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

Вариант 2

Студент **Ширяев А.А.**

Группа **ИУ7-33Б**

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

Студент _____ **Ширяев А.А.**

Преподаватель _____**Силантьева. А.В**

2024

Цель работы — получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

Условие задачи

Вариант 2

Построить дерево в соответствии со своим вариантом задания. Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

Построить двоичное дерево поиска, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Определить количество вершин дерева, содержащих слова, начинающиеся на указанную букву. Выделить эти вершины цветом. Сравнить время поиска начинающихся на указанную букву слов в дереве и в файле.

ПРИМЕЧАНИЕ! В отчёте будут использоваться обозначения для действий:

<read> - Чтение дерева из файла

<add> - Добавление элемента в дерево

<remove> - Удаление элемента из дерева по ключу

<search> - Поиск дерева по ключу

<pre_order_output> - Префиксный обход дерева и создание файла с визуализацией дерева

<in_order_output> - Инфиксный обход дерева и создание файла с визуализацией дерева

<post_order_output> - Постфиксный обход дерева и создание файла с визуализацией дерева

<find> - Нахождение элементов, начинающихся на букву и их выделение

<stat>- Вывод статистики по работе дерева в зависимости от высоты и степени ветвления и сравнение времени поиска начинающихся на указанную букву слов в дереве и в файле.

<exit> - Выход из программы

(ПРИМЕЧАНИЕ! Под деревом подразумевается двоичное дерево поиска, в вершинах которого слова)

Описание техзадачи

Описание исходных данных

Данные на входе: Меню. Код действия. Далее для каждого действия:

<read> - Файл со строками

<add> - Строка нового элемента, дерево (при наличии)

<remove> - Строка-ключ одного из элементов дерева, дерево

<search> - Строка-ключ одного из элементов дерева, дерево

<pre_order_output> - Дерево

<in_order_output> - Дерево

<post_order_output> - Дерево

<find> - Буква, дерево

<stat> - Буква, файл для статистики по покраске элементов, начинающихся на букву

<exit> - -

Данные на выходе:

<read> - Дерево

<add> - Дерево

<remove> - Дерево

<search> - Информация о нахождении элемента, кол-во сравнений

<pre_order_output> - Дерево, изображение-визуализация дерева

<in_order_output> - Дерево, изображение-визуализация дерева

<post_order_output> - Дерево, изображение-визуализация дерева

<find> - Дерево

<stat> - Статистика зависимости времени работы сортировки и поиска дерева от высоты и степени ветвления

<exit> - -

Описание задачи, реализуемой программой

Программа реализует ряд действий:

<read> - Чтение дерева из файла

<add> - Добавление элемента в дерево

<remove> - Удаление элемента из дерева по ключу

<search> - Поиск дерева по ключу

<pre_order_output> - Префиксный обход дерева и создание файла с визуализацией дерева

<in_order_output> - Инфиксный обход дерева и создание файла с визуализацией дерева

<post_order_output> - Постфиксный обход дерева и создание файла с визуализацией дерева

<find> - Нахождение элементов, начинающихся на букву и их выделение

<stat>- Вывод статистики по работе дерева в зависимости от высоты и степени ветвления и сравнение времени поиска начинающихся на указанную букву слов в дереве и в файле.

<exit> - Выход из программы

Способ обращения к программе

Для обращения к программе запускается файл app.exe.

Описание возможных аварийных ситуаций и ошибок пользователя

Программа может не вывести результат, а вывести сообщение об ошибке. Данная ситуация может произойти при условии:

Ошибки пользователя

1. Неверный код действия

Аварийные ситуации

1. Программа не смогла выделить необходимую память для работы (для добавления элемента в дерево)
2. Программа создала некорректный файл с деревом (если в статистике слишком много элементов)

Описание внутренних СД

Информация о вершине

Информация о вершине представляет собой структуру на языке Си, состоящую из:

```c
struct node
{
    char *data;
    color_t color;

    node_t *left;
    node_t *right;
};
```

