Bài 1:

from collections import deque

#tìm giá trị lớn nhất trong mỗi cửa sổ:

def sliding\_window\_max(num\_list, k):

    q = deque()  # Deque lưu các chỉ số

    result = []

    for i, num in enumerate(num\_list):

        # Loại bỏ phần tử không còn trong cửa sổ

        if q and q[0] < i - k + 1:

            q.popleft()

        # Loại bỏ tất cả các phần tử trong deque nhỏ hơn num vì chúng sẽ không thể là max

        while q and num\_list[q[-1]] <= num:

            q.pop()

        # Thêm chỉ số hiện tại vào deque

        q.append(i)

        # Thêm max vào kết quả khi đã qua đủ k phần tử

        if i >= k - 1:

            result.append(num\_list[q[0]])

    return result

# Xử lý chuỗi và đếm từ

def process\_string\_and\_dict(text):

    # Chuyển chuỗi thành chữ thường và tách các từ

    words = text.lower().split()

    # Sử dụng dictionary để đếm số lần xuất hiện của từng từ

    word\_count = {}

    for word in words:

        if word in word\_count:

            word\_count[word] += 1

        else:

            word\_count[word] = 1

    return word\_count

# Đọc nội dung từ file và xử lý mỗi dòng chuỗi

def read\_and\_process\_file(file\_path):

    with open(file\_path, 'r') as file:

        lines = file.readlines()

    # Xử lý mỗi dòng và lưu kết quả vào list

    processed\_lines = []

    for line in lines:

        processed\_lines.append(process\_string\_and\_dict(line.strip()))

    return processed\_lines

# 1. Tìm giá trị lớn nhất trong mỗi cửa sổ với num\_list và k

num\_list = [3, 4, 5, 1, -44, 5, 10, 12, 33, 1]

k = 3

print("Max in sliding window:", sliding\_window\_max(num\_list, k))

# 2. Xử lý chuỗi và đếm từ

text = "Hello world hello Python World"

print("Word count:", process\_string\_and\_dict(text))

\_ tìm giá trị lớn nhất trong mỗi cửa sổ:

\_ xử lý chuỗi và đếm từ trong dictionary

\_ đọc nội dung từ file và xử lý chuỗi

Output:

Max in sliding window: [5, 5, 5, 5, 10, 12, 33, 33]

Word count: {'hello': 2, 'world': 2, 'python': 1}

Bài 2:

def count\_letters(word):

    # Chuyển tất cả chữ cái về dạng chữ thường để việc so sánh dễ dàng hơn

    word = word.lower()

    # Khởi tạo dictionary để lưu kết quả

    letter\_count = {}

    # Duyệt qua từng chữ cái trong từ

    for letter in word:

        if letter in letter\_count:

            letter\_count[letter] += 1

        else:

            letter\_count[letter] = 1

    return letter\_count

word = input("Nhập một từ: ")

result = count\_letters(word)

print(result)

Các bước thực hiện:

\_Chuyển các kí tự về chữ thường.  
\_Khởi tạo dictionary để lưu kết qur

\_Duyệt từng chữ cái

Nhập một từ: hidden

{'h': 1, 'i': 1, 'd': 2, 'e': 1, 'n': 1}

Bài 3:

import re

# Hàm để đọc nội dung file và trả về một list các từ

def read\_file(file\_path):

    try:

        with open(file\_path, 'r', encoding='utf-8') as file:

            content = file.readlines()

        return content

    except FileNotFoundError:

        print("File không tồn tại.")

        return []

# Hàm để tách các từ từ mỗi dòng và chuyển thành chữ thường

def extract\_words(lines):

    words = []

    for line in lines:

        # Dùng regex để tách từ, chỉ lấy chữ cái

        words += re.findall(r'\b[a-zA-Z]+\b', line.lower())

    return words

# Hàm để đếm số lần xuất hiện của các từ trong danh sách

def count\_word\_occurrences(words):

    word\_count = {}

    for word in words:

        if word in word\_count:

            word\_count[word] += 1

        else:

            word\_count[word] = 1

    return word\_count

# Hàm chính để xử lý toàn bộ quá trình

def count\_words\_in\_file(file\_path):

    lines = read\_file(file\_path)  # Đọc nội dung file

    if not lines:  # Nếu file không tồn tại hoặc không có nội dung

        return {}

    words = extract\_words(lines)  # Tách từ

    word\_count = count\_word\_occurrences(words)  # Đếm từ

    return word\_count

# Ví dụ sử dụng:

file\_path = 'D:/bai\_tap\_ctdl\_gt/cpp/thi/P1\_data.txt'  # Thay đổi với đường dẫn thực tế đến file .txt

result = count\_words\_in\_file(file\_path)

print(result)

