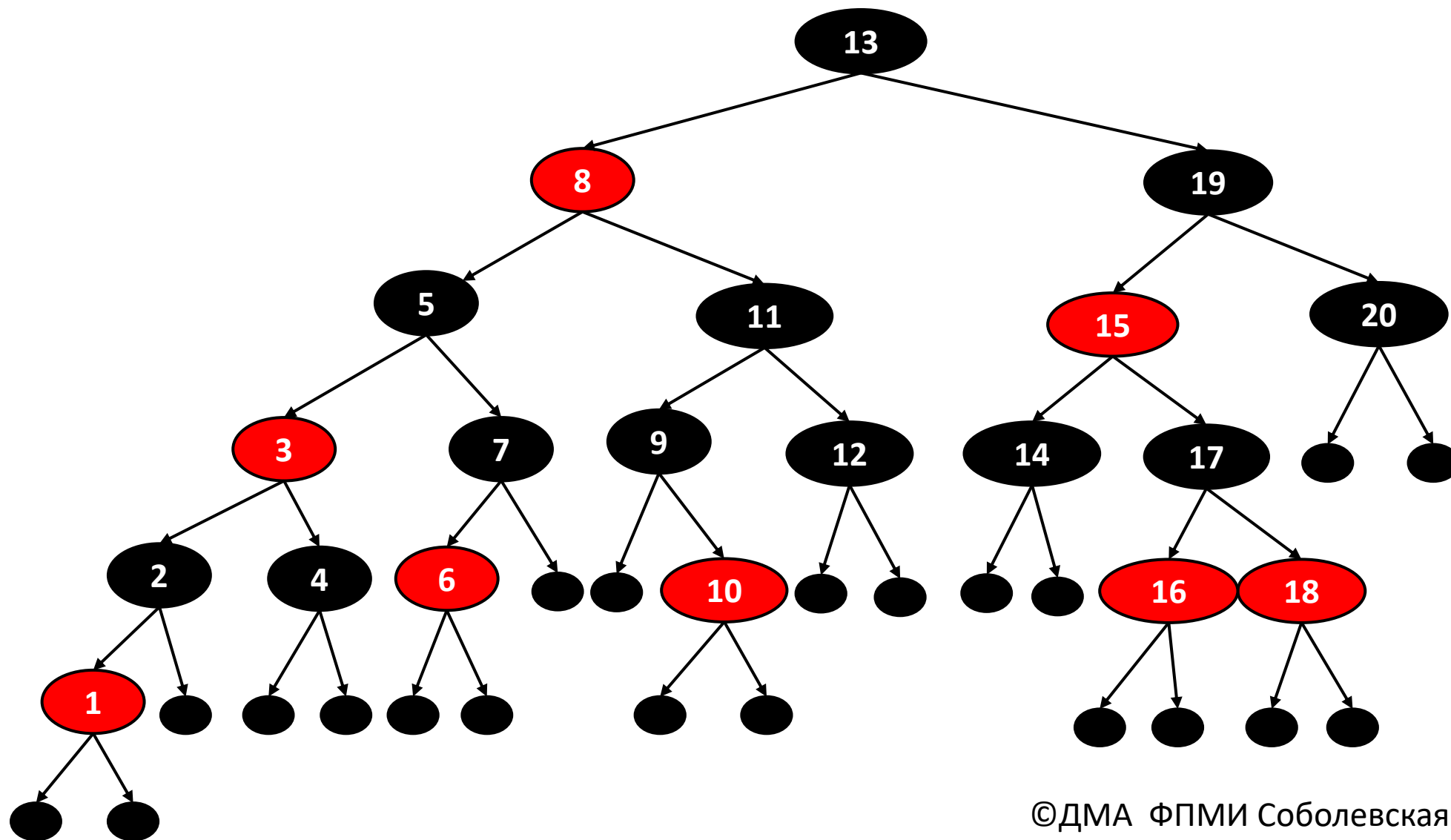


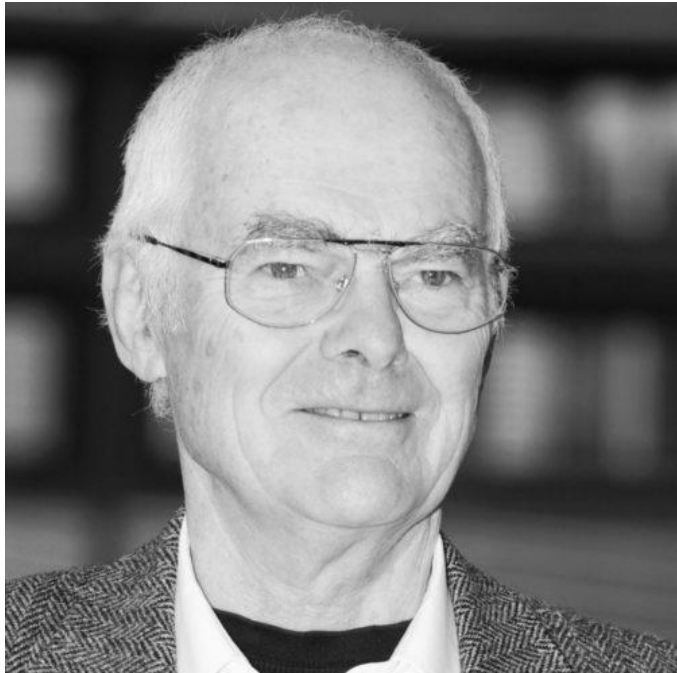


ОРГАНИЗАЦИЯ ПОИСКА

Красно-чёрное дерево (англ. red-black tree)



Изобретателем красно-чёрного
дерева считают немецкого учёного:

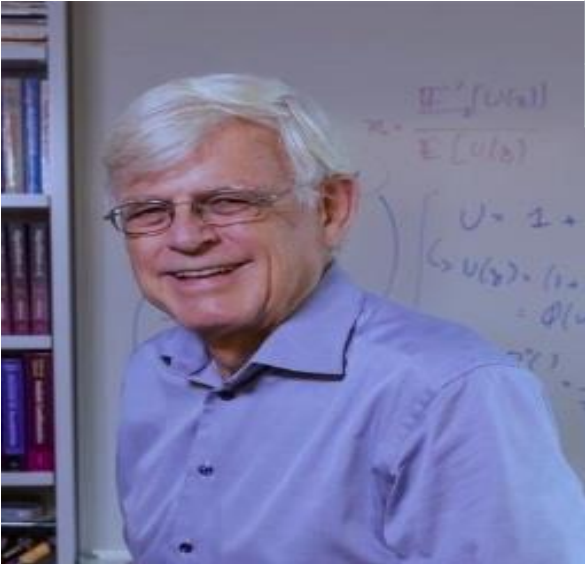
	Рудольф Байер	
	Rudolf Bayer	
	Дата рождения	7 мая 1939
	Страна	Германия
	Научная сфера	информатика
	Место работы	Мюнхенский технический университет
	Известен как	изобретатель красно-чёрного дерева , UV-дерева , В-дерева

В 1978 году в статье **Л. Гибаса** и **Р. Седжвика**
структура данных получила название «**красно-чёрное** дерево»:



Леонидас Джон (Иоаннис) Гибас греч . Λεωνίδας Γκίμπας	
Национальность	Греческий - Американский
Научная карьера	
профессор компьютерных наук и электротехники Пола Пиготта в Стэнфордском университете , где он возглавляет группу геометрических вычислений и является сотрудником лабораторий компьютерной графики и искусственного интеллекта	
Ученик (докторант) Дональда Кнута	

✓ по словам Л. Гибаса, они использовали ручки двух цветов;



Роберт Седжвик	
Robert Sedgwick	
Дата рождения	20 декабря 1946
Страна	США
Научная сфера	Информатика
Место работы	Принстонский университет

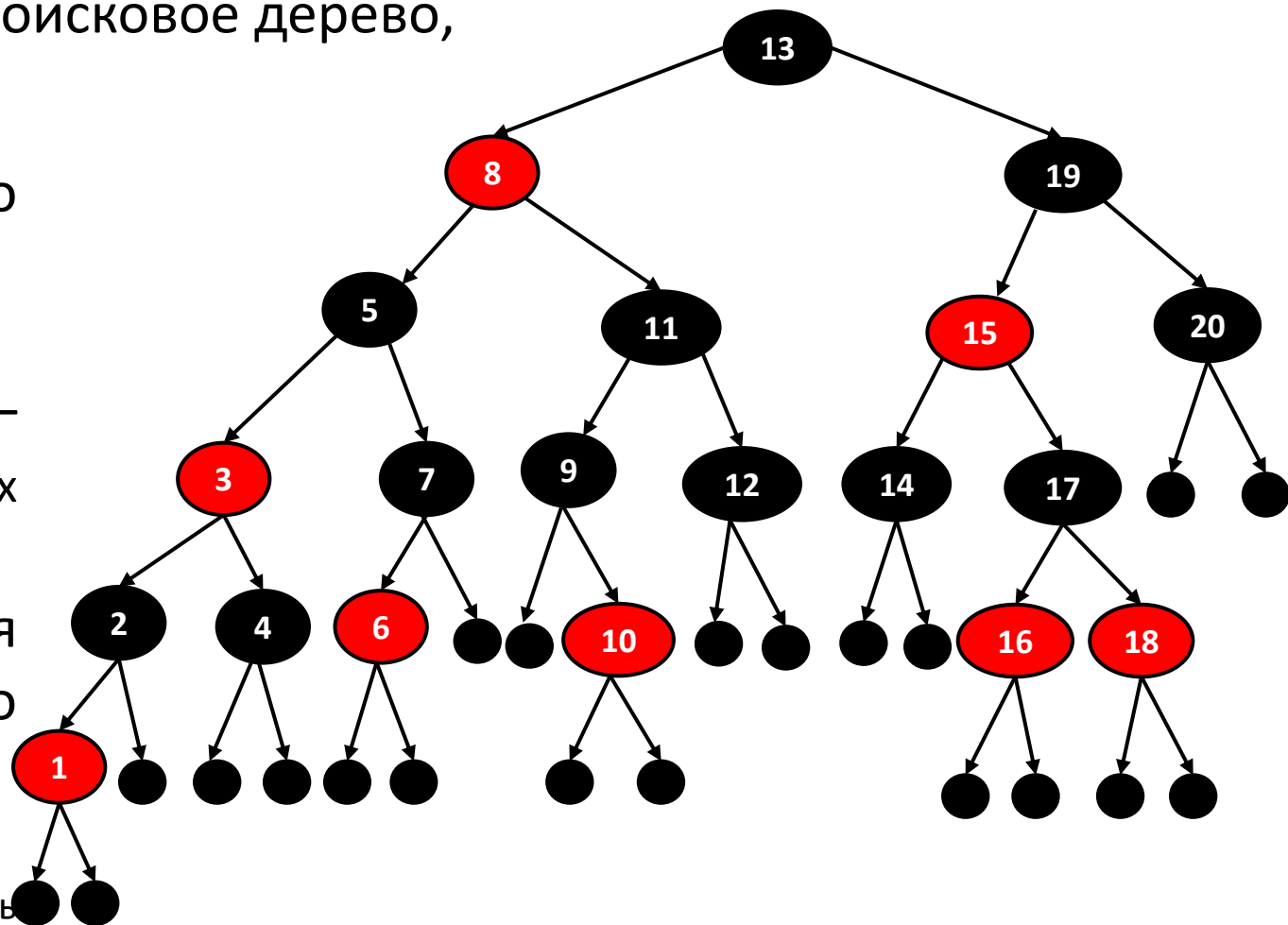
✓ по словам Р. Седжвика, красный и черный цвет лучше всех смотрелись на лазерном принтере Xerox.

Красно-чёрное дерево

- является примером бинарного поискового дерева;
- специальные операции балансировки гарантируют, что высота красно-чёрного дерева не превзойдёт $O(\log n)$;
- при этом время выполнения процедур балансировки (перекрашивания вершин, повороты) также ограничено этой величиной.

Красно-чёрное дерево – это бинарное поисковое дерево, для которого выполняются **RB**-свойства:

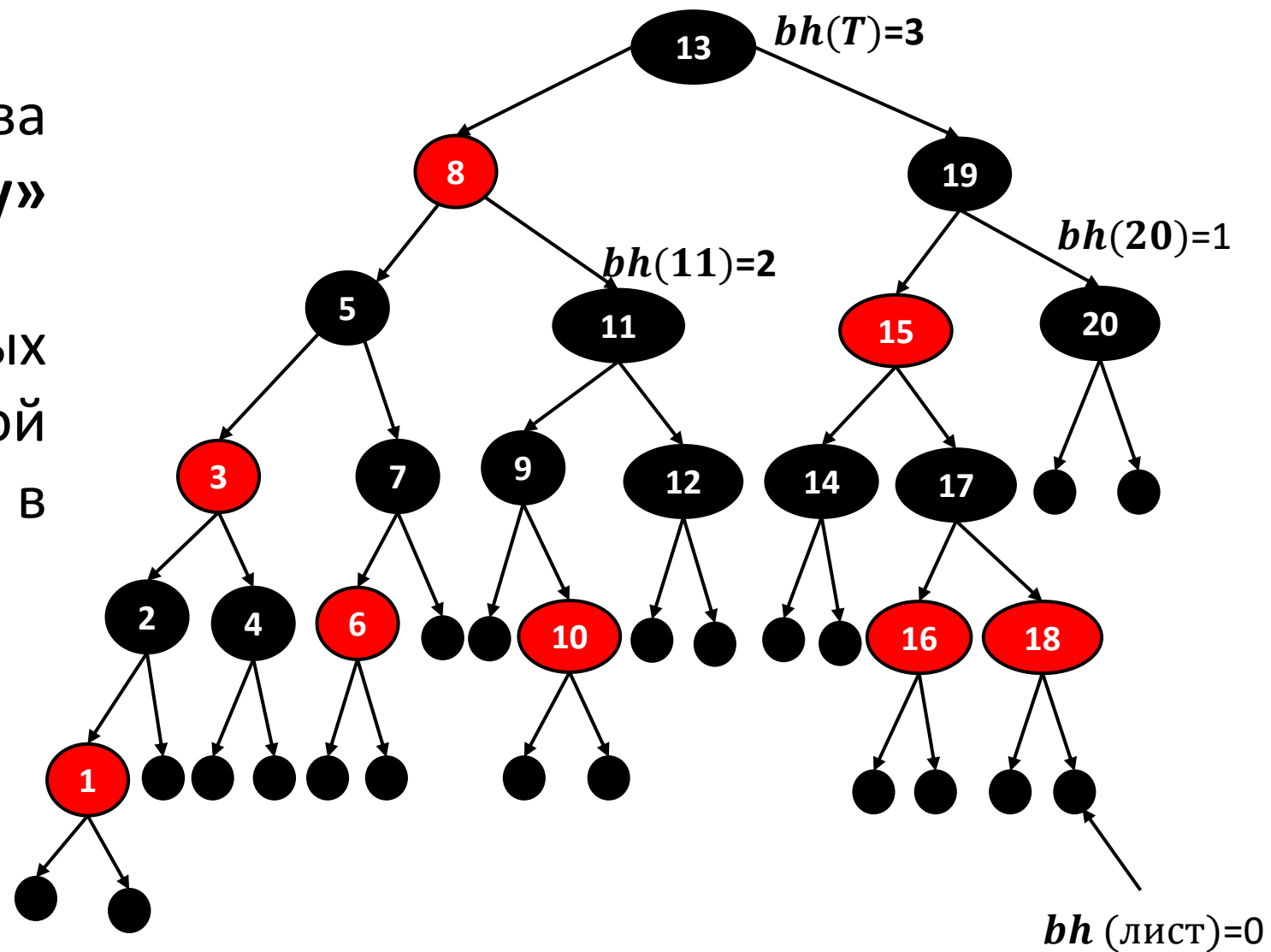
- 1) каждая вершина является либо **красной**, либо **чёрной**;
- 2) каждый **лист** – **чёрный**;
- 3) у **красной** вершины **оба сына** – **чёрные** (нет двух подряд идущих красных вершин);
- 4) любой путь из корня в листья содержит одинаковое количество **чёрных** вершин;
- 5) **корень** – **чёрный**
(это требование не обязательно, так как корень всегда можно перекрасить в чёрный цвет и это не нарушит ни одно из RB-свойств).



