Date: 14.04.24
ExNo: 7

Reg No: 3122225001127
Name: Shaun Allan H

ASSIGNMENT – 7

AIM:

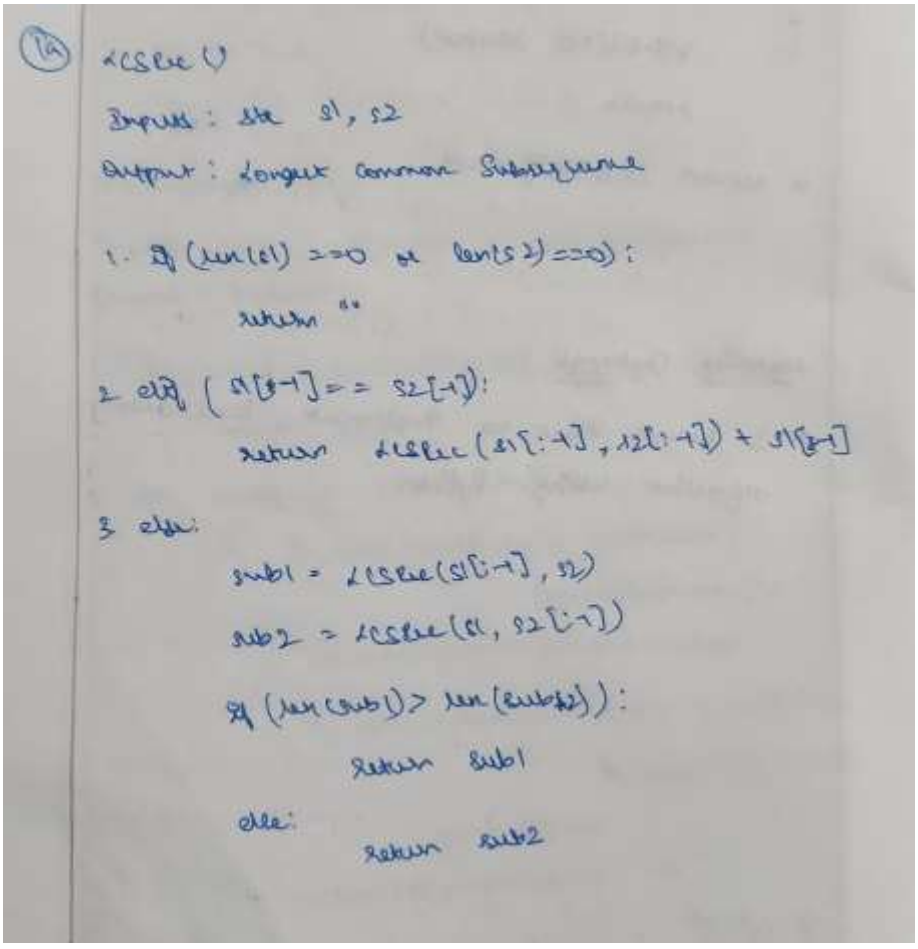
To solve and implement the given problems using Dynamic Programming

Qn1:

1. Given two sequences $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$, the longest common sub-sequence problem (LCS) seeks to find a maximum length common sub-sequence of X and Y.
   For example, if $X = \langle A, B, C, B, D, A, B \rangle$ and $Y = \langle B, D, C, A, B, A \rangle$, then the sequences $\langle B, C, B, A \rangle$ and $\langle B, D, A, B \rangle$ are the longest common sub-sequences, since X and Y have no common sub-sequence of length 5 or greater.

(a) Obtain a recursive formula for the LCS problem.

