Lecture : DP-3

Agenda:

- 0-1 knapsack
- Unbounded knapsack
- fractional knapsack.

Qu1 **0-1 knapsack.**

Given n items, each with a weight and a value, find max value which can be obtained by picking items such that total weight of all items <= K.

Note: 1) Every item can be picked only 1 time

2) We can't pick any items partially.

Example: n = 4 items, capacity (K) = 50

$$\begin{array}{cc} & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ wt[\,] = & \left[\begin{array}{cccc} 20 & 10 & 30 & 40 \end{array} \right] \end{array}$$

$$val[\,] = [\ 100 \quad 60 \quad 120 \quad 150\ ]$$

Ans = 0th and 2nd [220]

Idea1: Choose items with max value.

n = 4 items , capacity (K) = ~~50~~ 10

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & \\ wt[\,] = [& 20 & 10 & 30 & 40 &] \end{array}$$

val[ ] = [100   60   120   150]

Ans = $\underset{(3rd)}{150} + \underset{(1st)}{60}$ = 210 [Wrong]

Idea2: Pick items with max ppu.

n = 4 items , capacity (K) = ~~50~~ ~~40~~ 20

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & \\ wt[\,] = [& 20 & 10 & 30 & 40 &] \end{array}$$

val[ ] = [100   60   120   150]

Price per unit = [5   6   4 $\frac{15\cancel{0}}{4\cancel{0}}$ =] 3.75

Ans = $\underset{(1st)}{60} + \underset{(0th)}{100}$ = 160 [Wrong]

Both the greedy idea failed.

Idea3: Get all subsets whose sum <=K and get max value.

TC: (2^n)

n = 4 items , capacity (K) = 50

$$wt[] = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline \end{array}$$

$$\begin{array}{l} \\ wt[] = \end{array} \begin{array}{c} \begin{array}{cccc} \color{red}{0} & \color{red}{1} & \color{red}{2} & \color{red}{3} \end{array} \\ \begin{bmatrix} 20 & 10 & 30 & 40 \end{bmatrix} \end{array}$$

