Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université de Carthage

Ecole Nationale d'Ingénieurs de
Carthage

ENICARTHAGE
المدرسة الوطنية للمهندسين بقرطاج
Ecole Nationale d'Ingénieurs de Carthage

وزارة التعليم العالي و البحث
العلمي
جامعة قرطاج
المدرسة الوطنية للمهندسين
بقرطاج

Année universitaire 2017-2018

Test en
Algorithmique Avancée et Complexité

Le problème du sac à dos

Madame Dupont doit partir en voyage ; elle emporte un sac à dos qui peut supporter au maximum une masse M, M ≥ 0. Par ailleurs, elle dispose de N objets, N ≥ 0, numérotés de 1 à N. Pour tout i appartenant à 1, . . . , N, l'objet de numéro i possède deux caractéristiques : sa valeur notée v_i et sa masse noté m_i qui sont des entiers strictement positifs. Madame Dupont désire emporter certains de ces objets dans son sac à dos ; la masse totale des objets emportés ne doit pas dépasser M ; elle cherche à maximiser la somme des valeurs des objets qu'elle emporte parmi les contenus possibles ne dépassant pas au total la masse M.

On note V [N, M] la valeur maximale réalisable avec des objets sélectionnés parmi les N objets présents et la masse maximale M, c'est-à-dire :

$$V[N,M] = \max_{\substack{x \in \{0,1\}^N \\ \sum_{k=1}^{N} x_k m_k \leq M}} \sum_{k=1}^{N} x_k v_k.$$

On peut accessoirement vouloir connaître la composition du sac. Dans ce cas il faudra retourner la valeur de x ∈ {0, 1} qui conduit à V [N,M].

1. Justifier la formule de récurrence pour V [N, M] suivante :

$$V[i,j] = \begin{cases} \max\left(V[i-1,j], v_i + V[i-1,j-m_i]\right) & \text{si } m_i \leq j \\ V[i-1,j] & \text{sinon} \end{cases}$$

   Avec les conditions initiales suivantes :

$$\begin{cases} V[0,j] = 0 & \text{si } j \geq 0 \\ V[i,0] = 0 & \text{si } i \geq 0. \end{cases}$$

2. Écrire une fonction récursive qui calcule la valeur maximale V [i, j]

3. Énoncer la récurrence qui régie la complexité de cette fonction.

4. Calculer la complexité de cette fonction en développant cette récurrence.

5. Etant donné M = 5 et les objets suivants:

N° objet	1	2	3	4
Masse	2	1	3	2
Valeur	12	10	20	15

Remplir le tableau V [0….4, 0…..5]

$V[i, j]$						
i \ j	0	1	2	3	4	5
0						
1						
2						
3						
4						

6. Ecrire l'algorithme correspondant

7. Calculer sa complexité

"Le Problème du "Sac à Dos"

Résolution Par Programmation Dynamique

La 1ère étape de Conception d'un algo de Prog_Dym consiste à exprimer la solᵒ d'une instance en fct d'une instance plus petite.

I/ On considère une instance constituée des i premiers objets $1 \leq i \leq N$ avec les masses $m_1, m_2, \dots, m_i$ et les valeurs $v_1, v_2, \dots, v_i$ et d'un sac à dos de capacité j / $1 \leq j \leq M$, soit $V[i, j]$ la valeur optimale de la solᵒ optimale de cette instance:

$$V[i,j] = \max_{\substack{x \in \{0,1\} \\ \sum_{k=1}^{i} x_k m_k \leq j}} \sum_{k=1}^{i} x_k v_k$$

On a plusieurs cas à considérer.

1_ le $i^{ème}$ objet est plus lourd que la capacité du sac

$\Rightarrow V[i, j] = V[i-1, j]$ / $\underline{m_i > j}$

2_ il est possible de prendre le $i^{ème}$ objet :

a_ On le prend et la valeur du butin est celle du $i^{ème}$ objet plus celle du butin qu'on peut constituer à partir des $i-1$ 1ers objets avec un sac de masse $(j - m_i)$

$\Rightarrow V[i, j] = v_i + V[i-1, j-m_i]$ / $\underline{m_i \leq j}$

b_ On ne le prend pas et le prob se ramène à la recherche du meilleur butin parmi les $i-1$ 1ers objets : $V[i, j] = [i-1, j]$ / $\underline{m_i \leq j}$

$\Rightarrow$ la valeur maximale du sous-ensemble optimal des i 1er objets est donc le max de ces 2 dernières valeurs.

