



# Алгоритмы и структуры данных

Лекция 5. Структуры данных: деревья

(с) Глухих Михаил Игоревич, [glukhikh@mail.ru](mailto:glukhikh@mail.ru)

# Деревья

- Разновидность графа  
(неориентированный граф без циклов)

# Деревья

- ▶ Разновидность графа  
(неориентированный граф без циклов)
- ▶ Классификация
  - ▶ N-арное
    - ▶ Префиксное дерево
    - ▶ ...
  - ▶ Бинарное
    - ▶ Бинарное дерево поиска
    - ▶ ...

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet
    - ▶ Как сравниваем?

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet
    - ▶ Comparable OR Comparator

# Бинарные деревья поиска

- Элемент = Ключ + Данные
- Ключи у разных элементов не совпадают и поддерживают отношение полного порядка

# Отношение полного порядка

## ► Total ordering

►  $\leq$

►  $a \leq b \text{ AND } b \leq a \iff a == b$

►  $a \leq b \text{ AND } b \leq c \implies a \leq c$

►  $a \leq b \text{ OR } b \leq a \iff \text{ALWAYS}$

# Отношение полного порядка

## ► Total ordering

►  $\leq$

►  $a \leq b \text{ AND } b \leq a \quad \iff \quad a == b$

►  $a \leq b \text{ AND } b \leq c \quad \iff \quad a \leq c$

►  $a \leq b \text{ OR } b \leq a$

►  $<$

►  $a < b \text{ AND } b < a \quad \iff \quad \text{NEVER}$

►  $a < b \text{ AND } b < c \quad \implies \quad a < c$

►  $a < b \text{ OR } b < a \text{ OR } a == b \quad \iff \quad \text{ALWAYS}$



# Бинарные деревья поиска

- Элемент = Ключ + Данные
- Ключи у разных элементов не совпадают и поддерживают отношение порядка
- Операции
  - Search (Key)
  - Insert (Key, Value)
  - Remove (Key)
- Дополнительные операции
  - Maximum() / Minimum()

# Бинарные деревья поиска

- Элемент = Ключ + Данные
- Ключи у разных элементов не совпадают и поддерживают отношение порядка
- Операции
  - Search (Key)
  - Insert (Key, Value)
  - Remove (Key)
- Дополнительные операции
  - Maximum() / Minimum()
  - Successor(Key) / Predecessor(Key)

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable
    - ▶ first() / last()

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable
    - ▶ first() / last()
    - ▶ headSet() / tailSet() / subSet()

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable
    - ▶ first() / last()
    - ▶ headSet() / tailSet() / subSet()
  - ▶ NavigableSet extends SortedSet
    - ▶ higher() / lower() / floor() / ceiling()

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable
    - ▶ first() / last()
    - ▶ headSet() / tailSet() / subSet()
  - ▶ NavigableSet extends SortedSet
    - ▶ higher() / lower() / floor() / ceiling()
    - ▶ descendingSet() / descendingIterator()

# Бинарные деревья поиска

- ▶ Java-интерфейсы
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable
    - ▶ first() / last()
    - ▶ headSet() / tailSet() / subSet()
  - ▶ NavigableSet extends SortedSet
    - ▶ higher() / lower() / floor() / ceiling()
    - ▶ descendingSet() / descendingIterator()
    - ▶ pollFirst() / pollLast()



# Бинарные деревья поиска

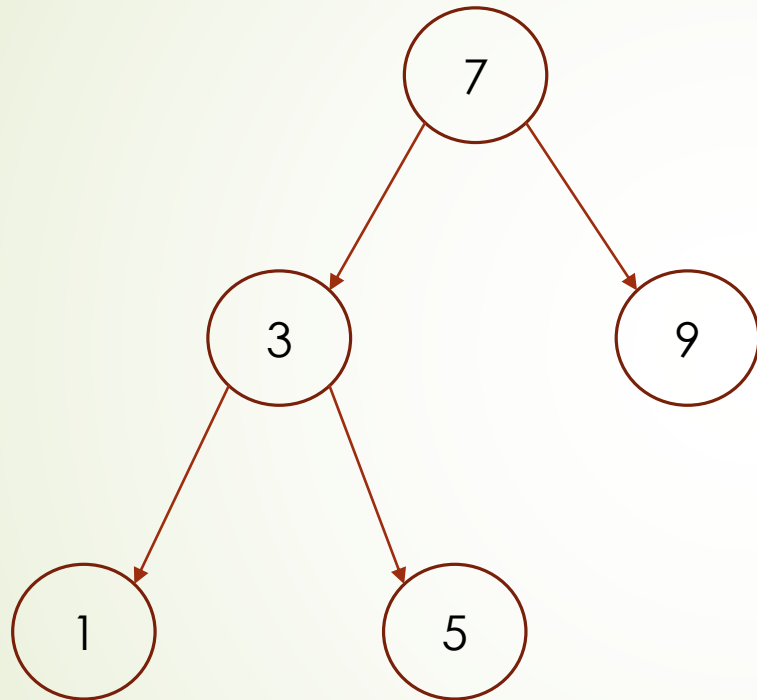
- ▶ **Java-интерфейсы**
  - ▶ SortedSet extends Set
    - ▶ Как сравниваем = Comparator / Comparable
    - ▶ first() / last()
    - ▶ headSet() / tailSet() / subSet()
  - ▶ NavigableSet extends SortedSet
    - ▶ higher() / lower() / floor() / ceiling()
    - ▶ descendingSet() / descendingIterator()
    - ▶ pollFirst() / pollLast()
- ▶ **Java-классы**
  - ▶ TreeSet = на основе красно-чёрного дерева

# Бинарные деревья поиска

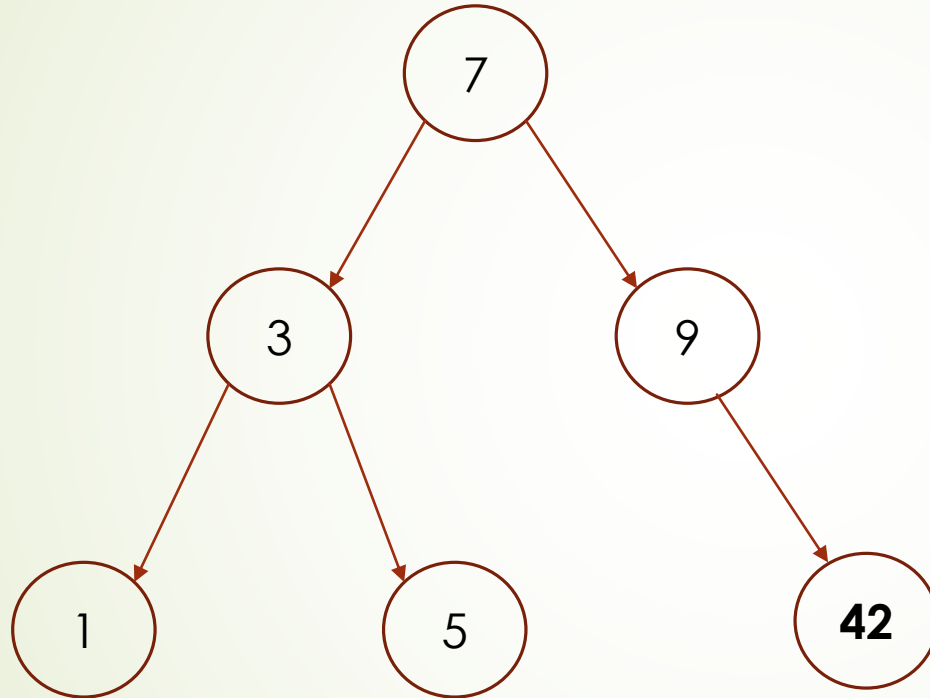
## ► Варианты реализации

- А.** Листья пустые, внутренние узлы содержат ключи и ссылки (ненулевые) ровно на два потомка
- В.** Все узлы содержат ключи и ссылки (возможно, нулевые) на два возможных потомка (см. урок 3, BinaryTree)

