МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ДОКЛАД
о практическом задании по дисциплине АиСД
«Сбалансированные деревья: AA-tree и Splay-tree»

направление подготовки 09.03.03 «Прикладная информатика»
профиль «Прикладная информатика в компьютерном дизайне»

Студент группы
Б9121-09.03.03пикд
Панкратова Екатерина Денисовна
_____
(подпись)
«_____» _____ 2023г.

Доклад защищен
с оценкой _____

Руководитель практики
Доцент ИМКТ А.С. Кленин
_____
(подпись)
«_____» _____2023г.

Регистрационный номер _____
«____» _____2023г.

г. Владивосток
2023

Содержание

СОДЕРЖАНИЕ 2

АННОТАЦИЯ 3

1. ВВЕДЕНИЕ 4

1.1. ГЛОССАРИЙ 4

1.2. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4

1.3. ПОСТАНОВКА ЗАДАЧИ 5

2. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ 6

3. ПРОЕКТ 7

4.1. СРЕДСТВА РЕАЛИЗАЦИИ 7

4.2. СТРУКТУРЫ ДАННЫХ 7

4.3. АЛГОРИТМ AA-TREE 7

4.4. АЛГОРИТМ SPLAY-TREE 11

5. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ 14

6. ИССЛЕДОВАНИЕ 14

ЗАКЛЮЧЕНИЕ 16

СПИСОК ИСТОЧНИКОВ 17

Аннотация

В данной работе представлены алгоритмы AA-tree и Splay-tree.
Цель:

- Разработать алгоритмы
- Описать реализацию алгоритмов
- Протестировать алгоритмы
- Провести исследование

Результаты: созданы библиотеки алгоритмов AA-tree и Splay-tree и автоматическая тестирующая система.

1. Введение

AA-tree – модификация красно-черного дерева, предложенная Арне Андерссоном в 1993 году. Это сбалансированное дерево, используемое для эффективного хранения и извлечения упорядоченных данных. [15]

Splay-tree – двоичное дерево поиска, созданное Робертом Тарьяном и Даниелем Слейтор в 1983 году. Поддерживается свойство сбалансированности. Позволяет находить те данные, которые использовались недавно. В то время как большинство подобных деревьев предназначены для уменьшения времени худшего случая для одной операции, данное дерево создано, чтобы сократить время для последовательности операций. [5] [28]

Цель: разработать и описать реализацию данных алгоритмов, протестировать их с помощью автоматических тестов.

Задачи:

1. Изучить теоретический материал по AA-tree и Splay-tree
2. Описать AA-tree и Splay-tree
3. Реализовать AA-tree и Splay-tree
4. Придумать тесты, проверяющие корректность работы алгоритмов
5. Придумать тесты, показывающие время работы алгоритмов
6. Реализовать тесты к AA-tree и Splay-tree
7. Описать результаты тестирования AA-tree и Splay-tree

1.1. Глоссарий

AA-tree – модификация красно-черного дерева, предложенная Арне Андерссоном в 1993 году. Это сбалансированное дерево, используемое для эффективного хранения и извлечения упорядоченных данных. [16]

Splay-tree – двоичное дерево поиска, созданное Робертом Тарьяном и Даниелем Слейтор в 1983 году. Поддерживается свойство сбалансированности. Позволяет находить те данные, которые использовались недавно. [5]

Бинарное дерево поиска — дерево, в котором узлы располагаются таким образом, что каждый узел с меньшим значением (относительно родителя) находится в левой части дерева, а с большим — в правой. [24]

Горизонтальное ребро — ребро, соединяющее вершины с одинаковым уровнем. [17]

Красно-чёрное дерево — один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный». [25]

Сбалансированное дерево поиска – такое дерево, в котором высота левого и правого поддеревьев отличаются не более чем на единицу. [24]

Уровень вершины — вертикальная высота соответствующей вершины. [17]

Эффективность алгоритма — это свойство алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом. [26]

1.2. Описание предметной области

Направление исследований: сбалансированные деревья AA-tree и Splay-tree, их реализация и тестирование.

История создания алгоритмов.

AA-tree было придумано Арне Андерсоном, который решил, что для упрощения балансировки дерева нужно ввести понятие уровня вершины. Если представить себе дерево растущим сверху вниз от корня (то есть «стоящим на листьях»), то уровень любой листовой

вершины будет равен 1. В своей работе Арне Андерсон приводит простое правило, которому должно удовлетворять AA-tree: к одной вершине можно присоединить другую вершину того же уровня, но только одну и только справа. [16]

Таким образом, введенное понятие уровня вершины не всегда совпадает с реальной высотой вершины (расстояния от земли), но дерево сохраняет балансировку при следовании правилу «одна правая связь на одном уровне». [16]

Splay-tree было придумано в середине восьмидесятых, когда Роберт Тарьян и Даниель Слейтор предложили несколько красивых и эффективных структур данных. Все они имеют несложную базовую структуру. [4]

Splay-tree — это самобалансирующееся бинарное дерево поиска. Дереву не нужно хранить никакой дополнительной информации, что делает его эффективным по памяти. После каждого обращения, даже поиска, Splay-tree меняет свою структуру. [4]

1.3. Постановка задачи

Код должен быть оформлен в виде библиотеки (отдельного модуля) на выбранном языке программирования.

