Context grep

Charles L. A. Clarke   Gordon V. Cormack

Department of Computer Science
University of Waterloo, Waterloo, Ontario N2L 3G1, Canada
{claclark,gvcormac}@plg.uwaterloo.ca

Technical Report CS-96-41
November 30, 1996

Abstract

The Unix `grep` utility searches a text file for a pattern defined by a regular expression and prints the lines containing the pattern. Unfortunately, a line of text is not always the appropriate unit for search and retrieval. Our solution is to treat a newline as an ordinary character and to allow pattern matching across the entire file. Several issues must be addressed to make this free text searching useful. In particular, the standard approach of taking the "leftmost longest match" tends to select inappropriately large fragments of the text. Instead we follow the principle of always taking the shortest match and allow these matches to overlap but not nest. The resulting tool is novel, expressive and simple. Matches can be reported across lines and multiple matches can be reported within a line. Appropriate structure may be imposed by using a regular expression to define a search universe. Elements may then be selected from this universe by matching with a second regular expression.

The current release of the search tool is available at

`ftp://plg.uwaterloo.ca/pub/mt/cgrep`

A detailed man page for the program is included as part of the distribution.

1 Introduction

The most widely available regular expression search utility, the Unix `grep` command, imposes a very rigid structure on search and retrieval. Patterns must be contained on a single line, and only lines containing or not containing the pattern may be selected for retrieval. Several attempts have been made to loosen the restrictions. The `agrep` search program allows the user to define record delimiters for searching [10]. The structural regular expressions provided in the text editor `sam` allow the use of regular expressions under a "left-most longest match" rule to divide a file into records for searching [7, 8].

In a related paper [3] we argue that the utility of a regular expression search program is significantly increased if it is based on an approach that may be summarized by the following three rules:

1. Don't divide the input text into lines (or other records).

2. Report only shortest matches (the *shortest-match rule*).

3. Allow matches to overlap, but not to nest (the *overlap rule*).

Together, the first two rules allow matches across multiple lines, multiple matches within single lines, and matching over structural elements defined by regular expressions. Together, the last two rules give us a simple way to describe precisely the matches that will be reported by the search program: every substring of the input text that matches the regular expression and that does not itself contain a matching substring. The approach is motivated by our experience with structured text databases [1, 4]. We have termed the combination of the shortest-match rule and the overlap rule *shortest substring matching*.

The approach has been used as the basis for a search tool, `cgrep`, that has proved to be of significant value for a variety applications. In searching and organizing folders of mail messages and news articles, the tool allows the selection and extraction of articles based on combinations of the contents of the header lines and patterns occurring in the body. The tool has been used to assist in sorting, organizing and indexing structured documents from the TREC collection [5]. Other applications have included the extraction of World-Wide-Web hypertext links (URL's) from NetNews articles, searching files of machine code in binary format, searching comments in source code files, and counting actual occurrences of patterns in files (as opposed to lines containing the patterns). Formulating equivalent operations with existing text processing tools has proven difficult and frustrating. This report examines the function and performance of the tool by way of several examples.

The current release of the `cgrep` search tool is available at

`ftp://plg.uwaterloo.ca/pub/mt/cgrep`

A detailed man page for the program is included as part of the distribution.

2 Basic Searching

We begin with simple document processing examples. As a group, these examples are intended to demonstrate the utility of `cgrep` and the necessity of both the shortest-match rule and the overlap rule.

The regular expression syntax of `cgrep` is based on that of POSIX 1003.2 extended regular expressions[6]. This regular expression syntax is based in turn on that of `egrep`. In addition, `cgrep` provides macros, an intersection operator, and several other useful extensions. In particular, the characters "<" and ">" may be used to match the beginning and end of file respectively.

Example 1: Multiple Matches within a Line

Sometimes interesting text patterns can be quite short and multiple instances may be found on a single line. The command

```
cgrep '\\cite\{.*\}' paper.tex
```

will pull a list of citations from a document in LaTeX format. If multiple citations appear on a line, all will be reported. Occurrences of "\", "{" and "}" appearing in the pattern are escaped with the backslash character "\" to prevent special meanings they have to `cgrep`. In all cases, an escape sequence consisting of a "\" followed by any punctuation character may be used to represent the literal punctuation mark, avoiding any special meaning of the character.

Example 2: Multi-Line Matching

Matching across lines is straightforward with `cgrep`. The command

```
cgrep '^.*United[[:space:]]+States.*$' constitution.txt
```

matches lines containing the name of the country. The two words composing the name may be separated by one or more white-space characters. The bracket expression "`[[:space:]]`" matches any character for which the *ctype* macro `isspace` would return true. This search will report both

```
We the people of the United States, in order to form a more
```

and

```
age of thirty years, and been nine years a citizen of the United
States and who shall not, when elected, be an inhabitant of that
```

The shortest match rule guarantees that only the smallest numbers of lines necessary are printed.

