План 7 Лекции

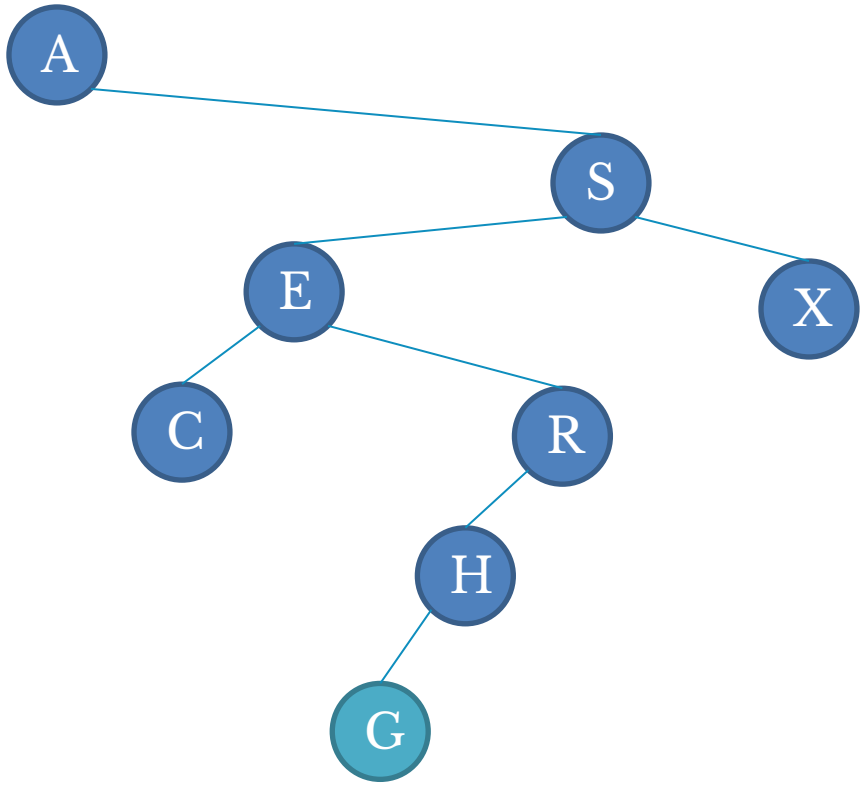
- Бинарные деревья поиска
- Вставка узла в корень БДП
- Рандомизированные БДП
- 2-3-4 деревья (2-3 деревья)
- Сбалансированные бинарные деревья
- Красно-черные деревья
- AVL-деревья

Бинарные деревья поиска

- Каждый узел является объектом с атрибутами:
  - Key
  - Left
  - Right
  - P (Parent) (опционально!!!)
    - Необходим только при “классической реализации удаления”
    - Существует альтернативное удаление без использования знания о родителе (путем слияния двух деревьев в одно).

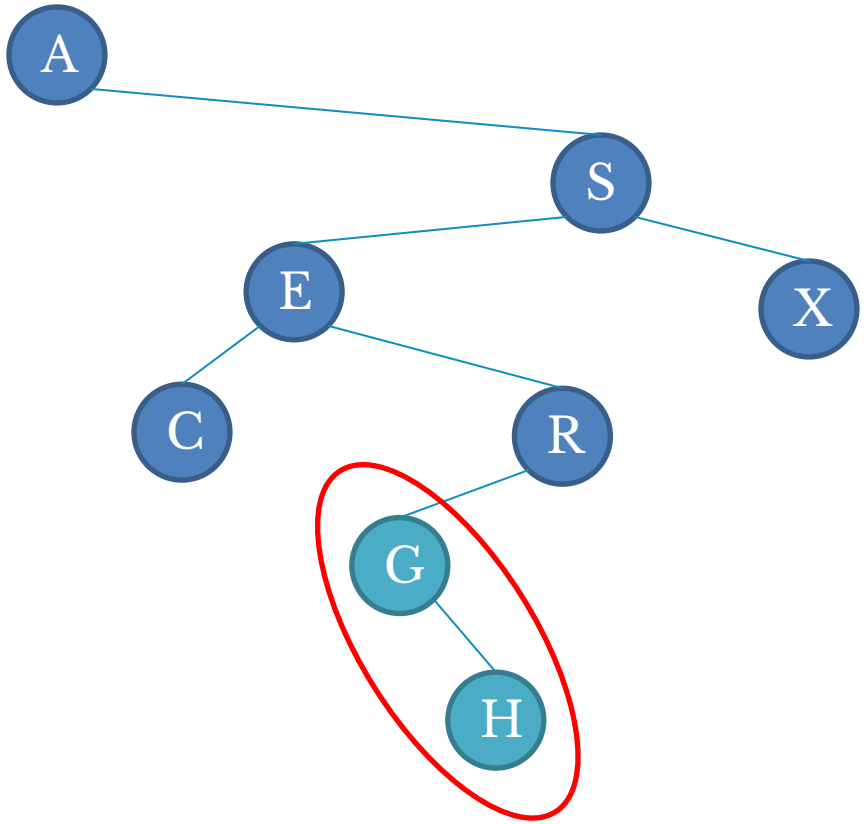
Вставка узла в корень БДП

- Идея: вставка элемента по классическому алгоритму. Далее, путем поворотов переносим узел с элементом в корень.



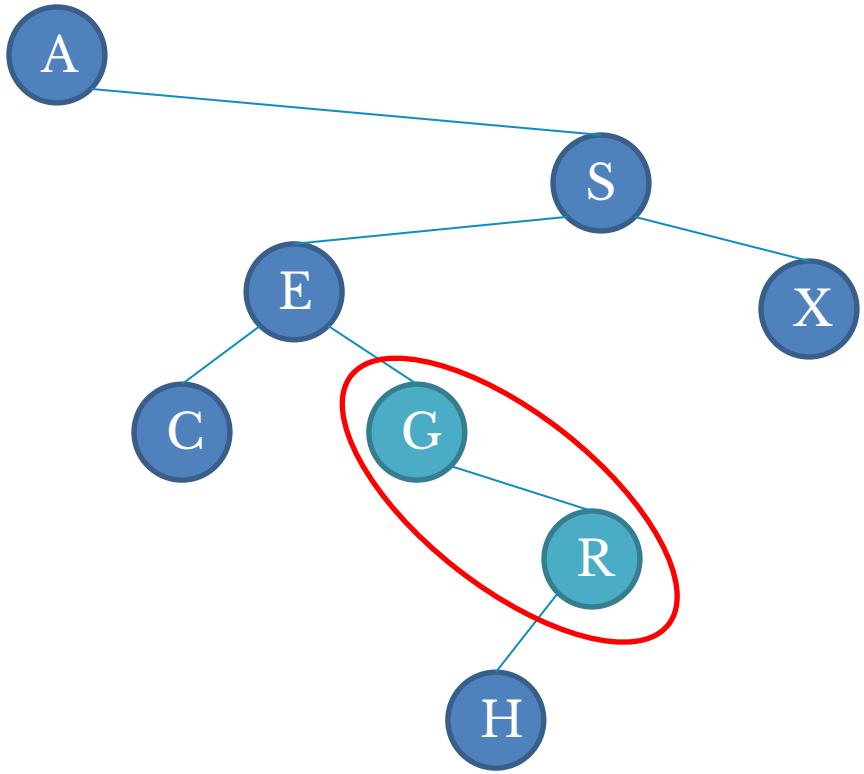
Вставка узла в корень БДП

- Идея: вставка элемента по классическому алгоритму. Далее, путем поворотов переносим узел с элементом в корень.



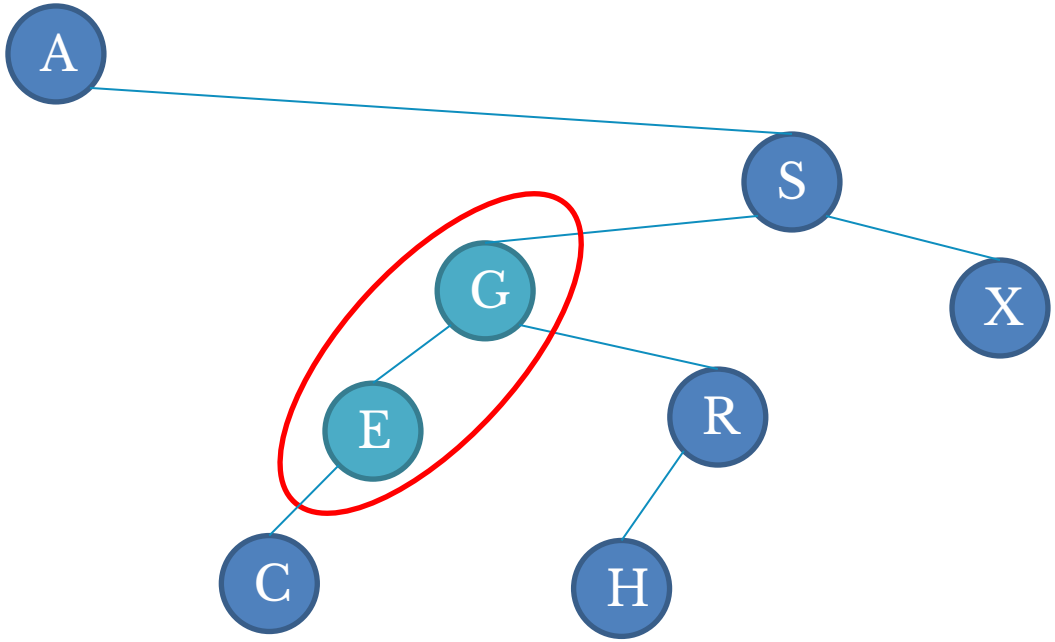
Вставка узла в корень БДП

- Идея: вставка элемента по классическому алгоритму. Далее, путем поворотов переносим узел с элементом в корень.



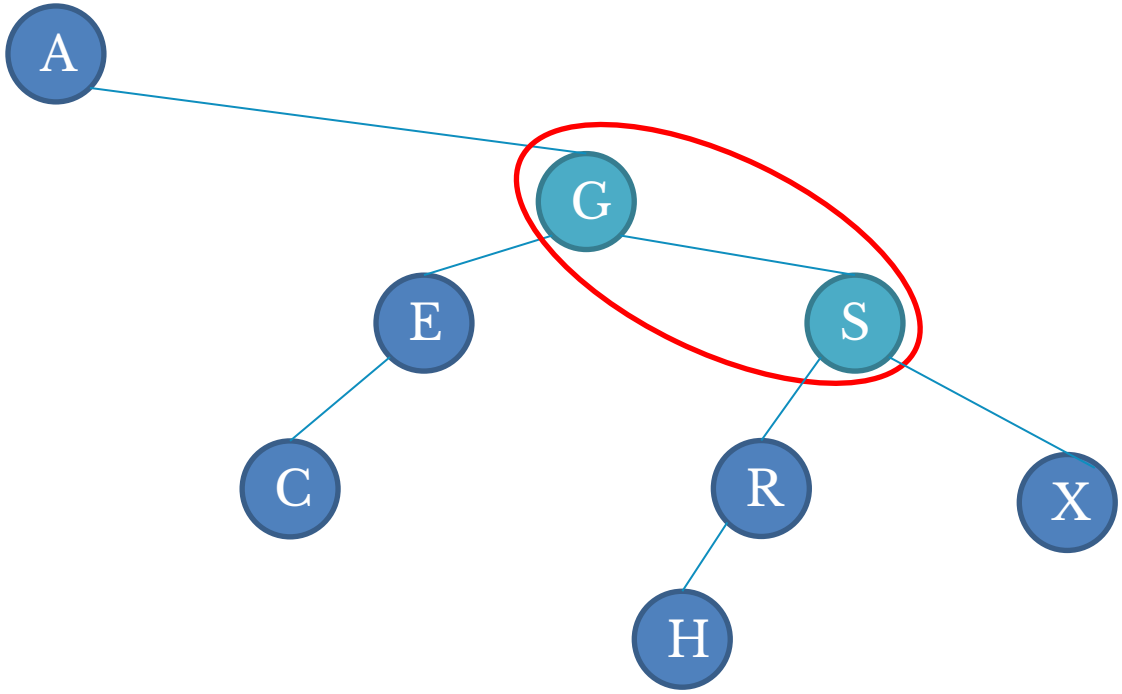
Вставка узла в корень БДП

- Идея: вставка элемента по классическому алгоритму. Далее, путем поворотов переносим узел с элементом в корень.



