IFT2015 automne 2012 — Devoir 1

Miklós Csűrös

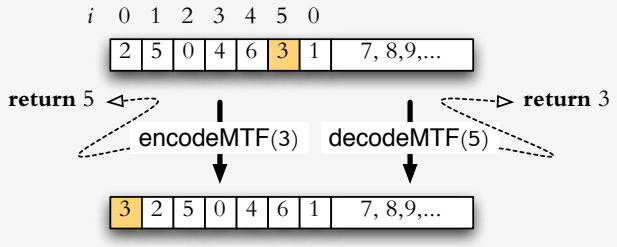
24 octobre 2012

1 Encodage de séquences d'entiers (30+5 points)

À remettre avant 20 :15 le 1er novembre. Remettez un rapport écrit par courriel (à csuros@iro...) en format PDF. Travaillez seul.

1.1 Move-to-front

La transformation **move-to-front** (MTF) sert à transmettre une séquence d'entiers. Pour encoder une séquence $n_0, n_1, n_2, \ldots$ d'entiers non-négatifs ($n_i = 0, 1, 2, \ldots$), on maintient une permutation π de tous les nombres naturels. Initialement, on a identité : $\pi[n] = n$ pour tout n.



Pour transmettre le prochain entier n, on identifie son unique position i dans la permutation ($\pi[i] = n$), et on émet i. En même temps, on performe une rotation circulaire de $\pi[0..i]$ qui place n en position 0. Dans la permutation résultante π', on a $\pi'[0] = \pi[i]$, $\pi'[j] = \pi[j-1]$ pour $j = 1, 2, \ldots, i$ et $\pi'[j] = \pi[j]$ pour tout $j > i$.
Pour décoder, on fait la même rotation sauf que l'argument est l'indice i, et on doit retourner $\pi[i]$ (avant la rotation).

Exemple : la transformation MTF de la liste $(1, 2, 5, 2, 1, 4, 3, 0)$ donne $(1, 2, 5, 1, 2, 5, 5, 5)$:

i	n_i	encodeMTF(n_i)	π (après rotation)
0	1	1	$(1\ \ 0\ \ 2 \cdots \infty)$
1	2	2	$(2\ \ 1\ \ 0\ \ 3 \cdots \infty)$
2	5	5	$(5\ \ 2\ \ 1\ \ 0\ \ 3\ \ 4\ \ 6 \cdots \infty)$
3	2	1	$(2\ \ 5\ \ 1\ \ 0\ \ 3\ \ 4\ \ 6 \cdots \infty)$
4	1	2	$(1\ \ 2\ \ 5\ \ 0\ \ 3\ \ 4\ \ 6 \cdots \infty)$
5	4	5	$(4\ \ 1\ \ 2\ \ 5\ \ 0\ \ 3\ \ 6 \cdots \infty)$
5	3	5	$(3\ \ 4\ \ 1\ \ 2\ \ 5\ \ 0\ \ 6 \cdots \infty)$
6	0	5	$(0\ \ 3\ \ 4\ \ 1\ \ 2\ \ 5\ \ 6 \cdots \infty)$

1.2 Comment encoder un nombre ?

Supposons qu'on veut transmettre une séquence de nombres naturels $n_0, n_1, n_2, \ldots$ ($n_i \geq 0$). La difficulté est qu'on ne peut utiliser que deux symboles, comme en code Morse : 0 et 1. (Il s'agit de l'abstraction d'un fichier binaire.)
On définit donc un code $\rho(n)$ (séquence de symboles de 0 et 1) pour tout n, et on transmet les codes $\rho(n_0)$, $\rho(n_1)$, …, l'un après l'autre. Le code doit permettre le décodage sans ambiguïté après concaténation. Le code devrait être économique et utiliser aussi peu de symboles pour encoder n que possible.

Encodage unaire. L'exemple le plus simple est le *code unaire* $\alpha(n)$:

$$\alpha(n) = \underbrace{1\,1\ldots1}_{n \text{ fois}}\,0 \tag{1.1}$$

Clairement, il est possible de décoder une séquence de codes concaténés $\alpha(n_0)$, $\alpha(n_1)$, …, car il suffit de compter les 1s jusqu'au permier 0.