\_Tách từ

\_Đếm số lần xuất hiện từ trong danh sách

\_Xử lý toàn bộ quá trình

Output:

{'he': 1, 'who': 3, 'conquers': 1, 'himself': 1, 'is': 3, 'the': 5, 'mightiest': 1, 'warrior': 1, 'try': 2, 'not': 1, 'to': 3, 'become': 2, 'a': 7, 'man': 6, 'of': 4, 'success': 3, 'but': 1, 'rather': 1, 'value': 1, 'one': 4, 'with': 4, 'courage': 1, 'makes': 1, 'majority': 1, 'secret': 1, 'in': 4, 'life': 2, 'for': 3, 'be': 1, 'ready': 1, 'his': 2, 'opportunity': 1, 'when': 2, 'it': 2, 'comes': 2, 'successful': 2, 'will': 2, 'profit': 1, 'from': 1, 'mistakes': 1, 'and': 1, 'again': 1, 'different': 1, 'way': 1, 'can': 3, 'lay': 1, 'firm': 1, 'foundation': 1, 'bricks': 1, 'others': 1, 'have': 1, 'thrown': 1, 'at': 1, 'him': 1, 'usually': 1, 'those': 1, 'are': 1, 'too': 1, 'busy': 1, 'looking': 1, 'we': 3, 'cannot': 1, 'solve': 1, 'problems': 1, 'kind': 1, 'thinking': 1, 'employed': 1, 'came': 1, 'up': 1, 'them': 1, 'just': 2, 'small': 1, 'positive': 1, 'thought': 1, 'morning': 1, 'change': 1, 'your': 1, 'whole': 1, 'day': 1, 'you': 3, 'get': 2, 'everything': 1, 'want': 2, 'if': 1, 'help': 1, 'enough': 1, 'other': 1, 'people': 1, 'what': 1, 'they': 1}

Bài 4:

def initialize\_dp\_matrix(m, n):

    """Khởi tạo ma trận dp với kích thước (m+1) x (n+1) và thiết lập các giá trị ban đầu."""

    dp = [[0] \* (n + 1) for \_ in range(m + 1)]

    # Khởi tạo các giá trị cho dp

    for i in range(m + 1):

        dp[i][0] = i  # Khoảng cách từ S[0...i] đến chuỗi rỗng

    for j in range(n + 1):

        dp[0][j] = j  # Khoảng cách từ chuỗi rỗng đến T[0...j]

    return dp

def compute\_levenshtein\_distance(source, target):

    """Tính toán khoảng cách Levenshtein giữa hai chuỗi source và target."""

    m = len(source)

    n = len(target)

    # Khởi tạo ma trận dp

    dp = initialize\_dp\_matrix(m, n)

    # Điền ma trận dp bằng cách tính toán 3 phép biến đổi

    for i in range(1, m + 1):

        for j in range(1, n + 1):

            if source[i - 1] == target[j - 1]:

                dp[i][j] = dp[i - 1][j - 1]  # Nếu giống nhau, không cần thay thế

            else:

                dp[i][j] = min(

                    dp[i - 1][j] + 1,   # Xoá

                    dp[i][j - 1] + 1,   # Thêm

                    dp[i - 1][j - 1] + 1  # Thay thế

                )

    return dp[m][n]

def levenshtein\_distance(source, target):

    """Hàm chính tính toán khoảng cách Levenshtein giữa hai chuỗi."""

    return compute\_levenshtein\_distance(source, target)

# Ví dụ sử dụng:

source = input("Nhập chuỗi nguồn (source): ")

target = input("Nhập chuỗi đích (target): ")

result = levenshtein\_distance(source, target)

print(f"Levenshtein distance between '{source}' and '{target}': {result}")

\_Khởi tạo ma trận

\_Tính toán khoảng cách từng từ

\_In ra màn hình

Output:

Nhập chuỗi nguồn (source): hint

Nhập chuỗi đích (target): hive

Levenshtein distance between 'hint' and 'hive': 2