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

Le plus grand nombre ne peut avoir plus de trois M pour les milliers. Les centaines, dizaines et unités peuvent toutes valoir 9. La réponse est donc 3999, c'est-à-dire MMMCMXXCIX.

Le nombre le plus long doit aussi avoir trois M correspondant au chiffre des milliers, mais ensuite chaque chiffre 8 correspond à quatre caractères, le maximum possible. La réponse est donc 3888, c'est-à-dire MMMDCCCLXXXVIII qui comporte 15 caractères.

Question 2

Notons que chaque symbole correspond à une quantité précise qui doit être soit ajoutée, soit retranchée du total, suivant le symbole qui suit. On peut par exemple commencer par convertir chaque symbole en la quantité correspondante, puis calculer l'entier représenté en faisant une suite d'additions et de soustractions, puis convertir cet entier en tableau de chiffres.

```
let rom_to_dec (r:char array) : int array =
  let n = Array.length r in

  (* Création du tableau des valeurs *)
  let valeurs = Array.make n 0 in
  for i=0 to n-1 do
    valeurs.(i) <- match r.(i) with
      | 'I' -> 1
      | 'V' -> 5
      | 'X' -> 10
      | 'L' -> 50
      | 'C' -> 100
      | 'D' -> 500
      | _   -> 1000
  done;

  (* Conversion en entier *)
  let entier = ref valeurs.(n-1) in
  for i=0 to n-2 do
    if valeurs.(i) >= valeurs.(i+1)
```

```
    then entier := !entier + valeurs.(i)
  else entier := !entier - valeurs.(i)
done;

(* Conversion de l'entier en tableau de chiffres,
    poids faible en tête *)
let d = Array.make 4 0 in
let rec aux entier i =
  if entier <> 0 then (
    d.(i) <- entier mod 10;
    aux (entier/10) (i+1)
  )
in aux !entier 0;

(* Inversion du tableau pour mettre les poids forts en tête *)
let tmp = d.(0) in
d.(0) <- d.(3); d.(3) <- tmp;
let tmp = d.(1) in
d.(1) <- d.(2); d.(2) <- tmp;
d
```

Notons que les instructions de conversion d'entier en tableau de chiffres et d'inversion de l'ordre d'un tableau sont des exercices standards qui doivent être maîtrisés par les candidats. Une autre solution aurait consisté à lire le tableau de caractères romains, en séparant les parties utilisées pour le chiffre des milliers, celui des centaines, celui des dizaines et celui des unités. Les milliers sont faciles à convertir (il suffit de compter le nombre de M). Pour les autres chiffres, une même fonction peut être appliquée pour convertir les centaines, les dizaines ou les unités, en remplaçant M, D et C respectivement par C, L et X pour les dizaines et par X, V et I pour les unités. On obtiendrait ainsi une fonction de traduction directe sans passer par l'évaluation du nombre.

Question 3

L'algorithme naturel consiste à lire chaque décimale et à écrire la suite de caractères correspondante. On peut appeler une fonction auxiliaire `chiffre` qui fera le même travail pour les centaines, les dizaines et les unités, en remplaçant les symboles adéquatement. Soit k le chiffre à convertir, et j la première case libre du tableau.

On suppose que le nombre ici que le nombre à coder est inférieur à 3999 et donc qu'on peut prendre $|\mathtt{d}| = 4$.

```
let dec_to_rom (d:int array) : char array =
  let r = Array.make 15 '0' in
  let j = ref 0 in

  let chiffre (x:char) (v:char) (i:char) (k:int) : unit =
    match k with
      | 1 -> r.(!j) <- i; j := !j+1
      | 2 -> r.(!j) <- i; r.(!j+1) <- i; j := !j+2
```

```
      | 3 -> r.(!j) <- i; r.(!j+1) <- i; r.(!j+2) <- i; j := !j+3
      | 4 -> r.(!j) <- i; r.(!j+1) <- v; j := !j+2
      | 5 -> r.(!j) <- v; j := !j+1
      | 6 -> r.(!j) <- v; r.(!j+1) <- i; j := !j+2
      | 7 -> r.(!j) <- v; r.(!j+1) <- i; r.(!j+2) <- i; j := !j+3
      | 8 -> r.(!j) <- v; r.(!j+1) <- i; r.(!j+2) <- i; r.(!j+3) <- i;
             j := !j+4
      | 9 -> r.(!j) <- i; r.(!j+1) <- x; j := !j+2
      | _ -> ()
  in
  (* On traduit les chiffres un à un *)
  chiffre 'M' 'M' 'M' d.(0);
  chiffre 'M' 'D' 'C' d.(1);
  chiffre 'C' 'L' 'X' d.(2);
  chiffre 'X' 'V' 'I' d.(3);
  Array.sub r 0 !j
```