Le Problème du "Sac à Dos"

Résolution Par Programmation Dynamique

la $1^{ère}$ étape de Conception d'un algo de Prog-Dyn consiste à exprimer la sol° d'une instance en fct d'u. instance plus petite :

I/ On considère une instance constituée des i premiers objets $1 \leq i \leq N$ avec les masses $m_1, m_2, \ldots, m_i$ et les valeurs $v_1, v_2, \ldots, v_i$ et d'un sac à dos de capacité j / $1 \leq j \leq M$, soit $V[i, j]$ la valeur optimale de la sol° optimale de cette instance :

$$V[i,j] = \max_{\substack{x \in \{0,1\} \\ \sum_{k=1}^{i} x_k m_k \leq j}} \sum_{k=1}^{i} x_k v_k$$

On a plusieurs cas à considérer.

1 - le $i^{ème}$ objet est plus lourd que la capacité du sac

$\Rightarrow$ $V[i, j] = V[i-1, j]$ / $m_i > j$

2 - il est possible de prendre le $i^{ème}$ objet :

a - On le prend et la valeur du butin est celle du $i^{ème}$ objet plus celle du butin qu'on peut constituer à partir des $i-1$ 1^{ers} objets avec un sac de masse $(j - m_i)$

$\Rightarrow V[i, j] = v_i + V[i-1, j - m_i]$ / $m_i \leq j$

b - On ne le prend pas et le prob se ramène à la recherche du meilleur butin parmi les $i-1$ 1^{ers} objets : $V[i, j] = [i-1, j]$ / $m_i \leq j$

$\Rightarrow$ la valeur maximale du sous-ensemble optimal des i 1^{er} objets est donc le max de ces 2 dernières valeurs.

II | Fonction SAD-Rec (M, V : tab ; i, j : entier) : entier

Début

Si ($i = 0$) ou ($j = 0$) alors Retourner (0)

Sinon

Si M[i] > j alors

Retourner(SAD-Rec (M, V, i-1, j))

Sinon

j-M[i

Retourner(Max(SADRec(M,V,i-1,j) ; V[i]+SADRec(M,V,i-1,

Fsi

Fsi

Fin

III | $T(n) = 2T(n-1) + O(1)$

IV | $T(n) \geq 2T(n-1)$

$\geq 2[2T(n-2)]$

$\geq 2[2(2T(n-3))]$

$\geq 2^n \Rightarrow T(n) = \Omega(2^n)$

V

$i \backslash j$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

15 + 22

$\Rightarrow$ La valeur Max est donc $V[4,5] = 37$

VI

Procédure SAD_Dyn (var V : matrice, Mas, Val : tab, N, M : entier).

```
Début
    pour i de 0 à N Faire
        V[i,0] <- 0
    Fpour
    pour j de 0 à M Faire
        V[0,j] <- 0
    Fpour

    pour i de 1 à N Faire
        pour j de 1 à M faire
            si Mas[i] > alors V[i,j] <- V[i-1,j]
            sinon
            V[i,j] <- Max(Val[i] + V[i-1, j-Mas[i]] ; V[i-1, j])
            Fsi
        Fpour
    Fpour
Fin
```

