**AA Дерево**

Представляет собой сбалансированное бинарное дерево с более простым алгоритмом, представленным Арне Андерсоном в 1993г. Бинарные сбалансированные деревья были введены еще Рудольфом Байером в 1971 году, однако алгоритм балансировки в прочих алгоритмах построения бинарного дерева остается довольно громоздким.

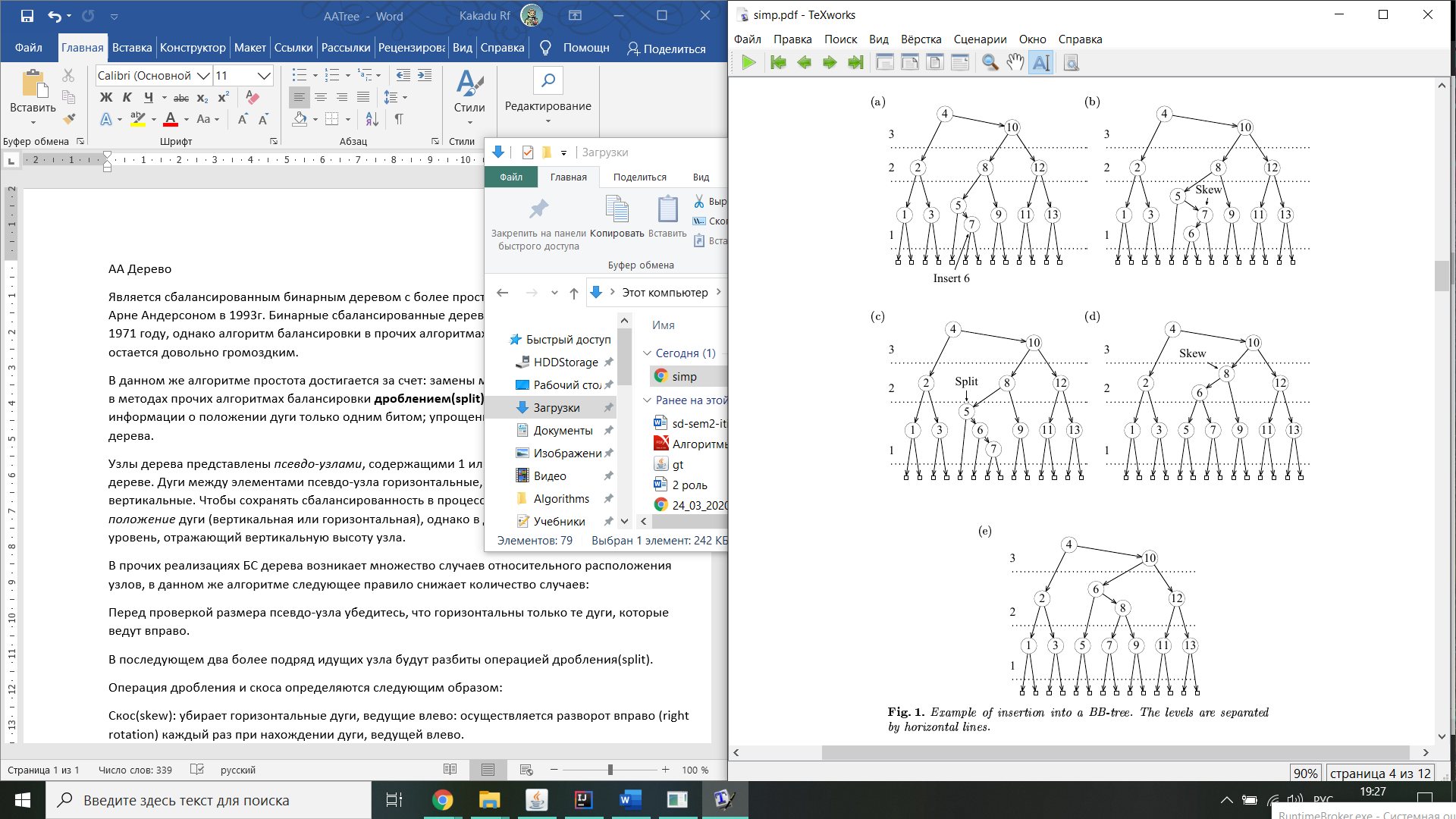
В данном же алгоритме простота достигается за счет:

* замены множественного количество случаев в методах прочих алгоритмах балансировки дроблением(split) и перекосом(skew);
* представлением информации о положении дуги только одним битом;
* упрощения операции удаления узла из дерева.

Узлы дерева представлены *псевдо-узлами*, содержащими 1 или 2 узла, аналогичные узлам в 2-3 дереве. Дуги между элементами псевдо-узла горизонтальные, дуги между псевдо-узлами – вертикальные. Чтобы сохранять сбалансированность в процессе обновления дерева используется *положение* дуги (вертикальная или горизонтальная), однако в данной реализации используется уровень, отражающий вертикальную высоту узла.

