Count number of coins required to make a given value

Greedy algorithms do not necessarily give the optimal solution . I n the Link provided they have given the have used Dynamic programming Approach

Dynamic problem Breaks down the code to find the optimal Solution. It builds up a final solution using the sub problems

Steps for Coin Change Problem Using DP:

1. Create a table in which rows represent possible change amounts (0 to the target amount) and Columns represent different coin denominations.
2. Fill in the base cases as 0
3. Fill in the rest of the table:
    - For each change amount (row) and coin denomination (column):
        - Check if the coin is smaller than or equal to the change amount.
        - If it is, calculate the minimum number of coins needed for that change amount using two options:
            - Option 1: Use the current coin plus the minimum number of coins needed for the remaining change (change amount minus coin value).
            - Option 2: Don't use the current coin, and use the minimum number of coins from the previous row (same change amount, different coin).
        - Choose the option that results in the smaller number of coins and store it in the table.
4. Find the optimal solution:
    - The minimum number of coins for the target change amount is in the last row, last column of the table.
5. Backtrack to find the actual coins used:
    - Start from the last row, last column of the table.
    - Follow the decisions made in step 3 to trace back the coins used.

Example:

For denominations [1, 2, 5, 8, 10] and change 7, the DP table would look like this:

1	2	5	8	10	
0	0	0	0	0	0
1	1	1	1	1	1
2	2	1	1	1	1
3	3	2	1	1	1
4	4	2	2	1	1
5	5	3	1	1	1
6	6	3	2	1	1
7	7	3	2	2	1

The optimal solution is 2 coins (5 and 2), as found in the last row, last column.

Start from the last row, last column (7, 2).

- Check which option was chosen to fill that cell (in this case, Option 2, as it's smaller than Option 1).
- Follow Option 2, which refers to the previous row (6, 2).
- Repeat this process until you reach a base case (0 coins).
- The traced path reveals the coins used: 5 and 2.

- count() function:

    - Recursively explores different combinations of coins to make the sum.
    - Uses a 2D DP table (dp) to store previously calculated results, enhancing efficiency.
- Base cases:
    - If the sum is 0, there's 1 way to make it (using no coins).
    - If the number of coins is 0 or the sum is negative, there's no way to make it.
- Recursive calls:
    - Considers two options for each coin:
        - *Include* the current coin and explore making the remaining sum with the same set of coins.
        - *Exclude* the current coin and explore making the sum with the remaining coins.
- Memoization:
    - Stores calculated results in dp for reuse, avoiding redundant computations.

Scenarios:

Scenario 1: coins = [1, 2, 3], sum = 5

- Output: 5
- Explanation: 5 ways to make 5:
    - 1+1+1+1+1
    - 1+1+3
    - 1+3+1
    - 3+1+1
    - 2+3

Scenario 2: coins = [2, 5, 3, 6], sum = 10

- Output: 5
- Explanation: 5 ways to make 10:

    - 2+2+2+2+2
    - 2+3+5
    - 3+2+5
    - 5+2+3
    - 2+2+3+3

Scenario 3: coins = [1, 4], sum = 6

- Output: 2
- Explanation: 2 ways to make 6:
    - 1+1+1+1+1+1
    - 4+2