

COMPOSITION D'INFORMATIQUE

Arbre des suffixes

Jean Berstel*

14 février 2001

Avertissement On attachera une grande importance à la clarté, à la précision, à la concision de la rédaction.

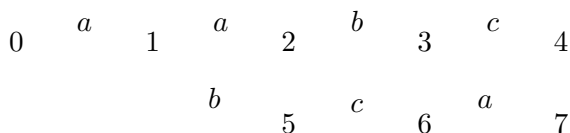
La manipulation des grandes bases de données textuelles que constituent les dictionnaires, les génomes, ou les textes circulant sur le *World Wide Web* fait appel à des structures de données élaborées et à des algorithmes efficaces. L'arbre des suffixes est utilisé dans ces contextes pour la constitution d'index, car il permet une recherche rapide.

Soit A un ensemble fini appelé *alphabet* dont les éléments sont appelés des *lettres*. Un *mot* sur A est une suite finie d'éléments de A , que l'on note par simple juxtaposition. La longueur d'un mot u est le nombre de lettres figurant dans u . Ainsi, $ababb$ est un mot de longueur 5 sur $A = \{a,b\}$. L'unique mot de longueur 0 est le mot vide, noté ε . Si u et v sont deux mots, leur produit, noté uv , est le mot obtenu en mettant u et v bout à bout. Ainsi, si $u = ban$ et $v = anes$, $uv = bananes$. Un mot u est *préfixe* (resp. *suffixe*) d'un mot v s'il existe un mot x tel que $ux = v$ (resp. $xu = v$). Un mot de longueur n possède $n + 1$ préfixes (y compris le mot vide et lui-même). Un mot u est *facteur* d'un mot v s'il existe deux mots x, y tels que $xuy = v$. Ainsi, le mot nan est facteur de $bananes$. Un préfixe (resp. suffixe) est toujours un facteur.

1 Arbres littéraux

Un *arbre littéral* est un arbre dont les arcs sont étiquetés par des lettres. Deux arcs issus d'un même sommet portent des étiquettes différentes. L'étiquette d'un chemin dans l'arbre est le mot formé des étiquettes des arcs qui le composent. Les *mots représentés* par un arbre littéral sont les étiquettes des chemins issus de la racine.

On dira qu'un sommet p et un mot x *se correspondent* si x est l'étiquette du chemin de la racine à p . Les *mots maximaux* d'un arbre littéral sont les mots qui correspondent aux feuilles.



Dans l'exemple ci-dessus, les mots représentés sont ε (le mot vide) et les mots $a, aa, ab, aab, abc, aabc, aaba$. Les mots maximaux de l'arbre sont $aabc, aaba$ et abc .

Question 1 Tracer l'arbre littéral (sans numéroter les sommets) dont les mots maximaux sont aba, abb, ba, bb, bca .

*avec les participations de Georges Gonthier et de Jean-Jacques Lévy

Un arbre littéral est représenté comme un graphe. Les sommets sont des entiers, numérotés à partir de 0. Le numéro de la racine est 0. Les arcs issus d'un sommet sont représentés par une liste dont la définition est

```
class Liste {
    char etiq;        // étiquette
    int sommet;      // sommet d'arrivée de l'arc
    Liste suivant;   // suivant dans la liste
    Liste(char e, int s, Liste x) { etiq = e; sommet = s; suivant = x; }
}
```

La représentation d'un arbre littéral est composée d'un tableau succ de listes de successeurs et de son nombre de sommets n :

```
class Arbre {
    static final int nMax = ...;
    Liste[] succ;    // listes de successeurs
    int n;          // taille
    Arbre () { n = 1; succ = new Liste[nMax]; }
}
```

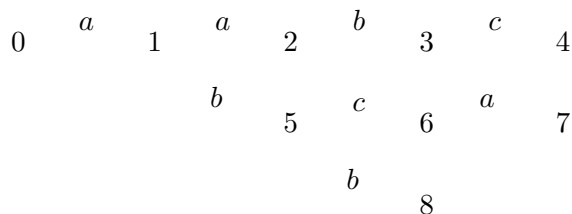
La constante nMax est supposée assez grande pour qu'il n'y ait jamais de débordement dans le tableau succ. Un arbre littéral a toujours au moins un sommet (sa racine, numérotée 0). Pour l'arbre a correspondant à l'exemple ci-dessus, on a, après construction, a.n = 8, et la liste a.succ[3] a deux éléments, dont les champs etiq et sommet valent 'c', 4 et 'a', 7 respectivement. On a aussi a.succ[4] = null par exemple.

