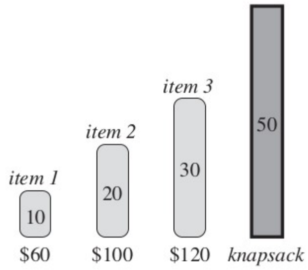

0-1 Knapsack problem

- n items available
- item i is worth v_i and weighs w_i pounds (both integer)
- capacity of backpack W, integer

Construct a dynamic programming solution when the thief can either choose to take an item (1) or not (0). It should run in $O(nW)$ time.

Build up a table c[0:n, 0:W] with entry c[i, w] representing the value of the optimal solution using items 1:i for a knapsack of capacity w with W = 5. Then write an algorithm that computes this table.

In this example (dividing weights and costs by 10):

c =
[0 0 0 0 0 0 0
0 6 6 6 6 6 6
0 6 10 16 16 16 16
0 6 10 16 18 22 28]

DYNAMIC-0-1-KNAPSACK(v, w, n, W)
  let $c[0..n, 0..W]$ be a new array
  for $w = 0$ **to** W
    $c[0, w] = 0$
  for $i = 1$ **to** n
    $c[i, 0] = 0$
    for $w = 1$ **to** W
      if $w_i \leq w$
        if $v_i + c[i-1, w-w_i] > c[i-1, w]$
          $c[i, w] = v_i + c[i-1, w-w_i]$
        else $c[i, w] = c[i-1, w]$
      else $c[i, w] = c[i-1, w]$

To reconstruct the solution, trace back from c[n,W]. When looking at entry c[i, w], compare with c[i-1, w]. If these are different, the optimal solution uses item i; next look at c[i-1, w-w_i]. If not, move to c[i-1, w].

Prove that the *fractional* knapsack problem has the greedy-choice property. Try “cut-and-paste”: assume some other solution is given, then show that picking the greedy choice can only yield more in the knapsack.

In the fractional problem, we pick an item based on the ratio of cost per weight $r = v / w$. Assume that some other non-greedy solution is given. Then there is an item (or some amount of some item) that’s taken with lower r that we can replace with an item with larger r. This only makes the solution better, and we can keep doing this until we arrive at the greedy solution. Thus greedy is optimal.

The reason why this argument doesn’t work for 0/1 is that you can’t take fractions of an item in that case. For the above solution to always work, you’d need fractions in some cases. (Construct a counterexample if you aren’t sure.)