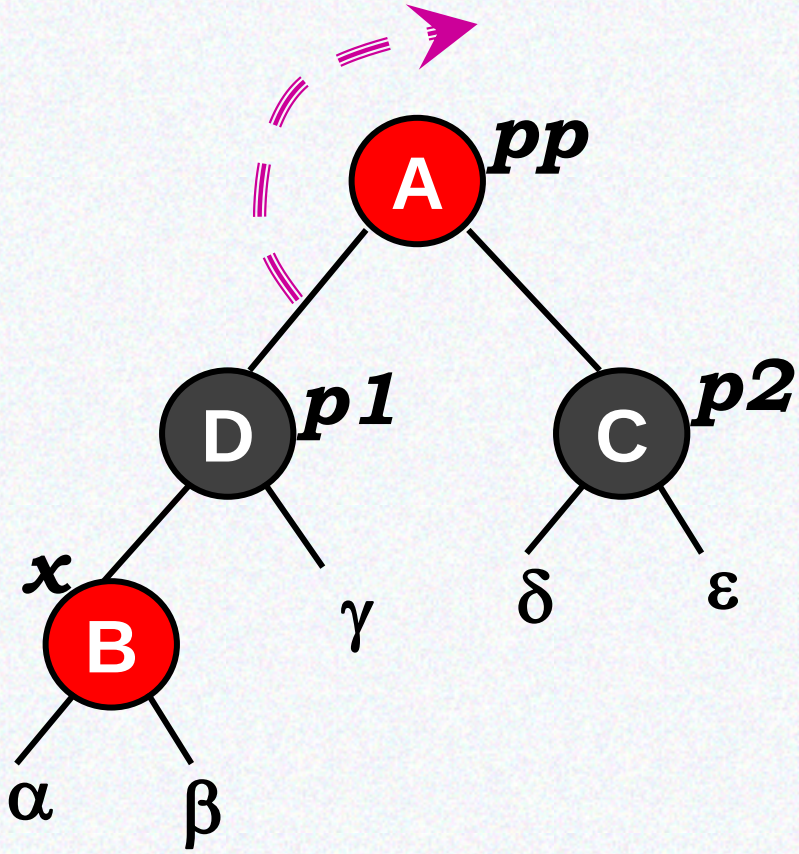
Узел $p1$ – красный

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел $p2$ – черный

Узел x – в левом поддереве узла $p1$



Коррекция:

шаг 1 – перекрасить узел $p1$ в черный цвет, узел pp – в красный цвет

шаг 2 – правый поворот вокруг узла pp

Случай 3

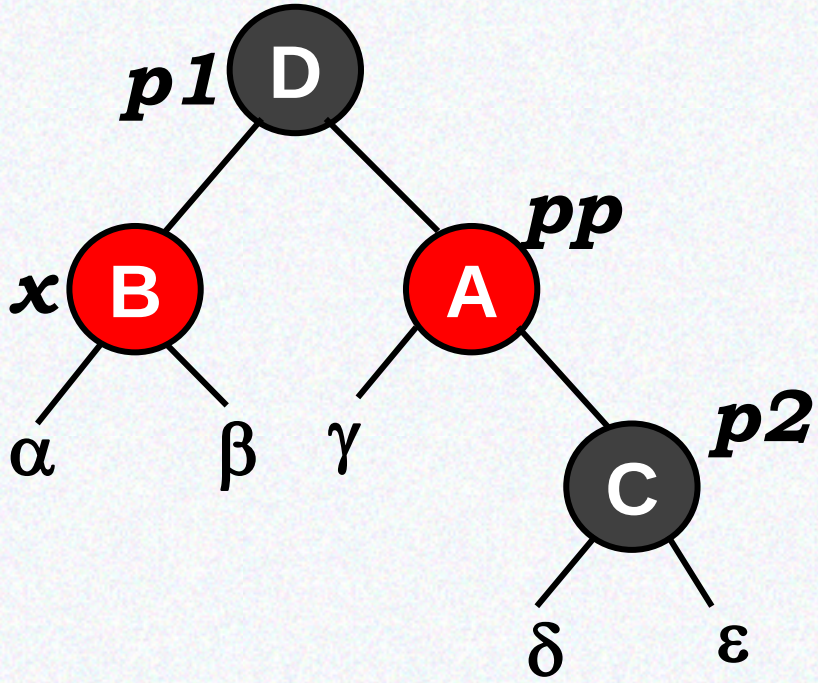
Узел $p1$ – красный

Узел pp – черный

Узлы α, β, γ, δ, ε – черные

Узел $p2$ – черный

Узел x – в левом поддереве узла $p1$



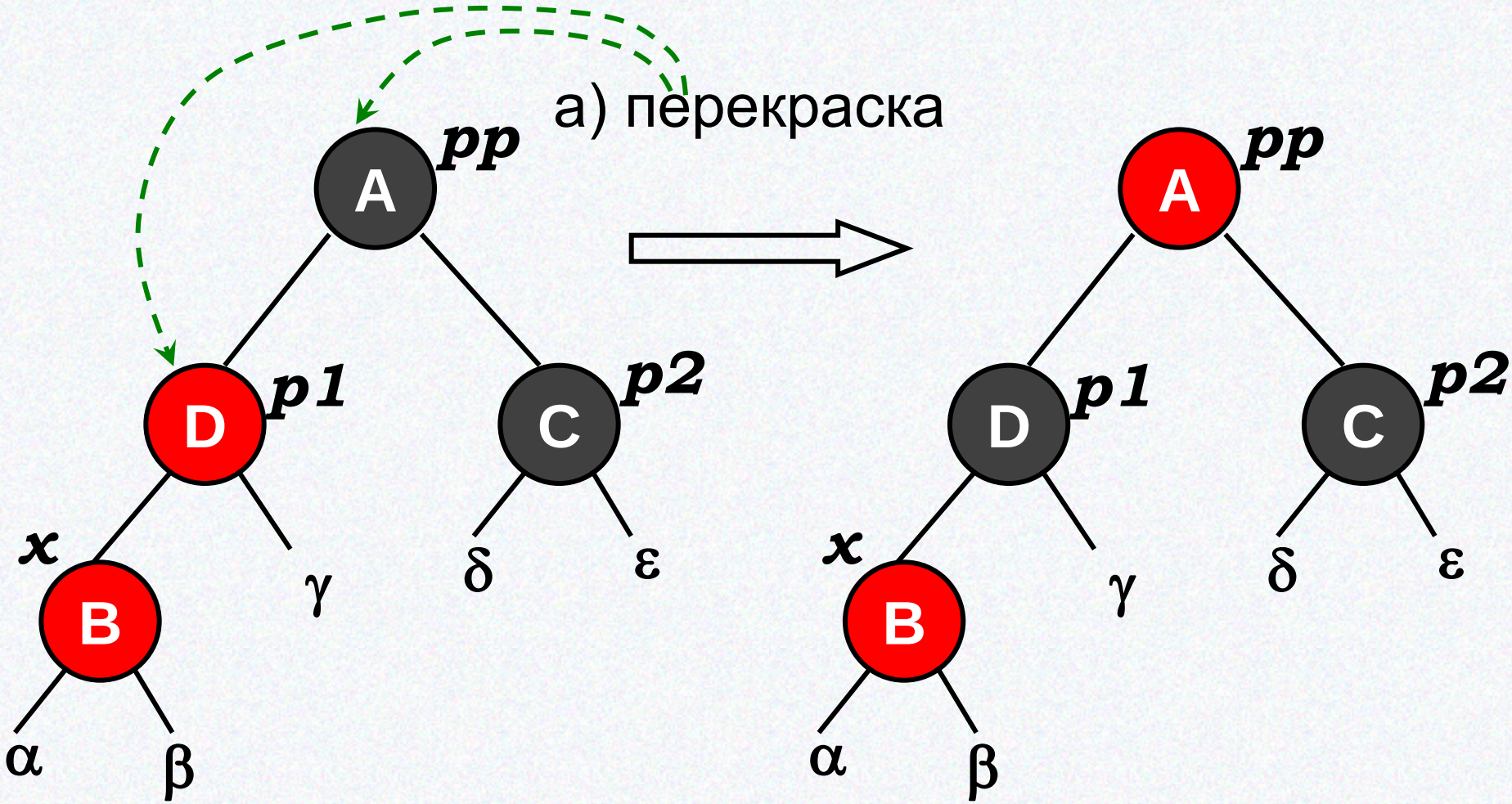
Коррекция:

шаг 1 – перекрасить узел $p1$ в черный цвет, узел pp – в красный цвет

шаг 2 – правый поворот вокруг узла pp

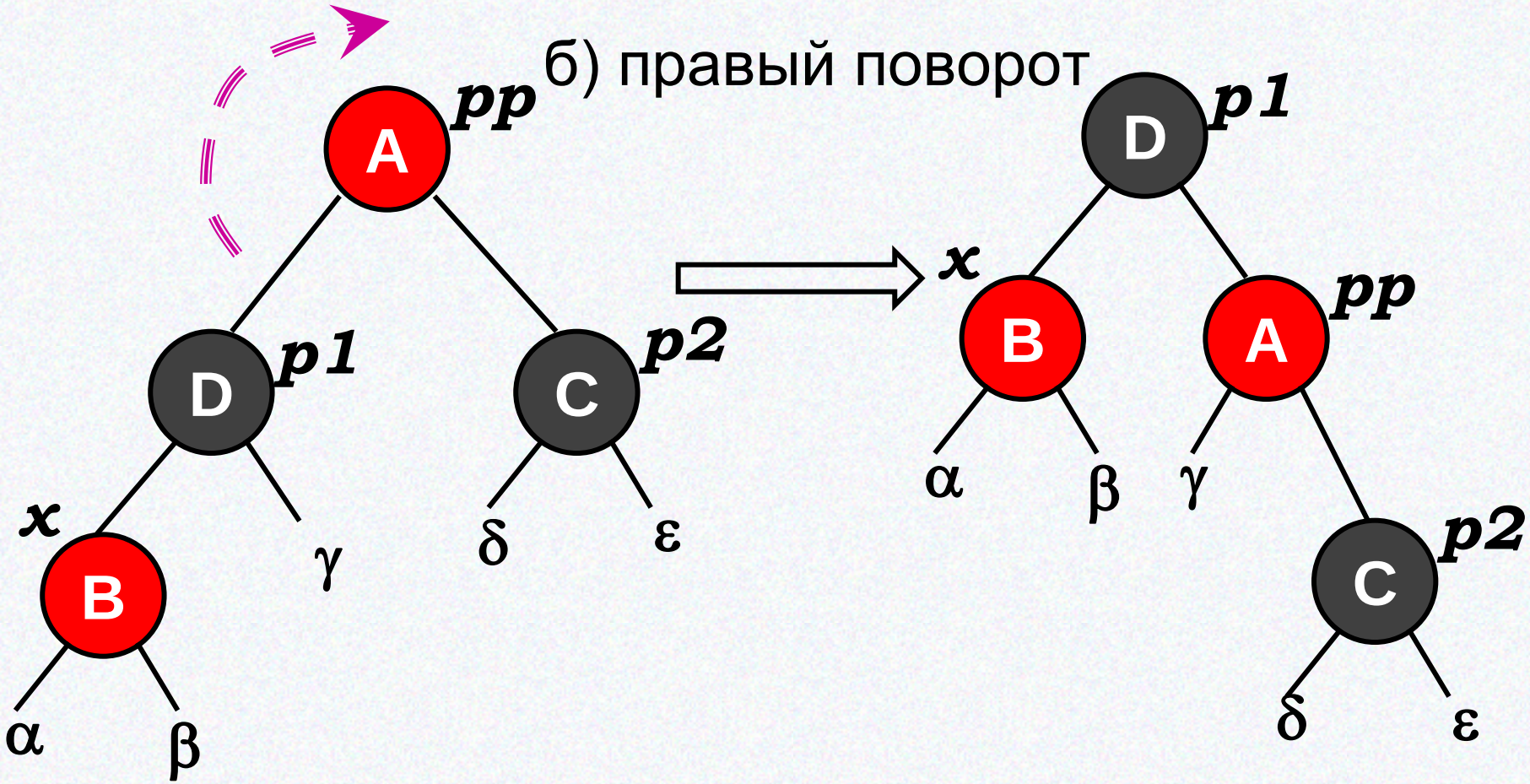
Случай 3

а) перекраска

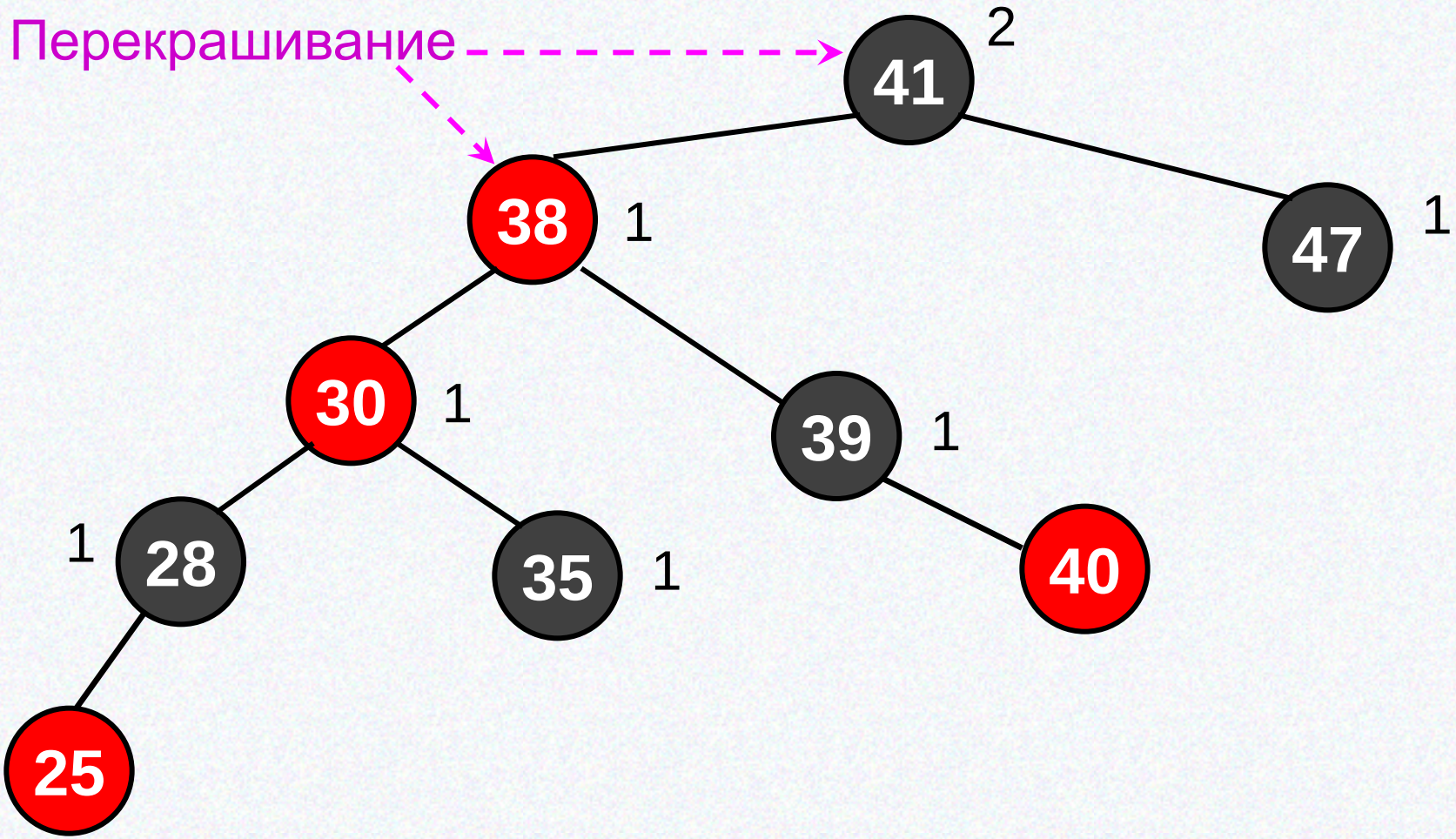


Случай 3

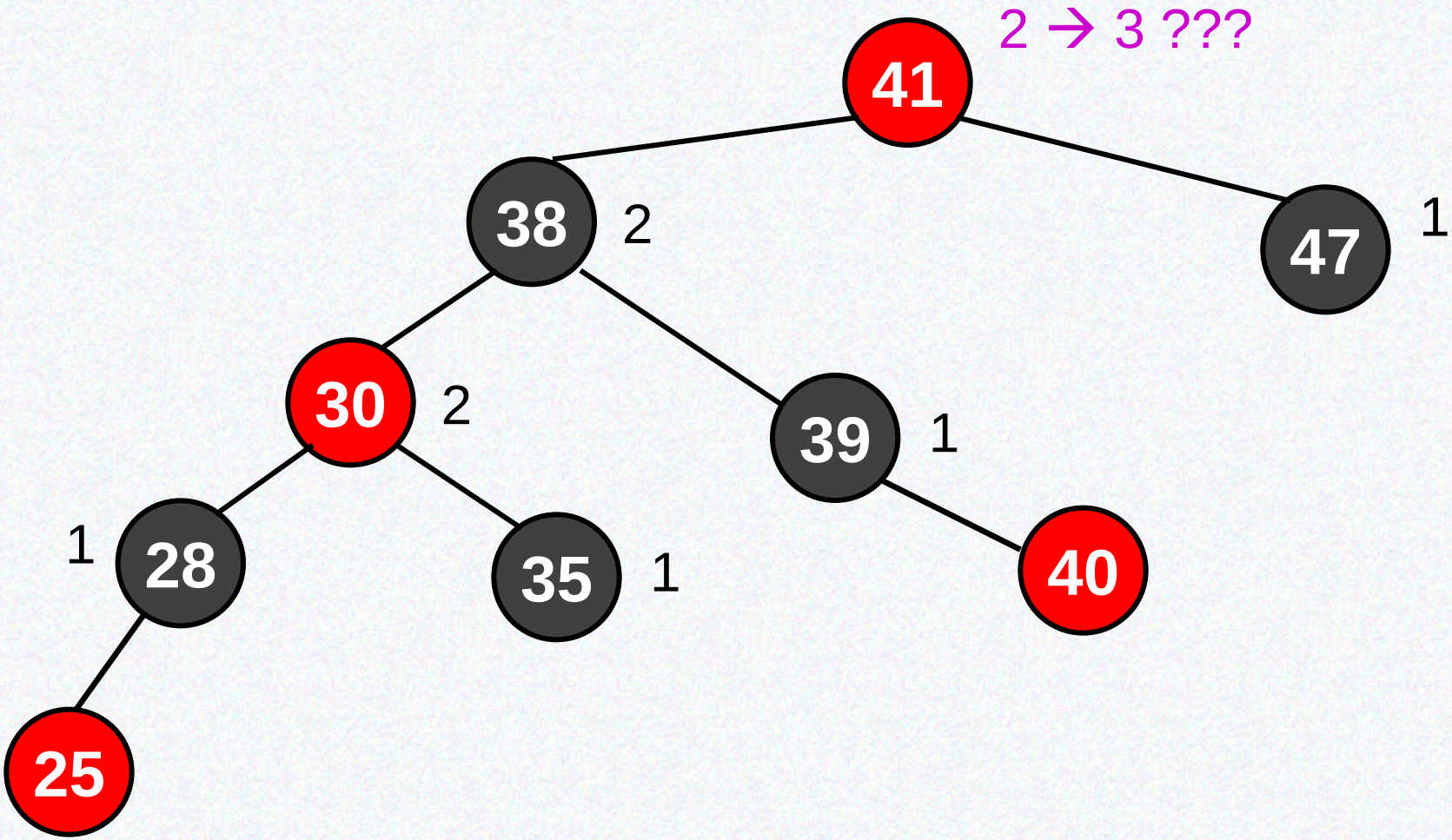
б) правый поворот



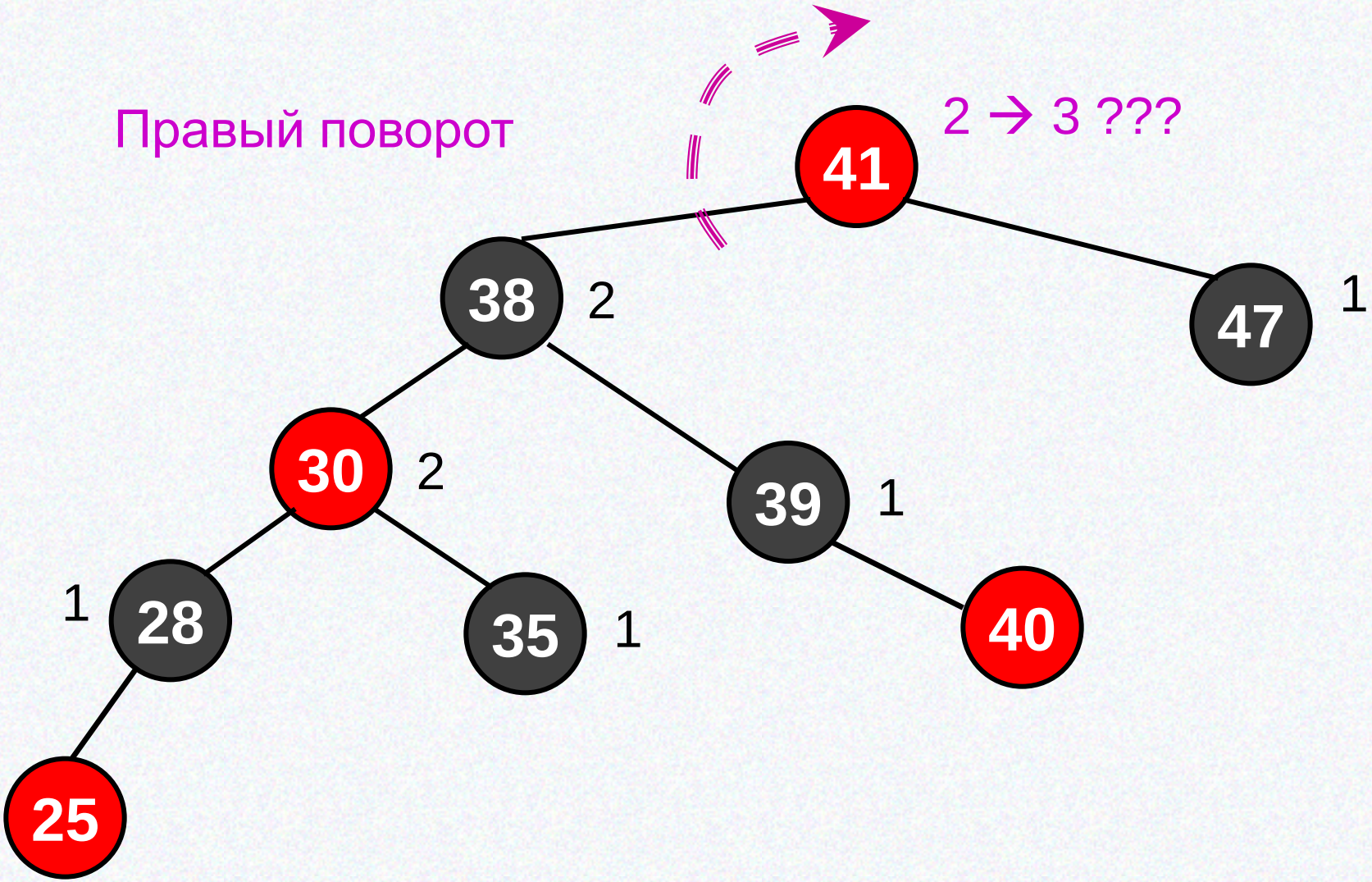
Вставка узла 40 (продолжение)



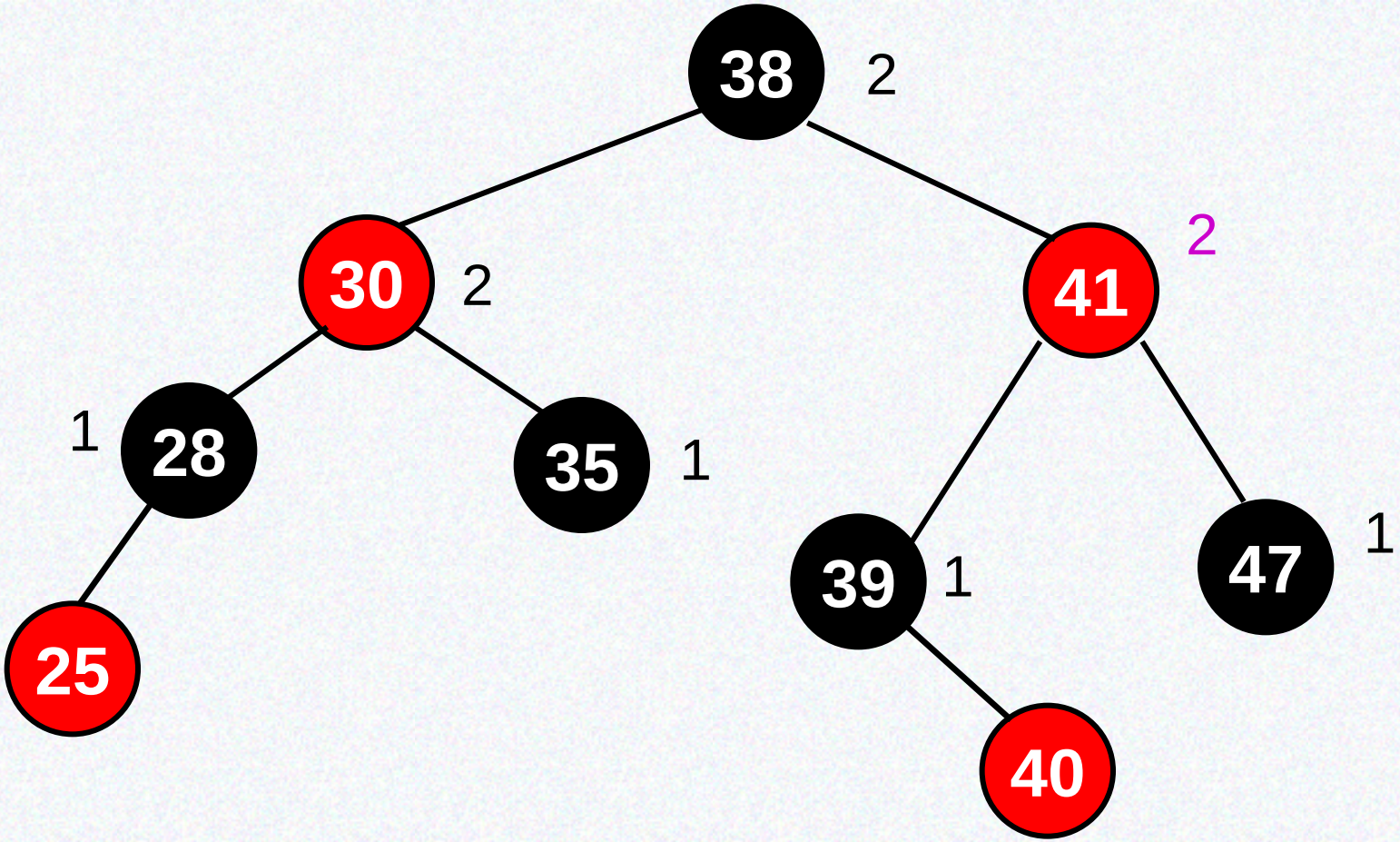
Вставка узла 40 (продолжение)



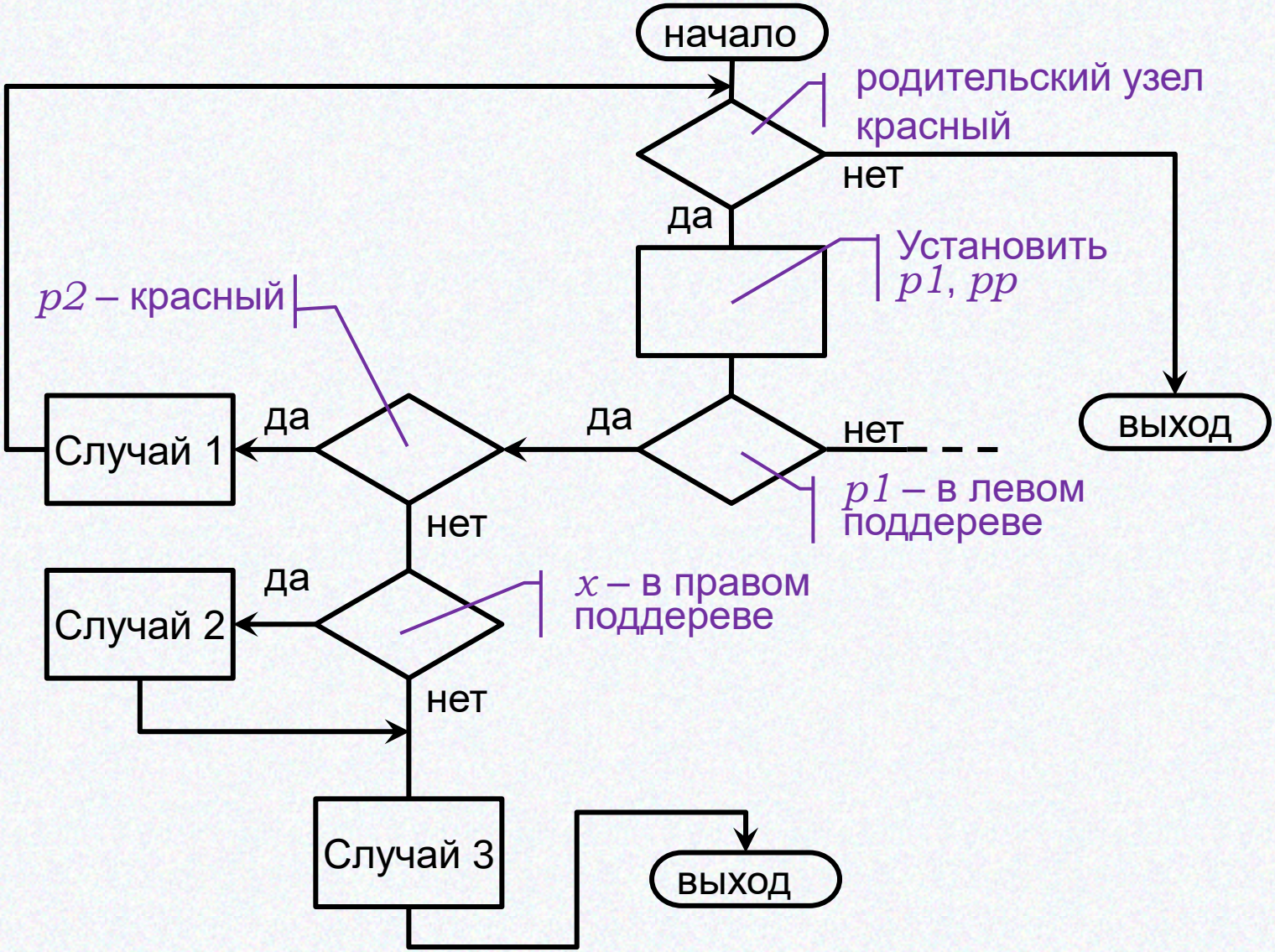
Вставка узла 40 (продолжение)



Вставка узла 40 (продолжение)



Алгоритм *RB_Insert_Fixup(x)*



Алгоритм *RB_Insert_Fixup(x)*

x – вставленный узел

while родительский узел для x – красный
(*x->parent* == *RED*) {

Алгоритм *RB_Insert_Fixup(x)*

x – вставленный узел

while родительский узел для *x* – красный
(*x->parent == RED*) {

p1 = x->parent – родительский узел для *x*

pp = p1->parent – родительский узел для *p1*

Алгоритм *RB_Insert_Fixup(x)*

x – вставленный узел

while родительский узел для *x* – красный
(*x->parent == RED*) {

p1 = x->parent – родительский узел для *x*

pp = p1->parent – родительский узел для *p1*

if p1 = pp->left { #1 – начало

Узел вставляется в левое поддерево

p2 = pp->right

Алгоритм *RB_Insert_Fixup(x)*

if p2->color == RED {

случай 1: перекрасить вершины:

Алгоритм *RB_Insert_Fixup(x)*

```
if p2->color == RED {
   случай 1: перекрасить вершины:
    p1->color = BLACK
    p2->color = BLACK
     pp->color = RED
    x = pp
    continue
}
```

Алгоритм *RB_Insert_Fixup(x)*

else if x в правом поддереве *p1*
(*x == p1->right*) {

случай 2:

Алгоритм *RB_Insert_Fixup(x)*

else if x в правом поддереве *p1*
(*x == p1->right*) {

случай 2:

x = p1

левый поворот: *left_rotate(x)*

p1 = x->parent

}

Алгоритм *RB_Insert_Fixup(x)*

Случай 3:

перекрасить вершины:

p1->color = BLACK

pp->color = RED

правый поворот:

Right_rotate(pp)

Алгоритм *RB_Insert_Fixup(x)*

Случай 3:

перекрасить вершины:

p1->color = BLACK

pp->color = RED

правый поворот:

Right_rotate(pp)

} #1 – конец

Алгоритм *RB_Insert_Fixup(x)*

else {

узел вставляется в правое поддерево:

повторить коды от #1 – начало до #1 – конец, заменив в них *left* на *right*, и наоборот

}

Алгоритм *RB_Insert_Fixup(x)*

else {

узел вставляется в правое поддерево:

повторить коды от #1 – начало до #1 – конец, заменив в них *left* на *right*, и наоборот

}

} – конец цикла

root->color = BLACK

Пример

Дана последовательность ключей:

1, 2, 3, 4, 5, …

Пример

Дана последовательность ключей:

1, 2, 3, 4, 5, …

а) Вставить в обычное двоичное дерево поиска

Пример

Дана последовательность ключей:

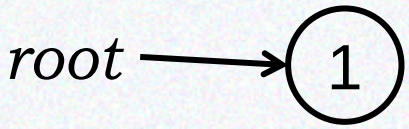
1, 2, 3, 4, 5, …

а) Вставить в обычное двоичное дерево поиска

б) Вставить в RB-дерево поиска

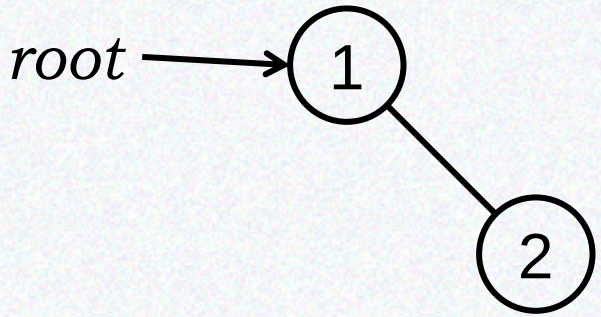
Пример

а) Обычное двоичное дерево поиска



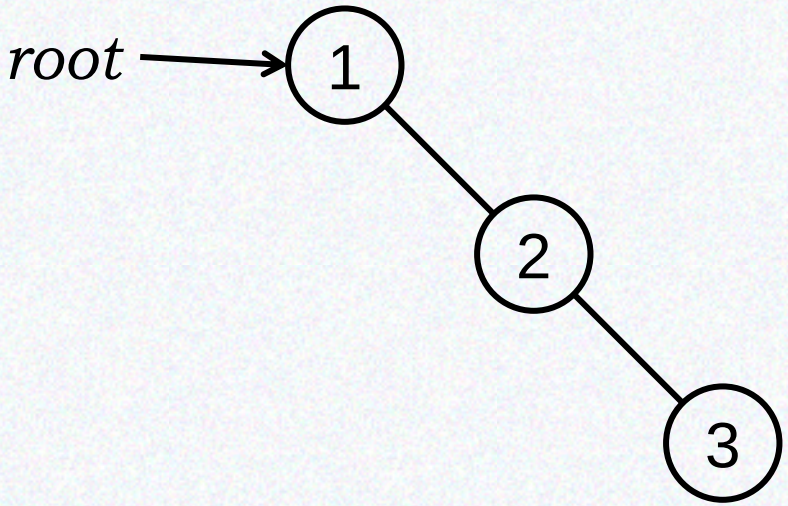
Пример

а) Обычное двоичное дерево поиска



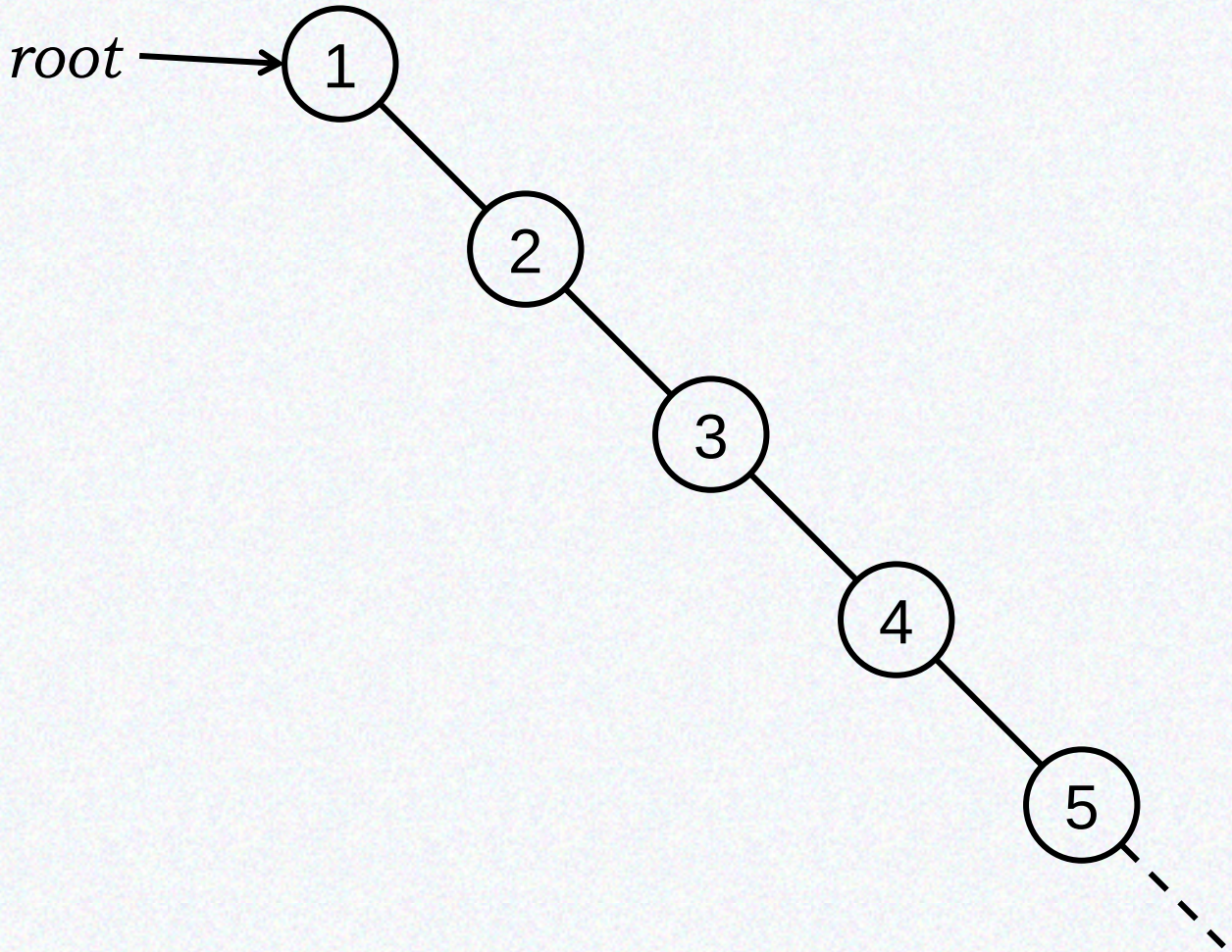
Пример

а) Обычное двоичное дерево поиска



Пример

а) Обычное двоичное дерево поиска



Пример

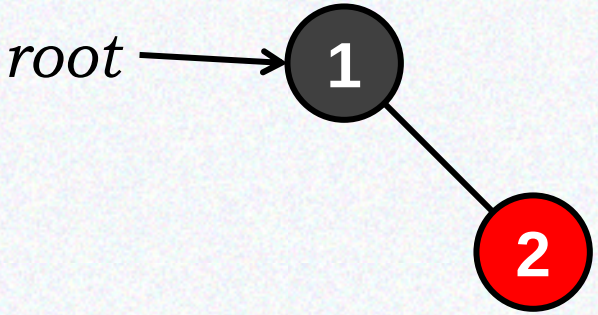
а) RB- дерево поиска

Пример

а) RB- дерево поиска

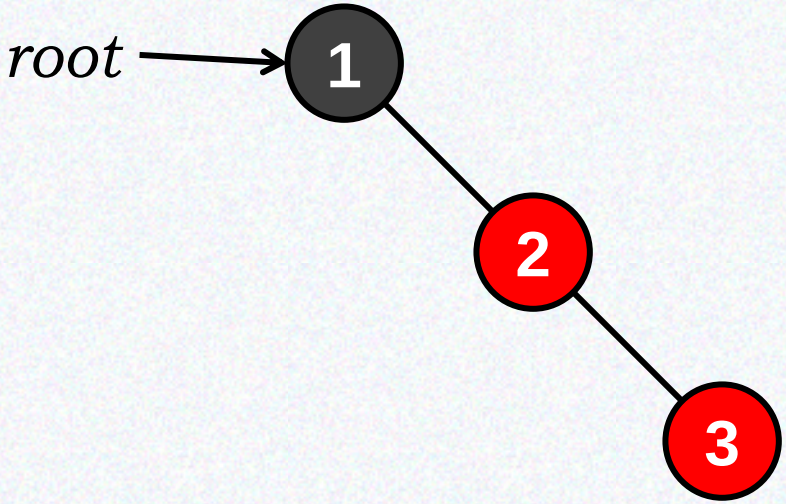
Пример

а) RB- дерево поиска



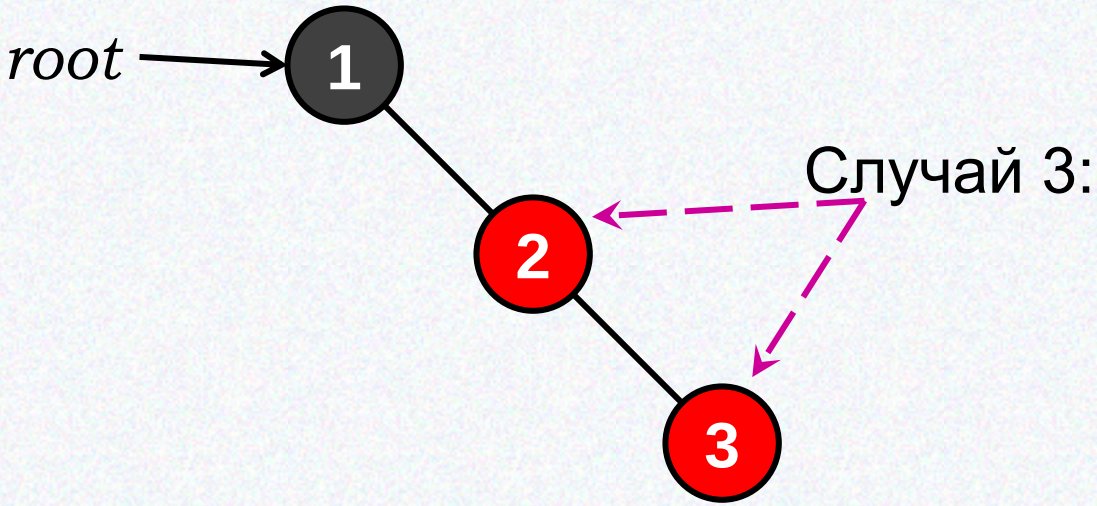
Пример

а) RB- дерево поиска



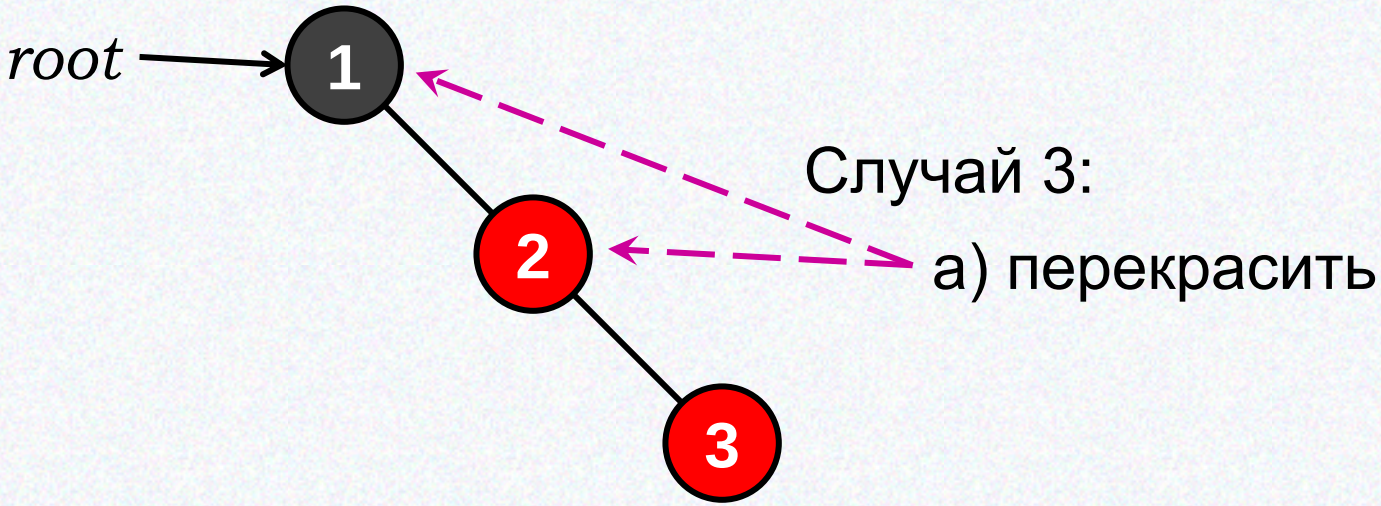
Пример

а) RB- дерево поиска



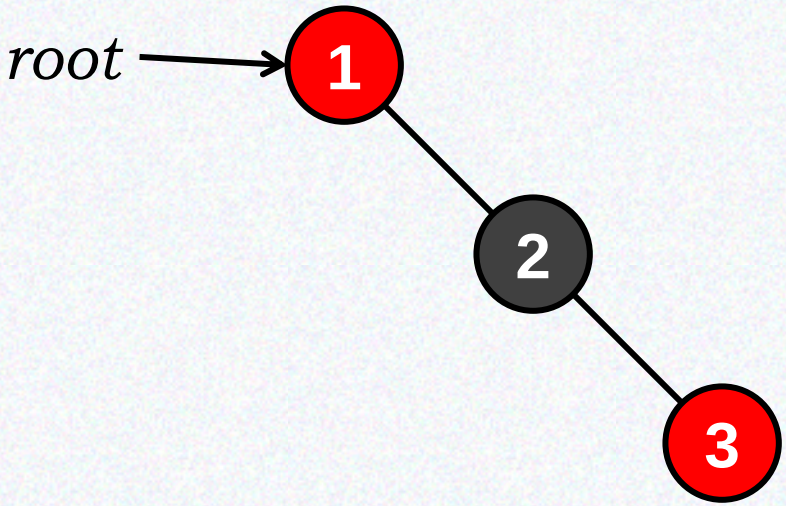
Пример

а) RB- дерево поиска



Пример

а) RB- дерево поиска

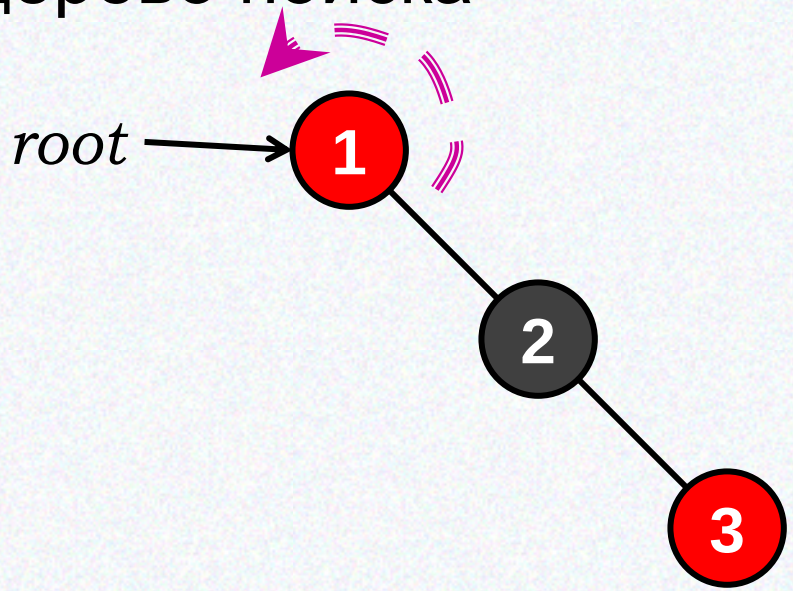


Случай 3:

а) перекрасить

Пример

а) RB- дерево поиска

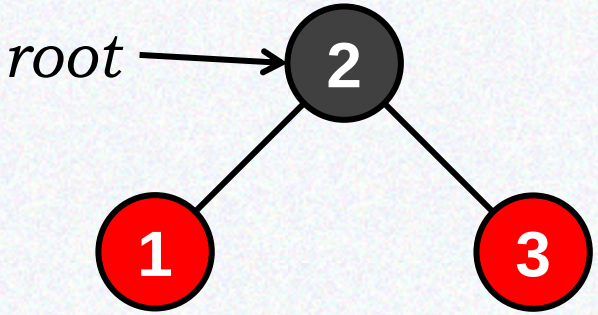


Случай 3:

