



NATIONAL RESEARCH
UNIVERSITY

АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Red-Black Trees **Красно-Черные деревья**

Рамон Антонио Родригес Залепинос
arodriges@hse.ru

Структура курса Алгоритмы и Структуры Данных 2020

	№	Дата	Тема лекции	Краткое описание
Модуль № 1	1	08 сен	Введение	О курсе, пользе быстрых алгоритмов, литература
	2	15 сен	Асимптотика	Фундамент всех дальнейших тем: оценка сложности
	3	22 сен	Базовые структуры данных 1	Стек, разные виды списков, дек, очередь
	4	29 сен	Базовые структуры данных 2	Двоичные деревья поиска (база для всех деревьев)
	5	06 окт	Bitmaps	Быстро \cup , \cap INT, large CPU caches, WAN, Concise, Roaring
	6	13 окт	Хеширование, хэш таблицы 1	Быстро найти key в множестве; хеш. цепочками, анализ
Модуль № 2	7	1 27 окт	Хеширование, хэш таблицы 2	Открытая адресация, кэш CPU, кукушкино хеширование
	8	2 03 ноя	Фильтры	Быстро проверить \in множеству; Ф. Блума, Кукушкин Ф.
	9	3 10 ноя	СД для вторичной памяти	Все данные не помещаются в RAM, B-tree, efficient I/O
	10	4 17 ноя	Пространственные СД	Помимо int, str \exists другие типы; 80% геоданных; R-дерево
	11	5 24 ноя	Параллельные СД 1	Speedup за счет multicore CPU; повышение утилизации
	12	6 01 дек	Параллельные СД 2	Параллельный SkipList – понятен для бакалавриата
	13	7 08 дек	Деревья в оперативной памяти	Красно-черное, другие виды: сложнейшее в конце курса
	14	8 15 дек	Современные тренды	Резюме, machine learning в СД, распределенные системы
СЕССИЯ с 21.12.2020				на чем основано



- недельный календарный тематический план
- последовательное раскрытие материала
- лекции, семинары и ДЗ синхронизированы
- цель – максимально дать знания и научить

Базовые структуры данных

Screenshot из Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн - Алгоритмы. Построение и анализ.

Часть III. Структуры данных 259

Глава 10. Элементарные структуры данных 264

Глава 11. Хеширование и хеш-таблицы 285

Глава 12. Бинарные деревья поиска 319

Глава 13. Красно-черные деревья 341

Глава 14. Расширение структур данных 372

Часть IV. Усовершенствованные методы разработки и анализа 389

- **Мы прошли с Вами все базовые структуры данных** (красно-черное дерево в конце курса – одна из самых сложных структур данных)
- **Хеш-таблицы важнее, чем иногда кажется:** не нужно их недооценивать; некоторые представления об эффективности СД развеяны: на семинаре вы собственноручно, на практике сравнили производительность красно-черных деревьев и хэш-таблиц (если не нужны next&prev)
- Хеш-таблицы, хеширование: **старая и активная область R&D** (редкое сочетание, recall Cuckoo)

Польза рассмотренных тем курса **АиСД2020**

		Польза для будущей карьеры
Модуль № 1	№ 1 Введение	<ul style="list-style-type: none"> • Общие понятия о пользе быстрых структур данных • Теоретические основы всех тем по алгоритмам и СД
	№ 2 Асимптотика	
	№ 3 Базовые структуры данных 1	<ul style="list-style-type: none"> • Классические основные, базовые темы
	№ 4 Базовые структуры данных 2	
	№ 5 Bitmaps	<ul style="list-style-type: none"> • INT – основной и часто встречающийся тип данных; bitmaps – самые быстрые СД над множествами из INT'ов для операций \in, \cap и $\cup \Rightarrow$ пригодятся в software
	№ 6 Хеширование, хэш таблицы 1	<ul style="list-style-type: none"> • Хэш-таблица, классическая СД: самый быстрый способ найти объект по ключу (string, int, ...); если не нужны next/prev, хеш-т. быстрее trees: убедились на семинарах
Модуль № 2	Тема лекции	
	№ 7 Хеширование, хэш таблицы 2	<ul style="list-style-type: none"> • Классическая СД: самый быстрый способ быстро, <i>но нечетко</i>, проверить \in ключа (string, int, ...) множеству • Как эффективно организовать хранение и доступ к данным во вторичной памяти, <i>в том числе в облаке</i>, если они не помещаются целиком в RAM?
	№ 8 Фильтры	
	№ 9 СД для вторичной памяти	<ul style="list-style-type: none"> • Вы познакомились с особенностями и сложностями пространственных СД; 80% данных имеют геопривязку; Apple, Yandex, Facebook, Amazon нужны соотв. кадры
	№ 10 Пространственные СД	
	№ 11 Параллельные СД 1	<ul style="list-style-type: none"> • Благодаря простоте SkipList, Вы познакомились с подходами ускорения СД на многоядерных CPU
	№ 12 Параллельные СД 2	
	№ 13 Деревья в оперативной памяти	<ul style="list-style-type: none"> • Теоретически интерес представляют сложные деревья
	№ 14 Современные тренды	<ul style="list-style-type: none"> • Подведем итоги курса, посмотрим на новейшие темы: machine learning для СД и распределенные системы
<ul style="list-style-type: none"> • Где используются изученные структуры данных? – это универсальные структуры данных, не привязаны к конкретной сфере компьютерных наук • Вспомним постановки задач из каждой лекции, напр. множество целых чисел (INT), тогда Вам нужны bitmaps для быстрых операций пересечения и объединения 		

Разносторонность тем курса AiСД2020

	№	Тема лекции	Польза для будущей карьеры
Модуль № 1	1	Введение	• Не всегда алгоритм с лучшей асимптотикой быстрее: влияют паттерны доступа в RAM, кэш CPU, др. факторы
	2	Асимптотика	
	3	Базовые структуры данных 1	• Список с ограничителем, XOR список, Y-связный список, проблемы с двоичным деревом поиска
	4	Базовые структуры данных 2	
	5	Bitmaps	• Многие используют bitmaps: Yandex ClickHouse, Apache Spark, LinkedIn Pinot, Microsoft Visual Studio; высокая скорость за счет простых операций и кэшей CPU
	6	Хеширование, хэш таблицы 1	• Вероятностный анализ хеширования
Модуль № 2		Тема лекции	
	7	Хеширование, хэш таблицы 2	• Кукушкино хеширование (2014 г.)
	8	Фильтры	• Фильтр Блума (1970 г.), Кукушкин Фильтр (2016 г.)
	9	СД для вторичной памяти	• Особенности вторичной памяти: HDD, SSD, Облако • B-дерево (1970 г.)
	10	Пространственные СД	• Вдобавок к традиционным int & string • Городское планирование, чрезвычайные ситуации, лесное хозяйство, качество воздуха, транспорт • R-дерево (1984 г.)
	11	Параллельные СД 1	• Многопоточное программирование, multicore CPU
	12	Параллельные СД 2	• ConcurrentSkipList (2006 г.): понятная версия для 2курса
	13	Деревья в оперативной памяти	• Красно-черные деревья (1972 г.)
	14	Современные тренды	• Machine learning для СД и распределенные системы

- Фундаментальность: bitmaps (30+ лет), хеширование (много лет), фильтры, R-tree, B-tree
- Собеседования: зависит от уровня компании; также надо строить эффективные системы, не только проходить собеседования
- Изученные СД применяются во многих областях и во многих программных системах, **в том числе (но не только) в СУБД**, компьютерных сетях, распределенных системах, ...

«Постановка задачи»

Входные данные

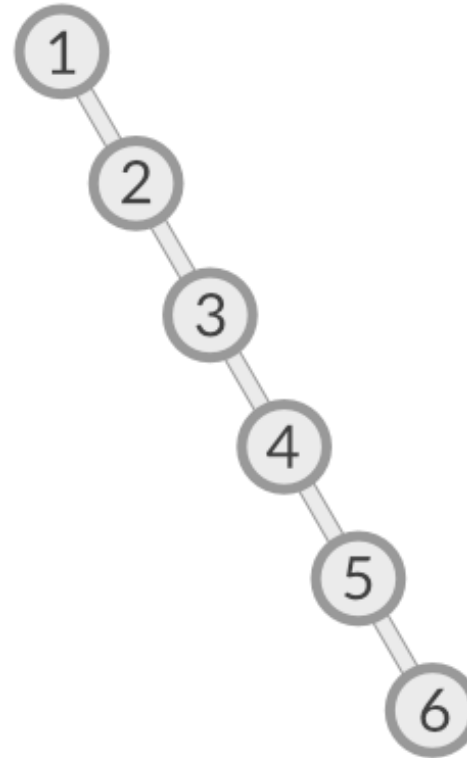
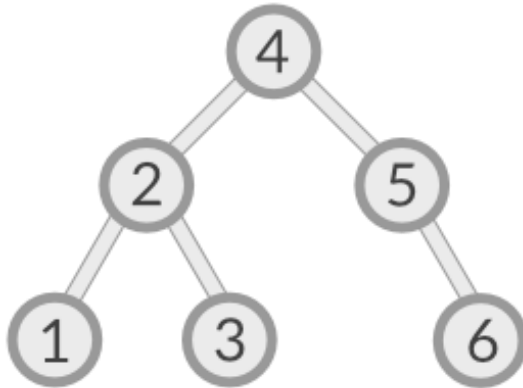
- Ключи $key \in K$ (может быть полезная нагрузка)

Цель – эффективное выполнение словарных операций

- По ключу
- Insert, delete, search, previous, next, ...

