DP 3

Good Evening

Today's Agenda

* Target sum
* Fractional knapsack
* 0/1 Knapsack
* Unbounded Knapsack (0-N knapsack)

Target Sum

Description

You are given a set of non-negative integers and a target sum. The task is to determine whether there exists a subset of the given set whose sum is equal to the target sum.

A[] = { 3 34 4 12 5 2 } tar = 9

Ans = true

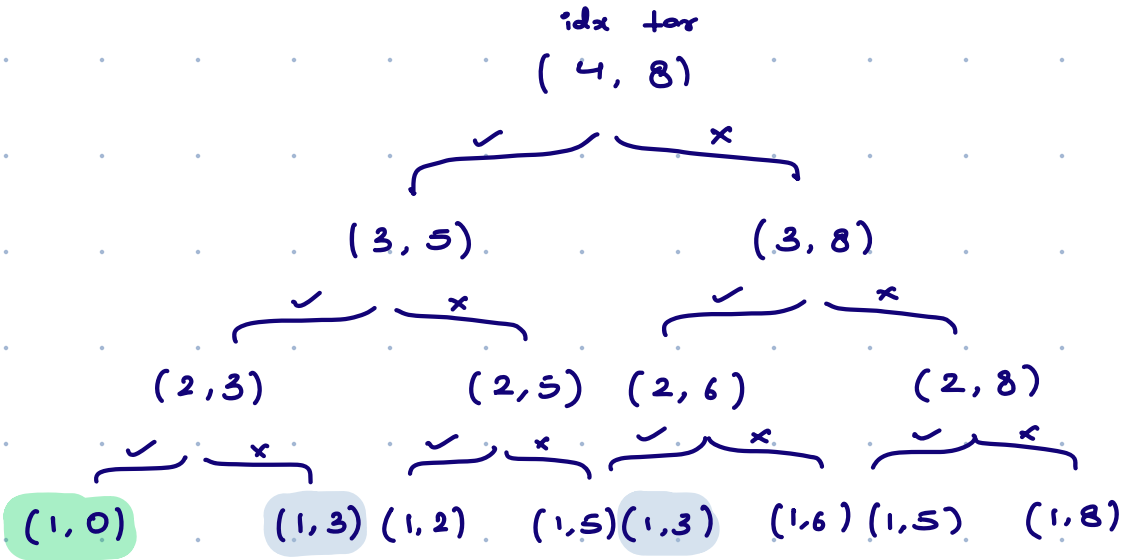
A[] = { 7 2 3 2 3 } tar = 8

Ans = true

Idea → Consider all subsets & look if sum of subset is equals to target.

A[] = { 7 2 3 2 3 } tar = 8

0 1 2 3 4



```
                                                    n-1
                                                     ↑
boolean targetsum(int[] A, int i, int tar){

    if (tar == 0) return true;

    if (tar < 0 || i<0) return false;

    boolean f1 = targetsum(A, i-1, tar)          // not pick

    boolean f2 = targetsum(A, i-1, tar - A[i])   // pick

    return f1 || f2;

}
```

TC : $O(2^n)$

SC : $O(n)$

Top Down Approach

boolean [][] dp → True / False ✗

int [][] dp → -1 Default value / 0 False / 1 True ✓

```
int[][] dp = new int[N][tar+1];

Arrays.fill(dp, -1)    // fill dp with -1;
```

```
int targetsum(int[] A, int i, int j, int[][] dp){
    if(j==0) return 1
    if(j<0 || i<0) return 0;
    if(dp[i][j] != -1) return dp[i][j]
    int f1 = targetsum(A, i-1, j, dp)        // not pick
    int f2 = targetsum(A, i-1, j-A[i], dp);  // pick
    if(f1==1 || f2==1) return dp[i][j]=1;
    else return dp[i][j]=0;
}
```

(i → n-1, j → tar)

TC : O(n*tar)

SC : O(n*tar)

