**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 сбалансированные БДП»**

**Студент гр. 8381** _____ **Переверзев Д.Е.**

**Преподаватель** _____ **Жангиров Т.Р.**

**Санкт-Петербург**
**2019**

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

Студент Переверзев Д.Е.

Группа 8381

Тема работы :Идеально сбалансированные БДП.

Исходные данные: исследование БДП, генерация входных данных, использование их для измерения количественных характеристик структур данных, алгоритмов, действий, сравнение экспериментальных результатов с теоретическими.

Содержание пояснительной записки:
«Содержание», «Введение», «Создание функций построения БДП», «Визуализация данных», «Исследование» «Заключение», «Список использованных источников», «Приложение».

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

| | | |
|---|---|---|
| Студент | | Переверзев Д.Е. |
| Преподаватель | | Жангиров Т.Р. |

## АННОТАЦИЯ

В ходе выполнения курсовой работы была написана программа, которая может обрабатывать входную строку, строить по ней идеальное бинарное дерево. Возможна генерация входных строк по заданный аргументам ( количество листьев в дереве и количество тестов, а также метод распределения :худший или средний). Сгенерированные данный записываются в базу данных. Также сделана визуализация данных и их сравнение по нескольким параметрам, а именно построение нескольких графиков с изображением зависимости разных параметров от количества шагов.

## SUMMARY

During the course of the course work, a program was written that can process the input string, build an ideal binary tree on it. It is possible to generate input strings by given arguments ( the number of leaves in the tree and the number of tests, as well as the distribution method :worst or average). The generated data is written to the database. Visualization of data and their comparison on several parameters, namely construction of several graphs with the image of dependence of different parameters on number of steps is also made.

## СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Цель работы: дать стабильную программу, производящую выбранную пользователем обработку данных в виде заданной или сгенерированной строки. Реализация программы должна содержать работу с динамически выделенной памятью и использование стандартных библиотек, также имеет графическую визуализацию и провести исследование тестов.

Исследование должно содержать:

- Анализ задачи, цели, технологию проведения и план экспериментального исследования.
- Генерацию представительного множества реализаций входных данных (с заданными особенностями распределения (для среднего и для худшего случаев)).
- Выполнение исследуемых алгоритмов на сгенерированных наборах данных. При этом в ходе вычислительного процесса фиксируется как характеристики (например, время) работы программы, так и количество произведенных базовых операций алгоритма.
- Фиксацию результатов испытаний алгоритма, накопление статистики.
- Представление результатов испытаний, их интерпретацию и сопоставление с теоретическими оценками.

## СОЗДАНИЕ ФНКЦИЙ ПОСТРОЕНИЯ БДП

### 1. Структура листа бинраного дерева

*struct TREE;*

Содержит указатели на левую и правую ветку, значение лиса и свой уровень глубины.

### 2. Функция вставки элемента в БДП

*TREE *insert(TREE *p, string val, int &steps);*

Функция опускается по бинарному дереву до тех пор, пока не найдет нужное место для вставки элемента, если необходимо сбалансировать дерево, вызывает функцию поворота для нужных ветвей.

### 3. Функция баланса

*TREE *balance(TREE *p, int &steps);*

Сравнивает глубину элементов ветки и балансирует дерево, если необходимо.

Использует функции левого и правого поворотов.

*TREE *rotateright(TREE *p, int &steps);*

*TREE *rotateleft(TREE *q, int &steps);*

### 4. Функция обхода БДП

*void bypass(TREE *&tree, string &bin_str);*

Функция обходит БДП и строит скобочную запить дерева, для дальнейшей визуализации.

### 5. Функция генерации рандомной строки

*string gen_random(int num);*

Генерирует строку содержащую псевдослучайные пятизначные числа в количестве *num.*

# ВИЗУАЛИЗАЦИЯ ДАННЫХ

## 1. Создание UI

Был создан WEB UI, сервер написан на *node.js* с использованиям аддонов написаных на c++ с использованием библиотеки *node.h.*

Клиентская часть нарисована разметкой HTML с использованием CSS для настройки стилей страниц и языком JavaScript для реализации действий и связей на странице. Также использован XMLHttpRequest , который предоставляет клиенту функциональность для обмена данными между клиентом и сервером.

## 2. Визуализация построения идеального БДП

Для визуализации БДП создавался объект представляющий дерево, с помощью настроенных под данный случай функций библиотеки *d3.mini.js* строится БДП. Также предусмотрены пошаговое построение дерева и чтения входных данных из файла

## 3. Генерация псевдослучайных данных

На стороне клиента имеется форма с полями необходимыми для заполнения, используя заданные значения, а именно:

- Количество листьев
- Количество тестов
- Способ распределения данных в входной строке

По заданным значениям происходит генерация данных и их тестирование, в базу данных записываются все сведения проведенных тестов, также в отдельные таблицу(зависит от способа распределения) записываются средние значения полученных данных.

## 4. Визуализация зависимостей

Для визуализации зависимостей клиент отправляет запрос на получение необходимых значений хранящихся в таблице на сервере, после получения данных строится таблица с ними на клиентской части, после этого возможно построение графиков зависимостей по заданным значениям.

