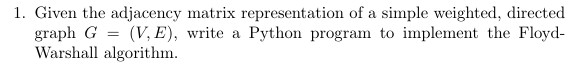
**ASSIGNMENT – 8**

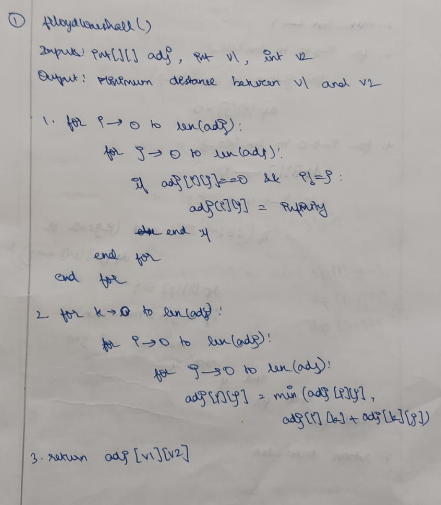
**AIM:**

To solve and implement the given problems using Dynamic Programming

**Qn1:**

****

**Psuedo Code:**

****

**Source Code:**

def ffloydWarshall(adj, v1, v2):

    adj = [[float('inf') if adj[i][j]==0 and not i==j else adj[i][j] for j in range(len(adj))] for i in range(len(adj))]

    for k in range(len(adj)):

        for i in range(len(adj)):

            for j in range(len(adj)):

                adj[i][j] = min(adj[i][j], adj[i][k]+adj[k][j])

    return adj[v1][v2]

adj = [ [ 0, 4, 0, 0, 0, 0, 0, 8, 0 ],

        [ 4, 0, 8, 0, 0, 0, 0, 11, 0 ],

        [ 0, 8, 0, 7, 0, 4, 0, 0, 2 ],

        [ 0, 0, 7, 0, 9, 14, 0, 0, 0 ],

        [ 0, 0, 0, 9, 0, 10, 0, 0, 0 ],

        [ 0, 0, 4, 14, 10, 0, 2, 0, 0 ],

        [ 0, 0, 0, 0, 0, 2, 0, 1, 6 ],

        [ 8, 11, 0, 0, 0, 0, 1, 0, 7 ],

        [ 0, 0, 2, 0, 0, 0, 6, 7, 0 ] ]

print("ADJACENCY MATRIX\n")

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

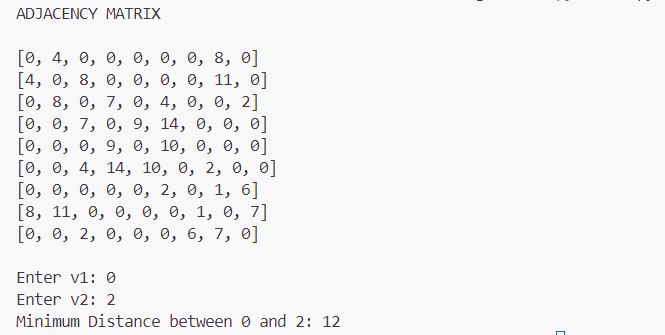
print()

v1 = int(input("Enter v1: "))

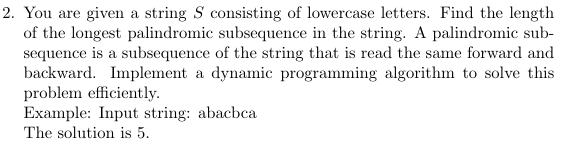
v2 = int(input("Enter v2: "))

print(f"Minimum Distance between {v1} and {v2}: " + str(ffloydWarshall(adj, v1, v2)))