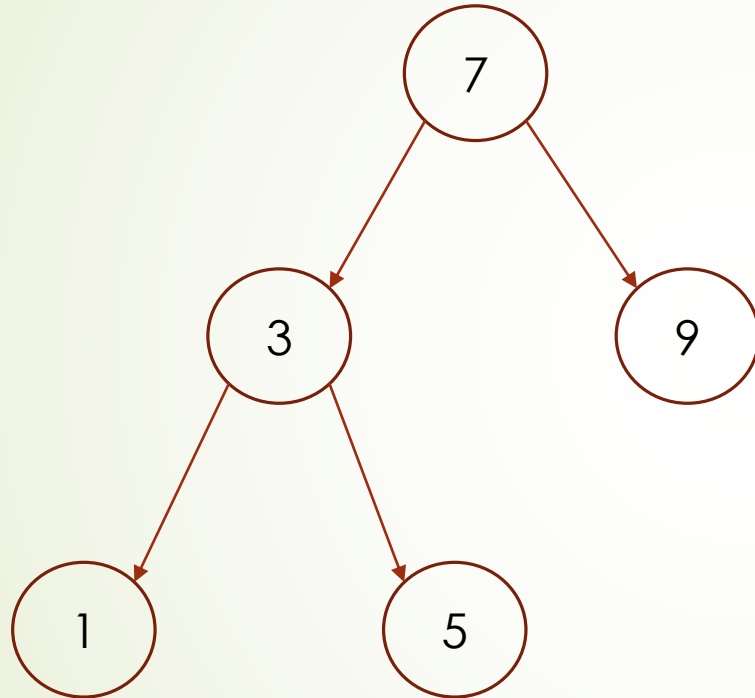
# Операции: вставка 42



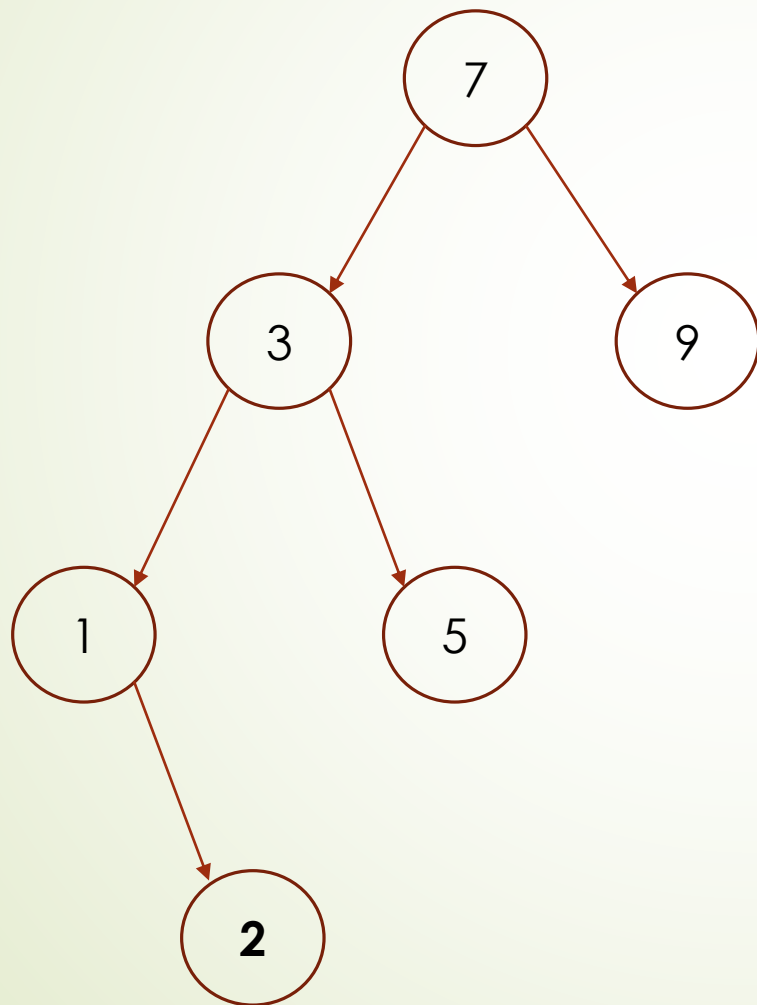
# Операции: вставка 42



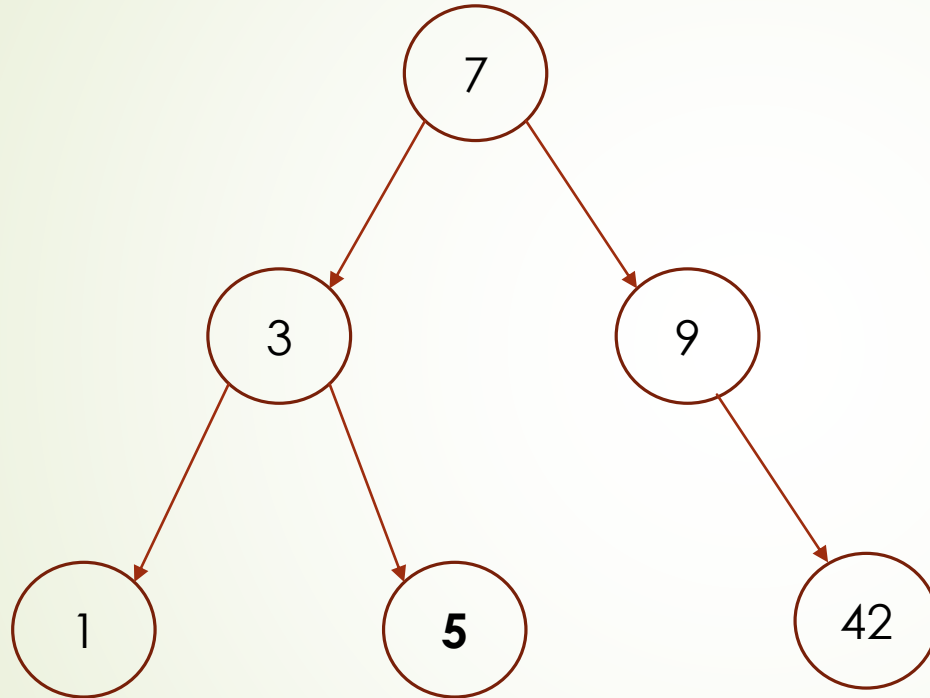
# Операции: вставка 2



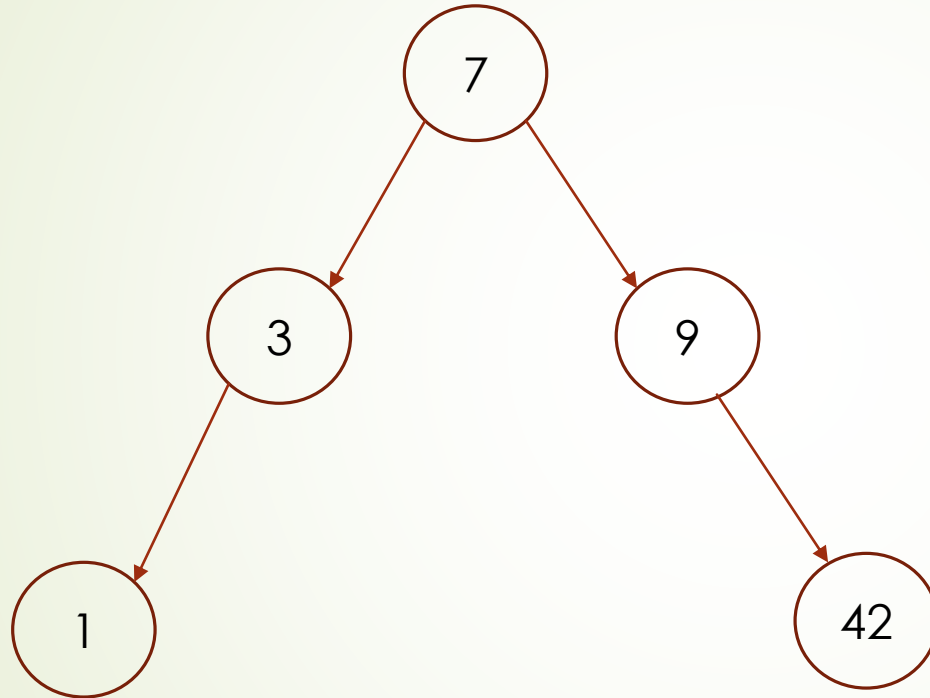
# Операции: вставка 2



# Операции: удаление 5

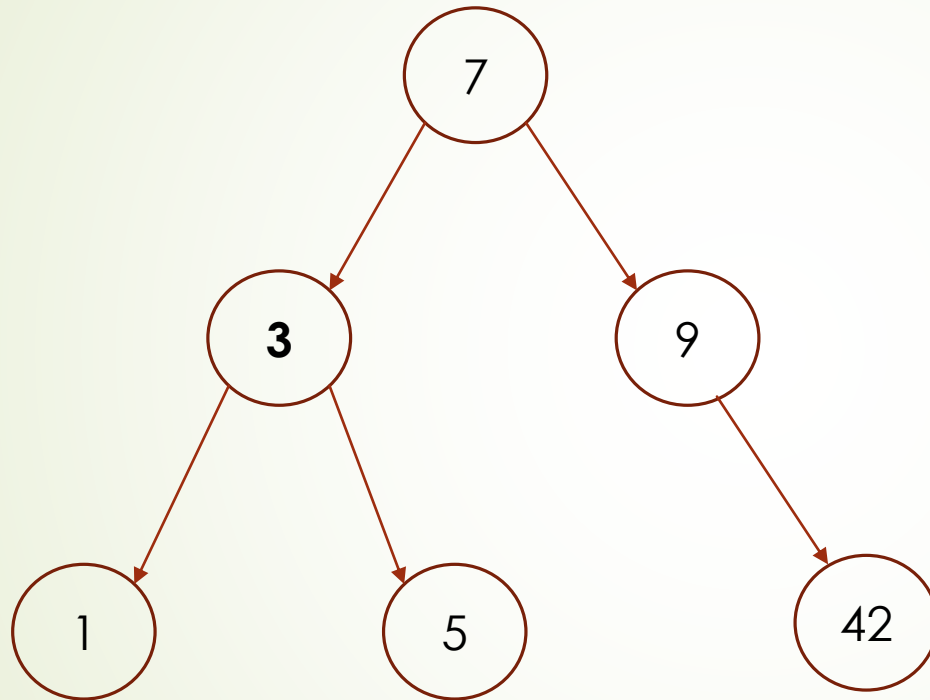


# Операции: удаление 5

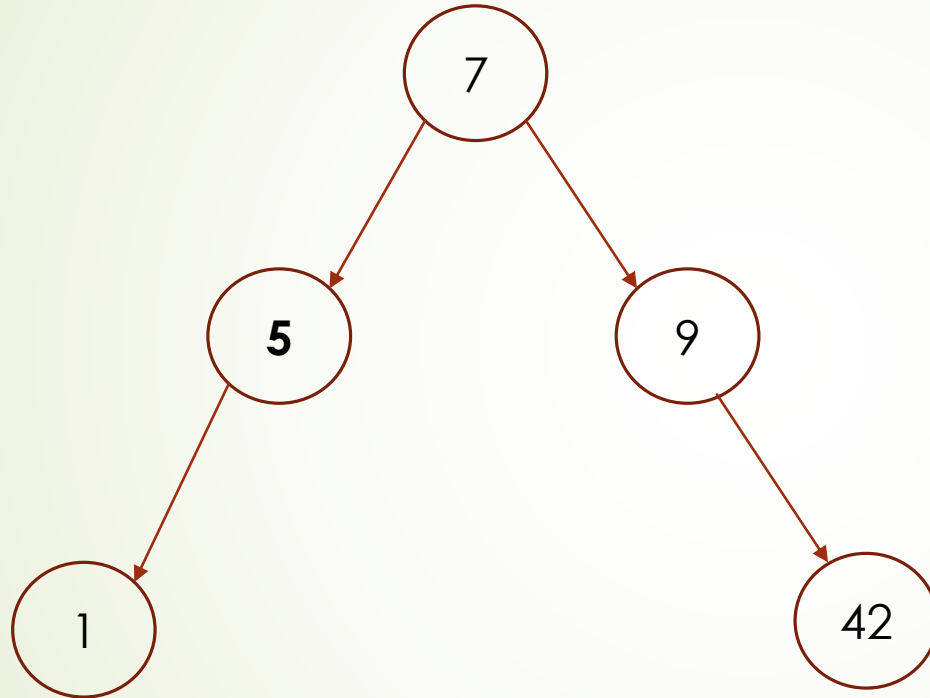




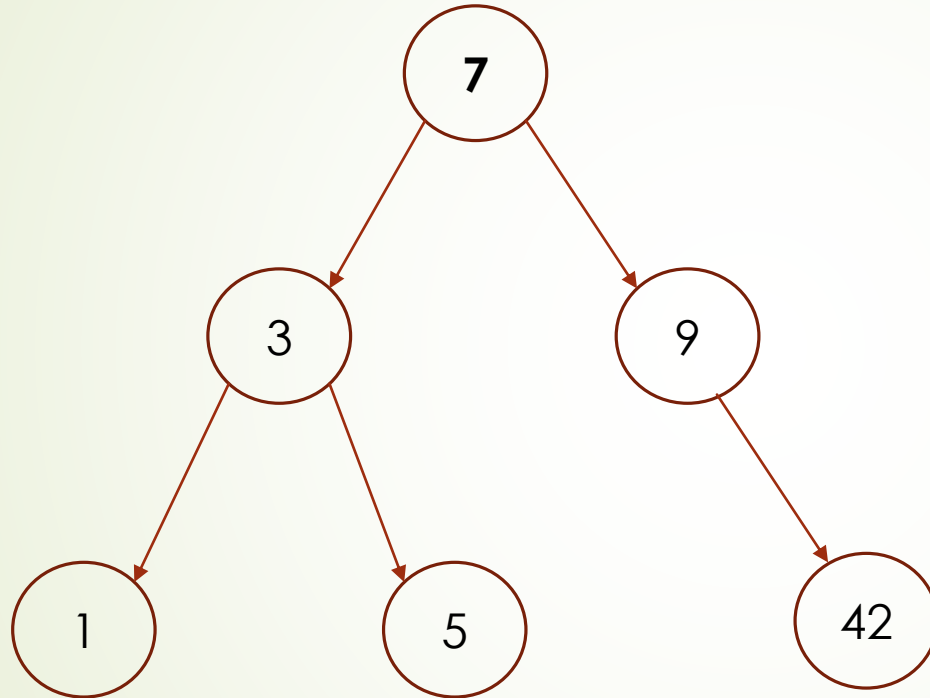
# Операции: удаление 3



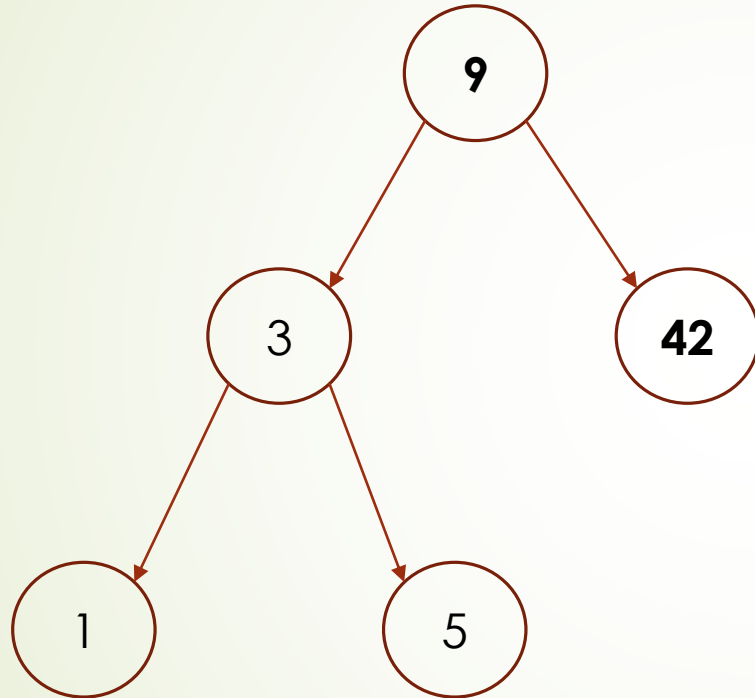