Un mot *w* est représenté par un objet w de type String. Les seules opérations à utiliser sur le type String sont w.length() qui retourne sa longueur, et w.charAt(i) qui retourne la lettre en position i dans w. Les positions sont numérotées à partir de 0.

Question 2 Ecrire une fonction static int filsDe(int p, char c, Arbre a) qui retourne le sommet qui est l'extrémité de l'arc d'étiquette c issu de p ($0 \leq p < a.n$), si cet arc existe, et -1 sinon.

Question 3 Ecrire une fonction static int descendantDe(int p, String w, Arbre a) qui retourne le sommet qui est l'extrémité du chemin d'étiquette w issu de p ($0 \leq p < a.n$), si un tel chemin existe dans l'arbre, et -1 sinon.

Insérer un mot *w* dans un arbre littéral revient à créer les sommets et les arcs nécessaires pour que l'arbre contienne aussi le chemin issu de la racine dont l'étiquette est *w*. Ainsi, après insertion de *abb*, l'arbre de l'exemple initial devient

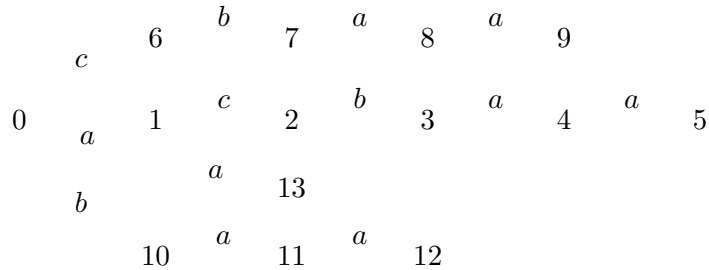


Question 4 Ecrire une fonction static void inserer(String w, Arbre a) qui insère le mot w dans l'arbre a. (Rappel: on suppose que l'arbre contient au moins un sommet).

Question 5 Ecrire une fonction static Arbre nouvelArbre(String[] x) qui retourne un arbre littéral obtenu par les insertions successives des mots du tableau x.

2 Arbre des suffixes

Etant donné un mot w , l'arbre des suffixes de w est l'arbre littéral obtenu par insertion de tous les suffixes du mot w . Il est facile de voir qu'il représente exactement l'ensemble des facteurs de w . Par exemple, l'arbre des suffixes du mot $w = acbaa$ est



On rappelle qu'un sommet p et un mot x se correspondent si x est l'étiquette du chemin de la racine à p . Un sommet est *terminal* s'il correspond à un suffixe de w . Dans la figure ci-dessus, on a doublement encerclé les sommets terminaux de l'arbre des suffixes du mot $w = acbaa$.

Question 6 Tracer l'arbre des suffixes du mot $ababbb$.

On augmente la définition de la classe `Arbre` en ajoutant un tableau booléen `terminal` de taille `nMax` avec la propriété que `a.terminal[p]` vaut `true` si et seulement si `p` est un sommet terminal dans l'arbre `a`.

Question 7 Indiquer comment modifier les fonctions des questions 4 et 5 pour obtenir:

- une fonction `static void insererSuffixe(int i, String w, Arbre a)` qui insère le suffixe du mot w commençant en position i dans l'arbre `a` (le suffixe commençant en position i de $w = a_0a_1 \cdots a_{N-1}$, où a_0, a_1, \dots, a_{N-1} sont des lettres, est le mot $a_i a_{i+1} \cdots a_{N-1}$).
- une fonction `static Arbre nouvelArbreSuffixe(String w)` qui retourne l'arbre des suffixes du mot w .