Encodage binaire. L'encodage binaire standard $\beta(n)$ est beaucoup plus efficace, parce qu'il utilise seulement $d = \lceil \lg(n+1) \rceil$ symboles, où d est le plus petit entier non-négatif tel que $n < 2^d$. Pour $n = \sum_{i=0}^{d-1} x_i \cdot 2^{(d-1)-i}$ (avec $x_i = 0, 1$) on transmet la séquence $x_{d-1}, x_{d-2}, \ldots$. En particulier $\beta(0) = \varepsilon$ (chaîne vide), $\beta(1) = 1$, $\beta(2) = 1\,0$, etc.

La concaténation du code binaire est ambigu : $(1, 2, 1)$ et $(6, 1)$ mènent à la même séquence 1 1 0 1. Choisir une taille d assez large et encoder chaque n_i sur d bits est possible pour éviter l'ambiguité (p.e., $(1, 2, 1) \mapsto$ 0 0 1 0 1 0 0 0 1, et $(6, 1) \mapsto$ 1 1 0 0 0 1 avec $d = 3$) mais on gaspille d'espace quand beaucoup des éléments sont petits. En plus, un tel choix d n'est possible que si on connaît la séquence au début.

Encodage γ. On peut construire un code universel en préfixant $\beta(n)$ par sa longueur encodé en *unaire* : si $\beta(n)$ comprend d symboles, alors on écrit d fois 1, suivi par un seul 0, et l'encodage binaire sur d bits.

$$\gamma(n) = \alpha\Big(\big|\beta(n)\big|\Big) \oplus \beta(n), \tag{1.2}$$

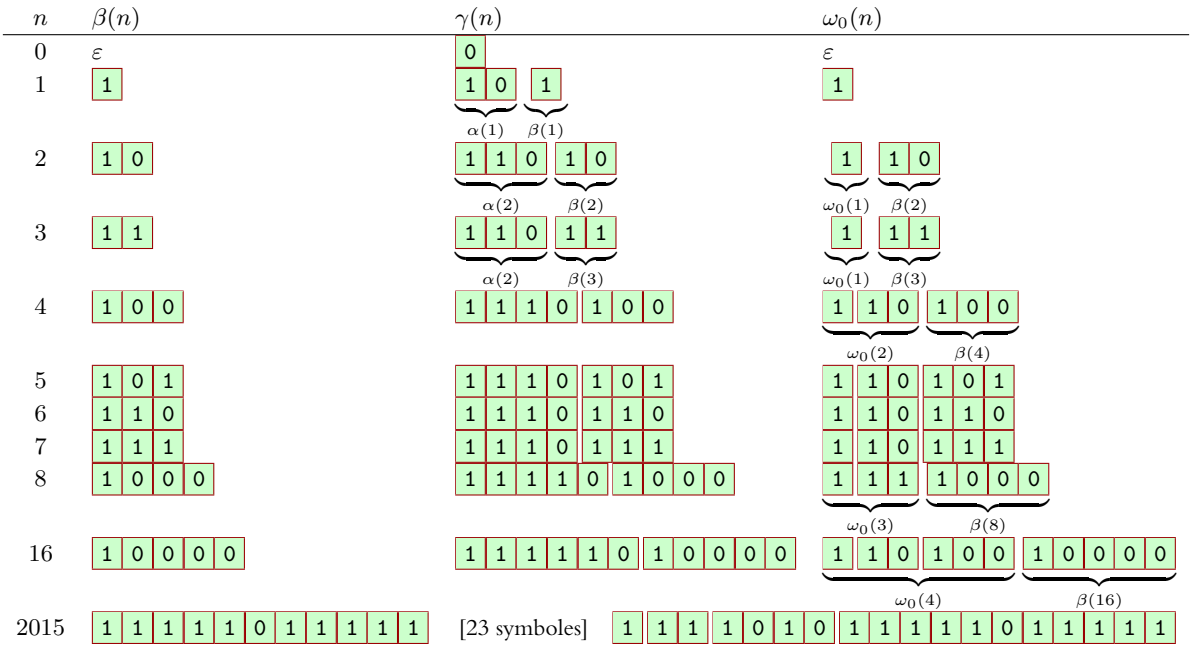
où $\oplus$ dénote la concaténation. On peut décoder la liste à partir des codes $\gamma(n_i)$ concaténés : compter les 1s au début ($= d$), jusqu'au premier 0, et décoder les d bits suivants contenant $\beta(n_i)$.

