Design and Analysis of Algorithms

Lecture-35

Dharmendra Kumar (Associate Professor)
Department of Computer Science and Engineering
United College of Engineering and Research,
Prayagraj

0-1 Knapsack using dynamic programming

- Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W.

- Based on the nature of the items, Knapsack problems are categorized as
  1. Fractional Knapsack
  2. 0-1 Knapsack

0-1 Knapsack using dynamic programming

Fractional Knapsack

In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction $\boldsymbol{x_i}$ of $\mathbf{i}^{\mathbf{th}}$ item.

0-1 Knapsack

In this version of Knapsack problem, we cannot break an item, either pick the complete item or don't pick it (0-1 property).

0-1 Knapsack using dynamic programming

Dynamic programming approach for solving 0-1 Knapsack problem is explained as following:-

Let c[i,w] denotes the value of the solution for items 1,2,3,….,i and maximum weight w.

It is defined as

$$
c[i,w] = \begin{cases} 0 & \text{, if } i = 0 \text{ or } w = 0 \\ c[i-1, w] & \text{, if } i > 0 \text{ and } w_i > w \\ \max\{ v_i + c[i-1,w-w_i], c[i-1,w] \} & \text{, if } i > 0 \text{ and } w_i \leq w \end{cases}
$$

0-1 Knapsack using dynamic programming

Dynamic programming algorithm for solving 0-1 Knapsack problem is the following:-

Dynamic-0-1-Knapsack(v, w, n, W)

```
for l← 0 to W
        c[0,l] ← 0
for i← 1 to n
        c[i,0] ← 0
        for l← 1 to W
                if w_i ≤ l then
                        if v_i + c[i-1,l-w_i] > c[i-1,l] then
                                c[i,l] ← v_i + c[i-1,l-w_i]
                        else
                                c[i,l] ← c[i-1,l]
                else
                        c[i,l] ← c[i-1,l]
return c
```

Time complexity of this algorithm = $O(nW)$

0-1 Knapsack using dynamic programming

Example: Solve the following 0-1 knapsack.

$n = 4,\ W = 5$

$(v_1, v_2, v_3, v_4) = (3, 4, 5, 6)$

$(w_1, w_2, w_3, w_4) = (2, 3, 4, 5)$

Solution:

Here, we have to calculate c[4, 5].

$c[4,5] = \max\{ v_4 + c[3,0], c[3,5] \} = \max\{ 6 + 0, c[3,5] \}$

$= \max\{ 6, c[3,5] \} = \max\{ 6, 7\} = \mathbf{7}$

$c[3,5] = \max\{ v_3 + c[2, 1], c[2, 5] \} = \max\{ 5 + 0, c[2, 5] \}$

$= \max\{ 5, c[2, 5] \} = \max\{ 5, 7\} = 7$

$c[2,5] = \max\{ v_2 + c[1, 2], c[1, 5] \} = \max\{ 4 + 3, 3\} = 7$

Therefore, the value of the optimal solution is 7. And the optimal solution is (1, 2).

0-1 Knapsack using dynamic programming

Solution by Tabular Method

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

This table is used to find the value of the optimal solution.

Therefore, value of optimal solution = 7

0-1 Knapsack using dynamic programming

Solution by Tabular Method

	1	2	3	4	5
1	↑	←	←	←	←
2	↑	↑	←	←	←
3	↑	↑	↑	←	↑
4	↑	↑	↑	↑	↑

This table is used to find the optimal solution.

Therefore, optimal solution = (1, 2)

0-1 Knapsack using dynamic programming

Example: Solve the following 0-1 knapsack.

$n = 6, \ W = 100$

$(v_1, v_2, v_3, v_4, v_5, v_6) = (40, 35, 20, 4, 10, 6)$

$(w_1, w_2, w_3, w_4, w_5, w_6) = (100, 50, 40, 20, 10, 10)$

Solution:

	0	10	20	30	40	50	60	70	80	90	100
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	40
2	0	0	0	0	0	35	35	35	35	35	40
3	0	0	0	0	20	35	35	35	35	55	55
4	0	0	4	4	20	35	35	39	39	55	55
5	0	10	10	14	20	35	45	45	49	55	65
6	0	10	16	16	20	35	45	51	51	55	65

0-1 Knapsack using dynamic programming

From table in the previous slide, the value of optimal solution will be 65.

Following is the table to compute the optimal solution.

The optimal solution will be (2, 3, 5).

	10	20	30	40	50	60	70	80	90	100
1	↑	↑	↑	↑	↑	↑	↑	↑	↑	←
2	↑	↑	↑	↑	←	←	←	←	←	↑
3	↑	↑	↑	←	↑	↑	↑	↑	←	←
4	↑	←	←	↑	↑	↑	←	←	↑	↑
5	←	←	←	↑	↑	←	←	←	↑	←
6	↑	←	←	↑	↑	↑	←	←	↑	↑