TP

5

RÉCURSIVITÉ

Exercice 1 – exponentiation rapide

On considère la méthode d'exponentiation rapide fondée sur la décomposition suivante :

$$x^n = \begin{cases} 1 & \text{si } n = 0 \\ (x^2)^{\frac{n}{2}} & \text{si } n \text{ est pair} \\ x \cdot (x^2)^{\frac{n-1}{2}} & \text{si } n \text{ est impair} \end{cases}$$

Écrire une fonction récursive `exporap(x, n)` qui renvoie x^n en utilisant la décomposition précédente.

Exercice 2

Écrire une fonction récursive `maxr(a)` qui prend pour argument une liste a de nombres et qui renvoie le plus grand des nombres de la liste (sans utiliser la fonction `max`).

Exercice 3

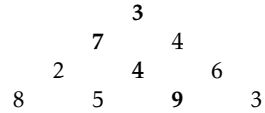
On considère la suite de Fibonacci définie sur $\mathbb{N}$ par $F_0 = 0$, $F_1 = 1$ et :

$$\forall n \in \mathbb{N},\ F_n = F_{n-1} + F_{n-2}.$$

Écrire une fonction récursive **avec mémoïsation** d'en-tête `F(n)` qui calcule le terme F_n.

Exercice 4

On considère la situation étudiée au TP4 suivante. En partant du sommet du triangle ci-dessous et en se déplaçant vers les nombres adjacents de la ligne inférieure, le total maximum que l'on peut obtenir pour relier le sommet à la base est égal à 23 :



$$3 + 7 + 4 + 9 = 23.$$

On modélise un tel triangle à l'aide d'une liste de listes `t`.
Pour tout $(i, j) \in [\![0, n-1]\!]^2$ avec $i \leqslant j$, on note $S_{i,j}$ le plus grand total en partant du coefficient à la ligne i colonne j.
Le tableau `t` permet d'obtenir directement les valeurs $S_{n-1,j}$, et pour $0 \leqslant j \leqslant i \leqslant n-2$, on dispose de la relation de récurrence suivante :

$$S_{i,j} = t[i][j] + \max\{S_{i+1,j}, S_{i+1,j+1}\}.$$

Écrire une fonction récursive **avec mémoïsation** d'en-tête `S(t)` qui calcule le total maximal pour un chemin reliant le sommet du triangle à sa base.
Faire ensuite les modifications adéquates pour obtenir un chemin réalisant ce maximum.

Exercice 5 - reprise du TP3

Une entreprise de livraison dispose de plusieurs locaux en France et chacun possède plusieurs camions de livraisons. Celle-ci souhaite optimiser le chargement de ses camions pour diminuer ses frais de fonctionnement. Chaque camion de l'entreprise peut charger une cargaison jusqu'à un poids maximal noté P_{max}. L'entreprise dispose de différentes informations provenant de ses clients :

- le poids de chaque produit p_i (chaque client propose un seul produit) ;
- la valeur v_i associée au transport de chaque produit : c'est-à-dire l'argent gagné par l'entreprise si elle réalise le transport de ce produit.

En considérant que l'entreprise dispose de n clients, l'entreprise cherche donc à trouver une liste d'indices notée I contenue dans $\{1,\dots,n\}$ telle que :

$$\sum_{i\in I} p_i \leqslant P_{max} \quad \text{(respect du poids maximal)} \quad (1)$$

et :

$$\sum_{i\in I} v_i \text{ soit maximal (optimisation du profit pour l'entreprise).} \quad (2)$$

Dans toute la suite, les poids seront donnés en centaines de kilogrammes et les valeurs en centaines d'euros. On stocke alors les différentes informations dans trois listes :

- `Pr` est la liste des produits proposés par les clients (numérotés de 1 à n inclus) ;
- $P = [p_1,\dots,p_n]$ est la liste des poids associés aux produits ;
- $V = [v_1,\dots,v_n]$ est la liste des valeurs associées aux produits.

On considère pour simplifier que les poids des produits sont des entiers, ainsi que P_{max}. Nous introduisons une méthode récursive pour résoudre ce problème d'optimisation :

- pour chacun des produits, deux choix sont possibles : il fait partie de la cargaison ou non ;
- la récursivité s'effectuera sur la liste des indices de `Pr` : le premier appel de la fonction se fera en utilisant l'indice n, puis l'indice $n-1$ et ainsi de suite jusqu'à l'indice 0 (correspondant au cas où il n'y a plus de produits) ;
- pour $i \in \{0,1,\dots,n\}$ et $\omega \in \{0,1,\dots,P_{max}\}$, on note $S(i,\omega)$ la valeur maximale cumulée des produits que l'on peut placer dans un camion d'une capacité maximale (en poids) de ω avec la liste constituée des i premiers produits de `Pr`.

