Corrigé de la composition d'informatique-mathématiques

ENS - MP - 2017

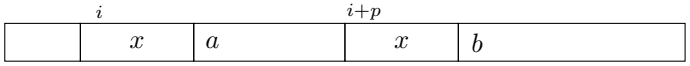
Partie I - Existence d'un carré

1 Les valeurs renvoyées sont 3, 2, 2 et 1.

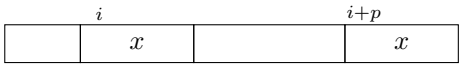
2 La valeur renvoyée j_{final} est le plus grand entier j vérifiant :

$$j \leq p,\ i+j+p \leq |m| \text{ et } m_i m_{i+1} \dots m_{i+j-1} = m_{i+p} m_{i+p+1} \dots m_{i+p+j-1}.$$

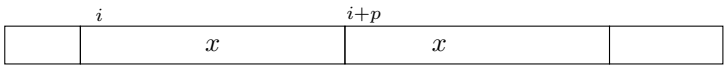
Trois cas sont possibles :

$m =$ 

avec $|x| = j_{final} < p$ et $a \neq b$

$m =$ 

avec $|x| = j_{final} < p$

$m =$ 

avec $|x| = j_{final} = p$

La fonction renvoie donc la valeur p si et seulement si m contient un carré de période p à la position i.

3 Il suffit de tester, pour chaque position i possible ($0 \leq i \leq |m| - 2p$), si m contient un carré de période p à la position i. Le booléen `resultat` contient la valeur `false` au début du calcul et prend la valeur `true` dès qu'un carré est détecté, ce qui permet de sortir de la boucle dès que la réponse à la question est connue.

```
let carre_pe m p =
   let reponse = ref false in
   let i = ref 0 in
   while (not !reponse) && !i + 2*p <= vect_length m do
      reponse := (carre_pe_pos m p !i) = p ;
      i := !i +1
   done;
   !reponse;;
```

4 On teste cette fois si m contient un carré de période p, pour p variant de 1 à la partie entière de $|m|/2$, en arrêtant une nouvelle fois le calcul dès qu'un carré a été détecté :

```
let carre_naif m =
   let reponse = ref false in
   let p = ref 1 in
   while (not !reponse) && 2* !p <= vect_length m do
      reponse := carre_pe m !p ;
      p := !p +1
   done;
   !reponse;;
```

5 Il me semble difficile de répondre avec précision à cette question. La fonction `carre_pe_pos` a un temps de calcul égal à 1 de plus que la valeur qu'elle renvoie. Quand on l'applique à un mot sans carré, ce temps de calcul peut donc varier entre 1 et p et on ne peut pas être dans le pire des cas pour un même mot m arbitrairement long et pour toutes valeurs de i et de p. Si on applique la fonction `carre_naif` à un mot sans carré m, le pire des cas aura un temps de calcul T vérifiant :

$$\sum_{p=1}^{|m|/2}\left(\sum_{i=0}^{|m|-2p} 1\right) \leq T \leq \sum_{p=1}^{|m|/2}\left(\sum_{i=0}^{|m|-2p} p\right)$$

ce qui donne $T = \Omega(|m|^2)$ et $T = O(|m|^3)$. Cet encadrement suffit, puisque nous allons construire dans la suite du problème une fonction qui effectue le travail avec une complexité $O(|m| \times log(|m|))$, qui est négligeable devant $|m|^2$.

6 Les mots sans carré de longueur au plus 3 sur $\{a, b\}$ sont les mots ε, a, b, ab, ba, aba et bab. Pour qu'un mot m de longueur ≥ 4 soit sans carré, il doit commencer par un mot sans carré de longueur 3, ce qui donne comme seules possibilités $m = abaa...$, $m = abab...$, $m = baba...$ et $m = baaa...$: dans tous les cas, m contient un carré. Tout mot de longueur au moins égale à 4 contient donc un carré.

```
let carre_deux_lettres m = match (vect_length m) with
   | n when n <= 1 -> true
   | 2 -> m.(0) <> m.(1)
   | 3 -> m.(0) <> m.(1) && m.(0)=m.(2)
   | _ -> false;;
```

Partie II - Construction d'un mot infini sans carré

7 On trouve facilement $M = 00100110\ldots$ et M contient un carré de période 1 à la position 0 et un carré de période 3 à la position 0.

