МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»
КАФЕДРА № 52

ПРЕПОДАВАТЕЛЬ

Доцент, канд. техн. наук		Линский Е.М.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О КУРСОВОЙ РАБОТЕ

по курсу: ОСНОВЫ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №	5022	04.12.2021	Штеле Б.Э.
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2020

Оглавление

Постановка задачи 3

Алгоритм 4

Пошаговый пример для алгоритма 5

Псевдокод реализации алгоритма 8

Инструкция пользователя 9

Тестовые примеры 10

Список литературы 12

Постановка задачи

Задачей данной курсовой работы является разработка программы, которая реализует алгоритм построения дерева поиска (оно может быть несбалансированным). Программа из текстового файла считывает набор чисел, а с клавиатуры получает число для поиска.

Алгоритм

Бинарные деревья-представляют собой структуру данных, которые поддерживают многие операции с динамическими множествами, включая поиск элемента, минимального или максимального значения и т.д. Основные операции в бинарном дереве поиска выполняются за время, пропорциональное его высоте. Для полного бинарного дерева с n узлами эти операции выполняются за время O(log n) в худшем случае, данная сложность справедлива для сбалансированных деревьев.

Такое дерево может быть представлено при помощи связной структуры данных, в которой каждый узел является объектом. В нем присутствуют поля key-ключ, left-левый дочерний узел, right-правый дочерний узел. Если дочерние узлы отсутствуют, то заданным полям присваивается значение nullptr. Ключи в бинарном дереве поиска хранятся таким образом, чтобы в любой момент удовлетворять **следующему свойству бинарного дерева**

Если узел **x** – узел бинарного дерева поиска, а узел **y** находится в левом поддереве **x**, то **key[y] < key[x].** Если узел **y** находится в правом поддереве **x**, то **key[x]<key[y].**

Рассмотрим реализацию бинарного дерева поиска на языке C++.

```
class BinaryTree
{
private:
    struct Node
    {
        float x;
        Node* l = nullptr;
        Node* r = nullptr;
    };
    int countTr;
    void del(Node*& Tree);
    void push(float x, Node*& Tree);
    Node* Tree;
public:
    void CreateTree(std::string a);
    BinaryTree();
    ~BinaryTree();
    bool find_el(float x);
};
```

Для этого нам потребуется создать класс BinaryTree со следующими полями.

Приватные поля: Node-узел бинарного дерева поиска со следующими полями: x – ключ, l – левый дочерний узел, r – правый дочерний узел, countTr – количество элементов в дереве, функция del – удаляет узел, функция push – вставляет узел, Tree – корень дерева поиска.

Публичные поля: CreateTree – строит дерево с помощью чисел хранящихся в файле, BinaryTree – конструктор, ~BinaryTree – деструктор, функция find_el – реализует поиск заданного элемента в дереве.

Рассмотрим алгоритм построения бинарного дерева поиска. В функцию CreateTree строка std::string в которой содержится путь к файлу.Преобразовав число из типа string в тип float, вызывается функция push которая ставит элемент в дерево в соответствии с правилом бинарного дерева поиска, при этом первый элемент всегда становится корнем дерева (верно для несбалансированного дерева).

Рассмотрим алгоритм поиска элемента. В функцию find_el передается число. Затем число сравнивается с ключом корня дерева если число больше ключа, следовательно, нужно проверять правое поддерево, если меньше, то левое. Так продолжается до тех пор, пока заданное число не будет найдено или мы не дойдем до конца дерева.

Пошаговый пример для алгоритма

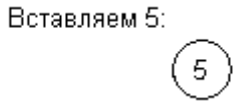


Рис. 1

Рассмотрим построение бинарного дерева поиска и поиск заданного элемента на примере на примере. На Рис.1. показано как будет выглядеть дерево после вставки первого элемента – корня.

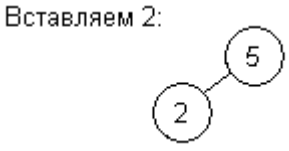


Рис. 2

После вставки второго элемента дерево примет следующий вид, Рис. 2. Число 2 меньше чем число 5, поэтому 2 становится левым сыном 5.

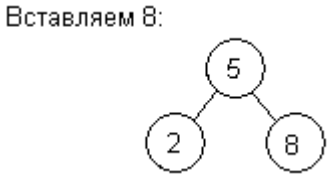


Рис. 3

Число 8 сравнивается с 5, 8 больше 5, поэтому 8 становится правым сыном 5.

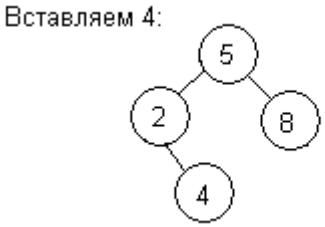
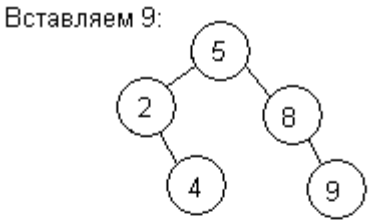
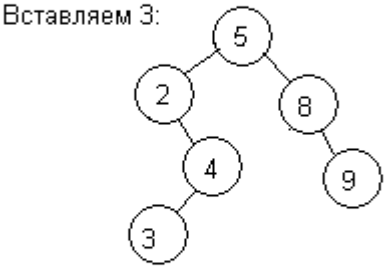


Рис. 4

Число 4 сравнивается со значением корня, заданное число меньше, поэтому сравниваем его с левым поддеревом. 4>2, далее узлов нет, значит 4 становится правым сыном двойки. Рис.4





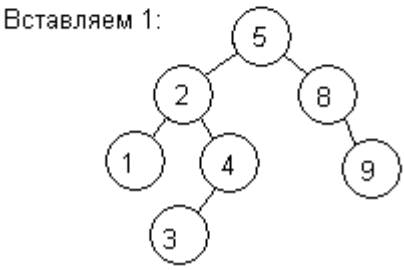


Рис. 5

Далее аналогично описанному выше способу достраиваем дерево до состояния показанного на Рис. 5

Рассмотрим алгоритм поиска. На вход поступает число 3.

