электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51038.

УДК 004.651.53

Сравнение АВЛ и Красно-черного дерева

***Опрышко А. В.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Системы обработки информации и управления»

Научный руководитель: Гапанюк Ю.Е., к.т.н., доцент,
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана
gapyu@bmstu.ru

Дерево — одна из наиболее широко распространённых структур данных, эмулирующая древовидную структуру в виде набора связанных узлов.

Двоичные деревья представляют собой иерархическую структуру, в которой каждый узел имеет не более двух потомков. То есть двоичное дерево либо является пустым, либо состоит из данных и двух поддеревьев (каждое из которых может быть пустым). При этом каждое поддерево в свою очередь тоже является деревом. Поиск на таких структурах не дает выигрыша по сравнению с линейными структурами того же размера, так как необходимо в худшем случае выполнить обход всего дерева. Поэтому интерес представляют двоичные деревья поиска.

Двоичное дерево поиска — это двоичное дерево, для которого выполняются следующие свойства:

- Оба поддерева — левое и правое, являются двоичными деревьями поиска.
- У всех узлов левого поддерева узла X значения ключей меньше, нежели значение ключа данных самого узла X.
- У всех узлов правого поддерева узла X значения ключей данных не меньше, нежели значение ключа данных узла X.

АВЛ и Красно-черное дерево являются разновидностью самобалансирующихся двоичных деревьев поиска.

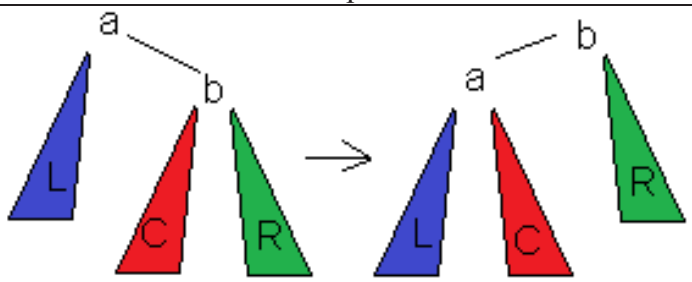
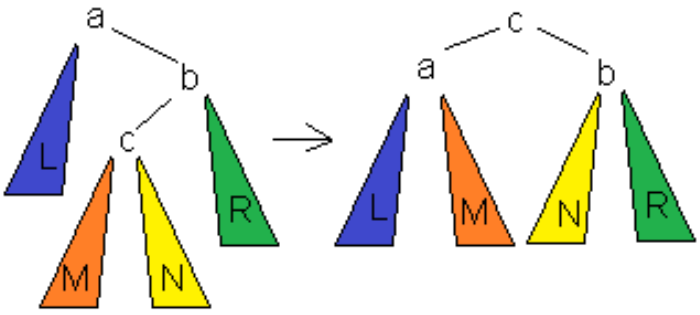
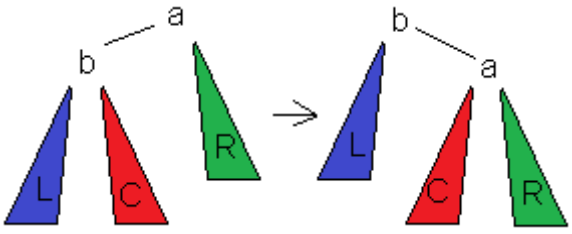
В данной работе рассмотрены алгоритмы работы АВЛ и Красно-черного дерева и сравнены показатели эффективности на основе операция: добавления, удаления и поиска.

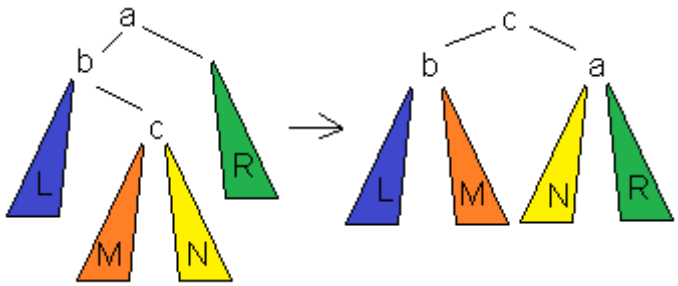
АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

Структура узла:

```
struct CNode {
        int Key;
        int Height;
        CNode* Left;
        CNode* Right;
};
```

Балансировка. Относительно АВЛ-дерева балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев равна 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится меньше либо равна 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины. В таблице представлены 4 типа вращений, применяемые при балансировке в АВЛ дереве.

