Мини-задача #26 (1 балл)

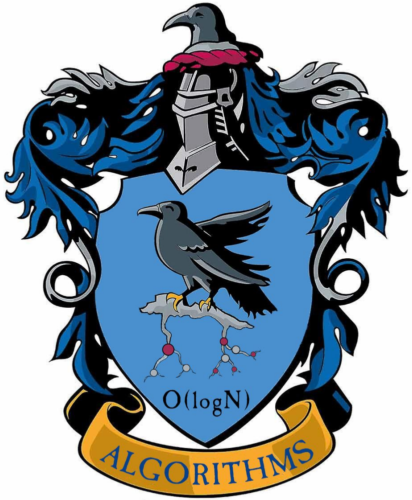
Реализовать процедуру которая принимает BST, а возвращает сбалансированное BST из тех же значений.

Сбалансированным будем называть BST, в котором высота поддеревьев для каждой вершины отличается не более, чем на 1.

https://leetcode.com/problems/balance-a-binary-search-tree

Алгоритмы и структуры данных

АВЛ-дерево, красно-чёрное дерево



Бинарные деревья поиска

Операции:

```
1.  find(value)      -> O(height)
2.  select(i)        -> O(height)
3.  min/max          -> O(height)
4.  pred/succ(ptr)   -> O(height)
5.  rank(value)      -> O(height)
6.  вывод в пор.     -> O(N)
    возрастания
--------------------------------

7.  insert(value)    -> O(height)
8.  remove(value)    -> O(height)
```

Назревает проблема!

Обещали логарифм, а дают какой-то O(height)

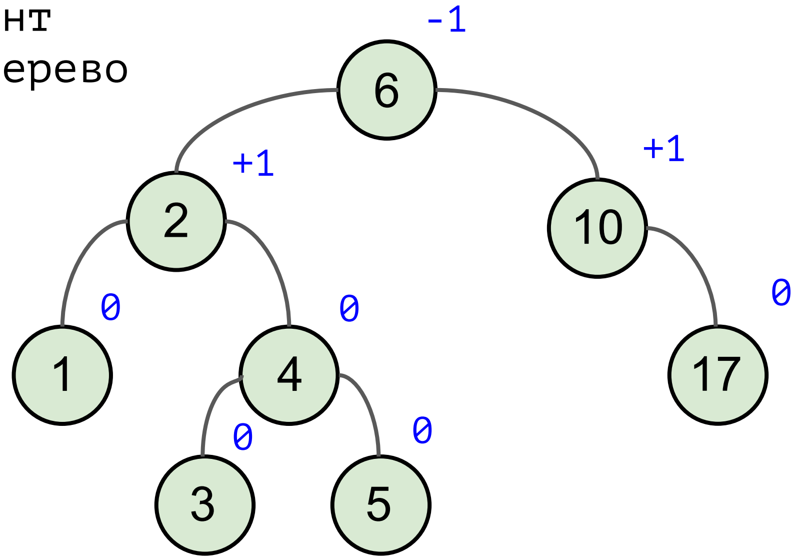
Значит нам нужны такие BST, чтобы высота у дерева всегда была logN

АВЛ-деревья

Авторы: Адельсон-Вельский Г.М. и Ландис Е. М., 1962 год

Идея: возьмем обычное BST и добавим ограничение, что для любой вершины высоты левого и правого поддеревьев отличаются не более, чем на 1.

Далее поддерживаем этот инвариант во всех операциях, изменяющих дерево (т.е. insert и remove)



АВЛ-деревья

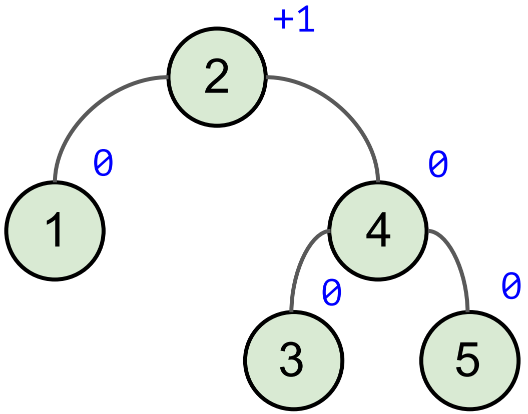
Операции:

```
1.  find(value)     -> O(logN)
2.  select(i)       -> O(logN)
3.  min/max         -> O(logN)
4.  pred/succ(ptr) -> O(logN)
5.  rank(value)     -> O(logN)
6.  вывод в пор.    -> O(N)
    возрастания
---------------------------------
7.  insert(value)  -> O(???)
8.  remove(value)  -> O(???)
```

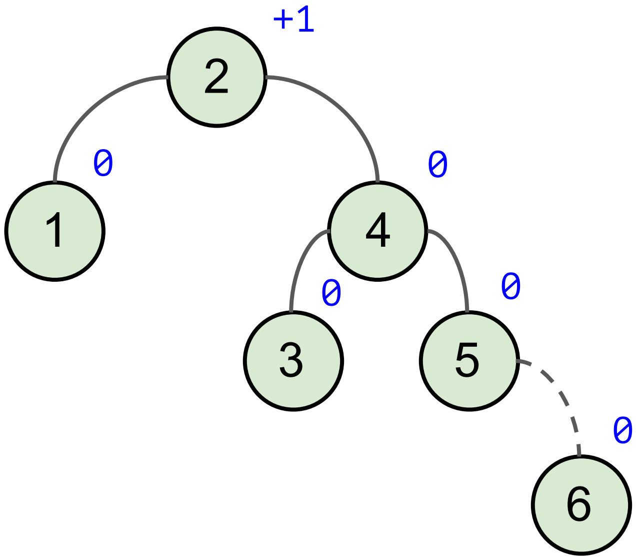
В АВЛ-дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

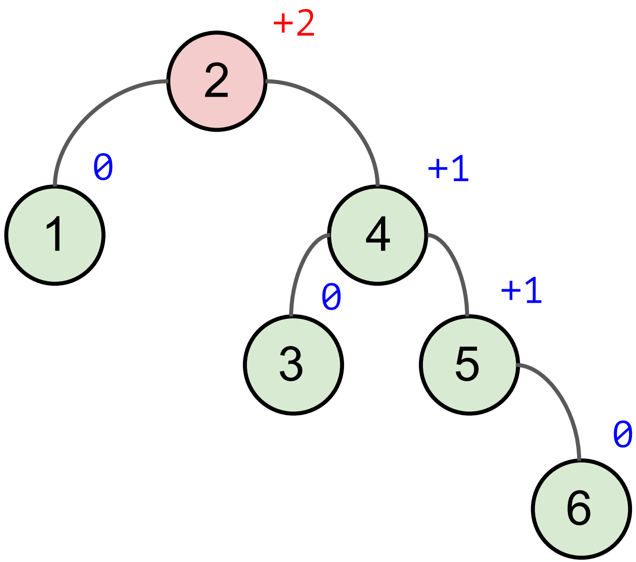
АВЛ-деревья: вращения



АВЛ-деревья: вращения

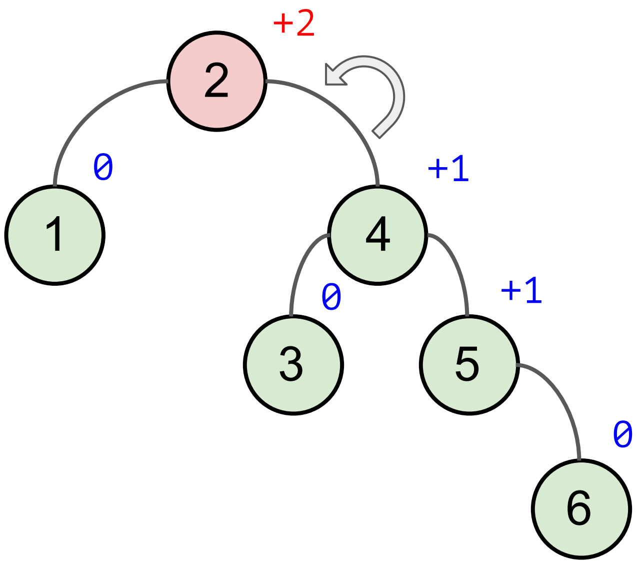


АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

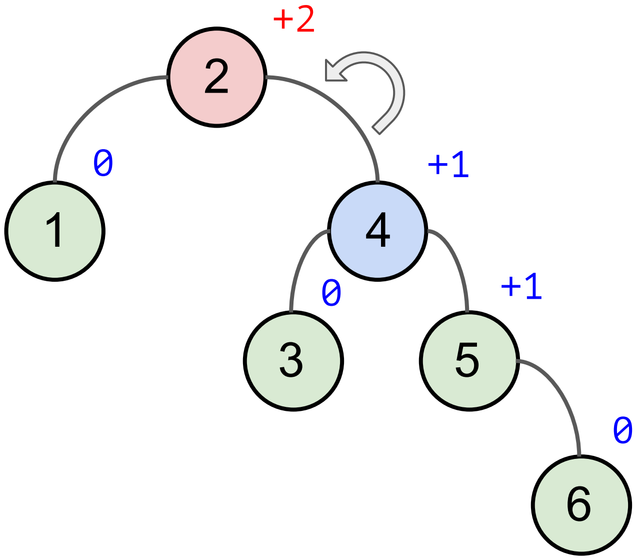
АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

Тогда выполним **малое** левое вращение.

АВЛ-деревья: вращения

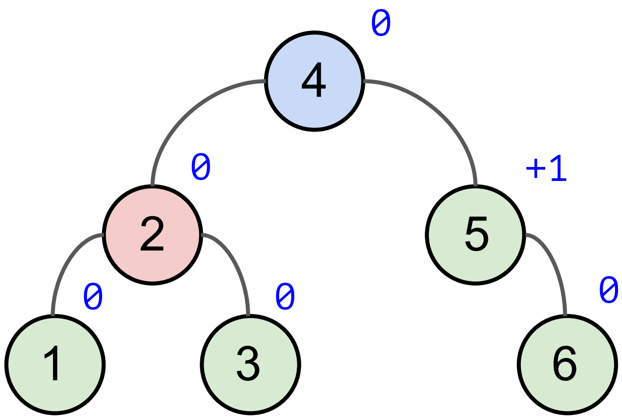


Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

АВЛ-деревья: вращения

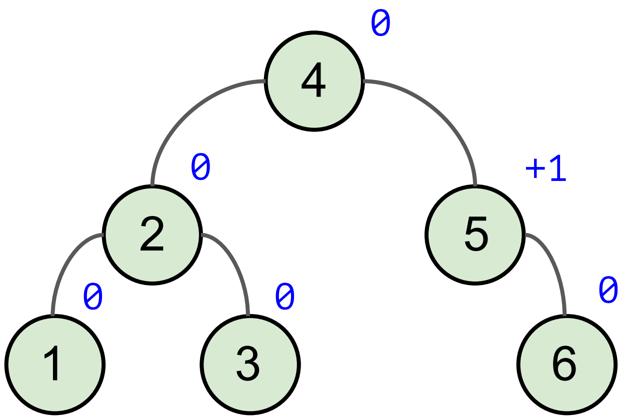


Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

АВЛ-деревья: вращения



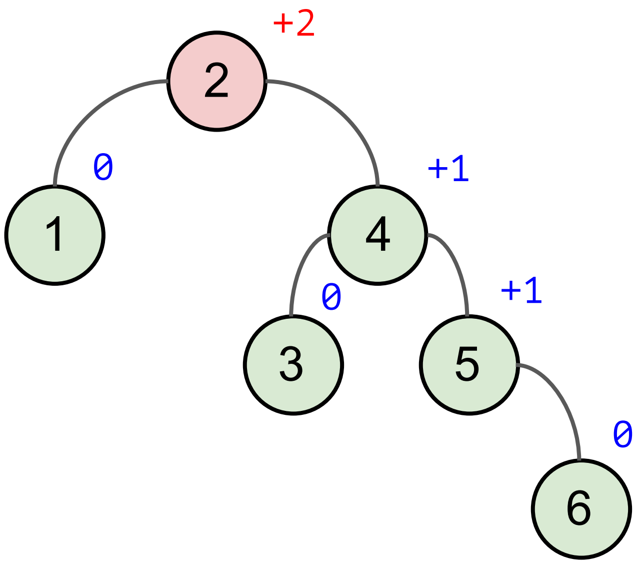
Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

После этого баланс в поддереве восстанавливается, а сама операция занимает O(1)

АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

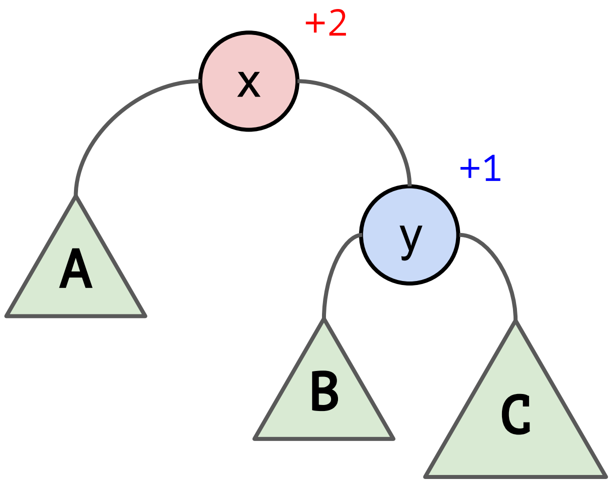
Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

После этого баланс в поддереве восстанавливается, а сама операция занимает O(1)

Обобщим без привязки к конкретному дереву.

АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

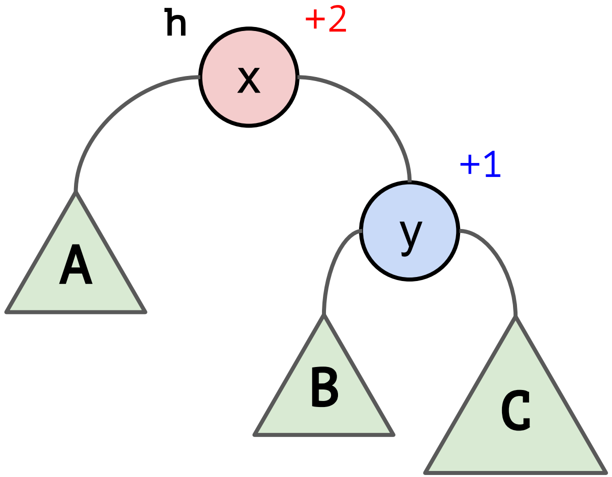
Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

После этого баланс в поддереве восстанавливается, а сама операция занимает O(1)

Обобщим без привязки к конкретному дереву.

АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

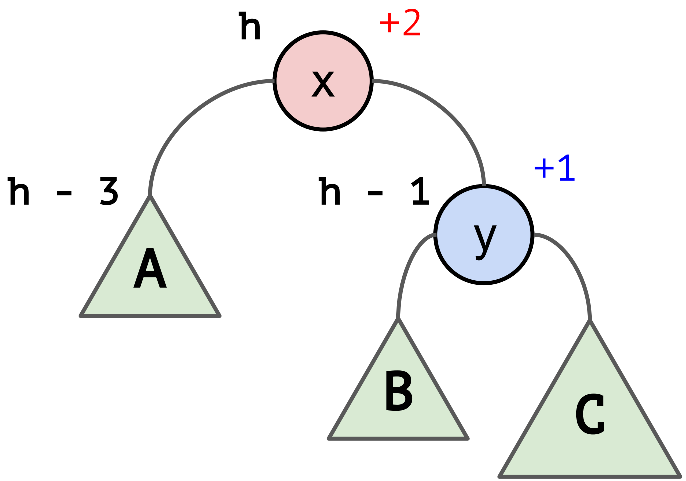
Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

После этого баланс в поддереве восстанавливается, а сама операция занимает O(1)

Обобщим без привязки к конкретному дереву. Запишем высоты.

АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

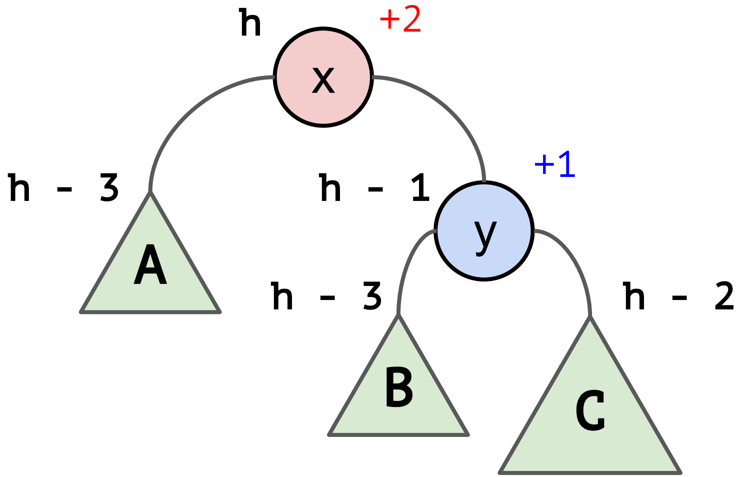
Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

После этого баланс в поддереве восстанавливается, а сама операция занимает O(1)

Обобщим без привязки к конкретному дереву. Запишем высоты.

АВЛ-деревья: вращения



Баланс нарушился, основное свойство АВЛ дерево - нарушено, т.к. получили разность высот поддеревьев >1

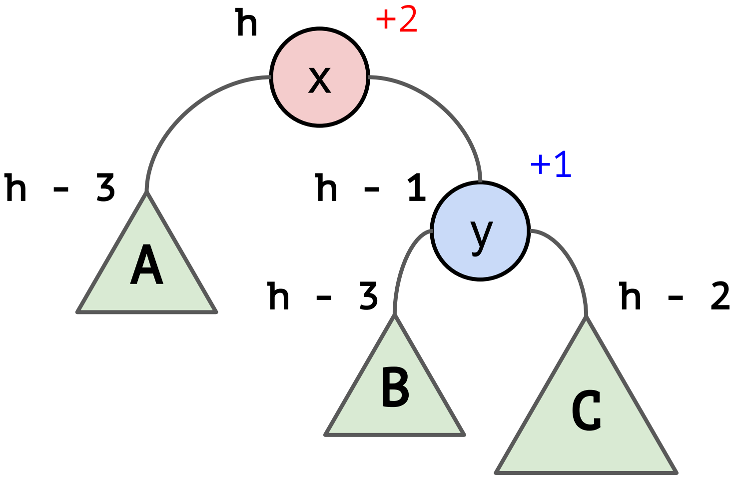
Тогда выполним **малое** левое вращение.

Корнем объявим правого сына проблемной вершины, детей переподвесим

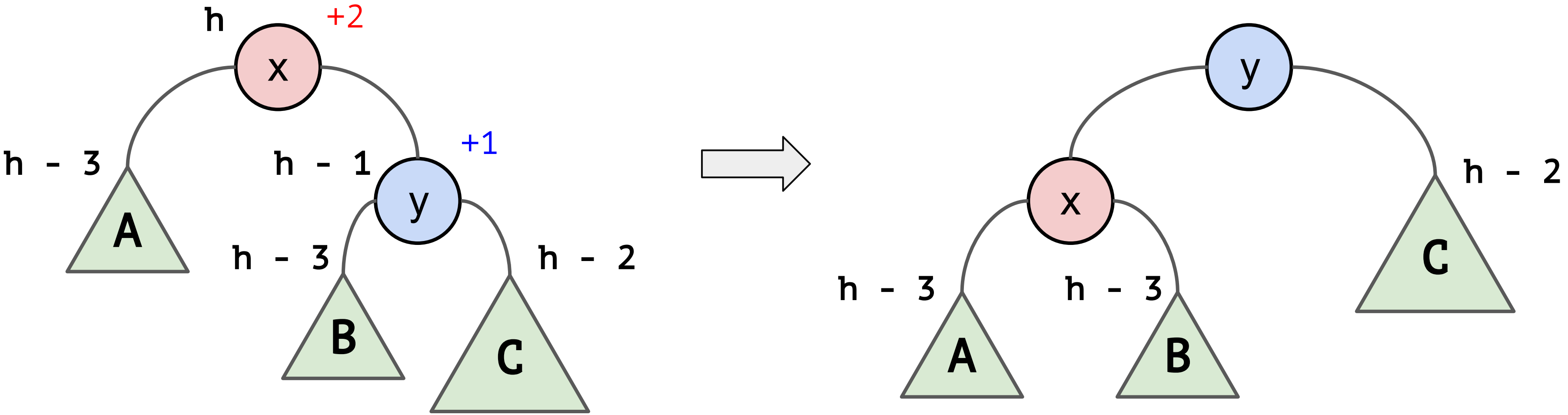
После этого баланс в поддереве восстанавливается, а сама операция занимает O(1)

Обобщим без привязки к конкретному дереву. Запишем высоты.

АВЛ-деревья: вращения

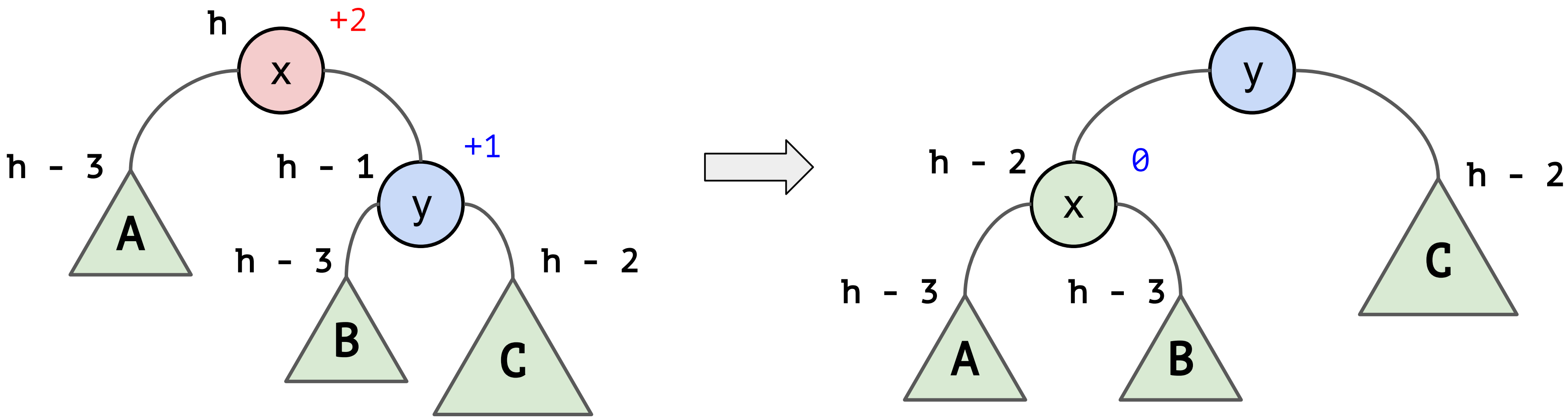


АВЛ-деревья: вращения



Высоты поддеревьев A, B и C не меняются.
Более того, они сбалансированы (мы же их не меняли).

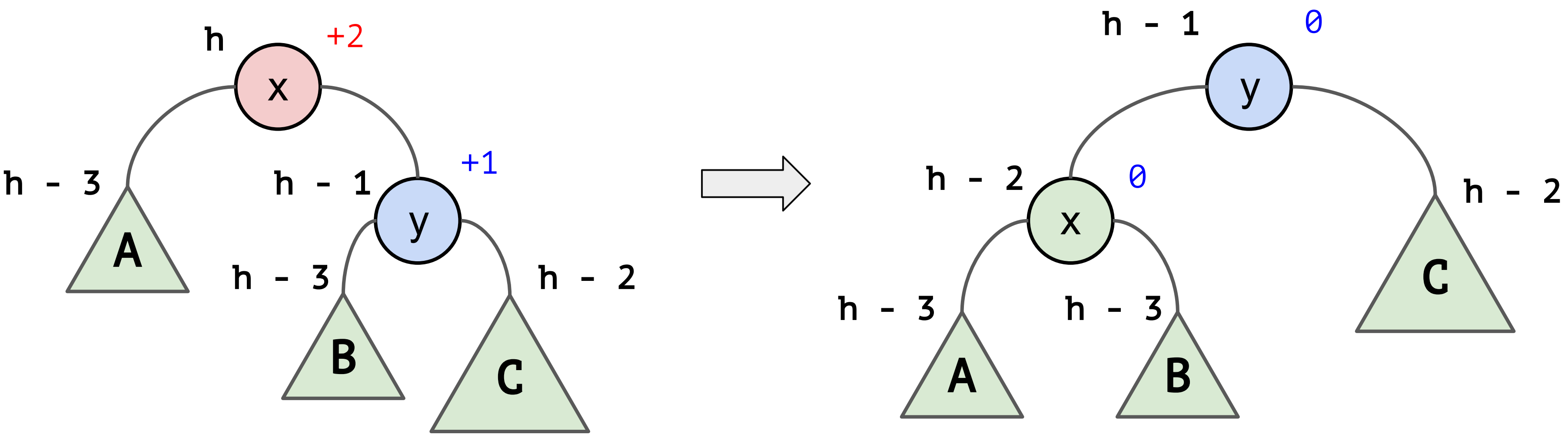
АВЛ-деревья: вращения



Высоты поддеревьев A, B и C не меняются.
Более того, они сбалансированы (мы же их не меняли).