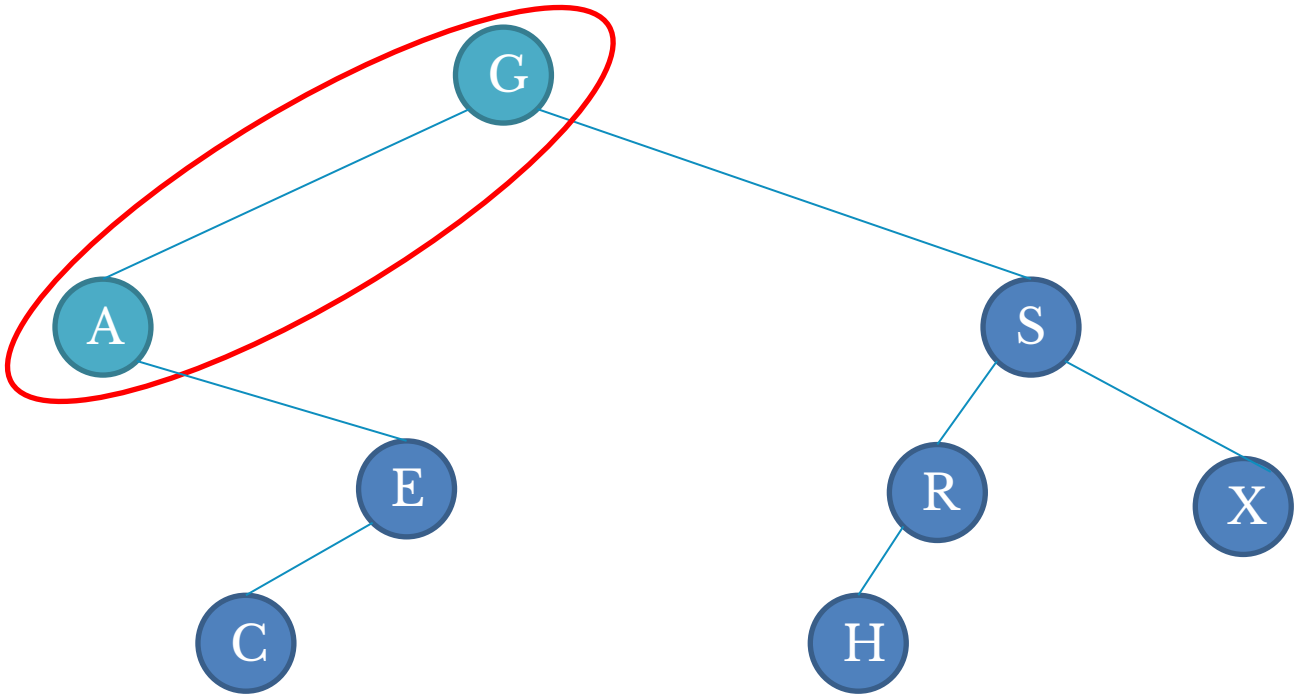
Вставка узла в корень БДП

- Идея: вставка элемента по классическому алгоритму. Далее, путем поворотов переносим узел с элементом в корень.



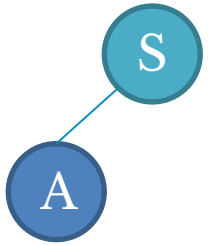
Вставка узла в корень БДП

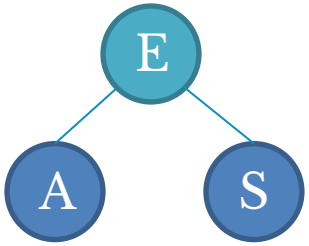
- Идея: вставка элемента по классическому алгоритму. Далее, путем поворотов переносим узел с элементом в корень.

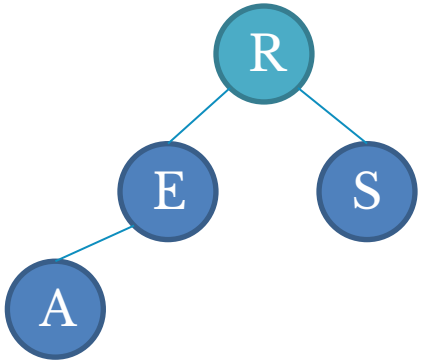


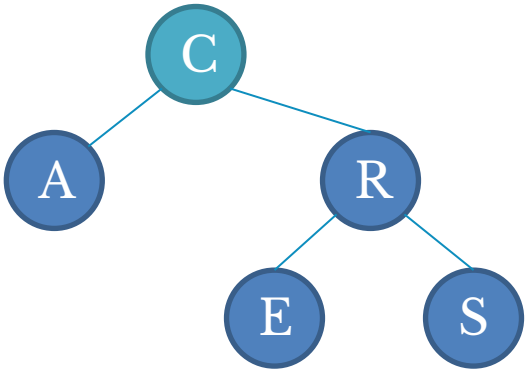
Вставка узла в корень БДП

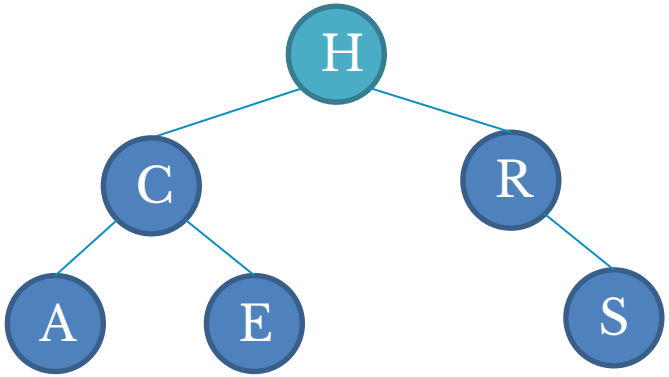
A S E R C H









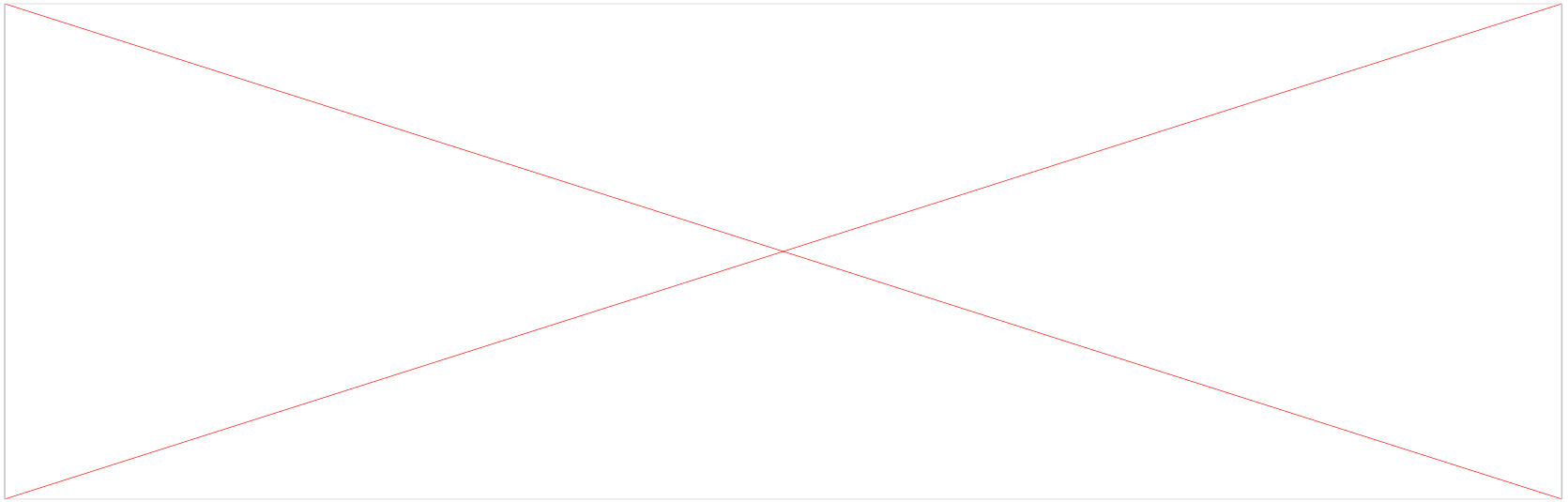


Рандомизированные БДП

- Идея: при вставке нового узла в дерево из N узлов вероятность появления нового узла в корне должна быть равна $1/(N+1)$. Поэтому при вставке нужно принять случайное решение использовать вставку в корень с этой вероятностью.

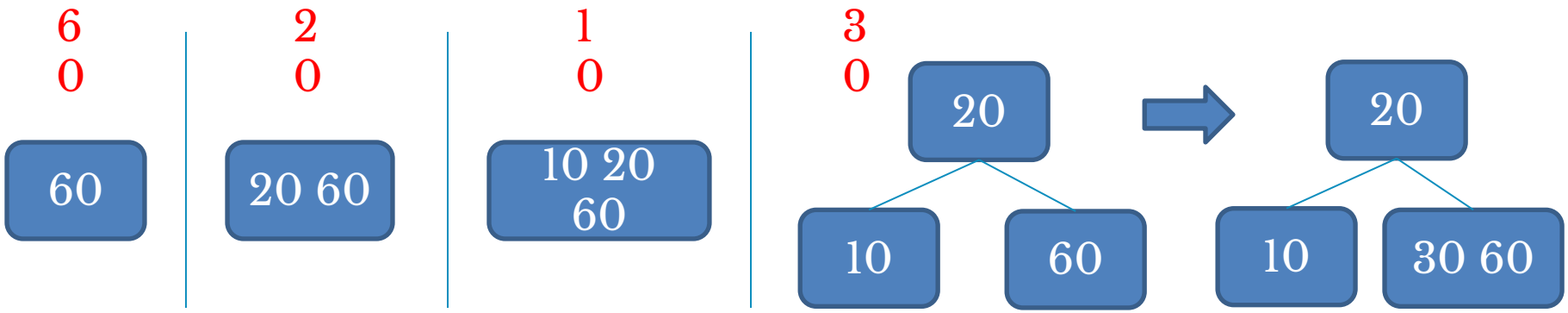
2-3-4 деревья

- 2-3-4 дерево поиска – это дерево содержащее 3 типа узлов:
- 2-узлы: с одним ключом и двумя ссылками
- 3-узлы: с двумя ключами и тремя ссылками
- 4-узлы:с тремя ключами и четырьмя ссылками



