# Лабораторная работа №6 по дисциплине “Типы и структуры данных”

Подготовил: Ширяев Алексей

Группа: ИУ7-33Б

[**Лабораторная работа №6 по дисциплине “Типы и структуры данных” 1**](#_gjdgxs)

[**Условие задачи 3**](#)

[**Описание техзадачи 4**](#)

[**Описание исходных данных 4**](#)

[**Описание задачи, реализуемой программой 5**](#)

[**Способ обращения к программе 5**](#)

[**Описание возможных аварийных ситуаций и ошибок пользователя 5**](#)

[**Описание внутренних СД 6**](#)

[**Информация о вершине 6**](#)

[**Информация о цвете 7**](#)

[**Информация о строке 7**](#)

[**Описание алгоритма 8**](#)

[**Набор тестов 10**](#)

[**Позитивные тесты 10**](#)

[**Негативные тесты 14**](#)

[**Сравнение эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления. 16**](#)

[**Контрольные вопросы 19**](#)

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

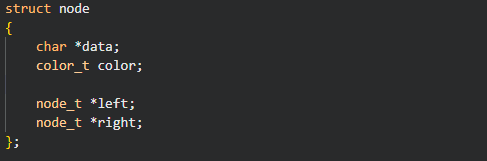
Аварийные ситуации

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:



\*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:



NONE - без цвета

RED - красный цвет

GREEN - зелёный цвет

BLUE - синий цвет

## Информация о строке



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>



* 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>



* 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>



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

<search>



* 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>







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



<find>



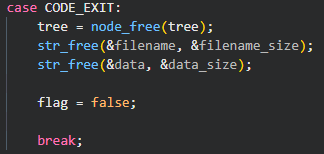
* Считывается символ, с которого должна начинаться строка
* Программа проходится по дереву, закрашивая вершины, строка которых начинается с указанного символа

<stat>



* 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>



* 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 по указанному ключу | 1  test1.txt  4  573 | DATA WAS FOUNDED 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 элементов, начинающихся на данный символ | 1  test1.txt  8  1  6 | \*Файл с деревом, где закрашены 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

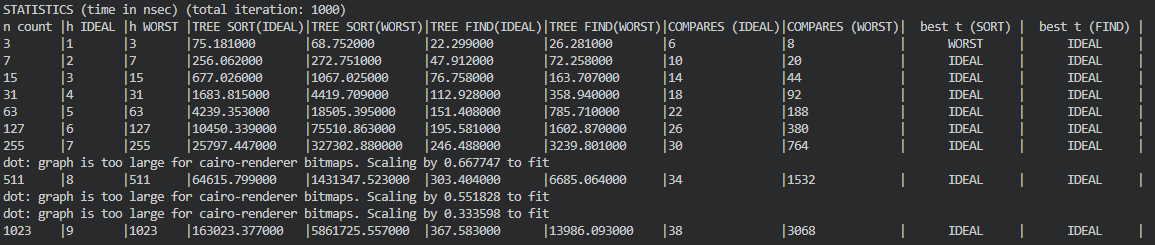
| Номер теста | Входные данные | Выходные данные |
| --- | --- | --- |
| 01 - Код неправильный | 15 | INVALID CODE |
| 02 - Код с иными символами | 1.1 | INVALID CODE |
| 03 - <remove> Нет дерева | 3 | NO DATA |
| 04 - <add> Найден эквивалентный элемент | 1  test1.txt  2  1 | THIS VALUE IS ALREADY IN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



Статистика

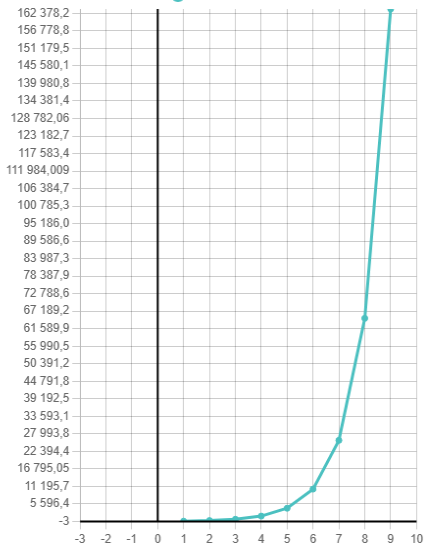


(ПРИМЕЧАНИЕ! Внутри таблица прерывается сообщениями утилиты dot. Это происходит из-за величины дерева и является совершенно нормальным поведением программы)

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

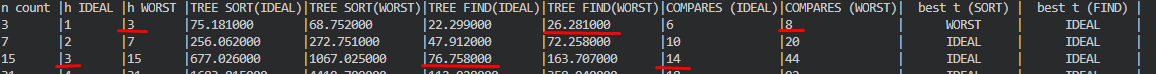
На основе полученных данных можно сделать следующие выводы:

* При маленьких высотах времена примерно равные (Можем увидеть, что WORST вариант сортируется немного быстрее, чем IDEAL, так как в функции сортировки сначала проверяется правый потомок, а WORST является списком-деревом, элементы которого сортированы ПО ВОЗРАСТАНИЮ (т.е используются только правые потомки))
* При возрастании высоты время сортировки и поиска возрастает, несмотря на степень ветвления.



