

Algorithmique

GALAI Nam EG

Devoir

Houda

- 121116d4 -

Partie I-8

a)

Algo-Gloubon (S: entier, L: tableau de m entiers) :

S1: entier;

i: entier

T: tableau de m entiers

S1 \leftarrow 0

i \leftarrow 0

T[i] \leftarrow 0

tant que (S1 \leq S) faire :

 si (S1 + L[i] \leq S)

 S1 \leftarrow S1 + L[i]

 T[i] \leftarrow T[i] + 1

 else

 i \leftarrow i + 1

 T[i] \leftarrow 0

 fin si

fin tant que

retourner (T)

b) au pire cas

On arrive à $S_1 = S$ lorsqu'on force un bout de tableau, autrement dit, lorsqu'on utilise toutes les pièces $\{t_1, \dots, t_m\}$

on aboutit à rendre la somme S .

On a $O(n)$: la taille des données.

D'autre part, on a pour $i \in [m]$ $S/L[i] \leq S/L[m-1]$

$S/L[i]$ pour faire le moins
nécessaire des pièces t_i

\Rightarrow Pour la boucle dans que, on aura la complexité

est de la forme $O(S/L[m-1])$.

\Rightarrow La complexité ~~est~~ de l'algorithme s'écrit alors

$$C \sim O(n) + O(S/L[m-1])$$

au meilleur cas

L'algorithme arrive à rendre S dans $L[0]$.

\Rightarrow Pour la boucle dans que, on aura la complexité
est de la forme: $O(S/L[0])$

\Rightarrow La complexité de l'algorithme s'écrit alors

$$C \sim O(1) + O(S/L[0])$$

$$\Rightarrow \boxed{C \sim O(S/L[0])}$$

9) On a $L = \{t_m, \dots, t_2\}$ l'ensemble qui désigne les types de pièces données.

On définit L' tel que $L' \subseteq L$ avec L' l'ensemble des pièces (ordonnancement des pièces de façon optimale)

↳ L' est un indépendant

Autrement dit, dans cet algorithme glouton L' représente la version de grebeau de l'algorithme.

⇒ Les indépendants associés à cet algorithme glouton sont les ensembles de grebeau qui désignent une partition optimale des pièces.

d) Ce système d'indépendance n'est pas un matroïde car

on peut trouver des ensembles grebeau pas optimal.

exemples Si $S=14$ et $L=\{6, 4, 3, 1\}$

selon glouton ma partition sera $14=6+6+1+1$

⇒ 4 pièces,

alors que l'optimal sera $14=6+4+4$

⇒ 3 pièces.

Partie II-6

a/ On a $V(i, j)$ le nombre minimal de pièces de types choisis dans (t_1, \dots, t_i) de somme j .

Notre problème consiste à trouver une somme S .

Alors $j=S$.

Optimalement, on cherche à la trouver avec le moins de nombre de pièces, alors la restriction pour un nombre i de pièces (avec $i \leq m$), ne va pas donner une solution optimale,

Car le plus qu'on a de nombre de types de pièces, le mieux notre solution sera en terme d'optimale.

$$\text{Alors } i = \max(t_1, \dots, t_m) = m$$

\Rightarrow Conclusion : Pour qu'on aboutit à une solution optimale,

$$\text{il faut que } \boxed{i=m \text{ et } j=S} \Rightarrow \boxed{V(m, S)}$$

b/ Soit $V(i, j)$ le nombre minimal de pièces de somme j .

Pour ~~$V(i, j)$~~ $V(i, 0)$, on la somme est égale à 0.

Alors quelque soit le nombre des pièces i , $\boxed{V(i, 0) = 0; \forall i}$.

* Pour $i \geq 1$; on a $V(i, y) \leq V(i-1, y)$.

Justifications

Paradoxalement, pour $i \geq 1$; on a plus de types de pièces que pour $(i+1)$, donc intuitivement, si je me ~~peux~~ prends y à partir de i pièces, $V(i, y) = 0$, et après un certain i , dès que je prends la ~~jeux~~, $V(i, y)$ va prendre la même valeur pour tous $i \geq i$; alors que, si je me des 0 pour ces $V(i, y)$, puisque mon programme va prendre le minimum des possibilités, il va prendre les 0.

Donc, ~~je~~ je dois associer aux cas où je ~~peux~~ pas prendre avec la somme avec i , l'infini ∞ , pour que quelque soit la valeur d'après, le minimum me va pas prendre ces cas d'incapacités.

9) On note $K_{i,j} = \{(k, m) \in \mathbb{N}^2 / k+m+t_i = j\}$

↳ k représente le reste de la somme j avec les pièces qui représentent les précédents t_i .

Autrement dit $k = \sum_{p=1}^{i-1} m_p t_p$.

↳ k peut s'écrire donc sous la forme $k' + m t_i$; $k' \leq i$

Alors $V(i, j)$ peut s'écrire de la forme de $V(i-1, k') + m$

d'autre part, pour un i quelconque $V(i, j)$ peut aussi avoir la valeur de $V(i-1, j)$ (comme expliqué dans la question précédente).

Alors $V(i, j)$ prend la valeur minimale entre ces deux.

$$\Rightarrow V(i, j) = \min (V(i-1, j); V(i-1, k') + m)$$

d/

algo-dynamique (S₀ entier, L₀ tableau de montiers)

, m^o entier)

i, j, k, l, min S entiers
k = 0

Si S = 0 faire S

Pour i de 1 à m faire S

V[i][0] = 0

fin Pour

Si min S

Pour i de 1 à m faire S

Si 0 < V[i][S] < 9999 faire

Grebowicz (V[i][S])

fin S

fin Pour

Si min S

min = 9999

Pour i de 1 à m faire S

Si L[i] < S faire S

k = k + 1

$S \leftarrow S - L[i]$

Pour i de 1 à n faire

$S \leftarrow (S - L[i])$ faire

Recursivité avec le
nouveau S'

On a fait $S \leftarrow S - L[i]$

$f_a = f_a + \text{algo-dynamique}(S, L, i, m)$

$f_a = f_a + \text{algo-dynamique}(S, L, i, m)$

fin si Si $f_a < \text{min}$

$\text{min} = f_a$

$f_a = f_a + \text{algo-dynamique}(S, L, i, m)$

fin Pcm

fin Ssi

fin Ssi

fin Pcm

$\leftarrow n$

fin si

retourner (min)

2) Remplissage du tableau dans le main

Pcm un tableau 2 dimensions initialisé à $V[i][j] = 9999$

Pour j de 0 à 8 faire

 + 1 if j est 0 ou 5 ou 8

Pcm i de 1 à n faire

$V[i][j] = \text{algo-dynamique}(j, L, m)$

fin Pcm

fin Pcm.