Министерство науки и высшего образования 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 Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6 «ОБРАБОТКА ДЕРЕВЬЕВ»

Студент Гаврилюк Владислав Анатольевич

Группа ИУ7 – 31Б

Преподаватель Барышникова Марина Юрьевна
Проверил

2023 г

Описание условия задачи

В файловой системе каталог файлов организован в виде бинарного дерева. Каждый узел обозначает файл, содержащий имя и атрибуты файла, в том числе и дату последнего обращения к файлу. Написать программу, которая обходит дерево и удаляет из него все файлы, последнее обращение к которым происходило до определенной даты. Вывести исходные и измененные деревья в виде дерева. Сравнить время удаления в дереве, построенном по алфавиту, со временем перестроения дерева по дате обращения и удаления в нем.

Описание технического задания

Входные данные:

1. Программа принимает на вход данные о файле в формате:

-rwxrwxrwx ДД.ММ.ГГГГ ЧЧ:ММ имя_файла

где:

1. -rwxrwxrwx - права доступа к файлу. Начинается всегда с '-', т.к. можно добавлять только файлы. Далее первые 3 буквы - права пользователя, следующие 3 - права группы, и последние 3 буквы - права для остальных. 'r' - право на чтение, 'w' - право на запись, 'x' - право на выполнение. Если указанного права нет, ставится '-'. Все символы пишутся слитно.

2. ДД.ММ.ГГГГ ЧЧ:ММ - дата последнего обращения к файлу.

3. имя_файла - имя файла не должно превышать 50 знаков. Может содержать латинские буквы, арабские цифры, точку, нижнее подчеркивание и тире.

Как можно заметить, формат очень близок к реальному представлению данных в ОС Linux.

В аргументах командной строки можно указать входной и выходной файлы, откуда будут считаны исходные данные и куда, соответственно, будут записаны результирующие. (Указание файлов не является обязательным).

Выходные данные:

После выполнения каждой операции программа создает .png файлы с графическим представлением результата работы программы (Состояние дерева файлов до и после выполнения). Также в терминал можно вывести текущее состояние директории в виде отсортированного по имени списка файлов. Статистические данные выводятся в терминал.

Задачи программы:

1. Добавление файла в директорию (добавление узла в дерево).

2. Удаление файла из директории по имени (удаление узла из дерева).

3. Удаление всех файлов из директории, обращение к которым происходило до указанной даты, используя бинарное дерево поиска, построенное по алфавиту.

4. Удаление всех файлов из директории, обращение к которым происходило до указанной даты, используя бинарное дерево поиска, построенное по датам последнего обращения.

5. Сравнение эффективности двух реализаций.

6. Вывод текущего состояния директории в виде отсортированного по имени списка файлов.

Обращение к программе:

Команда запуска: **./app.exe [in.txt out.txt]**

in.txt - файл с исходными данными. В первой строке содержится количество файлов, далее данные о каждом файле записываются на новой строке в указанном ранее формате.

out.txt - файл, куда будет записан результат работы программы.

В меню пользователь указывает номер нужной операции (от 1 до 6), а затем вводит необходимые данные для выбранной операции. Для завершения программы необходимо ввести 0.

Аварийные ситуации:

- Указан некорректный номер операции
- Данные введены не в указанном формате
- Невозможность удаления файла, т.к. директория пуста
- Невозможность выделения памяти для указанной последовательности

Ошибочные ситуации сопровождаются поясняющим сообщением об ошибке.

Описание структур данных

В программе представлен следующий набор структур:

1. **node_t** - узел бинарного дерева.

```
typedef struct node_t node_t;
struct node_t
{
    char name[MAX_FILE_NAME_LEN + 1];
    attributes_t *attr;
    date_t *last_access;
    node_t *left;
    node_t *right;
};
```

---

char name[MAX_FILE_NAME_LEN + 1] - строка, содержащая имя файла (MAX_FILE_NAME_LEN = 50)

attributes_t *attr - указатель на структуру, содержащую атрибуты файла (права доступа)

date_t *last_access - указатель на структуру, содержащую дату последнего обращения к файлу.

node_t *left - указатель на левый дочерний узел.

node_t *right - указатель на правый дочерний узел.

2. **attributes_t** - структура, содержащая атрибуты файла (права доступа).

```
typedef struct attributes_t

{

     char user[NUM_OF_ACCESS_TYPES]; // права доступа
пользователя [rwx]

     char group[NUM_OF_ACCESS_TYPES]; // права доступа
группы [rwx]

     char others[NUM_OF_ACCESS_TYPES]; // права доступа для
остальных [rwx]

} attributes_t;
```

---

char user[NUM_OF_ACCESS_TYPES] - массив из 3 символов, соответствующих правам доступа пользователя.

char group[NUM_OF_ACCESS_TYPES] - массив из 3 символов, соответствующих правам доступа членам группы.

char others[NUM_OF_ACCESS_TYPES] - массив 3 символов, соответствующих правам доступа для остальных.

3. **date_t** - структура, содержащая дату последнего обращения.

```
typedef struct date_t

{

    size_t hours;

    size_t minutes;

    size_t day;

    size_t month;

    size_t year;

} date_t;
```

---

size_t hours - часы.

size_t minutes - минуты.

size_t day - день.

size_t month - месяц.

size_t year - год.

Для реалистичности год может варьироваться от 2000 до 2023.

Описание алгоритмов

1. **Pre-order traversal**

Выполняется действие над корнем, затем алгоритм рекурсивно повторяется для левого поддерева, затем для правого.

2. **In-order traversal**

Сначала рекурсивно обрабатывается левое поддерево, затем выполняется действие над корнем, потом рекурсивно обрабатывается правое поддерево.

3. **Добавление в бинарное дерево поиска**

Текущий узел сравнивается с новым. Если по сравниваемому параметру новый узел больше (или равен), то далее новый узел будет сравниваться с дочерним правым, в обратном случае - с дочерним левым. Цикл будет продолжаться, пока мы не достигнем узла, в котором необходимый узел будет отсутствовать и на это место будет поставлен новый.