*data - указатель на строку вершины

color - значение цвета

*left - указатель на левого потомка вершины

*right - указатель на правого потомка вершины

Информация о цвете

Информация о цвете представляет собой перечисляемый тип на языке Си со значениями:

```c
typedef enum {NONE, RED, GREEN, BLUE} color_t;
```

NONE - без цвета

RED - красный цвет

GREEN - зелёный цвет

BLUE - синий цвет

Информация о строке

```c
char *data = NULL;
```

Информация о строке представлена в виде указателя на символьный тип char

Описание алгоритма

Для начала считывается код действия. Далее в зависимости от выбранного действия:

<read>

```
int node_read_by_file(char *filedata, node_t **root)
```

- С клавиатуры считывается название файла.
- Файл проверяется на корректность
- Программа читает информацию в файле, выделяет память под элемент, заполняет его и добавляет в новое дерево.

<add>

```
node_t *node_add(node_t *node, node_t *elem)
```

- Считывается строка-ключ нового элемента
- Выделяется память под новый элемент дерева
- Проверяется наличие элемента с такой же строкой-ключом
- Новый элемент добавляется в дерево (при отсутствии эквивалентного)

<remove>

```
void node_delete(node_t **node, char *data)
```

- Считывается строка-ключ элемента
- Проверяется наличие элемента с такой же строкой-ключом
- Программа удаляет элемент (при наличии)

<search>

```
node_t *node_search(node_t *node, char *data, int *compares)
```

- Считывается строка-ключ элемента
- Производится поиск элемента с такой же строкой-ключом
- Выводится результат поиска
- Выводится кол-во сравнений

<pre_order_output> <in_order_output> <post_order_output>

```
void node_output_pre_order(node_t *node, FILE *f)
```

```
void node_output_in_order(node_t *node, FILE *f)
```

```
void node_output_post_order(node_t *node, FILE *f)
```

- Выполняется соответствующий обход дерева
- Программа преобразует информацию о дереве в код на языке DOT
- Создается изображение дерева

```
void node_export_to_dot_eli(FILE *f, const char *node_data, node_t *node)
```

<find>

```
size_t node_count_and_color(node_t *head, char c)
```

- Считывается символ, с которого должна начинаться строка
- Программа проходится по дереву, закрашивая вершины, строка которых начинается с указанного символа

<stat>

```
int node_statistics(char *filename, char c)
```

- Считывается название файла для статистики и символ, по которому будут искаться элементы в дереве и в файле
    - Создаются: Идеально сбалансированное дерево и список-дерево
    - Замеряются: Времена сортировки и поиска максимально вложенного элемента, а также кол-во сравнений при поиске (кол-во сравнений при поиске и кол-во сравнений при добавлении будут равными)
    - Высчитываются средние значения времён
    - Выводится таблица с подсчитанными данными
- Далее введенное название файла проверяется на корректность
- Рассчитываются средние времена поиска. Полученные данные сравниваются

<exit>

```
case CODE_EXIT:
    tree = node_free(tree);
    str_free(&filename, &filename_size);
    str_free(&data, &data_size);

    flag = false;

    break;
```

- Освобождается память, выделенная под динамически выделенные данные программы (Дерево и строки, необходимые для выполнения действий с вводом строк)
- Программа завершает свою работу

Набор тестов

В ходе выполнения лабораторной работы были написаны тесты для проверки работы программы

Позитивные тесты

Номер теста	Входные данные	Ожидаемые выходные данные
01 - <read> Чтение корректного файла	1 test2.txt 6	IN-ORDER: 0 1 1 214 124126 1243124 15 26 351 421

		48 5 531 573 6 8p076 9 975 maepet mamaprivet
02 - <add> Добавление вершины в дерево	2 a 2 b 6	IN-ORDER: a b
03 - <remove> Исключение вершины из дерева	2 a 2 b 3 a 6	IN-ORDER: b
04 - <search> Поиск вершины	1	DATA WAS FOUNDED