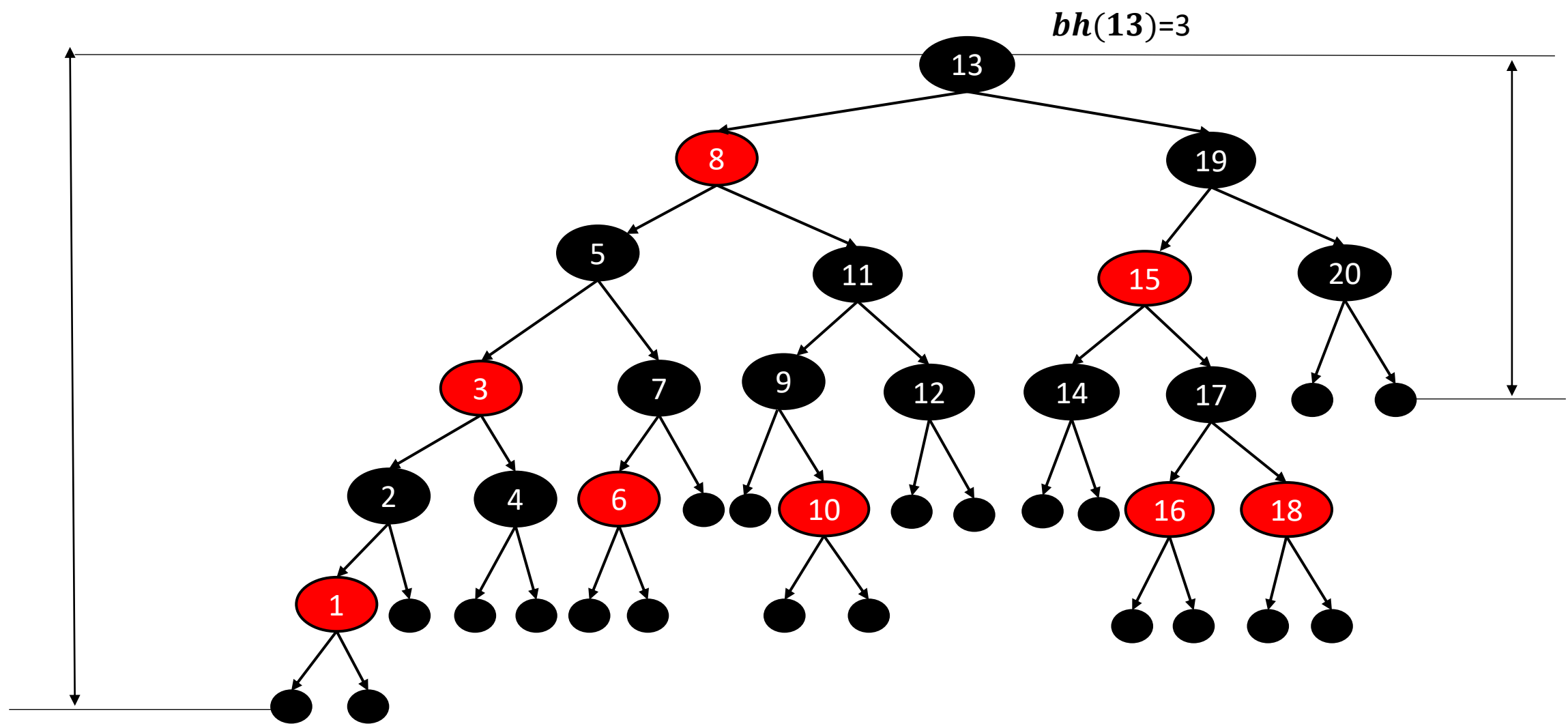
Если к каждой вершине, которая не имеет сыновей, добавить фиктивный чёрный лист, то мы получим, что каждая внутренняя вершина дерева содержит ключевое значения, а все листья – фиктивные.

Для каждой вершины v дерева определим её «чёрную высоту» (англ. **black-height**):

$bh(v)$ – количество чёрных вершин (без учёта самой вершины v) на пути из v в лист.



Чёрной высотой дерева будем считать чёрную высоту его корня.



Для каждой вершины v любой путь из v в листья содержит одинаковое число чёрных вершин (свойство 4).

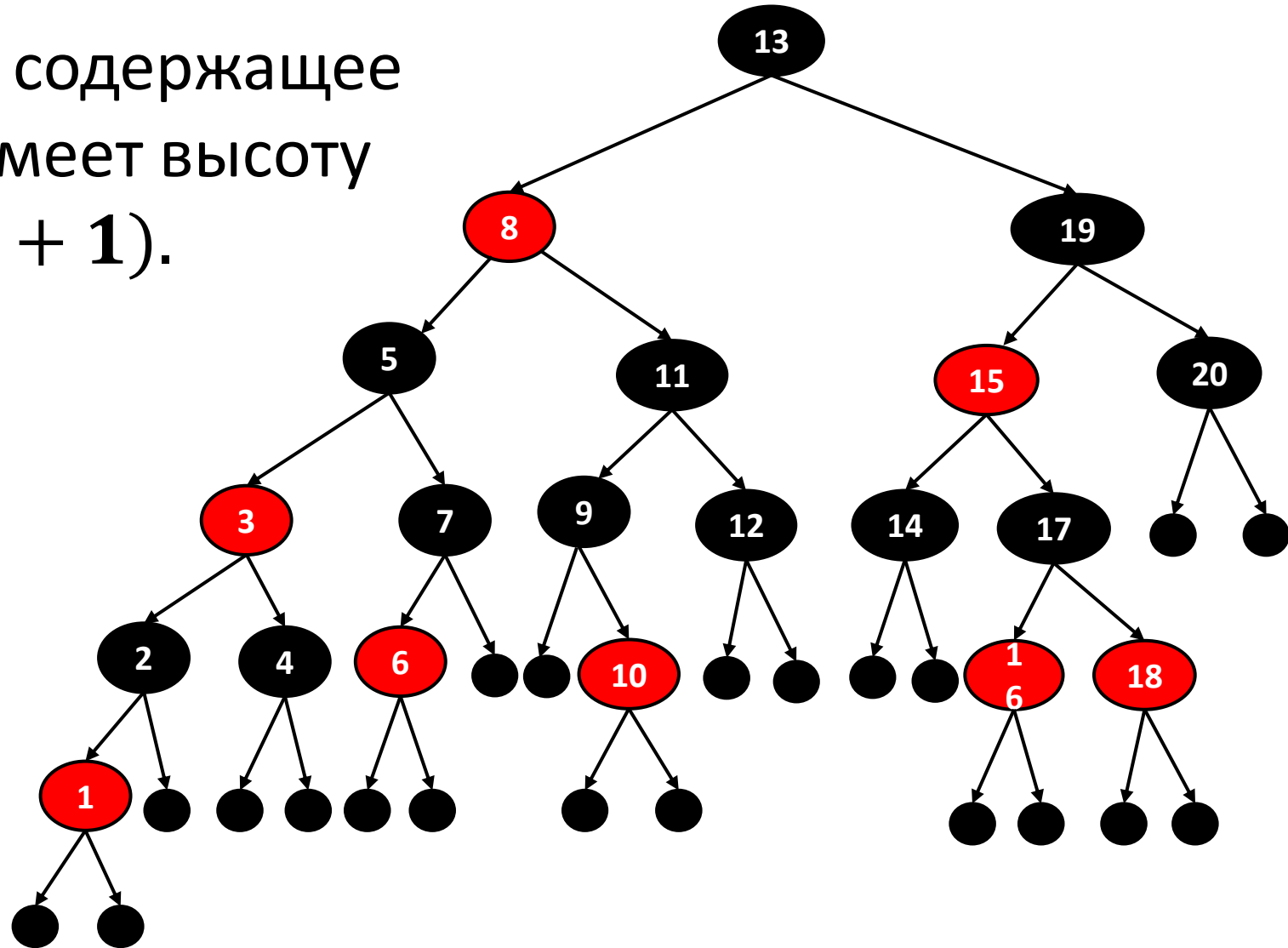
Для любой вершины v справедливо:

$$bh(v) \leq LengthPath(v, leaf) \leq 2 \cdot bh(v),$$

$$bh(v) \leq h(v) \leq 2 \cdot bh(v)$$

Теорема

Красно-чёрное дерево, содержащее n внутренних вершин, имеет высоту $h(T) \leq 2 \cdot \log(n + 1)$.

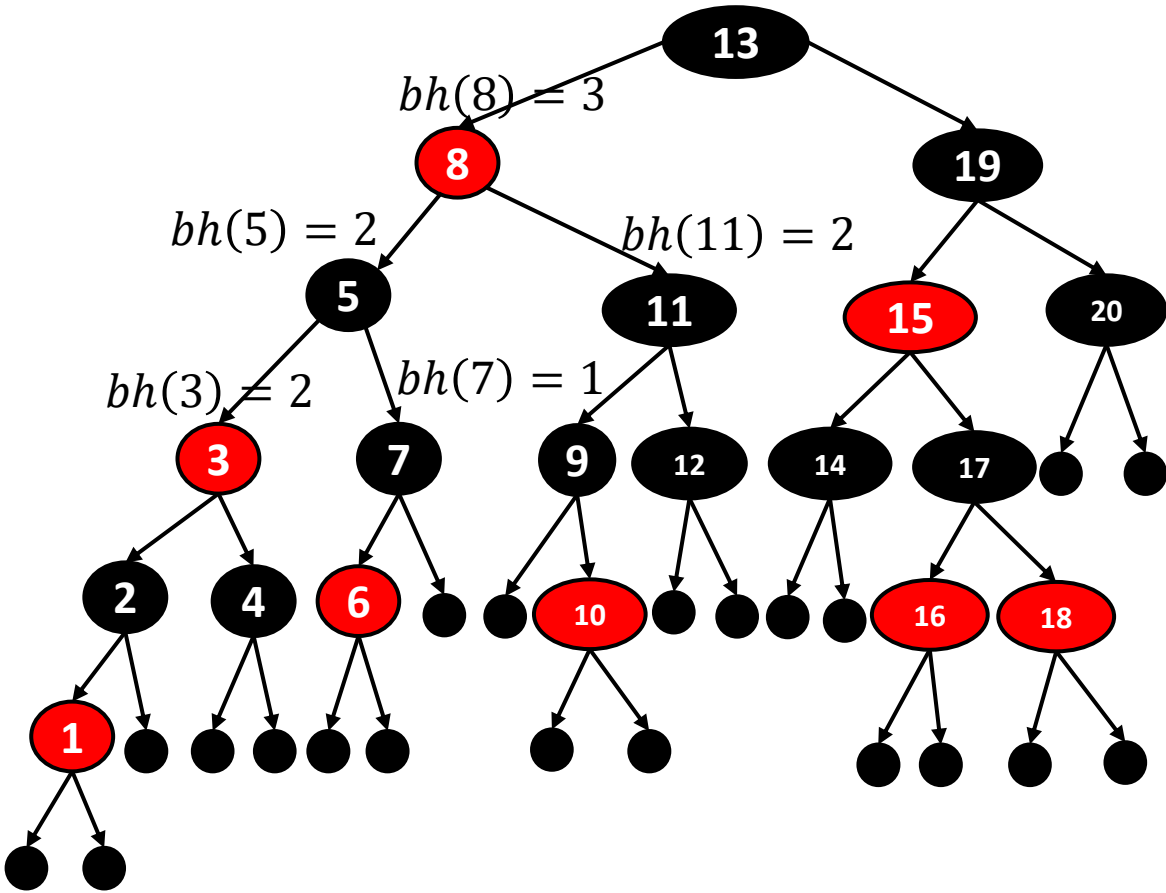


Доказательство

1) Сначала, используя метод математической индукции (по высоте h), докажем, что поддерево с корнем в вершине v содержит как минимум $(2^{bh(v)} - 1)$ внутренних вершин.

База индукции. Если вершина v — лист, то $bh(v) = 0$ и $2^{bh(v)} - 1 = 2^0 - 1 = 0$.

