ISM MPSI/PCSI, Informatique
Année 2025/2026

TP Informatique 12

⚠ On rappelle qu'un script (fichier `*.py`) doit être enregistré et exécuté (touche `F5`) pour que les fonctions saisies dans le script soient utilisables dans la console et que la combinaison de touches `Ctrl+C` permet de casser une boucle infinie.

Exercice 1

On s'intéresse au problème du *rendu de monnaie* : on souhaite rendre la monnaie en minimisant le nombre de pièces et billets.

Par exemple, dans le cadre du système de la zone euro, le système de monnaie, en omettant les centimes, est constitué du jeu de billets et pièces décrit par la liste

```
[500,200,100,50,20,10,5,2,1]
```

On considère dans ce qui suit qu'un système de monnaie est une liste triée par ordre décroissant se terminant par `1`, ce qui garantit de pouvoir rendre la monnaie sur tout montant entier.

On utilise l'algorithme glouton suivant : tant qu'il reste quelque chose à rendre, on choisit le plus gros billet ou pièce qu'on peut rendre.

1. Écrire une fonction `rendu(v,S)` d'argument un entier `v`, une liste `S` et qui renvoie la liste des pièces ou billets à rendre pour le montant `v` et le système de monnaie `S`.
2. Tester la fonction `rendu` avec le système de monnaie

   ```
   [400,300,100,40,30,10,4,3,1]
   ```

   Minimise-t-on le nombre de pièces et billets rendus avec l'algorithme glouton ?

Exercice 2

En algorithmique, le problème du sac à dos, parfois noté (KP) (de l'anglais Knapsack Problem) est un problème d'optimisation combinatoire. Il modélise une situation analogue au remplissage d'un sac à dos, ne pouvant supporter plus d'un certain poids, avec tout ou partie d'un ensemble donné d'objets ayant chacun un poids et une valeur. Les objets mis dans le sac à dos doivent maximiser la valeur totale, sans dépasser le poids maximum.

On modélise les objets à l'aide de couples (v_i, p_i), $1 \leqslant i \leqslant n$, où n est le nombre d'objets, v_i est la valeur de l'objet i et p_i son poids. On note P le poids total supporté par le sac à dos.
L'objectif est de maximiser la valeur totale des objets rangés dans le sac, sous la contrainte que leur poids total reste inférieur ou égal à P. Il s'agit d'un problème difficile[1], dont la résolution exacte ou approchée peut être obtenue à l'aide de nombreuses méthodes.

On s'intéresse dans cet exercice à l'application d'une stratégie gloutonne pour remplir un sac à dos. On introduit pour cela l'*efficacité* de l'objet i comme étant le rapport $e_i = \frac{v_i}{p_i}$.

1. C'est un problème NP-complet

1. Écrire une fonction `trieff(L)` prenant en argument une liste de couples représentant les objets à ranger dans le sac à dos, et renvoyant une liste de ces mêmes objets triés par efficacité décroissante.

   On «rappelle» que l'instruction `sorted(L)` *renvoie une liste contenant les éléments de* `L` *triés dans l'ordre croissant. Avec les paramètres* `key` *et* `reverse=True`*, on peut obtenir une liste triée suivant les critères de* `key` *et dans l'ordre décroissant.*
   Par exemple l'instruction `sorted(L, key = lambda couple : couple[0]/couple[1], reverse=True)` *renvoie une liste contenant les éléments de* `L` *triés dans l'ordre décroissant des valeurs des rapports du premier élément par le deuxième élément des couples de* `L`*.*

2. Écrire une fonction `remplissage(L,P)` prenant en argument `L` une liste de couples représentant les objets à ranger dans le sac à dos, et `P` le poids total supporté par le sac, et renvoyant la liste des objets à placer dans le sac à dos, leur poids total, et la valeur contenue dans le sac.

3. Tester la fonction précédente avec les objets définis par la liste de couples :

$$(1,2),(2,5),(3,7),(7,12),(10,9)$$

   Le résultat est-il optimal ?

Exercice 3

On rappelle que la suite de Fibonacci $(\mathrm{F}_n)_{n\in\mathbb{N}}$ est définie par la donnée de $\mathrm{F}_0 = 0$, $\mathrm{F}_1 = 1$ et la relation de récurrence : $\forall\, n \in \mathbb{N}$, $\mathrm{F}_{n+2} = \mathrm{F}_{n+1} + \mathrm{F}_n$.

Théorème de Zeckendorf :
Pour tout entier naturel non nul n, il existe un unique entier k et un unique k-uplet d'entiers $(c_1, \ldots, c_k)$, vérifiant :

- $c_1 \geqslant 2$
- pour tout i appartenant à $[\![1, k-1]\!]$, $c_i + 1 < c_{i+1}$

tels que :

$$n = \sum_{i=1}^{k} \mathrm{F}_{c_i}$$

Cette décomposition s'appelle la décomposition de Zeckendorf du nombre n.

Écrire une fonction `Zeckendorf(n)` prenant en argument un entier n non nul et renvoyant sa décomposition de Zeckendorf sous forme d'une liste d'entiers.