# Операции: удаление 3



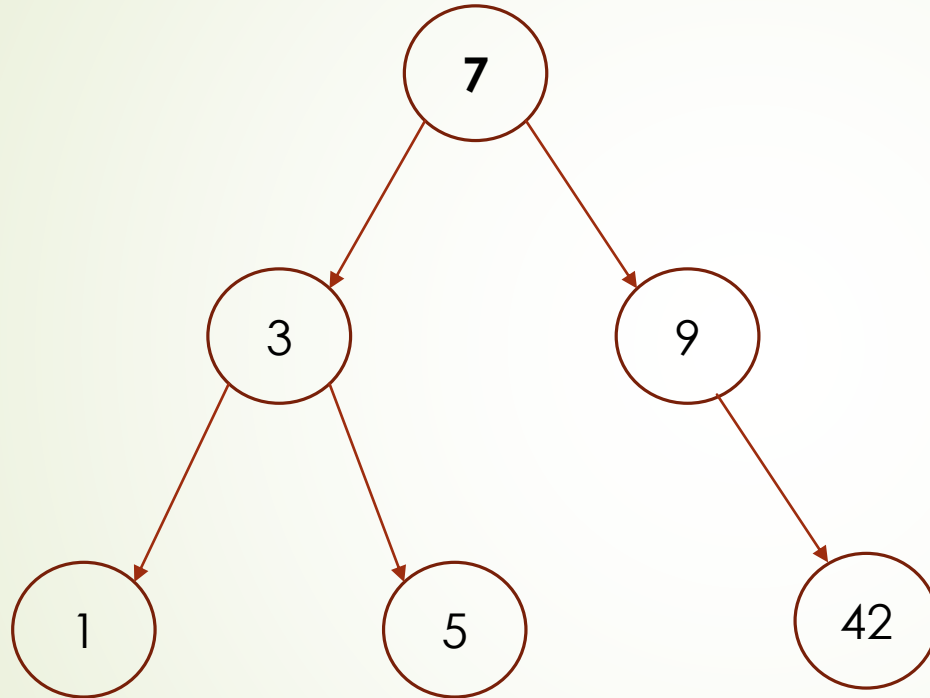
# Операции: удаление 7 (а)



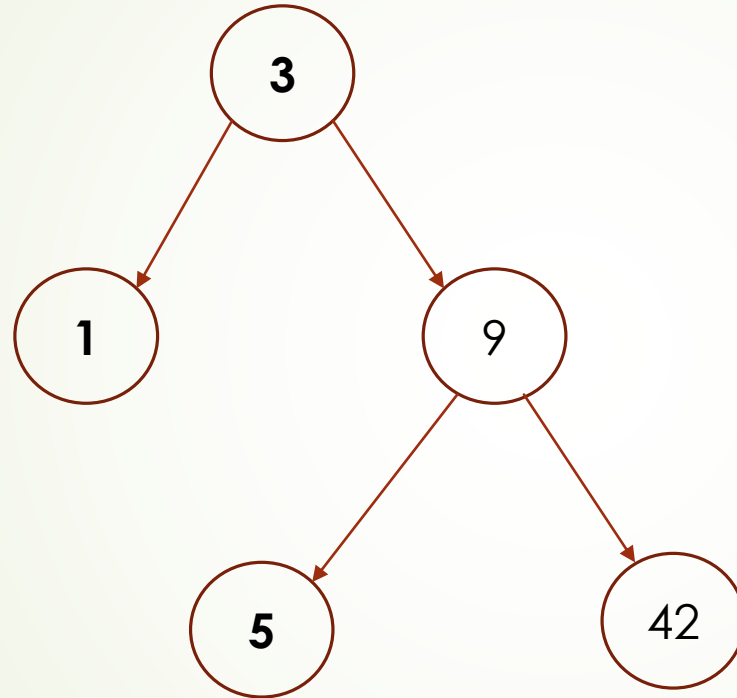
# Операции: удаление 7 (а)



# Операции: удаление 7 (б)



# Операции: удаление 7 (б)



# Красно-чёрные деревья

## ➤ Свойства

- Узел = Чёрный ИЛИ Красный
- Корневой Узел = Чёрный
- (Нулевые Узлы = Чёрные) (их может и не быть)
- Потомки Красного Узла = Чёрные
- Число Чёрных узлов на любом пути вниз = Одинаково (M)

# Красно-чёрные деревья

## ➤ Свойства

- Узел = Чёрный ИЛИ **Красный**
- Корневой Узел = Чёрный
- (Нулевые Узлы = Чёрные)
