info803 : informatique

Cours / TD : programmation dynamique, algorithmes gloutons

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22
téléphone : 04 79 75 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/

Exercice 1 : comment faire de la monnaie

La boulangère dispose d'une quantité illimitée de pièces de 1, 2, 5 et 10 centimes. Elle peut obtenir n'importe quelle somme à partir de ces pièces, mais comment faire pour minimiser le nombre total de pièces utilisées ?

Question 1 : proposez un algorithme simple pour trouver une solution optimale. L'algorithme devra prendre en argument la somme S voulue et devra renvoyer une solution qui minimise le nombre total de pièces utilisées. Montrez que votre algorithme fonctionne et que la solution optimale est unique.

Question 2 : essayez d'adapter votre algorithme pour des pièces de monnaies de valeurs quelconques $p_1 < p_2 < \cdots < p_n$.

Question 3 : sur la planète Triton-42, les habitants utilisent des pièces de 1, 4 et 6 zorkmids. Que pensez-vous de l'algorithme précédent sur la planète Triton-42 ?

Question 4 : en utilisant la programmation dynamique, calculer le nombre minimal de pièces pour faire la monnaie sur S avec des pièces de valeurs $p_1, \dots, p_n$. Pour ceci, généralisez le problème et cherchez le nombre $M(S, i)$: "nombre minimal de pièces pour atteindre S si on n'utilise que des pièces de valeurs $p_1, \dots, p_i$".
Adaptez l'algorithme pour trouver également la quantité de chaque pièce nécessaire.

Question 5 : cherchez des familles de pièces pour lesquelles l'algorithme de la question 1 fonctionne. Comparez les complexités des deux algorithmes.

Exercice 2 : algorithmes gloutons, programmation d'une chaîne sportive

Un algorithme est dit "glouton" s'il essaie toujours de prendre la meilleure solution localement. (Il veut "beaucoup, tout de suite".) Comme le montre l'exercice précédent, les algorithmes gloutons ne donnent pas toujours la meilleure solution. Comme ils sont en général très efficaces, il est quand même important de savoir les reconnaître et quand les utiliser..

On s'intéresse maintenant à la programmation d'une chaîne de télévision sportive. À tout instant, il y a de nombreux évènements sportifs ; le directeur de la chaîne essaie de diffuser le plus d'évènements en direct possibles, indépendamment de leur durée. Pour chaque jour on dispose donc d'une liste d'évènements numérotés de 1 à n. À chaque évènement i est associé une heure de début d_i et une heure de fin f_i. Comment faire pour caser le plus d'évènements dans la même journée ?

Question 1 : on commence par utiliser un algorithme glouton qui met les évènements courts en premier (on maximise le nombre d'évènements potentiels). Quelle est la complexité ? Est-ce que ça marche ?

Question 2 : on change la stratégie et on positionne en premier les évènements qui commencent en premier (on minimise le temps d'attente). Décrire l'algorithme ; quelle est sa complexité ? Est-ce que ça marche ?

Question 3 : on essaie une dernière possibilité en triant les évènements par horaire de fin. Décrire l'algorithme ; quelle est sa complexité ? Est-ce que ça marche ?

Exercice 3 : codage de Huffman

Pour encoder un texte écrit avec n symboles (les caractères alphanumériques par exemple), il est pratique de considérer les codage binaires : chaque symbole est représenté par une suite de `0` et de `1`, et le texte final est représenté par la concaténation des codes de ses caractères.

Le codage binaire le plus connu est sans doute le code ASCII (American Standard Code for Information Interchange) : c'est un codage de longueur fixe, ou chaque lettre (et quelques autres symboles) sont représentés sur 7 bits. Certaine extensions utilisent le 8-ème bit d'un octet pour rajouter les accents. Un codage binaire de longueur fixe est facile à décoder : on découpe le texte codé en morceau de bonne longueur, et on décode chaque morceau...

Les codages *préfixes* sont des codages de taille variable (tous les caractères ne sont pas représentés par le même nombre de bits) qui sont eux aussi faciles à décoder.

Pour compresser un texte, nous allons essayer de représenter les caractères fréquents par des petites suites de bits, et les caractères peu fréquents par des grandes suites de bits. Un tels codage est *optimal* par rapport à un texte T s'il minimise le nombre de bits nécessaire pour l'encodage du texte T.

Question 1 : un codage binaire est *préfixe* si aucun de ses codes n'est préfixe d'un autre code. Montrez pourquoi un codage préfixe est facile à décoder. Comment peut-on utiliser la structure de données "arbre binaire" pour décoder ?

On suppose maintenant que l'on veut encoder un texte et que l'on connaît le nombre d'occurrences de chaque symbole. On veut créer un codage préfixe qui associe un petit code aux symboles fréquents et un grand code aux symboles peu fréquents. Pour ça, on utilisera un arbre binaire complets.

Pour exemple, on considérera un "texte" contenant uniquement les lettres `a`, `b`, `c`, `d`, `e` et `f` avec les fréquences suivantes :

`a` : 55, `b` : 14, `c` : 12, `d` : 32, `e` : 10, `f` : 7

Question 2 : combien de bits seront nécessaires pour encoder le texte avec un codage de taille fixe ? Dessinez l'arbre du code.

Question 3 : un algorithme naïf consiste à associer `0` pour la lettre la plus fréquente, `10` pour la lettre suivante, `110`, ..., `11`$\cdots$`10` pour l'avant dernière lettre et `11`$\cdots$`11` pour la dernière lettre. Dessinez l'arbre correspondant au texte de l'exemple et calculer la longueur du codage.

Question 4 : malheureusement, l'algorithme glouton précédent ne donne pas un codage optimal. L'algorithme de Huffman va associer des codes aux symboles de la manière suivante :

- on trie les symboles par fréquence
- on fusionne les deux symboles de plus basse fréquence dans un arbre binaire, et la fréquence cet arbre devient la somme des deux fréquences
- on insère cet arbre dans la liste des symboles. (on a auparavant supprimé les deux symboles fusionnés...)
- on recommence jusqu'à obtenir un unique arbre binaire

Expliquez pourquoi cet algorithme peut être considéré comme un algorithme glouton. Utilisez cet algorithme pour construire le code associé a texte de l'exemple. Que devient la taille du texte encodé ?

Question 5 : démontrez les lemmes suivants et concluez.

Lemme. *Il existe toujours un codage binaire optimal pour lequel les deux caractères les moins fréquents ont des codes qui ne diffèrent que par le dernier bit.*

Lemme. *Soit x et y sont les deux lettres de plus basses fréquences, et C' est un codage optimal pour le texte où x et y sont identifiées en un seul symbole z (avec donc un nombre d'occurences égal la somme des nombres d'occurences de x et y). Si on remplace le noeud de z dans ce codage optimal par un branchement sur deux soeurs x et y, on obtient un codage optimal pour le texte de départ...*