Psuedo Code:



Date: 14.04.24
ExNo: 7

Reg No: 3122225001127
Name: Shaun Allan H

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:

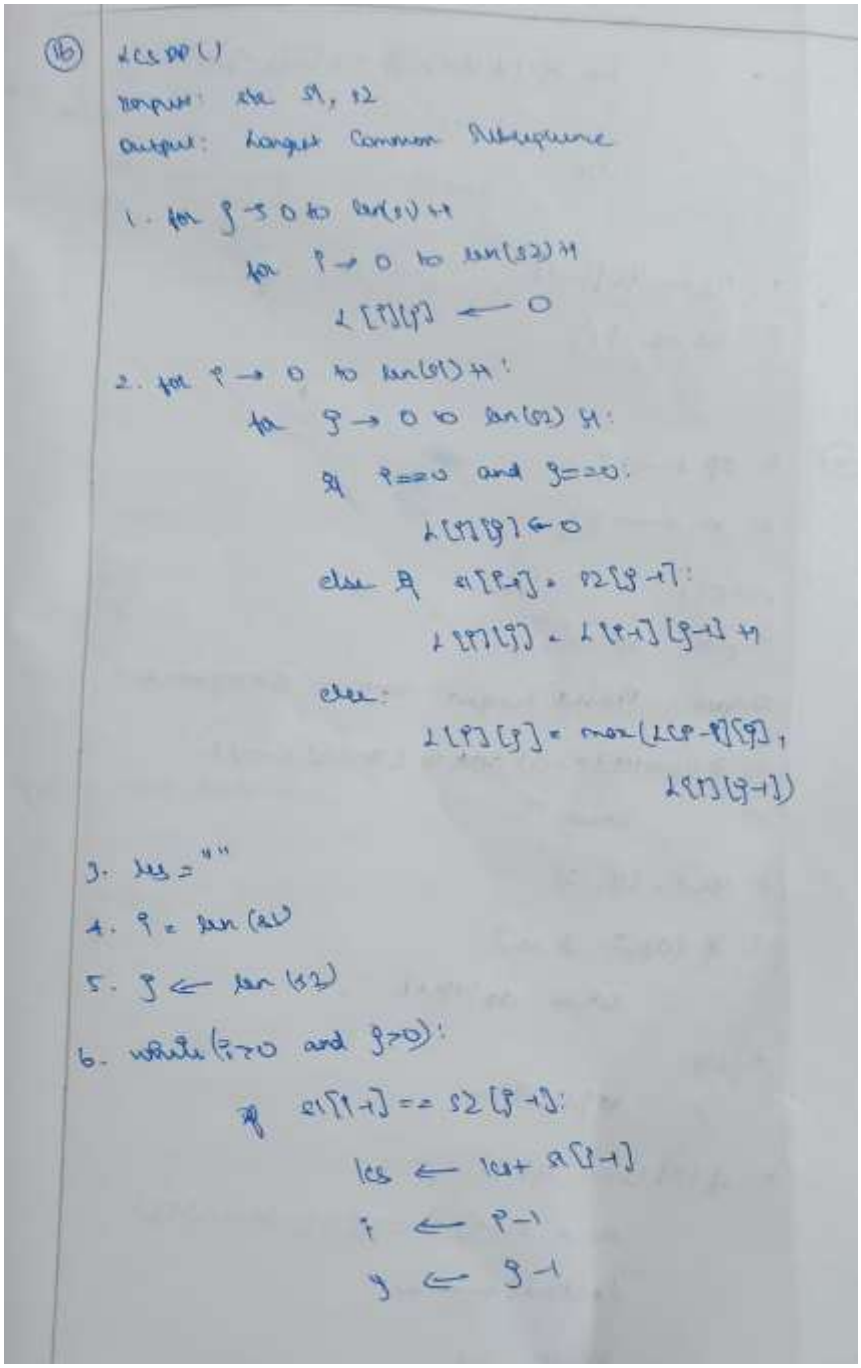
```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment7> python
 1a.py
Enter s1: abcbdab
Enter s2: bdcaba
LCS:  bdab
```

(b) Design a dynamic programming algorithm to solve the LCS problem using the recursive formula in Q 1(a). Write the Python code to implement the same.