Проблемы с binary search tree

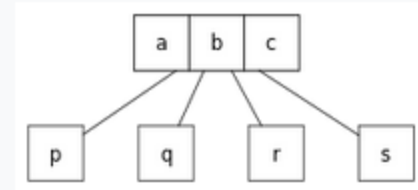
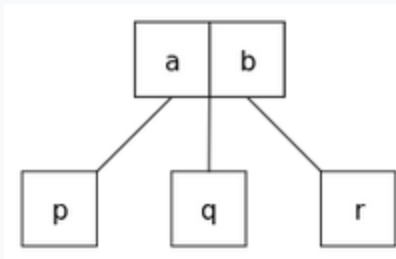
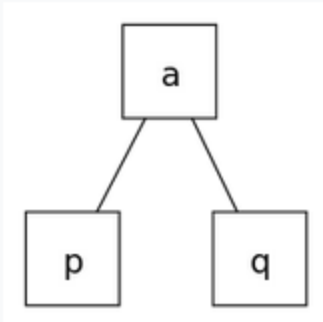
- сложность операций пропорциональна высоте дерева
- может вырождаться в обычный список



2-3 деревья, 2-3-4 деревья

это B-деревья

- 2-3 деревья: от 2 до 3 детей
- 2-3-4 деревья: от 2 до 4 детей



[link](#)

Базовые операции	Худший случай
Занимаемое место	$O(n)$
Поиск по ключу	$O(\log n)$
Вставка по ключу	$O(\log n)$
Удаление по ключу	$O(\log n)$

Красно-черное дерево

Rudolf Bayer, 1972 г.

Q: какие еще стр. данных разработал R.Bayer?

Вид сбалансированного дерева

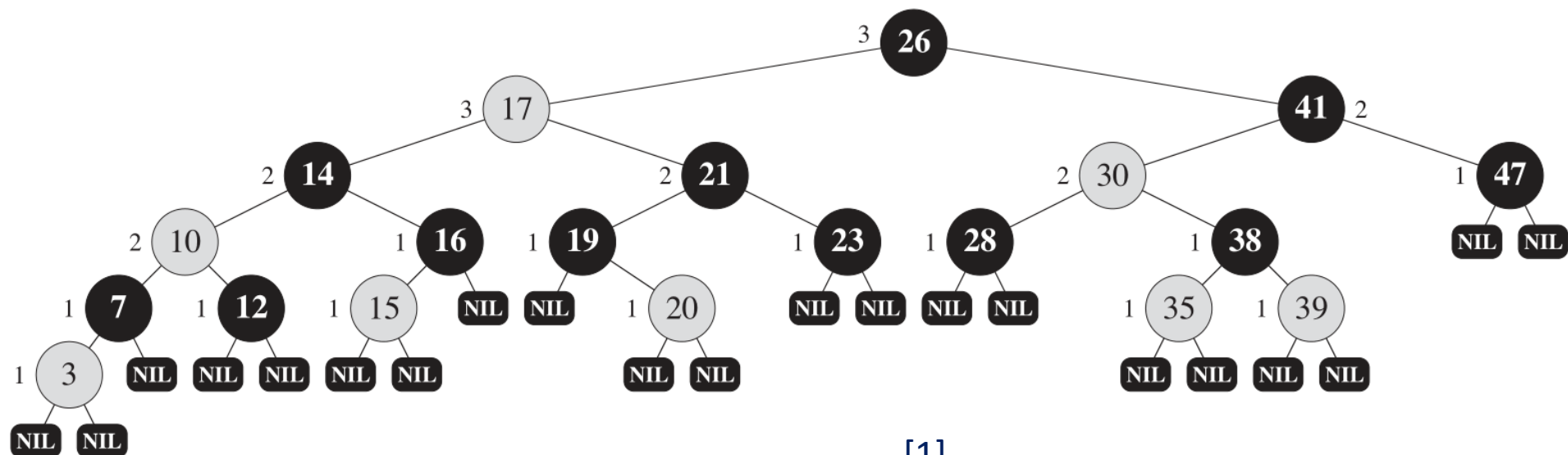
- Хранит n элементов
- Каждый элемент имеет ключ key
- Для типа ключа определены операции сравнения, напр., $<$

Базовые операции	Худший случай
Занимаемое место	$O(n)$
Поиск по ключу	$O(\log n)$
Вставка по ключу	$O(\log n)$
Удаление по ключу	$O(\log n)$

Note: красно-черное дерево сложнее всех структур данных, которые мы с вами учили so far

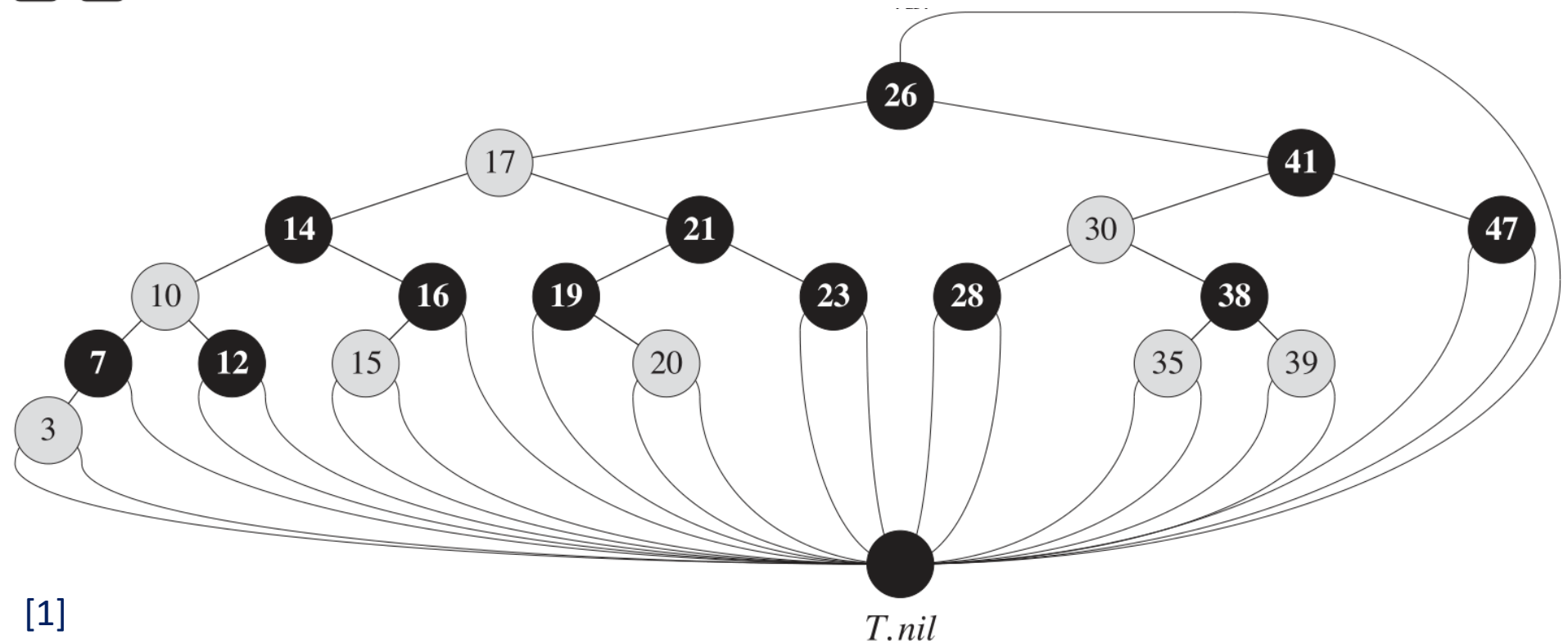
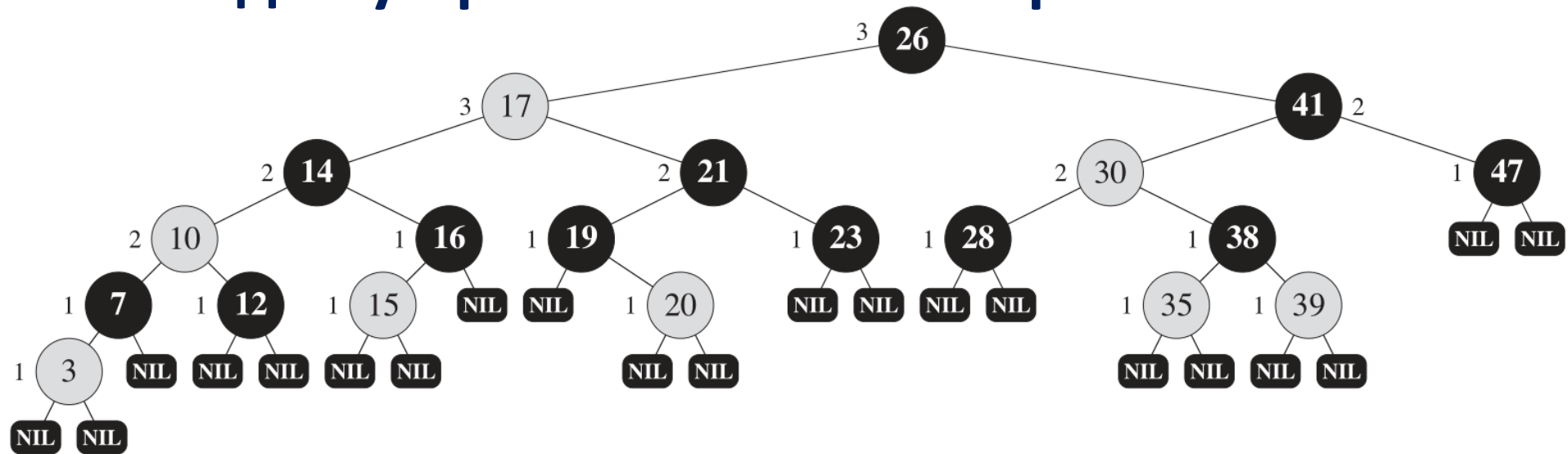
Идеи устройства RB-tree

