Informatique

2
Dictionnaires et programmation dynamique

TD2-4

Partition équilibrée d'entiers positifs

Présentation

Soit E un ensemble fini composé de n entiers strictement positifs $E = \{e_0, e_1, \dots, e_{n-1}\}$. On souhaite déterminer deux sous-ensembles F et G formant une partition de E, c'est-à-dire $E = F \cup G$ et $F \cap G = \emptyset$ tels que la somme des éléments de F soit aussi proche que possible de la somme des éléments de G. On l'appelle partition équilibrée d'entiers positifs (PEEP).
Ce problème revient à chercher un sous ensemble F de E dont la somme des élément S_F est la plus proche de la demi-somme $S = \frac{S_E}{2}$ des éléments de E.

Applications

A partir du moment où un entier représente une donnée réelle (niveaux de joueurs, prix de différents éléments, poids), cet algorithme permet de répartir de manière équilibrée l'ensemble des éléments en deux familles similaires. On peut par exemple répartir :

- Des joueurs dans deux équipes afin d'avoir des niveaux homogènes
- Des sacs ou des passagers de poids différents dans un avion afin d'équilibrer celui-ci
- Un héritage
- Un ensemble de pièces

Il est aussi possible d'utiliser cet algorithme pour trouver une choix d'éléments permettant de respecter un poids total imposé, par exemple les bagages à choisir parmi une liste afin d'en prendre le plus possible sans dépasser une limite de poids prédéfinie.

Cette liste d'exemples n'est pas exhaustive.

Algorithme par force brute

On va dans un premier temps programmer une méthode explorant toutes les possibilités afin de trouver la partition F d'une PEEP (F,G) d'une liste E. F est définie comme la partition de somme des termes la plus grande possible et inférieure ou égale à $S = \left\lfloor \frac{S_E}{2} \right\rfloor$

Question 1: Créer la fonction explore(E) renvoyant l'ensemble des sous listes que l'on peut créer à partir de E

Aide : on ajoutera à l'ensemble des résultats en cours, les mêmes résultats complétés du nouvel élément de E

Vérifier :

```
>>> explore([2,5,10])
[[], [2], [5], [2, 5], [10], [2, 10], [5, 10], [2, 5, 10]]
```

A 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 [s,i] où s est la somme de tous les éléments de la sous liste Exp[i]. Avant renvoie, on triera Ls avec la méthode sort

Vérifier :

```
>>> Exp = explore([2,5,10])

>>> sommes(Exp)
[[0, 0], [2, 1], [5, 2], [7, 3], [10, 4], [12, 5], [15, 6], [17, 7]]
```

Dans l'exemple ci-dessus, la somme des termes vaut $S_E = 2 + 5 + 10 = 17$. On souhaite trouver une répartition telle qu'un sous ensemble d'éléments soit le plus proche possible de la demi-somme $S_{max} = \left\lfloor \frac{S_E}{2} \right\rfloor = 8$. La solution est donc le couple $[7,3]$ (somme 7 atteinte avec les éléments à l'indice 3 dans Exp). On réparti donc les éléments en deux familles $F = [2,5]$ et $[10]$ de sommes respectives 7 et 10.

Question 4: Créer la fonction solution(Ls,Smax) prenant en argument la liste Ls renvoyée par la fonction « sommes » et la somme Smax espérée, et renvoyant l'indice dans Exp de F et la somme atteinte

Question 5: Utiliser les fonctions réalisées pour trouver F dans [75,69,43,90,110,58,62,72,86,25]

Cette méthode est évidemment très coûteuse et on lui préfèrera la suite.

Présentation de la méthode en programmation dynamique

Pour trouver F et G, on utilise une méthode de programmation dynamique traitant le sous problème suivant :

Pour $\boldsymbol{i} \in [\![\mathbf{0}, \boldsymbol{n}]\!]$ et $\boldsymbol{j} \in [\![\mathbf{0}, \boldsymbol{S}]\!]$ avec $\boldsymbol{S} = \left\lfloor \frac{S_E}{2} \right\rfloor$, est-il possible de trouver un sous-ensemble avec au plus $\boldsymbol{i}$ éléments parmi les premiers éléments $\boldsymbol{E_i} = \{\boldsymbol{e_0}, \boldsymbol{e_1}, \dots, \boldsymbol{e_{i-1}}\}$ de $\boldsymbol{E}$ dont la somme est égale à $\boldsymbol{j}$?

