Imperial College London – Department of Computing

MSc in Computing Science

580: Algorithms
Tutorial: Dynamic Programming

1. A thief can carry k kilograms of loot in his *knapsack*. He robs a shop containing N items. Item i is worth b_i bitcoin and weighs k_i kilos. The thief wants to decide which items to take to maximise the total value he steals.

   (a) How would you decompose this problem into subproblems? Does the problem have optimal substructure and overlapping subproblems?

   (b) Write an algorithm that, given an array B such that $B[i]$ is the value of item i and an array K such that $K[i]$ is the weight of item i, and a maximum weight k, solves the problem in $O(kN)$ time.

   (c) Since he is greedy, the thief attempts to use the following strategy: the next item chosen should always be the one with the greatest value per kilogram, from those remaining. Show that this strategy is not guaranteed to give the optimal solution.

Answer:

(a) First, it is important to note a difference between this problem and the rod cutting problem. In this problem, each item can only be stolen *once*. So, item i can only be used once in a solution, whereas you can make as many rods of length i as you like, within the total of N. So, this problem (and its subproblems) has two dimensions: the space available in the knapsack, and the items (still) available. So, the solution you are looking for is $V(N, k)$, the maximum value that can be achieved for a knapsack of size k, by choosing from items 1 to N.

A possible trap is to start with the problem of finding $V(N, k)$ and divide this into one possibility for each available item: $b_i + V(?, k - k_i)$. This is awkward to implement because you have to track which items are still available somehow (hence the "?"). It also duplicates options by considering the items in all possible orders. When considering item N, there are just two possibilities. Either you take item N, giving a possible solution $b_N + V(N - 1, k - k_N)$, or you do not giving a possible solution $V(N - 1, k)$. There is no need to separately consider picking item 1 or item 2 first, because these items can still be chosen within the second option $V(N - 1, k)$.

So, $V(i, w)$, the maximum value that can be gained if there is room for w kilos in the knapsack and items 1 to i are available is:

$$V(i, w) = max(b_i + V(i - 1, w - k_i), V(i - 1, w))$$

Including base cases and edge cases we have:

- $V(i, 0) = 0, \quad$ for all i
- $V(0, w) = 0, \quad$ for all w
- $V(i, w) = V(i-1, w), \quad$ if $k_i > w$
- $V(i, w) = \max(V(i-1, w), V(i-1, w-k_i) + b_i), \quad$ otherwise

and the overall solution is $V(N, k)$. The optimal solution is defined using optimal solutions to subproblems, and these subproblems occur multiple times, so the problem has optimal substructure and overlapping subproblems.

(b)

```
1:  procedure KNAPSACK(B = [B_1, ..., B_N], K = [K_1, ..., K_N], k)
2:      V = new array[N + 1][k + 1]
3:      for i = 0 to N do
4:          V[i][0] = 0
5:      end for
6:      for i = 0 to k do
7:          V[0][i] = 0
8:      end for
9:      for i = 1 to N do
10:         for w = 1 to k do
11:             if K[i] > w then
12:                 V[i][w] = V[i − 1][w]
13:             else
14:                 V[i][w] = MAX(B[i] + V[i − 1][w − K[i]],V[i − 1][w])
15:             end if
16:         end for
17:     end for
18:     return V[N][k]
19: end procedure
```

(c) Since the strategy must always work, it is sufficient to find a single *counter example* to show that it is incorrect. So, if the shop contains three items with weights and values as follows:

i	1	2	3
k_i	5	10	12
b_i	5	10	13

and the thief's knapsack can hold 15 kg, then choosing the item with the greatest value per kg (item 3) will give a total value of 13, whereas choosing the other 2 items together will give a value of 15. So, the strategy is not optimal.