Chapitre n° 12 : commentaires

1 Algorithmes de tris

1.1 Tri par sélection

1.1.1 Principe de l'algorithme

1.1.2 Programmation en Python

1.1.3 Terminaison de l'algorithme

1.1.4 Correction de l'algorithme

1.1.5 Coût de l'algorithme

1.2 Tri par insertion

1.2.1 Principe de l'algorithme

1.2.2 Programmation en Python

1.2.3 Terminaison de l'algorithme

1.2.4 Correction de l'algorithme

1.2.5 Coût de l'algorithme

2 Algorithme des plus proches voisins

2.1 Présentation des algorithmes d'apprentissage

2.2 Notion de distance

2.3 Exemple

3 Algorithmes gloutons

3.1 Présentation des algorithmes d'optimisation

3.2 Problème du sac à dos

On a maintenant les fonctions qui nous permettent (chacune selon un critère) de choisir le choix optimal localement.

Il faut ensuite créer la fonction *glouton*, qui appellera, selon le choix d'argument, une des trois fonctions précédentes.

Nous avons vu des algorithmes de tri en début de chapitre, mais dans ce programme nous allons utiliser une fonction de tri fournie avec Python : *sorted*. Il faut obligatoirement fournir en argument la liste qui doit être triée, et affecter

la liste triée dans une variable.
Mais il est possible de préciser d'autres arguments, ce qui est fait ici. On précise d'abord le critère de tri (une des trois fonctions précédentes, selon le critère choisi pour l'algorithme glouton).
Ensuite, comme on souhaite sélectionner l'élément ayant la plus grande valeur selon le critère choisi, on pourrait ensuite extraire le dernier élément de la liste, mais il est beaucoup plus pratique de trier la liste par ordre décroissant, et de prendre à chaque fois le premier.

Il faut comprendre le code de cette fonction glouton. Il faudrait pouvoir dérouler l'algorithme dans un cas donné, notamment dans le cas de l'exemple présenté.
Si ce code n'est pas clair, il ne faut pas hésiter à poser des questions par mail.

3.3 Problème du rendu de monnaie

La présentation avec le n-uplet peut paraître abstraite, mais avec l'exemple concret des euros ensuite ça doit devenir plus clair.

La contrainte du problème est simple à identifier : la somme des valeurs des pièces doit être égale à la somme à rendre.

Le critère de solution optimale est simple ici : le nombre de pièces utilisé doit être minimal.

L'exemple présenté ensuite consiste à voir comment rendre 8 €.
Le nombre de combinaisons possibles étant assez faible, il est possible de toutes les parcourir pour trouver la solution optimale. C'est ce que fait le cours, d'abord « à la main », puis en proposant un programme pour automatiser la démarche.

On peut observer qu'avec trois types de pièces impliquées, ce programme utilise trois boucles imbriquées, le coût peut potentiellement devenir cubique si on augmente la valeur de la somme à rendre. Et si le nombre de type de pièce augmente, la puissance intervenant dans le coût aussi : ça sera très vite prohibitif !

C'est pour cela qu'on applique ensuite un algorithme glouton pour résoudre ce problème.
La variable *solution* est une liste de n éléments, où n est le nombre de type de pièces et billets existants. Au départ, *solution* est une liste de n 0, car aucune pièce n'a encore été rendue.

i est initialisé à la valeur $n - 1$, c'est à dire l'indice du dernier élément. L'algorithme glouton va d'abord tester s'il peut rendre une pièce (ou un billet de plus grande valeur). Si la valeur de la pièce est supérieure à la somme à rendre, i est décrémenté de 1, on regardera alors s'il est possible de rendre une pièce de deuxième plus grande valeur. Et ainsi de suite, on parcourt le système de pièces possibles, par valeurs décroissantes, jusqu'à trouver la pièce de plus grande valeur, ne dépassant pas la somme restant à rendre.

Par exemple, s'il faut rendre $r = 8$€, les billets de 500€, 200€, 100€, 50€, 20€ et 10€ ont une valeur supérieure à r.
En revanche, le billet de 5€ est de valeur inférieure à r. Il est donc rendu, et la nouvelle valeur de r vaut $r = 8 - 5 = 3$€.
5€ est de valeur trop importante, mais pas 2€. On rend alors une pièce de 2€, et il reste une somme à rendre de $r = 3 - 2 = 1$€.
2€ est de valeur trop importante, mais pas 1€. On rend alors une pièce de 1€, et il reste une somme à rendre de $r = 1 - 1 = 0$€. L'algorithme prend fin
Au final, l'algorithme glouton donne la solution optimale : un billet de 5€, une pièce de 2€ et une pièce de 1€.

Il faut comprendre le code de cette fonction glouton. Il faudrait pouvoir dérouler l'algorithme dans un cas donné.
Si ce code n'est pas clair, il ne faut pas hésiter à poser des questions par mail.

Le problème du rendu de monnaie est intéressant, car pour certains systèmes de monnaies (comme l'euro) l'algorithme glouton fournit systématiquement la solution optimale, mais ça ne sera pas le cas avec d'autres systèmes.