DP-23

# Unbounded Knapsack

**Problem Statement** Suggest Edit

You are given 'N' items with certain 'PROFIT' and 'WEIGHT' and a knapsack with weight capacity 'W'. You need to fill the knapsack with the items in such a way that you get the maximum profit. You are allowed to take one item multiple times.

**For Example**

```
Let us say we have 'N' = 3 items and a knapsack of capacity 'W' =  10
'PROFIT' = { 5, 11, 13 }
'WEIGHT' = { 2, 4, 6 }

We can fill the knapsack as:

1 item of weight 6 and 1 item of weight 4.
1 item of weight 6 and 2 items of weight 2.
2 items of weight 4 and 1 item of weight 2.
5 items of weight 2.

The maximum profit will be from case 3 i.e '27'. Therefore maximum
profit = 27.
```

wt → { 2, 4, 6 }

value → { 5, 11, 13 }

weight → 10 (max^m)

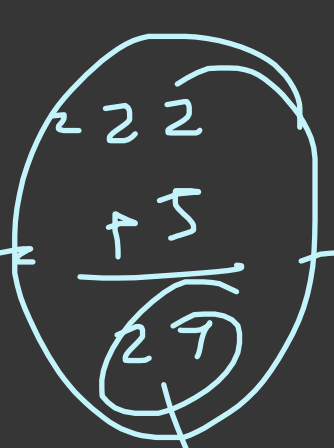
(infinite supply)

So, wt → {2, 4, 6}

val → {5, 11, 13}

W = 10

(infinite supply)



let say:

4 4 weight 2 → 11+11 = 22

1 weight 1 → 5 = 15

10

27

max^m profit

6+2

13+5 = 18

6, 4 → 10

13+11 = 24

27 → Ans

2×5 = 5×5 = 25

Recursion:

So, in Knapsack we have find max profit in under given weight.

So, there are 2 cases that we pick the element or not the pick the element.

→ if we not pick that element we are not adding the element to our bag so we move to next element and add value as 0

→ if we pick it, first we check if the element weight is less than maimy bag weight capacity then we add value of the element and reduce the bag weight with element weight

## Recursion Relation:


① f( ind, W )

③ Base Case: if (i == 0) if (wt[0] <= W) return $\frac{W}{wt[0]}$ x value[0],

else return 0,

② Explore: not pick = 0 + f(ind-1, W)

pick = 0,

if ( wt[i] <= W) pick = val[i] + f( ind, W-wt[i]),

return max (pick, not pick).

T.C. : $O(2^N)$

S.C. : O(N)

After Memoization

T.C. : $O(N * W)$

S.C: $O(N * W) + O(N)$

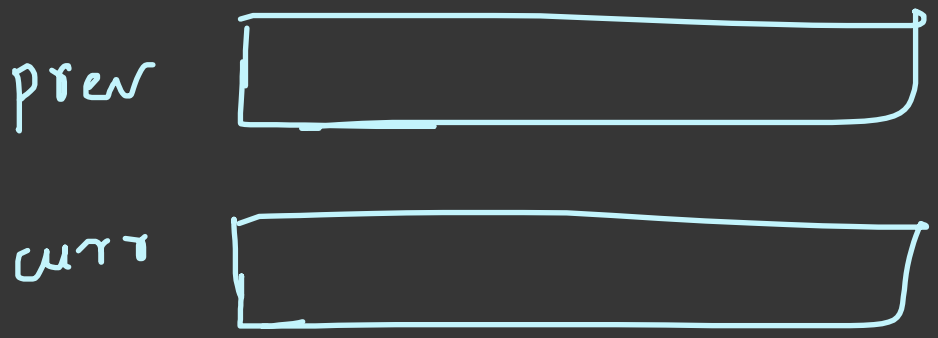
→ Remove by Tabulation

Tabulation.

- Base Case

```
for (i = 0; i <= W; i++)
    dp[0][i] = (i / wt[0]) * val[0];
```

# Space Optimisation (using 1D one array)




not take = prev[w]

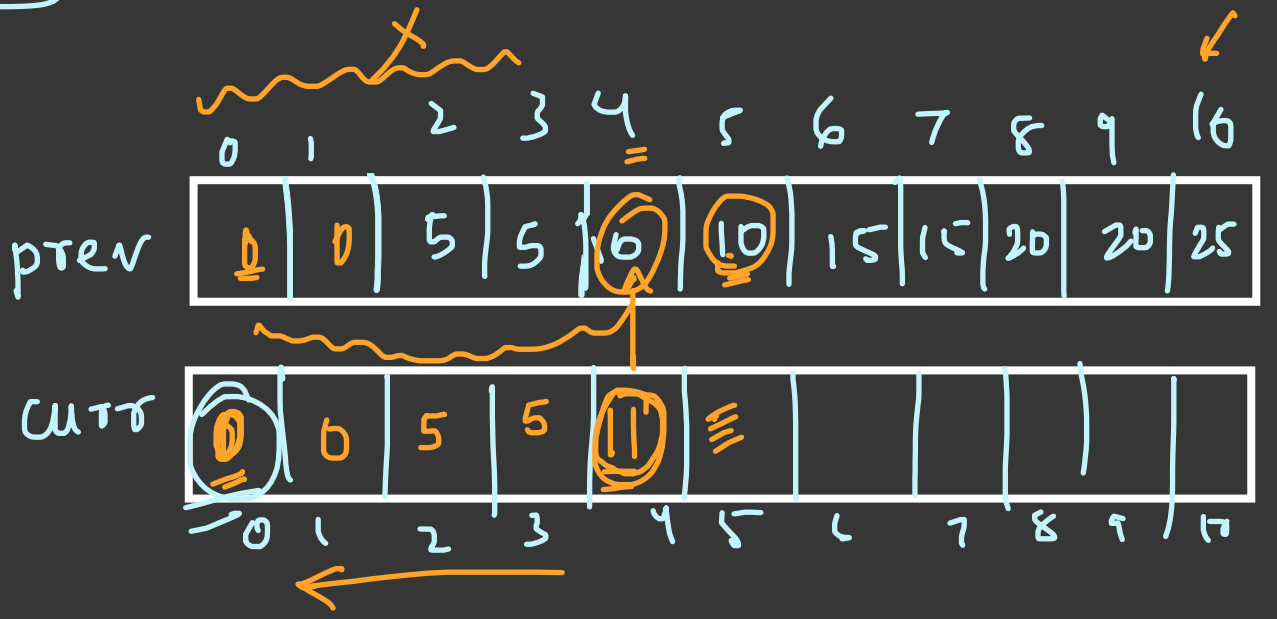
take = val[i] + curr[w-wt]

Let take a example,

W = 10

wt = { 2, 4, 6 }

val = { 5, 11, 13 }




not pick = 0 + 0 = 0

pick = 11 + 0

4 - 4 = 0

```cpp
int unboundedKnapSackTS(int W, vector<int> &wt, vector<int> &val){
    int n = wt.size();
    vector<int> prev(W+1,0), curr(W+1,0);

    for(int w = 0; w<=W;w++){
        prev[w] = (w/wt[0])*val[0];
    }

    for(int i=1;i<n;i++){
        for(int w = 0;w<=W;w++){
            int notPick = 0 + prev[w];
            int pick = 0;
            if(wt[i] <= w)    pick = val[i] + curr[w-wt[i]];
            curr[w]=max(notPick,pick);
        }
        prev = curr;
    }

    return prev[W];
}
```