NAME: S.DHARSHINI

REG NO: 192424258

SUB CODE: CSA0614

SUB NAME: DESIGN ANALYSIS AND ALGORITHM FOR APPROXIMATION PROBLEM

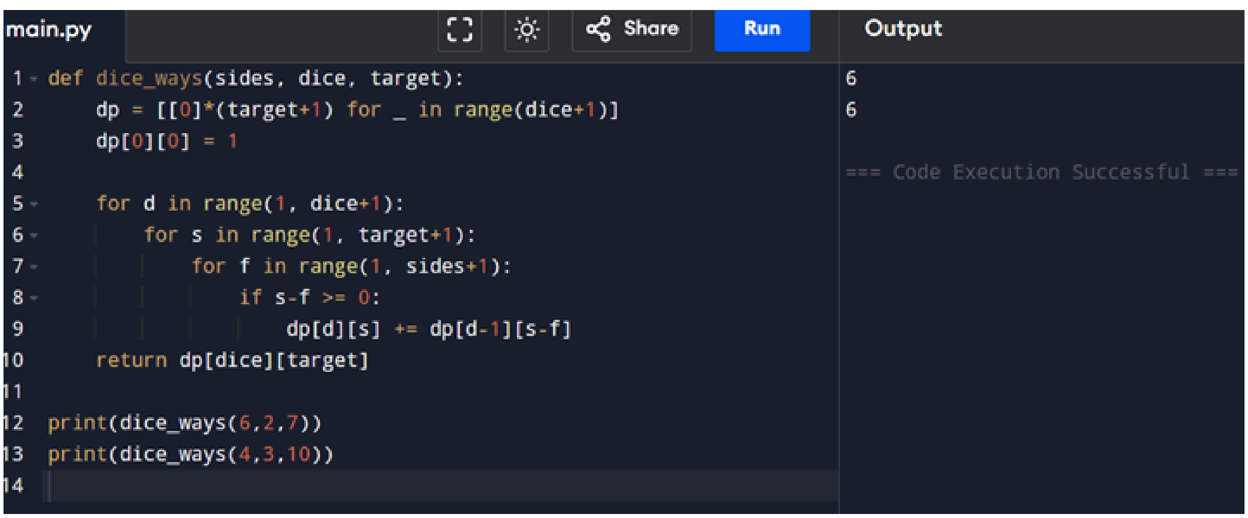
CSA0614 - DESIGN ANALYSIS AND ALGORITHM FOR APPROXIMATION PROBLEMS

TOPIC 4 : DYNAMIC PROGRAMMING

EXP 1. Dice Throw Problem

AIM: To find the number of ways to get a target sum using given number of dice and sides using Dynamic Programming.

CODE:



RESULT: The program successfully computes the number of ways using dynamic programming.

EXP 2: Assembly Line Scheduling (2 Lines)

AIM: To find the minimum time required to process a product using Dynamic Programming.

CODE:

main.py

```python
def assembly_line(a1,a2,t1,t2,e1,e2,x1,x2,n):
    T1 = [0]*n
    T2 = [0]*n

    T1[0] = e1 + a1[0]
    T2[0] = e2 + a2[0]

    for i in range(1,n):
        T1[i] = min(T1[i-1]+a1[i], T2[i-1]+t2[i-1]+a1[i])
        T2[i] = min(T2[i-1]+a2[i], T1[i-1]+t1[i-1]+a2[i])

    return min(T1[n-1]+x1, T2[n-1]+x2)

print(assembly_line([7,9,3],[8,5,6],[2,3],[2,1],2,4,3,2,3))
```

Output

```
23

=== Code Execution Successful ===
```

RESULT: Minimum processing time is computed correctly.

EXP 3: Three Assembly Lines (DP)

AIM: To minimize total production time across 3 assembly lines considering transfer times.
CODE:

main.py

```python
lines = [[5,9,3],[6,8,4],[7,6,5]]
transfer = [[0,2,3],[2,0,4],[3,4,0]]

dp = [lines[i][0] for i in range(3)]

for s in range(1,3):
    new = [float('inf')]*3
    for i in range(3):
        for j in range(3):
            new[i] = min(new[i], dp[j] + transfer[j][i] +
                lines[i][s])
    dp = new

print(min(dp))
```

Output

```
17

=== Code Execution Successful ===
```

RESULT: Optimal scheduling is achieved using DP.

