Федеральное государственное автономное образовательное учреждение высшего образования

«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

ОТЧЕТ

по семестровой работе

“**Дерево АА**”

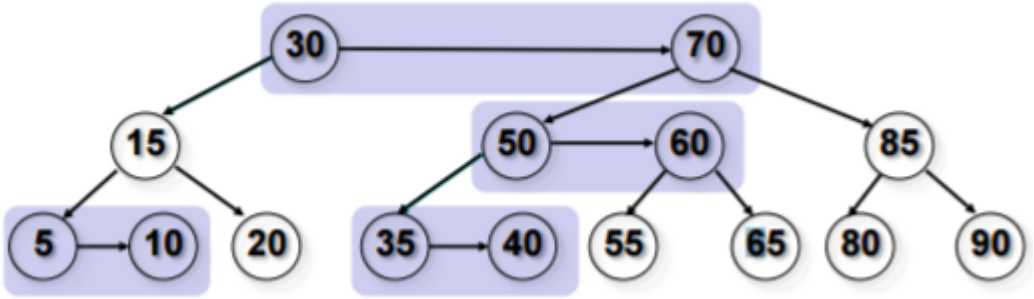
Группа № 11-002

Кузнецова Александра

Введение

- **Что это за структура данных?**
- **АА-дерево** (англ. *AA-Tree*) — структура данных, представляющая собой сбалансированное двоичное дерево поиска, которое является разновидностью красно-черного дерева с дополнительными ограничениями.
- АА-дерево названо по первым буквам имени и фамилии изобретателя, Арне Андерссона, который впервые предложил данную модификацию красно-черного дерева в 1993 году.

- **Как она представляется графически (схема)?**



-

- **Свойства дерева АА:**

- Уровень каждого листа равен 1

- 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.

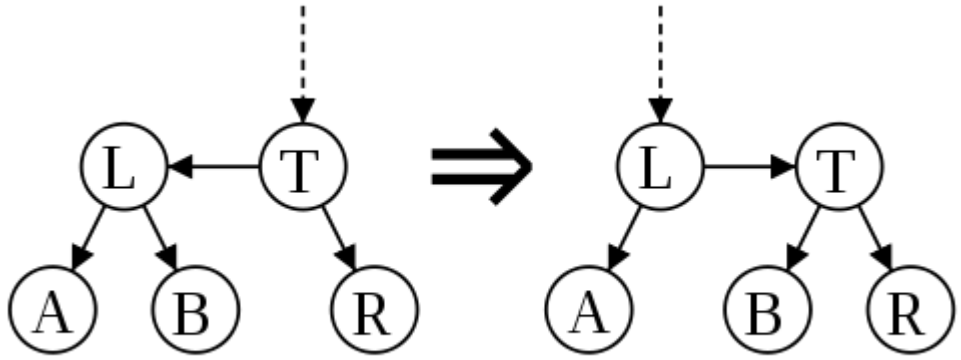
- **Связь с красно - чёрным деревом:**

В отличие от красно-черных деревьев, к одной вершине можно присоединить вершину только того же уровня, только одну и только справа (другими словами, красные вершины могут быть добавлены только в качестве правого ребенка).

Теоретический анализ структуры данных

- **История создания дерева:**
- АА-дерево было придумано Arne Andersson, который решил, что для упрощения балансировки дерева нужно ввести понятие **уровень** (*level*) 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 Arne Andersson приводит простое правило, которому должно удовлетворять АА-дерево:
-
- *К одной вершине можно присоединить другую вершину того же уровня но **только одну** и **только справа**.*
- **Эффективность дерева АА:**
- Оценка на высоту деревьев соответствует оценке для красно-черного дерева,
- $2 \cdot \log2(N)$
- , так как АА-дерево сохраняет структуру красно-черного дерева. Следовательно все операции происходят за
- $O(\log N)$
- , потому что в сбалансированном двоичном дереве поиска почти все операции реализуются за
- $O(h)$
- . Скорость работы АА-дерева эквивалентна скорости работы красно-черного дерева, но так как в реализации вместо цвета обычно хранят «уровень» вершины, дополнительные расходы по памяти достигают байта.

