**Министерство образования и науки Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н.Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ ИУ

КАФЕДРА ИУ7

# ОТЧЕТ ПО ЛР6 ТИПОВ И СТРУКТУР ДАННЫХ, Вариант 3

Студент Андреев Александр Алексеевич
фамилия, имя, отчество

Группа ИУ7-34Б

Тип практики учебная

Студент _____ _____
подпись, дата фамилия, и.о.

Преподаватель _____ _____
подпись, дата фамилия, и.о.

Оценка _____

13 декабря 2020

# Оглавление

# 1. Цель работы

Построить дерево, вывести его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; построить хеш-таблицу и вывести ее на экран, устранить коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировать таблицу; сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска (ДДП) и в хеш-таблицах. Сравнить эффективность реструктуризации таблицы для устранения коллизий с эффективностью поиска в исходной таблице.

# 2. Описание условия задачи

В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного целого числа в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

# 3. Описание ТЗ, включающее внешнюю спецификацию

## a. Описание исходных данных

Исходные данные программа получает из файла, в котором расположены целые числа, записанные через пробел.

Исходные команды программа получает из командной строки, где к каждому вводимому параметру заданы условия ввода и ограничения.

В случае некорректного ввода программа не завершит программу аварийно, а выдаст информацию о соответствующей ошибке.

## b. Описание задачи, реализуемой программой

Программа должна уметь совершать следующие действия:

- Строить ДДП в формате .png
- Строить АВЛ в формате .png
- Строить хеш-таблицу в формате .png
- Находить числа или говорить об их отсутствии в ДДТ

- Находить числа или говорить об их отсутствии в АВЛ
- Находить числа или говорить об их отсутствии в хеш-таблице
- Находить числа или говорить об их отсутствии в файле
- Выводить информацию об эффективности

## c. Способ обращения программы

Скомпилированная программа запускается командой **“./a.out”** на Unix и **“./a.exe”** на Windows.

При запуске программа должна вывести описание и основное меню:

Рис. 1

```
Программа сравнения работы с деревом обычным, сбалансированным и
хэш-таблицами.
1 -> Построение обычного дерево
 2 -> Построение сбалансированного дерево
 3 -> Поиск числа в ДДП
 4 -> Поиск числа в сбалансированном дереве
 5 -> Построение хеш-таблицы
 6 -> Печать хеш-таблицы
 7 -> Поиск числа в хеш-таблице
 8 -> Поиск числа в файле
 9 -> Сравнение эффективности
 0 -> Выход из программы
```

Далее после ввода соответствующей команды программа должна выполнять команды пользователя.

1 - Построение обычного дерево (см. Рис. 2)
2 - Построение сбалансированного дерево (см. Рис. 3)
3 - Поиск числа в ДДП (см. Рис. 4)
4 - Поиск числа в сбалансированном дереве (см. Рис. 5)
5 - Построение хеш-таблицы
6 - Печать хеш-таблицы (см. Рис. 6)
7 - Поиск числа в хеш-таблице
8 - Поиск числа в файле
9 - Выводится сравнительная информация об эффективности программы (Рис. 7)
0 - Программа завершает свою работу

Рис. 2 (Печать обычного дерева)

