МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Идеально сбалансированное БДП. Исследование.

Студентка гр. 9304 _____ Паутова Ю.В.

Преподаватель _____ Филатов Ар.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Паутова Ю.В.

Группа 9304

Тема работы: Идеально сбалансированное БДП. Исследование.

Исходные данные:

Написать программу на языке программирования C++, осуществляющую сравнение идеально сбалансированного БДП с простым БДП по высоте и по времени поиска элемента, а также с std::map по времени поиска.

Содержание пояснительной записки:

«Содержание»

«Введение»

«Описание кода программы»

«Исследование»

«Заключение»

«Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 24.12.2020

Дата защиты реферата: 29.12.2020

Студентка _____ Паутова Ю.В.

Преподаватель _____ Филатов Ар.Ю.

АННОТАЦИЯ

В данной курсовой работе была создана программа для исследования идеально сбалансированного БДП путём сравнения его с простым БДП и классом std::map (на основе красно-черного дерева). Для исследования рассматриваются средний и худший случаи построения простого БДП, то есть деревья строятся на основе случайной последовательности (средний случай) или на основе строго возратсающей последовательности (худший случай). На основе полученных данных строятся сравнительные графики высот деревьев или времени поиска.

SUMMARY

This course work is a program was created to study a ideally balanced BST by comparing it with a simple BST and the std::map class (based on a red-black tree). For the research the average and worst cases build a simple BST, that is the trees based on random sequences (average case) or on the basis of strict increasing sequence (worst case). Based on the obtained data, comparative graphs of tree heights or search time are constructed.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

Изучить такую структуру данных, как идеально сбалансированное БДП, разработать программный код для исследования данной структуры путём сравнения её с другими БДП.

Для достижения поставленной цели требуется решить следующие задачи:

1.Изучить такие структуры данных, как идеально сбалансированное БДП, БДП и std::map.

2.Разработать код для реализации задачи, которая задана в условии курсовой работы.

3.Собрать проект.

4.Протестировать программу.

1. ЗАДАЧА

Вариант 22

«Исследование» - генерация входных данных, использование их для измерения количественных характеристик структур данных, алгоритвом, действий, сравнение экспериментальных результатов с теоретическими.

Написать программу на языке программирования C++, реализующую идеально сбалансированное БДП и осуществляющую его сравнение с простым БДП по высоте и по времени поиска элемента, а также с std::map по времени поиска.

2. ОПИСАНИЕ КОДА ПРОГРАММЫ

2.1. Класс BinTree

Данный класс является реализацией идеально сбалансированного бинарного дерева поиска на языке программирования C++.

Класс содержит такие поля, как:

- ***Elem E*** — элемент для поиска.
- ***bool is_find*** — хранит информацию был найдет элемент или нет(true/false).
- ***int n*** — количество узлов в дереве.
- ***std::vector<Elem> sequence*** — последовательность, по которой строится дерево.
- ***std::shared_ptr<BinTreeNode<Elem>> head*** — указатель на корень дерева.

Методы:

- ***BinTree(std::vector<Elem>& sequence)*** — конструктор класса, который принимает ссылку на вектор, в котором хранится последовательность, по которой строится дерево. В нём вызывается метод make_tree(), в котором создается дерево.
- ***~BinTree()*** — деструктор класса.
- ***BinTree(BinTree&& other)*** — конструктор-перемещения.
- ***BinTree& operator=(BinTree&& other)*** — оператор-перемещения.
- ***BinTree(BinTree& other)*** — конструктор-копирования.
- ***BinTree& operator=(BinTree& other)*** — оператор-копирования.
- ***std::shared_ptr<BinTreeNode<Elem>> copy (std::shared_ptr<BinTreeNode<Elem>> cur)*** — копирует дерево по узлам.
- ***void make_tree(std::vector<Elem> sequence)*** — заполняет поля n и sequence, сортирует sequence, вызывает метод make_node(), который создает узлы дерева и связывает их указателями, в результате чего

получается реализация идеально сбалансированное БДП на базе указателей.

- ***std::shared_ptr<BinTreeNode<Elem>> make_node(int n)*** — создает узел дерева. Принимет количество узлов в последовательности, по которой строится левое и правое поддеревья: n делится пополам, берется целая часть от деления на 2 и получается новое количество узлов, которое передается для построения левого поддерева; элемент последовательности с индексом (n/2)+1 сохраняется в корень; n-(n/2)-1 новое количество узлов, которое передается для построения правого поддерева.
- ***void set_E(Elem E)*** — заполняет поле E.
- ***void find(Elem E)*** — вызывает метод search для поиска переданого элемента в дереве.
- ***void search(std::shared_ptr<BinTreeNode<Elem>>& cur)*** — осуществляет обход дерева и поиск заданного элемента.
- ***void print()*** — использует обход дерева в ширину и выводит дерево в консоль.
- ***std::shared_ptr<BinTreeNode<Elem>> get_head()*** — возвращает указатель на корень дерева.
- ***int height()*** — возвращает высоту дерева.
- ***bool empty()*** — возвращает true, если дерево пустое, иначе - false.
- ***void back_tracking(std::shared_ptr<BinTreeNode<Elem>>& cur)*** — осуществляет ЛКП обход дерева и восстанавливает последовательность, по которой было построено дерево.

2.2. Класс BST

Данный класс является реализацией бинарного дерева поиска на языке программирования C++.

Класс содержит такие поля, как:

- ***int E*** — элемент для поиска.

- ***bool is_find*** — хранит информацию был найдет элемент или нет(true/false).
- ***std::vector<int> sequence*** — последовательность, по которой строится дерево.
- ***std::shared_ptr<BinTreeNode<int>> head*** — указатель на корень дерева.