1. **RB-tree** – бинарное (двоичное) дерево поиска
2. Каждый узел – красный (RED) либо черный (BLACK)
3. Приблизненно сбалансированное дерево – длины любых двух простых путей от корня до листа отличаются не более в $2 \times$ раза
4. Каждый узел содержит атрибуты *color*, *key*, *left*, *right* и *p(parent)*
5. Используется ограничитель *NIL* для указателей (если нет дочернего либо родительского узла)



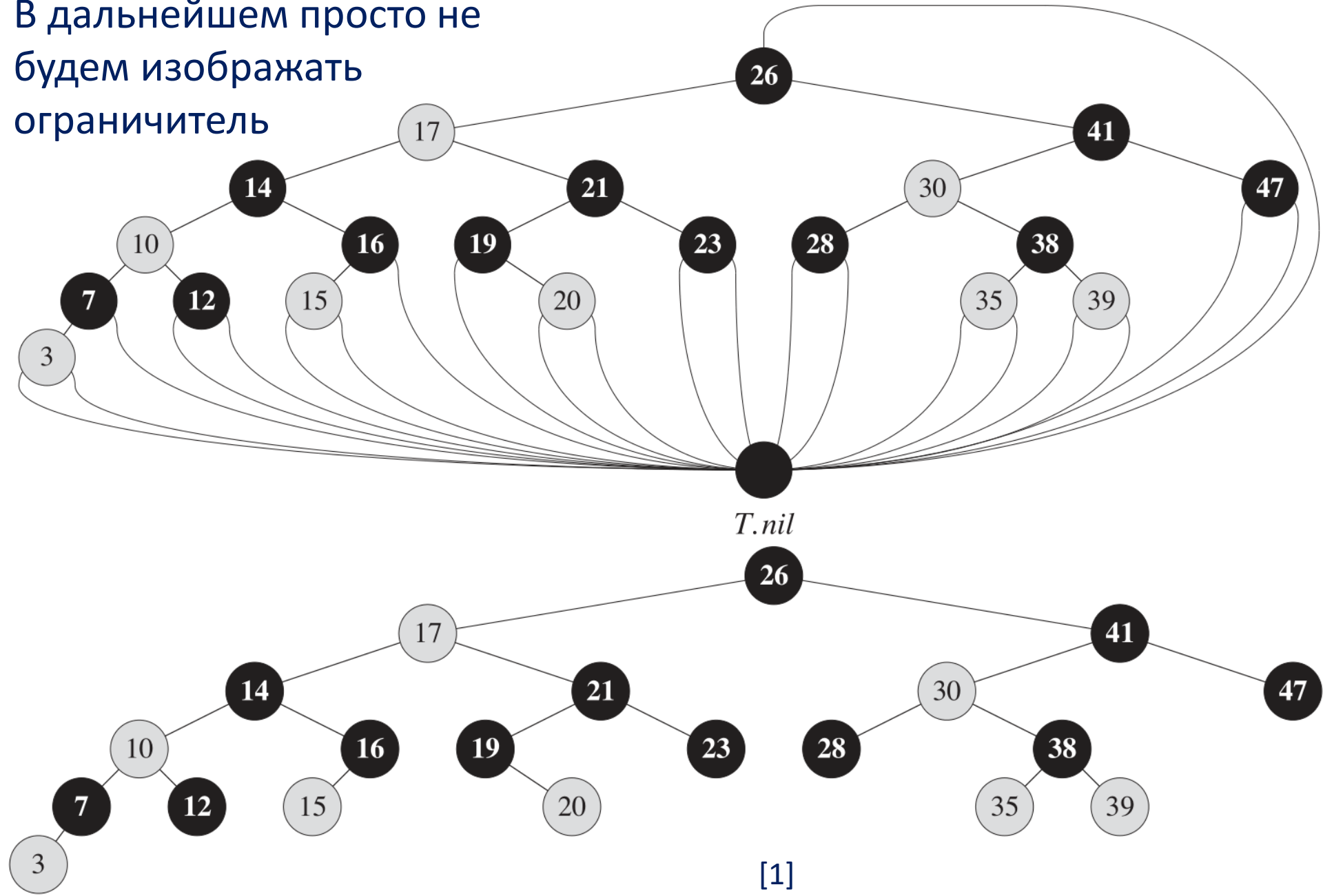
[1]

Идеи устройства RB-tree: ограничитель



Идеи устройства RB-tree : ограничитель

В дальнейшем просто не
будем изображать
ограничитель



Поиск в RB-tree

Поскольку **RB-tree** обладает теми же свойствами, что и бинарное дерево поиска, алгоритм поиска аналогичен

TREE-SEARCH(x, k)

1 **if** $x == \text{NIL}$ or $k == x.\text{key}$

2 **return** x

3 **if** $k < x.\text{key}$

4 **return** TREE-SEARCH($x.\text{left}, k$)

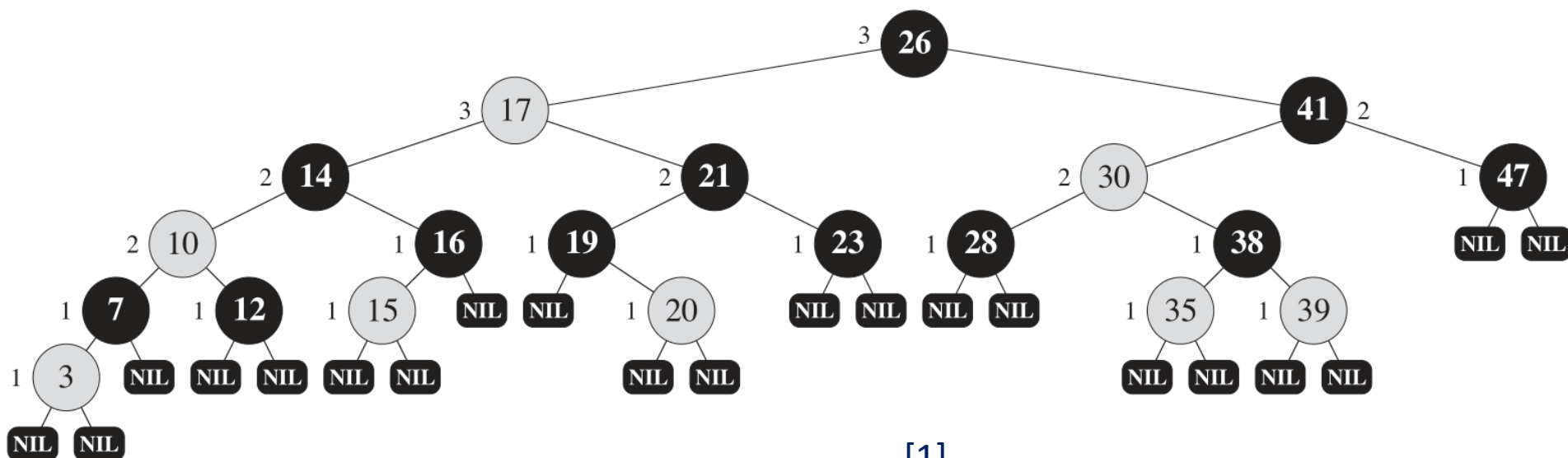
5 **else return** TREE-SEARCH($x.\text{right}, k$)