по указанному ключу	test1.txt 4 573	SUCCESSFULLY Total compares: *Кол-во сравнений*
05 - <pre-order_output> Вывод префиксного обхода дерева	1 test1.txt 5	PRE-ORDER: mamaprivet 5 1 0 1243124 1 214 124126 421 351 26 15 48 6 573 531 9 8p076 975

		maepet
06 - <post_order_output> Выход из программы	1 test1.txt 7	POST-ORDER: 0 124126 1 214 15 26 351 48 421 1243124 1 531 573 8p076 maepet 975 9 6 5 mamaprivet
07 - <find> Нахождение	1	*Файл с деревом, где закрашены

элементов, начинающихся на данный символ	test1.txt 8 1 6	найденные элементы*
08 - <stat> Вывод статистики	9 1 test1.txt	*Статистика по зависимости от высоты и степени ветвления* *Статистика по поиску в дереве и в файле*
09 - <exit> Выход из программы	10	-

Негативные тесты

Номер теста	Входные данные	Выходные данные
01 - Код неправильный	15	INVALID CODE
02 - Код с иными символами	1.1	INVALID CODE
03 - <remove> Нет дерева	3	NO DATA
04 - <add> Найден эквивалентный элемент	1 test1.txt 2	THIS VALUE IS ALREADY IN TREE

	1	
05 - <search> Нет дерева	4	NO DATA
06 - <remove> Нет элемента с данным ключом	1 test1.txt 3 yotumidore	ELEMENT IS NOT FOUND
07 - <pre_order_output> Нет дерева	5	NO DATA
08 - <in_order_output> Нет дерева	5	NO DATA
09 - <post_order_output> Нет дерева	5	NO DATA
10 - <stat> Некорректное название файла	9 1 tmp.tmp	*Статистика по зависимости от высоты и степени ветвления* INVALID FILE TO CONTINUE STATISTICS

Сравнение эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

Ниже представлена статистика по зависимости времени от высоты и степени ветвления (для получения значения использовалось 1000 итераций (Значение MAX_ITER_COUNT))

```
#define MAX_ITER_COUNT 1000
```

Статистика

```
STATISTICS (time in nsec) (total iteration: 1000)
n count |h IDEAL |h WORST |TREE SORT(IDEAL)|TREE SORT(WORST)|TREE FIND(IDEAL)|TREE FIND(WORST)|COMPARES (IDEAL)|COMPARES (WORST)|  best t (SORT) |  best t (FIND) |
3       |1       |3       |90.571000       |91.716000       |34.914000       |45.633000       |2               |3               |      IDEAL     |      IDEAL     |
7       |2       |7       |277.573000      |297.532000      |39.132000       |66.330000       |3               |7               |      IDEAL     |      IDEAL     |
15      |3       |15      |634.951000      |1115.957000     |45.743000       |125.744000      |4               |15              |      IDEAL     |      IDEAL     |
31      |4       |31      |1601.525000     |4449.674000     |57.877000       |253.410000      |5               |31              |      IDEAL     |      IDEAL     |
63      |5       |63      |3474.711000     |29574.768000    |86.390000       |534.545000      |6               |63              |      IDEAL     |      IDEAL     |
127     |6       |127     |11817.507000    |78717.875000    |65.876000       |1140.008000     |7               |127             |      IDEAL     |      IDEAL     |
255     |7       |255     |28960.813000    |360859.967000   |70.547000       |2124.044000     |8               |255             |      IDEAL     |      IDEAL     |
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.667747 to fit
511     |8       |511     |73403.815000    |1466875.876000  |80.143000       |4452.003000     |9               |511             |      IDEAL     |      IDEAL     |
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.551828 to fit
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.333598 to fit
1023    |9       |1023    |169478.264000   |6166056.509000  |80.120000       |11983.721000    |10              |1023            |      IDEAL     |      IDEAL     |
```

(ПРИМЕЧАНИЕ! Внутри таблица прерывается сообщениями утилиты dot. Это происходит из-за величины дерева и является совершенно нормальным поведением программы)

