Todays Content:

- → 0/1 knapsack
- → unbounded knapsack

Steps:

a. Sub Problems/ Overlapping

b. dp State:

c. dp eqn:

d. Final ans:

e. Dp Table: TC: SC

f. Code

Given N items each with a weight & value, find max value which can be obtained by picking items such that total weight of all items $\leq k$ (k given)

Note1: Every item can be picked at max 1 time

Note2: We cannot take a part of item

Eg: N=4 items k=50

Items:	0	1	2	3
Weight[]:	20	10	30	40
value[]:	100	60	120	150
val/w :	5	6	4	3.75

Idea1:
- : pick items in dec order value
  - * not working
- : pick according v/w ratio
  - * not working
- → Check correctness of idea:

Case1: items: 1 3
weight : 10 40
value : 60 150 = 210

Case2: items: 0 2
weight : 20 30
value : 100 120 = 220

Idea: Every item has 2 choices
- : 2^n Combinations
- : For every combination,
  - check if Total weigh of com $\leq k$
  - update ans & get overall max val
- → TC: $O(2^n * n)$ // *n because we iterate & check.

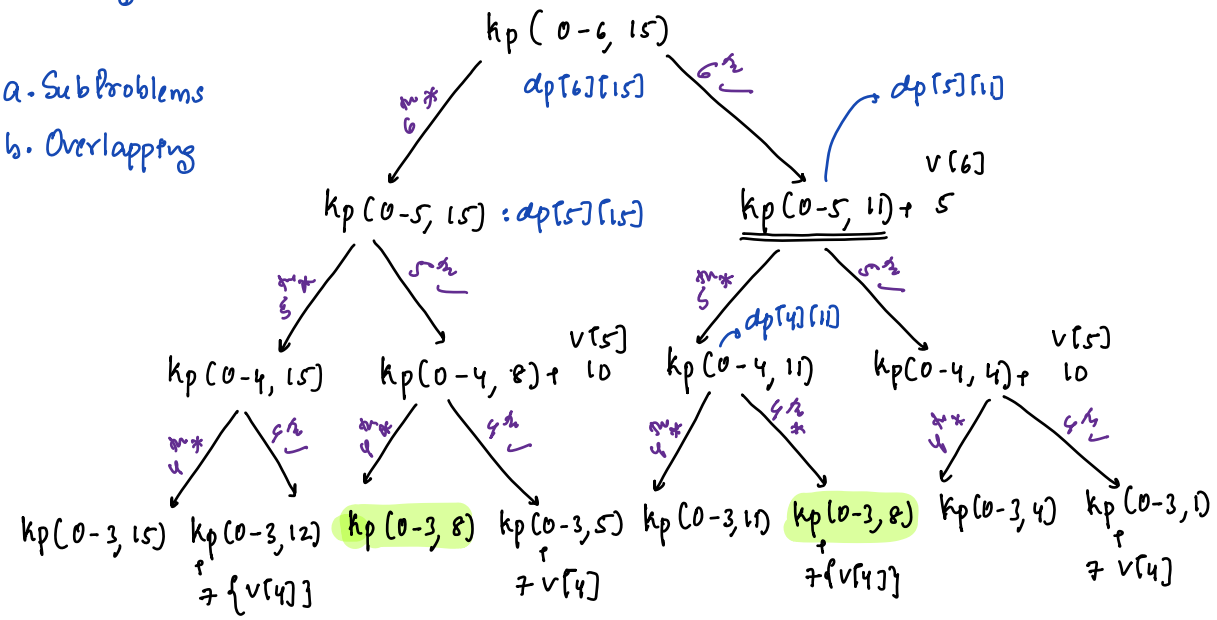
K = 15

N = 7	0	1	2	3	4	5	6
w[]	4	1	5	4	3	7	4
v[]	3	2	8	3	7	10	5

// Using items (0-6) & max w <= 15, get max value can be obtained.

