Красно-чёрные деревья (red-black trees). Добавление узла в красно-чёрное дерево. Удаление узла из красно-чёрного дерева. Доказательство утверждения о высоте красно-чёрного дерева.

Красно-чёрные деревья (red-black trees)

Красно-чёрное дерево - это бинарное дерево поиска, для которого выполняются **красно-черные свойства:**

1. Каждый узел дерева является либо красным *(red)*, либо черным *(black)*
2. Корень дерева является **черным узлом**
3. Каждый лист дерева (NULL) является черным узлом
4. У **красного** узла оба дочерних узла – **черные**
5. У любого узла все пути от него до листьев, являющихся его потомками, содержат *одинаковое количество черных узлов*

(В процессе добавления узла могут нарушены свойства 2 и 4)

Вычислительная сложность операций красно-черного дерева:

Для всех случаев вычислительная сложность будет **O(**$log(n)$**)**, т.к дерево самобалансируется.

Структура узла красно-черного дерева:

- *parent* – указатель на родительский узел
- *left* – указатель на левый дочерний узел
- *right* – указатель на правый дочерний узел
- *color* – цвет узла (RED, BLACK или 0, 1)
- *key* – ключ
- *value* – значение

(*Будем считать, что все листья – это указатель на один и тот же ограничивающий узел черного цвета* ***NULL****.*)

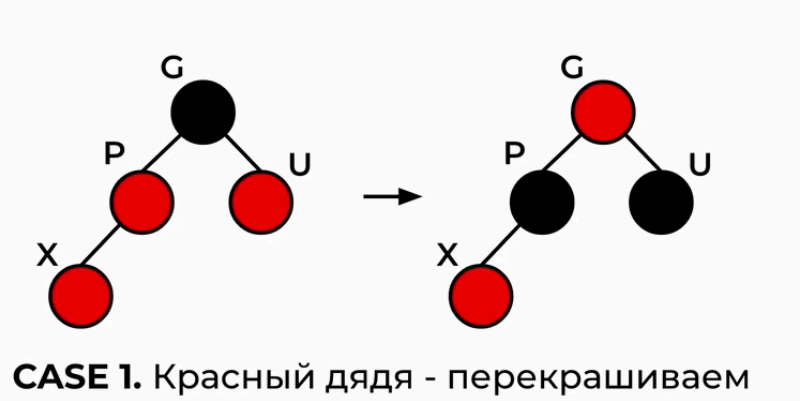
Добавление узла в красно-чёрное дерево

1. Находим лист для вставки нового элемента по такому же принципу как и в бинарном дереве поиска
2. Создаем элемент и окрашиваем его в **красный** цвет
3. Восстанавливаем свойства красно-черного дерева: перекрашиваем узлы и выполняем повороты

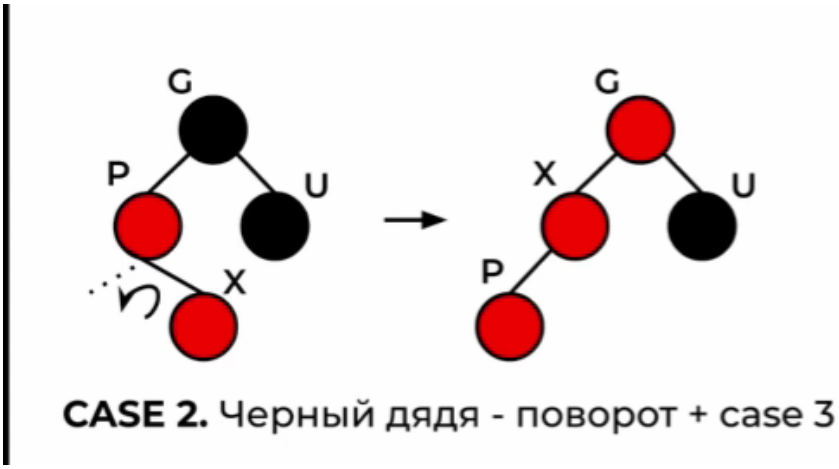
Восстановление свойств красно-черного дерева

- Возможны **6 случаев** нарушения свойства красно-черного дерева **(3 из них симметричны другим)**
- Восстановление свойств начинаем с нового элемента и продвигаемся вверх к корню дерева

