

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: Случайные БДП с рандомизацией
Вариант 11

Студент гр. 8304

Барышев А.А.

Преподаватель

Фирсов К.В.

Санкт-Петербург

2019

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студент Барышев А.А.

Группа 8304

Тема работы: исследование случайного БДП с рандомизацией

Исходные данные: Написать программу, реализующую случайное БДП с рандомизацией. Сгенерировать наборы входных данных, использовать их для измерения количественных характеристик структуры данных, алгоритма, реализующего БДП. Сравнить экспериментальные результаты с теоретическими.

Содержание пояснительной записки: «Введение», «Описание алгоритма», «Описание структур данных и функций», «Тестирование», «Исследование», «Выводы», «Список использованных источников», «Приложение А. Код работы»

Предполагаемый объем пояснительной записки:

Не менее 34 страниц.

Дата выдачи задания: 14.10.2019

Дата сдачи реферата:

Дата защиты реферата:

Студент

Барышев А.А.

Преподаватель

Фирсов К.В.

АННОТАЦИЯ

Данная курсовая работа посвящена исследованию случайного БДП с рандомизацией. Исследование представляет собой большое количество тестов, некоторый статистический анализ данных. Представлено исследование на 200 сгенерированных наборах данных случайного размера, а также 1300 запусков программы для «худшего случая» последовательности нерандомизированного БДП.

SUMMARY

This course work is devoted to the study of randomized BDP. The study is a large number of tests, some statistical analysis of the data. A study is presented on 200 generated random-size datasets, as well as 1000 program runs for the "worst case" sequence of non-randomized BDP.

СОДЕРЖАНИЕ

	Введение	5
1.	Описание программы	6
1.1.	Программы для решения задачи	6
1.2.	Скрипт на bash	6
2.	Описание структур данных и функций	7
2.1.	Описание случайной БДП с рандомизацией	7
2.2.	Описание функции Find	8
3.	Тестирование	9
3.1.	Худший случай	9
3.2.	Случайные наборы	13
4.	Исследование	16
4.1.	Распределение корней в «худшем случае»	16
4.2.	Средняя оценка высоты для каждого корня	17
4.3.	Среднее время поиска элемента	18
	Выводы	19
	Список использованных источников	20
	Приложение А. Код работы	21

ВВЕДЕНИЕ

Целью данной работы является реализация и экспериментальное машинное исследование алгоритмов быстрого поиска на основе случайного БДП с рандомизацией.

1. ОПИСАНИЕ ПРОГРАММЫ

1.1. Программы для решения задачи

Для выполнения поставленной задачи необходимо было создать систему программ, которая будет обрабатывать, принимать и генерировать данные. Код `main.cpp` позволяет генерировать наборы корней и высот для последовательности чисел от 10 до 1. Данный код множество раз принимает на вход данную последовательность, чтобы выявить закономерности: какой корень является «лучшим случаем» рандомизированной вставки(наименьшая средняя высота), какой худшим, наибольшее количество бинарных деревьев с одним и тем же корнем, вероятность лучшего или худшего корней и т.д.

Код `ReadStatistics.cpp` позволяет обработать полученные данные.

Код `Generation.cpp` создаёт файл со случайными наборами данных, которые считает код `mainForRand.cpp`, найдя среднее значение времени, которое необходимо затратить на поиск случайного элемента рандомизированной БДП.

1.2. Скрипт на bash

Для последовательного запуска кода использовался скрипт на `bash`, который также приведён в Приложении А.

2. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

2.1. Описание случайной БДП с рандомизацией

Двоичное дерево поиска (англ. *binary search tree*, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (*свойства дерева поиска*):

Оба поддерева — левое и правое — являются двоичными деревьями поиска.

У всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла X .

У всех узлов *правого* поддерева произвольного узла X значения ключей данных *больше либо равны*, нежели значение ключа данных самого узла X .

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения *меньше*.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных. Однако это касается реализации, а не природы двоичного дерева поиска.