Etant donné un mot w de longueur N , on note $C_w(k)$ le nombre de facteurs distincts de longueur k de w . On a $C_w(0) = C_w(N) = 1$, et $C_w(1)$ est le nombre de lettres distinctes apparaissant au moins une fois dans w . Par exemple, si $w = aaaabcabc$, on a $C_w(3) = 5$. On note $F(w)$ le nombre total de facteurs distincts du mot w .

Question 8 On suppose que l'arbre des suffixes `a` du mot w de longueur N est calculé. Ecrire une fonction `static int [] nombreDeFacteurs(int N, Arbre a)` qui retourne un tableau `cw` tel que `cw[i] = C_w(i)` pour tout i ($0 \leq i \leq N$). Votre fonction devra être en temps proportionnel à $F(w)$.

Question 9 a) Donner une borne inférieure et une borne supérieure pour $F(w)$ en fonction de la longueur de w , et des exemples où ces bornes sont atteintes.

b) Calculer $F(a^n b^n)$ pour $n \geq 0$, où a, b sont des lettres et a^n est défini par $a^0 = \varepsilon$ et $a^{n+1} = a^n a$.

Le *degré* d'un sommet p est le nombre de ses successeurs. Si le facteur x de w correspond à p , le degré de x est le degré de p . C'est aussi le nombre de lettres a telles que xa est facteur de w . Par exemple, le mot aa est de degré 2 dans $aaab$. On note $d(x)$ le degré d'un facteur x de w .

Question 10 a) Montrer que pour tout k ($0 \leq k \leq N$), il existe au plus un facteur de w de longueur k et de degré 0.

b) Montrer que si y est suffixe de x , et x est un facteur de w , alors $d(y) \geq d(x)$.

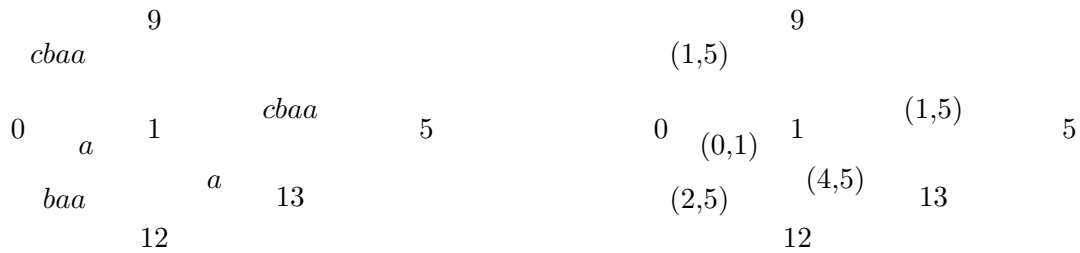
c) Démontrer que si $C_w(k) > C_w(k+1)$ pour un $k < N$, alors $C_w(k) = 1 + C_w(k+1)$.

d) Démontrer que si $C_w(k) = C_w(k+1)$ pour un $k < N-1$, alors $C_w(k+1) \geq C_w(k+2)$.

3 Compacter l'arbre des suffixes

L'inconvénient majeur de l'arbre des suffixes d'un mot est sa taille. On va le compacter en supprimant les sommets internes inutiles.

L'*arbre compact* des suffixes d'un mot w est obtenu à partir de l'arbre des suffixes en supprimant les sommets de degré 1 qui ne sont pas terminaux. Plus précisément, si (s, s_1, \dots, s_n, t) est un chemin d'étiquette x dans l'arbre des suffixes, avec s_1, s_2, \dots, s_n de degré 1 non terminaux, et s, t de degré $\neq 1$ ou terminaux, alors ce chemin est remplacé, dans l'arbre compact, par l'arc (s, t) , dont l'étiquette est le mot x . Par exemple, l'arbre compact des suffixes du mot $w = acbaa$ est donné dans la partie gauche de la figure ci-dessous (on a conservé les numéros des sommets dans l'arbre du début de la section précédente).