Должны быть представлены автоматические тесты.

Язык программирования должен позволять максимально эффективную реализацию, то есть реализация алгоритма не должна быть существенно медленнее реализации на C++.

Оформление и структура кода должны подчиняться обычным критериям качества.

2. Функциональные требования

Алгоритмы должны:

- Быть оформлены в виде библиотеки (.h)
- Иметь функции: удаления, поиска, вставки элемента
- Иметь дополнительные функции, присущие AA-tree и Splay-tree

Тесты должны:

- Выполнять автоматическое тестирование алгоритмов
- Определять время выполнения алгоритма

3. Проект

4.1. Средства реализации

В качестве языка программирования был выбран C++, так как он быстрый, подходит для целей работы и при этом более удобный и современный, чем C.

В качестве среды разработки был выбран Visual Studio Code, как один из самых популярных средств написания кода.

4.2. Структуры данных

В данной работе используется структура данных Node – узел дерева. В ней содержатся поля типа public.

В Splay-tree имеются поля value – значение в узле, left и right – указатели на левого и правого потомков, parent – указатель на родителя.

```
struct Node
{
    int value;
    Node *left;
    Node *right;
    Node *parent;

    Node(int value) {
        this->value = value;
        this->left = this->right = this->parent = nullptr;
    }
     };
```

В алгоритме AA-tree добавлено поле level – для хранения уровня вершины.

```
struct AANode
{
    int value;
    int level;
    AANode* left;
    AANode* right;
    AANode* parent;

    AANode(int value) {
        this->level = 1;
        this->value = value;
        this->left = this->right = this->parent = nullptr;
    }
     };
```

4.3. Алгоритм AA-tree

Свойства AA-дерева:

1. Уровень каждого листа равен 1.
2. Уровень каждого левого ребенка ровно на один меньше, чем у его родителя.
3. Уровень каждого правого ребенка равен или на один меньше, чем у его родителя.
4. Уровень каждого правого внука строго меньше, чем у его прародителя.
5. Каждая вершина с уровнем больше 1 имеет двоих детей.

Связь с красно-чёрным деревом

В отличие от красно-черных деревьев, красные вершины могут быть добавлены только в качестве правого ребенка. [17]

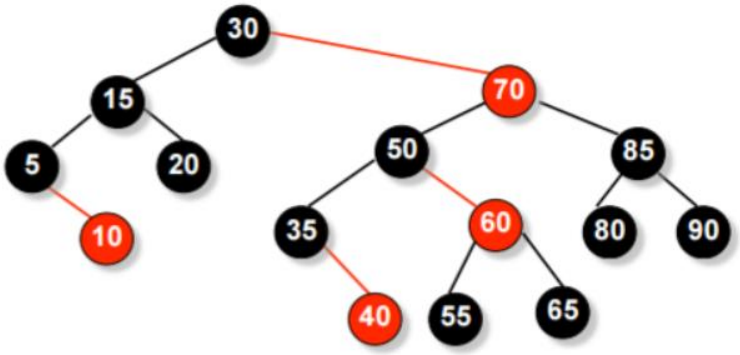

Рисунок 1. Пример красно-черного дерева

Рассмотрим то же дерево, но с информацией об уровне каждой вершине. Горизонтальные ребра обозначают связи между ребрами одного уровня. [17]

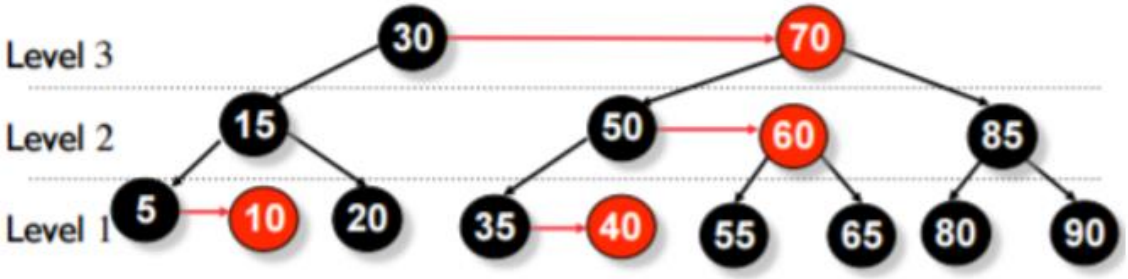

Рисунок 2. Пример дерева с информацией об уровне вершины

В AA-tree вместо значения цвета в вершине хранится информация только о ее уровне. [17]