Для целей реализации двоичное дерево поиска можно определить так:

Двоичное дерево состоит из узлов (вершин) — записей вида (data, left, right), где data — некоторые данные, привязанные к узлу, left и right — ссылки на узлы, являющиеся детьми данного узла — левый и правый сыновья соответственно. Для оптимизации алгоритмов конкретные реализации предполагают также определения поля parent в каждом узле (кроме корневого) — ссылки на родительский элемент.

Данные (data) обладают ключом (key), на котором определена операция сравнения «меньше». В конкретных реализациях это может быть пара (key, value) — (ключ и значение), или ссылка на такую пару, или простое определение операции сравнения на необходимой структуре данных или ссылке на неё.

Для любого узла X выполняются свойства дерева поиска: $\text{key}[\text{left}[X]] < \text{key}[X] \leq \text{key}[\text{right}[X]]$, то есть ключи данных родительского узла больше ключей данных левого сына и нестрогие меньше ключей данных правого.

Особенность рандомизированного БДП заключается в том, что в какой-то момент, при вставке следующего элемента, он оказывается в корне. Таким образом, с некоторой вероятностью, может поменяться корень данного БДП, что повлечёт другой порядок вставки прочих элементов.

2.2. Описание функции Find

Для данной функции было найдено среднее время, необходимое для её выполнения. Функция «Find» имеет следующий прототип:

```
node* Find(node*, int);
```

Принцип работы данной функции основан на двух вещах:

- 1) Бинарное дерево – рекурсивная АТД;
- 2) Все узлы бинарного дерева поиска больше, чем узлы его левого поддерева, и, напротив, все узлы меньше, чем узлы правого поддерева.

3. ТЕСТИРОВАНИЕ

3.1. Худший случай

Под «худшим случаем» подразумевается последовательность чисел, последовательно возрастающая или убывающая, что приводит к тому, что структура БДП оказывается линейной, т.е. нет левых или правых поддеревьев.

Таким образом, чтобы избежать такой структуры(в которой операции над элементами значительно медленнее) используется рандомизированная вставка.

Фрагмент тестирования для последовательности от 10 до 1:

Was introduce:

7 9 10

8

1 2 4 5 6

3

Was introduce:

2 5 6 7 8 9 10

3 4

1

Was introduce:

5 6 10

7 9

8

1 2 3 4

Was introduce:

5 6 10

7 9

8

1 2 3 4

Was introduce:

7 8 9 10

5 6

1 2 3 4

Was introduce:

7 8 9 10

5 6

1 2 3 4

Was introduce:

6 8 9 10

7

2 3 4 5

1

Was introduce:

3 8 9 10

5 7

6

4

2

1

Was introduce:

3 8 9 10

5 7

6

4

2

1

Was introduce:

8 9 10

1 2 3 4 5 6 7

Was introduce:

10

3 7 8 9

4 6

5

1 2

Was introduce:

10

3 7 8 9

4 6

5

1 2

Was introduce:

5 7 8 9 10

6

1 4

2 3

Was introduce:

6 7 8 9 10

5

1 3 4

2

Was introduce:

6 7 8 9 10

5

1 3 4

2

Was introduce:

3 8 9 10

6 7

4 5

2

1

Was introduce:

5 7 8 9 10

6

4

3

1 2

3.2. Случайные наборы

Фрагмент тестирования для сгенерированных наборов данных(ниже примечание: какой элемент будет находиться с помощью функции Find):

Was introduce:

95

37 94

67 81

55 58

13

0 8

Will find: 95;

Was introduce:

41 78

53

48 49

38

27

0

Will find: 0;

Was introduce:

45 84 91

85

83

51 57

42

21

0 11 16

Will find: 0;

Was introduce:

72 81

51 70

0 18 20 35

Will find: 72;

Was introduce:

51 69 94 99

52 64

51

16 34

0

Will find: 51;

Was introduce:

53 68 70

4 20 26

0

Will find: 0;

