Dp knapsack 6
Friday, 8 July 2021 8:05 PM

Fractional: I can take 1 unit of any item
Greedy Algo

W = 50 kg

N =	1	2	3	4
W[i] :	20	10	20	40
V[i] :	100	60	120	150
V/W :	5	6	6	15/4

0-1 : {Either take one item Entirely or leave it}

Greedy: (Dynamic) *

1) Max V[i]: 4 : 3/60 : 210

2) Max V[i]/W : 2 : 1 remain = 20
We cannot pick item 3?
→ Insufficient space
: 160

⇒ N items, N Value, N weight [W = wt[n], W]
[Knapsack weight]

{ [N][W] = { Max value which can be obtained using first N items within knapsack weight ≤ W } }

dp[i][j] = { Max value which can be obtained using first i items, within knapsack weight ≤ j } [i-1]

Dp expression: i th knapsack weight j

max { (i-1, j) [V1] , (i-1, j - W[i]) + V[i] [V2] } remain weight

dp[i, j] = max { dp[i-1, j] ; dp[i-1, j - W[i]] + V[i] }
if: j - W[i] ≥ 0
j ≥ W[i]
optimal subproblem
If weight ≥ 0, dp expression will take care

dp[n][W] = ?

N = 12, K = 50
6 Elements
W[ ] : 5 3 8 2 7 4
V[ ] : . . . . . .
6 Elements
[dp[12][50]] max value
[dp[6][25]] max value
6 Elements *
Overlapping subproblem
Cons
Cons
[Overlapping sub problem]

Dp table: dp[N+1][W+1]
Base cond: j = 0 ; dp[0][j] = 0
TC: [N+1][W+1] → O(1) dp table
SC: O(N*W)
Final ans: dp[N][W]

{ If There are no items
We cannot get any value }

{ not needed } ∀ j=0 dp[i][0] = 0

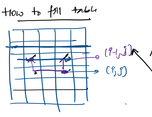
Eg: N = 5, W = 8 :

	1	2	3	4	5
W[i] :	3	6	5	2	4
V[i] :	12	20	15	6	10

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	12	12	12	12	12	12
2	0	0	0	12	12	12	20	20	20
3	0	0	0	12	12	15	20	20	27
4	0	0	6	12	12	18	20	21	27
5	0	0	6	12	12	18	20	22	28

dp[i][j] = max { dp[i-1][j] , dp[i-1][j-w[i]] + V[i] }
12 + 15

How to fill table



(i-1, j)
(i, j)

L → R
T → B

Note:
Space optimization:
It can be done in 2 rows:
→ We cannot know items which are picked

dp[i][j][W+1]
∀ j=0 dp[0][j] = 0

row = row 1
row → row 0
row → row I : I % 2
row → row 0 : 4 % 2

5th = 1 % 2

Base Cases

for (int i = 1; i < N; i++) {
  for (int j = 0; j < W; j++) {
    dp[i][j] = dp[i-1][j]
    if (j >= w[i]) { j: remaining
      dp[i][j] = Max(dp[i][j], dp[i-1][j - w[i]] + V[i])
    }
  }
}

w[i]: Weight of ith item

Final Ans: dp[N][W]

for (int i = 1; i < N; i++) {
  for (int j = 0; j < W; j++) {
    dp[i%2][j] = dp[(i-1)%2][j]
    if (j >= w[i]) {
      dp[i%2][j] = max(dp[i%2][j], dp[(i-1)%2][j - w[i]] + V[i])
    }
  }
}

weight remain
value earned

Final Ans:
dp[N%2][W]

SC: O(N)