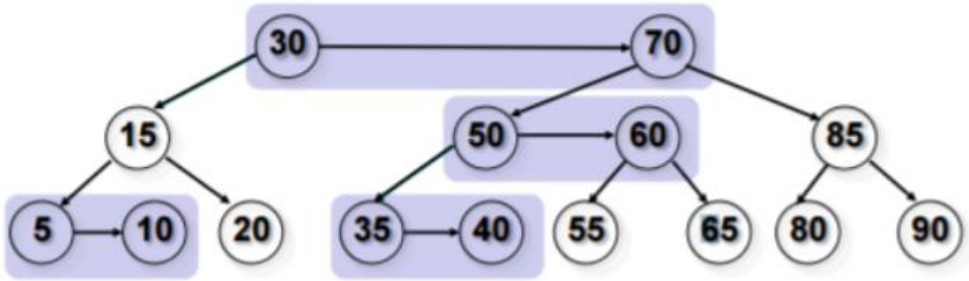

Рисунок 3. AA-tree

Для поддержки баланса красно-черного дерева необходимо обрабатывать 7 различных вариантов расположения вершин. [17]

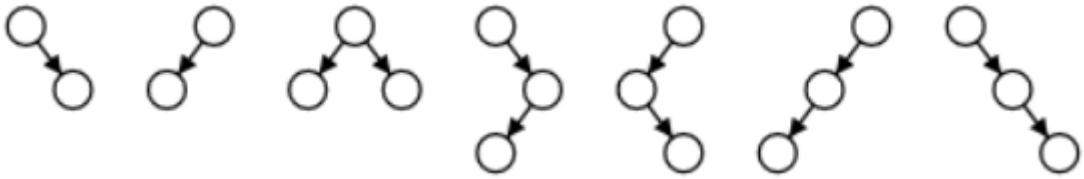

Рисунок 4. Варианты расположения вершин в красно-черном дереве

В АА-дереве из-за строгих ограничений необходимо обрабатывать только два вида возможных расположений вершин, чтобы проверить соблюдается ли главное правило «одна правая горизонтальная связь». То есть надо проверить нет ли левой горизонтальной связи или двух последовательных правых горизонтальных связей. [17]

Рисунок 5. Варианты расположения вершин в AA-tree

В АА-дереве разрешены правые ребра, не идущие подряд, и запрещены все левые горизонтальные ребра. Эти более жесткие ограничения, аналогичные ограничениям на красно-черных деревьях, приводят к более простой реализации балансировки АА-дерева. [17]

Для балансировки АА-дерева нужны следующие две операции: Skew и Split.

Skew

Skew() — устранение левого горизонтального ребра. Делаем правое вращение, чтобы заменить поддерево, содержащее левую горизонтальную связь, на поддерево, содержащее разрешенную правую горизонтальную связь. [17]

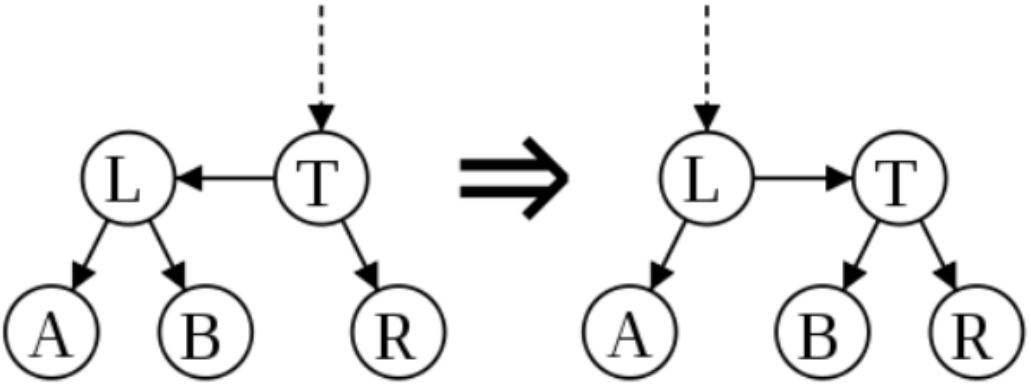

Рисунок 6. Пример операции Skew()

Split

Split() — устранение двух последовательных правых горизонтальных ребер. Делаем левое вращение и увеличиваем уровень, чтобы заменить поддерево, содержащее две или более последовательных правильных горизонтальных связи, на вершину, содержащую два поддерева с меньшим уровнем. [17]

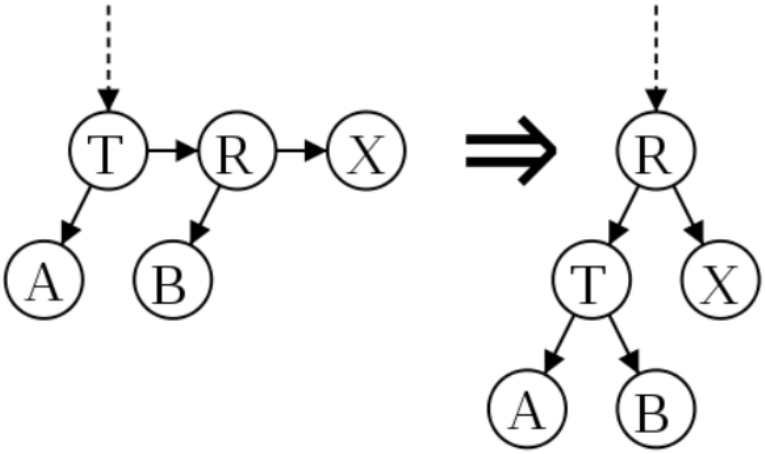

Рисунок 7. Пример операции Split()

