1 Outils

Ce chapitre présente le cadre algorithmique et combinatoire dans lequel sont développés les chapitres suivants. Il précise pour commencer les notions et notations utilisées pour travailler sur les mots, les langages et les automates. La suite est principalement consacrée à l'introduction des structures de données retenues pour la réalisation d'automates et à la présentation de résultats combinatoires. Cette organisation s'appuie sur la constatation que les algorithmes efficaces de traitement du texte reposent sur l'un ou l'autre des ces aspects.

La section 1.2 fournit quelques propriétés combinatoires des mots qui reviennent dans bon nombre de preuves de validité d'algorithmes ou d'évaluations de leurs performances. Il s'agit principalement de résultats sur les périodicités qui apparaissent dans certains mots.

Le formalisme de description des algorithmes est présenté dans la section 1.3 qui est surtout centrée sur le type d'algorithmes décrits dans l'ouvrage et qui introduit quelques objets standard concernant la gestion de files et d'automates.

La section 1.4 détaille plusieurs procédés pour implanter des automates en mémoire, procédés qui contribuent notamment aux résultats des chapitres 2, 5 et 6.

Les premiers algorithmes de localisation de mots sont présentés dans la section 1.5. Les techniques de fenêtre glissante, d'automate de recherche et d'utilisation de mots machine qui y sont décrites sont reprises et améliorées dans les chapitres 2, 3 et 8, en particulier.

La section 1.6 est le joyau algorithmique du chapitre. Elle présente deux méthodes algorithmiques fondamentales utilisées pour le traitement du texte. Elles servent à calculer la table des bords et la table des préfixes d'un mot qui constituent deux tables essentielles car elles condensent une partie des propriétés combinatoires du mot. Leur utilisation ou adaptation est considérée dans les chapitres 2 et 3, mais revient ponctuellement dans d'autres chapitres.

Enfin, on notera que l'intuition s'appuie quelquefois sur des figures dont le style est introduit dans le chapitre et conservé ensuite.

1.1 Mots et automates

On introduit dans cette section les notations sur les mots, les langages et les automates.

Alphabet et mots

Un ***alphabet*** est un ensemble fini non vide dont les éléments sont appelés des ***lettres***. Un ***mot*** sur un alphabet A est une suite finie d'éléments de A. La suite de zéro lettre est appelée le ***mot vide*** et notée ε. Pour simplifier, les délimiteurs et les séparateurs utilisés habituellement dans les notations des suites sont supprimés et l'on écrit un mot comme la simple juxtaposition des lettres qui le composent. Ainsi, ε, `a`, `b` et `baba` sont-ils des mots sur tout alphabet qui contient les deux lettres `a` et `b`. L'ensemble de tous les mots sur l'alphabet A est noté A^*, et l'ensemble de tous les mots sur l'alphabet A excepté le mot vide ε est noté A^+.

La ***longueur*** d'un mot x est définie comme la longueur de la suite associée au mot x et est notée $|x|$. On note $x[i]$, pour $i = 0, 1, \ldots, |x| - 1$, la lettre à l'indice i de x en convenant de commencer la numérotation des indices à partir de 0. Lorsque $x \neq \varepsilon$, on dit plus spécifiquement de chaque indice $i = 0, 1, \ldots, |x| - 1$ qu'il est une ***position*** sur x. Il s'ensuit que la j-ième lettre de x est la lettre à la position $j - 1$ sur x et que :

$$x = x[0]x[1] \ldots x[|x| - 1] \ .$$

D'où aussi une définition élémentaire de l'identité de deux mots quelconques x et y :

$$x = y$$

si et seulement si

$$|x| = |y| \text{ et } x[i] = y[i] \text{ pour } i = 0, 1, \ldots, |x| - 1 \ .$$

L'ensemble des lettres sur lequel est formé le mot x est noté $alph(x)$. Par exemple, si $x = \mathtt{abaaab}$, on a $|x| = 6$ et $alph(x) = \{\mathtt{a}, \mathtt{b}\}$.

Le ***produit*** – on dit aussi la ***concaténation*** – de deux mots x et y est le mot composé des lettres de x puis de celles de y dans cet ordre. On le note xy ou encore $x \cdot y$ pour faire apparaitre une décomposition du mot résultant. L'élément neutre pour le produit est ε. Pour tout mot x et tout naturel n, on définit la n-ième ***puissance*** du mot x, notée x^n, par $x^0 = \varepsilon$ et $x^k = x^{k-1}x$ pour $k = 1, 2, \ldots, n$. Sont notés respectivement zy^{-1} et $x^{-1}z$ les mots x et y lorsque $z = xy$. Le ***renversé*** – ou ***image miroir*** – du mot x est le mot $x^\sim$ défini par :

$$x^\sim = x[|x| - 1]x[|x| - 2] \ldots x[0] \ .$$

Un mot x est un ***facteur*** d'un mot y s'il existe deux mots u et v tels que $y = uxv$. Lorsque $u = \varepsilon$, x est un ***préfixe*** de y ; et lorsque

b	a	b	a	a	b	a	b	a

Figure 1.1 Une occurrence du mot aba dans le mot babaababa à la position (gauche) 1.

$v = \varepsilon$, x est un ***suffixe*** de y. Le mot x est un ***sous-mot*** de y s'il existe $|x|+1$ mots $w_0, w_1, \ldots, w_{|x|}$ tels que $y = w_0x[0]w_1x[1]\ldots x[|x|-1]w_{|x|}$; de manière moins formelle, x est un mot obtenu de y en lui supprimant $|y|-|x|$ lettres. Un facteur ou un sous-mot x d'un mot y est qualifié de ***propre*** si $x \neq y$. On note respectivement $x \preceq_{\mathit{fact}} y$, $x \prec_{\mathit{fact}} y$, $x \preceq_{\mathit{préf}} y$, $x \prec_{\mathit{préf}} y$, $x \preceq_{\mathit{suff}} y$, $x \prec_{\mathit{suff}} y$, $x \preceq_{\mathit{smot}} y$ et $x \prec_{\mathit{smot}} y$ lorsque x est un facteur, un facteur propre, un préfixe, un préfixe propre, un suffixe, un suffixe propre, un sous-mot et un sous-mot propre de y. On peut vérifier que $\preceq_{\mathit{fact}}$, $\preceq_{\mathit{préf}}$, $\preceq_{\mathit{suff}}$ y et $\preceq_{\mathit{smot}}$ sont des relations d'ordre sur A^*.

L'***ordre lexicographique***, noté $\leq$, est un ordre sur les mots induit par un ordre sur les lettres noté de la même façon. Il est défini comme suit. Pour $x, y \in A^*$, $x \leq y$ si et seulement si, soit $x \preceq_{\mathit{préf}} y$, soit x et y se décomposent sous la forme $x = uav$ et $y = ubw$ avec $u, v, w \in A^*$, $a, b \in A$ et $a < b$. Ainsi, ababb $<$ abba $<$ abbaab en supposant a $<$ b.

Étant donnés un mot non vide x et un mot y, on dit qu'il y a une ***occurrence*** de x dans y, ou, plus simplement, que x ***apparait*** dans y, lorsque x est un facteur de y. Toute occurrence, ou toute apparition, de x peut être caractérisée par une position sur y. Ainsi dit-on qu'une occurrence de x ***débute*** à la ***position (gauche)*** i sur y lorsque $y[i\,..\,i+|x|-1] = x$ (voir figure 1.1). Il est quelquefois plus agréable de considérer la ***position droite*** $i+|x|-1$ en laquelle cette même occurrence ***se termine***. Par exemple, les positions gauches et droites auxquelles apparait le mot $x =$ aba dans le mot $y =$ babaababa sont :

i	0	1	2	3	4	5	6	7	8
$y[i]$	b	a	b	a	a	b	a	b	a
positions gauches		1			4		6		
positions droites				3			6		8

La ***position de la première occurrence*** $\mathit{pos}(x)$ de x dans y est la position minimale en laquelle débute l'occurrence de x sur yA^*. Avec les notations sur les langages rappelées ci-après, on a :

$$\mathit{pos}(x) = \min\{|u| : \{ux\}A^* \cap \{y\}A^* \neq \emptyset\} \ .$$

Les notations avec des crochets pour les lettres des mots sont étendues aux facteurs. On définit le facteur $x[i\,..\,j]$ du mot x par :

$$x[i\,..\,j] = x[i]x[i+1]\ldots x[j]$$

pour tous entiers i et j satisfaisant $0 \leq i \leq |x|$, $-1 \leq j \leq |x|-1$ et $i \leq j+1$. Lorsque $i = j+1$, le mot $x[i\,..\,j]$ est le mot vide.

Langages

Tout sous-ensemble de A^* est un ***langage*** sur l'alphabet A. Le produit défini sur les mots est étendu aux langages en posant

$$XY = X \cdot Y = \{xy : (x, y) \in X \times Y\}$$

pour tous langages X et Y. On étend de même la notion de puissance en posant $X^0 = \{\varepsilon\}$ et $X^k = X^{k-1}X$ pour $k \geq 1$. L'***étoile*** de X est le langage :

$$X^* = \bigcup_{n \geq 0} X^n \quad .$$

Est noté X^+ le langage défini par :

$$X^+ = \bigcup_{n \geq 1} X^n \quad .$$

Remarquons que ces deux dernières notations sont compatibles avec les notations A^* et A^+. À la fois par abus et pour ne pas surcharger les notations, un langage réduit à un seul mot peut être nommé par le mot lui-même si cela n'entraine aucune confusion. Par exemple, l'expression abusive $A^*\mathtt{abaaab}$ désigne le langage des mots qui ont comme suffixe le mot $\mathtt{abaaab}$, sous l'hypothèse $\{\mathtt{a}, \mathtt{b}\} \subseteq A$.

La notion de longueur est étendue aux langages en posant :

$$|X| = \sum_{x \in X} |x| \quad .$$

De la même façon, on définit $alph(X)$ par

$$alph(X) = \bigcup_{x \in X} alph(x)$$

et $X^\sim$ par

$$X^\sim = \{x^\sim : x \in X\} \quad .$$

Les ensembles des facteurs, des préfixes, des suffixes et des sous-mots des mots d'un langage X sont des langages particuliers souvent considérés dans la suite de l'ouvrage ; ils sont notés respectivement $Fact(X)$, $Préf(X)$, $Suff(X)$ et $SMot(X)$.

Le ***contexte droit*** d'un mot y relativement à un langage X est le langage :

$$y^{-1}X = \{y^{-1}x : x \in X\} \quad .$$

La relation d'équivalence définie par l'identité des contextes droits est notée $\equiv_X$, ou simplement[1] $\equiv$. Ainsi

$$y \equiv z \text{ si et seulement si } y^{-1}X = z^{-1}X$$

1. Comme dans toute la suite de l'ouvrage, les notations ne sont indicées par l'objet auquel elles réfèrent qu'en cas d'ambigüité éventuelle.

pour $y, z \in A^*$. Par exemple, lorsque $A = \{\mathtt{a}, \mathtt{b}\}$ et $X = A^*\{\mathtt{aba}\}$, la relation $\equiv$ admet quatre classes d'équivalence, à savoir $\{\varepsilon, \mathtt{b}\} \cup A^*\{\mathtt{bb}\}$, $\{\mathtt{a}\} \cup A^*\{\mathtt{aa}, \mathtt{bba}\}$, $A^*\{\mathtt{ab}\}$ et $A^*\{\mathtt{aba}\}$. Pour tout langage X, la relation $\equiv$ est une relation d'équivalence qui est compatible avec la concaténation. C'est la ***congruence syntaxique droite*** associée à X.

Expressions et langages rationnels

Les ***expressions rationnelles*** sur un alphabet A et les langages qu'elles décrivent, les ***langages rationnels***, sont définis récursivement comme suit :

- `0` et `1` sont des expressions rationnelles qui décrivent respectivement $\emptyset$ (l'ensemble vide) et $\{\varepsilon\}$;
- pour toute lettre $a \in A$, $\boldsymbol{a}$ est une expression rationnelle qui décrit le singleton $\{a\}$;
- si $\boldsymbol{x}$ et $\boldsymbol{y}$ sont des expressions rationnelles décrivant respectivement les langages rationnels X et Y, alors $(\boldsymbol{x}+\boldsymbol{y})$, $(\boldsymbol{x}\cdot\boldsymbol{y})$ et $(\boldsymbol{x})^*$ sont des expressions rationnelles qui décrivent respectivement les langages rationnels $X \cup Y$, $X \cdot Y$ et X^*.

L'ordre de priorité des opérations sur les expressions rationnelles est `*`, $\cdot$, puis `+`. D'où d'éventuelles simplifications d'écriture qui permettent d'omettre le symbole $\cdot$ et certaines paires de parenthèses. Le langage décrit par une expression rationnelle $\boldsymbol{x}$ est noté $Lang(\boldsymbol{x})$.

Automates

Un ***automate*** M sur l'alphabet A est composé d'un ensemble fini Q d'***états***, d'un état[2] ***initial*** q_0, d'un ensemble $T \subseteq Q$ d'états ***terminaux*** et d'un ensemble $F \subseteq Q \times A \times Q$ de ***flèches*** – ou ***transitions***. On note l'automate M par le quadruplet :

(Q, q_0, T, F) .

On dit d'une flèche (p, a, q) qu'elle sort de l'état p et qu'elle entre dans l'état q ; l'état p est la ***source*** de la flèche, la lettre a son ***étiquette***, l'état q sa ***cible***. Le nombre de flèches sortant d'un état donné est appelé le ***degré (sortant)*** de l'état. Le ***degré entrant*** d'un état est défini de façon duale. Par analogie avec les graphes, l'état q est un ***successeur*** par la lettre a de l'état p lorsque $(p, a, q) \in F$; dans le même cas, on dit du couple (a, q) qu'il est un ***successeur étiqueté*** de l'état p.

2. La définition standard des automates considère un ensemble d'états initiaux au lieu d'un seul état initial comme c'est le cas dans tout l'ouvrage. On laisse le soin au lecteur de se convaincre qu'il est possible de faire correspondre à tout automate défini de façon standard un automate à un seul état initial qui reconnait le même langage.

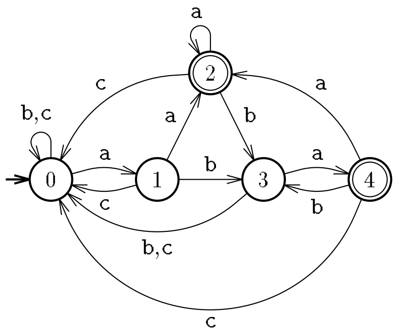


Figure 1.2 Représentation d'un automate sur l'alphabet $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Les états de l'automate sont numérotés de 0 à 4, son état initial est 0 et ses états terminaux sont 2 et 4. L'automate possède $3 \times 5 = 15$ flèches. Le langage qu'il reconnait est celui décrit par l'expression rationnelle `(a+b+c)*(aa+aba)`, c'est-à-dire l'ensemble des mots sur l'alphabet des trois lettres a, b et c se terminant par aa ou aba.

Un ***chemin*** de longueur n dans l'automate $M = (Q, q_0, T, F)$ est une suite de n flèches consécutives

$$\langle (p_0, a_0, p'_0), (p_1, a_1, p'_1), \ldots, (p_{n-1}, a_{n-1}, p'_{n-1}) \rangle \ ,$$

c'est-à-dire telles que

$$p'_k = p_{k+1}$$

pour $k = 0, 1, \ldots, n-2$. L'***étiquette*** du chemin est le mot $a_0 a_1 \ldots a_{n-1}$, son ***origine*** l'état p_0, sa ***fin*** l'état p'_{n-1}. Par convention, il existe pour chaque état p un chemin de longueur nulle d'origine et de fin p ; l'étiquette d'un tel chemin est ε, le mot vide. Un chemin dans l'automate M est ***réussi*** si son origine est l'état initial q_0 et si sa fin est dans T. Un mot est ***reconnu*** – ou ***accepté*** – par l'automate s'il est l'étiquette d'un chemin réussi. Le langage formé des mots reconnus par l'automate M est noté $Lang(M)$.

Souvent, bien plus que sa donnée formelle, un diagramme plan permet d'appréhender le fonctionnement d'un automate. On représente les états par des cercles et les flèches par des arcs dirigés de la source vers la cible et étiquetés par la lettre correspondante. Lorsque plusieurs flèches ont même source et même cible, on confond les arcs et l'on étiquette l'arc résultant par une énumération des lettres. L'état initial est distingué par une courte flèche entrante et les états terminaux sont doublement cerclés. Un exemple est montré figure 1.2.

Un état p d'un automate $M = (Q, q_0, T, F)$ est ***accessible*** s'il existe un chemin dans M d'origine q_0 et de fin p. Un état p est ***coaccessible*** s'il existe un chemin dans M d'origine p et de fin dans T.

