Deuxième partie

Recherche de motifs dans une arborescence

Chapitre 7

Recherche de motifs dans une arborescence

Ce chapitre est une introduction au problème de la recherche de motifs dans une arborescence. Après quelques définitions (section 7.1), nous nous intéressons au cas des mots et nous présentons deux structures de données classiques, l'automate de Aho-Corasick et l'arborescence des suffixes, que nous utilisons par la suite. Nous poursuivons par une rapide revue des principaux articles parus sur le sujet de la recherche de motifs dans une arborescence, avant de conclure par l'extension d'un de ces algorithmes dû à Hoffmann et O'Donnell [91].

Notation. Dans la suite, toutes les complexités considérées, sauf mention contraire, sont relatives à la complexité dans le pire des cas.

7.1 Introduction

Dans cette seconde partie de notre mémoire, nous considérons des arborescences ordonnées et étiquetées sur un alphabet Σ fixé, que nous désignons par le terme arborescences sur Σ, définies récursivement de la manière suivante.

Définition 167. Une arborescence A sur Σ est constituée d'un sommet, appelé racine de A, muni d'une étiquette appartenant à Σ, relié à un *ensemble totalement ordonné* (éventuellement vide) d'arborescences sur Σ.

On remarque que contrairement aux arborescences considérées dans la première partie de ce mémoire, une même étiquette peut apparaître plusieurs fois dans une arborescence sur Σ.

Définition 168. Soit A une arborescence et x un noeud de A. Si x est relié à la liste $A_1, \ldots ,A_k$ d'arborescences de racines respectives $x_1, \ldots , x_k$, on appelle x_i le $i^{ème}$ fils de x. On dit que x_i est un frère gauche (resp. droit) de x_j si et seulement si $i < j$ (resp. $i > j$).

Recherche de motifs dans une arborescence quelconque. Soient M et T deux arborescences. Intuitivement, on dit que T (le *texte*) possède une occurrence de M (le *motif*) en un sommet x si, en plaçant la racine de M en x, on arrive à une situation dans laquelle tout sommet (resp. arête) de M recouvre un sommet (resp. arête) de T ayant la même étiquette. On traduit formellement cette notion de la façon suivante.

Définition 169. Soient M et T deux arborescences sur Σ, et x un sommet de T. Le texte T possède une *occurrence de M en x* si et seulement si il existe une correspondance entre les sommets de M et ceux de T telle que :

1. tout sommet de M est en correspondance avec un sommet de T ;
2. la racine de M est en correspondance avec x ;
3. si un sommet y de M est en correspondance avec un sommet z de T, y et z ont la même étiquette ;
4. un noeud y de M est en correspondance avec un noeud z de T, alors chaque fils de y correspond à un fils de z ;
5. si y_1 et y_2 sont deux sommets de M tels que y_1 est un frère gauche (resp. droit) de y_2, et si z_1 et z_2 sont les sommets de T correspondant respectivement à y_1 et y_2, alors z_1 est un frère gauche (resp. droit) de z_2.

Par exemple, dans la figure suivante, le texte T comporte une seule occurrence du motif M ("matérialisée" par les arêtes en pointillé).

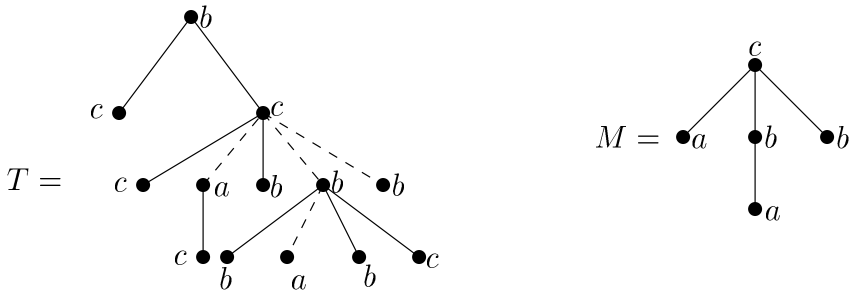


FIG. 7.1 – Illustration de la notion d'occurrence d'un motif dans un texte

Définition 170. Le problème de la recherche de motifs dans une arborescence, que nous appelons problème RMA, consiste, étant données deux arborescences T et M, à trouver tous les sommets x de T tels que T admette au moins une occurrence de M en ce sommet x.

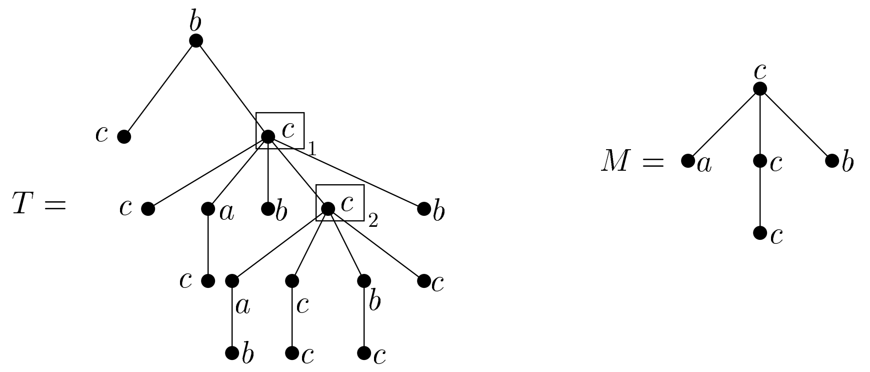


FIG. 7.2 – Le motif M apparaît deux fois dans le texte (en les sommets encadrés)

Cependant, comme nous le verrons dans la section suivante, les différents travaux traitant de recherche de motifs dans une arborescence s'intéressent à une restriction de ce problème, basée sur la notion d'*occurrence compacte d'un motif dans un texte.*

Définition 171. Soient M et T deux arborescences sur Σ, et x un sommet de T. On dit que le texte T possède une *occurrence compacte de M en x* si et seulement si il existe une correspondance entre les sommets de M et ceux de T telle que :

1. tout sommet de M est en correspondance avec un sommet de T ;
2. la racine de M est en correspondance avec x ;
3. si un sommet y de M est en correspondance avec un sommet z de T, y et z ont la même étiquette ;
4. si le nœud y de M correspond au nœud z de T, alors le $i^{ème}$ fils de y correspond au $i^{ème}$ fils de z.

On appelle alors *recherche de motifs compacts* dans une arborescence (noté RMCA) le problème RMA dans lequel on ne cherche à détecter que les occurrences compactes du motif dans le texte. Par exemple, dans la figure 7.2 seule la l'occurrence de M dans T (de racine le sommet encadré indicé 2) est une occurrence compacte.

Cette version restreinte du problème de la recherche de motifs a reçu plus d'attention que le problème général, en partie car elle peut être reformuler immédiatement en termes d'arborescences binaires, qui se prêtent mieux à un traitement algorithmique que les arborescences quelconques.

Définition 172. Une arborescence binaire A sur Σ est une arborescence dans laquelle tout noeud a un ou deux fils, appelés ses fils gauche et droit : dans le cas où un sommet x de A n'a qu'un seul fils, ce dernier est soit fils gauche de x, soit fils droit de x.

Il existe une bijection classique entre les arborescence sur Σ à n sommets et les arborescences binaires sur Σ à n sommets telles que la racine n'a qu'un fils gauche. Étant donnée une arborescence A sur Σ, cette bijection consiste à disposer la liste des fils $\{y_1,y_2,\dots,y_p\}$ d'un sommet x sur une même branche droite de sorte que

- y_1 soit fils gauche de x,
- y_{i+1} soit fils droit de y_i pour $1 \leq i < p$.

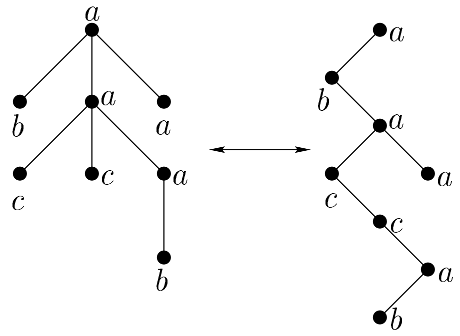


FIG. 7.3 – La transformation fondamentale de Knuth

La construction inverse est immédiate et le caractère bijectif de cette transformation (souvent appelée *transformation fondamentale de Knuth* [102, section 2.3.2]) se vérifie sans difficulté, tout comme la propriété suivante.

Propriété 173. Soient T et M deux arborescences sur Σ, T' et M' les arborescences binaires sur Σ correspondantes, x un sommet de T et x' le sommet correspondant de T'. Il y a une occurrence compacte de M en x si et seulement il y a une occurrence compacte de M' en y.

Cette propriété permet de ramener le problème RMCA pour les arborescences quelconques au problème équivalent pour les arborescences binaires telles que la racine a un fils gauche. Par contre, on vérifie immédiatement que le problème RMA ne possède pas de traduction directe en termes d'arborescences binaires.

Recherche de motifs dans un terme. La notion de *terme* (ou *arborescence d'expression*) est une extension de la notion d'arborescence, qui apparaît notamment dans les langages de programmations, les systèmes de preuve automatique, ou plus généralement les systèmes de réécriture. On considère dans ce cas chaque symbole s de Σ comme un symbole fonctionnel auquel est associé une *arité* notée $\nu(s)$.

Définition 174. Un terme sur (Σ,ν) est une arborescence sur Σ telle que chaque sommet x a un nombre de fils qui est exactement égal à l'arité $\nu(e(x))$ de son étiquette $e(x)$ appartenant à Σ.

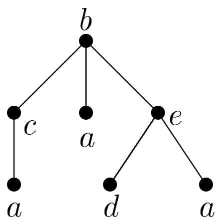


FIG. 7.4 – Un terme avec $\nu(a) = \nu(d) = 0$, $\nu(c) = 1$, $\nu(e) = 2$ et $\nu(b) = 3$

Dans le cas des termes, on le fait que l'arité d'un sommet est fixée par son étiquette implique que le problème RMA est équivalent au problème RMCA moyennant l'ajout d'une étiquette dite "variable" pour le motif M.

Définition 175. Un *motif* sur (Σ,ν) est un terme sur $(\Sigma\cup\{v\},\nu)$ (on appelle v la *variable*), avec $\nu(v) = 0$, et un *texte* sur (Σ,ν) est un terme sur (Σ,ν).

On définit alors la notion d'occurrence d'un motif M dans un terme T en remplaçant la condition 2 de la définition 169 (ou de la définition 171, les deux définitions devenant alors équivalentes pour les termes) par

2. si un sommet y de M est en correspondance avec un sommet z de T, soit y et z ont même étiquette, soit l'étiquette de y est v.

Dans la figure suivante, on a $\Sigma = \{a,b,c,d\}$ et $\nu(a) = 4$, $\nu(b) = 1$ et $\nu(c) = \nu(d) = 0$.

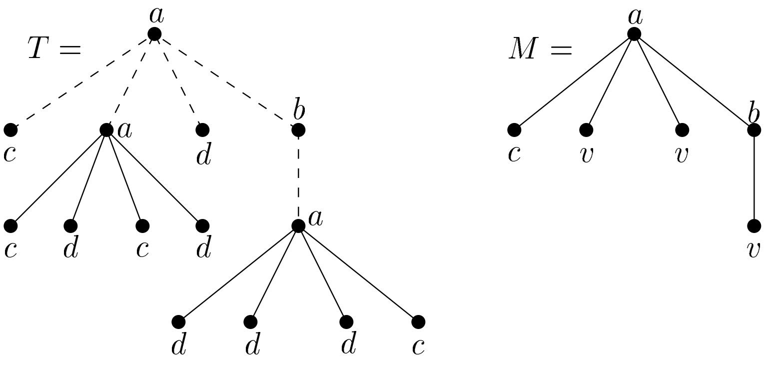


FIG. 7.5 – Une occurrence de M dans T (en la racine)

7.2 Recherche de motifs dans un mot

Soient M et T deux mots sur un alphabet Σ. Rechercher les occurrences du motif M dans le texte T consiste à repérer l'ensemble des facteurs de T identiques à M. Du fait de ses nombreuses applications pratiques (logiciels pour l'édition de texte, analyse de séquences biologiques, commandes de systèmes d'exploitation, etc), ce problème a été largement étudié comme l'illustre l'ouvrage de synthèse de Crochemore et Rytter [52].

Un mot de longueur n pouvant naturellement être considéré comme une arborescence à n sommets dont chaque noeud a exactement un fils (une telle arborescence comporte donc une seule feuille), le problème de la recherche de motifs dans une arborescence est une extension naturelle du problème restreint aux mots. Comme il existe plusieurs manières de coder une arborescence par un ensemble de mots, la plupart des algorithmes résolvant le problème RMA sont basés sur des algorithmes ou des structures de données issus du problème de la recherche de motifs dans les mots.

On peut diviser les algorithmes opérant sur les mots en deux familles. La première regroupe les algorithmes basés sur un prétraitement du motif M qui permet d'accélérer le parcours du texte T. Ce principe est notamment adapté au cas où le texte n'est pas fixé et ne peut donc être prétraité. On dit alors que le texte est *dynamique*. Ce sont des algorithmes appartenant à cette famille qui sont implantés dans les logiciels d'édition de texte ou la commande `grep` du système UNIX par exemple (les algorithmes les plus célèbres de cette nature sont dûs à Knuth, Morris et Pratt [104], Aho et Corasick [1], Boyer et Moore [26] et permettent, après un prétraitement en temps et espace $O(|M|)$ du motif M, de détecter les occurrences de ce motif en temps $O(|T| + |M|)$).

La seconde famille regroupe les algorithmes opérant sur un texte fixé (dit *statique*). Ils sont basés sur un prétraitement du texte T, consistant à construire une structure de données (occupant un espace en $O(|T|)$) indexant les suffixes de T, à partir de laquelle on peut rechercher rapidement (en temps linéaire ou quasi-linéaire par rapport à $|M|$) les occurrences du motif M dans T. Les structures de données classiques utilisées dans ce cas sont l'arborescence des suffixes [118], l'automate des suffixes [20, 51] ou encore le tableau des suffixes [117].

Dans les deux paragraphes qui suivent, nous présentons deux structures de données utilisées pour résoudre le problème de la recherche de motifs dans les mots. Nous utiliserons ces structures de données dans les algorithmes de recherche de motifs dans une arborescence décrites dans la dernière section de ce chapitre et dans les deux derniers chapitres.

On rappelle qu'on appelle *préfixe* (resp. *suffixe*) d'un mot M tout mot M_1 tel qu'il existe un mot M_2 vérifiant $M = M_1M_2$ (resp. $M = M_2M_1$). Si $M_1 \neq M$, on dit que M_1 est un préfixe (resp. suffixe) *propre* de M. Par exemple, *abca* est un préfixe propre de *abcacba* et un suffixe propre de *acbabca*.

7.2.1 Le cas dynamique et l'automate de Aho-Corasick

Cet algorithme aborde le problème plus général consistant à se donner un ensemble $\{M_1, \dots, M_k\}$ de mots sur Σ (les motifs) et à rechercher toutes les occurrences de l'un des motifs $M_1, \dots, M_k$ dans le texte T (voir [52, Section 7.1] et [1]). Le prétraitement des motifs $\{M_1, \dots, M_k\}$ est basé sur une structure de donnée classique, appelée *arbre digital de recherche* (que nous notons ADR).

Définition 176. Soit $\{M_1, \dots, M_k\}$ un ensemble de mots sur Σ. L'arborescence digitale de recherche (ADR) associée à $\{M_1, \dots, M_k\}$ est l'arborescence dont les arêtes sont étiquetées par des lettres de Σ et vérifiant :

- deux arêtes issues d'un même sommet ont des étiquettes différentes,
- tout chemin dans l'ADR débutant à la racine est étiqueté par un mot préfixe de l'un des mots $\{M_1, \dots, M_k\}$,
- tout mot de $\{M_1, \dots, M_k\}$ étiquette un chemin débutant à la racine de l'ADR.

Pour chaque sommet u de cette ADR, on note $m(u)$ le mot formé des étiquettes des arêtes du chemin allant de la racine à u (on dit alors que le sommet u reconnaît $m(u)$ et que u est un *sommet final* si il existe i tel que $M_i = m(x)$). De plus, en orientant les arêtes de la racine vers les feuilles, on transforme cette ADR en un automate déterministe (mais non forcément complet ou minimal) reconnaissant le langage formé des mots $M_1, \dots, M_k$: on parle d'état au lieu de sommets (la racine devient l'état initial et les sommets finaux, les états finaux) et de transitions au lieu d'arêtes.

Le prétraitement de $\{M_1, \dots, M_k\}$ comporte deux étapes. Tout d'abord, on construit l'automate correspondant à l'ADR associée à $\{M_1, \dots, M_k\}$, que l'on complète ensuite. Ainsi, pour chaque état u et pour chaque lettre $a \in \Sigma$ telle qu'aucune transition étiquetée a ne quitte u, on ajoute une transition (étiquetée a) de u vers l'état reconnaissant le plus long suffixe propre du mot $m(u).a$ (qui est donc aussi préfixe d'un des M_i). On note $AC(M_1,\ldots,M_k)$ l'automate ainsi construit.

Dans la figure suivante, on représente l'état de $AC(M_1,\ldots,M_k)$) (resp. le sommet de l'ADR associé à $\{M_1,\ldots,M_k\}$) reconnaissant M_i par un carré contenant l'entier i et les transitions ajoutées lors de la seconde phase de prétraitement en pointillé.

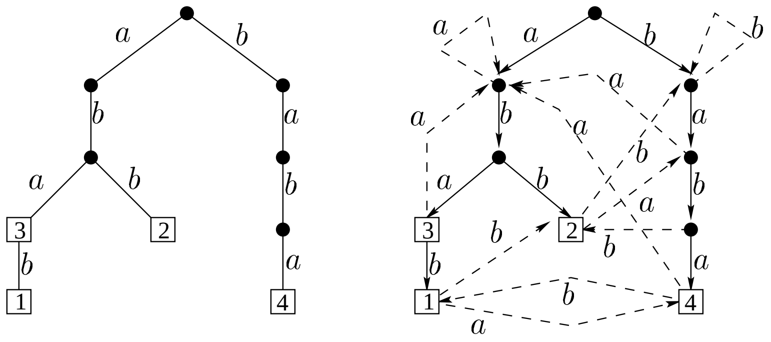


FIG. 7.6 – L'ADR de ($abab$, abb, aba, $baba$) et l'automate $AC(abab, abb, aba, baba)$

Pour rechercher les positions des occurrences des motifs $(M_1,\ldots,M_k)$ dans un texte T, il suffit d'appliquer l'algorithme suivant (on note $T[i]$ la $i^{ème}$ lettre d'un mot T).

Algorithme 9: Recherche de motifs avec l'automate de Aho-Corasick.

Données : $AC(M_1,\ldots,M_k)$ et T de longueur n

début

Soit r l'état initial de $AC(M_1,\ldots,M_k)$ et $u := r$;

pour i allant de 1 à n **faire**

$u := \delta(u,T[i])$;

si x est un état final étiqueté j **alors**

Afficher "Occurrence de M_j en position $i - |M_j| + 1$";

fin

Il reste à étudier la complexité en temps et en espace de cet algorithme (on pose $|T| = n$ et $\sum_i |M_i| = m$). La phase de recherche de motifs s'effectue clairement en temps $O(n)$ (l'automate étant complet, on peut stocker les transitions quittant un état par un tableau indexé par les lettres de Σ assurant ainsi un accès en temps $O(1)$ à chacun de ces arcs). L'automate $AC(M_1,\ldots,M_k)$ occupe un espace en $O(m \times |\Sigma|)$, et Aho et Corasick ont proposé un algorithme permettant de construire cet automate temps $O(m \times |\Sigma|)$ [1].

Remarque 177. Pour être plus précis, on remarque que si l'ADR associée à $\{M_1,\ldots,M_k\}$ comporte p sommets, la transformation de cette ADR en l'automate $AC(M_1,\ldots,M_k)$ s'effectue en temps $O(p \times |\Sigma|)$.

Remarque 178. Dans le cas d'un alphabet Σ de taille importante, un automate complet occupe souvent trop de place. On peut cependant ramener cet espace à $O(m)$ en ne complétant pas l'automate correspondant à l'ADR et en organisant les transitions quittant un sommet dans une arborescence binaire de recherche équilibrée par exemple (voir [50] pour une présentation de cette structure de données). En calculant pour chaque état

x une transition spéciale (car non étiquetée) reliant u à l'état v tel que $m(v)$ est le plus grand suffixe propre de $m(u)$ reconnu par l'automate, on peut alors effectuer la phase de recherche de motifs en temps $O(n \times \log(|\Sigma|))$. Cependant, cette technique ne peut être employée efficacement dans le cas de la recherche de motifs dans une arborescence, comme nous le verrons dans les deux prochaines sections. Pour cette raison nous privilégions cette version utilisant un automate complet.

Notation. Dans la suite de ce mémoire, nous appelons *structure de branchement* la structure de données utilisée pour stocker des transitions étiquetées sur Σ dans un automate ou une arborescence aux arêtes étiquetées, cette structure pouvant être de deux types (dans les applications qui nous intéressent) :

- des tableaux indexés par les lettres de Σ (chaque état/sommet nécessite un espace en $O(|\Sigma|)$ pour stocker les transitions/arêtes le quittant et l'accès à une transition/arête s'effectue en temps $O(1)$),
- des arborescences binaires de recherche équilibrées (un état/sommet ayant k transitions/arêtes le quittant nécessite un espace en $O(k)$ et l'accès à chaque transition/arête s'effectue en temps $O(\log(|\Sigma|))$).

7.2.2 Le cas statique et l'arborescence des suffixes

Arborescence des suffixes. Soit T un mot de longueur n sur Σ. On note $Suf(T) = \{T[1,n], \ldots, T[n,n]\}$ (où $T[i,j]$ est le facteur de T commençant en position i et finissant en position j) l'ensemble des suffixes de T, ordonnés par longueur décroissante. Par exemple, pour $T = acbab$, on a $Suf(T) = \{acbab, cbab, bab, ab, b\}$.