Методы:

- ***BST(std::vector<int>&)*** — конструктор класса, который принимает ссылку на вектор, в котором хранится последовательность, по которой строится дерево. В нём вызывается метод make_tree(), в котором создается дерево.
- ***~BST()*** — деструктор класса.
- ***BST(BST&& other)*** — конструктор-перемещения.
- ***BST& operator=(BST&& other)*** — оператор-перемещения.
- ***BST(BST& other)*** — конструктор-копирования.
- ***BST& operator=(BST& other)*** — оператор-копирования.
- ***std::shared_ptr<BinTreeNode<int>>copy (std::shared_ptr<BinTreeNode<int>>)*** — копирует дерево по узлам.
- ***void make_tree(std::vector<int> sequence)*** — заполняет поле sequence, в цикле, который пробегается по всей последовательности, вызывает метод insert(), который вставляет новый узел с переданым значением в дерево и связывает его с остальными указателями, в результате чего получается реализация БДП на базе указателей.
- ***void insert(int data_to_insert)*** — используется для вставки узла в дерево.
- ***void set_E(int E)*** — заполняет поле E.
- ***void find(int E)*** — вызывает метод search для поиска переданого элемента в дереве.
- ***void search(std::shared_ptr<BinTreeNode<int>>&)*** — осуществляет обход дерева и поиск заданного элемента.
- ***void printBST(std::shared_ptr<BinTreeNode<int>>&)*** — выводит скобочную запись дерева в консоль.

•***std::shared_ptr<BinTreeNode<int>>& get_head()*** — возвращает ссылку на корень дерева.

•***int height(std::shared_ptr<BinTreeNode<int>>&)*** — возвращает высоту дерева.

•***bool empty()*** — возвращает true, если дерево пустое, иначе - false.

•***void tracking(BST& tree)*** — осуществляет обход дерева в ширину.

2.3. Класс Research

Данный класс является реализацией задания курсовой работы: «Исследование» - генерация входных данных, использование их для измерения количественных характеристик структур данных, действий, сравнение экспериментальных результатов с теоретическими (происходит с помощью отображения результатов на графике).

Класс имеет такие методы, как:

•***void research_height(bool)*** — исследование высоты идеально сбалансированного БДП: метод принимает аргумент типа bool, который отвечает за случай. То есть если было передано true, то исследуется средний случай: открывается файл «res_ha.txt», в котторый будут сохранятся результаты вычисления высоты при заднном количестве элементов; затем открывается цикл for, пробегающий значения от 1 до 30; внутри цикла вызвается метод generate_randomseq(), создаются указатели на классы BinTree и BST и объекты этих классов с переданной им сгенерированной последовательбностью, вычисляется высота каждого дерева, результаты записываются в файл в формате: «<количество узлов в дереве> <высота идеально сбалансированного БДП> <высота БДП>»; после завершения цикла файл закрывается. Если было передано false, то открывается файл «res_hw.txt», а в цикле вызывается метод generate_sequence(), далее алгоритм соответствует первому случаю.

•***void research_time(bool)*** — исследование времени поиска в идеально сбалансированном БДП: метод принимает аргумент типа bool,

который отвечает за случай. То есть если было передано true, то исследуется средний случай: открывается файл «res_ta.txt», в котторый будут сохранятся результаты вычисления времени поиска при заднном количестве элементов; создается объект класа std::map: затем открывается цикл for, пробегающий значения от 1000 до 10000; внутри цикла вызвается метод generate_randomseq() и выбирается случайный элемент для поиска, создаются указатели на классы BinTree и BST и объекты этих классов с переданной им сгенерированной последовательбностью, также заполняется поле E каждого объекта, вычисляется время поиска по каждому дереву, std::map и время вычисления программой $\log_2 i$ с помощью библиотеки <chrono>, результаты записываются в файл в формате: «<количество узлов в дереве> <время поиска в идеально сбалансированного БДП> <время поиска в БДП> <время поиска в std::map> <время вычисления $\log_2 N$>»; после завершения цикла файл закрывается. Если было передано false, то открывается файл «res_tw.txt», а в цикле вызывается метод generate_sequence(), далее алгоритм соответствует первому случаю.

- ***std::vector<int> generate_sequence(int size)*** — создает и возвращает вектор размера size, в котором хранится строго возрастающая последовательность.
- ***std::vector<int> generate_randomseq(int size)*** — создает и возвращает вектор размера size, в котором хранится случайная последовательность.

2.4. Класс BinTreeNode

Данный класс является реализацией узла дерева.

Класс содержит такие поля, как:

- ***Elem data*** — значение, которое хранится в данном узле.
- ***std::shared_ptr<BinTreeNode<Elem>> left*** — указатель на левое поддерево.

- ***std::shared_ptr<BinTreeNode<Elem>> right*** — указатель на правое поддерево.

2.5. Функция main()

Если функции были переданы аргументы, то объект исследования берется мз второго аргумента, а случай из третьего (т. к. первый аргумент это название исполняемого файла), иначе объект исследования и случай вводятся пользователем по требования программы. Если что-то было передано или введено с ошибкой, выводится сообщение и просьба ввести заново.

После введения нужных данных создается объект класса Research и в зависимости от введённых данных вызываются соответствующие методы класса с конкретными аргументами и выводится сообщение-подсказка, в которой сообщается что нужно сделать после завершения программы, чтобы увидеть результаты исследования на графике.

3. ИССЛЕДОВАНИЕ

3.1. Высота дерева

В теории высота идеально сбалансированного БДП удовлетворяет условию:

$h \geq \log_2(N+1)$, где N — количество узлов в дереве.

Для вычисления высоты идеально сбалансированного БДП в программе была реализована рекурсивная функция: она производит обход дерева в глубину, начиная с корня. Если переданный ей узел дерева пустой, она возвращает 0. Если высота левого поддерева больше высоты правого, то она возвращает высоту левого поддерева +1, иначе возвращает высоту правого +1.

Так же высота идеально сбалансированного БДП сравнивается с высотой БДП, соответствующего той же последовательности, по которой было построено идеально сбалансированное. В среднем случае высота БДП:

$h \approx \log_2(N)$, где N — количество узлов в дереве.

В худшем случае (дерево строится по строго возрастающей последовательности) высота БДП:

$h = N$, где N — количество узлов в дереве.

