TD 09 Algorithme glouton

Lycée Louis Thuillier - Informatique - PCSIB -

Problème du rendu de monnaie

Définition et première approche du problème

On s'intéresse au problème de rendu de monnaie : on veut rendre à un client un montant d'argent spécifique, noté n. On possède pour cela différents billets et pièces de valeur 500, 200, 100, 50, 20, 10, 5, 2, et 1 euros (on considère que n est un entier, tant pis pour les centimes).
On souhaite néanmoins optimiser le processus en faisant en sorte **de lui rendre la monnaie en utilisant le plus petit nombre de billets et de pièces possibles**.

Un tel problème est difficile a priori à résoudre puisqu'il faut non seulement trouver une solution, c'est-à-dire ici trouver une combinaison de pièces et de billets qui fait la somme n voulue, mais en plus cette solution doit être la solution optimale, qui utilise le moins de pièce/billets possible.

> ***Application 1 :*** *On définit un rendu de monnaie sous la forme d'un tableau à deux dimensions de 2 lignes et 9 colonnes où :*
> ▷ *la première ligne contient les valeurs des billets*
> ▷ *la seconde ligne contient le nombre rendu de monnaie correspondant*
>
> 1. *Ecrire une fonction le_compte_est_bon qui, prenant en argument un tableau de rendu de monnaie et un nombre n renvoie $True$ si le rendu de monnaie R est égal à la somme n, $False$ sinon.*
> 2. *A l'aide de la fonction précédente, écrire une fonction nombre_de_billets qui, prenant en argument un rendu de monnaie R et une somme n, renvoie :*
>   ▷ *$False$ si le rendu de monnaie n'est pas égale à n*
>   ▷ *le nombre N de billets si le rendu de monnaie est correct*

Solution optimale du rendu de monnaie

Pour trouver la solution optimale, on pourrait générer tous les rendus de monnaie, vérifier ceux qui marchent puis rechercher parmi ceux là celui qui utilise le moins de billets. Néanmoins cette méthode est longue et fastidieuse, le nombre de possibilité de rendu de monnaie étant énorme. On va plutôt chercher à construire pas à pas la solution optimale.

Pour cela, on va utiliser le principe suivant :
▷ on va utiliser le billet de plus grande valeur, en faisant attention à ne pas dépasser la somme totale restante à rendre
▷ on l'exclue de la liste des billets disponible
▷ on calcule la somme qui reste à rendre
▷ on passe à la coupure suivante et ainsi de suite

> ***Exemple 1 :*** *On veut rendre 422 euros :*
> ▷ *en partant de 500, on donne 0 billets*
> ▷ *avec 200 on donne 2×200 soit 400 euros avec deux billets*
> ▷ *il reste à rendre $422 - 400 = 22$ euros*
> ▷ *on passe à la coupure 100*
> ▷ *...*
> *En fin de compte on rende $422 = 2 \times 200 + 2 \times 20 + 1 \times 2$ 2 billets de 200 euros, 2 de 20 euros et 1 de 2 euros.*

On peut montrer, mais on ne le fera pas ici, que cette solution est optimale.

> *Application 2 :* ***En classe***
> *Ecrire une fonction "monnaie" qui, recevant un montant entier, renvoie une liste dont les éléments correspondent aux nombres de billets ou de pièces de chaque valeur rendus (on rangera les coupures par ordre décroissant).*

Algorithme glouton

Définition du problème

Le problème de rendu de monnaie est un problème **d'optimisation combinatoire** : on cherche une combinaison (ici le nombre de billets de 500/200/.../1 euros) qui
1. soit solution du problème : la somme des billes est égale à la somme qu'on doit rendre
2. soit optimale : le nombre de billets utilisés est minimal

La première condition définit **les solutions réalisables**, la deuxième permet de définir la (ou les) **solution optimale**.
Tout problème d'optimisation combinatoire est alors défini par
- ▷ un ensemble : l'ensemble des solutions réalisables
- ▷ une application qu'on cherche à minimiser/maximiser

Algorithme glouton

La solution trouvée précédemment est obtenue à l'aide d'un **algorithme glouton**, aussi appelée stratégie gloutonne.

Définition. Algorithme glouton
Une stratégie gloutonne permet de construire une solution réalisable d'une problème d'optimisation combinatoire en procédant par étape. Le principe d'une stratégie gloutonne est de construire la solution globale en choisissant à chaque étape la solution optimale locale.