Un automate $M = (Q, q_0, T, F)$ est ***déterministe*** si pour tout couple $(p, a) \in Q \times A$ il existe au plus un état $q \in Q$ tel que $(p, a, q) \in F$. Dans un tel cas, il est naturel de considérer la ***fonction de transition***

$\delta : Q \times A \to Q$

de l'automate définie pour toute flèche $(p, a, q) \in F$ par

$\delta(p, a) = q$

et non définie ailleurs. La fonction δ s'étend facilement aux mots. Il suffit de considérer son prolongement $\bar{\delta} : Q \times A^* \to Q$ défini récursivement par $\bar{\delta}(p, \varepsilon) = p$ et $\bar{\delta}(p, wa) = \delta(\bar{\delta}(p, w), a)$ pour $p \in Q$, $w \in A^*$ et $a \in A$. Il s'ensuit que le mot w est reconnu par l'automate M si $\bar{\delta}(q_0, w) \in T$. Généralement, la fonction δ et son prolongement $\bar{\delta}$ sont notés de la même manière.

L'automate $M = (Q, q_0, T, F)$ est ***complet*** lorsque pour tout couple $(p, a) \in Q \times A$ il existe au moins un état $q \in Q$ tel que $(p, a, q) \in F$.

Proposition 1.1
Pour tout automate, il existe un automate déterministe et complet qui reconnait le même langage. ∎

Rendre complet un automate n'est pas difficile : il suffit d'ajouter à l'automate un état ***rebut***, puis de le rendre cible de toutes les transitions non définies. Il est un peu plus délicat en revanche de ***déterminiser*** un automate, c'est-à-dire obtenir d'un automate $M = (Q, q_0, T, F)$ un automate déterministe reconnaissant le même langage. On peut utiliser la méthode dite de ***construction par sous-ensembles*** : soit M' l'automate dont les états sont les parties de Q, l'état initial le singleton $\{q_0\}$, les états terminaux les parties de Q dont l'intersection avec T est non vide, et les flèches les triplets (U, a, V) où V est l'ensemble des successeurs par la lettre a des états p appartenant à U ; alors M' est un automate déterministe qui reconnait le même langage que M. Dans la pratique, on ne construit pas l'automate M' en entier, mais seulement sa partie accessible en commençant à partir de l'état initial $\{q_0\}$.

Un langage X est ***reconnaissable*** s'il existe un automate M tel que $X = Lang(M)$. Suit l'énoncé d'un théorème fondamental de la théorie des automates qui établit le lien entre langages reconnaissables et langages rationnels sur un alphabet donné.

Théorème 1.2 (théorème de Kleene)
Un langage est reconnaissable si et seulement si il est rationnel. ∎

Si X est un langage reconnaissable, l'***automate minimal*** de X, noté $\mathcal{M}(X)$, est déterminé par la congruence syntaxique droite associée à X. C'est l'automate dont l'ensemble des états est $\{w^{-1}X : w \in A^*\}$, l'état initial X, l'ensemble des états terminaux $\{w^{-1}X : w \in X\}$, et l'ensemble des flèches $\{(w^{-1}X, a, (wa)^{-1}X) : (w, a) \in A^* \times A\}$.

Proposition 1.3
L'automate minimal $\mathcal{M}(X)$ d'un langage X est l'automate ayant le moins d'états parmi les automates déterministes et complets qui reconnaissent le langage X. L'automate $\mathcal{M}(X)$ est l'image homomorphe de tout automate reconnaissant X. ■

On dit souvent d'un automate qu'il est minimal alors qu'il n'est pas complet. En fait, et par abus, cet automate est bien minimal si l'on prend soin de lui rajouter un état rebut.

Chacun des états d'un automate, ou parfois même chacune de ses flèches, peut être agrémenté d'une ***sortie***. Il s'agit d'une valeur ou d'un ensemble de valeurs associé à l'état ou à la flèche en question.

1.2 Un peu de combinatoire

On considère la notion de périodicité sur les mots pour laquelle on donne les propriétés de base. On commence par présenter deux familles de mots qui ont des propriétés combinatoires intéressantes au regard des questions de périodicités et de répétitions examinées dans plusieurs chapitres.

Quelques mots en particulier

Les ***nombres de Fibonacci*** sont définis par la récurrence :

$$\begin{aligned} F_0 &= 0 \ , \\ F_1 &= 1 \ , \\ F_n &= F_{n-1} + F_{n-2} \qquad \text{pour } n \geq 2 \ . \end{aligned}$$

Ces nombres fameux vérifient des propriétés toutes plus remarquables les unes que les autres. Parmi celles-ci, on en signale juste deux :

- pour tout naturel $n \geq 2$, $\mathrm{pgcd}(F_n, F_{n-1}) = 1$;
- pour tout naturel n, F_n est l'entier le plus proche de $\Phi^n/\sqrt{5}$, où $\Phi = \frac{1}{2}(1+\sqrt{5}) = 1{,}61803\ldots$ est le ***nombre d'or***.

Les ***mots de Fibonacci*** sont définis sur l'alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ par la récurrence suivante :

$$\begin{aligned} f_0 &= \varepsilon \ , \\ f_1 &= \mathtt{b} \ , \\ f_2 &= \mathtt{a} \ , \\ f_n &= f_{n-1} f_{n-2} \qquad \text{pour } n \geq 3 \ . \end{aligned}$$

Notons que la suite des longueurs des mots est exactement la suite des nombres de Fibonacci, c'est-à-dire que l'on a $F_n = |f_n|$. Voici en exemple les dix premiers nombres et mots de Fibonacci :

n	F_n	f_n
0	0	ε
1	1	`b`
2	1	`a`
3	2	`ab`
4	3	`aba`
5	5	`abaab`
6	8	`abaababa`
7	13	`abaababaabaab`
8	21	`abaababaabaababaababa`
9	34	`abaababaabaababaababaabaababaabaab`

L'intérêt des mots de Fibonacci est qu'ils satisfont des propriétés combinatoires intéressantes et contiennent un grand nombre de répétitions.

Les mots de de Bruijn considérés ici sont définis sur l'alphabet $A = \{\mathtt{a}, \mathtt{b}\}$ et sont paramétrés par un naturel non nul. Un mot non vide $x \in A^+$ est un ***mot de de Bruijn*** d'ordre k si chaque mot sur A de longueur k apparait une fois et une seule dans x. Un premier exemple : `ab` et `ba` sont les deux seuls mots de de Bruijn d'ordre 1. Un second exemple : le mot `aaababbbaa` est un mot de de Bruijn d'ordre 3 puisque ses facteurs de longueur 3 sont les huit mots de A^3, à savoir `aaa`, `aab`, `aba`, `abb`, `baa`, `bab`, `bba` et `bbb`.

L'existence d'un mot de de Bruijn d'ordre $k \geq 2$ se vérifie à l'aide de l'***automate*** dont

- les états sont les mots du langage A^{k-1} ;
- les flèches sont de la forme (av, b, vb) avec $a, b \in A$ et $v \in A^{k-2}$,

l'état initial et les états terminaux n'étant pas précisés (une illustration est montrée figure 1.3). On constate qu'exactement deux flèches sortent de chacun des états, l'une étiquetée par `a`, l'autre par `b` ; et qu'exactement deux flèches entrent dans chacun des états, toutes deux étiquetées par la même lettre. Le graphe associé à l'automate vérifie donc la condition d'Euler : le degré sortant et le degré entrant de chaque état sont identiques. Il s'en déduit qu'il existe un circuit eulérien partant de chaque

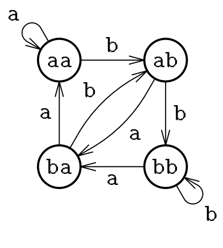


Figure 1.3 L'automate de de Bruijn à l'ordre 3 sur l'alphabet $\{\mathtt{a}, \mathtt{b}\}$. L'état initial de l'automate n'est pas précisé.

état. Maintenant, soit

$\langle (u_0, a_0, u_1), (u_1, a_1, u_2), \ldots, (u_{n-1}, a_{n-1}, u_0) \rangle$

l'un des chemins correspondant. Alors le mot $u_0 a_0 a_1 \ldots a_{n-1}$ est un mot de de Bruijn d'ordre k, chacune des flèches du chemin étant identifiée à un facteur de longueur k. Il s'ensuit du même coup qu'un mot de de Bruijn d'ordre k a pour longueur $2^k + k - 1$ (soit $n = 2^k$ avec la notation précédente). On vérifie également que le nombre de mots de Bruijn d'ordre k est exponentiel en k.

Les mots de de Bruijn sont souvent utilisés comme exemple de cas limites en ce sens où ils contiennent tous les facteurs de longueur donnée.

Périodicité et bords

Soit x un mot non vide. Un entier p tel que $0 < p \leq |x|$ est une ***période*** de x si :

$x[i] = x[i + p]$

pour $i = 0, 1, \ldots, |x| - p - 1$. Remarquons que la longueur d'un mot non vide est une période de ce mot, de telle sorte que tout mot non vide possède au moins une période. On définit ainsi sans ambigüité aucune ***la période*** d'un mot non vide x comme la plus petite de ses périodes. On la note $pér(x)$. Par exemple, 3, 6, 7 et 8 sont des périodes du mot $x =$ `aabaabaa` et la période de x est $pér(x) = 3$.

On remarque que si p est une période de x, il en est de même de ses multiples kp, pour k entier satisfaisant $0 < k \leq \lfloor |x|/p \rfloor$.

Proposition 1.4
Soient x un mot non vide et p un entier tel que $0 < p \leq |x|$. Alors les cinq propriétés suivantes sont équivalentes :

1. *L'entier p est une période de x.*
2. *Il existe deux mots uniques $u \in A^*$ et $v \in A^+$ et un entier $k > 0$ tels que $x = (uv)^k u$ et $|uv| = p$.*
3. *Il existe un mot t et un entier $k > 0$ tels que $x \preceq_{\text{préf}} t^k$ et $|t| = p$.*
4. *Il existe trois mots u, v et w tels que $x = uw = wv$ et $|u| = |v| = p$.*
5. *Il existe un mot t tel que $x \preceq_{\text{préf}} tx$ et $|t| = p$.*

Preuve $1 \Rightarrow 2$: si $v \neq \varepsilon$ et $k > 0$, alors k est le dividende de la division entière de $|x|$ par p. Maintenant, si le triplet (u', v', k') satisfait les mêmes conditions que le triplet (u, v, k), on a $k' = k$ puis, pour une question de longueur, $|u'| = |u|$. Il vient à la suite que $u' = u$ et $v' = v$. Ce qui montre l'unicité de la décomposition sous réserve de son existence. Soient k et r le dividende et le reste de la division euclidienne de $|x|$ par p, puis u et v les deux facteurs de x définis par $u = x[0 \,..\, r - 1]$ et $v = x[r \,..\, p - 1]$.

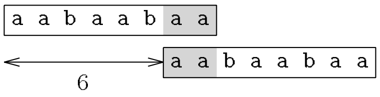


Figure 1.4 Dualité entre les notions de bords et périodes. Le mot aa est un bord du mot aabaabaa; il lui correspond la période $6 = |\mathsf{aabaabaa}| - |\mathsf{aa}|$.

Alors $x = (uv)^k u$ et $|uv| = p$. Cela montre l'existence du triplet (u, v, k) et achève la preuve de la propriété.

$2 \Rightarrow 3$: il suffit de considérer le mot $t = uv$.

$3 \Rightarrow 4$: soit w le suffixe de x défini par $w = t^{-1}x$. Comme $x \preceq_{préf} t^k$, w est également un préfixe de x. D'où l'existence des deux mots u $(= t)$ et v tels que $x = uw = wv$ et $|u| = |v| = |t| = p$.

$4 \Rightarrow 5$: puisque $uw \preceq_{préf} uwv$, on a $x \preceq_{préf} tx$ avec $|t| = p$ en posant simplement $t = u$.

$5 \Rightarrow 1$: soit i un entier tel que $0 \leq i \leq |x| - p - 1$. Alors :

$$\begin{array}{rcll} x[i+p] & = & (tx)[i+p] & (\text{car } x \preceq_{préf} tx) \\ & = & x[i] & (\text{car } |t| = p) \ . \end{array}$$

Ce qui montre que p est une période de x. ∎

On remarque en particulier que la propriété 3 s'exprime de façon un peu plus générale en remplaçant $\preceq_{préf}$ par $\preceq_{fact}$ (exercice 1.4).

Un ***bord*** d'un mot non vide x est un facteur propre de x qui est à la fois un préfixe et un suffixe de x. Ainsi, ε, a, aa et aabaa sont-ils les bords du mot aabaabaa.

Les notions de bords et de périodes sont duales comme le montre la propriété 4 de la proposition précédente (voir figure 1.4). La proposition qui suit exprime cette dualité en termes différents.

On introduit la fonction $Bord\colon A^* \to A^*$ définie pour tout mot non vide x par :

$$Bord(x) = \text{le plus long des bords de } x \ .$$

On dit de $Bord(x)$ qu'il est ***le bord*** de x. Par exemple, le bord de tout mot de longueur 1 est le mot vide et celui du mot aabaabaa est aabaa. Remarquons également que, lorsqu'il est défini, le bord d'un bord d'un mot x donné est aussi un bord de x.

Proposition 1.5

Soient x un mot non vide et n le plus grand des entiers k pour lequel $Bord^k(x)$ est défini (soit $Bord^n(x) = \varepsilon$). Alors

$$\langle Bord(x), Bord^2(x), \ldots, Bord^n(x)\rangle \tag{1.1}$$

est la suite des bords de x classés par ordre décroissant de longueur, et

$$\langle |x| - |Bord(x)|, |x| - |Bord^2(x)|, \ldots, |x| - |Bord^n(x)|\rangle \tag{1.2}$$

est la suite des périodes de x classées en ordre croissant.

Preuve On procède par récurrence sur la longueur des mots. L'énoncé de la proposition est valide lorsque la longueur du mot x est égale à 1, la suite des bords étant réduite à $\langle\varepsilon\rangle$ et celle des périodes à $\langle|x|\rangle$.

Soit x un mot de longueur supérieure à 2. Alors tout bord de x différent de $Bord(x)$ est un bord de $Bord(x)$, et réciproquement. Il vient par récurrence que la suite (1.1) est exactement la suite des bords de x. Maintenant, si p est une période de x, la proposition 1.4 assure l'existence de trois mots u, v et w tels que $x = uw = wv$ et $|u| = |v| = p$. Alors w est un bord de x et $p = |x| - |w|$. Il s'ensuit que la suite (1.2) est bien la suite des périodes de x. ∎

Lemme 1.6 (lemme de périodicité)
Si p et q sont des périodes d'un mot non vide x telles que

$$p + q - \mathrm{pgcd}(p, q) \leq |x| \ ,$$

alors $\mathrm{pgcd}(p, q)$ *est aussi une période de x.*

Preuve Par récurrence sur $\max\{p, q\}$. Le résultat est trivial lorsque $p = q = 1$ et, de façon plus générale, lorsque $p = q$. On peut dès lors supposer pour toute la suite que $p > q$.

D'après la proposition 1.4, le mot x s'écrit à la fois sous la forme uy avec $|u| = p$ et y un bord de x et sous la forme vz avec $|v| = q$ et z un bord de x.

Alors $p - q$ est une période de z. En effet, puisque $p > q$, y est un bord de x de longueur strictement inférieure à celle du bord z. Donc y est un bord de z. Il s'ensuit que $|z| - |y|$ est une période de z. Or $|z| - |y| = (|x| - q) - (|x| - p) = p - q$.

Mais q est également une période de z. En effet, puisque $p > q$ et $\mathrm{pgcd}(p, q) \leq p - q$, on obtient $q \leq p - \mathrm{pgcd}(p, q)$. On a d'autre part $p - \mathrm{pgcd}(p, q) = p + q - \mathrm{pgcd}(p, q) - q \leq |x| - q = |z|$. Il s'en déduit que $q \leq |z|$. Ce qui montre que la période q de x est également une période de son facteur z.

De plus, on a $(p - q) + q - \mathrm{pgcd}(p - q, q) = p - \mathrm{pgcd}(p, q)$, qui, comme on l'a vu ci-dessus, est une quantité inférieure à $|z|$.

On applique l'hypothèse de récurrence à $\max\{p - q, q\}$ relativement au mot z, et l'on obtient ainsi que $\mathrm{pgcd}(p, q)$ est une période de z.

Les conditions sur p et q (celle du lemme et $\mathrm{pgcd}(p, q) \leq p - q$) entrainent $q \leq |x|/2$. Et comme $x = vz$ et que z est un bord de x, v est un préfixe de z. Il a de plus une longueur qui est un multiple de $\mathrm{pgcd}(p, q)$. Soit t le préfixe de x de longueur $\mathrm{pgcd}(p, q)$. Alors v est une puissance de t et z est un préfixe d'une puissance de t. Il vient ensuite de la proposition 1.4 que x est un préfixe d'une puissance de t, et donc que $|t| = \mathrm{pgcd}(p, q)$ est une période de x. ∎

Pour illustrer ce résultat, considérons un mot x qui admette à la fois 5 et 8 comme périodes. Alors, si l'on suppose de plus que x est a b a a b | a b a a b | a b a a b | a b a a b

a b a a b a b a a b a

a b a a b a b a | a b a a b a b a | a b a a b a b a

Figure 1.5 Application du lemme de périodicité. Le mot abaababaaba de longueur 11 possède 5 et 8 comme périodes. Il n'est pas possible de le prolonger sur la gauche comme sur la droite tout en conservant ces deux périodes. En effet, si 5 et 8 sont des périodes d'un certain mot, mais que 1 – le plus grand commun diviseur de 5 et 8 – ne l'est pas, alors ce mot est de longueur strictement inférieure à $5+8-\mathrm{pgcd}(5,8)=12$.

composé d'au moins deux lettres distinctes, $\mathrm{pgcd}(5,8) = 1$ n'est pas une période de x, et, par application du lemme, la longueur de x est strictement inférieure à $5 + 8 - \mathrm{pgcd}(5,8) = 12$. C'est par exemple le cas des quatre mots de longueur supérieure à 8 qui sont des préfixes du mot abaababaaba de longueur 11. Une autre illustration du résultat est proposée figure 1.5.

