IFT2015 automne 2013 — Devoir 1

Miklós Csűrös

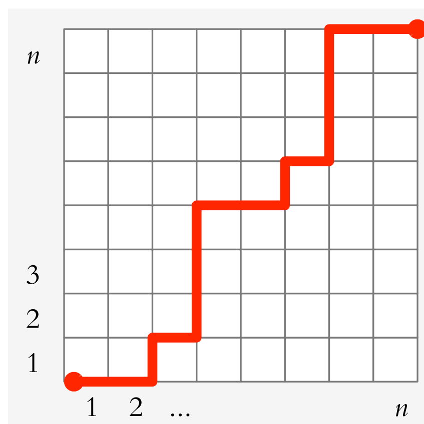
5 septembre 2013

À remettre avant 20 :15 le 12 septembre. Remettez un rapport écrit par email (à csuros@iro...) en format PDF. (La taille du fichier ne doit pas dépasser 2^{20} octets.)

Vous avez le droit de travailler en un équipe de 2 sur problème 3.d et le soumettre plus tard, avant 20 :15 le 16 septembre. Travaillez seul sur tous les autres problèmes.

1.1 Chemins sur une grille (15 points)

On veut compter les chemins sur une grille $n \times n$.



Un chemin sur la grille $n \times n$ est une séquence $(x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m)$ avec $(x_0, y_0) = (0, 0)$, $(x_m, y_m) = (n, n)$, et $(x_i, y_i) \in \{(x_{i-1}+1, y_{i-1}), (x_{i-1}, y_{i-1}+1)\}$ pour tout $i = 1, 2, \ldots, m$. (Dans d'autres mots, on peut déplacer seulement vers le droit ou vers le haut par une case.)

a. (5 points) ►Démontrer que le nombre de chemins $C(n)$ est

$$C(n) = \binom{2n}{n} = \frac{(2n)!}{(n!) \cdot (n!)}.$$

b. (10 points) ►Utiliser la formule de Stirling pour caractériser la croissance de $\ln C(n)$ par une fonction f (aussi simple que possible) t. q. $\ln C(n) \sim f(n)$ ou $\lim_{n\to\infty} \frac{\ln C(n)}{f(n)} = 1$.

1.2 Comment encoder les entiers ? (35+15 points)

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 : $\boxed{0}$ et $\boxed{1}$. (Il s'agit de l'abstraction d'un fichier binaire.)
On définit donc un code $\rho(n)$ (séquence de symboles de $\boxed{0}$ et $\boxed{1}$) pour tout n, et on transmet les codes $\rho(n_0)$, $\rho(n_1)$, $\ldots$, l'un après l'autre. En plus d'être *déchiffrable* (on peut récupérer la séquence n_0, n_1, $\ldots$ sans ambiguïté à partir de la chaîne concaténée), ρ devrait être *instantané* : on veut déchiffrer n_i dès qu'on reçoit tous les bits de $\rho(n_i)$.

Encodage unaire. L'exemple le plus simple est le *code unaire* $\mathsf{u}(n)$:

$$\mathsf{u}(n) = \underbrace{\boxed{1}\boxed{1}\ldots\boxed{1}}_{n \text{ fois}}\boxed{0} \tag{1.1}$$

Clairement, il est possible de décoder une séquence de codes concaténés $\mathsf{u}(n_0)$, $\mathsf{u}(n_1)$, $\ldots$, car il suffit de compter les $\boxed{1}$s jusqu'au premier $\boxed{0}$. Mais ce code n'est pas économique de tout …

Encodage binaire. L'encodage binaire standard $\mathsf{b}(n)$ est beaucoup plus efficace, parce qu'il utilise seulement d symboles où d est le plus petit nombre naturel tel que $n < 2^d$. Pour $n = \sum_{i=0}^{d-1} x_i \cdot 2^i$ (avec $x_i = 0, 1$) on transmet la séquence $x_{d-1}, x_{d-2}, \ldots, x_0$, en commençant par le bit de poids fort. En particulier $\mathsf{b}(0) = \lambda$ (chaîne vide), $\mathsf{b}(1) = \boxed{1}$, $\mathsf{b}(2) = \boxed{1}\boxed{0}$, etc.

Par contre, b n'est pas déchiffrable : $(1, 2, 1)$ et $(6, 1)$ mènent à la même séquence $\boxed{1}\boxed{1}\boxed{0}\boxed{1}$.