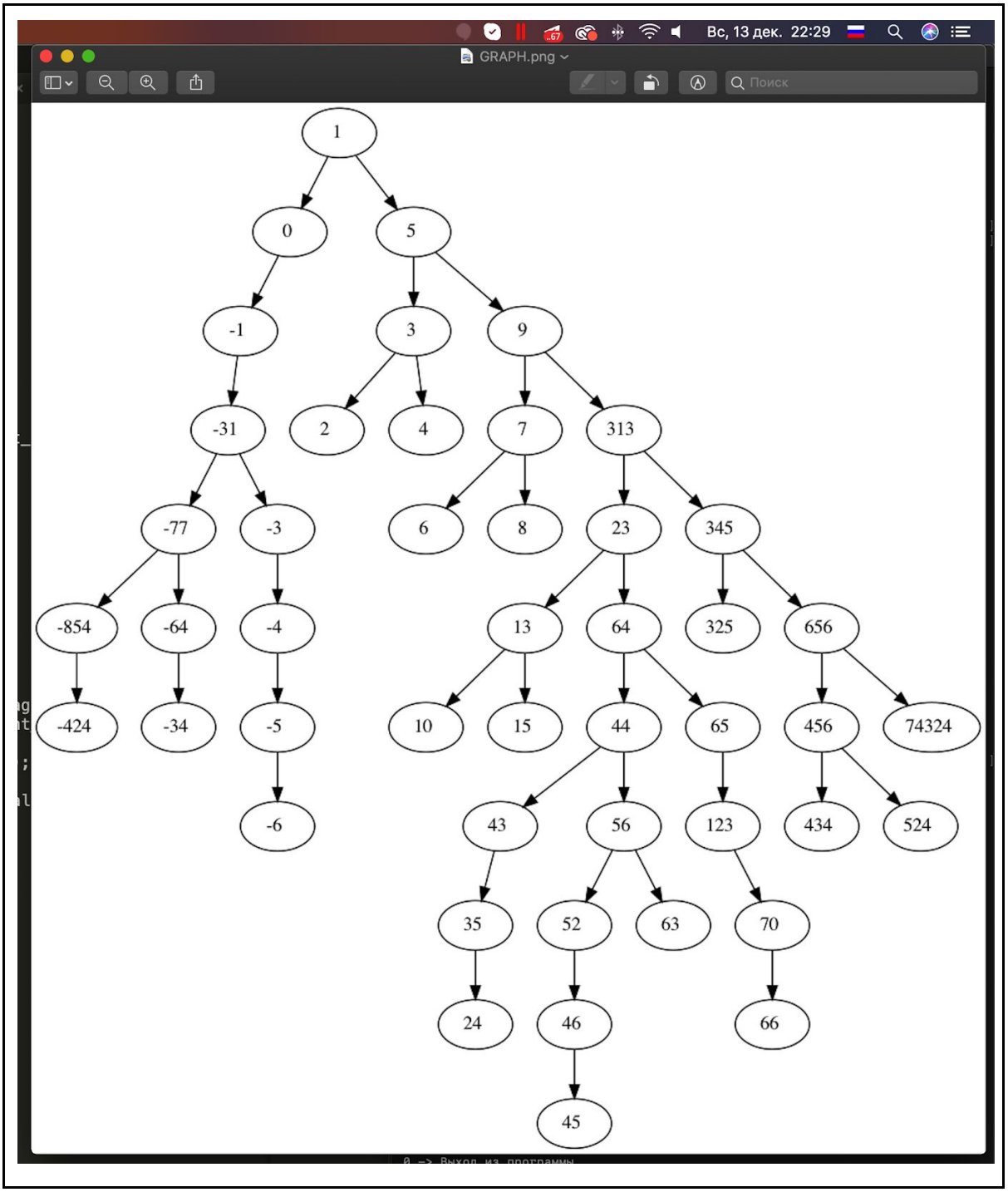

Вс, 13 дек. 22:29
GRAPH.png
Поиск
1
0
5
-1
3
9
-31
2
4
7
313
-77
-3
6
8
23
345
-854
-64
-4
13
64
325
656
-424
-34
-5
10
15
44
65
456
74324
-6
43
56
123
434
524
35
52
63
70
24
46
66
45

Рис. 3 (Печать сбалансированного дерева)

