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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Случайные бинарные деревья поиска – текущий контроль

Студент гр. 9304 _____ Силкин В.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Силкин В.А.

Группа 9304

Тема работы : Случайные бинарные деревья поиска – текущий контроль.

Исходные данные:
Случайные БДП – вставка и исключение. Вставка в корень БДП. **Текущий контроль**

Содержание пояснительной записки:
«Содержание», «Введение», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 24 страниц.

Дата выдачи задания: 23.11.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент _____ Силкин В.А.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

Была реализована программа для генерации случайных бинарных деревьев поиска с узлами с целочисленными данными, в них реализована функция вставки, вставки в корень и удаления узлов, в которых находятся указанные значения. Также создаётся 2 файла: с заданиями и ответами на эти задания.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: изучение случайных бинарных деревьев поиска, их реализация с помощью языка программирования C++. Создать функции вставки, вставки в корень и удаления, а также способы генерации заданий и ответы для них.

1. ПОСТАНОВКА ЗАДАЧИ

Вариант 9

Случайные БДП – вставка и исключение. Вставка в корень БДП.

Текущий контроль

«Текущий контроль» - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; все задания должны касаться конкретных экземпляров структуры данных (т.е. не должны быть вопросами по теории); ответы должны позволять удобную проверку правильности выполнения заданий.

2. ОПИСАНИЕ АЛГОРИТМА

2.1. Алгоритм генерации случайного бинарного дерева поиска

Алгоритм составляет бинарное дерево поиска из массива данных. Алгоритм работает следующим образом:

1. Поиск начинается из корня.
2. Если искомое значение ноды меньше, чем в текущей, алгоритм начинается заново из левого потомка текущей ноды.
3. Если искомое значение ноды больше, чем в текущей, алгоритм начинается заново из правого потомка текущей ноды.
4. Если искомое значение ноды равно значению текущей, алгоритм завершается, к количеству найденных значений в этой ноде прибавляется 1.
5. Если алгоритм пришёл в пустую ноду, то создаётся новая нода с искомым значением, количество найденных значений задаётся как 1.

Случайные БДП отличаются тем, что имеют затраты поиска при худшей генерации O(n), средние затраты – $O(\log_2 n)$, но сама генерация дерева довольно быстрая.

3. ВЫПОЛНЕНИЕ РАБОТЫ

3.1. Структуры и классы

Node

В классе хранятся данные узла: его значение, счётчик найденных значений, левый и правый потомки, а также глубина, на которой находится узел.

Tree

Класс, который отвечает за операции с деревьями, а также хранит корень этого дерева.

Control

Класс, отвечающий за текущий контроль, генерирует задания и ответы для этих заданий

3.2. Функции

node()

Задаёт потомков как nullptr, кол-во найденных значений – 0.

void draw_line(int, node*)

Вспомогательный приватный метод для метода horizontal, распечатывает дерево послойно сверху вниз.

int inPKL(node*, ostream&)

Основной метод для рекурсивного выведения дерева на экран. Обходит дерево алгоритмом ПКЛ. Вызывается методом graphPKL, выводит построение в подаваемый поток.

void lkp(node*, ostream&)

Обходит дерево методом ЛКП, и выводит его в подаваемый поток. Вызывается методом LKP.

void searchAndInsert(int, node*&,int)

Метод для построения дерева и включения новых узлов, вызывается конструкторами и insertElem.

void searchAndDelete(int, node*&)

Удаляет ноду, и ставит на её место ноду с максимально близким значением. Вызывается методом deleteElem.

void addlevel(node*)

Вспомогательный метод для insertElemInRoot, добавляет 1 к глубине каждой ноды, начиная с поданой.

void klp(node*, ostream&)

Обходит дерево методом КЛП, и выводит его в подаваемый поток. Вызывается методами KLP.

tree(string)

Генерирует дерево, на основе цифр, которые идут в данной строке.

tree(int)

Генерирует дерево только из корня, и задаёт ему подаваемое значение.

void horizontal()

Послойно выводит дерево в консоль.

void LKP(ostream&)

Выводит ЛКП обход в подаваемый поток.

void insertElem(int)

Ищет или включает элемент в дерево с подаваемым значением.

void deleteElem(int)

Удаляет элемент с подаваемым значением.

void graphPKL(ofstream&)

Выводит дерево в подаваемый поток в наглядном виде.

void graphPKL()

Выводит дерево в поток cout.

void insertElemInRoot(int)

Вставляет элемент с подаваемым значением в корень дерева.

void KLP(ostream&)

Выводит КЛП обход в подаваемый поток.

void KLP()

Выводит КЛП обход в поток cout.

void generateTest(int)

Генерирует случайное из 2 типов заданий и ответ на него. Количество заданий зависит от подаваемого значения. Задания сохраняются в test.txt, ответы в answer.txt.

