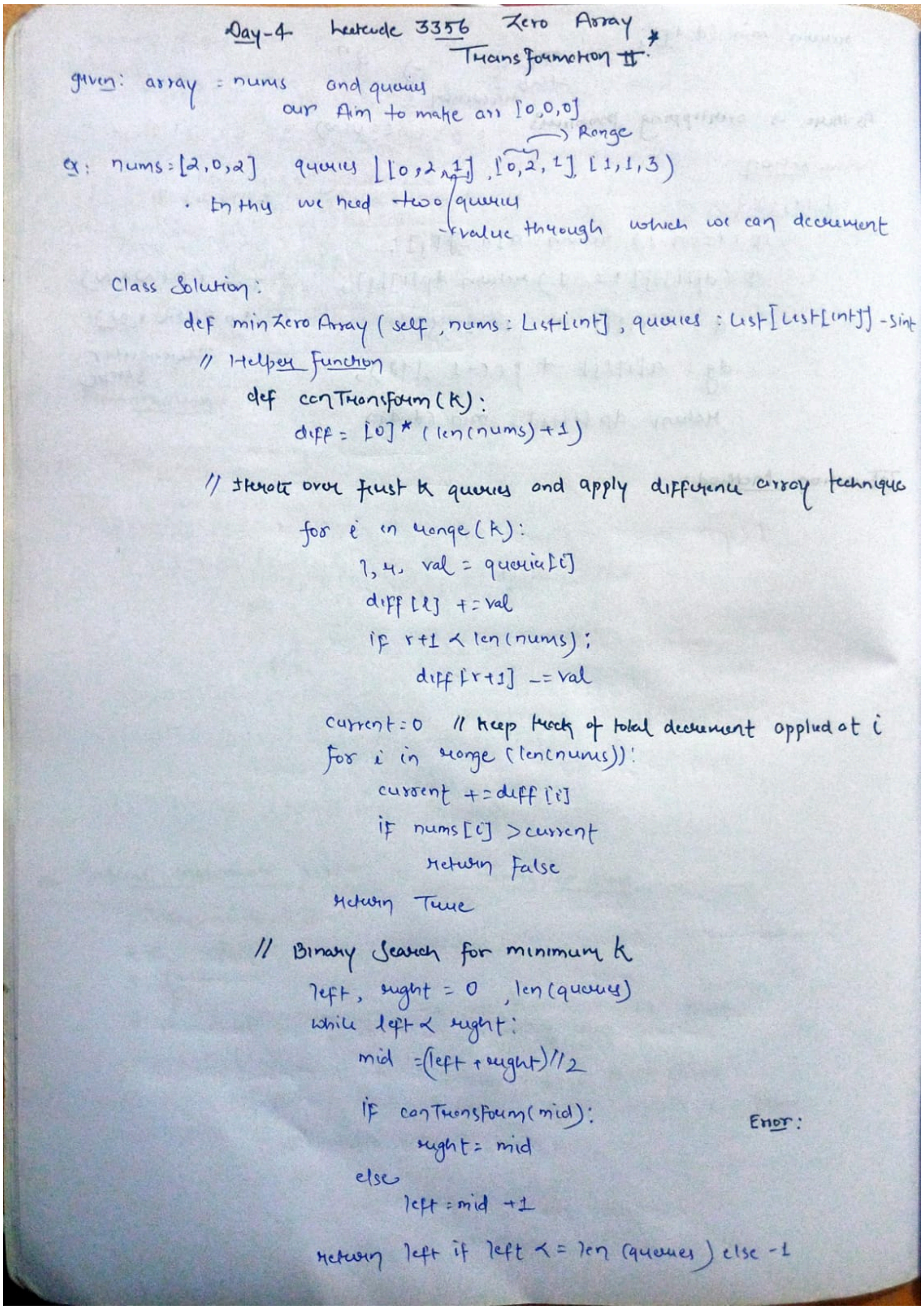
Day_4: DSA



Day-4 Leetcode 3356 Zero Array Transformation II *

given: array = nums and queries
our Aim to make arr [0,0,0]

Ex: nums=[2,0,2] queries [[0,2,1], [0,2,1], [1,1,3])
→ Range
- In this we need two queries
  value through which we can decrement