а) перекрасить
б) повернуть

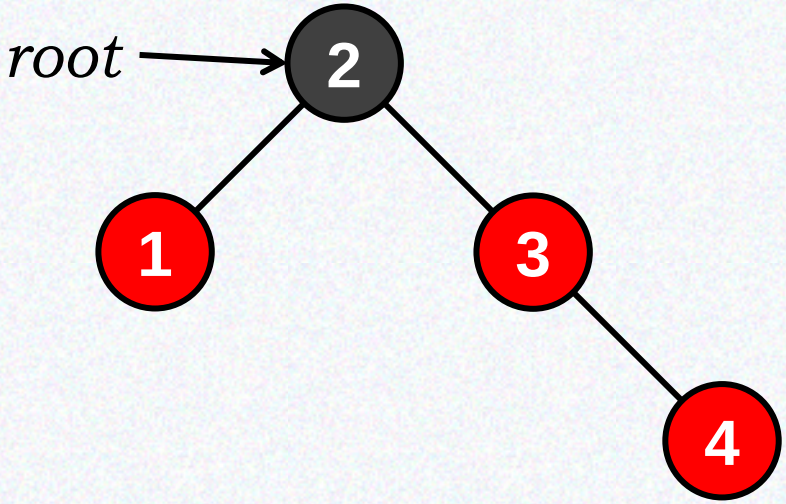
Пример

а) RB- дерево поиска



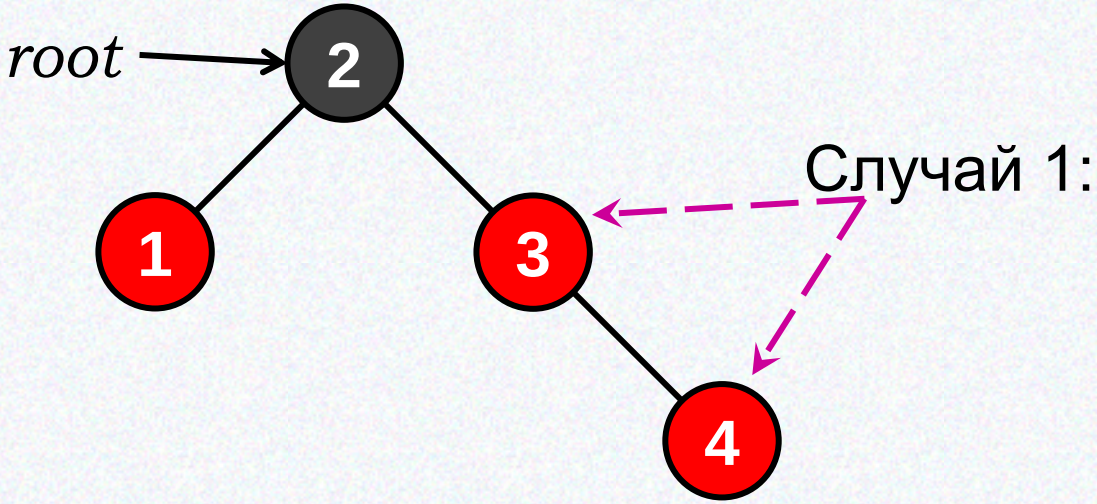
Пример

а) RB- дерево поиска



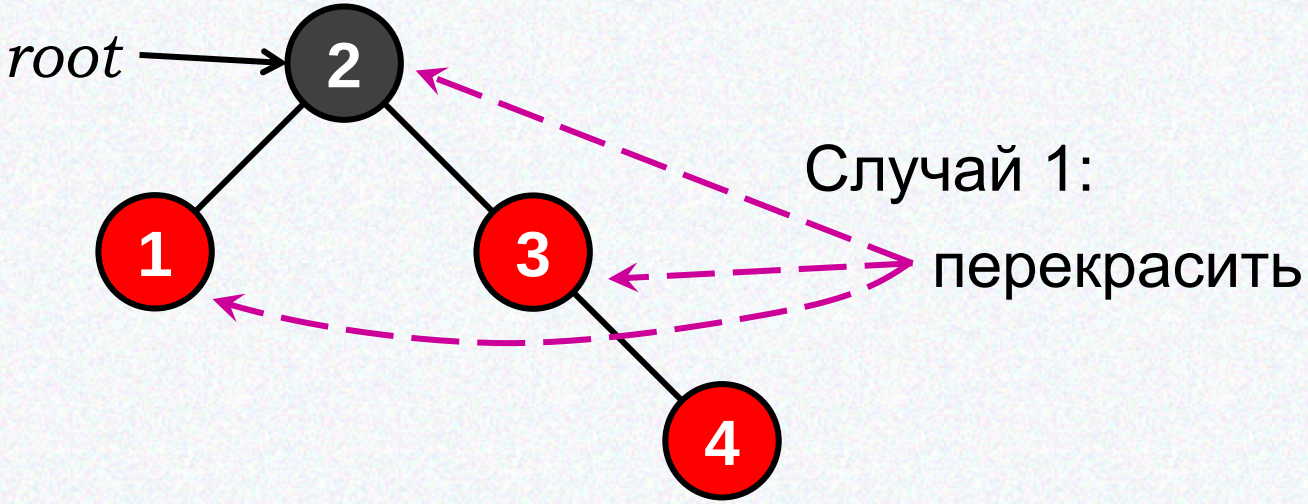
Пример

а) RB- дерево поиска



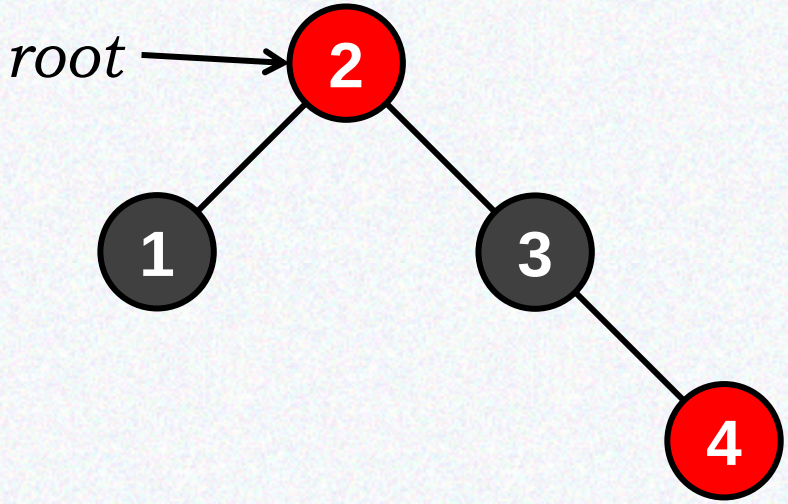
Пример

а) RB- дерево поиска



Пример

а) RB- дерево поиска

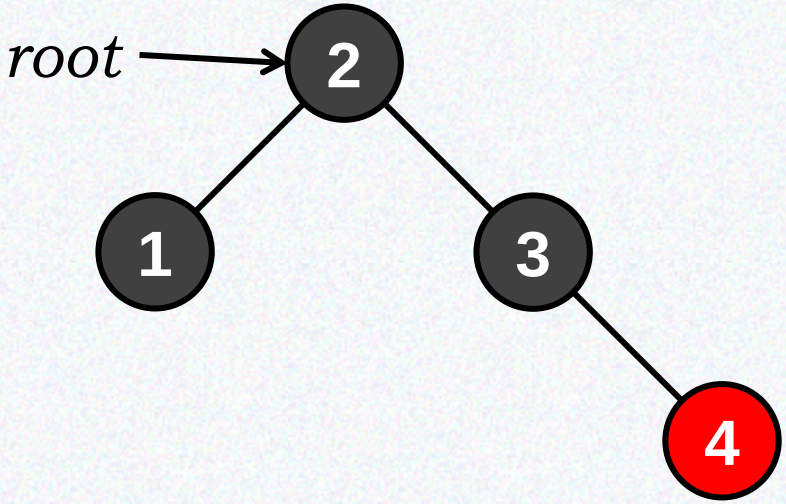


Случай 1:

перекрасить

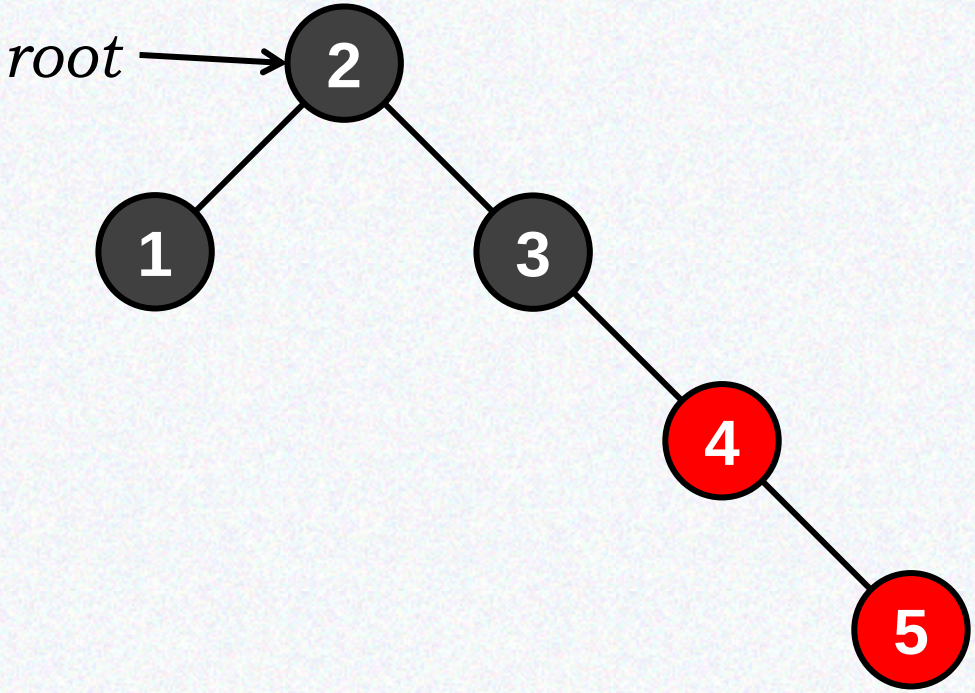
Пример

а) RB- дерево поиска



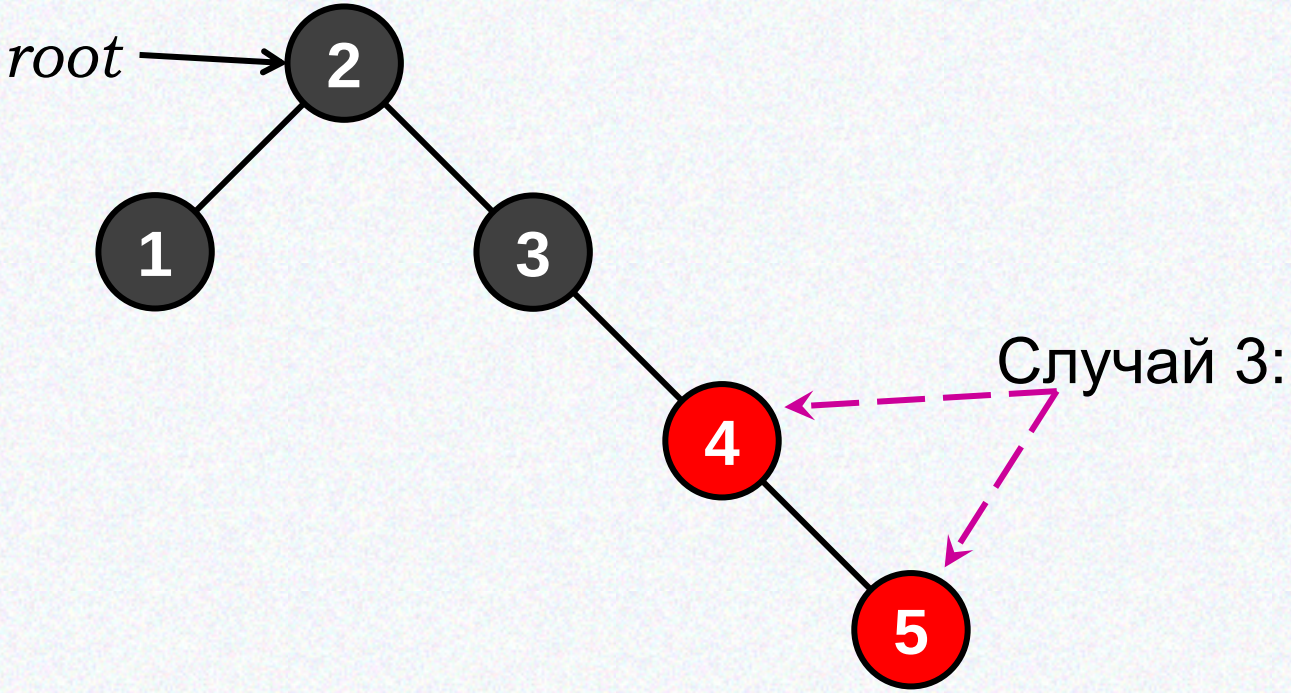
Пример

a) RB- дерево поиска



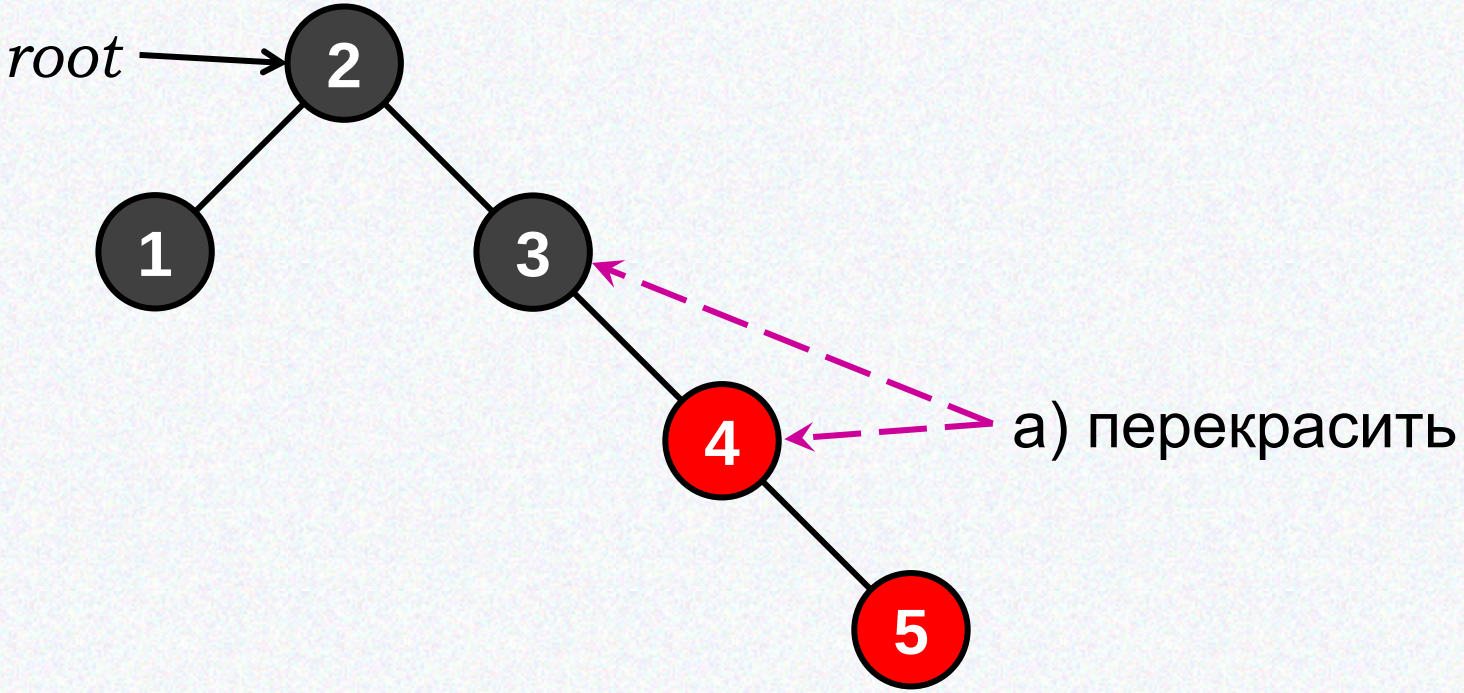
Пример

а) RB- дерево поиска



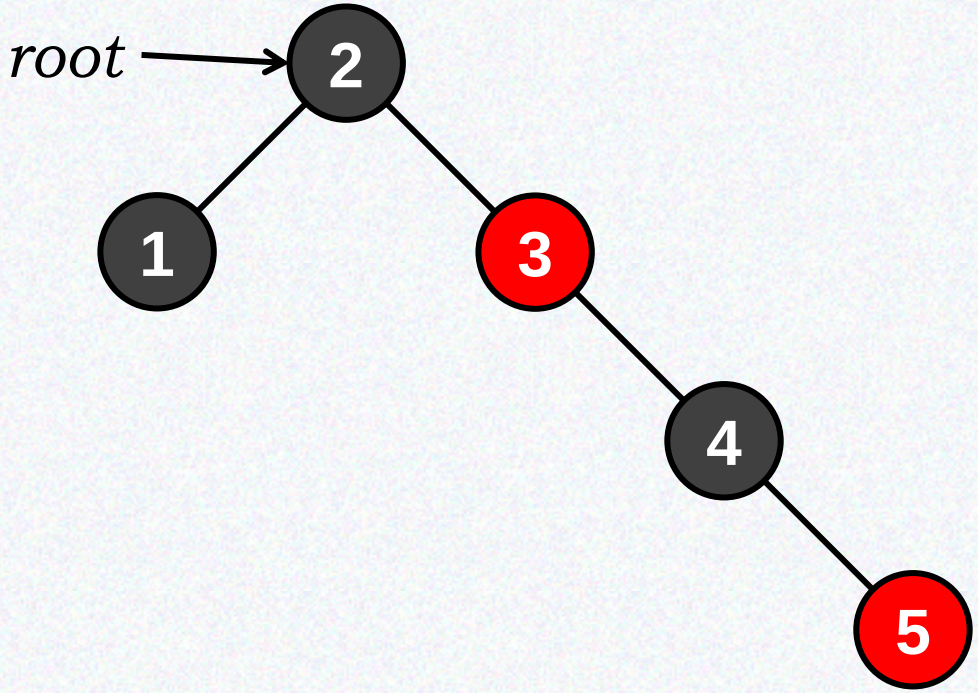
Пример

а) RB- дерево поиска



Пример

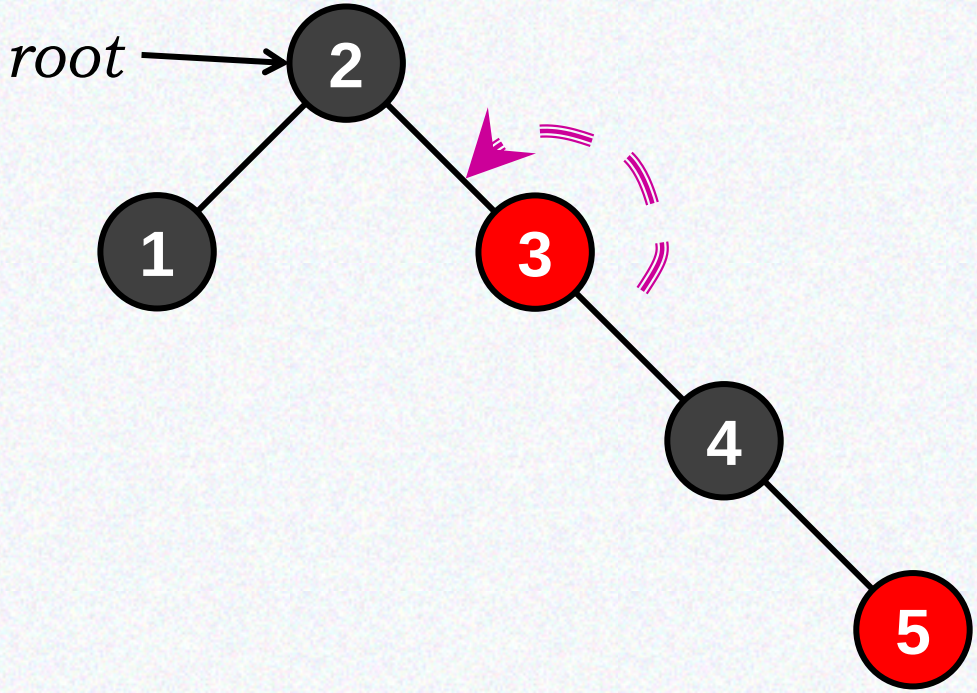
а) RB- дерево поиска



а) перекрасить

Пример

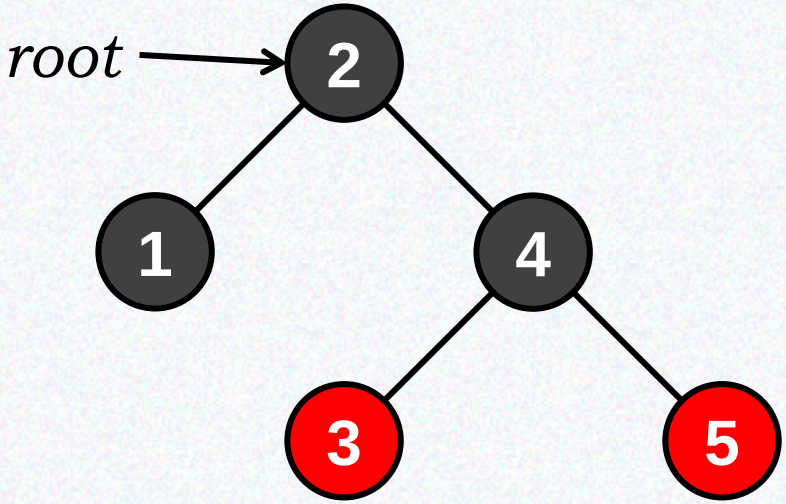
а) RB- дерево поиска



а) перекрасить
б) повернуть

Пример

а) RB- дерево поиска



Анализ эффективности

Вставка в RB- дерево:

Анализ эффективности

Вставка в RB- дерево:

- вставка в двоичное дерево – *O(h)*

Анализ эффективности

Вставка в RB- дерево:

- вставка в двоичное дерево – *O(h)*
- коррекция RB- дерева

Анализ эффективности

Вставка в RB- дерево:

- вставка в двоичное дерево – *O(h)*
- коррекция RB- дерева

Коррекция:

Анализ эффективности

Вставка в RB- дерево:

- вставка в двоичное дерево – *O(h)*
- коррекция RB- дерева

Коррекция:

для случая 1 – итерации: *O(h)*

Анализ эффективности

Вставка в RB- дерево:

- вставка в двоичное дерево – *O(h)*
- коррекция RB- дерева

Коррекция:

для случая 1 – итерации: *O(h)*

для случаев 2 и 3 – максимум два поворота

Анализ эффективности

Вставка в RB- дерево:

- вставка в двоичное дерево – $O(h)$
- коррекция RB- дерева

Коррекция:

для случая 1 – итерации: $O(h)$

для случаев 2 и 3 – максимум два поворота

$h \leq 2\lg(n + 1)$, вставка – $O(\lg(n))$

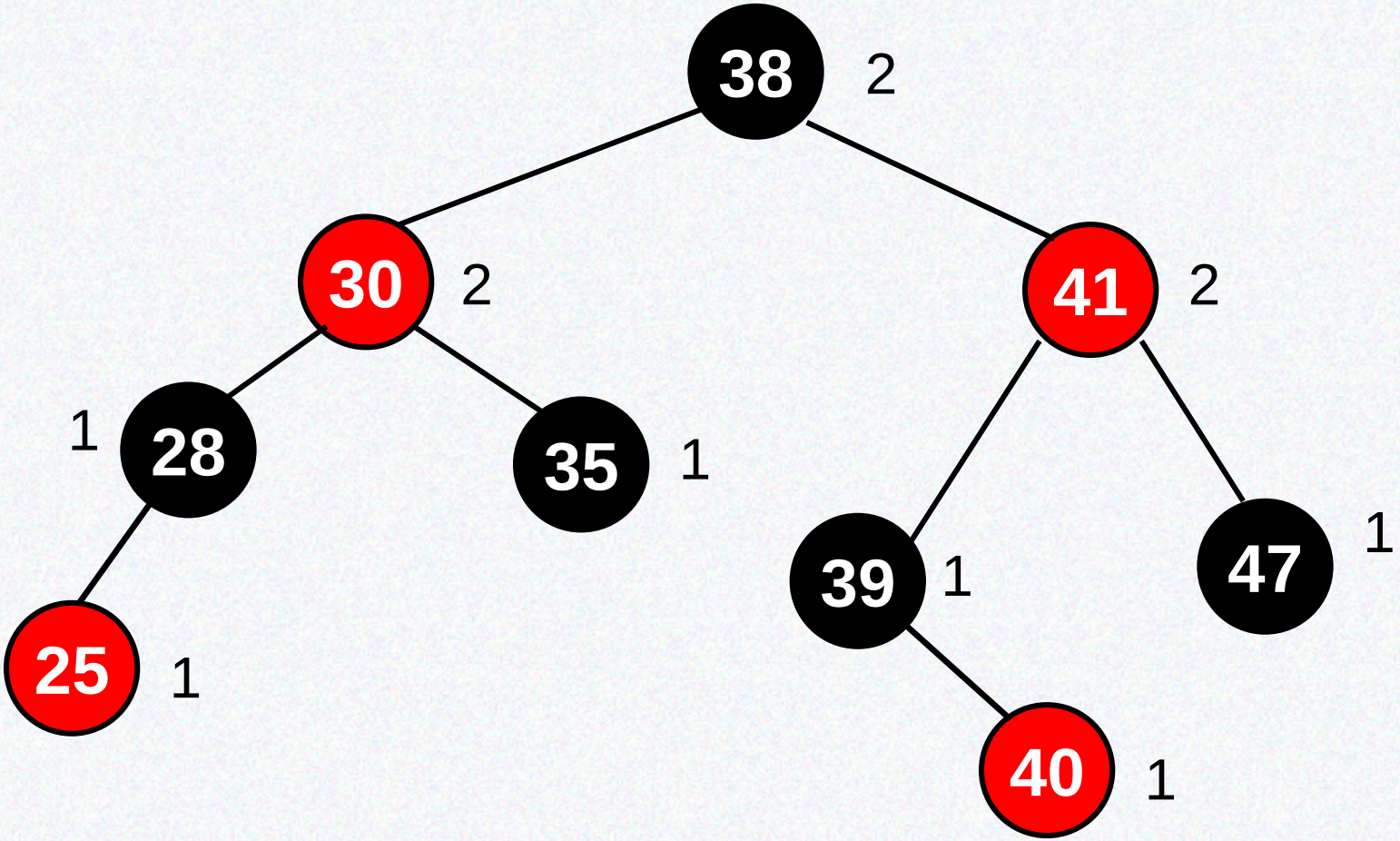
Удаление

- Удалить узел как в бинарном дереве поиска; учесть представление листьев

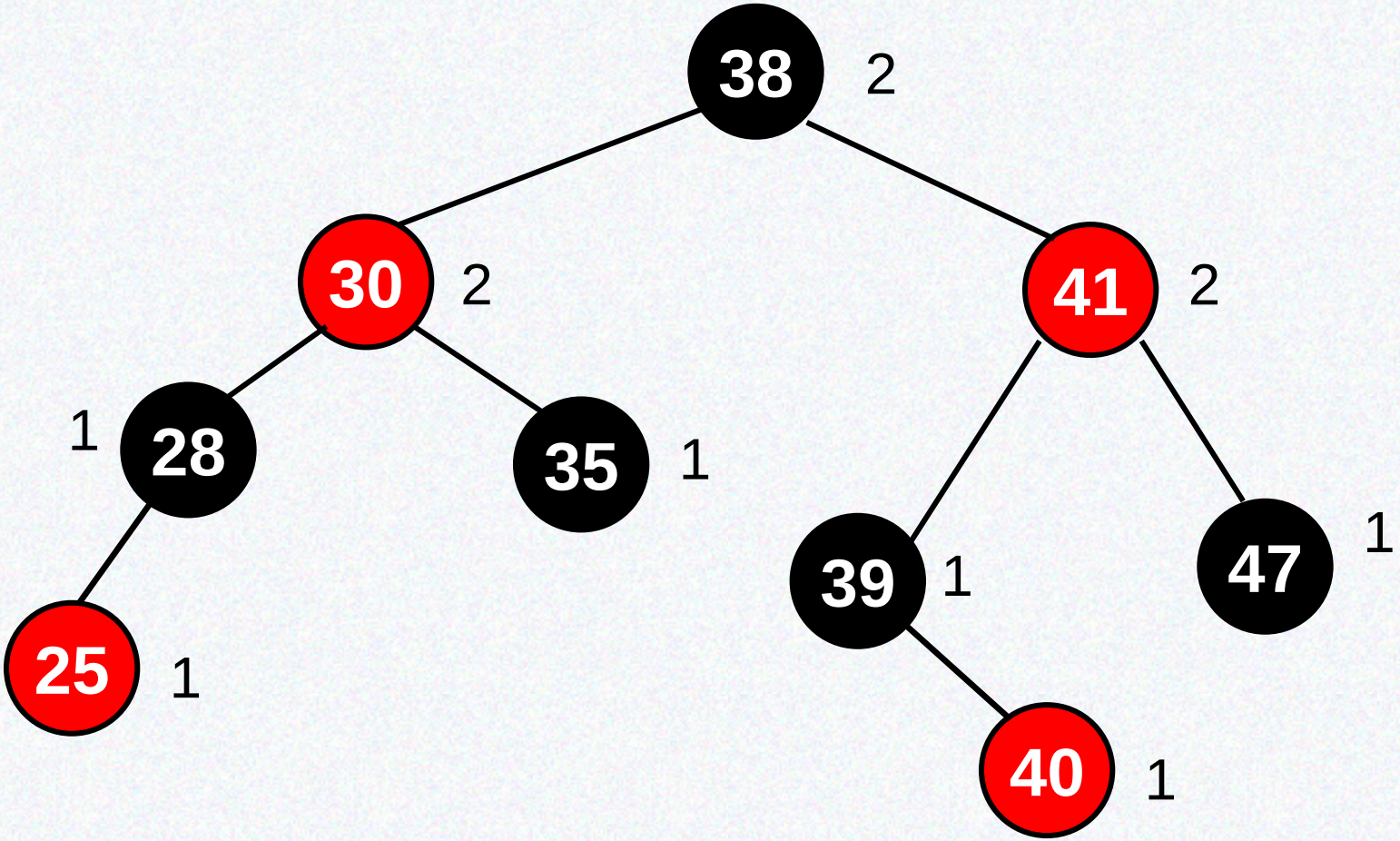
Удаление

- Удалить узел как в бинарном дереве поиска; учесть представление листьев
- Восстановить красно-черные свойства дерева

Удаление

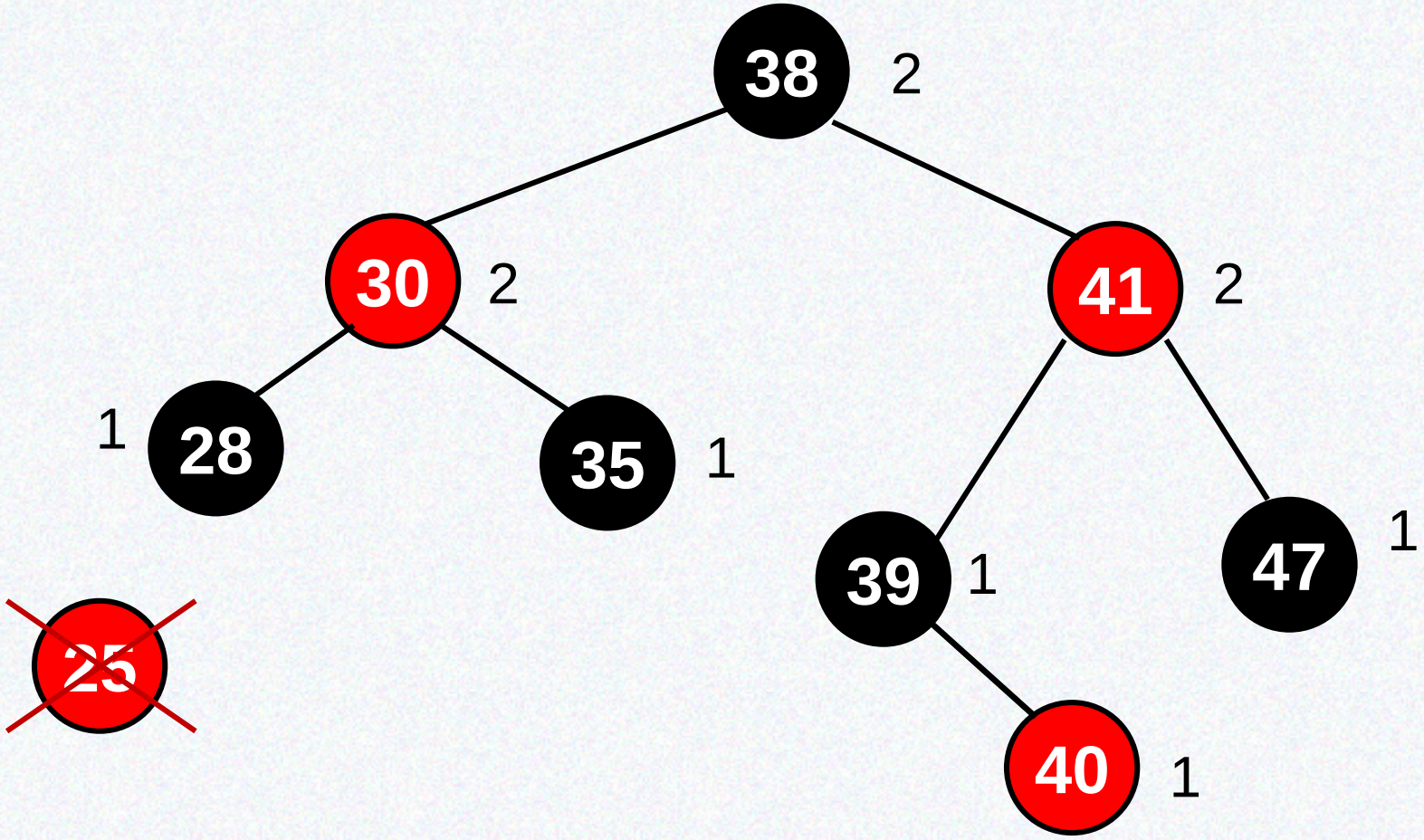


Удаление



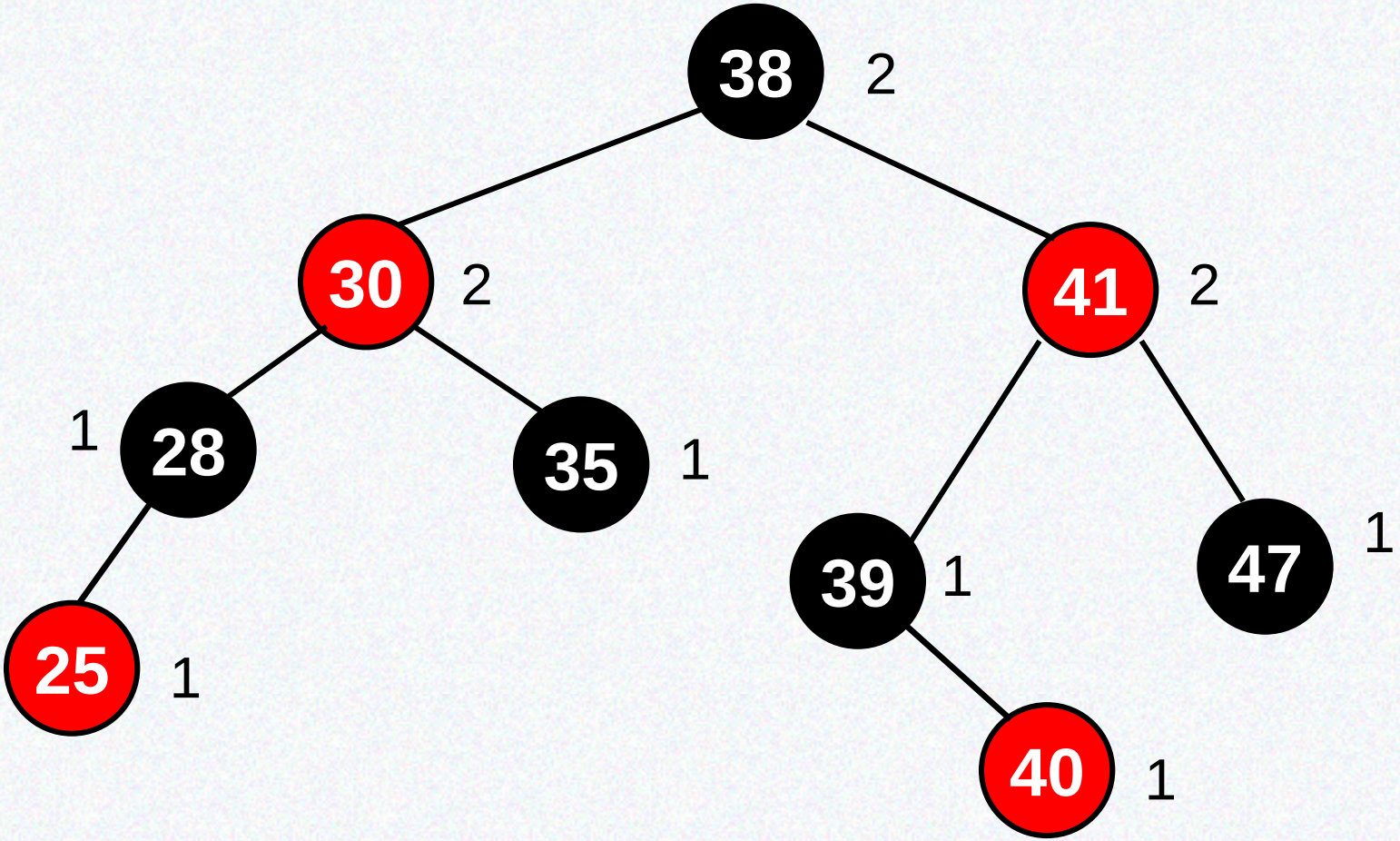
Удалить 25

Удаление



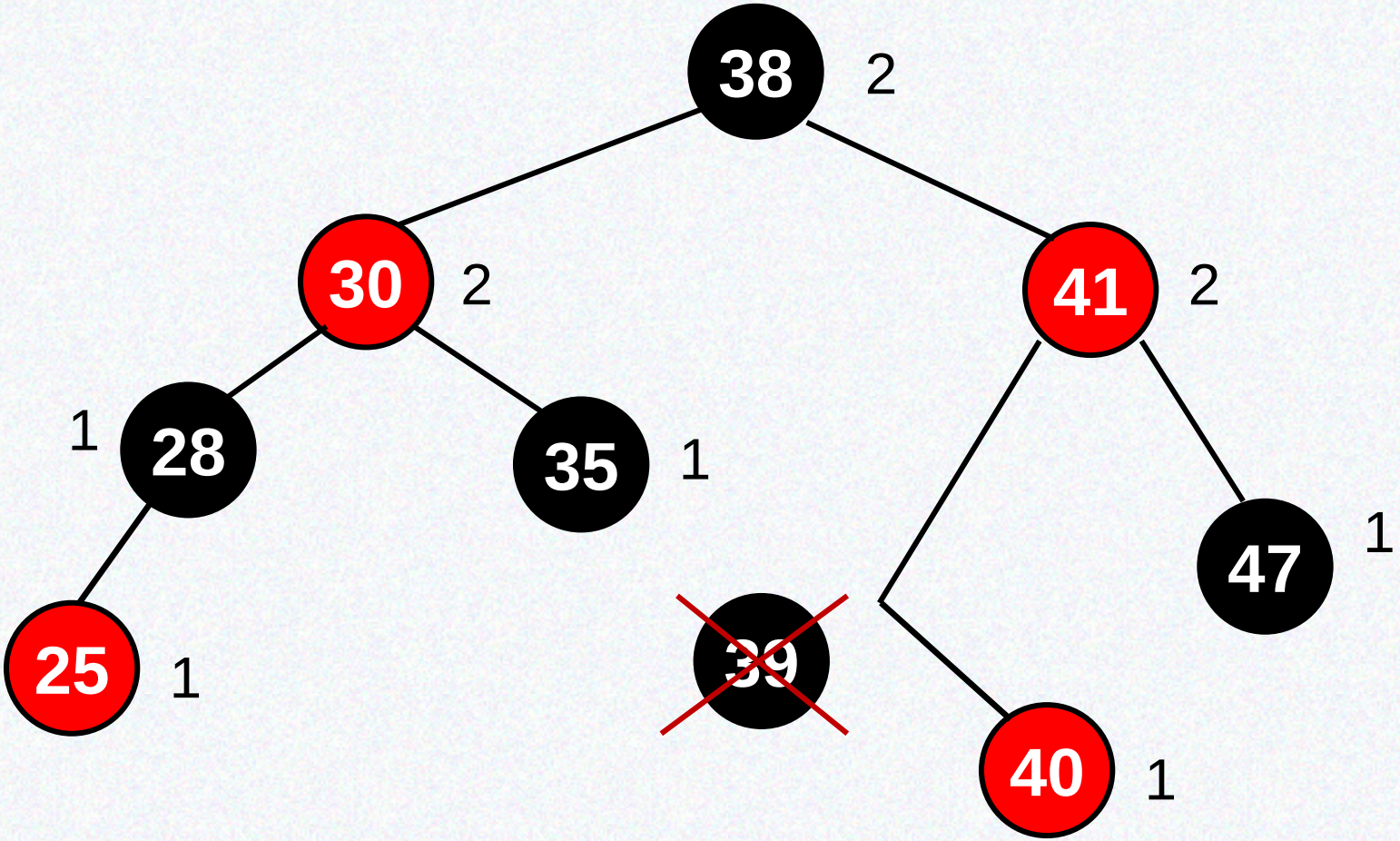
Удалить 25

Удаление



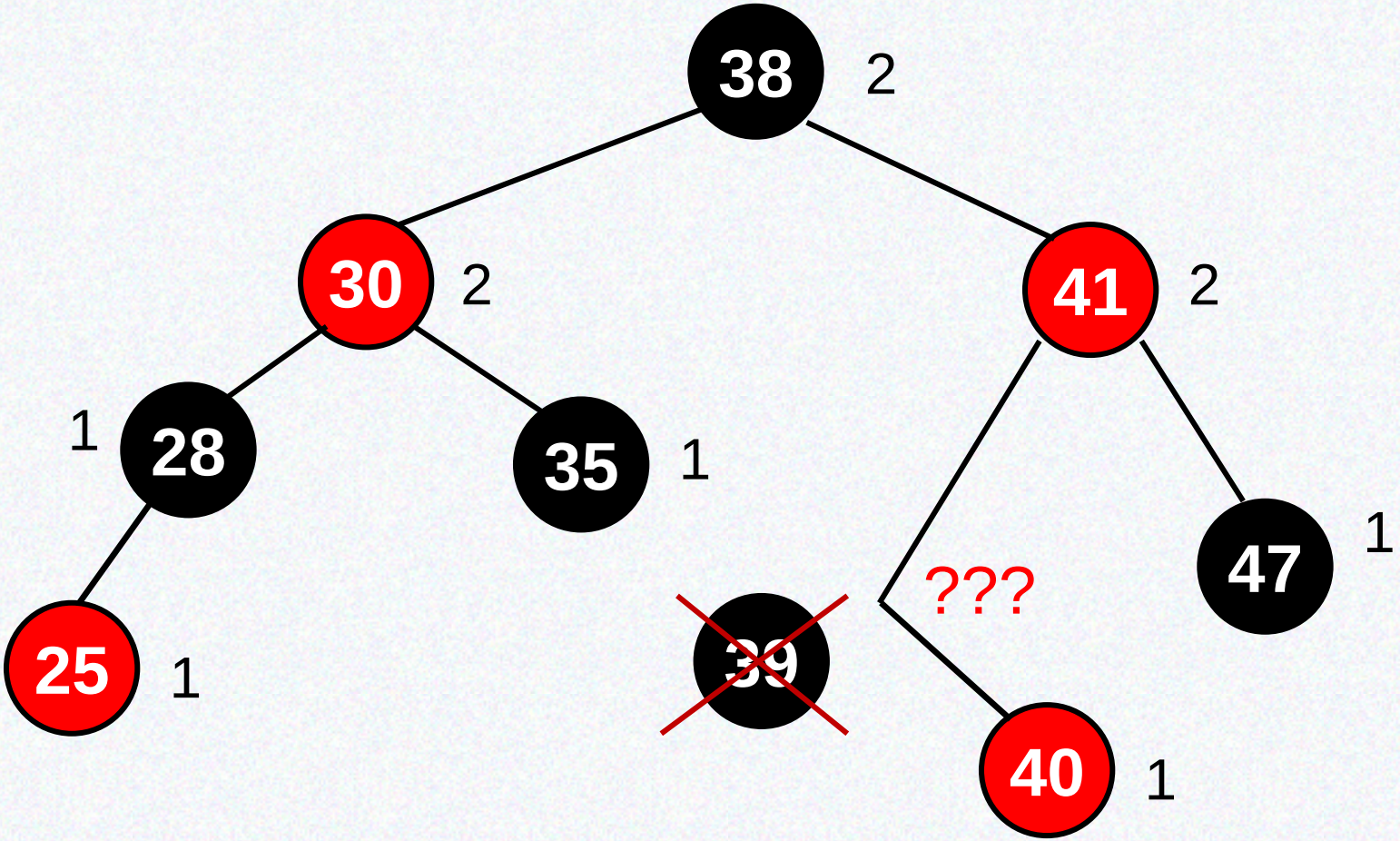
Удалить 39

Удаление



Удалить 39

Удаление



Удалить 39

Алгоритм *RB_Delete(x)*

```
if x->left == Elist или x->right == Elist
   y = x
else
   y = successor(x)
if y->left ≠ Elist
   p = y->left
else
   p = y->right
```

Алгоритм *RB_Delete(x)*

```
p->parent = y->parent
if y->parent == EList
   root = p
else
   if y->parent->left == y
      y->parent->left = p
   else
      y->parent->right = p
```

→ отличается!

