IN 101 - TD 04
8 octobre 2010

Énoncé

ENSTA ParisTech

Exercice 1 : Algorithme glouton pour le rendu de monnaie

On veut programmer un algorithme glouton pour le rendu de monnaie. On ne s'intéresse qu'à des sommes de monnaie entières et on veut donc pouvoir écrire la décomposition d'un entier entré en paramètre sur la ligne de commande comme somme des éléments 500, 200, 100, 50, 20, 10, 5, 2 et 1. Écrivez un algorithme glouton qui affiche la liste des billets/pièces à rendre. Utilisez pour cela un tableau d'entiers `int s[9] = {500, 200, 100, 50, 20, 10, 5, 2, 1}`. L'ensemble de l'algorithme peut être programmé dans le `main`.

Modifiez la valeur de quelques billets/pièces de la liste de façon à ne plus avoir de pièce de 1 (par exemple `int s[9] = {500, 200, 100, 50, 20, 10, 5, 3, 2}`). Maintenant l'algorithme glouton risque de ne pas pouvoir trouver de solution, même s'il en existe une. Vérifiez que votre algorithme fonctionne aussi (ne fait pas d'erreur de segmentation) dans les cas où il n'arrive pas à rendre la monnaie. Vous pouvez lui faire afficher un message spécial dans ce cas.

Exercice 2 : Calcul de coefficients binomiaux

On veut programmer une fonction qui calcule les coefficients binomiaux $\binom{n}{p} = C_p^n$, le nombre de façons de choisir p éléments parmi n. Pour cela on va utiliser plusieurs techniques différentes.

2.a] Il est connu que $\binom{n}{p} = \frac{n!}{p!(n-p)!}$. Programmez une fonction `factoriel` ayant pour signature `int factoriel(int n)` qui retourne $n!$ (cette fonction peut être récursive ou non, cela n'a pas d'importance). En utilisant cette fonction, implémentez une première fonction `binom1` ayant pour signature `int binom1(int n, int p)` qui calcule $\binom{n}{p}$. Vérifiez que votre fonction retourne bien les bonnes valeurs de $\binom{4}{2}$, $\binom{10}{3}$ et $\binom{7}{4}$ par exemple.

2.b] Que retourne votre fonction pour $\binom{100}{3}$? Essayez de comprendre ce qui pose problème.

2.c] Pour ne jamais manipuler d'entiers plus grands que $\binom{n}{p}$ à aucun moments des calculs, on va utiliser la formule de récurrence du triangle de Pascal $\binom{n}{p} = \binom{n-1}{p-1} + \binom{n-1}{p}$. Programmez une fonction récursive `binom2` (ayant la même signature que `binom1`) qui calcule les coefficients binomiaux à partir de cette formule. ⚠ Il faut deux conditions d'arrêt ici.
Vérifiez que `binom2` retourne les mêmes valeurs que `binom1` sur des petites valeurs. Vérifiez que $\binom{100}{3}$ vaut bien 161700. Que vaut $\binom{100}{6}$?

2.d] Le calcul de $\binom{100}{6}$ avec `binom2` peut prendre un peu de temps. Pourquoi ?

2.e] Pour améliorer cela on propose d'utiliser le principe de la programmation dynamique. Implémentez une fonction `binom3` similaire à `binom2`, mais utilisant une variable globale `tab` (un tableau à 2 dimensions) pour faire de la programmation dynamique. Vérifiez que `binom3` retourne les mêmes valeurs que `binom1` et `binom2` sur des petites valeurs. Le calcul de $\binom{100}{6}$ devrait maintenant être instantané. Quelle est la complexité de cette algorithme ?

2.f] (Pour ceux qui vont vite) Implémentez une fonction `binom4` encore plus rapide utilisant : $\binom{100}{96} = \binom{100}{4} = \frac{100 \cdot 99 \cdot 98 \cdot 97}{1 \cdot 2 \cdot 3 \cdot 4}$ et donc $\binom{100}{5} = \binom{100}{4} \times \frac{96}{5}$. ⚠ On ne veut que des calculs d'entiers et on veut des calculs exactes. Pour ne jamais manipuler d'entiers plus grands que $\binom{n}{p}$ il est donc nécessaire de réduire la fraction $\frac{n-p+1}{p} = \frac{n'}{p'}$, puis diviser $\binom{n}{p-1}$ par le nouveau dénominateur p' avant de faire le produit par le nouveau numérateur n'.

Exercice 3 : Exponentiation binaire

Le problème de l'exponentiation consiste, étant donnés deux entiers a et b, à calculer a^b. Afin de réaliser efficacement ce calcul, l'idée des algorithmes dits d'*exponentiation binaire* est de décomposer b en base 2 :

$$b = \sum_{i=0}^{k} b_i \times 2^i \quad \text{avec} \quad b_i \in \{0, 1\}$$

On peut ensuite remarquer

$$a^b = a^{\left(\sum_{i=0}^{k} b_i \times 2^i\right)} = \prod_{i=0}^{k} a^{b_i \times 2^i} = \prod_{i=0}^{k} \left(a^{(2^i)}\right)^{b_i}$$

Afin de calculer a^b, il suffit donc d'être capable de calculer les $a^{(2^i)}$ qui s'obtiennent par élévation successive au carré et d'en multiplier ensuite certains entre eux (ceux pour lesquels b_i vaut 1).

3.a] Quelle est la complexité, en termes de nombre de multiplications d'entiers, de l'algorithme naïf d'exponentiation réalisant l'opération $a \times a \times \ldots \times a$?

3.b] Écrire un algorithme décomposant un entier en base 2 et qui affiche les bits de la décomposition. Quelle est sa complexité ?

3.c] En déduire une fonction capable de calculer rapidement a^b en stockant les puissances $a^{(2^i)}$ dans un tableau de k cases et en utilisant la décomposition précédente. Choisissez un k tel que $b < 2^k$. Quelle est sa complexité temporelle, dans le cas le pire et en moyenne ?

3.d] Modifier la fonction précédente de manière à ce que sa complexité en espace soit constante (c'est-à-dire indépendante de la taille de l'exposant b, et donc, sans utiliser de tableau pour stocker les puissances de a).

Exercice 4 : Application à la cryptographie

Le système de chiffrement RSA, très utilisé aujourd'hui, repose sur l'opération d'exponentiation modulaire, puisque, à partir d'un message m représenté par un entier, on obtient son chiffré c en calculant :

$$c = m^e \mod n$$

où mod n désigne la réduction modulaire « modulo n ».

4.a] Comment adapter l'algorithme précédent pour intégrer les réductions modulaires ?

4.b] Les nombres manipulés (n, m et e) sont typiquement longs de 1024 bits. Dans ce cas, combien d'opérations modulaires sont nécessaires pour calculer le chiffré d'un message ?

4.c] On peut dans certains cas se contenter d'exposants plus petits. Un choix classique est $e = 65537$. Pourquoi ce nombre plutôt qu'un autre nombre de 17 bits ?