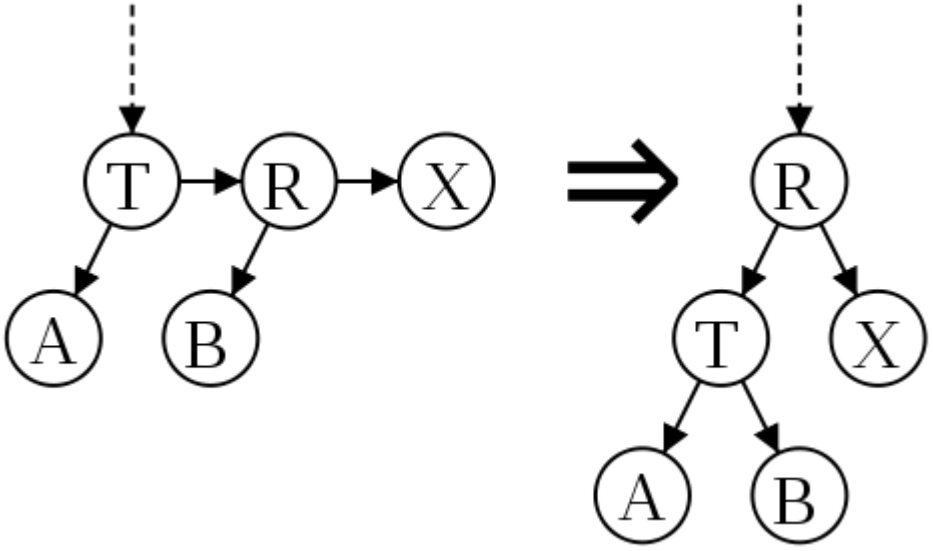
- ## Интересные сведения о дереве AA:
- фактически, это самое быстрое (или одно из самых быстрых) бинарное дерево с простой реализацией.
- **Проблемы, преимущества и особенности данной структуры данных:**
- Как известно, основная проблема бинарного дерева — его балансировка, то есть противодействие вырождению в обычный связный список. Обычно при балансировке дерева возникает много вариантов взаимного расположения вершин, каждый из которых нужно учесть в алгоритме отдельно.
- 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».
-
- Для балансировки AA-дерева нужно всего **2** (**две**) операции.
- Это операции названы в оригинальной работе *skew* и *split*, соответственно. На приведенных рисунках горизонтальная стрелка обозначает связь между вершинами одного уровня, а наклонная (вертикальная) — между вершинами разного уровня.
- ## Skew(t)
- — устранение левого горизонтального ребра. Делаем правое вращение, чтобы заменить поддерево, содержащее левую горизонтальную связь, на поддерево, содержащее разрешенную правую горизонтальную связь.
-



-

Split(t)