Шаг индукции.
Предположим, что для всех вершин дерева высоты $\leq h$ свойство выполняется.
Рассмотрим вершину v высоты $h + 1 > 0$. Вершина v — внутренняя вершина дерева, значит у неё есть оба поддерева. Если у вершины v чёрная высота $bh(v)$, то её сыновья могут иметь чёрную высоту либо $(bh(v) - 1)$ (если сын — чёрный), либо $bh(v)$ (если сын — красный).
С учётом индукционного предположения для сыновей вершины v получим, что число внутренних вершин поддерева с корнем в вершине v :

$$\geq 2 \cdot (2^{bh(v)-1} - 1) + 1 = 2^{bh(v)} - 1.$$


Доказательство (продолжение)

2) Теперь рассмотрим произвольный путь из корня дерева в лист. По крайней мере половина вершин на пути, не считая сам корень, должны быть чёрными.

Следовательно

$$bh(T) \leq h(T) \leq 2 \cdot bh(T)$$

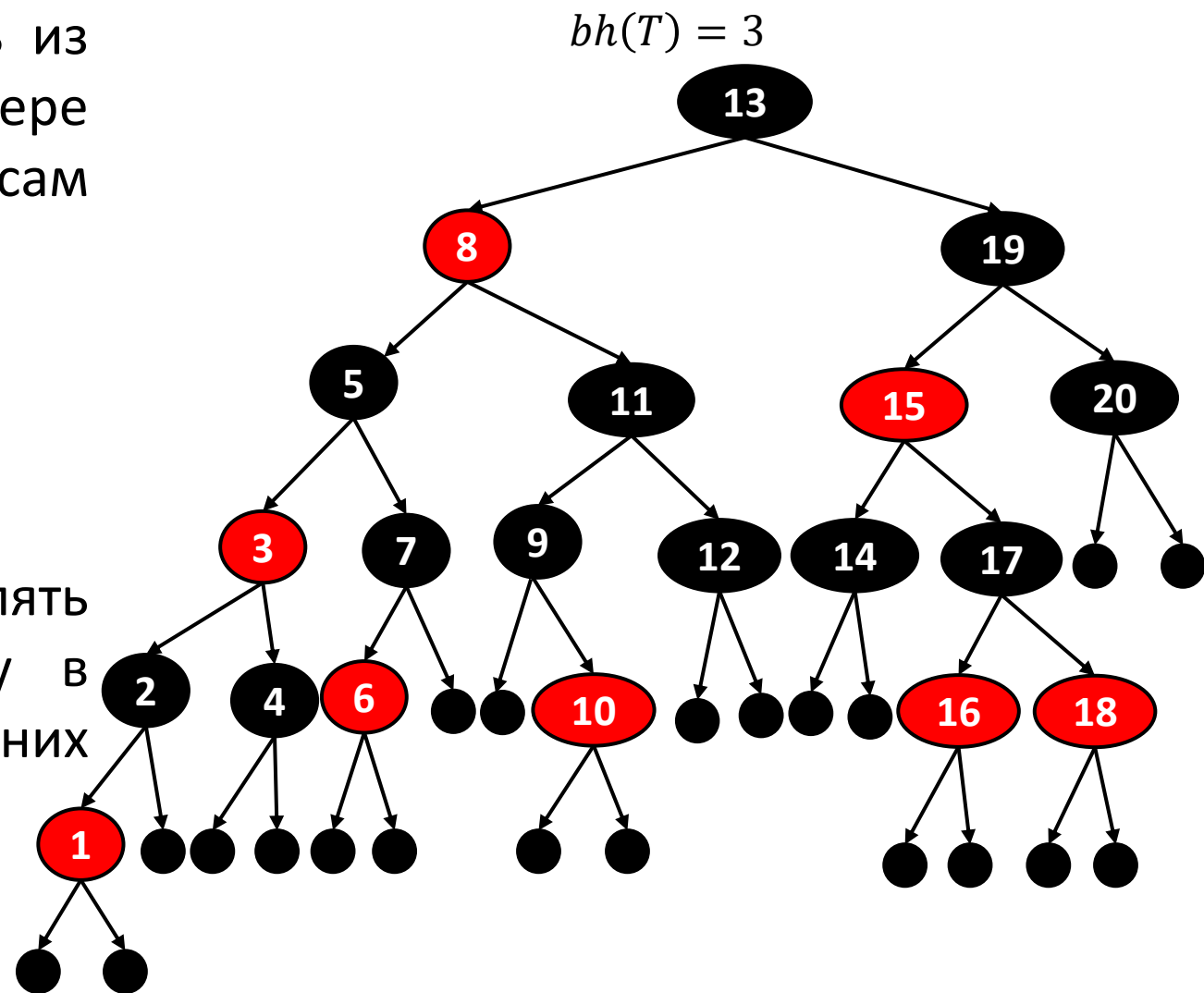
$$\boxed{\frac{h(T)}{2} \leq bh(T) \leq h(T)},$$

т.е. чёрная высота корня должна составлять как минимум $h(T)/2$ и по доказанному в пункте 1) свойству, для числа внутренних вершин дерева справедливо неравенство:

$$n \geq 2^{bh(T)} - 1 \geq 2^{\frac{h(T)}{2}} - 1.$$

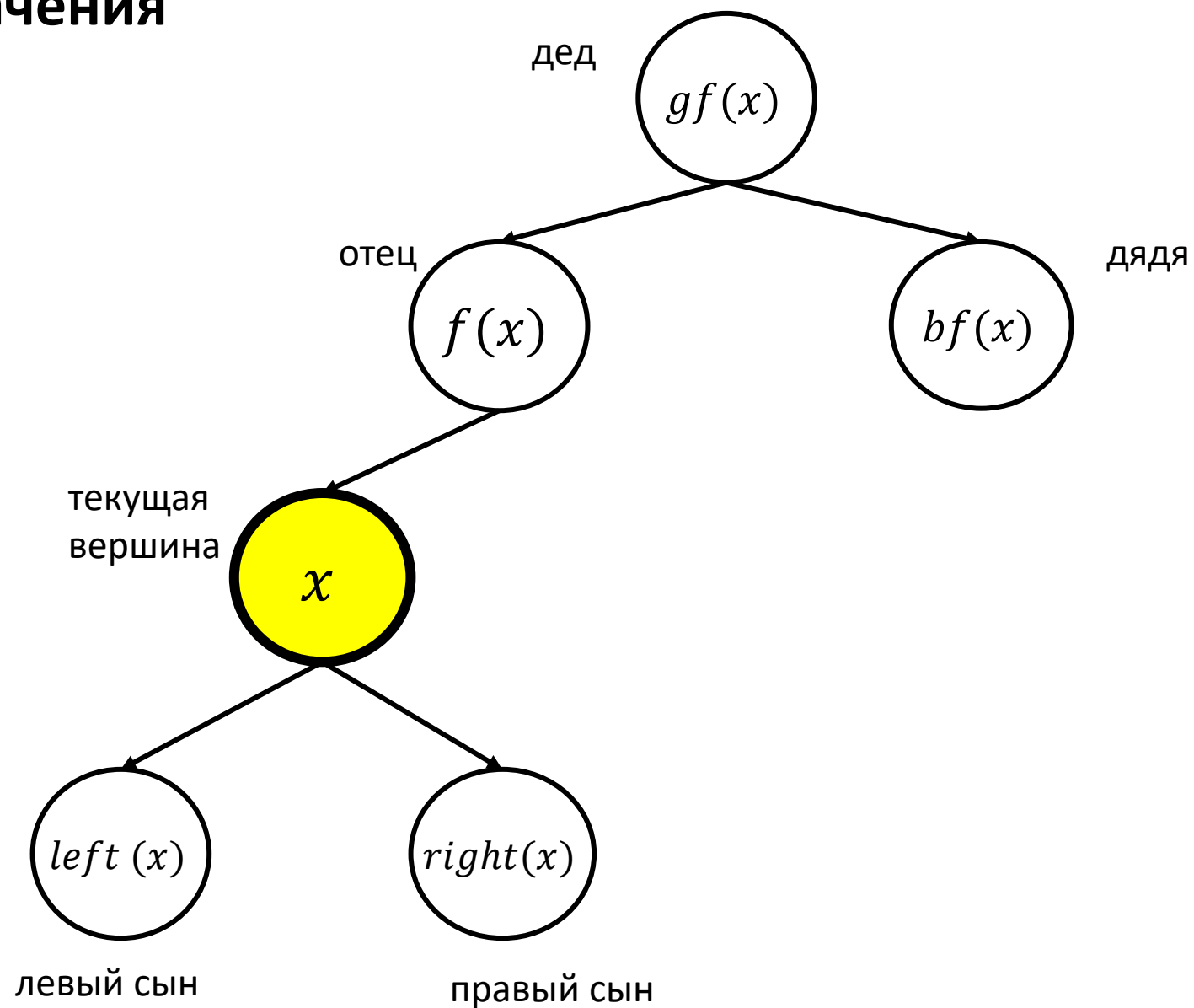
Логарифмируем неравенство и получаем утверждение теоремы:

$$h(T) \leq 2 \cdot \log_2(n + 1).$$

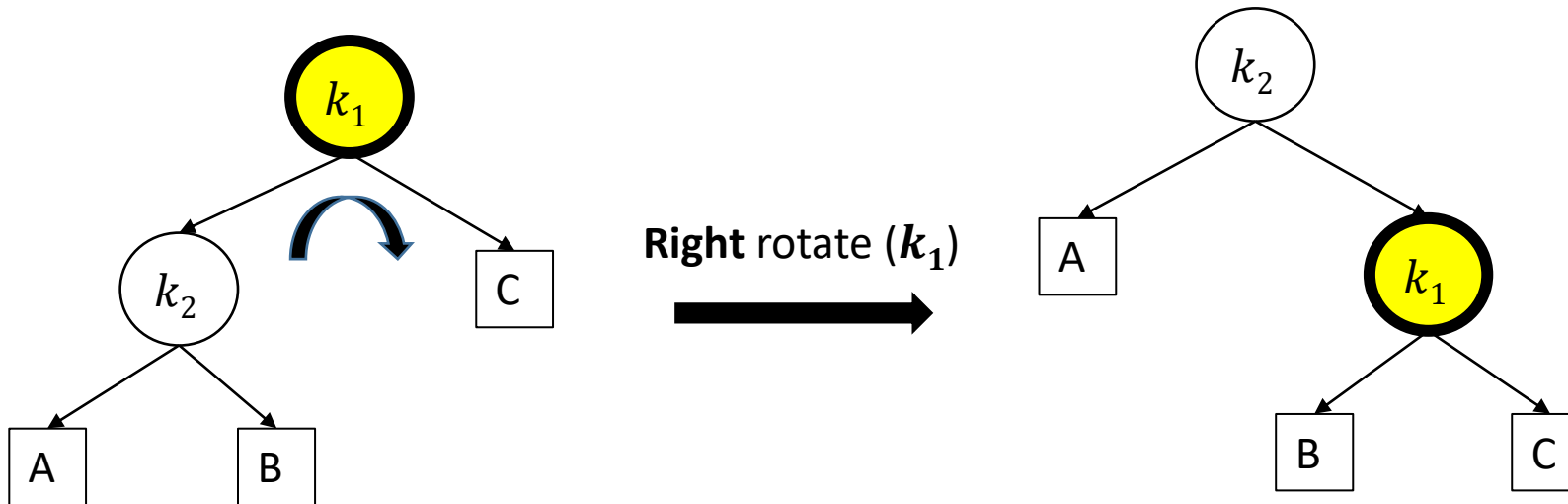
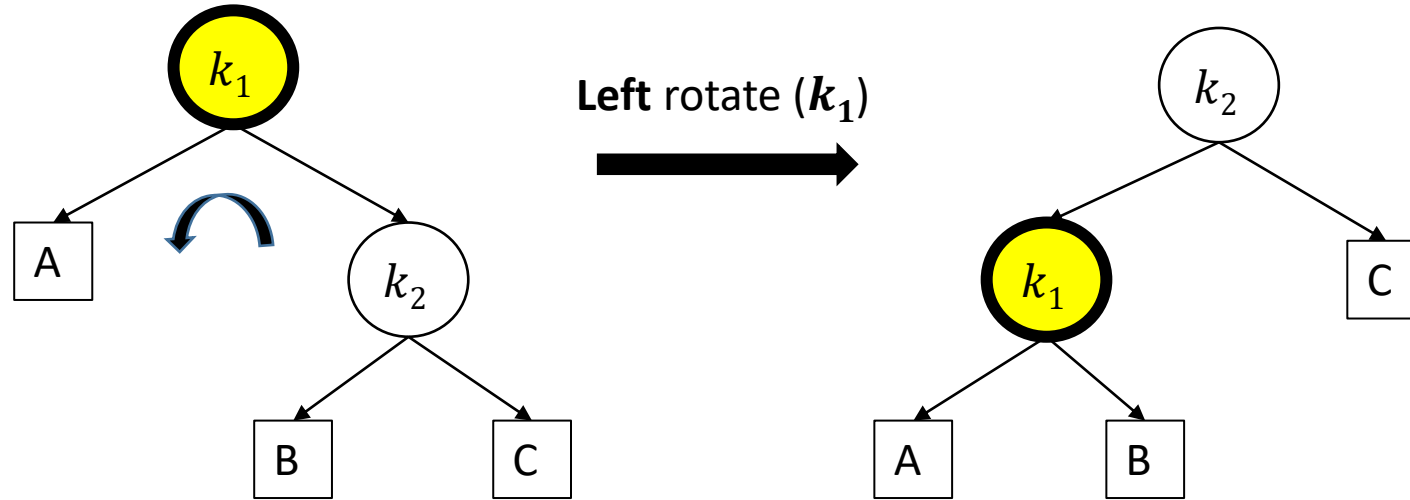


Доказательство завершено.