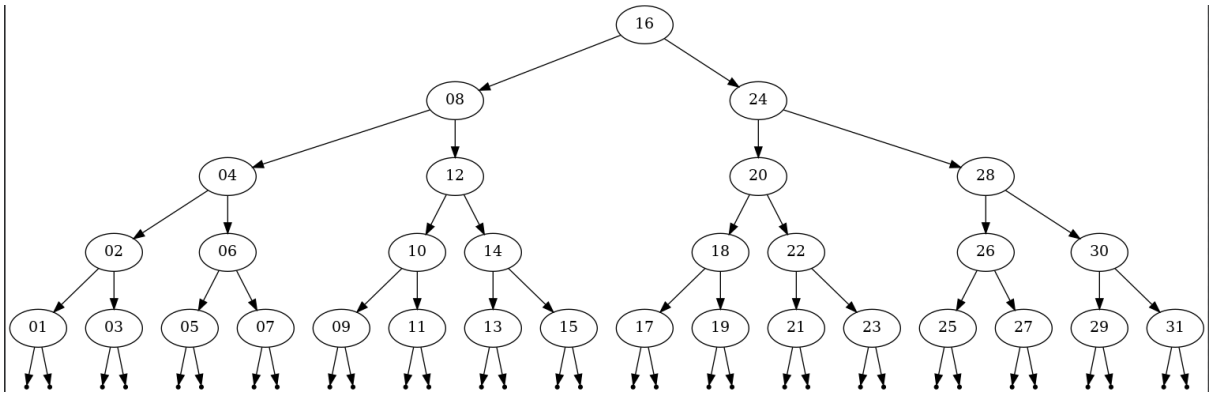
(ПРИМЕЧАНИЕ! WORST - список-дерево, IDEAL - идеально сбалансированное дерево. Для получения результатов используется элемент с максимальной глубиной)

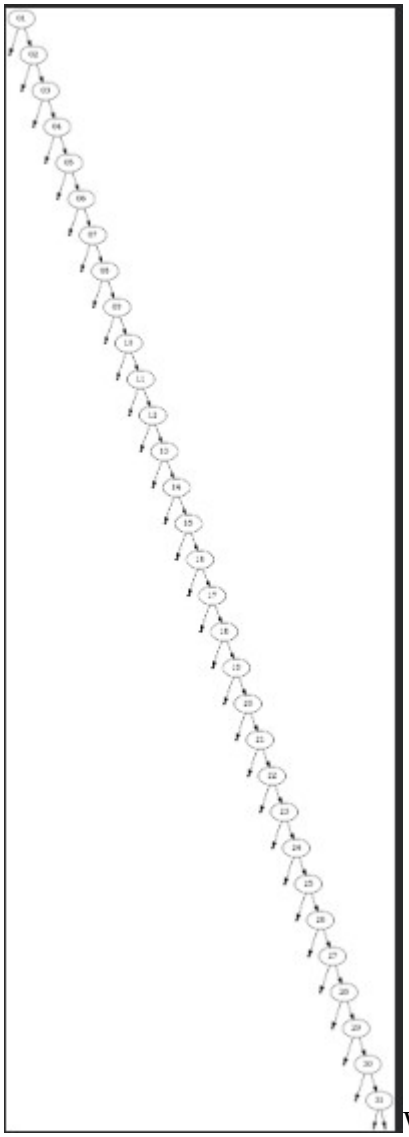
На основе полученных данных можно сделать следующие выводы:

- При маленьких высотах времена работы операций примерно равные
- При возрастании высоты время сортировки и поиска у 2-х деревьев возрастает, несмотря на степень ветвления.
- При увеличении высоты IDEAL дерева время поиска, в отличие от WORST дерева, изменяется на небольшую величину. Это связано с различной сложностью поиска (у IDEAL дерева — $O(\log_2 n)$, а у WORST дерева - $O(n)$)
- Степень ветвления сильно влияет на время работы : при равных кол-вах элементов, но разной степенью наблюдается колоссальная разница во времени при сортировке и поиске: при возрастании кол-ва элементов IDEAL дерево во много раз превосходит по времени WORST дерево.