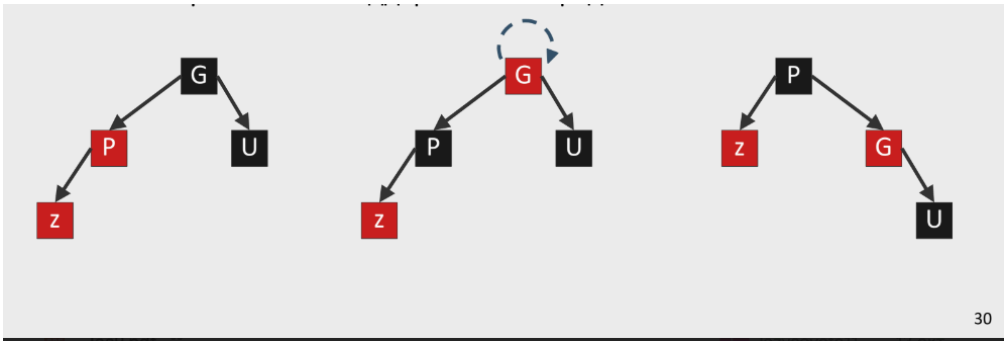
1. Случай 1:
   - Узел x красный
   - Дядя U узла x — красный
   - Узел P, корень левого поддерева своего родителя G – красный



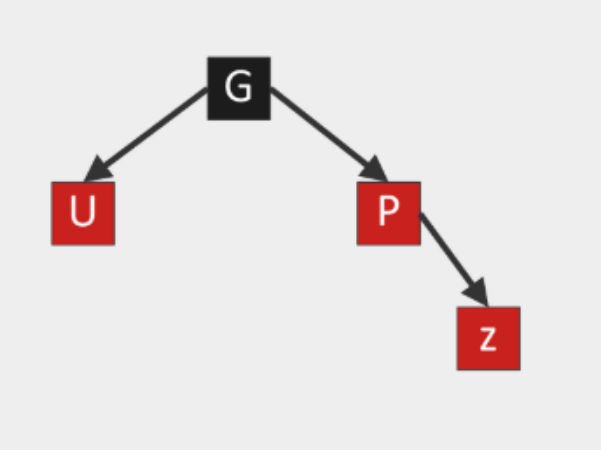
2. Случай 2:
   - Дядя U узла x – черный
   - Узел x, правый потомок P – красный
   - Родительский узел P узла x – красный
   - Узел P – корень левого поддерева своего родителя G



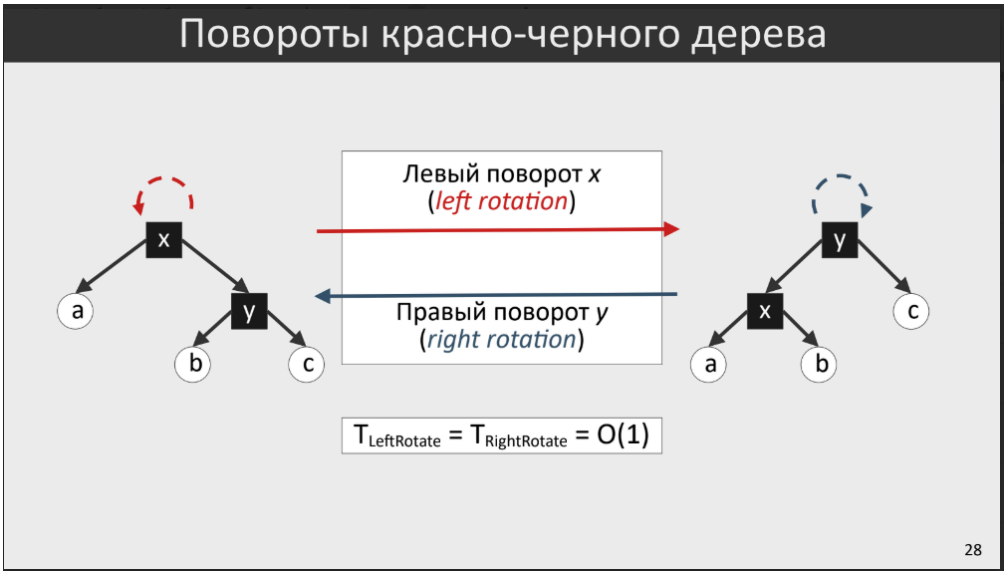
3. Случай 3:
    - Дядя U узла z – черный
    - Узел z, левый потомок P – красный
    - Родительский узел P узла z – красный
    - Узел P – корень левого поддерева своего родителя G
      Перекрашиваем вершины:
      P – черный
      G – красный
      Поворачиваем дерево G вправо



4. Случаи 4, 5 и 6 **симметричны** случаям 1, 2 и 3:
    - Узел P – это корень правого поддерева своего родителя G
    - Узел z красный
    - Родительский узел P узла z красный
    - Узел U черный или красный
    - Узел z – левый или правый дочерний элемент P



Небольшая ремарка про повороты:



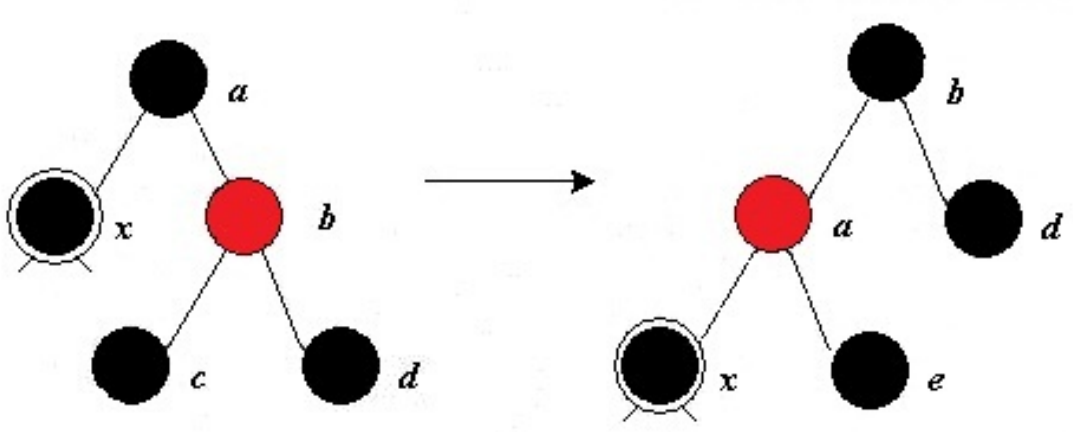
Удаление узла из красно-чёрного дерева.

При удалении вершины могут возникнуть три случая в зависимости от количества её детей:

1. Если у вершины **нет** детей, то изменяем указатель на неё у родителя на nil.
2. Если у неё **только один** ребёнок, то делаем у родителя ссылку на него вместо этой вершины.
3. Если же имеются **оба ребёнка**, то находим вершину со следующим значением ключа. У такой вершины нет левого ребёнка. Удаляем уже эту вершину описанным во втором пункте способом, скопировав её ключ в изначальную вершину.
   После удаления необходимо восстановить свойства красно-черного дерева:

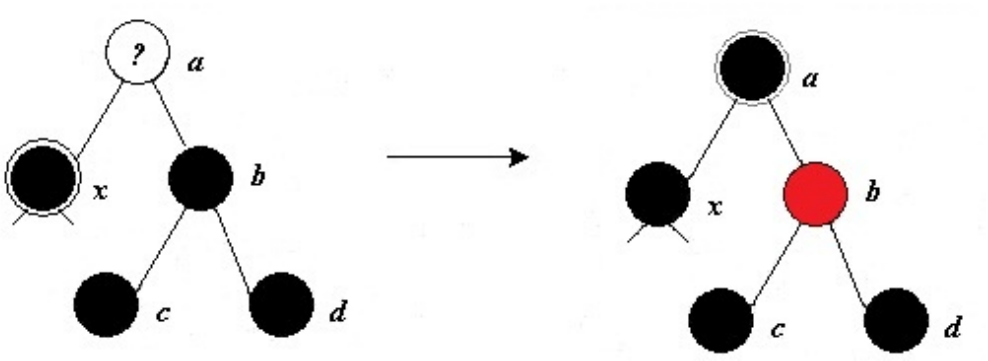
Рассмотрим ребёнка удалённой вершины:

1. Если **брат** этого ребёнка **красный**, то делаем вращение вокруг ребра между отцом и братом, тогда брат становится родителем отца. Красим его в чёрный, а отца - в красный цвет

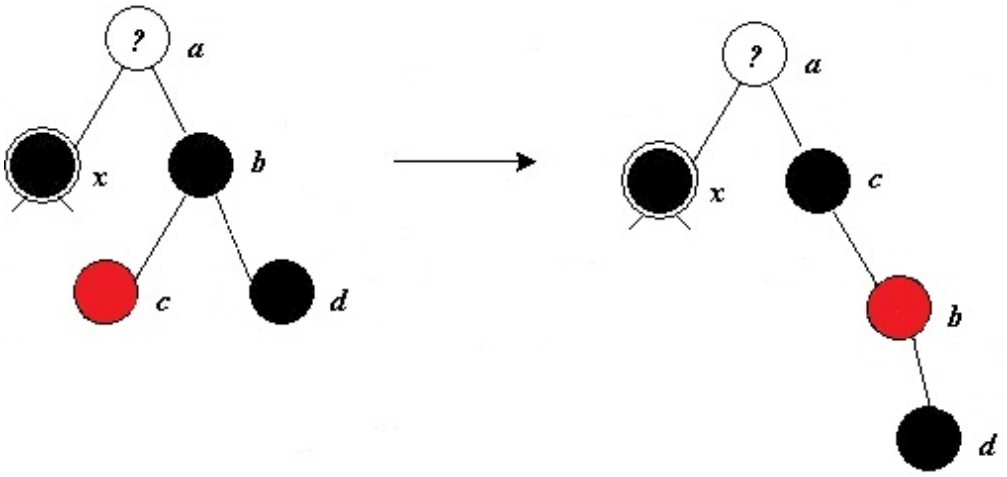


2. Если **брат** текущей вершины был **чёрным**, то получаем три случая:

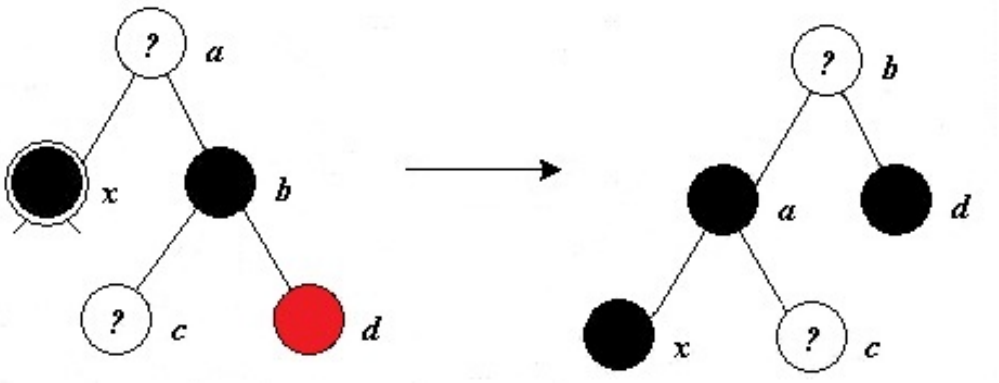
1. **Оба ребёнка у брата чёрные**. Красим брата в красный цвет и рассматриваем далее отца вершины.



2. Если у брата **правый ребёнок чёрный**, **а левый красный**, то перекрашиваем брата и его левого сына и делаем вращение.



3. У брата **правый ребёнок красный**, то перекрашиваем брата в цвет отца, его ребёнка и отца - в чёрный, делаем вращение и выходим из алгоритма.



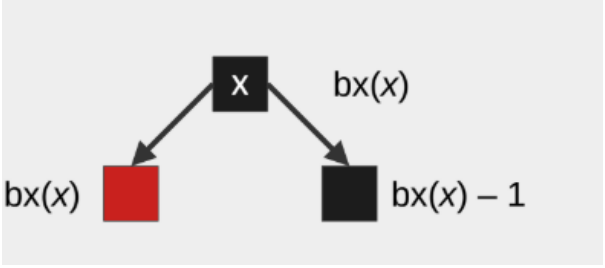
(*Продолжаем тот же алгоритм, пока текущая вершина чёрная и мы не дошли до корня дерева. При удалении выполняется не более трёх вращений.*)

Доказательство утверждения о высоте красно-чёрного дерева

- **Черная высота** bh **(x)** узла *(black height)* – это количество черных узлов на пути от узла x *(не считая его)* до листа
- Черная высота дерева – это черная высота его **корня**

Лемма:
Красно-черное дерево с **n** внутренними узлами имеет высоту, не превышающую $2log_2(n+1)$

Доказательство

- Покажем по индукции, что любое поддерево с вершиной в узле x содержит не менее $2^{bh(x)}-1$ внутренних узлов
- **Базис индукции**
  Если высота h узла x равна 0, то узел x – это лист (*NULL*), а его поддерево содержит не менее $2^{bh(x)}–1 = 2^0–1 = 0$ внутренних узлов
- **Шаг индукции**
  Рассмотрим узел x, который имеет положительную высоту **h(x)** – внутренний узел с **двумя** потомками
  Каждый дочерний узел имеет черную высоту либо **bh(x)**, либо **bh(x) – 1**, в зависимости от его цвета
  

- Поскольку высота потомка x меньше высоты узла x, мы можем использовать предположение индукции и сделать вывод о том, что каждый потомок x имеет как минимум $2^{bh(x)-1}–1$ внутренних узлов
- Тогда все дерево с корнем в узле x содержит не менее $(2^{bh(x)-1}–1) + (2^{bh(x)-1}–1) + 1 = 2^{bh(x)}–1$ внутренних узлов.
- Получили, что в дереве x число n внутренних узлов $n \geq 2^{bh(x)}–1$
- По свойству 4 как минимум половина узлов на пути от корня к листу чёрные, тогда $bh(x) \geq h(x)/2$
- Следовательно:
  $n \geq 2^{h(x)/2}–1$
  $log_2(n+1) \geq h(x)/2$
  $h(x) \leq 2log_2(n+1)$