Обозначения



Для поддержки свойств красно-чёрных деревьев выполняются вращения, каждое из которых выполняется за $O(1)$:

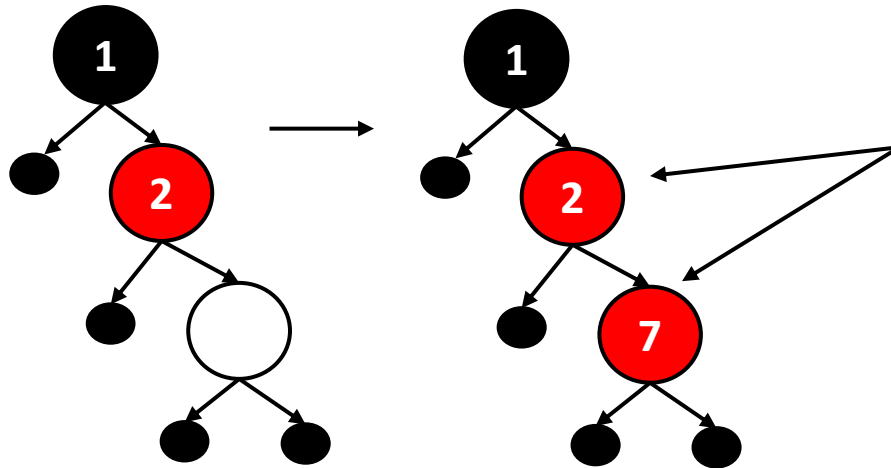
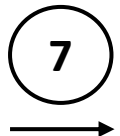
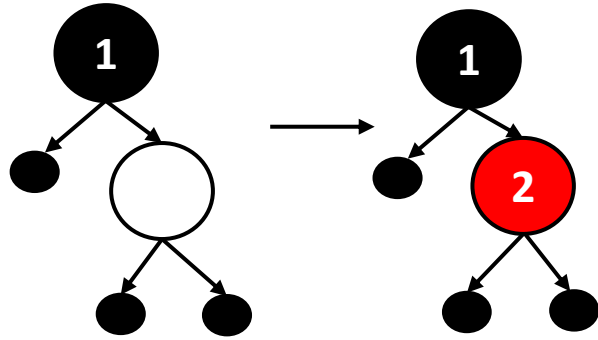
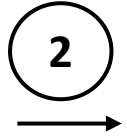
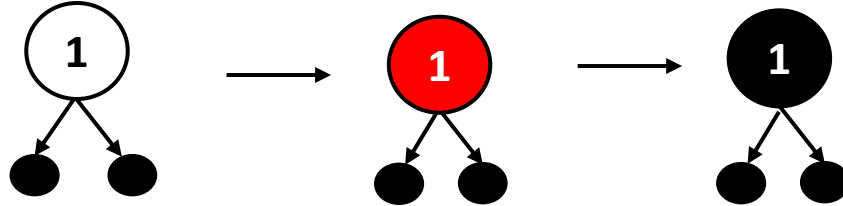
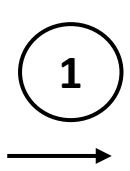


Добавление вершины в дерево

1. Осуществляем поиск места для добавляемой вершины x по аналогии с тем, как это делали в бинарном поисковом дереве.
2. Добавляем вершину x на место чёрного фиктивного листа и добавляем ей в качестве сыновей две фиктивные чёрные вершины.
3. Красим вершину x в красный цвет.
4. Если после добавления произошло нарушение RB-свойства (может нарушиться только одно: появились две подряд идущие красные вершины), то выполняем процедуру, восстанавливающую RB-свойства: перекраска вершин и, возможно, не более двух поворотов.

Пример.

Для последовательности чисел: 1, 2, 7, 3, 8, 14, 9 построить RB-дерево.



появились две подряд идущие красные вершины – нарушение RB- свойства

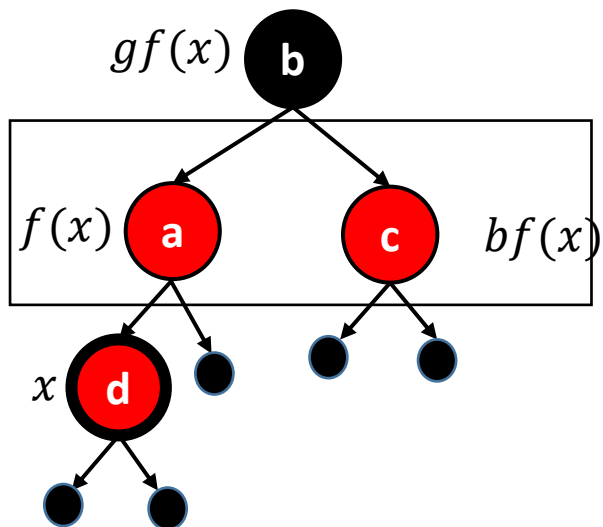
Процедуры, восстанавливающие RB-свойства:

(1) перекраски

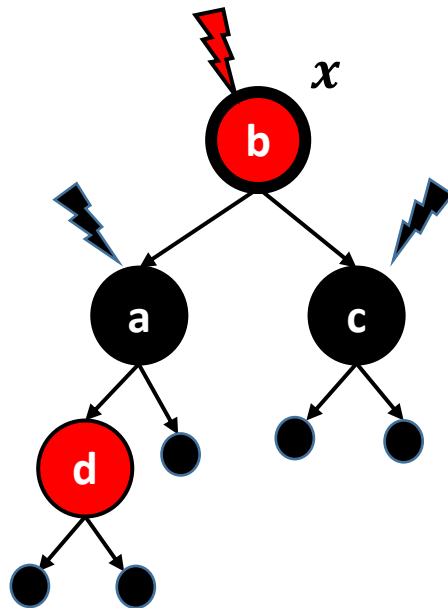
(2) вращения

1-й случай:

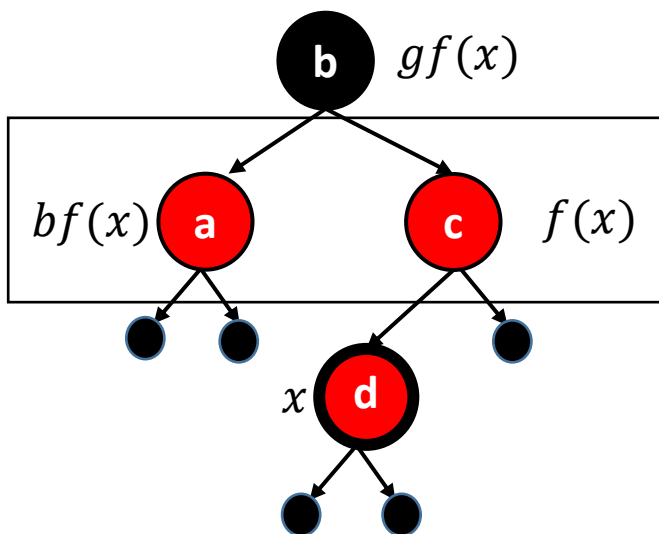
отец и дядя - красные



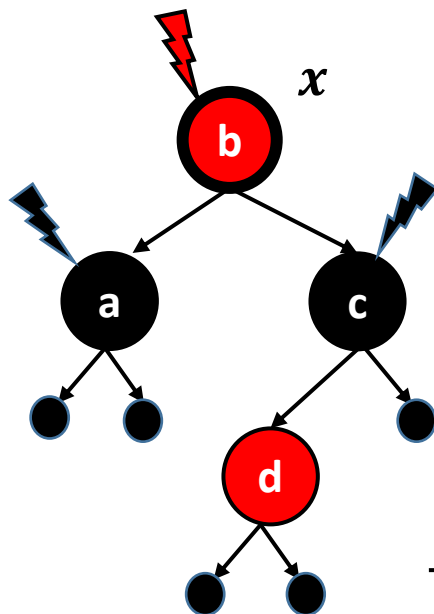
перекраска



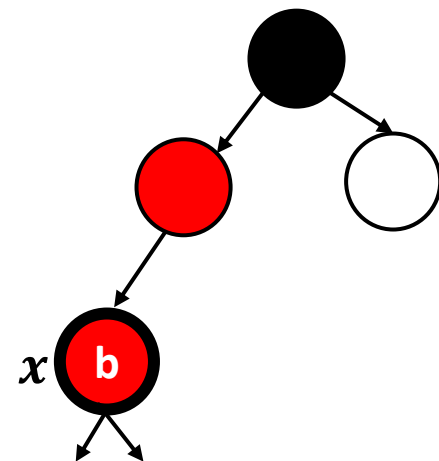
продолжаем
рекурсивно
восстановление
RB-свойства



перекраска



RB-свойства
восстановлены

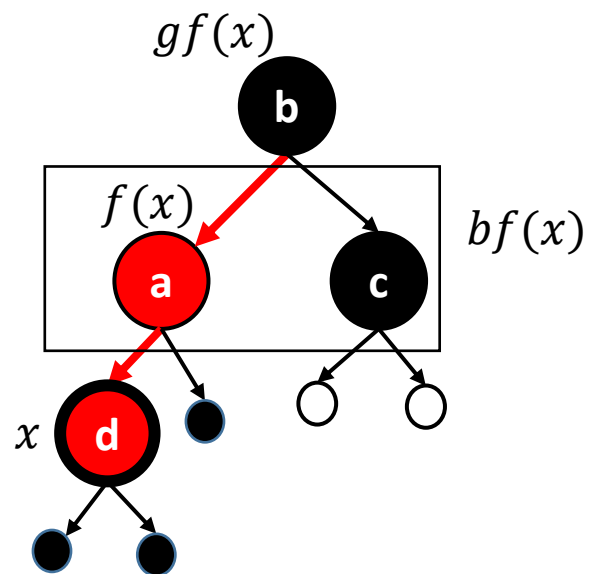


случай 1 или 2

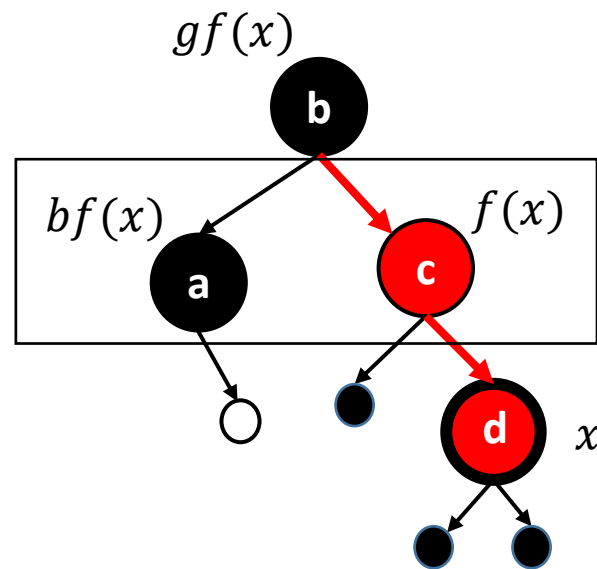
2-й случай:

отец – **красный** , дядя – **чёрный**

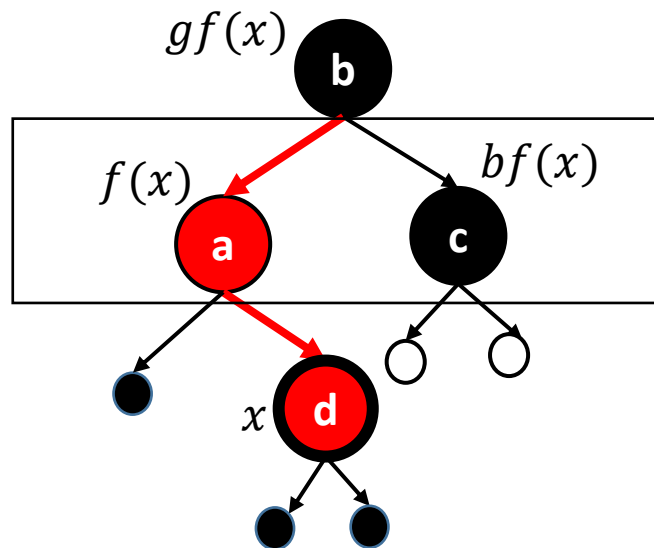
а)



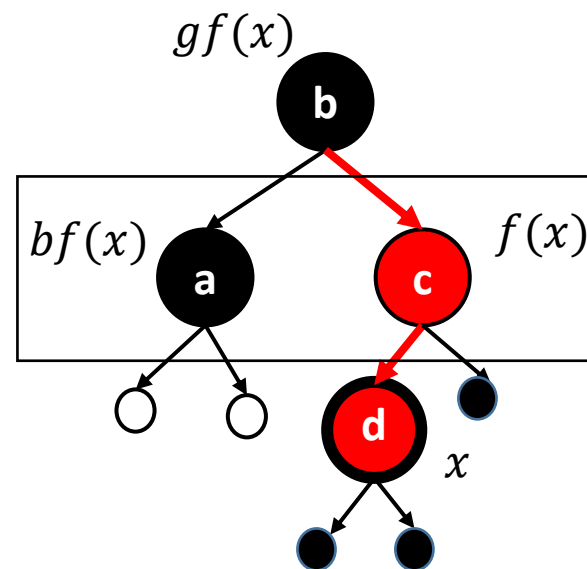
б)



в)



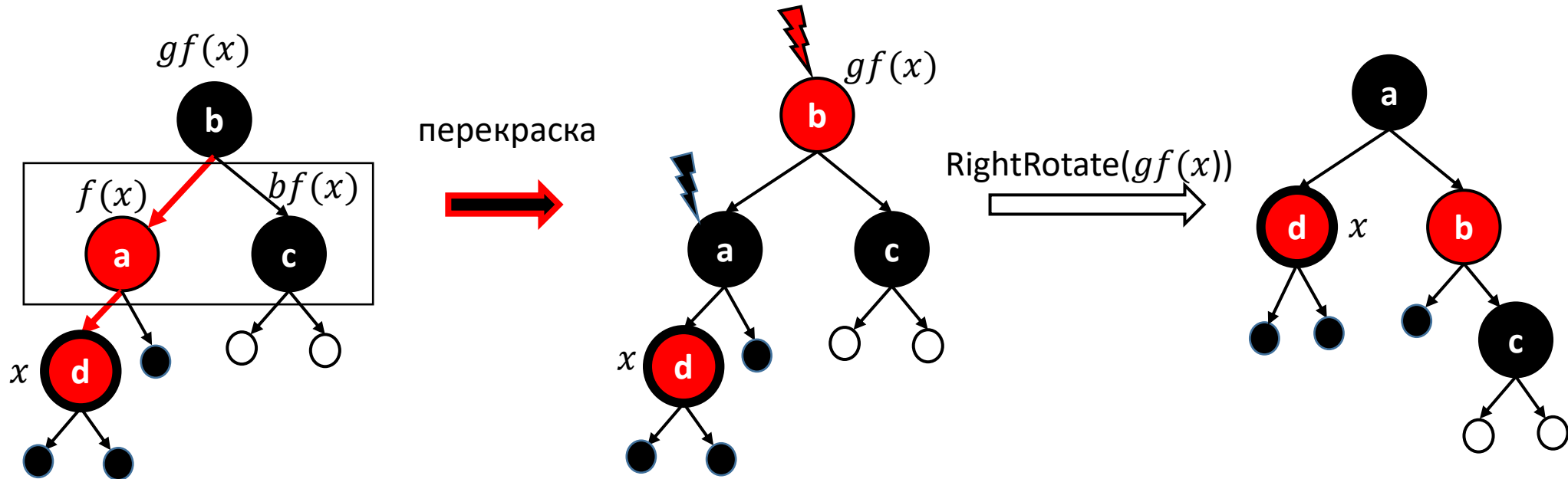
г)



