III - Algorithmes gloutons

a) Le principe de ce type d'algorithmes

Dans cette partie, on s'intéresse à des problème d'optimisation c'est-à-dire trouver la meilleure solution à un problème parmi toutes les solutions possibles. Sur un nombre assez faible de données, il est souvent possible d'énumérer toutes les solutions et ainsi trouver de manière certaine la meilleure d'entre elles.
Cependant avec des données plus importantes, cette méthode ne sera pas efficace car trop longue. Il sera alors possible d'utiliser ce qu'on appelle **un algorithme glouton** (*Greedy Algorithm*, greedy signifiant « vorace »).

Définition : Cette méthode offre une solution rapide afin d'optimiser la résolution de problèmes comportant un très grand nombre de solutions.
Cependant la solution trouvée n'est pas forcément optimale.

Cette méthode construit une solution pas à pas. À chaque étape, elle prend la direction la plus prometteuse.

Un premier exemple :
Les habitants d'un village décident de placer des bornes wifi d'une portée de 500 m pour assurer la couverture Internet des 14 maisons.
Ils doivent placer chaque borne Wifi sur une maison et en nombre minimum pour limiter les coûts.
Représentation du village

1. Placez des bornes Wifi sur la figure en appliquant un algorithme glouton que vous détaillerez par écrit.

2. Cet algorithme glouton est-il optimal ?

b) Problème du rendu de monnaie

Le problème : Un distributeur doit rendre la monnaie et on souhaite le programmer pour qu'il rende le moins de pièces (ou billets) possible.

Nous allons nous limiter à des valeurs entières, mais cela fonctionnerait également avec les centimes.
Donc nous allons nous limiter aux valeurs 1 ; 2 ; 5 ; 10 ; 20 ; 50 et il faut que, pour toute somme comprise entre 0 et 100, le distributeur puisse déterminer le nombre de pièces (ou billets) de chaque sorte à rendre pour en rendre le moins possible.
Nous allons appliquer un algorithme glouton pour répondre à ce problème.

Le principe : on rend des pièces (ou billets) une à une en faisant ce qui parait être le meilleur choix possible à chaque instant (c'est le principe de la méthode gloutonne).
On choisit alors de faire décroitre le plus vite possible la somme à rendre en rendant à chaque fois la plus grande pièce (ou billet) possible.

Par exemple, si on doit rendre 14 €, alors on rend tout d'abord 10 €, puis comme il reste 4 €, on rend 2 € puis 2 €.
On obtient ainsi 1 billet de 10 € et 2 pièces de 2 €.

Nous allons programmer en python une fonction renvoyant sous la forme d'un dictionnaire les pièces à rendre (ce seront les clés du dictionnaire) et leur nombre (ce seront les valeurs).
Pour prendre la décision, il faudra fournir deux arguments à la fonction :

- la liste des pièces dont on dispose ;

- la somme à rendre.

```python
def renduMonnaie(pieces, somme):
    """pieces est la liste des pièces dont on dispose (on ordonnera ces pièces dans
    l'ordre décroissant au cours du programme)
    somme est le montant qui doit être rendu
    Cette fonction renvoie un dictionnaire où les clés seront les pièces àrendre et les
    valeurs le nombre de chacune de ces pièces"""
    #sommeARendre sera la valeur qu'il restera à rendre au fur et à mesure qu'on aura
    #choisi des pièces
    sommeARendre=somme
    listePiece = sorted(pieces,reverse=True) #on classe les pièces de la plus grande à
    #la plus petite
    dictPieces = {}    #accueillera le dictionnaire à renvoyer
    numPiece = 0  #position de la pièce qu'on cherche à rendre (la plus grande possible)
    while sommeARendre>0:
        if listePiece[numPiece]<=sommeARendre :
            #donc on rend la pièce d'index numPiece
            #on crée une nouvelle clé du dictionnaire si on n'a pas encore rendu cette pièce
            #sinon, on ajoute 1 à la valeur associée à cette clé
            if listePiece[numPiece] in dictPieces:
                dictPieces[listePiece[numPiece]] += 1
            else :
                dictPieces[listePiece[numPiece]] = 1
            #on enlève la valeur de cette pièce à la somme à rendre
            sommeARendre -= listePiece[numPiece]
        else :
            #on regarde la pièce suivante
            numPiece +=1
    return dictPieces
```

Le système monétaire que nous avons est un système pour lequel l'algorithme glouton donne un rendu optimal (la réponse obtenue est toujours la meilleure).

