TD #20 — Compression

Exercice 1. *Stop*

On propose la technique suivante pour construire un code $\mathcal{C} : \Sigma_1 \to \Sigma_2^*$ de longueur variable, pour une fonction de coût f :

— On choisit un caractère $s \in \Sigma_2$ quelconque qui servira à indiquer la fin des mots de code.
— On trie les lettres $c \in \Sigma_1$ par coût f décroissant, et l'on définit le code de la $i$ème lettre (en partant de zéro) dans cette énumération comme $\mathbb{I} \cdot s$, où $\mathbb{I}$ est l'écriture de i en base $\#\Sigma_2 - 1$ en utilisant les lettres de $\Sigma_2 \setminus \{s\}$ comme chiffres.

1. Donnez le code $\mathcal{C}_1$ obtenu avec la technique ci-dessus pour $\Sigma_1 = \{a, b, c, d, e\}$, de coûts $f(a) = 7$, $f(b) = 1$, $f(c) = 2$, $f(d) = 3$, $f(e) = 9$, $\Sigma_2 = \{0, 1\}$, et en utilisant $s = 0$.
2. Donnez un code $\mathcal{C}_2$ obtenu pour les mêmes alphabets & coût par l'algorithme « de Huffman ».
3. Mêmes questions pour $f(a) = 3$, $f(b) = 2$, $f(c) = 2$, $f(d) = 3$, $f(e) = 5$.

Exercice 2. *Code suffixe*

Soit un code $\mathcal{C} : \Sigma_1 \to \Sigma_2^*$ (supposé injectif), on dit qu'il est *suffixe* s'il n'existe pas de $c_1, c_2 \in \Sigma_1$ tels que $\mathcal{C}(c_1) = p \cdot \mathcal{C}(c_2)$ (autrement dit tels que l'encodage de c_2 est un suffixe de l'encodage de c_1).

1. Montrez que le code $\mathcal{C}$ étendu aux mots sur Σ_1 comme $\mathcal{C}(m) = \mathcal{C}(m_0 m_1 \cdots m_{\ell-1}) = \mathcal{C}(m_0)\mathcal{C}(m_1)\cdots\mathcal{C}(m_{\ell-1})$ est décodable de façon unique (est inversible).
2. Donnez un exemple simple de construction de code (injectif) qui soit à la fois préfixe et suffixe.
3. En quoi un code préfixe peut être préférable à un code suffixe pour le décodage du code étendu aux mots ?

Exercice 3. *Compression répétée*

On suppose vouloir compresser un fichier en utilisant l'algorithme « de Huffman » pour construire un code $\mathcal{C} : [\![256]\!] \to \{0, 1\}^*$ (autrement dit, on souhaite encoder les octets comme des mots binaires (de taille *a priori* variable)) et ensuite encoder chaque octet x du fichier par son code $\mathcal{C}(x)$.

1. Donnez le taux de compression maximal possible (le rapport entre la taille du fichier initial et celle du fichier compressé).

2. Donnez un exemple de contenu de fichier pour lequel celui-ci est atteint.
3. Montrez que la compression répétée (avec l'algorithme ci-dessus) d'un fichier contenant un unique octet peut **toujours** produire un fichier d'*un seul* octet (en supposant que la description du code est stockée séparément du fichier compressé). On pourra ignorer toute subtilité (notamment de *padding*) liée à l'encodage d'un mot binaire comme suite d'octets (on peut par exemple supposer que le fichier initial est de taille une puissance de deux et que l'encodage utilisé stocke séparément le nombre total de bits).
4. Pourquoi la question précédente n'implique-t'elle pas que l'on puisse atteindre un taux de compression (asymptotiquement) infini ?
5. Pourquoi une telle compression répétée risque-t'elle d'être « peu intéressante » pour un fichier quelconque ?

Exercice 4. *Représentation arborescente d'un code préfixe binaire*

On rappelle qu'une valeur t de :

```
type code_tree = E | L of int |
                 N of code_tree * code_tree
```

dont toutes les feuilles sont d'étiquettes distinctes représente un code préfixe binaire $\mathcal{C} : \mathcal{S} \to \{0,1\}$ (pour $\mathcal{S} \subset \mathbb{N}$) de la façon suivante : le code de x est donné par la numérotation binaire dans t de la feuille d'étiquette x.

1. Écrivez une fonction OCaml :
   `code2tree : (int * int list) list -> code_tree`
   qui construit un arbre représentant le code (préfixe) donné en argument sous la forme d'une liste de couples `(x, cx)` où `x` représente une lettre $\in \mathcal{S}$ de façon naturelle et `cx` son code sous la forme d'une liste de `0` et `1`.

Exercice 5. *LZW sans tableau associatif*

On rappelle que l'algorithme de compression LZW utilise une fonction partielle $\mathcal{T} : [\![n]\!] \to \Sigma^*$ pour certains n & Σ, et que cette fonction doit pouvoir être efficacement inversible (en particulier, on doit pouvoir efficacement déterminer si $m \in \Sigma^*$ possède un antécédent pour $\mathcal{T}$).

1. Expliquez comment $\mathcal{T}$ et sont inverse $\mathcal{T}^{-1}$peuvent être représentés en utilisant une structure de donnée de tableau (éventuellement associatif).
2. Quelle « structure » possèdent les images de $\mathcal{T}$? Notamment, soit $m \in \Sigma^*$ de longueur > 1, que peut on déduire du fait que c'est une image de $\mathcal{T}$?

On suppose ci-dessous que les lettres de Σ sont représentés par des `int`.

3. Expliquez comment la structure mise en évidence à la question précédente permet de représenter un motif $m \in \Sigma^*$ par un type (par exemple) `type pat = P of pat option * int`.
4. Déduire de ce qui précède comment un tableau à deux dimensions $n \times \#\Sigma$ peut être utilisé pour calculer l'antécédent d'un motif par $\mathcal{T}$ en temps linéaire en sa longueur.