Тема 05: Деревья. Бинарные деревья поиска

Цель работы: Изучить древовидные структуры данных, их свойства и применение. Освоить основные операции с бинарными деревьями поиска (BST). Получить практические навыки реализации BST на основе узлов (pointer-based), рекурсивных алгоритмов обхода и анализа их эффективности. Исследовать влияние сбалансированности дерева на производительность операций.

Теория (кратко):

- **Дерево:** Рекурсивная структура данных, состоящая из узлов, где каждый узел имеет значение и ссылки на дочерние узлы.
- **Бинарное дерево поиска (BST):** Дерево, для которого выполняются следующие условия:
  - Значение в левом поддереве любого узла **меньше** значения в самом узле.
  - Значение в правом поддереве любого узла **больше** значения в самом узле.
  - Оба поддерева являются бинарными деревьями поиска.
- **Основные операции BST:**
  - **Вставка (Insert):** Сложность: в среднем O(log n), в худшем (вырожденное дерево) O(n).
  - **Поиск (Search):** Сложность: в среднем O(log n), в худшем O(n).
  - **Удаление (Delete):** Сложность: в среднем O(log n), в худшем O(n). Имеет три случая: удаление листа, узла с одним потомком, узла с двумя потомками.
  - **Обход (Traversal):**
    - **In-order (левый-корень-правый):** Посещает узлы в порядке возрастания. Сложность O(n).
    - **Pre-order (корень-левый-правый):** Полезен для копирования структуры дерева. Сложность O(n).
    - **Post-order (левый-правый-корень):** Полезен для удаления дерева. Сложность O(n).
- **Сбалансированные деревья:** Деревья с контролем высоты (например, AVL, Красно-черные), которые гарантируют время операций O(log n) даже в худшем случае.

Практика (подробно):

Задание:

1. Реализовать бинарное дерево поиска на основе узлов с основными операциями.
2. Реализовать различные методы обхода дерева (рекурсивные и итеративные).
3. Реализовать дополнительные методы для работы с BST.
4. Провести анализ сложности операций для сбалансированного и вырожденного деревьев.
5. Визуализировать структуру дерева.

Шаги выполнения:

1. **Создание проекта:** Создать файлы `binary_search_tree.py`, `tree_traversal.py`, `analysis.py`.
2. **Реализация BST (в `binary_search_tree.py`):**
   - Реализовать класс `TreeNode`.
   - Реализовать класс `BinarySearchTree` с методами:
     - `insert(value)`
     - `search(value)`
     - `delete(value)`

        - `find_min(node)` (поиск минимума в поддереве)
        - `find_max(node)` (поиск максимума в поддереве)
    - **После каждого метода указать его временную сложность в худшем и среднем случае.**
3. **Реализация обходов (в `tree_traversal.py`):**
    - Реализовать рекурсивные версии обходов (in-order, pre-order, post-order) для печати элементов.
    - Реализовать итеративную версию in-order обхода с использованием стека.
4. **Реализация дополнительных методов (в `binary_search_tree.py`):**
    - `is_valid_bst()`: Проверка, является ли дерево корректным BST.
    - `height(node)`: Вычисление высоты дерева/поддерева.
5. **Тестирование:**
    - Написать unit-тесты для проверки корректности работы всех операций.
    - Проверить свойства BST после вставки и удаления.
6. **Экспериментальное исследование (в `analysis.py`):**
    - Построить сбалансированное дерево (вставляя элементы в случайном порядке) и вырожденное (вставляя элементы в отсортированном порядке).
    - Замерить время выполнения 1000 операций поиска в деревьях разного размера и разной степени сбалансированности.
    - **ВАЖНО: Все замеры проводить на одной вычислительной машине.**
7. **Визуализация:**
    - Реализовать простую текстовую визуализацию дерева (вывод в виде отступов или скобочной последовательности).
    - Построить графики зависимости времени операций от количества элементов для сбалансированного и вырожденного случаев.
8. **Анализ результатов:**
    - Сравнить практическую и теоретическую сложность операций.
    - Проанализировать влияние структуры дерева на производительность.
9. **Оформление отчета:** Результаты оформить в файле `README.md`. Код должен соответствовать PEP8.
10. **Контроль версий:** Стратегия ветвления – GitHub Flow.

Критерии оценки:

- **Оценка «3» (удовлетворительно):**
    - Реализовано BST с операциями вставки, поиска и рекурсивным in-order обходом.
    - В коде присутствуют комментарии с оценкой сложности.
    - Проведены базовые замеры времени поиска для двух конфигураций дерева.
- **Оценка «4» (хорошо):**
    - Выполнены все критерии на «3».
    - Реализованы удаление элемента и все 3 рекурсивных обхода.
    - Реализована проверка `is_valid_bst`.
    - Код хорошо отформатирован и полностью прокомментирован.
    - Проведены замеры для деревьев разного размера.
    - Построены графики зависимости времени от количества элементов.

- **Оценка «5» (отлично):**
    - Выполнены все критерии на «4».
    - Приведены характеристики ПК для тестирования.
    - Реализован итеративный in-order обход.
    - Реализованы методы `find_min`, `find_max`, `height`.
    - Проведен полный сравнительный анализ производительности для сбалансированного и вырожденного случаев.
    - Реализована текстовая визуализация дерева.
    - В отчете присутствует детальный анализ с выводами о важности балансировки.

Рекомендованная литература

1. **Юрий Петров: "Программирование на Python"** — онлайн-курс и учебные материалы.
    - *Ссылка для изучения:* https://www.yuripetrov.ru/edu/python/index.html
2. **Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.** Алгоритмы: построение и анализ, 3-е издание. — М.: Вильямс, 2022. — 1328 с.
    - *(Оригинальное название: Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. Introduction to Algorithms, 3rd Edition)*
3. **Скиена, С.** Алгоритмы. Руководство по разработке, 3-е издание. — СПб.: БХВ-Петербург, 2022. — 720 с.
    - *(Оригинальное название: Skiena, Steven S. The Algorithm Design Manual, 3rd ed.)*