Was introduce:

34 65 81 93

88

51

0 31

9 15

Will find: 0;

Was introduce:

0 77

32 42 46 69 74

67

19 27

19

Will find: 77;

Was introduce:

0 61 63 65 95

63

46

34 42

28

Will find: 63;

Was introduce:

36 80 98

90

80

49

35

0 12

Will find: 35;

4. ИССЛЕДОВАНИЕ

4.1. Распределение корней в «худшем случае»

```
root distribution:
Quantity roots with value '1' is: 162
Quantity roots with value '2' is: 143
Quantity roots with value '3' is: 133
Quantity roots with value '4' is: 142
Quantity roots with value '5' is: 161
Quantity roots with value '6' is: 158
Quantity roots with value '7' is: 128
Quantity roots with value '8' is: 103
Quantity roots with value '9' is: 88
Quantity roots with value '10' is: 114
Least probable root meets over 88; It is '9'
Most probable root meets over 162; It is '1'
The program has been run 1332 times
```

```
root distribution:
Quantity roots with value '1' is: 155
Quantity roots with value '2' is: 166
Quantity roots with value '3' is: 136
Quantity roots with value '4' is: 154
Quantity roots with value '5' is: 163
Quantity roots with value '6' is: 137
Quantity roots with value '7' is: 136
Quantity roots with value '8' is: 108
Quantity roots with value '9' is: 75
Quantity roots with value '10' is: 102
Least probable root meets over 75; It is '9'
Most probable root meets over 166; It is '2'
The program has been run 1332 times
```

Распределение корней таково, что наименьшее количество приходится на элементы, расположенные в последовательности ниже прочих(чем дальше от начала последовательности, тем ниже вероятность вставки в корень). Довольно много корней приходится на середину последовательностей.

4.2. Средняя оценка высоты для каждого корня

```
Medium value of Height for 1 is: 6.72903; max height is: 10; min height is: 5;  
Medium value of Height for 2 is: 6.3012; max height is: 9; min height is: 5;  
Medium value of Height for 3 is: 5.97059; max height is: 8; min height is: 5;  
Medium value of Height for 4 is: 5.62987; max height is: 7; min height is: 4;  
Medium value of Height for 5 is: 5.18405; max height is: 6; min height is: 4;  
Medium value of Height for 6 is: 5.16788; max height is: 6; min height is: 4;  
Medium value of Height for 7 is: 5.42647; max height is: 7; min height is: 4;  
Medium value of Height for 8 is: 5.7963; max height is: 8; min height is: 4;  
Medium value of Height for 9 is: 6.22667; max height is: 8; min height is: 5;  
Medium value of Height for 10 is: 6.68627; max height is: 9; min height is: 5;
```

```
Medium value of Height for 1 is: 6.51852; max height is: 10; min height is: 5;  
Medium value of Height for 2 is: 6.34266; max height is: 9; min height is: 5;  
Medium value of Height for 3 is: 5.90226; max height is: 8; min height is: 4;  
Medium value of Height for 4 is: 5.47183; max height is: 7; min height is: 4;  
Medium value of Height for 5 is: 5.24224; max height is: 6; min height is: 4;  
Medium value of Height for 6 is: 5.13924; max height is: 6; min height is: 4;  
Medium value of Height for 7 is: 5.375; max height is: 7; min height is: 4;  
Medium value of Height for 8 is: 5.85437; max height is: 8; min height is: 4;  
Medium value of Height for 9 is: 6.09091; max height is: 8; min height is: 5;  
Medium value of Height for 10 is: 6.59649; max height is: 10; min height is: 5;
```

Исходя из результатов, лучший случай распределения элементов приходится на середину входных последовательностей. Тогда высота наиболее близка к значению, равному 5-и.