Définition 179. L'arborescence des suffixes (AS) d'un mot T, notée $AS(T)$, est obtenue en comprimant chaque chemin de l'ADR de $Suf(T)$ ne comportant que des sommets non finaux ayant un seul fils en une seule arête étiquetée par le facteur de T correspondant au chemin ainsi comprimé. On obtient ainsi une arborescence dans laquelle chaque arête est étiquetée par un facteur $T[i,j]$ de T qui peut donc être codée par le couple d'entiers (i,j) appelé désormais l'étiquette de l'arête.

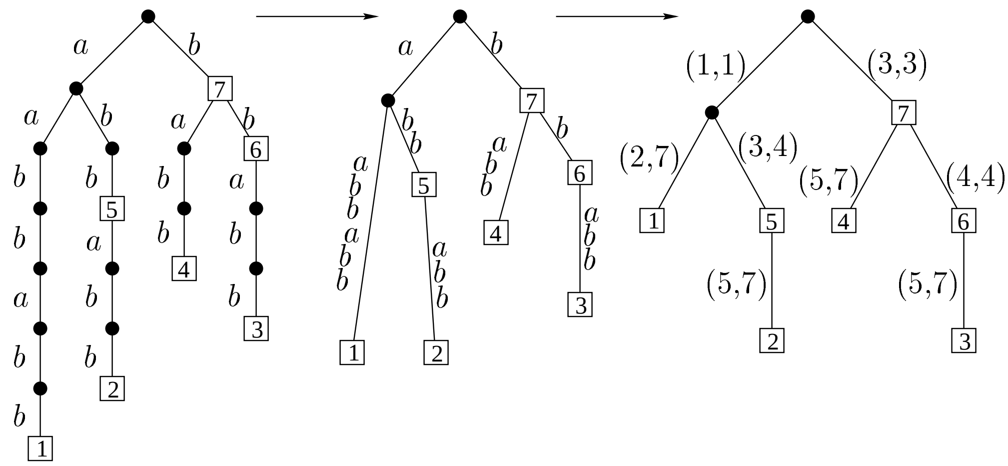


FIG. 7.7 – De l'ADR de $Suf(aabbabb)$ à l'arborescence des suffixes de $aabbabb$

Recherche de motifs. Dans le cas statique, le prétraitement de l'algorithme de recherche d'un motif dans un mot T (le texte fixé) consiste alors à construire l'arborescence des suffixes $AS(T)$. La phase de recherche consiste, étant donné un motif M, à appliquer l'algorithme suivant.

Algorithme 10: Recherche de motif avec l'arborescence des suffixes.

Données : T, $AS(T)$ et M
début
  soit r la racine de $AS(T)$, $u := r$ et $i := 1$;
  tant que $i \leq |M|$ **faire**
    si il existe une arête (j,k) (allant de u vers v) telle que $T[j] = M[i]$ **alors**
      $l := j$;
      tant que $i \leq |M|$, $l \leq k$ et $T[l] = M[i]$ **faire** $l := l + 1$ et $i := i + 1$;
1   **si** $i > |M|$ **alors** parcourir la sous-arborescence de $AS(T)$ de racine v pour obtenir toutes les positions des occurrences de M dans T (elles sont données par les sommets finaux de cette sous-arborescence);
      sinon si $l > k$ **alors** $u := v$ **sinon** sortir;
    sinon sortir;
fin

En remarquant que l'arborescence des suffixes comporte exactement $n = |T|$ sommets finaux et que tout nœud non final possède au moins deux fils, on peut affirmer que $AS(T)$ a au plus $2n$ sommets. Tout comme pour l'automate AC introduit précédemment, la complexité en espace de $AS(T)$ dépend de la structure de branchement utilisée. L'utilisation de tableaux indexés par les lettres de Σ implique une complexité en espace en $O(n \times |\Sigma|)$, alors qu'utiliser des arborescences binaires de recherche équilibrées entraîne une complexité en espace en $O(n)$ [1]. La phase de recherche de motifs proprement dite (l'algorithme 10) ne nécessite aucun espace additionnel et, si il y a k occurrences de M dans T, elle s'effectue clairement en temps $O(m + k)$ ou $O(m \times \log(|\Sigma|) + k)$ selon la structure de branchement utilisée (le terme k provient du fait qu'une fois un mot reconnu, la sous-arborescence de $AS(T)$ parcourue lors de l'étape 1 a au plus k sommets finaux et donc au plus $2k$ sommets). Il reste à examiner la complexité en temps du prétraitement du texte T, c'est-à-dire la complexité en temps de la construction de l'arborescence des suffixes $AS(T)$. Il existe plusieurs algorithmes permettant de construire l'arborescence des suffixes d'un mot de longueur n sur Σ en temps $O(n \times \log(|\Sigma|))$ si on utilise des arborescences binaires de recherche équilibrées, comme nous le verrons dans le paragraphe suivant et le chapitre 9).

Quelques rappels sur l'arborescence des suffixes. L'arborescence des suffixes, introduite par Weiner [164], s'est imposée comme une structure de données fondamentale en algorithmique des mots (comme le montrent par exemple les articles de synthèse d'Apostolico [5, 6]). Dans le paragraphe précédent, nous avons vu son utilité dans le cadre de la recherche de motifs dans un mot. L'arborescence des suffixes s'avère aussi utile pour

1. Gusfield [86] remarque que, dans la plupart des cas, les nœuds situés à faible distance de la racine peuvent avoir un degré important alors que les autres nœuds ont un degré faible. Il semble donc intéressant de mélanger les structures de branchement au sein d'une même arborescence pour obtenir un compromis entre espace et temps de branchement.

résoudre un problème proche, la recherche de motifs approchés [32, 159, 45, 109]. On peut aussi remarquer que l'arborescence des suffixes d'un mot T permet de trouver les facteurs de T qui se répètent : si un noeud u de $AS(T)$ a, parmi ses descendants, k sommets finaux, on en déduit que le facteur $m(u)$ apparaît k fois dans T (ces occurrences pouvant cependant se chevaucher). Cette propriété peut s'avèrer extrêmement utile pour la compression de textes [133].

Le premier algorithme non trivial pour la construction de l'arborescence des suffixes d'un mot T de longueur n est dû à Weiner [164]. Cet algorithme construit l'arborescence des suffixes $AS(T)$ incrémentalement en insérant les suffixes de T dans une ADR initialement vide, du plus petit au plus long suffixe, en temps et espace $O(n \times |\Sigma|)$. Par la suite, McCreight [118] a proposé un nouvel algorithme, basé sur un principe analogue à celui de Weiner, mais en insérant les suffixes de T du plus long au plus petit, ce qui a pour effet de construire $AS(T)$ en temps $O(n \times \log(|\Sigma|))$. Nous présentons en détail cet algorithme dans le prochain chapitre. Plus récemment, Ukkonen [160] a proposé un algorithme élégant, linéaire en temps, qui présente l'avantage d'être "en ligne" : l'arborescence des suffixes $AS(T)$ est construite incrémentalement au fur et à mesure que l'on lit le texte T. Dans un article de synthèse, Giegerich et Kurtz [71] ont unifié la présentation de ces trois algorithmes. Enfin, parmi les algorithmes de construction d'une arborescence des suffixes, on peut aussi noter les travaux d'Apostolico et al. [8] proposant un algorithme parallèle de construction de $AS(T)$, qui, dans le cas séquentiel, est linéaire en temps (modulo le temps de branchement), et de Farach [60], qui donne le premier algorithme indépendant de la taille de l'alphabet Σ à condition que Σ soit l'alphabet constitué des symboles $\{1,2,\ldots,n\}$.

Plus récemment, plusieurs auteurs se sont intéressés aux propriétés pratiques de cette structure de données. En effet, bien qu'elle ait des propriétés théoriques optimales, l'arborescence des suffixes s'avère souffrir de certaines limitations pratiques. Par exemple, pour un texte anglais classique, l'espace occupé en moyenne par l'arborescence des suffixes est d'environ 15 octets par lettre du texte initial. Le tableau des suffixes, proposé par Manber et Myers [117] et le cactus des suffixes défini par Kärkkäinen [98] semblent être des alternatives à l'arborescence des suffixes ayant un intérêt d'un point de vue pratique. Dans ce même domaine, on peut aussi se référer aux travaux d'Andersson [4], qui propose une implantation efficace de l'arborescence des suffixes, ou à ceux de Giegerich et Kurtz [70]. Toujours dans une optique d'optimisation de l'espace occupé par l'arborescence des suffixes, certains auteurs se sont intéressés au cas où on ne veut indexer qu'un sous-ensemble des suffixes d'un texte (par exemple dans le cas d'un texte composé de mots) et ont proposé des algorithmes efficaces pour ce problème [3, 99].

7.3 Résultats connus pour les arborescences

Dans ce paragraphe, nous passons rapidement en revue les principaux travaux et résultats obtenus pour le problème de la recherche de motifs dans une arborescence. Le but de cette section n'est pas tant de décrire dans le détail les algorithmes connus, ces derniers étant souvent assez complexes, mais de mettre en avant les principes sur lesquels ils reposent.

Notation. Par la suite, n représente le nombre de sommets du texte T et m le nombre de sommets du motif M. De plus, pour un sommet x d'une arborescence A, on note A_x la sous-arborescence de A de racine x et $e(x)$ son étiquette.

L'algorithme naïf. L'algorithme le plus naturel (aussi appelé *algorithme naïf* pour détecter les occurrences d'une arborescence M dans une arborescence T est basé sur une procédure récursive de comparaison d'arborescences.

Algorithme 11: Algorithme naïf

Données : M et T
début
  pour Chaque sommet x de T **faire**
    si Comparer(M**,** T_x**)** = Vrai **alors** afficher "Occurrence de M en x";
fin

Algorithme 12: Comparer

Données : deux arborescences A et A'
début
1 soient x la racine de A et y celle de A';
  si $e(x) \neq e(y)$ **alors retourner** Faux;
  sinon
2   soient $x_1, \ldots, x_k$ les fils de x et $y_1, \ldots, y_l$ ceux de y, $i := 1$ et $j := 1$;
    tant que $i \leq k$ et $j \leq l$ **faire**
      si Comparer(A_{x_i}**,** A'_{y_j}**)** = Vrai **alors** i := i+1;
3      $j := j + 1$;
    si $i = k + 1$ **alors retourner** Vrai **sinon retourner** Faux;
fin

Pour traiter le problème RMCA, on peut simplifier la procédure **Comparer** en remplaçant le bloc situé entre les lignes 2 et 3 par le bloc suivant.

---

soient $x_1, \ldots, x_k$ les fils de x, $y_1, \ldots, y_l$ ceux de y et $i := 1$;
tant que $i \leq k$, $i \leq l$ et **Comparer(**A_{x_i}**,** A'_{y_i}**)** = Vrai **faire** $i := i + 1$;

La complexité en temps de cet algorithme est déterminée, dans les deux cas, par le nombre d'appels à la procédure **Comparer**, ou de manière équivalente, par le nombre de comparaisons entre les étiquettes des sommets de T et celles des sommets de M (ligne 1 de l'algorithme 12).

Définition 180. Soient A une arborescence et x un sommet de A. On appelle *hauteur de* x, noté $h(x)$, le nombre de sommets situés sur le chemin allant de la racine de A à x. On appelle hauteur de A, noté $h(A)$, le nombre de sommets présents sur le plus long chemin reliant la racine de A à une de ses feuilles.

Proposition 181. [54] *Soient T et M deux arborescences de tailles respectives n et m. L'algorithme naïf détecte toutes les occurrences (compactes ou non) de M dans T en temps $O(n \times h(M))$.*

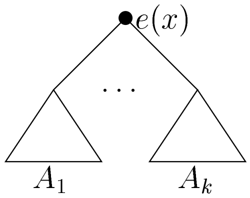
Flajolet et Steyaert [149], dans un article illustrant l'utilisation de techniques de combinatoire analytique pour l'analyse en moyenne d'algorithmes, ont prouvé le résultat suivant.

Proposition 182. *Soient T et M deux termes de tailles respectives n et m. L'algorithme naïf détecte toutes les occurrences (compactes ou non) de M dans T en temps $O(n+m)$* en moyenne.

Remarque 183. Dans le problème restreint qui consiste à rechercher l'occurrence de M en un sommet donné x de T, la complexité en temps de l'algorithme naïf dépend fortement de la notion d'occurrence prise en compte (compacte ou non). Pour le cas d'une occurrence compacte, l'algorithme naïf résoud ce problème en temps $O(m)$, alors que dans le cas d'une occurrence quelconque, la complexité en temps est en $O(|T_x|)$.

L'algorithme montant de Hoffmann et O'Donnell. Les premiers algorithmes non triviaux pour la recherche de motifs dans une arborescence sont dûs à Hoffmann et O'Donnell [91]. Dans cet article, les auteurs proposent deux algorithmes, aux performances très différentes.

Le premier est dédié aux termes, et plus précisément au problème de la recherche de motifs dont au moins une des feuilles est étiquetée par la variable v (le cas où M ne comporte aucun sommet étiqueté v peut être résolu en temps linéaire en utilisant le codage d'un terme par un mot [115]). Le principe de cet algorithme consiste à considérer l'ensemble $F(M)$ des sous-arborescences de M (pour chaque arborescence A de $F(M)$ il existe au moins un sommet y de M tel que $M_y = A$) et à associer à chaque sommet x de T le sous-ensemble des arborescences de $F(M)$ ayant une occurrence en x. On note $MS(x)$ cet ensemble d'arborescences. Pour une feuille x d'étiquette a, l'ensemble $MS(x)$ comporte les deux arborescences réduites à un seul sommet, étiqueté v (la variable) ou a. Partant de cette constatation, l'algorithme détermine les ensembles $MS(x)$ récursivement (en remontant des feuilles vers la racine) : si x est un sommet de degré k (ayant k fils notés $x_1, \ldots ,x_k$, dont on suppose connaître les ensembles $MS(x_i)$), $MS(x)$ comprend l'arborescence réduite au sommet étiqueté v et toutes les arborescences de $F(M)$ de la forme



telles que A_i appartient à $MS(x_i)$. En repérant les ensembles $MS(x)$ pour lesquels M appartient à $MS(x)$, on réduit ainsi la phase de recherche à un parcours de T, qui s'effectue en temps $O(n)$. Le point crucial consiste donc à calculer, pour tout noeud x, $MS(x)$ à partir de l'étiquette a de x et des ensembles $MS(x_i)$ de ses fils x_i. Pour cela, l'algorithme d'Hoffmann et O'Donnell nécessite la calcul, pour chaque symbole a de Σ, d'une table de dimension k si a est d'arité k, indexée par des sous-ensembles de $F(M)$. La principale tâche réside dans le calcul de ces tables, qui peuvent cependant occuper un espace en $O(2^m)$. Hoffmann et O'Donnell introduisent une sous-classe de l'ensemble des motifs possibles, appelée classe des motifs simples, pour lesquels le calcul de ces tables peut être effectué en temps $O(m^2 \times d + m^{d+1} \times |\Sigma|)$ et en espace $O(m^2 + |\Sigma| \times m^d)$, où d est l'arité maximale des symboles de Σ. Hoffmann et O'Donnell affirment de plus que dans les applications pour lesquelles ils ont utilisé cet algorithme (notamment l'implantation d'un interpréteur LISP), le fait de se restreindre à cette classe des motifs simples n'est pas contraignant. Cet algorithme a ensuite suscité plusieurs articles donnant une amélioration de la complexité en espace du prétraitement, travaux dûs notamment à Chase [33], Cai, Paige et Tarjan

[28] ou Thorup [155]. Les techniques mises en œuvre sont cependant toujours complexes, et il est clair qu'elles sont étroitement liées à la notion d'arité d'un sommet fixée par son étiquette et ne peuvent donc être appliquées qu'aux termes.

L'algorithme descendant de Hoffmann et O'Donnell. Le second algorithme proposé dans [91] aborde le problème de la recherche des occurrences compactes d'un motif dans un texte. Il suit un principe différent du précédent et privilégie un prétraitement plus léger au détriment, bien sûr, de la phase de recherche. Dans un premier temps, chaque sommet x de M autre que la racine est dédoublé de la manière suivante : si x est le $i^{ème}$ fils de son père, on remplace x par deux sommets étiquetés i et $e(x)$, et reliés par une arête (voir figure 7.8). On construit ensuite, à partir de cette arborescence, l'ADR des mots correspondant aux chemins allant de la racine vers les feuilles. Dans un second temps, à partir de cette ADR (dédoubler les sommets assure notamment que deux de ces mots ne peuvent être égaux), on construit l'automate de Aho-Corasick de cet ensemble de mots en le complétant, avec toutefois la restriction que pour chaque état u, si la dernière lettre de $m(u)$ est un entier (resp. une lettre de Σ), on ne complète les transitions issues de u que par les lettres de Σ (resp. les entiers) nécessaires. Le fait d'avoir dédoublé les sommets assure que l'automate ainsi obtenu est déterministe. Finalement, dans une dernière phase de prétraitement, on associe à tout état final u la longueur du mot $m(u)$ reconnu en ce sommet et une transition non étiquetée vers l'état final v reconnaissant le plus long suffixe propre de $m(u)$ (vers l'état initial si aucun suffixe propre de $m(u)$ n'est reconnu par l'automate) et on note $s(u)$ cette transition. Dans la figure suivante, on a représenté les états finaux par des carrés (l'entier situé à l'intérieur indiquant la longueur du mot reconnu), les transitions ajoutées à l'ADR associée à M par des pointillés longs et les transitions $s(u)$ par des pointillés courts. On note $AC(M)$ l'automate ainsi obtenu.

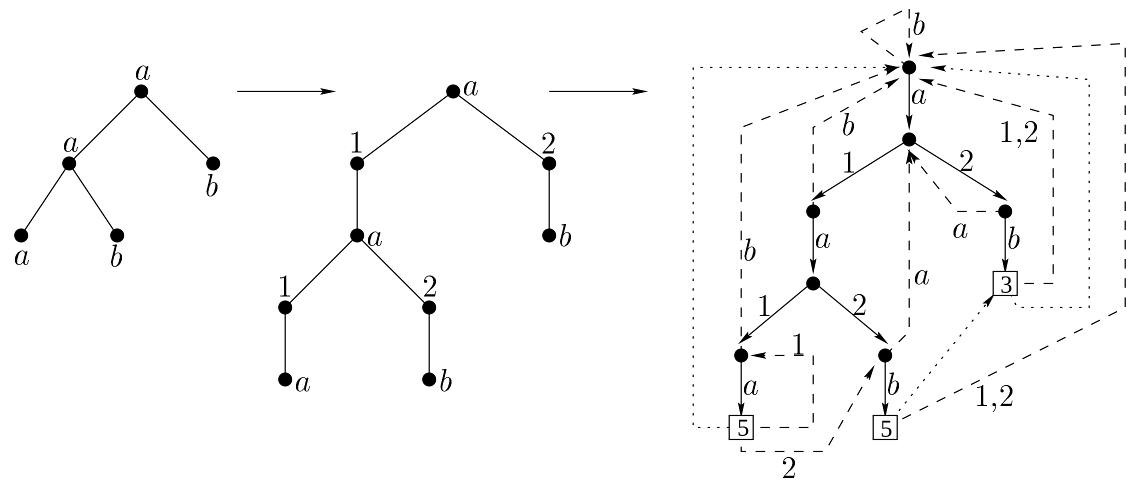


FIG. 7.8 – Une arborescence et l'automate correspondant

La construction de l'ADR se fait en temps $O(m)$, puisqu'elle repose uniquement sur une modification de M. Cette ADR comportant $2m$ sommets on déduit de la remarque 177 que le calcul des arcs complétant cet automate se fait en temps $O(m \times |\Sigma|)$. Finalement le calcul des liens $s(u)$ s'effectue en temps $O(m)$ et le prétraitement nécessite donc un espace $O(m \times |\Sigma|)$ et un temps $O(m \times |\Sigma|)$.

La phase de recherche de motif consiste, après avoir dédoublé les sommets du texte T de la même manière que pour M, à "lire" l'arborescence T dans l'automate $AC(M)$ (tout comme on lisait un texte T dans l'automate $AC(M)$). On associe cependant à tout sommet de T un compteur (initialisé à 0) enregistrant le nombre de chemins issus de ce sommet et correspondant à un chemin racine-feuille du motif. On a ainsi une occurrence de M en x si et seulement si à la fin de la phase de recherche de motifs, le compteur associé à x est égal au nombre de feuilles de M. On lit T dans $AC(M)$ en effectuant un parcours en profondeur de T (dont on a dédoublé les sommets) et en suivant, à chaque sommet x visité, la transition correspondant à son étiquette. On maintient de plus une pile contenant les couples (y,u) où y est un ancêtre de x et u l'état de l'automate atteint après avoir visité y (on utilise un tableau pour stocker cette pile, notée P). La principale opération concerne les états finaux de $AC(M)$: si en lisant le sommet x de T, on arrive sur un état final u de $AC(M)$, pour tous les sommets finaux v de $AC(M)$ reconnaissant un suffixe (non forcément propre) de $m(u)$ (on accède à ces sommets par les transitions s), si $m(v)$ est de longueur k, on incrémente le compteur associé au $k^{ème}$ ancêtre de x (on accède à ce sommet en temps $O(1)$ à l'aide de la pile P). On appelle cet algorithme **HOD**. Hoffmann et O'Donnell ont montré que l'algorithme **HOD** détecte les occurrences compactes de M dans T en temps $O(n \times l(M))$, où $l(M)$ est la taille du plus grand ensemble de feuilles $\{f_1,\dots,f_k\}$ de M telles que le mot porté par le chemin menant de la racine à f_i $i < n$) est un suffixe du mot associé à la feuille f_{i+1}. En remarquant qu'il existe des motifs à m sommets tels que $l(M)$ est en $O(m)$, on constate que cet algorithme n'améliore en fait pas la borne quadratique en $O(n \times m)$ dans le pire des cas de l'algorithme naïf.

Le fait que **HOD** soit limité aux occurrences compactes de M provient de la façon dont on dédouble les sommets de M lors du prétraitement, qui fait intervenir l'ordre des fils d'un sommet de manière trop stricte. Dans la section suivante, nous proposons une adaptation de cet algorithme au cas des occurrences quelconques du motif.

