Agenda

1) 0/1 Knapsack

2) Unbounded Knapsack

3) Iterative DP

Q.1 **0-1 Knapsack**

Given N items, each with a weight and value, find **max value** which can be obtained by picking items such that total weight of picked items <= K. (K is given)

Note: i) Every item can be picked atmax 1 time.

ii) we can't take part of an item.

	0	1	2	3
wt:	20	10	30	40
val:	100	60	120	150
vpk:	5	6	4	3.75

K = 50

cap = ~~50~~ ~~40~~ 20

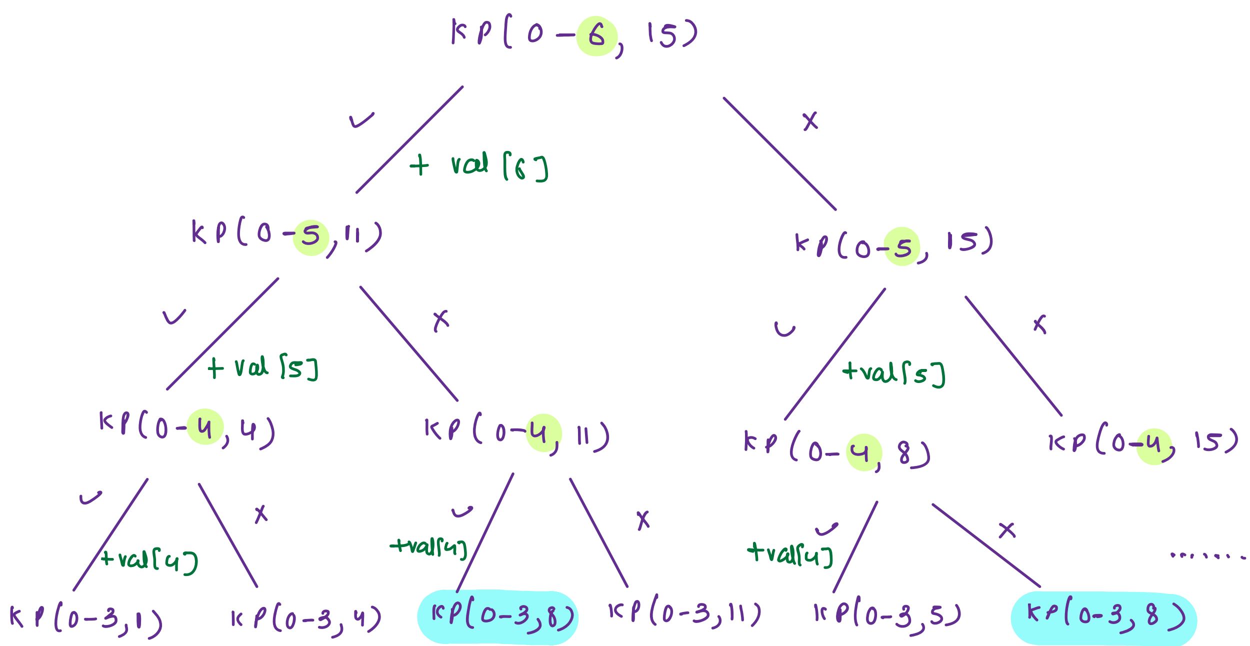
ans = ~~0~~ ~~60~~ 160 ✗

(greedy won't work)

The idea of subsequence can be applied to see all possibilities of item selection but take care of K as well in logic.

	0	1	2	3	4	5	6
wt :	4	1	5	4	3	7	4
val:	3	2	8	3	7	10	5

K = 15



dp[ ][ ] → dp[n][k+1]

↓ index (0 to n-1); ↳ cap

↓ index ↓ cap

```
int solve(int[] wt, int[] val, int k) {
    int n = wt.length;
    dp = new int[n][k+1];
    // fill dp with -1
    return helper(wt, val, n-1, k);
}
```

```
int [] [] dp;    ( dp -> new int [n] [k+1] )

int helper (int[] wt, int [] val, int i, int k) {
        if( i<0 || k==0) {
              return 0;
        }
        if( k < 0) {
              return -∞;
        }
        if(dp[i] [k] != -1) {
              return dp[i] [k];
        }
        int a= helper(wt,val, i-1, k-wt[i]) + val[i];

        int b= helper(wt,val, i-1, k);

        int ans= Math.max (a,b);

        dp[i] [k] = ans;

        return ans;
}
```

TC : O(nk)

SC: O(nk)

Q-2 **Unbounded Knapsack**

Given N items, each with a weight and value, find **max value** which can be obtained by picking items such that total weight of picked items <= K. (K is given)

Note: i) Every item can be picked unlimited no. of times.

ii) we can't take part of an item.