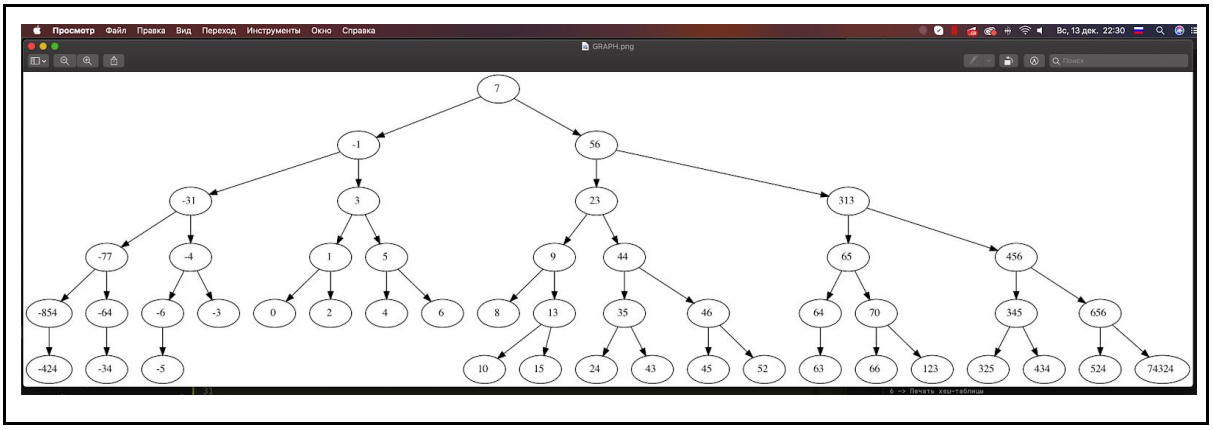


Рис. 4 (Поиск числа в обычном дереве)