Тип вращения в АВЛ дереве	Условие применения	Графическое изображение вращений в АВЛ дереве
Малое левое вращение	Разница между высотой b-поддерева и L равна 2 и высота C меньше либо равна высоте R	
Большое левое вращение	Разница между высотой b-поддерева L равна 2 и высота c-поддерева больше высоты R.	
Малое правое вращение	Разница между высотой b-поддерева и R равна 2 и высота C меньше либо равна высоте L.	

Большое правое вращение	Разница между высотой b-поддерева и R равна 2 и высота c-поддерева больше высоты L.	

Алгоритм, выполняющий балансировку, сводится к проверке условий и выполнению поворотов.

Алгоритм добавления вершины:

1. Идем по пути поиска, пока не убедимся, что ключа в дереве нет.
2. Включение новой вершины в дерево.
3. При выходе из рекурсии проверяем показатель сбалансированности.

   Если необходимо — балансируем.

Алгоритм удаления вершины:

1. Идем по пути поиска, пока не найдем удаляемый ключ.
2. Если вершина — лист, то удалим её и при выходе из рекурсии выполним балансировку. Иначе найдём самую близкую по значению вершину в поддереве наибольшей высоты (правом или левом) и переместим её на место удаляемой вершины, при этом вызвав процедуру её удаления.

Красно-чёрное дерево — это самобалансирующее двоичное дерево поиска, выполняющее следующие требования:

1. Каждый узел промаркирован красным или чёрным цветом
2. Корень и конечные узлы (листья) дерева – чёрные
3. У красного узла родительский узел – чёрный
4. Все простые пути из любого узла x до листьев содержат одинаковое количество чёрных узлов
5. Чёрный узел может иметь чёрного родителя

Структура узла:

```
struct CNode {
    int Key;
    CNode *Left;
    CNode *Light;
    CNode *Parrent;
    int Color;
```

};

Алгоритм добавления вершины:

Каждый элемент вставляется вместо листа, поэтому для выбора места вставки идём от корня до тех пор, пока указатель на следующего сына не станет NULL. Вставляем вместо него новый элемент с двумя листами потомками и красным цветом. Далее выполняем балансировку.

1. Если отец нового элемента черный, то завершаем алгоритм.
2. Если отец нового элемента красный, то достаточно рассмотреть два случая:

2.1. "Дядя" этого узла тоже красный. Перекрашиваем "отца" и "дядю" в чёрный цвет, а "деда" - в красный. При этом "дед" может нарушать свойство дерева ("прадед" может быть красным). Далее рекурсивно пытаемся восстановить свойства дерева, продвигаясь к предкам. Этот случай представлен на рисунке 1.

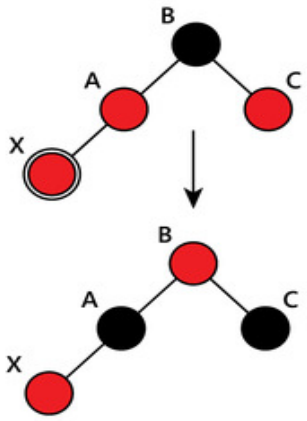


Рис. 1. Случай 1: "Дядя" – красный

2.2. "Дядя" чёрный. Если добавляемый узел X был правым потомком "отца" A, то необходимо сначала выполнить левое вращение, которое сделает "отца" A левым потомком X. Выполняем правый поворот и перекрашиваем A и B. Если "дядя" левый, то порядок действий симметричен описанному. Этот случай представлен на рисунке 2.

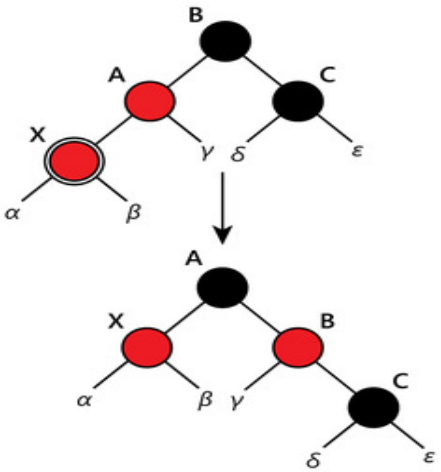


Рис. 2. Случай 2: "Дядя" – черный

Алгоритм удаление вершины

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 фиктивный лист.
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.

При удалении красной вершины свойство дерева не нарушается. Восстановление свойств красно-черного дерева потребуется только при удалении черной вершины:

1. Удаление черной вершины с потомком.

Единственным потомком черной вершины может быть только красная вершина. Иначе нарушилось бы свойство постоянства черной глубины для потомка и его соседней фиктивной вершины. Для балансировки выполним действия: в черную вершину заносим данные красной, удаляем красную.