Ce n'est plus le cas avec un système moins académique.
Prenons l'exemple d'un système ne contenant que les pièces 1 ; 6 et 10 et pour lequel on devait rendre 18 €.

1. Trouvez la meilleure solution possible permettant de rendre un minimum de pièces.
2. Quel résultat est renvoyé par l'algorithme glouton ?

Solution :

Remarque : on pourrait trouver un système monétaire pour lequel l'algorithme glouton ne fonctionnerait pas : comment rendre 7 avec des pièces de 2 et de 3.
Nous savons bien sûr qu'il suffit de rendre une pièce de 3 et deux pièces de 2, mais comme l'algorithme glouton rend toujours la pièce avec la plus grande valeur, il va rendre deux pièces de 3 et il ne pourra pas rendre la valeur 1 restant. Avec ce système, notre algorithme ne fonctionnera pas, une erreur se produira quand numPiece sera inférieur à 0 (enfin pas tout à fait car python accepte les index négatifs, jusqu'à -len(listePiece), donc l'erreur arrivera quand numPiece vaudra -3 ce qui arrivera à un moment).
On peut éviter l'erreur en ajoutant la condition `numPiece>=0` dans la boucle `while`. Par contre à la fin, il faudra tester si `sommeARendre` vaut bien 0 et envoyer un message d'erreur si ce n'est pas le cas.

c) Problème du sac à dos

Le problème : Nous sommes devant un ensemble d'objets, chacun ayant une valeur et un poids. Nous souhaitons mettre dans le sac un maximum d'objets avec la valeur maximale sachant que le sac ne peut supporter qu'un certain poids maximal.

Dans ce problème, il convient de privilégier les objets légers qui ont une forte valeur au détriments d'objets lourds avec une valeur plus faible. Mais que faire avec des objets légers de faible valeur ou des objets lourds de forte valeur ?
Nous allons appliquer un algorithme glouton pour résoudre ce problème en définissant notre ordre de priorité.

Le problème : On se propose de définir un ordre de choix des objets en fonction de leur valeur et leur poids.
Comme on souhaite obtenir les objets avec la plus grande valeur et un poids minimal, on va prendre le rapport valeur/poids comme critère de sélection.
Donc tant que le sac n'est pas plein et qu'on peut placer un objet, on choisit celui dont le rapport valeur/poids est le plus important.

Nous allons établir le principe de l'algorithme, la traduction en python pouvant faire l'objet d'un travail supplémentaire :
On dispose d'une liste d'objets (ici, je prendrai une liste de tuples, le premier élément étant le nom de l'objet, puis sa valeur et enfin son poids), ce sera le premier argument de la fonction, le second étant le poids total que peut accepter le sac à dos.

```
fonction sacADos(listeObjets, poidsMax) :
    listeOrdonnee sera la liste d'objets par ordre décroissant du rapport valeur/poids
    i ← 0
    poidsDansSac ← 0
    poidsRestant ← poidsMax
    valeurDansSac ← 0
    listeSac ← [] (liste des objets mis dans le sac)
    Tant que le sac n'est pas plein et qu'il me reste des objets possibles :
      si le poids de l'élément d'index i est inférieur à poidsRestant
        on ajoute dans la liste cet objet dans le sac
        valeurDansSac ← valeurDansSac+valeur de l'objet
        poidsDansSac ← poidsDansSac+poids de l'objet
        poidsRestant ← poidsRestant-poids de l'objet
      i ← i+1 (on passe à l'objet suivant)
    renvoyer listeSac
```

Tester l'algorithme avec la liste suivante :
('objet1', 130, 14), ('objet2', 32, 2), ('objet3', 22, 5), ('objet4', 15, 4), ('objet5', 20, 6), ('objet6', 75, 8), ('objet7', 95, 11), ('objet8', 65, 7)
Quel objets l'algorithme nous propose-t-il de choisir ? Quelle est la valeur totale et le poids des différents objets choisis ?

Cet algorithme ne rend pas toujours la solution optimale, notamment quand on a un objet dont le poids est très important. Dans l'exemple ci-dessus, il existe une solution plus intéressante que celle que vous êtes censés trouver : objet1, objet4 et objet2 dont la valeur totale serait égale à 177 et le poids égal à 20.

Exercice : Créer une deuxième fonction, dont le principe serait le même, mais cette fois-ci on aurait une liste de dictionnaires ayant pour clés "Nom", "Valeur", "Poids".