Вставка элемента

Вставка нового элемента происходит как в обычном дереве поиска, только на пути вверх необходимо делать ребалансировку, используя Skew() и Split(). [17]

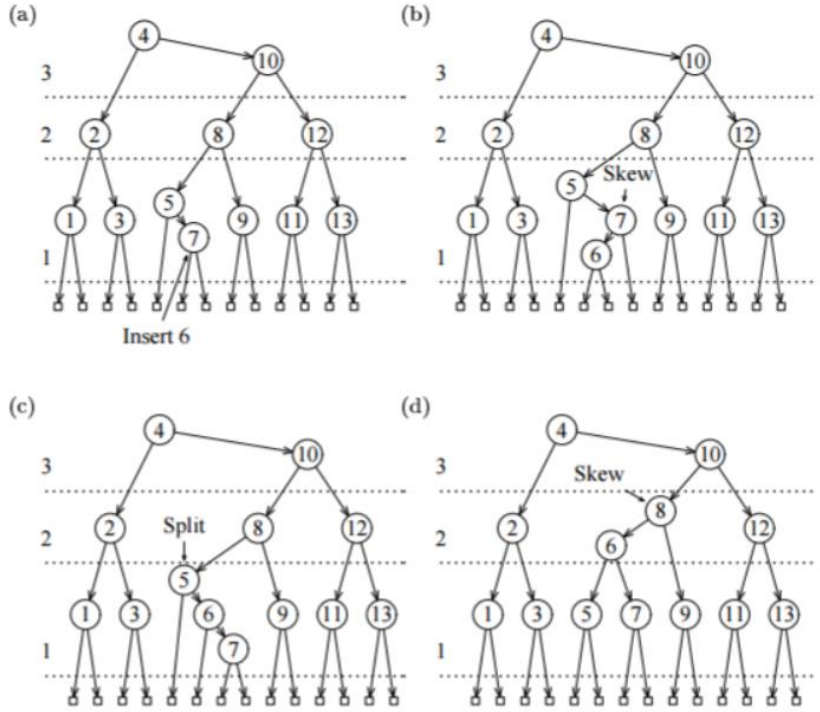

Рисунок 8.1. Пример вставки элемента

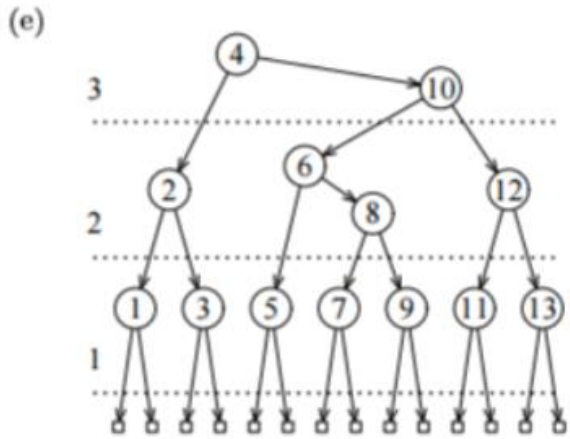

Рисунок 8.2. Пример вставки элемента

Удаление вершины

Как и в большинстве сбалансированных бинарных деревьев, удаление внутренней вершины можно заменить на удаление листа, если заменить внутреннюю вершину на ее ближайшего «предшественника» или «преемника». «Предшественник» находится в начале последнего левого ребра, после которого идут все правые ребра. «Преемник» может быть найден после одного правого ребра и последовательности левых ребер, пока не будет найден указатель на NULL. В силу свойства всех узлов уровня более чем 1, имеющих двух детей, предшественник или преемник будет на уровне 1. [17]

Будем использовать дополнительную функцию DecreaseLevel(), она будет обновлять уровень вершины, которую передают в функцию, в зависимости от значения уровня дочерних вершин. [17]

Чтобы сохранять баланс дерева необходимо делать Skew(), Split() и DecreaseLevel() для каждой вершины. [17]