В прочих реализациях БС дерева возникает множество случаев относительного расположения узлов, в данном же алгоритме следующее правило значительно снижает количество случаев:

*Убедитесь, что горизонтальна только та дуга, что ведёт вправо (1).*

В последующем более двух подряд идущих узла будут разбиты операцией дробления(split).

Операция дробления и перекоса определяются следующим образом:

**Перекос** (skew): убирает горизонтальные дуги, ведущие влево: осуществляется разворот вправо (right rotation) каждый раз при нахождении дуги, ведущей влево.

**Дробление** (split): при размере псевдо-узла, превышающем 2, увеличивается уровень каждого второго узла. Осуществляется путем разворота влево (left rotation)

Операция добавления:

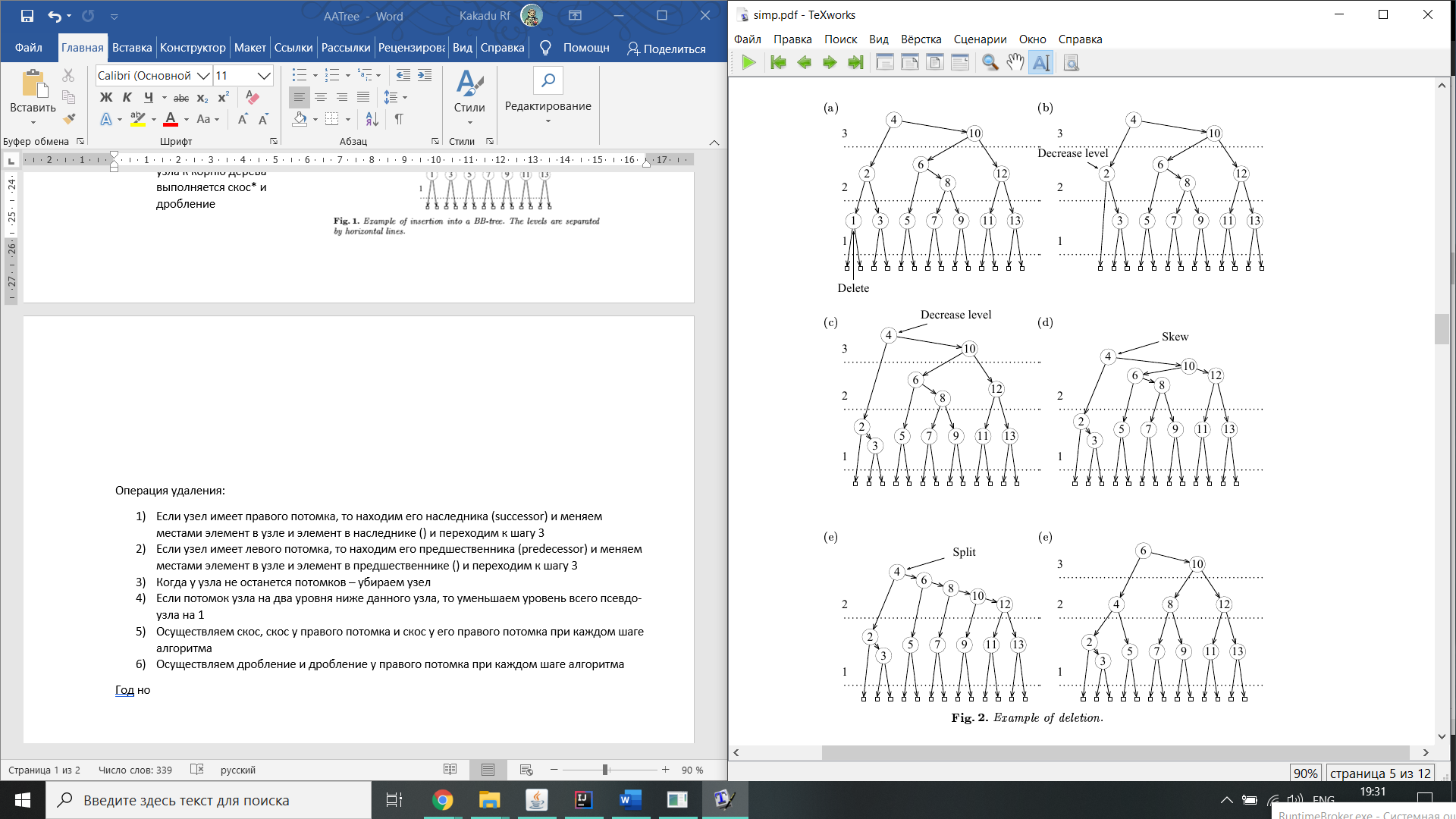
1. На уровне 1 добавляется узел
2. На всем пути от нового узла к корню дерева выполняется перекос и дробление, гарантирующие соблюдения правила (1)

