

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: «Бинарное дерево поиска»

Студент гр. 9382

Сорокумов С.В.

Преподаватель

Фирсов М. А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Сорокумов С.В.

Группа 9382

Тема работы: Обработка текста

Исходные данные:

"Демонстрация" - визуализация структур данных, алгоритмов, действий.

Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

Содержание пояснительной записки:

«Содержание», «Введение», «Структура программы», «Тестирование», «Выводы»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 14.12.2019

Дата защиты реферата: 16.12.2019

Студент		Сорокумов С.В.
Преподаватель		Фирсов М. А.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, которая строит бинарное дерево поиска с рандомизацией. Также был приведён подробный разбор постройки дерева, каждый шаг программы выводится в консоль для наглядного разбора.

Основной код программы приведён в приложении А.

SUMMARY

In this course work, a program was implemented that builds a binary search tree with randomization. Also, a detailed analysis of the construction of a tree was given, each step of the program is displayed in the console for visual analysis.

The main program code is given in Appendix A.

СОДЕРЖАНИЕ

Введение	5
Задание	6
Описание структур данных	7
Описание алгоритма	8
Описание интерфейса пользователя	9
Тестирование	19
Вывод	21
Приложение А	22

ВВЕДЕНИЕ

Цель работы.

Ознакомиться с такой структурой данных, как бинарное дерево поиска, и научиться применять БДП на практике.

Основные теоретические положения.

Бинарное дерево поиска (БДП) — это бинарное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

1. Оба поддерева — левое и правое — являются БДП.
2. У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X .
3. 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 нестрого меньше ключей данных правого.

Пример БДП представлен на рис. 1.

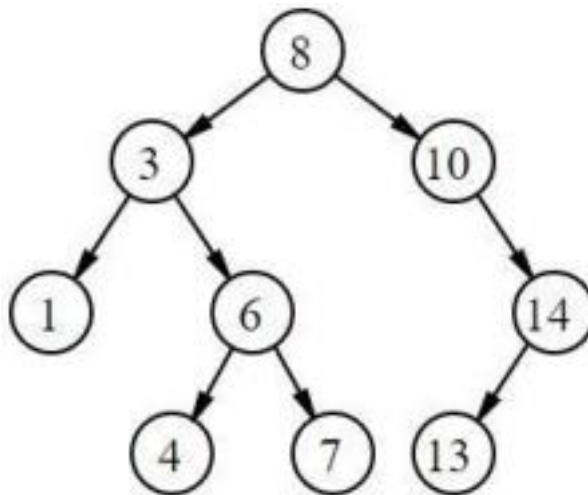


Рисунок 1 – Пример БДП

Задание.

Вариант 10.

Случайные БДП с рандомизацией 1+2а. Демонстрация.

1) По заданному файлу F (типа *file of Elem*), построить БДП определённого типа;

2) Для построенного БДП проверить, входит ли в него элемент e типа *Elem*, и если не входит, то добавить элемент e в дерево поиска.

Демонстрация - визуализация структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

Пояснение задания.

На вход программе подаётся файл со случайной последовательностью символов (ASCII). Требуется: построить случайное БДП с рандомизацией, для построенного БДП проверить, входит ли в него элемент e типа *Elem*, и если не входит, то добавить элемент e в дерево поиска, а также должна присутствовать визуализация структур данных, алгоритмов, действий. Алгоритм действий программы должен быть описан подробно и понятно (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

Описание структур данных

`struct node` - структура для представления узлов дерева, которая включает в себя.

`int key;` - целочисленная переменная, отвечающая за ключ элемента.

`int size;` - целочисленная переменная, отвечающая за размер элемента.

`node* left;` - указатель на левое поддерево.

`node* right;` - указатель на правое поддерево.

`node(int k)` - конструктор ноды.

`node* find(node* p, int k)` - рекурсивно реализует алгоритм поиска.

Принимает указатель на ноду (`node* p`) и целочисленный ключ (`int k`).

Возвращает указатель на ноду.

`node* insert(node* p, int k)` - рекурсивно реализует алгоритм вставки.

Принимает указатель на ноду (`node* p`) и целочисленный ключ (`int k`).

Возвращает указатель на ноду.

`int getsize(node* p)` - функция получения размера дерева.

Принимает указатель на ноду.

Возвращает целочисленное значение - размер.

`void fixsize(node* p)` - функция установление корректного размера дерева.

Принимает указатель на ноду.

`node* rotateright(node* p)` - функция поворота направо относительно узла.

Принимает указатель на ноду.

Возвращает указатель на ноду.

`node* rotateleft(node* q)` - функция поворота налево относительно узла.

Принимает указатель на ноду.

Возвращает указатель на ноду.

`node* insertroot(node* p, int k)` - вставка нового узла в корень дерева.

Принимает указатель на ноду и целочисленное значение - ключ.

Возвращает указатель на ноду.

`node* insert(node* p, int k)` - рандомизированная вставка нового узла в корень дерева.

Принимает указатель на ноду и целочисленное значение - ключ.

Возвращает указатель на ноду.

Описание алгоритма.

На вход программа ожидает файл со случайной последовательностью символов (ASCII). Программа пробует открыть файл. Если это не удастся, выводится сообщение об ошибке и программа экстренно завершается.

