ECOLE DES PONTS PARISTECH,
SUPAERO (ISAE), ENSTA PARISTECH,
TELECOM PARISTECH, MINES PARISTECH,
MINES DE SAINT-ETIENNE, MINES DE NANCY,
TELECOM BRETAGNE, ENSAE PARISTECH (FILIERE MP)
ECOLE POLYTECHNIQUE (FILIERE TSI)

CONCOURS 2013

EPREUVE d'INFORMATIQUE

Filière : MP

Durée de l'épreuve : 3 heures.
L'utilisation d'une calculatrice est autorisée.

Sujet mis à la disposition des concours :
CYCLE INTERNATIONAL, ECOLES DES MINES, TELECOM SUDPARIS, TPE-EIVP.

L'énoncé de cette épreuve comporte 14 pages.

Les candidats sont priés de mentionner de façon apparente
sur la première page de la copie :
INFORMATIQUE - MP

Recommandations aux candidats

- **Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.**
- **Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré.**
- **Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.**

Composition de l'épreuve

L'épreuve comporte un seul problème.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Préliminaire concernant la programmation. Il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début de problème le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve**. Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que nécessaire. Par ailleurs, pour écrire une fonction ou une procédure en langage de programmation, le candidat pourra définir des fonctions ou des procédures auxiliaires qu'il explicitera, ou faire appel à d'autres fonctions ou procédures définies dans les questions précédentes.
Dans l'énoncé du problème, un même identificateur écrit dans deux polices de caractères différentes désigne la même entité, mais du point de vue mathématique pour la police écrite en italique (par exemple : p) et du point de vue informatique pour celle écrite en romain (par exemple : `p`).

Le but du sujet est d'étudier plusieurs algorithmes de recherche d'un mot, appelé *motif*, dans un texte.

Un *alphabet* Σ est un ensemble fini d'éléments appelés *lettres*. Un *mot* sur Σ est une suite finie, éventuellement vide, de lettres de Σ ; la *longueur* d'un mot u est le nombre de lettres composant u ; le mot de longueur nulle est noté ε. L'ensemble des mots sur Σ est noté Σ^*. L'alphabet Σ utilisé est un sous-ensemble de l'alphabet usuel composé des 26 lettres de a à z. Si u est un mot, on note $l(u)$ la longueur de u ; la notation u_i désigne la $(i + 1)^{e}$ lettre de u. Le mot u s'écrit donc : $u = u_0u_1...u_{l(u)-1}$. On dit que la lettre u_i est à la *position* i.

On dit qu'un mot f sur Σ est un *facteur* d'un mot u sur Σ s'il existe deux mots v et w sur Σ avec $u = vfw$, où vfw désigne le mot obtenu en concaténant v, f et w ; on appelle alors *position* de f l'indice de u où f commence ; on dit que *f figure dans u à la position $l(v)$*. Si v est le mot de longueur nulle, on dit que f est un *préfixe* de u ; si w est le mot de longueur nulle, on dit que f est un *suffixe* de u. Il peut exister dans un mot plusieurs occurrences d'un même facteur. Le mot de longueur nulle, ε, est considéré comme étant préfixe et suffixe de tout mot.

Dans tout l'énoncé, on considère deux mots t et m sur un alphabet Σ **de longueurs non nulles**, appelés respectivement *texte* et *motif*. On suppose que toute lettre de l'alphabet Σ apparaît au moins une fois dans t.

On veut résoudre le problème (P) suivant :

(P) déterminer la position de chaque occurrence de m dans t.

On supposera toujours qu'on a : $l(m) \leq l(t)$; on ne vérifiera pas cette condition dans la programmation.
Par exemple, quand $m = abab$ et $t = aaababab bbbababbbaa$, m est un facteur de t et figure aux positions 2, 4 et 11.

Indications pour la programmation

Caml

Les mots sont codés en utilisant le type `string` de Caml.
Si `u` est de type `string` et si `i` est un entier correspondant à un indice d'une lettre de `u`, `string_length u` renvoie le nombre de lettres de `u` et `u.[i]` est le caractère se trouvant dans `u` à l'indice `i`.

Fin des indications pour Caml

Téléchargé gratuitement sur www.Doc-Solus.fr .

Pascal
On utilise les définitions suivantes :

```
const MAX_LONGUEUR = 100;
const MAX_SIGMA = 26;

type tab_char = array[0 .. MAX_LONGUEUR - 1] of char;
type tab_int_sigma = array[0 .. MAX_SIGMA - 1] of integer;

type pile = RECORD
   nb : integer;
   table : array[0 .. MAX_LONGUEUR - 1] of integer;
end;
```

La constante `MAX_LONGUEUR` donne la longueur maximum des mots considérés.
La constante `MAX_SIGMA` donne le nombre maximum de lettres de l'alphabet.

Les mots sont toujours codés en utilisant des tableaux de type `tab_char` ; les lettres du mot sont écrites consécutivement dans ce tableau **à partir de l'indice 0**.
Fin des indications pour Pascal

Première partie : algorithme simple

❒ 1 – Soit s un entier positif ou nul **vérifiant la relation** $s + l(m) \leq l(t)$. Il s'agit d'écrire une fonction *est_present* qui permette de savoir si m figure dans t à la position s. La fonction parcourt les lettres du motif m de la gauche vers la droite et arrête la recherche dès que possible.
Caml : Écrire en Caml une fonction nommée `est_present` telle que, si :

- `m` et `t`, de type `string`, codent m et t,
- `s` code l'entier s,

alors `est_present m t s` renvoie le booléen `true` si m figure dans t à la position s et le booléen `false` dans le cas contraire.
Pascal : Écrire en Pascal une fonction nommée `est_present` telle que, si :

- `m` et `t`, de type `tab_char`, contiennent m et t,
- `lm`, de type `integer`, contient la longueur de m,
- `s`, de type `integer`, contient la valeur de s,

alors `est_present(m, lm, t, s)` renvoie le booléen `true` si `m` figure dans `t` à la position `s` et le booléen `false` dans le cas contraire.

❒ 2 – Préciser la complexité de la fonction `est_present` dans le pire cas et le meilleur cas.

❒ 3 – Il s'agit d'écrire une fonction nommée *positions* qui résout le problème (P) en utilisant la fonction *est_present*.
Caml : Écrire en Caml la fonction `positions` telle que si `m` et `t`, de type `string`, codent m et t, alors `positions m t` renvoie une liste contenant les positions de m dans t.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Pascal : Écrire en Pascal la fonction `positions` telle que si :

- `m` et `t`, de type `tab_char`, contiennent *m* et *t*,
- `lm` et `lt`, de type `integer`, contiennent les longueurs de *m* et *t*,

alors `positions(m, lm, t, lt)` renvoie un résultat de type `pile` contenant, dans le champ (ou membre) `nb`, le nombre de positions de *m* dans *t* et, dans le champ `table`, la liste des positions de *m* dans *t*.

❒ 4 – Préciser, pour une valeur de $l(t) \geq 1$ quelconque et $l(m) \leq l(t)$ quelconque, la complexité dans le pire cas de la fonction *positions* ; on exprimera cette complexité en fonction de $l(t)$ et $l(m)$. Donner un exemple de deux mots *m* et *t* pour lesquels cette complexité est atteinte.

Deuxième partie : une amélioration de la méthode précédente

Dans la recherche simple, lorsque la recherche du motif *m* en position *s* est terminée, on poursuit la recherche en position $s + 1$. On peut améliorer cet algorithme en poursuivant si possible la recherche un peu plus loin dans le texte *t* ; pour cela, on suit la méthode décrite plus bas nommée *algorithme amélioré*.

On note nb_Σ le cardinal de Σ. On numérote les lettres de Σ de 0 à $nb_\Sigma - 1$.
On suppose que l'on a défini en langage de programmation une fonction *numero* telle que :

- en **Caml**, si `x`, de type `char`, code une lettre *x* de Σ, `numero x` renvoie le numéro de la lettre *x* dans Σ ;
- en **Pascal**, si `x`, de type `char`, code une lettre *x* de Σ, `numero(x)` renvoie une valeur de type `integer` donnant le numéro de la lettre *x* dans Σ.

On ne demande pas d'écrire en langage de programmation la fonction *numero*. On suppose que cette fonction a une complexité constante.

On introduit un tableau *D* d'entiers défini comme suit. Si l'entier *k* vérifie $0 \leq k \leq nb_\Sigma - 1$, la case d'indice *k* du tableau *D* contient la position de la dernière occurrence de la lettre de numéro *k* dans le motif *m*. Par convention, si cette lettre n'est pas présente dans le motif *m*, cette position est égale à –1. Par exemple, pour $\Sigma = \{a, b, c\}$ et pour le motif *bbaa*, on a $nb_\Sigma = 3$, le numéro de la lettre *a* est 0, celui de la lettre *b* est 1, celui de la lettre *c* est 2 et le tableau *D* contient :

- dans la case d'indice 0, la valeur 3,
- dans la case d'indice 1, la valeur 1,
- dans la case d'indice 2, la valeur –1.

❒ 5 – Il s'agit de construire le tableau *D* en un temps linéaire en la longueur $l(m)$ du motif *m*. On justifiera la complexité linéaire de la construction du tableau *D*.
Caml : Écrire en Caml une fonction nommée `calcul_D` telle que, si :

- `m`, de type `string`, code le motif *m*,
- `nbSigma` contient le nombre de lettres de l'alphabet Σ,

alors `calcul_D m nbSigma` renvoie un vecteur (ou tableau) codant le tableau *D*.
Pascal : Écrire en Pascal une fonction nommée `calcul_D` telle que, si :

- `m`, de type `tab_char`, contient *m*,
- `lm`, de type `integer`, contient la longueur de *m*,
- `nbSigma`, de type `integer`, contient le nombre de lettres de l'alphabet Σ,

Téléchargé gratuitement sur www.Doc-Solus.fr .

alors `calcul_D(m, lm, nbSigma)` renvoie un tableau de type `tab_int_sigma` codant le tableau *D*.

Pour décrire l'algorithme amélioré, on introduit la notion de *fenêtre de recherche*. Cette fenêtre dépend d'un indice ϕ vérifiant $0 \leq \phi \leq l(t) - l(m)$; cette fenêtre contient les $l(m)$ lettres consécutives de *t* comprises entre les indices ϕ et $\phi + l(m) - 1$; ces lettres sont mises en regard avec les $l(m)$ lettres du motif *m*. On dit alors que la fenêtre *est à la position* ϕ.
Dans l'algorithme, la fenêtre se déplace toujours de la gauche vers la droite ; le motif se déplace avec la fenêtre alors que le texte ne bouge pas.
On considère l'exemple défini par :

Exemple1 : *t1 = ababbaaacabbaab*
m1 = bbaa

Au départ de l'algorithme amélioré, $\phi = 0$ et la fenêtre de recherche est en gris sur la figure 1.

Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t1	*a*	*b*	*a*	*b*	***b***	*a*	*a*	*a*	*c*	*a*	*b*	*b*	*a*	*a*	*b*
m1	*b*	***b***	*a*	*a*											

Figure 1 : fenêtre de recherche à la position 0

On examine cette position de la fenêtre en comparant le facteur *abab* de *t1* qui est dans la fenêtre avec le motif *m1* qui est en regard et on constate que ce facteur n'est pas une occurrence de *m1* dans *t1*.
On considère alors la lettre de *t1* qui se trouve à droite de la fenêtre de recherche ; c'est la lettre *b* à l'indice 4 de *t1* en gras sur la figure 1 ; on note *clé* cette lettre. On remarque que si on considère la fenêtre de recherche positionnée à l'indice 1 ou à l'indice 2, la lettre du motif *m1* qui se trouve en regard de *clé* n'est pas la lettre *b*, c'est la lettre *a* dans les deux cas : l'examen de ces deux positions de la fenêtre ne permettrait pas de découvrir une occurrence du motif dans le texte. C'est pourquoi on n'examine pas les positions 1 et 2 de la fenêtre. En revanche, si on met la fenêtre en position 3, *clé* est en regard de la dernière apparition de la lettre *b* dans *m1*, en gras sur la figure 1 ; on retient cette position pour savoir si le motif figure on non à l'indice 3 du texte.

La fenêtre de recherche est maintenant en gris sur la figure 2 :

Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t1	*a*	*b*	*a*	*b*	*b*	*a*	*a*	***a***	*c*	*a*	*b*	*b*	*a*	*a*	*b*
m1				*b*	*b*	*a*	***a***								

Figure 2 : fenêtre de recherche à la position 3

On examine la fenêtre et on constate que *t1* contient une occurrence de *m1* à la position 3.

Maintenant, *clé* est la lettre de *t1* qui se trouve à l'indice 7 en gras sur la figure 2 : il s'agit de la lettre *a*. On déplace la fenêtre pour que *clé* se trouve en regard de la dernière occurrence de la lettre *a* dans *m1*, en gras sur la figure 2 ; on déplace en conséquence la fenêtre d'une seule case, elle se trouve alors à la position 4 et est représentée sur la figure 3. On examine la fenêtre et on constate que *t1* ne contient pas une occurrence de *m1* à l'indice 4.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t1	*a*	*b*	*a*	*b*	*b*	*a*	*a*	*a*	***c***	*a*	*b*	*b*	*a*	*a*	*b*
m1					*b*	*b*	*a*	*a*							

Figure 3 : fenêtre de recherche à la position 4

On continue avec le même principe pour trouver d'autres éventuelles occurrences de *m1* dans *t1*. Maintenant, *clé* est la lettre de *t1* qui se trouve à l'indice 8, en gras sur la figure 3 : il s'agit de la lettre *c*. Le déplacement suivant consistera à décaler la fenêtre vers la droite en n'examinant pas les positions qui mettraient en regard de *clé* une lettre de *m1* différente de la lettre *c*.

❒ 6 – Dans l'exemple de la figure 3, indiquer la prochaine position de la fenêtre de recherche et dire si cette position correspond ou non à une occurrence de *m1* dans *t1*.

