Passeport pour l'informatique

# EXERCICES D'ALGORITHMIQUE

Oraux d'ENS

Ouvrage collectif — Coordination Jean-Claude Bajard

INTERNATIONAL THOMSON PUBLISHING

# 1

# Arithmétique et calculs numériques

« *Écrire. Calculer. Résoudre.* »

## 1.1 Parties d'un ensemble

On considère un ensemble $E = \{0, ..., N-1\}$. On représente une partie de $E$ par un tableau `p` d'entiers de $\{0, 1\}$ de taille $N$ tel que `p.(i)=1` si et seulement si l'élément $i$ est dans la partie représentée par `p`. On associe à chaque partie `p` son numéro défini par numéro(`p`) $= \sum_{i=0}^{N-1}$ `p.(i)`$2^i$.

Question 1

Écrire un algorithme qui, étant donnée une variable `p` représentant une partie, calcule numero(`p`). Quel est le nombre de multiplications par 2 effectuées par votre algorithme ? Pouvez-vous diminuer ce nombre ?

Question 2

Étant donné un entier positif ou nul $k$, montrer qu'il existe au plus une partie de `p` telle que numero(`p`) $= k$. Écrire un algorithme qui donne cette partie `p` lorsqu'elle existe. On pourra utiliser la fonction OCaml `mod` (si `a` et `b` sont deux entiers positifs, `a mod b` est le reste de la division euclidienne de `a` par `b`).

Question 3

Écrire un algorithme qui, étant donnée une partie `p`, calcule la partie `q` (lorsqu'elle existe) telle que numero(`q`) $=$ numero(`p`) $+ 1$.

Question 4

Écrire un algorithme qui énumère toutes les parties de $E$.

CORRIGÉ

Question 1

On donne la fonction :

```
let numero (p:int array) : int =
  let n = Array.length p in
  let s = ref 0 in
  let pow = ref 1 in
  for i=0 to n-1 do
    s := !s + p.(i) * !pow;
    pow := 2 * !pow
  done;
  !s
```

Le nombre de multiplications effectuées par cet algorithme est de $N$. La puissance maximale de 2 à calculer étant $2^{N-1}$, $N-1$ multiplications sont nécessaires. Il est facile d'écrire un algorithme, peut-être un tout petit peu moins naturel que celui présenté, effectuant $N-1$ multiplications.

Question 2

Soit $k \in \mathbb{N}$. S'il existe une partie `p` de $E$ telle que numero(`q`) $= k$, la suite `p.(n-1)`, `p.(n-2)`, ..., `p.(0)` est exactement l'écriture en binaire du nombre $k$. Ainsi, si la partie `p` existe, elle est unique. De plus, `p` existe si et seulement si $0 \leq k \leq \sum_{i=0}^{N-1} 2^i = 2^N - 1$.

Dans ce cas, les éléments `p.(i)` sont obtenus en calculant successivement `k mod 2`, `(k/2) mod 2`, ...

```
let partie (k:int) (n:int) : int array =
  let p = Array.make n 0 in

  let k = ref k in
  for i = 0 to n-1 do
    p.(i) <- !k mod 2;
    k := !k/2;
  done;
  if !k <> 0 then failwith "k est trop grand";
  p
```

Cette fonction a l'inconvénient de faire des calculs inutiles si $k$ est petit par rapport à $2^N$. La fonction suivante est, de ce point de vue, plus efficace.

```
let partie2 (k:int) (n:int) : int array =
  let p = Array.make n 0 in
```

```
  let rec aux k i =
    if i > n then failwith "k est trop grand";
    if k > 0 then (
      p.(i) <- k mod 2;
      aux (k/2) (i+1)
    )
  in aux k 0;
  p
```

Question 3

Cette question revient à déterminer le successeur d'un entier écrit en binaire. Il suffit de parcourir les chiffres de cet entier de droite à gauche et de transformer les 1 en 0 jusqu'au moment où on trouve un 0 que l'on transforme en 1.

Exemple : $\begin{array}{r} 101011 \\ +1 \\ \hline 101100 \end{array}$

```
  let n = Array.length p in
  let q = Array.copy p in

  let rec aux i =
    if i > 0 then (
      if p.(i) = 1 then (
        q.(i) <- 0;
        aux (i-1)
      ) else q.(i) <- 1
    ) else if p.(i) = 1 then
      failwith "pas de successeur"
    else q.(i) <- 1
  in aux (n-1);
  q
```

Question 4

Il suffit d'appliquer l'algorithme de la question précédente de manière itérative à partir de la partie vide.

```
let affiche_partie (p:int array) =
  let n = Array.length p in
  Printf.printf "{";
  let first = ref true in
  for i=0 to n-1 do
    if p.(i) = 1 && !first then
      (Printf.printf "%d" i; first := false)
    else if p.(i) = 1 then
      Printf.printf ", %d" i
  done; Printf.printf "}\n"
```

```
let enumere (n:int) : unit =
  (* initialisation de la partie vide *)
  let p = ref (Array.make n 0) in
  try
    while true do
      affiche_partie !p;
      p := plus_un !p
    done;
  with _ -> ()
```

Notons qu'il serait plus efficace de définir une fonction `plus_un_en_place (p:int array) : int` qui modifie `p` en place plutôt que créer un nouveau tableau à chaque itération.

⋆⋆⋆⋆⋆⋆⋆⋆

## 1.2 Conversion d'écriture romaine-décimale

On rappelle le système d'écriture des nombres utilisés par les romains. Il utilise les symboles I=1, V=5, X=10, L=50, C=100, D=500, M=1000. On rappelle quelques exemples dont le candidat pourra s'inspirer pour déduire les règles d'écriture des nombres :

(I, II, III, IV, V, VI, VII, VIII, IX, X) = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10).

XXXIX = 39, XL = 40.

(XLI, XLII, XLIII, XLIV, XLV, XLVI, XLVII, XLVIII, XLIX, L) = (41, 42, 43, 44, 45, 46, 47, 48 , 49, 50).

En particulier, on ne peut jamais avoir quatre symboles identiques consécutifs. Un nombre $n$ est supposé codé dans un tableau d'entiers s'il est écrit dans le système décimal, le chiffre le plus significatif étant en tête du tableau, et dans un tableau de caractères s'il est écrit dans le système romain.

Question 1

Quel est le plus grand nombre qui puisse être écrit ? Quel est le nombre dont l'écriture est la plus longue ?

Question 2

Donner un algorithme qui prend en entrée un nombre écrit dans le système romain et donne en sortie son écriture dans le système décimal usuel. Écrire la fonction correspondante.

Question 3

Donner un algorithme de conversion de décimal en romain. Écrire la fonction correspondante.

Question 4

Proposer un algorithme d'addition de deux nombres dans le système romain.

Question 5

Que pensez-vous du nombre moyen de caractères nécessaires pour écrire un nombre en romain ?

— CORRIGÉ —

Question 1