# ТЕСТИРОВАНИЕ

## 1. Исследование сгенерированных данных

Сгенерировав несколько входных строк с разными заданными свойствами, были проведены тесты с полученными строками. Полученные данные были записаны в базу данных. Затем происходит отправка данных клиенту и строится таблица с ними:

Average_case* UPDATE

| | | | | |
|---|---|---|---|---|
| 13 | 20 | 120 | 40500 | 80 |
| 14 | 40 | 150 | 53200 | 102 |
| 15 | 40 | 200 | 74275 | 134 |
| 16 | 20 | 250 | 83350 | 165 |
| 17 | 20 | 300 | 107050 | 204 |
| 18 | 120 | 400 | 149158.328125 | 275 |

Worst_case* UPDATE

| | | | | |
|---|---|---|---|---|
| 13 | 20 | 120 | 34900 | 113 |
| 14 | 40 | 150 | 46525 | 142 |
| 15 | 40 | 200 | 59700 | 192 |
| 16 | 20 | 250 | 73550 | 242 |
| 17 | 20 | 300 | 90750 | 291 |
| 18 | 120 | 400 | 130483.3359375 | 391 |

Рисунок 1 — Таблицы с усреднёнными значениями

Получив данные, строим графики зависимостей:

1. Время строительства идеального БДП
2. Количество поворотов при строительстве идеального БДП

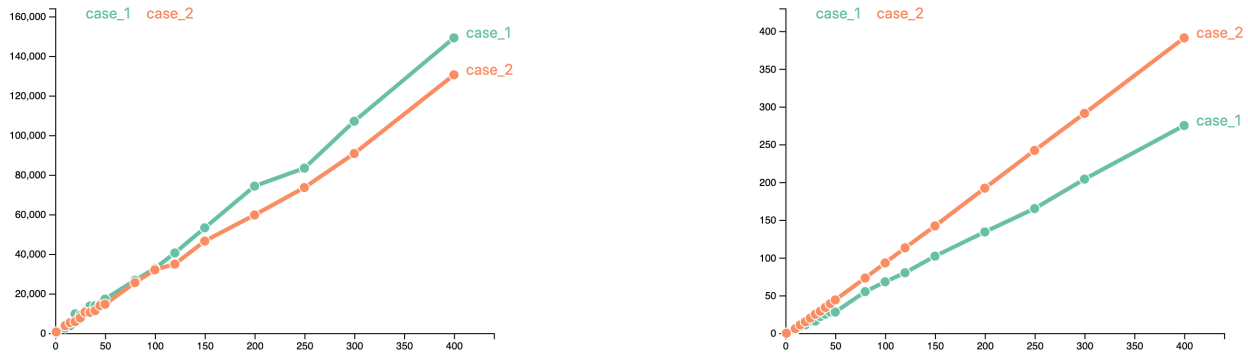


Рисунок 2 — Графики усредненных значений

Аналогично строится график зависимости времени от количества шагов для несбалансированного БДП:

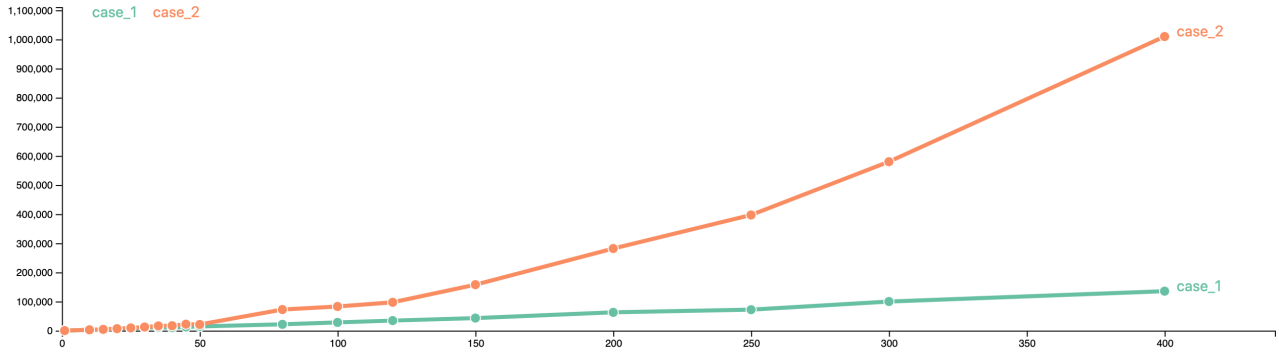


Рисунок 3 — График зависимости времени от количества шагов при строительстве несбалансированного БДП

## 2. Построение идеального БДП по пользовательским данным

Реализована возможность строить идеальное БДП по пользовательским данным, еще реализован пошаговый решим строительства идеального БДП.

