EA4 – Éléments d'algorithmique
Complément : construction de la table des suffixes

Soit `t` un texte de longueur n sur un alphabet A, représenté par un tableau. On appelle *suffixe d'indice i* de `t` le facteur (sous-tableau) `t[`i`:]`. La *table des suffixes* de `t` est le tableau `Suff(t)` contenant les entiers 0 à $n-1$, triés selon l'ordre $\prec$ suivant :

$$i \prec j \text{ si } \texttt{t[}i\texttt{:]} \text{ précède } \texttt{t[}j\texttt{:]} \text{ pour l'ordre lexicographique.}$$

Théorème. *La table des suffixes* ***`Suff(t)`*** *de* **`t`** *peut être construite en temps* $\Theta(n)$.

Étapes de la construction de la table des suffixes

1. ajouter à l'alphabet A une lettre `'0'`, inférieure à toutes les autres lettres, et compléter `t` avec 0, 1 ou 2 caractères `'0'` pour que $n = \texttt{len(t)} \equiv 0 \mod 3$
2. partitionner l'ensemble des suffixes de `t` en 3 sous-ensembles de même cardinal $\frac{n}{3}$: $S = S_0 \sqcup S_1 \sqcup S_2$, où $S_i = \{\texttt{t[j:]} \mid j \equiv i \mod 3\}$
3. soit $E = S_1 \sqcup S_2$ l'ensemble des *suffixes-échantillons* de `t` ; définir un texte `aux` dont l'ensemble des suffixes correspond à E par une bijection respectant l'ordre
4. calculer récursivement `Suff(aux)`, et trier E en conséquence
5. trier S_0
6. fusionner S_0 et E

Exemple : `texte = "quelbonbonbon"` devient `texte = "quelbonbonbon00"`, de longueur $n = 15$, et ses trois ensembles de 5 suffixes sont :
$S_0 =$ `["quelbonbonbon00", "lbonbonbon00", "nbonbon00", "nbon00", "n00"]`,
$S_1 =$ `["uelbonbonbon00", "bonbonbon00", "bonbon00", "bon00", "00"]`,
$S_2 =$ `["elbonbonbon00", "onbonbon00", "onbon00", "on00", "0"]`.

Calcul de la complexité

L'enjeu est de réussir à effectuer les étapes 3, 5 et 6 en temps linéaire ; admettons provisoirement que c'est possible. L'appel récursif de l'étape 4 portant sur un tableau de longueur $\frac{2}{3}n$, la complexité vérifierait donc :

$$C(n) = C\left(\tfrac{2}{3}n\right) + \Theta(n),$$

i.e. $C(n) \leqslant C\left(\frac{2}{3}n\right) + \alpha n$ pour une certaine constante α ; donc, pour tout $k < \log_{\frac{2}{3}} n$:

$$C\left(\left(\tfrac{2}{3}\right)^k n\right) \leqslant C\left(\left(\tfrac{2}{3}\right)^{k+1} n\right) + \alpha \left(\tfrac{2}{3}\right)^k n$$

d'où finalement

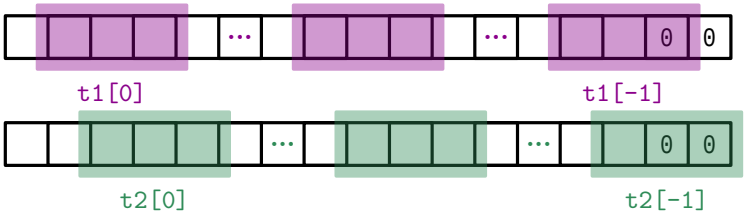
$$C(n) \leqslant \alpha \sum_k \left(\tfrac{2}{3}\right)^k n = 3\alpha n = \Theta(n),$$

ce qui prouve donc le théorème.

