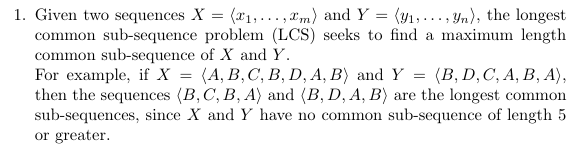
**ASSIGNMENT – 7**

**AIM:**

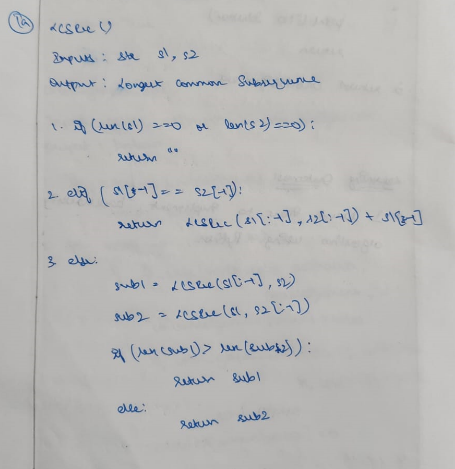
To solve and implement the given problems using Dynamic Programming

**Qn1:**

****

****

**Psuedo Code:**

****

**Source Code:**

def LCSRec(s1, s2):

    if len(s1)==0 or len(s2)==0:

        return ""

    elif s1[-1] == s2[-1]:

        return LCSRec(s1[:-1], s2[:-1]) + s1[-1]

    else:

        sub1 = LCSRec(s1[:-1], s2)

        sub2 = LCSRec(s1, s2[:-1])

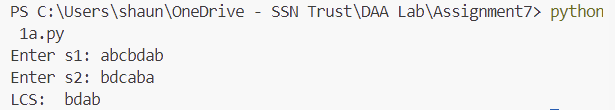
        return sub1 if len(sub1)>len(sub2) else sub2

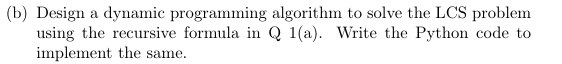
s1 = input("Enter s1: ")

s2 = input("Enter s2: ")

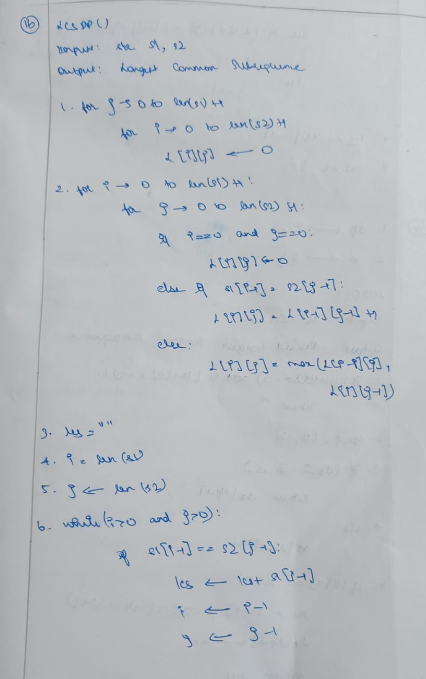
print("LCS: ", LCSRec(s1, s2))

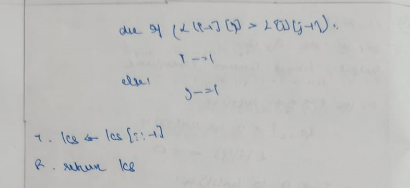
**Output:**

****

****

**Psuedo Code:**

****

****

**Source Code:**

def LCSDP(s1, s2):

    L = [[0 for i in range(len(s2)+1)] for j in range(len(s1)+1)]

    for i in range(len(s1)+1):

        for j in range(len(s2)+1):

            if i==0 or j==0:

                L[i][j] = 0

            elif s1[i-1] == s2[j-1]:

                L[i][j] = L[i-1][j-1] + 1

            else:

                L[i][j] = max(L[i-1][j], L[i][j-1])

    lcs = ""

    i = len(s1)

    j = len(s2)

    while i>0 and j>0:

        if s1[i-1] == s2[j-1]:

            lcs += s1[i-1]

            i -= 1

            j -= 1

        elif L[i-1][j] > L[i][j-1]:

            i -= 1

        else:

            j -= 1

    lcs = lcs[::-1]

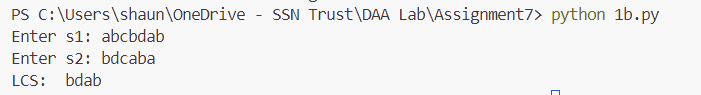
    return lcs

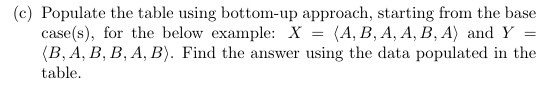
s1 = input("Enter s1: ")

s2 = input("Enter s2: ")

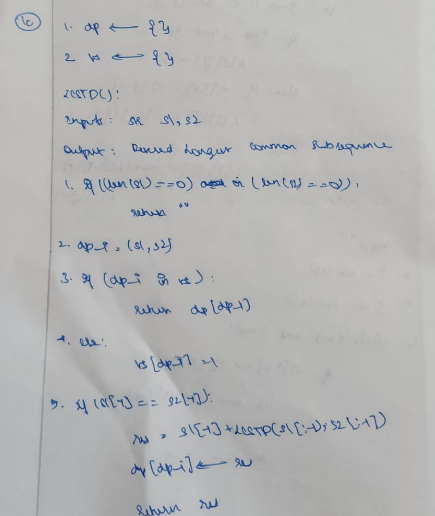
print("LCS: ", LCSDP(s1, s2))

