Current PSP 54% → 57%

Start at 9:05 PM

Agenda

1. Fractional Knapsack
2. 0-1 Knapsack
3. Unbounded Knapsack

Knapsack

Given N objects with their values V_i & their weight W_i. A bag is given with capacity W that can be used to carry some objects such that the total sum of objects weights W and sum of profits in the bag is maximized or sum of losses in the bag is minimized

try Knapsack when

1. every object has two attributes value / weight
2. You are given a maximum capacity

Q → Fractional Knapsack

Given N cakes with happiness and weight. Find maximum happiness that can be kept in bag with capacity C E cakes can be divided y

C = 40

	0	1	2	3	4
Happiness	3	8	10	2	5
Weight	10	4	20	8	15

happiness = ~~16~~ ~~18~~ 23

capacity = ~~30~~ ~~16~~ 1

Approach

Find the happiness / weight ratio

	0	1	2	3	4
Happiness	3	8	10	2	5
Weight	10	4	20	8	15
Hi / Wi	0.3	2	0.5	0.25	0.33

sort based on Hi/Wi →

	0	1	2	3	4
	8	10	5	3	2
	4	20	15	10	8
	2	0.5	0.33	0.3	0.25

Capacity = 40

Hi	Wi	Capacity	ans
8	4	36	8
10	20	16	18
5	15	1	23
3	10	0	23.3

→ Fractional Knapsack

Ans = 23.3

pseudo code

T.C = O(nlogn) S.C = O(n)

```
class Item {
    double cost;
    int weight;
    int happiness;
    Item ( h, w ) {
        this.happiness = h;
        this.weight = w;
        this.cost = h/w;
    }
}

// sort based on cost in desc

double ans = 0.0;
for (i=0; i<n; i++) {
    if ( item[i].weight <= capacity ) {
        ans += item[i].happiness;
        capacity -= item[i].weight;
    }
    else {
        ans += capacity * item[i].cost;
    }
}
```

Q → Flipkart's Upcoming Special Promotional Event

Flipkart is planning a special promotional event where they need to create an exclusive combo offer. The goal is to create a combination of individual items that together offer the highest possible level of customer satisfaction (indicating its popularity and customer ratings) while ensuring the total cost of the items in the combo does not exceed a predefined combo price.

Q1 → can this combo have partial item?

100 % No

Quiz 1

Key difference between Fractional & 0-1 Knapsack

Items can be partially included in Fractional

Items cannot be partial in 0-1 Knapsack

customer satisfaction → Happiness

Example

Capacity (C) = 7

Hi =	4	1	5	7
Wi =	3	2	4	5

will Hi/Wi work

No

Greedy does not work here

Brute force

Hi =	4	1	5	7
Wi =	3	2	4	5

consider dont consider

Generate all possible subset

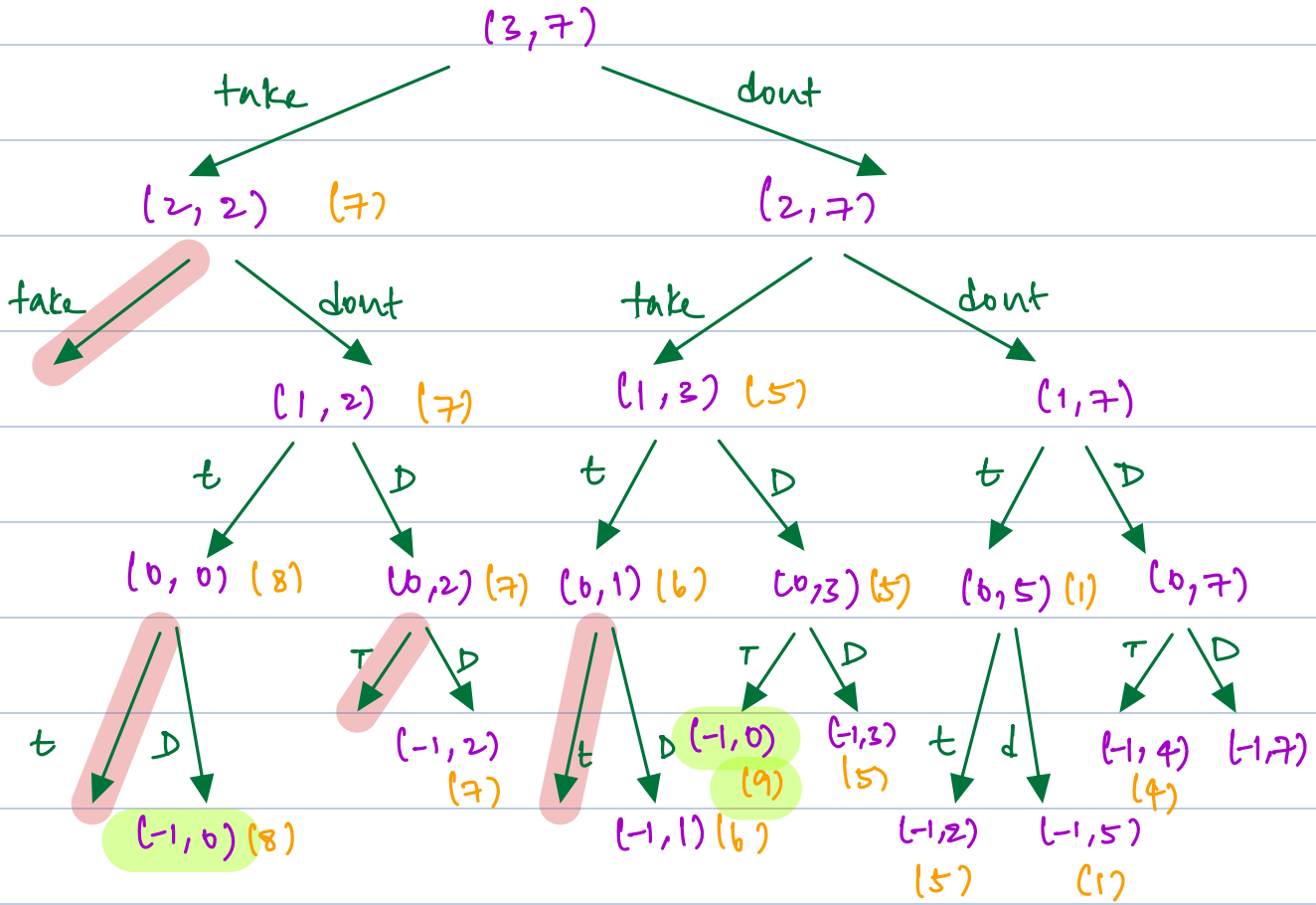
→ check if generate subset is within given capacity & store max happiness for all such subsets

Total subsets = 2^n

	0	1	2	3
Hi =	4	1	5	7
Wi =	3	2	4	5

Capacity (C) = 7

(index, capacity) → take / don't take

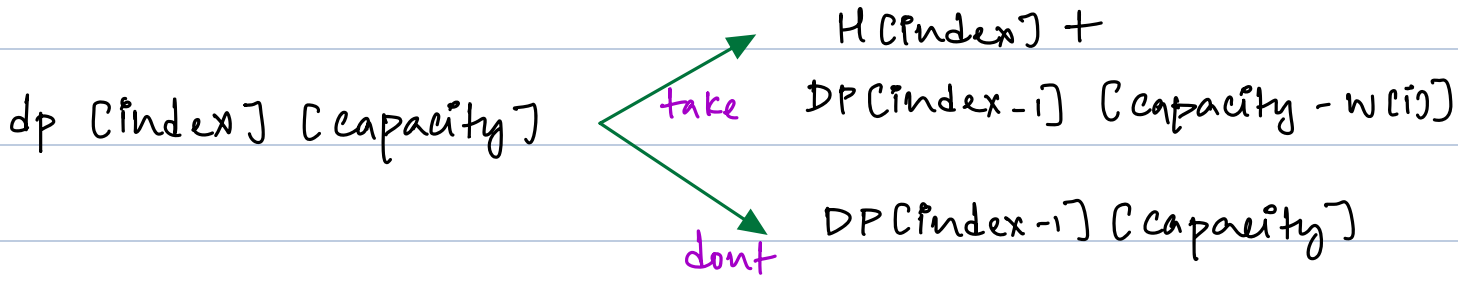


Optimal substructure → yes

Overlapping subproblem → yes

DP state → (index, capacity)

DP[N][C]



pseudo code

```
int [] [] DP = new int [N] [c];
int solve (H[], w[], index, capacity) {
    if (index < 0) return 0 // Base condition
    if (DP[in] [c..] != -1) return DP[in] [c]
    dont = solve (H, w, index-1, capacity)
    if (capacity >= w[index]) {
        take = solve (H, w, index-1, cap - w[i])
               + H[i]
    }
    else take = 0;
    Dp [index] [capacity] = max (take, dont);
    return DP[index] [capacity];
}
```

T.C = O(nc) S.C = O(nc)

10:20 pm ⟶ 10:30 pm

Unbounded Knapsack

① Objects cannot be divided

② You have infinite objects

N toys, happiness and weight. Maximize happiness for capacity C

Objects cannot be divided

You have infinite objects

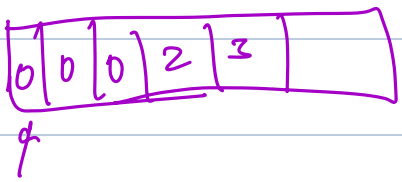
N = 3

C = 8

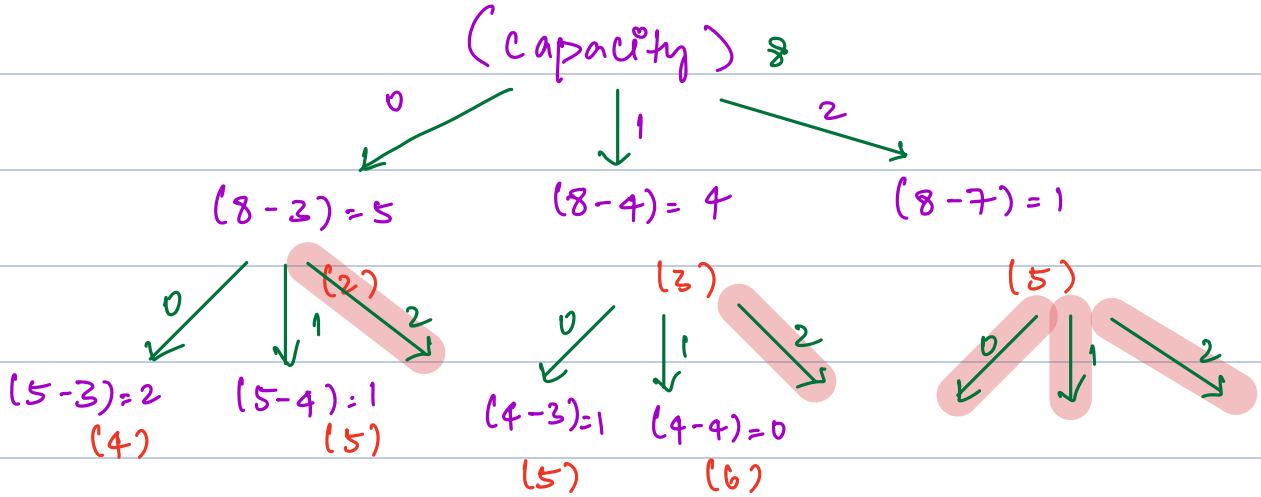
H = {2, 3, 5}

W = {3, 4, 7}

pick this object 2 times



Brute force



DP state

We are only worried about capacity

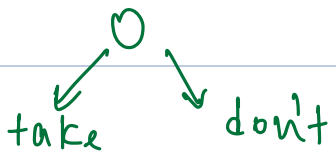
(c+1) states

pseudo code

```
Int [] dp = new Int [c+1];
// Initialize dp[i] = -1;
// smallest problem
dp[0] = 0;
for (i=1; i<= c; i++) {
    for (j= 0; j<n; j++) {
        if ( i >= w[j]) {
            dp[i] = max( dp[i], H[i] +
                         dp[i- w[j]])
        }
    }
}
DP[W] ✓
```

T.C = N * C S.C = O(C)

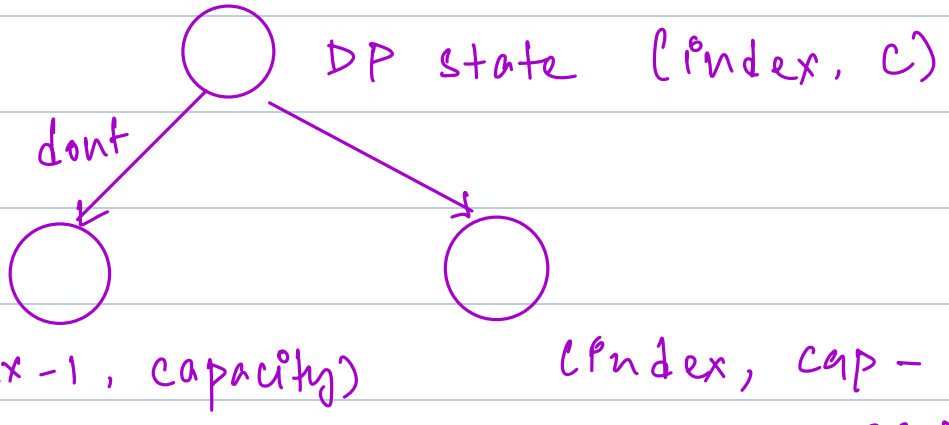
// Recurrence



```
int [] [] DP = new int [N] [c];
int solve (H[], W[], index, capacity) {
    if (index < 0) return 0  // Base condition
    if (DP[in] [c..] != -1) return DP[in] [c]
    dont = solve (H, W, index-1, capacity)
    if (capacity >= W[index]) {
        take = solve (H, W, index, cap - W[i])
                + H[i]
    }
    else take = 0;
    Dp [index] [capacity] = max (take, dont);
    return DP[index] [capacity];
}
```

T.C : N*C

S.C : N*C

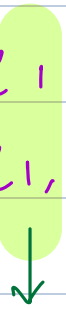


Quiz

C = 100

H = { 1, 80 }

W = { 1, 50 }



Happiness = 1 * 100

= 100

Contest ⟶ 26th April

Trees, Heaps, Greedy