Simulation de la commande grep

Sujet proposé par Daniel Krob

dk@lix.polytechnique.fr

1 La recherche de motif avec grep

L'utilitaire grep[1] est une commande Unix qui permet de rechercher dans un ensemble de fichiers toutes les lignes qui contiennent un motif spécifié à l'aide d'une expression régulière.

Exemple : L'exécution de la commande

```
egrep 'chercher|galere' Fourberies-de-Scapin
```

permet par exemple de rechercher dans le fichier "Fourberies-de-Scapin" toutes les lignes qui contiennent soit la chaîne de caractère "chercher", soit la chaîne de caractère "galere".

1.1 Représentation des expressions régulières

La syntaxe des expressions régulières étendues [2] manipulées par grep est décrite ci-dessous :

```
A regular expression is a pattern that describes a set of strings.
Regular expressions are constructed analogously to arithmetic
expressions, by using various operators to combine smaller
expressions.

Grep understands  two different versions of regular expression
syntax: "basic" and "extended." In GNU grep, there is no
difference in available functionality using either syntax. In
other implementations, basic regular expressions are less
powerful. The following description applies to extended regular
expressions; differences for basic regular expressions are
summarized afterwards.

The fundamental building blocks are the regular expressions that
match a single character. Most characters, including all letters
and digits, are regular expressions that match themselves. Any
metacharacter with special meaning may be quoted by preceding it
with a backslash.

A bracket expression is a list of characters enclosed by [ and ].
It matches any single character in that list; if the first
character of the list is the caret ^ then it matches any character
```

[1] grep est l'acronyme de *global regular expression print.*

[2] C'est-à-dire incluant la possibilité d'utiliser la complémentation.

not in the list. For example, the regular expression [0123456789] matches any single digit when the regular expression [^0123456789] matches all characters that are not digits.

Within a bracket expression, a range expression consists of two characters separated by a hyphen. It matches any single character that sorts between the two characters, inclusive, using the locale's collating sequence and character set. For example, in the default C locale, [a-d] is equivalent to [abcd]. Many locales sort characters in dictionary order, and in these locales [a-d] is typically not equivalent to [abcd]; it might be equivalent to [aBbCcDd], for example. To obtain the traditional interpretation of bracket expressions, you can use the C locale by setting the LC_ALL environment variable to the value C.

Finally, certain named classes of characters are predefined within bracket expressions, as follows. Their names are self explanatory, and they are [:alnum:], [:alpha:], [:cntrl:], [:digit:], [:graph:], [:lower:], [:print:], [:punct:], [:space:], [:upper:], and [:xdigit:]. For example, [[:alnum:]] means [0-9A-Za-z], except the latter form depends upon the C locale and the ASCII character encoding, whereas the former is independent of locale and character set. (Note that the brackets in these class names are part of the symbolic names, and must be included in addition to the brackets delimiting the bracket list.) Most metacharacters lose their special meaning inside lists. To include a literal] place it first in the list. Similarly, to include a literal ^ place it anywhere but first. Finally, to include a literal - place it last.

The period . matches any single character. The symbol \w is a synonym for [[:alnum:]] and \W is a synonym for [^[:alnum]].

The caret ^ and the dollar sign $ are metacharacters that respectively match the empty string at the beginning and end of a line. The symbols \< and \> respectively match the empty string at the beginning and end of a word. The symbol \b matches the empty string at the edge of a word, and \B matches the empty string provided it's not at the edge of a word.

A regular expression may be followed by one of several repetition operators:

?	The preceding item is optional and matched at most once.
*	The preceding item will be matched zero or more times.
+	The preceding item will be matched one or more times.
{n}	The preceding item is matched exactly n times.
{n,}	The preceding item is matched n or more times.
{n,m}	The preceding item is matched at least n times, but not more than m times.

