

TP M2 ALG

Transformée de Burrows Wheeler et FM-index

C. Lemaitre, P. Peterlongo, M. Temperville

2026

Tableau des suffixes

Pour ce TP, nous utiliserons une (excellente) implémentation de manipulation des chaînes de caractères : <https://github.com/louisabraham/pydivsufsort>. Installation avec *pip install pydivsufsort*

Voici un exemple d'utilisation :

```
from pydivsufsort import divsufsort
s = 'GGCGGCACCGC$'
sa = divsufsort(s)
print('i\tSA\tf\tSuffixes')
for i in range(len(s)):
    print(f'{i}\t{sa[i]}\t{s[sa[i]]}\t{s[sa[i]:]}')
```

Produira la sortie suivante :

i	SA	f	Suffixes
0	11	\$	\$
1	6	A	ACCGC\$
2	10	C	C\$
3	5	C	CACCGC\$
4	7	C	CCGC\$
5	8	C	CGC\$
6	2	C	CGGCACCGC\$
7	9	G	GC\$
...			

Implementation BWT

Q 1. Codez (et testez...) la fonction *get_bwt(s, sa)* renvoyant la transformée de burrows wheeler *bwt* du texte *s* dont le *sa* a été calculé.

Q 2. Codez la fonction *get_n(bwt)* renvoyant un dictionnaire *n* tel que $n[\alpha]$ =nombre d'occurrences de caractères plus petits que α dans *bwt*. C'est aussi l'indice de la ligne où apparait le premier suffixe débutant par α . Par exemple $[G] = 7$.

Q 3. Codez la fonction *get_r(bwt)* renvoyant un tableau tel que $r[i]$ = rang dans *bwt* du caractère *bwt[i]*. Exemple : $r[3]$ est égal à 2 car *bwt[3]* est le deuxième 'G' apparaissant dans *bwt*.

Q 4. Codez la fonction *left_first*(α, k, n) (appelée *LF* en cours/TD) qui renvoie la ligne l telle que $sa[l]$ est la position du $k^{\text{ième}}$ suffixe (ordonnés dans l'ordre lexico) débutant par le caractère α . Plus simplement, avec f tel que $f[i] = s[sa[i]]$, $lf(\alpha; k)$ est la ligne où le $k^{\text{ième}}$ α apparaît dans f .

Q 5. Codez la fonction *bwt_2_seq*(bwt, n, r) qui retrouve en temps linéaire la séquence s dont la *bwt* est originaire.

Q 6. Codez la fonction *contains*(p, bwt, n, r) qui renvoie *true* si p apparait dans la séquence s dont la *bwt* est originaire, *false* sinon.

Q 7. Codez la fonction *nb_occurrences*(p, bwt, n, r) qui renvoie le nombre d'occurrences de p dans la séquence s dont la *bwt* est originaire, *false* sinon.

Validations sur des exemples *jouets* (e.g. $s = GGCGGCACCGC\$$)

Q 8. Vérifiez le plus de conditions possibles (recherche du premier ou dernier mot du texte indexé, recherche du mot vide, recherche d'un mot plus long que le texte, ...)

Application sur une portion du génome d'E. coli.

Télécharger le fichier `bioinformatique.rennes.inria.fr/data/ecoli_sample.fa` qui contiennent les premiers 150 kbp d'E. coli. Construire la 'BWT' sur la séquence contenue dans ce fichier.

Q 9. Recherchez par exemple le pattern 'CGCTCTGTGTGACAAGCCGGAAACCGCCCAG' (une sous séquence des premiers 150 kbp d'E. coli) via la BWT.

Q 10. Appliquez l'algorithme naïf et l'algorithme de Karp Rabin sur ces mêmes données et comparez les résultats.

Pire et meilleur des cas

Q 11. Rappelez quel est le pire des cas des méthodes évoquées précédemment. Proposez des tests et des résultats.

Bonus

Q 12. [Bonus] Codez la fonction *occurs*(s, sa, p) : indiquant si p a des occurrences dans s en utilisation une approche dichotomique (en $O(|p| \cdot \log |s|)$).

Q 13. [Bonus] Ajoutez les résultats par recherche dichotomique à tous les résultats précédemment obtenus.

Q 14. [Bonus (facile)] Codez la fonction *get_occurrences*(p, bwt, n, r, sa) qui renvoie la liste des positions des occurrences de p dans la séquence s dont la *bwt* est originaire.

Q 15. [Bonus (délicat)] Proposez et implémentez un moyen de sous échantillonner les ranks pour limiter l'impact mémoire, au prix d'une augmentation des temps de query.