TP D'INFORMATIQUE 7
Algorithmes gloutons

1 Problème du rendu de monnaie

On s'intéresse dans cette partie au problème suivant : on cherche à rendre une certaine quantité en monnaie en utilisant le plus petit nombre de pièces, sachant qu'on dispose d'une quantité illimitée de chaque pièce.

1.1 Formalisation du problème

On appelle **système** un $m-$uplet d'entiers $s = (s_i)_{0\leq i<m}$ vérifiant : $1 = s_0 < s_1 < \cdots < s_{m-1}$, correspondant aux valeurs des pièces (ou billets) en service.
Par exemple, pour le système en zone euro, le système est : $(1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000)$ si on prend en compte billets et pièces, exprimés en centimes d'euro.
Soit $x \in \mathbb{N}$, le montant à rendre. Une **représentation** de x dans ce système s est un $m-$uplet $k = (k_i)_{0\leq i<m}$ vérifiant $x = \sum_{i=0}^{m-1} k_i s_i$. Autrement dit, k_i est le nombre de pièces de valeur s_i qui seront rendues.
Pour épargner les poches des clients, on souhaite minimiser le poids de cette représentation, et donc le nombre de pièces rendues :

$$w(k) = \sum_{i=0}^{m-1} k_i$$

Les systèmes et les représentations seront donnés sous forme de liste, `[3, 1, 4]` est par exemple une représentation de 30 dans le système `[1, 3, 6]`.

1. Écrire en Python une fonction `est_un_systeme` qui prend en argument une liste et qui renvoie `True` si c'est un système et `False` sinon.

1.2 Implémentation gloutonne

Un algorithme glouton pour résoudre un problème d'optimisation est un algorithme qui a chaque étape fait toujours le choix qui semble à court terme le plus avantageux. Ici, l'algorithme glouton pour rendre une somme $x > 0$ consiste à toujours rendre la pièce la plus grande possible. Par exemple avec le système $s = (1, 2, 5, 10, 50)$, l'algorithme rendra 27 en commençant par une pièce de 10 (car 50 dépasserait la somme à rendre), puis encore 10, puis 5 puis 2, donnant donc la représentation `[0, 1, 1, 2, 0]`.

2. Écrire une fonction `glouton_monnaie` prenant en argument la somme à rendre `x` et le système `S`, et renvoyant la représentation de `x` dans `S` déterminée par l'algorithme glouton.

3. On souhaite obtenir une version récursive de cet algorithme glouton. Comme on ne peut pas efficacement faire un appel récursif sur une sous-liste en Python (car chaque élément de cette sous-liste sera recopié depuis la liste initiale), nous allons passer par une fonction auxiliaire prenant en argument supplémentaire le nombre de pièces à prendre en compte dans le système.

   (a) Écrire une fonction `glouton_monnaie_aux` prenant en argument la somme à rendre `x`, le système `S` et un entier `i`, et renvoyant la représentation selon l'algorithme glouton de `x` dans le système `S[0:i]`, c'est-à-dire en n'utilisant que les `i` premières valeurs du système `S`. Cette fonction devra être récursive en `i`.

   (b) En déduire une fonction `glouton_monnaie_rec` donnant le même résultat que la fonction `glouton_monnaie`, mais en appelant la fonction `glouton_monnaie_aux`.

1.3 Système canonique

Un système est dit *canonique* si l'algorithme glouton donne toujours la solution optimale, c'est-à-dire le nombre de pièces minimal.

4. Avant la réforme de 1971 introduisant un système décimal, le Royaume-Uni utilisait le système $(1, 3, 6, 12, 24, 30)$. Montrer que ce système n'est pas canonique.

5. Donne un système non canonique pour $m = 3$.

6. Justifier que tout système pour $m = 2$ est canonique.

Heureusement, on peut montrer que le système monétaire de la zone euro est canonique.

2 Réservation d'une salle

La salle Médiathèque du lycée est partagée entre plusieurs classes. Chacun des n cours i pouvant s'y dérouler est caractérisé par une date de début d_i et une date de fin f_i. On souhaite que le maximum de cours ait lieu dans la salle, deux cours ne pouvant avoir lieu en même temps, leurs intervalles de temps devant donc être disjoints.
Ainsi, on dit qu'un ensemble $J \subset [\![0, n-1]\!]$ de cours forme une **réservation valide** si

$$\forall (i,j) \in J^2, i \neq j \;\Rightarrow (\; d_i \geq f_j \quad \text{ou} \quad f_i \leq d_j)$$

et le problème est alors de trouver une réservation valide de plus grand cardinal possible.

Il serait possible de résoudre ce problème par force brute : pour chaque partie J de $[\![0, n-1]\!]$, on peut tester si elle correspond à une réservation valide, et faire une recherche de maximum sur le cardinal des réservations valides ainsi parcourues. Le problème de cette approche est son coût en temps : il y a 2^n parties, donc les considérer une par une engendrerait une complexité exponentielle, et en pratique un algorithme extrêmement lent quand n devient grand.

Pour cette raison, nous cherchons plutôt à résoudre ce problème à l'aide d'un algorithme glouton. Il faut donc choisir un critère sur lequel se baser pour sélectionner à chaque étape le cours à ajouter à la réservation. Le choix du bon critère peut faire la différence entre un algorithme glouton optimal ou non.

2.1 Choix du critère glouton

On considère d'abord les 3 critères suivants :

- la durée du cours ;
- la date de début du cours ;
- le nombre d'intersections du cours avec un autre cours.

Pour chacun de ces critères, on obtient une stratégie gloutonne en triant les cours dans l'ordre croissant de ce critère, puis à chaque étape en choisissant d'ajouter à la réservation le premier cours compatible avec les cours déjà sélectionnés.

7. Pour chacune de ces stratégies, montrer par un exemple qu'elle ne donne pas forcément une solution optimale.

2.2 Implémentation

On considère à présent le critère de la date de fin du cours, dont on admet qu'il permet à l'algorithme glouton d'être optimal.

On veut alors implémenter cet algorithme en Python. On représente l'entrée du problème par une liste dont chaque élément est un triplet composé de deux entiers correspondant aux dates de début et de fin du cours, et d'une chaîne de caractère qui identifie la classe, et on veut obtenir en sortie une liste de même format contenant les cours sélectionnés dans la réservation.

8. L'instruction `list.sort(L, key=f)` permet de trier la liste `L` selon la fonction `f`, de façon à ce qu'après exécution de cette instruction, deux éléments consécutifs a et b de L vérifieront $f(a) \leq f(b)$
   Écrire la fonction `f` correspondant au critère donnant l'algorithme optimal.
9. Écrire une fonction `reservation_glouton` implémentant l'algorithme glouton optimal.
10. Sachant qu'un tri peut se faire en $(n \log n)$ opérations, estimer la complexité de l'algorithme glouton.
11. Démontrer que le critère des dates de fin de cours donne toujours une solution optimale.
12. Implémenter l'algorithme par force brute et vérifier qu'il donne des solutions de même cardinal que l'algorithme glouton optimal.