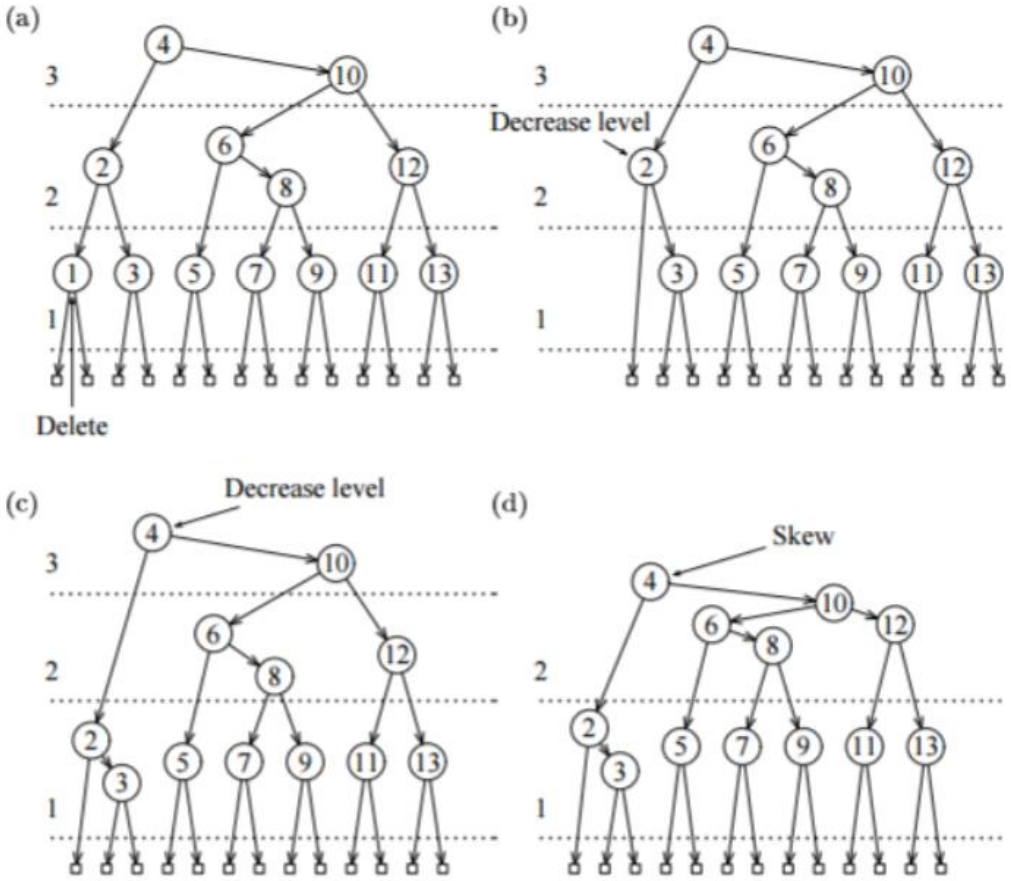

Рисунок 9.1. Пример удаления вершины

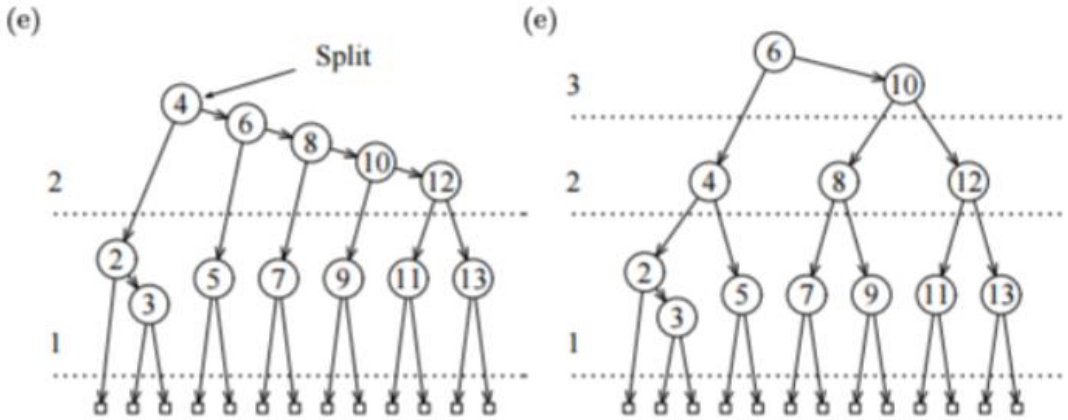

Рисунок 9.2. Пример удаления вершины

Преимущества

Дерево сохраняет балансировку при следовании правилу «одна правая связь на одном уровне», что делает это дерево быстрым и простым в реализации. [19]

Для балансировки АА-дерева нужно всего две операции Skew и Split. [19]

4.4. Алгоритм Splay-tree

Это дерево принадлежит классу «саморегулирующихся деревьев», которые поддерживают необходимый баланс ветвления дерева, чтобы обеспечить выполнение операций поиска, добавления и удаления за логарифмическое время от числа хранимых элементов. Это реализуется без использования каких-либо дополнительных полей в узлах дерева. Вместо этого «расширяющие операции» (splay operation), частью которых являются вращения, выполняются при каждом обращении к дереву. [6]

Splay (расширение)

Основная операция дерева. Заключается в перемещении вершины в корень при помощи последовательного выполнения трёх операций: Zig, Zig-Zig и Zig-Zag. Обозначим вершину, которую хотим переместить в корень за x, её родителя — p, а родителя p (если существует) — g. [6]

Zig: выполняется, когда p является корнем. Дерево поворачивается по ребру между x и p. Существует лишь для разбора крайнего случая и выполняется только один раз в конце, когда изначальная глубина x была нечётна. [6]