Question 4

Il n'existe en fait pas de méthode plus facile que de d'abord faire la conversion en entier. On va donc simplement utiliser les fonctions précédentes. On commence par définir une fonction `tab_to_int` qui permet de calculer la valeur d'un entier écrit en décimal :

```
let tab_to_int (d:int array) : int =
  let n = Array.length d in
  let entier = ref d.(0) in
  for i=1 to n-1 do
    entier := 10 * !entier + d.(i)
  done;
  !entier
```

Puis on obtient :

```
let add (r1:char array) (r2:char array) =
  (* Conversion en tableaux de chiffres *)
  let d1 = rom_to_dec r1 in
  let d2 = rom_to_dec r2 in
  (* Conversion en entier *)
  let n1 = tab_to_int d1 in
  let n2 = tab_to_int d2 in
  let somme = n1 + n2 in
  (* Conversion de l'entier somme en tableau de chiffres,
    poids faible en tête *)
  let d = Array.make 4 0 in
  let rec aux entier i =
    if entier <> 0 then (
      d.(i) <- entier mod 10;
      aux (entier/10) (i+1)
    )
```

```
in aux somme 0;

(* Inversion du tableau pour mettre les poids forts en tête *)
let tmp = d.(0) in
d.(0) <- d.(3); d.(3) <- tmp;
let tmp = d.(1) in
d.(1) <- d.(2); d.(2) <- tmp;

(* Conversion en caractères romains *)
dec_to_rom d
```

Question 5

Calculons la longueur moyenne d'un nombre compris entre 0 et 3999 quand il est écrit en romain.

Le chiffre des milliers prend 0, 1, 2 ou 3 caractères donc la longueur de sa traduction est : 0 pour les 1000 premiers nombres, 1 pour les 1000 suivants, 2 pour les 1000 suivants et 3 pour les 1000 suivants, soit en moyenne 2,5.

Le chiffre des centaines a une longueur moyenne de $(0+1+2+3+2+1+2+3+4+2)/10$, soit 2 caractères.

Le calcul est le même pour les dizaines et pour les unités. Un nombre s'écrit donc en moyenne avec $1,5 + 3 \times 2 = 7,5$ caractères. En revanche, un nombre écrit en notation décimale et compris entre 0 et 3999 utilise utilise 1 chiffre pour les 10 premiers nombres, 2 pour les 90 suivants, 3 pour les 900 suivants et 4 pour les 3000 suivants, soit en moyenne : $(10 \times 1 + 90 \times 2 + 900 \times 3 + 3000 \times 4)/4000 = 3,7225$.

Notons qu'on aurait pu ajouter une question subsidiaire plus difficile : proposer un algorithme qui teste si un tableau de caractères est bien une représentation valide d'un nombre en écriture romaine.

⋆⋆⋆⋆⋆⋆⋆⋆

1.3 Polynômes à trois variables

On veut représenter des polynômes à coefficients entiers sur trois variables X, Y et Z. Un monôme est un élément de la forme $aX^bY^cZ^d$ où a est un entier et b, c, d des entiers positifs ou nuls. L'entier a est le coefficient du monôme et le triplet (b, c, d) le degré du monôme. Un polynôme P est un ensemble non vide de monômes. Dans un polynôme, on peut regrouper les monômes de même degré en faisant la somme de leurs coefficients (bien sûr, si cette somme est nulle, il est sans intérêt de faire apparaître le monôme correspondant dans l'écriture du polynôme). Un polynôme est dit réduit s'il contient au plus un monôme de degré fixé.