— 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

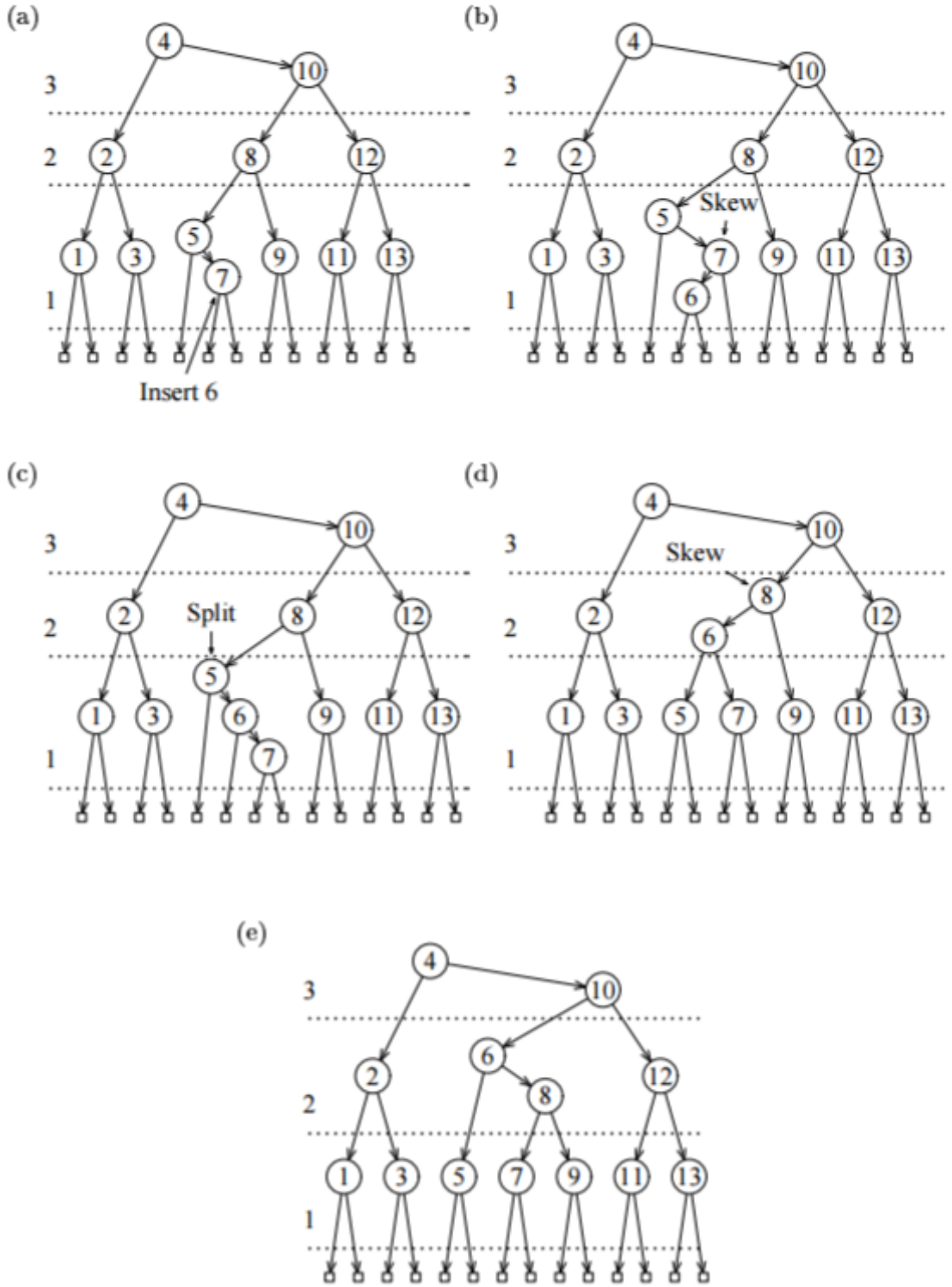


два под
дерева с меньшим уровнем.

Вставка элемента

Вставка нового элемента происходит как в обычном дереве поиска, только на пути вверх необходимо делать ребалансировку, используя skew и split. Ниже представлена рекурсивная реализация алгоритма.

Пример вставки нового элемента (на рис. уровни разделены горизонтальными линиями):