Создается элемент структуры. Управление элементами структуры в главном файле будет производиться посредством функция, находящиеся в `binSTree.h`. На вход которым будут подаваться только элементы для добавления в БДП.

Начинается посимвольное считывание из входного файла. Параллельно выводятся некоторые поясняющие сообщения о ходе работы алгоритма, плюс на консоль выводится само БДП (здесь можно увидеть уровни дерева, правые и левые поддеревья, количество узлов поддерева напротив конкретного

элемента).

Построение дерева подставляет собой псевдослучайный процесс (рандомизированный). Суть рандомизации заключается в том, чтобы при добавлении очередного элемента помещать его либо в корень дерева/поддерева (рекурсивно), либо в его законное место в БДП. При этом обход ЛКП всегда выдает алфавитную последовательность.

Описание интерфейса пользователя

Поскольку основной упор в работе был сделан на визуализацию алгоритма, стоит продемонстрировать его работу на одном из тестов.

Возьмем Test: hdmaf!123456.

```
Введите f если хотите загрузить из файла, иначе любой другой символ для работы с консолью (Только латинские символы)
h
Введите количество элементов
12
Введите данные
hdmaf!123456

Шаг 1:
Получен элемент h
Вставка согласно БДП!
*****
Дерево после 1 шага:
> h

Шаг 2:
Получен элемент d
'd' < 'h'
Переход <-
Вставка согласно БДП!
*****
Дерево после 2 шага:
> h
└─▶ d

Шаг 3:
Получен элемент m
'm' > 'h'
Переход ->
Вставка согласно БДП!
*****
```

Рисунок 1. Ввод данных и начало построения (шаги 1-3).

```

*****
Дерево после 3 шага:
> h
  > m
  > d

Шаг 4:
Получен элемент a
'a' < 'h'
Переход <-
'a' < 'd'
Переход <-
Вставка согласно БДП!
*****
Дерево после 4 шага:
> h
  > m
  > d
  > a

Шаг 5:
Получен элемент f
'f' < 'h'
Переход <-

Выпал шанс рандомизации, вставляю в корень, на место 'd'
Для этого вставляю элемент в БДП, а затем буду перемещать его при помощи вращений к корню.
Поддерево с вершиной 'd' до поворота:
> d
  > f
  > a
*****rotatel*****
Корнем становится корень правого поддерева

```

Рисунок 2. Построение дерева (шаг 4-5).

```

Корнем становится корень правого поддерева.
Предыдущий корень перемещается на место корня левого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место правого корня предыдущего корня.
*****rotatel*****
После:
> f
  > d
  > a
*****
Дерево после 5 шага:
> h
  > m
  > f
  > d
  > a

Шаг 6:
Получен элемент !
'!' < 'h'
Переход <-
'!' < 'f'
Переход <-
'!' < 'd'
Переход <-
'!' < 'a'
Переход <-
Вставка согласно БДП!
*****
Дерево после 6 шага:
> h
  > m
  > f
  > d
  > a

```

Рисунок 3. Построение дерева (шаг 6).

```

дерево после 6 шага:
> h
> m
> f
> d
> a
> !

Шаг 7:
Получен элемент 1
'1' < 'h'
Переход <-

Выпал шанс рандомизации, вставляю в корень, на место 'f'
Для этого вставляю элемент в БДП, а затем буду перемещать его при помощи вращений к корню.
Поддерево с вершиной '!' до поворота:
> !
> 1
*****rotateL*****
Корнем становится корень правого поддерева.
Предыдущий корень перемещается на место корня левого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место правого корня предыдущего корня.
*****rotateL*****
После:
> 1
> !
Поддерево с вершиной 'a' до поворота:
> a
> 1
> !
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> a
> !
Поддерево с вершиной 'd' до поворота:
> d
> 1
> a
> !
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> d
> a
> !
Поддерево с вершиной 'f' до поворота:
> f
> 1
> d
> a
> !
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> f
> d
> a
> !

```

Рисунок 4. Начало шага 7.

```

Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> a
> !
Поддерево с вершиной 'd' до поворота:
> d
> 1
> a
> !
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> d
> a
> !
Поддерево с вершиной 'f' до поворота:
> f
> 1
> d
> a
> !
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдщем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> f
> d
> a
> !

```

Рисунок 5. Окончание шага 7.

```

предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
> 1
> f
> |
> L d
> |
> L a
> |
> !
*****
Дерево после 7 шага:
> h
> m
> |
> 1
> f
> |
> L d
> |
> L a
> |
> !
Шаг 8:
Получен элемент 2
'2' < 'h'
Переход <-
'2' > '1'
Переход ->
'2' < 'f'
Переход <-
'2' < 'd'
Переход <-
'2' < 'a'
Переход <-
Вставка согласно БДП!

