Solutions TP5

IFT2125 - Hiver 2016

23 février

Numéro 1

Problème 8.1, p.278, livre Brassard-Bratley : Démontrez que le nombre total d'appels récursifs fait par l'algorithme suivant de calcul directe de $\binom{n}{k} = C(n,k)$ est $2\binom{n}{k} - 2$:

Algorithme $C(n,k)$ (pour $0 \leq k \leq n$)
    SI $k = 0$ ou $k = n$ ALORS
        RETOURNER 1
    SINON
        RETOURNER $C(n-1,k-1) + C(n-1,k)$

Solution : Soit NbAppels(n,k) le nombre d'appels récursifs faits par $C(n,k)$. Montrons que

$$\text{NbAppels}(n,k) = 2\binom{n}{k} - 2$$

pour tout $0 \leq k \leq n$.

Nous remarquons d'abord que seule la ligne 2 est exécutée lorsque $k = 0$ ou $k = n$. Dans ce cas, il n'y a donc aucun appel récursif et nous avons bien

$$\text{NbAppels}(n,k) = 2\binom{n}{0} - 2 = 2\binom{n}{n} - 2 = 0.$$

Procédons maintenant par induction sur n pour montrer la proposition initiale. Si $n = 0$, alors $k = 0$. Ce cas vient d'être traité. Supposons que l'hypothèse d'induction est vraie pour $n-1$. Si $k = 0$ ou $k = n$, alors nous avons terminé. Autrement, un appel à $C(n,k)$ exécute la ligne 4 de

l'algorithme. Dans ce cas, nous avons

$$\begin{aligned}
\text{NbAppels}(n,k) &= \text{NbAppels}(n-1,k-1) + \text{NbAppels}(n-1,k) + 2 \\
&= \left(2\binom{n-1}{k-1} - 2\right) + \left(2\binom{n-1}{k} - 2\right) + 2. \\
&= 2\left(\binom{n-1}{k-1} + \binom{n-1}{k}\right) - 2 \\
&= 2\binom{n}{k} - 2
\end{aligned}$$

où la seconde ligne découle de l'hypothèse d'induction et la quatrième de la définition récursive du coefficient binomial. □

Numéro 2

Problème 8.9, p.278, livre Brassard-Bratley : Adaptez l'algorithme de programmation dynamique pour le retour de la monnaie pour qu'il fonctionne correctement même si on a un nombre limité de pièces d'une certaine valeur.

Voici l'algorithme de retour de monnaie donné dans B& B :

```
function retour-monnaie(d, N)
    Définir un tableau C de taille (n + 1) × (N + 1)
    for i ← 1 to n do C[i, 0] ← 0;
    for j ← 0 to n do C[0, j] ← +∞;
    for i ← 1 to n do
        for j ← 1 to N do
            if (j ≥ d_i) then
                C[i, j] ← min(C[i − 1, j], 1 + C[i, j − d_i])
            else
                C[i, j] ← C[i − 1, j]
    return C[n, N]
```

Solution : Il est possible de faire une variante de cette algorithme qui considère un nombre fini de pièce. Nous ajoutons tout simplement un compteur b_i, pour les pièces de valeur i qui indique combien de pièces peuvent encore être utilisése. Lorsque b_i atteint 0, on utilise la solution optimale qui n'utilise pas les pièces de valeur i.

```
function retour-monnaie2(d, b, N)
    Définir un tableau C de taille (n+1) × (N+1)
    for i ← 1 to n do C[i, 0] ← 0;
    for j ← 0 to n do C[0, j] ← +∞;
    for i ← 1 to n do
        for j ← 1 to N do
            if (j ≥ d_i) ∧ (b_i > 0) then
                t ← 1 + C[i, j − d_i]
                C[i, j] ← min(C[i − 1, j], t)
                if (C[i, j] = t) then b_i ← b_i − 1;
            else
                C[i, j] ← C[i − 1, j]
    return C[n, N]
```

Il est aussi possible de résoudre les problèmes de programmation dynamique de façon "top-down", c'est-à-dire en commençant par la case qui nous intéresse et en mémoisant les résultats des sous-problèmes. Cette approche peut nous éviter dans certains cas de calculer toute la table.

Numéro 3

Problème 8.10, p.278, livre Brassard-Bratley : Dans le cours, pour le problème du sac à dos, nous avons fait l'exemple où la capacité du sac était $W = 11$ avec 5 objets de poids respectifs $w_1 = 1, w_2 = 2, w_3 = 5, w_4 = 6, w_5 = 7$ et de valeurs respectives $v_1 = 1, v_2 = 6, v_3 = 18, v_4 = 22, v_5 = 28$. Retravaillez cet exemple mais en numérotant les objets dans l'ordre inverse (i.e $w_1 = 7, w_2 = 5, \ldots w_5 = 1$ et $v_1 = 28, v_2 = 22, \ldots, v_5 = 1$). Quels éléments de la table devraient demeurés inchangés ?