Two regular expressions may be concatenated; the resulting regular expression matches any string formed by concatenating two substrings that respectively match the concatenated

```
subexpressions.

Two regular expressions may be joined by the infix operator |; the
resulting regular expression matches any string matching either
subexpression.

Repetition takes precedence over concatenation, which in turn
takes precedence over alternation. A whole subexpression may be
enclosed in parentheses to override these precedence rules.

The backreference \n, where n is a single digit, matches the
substring previously matched by the nth parenthesized
subexpression of the regular expression.

In basic regular expressions the metacharacters ?, +, {, |, (, and
) lose their special meaning; instead use the backslashed versions
\?, \+, \{, \|, \(, and \).

Traditional egrep did not support the { metacharacter, and some egrep
implementations support \{ instead, so portable scripts should avoid {
in egrep patterns and should use [{] to match a literal {.

GNU egrep attempts to support traditional usage by assuming that {
is not special if it would be the start of an invalid interval
specification. For example, the shell command egrep '{1' searches
for the two-character string {1 instead of reporting a syntax
error in the regular expression. POSIX.2 allows this behavior as
an extension, but portable scripts should avoid it.
```

1.2 Syntaxe d'utilisation de grep

La syntaxe d'utilisation de base de `grep` est la suivante :

```
grep motif fichier
```

où `motif` est une expression régulière étendue représentée à l'aide de la syntaxe qui a été décrite dans le paragraphe précédent et où `fichier` représente le nom d'un fichier à examiner.

La sortie produite par cette ligne de commande est alors simplement la liste – qui est dirigée vers la sortie standard, c'est-à-dire affichée à l'écran – de toutes les lignes du fichier `fichier` qui contiennent le motif `motif`, c'est-à-dire qui appartiennent au langage rationnel représenté par l'expression régulière `motif`.

Exemple : La ligne de commande

```
grep '^y.*y$' /usr/dict/words'
```

recherche dans le fichier `/usr/dict/words`, c'est-à-dire dans le dictionnaire Unix des mots anglais, l'ensemble des lignes qui commencent par la lettre `y` et qui se terminent par la lettre `y`. Cette commande renvoie ici la liste :

```
yeasty
yeomanry
yesterday
```

2 Travail demandé

On demande de réaliser un programme Java qui implémente *partiellement* la commande `grep` telle qu'elle a été décrite au paragraphe précédent.

On dégagera pour cela un sous-ensemble de l'ensemble des expressions régulières généralisées décrites au paragraphe précédent, qui sera suffisamment simple pour ne pas alourdir artificiellement le projet par trop de prises en compte de détails de syntaxe. Le champ d'application de la commande `grep` implémentée dans le cadre de ce projet se réduira bien entendu aux catégories d'expressions régulières que vous aurez choisies.

Indication : L'algorithme sur lequel s'appuie la commande `grep` est bien entendu celui qui reconnait si un mot donné appartient au langage rationnel représenté par une expression régulière donnée. Cet algorithme pourra typiquement être implémenté en construisant un automate fini de petite taille équivalent à une expression régulière donnée, puis en testant si chaque ligne d'un fichier est reconnue par cet automate. La construction d'un tel automate pourra être effectuée en utilisant par exemple l'algorithme de Glushkov décrit dans [2].

3 Extensions possibles

L'instruction `grep` possède de nombreuses options que l'on trouvera à l'entrée `grep` du manuel Unix (faire `man grep` sous Unix). On pourra ainsi étendre le sujet en implémentant en Java une ou plusieurs de ces options d'utilisation de la commande `grep`.

Une autre extension consiste à étudier l'impact de la modification du noyau de la commande `grep` en réalisant des implémentations comparatives d'autres algorithmes de reconnaissance de mots par des expressions régulières (voir [2] à ce sujet).

References

[1] Abrahams P.W., Larson B.R., *Unix for the impatient*, Addison Wesley, 1996 [3]

[2] Pin J.E., *Automates finis et applications*, Polycopié de majeure, Département d'informatique, Ecole Polytechnique

[3] Traduction française publiée chez Vuibert Informatique