# **Лабораторная работа №6 по дисциплине «Типы и структуры данных»**

# Обработка деревьев

Говязин Сергей (ИУ7-31)

**Цель работы:**

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

**Задание (Вариант 0):**

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 из букв вводимой строки. Вывести его на экран в виде дерева. Выделить цветом все буквы, встречающиеся более одного раза. Удалить из дерева эти буквы. Вывести оставшиеся элементы дерева при постфиксном его обходе. Сравнить время удаления повторяющихся букв из дерева и из строки.

**Исходные данные:**

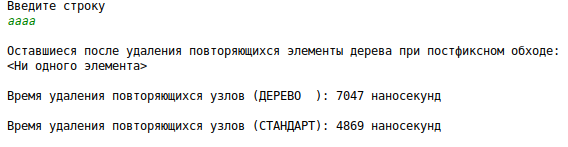
Доступ к программе осуществляется через консоль. Вводится непустая строка.

**Тесты**

Проверить ввод пустой строки, строки без повторяющихся букв, строки, состоящей из одинаковых букв

**Интерфейс программы:**

1) Интерфейс



2) Отображение дерева

**Внутренние структуры данных:**

**Node — вершина дерева**

**template**<**class** T>

**struct** Node {

Node<T> \*left = **NULL**;

Node<T> \*right = **NULL**;

T data;

**unsigned int** id;

Node(T \_data, **unsigned int** \_id) : data(\_data), id(\_id) {}

};

**Интерфейс — набор методов дерева**

**void** addNode(T data);

**void** remove(T data);

vector<T> LRCVector();

**void** visualize(**bool** full = **false**, string filename = **"graph.gr"**, string pic = **"graph.png"**);

**void** visualizeColorRepeating(**bool** full = **false**, string filename = **"graph.gr"**, string pic = **"graph.png"**);

**void** removeDublicates();

**Тесты**

|  |  |  |
| --- | --- | --- |
| **Строка** | **Постфиксный обход до удаления** | **Постфиксный обход после удаления** |
| abcd | b c d a | b c d a |
| bcda | a c d b | a c d b |
| addNode | N d d d e o a | N e o a |
| visualizeColorRepeating | C R a a e e e g i i i l l n o o p r s t u z v | C R g n p r s t u z v |
| aaaaa | a a a a a | <Ни одного элемента> |
| ababababa | a a a a b b b b a | <Ни одного элемента> |

**Сравнение времени удаления повторяющихся символов их строки (реализация деревом против стандартной реализации)**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Строка (длина)** | **Время удаления (Дервом), наносекунд** | **Время удаления (Стандартным способом), наносекунд** |
| **1** | **ababababa** | **12158** | **8070** |
| **2** | **qqwweerrttyyuuiiooppaassddffgghhjjkkllzzxxccvvbbnnmmmm** | **43252** | **125425** |
| **3** | **843** | **987241** | **4018607** |
| **4** | **151** | **316807** | **149907** |

Как видим, выгодно использовать дерево для хранения и удаления дубликатов-символов из строки, когда длина строки достаточно высока, превышает 150 символов, и встречается множество повторяющихся символов. Как показывает практика, в тексте, превышающем длиной 200 символов (на Латыни), все символы повторются. Если эти условия не выполняются, выгоднее и быстрее получается реализация без дерева.

**Вывод:**

Основным преимуществом двоичного дерева перед другими структурами данных является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки. Хранение и обработка дерева требует аккуратного обращение с памятью.

**Контрольные вопросы**

1. Что такое дерево?

Дерево – нелинейная структура данных, которая используется для представления иерархических связей «один ко многим». Дерево с базовым типом Т определяется рекурсивно: это либо пустая структура (пустое дерево), либо узел типа Т с конечным числом древовидных структур того же типа – поддеревьев.

2. Как выделяется память под представление деревьев?

Выделение памяти под деревья определяется типом их представления. Это может быть таблица связей с предками (№ вершины - № родителя), или связный список сыновей. Оба представления можно реализовать как с помощью матрицы, так и с помощью списков. При динамическом представлении деревьев (когда элементы можно удалять и добавлять) целесообразнее использовать списки – т.е. выделять память под каждый элемент динамически.

3. Какие бывают типы деревьев?

АВЛ-деревья, сбалансированные деревья, двоичные, двоичного поиска.

4. Какие стандартные операции возможны над деревьями?

Основные операции с деревьями: обход (инфиксный, префиксный, постфиксный), поиск элемента по дереву, включение и исключение элемента из дерева.

5. Что такое дерево двоичного поиска?

Дерево двоичного поиска – дерево, в котором все левые потомки «моложе» предка, а все правые – «старше». Это свойство выполняется для любого узла, включая корень.