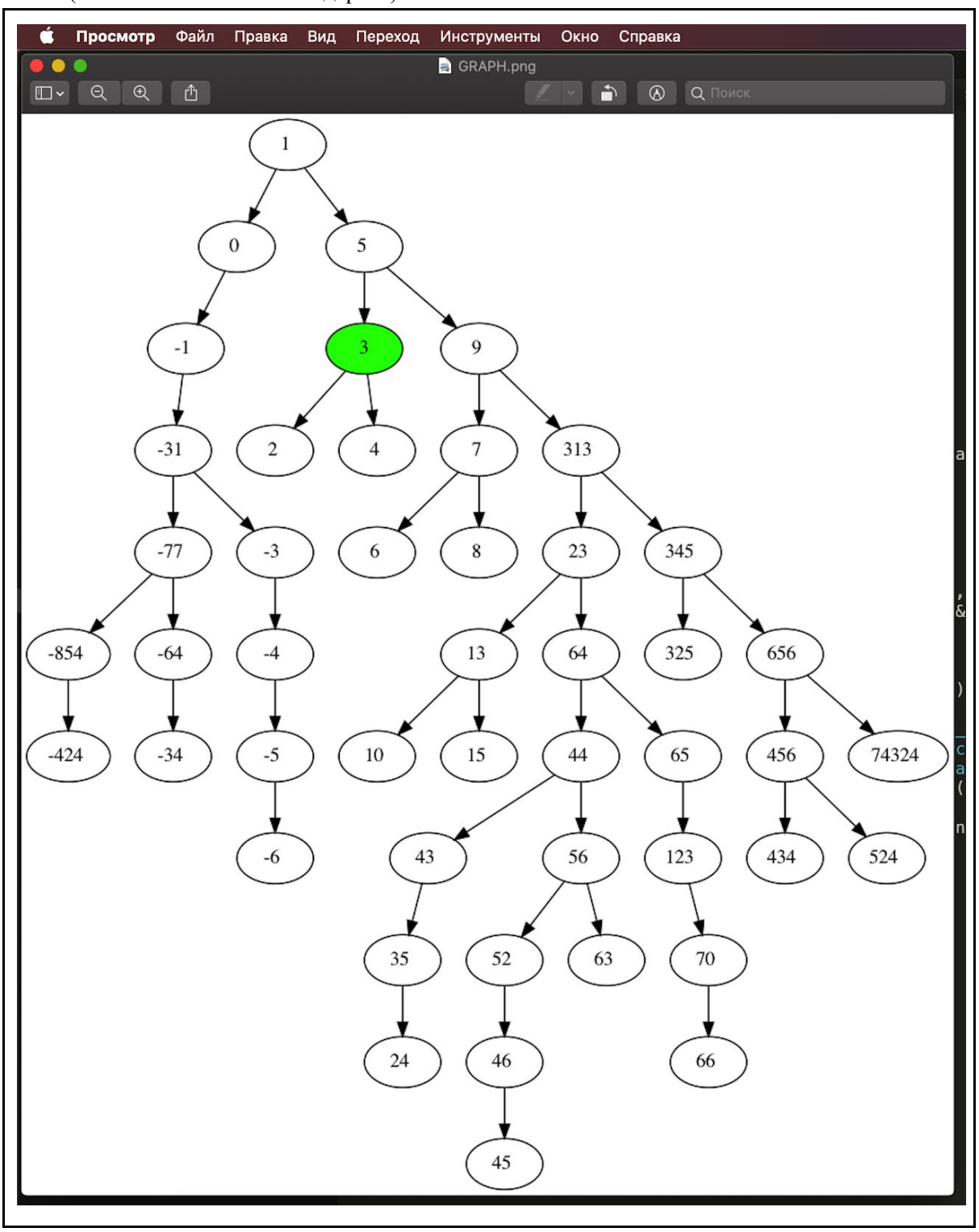


Рис. 5 (Поиск числа в сбалансированном дереве)

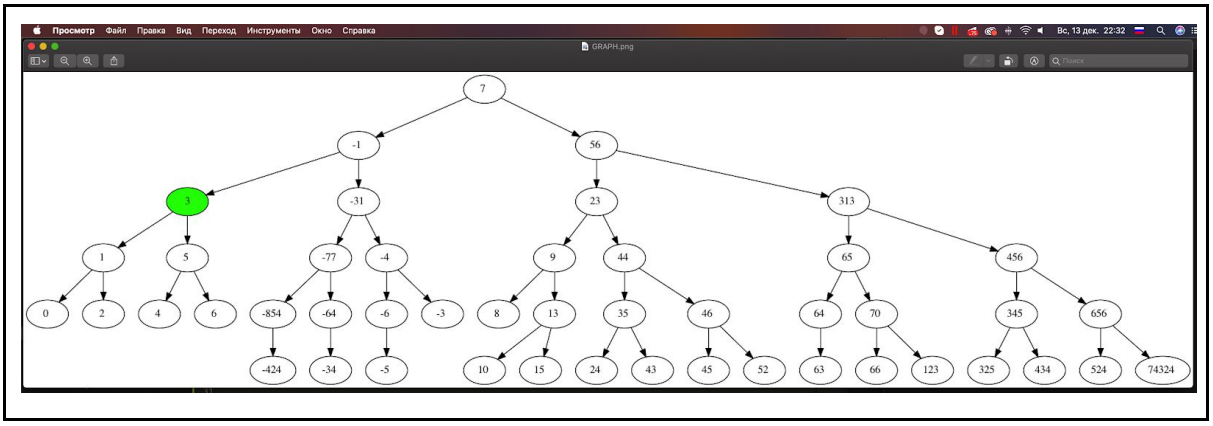



Рис. 6 (Печать хэш-таблицы)

