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 1

Студент **Абдуллаев Ш. В.**

Группа **ИУ7-34Б**

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

Студент _____ **Абдуллаев Ш. В.**

Преподаватель _____**Силантьева А. В.**

2024

Описание условия задачи

Построить двоичное дерево из чисел файла. Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Определить количество узлов дерева на каждом уровне. Добавить число в дерево и в файл. Сравнить время добавления чисел в указанные структуры.

Описание исходных данных

Программа представляет собой консольное приложение для работы с бинарным деревом поиска. Основная функциональность заключается в следующих операциях:

- Вывод дерева: дерево можно отобразить как графическую структуру в консоли.
- Поиск элементов: пользователь вводит число для поиска. Программа выводит результат поиска.
- Добавление элементов: пользователь вводит число для добавления. Если число уже существует, программа уведомляет об этом.
- Удаление элементов: пользователь вводит число для удаления. Если число отсутствует, программа уведомляет, что удаление невозможно.
- Обходы дерева: постордерный (последовательный), инордерный (симметричный) и преордерный (прямой) обходы.

Имеются следующие ограничения:

Ожидаемый формат данных: только целые числа. Ввод некорректных данных (например, букв или символов) завершает программу с сообщением об ошибке.

Работа с пустым деревом: некоторые операции (поиск, удаление, обходы) недоступны, если дерево пустое.

Чтение данных из файла: при повторной попытке прочитать дерево из файла программа уведомляет, что данные уже были загружены.

Допустимый ввод	Недопустимый ввод
Выберите команду: 1	Выберите команду: q
Введите число для поиска: 42	Введите число для поиска: abc
Введите число для добавления: 15	Введите число для поиска: 15abc
Введите число для удаления: -8	Введите число для поиска: 7x

Таблица 1. Примеры ввода

Описание результатов

1. Выйти из программы: завершение работы программы, освобождение памяти, если необходимо.
2. Считать двоичное дерево из чисел файла: чтение данных из указанного файла и построение на их основе двоичного дерева. Если дерево уже существует, повторное считывание невозможно
3. Вывести двоичное дерево в виде графа: отображение текущего состояния дерева в текстовом графическом представлении, показывающем иерархическую структуру узлов.
4. Проверить наличие числа в дереве: поиск заданного пользователем числа в двоичном дереве. Если число найдено, выводится соответствующее сообщение.
5. Добавить число в дерево: добавление нового числа в двоичное дерево. Если число уже существует, выводится сообщение о невозможности добавления.
6. Удалить число из дерева: удаление указанного числа из дерева. Если число не найдено, пользователю сообщается о невозможности выполнения операции.
7. Вывести дерево в обратном (постордерном) обходе: последовательный вывод всех узлов дерева в порядке "левое поддерево → правое поддерево → корень".
8. Вывести дерево в симметричном (инордерном) обходе: последовательный вывод всех узлов дерева в порядке "левое поддерево → корень → правое поддерево".
9. Вывести дерево в прямом (преордерном) обходе: последовательный вывод всех узлов дерева в порядке "корень → левое поддерево → правое поддерево".

10. Определить количество узлов дерева на каждом уровне: вычисление и вывод количества узлов дерева для каждого уровня его глубины.
11. Добавить число в файл: добавление нового числа в файл с данными. Если операция прошла успешно, выводится соответствующее сообщение.
12. Сравнить время добавления чисел в дерево и файл: измерение времени выполнения операций добавления чисел в двоичное дерево и в файл, с последующим сравнением этих значений.

Описание задачи, реализуемой программой

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

Способ обращения к программе

Обращения к программе пользователем происходит с помощью вызова исполняемого файла (app.exe).

Описание возможных аварийных ситуаций и ошибок пользователя

- NO_DATA_ERROR: ошибка отсутствия файла данных. Возникает, если в программу не передан путь к файлу при запуске.
- INPUT_COMMAND_ERR: ошибка ввода команды. Возникает, если пользователь вводит некорректное значение при выборе команды из меню.
- INPUT_FOR_FIND_ERR: ошибка ввода числа для поиска. Возникает, если пользователь ввел некорректное значение при попытке найти число в дереве.
- INPUT_FOR_INSERT_ERR: ошибка ввода числа для добавления в дерево. Возникает, если пользователь вводит некорректное значение при добавлении нового узла.
- INPUT_FOR_REMOVE_ERR: ошибка ввода числа для удаления из дерева. Возникает, если пользователь ввел некорректное значение при удалении узла.

- INPUT_FOR_APPEND_TO_FILE_ERR: ошибка ввода числа для записи в файл. Возникает, если пользователь вводит некорректное значение при попытке добавить новое число в файл.