Примеры данных:

IDEAL дерево с высотой 4 и количеством элементов 31. Степень ветвления 2



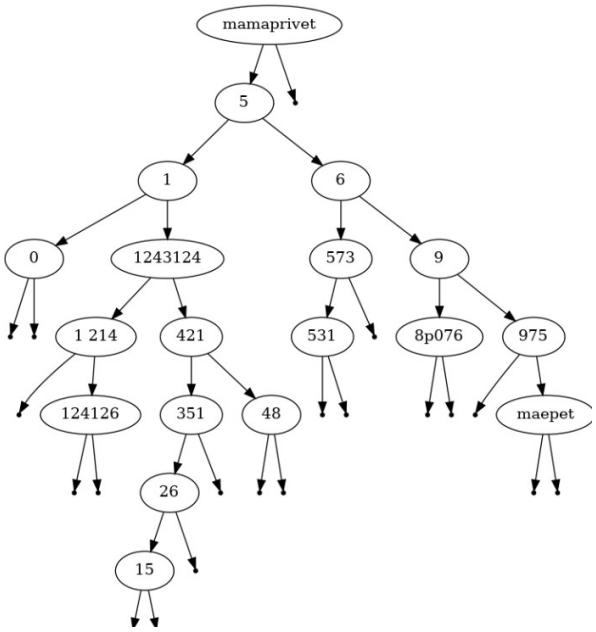


WORST дерево при высоте 31 и количеством элементов 31. Степень ветвления 1

Сравнение времени поиска слов, начинающихся на указанную букву, в дереве и в файле

```
Average tree search: 136.186000
Average file search: 1096044.170000
tree method is faster than file method on 804814.129206%
```

При сравнении времени поиска слов, начинающихся на символ «5», в дереве и в файле (вся информация для сравнения предоставлена в файле test1.txt) наблюдается колоссальная разница во времени (время показано в наносекундах). Такой большой отрыв обусловлен эффективностью работы с деревом, а также большими затратами при работе с файлом.



test1.txt

lab6 > test1.txt

```
1   mamaprivet
2   5
3   6
4   1
5   0
6   9
7   1243124
8   421
9   351
10  573
11  975
12  531
13  8p076
14  1
15  maepet
16  1 214
17  124126
18  26
19  48
20  15
21
```

Выводы по проделанной работе

Двоичные деревья поиска удобно использовать при наличии иерархических связей между элементами. С помощью них реализуются эффективные по времени алгоритмы сортировки и поиска. При увеличении количества элементов, двоичное дерево превосходит обработку других структур данных, но может быть неэффективным по памяти, так как для хранения каждого узла требуется и указатели на потомков, что увеличивает затраты по памяти.

Скорость работы сортировки и поиска дерева зависят от высоты дерева и степени ветвления.

Скорость поиска слов, начинающихся на букву, в дереве значительно превосходит скорость поиска слов в файле.

Контрольные вопросы

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

Дерево - нелинейная структура данных, используемая при представлении иерархических связей, имеющих отношения “один ко многим”

Память под дерево выделяется **динамически** (в процессе выполнения программы).

2. Какие бывают типы деревьев?

K-ичное дерево, двоичное дерево поиска, сбалансированные деревья поиска (AVL, RBT, B-деревья), деревья оптимального поиска

3. Какие стандартные операции возможны над деревьями?

Обход (префиксный, инфиксный, постфиксный), включение, исключение, поиск, сортировка

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска - дерево, обладающая следующими свойствами:

1) Степень ветвления <= 2 - “левый” и “правый” непосредственный потомок
2) “Левый” потомок всегда меньше вершины узла, но правый потомок всегда “больше” вершины узла.
3) Каждый потомок - двоичное дерево поиска