



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 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

ДЕРЕВЬЯ

Студент Жаринов М. А.

Вариант 4

Группа ИУ7-32Б

Название предприятия НУК ИУ МГТУ им. Н. Э. Баумана

Студент _____ Жаринов М. А.

Преподаватель _____ Силантьева А. В.

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.

Вывести дерево на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов.

Техническое задание

Исходные данные:

Каталог файлов: Текстовый файл, содержащий некоторое количество структур, каждое поле на новой строке, количество структур в первой строке файла.

Дата обращения: Дата в виде трех целых чисел

Выходные данные:

Бинарные деревья поиска по алфавиту и по дате в виде дерева, в текстовом виде (обходы), деревья с удаленными файлами, обращение к которым было до определенной даты, результаты сравнения способов удаления .

Реализуемые задачи:

Пункт меню 0 – Выход из программы

Пункт меню 1 – Чтение дерева из файла

Пункт меню 2 – Перестроение дерева по другому полю

Пункт меню 3 – Вывод дерева в виде дерева

Пункт меню 4 – Печать дерева в префиксном обходе

Пункт меню 5 – Печать дерева в инфиксном обходе

Пункт меню 6 – Печать дерева в постфиксном обходе

Пункт меню 7 – Добавление элемента в дерево

Пункт меню 8 – Удаление элемента из дерева

Пункт меню 9 – Поиск элемента в дереве

Пункт меню 10 – Удаление элементов с обращением до определенной даты

Пункт меню 11 – Проведение сравнительного анализа удаления при построении дерева по различным ключам

Способ обращения к программе:

Строка запуска программы ./app.exe

Аварийные ситуации:

1. Ошибка ввода при чтении полей структуры. Код ошибки 3
2. Недостаток свободного места в оперативной памяти. Код ошибки 2

Все остальные ошибочные ситуации (некорректный пункт меню, несуществующий файл, отсутствующий элемент в дереве, пустое дерево) не являются аварийными и вызывают возвращение в меню.

Внутренние структуры данных

Дата последнего обращения хранится в структуре `date_t`.

```
typedef struct
{
    short day;
    short month;
    int year;
} date_t;
```

Поле `day`– день

Поле `month`– месяц

Поле `year`– год

Элементы дерева хранятся в структуре `file_t`.

```
    char *name;
    date_t date;
    bool hidden;
    bool system;
} file_t;
```

Поле `name`– название файла

Поле `date`– дата обращения к файлу

Поле `hidden`– атрибут для скрытых файлов

Поле `system`– атрибут для системных файлов

Вершины дерева хранятся в структуре `tree_node_t`.

```
typedef struct tree_node
{
    file_t file;
    struct tree_node *left;
    struct tree_node *right;
} tree_node_t;
```

Поле `file` –значение элемента

Поле `left` –указатель на левого потомка

Поле `right` –указатель на правого потомка

Дерево хранится в структуре `tree_t`.

```
typedef struct
{
    tree_node_t *root;
    bool sorted_by_date;
} tree_t;
```

Поле `root` –указатель на корень дерева

Поле `sorted_by_date` –флаг способа сортировки дерева

Описание алгоритма

Программа запрашивает номер элемента в меню, считывает его и выполняет соответствующее действие:

Пункт меню 0 – Выход из программы

Пункт меню 1 – Чтение дерева из файла

1. Программа считывает имя файла и открывает указанный файл
2. Программа считывает количество структур в файле
3. Программа считывает заданное число структур по алгоритму
4. Считывание строки – имени файла
5. Считывание дня последнего обращения – целое число от 1 до 31
6. Считывание месяца последнего обращения – целое число от 1 до 12
7. Считывание года последнего обращения – целое число от 1 до 2100
8. Считывание флага скрытости – 2 для скрытого, 1 для обычного
9. Считывание флага системности – 2 для системного, 1 для обычного
10. Добавление элемента в дерево поиска по имени

Пункт меню 2 – Перестроение дерева по другому полю

1. Программа инициализирует новое пустое дерево поиска по противоположному полю к существующему
2. Программа обходит дерево префиксным обходом и добавляет все элементы в новое дерево.
3. При ошибке вставки (повторение ключа), программа очищает новое дерево и печатает ошибку