Soit $w = a_0a_1 \dots a_{N-1}$ un mot, avec a_0, a_1, \dots, a_{N-1} des lettres. On représente l'arbre compact des suffixes de w de façon économique en remplaçant l'étiquette x d'un arc par un couple (i, j) tel que $x = a_i a_{i+1} \dots a_{j-1}$ comme dans l'arbre ci-dessus à droite.

Question 11 Calculer l'arbre compact des suffixes du mot $w = ababbb$.

Question 12 Démontrer que l'arbre compact des suffixes d'un mot de longueur $N \geq 2$ a au plus $2N - 1$ sommets.

Pour représenter un arbre des suffixes compact, à chaque arc est maintenant associé un intervalle semi-ouvert décrivant l'étiquette. Les nouvelles classes pour les listes et arbres sont donc

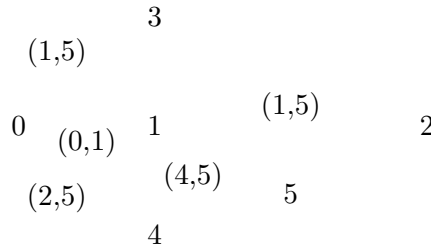
```
class ListeA {
    int debut, afin; // intervalle semi-ouvert [debut, après-la-fin[
    int sommet;     // sommet d'arrivée de l'arc
    ListeA suivant; // suivant dans la liste
    ListeA(int d, int a, int s, ListeA x) {
        debut = d; afin = a; sommet = s; suivant = x; }
}

class ArbreA {
    static final int nMax = ...;
    ListeA[] succ; // listes de successeurs
    boolean[] terminal; // marques des sommets terminaux
    int n; // taille
    ArbreA () { n = 1; succ = new ListeA[nMax]; terminal = new boolean[nMax]; }
}
```

Question 13 Ecrire une fonction `static int longueurMot(Arbre a)` qui retourne la longueur du mot dont `a` est l'arbre des suffixes.

Question 14 Ecrire une fonction `static ArbreA transformer(Arbre a)` qui retourne l'arbre compact correspondant à l'arbre des suffixes `a`. (On conservera les mêmes numéros pour les sommets des deux arbres et on ne se servira que de la longueur du mot dont `a` est l'arbre des suffixes).

Question 15 Ecrire une fonction `static ArbreA epurer(ArbreA a)` qui retourne un arbre compact équivalent à l'arbre compact `a` avec des numéros de sommets consécutifs (comme dans la figure suivante).



4 Construction directe de l'arbre compact des suffixes

Dans cette partie, on construit l'arbre compact des suffixes sans passer par l'arbre littéral. On gagne en place, mais le temps est encore quadratique.

L'algorithme procède par insertions des suffixes par longueur décroissante dans un arbre initialement réduit à un sommet. Durant l'insertion d'un suffixe, on détecte le plus long préfixe figurant déjà dans l'arbre, et on ajoute, si nécessaire, un nouvel arc étiqueté par la totalité du mot restant. L'exploitation du préfixe présent dans l'arbre nécessite parfois de briser un arc en deux.

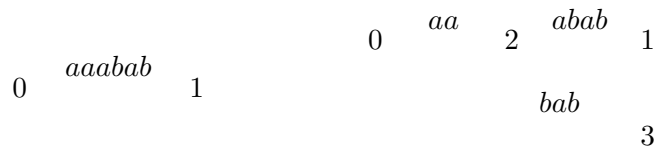


FIG. 1 – *A gauche, après insertion de aaabab, et à droite après insertion de aabab*

Voici un exemple. Soit $w = aaabab$. Après insertion du mot, on obtient l'arbre à gauche dans la figure 1. Pour le suffixe $= aabab$, le plus long préfixe dans l'arbre est aa . On *brise* l'arc $(0,aaabab,1)$ en deux arcs $(0,aa,2)$ et $(2,abab,1)$, et on ajoute un nouvel arc $(2,bab,3)$. On obtient alors l'arbre à droite dans la figure 1.