❒ 7 – Décrire la suite du déroulement de l'algorithme appliqué à *Exemple1* jusqu'à ce qu'on puisse conclure que toutes les occurrences de *m1* dans *t1* ont été déterminées. On poursuivra les déplacements de la fenêtre vers la droite en utilisant le même principe que précédemment.

❒ 8 – On revient au déroulement de l'algorithme amélioré dans le cas général. On suppose que la fenêtre est à la position ϕ et que l'on a $l(t) > \phi + 2l(m)$. En utilisant le tableau *D* et la fonction *numero* décrits plus haut, indiquer la position suivante de la fenêtre.

❒ 9 – Il s'agit d'écrire une fonction *positions2* qui résout le problème (*P*) en appliquant l'algorithme amélioré.

Caml : Écrire en Caml la fonction `positions2` telle que si :

- `m` et `t`, de type `string`, codent *m* et *t*,
- `nbSigma` contient le nombre de lettres de l'alphabet Σ,

alors `positions2 m t nbSigma` renvoie une liste contenant les positions de *m* dans *t* déterminées selon l'algorithme amélioré.

Pascal : Écrire en Pascal la fonction `positions2` telle que si :

- `m` et `t`, de type `tab_char`, contiennent *m* et *t*,
- `lm` et `lt`, de type `integer`, contiennent les longueurs de *m* et *t*,
- `nbSigma`, de type `integer`, contient le nombre de lettres de l'alphabet Σ,

alors `positions2(m, lm, t, lt, nbSigma)` renvoie un résultat de type `pile` contenant, dans le champ `nb`, le nombre de positions de *m* dans *t* et, dans le champ `table`, la liste des positions de *m* dans *t* déterminées selon l'algorithme amélioré.

❒ 10 – Préciser, en fonction de $l(t)$ et $l(m)$, la complexité dans le meilleur cas de la fonction *positions2*. On ne prendra pas en compte la complexité du calcul du tableau *D*. Donner un exemple qui atteint cette complexité.

Dans toute la suite, on se propose de résoudre le problème (*P*) à l'aide d'automates.

Un *automate* *A* est décrit par un quintuplet (Σ, Q, I, F, T), où :

- Σ est un alphabet ;
- Q est un ensemble fini et non vide appelé *ensemble des états* de *A* ;
- $I \subseteq Q$ est appelé *ensemble des états initiaux* de *A* ;

Téléchargé gratuitement sur www.Doc-Solus.fr .

- $F \subseteq Q$ est appelé *ensemble des états finals* de A ;
- $T \subseteq Q \times \Sigma \times Q$ est appelé l'*ensemble des transitions* ; étant donnée une transition $(p, x, q) \in T$, on dit qu'elle va de l'état p à l'état q et qu'elle est d'étiquette x ; on pourra la noter $p \xrightarrow{x} q$; on dit aussi que p est l'*origine* de la transition et q son *extrémité*.

Un *calcul* de A est une suite de la forme $p_0 \xrightarrow{x_0} p_1 \xrightarrow{x_1} p_2 \dots p_{k-1} \xrightarrow{x_{k-1}} p_k$ où, pour $0 \leq i \leq k-1$, $p_i \xrightarrow{x_i} p_{i+1}$ est une transition ; p_0 est l'*origine* du calcul, p_k son *extrémité*. L'*étiquette* du calcul est le mot $x_0x_1x_2\dots x_{k-1}$ formé par la suite des étiquettes des transitions successives.
Un calcul de A est dit *réussi* si son origine est dans I et son extrémité dans F. Un mot m sur Σ est *reconnu* par A s'il est l'étiquette d'un calcul réussi. Le *langage reconnu* par A, noté $L(A)$, est l'ensemble des mots reconnus par A.

L'automate A est dit *déterministe* si I contient exactement un élément et si, pour tout $p \in Q$ et tout $x \in \Sigma$, il existe au plus un état $q \in Q$ avec $(p, x, q) \in T$.
L'automate A est dit *complet* si, pour tout $p \in Q$ et tout $x \in \Sigma$, il existe au moins un état $q \in Q$ avec $(p, x, q) \in T$.
Si A est un automate déterministe complet, on définit sa fonction de transition δ de $(Q \times \Sigma)$ dans Q par : $\delta(p, x) = q$ si et seulement si $(p, x, q) \in T$. On définit alors, récursivement, une fonction δ^* de $(Q \times \Sigma^*)$ dans Q par :

- si $p \in Q$, $\delta^*(p, \varepsilon) = p$,
- si $p \in Q$, $u \in \Sigma^*$ et $x \in \Sigma$, $\delta^*(p, ux) = \delta(\delta^*(p, u), x)$.

Dans tout le sujet, les automates considérés auront **un seul état initial**. On notera *nbQ* le nombre d'états d'un automate ; les états seront numérotés de 0 à *nbQ* – 1. **L'état initial possédera toujours le numéro 0**. Si p est le numéro d'un état, on dira qu'il s'agit de l'état p, identifiant ainsi un état avec son numéro.

Un automate peut être représenté par un dessin comme il est fait ci-dessous pour représenter l'automate *Aex*, qui est déterministe et non complet.

Aex possède trois états : 0, 1 et 2.
L'état 0 est l'état initial.
Aex possède un seul état final, l'état 2.
Aex possède trois transitions, les transitions $(0, a, 1)$, $(1, a, 1)$ et $(1, b, 2)$.

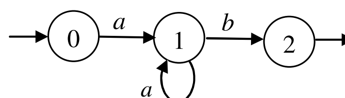

Figure 4 : l'automate *Aex*

Troisième partie : implémentation d'un automate

L'automate *Aex* représenté sur la figure 4 sert d'exemple ci-dessous pour illustrer le codage d'un automate en langage de programmation.

Indications pour Caml
On utilise une constante définie par :
```
let MAX_Q = 100 ;;
```
`MAX_Q` donne le nombre maximum d'états des automates considérés.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Pour représenter l'ensemble des états finals d'un automate, on utilise une liste, de type `int list`, qui contient les numéros des états finals.
Si p est un état d'un automate, une transition d'origine p est codée par un couple de type `(char * int)` contenant l'étiquette de cette transition et le numéro de l'extrémité de cette même transition.
Si p est un état d'un automate, on représente l'ensemble des transitions d'origine p par la liste des transitions d'origine p ; cette liste est de type `(char * int) list`.
L'ensemble des transitions d'un automate A est codé par un vecteur ; si p vérifie les inégalités $0 \le p \le nbQ - 1$, l'élément d'indice p de ce vecteur contient la liste des transitions d'origine p. Ce vecteur est ainsi en Caml de type `(char * int) list vect`.
Pour définir un automate A, on utilise le type enregistrement (type produit) suivant :

```
type automate =   {nbQ : int;
                   F   : int list;
                   T   : (char * int) list vect;
                  };;
```

dans lequel les champs (ou membres) correspondent :

- pour `nbQ`, au nombre d'états de A,
- pour `F`, à la liste des états finals de A,
- pour `T`, à l'ensemble des transitions de A.

L'automate *Aex* de la figure 4 peut être défini par :

```
let T_Aex = make_vect 3 [];;
T_Aex.(0) <- [(`a`,1)];
T_Aex.(1) <- [(`a`,1); (`b`,2)];;

let Aex = {nbQ = 3 ; F = [2]; T = T_Aex};;
```

Fin des indications pour Caml

Indications pour Pascal
On ajoute les définitions suivantes aux définitions données plus haut :

```
const MAX_Q = 100;

type transition = RECORD
   etiquette : char;
   extremite : integer;
end;

type tab_int_Q = array[0 .. MAX_Q - 1] of integer;
type tab_transition = array[0 .. MAX_SIGMA - 1] of transition;
type tab_tab_transition = array[0 .. MAX_Q - 1] of tab_transition;

type automate = RECORD
        nbQ : integer;
        nbF : integer;
        F : tab_int_Q;
        nbT : tab_int_Q;
        T : tab_tab_transition;
end;
```

La constante `MAX_Q` donne le nombre maximum d'états des automates considérés.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Si p est un état d'un automate, une transition d'origine p est codée par un enregistrement de type `transition`, le champ (ou membre) `etiquette` contenant l'étiquette de cette transition et le champ `extremite` le numéro de l'extrémité de cette même transition.
Si p est un état d'un automate, on représente l'ensemble des transitions d'origine p en utilisant un tableau de type `tab_transition` (on ne codera que des automates déterministes avec le type `automate`).
L'ensemble des transitions d'un automate est codé par un tableau de type `tab_tab_transition`, la case d'indice p de ce tableau contenant la liste des transitions d'origine p.
Un automate déterministe A sera codé par un enregistrement de type `automate` dans lequel les champs (ou membres) correspondent :

- pour `nbQ`, au nombre d'états de A ;
- pour `nbF`, au nombre d'états finals de A ;
- pour `F`, à un tableau contenant la liste des états finals de A ;
- pour `nbT`, à un tableau donnant les nombres de transitions issues de chaque état ; plus précisément, si `p` est compris entre 0 et `nbQ - 1`, `nbT[p]` contient le nombre de transitions d'origine p ;
- pour `T`, à l'ensemble des transitions de A.

L'automate Aex de la figure 4 peut être défini par une variable `Aex` de type `automate` avec les instructions :

```
Aex.nbQ := 3;
Aex.nbF := 1;
Aex.F[0]:= 2;
Aex.nbT[0] := 1;
Aex.T[0][0].etiquette := 'a';
Aex.T[0][0].extremite := 1;
Aex.nbT[1] := 2;
Aex.T[1][0].etiquette := 'a';
Aex.T[1][0].extremite := 1;
Aex.T[1][1].etiquette := 'b';
Aex.T[1][1].extremite := 2;
Aex.nbT[2] := 0;
```

Fin des indications pour Pascal

Soit A un automate **déterministe** sur un alphabet Σ.

❒ 11 – Il s'agit de savoir si un état est final ou non. On rappelle qu'un état est identifié avec son numéro.
Caml : Écrire en Caml une fonction nommée `est_final` telle que si :

- `A`, de type `automate`, code l'automate A,
- `p` est un entier codant un état de A,

alors `est_final A p` renvoie le booléen `true` si p est un état final de A et `false` sinon. Indiquer la complexité de cette fonction.
Pascal : Écrire en Pascal une fonction nommée `est_final` telle que si :

- `A`, de type `automate`, code l'automate A,
- `p`, de type `integer`, contient un état de A,

alors `est_final(A, p)` renvoie un booléen qui vaut `true` si p est un état final de A et `false` dans le cas contraire. Indiquer la complexité de cette fonction.

Téléchargé gratuitement sur www.Doc-Solus.fr .

❒ 12 – On suppose que p est un état et x une lettre ; on veut connaître l'état q atteint à partir de l'état p par la transition d'étiquette x si cette transition existe.
On rappelle que le cardinal de l'alphabet est majoré par une constante (égale à 26).
Caml : Écrire en Caml une fonction nommée `etat_suivant` telle que si :

- `A`, de type `automate`, code l'automate A,
- `p` est un entier codant un état de A,
- `x` est une lettre appartenant à Σ,

alors `etat_suivant A p x` renvoie un entier codant l'état q tel que (p, x, q) soit une transition de A si cette transition existe et `-1` sinon. Indiquer la complexité de cette fonction.
Pascal : Écrire en Pascal une fonction nommée `etat_suivant` telle que si :

- `A`, de type `automate`, code l'automate A,
- `p`, de type `integer`, contient le numéro d'un état de A,
- `x`, de type `char`, contient une lettre appartenant à Σ,

alors `etat_suivant(A, p, x)` renvoie une valeur de type `integer` contenant l'état q tel que (p, x, q) soit une transition de A si cette transition existe et `-1` sinon. Indiquer la complexité de cette fonction.

❒ 13 – Il s'agit de déterminer l'état, s'il existe, qui est extrémité du calcul dont l'origine est l'état initial et dont l'étiquette est un mot donné.
Caml : Écrire en Caml une fonction nommée `execution` telle que si :

- `A`, de type `automate`, code l'automate A,
- `u`, de type `string`, code un mot u sur Σ,

alors `execution A u` renvoie un entier codant l'état q qui est l'extrémité du calcul dont l'origine est l'état initial et dont l'étiquette est u, si q existe, et `-1` sinon. Indiquer la complexité de cette fonction.
Pascal : Écrire en Pascal une fonction nommée `execution` telle que si :

- `A`, de type `automate`, code l'automate A,
- `u`, de type `tab_char`, contient un mot u sur Σ,
- `lu`, de type `integer`, contient la longueur de u,

alors `execution(A, u, lu)` renvoie une valeur de type `integer` donnant l'état q qui est extrémité du calcul dont l'origine est l'état initial et dont l'étiquette est u, si q existe, et `-1` sinon. Indiquer la complexité de cette fonction.

❒ 14 – Il s'agit de savoir si un mot est reconnu ou non par un automate.
Caml : Écrire en Caml une fonction nommée `reconnait` telle que si :

- `A`, de type `automate`, code l'automate A,
- `u`, de type `string`, code le mot u sur Σ,

alors `reconnait A u` renvoie le booléen `true` si le mot u est reconnu par A et `false` sinon. Indiquer la complexité de cette fonction.
Pascal : Écrire en Pascal une fonction nommée `reconnait` telle que si :

- `A`, de type `automate`, code l'automate A,
- `u`, de type `tab_char`, contient le mot u sur Σ,
- `lu`, de type `integer`, contient la longueur de u,

alors `reconnait(A, u, lu)` renvoie le booléen `true` si le mot u est reconnu par A et `false` sinon. Indiquer la complexité de cette fonction.

Téléchargé gratuitement sur www.Doc-Solus.fr .

❒ 15 – On considère un automate A **déterministe complet** sur un alphabet Σ, un état p de A et une lettre x de Σ. L'objectif de la question est de concevoir une fonction permettant de modifier l'extrémité de la transition d'origine p et d'étiquette x.

