

Splay деревья. AA деревья

09.01.2023



Курпас Артём Викторович
ДФУ Б9121-09.03.03ПИКД

Node (узел, элемент x)

Содержит:

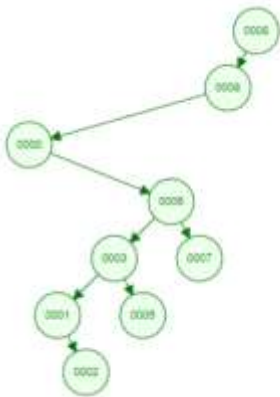
- Поле с данными (ключ)
- Ссылку на предка
- Ссылку на правого и левого потомков

Основные операции

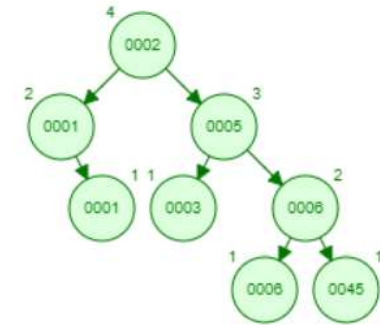
- Дерево – некоторое множество элементов S
- $access(x)$ – получение доступа в эл-ту x
- $insert(x)$ – вставка эл-та x
- $delete(x)$ – удаление эл-та x

Splay-деревья

- От англ. «растопыриваться»
- «самобалансирующаяся» структура данных
- было придумано Робертом Тарьяном и Даниелем Слейтером в 1983 году.



Splay-деревя



AVL-деревя

Преимущества

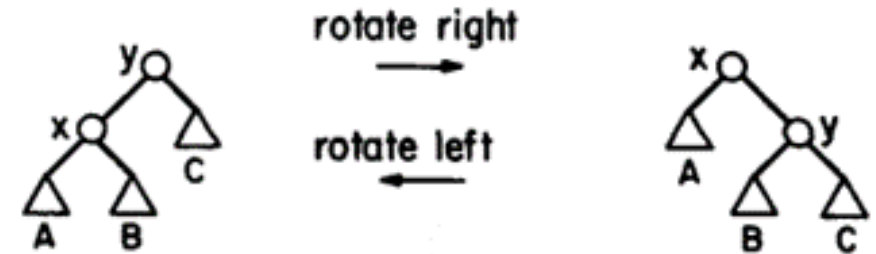
- Требуется меньше места
- Простая реализация

Splaying

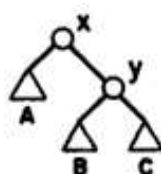
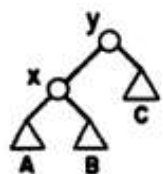
- Эвристика
- Выталкивание текущего эл-та x в корень дерева
- Делает быстрее доступ к недавним эл-там

Вращения (левое, правое)

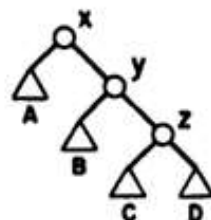
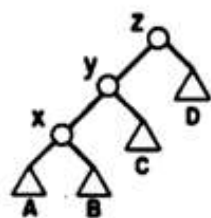
- Вспомогательная операция
- Сложность: $O(1)$
- Используются в $Zig(x)$, $Zig - Zig(x)$, $Zig - Zag(x)$



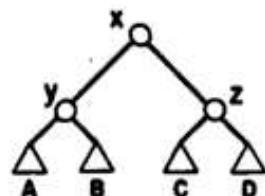
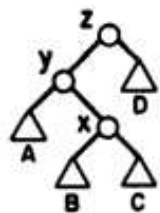
Операции $splay(x)$



(a)



(b)



(c)

- Вспомогательные операции
- A) $Zig(x)$ – одиночное вращение
- B) $Zig - Zig(x)$ – два одинаковых вращения подряд
- C) $Zig - Zag(x)$ – два разных вращения подряд

$Splay(x)$

- Вспомогательная операция
- Поднимает x в корень
- Использует $Zig(x)$, $Zig - Zig(x)$, $Zig - Zag(x)$
- Сложность: $O(h) \Leftrightarrow O(\log_2 n)$

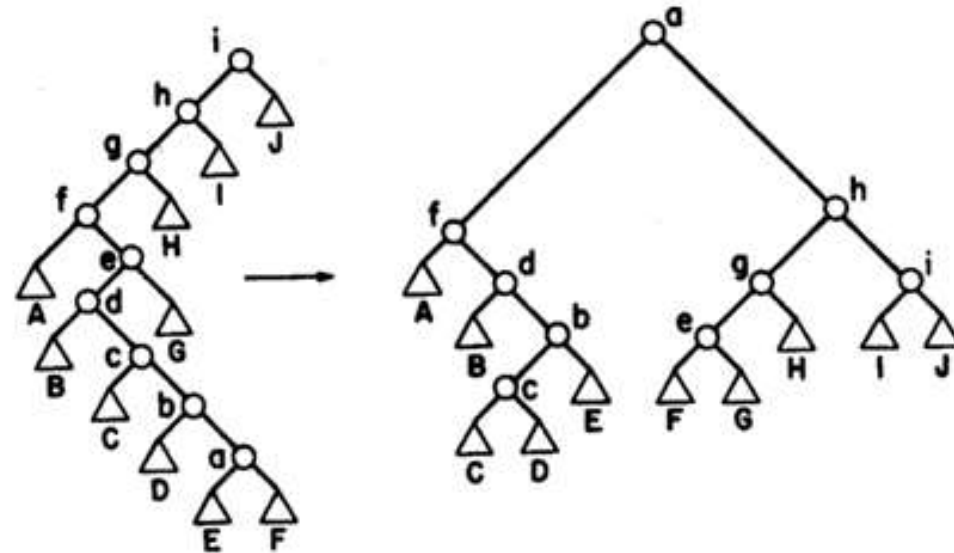


рис. Применение *splay* к узлу *a*

Работа с деревом. $access(x)$

- Начинаем от корня
- Спускаясь, ищем x
- Вызываем $splay(x)$
- $T(access(x)) \leq 2T(splay(x)) \Leftrightarrow O(\log_2 n)$

