Assignment 1

Given two **3-dimensional matrices** M_1 and M_2 of dimensions $k \times m \times n$. Each **slice** of the 3D matrices M_1 and M_2 at depth i ($i = 1, 2, 3, ..., k$) is a **2D submatrix** of size $m \times n$. design an efficient algorithm to compute the following:

1. The **difference** between two consecutive $(m \times n)$ submatrices in both M_1 and M_2.

   The **difference between consecutive slices** in a matrix M is defined as:

   $$D_M(i) = M[i] - M[i-1], \quad \text{for } 1 \leq i < k$$

2. The **difference** between the i^{th} $(m \times n)$ submatrix in M_1 and the i^{th} $(m \times n)$ submatrix in M_2.

   The **difference between corresponding slices** in M_1 and M_2 is defined as:

   $$D_{M_1,M_2}(i) = M_1[i] - M_2[i], \quad \text{for } 1 \leq i \leq k$$

Input

- Two **3D integer matrices** M_1 and M_2 of size $k \times m \times n$.

Output

- A matrix representing the differences between consecutive **slices** in M_1.
- A matrix representing the differences between consecutive **slices** in M_2.
- A matrix representing the differences between corresponding **slices** in M_1 and M_2.

Constraints

- $1 \leq k, m, n \leq 500$
- $-10^6 \leq M_1[i][j][l], M_2[i][j][l] \leq 10^6$

Assignment 2

Problem Statement 2.1

You are given an array **nums** of size **n** consisting of integers. Your task is to preprocess the array to efficiently compute the sum of elements within a given range. Specifically, you need to answer multiple queries, where each query is given as a pair $(\text{left}, \text{right})$ and requires computing the sum of elements from index **left** to index **right** (both inclusive).

Input Format

- An integer n representing the size of the array.
- An array nums of length n containing integer values.
- An integer q representing the number of queries.
- q pairs of integers $(\text{left}, \text{right})$ representing the range for each query.

Output Format

For each query, output a single integer representing the sum of elements from index **left** to index **right** (inclusive).

Constraints

- $1 \leq n \leq 10^5$
- $-10^4 \leq \text{nums}[i] \leq 10^4$
- $1 \leq q \leq 10^5$
- $0 \leq \text{left} \leq \text{right} < n$

Example

Input:

```
n = 10
nums = [3, 2, -1, 6, 5, 4, -3, 3, 7, 2]
q = 3
Queries = [(1, 4), (3, 7), (0, 9)]
```

Output:

```
12
15
28
```

Explanation:

- Sum from index 1 to 4 $\rightarrow 2+(-1)+6+5 = 12$
- Sum from index 3 to 7 $\rightarrow 6+5+4+(-3)+3 = 15$
- Sum from index 0 to 9 $\rightarrow$ Total sum of the array $= 28$

Implementation Guidelines

- Precompute the **prefix sum array** in $O(n)$ time.
- Answer each query in $O(1)$ time using the formula:

$$\text{Range Sum}(\text{left}, \text{right}) = \text{prefixSum}[\text{right}] - \text{prefixSum}[\text{left} - 1]$$

If left $= 0$, then Range Sum(left, right) $=$ prefixSum[right].

- -

Problem Statement 2.2

You are given a **2D matrix** of size $m \times n$, consisting of integers. Your task is to preprocess the matrix so that you can efficiently compute the sum of elements within a given submatrix. Specifically, you need to answer multiple queries where each query is given as a tuple (r_1, c_1, r_2, c_2) and requires computing the sum of all elements in the submatrix defined by the top-left corner (r_1, c_1) and the bottom-right corner (r_2, c_2) (both inclusive).

Input Format

- Two integers m and n representing the number of rows and columns in the matrix.
- A $m \times n$ matrix of integers.
- An integer q representing the number of queries.
- q queries, each given as four integers (r_1, c_1, r_2, c_2) defining the top-left and bottom-right coordinates of the submatrix.

Output Format

For each query, output a single integer representing the sum of elements within the given submatrix.