Il est important de noter que le choix est définitif (on ne fait pas machine arrière).

> *Application 3 : (Option, à faire à la maison)*
> *On considère le problème de combinatoire suivant : étant donné une liste L de chiffres, construire le nombre le plus grand en n'utilisant qu'une fois chacun des chiffres.*
> 1. *Donner la solution pour la liste* $[3, 8, 6, 4]$
> 2. *Proposer une stratégie gloutonne au problème général. Dans un premier temps on ne demande pas d'écrire le programme juste de décrire les étapes.*
>   *Cette stratégie est-elle optimale ?*
> 3. *Ecrire un programme "nombre" qui, prenant en entrée une liste de chiffres L, renvoie le plus grand nombre constitué de tous les chifres de L*

Attention ! Si la stratégie gloutonne donne toujours une solution réalisable, elle ne donne pas toujours la solution optimale ! !

algorithme glouton $\iff$ succession de choix optimaux à chaque étape

Application : chemin de poids maximal

A faire à la maison et à rendre par mail ou sous clef en point ".py" avant la fin de la prochaine séance.
Suivant le niveau de chacun, les questions (*) sont optionnelles.

▶ Notion de chemin

On considère un tableau d'entier à deux dimensions. On appelle "chemin" les trajectoires qui partent de la case en haut à gauche pour arriver à la case en bas à droite en ne se déplaçant que vers la droite ou vers le bas.
Pour représenter un chemin, on utilisera une chaîne de caractères où "d représente un mouvement vers la droite et "b" un mouvement vers le bas.

Exemple : On prend le tableau suivant

$$\begin{pmatrix} 2 & 16 & 1 & 3 \\ 8 & 90 & 1 & 23 \\ 3 & 78 & 41 & 1 \end{pmatrix}$$

Le chemin "*ddbdb*" est donc $2 \to 16 \to 1 \to 1 \to 23 \to 1$.

1. Ecrire une fonction "*chemin*" qui prend en entier un tableau T et un chemin c sous la forme d'une chaîne de caractères et renvoie une liste constituée des nombres rencontrés lors du déplacement.

▶ Poids d'un chemin

On appelle poids du chemin la somme des entiers rencontrés au cours du déplacement.
Exemple : le poids du chemin $2 \to 16 \to 1 \to 1 \to 23 \to 1$ est $2 + 16 + 1 + 1 + 23 + 1 = 44$.

2. Ecrire une fonction *poids_chemin* qui prend en entier un tableau T et un chemin c sous la forme d'une chaîne de caractères et renvoie le poids du chemin.

On cherche le chemin de poids maximal.

▶ Stratégie gloutonne

3. Proposer une stratégie gloutonne à ce problème d'optimisation.
   (a) dans un premier temps, décrire juste les étapes.
   (b) écrire un programme "*poids_max_glouton*" qui prend en entrée un tableau T et renvoie le poids du chemin trouvé par stratégie gloutonne
   (c) Cette méthode est-elle optimale ?
   On pourra proposer un contre-exemple simple pour répondre.

4. Pour trouver la solution optimale, nous allons développer une autre approche. : on va chercher à construire un tableau de même taille que T où chaque case contient le poids du chemin maximal arrivant à cette case.
   Exemple : Avec le tableau précédent :

$$\begin{pmatrix} 2 & 16 & 1 & 3 \\ 8 & 90 & 1 & 23 \\ 3 & 78 & 41 & 1 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 18 & 19 & 22 \\ 10 & 108 & 109 & 132 \\ 13 & 186 & 227 & 228 \end{pmatrix}$$

   (a) donner le tableau des poids maximaux locaux associé à :

$$\begin{pmatrix} 1 & 2 & 0 & 1 \\ 2 & 3 & 0 & 2 \\ 3 & 2 & 1 & 1 \end{pmatrix}$$

   Astuce : on construira le tableau pas à pas en partant de en haut à gauche et en remplissant chaque case en comparant le poids maximal local des deux cases qui y mènent.

   (b) (*) Écrire une fonction "*poids_max_local*" qui prend en entrée un tableau T et renvoie un tableau de même taille dont chaque case contient le poids d'un chemin maximal arrivant à cette case.
   (c) A partir du tableau des poids maximaux, proposer une façon simple de trouver le chemin de poids maximal
   (d) (*) Écrire une fonction "*chemin_max*" qui prend en entrée un tableau T et renvoie le chemin de poids maximal sous la forme d'une chaine de caractères de "d" et de "b".