Knapsack problem/0-1

Source: https://rosettacode.org/wiki/Knapsack_problem/0-1

A tourist wants to make a good trip at the weekend with his friends. They will go to the mountains to see the wonders of nature, so he needs to pack well for the trip. He has a good knapsack for carrying things, but knows that he can carry a maximum of only 4kg in it and it will have to last the whole day.

He creates a list of what he wants to bring for the trip but the total weight of all items is too much. He then decides to add columns to his initial list detailing their weights and a numerical value representing how important the item is for the trip.

Here is the list:

Table of potential knapsack items

item	weight (dag)	value
map	9	150
compass	13	35
water	153	200
sandwich	50	160
glucose	15	60
tin	68	45
banana	27	60
apple	39	40
cheese	23	30
beer	52	10
suntan cream	11	70
camera	32	30
T-shirt	24	15
trousers	48	10
umbrella	73	40
waterproof trousers	42	70
waterproof overclothes	43	75
note-case	22	80
sunglasses	7	20
towel	18	12
socks	4	50
book	30	10
knapsack	≤400 dag	?

The tourist can choose to take any combination of items from the list, but only one of each item is available. He may not cut or diminish the items, so he can only take whole units of any item.

Task: Show which items the tourist can carry in his knapsack so that their total weight does not exceed 400 dag [4 kg] and their total value is maximized.

Brute force algorithm

Code:

```
from itertools import combinations

def anycomb(items):
    ' return combinations of any length from the items '
    return ( comb
             for r in range(1, len(items)+1)
             for comb in combinations(items, r)
             )

def totalvalue(comb):
    ' Totalise a particular combination of items'
    totwt = totval = 0
    for item, wt, val in comb:
        totwt  += wt
        totval += val
    return (totval, -totwt) if totwt <= 400 else (0, 0)

items = (
    ("map", 9, 150), ("compass", 13, 35), ("water", 153, 200), ("sandwich", 50, 160),
    ("glucose", 15, 60), ("tin", 68, 45), ("banana", 27, 60), ("apple", 39, 40),
    ("cheese", 23, 30), ("beer", 52, 10), ("suntan cream", 11, 70), ("camera", 32, 30),
    ("t-shirt", 24, 15), ("trousers", 48, 10), ("umbrella", 73, 40),
    ("waterproof trousers", 42, 70), ("waterproof overclothes", 43, 75),
    ("note-case", 22, 80), ("sunglasses", 7, 20), ("towel", 18, 12),
    ("socks", 4, 50), ("book", 30, 10),
    )
bagged = max( anycomb(items), key=totalvalue) # max val or min wt if values equal
print("Bagged the following items\n  " +
      '\n  '.join(sorted(item for item,_,_ in bagged)))
val, wt = totalvalue(bagged)
print("for a total value of %i and a total weight of %i" % (val, -wt))
```

Dynamic programming algorithm

Code:

```
try:
    xrange
except:
    xrange = range

def totalvalue(comb):
    ' Totalise a particular combination of items'
    totwt = totval = 0
    for item, wt, val in comb:
        totwt  += wt
        totval += val
    return (totval, -totwt) if totwt <= 400 else (0, 0)

items = (
    ("map", 9, 150), ("compass", 13, 35), ("water", 153, 200), ("sandwich", 50, 160),
    ("glucose", 15, 60), ("tin", 68, 45), ("banana", 27, 60), ("apple", 39, 40),
    ("cheese", 23, 30), ("beer", 52, 10), ("suntan cream", 11, 70), ("camera", 32, 30),
    ("t-shirt", 24, 15), ("trousers", 48, 10), ("umbrella", 73, 40),
    ("waterproof trousers", 42, 70), ("waterproof overclothes", 43, 75),
    ("note-case", 22, 80), ("sunglasses", 7, 20), ("towel", 18, 12),
    ("socks", 4, 50), ("book", 30, 10),
    )

def knapsack01_dp(items, limit):
    table = [[0 for w in range(limit + 1)] for j in xrange(len(items) + 1)]

    for j in xrange(1, len(items) + 1):
        item, wt, val = items[j-1]
        for w in xrange(1, limit + 1):
            if wt > w:
                table[j][w] = table[j-1][w]
            else:
                table[j][w] = max(table[j-1][w],
                                  table[j-1][w-wt] + val)

    result = []
    w = limit
    for j in range(len(items), 0, -1):
        was_added = table[j][w] != table[j-1][w]

        if was_added:
            item, wt, val = items[j-1]
            result.append(items[j-1])
            w -= wt

    return result


bagged = knapsack01_dp(items, 400)
print("Bagged the following items\n  " +
      '\n  '.join(sorted(item for item,_,_ in bagged)))
val, wt = totalvalue(bagged)
print("for a total value of %i and a total weight of %i" % (val, -wt))
```

Recursive dynamic programming algorithm

Code:

```
def total_value(items, max_weight):
    return  sum([x[2] for x in items]) if sum([x[1] for x in items]) < max_weight else 0

cache = {}
def solve(items, max_weight):
    if not items:
        return ()
    if (items,max_weight) not in cache:
        head = items[0]
        tail = items[1:]
        include = (head,) + solve(tail, max_weight - head[1])
        dont_include = solve(tail, max_weight)
        if total_value(include, max_weight) > total_value(dont_include, max_weight):
            answer = include
        else:
            answer = dont_include
        cache[(items,max_weight)] = answer
    return cache[(items,max_weight)]

items = (
    ("map", 9, 150), ("compass", 13, 35), ("water", 153, 200), ("sandwich", 50, 160),
    ("glucose", 15, 60), ("tin", 68, 45), ("banana", 27, 60), ("apple", 39, 40),
    ("cheese", 23, 30), ("beer", 52, 10), ("suntan cream", 11, 70), ("camera", 32, 30),
    ("t-shirt", 24, 15), ("trousers", 48, 10), ("umbrella", 73, 40),
    ("waterproof trousers", 42, 70), ("waterproof overclothes", 43, 75),
    ("note-case", 22, 80), ("sunglasses", 7, 20), ("towel", 18, 12),
    ("socks", 4, 50), ("book", 30, 10),
    )
max_weight = 400

solution = solve(items, max_weight)
print "items:"
for x in solution:
    print x[0]
print "value:", total_value(solution, max_weight)
print "weight:", sum([x[1] for x in solution])
```

Output:

```
Bagged the following items
  banana
  compass
  glucose
  map
  note-case
  sandwich
  socks
  sunglasses
  suntan cream
  water
  waterproof overclothes
  waterproof trousers
for a total value of 1030 and a total weight of 396
```