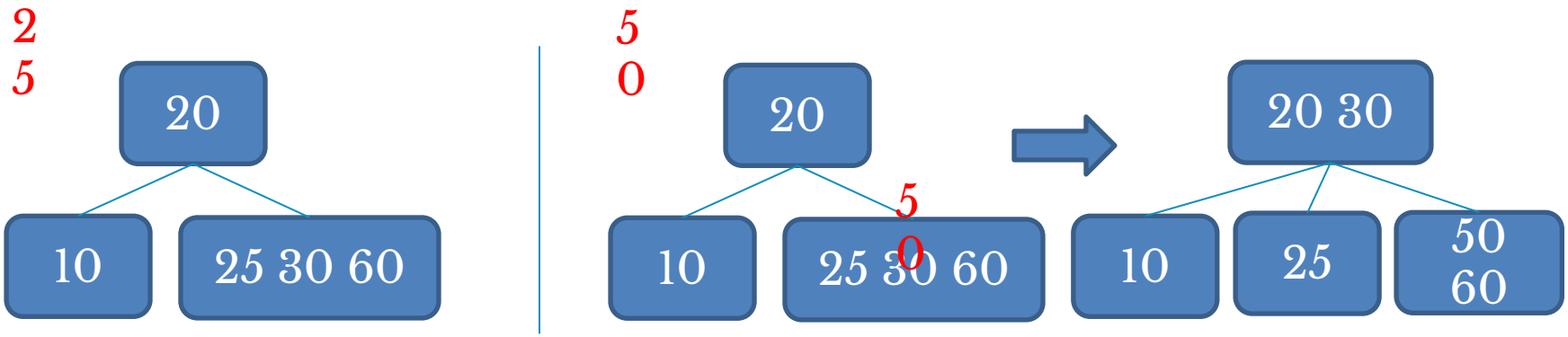
2-3-4 деревья

- Идея: При поиске потенциального родительского узла, при приходе вниз разделять все 4-узлы
- 1. Если узел – 4-узел
- Разделить 3 значения на левый и правый 2-узла, сделав при этом центральный элемент родителем этих узлов.
- Продолжить поиск
- 2a. Если узел – лист – простая вставка
- 2b. Продолжать поиск в дочерних узлах.
- Пример: 60, 20, 10, 30, 25, 50, 80



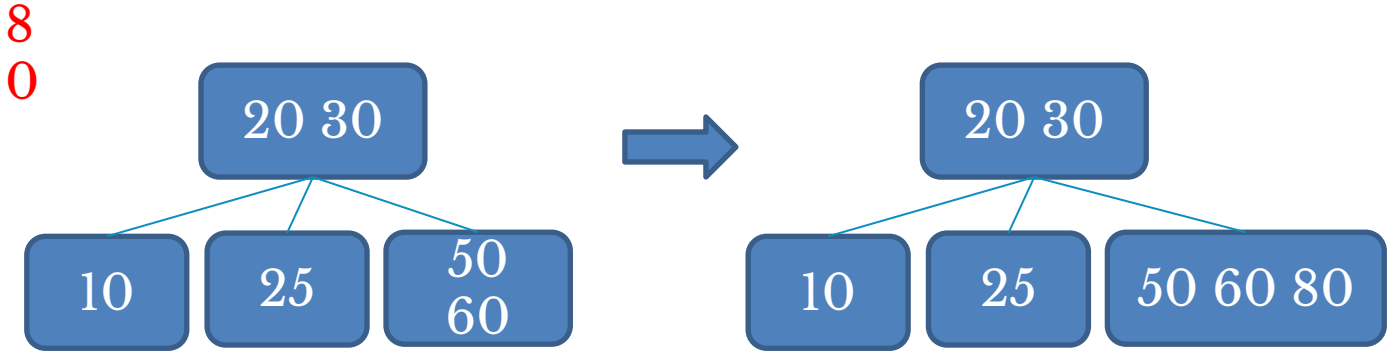
2-3-4 деревья

- Идея: При поиске потенциального родительского узла, при приходе вниз разделять все 4-узлы
- 1. Если узел – 4-узел
- Разделить 3 значения на левый и правый 2-узла, сделав при этом центральный элемент родителем этих узлов.
- Продолжить поиск
- 2a. Если узел – лист – простая вставка
- 2b. Продолжать поиск в дочерних узлах.
- Пример: 60, 20, 10, 30, 25, 50, 80



2-3-4 деревья

- Идея: При поиске потенциального родительского узла, при приходе вниз разделять все 4-узлы
- 1. Если узел – 4-узел
- Разделить 3 значения на левый и правый 2-узла, сделав при этом центральный элемент родителем этих узлов.
- Продолжить поиск
- 2a. Если узел – лист – простая вставка
- 2b. Продолжать поиск в дочерних узлах.
- Пример: 60, 20, 10, 30, 25, 50, 80



Сбалансированные деревья

- В наихудшем случае производительность бинарного дерева поиска не лучше, чем производительность связного списка
- Сбалансированное дерево — дерево поиска, высота которого равна $O(\lg n)$

В наихудшем случае производительность бинарного дерева поиска не лучше, чем производительность связного списка

Сбалансированное дерево — дерево поиска, высота которого равна $O(\lg n)$

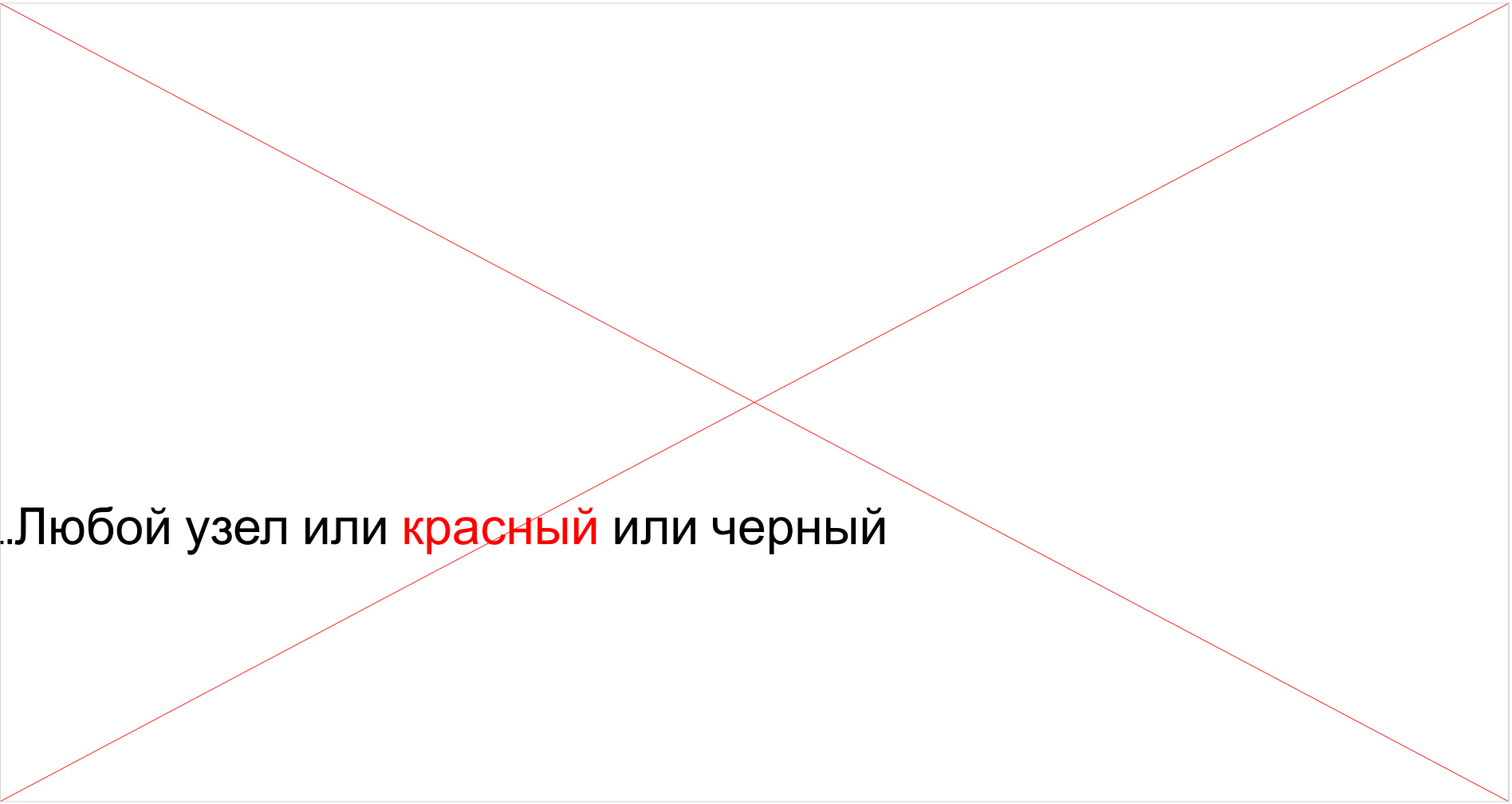
Красно черные деревья

- Требуется в каждом узле информация о цвете узла
- RB-tree = BST tree + 4 свойств:
1. Любой узел или красный или черный
2. Корень и листья (NIL) дерева – черные
3. Если узел красный, то оба его дочерних узла черные
4. Для любого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

Требуется в каждом узле информация о цвете узла

- *RB-tree = BST tree + 4 свойств:*
  1. *Любой узел или красный или черный*
  2. *Корень и листья (NIL) дерева – черные*
  3. *Если узел красный, то оба его дочерних узла черные*
  4. *Для любого узла x все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов $= bh(x)$*