2. Удаление черной вершины без потомков.

Для этого удалим черную вершину. Лист на месте удаленной вершины обозначим X. Путь в X имеет меньшее количество черных вершин (черную глубину), чем в

других вершинах. Будем помнить об этом и называть X дважды черным. Теперь восстановим свойства красно-черного дерева:

1) Если вершина X – корень. Оставим корень просто черным (один раз черным). Так черная глубина всего дерева уменьшится на 1. Этот случай представлен на рисунке 3.

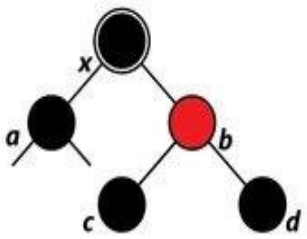


Рис. 3. Случай 1: Вершина X – корень

2) Если "брат" b вершины x – красный. Делаем вращение вокруг ребра между "отцом" a и "братом" b, тогда брат становится родителем «отца". Красим нового "деда" b в чёрный, а "отца" a – в красный цвет. Этот случай представлен на рисунке 4.

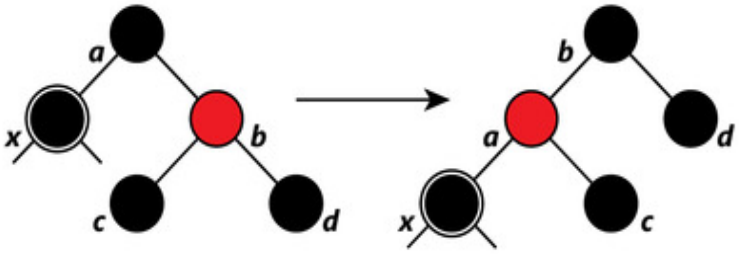


Рис. 4. Случай 2: "Брат" b вершины x – красный

3) Если "брат" b вершины x – черный, и оба дочерних узла "брата" c и d – черные. c и d могут быть листьями. Красим "брата" b в красный цвет. Так поддеревья x и b теперь недокрашены в черный. Если "отец" a был красного цвета, то красим его в черный и завершаем работу. При этом черная глубина a восстановится. Иначе, считаем "отца" a дважды черным, рассматриваем его как x. Этот случай представлен на рисунке 5.

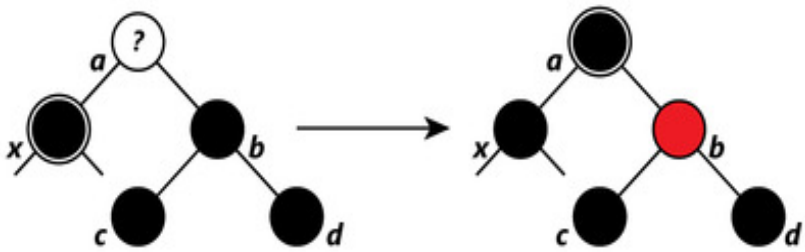


Рис. 5. Случай 3: "Брат" b вершины x – черный, оба дочерних узла "брата" c и d – черные

4) Если "брат" b вершины x – черный, левый ребенок "брата" c – красный, а правый d – черный. Делаем правое вращение c – b. Красим b в красный цвет. Красим c в черный цвет. Так у "брата" правый ребенок станет красным. Этот случай представлен на рисунке 6.

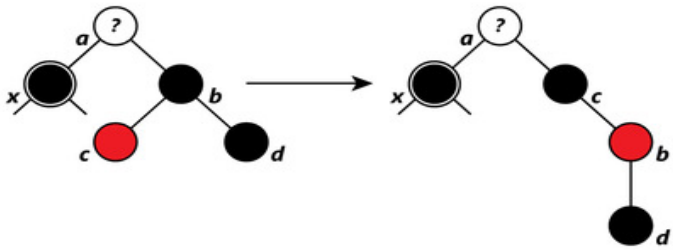


Рис. 6. Случай 4: "Брат" b вершины x – черный, левый ребенок "брата" c – красный, а правый d – черный

5) Если "брат" b вершины x – черный, правый ребенок брата d – красный. Делаем левое вращение b – a. Красим b в цвет, который был у a. Красим a в черный цвет. Так черная глубина x увеличится на 1, то есть восстановится. Этот случай представлен на рисунке 7.

