Date: 02.05.2024
Ex. No: 8

Name: Mugilkrishna D U
Reg. No.: 3122225001073

UCS 2403 Design & Analysis of Algorithms

Assignment 8

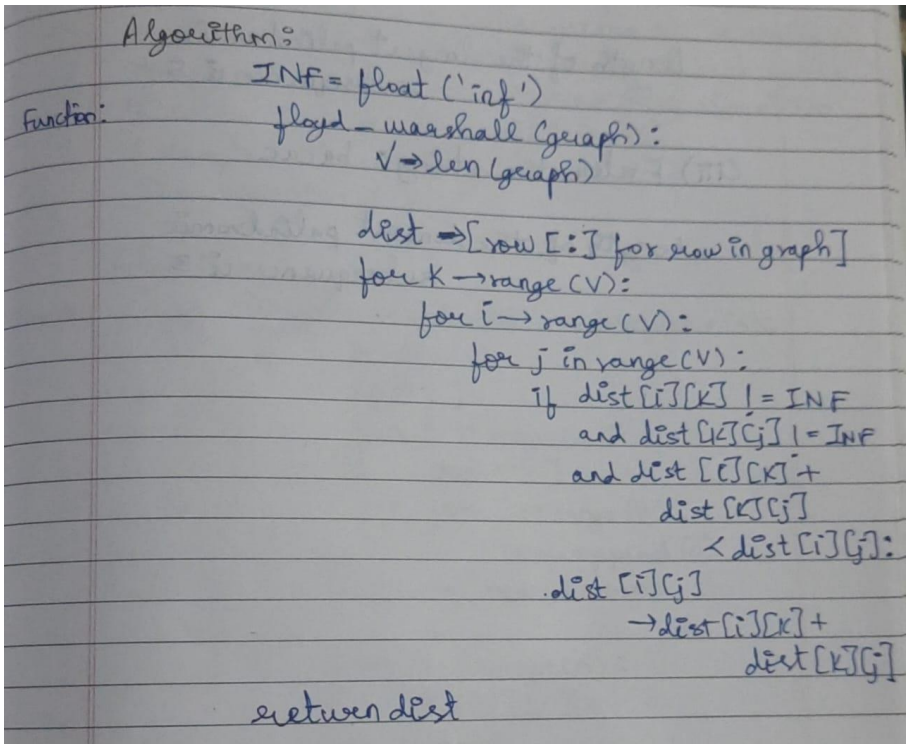
Date of Exercise: 02.05.2024

Aim: To gain understanding and proficiency on Dynamic Programming

Question 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.

Algorithm:



Code:

```python
INF = float('inf')

def floyd_warshall(graph):
    V = len(graph)

    dist = [row[:] for row in graph]

    # Floyd-Warshall algorithm
    for k in range(V):
        for i in range(V):
```

Name: Mugilkrishna D U
Reg. No.: 3122225001073

```
            for j in range(V):
                if dist[i][k] != INF and dist[k][j] != INF and dist[i][k] +
dist[k][j] < dist[i][j]:
                    dist[i][j] = dist[i][k] + dist[k][j]

    return dist

V = int(input("Enter the number of vertices: "))
print("-----ADJACENCY MATRIX-----")
graph = []
for i in range(V):
    row = []
    for j in range(V):
        weight_input = input(f"Enter weight from vertex {i} to vertex {j} (enter
'INF' for infinity): ")
        if weight_input.upper() == 'INF':
            weight = INF
        else:
            weight = int(weight_input)
        row.append(weight)
    graph.append(row)

result = floyd_warshall(graph)

print("-----SHORTEST PATH MATRIX-----")
for i in range(V):
    for j in range(V):
        if result[i][j] == INF:
            print('INF', end='\t')
        else:
            print(result[i][j], end='\t')
    print()
```

Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.1.py"
Enter the number of vertices: 5
-----ADJACENCY MATRIX-----
Enter weight from vertex 0 to vertex 0 (enter 'INF' for infinity): 0
Enter weight from vertex 0 to vertex 1 (enter 'INF' for infinity): 4
Enter weight from vertex 0 to vertex 2 (enter 'INF' for infinity): INF
Enter weight from vertex 0 to vertex 3 (enter 'INF' for infinity): 5
Enter weight from vertex 0 to vertex 4 (enter 'INF' for infinity): INF
Enter weight from vertex 1 to vertex 0 (enter 'INF' for infinity): INF
Enter weight from vertex 1 to vertex 1 (enter 'INF' for infinity): 0
Enter weight from vertex 1 to vertex 2 (enter 'INF' for infinity): 1
Enter weight from vertex 1 to vertex 3 (enter 'INF' for infinity): INF
Enter weight from vertex 1 to vertex 4 (enter 'INF' for infinity): 6
Enter weight from vertex 2 to vertex 0 (enter 'INF' for infinity): 2
Enter weight from vertex 2 to vertex 1 (enter 'INF' for infinity): INF
Enter weight from vertex 2 to vertex 2 (enter 'INF' for infinity): 0
Enter weight from vertex 2 to vertex 3 (enter 'INF' for infinity): 3
Enter weight from vertex 2 to vertex 4 (enter 'INF' for infinity): INF
Enter weight from vertex 3 to vertex 0 (enter 'INF' for infinity): INF
Enter weight from vertex 3 to vertex 1 (enter 'INF' for infinity): INF
Enter weight from vertex 3 to vertex 2 (enter 'INF' for infinity): 1
Enter weight from vertex 3 to vertex 3 (enter 'INF' for infinity): 0
Enter weight from vertex 3 to vertex 4 (enter 'INF' for infinity): 2
Enter weight from vertex 4 to vertex 0 (enter 'INF' for infinity): 1
Enter weight from vertex 4 to vertex 1 (enter 'INF' for infinity): INF
Enter weight from vertex 4 to vertex 2 (enter 'INF' for infinity): INF
Enter weight from vertex 4 to vertex 3 (enter 'INF' for infinity): 4
Enter weight from vertex 4 to vertex 4 (enter 'INF' for infinity): 0
-----SHORTEST PATH MATRIX-----
0       4       5       5       7
3       0       1       4       6
2       6       0       3       5
3       7       1       0       2
1       5       5       4       0
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

Time Complexity:

The Time Complexity of the Floyd Warshall algorithm is O(V^3), where V is the number of vertices in the graph with E number of edges (G = (V, E))

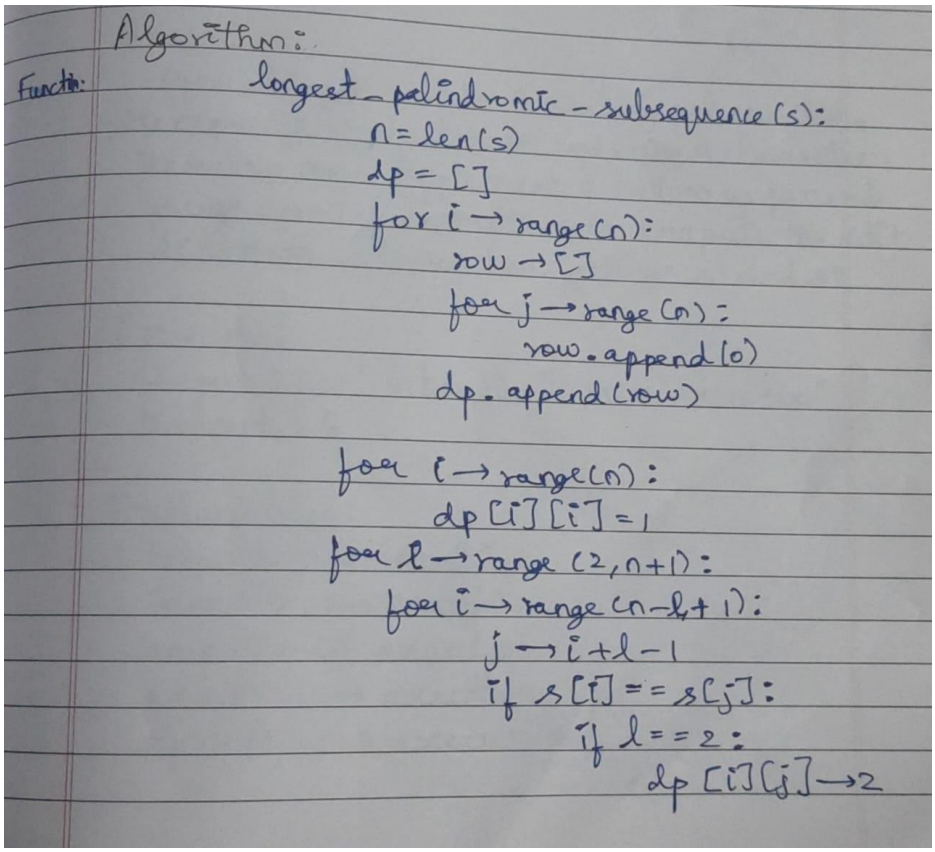
Question 2:

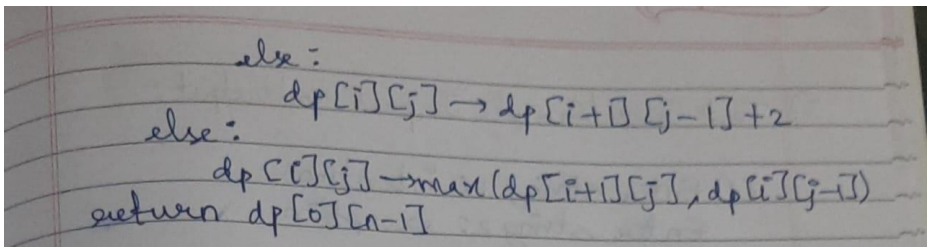
You are given a string S consisting of lowercase letters. Find the length of the longest palindromic subsequence in the string. A palindromic sub-sequence is a subsequence of the string that is read the same forward and backward. Implement a dynamic programming algorithm to solve this problem efficiently.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

Date: 02.05.2024
Ex. No: 8

Name: Mugilkrishna D U
Reg. No.: 3122225001073

Algorithm:





Code:

```
def longest_palindromic_subsequence(s):
    n = len(s)
    dp = []

    for i in range(n):
        row = []
        for j in range(n):
            row.append(0)
        dp.append(row)

    for i in range(n):
        dp[i][i] = 1
```

```
    for l in range(2, n + 1):
        for i in range(n - l + 1):
            j = i + l - 1
            if s[i] == s[j]:
                if l == 2:
                    dp[i][j] = 2
                else:
                    dp[i][j] = dp[i + 1][j - 1] + 2
            else:
                dp[i][j] = max(dp[i + 1][j], dp[i][j - 1])
    return dp[0][n - 1]

string = str(input("Enter the string in lowercase letters: "))
print("Length of the longest palindromic subsequence:",
longest_palindromic_subsequence(string))
```

Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.2.py"
Enter the string in lowercase letters: abacbca
Length of the longest palindromic subsequence: 5
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.2.py"
Enter the string in lowercase letters: ababba
Length of the longest palindromic subsequence: 5
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.2.py"
Enter the string in lowercase letters: bacac
Length of the longest palindromic subsequence: 3
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

Time Complexity:

The Time Complexity for the longest palindromic sequence problem is O(n^2)

Question 3:

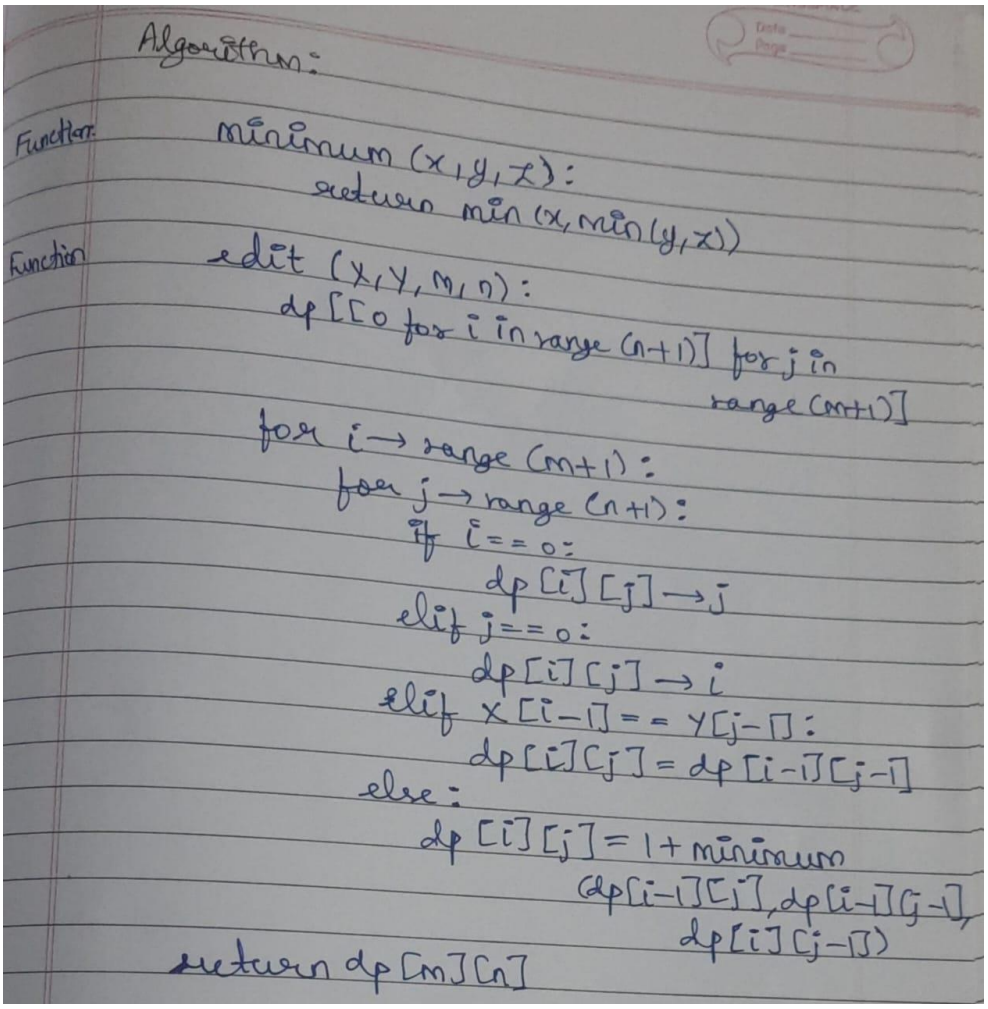
In computational linguistics and computer science, edit distance is a string metric, i.e. a way of quantifying how dissimilar two strings are to one another, that is measured by counting the minimum number of operations required to transform one string into the other (Source: Wikipedia).

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 s1 and s2.
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'

Algorithm:



Code:

```
def minimum(x,y,z):
    return min(x,min(y,z))

def edit(X,Y,m,n):
    dp=[[0 for i 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 X[i-1] == Y[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + minimum(dp[i-1][j], dp[i-1][j-1], dp[i][j-1])
    return dp[m][n]

X = input("Enter string 1: ")
Y = input("Enter string 2: ")
print("The number of operations to edit the strings is ",edit(X, Y, len(X),
len(Y)))
```

Output:

```
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.3.py"
Enter string 1: intention
Enter string 2: exception
The number of operations to edit the strings is  4
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.3.py"
Enter string 1: intention
Enter string 2: execution
The number of operations to edit the strings is  5
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode> & "C:/Users/Mugilkrishna D U/AppData/Local
/Programs/Python/Python311/python.exe" "c:/Users/Mugilkrishna D U/OneDrive/Desktop/My Files/SSN/SEM4/DESIG
N AND ANALYSIS OF ALGORITHMS/LAB/8.3.py"
Enter string 1: inception
Enter string 2: execution
The number of operations to edit the strings is  5
PS C:\Users\Mugilkrishna D U\OneDrive\Desktop\My Files\.vscode>
```

Time Complexity:

The Time Complexity for computing the edit distance between two strings is O(m*n), where m and n are the lengths of the strings

Learning Outcome:

Upon completing this exercise, I have understood the applications of Dynamic Programming and it's various uses for solving problems in an effective manner. I have now learnt to implement the Floyd-Warshall algorithm. I have also understood how to find the longest palindromic subsequence length and how to compute the edit distance between two strings using Dynamic Programming