2-й случай:

отец – **красный** , дядя – чёрный

a)

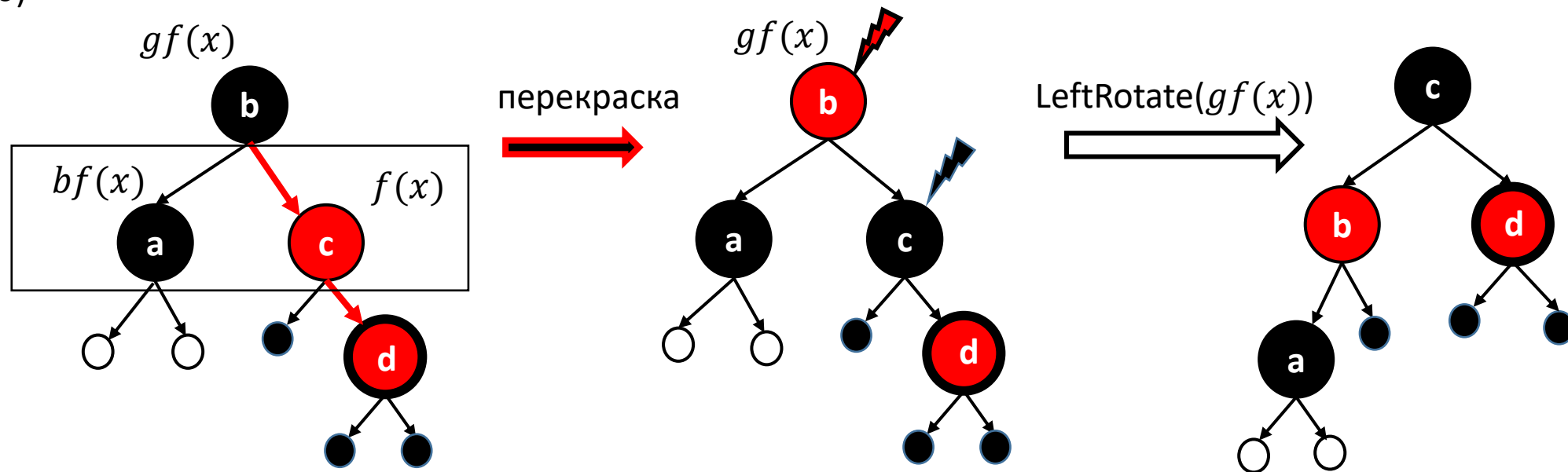


RB-свойство
восстановлено

2-й случай:

отец – **красный** , дядя – чёрный

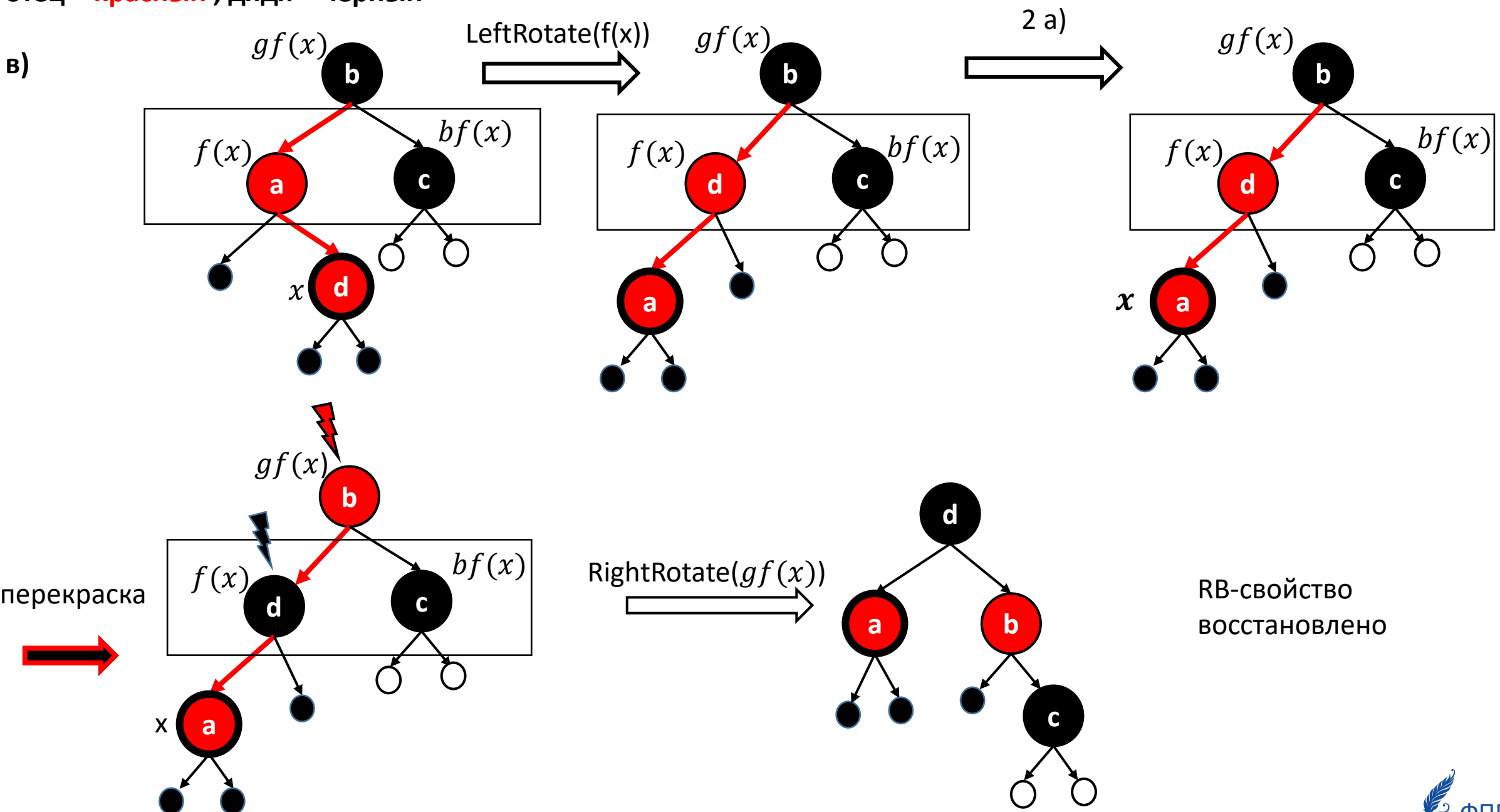
б)



RB-свойство
восстановлено

2-й случай:
отец – **красный** , дядя – **чёрный**

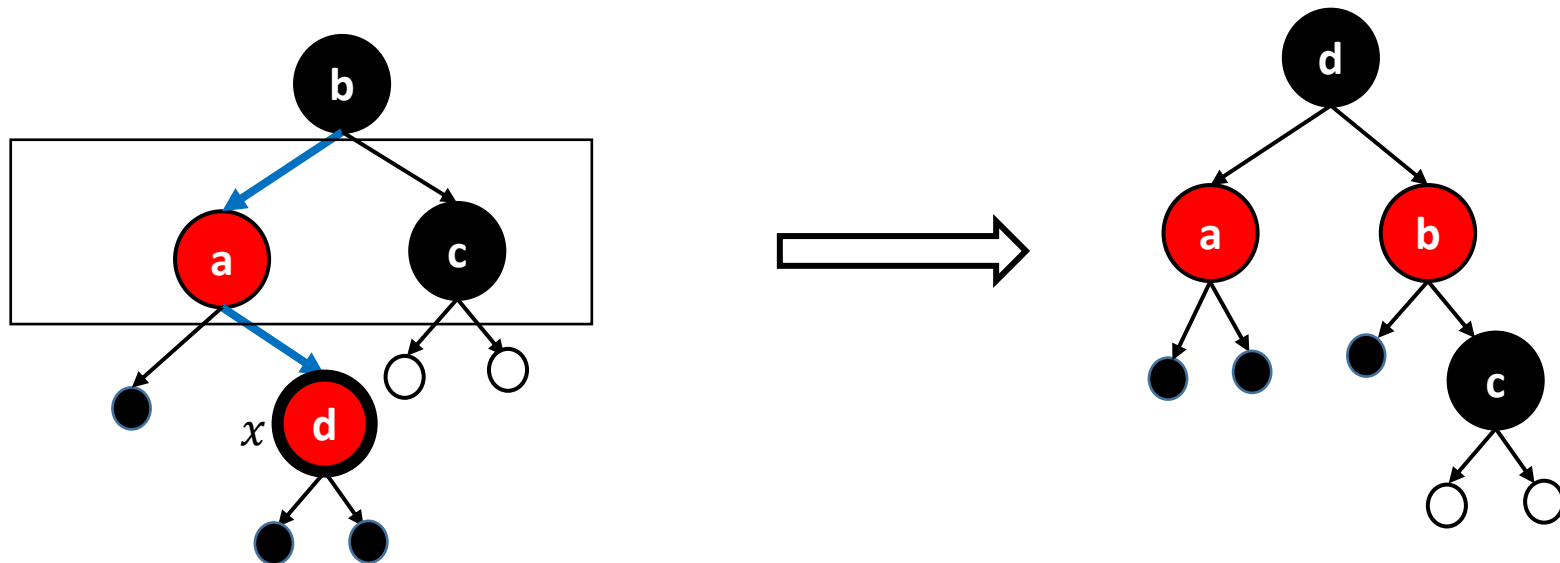
в)



2-й случай:

красный , дядя – чёрный

в) ИТОГ

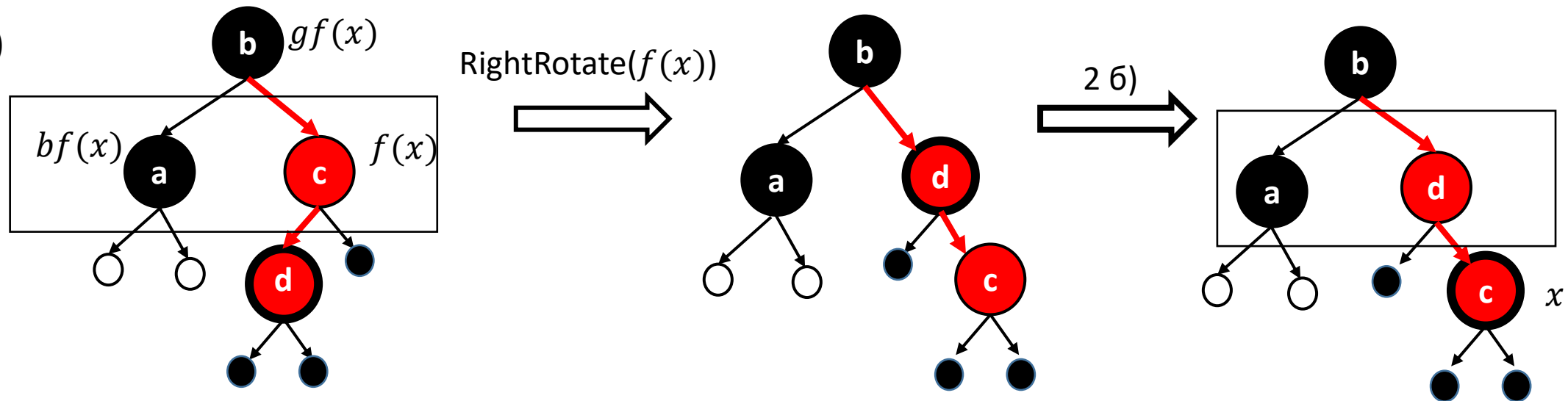


RB-свойство
восстановлено

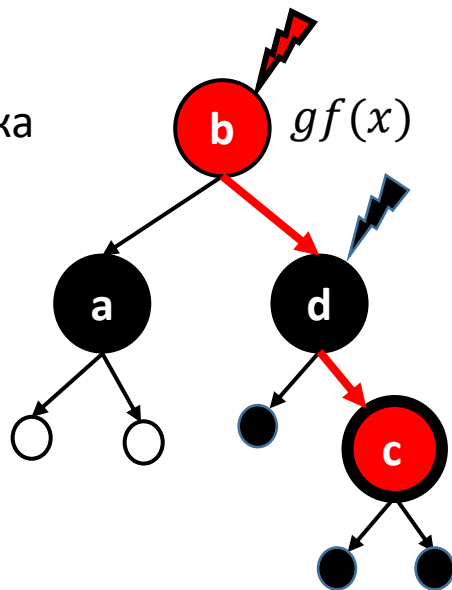
2-й случай:

отец — **красный** , дядя — **чёрный**

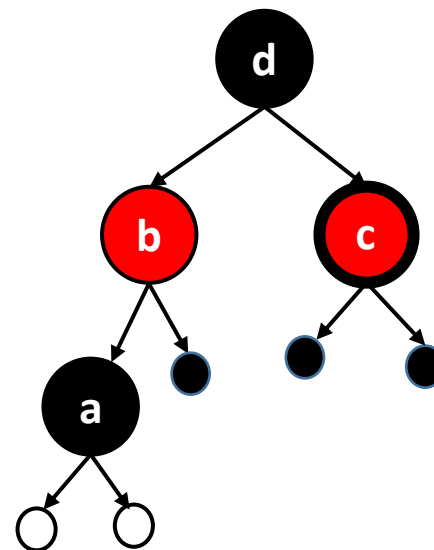
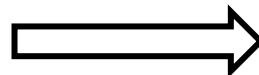
г)



