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 №06 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Обработка деревьев.

Студент **Батуев А.Г.**

Группа **ИУ7-36Б**

Вариант **1**

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

Студент _____ **Батуев А.Г.**

Преподаватель _____ **Никульшина Т.А.**

2024 г.

Условие задачи

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

Задание по варианту.
В текстовом файле содержатся целые числа. Построить двоичное дерево из чисел файла. Вывести его на экран в виде дерева. Определить количество узлов дерева на каждом уровне. Добавить число в дерево и в файл. Сравнить время добавления чисел в указанные структуры.

Техническое задание

На вход **программа получает**:

- опция из меню (цифра от 0 до 8)
- файл с данными
- данные от пользователя

Программа выводит:

- дерево в формате .dot
- элементы дерева в отсортированном виде
- кол-во узлов на каждом уровне
- результаты тестирования

Программа должна реализовывать создание, хранение, вывод дерева, а также операции добавления, удаления элемента из дерева, экспорт дерева в формат dot и тестирование основных функций (добавление, удаление, сортировка).

Обращение к программе осуществляется по указанию названия программы (./app.exe).

Необходимо внимательно **обработать возможные аварийные ситуации**, которые могут включать:

- неправильный ввод данных
- переполнение памяти
- удаление из пустого дерева

Описание внутренних структур данных

vertex_t — это структура, описывающая узел бинарного дерева. Узел содержит:

- value - целое число, представляющее значение узла.
- left - указатель на левый дочерний узел.
- right - указатель на правый дочерний узел.

tree_t — это структура, описывающая бинарное дерево. Она включает:

- root - указатель на корень дерева, который является узлом типа vertex_t.
- size - целое число, определяющее количество узлов в дереве.
- height - целое число, представляющее высоту дерева (длина самого длинного пути от корня до листа).

```
typedef struct node
{
    int value;               // Значение узла
    struct node *left;       // Левый потомок
    struct node *right;      // Правый потомок
} vertex_t;

// Структура дерева
typedef struct
{
    vertex_t *root;          // Корень дерева
    int size;                // Количество узлов
    int height;              // Высота дерева
} tree_t;
```

Основные функции программы

Программа реализует следующие основные функции:

- void action(int option, tree_t *tree); Функция меню:
    1. Прочитать дерево из файла
    2. Добавить элемент в дерево
    3. Удалить из дерева
    4. Найти в дереве
    5. Сохранить в формате dot
    6. Обход
    7. Отсортировать
    8. Протестировать

0. Выход

- void write_into_file(int value, char *filename); Записывает число в конец файла.
- tree_t *init_tree(); Создает и инициализирует пустое дерево. Возвращает указатель на структуру дерева.
- void free_tree(tree_t *tree); Освобождает память, выделенную под дерево, включая все его узлы.
- void add_node(tree_t *tree, char *filename); Добавляет узел в дерево с данными, прочитанными из файла.
- void delete_node(tree_t *tree); Удаляет узел из дерева.
- void read_tree(tree_t *tree, char *filename); Читает данные дерева из файла и заполняет дерево.
- void export_tree_to_dot(vertex_t *root, char *filename);
- Экспортирует дерево в файл формата DOT для визуализации с помощью Graphviz.
- void travel(tree_t *tree); Постфиксный обход дерева
- void find(vertex_t *root, int value, int steps); Ищет значение в дереве, начиная с указанного узла root. Также подсчитывает количество шагов, сделанных для поиска.
- void sort(tree_t *tree); Выводит элементы дерева в отсортированном виде

Описание основных алгоритмов

Алгоритм добавления узла.

- Если дерево пустое, создается корень.
- Если значение меньше текущего узла, рекурсивно добавляется в левое поддерево.
- Если значение больше текущего узла, рекурсивно добавляется в правое поддерево.

Алгоритм удаления узла.

- Если значение узла меньше текущего узла, то продолжается рекурсивный поиск в левом поддереве.
- Если значение узла больше текущего узла, то продолжается рекурсивный поиск в правом поддереве.