```

Рисунок 6. Визуализация дерева после шага 7.

```

Вставка согласно БДП!
*****
Дерево после 8 шага:
> h
> m
> |
> 1
> f
> |
> L d
> |
> L a
> |
> 2
> |
> !
Шаг 9:
Получен элемент 3
'3' < 'h'
Переход <-
Выпал шанс рандомизации, вставляю в корень, на место '1'
Для этого вставляю элемент в БДП, а затем буду перемещать его при помощи вращений к корню.
Поддерево с вершиной '2' до поворота:
> 2
> 3
> |
> 3
*****rotateL*****
Корнем становится корень правого поддерева.
Предыдущий корень перемещается на место корня левого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место правого корня предыдущего корня.
*****rotateL*****
После:
> 3
> 2
> |
> a
> |
> 3
Поддерево с вершиной 'a' до поворота:
> a
> |
> 3

```

Рисунок 7. Визуализация дерева после шага 8, начало шага 9.

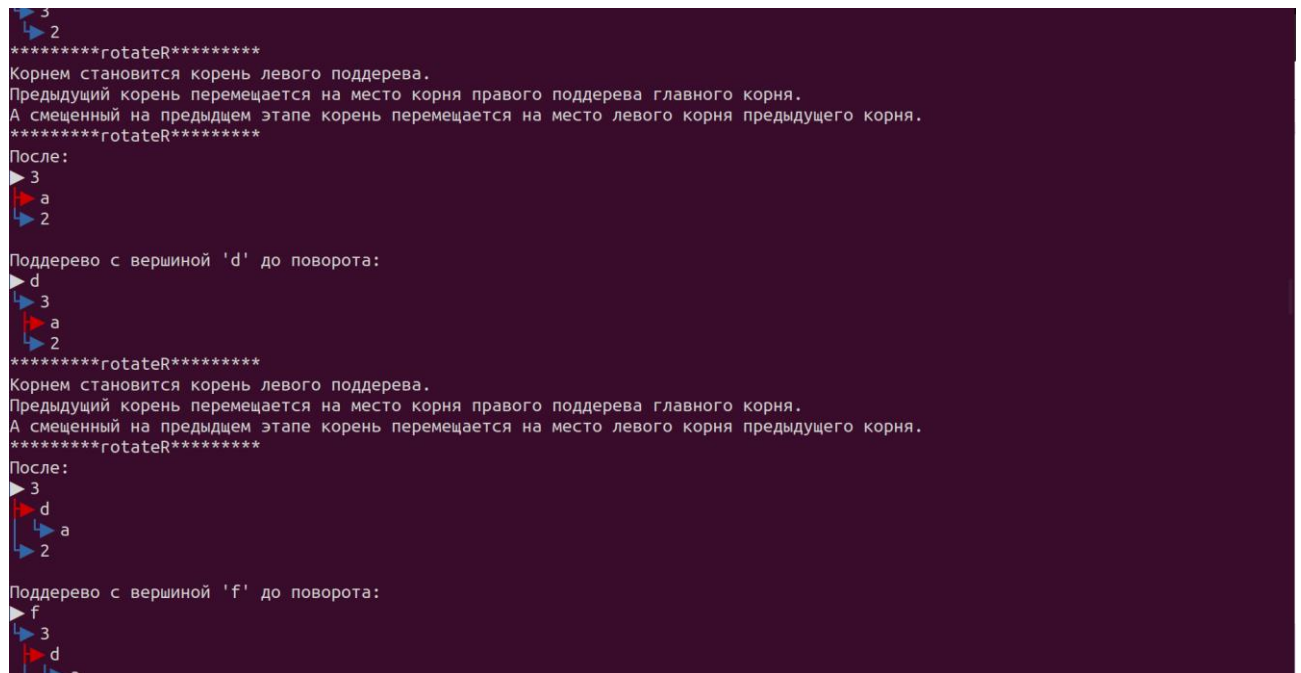


Рисунок 8. Построение дерева на шаге 9.

