Informatique I

La notation tiendra compte de la rigueur des raisonnements et de la clarté des explications.
Chaque question peut être traitée en admettant le résultat des questions précédentes.

Le problème s'intéresse à la recherche de motifs dans un texte et à quelques applications possibles.

Notations et définitions

- A est un alphabet supposé connu, contenant au moins deux lettres.
- A^* est l'ensemble des mots sur l'alphabet A. Dans la suite, t, u, v, w, x, y, z désignent des mots.
- $|u|$ désigne la longueur d'un mot u de A^*.
- On note ε le mot vide. On a donc $|\varepsilon| = 0$.
- $|X|$ est le cardinal d'un ensemble fini X.
- On dit que u est *facteur* de v si $v = u_1 u u_2$, avec $u_1, u_2 \in A^*$.
- Si $v = u_1 u u_2$ et $|u_1| = k - 1$, on dit que u a une *occurrence* dans v en *position* k.
- On dit que u est *préfixe* de v si $v = uu'$, avec $u' \in A^*$.
- On dit que u est *suffixe* de v si $v = u'u$, avec $u' \in A^*$.

La complexité en temps des algorithmes sera évaluée en nombre de comparaisons de lettres.

I Algorithmes de recherche

Dans toute cette partie, on s'intéresse au problème RECHERCHE-MOTIF suivant :

Donnée : – deux entiers m et n,
– un mot $x = x_1 \cdots x_m$ ($x_i \in A$) appelé le *motif*,
– un mot $t = t_1 \cdots t_n$ ($t_i \in A$) appelé le *texte*.

Question : Le mot x est-il facteur du mot t? Si oui, quelle est la position de la première occurrence de x dans t?

Un algorithme « naïf » consiste à tester l'égalité $x_1 \cdots x_m = t_k \cdots t_{k+m-1}$ pour k allant de 1 à $n - m + 1$. Chacun de ces tests se fait en comparant les mots lettre à lettre de gauche à droite. Durant le test $x_1 \cdots x_m = t_k \cdots t_{k+m-1}$, l'entier k est appelé la *position de comparaison* du motif.

Lorsqu'un test en position de comparaison k est négatif sur la $i^{\text{ème}}$ lettre de x ($x_i \neq t_{k+i-1}$), les algorithmes que l'on va considérer effectuent un *décalage* d du motif : la nouvelle position de comparaison est $k + d$. Dans l'algorithme naïf, le déplacement d vaut toujours 1, et le test $x_1 \cdots x_m = t_\ell \cdots t_{\ell+m-1}$ ($\ell = k + 1$ étant la nouvelle position de comparaison) reprend au début des deux mots. La figure 1 montre comment l'algorithme décale le motif en cas d'échec d'un des tests $x_i = t_{k+i-1}$.

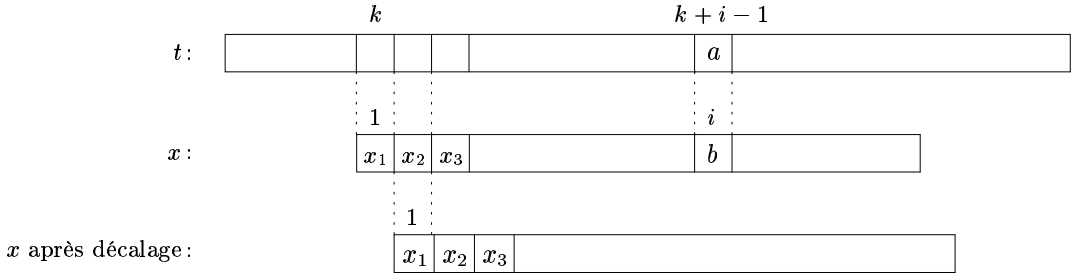

FIG. 1: *Décalage du motif en cas d'échec du test $x_i = t_{k+i-1}$ dans l'algorithme naïf*

La fonction de la figure 2 donne une version de cet algorithme. S'il n'y a pas d'occurrence de x dans t, la fonction calcule la valeur -1. Sinon, elle calcule la position de la première occurrence de x dans t. Dans ce programme, la première case d'un tableau est numérotée 1. Ainsi, on accède à la $i^{\text{ème}}$ lettre de x par `x[i]`. De plus, les tests sont évalués de façon paresseuse : par exemple, dans un test `c1 et c2`, l'expression `c2` n'est évaluée que si l'expression `c1` est vraie.

```
1    fonction recherche_naïve (x, t, m, n)
2        k := 1;
3        tantque k <= n-m+1 faire
4            i := 1;
5            tantque i <= m et x[i] = t[k+i-1] faire i := i+1 fintantque
6            si i = m+1 alors retourner k finsi
7            k := k+1;
8        fintantque
9        retourner -1;
```

FIG. 2: *Une fonction de recherche de motif dans un texte*

I.1 Calculer le nombre de comparaisons de cet algorithme dans le cas le pire en fonction de m et n. Donner un exemple de mots x et t pour lesquels le cas le pire est réalisé.

Solution *On entre dans la boucle principale de la fonction de la figure 2 au maximum $n-m+1$ fois. La boucle interne fait varier i de 1 à m, donc il y a au maximum $m(n-m+1)$ comparaisons de caractères. Ce nombre est atteint lorsque chacune des comparaisons $x_1 \cdots x_m = t_k \cdots t_{k+m-1}$ échoue sur la dernière lettre, soit par exemple pour le motif $x = a^{m-1}b$ et le texte $t = a^n$.* □

I.2 On rappelle qu'il est supposé que $|A| \geqslant 2$. On suppose que chaque lettre de l'alphabet a une probabilité $\frac{1}{|A|}$ d'apparaître à chaque position (dans le texte comme dans le motif). Calculer le nombre moyen de comparaisons de lettres faites lors de la comparaison de x à $t_k \cdots t_{k+m-1}$. Montrer que ce nombre est plus petit que 2. En déduire que l'algorithme effectue moins de $2n$ comparaisons en moyenne.