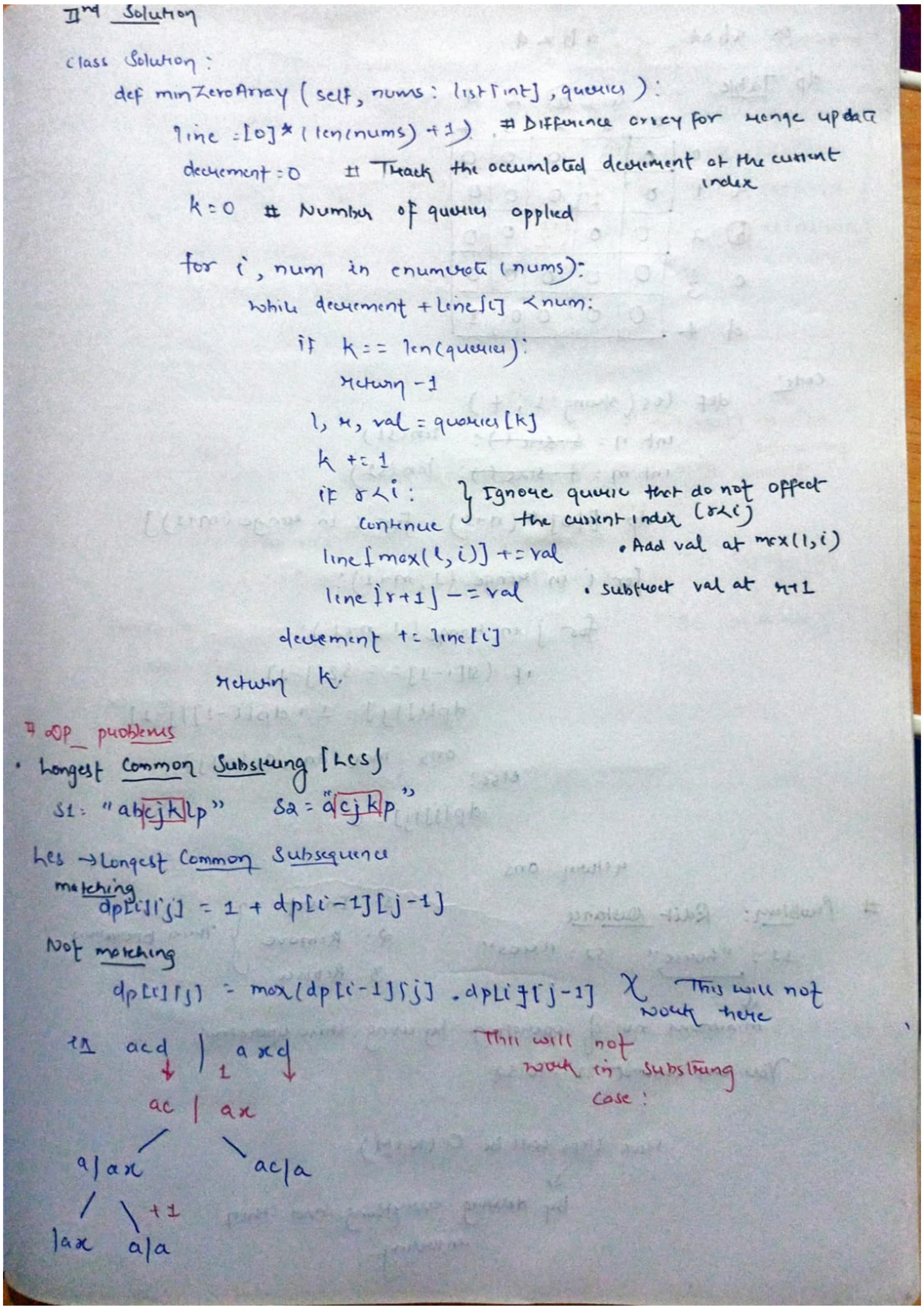
```
Class Solution:
    def minZeroArray(self, nums: List[int], queries: List[List[int]] -> int
      // Helper Function
        def canTransform(k):
            diff = [0] * (len(nums)+1)

      // Iterate over first k queries and apply difference array technique
            for i in range(k):
                l, r, val = queries[i]
                diff[l] += val
                if r+1 < len(nums):
                    diff[r+1] -= val
            current=0  // keep track of total decrement applied at i
            for i in range(len(nums)):
                current += diff[i]
                if nums[i] > current
                    return False
            return True

        // Binary Search for minimum k
            left, right = 0 , len(queries)
            while left < right:
                mid = (left + right)//2

                if canTransform(mid):                    Error:
                    right = mid
            else
                left = mid +1

        return left if left <= len(queries) else -1
```



