

## FEUILLE D'EXERCICES N°16 DE L'OPTION D'INFORMATIQUE.

**RECHERCHE DE MOTIFS DANS UN TEXTE.**

L'objet de cette partie est l'étude de différentes méthodes de programmation qui répondent à la question : «Étant données deux chaînes de caractères `source` et `motif`, est-ce que `motif` est une sous-chaîne de `source`?»

Dans toute cette partie, les tableaux sont indexés à partir de 0.

**I.A - Présentation**

Les définitions et notations introduites ici sont valables dans toute la suite de la partie I.

Soit un alphabet  $A$ , ensemble fini de symboles aussi appelés lettres.

- $A^*$  désigne l'ensemble des séquences finies, éventuellement vides, d'éléments de  $A$ , c'est-à-dire l'ensemble de tous les mots, éventuellement vides, sur  $A$ ;
- pour  $w \in A^*$ ,  $|w|$  désigne la longueur de  $w$ , c'est-à-dire le nombre de symboles composant  $w$ ;
- pour  $w \in A^*$  et  $i$  entier ( $0 \leq i < |w|$ ),  $w(i)$  désigne la  $i + 1^{\text{ème}}$  lettre de  $w$ ;
- pour  $x, y \in A^*$ ,  $xy$  désigne le mot issu de la concaténation de  $x$  avec  $y$ ;
- pour un mot  $z \in A^*$ , l'ensemble `sousmot(z)` est défini par :
 
$$\text{sousmot}(z) = \{w \in A^* \mid (\exists z_1, z_2 \in A^*), z = z_1 w z_2\};$$
 si  $w \in \text{sousmot}(z)$ , on dira que  $w$  est un sous-mot de  $z$ ;
- pour deux mots  $w, z \in A^*$ ,  $w$  est un préfixe de  $z$  si et seulement s'il existe  $y \in A^*$  tel que  $z = wy$ ; on dit alors que  $y$  est un suffixe de  $z$ .

Avec ce formalisme, la question étudiée s'écrit :

Soient deux mots `source` et `motif` dans  $A^*$  avec  $|\text{motif}| \leq |\text{source}|$ , est-ce que `motif`  $\in$  `sousmot(source)`?

**I.B - Méthode itérative**

Dans la partie **I.B**, les mots sont représentés par des tableaux de caractères. L'algorithme itératif suivant permet de répondre à la question.

```

1 RECHERCHE ITÉRATIVE BRUTE(motif, source).
2 i <- 0
3 j <- 0
4 tant que i < |motif| et j < |source|
5 faire si motif(i) = source(j)
6   alors
7     i <- i + 1
8     j <- j + 1
9   sinon
10    j <- j - i + 1
11    i <- 0
12 si i = |motif|
13   alors retourner vrai
14   sinon retourner faux

```

**I.B.1)** Donner un invariant de boucle. Indication : on pourra introduire l'indice  $k$  initialisé à 0 et correspondant à la mise à jour de  $j$  à la ligne 10.

**I.B.2)** Montrer la terminaison de l'algorithme.

**I.B.3)** Traduire l'algorithme en CAML en utilisant les définitions suivantes.

En CAML :

```
Recherche_iter_brute : 'a vect -> 'a vect -> bool = <fun>
```

**I.B.4)** Dans cette question, l'alphabet  $A$  possède uniquement deux lettres :  $A = \{a, b\}$ . Les mots **source** et **motif** ont pour longueurs respectives  $n$  et  $p$ . Déterminer la complexité de l'algorithme en nombre de comparaisons de caractères en fonction de  $n$  et  $p$  dans le pire cas. Donner un exemple où **source** et **motif** sont de longueurs respectives  $n$  et  $p$  et pour lesquels la complexité de l'algorithme est la plus défavorable.

### I.C - Méthode récursive

Dans la partie **I.C**, les mots sont représentés par une structure de liste munie des opérations pré-définies suivantes :

```
Listevide : liste -> booléen
```

```
Construit : lettre
```

```
Tete : liste -> lettre
```

```
Queue : liste -> liste
```

Le raisonnement suivant permet de construire une méthode récursive :

- Le mot **motif** est identique à un sous-mot de **source** quand **motif** est un préfixe de **source** ou quand **motif** est identique à un sous-mot de **source** privé de son premier caractère.
- Le mot **motif** est un préfixe de **source** quand **motif** est vide ou quand il satisfait les deux conditions suivantes :
  - le premier caractère de **motif** est identique au premier caractère de **source** ;
  - **motif** privé de son premier caractère est un préfixe de **source** privé de son premier caractère.

**I.C.1)** Écrire la fonction de recherche récursive en CAML, en utilisant la fonction **Est\_Prefixe** et les prototypes donnés ci-dessous.