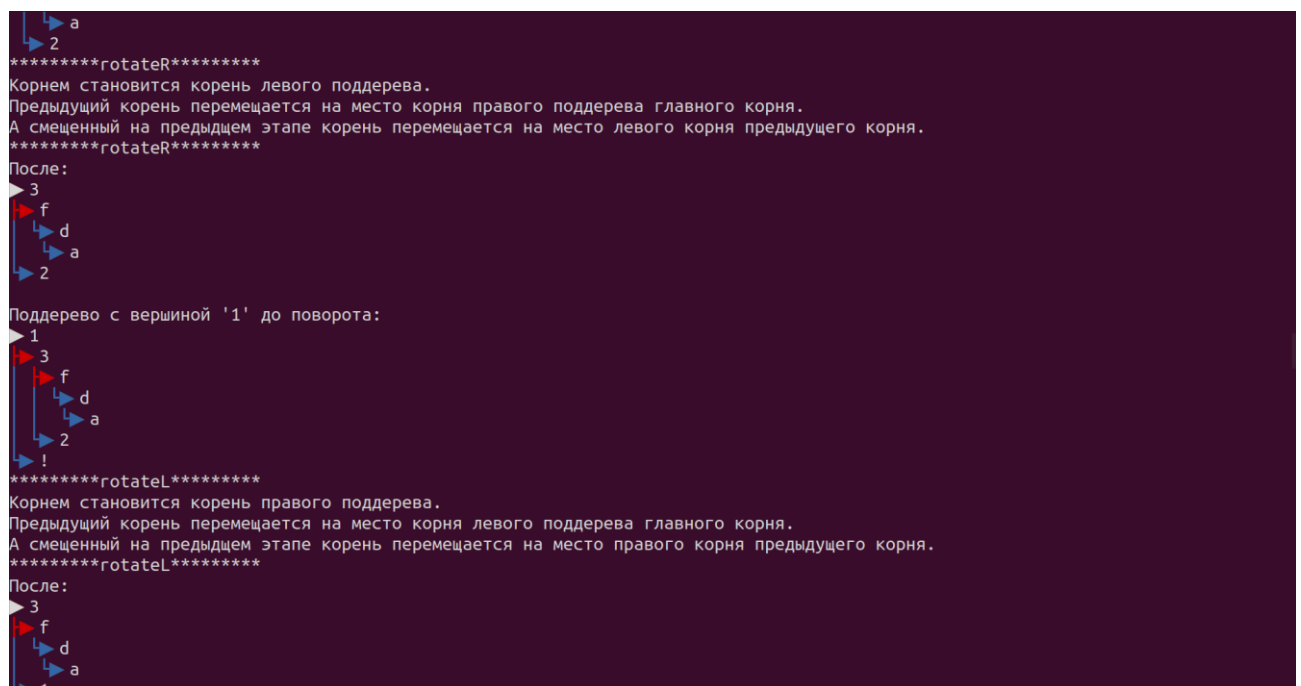


Рисунок 9. Продолжение шага 9.



Рисунок 10. Визуализация дерева после шага 9, шаг 10.

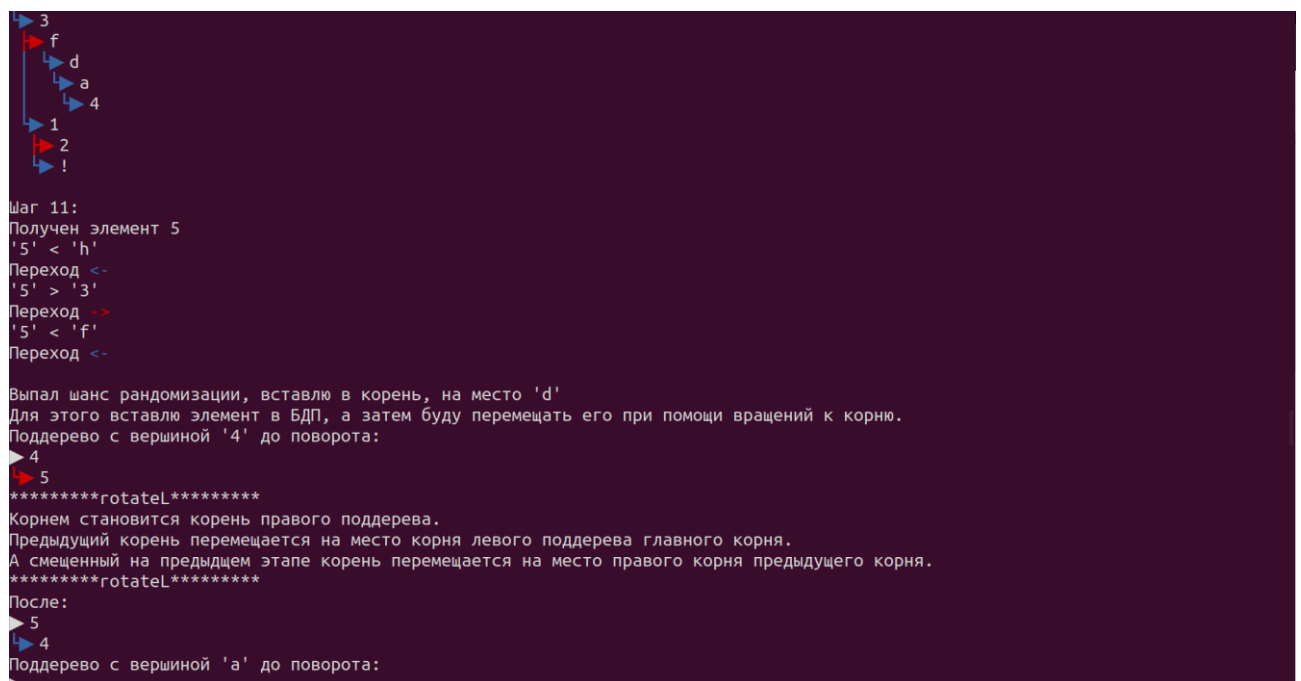


Рисунок 11. Визуализация дерева после шага 10, начало шага 11.

```

a
└─ 5
  └─ 4
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
5
└─ a
  └─ 4

Поддерево с вершиной 'd' до поворота:
d
└─ 5
  └─ a
    └─ 4
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
5
└─ d
  └─ a
    └─ 4

*****
Дерево после 11 шага:
h
└─ m
  └─ 3
    └─ f
      └─ 5
        └─ d
          └─ a
            └─ 4
        └─ 1
          └─ 2
            └─ !