Работа с деревом. $access(x)$

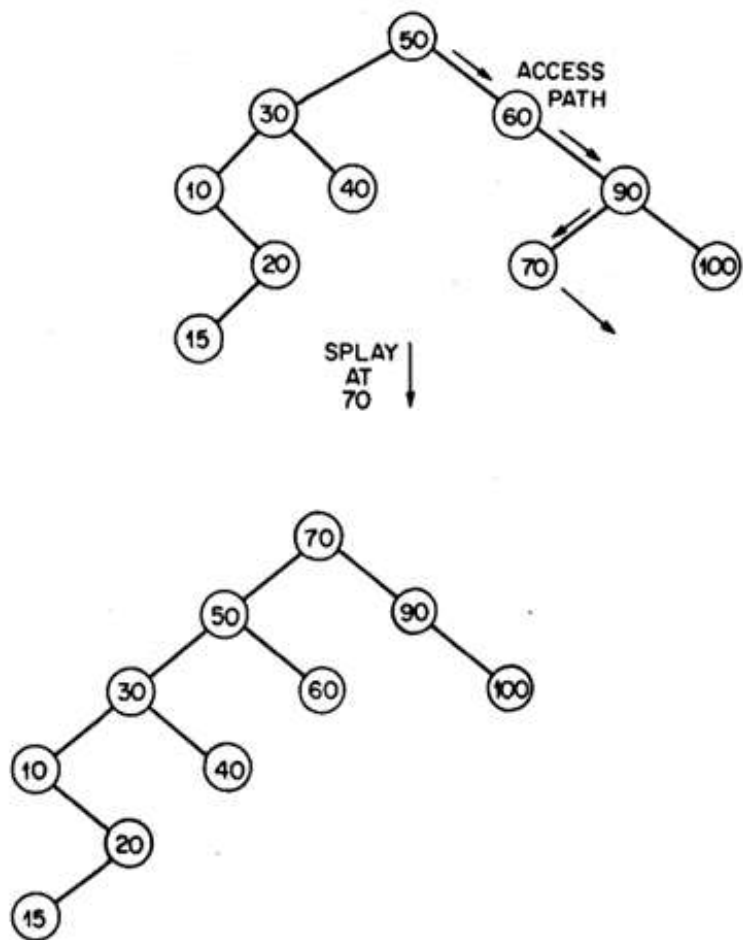


рис. Попытка найти узел с ключом 80.

Работа с деревом. *insert(x)*

- Начинаем от корня
- Спускаясь, ищем пустой узел
- Производим вставку
- Вызываем *splay(x)*
- $T(\text{insert}(x)) \leq 2T(\text{splay}(x)) \Leftrightarrow O(\log_2 n)$

Работа с деревом. $join(t_1, t_2)$

- Слияние двух поддеревьев
- Ищем максимальный эл-т i в t_1
- Вызываем $splay(i)$
- $i \rightarrow right = t_2$
- Возвращаем новое дерево t

Работа с деревом. *delete(x)*

- Выполняем *access(x)*
- Заменяем t (корень) на $join(t_1, t_2)$
- Очищаем память
- $T(delete(x)) = T(access(x)) \leq 2T(splay(x)) \Leftrightarrow O(\log_2 n)$

Работа с деревом. $insert(x)$ и $delete(x)$

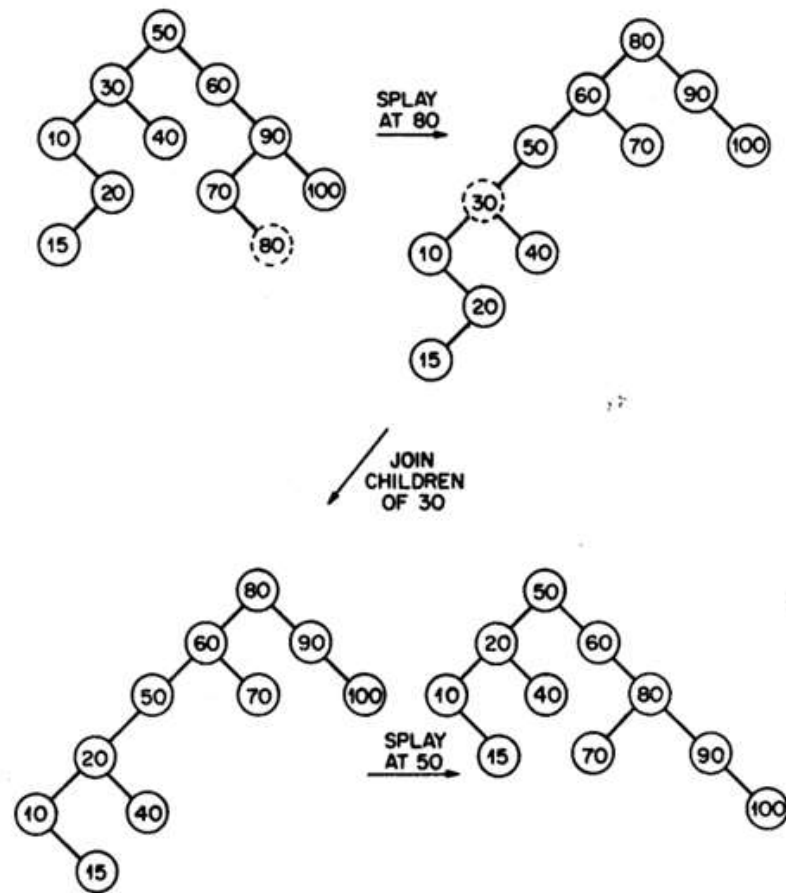


Рис. Реализации вставки и удаления.
За вставкой ключа 80 последовало
удаление узла с ключом 30

АА-деревья

- Сбалансированная структура данных
- Разновидность красно-чёрного дерева
- Было придумано Арне Андерссоном в 1993 году.

