

Examen d'algorithmique

Mercredi 14 janvier 2015 12h-15h / Aucun document autorisé

Mode d'emploi : Le barème est donné à titre indicatif. La qualité de la rédaction des algorithmes et des preuves sera très fortement prise en compte pour la note. On peut toujours supposer une question résolue et passer à la suite.

Exercice 1 : Diviser-pour-régner (3 points)

On considère l'algorithme ci-dessous :

```
Def P(Liste L[1..n]) :  
Si L est vide Alors Retourner [ ]  
Si L est [x] Alors Retourner [x]  
Si L est [x;y] Alors Retourner [y;x]  
Sinon  
  L1 = P(L[1.. n/3])  
  L2 = P(L[1+n/3..2n/3])  
  L3 = P(L[1+2n/3..n])  
  Retourner L3:L2:L1
```

Quelques précisions... le symbole ":" représente la concaténation de deux listes. On écrit "L est [x]" lorsque L est composé d'un unique élément dont la valeur est indiquée par la variable x, et "[x;y]" pour indiquer qu'il y a deux éléments dont les valeurs sont stockées dans les variables x et y... Enfin "L[i..j]" représente une copie de la sous-liste d'indices i à j (inclus).

1. Décrire ce que fait l'algorithme P appelé sur la liste [4; 1; 8; 7; 5; 3; 2; 9; 14; 17; 6]
2. Que fait l'algorithme P ?
3. Quelle est sa complexité si on suppose que les opérations de découpage et de concaténation prennent un temps constant ?
4. Quelle est sa complexité si on suppose que les opérations de découpage et de concaténation prennent un temps linéaire dans la taille de L ?

Exercice 2 : Partition équitable d'un ensemble de valeurs - 6 points

Étant donnée une liste de n valeurs entières, on veut les répartir dans deux sacs de manière la plus équilibrée possible (tels que les sommes des éléments de chaque sac soient le plus proche possible). On va chercher un algorithme qui calcule la **valeur** (= la somme des éléments contenu dans un sac) des deux sacs. Par exemple, si $L = \{6, 1, 3, 1\}$, un sac contiendra les objets $\{6\}$ et l'autre $\{1, 3, 1\}$ et les valeurs des sacs seront donc 6 et 5.

1. On suppose que l'on dispose d'une fonction $\text{Sac}(L, v)$ qui renvoie VRAI si et seulement si il est possible de faire, avec les objets de L , un sac dont la valeur vaut v . Par exemple, pour $L = \{2, 8, 3, 7\}$ et $v = 12$, c'est possible avec le sac $\{2, 3, 7\}$, mais si on prend $v = 14$, ce n'est pas possible...

Proposer un algorithme qui, étant donnée L , retourne des valeurs pour deux sacs le plus équilibrés possible. Par exemple, pour $L = \{6, 1, 3, 1\}$, l'algorithme retournera la paire $(6, 5)$ ou $(5, 6)$.

Donner sa complexité en supposant que la fonction $\text{Sac}(-, -)$ prend un temps constant.

2. Appliquer votre algorithme sur $L = \{2, 8, 9, 3, 7, 4\}$, puis sur $L = \{40, 20, 1, 2, 8\}$.
3. On cherche à présent à trouver un algorithme pour la procédure $\text{Sac}(L, v)$. Pour cela, on va calculer (et stocker dans un tableau) les valeurs de $\text{Sac}(L, v)$ pour $0 \leq v \leq V_{\max}$ où $V_{\max}$ est un entier fixé.
 - (a) Proposer un algorithme de programmation dynamique pour construire un tableau Booléen à deux dimensions $T[i, v]$ où $0 \leq i \leq n$ et $0 \leq v \leq V_{\max}$ tel que $T[i, v]$ vaut VRAI si et seulement si il est possible de faire un sac de valeur v avec les i premiers objets L . Justifier l'algorithme et donner sa complexité.
 - (b) Appliquer l'algorithme sur l'exemple $L = \{2, 8, 9, 3, 7, 4\}$ et $V_{\max} = 20$.
4. En déduire, l'algorithme complet pour le calcul des valeurs des deux sacs équilibrés (celui de la question 1 avec la procédure pour $\text{Sac}(L, v)$ de la question 3). Donner sa complexité.

Exercice 3 : Remplir un camion (8 points)

On s'intéresse ici à plusieurs problèmes où il faut remplir un camion avec des objets ayant des valeurs et des poids différents en cherchant à maximiser la valeur stockée dans le camion sans dépasser un poids total maximal K donné, et en respectant aussi diverses contraintes. Pour chacun de ces problèmes, on cherchera des algorithmes les plus efficaces possibles.

1. D'abord on s'intéresse au problème classique suivant :

Input : Un poids maximal K et un ensemble $\mathcal{E}$ de n ($n \geq 1$) objets (numérotés de 1 à n) pour lesquels on note $p(i)$ le poids du i -ième objet et $v(i)$ sa valeur.

Output : un sous-ensemble de $\mathcal{E}$ dont la somme des poids est inférieure ou égale à K et qui maximise la somme des valeurs. Formellement, on cherche donc des

$\epsilon_i \in \{0, 1\}$ pour $1 \leq i \leq n$ tels que $\sum_{i=1}^n \epsilon_i \cdot p(i) \leq K$ et tels qu'il n'existe aucun ensemble de valeurs $\epsilon'_i \in \{0, 1\}$ vérifiant :

$$\sum_{i=1}^n \epsilon'_i \cdot p(i) \leq K \quad \text{et} \quad \sum_{i=1}^n \epsilon'_i \cdot v(i) > \sum_{i=1}^n \epsilon_i \cdot v(i)$$

Proposer un algorithme de programmation dynamique pour résoudre ce problème. Justifier l'algorithme et donner sa complexité.