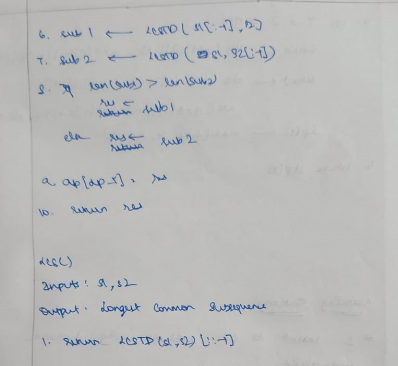
**Output:**

****

****

**Psuedo Code:**

****

****

**Source Code:**

dp = {}

vs = {}

def LCSTD(s1, s2):

    if len(s1)==0 or len(s2)==0:

        return ""

    dp\_i = (s1, s2)

    if dp\_i in vs:

        return dp[dp\_i]

    else:

        vs[dp\_i] = 1

    if s1[-1] == s2[-1]:

        res = s1[-1] + LCSTD(s1[:-1], s2[:-1])

        dp[dp\_i] = res

        return res

    sub1 = LCSTD(s1[:-1], s2)

    sub2 = LCSTD(s1, s2[:-1])

    res = sub1 if len(sub1)>len(sub2) else sub2

    dp[dp\_i] = res

    return res

def LCS(s1, s2):

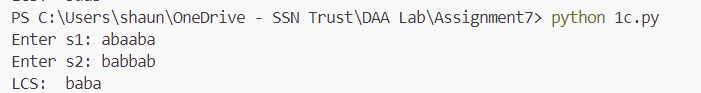
    return LCSTD(s1, s2)[::-1]

s1 = input("Enter s1: ")

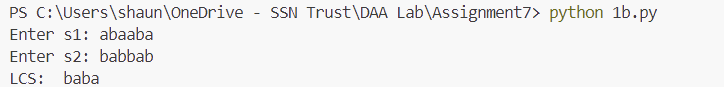
s2 = input("Enter s2: ")

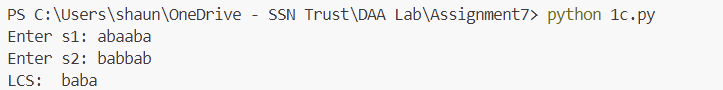
print("LCS: ", LCS(s1, s2))

**Output:**

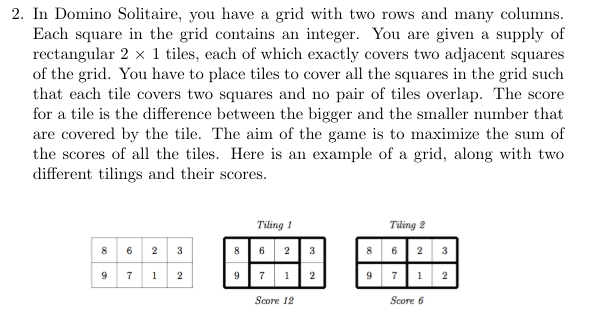
****

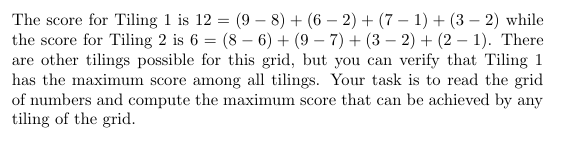
****

****

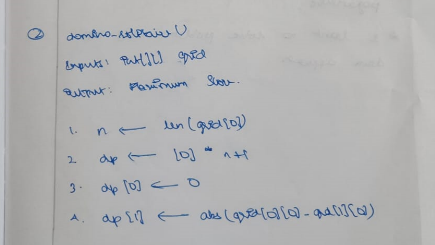
****

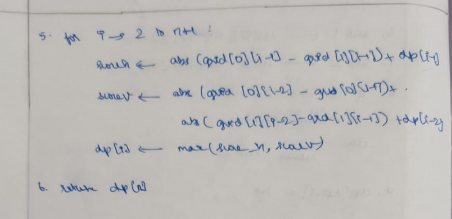
**Qn2:**

****

****

**Psuedo Code:**

****

****

**Source Code:**

def domino\_solitaire(grid):

    n = len(grid[0])

    dp = [0] \* (n + 1)

    dp[0] = 0

    dp[1] = abs(grid[0][0] - grid[1][0])

    for i in range(2, n + 1):

        score\_h = abs(grid[0][i-1] - grid[1][i-1]) + dp[i-1]

        score\_v = abs(grid[0][i-2] - grid[0][i-1]) + abs(grid[1][i-2] - grid[1][i-1]) + dp[i-2]

        dp[i] = max(score\_h, score\_v)

    return dp[n]

grid = [

    [8,6,2,3],

    [9,7,1,2]

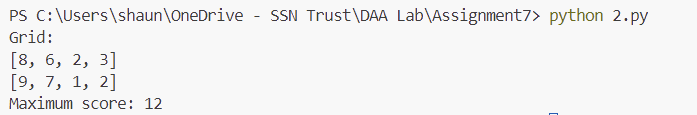
]

print("Grid:")

print(\*grid, sep="\n")

print("Maximum score:", domino\_solitaire(grid))

**Output:**

****

**Learning Outcomes:**

* I learnt to analyse and implement dynamic programming approach
* I learnt how to implement top-down approach
* I learnt about memoization