Date: 29.04.24
ExNo: 8

Reg No: 3122225001127
Name: Shaun Allan H

ASSIGNMENT – 8

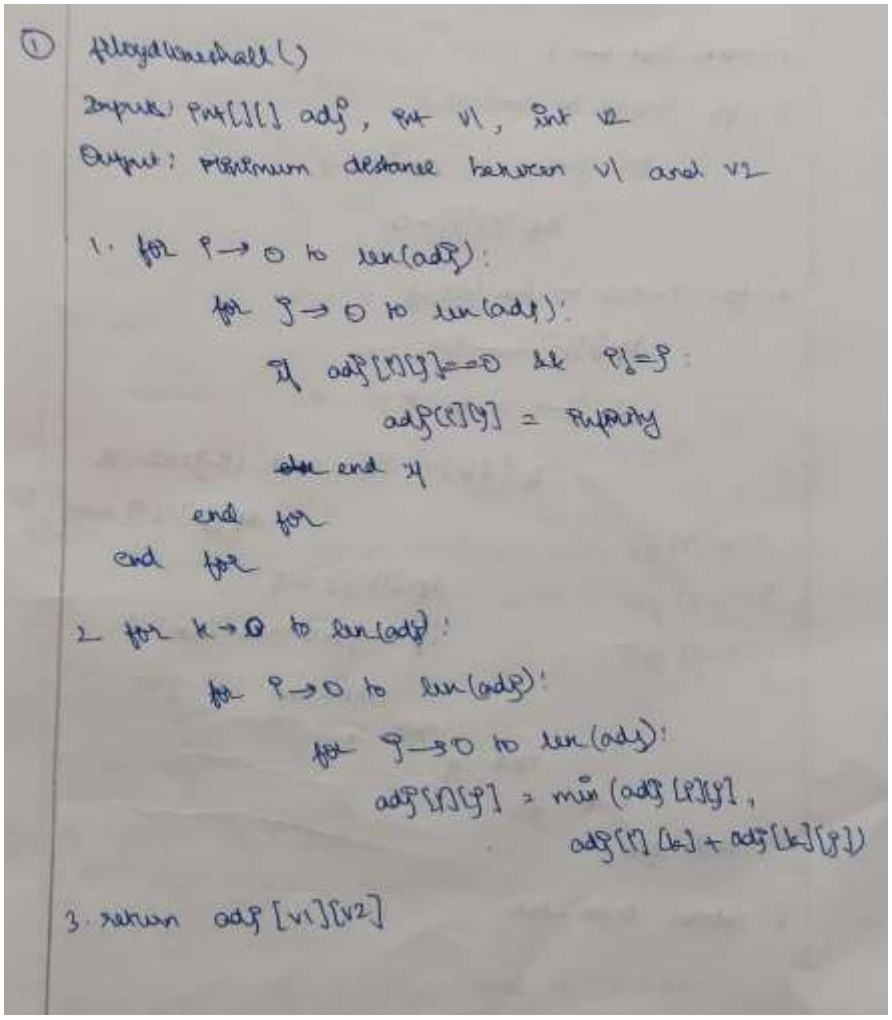
AIM:

To solve and implement the given problems using Dynamic Programming

Qn1:

1. Given the adjacency matrix representation of a simple weighted, directed graph $G = (V, E)$, write a Python program to implement the Floyd-Warshall algorithm.

Psuedo Code:



Date: 29.04.24
ExNo: 8

Reg No: 3122225001127
Name: Shaun Allan H

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:

```
ADJACENCY MATRIX

[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]

Enter v1: 0
Enter v2: 2
Minimum Distance between 0 and 2: 12
```