Запишем высоты и балансы для x и y

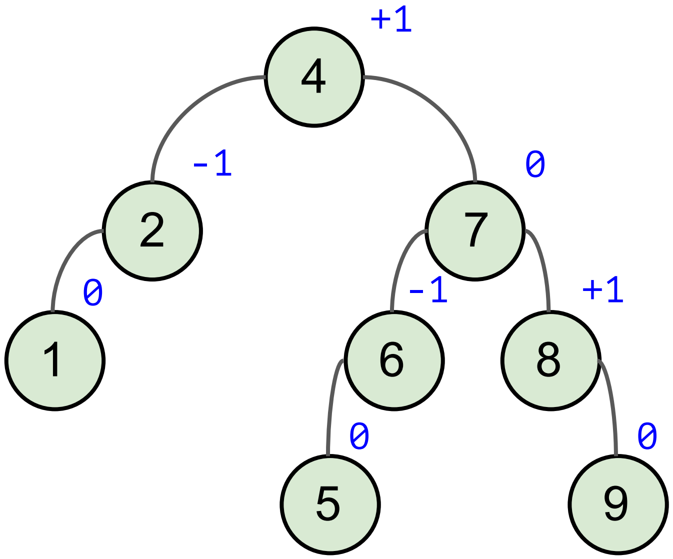
АВЛ-деревья: вращения



Высоты поддеревьев A, B и C не меняются.
Более того, они сбалансированы (мы же их не меняли).

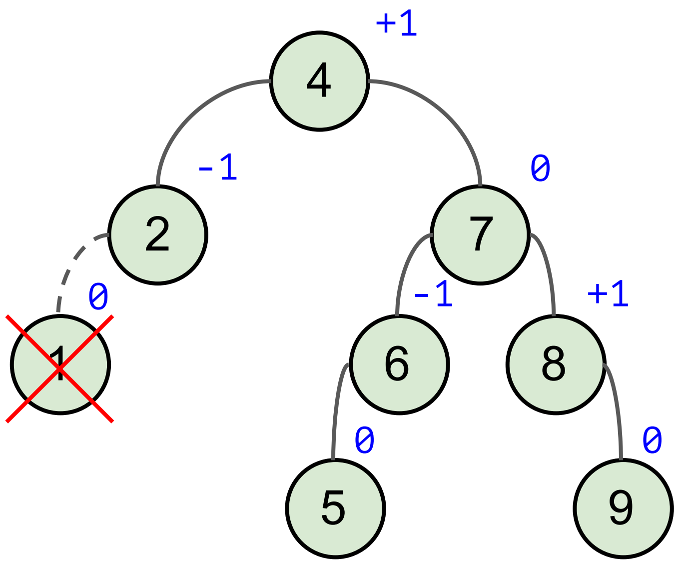
Запишем высоты и балансы для x и y

АВЛ-деревья: вращения



Похожая ситуация: в корне дисбаланс, а у правого сына баланс 0

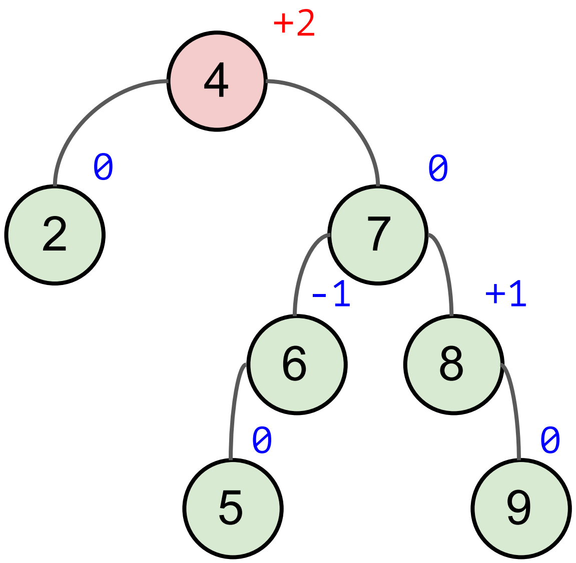
АВЛ-деревья: вращения



Похожая ситуация: в корне дисбаланс, а у правого сына баланс 0

(например, такое могло бы случится, если бы мы удалили самую левую вершину)

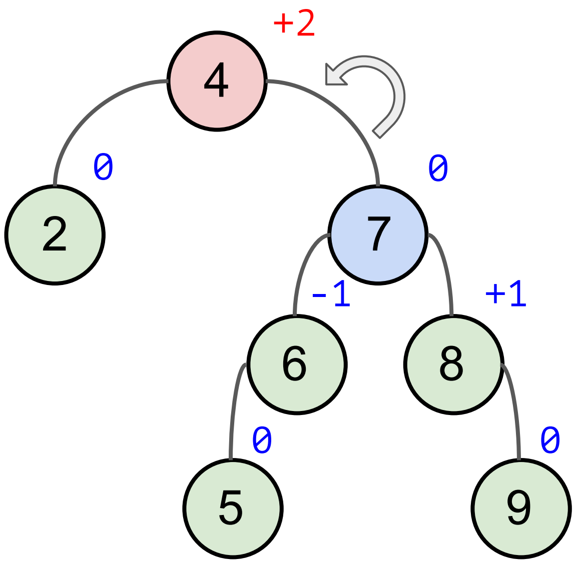
АВЛ-деревья: вращения



Похожая ситуация: в корне дисбаланс, а у правого сына баланс 0

(например, такое могло бы случится, если бы мы удалили самую левую вершину)

АВЛ-деревья: вращения

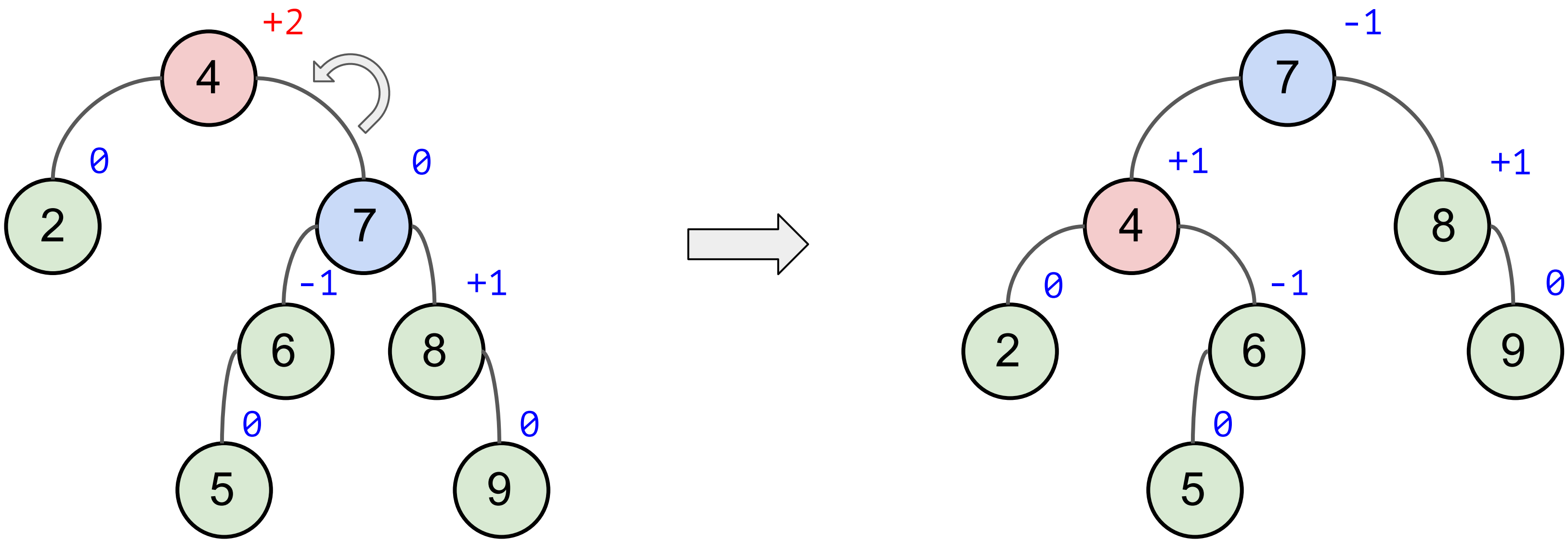


Похожая ситуация: в корне дисбаланс, а у правого сына баланс 0

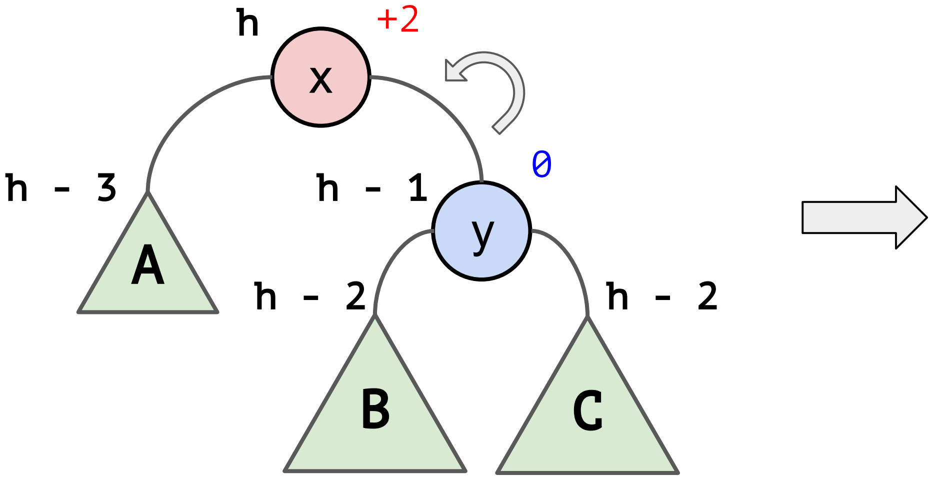
(например, такое могло бы случится, если бы мы удалили самую левую вершину)

Опять сработает **малое** левое вращение

АВЛ-деревья: вращения

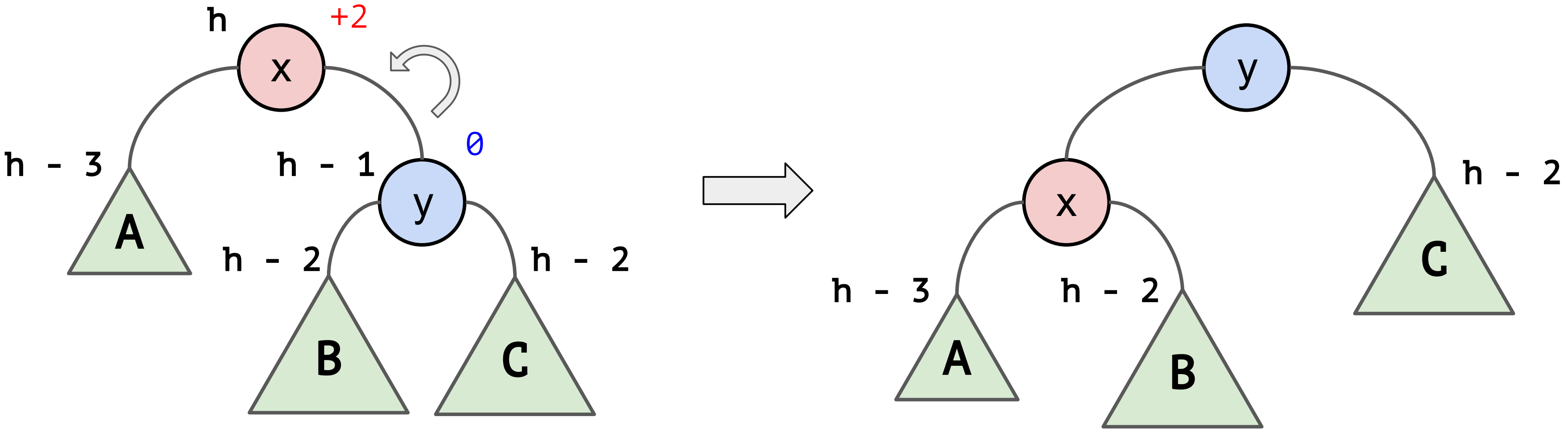


АВЛ-деревья: вращения



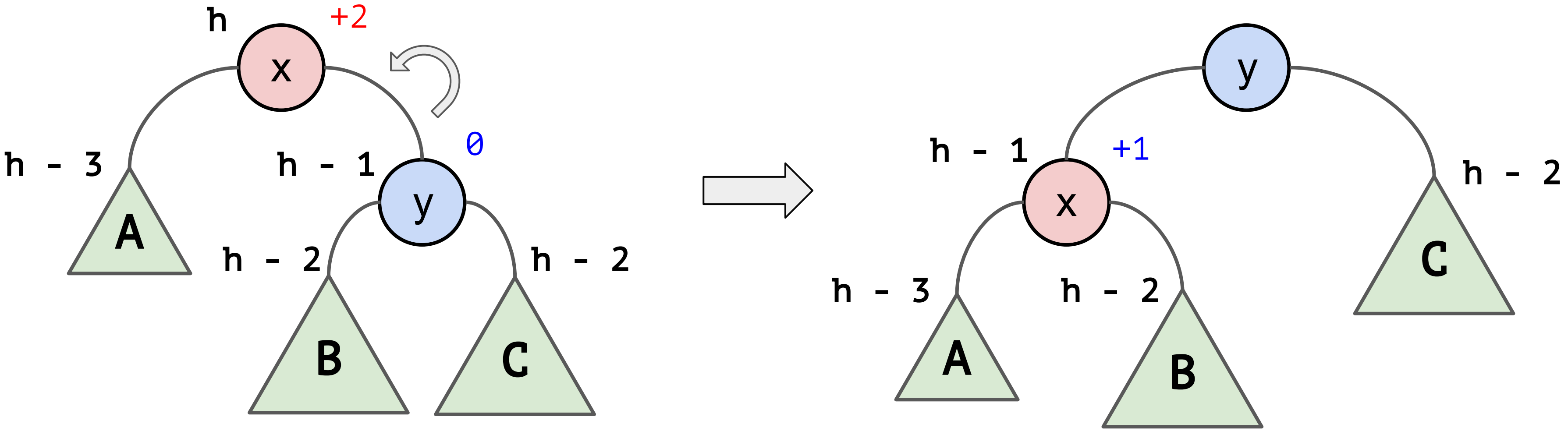
Снова обобщим до произвольных сбалансированных поддеревьев.

АВЛ-деревья: вращения



Снова обобщим до произвольных сбалансированных поддеревьев.

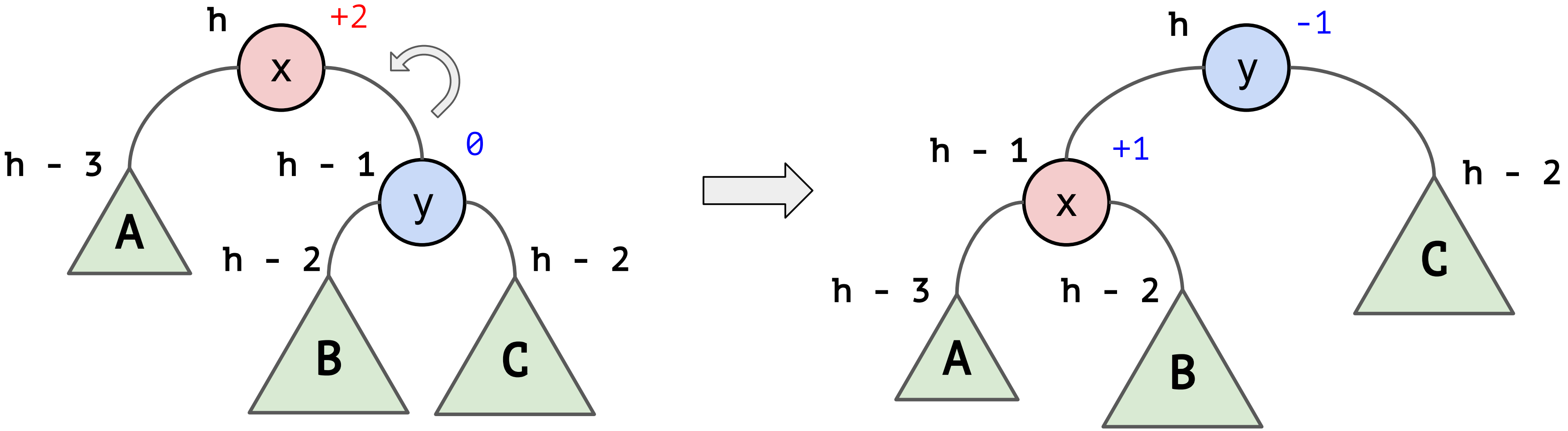
АВЛ-деревья: вращения



Снова обобщим до произвольных сбалансированных поддеревьев.

Снова выпишем высоты и балансы для x и y.

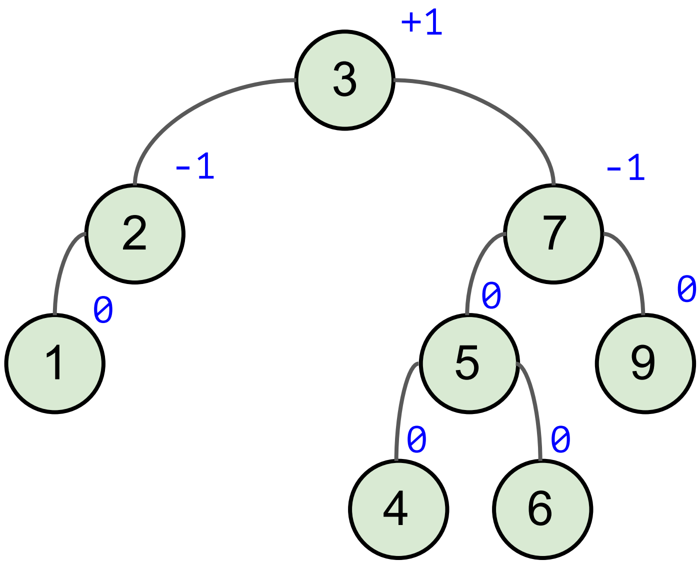
АВЛ-деревья: вращения



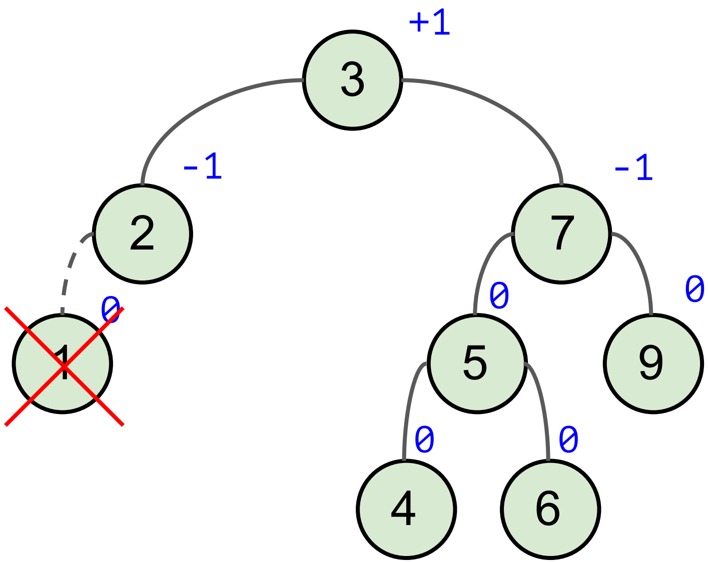
Снова обобщим до произвольных сбалансированных поддеревьев.

Снова выпишем высоты и балансы для x и y. Снова сбалансированно.

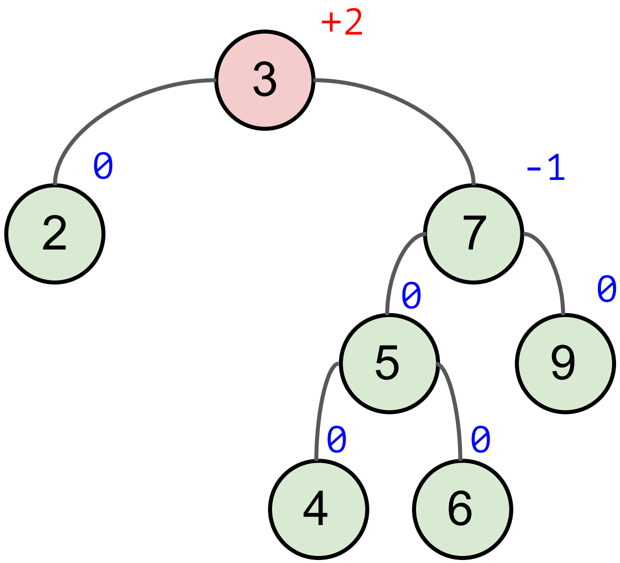
АВЛ-деревья: вращения



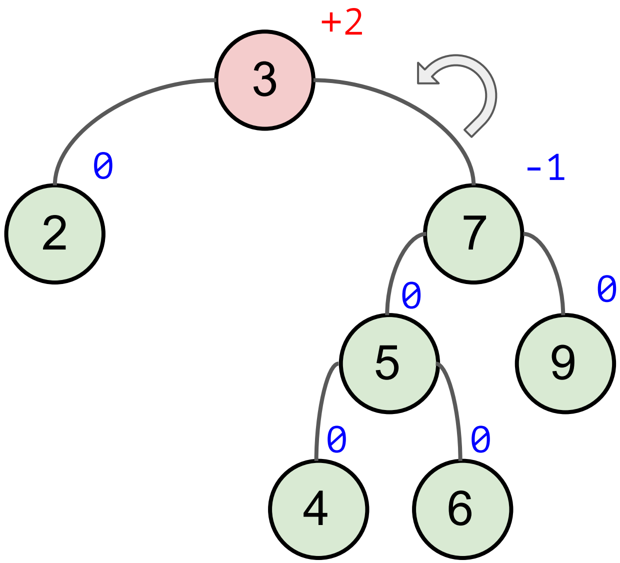
АВЛ-деревья: вращения



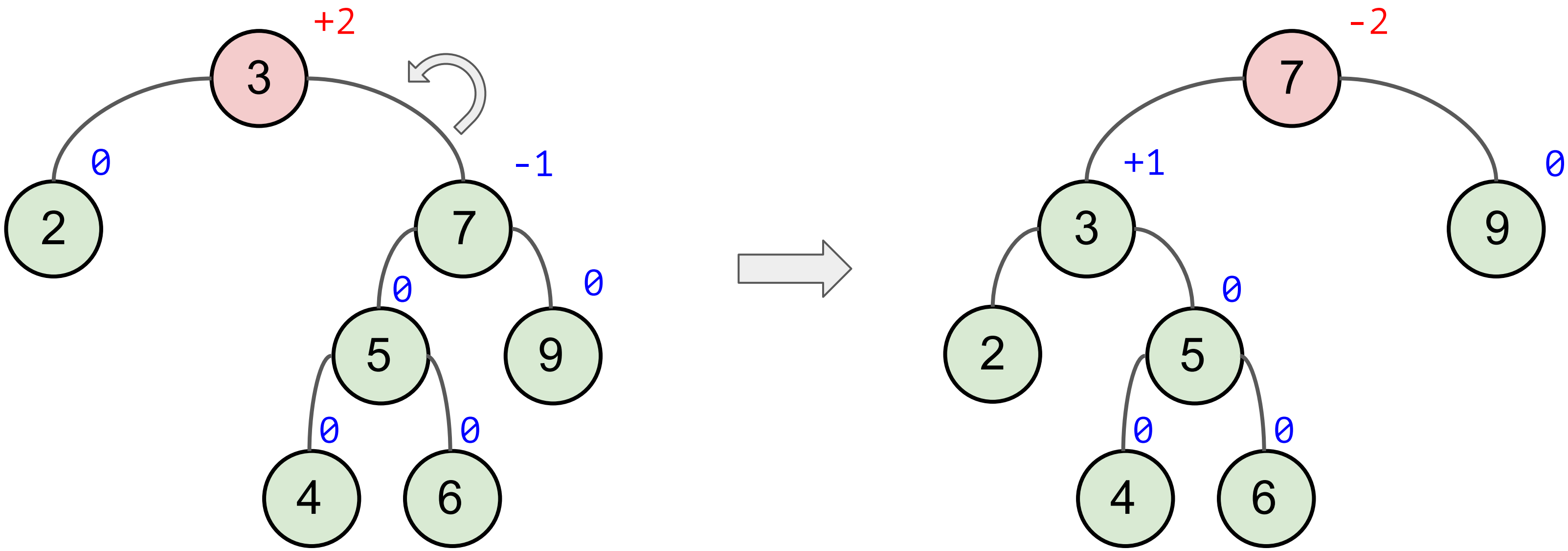
АВЛ-деревья: вращения



АВЛ-деревья: вращения

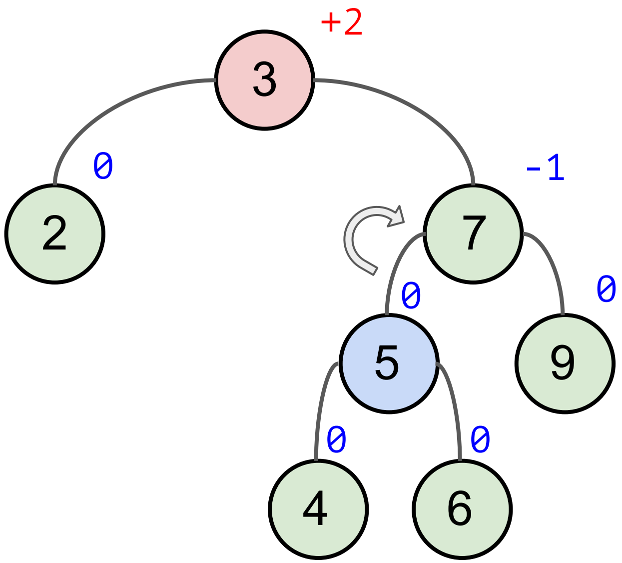


АВЛ-деревья: вращения



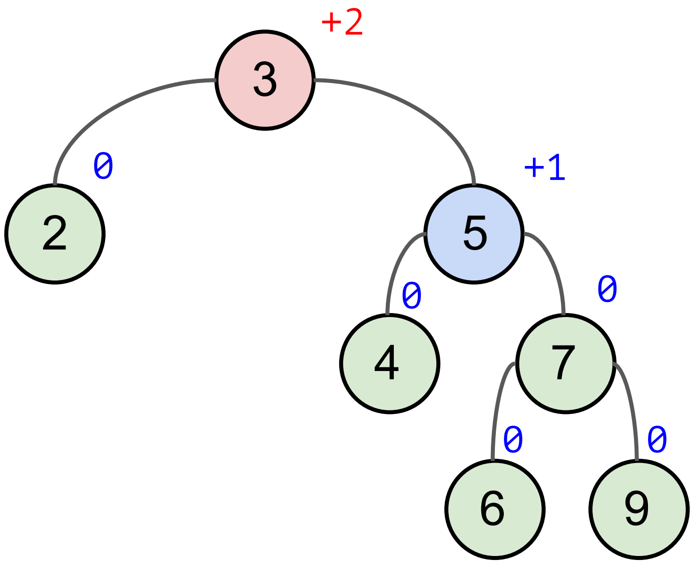
Малое левое вращение не помогает, дерево все еще не сбалансировано!