```

Рисунок 12. Окончание шага 11.

```

*****
Дерево после 11 шага:
h
└─ m
  └─ 3
    └─ f
      └─ 5
        └─ d
          └─ a
            └─ 4
        └─ 1
          └─ 2
            └─ !

Шаг 12:
Получен элемент 6

Выпал шанс рандомизации, вставлю в корень, на место 'h'
Для этого вставлю элемент в БДП, а затем буду перемещать его при помощи вращений к корню.
Поддерево с вершиной 'a' до поворота:
a
└─ 6
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
6
└─ a

Поддерево с вершиной 'd' до поворота:
d

```

Рисунок 13. Визуализация дерева после шага 11, начало шага 12.

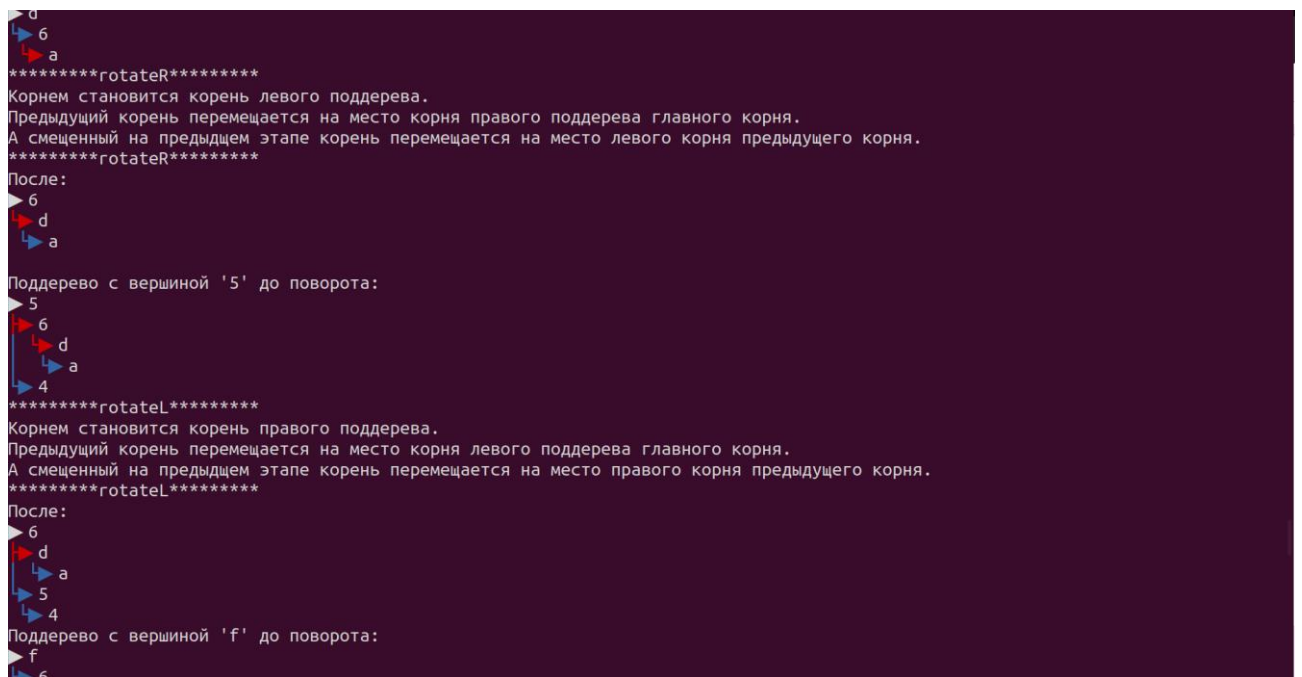


Рисунок 14. Продолжение шага 12.