8 Deux entiers congrus modulo 2^k ont les mêmes k premiers bits dans leurs représentations binaires. Comme l'écriture binaire de $2^{k-1} - 1$ est $(\underbrace{1, \ldots, 1}_{k-1 \text{ bits}}, 0, 0\ldots)$, celle de $i(d, k)$ est de la forme $(\underbrace{1, \ldots, 1}_{k-1 \text{ bits}}, 0, b_0, b_1, \ldots)$, et donc $p(i(d, k)) = k - 1$.

9 Quand on additionne $i(d, k) \sim (\underbrace{1, \ldots, 1}_{k-1 \text{ bits}}, 0, b_0, b_1, \ldots)$ et $2^k \sim (\underbrace{0, \ldots, 0}_{k-1 \text{ bits}}, 0, 1, 0, \ldots)$, on obtient :

$$i(d, k) + 2^k \sim \begin{cases} (\underbrace{1, \ldots, 1}_{k-1 \text{ bits}}, 0, 1, b_1, b_2, \ldots) & \text{si } b_0 = 0 \\ (\underbrace{1, \ldots, 1}_{k-1 \text{ bits}}, 0, 0, ?, ?, \ldots) & \text{si } b_0 = 1 \end{cases}$$

On en déduit que $c(i(d, k) + 2^k) = 1$ si $b_0 = 0$ et $c(i(d, k) + 2^k) = 0$ si $b_0 = 1$, ce qui est le résultat demandé, puisque $b_0 = c(i(d, k))$.

10 Si d est une position quelconque, nous avons montré que le mot M ne contient pas de carré de période $p = 2^k$ à la position d, puisque nous avons construit un indice i (égal à $i(d, k)$) tel que $d \leq i < d+p$ et $c(i) \neq c(i+p)$.

11 Soit p un entier pair non nul, que l'on peut écrire sous la forme $p = 2^k q$ où q est un entier impair et $k \geq 1$. Pour $d \geq 0$, il existe un unique $i(d,p)$ dans $\{d, d+1, \ldots, d+p-1\}$ congru à $2^{k-1}-1$ modulo p. On peut alors reprendre pas à pas la preuve précédente, puisque $i(d,p)$ est également congru à $2^{k-1}-1$ modulo 2^k.

12 Considérons l'alphabet à 4 lettres $A = \{a,b,c,d\}$ et identifions les lettres a,b,c,d aux mots $00, 01, 10, 11$. Le mot M est alors identifié à un mot infini M' sur l'alphabet A : $M = (00)(10)(01)(10)\ldots$ donne $M' = acbc\ldots$; plus précisément, en notant $\phi : A^* \to \{0,1\}^*$ le morphisme de monoïde défini par $\varphi(a) = 00$, $\varphi(b) = 01$, $\varphi(c) = 10$ et $\varphi(d) = 11$, M' est l'unique mot sur A dont "l'image par φ" est égale à M.

Si le mot M' contenait un carré de période p, M contiendrait un carré de période $2p$, ce qui est absurde : M est donc un mot sans carré.

Partie III - Recherche efficace des carrés

13 On obtient facilement :

i	0	1	2	3	4	5	6	7	8	9	10
v_i	c	b	a	c	b	a	b	c	b	a	b
$lms(u,v,i)$	0	1	0	0	1	0	6	0	1	0	4
$lmp(v,v,i)$	11	0	0	3	0	0	0	3	0	0	0

14 Les mots $v_0 \ldots v_{i-p}$ et $v_{j+1} \ldots v_i$ ont même longueur donc $j = p-1$. Nous avons alors :

$$u_{|u|-2p+i+1} \ldots u_{|u|-1} v_0 \ldots v_{i-p} = w = v_{i-p+1} \ldots v_j v_{j+1} \ldots v_i$$

donc $u_{|u|-2p+i+1} \ldots u_{|u|-1} = v_{i-p+1} \ldots v_j$ en simplifiant à droite par le mot $v_0 \ldots v_{i-p} = v_{j+1} \ldots v_i$.

