

## TP2 - Le problème du rendu de monnaie

### I Retour sur les Tris

1. Ecrire une fonction **extraire** qui accepte en paramètres, une liste de liste **l** et un entier **n** dans  $[0; 2]$ . La fonction doit retourner la liste triée en fonction du  $n$ -ième élément de chaque sous-liste. On pourra choisir le tri insertion par exemple.

#### Console Python

```
>>> L= [ [1, 5, 6], [8, 10, 2], [3, 3, 5], [4, 8, 1] ]
>>> extraire(L,0)
[[1, 5, 6], [3, 3, 5], [4, 8, 1], [8, 10, 2]]
>>> extraire(L,1)
[[3, 3, 5], [1, 5, 6], [4, 8, 1], [8, 10, 2]]
>>> extraire(L,2)
[[4, 8, 1], [8, 10, 2], [3, 3, 5], [1, 5, 6]]
```

2. Pourquoi le code précédent ne fonctionne-t-il pas (tel quel) avec une liste de tuples?
3. A l'aide de l'exercice 7, expliquer le sens de l'instruction `List.sort(key = lambda a : a[1])`

### II Algorithme du sac à dos

#### A Critère de Poids

Nous allons écrire un premier algorithme glouton de critère "**placer d'abord dans le sac, les objets les plus lourds.**" En cas d'égalité, on prendra "au hasard" un objet parmi les indécis. On pourra utiliser la méthode `list.sort()` comme définie sur la documentation officielle Python

1. Ecrire une fonction **gloutonP** de paramètres **l** et **maxi**, respectivement une liste de tuples (valeur en €, poids en kg) et un entier correspondant au poids maximal supporté par le sac. La fonction doit retourner un triplet **reponse, valeur, poids** comme dans l'exemple ci-dessous :

#### Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonP(liste,30)
([1, 1, 0, 0], 11, 25)
```

Le fichier **Datas.txt** contient un ensemble de données correspondant à 5 magasins fictifs : t4, t7, t8, t10 et t15. Chaque ligne du fichier contient, le nom du magasin, une liste d'objets et le poids maximal autorisé dans le sac du voleur.

2. A l'aide de la fonction **perf\_counter** du module **time**, calculer la durée d'exécution sur les ensembles de données fournis dans le fichier Datas.txt. Les durées seront obtenues en réalisant une moyenne sur 100 appels de la fonction **gloutonP**. Noter vos résultats dans les tableaux en annexe A (avec 4 chiffres significatifs sur le temps).

#### B Critère de Valeur

Pour notre deuxième algorithme glouton, le critère retenu est "**placer d'abord dans le sac, les objets de plus grande valeur**". Ecrire une fonction **gloutonV** qui accepte les mêmes paramètres que `gloutonP` et qui retourne le triplet **reponse, valeur, poids** comme dans l'exemple ci-après; réaliser ensuite les mêmes tests à l'aide du fichier **datas.txt** pour compléter les tableaux de l'annexe A.

## Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonV(liste,30)
([1, 1, 0, 0], 11, 25)
```

## C Critère de Valeur Pondérée par le Poids

Pour notre dernier algorithme glouton, le critère retenu est "**placer d'abord dans le sac, les objets dont le rapport  $\frac{\text{valeur}}{\text{poids}}$  est le plus élevé.** Ecrire une fonction **gloutonVPP** qui accepte les mêmes paramètres que gloutonP et retourne le triplet **reponse, valeur, poids** comme dans l'exemple ci-après; réaliser ensuite les mêmes tests à l'aide du fichier **datas.txt** pour compléter les tableaux en annexe A.

## Console Python

```
>>> liste=[(7,13), (4,12), (3,8), (3,10)]
>>> gloutonVPP(liste,30)
([1, 1, 0, 0], 11, 25)
```

## D Comparaison des trois critères

1. Comparer les durées d'exécution des trois algorithmes gloutons. Commenter.
2. Discuter du choix du critère de sélection des algorithmes gloutons en fonction de  $n$ .

### Annexe A : comparaison des performances des différents algorithmes

| Durée d'exécution ( en s)* de différents algorithmes en fonction du nombre d'objets pour le KP |          |          |          |          |       |
|--|----------|----------|----------|----------|-------|
| Algorithme   | n=4      | n=7      | n=8      | n=10     | n=15  |
| GLOUTONP   |          |          |          |          |       |
| GLOUTONV   |          |          |          |          |       |
| GLOUTONVPP   |          |          |          |          |       |
| Algorithme Naïf  | 1.291e-4 | 2.129e-3 | 4.348e-3 | 2.100e-2 | 1.137 |

\* Calculs effectués sur 1000 itérations pour chaque fonction; MacBook Pro IntelCore i5 2.4 GHz double coeur

| Réponses** des différents algorithmes en fonction du nombre d'objets pour le KP |                        |                                       |   |   |   |
|---|------------------------|---------------------------------------|---|---|---|
| Algorithme  | n=4                    | n=7                                   | n=8                                     | n=10  | n=15  |
| GLOUTONP  |                        |                                       |   |   |   |
| GLOUTONV  |                        |                                       |   |   |   |
| GLOUTONVPP  |                        |                                       |   |   |   |
| Sol. Optimale   | ([1,1,0,0],<br>11, 25) | ([0, 1, 0, 1, 0, 0, 1],<br>1735, 169) | ([1, 1, 1, 1, 0, 1, 0, 0],<br>280, 102) | ([1, 1, 1, 1, 0, 1, 0, 0, 0,<br>0], 309, 165) | ([0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0,<br>1, 1, 0], 1458, 749) |

Pour aller plus loin ... : exercice 8 de la feuille d'exercices