Красно-Черные деревья

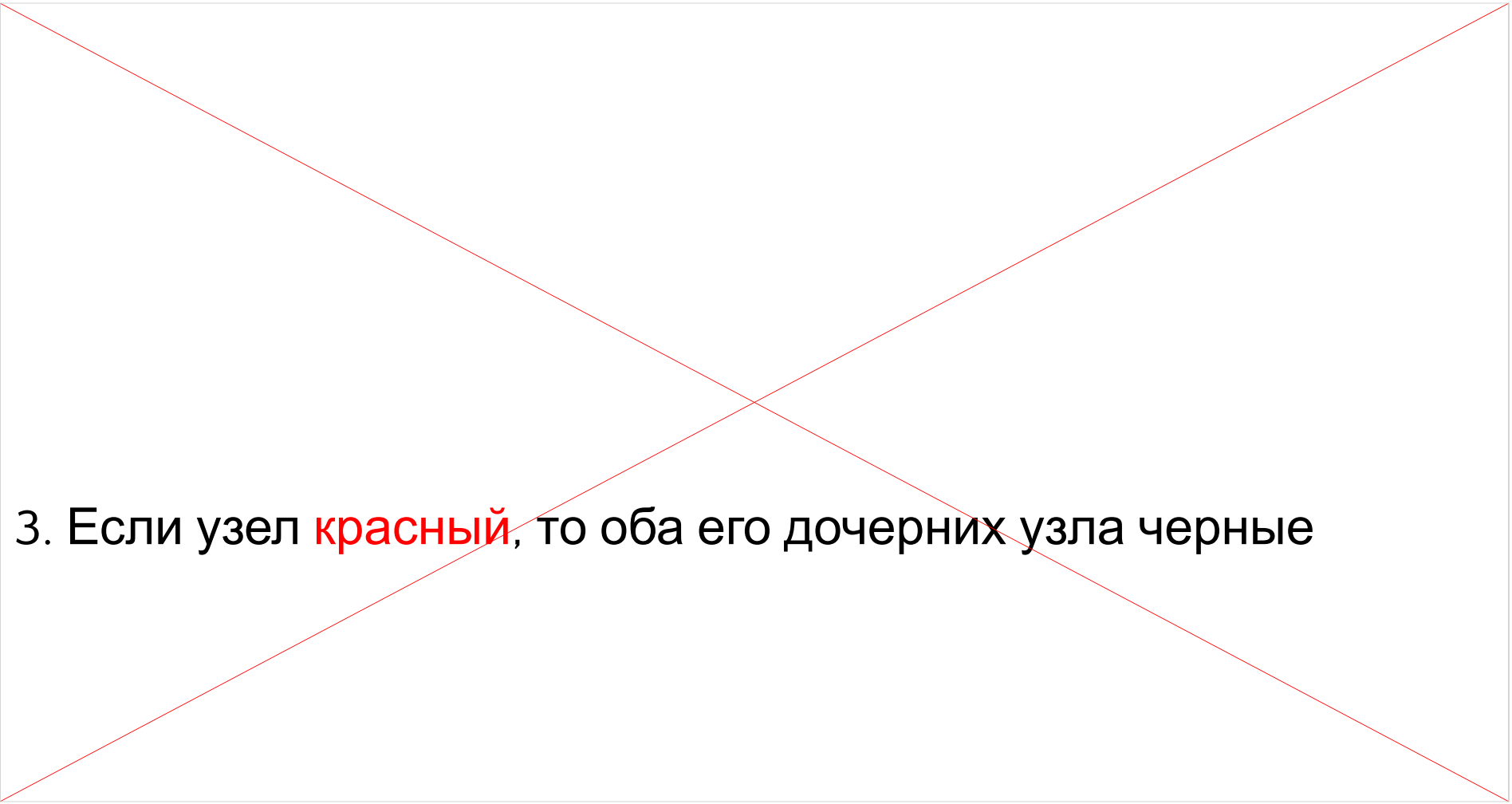


- Любой узел или красный или черный

Красно-Черные деревья

2. Корень и листья (NIL) дерева – черные

Красно-Черные деревья



3. Если узел красный, то оба его дочерних узла черные

Красно-Черные деревья

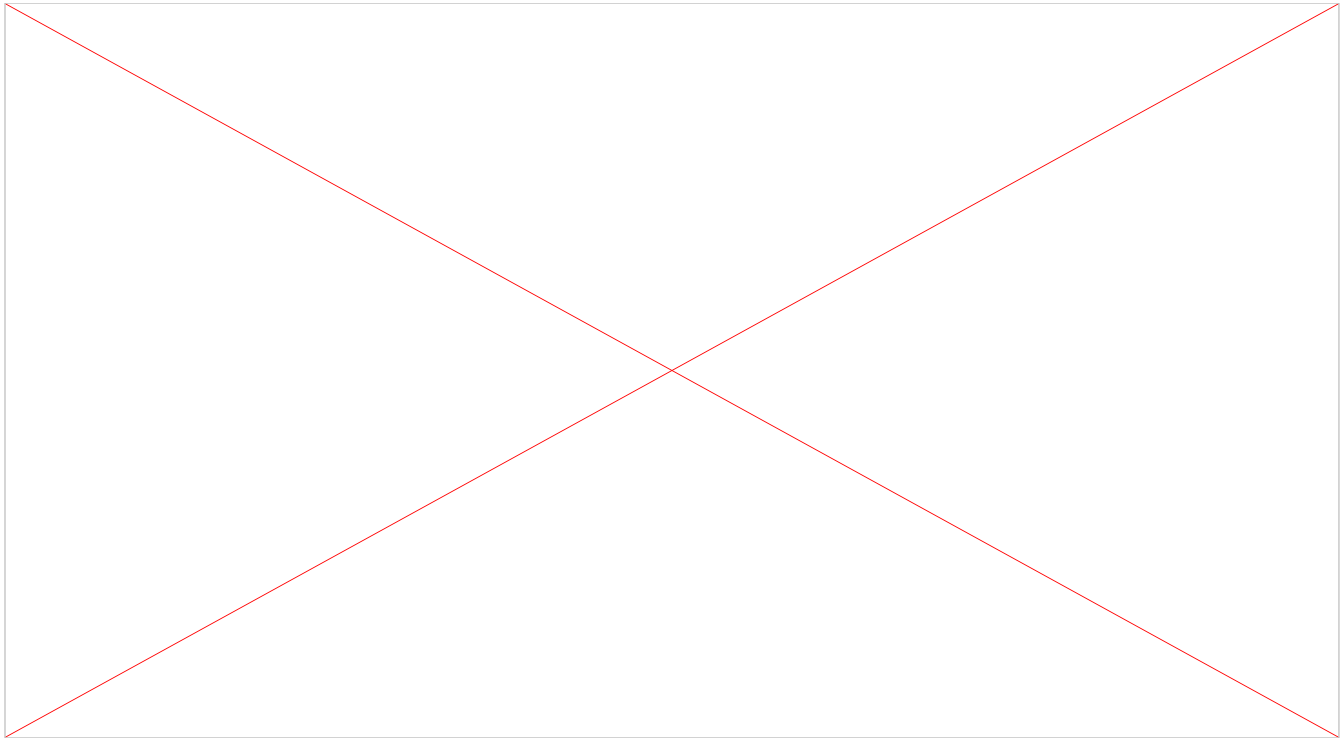
4. Для любого узла все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов

4. Для любого узла x все пути от него до листьев, являющихся потомками данного узла, содержат одно и то же количество черных узлов $= bh(x)$

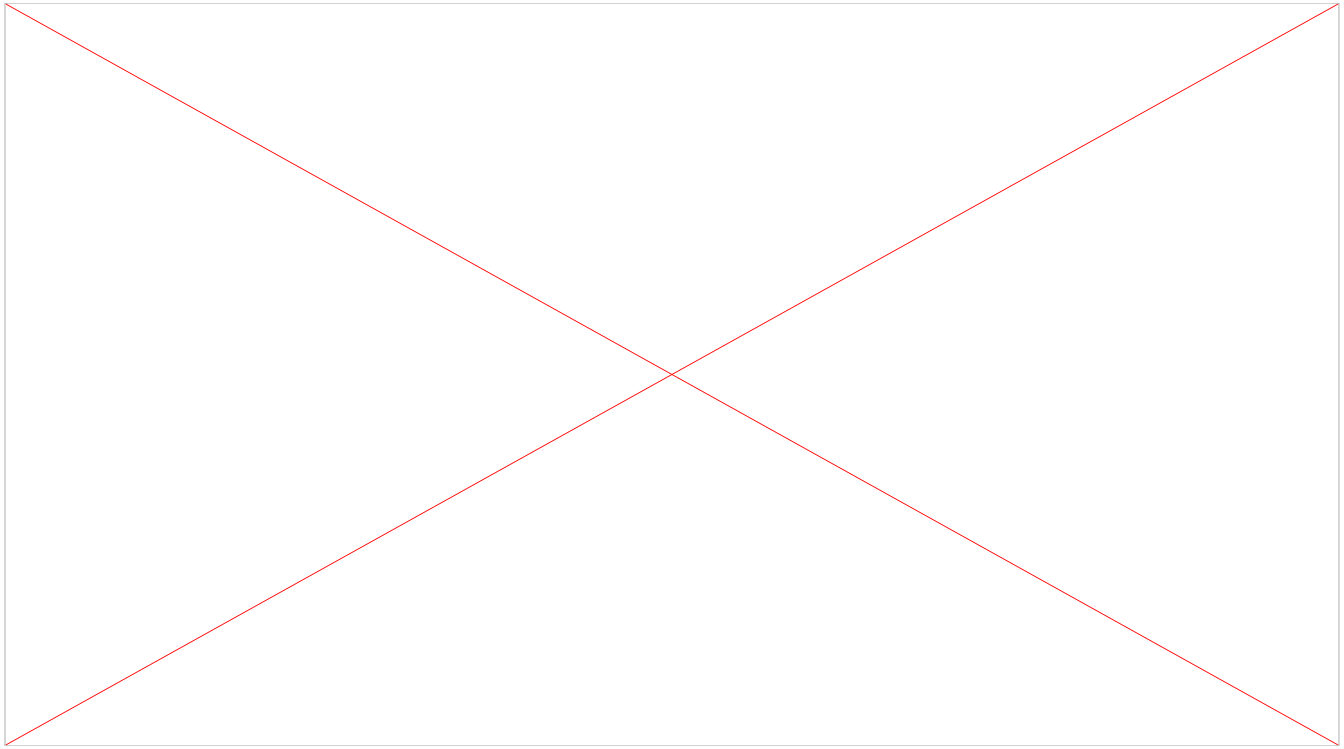
Высота Красно-Черного дерева

- Лемма
  - Высота RB-дерева с n внутренними узлами не более чем $2 * \lg(n+1)$
- Лемма
  - Поддерево любого узла x содержит как минимум $2^{bh(x)} - 1$ внутренних узлов