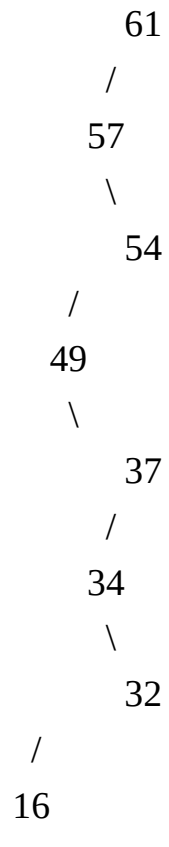
4. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Программа компиллируется утилитой make, при нахождении в папке с файлом Makefile. Во время её работы всплывает диалоговая строка, в которой нужно указать число, для задания количества заданий, или n для завершения программы. Любой другой символ задаст количество заданий как 5.

4.1. Пример 1

Файл test.txt:

1. Сделайте КЛП обход по предоставленному дереву



---

2. Нарисуйте случайное БДП, составленное из ввода чисел: 35 49 91 14 89 67 70 44 87

---

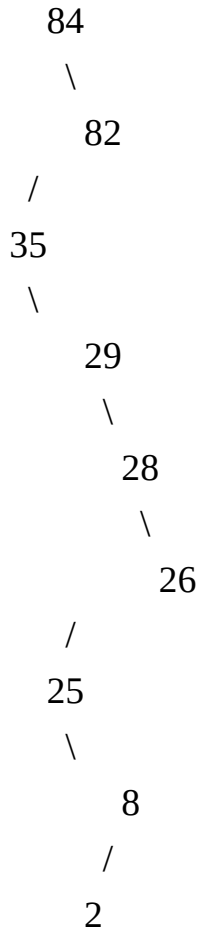
3. Нарисуйте случайное БДП, составленное из ввода чисел: 21 57 20 5 47 87 4 39 36

---

4. Нарисуйте случайное БДП, составленное из ввода чисел: 89 57 54 81 54 47 84 7 6

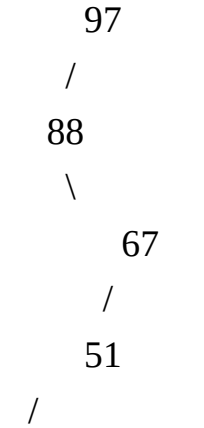
---

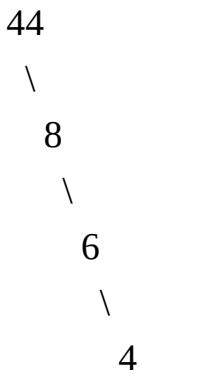
5. Сделайте КЛП обход по предоставленному дереву



---

6. Сделайте КЛП обход по предоставленному дереву



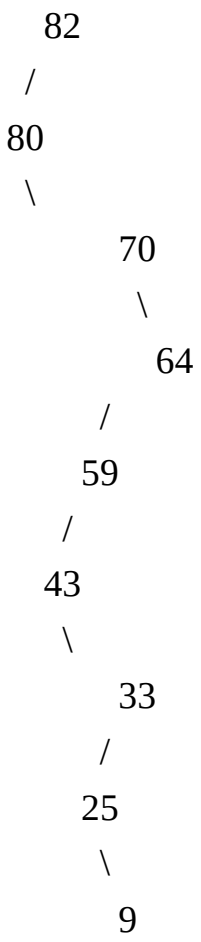


---

7. Нарисуйте случайное БДП, составленное из ввода чисел: 81 29 17 34 89 27 6 60 88