Описание внутренних структур данных

Программа содержит структуру, которая используется для хранения узлов бинарного дерева, представленную в Листинге 1.

```
typedef struct Node
{
    int value;
    struct Node *left;
    struct Node *right;
} Node;
```

Листинг 1. Структура Node

Рассмотрим каждое поле структуры:

1. value: целочисленное поле типа int, которое хранит значение текущего узла дерева. Это значение представляет данные, связанные с узлом.
2. left: указатель на структуру Node, указывающий на левое поддерево текущего узла. Если левое поддерево отсутствует, это поле принимает значение NULL.
3. right: указатель на структуру Node, указывающий на правое поддерево текущего узла. Если правое поддерево отсутствует, это поле принимает значение NULL.

Описание алгоритма

1. Инициализация системы и выделение памяти.
2. Основной цикл программы начинается. Пользователю предоставляется меню с различными опциями, и программа ожидает ввода выбора пользователя.
3. В зависимости от выбора пользователя, программа выполняет следующие действия:
    - Выйти из программы

- Считать двоичное дерево из чисел файла
- Вывести двоичное дерево в виде графа
- Проверить наличие числа в дерев

  Алгоритм функции реализует рекурсивный поиск заданного значения в двоичном дереве поиска. Если текущий узел пустой, возвращается 0 (значение не найдено). Если значение текущего узла совпадает с искомым, возвращается 1 (значение найдено). Если искомое значение меньше текущего, продолжается поиск в левом поддереве, иначе — в правом.
- Добавить число в дерево

  Алгоритм функции выполняет рекурсивное добавление нового значения в двоичное дерево поиска. Если текущий узел пустой, создается новый узел с заданным значением и возвращается как результат. Если значение больше текущего узла, функция рекурсивно вызывается для правого поддерева; если меньше — для левого. После завершения вставки возвращается корень дерева, сохраняя его структуру.
- Удалить число из дерева

  Алгоритм функции удаляет заданное значение из двоичного дерева поиска. Если дерево пустое, функция устанавливает флаг found в 0 и возвращает NULL. В противном случае, значение рекурсивно ищется в левом или правом поддереве. Если значение найдено, устанавливается found = 1, и выполняется удаление узла. Если узел имеет один дочерний элемент, возвращается его поддерево. Если оба поддерева существуют, значение заменяется минимальным элементом правого поддерева, после чего этот элемент удаляется. Функция возвращает корень дерева после изменения.
- Вывести дерево в обратном (постордерном) обходе

  Алгоритм функции выполняет обход дерева в обратном (постордерном) порядке. Сначала рекурсивно обрабатывается левое поддерево, затем правое поддерево, и только после этого выводится значение текущего узла. Этот порядок обхода используется, например, для освобождения памяти

или удаления узлов в дереве, поскольку сначала обрабатываются дочерние узлы, а затем сам узел.

- Вывести дерево в симметричном (инордерном) обходе

  Алгоритм функции выполняет обход дерева в симметричном (инордерном) порядке. Сначала рекурсивно обрабатывается левое поддерево, затем выводится значение текущего узла, после чего рекурсивно обрабатывается правое поддерево. Это позволяет вывести значения узлов дерева в порядке возрастания для бинарного поиска, поскольку в таком дереве все значения в левом поддереве меньше значения корня, а все значения в правом поддереве — больше.

- Вывести дерево в прямом (преордерном) обходе

  Алгоритм функции выполняет обход дерева в преордерном порядке (сначала посещается корень, затем левое поддерево, затем правое поддерево). Если текущий узел не пустой, выводится его значение. Затем рекурсивно вызываются функции обхода для левого и правого поддеревьев. Такой подход обеспечивает, что корень каждого поддерева будет выведен перед его дочерними узлами.

- Определить количество узлов дерева на каждом уровне
- Добавить число в файл
- Сравнить время добавления чисел в дерево и файл

4. После выполнения каждой операции программа возвращает пользователя в главное меню, где он может выбрать следующее действие.