4.3. Среднее время поиска элемента

На 200 различных случайных наборах данных, не превышающих 15 элементов, был запущен код, который считает среднее время, необходимое для поиска заданного элемента. Данное время было равно 0.000757576, что связано по большей части с тем, что среди данных присутствовала часть простейших случаев, когда искомым элементов являлся корень или ближайшее к нему поддерево. В случае отбрасывания подобных элементов получится около 0.00100943, что говорит о том, что алгоритм очень эффективен(данное время сравнимо с выполнением цикла в несколько итераций).

ВЫВОДЫ

В результате выполнения данной курсовой работы, была доказана эффективность случайного БДП с рандомизацией, а также получены статистические данные для данной АТД.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1) Статья из википедии: [https://ru.wikipedia.org/wiki/
%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE
%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_
%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0)
- 2) [https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B0%D0%BD
%D0%B4%D0%BE%D0%BC
%D0%B8%D0%B7%D0%B8%D1%80%D0%BE
%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_
%D0%B1%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%BE
%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_
%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0](https://neerc.ifmo.ru/wiki/index.php?title=%D0%A0%D0%B0%D0%BD%D0%B4%D0%BE%D0%BC%D0%B8%D0%B7%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B1%D0%B8%D0%BD%D0%B0%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%D0%B0)

ПРИЛОЖЕНИЕ А

КОД РАБОТЫ

СКРИПТ:

```
#!/BIN/BASH
G++ -WALL -WERROR SOURCE/MAIN.CPP -O MAIN
./MAIN
G++ -WALL -WERROR SOURCE/READSTATISTICS.CPP -O READSTATISTICS
./READSTATISTICS
G++ -WALL -WERROR SOURCE/GENERATION.CPP -O GENERATION
./GENERATION
G++ -WALL -WERROR SOURCE/MAINFORRAND.CPP -O MAINFORRAND
./MAINFORRAND
G++ -WALL -WERROR SOURCE/MEDIUMTIME.CPP -O MEDIUMTIME
./MEDIUMTIME
RM MEDIUMTIME
RM GENERATION
RM MAINFORRAND
RM MAIN
RM READSTATISTICS
```

КОД MAIN.CPP:

```
#INCLUDE <IOSTREAM>

#INCLUDE <FSTREAM>

#INCLUDE <CSTDLIB>

#INCLUDE <CTIME>

#INCLUDE <STRING>

#INCLUDE "SECONDARYFUNCTIONS.H"

#INCLUDE "BINTREESEARCH.H"

INT MAIN() {

    SRAND(TIME(0));

    INT ARRAYNUMBER = 0;

    STD::OFSTREAM FOUTVIZ;

    FOUTVIZ.OPEN("RESULTVIZUALIZEWORST.TXT", STD::IOS::APP);

    STD::OFSTREAM FOUTROOTS;

    FOUTROOTS.OPEN("RESULTROOTSWORST.TXT", STD::IOS::APP);

    STD::OFSTREAM FOUTHEIGHTS;

    FOUTHEIGHTS.OPEN("RESULTHEIGHTSWORST.TXT", STD::IOS::APP);

    STD::IFSTREAM FENTER("TESTS/TESTDATAWORST.TXT");

    STD::STRING EXPR;

    INT* ARRAYNUM = NEW INT[200];
```

```

        WHILE (STD::GETLINE (FENTER,  EXPR)) {
            BINSEARCHTREE H = NULL;
            ARRAYNUMBER = GETNUMBERS (ARRAYNUM,  EXPR);
            FOR (INT I = 0; I < ARRAYNUMBER; I++) {
                H = INSERT (H,  ARRAYNUM[I]);
            }
            FOUTROOTS << H->KEY << " ";
            FOUTHEIGHTS << HBT (H) << " ";
            FOUTVIZ << " _____" <<
STD::ENDL;
            FOUTVIZ << "WAS INTRODUCE: " << STD::ENDL;
            DISPLAYBT (H,  1,  FOUTVIZ);
            DESTROY (H);
        }

        DELETE ARRAYNUM;
        FENTER.CLOSE();
        FOUTROOTS.CLOSE();
        FOUTVIZ.CLOSE();
        FOUTHEIGHTS.CLOSE();
        RETURN 0;
    }