Преимущества

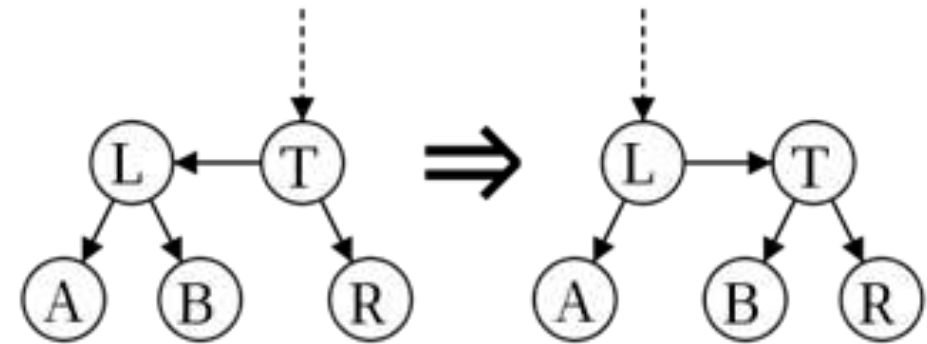
- Простая реализация
- Гарантированная производительность*

Свойства

- Уровень каждого листа равен 1 (или 0)
- Уровень каждого левого ребенка ровно на один меньше, чем у его родителя
- Уровень каждого правого ребенка равен или на один меньше, чем у его родителя
- Уровень каждого правого внука строго меньше, чем у его прародителя
- Каждая вершина с уровнем больше 1 имеет двоих детей

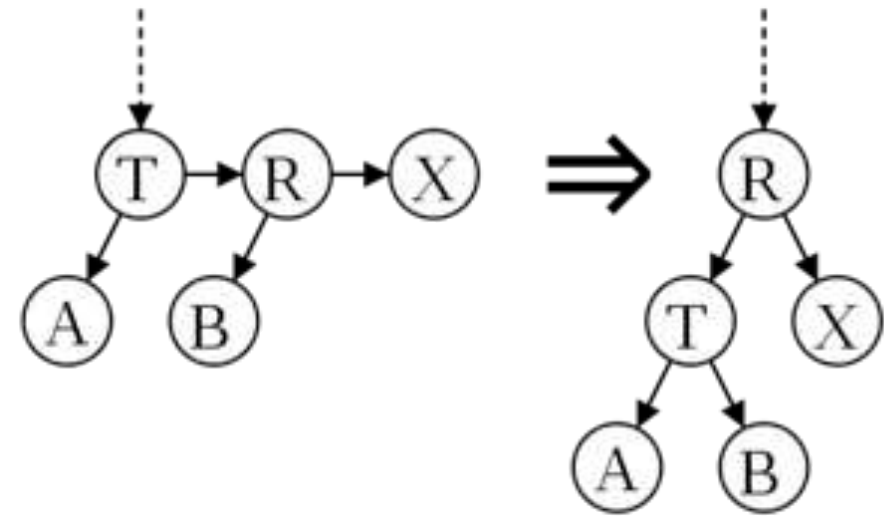
Операции балансировки. $skew(x)$

- $skew(x)$ – устранение левого горизонтального ребра
- Правое вращение
- Сложность: $O(1)$



Операции балансировки. $split(x)$

- $split(x)$ – устранение двух последовательных правых горизонтальных ребер
- R «выталкивается»
- Левое вращение
- Сложность: $O(1)$