При строительстве идеального БДП с пользовательской строкой:

«J g ft u h q sh h iu782 r4r7 w7247 f 7e bc874. B i743hjbre 73 jhbd7 wbh w. Ygf vfcj 98ru »

Получается такое бинарное дерево:

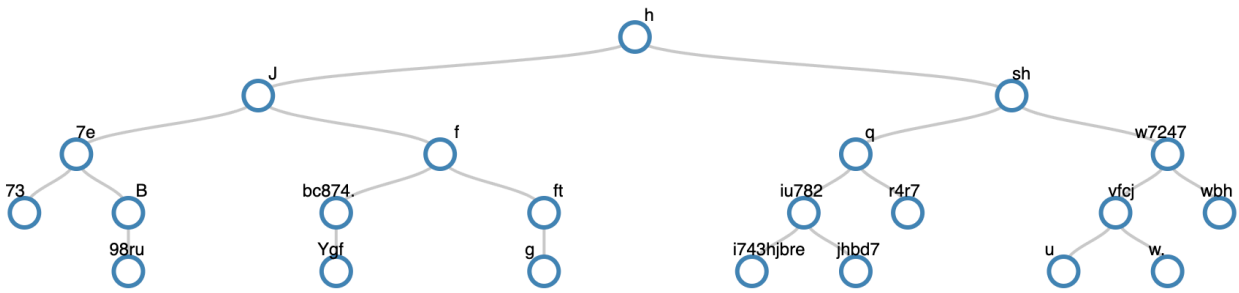


Рисунок 4 — Изображение идеального БДП

Количество листьев меньше чем количество элементов в строке, так как происходит проверка на лишние элементы и символы, если такие есть, то они удаляются из строки.

## ИССЛЕДОВАНИЕ

### 1. Генерацию представительного множества

Данные генерируются либо для среднего, либо для худшего значения, в генерируемой строке нет повторяющихся элементов для более точных тестов.

При выборе худшего распределения генерируется строка значений, чтобы при построении БДП представляло собой список, где каждый элемент имеет только одного ребенка.

Среднее распределение наоборот не допускает того, чтобы из входной строки получился список.

### 2. Фиксацию результатов испытаний

При исследовании БДП строится два бинарных дерева(сбалансированное и нет). При постройке идеального БДП используется функция балансировки (поворотов), для исследования алгоритм считает количество поворотов необходимых для строительства идеального БДП, также считается время строительства идеального БДП.

### 3. Анализ результатов испытаний

Исследования зависимости время строительства не сбалансированного БДП от количества элементов при двух способах распределения данных в входной строке:

- Распределения данных для достижения худшего результата строительства БДП, а именно построения списка:

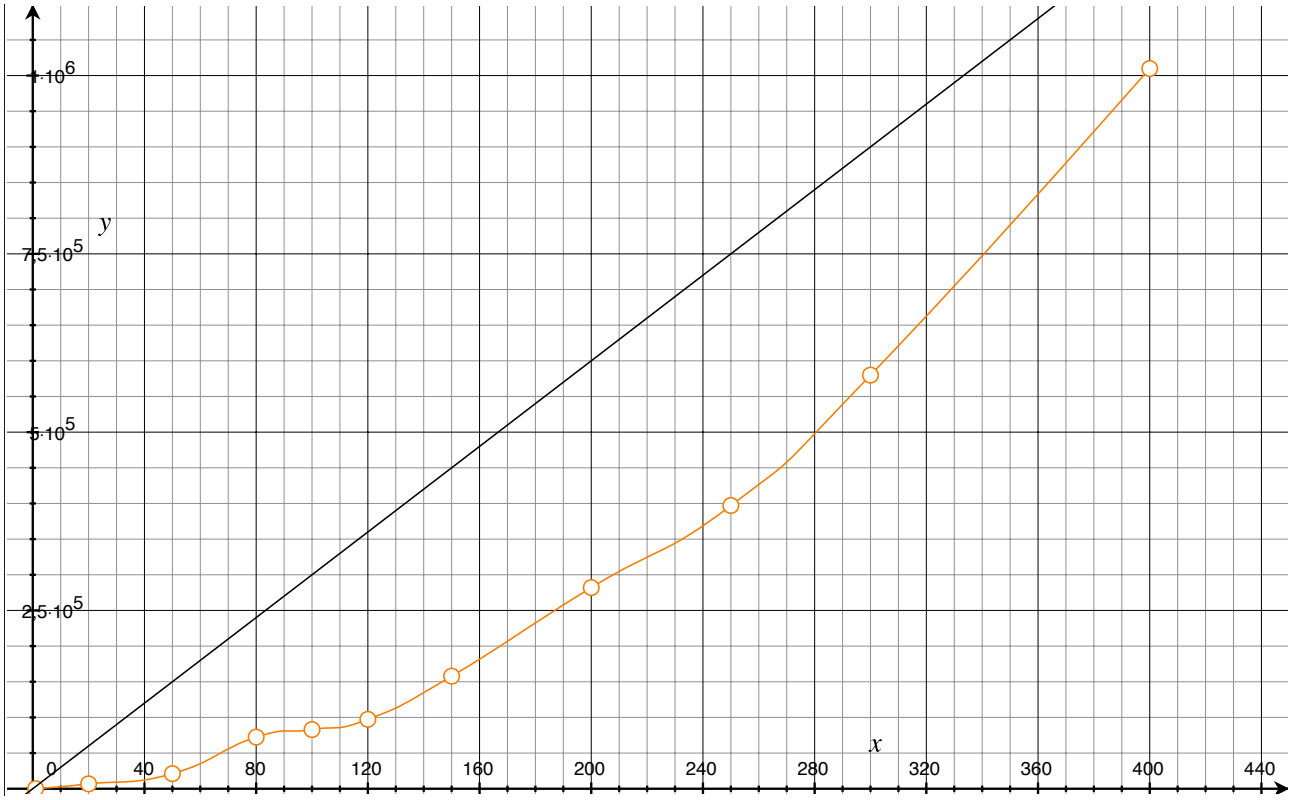


График 1 — сравнение графика времени строительства БДП и O(n)

• Распределения данных для достижения среднего результата строительства БДП, а именно построения списка:

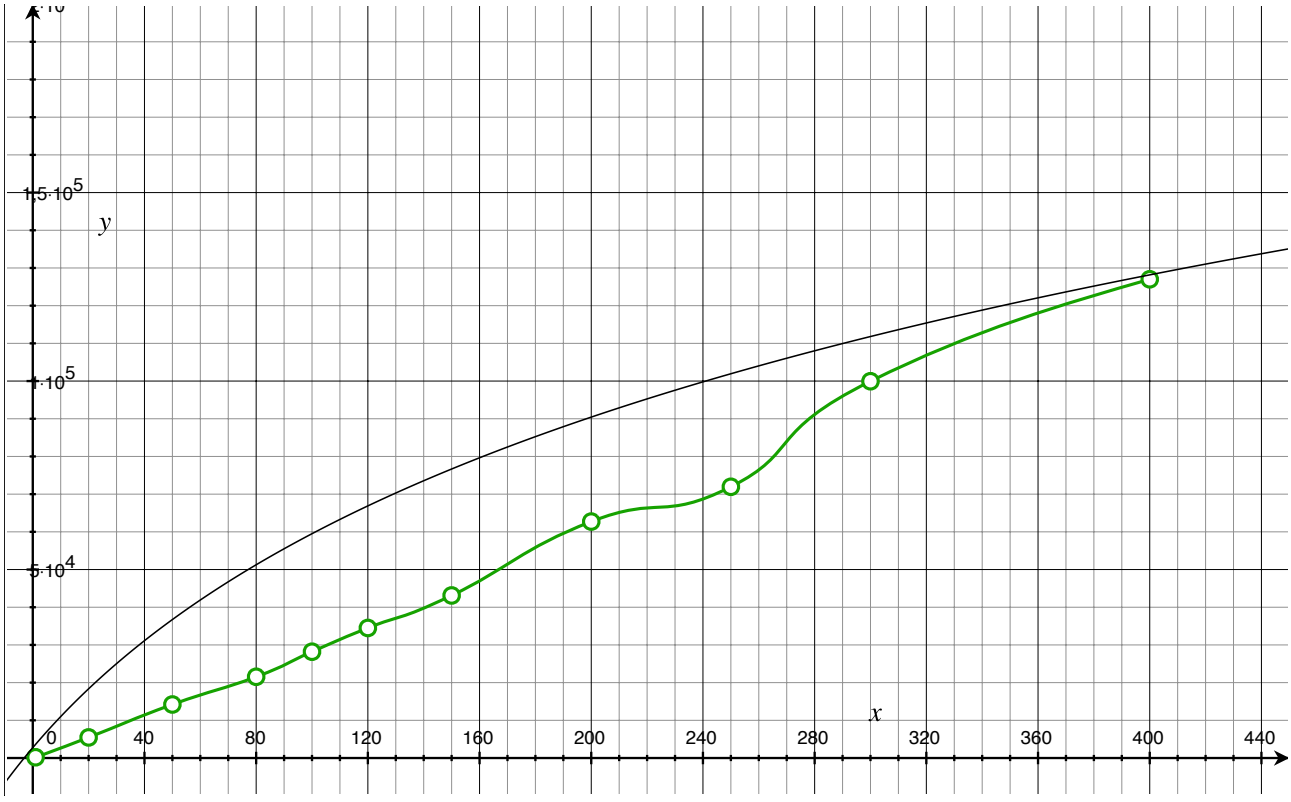


График 2 — сравнение графика времени строительства БДП и O(log n)

Исследования зависимости количество поворотов строительства сбалансированного БДП от количества элементов при двух способах распределения данных в входной строке:

- Оранжевый — худший случай
- Зеленый — средний случай

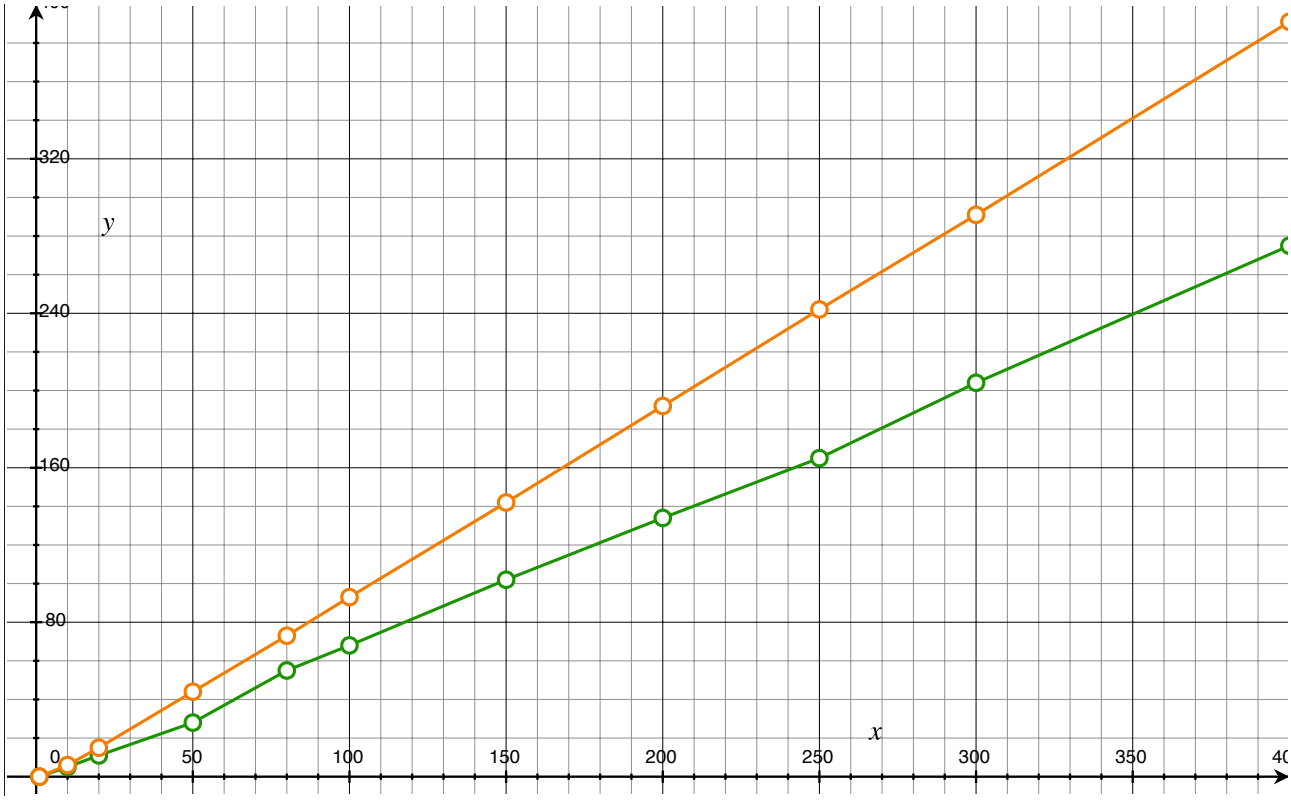


График 3 — сравнение количества поворотов при разных случаях

## ЗАКЛЮЧЕНИЕ

При исследование выполненных тестов можно заметить, что практические данные сопоставимы с теоретическими и почти не выбиваются за предел теоретически допустимых значений, время постройки БДП в среднем случае возрастает со скоростью O(log n), в худшем случае O(n). В сбалансированном БДП время строительства возрастает в линейной скоростью. Количество поворотов ветвей при построении идеального бинарного дерева поиска в худшем и среднем случаях тоже возрастает с линейной скорость, но при худшем случае количество поворотов возрастает быстрее, чем в среднем случае, что соответсвует теоретическим данным. Из всего этого следует что БДП строится правильно, так как полученные данные сопоставимы с теоретическими.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. https://habr.com/
2. https://neerc.ifmo.ru/
3. https://tproger.ru/translations/binary-search-tree-for-beginners/
4. https://nodejs.org/
5. https://rsdn.org/article/alg/bintree/avl.xml
6. https://www.ibm.com/developerworks/ru/
7. https://learn.javascript.ru/json