* Bottom Up Approach

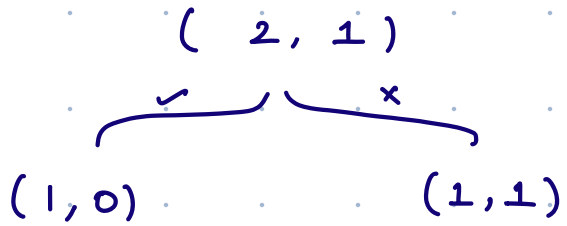
A[] = {2, 1, 4} (0, 1, 2) tar = 6

Ek ↓ \ Tar →	0	1	2	3	4	5	6
0	1	0	0	0	0	0	0
2 (1)	1	0	1	0	0	0	0
1 (2)	1	1	1	1	0	0	0
4 (3)	1	1	1	1	1	1	1

TC : O(n*tar)

SC : O(n*tar)

no. of ele, tar)



Fractional KnapSack

Given N cakes with their happinness & weight. Find max total happinness that can be kept in a bag with capacity = cap

Note → Cake can be divided to pastries

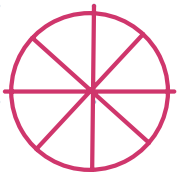
N = 5
cap = 40

happiness [ ] = { 3 8 10 2 5 }
weight [ ] = { 10 4 20 8 15 }

Idea 1 → Greedy → Pick the cake with maximum happiness first.

$\Sigma h = 10 + 8 + 5 + 0.3 \Rightarrow 23.3$

$\Sigma w = 20 + 4 + 15 + 1$



10 kg → 3 happinness

1 kg → $\frac{3}{10}$ happiness

N = 5 happiness [ ] = { 3 8 10 2 5 }

cap = 40 weight [ ] = { 10 4 20 8 15 }

Idea 2 → Pick the least weights item first.

$\Sigma h = 8 + 2 + 3 + 5 + 1.5 \Rightarrow 19.5$

$\Sigma w = 4 + 8 + 10 + 15 + 3$

20 kg → 10 happiness

$1\ kg = \frac{10}{20}$

$3\ kg = \frac{10}{20} * 3$ happiness

* Idea 3 → Pick the cake with maximum $\frac{\text{happiness}}{\text{weight}}$ ratio

Hap [ ] = { 4 8 10 2 15 }

wt [ ] = { 4 4 20 8 16 }

bag = 40

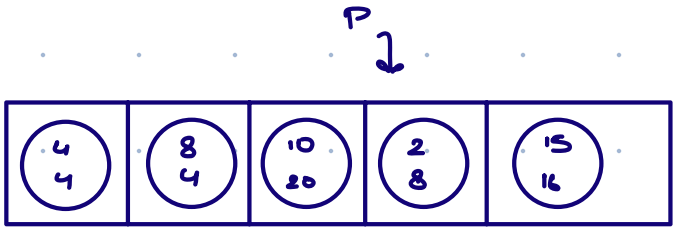
Hap/wt = { 1 2 0.5 0.25 0.93 }

Pick elements in order of max hap/wt ratio.

$\Sigma h = 8 + 4 + 15 + 8 = 35$

$\Sigma wt = 4 + 4 + 16 + 16 = 40$

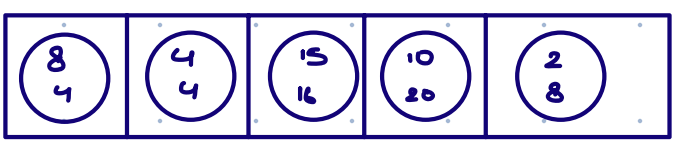
```
public class pair {
    int hp;
    int wt;
    pair (int h, int w) {
        hp = h;
        wt = w;
    }
}
```




```
Pair [] p = new pair [n];

for (i = 0; i < n; i++) {
    p[i] = new pair (hap[i], wt[i]);
}
```

```
Arrays.sort (p, new Comparator<pair> {

    public int compare (Pair a, Pair b) {

        double r1 = a.hp / a.wt * 1.0

        double r2 = b.hp / b.wt * 1.0

        if (r1 < r2) return 1;
        else if (r1 > r2) return -1;
        else return 0;
    }
}
```

p =



```
ans = 0
for (i=0; i<n; i++){
    if ( p[i].wt ≤ cap ){
        ans += p[i].hp;
        cap = cap - p[i].wt;
    }
    else {
        ans += ( p[i].hp / p[i].wt * 1.0 * cap );
        break;
    }
}
return ans;
```