4. **Удаление в бинарном дереве поиска**

Удаляемый узел меньше текущего, то рекурсивно запускаем удаление для левого поддерева, в обратном случае - для правого. Если текущий узел и есть удаляемый, то есть 3 варианта удаления:

   a. Если узел не имеет дочерних узлов. Просто удаляем узел из дерева и освобождаем память.
   b. Если узел имеет один дочерний узел. Заменим текущий узел на дочерний и освобождаем память из-под текущего.

c. Если узел имеет 2 дочерних узла. Заменим текущий узел на минимальный узел из правого поддерева, чтобы не нарушить условие бинарного дерева поиска.

5. Удаление всех узлов, обращение к которым было до указанной даты в бинарном дереве поиска, построенном по алфавиту.

Рекурсивно (post-order traversal) удаляем в каждом поддереве подходящие узлы с учетом условий алгоритма удаления в бинарном дереве поиска.

6. Удаление всех узлов, обращение к которым было до указанной даты в бинарном дереве поиска, построенном по дате последнего обращения.

Если последнее обращение текущего узла было до указанной даты, то мы удаляем всё левое поддерево и освобождаем из-под него память. Возвращаем результат рекурсивной обработки правого поддерева. Если последнее обращение текущего узла было после указанной даты (или в указанную дату), то рекурсивно обрабатываем левое поддерево. Результат записываем в указатель на дочернее левое поддерево и возвращаем указатель на текущий элемент.

Сравнение эффективности

Сравнение эффективности удаления всех узлов, обращение к которым было до указанной даты в бинарном дереве поиска, и перестроения в бинарное дерево поиска по дате и последующего удаления из него. Всего проведено 2 эксперимента:

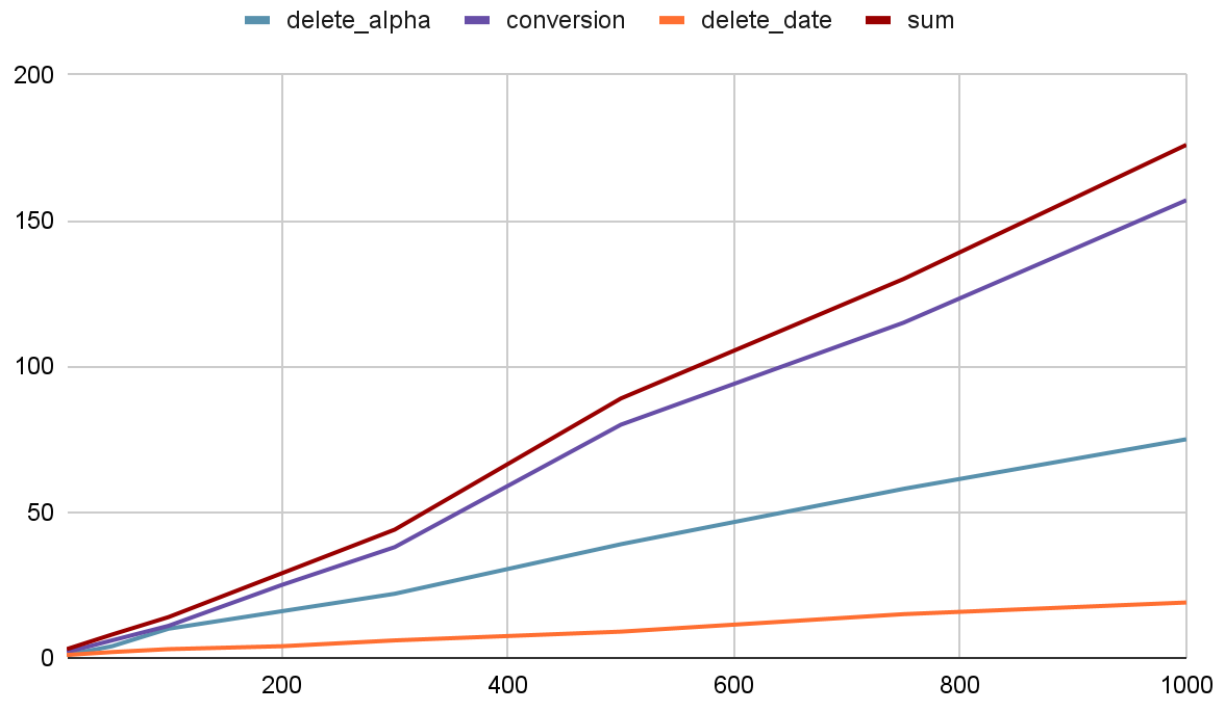
1. Сравнение эффективности 5 в зависимости от количества узлов. (Т.к. деревья заполнялись случайными данными, то высота дерева будет прямо пропорциональна количеству узлов). Удаление производилось по средней для диапазона [2000-2023] дате 15.06.2011 12:00.
2. Сравнение эффективности в зависимости от степени ветвления.

Эксперимент 1 (данные в мкс)

Количество узлов	**Удаление из дерева по алфавиту**	**Конвертация в дерево по дате**	**Удаление из дерева, построенного по дате**	**Конвертация и удаление в сумме**
10	1	2	1	3

50	4	6	2	8
100	10	11	3	14
200	16	25	4	29
300	22	38	6	44
500	39	80	9	89
750	58	115	15	130
1000	75	157	19	176

Зависимость времени работы от количества узлов (случайное заполнение)



delete_apha - удаление из бинарного дерева поиска, построенного по алфавиту,

delete_date - удаление из бинарного дерева поиска, построенного по дате.

conversion - преобразование из бин.дерева по алфавиту в бин.дерево по дате.

sum = delete + conversion

Зависимость времени работы от степени ветвления (при 30 узлах в дереве)

Степень ветвления	**Удаление из дерева по алфавиту**	**Конвертация в дерево по дате**	**Удаление из дерева, построенного по дате**	**Конвертация и удаление в сумме**
Ветвление отсутствует. Дерево вырожденное	3	7	3	10
Средняя степень ветвления (случайное распределение)	2	3	2	5
Высокая степень ветвления. Сбалансированное дерево	2	2	2	4

