Тема 03: Рекурсия

Цель работы: Освоить принцип рекурсии, научиться анализировать рекурсивные алгоритмы и понимать механизм работы стека вызовов. Изучить типичные задачи, решаемые рекурсивно, и освоить технику мемоизации для оптимизации рекурсивных алгоритмов. Получить практические навыки реализации и отладки рекурсивных функций.

Теория (кратко):

- **Рекурсия:** Процесс, при котором функция прямо или косвенно вызывает саму себя для решения задачи.
- **Базовый случай (условие выхода):** Обязательное условие, которое прекращает рекурсивные вызовы и предотвращает зацикливание.
- **Рекурсивный шаг:** Шаг, на котором задача разбивается на более простую подзадачу того же типа и производится рекурсивный вызов.
- **Глубина рекурсии:** Количество вложенных вызовов функции. Ограничена размером стека вызовов.
- **Стек вызовов (Call Stack):** Структура данных, которая хранит информацию о незавершенных вызовах функций (локальные переменные, адрес возврата).
- **Мемоизация (Memoization):** Техника оптимизации, позволяющая избежать повторных вычислений результатов функций для одних и тех же входных данных путем сохранения ранее вычисленных результатов в кеше (например, в словаре).

Практика (подробно):

Задание:

1. Реализовать классические рекурсивные алгоритмы.
2. Проанализировать их временную сложность и глубину рекурсии.
3. Реализовать оптимизацию рекурсивных алгоритмов с помощью мемоизации.
4. Сравнить производительность наивной рекурсии и рекурсии с мемоизацией.
5. Решить практические задачи с применением рекурсии.

Шаги выполнения:

1. **Создание проекта:** Создать файлы `recursion.py`, `memoization.py`, `recursion_tasks.py`.
2. **Реализация рекурсивных алгоритмов (в `recursion.py`):**
    - Вычисление факториала числа `n`.
    - Вычисление `n`-го числа Фибоначчи.
    - Быстрое возведение числа `a` в степень `n` (через степень двойки).
    - **После каждой функции в комментарии указать её временную сложность и глубину рекурсии.**
3. **Оптимизация с помощью мемоизации (в `memoization.py`):**
    - Реализовать мемоизированную версию функции для чисел Фибоначчи.
    - Сравнить количество рекурсивных вызовов и время работы наивной и мемоизированной версии для `n=35`.
4. **Решение практических задач (в `recursion_tasks.py`):**
    - Реализовать алгоритм бинарного поиска с использованием рекурсии.

    - Реализовать рекурсивный обход файловой системы (вывод дерева каталогов и файлов, начиная с заданного пути).
    - Решить задачу "Ханойские башни" для `n` дисков.
5. **Экспериментальное исследование:**
    - Замерить время выполнения наивного и мемоизированного вычисления чисел Фибоначчи для разных `n`.
    - Измерить максимальную глубину рекурсии для обхода файловой системы на глубоко вложенной структуре каталогов.
    - **ВАЖНО: Все замеры проводить на одной вычислительной машине.**
6. **Визуализация:**
    - Построить график времени выполнения рекурсивного вычисления Фибоначчи с мемоизацией и без.
    - Для задачи "Ханойские башни" вывести на экран последовательность перемещений дисков.
7. **Анализ результатов:**
    - Объяснить экспоненциальный рост времени выполнения наивного алгоритма Фибоначчи.
    - Проанализировать, как мемоизация меняет сложность алгоритма.
8. **Оформление отчета:** Результаты оформить в файле `README.md`. Код должен соответствовать PEP8.
9. **Контроль версий:** Стратегия ветвления – GitHub Flow.

Контрольные вопросы

1. Что такое базовый случай и рекурсивный шаг в рекурсивной функции? Почему отсутствие базового случая приводит к ошибке?
2. Объясните, как работает механизм мемоизации. Как он меняет временную сложность вычисления чисел Фибоначчи по сравнению с наивной рекурсией?
3. В чем заключается основная проблема глубокой рекурсии и как она связана со стеком вызовов?
4. Задача о Ханойских башнях решается рекурсивно. Опишите алгоритм решения для 3 дисков.
5. Рекурсивный и итеративный алгоритмы могут решать одни и те же задачи. Назовите преимущества и недостатки каждого подхода.

Критерии оценки:

- **Оценка «3» (удовлетворительно):**
    - Реализованы 3 рекурсивные функции (факториал, Фибоначчи, быстрая степень).
    - В коде присутствуют комментарии с оценкой сложности.
    - Проведены базовые замеры времени для наивного вычисления Фибоначчи.
- **Оценка «4» (хорошо):**
    - Выполнены все критерии на «3».
    - Реализована мемоизация для чисел Фибоначчи.
    - Решены 2 практические задачи (бинарный поиск и Ханойские башни).
    - Код хорошо отформатирован и полностью прокомментирован.
    - Построен график сравнения наивного и мемоизированного подхода.
- **Оценка «5» (отлично):**
    - Выполнены все критерии на «4».

- Приведены характеристики ПК для тестирования.
- Реализована задача рекурсивного обхода файловой системы.
- Проведен полный анализ изменения сложности алгоритмов с применением мемоизации.
- В отчете присутствует детальный анализ результатов, включая объяснение работы стека вызовов и ограничений рекурсии.
- Решены все 3 практические задачи.

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.)*