### Algorithm: Dynamic Programming (Knapsack Problem)

The knapsack problem is a problem in combinatorial optimization. Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

#### Initialize Data Structures:

* Use a 2D array to keep track of the maximum value for each weight.

#### Build the Solution:

* Iterate through each item and update the array based on whether the item is included or not.

#### Retrieve the Result:

* The value in the last cell of the array is the maximum value that can be obtained.

#### Implementation:

| **def** knapsack(weights: List[int], values: List[int], capacity: int) -> int:  n = len(weights)  dp = [[0 **for** \_ **in** range(capacity + 1)] **for** \_ **in** range(n + 1)]  **for** i **in** range(1, n + 1):  **for** w **in** range(1, capacity + 1):  **if** weights[i - 1] <= w:  dp[i][w] = max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1])  **else**:  dp[i][w] = dp[i - 1][w]  **return** dp[n][capacity]  *# Example usage:*  weights = [1, 3, 4, 5]  values = [1, 4, 5, 7]  capacity = 7  print(knapsack(weights, values, capacity)) *# Output: 9* |
| --- |

#### Explanation:

Initialize:

* dp: A 2D array initialized with 0.

Build the Solution:

* Iterate through each item and update the array based on whether the item is included or not.

Retrieve the Result:

### The value in the last cell of the array is the maximum value that can be obtained.