Complexité : $T(N,M) = N * M + N + M$

si $N = M \Rightarrow \boxed{T(N) = O(N^2)}$

École Nationale d'Ingénieurs de Carthage
Département d'Informatique
***

Examen de la session de contrôle en

Algorithmique Avancée & Complexité

12 juillet 2018

Classe : 2IngInfo$_{A,B,C,D,E}$ Durée : $1^h30'$ Nombre de pages : 3 pages

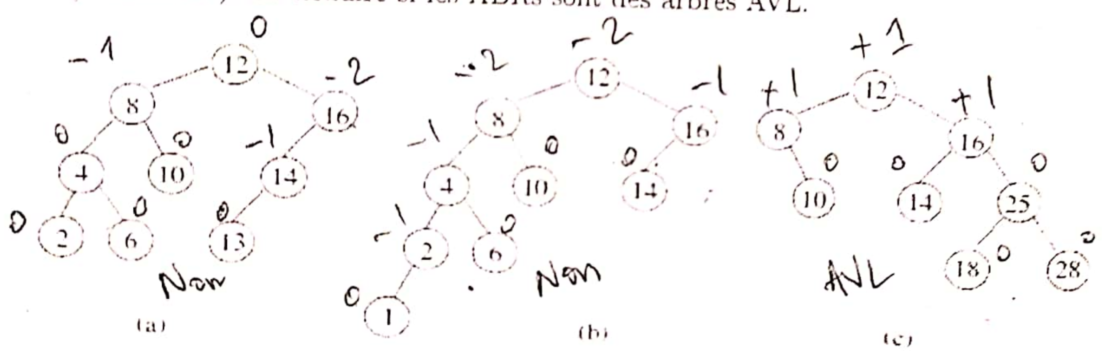
---

Il est conseillé de lire attentivement les énoncées avant de se plonger sur la feuille des réponses. ***Les copies propres et bien soignées sont très appréciées. Il sera tenu compte de la lisibilité des réponses.*** *Le barème donné est indicatif. Aucun document n'est autorisé.*
Bonne chance !

Exercice 1 (6 pts)

1. Étiqueter les arbres binaires de recherche de la Figure 1 par les facteurs d'équilibrage (« *balance* »). En déduire si les ABRs sont des arbres AVL.

2



(a) (b) (c)

FIGURE 1 – Arbres Binaires de Recherche

2

2. Le ou les arbres qui ne sont pas AVL, pourront le devenir en effectuant un rééquilibrage en utilisant des rotations. Appliquer les rotations nécessaires.

2

3. Sachant qu'il vérifie bel et bien la propriété des ABR, écrire la fonction récursive qui vérifie si un arbre de type AVL, passé en paramètre, respecte la propriété des AVL.

Problème I

Nous traitons dans ce problème une varaiante simple d'un problème classique et bien connu en algorithmique sous le nom du problème du « `sac à dos` »(ou encore « `bin packing problem` »en anglais).

Le problème du « `sac à dos` »est un problème d'optimisation qui peut être envisagé suivant plusieurs approches de programmation selon les variantes rencontrées.

Le problème se pose dans les termes suivants :

étant donnés n objets de valeurs $c_1, c_2, \ldots, c_n$ et de poids respectifs $\omega_1, \omega_2, \ldots, \omega_n$, comment remplir un sac à dos maximisant la valeur emportée

$$C_{max} = \sum_{i \in I} c_i$$

tout en respectant la contrainte

$$\sum_{i \in I} \omega_i \leq \Omega_{max}$$

- I désigne l'ensemble d'objets retenus dans le sac à dos parmi tous les objets disponibles ;
- C_{max} désigne la fonction objectif à maximiser ;
- Ω_{max} désigne la capacité maximale du sac à dos en terme de poids.

Partie A. Caractérisation de la solution (5 pts)

Soit l'ensemble suivant d'objets caractérisés par les valeurs et les poids détaillés dans le tableau 1 :

TABLE 1 – Exemple de problème de sac à dos

	x_1	x_2	x_3	x_4	x_5
C_i	3	3	1	5	4
ω_i	3	4	1	6	5

1

La capacité maximale de notre sac à dos est : $\Omega_{max} = 15$.

1. Deviner la solution optimale pour cet exemple. La solution optimale consiste à énumérer les objets retenus dans le sac à dos (les éléments de I) et leur valeur tôtale.

Pour résoudre ce problème, nous allons noter la fonction $f(n, \Omega_{max})$ la valeur maximale qu'il est possible d'atteindre avec les n objets :

$$f(n, \Omega_{max}) = \begin{cases} max(c_n + f(n-1, \Omega_{max} - \omega_n), f(n-1, \Omega_{max})), & \omega_n \leq \Omega_{max} \\ f(n-1, \Omega_{max}), & \text{sinon.} \end{cases} \quad (1)$$

2

2. Justifier la récurrence (1) ;

2

3. Montrer, à travers un petit exemple, que les sous-problèmes sont superposés (C'est-à-dire les mêmes sous-problèmes sont rencontrés plusieurs fois).

Partie B. Approche récursive (4 pts)

2

4. Écrire une fonction récursive qui calcule la valeur maximale du sac à dos. La fonction, qui est une application directe de la reccurence (1), a le prototype suivant `fonction f(C: tableau[1..N], W: tableau[1..N], i : entier): entier`. En considérant que $C[i]$ et $W[i]$ contiennent respectivement la valeur et le poids du i^{eme} objet

2

5. Donner une estimation asymptotique de la complexité au pire des cas de cette procédure.

Partie C. Résolution par programmation dynamique (5 pts)

La récurrence (1) se prête très bien à la programmation dynamique : nous allons utiliser un tableau F bidimensionnel de taille $(n+1) \times (\Omega_{max}+1)$ destiné à contenir les valeurs de $F[i,j]$ pour $i \in [0, N]$ et $j \in [0, \Omega_{max}]$. Nous prendrons bien entendu comme valeurs initiales $F[0,j] = 0$.