Appliquer votre algorithme sur l'exemple suivant : $K = 15$ et $\mathcal{E}$ défini par :

i	1	2	3	4	5
p	20	10	3	2	4
v	100	20	9	4	15

2. Reprendre le problème précédent pour les deux cas particuliers ci-dessous :
 - (a) Lorsque $v(i) = p(i)$ pour tout $1 \leq i \leq n$.
 - (b) Lorsque $v(i) = 1$ pour tout $1 \leq i \leq n$.
3. Maintenant, on suppose que les objets sont fractionnables : on peut mettre la moitié de l'objet 1, le tiers de l'objet 2, ... On cherche donc des valeurs rationnelles r_i dans l'intervalle $[0; 1]$ telles que ces r_i vérifient $\sum_{i=1}^n r_i \cdot p(i) \leq K$ et maximisent la valeur

totale $\sum_{i=1}^n r_i \cdot v(i)$.

Proposer un algorithme. Justifier l'algorithme et donner sa complexité. A quelle famille appartient-il ?

Reprendre l'exemple de la question 1 et appliquer votre algorithme.

4. On ajoute désormais à une contrainte supplémentaire : les objets (représentés par leur numéro) sont répartis dans plusieurs sous-ensembles $U_1, \dots, U_m$ et on ne peut choisir qu'au plus un élément de chaque U_j pour remplir le camion. Par exemple, si $U_1 = \{1, 2\}$, $U_2 = \{3, 4, 5\}$ et $U_3 = \{6, 7\}$, alors on peut remplir le camion avec $\{1, 3, 7\}$, $\{1, 3\}$, ou $\{2, 3, 7\}$, ... mais pas avec $\{1, 2, 7\}$, ou $\{1, 4, 5, 7\}$, ...

On cherche donc un ensemble de valeurs $\epsilon_i \in \{0, 1\}$ avec $0 \leq i \leq n$ tel que (1) pour tout sous-ensemble U_j et pour tous $k, h \in U_j$, on a $\epsilon_k = 0 \vee \epsilon_h = 0$ et (2)

$\sum_{i=1}^n \epsilon_i \cdot p(i) \leq K$, et qui maximise la somme $\sum_{i=1}^n \epsilon_i \cdot v(i)$.

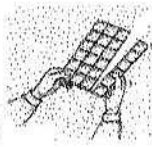
Proposer un algorithme pour trouver la valeur maximale possible pour un ensemble $\mathcal{E}$ et ses sous-ensembles $U_1, \dots, U_m$ et pour la borne K . On pourra calculer les valeurs $T[j, k]$ correspondant à la valeur maximale possible lorsque l'on se restreint aux sous-ensembles $U_1, \dots, U_j$ et pour un poids maximal k ...

Appliquer votre algorithme sur l'exemple suivant : $K = 15$ et $\mathcal{E}$ et sa partition $U_1 = \{x_1, x_2\}$, $U_2 = \{x_3, x_4\}$ et $U_3 = \{x_5, x_6\}$ défini par :

	U_1		U_2		U_3	
i	1	2	3	4	5	6
p	10	4	3	2	8	6
v	20	15	9	4	2	1

5. On considère le même problème qu'à la question précédente mais on suppose que les objets sont fractionnables : dans chaque sous-ensemble, nous pouvons prendre une fraction (dans l'intervalle $[0; 1]$) d'un unique objet...

Montrer qu'il n'est pas possible de limiter chaque sous-ensemble à l'objet ayant la "valeur massique" (i.e. $\frac{v(x_i)}{p(x_i)}$) la plus grande et d'utiliser l'algorithme de la question 3.

**Exercice 2 : (3 points)**

Un tablette de chocolat est composée de $n \times m$ carreaux. On souhaite séparer tous les carreaux, et pour cela la seule opération dont on dispose est de prendre un morceau de chocolat et de le couper en deux verticalement ou horizontalement (en suivant une ligne ou une colonne).

1. Proposer un algorithme pour séparer tous les carreaux. Quelle est sa complexité ? Appliquer le sur une tablette 2×5 .
2. Quel est le nombre minimal d'opérations permettant de séparer tous les carreaux (quel que soit l'algorithme) ? Votre algorithme est-il optimal ?

Annexe**Master theorem :**

Soient $a \geq 1$, $b > 1$, $f(n)$ une fonction positive et
$$t(n) = \begin{cases} a \cdot t(\frac{n}{b}) + f(n) & \text{si } n > 1 \\ \Theta(1) & \text{si } n = 1 \end{cases};$$

- Si $f(n) = O(n^{\log_b a - \epsilon})$ avec $\epsilon > 0$, alors $t(n) = \Theta(n^{\log_b a})$
- Si $f(n) = \Theta(n^{\log_b a})$, alors $t(n) = \Theta(n^{\log_b a} \cdot \log n)$
- Si $f(n) = \Omega(n^{\log_b a + \epsilon})$ pour $\epsilon > 0$ et si $\exists c < 1$ tel que $a \cdot f(\frac{n}{b}) \leq c \cdot f(n)$ pour n assez grand, alors $t(n) = \Theta(f(n))$