More on Unbounded Knapsack Problem

Motive

After reading the research "A polynomially solvable special case of the unbounded knapsack problem" by Moshe Zukerman, Long Jia, Timothy Neame and Gerhard J. Woeginger, I find Unbounded Knapsack Problem quite interesting. Hence I would like to share what I've learned about this model of Knapsack Problem.

Introduction

Unbounded Knapsack Problem (UKP) is one of many modeled version of the original Knapsack Problem. The difference from 0/1 Knapsack Problem (0/1 KP) is that there are no limits to the number of items available. In other words, as along the sum is within weight limit, you can take as many copies of each item as possible.

Both UKP and 0/1 KP are fundamental problems in combinatorial optimization and have practical applications in real life. Whilst 0/1 KP handles binary decision scenarios, such as profit maximization within budget constraint, UKP handles scenarios where resources are unlimited, such as production planning.

Approach

I will introduce the formula and concept of UKP. Then I will do an example on both 0/1 KP and UKP with dynamic programing. Result tables are included whilst calculations are omitted for sake of simplicity, though C++ codes are used to reaffirm accuracy.

Formulation

Let:
n be number of items
w_i be the weight of the i-th item
v_i be value of i-th item
C be the knapsack capacity
dp be a 2D array where $dp[i][j]$ is maximum value

Formula

$$dp[i][j] = \max(dp[i-1][j], dp[i][j-w_i] + v_i)$$

Formula Explanation

$dp[i-1][j]:$ Maximum value obtained not including i-th item
$dp[i][j-w_i] + v_i:$ Maxumum value obtained including i-th item.
Subtract weight w_i from current capacity j

Example

Using the example from lecture notes p.6, I'll solve both 0/1 KP and UKP with dynamic programming. Then use C++ to prove accuracy and compare the two KP's.

Sample Question

Given N = 5, W = 6. The table is as follow.

Item	Weight	Value
1	3	4
2	1	2
3	2	3
4	3	5
5	1	1

Case 1: 0/1 KP

Dynamic Programming

By applying the formula of 0/1 KP Dynamic Programming given in lecture notes:

$$V[i][w] = \max(V[i-1][w], V[i-1][w-w_i] + v_i)$$

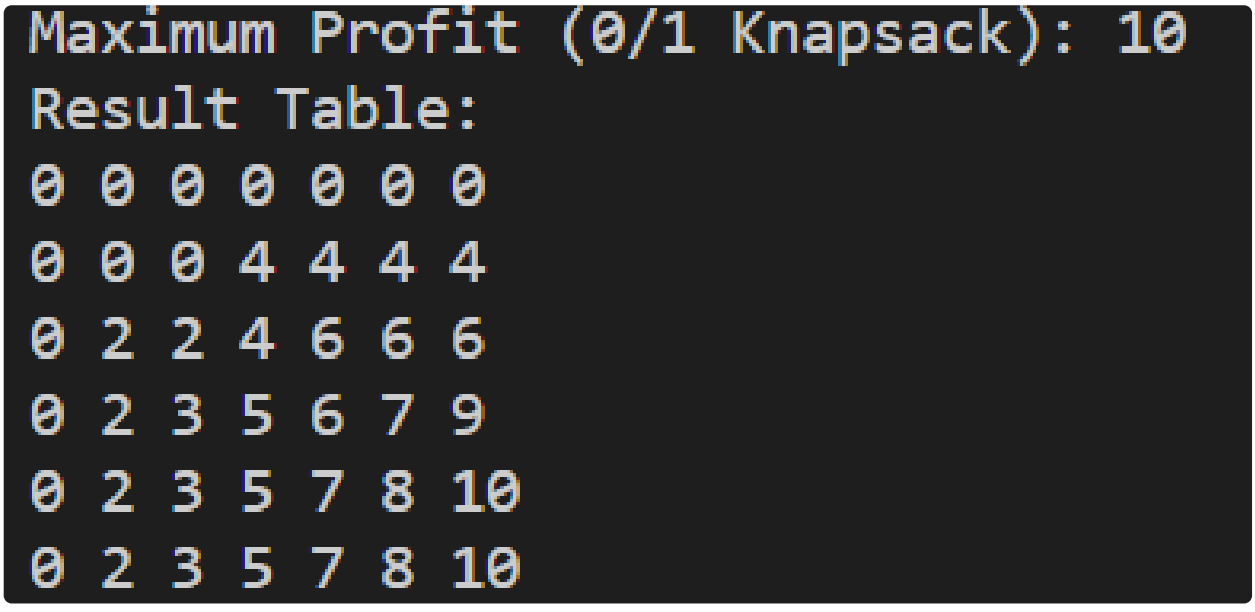
we can get the following table.

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	4
2	0	2	2	4	6	6	6
3	0	2	3	5	6	7	9
4	0	2	3	5	7	8	10
5	0	2	3	5	7	8	10

The answer is coherent with lecture notes.

Coding

I also used C++ codes to confirm accuracy. Codes are in attachment. Below is the output.



Case 2: UKP

Dynamic Programming

By applying the formula defined above, we can get the following table.

	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	0	4	4	4	8
2	0	2	4	6	8	10	12
3	0	2	4	6	8	10	12
4	0	2	4	6	8	10	12
5	0	2	4	6	8	10	12

The answer is larger than that in lecture notes. Logically it makes sense, as there are less limitations.

Remark: Note that results of UKP may not always be larger than that of 0/1 KP.

Coding

To convert the code for 0/1 KP to solve UKP, there is one crucial change. Instead of considering $dp[i-1][j-wt[i-1]]$, it's modified to $dp[i][j-wt[i-1]]$ to represent the possibility of using the same item multiple times.

The new maximum value and code output is as follow:

```
Maximum Profit (Unbounded Knapsack): 12
Result Table:
0 0 0 0 0 0 0
0 0 0 4 4 4 8
0 2 4 6 8 10 12
0 2 4 6 8 10 12
0 2 4 6 8 10 12
0 2 4 6 8 10 12
```

Reference

M. Zukerman, L. Jia, T. D. Neame and G. J. Woeginger, “A polynomially Solvable special case of the unbounded knapsack problem”, Operations Research Letters, vol. 29, no. 1, August 2001, pp. 13-16.

ChatGPT, https://chat.openai.com/

Knapsack calculator. (n.d.). Augustine Aykara. https://augustineaykara.github.io/Knapsack-Calculator/

(n.d.). Stanford University. https://web.stanford.edu/class/archive/cs/cs106b/cs106b.1228/section/section4/Knapsack%20Problem%20(Optional%20Section%20Slides).pdf