Les algorithmes de Kosaraju et Dubiner-Galil-Magen. La première amélioration de la borne quadratique pour la recherche de motifs dans une arborescence est due à Kosaraju [105], qui propose un algorithme en temps $O(n \times m^{3/4} \times \log(m))$ pour le problème RMCA. En reprenant le même principe que l'algorithme de Kosaraju, Dubiner, Galil et Magen [54] (voir aussi la présentation de cet algorithme par Crochemore et Rytter [52, section 15.2]) ont amélioré cette borne et obtenu un algorithme en temps $O(n \times \sqrt{m} \times \log(m))$.

Ces deux algorithmes nécessitent que l'étiquette de chaque sommet (du texte et du motif) contienne l'information "ordre parmi ses frères" : si le sommet x, étiqueté a, est le $i^{ème}$ fils de son père, on transforme l'étiquette de x en un couple (a,i). Si M est un motif à k feuilles $f_1,\dots,f_k$, on note $m(f_i)$ le mot correspondant au chemin allant de la racine de M à f_i. On a ainsi une occurrence de M en un sommet x de T, si et seulement si il existe k descendants $x_1,\dots,x_k$ de x tels que le mot allant de x à x_i est $m(f_i)$.

Pour obtenir une borne en $O(n \times q \times \log(m))$ ($q = m^{3/4}$ chez Kosaraju [105] et $q = \sqrt{m}$ chez Dubiner et al [54]), ces auteurs proposent des algorithmes opérant en trois étapes. La première consiste à calculer un ensemble d'arborescences $M_1,\dots,M_l$ obtenues en restreignant M aux sommets présents sur les chemins allant de la racine à un sous-ensemble des feuilles $f_1,\dots,f_k$ de façon à ce que les M_i vérifient des propriétés permettant de détecter les occurrences des M_i dans T en temps $O(n \times q)$ en utilisant un algorithme simple (l'algorithme naïf si $h(M_i)$ est faible, ou l'algorithme **HOD** si $l(M_i)$ est faible, où $l(M_i)$ est le paramètre défini dans la description de l'algorithme **HOD**). Si on note M' l'arborescence obtenue à partir de M en supprimant les sommets dont tous les descendants appartiennent aux M'_i, la deuxième étape consiste à rechercher les occurrences de M' dans T. Il s'agit là de l'étape difficile de ces deux algorithmes, nécessitant de faire appel à un algorithme de recherche de motifs dans un mot avec symboles "don't care" et, dans le cas de l'algorithme de Dubiner et al, à des propriétés de périodicité des chemins racine-feuilles de M' (intuitivement, il est naturel de jouer sur cette information car les chemins de faible longueur ont été traités avec les M'_i). Finalement,comme l'information "ordre parmi ses frères" a été ajoutée aux étiquettes des sommets, on peut dire qu'on a une occurrence de M en un sommet x de T si et seulement si on a une occurrence de chacune des arborescences M_i et M' en x.

On peut noter que pour calculer l'ensemble des arborescences M_i, Kosaraju introduit une notion d'arborescence des suffixes d'une arborescence dans laquelle un suffixe d'une arborescence M est le mot formé des étiquettes du chemin reliant la racine de M à l'une de ses feuilles. Kosaraju calcule cette arborescence des suffixes en temps $O(m \times \log(m))$, en utilisant la version séquentielle de l'algorithme parallèle mis au point dans [8] (la construction de cette arborescence des suffixes a ensuite été améliorée par Breslauer [27] et Shibuya [141]).

Comme dans le cas de l'algorithme **HOD**, il ne semble pas possible de pouvoir étendre ces algorithmes au problème RMA du fait de la présence nécessaire de l'information "ordre" dans l'étiquette des sommets de M et T.

Les algorithmes de Cole, Hariharan et Indyk. Dans [46], Cole et Hariharan empruntent une nouvelle voie, en proposant un algorithme probabiliste permettant de résoudre le problème RMCA en temps $O(n \times \log^3(m))$. Dans un premier temps, Cole et Hariharan introduisent le problème de la recherche de motifs dans les *arborescences binaires vertébrées*, c'est-à-dire les arborescences binaires dans lesquelles un chemin reliant la racine à un sommet ayant au plus un fils est distingué, ce chemin étant appelé la colonne vertébrale de l'arborescence. Ainsi, pour ces arborescences binaires vertebrées, l'existence d'une occurrence de M en un sommet x de T, nécessite que la colonne vertébrale de M corresponde à une partie de la colonne vertébrale de T (x doit donc appartenir à cette colonne). Les auteurs montrent ensuite que l'on peut réduire le problème RMCA à un ensemble d'instances du problème de la recherche de motifs dans des arborescences binaires vertébrées de tailles cumulées $O(n + m)$, cette réduction s'effectuant en temps $O(n + m)$. Cole et Hariharan introduisent ensuite une généralisation du problème de la recherche de motifs dans des mots, appelé problème du "subset matching" que l'on peut définir comme suit. Soit $P(\Sigma)$ l'alphabet constitué de tous les sous-ensembles de Σ, alphabet sur lequel on définit la notion de mot généralisé. Par exemple, si $\Sigma = \{a,b,c,d\}$, le mot $M = (a,c)()(a,c)()(b)$ est un mot généralisé sur Σ. Ainsi, si M et T sont deux mots généralisés sur Σ, tels que M est constitué de m sous-ensembles de Σ (M est de longueur m), on a une occurrence de M en $T[i]$ si et seulement si, pour $1 \leq j \leq m$, $M[j] \subseteq T[i+j-1]$. Par exemple, si $T = (c,d)(b,d)(a,b,c)()(a,c)()(b,c)()(a,b,c)(a,d)(a,d)(b)$, on a une seule occurrence de M en T, située en position 3. Cole et Hariharan montrent alors que la recherche de motifs dans des arborescences binaires vertébrées se réduit en temps linéaire au problème précédent. Finalement, ils proposent deux algorithmes pour résoudre le problème "subset matching" : un algorithme déterministe en temps $O(n \times \sqrt{m} \times \log(m))$ et un algorithme non déterministe en temps $O(n \times \log^3(m))$. En utilisant des résultats algorithmiques sur le codes superimposés [94], Cole, Hariharan et Indyk [47] ont ensuite proposé un algorithme déterministe permettant de résoudre le problème "subset matching", et donc le problème de la recherche de motifs dans une arborescence, en temps $O(n \times \log^3(n))$.

On peut noter que la notion de colonne vertébrale introduite dans cette approche impose que chaque sommet de la colonne vertébrale d'une arborescence a au plus un fils n'appartenant pas à cette colonne vertébrale, cette condition étant nécessaire lors la phase de recherche de motifs. Cet algorithme semble donc limité aux arborescences binaires et au problème RMCA.

Coder une arborescence par des mots. Dans une série de trois articles [112, 113, 114] (voir aussi les travaux de Grossi, Luccio et Pagli [85]), Luccio et Pagli introduisent une notion de distance entre deux arborescences A_1 et A_2, correspondant au nombre d'arborescences à insérer ou supprimer dans A_1 pour obtenir A_2. Ils proposent un algorithme simple, qui, étant donnés deux arborescences T et M et un entier k, détecte tous les sommets x de T pour lesquels T_x est à distance au plus k de M [2]. Cet algorithme se compose de trois étapes.

La première étape consiste à coder T (resp. M) par trois mots [3] S_T, F_T et P_T (resp. S_M, F_M et P_M), chacun de taille $O(n)$ (resp. $O(m)$), puis à concaténer les deux mots S_T et S_M en un seul mot S, et à calculer l'arborescence des suffixes $AS(S)$ de ce mot.

La deuxième étape consiste à prétraiter $AS(S)$ de sorte qu'étant donnés deux sommets u et v de $AS(S)$, on puisse calculer le plus proche ancêtre commun de u et v en temps $O(1)$ (et donc le plus long préfixe commun de $m(u)$ et $m(v)$). Harel et Tarjan [90] ont montré que ce prétraitement peut être effectué en temps et espace linéaire, soit en $O(n+m)$ dans notre cas.

Remarque 184. Ce résultat, désormais classique, est souvent utilisé pour la mise au point d'algorithmes sur les mots utilisant l'arborescence des suffixes (voir par exemple [86, partie II]). Il est dû à Harel et Tarjan [90], et a été ensuite amélioré par Schieber et Vishkin [137] et Berkman et Vishkin [17]. Récemment, Bender et Farach-Colton [13] ont proposé une nouvelle technique dont la mise en œuvre est plus aisée.

Finalement, la phase de recherche de motifs proprement dite consiste à lire n fois en parallèle un suffixe de S_T et S_M en utilisant $AS(S)$, F_T, P_T, F_M et P_M pour éviter de parcourir des facteurs identiques, ce qui permet d'effectuer chacun de ces parcours en temps $O(k)$. Cette phase a une complexité en temps en $O(k \times n)$.

On peut noter que l'idée du codage d'une arborescence par des mots a été utilisée notamment par Ramesh et Ramakrishnan [130] pour le problème de la recherche de motifs dans des termes, ou par Mäkinen [115] (voir aussi les articles de Dublish [55], Grossi [84, 83] et Verma [161]).

Quelques commentaires. Pour conclure ce rapide tour d'horizon des travaux abordant le problème de la recherche de motifs dans une arborescence, on peut souligner trois points essentiels.

Premièrement, il n'existe pas, à notre connaissance, d'algorithme traitant le problème RMA dans le cas des arborescences générales. Tous les algorithmes proposés portent soit sur les termes, soit sur le problème RMCA. Dans la section suivante, nous proposons une adaptation de l'algorithme descendant de Hoffmann et O'Donnell pour le problème RMA.

2. Cette notion de distance est une simplification de la notion de distance entre deux arborescences introduite par Tai [152] et Zhang et Shasha [167].

3. Nous décrivons ce codage dans le chapitre suivant, dans lequel nous présentons un algorithme utilisant cette approche.

Deuxièmement, on remarque que même le problème RMCA a reçu peu d'attention (comme nous venons de le voir, les travaux sur ce sujet sont peu nombreux), ce qui s'explique sans doute par sa difficulté. Ainsi les techniques employées pour obtenir un algorithme en temps quasi-linéaire (le seul algorithme ayant une complexité de cet ordre est celui de Cole, Hariharan et Indyk) sont complexes et lourdes à implanter. C'est pourquoi, bien qu'aucune étude expérimentale sur ce sujet n'ait été réalisée, on peut penser que ces algorithmes sont peu compétitifs, relativement au temps d'exécution moyen, en comparaison avec l'algorithme naïf qui se caractérise par sa simplicité et son bon comportement en moyenne (les résultats expérimentaux que nous fournissons dans la section suivante semblent indiquer un comportement linéaire en moyenne). C'est ainsi que dans la plupart des applications nécessitant d'effectuer une recherche de motifs dans une arborescence, l'algorithme implanté est d'ailleurs l'algorithme naïf (à plus forte raison lorsque le problème traité est le problème RMA et non le problème RMCA).

Finalement, on peut remarquer qu'aucun des algorithmes proposés dans la littérature ne semble adapté au cas statique. En effet, comme nous l'avons vu dans la deuxième section de ce chapitre, on peut, dans le cas des mots, prétraiter "une fois pour toute" le texte (en construisant son arborescence des suffixes, ou son tableau des suffixes qui possède de meilleures propriétés pratiques), en temps et espace linéaires, et effectuer ainsi une recherche du motif en temps quasi-linéaire. Cette technique est particulièrement utile pour rechercher rapidement un ou des motifs dans une banque de données de textes statiques (comme des séquences biologiques par exemple). Dans le chapitre suivant, nous proposons une adaptation de l'algorithme naïf basée sur le codage d'une arborescence par des mots et l'utilisation de l'arborescence des suffixes, qui si elle ne modifie pas la complexité quadratique de l'algorithme naïf, semble posséder de bonnes propriétés pratiques dans le cas des termes et du problème RMCA.

7.4 Adaptation de l'algorithme HOD

Dans cette section, nous décrivons une modification de l'algorithme **HOD** (qui traite le problème RMCA) afin de résoudre le problème RMA. Cette modification, bien que très simple, ne semble figurer dans aucun article antérieur sur le sujet. Nous commençons par rappeler la définition du *parcours en profondeur préfixe* d'une arborescence.

Définition 185. Le parcours en profondeur préfixe d'une arborescence A est le parcours consistant à visiter la racine de A, puis à parcourir récursivement chacun des sous-arborescences issues de cette racine dans l'ordre où ils apparaissent. Lors de ce parcours, on associe un indice à chaque sommet lors de ce parcours (indice i pour le $i^{ème}$ sommet visité).

On suppose ici que tout sommet x du texte T connaît la taille $|T_x|$ de la sous-arborescence dont il est racine et l'indice qui lui est associé lors d'un parcours en profondeur préfixe (on note cette information $ind(x)$). Ce prétraitement du texte s'effectue en temps et espace $O(n)$.

Dans un premier temps, on associe à toute feuille x du motif les informations suivantes (voir figure 7.9) :

- si x est la $i^{ème}$ feuille visitée lors d'un parcours en profondeur préfixe de M, on note $num(x) = i$ (on dit que x est la $i^{ème}$ feuille de M) ;
- on note $l1(x)$ le nombre d'arêtes du chemin allant de x à la racine de M ;

- si $num(x) = i$ et $i > 1$ (x n'est pas la première feuille), on note $l2(x)$ le nombre d'arêtes du chemin allant de x au plus proche ancêtre commun de x et de la $(i-1)^{ème}$ feuille de M (on a donc $l2(x) > 0$) ; sinon, $l2(x) = 0$, pour la première feuille de x ($num(x) = 1$).

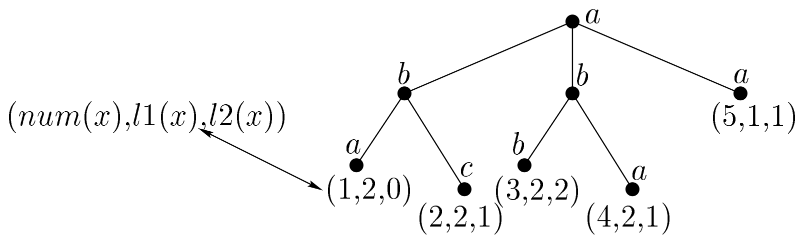


FIG. 7.9 – Informations associées aux feuilles du motif

Notation. Soient A une arborescence et x, y deux sommets de A tels que y est un descendant de x. On note $C_A(x,y)$ ($C_A(y)$ si x est la racine de A) le mot formé des étiquettes des sommets du chemin allant de x à y, les sommets x et y inclus (si x est le $k^{ème}$ ancêtre de y, ce mot comporte donc $k+1$ lettres).

On construit ensuite l'automate de Aho-Corasick de l'ensemble des mots $C_M(f_i)$ associés à chaque feuille f_i de M (f_i désignant la $i^{ème}$ feuille de M). On associe de plus à tout état final u de cet automate (représenté par un carré dans la figure suivante) l'ensemble des triplets $(i,l1(f_i),l2(f_i))$ tels que le mot correspondant au chemin allant de la racine de M à la feuille f_i est égal à $m(u)$. Finalement, on ajoute les transitions $s(u)$ de la même manière que pour l'automate AC calculé par l'algorithme **HOD**, et on obtient ainsi un automate que l'on note aussi $AC(M)$. Dans la figure 7.10, on n'a pas représentée les transitions $s(u)$, celles-ci reliant chaque état final à l'état initial (pour cet exemple).

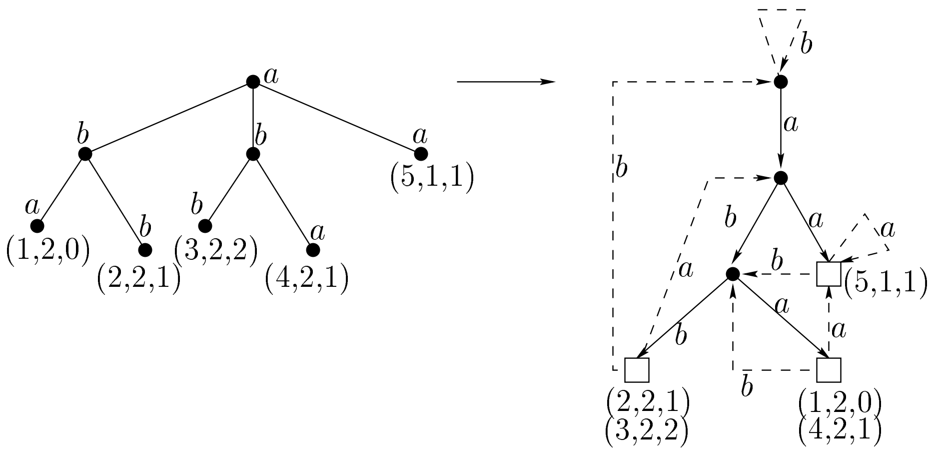


FIG. 7.10 – Un motif M et l'automate $AC(M)$ correspondant

Définition 186. Soit M une arborescence ayant k feuilles. On note M_i l'arborescence associée à la feuille f_i et obtenue en supprimant de M tous les sommets n'ayant, parmi leurs descendants, aucune feuille f_j telle que $j \leq i$. On appelle M_i le $i^{ème}$ préfixe de M car cette arborescence correspond à la suppression dans M de tous les sommets ayant un indice supérieur à celui de la feuille f_i.

On peut remarquer que, pour un sommet x de T, on détecte une occurrence de M_1 en x si et seulement si il existe un descendant y de x tel que $C_T(x,y) = C_M(f_1)$. Supposons maintenant que l'on ait trouvé une occurrence de M_i en un sommet x de T (on note z le sommet de T correspondant à f_i), et qu'il existe un descendant y de x tel que $C_T(x,y) = C_M(f_{i+1})$. On a alors une occurrence de M_{i+1} en x telle que y corresponde à la feuille f_{i+1} si et seulement si les conditions suivantes sont satisfaites (voir figure 7.11) : si $k = l2(f_{i+1})$, si y_1 est le $k^{ème}$ ancêtre de y et y_2 le $(k-1)^{ème}$ ancêtre de y (y_1 est le père de y_2) alors z appartient à la sous-arborescence T_{y_1} et le chemin allant de y_1 à z passe par un frère gauche de y_2, ce qui se traduit formellement par

$$ind(y_1) < ind(z) < ind(y_1) + |T_{y_1}| - 1 \text{ et } ind(y_2) > ind(z). \tag{7.1}$$

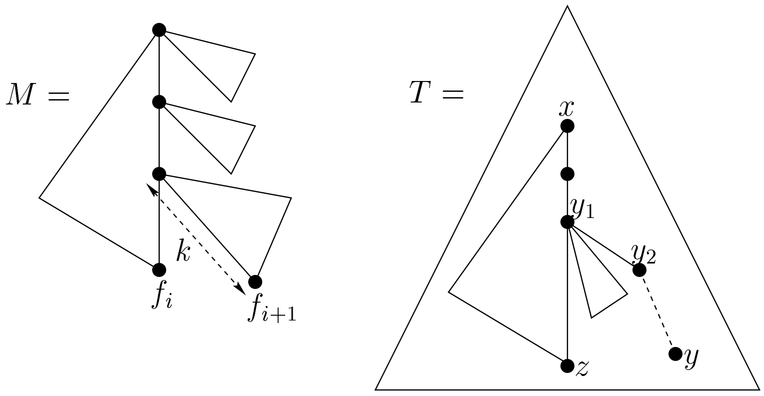


FIG. 7.11 – $C_T(x,y) = C_M(f_{i+1})$

Le principe de la phase de recherche de motifs consiste à parcourir en parallèle l'arborescence T (par un parcours en profondeur préfixe) et l'automate $AC(M)$ de sorte qu'après avoir visité un sommet s de T, on se trouve dans un état t de l'automate $AC(M)$ correspondant à la lecture du mot $C_T(s)$. Durant ce parcours , on maintient une pile P (gérée à l'aide d'un tableau) telle qu'à tout instant, le sommet de pile contienne le quadruplet (s,j,k,t) tel que

- s est le sommet courant de T (le dernier sommet visité lors du parcours en profondeur préfixe de T),
- j est l'indice du plus grand préfixe de M (définition 186) reconnu en s,
- k est l'indice du sommet z de T_s correspondant à la $j^{ème}$ feuille de M ($ind(z) = k$),
- t est l'état de $AC(M)$ atteint après avoir visité le sommet s,

et on maintient dans ce tableau (cette pile) les informations correspondant aux ancêtres du dernier sommet s visité (le quadruplet correspondant à s est le dernier élément du tableau alors que la quadruplet correspondant à la racine de T en est le premier élément). On en déduit que P contient à tout instant au plus $h(T)$ quadruplets (on rappelle que $h(T)$ est la hauteur de T). Finalement, pour détecter les occurrences de M dans T, on maintient les informations données par les quadruplets de P de la manière suivante : si, après avoir visité le sommet s on arrive sur un état final t de $AC(M)$, pour chaque triplet associé à un état final de l'automate reconnaissant un suffixe de longueur k de $m(t)$ (on parcourt ces états finaux grâce aux transitions $s(t)$ de l'automate), on modifie le quadruplet de P associé au $k^{ème}$ ancêtre de s lorsque la condition (7.1) est vérifiée. On obtient ainsi l'algorithme suivant, que nous appelons **HOD2**.

Algorithme 13: HOD2 : recherche du motif M dans T avec l'automate $AC(M)$.

Données : $AC(M)$ et T
début
    soient $init$ l'état initial de $AC(M)$ et r la racine de T;
    $top := 1$, $t := \delta(init, e(x))$ et $P[top] := (r, 0, 0, t)$;
    tant que $top > 0$ **faire**
        $(s, j, k, t) := P[top]$;
        si tous les fils de s ont été visités **alors**
            $top := top - 1$;
            si j est égal au nombre de feuilles de M **alors** afficher "Occurrence de M en s";
        sinon
            soit y le premier fils de s non visité;
            $top := top + 1$, $t := \delta(t, e(y))$ et $P[top] := (y, 0, 0, t)$;
            si t est un état final **alors**
                tant que $t \neq init$ **faire**
                    pour chaque triplet (i, l_1, k) associé à l'état t **faire**
1 $(x, j_1, k_1, t_1) := P[top - l_1]$;
                        $(y_1, j_2, k_2, t_2) := P[top - k]$;
                        $(y_2, j_3, k_3, t_3) := P[top - k + 1]$;
                        si $(j_1 = i - 1)$ et $((k_1 = 0)$ ou $(ind(y_1) < k_1 < ind(y_1) + |T_{y_1}| - 1$ et $ind(y_2) > k_1))$ **alors**
2 $P[top - l_1] := (x, j_1 + 1, ind(y), t_1)$;
                    $t := s(t)$;
fin

La complexité du prétraitement est identique à celle du prétraitement de l'algorithme descendant de Hoffmann et O'Donnell : la transformation de M en ADR est réalisée en temps $O(m)$, la complétion de cette ADR pour obtenir un automate complet s'effectue en temps $O(m \times |\Sigma|)$ et le calcul des transitions $s(t)$ se fait en temps $O(m)$. Lors de la phase de recherche du motif, le nombre d'opérations effectuées lorsque l'on visite un sommet x de T est proportionnel au nombre d'exécutions du bloc situé entre les lignes marquées 1 et 2, qui vaut au plus $l(M)$. On obtient alors le résultat suivant.