Алгоритм *RB_Delete(x)*

```
if y ≠ x {
   x->key = y->key
   копирование сопутствующих данных
}
```

Алгоритм *RB_Delete(x)*

if y ≠ x {

x->key = y->key

копирование сопутствующих данных

}

if y->color == BLACK

RB_Delete_Fixup(p)

Успех

Нарушение свойств дерева при удалении

Если удаляется красный узел:

Нарушение свойств дерева при удалении

Если удаляется красный узел:

- никакая черная высота в дереве не изменяется

Нарушение свойств дерева при удалении

Если удаляется красный узел:

- никакая черная высота в дереве не изменяется
- никакие красные узлы не становятся соседними

Нарушение свойств дерева при удалении

Если удаляется красный узел:

- никакая черная высота в дереве не изменяется
- никакие красные узлы не становятся соседними
- корень остается черным

Нарушение свойств дерева при удалении

Если удаляется черный узел:

Нарушение свойств дерева при удалении

Если удаляется черный узел:

1. Каждый узел является красным или черным

Нарушение свойств дерева при удалении

Если удаляется черный узел:

1. Каждый узел является красным или черным

   не нарушается

Нарушение свойств дерева при удалении

Если удаляется черный узел:

1. Каждый узел является красным или черным
   не нарушается
2. Корень дерева является черным

Нарушение свойств дерева при удалении

Если удаляется черный узел:

1. Каждый узел является красным или черным
   не нарушается
2. Корень дерева является черным
   может быть нарушено

Нарушение свойств дерева при удалении

Если удаляется черный узел:

1. Каждый узел является красным или черным
   не нарушается
2. Корень дерева является черным
   может быть нарушено
3. Каждый лист дерева является черным

Нарушение свойств дерева при удалении

Если удаляется черный узел:

1. Каждый узел является красным или черным
   не нарушается
2. Корень дерева является черным
   может быть нарушено
3. Каждый лист дерева является черным
   не нарушается

Нарушение свойств дерева при удалении

4. Если узел – красный, то оба его дочерних узла – черные

Нарушение свойств дерева при удалении

4. Если узел – красный, то оба его дочерних узла – черные

может быть нарушено

Нарушение свойств дерева при удалении

4. Если узел – красный, то оба его дочерних узла – черные

   может быть нарушено

5. Для каждого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

Нарушение свойств дерева при удалении

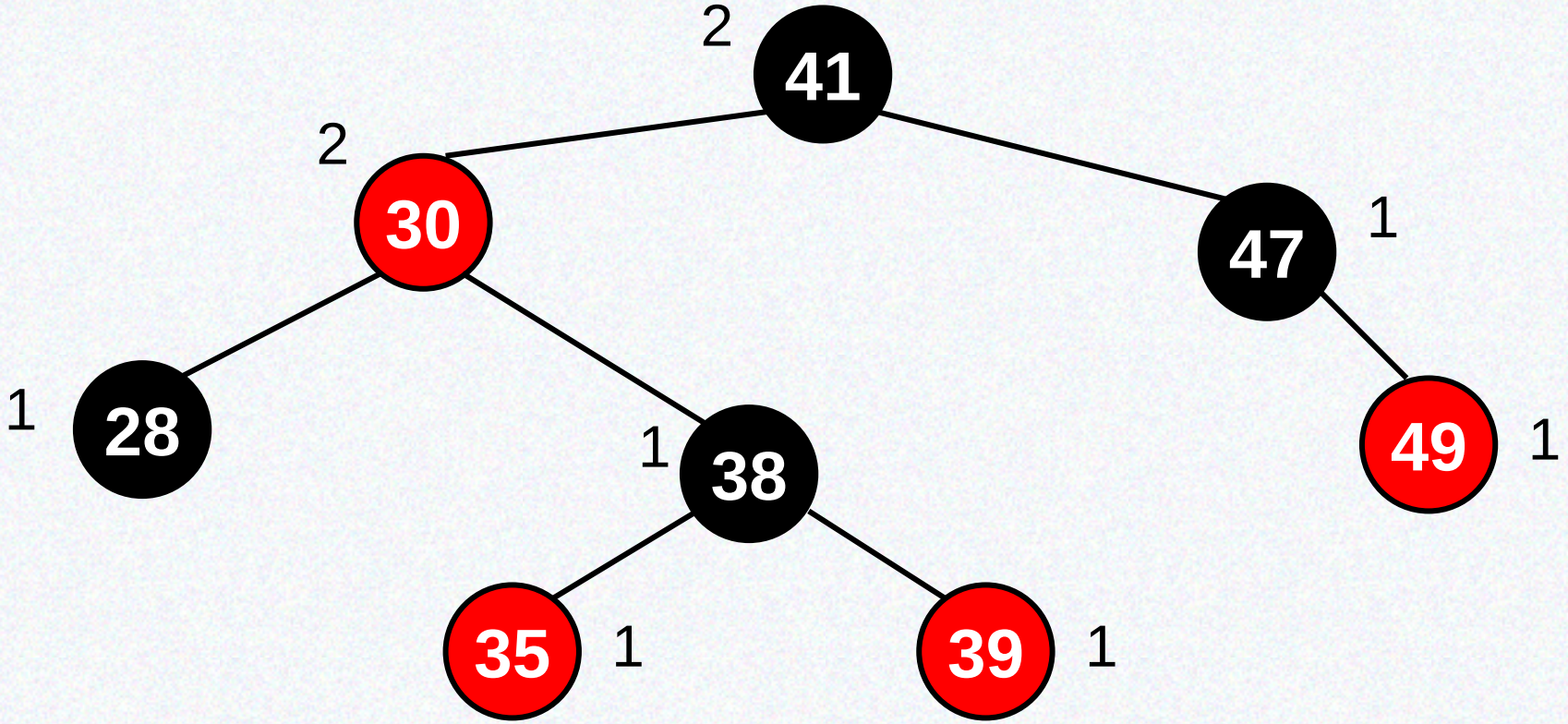
4. Если узел – красный, то оба его дочерних узла – черные

   может быть нарушено

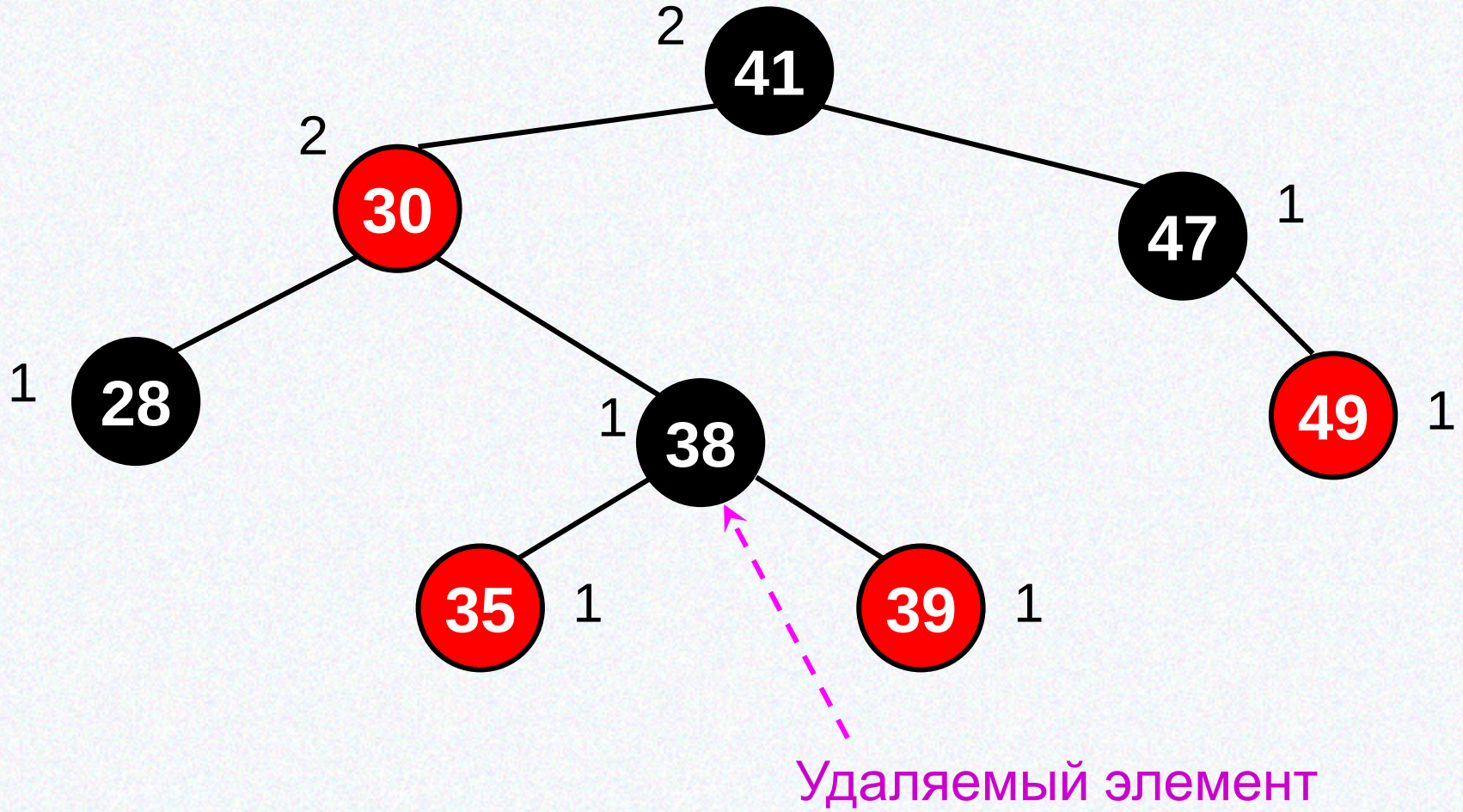
5. Для каждого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

   нарушается!!!