La résolution du problème revient à remplir le tableau F ligne par ligne. En effet les cases $F[i,j]$ ne dépendent que des cases $F[i-1, j-W[i]]$ et $F[i-1, j]$ qui se trouvent à la ligne précédente.

La valeur maximale du sac à dos est la valeur la plus grande de la dernière ligne du tableau F.

6. Remplir le tableau 2 en se référant à l'exemple du tableau 1. Retrouver la valeur C_{max} que vous avez devinée dans la question 1.

TABLE 2 – Tableau F

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x_1	1	0															
x_2	2	0															
x_3	3	0															
x_4	4	0															
x_5	5	0															

2

7. Écrire la procédure `Sac_a_dos(C: tableau[1..N], W: tableau[1..N], var F : tableau [0..N, 0..`Ω_{max}`])` qui calcule la matrice F ;
8. Calculer la complexité de cette procédure. En déduire une estimation asymptotique en notation $\mathcal{O}$

1

$F[1, 1] = 0$

$F[1, 2] = 0$

$F[1, 3] = 3$

Scanned with CamScanner

Corrigé - Examen Contrôle "Algo. Avancées"
2017-2018

Problème : Sac-à-Dos

1/ $f(\{x_1, x_2, x_3, x_4\}, 15) = x_1 + x_3(3, 4) + x_4(5, 6) + x_5(4, 5)$

1

$$\Rightarrow \boxed{\begin{array}{l} C = 13 \\ \omega = 15 = \Omega_{max} \end{array}}$$

2/ soit i le $i^{ème}$ objet parmi n à sélectionner et j son poids.

a/ $f[i, j]$ est le $i^{ème}$ objet parmi le i objet restants dans le sac de capacité max j.

- $f(n-1, \Omega_{max})$: le $n^{ème}$ objet est plus ~~je sel~~ lourd que le sac donc $\Rightarrow \boxed{f(n-1, \Omega_{max})}$

2

b/ il est possible de prendre le $n^{ème}$ objet :

- on le prend et la valeur du butin est celle du $n^{ème}$ objet plus celle du butin que l'on peut constituer à partir des $n-1$ premiers objets qui reste avec un sac de masse $(\Omega_{max} - \omega_n)$

$\Rightarrow$

$$\boxed{f(n, \Omega_{max}) = C_n + f(n-1, \Omega_{max} - \omega_n)}$$

- on ne le prend pas et le pb se ramène à la recherche du meilleure butin parmi les $n-1$ premiers objets:

$$\boxed{f(n, \Omega_{max}) = f(n-1, \Omega_{max})}$$

3/ 2

4/ Partie B

fonction SAD_rec (c, w: tableau[1...N] entier, i, Ω entier) : entier

Début

Si $w[i] > \Omega$ alors

Retourner (SAD_rec ($c, w, i-1$))

Sinon

Retourner (max (SAD_rec ($c, w, i-1, \Omega$)), SAD_rec ($c, w, i-1, \Omega_{max} - w[i]$) + $c[i]$)

fsi

2

5- $T(n) = 2T(n-1) + O(1)$

$= 2(2T(n-2) + O(1))$

$= 2(2(2(2 \ldots (2T(1) + O(1) + O(2))+ \ldots) + O(1)$

$\geq 2^{n-1} T(1) \Rightarrow \boxed{T(n) = \Omega(2^n)}$

Partie C

1- Procédure Sac_a_Dos(C, W : tab ; var F : matrice)

Debut

pour i de 0 à N faire

$F[i, 0] \leftarrow 0$

fpour

pour j de 0 à Ωmax faire

$F[0, j] \leftarrow 0$

fpour

2

pour i de 1 à N faire

pour j de 1 à Ω_{max} Faire

si $W[i] > j$ alors

$F[i, j] \leftarrow F[i-1, j]$.

sinon

$F[i, j] \leftarrow Max(F[i-1, j], C[i] + F[i-1, j - W[i]])$

fsi

fpour

fpour

8- $T(N, \Omega_{max}) = N * \Omega_{max} \Rightarrow O(n)$ 1

si $N = \Omega_{max} \Rightarrow \boxed{O(n^2)}$