Пункт меню 7 – Добавление элемента в дерево

1. Программа запрашивает и считывает информацию о файле
2. Программа вставляет в дерево элемент следующим рекурсивным алгоритмом:
3. Если текущий узел пуст, то вставить в него
4. Если текущий узел равен по ключу, то вернуть ошибку
5. Если текущий узел меньше по ключу вставляемого, то вставить в правое поддерево, иначе в левое

Пункт меню 8 – Удаление элемента из дерева

1. Программа запрашивает и считывает имя файла
2. Программа ищет заданный элемент по алгоритму поиска (см. ниже)
3. Программа удаляет найденный элемент с помощью следующего

рекурсивного алгоритма:

4. Если элемент больше по ключу удаляемого, то удалить в левом поддереве, если меньше, то в правом
5. Если элемент равен по ключу, то
6. Если правый потомок пуст, то записать на место текущего узла левый потомок
7. Иначе если левый пуст, то записать правого потомка
8. Иначе найти самый левый элемент в правом поддереве, записать его данные в текущий элемент, и вызывать функцию удаления этого элемента в правом поддереве

Пункт меню 9 – Поиск элемента в дереве

1. Программа запрашивает и считывает имя файла
2. Если дерево отсортировано по имени, то программа ищет элемент с помощью следующего алгоритма:
3. Если текущий элемент больше по ключу, то искать в левом поддереве, если меньше, то в правом
4. Если элемент равен, то вернуть его
5. Если дерево отсортировано по дате, то поиск осуществляется с помощью проверки всех элементов при префиксном обходе

Пункт меню 10 – Удаление элементов с обращением до определенной даты

1. Программа запрашивает и считывает дату обращения, до которой надо удалить файлы.
2. Если дерево отсортировано по имени, то программа удаляет все нужные файлы при полном постфиксном обходе

3. Если дерево отсортировано по дате, то программа удаляет файлы в правом поддереве текущего элемента, только если его самого надо удалить (он меньше заданной даты)

Пункт меню 11 – Проведение сравнительного анализа удаления при построении дерева по различным ключам

1. Программа запускает цикл по трем тестовым случаям – удаление 0 элементов, удаление половины элементов, удаление всех элементов

2. Программа находит среднее время выполнения 1000 следующих операций:

3. Удаление в дереве поиска по имени

4. Перестроение дерева

5. Удаление в дереве поиска по дате

6. Деревья считываются из заранее подготовленного файла с 1000 структур

7. Программа печатает получившиеся результаты

Основные функции

Функция для обработки выбранного пункта меню. Принимает номер выбранного пункта меню и указатель на дерево. Возвращает код ошибки

```
int process_menu(menu_item_t menu_item, tree_t *tree);
```

Функция вставки элемента в дерево. Принимает указатель на дерево, указатель на узел для вставки. Возвращает код ошибки

```
int universal_tree_insert(tree_t *tree, tree_node_t *node);
```

Функция удаления элемента из дерева. Принимает указатель на дерево, имя файла к удалению. Возвращает код ошибки

```
int universal_tree_delete(tree_t *tree, const char *name);
```

Функция поиска элемента в дереве. Принимает дерево, имя файла к поиску, изменяет по указателю указатель на найденный узел

```
int universal_tree_find(tree_t tree, const char *name, tree_node_t **found);
```


Функция освобождения дерева. Принимает указатель на указатель на корень, флаг необходимости очистки имен файлов

```
void bin_tree_free(tree_node_t **tree, bool all);
```

Функция экспорта дерева в файл на языке DOT. Принимает указатель на файловую переменную, имя дерева, указатель на корень дерева

```
void bin_tree_export_to_dot(FILE *f, const char *tree_name, tree_node_t *root);
```

Функция удаления из дерева всех элементов с датой обращения меньше заданной. Принимает указатель на дерево, указатель файл с записанной в нем датой.

```
void universal_del_old_files(tree_t *tree, file_t *sample);
```

Функция печати дерева в префиксном обходе. Принимает указатель на корень дерева

```
void bin_tree_print_pre(tree_node_t *tree);
```

Функция печати дерева в инфиксном обходе. Принимает указатель на корень дерева

```
void bin_tree_print_in(tree_node_t *tree);
```

Функция печати дерева в постфиксном обходе. Принимает указатель на корень дерева

```
void bin_tree_print_post(tree_node_t *tree);
```

Функция изменения способа сортировки дерева. Принимает указатель на дерево и флаг необходимости печати сообщений. Возвращает код ошибки

```
int tree_change_sort(tree_t *tree, bool verbose);
```

Функция анализа разных способов удаления.

```
void run_analysis(void);
```

Набор тестов

Обработка пунктов меню

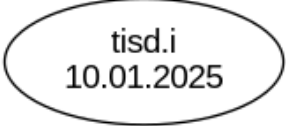
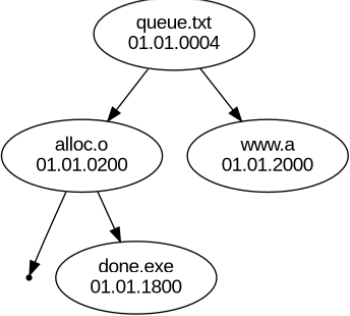
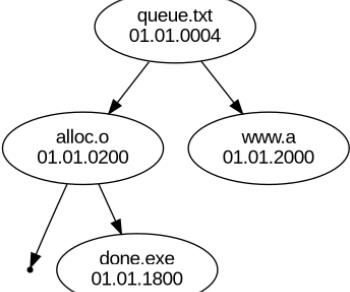
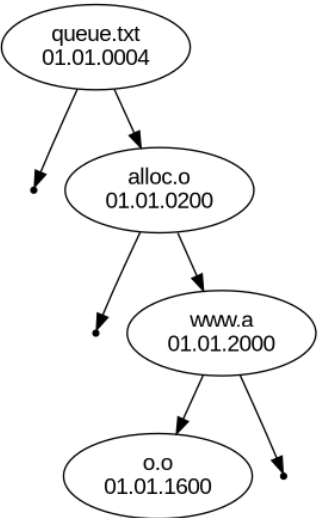
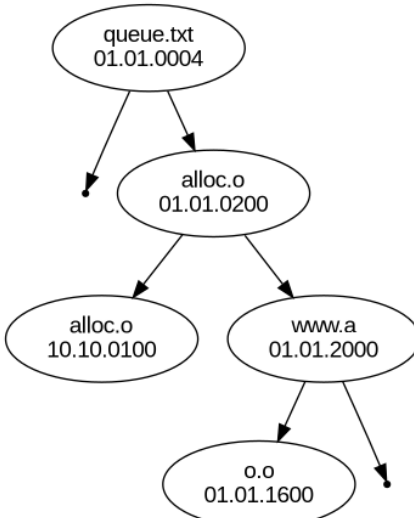
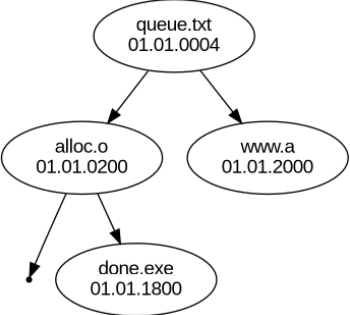
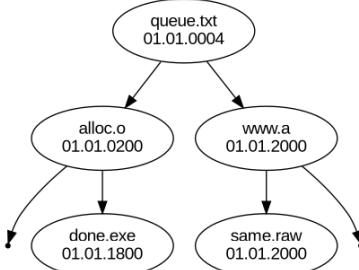
Описание теста	Вход	Результат
Отрицательное число	-1	Для выбора пункта меню введите целое число от 0 до 11
Число больше 11	12	Для выбора пункта меню введите целое число от 0 до 11
Пустой ввод		Для выбора пункта меню введите целое число от 0 до 11
Максимальный пункт меню	11	[сведения о скорости удалений]
Минимальный пункт меню	0	[выход из программы]
Символ вместо числа	a	Для выбора пункта меню введите целое число от 0 до 11
Выбор пунктов меню, которые требуют непустое дерево, при пустом дереве	3	Пустое дерево!

Считывание элемента

Описание теста	Ввод	Вывод
Ввод корректного элемента	aaaaaa.tisd 23 11 2018 1 2	Элемент добавлен в дерево
Ввод некорректной даты	aaaaaa.tisd 23 13 2018	Ошибка чтения даты обращения файла!

