

## IC Devoir 103 - 2h - Compression bzip

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  $\mathcal{O}(n^\alpha)$ , s'il existe  $K > 0$  tel que pour tout  $n$ ,  $T_n(f) \leq Kn^\alpha$ .

Afin d'optimiser la complexité, on n'utilisera pas les méthodes connues du type list de PYTHON.

Dans ce sujet, il sera question de l'algorithme de Burrows-Wheeler qui compresse très efficacement des données textuelles. Le texte d'entrée à compresser sera représenté par un tableau  $t$  (liste) contenant des entiers compris entre 0 et 255 inclus.

### 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.

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

2. Écrire la fonction `min(t)` qui prend en argument le tableau  $t$  ; et qui retourne le plus petit entier de l'intervalle  $\llbracket 0, 255 \rrbracket$  qui apparaît le moins souvent dans le tableau  $t$ . (Le nombre d'occurrences de cet entier peut être nul)

L'entier `min(t)` servira de *marqueur*. On note  $m$  pour ce marqueur et, pour simplifier, on suppose que son nombre d'occurrences est nul. Donc  $r[m]$  contient la valeur 0 quand  $r = \text{occurrences}(t)$ .

La compression par redondance du texte  $t$  fonctionne comme suit :

- toute répétition maximale contigüe d'une lettre où  $t[i]$ ,  $t[i+1]$ , ...,  $t[j]$  contiennent  $k$  (avec  $t[i-1]$  et  $t[j+1]$  ne contenant pas  $k$ ) est codée par les trois entiers  $m$ ,  $(j-i)$ ,  $k$
- toute apparition unique d'une lettre est codée par cette même lettre.

Par exemple, considérons le tableau  $t$  :

$[0, 0, 3, 2, 3, 3, 3, 3, 3, 5]$

Le marqueur est donc 1 car 1 n'apparaît pas dans ce tableau.

Le texte  $tt$  compressé est alors  $[1, 1, 1, 0, 3, 2, 1, 5, 3, 5]$  :

$$\underbrace{1}_m, \underbrace{1, 1, 0}_{0,0}, \underbrace{3}_3, \underbrace{2}_2, \underbrace{1, 5, 3}_{3,3,3,3,3,3}, \underbrace{5}_5$$

3. Compléter la fonction `tailleCodage(t)` suivante (au proposer une fonction équivalente de votre choix) qui prend comme argument le tableau  $t$  ; et qui calcule la taille  $nn$  du texte compressé. Dans l'exemple précédent,  $nn$  vaut 10.

```

1 def tailleCodage(t):
2     nn=1 # le marqueur
3     i,j=0,1
4     while i+j<len(t):
5         # calcul du nombre de repetitions
6         if t[i+j]==t[i]:
7             j=...
8         # si la lettre est isolee
9         elif j==1:
10            nn,i=...
11        # si la lettre n'est pas isolee
12        else:
13            nn,i,j=...
14        # gestion de la derniere lettre
15        if ...:
16            ...
17        else:
18            ...
19        return(nn)

```

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

Pour pouvoir décoder un texte  $tt$  ainsi compressé, il suffit de connaître le marqueur utilisé. Or ce marqueur est le premier entier du texte compressé.

### 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 contigües. 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 :

- (i) – On regarde toutes les rotations du texte. Dans notre cas, il y en a 8 qui sont :
- (ii) – On trie ces rotations par ordre lexicographique (l'ordre du dictionnaire).

concours	concours
oncoursc	courscon
ncoursco	ncoursco
courscon	oncoursc
oursconc	oursconc
ursconco	rsconcou
rsconcou	sconcour
sconcour	ursconco

(iii) – Le texte résultant est formé par toutes les dernières lettres des mots dans l'ordre précédent, soit `snoccuro` dans l'exemple, ainsi que de l'indice de la lettre dans ce texte résultant qui est la première lettre du texte original, soit `3` dans notre exemple. On appelle cet entier la *clé* de la transformation.

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 mêmes lettres sur la première colonne, cela conduit à rapprocher aussi les lettres de la dernière colonne 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 ooohcpc iuryqo`.

5. Écrire une fonction `rot(t, i)` qui prend comme arguments le texte `t` ; et qui retourne sa *i*ème rotation, c'est-à-dire les lettres `t[i], t[i+1], ...`. Par exemple, `rot(t, 0)` retourne le texte d'entrée `concours`, `rot(t, 1)` retourne `oncoursc`, `rot(t, 2)` retourne `ncoursco`, etc.