10:18 pm → 10:28 pm

0/1 knapSack

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 ≤ cap

Note 1 :- Every item can be picked at max 1 time

Note 2 :- We can not take a part of item

Eg:- N = 4 items K = 50

value[] = { 100 60 120 150 }
0 1 2 3

wt[] = { 20 10 30 40 }

v/wt = 5 6 4 3.75

Idea 1 → Pick item with maximum value/weight ratio

$\Sigma v = 60 + 100 = 160$ ✗

$\Sigma wt = 10 + 20$

Idea 2 = Pick item with maximum value

$\Sigma v = 150 + 60 = 210$ ✗

$\Sigma wt = 40 + 10$

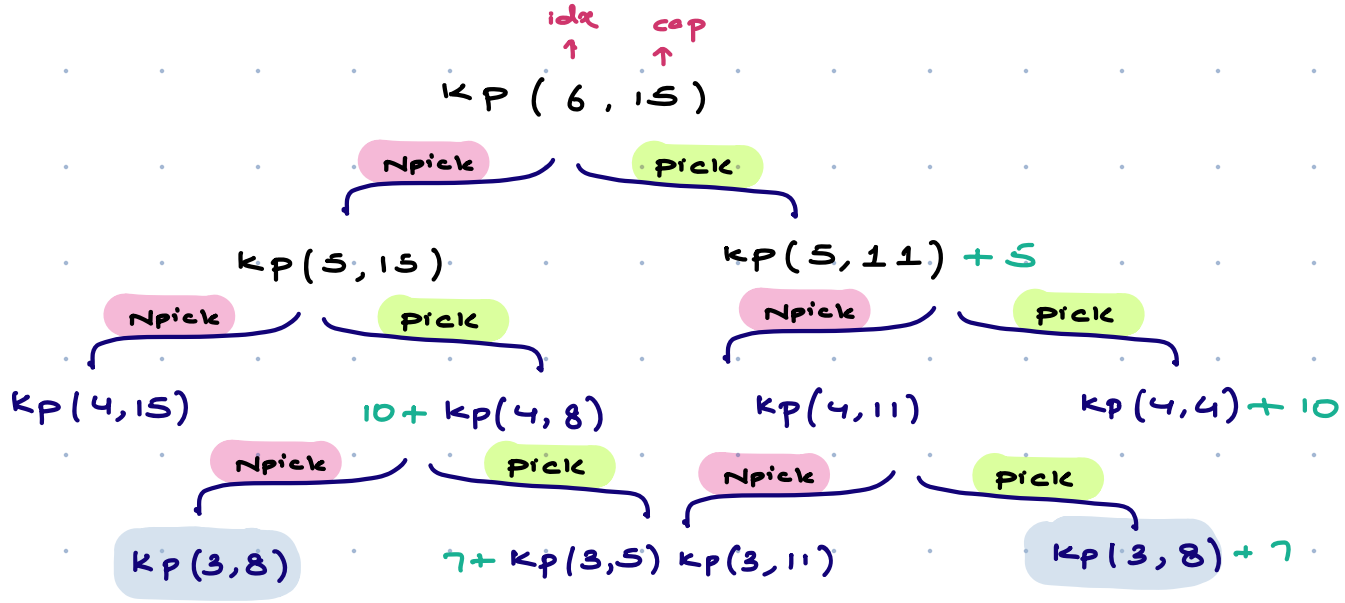
Correct ans $\Sigma v = 120 + 100 = 220$ ✓

$\Sigma wt = 30 + 20 = 50$

Idea → Consider all the subsets of items & select the subset which provides the maximum value.

wt[] = { 4 1 5 4 3 7 4 }
0 1 2 3 4 5 6

val[] = { 3 2 8 3 7 10 5 } Cap = 15




```
int [][] dp = new int [n][cap+1];   // fill -1
```

```
int knapsack ( int i, int j, int [] wt, int [] val, dp ){
    if (i<0 || j<0) return 0
    if (dp[i][j] != -1) return dp[i][j];
    int reject = knapsack(i-1, j, wt, val, dp);
    int select = 0
    if (wt[i] <= j){
        select = knapsack(i-1, j - wt[i], wt, val, dp) + val[i];
    }
    return dp[i][j] = max(reject, select);
}
```