- Потомки **Красного Узла** = Чёрные
- Число Чёрных узлов на любом пути вниз = Одинаково ( $M$ )
  
- Отсюда: число **Красных узлов** на любом пути вниз  $< M$
- Отсюда: число ВСЕХ узлов на любом пути вниз  $< 2M - 1$



## Операция с деревом – вставка

- Вставляем **КРАСНЫЙ** узел
- Если это первый узел – перекрашиваем в **ЧЁРНЫЙ**

# Пример

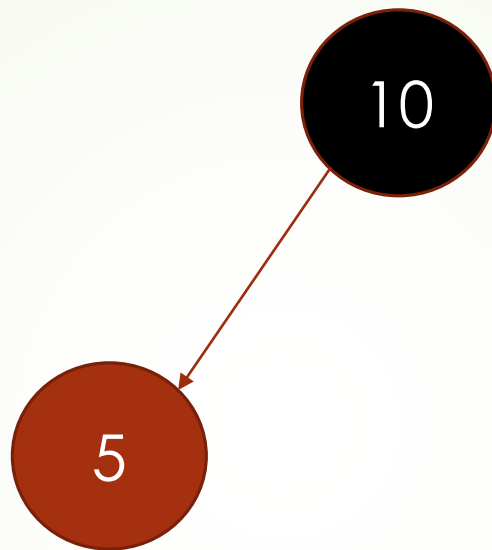


10

## Операция с деревом – вставка

- Вставляем **КРАСНЫЙ** узел
- Если это первый узел – перекрашиваем в ЧЁРНЫЙ
- Если родитель узла ЧЁРНЫЙ – ОК