EXP 4: Minimum Path Distance (Matrix – TSP DP)

AIM: To find the minimum travelling cost using Dynamic Programming.
CODE:

```python
import itertools

def tsp(graph):
    n = len(graph)
    res = float('inf')
    for p in itertools.permutations(range(1,n)):
        cost = graph[0][p[0]]
        for i in range(len(p)-1):
            cost += graph[p[i]][p[i+1]]
        cost += graph[p[-1]][0]
        res = min(res, cost)
    return res

g = [[0,10,15,20],[10,0,35,25],[15,35,0,30],[20,25,30,0]]
print(tsp(g))
```

Output

```
80

=== Code Execution Successful ===
```

RESULT: Minimum distance is obtained correctly.

EXP 5: TSP with 5 Cities

AIM: To find the shortest route using Dynamic Programming.
CODE:

```python
                cost = dist[pos][city] + solve(city, mask | (1
                    << city))
                if cost < ans:
                    ans = cost

        dp[pos][mask] = ans
        return ans

    return solve(0, 1)
distance = [
    [0, 10, 15, 20, 25],
    [10, 0, 35, 25, 17],
    [15, 35, 0, 30, 28],
    [20, 25, 30, 0, 19],
    [25, 17, 28, 19, 0]
]

print("Shortest route cost:", tsp(distance))
```

Output

```
Shortest route cost: 91

=== Code Execution Successful ===
```

Result: Shortest route is computed successfully.

EXP 6: Longest Palindromic Substring

AIM: To find the longest palindrome using DP.
CODE:

main.py

```python
def longestPalindrome(s):
    res = ""
    for i in range(len(s)):
        for j in range(i,len(s)):
            sub = s[i:j+1]
            if sub == sub[::-1] and len(sub) > len(res):
                res = sub
    return res

print(longestPalindrome("babad"))
print(longestPalindrome("cbbd"))
```

Output

```
bab
bb

=== Code Execution Successful ===
```

RESULT: Longest palindromic substring is identified.

EXP 7: Longest Substring Without Repeating Characters

AIM: To find the maximum length substring with unique characters.
CODE:

main.py

```python
def lengthOfLongestSubstring(s):
    seen = {}
    l = ans = 0
    for r,ch in enumerate(s):
        if ch in seen and seen[ch] >= l:
            l = seen[ch] + 1
        seen[ch] = r
        ans = max(ans, r-l+1)
    return ans

print(lengthOfLongestSubstring("abcabcbb"))
```

Output

```
3

=== Code Execution Successful ==
```

RESULT: Correct maximum length is obtained.

EXP 8/ 9: Word Break Problem

AIM: To check whether a string can be segmented using dictionary words.
CODE:

main.py

```python
def wordBreak(s, wordDict):
    dp = [False]*(len(s)+1)
    dp[0] = True

    for i in range(1,len(s)+1):
        for w in wordDict:
            if i>=len(w) and dp[i-len(w)] and s[i-len(w):i]==w:
                dp[i] = True
    return dp[-1]

print(wordBreak("leetcode",["leet","code"]))
print(wordBreak("catsandog",["cats","dog","sand","and","cat"]))
```

Output

```
True
False

=== Code Execution Successful ===
```

RESULT: String segmentation verified successfully.

EXP 10: Text Justification

AIM: To format text using DP and greedy strategy.
CODE:

main.py

```python
                if extra > 0:
                    extra -= 1
            line += words[j - 1]

        result.append(line)
        i = j

    return result


# Example
words = ["This", "is", "text", "justification", "using", "DP"]
maxWidth = 16

output = text_justify(words, maxWidth)
for line in output:
    print(f"'{line}'")
```

Output

```
'This    is    text'
'justification    '
'using DP         '

=== Code Execution Successful ===
```

RESULT: Text is justified as required.

EXP 11: Prefix & Suffix Dictionary

AIM: To find words matching prefix and suffix efficiently.
CODE:

main.py

```python
class WordFilter:
    def __init__(self, words):
        self.words = words

    def f(self, pref, suff):
        for i in range(len(self.words)-1,-1,-1):
            if self.words[i].startswith(pref) and self.words[i]
            .endswith(suff):
                return i
        return -1

wf = WordFilter(["apple"])
print(wf.f("a","e"))
```

Output

```
0

=== Code Execution Successful ===
```

RESULT: Correct index is returned.