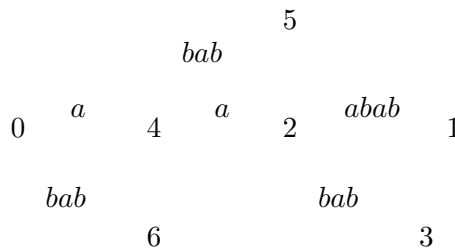


FIG. 2 – *Après aabab et bab.*

Pour l'insertion du suffixe $abab$, on brise l'arc $(0,aa,2)$ en $(0,a,4)$ et $(4,a,2)$ et on ajoute $(4,bab,5)$. L'insertion de bab ne provoque pas de bris, et se solde par l'ajout de l'arc $(0,bab,6)$. On obtient donc l'arbre de la figure 2.

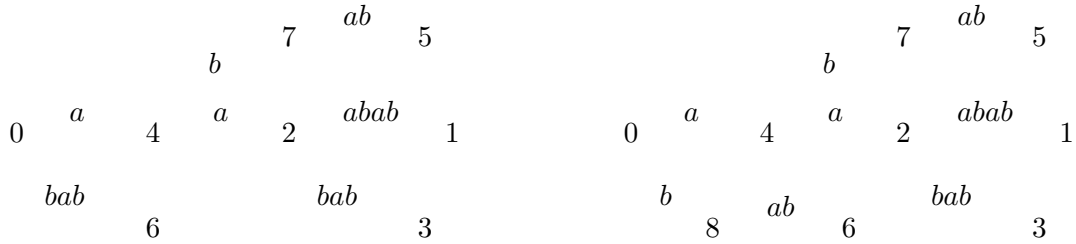


FIG. 3 – A gauche après ab , et à droite l'arbre final

L'insertion de ab provoque le bris de l'arc $(4,bab,5)$ sans création d'une feuille supplémentaire (figure 3 à gauche). L'arbre final est dessiné dans la figure 3, à droite.

On se donne un mot $w = a_0a_1 \cdots a_{N-1}$ avec $N > 0$ et a_0, a_1, \dots, a_{N-1} des lettres.

Question 16 Ecrire une fonction `static int lpc(int i, int j, int k, int l, String w)` qui retourne la longueur du plus long mot qui est préfixe à la fois du mot $a_i a_{i+1} \cdots a_{j-1}$ et du mot $a_k a_{k+1} \cdots a_{l-1}$.

On utilise les structures introduites dans la partie précédente, à savoir les classes `ArbreA` et `ListeA`.

Question 17 Ecrire une fonction `static void insererSuffixe(int i, String w, int p, ArbreA a)` qui insère le suffixe de w commençant en position i dans l'arbre a à partir du sommet p .

Question 18 Ecrire une fonction `static ArbreA nouvelArbre(String w)` qui retourne l'arbre des suffixes compact du mot w en appliquant l'algorithme décrit plus haut.

5 Vers la construction de l'arbre compact en temps linéaire

Dans cette partie, on amorce la construction de l'arbre compact en temps et en place linéaires.

Question 19 Soit w un mot, et soit x un facteur de w . Montrer que si x correspond à un sommet p dans l'arbre des suffixes de w , on peut calculer le sommet p en temps $O(E \times k)$ où E est le nombre d'arcs du chemin de la racine à p et k est le nombre de lettres de l'alphabet considéré.

Question 20 Soit p un sommet de l'arbre des suffixes compact de w autre que la racine, et soit x le facteur correspondant à p . Posons $x = ay$, où a est la première lettre de x . Montrer que y correspond à un sommet q de l'arbre.

On appelle *lien suffixe* de p le sommet q défini dans la question précédente.

Question 21 Ecrire une fonction `static int[] lienSuffixe(String w, ArbreA a)` qui retourne, dans un vecteur, les liens suffixes des sommets de l'arbre a des suffixes de w .

Weiner (1973), McCreight (1976) et Ukkonen (1992) ont trouvé un algorithme pour construire l'arbre des suffixes en un temps linéaire. L'algorithme de Weiner fut considéré comme l'algorithme de l'année 1973 par Knuth.