IInd Solution

```
class Solution:
    def minZeroArray(self, nums: list[int], queries):
        line = [0] * (len(nums) + 1)   # Difference array for range update
        decrement = 0     # Track the accumulated decrement of the current index
        k = 0   # Number of queries applied

        for i, num in enumerate(nums):
            while decrement + line[i] < num:
                if k == len(queries):
                    return -1
                l, r, val = queries[k]
                k += 1
                if r < i:        } Ignore queries that do not affect
                    continue     } the current index (r<i)
                line[max(l, i)] += val      • Add val at max(l,i)
                line[r+1] -= val            • subtract val at r+1
            decrement += line[i]
        return k
```

DP problems

• Longest Common Substring [LCS]

S1: "ab$\boxed{cjk}$lp"   S2 = "a$\boxed{cjk}$p"

Lcs → Longest Common Subsequence

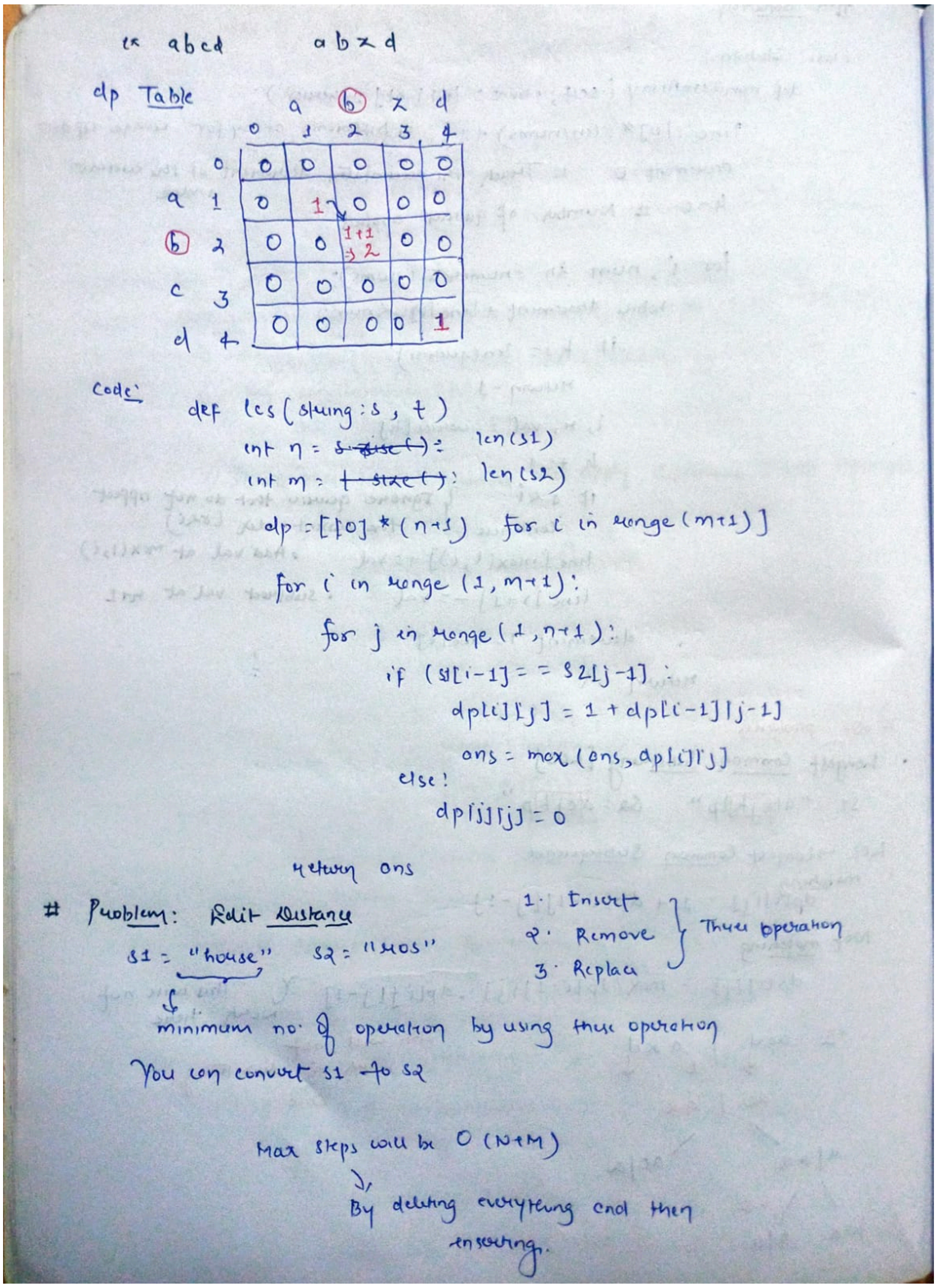
matching