On veut montrer dans ce qui suit que l'on ne peut pas affaiblir la condition requise sur les périodes dans l'énoncé du lemme de périodicité. Plus précisément, on donne l'exemple de mots x qui possèdent deux périodes p et q telles que $p+q-\mathrm{pgcd}(p,q) = |x|+1$ mais qui ne satisfont pas à la conclusion du lemme. (Voir aussi exercice 1.5.)

Soit $\beta\colon A^* \to A^*$ la fonction définie par

$$\beta(uab) = uba$$

pour tout mot $u \in A^*$ et toutes lettres $a, b \in A$.

Lemme 1.7
Pour tout naturel $n \geq 3$, $\beta(f_n) = f_{n-2}f_{n-1}$.

Preuve Par récurrence sur n. Le résultat est trivial lorsque $3 \leq n \leq 4$. Si $n \geq 5$, on a :

$$\begin{aligned}
\beta(f_n) &= \beta(f_{n-1}f_{n-2}) && \text{(par définition de } f_n) \\
&= f_{n-1}\beta(f_{n-2}) && \text{(car } |f_{n-2}| = F_{n-2} \geq 2) \\
&= f_{n-1}f_{n-4}f_{n-3} && \text{(par hypothèse de récurrence)} \\
&= f_{n-2}f_{n-3}f_{n-4}f_{n-3} && \text{(par définition de } f_{n-1}) \\
&= f_{n-2}f_{n-2}f_{n-3} && \text{(par définition de } f_{n-2}) \\
&= f_{n-2}f_{n-1} && \text{(par définition de } f_{n-1})\ .
\end{aligned}$$
■

Pour tout naturel $n \geq 3$, on définit le mot g_n comme le préfixe de longueur $F_n - 2$ de f_n, c'est-à-dire f_n privé de ses deux dernières lettres.

Lemme 1.8
Pour tout naturel $n \geq 6$, $g_n = {f_{n-2}}^2 g_{n-3}$.

Preuve On a :

$$\begin{aligned}
f_n &= f_{n-1}f_{n-2} && \text{(par définition de } f_n\text{)}\\
&= f_{n-2}f_{n-3}f_{n-2} && \text{(par définition de } f_{n-1}\text{)}\\
&= f_{n-2}\beta(f_{n-1}) && \text{(d'après le lemme 1.7)}\\
&= f_{n-2}\beta(f_{n-2}f_{n-3}) && \text{(par définition de } f_{n-1}\text{)}\\
&= {f_{n-2}}^2\beta(f_{n-3}) && \text{(car } |f_{n-3}| = F_{n-3} \geq 2\text{)} .
\end{aligned}$$

Le résultat annoncé s'en déduit immédiatement. ■

Lemme 1.9
Pour tout naturel $n \geq 3$, $g_n \preceq_{préf} {f_{n-1}}^2$ et $g_n \preceq_{préf} {f_{n-2}}^3$.

Preuve On a :

$$\begin{aligned}
g_n &\preceq_{préf} f_nf_{n-3} && \text{(car } g_n \preceq_{préf} f_n\text{)}\\
&= f_{n-1}f_{n-2}f_{n-3} && \text{(par définition de } f_n\text{)}\\
&= {f_{n-1}}^2 && \text{(par définition de } f_{n-1}\text{)} .
\end{aligned}$$

La seconde relation est valide lorsque $3 \leq n \leq 5$. Lorsque $n \geq 6$, on a :

$$\begin{aligned}
g_n &= {f_{n-2}}^2g_{n-3} && \text{(d'après le lemme 1.8)}\\
&\preceq_{préf} {f_{n-2}}^2f_{n-3}f_{n-4} && \text{(car } g_{n-3} \preceq_{préf} f_{n-3}\text{)}\\
&= {f_{n-2}}^3 && \text{(par définition de } f_{n-2}\text{)} .
\end{aligned}$$

■

Maintenant, soit n un naturel tel que $n \geq 5$, de telle sorte que le mot g_n soit à la fois défini et de longueur supérieure à 2. Il vient alors :

$$\begin{aligned}
|g_n| &= F_n - 2 && \text{(par définition de } g_n\text{)}\\
&= F_{n-1} + F_{n-2} - 2 && \text{(par définition de } F_n\text{)}\\
&\geq F_{n-1} && \text{(car } F_{n-2} \geq 2\text{)} .
\end{aligned}$$

Il résulte de cette inégalité, du lemme 1.9 et de la proposition 1.4 que F_{n-1} et F_{n-2} sont deux périodes de g_n. Remarquons également que, puisque $\mathrm{pgcd}(F_{n-1}, F_{n-2}) = 1$, on a aussi :

$$\begin{aligned}
F_{n-1} + F_{n-2} - \mathrm{pgcd}(F_{n-1}, F_{n-2}) &= F_n - 1\\
&= |g_n| + 1 .
\end{aligned}$$

Si donc la conclusion du lemme de périodicité s'appliquait au mot g_n et ses deux périodes F_{n-1} et F_{n-2}, g_n serait la puissance d'un mot de longueur 1. Or les deux premières lettres de g_n sont distinctes. Ce qui indique que la condition du lemme de périodicité est en un certain sens optimale.

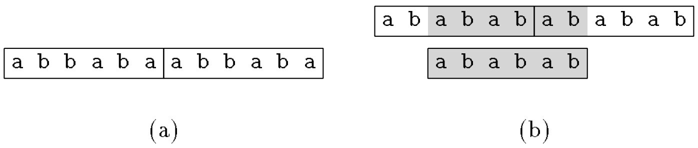


Figure 1.6 Application du lemme de primitivité. **(a)** Le mot x = abbaba ne possède pas d'occurrence « non triviale » dans son carré x^2 – c'est-à-dire qui ne soit ni un préfixe ni un suffixe de x^2 – car x est primitif. **(b)** Le mot x = ababab possède une occurrence « non triviale » dans son carré x^2 car x n'est pas primitif : $x = (\mathtt{ab})^3$.

Puissances, primitivité et conjugaison

Lemme 1.10
Soient x et y deux mots. S'il existe deux naturels non nuls m et n tels que $x^m = y^n$, x et y sont des puissances d'un certain mot z.

Preuve Il suffit de montrer le résultat dans le cas non trivial où ni x ni y ne sont vides. Deux sous-cas peuvent alors être distingués, selon que $\min\{m, n\}$ est égal ou non à 1.

Si $\min\{m, n\} = 1$, il suffit de considérer le mot $z = y$ si $m = 1$ et $z = x$ sinon.

Sinon, $\min\{m, n\} \geq 2$. On remarque alors que $|x|$ et $|y|$ sont des périodes du mot $t = x^m = y^n$ qui vérifient la condition du lemme de périodicité : $|x| + |y| - \mathrm{pgcd}(|x|, |y|) \leq |x| + |y| - 1 < |t|$. En conséquence de quoi il suffit de considérer le mot z défini comme le préfixe de t de longueur $\mathrm{pgcd}(|x|, |y|)$ pour obtenir le résultat annoncé. ■

Un mot non vide est ***primitif*** s'il n'est puissance d'aucun autre mot que lui-même. Autrement dit, un mot $x \in A^+$ est primitif si et seulement si toute décomposition de la forme $x = u^n$ avec $u \in A^*$ et $n \in \mathbf{N}$ implique $n = 1$. Par exemple, le mot abaab est primitif, tandis que les mots ε et $\mathtt{bababa} = (\mathtt{ba})^3$ ne le sont pas.

Lemme 1.11 (lemme de primitivité)
Un mot non vide est primitif si et seulement si il n'est un facteur de son carré qu'en tant que préfixe et suffixe. Autrement dit, pour tout mot non vide x,

x primitif

si et seulement si

$yx \preceq_{\text{préf}} x^2$ *implique* $y = \varepsilon$ *ou* $y = x$.

Une illustration de ce résultat est proposée figure 1.6.

Preuve Si x est un mot non vide non primitif, il existe $z \in A^+$ et $n \geq 2$ tels que $x = z^n$. Puisque x^2 se décompose sous la forme $z \cdot z^n \cdot z^{n-1}$, le mot x apparait à la position $|z|$ sur x^2. Cela montre que tout mot non vide non primitif est un facteur de son carré sans en être seulement un préfixe et un suffixe.

Réciproquement, soit x un mot non vide tel que son carré x^2 s'écrive sous la forme yxz avec $y, z \in A^+$. Pour une question de longueur, il vient tout d'abord que $|y| < |x|$. Ensuite, et puisque $x \preceq_{\textit{préf}} yx$, on obtient de la proposition 1.4 que $|y|$ est une période de x. Ainsi $|x|$ et $|y|$ sont-elles des périodes de yx. D'après le lemme de périodicité, on en déduit que $p = \text{pgcd}(|x|, |y|)$ est également une période de yx. Maintenant, comme $p \leq |y| < |x|$, p est aussi une période de x. Et comme p divise $|x|$, on en déduit que x est de la forme t^n avec $|t| = p$ et $n \geq 2$. Cela montre que le mot x n'est pas primitif. ∎

Proposition 1.12
Pour tout mot non vide, il existe un et un seul mot primitif dont il est une puissance.

Preuve La preuve de l'existence découle d'une récurrence triviale sur la longueur des mots. On s'attache maintenant à montrer l'unicité.

Soit x un mot non vide. Si l'on suppose que $x = u^m = v^n$ pour deux mots primitifs u et v et deux naturels non nuls m et n, alors u et v sont nécessairement des puissances d'un mot $z \in A^+$ d'après le lemme 1.10. Il s'ensuit que $z = u = v$, ce qui montre l'unicité et termine la preuve. ∎

Si x est un mot non vide, on dit du mot primitif z dont x est la puissance qu'il est la ***racine*** de x, et du naturel n tel que $x = z^n$ qu'il est l'***exposant***[3] de x.

Deux mots x et y sont ***conjugués*** s'il existe deux mots u et v tels que $x = uv$ et $y = vu$. Par exemple, les mots abaab et ababa sont conjugués. Il est clair que la conjugaison est une relation d'équivalence. Elle n'est pas compatible avec le produit.

Proposition 1.13
Deux mots non vides sont conjugués si et seulement si leurs racines le sont également.

Preuve La preuve de la réciproque est immédiate.

Pour la preuve de l'implication directe, on considère deux mots conjugués non vides x et y dont on note z et t puis m et n les racines puis les exposants respectifs. Puisque x et y sont conjugués, il existe $z', z'' \in A^+$ et $p, q \in \mathbf{N}$ tels que $z = z'z''$, $x = z^p z' \cdot z'' z^q$, $y = z'' z^q \cdot z^p z'$ et

---

3. De manière plus générale, l'exposant de x est la quantité $|x|/\textit{pér}(x)$ qui n'est pas nécessairement entière (voir exercice 9.2).

$m = p + q + 1$. On en déduit que $y = (z''z')^m$. Maintenant, comme t est primitif, le lemme 1.10 entraine que $z''z'$ est une puissance de t. D'où l'existence d'un naturel non nul k tel que $|z| = k|t|$. Par symétrie, il existe un naturel non nul ℓ tel que $|t| = \ell|z|$. Il s'ensuit que $k = \ell = 1$, que $|t| = |z|$, puis que $t = z''z'$. Cela montre que les mots z et t sont conjugués. ■

Proposition 1.14
Deux mots non vides x et y sont conjugués si et seulement si il existe un mot z tel que $xz = zy$.

Preuve $\Rightarrow$: si x et y se décomposent sous la forme $x = uv$ et $y = vu$ avec $u, v \in A^*$, alors le mot $z = u$ convient puisque $xz = uvu = zy$.

$\Leftarrow$: dans le cas non trivial où $z \in A^+$, on obtient par une récurrence immédiate que $x^k z = zy^k$ pour tout $k \in \mathbf{N}$. Soit maintenant n le naturel (non nul) tel que $(n-1)|x| \leq |z| < n|x|$. Il existe donc $u, v \in A^*$ tels que $x = uv$, $z = x^{n-1}u$ et $vz = y^n$. Il s'en déduit que $y^n = vx^{n-1}u = (vu)^n$. Finalement, puisque $|y| = |x|$, on a $y = vu$, ce qui montre que x et y sont conjugués. ■

1.3 Algorithmes et complexité

Dans cette section, on présente des éléments algorithmiques utilisés dans la suite de l'ouvrage. Ils comprennent les conventions d'écriture, l'évaluation de la complexité des algorithmes et quelques objets standard.

Convention d'écriture des algorithmes

Le style du langage d'expression algorithmique employé ici est relativement proche des langages de programmation réels mais se place à un niveau d'abstraction supérieur.

On adopte les conventions suivantes :

- l'indentation signifie la structure de bloc inhérente aux instructions composées ;
- les lignes de code sont numérotées à la seule fin de pouvoir être référencées dans le texte ;
- l'accès à un attribut particulier d'un objet est signifié par le nom de l'attribut suivi de l'identificateur associé à l'objet entre crochets ;
- une variable qui représente un objet donné (table, file, arbre, mot, automate) est un pointeur vers cet objet ;
- les paramètres passés lors d'appels de procédures ou de fonctions le sont par valeur ;
- la portée des variables des procédures et des fonctions est locale sauf mention contraire ;

- l'évaluation des expressions booléennes s'effectue de la gauche vers la droite de façon paresseuse.

On considère, à l'instar d'un langage comme le C, l'instruction itérative **faire-tantque** – utilisée en lieu et place de l'instruction traditionnelle **répéter-jusque** – et l'instruction **rupture** qui provoque la terminaison de la boucle la plus interne dans laquelle elle figure.

Adaptée au traitement séquentiel de mots, on utilise la formulation :

pour chaque lettre a de u, séquentiellement **faire**
  traitement de a

pour tout mot u. Elle signifie que les lettres $u[i]$, $i = 0, 1, \ldots, |u| - 1$, composant u sont traitées les unes à la suite des autres dans le corps de la boucle : d'abord $u[0]$, puis $u[1]$, et ainsi de suite. Elle sous-entend que la longueur du mot u n'a pas nécessairement à être connue à l'avance, l'arrêt de la boucle pouvant se réaliser à l'aide d'un marqueur de fin de mot. Dans le cas où la longueur du mot u est connue, cette formulation est équivalente à une formulation du type :

pour $i \leftarrow 0$ **à** $|u| - 1$ **faire**
  $a \leftarrow u[i]$
  traitement de a

où la variable entière i est libre (son utilisation ne provoque pas de conflit avec l'environnement).

Algorithmes de recherche de motifs

Un ***motif*** représente un langage non vide ne contenant pas le mot vide. Il peut être décrit par un mot, par un ensemble fini de mots, ou autrement. Le problème de la ***recherche de motifs*** est celui de la localisation d'occurrences de mots du langage dans d'autres mots – ou dans des ***textes*** pour parler de manière moins formelle. Les notions d'occurrence, d'apparition et de position sur les mots sont étendues aux motifs.

Selon le problème spécifié, l'entrée d'un algorithme de recherche d'un motif est un mot x, un langage X ou autre, et un texte y, accompagnés ou non de leurs longueurs.

La sortie peut prendre plusieurs formes. En voici quelques unes :

- pour implanter un algorithme qui teste si le motif apparait dans le texte ou non, sans précision sur les positions des occurrences éventuelles, la sortie est simplement la valeur booléenne VRAI dans la première éventualité et FAUX dans la seconde ;
- lors d'une recherche séquentielle, il est opportun de produire un mot $\bar{y}$ sur l'alphabet $\{\mathtt{0}, \mathtt{1}\}$ qui code l'existence de positions droites d'occurrences. Le mot $\bar{y}$ est tel que $|\bar{y}| = |y|$ et $\bar{y}[i] = \mathtt{1}$ si et seulement si i est la position droite d'une occurrence du motif dans y ;

– la sortie peut aussi prendre la forme d'un ensemble P de positions gauches – ou droites – d'occurrences du motif dans y.

