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 20 страниц.

Дата выдачи задания: 14.10.2019

Дата сдачи реферата: 17.12.2019

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 100 сгенерированных наборах данных случайного размера, а также 500 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 100 generated random-size datasets, as well as 500 program runs for the" worst case " sequence of non-randomized BDP.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

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

1. ОПИСАНИЕ АЛГОРИТМА

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 1 до 10. Данный код запускается множество раз, чтобы выявить закономерности: какой корень является «лучшим случаем» рандомизированной вставки, какой худшим, наибольшее количество бинарных деревьев с одним и тем же корнем 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 и многократного 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: key[left[X]] < key[X] ≤ key[right[X]], то есть ключи данных родительского узла больше ключей данных левого сына и нестрого 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 1 до 10:

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:

```
_____________________________________
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»

```
Quantity roots with value '1' is: 57
Quantity roots with value '2' is: 63
Quantity roots with value '3' is: 51
Quantity roots with value '4' is: 47
Quantity roots with value '5' is: 58
Quantity roots with value '6' is: 63
Quantity roots with value '7' is: 56
Quantity roots with value '8' is: 30
Quantity roots with value '9' is: 31
Quantity roots with value '10' is: 43
Least probable root meets over 30; It is '8'
Most probable root meets over 63; It is '2'
The program has been run 499 times
```

Распределение корней таково, что наименьшее количество приходится на элементы, расположенные в последовательности ниже прочих, но не на самом низком уровне. Довольно много корней приходится на середину последовательностей.

4.2. Средняя оценка высоты для каждого корня

```
Medium value of Height for 1 is: 6.80702; max height is: 10; min height is: 5;
Medium value of Height for 2 is: 6.1746; max height is: 9; min height is: 5;
Medium value of Height for 3 is: 5.92157; max height is: 8; min height is: 5;
Medium value of Height for 4 is: 5.74468; max height is: 7; min height is: 4;
Medium value of Height for 5 is: 5.32759; max height is: 6; min height is: 4;
Medium value of Height for 6 is: 5.28571; max height is: 6; min height is: 4;
Medium value of Height for 7 is: 5.57143; max height is: 7; min height is: 5;
Medium value of Height for 8 is: 5.83333; max height is: 8; min height is: 5;
Medium value of Height for 9 is: 6; max height is: 8; min height is: 5;
Medium value of Height for 10 is: 6.67442; 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. Средняя оценка высоты для каждого корня

На 100 различных случайных наборах данных, не превышающих 20 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 ~0.00112, 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
2) Статья из открытого источника: https://habr.com/ru/post/145388/

ПРИЛОЖЕНИЕ А

КОД РАБОТЫ

СКРИПТ:

```
#!/BIN/BASH
FOR ((I = 1; I < 500; I++))
DO
G++ -WALL -WERROR SOURCE/MAIN.CPP -O MAIN
./MAIN
ECHO "TEST $I PASSED SUCCESSFUL"
DONE
G++ -WALL -WERROR SOURCE/READSTATISTICS.CPP -O READSTATISTICS
./READSTATISTICS
FOR ((A = 1; A < 100; A++))
DO
G++ -WALL -WERROR SOURCE/GENERATION.CPP -O GENERATION
G++ -WALL -WERROR SOURCE/GENERATION.CPP -O GENERATION
ECHO "DATA $A WAS GENERATED"
./GENERATION
DONE
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;
}
```