Informatique

2
Dictionnaires et programmation dynamique

TD2-5

Ordonnancement de tâches pondérées

Présentation

Soit E un ensemble fini composé de n objets $E = \{e_0, e_1, \dots, e_{n-1}\}$ tels que chaque objet possède un poids $p_i > 0$ (contrainte) et une valeur $v_i > 0$ (objectif) stockés dans les listes $L_p = [p_0, \dots, p_{n-1}]$ et $L_v = [v_0, \dots, v_{n-1}]$. On souhaite déterminer le sous ensemble F de E maximisant la somme total de des valeurs de ses éléments, pour un poids limité à une valeur maximale P_{max}.

Applications

L'application classiquement trouvée pour cet algorithme est celle du sac à dos d'un voleur qui veut partir avec un gros butin. En effet, pour une contrainte de poids (15 kg), de volume (20L), de temps (10 minutes), il cherche à emporter le plus de valeur possible.

D'une manière générale, et on trouvera beaucoup d'applications, il faut répondre à la question :

Que faut-il choisir pour maximiser mon gain sans dépasser la contrainte à laquelle chaque choix participe

Algorithme par force brute

On va dans un premier temps programmer une méthode explorant toutes les possibilités afin de trouver une solution au problème posé.

Si on appelle E l'ensemble des éléments e_i disponibles ayant un poids p_i dans la liste L_p et une valeur v_i dans la liste L_v, on souhaite créer l'ensemble des choix possibles parmi E en stockant dans une liste des sous listes du type $[L_{p_i}, L_{v_i}]$ avec L_{p_i} la liste de l'ensemble des poids des objets choisis, et L_{v_i} la liste de l'ensemble de leurs valeurs.

Question 1: Créer la fonction explore(Lp,Lv) renvoyant la liste attendue
Aide : on pourra ajouter chaque élément $[p_i, v_i]$ à l'ensemble des résultats obtenus précédemment

Vérifier :
```
>>> explore([2,5,3],[1,2,3])
[[], [[2, 1]], [[5, 2]], [[2, 1], [5, 2]], [[3, 3]], [[2, 1]
, [3, 3]], [[5, 2], [3, 3]], [[2, 1], [5, 2], [3, 3]]]
```
En ne regardant que le terme de gauche de chaque liste de deux termes, à partir de L=[[]]
- J'ajoute 2 à chaque sous liste de L ([2]) que j'ajoute à L, soit L=[[],[2]]
- J'ajoute 5 à chaque sous liste de L ([5,[2,5]]) que j'ajoute à L, soit L=[[],[2],[5],[2,5]]
- Etc.

Question 2: Estimer la complexité en temps de explore
Question 3: Proposer la fonction sommes(Exp) prenant en argument l'ensemble des listes Exp issue de l'exploration, et renvoyant une liste Ls de listes de type [p,v,i] où p est le poids et v la valeur de tous les éléments de la sous liste Exp[i]. Avant renvoie, on triera Ls avec la méthode sort

Vérifier :
```
>>> Exp = explore([2,5,3],[1,2,3])

>>> sommes(Exp)
[[0, 0, 0], [2, 1, 1], [3, 3, 4], [5, 2, 2],
[5, 4, 5], [7, 3, 3], [8, 5, 6], [10, 6, 7]]
```

Question 4: Préciser l'intérêt d'utiliser la méthode sort réalisant un tri lexicographique
Question 5: Créer la fonction solution(Ls,Pmax) prenant en argument la liste Ls renvoyée par somme le poids max Pmax, et renvoyant l'indice de F dans Exp, ainsi que le poids p et la somme v obtenus
Question 6: Utiliser les fonctions réalisées pour trouver F dans Lp = [2,5,2,1,1,7] et Lv = [10,30,20,10,5,30]
Cette méthode est évidemment très coûteuse et on lui préfèrera la suite.

Présentation de la méthode en programmation dynamique

Pour trouver F, on utilise une méthode de programmation dynamique. Pour rendre l'explication concrète, supposons un sac à dos et des objets ayant un poids et une valeur, le sac devant obtenir la plus grande valeur pour un poids Pmax. On résout donc le sous-problème suivant :

Quelle est la valeur maximale atteinte avec des éléments parmi les i premiers éléments de E pour un poids du sac valant au plus j

On créer un tableau à deux dimensions $(n+1)(P_{max}+1)$ rempli des valeurs V atteintes :

$$\forall \boldsymbol{i} \in [\boldsymbol{0}, \boldsymbol{n}], \forall \boldsymbol{j} \in [\boldsymbol{0}, \boldsymbol{P_{max}}], \boldsymbol{D(i,j)} = \begin{cases} \boldsymbol{0 \; si \; i = 0} \\ \boldsymbol{max}\begin{pmatrix} \boldsymbol{D(i-1,j)} \\ \boldsymbol{D(i-1,j-p_{i-1}) + v_{i-1}} \end{pmatrix} \boldsymbol{si \; j \geq p_{i-1}} \\ \boldsymbol{D(i-1,j) \; sinon} \end{cases}$$

En effet : Pour trouver la valeur $V_{i,j}$ contenue dans le sac avec au plus i éléments parmi les premiers éléments de $E_i = \{e_0, e_1, \dots, e_{i-1}\}$ de E, on estime la condition : $j - p_{i-1} \geq 0 \Leftrightarrow j \geq p_{i-1}$ qui signifie que l'objet $e_{i-1}(p_{i-1}, v_{i-1})$ possède un poids p_{i-1} lui permettant de faire partie du sac de poids au plus égal à j, puis :