Tc : O(N * cap)

Sc : O(N * cap)

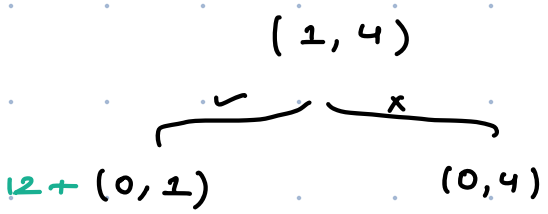
dp[i][j] = max value we can generate with i elements within capacity j.

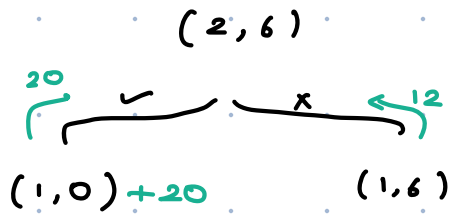
wt[] = {3 6 5 2 4}
0 1 2 3 4

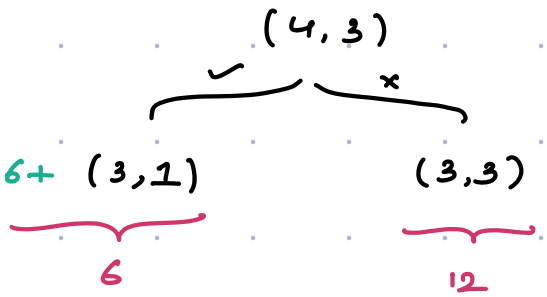
val[] = {12 20 15 6 10}

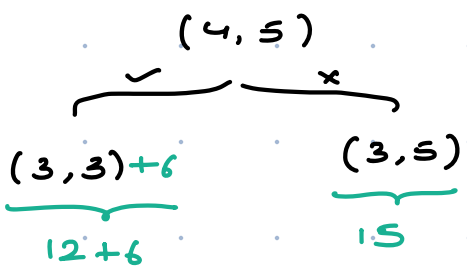
cap = 7

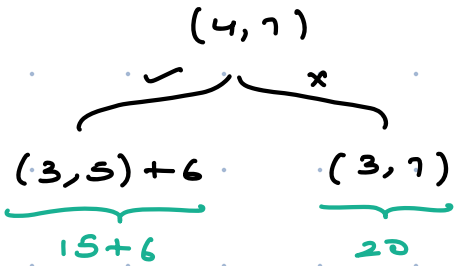
val	wt	Ele ↓ / Cap →	0	1	2	3	4	5	6	7
		0	0	0	0	0	0	0	0	0
12	3	1	0	0	0	12	12	12	12	12
20	6	2	0	0	0	12	12	12	20	20
15	5	3	0	0	0	12	12	15	20	20
6	2	4	0	0	6	12	12	18	20	21
10	4	5	0	0	6	12	12	18	20	22











$$dp[i][j] = \max(\underbrace{dp[i-1][j]}_{\text{not pick}}, \underbrace{val[i-1] + dp[i-1][j-wt[i-1]]}_{\text{pick}})$$

```
int [][] dp = new int[n+1][cap+1];

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

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

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

        else {
            rej= dp[i-1][j]
            sel = 0
            if ( wt[i-1] <=j )
                sel= dp[i-1][j- wt[i-1]] + val[i-1];

            dp[i][j]= max( sel, rej);
        }
    }
}
return dp[N][cap];
```

TC : O(n*cap)

SC : O(n*cap)

Unbounded Knapsack (0-N knapsack)

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 $\leq K$

Note 1 :- Every item can be picked infinite no. of times

Note 2 :- We can not take a part of item

Eg:- **N = 4 items** **K = 50** (→ cap)