```

КОД MEDIUMTIME.CPP:

```

#include <Iostream>
#include <fstream>
#include <cstdlib>

INT MAIN() {
    STD::IFSTREAM FILETIME;
    FILETIME.OPEN("RESULTTIMEFIND.TXT");
    FLOAT BUF;
    FLOAT MEDIUM = 0;
    INT COUNTER = 0;

    WHILE (FILETIME >> BUF) {

```

```

        IF(BUF != 0){
            MEDIUM = MEDIUM + BUF;
            COUNTER++;
        }
    }
    MEDIUM = MEDIUM/COUNTER;

    STD::COUT << " MEDIUM TIME FOR FIND ONE ELEM(FOR AN AVERAGE OF
" << COUNTER << " DATA) IS: " << MEDIUM << STD::ENDL;

    FILETIME.CLOSE();
    RETURN 0;
}

```

КОД GENERATION.CPP:

```

#include <Iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>

int main() {
    srand(time(0));
    std::ofstream file;
    file.open("TESTS/TESTDATARANDOM.TXT", std::ios::app);
    for(int i = 0; i < 200; i++){
        for(int i = 0; i < 5 + rand()%15; i++){
            file << rand()%100 << " ";
        }
        file << std::endl;
    }

    file.close();
    return 0;
}

```

КОД MAINFORRAND.CPP:

```
#INCLUDE <Iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <string>
#include "SECONDARYFUNCTIONS.H"
#include "BINTREESEARCH.H"

INT MAIN() {
    SRAND(TIME(0));
    INT ARRAYNUMBER = 0;
    STD::OFSTREAM FOUTVIZ;
    FOUTVIZ.OPEN("RESULTVIZUALIZE.TXT", STD::IOS::APP);
    STD::OFSTREAM FOUTROOTS;
    FOUTROOTS.OPEN("RESULTROOTS.TXT", STD::IOS::APP);
    STD::OFSTREAM FOUTHEIGHTS;
    FOUTHEIGHTS.OPEN("RESULTHEIGHTS.TXT", STD::IOS::APP);
    STD::OFSTREAM FOUTTIMEFIND;
    FOUTTIMEFIND.OPEN("RESULTTIMEFIND.TXT", STD::IOS::APP);
    STD::IFSTREAM FENTER("TESTS/TESTDATARANDOM.TXT");
    STD::STRING EXPR;
    INT* ARRAYNUM = NEW INT[200];
    INT STOP, START, FIND;

    WHILE(STD::GETLINE(FENTER, EXPR)) {
        BINSEARCHTREE H = NULL;
        ARRAYNUMBER = GETNUMBERS(ARRAYNUM, EXPR);
        FOR(INT I = 0; I < ARRAYNUMBER; I++) {
            H = INSERT(H, ARRAYNUM[I]);
        }
        FOUTROOTS << H->KEY << " ";
        FOUTHEIGHTS << HBT(H) << " ";
        FOUTVIZ << " _____ " <<
    }
    STD::ENDL;
```



```

        FOUTVIZ << "WAS INTRODUCE: " << STD::ENDL;
        FIND = ARRAYNUM[RAND() %ARRAYNUMBER];
        START = CLOCK();
        FIND(H, FIND);
        STOP = CLOCK();
        FOUTTIMEFIND << (STOP - START)/1000.0 << " ";
        DISPLAYBT(H, 1, FOUTVIZ);
        FOUTVIZ << " WILL FIND: " << FIND << ";" << STD::ENDL;
        DESTROY(H);
    }

    DELETE ARRAYNUM;
    FENTER.CLOSE();
    FOUTROOTS.CLOSE();
    FOUTVIZ.CLOSE();
    FOUTHEIGHTS.CLOSE();
    FOUTTIMEFIND.CLOSE();
    RETURN 0;
}