$dp[i][j] = 1 + dp[i-1][j-1]$

Not matching

$dp[i][j] = max(dp[i-1][j], dp[i][j-1])$ ✗ This will not work here

This will not work in substring case!

+1 acd | axd
↓ 1 ↓
ac | ax
a | ax   ac | a
| ax   a | a   +1



ex abcd abxd

dp Table

$$\begin{array}{cc|c|c|c|c|c}
 & & & a & \textcircled{b} & x & d \\
 & & 0 & 1 & 2 & 3 & 4 \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
a & 1 & 0 & 1 & 0 & 0 & 0 \\
\hline
\textcircled{b} & 2 & 0 & 0 & 1+1 \Rightarrow 2 & 0 & 0 \\
\hline
c & 3 & 0 & 0 & 0 & 0 & 0 \\
\hline
d & 4 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

Code:

```
def lcs(string: s, t)
    int n = ~~s.size()~~: len(s1)
    int m = ~~t.size()~~: len(s2)
      dp = [[0] * (n+1) for i in range(m+1)]

      for i in range (1, m+1):
          for j in range (1, n+1):
              if (s1[i-1] == s2[j-1]:
                  dp[i][j] = 1 + dp[i-1][j-1]
                   ans = max(ans, dp[i][j]
              else:
                  dp[i][j] = 0

      return ans
```

Problem: Edit Distance

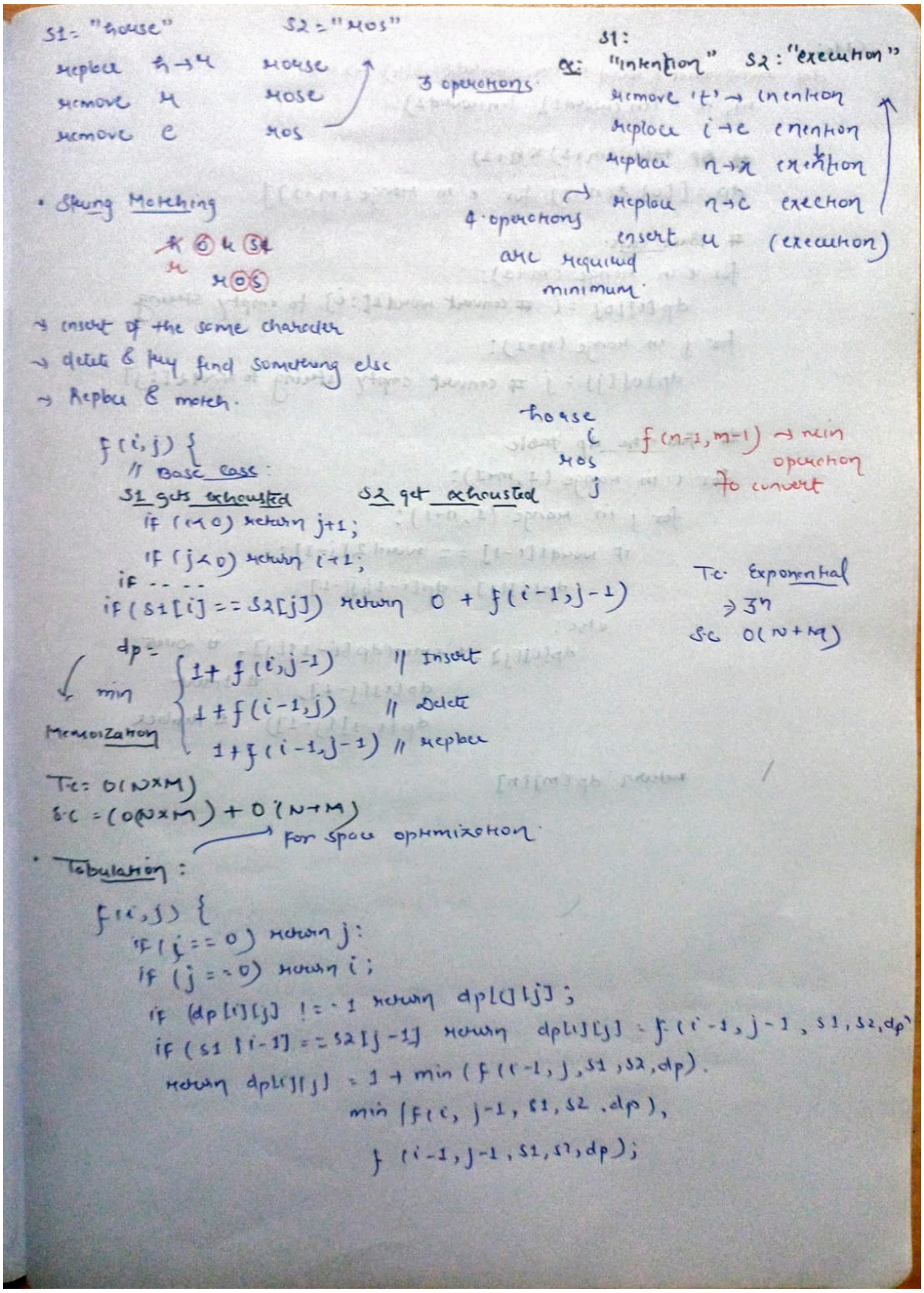
s1 = $\underbrace{\text{"house"}}$ s2 = "ros"

1. Insert
2. Remove
3. Replace

} Three operation

minimum no. of operation by using three operation You can convert s1 to s2

Max steps will be O (N+M)
↓
By deleting everything and then inserting.



S1 = "horse" S2 = "ros"

replace	h → r	rorse
remove	r	rose
remove	e	ros

3 operations

ex: S1: "intention" S2: "execution"

remove 't' →		inention
replace	i → e	enention
replace	n → x	exention
replace	n → c	exection
insert	u	(execution)

4 operations are required minimum.

- String Matching

  ~~h~~ o r ~~se~~
  r
  r o s

→ insert of the same character
→ delete & try find something else
→ Replace & match.

```
f(i, j) {
  // Base case:
  S1 gets exhausted        S2 get exhausted
   if (i<0) return j+1;
   if (j<0) return i+1;
  if -- --
 if (S1[i] == S2[j]) return 0 + f(i-1, j-1)
   dp = 
      min { 1 + f(i, j-1)    // Insert
          { 1 + f(i-1, j)    // Delete
          { 1 + f(i-1, j-1)  // replace
```

