

Arbre des suffixes : algorithme d'Ukkonen

Thierry Lecroq
Université de Rouen

Algorithme d'Ukkonen

L'algorithme de construction de l'arbre des suffixes d'un mot y de longueur n d'Ukkonen est un algorithme incrémental qui insère une à une les lettres de y dans l'arbre de la première à la dernière.

Algorithme d'Ukkonen

On ajoute le terminateur $\$$ au mot y :

$$y = y[0..n-1]\$$$

avec

$$\$ \notin \text{alph}(y)$$

Définition

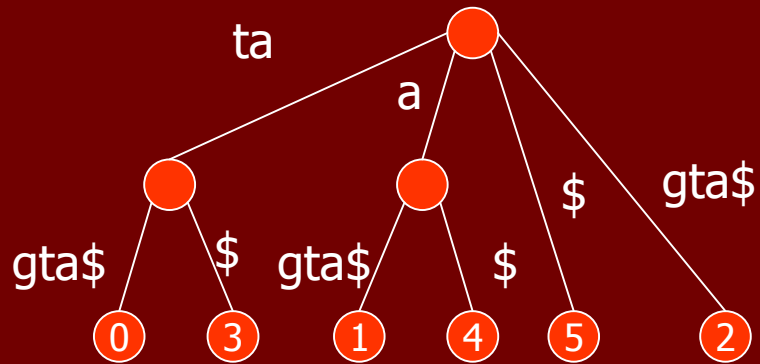
L'arbre implicite des suffixes de y est obtenu à partir de l'arbre des suffixes de $y\$$ en supprimant toutes les occurrences du symbole $\$$, puis en supprimant toutes les branches sans étiquettes, puis en supprimant tous les nœuds internes qui n'ont pas au moins deux descendants.

Définition

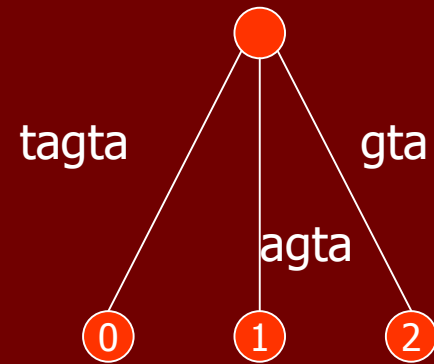
L'arbre implicite de suffixes de $y[0..i]$ est noté T_i .

Exemple

$y = \text{tagta}$



arbre des suffixes de tagta\$



arbre implicite des suffixes de tagta

Algorithme d'Ukkonen

On associe chaque nœud interne avec le mot formant l'étiquette depuis la racine à ce nœud interne.

Ainsi la racine est associée à ε .

On associe l'indice j à la feuille au bout du chemin étiqueté par $y[j..n]$ depuis la racine.

Un premier algorithme en $O(n^3)$

- n phases
- la phase $i+1$
 - construit T_{i+1} à partir de T_i
 - divisée en $i+2$ extensions
- l'extension j
 - insère $y[j..i+1]$ dans l'arbre en
 - recherchant la fin du chemin étiqueté par $y[j..i]$ depuis la racine de T_i
 - rajoutant $y[i+1]$ si nécessaire

algo UKKONEN₁(y, n)

construire T_0

pour $i \leftarrow 0$ à $n-1$ **faire** // phase $i+1$

pour $j \leftarrow 0$ à $i+1$ **faire** // extension j

trouver la fin du chemin étiqueté par

$y[j..i]$ depuis la racine

ajouter $y[i+1]$ si nécessaire

Règles d'extension des suffixes

Durant l'extension j de la phase $i+1$, l'algorithme trouve la fin du chemin étiqueté par $y[j..i]$ depuis la racine pour, éventuellement, ajouter $y[i+1]$.

Cet ajout se fait alors en accord avec 3 règles.

Règle 1

Le chemin étiqueté par $y[j..i]$ depuis la racine se termine sur une feuille, $y[i+1]$ est alors ajouté à la fin de l'étiquette de la branche menant à la cette feuille.

Règle 2

Le chemin étiqueté par $y[j..i]$ depuis la racine ne se termine pas sur une feuille.

Aucun chemin étiqueté par $y[i+1]$ ne commence après ce chemin.

Dans ce cas une nouvelle feuille est créée avec une branche y menant étiquetée par $y[i+1]$.

