Name: Vaishnavi Pravin Kolse
Class: BE - A
Roll No.37

Practical No.4

Aim: Write a program to solve a 0-1 Knapsack problem using dynamic programming or branch and bound strategy.

In [1]:
```
def knapsack_01(n, values, weights, W):
    # Create a 2D DP array to store the maximum values for each subproblem
    dp = [[0] * (W + 1) for _ in range(n + 1)]

    # Build the DP array
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                dp[i][w] = 0  # Base case: no items or knapsack capacity is 0
            elif weights[i - 1] <= w:
                # Take the item or leave it, whichever is more beneficial
                dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + v
            else:
                dp[i][w] = dp[i - 1][w]  # Skip the item if its weight is too

    # Backtrack to find the items that were selected
    selected_items = []
    i, w = n, W
    while i > 0 and w > 0:
        if dp[i][w] != dp[i - 1][w]:
            selected_items.append(i - 1)  # Add this item to the selected list
            w -= weights[i - 1]  # Reduce the remaining capacity
        i -= 1  # Move to the previous item

    return dp[n][W], selected_items

# Take input from the user
n = int(input("Enter the number of items: "))
values = list(map(int, input("Enter the values of the items separated by space
weights = list(map(int, input("Enter the weights of the items separated by spa
W = int(input("Enter the maximum capacity of the knapsack: "))

# Call the knapsack function
max_value, selected_items = knapsack_01(n, values, weights, W)

# Output the result
print("Maximum value:", max_value)
print("Selected items (0-indexed):", selected_items)
```

```
Enter the number of items: 4
Enter the values of the items separated by space: 3 4 5 6
Enter the weights of the items separated by space: 2 3 4 6
Enter the maximum capacity of the knapsack: 5
Maximum value: 7
Selected items (0-indexed): [1, 0]
```

In [ ]: