|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования C++»  Свой вариант | | |
|  | | Работу выполнил  студент группы ИТ-11-2024-2025 1 курса  Токсаров Артемий В.  «25» июня 2025 г. |
| Работу проверил  Кнутова Н.С.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование 3](#_Toc153130029)

[Код программы 3](#_Toc153130030)

[Инструкция по применению стилей и оформлению работы 4](#_Toc153130031)

# Постановка задачи

# Дано бинарное дерево. Лесник может отрезать ветки (узлы), чтобы сделать дерево симметричным. Задача. Найти минимальное количество отрезаемых веток (узлов), чтобы дерево стало зеркально симметричным. Вывести количество отрезанных лесником листьев (количество удалённых узлов). Условия: Дерево считается симметричным, если левое и правое поддеревья зеркально идентичны. Каждый удалённый узел считается как 1 единица работы лесника.

# Алгоритм решения

**Строение программы**

1. **Структуры данных**:

- Узел дерева (TreeNode) содержит значение и указатели на потомков

- Класс MakeTree для построения деревьев

- Класс TreeOperations для работы с деревьями

2. **Основные методы**:

- Рекурсивная генерация случайного дерева (randTree)

- Построение BST из ввода/файла (buildTreeFromInput/buildTreeFromFile)

- Симметризация дерева (makeTreeSymmetric и makeSymmetric)

- Визуализация дерева (printTreeVertical)

3. **Особенности реализации**:

- Вероятностный алгоритм генерации (70% chance создания узла)

- Рекурсивные алгоритмы обхода и модификации

- Вертикальный вывод дерева

**Алгоритм makeTreeSymmetric и makeSymmetric:**

**Функция makeTreeSymmetric**

Делает всё дерево симметричным, начиная с корня. На вход получает root (указатель на корень дерева).

1. Проверка на пустое дерево: если root == nullptr, возвращаем 0 (дерево уже симметрично).
2. Обработка корня с двумя поддеревьями:
   * Если у корня есть оба поддерева (root->left и root->right):

Вызываем makeSymmetric(root->left, root->right), чтобы сделать их зеркальными.

* + - Результат (количество удалённых узлов) записываем в cuts.

1. Обработка корня с одним поддеревом:
   * Если есть только левое поддерево (root->left):
     + Считаем количество узлов в нём (countNodes(root->left)).
     + Удаляем его (deleteTree(root->left)).
     + Обнуляем указатель (root->left = nullptr).
   * Если есть только правое поддерево (root->right):
     + Аналогично удаляем его и записываем количество удалённых узлов.
2. Возврат результата: возвращаем общее количество удалённых узлов (cuts).

**Функция makeSymmetric**

Рекурсивно делает два поддерева зеркально симметричными, подсчитывая количество удалённых узлов. На вход получает указатели a и b на левое и правое поддеревья.

1. Проверка на пустые поддеревья:

Если оба узла a и b пусты (nullptr), возвращаем 0 (ничего удалять не нужно).

1. Обработка отсутствия одного из поддеревьев:

Если a отсутствует (nullptr):

* + - Считаем количество узлов в поддереве b (countNodes(b)).
    - Удаляем всё поддерево b (deleteTree(b)).
    - Обнуляем указатель b = nullptr.
    - Возвращаем количество удалённых узлов (cuts).

Если b отсутствует (nullptr):

* + - Аналогично считаем и удаляем поддерево a.

1. Рекурсивная обработка зеркальных поддеревьев:

Вызываем makeSymmetric(a->left, b->right) (левое поддерево a сравнивается с правым поддеревом b).

Вызываем makeSymmetric(a->right, b->left) (правое поддерево a сравнивается с левым поддеревом b).

Суммируем количество удалённых узлов (cuts += ...).

1. Возврат результата: возвращаем общее количество удалённых узлов (cuts).

**Методы, использованные в работе программы**

**Класс**MakeTree**(построение деревьев)**

randTree(int maxDepth, int currentDepth = 0):

Рекурсивно генерирует случайное бинарное дерево заданной максимальной глубины.

С вероятностью 30% прекращает ветвление на каждом уровне.

checkInput(string input) -Проверяет, является ли строка числом.

Insert(TreeNode\* node, int value) -Вставляет новый узел с заданным значением в дерево поиска.

buildTreeFromInput() -Строит дерево поиска из чисел, введённых пользователем (до команды Q).

buildTreeFromFile(string filename) -Строит дерево поиска из чисел из файла.

**Класс**TreeOperations**(операции с деревьями)**

countNodes(TreeNode\* node) -Рекурсивно подсчитывает количество узлов в поддереве.

cloneTree(TreeNode\* node) -Создаёт копию дерева.

deleteTree(TreeNode\* node) -Рекурсивно удаляет все узлы дерева (освобождает память).

Height(TreeNode\* root) -Вычисляет высоту дерева.

makeSymmetric(TreeNode\*& a, TreeNode\*& b) -Рекурсивно делает два поддерева симметричными, возвращая количество удалённых узлов.

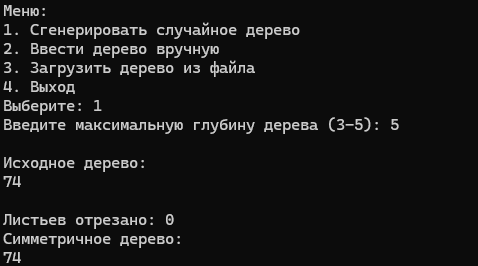
makeTreeSymmetric(TreeNode\* root)- Делает всё дерево симметричным. Возвращает общее количество удалённых узлов.

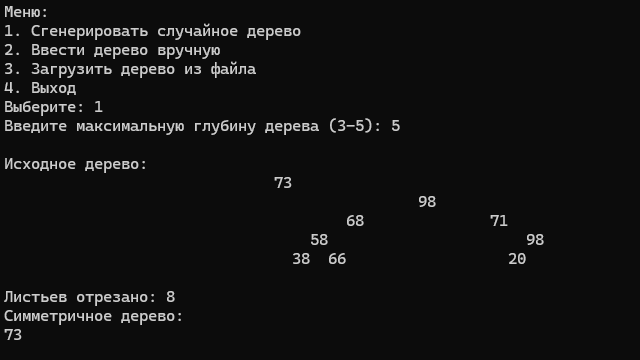
printTreeVertical(TreeNode\* root)- Выводит дерево в вертикально.

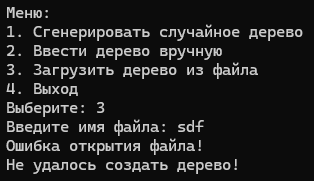
**Дополнительные функции**

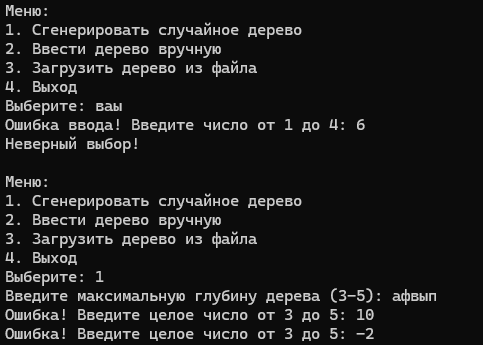
displayMenu()-Выводит текстовое меню с вариантами действий

# Тестирование









# Код программы

Ссылка на git: https://github.com/ToksarovArtemiy/C-Tree