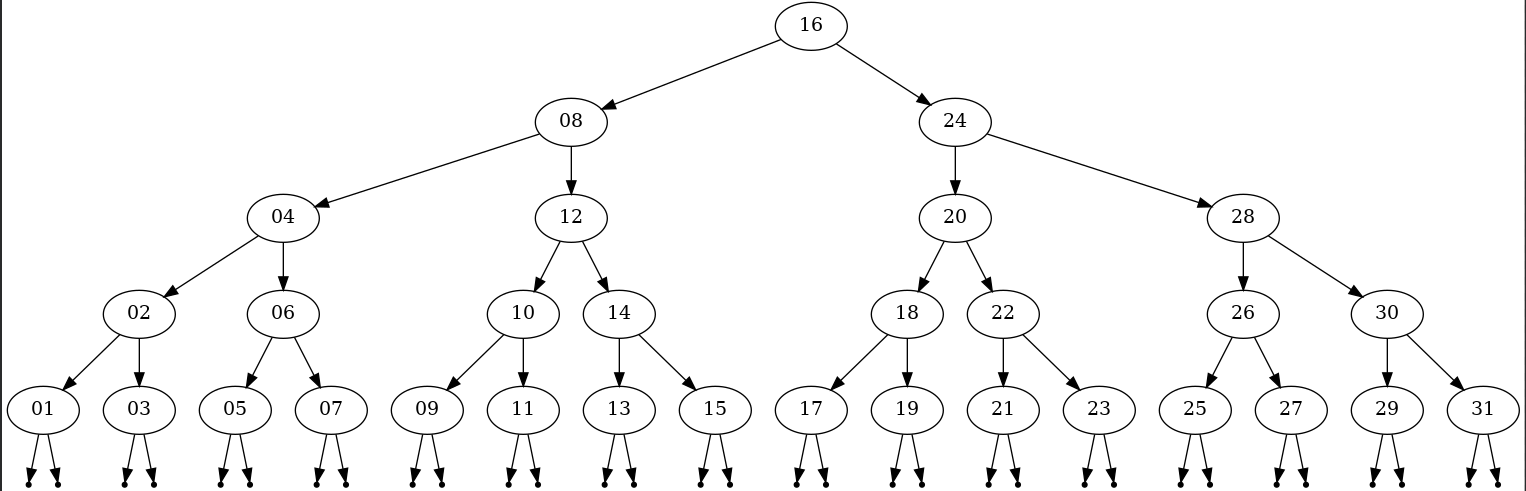
(Пример: Зависимость времени сортировки IDEAL от высоты)

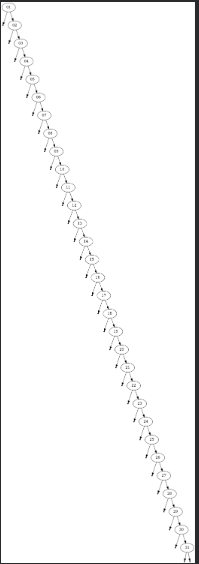
* При равных кол-вах элементов, но разной степенью ветвления наблюдается колоссальная разница во времени при сортировке и поиске: при возрастании кол-ва элементов IDEAL дерево во много раз превосходит по времени WORST дерево.
* Кол-во сравнений тоже влияют на скорость работы: При равной высоте, но разной степенью ветвления нахождение элемента в IDEAL дереве производилось за большее время, чем в WORST дереве. В данном случае такая разница из-за некоторых сравнений и левого потомка в IDEAL дереве (В реализации обхода дерева проверяется сначала равенство, потом правый потомок, потом левый потомок)



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

Выводы по проделанной работе

Дерево - тип данных, позволяющий более эффективно работать с данными.

Скорость работы сортировки и поиска дерева зависит от высоты дерева, степени ветвления. Также данные операции зависят от количества сравнений, которое в свою очередь зависит от реализации.

**(ПРИМЕЧАНИЕ! В ходе работы программы фрагментация не наблюдалась, но при работе с утилитой Valgrind фрагментация наблюдалась)**

# 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