# Théorie de l'information

M1 Informatique 2012-2013

## Fiche de TD 2

**Exercice 1. (Transformée de Burrows-Wheeler)**

1. Calculer la transformée de Burrows-Wheeler de `baabcaab`.
2. Calculer la transformée de Burrows-Wheeler de `TENTATION`.
3. Calculer le mot qui, par la transformée de Burrows-Wheeler, donne `NTTOEITNA,1` comme résultat.

**Exercice 2. (Codes et algorithme de Sardinas et Patterson)**

On rappelle qu'un ensemble $C$ fini de mots est un *code* si, pour tous mots $u_i \in C$ et $v_j \in C$ tels que $i \in [n]$ et $j \in [m]$, l'égalité

$$u_1 \cdot u_2 \dots u_n = v_1 \cdot v_2 \dots v_m$$

implique $n = m$ et $u_i = v_i$ pour tout $i \in [n]$.

1. Construire un code sur l'alphabet $\{\mathtt{a}, \mathtt{b}\}$.
2. Démontrer que l'ensemble $\{\mathtt{a}, \mathtt{ab}, \mathtt{ba}\}$ n'est pas un code.
3. Démontrer que tout ensemble $C$ de mots ayant tous la même longueur est un code.
4. Démontrer que tout ensemble $C$ de mots tel qu'aucun mot n'est préfixe propre d'un autre est un code.

L'*algorithme de Sardinas et Patterson* permet de décider si un ensemble $C$ fini de mots est un code. Il se décrit de la manière suivante :

(i) **initialisation :** $U_0 := C^{-1} \cdot C \setminus \{\epsilon\}$ ;

(ii) **itération :** $U_{i+1} := U_i^{-1} \cdot C \cup C^{-1} \cdot U_i$ ;

(iii) **arrêt** : si $\epsilon \in U_i$ alors $C$ n'est pas un code et si $U_i = U_j$ avec $i > j$, alors $C$ est un code.

On rappelle que si $L$ et $K$ sont deux ensembles de mots, l'ensemble $L^{-1} \cdot K$ est défini par

$$L^{-1} \cdot K := \{\text{mot } v : \text{ il existe } u \in L \text{ tel que } u \cdot v \in K\}.$$

5. Donner un argument pour justifier le fait que cette suite d'instructions termine sur l'entrée de tout ensemble fini de mots.

   En appliquant l'algorithme de Sardinas et Patterson, décider si les ensembles suivants sont des codes :

6. $C_1 := \{\mathtt{ab}, \mathtt{baa}, \mathtt{abba}, \mathtt{aabaa}\}$ ;
7. $C_2 := \{\mathtt{aaa}, \mathtt{aba}, \mathtt{abb}, \mathtt{abaab}\}$ ;
8. $C_3 := \{\mathtt{a}, \mathtt{abba}\}$ ;
9. $C_4 := \{\mathtt{b}, \mathtt{aa}, \mathtt{ab}, \mathtt{ba}\}$ ;
10. $C_5 := \{\mathtt{aa}, \mathtt{ab}, \mathtt{ba}, \mathtt{bb}, \mathtt{baaababa}\}$ ;
11. $C_6 := \{\mathtt{a}, \mathtt{ab}, \mathtt{bba}, \mathtt{bbab}, \mathtt{bbbb}\}$.

**Exercice 3. (Codes préfixes et longueur de mots)**

Soit $A := \{\mathtt{a}_1, \ldots, \mathtt{a}_k\}$ un alphabet de taille $k$. On considère une suite de nombres entiers positifs

$$s := (s_1, s_2, \ldots, s_n)$$

telle que

$$\sum_{i=1}^{n} \frac{s_i}{k^i} \leq 1. \tag{1}$$

On souhaite montrer par récurrence sur $n$ que si la suite $s$ vérifie (1), alors on peut construire un code préfixe $C_s$ sur l'alphabet $A$ qui contient exactement $s_i$ mots de longueur $i$.

1. Montrer que c'est possible si pour tout $i \in [n]$, on a $s_i \leq k - 1$.
2. Montrer que si la suite $s$ vérifie (1) et s'il existe $i \in [n]$ tel que $s_i \geq k$, la suite $s'$ définie pour tout $j \in [n]$ par

$$s'_j := \begin{cases} s_i - k & \text{si } j = i, \\ s_{i-1} + 1 & \text{si } j = i - 1, \\ s_j & \text{sinon,} \end{cases}$$

   vérifie aussi (1).
3. Terminer la démonstration par récurrence.
4. Décrire un algorithme qui, étant donnée une suite $s := (s_1, s_2, \ldots, s_n)$ vérifiant (1) construit un code préfixe sur $A$ avec exactement $s_i$ mots de longueur $i$.