

## 1 Principe

On appelle **algorithme glouton** un algorithme **produisant une solution pas à pas**, en faisant à chaque étape un choix qui optimise un **critère local**, dans l'espoir d'obtenir une **optimisation globale**.

Cependant, les algorithmes gloutons ne fournissent pas toujours une solution optimale.

Il existe tout de même des problèmes où ces algorithmes sont mis en œuvre car ils sont généralement assez simples à développer, et la solution exacte au problème peut être très compliquée à obtenir par un autre moyen.

On espère dans ces cas que les solutions apportées par les algorithmes restent suffisamment intéressantes.

## 2 Exemples

### 2.1 Le problème du rendu de monnaie

Le problème du rendu de monnaie consiste à rendre une somme d'argent donnée avec un minimum de pièces (critère global).

Un algorithme glouton propose de répondre à ce problème en le décomposant étape par étape. À chaque étape la pièce rendue est celle de plus grande valeur possible afin de minimiser la somme restant à rendre (critère local).

Cet algorithme optimise le nombre total de pièces rendues (critère global) si le système de pièces existantes est bien choisi (ex : 1, 2, 5 10). Mais ce n'est pas le cas pour n'importe quel système de pièces (ex : 1, 3, 4, 10).

Exemple d'implémentation :

```
def rendu_monnaie(somme_a_rendre, jeu_pieces):  
    """ Renvoie la liste des pièces à rendre pour atteindre la somme à rendre.  
    - Préconditions :  
        - somme_a_rendre est un nombre (avec 2 décimales maximum)  
        - jeu_pieces est une liste de nombres (pièces disponibles)  
    - Postcondition : pieces_rendues est une liste d'entiers  
        (les valeurs des pièces rendues)      """  
    pieces_rendues = []  
    i = 0  
    while somme_a_rendre > 0:  
        while jeu_pieces[i] > somme_a_rendre:  
            i += 1  
        piece = jeu_pieces[i]  
        pieces_rendues.append(piece)  
        somme_a_rendre = somme_a_rendre - piece  
    return pieces_rendues
```

## 2.2 Le problème du sac à dos

Le problème du sac à dos consiste à choisir parmi une collection d'objets de valeurs différentes et de poids différents lesquels choisir pour remplir son sac à dos, en respectant une contrainte : le sac à dos ne peut supporter qu'une charge maximum limitée.

Le critère d'optimisation global est d'obtenir le sac à dos de plus grande valeur.

Un algorithme glouton peut essayer de répondre à ce problème pas à pas. À chaque étape on choisit un objet selon un critère local fixé ; par exemple choisir l'objet dont le rapport valeur/poids est maximum, ou plus simplement l'objet de valeur maximale.

Quelques tests permettent de se rendre compte qu'aucun de ces critères locaux n'assure d'obtenir une solution optimale globale, mais on peut tout de même espérer qu'ils fournissent une solution suffisamment intéressante dans la plupart des cas.

Exemple d'implémentation :

```
def clef_tri(objet):  
    """ Renvoie la clef de tri valeur/poids de l'objet.  
    - Précondition : objet est un tuple (valeur, poids)  
    - Postcondition : renvoie un nombre qui est le rapport valeur/poids  
    """  
    (valeur, poids) = objet  
    return valeur/poids  
  
def sac_a_dos(objets, poids_maxi):  
    """ Renvoie la liste des objets retenus.  
    - Préconditions :  
        - objets est une liste d'objets à choisir  
          (chaque objet est une tuple (valeur, poids))  
        - poids_maxi est un nombre  
    - Postcondition : la valeur renvoyée est la liste d'objets retenus  
    """  
    objets_retenus = []  
    objets.sort(key=clef_tri, reverse=True) # tri de la liste d'objets  
    for objet in objets:  
        (valeur, poids) = objet  
        if poids < poids_maxi:  
            objets_retenus.append(objet)  
            poids_maxi = poids_maxi - poids  
    return objets_retenus
```