Proposition 187. *Soient T et M deux arborescences de tailles respectives n et m. L'algorithme* **HOD2** *détecte toutes les occurrences de M dans T en temps $O(n \times l(M))$.*

Nous donnons maintenant quelques résultats expérimentaux qui laissent penser que cet algorithme est malheureusement moins efficace que l'algorithme naïf. Nous avons mis en œuvre ces deux algorithmes en C et nous avons procédé de la manière suivante pour réaliser des tests. Nous avons engendré aléatoirement et uniformément une arborescence T sur Σ [2], puis une centaine d'arborescences M sur Σ de taille comprise entre 5 sommets et 20 sommets, et enfin, pour chacune d'entre elles, nous avons recherché les occurrences de M dans T en utilisant l'algorithme naïf, puis l'algorithme **HOD2**. Les statistiques suivantes ont été mesurées : nombre $Comp1$ de comparaisons entre les étiquettes d'un sommet de T et d'un sommet de M effectuées par l'algorithme naïf, nombre $Comp2$ d'exécutions du bloc situé entre les lignes marquées 1 et 2 de **HOD2**, temps t_1 pris par l'algorithme naïf et temps t_2 pris par l'algorithme **HOD2**. Nous obtenons les résultats suivants, synthétisés par le tableau suivant. On peut de plus ajouter qu'en nous limitant à des motifs ayant peu de feuilles, nous avons obtenu des résultats semblables.

$\|T\|$	$\|\Sigma\|$	$Comp2/Comp1$	t_2/t_1
100	1	2,73	2,68
100	3	2,52	3,04
100	5	3,60	3,71
100	7	4,79	3,84
1000	1	2,15	2,93
1000	3	2,05	3,85
1000	5	2.97	4,57
1000	7	3,99	4,82
10000	1	1.97	4,03
10000	3	1,92	4,79
10000	5	2,80	4,88
10000	7	3,76	5,52

TAB. 7.1 – Comparaison de l'algorithme naïf et de l'algorithme **HOD2**

Chapitre 8

Coder une arborescence par des mots

Comme nous l'avons mentionné dans le chapitre précédent, en conclusion de la revue des principaux travaux traitant de problème de la recherche de motifs dans une arborescence, aucun des algorithmes proposés n'est dédié au cas statique (le texte T est fixé). Dans ce chapitre, nous nous intéressons à ce problème et nous proposons une adaptation de l'algorithme naïf basée sur le codage d'une arborescence par des mots et l'utilisation de l'arborescence des suffixes. Si la complexité en temps (toujours dans le pire des cas) de l'algorithme que nous proposons en $O(n \times m)$, les résultats expérimentaux que nous obtenons, dans le cas de termes binaires ou du problème RMCA, semblent indiquer des performances en moyenne intéressantes (d'un point de vue pratique et théorique).

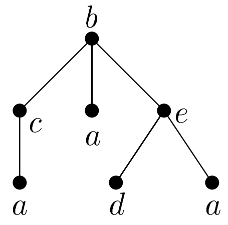
Dans la première section nous nous intéressons au cas des termes. Nous poursuivons en décrivant un algorithme pour le problème RMCA dans le cas des arborescences générales basé sur le même principe mais utilisant un codage d'une arborescence par des mots légèrement différent. Nous concluons en exposant quelques résultats expérimentaux.

Remarque 188. On peut noter que le codage d'une arborescence par des mots que nous utilisons dans les deux prochaines sections ci-dessous provient des travaux de Luccio et Pagli [112, 113, 114]. Ces auteurs l'utilisent pour résoudre certains problèmes de recherche de motifs dans une arborescence légèrement différents des problèmes que nous considérons dans ce mémoire.

8.1 Le cas des termes

Définition 189. Soit A un terme de taille n sur l'alphabet de symboles Σ. On note S_A et L_A les mots de longueur n sur les alphabets respectifs Σ et $[n]$ définis de la manière suivante : si x est le $i^{ème}$ sommet visité lors du parcours en profondeur préfixe de A, alors $S_A[i] = e(x)$ et $L_A[i] = i + |T_x|$.

On remarque que si x est le $i^{ème}$ sommet visité lors du parcours en profondeur de A, $L_A[i]$ donne la position dans S_A du symbole associé au sommet rencontré immédiatement après que toute la sous-arborescence de racine x ait été visitée.



$S_A = b.c.a.a.e.d.a$
$L_A = 8.4.4.5.8.7.8$

FIG. 8.1 – Codage d'un terme par 2 mots

Observation 190. Soit x un sommet de T et i l'indice du symbole $e(x)$ de S_T associé à x. On a une occurrence de M en x si et seulement si il existe m entiers $i_1 = i < i_2 < \cdots < i_m \leq n$ tels que, pour $1 \leq j \leq n$,
- si $S_M[j] \neq v$ alors $S_T[i_j] = S_M[j]$,
- si $S_M[j] \neq v$ alors $i_{j+1} = i_j + 1$, sinon $i_{j+1} = L_T[i_j]$.

On en déduit la version suivante de l'algorithme naïf. Comme nous nous plaçons dans le cas statique, nous supposons que le texte T a été prétraité de sorte que pour chaque symbole de Σ, on connaît ses occurrences dans S_T. On rappelle qu'on note n la taille du texte T et m la taille du motif M.

Algorithme 14: Algorithme naïf amélioré (pour les termes)

Données : S_T, L_T, S_M
début
  pour chaque $1 \leq i \leq n$ tel que $S_T[i] = S_M[1]$ **faire**
    $j := 2$ et $k := i + 1$;
1   **tant que** $j \leq m$ **faire**
      si $S_T[k] = S_M[j]$ **alors**
        $j := j + 1$ et $k := k + 1$;
      sinon si $S_M[j] = v$ **alors**
        $j := j + 1$ et $k := L_T[k]$;
      sinon sortir de la boucle **tant que**;
    si $j = m + 1$ **alors**
      afficher "Occurrence de M en i";
fin

On peut cependant remarquer que si il existe i_1 et i_2 tels que $S_T[i_1,n]$ et $S_T[i_2,n]$ ont un préfixe commun égal à un préfixe de S_M, ces deux facteurs identiques de S_T seront lus par cet algorithme. Le principe de l'amélioration que nous proposons consiste à éviter cette lecture multiple de certains facteurs identiques en utilisant l'arborescence des suffixes de S_T. Notre algorithme repose sur la définition de la reconnaissance d'un préfixe de S_M dans $AS(S_T)$, que nous énonçons ci-dessous avant de l'illustrer par la figure 8.2.

Définition 191. Soient (d,f) l'étiquette d'une arête de $AS(S_T)$ allant d'un sommet u vers un sommet w (à cette arête correspond le facteur $S_T[d,f]$) et l un entier tel que

$0 \leq l \leq f - d$. On dit que $((d,f),l)$ reconnaît le $k^{ème}$ préfixe de S_M si il existe k indices $(d - |m(u)|) = i_1 < i_2 < \cdots < i_k = (d + l)$ tels que, pour $1 \leq j \leq k$,
- si $S_M[j] \neq v$ alors $S_T[i_j] = S_M[j]$,
- si $S_M[j] \neq v$ alors $i_{j+1} = i_j + 1$, sinon $i_{j+1} = L_T[i_j]$.

Dans la figure suivante, on a représenté de manière abstraite les symboles de Σ par de petits carrés blancs.

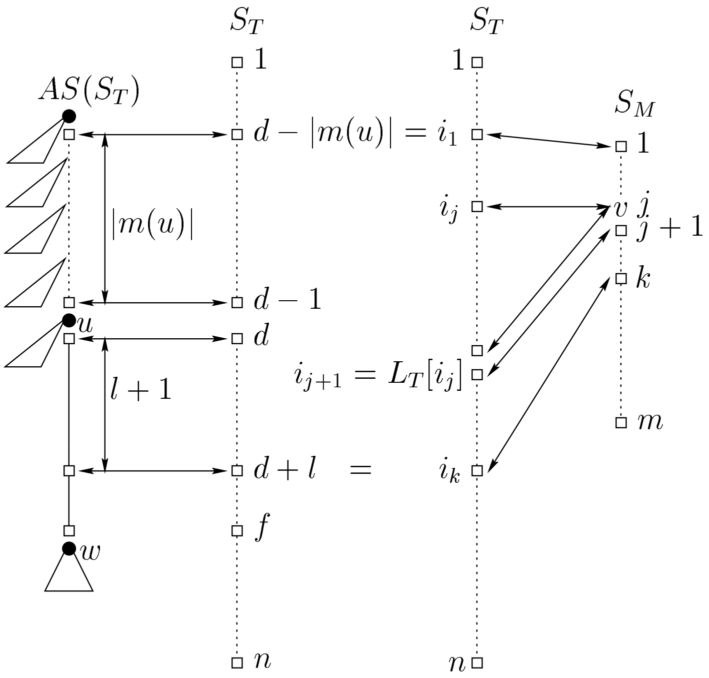


FIG. 8.2 – Illustration de la définition 191

Le principe de notre algorithme est de lire S_M dans $AS(S_T)$, la seule difficulté étant de gérer efficacement le cas où $S_M[j] = v$ (on dit alors qu'on effectue un *saut*). Pour cela on s'appuie sur l'observation suivante, que l'on illustre par la figure 8.3.

Observation 192. Soient $k < m$ tel que $S_M[k+1] = v$ et $((d,f),l)$ un couple reconnaissant le $k^{ème}$ préfixe de S_M (au sens de la définition 191). On suppose que (d,f) est l'étiquette d'une arête de $AS(S_T)$ allant d'un sommet u à un sommet w. Soient (d',f') l'étiquette d'une arête de $AS(S_T)$ allant d'un sommet u' descendant de u à un sommet w' descendant de w, h la longueur du mot étiquetant le chemin allant de $((d,f),l)$ à u' ($h = 0$ si $u' = u$ et $l = 0$, et $h = |m(u')| - |m(u)| - l$ sinon) et l' un entier tel que $0 \leq l' \leq f' - d'$. $((d',f'),l')$ reconnaît le $(k+1)^{ème}$ préfixe de S_M si et seulement si

$$L_T[d' - h + 1] = d' + l' + 1.$$

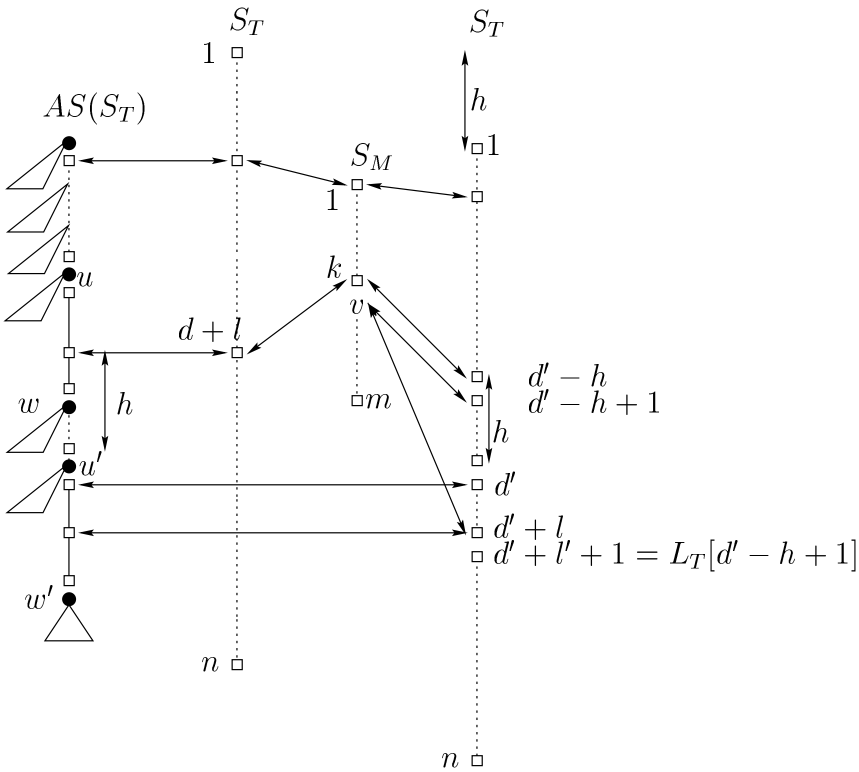


FIG. 8.3 – Illustration de l'observation 192

On définit alors deux procédures pour parcourir $AS(S_T)$. La première, appelée **Lecture**, est une modification de la procédure de recherche d'un mot dans une arborescence des suffixes, tant que le caractère courant de S_M est différent de v et fait appel à une procédure récursive **Sauts** dès que le caractère courant de S_M est la variable v (on compare $S_M[j] = v$ avec le $(l{+}1)^{ème}$ caractère de l'arc (d,f)). La procédure **Sauts** cherche alors tous les couples $((d',f'),l')$ reconnaissant le $(j+1)^{ème}$ préfixe de S_M (on dit alors qu'on a effectué un *saut* de $((d,f),l)$ vers $((d',f'),l')$). Dans les procédures décrites ci-dessous, on suppose que l'arête (d,f) (resp. (d',f')) va du sommet u (resp. u') vers le sommet w (resp. w').

Algorithme 15: Recherche de motifs accélérée (pour les termes)

Données : S_T, L_T, S_M et $AS(S_T)$
début
  soit r la racine de $AS(S_T)$;
  si une arête (d,f) telle que $S_T[d] = S_M[1]$ quitte r **alors**
    Lecture(1, $((d,f),0)$);
fin

Algorithme 16: Lecture (pour les termes)

Données : j, $((d,f),l)$
début
  $k := j$;
1 **tant que** $k \leq m$, $l \leq (f-d)$ et $S_T[d+l] = S_M[k]$ **faire**
    $k := k+1$ et $l := l+1$;
  si $k > m$ **alors**
2   parcourir la sous-arborescence de $AS(S_T)$ de racine w, tous les sommets finaux correspondant à une occurrence de M dans T;
  sinon si $l > (f-d)$ **alors**
    pour chaque arête (d',f') quittant w **faire**
      Lecture(k, $((d',f'),0)$);
  sinon si $S_M[k] = v$ **alors**
    Sauts(k, $((d,f),l)$, 0) **sinon** Échec;
fin

Algorithme 17: Sauts (pour les termes)

Données : j, $((d,f),l)$ et h (voir figure 8.3
début
  $g := L_T[d+l-h+1]$;
  si $g \leq f+1$ **alors**
    Lecture($j+1$, $((d,f),g-d)$);
  sinon pour toute arête (d',f') quittant w **faire**
    Sauts(j, $((d',f'),0)$, $h+(f-d+1)-l$);
fin

Proposition 193. *Soient T et M deux termes de tailles respectives n et m. L'algorithme 15 détecte toutes les occurrences de M dans T en temps $O(n \times m)$ dans le pire des cas.*

Preuve. La complexité en temps de cet algorithme dépend de trois paramètres : le nombre de comparaisons entre les caractères de S_T et S_M (nombre d'itérations de l'étape marquée 1 de **Lecture**), le nombre d'appels à **Sauts** et le parcours des sous-arborescences de $AS(S_T)$ en cas de succès (étape marquée 2 de **Lecture**). Le nombre de comparaisons de caractères, étant clairement au plus égal à celui de l'algorithme naïf, est en $O(n \times m)$. Le nombre de sommets de $AS(S_T)$ visités lors de l'étape marquée 2 de **Lecture** est en $O(k)$ si on a k occurrences de M dans T, et $k \leq n$. Finalement, en remarquant qu'il ne peut y avoir deux appels à **Sauts** concernant la même arête (d,f) que si on a effectué une comparaison entre un caractère de S_T et un caractère de S_M entre ces deux appels, on a bien $O(n \times m)$ appels à **Sauts**. □

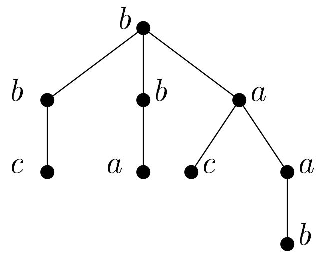
Nous donnons en fin de chapitre des résultats expérimentaux qui semblent indiquer que sur des termes aléatoires, cet algorithme est sensiblement plus rapide que l'algorithme naïf.

8.2 Le cas des arborescences quelconques

On peut bien entendu appliquer le même principe aux arborescences quelconques, mais le codage par des mots doit être modifié. On utilise en effet 3 mots de longueur $2n$ chacun pour coder une arborescence à n sommets.

Définition 194. Soit A une arborescence de taille n sur Σ. On définit les trois mots S_A, L_A et P_A de longueur $2n$ sur l'alphabet $\Sigma \cup \{0\}$ comme suit :

- le mot S_A est obtenu en codant, lors d'un parcours en profondeur préfixe de A, le premier (resp. dernier) passage sur le sommet x par $e(x)$ (resp. 0) : on dit que x est le sommet associé à $S_A[i] = e(x)$;
- si $S_A[i] = 0$ alors $L_A[i] = 0$; sinon, x étant le sommet associé à $S_A[i]$, $L_A[i] = i + 2|A_x|$;
- si $S_A[i] = 0$ ou $i = 1$, alors $P_A[i] = 0$; sinon, x étant le sommet associé à $S_A[i]$ et y le père de x (qu'on suppose associé à $S_A[j]$), on pose $P_A[i] = j$.



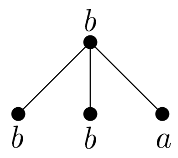
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$S_A =$	b	b	c	0	0	b	a	0	0	a	c	0	a	b	0	0	0	0
$L_A =$	19	6	5	0	0	10	9	0	0	18	13	0	17	16	0	0	0	0
$P_A =$	0	1	2	0	0	1	6	0	0	1	10	0	10	13	0	0	0	0

FIG. 8.4 – Codage d'une arborescence quelconque par des mots

Observation 195. Soient x un sommet de T et i l'indice du symbole $e(x)$ de S_T associé à x. On a une occurrence de M en x si et seulement si il existe $2m$ entiers $i_1 = i < i_2 < \cdots < i_{2m} \leq 2n$ tels que

- pour $1 \leq j \leq 2m$, $S_T[i_j] = S_M[j]$,
- si $j < 2m$ et $S_M[j+1] \neq 0$ alors $i_{j+1} = i_j + 1$
- si $j < 2m$ et $S_M[j+1] = 0$:
  - si $S_T[i_j + 1] = 0$ alors $i_{j+1} = i_j + 1$,
  - sinon, $i_{j+1} = L_T[P_T[i_j + 1]] - 1$.

Par exemple, si T est l'arborescence A de la figure précédente et M est l'arborescence suivante,



alors
$$S_M = b\ b\ 0\ b\ 0\ a\ 0\ 0,$$
et on a une occurrence de M à la racine de T, correspondant aux symboles de S_T entourés de crochets ci-dessous
$$[b]\ [b]\ c\ 0\ [0]\ [b]\ a\ 0\ [0]\ [a]\ c\ 0\ a\ b\ 0\ 0\ [0]\ [0].$$

On remarque que les lettres 0 du motif jouent un rôle analogue à la variable v pour les termes et on peut traduire l'algorithme naïf pour le problème RMCA de la manière suivante.

Algorithme 18: Algorithme naïf amélioré (problème RMCA et arborescences quelconques)

Données : S_T, L_T, P_T et S_M
début
  pour chaque $1 \leq i \leq 2n$ tel que $S_T[i] = S_M[1]$ **faire**
    $j := 2$ et $k := i + 1$;
    tant que $j \leq 2m$ **faire**
      si $S_T[k] = S_M[j]$ **alors**
        $j := j + 1$ et $k := k + 1$;
      sinon si $S_M[j] = 0$ **alors**
        $j := j + 1$ et $k := L_T[P_T[k]]$;
      sinon sortir de la boucle **tant que**;
    si $j = 2m + 1$ **alors**
      afficher "Occurrence en i";
fin

Nous pouvons procéder par analogie au cas des termes en considérant $AS(S_T)$ pour accélérer cet algorithme en évitant la comparaison de préfixes communs. On définit la notion de reconnaissance d'un préfixe de S_M dans $AS(S_T)$ de la manière suivante, conséquence de l'observation 195.

Définition 196. Soient (d,f) une arête de $AS(S_T)$ (allant d'un sommet u vers un sommet w) et l un entier tel que $0 \leq l \leq f - d$. On dit que $((d,f),l)$ reconnaît le $k^{ème}$ préfixe de S_M si il existe k entiers $(d - |m(u)|) = i_1 < i_2 < \cdots < i_k = (d + l)$ tels que :
- pour $1 \leq j \leq k$, $S_T[i_j] = S_M[j]$,
- si $j < k$ et $S_M[j + 1] \neq 0$ alors $i_{j+1} = i_j + 1$
- si $j < k$ et $S_M[j + 1] = 0$:
  - si $S_T[i_j + 1] = 0$ alors $i_{j+1} = i_j + 1$,
  - sinon, $i_{j+1} = L_T[P_T[i_j + 1]] - 1$.

