0-1 Knapsack Problem

17/02/2022
06:45PM

Problem Statement: you are given items, for each item you are given weight and value. You want to maximize the total value of all the items you are going to put in the knapsack that the total weight of the items is less than the knapsack capacity. What is the maximum total value?

So, if collection of item has no item available to pick or, collected item for weight reached to W, stop there after. Exit condition for your recursion

Let's understand problem with example,

wt : {10, 20, 30}
value : {60, 100, 120}

W (capacity of knapsack) = 50

- 50 > 10+20 = (60+100) = 160
- 50 ≯ 10+30 = (60+120) = 180
- 50 ≯ (20+30) = (100+120) = 220

So, let's analyze problem for the constraint to be optimized for reaching the solution.

With all constraint applied you can see that your maximum value will be 220.

Bruteforce / Recursive Solution.

With every item, possibility is either you pick or not to be picked. And so your total possibilities are $(2)^n$ where n is the number of items [As per constraint possible].

Also, if one item picked than that item is not available to be picked again.

Base Case You will say that you reached base case when your total capacity of knapsack reached to W or you collected all items from available items.

Meaning, either $n==0$ or $\Sigma wi \ngtr W$

So, if collection of item has no item available to pick or, collected item weight reached to W.
Comeout from your recursion loop.

Another case to be Analyzed here that-
if (weight(n) > W) that means this item can't be picked. and so you have only (n-1) choice of item to be picked.

So you have only two case, and you need to find the maximum out of two should be considered.

max (Solve(n-1, W), Solve(n-1, W-wt(n)))
① ②

One more point to be noticed here that- if you expand the recursion tree, you will see the many overlapping substructure. And so dynamic programming is the way to solve your problem.

OK, so let's approach the ~~first~~ problem through Recursion first.

① **Recursion**

```
/* List of weight for given n items */
/* n is the number of item */
/* W - max capacity of knapsack */
/* List of values corresponding to items */

int knapsack(int W, int n, int weight[], int value[],
             int cum_wt, int cum_item, int ind) {
             //   50       3
             //   cum_wt = 0, cum_item = 0, ind = 0
    if (cum_wt > W)          false
        return 0;
    if (cum_item >= n)       false
        return 0;
    if (ind >= n)            false
        return 0;
    int l; int r; int maxv = INT_MIN;
    //  l = INT_MIN   r = INT_MIN
```

```
    ~~for(int i = ind; i <= n; i++) {~~     you don't need this loop
    if (weight[i] <= (W - cum_wt)) {
        l = value[i] + knapsack(W, n, weight, values,
                    cum_wt + weight[i], cum_item+1,
                    ind+1)
                    // 60+100
    r = knapsack(W, n, weight, value, cum_wt,
                 cum_item, ind+1);
    maxv = max(l, r);

    return maxv;
```

W=50

10 ✓	20	30	wt
60	100	120	value

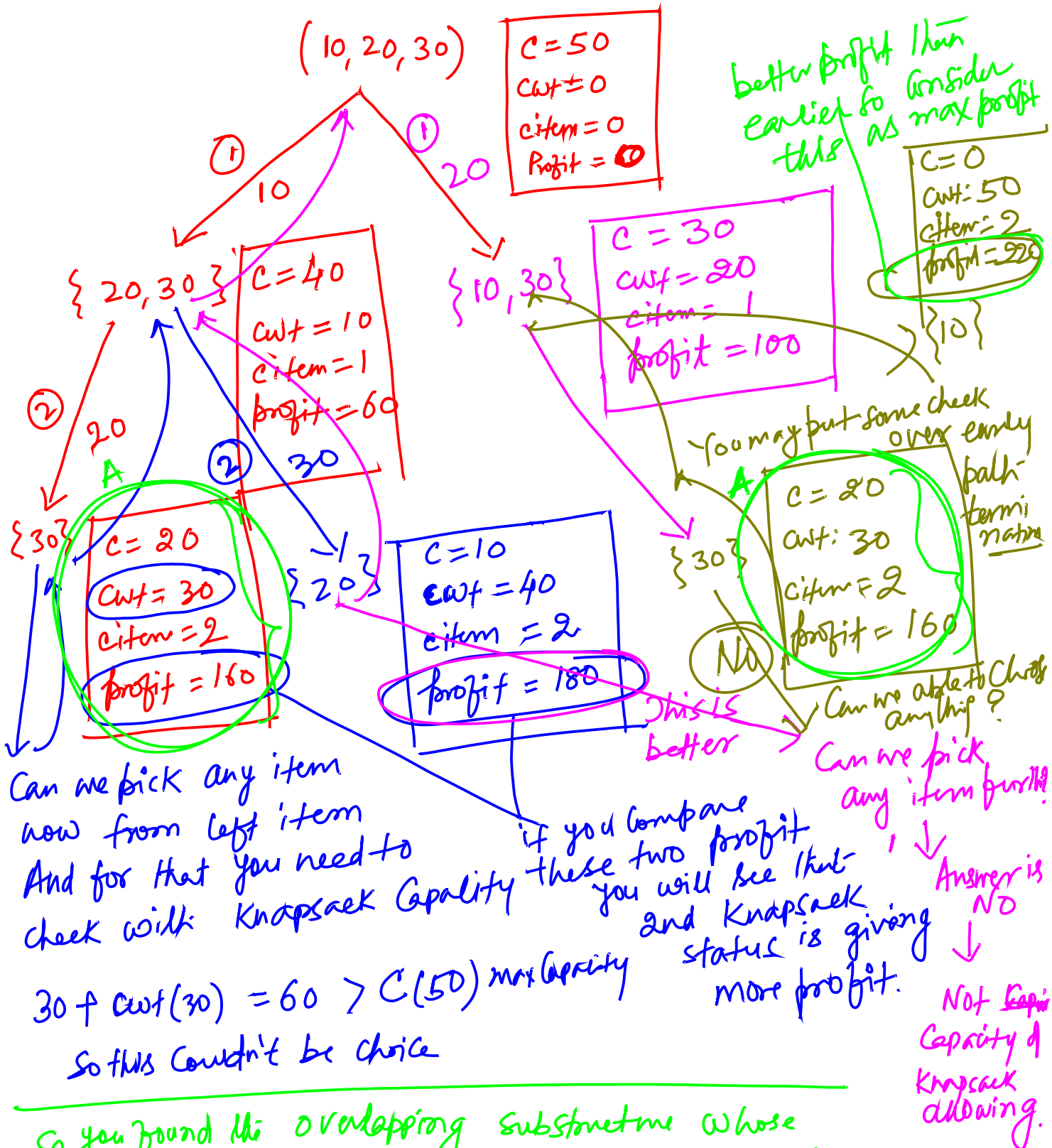
So, just going to write code in IDE to complete this code.

Memoization in Recursion

Let's consider, example again

wt : {10, 20, 30}
val : {60, 100, 120}

Knapsack properties
- Its maximum capacity is C = 50 (in this Example)



30 + cwt(30) = 60 > C(50) max capacity
So this couldn't be choice

So you found the overlapping substructure whose calculation can simply reused to avoid duplicate calculation.

So let's use memoization in your recursion
The two important thing need to maintain here
Capacity of a knapsack available, and no of items
left to be picked.