Набор тестов

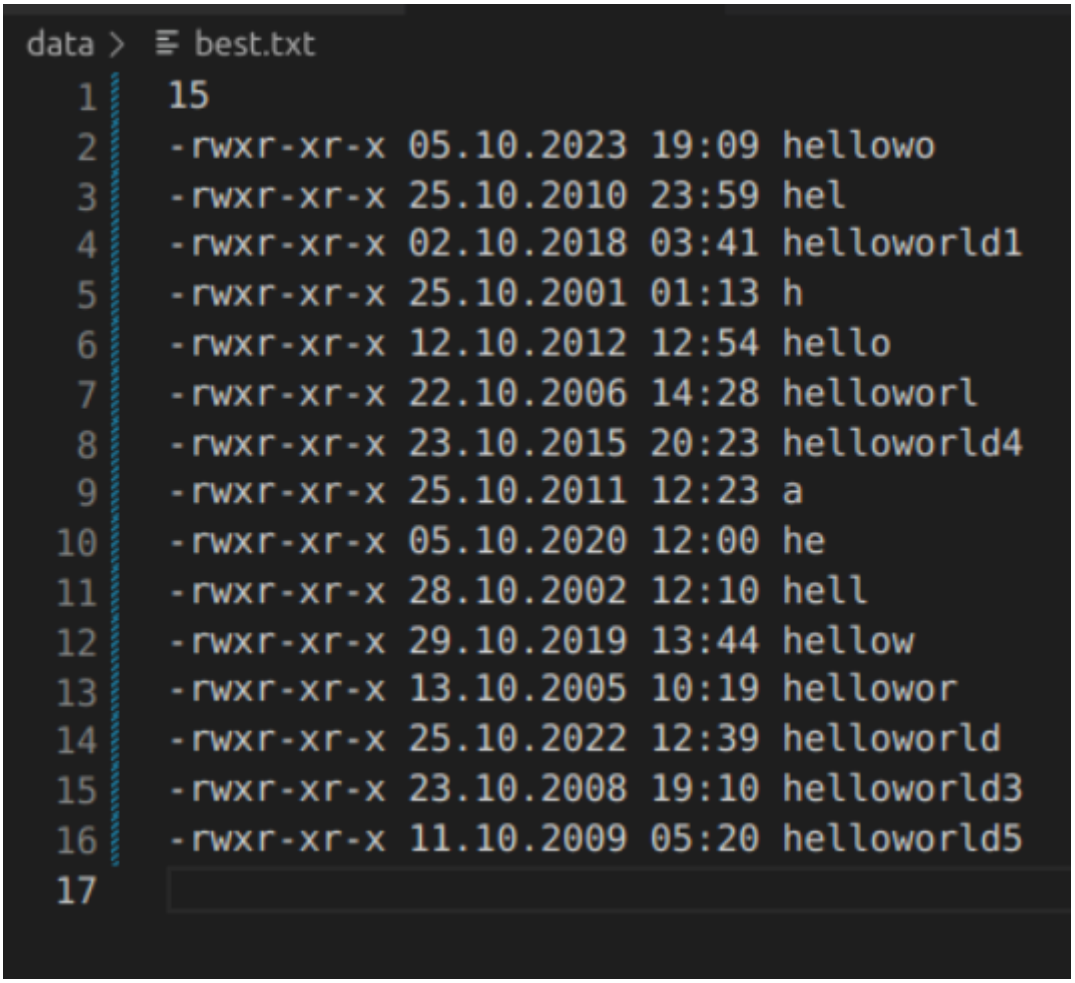
№ Теста	Ситуация	Результат
1	Указано неверное кол-во аргументов	[ERR]: Указаны некорректные аргументы.
2	Файл не существует	[ERR]: Ошибка при открытии файла.
3	Количество файлов превышает допустимое значение	[ERR]: В файле указано некорректное количество файлов.

4	Некорректные атрибуты файла (права доступа)	[ERR]: Права доступа файла указаны некорректно.
5	Некорректная дата последнего обращения	[ERR]: Дата последнего обращения указана некорректно.
6	Некорректное имя файла	[ERR]: В имени файла указаны недопустимые символы, или длина имени превышает 50 знаков.
7	Попытка добавления файла, имя которого уже существует в директории	[ERR]: Директория-дерево уже содержит файл с таким именем.
8	Попытка удаления файла из пустой директории	[ERR]: Невозможно выполнить операцию. Директория пуста.
9	Некорректные данные при вводе номера операции	[ERR]: Указан некорректный номер операции.
10	Попытка удаления несуществующего файла	[ERR]: Файл с такими параметрами не найден.
11	Добавление файла	[OK]: Операция завершена успешно.
12	Удаление файла	[OK]: Операция завершена успешно.
13	Удаление всех файлов из бин.дерева поиска по алфавиту, обращение к которым было до определенной даты	[OK]: Операция завершена успешно.
14	Удаление всех файлов из бин.дерева поиска по дате, обращение к которым было до определенной даты	[OK]: Операция завершена успешно.
15	Сравнение эффективности	[OK]: Операция завершена успешно.

16	Некорректные данные при вводе файла	[ERR]: Ошибка при попытке чтения данных.

Примеры работы программы

Программа считывает из файла 15 файлов:



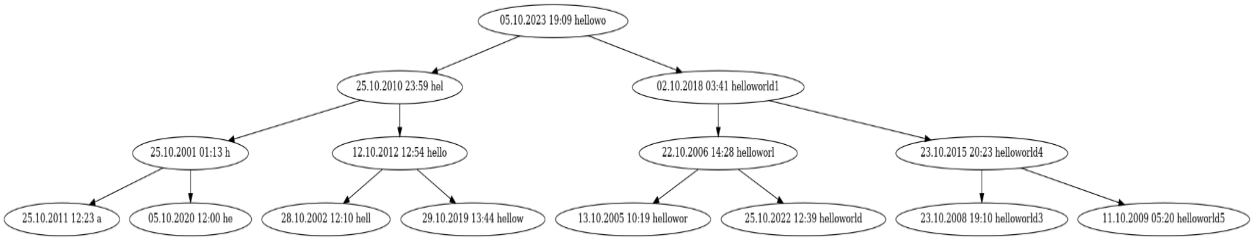
Данные будут организованы в виде бинарного дерева поиска, построенного по алфавиту. Следовательно, при **in-order обходе** данные будут выведены в алфавитном порядке (операция 6):