Для нахождения высоты БДП была использована та же рекурсивная функция, что и для идеально сбалансированного БДП.

3.2. Время поиска

Время поиска элемента в идеально сбалансированном БДП в теории $O(\log_2(N))$, где N — количество узлов в дереве.

Для поиска элемента в идеально сбалансированном БДП была реализована рекурсивная функция: она производит обход дерева в глубину. Если переданный ей узел пустой, то она завершается, иначе проверяет равен ли искомый элемент значению, которое хранится в данном узле. Если значения равны, то полю is_find присваивается значение true и функция завершается. Если значения не равны, тогда мы переходим на следующий уровень дерева:

если искомое значение меньше значения в узле, то идем в левое поддерево, иначе в правое.

Для определения времени поиска используется библиотека <chrono>. Сначала создается переменная, в которую сохраняется время перед вызовом метода find(), затем происходит сам поиск элемента, после чего снова создается переменная, в которую сохраняется время после завершения поиска. В файл с резальтатами передается количество микросекунд, вычисленное как разность между временем конца и временем начала поиска.

Время поиска в идеально сбалансированном БДП сравнивается с временем поиска в БДП и временем поиска метода find() класса std::map (в основе красно-чёрное дерево). Время поиска у метода find() класса std::map так же, как и у идеально сбалансированного дерева, $O(\log_2(N))$, где N — количество узлов в дереве. У БДП в теории время поиска в среднем случае $O(\log_2(N))$, в худшем O(N).

3.3. Результаты исследования

Результаты измерений высоты и времени поиска сохраняются в соответствующие файлы.

Таблица 1 — Файлы с результатами измерений

Объект исследования	**Случай**	**Название файла**
Высота дерева	Средний	res_ha.txt
	Худший	res_hw.txt
Время поиска	Средний	res_ta.txt
	Худший	res_tw.txt

По данным, хранящимся в данных файлах, строятся соответствующие им графики. На рисунках 1-4 представлены графики всех четырех исследований.

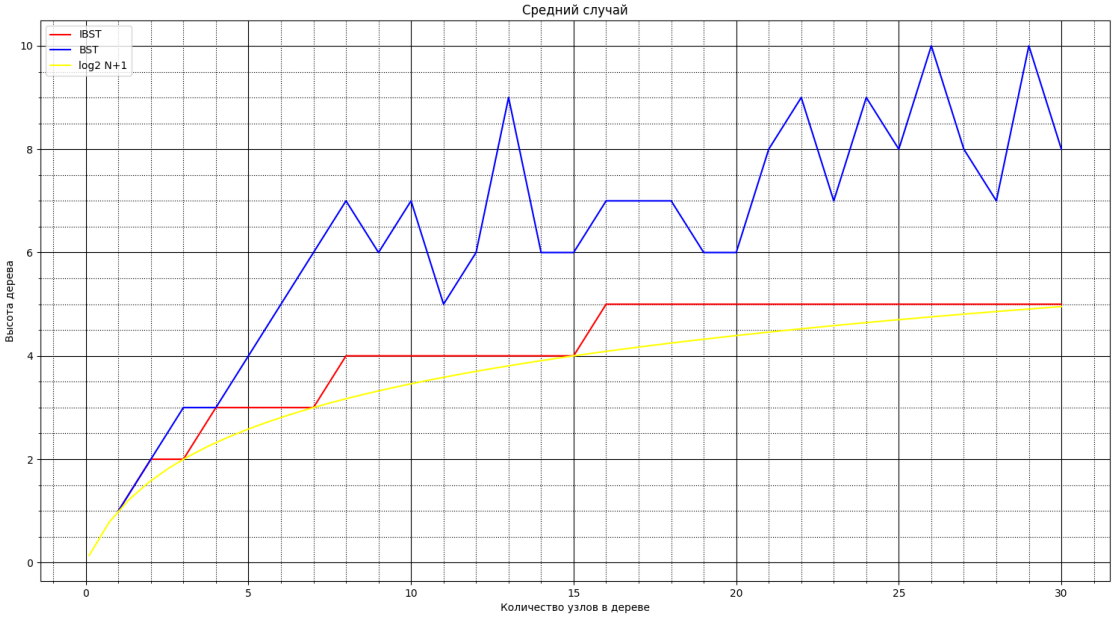

Рисунок 1 — Зависимость высоты дерева от количества узлов (средний случай)

Из графиков, представленных на рисунке 1: желтый - $\log_2(N+1)$, красный — высота идеально сбалансированного БДП, синий — высота БДП, видно, что высота идеально сбалансированного БДП на практике соответствует заявленой теоритической высоте $h \geq \log_2(N+1)$. Высота БДП отличается от $\log_2(N)$, но не критично. При малых N высота БДП соответствует высоте идеально сбалансированного БДП.