АВЛ-деревья: вращения



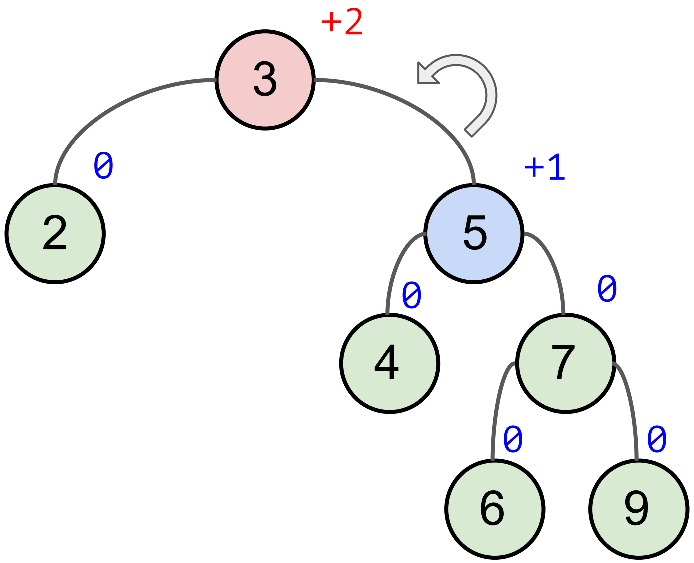
Работаем в два шага: 1) малое правое вращение вокруг 5-7

АВЛ-деревья: вращения



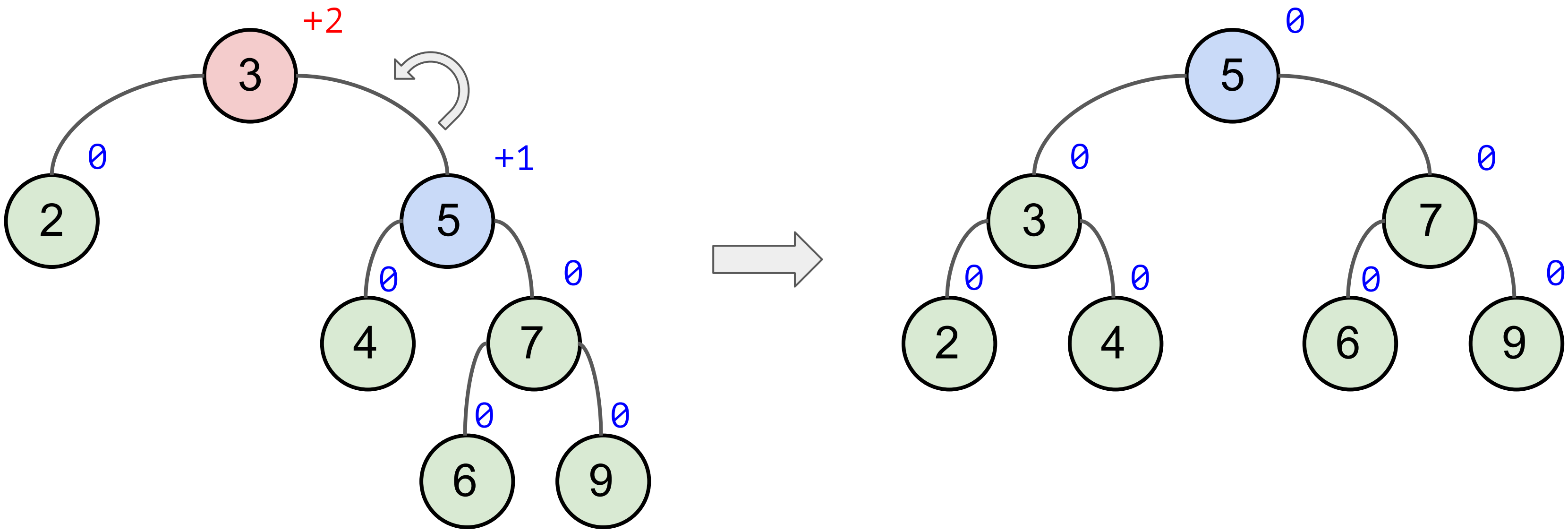
Работаем в два шага: 1) малое правое вращение вокруг 5-7

АВЛ-деревья: вращения



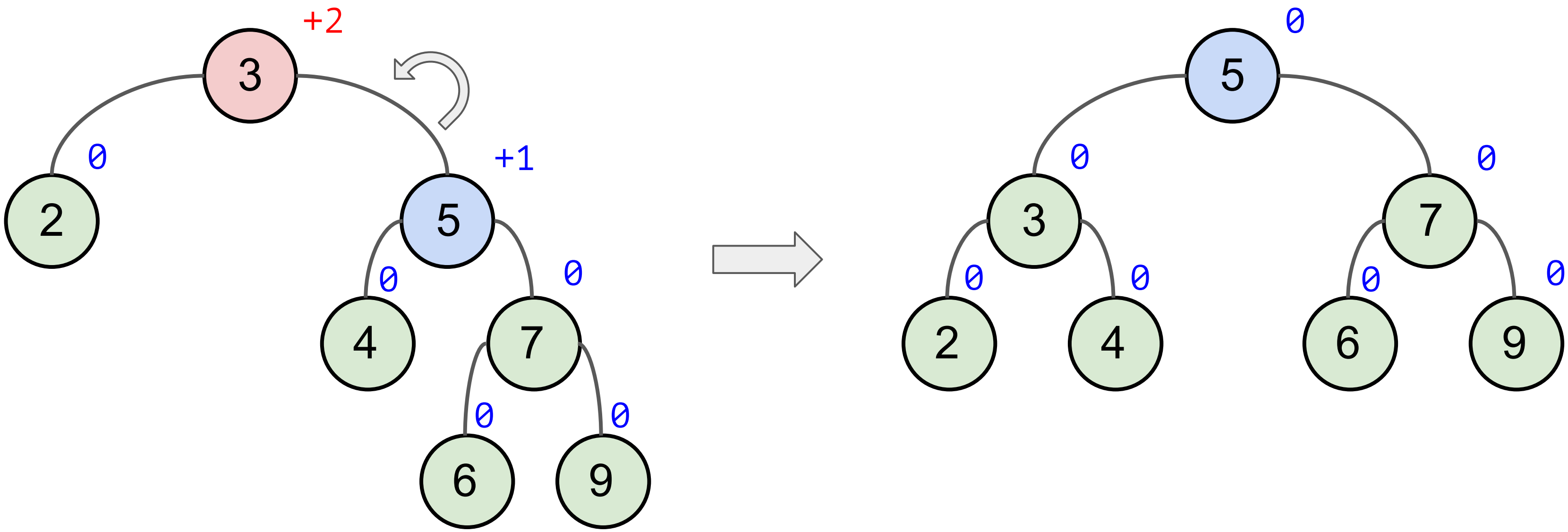
Работаем в два шага: 1) малое правое вращение вокруг 5-7
2) малое левое вращение вокруг 3-5

АВЛ-деревья: вращения



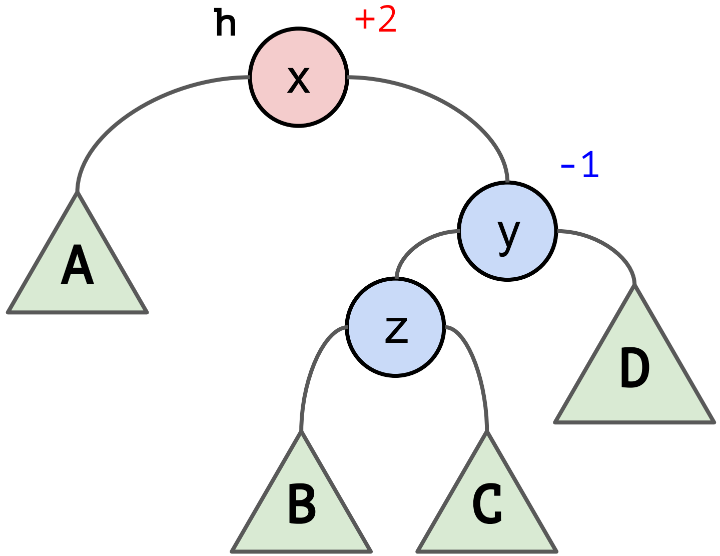

```
Работаем в два шага: 1) малое правое вращение вокруг 5-7
                     2) малое левое  вращение вокруг 3-5
```

АВЛ-деревья: вращения



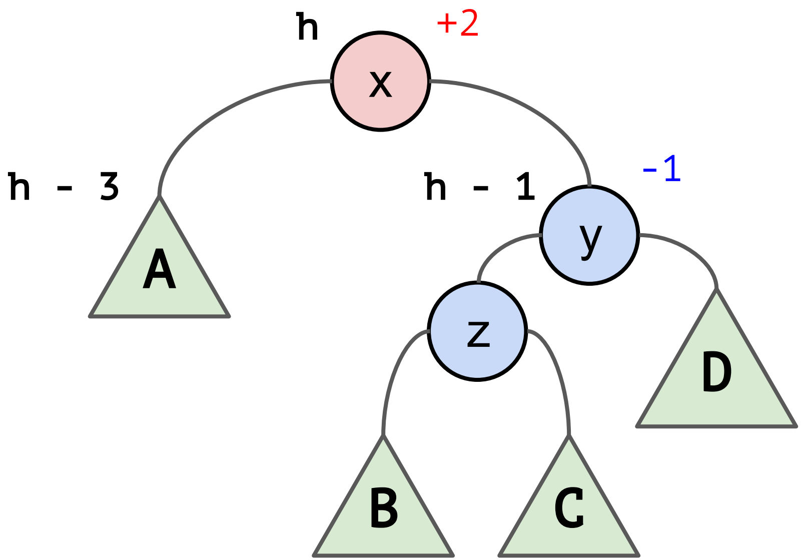

```
Работаем в два шага: 1) малое правое вращение вокруг 5-7
                     2) малое левое  вращение вокруг 3-5

Вместе эти два шага называются большим левым вращением (за O(1))
```

АВЛ-деревья: вращения



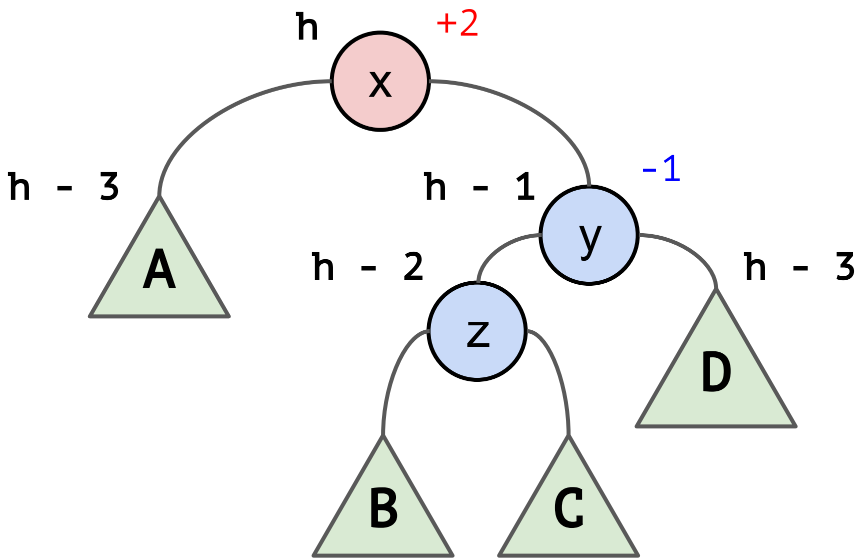
Обобщаем на произвольные деревья, распишем высоты.

АВЛ-деревья: вращения



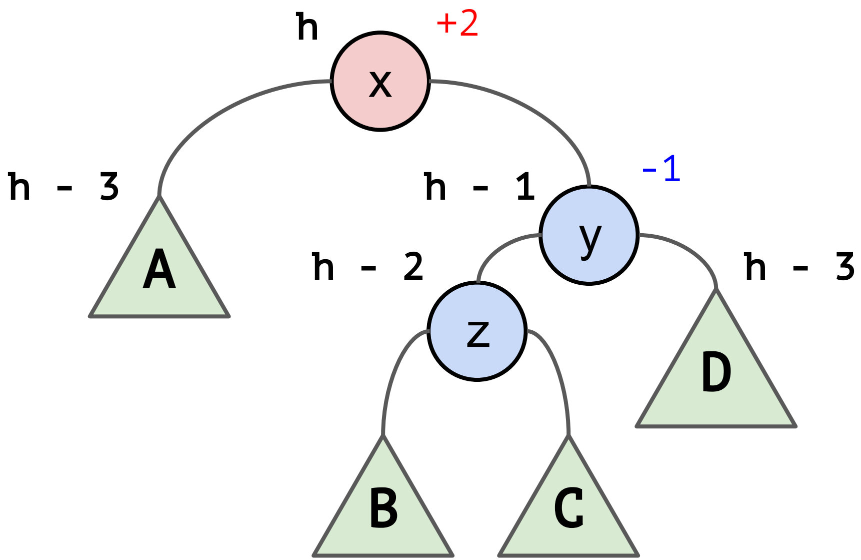
Обобщаем на произвольные деревья, распишем высоты.

АВЛ-деревья: вращения



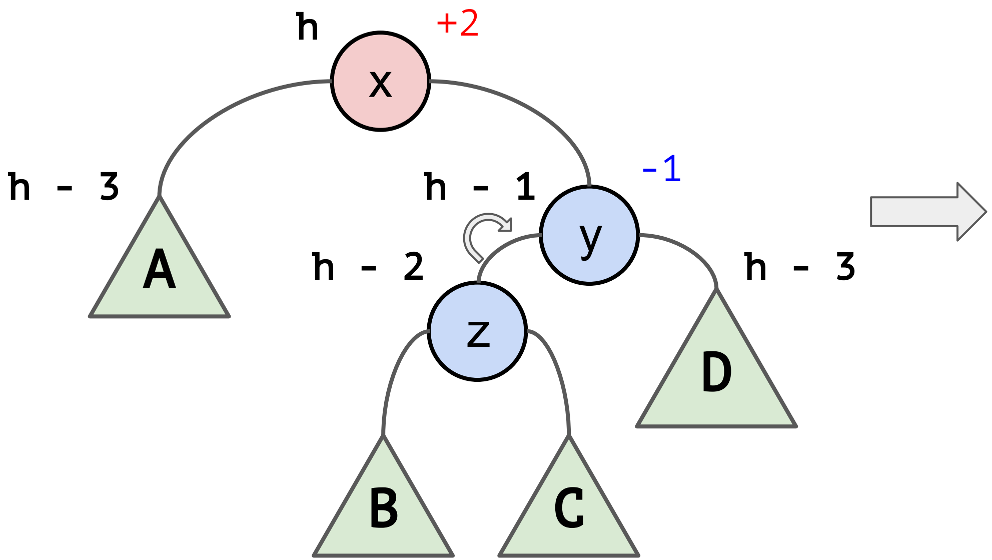
Обобщаем на произвольные деревья, распишем высоты.

АВЛ-деревья: вращения



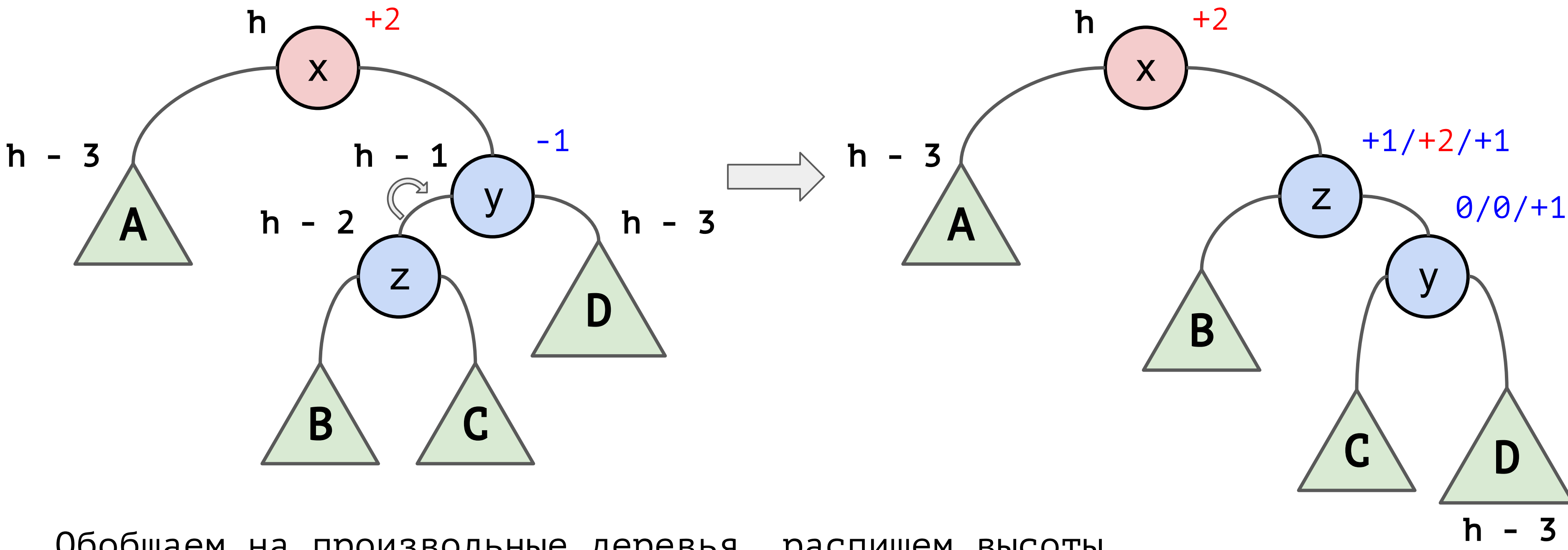
Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения



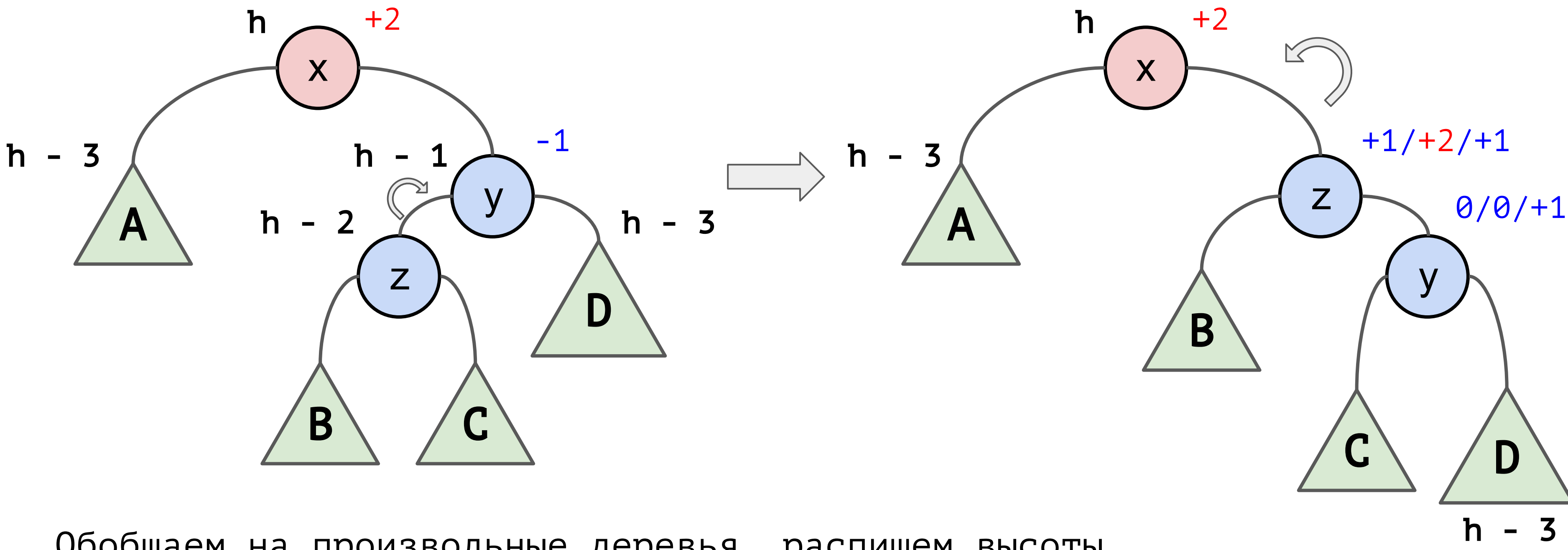
Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения



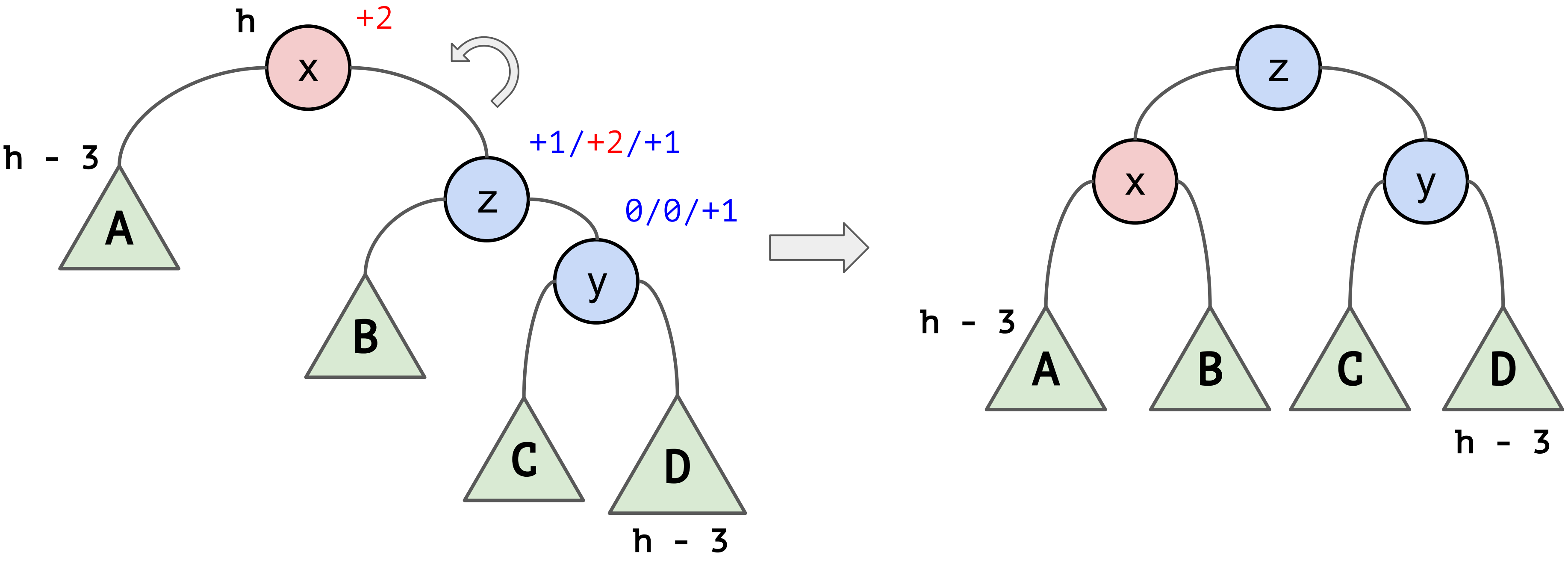
Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения



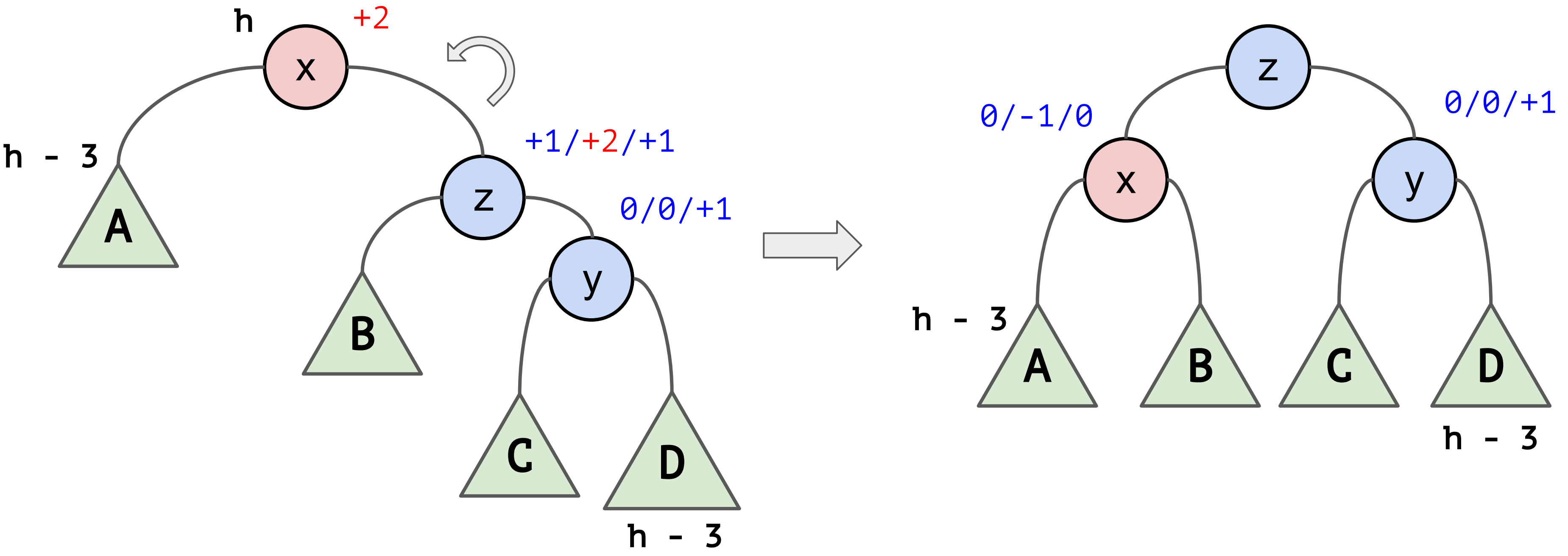
Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения



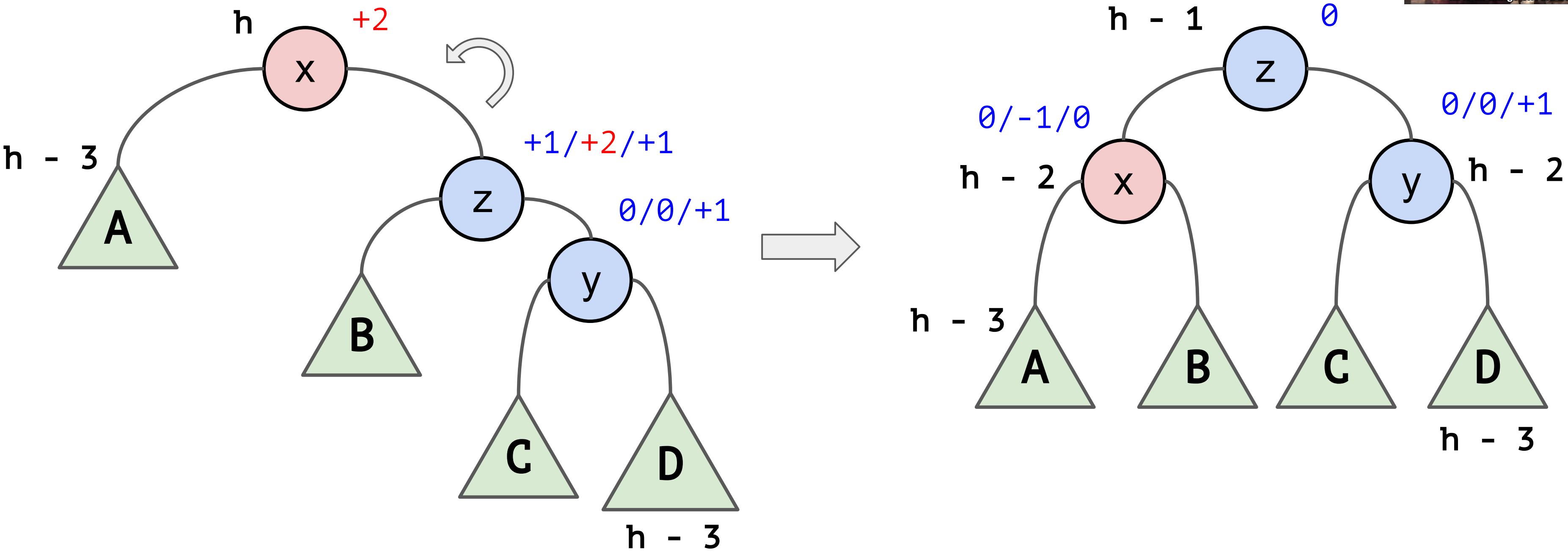
Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения



Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения



Обобщаем на произвольные деревья, распишем высоты.
Т.е. поддерево с вершиной в Z сбалансировано, то высоты B и C - (h-3, h-3) или (h-4, h-3) или (h-3, h-4)

АВЛ-деревья: вращения

Левые вращения используются в ситуации, когда у корня поддерева баланс +2 (перекос вправо) и:

- у правого сына баланс 0 и +1, тогда используем малое левое вращение

- у правого сына баланс -1, тогда используем большое левое вращение

Левые вращения используются в ситуации, когда у корня поддерева баланс +2 (перекос вправо) и:

- у правого сына баланс 0 и +1, тогда используем малое левое вращение

- у правого сына баланс -1, тогда используем большое левое вращение

Правые вращения используются в ситуации, когда у корня поддерева баланс -2 (перекос влево) и:

- у левого сына баланс 0 и -1, тогда используем малое правое вращение

- у левого сына баланс +1, тогда используем большое правое вращение

АВЛ-деревья

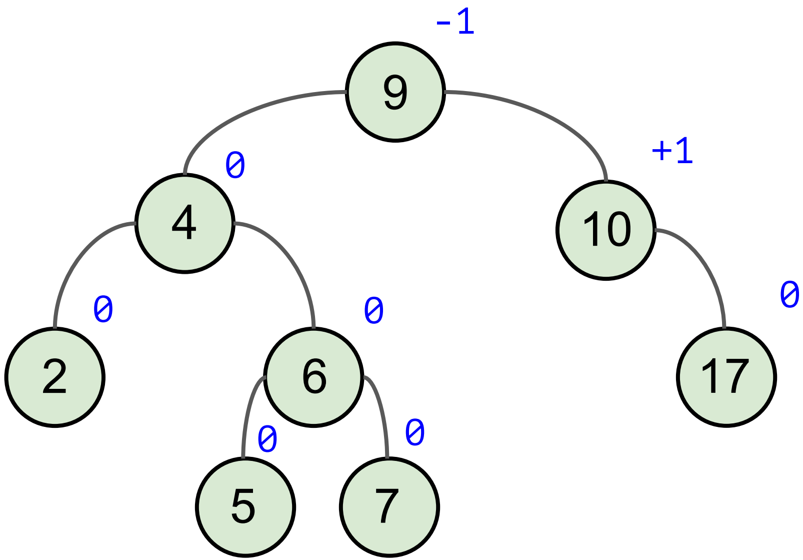
Операции:

```
1.  find(value)      -> O(logN)
2.  select(i)        -> O(logN)
3.  min/max          -> O(logN)
4.  pred/succ(ptr)   -> O(logN)
5.  rank(value)      -> O(logN)
6.  вывод в пор.     -> O(N)
    возрастания
---------------------------------

7.  insert(value)    -> O(???)
8.  remove(value)    -> O(???)
```

В АВЛ-дереве действительно высота всегда порядка logN

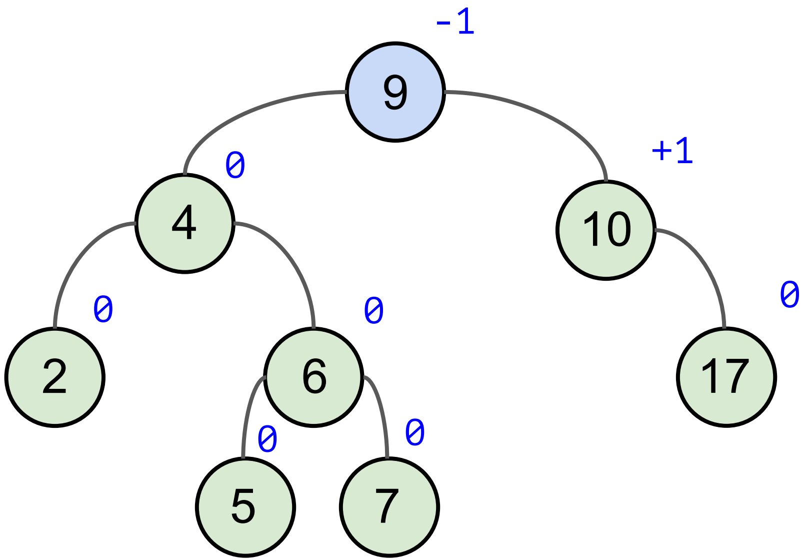
Но как поддержка инвариантов повлияет на добавление и удаление?

АВЛ-деревья: вставка



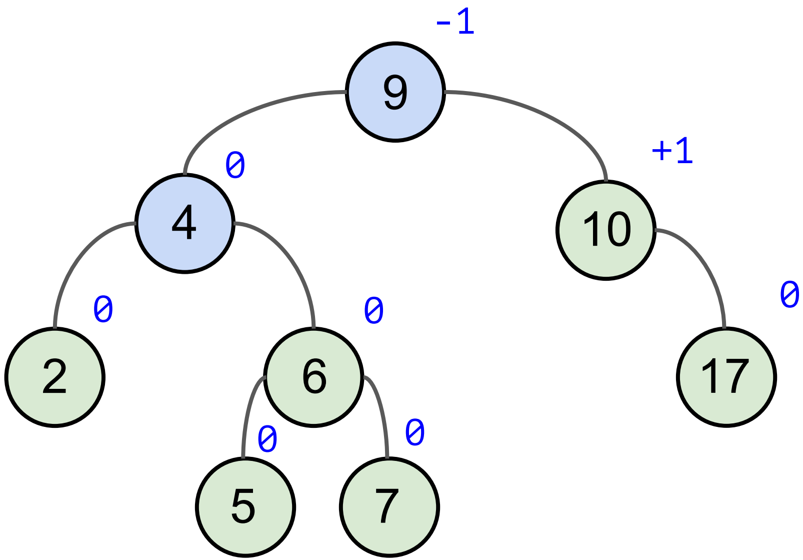
`insert(8)`

АВЛ-деревья: вставка



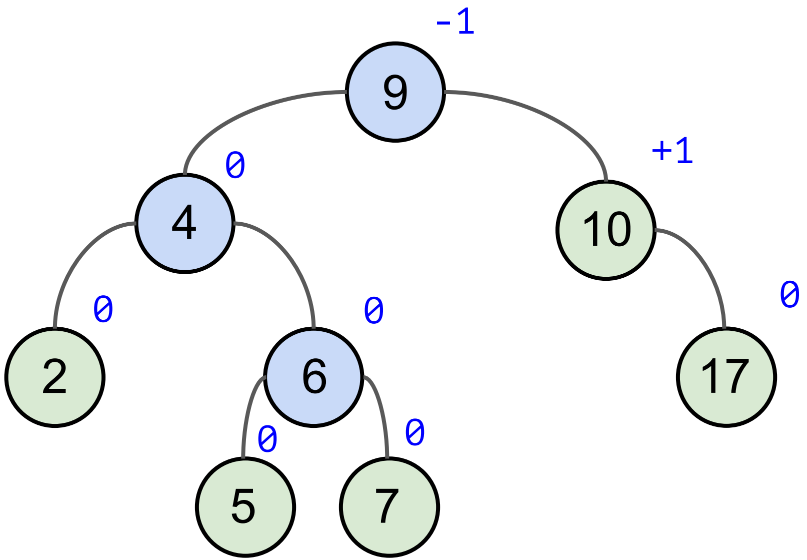
`insert(8)`

АВЛ-деревья: вставка



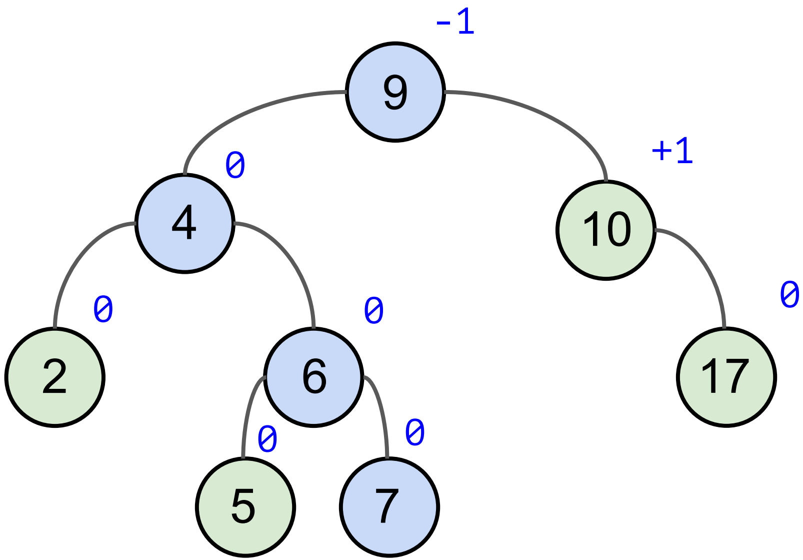
`insert(8)`

АВЛ-деревья: вставка



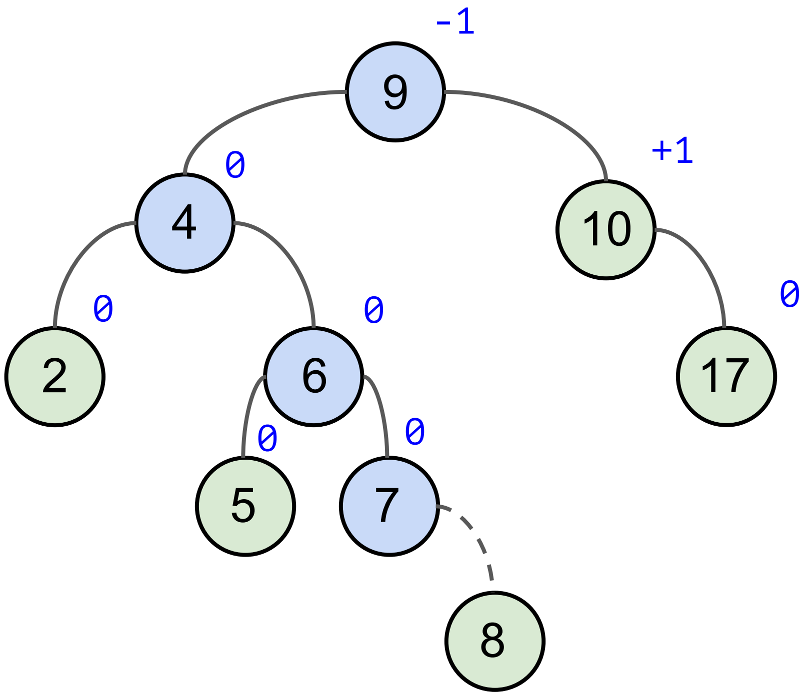
`insert(8)`

АВЛ-деревья: вставка



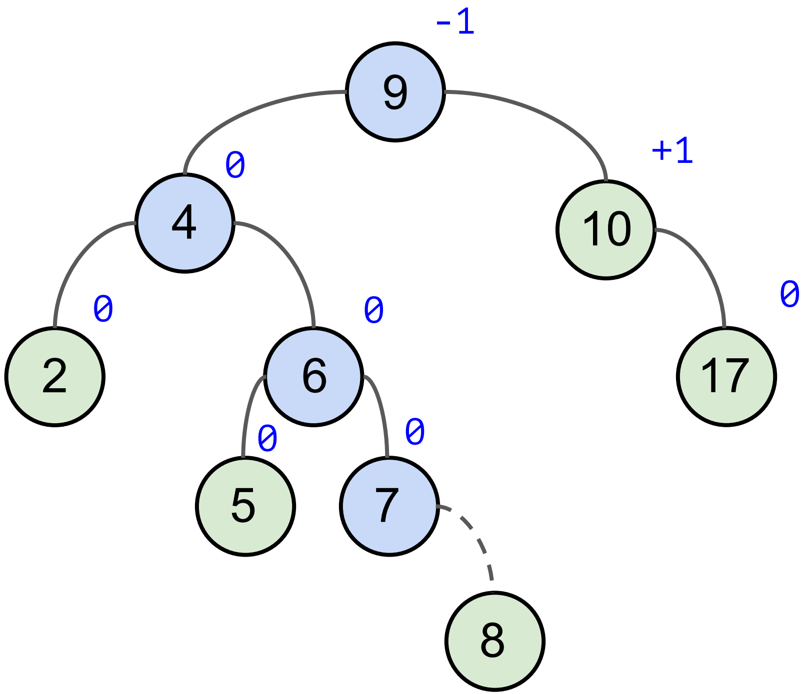
`insert(8)`

АВЛ-деревья: вставка



insert(8)

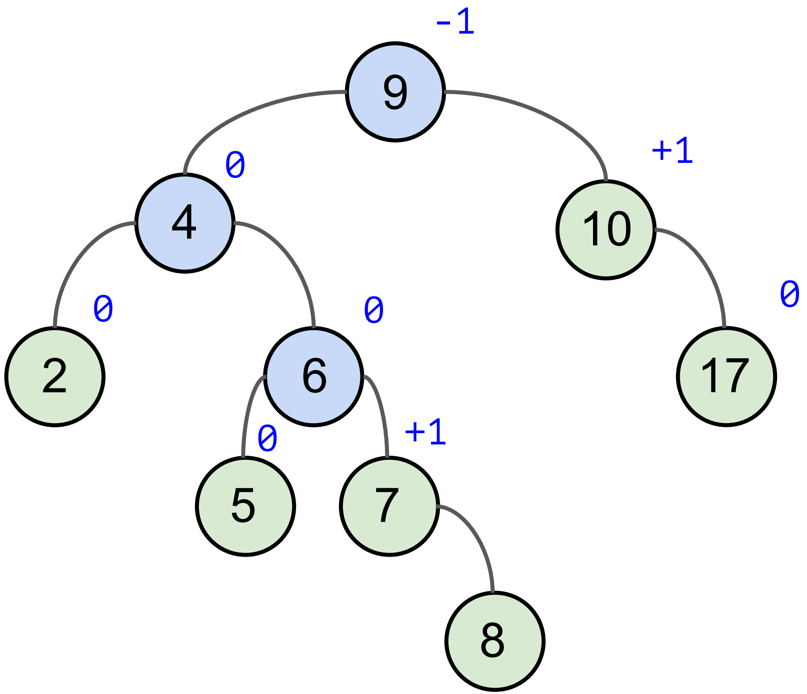
АВЛ-деревья: вставка



insert(8)

Идем обратным ходом, следим за балансами

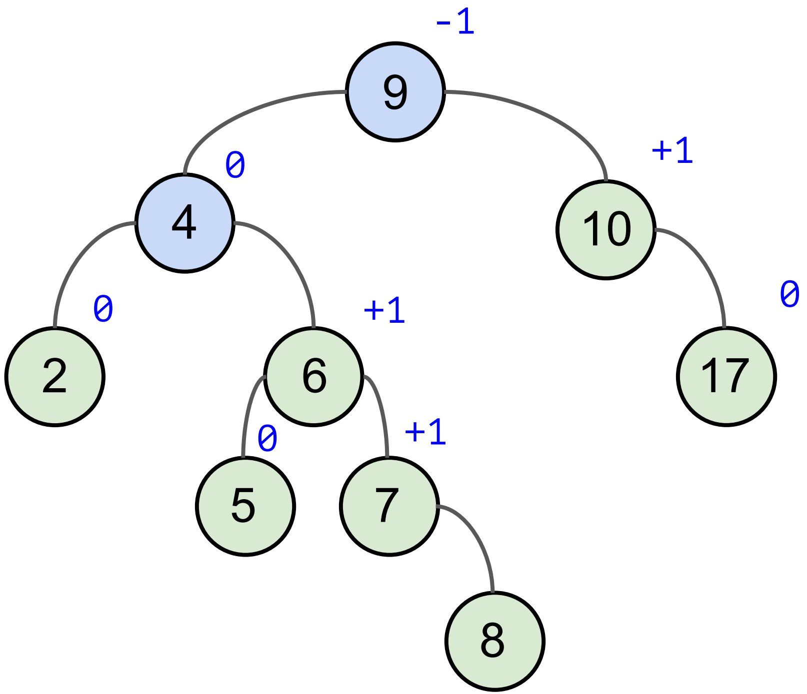
АВЛ-деревья: вставка



`insert(8)`

Идем обратным ходом, следим за балансами

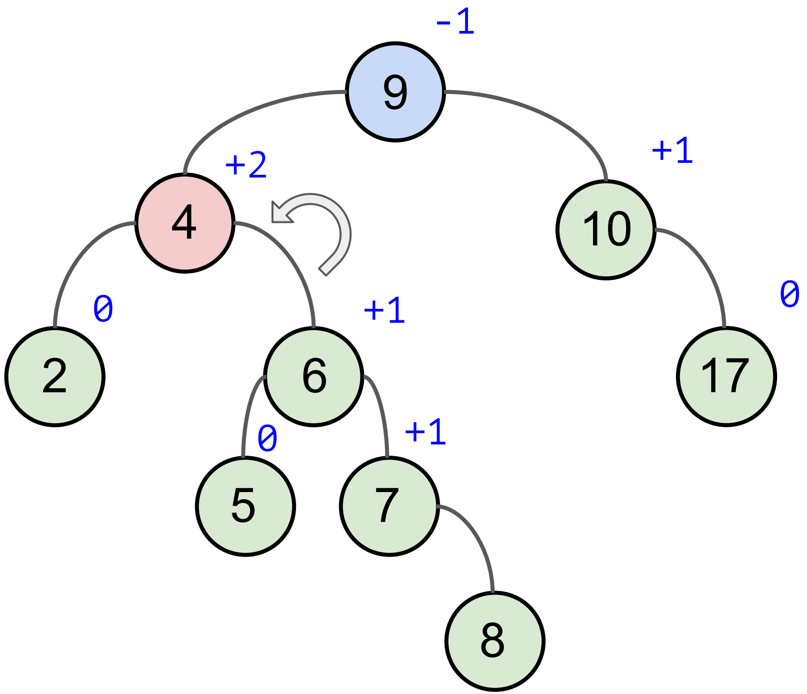
АВЛ-деревья: вставка



`insert(8)`

Идем обратным ходом, следим за балансами

АВЛ-деревья: вставка



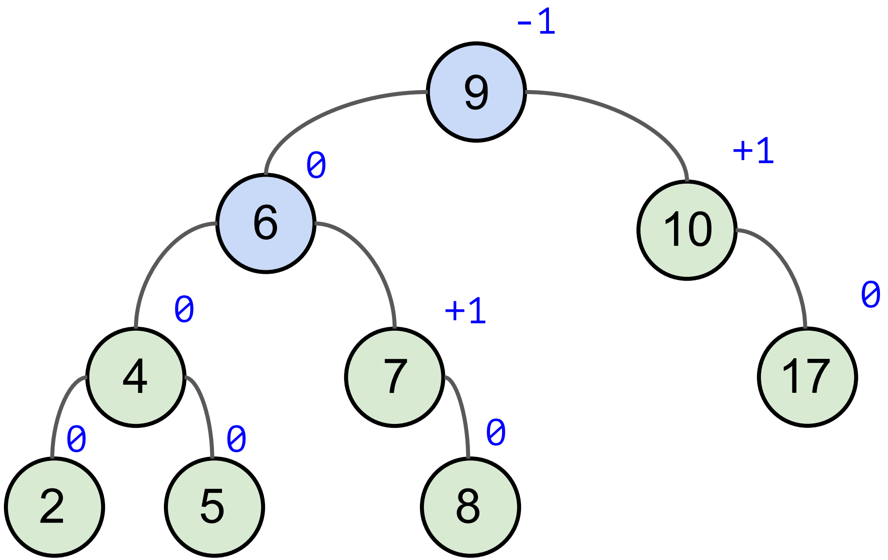
insert(8)

