Тема 02: Основные структуры данных. Анализ и применение

Цель работы: Изучить понятие и особенности базовых абстрактных типов данных (стек, очередь, дек, связный список) и их реализаций в Python. Научиться выбирать оптимальную структуру данных для решения конкретной задачи, основываясь на анализе теоретической и практической сложности операций. Получить навыки измерения производительности и применения структур данных для решения практических задач.

Теория (кратко):

- **Список (list) в Python:** Реализация динамического массива. Обеспечивает амортизированное время O(1) для добавления в конец (`append`). Вставка и удаление в середину имеют сложность O(n) из-за сдвига элементов. Доступ по индексу - O(1).
- **Связный список (Linked List):** Абстрактная структура данных, состоящая из узлов, где каждый узел содержит данные и ссылку на следующий элемент. Вставка и удаление в известное место (например, начало списка) выполняются за O(1). Доступ по индексу и поиск - O(n).
- **Стек (Stack):** Абстрактный тип данных, работающий по принципу LIFO (Last-In-First-Out). Основные операции: `push` (добавление, O(1)), `pop` (удаление с вершины, O(1)), `peek` (просмотр вершины, O(1)). В Python может быть реализован на основе списка.
- **Очередь (Queue):** Абстрактный тип данных, работающий по принципу FIFO (First-In-First-Out). Основные операции: `enqueue` (добавление в конец, O(1)), `dequeue` (удаление из начала, O(1)). В Python для эффективной реализации используется `collections.deque`.
- **Дек (Deque, двусторонняя очередь):** Абстрактный тип данных, позволяющий добавлять и удалять элементы как в начало, так и в конец. Все основные операции - O(1). В Python реализован в классе `collections.deque`.

Практика (подробно):

Задание:

1. Реализовать класс `LinkedList` (связный список) для демонстрации принципов его работы.
2. Используя встроенные типы данных (`list`, `collections.deque`), проанализировать эффективность операций, имитирующих поведение стека, очереди и дека.
3. Провести сравнительный анализ производительности операций для разных структур данных (`list` vs `LinkedList` для вставки, `list` vs `deque` для очереди).
4. Решить 2-3 практические задачи, выбрав оптимальную структуру данных.

Шаги выполнения:

1. **Создание проекта:** Создать файлы `linked_list.py`, `performance_analysis.py`, `task_solutions.py`.
2. **Реализация связного списка:**
    - Реализовать класс `Node` и класс `LinkedList`.
    - Реализовать методы: `insert_at_start` (O(1)), `insert_at_end` (O(n) или O(1) с хвостом), `delete_from_start` (O(1)), `traversal` (O(n)).
    - **После каждого метода в комментарии указать его асимптотическую сложность.**
3. **Анализ производительности (на основе встроенных структур):**
    - Использовать модуль `timeit` для замеров времени.

- **Сравнение `list` и гипотетического `LinkedList` для операций вставки:**
  - Замерить время 1000 вставок в начало структуры. Для `list` (`insert(0, item)`) это будет O(n) на каждую операцию, для `LinkedList` (`insert_at_start`) - O(1). Продемонстрировать кардинальную разницу.
- **Сравнение `list` и `deque` для реализации очереди:**
  - Замерить время 1000 операций `dequeue` (удаление из начала). Для `list` (`pop(0)`) это O(n), для `deque` (`popleft()`) - O(1). Продемонстрировать разницу.
- **ВАЖНО: Все замеры проводить на одной вычислительной машине.**

4. **Решение задач:**
   - Реализовать проверку сбалансированности скобок (`{[()]}`) с использованием стека (реализованного на `list`).
   - Реализовать симуляцию обработки задач в очереди печати (использовать `deque`).
   - Решить задачу "Палиндром" (проверка, является ли последовательность палиндромом) с использованием дека (`deque`).
5. **Визуализация:** Построить графики зависимости времени выполнения операций от количества элементов, наглядно демонстрирующие разницу в асимптотике.
6. **Оформление отчета:** Результаты оформить в файле `README.md`. Код должен соответствовать PEP8.
7. **Контроль версий:** Стратегия ветвления - GitHub Flow.

Критерии оценки:

- **Оценка «3» (удовлетворительно):**
  - Реализован класс `LinkedList` с методами вставки/удаления в начало.
  - Проведены базовые замеры производительности для `list` (вставка в начало).
  - Решена 1 практическая задача (скобки).
  - В коде присутствуют комментарии с оценкой сложности для ключевых операций.
- **Оценка «4» (хорошо):**
  - Выполнены все критерии на «3».
  - Реализован полноценный `LinkedList` с добавлением в конец (с поддержкой tail-указателя).
  - Проведено сравнение `list` и `deque` для операций очереди (`pop(0)` vs `popleft()`).
  - Код хорошо отформатирован, читаем и полностью прокомментирован.
  - Решены 2 практические задачи (скобки + палиндром/очередь).
  - Построены простые графики на основе замеров.
- **Оценка «5» (отлично):**
  - Выполнены все критерии на «4».
  - Приведены характеристики ПК для тестирования.
  - Проведен полный сравнительный анализ: `list` vs `LinkedList` для вставки в начало/конец, `list` vs `deque` для очереди.
  - Графики наглядно демонстрируют разницу в асимптотической сложности (линейный vs постоянный рост).
  - В отчете присутствует детальный анализ результатов, объясняющий, почему полученные графики времени подтверждают теоретическую сложность.

- Решены все 3 практические задачи с обоснованием выбора структуры данных для каждой.

Рекомендованная литература

1. Документация Python: `collections.deque`, `timeit`.
2. **Юрий Петров: "Программирование на Python"** — онлайн-курс и учебные материалы.
   - *Ссылка для изучения:* https://www.yuripetrov.ru/edu/python/index.html
3. **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)*
4. **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.)*