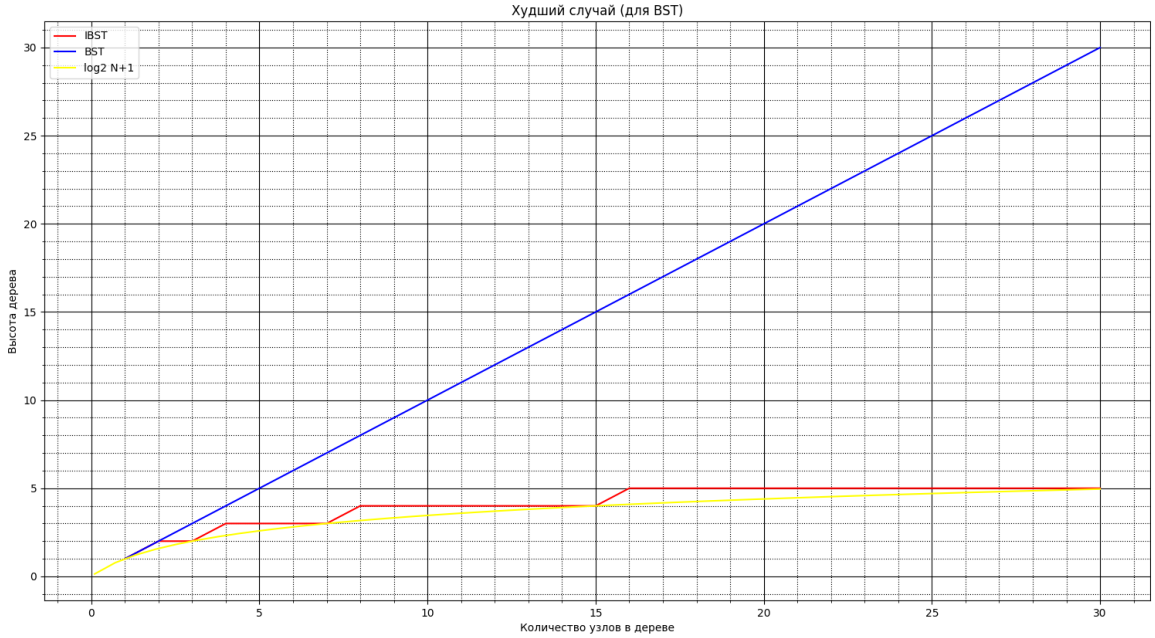

Рисунок 2 — Зависимость высоты дерева от количества узлов (худший случай)

Из графиков, представленных на рисунке 2: желтый - $\log_2(N+1)$, красный — высота идеально сбалансированного БДП, синий — высота БДП, видно, что высота идеально сбалансированного БДП на практике соответствует заявленой теоритической высоте $h \geq \log_2(N+1)$. Высота БДП на практике также соответствует заявленной в теории $h=N$.

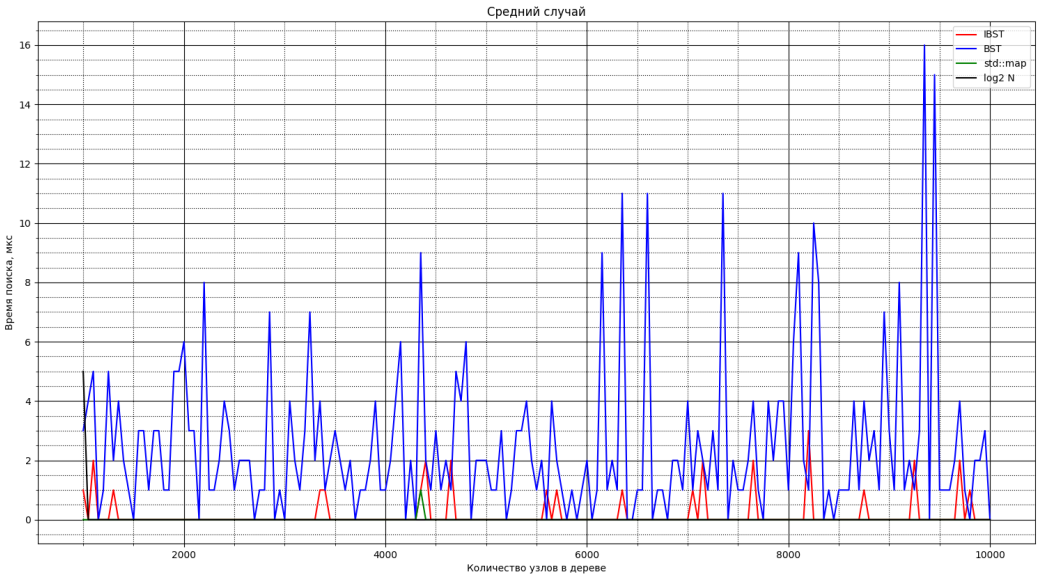

Рисунок 3 — Зависимость времени поиска от количества узлов (средний случай)

Из графиков, представленных на рисунке 3: черный - вычисление $\log_2(N+1)$, красный — время поиска в идеально сбалансированном БДП, синий — время поиска в БДП, зелёный — время поиска в std::map, видно, что время поиска в идеально сбалансированном БДП на практике соответствует заявленному в теории времени воиска, такому же как и у std::map, $O(\log_2(N))$. Так как в вычислении времени присутствует погрешность, на красном и зелёном графиках есть перепады. Время поиска в БДП больше времени поиска двух других объектов.

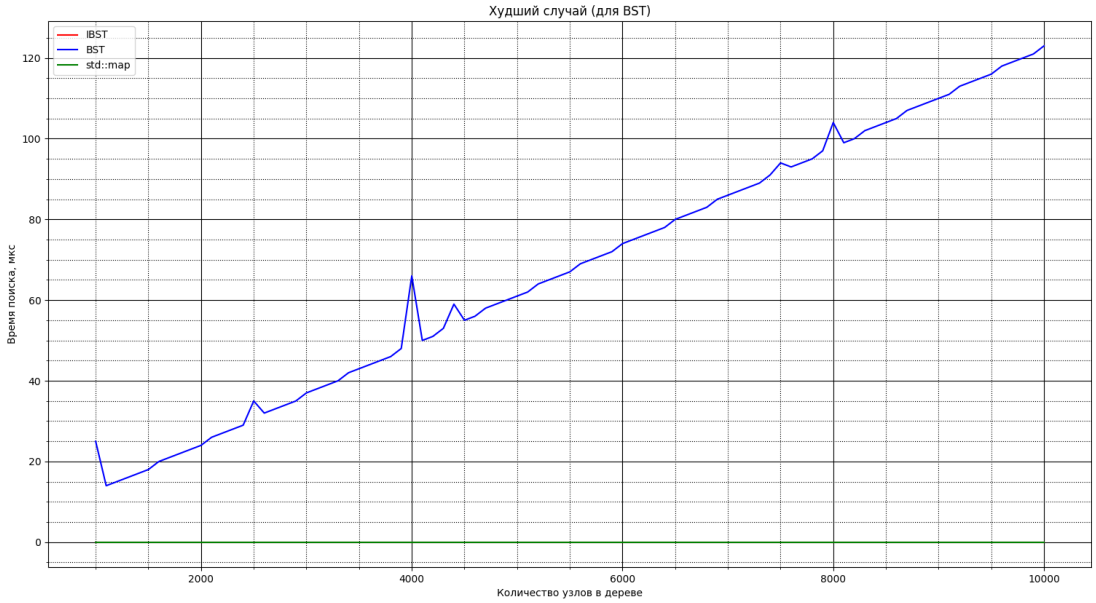

Рисунок 4 — Зависимость времени поиска от количества узлов (худший случай)