перекраска



$\text{LeftRotate}(gf(x))$

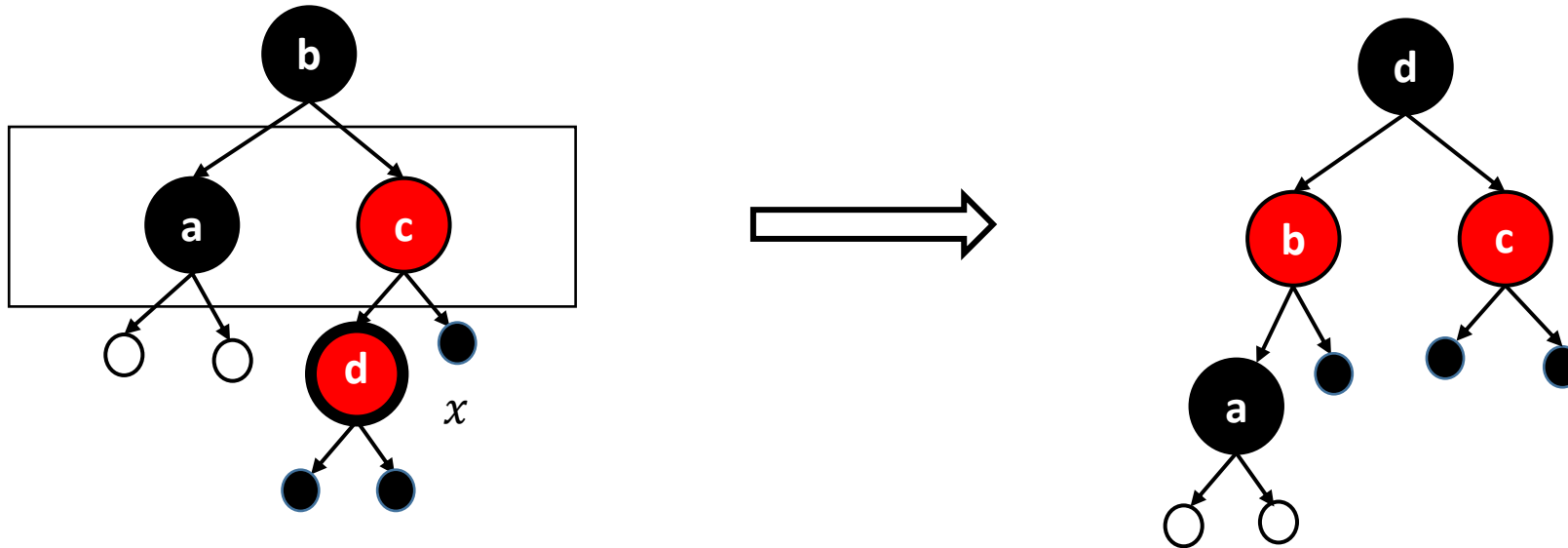


RB-свойство
восстановлено

2-й случай:

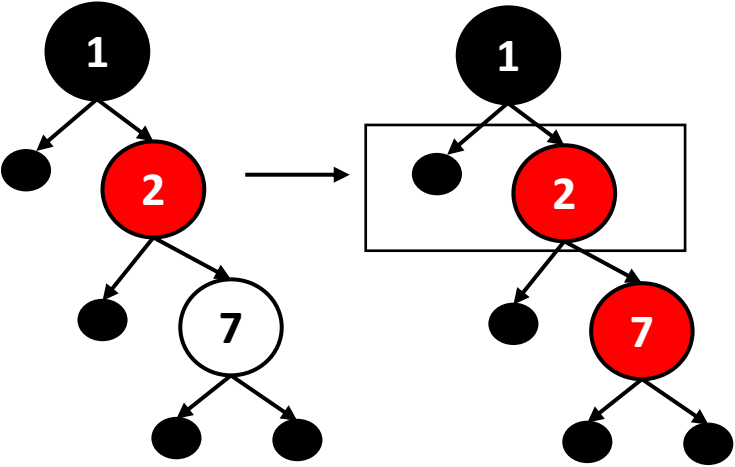
отец – **красный** , дядя – чёрный

г) ИТОГ

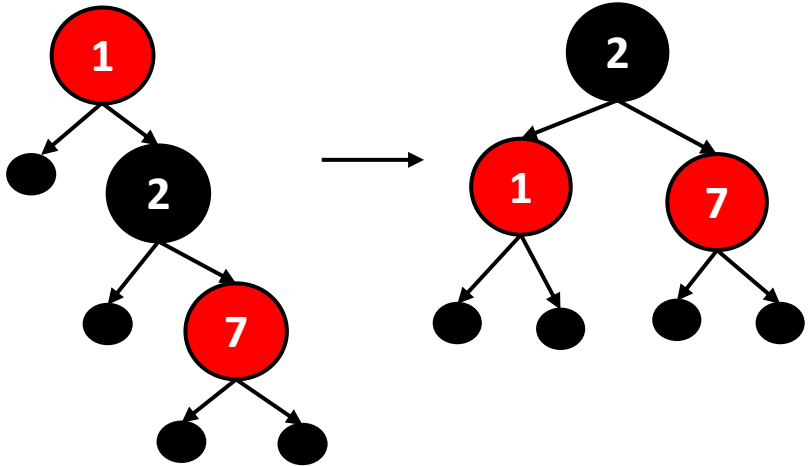


Пример (продолжение). Для последовательности чисел: 1, 2, 7, 3, 8, 14, 9 построить RB-дерево.

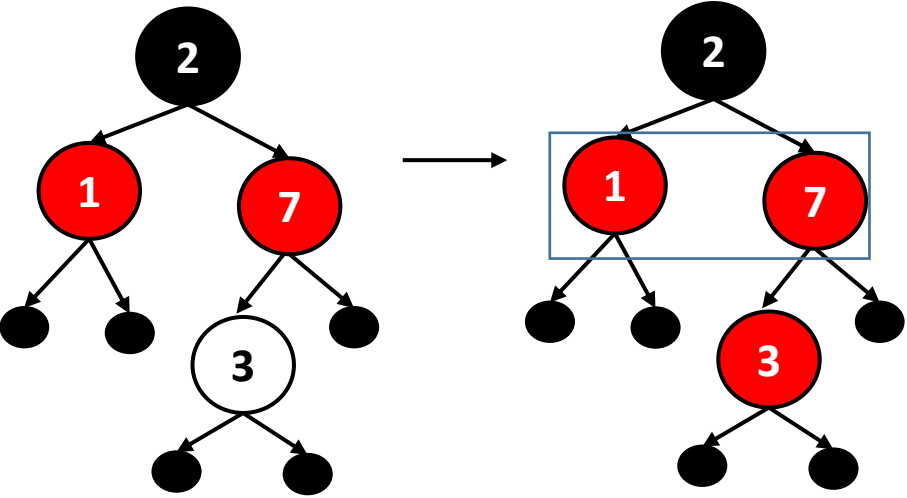
7
→



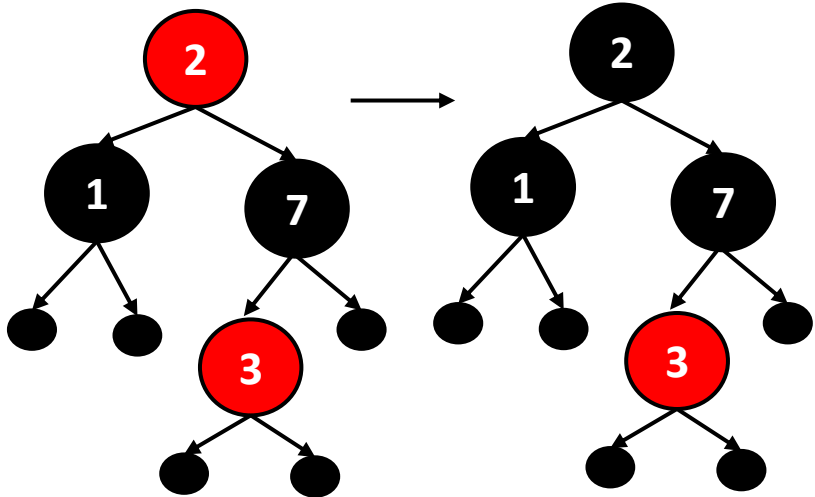
Случай 2 б)



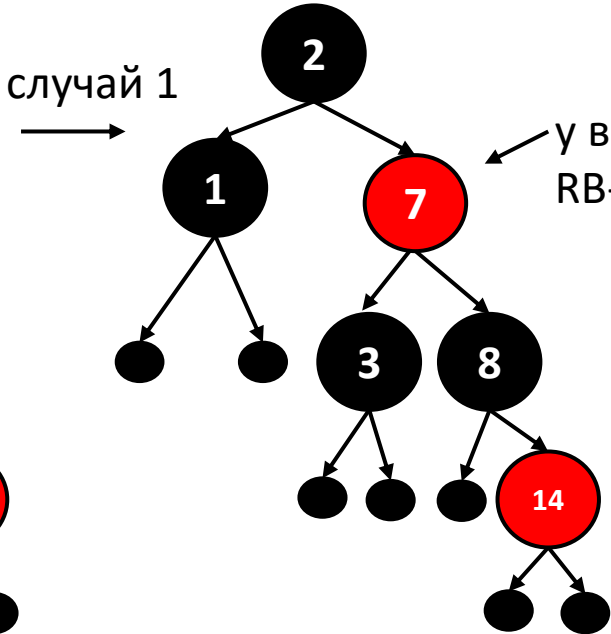
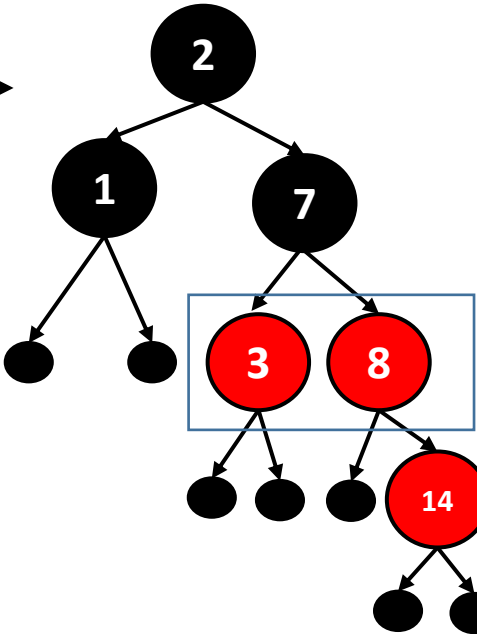
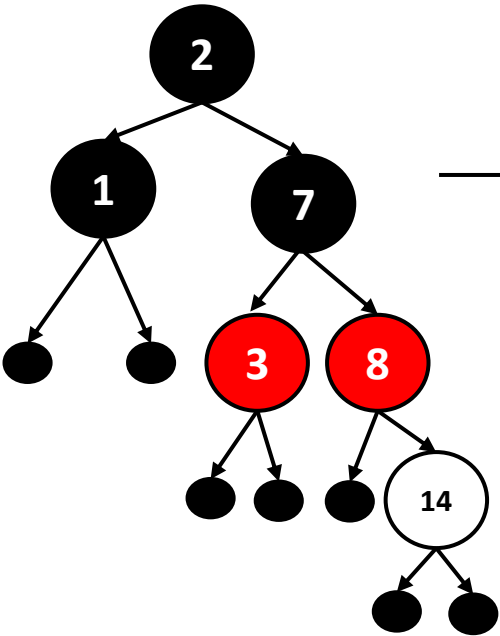
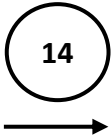
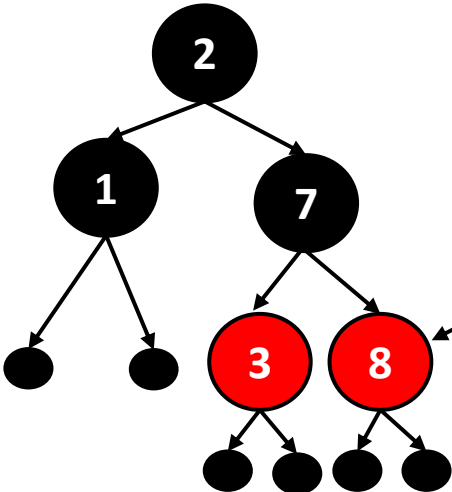
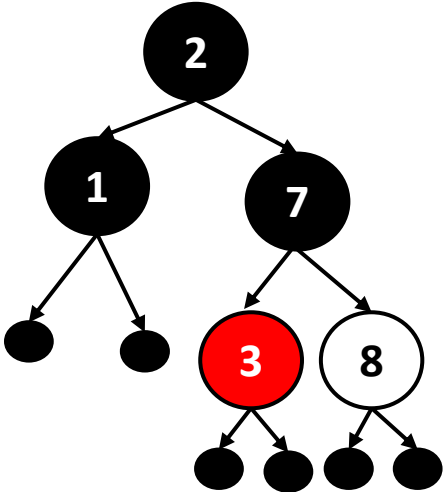
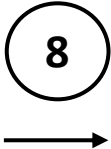
3
→



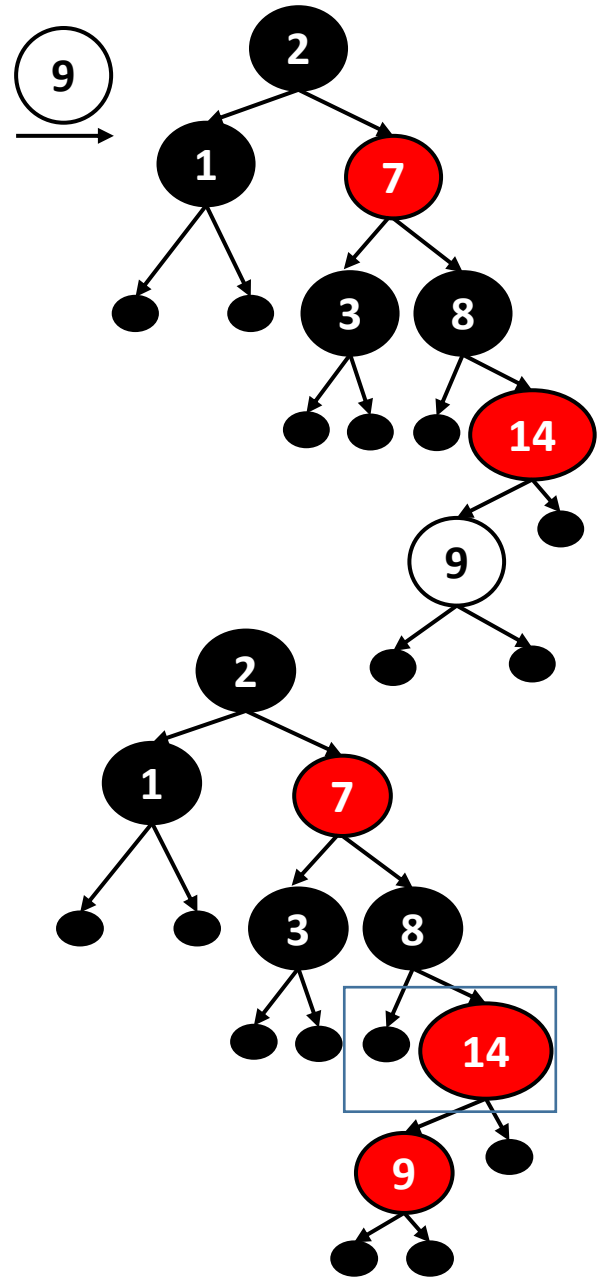
Случай 1



(продолжение)