Example 3: Extracting Structural Elements

Under some mild assumptions, the command

```
cgrep '/\*.*\*/' *.c
```

will print all comments in a group of C source files. The character "`*`" is escaped, by preceding it with a backslash, to prevent its interpretation as a closure operator. The assumptions necessary for the extraction to work correctly are basically that comments contain no nested start patterns ("`/*`"), that the comment start and end patterns do not appear in string constants, and that comments are not ill-formed in conditionally-compiled code (e.g. in "`#ifdef`'s"). Many comments in C source code will satisfy these assumptions and it is possible to write `cgrep` expressions to detect exceptions — an example will be provided later.

Example 4: Searching SGML Text

SGML files usually contain substantial structure that is not line oriented. HTML documents provide an example of documents formatted (approximately) according to SGML conventions. The command

```
cgrep -i '\<title\>.*\</title\>' /users/*/public_html/*.html
```

might be used to extract the titles of HTML documents from the top level of users' publicly accessible World-Wide-Web documents. The "`-i`" flag removes lower and upper case distinctions for matching purposes. The "`<`" and "`>`" characters are escaped, as they are used by `cgrep` to match file start and end characters. In this example and in previous examples the shortest substring matching guarantees that the patterns represented the beginning and end of a match are properly associated with one another.

Under a longest-match rule, which is standard for regular expression search [6], the command of Example 2 would report a single match for any text containing the name of the country. The match would cover the entire file, starting at the beginning of the first line and continuing to the end of the last line. The command of Example 1 would report a single match starting at the beginning of the first citation and continuing to the end of the last closing brace. Similar problems occur with Examples 3 and 4.

It can be shown that it is always possible to explicitly re-write a regular expression to accept only shortest matches [3, Theorem 5]. This is quite easy in some cases, such as the second example, where the pattern

```
^[^\n]*United[[:space:]]+States[^\n]*$
```

is sufficient. The escape sequence "`\n`" matches the newline character.

Example 3 requires more work. It's necessary to recognize the end pattern "`*/`" while allowing the comment to include "`*`" and "`/`" characters that are not part of the end pattern. One possibility is:

```
/\*([^/*]|/+[^/*]|\*+[^/*])*(/*|\**)\*/
```

The other examples exhibit similar difficulties.

Example 5: Intersection of Patterns

In addition to the standard POSIX 1003.2 operators, `cgrep` accepts "`&`" for the intersection of two regular expressions. The precedence of the intersection operator is the same as that of union ("`|`"). The union and intersection operators associate left to right.

The intersection operator "`&`" provides a way to match intervals of text that satisfy a combination of patterns. The command

```
cgrep '.{0,100}&.*God.*&.*Moses.*' exodus.txt
```

prints all intervals of text that are 100 characters or less in size and contain both "God" and "Moses". Allowing overlap between matches is important so that passages such as

```
Moses took the
        rod of God
```

and

```
God in his hand.

004:021 And the LORD said unto Moses
```

can both be reported.

3 Search Universes

The "`-U`" option of `cgrep` is used to specify a search universe. A second regular expression is used to select members of the search universe that contain an instance of the regular expression.

Example 6: Lines

With an appropriate universe definition, `cgrep` can mimic the behavior of standard Unix search utilities. The command

```
cgrep -U '^.*$' '[Vv]egetarians|[Vv]egans' mbox
```

prints all lines that contain references to vegetarians or vegans. Using the search universe of lines, the capabilities of `cgrep` are a superset of those of `egrep`.

Example 7: Groups of Lines

Groups of several lines can be used as a search universe. In this circumstance, the utility of allowing overlapping matches becomes obvious, as each group of lines will overlap with several others. The command,

```
cgrep -U '^.*^.*^.*$' '.*God.*&.*Moses.*' exodus.txt
```

will print groups of three lines that contain both “God” and “Moses”.

At this point, it is worth discussing the exact meaning of the “`^`” and “`$`” operators in `cgrep` and the default behavior for printing matches. In standard search tools, the “`^`” operator matches the beginning of a line and “`$`” matches the end of a line. Since `cgrep` allows matching across line boundaries, “`^`” and “`$`” have been adapted slightly: “`^`” matches a newline character or the beginning of the file; “`$`” matches a newline or the end of the file. In other words, “`^`” is equivalent to “`\n|<`” and “`$`” is equivalent to “`\n|>`”.

Following these definitions, each match reported by the command of Example 6 would be preceded and terminated by a newline character. Visually, each match would appear to be preceded by a blank line.