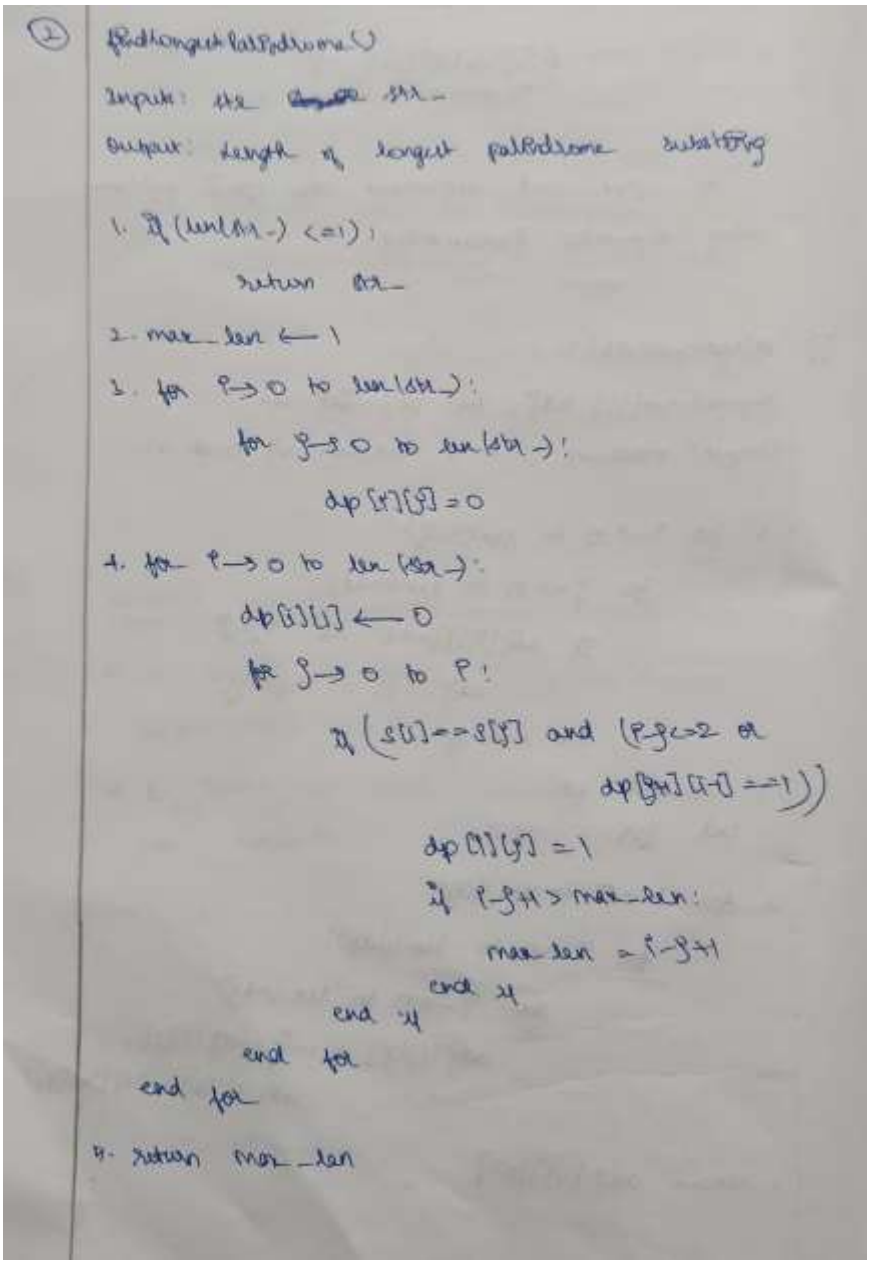
Date: 29.04.24
ExNo: 8

Reg No: 3122225001127
Name: Shaun Allan H

Qn2:

2. You are given a string S consisting of lowercase letters. Find the length of the longest palindromic subsequence in the string. A palindromic subsequence is a subsequence of the string that is read the same forward and backward. Implement a dynamic programming algorithm to solve this problem efficiently.
Example: Input string: abacbca
The solution is 5.