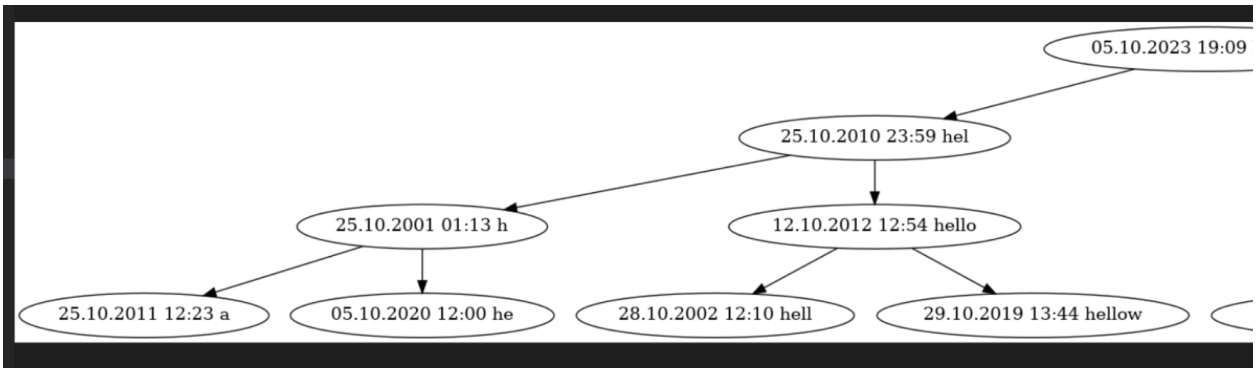
```
Введите номер операции: 6
-rwxr-xr-x 25.10.2011 12:23 a
-rwxr-xr-x 25.10.2001 01:13 h
-rwxr-xr-x 05.10.2020 12:00 he
-rwxr-xr-x 25.10.2010 23:59 hel
-rwxr-xr-x 28.10.2002 12:10 hell
-rwxr-xr-x 12.10.2012 12:54 hello
-rwxr-xr-x 29.10.2019 13:44 hellow
-rwxr-xr-x 05.10.2023 19:09 hellowo
-rwxr-xr-x 13.10.2005 10:19 hellowor
-rwxr-xr-x 22.10.2006 14:28 helloworl
-rwxr-xr-x 25.10.2022 12:39 helloworld
-rwxr-xr-x 02.10.2018 03:41 helloworld1
-rwxr-xr-x 23.10.2008 19:10 helloworld3
-rwxr-xr-x 23.10.2015 20:23 helloworld4
-rwxr-xr-x 11.10.2009 05:20 helloworld5

[OK]: Операция завершена успешно.
```

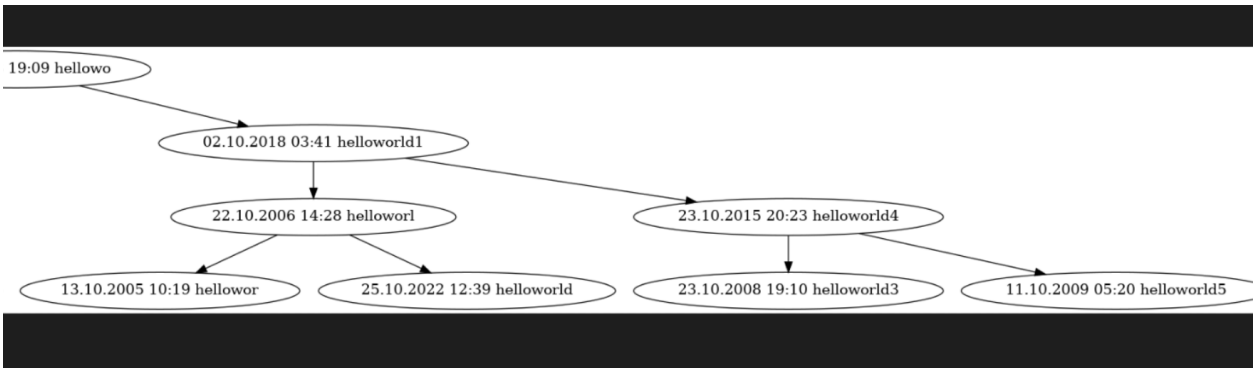
Добавим новый файл qwerty.txt с датой 15.06.2011 12:00 (операция 1):

```
Введите файл в формате '-rwxrwxrwx ДД.ММ.ГГГГ ЧЧ:ММ имя_файла': -rwxrwxrwx 15.10.2011 12:00 qwerty.txt
Rendering png file for alphabet_tree_after...
DONE.

[OK]: Операция завершена успешно.
```

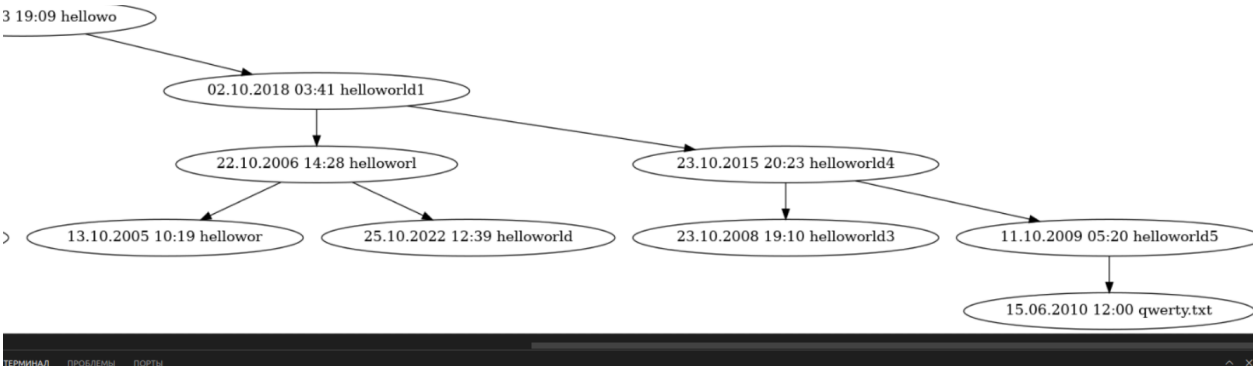
Дерево до добавления:

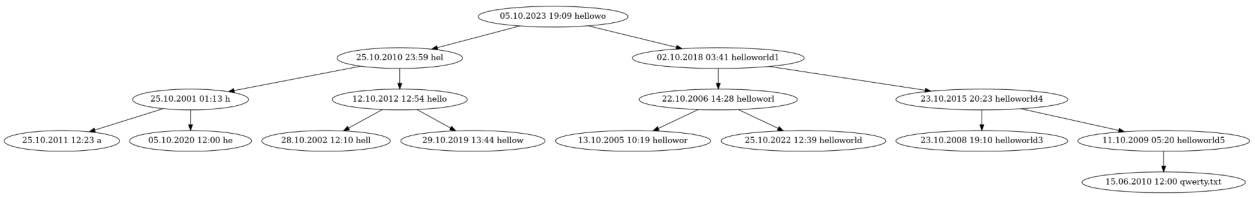






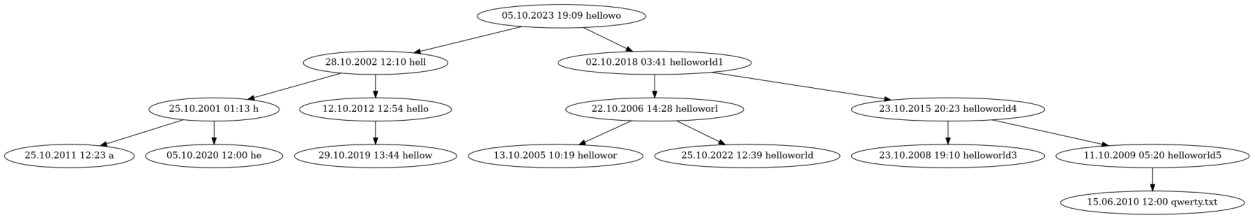
Дерево после добавления:

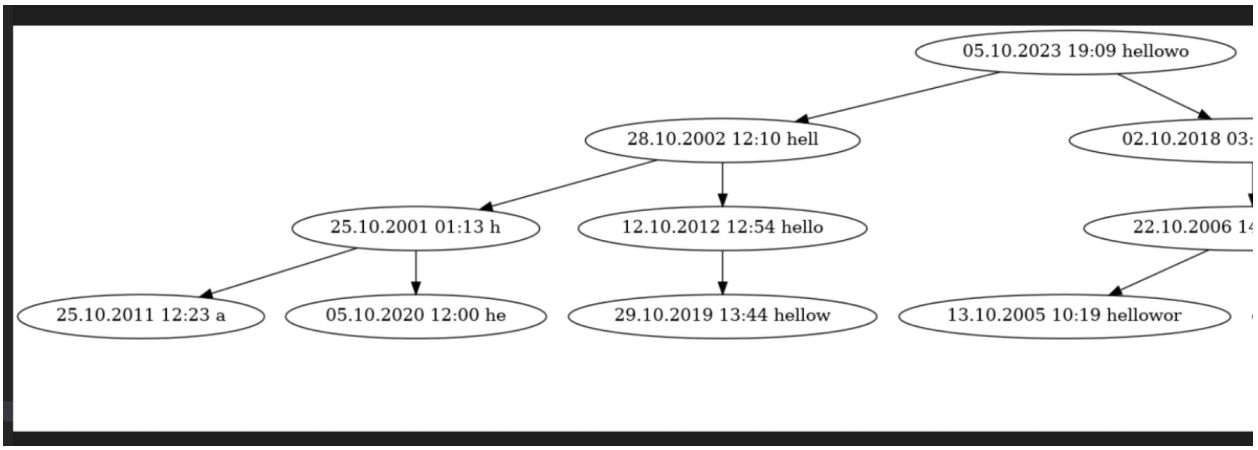




Теперь удалим файл с именем hel **(операция 2)**. Данный узел обладает двумя потомками, поэтому на его место встанет минимальный в правом поддереве узла hel (узел hell).

Результат:

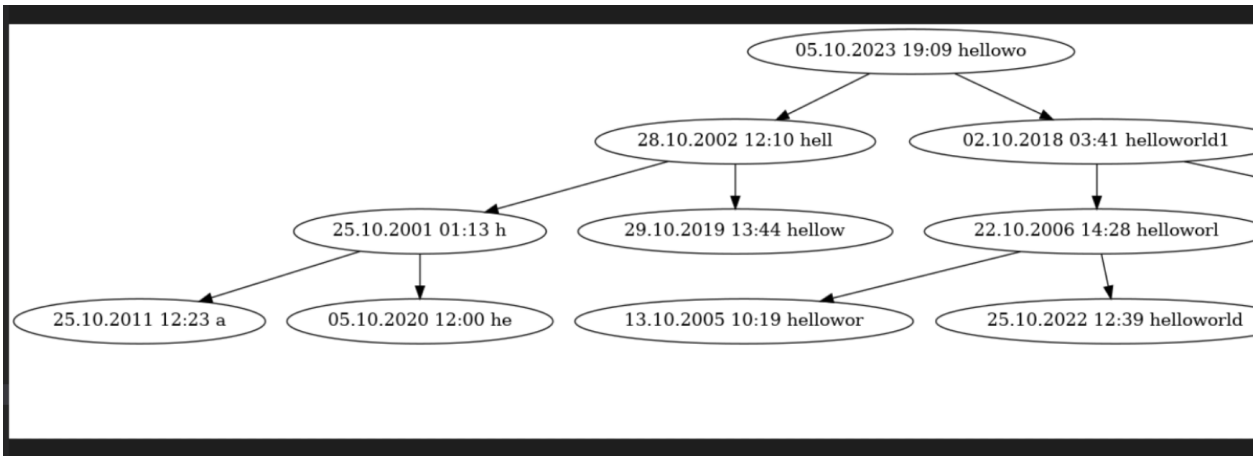




Выведем текущее состояние директории и убедимся, что файл qwerty.txt добавлен, а hel удален:

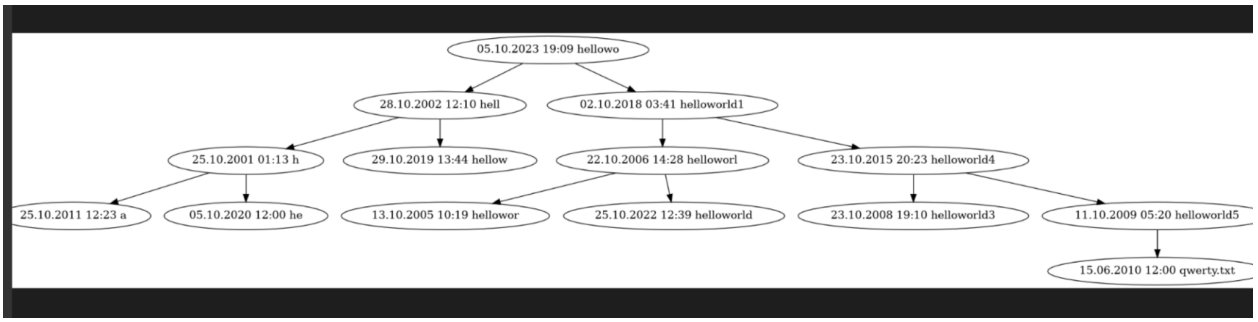
```
Введите номер операции: 6
-rwxr-xr-x 25.10.2011 12:23 a
-rwxr-xr-x 25.10.2001 01:13 h
-rwxr-xr-x 05.10.2020 12:00 he
-rwxr-xr-x 28.10.2002 12:10 hell
-rwxr-xr-x 12.10.2012 12:54 hello
-rwxr-xr-x 29.10.2019 13:44 hellow
-rwxr-xr-x 05.10.2023 19:09 hellowo
-rwxr-xr-x 13.10.2005 10:19 hellowor
-rwxr-xr-x 22.10.2006 14:28 helloworl
-rwxr-xr-x 25.10.2022 12:39 helloworld
-rwxr-xr-x 02.10.2018 03:41 helloworld1
-rwxr-xr-x 23.10.2008 19:10 helloworld3
-rwxr-xr-x 23.10.2015 20:23 helloworld4
-rwxr-xr-x 11.10.2009 05:20 helloworld5
-rwxrwxrwx 15.06.2010 12:00 qwerty.txt
```

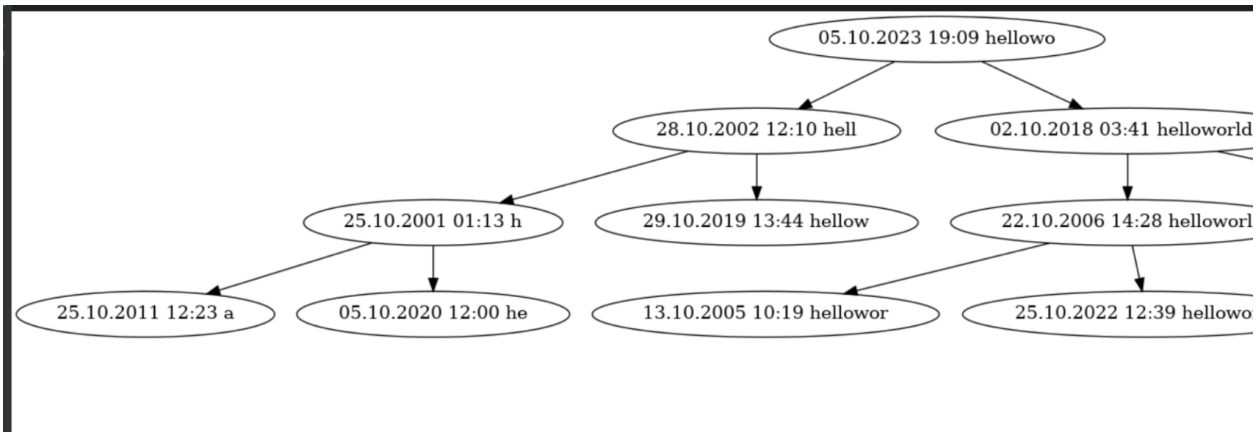
Удалим теперь файл **hello.** Он обладает всего одним дочерним узлом, поэтому, как и ожидалось, на его место встанет файл hellow.

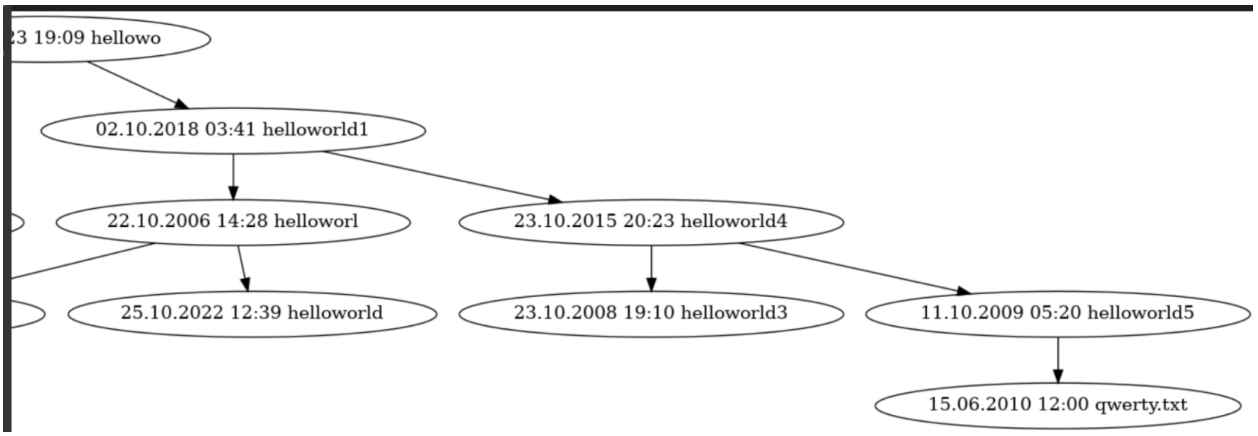


Теперь удалим все файлы из бинарного дерева поиска, построенного по алфавиту, последнее обращение к которым было произведено до 15.06.2011 12:00 (операция 3):