Observation 197. Soient $k < m$ tel que $S_M[k + 1] = 0$ et $((d,f),l)$ un couple reconnaissant le $k^{ème}$ préfixe de S_M. On suppose que (d,f) est l'étiquette d'une arête de $AS(S_T)$ allant d'un sommet u à un sommet w. Soient (d',f') l'étiquette d'une arête de $AS(S_T)$ allant d'un sommet u' descendant de u à un sommet w' descendant de w, h la longueur du mot étiquetant le chemin allant de $((d,f),l)$ à u' ($h = 0$ si $u' = u$ et $l = 0$, et $h =$

$|m(u')| - |m(u)| - l$ sinon) et l' un entier tel que $0 \leq l' \leq f' - d'$. $((d',f'),l')$ reconnaît le $(k+1)^{ème}$ préfixe de S_M si et seulement si

$$d' = d,\ l' = l + 1 \text{ et } S_T[d' + l'] = 0$$
$$\text{ou}$$
$$u' = w,\ l' = 0 \text{ et } S_T[d'] = 0$$
$$\text{ou}$$
$$L_T[P_T[d' - h + 1]] = d' + l' + 1.$$

En s'appuyant sur cette observation, on obtient, de la même façon que pour les termes, les procédures suivantes. On suppose que l'arête (d,f) (resp. (d',f')) va du sommet u (resp. u') vers le sommet w (resp. w').

Algorithme 19: Recherche de motifs accélérée (problème RMCA et arborescences quelconques)

---

début
- Soit r la racine de $AS(S_T)$;
- **si** une arête (d,f) telle que $S_T[d] = S_M[1]$ quitte r **alors**
  - **Lecture**$(1, ((d,f),0))$;

fin

Algorithme 20: Lecture (problème RMCA et arborescences quelconques)

Données : j, $((d,f),l)$

début
- $k := j$;
- **tant que** $k \leq 2m$, $l \leq (f - d)$ et $S_T[d + l] = S_M[k]$ **faire**
  - $k := k + 1$ et $l := l + 1$;
- **si** $k > 2m$ **alors**
  - parcourir la sous-arborescence de $AS(S_T)$ de racine w, tous les sommets finaux correspondant à une occurrence de M dans T;
- **sinon**
  - **si** $l > (f - d)$ **alors**
    - **si** il existe une arête (d',f') quittant w telle que $S_T[d'] = S_M[k]$ **alors**
      - **Lecture**$(k + 1, ((d',f'),0))$;
    - **sinon si** $S_M[k] = 0$ **alors**
      - **pour** toute arête (d',f') quittant w **faire**
        - **Sauts**$(j, ((d',f'),0), d' - P_T[d'])$;
  - **sinon si** $S_M[j] = 0$ **alors**
    - **Sauts**$(j, ((d,f),l), d + l - P_T[d + l])$;

fin

Algorithme 21: Sauts(problème RMCA et arborescences quelconques)

Données : j, $((d,f),l)$ et h
début
  $g := L_T[P_T[d+l-h+1]]$;
  si $g \leq f$ **alors**
    Lecture($j+1$, $((d,f),g-d)$);
  sinon
    pour toute arête (d',f') quittant y **faire**
      Sauts(j, $((d',f'),0)$, $h+(f-d+1)-l$);
fin

Proposition 198. *Soient T et M deux arborescences de tailles respectives n et m. L'algorithme 19 détecte toutes les occurrences compactes de M dans T en temps $O(n \times m)$ dans le pire des cas.*

Preuve. La preuve de ce résultat est identique à celle de la proposition 193.

8.3 Résultats expérimentaux

Nous avons implanté les algorithmes 14 et 15 (en C sur un PC-Linux) et les avons expérimentés sur des termes binaires. Pour cela nous avons suivi un processus expérimental analogue à celui du chapitre précédent. Nous avons engendré aléatoirement et uniformément un terme binaire T sur Σ, puis une centaine de motifs binaires M sur Σ de taille comprise entre 5 sommets et 20 sommets, et enfin, pour chacun d'entre eux, recherché leurs occurrences dans T en utilisant l'algorithme 14, puis par l'algorithme 15. Nous avons alors réalisé les statistiques suivantes : nombre $Comp1$ de comparaisons entre les étiquettes de deux sommets de T et M effectuées par l'algorithme naïf, nombre $Comp2$ de comparaisons effectuées par l'algorithme 15, nombre $Sauts$ d'appels à la procédure **Sauts**, temps t_1 pris par l'algorithme 14 et temps t_2 pris par l'algorithme 15. Dans la table 8.1, $mult$ désigne le nombre d'étiquettes possibles pour un nœud (resp. feuille) de T et de M (on a donc $|\Sigma| = 2 \times mult$). Nous présentons ces statistiques en comparant $Comp1$ à $Comp2$ et $Sauts$, et t_1 à t_2.

On remarque alors que l'algorithme 14 effectue effectivement beaucoup moins de comparaisons entre caractères de S_T et S_M que l'algorithme 15 (on rappelle que Steyaert et Flajolet ont montré que $Comp1$ est en $O(n)$ [149]), et que le nombre d'appels à la procédure **Sauts** est en $O(n)$. Cela se traduit en pratique par le fait que dès que $mult > 1$, notre algorithme semble être sensiblement plus intéressant, en termes de temps d'exécution, que l'algorithme 14. En remarquant de plus que l'algorithme 15 est très simple à implanter, il nous semble naturel de considérer cet algorithme comme un bon candidat "en pratique" pour traiter le problème de la recherche de motifs dans des termes statiques. Il serait cependant intéressant de vérifier cette affirmation non plus seulement sur des termes binaires aléatoires, mais sur des jeux de test issus de problèmes réels. Il serait également intéressant d'essayer de caractériser le comportement en moyenne des paramètres $Comp2$ (en utilisant des techniques analogues à celles utilisées par Steyaert et Flajolet dans [149]) et $Sauts$ (même si pour cette dernière statistique, les techniques de Steyaert et Flajolet ne semblent pas pouvoir être utilisées).

$\|T\|$	$mult$	$Comp1/Comp2$	$Comp1/Sauts$	t_1/t_2
100	1	13	5,3	0,6
100	2	23,9	13,9	1,2
100	3	24,4	21,5	1,6
100	5	26,4	33,9	2,2
100	7	17,5	44,3	3,4
1000	1	18,1	6,2	0,76
1000	2	60,6	17,7	2,2
1000	3	92,9	27,2	2,4
1000	5	104,1	44,4	3
1000	7	110	62,6	3,3
10000	1	20,9	7,1	0,8
10000	2	90,2	20,9	2,2
10000	3	185,2	32,2	3,1
10000	5	371,4	54,9	4,8
10000	7	483,6	74,1	7,1

TAB. 8.1 – Comparaison de l'algorithme naïf et de l'algorithme utilisant l'arborescence des suffixes - Cas des termes

Dans le cas des arborescences quelconques et du problème RMCA, nous avons obtenu un facteur d'accélération similaire pour l'algorithme utilisant l'arborescence des suffixes par rapport à l'algorithme naïf. On peut naturellement modifier légèrement l'algorithme utilisé dans le cadre du problème RMCA pour traiter le problème RMA. Cependant, les premières expérimentations réalisées sur des arborescences aléatoires semblent indiquer cet algorithme est légèrement plus lent que l'algorithme naïf.

Chapitre 9

Arborescence des suffixes d'une arborescence

Nous avons vu au chapitre précédent une première approche du problème de la recherche de motifs dans une arborescence (texte) statique, basée sur le codage de cette arborescence par un mot et l'utilisation de l'arborescence des suffixes de ce mot. Dans ce chapitre, nous proposons une autre approche de ce problème en introduisant le concept d'arborescence des suffixes d'une arborescence.

Dans la première section, nous définissons cette notion d'arborescence des suffixes d'une arborescence, la structure de données que nous proposons occupant un espace en $O(n)$ pour une arborescence A ayant n sommets. Dans la seconde section, nous rappelons l'algorithme de construction de l'arborescence des suffixes d'un mot dû à McCreight [118], dont nous nous inspirons pour définir un algorithme de construction de l'arborescence des suffixes d'une arborescence telle que nous l'avons définie. L'analyse de la complexité en temps de cet algorithme permet d'affirmer qu'il construit l'arborescence des suffixes d'une arborescence ayant n sommets en temps $O(n^2)$, mais nous conjecturons que cette complexité est en $O(n \times \log(n))$. Cette structure de donnée nous permet ensuite de résoudre certains problèmes de recherche d'un motif M de taille m dans un texte T de taille n en temps $O(m \times \log(n))$.

9.1 La structure de données

Arborescence des suffixes d'une structure abstraite. Dans ce paragraphe, nous adoptons un point de vue général et nous considérons une structure abstraite T appartenant à une famille $\mathcal{F}$ de structures possédant les propriétés suivantes :

- on peut définir une notion de *suffixe* sur les structures de $\mathcal{F}$,
- on peut associer à chaque suffixe T' de T un mot sur un alphabet Γ, noté $S(T')$.

On peut ainsi définir l'arborescence des suffixes $AS(T)$ de T comme étant l'ADR compressée des mots $S(T')$ associés aux suffixes T' de T, ou de manière équivalente par le fait que $AS(T)$ est la seule arborescence vérifiant les propriétés suivantes.

Propriété 199.

1. Chaque arête de $AS(T)$ est étiquetée par un mot sur Γ ; pour un sommet u de $AS(T)$, on note $m(u)$ le mot porté par le chemin allant de la racine de $AS(T)$ à u.
2. Pour tout suffixe T' de T, il existe un et un seul sommet u de $AS(T)$ tel que $m(u) = S(T')$ (on dit que u est le sommet final acceptant T').
3. Soient u et v deux sommets de $AS(T)$, ayant pour plus proche ancêtre commun le sommet w. Le mot $m(w)$ est alors le plus long préfixe commun de $m(u)$ et $m(v)$ (on en déduit que deux arêtes quittant un même nœud ne peuvent commencer par la même lettre de Γ et que tout nœud non final a au moins deux fils).
4. Pour tout sommet u de $AS(T)$, il existe au moins un suffixe T' de T tel que $m(u)$ est préfixe de $S(T')$.

On déduit immédiatement de cette propriété 199 le résultat suivant, qui exprime l'efficacité en espace de cette structure de données (moyennant un codage efficace de ses arêtes).

Proposition 200. *Soit T appartenant à $\mathcal{F}$. Si T comporte n suffixes, l'arborescence des suffixes $AS(T)$ de T comporte au plus $2n$ sommets.*

On qualifie "d'algorithme simple de recherche dans $AS(T)$" l'algorithme recherchant un mot sur Γ dans $AS(T)$ de manière similaire à l'algorithme 10 (page 141). Le résultat suivant montre l'intérêt de cette notion d'arborescence des suffixes pour la recherche de motif dans une structure T statique ou pour la compression de $\mathcal{F}$.

Proposition 201. *Soient T appartenant à $\mathcal{F}$, M un mot de longueur m sur Γ et $AS(T)$ l'arborescence des suffixes de T. L'algorithme simple consistant à "lire" M dans $AS(T)$ permet de déterminer en temps $O(m \times \log(min\{n,|\Gamma|\}))$ si il existe un suffixe T' de T tel que M est un préfixe de $S(T')$.*

Mots, matrices et arborescences. Comme nous l'avons vu lors du chapitre précédent, la notion d'arborescence des suffixes a été introduite à l'origine pour les mots par Weiner [164].

Dans les années 90, Giancarlo et Grossi [66, 67, 68] ont défini la notion d'arborescence des suffixes pour une matrice carrée, afin de détecter toutes les occurrences d'une matrice carrée $m \times m$ dans un texte (matrice carrée $n \times n$) en temps quasi-linéaire [1]relativement à m^2, après avoir construit l'arborescence des suffixes du texte. Ils utilisent cette approche pour l'analyse d'images représentées par des matrices de pixels (voir aussi l'article récent de Giancarlo et Guaiana [69]).

La première notion d'arborescence des suffixes d'une arborescence présente dans la littérature est due à Kosaraju [105]. Elle repose sur la notion suivante de suffixe d'une arborescence : pour une arborescence A sur Σ et x un sommet de A, le suffixe A_x de A correspond au mot porté par le chemin allant de x à la racine de A. Pour une arborescence A ayant n sommets, Breslauer [27] a proposé un algorithme construisant l'arborescence des suffixes de A en temps $O(n \times \log(|\Sigma|))$ (Shibuya [141] donne un algorithme en temps $O(n \times \log(\log(n)))$ si l'alphabet $\Sigma = \{1, \ldots, n\}$). Cette structure de données permet en particulier de trouver les k occurrences d'un mot ayant m lettres dans une arborescence ayant n sommets en temps $O(m \times \log(|\Sigma|) + k)$. Cependant il ne semble pas qu'elle permette

1. Par *quasi-linéaire*, nous entendons une complexité en temps linéaire pondérée par un facteur logarithmique, souvent dû au temps de branchement.

de résoudre des problèmes de recherche de motifs autre que des mots de façon plus efficace que l'algorithme naïf (c'est-à-dire en temps $O(n \times m)$).

Très récemment, Shibuya [141] a introduit la notion de B-arborescence des suffixes d'une arborescence k-aire A (chaque nœud a exactement k fils), reposant sur la notion suivante de suffixe : le suffixe de A enraciné en un de ses sommets x est la plus grande sous-arborescence de A complète (les sommets situés à la même hauteur sont soit tous des nœuds, soit tous des feuilles) de racine x. En se référant à la proposition 201, on peut alors détecter un motif M ayant m sommets dans un texte T (une arborescence k-aire) en temps $O(m \times \log(|\Sigma|))$ si M est une arborescence k-aire complète sur Σ.

Dans la dernière section du chapitre précédent, nous avons en fait déjà introduit une notion d'arborescence des suffixes d'une arborescence, basée sur les définitions suivantes :

- un suffixe d'une arborescence A est une sous-arborescence maximale de A (les feuille de cette sous-arborescence sont des feuilles de A) : on a donc un suffixe A_x pour chaque sommet x de A,
- le mot associé à un suffixe est obtenu par un parcours en profondeur de A_x.

Si, comme nous l'avons vu, cette structure de données semble s'avérer utile, en pratique, pour accélérer l'algorithme naïf de recherche de motifs dans une arborescence, son utilisation ne permet pas d'avoir un résultat analogue à la proposition 201 et n'améliore pas la complexité théorique de l'algorithme naïf (complexité quadratique).

Dans la suite de ce chapitre, nous proposons une nouvelle définition de l'arborescence des suffixes (conforme à la propriété 199), reposant sur la même notion de suffixe que précédemment (à savoir une sous-arborescence), mais utilisant le codage d'un suffixe par un mot calculé lors d'un *parcours en largeur* de ce suffixe. La structure de données ainsi obtenue permet de traiter en temps quasi-linéaire certains problèmes de recherche de motifs dans une arborescence.

Suffixe et préfixe d'une arborescence, et mots associés. Le principe guidant la définition de l'arborescence des suffixes d'une arborescence que nous proposons est de considérer un mot de longueur h comme une arborescence de hauteur h ayant un seul sommet par niveau (arborescence *filiforme*). Une arborescence de hauteur h (voir définition 180) peut donc être vue comme un "mot généralisé" (chaque niveau comporte non plus un sommet, mais un ensemble de sommets).

Définition 202. Soient A une arborescence de hauteur h et $1 \leq i \leq h$.

- On note $A[i]$ l'ensemble des sommets x de A situés à hauteur i : on qualifie $A[i]$ de $i^{ème}$ niveau de A.
- Si $A[i]$ comporte k sommets, on note $A[i,j]$ le $j^{ème}$ sommet ($1 \leq j \leq k$), à partir de la gauche (ou par ancienneté), de $A[i]$.

Définition 203. Soient A une arborescence et $x = A[i,j]$ un sommet de A.

- On appelle *suffixe de A enraciné en x*, noté $Suf(A,x)$ ou A_x, la sous-arborescence maximale de A de racine x.
- On appelle *préfixe de A finissant en x*, noté $Pref(A,x)$, la sous-arborescence de A constituée des sommets $x' = A[i',j']$ de A tels que soit $i' < i$, soit $i' = i$ et $j' \leq j$.

Il reste à définir l'alphabet Γ des mots $S(A_x)$ associés aux suffixes A_x d'une arborescence A. Pour cela, nous introduisons la notion de décalage d'un sommet et de T-lettre associée à un sommet.

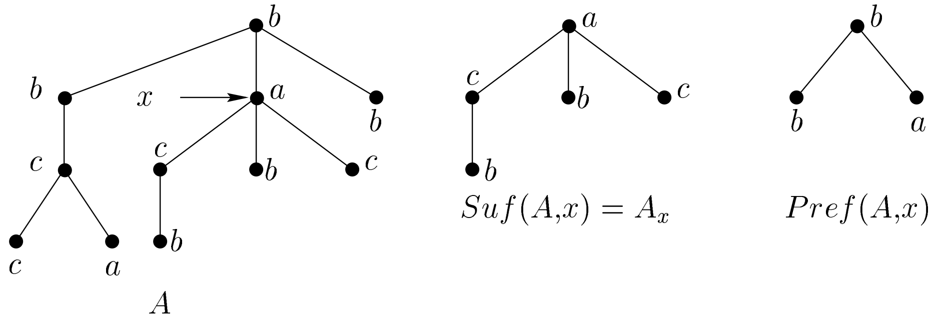


FIG. 9.1 – Illustration de la notion de suffixe et de préfixe d'une arborescence

Définition 204. Soient A une arborescence sur Σ et $x = A[i,j]$ un sommet de A. On appelle *décalage de x dans A*, noté $D_A(x)$, l'entier défini par

- si $x = A[1,1]$ est la racine de A, alors $D_A(x) = 1$,
- si $j = 1$, alors $D_A(x) = j'$ où $A[i-1,j']$ est le père de x,
- sinon, $D_A(x) = j' - j''$ où $A[i-1,j']$ est le père de x et $A[i-1,j'']$ le père de $A[i,j-1]$.

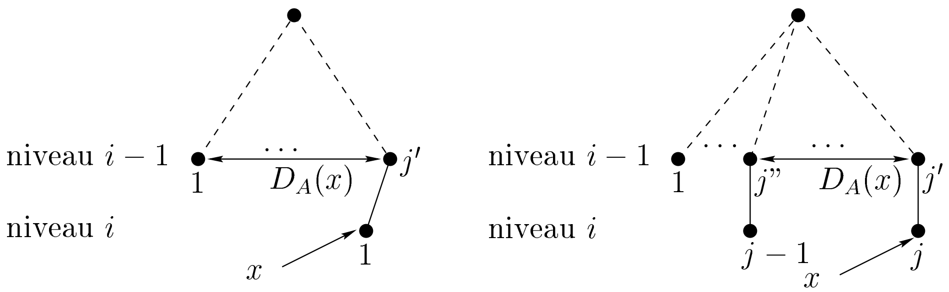


FIG. 9.2 – Le décalage d'un sommet

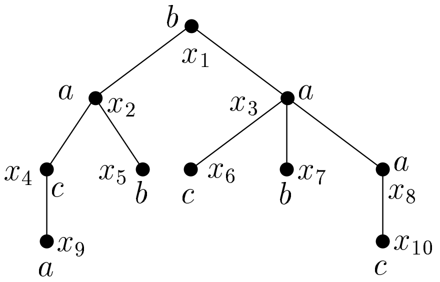
Définition 205. Soient A une arborescence sur Σ et $x = A[i,j]$ un sommet de A. La T-lettre associée à x dans A, notée x_A, est le triplet $(a_x, D_A(x), e(x))$ où $a_x = 1$ si $j = 1$ et $a_x = 0$ sinon, et $e(x)$ est l'étiquette de x.

L'alphabet Γ correspondant à une arborescence A est alors l'alphabet des T-lettres associées à A, c'est-à-dire des triplets (a,b,c) où a est un booléen, b un entier positif ou nul et c une lettre de Σ.

Définition 206. Le *parcours en largeur* d'une arborescence ordonnée A de hauteur h est le parcours consistant à visiter d'abord la racine, puis les sommets à hauteur 2 (on visite successivement $A[2,1]$, $A[2,2]$, …, $A[2,k]$, …), puis les sommets à hauteur 3, …, et pour finir les sommets à hauteur h.

Définition 207. Soit A une arborescence sur Σ ayant n sommets. On note $S(A)$ le mot de longueur n sur Γ tel que, si x est le $i^{ème}$ sommet visité lors du parcours en largeur de A, la $i^{ème}$ lettre de $S(A)$ est la T-lettre x_A associée à x dans A.

Notation. Par la suite, étant donnée une arborescence T sur Σ ayant n sommets, on note x_i le $i^{ème}$ sommet visité lors du parcours en largeur de T. De plus on désigne par S le mot $S(T)$ associé à T et, pour chaque sommet x_i de T, on note T_i l'arborescence suffixe T_{x_i} et S_i le mot $S(T_i)$.



$$S = (1,1,b)(1,1,a)(0,0,a)(1,1,c)(0,0,b)(0,1,c)(0,0,b)(0,0,a)(1,1,a)(0,4,c)$$

$$S_3 = (1,1,a)(1,1,c)(0,0,b)(0,0,a)(1,3,c)$$

FIG. 9.3 – Exemples de mot associé à une arborescence

L'arborescence des suffixes d'une arborescence. L'arborescence des suffixes d'une arborescence T ayant n sommets, notée $AS(T)$, est obtenue en compressant l'ADR des mots S_i pour $i = 1, \dots, n$. Comme plusieurs suffixes (mots S_i) d'une même arborescence peuvent être identiques, on associe à chaque sommet final u la liste des indices i tels que $m(u) = S_i$. Dans la figure suivante, on représente le sommet final acceptant le mot S_i par un rectangle contenant i.