We have found it convenient to automatically strip off these leading newlines. Similarly, it has been found convenient to terminate matches that would not normally end in a newline by adding a newline character. This automatic reformatting of the output is particularly helpful when the results of a `cgrep` search are to be piped to other Unix commands.

An option (“`-strict`”) is provided to prevent automatic reformatting of the output. The “`-strict`” option also changes the meaning of the “`^`” and “`$`” operators so that they match only the beginning and end of file, becoming equivalent to the “`<`” and “`>`” operators. In addition to automatic reformatting, an option (“`-tag`”) allows the user to specify start and end tags to be placed around matches.

Example 8: Mail Messages

The command

```
cgrep -U '^From .*(^From |>)' 'vegetarian' mbox
```

extracts all mail messages that contain the word “vegetarian”. Mail messages are assumed to start with the word “From” at the beginning of a line followed immediately by a space character. A mail message ends when another of these patterns is seen or the end of file is encountered. Unless it occurred at the end of the input file each match reported will be terminated by a line containing the word “From” followed by a space. In order to create a validly formatted mail folder the output must be piped through

```
sed '/^From $/d'
```

or equivalently

```
cgrep -V '^.*$' '^From $'
```

The latter command prints all lines that do not contain the pattern “`^From $`”. The `-V` option is used to specify an *anti-search universe*, which is the subject of the next example.

Example 9: Anti-Search Universes

Members of an anti-search universe are reported if they do not contain a match to a second regular expression. The command

```
cgrep -list -V '/\*.*/\*' '\*/' *.c
```

will print the names of files containing a comment with a nested comment-start pattern ("`/*`"), violating one of the assumptions of Example 3. The "`-list`" option specifies that only the names of files containing matches be reported.

4 Other Features of `cgrep`

Several other features of `cgrep` assist in its use: A macro facility is provided to help formulate search and retrieval operations involving commonly-used subexpressions. As already mentioned, a tagging facility allows the user to define delimiters to be placed around each match reported.

The syntax for macros is slightly ugly. Macro calls come in two varieties: *fast* and *tedious*. A fast call consists of a "`@`" character followed by an single alphabetic character. A tedious macro call has the form:

`[@`*name*`(`*parameter0*`,` *parameter1*`, ...)]`

where each of the up to 9 parameters is a regular expression. If the macro requires no parameters, the bracket-enclosed parameter list is omitted completely.

Macro definitions are placed in files that are read on program start-up. Any definitions in the file "`.cgreprc`" in the user's home directory are read first. The "`-defs`" option may be used to specify additional macro files. The definition of fast and tedious macros are identical. Any un-parameterized, single-letter macro is automatically usable as either a fast macro or a tedious macro. An un-parameterized macro definition has the form:

name`=`*regular-expression*

and a parameterized macro definition has the form:

name`#`*n*`=`*regular-expression*

where the number of parameters is indicated by a single digit following the "`#`" character. Within the body of a parameterized macro, the actual parameters may be referenced as "`#1`" through "`#9`" A macro name must start with an alphabetic character, and may include only alphanumeric characters and the character "`_`".

Example 10: Macros

Standard macros for handling mail may be defined in "`.cgreprc`" in the user's home directory:

```
Mail=^From .*(^From |>)
From#1=^From:[^$]*#1
Re#1=^Subject:[^$]*#1
```

The command

```
cgrep -U '[@Mail]' '(.*[@From([Cc]owan)].*)&(.*[@Re((fwd))].*)' mbox
```

would then extract all mail messages forwarded by Cowan.

Example 11: Tagging

Since the output from `cgrep` often consists of multi-line text fragments, the tagging facility allows matches to be easily distinguishable from one another and simplifies further searching on the output. The command,

```
cgrep -tag '<q>\n' '</q>\n' -U '^.*^.*^.*$' '.*God.*&.*Moses.*' exodus.txt
```

will tag each excerpt in the style of SGML. An "`@`" appearing in a tag is replaced by the name of the file where the occurrence was found. Any escape sequence valid in a regular expression (such as "`\n`") is valid in a tag. Thus, in this example, each of the tags generated by the command will appear alone on a line.

Example 12: Searching Binary Files

Files of binary data may be searched using `cgrep`. The command

```
cgrep '[^[:print:]][[:print:]]{4,}[\n\0]' cgrep
```

reports strings of four or more printable characters ending in a newline or null character that appear in the executable file "`cgrep`". This search is similar that performed by the Unix `strings` command.

Escape sequences allow non-printable characters, such as the null character ("`\0`"), to be used in a pattern. These escape sequences follow the syntax of ANSI C, including the sequences for hexadecimal and octal constants. Escape sequences not given special meaning by ANSI C always represent the literal character following the "`\`".

Each result reported by `cgrep` will begin with a non-printable character and end with a newline or null character. This example further illustrates how a longest-match requirement ("four or more") can be satisfied with a shortest-match search.

5 Performance

The table in figure 1 gives performance figures for the examples running over typical input text, reading and writing to a disk file. The programs were executed on a SPARCServer 670MP under SunOS 5.3 concurrently with the "normal" activities of a dozen or so other users, mostly Programming Language and Theory of Computation researchers in the Computer Science Department at the University of Waterloo. The input sources described as "US Constitution" and "Exodus" are electronic texts from Project Gutenburg [9]. The LaTeX source is that of reference [4]. The source to `nethack` supplied an example of C source code with lots of oddly structured comments. The HTML source is a collection of local users' World-Wide-Web home pages. The personal email was received by the author over several months. Elapsed times include both system and user times, and are averaged over several runs.

The figures show that the program is generally quite usable to search large files interactively. One exception is the command of Example 5, which processed its input at less than 1KB per second. This poor performance is due to a large number of active states during the matching process. The same command searching the entire Old Testament would take more than an hour. One solution to this problem is to pre-filter the input with a command such as that of Example 11, tagging each group of lines, and then running the actual search on the result.

The performance of `cgrep` is often inferior to more specialized tools on equivalent tasks. The command

```
egrep  '[Vv]egetarians|[Vv]egans' mbox
```

is equivalent to the `cgrep` command of Example 6 but is more than twice as fast, running in just over five seconds on the same input. The Unix command

```
strings -a cgrep
```

is equivalent to the `cgrep` command of Example 12 but is nearly three times faster, running in just under a tenth of a second. On the other hand, `egrep` and `strings` cannot perform the search and retrieval operations in the other examples.

In other cases, `cgrep` has proven to be faster than special purpose programs. The program `readmsg` is part of the *elm* mail package and can be used to extract mail messages based on patterns in the message body. The command

```
readmsg -ah -f mbox vegetarian
```

performs the same function as the `cgrep` command in Example 8 but takes 56 seconds. In this case, `cgrep` is 17% faster. Furthermore, `readmsg` cannot be used to perform more complex retrieval operations, such as that of Example 10.

6 Concluding Remarks

The `cgrep` utility is a unique and useful tool for search and retrieval. Its usefulness comes by freeing regular expressions from record boundaries and by using these free matches as the subject of further searching. Shortest substring matching reports *all* occurrences of a search pattern that do not themselves contain shorter occurrences of the pattern. Left-most longest match does not have this simple property — whether or not a match is reported depends on the other contents of the text.

References

[1] Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. Schema-independent retrieval from hetrogeneous structured text. In *Fourth Annual Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, April 1995.

Example Number	Input Description	Input Size (bytes)	Number of Matches	Output Size (bytes)	Elapsed Time (seconds)
1	LaTeX source	68K	33	0.49K	0.13
2	US Constitution	46K	85	5.2K	0.23
3	`nethack` source	1.6M	4647	231K	2.9
4	HTML source	544K	79	3.5K	0.55
5	Exodus	193K	49	2.6K	220
6	Personal email	12M	331	21K	11
7	Exodus	193K	88	12K	3.1
8	Personal email	12M	214	2.8M	46
9	`nethack` source	1.6M	90	0.75K	0.47
10	Personal email	12M	17	164K	138
11	Exodus	193K	88	13K	3.1
12	`cgrep` executable	46K	430	7.5K	0.27

Figure 1: Performance of `cgrep`.

[2] Charles L. A. Clarke and Gordon V. Cormack. On the use of regular expressions for searching text. Technical Report CS-95-07, University of Waterloo Computer Science Department, February 1995.

[3] Charles L. A. Clarke and Gordon V. Cormack. On the use of regular expression for searching text. *ACM Transactions on Programming Languages and Systems*, 1997. To appear. An earlier version of the paper was distributed as a technical report [2].

[4] Charles L. A. Clarke, Gordon V. Cormack, and Forbes J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56, 1995.

[5] Donna Harman. Overview of the first TREC conference. In *Proceedings of the 16th Annual International ACM SIGIR Conference*, pages 36–47, Pittsburgh, PA, June 1993.

[6] Institute of Electrical and Electronics Engineers. *Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 2 (Shell and Utilities) — Section 2.8 (Regular Expression Notation)*, September 1992. IEEE Std 1003.2.

[7] Rob Pike. Structural regular expressions. *Proceedings of the European UNIX User's Group Conference*, 1987.

[8] Rob Pike. The text editor `sam`. *Software — Practice and Experience*, 17(11):813–845, November 1987.

[9] Project Gutenberg. `ftp://mrcnext.cso.uiuc.edu/`.

[10] Sun Wu and Udi Manber. `agrep` —- A fast approximate pattern-matching tool. In *USENIX Winter 1992 Technical Conference*, pages 153–162, San Francisco, January 1992.