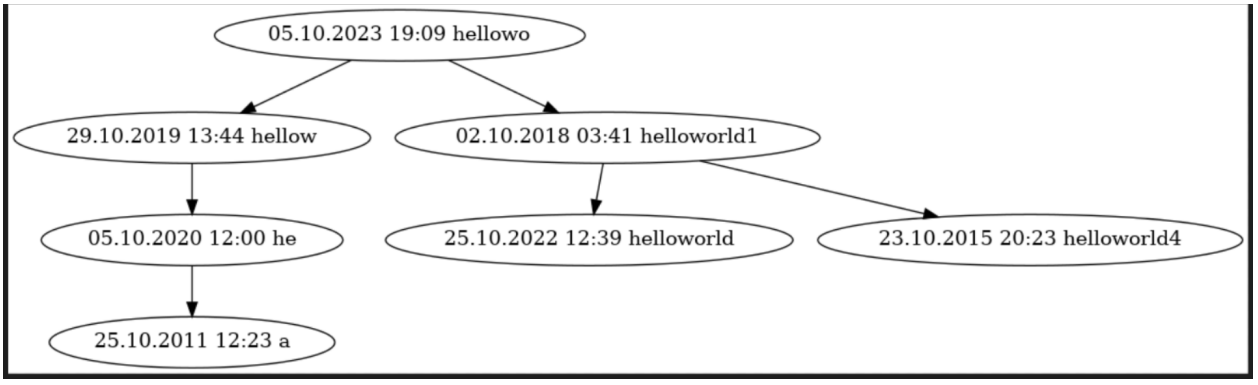
Дерево до удаления:







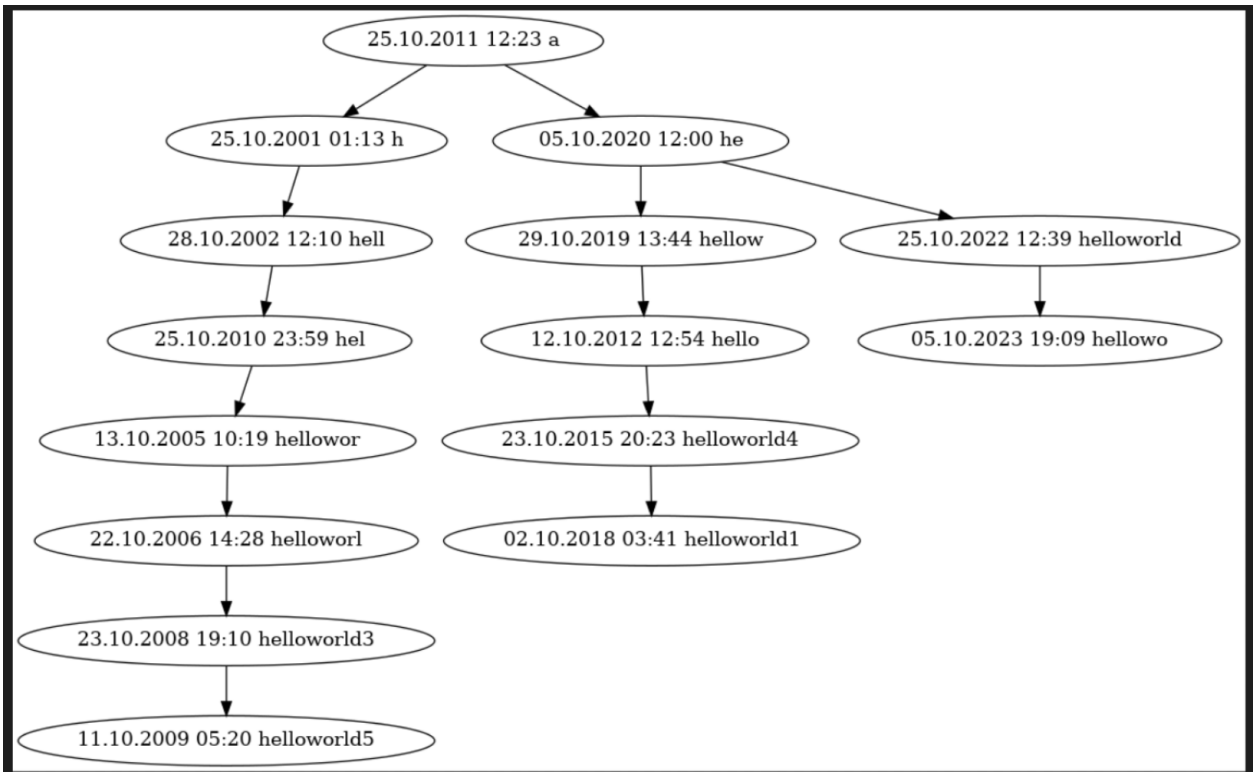
Дерево после удаления:



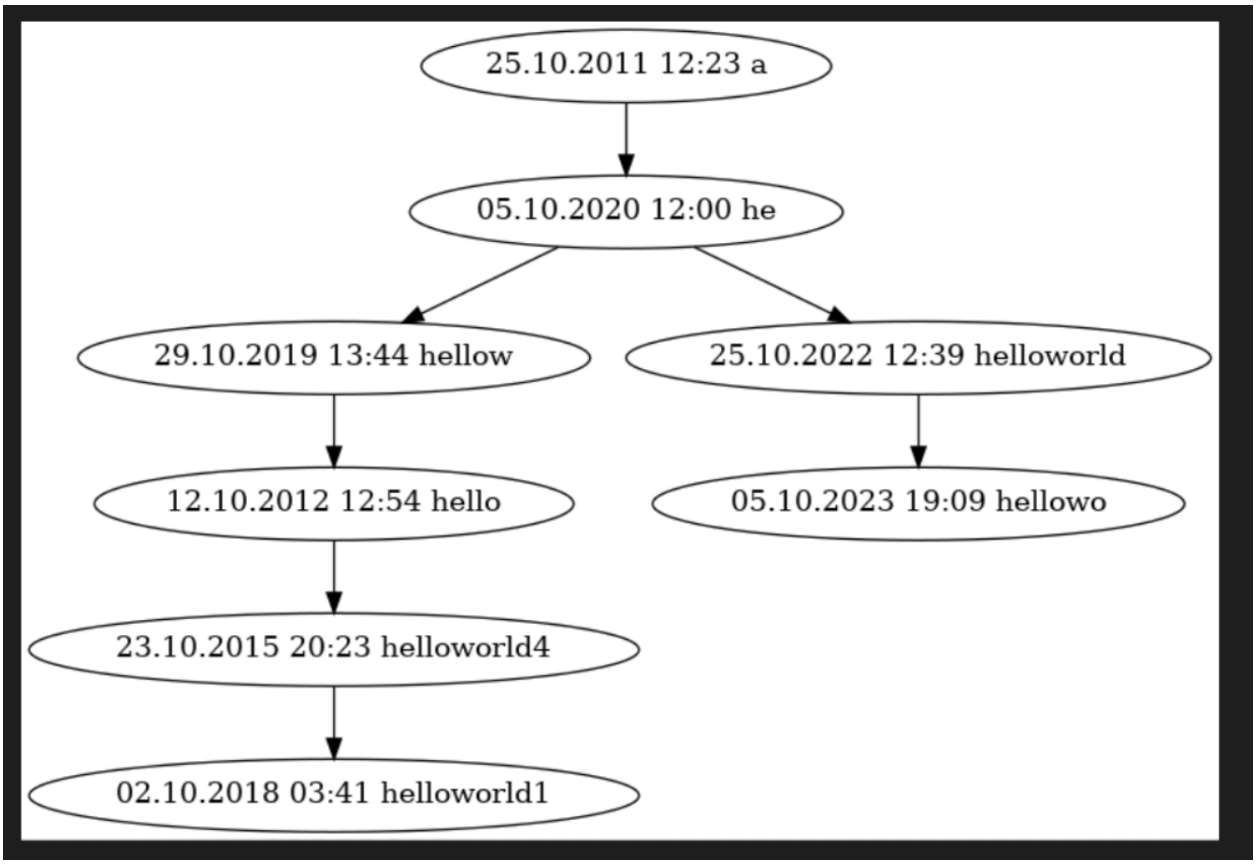
Как и предполагалось, остались только те файлы, к которым обращались позднее указанной даты (или в эту дату).

Считаем заново данные. Удалим все файлы из бинарного дерева поиска, построенного по дате, последнее обращение к которым было произведено до 15.06.2011 12:00 (операция 4):

Дерево до удаления:

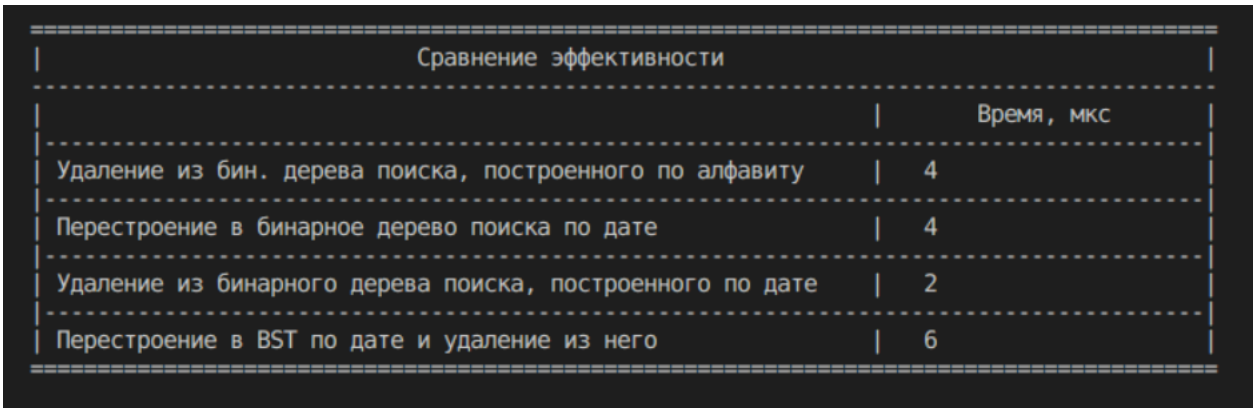


Дерево после удаления:



(В результате на 1 узел больше, чем в предыдущей операции, потому что там мы удалили узел hello, а тут нет).

Сравнение эффективности двух алгоритмов удаления (операция 5):



Сравнение эффективности	
	Время, мкс
Удаление из бин. дерева поиска, построенного по алфавиту	4
Перестроение в бинарное дерево поиска по дате	4
Удаление из бинарного дерева поиска, построенного по дате	2
Перестроение в BST по дате и удаление из него	6

Ответы на контрольные вопросы

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

2. Какие бывают типы деревьев?
Бинарное дерево, бинарное дерево поиска, красно-черное дерево, N-арное дерево. Также делят деревья на сбалансированные и несбалансированные.

3. Какие стандартные операции возможны над деревьями?
Обход дерева, вставка, удаление и поиск элемента.

4. Что такое дерево двоичного поиска?
Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше. (Если равенство допустимо, то равные предку потомки помещаются в в левое или правое поддерево на выбор программиста).

Вывод

Результатом данной лабораторной работы является готовая программа, реализующая работу с файлами в директории с помощью бинарных деревьев поиска.

Проанализировав зависимости времени работы алгоритмов от количества узлов, можно сделать вывод, что удаление из бинарного дерева поиска, построенного по дате последнего обращения, намного эффективнее удаления из дерева, построенного по алфавиту. Но из-за конвертации из одного дерева в другое второй способ удаления (конвертация в бин.дерево поиска по дате + удаление) оказался менее эффективным. Также стоит учесть, что эффективность алгоритмов зависит от степени ветвления деревьев. Таким образом, при вырожденном дереве теряется преимущество использования бинарных деревьев поиска и скорость алгоритмов сильно снижается.