

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: Случайное бинарное дерево поиска

Студент гр. 8304

Порывай П.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2019

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

СТУДЕНТ ПОРЫВАЙ П.А.

ГРУППА 8304

ТЕМА РАБОТЫ: СЛУЧАЙНОЕ БИНАРНОЕ ДЕРЕВО ПОИСКА (ТЕКУЩИЙ
КОНТРОЛЬ)

ИСХОДНЫЕ ДАННЫЕ: НА ВХОД ПОДАЮТСЯ СКОБОЧНЫЕ
ПРЕДСТАВЛЕНИЯ БДП, НАПИСАННЫЕ УЧЕНИКОМ

Дата сдачи курсовой:

Дата защиты курсовой:

Студент _____ Порывай. П.А.

Преподаватель _____ Фирсов М.А.

ОГЛАВЛЕНИЕ

Аннотация	3
Введение	4
Цель работы	5
Постановка задачи	5
Спецификация программы	5
Описание структур данных и функций	6
Описание интерфейса пользователя	7
Описание базовых алгоритмов	8
Пример работы программы	9
Тестирование	11
Заключение	14

АННОТАЦИЯ

В данной курсовой работе рассмотрена задача текущего контроля студента. Он должен написать ответы на созданную задачу, и получить результата

Работа выполнена в среде Visual studio.

Программный код, написан на языке C++

В данной программе реализована запись в файл, считывание из него
Результаты выводятся на консоль

Annotation

In this course work considers the task of monitoring the student. He must write the answers to the created task, and get the result

The work was done in Visual studio.

Program code written in C ++

This program implements writing to a file, reading from it

Results are displayed on the console.

ВВЕДЕНИЕ

В данной работе используются алгоритмы вставки элемента в бинарное дерево поиска, его удаления, на них строятся алгоритмы для выполнения задачи:

1 Алгоритм генерации задачи

2 Алгоритм вывода ответов для контроля

Алгоритм генерации задачи – необходим для вывода задачи задачи на построение случайного бдп по заданной последовательности чисел. Числа при каждом вызове программы генерируются разные .

2 Алгоритм вывода ответов для контроля — Фактически является доработанной функцией обхода КЛП дерева

ЦЕЛЬ РАБОТЫ

Написать программу с помощью, которой можно будет генерировать задачи на построение случайных бдп (с различным количеством данных)

Изучить и закрепить знания по теме случайные бинарные деревья поиска

Продемонстрировать работу алгоритмов

Постановка задачи

Случайное бинарное дерево поиска

«Текущий контроль» - создание программы для генерации задачи с ответами и текущий контроль студентов. Задания и ответы должны выводиться в файл в удобной форме.

Спецификация программы

Программа написана на языке C++

Создание и ввод задачи происходит в файл. Студент должен по созданному файлу с задачей написать ответ в другой файл. Программа сравнивает ответ студента с еще одним созданным файлом верных ответов Названия всех файлов задаются как аргументы программы

ОПИСАНИЕ СТРУКТУР ДАННЫХ

Для создания случайного бдп поиска была создана абстрактная структура `node`, которая имеет несколько полей (информацию, количество попыток записи в данный узел, указатели на правое и левое поддерево того же типа `node` – `base` `info`; `int` `count`; `node*` `lt`; `node*` `rt`;

Бинарное дерево поиска создавалось путем создания узла и добавлению к нему элементов с помощью функции **`insert_key()`** (алгоритм для нее описан далее)

Для алгоритма связанного с генерацией чисел для деревьев был создан массив указателей на массивы с целыми числами (матрица задания бинарного дерева поиска) `int **digits`. В конце выполнения алгоритма генерации задания массив освобождается через оператор `delete`. Опишу функцию реализующую алгоритм создания дерева далее

`void gen_tasks_answers(int quant_d, int quant_s, char *answers, char* tasks)` использует базовые алгоритмы создания, удаления, поиска в дереве. Сначала создается матрица `v`, в которую записываются числа сгенерированные посредством функций `srand(int a)`, `rand()` (`srand` генерирует последовательность чисел по заданному аргументу, `rand()` достает из списка сгенерированные числа), после вызывается функция **`printAns(int **digits, int quant_d, int quant_s, char *answer)`**, выводящая ответ в файл под названием указанным в аргументе вызова функции. В конце используем начальный массив чисел для записи задания в файл, освобождаем память (`delete`)

`printAns(int **digits, int quant_d, int quant_s, char *answer)` создает по матрице дерево используя **`insert_key(bt, digits[i][j])`**; в цикле выводит ответ в файл удаляет дерево все это происходит во внешнем цикле