Идем обратным ходом, следим за балансами

Если один из балансов обновился до +2/-2 - выбираем вращение и производим его

Нужно одно малое левое вращение

АВЛ-деревья: вставка



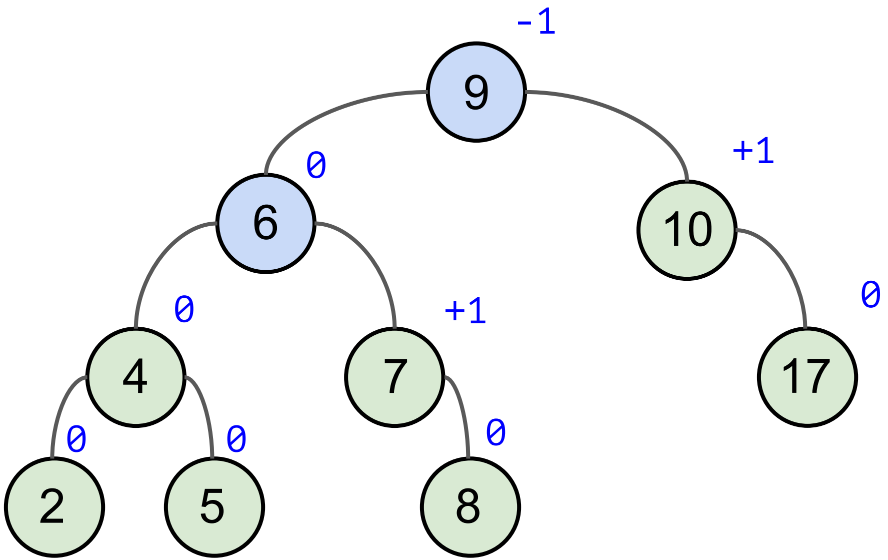
insert(8)

Идем обратным ходом, следим за балансами

Если один из балансов обновился до +2/-2 - выбираем вращение и производим его

Нужно одно малое левое вращение

АВЛ-деревья: вставка



insert(8)

Идем обратным ходом, следим за балансами

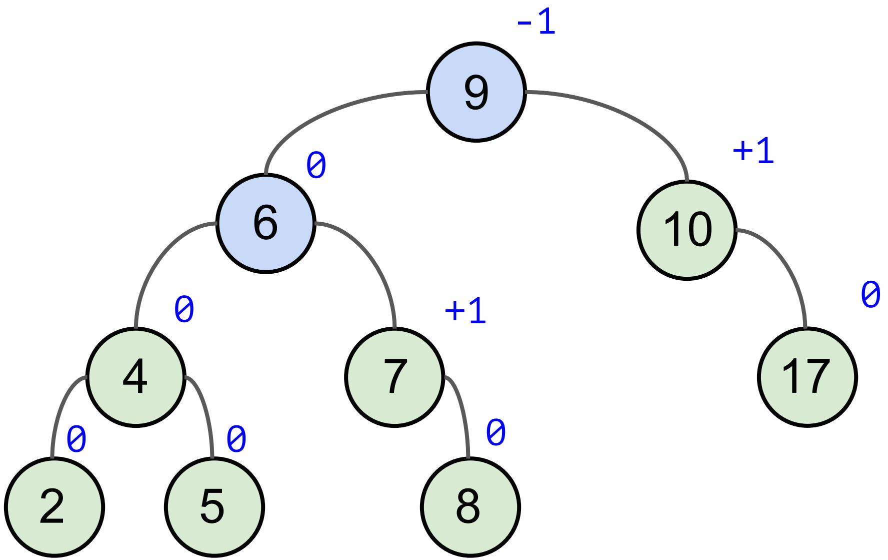
Если один из балансов обновился до +2/-2 - выбираем вращение и производим его

Нужно одно малое левое вращение

Если баланс не 0, идем выше, обновляем балансы и дальше.

АВЛ-деревья: вставка

В любом случае сложность - O(logN), т.к. вращение работают за O(1), а длина обратного хода - высота дерева



insert(8)

Идем обратным ходом, следим за балансами

Если один из балансов обновился до +2/-2 - выбираем вращение и производим его

Нужно одно малое левое вращение

Если баланс не 0, идем выше, обновляем балансы и дальше.

АВЛ-деревья

Операции:

```
1.  find(value)     -> O(logN)
2.  select(i)       -> O(logN)
3.  min/max         -> O(logN)
4.  pred/succ(ptr)  -> O(logN)
5.  rank(value)     -> O(logN)
6.  вывод в пор.    -> O(N)
    возрастания
--------------------------------
7.  insert(value)   -> O(???)
8.  remove(value)   -> O(???)
```

В АВЛ-дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

АВЛ-деревья

Операции:

```
1.  find(value)     -> O(logN)
2.  select(i)       -> O(logN)
3.  min/max         -> O(logN)
4.  pred/succ(ptr) -> O(logN)
5.  rank(value)     -> O(logN)
6.  вывод в пор.    -> O(N)
    возрастания
---------------------------------
7.  insert(value)   -> O(logN)
8.  remove(value)   -> O(???)
```

В АВЛ-дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

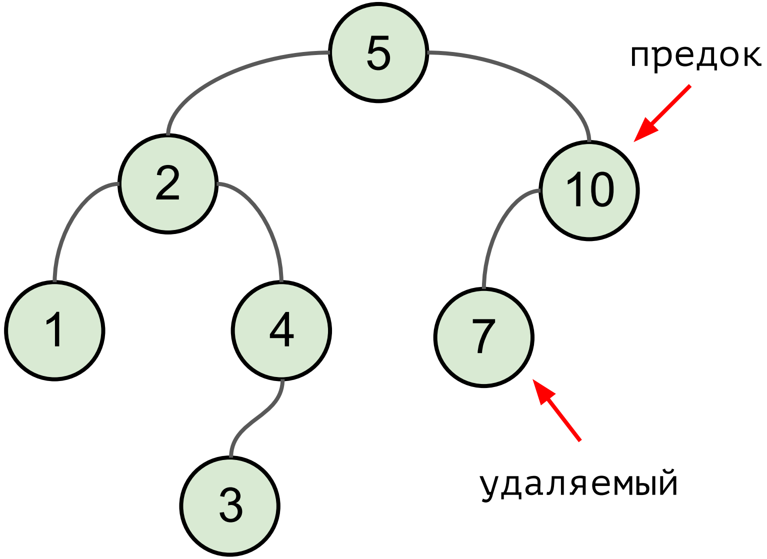
Мини-задача #26 (1 балл)

Реализовать процедуру которая принимает BST, а возвращает сбалансированное BST из тех же значений.

Сбалансированным будем называть BST, в котором высота поддеревьев для каждой вершины отличается не более, чем на 1.

https://leetcode.com/problems/balance-a-binary-search-tree

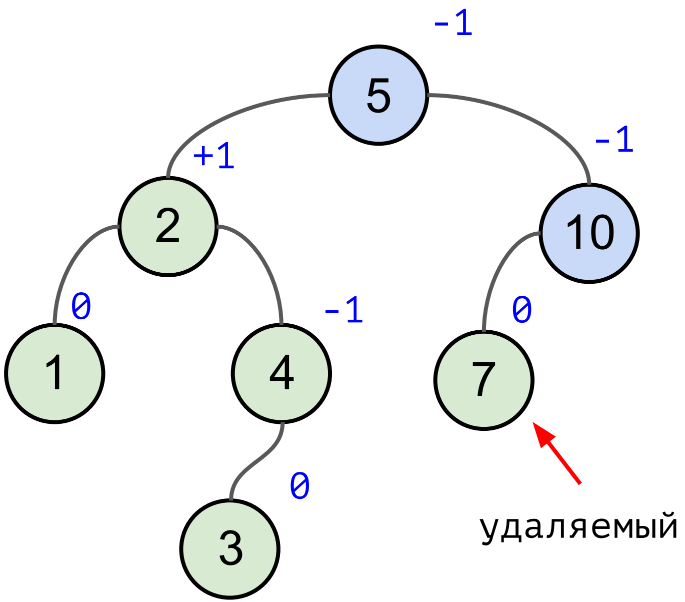
Бинарные деревья поиска: удаление



`remove(7)`

1) случай - удаляем лист

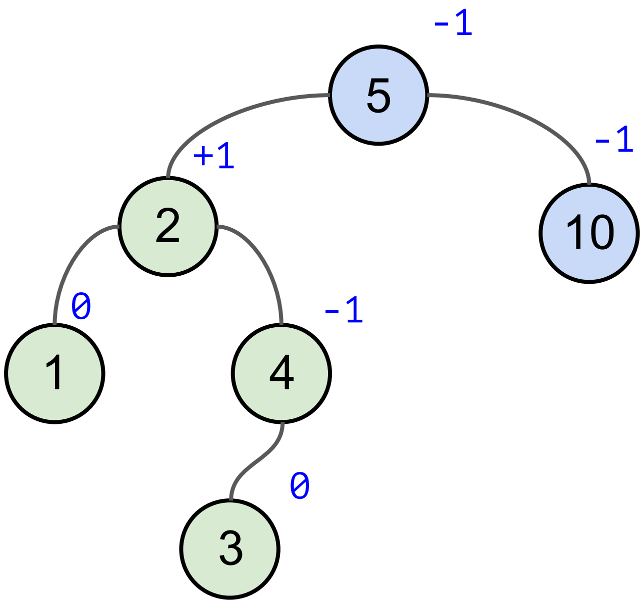
АВЛ-деревья: удаление



remove(7)

1) случай - удаляем лист

АВЛ-деревья: удаление

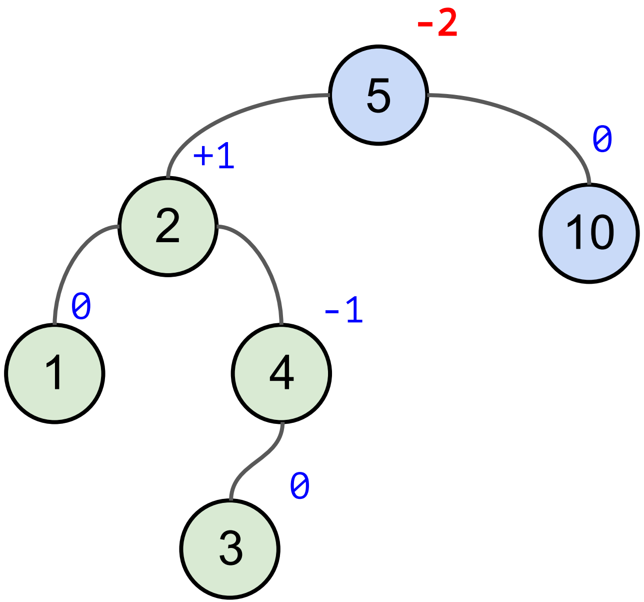


`remove(7)`

1) случай - удаляем лист

Опять - нужно идти обратным ходом и проверять балансы

АВЛ-деревья: удаление

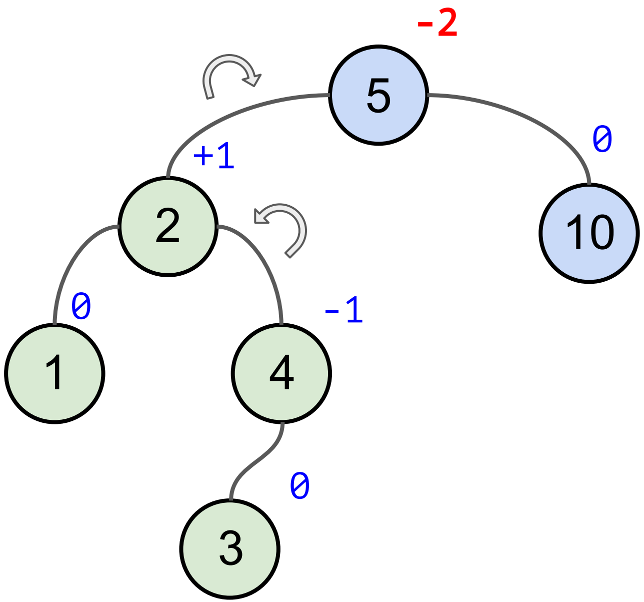



```
remove(7)
```

```
1) случай - удаляем лист
```

```
Опять - нужно идти
обратным ходом и
проверять балансы
```

АВЛ-деревья: удаление



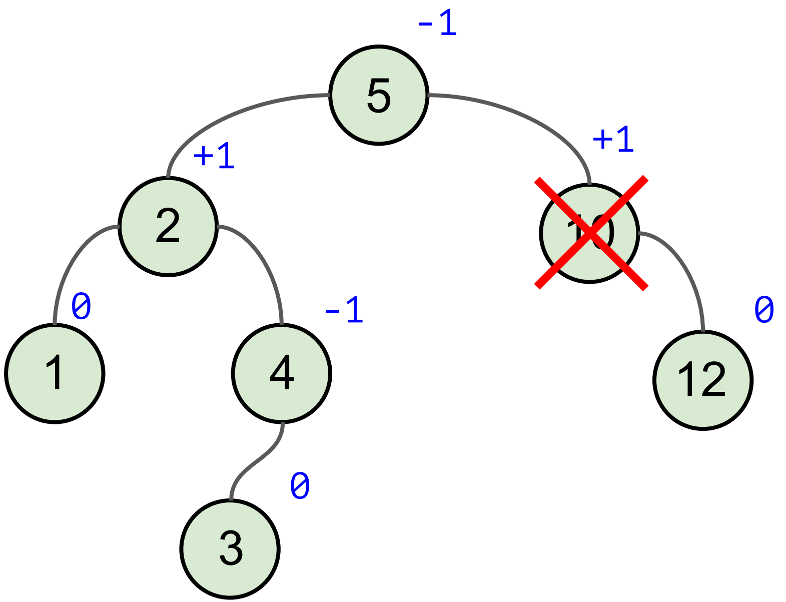
`remove(7)`

1) случай - удаляем лист

Опять - нужно идти обратным ходом и проверять балансы

В случае нарушения выбираем правильное вращение и проводим его

АВЛ-деревья: удаление

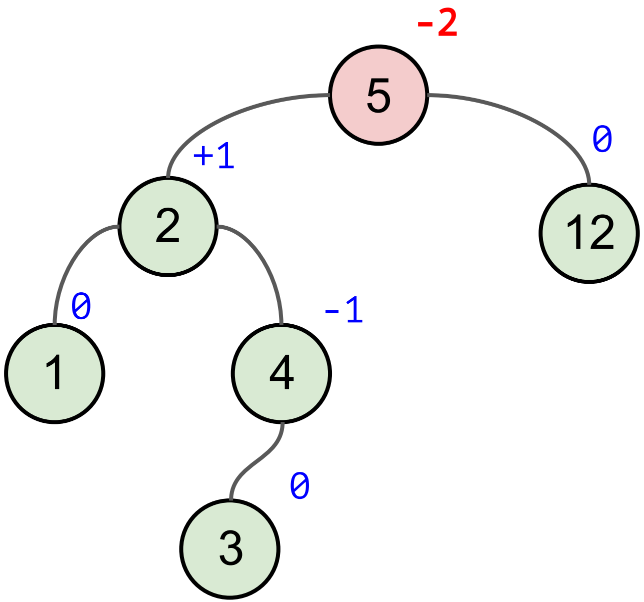


remove(10)

1) случай - удаляем лист

2) удаляем вершину с одним наследником

АВЛ-деревья: удаление



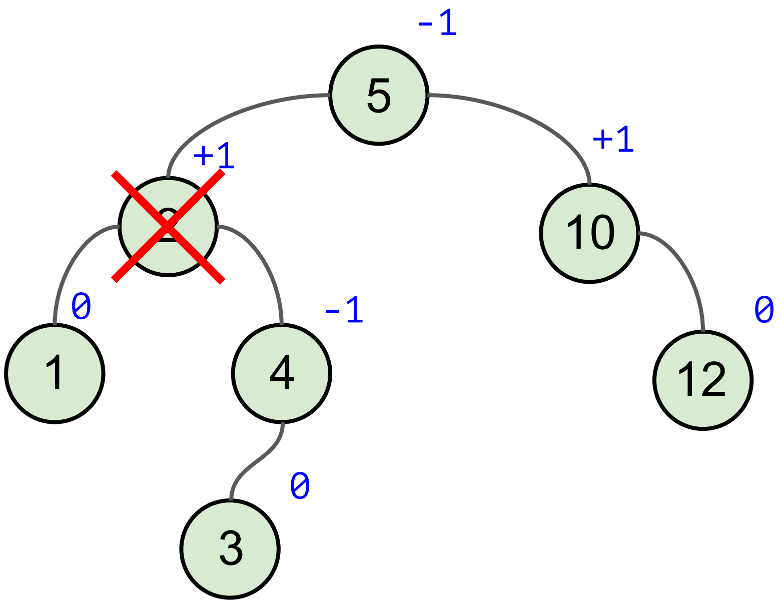
remove(10)

1) случай - удаляем лист

2) удаляем вершину с одним наследником

Аналогично: идем обратным ходом, выправляем балансы через вращения

АВЛ-деревья: удаление



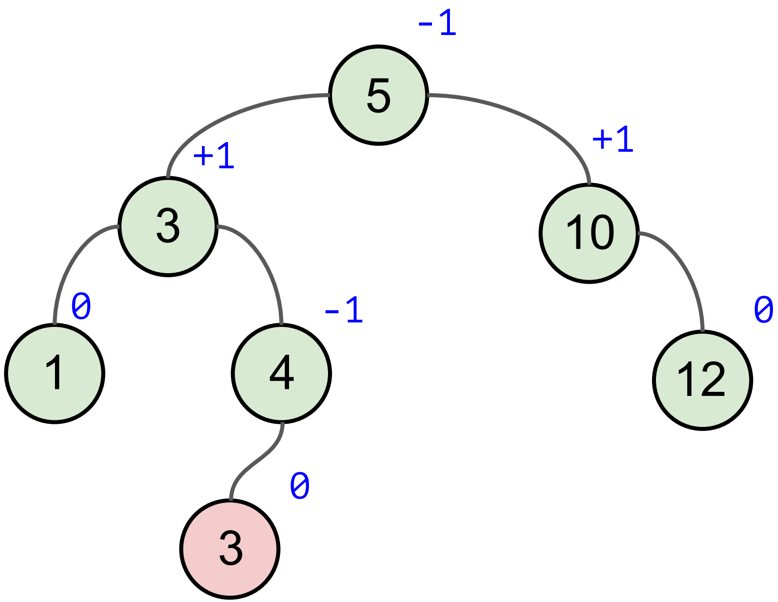

```
remove(2)
```

1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

АВЛ-деревья: удаление



remove(2)

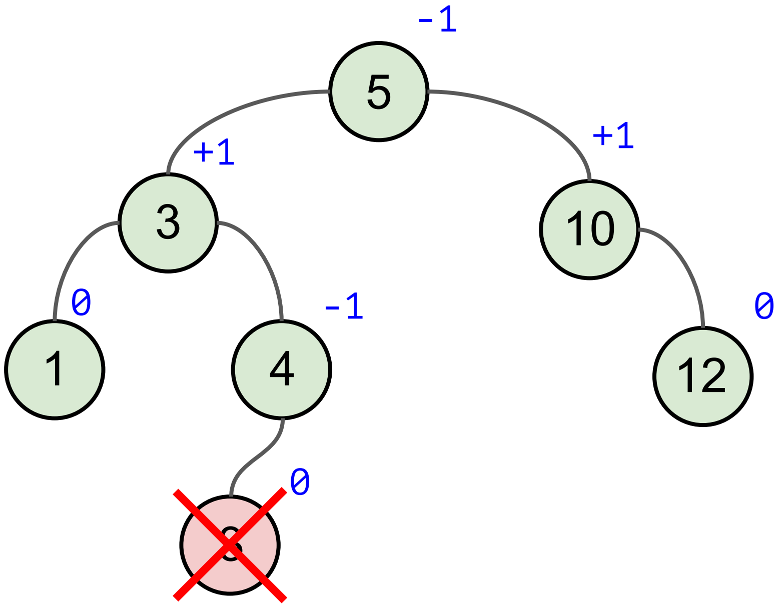
1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

Заменяем на минимум из правого поддерева

АВЛ-деревья: удаление



remove(2)

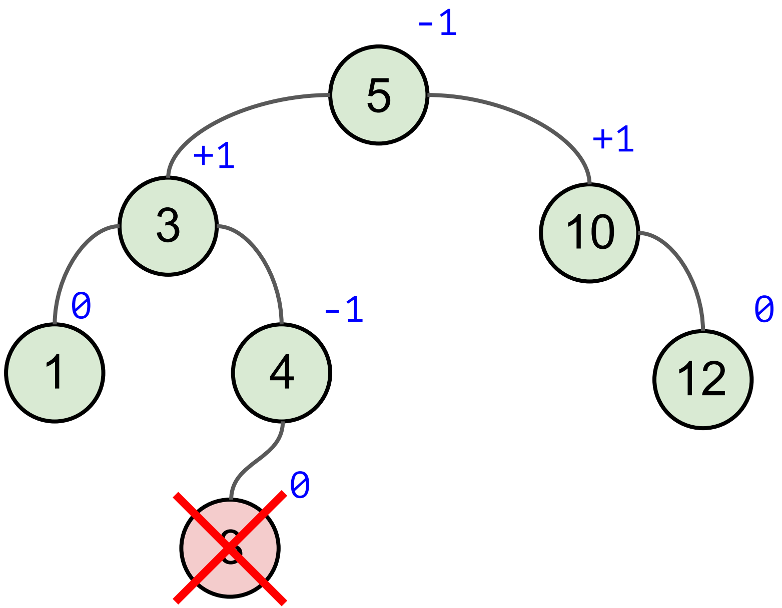
1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

Сводим задачу к предыдущим

АВЛ-деревья: удаление



В любом случае сложность - $O(\log N)$, т.к. вращение работают за O(1), а длина обратного хода - высота дерева

remove(2)

1) случай - удаляем лист

2) удаляем вершину с одним наследником

3) удаляем вершину с двумя наследниками

Сводим задачу к предыдущим

АВЛ-деревья

Операции:

```
1.  find(value)     -> O(logN)
2.  select(i)       -> O(logN)
3.  min/max         -> O(logN)
4.  pred/succ(ptr)  -> O(logN)
5.  rank(value)     -> O(logN)
6.  вывод в пор.    -> O(N)
    возрастания
---------------------------------

7.  insert(value)   -> O(logN)
8.  remove(value)   -> O(???)
```

В АВЛ-дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

АВЛ-деревья

Операции:

1. find(value) -> O(logN)
2. select(i) -> O(logN)
3. min/max -> O(logN)
4. pred/succ(ptr) -> O(logN)
5. rank(value) -> O(logN)
6. вывод в пор. возрастания -> O(N)

---

7. insert(value) -> O(logN)
8. remove(value) -> O(logN)

В АВЛ-дереве действительно высота всегда порядка logN

И все операции продолжают работать за логарифм!