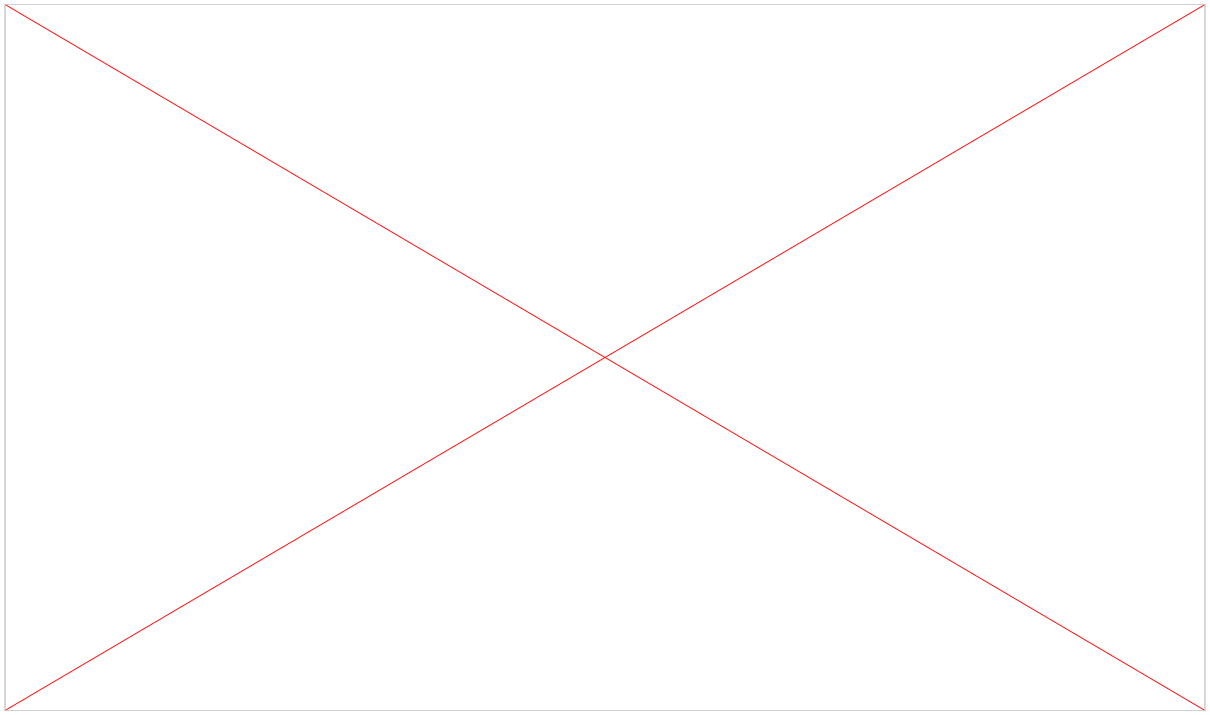
Красно-Черные деревья



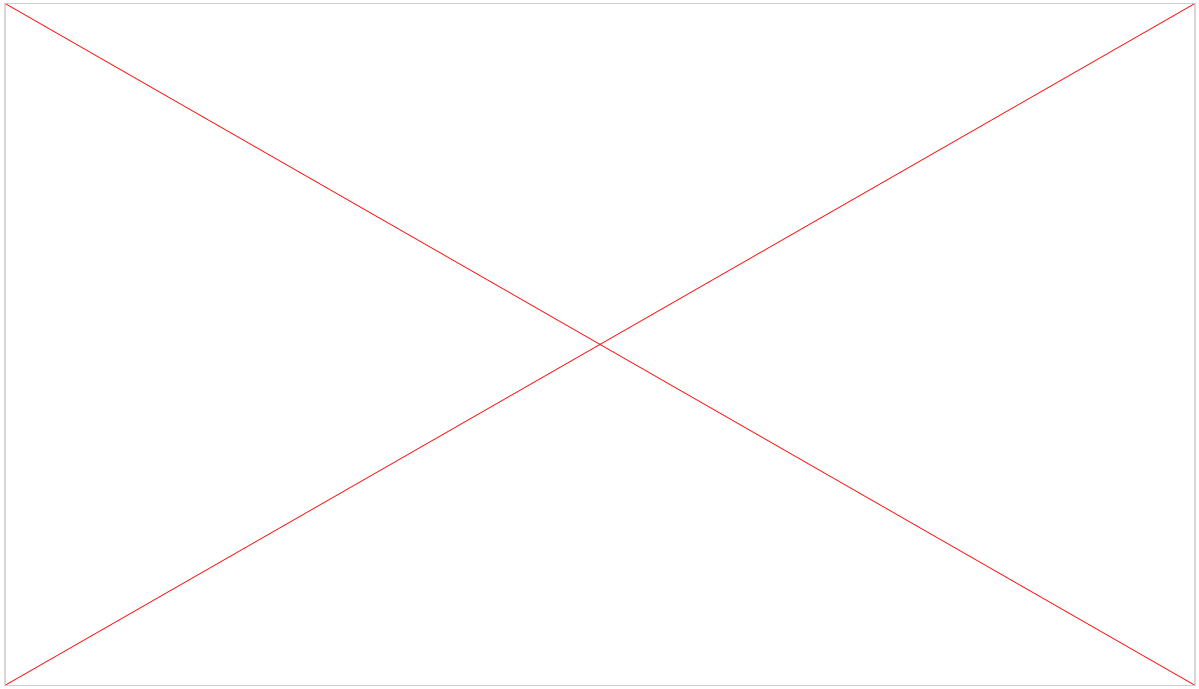
Красно-Черные деревья



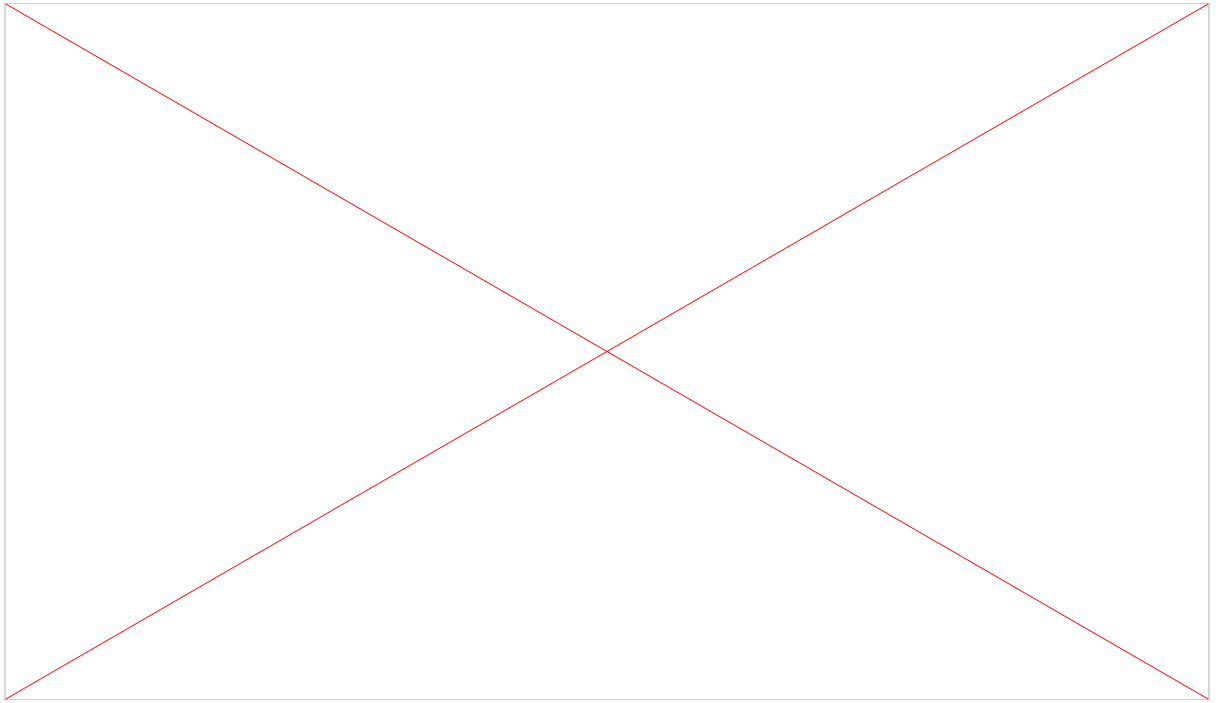
Красно-Черные деревья



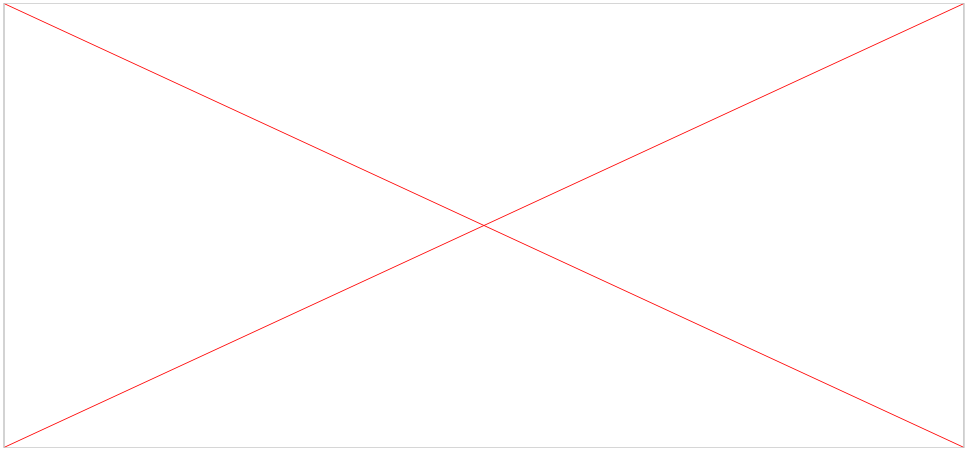
Красно-Черные деревья



Красно-Черные деревья



Красно-Черные деревья



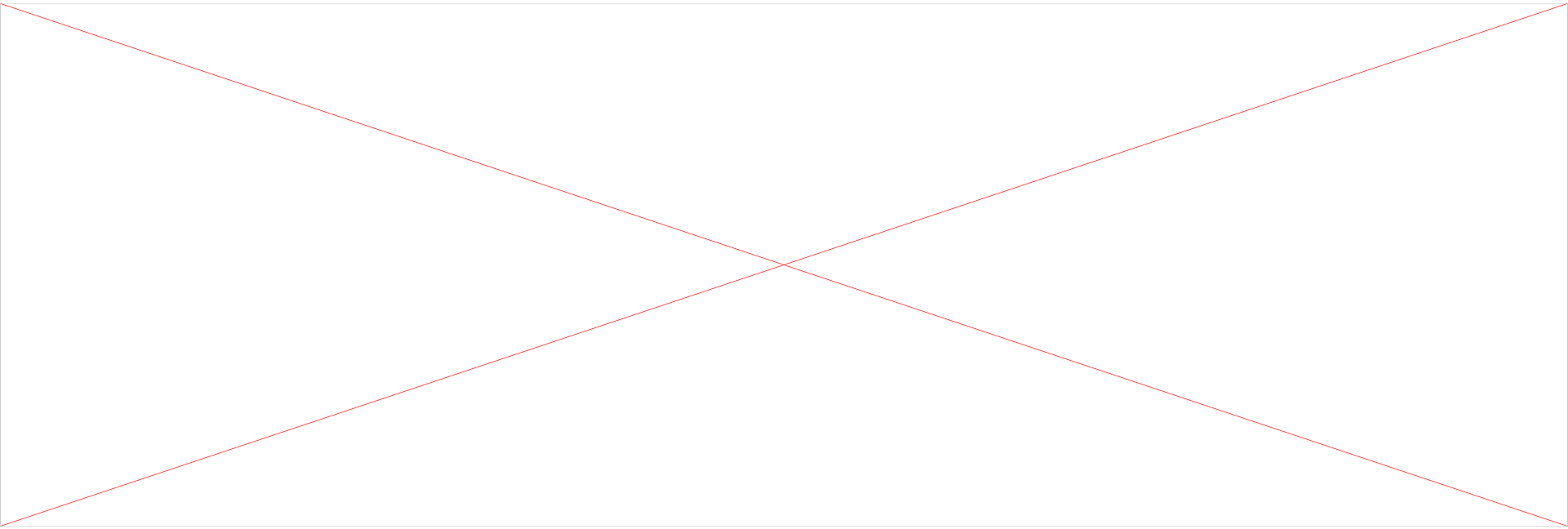
Запросы к красно-черному дереву

- Запросы Search, Min, Max, Successor, Predecessor выполняются за $O(\lg n)$ в RB-tree с n узлами.