Функции, которые полностью описываются базовыми алгоритмами опишу далее

Интерфейс пользователя

Сначала программа запрашивает количество данных, затем предлагает ввести ответ в текстовый документ, ввести ok

```
Введите количество чисел для каждого случайного бд
3
Введите количество бд
4
Сгенерировано новое задание. Перейдите в файл task2 чтобы узнать его
Ответы запишите построчно в ans_i2 и введите ok
ok
pasha@pasha-VirtualBox:~/ADS-8304/Poryvai/course_work$
```

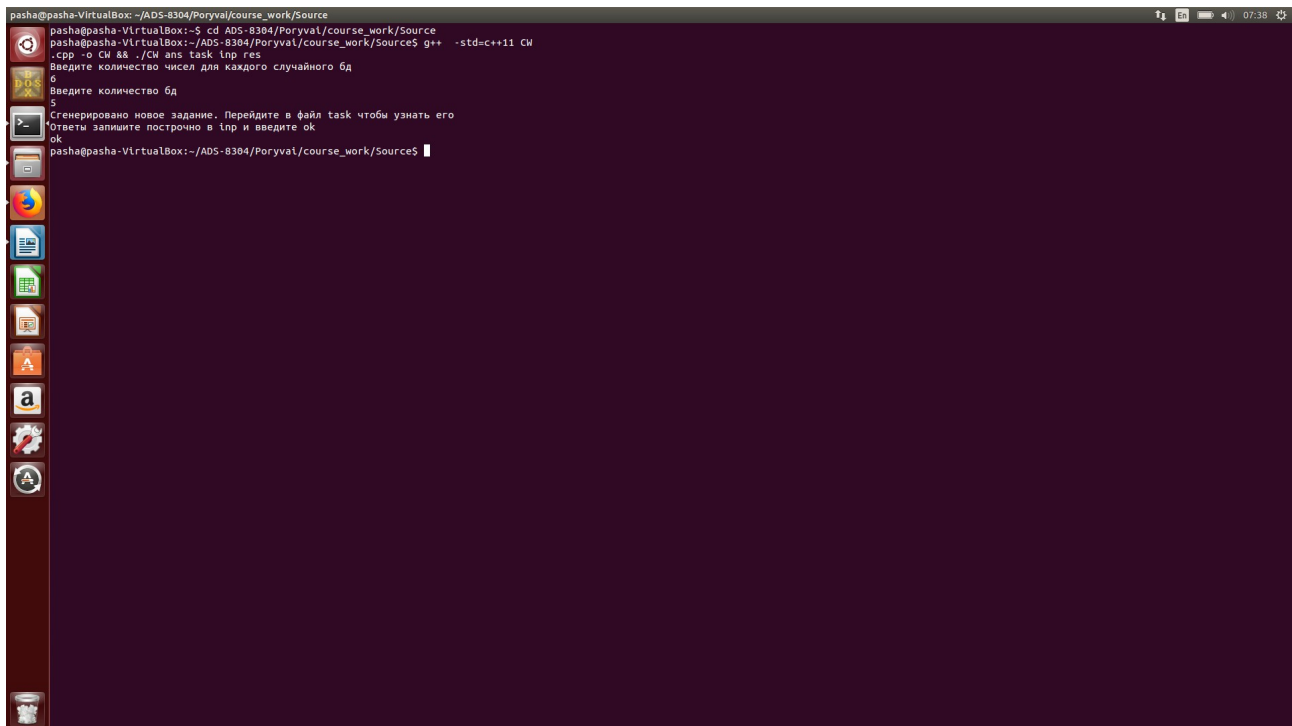

ОПИСАНИЕ БАЗОВЫХ АЛГОРИТМОВ ИСПОЛЬЗУЕМЫХ В ПРОГРАММЕ

Функция **insert_key(binTree &bt, int key)** использует базовый алгоритм - добавляет новый узел в дерево по следующим правилам: у всех узлов *левого* поддерева произвольного узла X значения ключей данных *меньше*, нежели значение ключа данных самого узла, у всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X. Работа этой функции реализована рекурсивно и с помощью конструкции if else ...