Encodage ω. On peut améliorer le code γ en spécifiant le nombre de bits avec un encodage plus efficace que l'unaire. On se sert de la récursivité :

$$\omega_0(n) = \begin{cases} \varepsilon & \text{vide si } n = 0 \\ \omega_0\Big(\big|\beta(n)\big| - 1\Big) \oplus \beta(n) & \{n > 0\} \end{cases} \tag{1.3}$$

Pour transmettre un entier n, on utilise le code ajusté $\omega_1(n) = \omega_0(n) \oplus 0$, et ainsi on peut décoder une liste de codes $\omega_1(n_i)$ sans problème.

Quelques exemples :

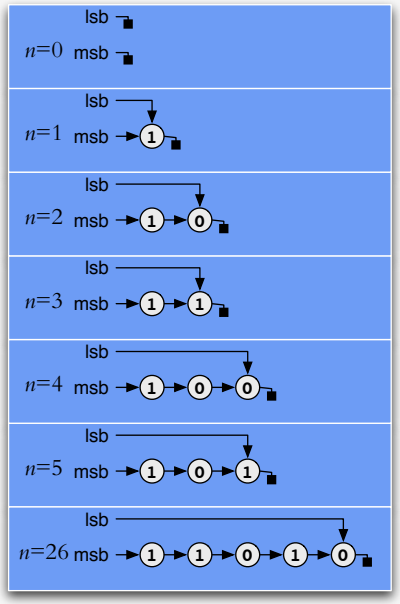


1.3 Problèmes

Move-to-front. (15 points) ► Développez une structure de données qui implante le type abstrait avec opérations encodeMTF et decodeMTF. Montrez l'implantation des opérations avec tous les détails.

Indice: *Pour stocker la permutation actuelle π, on peut utiliser une liste chaînée ou un tableau (avec expansion dynamique !). Notez qu'il n'est pas nécessaire de stocker π explicitement (cela serait de longueur ∞), mais seulement élargir la liste des éléments stockés jusqu'au moins i quand on doit accéder à $\pi[i]$ pas encore sur la liste.*

Encodage universel. Dans les problèmes suivants, une liste chaînée sert à stocker l'encodage d'un nombre naturel.



On veut une structure de données pour représenter des entiers non-négatifs de grandeur arbitraire. Un nœud x de la liste chaînée contient donc les variables x.next pour le prochain nœud (= null à la fin), et x.bit pour le bit même (qui prend la valeur 0 ou 1 pour dénoter 0 ou 1).
La tête de la liste (msb) est le nœud pour le bit de poids fort. On stocke aussi une référence au dernier nœud par la variable lsb (*least significant bit* — bit de poids faible). $n = 0$ correspond à la liste vide.
Les algorithmes d'encodage doivent retourner la paire (msb, lsb) pour la liste représentant le code.

a. (6 points) ► Donner un algorithme *récursif* encodeBeta(n) pour construire l'encodage binaire d'un nombre naturel n.

b. (2 points) ► Combien de symboles doit on utiliser pour encoder 10^{100} (un *googol*) dans les encodages différents (α, β, γ et ω_1) ?

c. (7 points) ► Donner un algorithme *récursif* encodeOmega(n) pour construire l'encodage $\omega_1(n)$.

d. (5 points boni) ► Donner un algorithme pour décoder une liste à partir de son encodage ω_1. L'entrée de l'algorithme est la tête (msb pour le premier élément) d'une liste de tous les codes concaténés. L'algorithme doit afficher les nombres n_i qui sont encodés sur la liste. Exemple : si la liste contient 1 1 0 1 1 1 0 1 1 0 0 0, l'algorithme doit afficher $7, 2, 0$.

correction le 10/24