Вставка в красно-черное дерево

- Операции INSERT и DELETE приводят к изменению RB-tree:
- Операция сама по себе
- Изменение цвета узла
- Перестройка дерева (через повороты)

Повороты



Вставка в красно-черное дерево

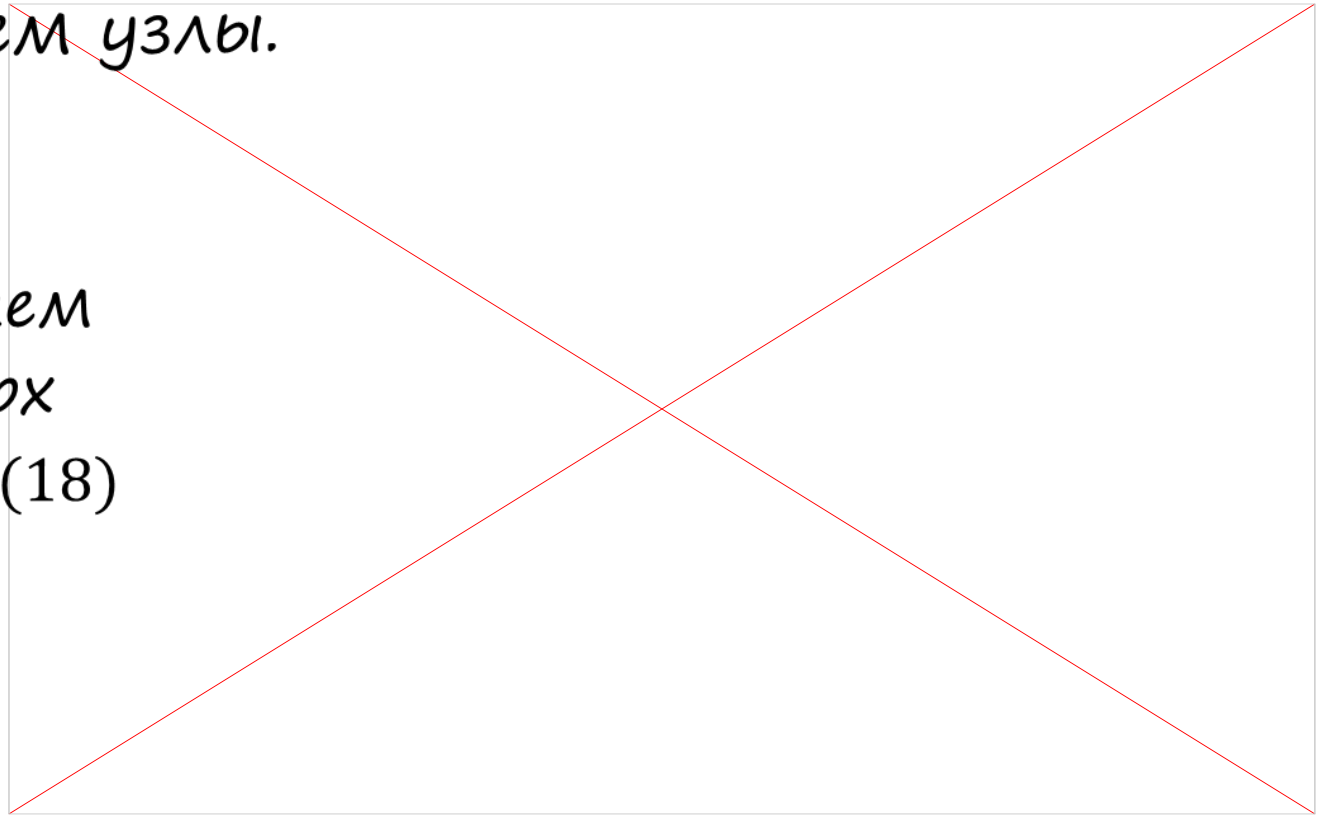
- Идея: Добавляем элемент в дерево и окрашиваем в красный цвет (могут нарушаться 2-е и 3-е). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.
- Пример:
  - Пример:

Вставка в красно-черное дерево

- Идея: Добавляем элемент в дерево и перекрашиваем в красный цвет (могут нарушаться 2 и 3). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.
- Пример:
  - Перекрашиваем
  - Двигаем вверх
    - $x = 15$
    - *Перекрашиваем*
    - *Двигаем вверх*
    - $Right - Rotate(18)$

Идея: Добавляем элемент в дерево и перекрашиваем в красный цвет (могут нарушаться 2 и 3). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.

Пример:

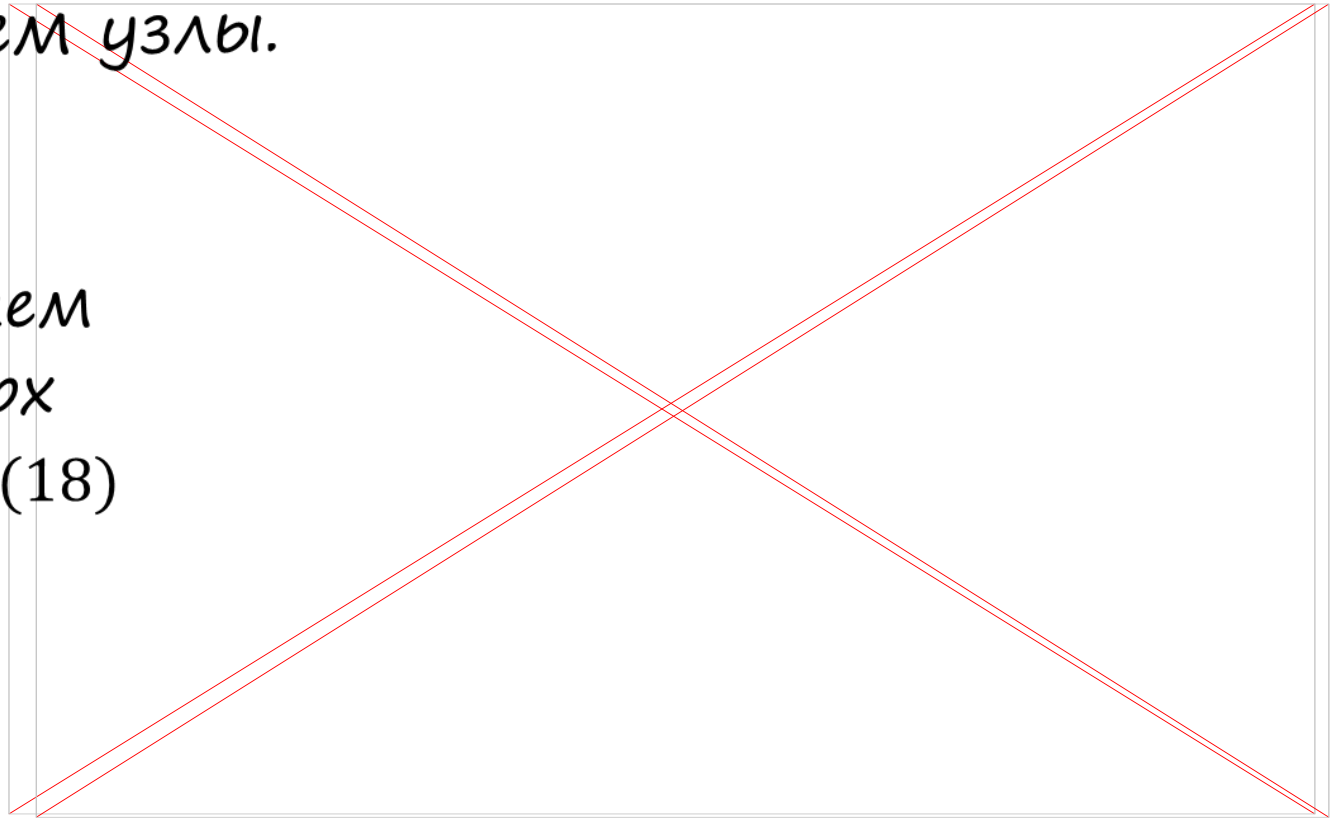


Вставка в красно-черное дерево

- Идея: Добавляем элемент в дерево и перекрашиваем в красный цвет (могут нарушаться 2 и 3). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.
- Пример:
  - Перекрашиваем
  - Двигаем вверх
    - $x = 15$
    - *Перекрашиваем*
    - *Двигаем вверх*
    - $Right - Rotate(18)$

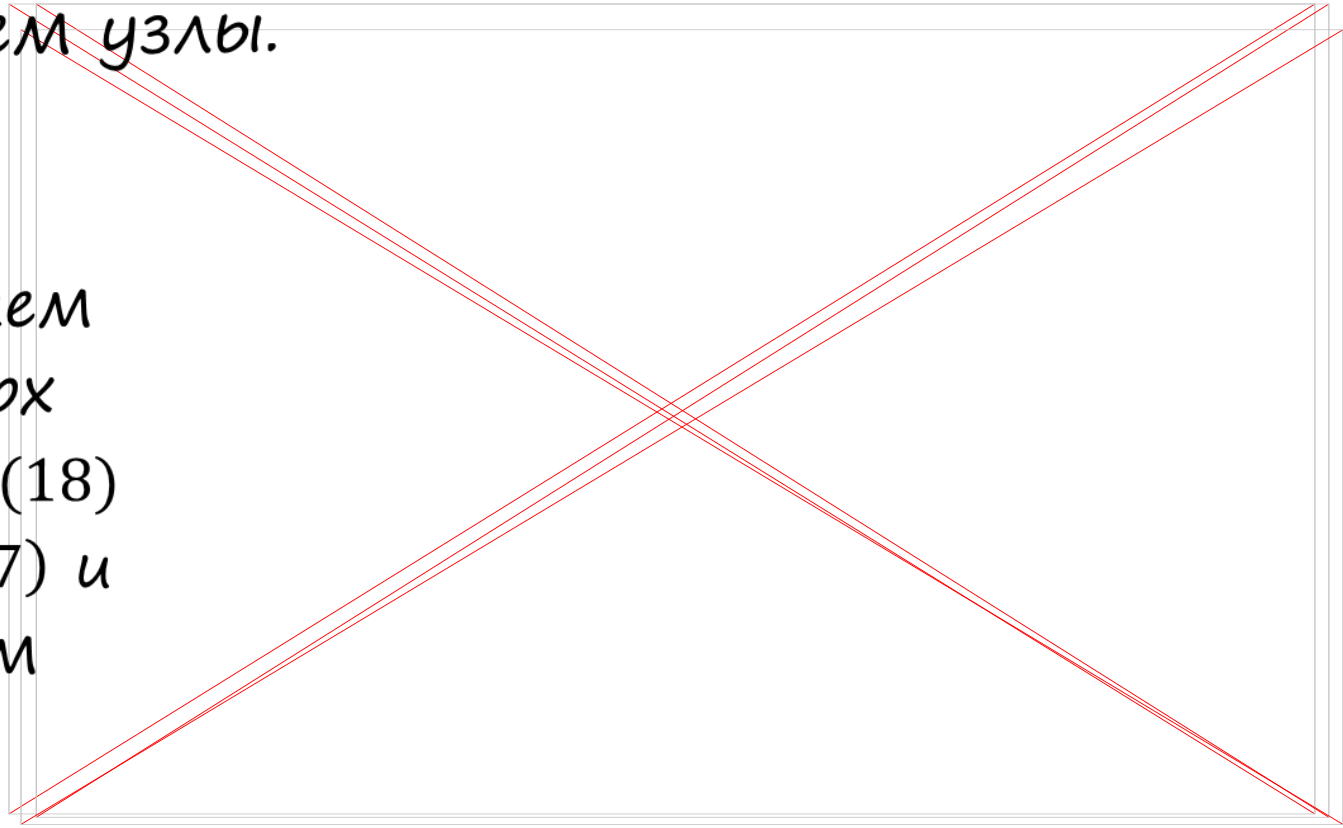
Идея: Добавляем элемент в дерево и перекрашиваем в красный цвет (могут нарушаться 2 и 3). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.