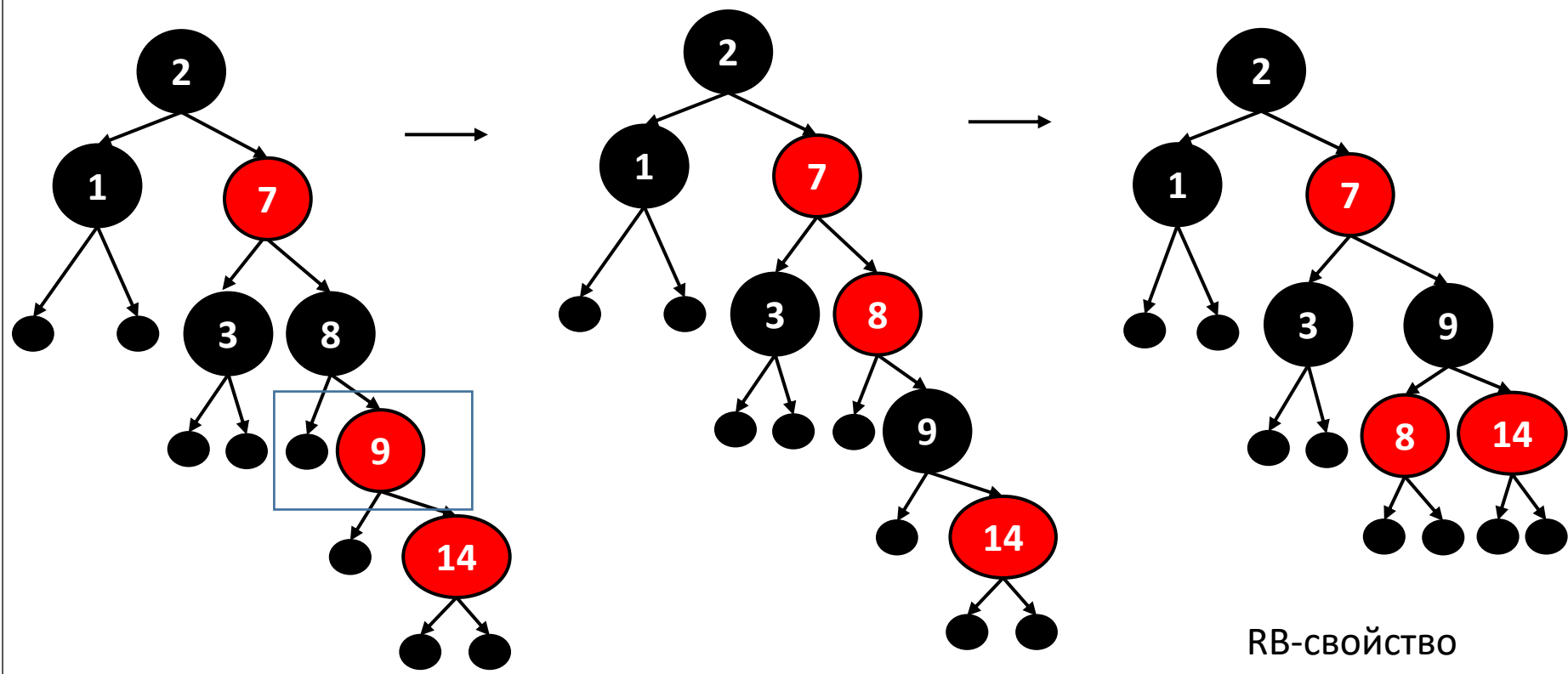


(продолжение)



случай 2 г)

восстановление RB-свойства



RB-свойство
восстановлено

Добавление элемента

- поиск отца – $O(\log n)$
- добавление – $O(1)$
- перекрашивания – $O(\log n)$
- повороты (не более 2-х) – $O(1)$

Следовательно, время добавления элемента - $O(\log n)$.

Удаление

1. Удаляем вершину из дерева по аналогии с тем, как это делали в бинарном поисковом дереве.
2. Пусть y – фактически удалённая вершина.
3. Если удалённая вершина y имела **красный цвет**, то все RB-свойства будут выполняться и операция удаления элемента завершена.
4. Если удалённая вершина y имела **черный цвет**, то любой путь, через неё проходивший, теперь содержит на одну чёрную вершину меньше. Нарушается RB-свойство, которое требует, чтобы любой путь из корня в листья содержал одинаковое количество чёрных вершин. Восстановим RB-свойство.

Обозначения

- y – фактически удалённая вершина
- x – единственный ребёнок вершины y (если детей y вершины y не было, то $x = NULL$)
- $f(x) = f(y)$
- вершину, которая может быть окрашена, как в красный, так и в чёрный цвет, будем обозначать на рисунках r/b

Если фактически удалённая вершина **у** имела **черный цвет**, то любой путь, через неё проходивший, теперь содержит на одну чёрную вершину меньше.

Поступим следующим образом:

- если вершина **х** - **красная**, то сделаем её **чёрной**, теперь все RB-свойства выполнены;
- если **х** - **чёрная**, то, будем при подсчёте числа чёрных вершин на пути от корня к листьям **считать её за две чёрные вершины**;



однако дерево не предполагает такие «двойные чёрные вершины», поэтому нужно выполнить процедуру, которая превращает полученное дерево в настоящее красно-чёрное дерево.

1-й случай

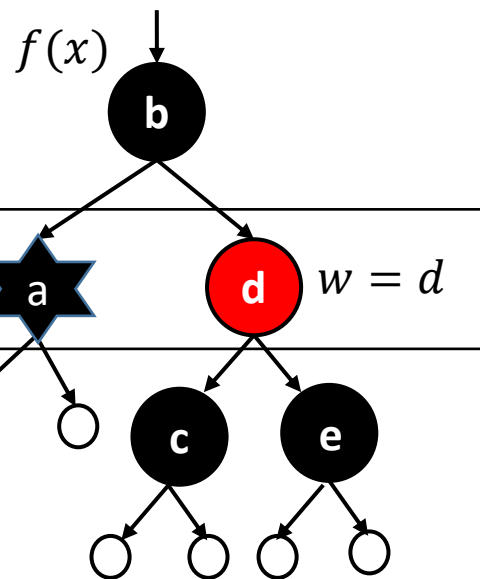
- ✓ x – чёрный и является левым сыном своего отца (ситуация правого сына выполняется симметрично),
- ✓ брат w – **красный**

случай 2 перекраска и завершение

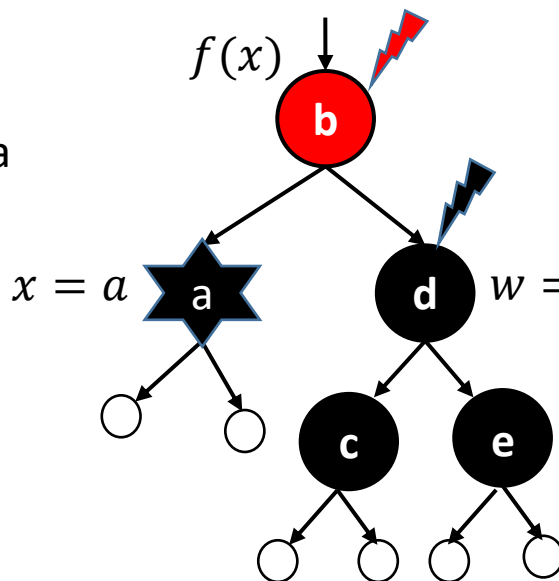
случай 3 перекраска и поворот

случай 4 (перекраска, поворот, завершение)

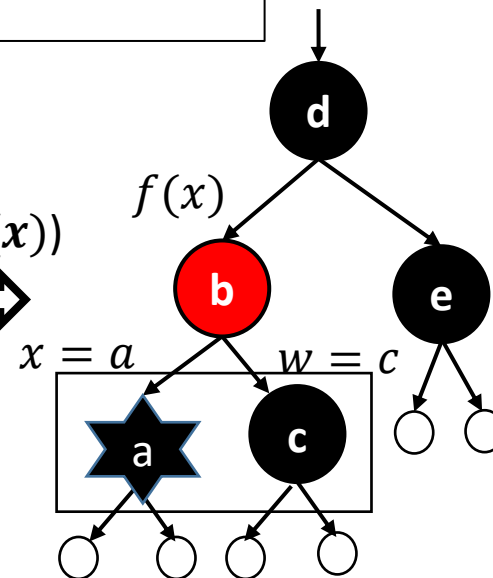
случай 4 (перекраска, поворот, завершение)



перекраска



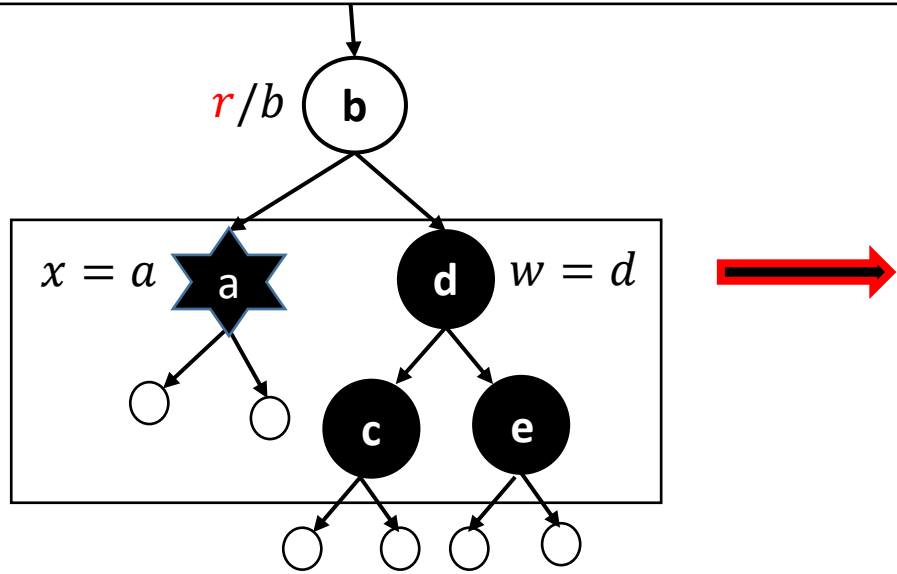
LeftRotate($f(x)$)



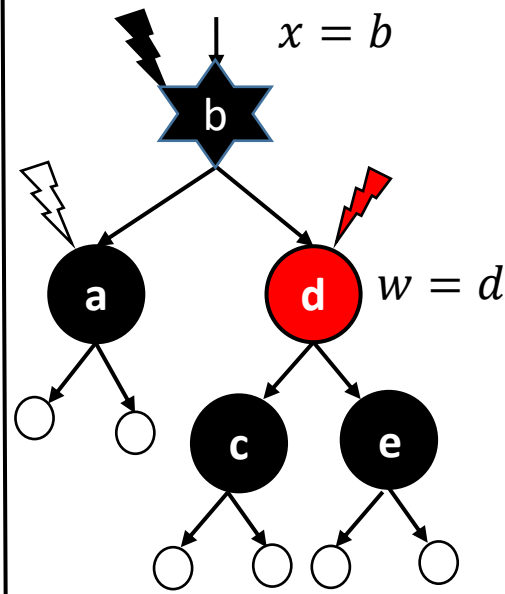
- 1) «новый брат» w вершины x – **чёрный**
- 2) вершину x считаем «**дважды чёрной**»,
далее рассматриваются случаи в зависимости от того, какого цвета дети у вершины w (новый брат x)

2 случай (возможно повторение)

- ✓ x – «дважды чёрный» и является левым сыном своего отца,
- ✓ $w, left(w), right(w)$ – чёрные

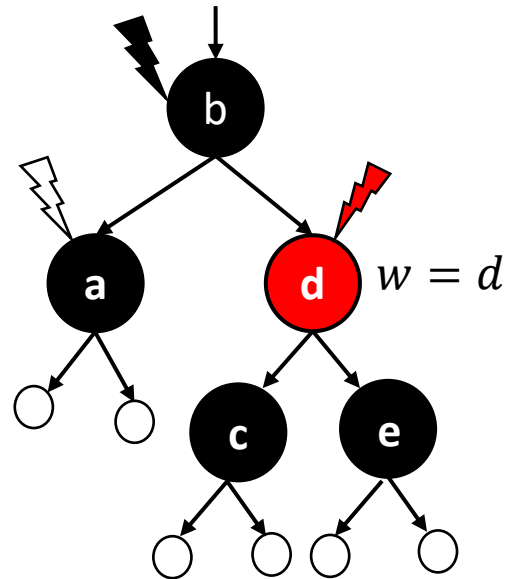


⚡ снятие «лишней» черной окраски



если вершина b была раньше чёрная, то она становится «дважды чёрной»;

продолжаем балансировку для вершины $x = b$



если вершина b была раньше красная, то она становится чёрной;

RB-свойства выполнены;

