Chapter 9

Dynamic Programming

The Knapsack problem:

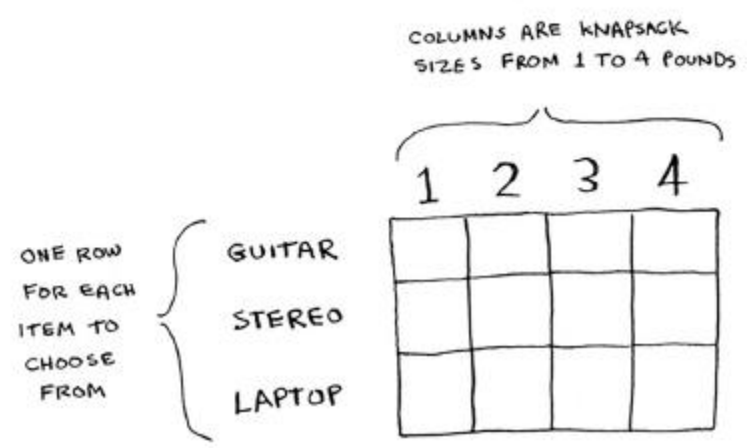
You have a knapsack that can carry up to 4 kg, you want to fill it with items of the highest possible value, you have 3 item to choose from:

1. Stereo -> size = 4 kg, price = 3000$
2. Laptop -> size = 3 kg, price = 2000$
3. Guitar -> size = 1 kg, price = 1500$

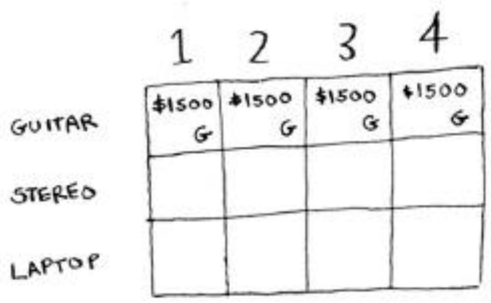
You can find the answer by getting each possible set but that consumes a lot of time with time complexity O(2^n), another way is to find the answer using greedy algorithm, but it would be an approximation, to find our answer we need **Dynamic Programming**.

Dynamic Programming:

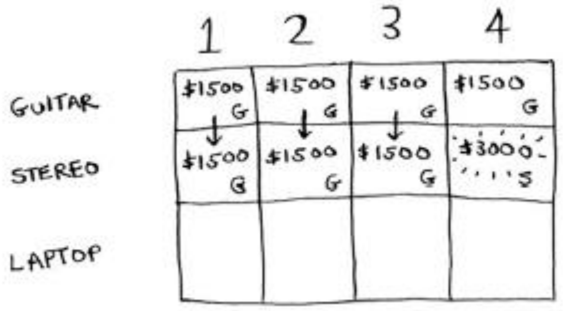
Dynamic programming starts by solving subproblems and builds up to solving the big problem. For the knapsack problem, you'll start by solving the problem for smaller knapsacks (or "sub-knapsacks") and then work up to solving the original problem. Every dynamic-programming algorithm starts with a grid. Here's a grid for the knapsack problem:


COLUMNS ARE KNAPSACK SIZES FROM 1 TO 4 POUNDS
1
2
3
4
ONE ROW FOR EACH ITEM TO CHOOSE FROM
GUITAR
STEREO
LAPTOP

Let's start with the first row, you want an answer for the question do you take the guitar or not, so for each sub-knapsack find if the Guitar fits or not if it does put the price of the guitar in the corresponding cell (for first row only)

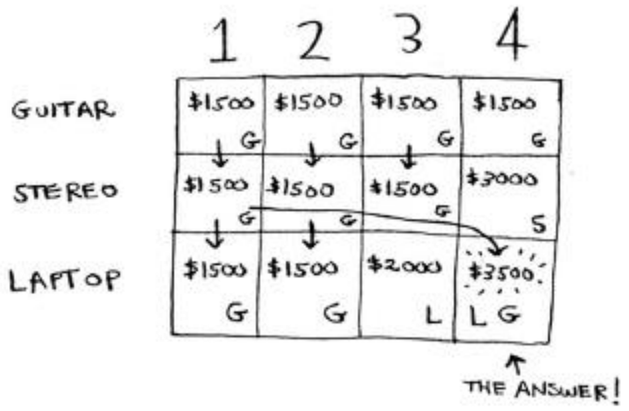


For the next row you ask yourself do you take the stereo or the guitar or both of them? You need to find if th stereo fits in the sub-Knapsack if it doesn't then you take the guitar so right down the guitar price, if it fits exactly choose the one with the highest price and write it down