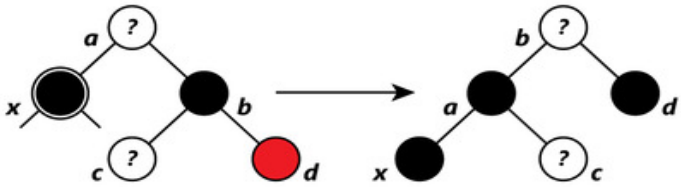


Рис. 7. Случай 5: “Брат” b вершины x – черный, правый ребенок брата d – красный

Экспериментально сравним эффективность АВЛ и Красно-черного дерева. Рассмотрим рост максимальной высоты дерева в зависимости от количества элементов. Тесты проводится на 25000000 элементах. Зависимость роста максимальной высоты АВЛ

дерева (представлена на рисунке 8) и Красно-черного дерева (представлена на рисунке 9).

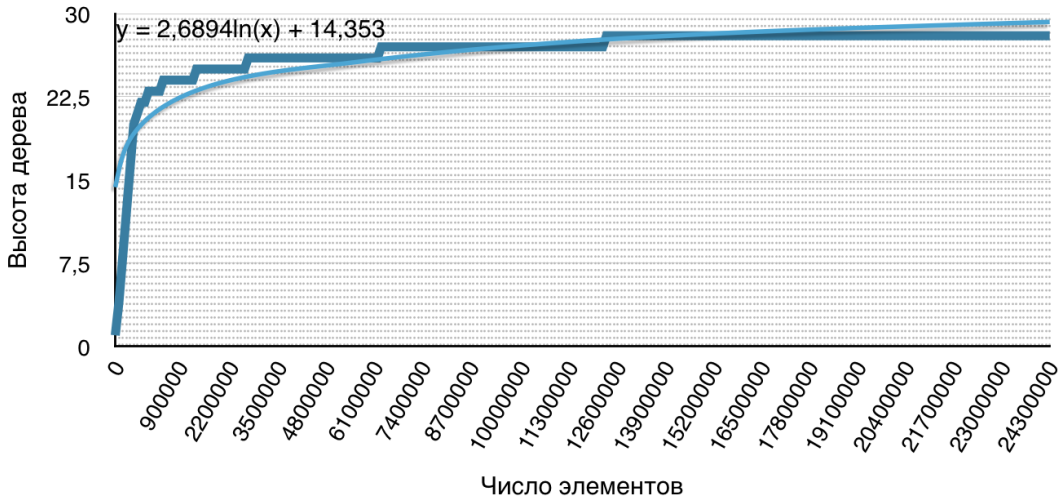


Рис. 8. Зависимость максимальной высоты АВЛ дерева от числа элементов в дереве

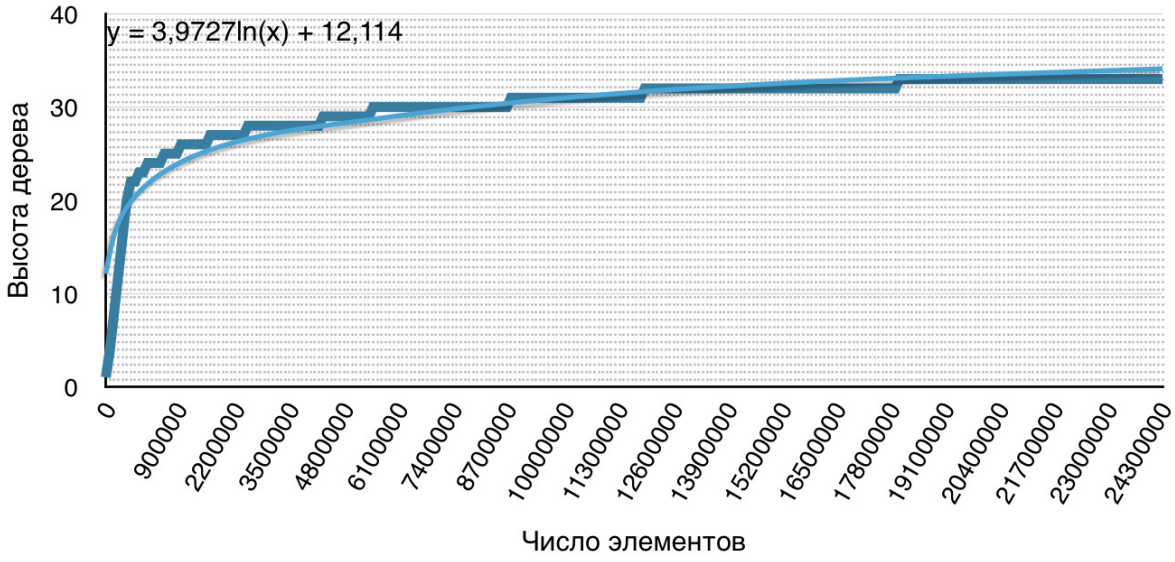


Рис. 9. Зависимость максимальной высоты Красно-черного дерева от числа элементов в дереве

Из этого следует, что максимальная высота красно-черного дерева растет быстрее, чем у АВЛ дерева, что приводит к увеличению времени работы алгоритма поиска в красно-черном дереве. Результаты представлены на рисунке 10 и на рисунке 11.

