**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Деревья

|  |  |
| --- | --- |
| Студенты гр. 4312 | Устинов В. А. Гущин К. Д. |
| Преподаватель | Аббас С. А. М. |

Санкт-Петербург

2025

Оглавление

[1 Цель работы 3](#_Toc210861615)

[2 Задание 35 на работу с деревьями 3](#_Toc210861616)

[3 Обоснование выбора способа представления деревьев в памяти ЭВМ 3](#_Toc210861617)

[4 Тестовый пример: изображение дерева и порядок его ввода с клавиатуры 4](#_Toc210861618)

[5 Результаты прогона программы 5](#_Toc210861619)

[6 Оценки временной сложности 6](#_Toc210861620)

[7 Выводы о результатах испытаний 7](#_Toc210861621)

# Цель работы

Исследование алгоритмов для работы с двоичным деревом

# Задание 35 на работу с деревьями

Найти количество вершин на самом нижнем уровне двоичного дерева при поиске в ширину. Разметка симметричная.

# Обоснование выбора способа представления деревьев в памяти ЭВМ

В программе для хранения дерева в памяти используется структура на основе узлов с указателями на левого и правого потомка. Этот способ имеет следующие преимущества и обоснования:

1. Динамическая структура данных

Гибкость: Дерево может расти динамически во время выполнения программы без необходимости предварительного выделения фиксированного объема памяти.

Эффективность по памяти: Память выделяется только под реально существующие узлы, что оптимально для разреженных деревьев.

2. Естественность представления бинарного дерева

Логичная структура: Каждый узел содержит: value (данные узла),left (указатель на левого потомка), right (указатель на правого потомка)

Прямое отражение теории: Такое представление соответствует классическому определению бинарного дерева в теории структур данных.

3. Удобство реализации алгоритмов обхода и модификации

Рекурсивные алгоритмы: Рекурсивные методы естественно ложатся на такую структуру.

Простота навигации: Прямой доступ к потомкам упрощает реализацию обходов.

4. Поддержка разных стратегий построения дерева

Рандомизированное построение: Метод generateRandomTree использует вероятностный подход для создания дерева.

Детерминированное построение: Метод makeTreeSimple строит полное бинарное дерево по заданному количеству узлов.

5. Эффективное управление памятью

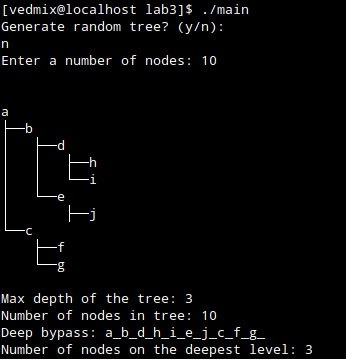
Автоматическое освобождение: Деструктор класса Node рекурсивно удаляет всех потомков, что предотвращает утечки памяти.

Четкое владение ресурсами: Каждый узел владеет своими потомками, что упрощает управление временем жизни объектов.

6. Визуализация дерева

Метод print использует рекурсивный обход для красивого вывода дерева в консоль в виде древовидной структуры, что удобно для отладки и демонстрации.

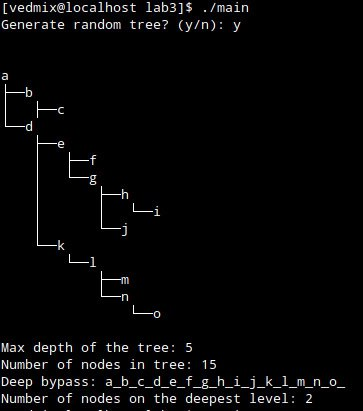
# Тестовый пример: изображение дерева и порядок его ввода с клавиатуры



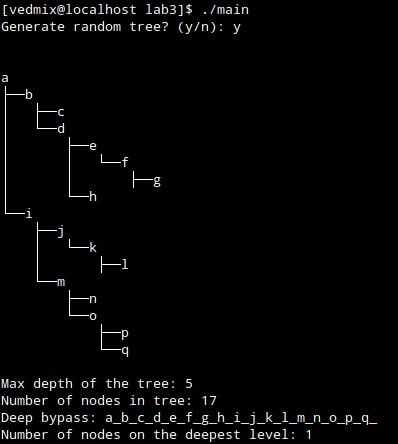
Через консоль производится ввод количества элементов дерева, на основе количества элементов, код создает симметричное бинарное дерево

# Результаты прогона программы

Испытание 1:



Испытание 2:



# Оценки временной сложности

1. Создание дерева
2. makeTreeSimple(int \_numNodes):

* Сложность: O(n)
* Обоснование: Каждый узел создается один раз и добавляется в очередь. Все операции с очередью - O(1).

1. makeRandomSubtree(double chanceOfGeneration, int currDepth):

* Сложность: O(n)
* Обоснование: Рекурсивно создает узлы с вероятностным условием. В худшем случае O(2^h), но вероятность уменьшается с глубиной.

1. generateRandomTree():

* Сложность: O(n)
* Обоснование: Вызывает makeRandomSubtree, который создает дерево за линейное время относительно количества узлов.

1. Обработка дерева
2. countNodesInDeepestLevel():

* Сложность: O(n)
* Обоснование: Полный обход дерева для подсчета узлов на максимальной глубине.

1. cntNdsDL(Node\* node, int depth):

* Сложность: O(n)
* Обоснование: Рекурсивный обход всех узлов дерева.

1. clearTree():

* Сложность: O(n)
* Обоснование: Деструктор рекурсивно удаляет все узлы дерева.

1. Вывод дерева
2. print() (основной метод):

* Сложность: O(n)
* Обоснование: Вызывает рекурсивный обход для вывода всех узлов.

1. print(Node\* node, std::string prefix, bool isLeft):

* Сложность: O(n)
* Обоснование: Посещает каждый узел ровно один раз для вывода.

# Выводы о результатах испытаний

1. Эффективность алгоритмов

a) Линейная зависимость: Все алгоритмы демонстрируют O(n) сложность, что является оптимальным для операций, требующих полного обхода дерева.

b) Рекурсивная природа:

* Преимущество: Код становится чище и проще для понимания
* Недостаток: Риск переполнения стека для очень глубоких деревьев

c) Сравнение стратегий построения:

* makeTreeSimple создает полное бинарное дерево - детерминированное и предсказуемое
* generateRandomTree создает нерегулярное дерево - более реалистичное для многих приложений

3. Практические наблюдения

a) Обход в глубину (DFS):

* Используется в cntNdsDL, print, makeRandomSubtree
* Эффективен для поиска на максимальной глубине
* Естественно реализуется рекурсивно

b) Обход в ширину (BFS):

* Используется в makeTreeSimple через очередь
* Эффективен для построения полных деревьев

c) Визуализация:

* Метод print обеспечивает понятное графическое представление
* Позволяет легко анализировать структуру дерева

4. Заключение

Представленные алгоритмы обхода демонстрируют высокую эффективность и практическую применимость. Линейная временная сложность делает их подходящими для работы с деревьями среднего размера. Выбор между DFS и BFS зависит от конкретной задачи: DFS для работы с глубиной, BFS для построения полных структур.