Items:	0	1	2	3
Weight[]:	20	10	30	40
Value[]:	100	60	120	150

Ans = 300 ↓

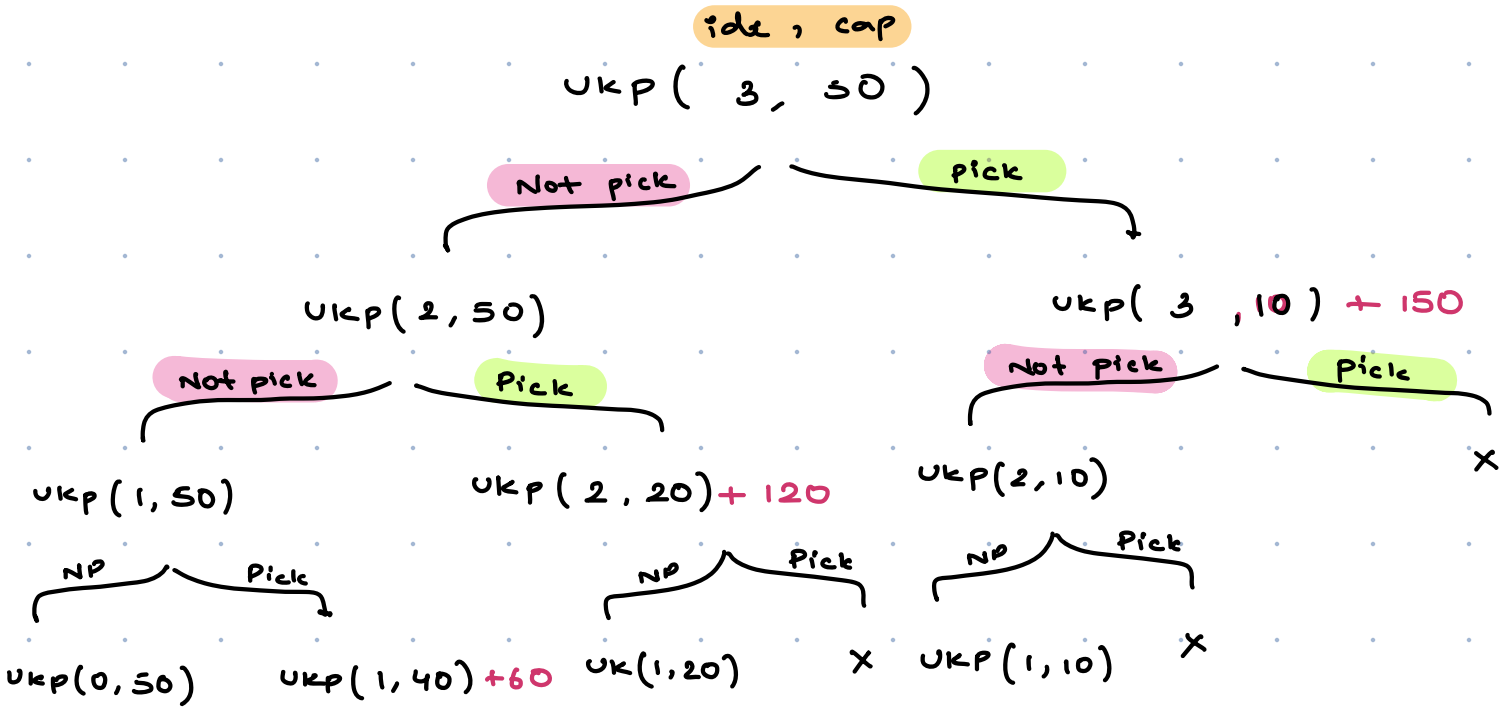
pick item with wt = 10, five times

weight[] = {1 50}

value[] = {1 30}

cap = 100

Ans = 100 + pick item with weight 1 100 times




```
int [][] dp = new int [n+1][cap+1];
for ( i=0; i<n; i++){
    for (j=0; j<n; j++) {
        if ( i==0 || j==0) dp[i][j]=0
        else {
            rej = dp[i-1][j]
            sel = 0
            if ( wt[i-1] ≤ j ){
                sel= val[i-1] + dp[ i ][j- wt[i-1]]
            }
            dp[i][j]= max( rej, sel);
        }
    }
}
return dp[N][cap];
```

TC : O(n*k)
SC : O(n*k)

SC : O(k)

Think on this