15 Supposons qu'il existe un carré w de période p dans uv se terminant à la position i de v et centré sur v. Par définition d'un carré, $p \geq 1$. D'autre part, le carré apparaît à la position $i' = |u| - 2p + i + 1$ sur le mot uv, avec $i' + p > |u|$, ce qui donne $p < i+1$, soit $p \leq i < |v|$. Enfin, ww étant à cheval sur u et v, $|u| - 2p + i + 1 \leq |u| - 1$, et donc $i \leq 2p-2$, soit $i < 2p-1$.

Nous avons ensuite démontré à la question précédente que $v_0 \ldots v_{i-p}$ est un préfixe de v qui apparaît dans v en commençant à la position $j+1 = p$, donc $lmp(v,v,p) \geq i - p + 1$, soit $i \leq p + lmp(v,v,p) - 1$.

De même, $u_{|u|-2p+i+1} \ldots u_{|u|-1}$ est un suffixe de u qui apparaît dans v et se termine à la position $j = p-1$, donc $lms(u,v,p-1) \geq 2p - i - 1$, soit $2p - lms(u,v,p-1) - 1 \leq i$.

Supposons réciproquement que a), b) et c) sont vérifiées. Nous avons alors $i \leq p + \underbrace{lmp(v,v,p)}_{\leq |v|-p} - 1 \leq |v| - 1$

donc i est bien une position dans v.

Comme $i - p + 1 \geq 1$, on peut écrire

$$v = v_0 \ldots v_{i-p} v_{i-p+1} \ldots v_i v_{i+1} \ldots v_{|v|-1}$$

On a également $|u| > |u| - 2p + i - 1 \geq lms(u,v,p-1) - 2p + i - 1 \geq 0$, donc peut écrire :

$$uv = u_0 \ldots u_{|u|-2p+i} \underbrace{u_{|u|-2p+i+1} \ldots u_{|u|-1} v_0 \ldots v_{i-p}}_{=w} \underbrace{v_{i-p+1} \ldots v_{p-1} v_p \ldots v_i}_{=w'} v_{i+1} \ldots v_{|v|-1}$$

Les conditions $2p-i-1 \leq lms(u,v,p-1)$ et $i-p+1 \leq lmp(v,v,p)$ donnent respectivement

$$u_{|u|-2p+i+1} \dots u_{|u|-1} = v_{i-p+1} \dots v_{p-1} \text{ et } v_0 \dots v_{i-p} = v_p \dots v_i$$

donc $w = w' \neq \varepsilon$ et uv contient un carré de période p se terminant à la position i de v et centré sur v.

16 Pour utiliser des notations identiques à celles du schéma précédent, on peut noter $i+1$ la longueur du suffixe de u qui apparaît dans un nouveau carré centré sur u. Nous obtenons alors le nouveau schéma :

$$u_0 \dots u_{|u|-i-2} \underbrace{u_{|u|-i-1} \dots u_{|u|+p-i-2}}_{\text{mot } w} \underbrace{u_{|u|+p-i-1} \dots u_{|u|-1} v_0 \dots v_{2p-i-2}}_{\text{mot } w} v_{2p-i-1} \dots v_{|v|-1}$$

ce qui donne comme ci-dessus que uv contient un carré centré sur u si et seulement s'il existe deux entiers p et i vérifiant :

(a) $1 \leq p \leq |u|$, et

(b) $p \leq i < 2p-1$, et

(c') $2p - lmp(v,u,|u|-p) - 1 \leq i \leq p + lms(u,u,|u|-p-1) - 1$

On aurait pu retrouver plus rapidement ces conditions en remarquant que, en notant $\tilde{x}$ l'image miroir d'un mot x, uv contient un carré centré sur u si et seulement si $\tilde{v}\tilde{u}$ contient un carré centré sur $\tilde{u}$. La condition (c) devient alors

(c") $2p - lms(\tilde{v},\tilde{u},p-1) - 1 \leq i \leq p + lmp(\tilde{u},\tilde{u},p) - 1$

et l'on retrouve la condition (c') en remarquant que pour tous mots x et y et pour tout entier $0 \leq i < |y|$, $lms(\tilde{x},\tilde{y},i) = lmp(x,y,|y|-1-i)$.

17 Chercher les nouveaux carrés de période p centrés sur v revient à chercher les entiers i vérifiant $7 \leq i \leq 12$ et $7 \leq i \leq 9$. On en déduit qu'il existe trois tels carrés, qui se terminent respectivement aux positions 7, 8 et 9 de v :

$$uv = cab(acbabcb)(acbabcb)ab = caba(cbabcba)(cbabcba)b = cabac(babcbac)(babcbab).$$

18 Tout d'abord, pour que la condition (b) soit vérifiée, il faut que $p \geq 2$. Il faut ensuite avoir

$$2p - lms(u,v,p-1) \leq p + lmp(v,v,p) - 1,$$

c'est-à-dire que $p \leq lms(u,v,p-1) + lmp(v,v,p)$. Ensuite, pour p tel que $2 \leq p < |v|$, il existe un i vérifiant (b) et (c) si et seulement si les intervalles

$$I_1 = [\![p, 2p-2]\!] \text{ et } I_2 = [\![2p - lms(u,v,p-1) - 1, p + lmp(v,v,p) - 1]\!]$$

ne sont pas disjoints. Comme :

$$\begin{aligned} I_1 \cap I_2 = \emptyset &\iff [2p-2 < 2p - lms(u,v,p-1) - 1] \text{ ou } [p + lmp(v,v,p) - 1 < p] \\ &\iff lms(u,v,p-1) = 0 \text{ ou } lmp(v,v,p) = 0 \end{aligned}$$

on en déduit la CNS pour qu'il existe un nouveau carré centré sur v de période p :

$$2 \leq p < |v|,\ p \leq lms(u,v,p-1) + lmp(v,v,p),\ lms(u,v,p-1) \neq 0 \text{ et } lmp(v,v,p) \neq 0.$$

On lit par exemple dans le tableau de la question 13 que seule la valeur $p = 7$ convient dans le cas particulier étudié :

i	0	1	2	3	4	5	6	**7**	8	9	10
v_i	c	b	a	c	b	a	b	c	b	a	b
$lms(u,v,i)$	0	1	0	0	1	0	**6**	0	1	0	4
$lmp(v,v,i)$	11	0	0	3	0	0	0	**3**	0	0	0

avec $7 \leq 6+3$

Il est ainsi possible de tester l'existence d'un tel nouveau carré en faisant varier p de 2 à $|v|-1$ et en stoppant le calcul dès que l'on a $lms(u,v,p-1) \neq 0$, $lmp(v,v,p) \neq 0$ et $p \leq lms(u,v,p-1) + lmp(v,v,p)$:

```
let nouveau_carre_v u v =
   let lms = calcul_lms u v in
   let lmp = calcul_lmp v v in
   let p = ref 2 and reponse = ref false in
   while (not !reponse) && !p < vect_length v do
      reponse := lms.(!p-1) <> 0 && lmp.(!p) <> 0 && !p <= lms.(!p-1) + lmp.(!p);
      p := !p +1
   done;
   !reponse;;
```

La complexité est en $\mathcal{O}(|u|+|v|) + \mathcal{O}(|v|) = \mathcal{O}(|u|+|v|)$.

19 Un nouveau carré de période p non centré sur u ou sur v apparaît si et seulement si $lms(u,v,p-1) = p$, puisqu'alors $v_0 \dots v_{p-1} = u_{|u|-p} \dots u_{|u|-1}$ est le plus long suffixe de u qui apparaît dans v en se terminant à la position $p-1$. Nous en déduisons la fonction :

```
let nouveau_carre u v =
   let lms = calcul_lms u v in
   let p = ref 1 and reponse = ref false in
   while (not !reponse) && !p < vect_length v do
      reponse := (lms.(!p-1) = !p) ;
      p := !p +1
   done;
   !reponse || (nouveau_carre_u u v) || (nouveau_carre_v u v);;
```

La complexité est de $\mathcal{O}(|u|+|v|)$ pour le calcul de `lms`, $\mathcal{O}(|v|)$ pour la boucle de longueur au plus $|v|$, $\mathcal{O}(|u|+|v|)$ pour le calcul (éventuel) de `nouveau_carre_u u v` et $\mathcal{O}(|u|+|v|)$ pour le calcul (éventuel) de `nouveau_carre_v u v`, d'où une complexité en $\mathcal{O}(|u|+|v|)$.

20 Il reste à mettre en place la stratégie de type diviser pour régner : si le mot m est de longueur 0 ou 1, on renvoie `true`. Sinon, on écrit $m = uv$ avec u de longueur égale à la partie entière de $|m|/2$ et on renvoie le booléen

`(carre u)` ou `(carre v)` ou `(nouveau_carre u v)`

(l'évaluation paresseuse permet de ne pas calculer inutilement `carre v` et/ou `(nouveau_carre u v)`). Cela donne :

```
let rec carre m = match vect_length m with
   | n when n < 2 -> false
   | n -> let u = sub_vect m 0 n/2 and v = sub_vect m n/2 (n-n/2) in
              (carre u) || (carre v) || (nouveau_carre u v) ;;
```

Le temps de calcul $T(n)$ dans le pire des cas (n est la longueur du mot m) vérifie la relation de récurrence :

$$\forall n \geq 2, T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \mathcal{O}(n)$$

le terme $\mathcal{O}(n)$ contenant les temps de création des vecteurs u et v et de calcul de `(nouveau_carre u v)`. On reconnaît la récurrence du tri fusion, qui donne $T(n) = \mathcal{O}(n\,log(n))$. La fonction carré a donc une complexité en $\mathcal{O}(|m| \times log(|m|))$.

21 Sans réfléchir, on remplit le tableau :

i	0	1	2	3	4	5	6	7	8	9
v_i	a	a	b	a	a	b	a	a	a	b
$pref_i$	10	1	0	5	1	0	2	3	1	0
g_i			2	2	8	8	8	8	10	10
f_i			1	$\begin{matrix}1\\2\end{matrix}$	3	3	3	$\begin{matrix}3\\6\end{matrix}$	7	7

22 a. Le mot $v_f \dots v_{g-1}$ est le plus long préfixe de v commençant à la position f, donc $v_f \dots v_{g-1} = v_0 \dots v_{g-f-1}$ et, si $g < |v|$, $v_g \neq v_{g-f}$.

22 b. Comme $g \geq i - 1 + pref_{i-1} \geq i - 1$, on a $g = i - 1$ quand $g < i$.

23 Nous avons $v_f \dots v_i \dots v_{g-1} = v_0 \dots v_{i-f} \dots v_{g-f-1}$ et $v_{i-f} \dots v_{i-f+k-1} = v_0 \dots v_{k-1}$.

1er cas : $k < k'$. Nous avons alors $f \leq i + k < g$, donc $v_i \dots v_{i+k} = \underbrace{v_{i-f} \dots v_{i+k-f-1}}_{v_0 \dots v_{k-1}} \underbrace{v_{i+k-f}}_{\neq v_k}$, ce qui prouve que $pref_i = k = pref_{i-f}$.

2ème cas : $k > g - i$. Nous avons cette fois $i - f \leq g - f - 1 < i - f + k - 1$, donc

$$m = v_i \dots v_{g-1} = v_{i-f} \dots v_{g-f-1} = v_0 \dots v_{k'-1}$$

De plus, $g < i + k \leq i < |v|$, donc g est une position dans v et $v_g \neq v_{g-f}$ comme dit à la question a. Enfin, $i - f \leq g - f \leq i - f + k - 1$, donc $v_g \neq v_{g-f} = v_{g-f-i+f} = v_{k'}$; m est donc le plus long préfixe de v commençant à la position i, ce qui prouve que $pref_i = |m| = k' = g - i$.