On créer un tableau de dimensions $(n+1)(S+1)$ rempli de booléens répondant à cette question :

$$\forall \boldsymbol{i} \in [\mathbf{0}, \boldsymbol{n}], \forall \boldsymbol{j} \in [\mathbf{0}, \boldsymbol{S}], \boldsymbol{D(i,j)} = \begin{cases} \boldsymbol{True\ si\ j = 0} \\ \boldsymbol{False\ si\ i = 0\ et\ j \neq 0} \\ \boldsymbol{D(i-1,j)\ ou\ \left(j \geq e_{i-1}\ et\ D(i-1, j-e_{i-1})\right)} \end{cases}$$

En effet :

- $D(i,0)$ signifie « atteindre une somme nulle » avec au plus les i premiers éléments de E: c'est vrai, avec aucun éléments
- $D(0,j)$ avec $j \neq 0$ signifie « atteindre la somme j avec 0 éléments », ce qui n'est pas possible
- S'il est possible d'atteindre la somme j avec les i premiers éléments de E, il est toujours possible d'atteindre j avec les $i+1$ premiers éléments
- Si e est le prochain élément ajouté, et s'il est possible d'atteindre une somme $j - e \geq 0$ (ie. $j \geq e$) avec les $i-1$ premiers éléments de E, il est possible d'atteindre j avec les i premiers éléments en ajoutant e. Ainsi, toute prise en compte d'un nouvel élément se fait en diagonale.

La plus grande valeur de $\boldsymbol{j}$ telle que $\boldsymbol{D(n,j) = True}$ indique la plus grande demi-somme atteignable avec tous les éléments de $\boldsymbol{E}$

Exemple de remplissage en notant 1=True et 0=False pour $E = \{1,2,7\}$ de demi-somme égale à 5 :

e_i	i\j	0	1	2	3	4	5
1	0	1	0	0	0	0	0
2	1	1	1	0	0	0	0
7	2	1	1	1	1	0	0
	3	1	1	1	1	0	0

Quelques détails :

- Case $(1,1)$: Est-il possible d'atteindre la somme $j = 1$ en $i = 1$ élément ?
  - $D(i-1,j) = D(0,1)$ vaut 0, ce n'est pas la condition vérifiée
  - $\left(j \geq e_{i-1}\ et\ D(i-1, j-e_{i-1})\right) = (1 \geq 1\ et\ D(0,0) = 1)$: il est possible d'atteindre 1 avec un élément car il est possible d'atteindre 0 en 0 élément auxquels on ajoute un élément valant 1
- Case $(1,2)$: Est-il possible d'atteindre la somme $j = 2$ en $i = 1$ premier élément de E ? La réponse est non.
- Case $(2,3)$: Après avoir compris la case $(1,1)$, regardez la flèche rouge.
- Toute nouvelle ligne contient toujours au minimum les True de la ligne précédente.
- Si la case tout en bas à droite $D(n,S) = True$, une solution de partitionnement parfait existe
- Signification de la dernière ligne : Est-il possible d'atteindre j avec tous les termes de E ?

Programmation de l'algorithme itératif

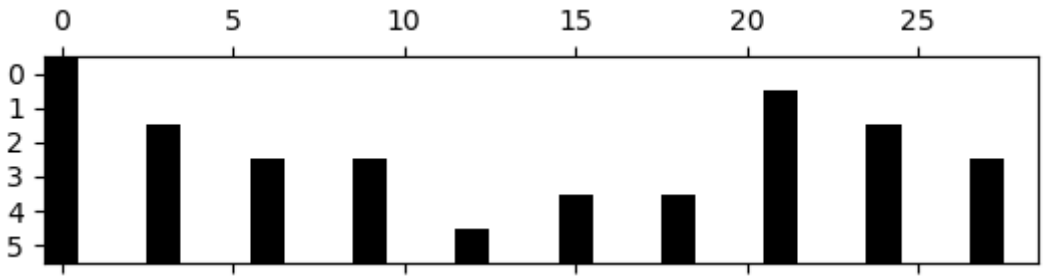
Question 6: Mettre en place la fonction PEEP_it(E) prenant en argument la liste d'entiers positifs E et renvoyant le dictionnaire attendu
Question 7: Créer une fonction matrice(f,E) renvoyant un array représentant le tableau associé au dictionnaire créé par f(E) et vérifier votre tableau
Question 8: Préciser la complexité de la fonction PEEP_it

On introduit le code suivant :

```
from matplotlib import pyplot as plt
plt.close('all')
def Spy(fig,M):
    plt.figure(fig)
    plt.spy(M)
    plt.show()

E = [21,3,6,15,12]
A = matrice(PEEP_it,E)
Spy(1,A)
```

Question 9: Utiliser cette fonction pour afficher les True dans la matrice issue de PEEP_it



Question 10: Mettre en place une fonction S_max(E) renvoyant la plus grande valeur de S atteignable depuis E