a. Subproblems

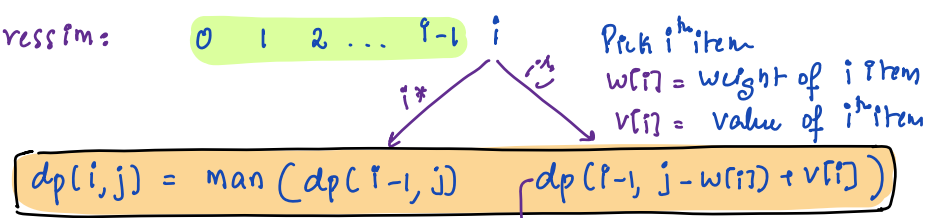
b. Overlapping



dp Steps:

→ dp State : dp(i, j) : From items [0, i] & Pick items such that
: total weight <= j {rem weight} & we get max value

→ dp Expression:



→ dp ans: From items [0, N-1]
Pick items such that
Total weight <= k & pick max val
Problem = dp[N-1][k]

states * TC for each

→ dp table: int dp[N][k+1] TC: O(N*k) * 1 SC: O(N*k) + stacksize

Pseudo Code:

```
int dp[N][K+1] = {INVALID/-1}
int knapSack 0/1 ( int i, int j, int w[], int v[] ) {
    if( i<0 || j ==0) { // we cannot pick any time return 0}
    if(dp[i][j]==-1) {
        int a = knapSack 0/1 ( i-1, j, w, v)
        if(j>=w[i]){
            a = max(a, knapSack 0/1 (i-1, j-w[i], w, v) + v[i])
        }
        dp[i][j] = a
    }
    return dp[i][j]
}
```

Note: We can store dp states using hashmap:

↳ HashMap<key, value> hm:

key: We need to store both variable i, j

key: pair<int, int>

key: String : String(i)+" "+String(j)

value: dp value of that state

```
HashMap<String, int> hm.
int knapSack 0/1 ( int i, int j, int w[], int v[] ) {
    if( i<0 || j ==0) { // we cannot pick any time return 0}
    if( hm.search(String(i)+" "+String(j)) ==false) { // first time
        int a = knapSack 0/1 ( i-1, j, w, v)
        if(j>=w[i]){
            a = max(a, knapSack 0/1 (i-1, j-w[i], w, v) + v[i])
        }
        hm.insert(String(i)+" "+String(j), a)
    }
    return hm[String(i)+" "+String(j)]
}
```

Adv: In hashmap only if a state is used we create space for that

Tracing:

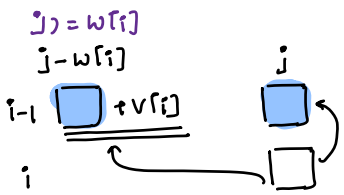
items :	0	1	2	3	4
w[] :	3	6	5	2	4
v[] :	12	20	15	6	10

items = 5 k = 7

dp[5][8]

items \ W (Weight →)	0	1	2	3	4	5	6	7
3 - 0	0	0	0	12	12	12	12	12
6 - 1	0	0	0	12	12	12	20	20
5 - 2	0	0	0	12	12	15	20	20
2 - 3	0	0	6	12	12	18	20	21
4 - 4	0	0	6	12	12	18	20	22

Task: Fill Entire matrix cell by cell. practice.



Fill Table:

if j >= w[i]

$dp(i,j) = \max(dp(i-1,j),\ dp(i-1, j-w[i]) + v[i])$

$dp(1,3) = \max(dp(0,3),\ dp(0,-3) + 20)$

$dp(1,6) = \max(dp(0,6),\ dp(0,6-6) + 20)$

$dp(2,5) = \max(dp(1,5),\ dp(1,5-5) + 15) = \max(12, 15)$

$dp(2,6) = \max(dp(1,6),\ dp(1,6-5) + 15) = \max(20, 15)$

$dp(3,5) = \max(dp(2,5),\ dp(2,5-2) + 6) = \max(15, 18)$

$dp(3,7) = \max(dp(2,7),\ dp(2,7-2) + 6) = \max(20, 21)$

$dp(4,7) = \max(dp(3,7),\ dp(3,3) + 10) = \max(21, 22) = 22$

items :	0	1	2	3	4
w[] :	3	6	5	2	4
v[] :	12	20	15	6	10

dp[5][7]