Question 1

Modéliser la situation décrite et écrire un algorithme qui, étant donné un polynôme, calcule une représentation réduite de ce polynôme. Quel est le nombre de comparaisons entre monômes effectuées par votre algorithme ?

Question 2

Donner une manière de ranger les polynômes dans la modélisation d'un polynôme afin qu'il existe un algorithme résolvant la question 1 en effectuant au plus N comparaisons entre monômes, où N est le nombre de monômes. Donner cet algorithme.

Question 3

Un polynôme est dit symétrique si le monôme $aX^bY^cZ^d$ apparaît dans P si et seulement si les monômes $aX^bY^dZ^c$, $aX^cY^bZ^d$, $aX^cY^dZ^b$, $aX^dY^bZ^c$ et $aX^dY^cZ^b$ apparaissent. Écrire un algorithme qui, étant donné un polynôme P, indique si ce polynôme est symétrique ou non.

Question 4

Un monôme est dit unitaire si son coefficient est égal à 1. Donner un algorithme qui énumère tous les monômes unitaires de degré (b, c, d) tels que $b + c + d = k$ fixé.

CORRIGÉ

Question 1

Une modélisation simple consiste à représenter un monôme par un 4-tuple d'entiers et un polynôme par un tableau de monômes (l'utilisation de types structurés permettrait une modélisation plus élégante mais ces types ne sont pas au programme). Nous sommes ainsi amenés à faire les déclarations suivantes :

```
type monome = int * int * int * int
type polynome = monome array
```

Le principe de l'algorithme consiste à prendre un monôme dans le polynôme et à chercher tous les monômes de même degré. Il ne faut pas oublier « d'effacer » les monômes traités en mettant leurs coefficients à 0.

```
let reduit (p:polynome) : polynome =
  let n = Array.length p in
  let q = Array.copy p in

  let k = ref 0 in (* prochaine case de q à remplir *)
  for i=0 to n-1 do
    let a, b, c, d = q.(i) in
    if a <> 0 then
      let coef = ref a in
      (* Parcours des monômes p.(i+1)... *)
```

```
      (* pour chercher ceux de même degré que p.(i) *)
      for j=i+1 to n-1 do
        let a', b', c', d' = q.(j) in
        if (b, c, d) = (b', c', d') then (
          coef := !coef + a';
          q.(j) <- (0,b,c,d); (* on efface le monôme *)
        )
      done;
      if !coef <> 0 then (
        q.(!k) <- (!coef, b, c, d);
        incr k
      )
    done;
```

Dans le pire cas, tous les monômes du polynôme sont de degrés distincts et le nombre de comparaisons entre monômes est $\frac{N(N-1)}{2}$.

Question 2

Pour n'avoir qu'au plus N comparaisons à effectuer, il suffit que les monômes de même degré soient rangés consécutivement dans le polynôme. C'est le cas par exemple dès qu'on définit un ordre sur le degré des monômes et que l'on range les monômes selon cet ordre. Sous cette hypothèse, l'algorithme devient alors :

```
let reduit_quand_trie (p:polynome) : polynome =
  let n = Array.length p in
  let q = Array.copy p in

  let k = ref 0 in (* prochaine case de q à remplir *)
  let coef = ref 0 in (* coefficient du monôme en cours de calcul *)
  let idx = ref 0 in (* p.(!idx) est le monôme en cours de calcul *)
  for i=0 to n-1 do
    let a, b, c, d = q.(!idx) in (* monôme courant *)
    let a', b', c', d' = q.(i) in
    if a <> 0 then
      let meme_deg = (b, c, d) = (b', c', d') in
      if meme_deg then
        coef := !coef + a'
      else (    (* On a trouvé un monôme de degré différent *)
        if !coef <> 0 then (
          q.(!k) <- (!coef, b, c, d);
          incr k
        );
        (* On met à jour les variables relatives au monôme *)
        idx := i;
        let c, _, _, _ = q.(i) in
        coef := c
      );
      (* On ajoute les monômes du degré restant *)
```

```
      if i=n-1 && !coef <> 0 then (
        q.(!k) <- (!coef, b', c', d');
        incr k
      )
  done;
  Array.sub q 0 !k
```

