TP `egrep` — Corrigé

David A. Madore

13 décembre 2017

INF105

Git : `2b6b885 Wed Dec 13 13:27:28 2017 +0100`

Le programme Unix `egrep` a pour fonction de rechercher (*filtrer*) les lignes d'un fichier dont une partie (au sens : facteur d'un mot) vérifie une certaine expression rationnelle. La syntaxe est : « `egrep` *regexp fichiers* » où *regexp* désigne l'expression à chercher et *fichiers* la liste des fichiers (un par argument) dans lesquels la recherche doit être menée : par exemple, « `egrep toto monfichier` » cherche dans `monfichier` toutes les lignes contenant `toto` ; en retour, `egrep` produit les lignes recherchées, éventuellement précédées du nom du fichier où elles ont été trouvées, et en colorisant éventuellement la partie qui vérifie l'expression (selon les options passées et/ou la configuration).

`egrep` comporte de nombreuses options et particularités d'expressions rationnelles, dont la liste peut être connue en consultant le manuel (« `man egrep` »). Par exemple, l'option `-c` sert à compter le nombre de lignes trouvées au lieu de les afficher.

Il faut prendre garde au fait que le shell Unix va interpréter certains caractères spéciaux : le moyen le plus simple de l'éviter est d'entourer le paramètre à protéger de caractères `'` (apostrophe droite, ou guillemet simple), qui protège tous les caractères jusqu'au prochain caractère `'` rencontré. Par exemple, « `egrep 'xy*z' monfichier` » cherche dans `monfichier` toutes les lignes dont une partie vérifie l'expression rationnelle `xy*z`, c'est-à-dire, les lignes contenant un `x` suivi d'un nombre quelconque (éventuellement nul) de `y` et d'un `z`.

Pour rechercher les lignes vérifiant une certaine expression rationnelle du début à la fin, on peut utiliser les caractères spéciaux `^` et `$` qui servent à ancrer l'expression au début et à la fin de la chaîne respectivement : par exemple, « `egrep '^x' monfichier` » renvoie les lignes de `monfichier` qui commencent par `x`, et « `egrep '^xy*z$' monfichier` » renvoie celles qui sont exactement formées d'un `x` (en début de ligne) suivi d'un nombre quelconque de `y` et d'un `z` (en fin de ligne).

Exercice 1.

(Le but de cet exercice est d'utiliser `egrep` dans un cadre proche du cadre théorique vu en cours.)

Le fichier `~madore/inf105/liste1` contient 24 mots sur l'alphabet $\{\mathtt{a},\mathtt{b}\}$, tous distincts, à raison de un par ligne. (On pourra commencer par jeter un coup d'œil au fichier, par exemple en tapant `cat ~madore/inf105/liste1` pour l'afficher dans le terminal.) Utiliser `egrep` pour répondre aux questions suivantes :

(a) Combien de mots de cette liste ne contiennent aucun `a` ?

(b) Combien de mots commencent par `b` ?

(c) Combien de mots sont formés d'un nombre quelconque de `a` suivi d'un nombre quelconque de `b` ?

(d) Combien de mots sont formés d'un nombre non nul de `a` suivi d'un nombre non nul de `b` ?

(e) Dans combien de mots chaque `a` est-il suivi d'(au moins) un `b` ?

(f) Dans combien de mots le nombre de `a` est-il congru à 1 modulo 3 ?

Corrigé. (a) `egrep -c '^b*$'` (ou bien `'^[^a]*$'`) renvoie 5.

(b) `egrep -c '^b'` (ou bien `'^b(a|b)*$'` ou encore `'^b[ab]*$'`) renvoie 9.

(c) `egrep -c '^a*b*$'` renvoie 14.

(d) `egrep -c '^aa*bb*$'` (ou bien `'^a+b+$'`) renvoie 5.

(e) `egrep -c '^b*(abb*)*$'` (ou bien `'^b*(ab+)*$'`) ou `'^(b|ab)*$'` renvoie 11.

(f) Si l'alphabet ne contenait que la lettre `a`, l'expression rationnelle désignant le langage constitué des mots formé d'un nombre de a congru à 1 modulo 3 s'écrirait `a(aaa)*`. On intercale des `b*` dans cette expression de manière à les ignorer : `egrep -c '^b*a(b*ab*ab*a)*b*$'` (ou bien `'^b*ab*(ab*ab*ab*)*$'` ou toutes sortes d'autres variantes) renvoie 11. ✓

Exercice 2.

(Le but de cet exercice est d'introduire quelques fonctionalités de `egrep` qui ne dépassent pas le cadre théorique des expressions rationnelles mais qui sont néanmoins utiles en pratique.)

