**Problem Statement**

You are working at the coupon counter at a fun fair and have an unlimited supply of coupons

with different denominations. When a customer comes to you to purchase coupons for a certain amount of money, you can give that customer any combination of coupons such that the total value equals the amount of money they gave you.

For example: If you have coupons with denominations of $8, $3, $2 and $1 and someone

wanted coupons for $3. You could give them coupons in the 3 different combinations: {1,1,1},

{1,2}, and {3}.

**Requirements:**

1. Formulate an efficient algorithm using Dynamic Programming to give the total number

of combinations in which you can return coupons to the customer for a given sum of

money.

2. Analyse the time complexity of your algorithm.

3. Implement the above problem statement using Python 3.7.

**Design:**

In order for the employ Dynamic Programming to give total number of combinations of coupons to the customer for a given sum of money, we should first ensure the problem has following property.

1. Optimal Substructure
2. Overlapping Subproblem

**Optimal Substructure**

A given problems has Optimal Substructure Property if optimal solution of the given problem can be obtained by using optimal solutions of its subproblems.

To count the total number of combinations, we can divide all set solutions into two sets.  
1) Solutions that do not contain mth coin (or Sm).  
2) Solutions that contain at least one Sm.  
Let count(S[], m, n) be the function to count the number of solutions, then it can be written as sum of count(S[], m-1, n) and count(S[], m, n-Sm).

Therefore, the problem has **optimal substructure property** as the problem can be solved using solutions to subproblems.

**Overlapping Subproblem**

Dynamic Programming is mainly used when solutions of same subproblems are needed again and again. In dynamic programming, computed solutions to subproblems are stored in a table so that these don’t have to be recomputed. So Dynamic Programming is not useful when there are no common (overlapping) subproblems because there is no point storing the solutions if they are not needed again.

Below recursive tree structure of the FUN fair problem shows that the total number of combinations can be obtained by solving sub problems and can be stored to reuse it for solving problems which is composed of sub problems.

C() --> count()

C({1,2,3}, 5)

/ \

/ \

C({1,2,3}, 2) C({1,2}, 5)

/ \ / \

/ \ / \

C({1,2,3}, -1) C({1,2}, 2) C({1,2}, 3) C({1}, 5)

/ \ / \ / \

/ \ / \ / \

C({1,2},0) C({1},2) C({1,2},1) C({1},3) C({1}, 4) C({}, 5)

/ \ / \ /\ / \

/ \ / \ / \ / \

. . . . . . C({1}, 3) C({}, 4)

/ \

/ \

. .

There are following two different ways to store the values so that these values can be reused:  
a) Memoization (Top Down)  
b) Tabulation (Bottom Up)

**a) Memoization (Top Down):** The memorized program for a problem is similar to the recursive version with a small modification that it looks into a lookup table before computing solutions. We initialize a lookup array with all initial values as NIL. Whenever we need the solution to a subproblem, we first look into the lookup table. If the precomputed value is there then we return that value, otherwise, we calculate the value and put the result in the lookup table so that it can be reused later.

**b) Tabulation (Bottom Up):**The tabulated program for a given problem builds a table in bottom up fashion and returns the last entry from table. For example, for the same Fibonacci number, we first calculate fib(0) then fib(1) then fib(2) then fib(3) and so on. So literally, we are building the solutions of subproblems bottom-up.

In general for the given problem, we can provide solution with three different algorithms,

Recursive Approach

Memoization (Top Down) Approach

Tabulation (Bottom Up) Approach

We have provided solution in all 3 approaches.

Time Complexity of the algorithm is as stated below,

|  |  |  |
| --- | --- | --- |
| S.No | Algorithm | Time Complexity |
| 1 | Recursive Approach | O(2^n) |
| 2 | **Tabulation (Bottom Up) Approach** | O(mn) where m is variety and n is purchase amount |
| 3 | Memoization (Top Down) Approach | O(n) where n is purchase amount. |