Функция **destroy(binTree& b)** — использует алгоритм удаления узлов «в глубину», при каждом вызове вызываем ее же с левым узлом, правым узлом , удаляем узел(delete), реализована также рекурсивно

Функция **printKLP(binTree b, std::ofstream& out)** использует базовый алгоритм обхода дерева (посещается корень дерева левый потомок, правый) при этом, расставляем скобки на каждом уровне глубины вызовов функции, что в итоге даст исчерпывающее текстовое представление случайного бдп

Пример работы программы



```
pasha@pasha-VirtualBox: ~/ADS-8304/Poryvat/course_work/Source
pasha@pasha-VirtualBox:~$ cd ADS-8304/Poryvat/course_work/Source
pasha@pasha-VirtualBox:~/ADS-8304/Poryvat/course_work/Source$ g++ -std=c++11 CW
.cpp -o CW && ./CW ans task lnp res
Введите количество чисел для каждого случайного бд
6
Введите количество бд
5
Сгенерировано новое задание. Перейдите в файл task чтобы узнать его
Ответы запишите построчно в lnp и введите ok
ok
pasha@pasha-VirtualBox:~/ADS-8304/Poryvat/course_work/Source$
```

Файл с ответами

(60(12(41))(89(62(67))))
(12(3)(18(40(21)(97))))
(44(22(2)(39(23)))(55))
(92(32(26)(76(52)))(97))
(11(77(65(30(60)))(88)))

Файл с заданием

Запишите скобочные представления случайных бинарных деревьев поиска заданных следующими последовательностями

60 89 12 62 41 67
12 18 3 40 97 21
44 22 39 23 55 2
92 97 32 26 76 52
11 77 65 88 30 60

Файл с вводом студента

(60(12(41))(89(62(67))))

(12(3)(18(40(21)(97))))

(44(22(2)(39(23)))(55))

(92(32(26 76 52 97))

(11(77(65(30(60)))(88)

Результат

(60(12(41))(89(62(67)))) Верно

(12(3)(18(40(21)(97)))) Верно

(44(22(2)(39(23)))(55)) Верно

(92(32(26 76 52 97)) Не верно

(11(77(65(30(60)))(88) Не верно

Тестирование

Задача

Запишите скобочные представления случайных бинарных деревьев поиска заданных следующими последовательностями

1 64 8 79 10 10 53

44 24 52 39 45 67 62

44 52 21 72 77 50 15

36 59 64 1 89 42 24

24 38 86 77 2 46 8

64 57 62 61 33 14 0

78 33 14 22 38 87 47

15 37 62 3 96 26 56

37 21 80 61 11 18 39

65 65 47 29 22 9 42

55 75 94 85 9 8 59

47 95 58 14 32 20 17

80 99 73 17 72 6 31

35 24 70 0 41 69 81

63 31 24 70 6 70 7

15 31 19 14 26 77 28

11 50 97 43 1 71 13

25 29 44 60 53 66 60

95 35 93 10 18 17 33

25 40 92 92 71 11 7

97 41 35 60 91 85 56

44 56 69 69 37 65 29

90 31 41 37 18 34 0

37 4 85 14 44 77 6

15 41 65 64 82 53 77

Ответы

(1(64(8(10(53)))(79)))
(44(24(39))(52(45)(67(62))))
(44(21(15))(52(50)(72(77))))
(36(1(24))(59(42)(64(89))))
(24(2(8))(38(86(77(46)))))
(64(57(33(14(0)))(62(61))))
(78(33(14(22))(38(47)))(87))
(15(3)(37(26)(62(56)(96))))
(37(21(11(18)))(80(61(39))))
(65(47(29(22(9))(42))))
(55(9(8))(75(59)(94(85))))
(47(14(32(20(17)))(95(58)))
(80(73(17(6)(72(31)))(99))
(35(24(0))(70(41(69))(81)))
(63(31(24(6(7)))(70))
(15(14)(31(19(26(28)))(77)))
(11(1)(50(43(13))(97(71)))
(25(29(44(60(53)(66)))))
(95(35(10(18(17)(33)))(93)))
(25(11(7))(40(92(71))))
(97(41(35)(60(56)(91(85)))))
(44(37(29))(56(69(65))))
(90(31(18(0))(41(37(34)))))
(37(4(14(6)))(85(44(77)))
(15(41(65(64(53))(82(77)))))

Ввод студента

(1(64(8(10(53)))(79)))