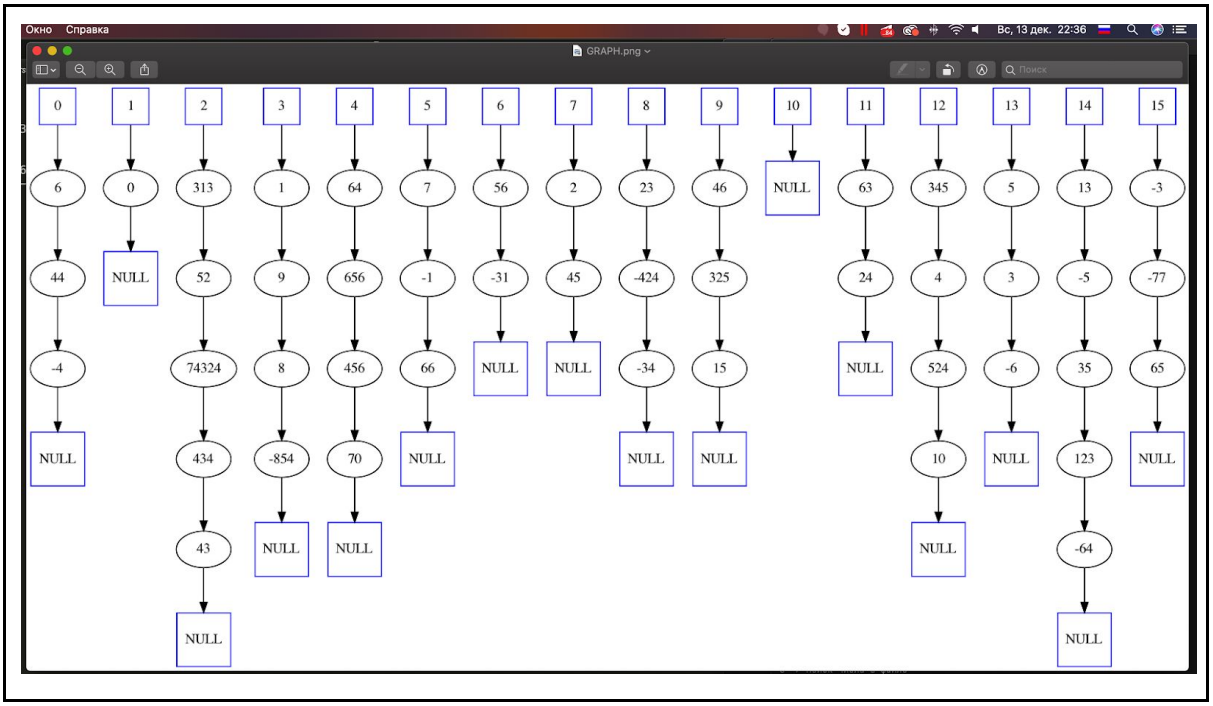
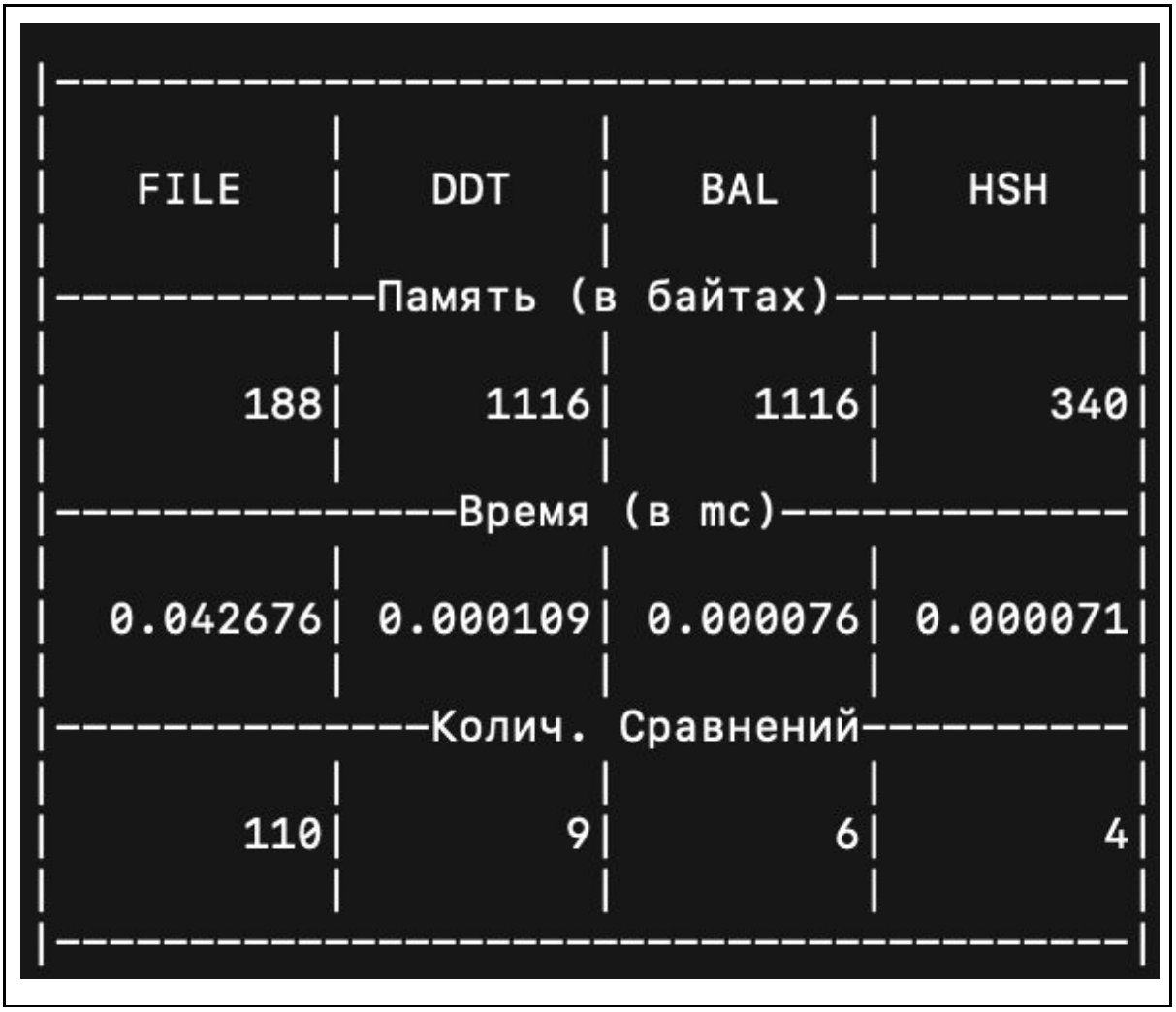