Из графиков, представленных на рисунке 4: красный — время поиска в идеально сбалансированном БДП, синий — время поиска в БДП, зелёный — время поиска в std::map, видно, что время поиска в идеально сбалансированном БДП на практике соответствует времени воиска в std::map. Так как время поиска в БДП больше времени поиска двух других объектов, то масштаб не позволяет точно отразить время поиска в идеально сбалансированном БДП и в std::map. Из графика зависимости времени поиска в БДП от количества узлов, видно, что оно соответствует теории, то есть O(N), так как график получился почти прямой (то есть прямая зависимость).

Из всего выше сказанного и графиков, представленных на рисунках 1-4, можно сделать вывод, что идеально сбалансированное дерево лучше подходит для хранения информации, так как его высота меньше высоты БДП, соответственно и время поиска по нему гораздо меньше, чем по простому БДП. Так же идеально сбалансированное БДП не имеет худшего случая в реализации в данной работе, так как его построение не зависит от переданной

последовательности, оно всегда строится по уже отсортированной последовательности.

4. ТЕСТИРОВАНИЕ

Ниже приведены примеры запуска программы:

```
julia@juliap:~/cw$ ./cw

What do you want to research?
If height, then enter "height" or "h". If search time, then enter "time" or "t"
Enter: height

What case do you want to research?
If average, then enter "average" or "a". If worst, then enter "worst" or "w"
Enter: average

To see a graph of research results, after the program finishes, enter the command:
        "make plot_height_average"

The program is finished

julia@juliap:~/cw$
```

Рисунок 5 — Запуск программы без аргументов командной строки

```
julia@juliap:~/cw$ ./cw

What do you want to research?
If height, then enter "height" or "h". If search time, then enter "time" or "t"
Enter: timr

The object of research is entered incorrectly
Re-enter: time

What case do you want to research?
If average, then enter "average" or "a". If worst, then enter "worst" or "w"
Enter: wort

The case of research is entered incorrectly
Re-enter: worst

To see a graph of research results, after the program finishes, enter the command:
        "make plot_time_worst"

The program is finished

julia@juliap:~/cw$
```

Рисунок 6 — Ошибочный ввод объекта и случай исследования

```
julia@juliap:~/cw$ ./cw height average

To see a graph of research results, after the program finishes, enter the command:
        "make plot_height_average"

The program is finished

julia@juliap:~/cw$
```

Рисунок 7 — Запуск программы с аргументами комнадной строки

ЗАКЛЮЧЕНИЕ

В ходе работы была изучена такая структура данных как идеально сбалансированное БДП. Была разработана программа на языке программирования C++, реализующая идеально сбалансированно БДП и простое БДП и осуществляющая исследование высоты и времени поиска путем сравнения. Идеально сбалансированное БДП оправдывает своё название: благодаря идеальной балансировке оно имеет наименьшую высоту из других БДП, что гарантирует быстрый поиск.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Справочник по C++ // cplusplus.com. URL: https://www.cplusplus.com/reference/

2. Бинарное дерево поиска // tproger.ru. URL: https://tproger.ru/translations/binary-search-tree-for-beginners/

3. map в C // codelessons.ru. URL: https://codelessons.ru/cplusplus/map-v-c-chto-eto-i-kak-s-etim-rabotat.html