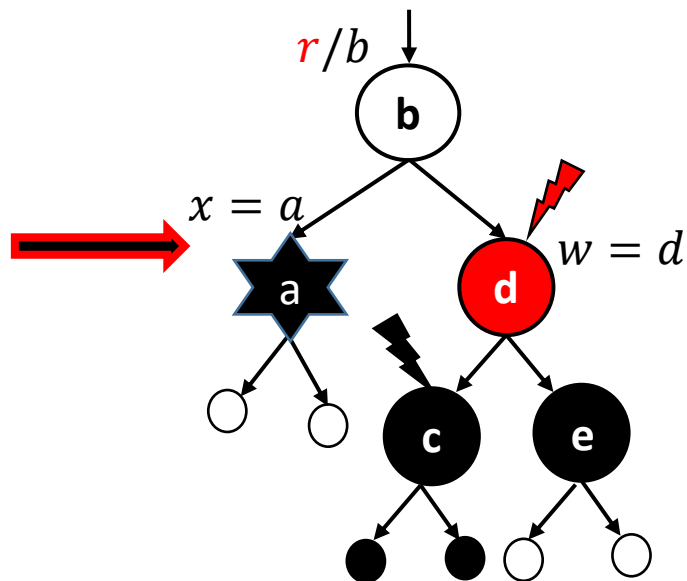
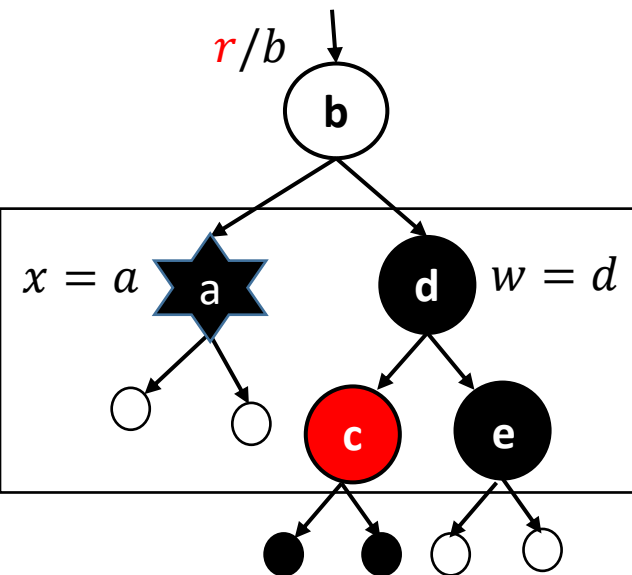
3-й случай

- ✓ x – «дважды чёрная»
- ✓ $w, right(w)$ – чёрная
- ✓ $left(w)$ – красная

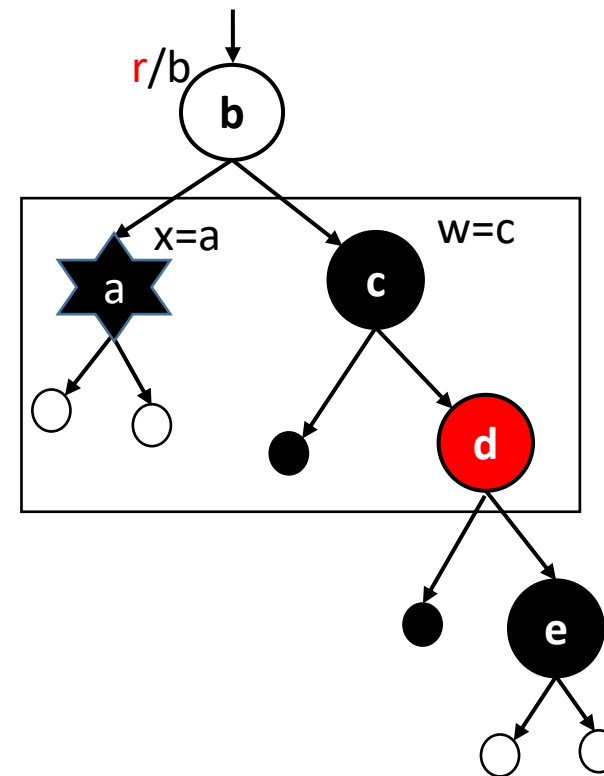
4-й случай

- ✓ x – «дважды чёрная»
- ✓ $w, left(w)$ – чёрная
- ✓ $right(w)$ – красная

завершение



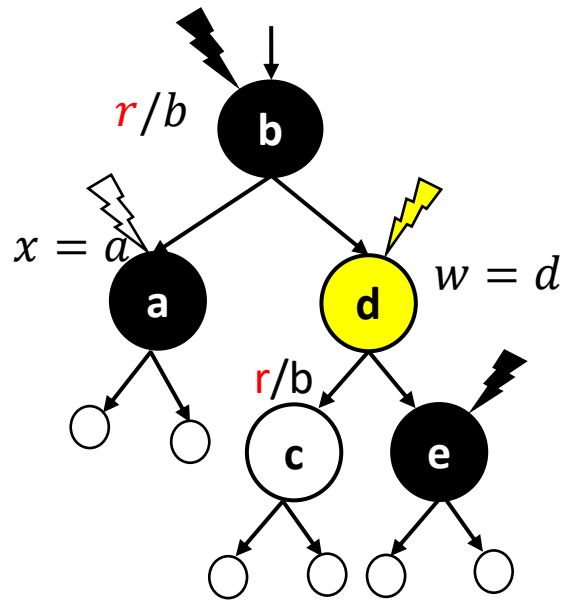
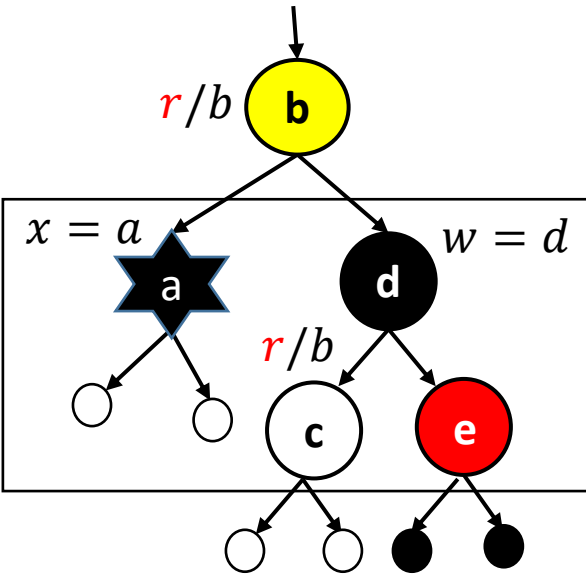
RightRotate(w)



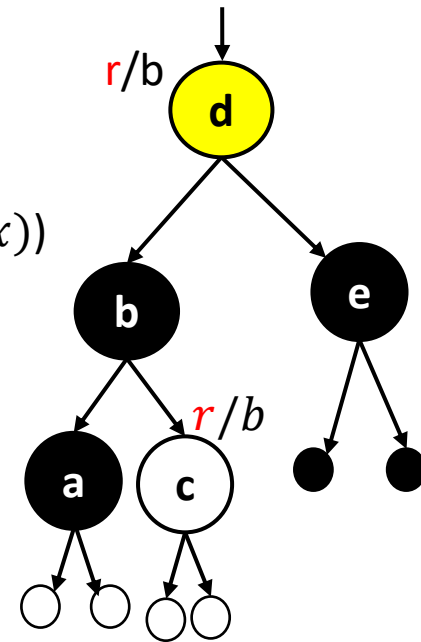
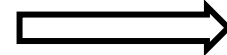
4 случай

- ✓ x – «дважды чёрная»
- ✓ $w, \text{left}(w)$ – чёрная
- ✓ $\text{right}(w)$ – красная

завершение



LeftRotate($f(x)$)

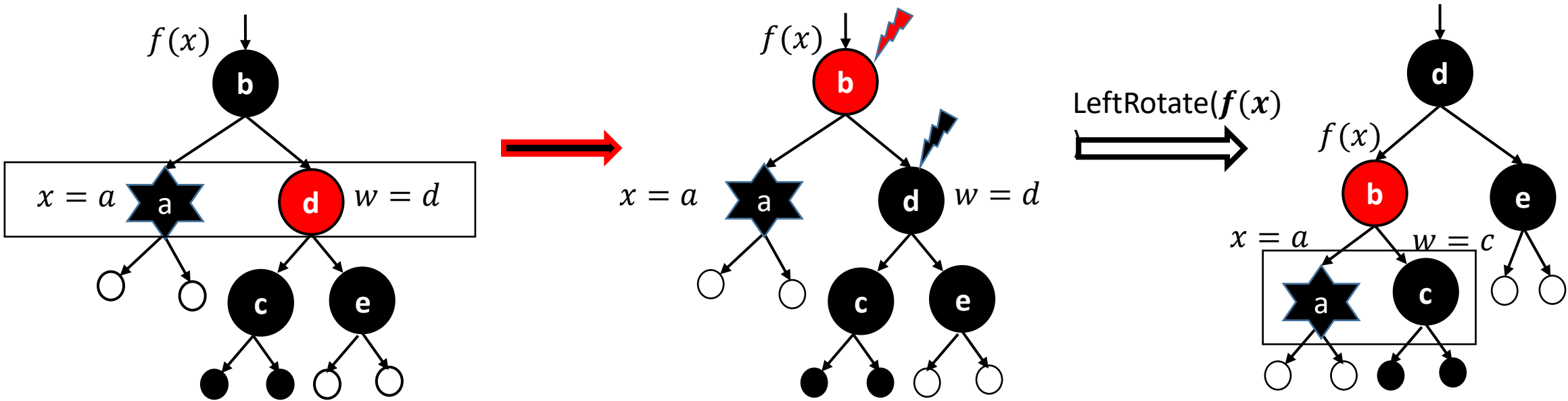


вершина d красится в тот же цвет,
который был изначально у $f(x)$ (вершины b)

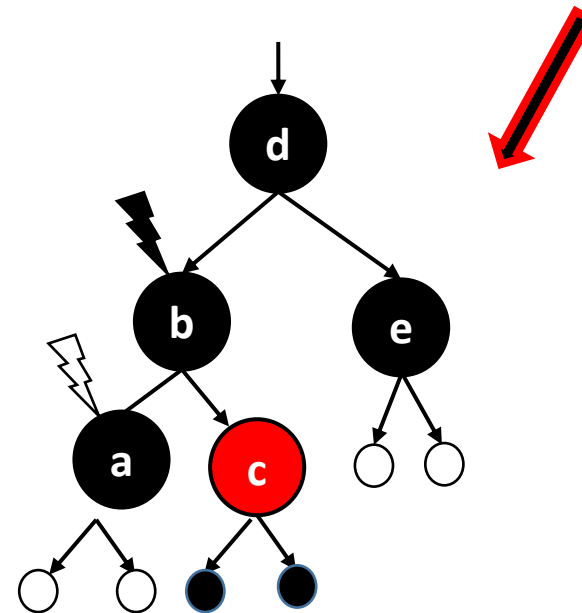
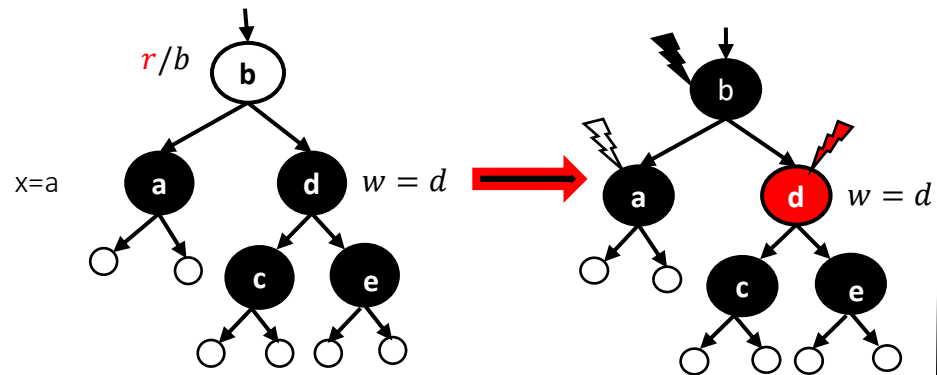
RB – свойства выполнены

1-й случай (продолжение) → если выполняется сведение к случаю 2 и завершение

✓ если у нового брата вершины x оба сына - чёрные



2-й случай



**RB -
свойства
выполнены**

Удаление

Случаи 1, 3 и 4.

Выполняются за время $O(1)$ (выполняется самое большое 3 вращения).

Случай 2.

При каждом выполнении этого случая цикл возможно продолжит свою работу. Одна итерация цикла выполняется за время $O(1)$, но при каждом повторении указатель на вершину x перемещается вверх по дереву, никакие вращения при этом не происходят, поэтому количество повторений случая 2 ограничено высотой дерева $h = O(\log n)$.

Таким образом время на восстановление RB-свойств после выполнения операции удаления элемента - $O(\log n)$.

Удаление

- поиск удаляемой вершины – $O(\log n)$
- непосредственное удаление вершины – $O(\log n)$
- все перекрашивания - $O(\log n)$
- повороты (будет выполнено не более 3-х) – $O(1)$

Следовательно, время удаления элемента - $O(\log n)$.

Высота

АВЛ

$$h < 1,44 \cdot \log(n + 2) - 0,328$$

Красно-чёрное

$$h \leq 2 \cdot \log(n + 1)$$

Высота АВЛ-дерева меньше, чем высота красно-чёрного дерева (на 38%).

Добавление элемента

АВЛ

- ✓ поиск отца для добавляемой вершины – $O(h)$
- ✓ непосредственное добавление вершины – $O(1)$
- ✓ поиск разбалансированной вершины - $O(h)$
- ✓ повороты (будет выполнен 1 поворот) – $O(1)$

Красно-чёрное

- ✓ поиск отца для добавляемой вершины – $O(h)$
- ✓ непосредственное добавление вершины – $O(1)$
- ✓ перекрашивания - $O(h)$
- ✓ повороты (будет выполнено не более 2-х) – $O(1)$

Удаление элемента

АВЛ

- ✓ поиск удаляемой вершины – $O(h)$
- ✓ непосредственное удаление – $O(1)$
- ✓ поворот – $O(1)$
- ✓ повторная балансировка (повороты) - $O(h)$

Красно-чёрное

- ✓ поиск удаляемой вершины – $O(h)$
- ✓ непосредственное удаление – $O(1)$
- ✓ повороты (не более 3-х) - $O(1)$
- ✓ перекрашивания - $O(h)$

Тесты показывают, что AVL-деревья быстрее красно-чёрных во всех операциях

<https://radius-server.livejournal.com/598.html>

<http://nathanbelue.blogspot.com/2012/05/red-black-versus-avl.html>

1. Структуры данных и алгоритмы: теория и практика: учеб. пособие / В.М. Котов, Е.П. Соболевская. Мн.: БГУ. 2004.– С.141–153.
2. Алгоритмы: построение и анализ / Т. Кормен [и др.] – М.: Вильямс, 2005. – С 336 –356.



БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ

Спасибо за внимание!