## ПРИЛОЖЕНИЕ А

**addon.cc**

```
// addon_3.cc
#include <node.h>
#include "cw/cw_methods.cpp"

namespace cw
{

using v8::Exception;
using v8::FunctionCallbackInfo;
using v8::Isolate;
using v8::Local;
using v8::Number;
using v8::Object;
using v8::String;
using v8::Value;

//cw

TREE *tree = NULL;
long long time_of_test = 0;   //milliseconds
long long time_of_test_2 = 0; //milliseconds
string elements_of_test = "";
int steps = 0;

void addon_build_tree(const FunctionCallbackInfo<Value> &args)
{
   Isolate *isolate = args.GetIsolate();
   Local<String> arg0 = Local<String>::Cast(args[0]);
   Local<Number> arg1 = Local<Number>::Cast(args[1]);
```

```
        String::Utf8Value expr(arg0);

        string tree_str = "(",
              value;
        int count = (int)(arg1->NumberValue());
        string input = string(*expr);
        istringstream in(input);

        //tree = NULL;

        //for time
        chrono::time_point<chrono::system_clock> start;

        for (; count > 0; count--)
        {
           in >> value;

           //turns
           elements_of_test += value + " "; //elems for data
           start = chrono::system_clock::now();
           tree = insert(tree, value, steps);
                                                    time_of_test +=
chrono::duration_cast<chrono::microseconds>(chrono::system_clock::now() -
start).count(); //time for data
        }

        bypass(tree, tree_str);

        Local<String> RET = String::NewFromUtf8(isolate, tree_str.c_str());
        args.GetReturnValue().Set(RET);
    }
```

```
void addon_take_time(const FunctionCallbackInfo<Value> &args) //send time_of_test
{
  Isolate *isolate = args.GetIsolate();

  Local<Number> RET = Number::New(isolate, time_of_test);
        // Local<String> RET = String::NewFromUtf8(isolate, to_string(time_of_test).c_str());
  args.GetReturnValue().Set(RET);
}

void addon_take_time_2(const FunctionCallbackInfo<Value> &args) //send time_of_test
{
  Isolate *isolate = args.GetIsolate();

  Local<Number> RET = Number::New(isolate, time_of_test_2);
        // Local<String> RET = String::NewFromUtf8(isolate, to_string(time_of_test).c_str());
  args.GetReturnValue().Set(RET);
}

void addon_take_elems(const FunctionCallbackInfo<Value> &args) //send elements_of_test
{
  Isolate *isolate = args.GetIsolate();
        Local<String> RET = String::NewFromUtf8(isolate, elements_of_test.c_str());
  args.GetReturnValue().Set(RET);
}
```

```
void addon_zeroing(const FunctionCallbackInfo<Value> &args) //zeroing data
{
  Isolate *isolate = args.GetIsolate();

  tree = NULL;
  time_of_test = 0; //milliseconds
  time_of_test_2 = 0;
  elements_of_test = "";
  steps = 0;

        Local<String> RET = String::NewFromUtf8(isolate,
to_string(time_of_test).c_str());
  args.GetReturnValue().Set(RET);
}

void addon_data_test(const FunctionCallbackInfo<Value> &args)
{
  Isolate *isolate = args.GetIsolate();
  Local<String> arg0 = Local<String>::Cast(args[0]);
  Local<Number> arg1 = Local<Number>::Cast(args[1]);
  String::Utf8Value expr(arg0);

  TREE *tree_1 = NULL;
  TREE *tree_2 = NULL;
  int count = (int)(arg1->NumberValue());
  string value;
  elements_of_test = string(*expr);
  istringstream in(elements_of_test);

  chrono::time_point<chrono::system_clock> start;
```

```
            for (; count > 0; count--)
            {
               in >> value;

               //turns;
               start = chrono::system_clock::now();
               tree_1 = insert(tree_1, value, steps);
                                                    time_of_test  +=
chrono::duration_cast<chrono::nanoseconds>(chrono::system_clock::now() -
start).count(); //time for data

               //without turns
               start = chrono::system_clock::now();
               tree_2 = insert_2(tree_2, value);
                                                    time_of_test_2  +=
chrono::duration_cast<chrono::nanoseconds>(chrono::system_clock::now() -
start).count(); //time for data
            }

            Local<String> RET = String::NewFromUtf8(isolate, "DONE");
            args.GetReturnValue().Set(RET);
         }

         void addon_data_random(const FunctionCallbackInfo<Value> &args)
         {
            Isolate *isolate = args.GetIsolate();
            Local<Number> arg0 = Local<Number>::Cast(args[0]);
            Local<String> RET = String::NewFromUtf8(isolate, gen_random((int)(arg0-
>NumberValue())).c_str());
            args.GetReturnValue().Set(RET);
```

```
}

void addon_take_steps(const FunctionCallbackInfo<Value> &args)
{
   Isolate *isolate = args.GetIsolate();

   Local<Number> RET = Number::New(isolate, steps);
   args.GetReturnValue().Set(RET);
}

void Init(Local<Object> exports)
{
   //cw
   NODE_SET_METHOD(exports, "build_tree", addon_build_tree);
   NODE_SET_METHOD(exports, "take_time", addon_take_time);
   NODE_SET_METHOD(exports, "take_time_2", addon_take_time_2);
   NODE_SET_METHOD(exports, "take_steps", addon_take_steps);
   NODE_SET_METHOD(exports, "take_elems", addon_take_elems);
   NODE_SET_METHOD(exports, "zeroing", addon_zeroing);
   NODE_SET_METHOD(exports, "data_test", addon_data_test);
   NODE_SET_METHOD(exports, "data_rand", addon_data_random);
}

NODE_MODULE(addon, Init)

} // namespace cw
```