Операция поиска:

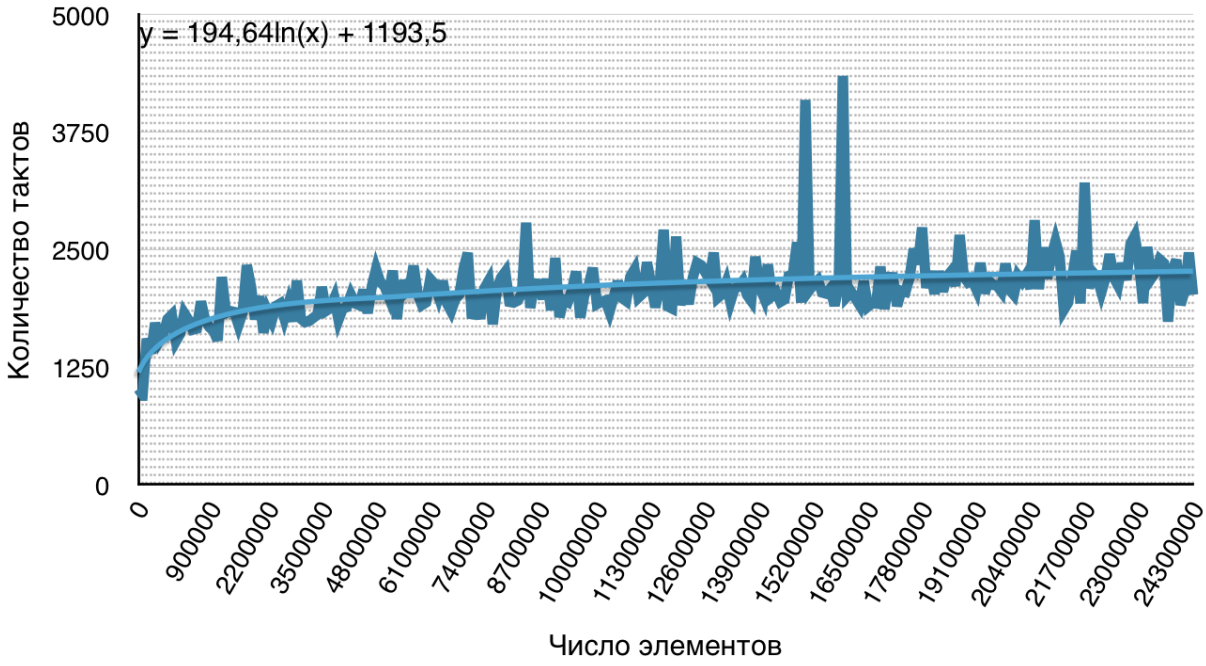


Рис. 10. Скорость выполнения операции поиска в зависимости от числа элементов в АВЛ дереве

