Algorithmique
TP1 : Greedy

Bonus :

l'exercice 1 de cette série (problème du rendu de la monnaie) fait l'objet d'un bonus. Veuillez suivre attentivement les spécifications de l'énoncé. Ceci est un travail **individuel**.

Pour obtenir les points de bonus, vous devrez, pour le dimanche 29 septembtre 23h59 au plus tard, rendre les fichiers suivants (avec votre nom et votre prénom à la place de 'NomPrenom') :

- Un fichier pdf (*NomPrenom.pdf*) répondant à toutes les questions de l'exercice.
- Un fichier python (*NomPrenom.py*) de votre implémentation de l'algorithme. N'envoyez pas le bytecode (fichier .pyc).

Les fichiers doivent être soumis sur Moodle dans le devoir 'Bonus 1'.

Le rapport doit être dactylographié au format pdf avec votre nom sur la première page.

Le script doit utiliser un version 3.x de Python et respecter strictement les spécifications pour l'input et l'output pour être pris en compte.

1 Rendu de monnaie britannique (Bonus, 7pts)

Avant la décimalisation, la monnaie britannique contenait les pièces suivantes :

- la *demi-couronne* d'une valeur de 30 pence ;
- le *florin* d'une valeur de 24 pence ;
- le *shilling* d'une valeur de 12 pence ;
- le *sixpence* d'une valeur de 6 pence ;
- le *threepence* d'une valeur de 3 pence ;
- le *penny*.

Dans cet exercice, on ne prendra pas en compte le demi penny (1/2 pence) ni le farthing (1/4 pence). Notez que 'pence' est le pluriel de 'penny'.

1. *(2 points)* Donnez un pseudocode correspondant à l'algorithme Greedy *vu en cours* pour résoudre ce problème.

2. *(1 point)* Pourquoi peut-on dire que c'est un algorithme de type Greedy ?
3. *(1 point)* Cet algorithme est-il toujours optimal ? Si oui, prouvez-le, sinon donnez un contre exemple.
4. *(3 points)* Implémentez cet algorithme en Python au travers d'une fonction `compute_change(money, coin_set)`, où `money` est la monnaie totale qui doit être changée et `coin_set` une simple liste ordonnée contenant la valeur des pièces disponibles pour le change. Cette fonction doit retourner une liste contenant, dans l'ordre, les pièces changées contre la somme initiale. Essayez de garder le code aussi simple que possible.

   Notez bien que votre code doit impérativement respecter le format de la fonction `compute_change` tel que décrit ci-dessus. Voici un exemple d'utilisation typique de votre script (tel qu'il sera testé au moment de l'évaluation), que nous nommons pour l'exemple "nomPrenom.py" :

```
from nomPrenom import compute_change
money = 100
coin_set = [30, 24, 12, 6, 3, 1]
change = compute_change(money, coin_set)
if change == [30, 30, 30, 6, 3, 1]:
    print("Correct answer")
else:
    print("Wrong answer")
```

   N.B : Le nom des pièces peut être ignoré dans cet exercice.

2 Espace disque

On dispose de n fichiers appelés $f_1, f_2, ..., f_n$ de taille s_i qu'on souhaite sauvegarder sur un disque de taille D, avec :

$$\sum_{i=1}^{n} s_i > D.$$

1. Supposons que l'on veuille remplir le plus de fichiers possible dans ce disque. Donner un algorithme glouton résolvant ce problème. Donne-t-il une solution optimale (du point de vue du nombre de fichiers, pas des performances !) ?
2. On souhaite maintenant réduire l'espace disque inutilisé. On utilise un algorithme glouton naïf qui prend toujours le plus grand fichier possible. Cela fonctionne-t-il ?

3 La traversée du désert

Une personne souhaite traverser un désert en emportant avec lui une bouteille d'eau. Sachant qu'il peut parcourir une distance d avant de devoir remplir sa bouteille de nouveau , qu'il part du point x_0, qu'il arrive au point x_n et qu'il y a des points d'eau aux points x_i tq $0 < i < n$

1. Quelle est la condition pour que ce problème ait une solution ?
2. Donnez - si elle existe - une solution avec $d = 10$, $x_0 = 0$, $x_n = 50$ et $X = \{5, 10, 17, 21, 37, 45\}$
3. Donnez un algorithme trouvant une solution pour tous les cas ou il y en a une.
4. Donnez une solution avec $d = 10$, $x_0 = 0$, $x_n = 22$ et $X = \{5, 10, 12, 20\}$
5. Cet algorithme est-il optimal ? Si oui prouvez-le, sinon trouvez un contre exemple.