(44(24(39))(52(45)(67(62))))
 (44(21(15))(52(50)(72(77))))
 (36(1(24))(59(42)(64(89))))
 (24(2(8))(38(86(77(46))))))
 (64(57(33(14(0)))(62(61))))
 (78(33(14(22))(38(47)))(87))
 (15(3)(37(26)(62(56)(96))))
 (37(21(11(18)))(80(61(39))))
 (65(47(29(22(9))(42))))
 (55(9(8))(75(59)(94(85))))
 (47(14(32(20(17)))(95(58)))
 (80(73(17(6)(72(31)))(99))
 (35(24(0))(70(41(69))(81)))
 (63(31(24(6(7)))(70))
 (15(14)(31(19(26(28)))(77)))
 (11(1)(50(43(13)(97(71))))
 (25(29)(66))))
 (10(18(17)(33)))(93)))
 (5(11(7))(40(92(71))))

(44(37(29))(65)))
 (908(0))(41(37(34))))
 (37(4(14))(85(44(77))))
 (41(65(64(53))(82(77))))

Результат

(1(64(8(10(53)))(79))) Верно
 (44(24(39))(52(45)(67(62)))) Верно
 (44(21(15))(52(50)(72(77)))) Верно
 (36(1(24))(59(42)(64(89)))) Верно
 (24(2(8))(38(86(77(46)))))) Верно

(64(57(33(14(0)))(62(61)))) Верно
 (78(33(14(22))(38(47)))(87)) Верно
 (15(3)(37(26)(62(56)(96)))) Верно
 (37(21(11(18)))(80(61(39)))) Верно
 (65(47(29(22(9))(42)))) Верно
 (55(9(8))(75(59)(94(85)))) Верно
 (47(14(32(20(17)))(95(58))) Верно
 (80(73(17(6)(72(31)))(99)) Верно
 (35(24(0))(70(41(69))(81))) Верно
 (63(31(24(6(7)))(70)) Верно
 (15(14)(31(19(26(28)))(77))) Верно
 (11(1)(50(43(13)(97(71)))) Не верно
 (25(29)(66)))) Не верно
 (10(18(17)(33)))(93)) Не верно
 (5(11(7))(40(92(71)))) Не верно
 Не верно
 (44(37(29))(65)))) Не верно
 (908(0))(41(37(34)))) Не верно
 (37(4(14))(85(44(77)))) Не верно
 (41(65(64(53))(82(77)))) Не верно

Заключение

Были получены навыки, теоретические знания работы со структурой случайного бинарного дерева.

Приложение А

Исходный код

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include<ctime>
#include<string.h>//delay
#include<stdlib.h>

typedef int base;
typedef unsigned int unInt;

struct node {
    base info;
    int count;
    node* lt;
    node* rt;
    // constructor
    node() { lt = nullptr; rt = nullptr; }
};

typedef node* binTree; // "представитель" бинарного дерева

binTree Create(void);
bool isNull(binTree);
base RootBT(binTree); // для непустого бин.дерева
binTree Left(binTree); // для непустого бин.дерева
binTree Right(binTree); // для непустого бин.дерева
binTree ConsBT(const base& x, binTree& lst, binTree& rst);
void destroy(binTree&);

binTree enterBT(std::string str, int& i);
void outBT(binTree b);
void displayBT(binTree b, int n, std::ofstream& out);
unInt sizeBT(binTree b);
//void printKLP(binTree b, std::ofstream& out);

void print_cons_write_file(std::string bin_tree, int i, std::ofstream& out);
```



```

bool check(std::istream& inp);

binTree Create()
{
    return nullptr;
}

//-----

bool isNull(binTree b)
{
    return (b == nullptr);
}

//-----

base RootBT(binTree b)          // для непустого бин.дерева
{
    if (b == nullptr) {

        std::cerr << "Error: RootBT(null) \n"; exit(1);
    }
    else return b->info;
}

//-----

binTree Left(binTree b)          // для непустого бин.дерева
{
    if (b == nullptr) {

        std::cerr << "Error: Left(null) \n"; exit(1);
    }
    else return b->lt;
}

//-----

binTree Right(binTree b)          // для непустого бин.дерева
{
    if (b == nullptr) {

        std::cerr << "Error: Right(null) \n"; exit(1);
    }
    else return b->rt;
}

//-----

```

```

binTree ConsBT(const base& info, binTree& lst, binTree& rst)
{
    binTree p;
    p = new node;
    if (p != nullptr) {
        p->info = info;
        p->lt = lst;
        p->rt = rst;
        return p;
    }
    else {
        std::cerr << "Memory not enough\n"; exit(1);
    }
}
//-----

void destroy(binTree& b)
{
    if (b != nullptr) {

        destroy(b->lt);
        destroy(b->rt);
        delete b;
        b = nullptr;

    }
}

void outBT(binTree b)
{
    if (b != nullptr) {

        std::cout << RootBT(b);
        outBT(Left(b));
        outBT(Right(b));