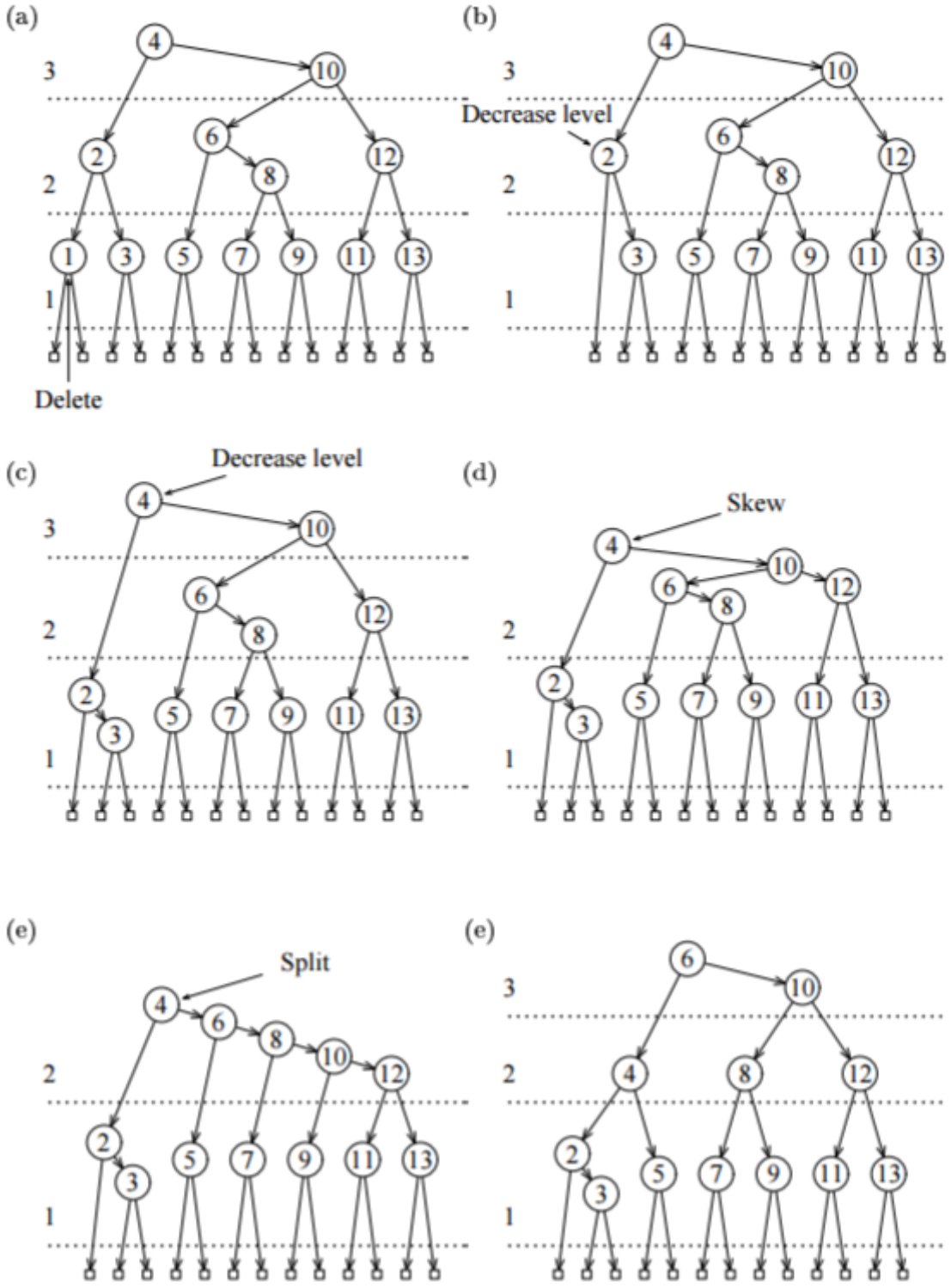
УДАЛЕНИЕ ВЕРШИНЫ:

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», в зависимости от реализации. «Предшественник» находится в начале последнего левого ребра, после которого идут

все правые ребра. По аналогии, «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, что делает их удаление тривиальным. Ниже представлена рекурсивная реализация алгоритма.

Будем использовать дополнительную функцию decreaseLevel , она будет обновлять уровень вершины, которую передают в функцию, в зависимости от значения уровня дочерних вершин.

Пример удаления вершины (на рис. уровни разделены горизонтальными линиями):


(a)
3
2
1
Delete
(b)
Decrease level
(c)
Decrease level
(d)
Skew
(e)
Split
(e)

Разработка проекта

Описание процесса работы над семестровым проектом:

- Работая над семестровым проектом, я планировала детально изучить структуру дерева АА.
- Я нашла и ознакомилась с информацией о данной структуре. Также я написала программу, реализующую работу дерева АА.Рассмотрела все особенности и интересные моменты относящиеся к работе данной структуры.