Date: 14.04.24 Reg No: 3122225001127

ExNo: 7 Name: Shaun Allan H

Psuedo Code:

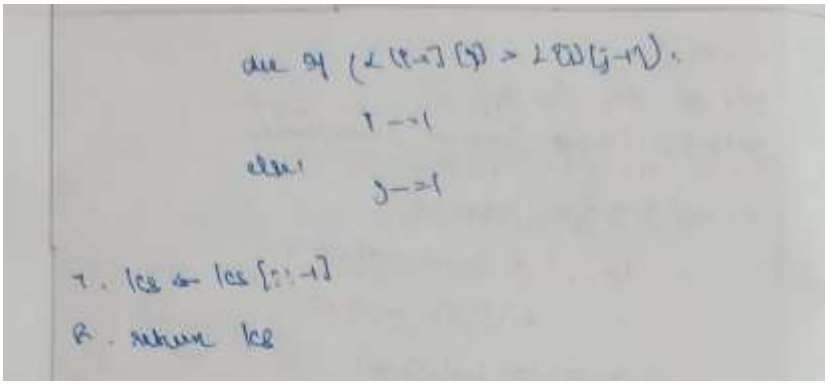


(15) LCS DP()

Input: str s1, s2

Output: Longest Common Subsequence

1. for i -> 0 to len(s1)+1
   for j -> 0 to len(s2)+1
   L[i][j] <- 0
2. for i -> 0 to len(s1)+1:
   for j -> 0 to len(s2)+1:
   if i==0 and j==0:
   L[i][j] <- 0
   else if s1[i-1] = s2[j-1]:
   L[i][j] = L[i-1][j-1]+1
   else:
   L[i][j] = max(LCP-1[j], L[i][j-1])
3. lcs = ""
4. i = len(s1)
5. j <- len(s2)
6. while(i>0 and j>0):
   if s1[i-1] == s2[j-1]:
   lcs <- lcs+s1[i-1]
   i <- i-1
   j <- j-1



else if (L[i-1][j] > L[i][j-1]):
i -> i-1
else:
j -> j-1

7. lcs <- lcs[::-1]
8. return lcs

Date: 14.04.24 Reg No: 3122225001127
ExNo: 7 Name: Shaun Allan H

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:

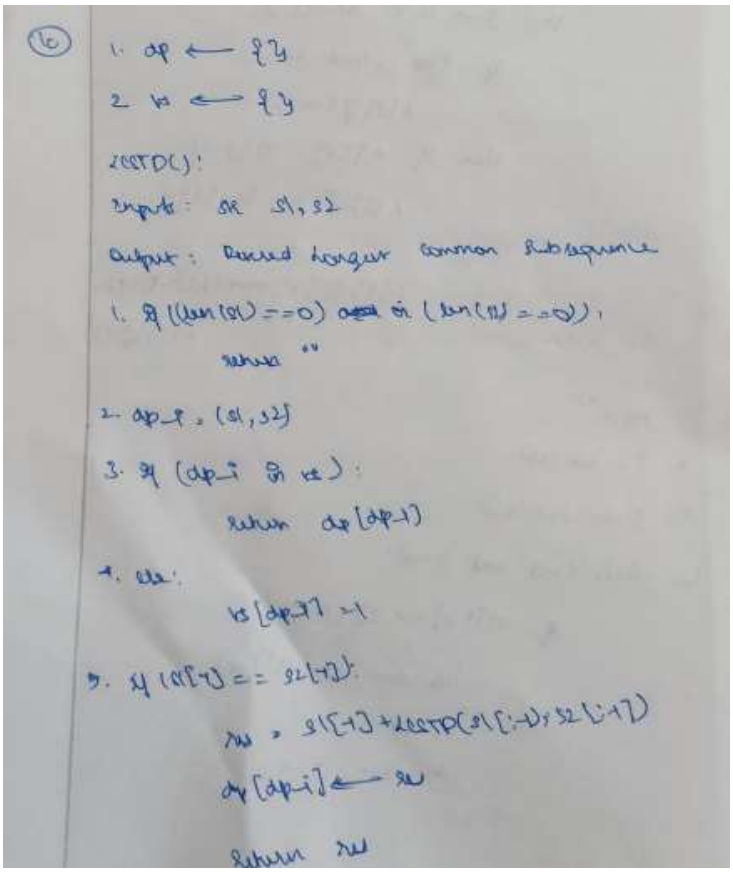
```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment7> python 1b.py
Enter s1: abcbdab
Enter s2: bdcaba
LCS:  bdab
```

Date: 14.04.24 Reg No: 3122225001127

ExNo: 7 Name: Shaun Allan H

(c) Populate the table using bottom-up approach, starting from the base case(s), for the below example: $X = \langle A, B, A, A, B, A \rangle$ and $Y = \langle B, A, B, B, A, B \rangle$. Find the answer using the data populated in the table.

