Devoir d'informatique

(candidats à l'option informatique)

Serge Dupont (2004)

Cette deuxième partie comporte deux problèmes indépendants. La formulation est succinte. On apportera donc un soin particulier à la présentation des réponses, et en particulier à la *modularité* : mieux vaut écrire beaucoup de petits programmes qu'un ou plusieurs gros. Si votre programme repose sur une ou plusieurs idées qui ne sont pas évidente d'après l'énoncé, commencer par une présentation de ces idées. De même, choisir des noms de variables locales explicites et commenter votre programme grâce au symbole #.

Il serait *grandement apprécié* d'indiquer pour chaque programme sa complexité, c'est-à-dire le nombre d'opérations élémentaires effectuées (au pire) en fonction de la taille de la donnée (*cf* ci-dessous).

1 Problème 1 : partitions d'un entier

Soit N un entier strictement positif. On appelle *partition* de N toute suite finie $(k_1, k_2, ..., k_r)$ d'entiers strictements positifs tels que $k_1 \geq k_2 \geq \cdots \geq k_r$ et $k_1 + k_2 + ... + k_r = N$. On note $p(N)$ le nombre de partitions de l'entier N. Dans ce problème, on représentera les partitions par des listes.

On définit l'ordre lexicographique sur les partitions de N par la propriétés suivantes : L_1 est plus grande que L_2 si pour un certain k, tous les termes $< k$ de L_1 et L_2 sont égaux et $L_1[k] > L_2[k]$. Ces listes n'ont pas forcément le même nombre de termes.

Ecrire un programme **succ** qui à une partition de N qui n'est pas la dernière (pour l'ordre lexicographique) associe la suivante dans l'ordre lexicographique, c'est-à-dire la plus grande parmi les plus petites.

Ecrire un programme **partition** qui à un entier strictement positif N associe la liste ordonnée décroissante de ses partitions. En déduire un programme **nbpart** qui à l'entier N associe son nombre de partitions. Jusqu'à quel entier N arrivez-vous à calculer **nbpart(N)** ? Comparer avec la fonction **combinat[partition]**. Dessiner la courbe représentative de p grâce à **plots[listplot]**. Tester vos résultats, votre vitesse de calcul et comparer avec celle de Maple.

2 Problème 2 : codage CLE d'un entier

Soit N un entier strictement positif. On rappelle que tout entier N admet une représentation binaire sous forme d'une suite finie L de 0 et de 1, dont le dernier terme $L[t]$ n'est pas un 0. Plus précisement : $N = \sum_{k=0}^{t} L[k] * 2^k$. L'entier t s'appelle la *taille* de N.

Dans le but d'écrire de manière courte les grands entiers, le codage CLE (Code Large Echelle) associe à l'entier N la liste C constituée des $L[k]$ qui valent 1 dans la décomposition en base deux de N. Par exemple, 67 s'écrit $L = [1, 0, 0, 0, 1, 1]$ en base deux et donc $C = [6, 1, 0]$ en code CLE.

Dans le but de simplifier les procédures qui suivent, on pourra choisir de représenter les nombres CLE par une suite *croissante*, afin que le chiffre des unités soit le premier. Avec ce choix, 67 s'écrit $[0, 1, 6]$.

2.1 Question 1

Ecrire un programme **convCLE** qui convertit un entier N en code CLE. Le résultat est donc une liste. Ecrire l'algorithme réciproque **deconvCLE**.

2.2 Question 2

Ecrire un programme **divisible_puiss_2**(**L** :: **list**, **k** :: **nonnegint**) qui renvoie *true* si le code CLE L est divisible par 2^k et *false* sinon. Ecrire des programmes **divisible3**(**L**) et **divisible5**(**L**) qui renvoient *true* si L est divisible par respectivement 3 et 5 et *false* sinon.

2.3 Question 3

Ecrire un programme **CLEsom(L,M)** qui donne le code CLE de la somme des nombres CLE L et M. (On fait les calculs directement en code CLE.) Ecrire de même un programme **CLEprod(L,M)** qui calcule le produit de L par M.

2.4 Question 4

Ecrire deux programmes **CLEquo(L,M)** et **CLEreste(L,M)** qui donnent respectivement le quotient et le reste dans la division de L par M en code CLE.

2.5 Application

Effectuer les calculs suivants en code CLE, et en profiter pour vérifier vos programmes :

```
> L1:=[11,5,3,0]: L2:=[34,11,5,3]: L3:=[18,16,8,4,3,2]:
> L4:=[19,16,9,4,3,2,1]:
> L1+L2; L3+L4; L1*L3; L2*L4; L4^2; iquo(L2,L3); irem(L2,L3);
L3^L1;
```