Soit e un prédicat de valeur VRAI si et seulement si une occurrence vient d'être localisée. Une fonction correspondant à la première des formes évoquées et prenant fin dès qu'une occurrence est localisée se doit d'intégrer dans son code une instruction comme :

1 **si** e **alors**
2   **retourner** VRAI

au cœur même de son procédé de recherche et de retourner la valeur FAUX à la terminaison de ce procédé. Pour la deuxième forme, il s'agit d'initialiser la variable $\bar{y}$ à ε, le mot vide, puis de modifier sa valeur par une instruction comme :

1 **si** e **alors**
2     $\bar{y} \leftarrow \bar{y} \cdot \mathtt{1}$
3 **sinon** $\bar{y} \leftarrow \bar{y} \cdot \mathtt{0}$

puis de la retourner à la terminaison. De même pour la troisième forme, où l'ensemble P est initialement vide, puis augmenté par une instruction comme :

1 **si** e **alors**
2     $P \leftarrow P \cup \{$la position courante sur $y\}$

puis enfin retourné.

Afin de ne pas avoir à présenter plusieurs variantes du code d'un même algorithme, on considère l'instruction spéciale suivante :

SIGNALER-SI(e) signifie, à l'endroit où elle est donnée, la localisation d'une occurrence du motif à la position courante sur le texte à la condition que le prédicat e ait pour valeur VRAI.

Expression de la complexité

Le modèle de calcul pour l'évaluation de la complexité des algorithmes est le modèle standard de machine à accès direct.

De façon générale, la complexité des algorithmes se traduit par des expressions intégrant la taille de l'entrée. Ce qui inclut la longueur du langage représenté par le motif, la longueur du mot sur lequel s'effectue la recherche et la taille de l'alphabet. On suppose que les lettres de l'alphabet sont de taille comparable à celle des mots machine, et, qu'en conséquence, la comparaison de deux lettres entre elles est une opération élémentaire qui s'effectue en temps constant.

On suppose que toute instruction SIGNALER-SI(e) s'exécute en temps constant [4] une fois le prédicat e évalué.

On utilise les notations préconisées par Knuth pour exprimer les ordres de grandeur. Soient f et g deux fonctions de $\mathbf{N}$ dans $\mathbf{N}$. On écrit « $f(n)$ est $O(g(n))$ » pour signifier qu'il existe une constante K et un naturel n_0 tels que $f(n) \leq Kg(n)$ pour tout $n \geq n_0$. De façon duale, on écrit « $f(n)$ est $\Omega(g(n))$ » s'il existe une constante K et un naturel n_0 tels que $f(n) \geq Kg(n)$ pour tout $n \geq n_0$. On écrit enfin « $f(n)$ est $\Theta(g(n))$ » pour signifier que f et g sont du même ordre, à savoir que $f(n)$ est à la fois $O(g(n))$ et $\Omega(g(n))$.

La fonction $f\colon \mathbf{N} \to \mathbf{N}$ est ***linéaire*** si $f(n)$ est $\Theta(n)$, ***quadratique*** si $f(n)$ est $\Theta(n^2)$, ***cubique*** si $f(n)$ est $\Theta(n^3)$, ***logarithmique*** si $f(n)$ est $\Theta(\log n)$, ***exponentielle*** s'il existe $a > 0$ pour lequel $f(n)$ est $\Theta(a^n)$.

On dit d'une fonction de deux paramètres $f\colon \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ qu'elle est linéaire lorsque $f(m,n)$ est $\Theta(m+n)$ et quadratique lorsque $f(m,n)$ est $\Theta(m \times n)$.

Quelques objets standard

Les files, les états et les automates sont des objets souvent utilisés dans la suite. Sans préjuger de leurs implantations effectives – pouvant d'ailleurs différer d'un algorithme à l'autre – on indique les attributs et les opérations définies au minimum sur ces objets.

Pour les files, on se contente de ne décrire que les opérations de base.

FILE-VIDE() crée puis retourne une file vide.

FILE-EST-VIDE(F) retourne VRAI si la file F est vide, et FAUX sinon.

ENFILER(F, x) ajoute l'élément x en queue de la file F.

TÊTE(F) retourne l'élément situé en tête de la file F.

DÉFILER(F) supprime l'élément situé en tête de la file F.

DÉFILEMENT(F) supprime l'élément situé en tête de la file F puis le retourne ;

LONGUEUR(F) retourne la longueur de la file F.

Les états sont des objets qui possèdent au moins les deux attributs *terminal* et *Succ*. Le premier indique si l'état est terminal ou non et le second est une implantation de l'ensemble des successeurs étiquetés de l'état. L'attribut correspondant à une sortie d'un état est noté *sortie*. Les deux opérations standard sur les états sont les fonctions NOUVEL-ÉTAT et CIBLE. Tandis que la première crée puis retourne un état non

4. On peut en fait toujours s'y ramener quand bien même le langage représenté par le motif ne serait pas réduit à un seul mot. Il suffit pour cela de ne produire qu'un descripteur – préalablement calculé – de l'ensemble des mots qui apparaissent à la position courante (au lieu par exemple de produire explicitement l'ensemble des mots). Reste ensuite à la charge d'un utilitaire de développer l'information si nécessaire.

terminal d'ensemble des successeurs étiquetés vide, la seconde retourne la cible d'une flèche dont ne sont données que la source et l'étiquette, ou la valeur spéciale NIL si une telle flèche n'existe pas. Le code de ces deux fonctions s'écrit en quelques lignes :

NOUVEL-ÉTAT()

allouer un objet p de type état
terminal$[p] \leftarrow$ FAUX
Succ$[p] \leftarrow \emptyset$
retourner p

CIBLE(p, a)

si il existe un état q tel que $(a, q) \in Succ[p]$ **alors**
    retourner q
sinon retourner NIL

Les objets du type automate possèdent au moins l'attribut *initial* qui spécifie l'état initial de l'automate. La fonction NOUVEL-AUTOMATE crée puis retourne un automate à un seul état posé comme étant son état initial, d'ensemble des successeurs étiquetés vide. Le code correspondant est le suivant :

NOUVEL-AUTOMATE()

allouer un objet M de type automate
$q_0 \leftarrow$ NOUVEL-ÉTAT()
initial$[M] \leftarrow q_0$
retourner M

1.4 Mise en mémoire d'automates

Certains algorithmes de recherche de motifs reposent sur des implantations particulières des automates déterministes qu'ils considèrent. Cette section détaille plusieurs procédés, comprenant les structures de données et les algorithmes, qui peuvent être utilisés pour implanter ces objets en mémoire.

Implanter un automate déterministe (Q, q_0, T, F) revient à mettre en mémoire, soit l'ensemble F de ses flèches, soit les ensembles des successeurs étiquetés de ses états, soit sa fonction de transition δ. Ce sont là des problèmes équivalents qui rentrent dans le cadre général du problème de la représentation des ***fonctions partielles*** (exercice 1.15). On distingue deux familles d'implantations :

- la famille des implantations ***pleines*** dans lesquelles figurent toutes les transitions ;

	a	b	c
0	1	0	0
1	2	3	0
2	2	3	0
3	4	0	0
4	2	3	0

Figure 1.7 La matrice de transition de l'automate de la figure 1.2.

– la famille des implantations ***réduites*** qui font appel à des techniques plus ou moins élaborées de compression destinées à réduire l'espace mémoire de la représentation.

Le choix de l'implantation influence le temps nécessaire au calcul d'une transition, c'est-à-dire à l'exécution de $\text{CIBLE}(p, a)$, pour tout état $p \in Q$ et toute lettre $a \in A$. Ce temps de calcul est appelé le ***délai*** en cela qu'il mesure également le temps nécessaire pour passer de la lettre courante sur l'entrée à la lettre qui la suit. Typiquement, deux modèles peuvent être opposés :

– le ***modèle branchements*** dans lequel δ est implantée à l'aide d'une matrice $Q \times A$ et où le délai est constant ;
– le ***modèle comparaisons*** dans lequel l'opération élémentaire est la comparaison de lettres et où le délai est $O(\log \operatorname{card} A)$, deux lettres quelconques pouvant être comparées en une unité de temps (hypothèse générale formulée dans la section 1.3).

On considère également dans la section suivante une technique élémentaire dite « modèle vecteur-binaire » dont le champ d'application est restreint : elle ne présente d'intérêt que lorsque la taille de l'automate est très faible.

Pour chacune des familles d'implantations, on précise les ordres de grandeur de l'espace mémoire nécessaire et du délai. Il y a toujours un compromis à trouver entre ces deux quantités.

Implantations pleines

La méthode la plus simple pour implanter la fonction δ est de ranger ses valeurs dans une matrice $Q \times A$, dite ***matrice de transition*** (une illustration est donnée figure 1.7). Il s'agit là d'une méthode de choix pour un automate déterministe complet sur un alphabet de taille relativement faible et lorsque les lettres peuvent être assimilées à des indices sur une table.

Proposition 1.15
Dans une implantation par matrice de transition, l'espace mémoire nécessaire est $O(\operatorname{card} Q \times \operatorname{card} A)$ et le délai $O(1)$. ∎

Dans le cas où l'automate n'est pas complet, la représentation reste correcte excepté le fait que l'exécution de l'automate sur le texte donné en entrée peut maintenant s'arrêter sur une transition non définie. La matrice peut toutefois être initialisée en temps $O(\mathrm{card}\, F)$ si l'on fait appel à une technique d'implantation des fonctions partielles comme celle proposée exercice 1.15. Les complexités annoncées plus haut tant pour l'espace mémoire nécessaire que pour le délai restent valides.

Un automate peut être implanté au moyen de matrices d'adjacence comme il est classique de le faire pour les graphes. On associe alors à chaque lettre de l'alphabet une matrice définie sur $Q \times Q$ à valeurs booléennes. Cette représentation n'est en général pas adaptée pour les applications développées dans cet ouvrage. Elle est cependant reliée à la méthode qui suit.

La méthode par ***liste de transitions*** consiste à implanter une liste des triplets (p, a, q) qui sont des flèches de l'automate. L'espace nécessaire est seulement $O(\mathrm{card}\, F)$. Ce faisant, on suppose que cette liste est rangée dans une table de hachage de façon à permettre un calcul rapide des transitions. La fonction de hachage correspondante est définie sur les couples $(p, a) \in Q \times A$. Étant donné un couple (p, a), l'accès à la transition (p, a, q), si elle est définie, se fait en moyenne en temps constant sous les habituelles hypothèses propres à ce type de technique.

Ces premiers types de représentations supposent implicitement que l'alphabet est fixé et connu à l'avance, ce qui les opposent aux représentations du modèle comparaisons considérées par la méthode exposée ci-dessous.

La méthode par ***ensembles des successeurs étiquetés*** consiste à utiliser une table t indicée sur Q dont chaque élément $t[p]$ donne accès à une implantation de l'ensemble des successeurs étiquetés de l'état p. L'espace nécessaire est $O(\mathrm{card}\, Q + \mathrm{card}\, F)$. Cette méthode est valable même lorsque la seule opération autorisée sur les lettres est la comparaison. En notant s est le maximum des degrés sortants des états, le délai est $O(\log s)$ si l'on a recours à une implantation efficace des ensembles de successeurs étiquetés.

Proposition 1.16
Dans une implantation par ensembles des successeurs étiquetés, l'espace mémoire nécessaire est $O(\mathrm{card}\, Q + \mathrm{card}\, F)$ et le délai $O(\log s)$ où s est le maximum des degrés sortants des états. ■

Remarquons que le délai est également $O(\log \mathrm{card}\, A)$ dans ce cas : en effet, l'automate étant supposé déterministe, le degré sortant de chacun des états est inférieur à $\mathrm{card}\, A$, soit $s \leq \mathrm{card}\, A$ avec les notations utilisées ci-dessus.

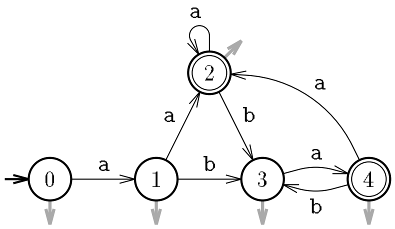


Figure 1.8 Implantation réduite par adjonction de successeurs par défaut. On considère l'automate de la figure 1.2 et l'on choisit l'état initial comme unique successeur par défaut (ce choix convient parfaitement pour les problèmes de recherche de motifs). Ceux des états qui admettent l'état initial comme successeur par défaut (en fait tous ici) sont indiqués par une courte flèche grisée. La cible de la transition de l'état 3 par la lettre a est l'état 4, et par toute autre lettre, ici b ou c, l'état initial 0.

Implantations réduites

Lorsque l'automate est complet, la complexité en espace peut être réduite en considérant un ***successeur par défaut*** pour le calcul des transitions à partir de n'importe quel état donné – l'état apparaissant le plus souvent dans un ensemble des successeurs étiquetés étant le meilleur des candidats possibles au titre de successeur par défaut. Le délai peut du même coup s'en trouver réduit puisque la taille des ensembles des successeurs étiquetés est moins importante. Pour les problèmes de recherche de motifs, le choix de l'état initial comme successeur par défaut convient parfaitement. Aussi convient-on d'indiquer qu'un état possède l'état initial comme successeur par défaut par une courte flèche grisée sortant de ce premier état (un exemple est montré figure 1.8.)

Une autre méthode consiste à utiliser une fonction de suppléance. L'idée est ici de réduire l'espace nécessaire à l'implantation de l'automate, en renvoyant, dans la plupart des cas, le calcul de la transition à partir de l'état courant à celui du calcul à partir d'un autre état mais pour la même lettre. Cette technique sert à implanter des automates déterministes dans le modèle comparaisons. Son principal avantage est – en général – de fournir des représentations de taille linéaire et d'obtenir simultanément un temps linéaire pour le calcul de séries de transitions quand bien même le calcul d'une seule transition ne se fait pas en temps constant.

Formellement, soient

$$\gamma : Q \times A \to Q$$

et

$$f : Q \to Q$$

deux fonctions. On dit que le couple (γ, f) représente la fonction de

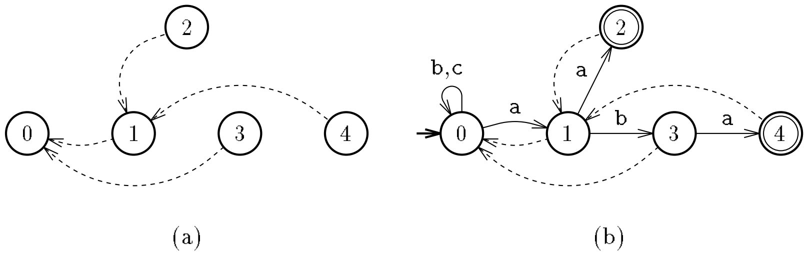


Figure 1.9 Implantation réduite par adjonction d'une fonction de suppléance. On reprend l'exemple de l'automate de la figure 1.2. **(a)** Une fonction de suppléance donnée sous la forme d'un graphe orienté. Comme ce graphe ne possède pas de circuit, la fonction définit un ordre sur l'ensemble des états. **(b)** L'automate réduit correspondant. Chaque lien état-état suppléant est indiqué par un arc dirigé dessiné en pointillé. Le calcul de la transition de l'état 4 par la lettre c est renvoyé à l'état 1, puis à l'état 0. L'état 0 est en effet le premier des états 4, 1 et 0, dans cet ordre, à posséder une transition définie pour c. Au final, la cible de la transition de l'état 4 par c est l'état 0.

transition δ d'un automate complet si et seulement si γ est une sous-fonction de δ, f définit un ordre sur Q, et pour tout couple $(p,a) \in Q \times A$

$$\delta(p,a) = \begin{cases} \gamma(p,a) & \text{si } \gamma(p,a) \text{ est défini} \ , \\ \delta(f(p),a) & \text{sinon} \ . \end{cases}$$

Lorsque qu'il est défini, on dit de l'état $f(p)$ qu'il est le ***suppléant*** de l'état p. On dit des fonctions γ et f qu'elle sont respectivement, et conjointement, une ***sous-transition*** et une ***fonction de suppléance*** de δ.

On convient d'indiquer le lien état-état suppléant par un arc dirigé dessiné en pointillé (voir l'exemple figure 1.9).

L'espace nécessaire pour représenter la fonction δ par les fonctions γ et f est $O(\text{card}\, Q + \text{card}\, F')$ dans le cas d'une implantation par ensembles des successeurs étiquetés où

$$F' = \{(p,a,q) \in F : \gamma(p,a) \text{ est défini}\} \ .$$

Notons au passage que γ est la fonction de transition de l'automate (Q, q_0, T, F').

Un exemple intégré

La méthode présentée maintenant est une combinaison des précédentes alliant un calcul rapide des transitions et une représentation compacte des transitions dues à l'utilisation conjointe de tables et d'une fonction de suppléance. Elle est connue sous le nom de « compression de table de transition ».

Deux attributs supplémentaires, *suppléant* et *base*, sont adjoints aux états, le premier à valeurs dans Q et le second dans $\mathbf{N}$. On considère aussi deux tables indexées dans $\mathbf{N}$ et à valeurs dans Q : *cible* et *contrôle*. Pour chaque couple $(p, a) \in Q \times A$, $\mathit{base}[p] + \mathit{rang}[a]$ est un indice sur *cible* et *contrôle*, en notant *rang* la fonction qui associe à toute lettre de A son rang dans A.