Генерация тестового набора данных

Для того, чтобы построить графики работы программы с различными входными данными на основе измерений времени, я придумала задачу: дан массив и нужно сказать сколько пар в нем дают в сумме число X.

Контрольные измерения

Коды для контрольных измерений:

1)

```
#pragma once

#include <bits/stdc++.h>
using namespace std;

struct AANode { //узел
	int level;//уровень
	int key;//ключ
	int value;
	AANode* left;//левый узел
	AANode* right;//правый узел
	AANode(int level_, int key_, int value_, AANode* left_, AANode* right_) {//конструктор
	level = level_;
	key = key_;
	value = value_;
	left = left_;
	right = right_;
	}
};

class AATree {//класс дерево
public:
	void Insert(int key, int value) {//вставлять
	Root = InternalInsert(Root, key, value);//корень,внутренняя вставка
	}

	void Delete(int key) {//удалить
	Root = InternalDelete(Root, key);//
```

```
    }

    void Print() {//печатать
    InternalPrint(Root);//
    std::cout << std::endl;
    }

    bool Search(int key) {//поиск
    int result = 0;
    return InternalSearch(Root, key, result);//внутренняя поиск
    }

    int Get(int key) {
    int result = 0;
    InternalSearch(Root, key, result);//внутренняя поиск
    return result;
    }

private:
    AANode* Split(AANode* node) {//узел
    if (node == nullptr || node->right == nullptr || node->right->right == nullptr) {
            return node;
    }
    if (node->level == node->right->right->level) {
            AANode* R_node = node->right;
            AANode* B_node = R_node->left;
            node->right = B_node;
            R_node->left = node;
            R_node->level++;
            return R_node;
    }
    return node;

    }

    AANode* Skew(AANode* node) {//скос
    if (node == nullptr || node->left == nullptr) {
            return node;
    }
    if (node->level == node->left->level) {
```

```
        AANode* L_node = node->left;
        AANode* B_node = L_node->right;
        L_node->right = node;
        node->left = B_node;
        return L_node;
}
return node;
}

AANode* InternalInsert(AANode* node, int key, int value) {//
if (node == nullptr) {
        AANode* result = new AANode(1, key, value, nullptr, nullptr);//
        return result;//
}
if (node->key > key) {
        node->left = InternalInsert(node->left, key, value);//
} else {
        node->right = InternalInsert(node->right, key, value);//
}
node = Split(node);//расщеплять
node = Skew(node);//скос
return node;
}

void InternalPrint(AANode* node) {//
if (node == nullptr) {
        return;
}
InternalPrint(node->left);//
std::cout << node->key << ", ";
InternalPrint(node->right);

}

void FixLevel(AANode* node) {//исправить уровень
if (node->left == nullptr || node->right == nullptr) {
        return;
}
int correct_level = std::min(node->left->level, node->right->level) + 1;//правильный уровень
if (correct_level < node->level) {
```

```
            node->level = correct_level;
            if (correct_level < node->right->level) {
            node->right->level = correct_level;
            }
}
}

AANode* InternalDelete(AANode* node, int key) {//
if (node == nullptr) {
            return node;
}
if (node->key > key) {
            node->left = InternalDelete(node->left, key);
} else if (node->key < key) {
            node->right = InternalDelete(node->right, key);
} else {
            if (node->left == nullptr && node->right == nullptr) {
            return nullptr;//
            }
            if (node->left == nullptr) {
            AANode* left_node = node->right;
            while (left_node->left) {
            left_node = left_node->left;
            }
            node->key = left_node->key;
            node->right = InternalDelete(node->right, left_node->key);
            } else {
            AANode* right_node = node->left;
            while (right_node->right) {
            right_node = right_node->right;
            }
            node->key = right_node->key;
            node->left = InternalDelete(node->left, right_node->key);//
            }
}
FixLevel(node);//
node = Skew(node);//
node->right = Skew(node->right);
if (node->right != nullptr) {//
            node->right->right = Skew(node->right->right);
```

```
        }
        node = Split(node);//
        node->right = Split(node->right);
        return node;//
        }

        bool InternalSearch(AANode* node, int key, int& value) {//
        if (node == nullptr) {
                return false;
        }
        if (node->key == key) {
                value = node->value;
                return true;
        }
        if (node->key > key) {
                return InternalSearch(node->left, key, value);//
        } else {
                return InternalSearch(node->right, key, value);//
        }
        }
        AANode* Root = nullptr;//
};
```

