ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Section d'Informatique et de Systèmes de Communication

Corrigé de la série 7 7 November 2011

1.Sac à dos rationnel

a) Intuitivement, on sent que l'algorithme va nous donner une solution optimale, il est plus difficile de le prouver formellement. Nous allons montrer que toute solution autre que celle obtenue par l'algorithme sera moins bonne (c'est-à-dire que la valeur totale sera inférieure).

Supposons d'abord (sans perdre de généralité) que nos objets sont ordonnés selon leur valeur relative (i.e. prix au kilo) $\frac{v_i}{w_i}$ de façon décroissante. On a donc:

$$\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n} \tag{1}$$

Soit $S_1 = (x_1, \ldots, x_n)$ la solution obtenue avec l'algorithme glouton, c'est-à-dire qu'on prend une fraction x_i du produit i.

L'algorithme glouton va commencer par prendre autant de que possible du produit 1 (puisque c'est celui qui a le plus de valeur par kilo), lorsqu'il ne reste plus de produit 1 il prendra autant que possible du produit 2 etc.. Il s'arrête lorsque le poids total aura atteint W. Si $x_i = 1$ pour tout i, clairement cette solution est optimale. Sinon, il y a donc un produit m pour lequel il va prendre une quantité $x_m < 1$ et s'arrêter (puisqu'il aura atteint un poids total de W). L'algorithme ne prendra donc rien des produits $x_{m+1}, \ldots, x_n$. On a donc:

- $\forall i < m, x_i = 1$
- $\forall i > m, x_i = 0$
- $\sum_{i=1}^{n} x_i w_i = W$

Soit $S_2 = (y_1, \ldots, y_n)$ une solution quelconque, on a donc:

$$\sum_{i=1}^{n} y_i w_i \leq W = \sum_{i=1}^{n} x_i w_i \Longrightarrow \sum_{i=1}^{n} (x_i - y_i) w_i \geq 0 \tag{2}$$

On remarque aussi que

- Si $i < m$ donc $x_i = 1$ et on a $x_i - y_i \geq 0$ et $\frac{v_i}{w_i} \geq \frac{v_m}{w_m}$
- Si $i > m$ donc $x_i = 0$ et on a $x_i - y_i \leq 0$ et $\frac{v_i}{w_i} \leq \frac{v_m}{w_m}$

Et on remarque que dans les deux cas on aura

$$(x_i - y_i)\frac{v_i}{w_i} \geq (x_i - y_i)\frac{v_m}{w_m} \tag{3}$$

Calculons la différence entre la valeur totale de S_1 et de S_2:

$$\begin{aligned} \text{val}(S_1) - \text{val}(S_2) &= \sum_{i=1}^{n} x_i v_i - \sum_{i=1}^{n} y_i v_i \\ &= \sum_{i=1}^{n} (x_i - y_i) w_i \frac{v_i}{w_i} \\ &\geq \sum_{i=1}^{n} (x_i - y_i) w_i \frac{v_m}{w_m} \\ &= \frac{v_m}{w_m} \sum_{i=1}^{n} (x_i - y_i) w_i \\ &\geq 0, \end{aligned}$$

On passe de la $2^{\text{ème}}$ à la $3^{\text{ème}}$ ligne en utilisant (3) et de la $4^{\text{ème}}$ à la $5^{\text{ème}}$ ligne en utilisant (2).

Ainsi la solution S_1 obtenue par l'algorithme glouton est au moins aussi bonne que toute autre solution $S_2 = (y_1, \ldots, y_n)$, elle est donc optimale.

b) On commence par calculer et classer les valeurs relatives des produits:

produit	w	v	v/w	classement
A	200	1600	8	3
B	10	100	10	2
C	30	450	15	1
D	100	600	6	4

On prend d'abord autant que possible du produit 1 (C), et puisque $w_1 < W$ on peut prendre tout ce qui est disponible, donc 30 grammes. Ensuite on veut prendre du produit 2 (B), encore une fois, puisque $w_1 + w_2 < W$ on prend la totalité des 10 grammes disponibles. Nous voulons ajouter autant que possible de produit 3 (A) aux 40 grammes que nous avons dejà, nous en prenons donc 10 grammes et nous avons atteint la limite de 50 grammes.