horse
    i   f(n-1, m-1) → min operation to convert
ros
  j

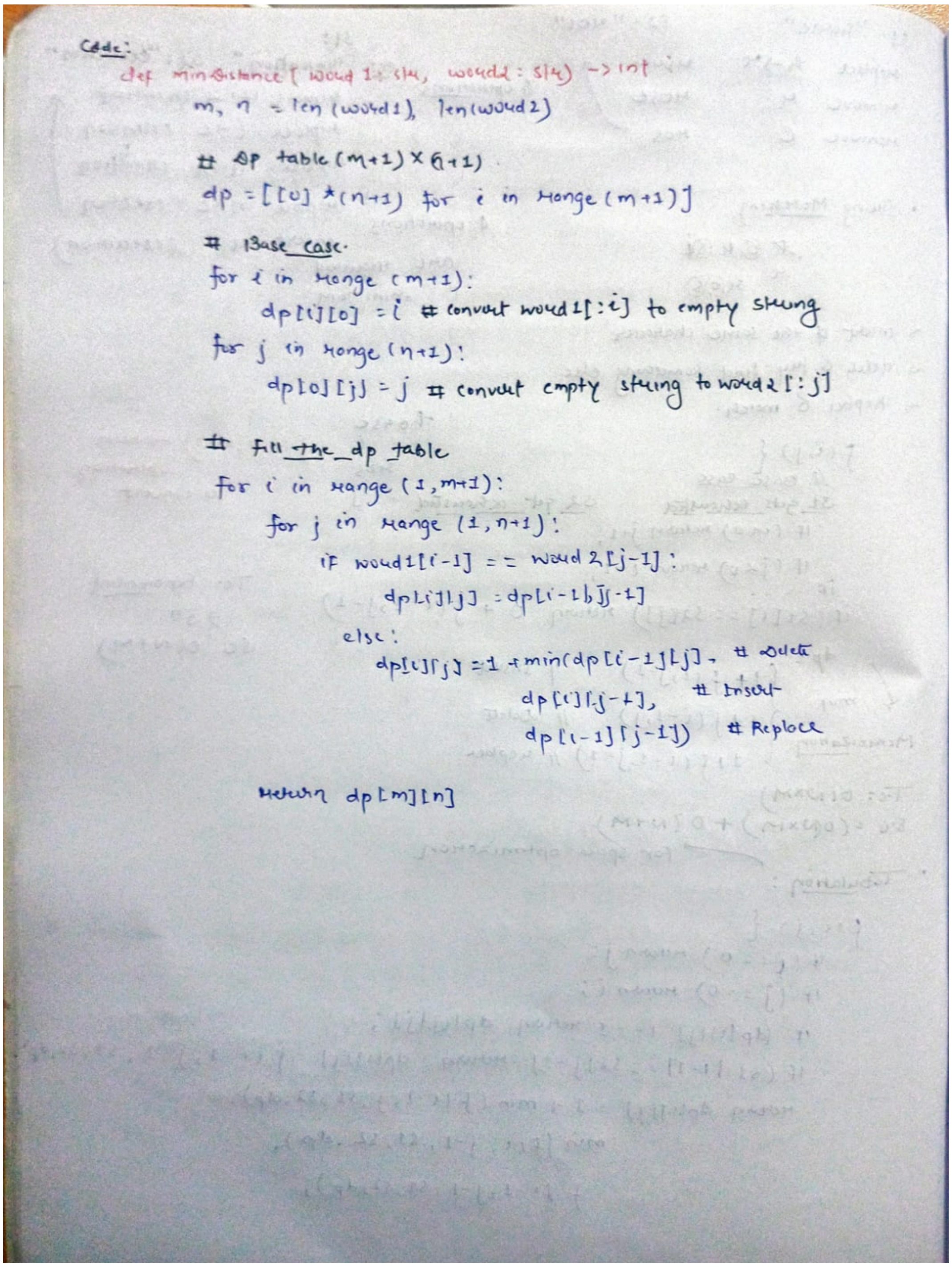
TC: Exponential
 ⇒ 3^n
SC O(N+M)

Memoization

TC: O(N×M)
SC = (O(N×M)) + O(N+M)
→ For space optimization

- Tabulation:

```
f(i, j) {
  if (i == 0) return j;
  if (j == 0) return i;
  if (dp[i][j] != -1 return dp[i][j];
  if (s1[i-1] == s2[j-1] return dp[i][j] = f(i-1, j-1, s1, s2, dp)
  return dp[i][j] = 1 + min ( f(i-1, j, s1, s2, dp).
                  min (f(i, j-1, s1, s2, dp),
                    f (i-1, j-1, s1, s2, dp);
```



Code:

```
def minDistance(word1: str, word2: str) -> int:
    m, n = len(word1), len(word2)

    # dp table (m+1) x (n+1)
    dp = [[0] *(n+1) for i in range(m+1)]

    # Base case.
    for i in range(m+1):
        dp[i][0] = i # convert word1[:i] to empty string
    for j in range(n+1):
        dp[0][j] = j # convert empty string to word2[:j]

    # Fill the dp table
    for i in range(1,m+1):
        for j in range(1,n+1):
            if word1[i-1] == word2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j],   # delete
                                   dp[i][j-1],   # Insert
                                   dp[i-1][j-1]) # Replace

    return dp[m][n]
```