Если узел найден:

- Узел-лист удаляется.
- Узел с одним потомком заменяется на своего потомка.
- Узел с двумя потомками заменяется минимальным узлом из правого поддерева.

Алгоритм поиска узла.
Аналогичен алгоритму удаления, но без удаления.

Алгоритм сортировки узла.
Выполняется симметричный (in-order) обход дерева:

- Рекурсивный обход левого поддерева.
- Вывод значения текущего узла.
- Рекурсивный обход правого поддерева.

Набор тестов

Функция добавления

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Добавление узла в пустое дерево	tree = NULL, value = 10	Узел с value = 10 становится корнем дерева.
Добавление узла, меньшего корня	tree.root = 10, value = 5	Узел с value = 5 добавлен как левый потомок корня.
Добавление узла, большего корня	tree.root = 10, value = 15	Узел с value = 15 добавлен как правый потомок корня.
Добавление узла, меньшего узла уровня 2	Дерево: 10 → 5, value = 2	Узел с value = 2 добавлен как левый потомок узла 5.
Добавление узла, большего узла уровня 2	Дерево: 10 → 15, value = 20	Узел с value = 20 добавлен как правый потомок узла 15.

Функция удаления

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Удаление листа	Дерево: 10 → 5, удаляется value = 5	Узел с value = 5 удален, узел 10 остается корнем.
Удаление узла с одним потомком (слева)	Дерево: 10 → 5 → 3, удаляется value = 5	Узел с value = 5 удален, узел 3 становится левым потомком корня.
Удаление узла с одним потомком (справа)	Дерево: 10 → 15 → 20, удаляется value = 15	Узел с value = 15 удален, узел 20 становится правым потомком корня.
Удаление узла с двумя потомками	Дерево: 10 → 5, 15, удаляется value = 10	Узел с value = 10 заменяется минимальным элементом правого поддерева (15), узел 15 удален.
Удаление отсутствующего узла	Дерево: 10 → 5, 15, удаляется value = 8	Дерево остается неизменным.

Функция поиска

Тестовый сценарий	**Входные данные**	**Ожидаемый результат**
Поиск значения в корне	Дерево: 10 → 5, 15, value = 10	Узел с value = 10 найден за 1 шаг.
Поиск значения в левом поддереве	Дерево: 10 → 5, 15, value = 5	Узел с value = 5 найден за 2 шага.
Поиск значения в правом поддереве	Дерево: 10 → 5, 15, value = 15	Узел с value = 15 найден за 2 шага.
Поиск отсутствующего	Дерево: 10 → 5,	Узел с value = 8 не найден,

значения	15, value = 8	поиск завершен за 2 шага.
Поиск в пустом дереве	tree = NULL, value = 10	Узел с value = 10 не найден.

Аналитическая часть

Оборудование.
Все замеры происходили не ноутбуке с подключенным питанием и в режиме высокой производительности.
Характеристики ноутбука:

- ОС: Ubuntu LTS 22.04
- Оперативная память: 16 Гб DDR4
- Процессор: Intel core i5-12500H

Анализ памяти.
Один узел дерева (vertex_t) занимает 24 байта:

- int value: 4 байта.
- struct node *left: 8 байт.
- struct node *right: 8 байт.
- Выровнено до 24 байт

Структура tree_t занимает 16 байт:

- vertex_t *root: 8 байт.
- int size: 4 байта.
- int height: 4 байта

Добавление нового элемента в дерево увеличивает занимаемый размер памяти на 24 байта. Отсюда на хранение 1000 элементов уходит 24016 байт памяти.

Анализ времени.

В представленном тестировании оценивалась производительность бинарных деревьев двух типов: широкого и длинного. Были протестированы операции вставки, поиска, сортировки, удаления и записи данных. Ниже приведен анализ результатов.