Набор тестов

Описание	**Результат**
Вывести двоичное дерево в виде графа	| | | /12 | | /20 | /15 | | \12 ^10

	| | /7 | \5 | | \3 | | | \1 | | | | \0
Добавить элемент в дерево Элемент: 3	Успешное добавление элемента в дерево Добавленный элемент: 3
Удалить элемент из дерева Элемент: 5	Успешное удаление элемента в дерево Удаленный элемент: 5
Проверить наличие числа в дереве Элемент: 123	Число 123 найдено в дереве
Вывести дерево в обратном (постордерном) обходе	Дерево в постордерном обходе: 10 5 3 1 0 7 15 12 20 123
Вывести дерево в симметричном (центрированном, инордерном) обходе	Дерево в инордерном обходе: 0 1 3 5 7 10 12 15 20 123
Вывести дерево в прямом (преордерном) обходе	Дерево в преордерном обходе: 0 1 3 7 5 12 123 20 15 10
Определить количество узлов дерева на каждом уровне	Количество узлов на каждом уровне: Уровень 0: 1 узлов Уровень 1: 2 узлов Уровень 2: 4 узлов Уровень 3: 2 узлов Уровень 4: 1 узлов

Таблица 2. Набор позитивных тестов

Описание	**Результат**
Попытка не передавать датасет	Возврат ошибки NO_DATA_ERROR

Попытка ввести символы вместо команды	Возврат ошибки INPUT_COMMAND_ERR
Попытка ввести не целое число при поиске элемента	Возврат ошибки INPUT_FOR_FIND_ERR
Попытка ввести не целое число при добавлении элемента	Возврат ошибки INPUT_FOR_INSERT_ERR
Попытка ввести не целое число при удалении элемента	Возврат ошибки INPUT_FOR_REMOVE_ERR
Попытка ввести не целое число при добавлении элемента в файл	Возврат ошибки INPUT_FOR_APPEND_TO_FILE_ERR

Таблица 3. Набор негативных тестов

Оценка эффективности

При запуске программы N = 100 раз, были получены следующие данные:

Количество уровней	**Время в дерево (нс)**	**Время в файл (нс)**	**Выигрыш дерева относительно файла (в разах)**
2	5076.52	350279.88	69
3	8344.12	776003.16	93
5	11528.19	1200261.46	104
7	35652.23	4354227.72	122
9	116167.50	31300219.73	269
10	230622.33	101130637.57	439
		Среднее:	183

Таблица 4. Сравнение эффективности бинарного дерева и файла

Программа, использующая бинарное дерево, демонстрирует высокую эффективность по сравнению с операциями над файлом для представленных объемов данных. Использование бинарного дерева для хранения и поиска

данных позволяет эффективно обрабатывать запросы и операции в сравнении с операциями ввода/вывода через файл.

Выводы

В ходе выполнения лабораторной работы была разработана программа для работы с бинарным деревом поиска, предоставляющая пользователю удобный интерфейс для взаимодействия с данными.

В результате тестирования эффективности программы на различных объемах данных, были получены временные показатели для операций с бинарным деревом и операциями с файлом. Согласно результатам тестов, программа, использующая бинарное дерево, демонстрирует высокую временную эффективность. Эффективность программы, подтверждает, что использование бинарного дерева поиска позволяет существенно ускорить операции по сравнению с файлом.

В целом, разработанная программа успешно решает поставленную задачу и предоставляет эффективные средства для работы с бинарным деревом поиска.

Контрольные вопросы

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

Дерево - структура данных, состоящая из узлов, связанных между собой рёбрами. Один из узлов называется корнем, остальные разделяются на узлы и листья. Узлы, соединенные ребрами, образуют поддеревья. Память под представление деревьев обычно выделяется динамически. Каждый узел дерева содержит информацию и указатели на своих потомков (или нулевые указатели, если потомков нет). Для каждого узла память выделяется отдельно при добавлении новых узлов.

2. Какие бывают типы деревьев?

Дерево двоичное: Каждый узел имеет не более двух потомков.

Дерево двоичного поиска: Узлы упорядочены так, что для каждого узла все узлы в его левом поддереве меньше его, а в правом — больше.

N-арное дерево: Каждый узел может иметь произвольное количество потомков. Распределенное дерево: используется в распределенных вычислениях и сетевых структурах.

AVL-дерево, красно-черное дерево: Сбалансированные бинарные деревья для эффективного поиска.

3. Какие стандартные операции возможны над деревьями?

Добавление узла: Вставка нового узла в дерево.

Удаление узла: Удаление существующего узла из дерева.

Поиск узла: Нахождение узла с определенным значением.

Обход дерева: Посещение всех узлов дерева в определенном порядке (прямой, обратный, симметричный).

Вывод дерева в виде строки: Представление дерева в текстовой или графической форме.

Изменение данных узла: Обновление значений в существующем узле.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска - бинарное дерево, в котором каждый узел имеет не более двух потомков. При этом для каждого узла выполнено следующее свойство: все узлы в левом поддереве меньше текущего узла, а все узлы в правом поддереве больше текущего узла. Это свойство делает дерево двоичного поиска эффективной структурой данных для поиска, вставки и удаления элементов, так как оно обеспечивает логарифмическую сложность этих операций в среднем случае.