Le fichier `~madore/inf105/liste2` contient des mots (tous distincts) sur l'alphabet ASCII ; ces lignes peuvent contenir, notamment, des espaces et des caractères spéciaux pour `egrep`.

(a) Combien de lignes ne contiennent que des lettres de l'alphabet latin (de A à Z sans diacritiques, majuscules comme minuscules) ? (On rappelle pour cela la syntaxe « `[abc]` » d'`egrep` qui permet de désigner un caractère parmi `a`, `b` et `c` (c'est donc synonyme de `(a|b|c)` mais uniquement pour des caractères individuels) et « `[a-f]` » qui désigne un caractère entre `a` et `f`.) Combien de lignes contiennent un caractère autre que les lettres de l'alphabet latin ? (On rappelle que « `[^a-f]` » permet de désigner un caractère n'appartenant pas à l'intervalle entre `a` et `f`.)

(b) Combien de lignes commencent par un `a` et finissent par un `z` ? (On rappelle pour cela la syntaxe « `.` » d'`egrep` qui permet de désigner un caractère quelconque.)

(c) Combien de lignes contiennent un astérisque ? (On rappelle qu'on peut « échapper » un caractère spécial pour `egrep` en le faisant précéder d'un backslash (`\`).) Au moins deux astérisques ? Exactement deux astérisques ?

(d) Combien de lignes contiennent exactement six caractères ? (On rappelle les syntaxes « r{`{`}n{`}`} » et « r{`{`}m`,`n{`}`} » et « r{`{`}m`,`${`}`} » et « r{`{`}`,`n{`}`} » pour chercher respectivement exactement n, entre m et n, au moins m, et au plus n occurrences consécutives [1] de l'expression régulière r.) Entre quatre et huit caractères ? Au moins trois fois le caractère `u` ? Exactement six fois le caractère `u` ?

Corrigé. (a) `egrep -c '^[A-Za-z]*$'` pour uniquement des lettres (renvoie 5), et `'[^A-Za-z]'` pour un caractère qui n'est pas une lettre (renvoie 7).

(b) `egrep -c '^a.*z$'` (renvoie 3)

(c) `egrep -c '\*'` pour un astérisque (renvoie 4), `'\*.*\*'` pour au moins deux (renvoie 3), et enfin `'^[^*]*\*[^*]*\*[^*]*$'` pour exactement deux astérisques (renvoie 2).

(d) `egrep -c '^.{6}$'` pour exactement six caractères (renvoie 2), `'^.{4,8}$'` pour entre quatre et huit caractères (renvoie 7), `'(u.*){3,}'` pour au moins trois fois `u` (renvoie 2), et `'^[^u]*(u[^u]*){6}$'` pour exactement six `u` (renvoie 1). ✓

Exercice 3.

(Le but de cet exercice est d'évoquer une fonctionalité de `egrep` souvent importante dans la pratique et qui dépasse le cadre des expressions rationnelles au sens mathématique : les *références arrière*.)

La syntaxe `\`n d'`egrep`, où n est un chiffre entre 1 et 9, constitue une *référence arrière* (ou *backreference* en anglais) : cette extension au mécanisme général des expressions rationnelles permet de demander une répétition du même facteur qui a été « capturé » par un bloc de parenthèses antérieur (les blocs de parenthèses étant numérotés dans l'ordre de leurs parenthèses ouvrantes : ainsi `\1` désigne la chaîne de caractères qui a été vérifié par le bloc de parenthèses s'ouvrant à la première parenthèse ouvrante). Par exemple, `(foo|bar)x*\1` recherche une occurrence de `foo` ou bien `bar`, suivi d'un nombre quelconque de `x`, suivi de la *même* chaîne `foo` ou `bar` que précédemment (c'est donc équivalent à `(foox*foo|barx*bar)` mais pas à `(foo|bar)x*(foo|bar)`).

1. I.e., r{`{4}`} équivaut à $rrrr$ et r{`{1,3}`} équivaut à $(r|rr|rrr)$ par exemple.

(a) Comment utiliser `egrep` pour rechercher les lignes de `liste1` qui sont formées d'un certain nombre de `a`, puis d'un `b`, puis du *même* nombre de `a` que précédemment ?

(b) Montrer qu'on ne pourrait pas faire cette recherche sans la fonctionalité additionnelle des références arrière, au sens où la recherche faite dans la question précédente ne définit pas un langage rationnel au sens mathématique.

Corrigé. (a) On utilise `egrep '^(a*)b\1$'` (qui renvoie trois lignes).

