|  |  |
| --- | --- |
| Изображение выглядит как герб, эмблема, символ, нашивка  Автоматически созданное описание | **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»**

***<Обработка деревьев*>**

**Вариант № 3**

Студент: **<*Ермаков И.Г..*>**

Группа: **<*ИУ7-32Б*>**

Преподаватель: **<*Барышникова М.Ю.*>**

**2024 г.**

**Содержание**

**Цель работы…………………………………………………………………………3**

**Условие задачи……………………………………………………………………...3**

**Описание ТЗ………………………………………………………………………...3**

**Входные данные……………………………………………………………...4**

**Выходные данные……………………………………………………………5**

**Действие программы………………………………………………………..5**

**Обращение к программе……………………………………………………5**

**Аварийные ситуации………………………………………………………..5**

**Описание структур данных……………………………………………………….6**

**Описание основных сигнатур функций…………………………………………8**

**Описание алгоритма……………………………………………………………….9**

**Сравнение эффективности работы алгоритма………………………………..10**

**Тесты………………………………………………………………………………..11**

**Вывод……………………………………………………………………………….13**

**Ответы на контрольные вопросы………………………………………………15**

**Цель работы**

Получить навыки применения двоичных деревьев, реализовать основные операции над деревьями: обход деревьев, включение, исключение и поиск узлов.

**Условие задачи**

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 бинарное дерево поиска, в вершинах которого находятся слова из текстового файла (предполагается, что исходный файл не содержит повторяющихся слов). Вывести его на экран в виде дерева. Удалить все слова, начинающиеся на указанную букву. Сравнить время удаления слов, начинающихся на указанную букву, в дереве и в файле

**Описание технического задания**

**Входные данные:**

Изначально, пользователя встречает окно выбора опций и описание задания

|  |
| --- |
| Доступные опции  1 - Сгенерировать файл для обработки указанной длины  2 - Построение бинарного дерева поиска из файла  3 - Графический вывод дерева  4 - Обход дерева  5 - Вставка элемента в дерево и одновременная генерация обновленного дерева  6 - Удаление элементов, начинающихся на введенный символ и одновременная генерация обновленного дерева  7 - Сравнение удаления и поиска в массиве и дереве  8 - Поиск элемента в дереве  9 - Очистка файла и дерева  0 - Выход |

После корректного выбора опции, пользователя встречает приглашение к вводу, для дальнейших операций

**Выходные данные**

В зависимости от каждой опции выводится соответствующий результат. В случае ошибки на экран выводится что пользователь сделал не так.

**Действие программы:**

Программа работает до тех пор пока пользователь не введет 0, в случае некорректного ввода программа будет предлагать пользователю ввести еще раз.

**Обращение к программе:**

Программа собирается с помощью утилиты make

Затем запускается ./app.exe file\_in где file\_in – файл, где генерируются данные для дерева

**Аварийные ситуации:**

Некорректный ввод опции

**Описание структур данных:**

|  |
| --- |
| typedef struct node\_t  {  char \*value;  struct node\_t \*left;  struct node\_t \*right;  } node\_t; |

Данная структура описывает дерево двоичного поиска, value – слово в каждом узле, указатели left и right соответственно указывают на левого и правого потомка.

**Описание основных сигнатур функций**

|  |
| --- |
| node\_t \*create\_node(char \*value);  node\_t \*insert(node\_t \*root, char \*value);  node\_t \*delete\_node(node\_t \*node);  node\_t \*search\_node(node\_t \*node, const char \*value);  void prefix\_traversal(node\_t \*root, bool is\_measuring);  void infix\_traversal(node\_t \*root, bool is\_measuring);  void postfix\_traversal(node\_t \*root, bool is\_measuring); |

Функция create\_node – создает узел дерева

Функция insert – вставляет элемент в дерево

Функция delete\_node – удаляет узел из дерева

Функция search\_node – совершает поиск элемента в дереве

Функции prefix\_traversal, infix\_traversal, postfix\_traversal – совершают обход дерева соответственно (префиксный, инфиксный, постфиксный).

**Описание алгоритма**

### Алгоритм удаления узлов в двоичном дереве поиска

Удаление узла в двоичном дереве поиска (BST) зависит от количества его дочерних узлов:

1. **Листовой узел (нет потомков):**
   * Просто удаляется, и его родитель получает NULL вместо ссылки на узел.
2. **Узел с одним потомком:**
   * Удаляемый узел заменяется своим единственным потомком (левым или правым).
3. **Узел с двумя потомками:**
   * Находим минимальный узел в правом поддереве удаляемого узла (наименьшее значение в его правой ветви).
   * Заменяем значение удаляемого узла значением найденного узла.
   * Удаляем найденный минимальный узел из правого поддерева.

Эта стратегия позволяет сохранить свойства дерева поиска (левое поддерево меньше, правое больше).

### Алгоритм поиска узла в двоичном дереве поиска

Поиск в BST опирается на свойства упорядоченности:

1. Сравниваем искомое значение с текущим узлом:
   * Если совпадает, узел найден.
   * Если искомое значение меньше текущего, переходим в левое поддерево.
   * Если больше, переходим в правое поддерево.
2. Если узел с искомым значением отсутствует, возвращаем NULL.

Поиск в сбалансированном BST выполняется за логарифмическое время — O(log n) в среднем.

### Обход дерева

Обходы дерева позволяют обработать все узлы в определенном порядке. Основные типы:

1. **Прямой обход (Prefix order):**
   * Обрабатываем текущий узел, затем левое поддерево, потом правое.
   * Пример: используется для копирования дерева.
2. **Симметричный обход (Infix order):**
   * Обрабатываем левое поддерево, текущий узел, затем правое поддерево.
   * Пример: возвращает элементы дерева в отсортированном порядке.
3. **Обратный обход (Postfix order):**
   * Обрабатываем левое поддерево, правое поддерево, затем текущий узел.
   * Пример: используется для удаления дерева.

Для реализации обходов часто используются рекурсивные функции, которые позволяют легко обрабатывать структуру дерева.

**Сравнение эффективности работы алгоритмов**

**Методология исследования:**

Для исследования времени работы функции на каждый размер проводится по 1000 измерений, затем значение усредняется.

Таблица замеров времени для функций удаления всех слов, начинающихся на данный символ:

|  |  |  |
| --- | --- | --- |
| Размер | Удаление элементов в двоичном дереве поиска (такты) | Удаление элементов в файле (такты) |
| 100 | 1577 | 1535417 |
| 500 | 10212 | 1534977 |
| 1000 | 22224 | 1730725 |
| 5000 | 141725 | 17526305 |
| 10000 | 342676 | 11782332 |

Таблица замеров времени для функций поиска слова

|  |  |  |
| --- | --- | --- |
| Размер | Стек на статическом массиве (такты) | Стек на динамическом массиве (такты) |
| 100 | 486 | 54128 |
| 500 | 706 | 84746 |
| 1000 | 733 | 117752 |
| 5000 | 1862 | 839464 |
| 10000 | 1865 | 841089 |

**Тесты:**

|  |
| --- |
| 1 - Сгенерировать файл для обработки указанной длины  2 - Построение бинарного дерева поиска из файла  3 - Графический вывод дерева  4 - Обход дерева  5 - Вставка элемента в дерево и одновременная генерация обновленного дерева  6 - Удаление элементов, начинающихся на введенный символ и одновременная генерация обновленного дерева  7 - Сравнение удаления и поиска в массиве и дереве  8 - Поиск элемента в дереве  9 - Очистка файла и дерева  0 - Выход |

|  |  |  |
| --- | --- | --- |
| Описание теста | Входные данные | Выходные данные |
| Некорректный ввод опции | -1 | Ошибка ввода опции |
| Некорректный ввод опции | 10 | Ошибка ввода опции |
| Некорректный ввод опции | a | Ожидалась скобка! |
| Ввод кол-ва элементов для генерации превышает максимум (10000) | 10001 | Максимальное кол-во элементов превышено! (10000) |
| Вывод всех обходов | 3 – размер дерева | Префиксный обход  3 1 2   Постфиксный обход  2 1 3   Инфиксный обход  1 2 3 |
| Вставка элемента | 4 | Файл с деревом обновлен! |
| Удаление элемента | 4 | Префиксный обход  1 5 2  Постфиксный обход  2 5 1  Инфиксный обход  1 2 5 |
| Поиск существующего элемента | 2 | Элемент найден |
| Поиск несуществующего элемента | 99 | Элемент не найден |

Этапы построения дерева

Исходное дерево

Вывод:

#### Удаление элементов, начинающихся на заданный символ

Сравнение времени выполнения операций в двоичном дереве поиска и в файле:

* Для 100 элементов:
  + Двоичное дерево поиска: 1577 тактов.
  + Файл: 1535417 тактов.
  + Удаление в дереве быстрее на: 99.9%
* Для 1000 элементов:
  + Двоичное дерево поиска: 22224 тактов.
  + Файл: 1730725 тактов.
  + Удаление в дереве быстрее на: 98.7%
* Для 10,000 элементов:
  + Двоичное дерево поиска: 342676 тактов.
  + Файл: 11782332 тактов.
  + Удаление в дереве быстрее на: 97.1%

#### 2. Поиск слова

Сравнение времени поиска слова:

* Для 100 элементов:
  + Двоичное дерево поиска: 486 тактов.
  + Файл: 54128 тактов.
  + Поиск на статическом массиве быстрее на: 99.1%
* Для 1000 элементов:
  + Двоичное дерево поиска: 733 тактов.
  + Файл: 117752 тактов.
  + Поиск на статическом массиве быстрее на: 99.4%
* Для 10,000 элементов:
  + Двоичное дерево поиска: 1865 тактов.
  + Файл: 841089 тактов.
  + Поиск на статическом массиве быстрее на: 99.8%

**Ответы на контрольные вопросы:**

**Что такое дерево?**  
**Дерево** — это иерархическая структура данных, состоящая из узлов. Узлы соединены ребрами, а структура начинается с одного корневого узла и разветвляется на поддеревья

**Выделение памяти**

В **динамическом представлении** узлы дерева реализуются с помощью структур, содержащих данные и указатели на потомков (например, для двоичных деревьев — два указателя: left и right).

**Какие бывают типы деревьев?**

**Общее дерево :** Узел может иметь произвольное количество потомков.

**Двоичное дерево :** Каждый узел может иметь не более двух потомков: левый и правый.

**Полное двоичное дерево :** Все узлы либо листовые, либо имеют ровно два потомка.

**Сбалансированное дерево :** Разница в высоте левого и правого поддерева для любого узла не превышает 1.

**Дерево двоичного поиска :** Для каждого узла все узлы в левом поддереве меньше текущего узла, а в правом — больше.

**Какие стандартные операции возможны над деревьями?**  
Добавление узлов, Удаление узлов, Поиск узлов, Обход дерева

**Что такое дерево двоичного поиска?**

Дерево двоичного поиска — это двоичное дерево, обладающее следующими свойствами:

Для каждого узла:

Все значения в левом поддереве меньше, чем значение текущего узла.

Все значения в правом поддереве больше, чем значение текущего узла.

Значения в узлах уникальны.