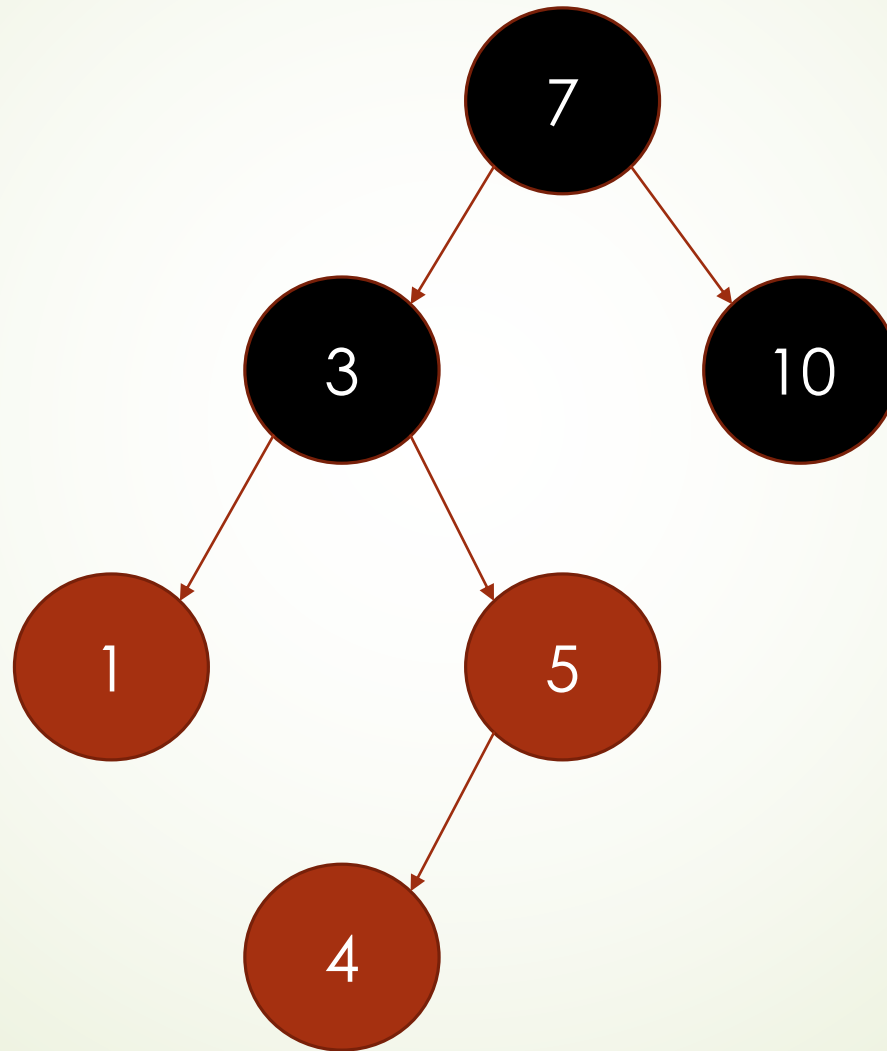
# Пример



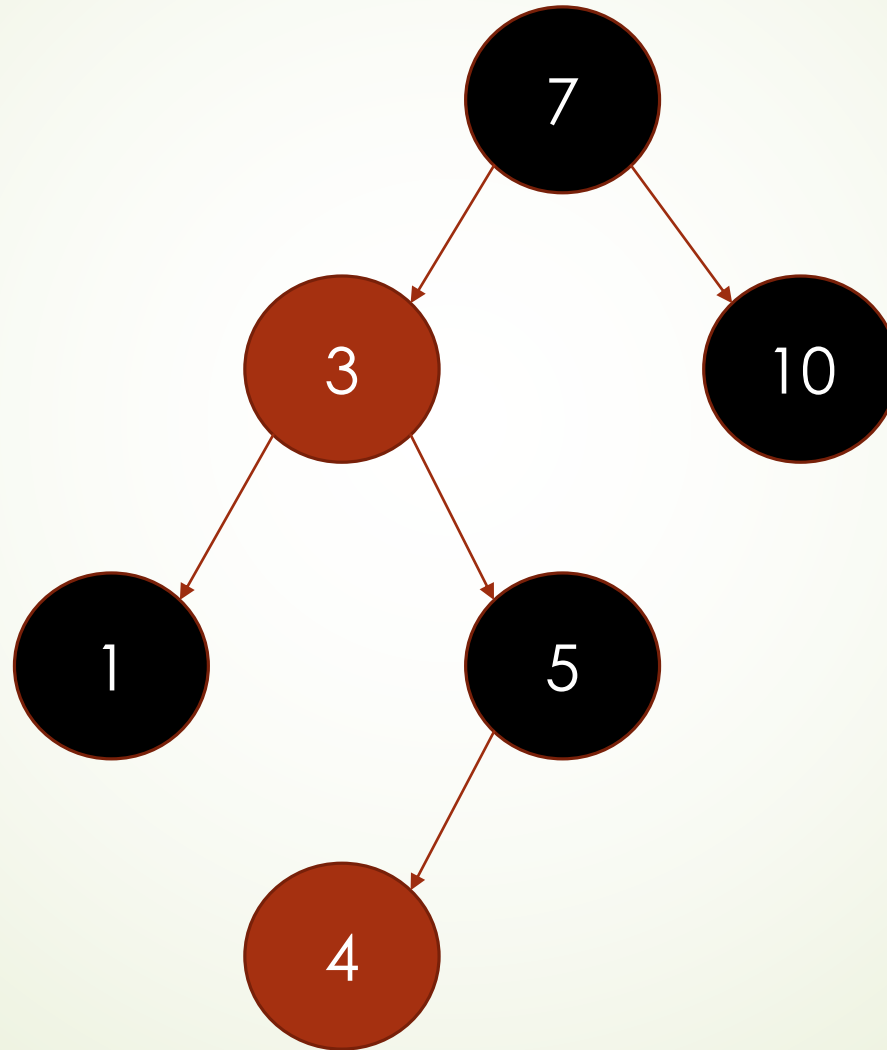
## Операция с деревом – вставка

- Вставляем **КРАСНЫЙ** узел
- Если это первый узел – перекрашиваем в ЧЁРНЫЙ
- Если родитель узла ЧЁРНЫЙ – ОК
- Если родитель и дядя – **КРАСНЫЕ**, делаем их ЧЁРНЫМИ, а дедушку – **КРАСНЫМ**