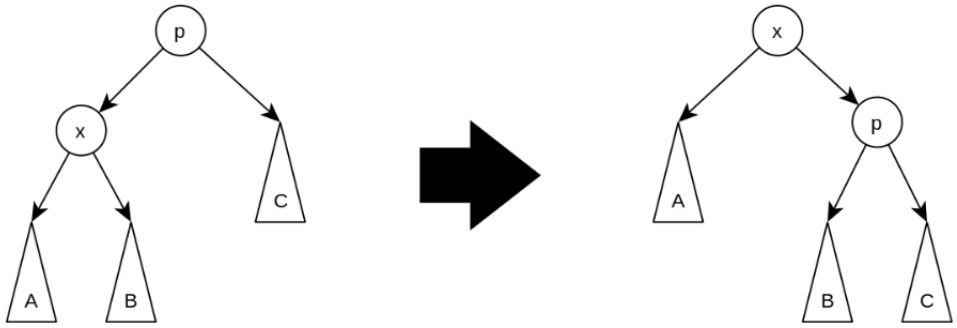

Рисунок 10. Пример операции Zig

Zig-Zig: выполняется, когда и x, и p являются левыми (или правыми) сыновьями. Дерево поворачивается по ребру между g и p, а потом — по ребру между p и x. [6]

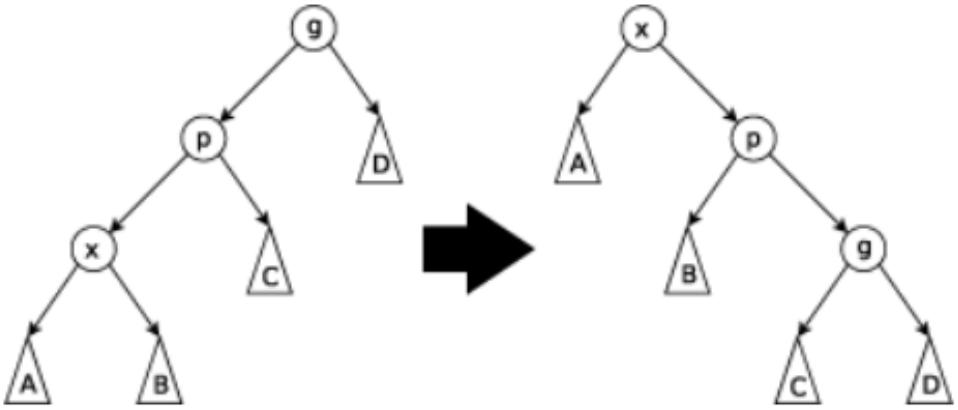

Рисунок 11. Пример операции Zig-Zig

Zig-Zag: выполняется, когда x является правым сыном, а p — левым (или наоборот). Дерево поворачивается по ребру между p и x, а затем — по ребру между x и g. [6]

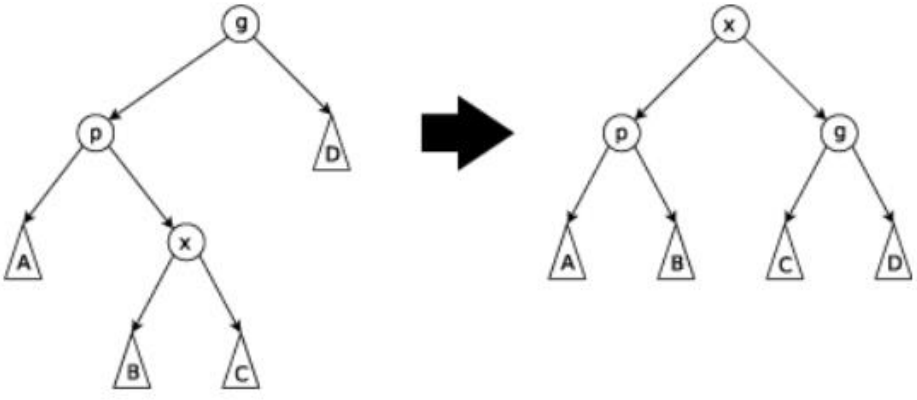

Рисунок 12. Пример операции Zig-Zag

Search (поиск элемента)

Поиск выполняется как в обычном двоичном дереве поиска. При нахождении элемента запускаем Splay() для него. [6]

Insert (добавление элемента)

Запускаем Split() от добавляемого элемента. [6]

Delete (удаление элемента)

Проводится в три этапа: находим элемент в дереве и делаем Splay() для него, делим дерево на два поддерева с помощью операции Split(), удаляем лишние связи и соединяем поддеревья в одно дерево с помощью операции Join(). [6]

Преимущества

Хорошая производительность, так как это самооптимизирующееся дерево, часто используемые узлы будут приближаться к корню, где к ним можно будет получить доступ быстрее. [13]

Небольшой объем занимаемой памяти, так как в Splay-деревьях не требуется хранить какие-либо дополнительные данные. [13]

Недостатки

Наиболее существенный недостаток такого дерева состоит в том, что высота может быть линейной. Например, это произойдет после доступа ко всем элементам в неубывающем порядке. В таком случае эффективность алгоритма уменьшается. [13]

Если доступ к элементам будет случайный, это увеличит время выполнения алгоритма из-за операций расширения. [13]

5. Реализация и тестирование

Для реализации поставленной задачи было создано:

1. Библиотека AA-tree
2. Библиотека Splay-tree
3. Система автоматического тестирования

Обе библиотеки имеют следующие публичные методы:

- GetRoot () – возвращает ссылку корневой узел
- Insert (int value) – вставка элемента
- Search (int value) – поиск элемента
- Delete (int value) – удаление элемента

Также они имеют не совпадающие методы.