Solution *Soit $p = \frac{1}{|A|}$ la probabilité d'apparition d'une lettre à une position donnée. Lors d'une comparaison $x_1 \cdots x_m = t_k \cdots t_{k+m-1}$, on fait exactement $j+1$ comparaisons lorsque*

$$x_1 \cdots x_j = t_k \cdots t_{k+j-1} \text{ et } x_{j+1} \neq t_{k+j} \text{ pour } j+1 < m$$
$$x_1 \cdots x_j = t_k \cdots t_{k+j-1} \text{ pour } j+1 = m$$

Or la probabilité pour que l'on ait $x_i = t_\ell$ est exactement p. Donc la probabilité de faire exactement $j+1$ comparaisons est $p^j(1-p)$ pour $j < m-1$ et p^{m-1} pour $j = m-1$. On fait donc en moyenne $\sum_{j=0}^{m-2}(j+1)p^j(1-p) + mp^{m-1}$ comparaisons. En ordonnant cette somme suivant les puissances de p, on trouve donc comme nombre moyen de comparaisons

$$\sum_{j=0}^{m-2}(j+1)p^j(1-p) + mp^{m-1} = \sum_{j=0}^{m-1} p^j = \frac{1-p^m}{1-p}$$

Ce nombre est plus petit que $\frac{1}{1-p}$, et comme par hypothèse $|A| \geqslant 2$, on a $\frac{1}{1-p} \leqslant 2$. Lors d'un test $x_1 \cdots x_m = t_k \cdots t_{k+m-1}$, on fait donc moins de 2 comparaisons en moyenne. Comme on fait ce test moins de n fois, le nombre moyen de comparaisons est majoré par $2n$. □

I.3 On cherche maintenant à améliorer les performances de l'algorithme dans le cas le pire. On fixe une position de comparaison k dans t, et on suppose que $x_1 \cdots x_m \neq t_k \cdots t_{k+m-1}$. L'*indice d'échec*

$i_e(k)$ en position de comparaison k est le plus petit entier i tel que $x_i \neq t_{k+i-1}$. Un décalage d est *inutile* en position de comparaison k lorsque $d < i_e(d) - 1$ et $x_1 \cdots x_{i_e(d)-d-1} \neq t_{k+d} \cdots t_{k+i_e(d)-2}$. Les autres décalages sont dits *utiles*.

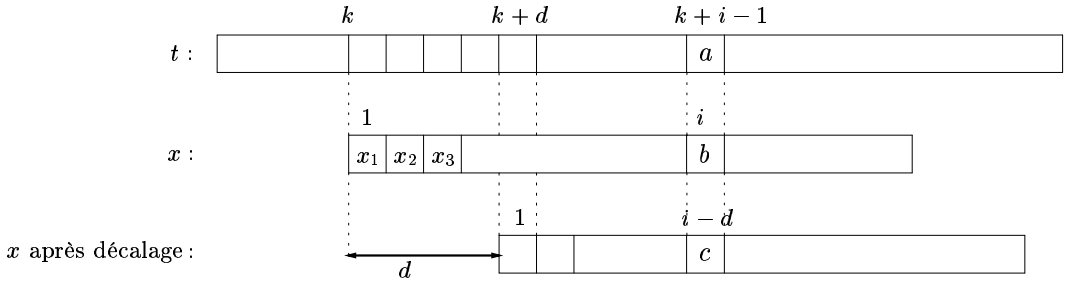

FIG. 3: *Décalage d du motif*

Montrer que si d est un décalage inutile en position de comparaison k, il n'y a pas d'occurrence de x en position $k + d$.

Solution *C'est immédiat : si le décalage d est inutile, on a* $x_1 \cdots x_{i_e(d)-d-1} \neq t_{k+d} \cdots t_{k+i_e(d)-2}$. *On ne peut donc pas avoir d'occurrence de x en position* $k + d$. □

Notation et définition

- On dit que u est un *bord* de v si $u \neq v$ et si u est à la fois préfixe et suffixe de v.
- Pour $v \in A^* \backslash \{\varepsilon\}$, Bord$(v)$ est ***le*** *bord maximal* de v, c'est-à-dire le bord le plus long de v.

Ainsi par exemple, le mot *ababa* admet comme bords *aba*, *a* et ε. Le bord maximal est *aba*. On notera que dans cet exemple, le préfixe *aba* et le suffixe *aba* se chevauchent dans *ababa*. Il n'en est pas de même pour le bord maximal de *abcab*, qui est *ab*.

I.4 On note p_i le préfixe de longueur i du motif x, pour $i = 0, \ldots, |x|$. Donner une condition nécessaire portant uniquement sur les p_j et ne faisant pas intervenir t pour que le décalage d soit utile. Calculer le plus petit décalage utile $d_u(k)$ en fonction de x et $i_e(k)$.

Solution *Soit d un décalage utile en position de comparaison k. D'après la définition de* $i_e(k)$ *on sait que* $x_1 \cdots x_{i_e(k)-1} = t_k \cdots t_{k+i_e(k)-2}$. *En particulier, en privant ces deux mots de leurs d premiers caractères*

$$x_{1+d} \cdots x_{i_e(k)-1} = t_{k+d} \cdots t_{k+i_e(k)-2}$$

Mais comme le décalage d est utile, on a aussi par définition

$$x_1 \cdots x_{i_e(k)-d-1} = t_{k+d} \cdots t_{k+i_e(k)-2} \tag{1}$$

On a donc $x_{1+d} \cdots x_{i_e(k)-1} = x_1 \cdots x_{i_e(k)-d-1}$, *donc* $p_{i_e(k)-d-1}$ *est nécessairement un bord de* $p_{i_e(k)-1}$.

Plus le décalage utile est petit, plus le bord $p_{i_e(k)-d-1}$ *de* $p_{i_e(k)-1}$ *est grand. Lorsque le décalage utile est minimum, c'est-à-dire lorsque* $d = d_u(k)$, $p_{i_e(k)-d_u(k)-1}$ *est le plus long bord de* $p_{i_e(k)-1}$ *:*

$$p_{i_e(k)-d_u(k)-1} = Bord(p_{i_e(k)-1}) \tag{2}$$

□

I.5 Après un décalage utile d (cf. figure 3), à quel indice $j(d)$ dans le mot x peut-on reprendre la comparaison $x_1 \cdots x_m = t_{k+d} \cdots t_{k+d+m-1}$?

Solution *Le décalage est utile, on a donc l'égalité (1). La comparaison peut donc reprendre à l'indice* $j(d) = i_e(k) - d$. □

I.6 On définit une fonction f_x qui va de $\{0,\ldots,m\}$ dans $\{-1,\ldots,m-1\}$ de la façon suivante :

$$f_x(0) = -1$$
$$f_x(i) = |\mathrm{Bord}(p_i)|$$

En supposant que l'on dispose de la fonction f_x, modifier la fonction donnée en figure 2 de façon à effectuer à chaque fois le plus petit décalage utile et à reprendre la comparaison $x_1 \cdots x_m = t_{k+d} \cdots t_{k+d+m-1}$ à l'indice $j(d_u(k))$.