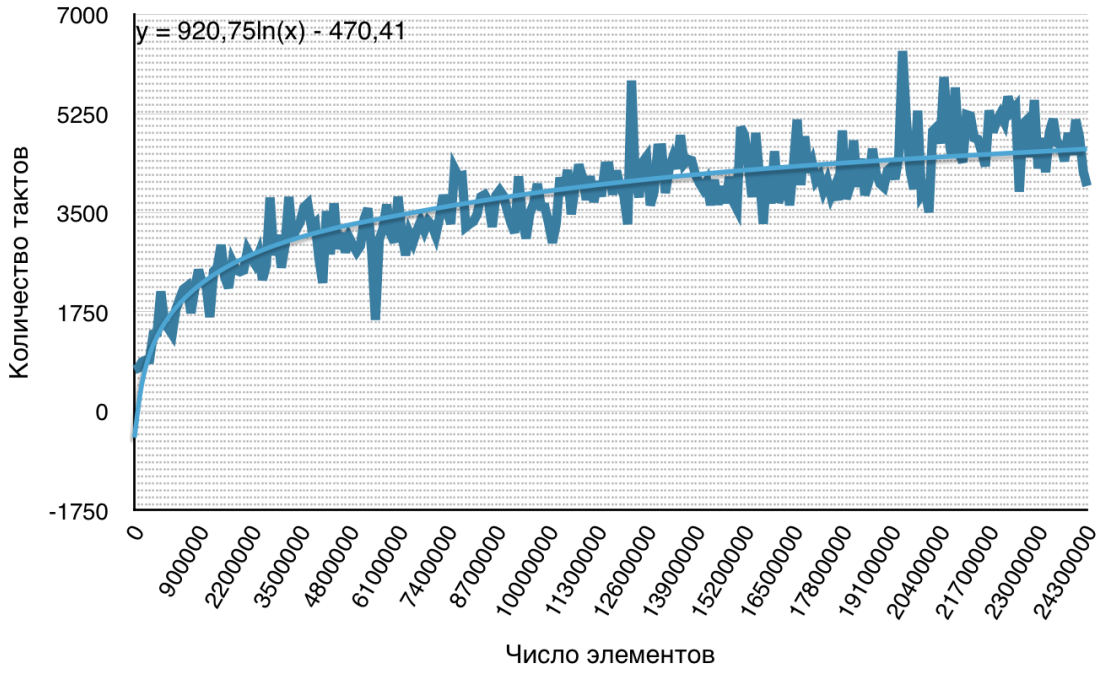


Рис. 11. Скорость выполнения операции поиска в зависимости от числа элементов в Красно-черном дереве

Операция вставки: Вставка требует до 2 поворотов в обоих видах деревьев. Из этого следует, что скорость выполнения операции вставки в АВЛ и Красно-черном дереве аналогична. Результаты представлены на рисунке 12 и на рисунке 13.

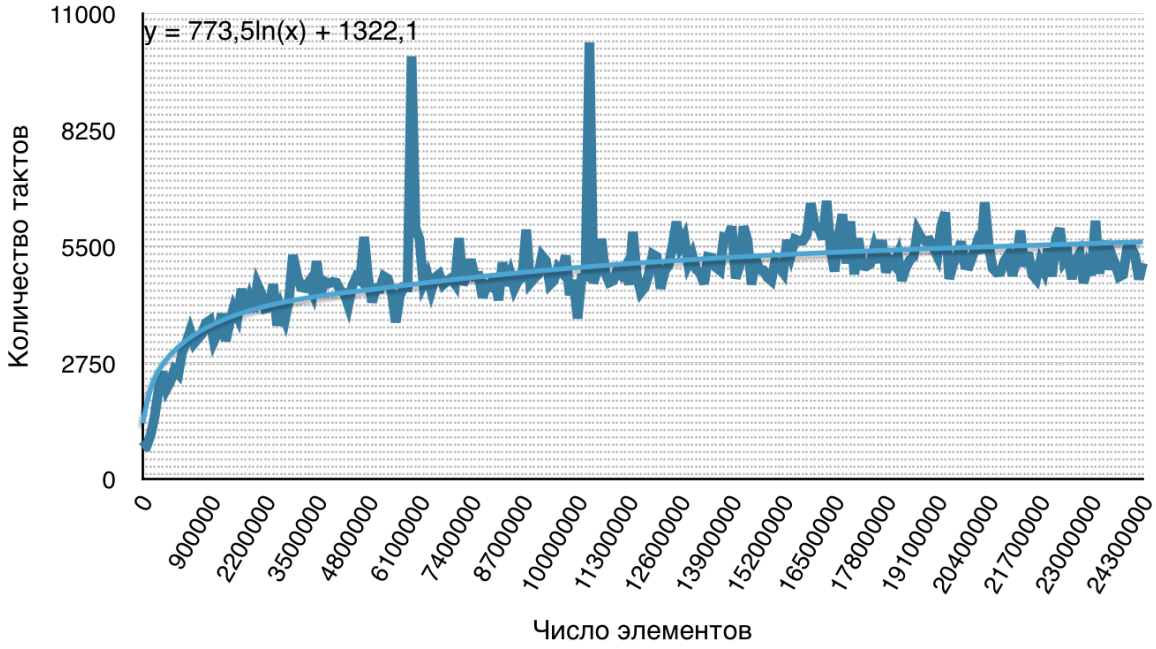


Рис. 12. Скорость выполнения операции вставки в зависимости от числа элементов в АВЛ дереве