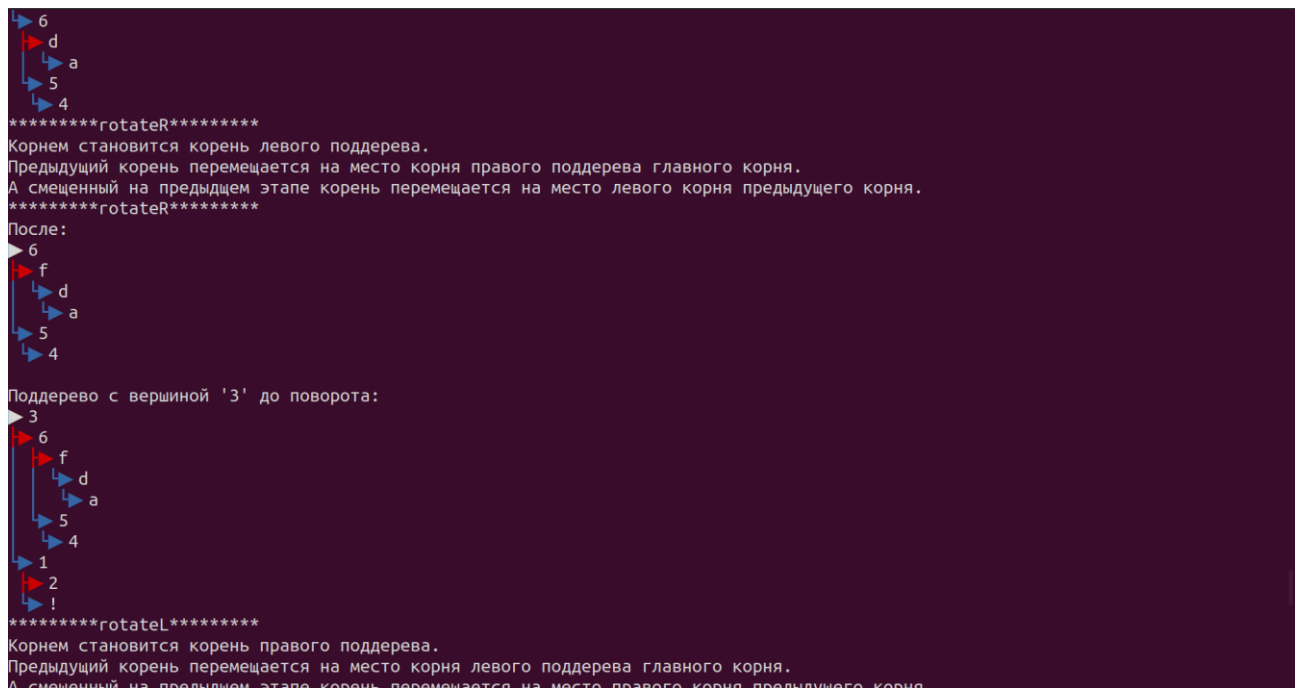


Рисунок 15. Продолжение шага 12.

```

А смещенный на предыдущем этапе корень перемещается на место правого корня предыдущего корня.
*****rotateL*****
После:
6
└─ f
   └─ d
      └─ a
         └─ 3
            └─ 5
               └─ 4
                  └─ 1
                     └─ 2
                        └─ !
Поддерево с вершиной 'h' до поворота:
h
└─ m
   └─ 6
      └─ f
         └─ d
            └─ a
               └─ 3
                  └─ 5
                     └─ 4
                        └─ 1
                           └─ 2
                              └─ !
*****rotateR*****
Корнем становится корень левого поддерева.
Предыдущий корень перемещается на место корня правого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место левого корня предыдущего корня.
*****rotateR*****
После:
6
└─ h
   └─ m
      └─ f
         └─ d
            └─ a
               └─ 3
                  └─ 5
                     └─ 4
                        └─ 1
                           └─ 2
                              └─ !

```

Рисунок 16. Продолжение шага 12.

```

6
└─ h
   └─ m
      └─ f
         └─ d
            └─ a
               └─ 3
                  └─ 5
                     └─ 4
                        └─ 1
                           └─ 2
                              └─ !
*****
Дерево после 12 шага:
6
└─ h
   └─ m
      └─ f
         └─ d
            └─ a
               └─ 3
                  └─ 5
                     └─ 4
                        └─ 1
                           └─ 2
                              └─ !
Дерево из консоли построено.
Всего символов прочитано: 12
Высота дерева = 5
Какой элемент найти?

```

Рисунок 17. Визуализация дерева после шага 12.

```

какой элемент найти?
0
Такого элемента нет в дереве
Произвожу вставку элемента 0 в дерево
'0' < '6'
Переход <-
'0' < '3'
Переход <-
'0' < '1'
Переход <-

Выпал шанс рандомизации, вставлю в корень, на место '!'
Для этого вставлю элемент в БДП, а затем буду перемещать его при помощи вращений к корню.
Поддереву с вершиной '!' до поворота:
> !
  0
*****rotatel*****
Корнем становится корень правого поддерева.
Предыдущий корень перемещается на место корня левого поддерева главного корня.
А смещенный на предыдущем этапе корень перемещается на место правого корня предыдущего корня.
*****rotatel*****
После:
> 0
  !
Дерево после добавления:
  6
  h
  m
  f
  d
  a
  3
  5
  4
  1
  2
  0
  !

```

Рисунок 18. Работа с пользователем.

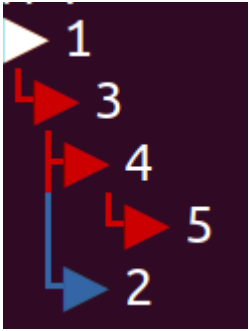
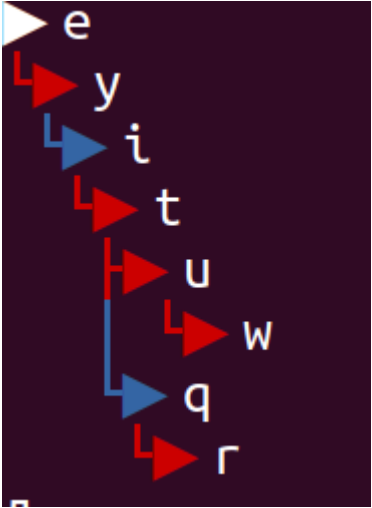
```

После:
> 0
  !
Дерево после добавления:
  6
  h
  m
  f
  d
  a
  3
  5
  4
  1
  2
  0
  !
Хотите ли продолжить работу с деревом? у - да, н - завершить программу

```

Рисунок 19. Визуализация дерева после работы с пользователем.

Тестирование.