Solution *De (2), on déduit qu'en cas d'échec à l'indice $i = i_e(k)$, le décalage d à effectuer est tel que $i-d-1 = f_x(i-1)$, d'où $d = i-1-f_x(i-1)$. L'indice $j(d) = i_e(k) - d$ vaut donc $j(d) = f_x(i-1)+1$. Si i vaut 1, on doit juste décaler le motif d'une position et reprendre à $i+1$, $k+1$.*

```
1     fonction recherche (x, t, m, n)
2         k := 1; i := 1;
3         tantque k <= n-m+1 faire
4             tantque i <= m et x[i] = t[k+i-1] faire i := i+1 fintantque
5             si i = m+1 alors retourner k finsi
6             d := i-1-f_x(i-1);
7             k := k+d;
8             i := i-d;
9             si i = 0 alors i := i+1 finsi
10        fintantque
11        retourner -1;
```

FIG. 4: *La fonction de recherche de motif dans un texte améliorée*

Si après l'instruction `i := i-d` *l'indice i est nul, on doit le repositionner à 1 et k doit avoir été incrémenté. Or si i est nul en 9, c'est que d calculé ligne 6 vaut 1, donc k a été correctement incrémenté ligne 7.* □

I.7 On dit qu'une comparaison de lettres de la forme `x[i]=t[j]` effectuée par l'algorithme est un *test positif* si $x_i = t_j$, et un *test négatif* sinon. En considérant dans la fonction écrite en I.6 la variation des quantités $k+i$ et k après une comparaison de lettres, majorer le nombre de tests positifs et le nombre de tests négatifs. Montrer que l'algorithme fait toujours moins de $2n-m$ comparaisons de lettres dans le cas le pire si l'on dispose de la fonction f_x. Montrer par un exemple que cette borne est atteinte.

Solution *Un test négatif ne fait pas décroître la quantité $i+k$ (lignes 7, 8, 9). Un test positif incrémente $i+k$ de 1. Or $i+k \leqslant (m+1)+(n-m+1) = n+2$ et la valeur initiale de $i+k$ est 2. Il y a donc au plus n tests positifs. D'autre part, un test négatif augmente k, qui vaut initialement 1, et un test positif le change pas k. Il y a donc au plus $n-m$ tests négatifs. Donc en tout, il y a au plus $n+(n-m)$ tests, soit $2n-m$.*

Cette borne est atteinte pour $x = a^{m-1}b$ et $t = a^n$. En effet, l'algorithme

- *effectue m comparaisons en position de comparaison 1 et échoue sur la dernière lettre,*
- *décale le motif d'une position et effectue deux comparaisons (une pour x_{m-1} et une pour x_m), et ce, jusqu'à la fin de t c'est-à-dire $n-m$ fois.*

On a donc en tout $m + 2(n-m) = 2n-m$ comparaisons de lettres. □

I.8 Calculer le nombre maximal de comparaisons qu'une lettre du texte peut subir avant que l'on passe à la lettre suivante.

Solution *On voit facilement que ce nombre est m, atteint par exemple pour $x = a^m$ et $t = a^{m-1}ba^m$ sur la lettre b de t.* □

I.9 On veut maintenant montrer que l'on peut calculer la fonction f_x en temps $O(m)$. Pour une fonction g, on note g^k la fonction $\underbrace{g \circ \cdots \circ g}_{k \text{ fois}}$ (quand elle existe).

a) Soit u un mot non vide. Comme la fonction Bord fait décroître la longueur, il existe un entier q tel que $\text{Bord}^q(u) = \varepsilon$. Soit p le plus petit entier tel que $\text{Bord}^p(u) = \varepsilon$. Montrer que l'ensemble des bords de u est $\mathscr{B}(u) = \{\text{Bord}(u), \text{Bord}^2(u), ..., \text{Bord}^p(u)\}$.

Solution *On constate que si y est un bord non vide de z et si x est un bord de y, alors x est un bord de z. Donc un mot de l'ensemble $\mathscr{B}(u)$ est un bord de u.*

Réciproquement, on montre par récurrence sur $|u|$ que si v est un bord de u, alors v est dans $\mathscr{B}(u)$. Pour $|u| = 1$, c'est évident. Supposons le résultat vrai pour $|u| < \ell$ et soit u de longueur ℓ. Soit v un bord de u. Si v est de longueur maximale, alors $v = Bord(u)$. Sinon, $|v| < |Bord(u)|$, et comme v et $Bord(u)$ sont des préfixes (resp. des suffixes) de u, v est aussi un bord de $Bord(u)$. Mais $|Bord(u)| < \ell$, et on conclut par récurrence. □

b) Soit a une lettre et $u \in A^*$ non vide. Montrer que $\text{Bord}(ua)$ est le plus long préfixe de u qui est dans l'ensemble $\mathscr{B}(u,a) = \{\text{Bord}(u)a, \text{Bord}^2(u)a, ..., \text{Bord}^p(u)a, \varepsilon\}$.

Solution *Soit v un bord de ua. Si $v \neq \varepsilon$, v est nécessairement de la forme wa, puisque v est suffixe de ua. Comme v est aussi préfixe de ua, w doit être un bord de u. D'après la question précédente, w est dans $\mathscr{B}(u)$. Donc $v \in \mathscr{B}(u,a)$ et $\mathscr{B}(ua) \setminus \{\varepsilon\} \subseteq \mathscr{B}(u,a)$. Inversement, tout mot de $\mathscr{B}(u,a)$ est un suffixe de ua. Donc $v \in \mathscr{B}(u,a)$ est un bord de ua si et seulement si v est un préfixe de ua, et donc un préfixe de u (puisque $|v| < |ua|$). $Bord(ua)$ est donc bien le plus long préfixe de u qui est dans l'ensemble $\mathscr{B}(u,a)$.* □

c) En déduire que

$$\text{Bord}(ua) = \begin{cases} \text{Bord}(u)a & \text{si Bord}(u)a \text{ est préfixe de } u \\ \text{Bord}(\text{Bord}(u)a) & \text{sinon} \end{cases}$$

Solution *Si $Bord(u)a$ est préfixe de u, alors $Bord(ua) = Bord(u)a$ d'après la question précédente. Sinon posons $v = Bord(u)$. On a $\mathscr{B}(v,a) = \mathscr{B}(u,a) \setminus \{va\}$. Puisque $Bord(ua) \neq va$, $Bord(ua)$ et $Bord(va)$ sont dans l'ensemble $\mathscr{B}(v,a)$, et $Bord(va)$ est même le plus long préfixe de u qui est dans cet ensemble. Il reste à montrer que $Bord(ua)$ est aussi le plus long préfixe de u dans $\mathscr{B}(v,a)$. Mais toujours d'après la question précédente, $Bord(ua)$ est le plus long préfixe de u dans l'ensemble $\mathscr{B}(u,a)$, et $\mathscr{B}(v,a) = \mathscr{B}(u,a) \setminus \{va\}$, ce qui permet de conclure.* □

d) Soit $j \in \{1, \ldots, m\}$, et $g = f_x$. Montrer que $g(j) = g^k(j-1) + 1$, où $k \geqslant 1$ est le plus petit entier tel que l'on ait soit $g^k(j-1) + 1 = 0$, soit $g^k(j-1) + 1 \neq 0$ et $x_{g^k(j-1)+1} = x_j$.

