Lycée Lakanal
PC

Tp programmation dynamique

Introduction :

A la nuit tombée, un voleur pénétre par effraction dans un musée.
On suppose que le voleur a une estimation parfaite du poids et du prix de chacun des objets exposés. En outre, il dispose d'un sac à dos de dimension suffisamment grande pour pouvoir contenir tous les objets que le voleur désire. En revanche, le malfrat ne peut pas porter un poids supérieur à p (p réel positif donné). La question pour le voleur est alors d'optimiser le remplissage de son sac à dos en fonction du poids et de la valeur de chacun des objets qu'il a repérés, et dont il a représenté la liste sous forme de tableau :

objet	poids (kg)	valeur (en euros)
objet 1	16	2500
objet 2	5	800
objet 3	8	950
objet 4	2	200
objet 5	5	400

Formalisation du problème général :

On considère un sac à dos dont la capacité est notée C.
On considère $n \in \mathbb{N}$ et n objets, dont on note $\rho_0, \ldots, \rho_{n-1}$ les poids et $v_0, \ldots, v_{n-1}$ les valeurs.
Dans un premier temps, nous supposerons que les poids et les valeurs sont des entiers strictement positifs. Dans les programmes, ces valeurs seront rentrées dans deux tableaux `rho` et `v`.
L'objectif est de produire des algorithmes permettant de choisir de manière relativement optimale un sous-ensemble X de $\{0, \cdots n-1\}$ tel que $\sum_{k\in X} \rho_k \leqslant C$ et pour lequel la valeur totale $\sum_{k\in X} v_k$ soit maximale.

Une heuristique gloutonne :

Dans un premier temps, on va adopter une heuristique gloutonne consistant à donner la priorité aux objets ayant le meilleur rapport valeur/poids.
Pour ce faire, dans la suite, pour tout $i \in \{0, \cdots n-1\}$, on notera $\alpha_i = \frac{v_i}{\rho_i}$ ce rapport.

1. Écrire une fonction `rapport_valeur_poids` prenant en arguments `rho` et `v` et renvoyant le tableau $[\alpha_0, \alpha_1, \ldots, \alpha_{n-1}]$

2. Écrire une fonction `meilleurObjetRestant` qui prend en entrée le tableau `alpha` des α_i ainsi que le tableau `pris` (ce tableau de booléens liste si l'objet i est encore disponible ou pas) et qui renvoie le numéro l'objet de meilleur rapport valeur/poids parmis les objets pas encore placés dans le sac à dos (dans le cas oú tous les objets ont déjà été pris, on renverra -1).
3. Écrire une fonction `meilleurObjetRestantMieux` qui améliore le fonctionnement de la fonction précédente en prenant en compte le cas où, au cours du remplissage, l'objet de priorité maximale ne rentre pas dans le sac à dos (dans ce cas le programme précédent s'arrêtait alors qu'il aurait fallu passer à l'objet de priorité suivante).
4. En déduire la fonction finale `glouton` prenant en arguments `v`, `rho`, la capacité du sac à dos et renvoyant la liste des objets choisis, ainsi que la valeur totale de ces objets.
5. Justifier la terminaison du programme précédent.
6. Calculer la complexité de cette fonction en fonction du nombre d'objets n (on comptera le nombre de comparaisons entre éléments du tableau `alpha`)
7. Indiquer comment, au moyen d'un d'un tri on peut rendre la stratégie gloutonne plus efficace. Quelle est la complexité de cette nouvelle méthode ?
8. Trouver un exemple dans lequel l'algorithme glouton ne fournit pas la solution optimale.

Programmation dynamique :

Pour tout $C \in \mathbb{N}$, et tout $i \in \{0, \cdots n-1\}$, on pose $V(C,i)$ la valeur maximale des objets qu'on peut mettre dans un sac de capacité C en choisissant uniquement des objets parmi les i premiers (soit dans $\{0, \cdots i-1\}$). Si $C \leqslant 0$, nous convenons que $V(C,i) = 0$

9. Que vaut $V(C,i)$ lorsque $C = 0$ ou $i = 0$?
10. Démontrer que pour tout $C \in \mathbb{N}$ et $i \in \{0, \cdots n-1\}$,

$$V(C, i+1) = \max\left(V(C,i), V\left(C - \rho_i, i\right) + v_i\right)$$

À présent, soit $C_0 \in \mathbb{N}$. La stratégie est de créer une matrice V de format $(C_0+1, n+1)$ telle que pour tout $(c,i) \in \{0, \cdots C_0\} \times \{0, \cdots n\}$, v[c][i] contiendra, une fois l'algorithme fini, $V(c,i)$.

11. Écrire une fonction prenant C_0, v, rho et renvoyant la valeur maximale des objets qu'on peut mettre dans un sac de capacité C_0. L'algorithme utilisé consistera à créer puis remplir la matrice v grâce á la formule de la question précédente.
12. Calculer la complexité de la fonction que vous venez d'écrire. Comparer cette complexité avec celle de la méthode naïve qui consiste à essayer toutes les combinaisons possibles d'objets.

Reconstruction d'une solution optimale :

Le programme précédent a un défaut majeur : il ne calcule que la valeur maximale des objets qu'on peut mettre dans le sac á dos, et non la liste des objets choisis.

On peut employer deux méthodes pour obtenir la liste proprement dite.

La première consiste à modifier le programme précédent pour remplir, simultanément à la matrice V, une autre matrice `listes0bjets` telle que pour tout $c, i \in \{0, \cdots c_0\} \times \{0, \cdots n\}$, listes0bjets $[c][i]$ contiendra la liste des objets, choisis dans $\{0, \cdots i-1\}$, pour remplir au mieux un sac de capacité c.

La seconde consiste à retrouver cette liste après coup en lisant le tableau V obtenu. L'idée est que si $V(c, i+1) = V(c,i)$, cela signifie que l'objet i n'a pas été choisi. Et réciproquement, si $V(c, i+1) = V\left(c - \rho_i, i\right) + V_i$ c'est que l'objet i a été pris.

13. Programmer une fonction prenant en entrée la matrice V remplie et renvoyant la liste effective des objets choisis.

Mémoïzation :

14. La méthode dynamique mise en oeuvre précédemment (de type bottom-up) conduit-elle à calculer des valeurs inutiles de V ?
15. Écrire une méthode up-bottom basée sur une fonction récursive optimisée par mémoïzation.

Tp programmation dynamique