Ввод вещественного числа вместо даты	aaaaaa.tisd 23 11.1 2018 1 2	Ошибка чтения даты обращения файла!
Ввод постороннего символа в дате	aaaaaa.tisd 23 4 2018t 1 2	Ошибка чтения даты обращения файла!
Ввод неправильного флага	aaaaaa.tisd 23 4 2018t 1 0	Ошибка чтения атрибута файла!
Пустая строка вместо имени	23 11 2010 ...	Ошибка чтения имени файла!
Ввод вещественного числа вместо флага	www 23 4 2018 1 1.0	Ошибка чтения атрибута файла!

Добавление элемента в дерево

Описание теста	Название и дата файла	Входное дерево	Выходное дерево
Добавление в пустое дерево	tisd.i 10.01.2025	Пустое дерево	
Добавление существующего элемента	www.a 2.4.2010		<p>Элемент уже есть в дереве!</p> 
Добавление в дерево поиска по дате элемента с существующим названием, но другой датой	alloc.o 10.10.100		
Добавление элемента с совпадающей датой в дерево поиска по имени	same.raw 1.1.2000		

Добавление элемента с существующей датой в дерево поиска по дате	wew.eee 1.1.1800		Элемент уже есть в дереве!
--	-------------------------	--	-------------------------------

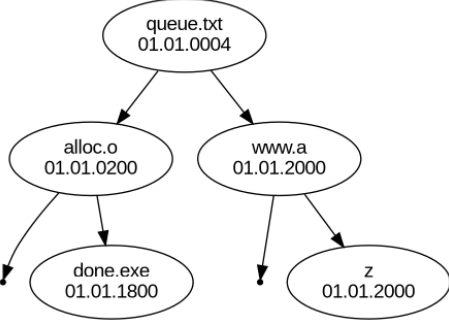

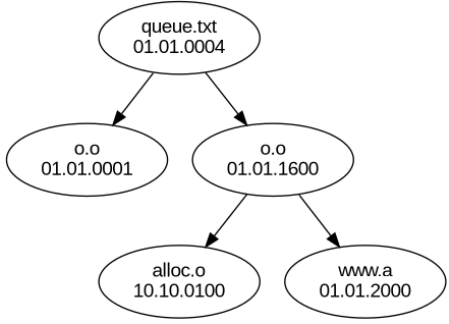

Удаление элемента из дерева

Описание теста	Название удаляемого файла	Входное дерево	Выходное дерево
Удаление элемента с обоими предками	queue.txt		
Удаление элемента без предков	done.exe		
Удаление единственного узла	www.a		Пустое дерево

Удаление в простом бинарном дереве	alloc.o	<pre> graph TD A("www.a 01.01.2000") --> B("alloc.o 01.01.0200") A --> C("zzz.ov 24.02.2022") B --> D("aaaaaa.tisd 23.11.1878") </pre>	<pre> graph TD A("www.a 01.01.2000") --> B("aaaaaa.tisd 23.11.1878") A --> C("zzz.ov 24.02.2022") </pre>
Удаление отсутствующего элемента	www	<pre> graph TD A("www.a 01.01.2000") --> B("alloc.o 01.01.0200") A --> C("zzz.ov 24.02.2022") B --> D("aaaaaa.tisd 23.11.1878") </pre>	Файл с заданным именем не найден

Перестроение дерева

Описание теста	Входное дерево	Выходное дерево
Успешное перестроение из даты в имя	<pre> graph TD A("queue.txt 01.01.0004") --> B("o.o 01.01.1600") A --> C(()) B --> D("alloc.o 10.10.0100") B --> E("www.a 01.01.2000") </pre>	<pre> graph TD A("queue.txt 01.01.0004") --> B("o.o 01.01.1600") A --> C("www.a 01.01.2000") B --> D("alloc.o 10.10.0100") B --> E(()) </pre>
Успешное перестроение из имени в дату	<pre> graph TD A("queue.txt 01.01.0004") --> B("o.o 01.01.1600") A --> C("www.a 01.01.2000") B --> D("alloc.o 10.10.0100") B --> E(()) </pre>	<pre> graph TD A("queue.txt 01.01.0004") --> B(()) A --> C("o.o 01.01.1600") C --> D("alloc.o 10.10.0100") C --> E("www.a 01.01.2000") </pre>

