# Compression bzip (X mp 2007)

Durée : 2 heures

---

Le temps d'exécution $T(f)$ d'une fonction $f$ est le nombre d'opérations élémentaires (addition, soustraction, multiplication, division, affectation, etc.) nécessaire au calcul de $f$. Lorsque ce temps d'exécution dépend d'un paramètre $n$, il sera noté $T_n(f)$. On dit que la fonction f s'exécute :

en temps $O(n^\alpha)$, s'il existe $K > 0$ tel que pour tout $n$, $T_n(f) \leqslant Kn^\alpha$.

Dans ce sujet, il sera question de l'algorithme de Burrows-Wheeler qui compresse très efficacement des données textuelles. L'*American Standard Code for Information Interchange* [1] (plus connu sous l'acronyme ASCII) est une norme informatique de codage de caractères qui attribue à chaque caractère de l'alphabet latin un entier codé sur 8 bits. C'est la raison pour laquelle le texte d'entrée à compresser sera représenté par un tableau $t$ contenant des entiers compris entre 0 et 255 inclus.

## Partie I. Compression par redondance

La compression par redondance compresse un texte d'entrée qui possède des répétitions consécutives de lettres (ou d'entiers dans notre cas). Dans un premier temps, on calcule les fréquences d'apparition de chaque entier dans le texte d'entrée. Puis on compresse le texte.

**Question 1.** Écrire la fonction `occurrences(t)` qui prend en argument un tableau d'entrée $t$ de longueur $n$ ; et qui renvoie un tableau $r$ de taille 256 tel que $r[i]$ est le nombre d'occurrences de $i$ dans $t$ pour $0 \leqslant i < 256$.

**Question 2.** Écrire la fonction `mini(t)` qui prend en argument le tableau $t$ de longueur $n$ ; et qui renvoie le plus petit entier de l'intervalle $[0, 255]$ qui apparaît le moins souvent dans le tableau $t$. (Le nombre d'occurrences de cet entier peut être nul.)

L'entier `mini`$(t)$ servira de marqueur. On note $\mu$ pour ce marqueur et, pour simplifier, on suppose que son nombre d'occurrences est nul. Donc $r[\mu] = 0$ quand $r =$ `occurrences`$(t)$. La compression par redondance du texte $t$ fonctionne comme suit : on rajoute $\mu$ au début du texte, puis toute répétition maximale contiguë d'une lettre où $t[i] = t[i+1] = \cdots = t[j] = k$ est codée par les trois entiers $\mu, (j-i), k$ ; toute apparition unique d'une lettre $k$ est codée par cette même lettre.

Par exemple, si le tableau $t$ contient les valeurs $(0, 0, 3, 2, 3, 3, 3, 3, 3, 3, 5)$. Le marqueur est donc 1 car 1 n'apparaît pas dans ce tableau. Le texte $t'$ compressé est alors

$$\underbrace{1}_{\mu}, \underbrace{1,1,0,}_{0,0} \underbrace{3}_{3}, \underbrace{2}_{2}, \underbrace{1,5,3}_{3,3,3,3,3}, \underbrace{5}_{5}$$

**Question 3.** Écrire la fonction `tailleCodage(t)` qui prend comme argument le tableau $t$ et calcule la taille $n'$ du texte compressé ($n' = 10$ dans l'exemple ci-dessus).

**Question 4.** Écrire la fonction `codage(t)` qui prend comme paramètre le tableau $t$ et renvoie un tableau d'entiers $t'$ représentant le texte compressé.

**Question 5.** Pour pouvoir décoder un texte $t'$ ainsi compressé, il suffit de connaître le marqueur utilisé. Or ce marqueur est le premier entier du texte compressé.
Écrire la fonction `decodage(tprime)` qui prend comme paramètre le tableau compressé $t'$ et renvoie le tableau initial.

## Partie II. Transformation de Burrows-Wheeler

Le codage par redondance n'est efficace que si le texte présente de nombreuses répétitions consécutives de lettres. Ce n'est évidemment pas le cas pour un texte pris au hasard. La transformation de Burrows-Wheeler est une transformation qui, à partir d'un texte donné, produit un autre texte contenant exactement les mêmes lettres mais dans un autre ordre où les répétitions de lettres ont tendance à être contiguës. Cette transformation est bijective.

Considérons par exemple le texte d'entrée `concours`. Pour simplifier la présentation, nous utilisons ici des caractères pour le tableau d'entrée. Cependant, dans les programmes, on considère toujours (comme dans la première partie) que le texte d'entrée est un tableau d'entiers compris entre 0 et 255 inclus. Le principe de la transformation suit les trois étapes suivantes :

1. Et plus précisément son extension Latin-1.

1. On regarde toutes les rotations du texte. Dans notre cas il y en a 8 qui sont :

   `concours oncoursc ncoursco courscon oursconc ursconco rsconcou sconcour`

2. On trie ces rotations par ordre lexicographique (l'ordre du dictionnaire).

   `concours courscon ncoursco oncoursc oursconc rsconcou sconcour ursconco`

3. Le texte résultant est formé par toutes les dernières lettres des mots dans l'ordre précédent, soit `snoccuro` dans l'exemple. L'indice, dans ce texte résultant, de la première lettre du texte original est appelé la *clef* de la transformation. Dans l'exemple, la clef est égale à 3.

