Lecture :- Dynamic programming - 3

Agenda

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

fractional knapsack problem

Given n cakes with their happines and weight. find maximum total happiness that can be kept with capacity = W

Note: cakes can be divided

Example n = 5 and w = 40

happines of 5 cakes ⇒

3	8	10	2	5

weight of 5 cakes ⇒

10	4	20	8	15

Goal ⇒ happines should be maximum and total sum of weights should be <= 40.

Approach

Cake	Per unit happiness
Cake 1	$3/10 = 0.3$
Cake 2	$8/4 = 2$
Cake 3	$10/20 = 0.5$
Cake 4	$2/8 = 0.25$
Cake 5	$5/15 = 0.33$

	0	1	2	3	4
h ⇒	3	8	10	2	5
w ⇒	10	4	20	8	15

weight = 40

Solution

	0	1	2	3	4
h ⇒	3	8	10	2	5
w ⇒	10	4	20	8	15

Arrange cakes in des order on basis of $\frac{\text{happiness}}{\text{weight}}$ and start picking element.

	0	1	2	3	4
h ⇒	8	10	5	3	2
w ⇒	4	20	15	10	8

Dry run

i	happiness value [0]	remaining weight [40]
0	8	40 − 4 = 36
1	10	36 − 20 = 16
2	5	16 − 15 = 1
3	10 kg = 3 1 kg = $\frac{3}{10}$ = 0.3	1 − 1 = 0

8 + 10 + 5 + 0.3 = 23.3

Pseudocode

```
double getMaxHappiness( wt[], happiness[],
                        int weight ) {

    // create a cake array.
```

(3,10,0.3)	(8,4,2)	(10,20,0.5)	(2,8,0.25)	(5,15,0.33)

```
    Arrays.sort( cakes, new comparator() {
        @ Override
        public int compare( cake c1, cake c2) {
            if( c1.cost > c2.cost) {
                return -1;
            }
            return 1;
        }
    }

        double total = 0.0;
        for( i=0; i<n; i++) {
            weight = weight - cakes[i].w;
            total += cakes[i].h;

            // Do for partial
            // what if weight <= 0
        }

    return total;
}
```

```
class Cake {
    int wt;
    int h;
    double cost;
}
```

TC: O(nlogn)
SC: O(n)

Qu Given n items, each with a weight and a value, find maximum 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 $n \Rightarrow 4$ items

capacity (K) $\Rightarrow$ 50

	0	1	2	3
wt $\Rightarrow$	10	20	30	40
val $\Rightarrow$	100	60	120	150

Ans $\Rightarrow$ 0th and 2nd = 250

Idea 1 choose items with max value.

$n \Rightarrow 4$ items

capacity (K) $\Rightarrow$ 50

wt $\Rightarrow$

0	1	2	3
20	10	30	40

val $\Rightarrow$

100	60	120	150

Ans $\Rightarrow$ $150 + 60 = 210$ [wrong]

Idea 2

$n \Rightarrow 4$ items

capacity (K) $\Rightarrow$ 50

wt $\Rightarrow$

0	1	2	3
20	10	30	40

val $\Rightarrow$

100	60	120	150

price per unit $\Rightarrow$

5	6	4	3.75

Ans $\Rightarrow$ $60 + 100 = 160$ [wrong]

wt = $\cancel{50}$ $\cancel{40}$ 20

Idea 3 Dynamic programming

$n \Rightarrow 4$ items

capacity (k) $\Rightarrow$ 50

	0	1	2	3
wt $\Rightarrow$	10	20	30	40
val $\Rightarrow$	100	60	120	150

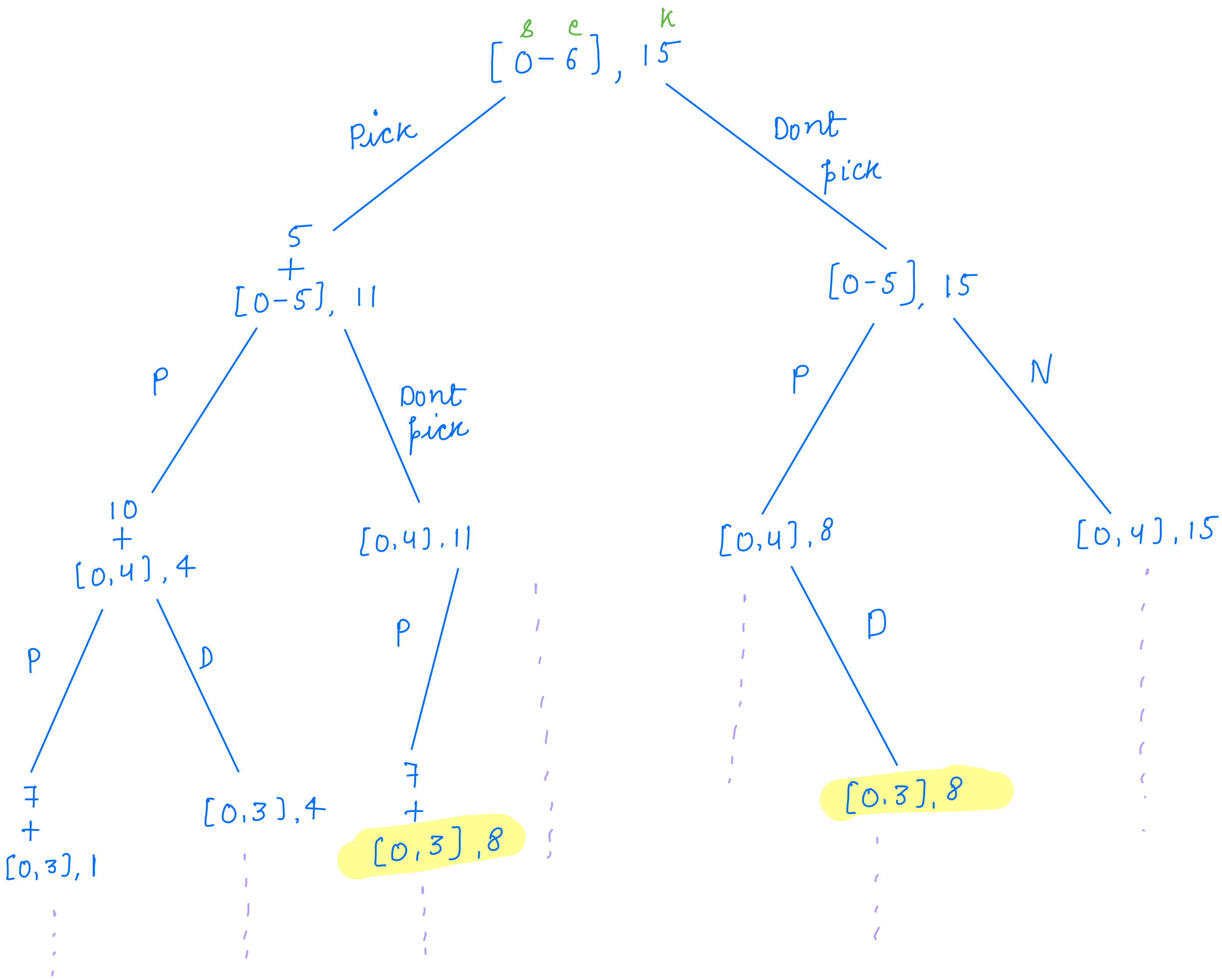
Ans $\Rightarrow$

T.C $\Rightarrow$

Idea 4

n ⇒ 7 items . capacity ⇒ 15

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



Overlapping subproblems

Changing factors

1. start idx X
2. end idx ✓
3. K ✓

dp[n+1] [k+1]

Recursive code

```
int 01Knapsack( wt[], val[] , int k , int end )  {
        if ( end ==0 )  {
            if ( wt[end] <= k ) {
                return  val[end];
            }
            return 0;
        }
        if ( k<=0 ) {
            return 0;
        }
        inc =  val[end] + 01knapsack(wt, val, k- wt[end],
                                        end-1);

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

        return max( inc, exc);
}
```

TC: 2^n

SC: $O(n)$

Memoised code

```
int 01KnapSack( wt[], val[], int k, int end, int[][] dp) {
    if (end == 0) {
        if (wt[end] <= k) {
            return val[end];
        }
        return 0;
    }
    if (k <= 0) {
        return 0;
    }
    if ( dp[end][k] != -∞) {
        return dp[end][k];
    }
    inc = val[end] + 01KnapSack(wt, val, k - wt[end],
                                end-1);

    exc = 01KnapSack(wt, val, k, end-1);
    dp[end][k] = max(inc, exc);
    return max(inc, exc);
}
```

Tabulative approach

dp[i][j] ⇒ Max profit you can get in a bag of capacity = K from [0–i] idx.

Example n = 5 and K = 8

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

K ⟶

val	wt		0	1	2	3	4	5	6	7	8
no item		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		
15	5	3	0	0	0	12	12	15	20		
6	2	4	0	0	6	12	12	18			
10	4	5	0	0	6	12					27

dp[3][6]

- Pick: 15 + [0–2], 1 ⇒ dp[2][1]
  15
- Dont pick: [0–2], 6
  dp[2][6]
  20

dp[4][5]

- P: 6 + [0,3], 3
  dp[3][3]
  12
  18
- D: [0,3], 5
  dp[3][5]
  15

dp[i][j]

- exc: dp[i-1][j]
- inc: dp[i-1][j − wt[i-1]] + val[i-1]

Tabulative code

```
int 01Knapsack( int[] wt, int[] val, int k) {
        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 {
                    exc = dp[i-1][j];
                    if (j-wt[i-1] >=0) {
                    inc = dp[i-1][j-wt[i-1]] + val[i-1];
                    }
                    dp[i][j] = max(exc, inc);
                }
            }
        }
} return dp[n][k];
```

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

Break: 8:44-8:54

Qu Given n items, each with a weight and a value, find maximum 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 item partially.

Example n ⇒ 4 items

capacity (K) ⇒ 50

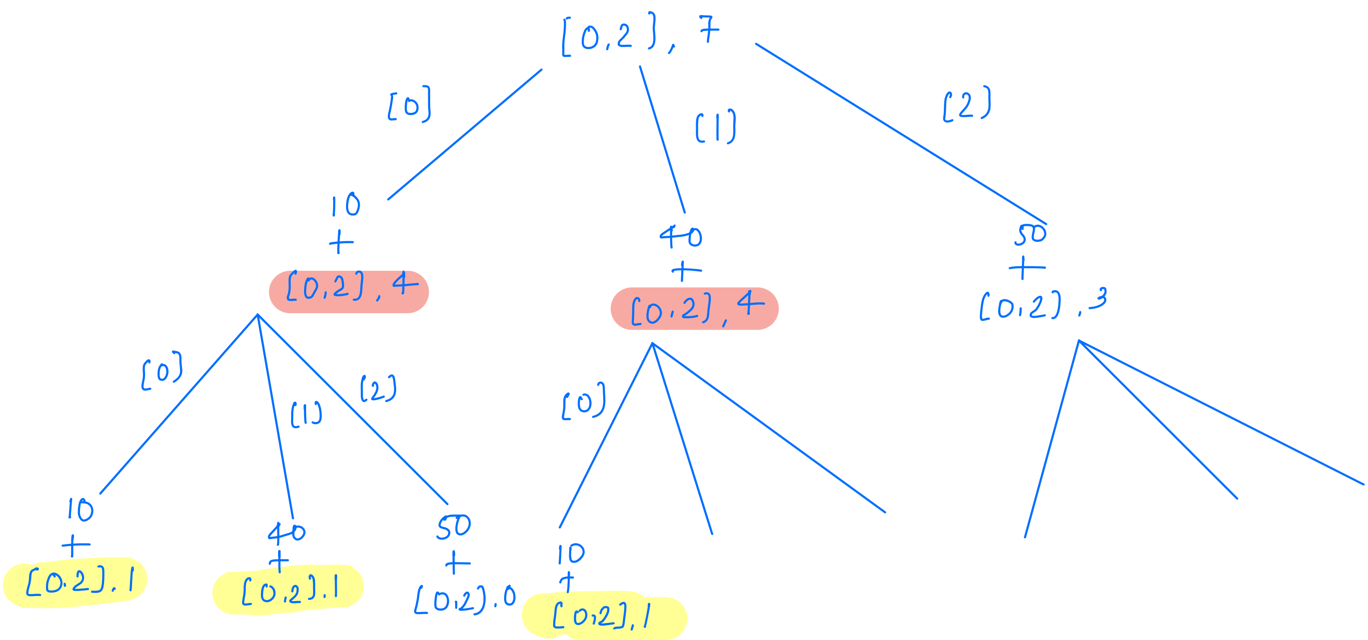
	0	1	2	3
wt ⇒	20	13	10	40
val ⇒	100	66	40	150

Ans ⇒ 100 + 100 + 40 = 240

Idea

K = 7

	0	1	2
val[] ⇒	10	40	50
wt[] ⇒	3	3	4



changing factor

s ✗
e ✗
K ✓

dp[K+1]

Overlapping subproblems

Optimal substructure

Tabulative approach

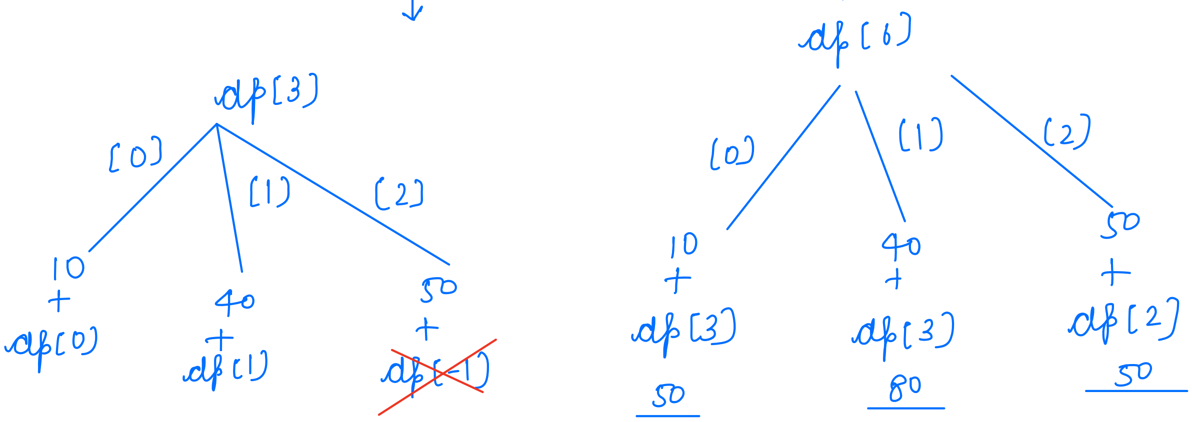
$dp[i] \Rightarrow$ max profit in a bag of capacity i

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



Pseudocode

```
int unboundedKnapsack(wt[], val[], K) {
        dp[K+1];
        for(i=1; i<=K; i++) {
                max = -∞;
                for(j=0; j<n; j++) {
                        if ( i - wt[j] >= 0) {
                        max = max(max, val[j] + dp[i - wt[j]]);
                        }
                }
                dp[i] = max;
        }
        return dp[K];
}
```

TC: $O(n*K)$

SC: $O(K)$

Thankyou :)