Si le chemin étiqueté par $y[j..i]$ depuis la racine ne se termine pas sur un nœud alors un nouveau nœud doit être créé et la branche cassée.

Règle 3

Le chemin étiqueté par $y[j..i]$ depuis la racine ne se termine pas sur une feuille.

Un chemin étiqueté par $y[i+1]$ commence après ce chemin.

Donc $y[j..i+1]$ est déjà dans l'arbre : on ne fait rien.

Lien suffixe

On définit le lien suffixe d'un nœud interne av
par

$$s(av) = v$$

avec a une lettre et v un mot.

Lemme 1

Si un nouveau nœud interne av est ajouté à l'arbre pendant l'extension j de la phase $i+1$ alors

- soit il y a déjà un nœud interne v dans l'arbre ;
- soit un nœud interne v va être créé dans l'extension $j+1$ de la phase $i+1$.

Preuve

Un nouveau nœud interne av est créé dans l'extension j (de la phase $i+1$) uniquement lorsque la règle d'extension 2 s'applique.

Cela signifie que dans l'extension j , le chemin étiqueté par av se poursuit par une lettre différente c de $y[i+1]$.

Donc lors de l'extension $j+1$, il y a un chemin v qui se poursuit par c .

Il y a alors 2 cas.

Preuve (suite)

- Soit le chemin ν est poursuivi uniquement par la lettre c et la règle 2 crée un nœud $s(av)$ à la fin du chemin ν ;
- Soit le chemin ν est poursuivi par au moins 2 lettres différentes et dans ce cas un nœud $s(av)$ existe déjà ;

dans les 2 cas, le lemme est prouvé. □

Corollaire 2

Dans l'algorithme d'Ukkonen, tout nouveau nœud interne créé aura un lien suffixe avant la fin de l'extension suivante.

Corollaire 3

Dans un arbre implicite des suffixes T_i , s'il y a un nœud interne av alors il y a un nœud interne v .

algo EXTENSION(j)

trouver le premier nœud v au dessus de $y[j-1..i]$ qui possède un lien suffixe

soit u l'étiquette de la branche entre v et $y[j-1..i]$

si $v \neq$ racine **alors**

$v \leftarrow s(v)$

suivre le chemin étiqueté par u depuis v

utiliser les règles d'extension pour s'assurer que $y[j..i+1]$ est dans l'arbre

si un nœud w avait été créé pendant l'extension $j-1$
alors

mettre $s(w)$ à jour

Astuce 1

Les étiquettes sont représentées par des couples (position, longueur).

Lorsque l'algorithme doit suivre le chemin étiqueté par u à partir de $s(v)$ (ou la racine), puisqu'on est assuré de trouver ce chemin le parcours de ce chemin s'effectue en temps proportionnel au nombre de nœuds du chemin.

Donc le temps nécessaire pour tous les parcours de ce type est $O(n)$.

Définition

La profondeur en nœud d'un nœud interne v est le nombre de nœuds sur le chemin de la racine au nœud v .

Lemme 4

Lorsque l'algorithme d'Ukkonen emprunte un lien suffixe de v à $s(v)$, à ce moment là, la profondeur en nœud de v est au plus un plus la profondeur en nœud de $s(v)$.

Théorème 5

En utilisant les liens suffixes et l'astuce 1, une phase de l'algorithme d'Ukkonen s'exécute en $O(n)$.

Corollaire 6

En utilisant les liens suffixes et l'astuce 1, l'algorithme d'Ukkonen s'exécute en $O(n^2)$.

Observation 1

Durant une phase $i+1$, si la règle d'extension 3 s'applique à l'extension j , elle s'appliquera aussi aux extensions $j+1$ à $i+1$.

Astuce 2

Une phase $i+1$ s'arrête dès que la règle d'extension 3 s'applique lors d'une extension.

Observation 2

Dès qu'une feuille étiquetée j est créée lors d'une phase, la règle d'extension 1 s'y appliquera pour chaque extension de toutes les phases suivantes.

Astuce 3

Les branches menant aux feuilles sont étiquetées par (i, ∞) .

Feuille

Soit f l'indice de la dernière feuille créée dans l'arbre en cours de construction.

algo PHASE(i)

$j \leftarrow f$

répéter

$j \leftarrow j+1$

EXTENSION(j)

jusqu'à $j = i+1$ **ou** la règle 3 s'applique

$f \leftarrow j-1$

algo UKKONEN(y, n)

construire T_0

$f \leftarrow 0$

pour $i \leftarrow 0$ à $n-1$ **faire**

 PHASE(i)

Théorème 7

L'algorithme d'Ukkonen s'exécute en temps $O(n)$.