Давайте поищем
ключи

35

20

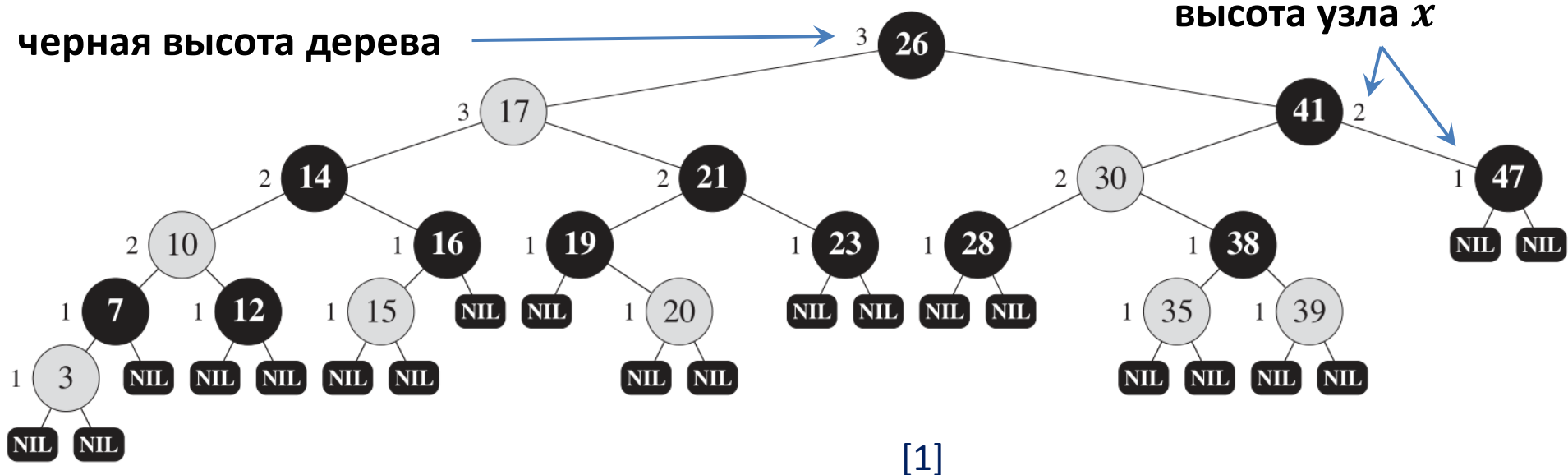
3



Определение RB-tree

Бинарное дерево поиска является **RB-tree** если оно удовлетворяет следующим **красно-черным свойствам**:

1. Каждый узел – либо **красный** либо **черный**
2. Корень дерева – **черный**
3. Каждый лист дерева (*NIL*) – **черный**
4. Если узел **красный**, то оба дочерних узла – **черные**
5. Для каждого узла все простые пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество **черных** узлов



Почему красно-черное дерево является приближенно сбалансированным?

- Сложность операций пропорциональна высоте дерева
- Высота дерева может быть любой...
- Высота ветвей – насколько высота одной ветви может отличаться от высоты другой ветви?

Лемма о высоте RB-Tree

Красно-черное дерево с n внутренними узлами имеет высоту, не превышающую $2\log(n + 1)$

- Покажем по индукции, что поддереву любого узла x содержит минимум $2^{bh(x)} - 1$ внутренних узлов
- Если высота x равна 0, то x – лист (NIL) и $2^{bh(x)} - 1 = 2^0 - 1 = 0$
- Если $bh(x) > 0$, то дочерний узел имеет высоту
 - $bh(x)$ (потомок **красный**) либо
 - $bh(x) - 1$ (потомок **черный**)
- Предположим по индукции, что в поддереве каждого потомка минимум $2^{bh(x)-1} - 1$ узлов
- \Rightarrow дерево с корнем в x имеет минимум внутренних узлов $(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$

Note: любой узел (кроме листа) всегда имеет 2 потомка

Лемма о высоте RB-Tree

Красно-черное дерево с n внутренними узлами имеет высоту, не превышающую $2\log(n + 1)$

- \Rightarrow дерево с корнем в x имеет минимум внутренних узлов
 $(2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1) + 1 = 2^{bh(x)} - 1$

Пусть высота дерева равна h . Согласно свойству 4, по крайней мере половина узлов на любом простом пути от корня к листу (не считая корень), должны быть черными.

$\Rightarrow bh(\text{root})$ как минимум равна $= h/2$

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

$$\log(n + 1) \geq h/2$$

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

Свойство 4.

Если узел **красный**, то оба дочерних узла – **черные**

(т.е. красные узлы встречаются, но их число строго ограничено числом черных узлов)

«Трюки» красно-черных деревьев

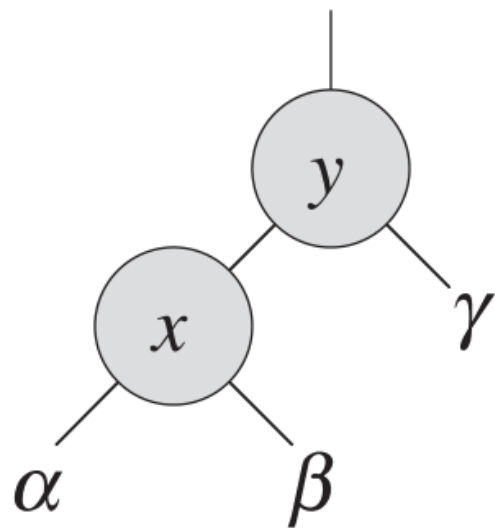
- Сложность операций пропорциональна высоте дерева
- Высота дерева может быть любой...
- Высоты ветвей – отличаются друг от друга не более чем в 2 раза
- Поиск, вставка, удаление за $O(\log n)$
 - Первый этап: принцип алгоритмов бинарных деревьев поиска – за $O(\log n)$ – может привести к нарушению красно-черных свойств
 - Второй этап: восстановление красно-черных свойств – тоже за $O(\log n)$ путем перекрашивания вершин и «трюков» с указателями (изменение их структур)

Поговорим о поворотах...

Повороты: левый и правый

- Локальные операции в дереве поиска, которые сохраняют свойство бинарного дерева поиска
- Работают за $O(1)$

[1]

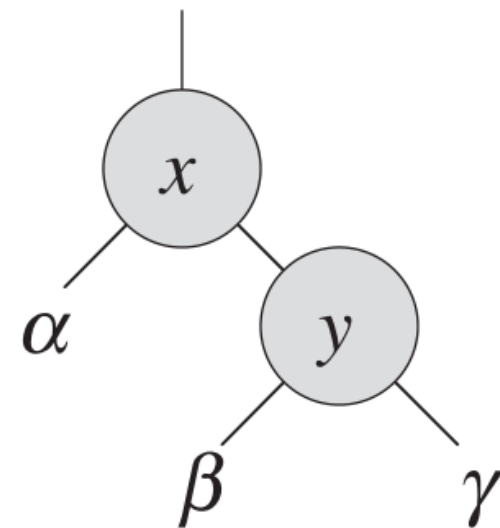


$y.left \neq NIL$

LEFT-ROTATE(T, x)



RIGHT-ROTATE(T, y)