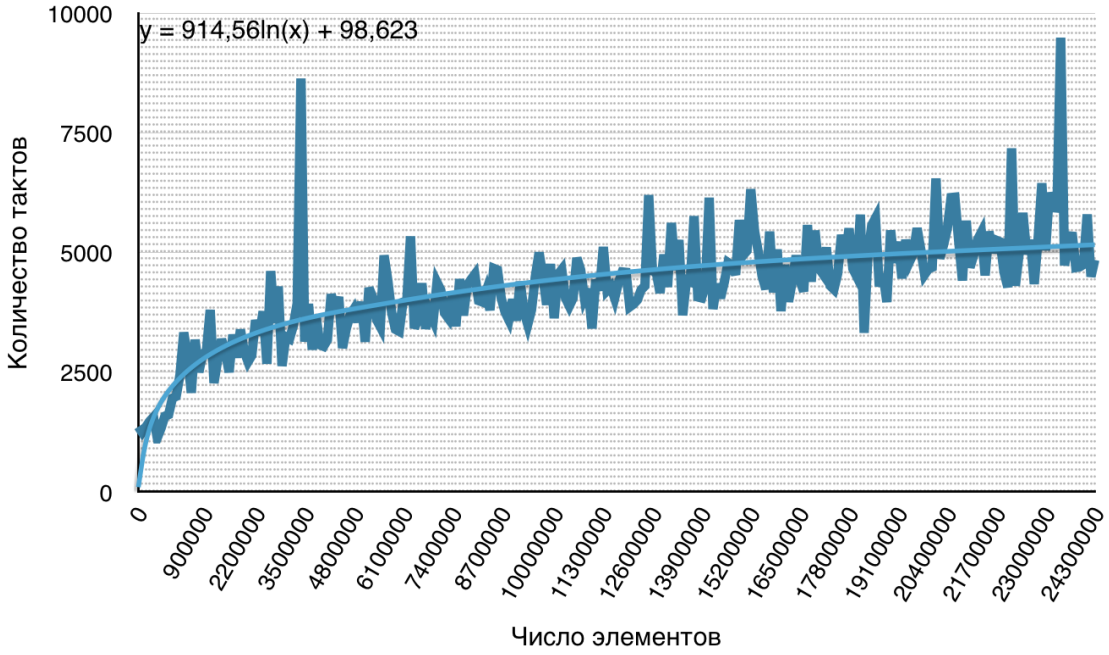


Рис. 13. Скорость выполнения операции вставки в зависимости от числа элементов в Красно-черном дереве

Операция удаления: Для удаления элемента из АВЛ дерева может потребоваться log(n) поворотов, в то время как в красно-черном дереве может потребоваться до 3

поворотов, следовательно, операция удаления из красно-черного дерева работает быстрее, чем в АВЛ дереве. Результаты представлены на рисунке 14 и на рисунке 15.

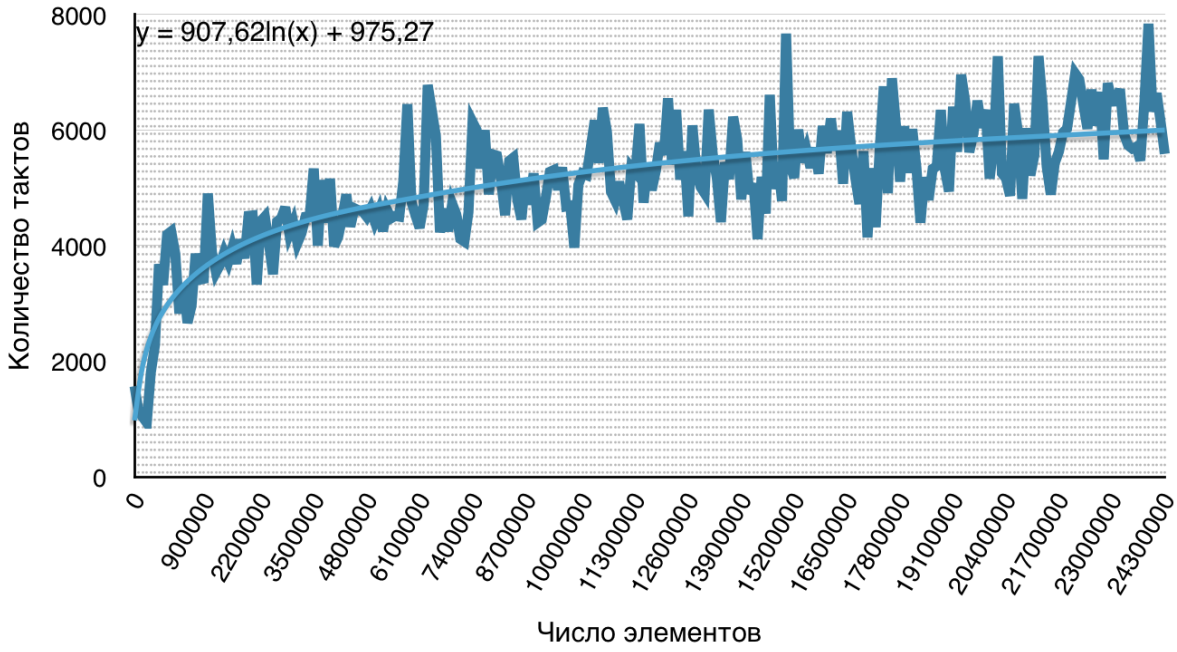


Рис. 14. Скорость выполнения операции удаления в зависимости от числа элементов в АВЛ дереве