Question 3

On va supposer que le polynôme considéré est réduit. On va prendre successivement les monômes et aller vérifier que tous les monômes obtenus par permutation du degré sont présents. On mettra à 0 les coefficients de tous les monômes traités.

```
let meme_degree_permutation (p:monome) (q:monome) : bool =
  let _, b, c, d = p in
  let _, b', c', d' = q in
  (b=b' && c=c' && d=d') || (b=b' && c=d' && d=c')
  || (b=c' && c=b' && d=d') || (b=c' && c=d' && d=b')
  || (b=d' && c=b' && d=c') || (b=d' && c=c' && d=b')

let est_symetrique (p:polynome) : bool =
  let n = Array.length p in
  let p = Array.copy p in
  try
    for i=0 to n-1 do
      let a, b, c, d = p.(i) in
      if a <> 0 then
        let nb_sym  = ref 0 in
        (* Parcours des monômes p.(i+1)... pour chercher ceux *)
        (* de degré égal à p.(i) à une permutation près *)
        for j=i+1 to n-1 do
          let a', b', c', d' = p.(j) in
          if a' <> 0 && meme_degree_permutation p.(i) p.(j) then (
            if a' = a then (
              incr nb_sym;
              p.(j) <- (0, b', c', d')
            ) else raise Exit
          )
        done;
        if !nb_sym <> 5 then raise Exit
    done;
    true
  with Exit -> false
```

Question 4

Cette question ne présente aucune difficulté particulière. Il suffit d'énumérer tous les monômes de coefficient 1 à l'aide de deux boucles « for » convenablement imbriquées.

```
let enumere (k:int) : unit =
  for i=0 to k do
    for j=0 to k-i do
      Printf.printf "X^%d Y^%d Z^%d\n" i j (k-i-j)
    done
  done
```

⋆⋆⋆⋆⋆⋆⋆⋆

1.4 Produit modulaire

Nous nous intéressons au produit modulaire de grands nombres. Cette opération est très utilisée en cryptographie.

Les nombres utilisés en cryptographie sont de grands entiers (environ mille chiffres binaires). Nous devons écrire des fonctions pour de grands entiers.

Question 1

Définir une structure de données pour ces grands entiers (en les considérant par exemple dans une grande base r : $A = \sum_{i=0}^{n} a_i r^i$ avec r une puissance de 2 judicieusement choisie).

Écrire une fonction d'addition.

Écrire une fonction de multiplication par un « chiffre » (dans la base r).

Nous présentons ici un algorithme dû à Peter Montgomery en 1985. La valeur du modulo M ne varie pas, certaines valeurs pourront être considérées comme pré-calculées.

Considérons l'algorithme suivant où A, B, Q, S et M sont de grands entiers, r représente la base.

`calcul`(A, B, M, S)
    $S \leftarrow 0$
    pour $i = 0$ à n faire
        $q_i \leftarrow (s_0 + a_i \times b_0)(r - m_0)^{-1} \mod r$
        $S \leftarrow S + a_i \times B + q_i \times M$
        $S \leftarrow S \div r$

où $(r - m_0)^{-1} \times m_0 \equiv -1 \bmod r$.

Question 2

Que peut-on dire de $S + a_i \times B + q_i \times M$? Que vaut S à la fin de l'exécution ? Quel est son ordre de grandeur si A est plus petit que $2 \times M$, B plus petit que M et M plus petit que r^n ?

Que peut-on dire de la division par r ?

Écrire la fonction OCaml correspondant à l'algorithme présenté.