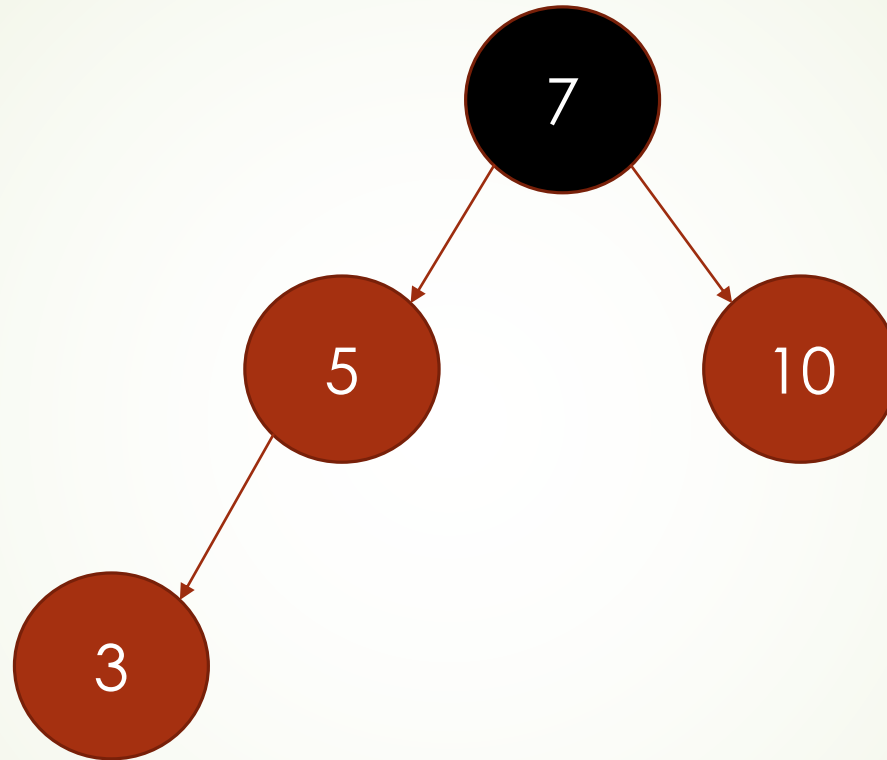
# Пример 1 (родитель и дядя красные)



# Пример 1 (родитель и дядя красные)

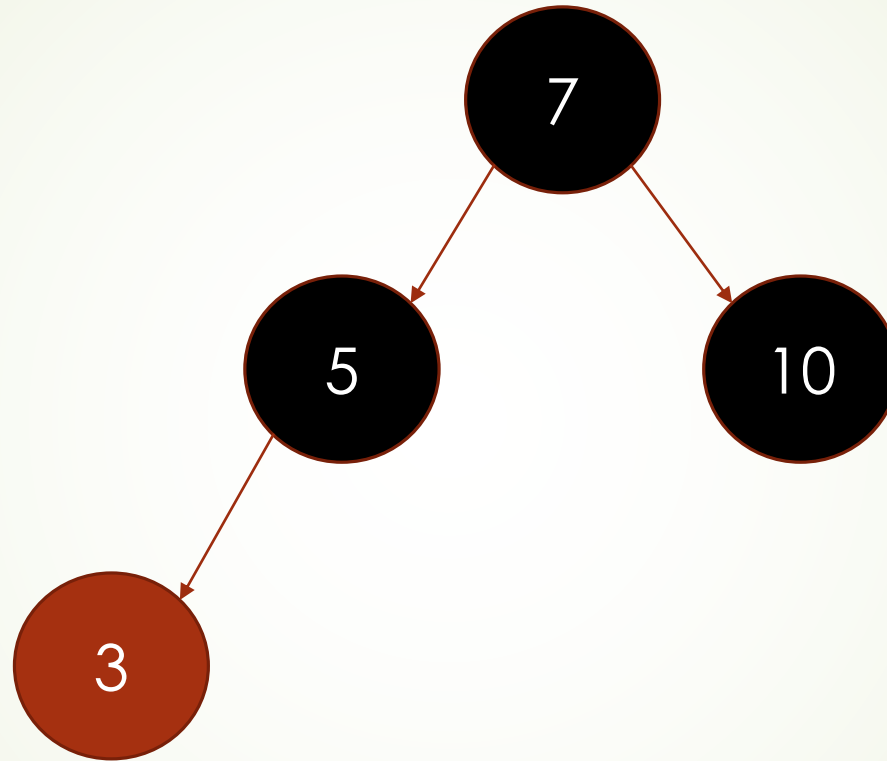


## Пример 2 (родитель и дядя красные)





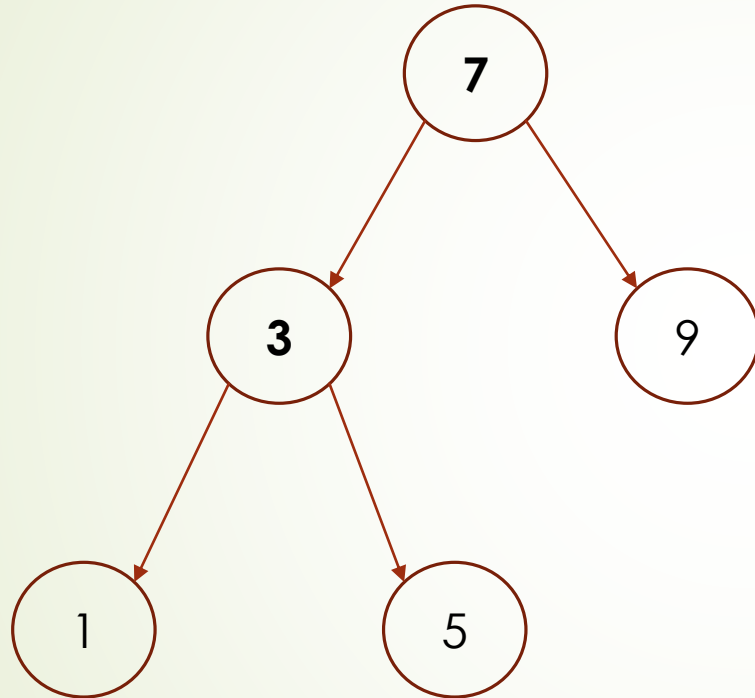
## Пример 2 (родитель и дядя красные)



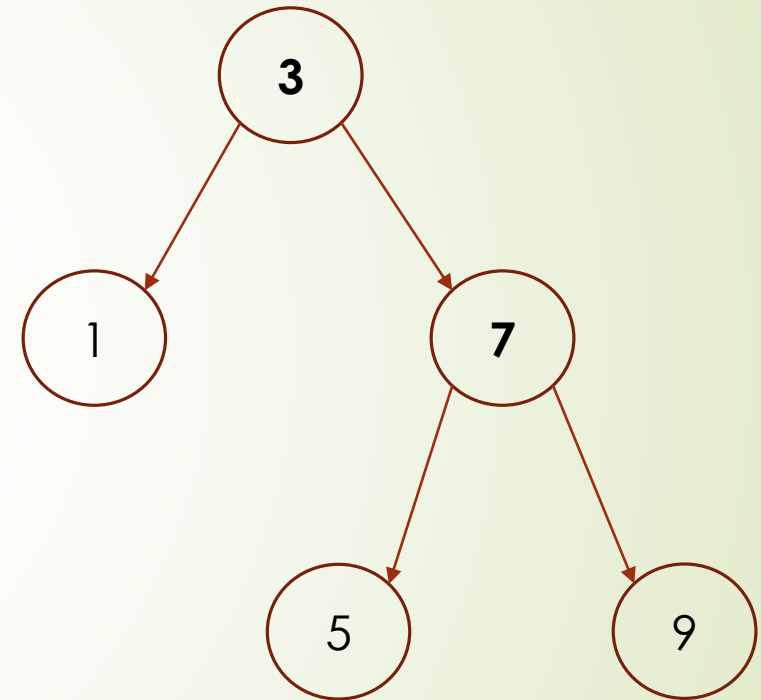
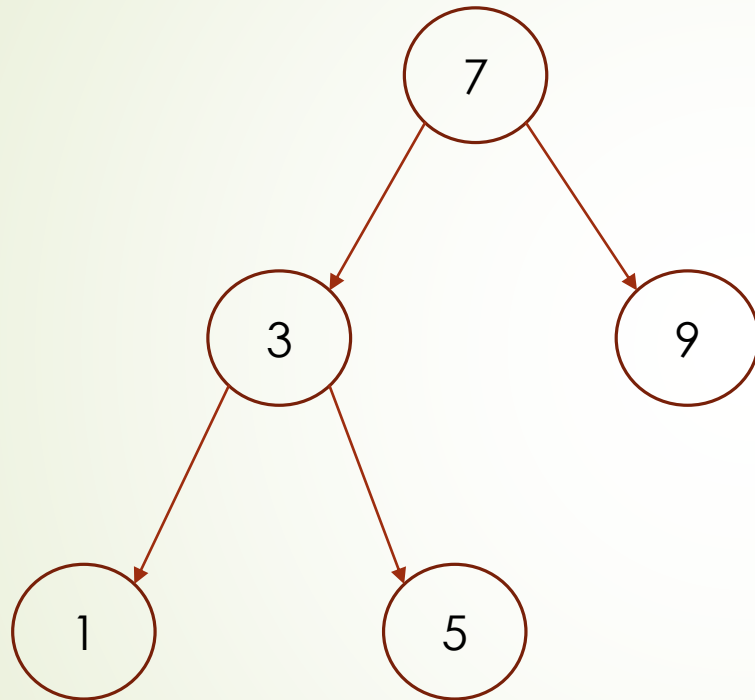
## Операция с деревом – вставка