$$val[] = \begin{bmatrix} 100 & 60 & 120 & 150 \end{bmatrix}$$

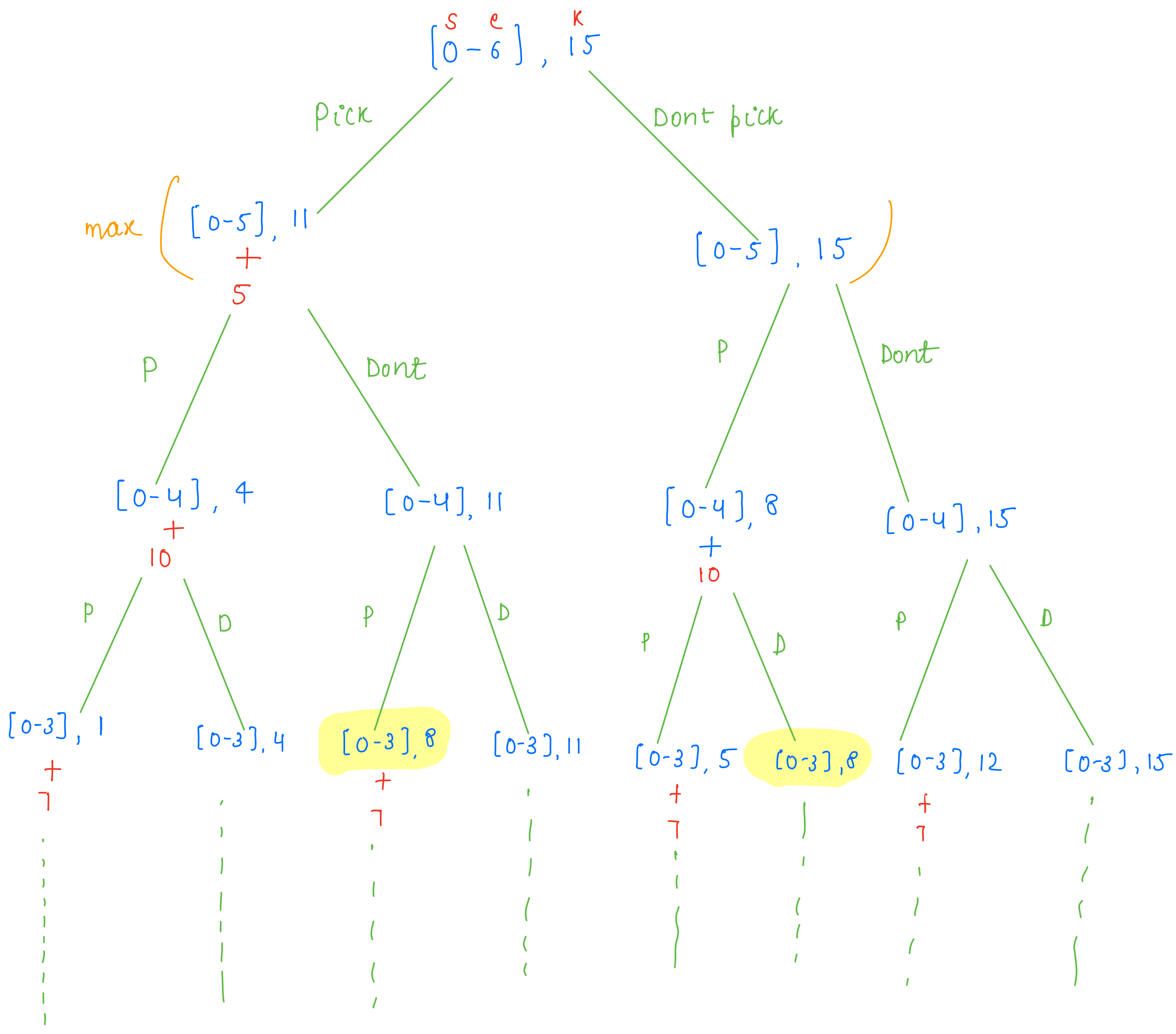
Ans: $\begin{bmatrix} 0 & 2 \end{bmatrix}$

$100 + 120 = \boxed{220}$

Idea 4: Dynamic programming.

n = 7 items, capacity (k) = 15

	0	1	2	3	4	5	6
wt[] =	4	1	5	4	3	7	4
val[] =	3	2	8	3	7	10	5



changing factors
1. end idx
2. k (capacity)

dp[↑][↑]
n+1 k+1

Overlapping subproblems
optimal substructure } DP

Recursive code

```
int 01knapsack( wt[], val[], K , end ) {

        if( end == 0) {
            if( wt[end] <= K) {
                return val[end];
            }
            return 0;
        }

        include = 01knapsack( wt, val, K - wt[end], end -1)
                      + val[end];

        exclude = 01knapsack( wt, val,   K        , end -1)

        return max( include, exclude);
}
```

Memoised code

```
int  01knapsack( wt[] , val[] , K , end , dp[][] )

        if( end == 0) {
            if( wt[ end] <= K) {
                return val[end];
            }
            return 0;
        }
        if( dp[ end] [ K] != -∞) {
            return dp[end] [K];
        }

        include = 01knapsack( wt, val, K - wt[end], end -1)
                    + val[end];

        exclude = 01knapsack( wt, val,  K         , end -1)

        dp[ end] [K] = max( include, exclude);

        return max( include, exclude);
}
```

Tabulative approach

dp[n+1] [k+1]

dp[i][j] = Max profit you can get in a bag of capacity j such that you can choose among first i items.

Example: n = 5, capacity (k) = 8

items:	0	1	2	3	4
wt[]:	3	6	5	2	4
val[]:	12	20	15	6	10

capacity ⟶

val	wt		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
12	3	1	0	0	0	12	12	12	12	12	12
20	6	2	0	0	0	12	12	12	20	20	20
15	5	3	0	0	0	12	12	15	20	20	27
6	2	4	0	0	6	12	12	18	20	21	27
10	4	5	0	0	6	12	12	18	20	22	27

dp[2][3]
- p: dp[1][-3]
- Dont: dp[1][3]

max(dp[1][-3] , dp[1][3])