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: IBST.h

```
#ifndef IBINTREE_H
#define IBINTREE_H

#include <string>
#include <sstream>
#include <vector>
#include <iostream>
#include <queue>
#include <algorithm>
#include <cmath>

#include "BinTreeNode.h"

template<typename Elem>
class BinTree{
public:

    BinTree(std::vector<Elem>& sequence){
        make_tree(sequence);
    }
    ~BinTree() = default;

    BinTree(BinTree&& other){
        std::swap(other.head, this->head);
    }
    BinTree& operator=(BinTree&& other){
        if (&other != this)
            this->head = std::move(other.head);
        return *this;
    }

    BinTree(BinTree& other){
        this->head = copy(other.head);
    }
    BinTree& operator=(BinTree& other){
        if (&other != this)
            this->head = copy(other.head);
        return *this;
    }
    std::shared_ptr<BinTreeNode<Elem>> copy(std::shared_ptr<BinTreeNode<E
lem>> cur){
        if (cur){
            std::shared_ptr<BinTreeNode<Elem>> node = std::make_shared<Bi
nTreeNode<Elem>>();
            node->left = copy(cur->left);
            node->right = copy(cur->right);
            node->data = cur->data;
            return node;
        }
        return nullptr;
    }
```

```
    void set_E(Elem E){
        this->E = E;
    }

    void find(Elem E){
        //this->E = E;
        search(this->head);
        //return is_find;
    }
    void search(std::shared_ptr<BinTreeNode<Elem>>& cur){
        if (cur){
            if (cur->data == this->E){
                this->is_find = true;
                return;
            }
            else{
                if (this→is_find){
                    return;
                }
                if(this->E < cur->data)
                    search(cur->left);
                else
                    search(cur->right);
            }
        }
    }

    void print(){ //обход в ширину, вывод на экран
        if (this->head){
            std::queue<std::shared_ptr<BinTreeNode<Elem>>> Q;
            std::stringstream lower_level;
            std::shared_ptr<BinTreeNode<Elem>> cur = std::make_shared<Bin
TreeNode<Elem>>();
            Q.push(this->head);
            int i = 1;
            int j = 0;
            int nodes_at_level = 0;
            int h = height();
            std::cout << '\n' << std::string(pow(2, h-i)+1, ' ');
            while(!Q.empty()){
                cur = Q.front();
                if (nodes_at_level == pow(2, i-1)){
                    i += 1;
                    j += 1;
                    std::cout << "\n\n";
                    if (i != h){
                        std::cout << std::string(pow(2, h-i), ' ');
                    }
                    nodes_at_level = 0;
                }
                if (i != h || h == 1){
                    std::cout << cur->data << std::string(pow(2, h-j)+1,
' ');
                    nodes_at_level += 1;
                }
                else{
                    std::cout << lower_level.str();
                    break;
```

```
                }

                if (cur->left){
                    Q.push(cur->left);
                    if(i == h-1){
                        lower_level << cur->left->data << "  ";
                    }
                }
                else{
                    if(i == h-1){
                        lower_level << "*  ";
                    }
                }
                if(cur->right){
                    Q.push(cur->right);
                    if(i == h-1){
                        lower_level << cur->right->data << "  ";
                    }
                }
                else{
                    if(i == h-1){
                        lower_level << "*  ";
                    }
                }
                Q.pop();
            }
        }
    }

    std::shared_ptr<BinTreeNode<Elem>>& get_head(){
        return this->head;
    }
    int height(std::shared_ptr<BinTreeNode<Elem>>& cur){
         if(curNode == nullptr){
            return 0;
        }
        if(height(curNode->left) > (height(curNode->right))){
            return height(curNode->left)+1;
        }
        return height(curNode->right)+1;
        //return ceil(log2(1 + n));
    }
    bool empty(){
        return !n;
    }

private:
    Elem E;
    bool is_find = false;
    int n; // количество узлов
    std::vector<Elem> sequence{};
    std::shared_ptr<BinTreeNode<Elem>> head;

    void make_tree(std::vector<Elem> sequence){
        this->sequence = sequence;
        this->n = sequence.size();
        std::sort(this->sequence.begin(), this->sequence.end());
        this->head = make_node(this->n);
```

```
    }

    std::shared_ptr<BinTreeNode<Elem>> make_node(int n){
        if (n == 0){
            return nullptr;
        }
        std::shared_ptr<BinTreeNode<Elem>> cur = std::make_shared<BinTree
Node<Elem>>();
        cur->left = make_node(n/2);
        cur->data = this->sequence.front();
        this->sequence.erase(this->sequence.begin());
        cur->right = make_node(n - (n/2) - 1);
        return cur;
    }

    void back_tracking(std::shared_ptr<BinTreeNode<Elem>>& cur){ //обход
ЛКП
        if (cur){
            back_tracking(cur->left);
            this->sequence.push_back(cur->data);
            back_tracking(cur->right);
        }
    }
};

#endif
```

Название файла: BST.h

```
#ifndef BINTREE_H
#define BINTREE_H

#include <string>
#include <vector>
#include <iostream>
#include <queue>

#include "BinTreeNode.h"

class BST{
public:
    BST(std::vector<int>&);
    ~BST() = default;

    BST(BST&& other);
    BST& operator=(BST&& other);

    BST(BST& other);
    BST& operator=(BST& other);
    std::shared_ptr<BinTreeNode<int>> copy(std::shared_ptr<BinTreeNode<in
t>>);

    void set_E(int E);

    void tracking(BST& cur);
    void printBST(std::shared_ptr<BinTreeNode<int>>&);
    void find(int E);
```

```
    void search(std::shared_ptr<BinTreeNode<int>>&);
    void insert(int data_to_insert);
    int height(std::shared_ptr<BinTreeNode<int>>&);
    std::shared_ptr<BinTreeNode<int>>& get_head();
    bool empty();

private:
    std::vector<int> sequence;
    int E;
    bool is_find;
    std::shared_ptr<BinTreeNode<int>> head;
    void make_tree(std::vector<int> tree);
};

#endif
```

Название файла: BST.cpp

```
#include "BST.h"

BST::BST(std::vector<int>& sequence){
    make_tree(sequence);
}

BST::BST(BST&& other){
    std::swap(other.head, this->head);
}
BST& BST::operator=(BST&& other){
    if (&other != this)
        this->head = std::move(other.head);
    return *this;
}

BST::BST(BST& other){
    this->head = copy(other.head);
}
BST& BST::operator=(BST& other){
    if (&other != this)
        this->head = copy(other.head);
    return *this;
}
std::shared_ptr<BinTreeNode<int>> BST::copy(std::shared_ptr<BinTreeNode<i
nt>> cur){
    if (cur){
        std::shared_ptr<BinTreeNode<int>> node = std::make_shared<BinTree
Node<int>>();
        node->left = copy(cur->left);
        node->right = copy(cur->right);
        node->data = cur->data;
        return node;
    }
    return nullptr;
}

void BST::set_E(int E){
    this->E = E;
}
```

```
void BST::make_tree(std::vector<int> tree){
    this->sequence = tree;
    while(!sequence.empty()){
        insert(sequence.front());
        sequence.erase(sequence.begin());
    }
}

void BST::printBST(std::shared_ptr<BinTreeNode<int>>& cur){
    std::cout << '(';
    if (cur){
        std::cout << cur->data;
        printBST(cur->left);
        printBST(cur->right);
    }
    std::cout << ')';
}

void BST::insert(int data_to_insert){
    if (!this->head){
        this->head = std::make_shared<BinTreeNode<int>>();
        this->head->data = data_to_insert;
        return;
    }
    std::shared_ptr<BinTreeNode<int>> node_to_insert {this->head};
    while(true){
        if(data_to_insert <= node_to_insert->data){
            if(!node_to_insert->left){
                node_to_insert->left = std::make_shared<BinTreeNode<int>>
();
                node_to_insert->left->data = data_to_insert;
                node_to_insert->left->left = node_to_insert->left->right
= nullptr;
                break;
            }
            else{
                node_to_insert = node_to_insert->left;
                continue;
            }
        }
        else{
            if(!node_to_insert->right){
                node_to_insert->right = std::make_shared<BinTreeNode<int>
>();
                node_to_insert->right->data = data_to_insert;
                node_to_insert->right->left = node_to_insert->right-
>right = nullptr;
                break;
            }
            else{
                node_to_insert = node_to_insert->right;
                continue;
            }
        }
    }
}
```

```
void BST::find(int E){
    //this->E = E;
    search(this->head);
    //return is_find;
}
void BST::search(std::shared_ptr<BinTreeNode<Elem>>& cur){
        if (cur){
            if (cur->data == this->E){
                this->is_find = true;
                return;
            }
            else{
                if (this→is_find){
                    return;
                }
                if(this->E < cur->data)
                    search(cur->left);
                else
                    search(cur->right);
            }
        }
}

int BST::height(std::shared_ptr<BinTreeNode<int>>& curNode){
        if(curNode == nullptr){
            return 0;
        }
        if(height(curNode->left) > (height(curNode->right))){
            return height(curNode->left)+1;
        }
        return height(curNode->right)+1;
}

std::shared_ptr<BinTreeNode<int>>& BST::get_head(){
    return this->head;
}

bool BST::empty(){
    return (!this->head);
}

void BST::tracking(BST& tree){
    if (tree.get_head()){
        std::queue<std::shared_ptr<BinTreeNode<int>>> Q;
        std::shared_ptr<BinTreeNode<int>> cur = std::make_shared<BinTreeN
ode<int>>();
        Q.push(tree.get_head());
        int path_length = 0;
        while(!Q.empty()){
            cur = Q.back();
            if (cur->left){
                Q.push(cur->left);
            }
            if(cur->right){
                Q.push(cur->right);
            }
            path_length++;
            Q.pop();
```

```
        }
    }
}
```

