ITC – TD n°9

Le problème du sac à dos

On s'intéresse ici à un problème classique d'algorithmique : le problème du sac à dos. La question est la suivante : disposant d'un sac à dos de contenance limitée, et d'un ensemble d'objets ayant un certain poids et une certaine valeur, comment remplir le sac à dos sans dépasser sa contenance et en maximisant la valeur emportée ? Il s'agit d'un problème dit NP complet, c'est-à-dire qu'il n'existe pas d'algorithme de complexité polynomiale trouvant la solution optimale au problème.

L'ensemble des objets sera stocké dans une liste `objets` de tuples (a_i, p_i, v_i) contenant le nom, le poids et la valeur de chaque objet. On peut accéder à une de ces caractéristiques de l'objet i par `objets[i][INOM]`, `objets[i][IPOIDS]` et `objets[i][IVAL]`.

Ici, nous allons nous intéresser à trois approches pour ce problème :

- l'algorithme d'exploration systématique, très coûteux en temps de calcul
- l'algorithme glouton, qui donne très rapidement une solution raisonablement correcte
- l'algorithme en programmation dynamique, qui exploite les spécificités du problème

pour accélérer l'exploration en ne testant pas certaines options inutiles

Sépcification

Toutes ces fonctions `exhaustif` , `glouton` et `dynamique` prendront en paramètre une liste d'objets $[(a_i, p_i, v_i)]$ (de taille N) et un entier c représentant la contenance du sac, et renverront une paire contenant la valeur totale du sous-ensemble d'objets choisis et une liste contenant le sous-ensemble en question.

On manipulera beaucoup de listes de listes ou listes de tuples dans ce TD. La fonction `deepcopy` est importée de façon à faire des copies efficaces et bien séparées des listes originales.

I. Deux fonctions utilitaires

1 – Coder une fonction `calculerValeur` qui prend en paramètre une liste d'objets et calcule la valeur totale de cette liste d'objets.

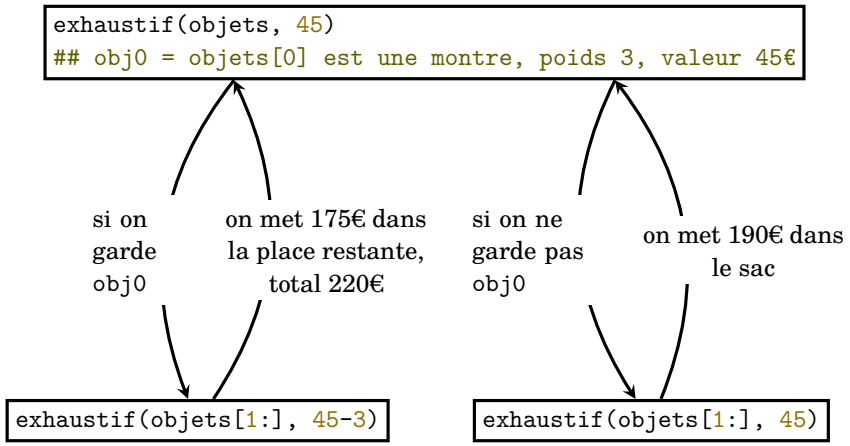
2 – Coder une fonction `calculerPoids` qui prend en paramètre une liste d'objets et calcule le poids total de cette liste d'objets.

II. Exploration exhaustive

On va explorer toutes les configurations possibles à l'aide d'un algorithme récursif testant toutes les possibilités. L'idée (inspirée de la génération exhaustive des permutations d'une liste) est la suivante :

- si la liste d'objets proposés est vide, on renvoie la liste vide avec une valeur de zéro ;
- sinon, on compare la valeur du meilleur remplissage sans le premier objet (mais avec tous les autres objets) avec la valeur du meilleur remplissage avec le premier objet imposé dans le sac (et potentiellement les autres dans la place restante), et on renvoie la meilleure des deux configurations.

Il est à noter que tester une configuration contenant un objet imposé de poids p dans un sac de contenance c revient à tester une configuration dans un sac vide de contenance $c - p$. Cela donne, par exemple, le cas suivant :



renverra un sac à dos où on a choisi de garder l'objet en position 0.

3 – Écrire la fonction `exhaustif` correspondante qui renvoie la solution au problème.