2)

```
#pragma once

#include <bits/stdc++.h>
using namespace std;

struct AANode { //узел
        int level;//уровень
        int key;//ключ
        int value;
        AANode* left;//левый узел
        AANode* right;//правый узел
        AANode(int level_, int key_, int value_, AANode* left_, AANode* right_) {//конструктор
        level = level_;
        key = key_;
        value = value_;
        left = left_;
        right = right_;
```

```
    }
};

class AATree {//класс дерево
public:
    void Insert(int key, int value) {//вставлять
    Root = InternalInsert(Root, key, value);//корень,внутренняя вставка
    }

    void Delete(int key) {//удалить
    Root = InternalDelete(Root, key);//
    }

    void Print() {//печатать
    InternalPrint(Root);//
    std::cout << std::endl;
    }

    bool Search(int key) {//поиск
    int result = 0;
    return InternalSearch(Root, key, result);//внутренняя поиск
    }

    int Get(int key) {
    int result = 0;
    InternalSearch(Root, key, result);//внутренняя поиск
    return result;
    }

private:
    AANode* Split(AANode* node) {//узел
    if (node == nullptr || node->right == nullptr || node->right->right == nullptr) {
            return node;
    }
    if (node->level == node->right->right->level) {
            AANode* R_node = node->right;
            AANode* B_node = R_node->left;
            node->right = B_node;
            R_node->left = node;
            R_node->level++;
```

```
        return R_node;
}
return node;

}

AANode* Skew(AANode* node) {//скос
if (node == nullptr || node->left == nullptr) {
        return node;
}
if (node->level == node->left->level) {
        AANode* L_node = node->left;
        AANode* B_node = L_node->right;
        L_node->right = node;
        node->left = B_node;
        return L_node;
}
return node;
}

AANode* InternalInsert(AANode* node, int key, int value) {//
if (node == nullptr) {
        AANode* result = new AANode(1, key, value, nullptr, nullptr);//
        return result;//
}
if (node->key > key) {
        node->left = InternalInsert(node->left, key, value);//
} else {
        node->right = InternalInsert(node->right, key, value);//
}
node = Split(node);//расщеплять
node = Skew(node);//скос
return node;
}

void InternalPrint(AANode* node) {//
if (node == nullptr) {
        return;
}
InternalPrint(node->left);//
```

```
std::cout << node->key << ", ";
InternalPrint(node->right);

}

void FixLevel(AANode* node) {//исправить уровень
if (node->left == nullptr || node->right == nullptr) {
        return;
}
int correct_level = std::min(node->left->level, node->right->level) + 1;//правильный уровень
if (correct_level < node->level) {
        node->level = correct_level;
        if (correct_level < node->right->level) {
        node->right->level = correct_level;
        }
}
}

AANode* InternalDelete(AANode* node, int key) {//
if (node == nullptr) {
        return node;
}
if (node->key > key) {
        node->left = InternalDelete(node->left, key);
} else if (node->key < key) {
        node->right = InternalDelete(node->right, key);
} else {
        if (node->left == nullptr && node->right == nullptr) {
        return nullptr;//
        }
        if (node->left == nullptr) {
        AANode* left_node = node->right;
        while (left_node->left) {
        left_node = left_node->left;
        }
        node->key = left_node->key;
        node->right = InternalDelete(node->right, left_node->key);
        } else {
        AANode* right_node = node->left;
        while (right_node->right) {
```

```
                right_node = right_node->right;
                }
                node->key = right_node->key;
                node->left = InternalDelete(node->left, right_node->key);//
                }
        }
        FixLevel(node);//
        node = Skew(node);//
        node->right = Skew(node->right);
        if (node->right != nullptr) {//
                node->right->right = Skew(node->right->right);
        }
        node = Split(node);//
        node->right = Split(node->right);
        return node;//
        }

        bool InternalSearch(AANode* node, int key, int& value) {//
        if (node == nullptr) {
                return false;
        }
        if (node->key == key) {
                value = node->value;
                return true;
        }
        if (node->key > key) {
                return InternalSearch(node->left, key, value);//
        } else {
                return InternalSearch(node->right, key, value);//
        }
        }
        AANode* Root = nullptr;//
};
```

