Отчет по лабораторной работе №3

Тема: Рекурсия

Дата: 2025-10-16
Семестр: 3 курс, 5 семестр
Группа: ПИЖ-б-о-23-2(2) **Дисциплина:** Анализ сложности алгоритмов
Студент: Виктор Блинов Александрович

Цель работы

Изучение методов рекурсивного решения задач, включая факториал, числа Фибоначчи, бинарный поиск, Ханойские башни, а также использование мемоизации для оптимизации рекурсивных алгоритмов.

Теоретическая часть

Рекурсия — это метод программирования, при котором функция вызывает сама себя. Рекурсивные алгоритмы удобно использовать для решения задач с повторяющейся структурой. Однако наивная рекурсия может быть медленной и потреблять много памяти из-за глубокого стека вызовов. Мемоизация позволяет хранить промежуточные результаты и значительно ускоряет вычисления.

Классические примеры рекурсивных задач:

- Факториал числа
- Числа Фибоначчи
- Бинарный поиск в отсортированном массиве
- Задача Ханойских башен
- Обход файловой системы

Практическая часть

Выполненные задачи

- ☑ Задача 1: Реализован рекурсивный расчет факториала
- ☑ Задача 2: Реализован наивный и мемоизированный расчет чисел Фибоначчи
- ☑ Задача 3: Реализован рекурсивный бинарный поиск и Ханойские башни
- ☑ Задача 4: Проведен рекурсивный обход файловой системы и измерена максимальная глубина рекурсии
- ☑ Задача 5: Сравнена производительность наивной и мемоизированной рекурсии с построением графика

Ключевые фрагменты кода

```python
# Рекурсивный факториал
def factorial(n: int) -> int:
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

# Наивная рекурсия Фибоначчи
def fibonacci(n: int) -> int:
    if n <= 1:
        return n
    return fibonacci(n - 1) + fibonacci(n - 2)

# Мемоизированная Фибоначчи
def fib_memo(n: int, cache=None) -> int:
    if cache is None:
        cache = {}
    if n in cache:
        return cache[n]
    if n <= 1:
        cache[n] = n
    else:
        cache[n] = fib_memo(n - 1, cache) + fib_memo(n - 2, cache)
    return cache[n]

# Рекурсивный бинарный поиск
def binary_search_recursive(arr, target, left=0, right=None):
    if right is None:
        right = len(arr) - 1
    if left > right:
        return -1
    mid = (left + right) // 2
    if arr[mid] == target:
        return mid
    elif arr[mid] > target:
        return binary_search_recursive(arr, target, left, mid - 1)
    else:
        return binary_search_recursive(arr, target, mid + 1, right)

# Ханойские башни
def hanoi_tower(n, source, target, auxiliary):
    if n == 1:
        print(f"Переместить диск 1 с {source} → {target}")
        return
    hanoi_tower(n - 1, source, auxiliary, target)
    print(f"Переместить диск {n} с {source} → {target}")
    hanoi_tower(n - 1, auxiliary, target, source)
```

Результаты выполнения

Пример работы программы

```
Бинарный поиск:
Элемент 7 имеет индекс: 3

Ханойские башни (3 диска):
Переместить диск 1 с A → C
Переместить диск 2 с A → B
Переместить диск 1 с C → B
Переместить диск 3 с A → C
Переместить диск 1 с B → A
Переместить диск 2 с B → C
Переместить диск 1 с A → C

Обход файловой системы:
|-- main.py
|-- recursion_tasks.py
|-- memoization.py
|-- ... (другие файлы и папки)

Максимальная глубина рекурсии: 1

Сравнение времени выполнения рекурсии с мемоизацией и без:
Наивная рекурсия (35): 0.9569 сек
С мемоизацией (35): 0.0001 сек

График сохранён в файл: fibonacci_plot.png
```

Характеристики ПК для тестирования

```
Процессор: 12th Gen Intel(R) Core(TM) i5-12450H 2.00 GHz
Оперативная память: 16 GB DDR4
ОС: Windows 11 Pro
Python: 3.10.10
```

Выводы

1. Рекурсия удобна для решения задач с повторяющейся структурой, но может быть ресурсоёмкой.
2. Мемоизация значительно ускоряет вычисления, особенно для экспоненциальных рекурсий, таких как числа Фибоначчи.
3. Рекурсивные алгоритмы позволяют наглядно решать задачи бинарного поиска, Ханойских башен и обхода файловой системы, с измерением глубины рекурсии.

Ответы на контрольные вопросы

1. **Что такое базовый случай и рекурсивный шаг в рекурсивной функции? Почему отсутствие базового случая приводит к ошибке?** - Базовый случай определяет условие, при котором рекурсия завершится, а рекурсивный шаг — это правило, по которому функция вызывает сама

себя с уменьшенным или изменённым аргументом. Отсутствие базового случая приводит к бесконечной рекурсии и ошибке переполнения стека.

2. **Объясните, как работает механизм мемоизации. Как он меняет временную сложность вычисления чисел Фибоначчи по сравнению с наивной рекурсией?** - Мемоизация сохраняет ранее вычисленные значения в кэше, чтобы повторно их не вычислять. Для чисел Фибоначчи это снижает временную сложность с O(2^n) до O(n).

3. **В чем заключается основная проблема глубокой рекурсии и как она связана со стеком вызовов?** - Основная проблема глубокой рекурсии заключается в возможном переполнении стека вызовов (stack overflow). Каждый рекурсивный вызов сохраняет локальные переменные и адрес возврата в стеке, при глубокой рекурсии стек может исчерпать память.

4. **Задача о Ханойских башнях решается рекурсивно. Опишите алгоритм решения для 3 дисков.** - Перемещаем два верхних диска на вспомогательную башню, затем перемещаем самый большой диск на целевую башню, после чего два верхних диска с вспомогательной башни перемещаем на целевую башню.

5. **Рекурсивный и итеративный алгоритмы могут решать одни и те же задачи. Назовите преимущества и недостатки каждого подхода.** - Рекурсивный: преимущества — лаконичный и наглядный код; недостатки — большая нагрузка на стек, медленнее при больших данных. Итеративный: преимущества — экономичнее по памяти и быстрее; недостатки — код может быть более сложным и менее читаемым.

Приложения

- Исходный код recursion.py
- Исходный код recursion_tasks.py
- Исходный код memoization.py