3ème cas : $k = k'$. Nous avons toujours $v_i \dots v_{g-1} = v_{i-f} \dots v_{g-f-1} = v_0 \dots v_{k-1}$; les préfixes de v commençant à la position i seront donc les mots de la forme $v_i \dots v_{g-1} w$ où w est un préfixe de $v_k \dots v_{|v|-1}$ qui apparaît dans $v_g \dots v_{|v|-1}$ à partir de la position 0. La longueur maximale d'un tel mot w est par définition $\ell = lmp(v_k \dots v_{|v|-1}, v_g \dots v_{|v|-1}, 0)$, et donc $pref_i = k + l = g - i + lmp(v_k \dots v_{|v|-1}, v_g \dots v_{|v|-1}, 0)$.

24 Quand $i = 1$, les tests des lignes 8 et 10 ne sont pas validés et on entre dans le bloc $13 - 20$; au début de ce bloc, on pose $f = g = 1$ et on incrémente g tant que $g < |v|$ et $v_g = v_{g-1}$. On sort de la boucle $16 - 18$ avec $g = pref_1 + 1$; au début de la seconde boucle, nous aurons $i = 2$ et la propriété :

$$\mathcal{P}_i : \begin{cases} \forall j \in [\![0, i-1]\!],\ \texttt{pref}.(j) = pref_j \\ g = max\{j + pref_j,\ 0 < j < i\} \\ f \in \{j,\ 0 < j < i \text{ et } j + pref_j = g\} \end{cases}$$

est vérifiée. Les propriétés démontrées à la question précédente prouve que la propriété $\mathcal{P}_i$ est un invariant de boucle, le seul problème étant de vérifier que les variable f et g sont bien modifiées quand elles doivent

l'être. Montrons précisément l'invariance de $\mathcal{P}_i$: supposons que $2 \leq i_0 \leq |v|-1$ et que $\mathcal{P}_i$ est vérifiée quand i prend la valeur i_0. Trois cas sont alors possible pendant le passage de $i = i_0$ à $i = i_0 + 1$:

1er cas : $i_0 < g$ et $pref_{i_0-f} < g - i$. On a bien $pref_{i_0} = pref_{i_0-f}$, donc on affecte à la ligne 8 la valeur $pref_{i_0}$ à $\texttt{pref}.(i_0)$; comme $i_0 + pref_{i_0} = i_0 + pref_{i_0-f} < g$, on a toujours

$$g = max\{j + pref_j,\ 0 < j < i_0 + 1\} \text{ et } f \in \{j,\ 0 < j < i_0 + 1 \text{ et } j + pref_j = g\} :$$

au début de la boucle $i = i_0 + 1$, $\mathcal{P}_i$ est vérifiée.

2ème cas : $i_0 < g$ et $pref_{i_0-f} > g - i$. On affecte une nouvelle fois à la ligne 11 la valeur $pref_{i_0}$ à $\texttt{pref}.(i_0)$; comme $i_0 + pref_{i_0} = g$, on a une nouvelle fois

$$g = max\{j + pref_j,\ 0 < j < i_0 + 1\} \text{ et } f \in \{j,\ 0 < j < i_0 + 1 \text{ et } j + pref_j = g\},$$

ce qui prouve que $\mathcal{P}_i$ est vérifiée au début de la boucle $i = i_0 + 1$.

3ème cas : $i_0 < g$ et $pref_{i_0-f} = g - i$. Notons f_0, g_0 les valeurs des références f et g au début de cette boucle. On a $i_0 + pref_{i_0} = g_0 + \ell \geq g_0$, donc

$$g_1 = max\{j + pref_j,\ 0 < j < i\} = i_0 + pref_{i_0} \text{ et } i_0 \in \{j,\ 0 < j < i_0 + 1 \text{ et } j + pref_j = g_1\} :$$

on affecte donc à la ligne 14 une valeur correcte à f. La ligne 15 ne modifie pas g, puis la boucle $16-18$ permet d'incrémenter g tant que $g < |v|$ et $v_g = v_{g-i_0}$: on sort de cette boucle avec $g = g_0 + \ell = g_1$. Enfin, la ligne 19 permet d'affecter à $\texttt{pref}.(i_0)$ la valeur $g_1 - i_0 = pref_{i_0}$: $\mathcal{P}_i$ est une nouvelle fois vérifiée au début de la boucle $i = i_0 + 1$.