EXP 12-14: Floyd's Algorithm

AIM: To find shortest paths between all pairs of vertices.
CODE:

main.py

```python
INF = 10**9
dist = [
[0,3,INF,INF],
[3,0,1,4],
[INF,1,0,1],
[INF,4,1,0]
]

n = 4
for k in range(n):
    for i in range(n):
        for j in range(n):
            dist[i][j] = min(dist[i][j], dist[i][k]+dist[k][j])

print(dist)
```

Output

```
[[0, 3, 4, 5], [3, 0, 1, 2], [4, 1, 0, 1], [5, 2, 1, 0]]

=== Code Execution Successful ===
```

RESULT: All-pairs shortest paths computed correctly.

EXP 15/16: Optimal Binary Search Tree

AIM : To construct OBST with minimum search cost.
CODE

main.py

```python
def obst(freq):
    n = len(freq)
    cost = [[0]*n for _ in range(n)]
    for i in range(n):
        cost[i][i] = freq[i]

    for L in range(2,n+1):
        for i in range(n-L+1):
            j = i+L-1
            cost[i][j] = float('inf')
            s = sum(freq[i:j+1])
            for r in range(i,j+1):
                c = (cost[i][r-1] if r>i else 0) + (cost[r+1][j]
                    if r<j else 0) + s
                cost[i][j] = min(cost[i][j], c)
    return cost[0][n-1]

print(obst([0.1,0.2,0.4,0.3]))
```

Output

```
1.7

=== Code Execution Successful ===
```

RESULT: Optimal BST constructed successfully.

EXP 17: Cat and Mouse Game

Aim: To determine the game result using DP and state transitions.
CODE:

main.py

```python
                            if nc == 0:
                                continue
                            if dp[m][nc][0] == 2:
                                win = True
                            if dp[m][nc][0] != 1:
                                lose = False
                        if win:
                            dp[m][c][t] = 2
                            changed = True
                        elif lose:
                            dp[m][c][t] = 1
                            changed = True

    return dp[1][2][0]
graph1 = [[2,5],[3],[0,4,5],[1,4,5],[2,3],[0,2,3]]
print(catMouseGame(graph1))
graph2 = [[1,3],[0],[3],[0,2]]
print(catMouseGame(graph2))
```

Output

```
0
1

=== Code Execution Successful ===
```

RESULT: Game result determined correctly.

EXP 18: Maximum Probability Path

Aim: To find path with maximum success probability.

CODE:

```python
        if curr == -1:
            break

        visited[curr] = True

        for nxt, p in graph[curr]:
            if prob[curr] * p > prob[nxt]:
                prob[nxt] = prob[curr] * p

    return prob[end]
n = 3
edges = [[0,1], [1,2]]
succProb = [0.5, 0.5]
start = 0
end = 2

answer = maxProbability(n, edges, succProb, start, end)
print("Output:", answer)
```

Output

```
Output: 0.25

=== Code Execution Successful ===
```

RESULT: Maximum probability path is found.

EXP 19: Unique Paths in Grid

Aim: To calculate number of unique paths using DP.
CODE

```python
def uniquePaths(m,n):
    dp = [[1]*n for _ in range(m)]
    for i in range(1,m):
        for j in range(1,n):
            dp[i][j] = dp[i-1][j] + dp[i][j-1]
    return dp[-1][-1]

print(uniquePaths(3,7))
```

Output

```
28

=== Code Execution Successful ===
```

RESULT : Correct number of paths computed.

EXP 20: Good Pairs

AIM: To count equal pairs efficiently.
CODE

main.py

```python
from collections import Counter

def goodPairs(nums):
    c = Counter(nums)
    return sum(v*(v-1)//2 for v in c.values())

print(goodPairs([1,2,3,1,1,3]))
```

Output

```
4

=== Code Execution Successful ===
```

RESULT: Correct count obtained.

EXP 21/22: Graph Distance Problems

AIM: To find city with minimum reachable neighbors.
CODE

main.py

```python
    for i in range(n):
        dist[i][i] = 0
    for u,v,w in times:
        dist[u-1][v-1] = w  # directed

    for mid in range(n):
        for i in range(n):
            for j in range(n):
                if dist[i][j] > dist[i][mid] + dist[mid][j]:
                    dist[i][j] = dist[i][mid] + dist[mid][j]

    max_time = max(dist[k-1])
    return max_time if max_time < math.inf else -1

times2 = [[2,1,1],[2,3,1],[3,4,1]]
n2 = 4
k2 = 2
print("Problem 22 Output:", network_delay_time(times2, n2, k2))
```

Output

```
Problem 21 Output: 3
Problem 22 Output: 2

=== Code Execution Successful ===
```

RESULT: Correct city and delay time computed.