Le calcul de la transition d'un état $p \in Q$ par une lettre $a \in A$ procède comme suit :

1. Si $\mathit{contrôle}[\mathit{base}[p] + \mathit{rang}[a]] = p$, $\mathit{cible}[\mathit{base}[p] + \mathit{rang}[a]]$ est la cible de la flèche de source p et d'étiquette a.
2. Sinon le traitement est répété récursivement sur l'état $\mathit{suppléant}[p]$ et la lettre a (à supposer que *suppléant* soit bien une fonction de suppléance).

Suit le code (non récursif) de la fonction correspondante.

Cible-par-compression(p, a)

tantque $\mathit{contrôle}[\mathit{base}[p] + \mathit{rang}[a]] \neq p$ **faire**
    $p \leftarrow \mathit{suppléant}[p]$
retourner $\mathit{cible}[\mathit{base}[p] + \mathit{rang}[a]]$

Dans le pire des cas, l'espace nécessaire est $O(\mathrm{card}\, Q \times \mathrm{card}\, A)$ et le délai $O(\mathrm{card}\, Q)$. Cela dit, cette méthode permet de réduire l'espace en $O(\mathrm{card}\, Q + \mathrm{card}\, A)$ avec un délai constant dans le meilleur des cas.

1.5 Techniques de base

On présente dans cette section des approches élémentaires au problème de la recherche de motifs. Elles comprennent la notion de fenêtre glissante commune à de nombreux algorithmes de recherche, l'utilisation d'heuristiques pour réduire le temps de calcul, la méthode générale basée sur un automate quand les textes doivent être traité de manière séquentielle et l'emploi de techniques qui reposent sur le codage binaire réalisé par les mots machine.

Notion de fenêtre glissante

Lorsque le motif est un mot non vide x de longueur m, il est agréable de considérer que le texte y de longueur n sur lequel s'effectue la localisation est examiné au moyen d'une ***fenêtre glissante***. La fenêtre délimite un facteur sur le texte – appelé le ***contenu de la fenêtre*** – qui a, dans la plupart des cas, la longueur du mot x. Elle glisse le long du texte de la gauche vers la droite.

La fenêtre étant à une position j donnée sur le texte, l'algorithme teste si le mot x apparait ou non à cette position, en comparant certaines lettres du contenu de la fenêtre avec les lettres du mot alignées en correspondance. On parle de ***tentative*** à la position j (voir l'exemple

y	a	a	b	a	a	b	a	b	a	a	b	a	b	b	b	a	a	a	b	b
x						a	a	b	b	a	a	a								

Figure 1.10 Une tentative de localisation du mot x = aabbaaa dans le mot y = aabaababaababbbaaabb. La tentative a lieu à la position 5 sur y. Le contenu de la fenêtre et le mot coïncident en quatre positions.

figure 1.10). Le cas échéant, l'occurrence est signalée. Lors de cette phase de test, l'algorithme acquiert une certaine quantité d'informations sur le texte laquelle peut être exploitée de deux façons :

- pour déterminer la longueur du prochain ***décalage*** de la fenêtre selon des règles qui sont propres à l'algorithme ;
- pour éviter des comparaisons lors des tentatives suivantes en mémorisant une partie de l'information recueillie.

Quand le décalage fait passer la fenêtre de la position j à la position $j + d$ ($d \geq 1$), on dit que le décalage est de ***longueur*** d. Pour répondre au problème posé, un décalage de longueur d pour une tentative à la position j doit être ***valide***, c'est-à-dire qu'il doit assurer que, pour $d \geq 2$, il n'y a pas d'occurrence du mot cherché x des positions $j+1$ à $j+d-1$ sur le texte y.

L'algorithme naïf

L'implantation la plus simple du mécanisme de la fenêtre glissante est l'algorithme dit naïf. La stratégie consiste ici à considérer une fenêtre de longueur m et à la faire glisser d'une position vers la droite après chaque tentative. Ce qui conduit, dès lors que la comparaison du contenu de la fenêtre et du mot est correctement implantée, à un algorithme correct.

On donne ci-dessous le code de l'algorithme. La variable j correspond à la position gauche de la fenêtre sur le texte. Il est entendu que la comparaison des mots à la ligne 2 s'effectue lettre à lettre selon un ordre préétabli.

Localiser-naïvement(x, m, y, n)
1 **pour** $j \leftarrow 0$ **à** $n - m$ **faire**
2 Signaler-si$(y[j \mathinner{..} j + m - 1] = x)$

Dans le pire des cas, l'algorithme Localiser-naïvement s'exécute en temps $\Theta(m \times n)$, comme par exemple lorsque x et y sont des puissances de la même lettre. Dans le cas moyen [5] en revanche, son comportement est plutôt bon, comme l'indique la proposition suivante.

5. Quand bien même les motifs et les textes considérés dans la pratique n'ont aucune raison d'être aléatoires, les cas moyens rendent compte de ce que l'on peut attendre d'un algorithme de recherche de motifs donné.

Proposition 1.17
Sous la double hypothèse d'un alphabet non réduit à une seule lettre et d'une distribution uniforme et indépendante des lettres de l'alphabet, le nombre moyen de comparaisons de lettres effectuées par l'opération LOCALISER-NAÏVEMENT(x, m, y, n) *est* $\Theta(n - m)$.

Preuve Soit c le cardinal de l'alphabet. Le nombre de comparaisons de lettres nécessaires pour déterminer si deux mots u et v de longueur m sont identiques ou non est en moyenne

$$1 + 1/c + \cdots + 1/c^{m-1} \ ,$$

indépendamment de la permutation sur les positions suivant laquelle sont comparées entre elles les lettres des mots. Lorsque $c \geq 2$, cette quantité est inférieure à $1/(1 - 1/c)$, elle-même inférieure à 2.

Il s'ensuit que le nombre moyen de comparaisons de lettres comptées durant l'exécution de l'opération est inférieur à $2(n - m + 1)$. D'où le résultat puisque $n-m+1$ comparaisons sont effectuées au minimum. ■

Heuristiques

Certains procédés élémentaires améliorent sensiblement le comportement global des algorithmes. On en détaille ici quelques-uns des plus significatifs. Ils sont décrits en rapport avec l'algorithme naïf. Mais la plupart des autres algorithmes peuvent les inclure dans leur code, l'adaptation s'avérant plus ou moins aisée. On parle d'heuristiques en cela que l'on n'est pas capable de mesurer vraiment leur apport.

Quitte à localiser toutes les occurrences du mot x dans le texte y par la méthode naïve, autant commencer par localiser les occurrences de sa première lettre, $x[0]$, dans le préfixe $y[0 \mathinner{.\,.} n-m+1]$ de y. Il restera ensuite à tester pour chaque apparition de $x[0]$ à une position j sur y l'identité éventuelle entre les deux mots $x[1 \mathinner{.\,.} m-1]$ et $y[j+1 \mathinner{.\,.} j+m-1]$. Comme l'opération de recherche de l'occurrence d'une lettre est généralement une opération de bas niveau des systèmes, la réduction du temps de calcul est souvent appréciable dans la pratique. Cette recherche élémentaire peut encore être améliorée de deux façons :

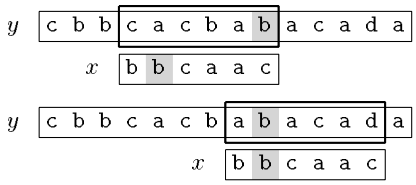
- en positionnant $x[0]$ comme sentinelle à la suite du texte y, de manière à avoir à tester le moins possible la fin du texte ;
- en cherchant non plus forcément $x[0]$, mais la lettre de x qui a la plus faible fréquence d'apparition dans les textes de la famille de y.

Il est à noter que la première technique présuppose qu'une telle altération de la mémoire est possible et qu'elle s'effectue en temps constant. Pour la seconde, outre la nécessité de posséder la fréquence des lettres de l'alphabet, le choix de la position de la lettre distinguée demande un calcul préalable sur x.

Un procédé d'ordre différent consiste à appliquer un décalage qui ne tienne compte que de la valeur de la lettre la plus à droite dans la fenêtre.

a	a	b	c	d
$\textit{dern-occ}[a]$	1	4	3	6

(a)



(b)

Figure 1.11 Décalage de la fenêtre glissante à l'aide de la table de la dernière occurrence, *dern-occ*, lorsque $x = \texttt{bbcaac}$. **(a)** Les valeurs de la table *dern-occ* sur l'alphabet $A = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$. **(b)** La fenêtre sur le texte y est à la position droite 8. La lettre à cette position, à savoir $y[8] = \texttt{b}$, apparait à la position maximale $k = 1$ sur $x[0\,..\,|x|-2]$. Un décalage valide consiste à faire glisser la fenêtre de $|x| - 1 - k = 4 = \textit{dern-occ}[\texttt{b}]$ positions sur la droite.

Soit j la position droite de la fenêtre. Deux cas antagonistes peuvent être envisagés selon que la lettre $y[j]$ apparait ou non dans $x[0\,..\,m-2]$:

- dans le cas où $y[j]$ n'apparait pas dans $x[0\,..\,m-2]$, le mot x ne peut apparaitre des positions droites $j+1$ à $j+m-1$ sur y ;
- dans l'autre cas, si k est la position maximale de l'occurrence de la lettre $y[j]$ sur $x[0\,..\,m-2]$, le mot x ne peut apparaitre des positions droites $j+1$ à $j+m-1-k-1$ sur y.

D'où des décalages valides à appliquer dans les deux cas : m pour le premier ; et $m-1-k$ pour le second. Remarquons qu'ils ne dépendent que de la lettre $y[j]$ et en aucune manière de sa position j sur y.

Pour formaliser l'observation précédente, on introduit la table

$$\textit{dern-occ}\colon A \to \{1, 2, \ldots, m\}$$

définie pour toute lettre $a \in A$ par

$$\textit{dern-occ}[a] = \min(\{m\} \cup \{m-1-k : 0 \le k \le m-2 \text{ et } x[k] = a\}) \ .$$

On appelle *dern-occ* la ***table de la dernière occurrence***. Elle exprime un décalage valide, $\textit{dern-occ}[y[j]]$, à appliquer après la tentative à la position droite j sur y. Une illustration est proposée figure 1.11. Suit le code du calcul de *dern-occ*. Il s'exécute en temps $\Theta(m + \text{card}\, A)$.

Dernière-occurrence(x, m)

```
pour chaque lettre a ∈ A faire
        dern-occ[a] ← m
pour k ← 0 à m − 2 faire
        dern-occ[x[k]] ← m − 1 − k
retourner dern-occ
```

On donne maintenant le code complet de l'algorithme Localiser-rapidement obtenu de celui de l'algorithme naïf par adjonction de la table *dern-occ*.

```
Localiser-rapidement(x, m, y, n)
1  dern-occ ← Dernière-occurrence(x, m)
2  j ← m − 1
3  tantque j < n faire
4       Signaler-si(y[j − m + 1 . . j] = x)
5       j ← j + dern-occ[y[j]]
```

Si la comparaison des mots à la ligne 4 débute en position $m-1$, la phase de recherche de l'algorithme Localiser-rapidement s'exécute en temps $\Theta(n/m)$ dans le meilleur des cas. Comme par exemple lorsqu'aucune des lettres aux positions congrues modulo m à $m-1$ sur y n'apparait dans x ; dans ce cas, une seule comparaison entre lettres est effectuée lors de chaque tentative[6] et le décalage est constamment égal à m. Le comportement de l'algorithme sur des textes en langage naturel est très bon. On peut montrer toutefois que dans le cas moyen (sous la double hypothèse de la proposition 1.17 et pour l'ensemble des mots de même longueur), le nombre de comparaisons par lettre de texte est asymptotiquement minoré par $1/\operatorname{card} A$, laquelle borne est indépendante de la longueur du mot cherché.

Machine de recherche

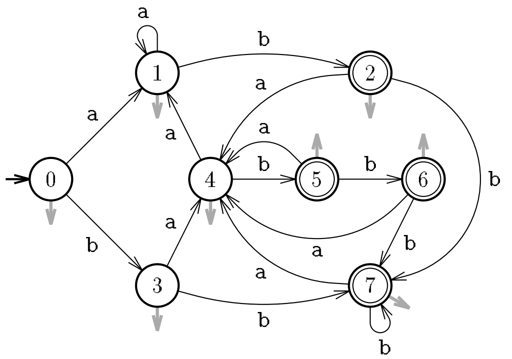
Certains automates peuvent servir de ***machine de recherche*** pour le traitement séquentiel de texte. On décrit dans cette partie deux algorithmes à base d'automate pour effectuer la localisation de motifs. On suppose les automates donnés ; le chapitre 2 présente la construction de certains de ces automates.

Considérons un motif $X \subseteq A^*$ et un automate déterministe M qui reconnait le langage A^*X (la figure 1.12(a) fournit un exemple). L'automate M reconnait les mots qui ont comme suffixe un mot de X. Pour localiser les mots de X qui apparaissent dans un texte y, il suffit de faire opérer l'automate M sur le texte y. Lorsque l'état courant est terminal, cela signifie que le préfixe courant de y – la partie de y déjà analysée par l'automate – appartient à A^*X ; ou, autrement dit, que la position courante sur y est la position droite d'une occurrence d'un mot de X. Cette remarque conduit à l'algorithme de localisation avec automate dont le code suit. Une illustration du fonctionnement de l'algorithme est présentée figure 1.12(b).

---

6. Remarquons qu'il s'agit là du meilleur des cas possibles pour un algorithme de localisation d'un mot de longueur m dans un texte de longueur n ; au moins $\lfloor n/m \rfloor$ lettres du texte doivent être inspectées pour conclure à la non-apparition du mot cherché dans le texte.

(a)



(b)

j	$y[j]$	état r	
		0	
0	c	0	
1	b	3	
2	a	4	
3	b	5	occurrence de ab
4	b	6	occurrences de babb et bb
5	a	4	

Figure 1.12 Localisation des occurrences d'un motif à l'aide d'un automate déterministe (à rapprocher de la figure 1.13). **(a)** Avec l'alphabet $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ et le motif $X = \{\mathtt{ab}, \mathtt{babb}, \mathtt{bb}\}$, l'automate déterministe représenté ci-dessus reconnait le langage A^*X. Les flèches grisées issues de chacun des états figurent les flèches ayant pour source ces mêmes états, pour cible l'état initial 0, et pour étiquette une de celles non déjà représentées. Pour localiser les occurrences des mots de X dans un texte y, il suffit de faire opérer l'automate sur y et d'indiquer chaque fois qu'un état terminal est atteint. **(b)** Exemple d'analyse avec $y =$ cbabba. De l'utilisation de l'automate, il résulte qu'il y a au moins une occurrence d'un mot de X aux positions 3 et 4 sur y, et aucune aux autres positions.

LA-DÉTERMINISTE(M, y)
1 $r \leftarrow$ *initial*$[M]$
2 **pour** chaque lettre a de y, séquentiellement **faire**
3 $\quad r \leftarrow$ CIBLE(r, a)
4 $\quad$ SIGNALER-SI(*terminal*$[r]$)

Proposition 1.18
Lorsque M est un automate déterministe qui reconnait le langage A^*X pour un motif $X \subseteq A^*$, l'opération LA-DÉTERMINISTE(M, y) *localise toutes les occurrences des mots de X dans le texte $y \in A^*$.*

Preuve Soit δ la fonction de transition de l'automate M. Comme l'automate est déterministe, il vient immédiatement que l'assertion

$$r = \delta(\mathit{initial}[M], u) \ , \tag{1.3}$$

où u est le préfixe courant de y, est satisfaite à la suite de l'exécution de chacune des instructions de l'algorithme.

Si une occurrence d'un mot de X se termine à la position courante, le préfixe courant u appartient à A^*X. Et donc, par définition de M et d'après la propriété (1.3), l'état courant r est terminal. Comme l'état initial n'est pas terminal (car $\varepsilon \notin X$), il s'en déduit que l'opération signale cette occurrence.

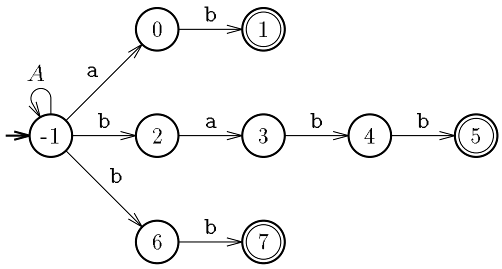
Réciproquement, supposons qu'une occurrence vienne d'être signalée. L'état courant r est donc terminal, ce qui, d'après la propriété (1.3) et par définition de M, implique que le préfixe courant u appartient à A^*X. Une occurrence d'un mot de X se termine donc à la position courante, ce qui achève la preuve du résultat annoncé. ■

Le temps d'exécution et l'espace supplémentaire nécessaire au fonctionnement de l'algorithme LA-DÉTERMINISTE dépendent uniquement de l'implantation de l'automate M. Par exemple, dans une implantation par matrice de transition, le temps d'analyse du texte est $\Theta(|y|)$, car le délai est constant, et l'espace supplémentaire, en plus de la matrice, est constant (voir proposition 1.15).

