Dynamic Programming Approach

In [2]:
```
def knapsack_dp(weights, values, W):
    n = len(weights)
    dp = [[0 for _ in range(W + 1)] for _ in range(n + 1)]

    for i in range(1, n + 1):
        for w in range(1, W + 1):
            if weights[i-1] <= w:
                dp[i][w] = max(values[i-1] + dp[i-1][w-weights[i-1]], dp[i-1][w])
            else:
                dp[i][w] = dp[i-1][w]

    return dp[n][W]

# Example usage
weights = [1, 2, 3, 4]
values = [10, 20, 30, 40]
W = 5
print(f"Maximum value in knapsack (DP): {knapsack_dp(weights, values, W)}")
```

```
Maximum value in knapsack (DP): 50
```

Branch and Bound Approach

In [4]:
```
class KnapsackBranchAndBound:
    def __init__(self, weights, values, W):
        self.weights = weights
        self.values = values
        self.W = W
        self.n = len(weights)

    def bound(self, node, capacity):
        if node.weight >= capacity:
            return 0
        profit_bound = node.profit
        j = node.level + 1
        total_weight = node.weight

        while j < self.n and total_weight + self.weights[j] <= capacity:
            total_weight += self.weights[j]
            profit_bound += self.values[j]
            j += 1

        if j < self.n:
            profit_bound += (capacity - total_weight) * self.values[j] / self.weigh

        return profit_bound

    def knapsack(self):
        Q = []
        u = Node(-1, 0, 0, 0)
        Q.append(u)
        max_profit = 0

        while Q:
            u = Q.pop(0)
            if u.level == self.n - 1:
                continue

            v = Node(u.level + 1, u.profit + self.values[u.level + 1], u.weight + s
            v.bound = self.bound(v, self.W)

            if v.weight <= self.W and v.profit > max_profit:
                max_profit = v.profit

            if v.bound > max_profit:
                Q.append(v)

            v = Node(u.level + 1, u.profit, u.weight, 0)
            v.bound = self.bound(v, self.W)

            if v.bound > max_profit:
                Q.append(v)

        return max_profit

class Node:
    def __init__(self, level, profit, weight, bound):
        self.level = level
        self.profit = profit
        self.weight = weight
        self.bound = bound

# Example usage
weights = [1, 2, 3, 4]
values = [10, 20, 30, 40]
W = 5
knapsack_bnb = KnapsackBranchAndBound(weights, values, W)
print(f"Maximum value in knapsack (B&B): {knapsack_bnb.knapsack()}")
```

```
Maximum value in knapsack (B&B): 50
```