Psuedo Code:



(c)

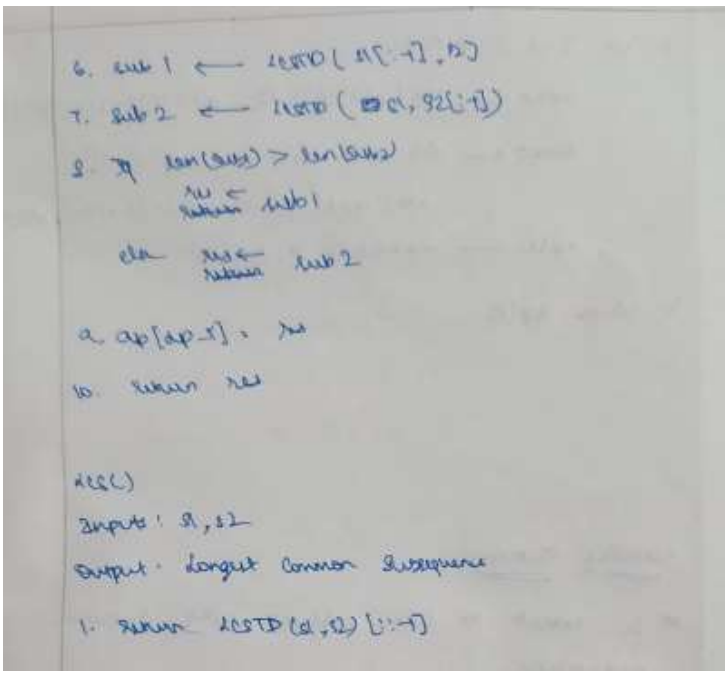
1. dp ← {}
2. ss ← {}

LCSTD():

Inputs: Str S1, S2

Output: Stored longest common subsequence

1. If ((len(S1) == 0) or (len(S2) == 0)):
   return ""
2. dp_i = (S1, S2)
3. If (dp_i in ss):
   return dp[dp_i]
4. else:
   ss[dp_i] = 1
5. If (S1[-1] == S2[-1]):
   res = S1[-1] + LCSTD(S1[:-1], S2[:-1])
   dp[dp_i] ← res
   return res



6. sub1 ← LCSTD(S1[:-1], S2)
7. sub2 ← LCSTD(S1, S2[:-1])
8. If len(sub1) > len(sub2)
   res ← return sub1
   else res ← return sub2
9. dp[dp_i] = res
10. return res

LCS()

Inputs: S1, S2

Output: Longest common subsequence

1. return LCSTD(S1, S2)[::-1]

Date: 14.04.24 Reg No: 3122225001127
ExNo: 7 Name: Shaun Allan H

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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment7> python 1c.py
Enter s1: abaaba
Enter s2: babbab
LCS:  baba
```

Date: 14.04.24 Reg No: 3122225001127
ExNo: 7 Name: Shaun Allan H

(d) Compare the output/answers obtained in Q 1(b) and Q 1(c) for the given example.

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment7> python 1b.py
Enter s1: abaaba
Enter s2: babbab
LCS:  baba
```

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment7> python 1c.py
Enter s1: abaaba
Enter s2: babbab
LCS:  baba
```

Date: 14.04.24 Reg No: 3122225001127
ExNo: 7 Name: Shaun Allan H

Qn2:

2. In Domino Solitaire, you have a grid with two rows and many columns. Each square in the grid contains an integer. You are given a supply of rectangular 2×1 tiles, each of which exactly covers two adjacent squares of the grid. You have to place tiles to cover all the squares in the grid such that each tile covers two squares and no pair of tiles overlap. The score for a tile is the difference between the bigger and the smaller number that are covered by the tile. The aim of the game is to maximize the sum of the scores of all the tiles. Here is an example of a grid, along with two different tilings and their scores.

8	6	2	3
9	7	1	2

Tiling 1

8	6	2	3
9	7	1	2

Score 12

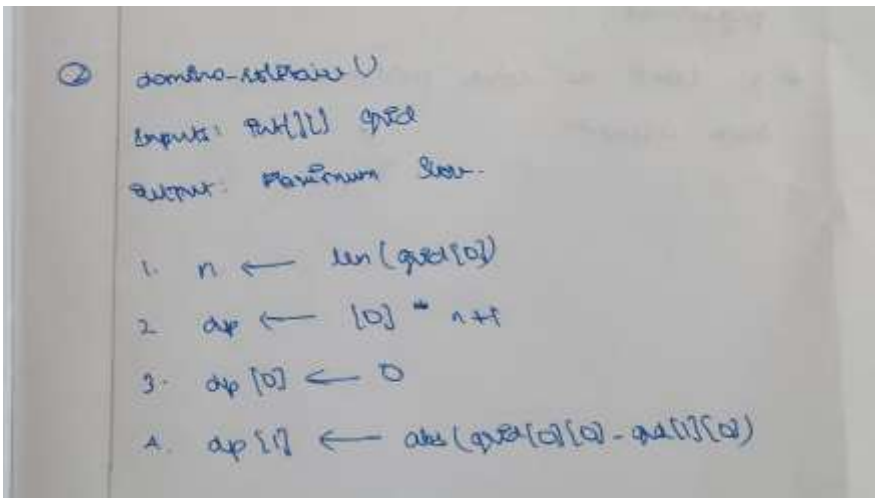
Tiling 2

8	6	2	3
9	7	1	2

Score 6

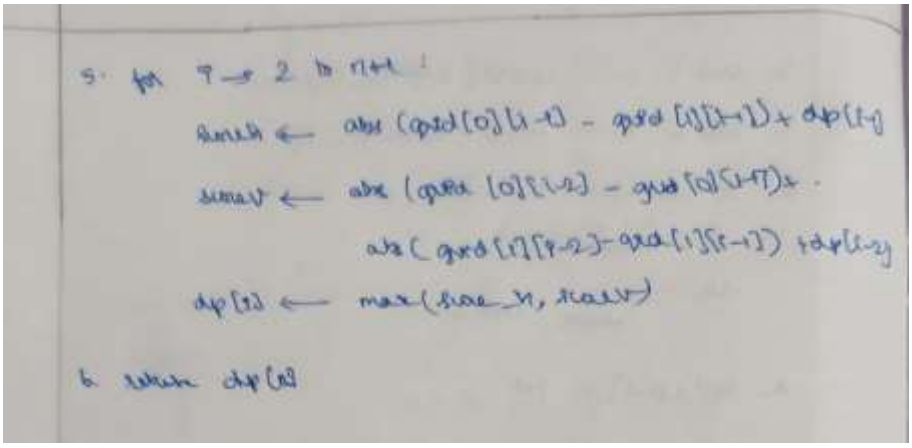
The score for Tiling 1 is $12 = (9 - 8) + (6 - 2) + (7 - 1) + (3 - 2)$ while the score for Tiling 2 is $6 = (8 - 6) + (9 - 7) + (3 - 2) + (2 - 1)$. There are other tilings possible for this grid, but you can verify that Tiling 1 has the maximum score among all tilings. Your task is to read the grid of numbers and compute the maximum score that can be achieved by any tiling of the grid.

Psuedo Code:



Date: 14.04.24 Reg No: 3122225001127

ExNo: 7 Name: Shaun Allan H



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))
```

Date: 14.04.24 Reg No: 3122225001127
ExNo: 7 Name: Shaun Allan H

Output:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment7> python 2.py
Grid:
[8, 6, 2, 3]
[9, 7, 1, 2]
Maximum score: 12
```

Learning Outcomes:

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