4ème cas : $i_0 \geq g$. On aura une nouvelle fois

$$g_1 = max\{j + pref_j,\ 0 < j < i\} = i_0 + pref_{i_0} \text{ et } i_0 \in \{j,\ 0 < j < i_0 + 1 \text{ et } j + pref_j = g_1\};$$

la ligne 15 donne à g la valeur i_0 et la boucle $16-18$ effectue un simple calcul de $pref_{i_0}$: au sortir de cette boucle, $g = i_0 + pref_{i_0}$ et on conclut comme dans le 3ème cas.

Ces arguments prouvent que l'algorithme termine et renvoie le tableau $pref$. L'analyse de la complexité est un peu plus délicate, car le temps de traitement d'un passage dans la boucle $6-21$ n'est pas constant, ce qui nécessite de faire une analyse amortie (en remarquant que le passage par une ligne du programme demande un temps constant) :

- on ne passe qu'une fois par les lignes $2-5$ et 22 ;
- on passe par la ligne 6 exactement $|u|$ fois ;
- on passe par la ligne 7 exactement $|u|-1$ fois ;
- on passe par les lignes 8, 10, 11, 14, 15 et 19 au plus $|u|-1$ fois.

Le seul problème est l'analyse des lignes $16-18$; on peut remarquer que tout au long du calcul, la variable g prend des valeurs qui augmentent. Chaque passage dans la boucle $16-18$ est suivie soit de l'incrémentation de g (quand $g < |v|$), soit de l'incrémentation de i ; comme $i + g \leq 2|v|$, le nombre total de passage dans la boucle $16-18$ est majoré par $2|v|$ (i varie de 1 à $|u|-1$ et g croit de 0 à au plus $|u|$).

On en déduit que le calcul de $pref$ se fait en temps de l'ordre de la longueur du mot v.

25 Soit $w = u\#v$ et $pref$ la table des préfixes de w (la construction de w puis le calcul de $pref$ se fait en temps de l'ordre de $|u| + |v|$). La lettre v_i apparaît à la position $|u| + i + 1$ dans le mots w et le plus long préfixe de w qui apparaît dans w à partir de la position $|u| + i + 1$ est le plus long préfixe de u qui apparaît dans v en commençant à la position i (comme la lettre # n'apparaît pas dans $\tilde{v}$, un préfixe de w qui apparaît à partir de la position $|u| + i + 1$ est un préfixe de u). On en déduit que $lms(u, v, i) = pref(|u| + i + 1)$, ce qui permet de construire le vecteur lms avec une complexité $\mathcal{O}(|u| + |v|)$:

```
let calcul_lmp u v =
   let n, m = vect_length u, vect_length v in
   let w = make_vect (n+m+1) '#' in
   for i = 0 to n-1 do w.(i) <- u.(i) done ;  (* on copie u dans w *)
   for i = 0 to m-1 do w.(n+i+1) <- v.(i) done ; (* on copie v dans w *)
   sub_vect (calcul_pref w) (n+1) m ;;
```

26 Il est clair que le tableau des suffixes de w est l'image miroir du tableau des préfixes de $\tilde{w}$, ce qui donne :

```
let calcul_suff v =
   let m = vect_length v in
   let w = make_vect  m 'a' in
   for i = 1 to m do w.(i-1) <- v.(m-i) done ;  (* on construit l'image miroir de v *)
   let pref, suff= calcul_pref w, make_vect m 0 in
   for i = 0 to m-1 do suff.(i) <- pref.(m-i-1) done ;
   suff ;;
```

27 La méthode est identique à celle de la question 25, en utilisant la table des suffixes du mot $w = v\#u$:

```
let calcul_lms u v =
   let n, m = vect_length u, vect_length v in
   let w = make_vect (n+m+1) '#' in
   for i = 0 to m-1 do w.(i) <- v.(i) done ;
   for i = 0 to n-1 do w.(m+i+1) <- u.(i) done ;
   sub_vect (calcul_suff w) 0 m ;;
```