On remarque que les deux `c` du texte de départ se retrouvent côte à côte après la transformation. En effet, comme le tri des rotations regroupe les mots débutant par les mêmes lettres, cela conduit à rapprocher aussi les dernières lettres qui les précèdent dans le texte d'entrée.
On le constate aussi sur la chaîne : `concours de l ecole polytechnique` dont la transformée par Burrows-Wheeler est `sleeeeen dlt ucn ooohcpcc iuryqol`.

En pratique, on ne va pas calculer et stocker l'ensemble des rotations du mot d'entrée. On se contente de noter par $rot_i$ la $i$-ème rotation du mot. Ainsi, dans l'exemple, $rot_0$ représente le texte d'entrée `concours`, $rot_1$ représente `oncoursc`, $rot_2$ représente `ncoursco`, etc.

**Question 6.** Écrire la fonction `comparerRotations(t, i, j)` qui prend comme arguments le texte $t$ de longueur $n$ et deux indices $i, j$ ; et qui renvoie, en temps linéaire par rapport à $n$ :

- 1 si $rot_i$ est plus grand que $rot_j$ dans l'ordre lexicographique ;
- $-1$ si $rot_i$ est plus petit que $rot_j$ dans l'ordre lexicographique ;
- 0 sinon.

On suppose disposer d'une fonction `triRotations(t)` qui trie les rotations du texte donné dans le tableau $t$ en utilisant la fonction `comparerRotation`. Elle renvoie un tableau d'entiers $r$ représentant les numéros des rotations ($rot_{r[0]} \leqslant rot_{r[1]} \leqslant \cdots \leqslant rot_{r[n-1]}$). Cette fonction réalise dans le pire des cas $O(n \log n)$ appels à la fonction de comparaison.

**Question 7.** Écrire une fonction `codageBW(t)` qui prend en paramètre le tableau $t$ de longueur $n$ ; et qui renvoie un couple $(c, t')$ où $c$ est la clef et $t'$ un tableau contenant le texte après transformation.

**Question 8.** Donner un ordre de grandeur du temps d'exécution de la fonction `codageBW` en fonction de $n$.

Pour réaliser l'ensemble du codage, il ne reste plus qu'à réaliser la compression par redondance sur la transformée $t'$ du texte d'entrée $t$.

## Partie III. Transformation de Burrows-Wheeler inverse

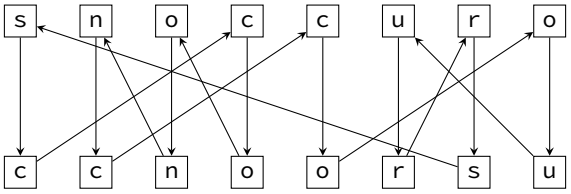
Pour décoder le texte $t'$ (`snoccuro` dans l'exemple) avec sa clef (3 dans l'exemple) obtenus après la transformation, on construit d'abord un tableau `triCars` de même taille que $t'$ qui contient les mêmes lettres que $t'$ mais dans l'ordre lexicographique croissant. Dans l'exemple, `triCars` $= \langle c, c, n, o, o, r, s, u \rangle$.

**Question 9.** Écrire la fonction `triCarsDe(tprime)` qui part du texte codé $t'$ de taille $n$ ; et qui renvoie, en temps linéaire par rapport à $n$, le tableau `triCars` décrit précédemment. On pourra utiliser la fonction `occurrences` de la question 1.

Puis on considère le texte codé $t'$ et le tableau `triCars` précédent (la clef est entourée d'un cercle).

| s | n | o | (c) | c | u | r | o |
|---|---|---|---|---|---|---|---|
| c | c | n | o | o | r | s | u |

À chaque lettre de la première ligne, on associe la lettre de la seconde à la même position. À chaque lettre de la deuxième ligne, on associe la même lettre de même rang dans la première ligne. La figure suivante montre ces deux correspondances.



On retrouve le texte de départ `concours` en partant de la clef et en suivant les flèches du dessin précédent.
Il faut donc construire le tableau `indices` tel que `indices`$[i]$ est l'indice de la lettre `triCars`$[i]$ dans le texte $t'$. Si plusieurs occurrences de cette lettre figurent dans $t'$, on fait correspondre celle qui figure au même rang dans $t'$. Le tableau `indices` donne donc la correspondance représentée par les flèches de la seconde ligne vers la première. Sur l'exemple, le tableau `indices` contient les valeurs $\langle 3, 4, 1, 2, 7, 6, 0, 5\rangle$.

**Question 10.** Écrire la fonction `trouverIndices(tprime)` prenant en paramètre le texte $t'$ codé de longueur $n$ et qui renvoie le tableau `indices` précédemment décrit. Quel est son temps d'exécution en fonction de $n$ ?

**Question 11.** Écrire une fonction `decodageBW(clef, tprime)` qui prend comme paramètre la clef $c$ et un texte $t'$ de longueur $n$ ; et renvoie le texte $t$ d'origine. Quel est son temps d'exécution en fonction de $n$ ?

**Question 12.** [Bonus [2]] Prouver la correction de l'algorithme précédent.

♦ ♦
♦

2. Par définition, on appelle *question bonus* toute question difficile et fort peu rémunératrice.