Reconnaissance de motifs avec erreurs

Luc Maranget [*]

Avril 2002

1 L'essentiel

Il s'agit d'écrire un programme `ajgrep` qui prend en arguments un entier k, un motif p et un nom de fichier, et qui affiche les lignes du fichier dont un sous-mot est filtré par le motif à k erreur près au plus.

2 Exposé du sujet

Les motifs considérés peuvent être de trois classes de difficulté croissante. Je présente ici le cas le plus simple où le motif se réduit à un mot m. Les principes de l'outil `ajgrep` sonts décrit par deux articles, disponibles sur la version web de ce sujet[1], qui sera accompagée d'une page de suivi. *Il est très important de consulter cette page de suivi régulièrement.*

2.1 Erreurs

Le mot m' diffère de au plus une erreur d'un autre mot m, dans les cas suivants :

Oubli On obtient m' en enlevant un caractère à m.

Insertion On obtient m' en insérant un caractère à m.

Substitution On obtient m' en changeant un caractère de m.

Notez que cette relation entre les mots est symétrique et reflexive (en substituant un caractère à l'identique).

On dira que les mots m et m' diffèrent de au plus k erreurs, lorsqu'il existe $k+1$ mots $e_0, \ldots, e_k$, avec $e_0 = m$, $e_k = m'$, et e_{i+1} diffère de au plus une erreur de e_i.

[*] `Luc.Maranget@inria.fr`

[1] `http://www.enseignement.polytechnique.fr/profs/informatique/Luc.Maranget/ajgrep/sujet.html`

2.2 Reconnaissance des sous-mots

Selon le cours, étant donné un motif p, on peut construire un automate fini qui reconnaît l'ensemble des mots filtrés par le motif et il n'est pas très difficile d'utiliser cet automate pour reconnaître les mots dont un sous-mot est filtré par p.

Dans le cas simple ou le motif se réduit à un mot $m = \alpha_1\alpha_2 \ldots \alpha_n$, de longueur n. L'ensemble des mots filtrés par m se réduit à m et l'automate est déterministe et facile à construire. L'automate possède $n+1$ états notés e_0, ... , e_n, son état initial est e_0, son état final est e_n et il y a une transition étiquetée par le caractère α_i entre les états e_{i-1} et e_i. Voici par exemple l'automate associé au mot `coucou`.

$$(0) \xrightarrow{\texttt{c}} (1) \xrightarrow{\texttt{o}} (2) \xrightarrow{\texttt{u}} (3) \xrightarrow{\texttt{c}} (4) \xrightarrow{\texttt{o}} (5) \xrightarrow{\texttt{u}} (6)$$

Utiliser cet automate pour reconnaître si un mot quelconque est égal à `coucou` n'a pas d'intérêt, mais cet automate se revèle pratique pour trouver si un texte quelconque contient `coucou`.

On procède à l'aide d'une suite d'*ensembles* d'états ainsi définie :

- E_0 est $\{e_0\}$.
- On passe de E_{i-1} à E_i en consommant un caractère β du texte : E_i est E_0 union l'ensemble des e_{j+1} tels que e_j appartient à E_{i-1} et qu'il existe une transition étiquetée par β entre e_j et e_{j+1}. On note $E_i = E_0 \cup \beta(E_{i-1})$.

Notons que, si à un instant quelconque l'état final e_n appartient à E_i, alors une occurence du mot m est identifiée. En effet, sans faire de preuve formelle, $e_j \in A_i$ signifie que les j derniers caractères vus forment un préfixe de m.

L'efficacité repose sur l'encodage des ensembles par des champs de bits, voire par des entiers quand la taille du mot m est inférieure à disons 31 ou 63. Dans ce dernier cadre, l'opération $\beta(E)$ se réalise par deux opérations élémentaires : un masquage et un décalage à gauche, tandis que les unions sont réalisées par des ou logiques. C'est pourquoi ces automates sont parfois dénommés *shift-or*.

Les automates se révèlent encore plus intéressants lorsqu'il s'agit de reconnaître les sous-mots du texte qui diffèrent de m de au plus k erreurs. On commence par definir une nouvelle fonction **Succ** sur les ensembles d'états, $\mathbf{Succ}(E)$ est l'ensembles des e_{j+1} où e_j décrit E.

En se limitant pour simplifier à une seule erreur, on se donne alors une deuxième suite d'ensembles d'états :

- $E_0^1 = E_0 \cup \mathbf{Succ}(E_0)$.
- Si β est le i-ème caractère du texte,

$$E_i^1 = E_0^1 \cup \beta(E_{i-1}^1) \cup \mathbf{Succ}(E_{i-1}) \cup \mathbf{Succ}(E_i) \cup E_{i-1}$$

Toujours sans faire de preuve formelle, on notera que $e_j \in E_i^1$ signifie qu'un suffixe du texte lu diffère du préfixe de taille j de m de au plus une erreur.

Notons μ_j le préfixe de taille j de m. Puisque, par exemple, $e_j \in E_{i-1}$ signifie qu'un suffixe μ du texte lu est égal à μ_j, alors $\mu\beta$ diffère de μ_j par une insertion et de μ_{j+1} par une substitution, ce qui explique les termes E_{i-1} et $\mathbf{Succ}(E_{i-1})$ dans la définition de E_i^1.

Il n'est pas très compliqué d'étendre cette idée à au plus k erreurs, en considérant $k+1$ suites d'ensembles d'états.

3 Ce qui est demandé

Votre programme doit donc détecter les lignes d'un texte dont un sous-mot est filtré par un un motif p à k erreurs près. On se simplifiera la vie en ne considérant que le jeu de caractère ASCII (128 caractères possibles, sans les accents), ou ISO-LATIN1 (256 caractères possibles). Le programme sera jugé sur sa correction et sur sa vitesse. C'est à dire que vous devez réaliser efficacement les opérations sur les ensembles d'états.

Dans un premier temps on pourra traiter le cas des motifs restreints à un mot. Puis on considérera des motifs plus généraux, comprenant les constructions suivantes :

Des attrape-tout Le motif << `.` >> filtre n'importe quel caractère, tandis que le motif << `#` >> filtre n'importe quel mot.

Les motifs optionnels Noté c`?`, ce motif filtre le mot vide et le mot formé de la lettre c. De même, le motif << `.?` >> filtre le mot vide et tous les mots réduits à un caractère.

Les articles décrivent le principe de ces extensions. Il s'agit de compiler un motif en un automate non-déterministe, puis d'employer cet automate pour détecter le filtrage à k erreurs près.

On pourra ensuite, de façon optionnelle, considérer d'autres extensions suggérées dans les articles. En particulier, un programme opérant sur la la totalité des expressions rationnelles sera apprécié.