Détails d'implantation

Les étiquettes des branches sont stockées dans les nœuds d'arrivée.

Un nœud contient les informations suivantes :

- *parent* : le nœud parent ;
- *s* : le lien suffixe ;
- *longueur* : la longueur de l'étiquette de la branche menant au nœud ;
- *position* : la position de l'étiquette de la branche menant au nœud.

algo ARBRE-DES-SUFFIXES(y, n)

racine \leftarrow nouveau nœud

feuille \leftarrow nouveau nœud

créer la transition (*racine*, $(0, \infty)$, *feuille*)

$s(\textit{racine}) \leftarrow \textit{racine}$

dernierNœud \leftarrow *racine*

nœud \leftarrow *racine*

$g \leftarrow 0$

$j \leftarrow 0$

pour $i \leftarrow 1$ à $n - 1$ **faire**

tantque $j \leq i$ **faire**

si $g = 0$ **ou** $g = \text{longueur}(\text{nœud})$ **alors**

si CIBLE-PAR-UNE-LETTRE(nœud , $y[i]$) est définie
alors

$\text{nœud} \leftarrow \text{CIBLE-PAR-UNE-LETTRE}(\text{nœud}, y[i])$

$g \leftarrow 1$

rupture // règle 3

sinon

$\text{feuille} \leftarrow$ nouveau nœud

créer la transition (nœud , (j, ∞) , feuille)

si $s(\text{dernierNœud})$ n'est pas défini **alors**

$s(\text{dernierNœud}) \leftarrow \text{nœud}$

$\text{dernierNoeud} \leftarrow \text{nœud}$

sinon

si $y[i] = y[\text{position}(\text{nœud})+g]$ **alors**

$g \leftarrow g + 1$

rupture // règle 3

sinon

$\text{nœudParent} \leftarrow \text{parent}(\text{nœud})$

$\text{nœud} \leftarrow \text{COUPE}(\text{nœudParent}, \text{nœud}, g)$

$\text{feuille} \leftarrow \text{nouveau nœud}$

créer la transition $(\text{nœud}, (j, \infty), \text{feuille})$

si $s(\text{dernierNœud})$ n'est pas défini **alors**

$s(\text{dernierNœud}) \leftarrow \text{nœud}$

$\text{dernierNoeud} \leftarrow \text{nœud}$

si $nœud \neq racine$ **alors**

si $g = longueur(nœud)$ **et** $s(nœud)$ est défini
alors

$nœud \leftarrow s(nœud)$

$g \leftarrow longueur(nœud)$

$j \leftarrow j + 1$

continuer

$nœudParent \leftarrow parent(nœud)$

si $nœudParent \neq racine$ **alors**

$nœud \leftarrow s(nœudParent)$

sinon

$nœud \leftarrow racine$

$g \leftarrow g - 1$

$h \leftarrow i - g$

tantque $g > 0$ **faire** // **descente rapide**

$nœud \leftarrow \text{CIBLE-PAR-UNE-LETTRE}(nœud, y[h])$

$g' \leftarrow \text{longueur}(nœud)$

si $g' > g$ **alors**

rupture

sinon

$g \leftarrow g - g'$

$h \leftarrow h + g'$

si $g = 0$ **alors**

si $s(\text{dernierNœud})$ n'est pas défini **alors**

$s(\text{dernierNœud}) \leftarrow nœud$

$\text{dernierNoeud} \leftarrow nœud$

si $nœud \neq \text{racine}$ **alors**

$g \leftarrow \text{longueur}(nœud)$

$j \leftarrow j + 1$

algo COUPE(*nœudParent*, *nœud*, *g*)

$p \leftarrow \text{position}(\text{nœud})$

$\ell \leftarrow \text{longueur}(\text{nœud})$

détruire la branche (*nœudParent*, *nœud*)

nouveauNoeud \leftarrow nouveau nœud

créer la transition (*nœudParent*, (*p*, ℓ), *nouveauNœud*)

créer la transition (*nouveauNœud*, ($p+g$, $\ell-g$), *nœud*)

retourner *nouveauNœud*