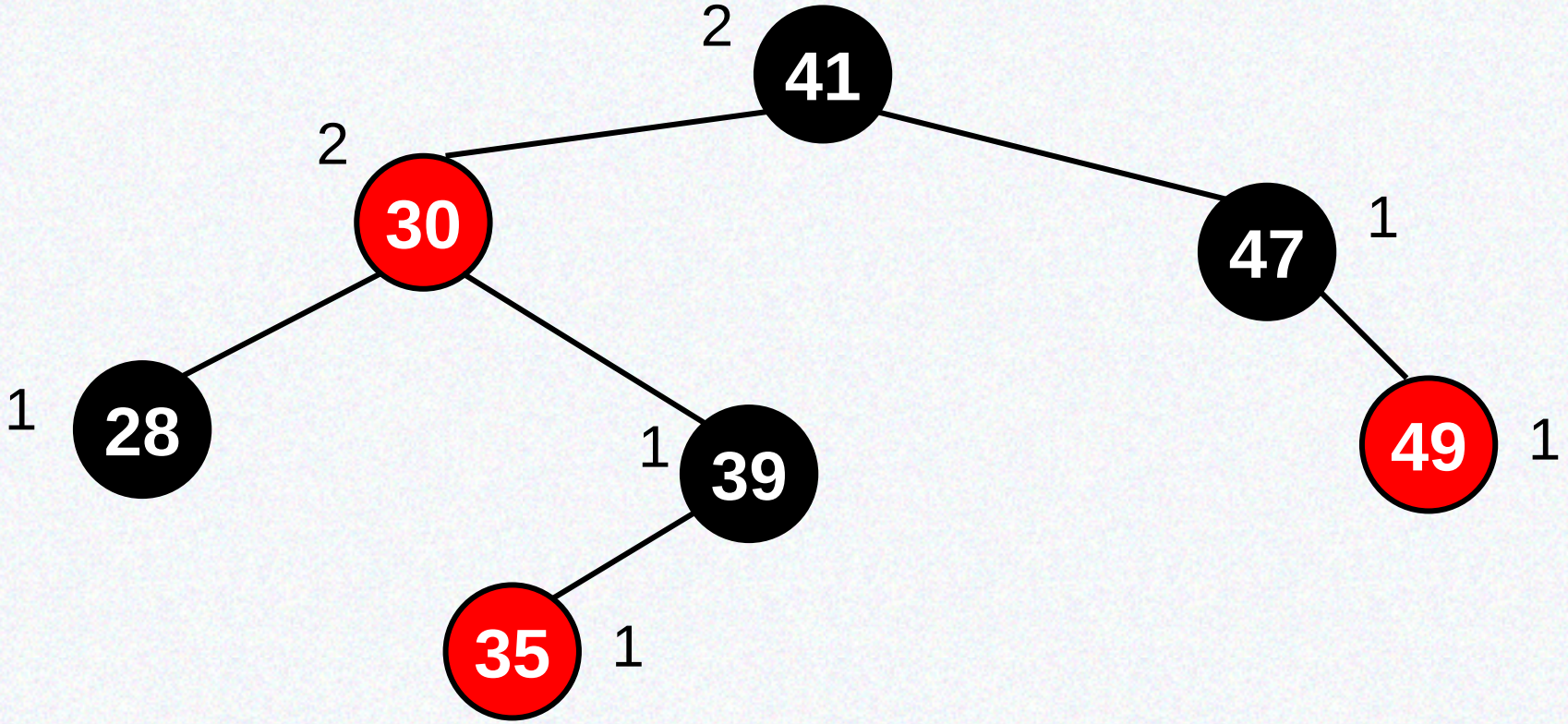
Примеры удаления



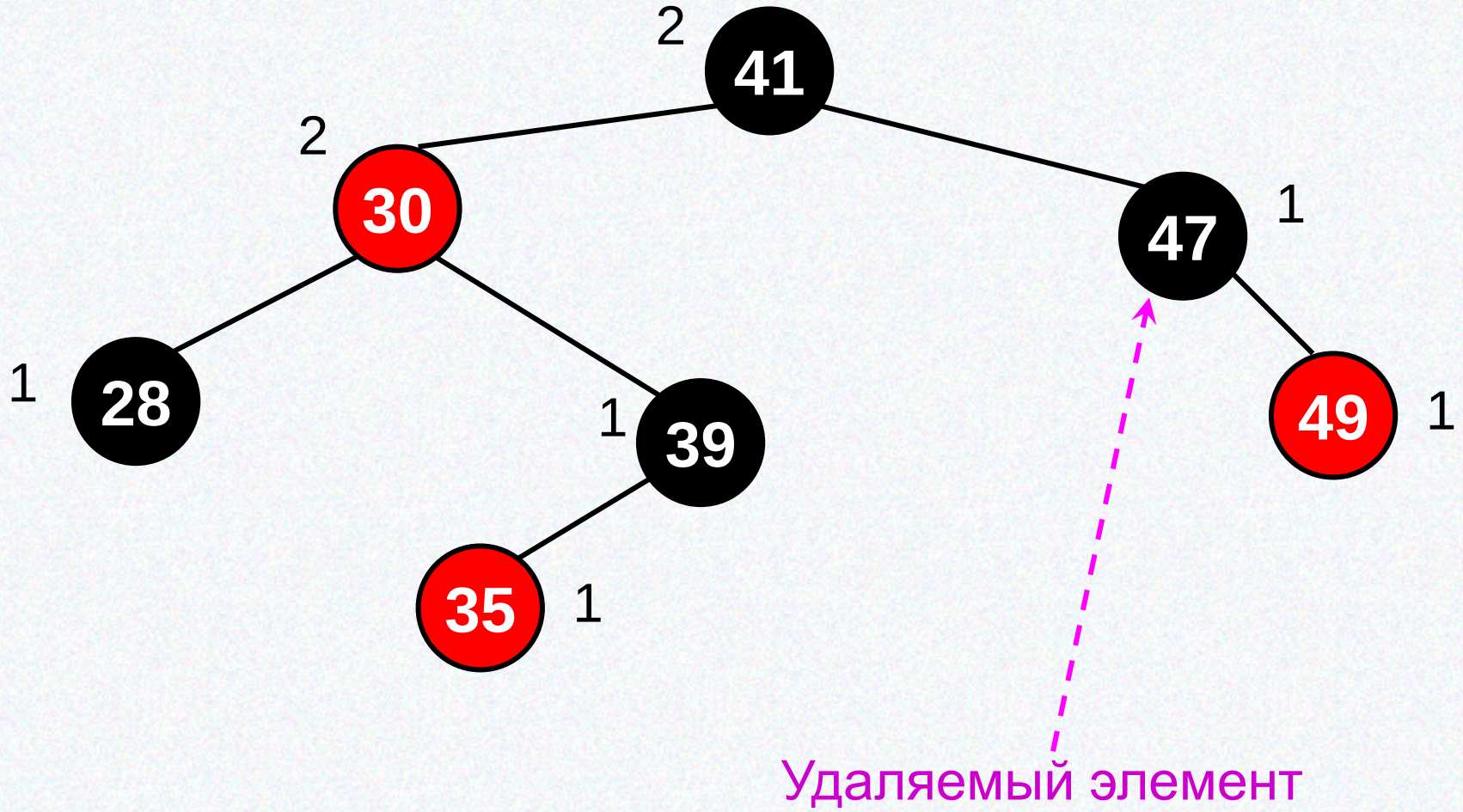
Примеры удаления



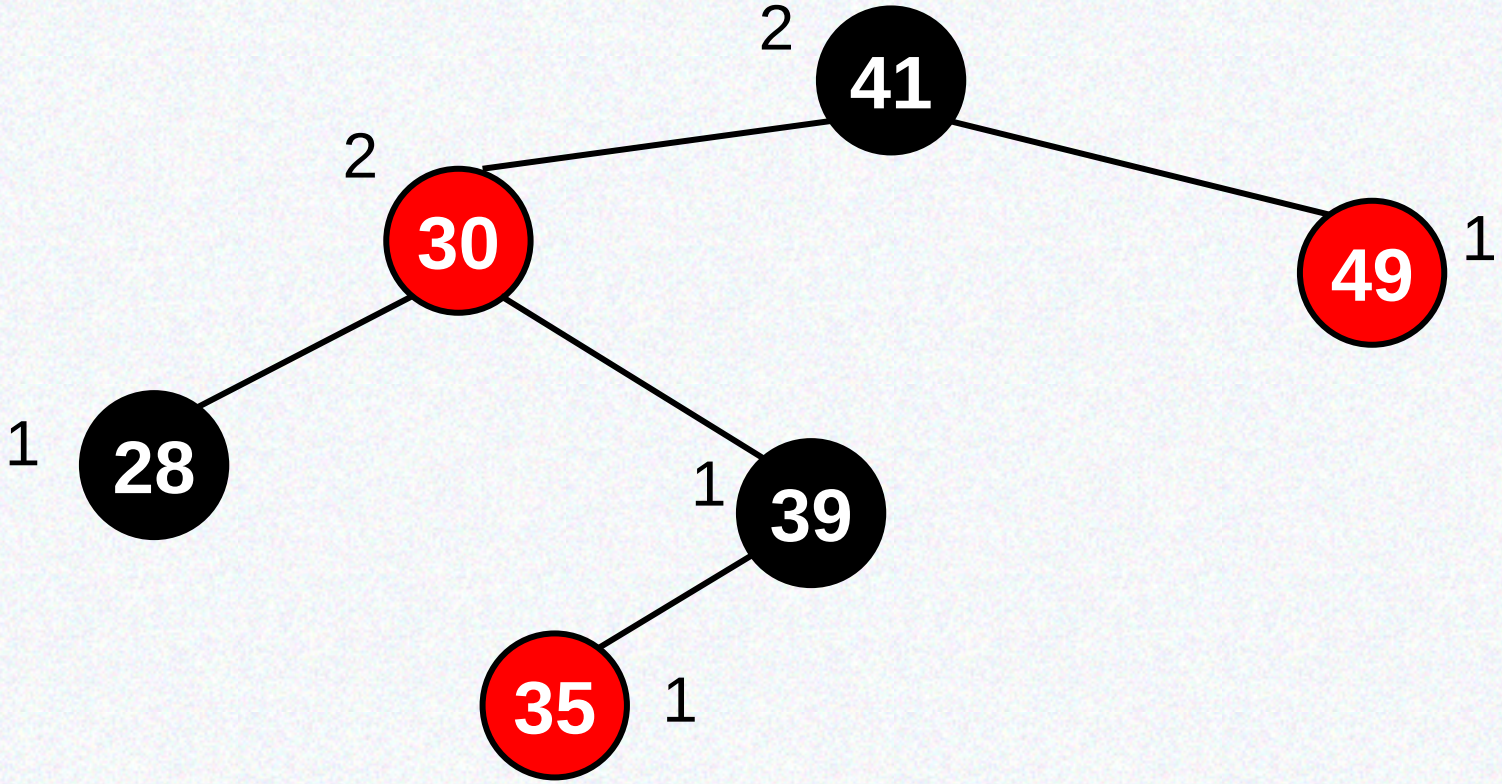
Примеры удаления



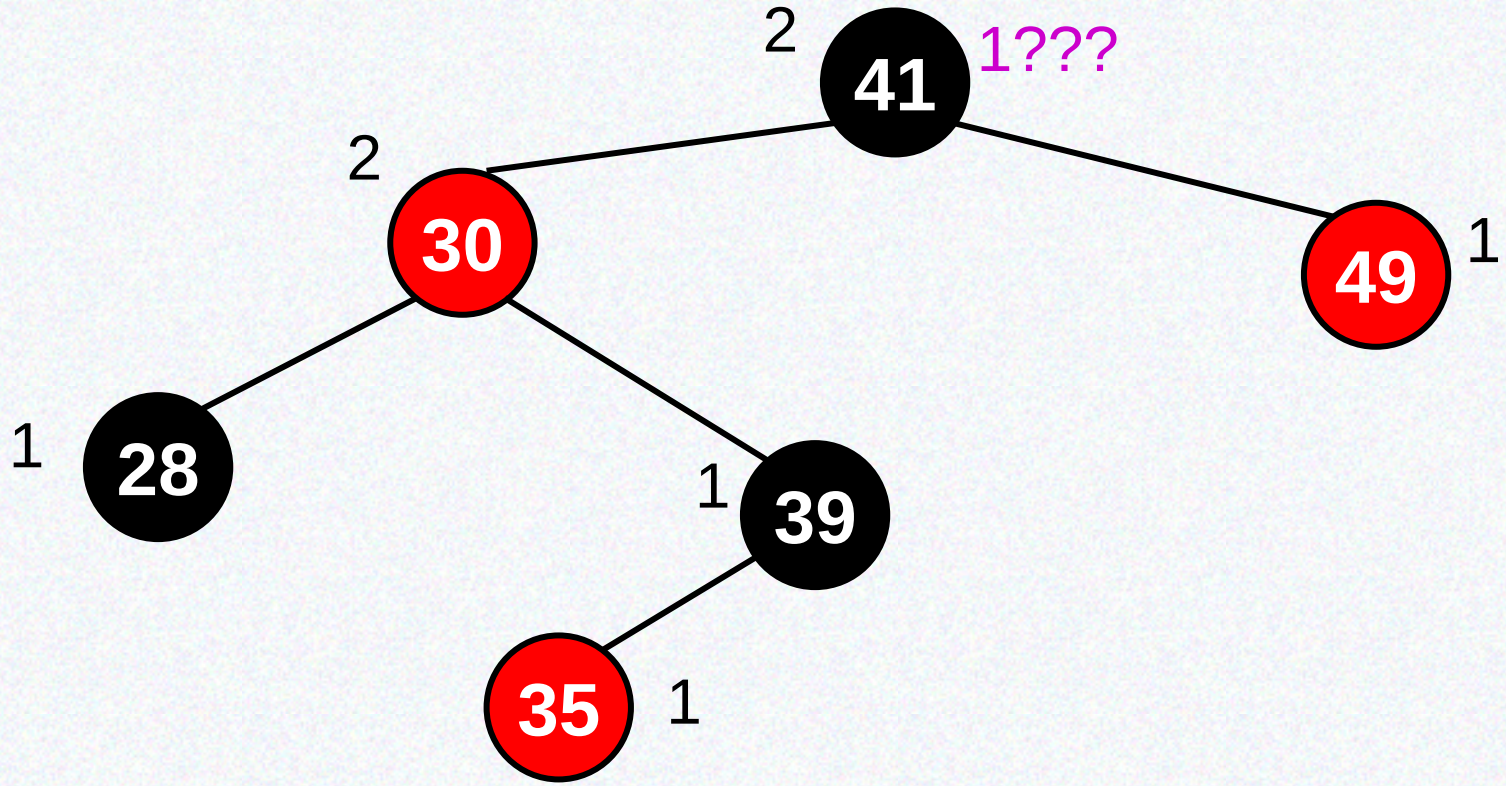
Примеры удаления



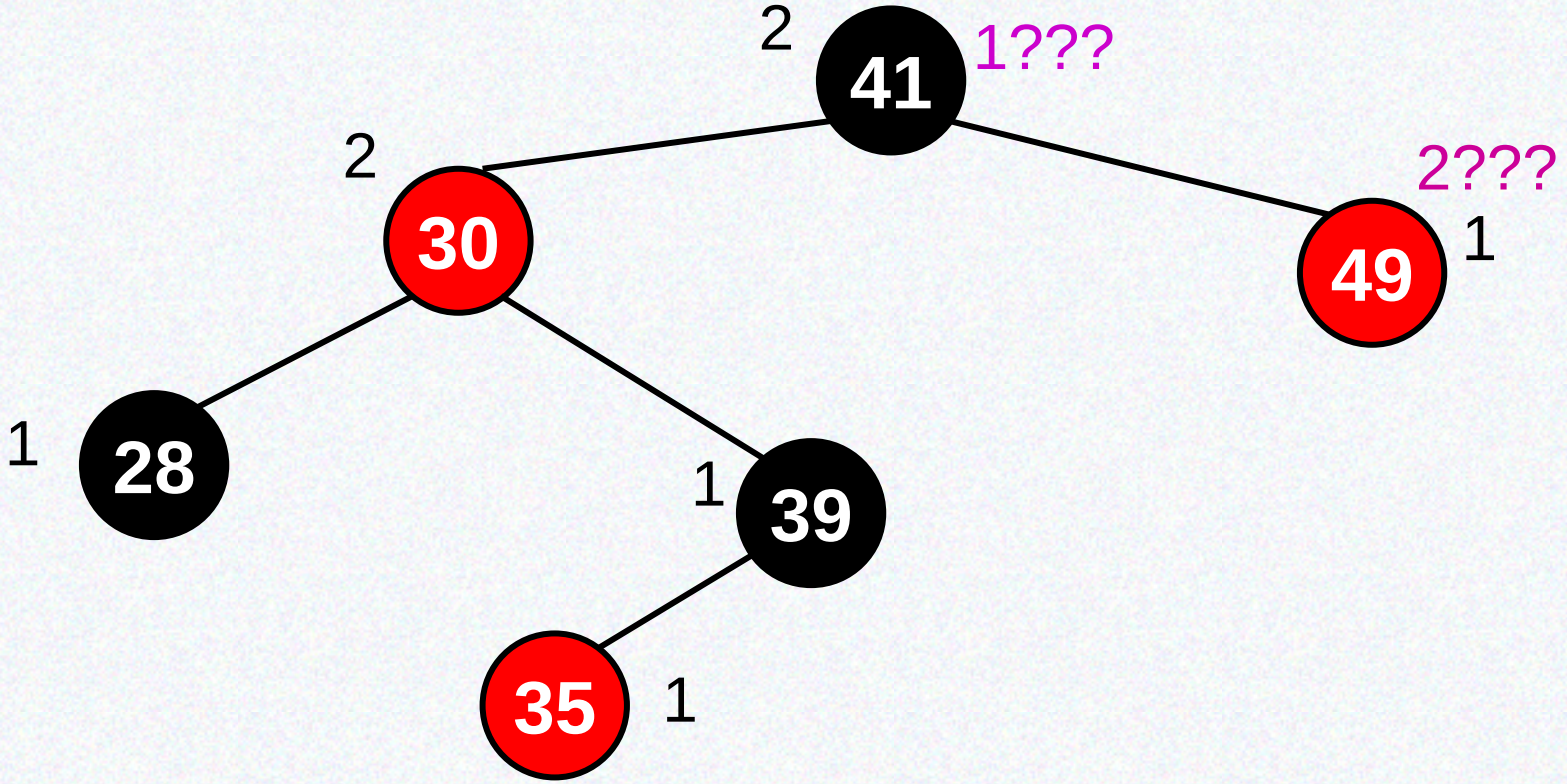
Примеры удаления



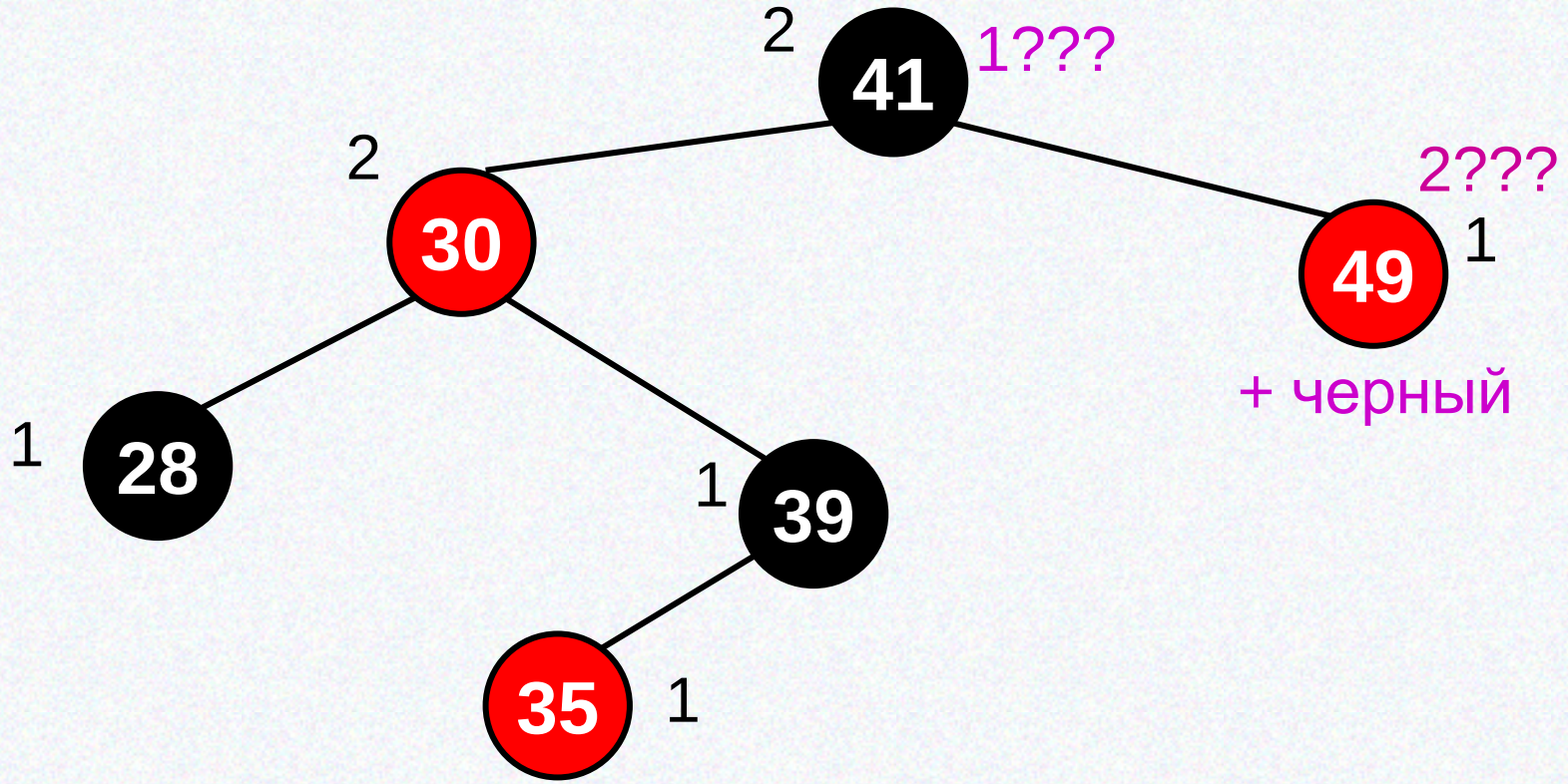
Примеры удаления



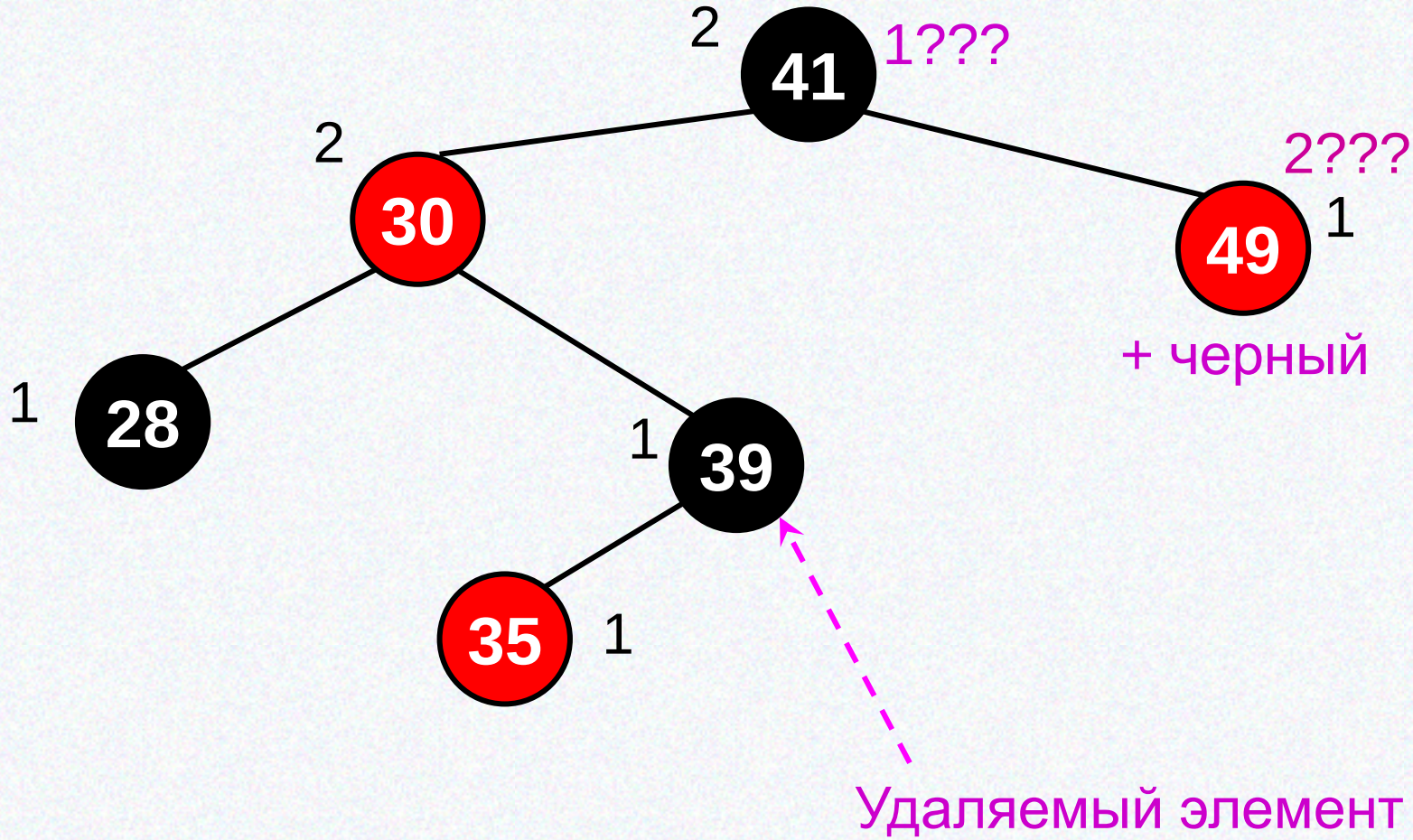
Примеры удаления



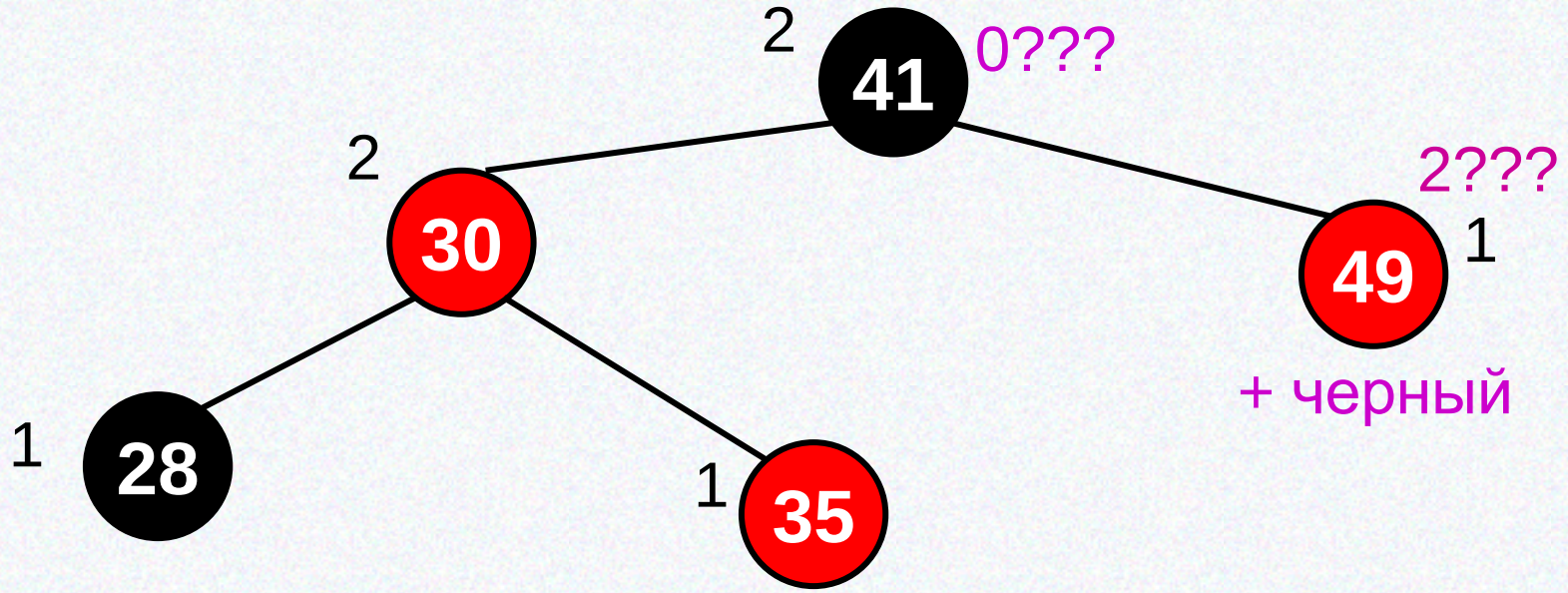
Примеры удаления



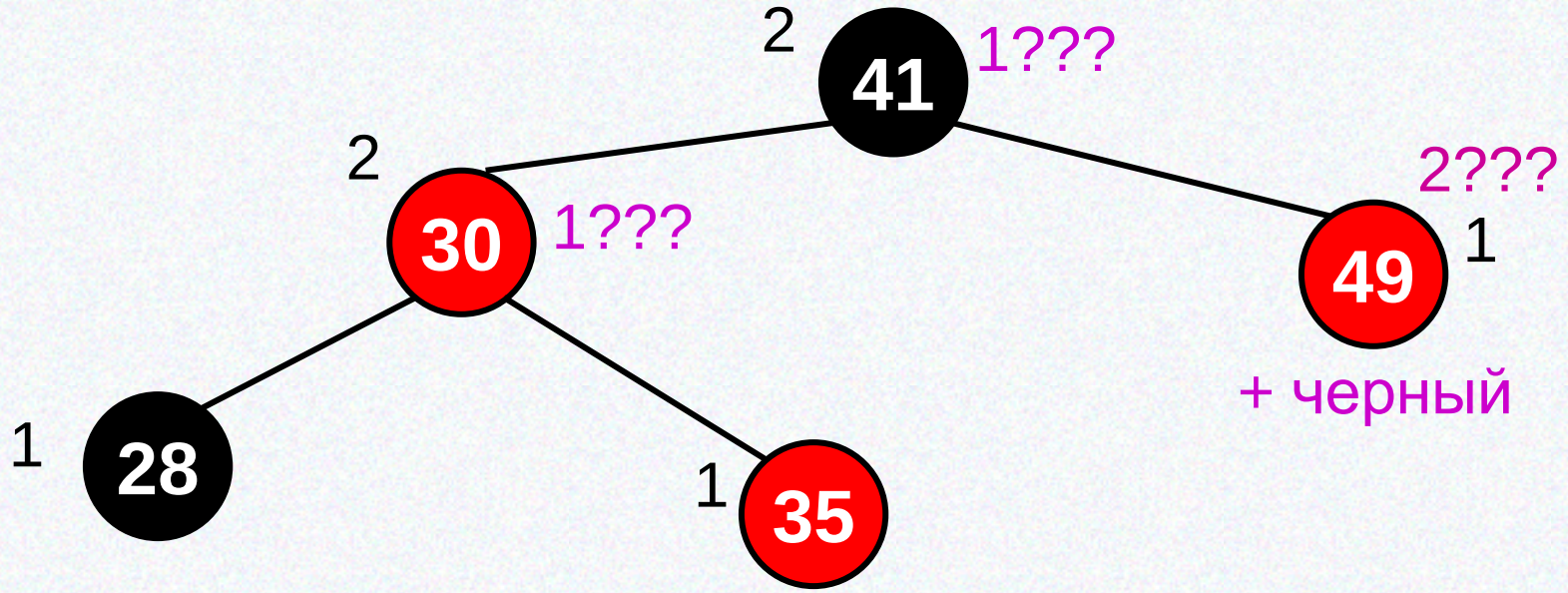
Примеры удаления



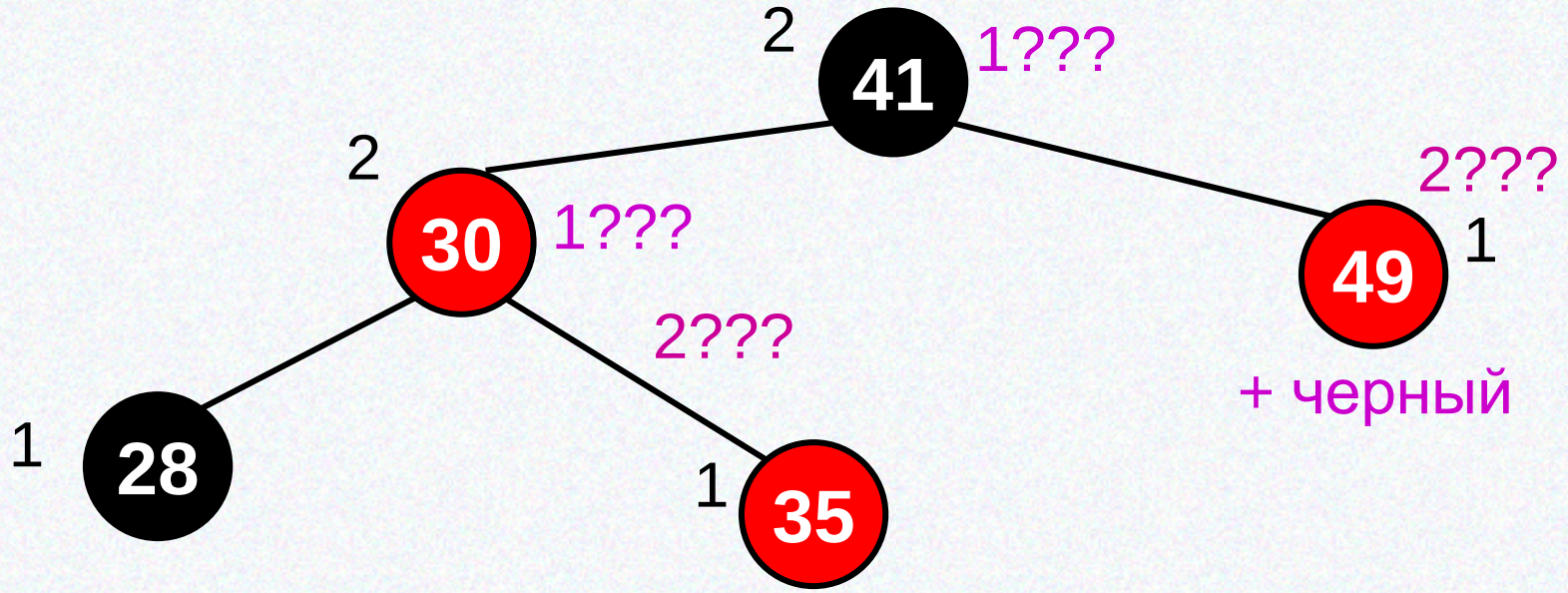
Примеры удаления



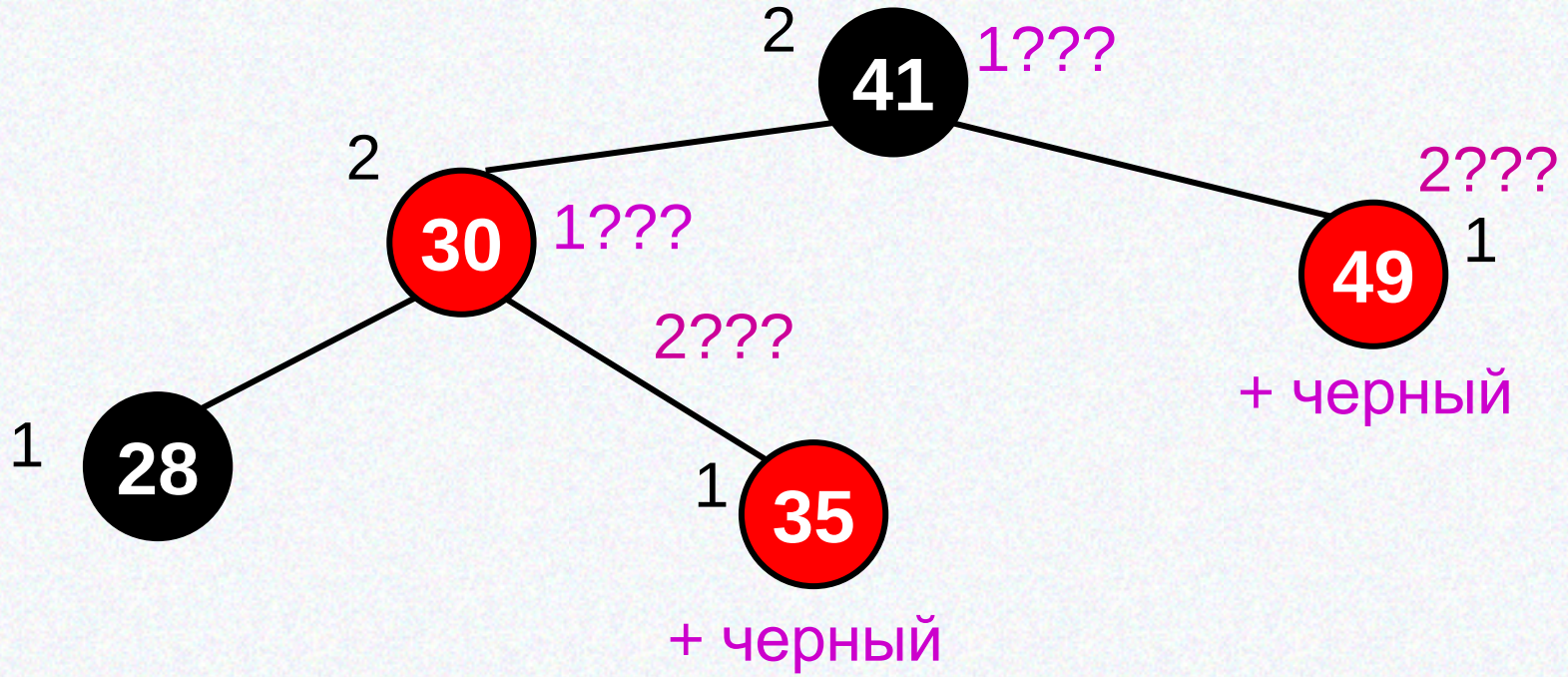
Примеры удаления



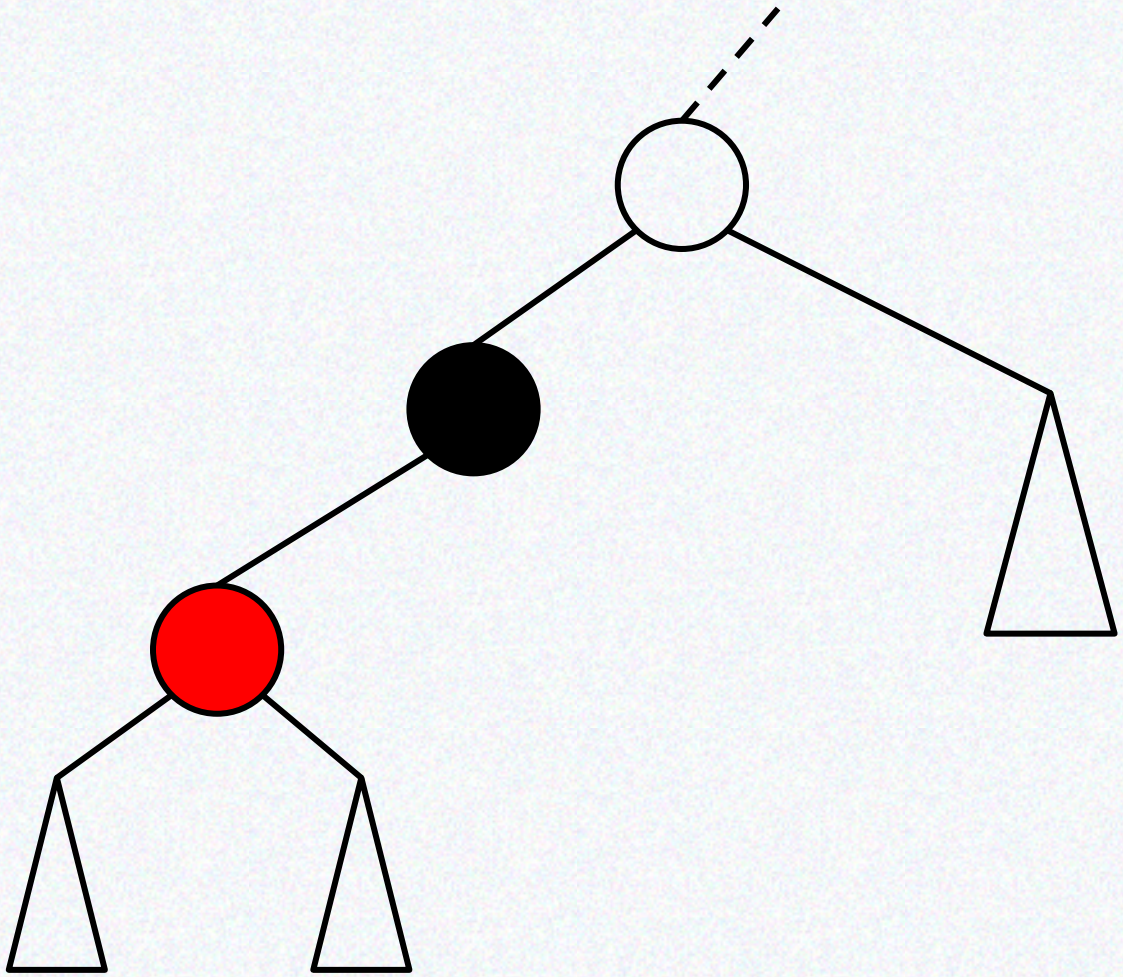
Примеры удаления



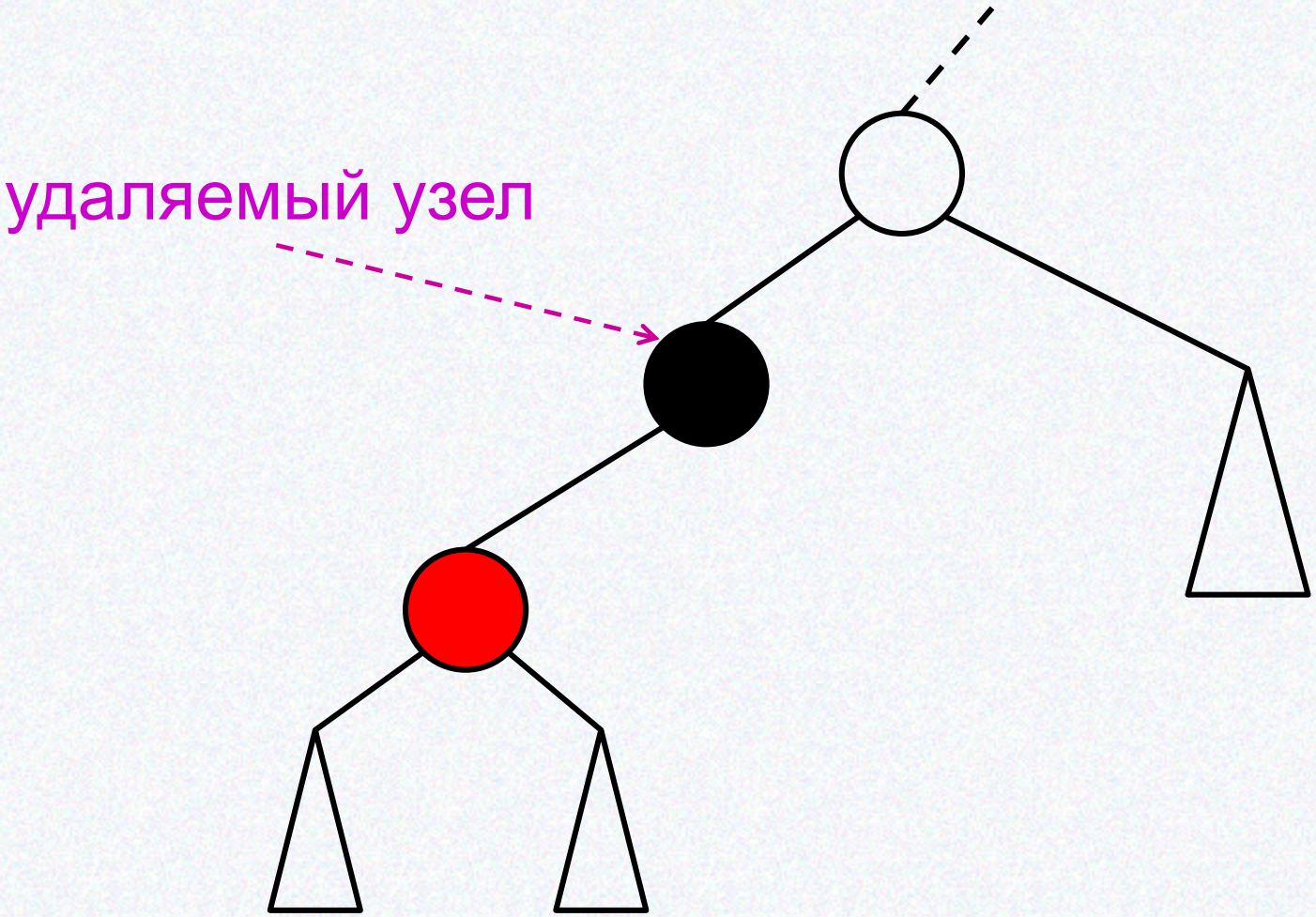
Примеры удаления



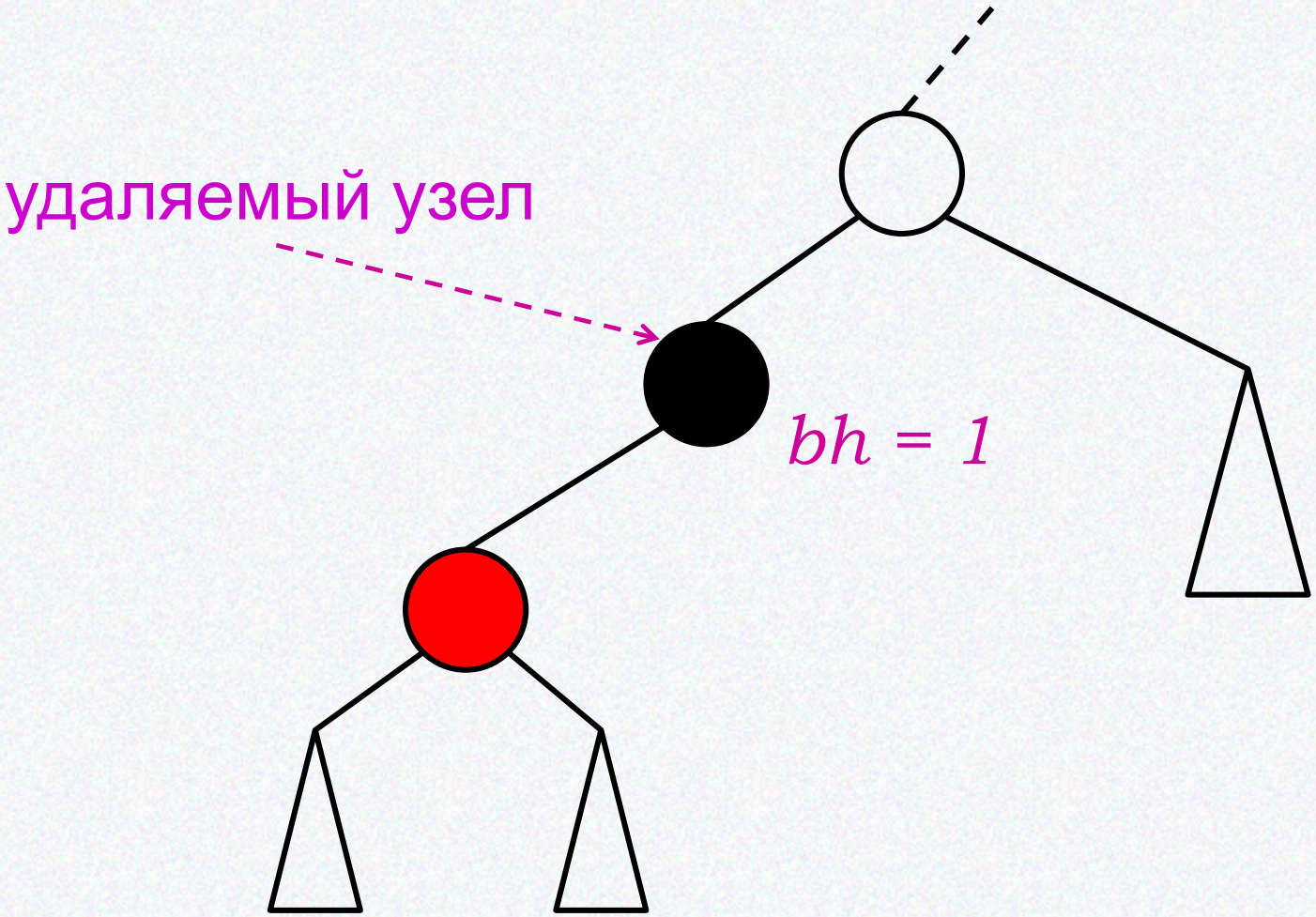
Примеры удаления



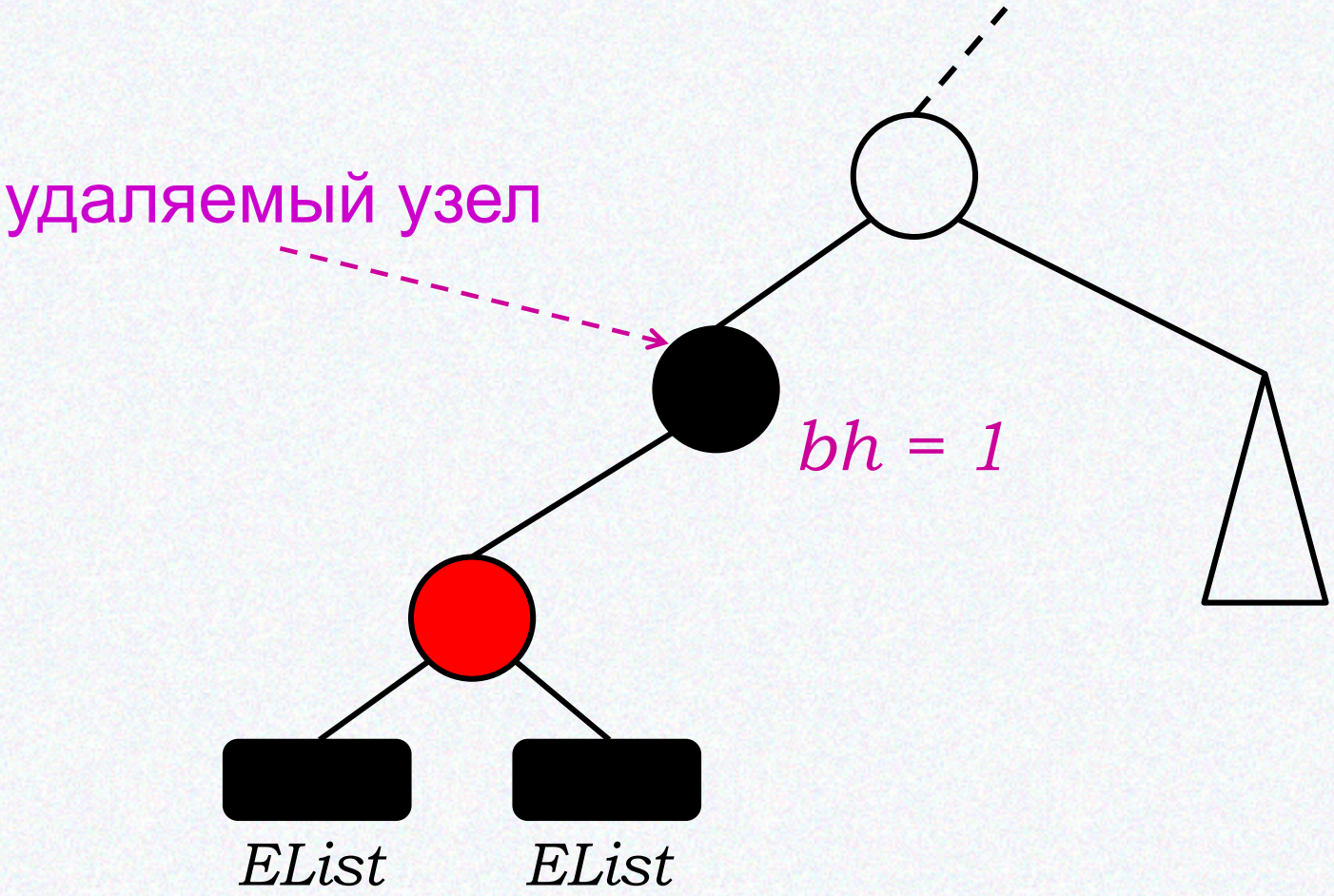
Примеры удаления



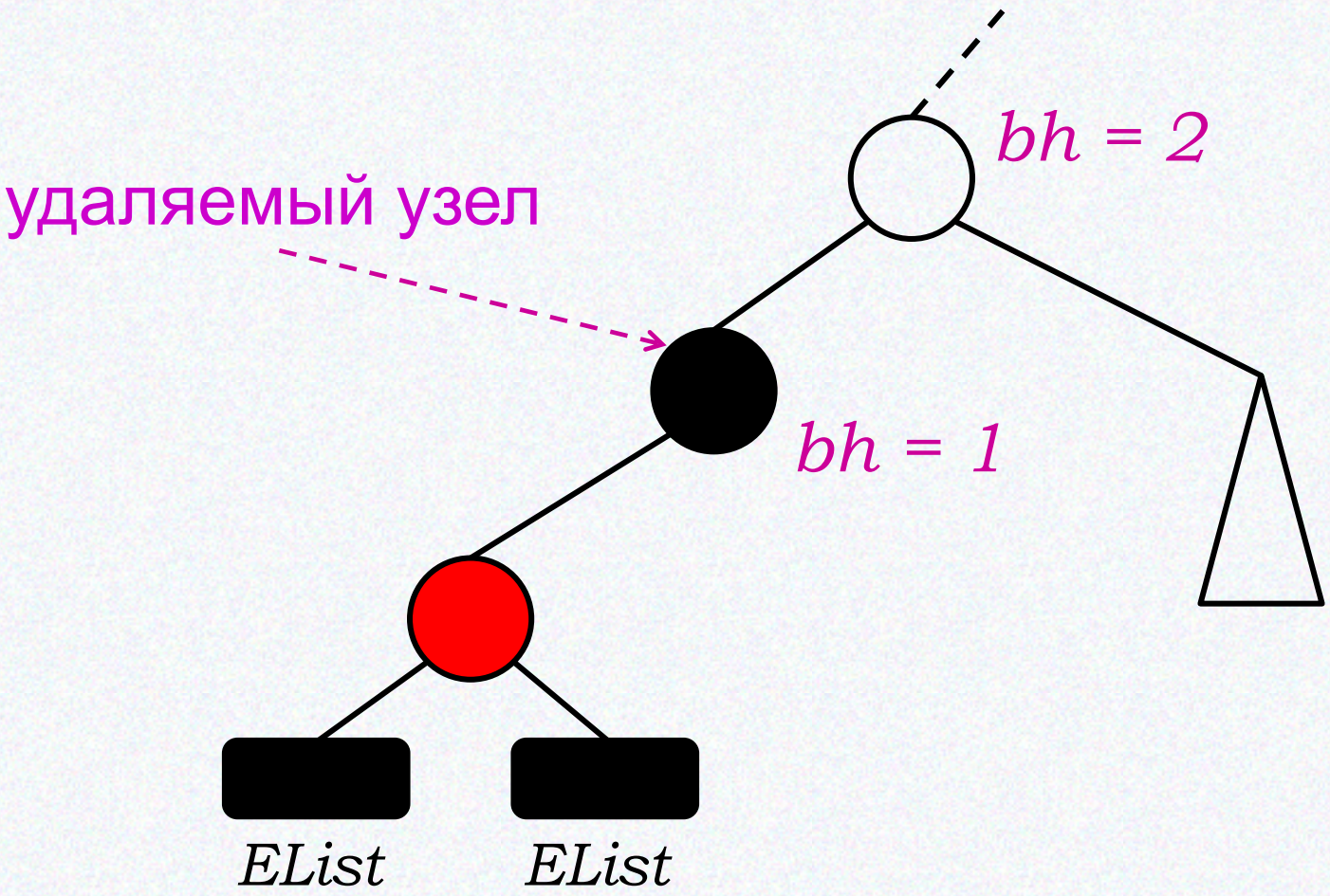
Примеры удаления



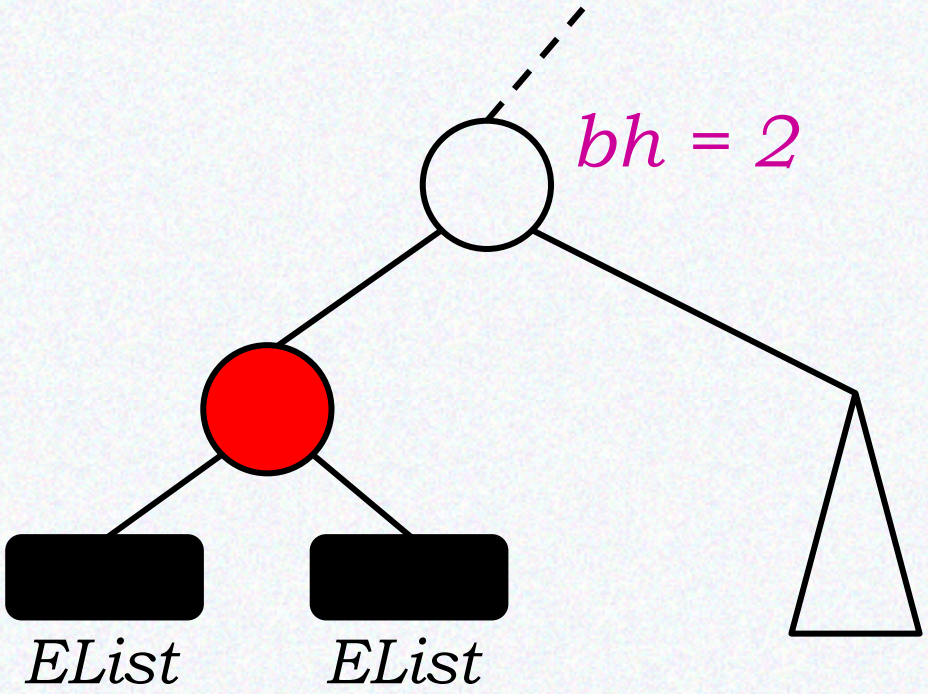
Примеры удаления



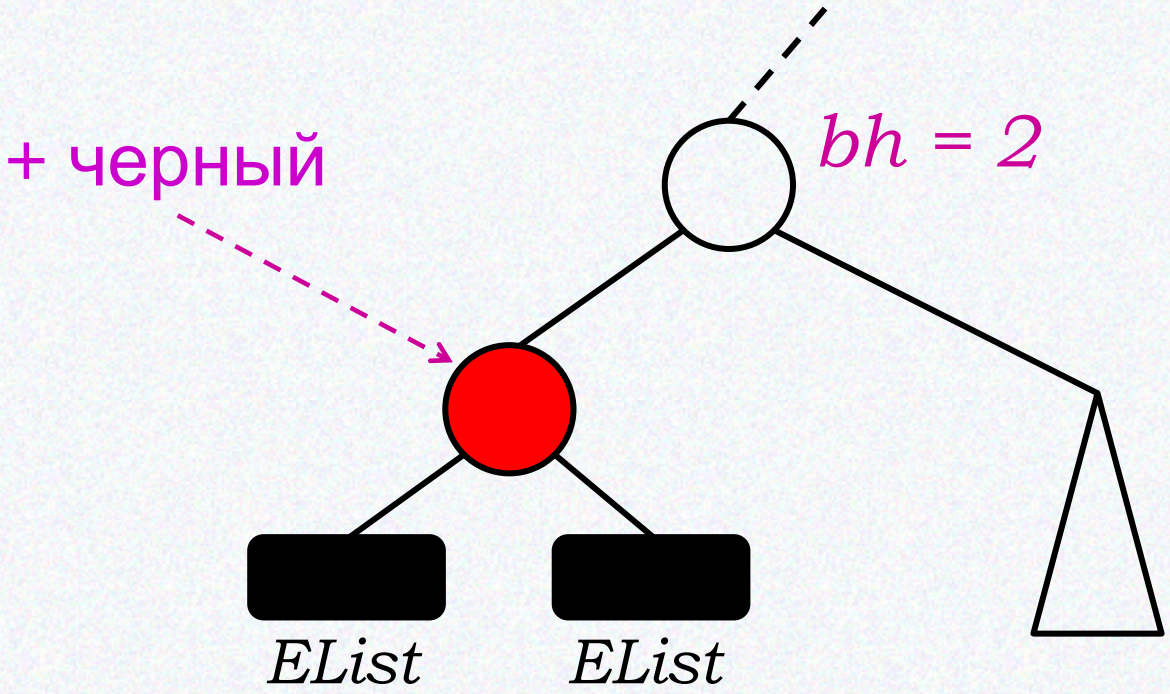
Примеры удаления



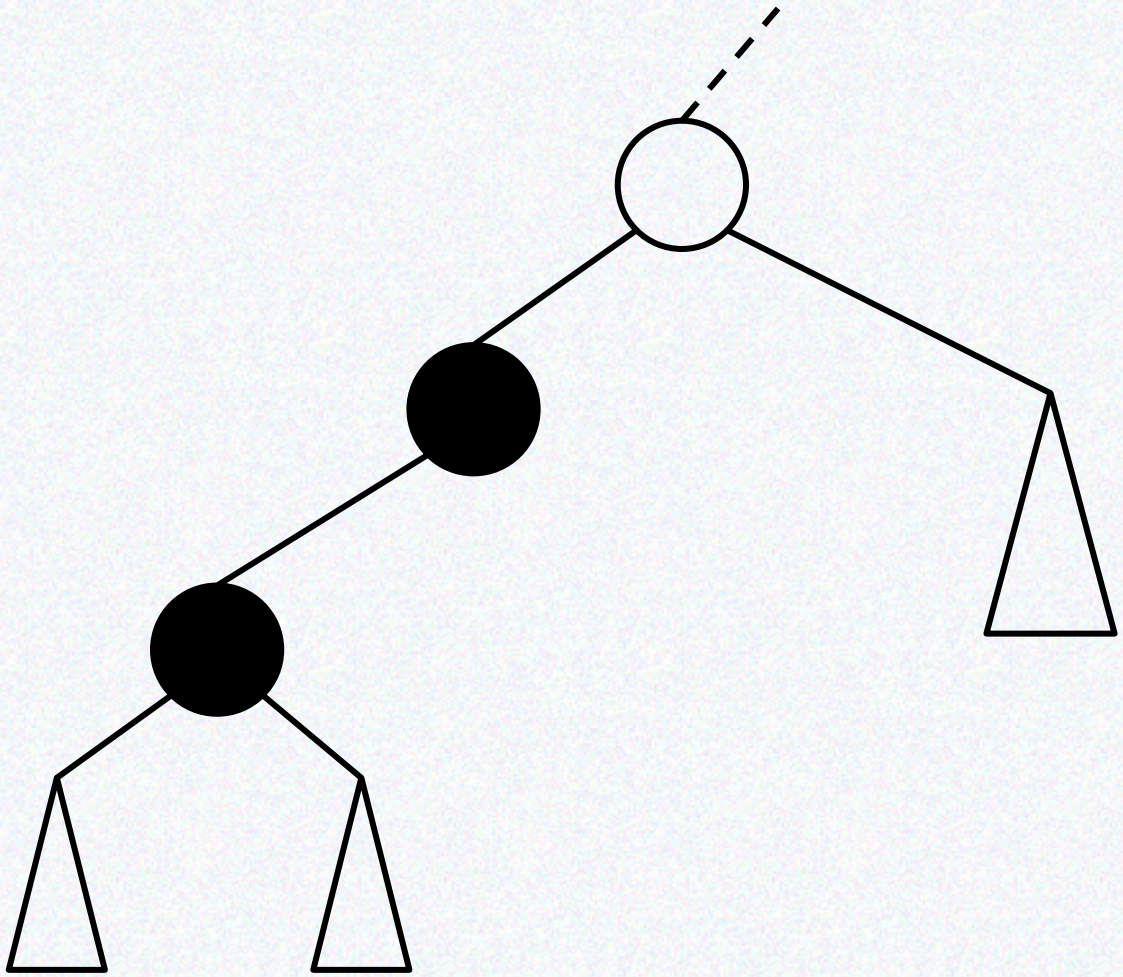
Примеры удаления



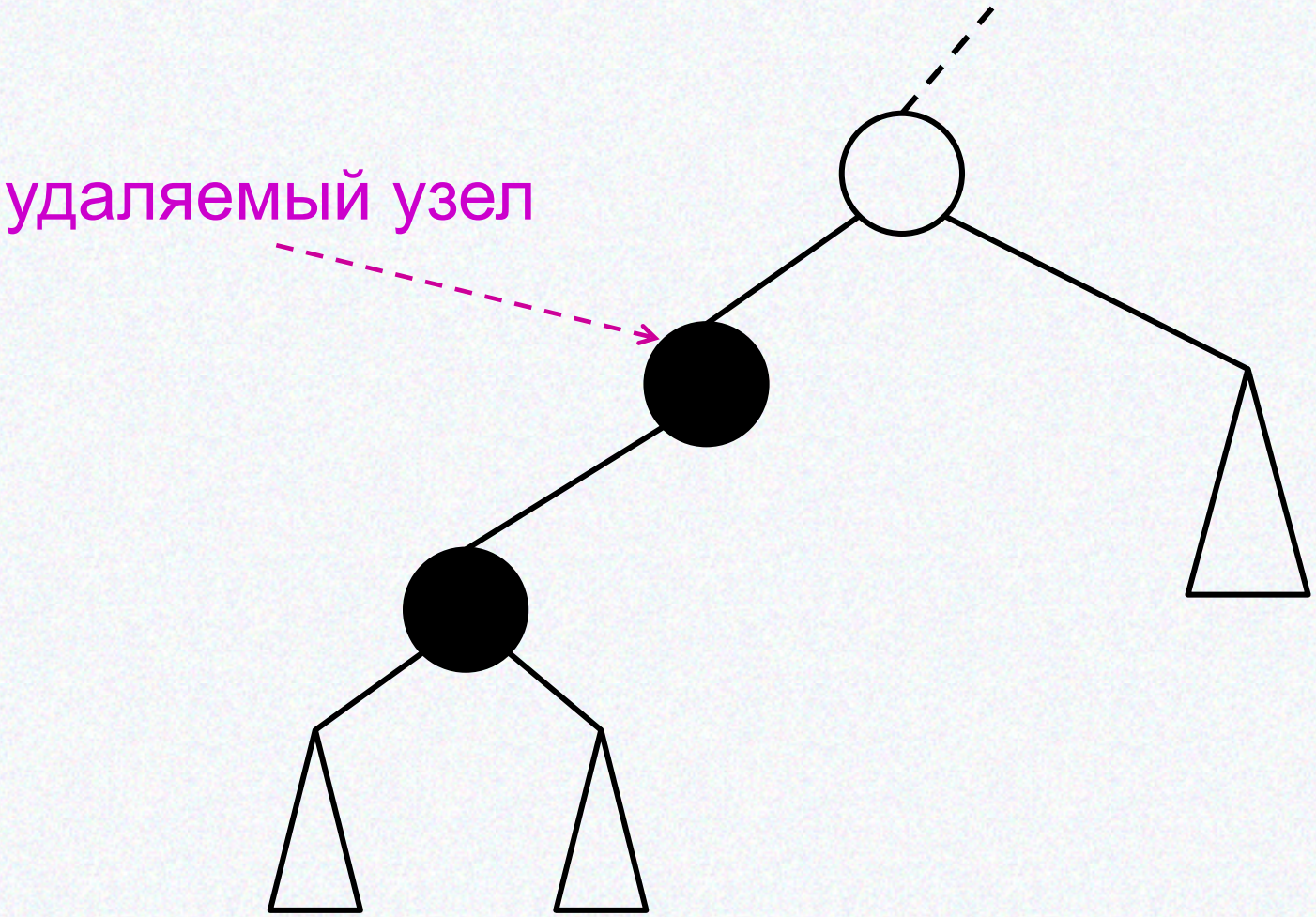
Примеры удаления



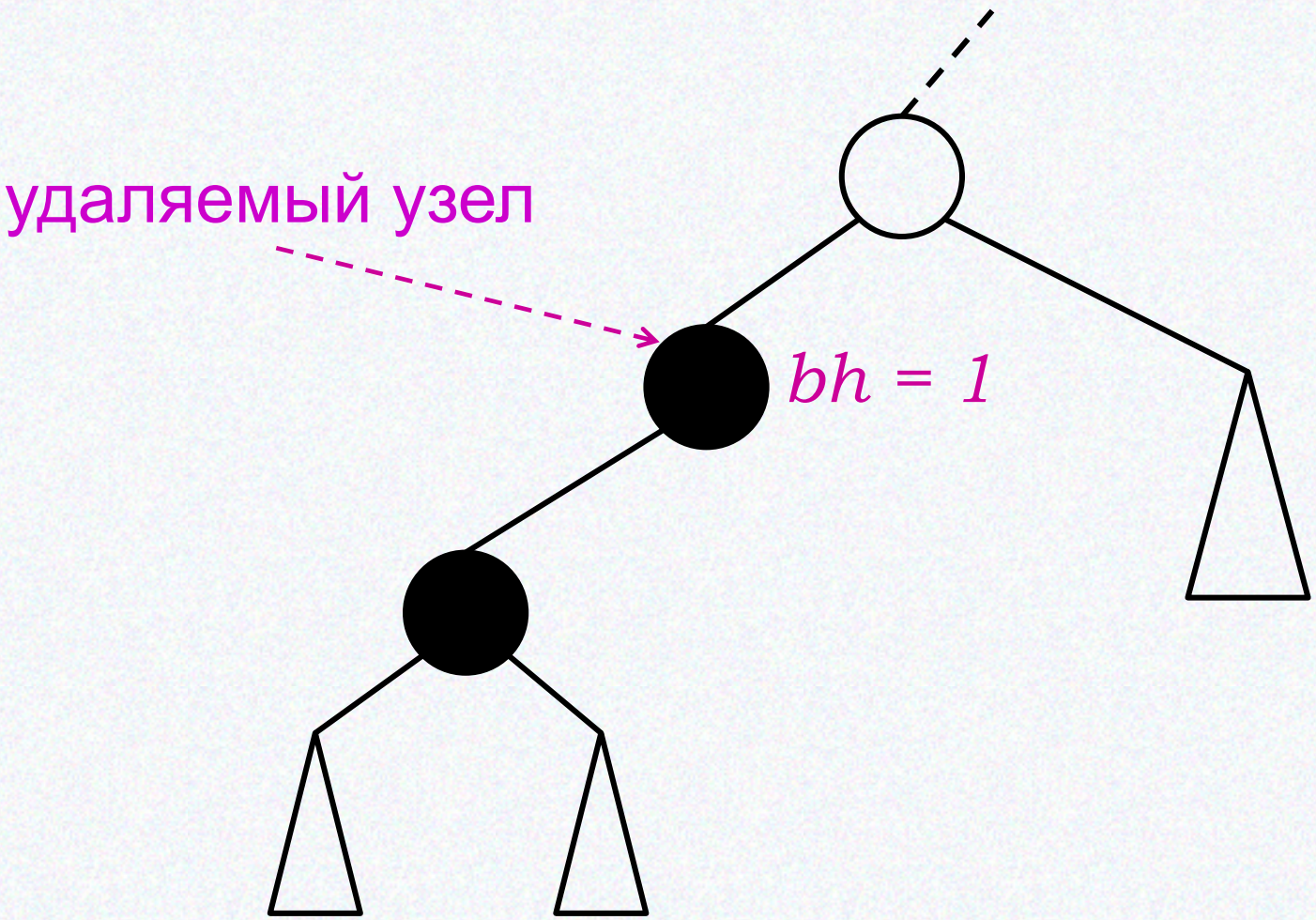
Примеры удаления



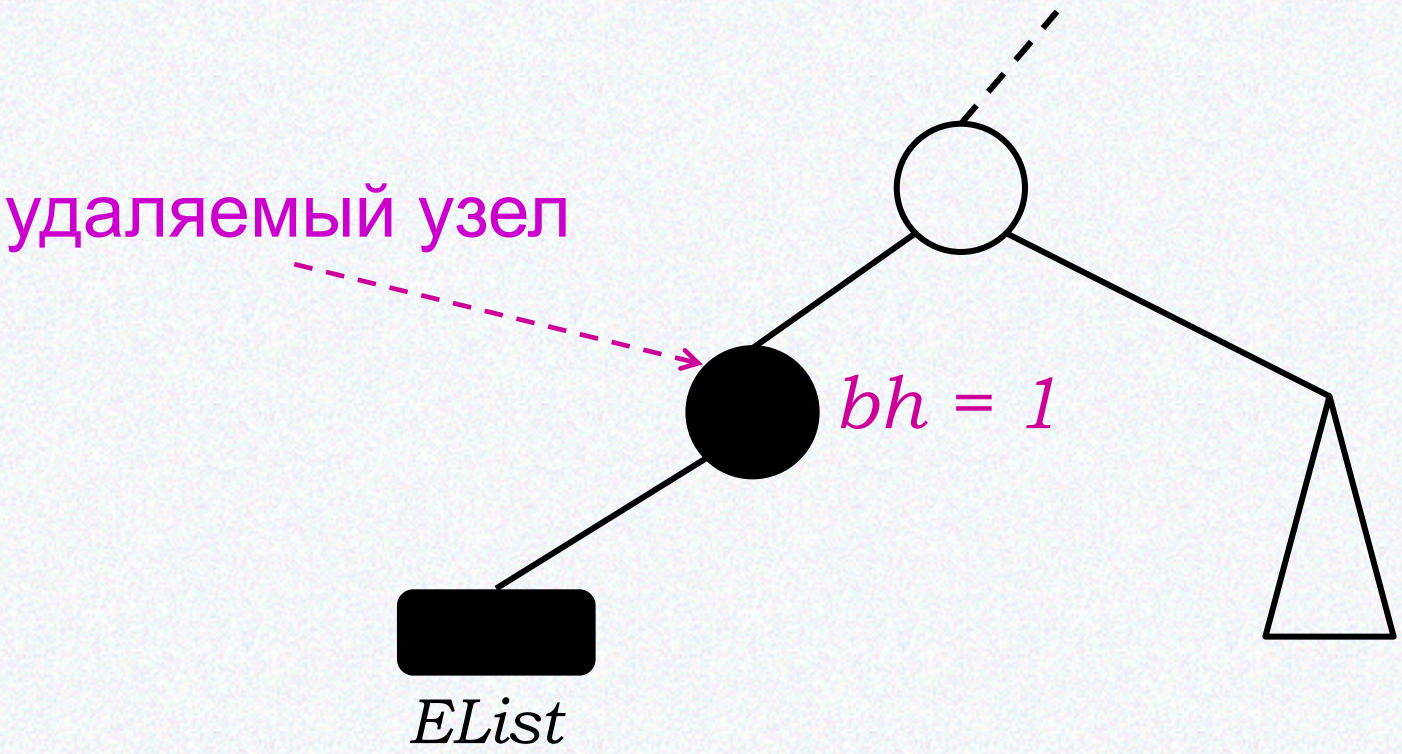
Примеры удаления



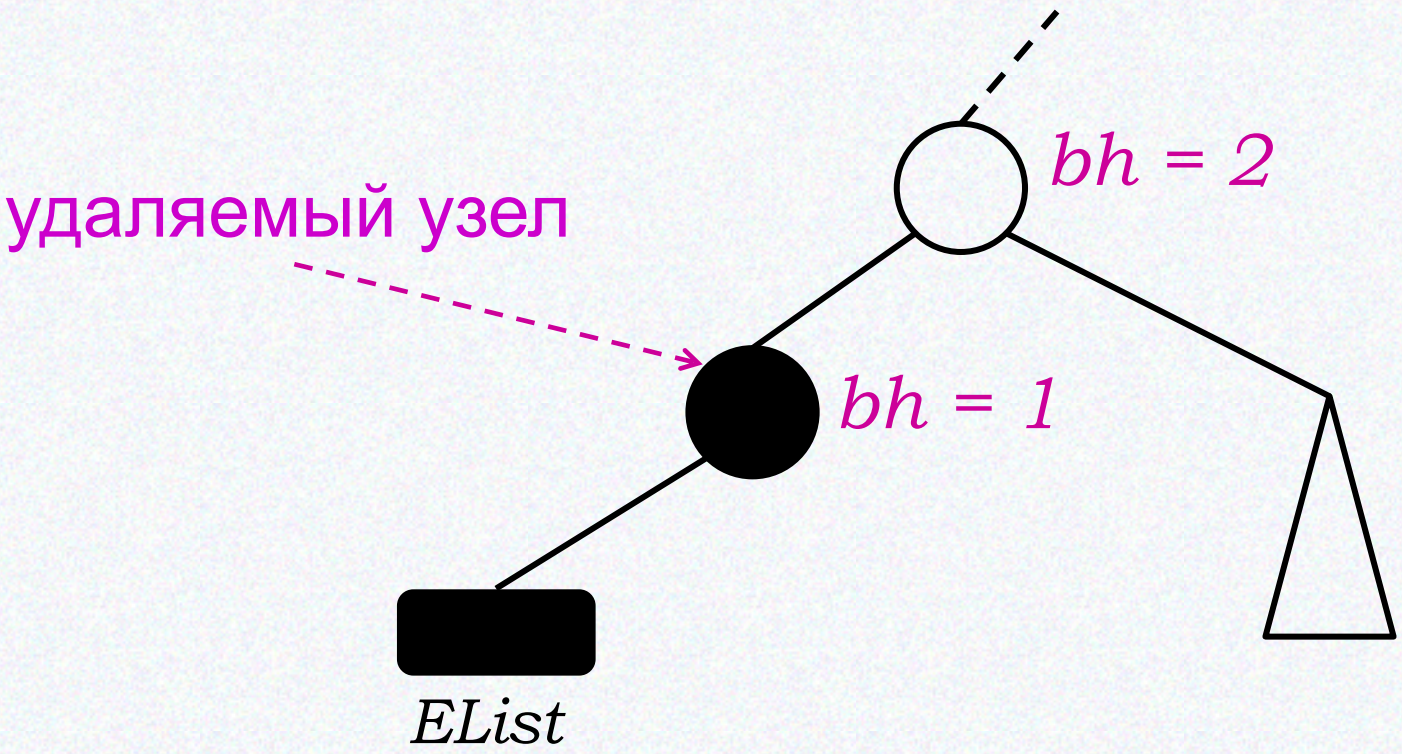
Примеры удаления



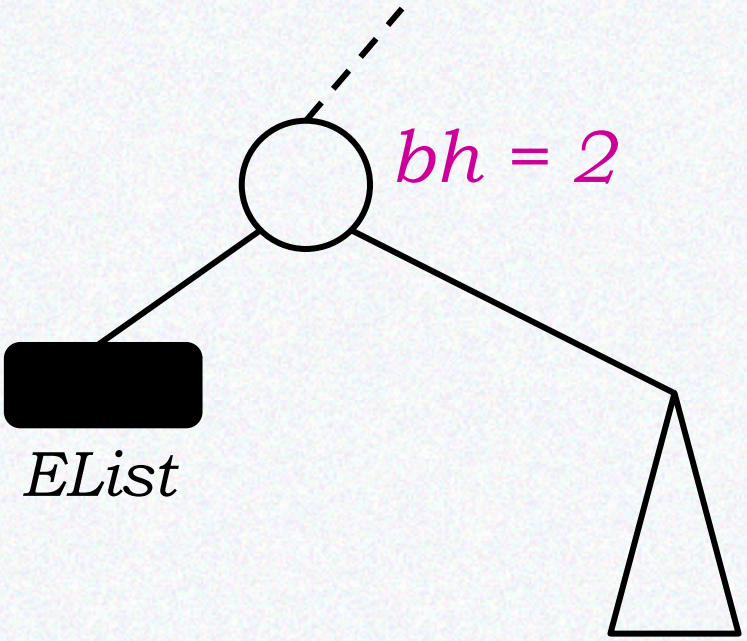
Примеры удаления



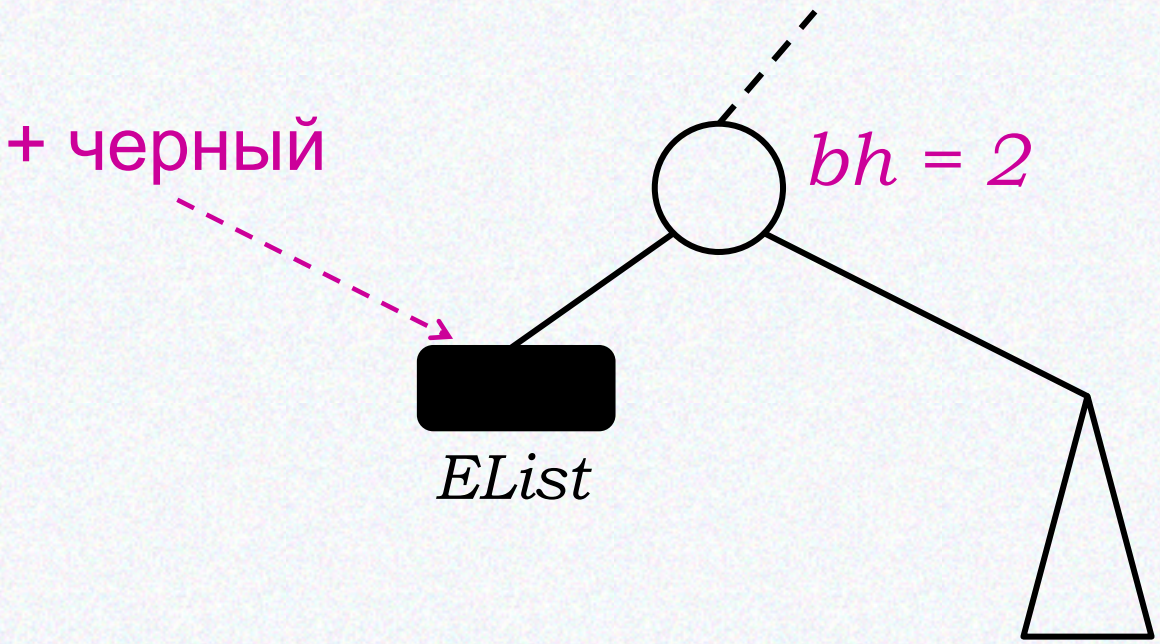
Примеры удаления



Примеры удаления



Примеры удаления



Восстановление черной высоты вершины

У потомка – дополнительная чернота:

Восстановление черной высоты вершины

У потомка – дополнительная чернота:

– «*дважды черный*» – атрибут $color$ узла имеет значение $BLACK$

Восстановление черной высоты вершины

У потомка – дополнительная чернота:

- «*дважды черный*» – атрибут $color$ узла имеет значение $BLACK$
- «*красно-черный*» – атрибут $color$ узла имеет значение RED

Восстановление черной высоты вершины

У потомка – дополнительная чернота:

- «*дважды черный*» – атрибут $color$ узла имеет значение $BLACK$
- «*красно-черный*» – атрибут $color$ узла имеет значение RED

- Восстанавливается свойство 5

Восстановление черной высоты вершины

У потомка – дополнительная чернота:

- «*дважды черный*» – атрибут $color$ узла имеет значение $BLACK$
- «*красно-черный*» – атрибут $color$ узла имеет значение RED

- Восстанавливается свойство 5
- Нарушается свойство 1 – значение атрибута $color$ узла не соответствует цвету узла

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

1. Узел является *красно-черным* –

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

1. Узел является *красно-черным* –

   сделать данный узел «единожды черным»

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

1. Узел является *красно-черным* –

   сделать данный узел «единожды черным»

2. Узел является *корнем дерева* –

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

1. Узел является *красно-черным* –

   сделать данный узел «единожды черным»

2. Узел является *корнем дерева* –

   убрать излишнюю черноту

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

1. Узел является *красно-черным* –

   сделать данный узел «единожды черным»

2. Узел является *корнем дерева* –

   убрать излишнюю черноту

3. Узел является *дважды черным* –

Восстановление цвета узла

Переместить дополнительную черноту вверх по дереву:

1. Узел является *красно-черным* –

   сделать данный узел «единожды черным»

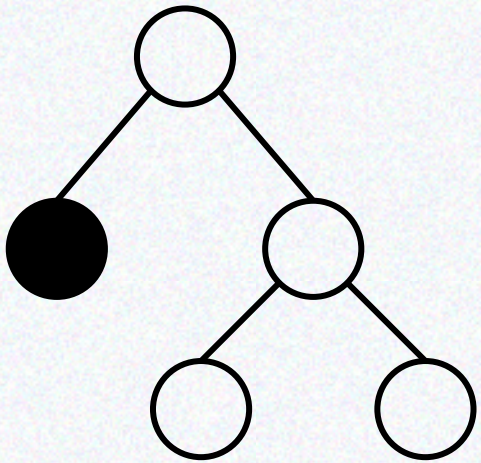
2. Узел является *корнем дерева* –

   убрать излишнюю черноту

3. Узел является *дважды черным* –
   выполнить повороты и перекраску, устраняющие двойную черноту

Восстановление цвета узла

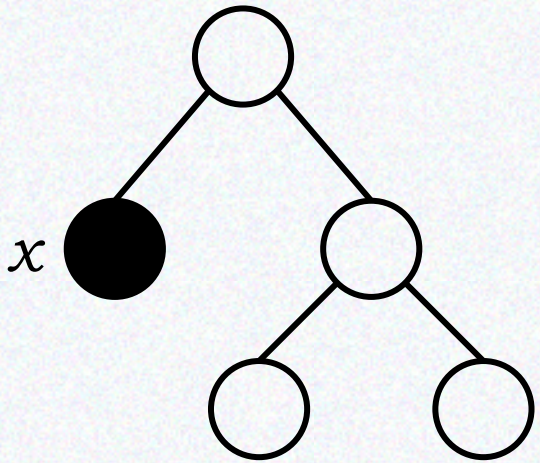
Обозначения:



Восстановление цвета узла

Обозначения:

x – анализируемый узел дерева

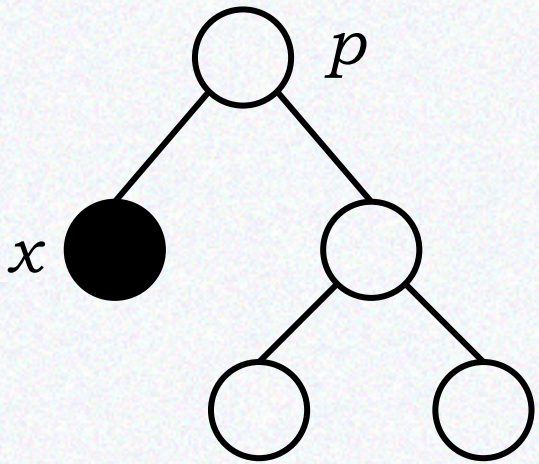


Восстановление цвета узла

Обозначения:

x – анализируемый узел дерева

$p = x\text{->}parent$ – родительский узел



Восстановление цвета узла

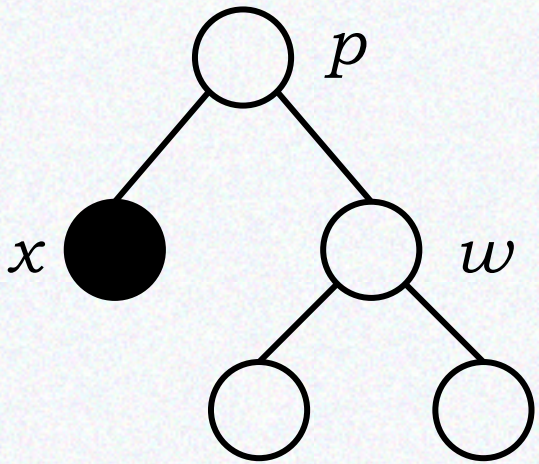
Обозначения:

x – анализируемый узел дерева

$p = x\text{->}parent$ – родительский узел

Пусть $x = p\text{->}left$

$w = p\text{->}right$ – второй потомок узла p



Восстановление цвета узла

Обозначения:

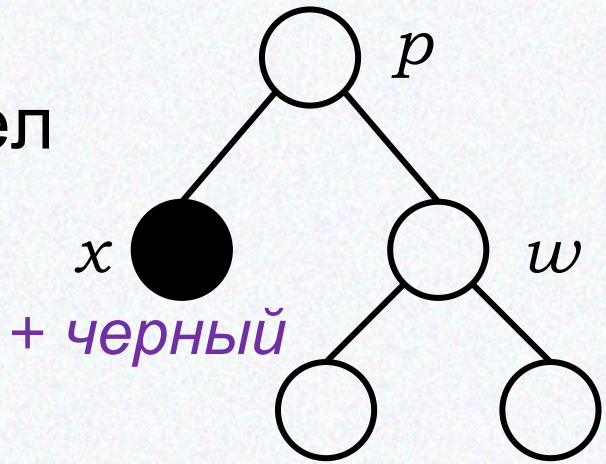
x – анализируемый узел дерева

$p = x\text{->}parent$ – родительский узел

Пусть $x = p\text{->}left$

$w = p\text{->}right$ – второй потомок узла p

x – *дважды черный узел*



Восстановление цвета узла

Обозначения:

x – анализируемый узел дерева

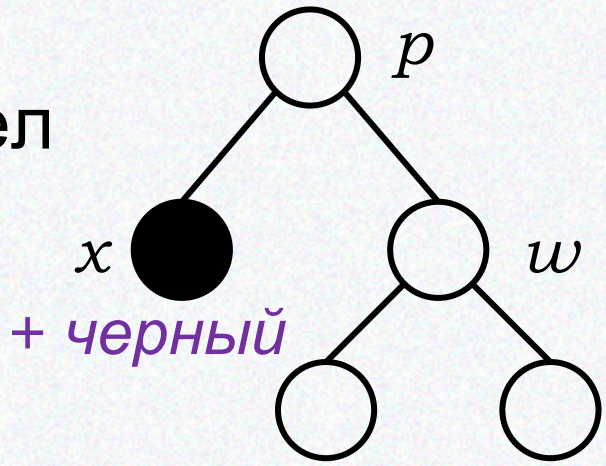
$p = x\text{->}parent$ – родительский узел

Пусть $x = p\text{->}left$

$w = p\text{->}right$ – второй потомок узла p

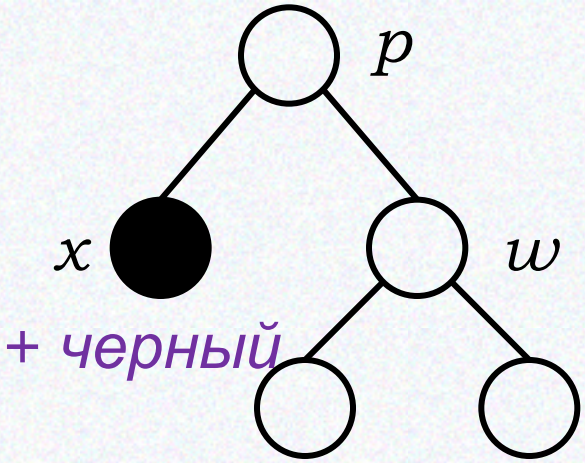
x – *дважды черный узел*

$w \neq EList$



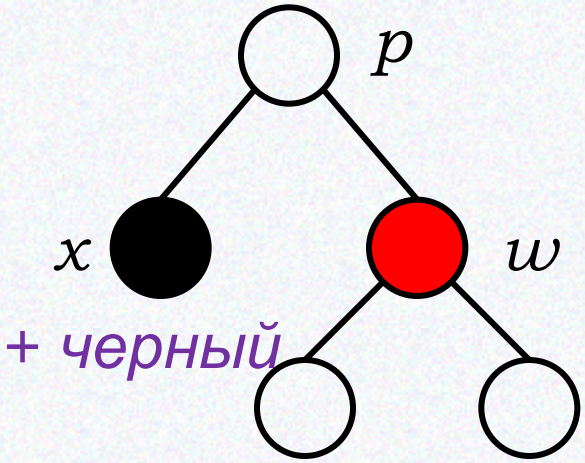
Случай 1

Узел w – красный



Случай 1

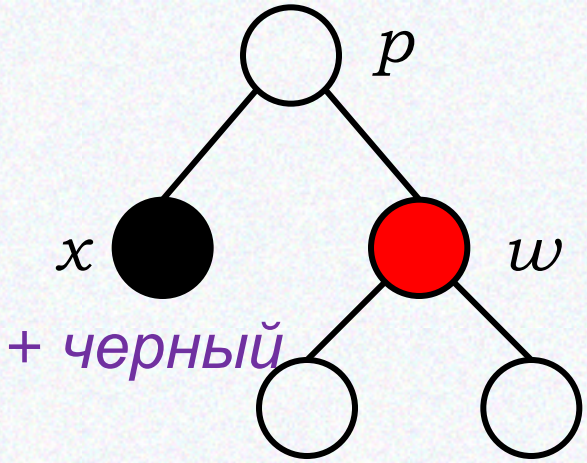
Узел w – красный



Случай 1

Узел w – красный

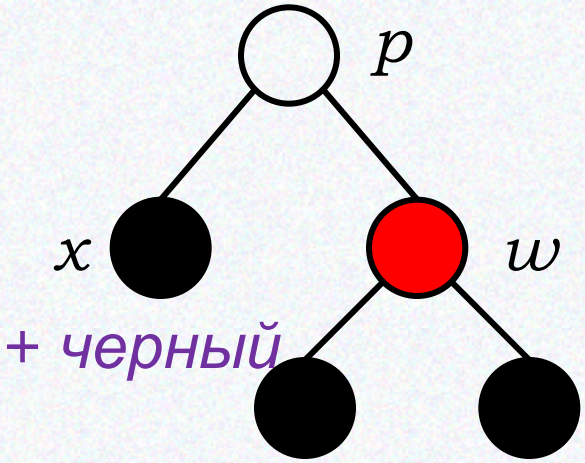
Потомки w – черные



Случай 1

Узел w – красный

Потомки w – черные

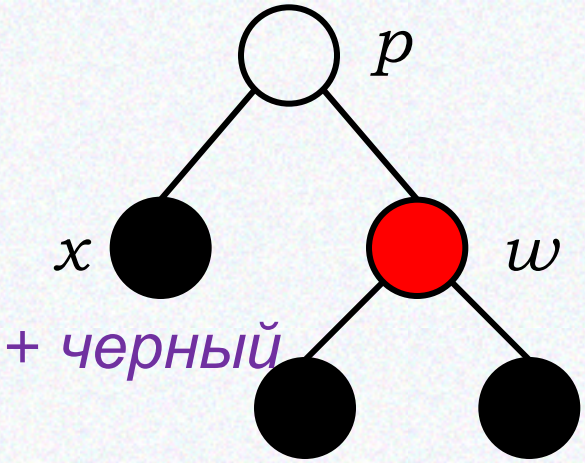


Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный

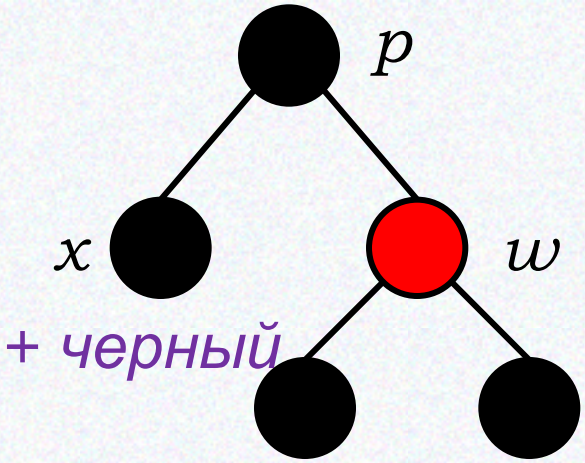


Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный

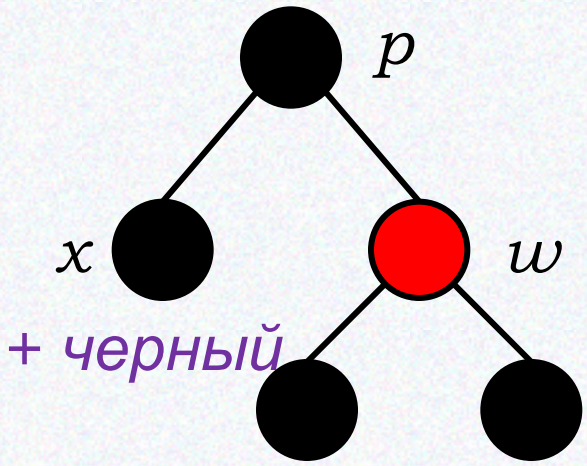


Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный



Коррекция:

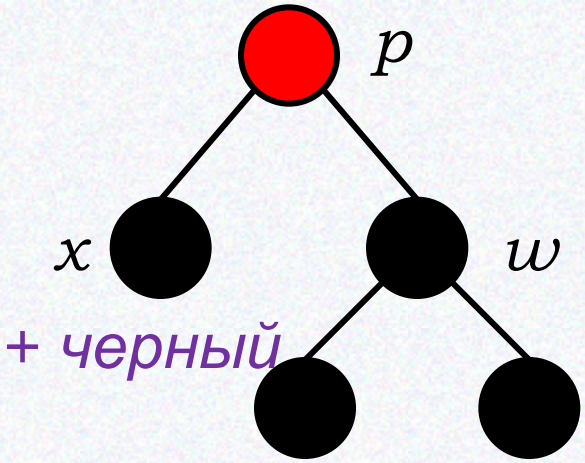
1. Обменять цвета w и p

Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный



Коррекция:

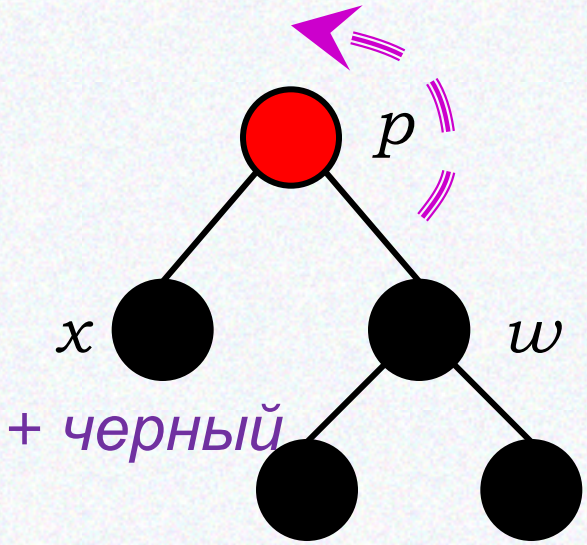
1. Обменять цвета w и p

Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный



Коррекция:

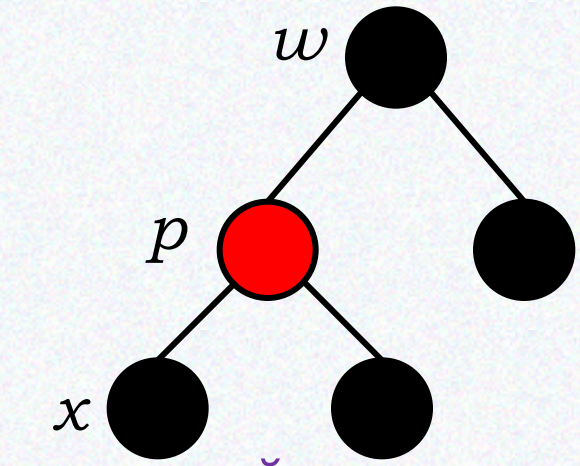
1. Обменять цвета w и p
2. Выполнить левый поворот вокруг узла p

Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный



+ черный

Коррекция:

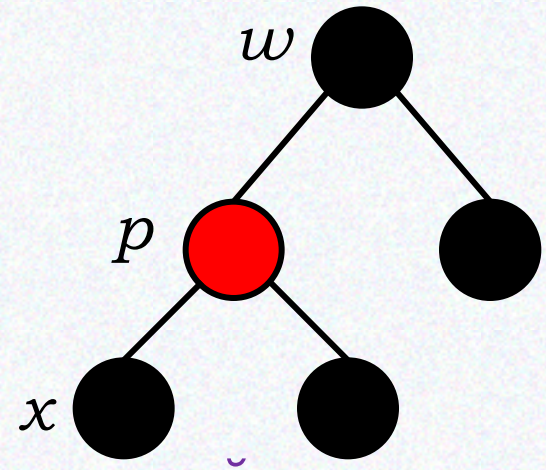
1. Обменять цвета w и p
2. Выполнить левый поворот вокруг узла p

Случай 1

Узел w – красный

Потомки w – черные

Узел p – черный



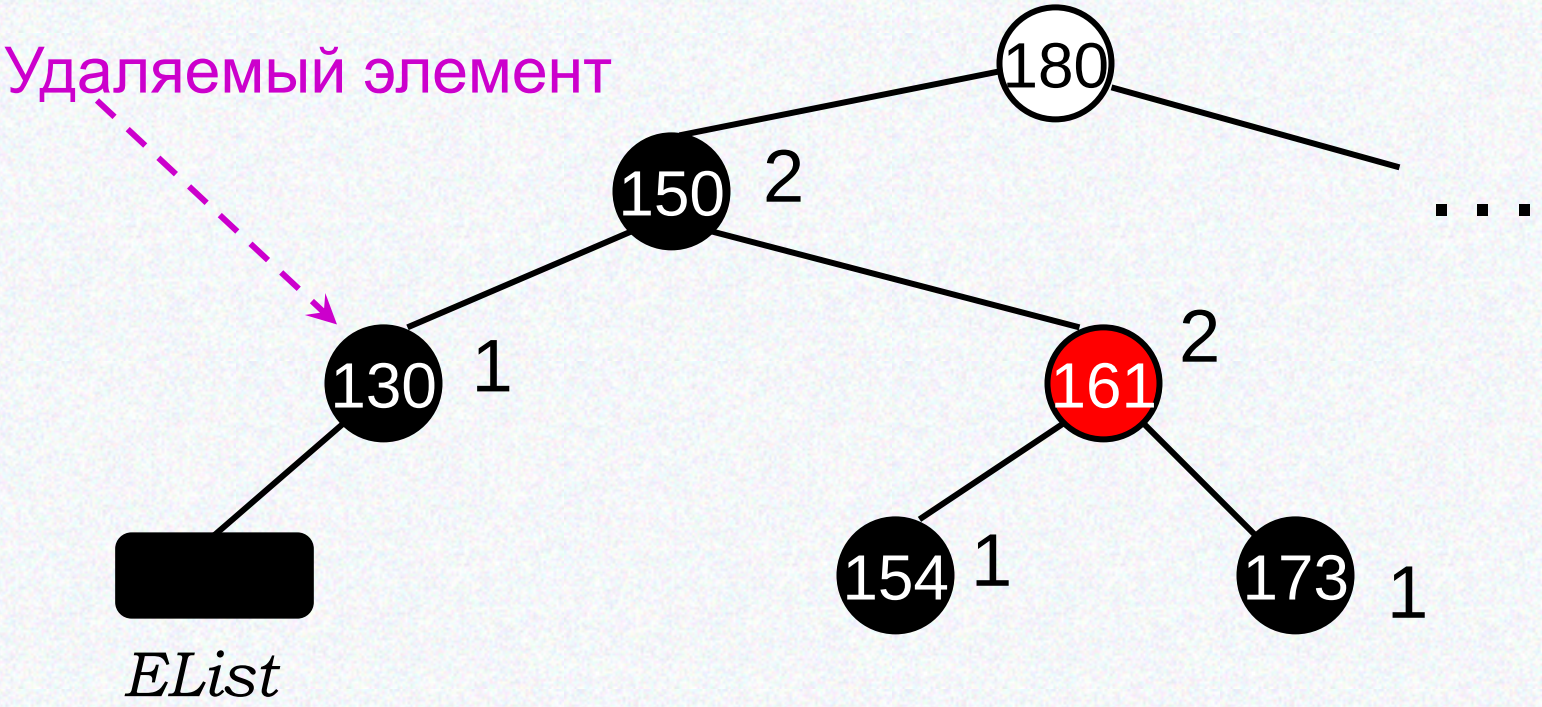
+ черный

Коррекция:

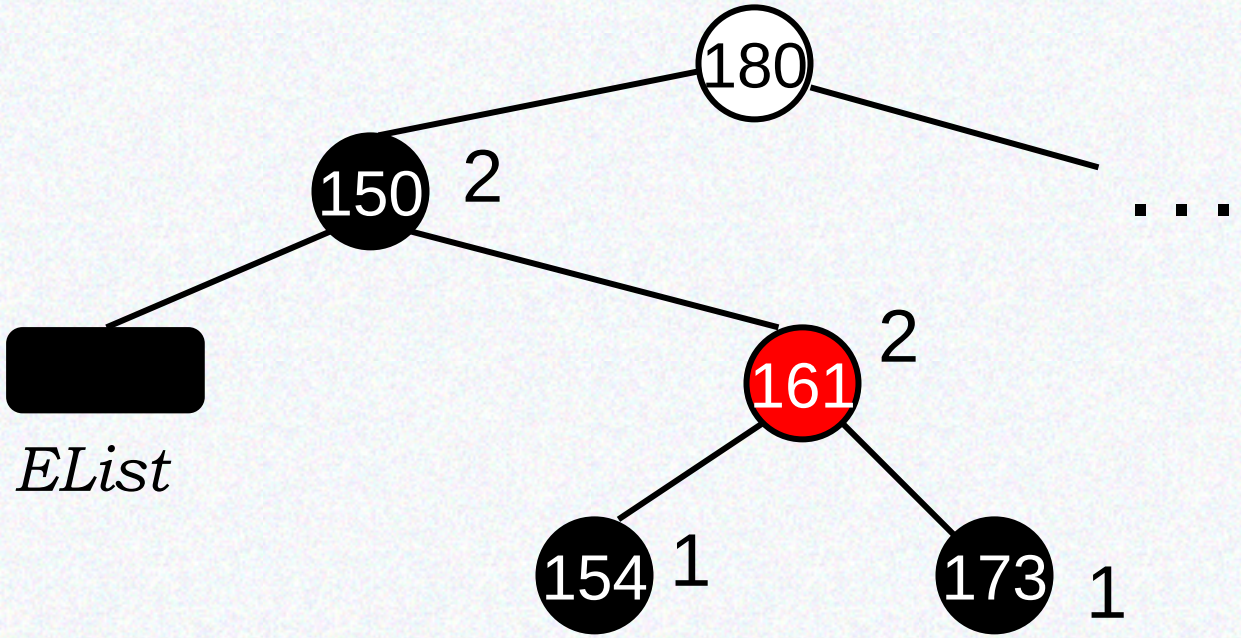
1. Обменять цвета w и p
2. Выполнить левый поворот вокруг узла p

Получим случаи 2, 3 или 4

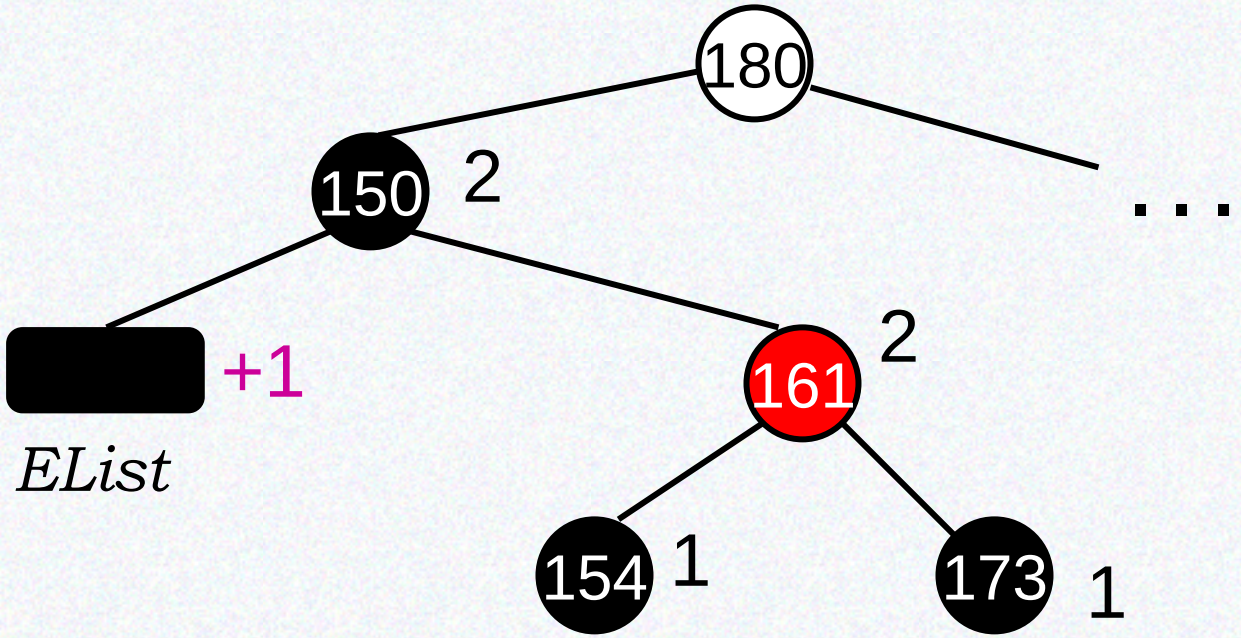
Случай 1



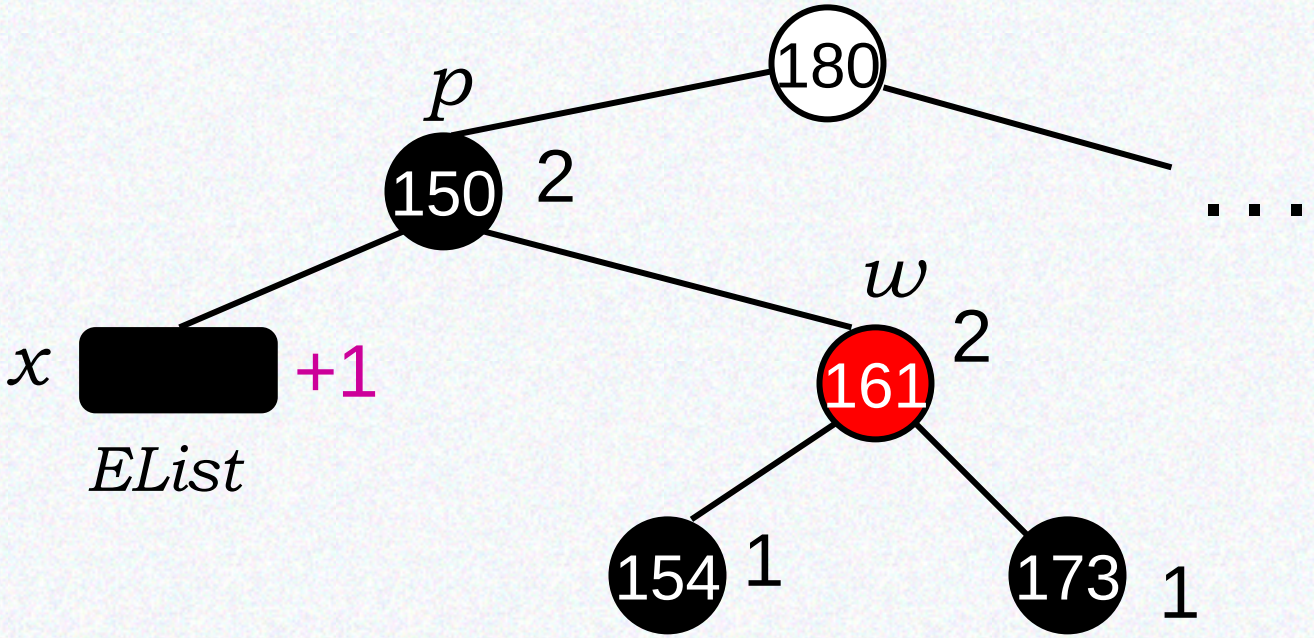
Случай 1



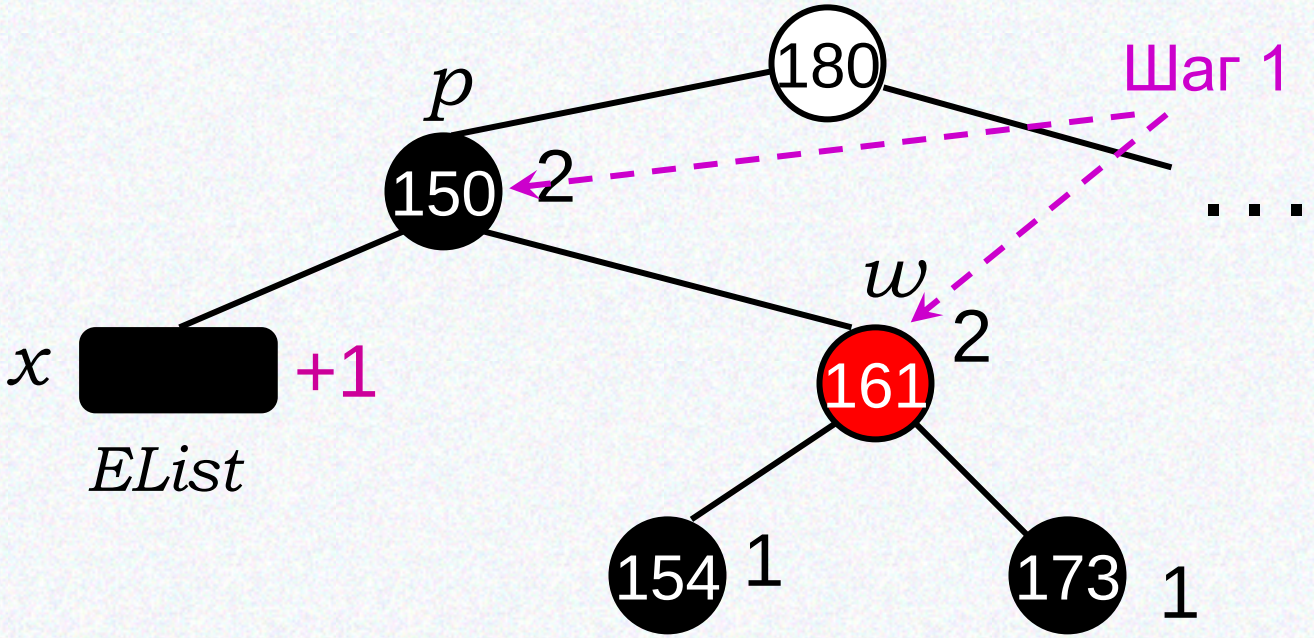
Случай 1



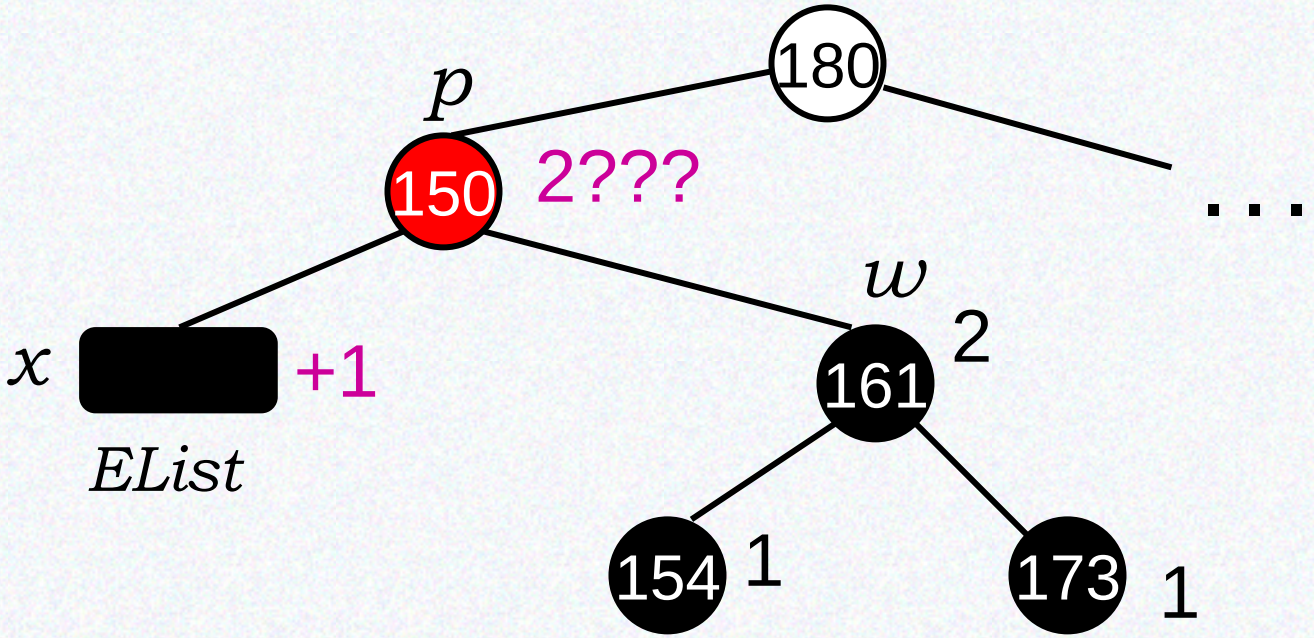
Случай 1



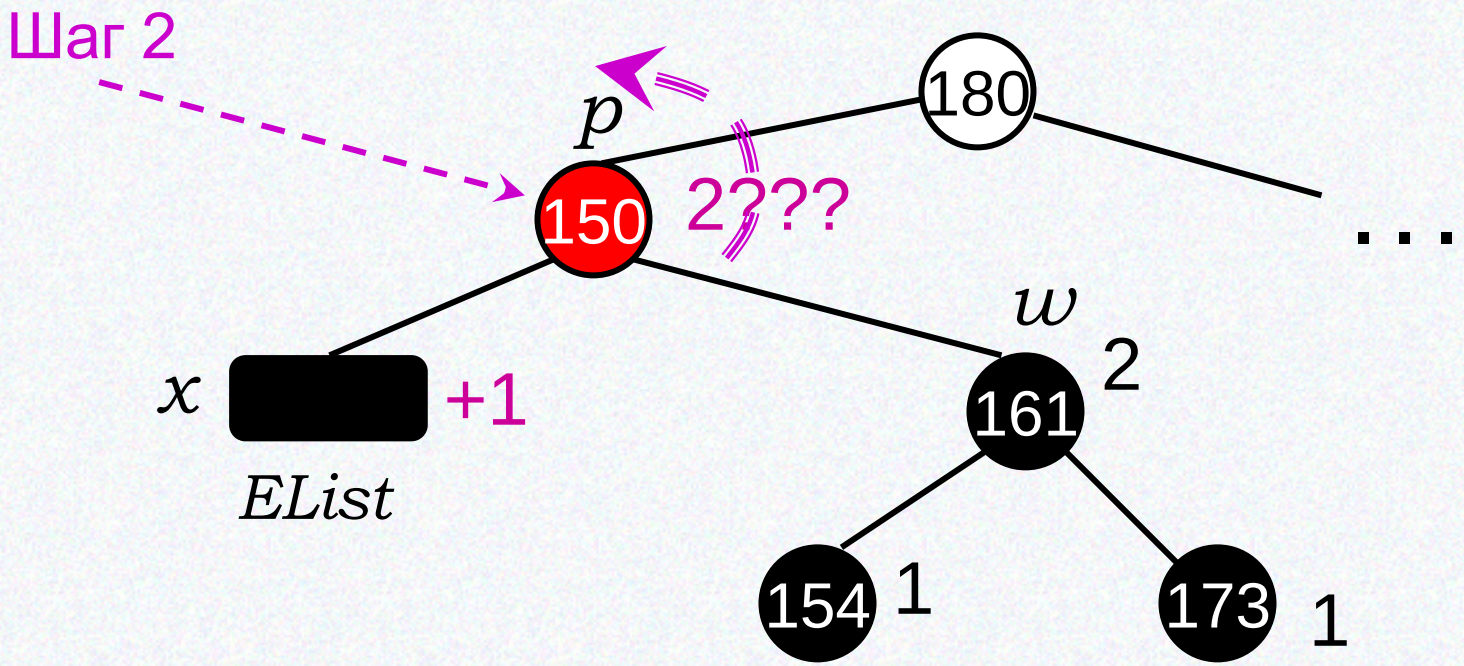
Случай 1



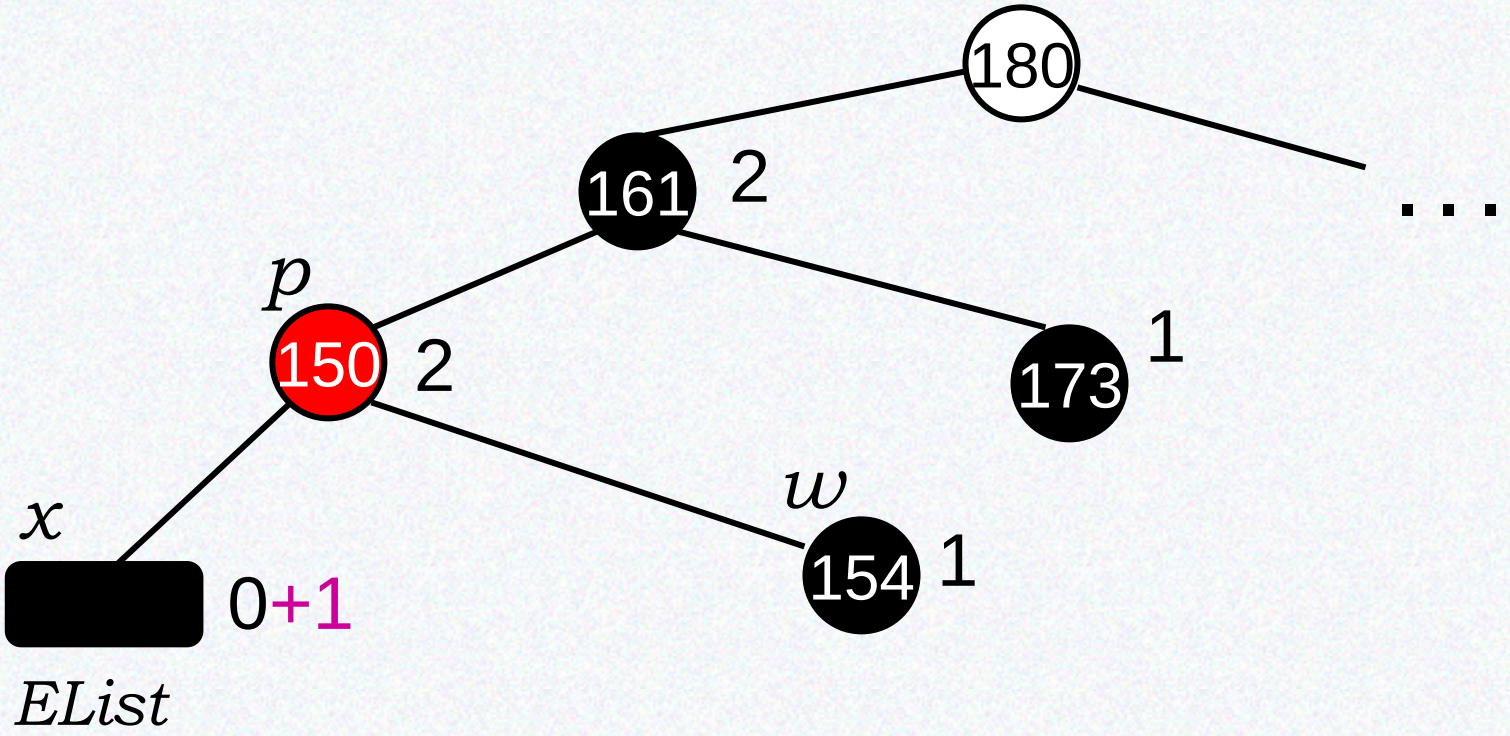
Случай 1



Случай 1



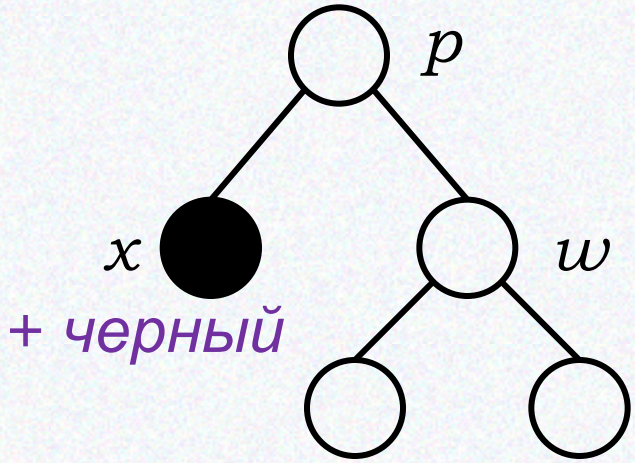
Случай 1



Случай 2

Узел w черный

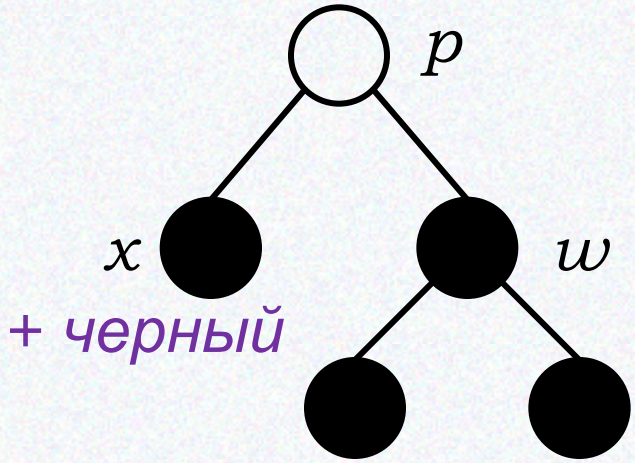
оба его потомка – черные



Случай 2

Узел w черный

оба его потомка – черные



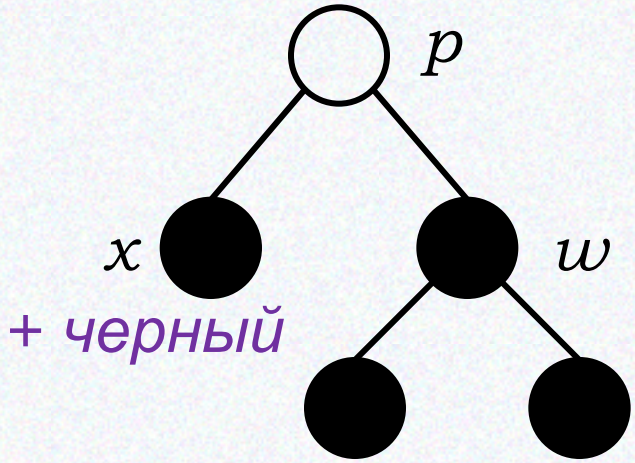
Случай 2

Узел w черный

оба его потомка – черные

Коррекция:

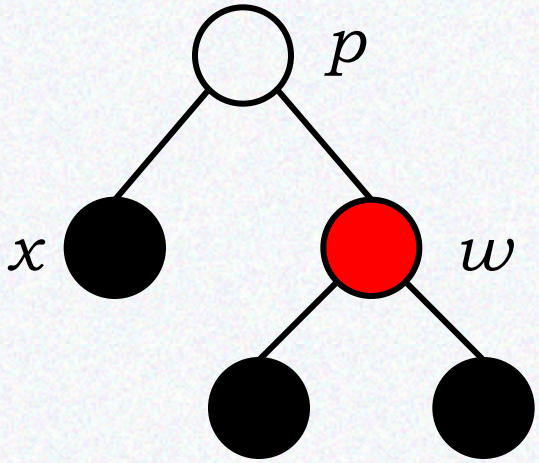
Шаг 1 – забрать черную окраску у x (станет единожды черным) и w (станет красным)



Случай 2

Узел w черный

оба его потомка – черные



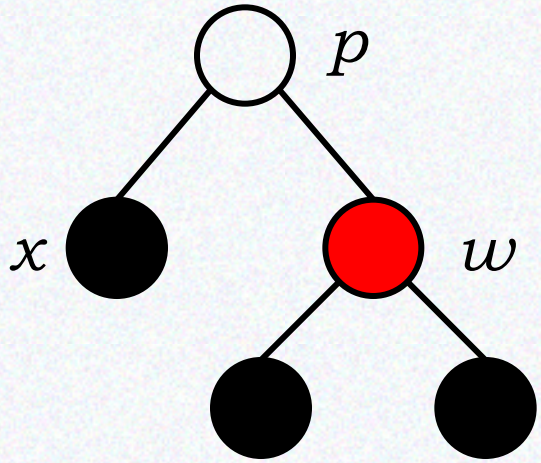
Коррекция:

Шаг 1 – забрать черную окраску у x (станет единожды черным) и w (станет красным)

Случай 2

Узел w черный

оба его потомка – черные



Коррекция:

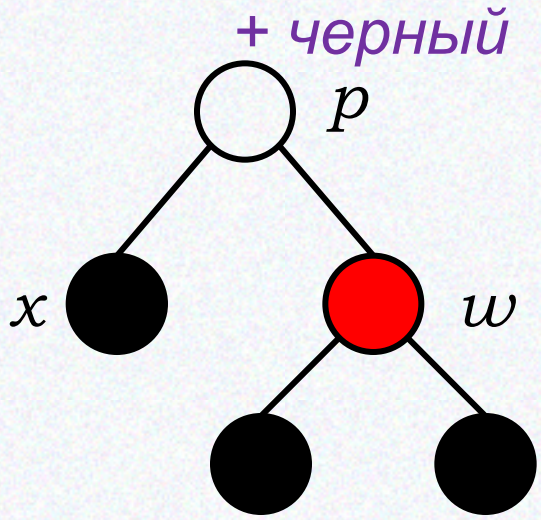
Шаг 1 – забрать черную окраску у x (станет единожды черным) и w (станет красным)

Шаг 2 – добавить дополнительный черный цвет узлу p

Случай 2

Узел w черный

оба его потомка – черные



Коррекция:

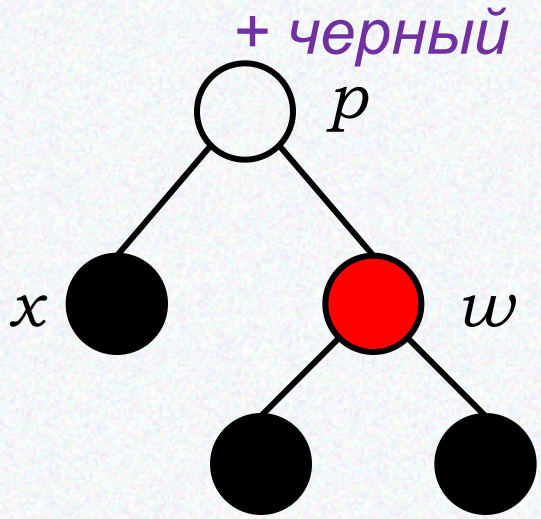
Шаг 1 – забрать черную окраску у x (станет единожды черным) и w (станет красным)

Шаг 2 – добавить дополнительный черный цвет узлу p

Случай 2

Узел w черный

оба его потомка – черные



Коррекция:

Шаг 1 – забрать черную окраску у x (станет единожды черным) и w (станет красным)

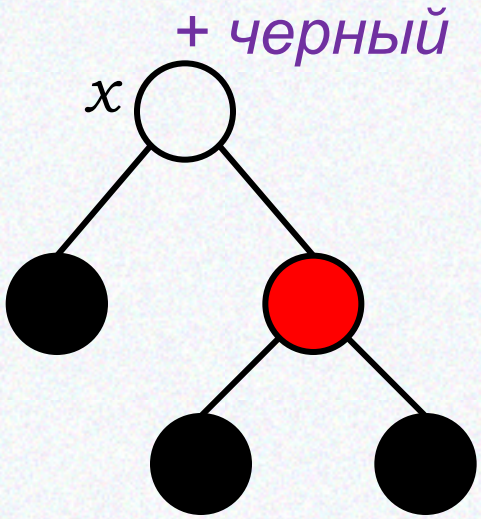
Шаг 2 – добавить дополнительный черный цвет узлу p

Шаг 3 – переустановить узел x, продолжить коррекцию дерева

Случай 2

Узел w черный

оба его потомка – черные



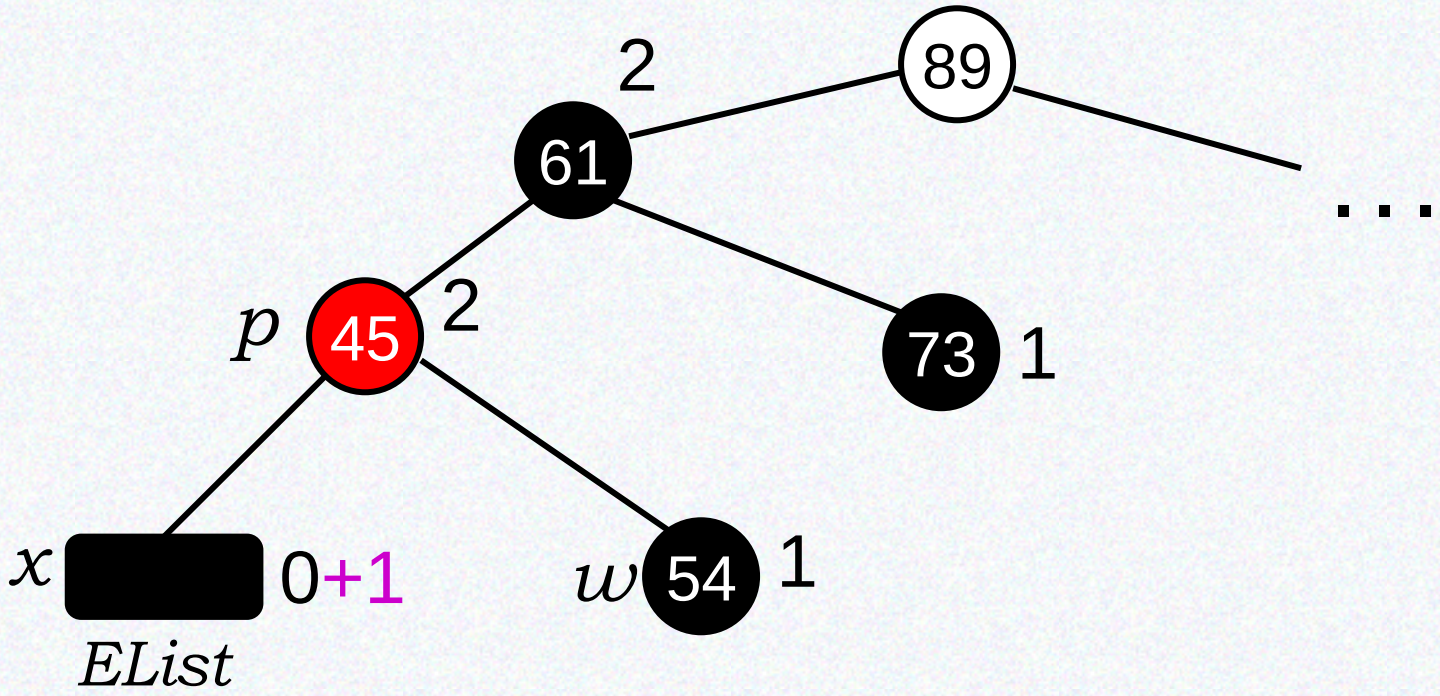
Коррекция:

Шаг 1 – забрать черную окраску у x (станет единожды черным) и w (станет красным)

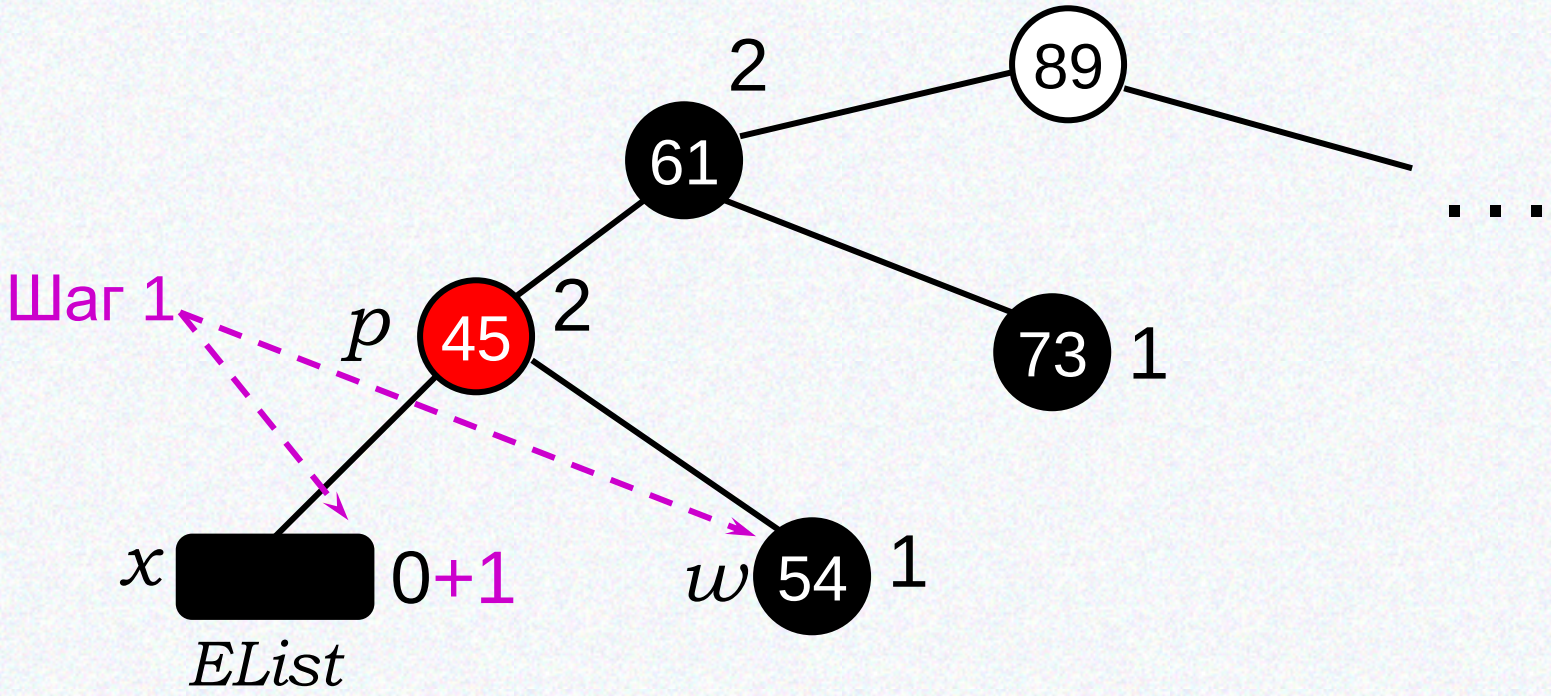
Шаг 2 – добавить дополнительный черный цвет узлу p