Programmation de l'algorithme récursif

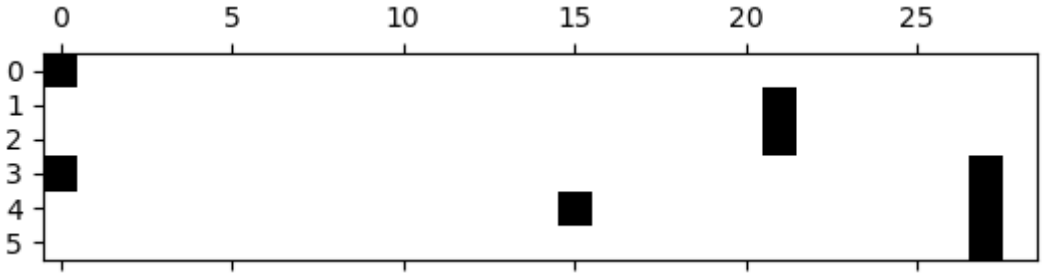
Question 11: Proposer une première fonction récursive PEEP_rec_ij(i,j,E) renvoyant le résultat True ou False attendu
Question 12: Préciser la complexité de la fonction PEEP_rec_ij dans le pire des cas à préciser
Question 13: Proposer une fonction PEEP_rec_mem(E) récursive mémoïsée renvoyant le même dictionnaire que PEEP_it

Si vous n'obtenez pas la même image (une image blanche ?), avez-vous pensé au fait que l'algorithme récursif peut ne pas passer sur toutes les cases ? il faut pourtant remplir tout le dictionnaire dans la question précédente.

On peut remarquer qu'il n'est pas utile de calculer tout le dictionnaire pour avoir le résultat de la somme réalisable que nous avions obtenu avec S_max, et que le fait de l'obtenir aura créé tout le chemin nécessaire pour remonter la solution.

Question 14: Proposer une fonction PEEP_rec_mem_opt(E) récursive mémoïsée renvoyant un dictionnaire optimisé contenant le stricte nécessaire
Remarque : Evidemment, on n'utilisera pas les résultats de la fonction S_max.

Vérifier que vous obtenez ceci :



Etapes intermédiaires

Que ce soit par l'algorithme récursif ou itératif, on obtient le même dictionnaire. On souhaite remonter les étapes permettant d'établir des deux ensembles F et G.

On appelle F la partition étudiée dans tous nos algorithmes dont la demi-somme ne dépasse pas S et G son complément dont la demi-somme dépasse S.
Il faut partir de la dernière ligne à la valeur de j renvoyée par la fonction S_max.
Selon la condition respectée :

- C1 : $D(i-1,j) = True$: On remonte d'une ligne $(i-1,j)$ et on n'ajoute pas l'élément e_i à F, on l'ajoute donc à G
- C2 : $j \geq e_{i-1}$ *et* $D(i-1,j-e_{i-1}) = True$: On va à la case $(i-1,j-e_{i-1})$ une ligne au-dessus et e_{i-1} cases à gauche, et on ajoute e_{i-1} à F

Alors, tant que $i > 0$ ou $j > 0$: je privilégie C1

- Si C1 ($\forall$C2) : Action associée à C1
- Sinon (C2) : action associée à C2
  Avoir les deux conditions False est impossible puisque par construction, arrivés à un True, il fallait des True avant. Il n'y a donc pas à traiter d'autres cas.

Soit le tableau pour $E = \{1,1,2,1\}$, objectif $S = 2$.

<table>
<tr><th>Chemin en privilégiant C1</th><th>Chemin en privilégiant C2</th></tr>
<tr><td>
<table>
<tr><th>e_i</th><th>i\j</th><th>0</th><th>1</th><th>2</th></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>3</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td>4</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td><td>
<table>
<tr><th>e_i</th><th>i\j</th><th>0</th><th>1</th><th>2</th></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>2</td><td>2</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>3</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td>4</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>$F = \{1,1\}$
$G = \{1,2\}$</td><td>$F = \{1,1\}$
$G = \{2,1\}$</td></tr>
</table>

Question 15: Créer la fonction PEEP(E) créant le dictionnaire et renvoyant F et G en remontant la solution comme proposé ci-dessus

Application 1

Voici la liste des passagers et de leurs poids devant embarquer dans un avion.

Plats	Tarif
PAX1	75
PAX2	69
PAX3	43
PAX4	90
PAX5	110
PAX6	58
PAX7	62
PAX8	72
PAX9	86
PAX10	25

Est-il possible d'équilibrer l'avion en répartissant parfaitement la moitié du poids à gauche, l'autre moitié à droite ? Quelle répartition réaliser ?

Application 2

Soit l'extrait d'une carte de restaurant avec les plats et leurs tarifs :

Plats	Tarif
Poulet	17
Bœuf	15
Salade	12
Crevettes	19
Pizza	18
Pâtes	13
Frites	7
Soupe	9

La somme disponible est 70€. L'objectif est de trouver un assortiment de plats pour plusieurs personnes, tous différents, permettant d'être au plus près de la somme disponible.

En adaptant les fonctions réalisées précédemment, trouver la somme dépensée, et les plats achetés