Algorithme glouton

Situation

Pour résoudre certains problèmes des techniques, comme l'énumération exhaustive de toutes les solutions, ont un coût machine qui les rend souvent peu pertinentes au regard de contraintes extérieures imposées (temps de réponse de la solution imposé, moyens machines limités).
Les algorithmes gloutons constituent une alternative dont le résultat n'est pas toujours optimal mais qui sont simples à mettre en oeuvre et dont l'execution est rapide.

Définition

Le principe de l'algorithme glouton (greedy algorithm) :
faire toujours un choix localement optimal dans l'espoir que ce choix ménera à une solution globalement optimale.

⚠ On n'est pas certain que le résultat soit optimal.

Le problème du sac à dos

Ce problème fait partie des 21 problèmes NP-complets identifiés par Richard Karp en 1972. Ces 21 problèmes sont réputés comme les problèmes les plus difficiles en optimisation combinatoire. Un grand nombre d'autres problèmes NP-complets peuvent se ramener à ces 21 problèmes de base. Nous pouvons retrouver le problème du sac à dos dans de nombreux domaines :

- en cryptographie, où il fut à l'origine du premier algorithme de chiffrement asymétrique en 1976 ;
- dans les systèmes financiers, où l'idée est la suivante : étant donné un certain montant d'investissement dans des projets, quels projets choisir pour que le tout rapporte le plus d'argent possible ;
- pour la découpe de matériaux, afin de minimiser les pertes dues aux chutes ;
- dans le chargement de cargaisons (avions, camions, bateaux…) ; ou encore, dès qu'il s'agit de préparer une valise ou un sac à dos pour une randonnée.

Le problème du sac à dos

On dispose d'un sac à dos, ne pouvant supporter plus d'un certain poids, et d'un ensemble d'objets ayant chacun un poids et une valeur.
Le problème du sac à dos consiste à remplir le sac avec les objets pour que la valeur des objets mis dans le sac à dos soit maximisée, sans dépasser le poids maximum.

Pseudo-code

Données :
`objets` : liste d'objets.
`poids_max` : poids maximum supporté par le sac
Résultat :
`sac` à une liste contenant les objets à prendre.
Algorithme :
initialiser `sac` à la liste vide ;
initialiser `poids_sac` à 0 ;
calculer la valeur (v_i/p_i) pour chaque objet i,
trier tous les objets par ordre décroissant de cette valeur,
pour chaque objets de la liste
    si poids_objet + poids_sac < poids_max :
        mettre dans le sac
        poids_sac ⟵ poids_sac + poids_objet

Eléments de programmation

Trier une liste de dictionnaire :
Pour trier une liste de dictionnaire, il faut fournir en parametre de la méthode `sort`, une fonction renvoyant la clé sur laquelle qui va servir à faire le classement.

```
informaticiens = [ {"nom": "Ada", "naissance" :1815 },
    {"nom":"Timothy John Berners-Lee", "naissance":1955 },
    {"nom":"Linus Benedict Torvalds", "naissance":1969 },
    {"nom":"Dennis MacAlistair Ritchie", "naissance":1941 },
    {"nom":"Tim Paterson", "naissance":1956 },
    {"nom":"Alan Mathison Turing", "naissance":1912 },
    {"nom":"Richard Matthew Stallman", "naissance" :1953 }]

def get_naissance(personnage):
    return personnage["naissance"]

informaticiens.sort(key=get_naissance)
print(informaticiens)
#informaticiens = [ {"nom": "Ada Lovelace", "naissance" :1815 },
    {"nom":"Timothy John Berners-Lee", "naissance":1955 },
    {"nom":"Linus Benedict Torvalds", "naissance":1969 },
    {"nom":"Dennis MacAlistair Ritchie", "naissance":1941 },
    {"nom":"Tim Paterson", "naissance":1956 },
    {"nom":"Alan Mathison Turing", "naissance":1912 },
    {"nom":"Richard Matthew Stallman", "naissance" :1953 }]

def get_naissance(personnage):
    return personnage["naissance"]

informaticiens.sort(key=get_naissance)
print(informaticiens)
#[{'nom': 'Ada Lovelace', 'naissance': 1815}, {'nom': 'Alan Mathison Turing', 'naissance': 1912}, {'nom': 'Dennis Mac
```

Rendue de monnaie

On veut programmer une caisse automatique pour qu'elle rende la monnaie de façon optimale, c'est-à-dire avec le nombre minimal de pièces et billets. La valeur des pièces et billets à disposition sont : 1, 2, 5, 10, 20, 50, 100 et 200 euros. On dispose d'autant d'exemplaires qu'on le souhaite de chaque pièce et billet.

Exemple : Anaïs veut acheter un objet qui coûte 53 euros. Elle paye avec un billet de 100 euros. La caisse automatique doit lui rendre 47 euros. La meilleure façon de rendre la monnaie est de le faire avec deux billets de 20, un billet de 5 et une pièce de 2 euros.

Algorithme

Données :
la somme à rendre et la liste des pièces et billets à disposition.

Résultat :
une liste des pièces et billets à rendre.

Algorithme :
initialiser `monnaie` à la liste vide ;
initialiser `somme_restante` à somme ;
tant que la `somme_restante` est strictement positive :
`valeur_choisie` ⟵ la plus grande valeur qui ne dépasse pas `somme_restante`,
on ajoute cette `valeur_choisie` à `monnaie`,
`somme_restante` ⟵ `somme_restante - valeur_choisie` ;
renvoyer `monnaie`.