Библиотека AA-tree имеет следующие приватные методы:

- SearchNode (AANode* node, int value) – поиск элемента
- InsertNode (AANode* node, int value) – вставка элемента
- Skew (AANode* node) – устранение левого горизонтального ребра
- Split (AANode* node) – устранение двух последовательных горизонтальных правых ребер
- DeleteNode (AANode* node, int value) – удаление элемента
- Predecessor (AANode* node) – поиск наибольшего элемента в левом поддереве
- Successor (AANode* node) – поиск наименьшего элемента в правом поддереве
- Decrease_level (AANode*& node) – обновление параметра level в узле
- Min (int left, int right) – нахождение минимального из уровней

Библиотека Splay-tree имеет следующие приватные методы:

- Splay (Node* node) – поднятие элемента к корню
- SearchTree (Node* node, int value) – стандартный поиск
- SearchNode (Node* node, int value) – поиск с вызовом Splay
- RightRotate (Node* node) – правый поворот
- LeftRotate (Node* node) – левый поворот
- NormalInsert (int value) – обычная вставка
- DeleteNode (Node* node, int value) – удаление элемента
- Join (Node* s, Node* t) – соединение деревьев
- Split (Node* x, Node* &s, Node* &t) – разделение дерева на два поддерева
- Max (Node* node) – поиск максимального числа

Всего написано 877 строк кода на C++. Из них 430 – тесты, 204 – библиотека AA-tree и 243 – библиотека Splay-tree.

Тестирование состоит из 20 тестов, проверяющих алгоритм на корректность.

6. Исследование

Гипотеза: алгоритм Splay-tree будет заметно уступать алгоритму AA-tree при работе с возрастающей последовательностью.

Исследование производилось на различном количестве данных: от 10^3 до 10^7.

Ниже представлены графики, полученные в ходе исследования.

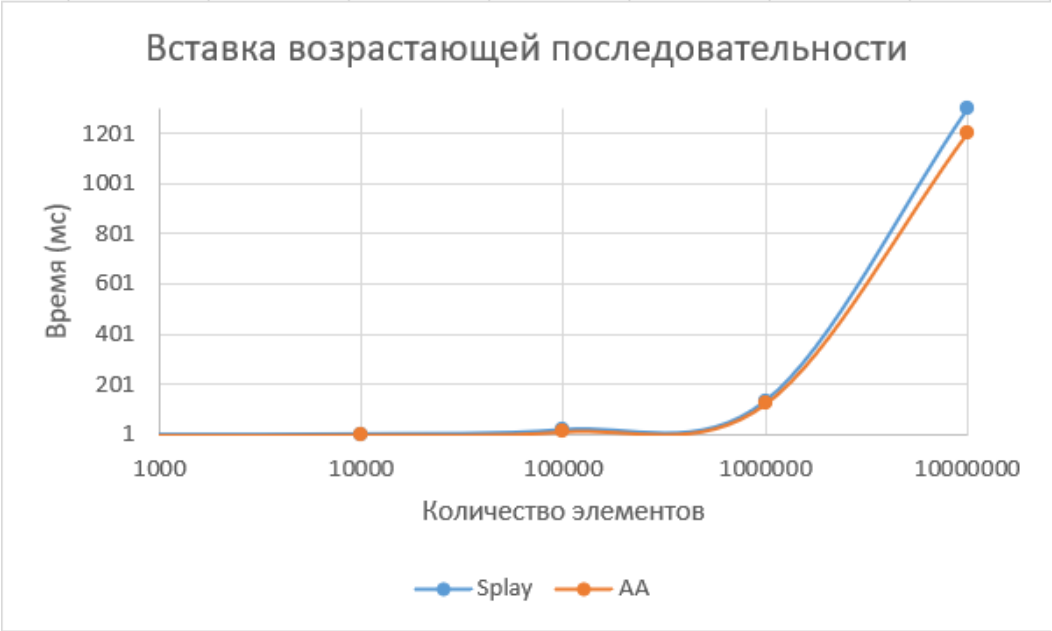

Рисунок 13. График вставки

В Splay-tree операция вставки занимает чуть больше времени, чем у AA-tree, так как Splay-tree перестраивается при каждом вызове операции вставки. В то же время AA-tree перестраивается только при вызове соответствующих функций.