```

КОД READSTATISTICS.CPP:

```

#include <Iostream>
#include <fstream>

STRUCT NUMBERROOT{
    INT COUNTER = 0;
    INT ARRAY[3000];
};

typedef STRUCT NUMBERROOT NUMBERROOT;

INT MAIN() {
    STD::IFSTREAM FILERROOTS;
    STD::IFSTREAM FILEHEIGHTS;
    FILERROOTS.OPEN("RESULTROOTSWORST.TXT");
    FILEHEIGHTS.OPEN("RESULTHEIGHTSWORST.TXT");
}

```

```

INT BUFROOTS;
NUMBERROOT ROOTSWORST[10];

WHILE(FILEROOTS >> BUFROOTS) {
    FILEHEIGHTS >> ROOTSWORST[BUFROOTS-
1].ARRAY[ROOTSWORST[BUFROOTS-1].COUNTER];
    ROOTSWORST[BUFROOTS-1].COUNTER++;
//    STD::COUT << "HEIGHT IS: " << ROOTSWORST[BUFROOTS-
1].ARRAY[ROOTSWORST[BUFROOTS-1].COUNTER - 1] << "; ROOT IS:" <<
BUFROOTS << "; COUNTER IS: " << ROOTSWORST[BUFROOTS-1].COUNTER <<
";" << STD::ENDL;
}

STD::COUT << "ROOT DISTRIBUTION: " << STD::ENDL;
INT MAX = 0;
INT MIN = 100000;
INT INDEXMIN, INDEXMAX, SUM;
SUM = 0;

FOR(INT I = 0; I < 10; I++){
    STD::COUT << "QUANTITY ROOTS WITH VALUE \" << I + 1 <<
"\ ' IS: " << ROOTSWORST[I].COUNTER << STD::ENDL;
    IF(ROOTSWORST[I].COUNTER > MAX) {
        MAX = ROOTSWORST[I].COUNTER;
        INDEXMAX = I + 1;
    }
    IF(ROOTSWORST[I].COUNTER < MIN) {
        MIN = ROOTSWORST[I].COUNTER;
        INDEXMIN = I + 1;
    }
    SUM = SUM + ROOTSWORST[I].COUNTER;
}

STD::COUT << "LEAST PROBABLE ROOT MEETS OVER " << MIN << "; IT
IS \" << INDEXMIN << "\ ' << STD::ENDL;

```

```

        STD::COUT << "MOST PROBABLE ROOT MEETS OVER " << MAX << "; IT
IS \" << INDEXMAX << "\" << STD::ENDL;

        STD::COUT << "THE PROGRAM HAS BEEN RUN " << SUM << " TIMES" <<
STD::ENDL;


        FLOAT MEDIUMVALUE = 0;
        MAX = 0;
        MIN = 100000;


        FOR(INT I = 0; I < 10; I++){
                IF(ROOTSWORST[I].COUNTER == 0){
                        CONTINUE;
                }
                FOR(INT J = 0; J < ROOTSWORST[I].COUNTER; J++){
                        MEDIUMVALUE = MEDIUMVALUE +
(FLOAT)ROOTSWORST[I].ARRAY[J];
                        IF(ROOTSWORST[I].ARRAY[J] > MAX){
                                MAX = ROOTSWORST[I].ARRAY[J];
                        }
                        IF(ROOTSWORST[I].ARRAY[J] < MIN){
                                MIN = ROOTSWORST[I].ARRAY[J];
                        }
                }
                MEDIUMVALUE = MEDIUMVALUE/(FLOAT)ROOTSWORST[I].COUNTER;
                STD::COUT << "MEDIUM VALUE OF HEIGHT FOR " << I + 1 << "
IS: " << MEDIUMVALUE << "; MAX HEIGHT IS: " << MAX << "; MIN HEIGHT
IS: " << MIN << ";" << STD::ENDL;
                MEDIUMVALUE = 0;
                MIN = 100000;
                MAX = 0;
        }


        FILEHEIGHTS.CLOSE();
        FILERROOTS.CLOSE();
        RETURN 0;