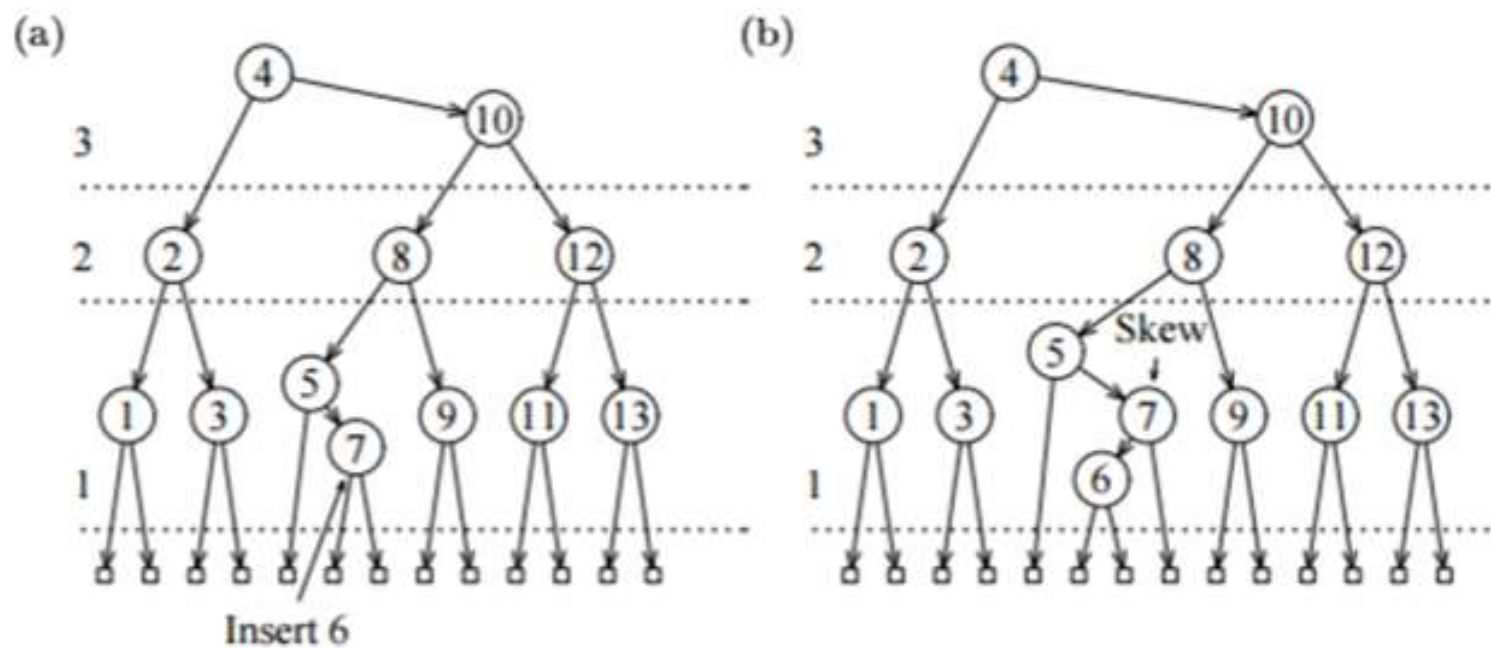
Работа с деревом. *access(x)*

- Начинаем от корня
- Спускаясь, ищем x
- $T(\text{access}(x)) \leq O(\log_2 n)$

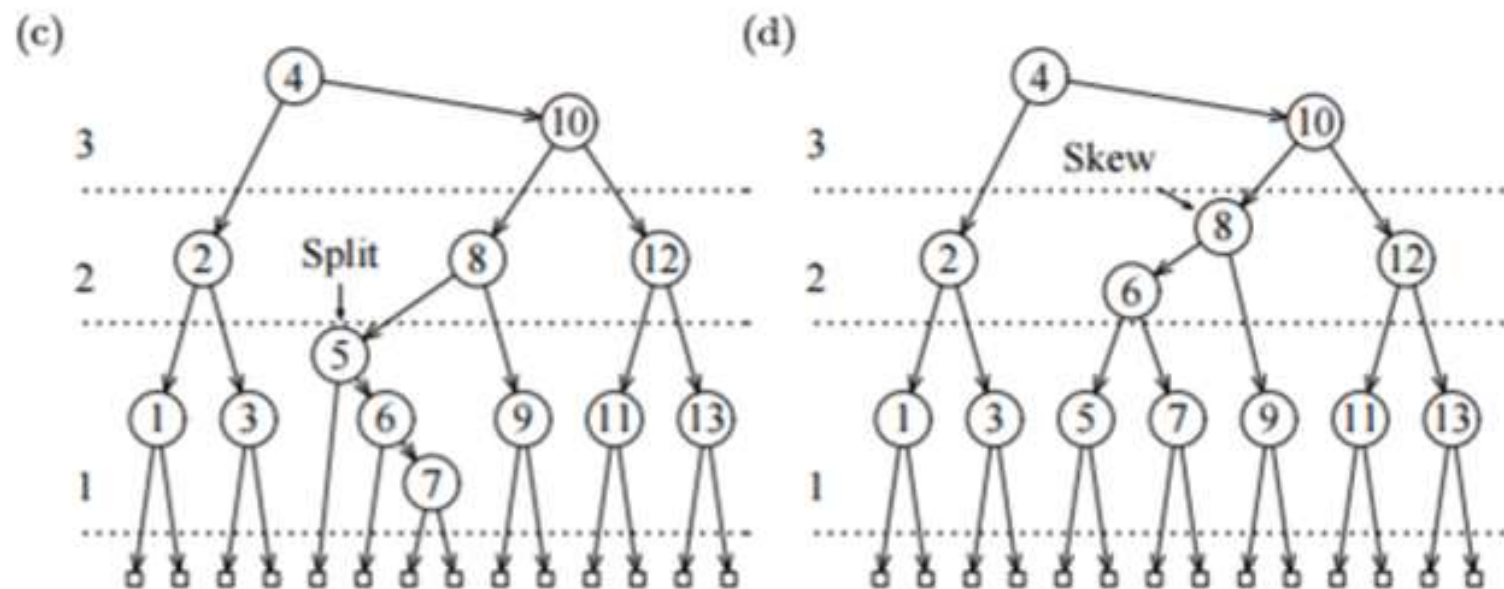
Работа с деревом. *insert(x)*

- Начинаем от корня
- Спускаясь, ищем пустой узел
- Производим вставку
- При подъёме к корню:
 - *skew(x)*
 - *split(x)*
- $T(\text{insert}(x)) \leq O(\log_2 n)$

