Tri des suffixes

Réf: Crochemore: Algorithmique du texte

L'objectif du classement est de calculer une permutation p des indices sur y qui satisfait la condition $y[p[0]..n-1] <_{lex} y[p[1]..n-1] <_{lex} \dots <_{lex} y[p[n-1]..n-1]$

Prop: Une telle permutation peut être calculée en temps $O(n \log n)$.

Démo
- On définit $pref_k(u) = u[0..k-1]$ et on va comparer les $pref_k$ des suffixes, pour k à croissance exponentielle. Plus précisément, on définit $R_k[i]$ le rang de $pref_k(y[i..n-1])$ dans la liste ordonnée des mots de l'ensemble $\{pref_k(u) : u \leq_{suff} y, u \neq \varepsilon\}$

- On peut voir R_k comme une relation d'équivalence sur les positions via $i \equiv_k j$ ssi $R_k[i] = R_k[j]$. La propriété clé étant que $\equiv_{2k}$ est un raffinement de $\equiv_k$. On va donc partir de $\equiv_1$, qui revient à identifier les lettres de y, pour arriver à $\equiv_n$, où chaque suffixe sera seul dans sa classe.

- Lemme: (Dedoublement). Pour k, i entiers, $k > 0$ et $0 \leq i < n$, $R_{2k}[i]$ est le rang du couple $(R_k[i], R_k[i+k])$ dans la liste classée en ordre lexico de ces couples (où on prendra $R_k[i] = -1$ pour $i \geq n$).

  Démo: * Poser $R_k[i] = -1$ pour $i \geq n$ revient à considérer le mot infini ya^∞, où a est une lettre strictement inférieure à toutes celles qui apparaissent dans y. Ainsi, pour $i \geq n$, le facteur de taille k correspondant est a^k, qui est bien inférieur à tous les autres, d'où la convention

  * Par définition, $R_{2k}[i]$ est le rang de $pref_{2k}(y[i..\ \ 1])$ dans la liste classée des facteurs de longueur $2k$ du mot ya^∞. On note

  $u(i) = pref_k(y[i..\ \ ])$

  $v(i) = pref_k(y[i+k..\ \ ])$

On a $pren_{2k}(y[i..\infty]) = u(i).v(i)$, d'où pour $0 \leq i \neq j \leq n$,
l'inégalité $pren_{2k}(y[i..\infty]) < pren_{2k}(y[j..\infty])$ est équiv
à $u(i).v(i) < u(j).v(j)$
$\Leftrightarrow (u(i), v(i)) < (u(j), v(j))$
$\Leftrightarrow (R_k(i), R_k(i+k)) < (R_k(j), R_k(j+k))$
D'où le résultat □

- On obtient alors l'algo suivant :

```
TRI-SUFFIXES (y, n)
    POUR i ← 0 à n-1 FAIRE p[i] ← i
    k ← 1
    POUR i ← 0 à n-1 FAIRE R_1[i] ← rang de y[i] dans la liste classée des lettres de y
    p ← TRI(p, n, R_1, 0)
    i ← max R_1
    TANT QUE i < n-1 FAIRE
        p ← TRI(p, n, R_k, k)
        p ← TRI(p, n, R_k, 0)
        R_2k ← RANG(p)
        i ← max R_2k
        k ← 2k
    RENVOYER p
```

k ← 1 → initialisation
Dédoublement k ← 2k

- TRI(p, n, R, k) trie la suite d'entiers $p[0]+k, p[1]+k, \dots, p[n-1]+k$ suivant leur clé R. De plus TRI doit vérifier une condition de stabilité qui est : si $R[p[i]+k] = R[p[j]+k]$, alors $p[i] < p[j]$ ssi $p'[i] < p'[j]$.
↳ on peut prendre le tri par bac qui est linéaire

- Pour prouver la correction, on montre l'invariant de boucle suivant :
$(pren_k(y[p[i]..n-1]) : i = 0..n-1)$ est croissante.

- Pour la complexité : on a au plus $\log_2 n$ étapes, chacune en $O(n)$
d'où le $O(n \log n)$ □