Psuedo Code:



Date: 29.04.24
ExNo: 8

Reg No: 3122225001127
Name: Shaun Allan H

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:

```
PS C:\Users\shaun\OneDrive - SSN Trust\DAA Lab\Assignment8> python 2.py
Enter string: abacbca
The solution is 5
```

Date: 29.04.24 Reg No: 3122225001127
ExNo: 8 Name: Shaun Allan H

Qn3:

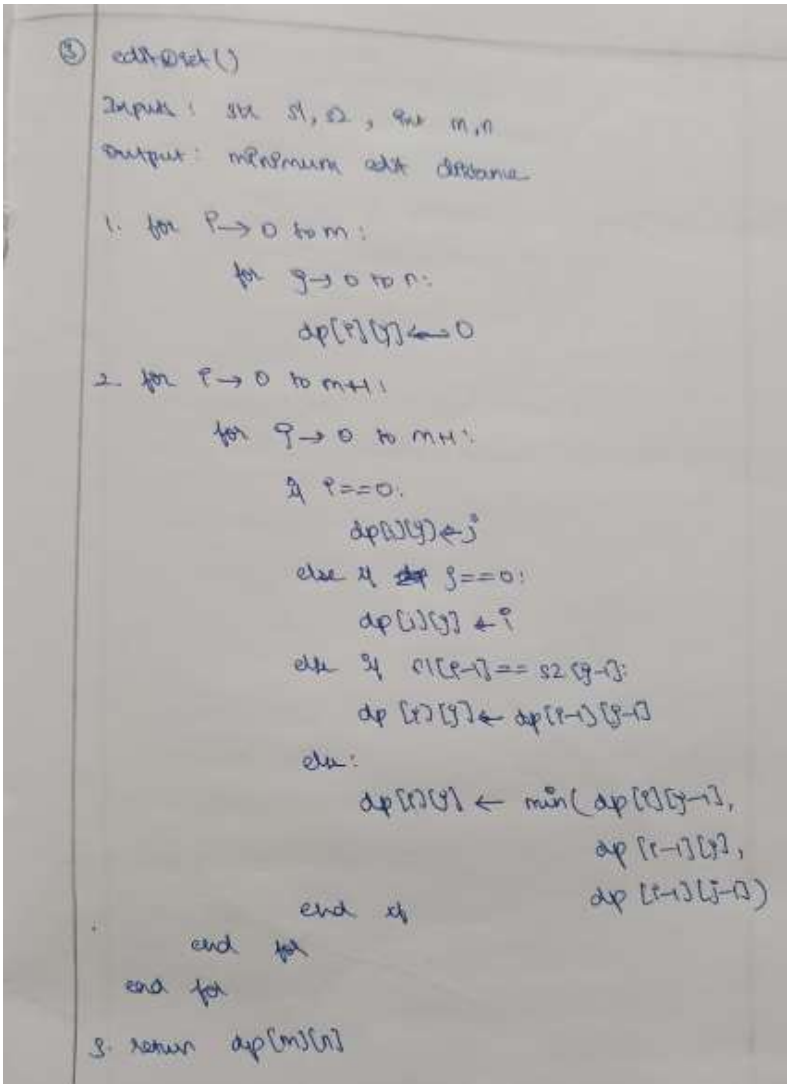
3. *In computational linguistics and computer science, edit distance is a string metric, i.e. a way of quantifying how dissimilar two strings are to one another, that is measured by counting the minimum number of operations required to transform one string into the other* (Source: Wikipedia).

   These operations include insert a character, remove a character or update a character. Develop and implement a bottom-up dynamic programming algorithm to compute the edit distance between two strings s_1 and s_2.

   Example:

```
Input: s1 = "intention", s2 = "execution"
Output: 5
Explanation:
intention -> inention (remove 't')
inention -> enention (replace 'i' with 'e')
enention -> exention (replace 'n' with 'x')
exention -> exection (replace 'n' with 'c')
exection -> execution (insert 'u')
```

Psuedo Code:



Date: 29.04.24 Reg No: 3122225001127
ExNo: 8 Name: Shaun Allan H

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:

```
Enter string 1: intention
Enter string 2: execution
Minimum edit distance: 5
```

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