**methods.cpp**

```
#include "cw_methods.h"
struct TREE
{
   string value;
   unsigned int height;
   TREE *left;
   TREE *right;
   TREE(string val)
   {
      value = val;
      left = right = 0;
      height = 1;
   }
};

void bypass(TREE *&tree, string &bin_str)
{
   bin_str += tree->value;
   if (tree->left)
   {
      bin_str += "(";
      bypass(tree->left, bin_str);
   }

   if (tree->right)
   {
      bin_str += "(";
      bypass(tree->right, bin_str);
   }
   bin_str += ")";
```

```
}

unsigned char height(TREE *p)
{
   return p ? p->height : 0;
}

int bfactor(TREE *p)
{
   return height(p->right) - height(p->left);
}

void fixheight(TREE *p)
{
   unsigned char hl = height(p->left);
   unsigned char hr = height(p->right);
   p->height = (hl > hr ? hl : hr) + 1;
}

TREE *rotateright(TREE *p, int &steps) // правый поворот вокруг p
{
   steps++;
   TREE *q = p->left;
   p->left = q->right;
   q->right = p;
   fixheight(p);
   fixheight(q);
   return q;
}

TREE *rotateleft(TREE *q, int &steps) // левый поворот вокруг q
```

```
{

   steps++;
   TREE *p = q->right;
   q->right = p->left;
   p->left = q;
   fixheight(q);
   fixheight(p);
   return p;
}

TREE *balance(TREE *p, int &steps) // балансировка узла p
{
   fixheight(p);
   if (bfactor(p) >= 2)
   {
      if (bfactor(p->right) < 0)
         p->right = rotateright(p->right, steps);
      return rotateleft(p, steps);
   }
   if (bfactor(p) <= -2)
   {
      if (bfactor(p->left) > 0)
         p->left = rotateleft(p->left, steps);
      return rotateright(p, steps);
   }
   return p; // балансировка не нужна
}
```

TREE *insert(TREE *p, string val, int &steps) // вставка ключа val в дерево с корнем p

```
{
    if (!p)
        return new TREE(val);
    if (strcmp(val.c_str(), p->value.c_str()) < 0)
        p->left = insert(p->left, val, steps);
    else
        p->right = insert(p->right, val, steps);
    return balance(p, steps);
}

TREE *insert_2(TREE *p, string val) // вставка ключа val в дерево с корнем
p
{
    if (!p)
        return new TREE(val);
    if (strcmp(val.c_str(), p->value.c_str()) < 0)
        p->left = insert_2(p->left, val);
    else
        p->right = insert_2(p->right, val);
    return p;
}

string gen_random(int num)
{


srand(chrono::duration_cast<chrono::nanoseconds>(chrono::high_resolution_clock::
now().time_since_epoch()).count());
    string rand_data = "";
    int a;
    for (int i = 0; i < num; i++)
```

```
    {
        do
        {
            a = rand() % 89999 + 10000;
        } while (strstr(rand_data.c_str(), to_string(a).c_str()) != NULL);
        rand_data += to_string(a) + " ";
    }
    return rand_data;
}
```

**methods.h**

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <sstream>
#include <chrono>
#include <random>

using namespace std;

struct TREE;

void bypass(TREE *&tree, string &bin_str);
unsigned char height(TREE *p);
int bfactor(TREE *p);
void fixheight(TREE *p);
TREE *rotateright(TREE *p, int &steps);        // правый поворот вокруг p
TREE *rotateleft(TREE *q, int &steps);         // левый поворот вокруг q
TREE *balance(TREE *p, int &steps);            // балансировка узла p
TREE *insert(TREE *p, string val, int &steps); // вставка ключа val в дерево
```

с корнем p

```
TREE *insert_2(TREE *p, string val);          // вставка ключа val в дерево с
корнем p
string gen_random(int num);
```

**index.js**

```
var express = require('express');
var app = express();
var mysql = require('mysql');
var path = require('path');
const cw = require('./build/Release/addon');



app.use(express.static(path.join(__dirname, '/front')));
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

//connect to port 3000
app.listen(3000, function () {
   console.log('Serever startd on port 3000!');
});

//conneect to mysql db
var conn = mysql.createConnection({
   host: 'localhost',
   user: 'alpha',
   password: 'tiger007',
   database: 'cw_db',
   multipleStatements: true
});
```

```
conn.connect((err) => {
    if (err) throw err;
    console.log("Connected!");
});


app.get('/cw_tree', function (req, res) {
    res.sendFile(__dirname + '/front/cw/cw_tree_screen.html');
});

app.post('/cw_tree', function (req, res) {

    var body = req.body;
    if (body.text_edited) {
        let elems = cw.take_elems();
        let len = elems.split(' ').length - 1;
        let time = cw.take_time();
        let step = cw.take_steps();
        let arr_data = [elems, time, step];

        if (len > 0) {
            let command = "insert into `all_tests` (`itemset`,`time`, `steps`) value
(?,?,?);";
            conn.query(command, arr_data, function (err, result) {
                if (err) throw err;
                else res.end(JSON.stringify(result));
            });
        }
        else res.end("_empty_data");
        //add_in_table_2("average_case", 1, len, time, step);
```

