UNIVERSITY INSTITUTE of COMPUTING
Asia's Fastest Growing University

Worksheet- 5

Student Name: Harsh **UID:** 24MCA20045

Branch: MCA **Section/Group:** 1(A)

Semester: 2 **Date of Performance:**02-04-2025

Subject Name: Design and Analysis of Algorithms Lab **Subject Code:** 24CAP-612

Q. a. Implement 0/1 Knapsack problem using dynamic programming.

1. Aim/Overview of the practical:

To implement the 0/1 Knapsack problem using dynamic programming to find the maximum value that can be obtained without exceeding the weight limit.

2. Task to be done

1. Define the problem and constraints.
2. Implement the DP approach using a 2D table (Bottom-up approach).
3. Fill the table based on item selection conditions.
4. Retrieve the maximum value stored in the last cell of the table.
5. Optimize space complexity using a 1D array if needed.

3. Algorithm:

1. Create a table to store the maximum value for each weight limit.
2. Iterate through each item and check if it can be included.
3. If the item is too heavy, keep the previous best value.
4. If it fits, take the maximum of excluding or including it.
5. The final table value gives the maximum value that can be carried.

4. Code for experiment/practical:

```
def knapsack (W, n, items):
    dp = {}

    for i in range (n + 1):
        for j in range (W + 1):
```

```
            dp[(i, j)] = 0

    for i in range (1, n + 1):
        weight, value = items [i - 1]
        for j in range (W + 1):
            if weight > j:
                dp[(i, j)] = dp[(i - 1, j)]
            else:
                dp[(i, j)] = max(dp[(i - 1, j)], value + dp[(i - 1, j - weight)])

    return dp[(n, W)]

n = int (input ("Enter the number of items: "))

items = tuple (tuple (map (int, input (f"Enter weight and value for item {i + 1}: ").split())) for i in range(n))

W = int (input ("Enter the maximum weight capacity of the knapsack: "))

max_value = knapsack (W, n, items)
print ("Maximum value in Knapsack:", max_value)
```

5. Result/Output/Writing Summary:

```
PS C:\VS code> & C:/Users/harsh/AppData/Local/Programs/Python/Python313/python.exe "c:/VS code/Python/DAA_Implementation.py"
Enter the number of items: 5
Enter weight and value for item 1: 2 10
Enter weight and value for item 2: 3 20
Enter weight and value for item 3: 4 30
Enter weight and value for item 4: 5 40
Enter weight and value for item 5: 6 50
Enter the maximum weight capacity of the knapsack: 7
Maximum value in Knapsack: 50
PS C:\VS code>
```

Q. b. Compute the transitive closure of a given directed graph using Warshall's algorithm.

1. Task To be Done:

- **Understand the Transitive Closure Concept**: Learn what it means to find the transitive closure of a directed graph, which involves determining if there is a path between each pair of vertices.
- **Implement Warshall's Algorithm**: Use Warshall's algorithm to calculate the transitive closure of a given directed graph. Warshall's algorithm is a classic dynamic programming approach to solve this problem.
- **Develop a Program**: Write a program that takes a directed graph as an adjacency matrix and computes its transitive closure using Warshall's algorithm.

2. Source Code:

```
def warshall(graph):
    n = len(graph)
    for k in range(n):
        for i in range(n):
            for j in range(n):
                graph[i][j] |= graph[i][k] and graph[k][j]
    return graph


# Example usage
graph = [
    [1, 1, 0, 1],
    [0, 1, 1, 0],
    [0, 0, 1, 1],
    [0, 0, 0, 1]
]
closure = warshall(graph)
print("Transitive closure of the graph:")
```

for row in closure:

print(row)

Output:

```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Prog
Transitive closure of the graph:
0 0 1 0
1 1 1 1
0 0 0 0
1 1 1 1

Process finished with exit code 0
```

5. Learning outcomes (What I have learnt):

- Gain a solid understanding of graph theory concepts, specifically transitive closure and how it relates to connectivity in directed graphs.
- Learn to implement and apply Warshall's algorithm for solving real-world graph problems.
- Improve skills in Java programming, especially in working with matrices and nested loops.
- Develop analytical skills in understanding the efficiency and application of dynamic programming to graph problems.

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance (Pre Lab Quiz)		5
2.	Worksheet		10
3.	Post Lab Quiz		5