Caml : On considère une liste, nommée `trans`, de type `(char * int) list`, codant l'ensemble des transitions d'origine p. Écrire en Caml une fonction `remplace` telle que, si :

- `x` code la lettre x,
- `q` code un état de A,

alors `remplace x trans q` renvoie une liste identique à `trans` sauf que le couple contenant l'étiquette `x` est remplacé par le couple `(x, q)`. Indiquer la complexité de cette fonction.

Pascal : On considère un tableau, nommé `trans`, de type `tab_transition`, codant l'ensemble des transitions d'origine p. Écrire en Pascal une fonction `remplace` telle que, si :

- `x`, de type `char`, contient la lettre x,
- `q`, de type `integer`, contient un état de A,

alors `remplace(x, trans, q)` renvoie un tableau de type `tab_transition` identique à `trans` sauf que le couple contenant l'étiquette `x` est remplacé par le couple `(x, q)`. Indiquer la complexité de cette fonction.

Quatrième partie : utilisation d'automates.

On considère un mot m sur Σ : $m = m_0 m_1 ... m_{l(m)-1}$.

❒ 16 – Dessiner un automate déterministe comportant $l(m) + 1$ états reconnaissant le langage réduit au seul mot m. On note A_m cet automate.

❒ 17 – Il s'agit de construire l'automate A_m en langage de programmation.

Caml : Écrire en Caml une fonction nommée `automate_de_mot` telle que, si `m`, de type `string`, code le mot m, alors `automate_de_mot m` renvoie un résultat de type `automate` codant l'automate A_m.

Pascal : Écrire en Pascal une fonction nommée `automate_de_mot` telle que, si :

- `m`, de type `tab_char`, contient le mot m,
- `lm`, de type `integer`, contient la longueur de m,

alors `automate_de_mot(m, lm)` renvoie un résultat de type `automate` codant l'automate A_m.

❒ 18 – Il s'agit d'utiliser l'automate A_m pour déterminer si un mot u sur Σ est préfixe du mot m.

Caml : Écrire en Caml une fonction nommée `est_prefixe` telle que, si `m` et `u`, de type `string`, codent les mots m et u, alors `est_prefixe m u` renvoie le booléen `true` quand u est préfixe de m et `false` dans le cas contraire.

On utilisera les fonctions `automate_de_mot` et `execution`.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Pascal : Écrire en Pascal une fonction nommée `est_prefixe` telle que, si :

- `m` et `u`, de type `tab_char`, contiennent les mots m et u,
- `lm` et `lu`, de type `integer`, contiennent les longueurs de m et u,

alors `est_prefixe(m, lm, u, lu)` renvoie le booléen `true` quand u est préfixe de m et `false` dans le cas contraire.
On utilisera les fonctions `automate_de_mot` et `execution`.

❒ 19 – On s'intéresse à présent au langage composé des mots dont m est suffixe. On le note LS_m. Montrer que ce langage est rationnel.

❒ 20 – Expliquer comment on peut modifier l'ensemble des transitions de A_m pour obtenir un automate non déterministe reconnaissant LS_m. On note AS_m cet automate.
On considère le cas où $\Sigma = \{a, b\}$ et $m = ab$. Dessiner l'automate AS_{ab}.

❒ 21 – Déterminiser l'automate AS_{ab} dessiné à la question précédente. On utilisera pour cela un algorithme de déterminisation d'automate.

Soit t un texte et m un motif. On note DS_m un automate **déterministe complet** reconnaissant le langage LS_m ; on suppose que DS_m a un seul état final.

❒ 22 – Décrire les principes d'un algorithme utilisant l'automate DS_m pour résoudre le problème (P). L'algorithme devra avoir une complexité de l'ordre de la longueur de t. Justifier la validité de cet algorithme avec soin.

❒ 23 – Il s'agit de programmer l'algorithme précédent.
Caml : On suppose que l'on dispose d'une fonction `DS` telle que, si `m` est de type `string`, alors `DS m` renvoie un résultat de type `automate` codant l'automate DS_m.
En utilisant cette fonction, écrire en Caml une fonction nommée `positions3` telle que, si `m` et `t`, de type `string`, codent le motif m et le texte t, alors `positions3 m t` résout le problème (P), c'est-à-dire renvoie une liste contenant les positions du motif m dans le texte t.
Pascal : On suppose que l'on dispose d'une fonction `DS` telle que, si :

- `m`, de type `tab_char`, contient le mot m,
- `lm`, de type `integer`, contient la longueur de m,

alors `DS(m, lm)` renvoie un résultat de type `automate` codant l'automate DS_m.
En utilisant cette fonction, écrire en Pascal une fonction nommée `positions3` telle que, si :

- `m` et `t`, de type `tab_char`, contiennent m et t,
- `lm` et `lt`, de type `integer`, contiennent les longueurs de m et t,

alors `positions3(m, lm, t, lt)` résout le problème (P), c'est-à-dire renvoie un résultat de type `pile` contenant, dans le champ `nb`, le nombre de positions de m dans t et, dans le champ `table`, la liste des positions du motif m dans le texte t.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Cinquième partie : automate des suffixes

Le but de cette la partie est d'écrire un automate déterministe complet, avec un seul état final, reconnaissant le langage LS_m.

