TP14

ALGORITHMES GLOUTONS

Lorsque l'on conçoit un algorithme pour résoudre un problème, il faut choisir une stratégie. Parmi les différentes stratégies, il existe la stratégie gloutonne que l'on va présenter ici.
Une **stratégie gloutonne** consiste à opérer le meilleur choix à chaque étape de la résolution du problème, en espérant obtenir une solution globale optimisée.

Exercice 1 Problème du sac à dos

On dispose d'un sac à dos, supportant au maximum 30 kg, et d'objets de différentes valeurs :

Objet	0	1	2	3
Valeur (euros)	7	3	4	4
Poids (kg)	13	7	11	12
Valeur/poids	0.54	0.43	0.36	0.33

Il s'agit d'analyser le programme suivant, **sans l'exécuter**.

```
objets = [[7,13], [3,7], [4,11], [4,12]]
n = len(objets)

poids_max = 30
sac = []
poids = 0
valeur = 0

for i in range(n):
    poids_objet = objets[i][1]
    if poids + poids_objet <= poids_max:
        sac.append(objets[i])
        poids = poids + poids_objet
        valeur = valeur + objets[i][0]

print('sac =',sac, 'poids =', poids, 'valeur =', valeur)
```

1. Faire un tableau des valeurs successives de i, poids_objet, poids, valeur et sac.

2. Que fait ce programme ?

3. L'ordre des objets a-t-elle une incidence sur le résultat ?

4. On cherche à avoir la plus grande valeur possible dans le sac.

   (a) La solution obtenue est-elle optimale ?

   (b) Expliquer pourquoi l'ordre choisi ici est tout de même pertinent.

> Un algorithme glouton offre un bon compromis entre la qualité de la solution et le temps de calcul mais ne fournit pas forcément une solution optimale.

Exercice 2 Rendu de monnaie

La personne en charge de la caisse d'un magasin doit rendre régulièrement de la monnaie aux clients qui payent en espèces. On va ici écrire un programme pour l'aider et lui dire précisément quelles pièces elle doit donner au client selon la somme à rendre.

On suppose que :

- la caisse dispose de pièces (ou billets) de valeurs entières $v_1 < v_2 < \cdots < v_n$ avec $v_1 = 1$ euro ;
- la quantité de monnaie disponible est très grande, elle ne sera pas à court ;
- la somme à rendre au client est une valeur entière notée s ;

Pour résoudre ce problème, on va utiliser un *algorithme glouton* : cela consiste à construire une solution globale en avançant pas à pas selon un principe qui est localement le plus optimal. Ici, cela veut dire qu'on commence par rembourser avec la pièce la plus grande possible.

Exemple : $s = 47$ euros à rendre.
Monnaie disponible : $v_1 = 1$, $v_2 = 2$, $v_3 = 5$, $v_4 = 10$, $v_5 = 20$.

- la plus grande valeur inférieure à 47 est 20 : on rend 1 billet de 20 euros ;
- il reste 27 euros : on rend donc 20 euros ;
- il reste 7 euros : on rend donc 5 euros ;
- il reste 2 euros : on rend donc 2 euros.

On va donc rendre : 2 billets de 20 euros, 1 billet de 5 euros et 1 pièce de 2 euros.

1. Compléter le programme suivant :

```
s = 47                    # somme à rendre
monnaie = [20, 10, 5, 2, 1] # monnaie par ordre décroissant

rendu = [] # liste des pièces à rendre

# D'où démarre t-on ?
i = .................  # indice dans la liste monnaie

while ........................ :

    # on cherche la plus grande pièce <= s

    piece = monnaie[i]









print(rendu)
```

2. Vérifier que l'on obtient bien le résultat attendu, puis tester avec d'autres valeurs de s.
3. Changeons maintenant la liste des pièces pour `monnaie = [4,3,1]`.
   (a) Déterminer, en utilisant l'algorithme glouton, quelles pièces rendre pour une somme de `s = 10` euros.

   (b) Si on veut minimiser le nombre de pièces rendues, cette solution est-elle optimale ?

   (c) Que se passe-t-il si on enlève la pièce de 1 euro ?