Constraints

- $1 \leq m, n \leq 10^3$
- $-10^4 \leq$ **matrix**[i][j] $\leq 10^4$
- $1 \leq q \leq 10^5$
- $0 \leq r_1 \leq r_2 < m$
- $0 \leq c_1 \leq c_2 < n$

Example

Input:

```
m = 4, n = 5
matrix = [
    [3, 2, 4, 1, 7],
    [1, 6, 2, 8, 3],
    [5, 9, 3, 4, 6],
    [7, 2, 8, 1, 5]
]
q = 2
Queries = [(1, 1, 2, 3), (0, 0, 3, 4)]
```

Output:

```
32
68
```

Explanation:

- Sum of submatrix $(1,1)$ to $(2,3)$:

$$6+2+8+9+3+4=32$$

- Sum of the entire matrix $(0,0)$ to $(3,4)$:

$$\text{Total sum} = 68$$

Expected Solution Approach

To efficiently compute the sum of any submatrix in $O(1)$ time after preprocessing, we use the **prefix sum matrix** where:

$$\text{prefixSum}[i][j] = \text{matrix}[i][j] + \text{prefixSum}[i-1][j] + \text{prefixSum}[i][j-1] - \text{prefixSum}[i-1][j-1]$$

Using this, the sum of any submatrix (r_1, c_1) to (r_2, c_2) can be computed as:

$$\text{Submatrix Sum} = \text{prefixSum}[r_2][c_2] - \text{prefixSum}[r_1-1][c_2] - \text{prefixSum}[r_2][c_1-1] + \text{prefixSum}[r_1-1][c_1-1]$$

(handling edge cases when $r_1 = 0$ or $c_1 = 0$).

Assignment 3

Implement a function that computes **WER** using **Levenshtein Distance**. The Word Error Rate (WER) is given by:

$$WER = \frac{S+D+I}{N}$$

where:

- S = Number of `substitutions` (words incorrectly transcribed).
- D = Number of `deletions` (words missing in the hypothesis).
- I = Number of `insertions` (extra words added in the hypothesis).
- N = Total number of words in the `reference (ground truth) sentence`.

To compute S, D, I, we use the ***Levenshtein Distance Algorithm***.

Algorithm for Computing Levenshtein Distance

The Levenshtein Distance between two-word sequences is the minimum number of operations (insertions, deletions, or substitutions) required to transform one sequence into another. It is commonly used to compute the word error rate (WER) in speech recognition.

Algorithm 1 Levenshtein Distance Algorithm

Input: Reference sentence R, Hypothesis sentence H
Output: Minimum edit distance between R and H
Let $m =$ length of R, $n =$ length of H
Create a matrix D of size $(m+1) \times (n+1)$
for $i = 0$ to m **do**
    $D[i][0] = i$ ▷ Initialize first column (deletion cost)
end for
for $j = 0$ to n **do**
    $D[0][j] = j$ ▷ Initialize first row (insertion cost)
end for
for $i = 1$ to m **do**
    for $j = 1$ to n **do**
        if $R[i-1] = H[j-1]$ **then**
            $cost = 0$
        else
            $cost = 1$ ▷ Substitution cost
        end if
        $D[i][j] = \min(D[i-1][j]+1, D[i][j-1]+1, D[i-1][j-1]+cost)$
    end for
end for
Return $D[m][n]$ ▷ Minimum edit distance

Mathematical Formulation

The recurrence relation for computing $D[i][j]$ is:

$$D[i][j] = \begin{cases} D[i-1][j]+1 & \text{(Deletion)} \\ D[i][j-1]+1 & \text{(Insertion)} \\ D[i-1][j-1]+\text{cost} & \text{(Substitution, if } R[i-1] \neq H[j-1]) \end{cases}$$

where the ***cost*** is 0 if $R[i-1] = H[j-1]$, otherwise 1.

Input Format

- A string `reference_sentence` (actual spoken sentence).
- A string `hypothesis_sentence` (transcribed sentence by the Speech Recognition system).

Constraints

- $1 \leq$ length of both sentences ≤ 100 words.
- Words consist of lowercase English letters and punctuation.

Output Format

Print the `WER` score rounded to 4 decimal places.

Examples

Example 1:

```
Input:
Reference:   "hello how are you"
Hypothesis:  "hello who are you"

Output:
0.25
```

Explanation:

- ”how” $\rightarrow$ ”who” (substitution)
- Total errors = 1, and reference has 4 words, so:

$$WER = \frac{1}{4} = 0.25$$

Example 2:

```
Input:
Reference:   "speech processing is interesting"
Hypothesis:  "speech is very interesting"

Output:
0.5
```

Explanation:

- ”processing” is `deleted`
- ”very” is an `insertion`
- Total errors = 2, and reference has 4 words, so:

$$WER = \frac{2}{4} = 0.5$$

Python Function Signature

```python
def compute_wer(reference_sentence: str, hypothesis_sentence: str) -> float:
    # Implementation here
```