Название файла: BinTreeNode.h

```
#ifndef BINTREENODE_H
#define BINTREENODE_H

#include <memory>

template<typename Elem>
class BinTreeNode{
public:
    Elem data;
    std::shared_ptr<BinTreeNode<Elem>> left {nullptr};
    std::shared_ptr<BinTreeNode<Elem>> right {nullptr};
};

#endif
```

Название файла: Research.h

```
#ifndef RESEARCH_H
#define RESEARCH_H

#include <vector>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <ctime>
#include <chrono>
#include <map>

#include "IBST.h"
#include "BST.h"

class Research{
public:
    void research_height(bool);
    void research_time(bool);

private:
    std::vector<int> generate_sequence(int size);
    std::vector<int> generate_randomseq(int size);
};

#endif
```

Название файла: Research.cpp

```
#include "Research.h"

void Research::research_height(bool is_average){
    std::string filename;
```

```
    if (is_average){
        filename = "results/res_ha.txt";
    }
    else{
        filename = "results/res_hw.txt";
    }
    std::ofstream out(filename);
    if (!out.is_open()){
        std::cout << "not open\n";
        return;
    }
    std::vector<int> sequence{};
    for (int i = 1; i <= 30; i++){
        if (is_average){
            sequence = this->generate_randomseq(i);
        }
        else{
            sequence = this->generate_sequence(i);
        }

        std::shared_ptr<BinTree<int>>
ibst = std::make_shared<BinTree<int>>(sequence);
        std::shared_ptr<BST> bst = std::make_shared<BST>(sequence);

        out << sequence.size() << ' ' << ibst->height(ibst->get_head()) <
< ' ' << bst->height(bst->get_head()) << '\n';

        sequence.clear();
    }

    out.close();
}

void Research::research_time(bool is_average){
    srand(time(NULL));
    std::string filename;
    if (is_average){
        filename = "results/res_ta.txt";
    }
    else{
        filename = "results/res_tw.txt";
    }
    std::ofstream out(filename);
    if (!out.is_open()){
        std::cout << "not open\n";
        return;
    }
    std::vector<int> sequence{};
    int elem_to_find;
    std::map<int, int> map;

    for (int i = 1000; i <= 10000;){
        if (is_average){
            sequence = this->generate_randomseq(i);
            elem_to_find = sequence[rand()%(i/2)+100];
            i += 50;
        }
        else{
```

```
            sequence = this->generate_sequence(i);
            elem_to_find = sequence[i/2];
            i += 100;
        }

        std::shared_ptr<BinTree<int>> ibst = std::make_shared<BinTree<int
>>(sequence);
        ibst->set_E(elem_to_find);
        std::shared_ptr<BST> bst = std::make_shared<BST>(sequence);
        bst->set_E(elem_to_find);
        for (int i = 0; i < (int)sequence.size(); i++){
            map[i] = sequence[i];
        }

        auto start_find_in_ibst = std::chrono::steady_clock::now();
        ibst->find(elem_to_find);
        auto end_find_in_ibst = std::chrono::steady_clock::now();

        auto start_find_in_bst = std::chrono::steady_clock::now();
        bst->find(elem_to_find);
        auto end_find_in_bst = std::chrono::steady_clock::now();

        auto start_find_in_map = std::chrono::steady_clock::now();
        map.find(elem_to_find);
        auto end_find_in_map = std::chrono::steady_clock::now();

        auto start_log = std::chrono::steady_clock::now();
        log2(i);
        auto end_log = std::chrono::steady_clock::now();

        auto find_in_ibst_mks = std::chrono::duration_cast<std::chrono::m
icroseconds>(end_find_in_ibst-start_find_in_ibst);
        auto find_in_bst_mks = std::chrono::duration_cast<std::chrono::mi
croseconds>(end_find_in_bst-start_find_in_bst);
        auto find_in_map_mks = std::chrono::duration_cast<std::chrono::mi
croseconds>(end_find_in_map-start_find_in_map);
        auto log_mks = std::chrono::duration_cast<std::chrono::microsecon
ds>(end_log-start_log);

        out << sequence.size() << ' ';
        out << find_in_ibst_mks.count() << ' ';
        out << find_in_bst_mks.count() << ' ';
        out << find_in_map_mks.count() << ' ';
        out << log_mks.count() << '\n';

        sequence.clear();
        map.clear();
    }

    out.close();
}

std::vector<int> Research::generate_sequence(int size){
    std::vector<int> sequence{};
    for (int i = 1; i<= size; i++){
        sequence.push_back(i);
    }
    return sequence;
```

```
}

std::vector<int> Research::generate_randomseq(int size){
    srand(time(NULL));
    std::vector<int> random_sequence{};
    for (int i = 1; i<= size; i++){
        random_sequence.push_back(i);
    }
    int end_swap;
    if (size < 1000){
        end_swap = size;
    }
    else{
        end_swap = size/10;
    }
    for(int j = 0; j <= end_swap; j++){
        int first = rand()%size;
        int second = first + rand()%(size-first);
        std::swap(random_sequence[first], random_sequence[second]);
    }
    return random_sequence;
}
```