№	Входные данные:	Результат:
1	12345	
2	qwertyui	

3	qwertyui12345678	
4	1111111111	
5	qqqqqqqqqq	
6	qqq111	
7	“Пустая строка”	<p>Введите данные Дерево из консоли построено. Всего символов прочитано: 0 Высота дерева = 0</p>

Вывод.

В процессе выполнения лабораторной работы были продуманы, созданы и реализованы на практике алгоритмы и методы работы со случайными БДП с рандомизацией. Также была реализована визуализация структур данных, алгоритмов, действий. Был получен опыт в подробной, понятной, с пояснениями демонстрации хода алгоритма.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: source.cpp

```
#include <fstream>
#include "binSTree.h"
#include <ctime>
#include <cstdlib>

int main()
{
    srand(time(NULL));
    setlocale(LC_ALL, "Russian");
    std::cout << "Введите f если хотите загрузить из файла,
    иначе любой другой символ для работы с консолью (Только
    латинские символы)" << std::endl;
    char symbol;
    char elem;
    node<char>* bt = nullptr;
    std::cin >> symbol;
    if (symbol == 'f') {
        std::ifstream file;
        std::string path;
        std::cout << "Введите название файла, откуда будет
        считываться дерево в формате <name.txt>" << std::endl;
        std::cin >> path;
        file.open(path);
        if (file.is_open()) {
            std::cout << "Файл успешно открыт, начинаю
            построение" << std::endl;
            std::cout
                << "Случайные БДП с рандомизацией имеют
            такую особенность, что добавление очередного элемента может
            производиться"
                << std::endl;
            std::cout
                << "как в корень элемента, используя
            вращения, так и в свое законное место в обычном случайном
            БДП."
                << std::endl;
        } else {
```

```

        std::cout << "Открыть файл не удалось, завершение
программы" << std::endl;
        return 0;
    }

    int step = 1;
    while (file >> elem) {
        std::cout << std::endl << "Шаг " << step << ":" <<
std::endl;
        std::cout << "Получен элемент " << elem <<
std::endl;
        bt = insert(bt, elem);
        std::cout << "*****" << std::endl;
        std::cout << "Дерево после " << step << " шага:"
<< std::endl;
        std::cout << "► ";
        step++;
        printBT(bt);
    }
    std::cout << "Дерево из файла построено." <<
std::endl;
    std::cout << "Всего символов прочитано: " << (step -
1) << std::endl;
    std::cout << "Высота дерева = " << heightBT(bt) <<
std::endl;
    } else{
        std::cout << "Введите количество элементов" <<
std::endl;
        int count;
        std::cin >> count;
        std::cout << "Введите данные" << std::endl;
        int step = 1;
        for (int i = 0; i < count; i++) {
            std::cin >> elem;
            std::cout << std::endl << "Шаг " << step << ":" <<
std::endl;
            std::cout << "Получен элемент " << elem <<
std::endl;
            bt = insert(bt, elem);
            std::cout << "*****" << std::endl;
            std::cout << "Дерево после " << step << " шага:"
<< std::endl;
            std::cout << "► ";

```



```

        step++;
        printBT(bt);
    }
    std::cout << "Дерево из консоли построено." <<
std::endl;
    std::cout << "Всего символов прочитано: " << (step -
1) << std::endl;
    std::cout << "Высота дерева = " << heightBT(bt) <<
std::endl;
}

    std::cout << "Какой элемент найти?" << std::endl;
    std::cin >> elem;
    int count = find(bt, elem);
    if (count == 0) std::cout << "Такого элемента нет в
дереве" << std::endl;
    else std::cout << "В дереве находится " << count << "
экз. введенного элемента" << std::endl;
    std::cout << "Произвожу вставку элемента " << elem << " в
дерево" << std::endl;
    bt = insert(bt, elem);
    std::cout << "Дерево после добавления:" << std::endl;
    std::cout << "► ";
    printBT(bt);

    char isWork = '\\0';
    while (isWork != 'n'){
        std::cout << "Хотите ли продолжить работу с деревом?
у - да, n - завершить программу" << std::endl;
        std::cin >> isWork;
        if (isWork == 'y'){
            std::cout << "Какой элемент добавить в дерево?"
<< std::endl;
            std::cin >> elem;
            bt = insert(bt, elem);
            std::cout << "Дерево после добавления:" <<
std::endl;
            std::cout << "► ";
            printBT(bt);
        }
    }
    return 0;
}