---

8. Сделайте КЛП обход по предоставленному дереву



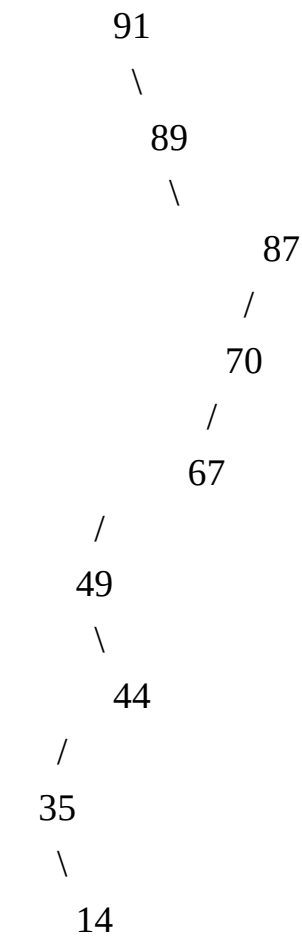
---

9. Нарисуйте случайное БДП, составленное из ввода чисел: 95 3 25 25 1 77 54 69 33

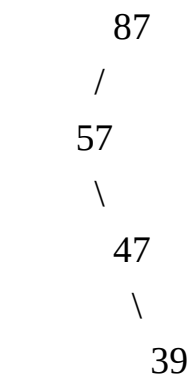
Файл answer.txt:

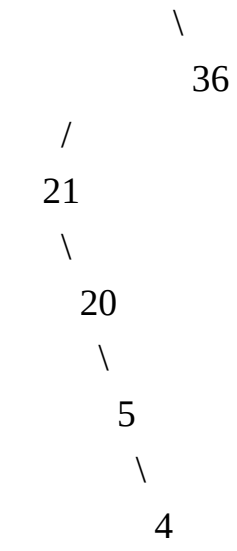
1. 16 49 34 32 37 57 54 61

2.

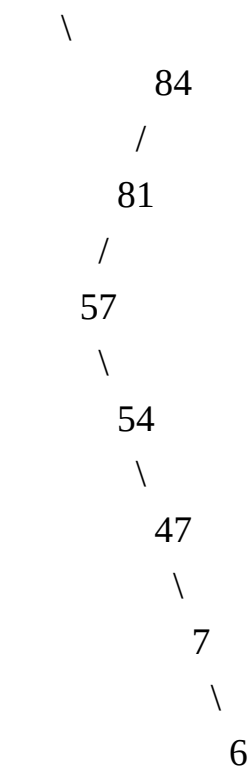


3.





4.



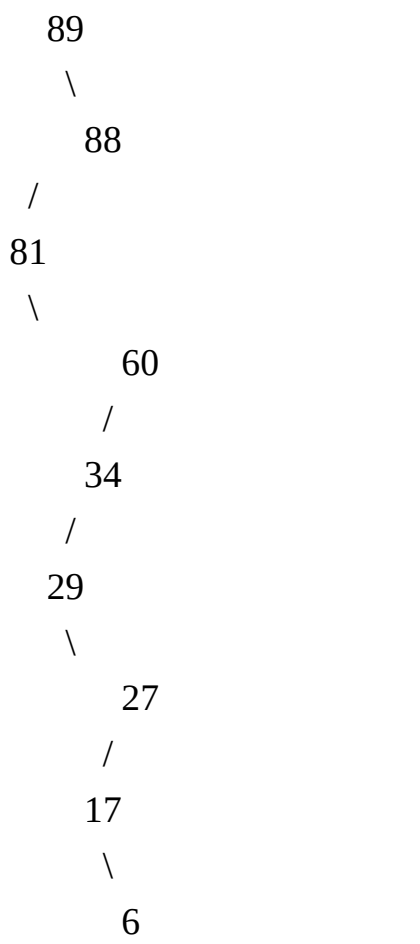
---

5. 35 25 2 8 29 28 26 84 82

---

6. 44 8 6 4 88 51 67 97

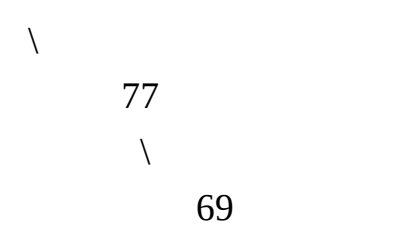
7.

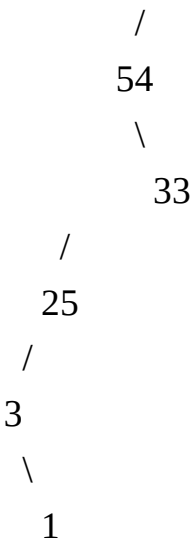


---

8. 80 43 25 9 33 59 70 64 82

9.