Пример:



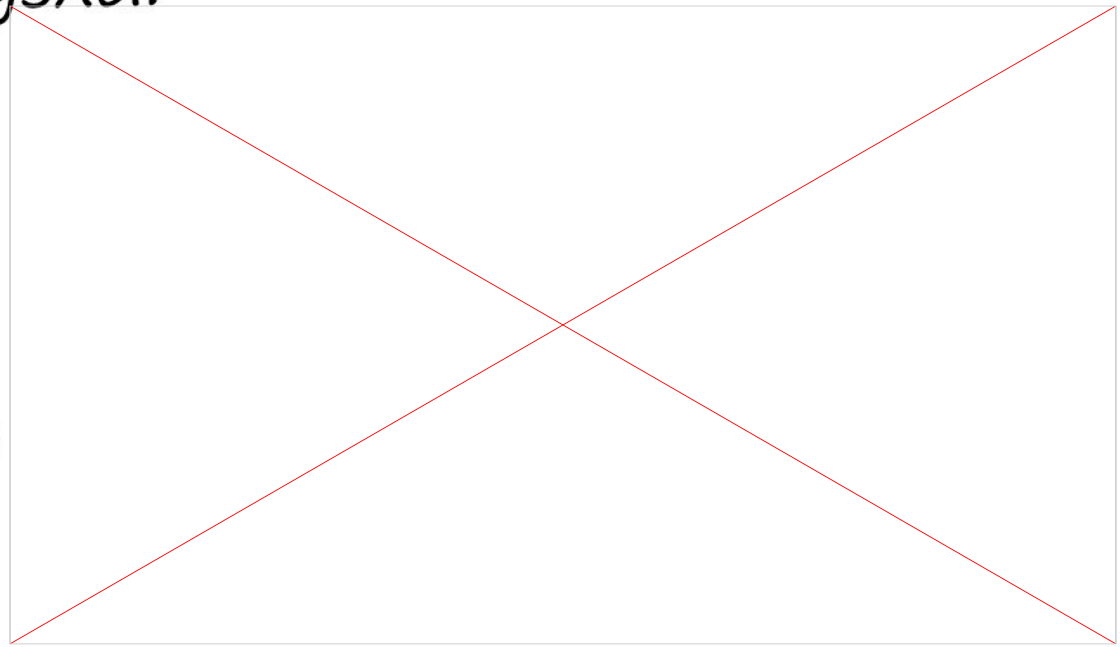
Вставка в красно-черное дерево

- Идея: Добавляем элемент в дерево и перекрашиваем в красный цвет (могут нарушаться 2 и 3). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.
- Пример:
  - Перекрашиваем
  - Двигаем вверх
  - и перекрашиваем
- Идея: Добавляем элемент в дерево и перекрашиваем в красный цвет (могут нарушаться 2 и 3). Пока не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.
- Пример:
  - Пример:
  - $x = 15$
    - Перекрашиваем
    - Двигаем вверх
  - $Right - Rotate(18)$
  - $Left - Rotate(7)$ и перекрашиваем



Вставка в красно-черное дерево

- Идея: Добавляем элемент x в дерево и окрашиваем в красный цвет (могут нарушатся 2 и 3). Пока x не вершина и имеет красного родителя перемещаем вверх элемент и перекрашиваем узлы.
- Пример:
  - $x = 15$
  - Перекрашиваем
  - Двигаем вверх
  - $Right - Rotate(18)$
  - $Left - Rotate(7)$ и перекрашиваем



Вставка в красно-черное дерево

- Возможны 3 случая при перестановках:
  - “Дядя” вершины красного цвета. Красим его и отца вершины в черный цвет, а “деда” в красный. Дед становится “иксом”
  - “Дядя” вершины черного цвета. Проверяем являются ли узел x и отец одинаковыми детьми (т.е. оба левые или правые). Если не являются, то делаем соответствующие вращения (левое или правое) (тем самым делая отца ребенком нового элемента). Дальнейший анализ делаем для бывшего отца.
    - Перестраиваем дерево, делая нового отца корнем данного поддерева, делая деда ребенком отца. красим отца в черный, деда в красный.

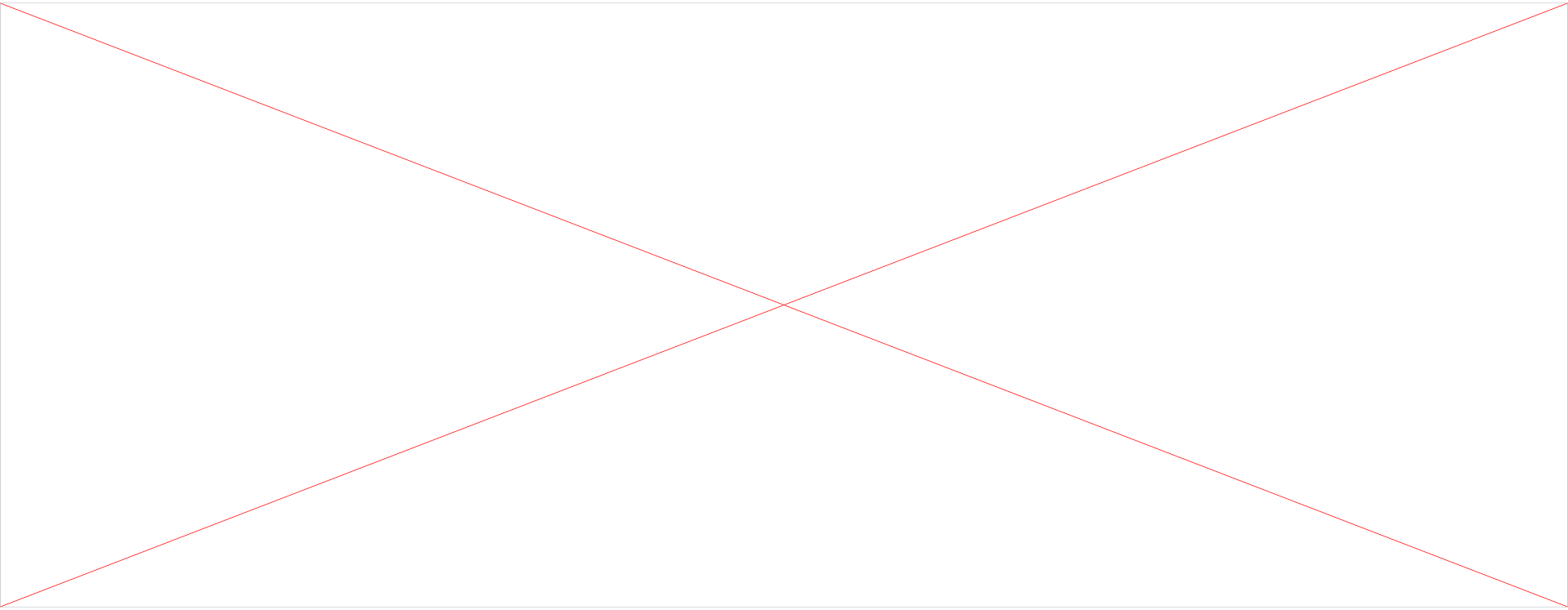
Возможны 3 случая при перестановках:

“Дядя” вершины красного цвета. Красим его и отца вершины в черный цвет, а “деда” в красный. Дед становится “иксом”

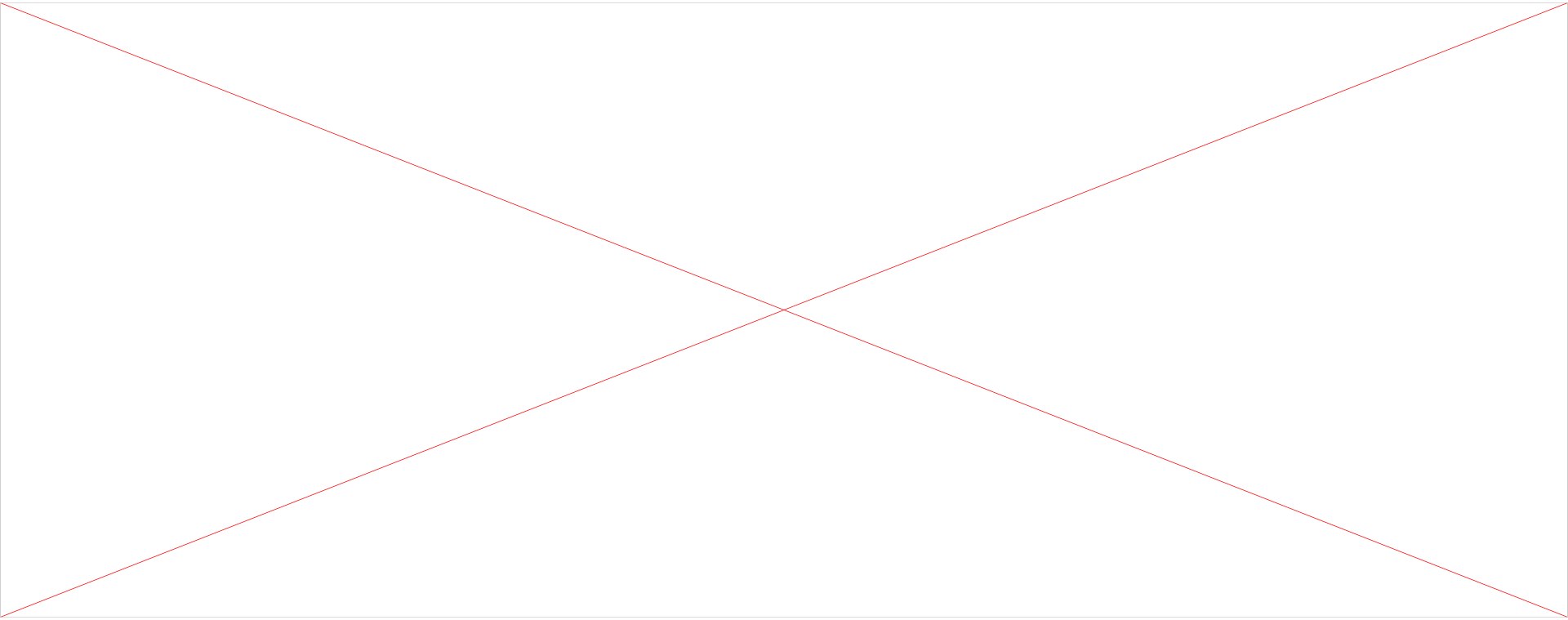
“Дядя” вершины черного цвета. Проверяем являются ли узел x и отец одинаковыми детьми (т.е. оба левые или правые). Если не являются, то делаем соответствующие вращения (левое или правое) (тем самым делая отца ребенком нового элемента). Дальнейший анализ делаем для бывшего отца.

- *Перестраиваем дерево, делая нового отца корнем данного поддерева, делая деда ребенком отца. красим отца в черный, деда в красный.*