(b) Il s'agit de montrer que le langage $L := \{a^n b a^n : n \in \mathbb{N}\}$ constitué des mots formées d'un certain nombre de a, suivis de b puis du même nombre de a, n'est pas rationnel. On utilise pour cela le lemme de pompage : supposons par l'absurde que L soit rationnel, et soit k tel que donné par le lemme de pompage ; considérons alors le mot $t := a^k b a^k \in L$: il devrait admettre une factorisation $t = uvw$ vérifiant (i) $|v| \geq 1$, (ii) $|uv| \leq k$ et (iii) $uv^i w \in L$ pour tout $i \in \mathbb{N}$. La propriété (ii), vu que t commence par a^k, garantit que u et v sont formés entièrement de la lettre a, disons $u = a^m$ et $v = a^n$ (où $m = |u|$ et $n = |v|$), en notant que $n \geq 1$ d'après (i) ; on a alors $w = a^{k-m-n} b a^k$, et $uv^i w = a^{m+in+(k-m-n)} v a^k = a^{k+(i-1)n} b a^k$ est censé appartenir à L pour tout i ; en prenant $i \neq 1$, on a une contradiction (puisque $k + (i-1)n \neq k$). ✓

Exercice 4.

(a) Expliquer pourquoi il existe un automate déterministe fini sur le langage $\{0, 1, 2, \ldots, 9\}$, ayant exactement 7 états, qui accepte le langage formé des représentations décimales des entiers naturels multiples de 7 (on ignorera les 0 initiaux, i.e., on n'imposera pas que l'écriture décimale soit normalisée).

(b) Utiliser un langage de programmation quelconque pour construire une expression rationnelle qui dénote le langage en question. Tester son fonctionnement.

Corrigé. (a) On construit l'automate dont l'ensemble des états est $\{0, 1, 2, \ldots, 6\}$ représentant les sept classes de congruence possible d'un entier modulo 7, la transition partant de $q \in \{0, 1, 2, \ldots, 6\}$ et étiquetée par $i \in \{0, 1, 2, \ldots, 9\}$ aboutissant à $10q + i$ modulo 7 (c'est-à-dire à l'état étiqueté par l'unique élément de $\{0, 1, 2, \ldots, 6\}$ qui est congru à $10q + i$ modulo 7), ou, si on préfère, $3q + i$.

(b) L'algorithme sera essentiellement le suivant : on initialise un tableau T à double entrées q_1 et q_2 (chacun variant de 0 à 6 inclus) de chaînes de caractères en plaçant dans la case $[q_1][q_2]$ soit le seul $i \in \{0, \ldots, 9\}$ tel que $q_2 \equiv 3q_1 + 1$ (vu comme une chaîne de caractères de longueur 1) soit la concaténation des tels i entourés par des crochets (pour utiliser la syntaxe d'`egrep` selon laquelle `[07]` désigne l'un des deux caractères `0` ou `7`). Ensuite, on effectue une boucle triple : pour q_e (l'état à éliminer) parcourant tous les états sauf un dans un certain ordre, disons en décroissant de 6 à 1, et q_1 et q_2 parcourant chacun indépendamment tous les états non encore éliminés (donc tous les états de 0 à $q_e - 1$ si on élimine en décroissant de 6 à 1), on remplace $T[q_1][q_2]$ par la chaîne

$$\text{“(”} + T[q_1][q_2] + \text{“|”} + T[q_1][q_e] + T[q_e][q_e] + \text{“}\star\text{”} + T[q_e][q_2] + \text{“)”}$$

où $+$ désigne la concaténation des chaînes de caractères et “x” désigne la chaîne contenant le seul caractère x. Au final, on retourne “^” $+ T[q_0][q_0] +$ “`*$`” (où q_0 est le seul état non éliminé).

Si on élimine les états en décroissant de 6 à 1, on obtient une chaîne de longueur 16 915 qui commence par

```
^((((((([07]|6[29]*3)|(5|6[29]*[18])(4|5[29]*[18])*(6|5[29]*3))
```

et qui finit par

```
][29]*3)|([07]|[18][29]*[18])(4|5[29]*[18])*(6|5[29]*3))))))*$
```

On peut par exemple tester son fonctionnement en vérifiant que `seq 0 10000 | egrep "$(calcule_regexp)"` (où `calcule_regexp` est le programme qui la calcule) et `seq 0 7 10000` produisent des sorties identiques (le programme Unix `seq` sert à produire une liste d'entiers). ✓

Exercice 5.

Sachant que `/usr/share/dict/american-english` contient un dictionnaire de mots anglais, trouver trois mots anglais qui commencent par les lettres « he » et finissent par les lettres « he » (i.e., qui ont « he » comme préfixe et « he » comme suffixe).

Corrigé. On peut utiliser la commande

```
egrep -i '^he.*he$' /usr/share/dict/american-english
```

qui renvoie les deux mots « headache » et « heartache », Mais il y a un piège, c'est que cette expression rationnelle omet une solution, à savoir le mot « he » lui-même. ✓