Лабораторная работа 1.

Вариант 1. Кролики и рекуррентные отношения

**Цель:** Разработать программу, вычисляющую численность популяции кроликов на заданный месяц n на основе рекуррентного отношения, учитывающего коэффициент плодовитости k.

**Задача:** Реализовать алгоритм, принимающий на вход натуральные числа n (месяц) и k (количество пар кроликов в помёте), и возвращающий общее число кроличьих пар на n-ый месяц согласно модифицированной модели Фибоначчи.

**Инструменты:**

• **Язык программирования:** Python (подходит для реализации рекурсивных и итеративных алгоритмов, имеет встроенную поддержку больших чисел).

• **Среда разработки (IDE)**, т. е. VS code

**Алгоритмы:**

1. **Рекурсивный алгоритм (с мемоизацией):**

Определение базовых случаев:

F(1) = 1 (одна пара кроликов в первый месяц)

F(2) = 1 (одна пара кроликов во второй месяц)

Рекурсивное определение:

\* F(n) = F(n-1) + k \* F(n-2) (число пар кроликов в n-ый месяц равно числу пар в предыдущем месяце плюс количество новых пар, которое равно k умноженному на число пар два месяца назад).

Мемоизация: Для оптимизации рекурсивного решения можно использовать мемоизацию (сохранение результатов вычислений для повторного использования). Это позволяет избежать повторных вычислений одних и тех же значений и значительно ускоряет работу алгоритма.

2. **Итеративный алгоритм (динамическое программирование):**

Создание массива (списка) для хранения значений F(i) от 1 до n.

Инициализация первых двух элементов массива: F[0] = 1, F[1] = 1.

Итеративное вычисление:

Для i от 2 до n-1: F[i] = F[i-1] + k \* F[i-2].

Возвращение F[n-1].

**Ошибки и способы их предотвращения:**

**Переполнение стека при рекурсивном решении (без мемоизации):** при больших значениях n рекурсивный алгоритм может привести к переполнению стека из-за большого количества вложенных вызовов функций.

**Решение:** Использовать итеративный алгоритм или рекурсивный алгоритм с мемоизацией.

**Выводы:**

1. Задача о кроликах Фибоначчи является классическим примером, демонстрирующим концепцию рекуррентных отношений и динамического программирования.
2. Итеративные алгоритмы, как правило, более эффективны, чем рекурсивные (без мемоизации), особенно при больших значениях n.