Лабораторні роботи №4-5 «Програмування на Python за допомогою штучного інтелекту ChatGPT»

1. Мета роботи:

Оволодіти базовими методами генерації алгоритмів за допомогою ChatGPT.

Освоїти принципи формування алгоритмів обробки ланих.

Ткаченко Костянтин Олександрович ІП-з31

Використаний **ШІ** – Groq

Виконані завдання - Відстань Левенштейна, Алгоритм Евкліда, Числа Фібоначчі

Код:

```
from openai import OpenAI

GROQ_API_KEY = "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"
MODEL = "llama-3.3-70b-versatile"


def create_groq_client():
    return OpenAI(
        api_key=GROQ_API_KEY,
        base_url="https://api.groq.com/openai/v1"
    )


def ask_groq(client, system_prompt, user_prompt, temperature=0.6,
max_tokens=1500):
    try:
        response = client.chat.completions.create(
            model=MODEL,
            messages=[
                {"role": "system", "content": system_prompt},
                {"role": "user", "content": user_prompt}
            ],
            temperature=temperature,
            max_tokens=max_tokens
        )
        return response.choices[0].message.content.strip()
    except Exception as e:
        print("\nпомилка при запиті")
        print(f"деталі: {e}")

        error_str = str(e).lower()
        if "authentication" in error_str:
            print("ключ проблема")
        elif "rate limit" in error_str or "429" in error_str:
            print("вичерпано ліміт ")
        elif "network" in error_str or "connection" in error_str:
            print("проблеми з інтеренотом")
        else:
            print("невідома помилка")

        return None


def print_separator(title):
```

```
    print("\n" + "=" * 70)
    print(f" {title} ")
    print("=" * 70 + "\n")


def main():
    client = create_groq_client()

    # Відстань Левенштейна
    print_separator("Відстань Левенштейна")
    levenshtein_prompt = (
        "Напиши функцію на Python для обчислення відстані Левенштейна
(Levenshtein distance) "
        "між двома рядками за допомогою динамічного програмування. "
        "Функція повинна повертати тільки число — відстань. "
        "Додай 3 приклади тестування з print."
    )
    result = ask_groq(client, "Ти експерт з алгоритмів на Python. Пиши
чистий, ефективний код.", levenshtein_prompt)
    if result:
        print(result)

    # Алгоритм Евкліда
    print_separator("Алгоритм Евкліда")
    euclid_prompt = (
        "Напиши на Python функцію для обчислення НСД (GCD) за алгоритмом
Евкліда "
        "(iterative версія з while). "
        "Додай 3-4 приклади тестування з виводом, дай саме відповіді у
кінці"

    )
    result = ask_groq(client, "Ти експерт з алгоритмів. Пиши чистий код на
Python.", euclid_prompt)
    if result:
        print(result)

    # Фібоначчі
    print_separator(" Фібоначчі ")
    fib_prompt_simple = (
        "Напиши на Python функцію для обчислення n-го числа Фібоначчі з
мемоїзацією.\n"
        "F(0) = 0, F(1) = 1\n"
        "Додай коментарі українською.\n"
        "Виведи приклади: F(0), F(5), F(10), F(20), дай саме відповіді у
кінці"
    )
    result = ask_groq(client, "Ти експерт з алгоритмів на Python. Пиши
чистий код з коментарями українською.",
                      fib_prompt_simple)
    if result:
        print(result)

    print("\n" + "=" * 70)
    print("РОБОТА ЗАВЕРШЕНА")
    print("=" * 70)


if __name__ == "__main__":
    main()
```

Відстань Левенштейна

```
========================================================================
 Відстань Левенштейна
========================================================================

Відстань Левенштейна - це кількість операцій (вставок, видалень, заміни), необхідних для перетворення одного рядка в інший. Нижче наведено фун

```python
def levenshtein_distance(s1, s2):
    """
    Обчислює відстань Левенштейна між двома рядками.

    :param s1: Перший рядок
    :param s2: Другий рядок
    :return: Відстань Левенштейна між двома рядками
    """
    m, n = len(s1), len(s2)
    dp = [[0] * (n + 1) for _ in range(m + 1)]

    # Ініціалізація таблиці
    for i in range(m + 1):
        dp[i][0] = i
    for j in range(n + 1):
        dp[0][j] = j

    # Заповнення таблиці
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            cost = 0 if s1[i - 1] == s2[j - 1] else 1
            dp[i][j] = min(dp[i - 1][j] + 1,      # Видалення
                           dp[i][j - 1] + 1,      # Вставка
                           dp[i - 1][j - 1] + cost)  # Заміна

    return dp[m][n]

# Тестування
print(levenshtein_distance("kitten", "sitting"))  # 3
print(levenshtein_distance("hello", "hello"))  # 0
print(levenshtein_distance("abc", "def"))  # 3
```

Ця функція використовує динамічне програмування для побудови таблиці `dp`, де `dp[i][j]` представляє відстань Левенштейна між першими `i` симв
```

Алгоритм Евкліда

```
========================================================================
 Алгоритм Евкліда
========================================================================

```python
def gcd(a, b):
    """
    Обчислює найбільш спільний дільник (НСД) двох чисел за допомогою ітеративної версії алгоритму Евкліда.

    :param a: Перше число
    :param b: Друге число
    :return: Найбільший спільний дільник чисел a і b
    """
    while b != 0:
        a, b = b, a % b
    return abs(a)

# Тестування функції
print("НСД чисел 48 і 18:", gcd(48, 18))
print("НСД чисел 54 і 24:", gcd(54, 24))
print("НСД чисел 17 і 5:", gcd(17, 5))
print("НСД чисел 120 і 90:", gcd(120, 90))
```

Відповіді:
6
6
1
```

Фібоначчі

```
================================================================================
  Фібоначчі
================================================================================

```python
# Створюємо словник для зберігання результатів обчислень
memo = {0: 0, 1: 1}

# Функція для обчислення n-го числа Фібоначчі з мемоїзацією
def fibonacci(n):
    """
    Обчислює n-е число Фібоначчі з мемоїзацією.

    Аргументи:
    n (int): Номер числа Фібоначчі для обчислення.

    Повертає:
    int: n-е число Фібоначчі.
    """
    # Якщо результат вже обчислений, повертаємо його з словника
    if n in memo:
        return memo[n]
    # Якщо результат ще не обчислений, обчислюємо його і зберігаємо у словнику
    else:
        # Обчислюємо n-е число Фібоначчі як суму двох попередніх чисел
        result = fibonacci(n-1) + fibonacci(n-2)
        # Зберігаємо результат у словнику для майбутнього використання
        memo[n] = result
        return result

# Приклади використання функції
print(f"F(0) = {fibonacci(0)}")
print(f"F(5) = {fibonacci(5)}")
print(f"F(10) = {fibonacci(10)}")
print(f"F(20) = {fibonacci(20)}")

# Відповіді:
# F(0) = 0
# F(5) = 5
# F(10) = 55
# F(20) = 6765
```
```