Красно-чёрные деревья

Авторы: Leonidas J. Guibas, Robert Sedgewick, 1978

Идея: возьмем обычное BST и добавим ограничения

Красно-чёрные деревья

Авторы: Leonidas J. Guibas, Robert Sedgewick, 1978

Идея: возьмем обычное BST и добавим ограничения

1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**

Красно-чёрные деревья

Авторы: Leonidas J. Guibas, Robert Sedgewick, 1978

Идея: возьмем обычное BST и добавим ограничения

1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**
3. двух красных вершин подряд быть не может (т.е. все дети красной обязательно **чёрные**)

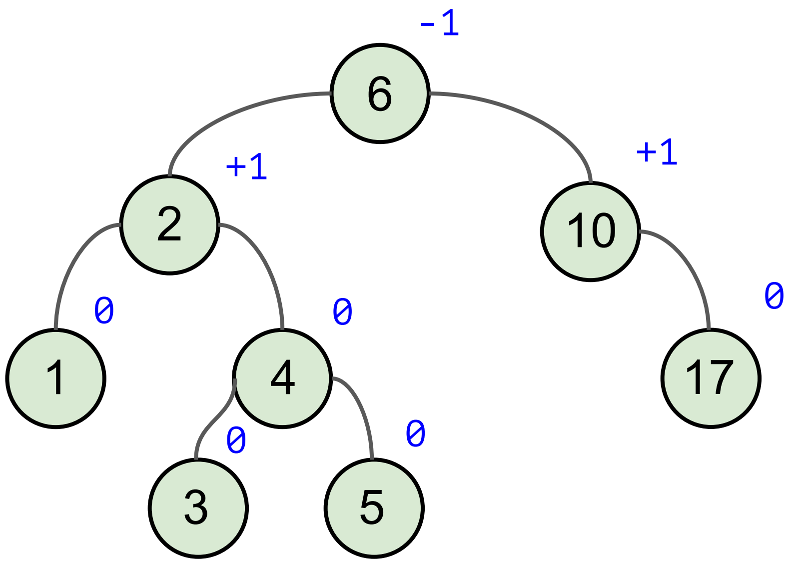
Красно-чёрные деревья

Авторы: Leonidas J. Guibas, Robert Sedgewick, 1978

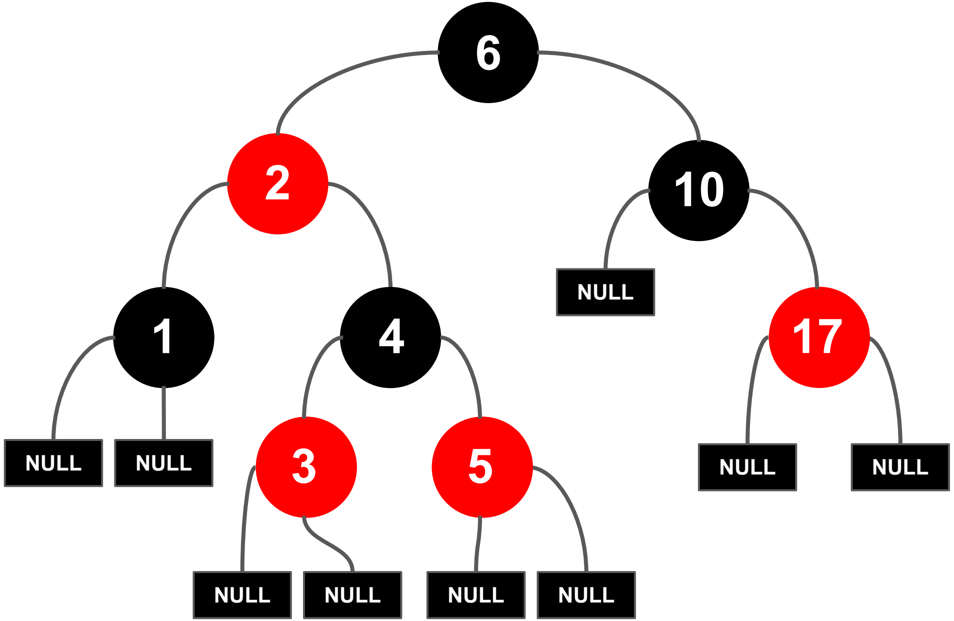
Идея: возьмем обычное BST и добавим ограничения

1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**
3. двух красных вершин подряд быть не может (т.е. все дети красной обязательно **чёрные**)
4. любой путь от корня до листа содержит одинаковое количество **чёрных** вершин

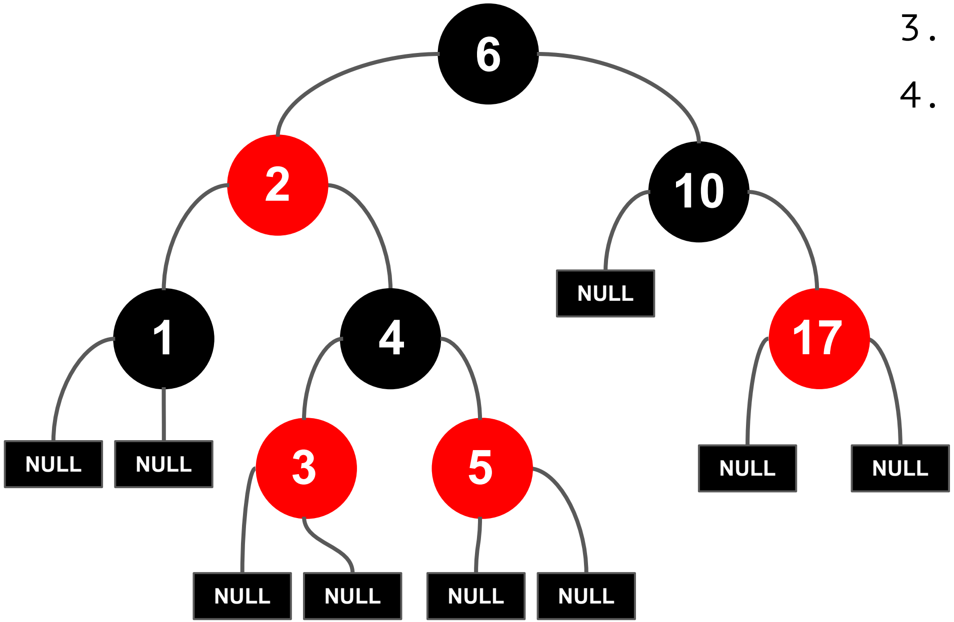
Красно-чёрные деревья



Красно-чёрные деревья

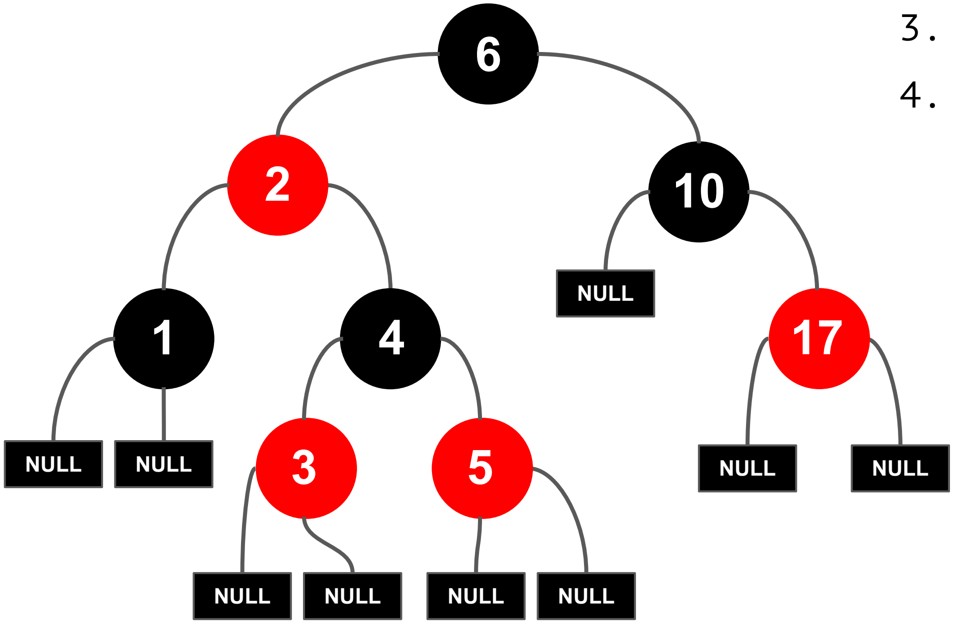


Красно-чёрные деревья



1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**
3. двух красных вершин подряд быть не может
4. любой путь от корня до листа содержит одинаковое количество **чёрных** вершин

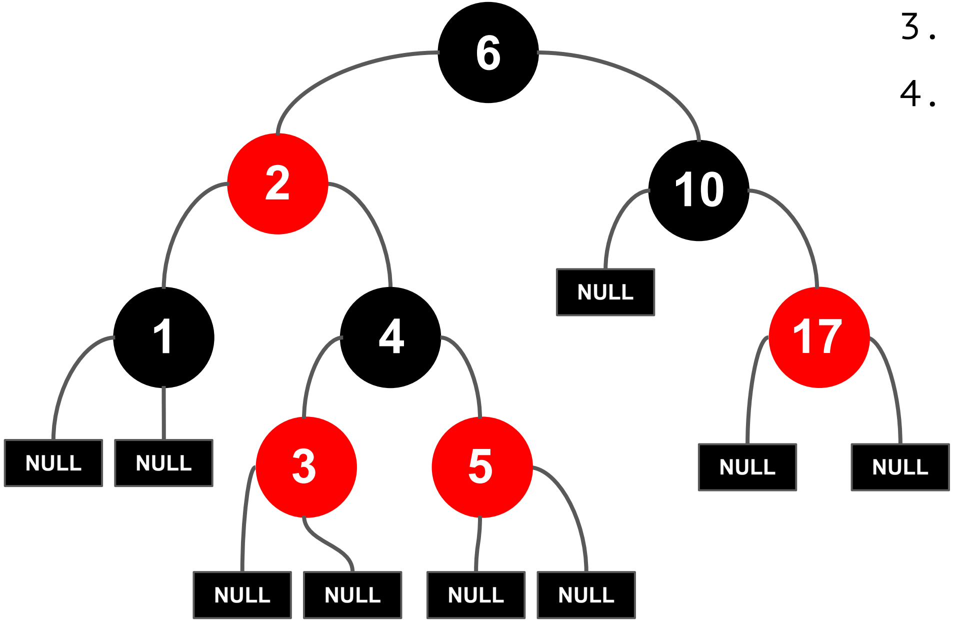
Красно-чёрные деревья



1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**
3. двух красных вершин подряд быть не может
4. любой путь от корня до листа содержит одинаковое количество **чёрных** вершин

В отличие от АВЛ, для NULL значений при визуализации часто изображают отдельную вершину, так удобнее доказывать четвертое свойство.

Красно-чёрные деревья

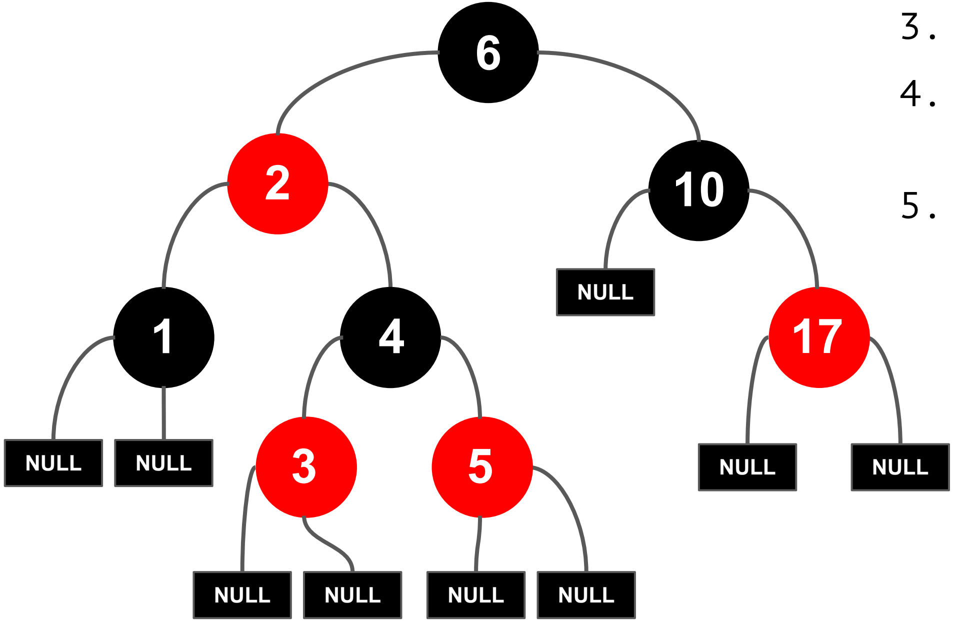


1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**
3. двух красных вершин подряд быть не может
4. любой путь от корня до листа содержит одинаковое количество **чёрных** вершин

В отличие от АВЛ, для NULL значений при визуализации часто изображают отдельную вершину, так удобнее доказывать четвертое свойство.

Но это не значит, что такие вершины и правда существуют при реализации, это абстрактное понятие.

Красно-чёрные деревья

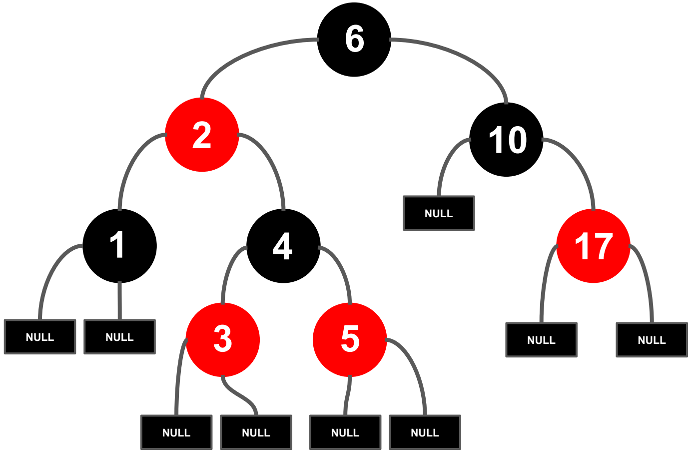


1. каждая вершина либо красная либо **чёрная**
2. корень **чёрный**
3. двух красных вершин подряд быть не может
4. любой путь от корня до листа содержит одинаковое количество **чёрных** вершин
5. любой лист - это фиктивная NULL вершина, у нее нет детей, а сама она окрашена в **чёрный** цвет.

У остальных вершин всегда ровно 2 сына.

Красно-чёрные деревья

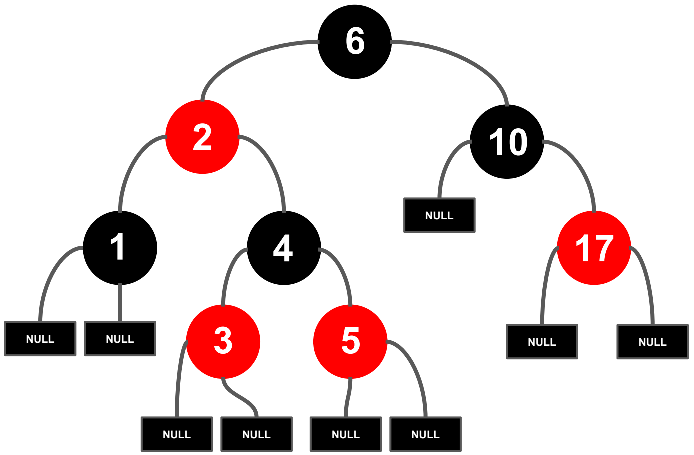
Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$



Красно-чёрные деревья

Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

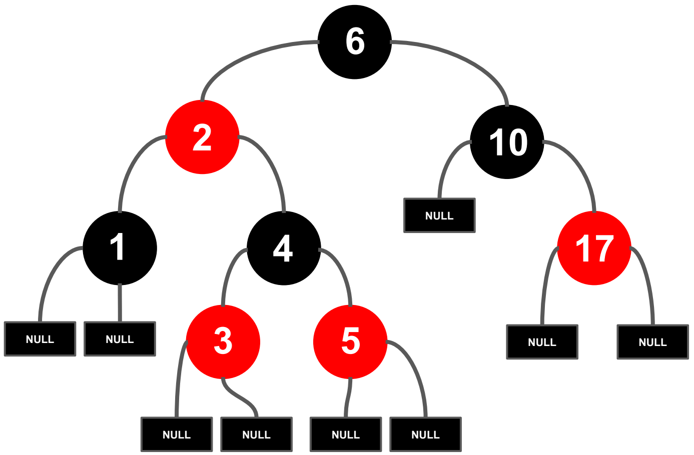


Красно-чёрные деревья

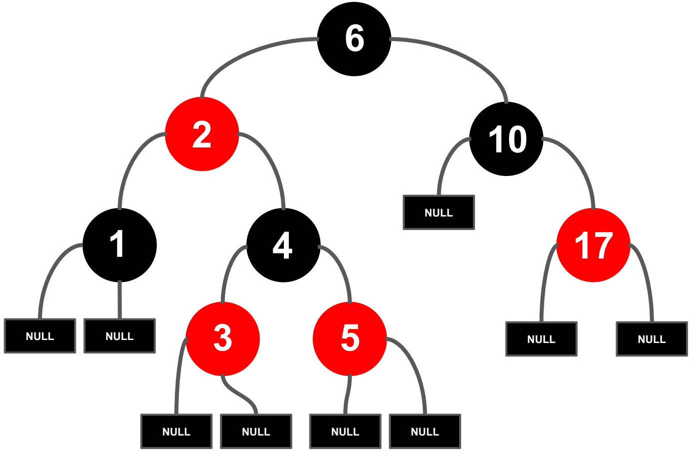
Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве;
bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

Тогда докажем, что в поддереве с корнем x находится реальных вершин больше, чем $2^{bh(x)} - 1$



Красно-чёрные деревья



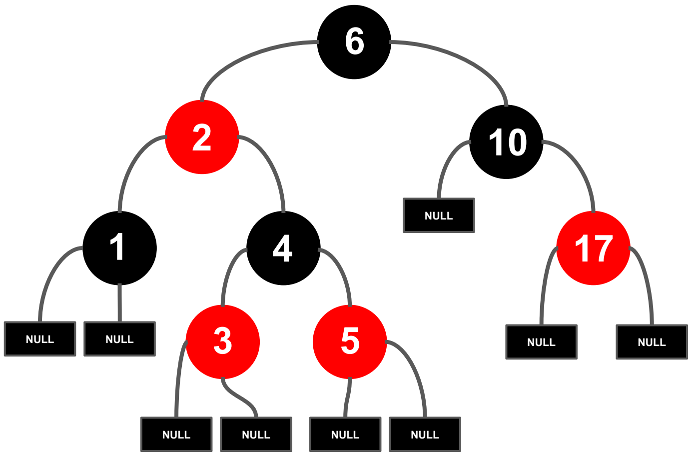
Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

Тогда докажем, что в поддереве с корнем x находится реальных вершин больше, чем $2^{bh(x)} - 1$

Докажем это по индукции по высоте поддерева с корнем в x: h(x)

Красно-чёрные деревья



Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

Тогда докажем, что в поддереве с корнем x находится реальных вершин больше, чем $2^{bh(x)} - 1$

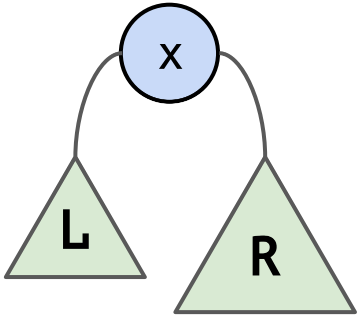
Докажем это по индукции по высоте поддерева с корнем в x: h(x)

База: h(x) = 0 => это фиктивный лист => bh(x) = 0 и $2^{bh(x)} - 1$ = 0. А реальных вершин - нет, так что все сходится.

Красно-чёрные деревья

База: h(x) = 0 => это фиктивный лист => bh(x) = 0 и $2^{bh(x)} - 1$ = 0. А реальных вершин - нет, так что все сходится.

Шаг: пусть верно для всех меньших высот, рассмотрим дерево с вершиной в x.

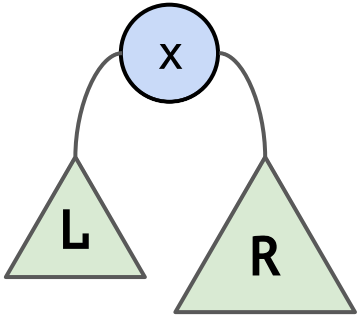


Как связаны чёрная высота x и чёрные высоты от вершин поддеревьев L и R?

Красно-чёрные деревья

База: h(x) = 0 => это фиктивный лист => bh(x) = 0 и $2^{bh(x)} - 1$ = 0. А реальных вершин - нет, так что все сходится.

Шаг: пусть верно для всех меньших высот, рассмотрим дерево с вершиной в x.



Как связаны чёрная высота x и чёрные высоты от вершин поддеревьев L и R?

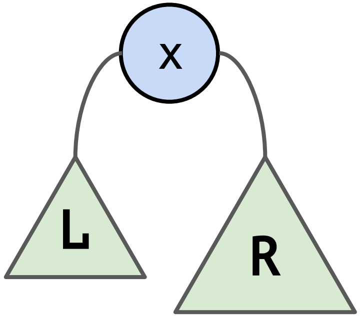
bh(L) ⩾ bh(x) - 1
bh(R) ⩾ bh(x) - 1

(т.к. отрезали мы либо чёрную и тогда путь уменьшился, либо красную и тогда он в точности равен).

Красно-чёрные деревья

База: h(x) = 0 => это фиктивный лист => bh(x) = 0 и $2^{bh(x)}-1$ = 0. А реальных вершин - нет, так что все сходится.

Шаг: пусть верно для всех меньших высот, рассмотрим дерево с вершиной в x.



Как связаны чёрная высота x и чёрные высоты от вершин поддеревьев L и R?

bh(L) ⩾ bh(x) - 1
bh(R) ⩾ bh(x) - 1

(т.к. отрезали мы либо чёрную и тогда путь уменьшился, либо красную и тогда он в точности равен).

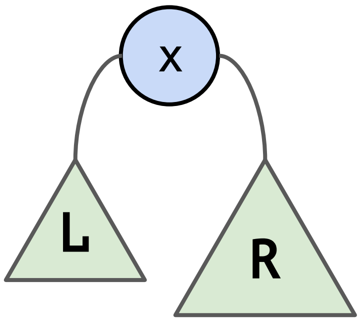
Тогда суммарно в вершин поддереве X:

$$size(x) = 1 + size(L) + size(R) \geq 1 + 2^{bh(L)} - 1 + 2^{bh(R)} - 1$$
$$\geq 2^{bh(x)-1} + 2^{bh(x)-1} - 1 = 2^{bh(x)} - 1$$

Красно-чёрные деревья

База: h(x) = 0 => это фиктивный лист => bh(x) = 0 и $2^{bh(x)} - 1$ = 0. А реальных вершин - нет, так что все сходится.

Шаг: пусть верно для всех меньших высот, рассмотрим дерево с вершиной в x.



Как связаны чёрная высота x и чёрные высоты от вершин поддеревьев L и R?

bh(L) ⩾ bh(x) - 1
bh(R) ⩾ bh(x) - 1

(т.к. отрезали мы либо чёрную и тогда путь уменьшился, либо красную и тогда он в точности равен).

Тогда суммарно в вершин поддереве X:

$$\boxed{size(x)} = 1 + size(L) + size(R) \geq 1 + 2^{bh(L)} - 1 + 2^{bh(R)} - 1$$

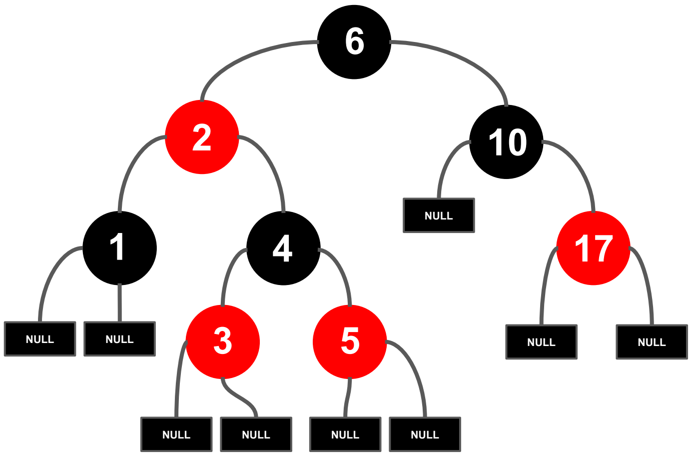
$$\boxed{\geq} 2^{bh(x)-1} + 2^{bh(x)-1} - 1 = \boxed{2^{bh(x)} - 1}$$

Красно-чёрные деревья

Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

Тогда $size(x) \geq 2^{bh(x)} - 1$

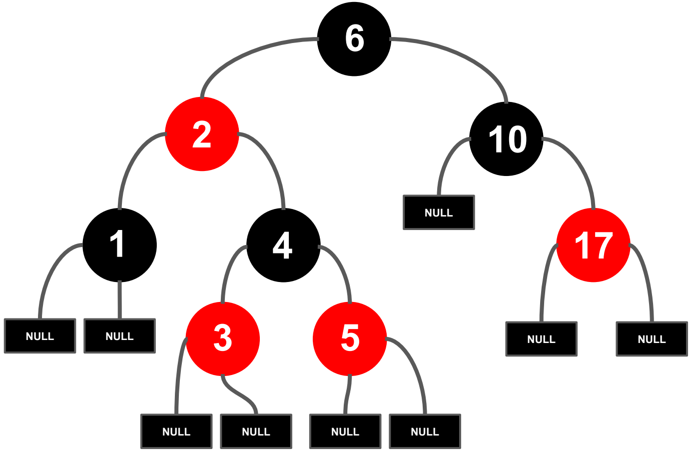


Красно-чёрные деревья

Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

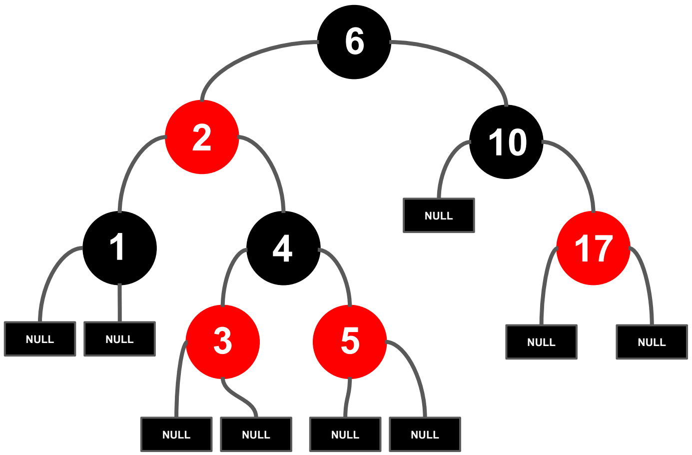
Тогда $size(x) \geq 2^{bh(x)} - 1$



Пусть теперь h - высота всего дерева. Утверждается, что $bh(root) \geq \frac{h}{2}$

Красно-чёрные деревья

Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$



Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

Тогда $size(x) \geq 2^{bh(x)} - 1$

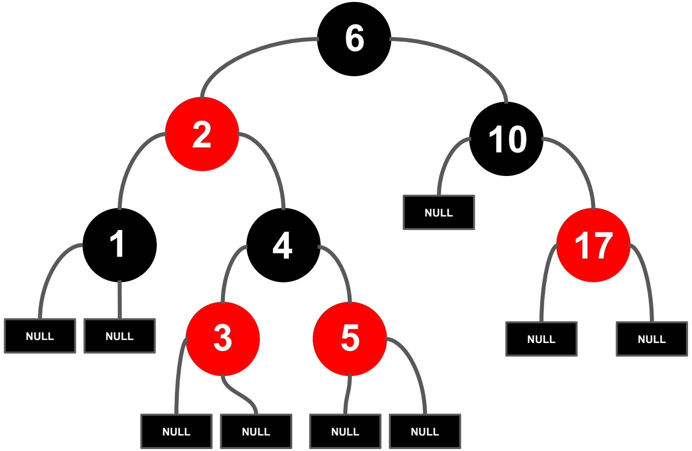
Пусть теперь h - высота всего дерева. Утверждается, что $bh(root) \geq \frac{h}{2}$
Действительно: т.к. не бывает двух красных вершин подряд, то **чёрная** высота уж точно будет не меньше половины обычной высоты.

