TP 3 – Algorithme glouton

Un algorithme est dit « glouton » lorsqu'il effectue, à chaque étape, le choix qui semble le meilleur à ce moment-là et qui ne revient pas sur sa décision. Ils s'utilisent souvent pour des problèmes d'optimisation, c'est à dire où on cherche à maximiser quelque chose avec des contraintes.

Nous allons illustrer ce type d'algorithme avec quelques exemples.

Question 1
Écrire une fonction pour calculer la somme des éléments d'une liste.

Question 2
Écrire une fonction pour déterminer si une liste est triée dans l'ordre décroissant.

Question 3
Écrire une fonction pour ranger une liste dans l'ordre décroissant.
*On pourra se souvenir du tri à bulle du **TP2** et utiliser la fonction réalisée à la **question 2**.*

Question 4 : Rendu de monnaie (*premier algorithme glouton*)

On souhaite réaliser une fonction `rendre_monnaie` qui prend en variables :

- un montant à rendre
- la liste des valeurs des pièces ou billets disponibles

et qui renvoie une liste de valeurs des pièces ou billets à rendre.

***Par exemple** :*
`rendre_monnaie(6, [1, 2, 4])` peut renvoyer `[4, 2]` ;
`rendre_monnaie(5, [1, 7, 3])` peut renvoyer `[3, 1, 1]`.

***Instructions qui peuvent s'avérer utiles** :*
L'instruction `liste.append(ajout)` ajoute l'élément `ajout` à la fin de la liste `liste`.
L'instruction `liste=[]` créé une liste vide appelée `liste`.

Est-on assuré que cet algorithme rend le minimum de pièces, i.e. cet algorithme est-il optimal ?

Question 5 : Emploi du temps de salles (*deuxième algorithme glouton*)

1. On cherche à gérer l'emploi du temps d'une salle de cours. Les créneaux des cours sont identifiés par un couple de réels $(d, f) \in [0, 24[^2$ où d est l'heure de début et f l'heure de fin.
Écrire une fonction `intersecte` pour déterminer si deux cours se chevauchent.

Par exemple :
`intersecte((1, 3), (2, 4))` doit renvoyer `True`
mais `intersecte((1, 2), (3, 5))` doit renvoyer `False`.

2. Écrire une fonction `intersecte_list` telle que, si c est un cours et L est une liste de cours, `intersecte_list`(c, L) renvoie True si c chevauche un des cours de L.

Par exemple : `intersecte_list((1, 3), [(2, 5), (4, 6)])` renvoie True
mais `intersecte_list((1, 3), [(4, 6), (5, 7)])` renvoie False.

3. Écrire une fonction `choisir_salle` telle que, si c est un cours, `choisir_salle`(c, salles) renvoie une salle disponible pour c, ou -1 si aucune salle n'est disponible.

Par exemple :
`choisir_salle((7, 10), [[(9, 11)], [(12, 15)]])` doit renvoyer 1
mais `choisir_salle((7, 10), [[(8, 10)], [(9, 12)]])` doit renvoyer -1.

4. Écrire une fonction `allocation_salles(cours)` qui renvoie une liste des salles utilisées par les cours de la liste cours en appliquant l'algorithme glouton.
On pourra compléter le code suivant :

```
def allocation_salles(cours):
    cours.sort(key=lambda c: c[0]) # trie les cours par ordre
    de dÃ©but croissant
    salles = []
    for c in cours: # parcours de la liste cours
        for s in range(len(salles)): # teste si on peut
   mettre le cours c en salle s
            if ... # si on peut mettre le cours c en salle s
                salles[s].append(c) # on ajoute le cours c Ã
    la liste des cours de la salle s
                break # on sort de la boucle for

    return salles
```

Listing 1 – Question 5.4

5. Écrire les trois fonctions suivantes :
- ranger une liste de cours par ordre croissant de leur début
- à partir d'une liste de cours rangés par ordre croissant de leur début, faire une affectation d'une salle sans chevauchement de cours
- à partir d'une liste de cours, affecter autant de salles que nécessaire pour caser tous les cours dans des salles.