<p>Совпадающие даты при перестроении</p>	 <pre> graph TD queue["queue.txt 01.01.0004"] --> alloc["alloc.o 01.01.0200"] queue --> www["www.a 01.01.2000"] alloc --> done["done.exe 01.01.1800"] www --> z["z 01.01.2000"] </pre>	<p>Невозможно сменить вид деревя!</p>  <pre> graph TD queue["queue.txt 01.01.0004"] --> alloc["alloc.o 01.01.0200"] queue --> www["www.a 01.01.2000"] alloc --> done["done.exe 01.01.1800"] www --> z["z 01.01.2000"] </pre>
<p>Совпадающие имена при перестроении</p>	 <pre> graph TD queue["queue.txt 01.01.0004"] --> oo1["o.o 01.01.0001"] queue --> oo2["o.o 01.01.1600"] oo2 --> alloc["alloc.o 10.10.0100"] oo2 --> www["www.a 01.01.2000"] </pre>	<p>Невозможно сменить вид деревя!</p>  <pre> graph TD queue["queue.txt 01.01.0004"] --> oo1["o.o 01.01.0001"] queue --> oo2["o.o 01.01.1600"] oo2 --> alloc["alloc.o 10.10.0100"] oo2 --> www["www.a 01.01.2000"] </pre>

Результаты сравнения

Для анализа я измерил время удаления в дереве, построенном по алфавиту, со временем перестроения дерева по дате обращения и удаления в нем в дереве из 1000 элементов.

Результаты сравнения (в наносекундах)

Тестовые случаи	Время удаления в изначальном дереве	Время перестроения	Время удаления при перестроении	Суммарное время при перестроении	Выигрыш времени от перестроения
Остаются все элементы	10081	153897	38	153935	-1426.98%
Удаляется половина элементов	26275	153935	13178	167113	-536.02%
Удаляются все элементы	28981	154055	29063	183118	-531.86%

Таким образом, при удалении всех элементов, время удаления в дереве поиска и простом бинарном дереве равно, а при отсутствии удаляемых элементов, удаление в дереве поиска в сотни раз быстрее. Однако это преимущество уничтожается временем перестроения из бинарного дерева в дерево поиска, так как эта операция занимает в разы дольше удаления, что приводит к тому, что суммарное время удаления в дереве поиска в 5-15 раз больше чем в бинарном дереве.

Долгое перестроение по сравнению с полным обходом обусловлено тем, что сложность перестроения $O(n \log_2 n)$, так как для каждого элемента нужно его вставить за $\log_2 n$, а сложность полного обхода $O(n)$, в то же время сложность удаления элементов находится в пределах от $O(\log_2 n)$ до $O(n)$ в зависимости от отношения количества удаляемых элементов к общему числу элементов.

Сниженное в почти три раза время удаления для бинарного дерева, в котором не нужно удалять элементы, по сравнению со случаями, где нужно удалять, связано с тем, что само удаление тоже занимает время, в частности на поиск самого левого листа в правом поддереве.

Вывод

При решении задач, которые требуют хранения данных с иерархической структурой, целесообразно, использовать такую структуру данных, как дерево.

При хранении данных, которые требуют не только быстрого поиска, но и быстрого удаления и вставки (то есть отсортированные динамические массивы не подходят) целесообразно, использовать такую структуру данных, как бинарное дерево поиска.

При выполнении операций над бинарным деревом поиска, в котором условие применения операции не совпадает с ключом, по которому построено дерево (или над простым бинарным деревом), целесообразность изначального перестроения дерева в дерево поиска, зависит от того, будет ли данная операция применяться многократно. В случае если операция будет применена однократно, то весь выигрыш от быстрого применения операции, исчезает из-за долгого перестроения дерева.

Ответы на контрольные вопросы

1. Что такое дерево? Как выделяется память под представление деревьев?

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

Дерево определяется рекурсивно либо как пустая структура (пустое дерево), либо как узел с конечным числом древовидных структур этого же типа.

2. Какие бывают типы деревьев?

Бывают: произвольные деревья (неограниченное количество потомков у вершины), бинарные деревья (максимум 2 потомка у вершины), бинарные деревья поиска (бинарное дерево, в котором все левые потомки меньше предка, а правые - больше), идеально сбалансированные деревья (число вершин в левом и правом поддеревьях отличается не более чем на единицу).

3. Какие стандартные операции возможны над деревьями?

Обход дерева(префиксный, инфиксный, постфиксный), 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 меньше предка, а правые – больше.