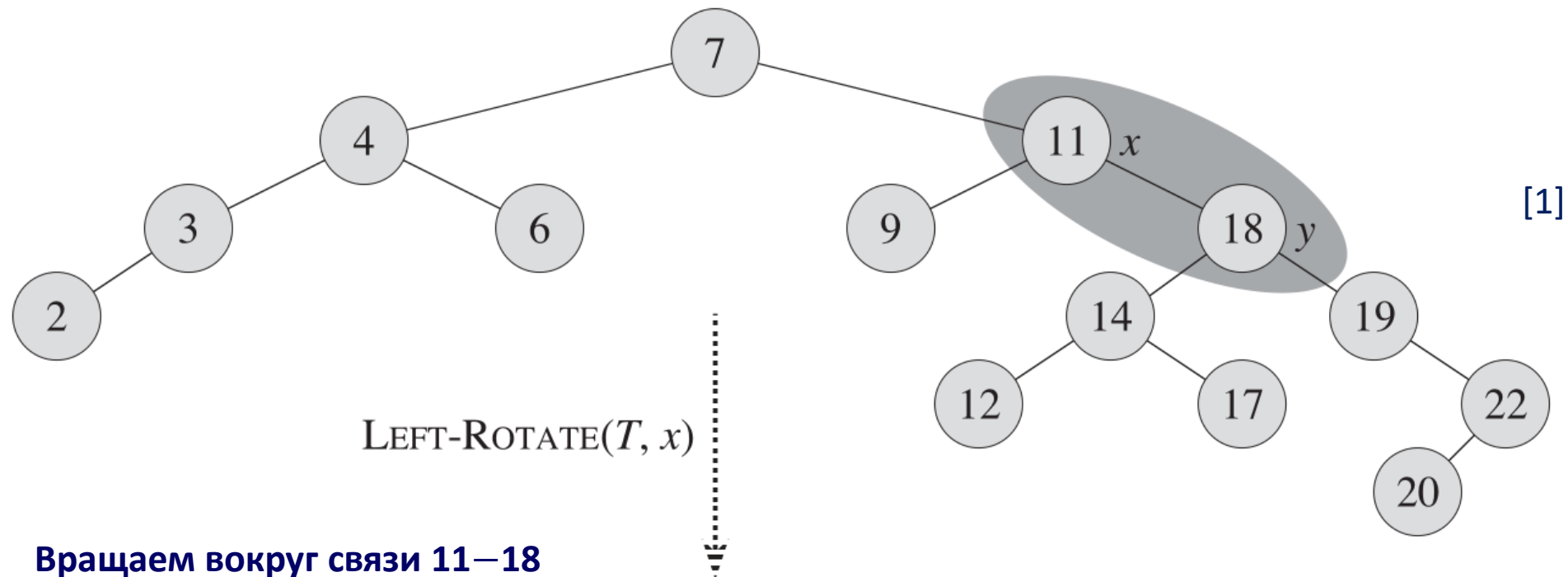


$x.right \neq NIL$

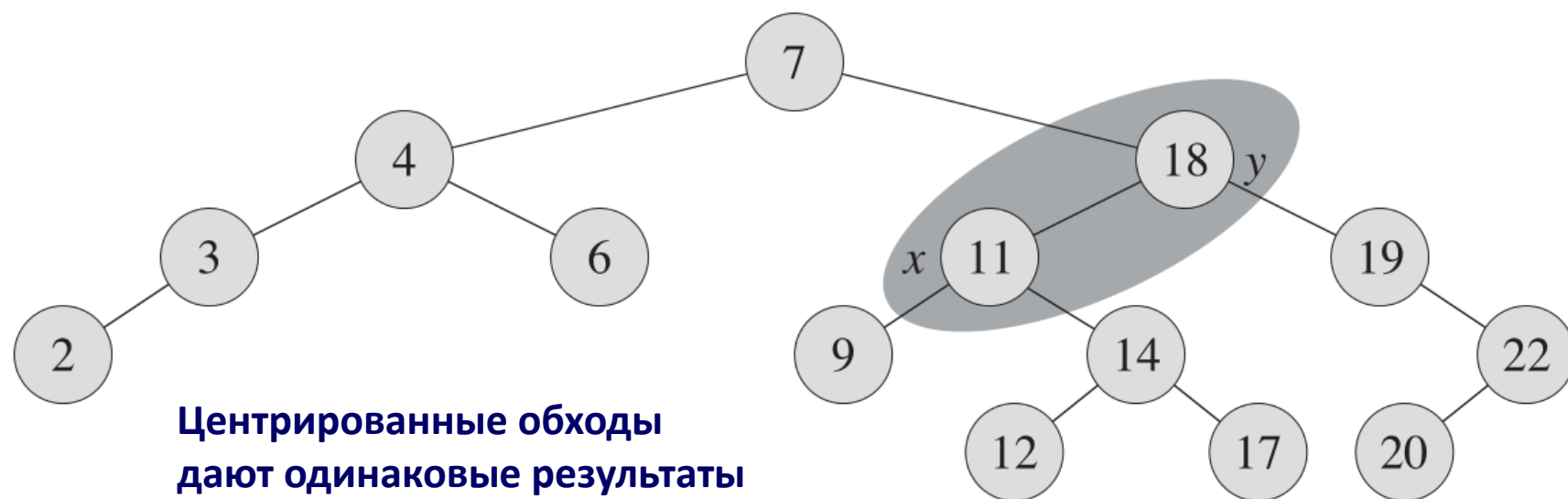
Вращаем вокруг связи $x - y$

Центрированный обход: $\alpha, x, \beta, y, \gamma$

Пример поворота



Вращаем вокруг связи 11–18



Центрированные обходы
дают одинаковые результаты

Вставка в RB-Tree

RB-INSERT(T, z)

```
1   $y = T.nil$ 
2   $x = T.root$ 
3  while  $x \neq T.nil$ 
4       $y = x$ 
5      if  $z.key < x.key$ 
6           $x = x.left$ 
7      else  $x = x.right$ 
8   $z.p = y$ 
9  if  $y == T.nil$ 
10      $T.root = z$ 
11 elseif  $z.key < y.key$ 
12      $y.left = z$ 
13 else  $y.right = z$ 
14  $z.left = T.nil$ 
15  $z.right = T.nil$ 
16  $z.color = RED$ 
17 RB-INSERT-FIXUP( $T, z$ )
```

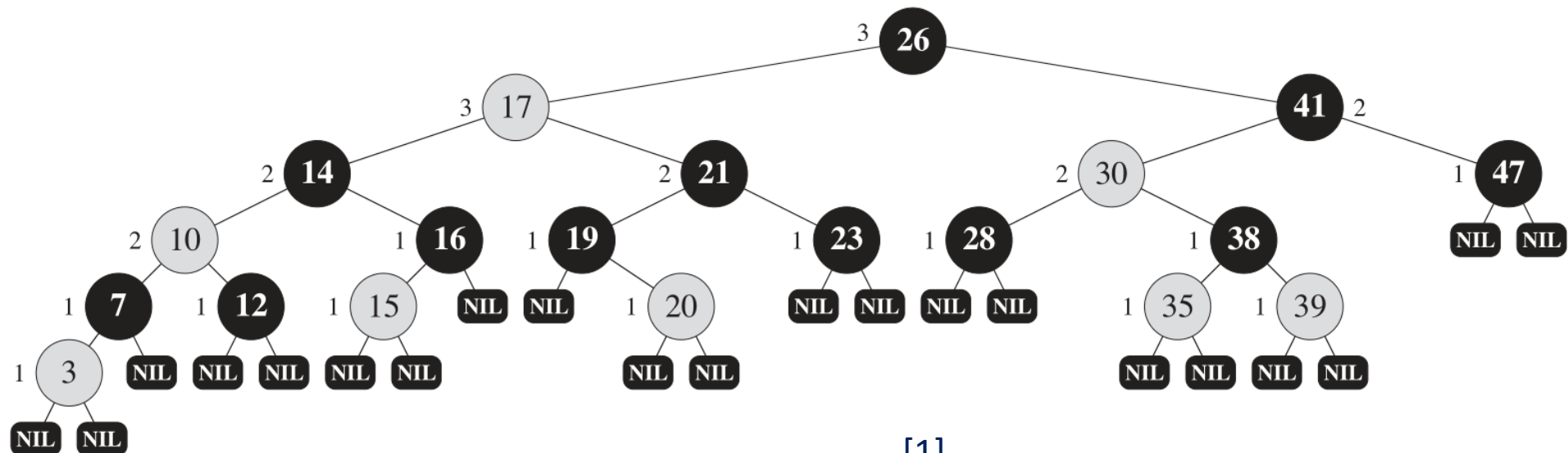
- Вставляем новый узел будто в двоичное дерево поиска
- Окрашиваем новый узел в **красный** цвет
- Выполняем **повороты** для восстановления **красно-черных** свойств

Особенности:

- Ограничитель *NIL*, а не *nullptr*
- $z.left = z.right = NIL$
- Назначение цвета новому узлу
- Вызов RB-Insert-Fixup

Какие свойства RB-tree могут быть нарушены при вставке нового узла?