Красно-чёрные деревья

Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

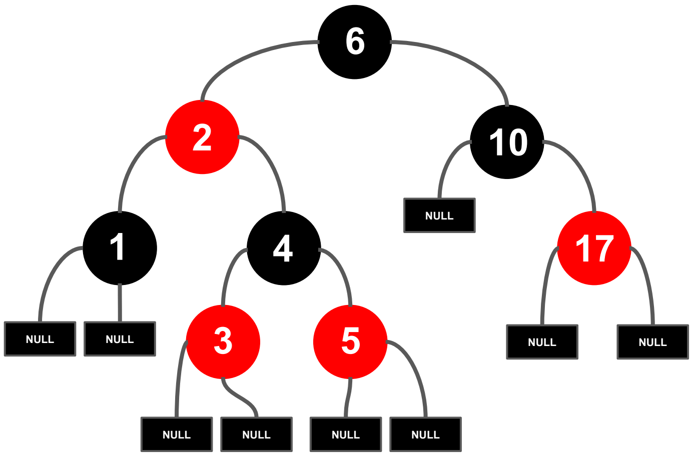
Тогда $size(x) \geq 2^{bh(x)} - 1$



Пусть теперь h - высота всего дерева. Утверждается, что $bh(root) \geq \frac{h}{2}$
Действительно: т.к. не бывает двух красных вершин подряд, то **чёрная** высота уж точно будет не меньше половины обычной высоты.

Тогда: $size(root) \geq 2^{bh(root)} - 1 \geq 2^{\frac{h}{2}} - 1$

Красно-чёрные деревья



Утверждение: для высоты h красно-чёрного дерева с n (реальными, не фиктивными) вершинами справедливо: $h \leq 2 * log_2(n+1)$

Доказательство: введем обозначения, пусть x - произвольная вершина в кч дереве; bh(x) - **чёрная** глубина вершины x, т.е. количество **чёрных** вершин на любом пути от x до листа, без учета x.

Тогда $size(x) \geq 2^{bh(x)} - 1$

Пусть теперь h - высота всего дерева. Утверждается, что $bh(root) \geq \frac{h}{2}$
Действительно: т.к. не бывает двух красных вершин подряд, то **чёрная** высота уж точно будет не меньше половины обычной высоты.

Тогда: $size(root) \geq 2^{bh(root)} - 1 \geq 2^{\frac{h}{2}} - 1 \Rightarrow n + 1 \geq 2^{\frac{h}{2}} \Rightarrow h \leq 2 * log_2(n+1)$ $\square$

АВЛ-деревья

Операции:

```
1.  find(value)     -> O(logN)
2.  select(i)       -> O(logN)
3.  min/max         -> O(logN)
4.  pred/succ(ptr)  -> O(logN)
5.  rank(value)     -> O(logN)
6.  вывод в пор.    -> O(N)
    возрастания
---------------------------------

7.  insert(value)   -> O(???)
8.  remove(value)   -> O(???)
```

В АВЛ-дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

Красно-чёрные деревья

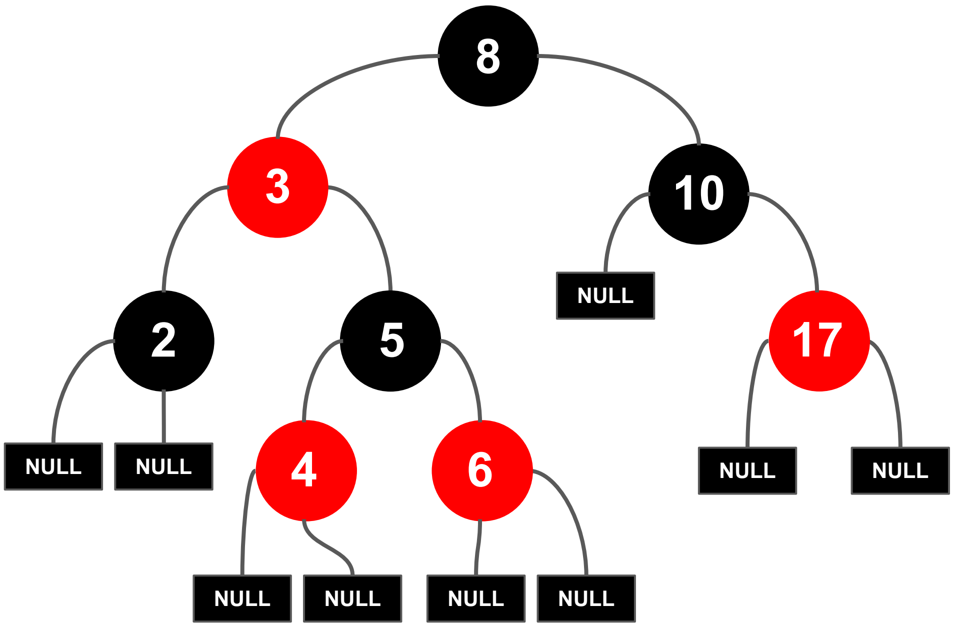
Операции:

```
1.  find(value)     -> O(logN)
2.  select(i)       -> O(logN)
3.  min/max         -> O(logN)
4.  pred/succ(ptr)  -> O(logN)
5.  rank(value)     -> O(logN)
6.  вывод в пор.    -> O(N)
    возрастания
----------------------------------

7.  insert(value)   -> O(???)
8.  remove(value)   -> O(???)
```

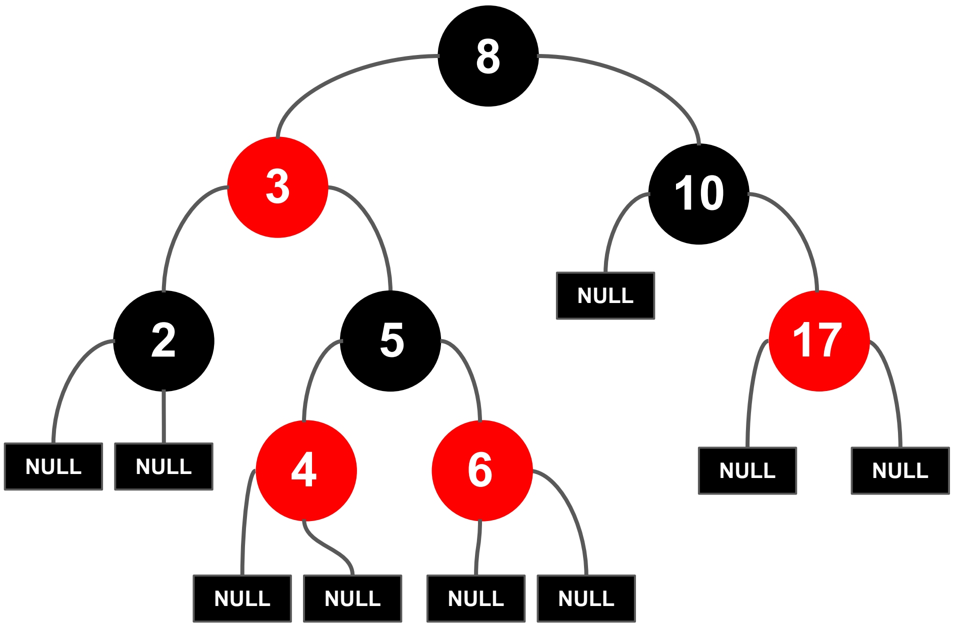
В красно-**чёрном** дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

Красно-чёрные деревья: вставка

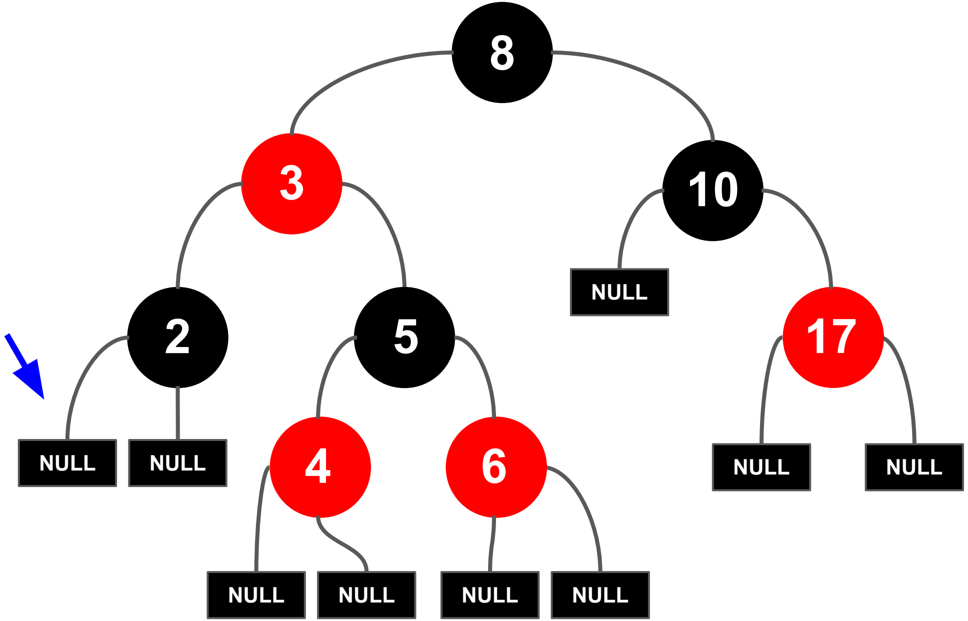


Красно-чёрные деревья: вставка




```
insert(1)
```

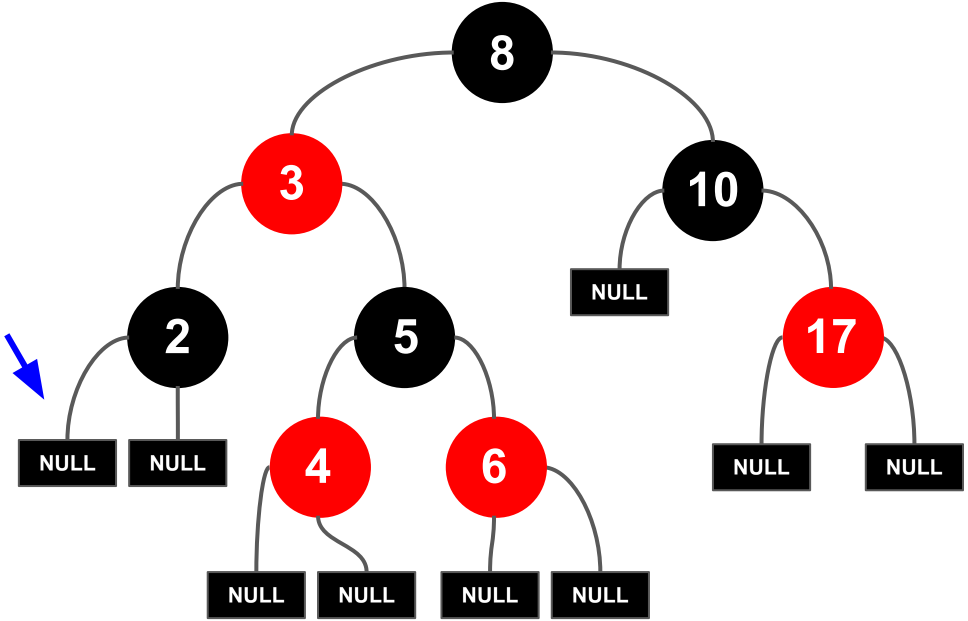
Красно-чёрные деревья: вставка




```
insert(1)
```

Сначала ищем позицию для вставки - как всегда.

Красно-чёрные деревья: вставка

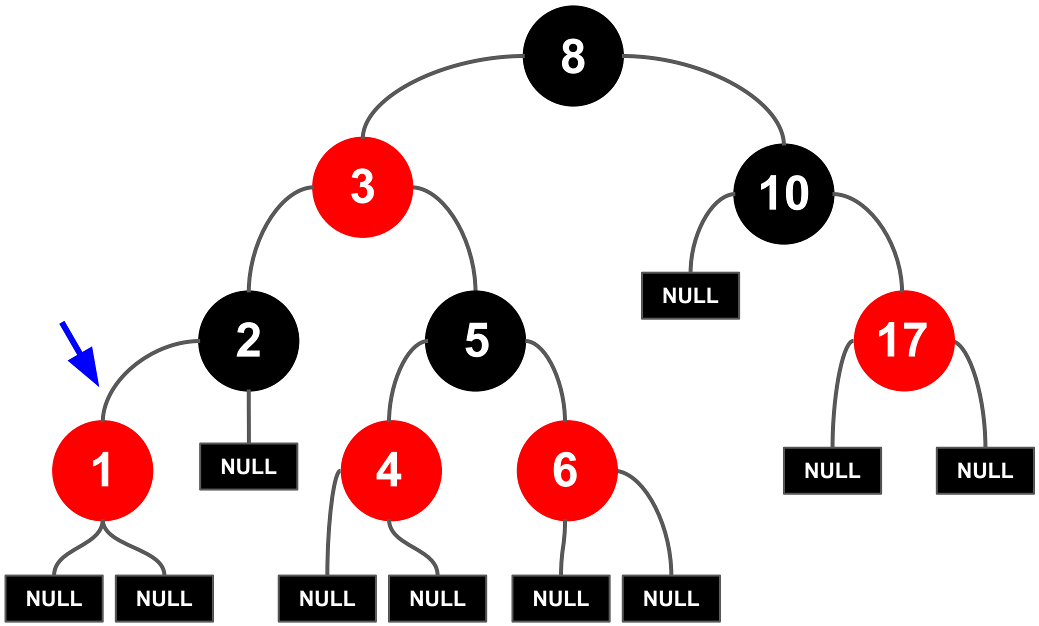


`insert(1)`

Сначала ищем позицию для вставки - как всегда.

Всегда будем пытаться вставить вершину красной.

Красно-чёрные деревья: вставка



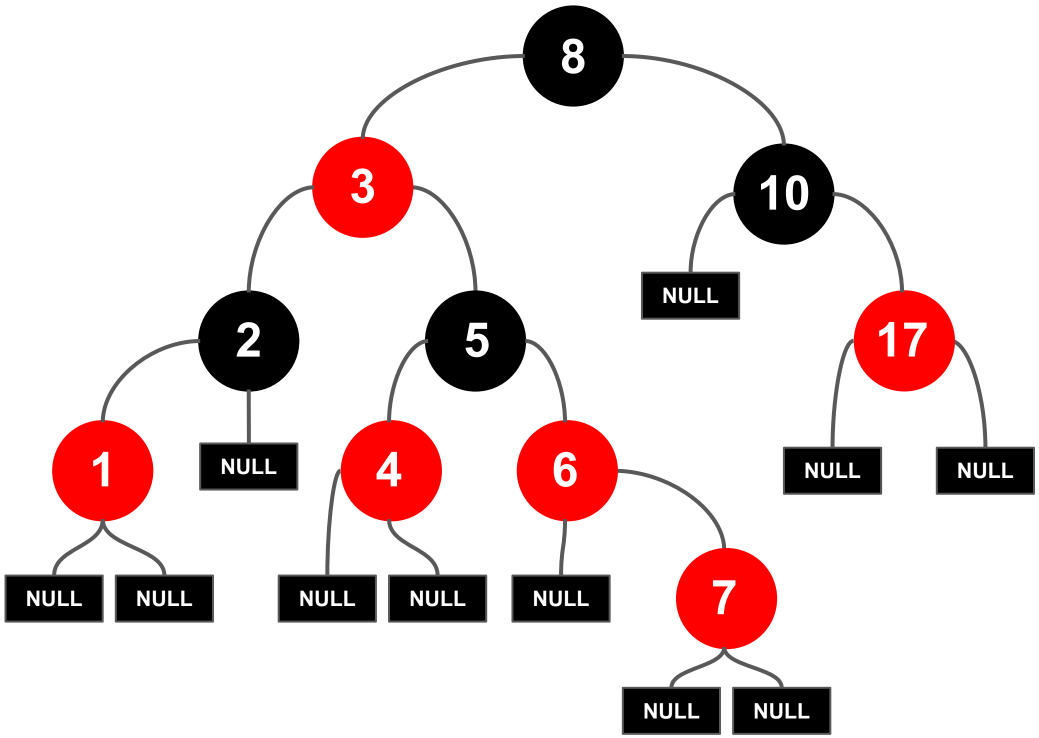
`insert(1)`

Сначала ищем позицию для вставки - как всегда.

Всегда будем пытаться вставить вершину красной.

Если его предок **чёрный** - нам повезло! Больше ничего делать не нужно.

Красно-чёрные деревья: вставка



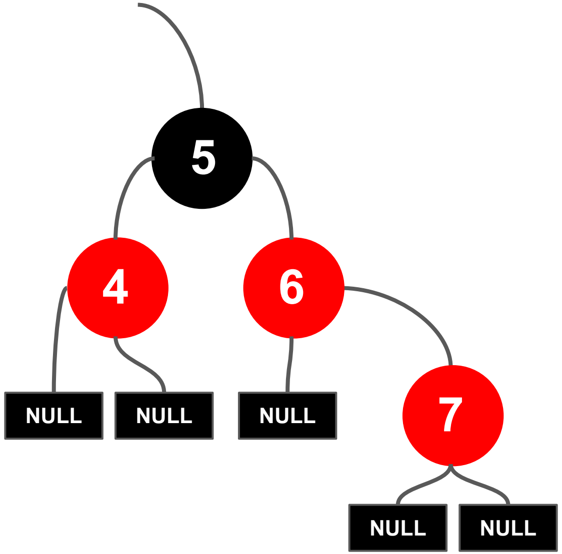

```
insert(7)
```

Сначала ищем позицию для вставки - как всегда.

Всегда будем пытаться вставить вершину красной.

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

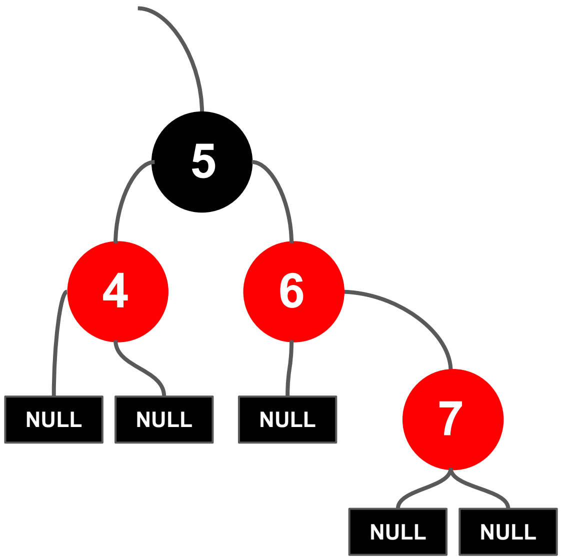
Красно-чёрные деревья: вставка




```
insert(7)
```

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Красно-чёрные деревья: вставка

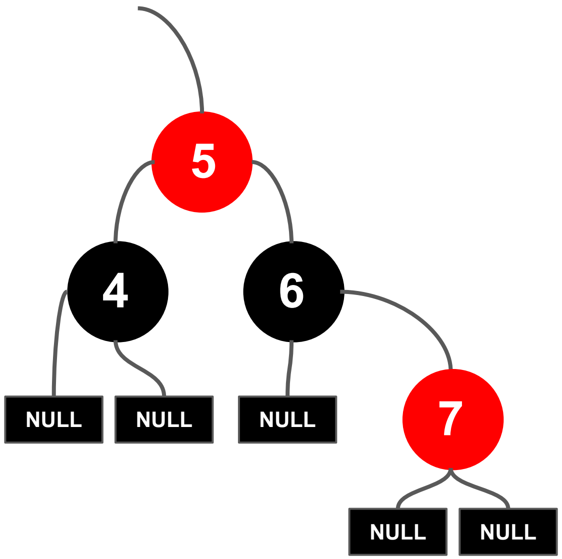


`insert(7)`

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Красно-чёрные деревья: вставка



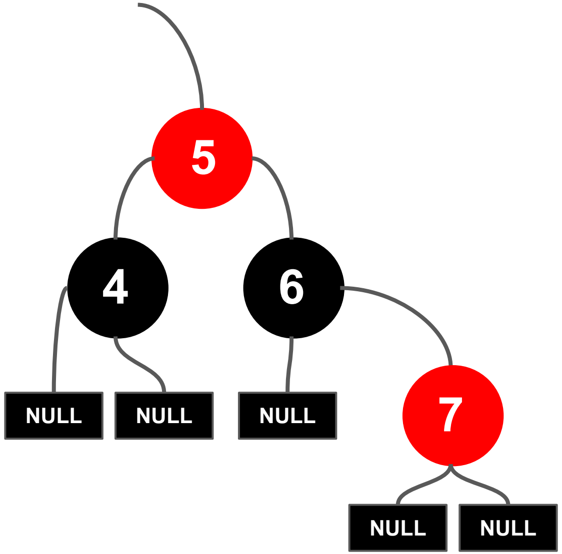
`insert(7)`

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Если красный - давайте перекрасим деда, дядю и отца.

Красно-чёрные деревья: вставка



`insert(7)`

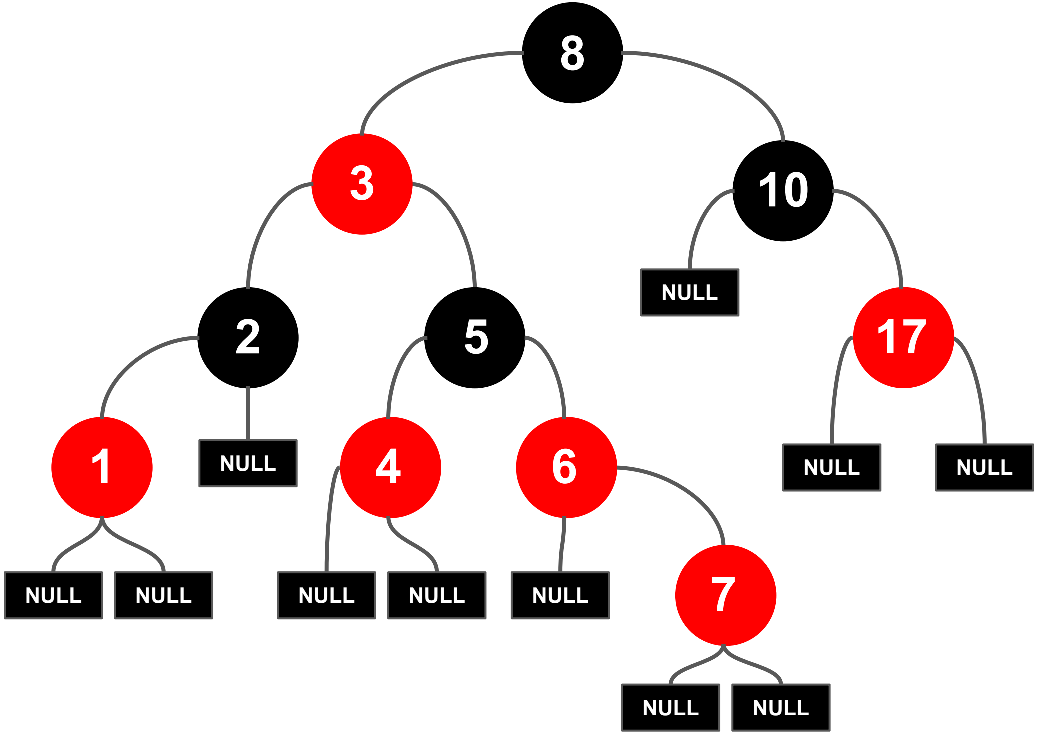
А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Если красный - давайте перекрасим деда, дядю и отца.

В поддереве больше нарушений нет.