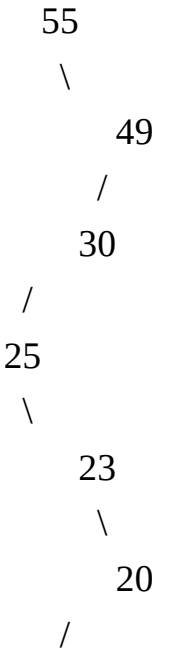
Пример 2.

Файл test.txt

1. Нарисуйте случайное БДП, составленное из ввода чисел: 16 23 2 6 36 14 40 56 64

---

2. Сделайте КЛП обход по предоставленному дереву



19
/
2

Файл answer.txt:

1.

64
/
56
/
40
/
36
/
23
/
16
\
14
/
6
/
2

---

2. 2 25 19 23 20 55 30 49

ЗАКЛЮЧЕНИЕ

Была реализована генерация случайных бинарных деревьев поиска на языке программирования C++, методы вставки и вставки в корень, удаления, а также функции, обеспечивающие генерацию вариантов текущего контроля.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Случайные двоичные деревья
   https://ru.qaz.wiki/wiki/Random_binary_tree

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл main.cpp

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

class node{
public:
    int value;
    int count;
    int level;
    node* left;
    node* right;
    node(): count(0), left(nullptr), right(nullptr){}
};


class tree{

    int max_lvl;

    void draw_line(int current_line, node* cur) {
        if(cur->level == current_line) {
            std::cout << " ";
            std::cout << cur->value;
        }
        if(cur->left != nullptr) draw_line(current_line, cur->left);
        if(cur->right != nullptr) draw_line(current_line, cur->right);
    }

    void lkp(node* cur, std::ostream &outstream) {
        if(cur == nullptr) return;
        lkp(cur->left, outstream);
        outstream << cur->value << " ";
        lkp(cur->right, outstream);
```

```
}

void searchAndInsert(int elem, node*& ptr, int level) {
    if (ptr == nullptr) {
        ptr = new node;
        ptr->value = elem;
        ptr->count = 1;
        ptr->level = level;
        if(level > max_lvl) {
            max_lvl = level;
        }
    } else if (elem < ptr->value) {
        searchAndInsert(elem, ptr->left, level+1);
    } else if (elem > ptr->value) {
        searchAndInsert(elem, ptr->right, level+1);
    } else {
        ptr->count++;
    }
}

void searchAndDelete(int elem, node*& ptr) {
    if(ptr) {
        if(elem == ptr->value) {
            if(ptr->right) {
                node* tmp = ptr->right;
                while(tmp->left) {
                    tmp = tmp->left;
                }
                ptr->value = tmp->value;
                ptr->count = tmp->count;
                searchAndDelete(tmp->value, tmp);
            } else if(ptr->left) {
                node* tmp = ptr->left;
                while(tmp->right) {
                    tmp = tmp->right;
                }
                ptr->value = tmp->value;
                ptr->count = tmp->count;
```

```
                    searchAndDelete(tmp->value, tmp);
                } else {
                    delete ptr;
                }
            } else if(elem < ptr->value) {
                searchAndDelete(elem, ptr->left);
            } else if(elem > ptr->value) {
                searchAndDelete(elem, ptr->right);
            }
        } else {
            std::cout << "Element \"" << elem << "\" not found.";
        }
    }

public:

    node *head;

    tree(std::string st): head(nullptr), max_lvl(1) {
        for(auto iter = st.begin(); iter!=st.end(); iter++) {
            char a = *iter;
            searchAndInsert(atoi(&a), head, 1);
        }
    }

    void horizontal() {
        for(int i = 1; i <= max_lvl; i++) {
            draw_line(i, head);
            std::cout << "\n";
        }
    }

    void LKP(std::ostream &outstream) {
        lkp(head, outstream);
        outstream << "\n";
    }

    void insertElem(int elem) {
```

```
        searchAndInsert(elem, head, 1);
    }

    void deleteElem(int elem) {
        searchAndDelete(elem, head);
    }
};

int main() {
    std::string st;
    std::getline(std::cin, st);
    tree sap(st);
    std::ofstream fname;
    fname.open("out.txt");
    sap.LKP(fname);
    fname.close();
    sap.horizontal();
    std::cout << '\n';
    sap.deleteElem(5);
    sap.horizontal();
    return 0;
}
```