Название файла: main.cpp

```
#include <stack>

#include "Research.h"

bool is_correct_object(std::string& object){
     return (object == "height" || object == "time" || object == "t" ||
object == "h");
}

bool is_corretc_case(std::string& case_str){
     return (case_str == "average" || case_str == "worst" || case_str ==
"a" || case_str == "w");
}

int main(int argc, char** argv){

    std::string object_of_research;
    std::string what_case;
    std::string info_object("\nWhat do you want to research?\nIf height,
then enter \"height\" or \"h\". If search time, then enter \"time\"
or \"t\"\nEnter: ");
    std::string info_case("\nWhat case do you want to research?\nIf
average, then enter \"average\" or \"a\". If worst, then enter \"worst\"
or \"w\"\nEnter: ");
    std::string error_object("\nThe object of research is entered
incorrectly\nRe-enter: ");
    std::string error_case("\nThe case of research is entered
incorrectly\nRe-enter: ");

    if (argc == 1){
        std::cout << info_object;
        std::getline(std::cin, object_of_research);
```

```
        while(!is_correct_object(object_of_research)){
            std::cout << error_object;
            std::getline(std::cin, object_of_research);
        }
        std::cout << info_case;
        std::getline(std::cin, what_case);
        while(!is_corretc_case(what_case)){
            std::cout << error_case;
            std::getline(std::cin, what_case);
        }
    }
    else{
        object_of_research = argv[1];
        while(!is_correct_object(object_of_research)){
            std::cout << error_object;
            std::getline(std::cin, object_of_research);
        }
        what_case = argv[2];
        while(!is_corretc_case(what_case)){
            std::cout << error_case;
            std::getline(std::cin, what_case);
        }
    }
    std::cout << "\nTo see a graph of research results, after the program
finishes, enter the command:\n\t";

    Research research;
    if (object_of_research.find("h") != std::string::npos){
        if (what_case.find("a") != std::string::npos){
            std::cout << "\"make plot_height_average\"\n";
            research.research_height(true);
        }
        else{
            std::cout << "\"make plot_height_worst\"\n";
            research.research_height(false);
        }
    }
    else{
        if (what_case.find("a") != std::string::npos){
            std::cout << "\"make plot_time_average\"\n";
            research.research_time(true);
        }
        else{
            std::cout << "\"make plot_time_worst\"\n";
            research.research_time(false);
        }
    }

    std::cout << "\nThe program is finished\n" <<std::endl;
    return 0;
}
```

Название файла: plot.py

```
import sys
import matplotlib.pyplot as plt
import numpy as np
```

```
file = sys.argv[1]
f = open(file, 'r')
x, y, z, w, log = [], [], [], [], []

for l in f:
    row = l.split()
    if row != []:
        x.append(float(row[0]))
        y.append(float(row[1]))
        z.append(float(row[2]))
        if(file == "results/res_ta.txt" or file == "results/res_tw.txt"):
            w.append(float(row[3]))
            log.append(float(row[4]))

f.close()

t = np.linspace(0.1, max(x))

fig, ax = plt.subplots(figsize=(12,12))
ax.plot(x, y, label = 'IBST', color = 'red')
ax.plot(x,z, label = 'BST', color = 'blue')
if (file == "results/res_ha.txt" or file == "results/res_hw.txt"):
    ax.plot(t, np.log2(t+1), label = 'log2 N+1', color = 'yellow')
else:
    ax.plot(x, w, label = 'std::map', color = 'green')
    if (file == "results/res_ta.txt"):
        ax.plot(x, log, label = 'log2 N', color = 'k')

ax.grid(which='major',color='k')
ax.minorticks_on()
ax.grid(which='minor',color = 'k',linestyle = ':')

ax.legend()
ax.set_xlabel('Количество узлов в дереве')
if (file == "results/res_ha.txt" or file == "results/res_hw.txt"):
    ax.set_ylabel('Высота дерева')
else:
    ax.set_ylabel('Время поиска, мкс')

fig.set_figwidth(15)
fig.set_figheight(15)

if (file == "results/res_ha.txt" or file == "results/res_ta.txt"):
    plt.title("Средний случай")
else:
    plt.title("Худший случай (для BST)")
plt.show()
```

Название файла: Makefile

```
cw: main.o BST.o IBST.o Research.o
    g++ -std=c++17 main.o BST.o Research.o -o cw

main.o: main.cpp Research.h
    g++ -Wall -std=c++17 -c main.cpp
```

```
BST.o: BST.cpp BST.h BinTreeNode.h
    g++ -Wall -std=c++17 -c BST.cpp

IBST.o: IBST.h BinTreeNode.h
    g++ -Wall -std=c++17 -c IBST.h

Research.o: Research.cpp BST.h IBST.h
    g++ -Wall -std=c++17 -c Research.cpp

plot_height_average:
    python3 plot.py results/res_ha.txt

plot_height_worst:
    python3 plot.py results/res_hw.txt

plot_time_average:
    python3 plot.py results/res_ta.txt

plot_time_worst:
    python3 plot.py results/res_tw.txt

clean:
    rm -rf *.o *.gch cw results/*.txt
```