On considère un alphabet Σ et un mot $m = m_0m_1...m_{l(m)-1}$ sur Σ.

On note $Pref(m)$ l'ensemble des préfixes de m.
Soit h_m l'application de Σ^* dans $Pref(m)$ définie par : pour tout mot u sur Σ, $h_m(u)$ est le plus long mot à la fois suffixe de u et préfixe de m.

❒ 24 – Préciser $h_m(u)$ pour tout mot u quand m est réduit à une seule lettre $x \in \Sigma$.

On revient au cas général : m est un mot quelconque sur Σ.

❒ 25 – Soit $u \in \Sigma^*$. Montrer que $u \in LS_m$ si et seulement si $h_m(u) = m$.

❒ 26 – Préciser $h_m(\varepsilon)$. Préciser $h_m(u)$ quand u est un préfixe de m.

❒ 27 – Montrer que $h_m(ux) = h_m(h_m(u)x)$ pour tout mot u et toute lettre x.

On définit un automate **déterministe et complet** sur l'alphabet Σ, noté S_m, de la façon suivante :

- les états de S_m sont les préfixes de m,
- l'état initial de S_m est ε,
- l'état final de S_m est m,
- pour tout préfixe u de m et toute lettre x, $(u, x, h_m(ux))$ est une transition de S_m et S_m n'admet pas d'autres transitions que les transitions de cette forme.

Convention : on numérote les états de S_m ; l'état initial ε est noté 0 et pour i compris entre 1 et $l(m)$, le préfixe $m_0m_1...m_{i-1}$ est noté i.

❒ 28 – On suppose que l'on a $\Sigma = \{a, b\}$. Dessiner l'automate S_ε.
Indication : cet automate n'a qu'un seul état, noté 0.

❒ 29 – On suppose que l'on a $\Sigma = \{a, b\}$ et on considère le mot ab. Dessiner l'automate S_{ab} en représentant les états par leurs numéros.
Indication : cet automate a trois états, les états ε, a, ab, numérotés 0, 1 et 2.

❒ 30 – On considère l'automate S_m et un mot u sur Σ ; on note δ la fonction de transition de S_m. Montrer l'égalité $\delta^*(\varepsilon, u) = h_m(u)$.

❒ 31 – Montrer que S_m reconnaît le même langage que DS_m, c'est-à-dire LS_m.

Téléchargé gratuitement sur www.Doc-Solus.fr .

Soit m un mot sur Σ et x une lettre de Σ. On pose $m' = mx$. On note $h = h_m$ et $h' = h_{m'}$. On admet les propriétés suivantes de la fonction h'.
Soit u' un préfixe de m' et y une lettre.

(i) Si u' est préfixe de m et $u' \neq m$: $h'(u'y) = h(u'y)$.
(ii) Si $u' = m$: $h'(my) = h(my)$ quand $y \neq x$ et $h'(mx) = mx$.
(iii) Si $u' = m'$: $h'(m'y) = h'(h(mx)y)$.

❒ 32 **–** Donner des règles simples de construction pour passer de S_m à $S_{m'}$.

❒ 33 **–** On considère le cas $\Sigma = \{a, b\}$ et le mot *aba*. Illustrer les règles énoncées à la question précédente en traçant l'automate S_{aba} à partir de l'automate S_{ab} obtenu à la question ❒ 29.

❒ 34 **–** Il s'agit de programmer la fonction `DS` pour l'alphabet $\Sigma = \{a, b\}$.
Caml : Écrire en Caml une fonction nommée `DS` telle que, si `m`, de type `string`, code un mot m, alors `DS m` renvoie un résultat, de type `automate`, codant l'automate S_m. On supposera si nécessaire qu'on a défini une constante entière `MAX_LONGUEUR` et que les mots considérés ont au plus `MAX_LONGUEUR` lettres.
Pascal : Écrire en Pascal une fonction nommée `DS` telle que, si :

- `m`, de type `tab_char`, contient le mot m sur Σ,
- `lm`, de type `integer`, contient la longueur de m,

alors `DS(m, lm)` renvoie un résultat, de type `automate`, codant l'automate S_m.

❒ 35 **–** On revient à un alphabet Σ quelconque. Donner la complexité de la fonction `positions3`.

Téléchargé gratuitement sur www.Doc-Solus.fr .