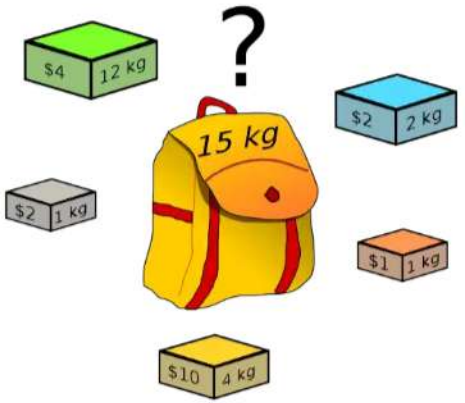
📂 2. 0/1 Knapsack Problem (GFG)

Space Optimization sol.ⁿ (DP PROBLEM)

Problem Statement:
*Given **N items** where each item has some **weight** and **profit** associated with it and also given a **bag** with **capacity W**, (**i.e., the bag can hold at most W weight in it**).*

Return Kya Karana Hai:
*The task is to put the items into the bag such that the **sum of profits** associated with them is the **maximum possible**.*

***Note:** The constraint here is we can either **put an item completely into the bag** or cannot put it at all*
(It is not possible to put a part of an item into the bag).

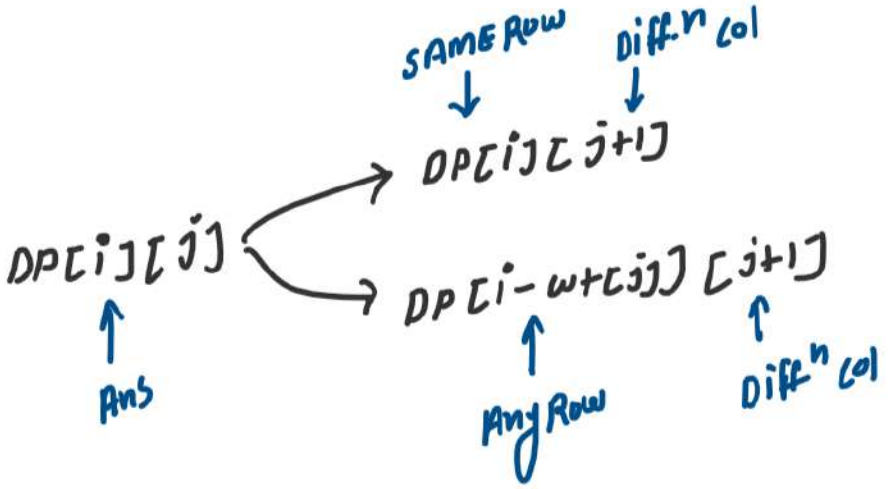


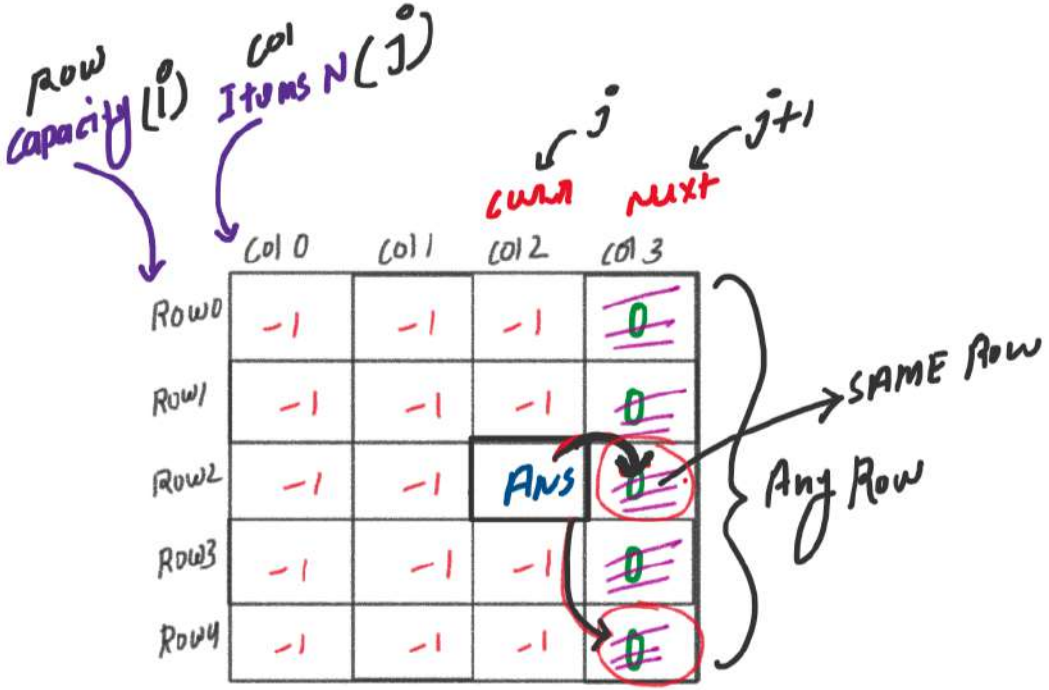
Knapsack Problem

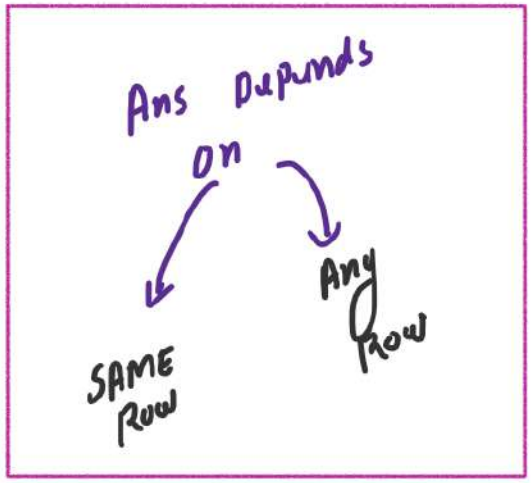
Approach 4: Space Optimization
Inclusive and Exclusive Pattern

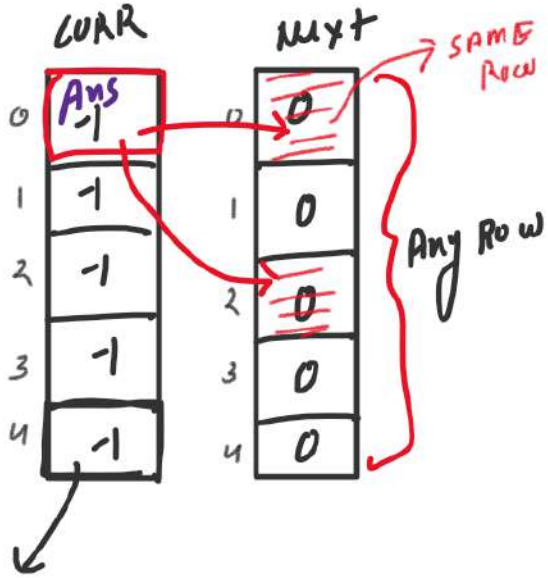
DRY RUN

Input:
N = 3, W = 4, weight[] = {4, 5, 1}, profit[] = {1, 2, 3}
Output: 3





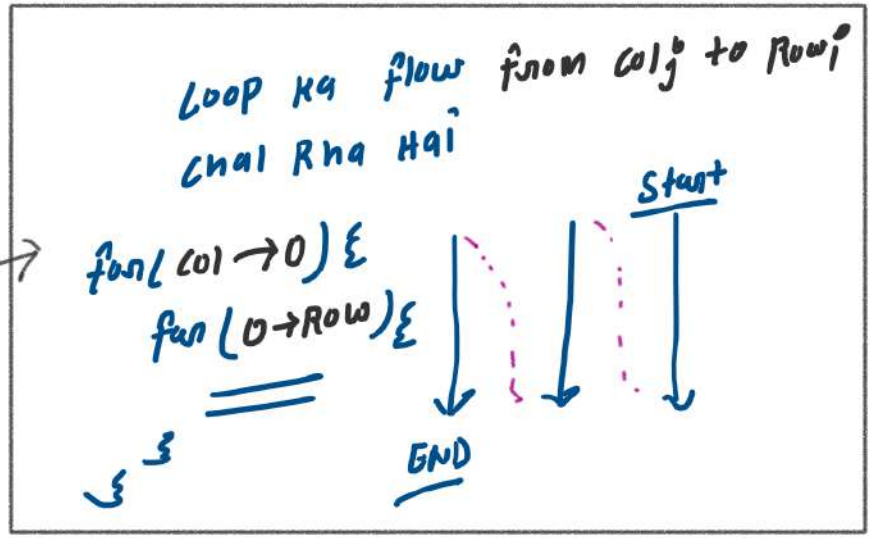




ARRAY CURR and Next ka size Capacity +1 Hona chaiye Kyunki har EK col par each Row ko fill kiya ja Rha hai.