	1	2	3	4
GUITAR	$1500 G	$1500 G	$1500 G	$1500 G
STEREO	$1500 G	$1500 G	$1500 G	$3000 S
LAPTOP				

What if it fits but there is extra space? You take the price of the item and add it to the price written in the sub-knapsack that has the same size of the space from the previous row, compare the added value to the previous maximum value in this column and write down the bigger



	1	2	3	4
GUITAR	$1500 G	$1500 G	$1500 G	$1500 G
STEREO	$1500 G	$1500 G	$1500 G	$3000 S
LAPTOP	$1500 G	$1500 G	$2000 L	$3500 L G

Notes:

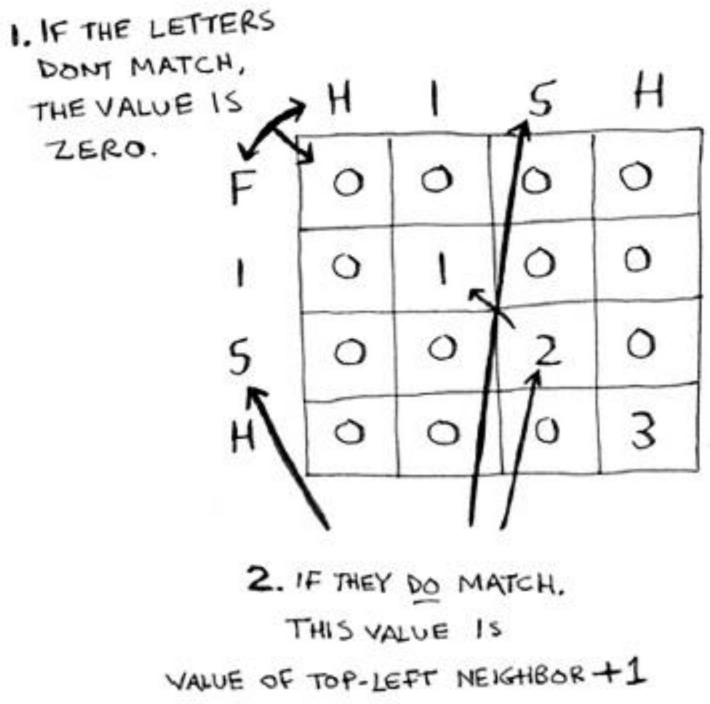
- If you add extra items, the final value may or may not change.
- The order of the rows doesn't affect the final value.
- If you fill the grid column-wise instead of row-wise it will not make a difference in this problem, but it may make a difference in other problems.
- If you add an item with smaller size (ex: 0.5 kg) you need to add columns for smaller sub-knapsacks (0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4)
- Dynamic programming doesn't work with fraction of an item (ex: half kg from a sack of rice) you either take the whole item or leave it.
- Dynamic programming only works when the subproblems are discrete and don't depend on other subproblems.
- The final solution will always consist of maximum of two sub-knapsack but each of them may contain another sub-knapsack.
- It is not necessary to fill the whole knapsack if it contains the highest value.

Longest common substring:

Dynamic programming is useful when you try to optimize the output given a constraint.

If you misspelled a word how does the computer know the correct word? It compares the word with thousands other words using dynamic programming with the goal to optimize the number of similar letters, if the letters are different put 0 else if it is the first letter put 1 otherwise put the number in the top left cell+1.

Comparing Hish (the misspelled) with fish (one of the possible correct words):


1. IF THE LETTERS DONT MATCH, THE VALUE IS ZERO.
H I S H
F I S H
0 0 0 0
0 1 0 0
0 0 2 0
0 0 0 3
2. IF THEY DO MATCH, THIS VALUE IS VALUE OF TOP-LEFT NEIGHBOR +1