On pose alors la relation de récurrence suivante :

$$S(i,\omega) = \begin{cases} 0 & \text{si } i = 0 \\ S(i-1,\omega) & \text{si } i > 0 \text{ et } p_i > \omega \\ \max(S(i-1,\omega), v_i + S(i-1,\omega-p_i)) & \text{si } i > 0 \text{ et } p_i \leqslant \omega \end{cases} \quad (3)$$

Écrire une fonction récursive `opti` ayant pour arguments les listes de poids et de valeurs P et V, un indice i, un poids ω, et renvoyant $S(i,\omega)$.

Vérifier que pour $n = 4$, $P_{max} = 8$ centaines de kilogrammes, Pr = [1, 2, 3, 4], P = [3, 2, 1, 4], V = [4, 3, 1, 9] (Par exemple ici, le produit 2 a un poids de deux centaines de kilogrammes et une valeur de trois centaines d'euros), le profit maximal est 1400 euros.

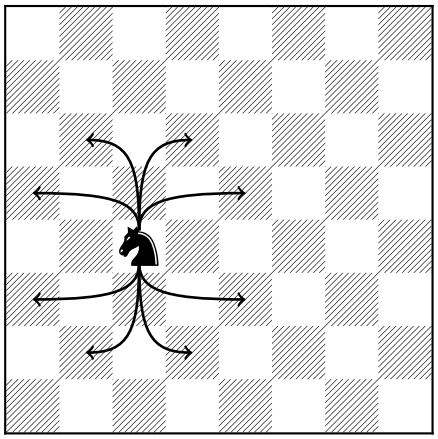
Amélioration de la fonction récursive : pour éviter les calculs redondants, si vous ne l'avez pas déjà fait, écrire une fonction récursive `opti_memo` utilisant le principe de mémoïsation à l'aide d'un dictionnaire `dico_opti` initialisé à `{}`.

Exercice 6 - exemple de backtracking

On considère un échiquier à n lignes et p colonnes. Un tel échiquier sera modélisé par un tableau `Numpy` et la case (x,y) pourra prendre plusieurs valeurs ayant différentes significations.
On dispose également d'une variable globale `Dep` qui est le tableau des 8 déplacements possibles du cavalier en bougeant de 2 cases dans une direction (verticale ou horizontale) et de 1 case perpendiculairement. Si le cavalier est loin des bords de l'échiquier alors il a 8 possibilités de déplacements, mais il en a moins s'il est près des bords.

```
import numpy as np

Dep = [(1,2),(1,-2),(-1,2),(-1,-2),(2,1),
       (2,-1),(-2,1),(-2,-1)]
```



Pour la gestion des piles, on pourra manipuler des listes avec append et pop.
On veut déterminer un chemin du cavalier passant une et une seule fois par toutes les cases de l'échiquier. On commence par remplir la grille de l'échiquier avec des 0. Lorsqu'une case est à 0, c'est qu'elle n'a pas été visitée, sinon elle contient l'instant de passage du cavalier.

1. Écrire une fonctions `cases_accessibles(E,x,y,n,p)` qui renvoie une pile des cases de l'échiquier `E` encore accessibles à partir de la position `(x,y)` (une case visitée n'est plus accessible).
2. On veut créer une fonction `parcours(x,y,n,p)` qui renvoie la matrice des instants de passage. On suit cet algorithme :
   - on initiale `E` et on fixe à 1 la case `(x,y)` ainsi que le compteur de mouvement `N`;
   - on initialise une pile `Coups` avec le triplet `(x,y,L)` où `L` est la pile des cases accessibles à partir de la position initiale;
   - tant que $N \neq n.p$ et que `Coups` est non vide, on dépile l'élément `(x,y,L)` au sommet de la pile des coups. Si `L` est vide, on a atteint un cul-de-sac et on revient d'un coup en arrière. Sinon, on empile dans `Coups` le triplet `(x,y,L')` où `L'` est la pile `L` privé de l'un de ses éléments `(xs,ys)`, puis on empile dans `Coups` le triplet `(xs,ys,Ls)` où `Ls` est la pile des cases accessibles à partir du sommet `(xs,ys)`... en faisant tout cela, on met à jour la matrice `E` ainsi que le nombre de coups `N`.

   On devrait obtenir par exemple :

```
>>>parcours(0,0,5,5)
array([[ 1, 12, 25, 18,  3],
       [22, 17,  2, 13, 24],
       [11,  8, 23,  4, 19],
       [16, 21,  6,  9, 14],
       [ 7, 10, 15, 20,  5]])
```

Pour ceux qui seraient rapides et, sinon, en guise de travail personnel, veillez à savoir programmer récursivement :

- l'obtention de l'écriture d'un entier naturel en base b;
- la recherche dichotomique dans une liste triée;
- le tri fusion ou le tri rapide.