Solution *On rappelle qu'on note p_j le préfixe de longueur j de x. On montre par récurrence sur j que*

$$Bord(p_j) = \begin{cases} Bord^k(p_{j-1})x_j & \text{où } k = \min\left\{\ell \geqslant 1 \mid Bord^\ell(p_{j-1})x_j \text{ est préfixe de } x\right\} \\ \varepsilon & \text{si un tel } \ell \text{ n'existe pas} \end{cases} \tag{3}$$

Pour $j = 1$, $Bord(p_0)$ n'est pas défini, donc l'ensemble $\left\{\ell \geqslant 1 \mid Bord^\ell(p_{j-1})x_j \text{ est préfixe de } x\right\}$ est vide. Comme on a $Bord(p_1) = \varepsilon$, le résultat est vrai. Pour $j > 1$, on applique le calcul de la question précédente avec $u = p_{j-1} \neq \varepsilon$ et en prenant comme lettre $a = x_j$.

$$Bord(p_j) = Bord(p_{j-1}x_j) = \begin{cases} Bord(p_{j-1})x_j & \text{si } Bord(p_{j-1})x_j \text{ est préfixe de } p_j \\ Bord\left(Bord(p_{j-1})x_j\right) & \text{sinon} \end{cases}$$

Or $Bord(p_{j-1})x_i$ est préfixe de p_{j-1} si et seulement si il est préfixe de x. La récurrence est alors immédiate.

On a $f_x(j) = |Bord(p_j)|$. Le résultat demandé se déduit de (3) par passage aux longueurs, et du fait que $Bord^k(p_{j-1})x_j$ est préfixe de x si et seulement si $x_j = x_{f_x^k(j-1)+1}$. □

e) En déduire un algorithme qui calcule tous les $f_x(j)$ pour $j = \{1, \ldots, m\}$ en temps $O(m)$.

Solution *La traduction de la question précédente en un algorithme est immédiate :*

```
1   procedure calcul_bords (x, m)
2       f_x[0] := -1;
3       pour j := 1 a m faire
4           l := f_x[j-1]+1;
5           tantque l > 0 et  x[j]  ≠ x[l] faire l := f_x[l-1]+1 fintantque
6           f_x[j] := l;
7       finpour
```

FIG. 5: *Une procédure de calcul des bords d'un mot x*

On note l'analogie avec la fonction de recherche. La preuve du fait que l'algorithme obtenu est linéaire est similaire : la variable `l` prend la valeur 0 dans la première itération de la boucle `pour...finpour`. Puis, elle ne peut être incrémentée dans les itérations 2 à m de la boucle `pour` que par l'instruction `l:=f_x[j-1]+1`. Mais `j-1` a la valeur qu'avait `j` dans l'itération précédente de la boucle. D'après la dernière instruction de la boucle (ligne 6) effectuée dans l'itération précédente, l'instruction `l:=f_x[j-1]+1` incrémente `l` exactement d'une unité. Donc la variable l est incrémentée $m-1$ fois d'une unité. D'autre part, la variable `l` décroît uniquement lors de l'instruction `l:=f_x[l-1]+1`, et elle décroît strictement. Or, `l` reste toujours positif ou nul, donc l'instruction `l:=f_x[l-1]+1` ne peut pas être réalisée plus de $m-1$ fois. Donc il y a au plus $m-1$ tests `x[j]≠x[l]` qui s'évaluent en vrai. D'autre part, un test `x[j]≠x[l]` qui s'évalue en faux fait sortir de la boucle `tantque` et passer à l'itération suivante de la boucle `pour`, donc provoque l'incrémentation de j. Ceci ne peut se produire que $m-2$ fois (lorsque `j` vaut 1, c'est la condition `l>0` qui fait sortir de la boucle `tantque`). Il y a donc au plus $(m-1)+(m-2) = 2m-3$ comparaisons de caractères. Ce nombre est d'ailleurs atteint pour le mot $x = a^{m-1}b$. □

I.10 En déduire un algorithme qui résout le problème RECHERCHE-MOTIF en temps $O(m+n)$.

Solution *L'algorithme consiste à rassembler les deux phases décrites en questions I.7 et I.9. La phase de calcul des bords du motif se fait en temps $O(m)$, et la phase de recherche du motif dans le texte en temps $O(n)$. Le tout se fait donc en temps $O(n+m)$.* □

II Applications

Dans cette partie, on pourra utiliser les résultats des questions I.9 et I.10.

II.1 Soit $x \in A^*$. Un facteur y de x est *répété* s'il a deux occurrences distinctes dans x. Ces occurrences peuvent se chevaucher (par exemple dans *abcabca*, le facteur *abca* est répété) ou non (par exemple dans ce même mot, *a* et *bca* sont répétés). Montrer que l'on peut déterminer le plus long préfixe répété d'un mot x en temps $O(|x|)$.

Solution *Il suffit d'adapter l'algorithme de la section I en cherchant le motif $x = x_1x_2\cdots x_m$ dans le texte $x_2\cdots x_m$, et de retenir la position de comparaison où i prend sa valeur maximale. La différence avec la fonction `recherche` écrite plus haut est que l'on laisse varier k jusqu'à m au lieu de $m-m+1$, mais ceci ne change pas la complexité $O(m)$.* □

II.2 Un mot $x \neq \varepsilon$ est dit *primitif* s'il n'est pas de la forme y^n avec $n \geqslant 2$. Par exemple $abac$ est primitif alors que $abab = (ab)^2$ n'est pas primitif. Montrer qu'un mot x est primitif si et seulement les seules occurrences de x dans xx sont celle commençant en position 1 et celle commençant en position $|x|+1$. Montrer que l'on peut déterminer en temps $O(|x|)$ si x est primitif.

Solution *Si $x \neq \varepsilon$ n'est pas primitif, il est de la forme y^n avec $n \geqslant 2$. Donc $xx = y^{2n}$ et on constate que x apparaît dans xx en position $|y|+1$. Or $1 < |y|+1 < |x|+1$. Réciproquement, si x apparaît dans xx à une position k avec $1 < k < |x|+1$, on peut écrire $x = uv = vu$ avec $u, v \in A^*\backslash\{\varepsilon\}$, comme le montre la figure 6.*

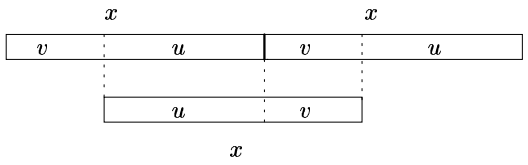

FIG. 6: *Occurrence non triviale de x dans xx*

On montre maintenant par récurrence sur $|uv|$ que si $u, v \neq \varepsilon$ et $uv = vu$, alors $\{u, v\} = \{z^k, z^\ell\}$ pour un mot z. Ceci terminera la preuve, car alors $x = z^{k+\ell}$ avec $k + \ell \geqslant 2$. Si $|u| = |v|$, alors $u = v$ et le résultat est évident. Sinon, on peut par exemple supposer que $|u| > |v|$. De $uv = vu$, on déduit l'existence de $w \in A^*\backslash\{\varepsilon\}$ tel que $u = vw = wv$. Mais $|vw| < |uv|$. Par hypothèse de récurrence, il existe donc un z tel que $w = z^r$ et $v = z^s$. On en déduit que $u = z^{r+s}$, ce qui termine la démonstration.