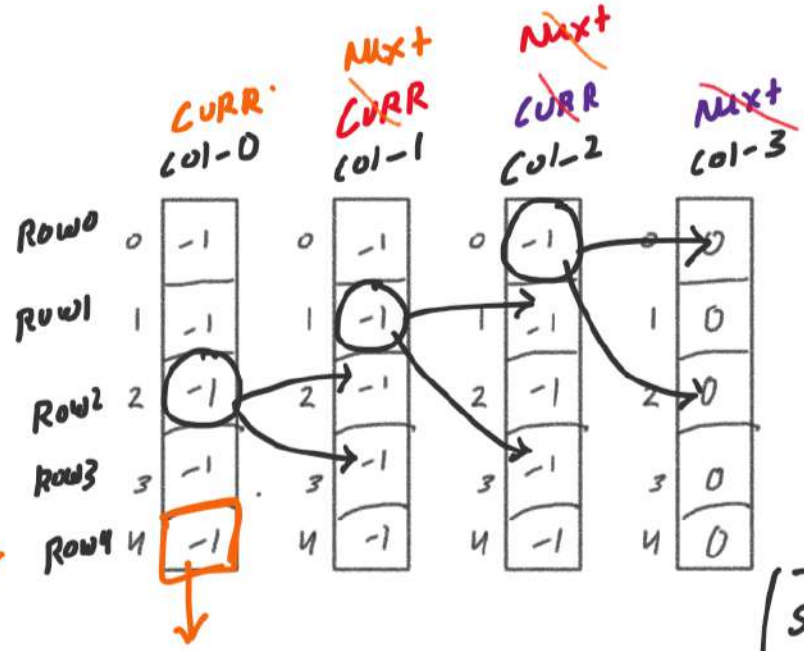
jis Hisab se Loop chal Rahe hai

Why Loop Intuition Change?



```cpp
// Problem 2: 0/1 Knapsack Problem (GFG)
// Approach 4: Space Optimization Approach

#include<iostream>
#include<vector>
using namespace std;

int solveUsingTabuSO(int capacity, int weight[], int profit[], int index, int n){
    vector<int> next(capacity+1, 0);
    vector<int> curr(capacity+1, -1);

    for(int col=n-1; col>=0; col--){
        for(int row=0; row<=capacity; row++){
            // Recursive relation (Inclusion or exclusion)
            int include = 0;
            if(weight[col] <= row){
                include = profit[col] + next[row - weight[col]];
            }
            int exclude = 0 + next[row];
            curr[row] = max(include, exclude);
        }
        // Shift Karna Bhool Jata hu
        next = curr;
    }
    // return ans
    return curr[capacity];
}

int main(){
    int capacity = 6;
    int n = 3;
    int weight[] = {1,2,3};
    int profit[] = {10,15,40};
    int index = 0;

    int ans = solveUsingTabuSO(capacity, weight, profit, index, n);
    cout << "Max Profit: " << ans << endl;
    return 0;
}
```

Why Return



[Max Profit
Loop ke According
Yaanhi Hona chaiye]