	0	1	2	3 ✓
wt:	20	10	30	40
val:	100	60	120	150

K = 50

greedy won't work

wt:	11	10
val:	22	19
v/wt	2	1.9

K = 20

kp(i, k)

✓ (+ val[i]) → kp(i, k-wt[i])

✗ → kp(i-1, k)

```
int solve(int[] wt, int[] val, int k) {
    int n = wt.length;
    dp = new int[n][k+1];
    // fill dp with -1
    return helper(wt, val, n-1, k);
}
```

```
int [ ] [ ] dp;    ( dp -> new int [n] [k+1] )

int helper ( int[ ] wt, int [ ] val, int i, int k) {
        if( i<0 || k==0) {
              return 0;
        }
        if( k < 0) {
              return -∞;
        }
       if(dp[i] [k] != -1) {
              return dp[i] [k];
       }
       int a= helper(wt,val, i, k-wt[i]) + val[i];

       int b= helper(wt,val, i-1, k);

       int ans= Math.max (a,b);

        dp[i] [k] = ans;

        return ans;
}
```

TC : O(nk)

SC: O(nk)

Iterative DP: Tabulation

Let's revisit some solved problems.

①. Fibonacci :

$fib(n) = fib(n-1) + fib(n-2)$

$n = 9$

	0	1	2	3	4	5	6	7	8	9
dp	0	1	1	2	3	5	8	13	21	34

$dp[i]$ = fibonacci of i

②. count ways from $(0,0)$ to $(n-1, m-1)$ → R ↓ D

	0	1	2	3
0				
1				
2				

$ways(i,j) = ways(i-1, j) + ways(i, j-1)$

How to travel :

→ the cells on which you are dependent should be solved before coming to you.

dp

	0	1	2	3
0	1	1	1	1
1	1	2	3	4
2	1	3	6	10

$dp[i][j]$: no. of ways to reach (i,j) starting from $(0,0)$

③. min path cost from (0,0) to (n-1, m-1)   → R   ↓ D

	0	1	2	3
0	2	1	3	2
1	4	7	1	8
2	6	3	10	5

mincost (i,j) = Min(mincost (i-1, j), mincost(i,j-1))
+ mat [i] [j]

dp

	0	1	2	3
0	2	3	6	8
1	6	10	7	15
2	12	13	17	20

→ if i==0 && j==0
dp[0] [0] = mat [0] [0]

→ if i==0
dp [i] [j] = dp [i] [j-1] + mat [i] [j]

→ if j==0
dp [i] [j] = dp [i-1] [j] + mat [i] [j]

→ rest
dp [i] [j] = min (dp [i-1] [j], dp [i] [j-1]) + mat [i] [j]

dp [i] [j]
= min cost to reach
(i,j) from (0,0)

④. 0-1 Knapsack

dp → [n] [K+1]

	0	1	2	3
wt :	3	6	4	2
val :	12	20	15	6

K = 7

$KS(i, K) = \max(KS(i-1, K-wt[i]) + val[i],$
$KS(i-1, K))$

K →

Items ↓

	0	1	2	3	4	5	6	7
0	0	0	0	12	12	12	12	12
1	0	0	0	12	12	12	20	20
2	0	0	0	12	15	15	20	27
3	0	0	6	12	15	18	21	27

dp [i] [K] : max value generated till i^{th} index when cap = K.

Doubts

$$mincost(i, j) = \min(mincost(i-1, j), mincost(i, j-1)) + mat[i][j]$$

$$A = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 2 & 1 & 3 & 2 \\ 1 & 4 & 7 & 1 & 8 \\ 2 & 6 & 3 & 10 & 5 \end{array}$$

$$dp : \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 2 & 3 & 6 & 8 \\ 1 & 6 & 10 & 7 & 15 \\ 2 & 12 & 13 & 17 & 20 \end{array}$$