|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 «Деревья» (ВАР. 1)**

Студент Смирнов Иван Владимирович

Группа ИУ7 – 32Б

Проверил Барышникова Марина Юрьевна

*2023 г.*

Оглавление

[Описание условия задачи 2](#_Toc151148746)

[Описание внутренних структур данных 5](#_Toc151148747)

[Описание алгоритмов 5](#_Toc151148748)

[Сравнение времени работы и ресурсных затрат 6](#_Toc151148749)

[Тесты 12](#_Toc151148750)

[Ответы на контрольные вопросы 15](#_Toc151148751)

[Вывод 16](#_Toc151148752)

# Описание условия задачи

Программа работает с типом данных «дерево», представленном в виде «бинарного дерева», выполняет определенные действия по указанию пользователя. Программа может построить дерево как от ручных добавлений/удалений элементов пользователя, так из файла с целыми числами. В программе предусмотрен вывод дерева как на экран терминала, так и в виде дерева с помощью утилиты graphviz. В программе реализованы функции подсчета количества узлов дерева на каждом уровне. Можно добавить число в дерево и в файл. При выполнении таких операций, как чтение дерева из файла, добавление/удаление/поиск элемента, программа выводит среднее время выполнения данного действия.   
Описание технического задания

1. **Исходные данные и результат**

**Входные данные:**

Пользователь выбирает пункты меню с соответствующими действиями.

|  |
| --- |
| Меню:  ---  1. Вывести дерево (graphviz)  2. Вывести дерево (терминал)  ---  3. Считать дерево (из файла)  4. Включить узел (в дерево и файл)(сравнить время добавления)  5. Вывести количество узлов на всех уровнях дерева  ---  6. Включить узел (в дерево)  7. Исключить узел (из дерева)  8. Поиск узла  ---  9. Обход дерева (pre-order)  10. Обход дерева (in-order)  11. Обход дерева (post-order)  ---  0. Выход  Введите ваш выбор: |

**Выходные данные:**

На экран выводятся результаты команд меню.

1. **Задачи, реализуемой программой**

**Программа умеет выводить дерево с помощью утилиты** graphviz и на экран консоли с соблюдением отступов. Программа умеет считывать дерево из файла с целыми числами; включать узел как в дерево, так и в файл; выводить количество узлов на всех уровнях дерева. В программе реализованы функции удаления/поиска элемента, а также функции обхода дерева (pre-order, in-order, post-order).

**Способ обращения к программе**

**Команда для сборки проекта: make release  
Программа запускается через терминал командой: ./app.exe**

1. **Возможные аварийные ситуации и ошибки пользователя**
   1. **Попытка ввода не целого числа в виде выбора команды**

**На выходе сообщение: «Пожалуйста, введите целое число!»**

* 1. **Попытка ввести несуществующий номер команды**

**На выходе сообщение: «Неверный выбор.»**

* 1. **Попытка вывести пустое дерево**

**На выходе сообщение: «Дерево пусто!»**

* 1. **Попытка считать дерево из несуществующего файла**

**На выходе сообщение: «Такого файла не существует!»**

* 1. **Попытка считать дерево из файла, в котором повторяются элементы**

**На выходе сообщение: «В файле найдены дубликаты вершин!»**

* 1. **Попытка добавить не целое число в дерево**

**На выходе сообщение: «Пожалуйста, введите целое число!»**

* 1. **Попытка добавить узел в дерево и в файл, заранее не указав файл**

**Автоматически выберется файл files/default.txt и значение добавится туда.**

* 1. **Попытка добавить в дерево уже существующий элемент**

**На выходе сообщение: «Элемент нельзя добавить, так как он уже существует!»**

* 1. **Попытка удалить несуществующий элемент**

**На выходе сообщение: «Узел не найден!»**

* 1. **Попытка найти несуществующий элемент**

**На выходе сообщение: «Узел не найден!»**

# **Описание внутренних структур данных**

Информация о дереве (в виде узлов):

|  |
| --- |
| // Структура для узла дерева  typedef struct Node {  int data;  struct Node\* left;  struct Node\* right;  } node\_t; |

node\_t – структура для хранения узла дерева, где data – значение узла; left – указатель на левый узел; right – указатель на правый узел.