Le second algorithme de cette partie s'applique lorsque l'on dispose d'un automate N reconnaissant le langage X lui-même, et non plus A^*X. En ajoutant à l'automate une flèche de son état initial vers lui-même étiquetée par a, pour chaque lettre $a \in A$, on obtient simplement un automate N' qui reconnait le langage A^*X. Mais l'automate N' n'est pas déterministe, ce qui empêche d'appliquer l'algorithme précédent. La figure 1.13(a) présente un exemple d'automate N' pour le même motif X que celui de la figure 1.12(a).

Dans une telle situation, la solution retenue habituellement consiste à simuler l'automate obtenu par déterminisation de N', en suivant en parallèle tous les chemins possibles d'étiquette donnée. Puisque seuls les états qui sont les fins des chemins permettent d'effectuer le test d'occurrence, on se contente de conserver l'ensemble R des états atteints. C'est ce que réalise l'algorithme LA-NON-DÉTERMINISTE ci-dessous. En réalité, il n'est même pas nécessaire de modifier l'automate N car les boucles sur son état initial peuvent également être simulées. Cela est réalisé à la ligne 4 de l'algorithme par l'ajout systématique de l'état initial à l'ensemble d'états. Durant l'exécution de l'automate sur l'entrée y, l'automate n'est pas dans un état donné, mais dans un ensemble d'états, R. Ce sous-ensemble de l'ensemble d'états est recalculé après l'analyse de la lettre courante de y. L'algorithme fait appel à la fonction CIBLES qui effectue une transition sur un ensemble d'états, laquelle fonction est une extension immédiate de CIBLE.

(a)



(b)

j	$y[j]$	ensemble d'états R	
		$\{-1\}$	
0	c	$\{-1\}$	
1	b	$\{-1, 2, 6\}$	
2	a	$\{-1, 0, 3\}$	
3	b	$\{-1, 1, 2, 4, 6\}$	occurrence de ab
4	b	$\{-1, 2, 5, 6, 7\}$	occurrences de babb et bb
5	a	$\{-1, 0, 3\}$	

Figure 1.13 Localisation des occurrences d'un motif à l'aide d'un automate non déterministe (à rapprocher de la figure 1.12). **(a)** L'automate non déterministe représenté reconnait le langage A^*X, avec l'alphabet $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ et le motif $X = \{\mathtt{ab}, \mathtt{babb}, \mathtt{bb}\}$. Pour localiser les occurrences des mots de X qui apparaissent dans un texte y, il suffit de faire opérer l'automate sur y et de signaler une occurrence à chaque fois qu'un état terminal est atteint. **(b)** Un exemple de simulation lorsque $y = \mathtt{cbabba}$. Le calcul revient à suivre simultanément tous les chemins possibles. Il résulte que le motif apparait aux positions droites 3 et 4 et nulle part ailleurs.

LA-NON-DÉTERMINISTE(N, y)

1 $q_0 \leftarrow$ *initial*$[N]$
2 $R \leftarrow \{q_0\}$
3 **pour** chaque lettre a de y, séquentiellement **faire**
4     $R \leftarrow$ CIBLES$(R, a) \cup \{q_0\}$
5     $t \leftarrow$ FAUX
6   **pour** chaque état $p \in R$ **faire**
7   **si** *terminal*$[p]$ **alors**
8             $t \leftarrow$ VRAI
9   SIGNALER-SI(t)

CIBLES(R, a)

$S \leftarrow \emptyset$
pour chaque état $p \in R$ **faire**
    pour chaque état q tel que $(a, q) \in Succ[p]$ **faire**
        $S \leftarrow S \cup \{q\}$
retourner S

Les instructions aux lignes 5–8 de l'algorithme LA-NON-DÉTERMINISTE donnent à la variable booléenne t la valeur VRAI lorsque l'intersection entre l'ensemble d'états R et l'ensemble des états terminaux est non vide. Une occurrence est ensuite signalée, ligne 9, le cas échéant. La figure 1.13(b) illustre le fonctionnement de l'algorithme.

Proposition 1.19
Lorsque N est un automate qui reconnait le langage X pour un motif $X \subseteq A^*$, l'opération LA-NON-DÉTERMINISTE(N, y) *localise toutes les occurrences des mots de X dans le texte $y \in A^*$.*

Preuve Notons q_0 l'état initial de l'automate N et, pour tout mot $v \in A^*$, R_v l'ensemble d'états défini par

$$R_v = \{q : q \text{ fin d'un chemin d'origine } q_0 \text{ et d'étiquette } v\} \ .$$

On peut vérifier, par récurrence sur la longueur des préfixes de y, que l'assertion

$$R = \bigcup_{v \preceq_{\mathit{suff}} u} R_v \ , \tag{1.4}$$

où u est le préfixe courant de y, est satisfaite à la suite de l'exécution de chacune des instructions de l'algorithme, hormis celle à la ligne 1.

Si une occurrence d'un mot de X se termine à la position courante, l'un des suffixes v du préfixe courant u appartient à X. Il vient alors, par définition de N, que l'un des états $q \in R_v$ est terminal, puis, d'après la propriété (1.4), que l'un des états de R est terminal. Il s'en déduit que l'opération signale cette occurrence puisqu'aucun des mots de X n'est vide.

Réciproquement, si une occurrence vient d'être signalée, c'est que l'un des états $q \in R$ est terminal. La propriété (1.4) et la définition de N impliquent alors l'existence d'un suffixe v du préfixe courant u qui appartient à X. Il s'ensuit qu'une occurrence d'un mot de X se termine à la position courante. Ce qui achève la preuve de la proposition. ■

La complexité de l'algorithme LA-NON-DÉTERMINISTE dépend à la fois de l'implantation retenue pour l'automate N et de la réalisation choisie pour manipuler les ensembles d'états. Si, par exemple, l'automate est déterministe, que sa fonction de transition est implantée par matrice de transition, et que les ensembles d'états sont implantés par vecteurs booléens dont les indices sont les états, la fonction CIBLES s'exécute en temps et en espace $O(\operatorname{card} Q)$, où Q est l'ensemble des états. Dans ce cas, l'analyse du texte y a lieu en temps $O(|y| \times \operatorname{card} Q)$ et utilise un espace supplémentaire $O(\operatorname{card} Q)$.

Dans les paragraphes qui suivent, on considère un exemple de réalisation de la simulation ci-dessus adapté au cas d'un très petit automate qui possède une structure arborescente.

Modèle vecteur-binaire

Le ***modèle vecteur-binaire*** fait référence à la possibilité d'utiliser les mots machine pour coder les états des automates de localisation. Lorsque la longueur du langage associé au motif à localiser n'est pas plus grande que la taille d'un mot machine comptée en nombre de bits, cette technique donne des algorithmes efficaces faciles à implanter. La technique est notamment employée section 8.4.

Ici, le principe reprend la méthode de simulation d'automate déterministe des paragraphes précédents en codant l'ensemble des états atteints par un vecteur binaire, et en réalisant le changement d'état par simple décalage contrôlé par un masque associé à la lettre considérée.

Commençons par préciser les notations utilisées dans la suite pour les vecteurs binaires. On assimile un vecteur binaire à un mot sur l'alphabet $\{\mathtt{0}, \mathtt{1}\}$. On note respectivement $\vee$ et $\wedge$ le « ou » et le « et » bit à bit. Ce sont des opérations binaires internes aux ensembles de vecteurs binaires de longueurs identiques qui, pour la première, met à 1 les bits du résultat si l'un des deux bits de même position des deux opérandes est égal à 1, et à 0 sinon, et, pour la seconde, met à 1 les bits du résultat si les deux bits de même position des deux opérandes sont égaux à 1, et à 0 sinon. On note encore $\dashv$ l'opération externe qui à un naturel k et un vecteur binaire fait correspondre le vecteur binaire de même longueur obtenu du premier en décalant les bits vers la droite de k positions et en complétant à gauche par des 0. Ainsi, $\mathtt{1001} \vee \mathtt{0011} = \mathtt{1011}$, $\mathtt{1001} \wedge \mathtt{0011} = \mathtt{0001}$, et $2 \dashv \mathtt{1101} = \mathtt{0011}$.

Considérons un ensemble X fini non vide de mots tous non vides. Soit N l'automate obtenu à partir des $\operatorname{card} X$ automates déterministes élémentaires qui reconnaissent les mots de X en fusionnant les états initiaux en un seul, disons q_0. Soit N' l'automate construit sur N en ajoutant les flèches de la forme (q_0, a, q_0), pour chaque lettre $a \in A$. L'automate N' reconnait le langage A^*X. La recherche des occurrences de mots de X dans un texte y est réalisée ici comme dans les paragraphes ci-dessus en simulant l'automate déterminisé de N' au moyen de N (voir figure 1.13(a)).

Posons $m = |X|$ et numérotons les états de N depuis -1 jusqu'à $m-1$ en utilisant un parcours en profondeur à partir de l'état initial q_0 – c'est le cas dans l'exemple de la figure 1.13(a). Codons maintenant chaque ensemble d'états $R \setminus \{-1\}$ par un vecteur r de m bits avec la convention suivante :

$$p \in R \setminus \{-1\} \text{ si et seulement si } r[p] = \mathtt{1} \ .$$

Soient r le vecteur de m bits qui code l'état courant de la recherche, $a \in A$ la lettre courante de y, et s le vecteur de m bits qui code l'état suivant. Il est clair que le calcul de s à partir de r et de a observe la règle suivante : $s[p] = \mathtt{1}$ si et seulement si il existe une flèche d'étiquette a, soit de l'état -1 vers l'état p, soit de l'état $p-1$ à l'état p avec $r[p-1] = \mathtt{1}$.

Considérons *init* le vecteur de m bits défini par $\mathit{init}[p] = \mathtt{1}$ si et seulement si il existe une flèche de source l'état -1 et de cible l'état p. Considérons aussi la table *masq* indicée sur A et à valeurs dans l'ensemble des vecteurs de m bits, définie pour toute lettre $b \in A$ par $\mathit{masq}[b][p] = \mathtt{1}$ si et seulement si il existe une flèche d'étiquette b et de cible l'état p. Alors r, a et s satisfont l'identité :

$$s = (\mathit{init} \vee (1 \dashv r)) \wedge \mathit{masq}[a] \ .$$

Celle-ci traduit en termes d'opérations bit à bit la transition effectuée ligne 4 de l'algorithme LA-NON-DÉTERMINISTE, hors l'état initial. Le vecteur binaire *init* code les transitions potentielles à partir de l'état initial, et le décalage d'un bit vers la droite celles à partir des états atteints. La table *masq* valide les transitions étiquetées par la lettre courante.

Il ne reste plus maintenant qu'à indiquer comment tester si l'un des états représentés par un vecteur r de m bits qui code l'état courant de la recherche est terminal ou non. Dans ce but, soit *term* le vecteur de m bits défini par $\mathit{term}[p] = \mathtt{1}$ si et seulement si l'état p est terminal. Alors l'un des états représenté par r est terminal si et seulement si :

$$r \wedge \mathit{term} \neq \mathtt{0}^m \ .$$

Suivent le code de la fonction PETIT-AUTOMATE qui calcule les vecteurs *init* et *term* ainsi que les valeurs de la table *masq*, puis celui de l'algorithme de localisation.

```
PETIT-AUTOMATE(X, m)
init ← 0^m
term ← 0^m
pour chaque lettre a ∈ A faire
        masq[a] ← 0^m
p ← −1
pour chaque mot x ∈ X faire
        init[p + 1] ← 1
        pour chaque lettre a de x, séquentiellement faire
                p ← p + 1
                masq[a][p] ← 1
        term[p] ← 1
retourner (init, term, masq)
```

```
LOCALISER-MOTS-COURTS(X, m, y)
(init, term, masq) ← PETIT-AUTOMATE(X, m)
r ← 0^m
pour chaque lettre a de y, séquentiellement faire
        r ← (init ∨ (1 ⊣ r)) ∧ masq[a]
        SIGNALER-SI(r ∧ term ≠ 0^m)
```

(a)

k	0	1	2	3	4	5	6	7
$init[k]$	1	0	1	0	0	0	1	0
$term[k]$	0	1	0	0	0	1	0	1
$masq[\mathsf{a}][k]$	1	0	0	1	0	0	0	0
$masq[\mathsf{b}][k]$	0	1	1	0	1	1	1	1
$masq[\mathsf{c}][k]$	0	0	0	0	0	0	0	0

(b)

j	$y[j]$	vecteur binaire r	
		00000000	
0	c	00000000	
1	b	00100010	
2	a	10010000	
3	b	01101010	occurrence de ab
4	b	00100111	occurrences de babb et bb
5	a	10010000	

Figure 1.14 Utilisation de vecteurs binaires pour la recherche des occurrences du motif $X = \{\mathsf{ab}, \mathsf{babb}, \mathsf{bb}\}$ (voir figure 1.13). **(a)** Les vecteurs *init* et *term*, et les valeurs de la table de vecteurs *masq* sur l'alphabet $A = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Ces vecteurs sont de longueur 8 car $|X| = 8$. Le premier vecteur code les transitions potentielles à partir de l'état initial. Le deuxième code les états terminaux. Les vecteurs de la table *masq* codent les occurrences d'une lettre de l'alphabet dans les mots de X. **(b)** Les valeurs successives du vecteur r qui code l'état courant de la recherche des occurrences des mots de X dans le texte $y = \mathsf{cbabba}$. La zone grisée qui marque certains bits indique qu'un état terminal vient d'être atteint.

Un exemple de calcul est traité figure 1.14.

Proposition 1.20
Pour s'exécuter, l'opération Localiser-mots-courts(X, m, y) *nécessite un temps* $\Theta(m \times \text{card}\, A + m \times |y|)$*. L'espace mémoire supplémentaire requis est* $\Theta(m \times \text{card}\, A)$*.*

Preuve Le temps nécessaire à l'initialisation des vecteurs binaires *init*, *term* et $masq[a]$ pour $a \in A$, est linéaire en leur taille, soit $\Theta(m \times \text{card}\, A)$. Les instructions aux lignes 4 et 5 s'exécutent chacune en temps $\Theta(m)$. Les complexités annoncées s'en déduisent. ■

Cela établi, lorsque la longueur m est inférieure au nombre de bits d'un mot machine, tout vecteur binaire de m bits peut s'implanter à l'aide d'un mot machine dont seuls les m premiers bits sont significatifs. Ce qui donne le résultat suivant.

Corollaire 1.21
Lorsque m est inférieur à la longueur d'un mot machine, l'opération Localiser-mots-courts(X, m, y) *s'exécute en temps* $\Theta(|y| + \text{card}\, A)$ *avec un espace mémoire supplémentaire* $\Theta(\text{card}\, A)$*.* ■

1.6 Base de techniques élaborées

On présente dans cette section deux méthodes fondamentales des algorithmes efficaces de localisation de motifs ou de recherche de régularités dans les mots. Il s'agit de deux tables, la table des bords et la table des préfixes, qui toutes deux mémorisent les occurrences des préfixes d'un mot qui apparaissent au sein de lui-même. Les tables peuvent être calculées en temps linéaire. Les algorithmes de calcul fournissent aussi des méthodes de localisation de mots qui sont étudiées en détail dans les chapitres 2 et 3 (un prélude est proposé exercice 1.24).

Table des bords

Soit x un mot de longueur $m \geq 1$. On définit la table

$bord\colon \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$

par

$bord[k] = |Bord(x[0\,..\,k])|$

pour $k = 0, 1, \ldots, m-1$. On dit de la table $bord$ qu'elle est la ***table des bords*** pour le mot x, sous-entendant qu'il s'agit des bords des préfixes non vides du mot. Voici en exemple la table des bords pour le mot x lorsque $x =$ `abbabaabbabaaaabbabbaa`:

k	0	1	2	3	4	5	6	7	8	9	10	11
$x[k]$	a	b	b	a	b	a	a	b	b	a	b	a
$bord[k]$	0	0	0	1	2	1	1	2	3	4	5	6

k	12	13	14	15	16	17	18	19	20	21
$x[k]$	a	a	a	b	b	a	b	b	a	a
$bord[k]$	7	1	1	2	3	4	5	3	4	1

Le lemme suivant fournit la relation de récurrence utilisée par la fonction Bords, donnée ensuite, pour effectuer le calcul de la table $bord$.

Lemme 1.22
Pour tout $(u, a) \in A^+ \times A$, on a

$$Bord(ua) = \begin{cases} Bord(u)a & \text{si } Bord(u)a \preceq_{\text{préf}} u \ , \\ Bord(Bord(u)a) & \text{sinon} \ . \end{cases}$$

Preuve On remarque pour commencer que si $Bord(ua)$ est un mot non vide, il est de la forme wa où w est un bord de u.

Si $Bord(u)a \preceq_{\text{préf}} u$, le mot $Bord(u)a$ est alors un bord de ua, et la remarque précédente montre qu'il s'agit du plus long mot de la sorte. Il s'ensuit que $Bord(ua) = Bord(u)a$ dans ce cas.