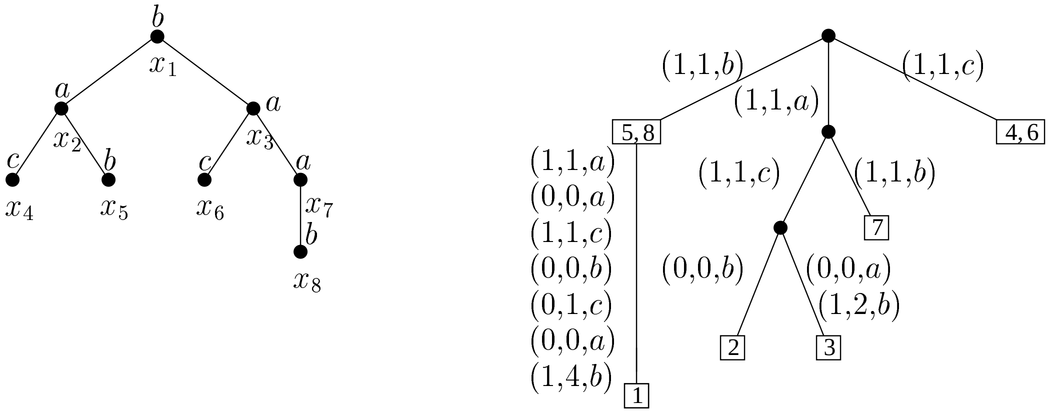
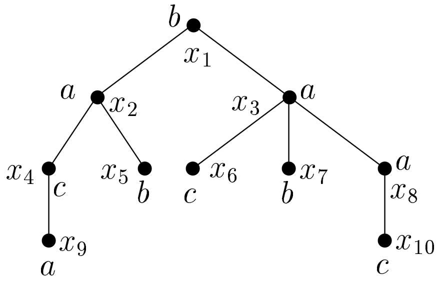


FIG. 9.4 – L'ADR compressée des suffixes d'une arborescence

Si la proposition 200 nous assure que $AS(T)$ comporte au plus $2n$ sommets, lorsque T en comprend n, pour obtenir une structure de donnée occupant un espace en $O(n)$, il est nécessaire de coder chaque mot sur Γ associé à une arête en espace $O(1)$. Dans le cas des mots, le codage d'un arc par un couple d'entiers vérifie cette condition tout en permettant de déterminer le facteur correspondant à une arête (d,f) en temps $O(f-d+1)$.

Dans le cas des arborescences, on peut remarquer que pour tout sommet x_i de T, tout facteur $S_i[l..l']$ de S_i peut être codé par le triplet (x_i,x',x'') où x' (resp. x'') est le sommet de T_i correspondant à la T-lettre $S_i[l]$ (resp. $S_i[l']$).



$$S_3 = (1{,}1{,}a)(1{,}1{,}c)(0{,}0{,}b)(0{,}0{,}a)(1{,}3{,}c)$$

$$S_3[2..4] = (1{,}1{,}c)(0{,}0{,}b)(0{,}0{,}a) \text{ est codé par } (x_3{,}x_6{,}x_8)$$

FIG. 9.5 – Codage d'un facteur par un triplet de sommets

On peut donc représenter le facteur associé à une arête de $AS(T)$ par un tel triplet, ce qui, dans le cas de notre exemple, donne l'arborescence représentée figure 9.6.

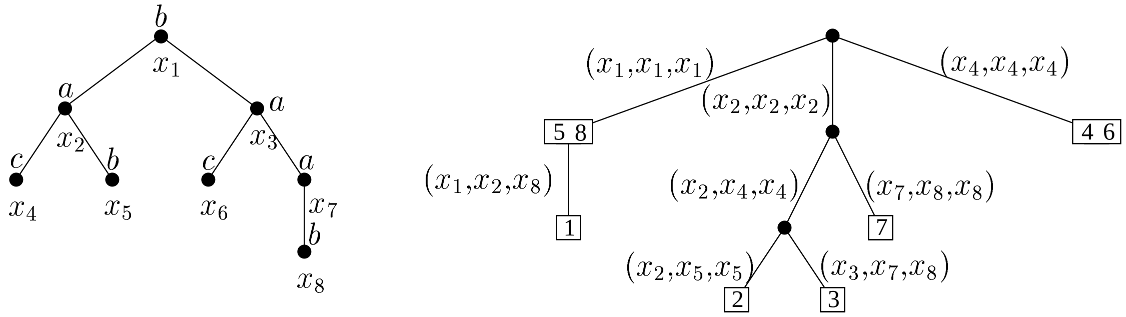


FIG. 9.6 – L'arborescence des suffixes d'une arborescence

On obtient ainsi une structure de données occupant un espace en $O(n)$. Les deux sections suivantes présentent un algorithme permettant de construire cette structure.

Remarque 208. Nous verrons dans la section 9.3 un prétraitement de l'arborescence T permettant, étant donnés trois sommets x, x' et x'' codant un mot comportant k T-lettres, d'obtenir ce mot en temps $O(k)$.

Notation. Par la suite, on considère tout sommet final associé à au moins deux suffixes comme un nœud.

Notation. Dans la suite de ce chapitre, on utilise la notation (x,x',x'') pour représenter indifféremment un mot composé de T-lettres et l'ensemble de sommets de l'arborescence

T_x lui correspondant. De plus, lorsqu'il n'y a pas ambiguïté, on identifie un sommet de cet ensemble avec la T-lettre correspondante.

9.2 L'algorithme de McCreight

Dans cette section, nous présentons l'algorithme de construction de l'arborescence des suffixes d'un mot T (de longueur n sur un alphabet Σ) dû à McCreight [118], algorithme que nous reprenons dans la section suivante pour construire l'arborescence des suffixes d'une arborescence. La présentation que nous en donnons est inspirée du livre de Crochemore et Rytter [52, section 5.3].

Un algorithme simple mais non optimal. Le principe de cet algorithme est de débuter avec l'ADR correspondant au seul mot T (on note cet ADR AS_1), puis de construire incrémentalement $AS(T)$ en insérant les suffixes propres de T dans cette ADR par ordre de longueur décroissante (on note AS_i l'ADR compressé ainsi obtenu après insertion du suffixe $T[i..n]$ et f_i le sommet final correspondant au suffixe $T[i..n]$).

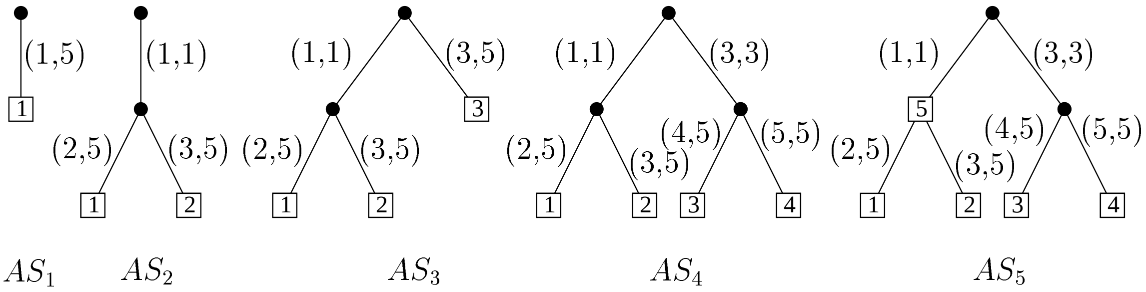


FIG. 9.7 – Les arborescences successives créées par l'algorithme de McCreight si $T = aabba$

Pour insérer le mot $T[i..n]$ dans AS_{i-1}, une solution simple consiste à utiliser une procédure qui, étant donnés un sommet u de AS_{i-1} (normalement la racine) et un mot W, suit lettre à lettre le plus long préfixe de W correspondant à un chemin issu de u et renvoie le couple (v,k) où v est le sommet final de ce chemin (v est créé si nécessaire) et k est la longueur du suffixe de W non reconnu. On appelle cette opération la *procédure de recherche lente*, notée **ChercherL**$(u,\ W)$.

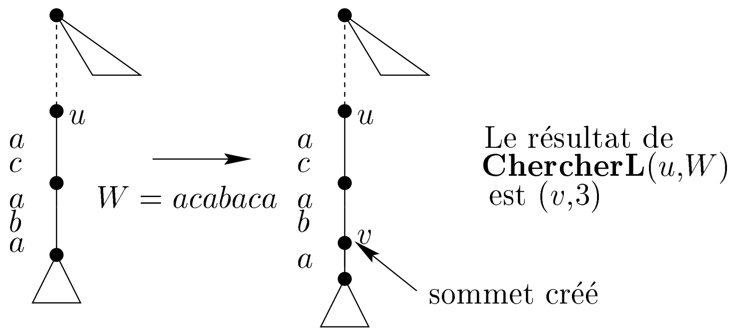


FIG. 9.8 – La procédure **ChercherL**

En effectuant les mêmes remarques que celles faisant suite à l'algorithme 10, il apparaît clairement que la complexité en temps de cette procédure est en $O((|m(v)| - |m(u)|) \times \log(|\Sigma|))$. Pour construire l'arborescence des suffixes $AS(T)$, on peut alors utiliser l'algorithme suivant, qui a une complexité en temps en $O(n^2)$.

Algorithme 22: Construction de l'arborescence des suffixes de T : algorithme simple

Donnée : mot T sur Σ de longueur n
début
  créer AS_1 (ADR à deux sommets, la racine r et la feuille f_1 dont l'arête est étiquetée $(1,n)$);
  pour i allant de 2 à n **faire**
    $(v,k) :=$ **ChercherL**$(r, T[i..n])$;
    si $k = 0$ **alors** $f_i := v$;
    sinon créer f_i comme fils de v relié à celui-ci par l'arête $(n - k + 1,n)$;
fin

Une première amélioration : les liens suffixes.

Notation. Dans la suite de cette section, on note g_i le premier nœud ancêtre de f_i (f_i est le sommet correspondant au suffixe $T[i..n]$) dans AS_i ($g_i = f_i$ si f_i est un nœud et g_i est le père de f_i si f_i est une feuille) et h_i le père[2] de g_i dans AS_i.

Pour accélérer ce premier algorithme, on s'appuie sur les résultats suivants.

Lemme 209. *Soient $1 \leq i \leq n$ et u un nœud de AS_i autre que g_i. Il existe dans AS_i un sommet v tel que $m(u) = a.m(v)$ où a est une lettre de Σ ($m(v)$ est le plus long suffixe propre de $m(u)$).*

Preuve. Posons $m(u) = a.s$, avec $a \in \Sigma$ et $s \in \Sigma^*$. Si $u \neq g_i$, u étant un nœud de AS_i, il existe deux suffixes $T[j_1..n]$ et $T[j_2..n]$ avec $j_1 < j_2 < i$ tels que $m(u)$ est le plus long préfixe commun de ces deux mots. On en déduit que le plus long préfixe commun des deux suffixes $T[j_1+1..n]$ et $T[j_2+1..n]$ est le mot s et, par définition de AS_i, il existe un sommet v de AS_i tel que $m(v) = s$. $\square$

2. Dans la suite de cette section, nous ne prenons pas en compte les cas $g_i = r$ ou $h_i = r$, où r est la racine de AS_i. Ces deux cas alourdissent la présentation de l'algorithme de McCreight, bien que leur prise en compte ne pose pas de problème particulier.

Propriété 210. Pour $2 \leq i \leq n$, AS_i a au plus un ou deux sommets de plus que AS_{i-1}, les sommets f_i et g_i, qui peuvent être confondus.

On peut utiliser ces résultats et supposer ainsi qu'après avoir inséré le suffixe $T[i+1..n]$ dans AS_i (et donc construit AS_{i+1}), tout sommet u, autre que la racine, qui était présent dans AS_i possède une arête spéciale (non étiquetée) reliant u au sommet v tel que $m(u) = a.m(v)$. On appelle cette arête *le lien suffixe de* u, noté $v = LS(u)$. Pour que cette hypothèse soit vérifiée, compte-tenu de la propriété 210, il suffit d'ajouter lors de la construction de AS_{i+1} un seul lien suffixe, à savoir $LS(g_i)$.

On représente les liens suffixes en pointillés sur la figure suivante.

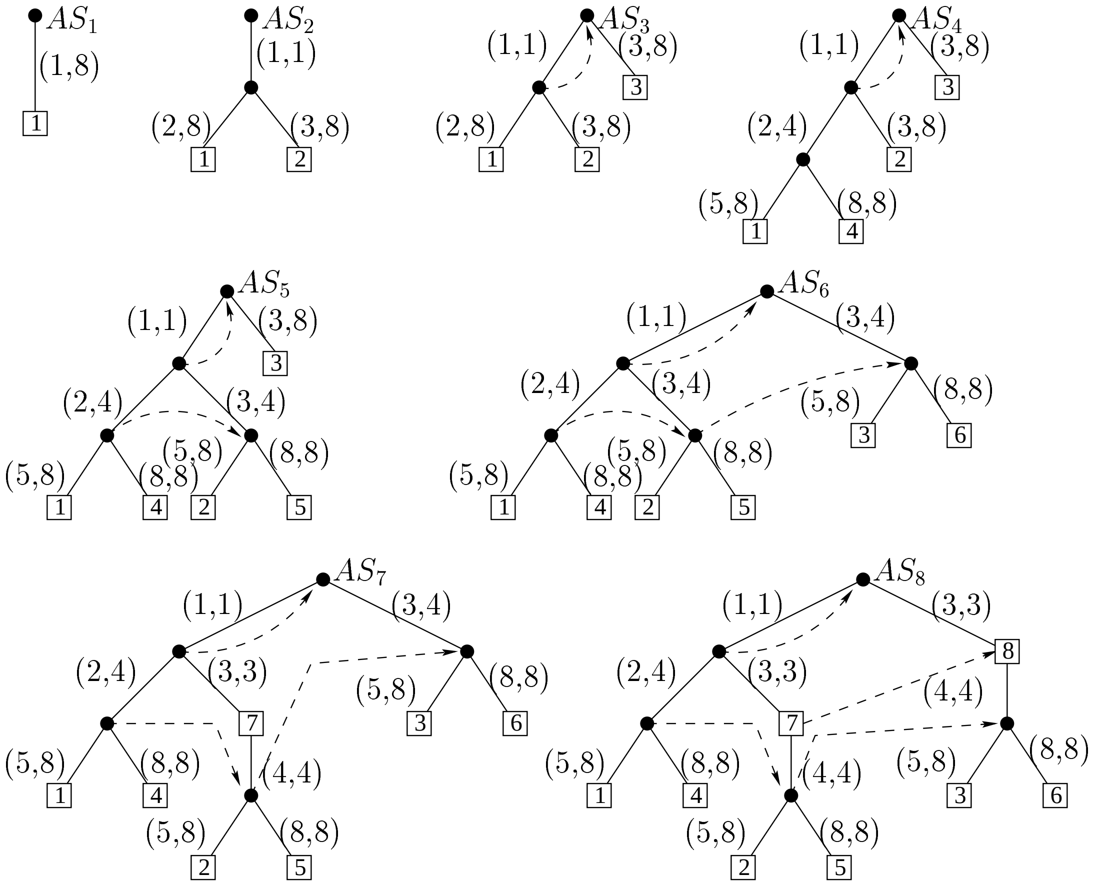


FIG. 9.9 – Les liens suffixes lors de la construction de $AS(aabaabab)$

Pour insérer le suffixe $T[i+1..n]$ dans AS_i, on peut alors suivre un schéma *Remonter-Traverser-Descendre* :

1. *remonter* jusqu'au père h_i de g_i (h_i possède un lien suffixe d'après le lemme 209),
2. suivre ce lien suffixe (*traverser*) pour aboutir au sommet $u = LS(h_i)$,
3. insérer depuis u le suffixe $T[i+1+|m(u)|..n]$ (*descendre*).

Ce schéma permet ainsi d'éviter d'avoir à relire le préfixe $m(u)$ de $T[i..n]$ comme cela était le cas dans l'algorithme simple présenté auparavant. Il nécessite cependant, comme on l'a remarqué précédemment, qu'après l'insertion de $T[i+1..n]$ dans AS_i, le nœud g_i

possède un lien suffixe. De plus, pour un mot T composé de n occurrences de la même lettre de Σ, on remarque que la complexité en temps reste quadratique en n. Pour améliorer cela, on combine l'utilisation des liens suffixes avec le recours à une procédure de "recherche" d'un mot dans un ADR plus efficace que **ChercherL**.

Une seconde amélioration : la recherche rapide. Cette amélioration repose sur le résultat suivant.

Lemme 211. *Dans l'arborescence AS_i, si $m(g_i) = a.s$, avec $a \in \Sigma$ et $s \in \Sigma^*$, alors il existe un sommet u tel que s est un préfixe de $m(u)$.*

Preuve. Si g_i est un sommet déjà présent dans AS_{i-1}, le sommet $v = LS(g_i)$ vérifie cette propriété et on a même $m(v) = s$. Sinon, g_i a été créé lors de l'insertion de $T[i..n]$. Comme g_i est par définition un nœud, il existe un suffixe $T[j..n]$ de T avec $j < i$ tel que $m(g_i)$ est un préfixe de $T[j..n]$. En prenant pour u le sommet f_{j+1} de AS_i, alors on constate que s est un préfixe de $m(u)$. □

Corollaire 212. *Soient (k,l) l'étiquette de l'arête reliant h_i à g_i dans AS_i et $u = LS(h_i)$. Il existe un descendant v de u tel que le mot correspondant à (k,l) est un préfixe du mot correspondant au chemin allant de u à v.*

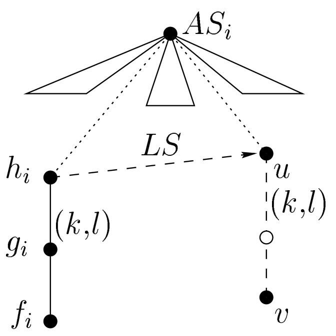


FIG. 9.10 – Illustration du Corollaire 212

Pour déterminer, lors de la partie *descendre*, le préfixe de $T[i+1+|m(u)|..n]$ correspondant à (k,l), il n'est donc pas nécessaire de le chercher en le lisant lettre à lettre car on sait qu'il sera entièrement reconnu à partir du sommet $u = LS(h_i)$. On utilise plutôt la procédure suivante de *recherche rapide* à partir d'un sommet u d'un mot W que l'on sait à priori être reconnu. Cette procédure renvoie comme résultat le sommet v (en le créant si nécessaire) tel que le chemin de u à v correspond au mot W.

Algorithme 23: ChercherR

Données : AS_i, un sommet u de AS_i et un mot W de longueur p
début
soit (i,j) l'arête quittant u (vers un sommet v) telle que $T[i] = W[1]$;
si $j - i + 1 = p$ **alors retourner** v;
sinon si $j - i + 1 < p$ **alors ChercherR**(v, $W[j - i + 2..p]$);
sinon insérer un sommet x entre u et v de sorte que l'arête reliant u à x (resp. x à v) est étiquetée $(i,i + p - 1)$ (resp. $(i + p,j)$);
fin

Propriété 213. La complexité d'un appel à **ChercherR**(AS_i, u, W) renvoyant comme résultat un sommet v est en $O(k \times \log(|\Sigma|))$, où k est le nombre de nœuds situés sur le chemin allant de u à v.

Ainsi, McCreight [118] a pu proposer l'algorithme suivant pour construire $AS(T)$, basé sur l'utilisation des liens suffixes et de **ChercherR** (rappelons que nous passons sous silence les cas particuliers $g_i = r$ ou $h_i = r$).

Algorithme 24: Algorithme de McCreight

Donnée : mot T sur Σ de longueur n
début
créer AS_1 (ADR à deux sommets, la racine r et la feuille f_1 dont l'arête est étiquetée $(1,n)$);
pour i allant de 2 à n **faire**
  soit (l_1,l'_1) l'arête reliant g_{i-1} à f_{i-1};
  soit (l_2,l'_2) l'arête reliant h_{i-1} à g_{i-1};
  $u := LS(h_{i-1})$;
  $x :=$**ChercherR**(u, $T[l_2..l'_2]$);
  $LS(g_{i-1}) := x$;
  $(g_i,k) :=$**ChercherL**(v, $T[l_1..l'_1]$);
  si $k = 0$ **alors** $f_i := g_i$;
  sinon créer f_i comme fils de g_i relié à celui-ci par une arête étiquetée $(n - k + 1,n)$;
fin

Observation 214. Si lors de l'étape de recherche rapide le nœud x est créé, il est clair que la seule arête quittant ce nœud ne commence pas par $T[l_2]$ et il n'est pas nécessaire d'effectuer la phase recherche lente. On peut de suite poser $g_i = x$ et insérer f_i comme fils de g_i, relié à ce dernier par une arête étiquetée (l_2,l'_2).

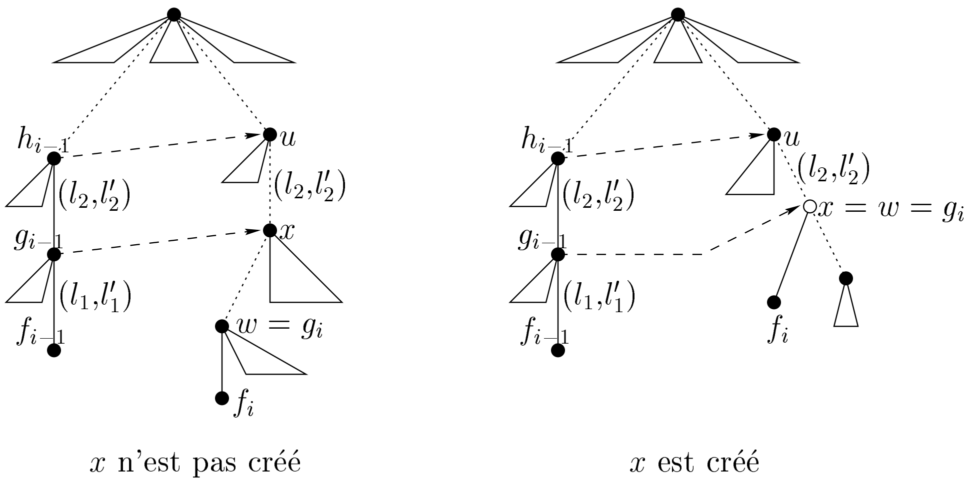


FIG. 9.11 – Illustration de l'observation 214

Proposition 215. *Soit T un mot de longueur n sur l'alphabet Σ. L'algorithme de Mc-Creight calcule l'arborescence des suffixes $AS(T)$ en temps $O(n \times \log(|\Sigma|))$.*

Preuve. La complexité en temps d'un appel à **ChercherL** est proportionnelle au nombre de comparaisons entre un caractère du mot cherché et un caractère de T. On peut remarquer que, lors de l'insertion de $T[i..n]$ dans AS_{i-1}, le nombre de caractères lus par **ChercherL** est $|m(g_i)| - |m(g_{i-1})| + 2$. On en déduit que le nombre total de comparaisons réalisées par les appels à **ChercherL** est exactement

$$\sum_{i=2}^{n}(|m(g_i)| - |m(g_{i-1})| + 2) = |m(g_n)| - |m(g_1)| + 2(n-1) = 3n.$$

La complexité totale des appels à **ChercherL** est donc en $O(n \times \log(|\Sigma|))$.