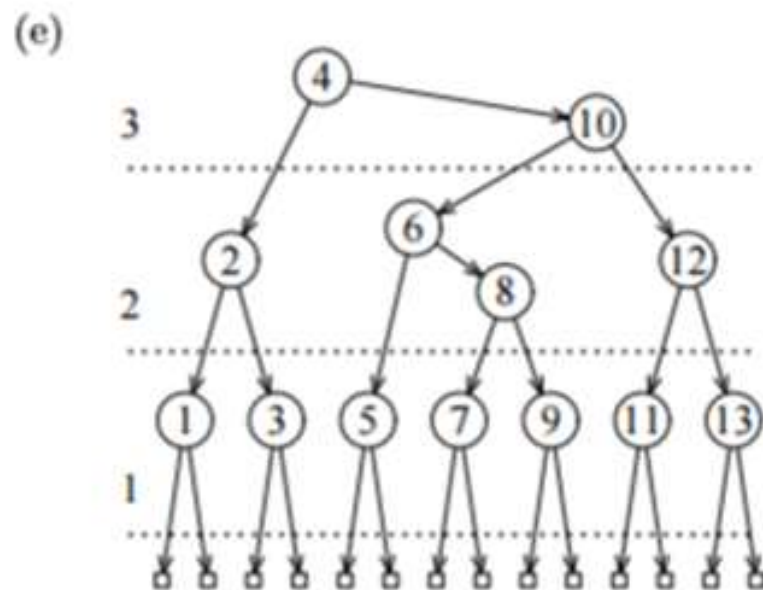
Работа с деревом. $insert(x)$



Работа с деревом. $insert(x)$



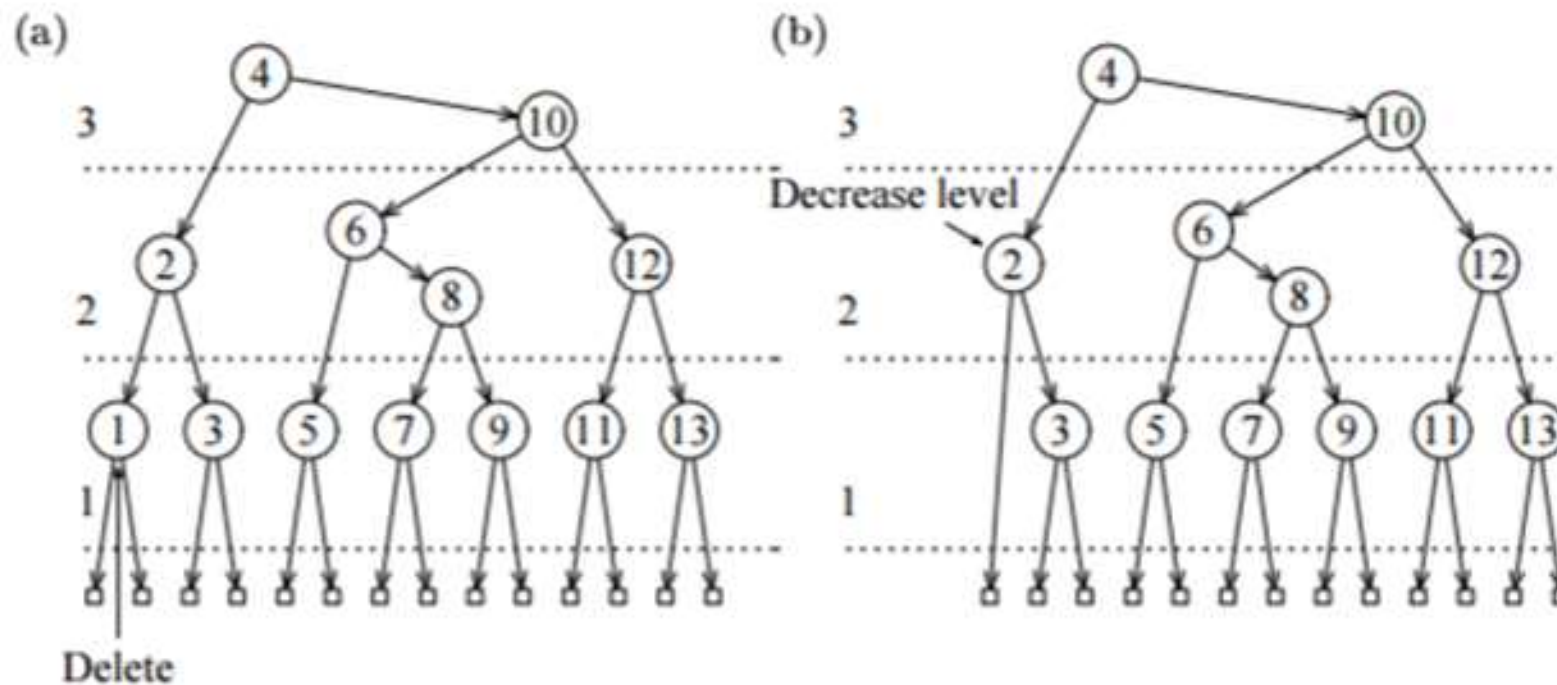
Работа с деревом. $insert(x)$



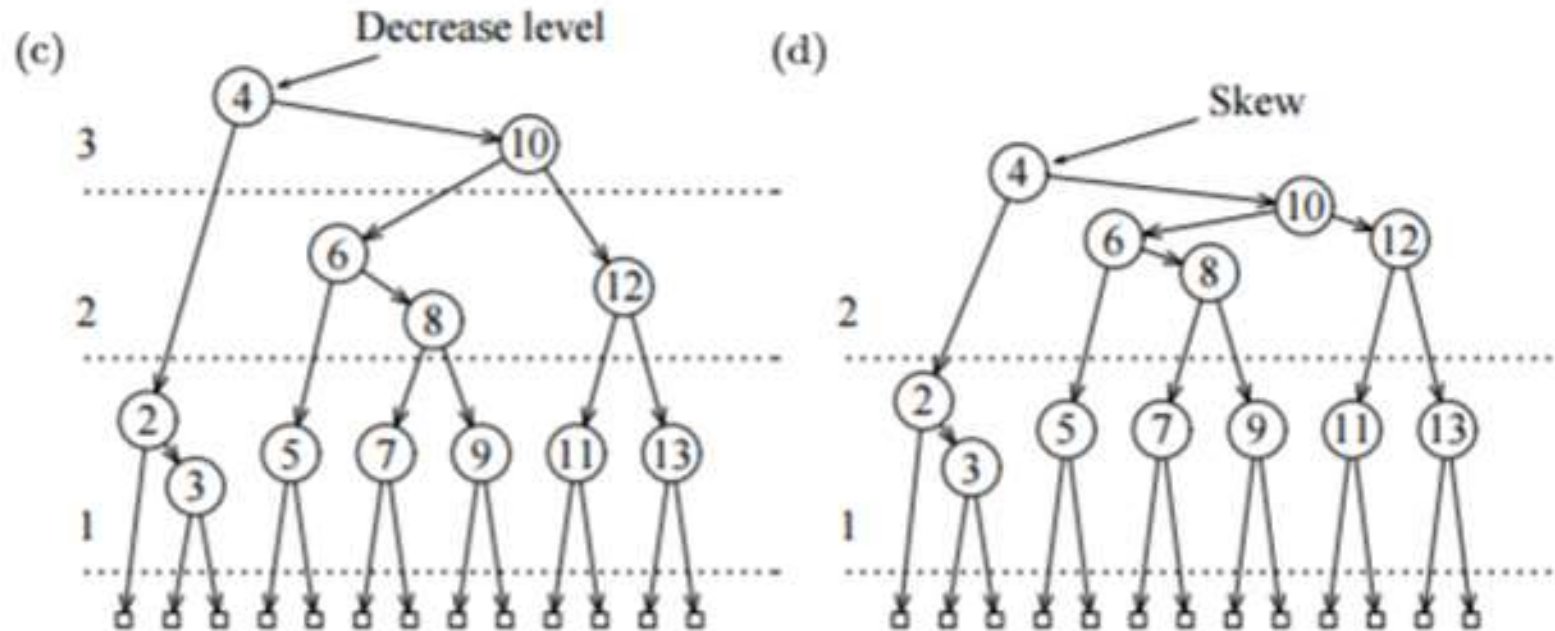
Работа с деревом. *delete(x)*

- *access(x)*
- x – лист:
 - Очищаем память
- x – не лист:
 - Заменяем x на «предшественника» или «преемника»
- При подъёме к корню:
 - Обновляем уровень всех вершин
 - *skew(x)*
 - *split(x)*
- $T(delete(x)) \leq O(\log_2 n)$