Рис. 7 (Оценка эффективности программы)



| FILE | DDT | BAL | HSH |
|---|---|---|---|
| Память (в байтах) | | | |
| 188 | 1116 | 1116 | 340 |
| Время (в mc) | | | |
| 0.042676 | 0.000109 | 0.000076 | 0.000071 |
| Колич. Сравнений | | | |
| 110 | 9 | 6 | 4 |

При наличии аварийных ситуаций и ошибок пользователя программа должна вывести соответствующее сообщение и не должна завершить свою работу абортно.

## d. Описание возможных аварийных ситуаций и ошибок пользователя

В представленной ниже **Таблице 1** отражены действия программы при различных возможных, допущенных пользователем, при использовании основного меню программы ошибках.

Табл.1

| Действие программы при различных ошибках | | |
|---|---|---|
| № | Действие пользователя | Реакция программы |
| 1 | Введен несуществующий номер или строка пункта меню | Программа выведет сообщение об ошибке “Ошибка ввода пункта меню. Повторите попытку.” |

| | | |
|---|---|---|
| 2 | Пользователь выводит на экран хеш-таблицы без ее создания | Программа выведет сообщение об ошибке “Ошибка ввода пункта меню. Повторите попытку.” |
| 3 | Поиск числа в хеш-таблицы без создания | Программа выведет сообщение об ошибке “Ошибка ввода пункта меню. Повторите попытку.” |

# 4. Описание внутренних СД

Реализация работы с деревом представлена следующим образом (См. Рис. 8);

Рис. 8

```
typedef struct tnode
{
   int value;
   int balance;
   struct tnode *left;
   struct tnode *right;
}t_node;
```

Реализация работы с хеш-таблицей представлена следующим образом (См. Рис. 9);

Рис. 9

```
typedef struct node_table
{
   int value;
   struct node_table *next;
}t_hash;
```

# 5. Алгоритм

При запуске программа выводит описание, основное меню и приглашение для ввода (Рис. 6):

Рис. 10

```
Программа сравнения работы с деревом обычным, сбалансированным и
хэш-таблицами.
1 -> Построение обычного дерево
 2 -> Построение сбалансированного дерево
 3 -> Поиск числа в ДДП
 4 -> Поиск числа в сбалансированном дереве
 5 -> Построение хеш-таблицы
```

6 -> Печать хеш-таблицы
7 -> Поиск числа в хеш-таблице
8 -> Поиск числа в файле
9 -> Сравнение эффективности
0 -> Выход из программы

После чего пользователь выбирает пункт меню. В случае некорректного выбора пункта - пользователь получает сообщение об ошибке.