# Описание алгоритмов

1. Пользователь вводит пункт меню как целое число.
2. При выборе способа вывода дерева, пользователь может выбрать, вывести на экран или через утилиту graphviz. Если выбор пользователя – утилита graphviz, то картинка с деревом автоматически открывается.
3. При чтении дерева из файла, программа заполняет дерево на основе данных из файла. Название файла запоминается программой, чтобы в дальнейшем можно было добавить вершину как в файл, так и в дерево.
4. Элемент добавляется в дерево на основе правила: «меньше – влево, больше – вправо».
5. Элемент удаляется из дерева на основе правила: Если у узла нет потомков, просто удаляем этот узел. Если у узла есть только один потомок, заменяем узел его потомком. Если у узла есть два потомка, находим наименьший элемент в правом поддереве, заменяем удаляемый узел этим элементом, а затем удаляем этот элемент из правого поддерева.
6. При подсчете количества узлов в дереве выводится также информация о том, сколько байт занимает дерево в оперативной памяти компьютера.
7. В программе можно выбрать один из трех вариантов обхода дерева (pre-order, in-order, post-order).
8. При выборе пункта 0 – программа завершается.

# Сравнение времени работы и ресурсных затрат

В программе предусмотрена функция, которая замеряет время (в мкс.) определенного процесса 100 раз и в качестве результата выдает среднее из 100 замеров. При выборе пунктов «3» – Считать дерево из файла; «4» – Включить узел в дерево и в файл; «8» – Поиск узла; программа дополнительно с выбранным действием выводит время его выполнения. При выборе пункта «4» выводится время как добавления элемента в дерево, так и добавление в файл. Результаты для следующего дерева из файла «files/1.txt» приведены ниже:

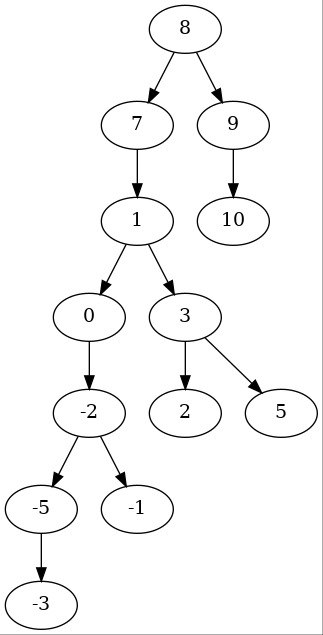


Рисунок 1. Изначальное дерево

|  |
| --- |
| … Введите название файла:  files/1.txt  Старое дерево уничтожено  Дерево создалось из данных из файла <files/1.txt> за: 14.730000мкс.  …  Введите ваш выбор: 4  Введите значение для включения: -6  Запись произойдет в файл <files/1.txt>  Число -6 добавлено в дерево и записано в файл files/1.txt.  Добавление элемента <-6> в дерево: 0.550000мкс.  Добавление элемента <-6> в файл: 1.020000мкс.  Эффективность добавления в дерево относительно добавления в файл: 46.078431%  …  Введите ваш выбор: 4  Введите значение для включения: 11  Запись произойдет в файл <files/1.txt>  Число 11 добавлено в дерево и записано в файл files/1.txt.  Добавление элемента <11> в дерево: 0.520000мкс.  Добавление элемента <11> в файл: 1.090000мкс.  Эффективность добавления в дерево относительно добавления в файл: 52.293578%  …  Введите ваш выбор: 5  Уровень 0: 1 узлов  Уровень 1: 2 узлов  Уровень 2: 2 узлов  Уровень 3: 3 узлов  Уровень 4: 3 узлов  Уровень 5: 2 узлов  Уровень 6: 2 узлов  Дерево занимает 360Б  …  Введите ваш выбор: 8  Введите значение для исключения (оно должно быть в дереве): -6  Узел присутствует в дереве!  Поиск элемента <-6> в дереве: 0.250000мкс.  …  Введите ваш выбор: 8  Введите значение для поиска (оно должно быть в дереве): 8  Узел присутствует в дереве!  Поиск элемента <8> в дереве: 0.200000мкс. |