```

```
}
```

КОД BINTREESEARCH.H:

```
#PRAGMA ONCE
```

```
STRUCT NODE
```

```
{
```

```
    INT KEY;
```

```
    INT SIZE;
```

```
    NODE* LEFT;
```

```
    NODE* RIGHT;
```

```
    NODE (INT K) {
```

```
        KEY = K;
```

```
        LEFT = RIGHT = 0;
```

```
        SIZE = 1;
```

```
    }
```

```
};
```

```
NODE* FIND (NODE*, INT);
```

```
NODE* INSERT (NODE*, INT);
```

```
INT GETSIZE (NODE* );
```

```
VOID FIXSIZE (NODE* );
```

```
NODE* ROTATERIGHT (NODE* );
```

```
NODE* ROTATELEFT (NODE* );
```

```
NODE* INSERTROOT (NODE*, INT);
```

```
NODE* JOIN (NODE*, NODE*);
```

```
NODE* REMOVE (NODE*, INT);
```

```
TYPDEF NODE *BINSEARCHTREE;
```

```
NODE* FIND (NODE* P, INT K)
```

```
{
```

```
    IF ( !P ) RETURN 0;
```

```
    IF ( K == P->KEY )
```

```
        RETURN P;
```

```
    IF ( K < P->KEY )
```

```

        RETURN FIND(P->LEFT,K);
    ELSE
        RETURN FIND(P->RIGHT,K);
}

INT GETSIZE(NODE* P)
{
    IF( !P ) RETURN 0;
    RETURN P->SIZE;
}

VOID FIXSIZE(NODE* P)
{
    P->SIZE = GETSIZE(P->LEFT)+GETSIZE(P->RIGHT)+1;
}

INT HBT(NODE* B)
{
    IF (B == NULL) RETURN 0;
    ELSE RETURN STD::MAX(HBT (B->LEFT), HBT(B->RIGHT)) + 1;
}

NODE* ROTATERIGHT(NODE* P)
{
    NODE* Q = P->LEFT;
    IF( !Q ) RETURN P;
    P->LEFT = Q->RIGHT;
    Q->RIGHT = P;
    Q->SIZE = P->SIZE;
    FIXSIZE(P);
    RETURN Q;
}

NODE* ROTATELEFT(NODE* Q)
{

```

```

    NODE* P = Q->RIGHT;
    IF( !P ) RETURN Q;
    Q->RIGHT = P->LEFT;
    P->LEFT = Q;
    P->SIZE = Q->SIZE;
    FIXSIZE(Q);
    RETURN P;
}

NODE* INSERTROOT(NODE* P, INT K)
{
    IF( !P ) RETURN NEW NODE(K);
    IF( K < P->KEY )
    {
        P->LEFT = INSERTROOT(P->LEFT, K);
        RETURN ROTATERIGHT(P);
    }
    ELSE
    {
        P->RIGHT = INSERTROOT(P->RIGHT, K);
        RETURN ROTATELEFT(P);
    }
}

NODE* INSERT(NODE* P, INT K)
{
    IF( !P ) RETURN NEW NODE(K);
    IF( RAND()%(P->SIZE+1) == 0 )
        RETURN INSERTROOT(P,K);
    IF( P->KEY>K )
        P->LEFT = INSERT(P->LEFT,K);
    ELSE
        P->RIGHT = INSERT(P->RIGHT,K);
    FIXSIZE(P);
    RETURN P;
}