On en déduit immédiatement qu'on peut déterminer en temps $O(|x|)$ si x est primitif. En effet, ceci revient à chercher une occurrence de x dans xx en position $k \neq 1, |x|+1$, et ceci se fait en temps linéaire par une adaptation triviale de l'algorithme de recherche. □

II.3 Deux mots x et y sont *conjugués* s'il existe $u, v \in A^*$ tels que $x = uv, \quad y = vu$. Montrer que l'on peut déterminer en temps $O(n)$ si deux mots de longueur n sont conjugués.

Solution *On vérifie aisément que x et y de même longueur sont conjugués si et seulement si x est facteur de y^2. On peut donc résoudre ce problème en temps $O(n+2n) = O(n)$ d'après I.* □

II.4 L'*image miroir* d'un mot $u = u_1 u_2 \cdots u_p$ ($u_i \in A$) est le mot $\tilde{u} = u_p \cdots u_2 u_1$, c'est-à-dire le mot dont la suite des lettres est celle de u lue « à l'envers ». Un mot u est appelé un *palindrome* si $u = \tilde{u}$. Montrer qu'étant donné un mot x, on peut calculer le plus long préfixe de x qui est un palindrome en temps $O(|x|)$.

Solution *Le plus long préfixe de x qui est un palindrome est $Bord(x\$\tilde{x})$, où \$ est une nouvelle lettre. À nouveau d'après I.9, on peut calculer ce mot en temps $O(|x|)$.* □

II.5 Un *carré* est un mot de la forme yy avec $y \neq \varepsilon$.

a) Montrer que x admet un préfixe qui est un carré si et seulement s'il existe $j \geqslant 2$ tel que $f_x(j) \geqslant j/2$.
Solution *Si x admet un préfixe yy qui est un carré, alors $|Bord(yy)| \geqslant |y|$ et on a le résultat voulu avec $j = 2|y|$. Réciproquement, supposons qu'il existe $j \geqslant 2$ tel que $f_x(j) \geqslant j/2$. On a alors la situation suivante :*

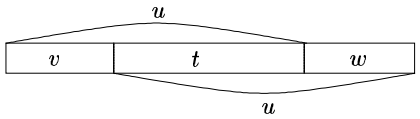

FIG. 7: *Le préfixe p_j lorsque $f_x(j) \geqslant j/2$*

Sur cette figure, le plus long bord de p_j est noté u. Par hypothèse, $|u| \geqslant |p_j|/2$, donc il y a « chevauchement ». Ceci permet de mettre en évidence les mots v, t, w de la figure, v (resp. w) étant le préfixe (resp. le suffixe) de p_j de longueur $|p_j| - |u|$. On note que comme u est le bord de p_j, $|u| < |p_j|$ donc v et w sont non vides.

Si $t = \varepsilon$, alors u^2 est préfixe de x. De manière plus génerale, si $|v| = |w| \geqslant |t|$, l'égalité $vt = tw$ implique que t est préfixe de v et suffixe de w. Donc $v = tv'$ et $w = w't$. L'égalité $vt = tw$ donne $tv't = tw't$, soit $v' = w'$, et le mot $tv'tv' = vv$ est un carré qui est préfixe de x.

Si au contraire $|v| = |w| \leqslant |t|$, l'égalité $vt = tw$ implique que t admet v comme préfixe, donc vv est à nouveau un carré qui est préfixe de x. □

b) Montrer que l'on peut déterminer si un mot x contient un carré comme facteur en temps $O(|x|^2)$.

Solution *Soit s_i le suffixe de longueur i de x. Pour i fixé, on sait qu'on peut calculer toutes les valeurs $f_{s_i}(j)$, $0 \leqslant j \leqslant i$ en temps $O(i)$. Mais $i \leqslant n$, on peut donc calculer toutes ces valeurs (i et j variant) en temps $O(|x|^2)$. D'après la question précédente, il suffit de parcourir l'ensemble des valeurs obtenues et de tester si $f_{s_i}(j) \geqslant j/2$ pour un certain $j \geqslant 2$ pour répondre à la question. Le problème est donc décidable en temps $O(|x|^2)$.* □

III Une distance d'édition

On dit que l'on passe d'un mot y à un mot z par

- une insertion si $y = st$ et $z = sat$, avec $s, t \in A^*$, $a \in A$.
- une suppression si $y = sat$ et $z = st$, avec $s, t \in A^*$, $a \in A$.

On définit la fonction suivante d de $A^* \times A^*$ dans $\mathbb{N}$: $d(y, z)$ est le plus petit entier k tel qu'il existe une suite de $k + 1$ mots $w_0, \ldots, w_k$ satisfaisant $y = w_0$, $z = w_k$, et telle que l'on passe de w_i à w_{i+1} par une insertion ou une suppression.

III.1 Montrer que d est une distance sur A^*, c'est-à-dire que d est positive et que l'on a

$$\text{a) } d(y,z) = 0 \Longleftrightarrow y = z, \quad \text{b) } d(y,z) = d(z,y), \quad \text{c) } d(y,z) \leqslant d(y,t) + d(t,z)$$

Solution *La fonction d est bien définie sur $A^* \times A^*$ et est positive. Le fait que $d(y, z) = 0$ si et seulement si $y = z$ provient de la définition de d. La symétrie provient de la symétrie de l'insertion par rapport à la suppression. Enfin, l'inégalité triangulaire vient du fait que s'il existe une suite pour passer de y à t de longueur $k + 1$ et une suite pour passer de t à z de longueur $\ell + 1$, on obtient une suite pour passer de y à z de longueur $k + \ell + 1$ en concaténant ces deux suites (sans répéter le terme « central » t, bien sûr).* □

III.2 Une *correspondance* f entre deux mots $y = y_1 \cdots y_p$ ($y_i \in A$) et $z = z_1 \cdots z_q$ ($z_i \in A$) est une fonction partielle de l'ensemble $Y = \{1, \ldots, p\}$ dans l'ensemble $Z = \{1, \ldots, q\}$ telle que

- f strictement croissante : si $i_1 < i_2$ et si $f(i_1)$, $f(i_2)$ sont définis, alors $f(i_1) < f(i_2)$.
- si $f(i)$ est défini, alors $y_i = z_{f(i)}$.