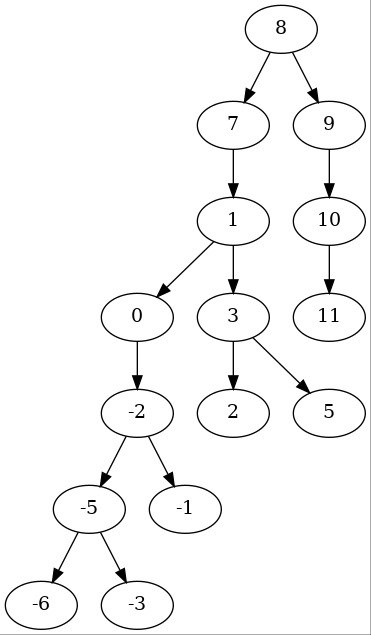


Рисунок 2. Измененное дерево

Из замерного эксперимента видно, что не только добавление элемента в дерево работает почти в 2 раза быстрее, чем добавление в файл, но и в среднем создание дерева вручную быстрее создания дерева через файл (0.55\*13=7.15<14.73, где 13 – число вершин в изначальном дереве). Значит дерево эффективнее создавать в программе, не используя сторонние файлы, однако необходимо придумать изначальный способ хранения значений узлов. По памяти измененное дерево, состоящее из 15 узлов, суммарно весит 360Б, значит каждый узел весит по 24Б. Поиск элемента дерева, находящегося на 0-ом уровне составило 0.20мкс., а на 6-ом уровне заняло всего 0.25мкс. Проанализируем, за сколько создаются деревья из файла, в котором 13 чисел изначально отсортированы (в прямом/в обратном порядке):

|  |
| --- |
| Введите ваш выбор: 3  Введите название файла:  files/right.txt  Старое дерево уничтожено  Считанное дерево:  13  12  11  10  9  8  7  6  5  4  3  2  1  Дерево создалось из данных из файла <files/right.txt> за: 7.720000мкс.  …  Введите название файла:  files/reverse.txt  Старое дерево уничтожено  Считанное дерево:  13  12  11  10  9  8  7  6  5  4  3  2  1  Дерево создалось из данных из файла <files/reverse.txt> за: 7.040000мкс. |

Как видно, времена примерно равны и оба примерно в 2 раза меньше, чем время, потраченное на создание дерева на основе файла «files/1.txt».

Деревья являются удобной структурой данных во многих случаях, особенно когда необходимо хранить и организовывать данные и выполнять операции поиска, вставки и удаления с высокой эффективностью. Однако необходимо обратить на несколько моментов при работе с деревьями:

* Обработка пустого дерева: Программа должна корректно обрабатывать ситуации, когда дерево пустое, чтобы избежать ошибок.
* Балансировка дерева: В случае использования сбалансированных деревьев (не в этой л/р), необходимо протестировать процессы вставки, удаления и балансировки, чтобы удостовериться в их корректной работе.
* Обработка дубликатов: Если дерево не должно содержать дубликатов (как в нашем случае), необходимо протестировать соответствующие проверки и операции вставки.

# Тесты

В следующих тестах символ «>» обозначает ввод пользователем пункта меню, а символ Х – полученное значение в ходе программы.

1. **Позитивные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| pos\_01 | >1 | Составленная модель дерева с помощью утилиты “graphviz” | Умеет ли программа выводить дерево с помощью “graphviz” |
| pos\_02 | >2 | (выведенное в терминал дерево) | Умеет ли программа выводить дерево на экран |
| pos\_03 | >3  (название существующего файла с целыми числами) | Старое дерево уничтожено  Считанное дерево:  (выведенное в терминал дерево)  Дерево создалось из данных из файла <название файла> за: **Х** мкс. | Умеет ли программа считывать дерево по данным из файла и замерять время данной операции |
| pos\_04 | >4  -6 | Запись произойдет в файл <название файла>  Число -6 добавлено в дерево и записано в файл <название файла>.  Добавление элемента <-6> в дерево: **Х** мкс.  Добавление элемента <-6> в файл: **Х** мкс.  Эффективность добавления в дерево относительно добавления в файл: **Х** % | Умеет ли программа добавлять элемент как в дерево, так и в файл и замерять время данных операций |
| pos\_05 | >5 | Уровень 0: **Х** узлов  Уровень 1: **Х** узлов  Уровень 2: **Х** узлов  ...  Уровень n: **Х** узлов  Дерево занимает **Х** Б | Умеет ли программа считать количество узлов на разных уровнях |
| pos\_06 | >6  (несуществующий элемент) | ... | Умеет ли программа добавлять узел только в само дерево |
| pos\_07 | >7  (существующий элемент) | ... | Умеет ли программа удалять узел из дерева |
| pos\_08 | >8  (существующий элемент) | Узел присутствует в дереве!  Поиск элемента <**Х**> в дереве: **Х** мкс. | Умеет ли программа находить узел в дереве и замерять данный процесс |
| pos\_09 | >9 | Pre-order обход: <> | Умеет ли программа обходить дерево |
| pos\_10 | >10 | In-order обход: <> | Умеет ли программа обходить дерево |
| pos\_11 | >11 | Post-order обход: <> | Умеет ли программа обходить дерево |