Красный цвет – элемент больше заданного.

Желтый цвет – элемент меньше заданного.

Зеленый цвет – элемент найден.

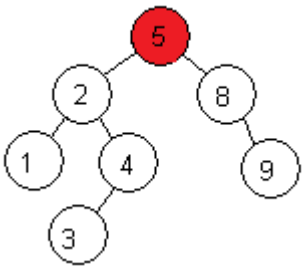


Рис. 6

Сравниваем числа 3 и 5, 3 < 5, значит идем проверять левое поддерево.

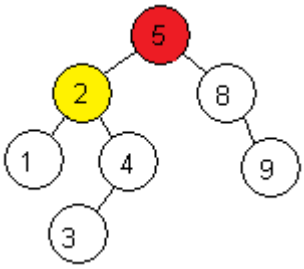


Рис. 7

Сравниваем числа 3и 2, 3> 2, значит идем проверять правое поддерево.

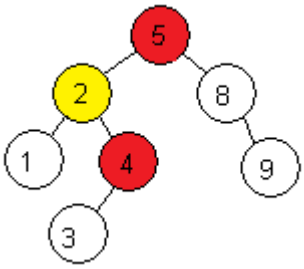


Рис. 8

Сравниваем 3 и 4, 3 < 4, продолжаем поиск в левом поддереве.

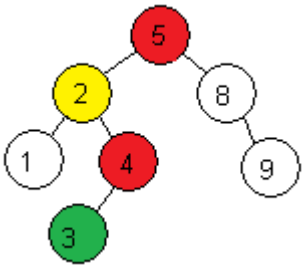


Рис. 9

Сравниваем 3 и 3, элемент найден.

Псевдокод реализации алгоритма поиска

```
bool BinaryTree::find_el(float искомое число) {
        if (количество чисел в дереве != 0) {
                Node* t;
                t = Tree;
                for (;;) {
                        if (искомое число == текущее числовое значение в Node) {
                                return true;\\Нашли элемент
                        }
                        if (искомое число > текущего значения в дереве) {
                                if (правое поддерево == nullptr)\\Не нашли элемент
                                        return false;
                                t = адрес правого поддерева;\\
                        }
                        else {
                                if (левое поддерево == nullptr)\\Не нашли элемент
                                        return false;
                                t = адрес левого поддерева;
                        }
                }
        }
        else
                throw std::logic_error("Its is empty");
}
```

Инструкция пользователя

Вначале пользователю необходимо создать входной файл, содержащий любые числа. Далее при запуске программы пользователь указывает путь к файлу. После этого пользователю нужно будет ввести искомое число. Затем программа выведет на экран результат поиска и пользователю будет предложено продолжить поиск или завершить программу.

Тестовые примеры

1. Входные данные:

number.txt:

2 34 56 32 57 -2.01 -2

45 24 -24

Результаты:

```
Enter the path to the file
C:\Users\ACER\source\repos\BinaryTreeSearch\Debug\number.txt
Enter your element
12
Element not found
Do you want to repeat? 1-yes 2-no
```

```
Do you want to repeat? 1-yes 2-no
1

Enter your element
-2
Element found
Do you wont to repeat? 1-yes 2-any number
```

```
Enter your element
-2.01
Element found
Do you wont to repeat? 1-yes 2-any number
```

2. Входные данные:

number.txt:

44 524 32 76 34 67 3 0 -2.5

56 23 968 40 49 346 349 604 332

454 654 675 96 4 455

Результаты:

```
C:\Users\ACER>C:\Users\ACER\source\repos\BinaryTreeSearch\Debug\BinaryTreeSearch.exe
Enter the path to the file
C:\Users\ACER\source\repos\BinaryTreeSearch\Debug\number.txt
Enter your element
455
Element found
Do you wont to repeat? 1-yes 2-any number
```

```
Enter your element
-2.5
Element found
Do you wont to repeat? 1-yes 2-any number
```

```
Do you wont to repeat? 1-yes 2-any number
1

Enter your element
1
Element not found
Do you want to repeat? 1-yes 2-no
```

3. Входные данные:

number.txt:

65 32 67 -34 12 56

123 55 653 6

2332 65

3

5

6

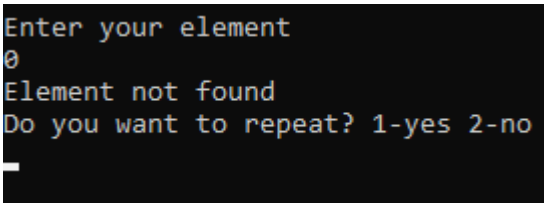
2000

Результаты:

```
Enter the path to the file
C:\Users\ACER\source\repos\BinaryTreeSearch\Debug\number.txt
Enter your element
3
Element found
Do you wont to repeat? 1-yes 2-any number
```

```
Do you wont to repeat? 1-yes 2-any number
1

Enter your element
2000
Element found
Do you wont to repeat? 1-yes 2-any number
```



Список литературы

а. Т. Кормен, Ч. Лейзерсон, Р. Ривест, «Алгоритмы: построение и анализ»
б. О.В. Сенюкова «Сбалансированные деревья поиска»