- Si $j \geq p_{i-1}$: on peut ajouter e_{i-1} au sac de poids $j - p_{i-1}$ dont la valeur $V_{i-1,j-p_{i-1}}$ a déjà été évaluée, et sa valeur croit : $V_{i,j} = V_{i-1,j-p_{i-1}} + v_{i-1}$
- Sinon : on n'ajoute rien dans le sac de poids au plus égal à j: $V_{i,j} = V_{i-1,j}$

La dernière case $\boldsymbol{D(n, P_{max})}$ donne la valeur maximale atteinte en piochant parmi tous les éléments de $\boldsymbol{E}$ pour un poids au plus égal à $\boldsymbol{P_{max}}$

Exemple de remplissage pour $L_p = [2,5,2]$ et $L_v = [1,2,3]$:

p_i	v_i	i\j	0	1	2	3	4	5
2	1	0	0	0	0	0	0	0
5	2	1	0	0	1	1	1	1
2	3	2	0	0	1	1	1	2
		3	0	0	3	3	4	4

Remarques :

- La valeur maximale sur une ligne i est la somme des valeurs des i premiers éléments
- Les poids étant strictement positifs, la première colonne sera forcément nulle
- A partir du moment où une colonne contient un nombre, il ne peut plus diminuer
- La flèche rouge illustre le remplissage de la case $(3,2)$ à partir de la case $(2,0)$ puisque le poids de $e_2 = 2$ peut être ajouté au sac vide pouvant peser $j = 2$ et la valeur passe de 0 à $v_2 = 3$
- Les ajouts de termes à la solution n'ont lieu qu'en diagonale (ajout de poids vers la droite et d'éléments vers le bas)
- A la dernière ligne :
    - Chaque case représente la valeur maximale atteinte en choisissant parmi l'ensemble des éléments de E pour un poids du sac valant au plus j
    - Si une valeur V est atteinte à la colonne j, la valeur à la colonne $j+1$ est forcément au moins égale à V puisque qui peut prendre plus, peut prendre moins. Ainsi, la dernière ligne est toujours croissante.

Programmation de l'algorithme itératif

Question 7: Mettre en place la fonction OTP_it(Lp,Lv,Pmax) prenant en argument les listes Lp des poids, Lv des valeurs et Pmax, et renvoyant le dictionnaire attendu
Question 8: Créer une fonction matrice(f,Lp,Lv,Pmax) renvoyant un array représentant le tableau associé au dictionnaire créé par f(Lp,Lv,Pmax) et vérifier votre tableau. On initialisera les valeurs à np.inf.
Question 9: Préciser la complexité de la fonction OTP_it

Programmation de l'algorithme récursif

Question 10: Proposer une première fonction récursive OTP_rec_ij(i,j,Lp,Lv,Pmax) renvoyant le résultat attendu
Question 11: Préciser la complexité de la fonction OTP_rec_ij dans le pire des cas à préciser
Question 12: Proposer une fonction OTP_rec_mem(Lp,Lv,Pmax) récursive mémoïsée renvoyant le même dictionnaire que PEEP_it à une différence près, à expliquer

Etapes intermédiaires

Que ce soit par l'algorithme récursif ou itératif, on obtient le même dictionnaire. On souhaite remonter les étapes permettant d'établir F.
A partir de la case $D(i,j) = (n, P_{max})$ indiquant la valeur maximale atteinte, et tant que $j > 0$ (aucun élément à ajouter):

- Si $D(i,j) = D(i-1,j)$, aucun ajout n'avait eu lieu
- Sinon, avec $c_1 = D(i-1,j)$ et $c_2 = D(i-1, j-p_{i-1}) + v_{i-1}$
    - Si la condition c_2 ne peut être évaluée : Il n'y a eu aucun ajout
    - Sinon :
        - Si $c_1 = max(c_1, c_2)$: Il n'y a eu aucun ajout
        - Sinon ($c_2 = max(c_1, c_2)$) : Il y a eu ajout de $e_{i-1}(p_{i-1}, v_{i-1})$

        N'avoir aucune des conditions vérifiée est impossible

Illustrons le chemin avec la table précédente :

p_i	v_i	i\j	0	1	2	3	4	5
2	1	0	0	0	0	0	0	0
5	2	1	0	0	1	1	1	1
2	3	2	0	0	1	1	1	2
		3	0	0	3	3	4	4

Question 13: Créer la fonction OTP(Lp,Lv,Pmax) créant le dictionnaire et renvoyant F en remontant la solution comme proposé ci-dessus. F sera la liste des listes [p,v] des ei choisis

Application

Un voleur peut porter 20 kg de matériel, et doit choisir dans la liste ci-dessous. Quel est le choix le plus judicieux ?

N°	Objet	Poids (g)	Valeur (€)
1	iPad	490	300
2	Bague	5	700
3	Téléviseur	15000	2700
4	Billet	1	500
5	iPhone	194	1200
6	Boite à bijoux	722	400
7	Collection	9564	2000
8	Casque audio	200	100
9	Lampe	3567	200
10	Barre de son	4587	350
11	Tableau	3250	8000
12	Routeur	752	100

Afficher dans la console :

- La solution retenue: liste des couples $e_i = [p_i, v_i]$
- Nombre d'objets
- Poids total atteint
- Valeur totale embarquée