Étape 3 : définition du texte auxiliaire On cherche à définir un texte ayant $\frac{2}{3}n$ su□xes – donc de longueur $\frac{2}{3}n$ – correspondant « naturellement » aux éléments de $E = S_1 \sqcup S_2$. Pour cela, on groupe les lettres par 3 pour définir de nouvelles « super-lettres ». En commençant à `t[1]`, on obtient un mot `t1` dont les su□xes sont en bijection naturelle avec S_1. En commençant à `t[2]`, on obtient un mot `t2` dont les su□xes sont en bijection naturelle avec S_2.
En concaténant `t1` et `t2`, on obtient un mot dont les su□xes « sont » les éléments de S_1 (auxquels on concatène `t2`) et ceux de S_2. Malheureusement, concaténer `t2` à la fin de tous les mots de S_1 peut a□ecter leur position par rapport aux éléments de S_2 dans l'ordre alphabétique. Une astuce pour contourner ce problème consiste à ajouter encore 2 caractères `'0'` :

(a) ajouter encore 2 caractères `'0'` à la fin de `t`

(b) définir `t1` et `t2` : mots sur l'alphabet A^3 dont la « projection » sur A^* est respectivement `t[1:-1]` et `t[2:]` :



(c) définir un nouvel alphabet ordonné comme A^3 : trier en temps linéaire les éléments du tableau `t1 + t2`, donc déterminer le rang de chaque « lettre-triplet » apparaissant dans l'un des deux mots ;

(d) `aux` est le mot obtenu à partir de `t1 + t2` en substituant son rang à chaque lettre-triplet.

Exemple : `texte = "quelbonbonbon"`
`t1 = ["uel", "bon", "bon", "bon", "000"],`
`t2 = ["elb", "onb", "onb", "on0", "000"]`
ordre des 6 triplets apparaissant dans `t1 + t2` :
`"000" < "bon" < "elb" < "on0" < "onb" < "uel"`
donc si on les numérote de 1 à 6 : `aux = "6222135541"`.

Lemme. *Les suffixes de* **aux** *sont ordonnés comme les suffixes de* **t** *correspondants.*

Démonstration. comme 0 est la plus petite lettre, le triplet $(\texttt{t[n-2]}, \texttt{t[n-1]}, 0)$ fait une coupure dans `t1 + t2` ; la comparaison lexicographique de deux su□xes (dont l'un contient le triplet) s'arrête à ce triplet, donc s'arrête en fait à la fin du mot `t`. □

Lemme. *L'étape (c) est de complexité linéaire en k, le nombre de triplets.*

Démonstration. Il s'agit d'un *tri par base* : commencer par trier selon la 3e composante, puis (de façon stable) selon la 2e, et enfin selon la 1re. Chacun des trois passages est linéaire car le nombre de valeurs distinctes est borné indépendamment de k par la taille de l'alphabet A, notée ℓ ; il su□t donc de prévoir ℓ « boîtes » (des listes chaînées par exemple) dans lesquelles les k triplets seront répartis, puis de regrouper les boîtes dans le bon ordre.
(remarque : ce n'est pas un tri en place.) □

Étapes 5 et 6 : tri de tous les su□xes à partir des su□xes-échantillons

Lemme. *Une fois S_1 trié, S_0 peut être trié en temps linéaire.*

Démonstration. Chaque élément de S_0 est formé d'une initiale et d'un élément de S_1. Donc un tri par base permet de trier S_0 en deux phases : d'abord selon l'élément de S_1, puis selon l'initiale. □

Lemme. *La fusion de E et de S_0 peut être effectuée en temps linéaire.*

Démonstration. À peu près comme le lemme précédent, en plus malin. Lors de la fusion de ces deux listes triées, il faut comparer chaque élément de S_0 à un élément de S_1 (là, c'est exactement comme au-dessus, initiale + élément de E), ou à un élément de S_2 (et là, il faut regarder les deux premières lettres, et le reste est dans E, donc un tri par base en 3 phases). □