Красно-чёрные деревья: вставка



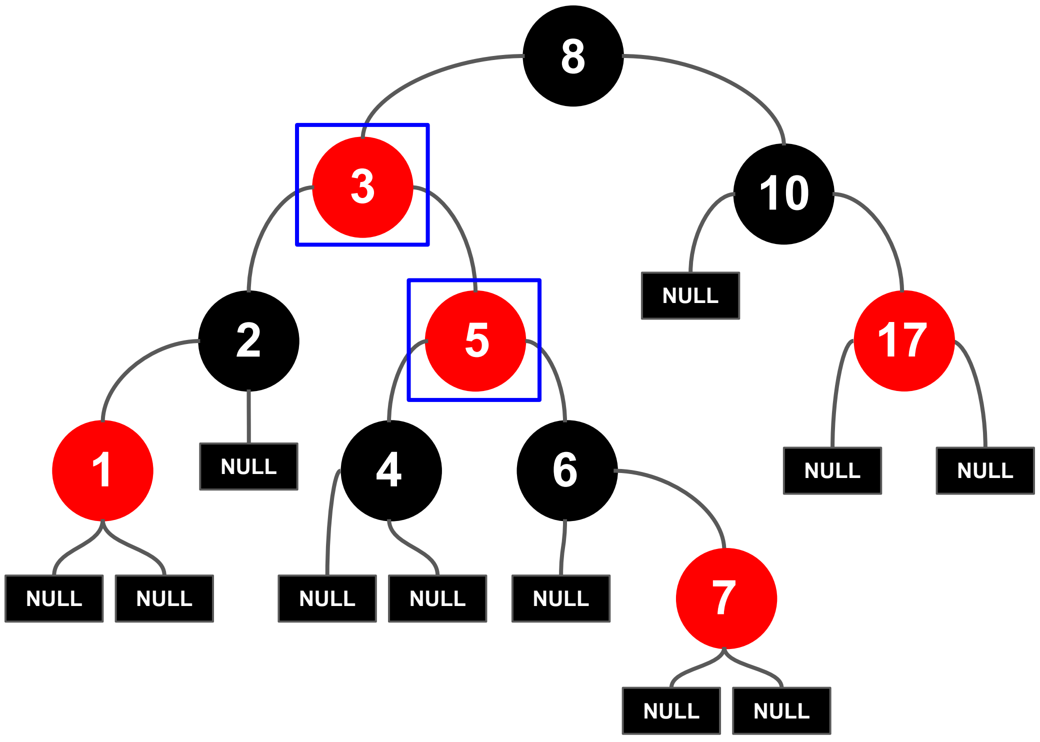
`insert(7)`

Сначала ищем позицию для вставки - как всегда.

Всегда будем пытаться вставить вершину красной.

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Красно-чёрные деревья: вставка



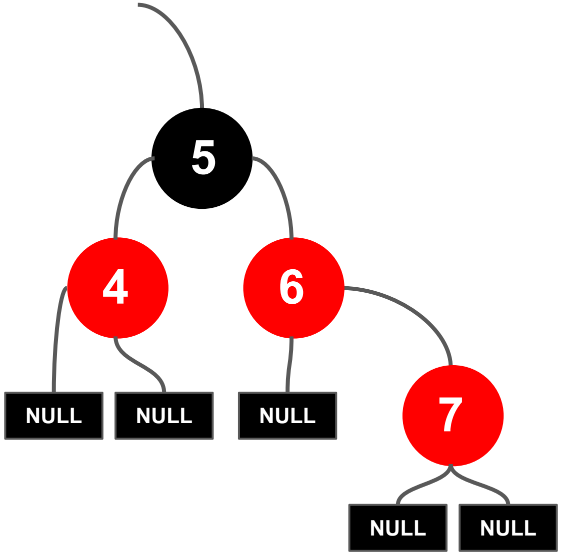

```
insert(7)
```

Сначала ищем позицию для вставки - как всегда.

Всегда будем пытаться вставить вершину красной.

Теперь проблема с цветом деда => рекурсивно исправляем проблему там.

Красно-чёрные деревья: вставка

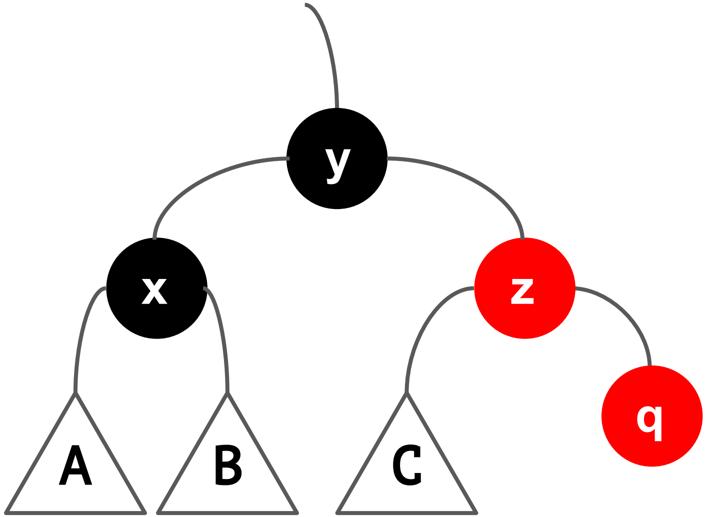



```
insert(7)
```

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Красно-чёрные деревья: вставка

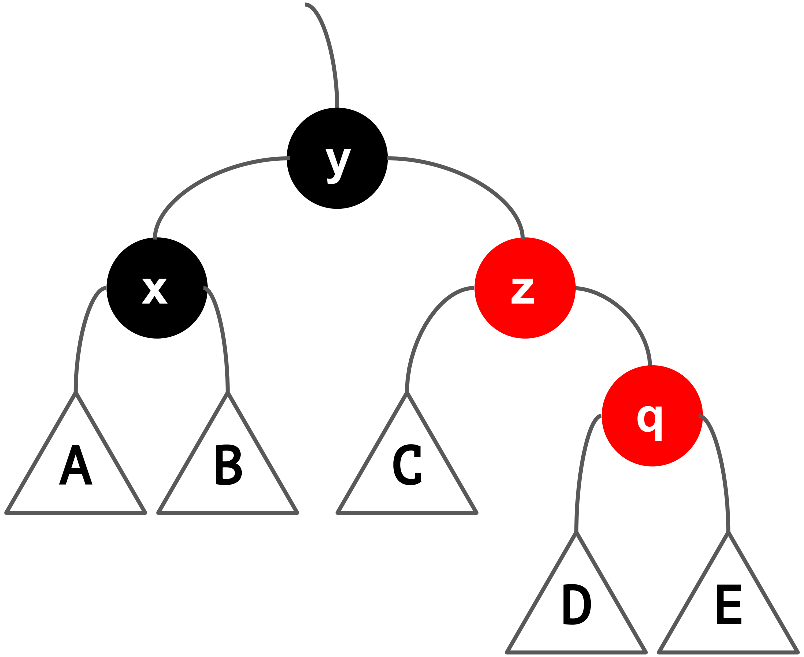


`insert(q)`

А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Красно-чёрные деревья: вставка

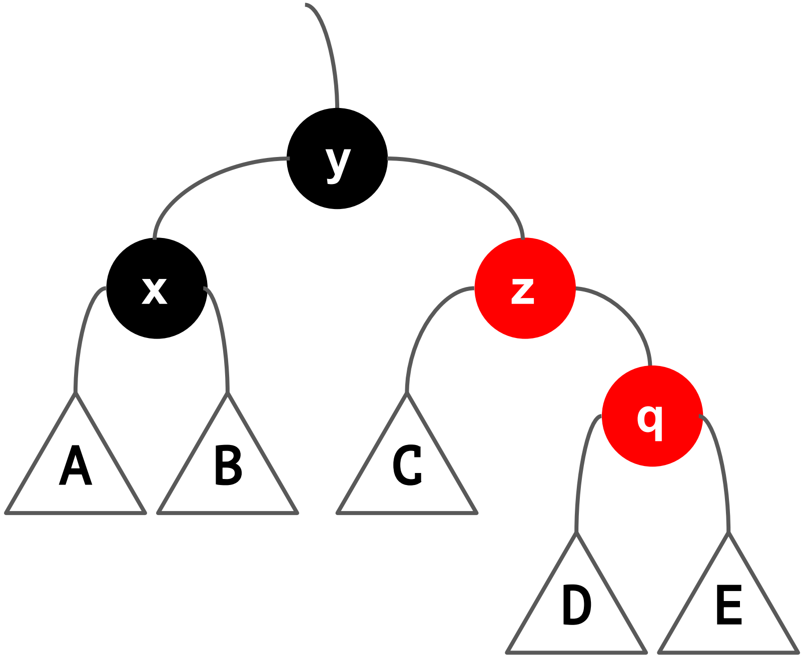


А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Общий случай, когда дядя (x) добавляемой вершины - **чёрный**.

Красно-чёрные деревья: вставка



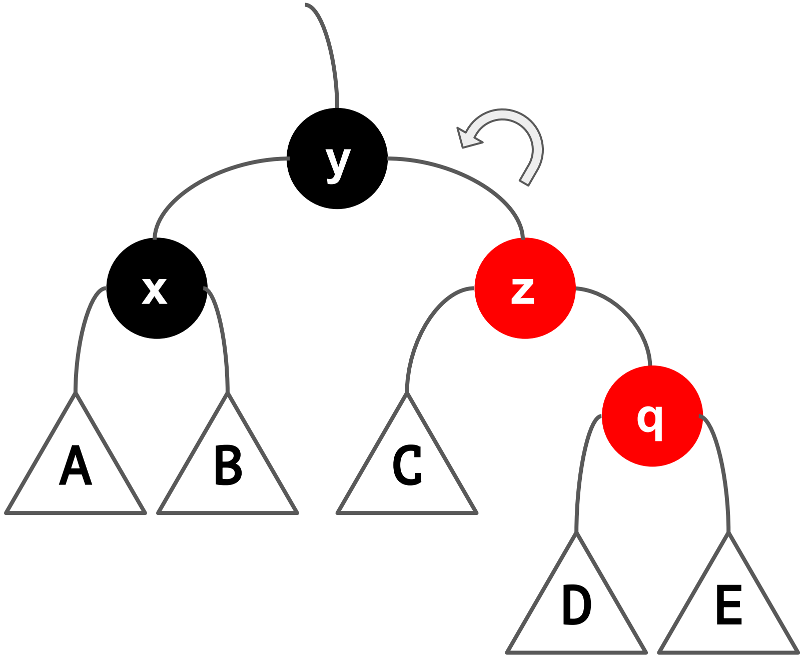
А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

Общий случай, когда дядя (x) добавляемой вершины - **чёрный**.

Обычная перекраска здесь не поможет, здесь нужен поворот.

Красно-чёрные деревья: вставка



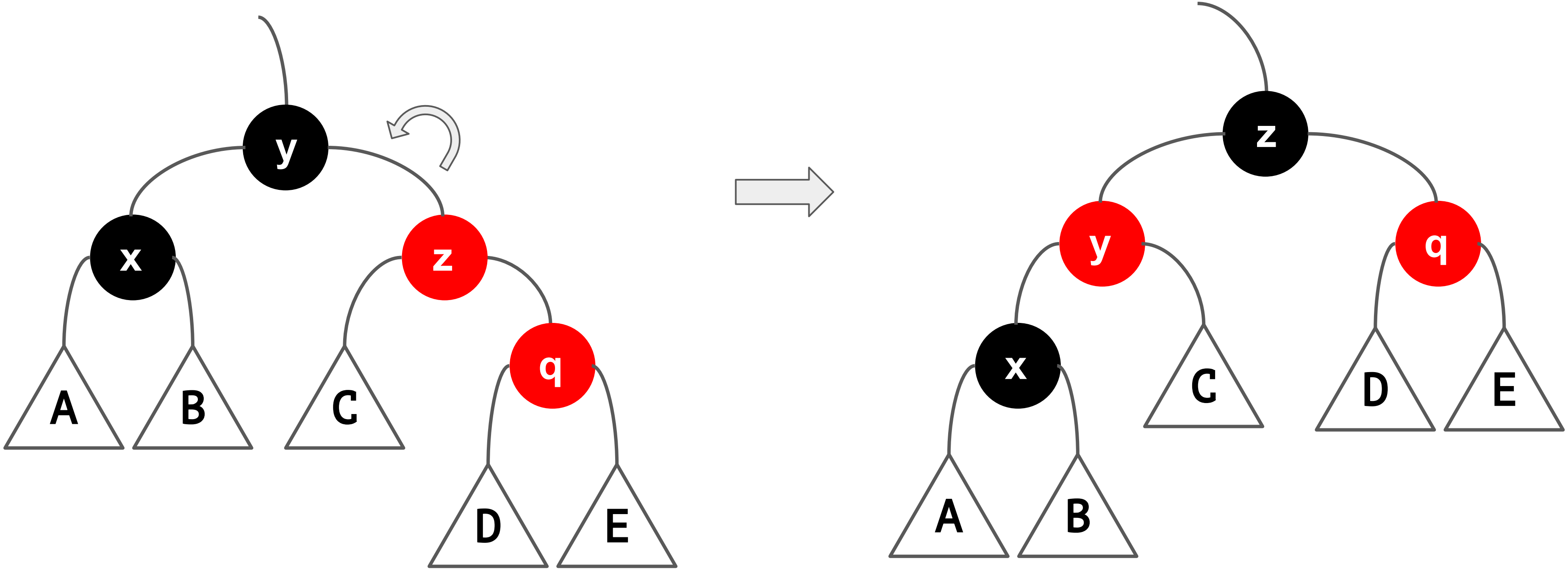
А вот если родитель тоже красный, то у нас нарушение определения к/ч дерева.

Тогда смотрим на **дядю** новой вершины. Он красный или **чёрный**?

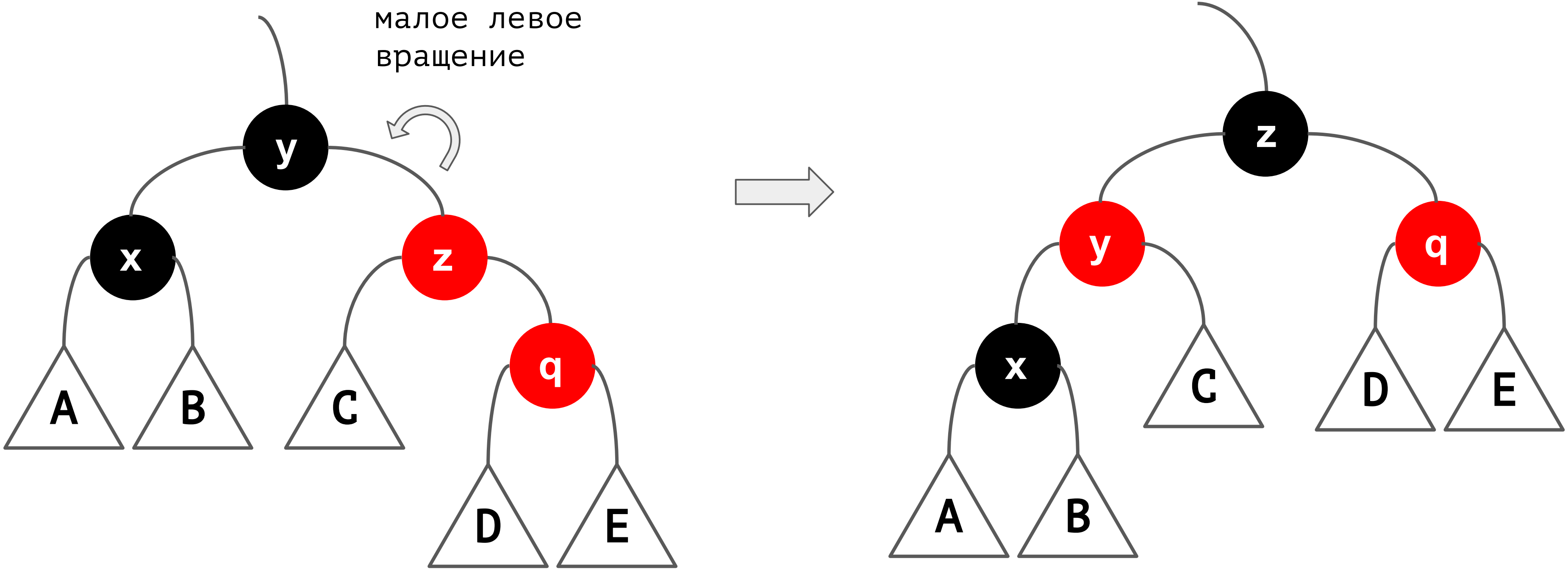
Общий случай, когда дядя (x) добавляемой вершины - **чёрный**.

Обычная перекраска здесь не поможет, здесь нужен поворот.

Красно-чёрные деревья: вставка



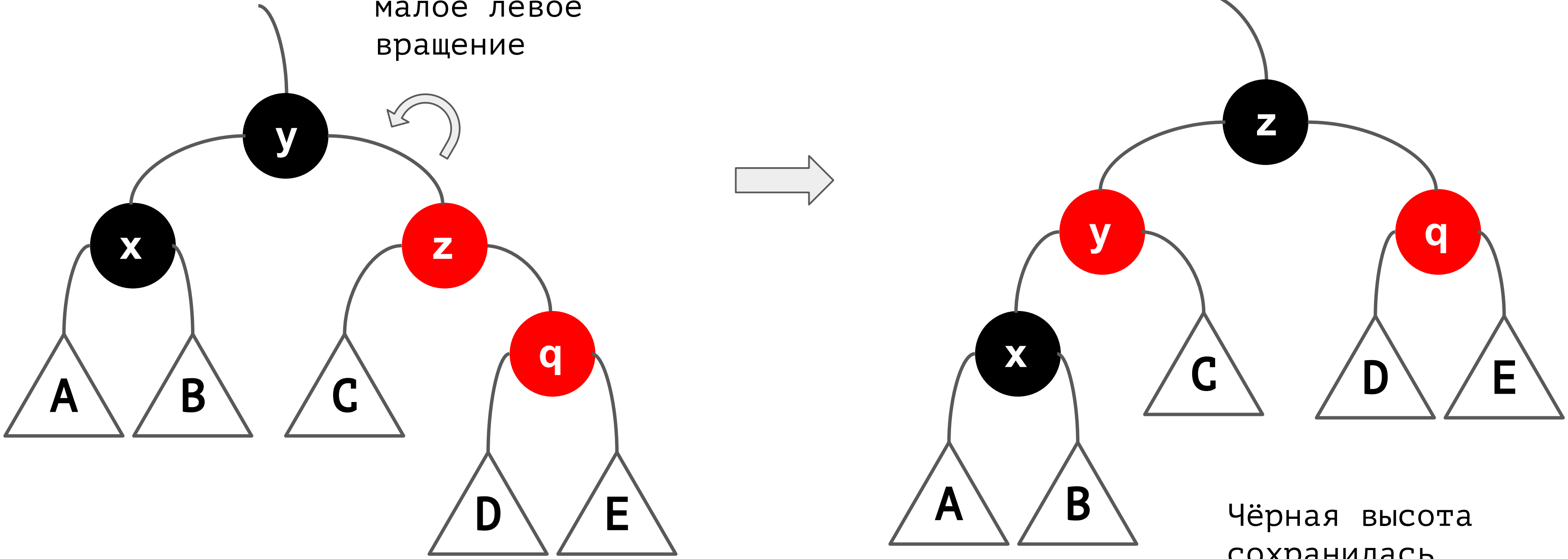
Красно-чёрные деревья: вставка



Красно-чёрные деревья: вставка

И выше больше ходить не надо, ничего не нарушилось!

малое левое вращение



Чёрная высота сохранилась (легко проверить)

Красно-чёрные деревья: вставка

Таким образом, вставка элемента в красно-чёрное дерево либо:

1. отрабатывает сразу после нахождения места для вставки (если повезло и предок **чёрный**)

Красно-чёрные деревья: вставка

Таким образом, вставка элемента в красно-чёрное дерево либо:

1. отрабатывает сразу после нахождения места для вставки (если повезло и предок **чёрный**)

2. или начинает перекрашивать рекурсивно по обратному ходу (не больше, чем logN)

Красно-чёрные деревья: вставка

Таким образом, вставка элемента в красно-чёрное дерево либо:

1. отрабатывает сразу после нахождения места для вставки (если повезло и предок **чёрный**)

2. или начинает перекрашивать рекурсивно по обратному ходу (не больше, чем logN)

3. может сделать до двух вращений, но потом сразу прервется.

Красно-чёрные деревья

Операции:

```
1.  find(value)      -> O(logN)
2.  select(i)        -> O(logN)
3.  min/max          -> O(logN)
4.  pred/succ(ptr)   -> O(logN)
5.  rank(value)      -> O(logN)
6.  вывод в пор.     -> O(N)
    возрастания
---------------------------------

7.  insert(value)    -> O(???)
8.  remove(value)    -> O(???)
```

В красно-**чёрном** дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

Красно-чёрные деревья

Операции:

1. find(value) -> O(logN)
2. select(i) -> O(logN)
3. min/max -> O(logN)
4. pred/succ(ptr) -> O(logN)
5. rank(value) -> O(logN)
6. вывод в пор. возрастания -> O(N)

---

7. insert(value) -> O(logN)
8. remove(value) -> O(???)

В красно-**чёрном** дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

Красно-чёрные деревья: удаление

Красно-чёрные деревья: удаление

Идея: как обычно у нас 3 варианта вершины для удаления (без реальных детей, с 1 сыном, с 2 сыновьями)

Но теперь еще рассматриваем варианты цветов предков и красим или поворачиваем.

Красно-чёрные деревья: удаление

Идея: как обычно у нас 3 варианта вершины для удаления
(без реальных детей, с 1 сыном, с 2 сыновьями)

Но теперь еще рассматриваем варианты цветов предков
и красим или поворачиваем.

Полный анализ всех случаев: https://youtu.be/T70nn4EyTrs?t=2695
(на экзамене разбора удаления не будет)

Красно-чёрные деревья: удаление

Идея: как обычно у нас 3 варианта вершины для удаления (без реальных детей, с 1 сыном, с 2 сыновьями)

Но теперь еще рассматриваем варианты цветов предков и красим или поворачиваем.

Полный анализ всех случаев: https://youtu.be/T70nn4EyTrs?t=2695 (на экзамене разбора удаления не будет)

Важно: асимптотика удаления, как обычно, O(logN). Но есть гарантия, что каждое удаление требует не больше 3 поворотов.

Красно-чёрные деревья: удаление

Идея: как обычно у нас 3 варианта вершины для удаления (без реальных детей, с 1 сыном, с 2 сыновьями)

Но теперь еще рассматриваем варианты цветов предков и красим или поворачиваем.

Полный анализ всех случаев: https://youtu.be/T70nn4EyTrs?t=2695
(на экзамене разбора удаления не будет)

Важно: асимптотика удаления, как обычно, O(logN). Но есть гарантия, что каждое удаление требует не больше 3 поворотов.

В АВЛ дереве именно для удаления такой гарантии нет, можем поворачивать вплоть для O(logN) раз.

Для вставки АВЛ тоже гарантирует ограниченное количество вращений.

Красно-чёрные деревья

Операции:

```
1.  find(value)      -> O(logN)
2.  select(i)        -> O(logN)
3.  min/max          -> O(logN)
4.  pred/succ(ptr)   -> O(logN)
5.  rank(value)      -> O(logN)
6.  вывод в пор.     -> O(N)
    возрастания
---------------------------------
7.  insert(value)    -> O(logN)
8.  remove(value)    -> O(???)
```

В красно-**чёрном** дереве действительно высота всегда порядка logN

Но как поддержка инвариантов повлияет на добавление и удаление?

Красно-чёрные деревья

Операции:

```
1. find(value)      -> O(logN)
2. select(i)        -> O(logN)
3. min/max          -> O(logN)
4. pred/succ(ptr)   -> O(logN)
5. rank(value)      -> O(logN)
6. вывод в пор.     -> O(N)
   возрастания
---------------------------------

7. insert(value)    -> O(logN)
8. remove(value)    -> O(logN)
```

В красно-**чёрном** дереве действительно высота всегда порядка logN

И все операции продолжают работать за логарифм!

АВЛ деревья VS Красно-чёрные деревья

АВЛ:

+ операции за O(logN)
+ глубина ~1.5*logN
+ добавление требует O(1) поворотов

– удаление требует O(logN) поворотов

АВЛ деревья VS Красно-чёрные деревья

АВЛ:

+ операции за O(logN)
+ глубина ~1.5*logN
+ добавление требует O(1) поворотов

– удаление требует O(logN) поворотов

Красно-**чёрные**:

+ операции за O(logN)
+ добавление требует O(1) поворотов
+ удаление требует O(1) поворотов

- глубина ⩽ 2*logN

Takeaways

- Сбалансированные деревья, как структура данных для поиска (и добавления/удаления, иначе бы подошёл массив)

Takeaways

- Сбалансированные деревья, как структура данных для поиска (и добавления/удаления, иначе бы подошёл массив)
- АВЛ и Красно-чёрные деревья, как немного разный подход для решения одной и той же задачи
- АВЛ - меньшая глубина, К/Ч - более быстрое удаление.