

TP Caml 8: Recherche de motif dans un texte

lionel.rieg@ens-lyon.fr

3 & 10 mars 2009

On s'intéresse ici à la recherche d'un motif (souvent un mot) dans un texte, comme par exemple lors une recherche dans un fichier PDF ou une page web avec Ctrl-F. On prend les conventions et notations suivantes :

- l'alphabet considéré ici est celui du codage ASCII (celui de Caml) contenant 256 caractères ; on peut accéder à l'indice de chaque caractère dans l'énumération par la fonction `int_of_char` ; la réciproque étant `char_of_int`
- le motif et le texte seront donnés comme deux chaînes de caractères
- le texte sera t , de longueur $|t|$; le motif sera p , de longueur $|p|$

1 Recherche sans prétraitement

1.1 Algorithme naïf

- **Question 1** Écrire un algorithme qui cherche si le motif apparaît à une position donnée dans le texte. Il sera de type

```
occur_check : string → string → int → bool
```

- **Question 2** Écrire un algorithme naïf de recherche d'un motif dans un texte, de type

```
naive_search : string → string → bool
```

- **Question 3** Modifier l'algorithme précédent pour qu'il renvoie la liste des positions des occurrences.
► **Question 4** Quelle est la complexité de cet algorithme dans le pire cas ? Un exemple du pire cas ? Et en moyenne, une idée ?

1.2 Algorithmes Shift-And et Shift-Or

On va améliorer l'algorithme précédent en effectuant simultanément lors de la lecture d'une lettre toutes les recherches où cette lettre peut intervenir dans le motif, ceci en simulant un automate non déterministe qui recherche le motif.

- **Question 5** Écrire un automate non déterministe à $|p| + 1$ états sans ε -transitions et à un seul état initial qui recherche le motif p .

Dans cet automate, on remarque d'une part qu'on autorise le motif à commencer après chaque lettre du texte et d'autre part que si la lettre lue ne permet pas d'avancer à l'état suivant dans l'automate, on « perd » l'état courant. Afin d'éviter le non déterminisme, on va plutôt simuler l'automate (déterministe) des parties qui correspond. Pour ce faire, on va représenter par des bits sur un mot machine E les différents états dans lesquels l'automate peut être en un point du texte. Le bit de poids 2^i de E représentera l'état $i + 1$ de l'automate : il vaudra 1 si l'état $i + 1$ est possible, 0 sinon. L'inconvénient de cette technique est qu'elle limite le nombre de caractères du motif à la longueur d'un mot machine. La simulation des transitions se fait alors par décalage puis filtrage des états avec un masque qui dépend des positions dans le motif de la lettre lue : vu comme un entier, E devient $2E + 1$ puis on effectue un « et » logique bit à bit (land). Voir Fig. 1 l'exécution de l'algorithme sur le mot « annale », recherché dans le texte « annale annuelle » :

- **Question 6** Implémenter le calcul des masques des lettres de l'alphabet pour un motif p donné en argument. Il sera de type :

```
sa_table : string → int vect
```

- **Question 7** Implémenter l'algorithme Shift-And. Il renverra la liste des positions où le motif apparaît.

L'algorithme utilise trois opérations lors la lecture d'une lettre : le décalage d'un bit et l'ajout du 1 à droite à l'état de l'automate puis le « et » avec le masque de bit correspondant à la lettre. On peut en économiser une en inversant le rôle des 1 et des 0 : il n'y a plus besoin dans ce cas d'ajouter un 1 à l'état de l'automate après son décalage. Cette variante est appelée algorithme Shift-Or.

- **Question 8** Implémenter l'algorithme Shift-Or. (le « ou » bit à bit se fait avec `lor`)

Table des caractères pour le motif « annale » :