    }
    else
        std::cout << '/';
}

//-----

void displayBT(binTree b, int n, std::ofstream& out)

```

```

{    // n - уровень узла
    if (b != nullptr) {
        std::cout << ' ' << RootBT(b);
        out << ' ' << RootBT(b);

        if (!isNull(Right(b))) {

            displayBT(Right(b), n + 1, out);
        }
        else {
            std::cout << "\n";
            out << "\n";
        } // вниз
        if (!isNull(Left(b))) {
            for (int i = 1; i <= n; i++) {
                std::cout << " "; // вправо
                out << " ";
            }
            displayBT(Left(b), n + 1, out);
        }
    }
}

//-----

//-----
unInt sizeBT(binTree b)
{
    if (isNull(b))
        return 0;
    else
        return sizeBT(Left(b)) + sizeBT(Right(b)) + 1;
}

//-----
void printKLP(binTree b, std::ofstream& out)
{
    if (!isNull(b)) {

        out<<"( ";
        out << RootBT(b);
    }
}

```

```

        printKLP(Left(b), out);
        printKLP(Right(b), out);
        out<<"");

    }

}

```

```

void insert_key(binTree &bt, int key) {

```

```

    if (bt == nullptr) {

        bt = new node;
        (bt)->info = key;
        bt->lt = nullptr;
        bt->rt = nullptr;
        bt->count = 0;

    }
    else if ((bt)->info > key) {

        insert_key( bt->lt, key);

    }
    else if ((bt)->info < key) {

        insert_key(bt->rt, key);

    }
    else if( bt->info == key )
        bt->count++;
}

```

```

void printAns( int **digits, int quant_d,int quant_s,char *answer ) {

```

```

    int i ,j;

    std::ofstream out;
    out.open(answer, std::ios::app);

    for(i =0; i < quant_s; i++){

        binTree bt = nullptr;

        for(j = 0; j < quant_d; j++)
            insert_key(bt , digits[i][j]);

        printKLP(bt,out);
        out<<"\n";
        destroy(bt);

    }
    out.close();

}

void gen_tasks_answers(int quant_d, int quant_s,char *answers,char* tasks ){

    int i,j;
    int **digits = new int*[quant_s];

    for(i = 0; i < quant_s; i++)
        digits[i] = new int[quant_d];

    srand(time(0));

    for(j = 0; j < quant_s; j++){

        for(i = 0; i < quant_d ; i++)
            digits[j][i] = rand() % 100;

    }
}

```

```

printAns(digits, quant_d, quant_s, answers);

std::ofstream out2(tasks);

out2<<"Запишите скобочные представления случайных бинарных деревьев поиска
заданных следующими последовательностями\n";

for(j = 0; j < quant_s; j++){

    for(i = 0; i < quant_d ; i++)
        out2<<digits[j][i]<<" ";

    out2<<"\n";

}

for(j = 0; j < quant_s; j++)
    delete digits[j];

delete digits;

out2.close();

}

```

```

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

    setlocale(LC_ALL, "Russian");

    int quant_digits, quant_strings;
    std::cout<<"Введите количество чисел для каждого случайного бд\n";

```

```
std::cin>>quant_digits;
std::cout<<"Введите количество бд\n";
std::cin>>quant_strings;
```

```
gen_tasks_answers(quant_digits , quant_strings, argv[1], argv[2]);//1
ответы 2 задание 3 ответ ученика4 результаты
```

```
std::ofstream ans_inp(argv[3]);
ans_inp.close();
```

```
std::cout<<"Сгенерировано новое задание. Перейдите в файл "<<argv[2]<<"
чтобы узнать его\n";
```

```
std::cout<<"Ответы запишите построчно в "<<argv[3]<<" и введите ok\n";
std::string arg;
std::cin>>arg;
if(arg == "ok"){
int i;
```

```
std::ofstream res(argv[4]);
std::ifstream ans_r(argv[1]);
std::ifstream ans_inp(argv[3]);
std::string str1;
std::string str2;
```

```
for(i = 0; i < quant_strings;i++){
```

```
    std::getline(ans_r, str1);
    std::getline(ans_inp, str2);

    if(str1 == str2){
        res<<str2<<" Верно"<<"\n";
    }
    else{
        res<<str2<<" Не верно"<<"\n";
    }
}
```

```
ans_r.close();
ans_inp.close();
```

```
}
```

```
else
    std::cout<<"Аргумент введен неверно\n";

return 0;
}
```