Introduction à l'informatique théorique – 2005 devoir 11

Alban Guyon

May 23, 2022

1 Questions

1 Pour représenter une liste de nombres entiers en un seul nombre entier on utilise la notation suivante:

$$[a_1, a_2, a_3, ..., a_n, 0, 0, 0, ...] = 2^{a_1} * 3^{a_2} * 5^{a_3} * ... * p^{a_n} \tag{1}$$

où les bases des exponents sont les nombres premiers.

Du fait que chaque entier positif a une factorisation unique en nombres premiers, chaque liste sera associé à un nombre unique supérieur ou égal à 1.

Une autre chose à noter est que toutes les listes sont infinis: une liste avec n éléments à réellement n éléments et une infinité de 0 qui suivent. En effet, les 0 à la fin d'une liste sont ignorés. Par exemple, les listes [2, 3, 0, 3, 2, 0, 0] et [2, 3, 0, 3, 2] sont équivalantes.

Dans le pseudo-code, on utilise la notation P_i pour représenter le i-ème nombre premier. Voici quelques fonctions sur les listes dans cette nouvelle représentation:

Algorithm 1 Vide

```
procedure VIDE()
    r_0 ← 1
```

Algorithm 2 EstVide?

```
procedure ESTVIDE?(r_1)
    if r_1 = 1 then
        r_0 ← TRUE
    else
        r_0 ← FALSE
```

Algorithm 3 Dans?

```
procedure DANS?(r_1, r_2)
    r_1 ← FALSE
    for i ← 0 to r_1 do        ▷ on utilise r_1 pour la limite max car taille de la liste est forcément inférieure
        if r_1 mod P_i^{r_2} = 0 and not r_0 then
            N ← n/P_i^{r_2}        ▷ on enlève r_1 fois le même facteur premier
            if N mod (P_i^{r_2}) ≠ 0 then        ▷ on s'assure qu'il n'y a pas d'autre diviseurs
                r_1 ← TRUE
```

Algorithm 4 Card

```
procedure CARD(r_1)
    r_0 ← 0
    for i ← 0 to r_1 do
        if n mod (P_i^{r_1}) = 0 then
            r_0 ← i + 1        ▷ l'indice du dernier element non nul + 1 est la taille
```

Algorithm 5 Ajouter

IND

procedure AJOUTER(r_2, r_1)
  $SIZE \leftarrow Card(r_1)$
  $r_0 \leftarrow r_1 * P_{SIZE}^{r_2}$

Pour la fonction Retirer, je vais d'abord définir les macos auxiliaires $IndexDe(r_2, r_1)$ et $index(r_2, r_1)$:
$IndexDe(r_2, r_1)$ retourne l'indice de l'élément r_2 dans la liste r_1.
$index(r_2, r_1)$ retourne l'élément à l'indice r_2 dans la liste r_1

Algorithm 6 IndexDe

procedure INDEXDE(r_2, r_1)
  $r_0 \leftarrow r_1$ ▷ on initialise l'indice à r_1 qui signifie que l'élément n'est pas dans la liste
  $FOUND \leftarrow$ FALSE
  for $i \leftarrow 0$ **to** r_1 **do**
    if not FOUND **then**
      $COPY \leftarrow r_1$
      $FLAG \leftarrow$ TRUE ▷ Pouvait-on enlever r_2 instaces de P_i?
      for $j \leftarrow 0$ **to** r_2 **do**
        if $COPY \mod P_j = 0$ **then**
          $COPY \leftarrow COPY/P_j$
        else
          $FLAG \leftarrow$ FALSE
      if $FLAG$ **and** $COPY \mod P_i \neq 0$ **then** ▷ Si P_i n'est plus un facteur de $COPY$
        $FOUND \leftarrow$ TRUE
        $r_0 \leftarrow i$

Algorithm 7 Index

procedure INDEX(r_2, r_1)
  $r_0 \leftarrow 0$
  $COPY \leftarrow r_1$
  for $i \leftarrow 0$ **to** r_1 **do** ▷ On compte le nombre de fois que l'on peut enlever P_{r_2} de r_1
    if $COPY \mod P_{r_2} = 0$ **then**
      $COPY \leftarrow COPY/P_{r_2}$
      $r_0 \leftarrow r_0 + 1$

On peut maintenant utiliser ces macros dans la macro Retirer:

Algorithm 8 Retirer

procedure RETIRER(r_2, r_1)
  $r_0 \leftarrow r_1$ ▷ Par défaut, on retourne juste la liste sans modification
  $IND \leftarrow IndexDe(r_2,\ r_1)$
  if $IND \neq r_1$ **then** ▷ Si l'élément est dans la liste
    $COPY \leftarrow r_1/P_{IND}^{r_2}$ ▷ On met l'élément que l'on veut enlever à 0
    for $i \leftarrow 0$ **to** r_1 **do** ▷ On décale toutes les valeurs suivantes un indice à gauche
      $VAL \leftarrow Index(i + 1,\ COPY)$
      $COPY \leftarrow COPY/P_{i+1}^{VAL}$
      $COPY \leftarrow COPY * P_i^{VAL}$
    $r_0 \leftarrow COPY$

Algorithm 9 Inter

```
procedure INTER(r_1, r_2)
    r_0 ← 1
    for i ← 0 to r_1 do
        VAL ← Index(i, r_1)
        if Dans?(r_1, VAL) and not Dans?(L, VAL) then    ▷ Si l'élément est dans les deux listes et on l'a pas déjà écrit
            r_0 ← Ajouter(VAL, r_0)
```

Pour représenter des ensembles d'ensembles, on peut utiliser une variante de l'algorithme précédent: chaque liste interne est d'abord remplacée par son nombre, et on obtiens une liste simple. Il suffit ensuite de calculer le nombre représentant la liste externe.

Pour décoder, on décode le nombre représentant la liste externe, puis on décode chaque liste interne.

2 Montrez la croissance de $B_4(x)$ en fonction de x.
Etape de base pour x = 1

$$B_4(1) = B_3(B_3(1)) = B_3(2^4 - 3) = B_3(2^{2^2} - 3) = 2^{2^{2^2}} - 3 \tag{2}$$

Etape general pour x = x

$$B_4(x) = B_3(B_3(...B_3(1))) = B_3(B_3(...B_3(2^2 - 3))) \tag{3}$$

avec x + 1 fois B_3
Etape d'induction pour x = x + 1

$$B_4(x+1) = B_3(B_3(...B_3(2^{2^2} - 3))) \tag{4}$$

avec x + 1 fois B_3

On voit que chaque $B_3(x)$ ajoute un étage de plus à la tour de puissance de 2 dans l'argument. On peut donc confirmer qu'il y a x + 3 étages de 2 dans la tour de puissance de 2.

3 Le programme qui peut faire augmenter la variable "a" le plus rapidement est le suivant:

Algorithm 10 Fast

```
procedure FAST()
    inc(a)
    inc(a)
    inc(a)
    inc(a)                                                    ▷ a = 4
    for i ← 0 to a do      ▷ Comme a > 3, ca vaut le coup d'utiliser 3 lignes pour incrémenter a par plus que 3
        inc(a)
                                                              ▷ a = 8
    for i ← 0 to a do
        inc(a)
                                                              ▷ a = 16
    for i ← 0 to a do
        inc(a)
                                                              ▷ a = 32
    for i ← 0 to a do
        inc(a)
    ...
```

On voit bien que la variable "a" à une croissance exponentielle de base 2. Le maximum que l'on peut augmenter la variable est de la doubler toutes les 3 lignes. En effet, après l lignes, "a" ne peut pas excéder $2^{(l/3)}$. Cela veut dire qu'un programme REPETERPASTROP ne peut jamais calculer n^k si k est supérieur au nombre de lignes divisé par 3.

4 Le fait de remplacer l'opération d'assignation par l'opération d'interchangement ne change pase la puissance de calcul L'opération d'assignation peut être remplacé par la macro suivante:

Algorithm 11 assign

```
procedure ASSIGN(A, B)
    a ←→ c
    for 0 to b do
        inc(a)
```

La valeur c n'a pas été assigne donc il est égal à 0. De plus, on choisit un registre different pour c à chaque fois que l'on veut utiliser la macro. Comme on a un nombre illimité de registres, on peut utiliser la macro pour toutes les opérations d'assignation. Le nouveau programme sera forcément plus long que le programme original mais il ne sera pas moins puissant.

5 La boucle POUR n'est pas plus puissante que la boucle REPETER car on peut utiliser une boucle REPETER pour simuler une boucle POUR de la forme:
POUR i ← a à b faire $< bloc >$ suivant

Algorithm 12 PourSimulation

```
procedure PourSimulation
    i ← a
    rep ← MOINS(b, a)          ▷ On utilise la macro MOINS pour être sur que le resultat est ≥ 0
    for 0 to rep do
        < bloc >
        i ← i + 1
```

Comme on peut remplacer toutes les boucles POUR par une boucle REPETER sans perdre de l'information, on déduit que la boucle POUR n'est pas plus puissante que la boucle REPETER, ce n'est simplement qu'un sucre syntaxique.