Encodage f. On peut construire un code instantané en préfixant $\mathsf{b}(n)$ par sa longueur encodé en unaire : si $\mathsf{b}(n)$ comprend d symboles, alors on écrit d fois $\boxed{1}$, suivi par un seul $\boxed{0}$, et l'encodage binaire sur d bits.

$$\mathsf{f}(n) = \mathsf{u}\Big(\big|\mathsf{b}(n)\big|\Big) \oplus \mathsf{b}(n), \tag{1.2}$$

où $\oplus$ dénote la concaténation et $|\ldots|$ la longueur en bits. Le code f est instantanément déchiffrable : compter les $\boxed{1}$s au début ($= d$), jusqu'au premier $\boxed{0}$, et décoder les d bits suivants contenant $\mathsf{b}(n_i)$.

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

$$\mathsf{g}_0(n) = \begin{cases} \lambda & \text{chaîne de longueur } 0 \text{ quand } n = 0 \\ \mathsf{g}_0\Big(\big|\mathsf{b}(n)\big| - 1\Big) \oplus \mathsf{b}(n) & \{n > 0\} \end{cases} \tag{1.3}$$

On rend le code déchiffrable par un bit $\boxed{0}$ apposé : on transmet $\mathsf{g}_1(n) = \mathsf{g}_0(n) \oplus \boxed{0}$. Par exemple, $\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{0}\boxed{0}$ encode la séquence $7, 2, 0$.

n	$\mathsf{b}(n)$	$\mathsf{f}(n)$	$\mathsf{g}_0(n)$
0	λ	0	λ
1	1	10 1	1
		u(1) b(1)	
2	10	110 10	1 10
		u(2) b(2)	$\mathsf{g}_0(1)$ b(2)
3	11	110 11	1 11
		u(2) b(3)	$\mathsf{g}_0(1)$ b(3)
4	100	1110 100	1 10 100
			$\mathsf{g}_0(2)$ b(4)
5	101	1110 101	1 10 101
6	110	1110 110	1 10 110
7	111	1110 111	1 10 111
8	1000	11110 1000	1 11 1000
			$\mathsf{g}_0(3)$ b(8)
16	10000	111110 10000	1 10 100 10000
			$\mathsf{g}_0(4)$ b(16)
2015	11111011111	[23 symboles]	1 11 1010 11111011111

a. (9 points) ▶ Combien de symboles doit on utiliser pour écrire 10^{100} (un *googol*) dans les encodages différents u, b, f et g_1 ? ▶ En général, quelle est la longueur exacte de u, b, et f en fonction de n ?

b. (13 points) ▶ Donner un algorithme *récursif* encodeBeta(n) qui affiche l'encodage b d'un nombre naturel n.

c. (13 points) ▶ Donner un algorithme *récursif* encodeOmega(n) qui émet l'encodage $\mathsf{g}_1(n)$. (Vous avez le droit d'appeler encodeBeta de **b**.)

d. ($15^{\heartsuit}$ points boni) Étudiez l'efficacité de la méthode de compression suivante qu'on va appeler `intZip`. Prendre un fichier spécifié à l'entrée, et compter la fréquence de chaque symbole[1] dedans. Encoder le symbole le plus fréquent par $\mathsf{g}_1(0)$, le deuxième plus fréquent par $\mathsf{g}_1(1)$, etc. : le k-ème plus fréquent par $\mathsf{g}_1(k-1)$ en général. Le fichier à la sortie est la séquence de g_1 pour les symboles de l'entrée.

Dans votre étude, prenez quelques fichiers «typiques» dans un contexte quelconque de taille $\geq$ 1Mb. Vous pouvez vous servir de la collection de données à `http://introcs.cs.princeton.edu/java/data/` ou n'importe quel autre jeu de données disponible sur l'Internet. Comparez la longueur de compression `intZip` avec la taille du fichier comprimé par outils populaires (p.e., `gzip`, `bzip`) de votre choix. Soumettez votre code pour calculer la longueur de compression (notez qu'il n'est pas nécessaire de construire le code g) en un seul archive JAR, et un rapport de vos expériences en un fichier PDF.

> REMARQUE. Lire le fichier deux fois comme ici n'est pas pratique. Les méthodes de compression qui utilisent l'encodage f ou g les appliquent à une séquence d'entiers produites «à la volée», en lisant le fichier de l'entrée. La clé est de choisir un heuristique qui transforme la séquence d'entrée en entiers typiquement petits pour exploiter les propriétés de g.

[1]Choisissez ce que c'est un «symbole» selon le contexte. Les caractères (`char`) ou les octets (`byte`) du fichier peuvent suffire, mais parfois (ADN, musique, etc.) c'est mieux de définir un symbole comme un bloc de caractères consécutifs.