Prouver la terminaison de l'algorithme.

En CAML :

```
List_Vide : 'a list -> bool = <fun>
```

```
Tete : 'a list -> 'a = <fun>
```

```
Queue : 'a list -> 'a list = <fun>
```

```
Construit : 'a -> 'a list -> 'a list = <fun>
```

```
Est_Prefixe : 'a list -> 'a list -> bool = <fun>
```

```
Recherche_Recursive : ' a list -> 'a list -> bool = <fun>
```

**I.C.2)** Écrire la fonction **Est\_Prefixe** en CAML en utilisant les prototypes de la question précédente.

**I.C.3)** Déterminer la complexité, en nombre de comparaisons de caractères, de cet algorithme dans le cas le plus défavorable. Comparer avec le résultat de la version itérative.

## I.D - Algorithme simplifié de Knuth-Morris-Pratt

Dans la partie **I.D**, les chaînes de caractères sont à nouveau représentées par des tableaux et les définitions de type de la partie **I.B** sont utilisées.

Cette partie présente une première amélioration, découverte par Knuth, Morris et Pratt, de la méthode itérative. L'idée de base est la suivante. Dans la méthode proposée partie **I.B**, dès que le premier caractère de motif est identique au caractère courant de *source*, la course à la comparaison commence et continue tant que les caractères coïncident, balayant ainsi les préfixes successifs de *motif*. Appelons faux-départ le plus grand de ces préfixes quand une différence est détectée. Par exemple, supposons que *motif* = "abbbbbbc". Si un échec de comparaison est rencontré en *motif*(5), le faux-départ est constitué du sous-mot "abbbb" qui se retrouve également dans *source*. Il est inutile de reprendre la recherche dans *source* avant l'indice courant, puisque *source* ne comprend pas d'autre *a* entre la position de départ et la position courante.

Cette situation est particulière. Dans le cas général, un suffixe de faux-départ peut coïncider avec un préfixe de *motif*. La course à la comparaison peut alors reprendre à partir de l'indice suivant le préfixe dans *motif* et partir de l'indice courant dans *source*.

Par exemple :

```
motif      a b c a a b c b b
source    ... a b c a a b c a ...
                ^
```

La caractéristique  $\wedge$  indique l'indice courant dans *source*. Le faux départ est détecté et vaut "abcaabc". Le suffixe "abc" du faux-départ en est également un préfixe, donc est un préfixe de *motif*. La recherche peut continuer par :

```
motif      a b c a a b c b b
source    ... a b c a a b c a ...
                ^
```

La position de reprise dans *motif* ne dépend que du mot *motif* et de l'indice dans *motif* lors de l'échec. Dans l'exemple précédent, l'échec a eu lieu à l'indice 7 dans *motif*, et l'indice de reprise vaut 3. Il est donc possible de prédéterminer les indices de reprise dans *motif* pour chaque position possible d'échec. Les indices de reprise seront stockés dans un tableau auxiliaire. Ce tableau auxiliaire, *aux*, de même longueur que *motif* est tel que *aux*(*i*) est l'indice de reprise dans *motif* en cas d'échec à l'indice *i*. Plus précisément, en cas d'échec lors de la comparaison de la lettre d'indice *i* dans *motif* et de la lettre d'indice *j* dans *source*, la comparaison reprendra à l'indice *aux*(*i*) dans *motif* et à l'indice *j* dans *source*. Néanmoins, si l'échec a eu lieu sur la première lettre de *motif* (*i* = 0), il convient de reprendre la comparaison à l'indice 0 dans *motif* et à l'indice *j* + 1 dans *source*. Pour faciliter le traitement de ce cas particulier, on prendra *aux*(0) = -1. Le tableau auxiliaire associé à l'exemple précédent vaut donc (-1, 0, 0, 0, 1, 1, 2, 3, 0).

**I.D.1)** Écrire en CAML, l'algorithme simplifié de Knuth, Morris et Pratt en supposant donnée la fonction *calculer\_tab\_aux* qui construit le tableau auxiliaire et en utilisant les prototypes ci-dessous.

En CAML :

```
recherche_KMP = 'a vect -> 'a vect -> int vect -> bool = <fun>
calculer_tab_aux = 'a vect -> int vect = <fun>
```

**I.D.2)** Dans cette question, *A* est un alphabet de trois lettres:  $A = \{a, b, c\}$ . Construire à la main le tableau auxiliaire pour le motif "abaababaabaab".

**I.D.3)** Écrire, en CAML, un algorithme qui construit le tableau auxiliaire en utilisant les prototypes de la question **I.D.1**. (Indication : il s'agit de rechercher à l'intérieur de *motif* les préfixes qui en sont également des sous-mots).