Algorithmes gloutons

Un algorithme glouton (greedy en anglais, qu'on pourrait aussi traduire par avide ou gourmand) fait à chaque étape le meilleur choix *local*, dans l'espoir d'obtenir un résultat optimum *global*. L'avantage de tels algorithmes est souvent leur efficacité : ils fournissent une solution non optimale du problème en un temps très court là où un algorithme exact peut très bien ne jamais terminer en pratique à cause de sa trop grande complexité. Dans le cas général, ils présentent l'inconvénient de ne pas toujours fournir une solution optimale du problème global.

1 Rendu de monnaie

Supposons qu'on cherche à programmer un distributeur (de boissons, de friandises, de sandwichs, de timbres, etc) de telle sorte qu'il rende automatiquement au client un nombre minimal de pièces à la suite d'un achat. En zone euro, les sommes correspondantes aux pièces (exprimées en centimes) sont 200, 100, 50, 20, 10, 5, 2 et 1.

Principe : à chaque étape de choix d'une pièce à rendre, on choisit celle de plus grande valeur possible.

Écrire une fonction `rendu(s, pieces=(200, 100, 50, 20, 10, 5, 2, 1))` qui retourne la liste des valeurs des pièces correspondant à la somme à rendre `s`.

Exemple. `rendu(479)` doit retourner `[200, 200, 50, 20, 5, 2, 2]`.

Respecter au maximum les principes de bonne programmation :

- annotations et documentation des fonctions à écrire avant le code proprement dit.
- découpage en fonctions courtes si besoin.

On pourra choisir une (ou plusieurs) fonction(s) impératives ou récursives. Il faut savoir faire les deux !

Quelle serait la structure de données la plus adaptée pour stocker le résultat ? Modifier les fonctions précédentes pour en tenir compte.

Quelle est la complexité de l'algorithme dans le pire cas ?

2 Problème du voyageur de commerce

La problématique est celle d'un livreur : il y a `n` points à visiter (chaque point est un couple de coordonnées), et partant d'un point de départ, on cherche le chemin qui minimise la distance totale parcourue.

Principe : le point suivant est le plus proche du point actuel parmi les points non visités.

Questions à se poser :

- Comment stocker un point ? Un chemin ?
- Comment afficher le résultat obtenu ?
- Quels points choisir pour tester ?

Écrire toutes les fonctions nécessaires pour implémenter cet algorithme et le tester.