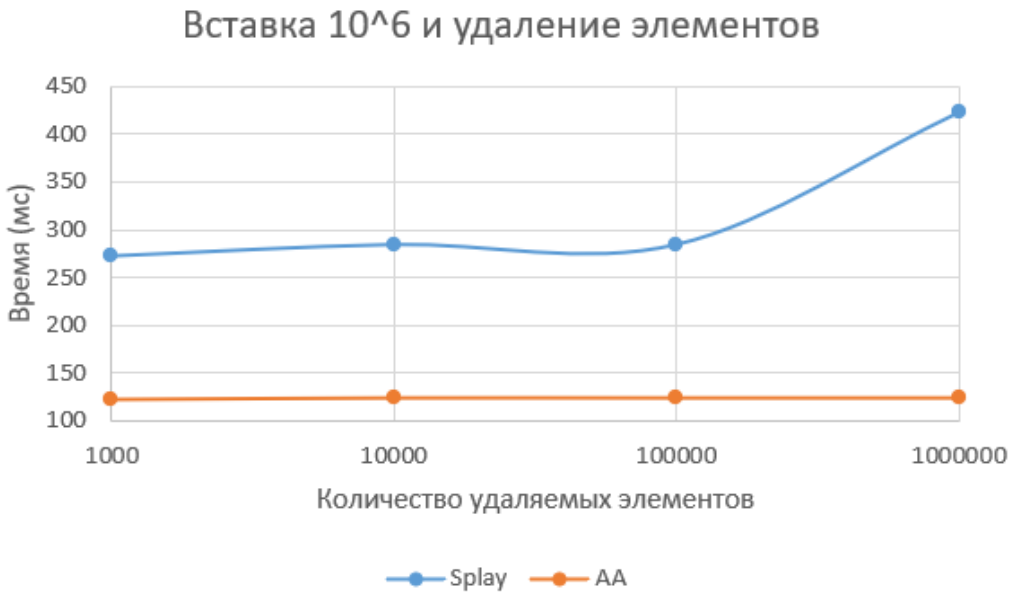

Рисунок 14. График вставки и удаления

По графикам видно, что Splay-tree тратит больше времени на выполнение данных операций, . Это доказывает, что при последовательных вставке и удалении Splay-tree проигрывает AA-tree из-за своих методов балансировки.

Заключение

В результате работы достигнуты следующие результаты:

1. Изучен материал по сбалансированным деревьям на примере AA-tree и Splay-tree
2. Изложены в форме научного доклада алгоритмы AA-tree и Splay-tree
3. Реализованы на языке C++ алгоритмы AA-tree и Splay-tree
4. Алгоритмы протестированы на 20 тестах
5. Проведено исследование с помощью 18 тестов
6. Проведено сравнительное исследование производительности алгоритмов AA-tree и Splay-tree
7. Результаты работы выложены на GitHub [27]

Список источников

1. https://www.youtube.com/watch?v=cILoJnFhGV0&t=342s&ab_channel=PavelMavrin
2. https://www.youtube.com/watch?v=xoRuox4Vh0s&ab_channel=Geekific
3. https://habr.com/ru/company/otus/blog/535316/
4. https://habr.com/ru/company/JetBrains-education/blog/210296/
5. https://neerc.ifmo.ru/wiki/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
6. https://ru.wikipedia.org/wiki/Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
7. https://www.youtube.com/watch?v=MoHHCiQnfuQ&ab_channel=%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5%D0%BD%D0%B0%D1%83%D0%BA%D0%B8
8. https://www.youtube.com/watch?v=almow4O2Cmg&ab_channel=%D0%9B%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B9%D0%A4%D0%9F%D0%9C%D0%98
9. https://wiki.algocode.ru/index.php?title=Splay-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
10. https://www.youtube.com/watch?v=Ex20GVEGf_s&ab_channel=%D0%94%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%D0%B7%D0%B0%D0%BD%D1%8F%D1%82%D0%B8%D1%8F%D0%9C%D0%A4%D0%A2%D0%98
11. https://www.youtube.com/watch?v=yldKfG_0rwU&ab_channel=%D0%9B%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%B9%D0%A4%D0%9F%D0%9C%D0%98
12. https://dic.academic.ru/dic.nsf/ruwiki/749023
13. https://ru.wikibrief.org/wiki/Splay_tree
14. https://www.cs.usfca.edu/~galles/visualization/SplayTree.html
15. https://en.wikipedia.org/wiki/AA_tree
16. https://habr.com/ru/post/110212/
17. https://www.geeksforgeeks.org/aa-trees-set-1-introduction/
18. https://neerc.ifmo.ru/wiki/index.php?title=AA-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
19. https://ppt-online.org/87041
20. https://studfile.net/preview/3571362/
21. https://cs.valdosta.edu/~dgibson/courses/cs3410/notes/ch19_6.pdf
22. https://www.youtube.com/watch?v=q1nvz79wEoM&ab_channel=PSClassesforCSE
23. https://www.youtube.com/watch?v=nTbD-36EA78&ab_channel=ExamPartner
24. http://proteus2001.narod.ru/gen/txt/8/aa.html
25. https://medium.com/@vitkarpov/cracking-the-coding-interview-4-2-9567d6986853
26. https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE-%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#:~:text=%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE%2D%D1%87%D1%91%D1%80%D0%BD%D0%BE%D0%B5%20%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%20(%D0%B0%D0%BD%D0%B3%D0%BB.,%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%2C%20%D1%83%D0%B4%D0%B0%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B8%20%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%20%D1%83%D0%B7%D0%BB%D0%B0.
27. https://github.com/Koshkaallmaznaya/algorithms-AA-tree-Splay-tree
28. https://medium.com/smucs/splay-tree-data-structure-558077e88f2f