1. Каждый узел – либо **красный** либо **черный**
2. Корень дерева – **черный**
3. Каждый лист дерева (*NIL*) – **черный**
4. Если узел **красный**, то оба дочерних узла – **черные**
5. Для каждого узла все простые пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество **черных** узлов



Вставка в RB-Tree: RB-Insert-Fixup

RB-INSERT-FIXUP(T, z)

```
1  while  $z.p.color == RED$ 
2      if  $z.p == z.p.p.left$ 
3           $y = z.p.p.right$ 
4          if  $y.color == RED$ 
5               $z.p.color = BLACK$ 
6               $y.color = BLACK$ 
7               $z.p.p.color = RED$ 
8               $z = z.p.p$ 
9          else if  $z == z.p.right$ 
10              $z = z.p$ 
11             LEFT-ROTATE( $T, z$ )
12              $z.p.color = BLACK$ 
13              $z.p.p.color = RED$ 
14             RIGHT-ROTATE( $T, z.p.p$ )
15         else (same as then clause
              with “right” and “left” exchanged)
16   $T.root.color = BLACK$ 
```

if true $\Rightarrow z.p.p$ всегда существует

- За один вызов **RB-Insert-Fixup** выполняется максимум 2 поворота
- Случай 2 преобразовывает в Случай 3 (выполняется всегда после Случая 2)

Случай 1

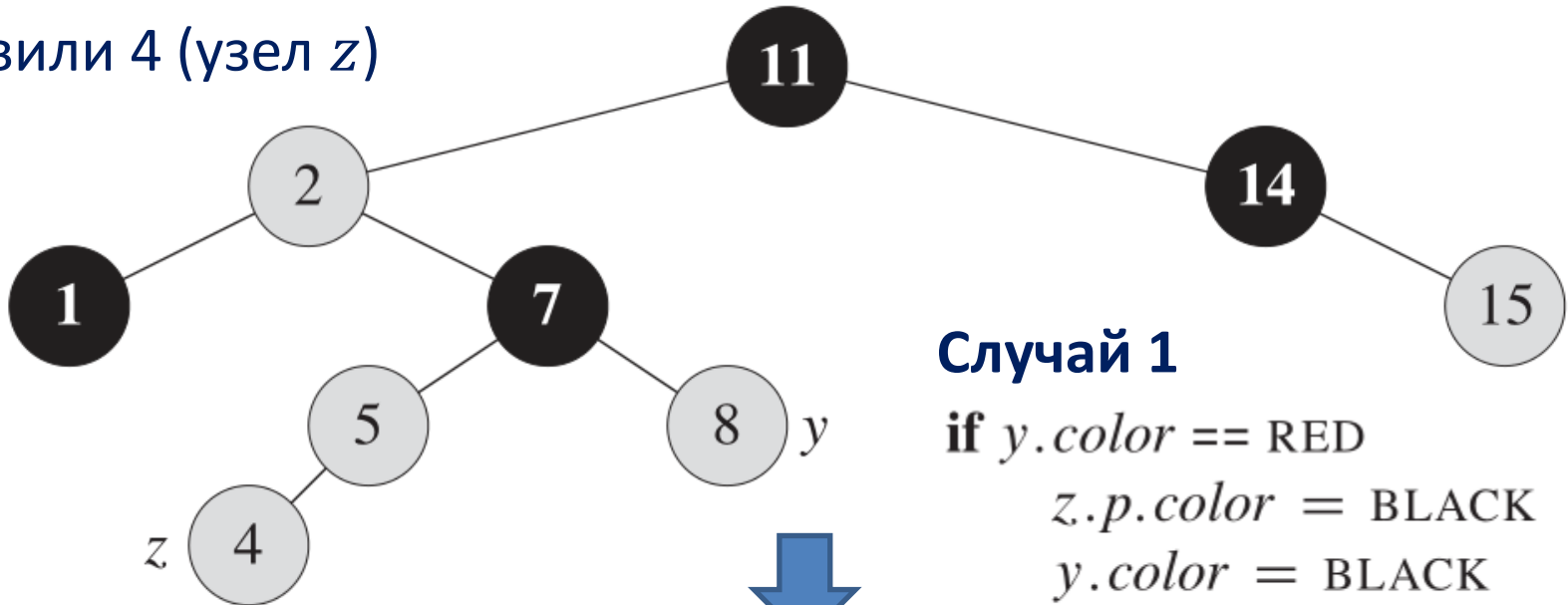
Случай 2

Случай 3

Еще три симметричных случая

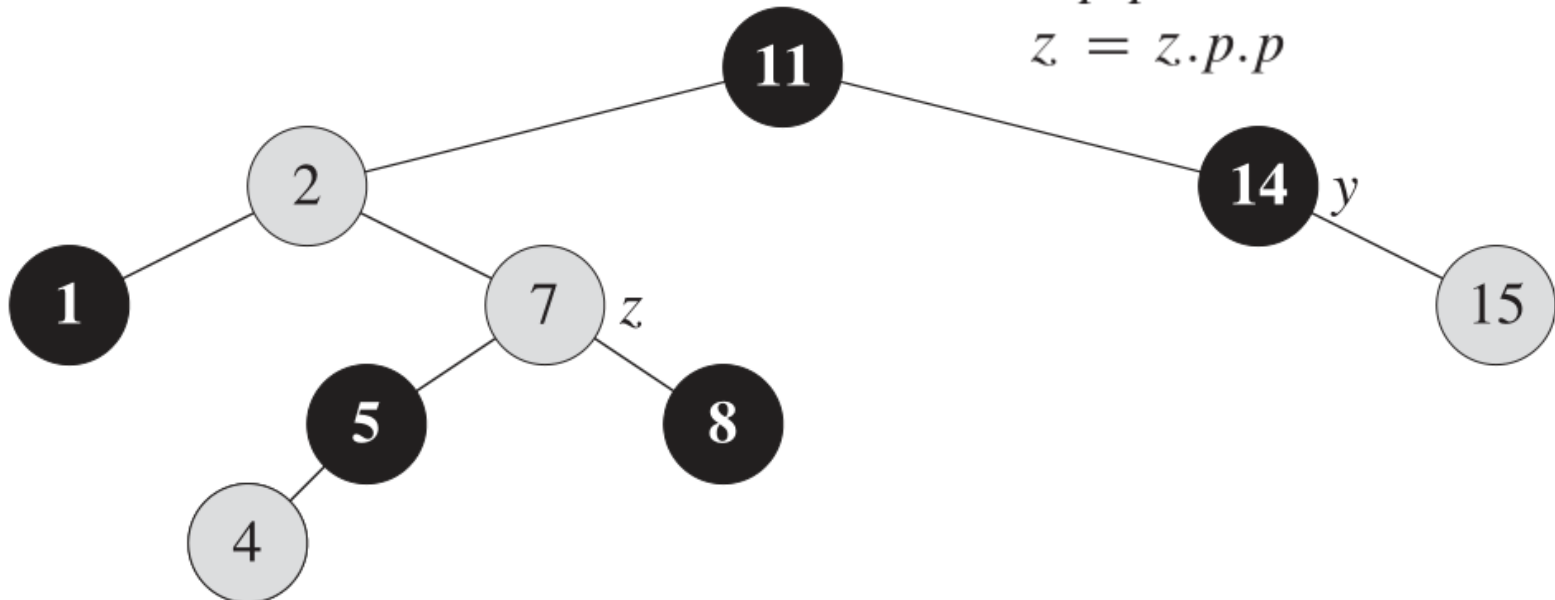
Вставка в RB-Tree: пример

Вставили 4 (узел z)