On peut voir une correspondance graphiquement, comme sur la figure 8.

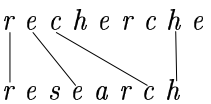

FIG. 8: *La correspondance* $1 \mapsto 1$, $2 \mapsto 4$, $3 \mapsto 7$, $8 \mapsto 8$ *entre* recherche *et* research

Soit f une correspondance et $G(f) = \{(i, f(i)) \mid i \in Y \text{ et } f(i) \text{ défini}\}$ son graphe. On définit

$$I(f) = \{i \in Y \mid \forall j \in Z,\ (i,j) \notin G(f)\}$$
$$J(f) = \{j \in Z \mid \forall i \in Y,\ (i,j) \notin G(f)\}$$

Le *coût* de f est l'entier $c(f) = |I(f)| + |J(f)|$. Sur l'exemple de la figure 8, ce coût est $|\{4,5,6,7,9\}| + |\{2,3,5,6\}| = 9$.

a) Soit g (resp. h) est une correspondance entre u et v (resp. entre v et w). Montrer que la fonction $h \circ g$ de $\{1, \ldots, |u|\}$ dans $\{1, \ldots, |w|\}$ est une correspondance entre u et w telle que $c(h \circ g) \leqslant c(h) + c(g)$.

Solution *La composée de fonction partielles croissantes est une fonction partielle croissante ; d'autre part, si $(h \circ g)(i)$ est définie, on a $u_i = v_{g(i)} = w_{h(g(i))}$ parce que g et h sont des correspondances. Donc $h \circ g$ est bien une correspondance entre u et w.*

Pour une fonction partielle f, on note $r(f)$ son rang, i.e., le nombre d'éléments sur lesquels f est définie. Si f est une correspondance entre deux mots y et z, on a $c(f) = |y| - r(f) + |z| - r(f) = |y| + |z| - 2r(f)$. Donc $c(g \circ h) \leqslant c(h) + c(g)$ équivaut à $|u| + |w| - 2r(g \circ h) \leqslant |u| + |v| - 2r(g) + |v| + |w| - 2r(h)$, soit $r(g) + r(h) - r(g \circ h) \leqslant |v|$. Mais ceci est vrai car $r(g) \leqslant |v|$, $r(h) \leqslant |v|$ et $r(g \circ h) \geqslant 0$. □

b) Montrer que $d(y, z)$ est égal au coût minimal d'une correspondance entre y et z.

Solution *Soit f une correspondance entre y et z. Comme f est strictement croissante, on obtient le même mot par suppression de y de toutes les lettres indicées par $I(f)$ et par suppression de z de toutes les lettres indicées par $J(f)$. Donc $d(y, z) \leqslant |I(f)| + |J(f)| = c(f)$.*

Réciproquement, soit $w_0 = z, w_1, \ldots, w_k = t$ une suite de longueur minimale telle que w_{i+1} est obtenu à partir de w_i par suppression ou insertion. On associe à chaque couple (w_i, w_{i+1}) une correspondance f_i de façon canonique. Si $w_i = sat$ et $w_{i+1} = st$, on pose $f_i(\ell) = \ell$ pour $\ell \leqslant |s|$, f_i n'est pas définie en $|s| + 1$, et $f_i(\ell) = \ell - 1$ pour $s + 2 \leqslant \ell \leqslant |w_i|$. Si au contraire $w_i = st$ et $w_{i+1} = sat$, on pose $f_i(\ell) = \ell$ pour $\ell \leqslant |s|$ et $f_i(\ell) = \ell + 1$ pour $s + 1 \leqslant \ell \leqslant |w_i|$. Dans ce dernier cas, la correspondance est représentée en figure 9. Clairement, f_i est une correspondance de coût 1.

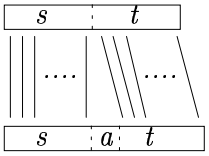

FIG. 9: *Correspondance entre st et sat*

D'après la question précédente, la correspondance $f = f_{k-1} \circ \cdots \circ f_1 \circ f_0$ entre les mots w_0 et w_k a donc un coût $c(f) \leqslant k = d(y, z)$. □

III.3 On va maintenant établir un algorithme pour calculer d.

a) Soient y, z deux mots et a, b deux lettres. Montrer que

$$d(ya, zb) = \min\big[d(y, zb) + 1,\ d(ya, z) + 1,\ d(y, z) + 2\delta(a \neq b)\big]$$

où $\delta(P)$ vaut 1 si la propriété P est vraie et 0 sinon.

Solution *Montrons que $d(ya, zb) \leqslant d(y, zb) + 1$. Si on a une suite de $k+1$ mots pour passer de y à zb, alors on a une suite de $k+2$ mots pour passer de ya à zb : il suffit de commencer par supprimer la dernière lettre de ya. On montre de manière symétrique que $d(ya, zb) \leqslant d(ya, z) + 1$. Enfin,*

$d(ya, zb) \leqslant d(y, z) + 2\delta(a \neq b)$ se montre en utilisant éventuellement une suppression (de a) et une insertion (de b) si $a \neq b$. On a donc $d(ya, zb) \leqslant \min\big[d(y, zb) + 1,\ d(ya, z) + 1,\ d(y, z) + 2\delta(a \neq b)\big]$.

Inversement, $d(ya, zb)$ est le coût minimal d'une correspondance f entre ya et zb. On note $\alpha = |y|+1$ et $\beta = |z| + 1$. Soit $f_{y,z}$ la correspondance de y à z dont le graphe est obtenu à partir de celui de f en enlevant les couples de la forme (α, j) et (i, β), $1 \leqslant i \leqslant \alpha$, $1 \leqslant j \leqslant \beta$. On a alors les possibilités suivantes :

- *α a une image et β a un antécédent. Comme f est croissante, on a $f(\alpha) = \beta$. Comme f est une correspondance, on a $y_\alpha = z_\beta$ soit $a = b$. Or $c(f_{y,z}) = c(f)$. D'après la question précédente, $d(y, z) \leqslant c(f_{y,z})$. On a donc $d(y, z) \leqslant d(ya, zb)$, soit $d(y, z) + 2\delta(a \neq b) \leqslant d(ya, zb)$.*
- *α n'a pas d'image et β n'a pas d'antécédent. Dans ce cas, $c(f_{y,z}) = c(f) - 2$, donc $d(y, z) \leqslant c(f) - 2 = d(ya, zb) - 2$, d'où encore $d(y, z) + 2\delta(a \neq b) \leqslant d(ya, zb)$.*
- *α n'a pas d'image et β a un antécédent i. Dans ce cas, la restriction de f à $\{1, \ldots, |y|\}$ est une correspondance $f_{y,zb}$ entre y et zb, et $c(f_{y,zb}) = c(f) - 1$. On a donc $d(y, zb) \leqslant c(f_{y,zb}) = c(f) - 1 = d(ya, zb) - 1$, soit $d(y, zb) + 1 \leqslant d(ya, zb)$.*
- *β n'a pas d'antécédent et $f(\alpha) = j$. Symétrique du précédent, ce cas donne $d(ya, z) + 1 \leqslant d(ya, zb)$.*