	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	12	15	20	20
3	0	0	6	12	12	18	20	21
4	0	0	6	12	12	18	20	22

if $j >= w[i]$

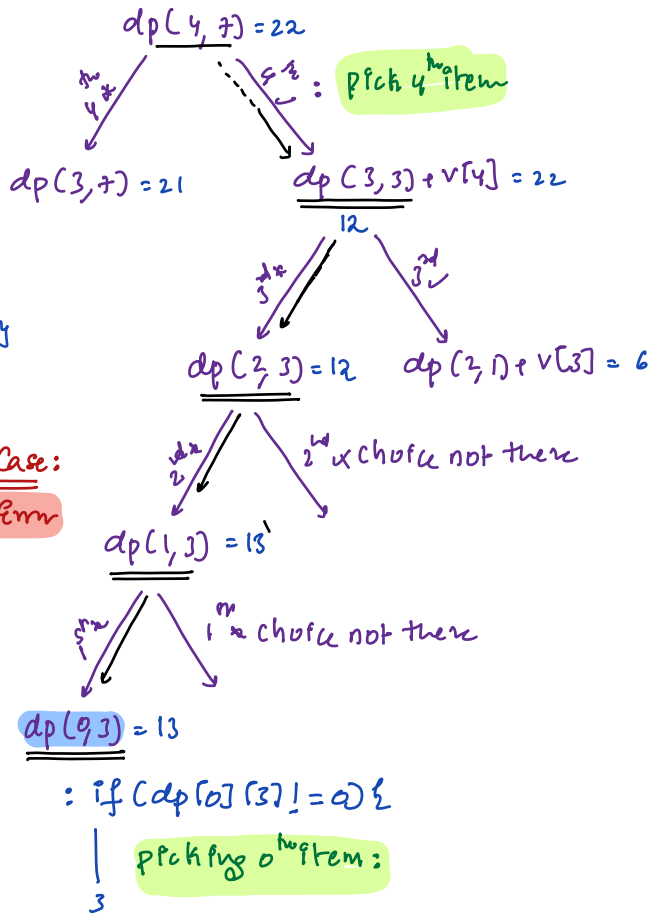
$$dp(i,j) = \max(dp(i-1,j),\ dp(i-1,\ j-w[i]) + v[i])$$

Printing Items:

Fill dp[N][K+1]

i = n-1 j = k

```
while(i>=0 && j>0){
    if(i==0){
        if(dp[0][j] != 0){ print(0); break;}
        else {// not taking; break;}
    }
    if( dp[i][j] == dp[i-1][j]){ // Edge Case: i=0 error
        // ith ele not picked
        i = i-1
    }
    else
        // ith ele picked
        print(i)
        i = i-1  j = j-w[i]
    }
}
```

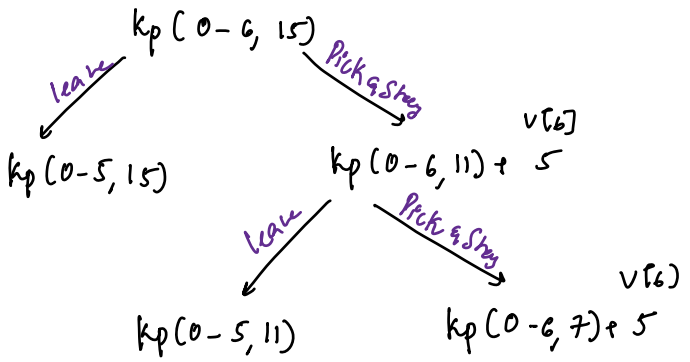


28) Exactly same as above problem

Note: A single item can be picked as many times as possible

K=15 N=7	0	1	2	3	4	5	6 ↓
w[]	4	1	5	4	3	7	4
v[]	3	2	8	3	7	10	5

Idea:



2 choices

a) leave item

b) Pick & Stay

→ dp State: dp[i,j]: From items [0, i] & Pick items such that : total weight <= j[rem weight] & we get max value

→ dp Expression:

dp[i,j]: 0 1 2 ... i-1 i

Pick i^{th} item & Stay

w[i] = weight of i item

v[i] = value of i^{th} item

i* i^th

dp[i,j] = max(dp[i-1, j], dp[i, j - w[i]] + v[i])

if: j[rem_weight] >= w[i]. : only than we can pick i^{th} item

exists: if j >= w[i]

Fill rem details.