En appliquant avec l'ordre initial, voici la table résultante :

Poids limite	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	0	0	0	0	0	0	0	0	0	0	0	0
$w_1 = 1, v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2, v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5, v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6, v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7, v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

En appliquant l'algorithme, nous obtenons le tableau suivant :

Poids limite	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	0	0	0	0	0	0	0	0	0	0	0	0
$w_1 = 7, v_1 = 28$	0	0	0	0	0	0	0	28	28	28	28	28
$w_2 = 6, v_2 = 22$	0	0	0	0	0	0	22	28	28	28	28	28
$w_3 = 5, v_3 = 18$	0	0	0	0	0	18	22	28	28	28	28	40
$w_4 = 2, v_4 = 6$	0	0	6	6	6	18	22	28	28	34	34	40
$w_5 = 1, v_5 = 1$	0	1	6	7	7	18	22	28	29	34	35	40

Le première ligne reste inchangée car l'initialisation ne change pas. La première colonne reste inchangée puisqu'il n'est jamais possible de prendre des objets lorsque le poids limite est 0 car $w_i > 0$. De plus, la dernière ligne reste inchangée car, à cette ligne, tous les objets peuvent être utilisés. La solution optimale est donc encore la même.

Numéro 4

Problème 8.11, p.278, livre Brassard-Bratley : Écrivez le pseudocode de l'algorihtme de programmation dynamique pour le problème du sac à dos.

```
function sac-a-dos(v, w, W)
    Définir un tableau V de taille (n + 1) × (W + 1)
    for j ← 0 to W do V[0, j] ← 0;
    for i ← 1 to n do
        for j ← 0 to W do
            if (j ≥ w_i) then
                V[i, j] ← max(V[i − 1, j], V[i − 1, j − w_i] + v_i)
            else
                V[i, j] ← V[i − 1, j]
    return V[n,W]
```

Numéro 5

Problème 8.12, p.278, livre Brassard-Bratley : Pour le problème du sac à dos, quand $j < w_i$, on pose $V[i-1, j-w_i] = -\infty$ Est-ce que la table de programmation dynamique peut contenir des entrées $-\infty$? Si oui, qu'est-ce que ces entrées indiquent ? Sinon, pourquoi ?

Non cela est impossible. Montrons que $V[i, j] \geq 0$ pour tout $0 \leq j \leq W$ par induction sur i. Si $i = 0$ alors $V[0, j] = 0$ par définition. Supposons que $V[i-1, j] \geq 0$ pour tout $0 \leq j \leq W$. Par définition, nous avons

$$V[i, j] = \max(V[i-1, j], V[i-1, j-w_i] + v_i).$$

Ainsi, par définition de max et par hypothèse d'induction,

$$V[i, j] \geq V[i-1, j] \geq 0. \quad \square$$

Notez que cela signifie tout simplement que la solution triviale (aucun objet sélectionné) est toujours une solution au problème.

Numéro 6

Problème 8.13, p.278, livre Brassard-Bratley : Il peut y avoir plus d'une solution optimale pour le problème du sac à dos. En utilisant la table de programmation dynamique construite pour résoudre le problème, peut-on trouver toutes les solutions optimales ou seulement une seule ? Si oui, comment ? Sinon, pourquoi ?

Oui. Lorsque $V[i-1,j] = V[i-1,j-w_i] + v_i$, il est aussi avantageux de choisir l'objet i que de ne pas le choisir.

Voici un exemple. Soit un sac à dos de capacité de 3 unités et les objets suivants : $[(v_1 = 3, w_1 = 3), (v_2 = 2, w_2 = 2), (v_3 = 1, w_3 = 1)]$. Deux solutions optimales existent soit $\{o_1\}$ ou $\{o_2, o_3\}$

Il est possible d'adapter les algorithmes de programmation dynamique pour pouvoir calculer toutes les solutions optimales. Il suffit de garder en mémoire pour chaque état le(s) état(s) précédent(s) qui lui donne sa valeur optimale. Il est recommandé d'utiliser la version récursive pour faire ce calcul de façon plus élégante.

Si on ne souhaite pas utiliser plus de mémoire, il est aussi possible de retrouver la solution en fonction de la table de programmation dynamique. Par contre, cette option est coûteuse en temps.

Numéro 7

Problème 8.15, p.279, livre Brassard-Bratley : Dans le problème du sac à dos considéré en classe, nous avons assumé qu'on avait n objets numérotés de 1 à n. Supposons que l'on a plutôt n types d'objets, avec une quantité très grande d'objets de chaque type. Formellement, cela revient à relaxer la contrainte que chaque x_i soit 0 ou 1 par la contrainte où x_i doit être un entier non négatif. Adaptez l'algorithme de programmation dynamique décrit au numéro 4 pour qu'il fonctionne avec ce nouveau problème.

Oui. Il suffit de rajouter $V[i, j - w_i] + v_i$ comme choix dans la récurrence.