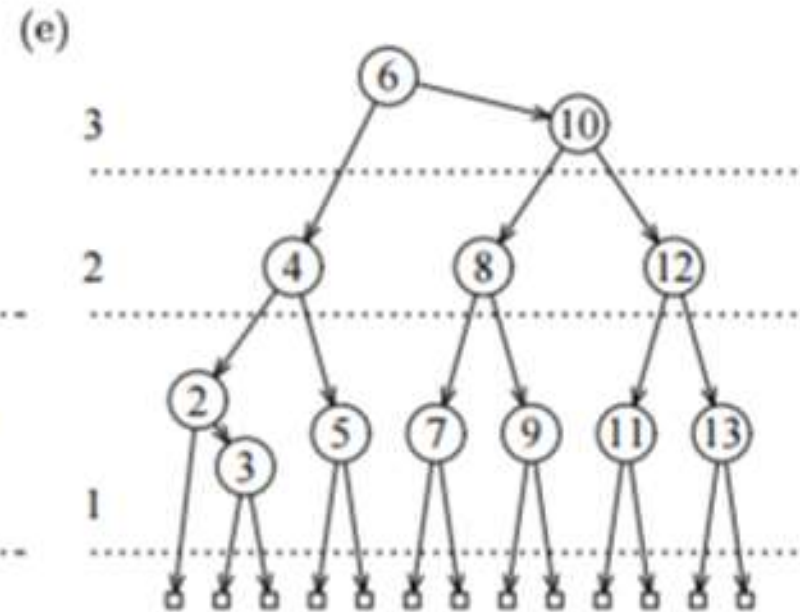
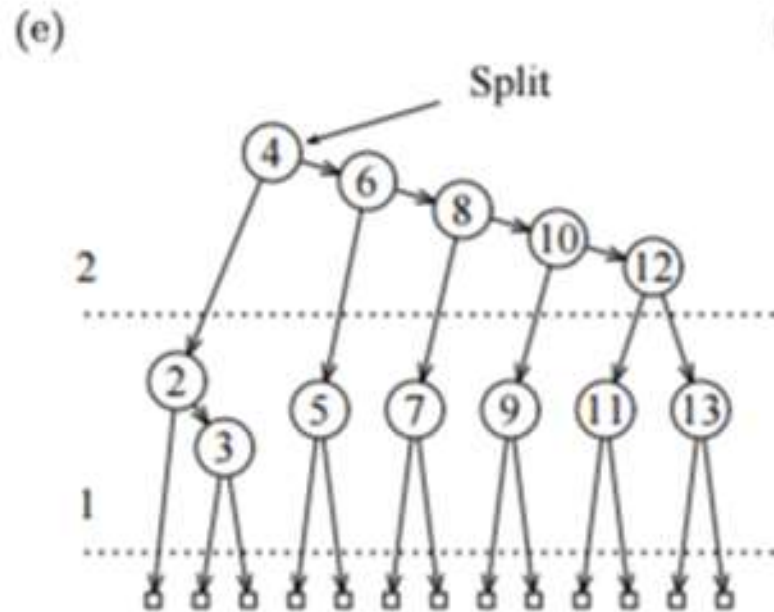
Работа с деревом. *delete(x)*