- Вставляем **КРАСНЫЙ** узел
- Если это первый узел – перекрашиваем в ЧЁРНЫЙ
- Если родитель узла ЧЁРНЫЙ – ОК
- Если родитель и дядя – **КРАСНЫЕ**, делаем их ЧЁРНЫМИ, а дедушку – **КРАСНЫМ**
- **Родитель – КРАСНЫЙ**, дядя – отсутствует -- повороты

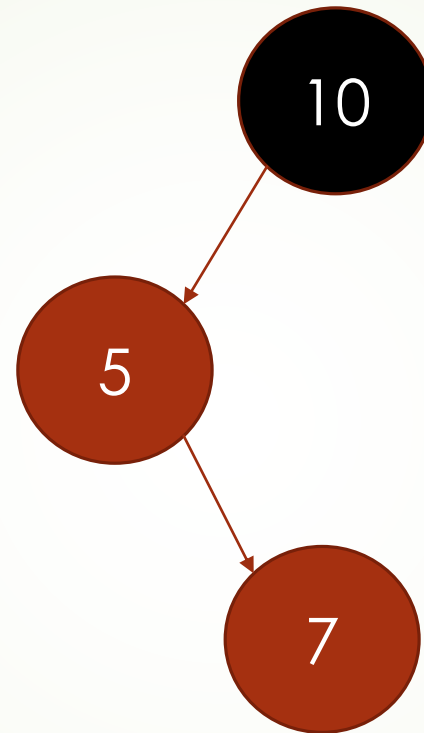
# Операция с деревом – поворот



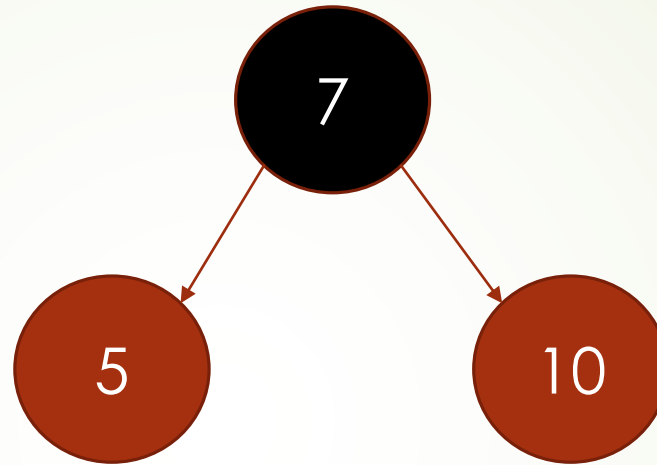
# Операция с деревом – поворот



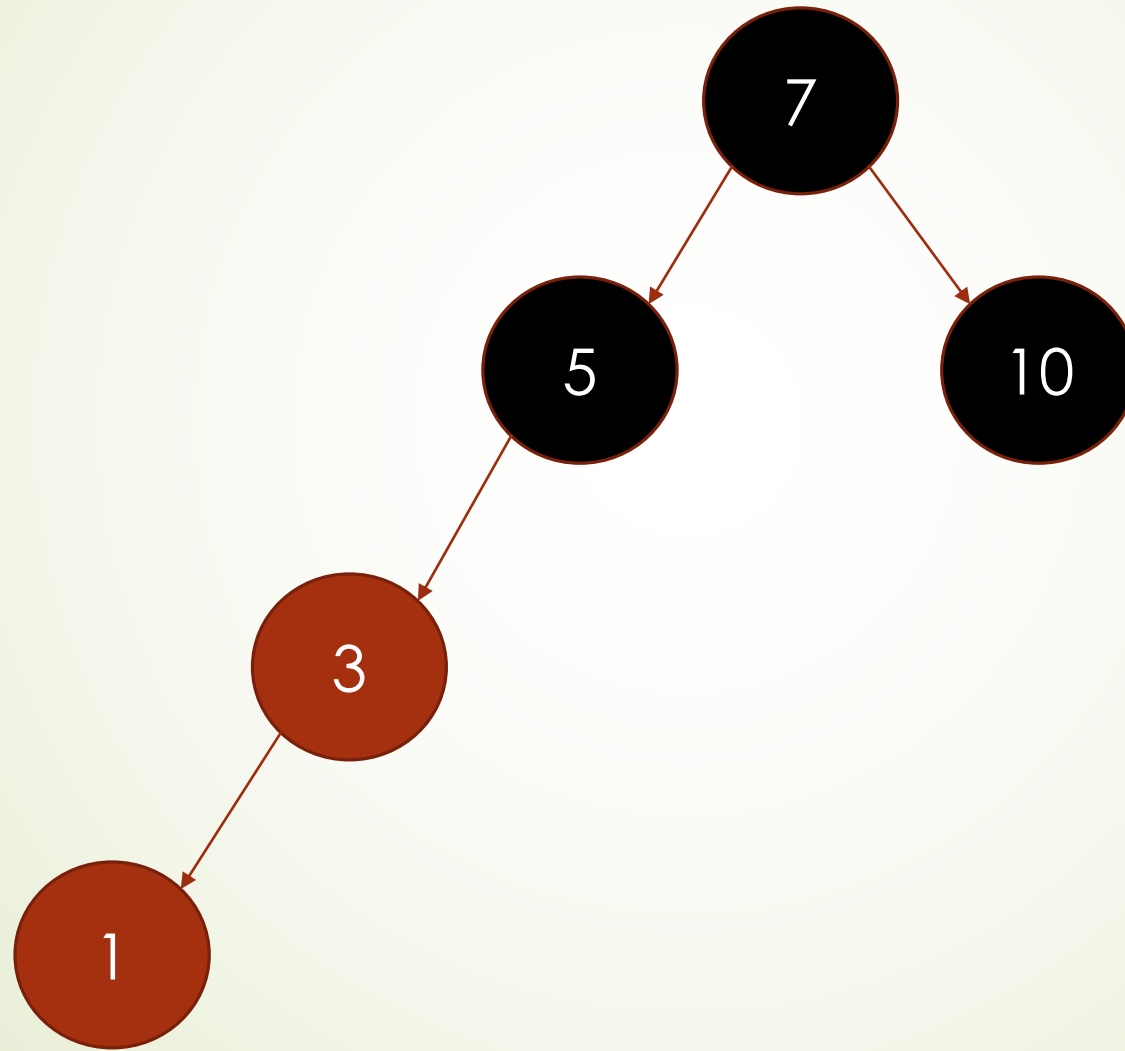
# Пример 1 (родитель красный, дяди нет)



