Problem Set 4

Noella James

03/04/2017

collaborators: none

Problem 4-2: OCD-2

A

The container sizes are 1, 3, and 4. Let L = 6. With the greedy algorithm, the containers chosen will be $4, 1, and\ 1$.
With the optimal solution, the containers chosen will be 3 and 3.
Therefore, the greedy algorithm uses 3 containers while the optimal solution uses only 2. Therefore, the greedy algorithm does not provide the optimal number of containers.

B

Let n_1 be the number of containers of capacity 1 gallon, n_3 be the number of containers of capacity 3 gallons, and n_4 be the number of containers of capacity 4 gallons. The solution is the minimize the value of $n_1 + n_3 + n_4$ such that $n_1 \times 1 + n_3 \times 3 + n_4 \times 4 = L$.
Assume that the capacities are $\{c_1, c_2, \ldots, c_n\}$ such that $c_1 < c_2 < \ldots < c_n$ for a given container i with capacity c_i and a capacity L' to be filled, the possibilities are as follows:
Let $\mathcal{O}$ be the optimal solution.
1. $c_i > L'$ thus $OPT(i, L') = OPT(i-1, L')$
2. $c_i \leq L'$ and $\notin \mathcal{O}$, thus $OPT(i, L') = OPT(i-1, L')$
3. $c_i \leq L'$ and $\in \mathcal{O}$, thus $OPT(i, L') = 1 + OPT(i, L' - c_i)$

For case 3, we use i instead of $i-1$ because the same size container may be used more than once.
The recurrence relationship can be stated as

if $c_i > L'$

$OPT(i, L') = OPT(i-1, L')$

otherwise

$OPT(i, L') = min(OPT(i-1, L'), 1 + OPT(i, L' - c_i))$

B-Algorithm

Algorithm 1 OCD-2

```
procedure OCD(L)                                        ▷ L is the capacity
    Array M[0...3, 0...L]
    Initialize M[0, l] = 0 for each l = 0, 1, ..., L
    Array C[0..3]
    C[0] ← 0
    C[1] ← 1
    C[2] ← 3
    C[3] ← 4
    Array Counter[1..3]
    Initialize Counter[i] ← 0 for each i = 1, 2, 3
    for i = 1...3 do
        for l = 0...L do
            if c_i > l then
                M[i, l] ← M[i − 1, l]
            else                          ▷ Find min(M[i − 1, l], 1 + M[i, l − C[i]]
                if M[i − 1, l] < 1 + M[i, l − C[i]] then
                    M[i, l] ← M[i − 1, l]
                else
                    M[i, l] ← 1 + M[i, l − C[i]]
                    Counter[i] ← Counter[i] + 1
                end if
            end if
        end for
    end for
    return M[3][L]
end procedure
```

B-Complexity

The complexity of this algorithm is the same as the Knapsack problem. Therefore, the complexity is $O(nL)$ where n is the number of distinct container types and L is the total capacity to be stored. However, in this specific problem, if n is very small and equal to 3, thus the complexity will be $O(3L)$ or $O(L)$.

B-Correctness

Lemma 0.1. *The algorithm* $OCD(L, nc[n])$ *correctly computes* $OPT(i, l)$ *for each* $i = 1, 2, \ldots, n$ *and* $l = 0, 1, \ldots, L$.

Proof. By definition, $OPT(0, l) = 0$ for all $l = 0, 1, \ldots, L$. Now, take some $j > 0$, and suppose by way of induction that $OCD(j, l)$ correctly computes $OPT(i, l)$ for all $i < j$ and for all $l = 0, 1, \ldots, L$. . By the induction hypothesis, we know that $OCD(j - 1, l) = OPT(j - 1, l)$ and $OCD(j, l - c_j) = 1 + OPT(j, l - c_j))$. Note that $OCD(j, l - c_j)$ gets computed before $ODC(j, l)$ and thus the value has been computed. Hence, it follows that $OPT(j, l) = min(OPT(j - 1, l), 1 + OPT(j, l - c_j)) = OCD(j, l)$ $\square$