TD informatique du chapitre 9 : Algorithmes gloutons

Exercice 1 : Problème du choix d'activités

Dans cet exercice, on va programmer les différents algorithmes gloutons vus en cours.
On utilisera la méthode sort afin de trier des listes de listes. Si l est une liste, alors

```
l.sort(key=lambda x :f(x))
```

modifie l en la triant selon les valeurs croissantes de la fonctions f. Par exemple :

```
>>> l=[[0,2],[1,0],[3,1]]
>>> l.sort(key=lambda x :x[0])
>>> l
[[0, 2], [1, 0], [3, 1]]
>>> l.sort(key=lambda x :x[1])
>>> l
[[1, 0], [3, 1], [0, 2]]
>>> l.sort(key=lambda x : x[0]+x[1])
>>> l
[[1, 0], [0, 2], [3, 1]]
```

1. Ecrire une fonction Duree prenant comme argument une liste de couples représentant un planning d'activités et renvoyant une liste de couples représentants les activités obtenues en appliquant un algorithme glouton de tri par durée.
2. (a) Ecrire une fonction NbIncompatible prenant comme argument une liste de couples représentant un planning d'activités et renvoyant une liste de triplets représentants le planning avec pour chaque triplet : l'heure de début, l'heure de fin et le nombre d'incompatibilités.
   (b) Ecrire une fonction Incompatible prenant comme argument une liste de couples représentant un planning d'activités et renvoyant une liste de couples représentants les activités obtenues en appliquant un algorithme glouton de tri par nombre d'incompatibilités.
3. Ecrire une fonction Debut prenant comme argument une liste de couples représentant un planning d'activités et renvoyant une liste de couples représentants les activités obtenues en appliquant un algorithme glouton de tri par date de début.
4. Ecrire une fonction Fin prenant comme argument une liste de couples représentant un planning d'activités et renvoyant une liste de couples représentants les activités obtenues en appliquant un algorithme glouton de tri par date de fin.

Exercice 2 : Le sac à dos

On dispose d'un sac à dos et de n objets dont la masse est connue. Afin de ne pas casser le sac à dos, une masse maximale est imposée. Chaque objet a une valeur donnée. On souhaite remplir le sac à dos avec des objets qui permettent de maximiser la valeur totale.
Par exemple, on a un sac à dos dont la capacité maximale est de 5kg et les objets suivants :

- un livre pesant 1kg de valeur 10 €,
- un paquet de biscuits pesant 1kg de valeur 3 €,
- une bouteille d'eau pesant 2kg de valeur 1 €,
- un ordinateur pesant 4kg de valeur 1000 €.

On choisit de prendre le livre et l'ordinateur afin d'avoir la valeur de 1010 €, avec une masse de 5kg.
On représentera les objets par une liste objets du type $[masse, valeur]$, ainsi pour l'exemple précédent :

```
objets=[[1,10],[1,3],[2,1],[4,1000]].
```

1. Dans cette question, on va tester différents algorithmes gloutons. On utilisera l'exemple suivant :

   ```
   objets=[[12,27],[9,19],[8,16],[7,15],[6,13],[5,11],[4,10],[3,6],[2,5],[1,2]] et max=40.
   ```

   (a) Ecrire une fonction Poids prenant comme arguments la liste objets et la masse maximale max qui, en utilisant un algorithme glouton plaçant les objets les plus légers en premier, renvoie la liste des objets choisis et la valeur totale du sac à dos.
   (b) Ecrire une fonction Valeur prenant comme arguments la liste objets et la masse maximale max qui, en utilisant un algorithme glouton plaçant les objets les plus chers en premier, renvoie la liste des objets choisis et la valeur totale du sac à dos.
   (c) Ecrire une fonction rapport prenant comme arguments la liste objets et la masse maximale max qui, en utilisant un algorithme glouton plaçant les objets ayant le meilleur rapport valeur/poids en premier, renvoie la liste des objets choisis et la valeur totale du sac à dos.
2. Aucune des solutions proposées ne donne un algorithme glouton exact. Pour le vérifier nous allons écrire toutes les solutions possibles et les comparer.
   (a) Pourquoi cette méthode est une mauvaise idée, alors qu'elle fonctionne?

(b) Ecrire une fonction `binaire` qui prend comme arguments deux entiers `n` et `b` et qui renvoie la liste de l'écriture binaire de `n` sur `b` bits (avec $n < 2^b$).
En utilisant cette fonction pour $0 \leq n < 2^b$, on obtient toutes les listes de b éléments à valeurs dans $\{0,1\}$.
En considérant que la valeur 0 correspond à un élément qui n'est pas dans le sac à dos et la valeur 1 à un élément qui est dans le sac à dos, on peut ainsi considérer toutes les compositions possibles du sac à dos.

(c) Ecrire une fonction `Brut` prenant comme arguments la liste `objets` et la masse maximale `max` qui renvoie la liste des objets choisis et la valeur totale du sac à dos.

(d) Conclure.