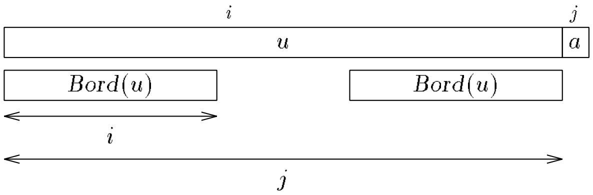


Figure 1.15 Schéma montrant la correspondance entre les variables i et j considérées en ligne 3 de la fonction Bords et l'énoncé du lemme 1.22.

Dans le cas contraire, $Bord(ua)$ est à la fois un préfixe de $Bord(u)$ et un suffixe de $Bord(u)a$. Comme il est de longueur maximale avec cette propriété, c'est bien le mot $Bord(Bord(u)a)$. ∎

La figure 1.15 schématise la correspondance entre les variables i et j de la fonction dont le code suit et l'énoncé du lemme.

```
Bords(x, m)
 1  i ← 0
 2  pour j ← 1 à m − 1 faire
 3      bord[j − 1] ← i
 4      tantque i ≥ 0 et x[j] ≠ x[i] faire
 5          si i = 0 alors
 6              i ← −1
 7          sinon i ← bord[i − 1]
 8      i ← i + 1
 9  bord[m − 1] ← i
10  retourner bord
```

Proposition 1.23
La fonction Bords *appliquée à un mot x et sa longueur m produit la table des bords pour x.*

Preuve La table *bord* est initialisée séquentiellement, du préfixe de x de longueur 1 à x lui-même. Lors de l'exécution de la boucle **tantque** aux lignes 4–7 est inspectée la suite des bords de $x[0\,.\,.\,j-1]$, d'après la proposition 1.5. À la sortie de cette boucle, on a $|Bord(x[0\,.\,.\,j])| = |x[0\,.\,.\,i]| = i+1$, conformément au lemme 1.22. La correction du code s'ensuit. ∎

Proposition 1.24
L'opération Bords(x, m) *s'exécute en temps $\Theta(m)$. Le nombre de comparaisons entre les lettres du mot x est compris entre $m-1$ et $2m-3$ lorsque $m \geq 2$.*

On convient dans toute la suite de qualifier la comparaison de deux lettres données de ***positive*** lorsque ces deux lettres sont identiques, et de ***négative*** dans le cas contraire.

Preuve Remarquons que le temps d'exécution est linéaire en le nombre de comparaisons effectuées entre les lettres de x. Il suffit donc d'établir la borne sur le nombre de comparaisons.

La quantité $2j - i$ croît d'au moins une unité après chaque comparaison de lettres : les variables i et j sont incrémentées à la suite d'une comparaison positive ; la valeur de i est diminuée d'au moins une unité et celle de j reste inchangée à la suite d'une comparaison négative. Lorsque $m \geq 2$, cette quantité vaut 2 à la première comparaison ($i = 0$ et $j = 1$) et au plus $2m - 2$ lors de la dernière ($i \geq 0$ et $j = m - 1$). Le nombre total de comparaisons est donc bien majoré par $2m - 3$ comme annoncé.

La borne de $2m - 3$ comparaisons est précise : elle est atteinte pour tout mot x de la forme $a^{m-1}b$ avec $a, b \in A$ et $a \neq b$. Ce qui achève la preuve. ■

Une autre preuve de la borne $2m - 3$ est proposée exercice 1.22.

Table des préfixes

Soit x un mot de longueur $m \geq 1$. On définit la table

$$\mathit{préf}\colon \{0, 1, \ldots, m-1\} \to \{0, 1, \ldots, m-1\}$$

par

$$\mathit{préf}[k] = |\mathit{lpc}(x, x[k \mathinner{..} m-1])|$$

pour $k = 0, 1, \ldots, m-1$, où

$\mathit{lpc}(u, v) =$ le ***plus long préfixe commun*** aux mots u et v .

La table *préf* est appelée la ***table des préfixes*** pour le mot x. Elle mémorise les préfixes de x qui apparaissent au sein du mot lui-même. On note que $\mathit{préf}[0] = |x|$. L'exemple qui suit montre la table des préfixes pour x dans le cas où $x =$ `abbabaabbabaaaabbabbaa`.

k	0	1	2	3	4	5	6	7	8	9	10	11
$x[k]$	a	b	b	a	b	a	a	b	b	a	b	a
$\mathit{préf}[k]$	22	0	0	2	0	1	7	0	0	2	0	1

k	12	13	14	15	16	17	18	19	20	21
$x[k]$	a	a	a	b	b	a	b	b	a	a
$\mathit{préf}[k]$	1	1	5	0	0	4	0	0	1	1

Certains algorithmes de recherche de mots (voir chapitre 3) utilisent la table *suff* qui n'est autre que l'analogue de la table des préfixes obtenue en considérant le renversé du mot x.

La méthode de calcul de *préf* présentée ci-dessous procède en déterminant $préf[i]$ par valeurs croissantes de la position i sur x. Une méthode naïve consisterait à évaluer chaque valeur $préf[i]$ indépendamment des valeurs précédentes par comparaisons directes ; mais on aboutirait alors à un calcul quadratique, dans le cas où x est la puissance d'une seule lettre, par exemple. L'utilisation de valeurs déjà calculées permet d'obtenir un algorithme linéaire. Pour cela on introduit, l'indice i étant fixé, deux valeurs g et f qui constituent les éléments clés de la méthode. Elles satisfont les relations

$$g = \max\{j + préf[j] : 0 < j < i\} \tag{1.5}$$

et

$$f \in \{j : 0 < j < i \text{ et } j + préf[j] = g\} \ . \tag{1.6}$$

On remarque que g et f sont définies quand $i > 1$. Le mot $x[f \mathinner{..} g-1]$ est ainsi un préfixe de x, soit aussi un bord de $x[0 \mathinner{..} g-1]$. C'est le mot vide lorsque $f = g$. On peut noter de plus que si $g < i$ on a alors $g = i - 1$, et que dans le cas contraire, par définition de f, on a $f < i \leq g$.

Le lemme qui suit fournit la justification du principe de fonctionnement de la fonction PRÉFIXES.

Lemme 1.25
Si $i < g$, on a la relation

$$préf[i] = \begin{cases} préf[i-f] & \text{si } préf[i-f] < g - i \ , \\ g - i + \ell & \text{sinon} \ , \end{cases}$$

où $\ell = |lpc(x[g-i \mathinner{..} m-1], x[g \mathinner{..} m-1])|$.

Preuve Posons $u = x[f \mathinner{..} g-1]$. Le mot u est un préfixe de x par définition de f et g. Posons aussi $k = préf[i-f]$. Par définition de *préf*, le mot $x[i-f \mathinner{..} i-f+k-1]$ est un préfixe de x mais pas $x[i-f \mathinner{..} i-f+k]$.

Dans le cas où $préf[i-f] < g-i$, une occurrence de $x[i-f \mathinner{..} i-f+k]$ débute à la position $i-f$ sur u – donc également à la position i sur x – ce qui montre que $x[i-f \mathinner{..} i-f+k-1]$ est le plus long préfixe de x débutant à la position i. On a ainsi $préf[i] = k = préf[i-f]$.

Dans le cas contraire, le mot $x[i \mathinner{..} g-1]$, qui est un suffixe de u, est un préfixe de $x[i-f \mathinner{..} i-f+k-1]$ donc de x. On en déduit immédiatement que $préf[i] = g - i + \ell$. ∎

Dans le calcul de *préf*, on initialise la variable g à 0 pour simplifier l'écriture du code de la fonction PRÉFIXES, et on laisse f initialement non défini. La première étape du calcul consiste ainsi à déterminer $préf[1]$ par

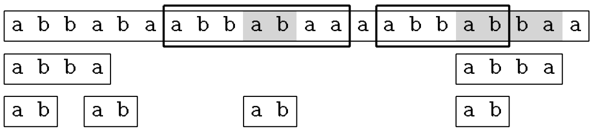


Figure 1.16 Illustration du principe de fonctionnement de la fonction Préfixes. Les facteurs encadrés $x[6\,..\,12]$ et $x[14\,..\,18]$ et les facteurs grisés $x[9\,..\,10]$ et $x[17\,..\,20]$ sont des préfixes du mot $x = \mathsf{abbabaabbabaaaabbabbaa}$. Pour $i = 9$, on a $f = 6$ et $g = 13$. La situation à cette position est la même qu'à la position $3 = 9 - 6$. On a $\mathit{préf}[9] = \mathit{préf}[3] = 2$ qui signifie que ab, de longueur 2, est le plus long facteur de position 9 qui est un préfixe de x. Pour $i = 17$, on a $f = 14$ et $g = 19$. Comme $\mathit{préf}[17 - 14] = 2 \geq 19 - 17$, on en déduit que le mot $\mathsf{ab} = x[i\,..\,g-1]$ est un préfixe de x. Il faut comparer x et $x[i\,..\,m-1]$ à partir des positions respectives 2 et g pour déterminer $\mathit{préf}[i] = 4$.

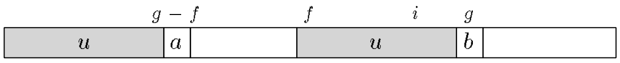


Figure 1.17 Variables i, f et g de la fonction Préfixes. La boucle principale admet pour invariants : $u = \mathit{lpc}(x, x[f\,..\,m-1])$ et donc $a \neq b$ avec $a, b \in A$, ainsi que $f < i$ lorsque f est définie. Le schéma correspond à la situation dans laquelle $i < g$.

comparaisons de lettres. L'utilité du résultat énoncé ci-dessus n'apparait au mieux qu'à partir du calcul de la valeur suivante. Une illustration du principe de fonctionnement de la fonction est donné figure 1.16. Un schéma montrant la correspondance entre les variables de la fonction et les notations utilisées dans l'énoncé du lemme 1.25 et dans celui de sa preuve est donné figure 1.17.

Préfixes(x, m)
```
préf[0] ← m
g ← 0
pour i ← 1 à m − 1 faire
    si i < g et préf[i − f] < g − i alors
        préf[i] ← préf[i − f]
    sinon (g, f) ← (max{g, i}, i)
        tantque g < m et x[g] = x[g − f] faire
            g ← g + 1
        préf[i] ← g − f
retourner préf
```

Proposition 1.26
La fonction Préfixes *appliquée à un mot x et à sa longueur m produit la table des préfixes pour x.*

a	b	b	a	b	a	a	b	b	a	b	a	a	a	a	b	b	a	b	b	a	a

Figure 1.18 Bords et préfixes. Dans le mot $x =$ abbabaabbabaaaabbabbaa, *préf*$[9] = 2$ et *bord*$[9 + 2 - 1] = 5 \neq 2$. On a également *bord*$[15] = 2$ et *préf*$[15 - 2 + 1] = 5 \neq 2$.

Preuve On peut vérifier que les variables f et g satisfont les relations (1.5) et (1.6) à chaque étape de l'exécution de la boucle.

On constate ensuite que, pour i fixé satisfaisant la condition $i < g$, la fonction applique la relation de l'énoncé du lemme 1.25, ce qui produit un calcul correct. Il reste donc à vérifier que le calcul est correct lorsque $i \geq g$. Mais dans cette situation, les instructions aux lignes 6–8 calculent $|lpc(x, x[i \,..\, m-1])| = |x[f \,..\, g-1]| = g - f$ qui est, par définition même, la valeur de *préf*$[i]$.

La fonction produit donc bien la table *préf*. ∎

Proposition 1.27
L'exécution de l'opération Préfixes(x, m) *prend un temps* $\Theta(m)$. *Moins de* $2m$ *comparaisons entre les lettres du mot* x *sont effectuées.*

Preuve Les comparaisons de lettres sont effectuées à la ligne 7. Toute comparaison entre lettres différentes conduit à incrémenter la variable g. Comme la valeur de g ne diminue jamais et qu'elle varie de 0 à m au maximum, il y a au plus m comparaisons négatives. Chaque comparaison positive fait passer à l'étape suivante de la boucle. Il y en a donc au plus m également. Soit au plus $2m$ comparaisons au total.

Le raisonnement précédent montre aussi que le temps total de toutes les exécutions de la boucle aux lignes 7–8 est $\Theta(m)$. Les autres instructions de la boucle 3–9 prennent un temps constant pour chaque valeur de i donnant à nouveau un temps global $\Theta(m)$ pour leurs exécutions et celle de la fonction. ∎

La borne de $2m$ sur le nombre de comparaisons effectuées par la fonction Préfixes est relativement fine. Par exemple, on obtient $2m - 3$ comparaisons pour un mot de la forme $a^{m-1}b$ avec $m \geq 2$, $a, b \in A$ et $a \neq b$. En effet, il faut $m - 1$ comparaisons pour calculer *préf*$[1]$, puis une comparaison pour chacune des $m - 2$ valeurs *préf*$[i]$ avec $1 < i < m$.

Relations entre bords et préfixes

Les tables *bord* et *préf* dont le calcul est décrit ci-dessus mémorisent toutes deux les occurrences des préfixes de x. On explicite ici une relation entre ces deux tables.

La relation n'est pas immédiate pour la raison qui suit, illustrée par la figure 1.18. Quand *préf*$[i] = \ell$, le facteur $u = x[i \,..\, i+\ell-1]$ est un préfixe de x mais ce n'est pas nécessairement le bord de $x[0 \,..\, i + \ell - 1]$, celui-ci pouvant être plus long que u. De même, lorsque $bord[j] = \ell$, le facteur $v = x[j - \ell + 1 \,.\,.\, j]$ est un préfixe de x mais ce n'est pas nécessairement le plus long préfixe de x apparaissant à la position $j - \ell + 1$.

La proposition qui suit montre comment la table *bord* s'exprime en fonction de la table *préf*. On peut en déduire un algorithme de calcul de la table *bord* à partir de la table *préf*.

Proposition 1.28
Soient $x \in A^+$ et j une position sur x. Alors :

$$bord[j] = \begin{cases} 0 & si\ I = \emptyset\ , \\ j - \min I + 1 & sinon\ , \end{cases}$$

où $I = \{i : 0 \le i \le j \text{ et } i + préf[i] - 1 \ge j\}$.

Preuve On remarque pour commencer que, pour $0 \le i \le j$, $i \in I$ si et seulement si $x[i\,.\,.\,j] \preceq_{préf} x$. En effet, si $i \in I$, on a $x[i\,.\,.\,j] \preceq_{préf} x[i\,.\,.\,i + préf[i] - 1] \preceq_{préf} x$, donc $x[i\,.\,.\,j] \preceq_{préf} x$. Réciproquement, si $x[i\,.\,.\,j] \preceq_{préf} x$, on en déduit, par définition de $préf[i]$, $préf[i] \ge j - i + 1$. Et donc $i + préf[i] - 1 \ge j$. Ce qui montre que $i \in I$.

On remarque à la suite que $bord[j] = 0$ si et seulement si $I = \emptyset$.

Il s'ensuit que si $bord[j] \neq 0$ (soit $bord[j] > 0$) et $k = j - bord[j] + 1$, on a $k \le j$ et $x[k\,.\,.\,j] \preceq_{préf} x$. Aucun facteur $x[i\,.\,.\,j]$, $i < k$, ne satisfait la relation $x[i\,.\,.\,j] \preceq_{préf} x$ par définition de $bord[j]$. Donc $k = \min I$ par la première remarque, et $bord[j] = j - k + 1$ comme annoncé. ∎

Le calcul de la table *préf* à partir de la table *bord* peut conduire à une itération, et ne semble pas donner une expression simple, comparable à celle de l'énoncé précédent (voir exercice 1.23).

Notes

Le chapitre contient les éléments de base pour une étude précise des algorithmes sur les mots. La plupart des notions qui y sont introduites est dispersée dans différents ouvrages dont nous citons ici ceux qui sont souvent considérés comme des références dans leurs domaines.

Les aspects combinatoires sur les mots sont traités dans l'ouvrage collectif de Lothaire [56]. On peut se reporter au livre d'Aho, Hopcroft et Ullman [48] pour les questions d'algorithmique : expression des algorithmes, structures de données et évaluation de la complexité. Nous nous sommes inspirés du livre de Cormen, Leiserson et Rivest [53] pour la présentation générale et le style des algorithmes. Concernant les automates et langages, on peut consulter le livre de Berstel [52] ou celui de Pin [58]. Les ouvrages de Berstel et Perrin (1985) ainsi que celui de Béal [50] contiennent des éléments de la théorie des codes (exercices 1.10 et 1.11

entre autres). Enfin, l'ouvrage d'Aho, Sethi et Ullman [49] décrit des méthodes d'implantation des automates.

La section 1.5 sur les techniques de base contient des éléments fréquemment retenus pour le développement final de logiciels utilisant des algorithmes de traitement de chaines de caractères. Il s'agit en particulier des heuristiques et de l'utilisation de mots machine. Cette dernière technique est reprise dans le chapitre 8 pour la localisation de motifs approchés. Ce type de technique a été initiée par Baeza-Yates et Gonnet (1992) et par Wu et Manber (1992). L'algorithme LOCALISER-RAPIDEMENT est dû à Horspool (1980). La recherche d'un mot au moyen d'une fonction de hachage est analysée par Karp et Rabin (1987).