Examinons maintenant la procédure **ChercherR**. La complexité de l'appel à **ChercherR** lors de la construction de l'arborescence AS_i est proportionnelle au nombre de sommets de AS_i traités lors de cet appel, nombre que nous notons int_i. On remarque que chaque sommet rencontré (sauf éventuellement le dernier) depuis le sommet $u = LS(h_{i-1})$ est un ancêtre de h_i. On en déduit que

$$int_i - 1 \leq |m(h_i)| - |m(u)| \leq |m(h_i)| - (|m(h_{i-1})| - 1)$$

(on ne peut transformer le second $\leq$ en $=$ car h_i et h_{i+1} peuvent être égaux à la racine). Ainsi

$$int_i \leq |m(h_i)| - |m(h_i)| + 2.$$

Un calcul similaire au précédent permet d'affirmer que la complexité des appels à **ChercherR** est en $O(n \times \log(|\Sigma|))$. □

9.3 Le cas des arborescences

Nous présentons maintenant une extension de l'algorithme de McCreight pour construire l'arborescence des suffixes $AS(T)$ d'une arborescence T.

Prétraitement de T. Comme nous l'avons mentionné en présentant la structure de données, nous considérons une arborescence T de hauteur h comme un "mot généralisé" de longueur h. Dans ce paragraphe, nous décrivons un prétraitement permettant de manipuler efficacement T et notamment de décrire un mot (x,x',x'') de k T-lettres en temps $O(k)$ (remarque 208). Dans un premier temps, on s'inspire de la représentation d'un mot par un tableau pour coder T (voir figure 9.12) :

- chaque sommet $x = T[i,j]$ autre que la racine est codé par le couple $(e(x),j')$ si le père de x est le sommet $T[i-1,j']$, la racine étant codée $(e(x),0)$;
- si le niveau $T[i]$ comporte k sommets $(1 \leq i \leq h(T))$, il est représenté par le tableau des couples associés aux sommets de $T[i]$ (ce tableau comportant k éléments) ;
- l'arborescence T est représentée par le tableau des h tableaux $T[i]$.

On suppose de plus qu'on connaît le nombre de sommets n_i présents sur chaque niveau i $(n_1 = 1)$.

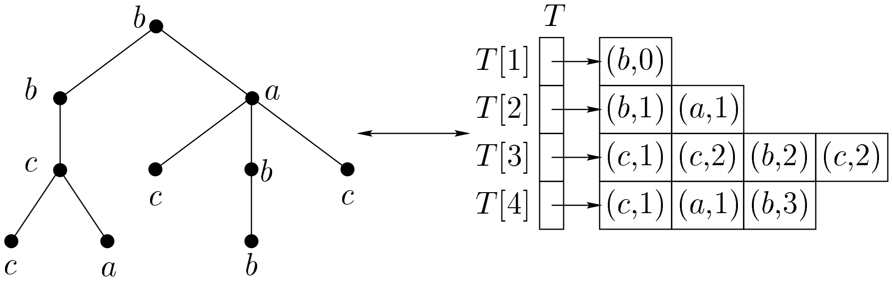


FIG. 9.12 – Codage d'une arborescence par des tableaux

Ce codage d'une arborescence est une généralisation de la représentation d'un mot sous forme de tableau (en considérant un mot comme une arborescence ayant une seule feuille, chaque niveau comporte exactement un sommet). Cependant, pour pouvoir se "déplacer" rapidement dans les tableaux codant T, nous avons besoin du résultat suivant dû à Berkman et Vishkin [18].

Définition 216. Soient $x = T[i,j]$ un sommet de T, p un entier tel que $0 \leq p \leq i-1$ et q un entier tel que $1 \leq q \leq h(T) - i$. On note $Anc(x,p)$ l'ancêtre de x situé sur le niveau $T[i-p]$ et $Succ(x,k)$ le premier sommet y, s'il existe, du niveau $T[i+q]$ tel que le sommet $Anc(y,q) = T[i,j']$ vérifie $j' \geq j$.

Ainsi, on peut noter que le successeur d'un sommet n'est pas toujours un de ses descendants.

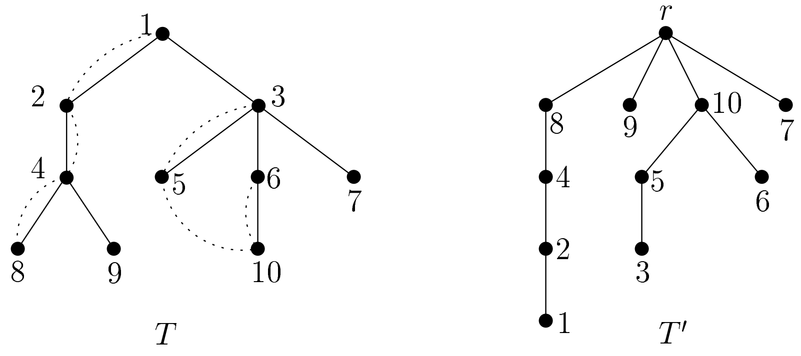
Théorème 217. [18] *Soient T une arborescence à n sommets, $x = T[i,j]$ et $Anc(x,k) = T[i',j']$. On peut prétraiter T en temps et espace $O(n)$ de sorte que, connaissant i, j et k, la détermination de i' et j' s'effectue en temps $O(1)$.*

L'algorithme de Berkman et Vishkin [18] étant extrêmement complexe (à décrire et à mettre en œuvre), il ne nous est pas possible, dans le cadre de ce mémoire, d'en faire une présentation synthétique.

Corollaire 218. *Soient T une arborescence à n sommets, $x = T[i,j]$ un sommet de T et $Succ(x,k) = T[i',j']$. On peut prétraiter T en temps et espace $O(n)$ de sorte que, connaissant i, j et k, la détermination de i' et j' s'effectue en temps $O(1)$.*

Preuve. Lors d'un parcours en profondeur de T, on associe à chaque sommet $x = T[i,j]$ de T le sommet $s_x = T[i+1,j']$ dont le père $T[i,j'']$ vérifie $j'' \geq j$, lorsqu'un tel sommet existe (ces associations sont matérialisées par les arêtes en pointillées sur la figure ci-dessous). Puis on construit une arborescence T' ayant $n+1$ sommets (les sommets de T et une racine r) de la manière suivante (illustrée par la figure ci-dessous dans laquelle nous avons numéroté les sommets de T) :

- si, pour un sommet x, s_x n'est pas défini, on crée une arête entre r et x,
- sinon on crée une arête entre s_x et x.



Par construction, on a alors $y = Succ(x,k)$ dans T si et seulement si $x = Anc(y,k)$ dans T'. La construction de T' s'effectuant en temps $O(n)$, on déduit du théorème 217 le résultat annoncé. □

Lemme 219. *Soient T une arborescence à n sommets, et x et x' deux sommets de T. On peut prétraiter T en temps et espace $O(n)$ de sorte qu'il est possible de déterminer si x' est un ancêtre de x en temps $O(1)$.*

Preuve. Il suffit d'associer à tout sommet y son indice $ind(y)$ lors d'un parcours en profondeur de T et la taille $|T_y|$ de la sous-arborescence T_y dont il est racine (voir section 7.4). □

Lemme 220. *Soit T une arborescence à n sommets. Après avoir appliqué les prétraitements des théorème 217, Corollaire 218 et lemme 219, on peut effectuer les opérations suivantes en temps $O(1)$:*

1. *étant donnés un sommet $x = T[i,j]$ et deux entiers k et l, déterminer i' et j' tels que $T_x[k,l] = T[i',j']$;*
2. *étant donnés un sommet $x = T[i,j]$ et $y = T[k,l]$ un ancêtre de x, calculer la T-lettre x_{T_y} ;*
3. *étant donnés un sommet $x = T[i,j]$ et $y = T[k,l]$ un ancêtre de x, déterminer i' et j' tels que $x' = T[i',j']$ soit le sommet suivant (resp. précédant) x lors d'un parcours en largeur de T_y ;*
4. *étant donnés un sommet $x = T[i,j]$ et un entier k, calculer le nombre de sommets présents dans le $k^{\text{ème}}$ niveau de T_x.*

Preuve. Examinons la première opération. Pour calculer i' et j', il suffit de déterminer le sommet $y = T_x[k,1] = T[i'',j''] = Succ(x,k)$ (Corollaire 218) et si x est ancêtre de ce sommet (lemme 219), alors $i' = i''$ et $j' = j'' + l$.
Pour la seconde opération, notons $x_{T_y} = (a_x,D_{T_y}(x),e(x))$. Pour calculer a_x, il suffit de

tester si $Succ(y,i-k) = x$ (Corollaire 218). Pour calculer $D_{T_y}(x)$, si $x = Succ(y,i-k)$ (et donc $a_x = 1$) alors $D_{T_y}(x) = j'$ où $T[i-1,j']$ est le père de $x = T[i,j]$ (théorème 217). Sinon ($a_x = 0$), $D_{T_y}(x) = j' - j''$ où $T[i-1,j']$ est le père de $x = T[i,j]$ et $T[i-1,j'']$ est le père de $T[i,j-1]$ (théorème 217).
Pour la troisième opération, si x est le dernier sommet de son niveau dans T_y (ce qui équivaut à $j = n_i$ ou $ind(T[i,j+1]) > ind(y) + |T_y|$), le sommet suivant x est $x' = Succ(t,i-k+1)$ et on peut déterminer i' et j' en temps constant grâce au Corollaire 218. Le raisonnement dans le cas où x' précède x est similaire.
Finalement, pour calculer le nombre de sommets présents dans le niveau $T_x[k]$, on détermine $Succ(x,k) = T[i+k,j']$. Si $j = n_j$ (x est le dernier sommet de son niveau dans T) ou $T[i+k,n_{i+k}]$ est un descendant de x, il y a $n_{i+k} - j' + 1$ sommets dans $T_x[k]$. Sinon $x' = T[i,j+1]$ étant le sommet suivant x sur son niveau, on détermine $Succ(x',k) = T[i+k,j'']$. Il y a alors $j'' - j'$ sommets dans $T_x[k]$. □

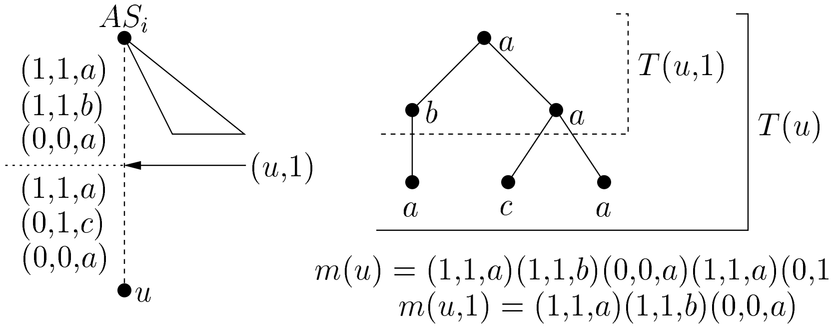
Le principe de l'algorithme de construction de l'arborescence des suffixes $AS(T)$ d'une arborescence. L'algorithme que nous décrivons par la suite est basé sur le même principe que l'algorithme de McCreight présenté section précédente, à savoir une construction incrémentale de $AS(T)$ basée sur un schéma "Remonter-Traverser-Descendre". Elle utilise une notion de liens suffixes et une procédure de recherche rapide d'un mot à partir d'un sommet de l'arborescence des suffixes. Rappelons les notations suivantes :

- x_i est le $i^{ème}$ sommet de l'arborescence T visité lors d'un parcours en largeur, T_i la sous-arborescence suffixe de T de racine x_i et S_i le mot de T-lettres $S(T_i)$;
- AS_i est l'ADR compressé des mots de T-lettres $S_1, \ldots ,S_i$;
- f_i est le sommet final de AS_i correspondant à la lecture du mot S_i, g_i le premier nœud ancêtre de f_i dans AS_i (g_i est égal à f_i si f_i est un nœud) et h_i le père de g_i dans AS_i.

Sommets implicites et liens suffixes. Dans ce paragraphe, nous étendons la notion de lien suffixe à l'arborescence des suffixes d'une arborescence. Dans un premier temps, nous introduisons la notion de sommet implicite.

Notation. Pour tout sommet u de AS_i, il existe une sous-arborescence de T, notée $T(u)$, telle que $m(u) = S(T(u))$ (il peut y avoir plusieurs occurrences de $T(u)$ dans T).

Définition 221. On appelle *sommet implicite* de AS_i un couple (u,l) où u est un sommet de AS_i autre que la racine et l un entier tel que, si $T(u)$ est de hauteur h, alors $0 \leq l \leq h$. On note $T(u,l)$ l'arborescence obtenue en supprimant les l derniers niveaux de $T(u)$ et $m(u,l)$ le mot associé à $T(u,l)$.



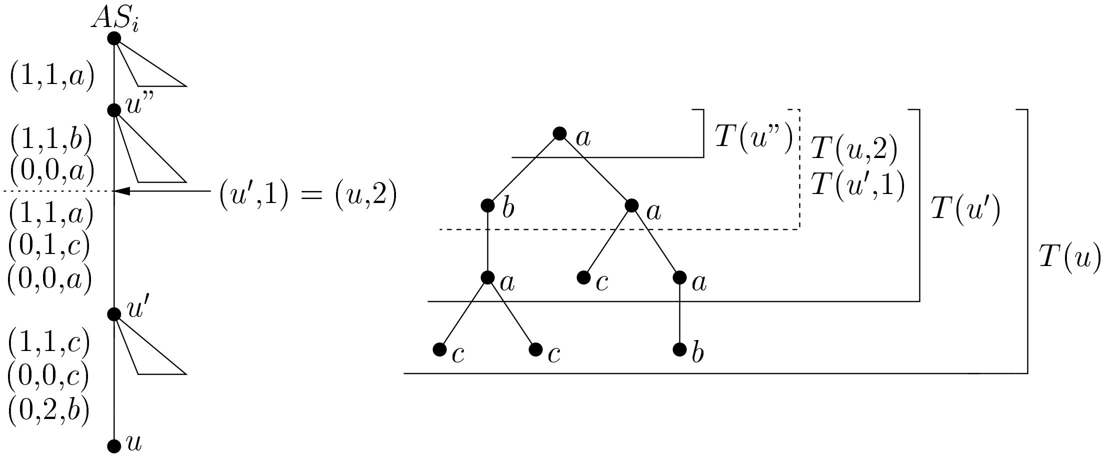
$$m(u) = (1{,}1{,}a)(1{,}1{,}b)(0{,}0{,}a)(1{,}1{,}a)(0{,}1{,}c)(0{,}0{,}a)$$
$$m(u{,}1) = (1{,}1{,}a)(1{,}1{,}b)(0{,}0{,}a)$$

FIG. 9.13 – Illustration de la notion de sommet implicite

On remarque alors immédiatement que si $l = 0$, le sommet implicite $(u{,}l)$ correspond en fait au sommet u ($T(u{,}0) = T(u)$). De plus, plusieurs sommets implicites pouvant représenter la même arborescence, on introduit la notion de sommet implicite minimal.

Définition 222. Un sommet implicite $(u{,}l)$ de AS_i est *minimal* si et seulement si il n'existe pas de sommet implicite $(u'{,}l')$ où u' est un ancêtre de u tel que $m(u'{,}l') = m(u{,}l)$.

Par exemple, soient u, u' et u'' trois sommets de AS_i tels que u' est le père de u et u'' le père de u', correspondant aux arborescences $T(u)$, $T(u')$ et $T(u'')$ représentées par la figure suivante.



Les sommets implicites $(u'{,}1)$ et $(u{,}2)$ correspondent à la même arborescence ($T(u'{,}1) = T(u{,}2)$), mais $(u{,}2)$ n'est pas minimal, alors que $(u'{,}1)$ l'est.

On associe naturellement à la notion de sommet implicite minimal une procédure de minimisation d'un sommet implicite $(u{,}l)$, qui consiste, étant donné $(u{,}l)$ à déterminer l'unique sommet implicite minimal $(u'{,}l')$ tel que $T(u'{,}l') = T(u{,}l)$. Son principe consiste simplement à remonter depuis le sommet u à son ancêtre u' en calculant, pour chaque arête suivie le nombre de niveaux remontés lors de cette étape.

Algorithme 25: Minimiser

Donnée : AS_p et un sommet implicite (u,l)
début
  $u' := u$ et $l' := l$;
  soient v le père de u' et $(x = T[i,j], T[i',j'], T[i'',j''])$ le mot de T-lettres porté par l'arête reliant v à u';
  tant que $i' > i'' - l'$ **faire**
    $u' := v$;
    si $T[i',j']$ est le premier sommet du niveau $i' - i + 1$ de T_x **alors**
      $l' := l' - (i'' - i' + 1)$;
    sinon $l' := l' - (i'' - i')$;
    soient v le père de u' et $(x = T[i,j], T[i',j'], T[i'',j''])$ l'arête reliant v à u';
  retourner (u',l');
fin

Pour deux sommets de T, $x = T[i,j]$ et $y = T[i',j']$ où y est un descendant de x, le sommet y est le premier sommet du niveau $i' - i + 1$ de T_x si et seulement si la T-lettre $y_{T_x} = (a_y, D_{T_x}(y), e(y))$ vérifie $a_y = 1$. Il suffit donc de calculer cette T-lettre y_{T_x} pour déterminer si y est le premier sommet d'un niveau dans T_x, ce qui peut être fait en temps constant (lemme 220). Le nombre d'itérations de la boucle **tant que** étant égal au nombre de sommets situés sur le chemin allant de u' à u, on en déduit la propriété suivante.

Propriété 223. La complexité en temps d'un appel **Minimiser**$((u,l))$ renvoyant comme résultat un sommet implicite (u',l') est en $O(k)$, où k est le nombre de sommets présents dans AS_p sur le chemin allant de u' à u.

Nous introduisons maintenant la notion de restriction d'une arête, illustrée par la figure 9.14.

Définition 224. Soient (x,x',x'') un mot de T-lettres et $\{x' = z_1, \ldots, z_l = x''\}$ les sommets de la sous-arborescence T_x correspondant à ce mot. Supposons que le sommet x possède d fils $\{y_1, \ldots, y_d\}$. Pour $1 \leq k \leq d$, on appelle $k^{ème}$ *restriction de* (x,x',x''), notée $Rest(k,(x,x', x''))$, le facteur (éventuellement vide) de $S(T_{y_k})$ composé des sommets de $\{z_1, \ldots, z_l\}$ présents dans T_{y_k}.

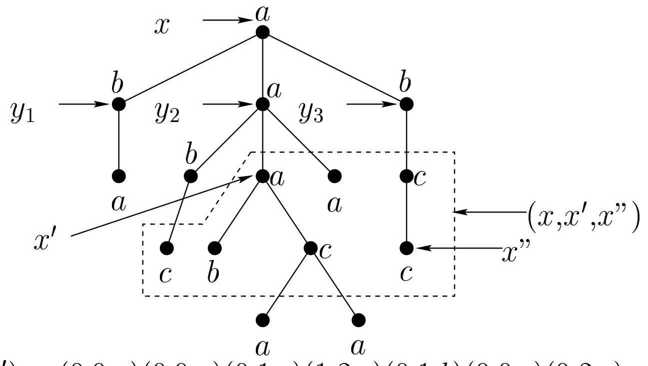
Propriété 225. Pour un mot (x,x',x'') donné, on calcule le triplet (y,y',y'') correspondant à $Rest(k,(x,x',x''))$ en temps $O(1)$.

En effet, soient $(x = T[i,j], x' = T[i',j'], x'' = T[i'',j''])$ et y le $k^{ème}$ fils de x :

- si x' est un descendant de y, alors $y' = x'$,
- si x' est un descendant d'un frère gauche (resp. droit) de y, alors $y' = T_y[i' - i, 1]$ (resp. $y' = T_y[i' - i + 1, 1]$),

et on détermine y'' de la façon similaire. Les lemmes 219 et 220 assurent que l'on peut réaliser ces calculs en temps $O(1)$.

Le lemme suivant est l'analogue du lemme 209 pour l'arborescence des suffixes et permet d'introduire la notion de lien suffixe. On peut cependant noter une différence



$(x,x',x'') = (0,0,a)(0,0,a)(0,1,c)(1,2,c)(0,1,b)(0,0,c)(0,2,c)$
$Rest(1,(x,x',x'')) = \epsilon$ (le mot vide)
$Rest(2,(x,x',x'')) = (0,0,a)(0,0,a)(1,1,c)(0,1,b)(0,0,c)$
$Rest(3,(x,x',x'')) = (1,1,c)(1,1,c)$

FIG. 9.14 – Restrictions d'un mot de T-lettres

fondamentale par rapport aux mots : le lien suffixe d'un sommet peut ne pas correspondre à un sommet de AS_i, mais à un sommet implicite.

Lemme 226. *Soient x_i un sommet de T et x_j son père ($j < i$) possédant d fils. Pour tout nœud u de AS_i autre que la racine et déjà présent dans AS_j, et tout entier k tel que $1 \leq k \leq d$, il existe un sommet implicite (v,l) dans AS_i tel que $m(v,l) = Rest(k,m(u))$. On dit alors que le couple (u,k) admet un lien suffixe valide, noté $LS(k,u)$.*

Preuve. Si u est un nœud de AS_j, il existe $j_1 < j$ tel que le préfixe de longueur $|m(u)|$ de S_{j_1} est égal à $m(u)$. x_{j_1} a donc degré d, et, si x_p est son $k^{ème}$ fils, on aura inséré S_p avant S_i. Comme $Rest(k,m(u))$ est préfixe de S_p, on en déduit qu'il existe un sommet v de AS_i tel que le mot $Rest(k,m(u))$ est préfixe de $m(v)$. □

Lemme 227. *Soit T une arborescence à n sommets. Le nombre de couples (u,k), où u est un sommet de $AS(T)$ et k un entier, qui admettent un lien suffixe valide est au plus de $2(n-1)$.*

Preuve. Comme dans le cas des mots, on peut remarquer que lors de l'insertion du suffixe S_i (de racine x_i) dans l'ADR compressé AS_{i-1}, on crée au plus deux sommets. Supposons que l'on associe à x_i l'ensemble des sommets de $AS(T)$ créés lors de l'insertion de S_i. On a donc, pour tout $i \in [n]$, au plus deux sommets associés à x_i. Soit u l'un de ces éventuels sommets et supposons x_i de degré d. Le nombre de couples (u,k) admettant un lien suffixe valide est au plus d, c'est-à-dire le nombre d'arêtes quittant x_i dans T. On a donc au plus $2d$ couples (u,k) admettant un lien suffixe tels que u est un sommet de $AS(T)$ associé à x_i, ce qui implique le résultat annoncé. □

Nous avons maintenant un des éléments indispensables au schéma "Remonter-Traverser-Descendre", à savoir les liens suffixes, et nous avons montré que l'espace additionnel nécessaire au stockage de ces liens est linéaire. Dans le paragraphe suivant, nous nous intéressons aux procédures de recherche d'un mot dans AS_i.