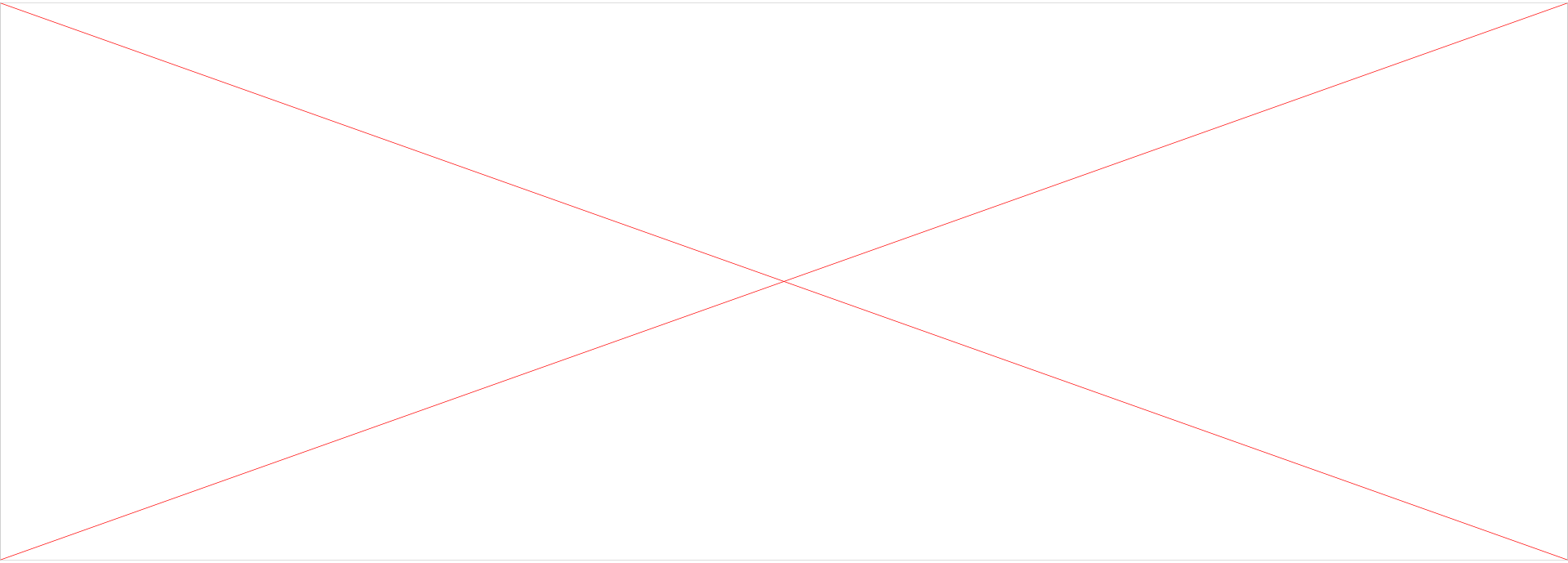
Случай 1



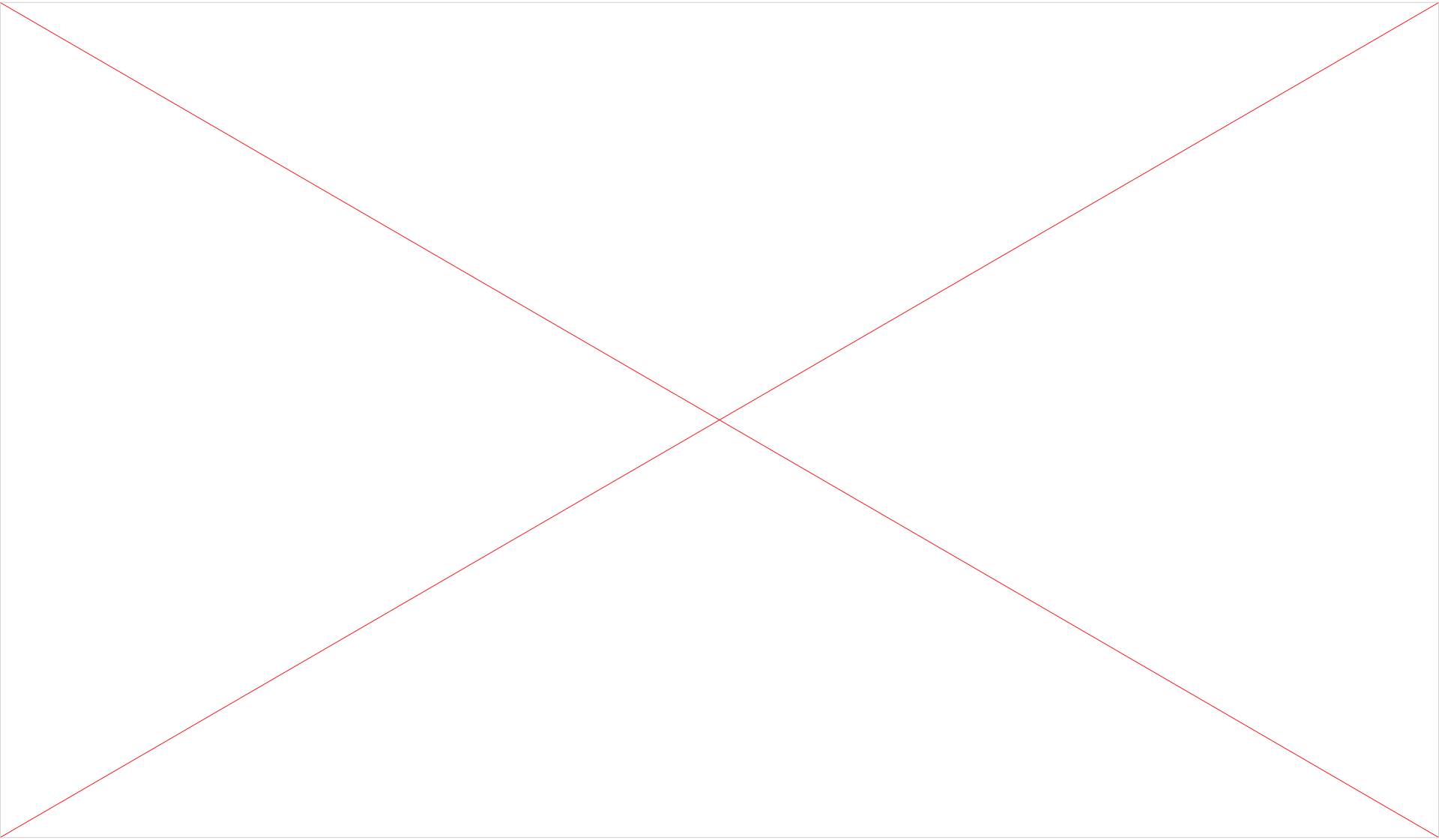
Случай 2



Случай 3

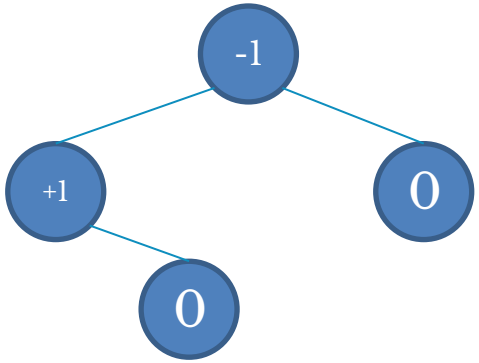


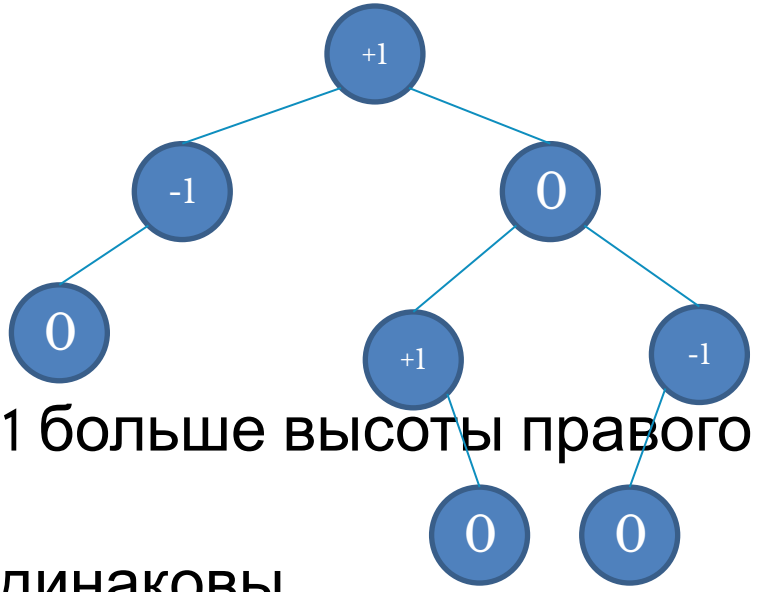
Вставка в красно-черное дерево



AVL дерево

- Свойство:
- Для любой вершины дерева высота её двух поддеревьев отличается не более чем на 1





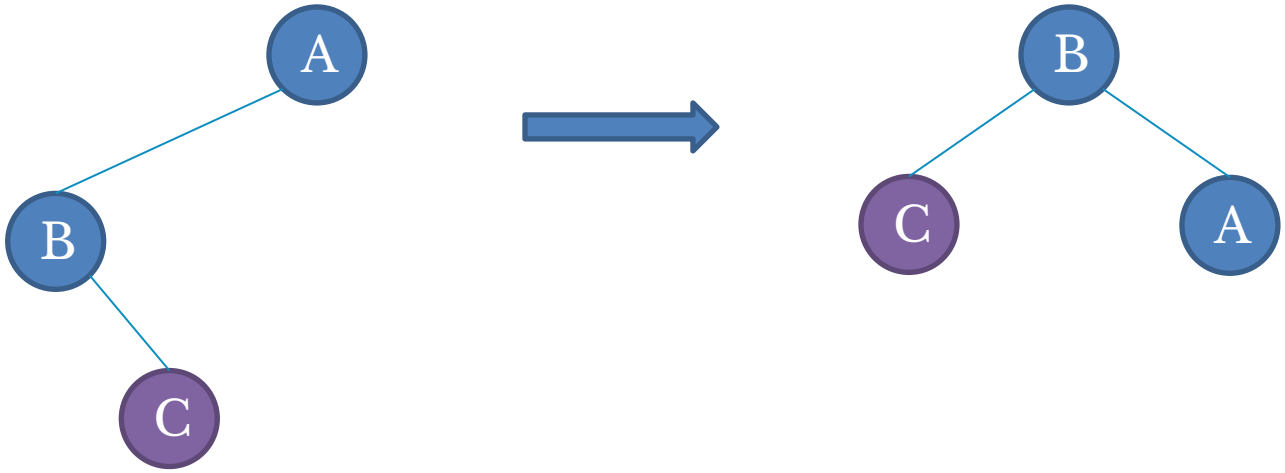
- -1: Высота левого поддерева на 1 больше высоты правого поддерева.
- 0: Высоты обоих поддеревьев одинаковы.
- +1: Высота правого поддерева на 1 больше высоты левого поддерева

AVL-дерево

- Теорема
  - *Теорема*
  - *AVL-дерево с n ключами имеет высоту $h = O(\lg n)$*
- AVL-дерево с ключами имеет высоту
- Доказательство
  - *Доказательство*
- Лемма
  - *Лемма*
- Пусть – минимальное число узлов в AVL дереве высоты , где – h-ое число Фибоначчи.
  - *Пусть m_h – минимальное число узлов в AVL дереве высоты $h \Rightarrow m_h = F_{h+2} - 1$, где F_h – h-ое число Фибоначчи.*
- Доказательство леммы (по индукции)
  - *Доказательство леммы (по индукции)*
- Пусть – минимальное число узлов в поддереве
- F = {1,1,2,3,5,8,…}
  - *Пусть m_h – минимальное число узлов в поддереве $x \Rightarrow m_{h+2} + m_{h+1} + m_h + 1$*
- Пусть для верно.
- .
  - $F = \{1,1,2,3,5,8,\ldots\} \Rightarrow m_1 = F_3 - 1 = 1$
  - *Пусть для* $m_h = F_{h+2} - 1$ *верно.* $\Rightarrow m_{h+1} = m_h + m_{h-1} + 1 = F_{h+2} - 1 + F_{h+1} - 1 + 1 = F_{h+3} - 1$ $(*)$.
  - $F_h = \phi^h, \phi = \frac{\sqrt{5}+1}{2} \Rightarrow (*)\colon n \geq \phi^{h+2} - 1 \Rightarrow n \geq \phi^h \Rightarrow \log_\phi n \geq h \Rightarrow h = O(\log_\phi n).$

AVL-дерево вставка

- При вставке элемента в дерево нарушается сбалансированность
  - *При вставке элемента в дерево нарушается балансированность* $|h_r - h_l| > 1$
- Исправляется путем левых и правых (простых и двойных поворотов)
  - *Исправляется путем левых и правых (простых и двойных поворотов)*



Сравнение AVL с RB

- RB:
  - $RB: n \geq 2^{\frac{h}{2}} - 1$
- AVL:
  - $AVL: n \geq \phi^h$
- =>при том же количестве элементов RB-дерево может быть выше AVL дерева, но не более чем в раз
  - =>*при том же количестве элементов RB-дерево может быть выше AVL дерева, но не более чем в* $\frac{\log \phi}{\log \sqrt{2}} = 1.388$ *раз.*