CS 6363.004 Algorithms: Programming Project

Siddhant Sahu

April 1, 2018

1 Finding Maximum Profit

1.1 Recurrence

We define $m(i, w)$ to be the maximum profit that can be generated using the first i items and w units of gold, where i and w are both non-negative integers. Clearly, the jeweler can't make any profit with 0 items. So, $m(0, w) = 0$. When the jeweler is unable to meet the minimum quantity ($k < n_i$) according to the contract, a fine $\min(c_i, f_i \cdot (n_i - k))$ is levied. The recurrence for finding maximum profit is:

$$m(i, w) = \begin{cases} \max(m(i-1, w-k \cdot w_i) + k \cdot p_i - \min(c_i, f_i \cdot (n_i - k))) & \text{if } 0 \leq k < n_i \\ \max(m(i-1, w-k \cdot w_i) + k \cdot p_i) & \text{if } n_i \leq k \leq x_i \end{cases}$$

where k represents the quantity (i.e. number of copies) of item i.

$$0 \leq k \leq min\left(x_i, \left\lfloor \frac{w}{w_i} \right\rfloor\right)$$

1.2 Proof of correctness

Feasibility Proof by induction on i. We claim that there exists a solution for $m(i, w)$.
Basis Step: Let $i = 0$. We can not make any profit with 0 items. Thus, $m(0, w) = 0$, which is correct. This establishes the basis step.
Inductive Step: For any (i, w) and $0 \leq k \leq min\left(x_i, \left\lfloor \frac{w}{w_i} \right\rfloor\right)$, we have two cases as mentioned in the recursion. In both cases, we compute $m(i-1, w-k \cdot w_i)$. Since, $i - 1 < i$, by induction hypothesis, $m(i-1, w-k \cdot w_i)$ is feasible. This completes the inductive step and therefore, the feasibility proof.

Optimality Let $Opt(i, w)$ be an optimal solution for the problem by using the first i items and w units of gold. We claim that $m(i, w) \geq Opt(i, w)$. Proof by induction on i.
Basis Step: Let $i = 0$. We can not make any profit with 0 items. Thus, $m(0, w) = 0 = Opt(0, w)$, which is correct.
Inductive Step: Consider $i > 0$ and $0 \leq k \leq min\left(x_i, \left\lfloor \frac{w}{w_i} \right\rfloor\right)$. We have two cases here.

- Case 1: $0 \leq k < n_i$, i.e. the jeweler has to pay a fine. Suppose $Opt(i, w)$ selects k' quantities of item i. Thus, $Opt(i, w) = Opt(i-1, w-k' \cdot w_i) + k' \cdot p_i - \min(c_i, f_i \cdot (n_i - k'))$

$$\begin{aligned} m(i, w) &= \max_{0 \leq k \leq min\left(x_i, \left\lfloor \frac{w}{w_i} \right\rfloor\right)} (m(i-1, w-k \cdot w_i) + k \cdot p_i - \min(c_i, f_i \cdot (n_i - k))) \\ &\geq m(i-1, w-k' \cdot w_i) + k' \cdot p_i - \min(c_i, f_i \cdot (n_i - k')) \\ &\geq Opt(i-1, w-k' \cdot w_i) + k' \cdot p_i - \min(c_i, f_i \cdot (n_i - k')) \\ &\geq Opt(i, w) \end{aligned}$$

  Since $i - 1 < i$, $m(i-1, w-k' \cdot w_i) \geq Opt(i-1, w-k' \cdot w_i)$ by induction hypothesis.

- Case 2: $n_i \leq k \leq x_i$. Proof is almost the same as case 1, without the fines.

This completes the proof for optimality.

1.3 Pseudocode

The input to MAX-PROFIT-CALCULATOR is the total units of gold available W and a list of items *items* with attributes w, p, n, x, f and c. Attribute naming convention is consistent with the problem description.

```
MAX-PROFIT-CALCULATOR(W, items)
N = items.length
let m[0..N, 0..W] be a new table
for w = 0 to W
    m[0, w] = 0
for i = 1 to N
    for w = 1 to W
        m[i, w] = −∞
        quantity = min(items[i].x, ⌊w/items[i].w⌋)
        for k = 0 to quantity
            q = m[i − 1, w − k · items[i].w] + k · items[i].p
            if k < items[i].n
                q = q − min(items[i].c, items[i].f · (items[i].n − k))
            if q > m[i, w]
                m[i, w] = q
return m
```

Maximum profit is $m[N, W]$. A quick inspection tells us the running time is $O(kNW)$, where k is the maximum value in $x[1..n]$. If $k = N$, running time is $O(N^2W)$.

2 Counting Number of Solutions

2.1 Recurrence

We define $c(i, w)$ to be the number of solutions that generate the maximum profit $m(i, w)$. With 0 items, the solution set is empty. With one item, there is exactly one solution. Thus, the recurrence is:

$$c(i,w) = \begin{cases} \sum_{k \in \operatorname{argmax} m(i,w)} c(i-1, w-k \cdot w_i) & \text{if } i \geq 2 \\ 1 & \text{if } i = 1 \\ 0 & \text{if } i = 0 \end{cases}$$

2.2 Pseudocode

The input to COUNT-SOLUTIONS is m, the table returned from MAX-PROFIT-CALCULATOR, W and a list of *items*.

```
COUNT-SOLUTIONS(m, W, items)
N = items.length
let c[0..N, 0..W] be a new table
for w = 0 to W
    c[0, w] = 0
for w = 0 to W
    c[1, w] = 1
for i = 2 to N
    for w = 1 to W
        quantity = min(items[i].x, ⌊w/items[i].w⌋)
        for k = 0 to quantity
            q = m[i − 1, w − k · items[i].w] + k · items[i].p
            if k < items[i].n
                q = q − min(items[i].c, items[i].f · (items[i].n − k))
            if q == m[i, w]
                c[i, w] = c[i, w] + c[i − 1, w − k · items[i].w]
return c
```

Total number of solutions to the problem is $c[N, W]$. The running time is same as MAX-PROFIT-CALCULATOR, i.e. $O(kNW)$ or $O(N^2W)$ if $k = N$.