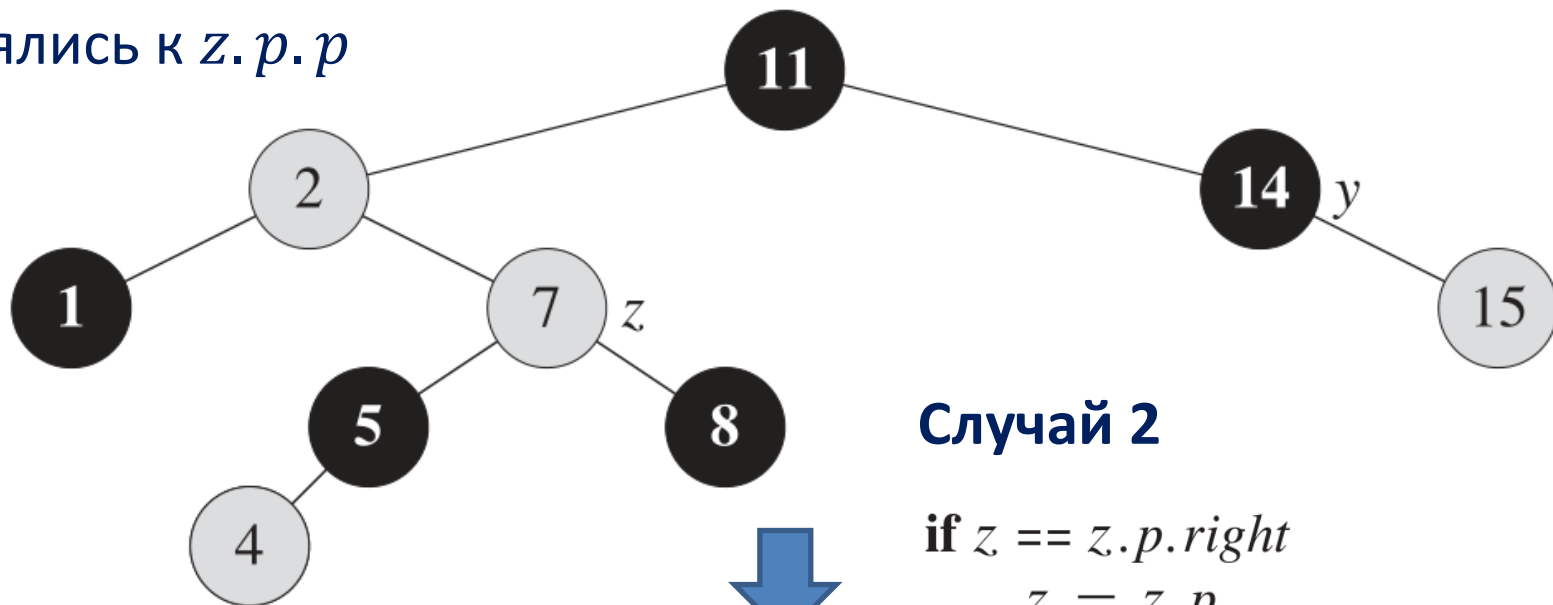
Случай 1

if $y.color == RED$
 $z.p.color = BLACK$
 $y.color = BLACK$
 $z.p.p.color = RED$
 $z = z.p.p$



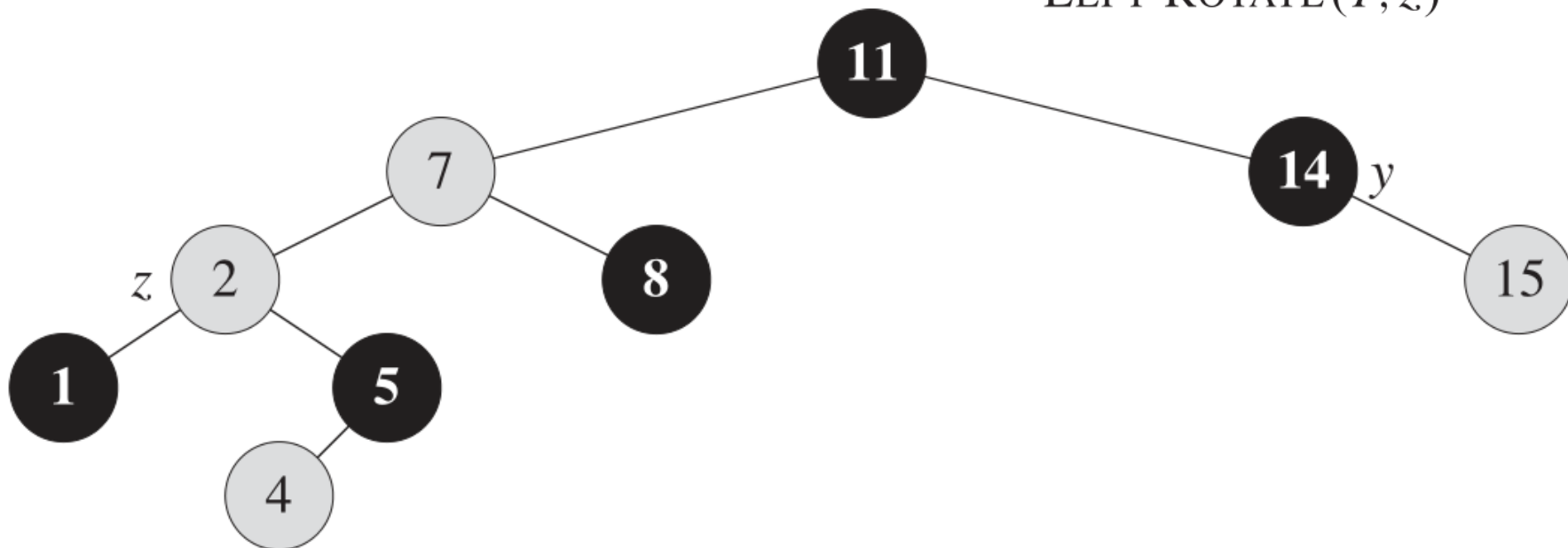
Вставка в RB-Tree: пример

Поднялись к $z.p.p$



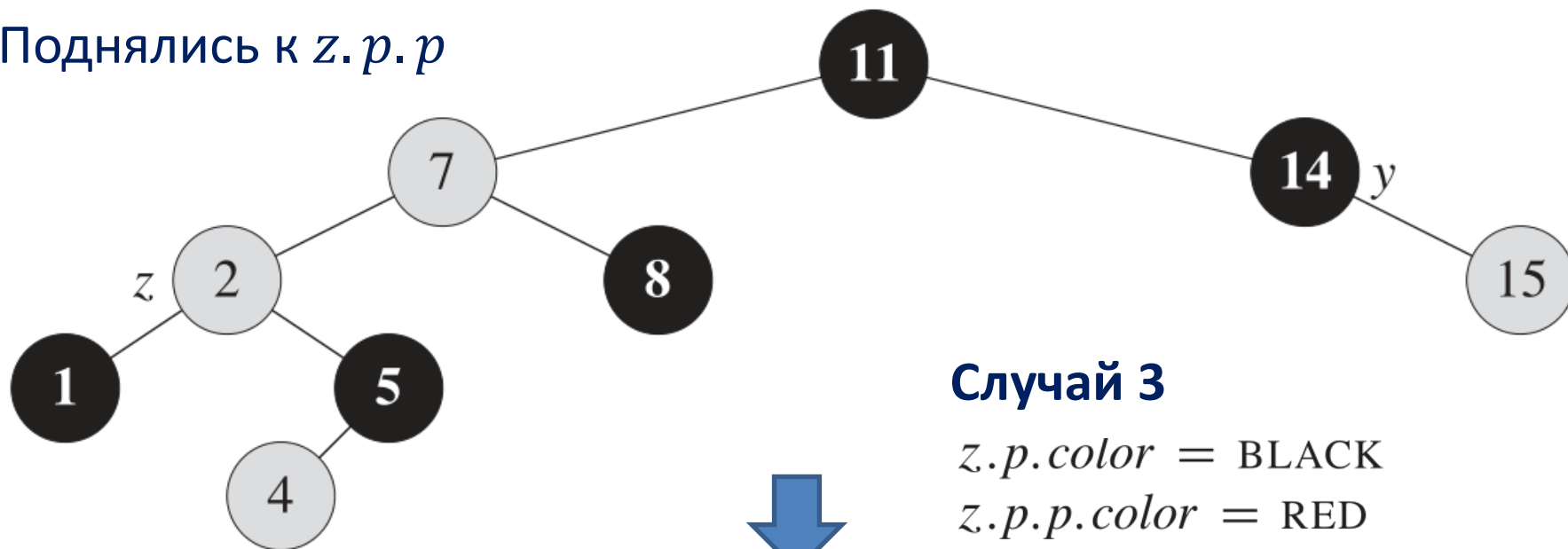
Случай 2

if $z == z.p.right$
 $z = z.p$
LEFT-ROTATE(T, z)