Хеш-функция для конкретного числа представляет из себя деление остатком на ближайшее простое число слева на оси координат к длине таблицы.

## Базовые функции программы и их описание

В обеих реализациях учет относительных единиц времени идет изменением значения temp_time за каждый проход и проверкой состояния на каждом его изменении.

### ***Меню***

**void menu(t_node *balanced_root, FILE *f, t_node *root);**
Основное меню программы.

### ***Поиск элемента***

**t_node * search_in_tree(t_node *tree, int val, int *amount, int *flag);**
Поиск элемента в стандартном дереве.

**t_hash *looking_in_list(t_hash *head, int searh_int, int *amount);**
Поиск элемента в сбалансированном дереве.

**int search_in_hash(t_hash **table, int searh_int, int table_len, int *amount, int k);**
Поиск элемента в хем-таблице дереве.

**int search_in_file(FILE *f, int val, int *amount_file);**
Поиск элемента в файле.

### ***Замер эффективности***

**void efficiency(void);**
Компаратор вызова замеров эффективности.

**unsigned long long get_simple_ddt_search_time(t_node **root, int *amount_ord, int *flag);**
Замер времени поиска элемента в обычном дереве.

**unsigned long long get_balanced_ddt_search_time(t_node **balanced_root, int *amount_bal, int *flag);**
Замер времени поиска элемента в сбалансированном дереве.

**unsigned long long get_hash_table_search_time(t_hash ***hash, int *amount_hash, int *k, int *table_len);**
Замер времени поиска элемента в хеш-таблице.

**unsigned long long get_file_search_time(FILE *operation_file, int *amount_file);**
Замер времени поиска элемента в файле.

### ***Вывод графика и работа с DOT***

**void export_to_dot(FILE *f, const char *tree_name, t_node *tree, int is_search, int search_val);**
Вывод в файл и преобразование в картинку.

**void apply_pre(t_node *tree, FILE *f, int is_search, int search_val);**
Обход дерева.

**void to_dot(t_node *tree, FILE *f, int is_search, int search_val);**
Вывод в специальном формате для ДОТ.

### ***Операционные функции***

**t_node* balance(t_node* p);**
Балансировка дерева.

**t_hash* create_hash_node(int val);**
Создание элемента связного списка.

**t_node *ord_add_node(t_node *tree, t_node *node);**
Добавление узла.

# 6. Набор тестов с указанием, что проверяется

В представленных ниже **Таблица 2** отражены тестирование устойчивости работы консольного меню программы и демонстрация устойчивости работы программы к различному типу выполняемых с ней операций пользователем соответственно.

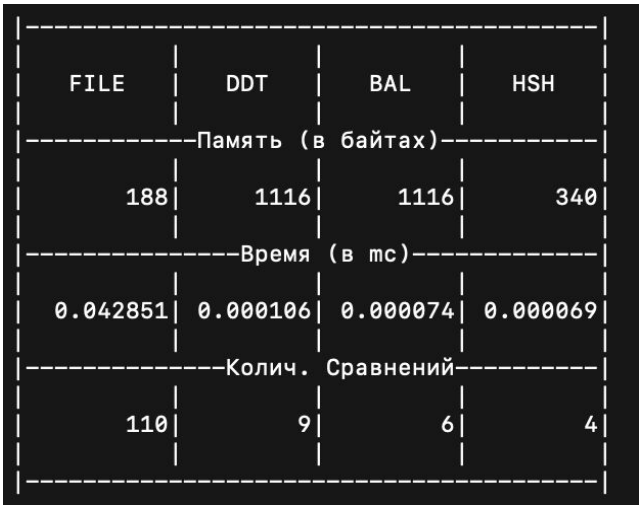
Табл. 2

| Тестирование устойчивости работы программы | | |
|---|---|---|
| Ввод пользователя | Реакция программы | Что проверяется данной операцией? |