Шаг 3 – переустановить узел x, продолжить коррекцию дерева

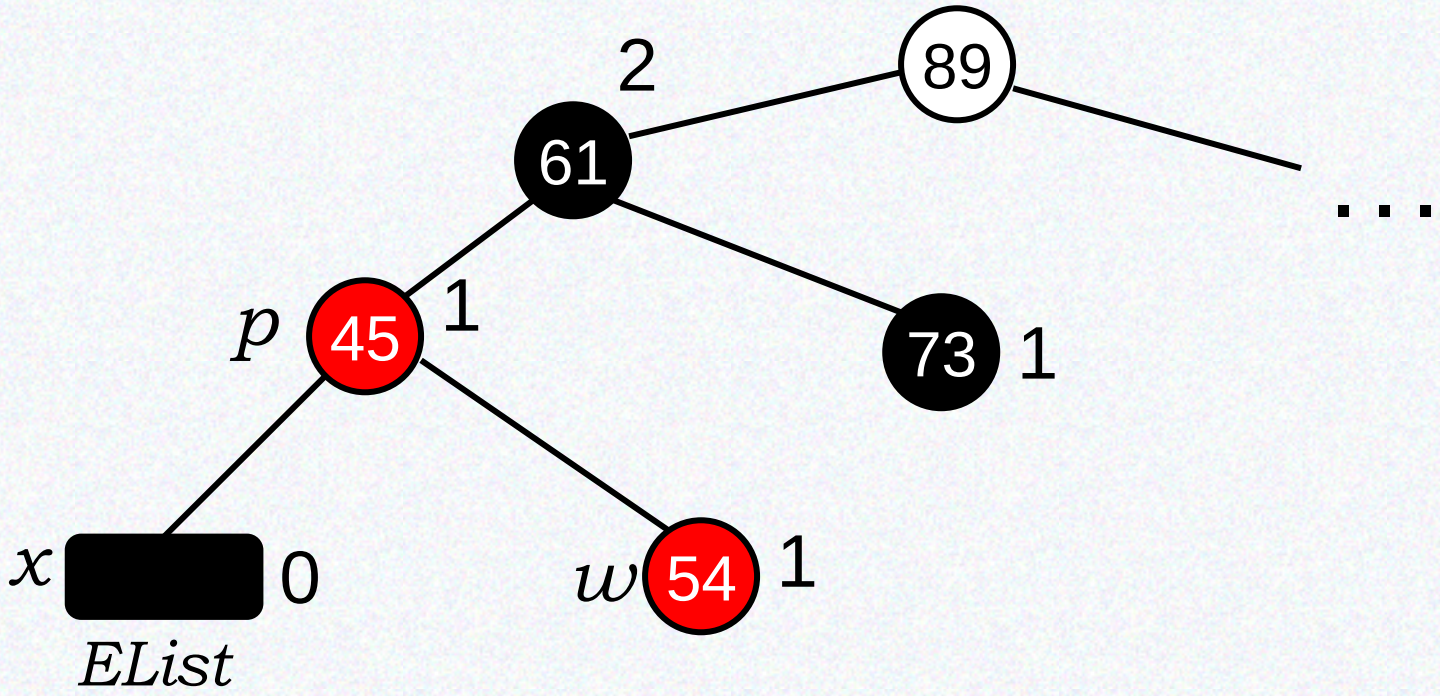
Случай 2



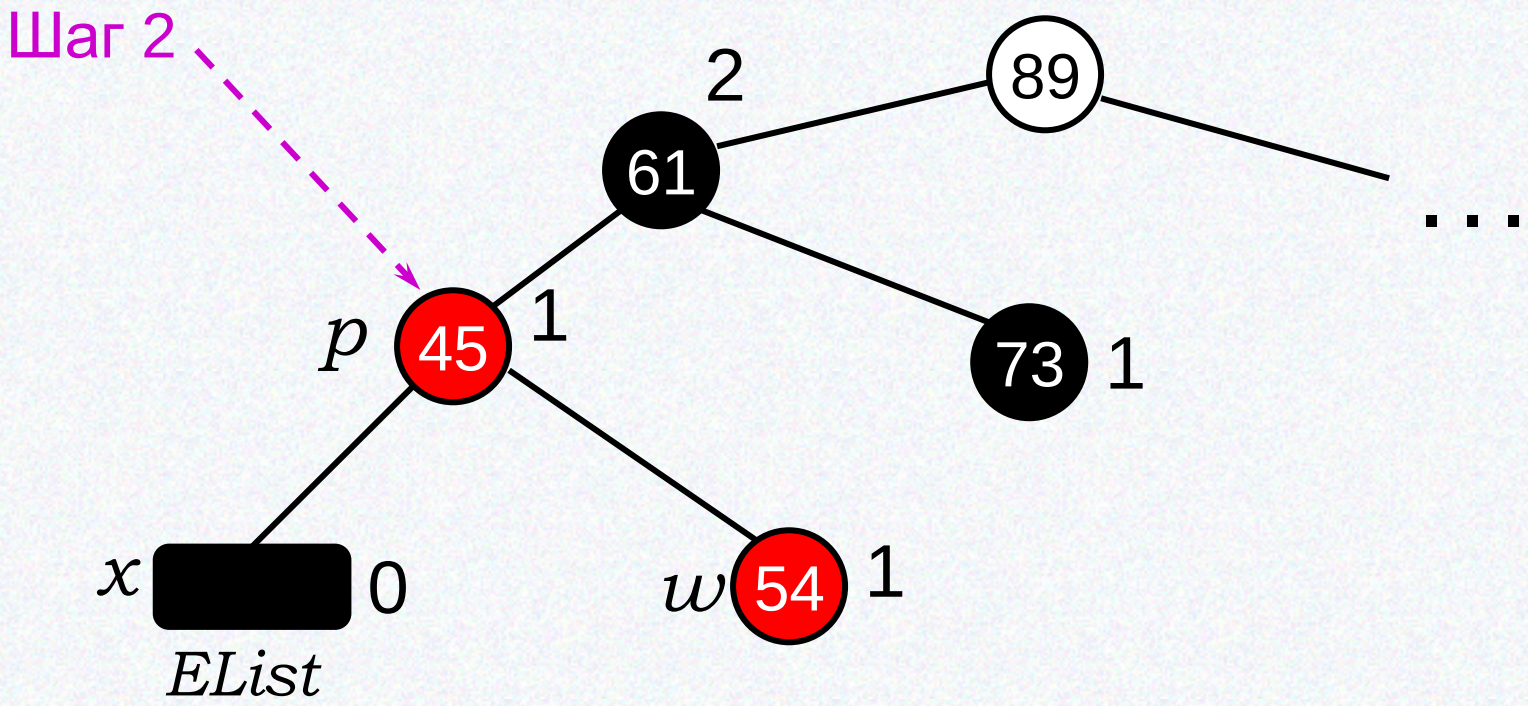
Случай 2



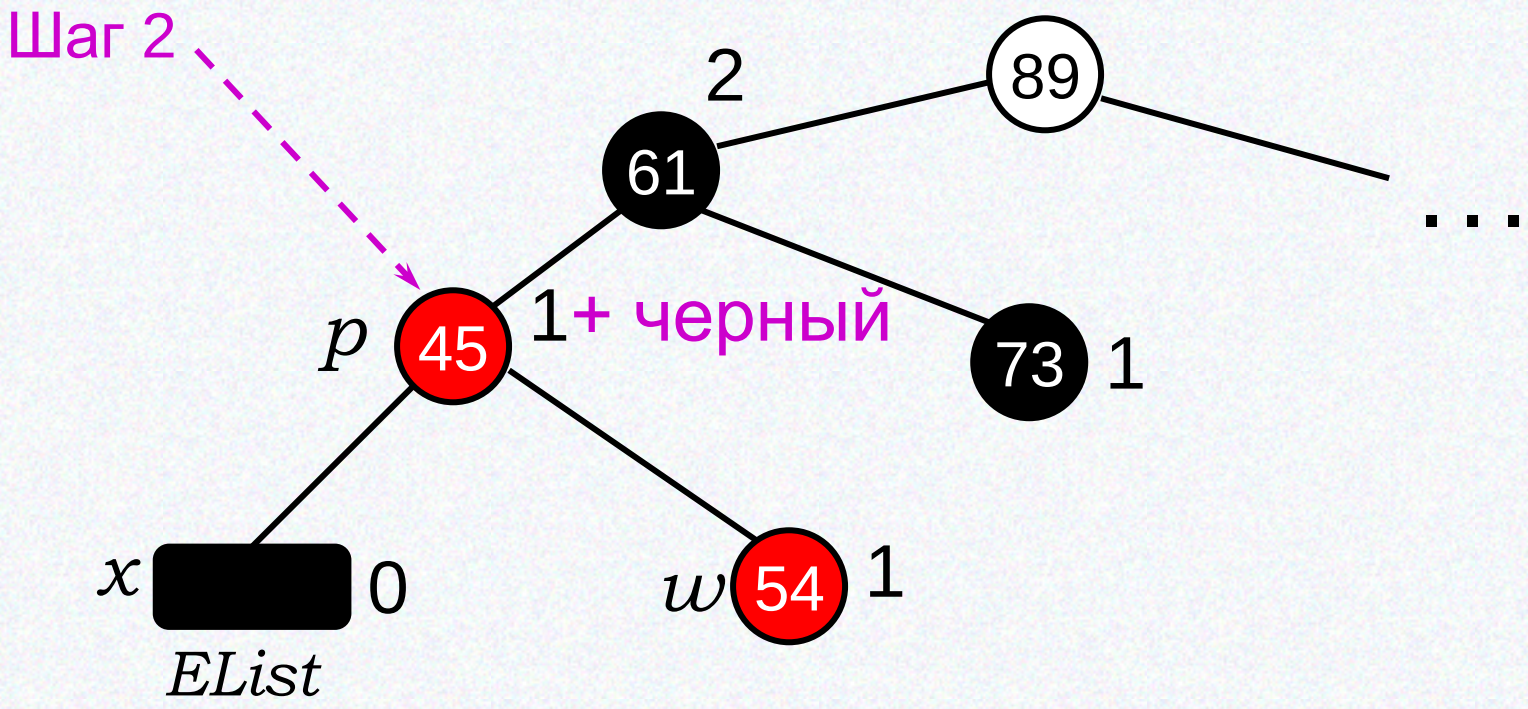
Случай 2



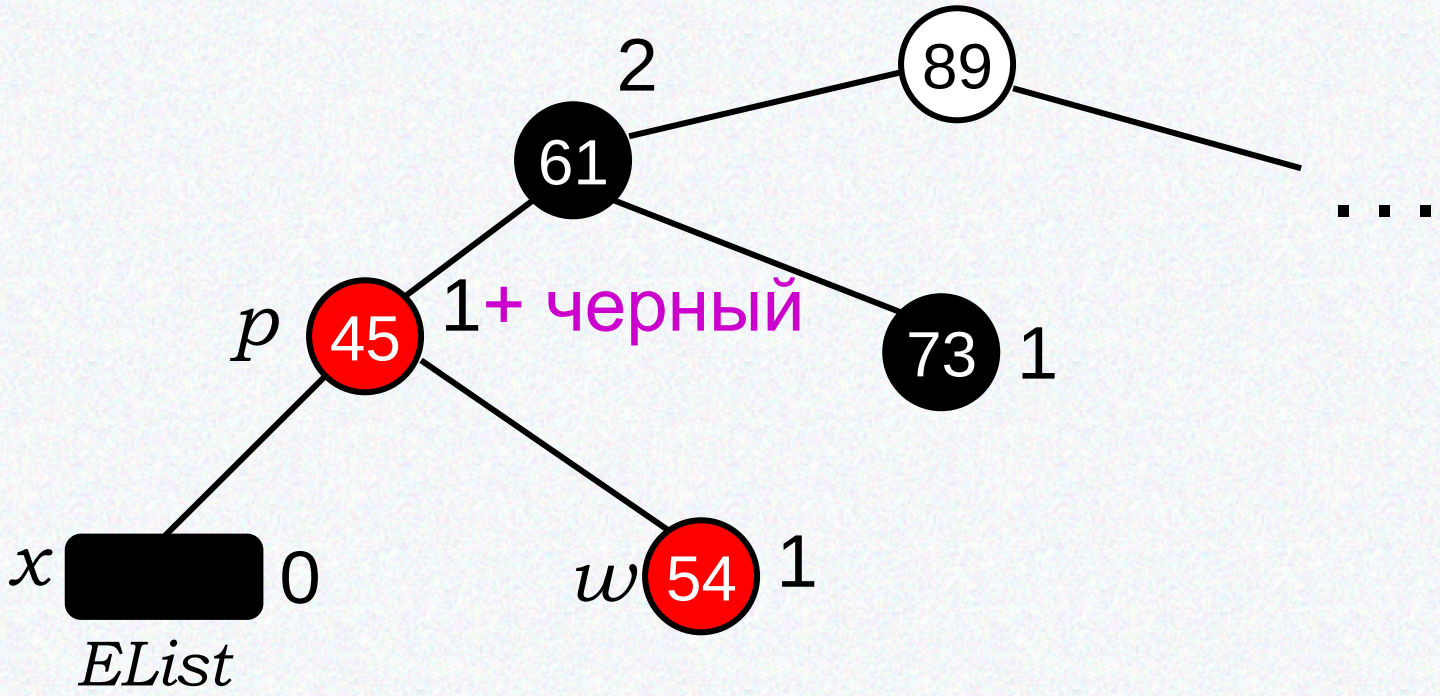
Случай 2



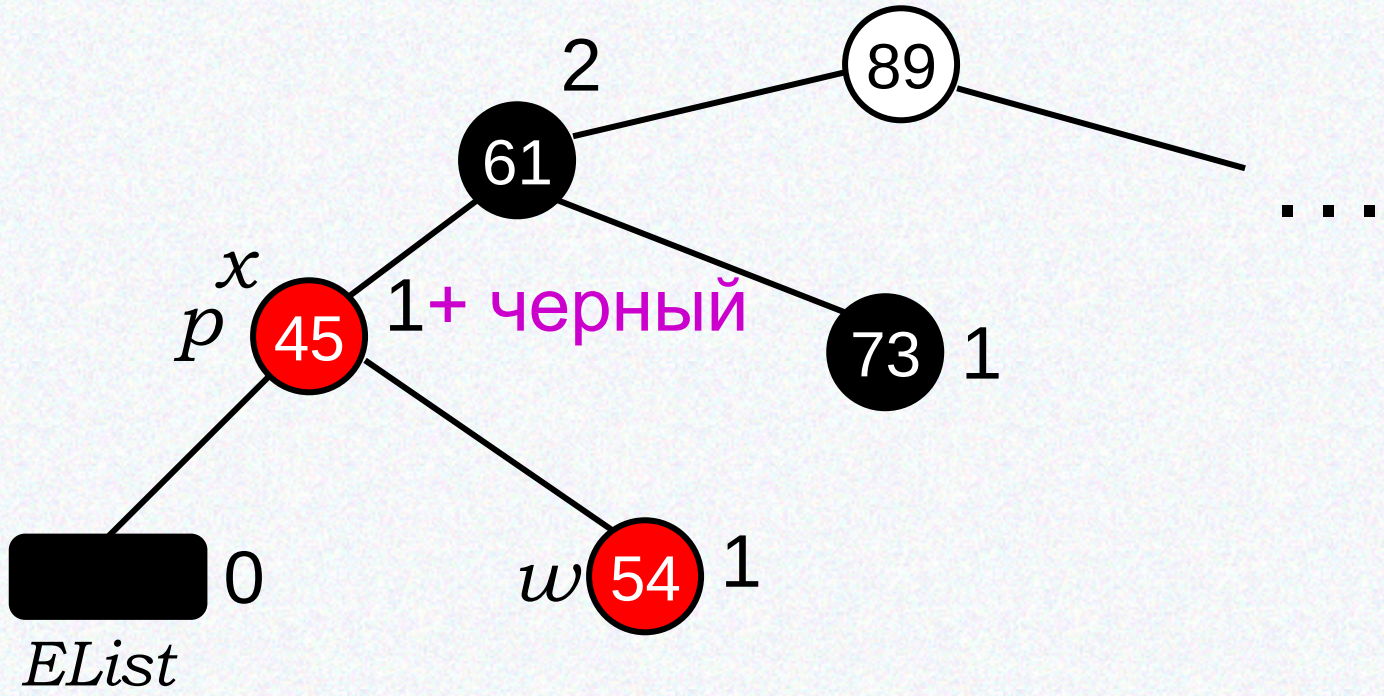
Случай 2



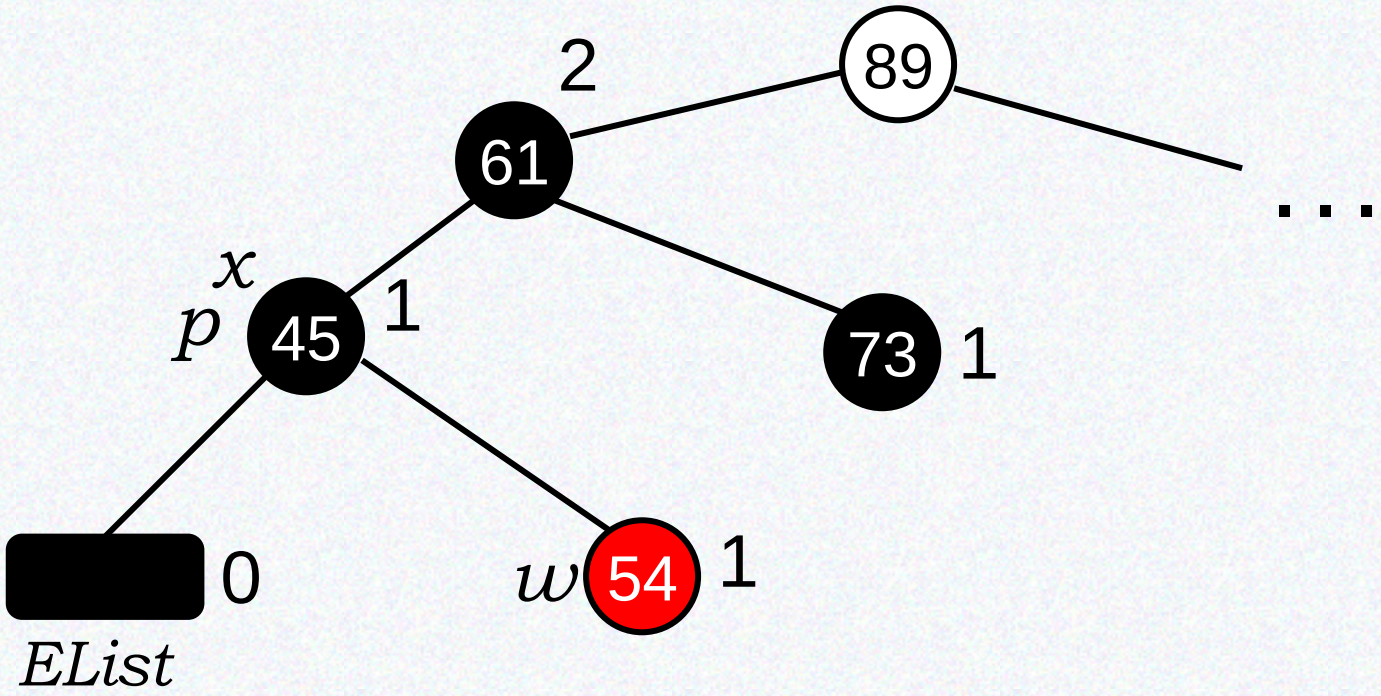
Случай 2



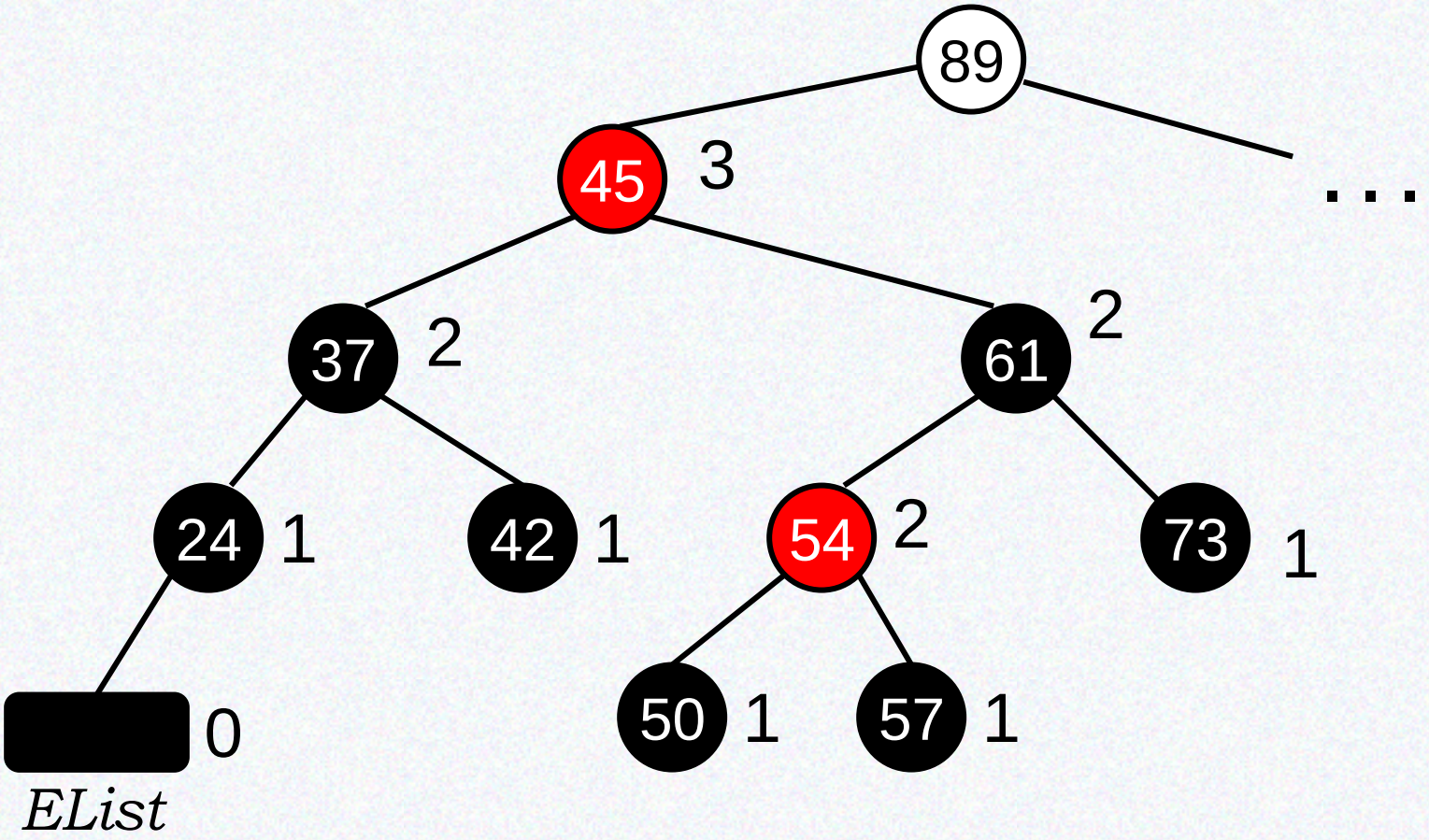
Случай 2



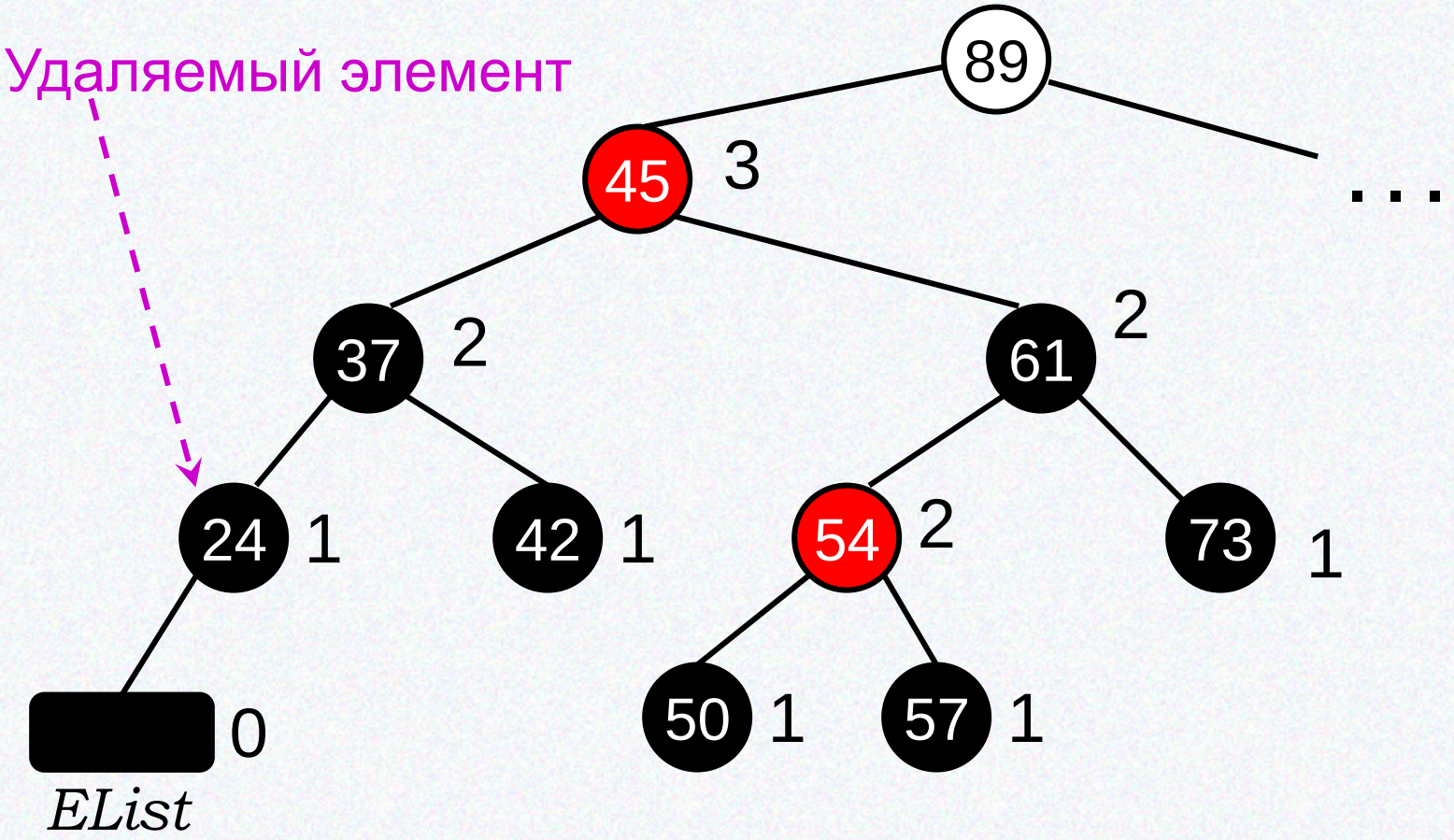
Случай 2



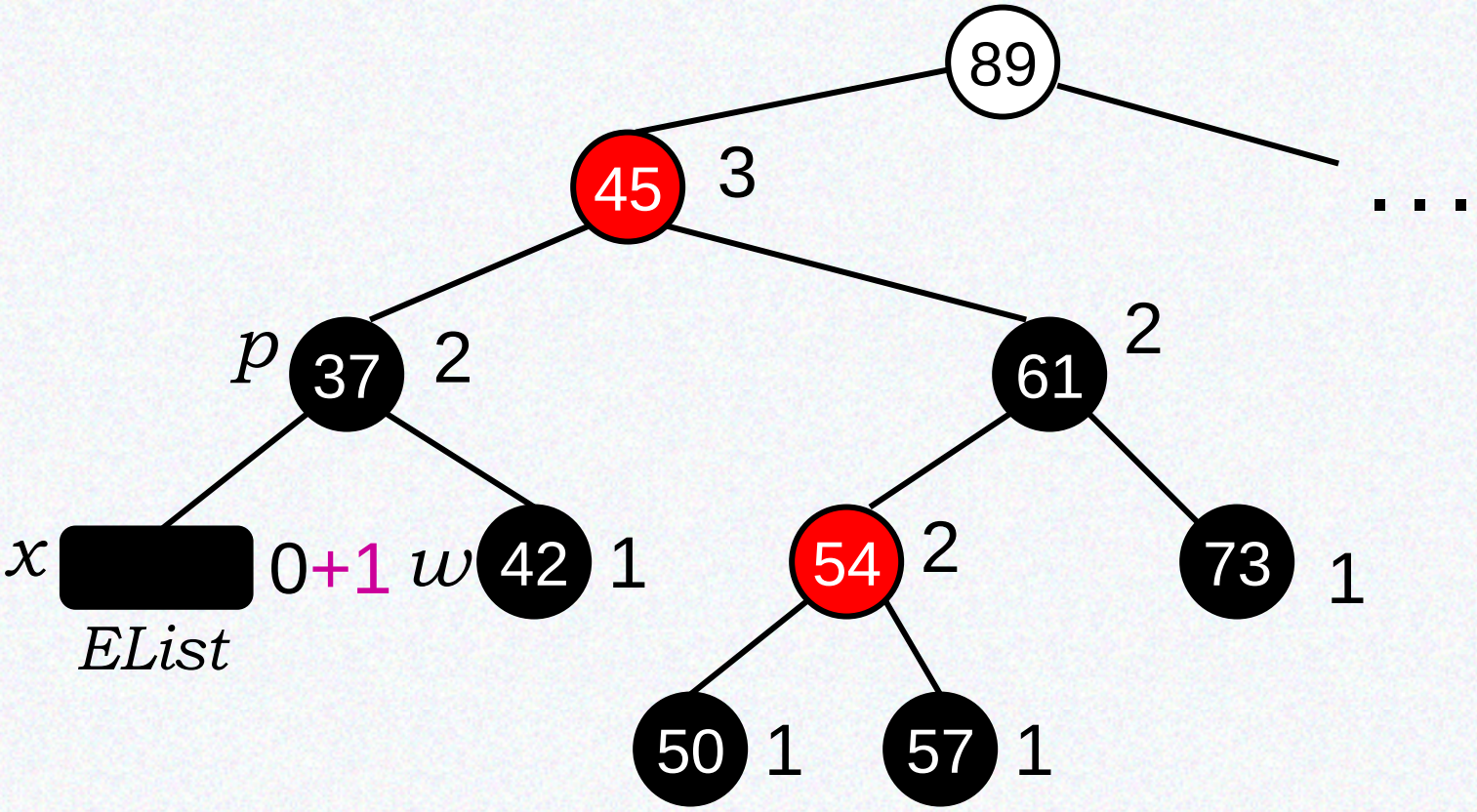
Случай 2



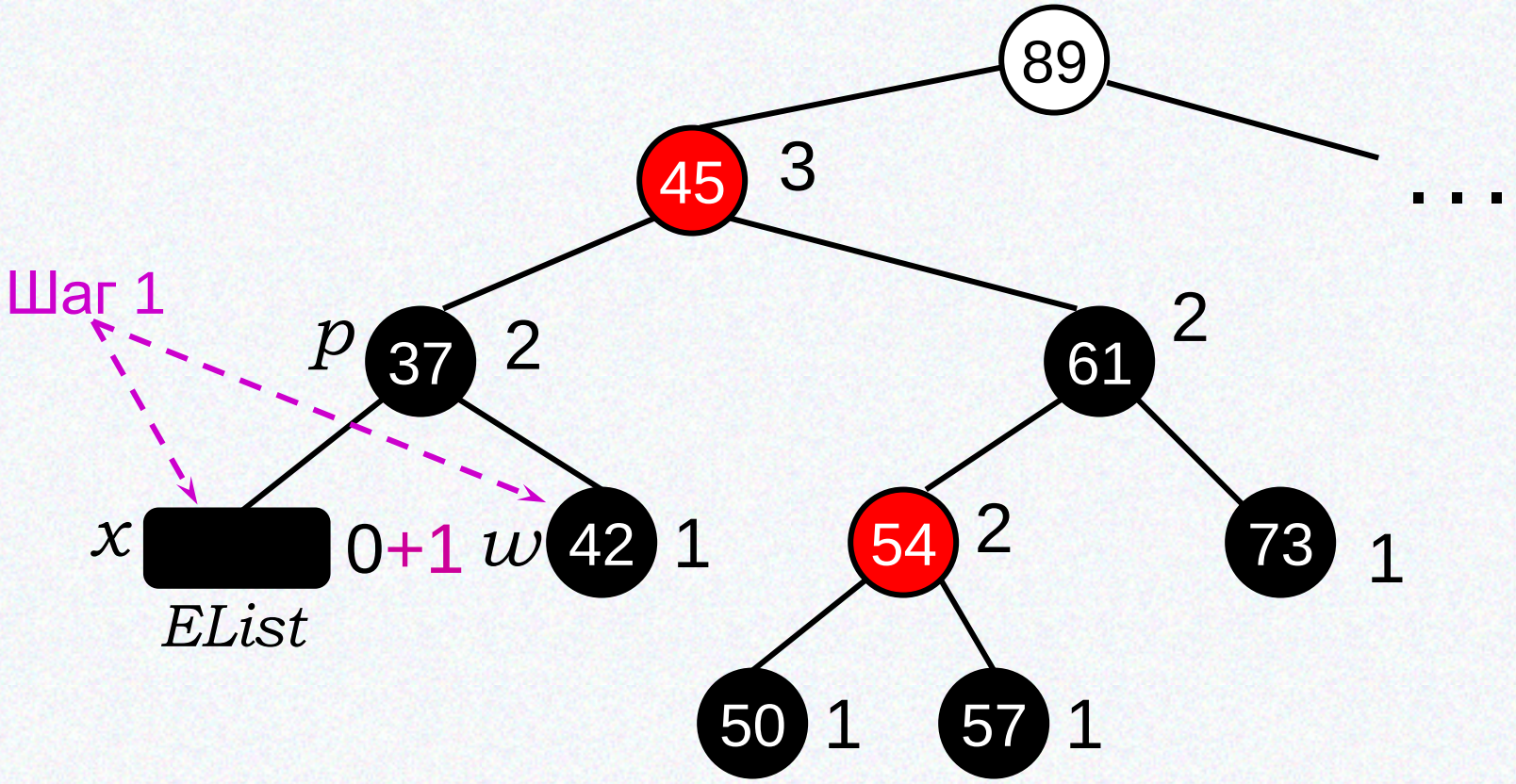
Случай 2


Удаляемый элемент
89
45
3
. . .
37
2
61
2
24
1
42
1
54
2
73
1
0
50
1
57
1
EList

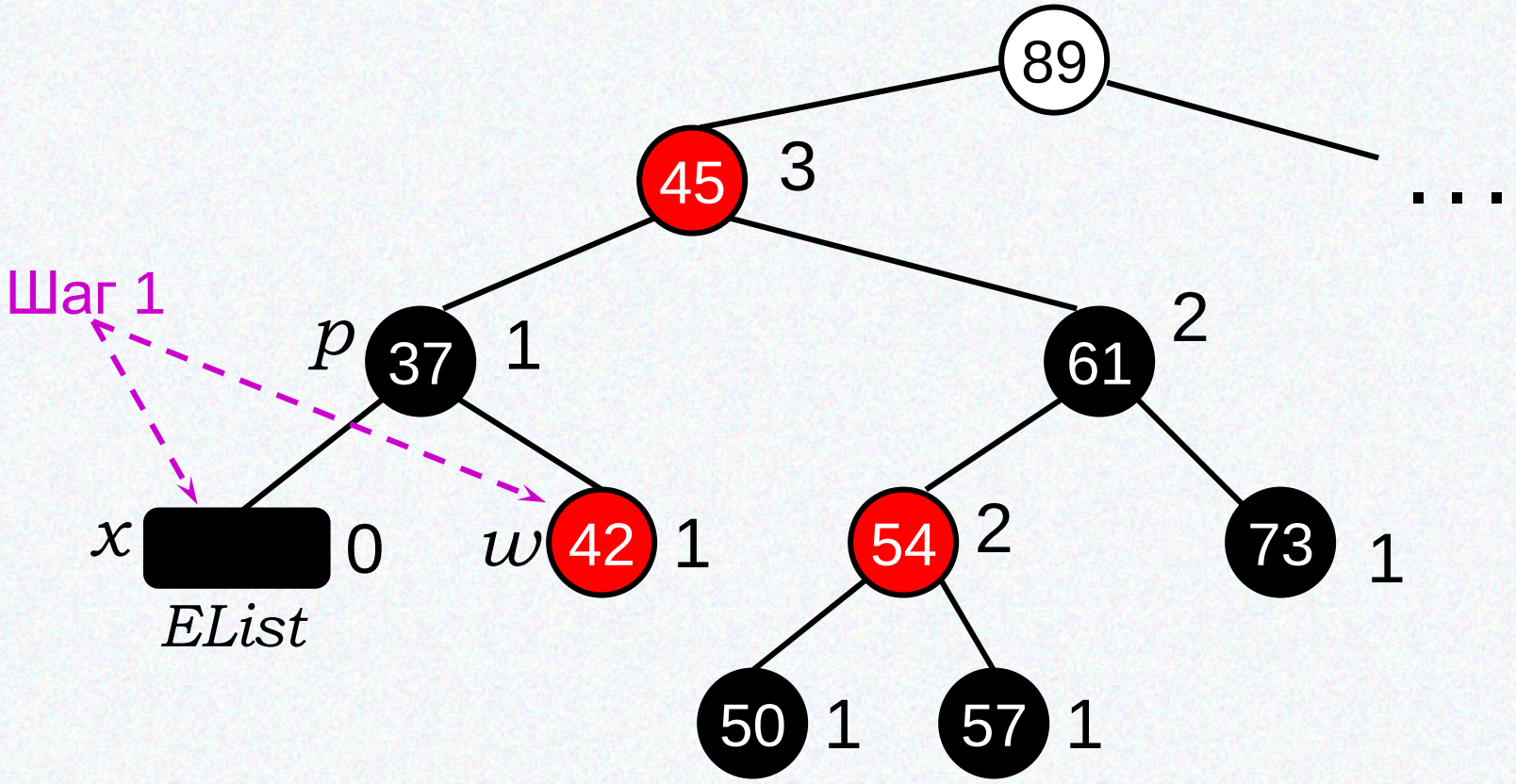
Случай 2



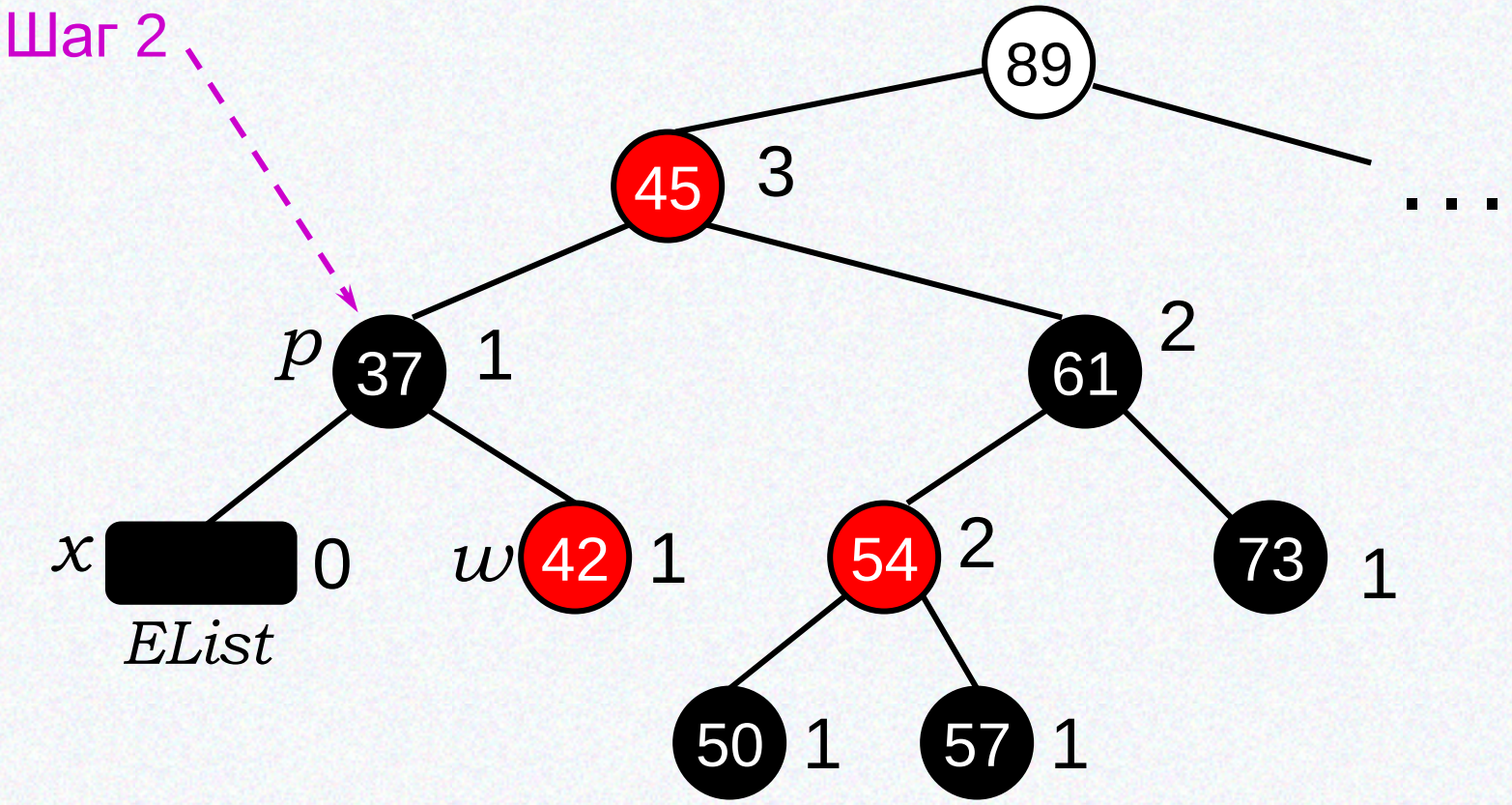
Случай 2



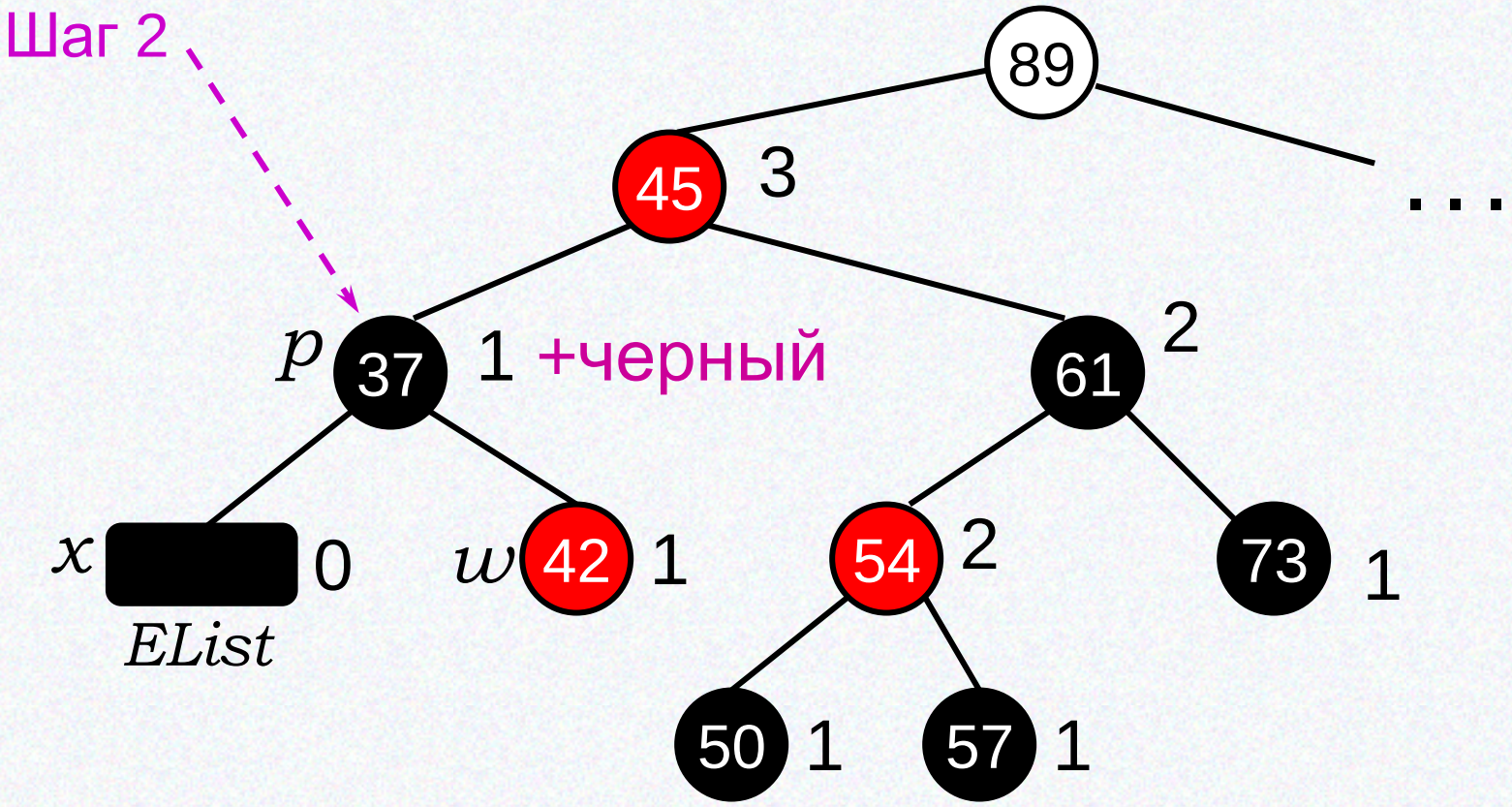
Случай 2



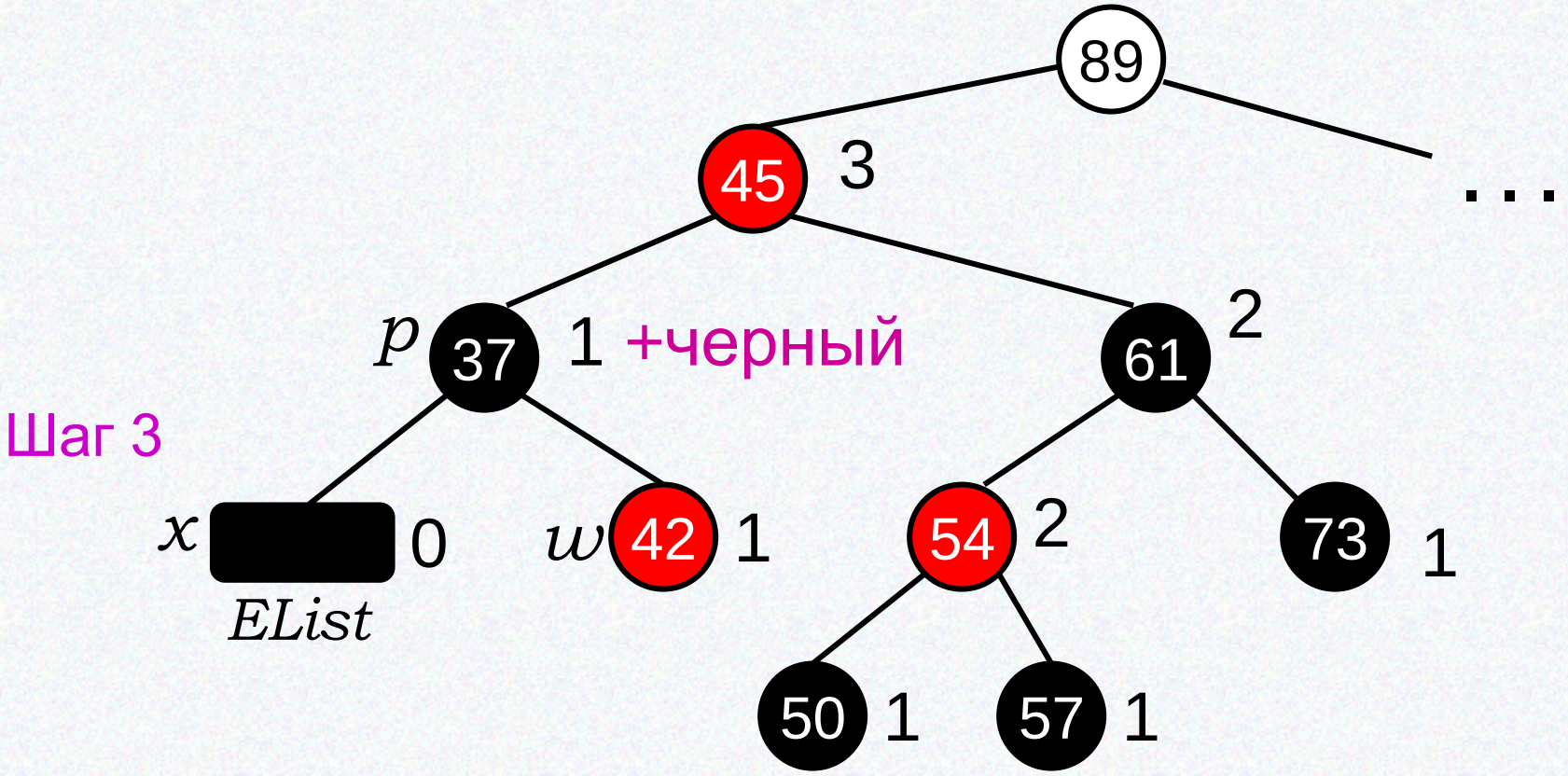
Случай 2



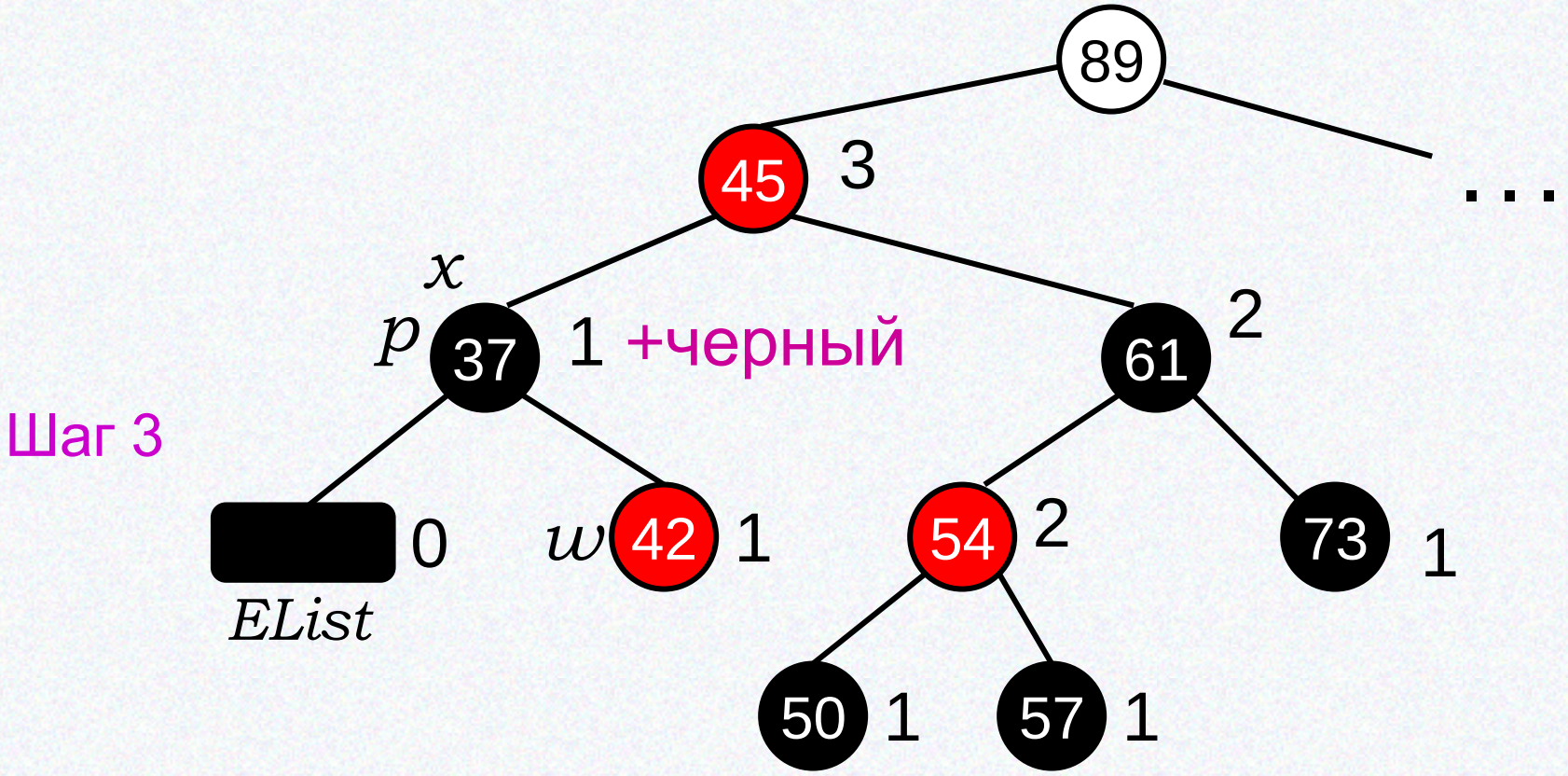
Случай 2



Случай 2



Случай 2

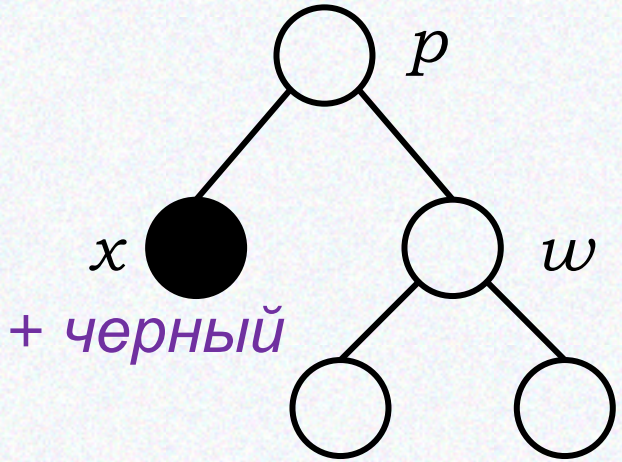


Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный

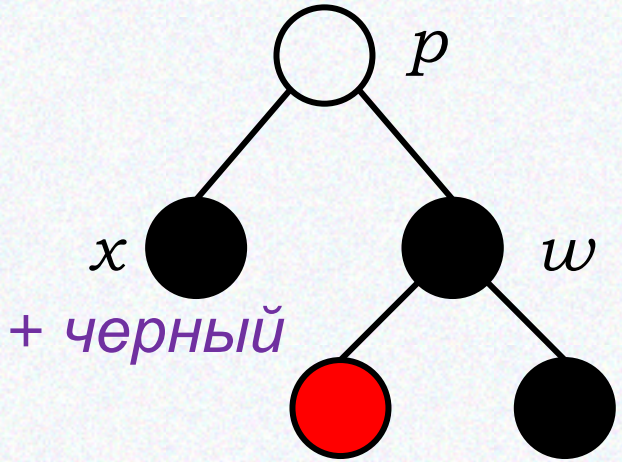


Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный

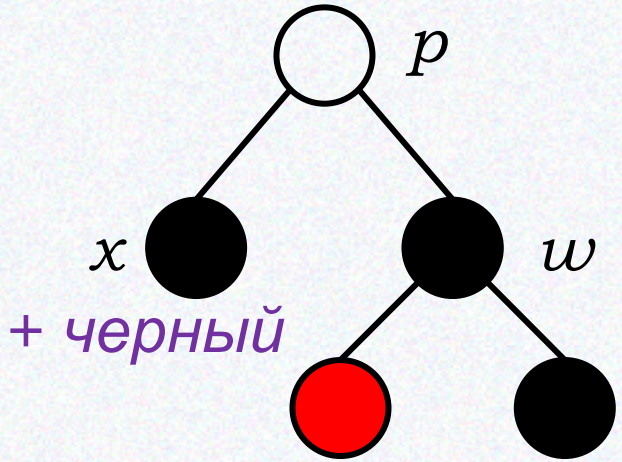


Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный



Коррекция:

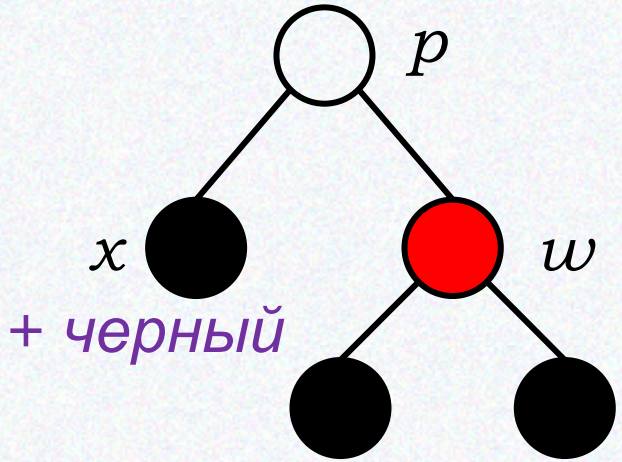
Шаг 1 – обменять цвета w и w->left

Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный



Коррекция:

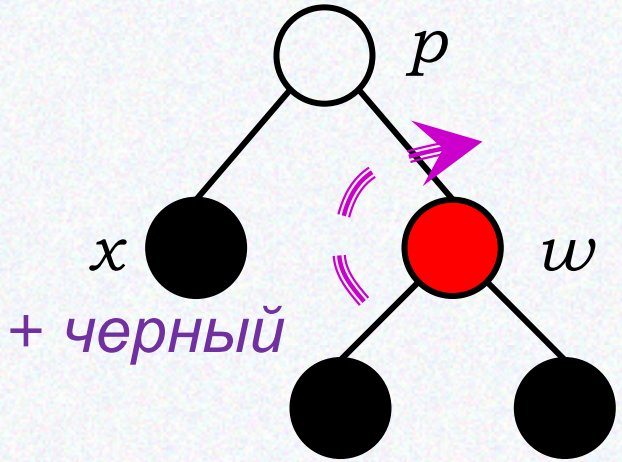
Шаг 1 – обменять цвета w и w->left

Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный



Коррекция:

Шаг 1 – обменять цвета w и w->left

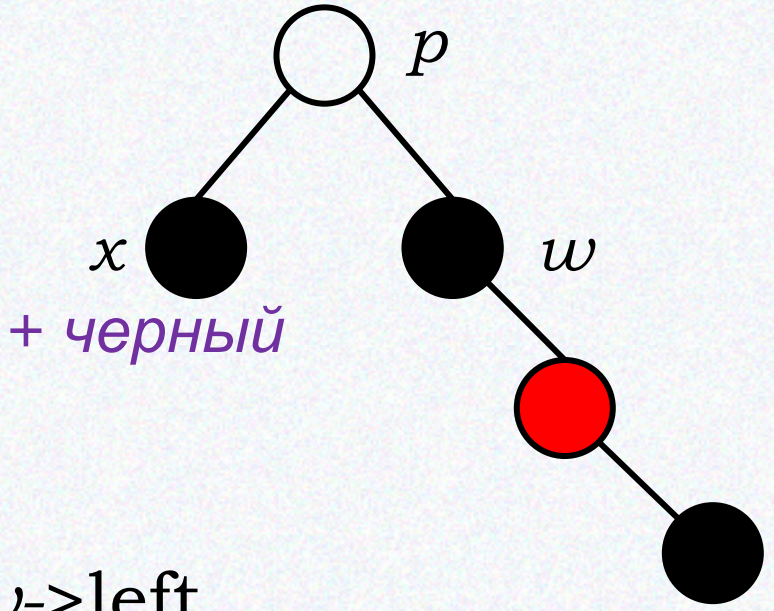
Шаг 2 – выполнить правый поворот вокруг w

Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный



Коррекция:

Шаг 1 – обменять цвета w и w->left

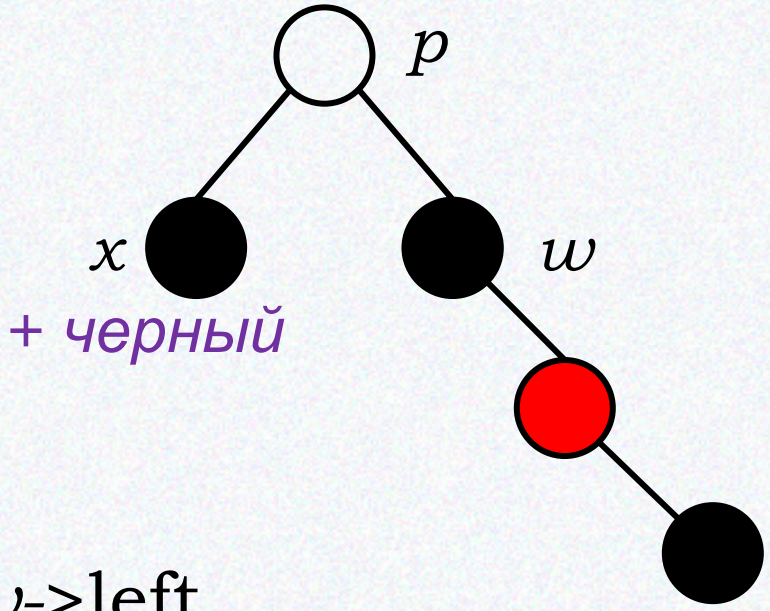
Шаг 2 – выполнить правый поворот вокруг w

Случай 3

Узел w – черный,

его правый дочерний узел – черный,

левый – красный



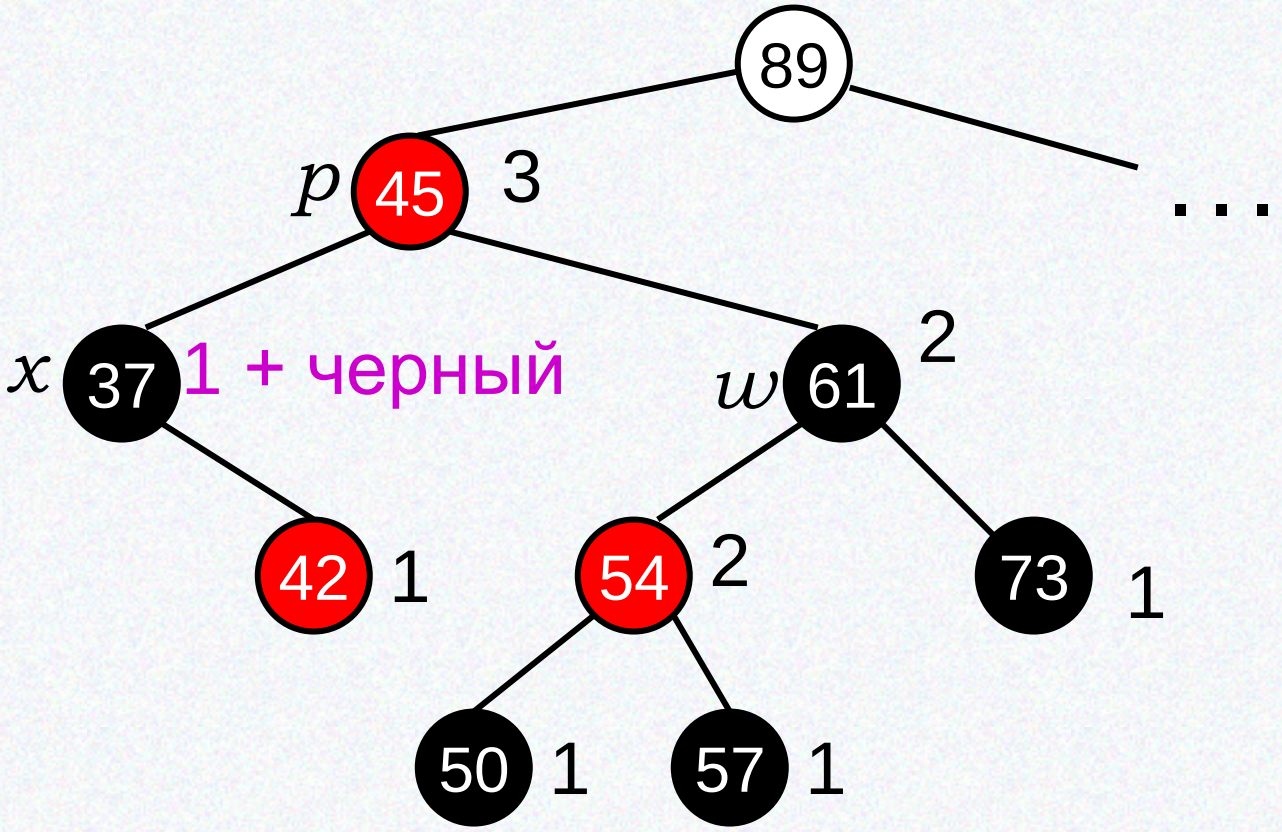
Коррекция:

Шаг 1 – обменять цвета w и w->left

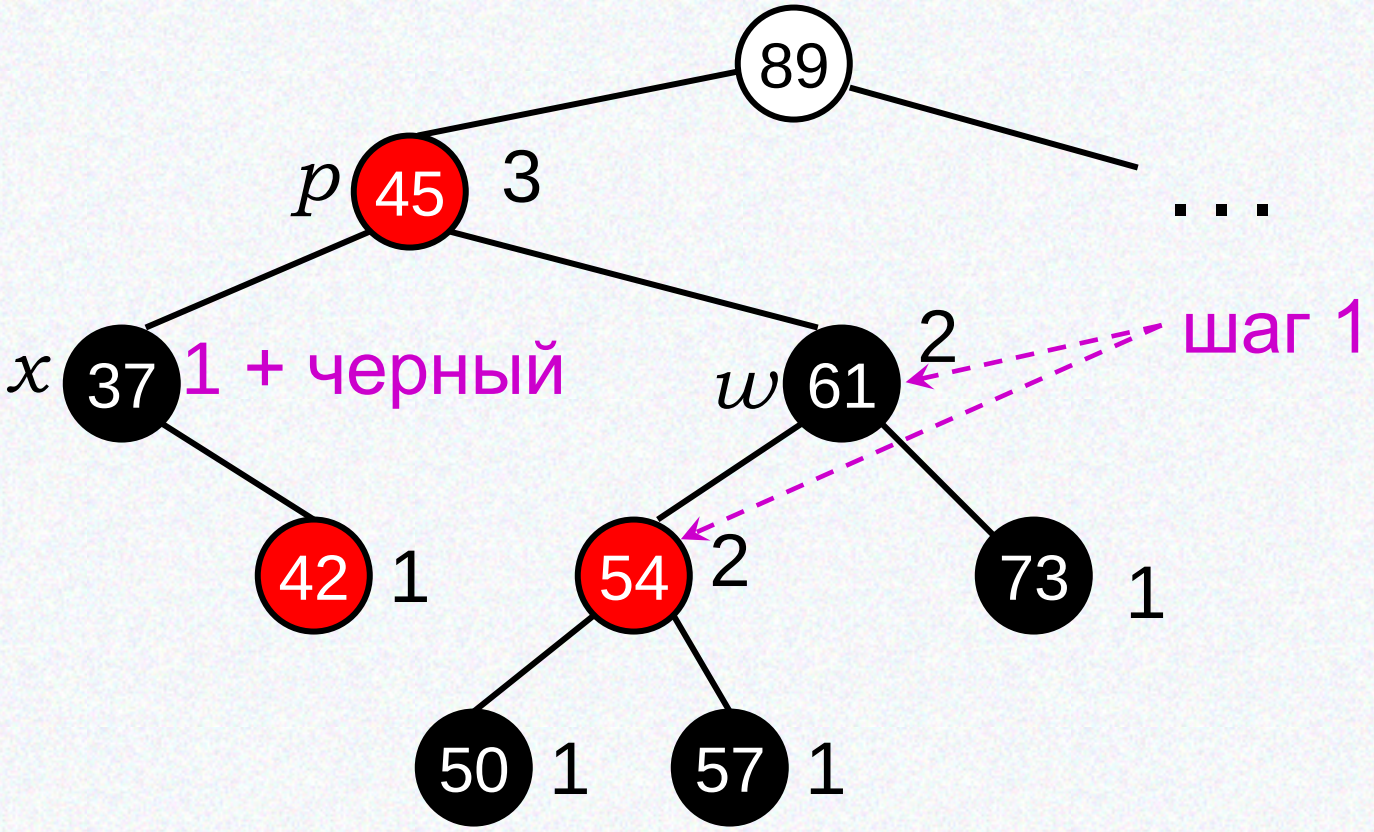
Шаг 2 – выполнить правый поворот вокруг w

Получим случай 4

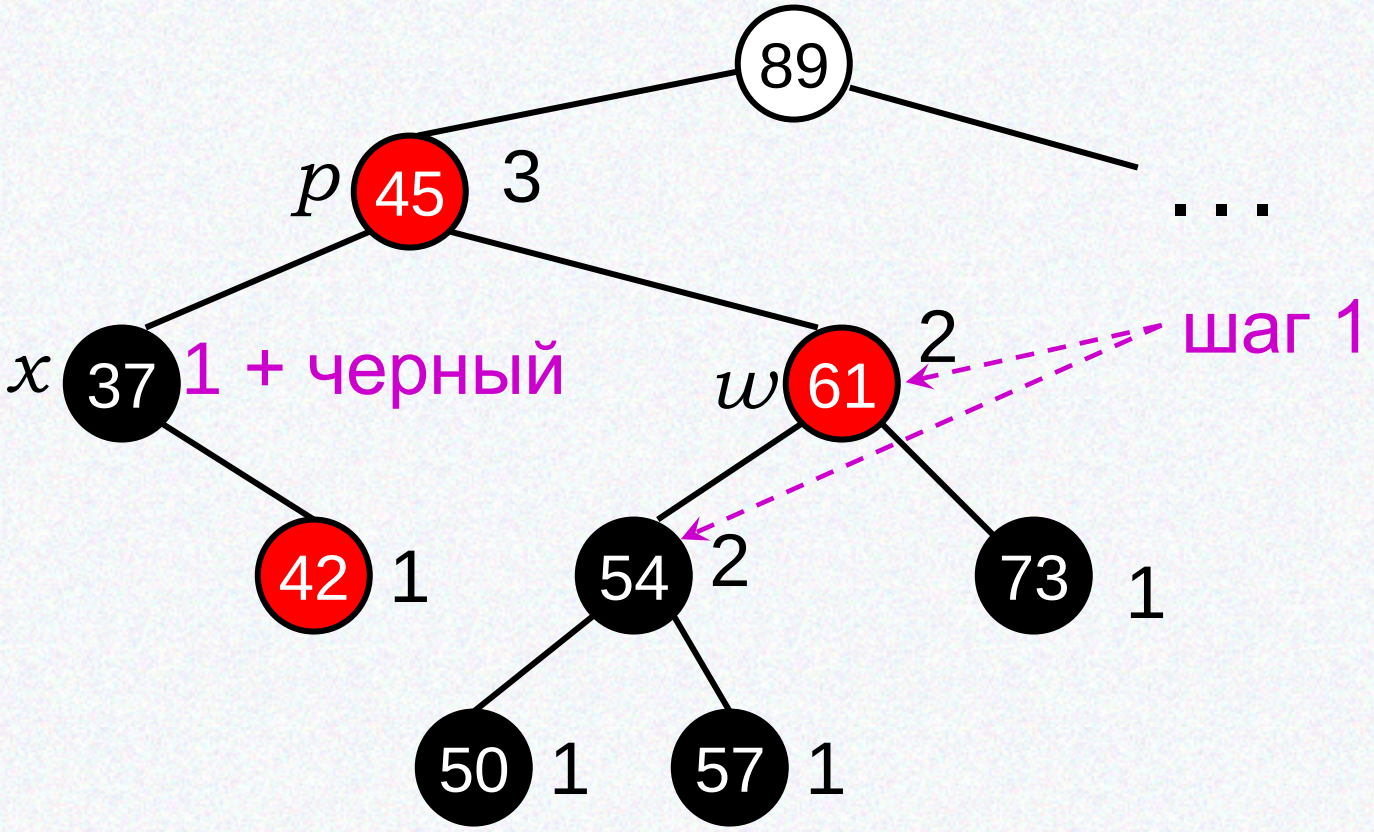
Случай 3



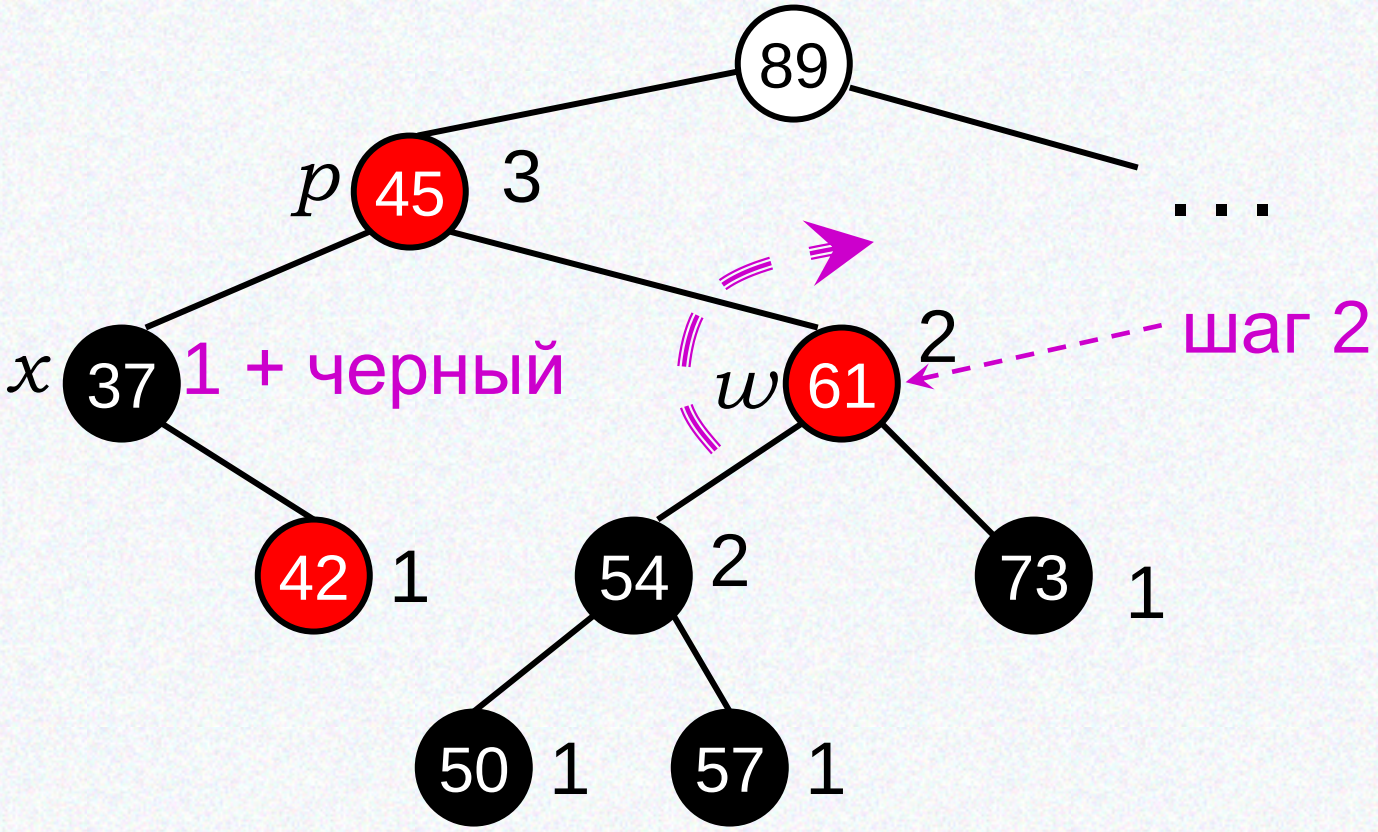
Случай 3



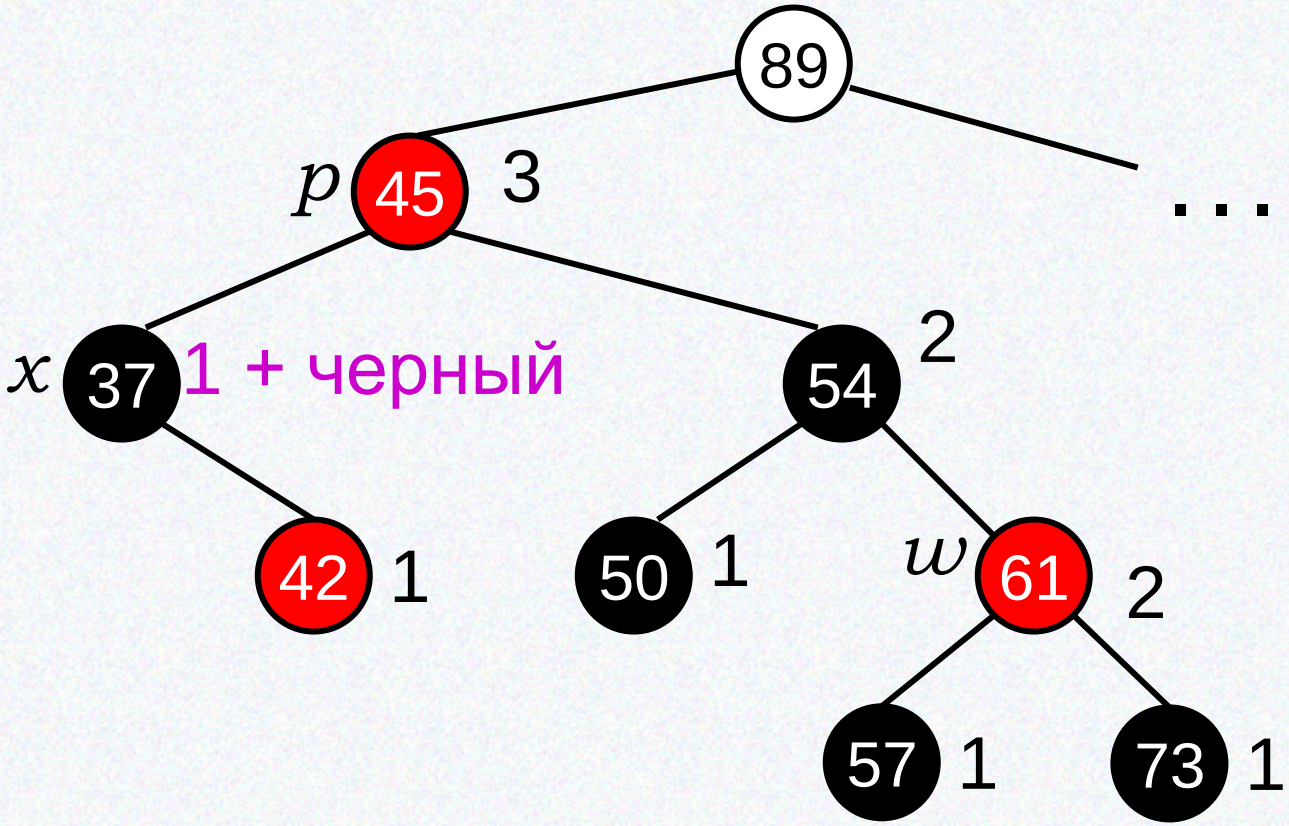
Случай 3



Случай 3



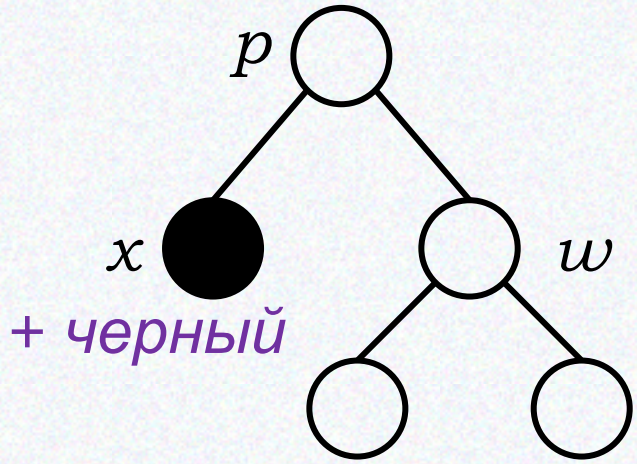
Случай 3



Случай 4

Узел w – черный,

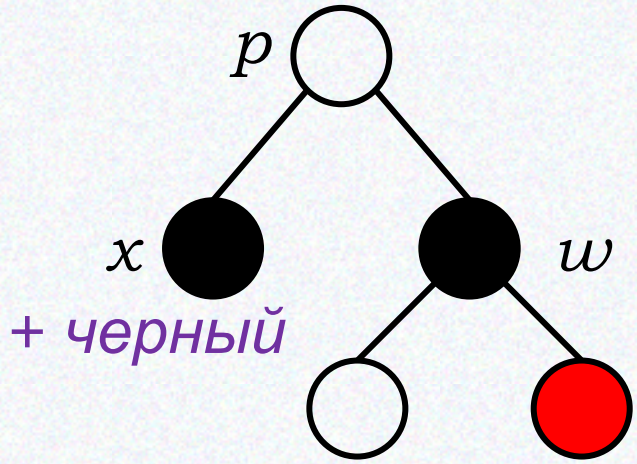
его правый дочерний узел – красный



Случай 4

Узел w – черный,

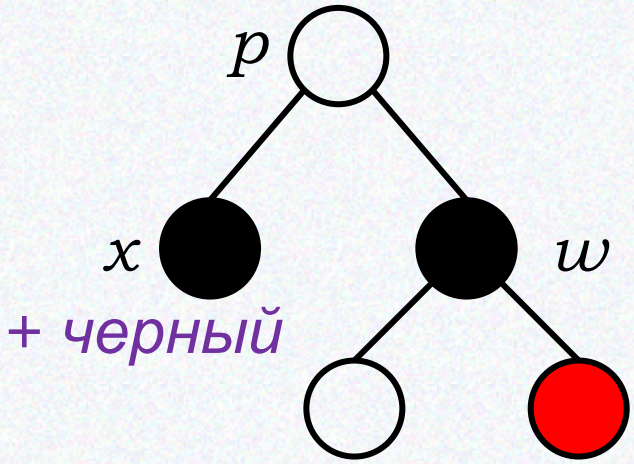
его правый дочерний узел – красный



Случай 4

Узел w – черный,

его правый дочерний узел – красный



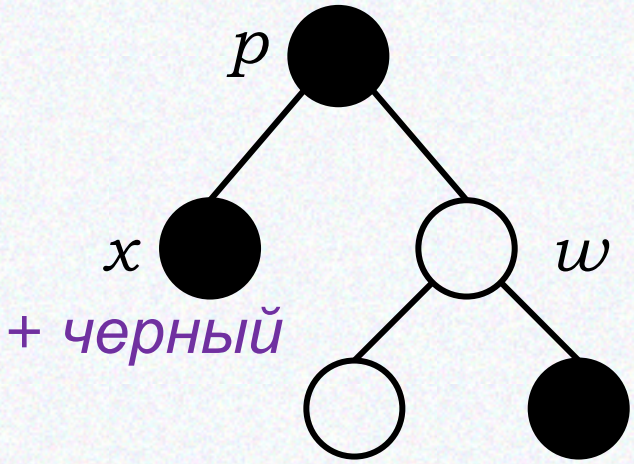
Коррекция:

Шаг 1 – обменять цвета у узлов w, p и правого дочернего узла w
(w->$right$ и p получают цвет w, а w – цвет p)

Случай 4

Узел w – черный,

его правый дочерний узел – красный



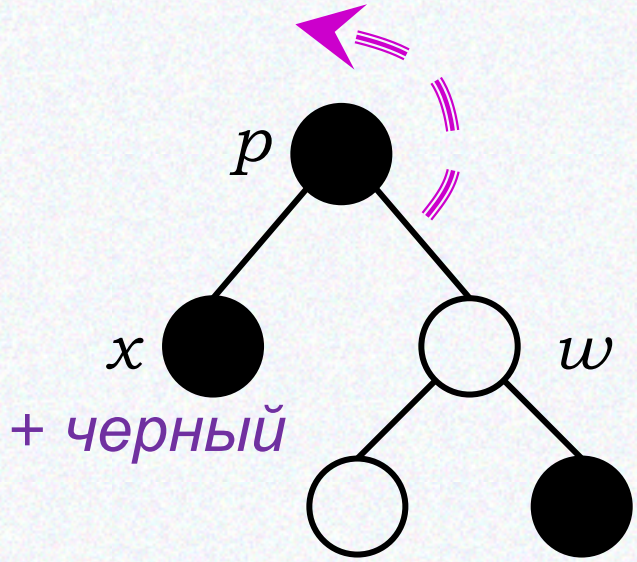
Коррекция:

Шаг 1 – обменять цвета у узлов w, p и правого дочернего узла w (w->$right$ и p получают цвет w, а w – цвет p)

Случай 4

Узел w – черный,

его правый дочерний узел – красный



Коррекция:

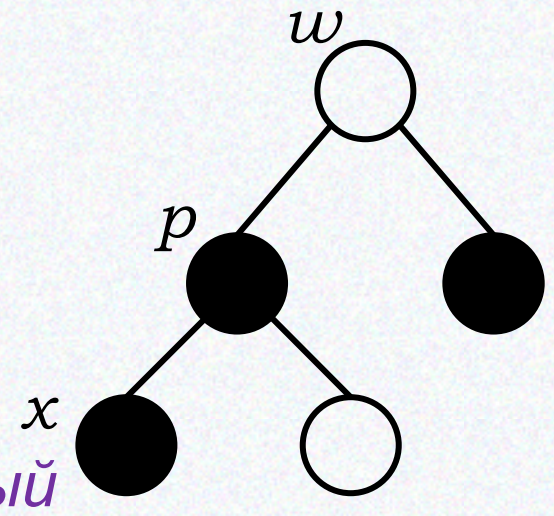
Шаг 1 – обменять цвета у узлов w, p и правого дочернего узла w
(w->$right$ и p получают цвет w, а w – цвет p)

Шаг 2 – левый поворот вокруг узла p

Случай 4

Узел w – черный,

его правый дочерний узел – красный



Коррекция:

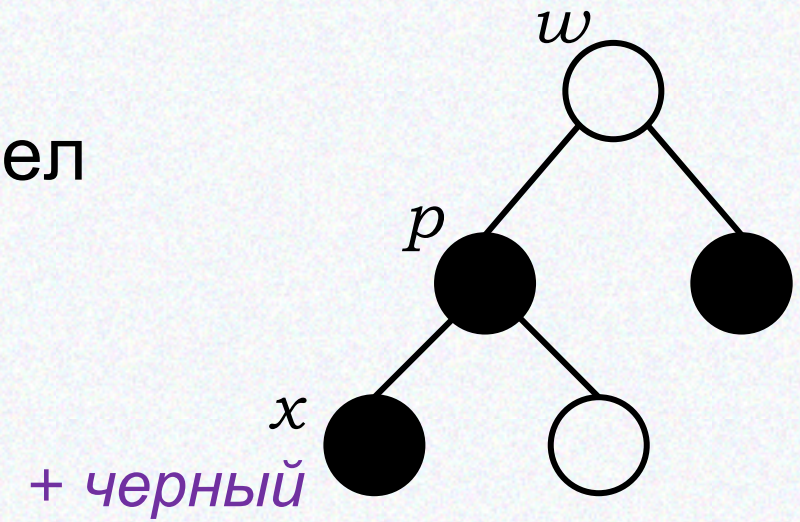
Шаг 1 – обменять цвета у узлов w, p и правого дочернего узла w
(w->$right$ и p получают цвет w, а w – цвет p)

Шаг 2 – левый поворот вокруг узла p

Случай 4

Узел w – черный,

его правый дочерний узел – красный



Коррекция:

Шаг 1 – обменять цвета у узлов w, p и правого дочернего узла w
(w->$right$ и p получают цвет w, а w – цвет p)

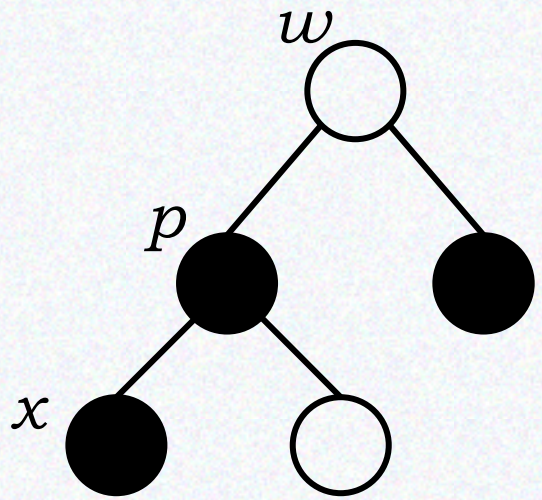
Шаг 2 – левый поворот вокруг узла p

Шаг 3 – отнять черный цвет у узла x

Случай 4

Узел w – черный,

его правый дочерний узел – красный



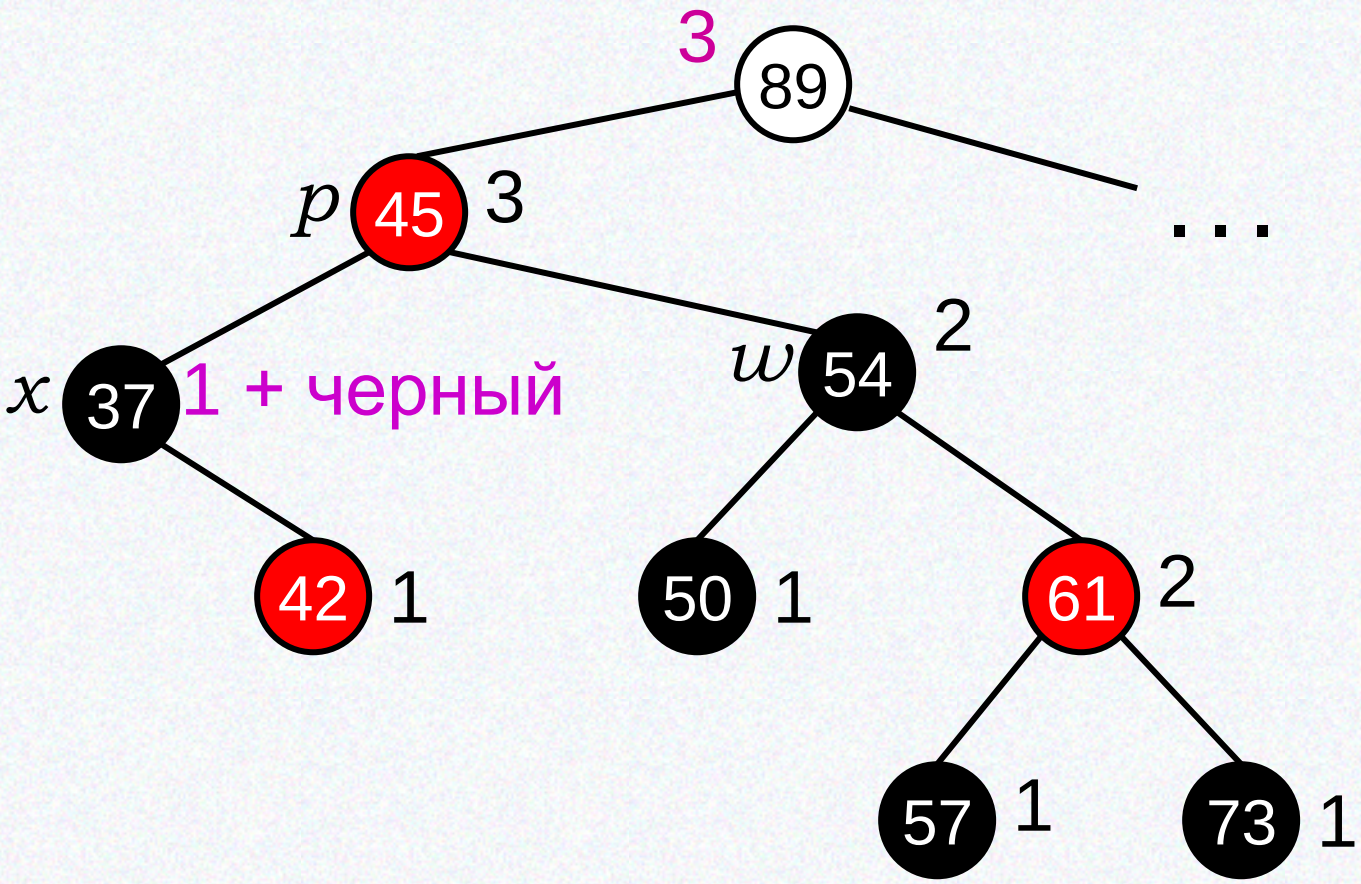
Коррекция:

Шаг 1 – обменять цвета у узлов w, p и правого дочернего узла w
(w->$right$ и p получают цвет w, а w – цвет p)