On a bien finalement $d(ya, zb) = \min\big[d(y, zb) + 1,\ d(ya, z) + 1,\ d(y, z) + 2\delta(a \neq b)\big]$. □

b) Décrire un algorithme pour calculer $d(y, z)$ en temps $O(|y||z|)$. Montrer qu'on peut se contenter de $O(\min(|y|, |z|))$ emplacements en mémoire pour stocker des valeurs intermédiaires durant le calcul.

Solution *On reprend l'écriture de y et z de la question III.2. Pour tout mot x, on a $d(\varepsilon, x) = d(x, \varepsilon) = |x|$, donc $d(y_1 \cdots y_i, \varepsilon) = i$ et $d(\varepsilon, z_1 \cdots z_j) = j$. D'après III.3, $d(y_1 \cdots y_i, z_1 \cdots z_j)$ se calcule en fonction de $d(y_1 \cdots y_{i-1}, z_1 \cdots z_j)$, $d(y_1 \cdots y_i, z_1 \cdots z_{j-1})$ et $d(y_1 \cdots y_{i-1}, z_1 \cdots z_{j-1})$ en temps constant. Or on a $d(y, z) = d(y_1 \cdots y_p, z_1 \cdots z_q)$.*

Un algorithme naturel fonctionnant en temps $O(|y||z|)$ utilise un tableau à deux dimensions, les entrées allant de $(0, 0)$ à $(|y|, |z|)$. L'algorithme calcule à l'entrée (i, j) la valeur de $d(y_1 \cdots y_i, z_1 \cdots z_j)$. Il a besoin pour cela uniquement des entrées $(i, j-1)$, $(i-1, j)$ et $(i-1, j-1)$. Il peut donc

- *initialiser les entrées $(0, j)$ à la valeur j et les entrées $(i, 0)$ à la valeur i.*
- *pour i allant de 1 à $|y|$, calculer la ligne i en faisant varier j de 1 à $|z|$.*

En procédant dans cet ordre, on connaît toujours les entrées en $(i, j-1)$, $(i-1, j)$ et $(i-1, j-1)$ quand on calcule l'entrée en (i, j).

Cet algorithme utilise un tableau de $(|y|+1)(|z|+1)$ cases. Il est facile de voir qu'on peut le modifier de manière à ce qu'il ne se contente que de deux lignes de ce tableau : la dernière ligne complètement calculée et la ligne en cours de calcul. Ainsi, il peut se contenter de $O(|z|)$ emplacements en mémoire. Le problème étant symétrique en y et z, on peut en fait écrire un algorithme qui travaille avec $O(\min(|y|, |z|))$ emplacements temporaires. □

III.4 On rappelle que u est un *sous-mot* de v si l'on peut écrire $u = u_1 \cdots u_k$ et $v = u'_0 u_1 u'_1 \cdots u_k u'_k$, avec $u_i \in A$ et $u'_i \in A^*$, *i.e.*, si la suite des lettres de u est une suite extraite de la suite des lettres de v.

Soient y et z deux mots. Montrer que l'on peut calculer la longueur du plus long sous-mot commun à y et z en temps $O(|y||z|)$ et en utilisant $O(\min(|y|, |z|))$ emplacements en mémoire pour stocker des valeurs intermédiaires durant le calcul.

Solution *Il suffit de remarquer que la longueur du plus long sous-mot commun à y et z est le rang d'une correspondance de coût minimal entre y et z, et vaut donc $\frac{1}{2}(|y| + |z| - d(y, z))$. On se ramène ainsi à la question précédente. On peut d'ailleurs adapter l'algorithme pour qu'il fournisse l'un des plus longs sous-mots communs à y et z toujours en temps $O(|y||z|)$.* □

IV Plus long facteur commun, plus long sous-mot commun

La définition d'un sous-mot est donnée en III.4. On considère les problèmes PLFC (Plus Long Facteur Commun) et PLSMC (Plus Long Sous-Mot Commun) suivants :

PLFC **Donnée :** Un alphabet A fini, un ensemble E de mots de A^*, et un entier $k \geqslant 1$.
Question : Existe-t-il un facteur commun w à tous les mots de E de longueur $|w| \geqslant k$?

PLSMC **Donnée :** Un alphabet A fini, un ensemble E de mots de A^*, et un entier $k \geqslant 1$.
Question : Existe-t-il un sous-mot commun w à tous les mots de E de longueur $|w| \geqslant k$?

IV.1 Donner un algorithme polynomial pour résoudre PLFC.

Solution *Un algorithme polynomial simple est basé sur le fait que la recherche de motif dans un texte est linéaire. On pose $E = \{u, u_1, \ldots, u_s\}$, où u est choisi arbitrairement (par exemple de longueur minimale). Pour chaque facteur x de longueur k de u, on recherche x dans chaque mot de $E\backslash\{u\}$, recherche qui se fait en temps $O(|x| + |u_1| + \cdots + |u_s|)$ (la phase de calcul des bords de x n'est à faire qu'une seule fois). Comme il y a moins de $|u|$ facteurs de longueur k dans u, on obtient un algorithme qui travaille en temps $O(|u||E|)$, où $|E|$ est la somme des longueurs des mots de E.* □

IV.2 Montrer que PLSMC est dans la classe NP.