1. **Негативные тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| Номер теста | Входные данные | Выходные данные | Что проверяется |
| neg\_01 |  | Сообщение: «Пожалуйста, введите целое число!» | Пустой ввод |
| neg\_02 | >-1 | Сообщение: «Неверный выбор.» | Правильность ввода пункта меню |
| neg\_03 | >100 | Сообщение: «Неверный выбор.» | Правильность ввода пункта меню |
| neg\_04 | >А | Сообщение: «Пожалуйста, введите целое число!» | Правильность ввода пункта меню |
| neg\_05 | >1  (длина дерева 0) | Сообщение: «Дерево пусто!» | Существование дерева |
| neg\_06 | >2  (длина дерева 0) | Сообщение: «Дерево пусто!» | Существование дерева |
| neg\_07 | >3  (несуществующий файл) | Сообщение:  «Такого файла не существует!» | Существование файла |
| neg\_08 | >3  (файл с дубликатами) | Сообщение:  «В файле найдены дубликаты вершин!» | Попытка добавить элемент, который уже есть в дереве |
| neg\_09 | >4  (существующий элемент) | Сообщение:  «Элемент нельзя добавить, так как он уже существует!» | Попытка добавить элемент, который уже есть в дереве |
| neg\_10 | >6  (существующий элемент) | Сообщение:  «Элемент нельзя добавить, так как он уже существует!» | Попытка добавить элемент, который уже есть в дереве |
| neg\_11 | >7  (не существующий элемент) | Сообщение:  «Узел не найден!» | Попытка удалить несуществующий элемент |
| neg\_12 | >8  (не существующий элемент) | Сообщение:  «Узел не найден!» | Попытка найти несуществующий элемент |
| neg\_13 | >4  (не целое число) | Сообщение: «Пожалуйста, введите целое число!» | Корректность ввода элемента |
| neg\_14 | >6  (не целое число) | Сообщение: «Пожалуйста, введите целое число!» | Корректность ввода элемента |
| neg\_15 | >7  (не целое число) | Сообщение: «Пожалуйста, введите целое число!» | Корректность ввода элемента |
| neg\_16 | >8  (не целое число) | Сообщение: «Пожалуйста, введите целое число!» | Корректность ввода элемента |

# Ответы на контрольные вопросы

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

Дерево - это абстрактная структура данных, состоящая из узлов, соединенных ребрами. Узлы могут содержать данные, и каждый узел имеет ноль или более дочерних узлов. Память для представления деревьев выделяется динамически в зависимости от количества узлов и их связей.

1. **Какие бывают типы деревьев?**

Двоичное дерево: каждый узел имеет не более двух дочерних узлов.

Дерево поиска: особый тип двоичного дерева, где для каждого узла все значения в левом поддереве меньше, чем значение узла, а все значения в правом поддереве больше, чем значение узла.

Красно-черное дерево: сбалансированное двоичное дерево, где каждый узел имеет дополнительный бит цвета.

B-дерево: сбалансированное дерево, используемое в базах данных и файловых системах.

1. **Какие стандартные операции возможны над деревьями?**

Стандартные операции:

* Вставка узла
* Удаление узла
* Поиск узла (в глубину, в ширину)
* Обход дерева (pre-order; in-order; post-order)
* Получение высоты дерева
* Балансировка дерева

1. **Что такое дерево двоичного поиска?**

Дерево двоичного поиска (Binary Search Tree, BST) - это особый тип двоичного дерева, где для каждого узла все значения в левом поддереве меньше, чем значение узла, а все значения в правом поддереве больше, чем значение узла. BST обеспечивает эффективный поиск, вставку и удаление элементов, так как они поддерживают упорядоченную структуру данных.

# Вывод

В рамках задания был реализован тип данных «дерево двоичного поиска». Реализовано 2 способа вывода дерева (graphviz и терминал), 2 способа создания дерева (через данные из файла и ручным добавлением/удалением элементов), 3 способа обхода дерева (pre-order; in-order; post-order), функция подсчета количества узлов на каждом уровне и объем занимаемой деревом памяти.