| | | |
|---|---|---|
| Введен несуществующий номер или строка пункта меню | Программа выведет сообщение об ошибке “Ошибка ввода пункта меню. Повторите попытку.” | Устойчивость программы к вводу неверной команды меню |
| Пользователь выводит на экран хеш-таблицы без ее создания | Программа выведет сообщение об ошибке “ Для операции поиска сначала необходимо построить хэш-таблицу....” | Проверка устойчивости программы к вызову операций над несозданной хеш-таблицей. |
| Поиск числа в хеш-таблицы без создания | | |
| Введено нецелое число | Программа выведет сообщение об ошибке “Ошибка ввода....” | Устойчивость программы к получению нецелых чисел |

# 7. Выводы по проделанной работе

По результатам проделанной работы я научился строить дерево, выводить его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнивать эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; строить хеш-таблицу и выводить ее на экран, устранять коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировать таблицу; сравнивать эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска (ДДП) и в хеш-таблицах, сравнивать эффективность реструктуризации таблицы для устранения коллизий с эффективностью поиска в исходной таблице.




Для того, чтобы увеличить точность, я произвел поиск числа, который отсутствует в файле 1000 раз.

В результате данных прогона могу сказать, что хеш-таблица наиболее эффективна по двум показателем - время и количество сравнений, но менее эффективна по памяти в сравнении с поиском в файле. Также надо отметить, что для уменьшения времени поиска в хеш-таблице количество коллизий должно быть наименьшим, они зависят от того, сколько я выделил памяти под нее и от равномерности заполнения хеш-таблицы хеш-функцией. Как раз для большей эффективности по времени нужно выделять больше памяти.

Сравнение по Памяти:
хеш таблицы эффективнее обычного и сбалансированного дерева на 66%
файл эффективнее на хеш таблицы на 44%

Сравнение по Времени:
хеш таблицы эффективнее обычного дерева на 8%
хеш таблицы эффективнее сбалансированного дерева на 27%
хеш таблицы эффективнее файла на 99%

# 8. Контрольные вопросы

**1. Что такое дерево?**

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

**2. Как выделяется память под представление деревьев?**

Выделение памяти под деревья определяется типом их представления. Это может быть таблица связей с предками или связный список сыновей. Оба представления можно реализовать в виде матрицы или списка. При реализации списком память выделяется динамически, при реализации матрицей статически.

**3. Какие стандартные операции возможны над деревьями?**

Основные операции с деревьями: обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

**4. Что такое дерево двоичного поиска?**

Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

**5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?**

Дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу называется идеально сбалансированным. Двоичное дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу называется АВЛ-деревом.

**6. Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?**

Поиск в АВЛ дереве имеет сложность O(log2n), в то время как в обычном ДДП сложность O(n).

**7. Что такое хеш-таблица, каков принцип ее построения?**

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей.

Хеш-функция – функция, которая ставит в соответствие каждому ключу индекс ячейки, где расположен элемент с этим ключом.

**8. Что такое коллизии? Каковы методы их устранения.**

Коллизия – ситуация, когда разным ключам соответствует одно значение хеш-функции. Существует несколько возможных вариантов разрешения коллизий: внешнее (открытое) хеширование (метод цепочек) и внутреннее (закрытое) хеширование (открытая адресация).

**9. В каком случае поиск в хеш-таблицах становится неэффективен?**

Поиск в хэш-таблице становится неэффективным при большом числе коллизий – сложность поиска возрастает.

При открытом хэшировании в случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется связный список. Таким образом, если для нескольких различных значений ключа возвращается одинаковое значение хеш- функции, то по этому адресу находится указатель на связанный список, который содержит все значения. Поиск в этом списке осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.

При закрытом хэшировании в этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку, до тех пор, пока не будет найден ключ K или пустая позиция в таблице.

**10. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах.**

В хэш-таблице минимальное время поиска O(1). В АВЛ дереве O(log2n). В дереве двоичного поиска O(h), где h – высота дерева.