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

- 1 si la *i*ème rotation est plus grand que la *j*ème dans l'ordre lexicographique,
- -1 si la *i*ème rotation est plus petit que la *j*ème 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 retourne un tableau d'entiers `r` représentant les numéros des rotations ordonnées (le résultat de `rot(t, r[0])` est plus petit que celui de `rot(t, r[1])`, qui est plus petit que celui de `rot(t, r[n-1])`, etc.). Cette fonction réalise dans le pire des cas  $\mathcal{O}(n \ln(n))$  appels à la fonction de comparaison où `n` est la longueur de `t`.

7. Écrire une fonction `codageBW(t)` qui prend en paramètre le tableau `t` ; et qui renvoie un tableau contenant le texte après transformation. (La clé sera stockée dans la dernière case de ce tableau)

8. Donner un ordre de grandeur du temps d'exécution de la fonction `codageBW` en fonction de `n` la taille de `t`.

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

### III. TRANSFORMATION DE BURROWS-WHEELER INVERSE

Pour décoder le texte `tt` (`snoccuro3` dans l'exemple) de taille `nn` de contenu égal à `n+1` obtenu après transformation, on construit d'abord un tableau `triCars` de taille `n` qui contient les mêmes lettres que le texte `tt` mais dans l'ordre lexicographique croissant. Dans l'exemple, `triCars` contient `[c, c, n, o, o, r, s, u]`.

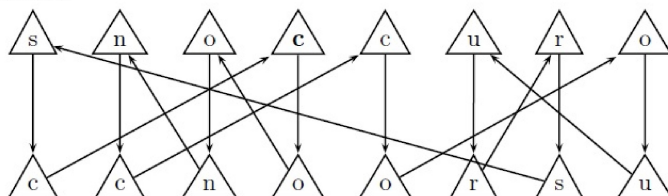
9. Écrire une fonction `frequencies(tt)` qui prend comme argument un tableau `tt` correspondant au texte codé (avec la clé dans la dernière case) ; et qui renvoie un tableau de taille `256` contenant le nombre d'occurrences de chaque lettre dans `tt`.

10. Écrire la fonction `triCarsDe(tt)` qui part du texte codé `tt` de taille `nn` ; et qui renvoie, en temps linéaire par rapport à `n`, le tableau `triCars` décrit précédemment.

Puis on considère le texte codé `tt` et le tableau `triCars` précédent (la clé est représentée soulignée).

s	n	o	<u>c</u>	c	u	r	o	3
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 clé (position de la lettre en caractère gras) 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 `tt`. Si plusieurs occurrences de cette lettre figurent dans `tt`, on fait correspondre celle qui figure au même rang dans `tt`. 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 `[3, 4, 1, 2, 7, 6, 0, 5]`.

11. Écrire la fonction `trouverIndices(tt)` prenant en paramètre le texte `tt` codé ; et qui retourne le tableau `indices` précédemment décrit.

Quel est son temps d'exécution en fonction de `n` la longueur du mot initial ?

12. Écrire une fonction `decodageBW(tt)` qui prend comme paramètre un texte `tt` ; et retourne le texte `t` d'origine.

Quel est son temps d'exécution en fonction de `n` ?

#### IV. COMPLÉMENTS - (BONUS)

13. Écrire une fonction `decodage(tt)` qui prend comme argument le tableau `tt` représentant un texte compressé par redondance et retourne le tableau `t` du texte originel.
14. Écrire un fonction `numerise(mot)` qui prend comme argument une chaîne de caractères `mot` et qui retourne la tableau d'entiers associés. On utilisera les fonctions PYTHON `chr` et `ord`. Exemple

```
>>> ord('A')
65
>>> chr(65)
'A'
```

15. Écrire un fonction `denumerise(t)` qui prend comme argument un tableau `t` d'entiers entre 0 et 255 et qui retourne la chaîne de caractères associée.
16. Écrire la fonction `triRotations(t)`, définie à la partie 2) qui trie les rotations du texte donné dans le tableau `t` en utilisant la fonction `comparerRotation`. En particulier, le tri fusion permet d'obtenir la complexité recherchée.
17. Écrire les fonctions `comprimeBW(mot)` et `decomprimeBW(tt)` qui finalisent le travail.