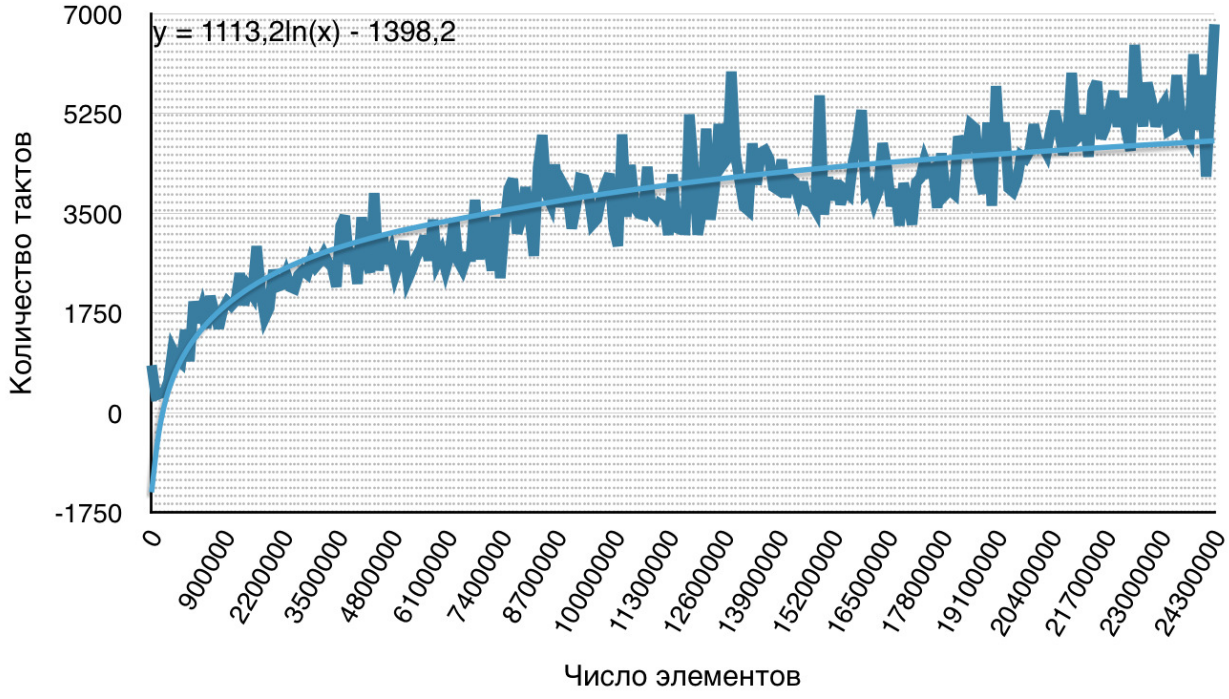


Рис. 15. Скорость выполнения операции удаления в зависимости от числа элементов в Красно-черном дереве

Выводы. В статье были рассмотрены алгоритмы работы АВЛ и Красно-черного дерева. Проведено сравнение показателей эффективности на основе операции поиска, вставки и удаления.

Красно-чёрные деревья являются наиболее активно используемыми на практике самобалансирующимися деревьями поиска. В частности, ассоциативные контейнеры библиотеки STL и класс TreeMap языка Java основаны на красно-чёрных деревьях. Однако, АВЛ дерево не получило широкое применение в промышленных библиотеках.

Список литературы

1. Ахо Альфред, Хопкрофт Джон Э., Ульман Д. Структуры данных и алгоритмы. М.: Вильямс. 2000. 384 с. [Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman Data Structures and Algorithms. Addison-Wesley, 1983. 427 p.]
2. Cormen Th.H., Leiserson Ch.E., Rivest R.L., Stein C. Introduction to Algorithms. 3rd Edition. The MIT Press Cambridge, Massachusetts London, England, 2009. 1312 p.
3. Кнут Д. Искусство программирования. В 4 т. Т. 1. Основные алгоритмы. М.: Вильямс, 2006. 720 с. [Donald E. Knuth. The Art of Computer Programming, vol. 1. Fundamental Algorithms, Third Edition. Addison-Wesley, 1997. 650 p.].