Shifting

Next = CURR

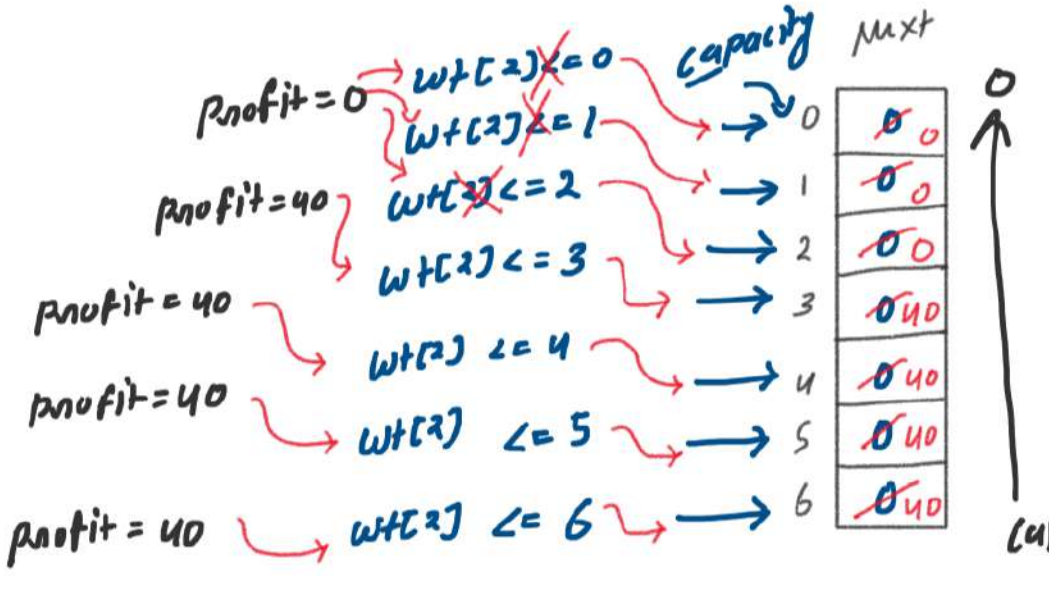
Optimization solution 2

Input:
N = 3, W = 6, weight[] = {1, 2, 3}, profit[] = {10, 15, 40}
Output: 65

0 1 2 0 1 2

When item index = 2

→ wt[2] = 3



Situation 1

↳ Inc = Pro[index] + Mxt[capacity - wt[index]]

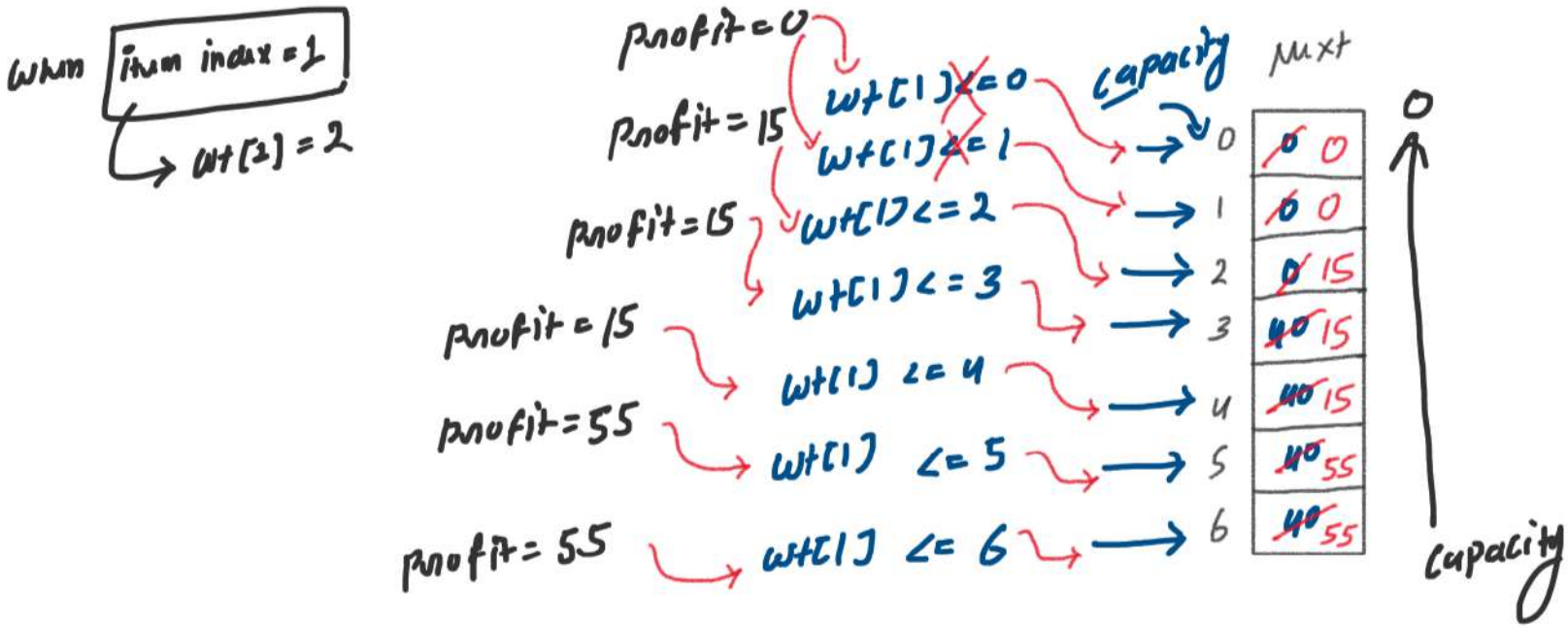
↳ Excl = 0 + Mxt[capacity]