Работа с деревом. *delete(x)*



Работа с деревом. *delete(x)*

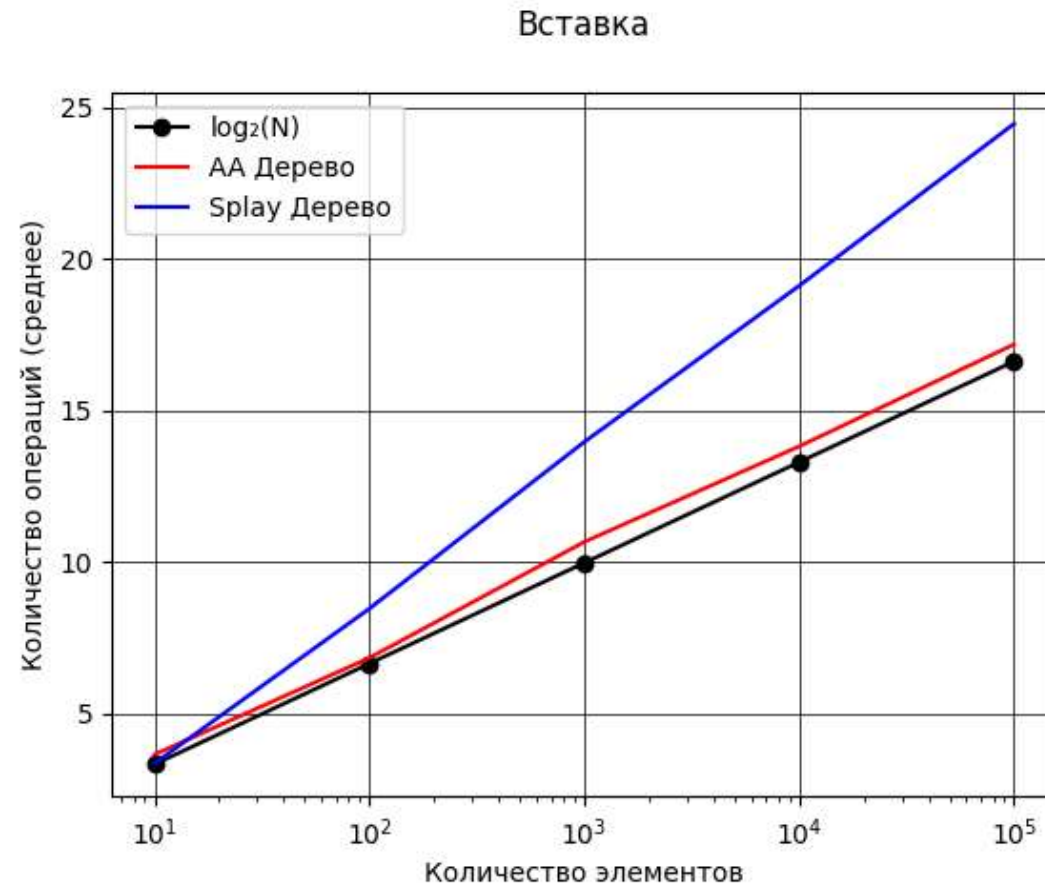


Тестирование и анализ производительности

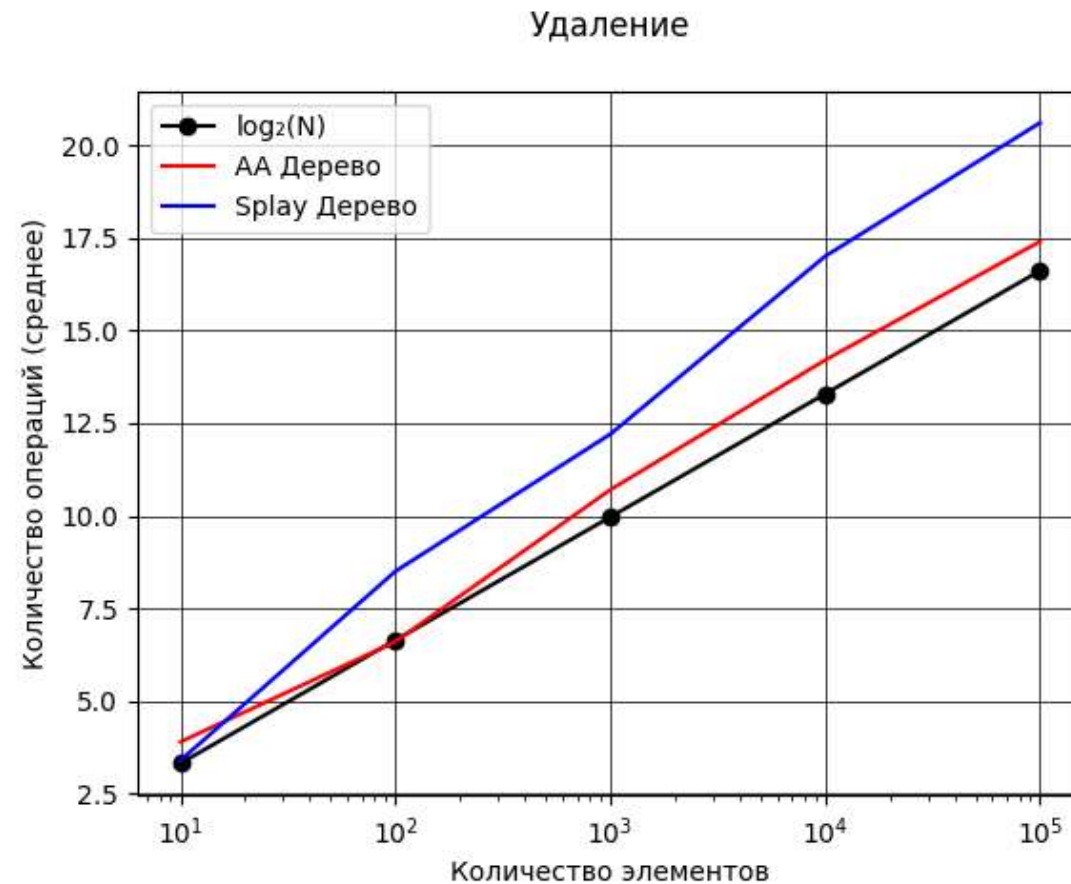
Методология тестирования

- *25 тестов*
 - *5 – на корректность*
 - *20 – на производительность*
- *run_tests.h – тестирующая программа*
- *test_generate.h – генератор случайных ключей*
- *chrono – замер времени*
- *get_insert_ops(), get_erase_ops(), get_access_ops() – встроенные методы для получения кол-ва операций*

Тестирование времени вставки (тесты 6-10)



Тестирование времени удаления (тесты 11-15)



Структура Splay-дерева

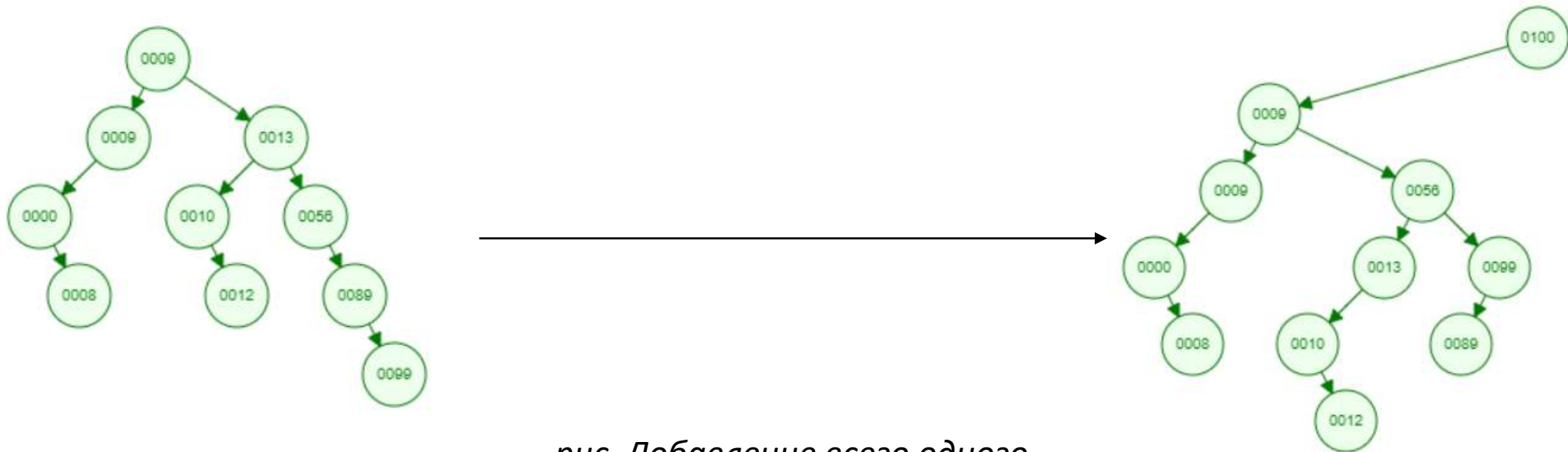
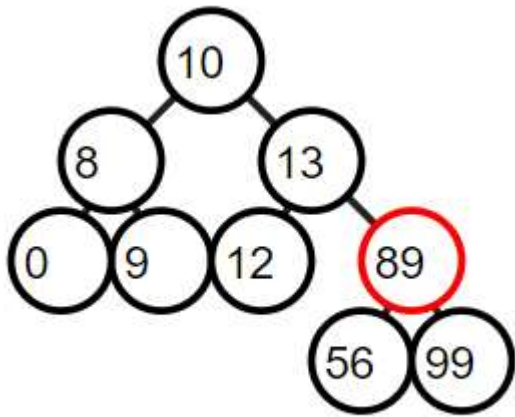
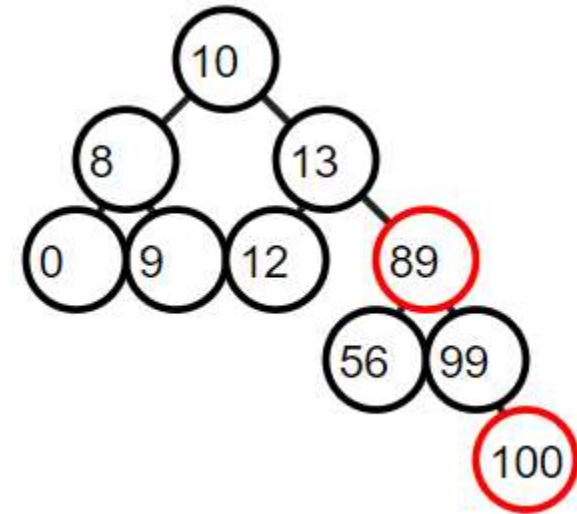


рис. Добавление всего одного элемента (100) может привести к сильной реструктуризации

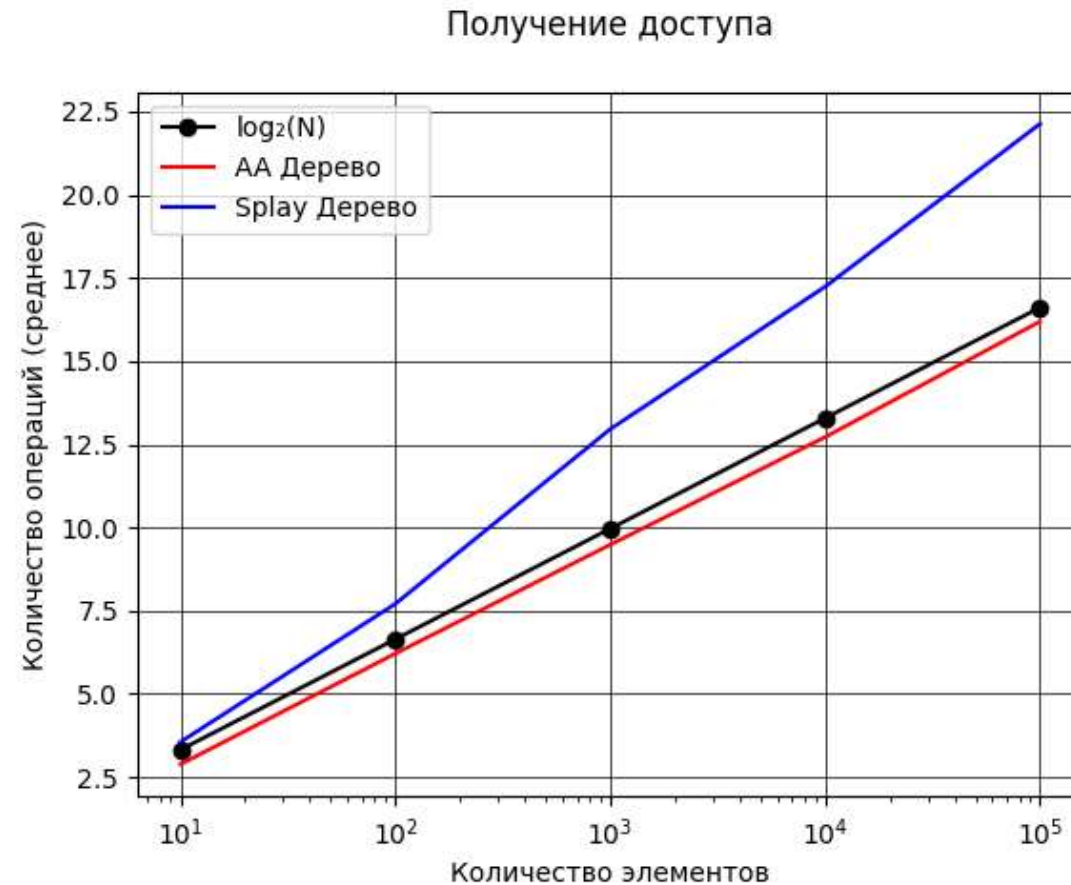
Структура АА-дерева



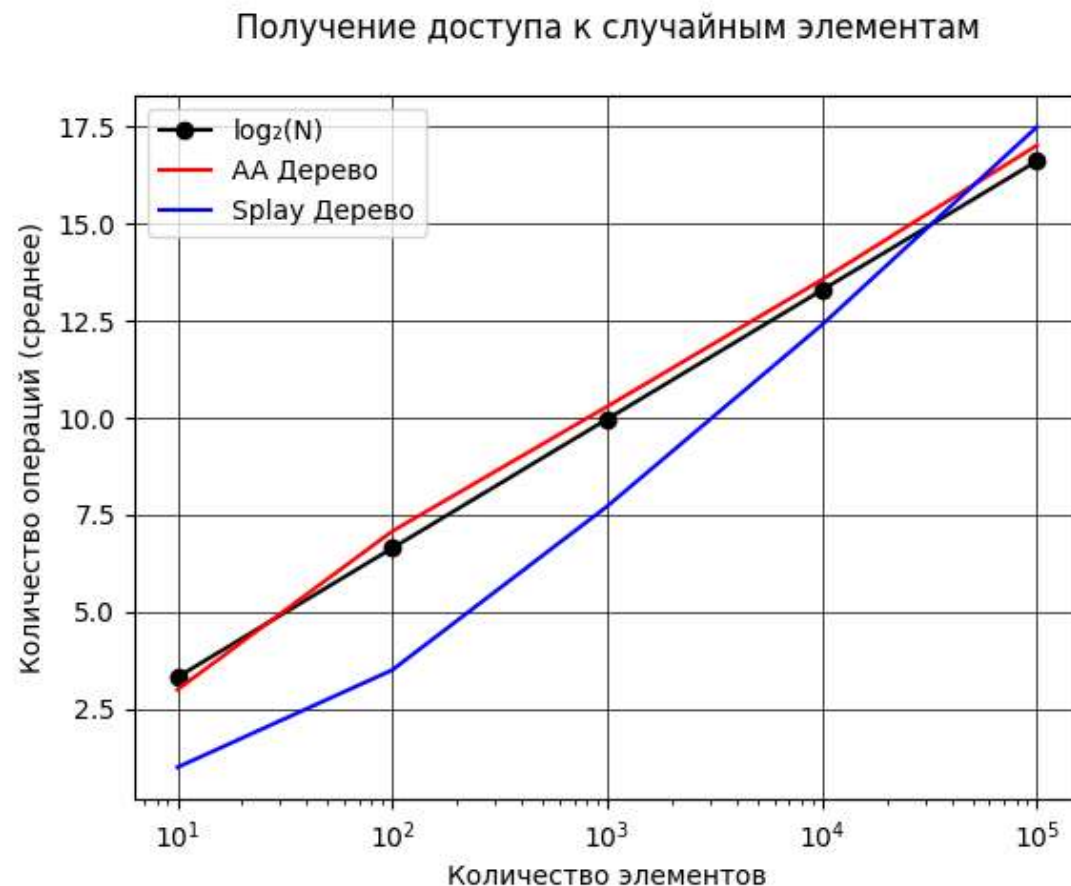
*рис. Добавление одного элемента
(100) не привело к значительным
изменениям в структуре*



Тестирование времени доступа (тесты 16-20)



Тестирование времени доступа к недавним элементам (тесты 21-25)



Спасибо за внимание!