Question 3

En déduire un algorithme évaluant le produit de $A \times B \bmod M$.

CORRIGÉ

Question 1

Nous proposons de représenter les nombres dans un tableau d'entiers machine où l'élément d'indice i est le chiffre de poids r^i de l'écriture en base r. Pour le choix de la base, nous considérons que les entiers machine ont au plus 15 chiffres binaires. Les opérations de base sont l'addition de deux chiffres et le produit de deux chiffres. Nous proposons donc de prendre $r = 2^7$.

Les deux fonctions demandées peuvent s'écrire :

```
type nombre = int array
let base = 128 (* 2^7 *)

let addition (a:nombre) (b:nombre) : nombre =
  let n = Array.length a in
  let s = Array.make n 0 in

  let retenue = ref 0 in
  for i=0 to n-1 do
    s.(i) <- a.(i) + b.(i) + !retenue;
    retenue := s.(i) / base;
    s.(i) <- s.(i) mod base
  done;
  if !retenue <> 0 then
    failwith "dépassement d'entier";
  s

let multiplication_chiffre (a:nombre) (c:int) : nombre =
  let n = Array.length a in
  let p = Array.make n 0 in

  let retenue = ref 0 in
  for i=0 to n-1 do
    p.(i) <- c * a.(i) + !retenue;
    retenue := p.(i) / base;
    p.(i) <- p.(i) mod base
  done;
  if !retenue <> 0 then
    failwith "dépassement d'entier";
  p
```

Question 2

À chaque itération, q_i est tel que $S + a_i \times B + q_i \times M$ est un multiple de r.

En fin d'exécution, $S = \frac{A \times B + Q \times M}{r^{n+1}}$.

À chaque itération, S est inférieur à $3 \times M$ vu les valeurs prises A et B. À la dernière itération, S est inférieur à $2 \times M$.

Le choix de la taille des tableaux n doit tenir compte du fait que S est au cours de l'itération de l'ordre de $3 \times r \times M$.

La division par r est un simple décalage.

Nous initialisons `m'` à $(r - m_0)^{-1} \bmod r$ et `r_carre` à $(r^{n+1})^2 \bmod M$, ce sont toutes les deux des variables de type `nombre`. Nous prenons par exemple $M = 123 \times r + 11$.

```
let n = 5
let m = [|11; 123; 0; 0; 0|]
let m' = 93
let r_carre = [|84; 85; 0; 0; 0|]

let multiplication_montg (a:nombre) (b:nombre) : nombre =
  let n = Array.length a in
  let s = ref [||] in

  for i=0 to n-1 do
    let t = multiplication_chiffre b a.(i) in
    s := addition !s t;
    let q = (!s.(0) * m') mod base in
    let t = multiplication_chiffre m q in
    s := addition !s t;
    (* division par r (décalage) *)
    for j=0 to n-2 do
      !s.(j) <- !s.(j+1)
    done;
    !s.(n-1) <- 0
  done;
  !s
```

Question 3

Pour évaluer le produit $A \times B \bmod M$, il suffit d'utiliser deux fois de suite l'algorithme de Montgomery. Au premier passage, nous évaluons :

$$C := A \times B \times (r^{n+1})^{-1} \bmod M(? + M)$$

avec $C < 2M$.

Au deuxième passage, nous obtenons :

$$C := C \times (r^{n+1})^2 \times (r^{n+1})^{-1} \bmod M(? + M) = A \times B \bmod M(? + M)$$

avec $C < 2M$.

On a donc $0 \leq C < 2M$. Si $C \geq M$, on effectue une soustraction pour se ramener à $0 \leq C < M$.

```
let multiplication (a:nombre) (b:nombre) : nombre =
  let c = multiplication_montg a b in
  let c = multiplication_montg c r_carre in

  (* On regarde si c < m *)
  try
    for i=n-1 downto 0 do
      if c.(i) < m.(i) then
        raise Exit
    done;
    let m' = multiplication_chiffre m (-1) in
    addition c m'
  with Exit -> c
```

$\star\star\star\star\star\star\star\star$