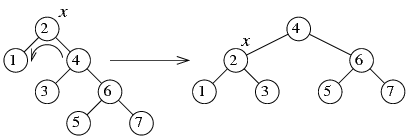
Смена вершины в Красно-Черных деревьях

Правое вращение вершины *x* реализуется аналогично.

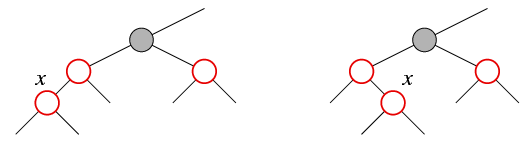
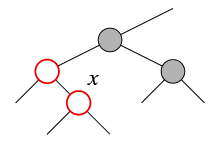
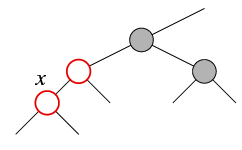
Операция левого или правого вращения вершины во многих случаях позволяет "исправить" балансировку дерева, как показывает следующий пример. В нем мы применяем вращение вершины *x* влево.



**Восстановление структуры красно-черного дерева после добавления вершины (ребалансировка)**

При добавлении вершины в дереве поиска новая вершина добавляется как терминальная после выполнения операции поиска. В красно-черном дереве новая вершина окрашивается изначально в красный цвет. Если ее родительская вершина черная, то все свойства красно-черного дерева выполняются и алгоритм добавления на этом заканчивается. Если же родительская вершина красная, то нарушается второе свойство в определении красно-черного дерева: у красной вершины дети должны быть черными. В этом случае выполняется процедура ребалансировки (восстановления структуры красно-черного дерева). Процедура ребалансировки носит итеративный характер. В ней рассматривается текущий узел *x* красного цвета, родительский узел (отец) которого тоже красный. В процессе ребалансировки метка *x* может перемещаться вверх по дереву, так что время восстановления не превосходит фиксированной константы, умноженной на высоту дерева, т.е. равно *O*(log *n*).

В процедуре ребалансировки рассматриваются 6 различных случаев. В случаях 1-3 отец узла *x* является **левым сыном** своего отца, т.е. деда *x*.

1. У узла *x* есть красный "дядя" (брат отца *x*). При этом *x* может быть левым или правым сыном своего отца — случаи эти рассматриваются аналогично.  
   
2. У узла *x* "дядя" (брат отца) либо черный, либо его вообще нет (т.е. сыном является лист или внешний узел, который мы также считаем черным), и узел *x* является **правым сыном** своего отца.  
   
3. У узла *x* "дядя" (брат отца) либо черный, либо его вообще нет, и узел *x* является **левым сыном** своего отца.  
   

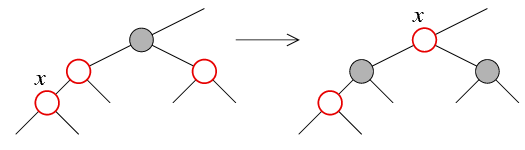
Случаи 4-6 зеркально симметричны случаям 1-3, в них отец узла *x* является **правым сыном** своего отца, т.е. деда *x*. Рассматриваются они аналогично случаям 1-3 заменой слова "левый" на "правый" и обратно, так что мы их описывать не будем.

Укажем, какие действия выполняются в каждом из случаев 1-3 для восстановления балансировки дерева.

**Случай 1 (красный дядя)**

Этот случай наиболее прост:

* перекрашиваем отца и дядю узла *x* в черный цвет;
* перекрашиваем деда узла *x* в красный цвет;
* перемещаем метку *x* вверх по дереву на деда узла *x*: и переходим в цикле к восстановлению нового узла *x* (деда предыдущего *x*)  
       *x* = *x*->parent->parent

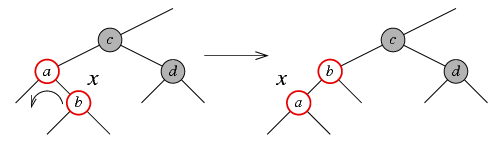


Цикл завершается, либо когда мы достигли корня дерева (если корень был перекрашен в красный цвет, то надо перекрасить его обратно в черный), либо когда отец узла *x* черный.

**Случай 2 (дядя черный или его вообще нет, узел *x* является правым сыном своего отца)**

Этот случай сводится к случаю 3 путем следующих преобразований:

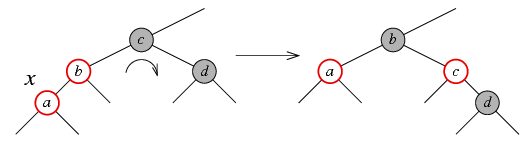
* перемещаем метку *x* на вверх по дереву: *x* = *x*->parent;
* левое вращение *x*.



**Случай 3 (дядя черный или его вообще нет, узел *x* является левым сыном своего отца)**

В этом случае выполняются следующие действия:

* перекрашиваем отца узла *x* в черный цвет;
* перекрашиваем деда (т.е. отца отца) *x* в красный;
* выполняем правое вращение деда *x*.

На этом алгоритм восстановления балансировки заканчивается (в случае 3 цикл завершается).  


Отметим, что всего в процедуре восстановления структуры красно-черного дерева (ребалансировки) может выполняться не более *O*(*h*) операций; поскольку для красно-черного дерева *h*  ≤  2 log2(*n*+1), получаем, что время выполнения ребалансировки равно *O*(log2 *n*). В этой процедуре выполняются перекрашивание узлов дерева и операции вращения, причем общее число вращений — не больше двух (одно вращение в случае 3 и два в случае 2).