CODE :

```
def knapSack_dp(m, W, P, n):
    """
    Solves the 0/1 Knapsack problem using Dynamic Programming (Tabulation).

    m: Knapsack capacity
    W: List of item weights
    P: List of item profits (values)
    n: Number of items
    """
    # Create a 2D DP table V[i][w] to store max profit for first i items and capacity w
    # (n+1 rows for items 0 to n, m+1 columns for weights 0 to m)
    V = [[0 for _ in range(m + 1)] for _ in range(n + 1)]

    # Fill the dp table using the recurrence relation
    # V[i][w] = max( V[i-1][w], P[i-1] + V[i-1][w - W[i-1]] )
    for i in range(1, n + 1):
        # Current item's profit and weight (using 0-based indexing for lists P and W)
        p_i = P[i-1]
        w_i = W[i-1]
        for w in range(m + 1):
            if w_i <= w:
                # Case 1: Item i is included
                profit_with_i = p_i + V[i-1][w - w_i]
                # Case 2: Item i is NOT included
                profit_without_i = V[i-1][w]
                V[i][w] = max(profit_without_i, profit_with_i)
            else:
                # Item i cannot be included (weight w_i is greater than current capacity w)
                V[i][w] = V[i-1][w]

    # Print the DP table (similar to the format in the document)
    print("DP Table (Item i vs. Capacity w):")
    # Header: Capacity w
    print("    | ", end="")
    for w in range(m + 1):
        print(f"{w:2}", end=" ")
    print("\n" + "---" * (m + 3))
    # Body: Item i
    for i in range(n + 1):
        print(f"{i:2} | ", end="")
        for w in range(m + 1):
            print(f"{V[i][w]:2}", end=" ")
        print()

    # Backtrack to find selected items
    selected = [0] * n
    current_w = m
    max_profit = V[n][m]

    for i in range(n, 0, -1):
```

```
        # Check if the profit came from the previous row (item i was NOT included)
        if V[i][current_w] == V[i-1][current_w]:
            # Item i was NOT included
            selected[i-1] = 0
        else:
            # Item i WAS included
            selected[i-1] = 1
            # Subtract item's weight and move to the state before including it
            current_w -= W[i-1]

    # Print selected items
    print("\n---")
    print("Sequence of Decisions (x1, x2, x3, x4):")
    print(f"Items: {selected} (1: included, 0: not included)")
    # Print maximum value
    print(f"Maximum Profit: {max_profit}")

    return max_profit

# Driver code

P = [1, 2, 5, 6] # Profits
W = [2, 3, 4, 5] # Weights
m = 8 # Knapsack capacity
n = len(P) # Number of items

knapSack_dp(m, W, P, n)
```

OUTPUT :

```
(base) kjcoemr@kjcoemr-HP-Pro-SFF-280-G9-Desktop-PC:~/Desktop/BE-15$ python
knapsack_dynamic_programming.py
DP Table (Item i vs. Capacity w):
   | 0  1  2  3  4  5  6  7  8
---------------------------------
 0 | 0  0  0  0  0  0  0  0  0
 1 | 0  0  1  1  1  1  1  1  1
 2 | 0  0  1  2  2  3  3  3  3
 3 | 0  0  1  2  5  5  6  7  7
 4 | 0  0  1  2  5  6  6  7  8

---
Sequence of Decisions (x1, x2, x3, x4):
Items: [0, 1, 0, 1] (1: included, 0: not included)
Maximum Profit: 8

(base) kjcoemr@kjcoemr-HP-Pro-SFF-280-G9-Desktop-PC:~/Desktop/BE-15$
```