Le traitement des notions de la section 1.6 est original. Le calcul de la table des bords est classique. Il s'inspire d'un algorithme de Morris et Pratt de 1970 (voir [6]) qui est à l'origine du premier algorithme de localisation d'un mot fonctionnant en temps linéaire. La table des préfixes synthétise différemment les mêmes informations sur un mot que la table précédente. La notion duale de table des suffixes est utilisée dans le chapitre 3. Gusfield [3] en fait un élément fondamental des méthodes de localisation de mots (son algorithme Z correspond à l'algorithme SUFFIXES du chapitre 3).

Du point de vue terminologique, un mot, encore appelé « chaine de caractères », a comme équivalent anglais *word*, *string* ou *sequence*. Le terme de facteur, qui traduit la structure algébrique de A^*, admet pour synonyme « segment » et se traduit souvent (en anglais) par *subword* ou *substring*, mais aussi par *factor*. Enfin, un sous-mot, qui est une sous-suite, se traduit (en anglais) par *subsequence*.

Exercices

1.1 (Calcul)

Quel est le nombre de préfixes, de suffixes, de facteurs et de sous-mots d'un mot donné? Discuter si nécessaire.

1.2 (Morphisme de Fibonacci)

Un ***morphisme*** f sur A^* est une application de A^* dans A^* qui satisfait les règles :

$$\begin{aligned} f(\varepsilon) &= \varepsilon \ , \\ f(x \cdot y) &= f(x) \cdot f(y) \qquad \text{pour } x, y \in A^* \ . \end{aligned}$$

Pour tout naturel n et tout mot $x \in A^*$, on note $f^n(x)$ le mot défini par $f^0(x) = \varepsilon$ et $f^k(x) = f^{k-1}(f(x))$ pour $k = 1, 2, \ldots, n$.

Considérons l'alphabet $A = \{\mathtt{a}, \mathtt{b}\}$. Soit φ le morphisme sur A^* défini par $\varphi(\mathtt{a}) = \mathtt{ab}$ et $\varphi(\mathtt{b}) = \mathtt{a}$. Montrer que le mot $\varphi^n(\mathtt{a})$ est identique à F_{n+2}, le mot de Fibonacci d'indice $n + 2$.

1.3 (Permutation)
On appelle permutation sur l'alphabet A un mot u qui satisfait la condition card $alph(u) = |u| = \text{card}\, A$. C'est donc un mot dans lequel toutes les lettres de l'alphabet apparaissent une fois et une seule.

Pour $k = \text{card}\, A$, montrer qu'il existe un mot de longueur inférieure à $k^2 - 2k + 4$ qui possède comme sous-mots toutes les permutations sur A. Écrire un algorithme de construction d'un tel mot. [*Aide :* voir Mohanty (1980).]

1.4 (Période)
Montrer que la condition 3 de la proposition 1.4 peut être remplacée par la condition suivante : il existe un mot t et un entier $k > 0$ tels que $x \preceq_{fact} t^k$ et $|t| = p$.

1.5 (Cas limite)
Montrer que le mot $(\mathtt{ab})^k\mathtt{a}(\mathtt{ab})^k\mathtt{a}$ avec $k \geq 1$ est cas limite pour le lemme de périodicité.

1.6 (Trois périodes)
Sur les triplets d'entiers positifs (p_1, p_2, p_3) classés, $p_1 \leq p_2 \leq p_3$, on définit l'opération de dérivation par : le dérivé de (p_1, p_2, p_3) est le triplet classé constitué des entiers p_1, $p_2 - p_1$ et $p_3 - p_1$. Soit (q_1, q_2, q_3) le premier triplet obtenu en itérant la dérivation à partir de (p_1, p_2, p_3) tel que $q_1 = 0$.

Montrer que si le mot $x \in A^*$ possède p_1, p_2 et p_3 comme périodes et que

$$|x| \geq \frac{1}{2}(p_1 + p_2 + p_3 - 2\,\text{pgcd}(p_1, p_2, p_3) + q_2 + q_3) \ ,$$

alors il possède aussi $\text{pgcd}(p_1, p_2, p_3)$ comme période. [*Aide :* voir Mignosi et Restivo [57].]

1.7 (Trois carrés)
Soient trois mots non vides u, v et w. Montrer que si l'on suppose que u est primitif et que $u^2 \prec_{préf} v^2 \prec_{préf} w^2$, on a $2|u| < |w|$ (voir proposition 9.17 pour une conséquence plus précise).

1.8 (Conjugués)
Montrer que deux mots conjugués non vides ont même exposant et des racines conjuguées.

Montrer que la classe de conjugaison de tout mot non vide x contient $|x|/k$ éléments où k est l'exposant de x.

1.9 (Périodes)
Soit p une période de x qui n'est pas multiple de $pér(x)$. Montrer que $p > |x| - pér(x)$.

Soient p et q deux périodes de x telles que $p < q$. Montrer que :

– $q - p$ est une période de $prem_{|x|-p}(x)$ et de $(prem_p(x))^{-1}x$;
– p et $q + p$ sont des périodes de $prem_q(x)x$.

(La définition de $prem_k$ est donnée en section 4.4.)

Montrer que si $x = uvw$, uv et vw ont comme période p et $|v| \geq p$, alors x a période p.

Supposons que x ait une période p et possède un facteur v de période r avec r diviseur de q. Montrer que r est aussi une période de x.

1.10 (Code)

Un langage $X \subseteq A^*$ est un ***code*** si tout mot de X^+ admet une décomposition unique en mots de X.

Montrer que le codage ASCII des caractères sur l'alphabet $\{\mathtt{0}, \mathtt{1}\}$ fournit un code au sens de cette définition.

Montrer que les langages $\{\mathtt{a}, \mathtt{b}\}^*$, $\mathtt{ab}^*$, $\{\mathtt{aa}, \mathtt{ba}, \mathtt{b}\}$, $\{\mathtt{aa}, \mathtt{baa}, \mathtt{ba}\}$ et $\{\mathtt{a}, \mathtt{ba}, \mathtt{bb}\}$ sont des codes. Montrer que ce n'est pas le cas des langages $\{\mathtt{a}, \mathtt{ab}, \mathtt{ba}\}$ et $\{\mathtt{a}, \mathtt{abbba}, \mathtt{babab}, \mathtt{bb}\}$.

Un langage $X \subseteq A^*$ est préfixe si la condition

$$u \preceq_{\textit{préf}} v \text{ implique } u = v$$

est satisfaite pour tous mots $u, v \in X$. La notion de langage suffixe est définie de façon duale.

Montrer que tout langage préfixe est un code. Même chose pour les langages suffixes.

1.11 (Théorème du défaut)

Soit $X \subseteq A^*$ un ensemble fini qui n'est pas un code. Soit $Y \subseteq A^*$ un code pour lequel Y^* est le plus petit ensemble de cette forme qui contient X^*. Montrer que $\operatorname{card} Y < \operatorname{card} X$. [*Aide* : tout mot $x \in X$ s'écrit sous la forme $y_1 y_2 \ldots y_k$ avec $y_i \in Y$ pour $i = 1, 2, \ldots, k$; montrer que la fonction $\alpha: X \to Y$ définie par $\alpha(x) = y_1$ est surjective mais n'est pas injective ; voir [56].]

1.12 (Commutation)

Montrer par le théorème du défaut (voir exercice 1.11), puis par le lemme de périodicité que, si $uv = vu$, pour deux mots $u, v \in A^*$, u et v sont des puissances d'un même mot.

1.13 (nlogn)

Soit $f: \mathbf{N} \to \mathbf{N}$ une fonction définie par :

$$\begin{aligned} f(1) &= a \ , \\ f(n) &= f(\lfloor n/2 \rfloor) + f(\lceil n/2 \rceil) + bn \qquad \text{pour } n \geq 2 \ \ , \end{aligned}$$

avec $a \in \mathbf{N}$ et $b \in \mathbf{N} \setminus \{0\}$. Montrer qu'alors $f(n)$ est $\Theta(n \log n)$.

1.14 (Filtre)
On considère un système dans lequel les caractères sont codés sur 8 bits. On veut développer un algorithme de localisation par automate pour des mots écrits sur l'alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

Décrire des structures de données pour réaliser l'automate à l'aide d'une matrice de transitions de taille $4 \times m$ (et non pas $256 \times m$), où m est le nombre d'états de l'automate, utilisant éventuellement un espace supplémentaire dont la taille est indépendante de m.

1.15 (Implantation de fonctions partielles)
Soit $f : E \to F$ un fonction partielle où E est un ensemble fini. Décrire une implantation de f permettant de réaliser chacune des quatre opérations suivantes en temps constant :

- initialiser f, de sorte que $f(x)$ est indéfini pour chaque $x \in E$;
- fixer la valeur de $f(x)$ à $y \in F$, pour chaque $x \in E$;
- tester si $f(x)$ est défini ou non, pour chaque $x \in E$;
- donner la valeur de $f(x)$, pour chaque $x \in E$.

On peut utiliser un espace $O(\text{card } E)$. [*Aide :* utiliser simultanément une table indexée par E et une liste des éléments x pour lesquels $f(x)$ est défini, avec des renvois entre table et liste.]

En déduire que la mise en table d'une telle fonction peut se faire en temps linéaire en le nombre d'éléments de E dont l'image par f est définie.

1.16 (Pas si naïf)
On considère ici une implantation un peu plus élaborée du mécanisme de la fenêtre glissante que celle décrite pour l'algorithme naïf. Parmi les mots x de longueur $m \geq 2$, elle distingue deux classes : celle pour laquelle les deux premières lettres sont identiques (soit $x[0] = x[1]$), et la classe antagoniste (soit $x[0] \neq x[1]$). Cette distinction élémentaire permet de décaler la fenêtre de deux positions vers la droite dans les cas suivants : le mot cherché x appartient à la première classe et $y[j+1] \neq x[1]$; le mot cherché x appartient à la seconde classe et $y[j+1] = x[1]$. D'autre part, si la comparaison du mot x et du contenu de la fenêtre s'effectue toujours lettre à lettre, celle-ci suit ici l'ordre $1, 2, \ldots, m-1$ puis 0 des positions sur x.

Donner le code d'un algorithme qui applique cette méthode.

Montrer que le nombre de comparaisons par lettre de texte est en moyenne (légèrement) strictement inférieur à 1 lorsque la moyenne est évaluée sur l'ensemble des mots de même longueur, que cette longueur est supérieure à 2 et que l'alphabet contient au moins quatre lettres. [*Aide :* voir Hancart (1993).]

1.17 (Bout de fenêtre)

Considérer la méthode qui, à l'instar de celle de l'algorithme LOCALISER-RAPIDEMENT utilisant la lettre la plus à droite dans la fenêtre pour effectuer un décalage, utilise les deux lettres les plus à droite dans la fenêtre (à supposer que le mot cherché soit de longueur supérieure à 2).

Donner le code d'un algorithme qui applique à cette méthode.

Dans quels cas semble-t-il intéressant ? [*Aide :* voir Zhu et Takaoka (1987) ou Baeza-Yates (1989).]

1.18 (Après la fenêtre)

Même énoncé que celui de l'exercice 1.17, mais en utilisant la lettre située immédiatement à droite de la fenêtre (attention au débordement à l'extrémité droite du texte). [*Aide :* voir Sunday (1990).]

1.19 (Sentinelle)

On revient encore sur le problème de la recherche des occurrences d'un mot x de longueur m dans un texte y de longueur n pour lequel on cherche à donner des implantations efficaces pour solutions élémentaires.

La technique de la sentinelle peut être utilisée pour rechercher la lettre $x[m-1]$ en effectuant les décalages à l'aide de la table *dern-occ*. Les décalages pouvant être de longueur m, on fixe $y[n \,.\,.\, n+m-1]$ à $x[m-1]^m$. Donner un code correct pour cette méthode à sentinelle.

De manière à accélérer le processus et diminuer les tests sur les lettres, il est possible d'enchainer plusieurs décalages sans tester les lettres du texte. Pour cela, on sauvegarde la valeur de *dern-occ*$[x[m-1]]$ dans une variable, disons d, puis on fixe la valeur de *dern-occ*$[x[m-1]]$ à 0. On peut ensuite enchainer des décalages jusqu'à ce que l'un d'entre eux soit de longueur 0. On teste alors les autres lettres de la fenêtre, en signalant une occurrence le cas échéant, et l'on applique un décalage de longueur d. Donner un code correct pour cette méthode. [*Aide :* voir Hume et Sunday (1991).]

1.20 (En C)

Réaliser une implantation en langage C de l'algorithme LOCALISER-MOTS-COURTS. À titre indicatif, les opérateurs $\vee$, $\wedge$ et $\dashv$ se codent `|`, `&` et `<<`. Étendre l'implantation pour qu'elle accepte un paramètre m quelconque (éventuellement strictement supérieur au nombre de bits d'un mot machine).

Comparer le code obtenu à celui de la source de la commande `agrep` d'Unix.

1.21 (Mots courts)

Décrire un algorithme de localisation de mots courts à la façon de l'algorithme LOCALISER-MOTS-COURTS, mais dans lequel les valeurs binaires `0` et `1` sont échangées.

1.22 (Borne)
Montrer que le nombre de comparaisons positives ainsi que le nombre de comparaisons négatives effectuées lors de l'opération BORDS(x, m) sont au plus égaux à $m - 1$. Retrouver ensuite la borne $2m - 3$ de la proposition 1.24.

1.23 (Table des préfixes)
Décrire un algorithme linéaire de calcul de la table *préf* connaissant la table *bord* du mot x.

1.24 (Localisation par les bords ou les préfixes)
Montrer que la table des bords pour le mot $x\$y$ peut être directement utilisée afin de localiser toutes les occurrences du mot x dans le mot y, où $\$ \notin alph(xy)$.

Même chose avec la table des préfixes pour le mot xy.

1.25 (Couverture)
Un mot u est une couverture d'un mot x si pour toute position i sur x, il existe une position j sur x pour laquelle $0 \leq j \leq i < j + |u| \leq |x|$ et $u = x[j \mathinner{..} j + |u| - 1]$.

Écrire un algorithme de calcul de la plus courte couverture d'un mot. En énoncer la complexité.

1.26 (Long bord)
Soit u un bord non vide du mot $x \in A^*$.

Soit $v \in A^*$ tel que $|v| < |u|$. Montrer que v est un bord de u si et seulement si il est un bord de x.

Montrer que x possède un autre bord non vide si u satisfait l'inégalité $|x| < 2|u|$. Montrer que x ne possède aucun autre bord satisfaisant la même inégalité si $pér(x) > |x|/4$.

1.27 (Sans bord)
On dit qu'un mot non vide u est sans bord si $Bord(u) = \varepsilon$, ou, de manière équivalente, si $pér(u) = |u|$.

Soit $x \in A^*$. Montrer que $C = \{u : u \preceq_{préf} x \text{ et } u \text{ sans bord}\}$ est un code suffixe (voir exercice 1.10).

Montrer que x se factorise de manière unique en $x_k x_{k-1} \ldots x_1$ selon les mots de C ($x_i \in C$ pour $i = 1, 2, \ldots, k$). Montrer que x_1 est le plus court mot de C et que x_k en est le plus long.

Décrire un algorithme linéaire de calcul de la factorisation.

1.28 (Suffixe maximal)
On note $SM(\leq, u)$ le suffixe maximal de $u \in A^+$ pour l'ordre lexicographique où, dans cette notation, $\leq$ désigne l'ordre sur l'alphabet. Soit $x \in A^+$.

Montrer que $|SM(\leq, x)| < pér(x)$.

On suppose que $SM(\leq, x) = x$ et l'on note $w_1, w_2, \ldots, w_k$ les bords de x en ordre décroissant de longueur (on a $k > 0$ et $w_k = \varepsilon$). Soient $a_1, a_2, \ldots, a_k \in A$ et $z_1, z_2, \ldots, z_k \in A^*$ tels que

$$x = w_1 a_1 z_1 = w_2 a_2 z_2 = \cdots = w_k a_k z_k \quad .$$

Montrer que $a_1 \leq a_2 \leq \cdots \leq a_k$.

Écrire un algorithme linéaire de calcul du suffixe maximal (pour l'ordre lexicographique) d'un mot $x \in A^+$. [*Aide :* utiliser l'algorithme de calcul des bords de la section 1.6 ou voir Booth (1980) ; voir aussi [2].]

1.29 (Factorisation critique)

Soit $x \in A^+$. Pour chaque position i sur x, on note

$$\begin{aligned} r\acute{e}p(i) = \min\{|u| : u \in A^+, A^*u \cup A^*x[0\,.\,.\,i-1] \neq \emptyset \text{ et} \\ uA^* \cup x[i\,.\,.\,|x|-1]A^* \neq \emptyset\} \end{aligned}$$

la période locale de x en i.

En notant $w = SM(\leq, x)$ (SM est défini dans l'exercice 1.28) et en supposant que $|w| \leq |SM(\leq^{-1}, x)|$, montrer que $r\acute{e}p_x(|x|-|w|) = p\acute{e}r(x)$. [*Aide :* noter que l'intersection des deux ordres induits sur les mots est l'ordre préfixe, et utiliser la proposition 1.4 ; voir Crochemore et Perrin (1991) ou [2].]