↳ Mxt[capacity] = max of Inc and Excl

Iteration 2

Input:
N = 3, W = 6, weight[] = {1, 2, 3}, profit[] = {10, 15, 40}
Output: 65

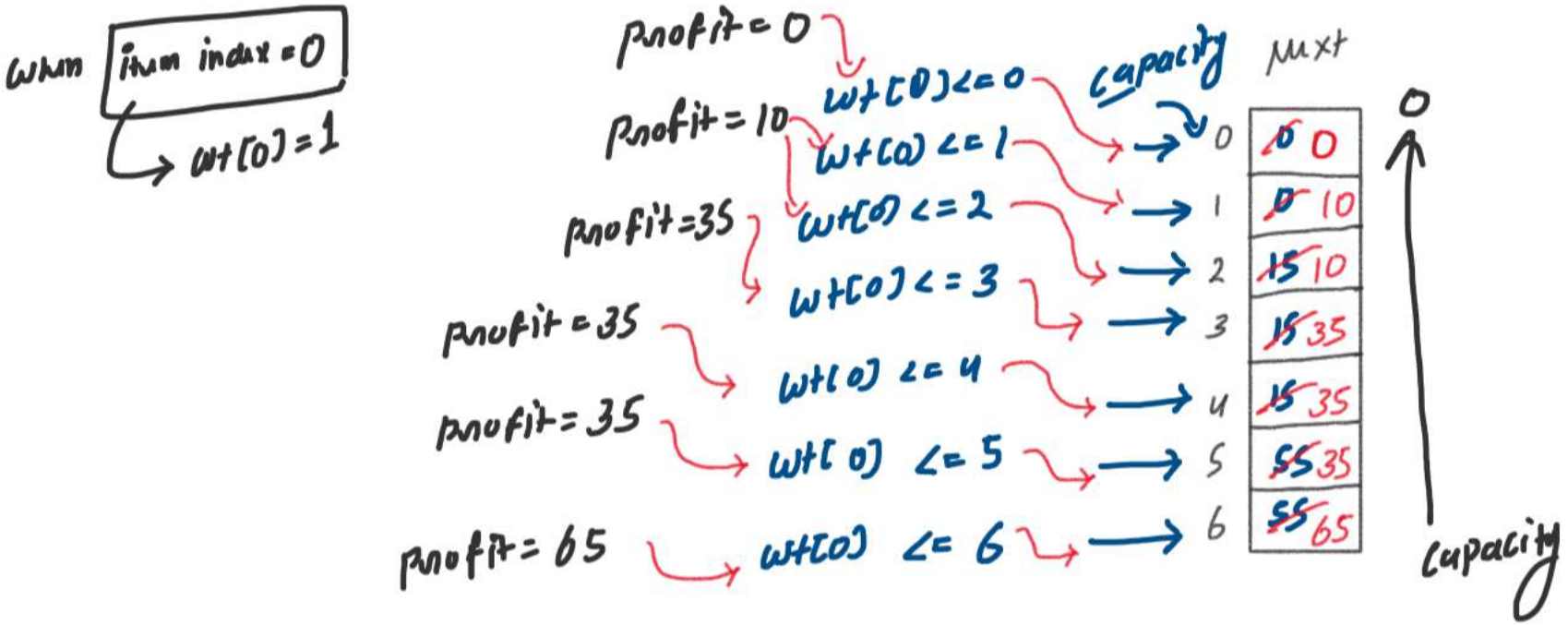
0 1 2 0 1 2



Iteration 3

```
Input:
N = 3, W = 6,  weight[] = {1, 2, 3}, profit[] = {10, 15, 40}
Output: 65
```

0 1 2 0 1 2



```cpp
// Problem 2: 0/1 Knapsack Problem (GFG)
// Approach 5: Space Optimization Approach 2

#include<iostream>
#include<vector>
using namespace std;

int solveUsingTabuSO2(int capacity, int weight[], int profit[], int index, int n){
    vector<int> next(capacity+1, 0);

    for(int col=n-1; col>=0; col--){
        for(int row=capacity; row>=0; row--){
            // Recursive relation (Inclusion or exclusion)
            int include = 0;
            if(weight[col] <= row){
                include = profit[col] + next[row - weight[col]];
            }
            int exclude = 0 + next[row];
            next[row] = max(include, exclude);
        }
    }
    // return ans
    return next[capacity];
}

int main(){
    int capacity = 50;
    int n = 3;
    int weight[] = {10, 20, 30};
    int profit[] = {60, 100, 120};
    int index = 0;

    int ans = solveUsingTabuSO2(capacity, weight, profit, index, n);
    cout << "Max Profit: " << ans << endl;
    return 0;
}
```