```

Название файла: binSTree.h

```
#include <iostream>
#include <string>

template <typename type>
struct node // структура для представления узлов дерева
{
    type key;
    int size;
    int count;
    node* left;
    node* right;
    node(int k) { key = k; left = right = 0; size = 1; count
= 1;}
};

template <typename type>
int find(node<type>* p, type k) // поиск ключа k в дереве p
{
    if( !p ) return 0; // в пустом дереве можно не искать
    if( k == p->key )
        return 1;
    if( k < p->key )
        return find(p->left,k);
    else
        return find(p->right,k);
}

template <typename type>
int getsize(node<type>* p) // обертка для поля size, работает
с пустыми деревьями (t=NULL)
{
    if( !p ) return 0;
    return p->size + p->count - 1;
}

template <typename type>
```

```

void fixsize(node<type>* p) // установление корректного
размера дерева
{
    if(p->left)
        fixsize(p->left);
    if(p->right)
        fixsize(p->right);
    p->size = getsize(p->left)+getsize(p->right)+1;
}

template <typename type>
node<type>* rotateright(node<type>* p) // правый поворот
вокруг узла p
{
    using namespace std;
    cout << "Поддерево с вершиной '" << p->key << "' до
поворота:" << endl;
    std::cout << "► ";
    printBT(p);
    cout << "*****rotateR*****" << endl;
    cout << "Корнем становится корень левого поддерева." <<
endl;
    cout << "Предыдущий корень перемещается на место корня
правого поддерева главного корня." << endl;
    cout << "А смещенный на предыдущем этапе корень
перемещается на место левого корня предыдущего корня." <<
endl;
    cout << "*****rotateR*****" << endl;
    node<type>* q = p->left;
    if( !q ) return p;
    p->left = q->right;
    q->right = p;
    q->size = p->size;
    fixsize(p);
    cout << "После:" << endl;
    std::cout << "► ";
    printBT(q);
    cout << endl;
    return q;
}

template <typename type>
node<type>* rotateleft(node<type>* q) // левый поворот вокруг
узла q

```

```

{
    using namespace std;
    cout << "Поддерево с вершиной '" << q->key << "' до
поворота:" << endl;
    std::cout << "► ";
    printBT(q);
    cout << "*****rotateL*****" << endl;
    cout << "Корнем становится корень правого поддерева." <<
endl;
    cout << "Предыдущий корень перемещается на место корня
левого поддерева главного корня." << endl;
    cout << "А смещенный на предыдущем этапе корень
перемещается на место правого корня предыдущего корня." <<
endl;
    cout << "*****rotateL*****" << endl;
    node<type>* p = q->right;
    if( !p ) return q;
    q->right = p->left;
    p->left = q;
    p->size = q->size;
    fixsize(q);
    cout << "После:" << endl;
    std::cout << "► ";
    printBT(p);
    return p;
}

```

```

template <typename type>
node<type>* insertroot(node<type>* p, type k) // вставка
нового узла с ключом k в корень дерева p
{
    if( !p ) return new node<type>(k);
    if( k < p->key )
    {
        p->left = insertroot(p->left,k);
        return rotateright(p);
    }
    if ( k > p->key )
    {
        p->right = insertroot(p->right,k);
        return rotateleft(p);
    }
    if (k == p->key)
    {

```

```

        p->count++;
    }
    return p;
}

template <typename type>
node<type>* insert(node<type>* p, type k) //
рандомизированная вставка нового узла с ключом k в дерево p
{
    using namespace std;
    if( !p ) {
        cout << "Вставка согласно БДП!" << endl;
        return new node<type>(k);
    }
    if( rand()%(p->size+1)==0 ) {
        cout << endl << "Выпал шанс рандомизации, вставляю в
корень, на место '" << p->key << "'" << endl;
        cout << "Для этого вставляю элемент в БДП, а затем
буду перемещать его при помощи вращений к корню." << endl;
        return insertroot(p, k);
    }
    else {
        if (p->key > k) {
            cout << "'" << k << "' < '" << p->key << "'" <<
endl;

            cout << "Переход \x1b[34m<-\x1b[0m" << endl;
            p->left = insert(p->left, k);
        } else {
            if (p->key == k) {
                cout << "Элемент в дереве уже был, увеличиваю
счетчик для этого элемента." << endl;
                p->count++;
            }
            else {
                cout << "'" << k << "' > '" << p->key << "'"
<< endl;

                cout << "Переход \x1b[31m->\x1b[0m" << endl;
                p->right = insert(p->right, k);
            }
        }
    }
    fixsize(p);
    return p;
}

```

```

template <typename type>
int heightBT(node<type>* bt) // Функция для подсчета высоты
BT
{
    if (!bt) return 0;
    else{
        return (heightBT(bt->left) > heightBT(bt->right) ?
heightBT(bt->left) : heightBT(bt->right)) + 1;
    }
}

template <typename type>
void printBT(node<type>* bt, std::string str="") // Функция
для печати BT
{
    if (bt == nullptr) return;
    std::string _str = str;
    std::cout << bt->key << std::endl;
    //std::cout << bt->key << " (" << bt->size << "/" << bt-
>count << ")" << std::endl;
    if (bt->right != nullptr)
    {
        std::cout << str;
    }
    if (bt->left == nullptr && bt->right != nullptr)
    {
        std::cout << "\x1b[31mL▶ \x1b[0m";
    }
    if (bt->left != nullptr && bt->right != nullptr)
    {
        std::cout << "\x1b[31m|▶ \x1b[0m";
    }
    if (bt->left != nullptr)
    {
        printBT(bt->right, str.append("\x1b[34m| \x1b[0m"));
    }
    else
    {
        printBT(bt->right, str.append(" "));
    }
    if (bt->left != nullptr)
    {
        std::cout << _str;
    }
}

```

```

    }
    if (bt->left != nullptr)
    {
        std::cout << "\x1b[34mL▶ \x1b[m";
    }
    printBT(bt->left, _str.append(" "));
}

```