**Наследник** (successor) – узел, полученный при одном шаге вправо и всех последующих шагах влево.

**Предшественник** (predecessor)– узел, полученный при одном шаге влево и всех последующих шагах вправо.

(При нахождении наследника (successor) или предшественника (predecessor) узла, мы находим ближайший по значению элемент к элементу в данном узле, поэтому операции замены данного элемента на этот элемент наследника (successor) или предшественника (predecessor) эквивалентны в контексте балансировке дерева, а значит и взаимозаменяемы при отсутствии правого или левого ребра)

Операция удаления:

1. Если узел имеет правого потомка, то находим его наследника (successor) и приравниваем элемент в узле к элементу в наследника (successor) и переходим к шагу 3, продолжая поиск по дереву пока не найдем элемент наследника (successor)
2. Если узел имеет левого потомка, то находим его предшественника (predecessor) и приравниваем элемент в узле к элементу в предшественнике (predecessor) и переходим к шагу 3, продолжая поиск по дереву пока не найдем предшественника (predecessor)
3. Когда у узла не останется потомков – удаляем узел
4. Если потомок узла на два уровня ниже данного узла, то уменьшаем уровень всего псевдо-узла на 1
5. Осуществляем перекос, перекос у правого потомка и перекос у его правого потомка при каждом шаге алгоритма, поскольку при уменьшении уровня псевдо-узла у соответствующих узлов могут получится левые соседи одного уровня
6. Осуществляем дробление и дробление у правого потомка при каждом шаге алгоритма, поскольку при уменьшении уровня псевдо-узла размер полученного псевдо-узла может превышать 2

**Задача:**

**findTwoBordering(T x):**

Для того, чтобы протестировать нашу структуру данных была поставлена следующая задача:

Даны n элементов и далее на вход принимается элемент x. Требуется найти два элемента, пограничных данному элементу (a, b: a <x, x <b), причем a и b – ближайшие к данному элементу значения среди всех n чисел.

Для решения данной задачу мы построим дерево, содержащее все n элементов, поскольку число неизвестно заранее, далее, поскольку мы выяснили ранее, что ближайшие к элементу числа – его предшественник(predecessor) и наследник(successor) (a и b соответственно), мы можем вычислить эти числа (Обратите внимание: одного из элементов пары возвращаемых значений может не быть)

Возможны 4 варианта расположения x:

1. x будет иметь и правого и левого потомка, тогда находим его наследника () и предшественника (predecessor).
2. x не будет иметь правого или левого потомка, значит находим соответственно его предшественника (predecessor) или наследника (successor) и узел, предшественником (predecessor) или наследником (successor) для которого он является
3. x не будет иметь и правого и левого потомка, тогда узел, наследником (successor) которого он является и узел, предшественником (predecessor) для которого он является.
4. x не будет представлен в дереве, тогда находим ближайший к нему элемент, проходя по дереву и, если возможно, находим узел, предшественником (predecessor) или наследником (successor) для которого он является (в зависимости от найденного элемента)

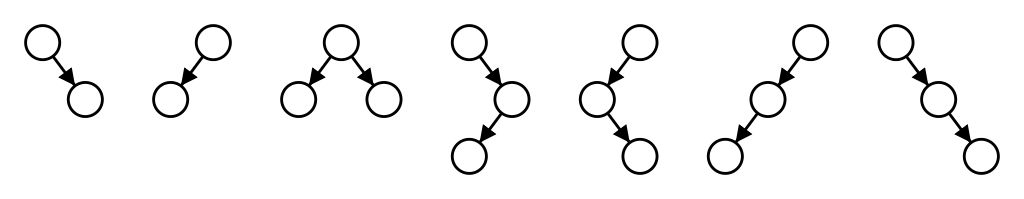
Временная сложность данного алгоритма - O(log(n)), т.к. successor – O(h) = O(log(n)), проход по пути от корня до вершины – O(h) = O(log(n))

**Сохранение структуры красно-черного дерева:**

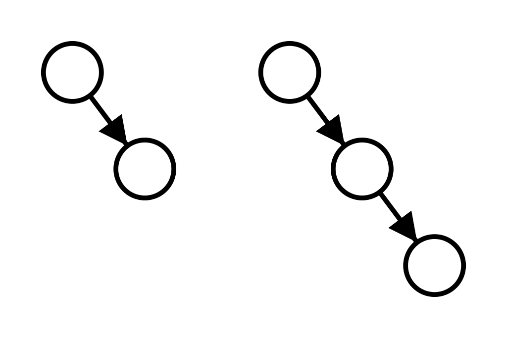
Поскольку на АА дерево наложены дополнительные ограничения (1), то в красно-черном дереве, аналогичном нашему АА дереву красные вершины будут только правыми наследниками черных вершин того же уровня, таким образом будут выполнятся все свойства красно-черного дерева:

1. Из сказанного выше, узел может быть красным и черным и имеет 2 потомков (при отсутствии оных – ссылок на null)
2. Корень черный, т.к. не является наследником другого узла
3. Все листья черные, т.к. по факту имеют уровень 0 и не является наследником другого узла того же уровня
4. Оба потомка красного узла – черные, т.к. либо являются листьями, либо являются первыми узлами псевдо-узла, а значит - не является наследником другого узла того же уровня
5. Любой простой путь от узла-предка до листового узла-потомка содержит одинаковое число чёрных узлов, т.к. каждый псевдо-узел имеет ровно 1 черный узел, а путь от любого псевдо-узла до любого листа содержит (уровень узла) вертикальных дуг, что соответствует количеству черных вершин.

Структура красно-черного дерева сохраняется при балансировке, т.к. 7 различных вариантов расположения вершин сводятся к 2 (согласно правилу (1)):

Вместо

Имеем



Далее представлено доказательство временной сложности красно-черного дерева:

**В красно-черном дереве с черной высотой hb количество внутренних вершин не менее − 1.**

**[**

Докажем по индукции по обычной высоте h(x), что поддерево любого узла x с черной высотой hb(x) содержит не менее  − 1 внутренних узлов. Здесь h(x) — кратчайшее расстояние от вершины x до какого-то из листьев, hb(x) - число чёрных вершин на пути из x в лист, аналогично уровню в АА дереве.

База индукции:

Если высота узла x равна 1, то x — это лист, hb(x)=1,  – 1 = 0

Переход:

Так как любая внутренняя вершина (вершина, у которой высота положительна) имеет двух потомков, то применим предположение индукции к ним — их высоты на единицу меньше высоты x. Тогда черные высоты детей могут быть  hb(x) или hb(x)− 1 — если потомок красный или черный соответственно.

Тогда по предположению индукции в каждом из поддеревьев не менее  вершин. Тогда всего в поддереве не менее 2⋅(  −1) + 1=,  −1 вершин (+1— мы учли еще саму вершину x).

Переход доказан. Теперь, если мы рассмотрим корень всего дерева в качестве x, то получится, что всего вершин в дереве не менее   −1.

Следовательно, утверждение верно и для всего дерева.

**]**

**Красно-чёрное дерево с N ключами имеет высоту h=O(logN).**

**[**

Рассмотрим красно-чёрное дерево с высотой h. Так как у красной вершины чёрные дети (по свойству 3) количество красных вершин не больше h/2. Тогда чёрных вершин не меньше, чем h/2−1

По доказанной лемме, для количества внутренних вершин в дереве N выполняется неравенство:

N⩾−1

Прологарифмировав неравенство, имеем:

log(N+1) ⩾ h/2

2log(N+1) ⩾ h/2

h⩽2log(N+1)

**]**

* Поскольку АА дерево является разновидностью красно-черного дерева, то операции нахождения элемента, добавления и удаления выполняются за О(log(n))
* Операции нахождения наследника (successor) или предшественника (predecessor) за O(log(n))
* Перекос (skew), дробление(split) и уменьшение уровня (decrease level) выполняются за O(C)

(Временная сложность указана в худшем случае)

Ниже представлены графики зависимости времени вставки, удаления и поиска от входных значений на случайных данных

Как видно из графиков, при увеличении входных данных время выполнения операций увеличивается с не более чем логарифмической скоростью

**Выводы:**

Плюсы:

* АА дерево - одно из самых простых в реализации
* Также в силу простоты реализации простое в освоении

Минусы:

* В данной реализации – больший расход памяти из-за хранения уровня каждого узла (байт на каждый узел)
* Большее количество разворотов(rotation), чем в красно-черном дереве.

**Источники:**

1. **A. Andersson. Balanced search trees made simple.**  
   In *Proc. Workshop on Algorithms and Data Structures*, pages 60--71. Springer Verlag, 1993.

<http://user.it.uu.se/~arnea/abs/simp.html>

1. **A. Andersson and S. Nilsson. Efficient implementation of suffix trees.**  
   *Software---Practice and Experience*, 25(2):129-141, 1995.

<http://user.it.uu.se/~arnea/abs/searchproc.html>

1. **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ =** *Introduction to algorithms. — 2-е изд. — М.:*[*Издательский дом «Вильямс»*](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1)*, 2011.*

[ISBN 978-5-8459-0857-5](https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F:%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%D0%B8_%D0%BA%D0%BD%D0%B8%D0%B3/9785845908575)