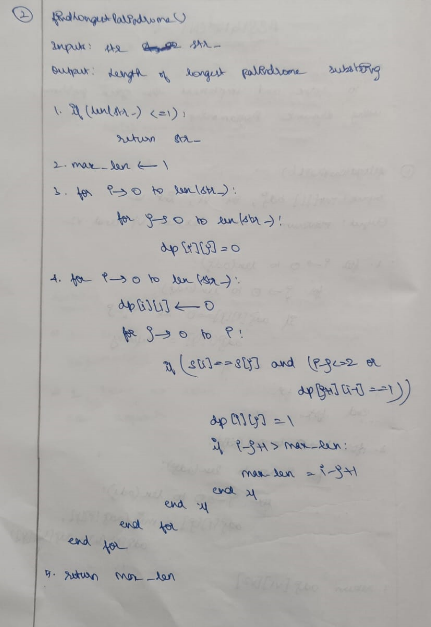
**Output:**

****

**Qn2:**

****

**Psuedo Code:**

****

**Source Code:**

def findLongestPalindrome(str\_):

    if len(str\_) <= 1:

        return str\_

    max\_len = 1

    dp = [[0 for j in range(len(str\_))] for i in range(len(str\_))]

    for i in range(len(str\_)):

        dp[i][i] = 1

        for j in range(i):

            if s[j] == s[i] and (i-j <= 2 or dp[j+1][i-1]):

                dp[j][i] = 1

                if i-j+1 > max\_len:

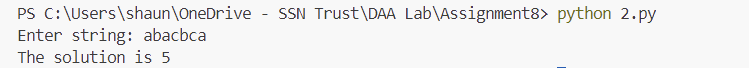
                    max\_len = i-j+1

    return max\_len

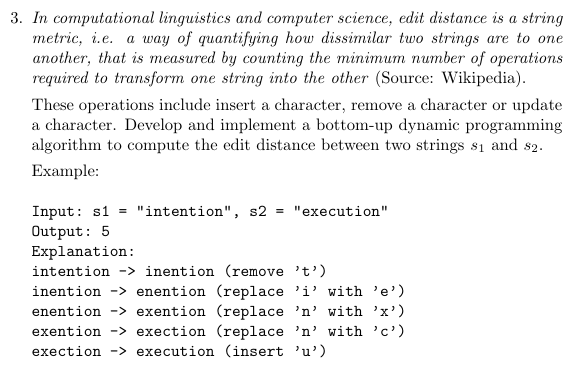
s = input("Enter string: ")

print("The solution is " + str(findLongestPalindrome(s)))

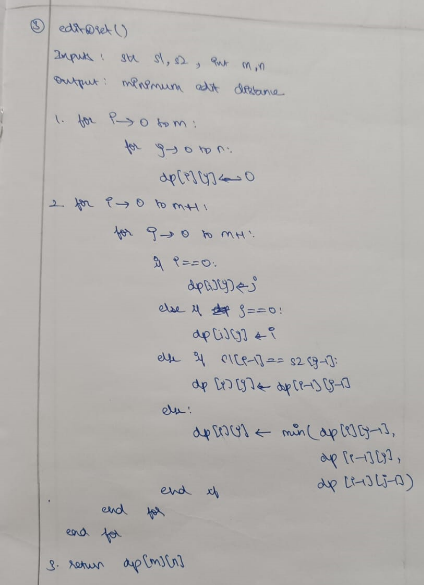
**Output:**

****

**Qn3:**

****

**Psuedo Code:**

****

**Source Code:**

def editDist(s1, s2, m, n):

    dp = [[0 for j in range(n+1)] for j in range(m+1)]

    for i in range(m+1):

        for j in range(n+1):

            if i==0:

                dp[i][j] = j

            elif j==0:

                dp[i][j] = i

            elif s1[i-1] == s2[j-1]:

                dp[i][j] = dp[i-1][j-1]

            else:

                dp[i][j] = 1 + min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1])

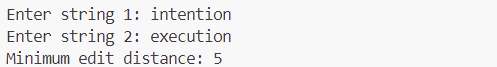
    return dp[m][n]

s1 = input("Enter string 1: ")

s2 = input("Enter string 2: ")

print("Minimum edit distance: " + str(editDist(s1, s2, len(s1), len(s2))))

**Output:**

****

**Learning Outcomes:**

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