Анализ результатов и выводы

n time

1331 0.852558

1730 1.15727

2249 1.51118

2923 2.04455

3799 2.80659

4938 3.6939

6419 4.88401

8344 6.70267

10847 9.35466

14101 11.6596

18331 15.992

23830 22.9324

30979 29.3777

40272 43.3465

52353 51.616

68058 73.1367

88475 99.8454

115017 127.156

149522 172.709

194378 274.123

252691 379.315

328498 467.043

427047 727.581

555161 1038.51

721709 1246.93

938221 1936.9

1219687 2419.68

1585593 3311.03

2061270 4723.2

2679651 7993.94

3483546 9485.89

4528609 15226.6

5887191 17242

7653348 24152.4

9949352 32591.2

12934157 44950.7

16814404 63935.5

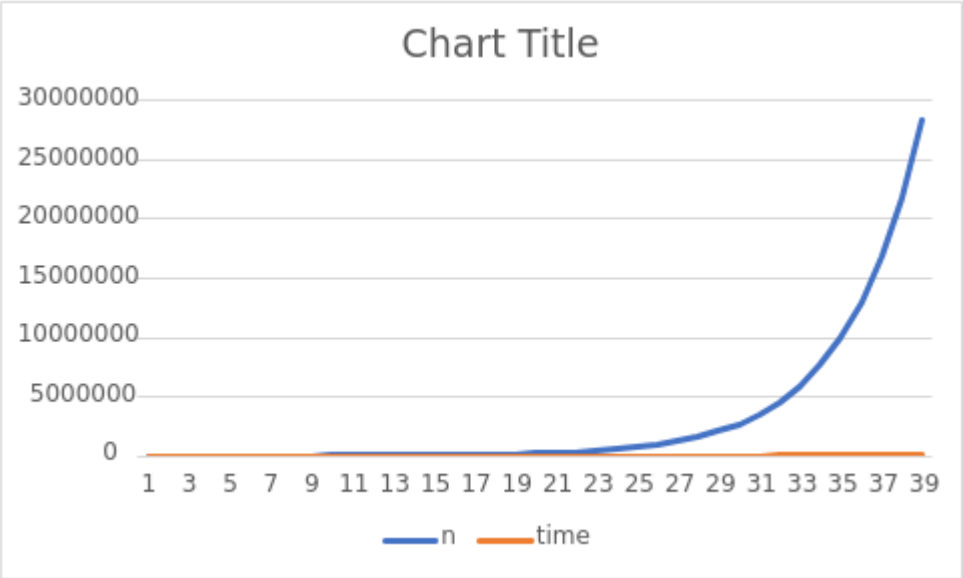
21858725 79452.4

28416342 136197

Такие результаты измерений у меня получились. 2 столбца: слева - введенные данные, справа - время работы программы.

Могу сделать вывод, что работа кода вполне удовлетворяет теоретической модели данных.

Также представляю для анализа графики работы моего кода:



Некоторые артефакты моей работы:

1. Ссылка на открытый репозиторий в GitHub: https://github.com/AlexandraKuznetsova1/SecondAisd
2. Ссылка на папку семестровой работы на Google Drive:https://docs.google.com/document/d/1EM4EopWstRb2B3T_50cRXhfm7mXXDXxZFbIhamwRxLs/edit#