```

```

}

NODE* JOIN(NODE* P, NODE* Q)
{
    IF( !P ) RETURN Q;
    IF( !Q ) RETURN P;
    IF( RAND()%(P->SIZE+Q->SIZE) < P->SIZE )
    {
        P->RIGHT = JOIN(P->RIGHT, Q);
        FIXSIZE(P);
        RETURN P;
    }
    ELSE
    {
        Q->LEFT = JOIN(P, Q->LEFT);
        FIXSIZE(Q);
        RETURN Q;
    }
}

NODE* REMOVE(NODE* P, INT KEY)
{
    IF(!P) RETURN P;
    IF(P->KEY == KEY)
    {
        NODE* Q = JOIN(P->LEFT, P->RIGHT);
        DELETE P;
        RETURN Q;
    }
    ELSE IF(KEY < P->KEY)
        P->LEFT = REMOVE(P->LEFT, KEY);
    ELSE
        P->RIGHT = REMOVE(P->RIGHT, KEY);
    RETURN P;
}

```

```

VOID DISPLAYBT(NODE* B, INT N, STD::OFSTREAM& FOUT) {
    IF (B != NULL) {
        FOUT << ' ' << B->KEY;
        IF(B->RIGHT != NULL) {
            DISPLAYBT(B->RIGHT, N + 1, FOUT);
        }
        ELSE FOUT << STD::ENDL;
        IF(B->LEFT != NULL) {
            FOR (INT I = 1; I <= N; I++)
                FOUT << " ";
            DISPLAYBT(B->LEFT, N + 1, FOUT);
        }
    }
}

```

```

VOID DISPLAY(NODE* B, INT N) {
    IF (B != NULL) {
        STD::COUT << ' ' << B->KEY;
        IF(B->RIGHT != NULL) {
            DISPLAY(B->RIGHT, N + 1);
        }
        ELSE STD::COUT << STD::ENDL;
        IF(B->LEFT != NULL) {
            FOR (INT I = 1; I <= N; I++)
                STD::COUT << " ";
            DISPLAY(B->LEFT, N + 1);
        }
    }
}

```

```

VOID DESTROY (NODE* &B)
{
    IF (B != NULL) {
        DESTROY (B->LEFT);
        DESTROY (B->RIGHT);
    }
}

```



```

        DELETE B;

        B = NULL;

    }

}

```

KOD SECONDARYFUNCTIONS.H:

```
#PRAGMA ONCE
```

```

INT STR_LEN (CHAR* WORD) {
    INT LONG_OF_WORD = 0;
    WHILE (WORD [LONG_OF_WORD] ) {
        LONG_OF_WORD++;
    }
    RETURN LONG_OF_WORD;
}

```

```

INT POW_INT (INT X, INT Y) {
    IF (Y == 0)
        RETURN 1;

    INT I;
    INT C = X;
    FOR (I = 1; I < Y; I++)
        X *= C;
    RETURN X;
}

```

```

INT UP_TO_INT (CHAR* STR) {
    INT I, J, COUNT;
    INT RESULT = 0;
    COUNT = 0;
    CHAR DIGITS[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8',
'9'};
    FOR (I = STR_LEN (STR) - 1; I >= 0; I--) {
        FOR (J = 1; J < 10; J++)
            IF (STR [COUNT] == DIGITS [J]) {
                RESULT = RESULT + J * POW_INT (10, I);
            }
    }
}

```

```

        BREAK;
    }
    COUNT++;
}
RETURN RESULT;
}

INT GETNUMBERS (INT* ARRAY, STD::STRING EXPR) {
    INT ARRAYNUMBER, STRPOSITION;
    ARRAYNUMBER = STRPOSITION = 0;
    CHAR NUMBERSTRING[100];
    FOR (INT I = 0; I <= (INT)EXPR.SIZE(); I++) {
        IF (EXPR[I] != ' ' && EXPR[I] != '\N' && EXPR[I]) {
            NUMBERSTRING[STRPOSITION] = EXPR[I];
            STRPOSITION++;
        }
        ELSE {
            NUMBERSTRING[STRPOSITION] = '\0';
            ARRAY[ARRAYNUMBER] = UP_TO_INT (NUMBERSTRING);
            ARRAYNUMBER++;
            STRPOSITION = 0;
        }
    }
    RETURN ARRAYNUMBER;
}

```