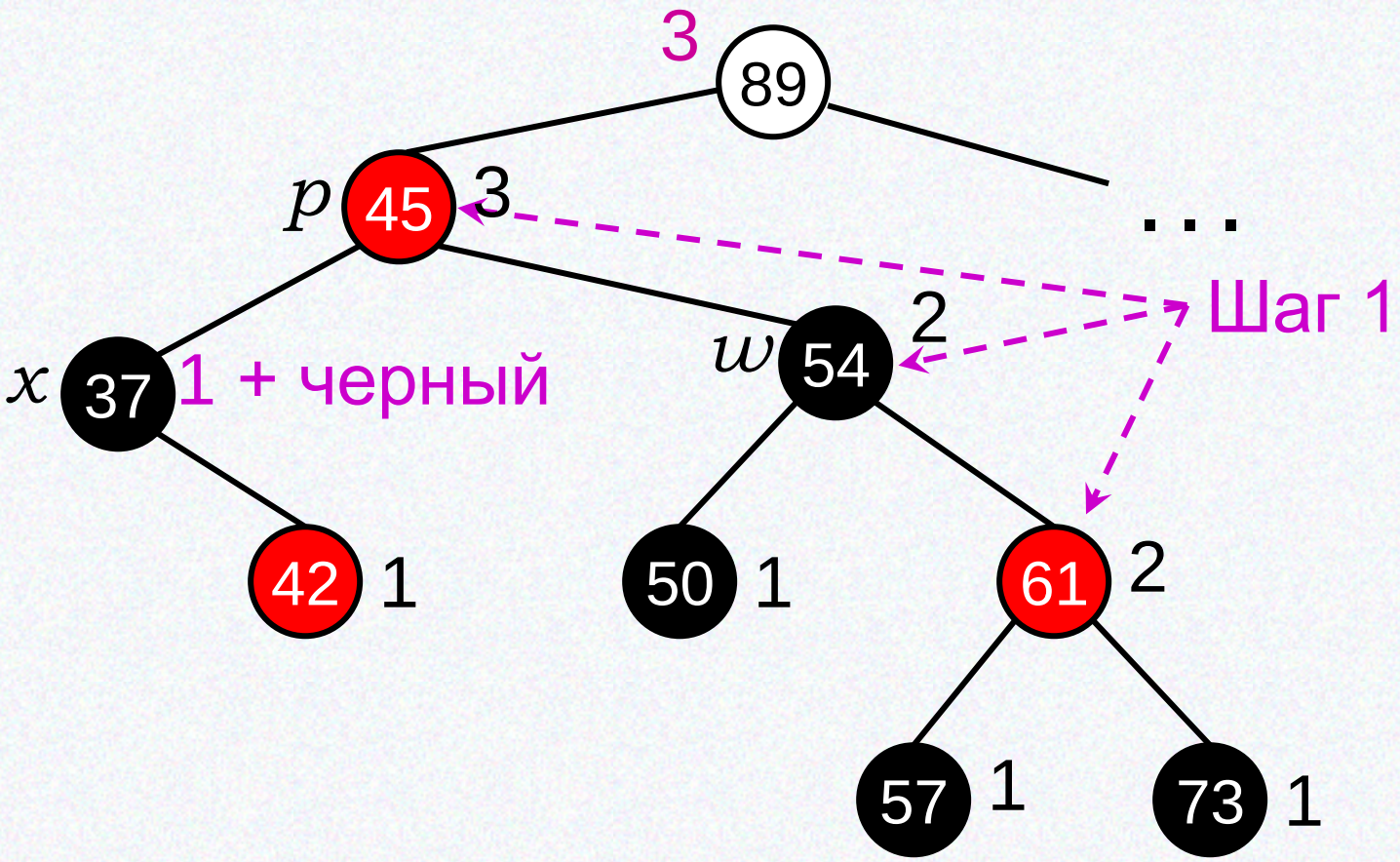
Шаг 2 – левый поворот вокруг узла p

Шаг 3 – отнять черный цвет у узла x

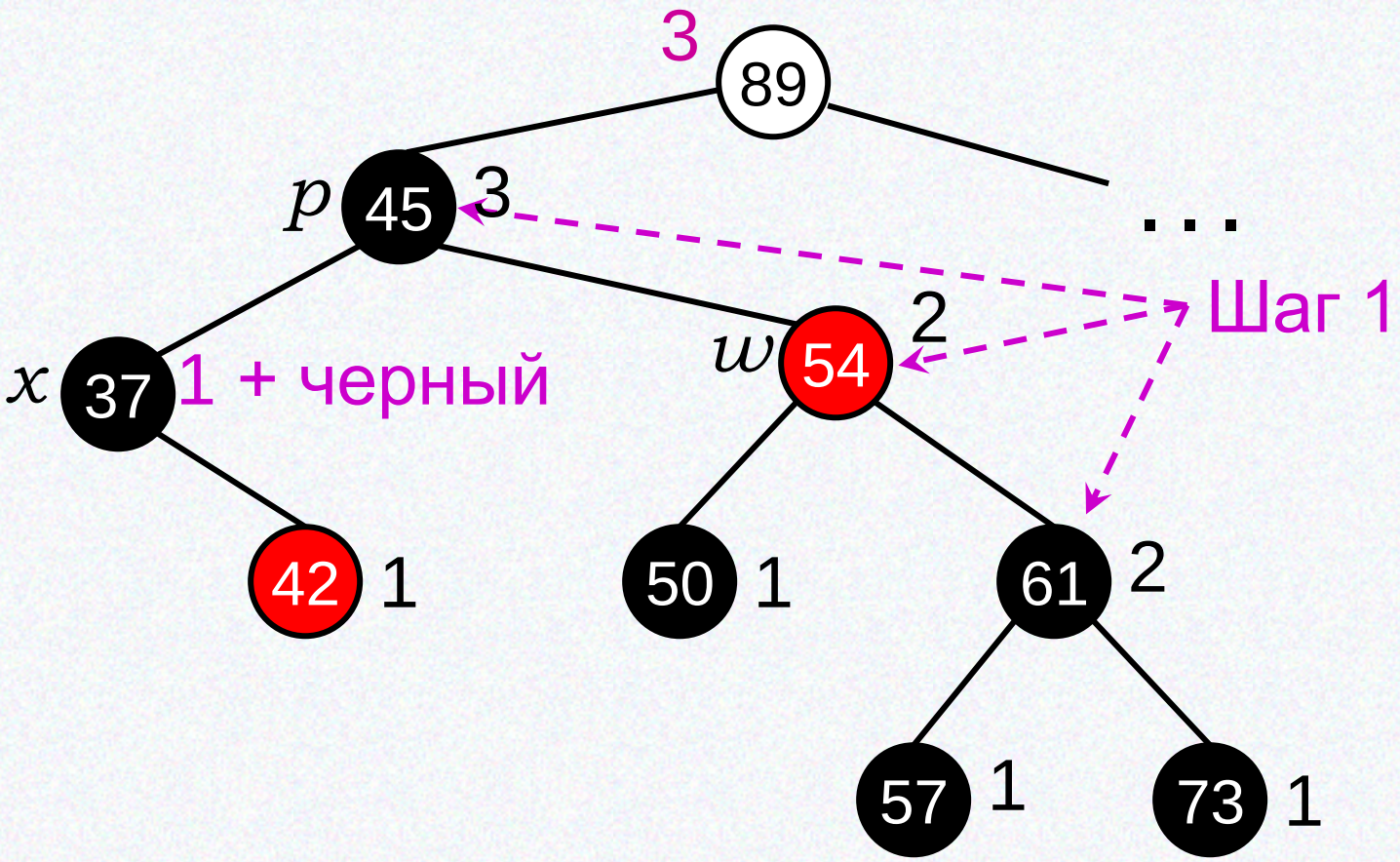
Случай 4



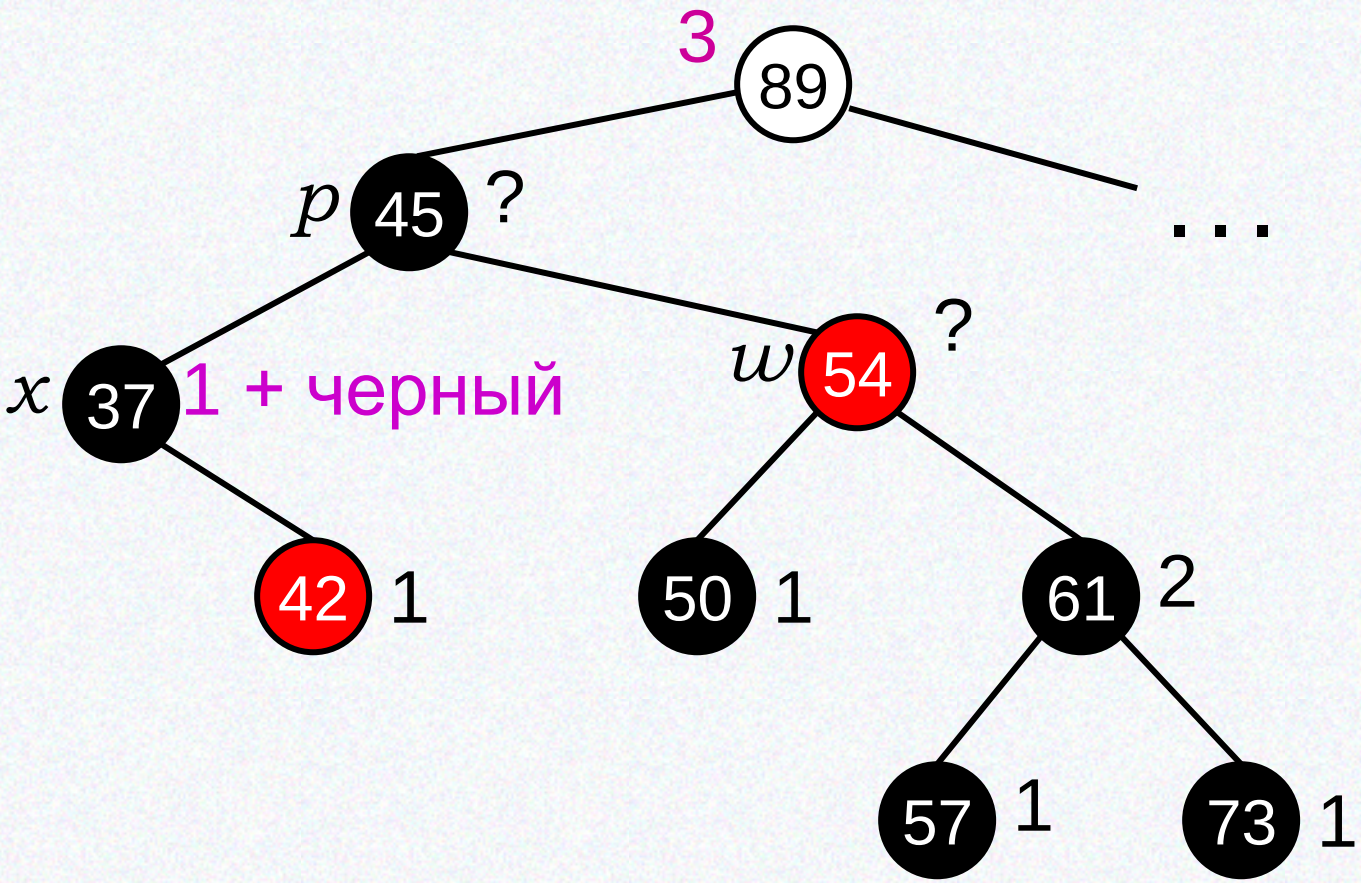
Случай 4



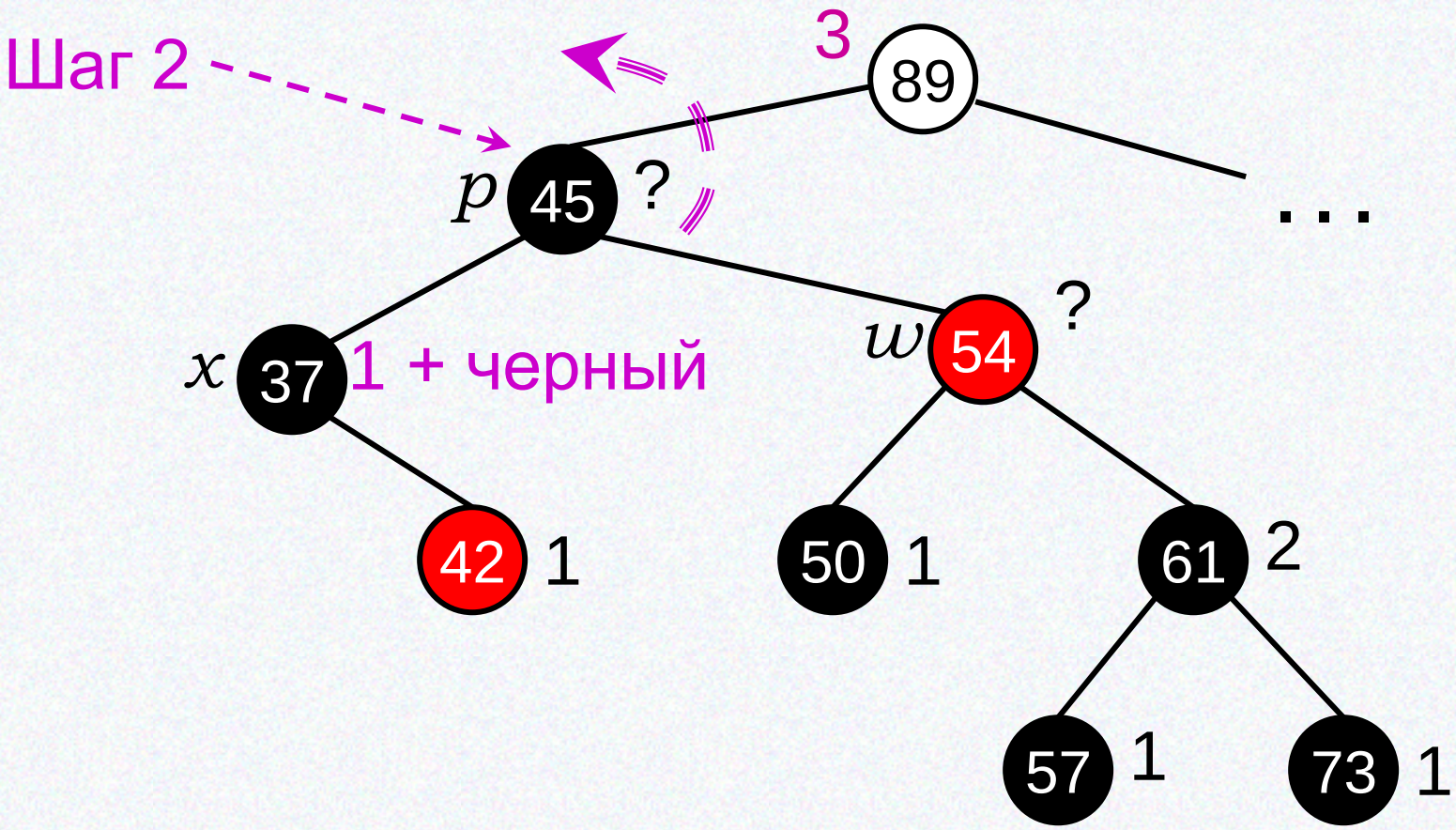
Случай 4



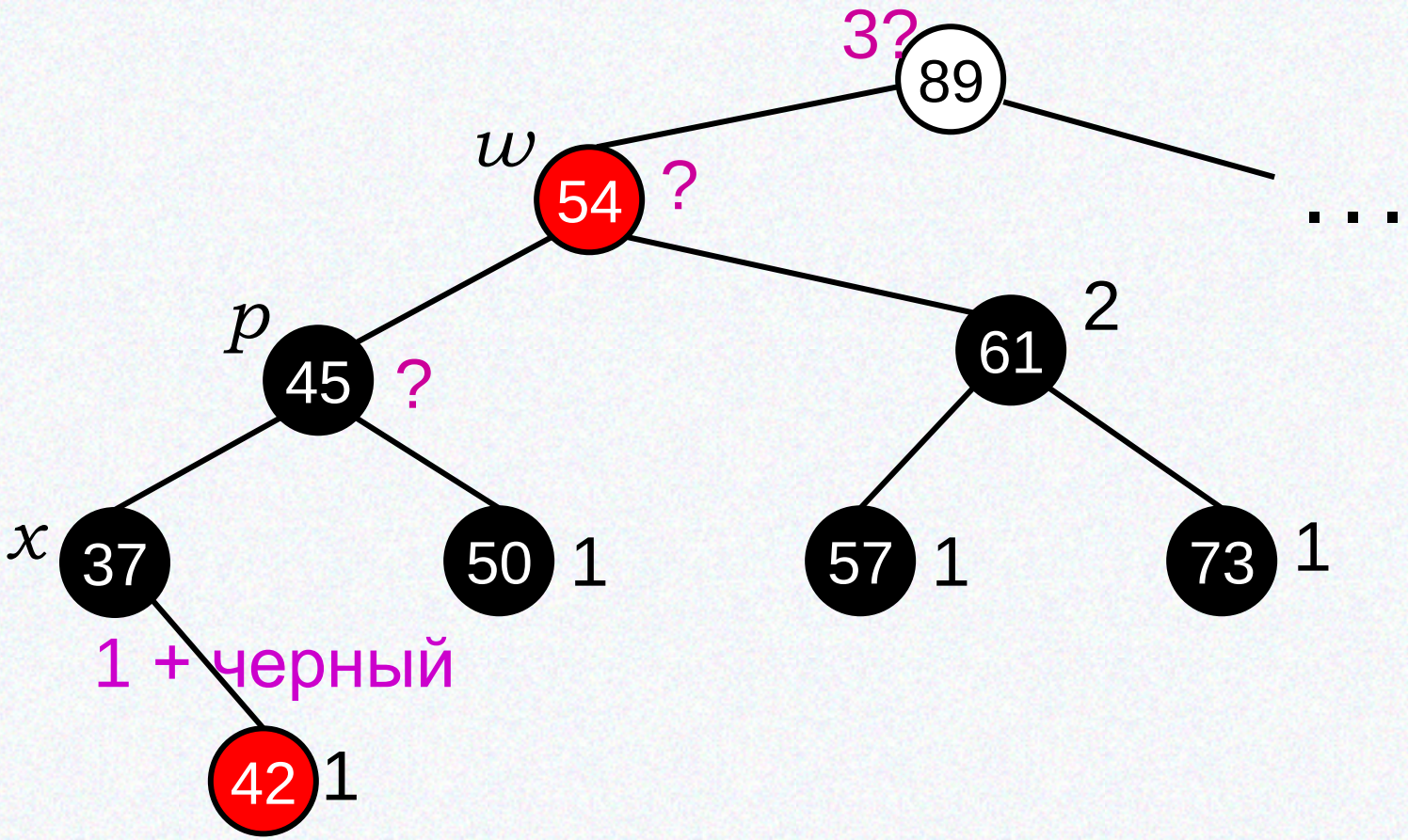
Случай 4



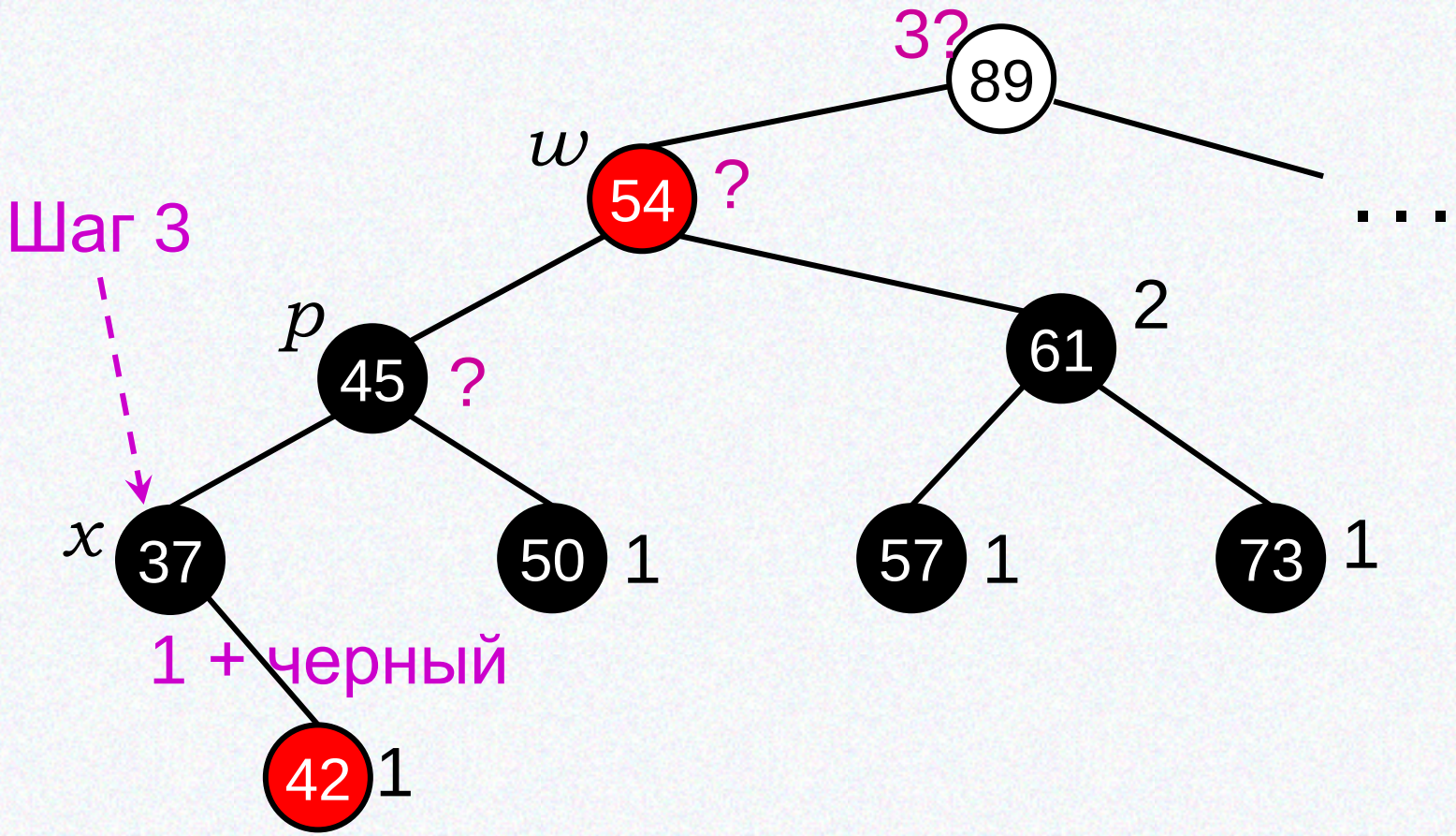
Случай 4



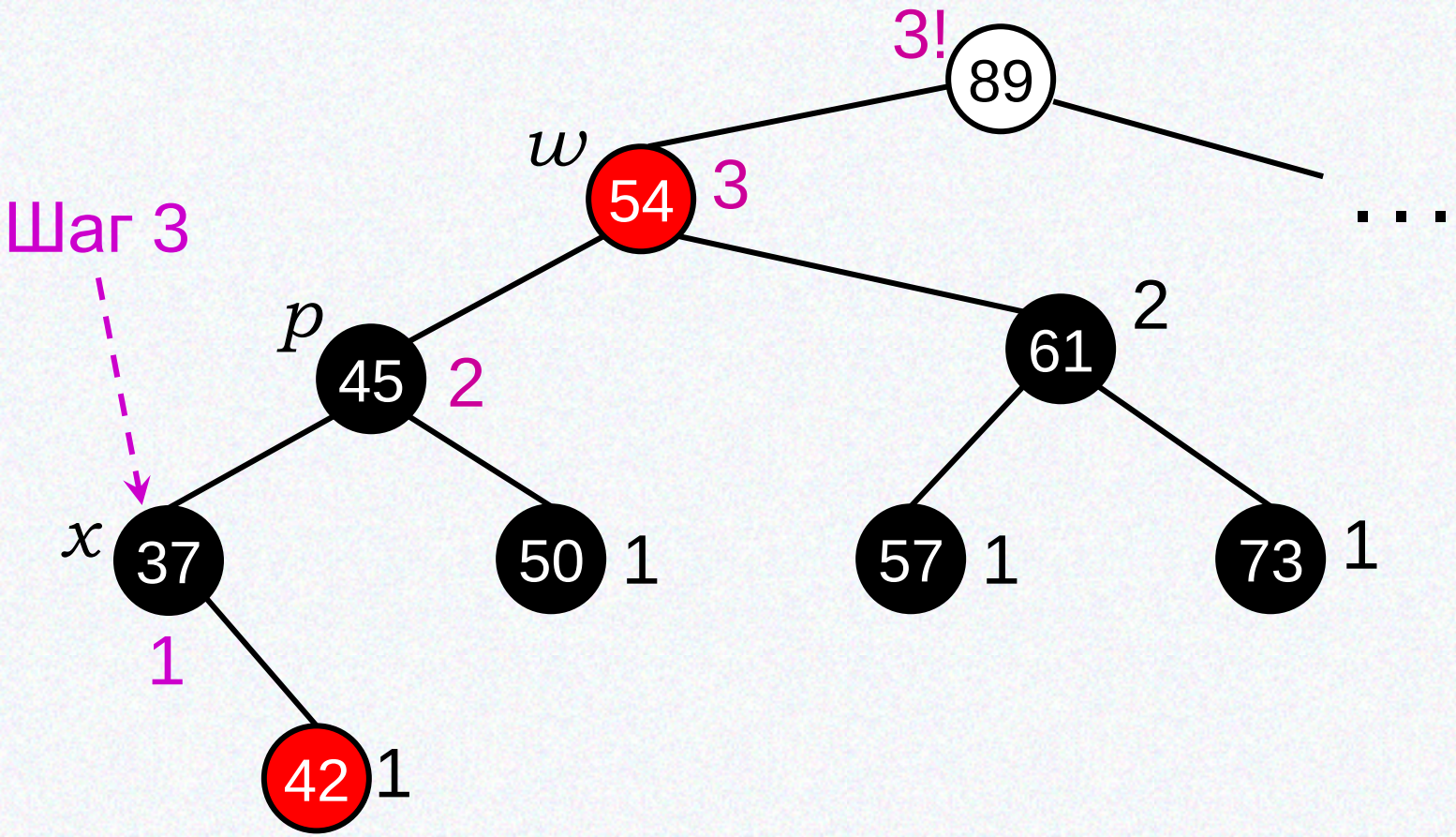
Случай 4



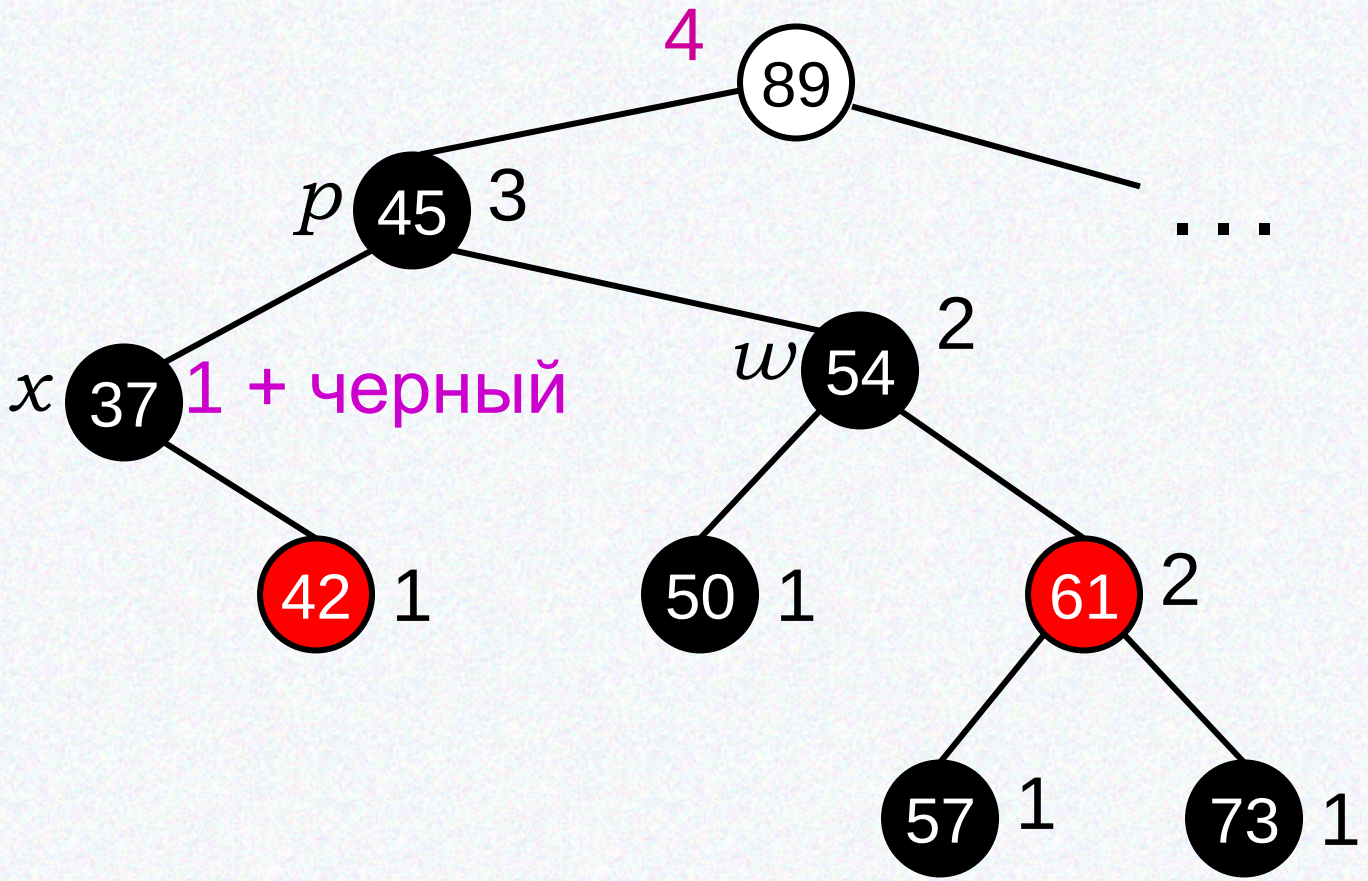
Случай 4



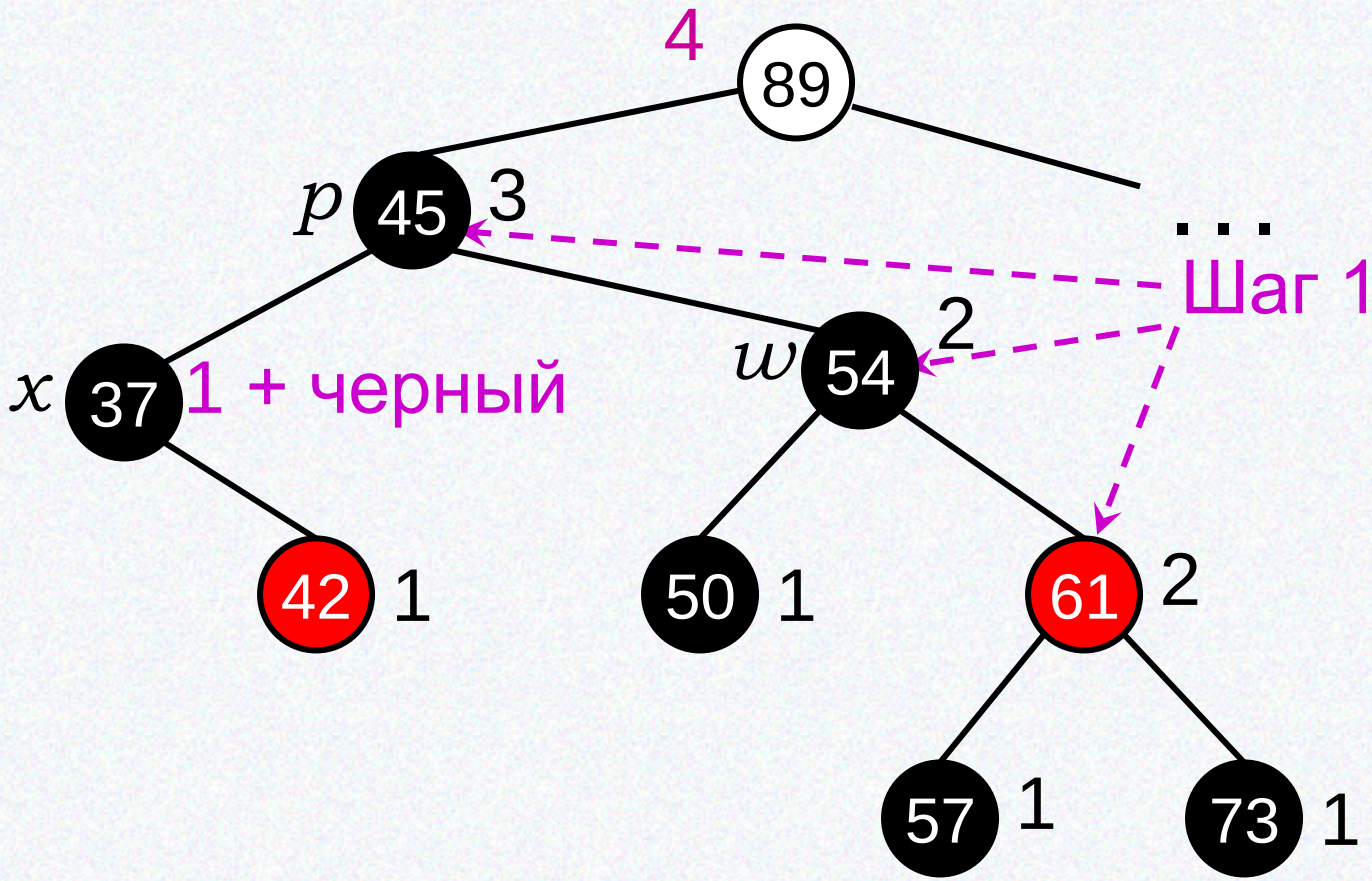
Случай 4



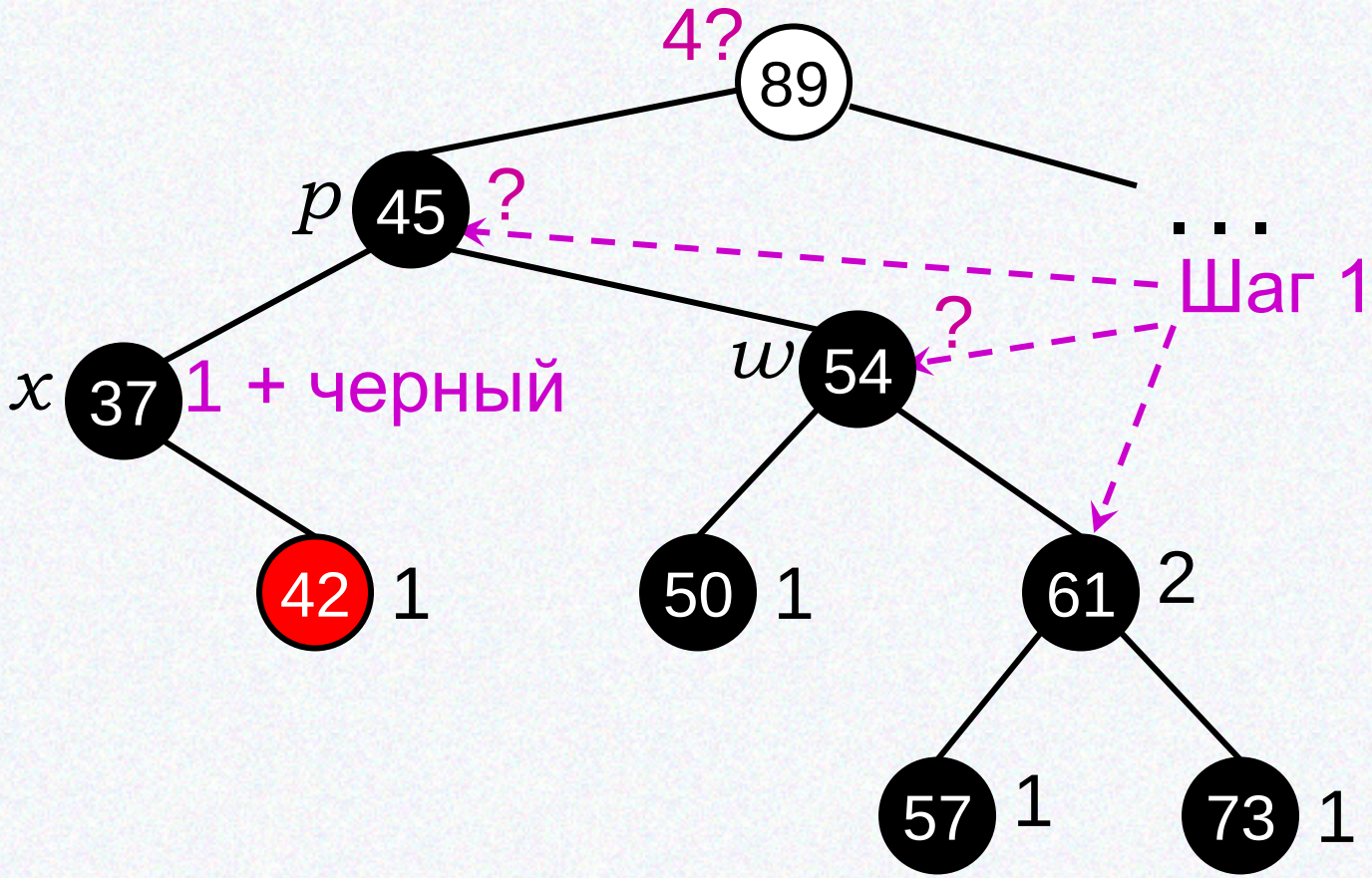
Случай 4



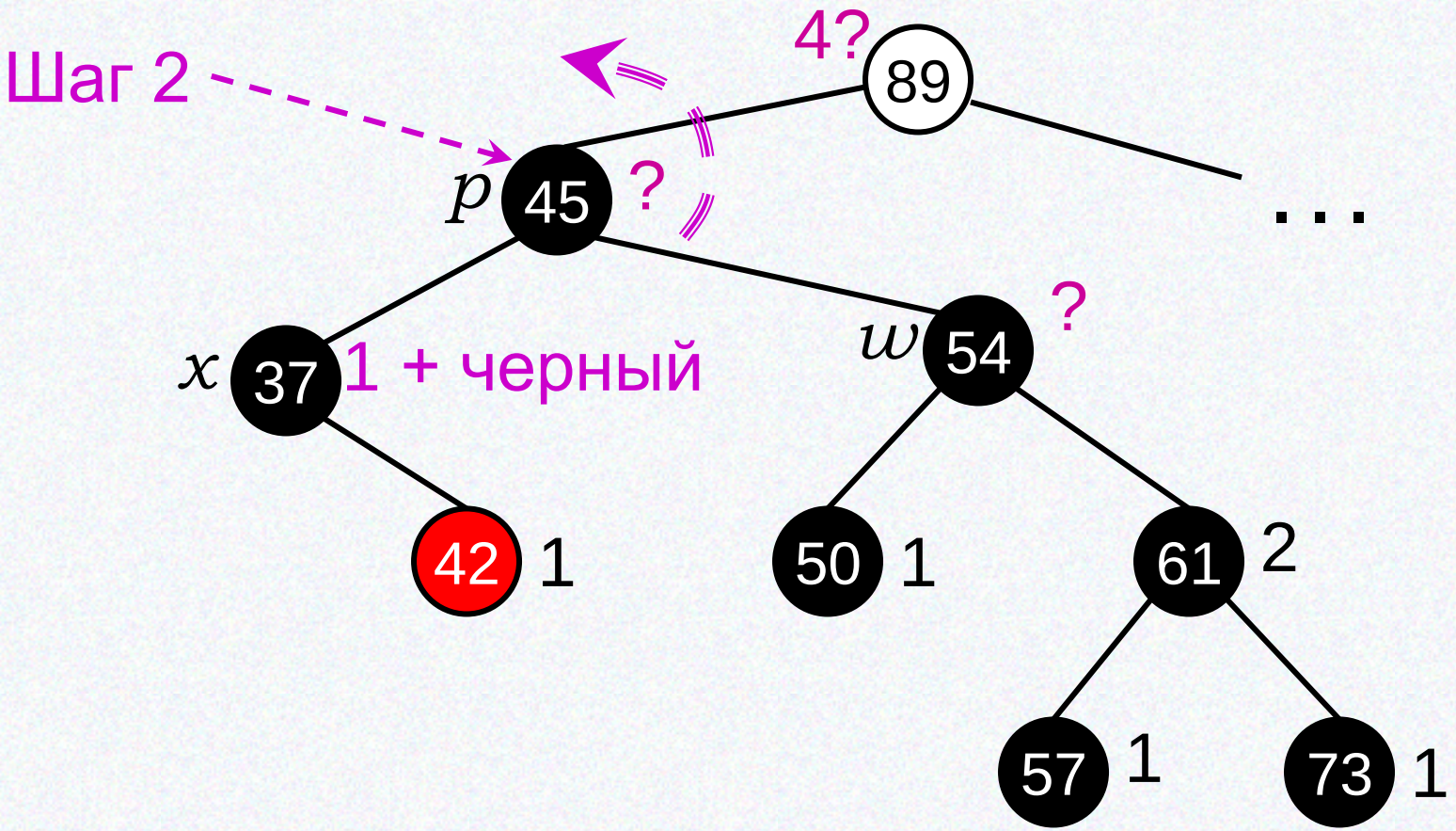
Случай 4



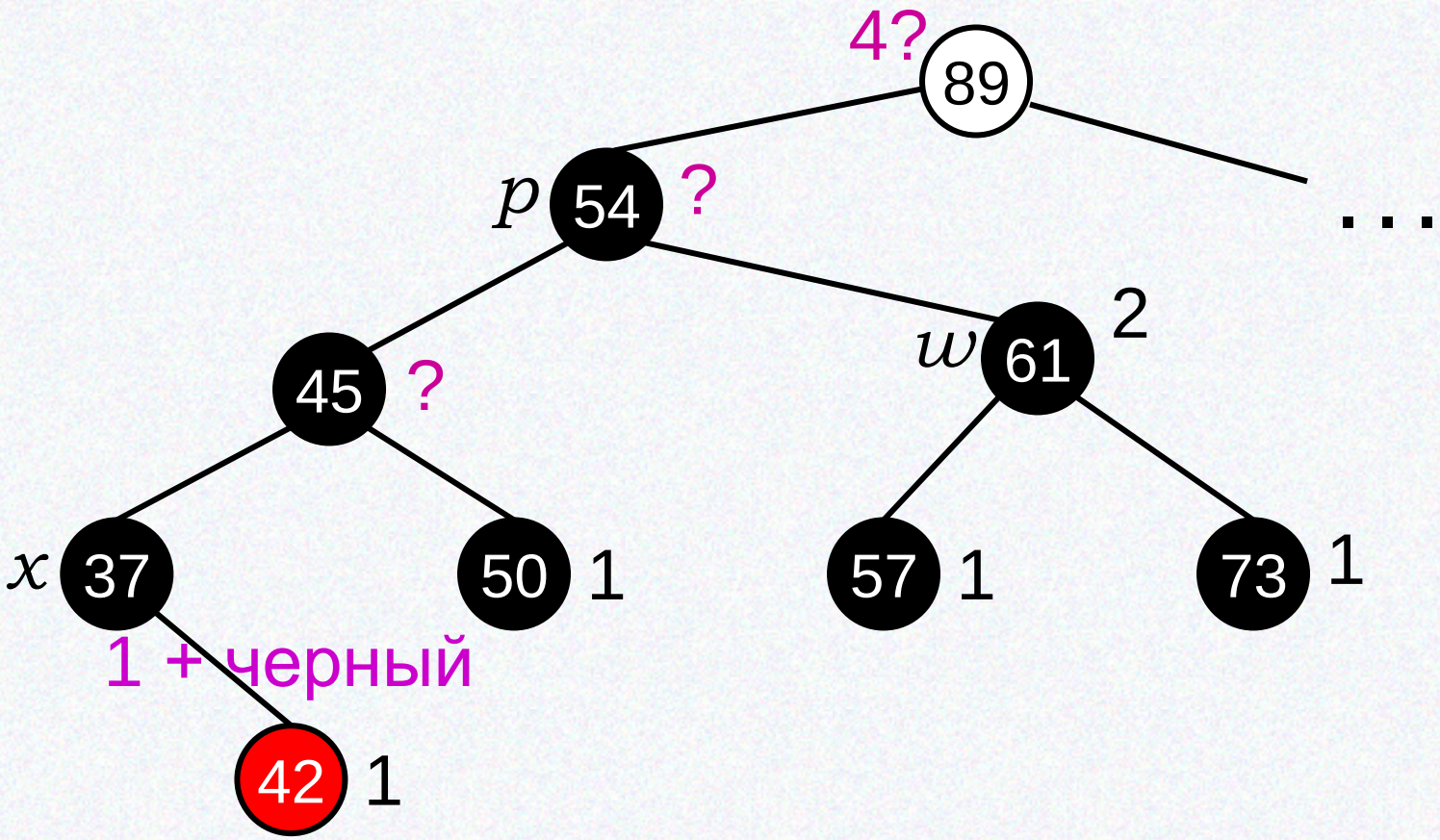
Случай 4



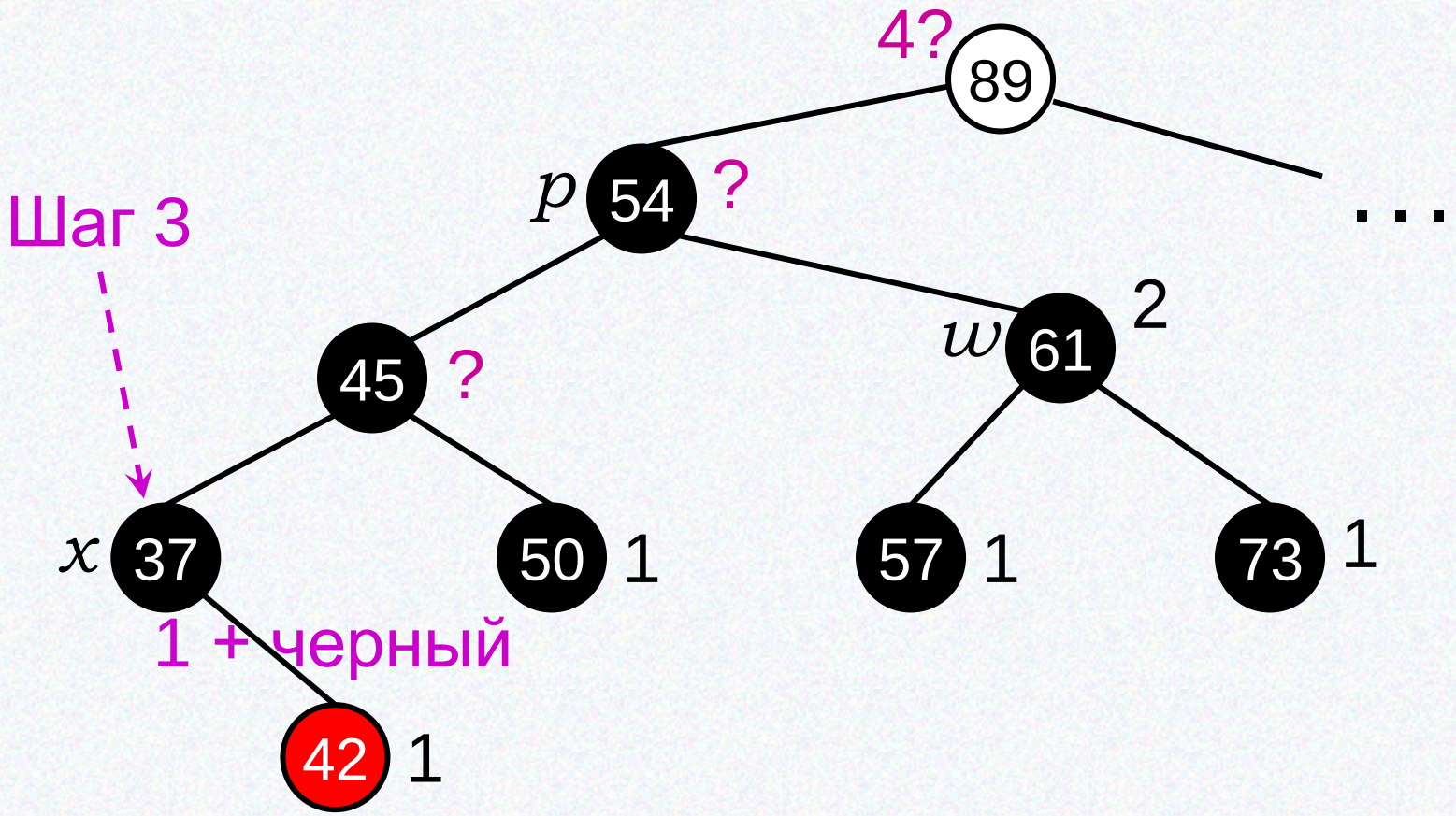
Случай 4



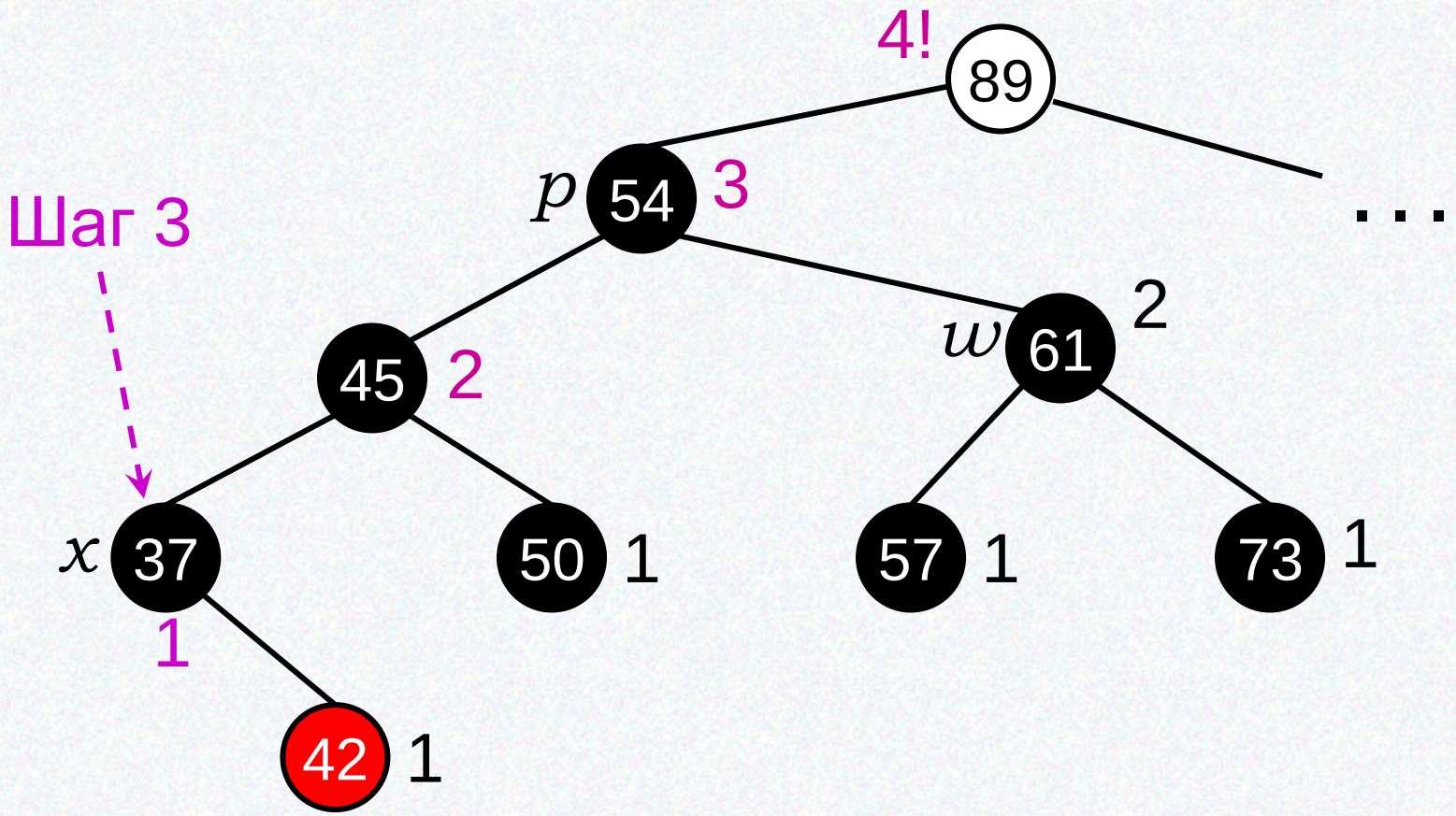
Случай 4



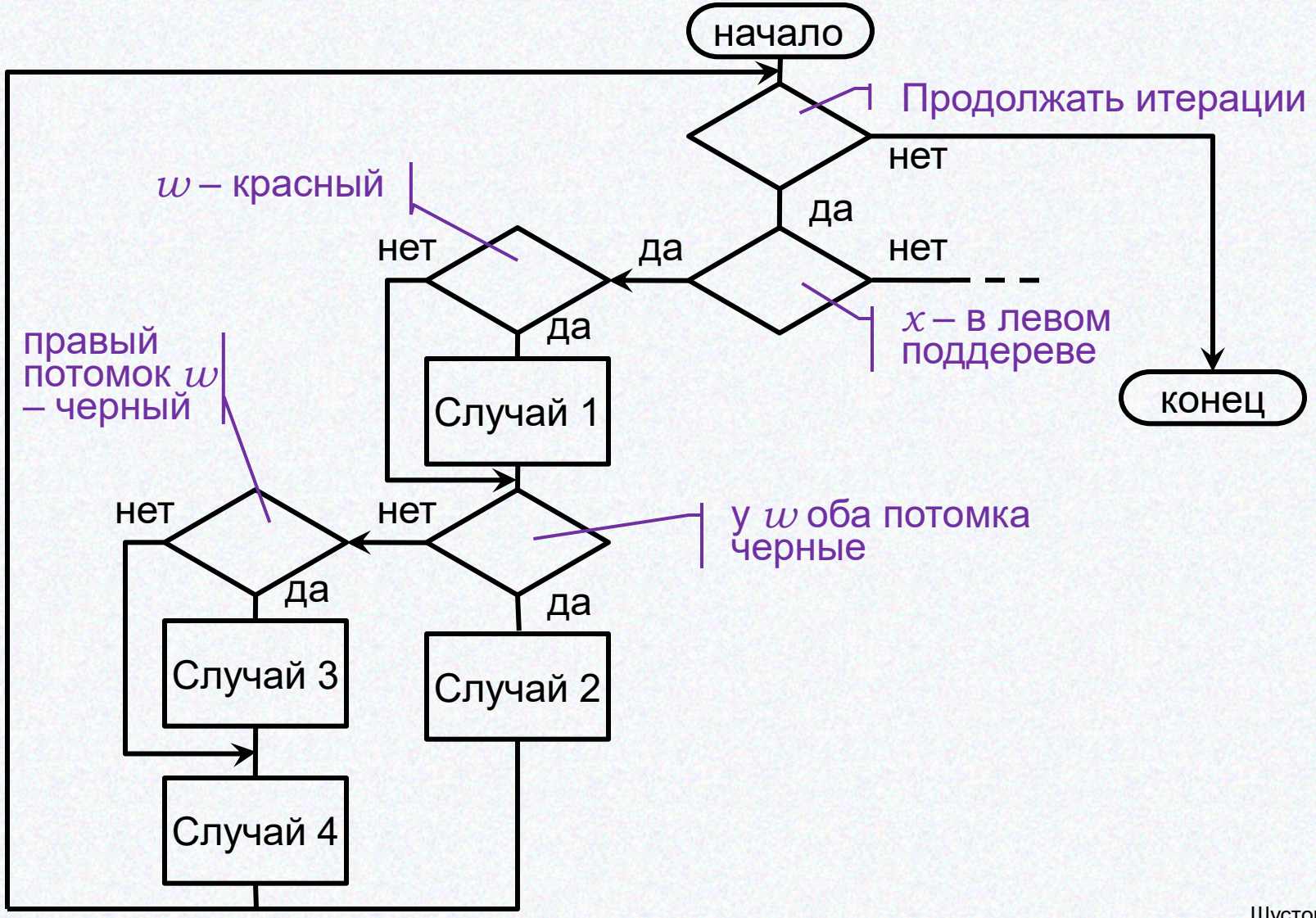
Случай 4



Случай 4



Восстановление цвета узла



RB_Delete_Fixup(x)

Обозначения:

RB_Delete_Fixup(x)

Обозначения:

x – анализируемый узел дерева

RB_Delete_Fixup(x)

Обозначения:

x – анализируемый узел дерева

$p = x\text{->}parent$ – родительский узел

RB_Delete_Fixup(x)

Обозначения:

x – анализируемый узел дерева

$p = x\text{->}parent$ – родительский узел

$w = p\text{->}right$ (или $p\text{->}left$) – второй потомок узла p

RB_Delete_Fixup(x)

Обозначения:

x – анализируемый узел дерева

$p = x\text{->}parent$ – родительский узел

$w = p\text{->}right$ (или $p\text{->}left$) – второй потомок узла p

x – дважды черный узел

RB_Delete_Fixup(x)

while продолжать итерации*:*

RB_Delete_Fixup(x)

while продолжать итерации*:*

x ≠ *root* и *x->color* == *BLACK* {

RB_Delete_Fixup(x)

while продолжать итерации:

$x \neq root$ и $x\text{->}color == BLACK$ {

проверка, в каком поддереве лежит x:

$p = x\text{->}parent$

RB_Delete_Fixup(x)

while продолжать итерации:

$x \neq root$ и $x\text{->}color == BLACK$ {

проверка, в каком поддереве лежит x:

$p = x\text{->}parent$

if x – в левом поддереве: $x == p\text{->}left$

{ # 1 – начало

RB_Delete_Fixup(x)

while продолжать итерации:

$x \neq root$ и $x\text{->}color == BLACK$ {

проверка, в каком поддереве лежит x:

$p = x\text{->}parent$

if x – в левом поддереве: $x == p\text{->}left$

{ # 1 – начало

$w = p\text{->}right$

RB_Delete_Fixup(x)

while продолжать итерации:

$x \neq root$ и $x\text{->}color == BLACK$ {

проверка, в каком поддереве лежит x:

$p = x\text{->}parent$

if x – в левом поддереве: $x == p\text{->}left$

{ # 1 – начало

$w = p\text{->}right$

анализ возможных ситуаций

RB_Delete_Fixup(x)

if w – красый: *w->color == RED* {

– *случай 1*

RB_Delete_Fixup(x)

if w – красый: w->*color* == *RED* {

– *случай 1*

поменять цвета у w и p:
w->*color* = *BLACK*
p->*color* = *RED*

RB_Delete_Fixup(x)

if w – красый: w->*color* == *RED* {

– *случай 1*

поменять цвета у w и p:
w->*color* = *BLACK*
p->*color* = *RED*

выполнить левый поворот вокруг p:
Left_Rotate(p)

RB_Delete_Fixup(x)

if w – красый: w->*color* == *RED* {

– *случай 1*

поменять цвета у w и p:

w->*color* = *BLACK*

p->*color* = *RED*

выполнить левый поворот вокруг p:

Left_Rotate(p)

w = p->*right*

} – *случай 1*

RB_Delete_Fixup(x)

if у *w* оба потомка черные:
w->left->color == BLACK и
w->right-color == BLACK
{ *– случай 2*

RB_Delete_Fixup(x)

if у w оба потомка черные:
w->*left*->*color* == *BLACK* и
w->*right*-*color* == *BLACK*
{ – *случай 2*

забрать черную окраску у w:
w->*color* = *RED*

RB_Delete_Fixup(x)

if у w оба потомка черные:
w->*left*->*color* == *BLACK* и
w->*right*-*color* == *BLACK*
{ – *случай 2*

забрать черную окраску у w:
w->*color* = *RED*

переместиться вверх по дереву:
$x = p$
} – *случай 2*

RB_Delete_Fixup(x)

else {

if правый потомок w черный:

w->right->color == BLACK

{ – *случай 3*

RB_Delete_Fixup(x)

else {

if правый потомок *w* черный:
w->right->color == BLACK
{ – *случай 3*

поменять цвета у *w* и *w->left*:
w->color = RED
w->left->color = BLACK

RB_Delete_Fixup(x)

else {

if правый потомок *w* черный:
w->right->color == BLACK
{ – *случай 3*

поменять цвета у *w* и *w->left*:
w->color = RED
w->left->color = BLACK

правый поворот вокруг *w*:
Right_Rotate(w)

RB_Delete_Fixup(x)

else {

if правый потомок *w* черный:
w->right->color == BLACK
{ – *случай 3*

поменять цвета у *w* и *w->left*:
w->color = RED
w->left->color = BLACK

правый поворот вокруг *w*:
Right_Rotate(w)

w = p->right
} – *случай 3*

RB_Delete_Fixup(x)

– случай 4

RB_Delete_Fixup(x)

– *случай 4*

поменять цвета:

w->color = p->color

p->color = BLACK

w->right->color = BLACK

RB_Delete_Fixup(x)

– случай 4

поменять цвета:

w->color = p->color

p->color = BLACK

w->right->color = BLACK

левый поворот вокруг узла *p*

Left_Rotate(p)

RB_Delete_Fixup(x)

– *случай 4*

поменять цвета:

w->color = p->color

p->color = BLACK

w->right->color = BLACK

левый поворот вокруг узла *p*

Left_Rotate(p)

x = root

}

} # 1 – конец

RB_Delete_Fixup(x)

else {

повторить коды между #1 – начало и #1 – конец, заменив *left* на *right*, и наоборот

}

}

RB_Delete_Fixup(x)

else {

повторить коды между #1 – начало и #1 – конец, заменив *left* на *right*, и наоборот

}

}

x->color = BLACK