Вставка в RB-Tree: пример

Поднялись к $z.p.p$

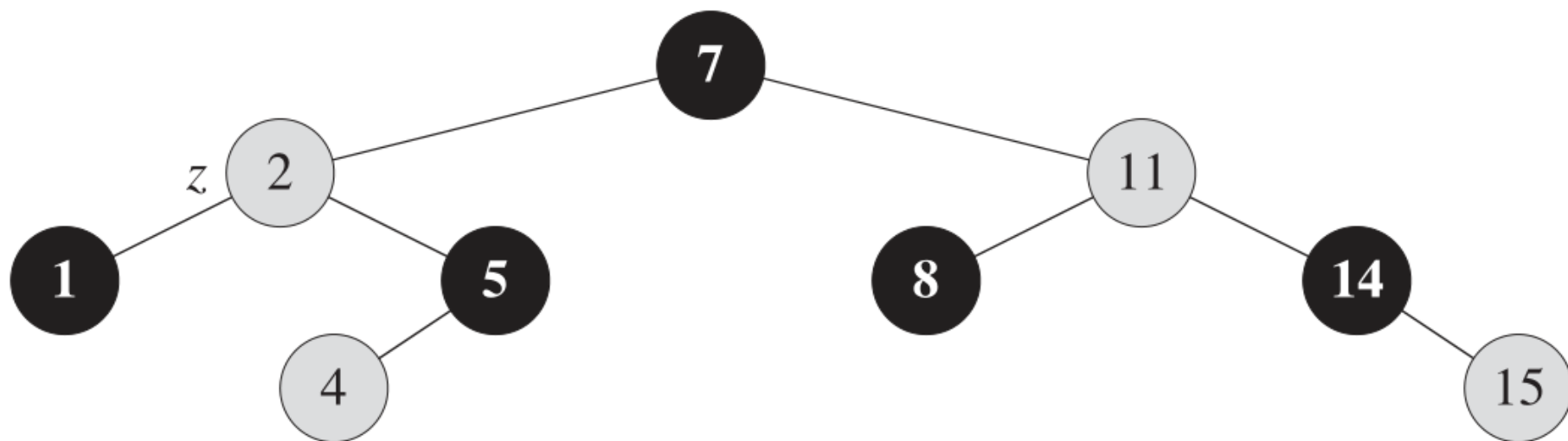


Случай 3

$z.p.color = \text{BLACK}$

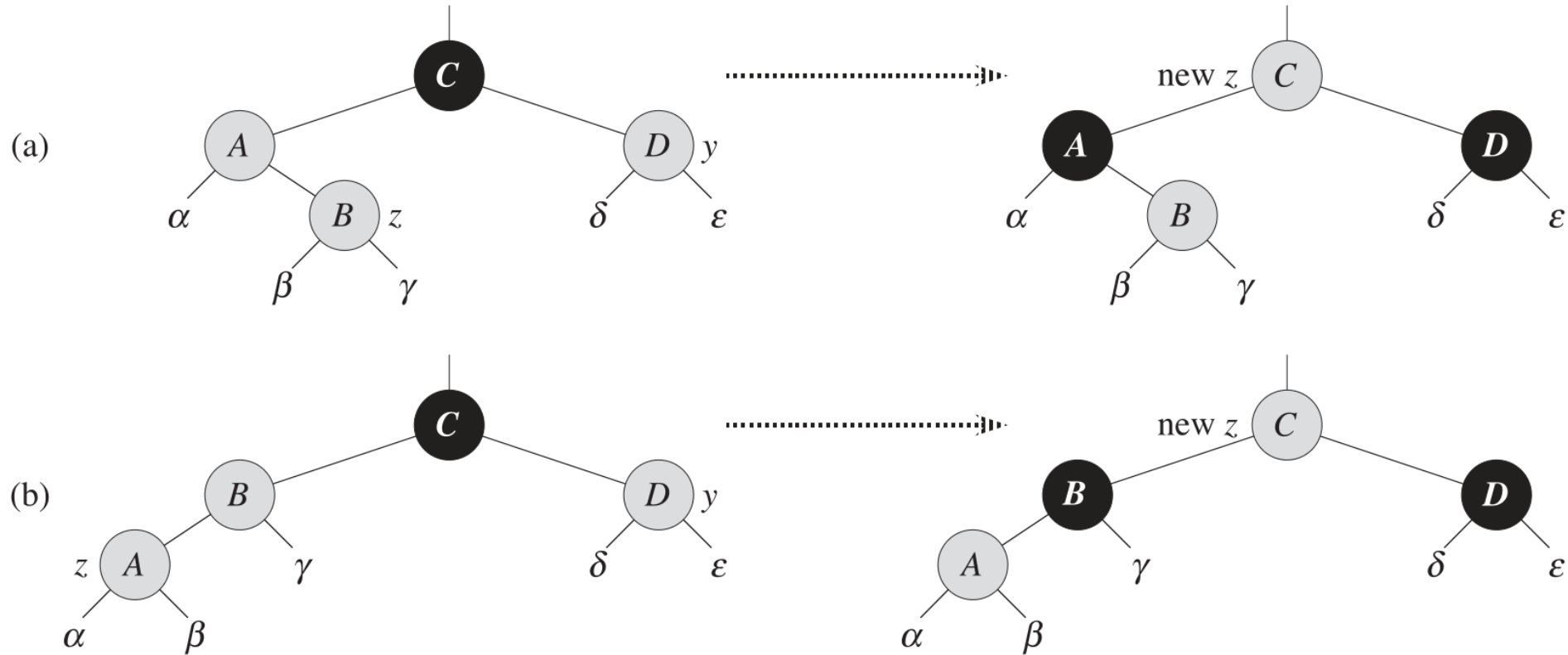
$z.p.p.color = \text{RED}$

$\text{RIGHT-ROTATE}(T, z.p.p)$



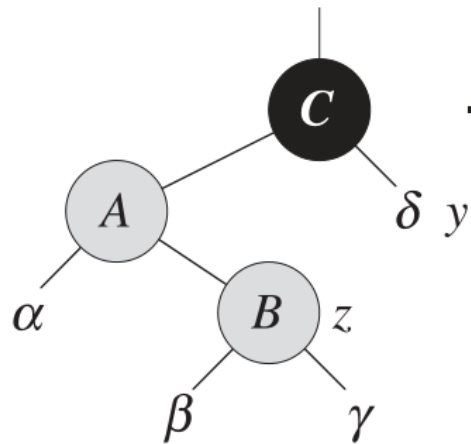
Случай 1 (два симметричных случая)

Можем перекрасить родителя и дядю в черный цвет

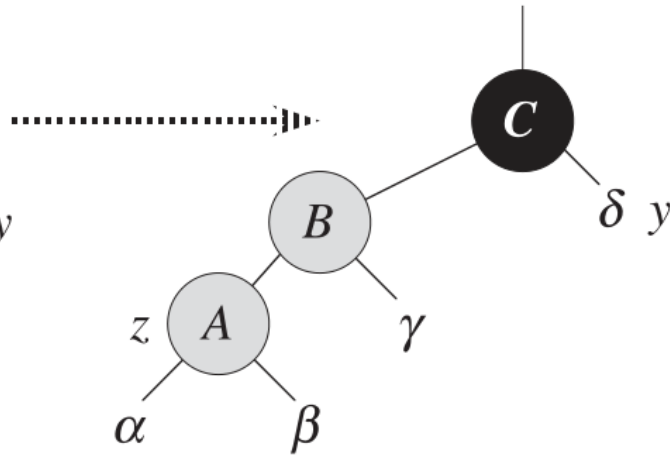


Случаи 2 и 3

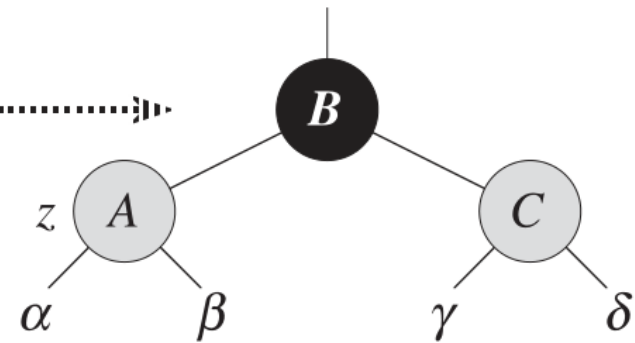
- Из случая 2 переходим в случай 3
- $\alpha, \beta, \gamma, \delta$ – черные



Случай 2



Случай 3



Анимация

<https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>

Практическое применение


Библиотечные реализации в языках программирования

std::map

<https://ru.cppreference.com/w/cpp/container/map>

Определён в заголовочном файле `<map>`

```
template<
    class Key,
    class T,
    class Compare = std::less<Key>,
    class Allocator = std::allocator<std::pair<const Key, T> >
> class map;
```

`std::map` — отсортированный ассоциативный контейнер, который содержит пары ключ-значение с неповторяющимися ключами. Порядок ключей задаётся функцией сравнения `Compare`. Операции поиска, удаления и вставки имеют логарифмическую сложность. Данный тип, как правило, реализуется как **красно-чёрное дерево** .

Java SE 12 & JDK 12

```
public class TreeMap<K,V>
    extends AbstractMap<K,V>
    implements NavigableMap<K,V>, Cloneable, Serializable
```

A Red-Black tree based `NavigableMap` implementation. The map is sorted according to the natural ordering of its keys, or by a `Comparator` provided at map creation time, depending on which constructor is used.

<https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/TreeMap.html>

Список литературы

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн - *Алгоритмы. Построение и анализ.*

Некоторые рисунки в данной презентации из [1]



NATIONAL RESEARCH
UNIVERSITY

Благодарю за внимание!

Рамон Антонио Родригес Залепинос
arodriges@hse.ru