Algorithms

Homework #3

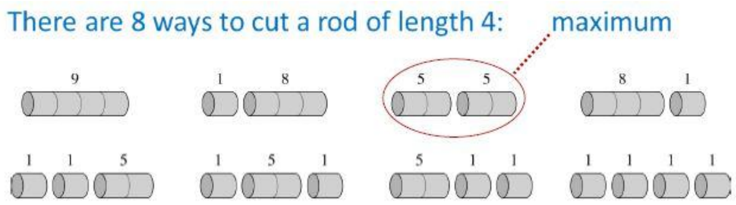
0016046 蔡佩珊

(1)

(counter-example)

n=4

length i	1	2	3	4
price p_i	1	5	8	9
density d_i	1	2.5	2.67	2.25



cut rod by the given strategy

length 4 => length 3 + length 1

but $p_3 + p_1$ = 8 + 1 = 9 is not maximum

so the given strategy does not always determine an optimal way to cut rod

(2)

suppose optimal solution x is obtained by cutting the rod of length n at length i

x = y + z

where y and z are optimal solutions to the subproblems of cutting the rods of length i and length n-i

if we now had limit l_i on the number of pieces of length i that we allowed to produce, for i=1,2,…,n

if both y and z has length i pieces and $x_i = y_i + z_i > l_i$ the optimal solution must be changed to satisfied

and the modified y' or z' are not optimal solutions of cutting the rods of length i and length n-i

anymore if $y_i < l_i$ or $z_i < l_i$

x' = y' + z'

(3)

(4)

(optimal-substructure)

x_i : the highest-numbered item in an optimal solution S for weight W and n items

S' = S - { x_i } must be optimal solution for weight W-w_i and n-1 items

$V_s = V_{s'} + v_i$

define $c[i,w]$ to be the value of solution for iterms 1,…,i and maximum weight w

$$c[i,w] = \begin{cases} 0 & , \text{if } i = 0 \text{ or } w = 0 \\ c[i-1,w] & , \text{if } w_i > \mathrm{w} \\ \max(v_i + c[i-1,w-w_i]\,, c[i-1,w])\,, & \text{if } i > 0 \text{ and } w \geq w_i \end{cases}$$

$\mathrm{DYNAMIC-01KANPSACK}(v,w,n,W)$

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<=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 $c[i,w] = c[i-1,w]$

// runs in O(*nW*) time

(5)

n: number of items

v_i: value of item *i*

w_i: weight of item *i*

W: weight capacity

$\mathrm{R} = \{\frac{v_1}{w_1}, \dots, \frac{v_n}{w_n}\}$

(1) choose an element *r* at random from R

(2) determine

$$\begin{cases} R_1 = \left\{\frac{v_i}{w_i} \middle| \frac{v_i}{w_i} > r, \text{for } 1 \leq i \leq n\right\}, W_1 = \sum_{i \in R_1} w_i \\ R_2 = \left\{\frac{v_i}{w_i} \middle| \frac{v_i}{w_i} = r, \text{for } 1 \leq i \leq n\right\}, W_2 = \sum_{i \in R_2} w_i \\ R_3 = \left\{\frac{v_i}{w_i} \middle| \frac{v_i}{w_i} < r, \text{for } 1 \leq i \leq n\right\}, W_3 = \sum_{i \in R_3} w_i \end{cases}$$

(3)

if W_1>W

    reurse on R_1 and return computed solution

else

    while (there is space in knapsack and R_2 not empty)

        add items from R_2

    if (knapsack full)

        retrun the items in R_1 and the items added from R_2

    else

W = W-(W_1+W_2)

recurse on R_3 and return the items in $R_1 \cup R_2$

and the items returned from the recursive call

// runs in O(n) time

(6)

n: cents to be changed

c: the largest coin value such that c<=n

while (the amount to be changed is not 0)

find c

n=n-c

(optimal substructure)

c_1 : the first coin with value of n_1 chosen by the given algorithm in an optimal solution S for n cents

C' = C - { c_1 } is an optimal solution for n-n_1 cents

otherwise, let B be an optimal solution

B∪{ c_1 } contains fewer coins than C and yet both makes up n cents, so C cannot be the optimal solution making change for n cents, a contradiction

(greedy choise property)

If c_1=a penny, then n<=4 cents

no other choice cannot to make the change, since any other coin is > 4 cents

If c_1=a nickel, then 5<=n<=9 cents

If the optimal choice uses at least 5 pennies => turn it to a nickel and use fewer coins

Otherwise, the optimal choice can only change 4 cents, not enough to make the change

If c_1=a dime, then 10<=n<=24 cents

If the optimal choice uses at least 5 pennies => turn it to a nickel and use fewer coins

Otherwise, if it uses 2 nickels => turn it to a dime

Otherwise, it can only make change for at most 9 cents, not enough to make the change

If c_1=a quarter, then 25<=n cents

If the optimal choice uses at least 5 pennies => turn it to a nickel and use fewer coins

Otherwise, if the optimal solution uses at most 2 dimes, it must use enough pennies to make up at least 25 cents, and we can turn it to a quarter and use fewer coins

Otherwise, the optimal solution uses at least 3 dimes => turn it to a quarter and a nickel and use fewer coins

Therefore the greedy choice property is satisfied