dp[3][8]
- p: dp[2][3] + 15
- D: dp[2][8]

dp[5][8]
- p: dp[4][4] + 10 = 22
- D: dp[4][8] 27

Tabulative | iterative code

```
int 01Knapsack( wt[], val[], K) {
        n = wt.length;
        dp[n+1][K+1];

        for( i=0; i<=n; i++) {

            for (j=0; j<=K; j++) {

                if( i==0 || j==0) {
                    dp[i][j] = 0;
                } else {

                    exclude = dp[i-1][j];
                    if ( j - wt[i-1] >=0) {
                        include = dp[i-1][j - wt[i-1]];
                    }

                }
                dp[i][j] = max(include, exclude);
            }
        }

    return dp[n][K];
}
```

TC: O(n*K)

SC: O(n*K) ⟶ Try optimising this space.

Break: 8:09: 8:19 AM

Qu: **Unbounded Knapsack**

Given n items, each with a weight and a value, find max value which can be obtained by picking items such that total weight of all items <= K.

Note: 1> Every item can be picked as many times as we want.

2> We can't pick any items partially.

Example: $K = 50$

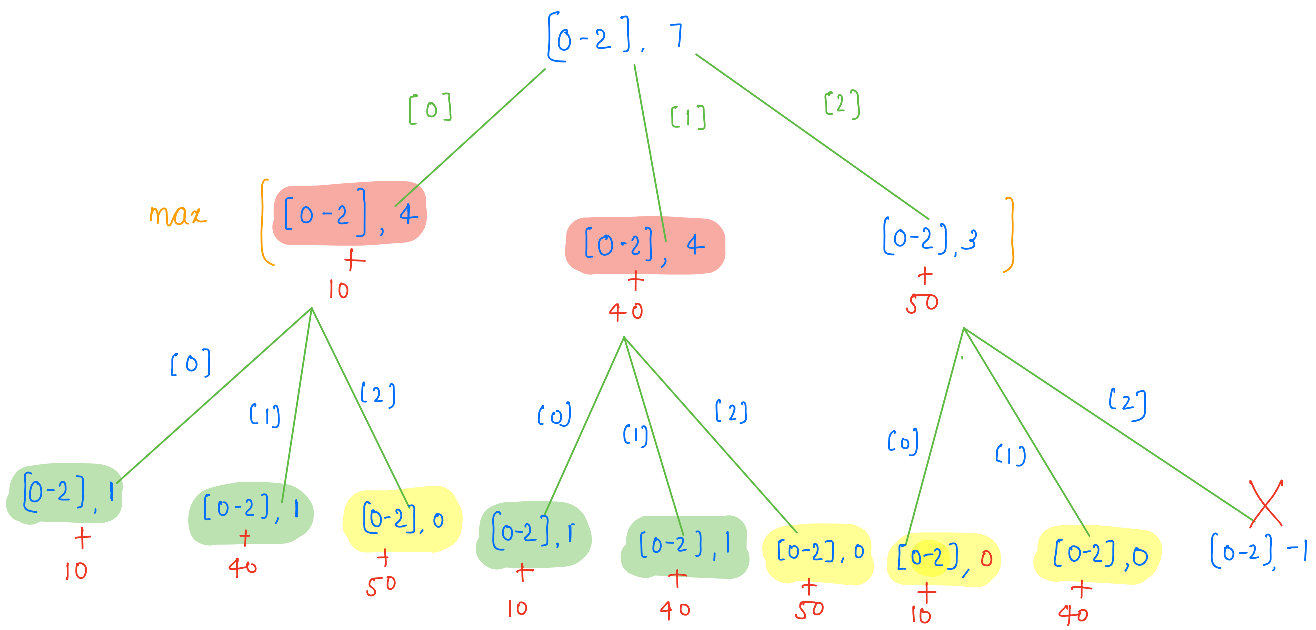
items:	0	1	2	3
wt[]	20	13	10	40
val[]:	100	66	40	150

ans = [0, 0, 2] = 100 + 100 + 40 = 240 Ans

Idea:

k = 7

	0	1	2
val[] :	10	40	50
wt[] :	3	3	4



Overlapping subproblems
Optimal substructure } DP

changing factors

1.) K (capacity) → 1 D array

dp[k+1]