La solution optimale est donc $(30, 10, 10, 0)$ (en prenant les produits dans l'ordre donné par le classement dans le tableau ci-dessus), et on obtient une valeur totale de $30 \cdot 15 + 10 \cdot 10 + 10 \cdot 8 = 630\$$.

2.Sac à dos 0/1 avec des poids rationnels

a) On fait n itérations, chacune à $\theta(W)$ opérations, donc en tout, c'est $\theta(nW)$.

b) On peut appliquer l'algorithme aussi si les valeurs sont des rationnels. Rien ne change, sauf que les entrées de la matrice C seront maintenant aussi des rationnels.

c) Si $W, (v_1, w_1), \ldots, (v_n, w_n)$ est un problème du sac à dos ayant une solution optimale, alors pour n'importe quelle constante $c > 0$, le problème du sac à dos de paramètres $c \cdot W, (v_1, c \cdot w_1), \ldots, (v_n, c \cdot w_n)$ est équivalent.

Si l'on multiplie W et les w_i par $q_1 q_2 \cdots q_n$, on obtient un problème avec des coefficients entiers (supposant que W était entier); et on peut donc appliquer l'algorithme 0/1 du cours à ce problème modifié.

Comme le temps de parcours est proportionnel à W, on a intérêt à avoir ce nombre aussi petit que possible. Le plus petit entier avec lequel on peut multiplier les poids du problème original pour avoir un problème avec des coefficients entiers est le plus petit multiple des q_i, $\text{lcm}(q_1, \ldots, q_n)$.

d) Si on prend les q_i premiers entre-eux (par exemple, des premiers distincts), on aura $\text{lcm}(q_1, \ldots, q_n) = q_1 q_2 \cdots q_n$. Donc, si la valeur de W était initialement 1, on aura

$W = q_1 q_2 \cdots q_n$ et, si les $w_i \leq 1$,

$$\begin{aligned}\text{length}(w_1, \ldots, w_n) &= \sum_{i_1}^{n} \log(q_i) + \log(p_i) \\ &\leq 2 \sum_{i_1}^{n} \log(q_i) \\ &= 2 \log(W),\end{aligned}$$

donc $W \geq \sqrt{2}^{\text{length}(w_1,\ldots,w_n)}$. L'algorithme du cours a donc un temps de parcours

$$\Omega(nW) = \Omega(n\sqrt{2}^{\text{length}(w_1,\ldots,w_n)}) = \Omega(\sqrt{2}^{\text{length}(w_1,\ldots,w_n)}).$$

e) La fonction length$(\cdot)$ représente la longueur de l'input (si on suppose qu'on prend toujours $W = 1$ et donc que W ne fait pas partie de l'input). En effet, on a besoin de $\log(p_i)$ bits pour représenter p_i, de $\log(q_i)$ bits pour représenter q_i et donc de $\log(p_i) + \log(q_i)$ bits pour w_i.

f) Supposons que la taille de l'input est $n \cdot N$ bits, ou n est le nombre d'objets du problème, et N le nombre de bits utilisé pour un couple (v_i, w_i).

Remarquons que les opérations de nombres avec N bits dont nous avons besoin (notamment, division, comparaisons et addition), sont tous polynomiaux en N. Nous parlerons de "opération arithmétique" par la suite.

Le pas du tri se fait en $O(n^2)$ opérations arithmétiques (si on choisit un mauvais algorithme de tri). Le deuxième pas est $O(n)$ opérations arithmétiques. En total, c'est

$$O(n^2) \cdot O(\text{poly}(N)) = O(n^2 \text{poly}(N)) = O(\text{poly}(n \cdot N))$$

opérations, où poly(N) désigne un polynôme en N.

g) Voir p.e. le corrigé du point suivant.

h) L'idée est de prendre un objet miniscule avec un bon rapport qui est juste assez grand pour bloquer l'autre (qui aura un rapport un peu moins bon, mais qui est beaucoup plus lourd). Par exemple, on peut choisir

$$\begin{array}{lllll} v_0 = \varepsilon & & v_1 = 1 & & \\ w_0 = \varepsilon/2 & & w_1 = 1 & & W = 1 \end{array}$$

Comme $v_0/w_0 = 2 > 1 = v_1/w_1$ l'algorithme choisit d'abord d'insérer l'objet 0. Ensuite il ne reste plus assez de place pour l'objet 1, et donc la solution aura la valeur ε.