Solution *Un algorithme* NP *permettant de résoudre* PLSMC *consiste à deviner un sous mot w de longueur k et à vérifier que w est sous-mot de chaque $u \in E$. La vérification du fait que w est sous-mot de u se fait en temps polynomial. Il suffit par exemple que u soit accepté par l'automate suivant :*

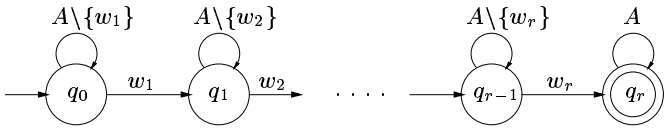

FIG. 10: *Reconnaissance des mots admettant $w = w_1 \cdots w_r$ comme sous-mot*

On peut calculer cet automate en temps $O(|w|)$ et le représenter en espace $O(|w|)$, indépendemment de $|A|$. L'information portée par les deux transitions partant de chaque état non final est en effet redondante. Par exemple, on peut ajouter pour chaque lettre w_i de w une nouvelle lettre $\overline{w}_i$, et chaque transition étiquetée $A\backslash\{w_i\}$ (donc habituellement par $|A| - 1$ transitions) peut être représentée symboliquement par une transition étiquetée $\overline{w}_i$. Lors de la lecture d'un mot u dans l'automate, une transition se fait alors simplement en comparant la lettre courante de u avec la lettre de w sortant de l'état courant. □

Étant donné un graphe non orienté $\mathscr{G} = (\mathscr{S}, \mathscr{A})$, l'ensemble $\mathscr{S} = \{s_1, \ldots, s_p\}$ désigne son ensenble de sommets, et l'ensemble $\mathscr{A}$ son ensemble d'arêtes. On notera $\{s_i, s_j\}$ une arête entre les deux sommets s_i et s_j, avec la convention $i < j$.

On considère le problème suivant, que l'on note CS (Couverture de Sommets). On sait que le problème CS est NP-complet :

CS **Donnée :** Un entier ℓ et un graphe non orienté $\mathscr{G} = (\mathscr{S}, \mathscr{A})$.
Question : Existe-t-il un sous-ensemble $\mathscr{S}' \subseteq \mathscr{S}$ de sommets tel que
– $|\mathscr{S}'| = \ell$, et
– pour chaque arête $\{u, v\}$ de $\mathscr{A}$ on a $u \in \mathscr{S}'$ ou $v \in \mathscr{S}'$ (ou les deux).

Soit $(\ell, \mathscr{G})$ une instance de CS. On note $\mathscr{S} = \{s_1, \ldots, s_p\}$ l'ensemble des sommets de $\mathscr{G}$, et $\mathscr{A} = \{a_1, \ldots, a_q\}$ l'ensemble de ses arêtes. On construit une instance (A, E, k) de PLSMC de la façon suivante.

- L'alphabet A est $\mathscr{S} = \{s_1, \ldots, s_p\}$.
- L'ensemble de mots E sur l'alphabet A contient $|\mathscr{A}| + 1$ mots : $E = \{u, u_1, \ldots, u_q\}$ où :

$$u = s_1 \cdots s_p,$$
$$u_r = \underbrace{s_1 \cdots s_{i-1} s_{i+1} \cdots s_p}_{\text{tous les sommets sauf } s_i} \cdot \underbrace{s_1 \cdots s_{j-1} s_{j+1} \cdots s_p}_{\text{tous les sommets sauf } s_j} \text{ pour chaque arête } a_r = \{s_i, s_j\} \ (i < j).$$

- $k = |\mathscr{S}| - \ell$.

IV.3 Montrer que le graphe $\mathscr{G}$ a une couverture de sommets de taille ℓ si et seulement si les mots de l'ensemble E ont un sous-mot commun de longueur k.

Solution *Supposons que $\mathscr{G}$ a une couverture de sommets de taille ℓ. Il existe donc un sous-ensemble $\mathscr{S}' \subseteq \mathscr{S}$ de sommets tel que $|\mathscr{S}'| = \ell$, et pour chaque arête $\{u, v\}$ de $\mathscr{A}$ on a $u \in \mathscr{S}'$ ou $v \in \mathscr{S}'$. Notons $B = \mathscr{S} \backslash \mathscr{S}'$, et considérons le mot $\tilde{u}$ obtenu à partir de u par suppression (au sens de III) des lettres de $\mathscr{S}'$. On a $\tilde{u} \in B^*$ et $|\tilde{u}| = k$. Il est clair que $\tilde{u}$ est un sous-mot de u. Soit maintenant u_r un mot de E. L'arête $a_r = \{s_i, s_j\}$ a au moins l'un de ses sommets dans $\mathscr{S}'$. Or si $s_i \in \mathscr{S}'$, alors $\tilde{u}$ est sous-mot de $s_1 \cdots s_{i-1} s_{i+1} \cdots s_p$, et si $s_j \in \mathscr{S}'$, alors $\tilde{u}$ est sous-mot de $s_1 \cdots s_{j-1} s_{j+1} \cdots s_p$. Donc $\tilde{u}$ est sous-mot de u_r. Finalement, tous les mots de E admettent $\tilde{u}$ comme sous-mot de longueur k.*

Réciproquement, soit $\tilde{u}$ un sous-mot de longueur k commun à tous les mots de E. En particulier, $\tilde{u}$ est sous-mot de u, donc $\tilde{u}$ est de la forme $s_{i_1} \cdots s_{i_m}$ avec $i_1 < \cdots < i_m$. Notons $\mathscr{S}'$ l'ensemble des sommets qui n'apparaissent pas dans l'écriture de $\tilde{u}$. On a $|\mathscr{S}'| = |u| - |\tilde{u}| = |\mathscr{S}| - k = \ell$. Montrons que toute arête de $\mathscr{G}$ admet un sommet dans $\mathscr{S}'$. Soit $a_r = \{s_i, s_j\} \in \mathscr{A}$, et supposons par l'absurde que ni s_i ni s_j n'est dans $\mathscr{S}'$. Alors $s_i s_j$ est un sous-mot de $\tilde{u}$, mais comme $\tilde{u}$ est un sous-mot de u_r par hypothèse, $s_i s_j$ est un sous-mot de u_r. Or c'est clairement faux. Donc $\mathscr{S}'$ est bien un ensemble de ℓ sommets couvrant $\mathscr{G}$. □

IV.4 Montrer que la fonction qui à l'instance $(\ell, \mathscr{G})$ de CS associe l'instance (A, E, k) de PLSMC définit une réduction polynomiale de CS à PLSMC.

Solution *Pour construire tous les mots u_i, il suffit de parcourir une fois le graphe, puisqu'à chaque arête correspond un mot u_i. Un tel parcours se fait en temps polynomial. De plus, la taille (en nombre de lettres) de l'ensemble de mots construit est inférieure à $2|\mathscr{S}|(|\mathscr{A}| + 1)$ et ces mots peuvent être facilement construits en temps $O(|\mathscr{S}||\mathscr{A}|)$. Il est donc clair que la fonction qui à l'instance $(\ell, \mathscr{G})$ de* CS *associe l'instance (A, E, k) de* PLSMC *se calcule en temps polynomial. D'après la question précédente, c'est une réduction de* CS *à* PLSMC. □

IV.5 Que peut-on déduire des questions précédentes ?

Solution *D'après IV.4, il y a une réduction polynomiale de* CS, *qui est* NP*-complet, à* PLSMC. *On en déduit que* PLSMC *est* NP*-dur. D'autre part, le problème* PLSMC *est dans la classe* NP *d'après IV.2. Donc* PLSMC *est* NP*-complet.* □