Тест вставки

Тип дерева	Количество элементов	Время вставки (с)	Размер	Высота
Широкое	1000	0.006927	1000	19
	2500	0.077201	2496	27
	5000	0.481036	4987	31
	7500	1.205820	7469	32
	10000	2.224208	9961	32
Длинное	1000	0.024134	1000	1000
	2500	0.150251	2500	2500

	5000	0.605951	5000	5000
	7500	1.412496	7500	7500
	10000	2.552401	10000	10000

Широкое дерево демонстрирует значительно лучшие показатели по времени вставки, чем длинное дерево. С увеличением количества элементов время вставки в длинное дерево растет значительно быстрее, что связано с его высокой высотой (равной количеству элементов). В сбалансированном дереве высота растет медленно

Тест поиска

Тип дерева	Количество элементов	Среднее время поиска одного элемента (с)
	1000	0.028580
	2500	0.037060
	5000	0.042154
	7500	0.048243
Широкое	10000	0.048124
	1000	4.688610
	2500	11.537624
	5000	22.788590
	7500	34.448991
Длинное	10000	45.221980

Широкое дерево снова показывает превосходство в скорости поиска. Среднее время поиска элемента остается практически постоянным даже при увеличении числа элементов, благодаря меньшей высоте дерева. Длинное дерево, напротив, демонстрирует значительное увеличение времени поиска, поскольку требуется пройти большое количество узлов.

Тест сортировки

Тип дерева	Количество элементов	Время сортировки (с)
	1000	0.000004
	2500	0.000010
	5000	0.000041
	7500	0.000064
Широкое	10000	0.000100
	1000	0.000010
	2500	0.000025
Длинное	5000	0.000053

	7500	0.000076
	10000	0.000104

Операция сортировки оказалась крайне быстрой для обоих типов деревьев, поскольку сортировка представляет собой обход элементов в определенном порядке, который не зависит от структуры дерева.

Тест удаления

Тип дерева	Количество элементов	Время удаления (с)
Широкое	1000	0.000027
	2500	0.000083
	5000	0.000163
	7500	0.000278
	10000	0.000344
Длинное	1000	0.000368
	2500	0.000875
	5000	0.001765
	7500	0.002647
	10000	0.003600

Для удаления элементов широкое дерево вновь показало лучшие результаты. Время выполнения операций значительно меньше по сравнению с длинным деревом. Особо заметно замедление в длинном дереве при удалении элементов в обратном порядке, что подчеркивает его неэффективность в худших сценариях.

Тест скорости записи в файл

Количество элементов	Время записи в файл (с)
1000	0.020631
2500	0.048574
5000	0.102441
7500	0.159584
10000	0.225184

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». Память выделяется под каждую вершину дерева.

2. *Какие бывают типы деревьев?*

- Бинарное дерево (Binary Tree). Каждый узел имеет не более двух потомков — левого и правого.
- Двоичное дерево поиска (Binary Search Tree). Бинарное дерево, в котором для каждого узла выполнено правило: все узлы в левом поддереве меньше текущего узла, а все узлы в правом поддереве больше текущего узла.
- Красно-чёрное дерево (Red-Black Tree). Двоичное дерево поиска с дополнительными правилами балансировки.

3. *Какие стандартные операции возможны над деревьями?*

Добавление, удаление, поиск элемента, а также обход дерева.

4. *Что такое дерево двоичного поиска?*

Если у каждой вершины дерева имеется не более двух потомков (левые и правые поддеревья), то такое дерево называется двоичным или бинарным.

Заключение

Бинарное дерево остается универсальной структурой данных, но его эффективность напрямую зависит от высоты дерева, т.к. если случается так, что дерево вырождается в односвязный список, то все преимущества такого типа данных уходят на нет. Высокое дерево из-за линейной высоты демонстрирует существенные потери производительности.
Что касается записи в файл, то эта скорость мало изменяется в сравнении с деревом, а на небольшом количестве записываемых элементов и вовсе проигрывает бинарному дереву, хотя с увеличением количества скорость записи в бинарное дерево начинает проигрывать файлу.