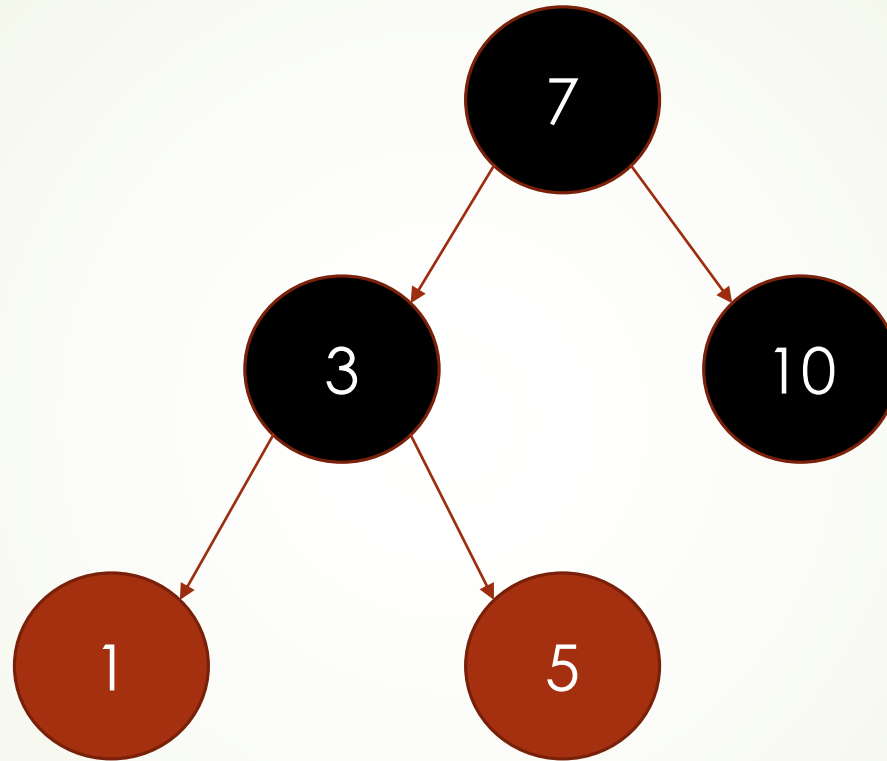
# Пример 1 (родитель красный, дяди нет)



## Пример 2 (родитель красный, дяди нет)



## Пример 2 (родитель красный, дяди нет)





# Красно-чёрные деревья

► Визуализация:

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

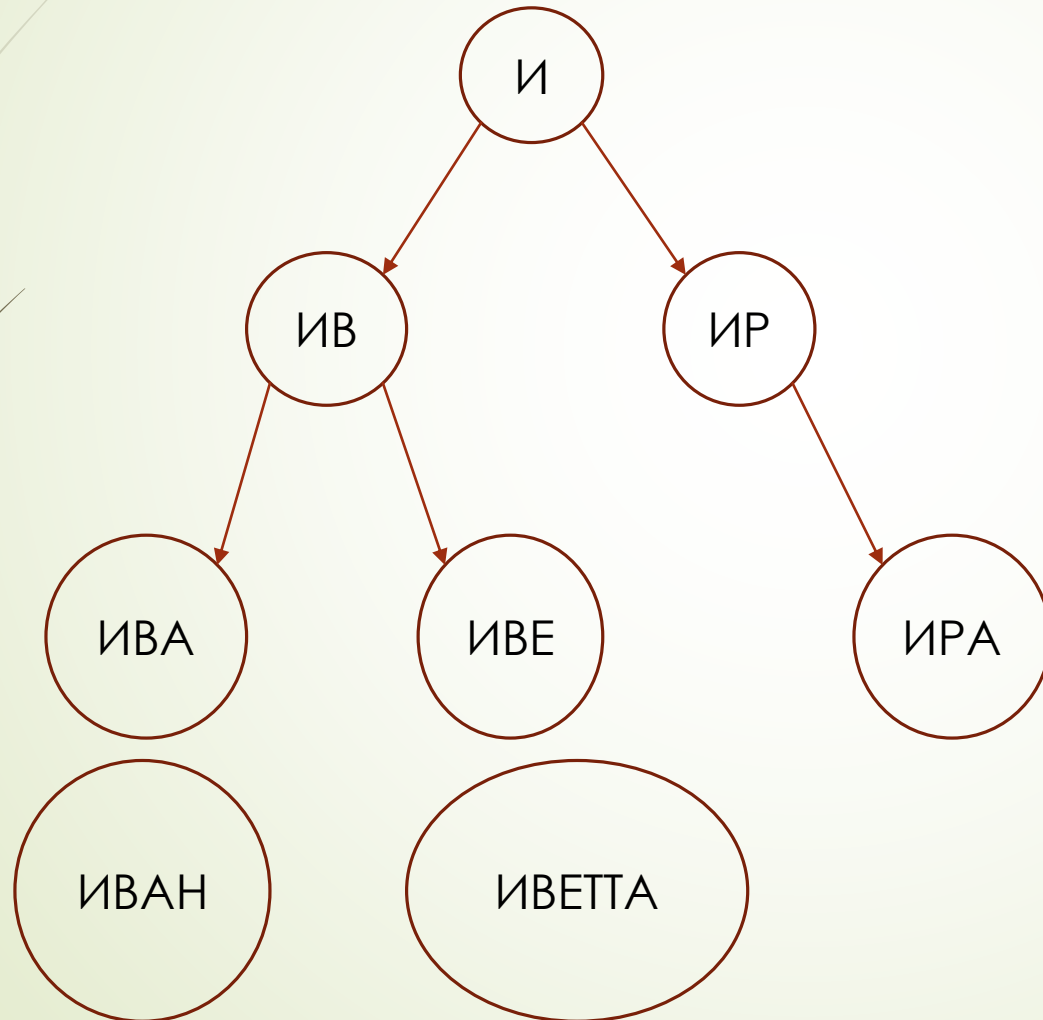
## Возможный интерфейс красно-чёрного узла

```
public interface RBNode<T> {  
    T getValue();  
    default boolean isRed() {  
        return true;  
    }  
    RBNode<T> getLeft();  
    RBNode<T> getRight();  
}
```

# Другие сбалансированные деревья

- AVL-дерево
  - Высоты разных ветвей отличаются максимум на 1
- B-дерево
  - Каждая вершина содержит список узлов, и на одного потомка больше, чем есть в этом списке (то есть не бинарное, а N-арное)
- Splay-дерево
  - Само-балансируется через операции Splay (перемещение вершины в корень), Split (разделение дерева на две части), Merge (соединение двух деревьев)

# Trie (префиксное дерево)



# Trie (применение)

- Префиксный поиск по словарю
- Поиск по строкам / спискам
- Трудоёмкость операций:  $O(\text{maxLength})$
- См. Trie в обучающем проекте (урок 3)

# Итоги

- Рассмотрено
  - Бинарные деревья поиска
    - Красно-чёрные
    - AVL, B-деревья
  - Префиксные деревья
- Далее
  - Таблицы и хэш-таблицы