En prenant l'objet 1 au lieu de l'objet 0, on aurait eu une solution avec valeur 1, ce qui est donc ε^{-1} fois mieux.

3. Les tours de Hanoi

a) Dans ce qui suit, ThirdOf est la fonction qui associe retourne pour deux nombres entre 1 et 3 donnés, le troisième nombre (par exemple $\textsc{ThirdOf}(3,1) = 2$).

Nous pouvons définir la fonction qui déplace une tour de taille n récursivement comme suit:

Call: HanoiTowerMove
Input: n, s, d, n: Taille de la tour, bâtons source et destination
Output: Une suite d'appels à MoveSlice.

```
if n = 1 then
    MoveSlice(s, d).
    return
t ← ThirdOf(s, d)
HanoiTowerMove(n − 1, s, t)
MoveSlice(s, d)
HanoiTowerMove(n − 1, t, d)
```

b) Notons T_n le nombre de MoveSlice nécessaire pour déplacer une tour de taille n. Nous avons clairement $T_1 = 1$. La définition récursive ci-dessus permet aussi de déduire la formule inductive

$$T_n = 2T_{n-1} + 1.$$

Nous prouvons maintenant par induction la formule fermée $T_n = 2^n - 1$. Elle est clairement vraie pour $n = 1$. Pour $n > 1$, nous concluons en appliquant l'hypothèse d'induction au cas $n - 1$ que

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2 \cdot (2^{n-1} - 1) + 1 \\ &= 2^n - 1. \end{aligned}$$

Ceci termine la preuve.

Remarquons que T_n croît exponentiellement vite, ce qui implique que même pour n de taille modérée, cet algorithme ne terminera pas en un temps raisonnable.

c) Notons $\tilde{T}_n$ le temps utilisé pour déplacer une tour de taille n en utilisant une méthode optimale quelconque. Il nous faut alors montrer que pour tout n on a $\tilde{T}_n \geq T_n$. Clairement, pour $n = 1$ on ne peut pas faire mieux qu'en une étape, et donc $\tilde{T}_1 \geq 1 = T_1$. Nous montrons maintenant qu'on a

$$\tilde{T}_n \geq 2\tilde{T}_{n-1} + 1,$$

pour tout $n > 1$, ce qui nous permettra de conclure.

Supposons que nous déplaçons une tour de taille n avec la méthode optimale. Il existe donc une étape m, $1 \leq m \leq \tilde{T}_n$ qui déplace le plus grand disque pour la dernière fois.

En l'étape m, nous avons:

(i) Le n-ème disque est placé sur le bâton but, et aucun des $n - 1$ disques plus petits peut déjà se trouver à cet endroit (parce que le n-ème disque est plus grand que les $n - 1$ autres disques de la tour).

(ii) Le bâton duquel le n-ème disque est pris contient, à cet instant, aucun des $n-1$ plus petits disques de la tour, parce que le n-ème disque est enlevé d'en haut. Il s'en suit que les $n-1$ plus petits disques forment une tour sur le troisième bâton.

Nous en déduisons déjà qu'après le m-ème mouvement nous avons encore besoin de déplacer la tour à $n-1$ disques au bon endroit, ce qui nécessite donc au moins $\tilde{T}_{n-1}$ étapes.

Avec des raisonnements analogues, si l'on considère l'étape m' du *premier* déplacement du n-ème disque, nous voyons que *avant* l'étape m' il nous a fallu déplacer une tour de taille $n-1$, ce qui ajoute encore une fois $\tilde{T}_{n-1}$ opérations.

Nous avons donc que $\tilde{T}_n \geq 2\tilde{T}_{n-1} + 1$, ou le $+1$ vient de l'étape m (nous ne comptons pas l'étape m', parce qu'il se peut que $m = m'$).

Le fait que la fonction $x \mapsto 2x + 1$ est croissante permet maintenant de conclure par induction que

$$\tilde{T}_n \geq T_n$$

pour tout n.