⇒ dp[i][0] → dp[0][i] ✓
1 >= w[i] * | dp[i][1] → dp[0][1] ✓
3 >= w[i] ✓
dp[0][0] + V[i]

dp[i][j] → dp[0][j] ✓
4 >= w[i] ✓
⇒ dp[0][0] + V[i] | dp[i][7] → dp[0][7] ✓
7 >= w[i] ✓
dp[0][4] + V[i]

dp[2][6] → dp[1][6] = 12
6 >= w[2] ✓
dp[1][0] + V[2] 20 | dp[2][5] → 12
5 >= w[2] ✓
dp[1][0] + V[2] 15

dp[3][6] → dp[2][6] = 20
6 >= w[3] ✓
dp[2][0] + V[3] = 15

(Point items taken) → (Point after 1 an) :
(i, j) → (i-1, j) ; (i-1, j - w[i]) + V[i]

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	0	12	12	12	12	12	12
2	0	0	0	12	12	12	20	20	20
3	0	0	0	12	12	15	20	20	27
4	0	0	6	12	12	18	20	21	27
5	0	0	6	12	12	18	20	22	28

// (i, j) = (i-1, j)
you are not picking ith item
// (i, j) = (i-1, j - w[i]) + V[i]
you are picking ith item

(5, 8) = (4, 8) = (3, 8) = (2, 3) + V[3] = (1, 3) = (0, 0) + V[1]
-5 -3 12
{ not picking 5th element } { not pick 4th } 3rd picked ✓ not pick 2nd picking 1st items
12 + 15 = 27

⇒ { Based on this, we can get elements we choosed }

int <int> i,
i = N, j = W
while (i > 0 && j > 0)
  if (dp[i][j] == dp[i-1][j]) {
    // we doesn't pick
    i--
  }
  else {
    // add (i)
    i--
    j = j - w[i]
  }

i → unbounded
0 - ∞ knap
We skip i..

TC: O(N+W) SC: O(N)
Dream of 1st

⇒ [0 - ∞] : knapsack ⇒ { Pick an element as many number of times }

dp[i][j] = { Max value which can be obtained using (first i) items & over weight ≤ j, [No restriction how many times we can pick] }

dp[i][j] :
j = 50, w[i] = 7
How many times can we pick i = ?
→ j/w[i]

leave : dp[i-1][j]
Pick 1/k : dp[i-1][j - 1*w[i]] + V[i]
Pick 2 of k : dp[i-1][j - 2*w[i]] + 2*V[i]
Pick 3 of k : dp[i-1][j - 3*w[i]] + 3*V[i]
Pick n of k : dp[i-1][j - n*w[i]] + n*V[i]
n = j/w[i]

dp[i][j] = max { ∀ k=0 j/w[i] dp[i-1][j - k*w[i]] + V[i]*k }
↳ Dp expression

TC: No of state * TC for each state
(N*W) * (N)

TC: (N)(W)(N)

N = 3, K = 100
w[] = 5
V[] = 4
dp[3][100] = max
dp[2][100] ← no pick
dp[2][95] + 4 1 pick
dp[2][90] + 8 2 pick
dp[2][85] + 12 3 pick
dp[2][80] + 16 4 pick
dp[2][0] + 80 20 pick

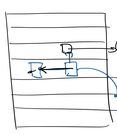
{ unbounded }
[0 - ∞] dp expression

dp[i][j] = max { dp[i-1][j] , skip
After pick ith → stay there
j -= w[i]
dp[i][j - w[i]] + V[i] }

Both are done

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

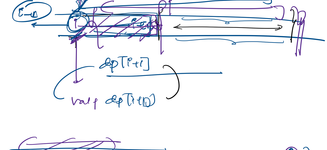
dp[3][100] = max
w[3] = 5
V[3] = 4
dp[3][100] ✓
4 + dp[3][95]
dp[2][95]
Cons 1
4 + dp[3][90]
dp[2][90]
Cons 2
4 + dp[3][85]



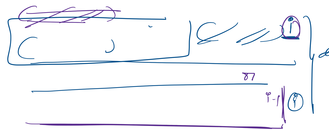
(i-1, j)
If we going backwards we are picking ith elem

States:
(N*n) * O(2)
{ TC: N*n
SC: O(N*n) }

N trips 2 * 0



dp[i+1]
dp[i+1]
val + dp[i-1][j]



dp[i] = dp[i-1] + val[i]
dp[i-1]