a	n	l	e	*
001001	000110	010000	100000	000000

1	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ 'a' \quad 001001 \\ E' \quad 000001 \end{array} \right.$	4	$\left\{ \begin{array}{l} 2E+1 \quad 001001 \\ 'a' \quad 001001 \\ E' \quad 001001 \end{array} \right.$	7	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ ' ' \quad 000000 \\ E' \quad 000000 \end{array} \right.$	10	$\left\{ \begin{array}{l} 2E+1 \quad 000101 \\ 'n' \quad 000110 \\ E' \quad 000100 \end{array} \right.$	13	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ 'l' \quad 000000 \\ E' \quad 000000 \end{array} \right.$
2	$\left\{ \begin{array}{l} 2E+1 \quad 000011 \\ 'n' \quad 000110 \\ E' \quad 000010 \end{array} \right.$	5	$\left\{ \begin{array}{l} 2E+1 \quad 010011 \\ 'l' \quad 010000 \\ E' \quad 010000 \end{array} \right.$	8	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ 'a' \quad 001001 \\ E' \quad 000001 \end{array} \right.$	11	$\left\{ \begin{array}{l} 2E+1 \quad 001001 \\ 'u' \quad 000000 \\ E' \quad 000000 \end{array} \right.$	14	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ 'l' \quad 000000 \\ E' \quad 000000 \end{array} \right.$
3	$\left\{ \begin{array}{l} 2E+1 \quad 000101 \\ 'n' \quad 000110 \\ E' \quad 000100 \end{array} \right.$	6	$\left\{ \begin{array}{l} 2E+1 \quad 100001 \\ 'e' \quad 100000 \\ E' \quad 100000 \end{array} \right.$	9	$\left\{ \begin{array}{l} 2E+1 \quad 000011 \\ 'n' \quad 000110 \\ E' \quad 000010 \end{array} \right.$	12	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ 'e' \quad 100000 \\ E' \quad 000000 \end{array} \right.$	15	$\left\{ \begin{array}{l} 2E+1 \quad 000001 \\ 'e' \quad 100000 \\ E' \quad 000000 \end{array} \right.$

FIG. 1 – Recherche du motif « annale » dans le texte « annale annuelle »

2 Recherche avec prétraitement du motif

Si on autorise un traitement préalable du texte ou du motif, on peut obtenir des complexités meilleures, en pire cas et même en moyenne ! Sans aller jusqu'aux meilleurs algorithmes théoriques (on peut chercher un motif quelconque en temps linéaire et en espace constant !) assez compliqués, on va voir ici deux des algorithmes utilisés en pratique.

2.1 Algorithme de Boyer-Moore-Horspool

La complexité en $O(|t| \cdot |p|)$ de l'algorithme naïf vient du fait que le décalage dans le texte ne se fait que d'une seule lettre à cause des possibilités de chevauchement. On veut améliorer cela, au moins en moyenne, pour obtenir une complexité sous-linéaire. Pour cela, on va lire le mot depuis la fin, de façon à pouvoir sauter une partie du motif non encore lue en cas d'erreur.

En détail, on utilise une table de saut dite *table du mauvais caractère* qui va donner une heuristique de la longueur du saut que l'on peut effectuer pour placer la nouvelle fin de mot. Pour une lettre α , la table contient la longueur du saut qui permet d'aligner α avec sa dernière occurrence dans le motif : c'est la distance entre la dernière lettre du motif et la position de la dernière occurrence de α dans le motif, sauf pour la dernière lettre du motif pour laquelle c'est l'avant dernière occurrence qui est prise en compte (exercice : pourquoi ?). Lorsqu'une lettre n'apparaît pas dans le motif, on considère qu'elle apparaît avant la première lettre du motif.

Par exemple, pour le motif « annale », la table sera (« * » représente toute autre lettre) :

a	n	l	e	*
2	3	1	6	6

► **Question 9** Écrire une fonction qui prend un motif et renvoie la table du mauvais caractère associée.

► **Question 10** Écrire l'algorithme de Boyer-Moore-Horspool. Quelle est sa complexité ?

Une version plus compliquée de cet algorithme permet d'assurer une complexité linéaire dans le pire cas.

2.2 Algorithme de Knuth-Morris-Pratt

Cet algorithme construit implicitement un automate non déterministe avec ε -transitions qui reconnaît le motif. Malgré le non déterminisme, on obtient quand même au final une complexité en $O(|t| + |p|)$. L'idée est de connaître pour chaque préfixe du motif qui correspond au texte, le plus long décalage que l'on peut effectuer dans le motif sans rater d'occurrences, c'est à dire qu'on cherche le *plus long préfixe (strict) d'un préfixe w du motif qui en soit aussi un suffixe*, qu'on appelle le *bord* de w . On note φ la fonction qui à un mot associe son bord.

► **Question 11** Si α est une lettre et w un mot, quel est le lien entre $\varphi(w\alpha)$ et $\varphi(w)\alpha$, $\varphi(\varphi(w))\alpha$, ... ?

► **Question 12** Écrire un algorithme qui, pour chaque position $i \in \llbracket 1, |p| \rrbracket$ dans le motif p , calcule la longueur de son bord, i.e. $|\varphi(p_0p_1 \dots p_i)|$, taille du plus long préfixe strict de $p_0p_1 \dots p_i$ qui en soit aussi un suffixe. Il sera de type :

failure_function : string \rightarrow int vect
--

Cette fonction d'échec donne la longueur du saut que l'on effectue lorsqu'on rencontre une lettre qui ne correspond pas : on aligne la partie déjà lue du motif avec son bord et on reprend la lecture à la lettre qui fait échec.

► **Question 13** Écrire l'algorithme de Knuth-Morris-Pratt.