```
        cw.zeroing();
    }
    if (body.build) {
        let arg1 = body.build.join(' ');
        let arg2 = body.build.length;
        let string_tree = cw.build_tree(arg1, arg2);
        res.end(string_tree);
    }
});


app.get('/cw_table', function (req, res) {
    res.sendFile(__dirname + '/front/cw/cw_table_screen.html');
});

app.post('/cw_table', function (req, res) {
    let body = req.body;
    if (body.obj_out) {
        let command = "select * from `all_tests`";
        if (body.obj_out.from != "") {
            command += " where id>=" + body.obj_out.from;
            if (body.obj_out.to != "")
                command += " and id<=" + body.obj_out.to;
        }
        else if (body.obj_out.to != "") {
            command += " where id<=" + body.obj_out.to;
        }
        conn.query(command, function (err, result) {
            if (err) throw err;
            res.end(JSON.stringify(result));
```

```
    });

}

if (body.obj_gen) {

    let len = body.obj_gen.len,
        num = body.obj_gen.num,
        flag = body.obj_gen.flag;
    let obj_new_data = {
        "num": num,
        "len": len,
        "time": 0.0,
        "time_2": 0.0,
        "step": 0.0
    };

    let table_name;
    if (flag == "random") { table_name = "average_values"; }
    else if (flag == "average") { table_name = "average_case"; }
    else if (flag == "worst") { table_name = "worst_case"; }

    for (let i = 0; i < num; i++) {
        cw.zeroing();
        let str_elems = cw.data_rand(len);
        if (flag == "average") {
            if (str_elems.split(' ').sort().join(' ') == str_elems) {
                i--;
                continue;
            }
        }
```

```
                else if (flag == "worst") {
                    str_elems = str_elems.split(' ').sort().join(' ');
                }
                let rand_ret = cw.data_test(str_elems, len);
                let elems = cw.take_elems();
                let time = cw.take_time();
                let time_2 = cw.take_time_2();//new
                let step = cw.take_steps();


                    let command = "insert into `all_tests` (`itemset`, `time`, `time_2`, `steps`) value (?,?,?,?);";
                conn.query(command, [elems, time, time_2, step], function (err, result) {
                    if (err) throw err;
                });
                obj_new_data.time = (obj_new_data.time * (i) + time) / (i + 1);
                obj_new_data.time_2 = (obj_new_data.time_2 * (i) + time_2) / (i + 1);
                obj_new_data.step = (obj_new_data.step * (i) + step) / (i + 1);
            }
                add_in_table_2(table_name, obj_new_data.num, obj_new_data.len, obj_new_data.time, obj_new_data.time_2, obj_new_data.step);

            res.end("done");
        }



    });
```

```
app.get('/cw_graph', function (req, res) {
    res.sendFile(__dirname + '/front/cw/cw_graph_screen.html');
});


app.post('/cw_graph', function (req, res) {
    let body = req.body;
    if (body.data_to_table) {
        take_data(body.data_to_table, res);
    }
});

app.get('/cw_graph_dop', function (req, res) {
    res.sendFile(__dirname + '/front/cw/cw_graph_dop_screen.html');
});


app.post('/cw_graph_dop', function (req, res) {
    let body = req.body;
    if (body.data_to_table) {
        take_data(body.data_to_table, res);
    }
});

//end

function take_data(table_name, res) {
    let command = "select * from ??";
    conn.query(command, [table_name], function (err, result) {
        if (err) throw err;
        res.end(JSON.stringify(result));
```

```
    });
}

function add_in_table_2(table_name, num, len, time, time_2, step) {
    let flag = false;
    let command = "select * from ?? where number_of_leaves=?";
    conn.query(command, [table_name, len], function (err, result) {
        if (err) throw err;
        //console.log(result);
        if (result.length == 0) {
            insert_2([table_name, num, len, time, time_2, step])
        }
        else {
            update(table_name, result, num, len, time, time_2, step)
        }
    });
}
//update average_values set time = 1577.7272949219 where id=3426;




function insert_2(arr_data) {
        let command = "insert into ?? (`number_of_tests`, `number_of_leaves`, `time`, `time_2`, `steps`) value (?,?,?,?,?)";
    conn.query(command, arr_data, function (err, result) {
        if (err) throw err;
    });
}

function update(table_name, result, num, len, time, time_2, step) {
```

```
        let command = "update ?? set ??=? where ??=?";
        //number_of_tests update
                conn.query(command, [table_name, "number_of_tests",
result[0].number_of_tests + num, "number_of_leaves", len], function (err, result) {
            if (err) throw err;
        });
        command = "update ?? set ??=? where ??=?";
        //time update
                conn.query(command, [table_name, "time", (result[0].time *
result[0].number_of_tests + time * num) / (result[0].number_of_tests + num),
"number_of_leaves", len], function (err, result) {
            if (err) throw err;
        });

        command = "update ?? set ??=? where ??=?";
        //time_2 update
                conn.query(command, [table_name, "time_2", (result[0].time_2 *
result[0].number_of_tests + time_2 * num) / (result[0].number_of_tests + num),
"number_of_leaves", len], function (err, result) {
            if (err) throw err;
        });

        command = "update ?? set ??=? where ??=?";
        //steps update
                conn.query(command, [table_name, "steps", (result[0].steps *
result[0].number_of_tests + step * num) / (result[0].number_of_tests + num),
"number_of_leaves", len], function (err, result) {
            if (err) throw err;
        });

    }
```