Tabulative approach

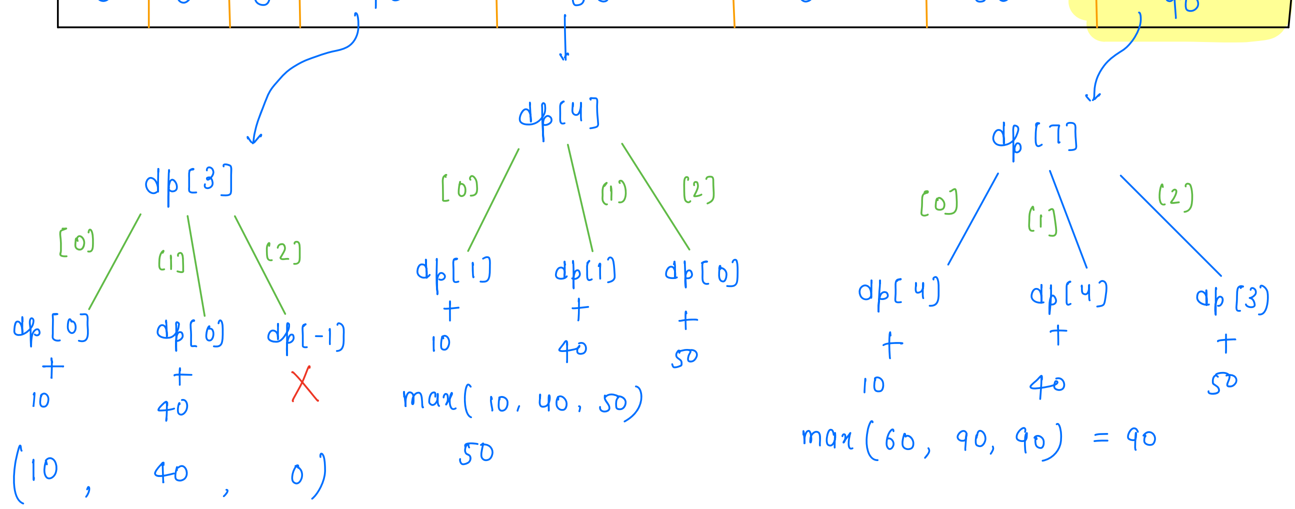
dp[i] = max profit in a bag of capacity i.

Dry run:

k = 7

	0	1	2
val[ ] :	10	40	50
wt[ ] :	3	3	4

0	1	2	3	4	5	6	7
0	0	0	40	50	50	80	90



Tabulative | Iterative code

```
int unboundedknapsack(wt[], val[], K) {

    n = wt.length;

    dp[K+1];

    dp[0] = 0;

    for(i=1; i<=K; i++) {

        max = -∞;

        for(j=0; j<n; j++) {
            if (i-wt[j] >= 0) {
            max = Math.max(max, val[j] +
                                dp[i - wt[j]]);
            }
        }

        dp[i] = max;
    }

    return dp[K];
}
```

TC: O(n * K)

SC: O(K)

Qu: **fractional knapsack**

Given n items, each with a weight and a value, which can be obtained by picking items such that total weight of all items <= K.

Note: 1> Every item can be picked only 1 time

2> We ~~can't~~ can pick any items partially.

Eg:

2 kg gold — ₹1000

1 kg silver — ₹300

Capacity of bag = 1 kg

0-1 / unbounded knapsack — pick 1 kg of silver [300]

fractional knapsack — pick 1 kg of gold [500]

Idea:

$k = \cancel{50}\ \cancel{40}\ \cancel{20}\ 0$

	0	1	2
wt[] :	10	20	30
val[] :	60	100	120
ppu :	6	5	4

ans :

60 + 100 + [pick 2nd item partially]
20 * 4 = 80

60 + 100 + 80 = $\boxed{240 \text{ Ans}}$

Assignment: Try the coding yourself.

```
class pair {
    wt;
    val;        ⟶  sort on basis of ppu
    ppu;
}
```

Thankyou :)