Les procédures de recherche lente et rapide. La procédure de recherche lente d'un mot de T-lettres (x,x',x'') dans AS_i est similaire à la procédure correspondante dans le cas des mots (la procédure **ChercherL**). Elle présente les caractéristiques suivantes :

- le point de départ de la recherche est un sommet implicite (u,l),
- le résultat retrouvé est le couple (v,z) où v est un sommet (créé si nécessaire) tel que $m(v)$ est le plus long préfixe de $m(u,l).(x,x',x'')$ reconnu dans AS_i, et z le premier sommet de (x,x',x'') n'appartenant pas à ce préfixe reconnu ($z = NULL$, si (x,x',x'') est entièrement reconnu).

On note **A-ChercherL**$((u,l),\ (x,x',x''))$ cette procédure. En remarquant que

- pour un sommet implicite (u,l) donné, le prétraitement de T permet d'obtenir en temps constant la première T-lettre de l'arête reliant u à son père n'appartenant pas à $m(u,l)$,
- on peut lire un mot (x,x',x'') de k T-lettres en temps $O(k)$ (conséquence du lemme 220),

on obtient le résultat suivant.

Propriété 228. Soit T une arborescence à n sommets. La complexité en temps d'un appel à **A-ChercherL** lors de la construction de $AS(T)$ est en $O(k \times \log(n))$, où k est la longueur du plus long préfixe de (x,x',x'') reconnu lors de cet appel.

La procédure de recherche rapide d'un T-mot (x,x',x'') dans une sous-arborescence de AS_i, sachant que ce T-mot est reconnu, suit le même principe que dans le cas des mots (procédure **ChercherR**). Toutefois, cette recherche débute en un sommet implicite (u,l) et elle renvoie comme résultat le sommet implicite minimal (u',l') tel que $m(u',l') = m(u,l).(x,x',x'')$. On la note **A-ChercherR**$((u,l),\ (x,x',x''))$. Elle s'appuie sur le résultat suivant.

Propriété 229. Soient deux mots $W = (x,x',x'')$ et $W' = (y,y',y'')$ dont on sait que l'un est préfixe de l'autre. On peut, en temps $O(1)$, déterminer

- si W est préfixe de W' et quel est le suffixe (y,z,y'') de W' non présent dans W ;
- si W' est préfixe de W et quel est le suffixe (x,z,x'') de W non présent dans W' ;
- si $W = W'$.

En effet, on déduit du point 4 du lemme 220 que pour un mot $(x = T[i,j], x' = T[i',j'], x'' = T[i'',j''])$ donné et pour $i' \leq k \leq i''$, on peut calculer le nombre de sommets de $T_x[k]$ en temps $O(1)$. La propriété découle de ce constat et des faits suivants :

- $W = W'$ si et seulement si ces deux mots comportent le même nombre de niveaux et le même nombre de sommets sur leurs derniers niveaux ;
- W (resp. W') est préfixe de W' (resp. W) si et seulement si W (resp. W') “possède moins de niveaux” que W' (resp. W) ou le même nombre de niveaux mais moins de sommets sur son dernier niveau.

Algorithme 26: A-ChercherR

Données : AS_p, un sommet implicite (u,l) de AS_p et un mot (x,x',x'')

début
- **si** $l = 0$ **alors**
  - soit (y,y',y'') l'arête quittant u (vers un sommet v) telle que $y' = x'$ et $u' := u$;
- **sinon**
  - soient (y,z,y'') l'arête reliant u à son père u', (y,y',y'') le suffixe de ce mot correspondant aux l derniers niveaux et $v := u$;
- **si** $(x,x',x'') = (y,y',y'')$ **alors**
  - **retourner** $(v,0)$;
- **sinon**
  - **si** (y,y',y'') est préfixe (x,x',x'') **alors**
    - soit z le premier sommet de (x,x',x'') n'appartenant pas à (y,y',y'');
    - **A-ChercherR**($(v,0)$, (x,z,x''));
  - **sinon**
    - soit $z = T[i,j]$ le premier sommet de $(y,y',y'' = T[i'',j''])$ n'appartenant pas à (x,x',x'');
    - **retourner** $(v,j'' - j + 1)$;

fin

Comme dans le cas des mots, on vérifie aisément la propriété suivante.

Propriété 230. Soit T une arborescence à n sommets. La complexité en temps d'un appel à **A-ChercherR** lors de la construction de $AS(T)$ est en $O(k \times \log(n))$, où k est le nombre de sommets de AS_p visités lors de cet appel.

L'algorithme final et sa complexité en temps. Nous disposons maintenant de presque tous les éléments permettant de décrire l'algorithme de construction de $AS(T)$. Cet algorithme suit le même schéma que l'algorithme de McCreight, excepté sur le point suivant : les fils x_i d'un sommet x_j n'étant pas,en général, traités immédiatement après x_j, si g_j désigne le père de f_j dans AS_j, de nouveaux sommets étant insérés dans $AS_{j+1}, \dots$, AS_{i-1}, le père de f_j dans AS_{i-1} peut être différent de g_j. On doit donc associer à tout sommet x_j de T les informations suivantes :
- le sommet f_j acceptant T_{x_j},
- le père g_j de f_j dans AS_j et l'arête (x_j,x'_j,x''_j) reliant g_j à f_j,
- le père h_j de g_j dans AS_j et l'arête (y_j,y'_j,y''_j) reliant h_j à g_j.

Comme dans le cas de l'algorithme 24, pour simplifier la présentation de notre algorithme, nous passons sous silence le traitement des cas où $g_i = r$ ou $h_i = r$, r désignant la racine.

Algorithme 27: Construction de $AS(T)$

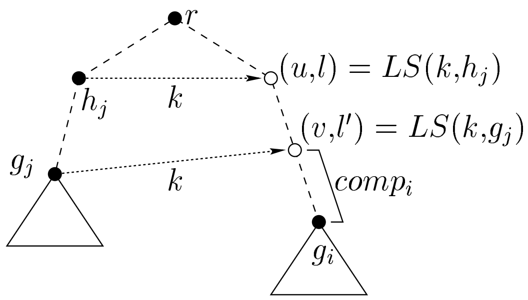
Donnée : T
début
créer AS_1 (ADR compressé à deux sommets, la racine r et la feuille f_1 associée au T-mot $S_1 = S(T_1)$);
pour chaque sommet x_i de T autre que la racine visité lors d'un parcours en largeur **faire**
soient x_j le père de x_i (x_i est le $k^{ème}$ fils de x_j) et, tels que définis précédemment les sommets f_j, g_j et h_j, et les arêtes (x_j,x'_j,x''_j) et (y_j,y'_j,y''_j);
1 $LS(k,h_j) :=$ **Minimiser**$(LS(k,h_j))$;
$(u,l) := LS(k,h_j)$;
2 $(v,l') :=$**A-ChercherR**$((u,l),\ (y_j,y'_j,y''_j))$;
3 $LS(k,g_j) := (v,l')$;
4 $(w,x) :=$**A-ChercherL**$((v,l'),\ (x_j,x'_j,x''_j))$;
si $x = NULL$ **alors** $f_i := w$;
sinon créer f_i comme fils de w relié à celui-ci par une arête (x_j,x,x''_j);
5 $LS(k,g_j) :=$ **Minimiser**$(LS(k,g_j))$;
fin

Pour terminer, nous cherchons à déterminer la complexité en temps de cet algorithme, qui nécessite de calculer la complexité des appels à **A-ChercherL**, la complexité des appels à **Minimiser** et la complexité des appels à **A-ChercherR**.

Dans les figures suivantes, un sommet blanc correspond à un sommet éventuellement implicite, les arêtes en pointillés à un chemin dans l'arborescence et les flèches en pointillés aux liens suffixes.

Lemme 231. *Soit T une arborescence à n sommets. La complexité en temps des appels à* **A-ChercherL** *lors de la construction de $AS(T)$ est en $O(n \times \log(n))$.*

Preuve. La complexité d'un appel à **A-ChercherL**$((u,l),\ (x,x',x''))$ est déterminée par la longueur du plus long préfixe de (x,x',x'') reconnu à partir de (u,l) (propriété 228). Notons $comp_i$ la longueur de ce préfixe lors de l'appel à **A-ChercherL** effectué lors de l'insertion de S_i dans AS_{i-1} (étape 4).



Le sommet x_i désignant le $k^{ème}$ fils de x_j, on a clairement

$$comp_i \leq |m(g_i)| - |Rest(k,m(g_j))| + 1.$$

On en déduit que

$$\sum_{i=2}^{n} comp_i \leq (n-1) + \sum_{i=2}^{n} |m(g_i)| - \sum_{j=1}^{n} \left(\sum_{k=1}^{deg(x_j)} |Rest(k,m(g_j))| \right)$$

où $deg(x_j) =$ est le nombre de fils de x_j. En remarquant que

$$|m(g_j)| = 1 + \sum_{k=1}^{deg(x_j)} |Rest(k,m(g_j))|$$

on en déduit que

$$\sum_{i=2}^{n} comp_i \leq (n-1) + \sum_{i=2}^{n} |m(g_i)| - \sum_{j=1}^{n} (|m(g_j)| - 1)$$

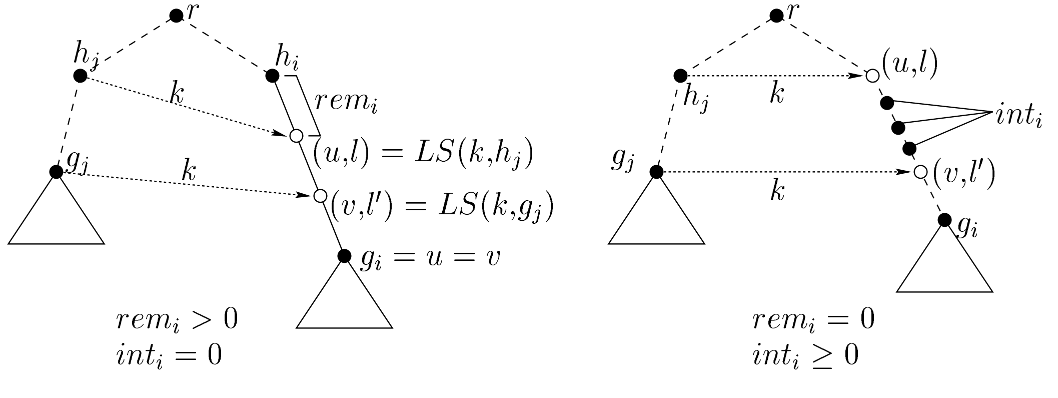
$$\Longrightarrow \sum_{i=2}^{n} comp_i \leq (2n-1) - |m(g_1)| < 2n.$$

Le facteur $\log(n)$ correspond pour sa part au temps de branchement (il y a au plus n arêtes quittant un sommet de AS_i). □

Lemme 232. *Soit T une arborescence à n sommets. La complexité en temps des appels à* **Minimiser** *lors de la construction de $AS(T)$ est en $O(n)$.*

Preuve. Le second appel à **Minimiser** (étape 5) a pour seul but de prendre en compte le fait que lors de l'appel à **A-ChercherL** (étape 3), un sommet a pu être créé, ce qui fait que le lien suffixe $LS(k,g_j)$ peut ne plus être minimal, alors qu'il l'était lors de sa création (étape 4). Comme on a ajouté au plus un sommet, cet appel s'effectue clairement en temps $O(1)$. Pour borner la complexité du premier appel à **Minimiser** (étape 1), on peut remarquer que, pour un sommet implicite (u,l) et un descendant v de u, le nombre total de sommets visités lors des minimalisations de (u,l) est au plus $|m(v)| - |m(u,l)|$. En remarquant que lors de la création du lien suffixe $LS(k,g_j) = (w,l')$, g_i est un descendant de w, on en déduit que la complexité totale des appels à **Minimiser** correspondant à l'étape 1 est au plus $\sum_{i=2}^{n} comp_i$, ce qui, compte tenu de la preuve du lemme précédent prouve le résultat annoncé. □

Il reste à examiner la complexité des appels à **A-ChercherR**. On rappelle que la complexité de l'appel à **A-ChercherR** lors de l'insertion de S_i dans AS_{i-1} est déterminée par le nombre int_i de sommets de AS_{i-1} visités lors de cet appel (propriété 230). On distingue deux cas, illustrés par la figure suivante



et on introduit la notation suivante (on rappelle qu'un sommet est considéré comme faisant partie de ses ancêtres) : si h_i est un ancêtre de u et $h_i \neq u$ (figure de gauche), $rem_i = |m(u,l)| - |m(h_i)| + 1$, sinon, $rem_i = 0$.

Lemme 233. *Soit T une arborescence à n sommets. La complexité en temps des appels à* **A-ChercherR** *lors de la construction de $AS(T)$ est en $O((n+k) \times \log(n))$, où $k = \sum_{i=2}^{n} rem_i$.*

Preuve. Comme dans le cadre des mots, pour $i = 2, \ldots, n$, on note $res_i = |m(f_i)| - |m(u,l)| + 1$. En remarquant que

$$\sum_{x_i \text{ fils de } x_j} res_i \leq res_j + rem_j - int_j,$$

on en déduit que

$$\sum_{j=2}^{n} int_j \leq \sum_{j=2}^{n} (res_j + rem_j) - \sum_{j=2}^{n} \left(\sum_{x_i \text{ fils de } x_j} res_i \right),$$

ce qui, si on note p le nombre de sommets présents sur les niveaux $T[1]$ (la racine de T) et $T[2]$ (les fils de cette racine), est équivalent à

$$\sum_{j=2}^{n} int_j \leq \sum_{j=2}^{n} res_j - \sum_{j=p+1}^{n} res_j + \sum_{j=2}^{n} rem_j.$$

En notant que $\sum_{j=1}^{p} res_j \leq n$, on obtient alors le résultat annoncé. □

Nous n'avons cependant pas été en mesure de montrer que $\sum_{i=2}^{n} rem_i$ est en $O(n)$, ce qui induirait une construction de $AS(T)$ en temps $0(n \times \log(n))$. Les nombreux tests que nous avons menés, combinés à de vains efforts pour construire un contre-exemple, nous amènent toutefois à formuler la conjecture suivante.

Conjecture 234. $\sum_{i=2}^{n} rem_i$ *est en $O(n)$.*

Le tableau suivant présente les résultats en moyenne de nos tests, effectués sur des arborescences quelconques. Dans ce tableau, N désigne le nombre de sommets de $AS(T)$, ACL la complexité des appels à **A-ChercherL**, ACR la complexité des appels à **A-ChercherR**, et Min la complexité des appels à **Minimiser**.

n	$\|\Sigma\|$	N	ACL	ACR	Min
100	1	44	102	22	1
100	3	54	100	13	1
100	5	59	98	10	0
100	7	62	95	7	0
1000	1	368	1018	186	3
1000	3	439	1017	122	1
1000	5	475	1017	109	1
1000	7	480	1013	88	0
10000	1	3222	10135	1613	24
10000	3	3816	10134	1093	7
10000	5	4063	10132	965	4
10000	7	4148	10131	833	2

TAB. 9.1 – Construction de $AS(T)$: résultats expérimentaux moyens

Applications à la recherche de motifs. Comme nous l'avons évoqué en début de chapitre, notre but en définissant une telle structure de données est de traiter le problème de la recherche d'un motif M de m sommets dans une arborescence statique T à n sommets, en utilisant une structure de données efficace en espace.

Dans le cas général, tel que nous l'avons défini dans le chapitre précédent, il ne semble pas que l'utilisation de $AS(T)$ permette d'améliorer la borne quadratique $O(n \times m)$ de l'algorithme naïf. En effet, soient M un motif, $S(M)$ le mot de T-lettres correspondant et u un sommet de $AS(T)$ (on rappelle qu'on note $T(u)$ la sous-arborescence maximale de T telle que $S(T(u)) = m(u)$). On dit que u est un sommet minimal acceptant M si et seulement si

- il existe une occurrence de M en la racine de $T(u)$,
- pour tout ancêtre $v \neq u$ de u, il n'existe pas d'occurrence de M en la racine de $T(v)$.

On en déduit que si w est un sommet final de $AS(T)$ descendant de u acceptant S_i, alors on a une occurrence de M en x_i dans T. On peut alors construire des couples (M,T) tels que le nombre de sommets minimaux acceptant M de $AS(T)$ est de l'ordre de $O(n)$ et que deux chemins quelconques dans $AS(T)$ allant de la racine r vers deux de ces sommets ont un plus long préfixe commun de taille constante. Dans ce cas, en utilisant seulement $AS(T)$, repérer tous les sommets minimaux acceptant M nécessite un temps $O(n \times m)$.

On peut cependant, dans le cas de la recherche de motifs dans des termes, pour le problème RMCA, utiliser $AS(T)$ pour accélerer en pratique l'algorithme naïf, et ce en reprenant le principe des algorithmes présentés dans le chapitre 8. Les tests que nous avons effectués dans ce cas indiquent toutefois que les algorithmes du chapitre 8, qui se caractérisent par leur simplicité, sont plus efficaces en pratique.

On peut aussi utiliser $AS(T)$ dans le cadre de la proposition 201, pour résoudre efficacement des cas restreints de recherche de motifs dans une arborescence. Dans le cas des termes, on déduit par exemple immédiatement de la proposition 201 le résultat suivant (on rappelle que le texte T est étiqueté sur Σ et le motif M sur $\Sigma \cup \{v\}$).

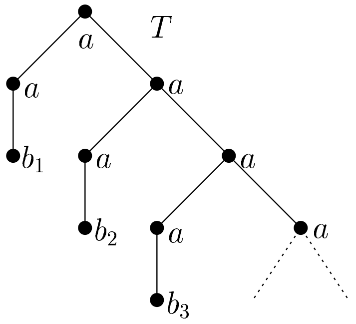


FIG. 9.15 – Un couple (T,M) maximisant la recherche des sommets minimaux acceptant M

Proposition 235. *Soit T un terme à n sommets et M un motif à m sommets tel que tous les sommets de son dernier niveau sont étiquetés v et aucun sommet étiqueté v n'est situé ailleurs que sur ce dernier niveau. Après construction de $AS(T)$, on peut détecter les k occurrences de M dans T en temps $O(m \times \log(n) + k)$.*

Dans le cas des arborescences quelconques, le motif M est dit *complet* si et seulement si toutes ses feuilles sont situées sur son dernier niveau (cette notion est une extension de la notion d'arborescence k-aire complète). Une occurrence de M dans T est dite *complète* si tout nœud de M de degré $d > 0$ correspond à un nœud de T de même degré d. Dans la figure suivante, la seule occurrence complète de M dans T est matérialisée par les arêtes en pointillés.

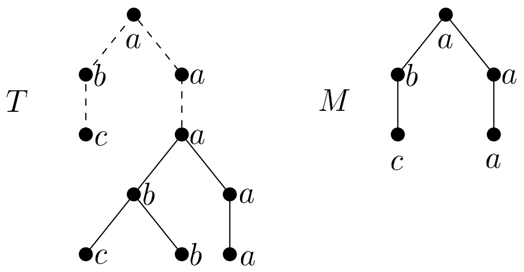


FIG. 9.16 – Illustration de la notion d'occurrence complète

On déduit de la proposition 201 et de la définition de $AS(T)$ le résultat suivant, qui généralise le résultat de Shibuya sur les arborescences k-aires complètes [141].

Proposition 236. *Soit T un texte à n sommets et M un motif complet à m sommets. Après construction de $AS(T)$, on peut détecter les k occurrences complètes de M dans T en temps $O(m \times \log(n) + k)$.*

9.4 Conclusion

Dans ce chapitre, nous avons proposé une approche du problème de la recherche de motifs dans une arborescence statique basée sur l'utilisation d'une structure de données indexant les suffixes du texte statique, l'arborescence des suffixes d'une arborescence. Si la notion de suffixe que nous avons choisie, consistant à voir une arborescence comme un mot généralisé et à regrouper les sommets de T par niveau, permet de définir une structure de données efficace en espace, elle ne permet cependant de traiter efficacement (en termes de complexité dans le pire des cas) que des instances restreintes du problème de la recherche de motifs dans une arborescence. De plus, les techniques nécessaires à la construction de $AS(T)$ sont lourdes, et d'un point de vue pratique, les algorithmes proposés dans le chapitre 8 semblent plus intéressants, du fait de leur simplicité et de leurs performances pratiques.

Cependant, nous pensons que d'un point de vue théorique, la structure de données que nous avons introduite est intéressante en elle-même, car elle correspond à une généralisation naturelle de l'arborescence des suffixes d'un mot. Il nous a ainsi paru naturel de définir un algorithme de construction de cette structure s'appuyant sur un algorithme de construction de l'arborescence des suffixes d'un mot, et l'algorithme de McCreight nous a semblé être le meilleur candidat. Toutefois, nous n'avons pas pu prouver une borne optimale pour notre algorithme et le problème de la construction de l'arborescence des suffixes d'une arborescence en temps quasi-linéaire reste à ce jour ouvert. Cependant, comme nous l'avons mentionné en début de chapitre, Shibuya a récemment traité le cas simplifié des arborescences régulières (tous les nœuds ont le même degré) et, se basant sur l'algorithme de Farach [60], a proposé un algorithme quasi-linéaire pour l'arborescence des suffixes d'une arborescence régulière. Il serait intéressant d'essayer d'étendre sa technique au cas des arborescences quelconques.