CS29206 Systems Programming Laboratory
Spring 2024

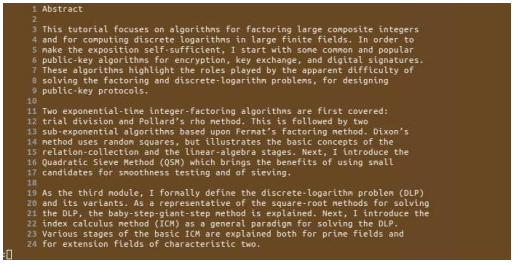
Regular Expressions

Abhijit Das
Pralay Mitra

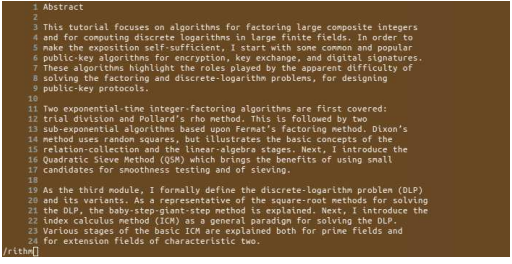
Regular expressions

- Same as those introduced in connection with regular languages in your FLAT course.
- Constructs are different.
- Used by less, grep, sed, awk, shells, and many text editors like vi and emacs.
- We use less to demonstrate the matches.
- Running with the –N option lets less show the line numbers.
- In the viewing mode, you can type / (forward slash) followed by a regular expression.
- All matches found are highlighted.
- Searches are made in each line.
- The newline character is not allowed in regular expressions.

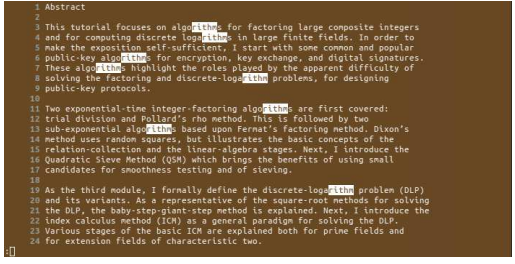
Viewing matches with less



```
Abstract

This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

As the third module, I formally define the discrete-logarithm problem (DLP)
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
Various stages of the basic ICM are explained both for prime fields and
for extension fields of characteristic two.
```



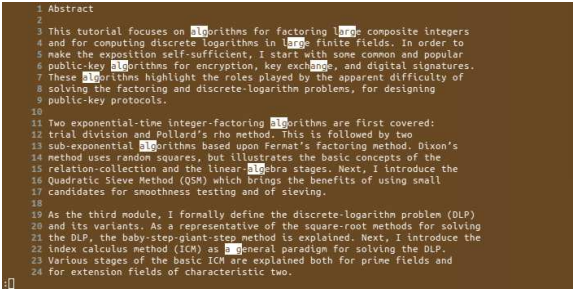
```
Abstract

This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

As the third module, I formally define the discrete-logarithm problem (DLP)
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
Various stages of the basic ICM are explained both for prime fields and
for extension fields of characteristic two.
/rithm
```



```
Abstract

This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

As the third module, I formally define the discrete-logarithm problem (DLP)
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
Various stages of the basic ICM are explained both for prime fields and
for extension fields of characteristic two.
:
```

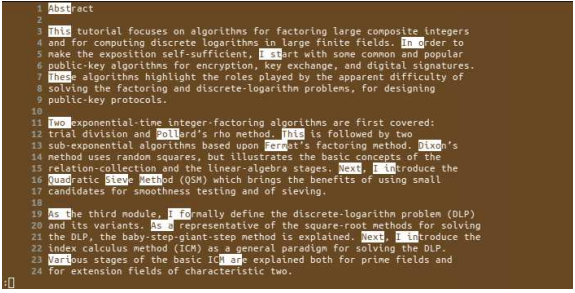
Matching any character

- Period (`.`) matches any single character.
- The pattern `a.g` matches the following.



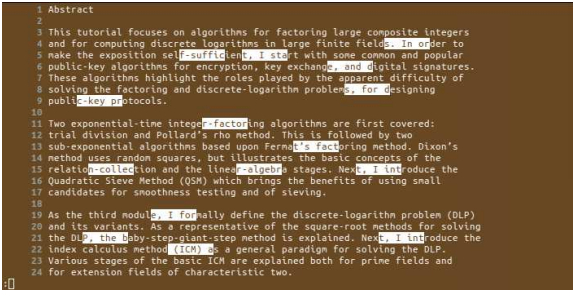
Matching a set of characters

- Delimit the set between `[` and `]`.
- `[Tt]` matches upper- or lower-case T.
- `[AEIOU]` matches any upper-case vowel.
- `[a-z]` matches any lower-case letter.
- `[a-zA-Z0-9]` matches any alphanumeric character.
- The regular expression `[A-Z][a-z ][a-z ].` gives the following result.



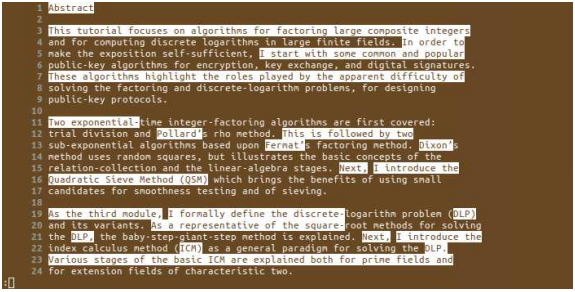
Negation of a set of characters

- Use `^` after `[`.
- `[^ ]` matches any non-space character.
- `[^aeiouAEIOU]` matches any character other than the vowels.
- `[^a-zA-Z]` matches any non-alphabetic character.
- The output for the search `[^AEIOU][^a-zA-Z ].....[a-drt]` is given below.



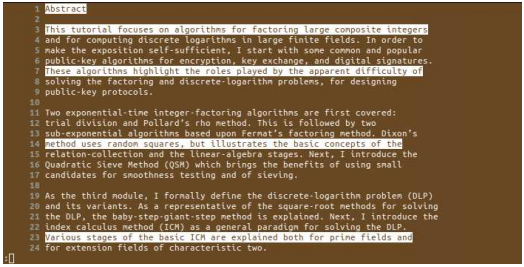
Matching zero or more characters

- Use `*`.
- `.*` matches any string. `..*` matches any non-empty string.
- `[a-z]*` matches any sequence of lower-case letters.
- `[^a-zA-Z]*` matches any sequence of non-alphabetic characters.
- Result of searching `[A-Z][a-zA-Z ]*[^ ]` is given below.
- Longest possible matches are reported, starting as early as possible.



Match at the beginning or at the end of a string

- If you want the match to start from the beginning, use `^` as the first symbol.
- If you want the match to finish at the end, use `$` as the last symbol.
- The pattern `^[A-Z][a-z]*` matches the first word of a line if the line starts with a capital letter.
- The pattern `[a-z]*$` matches the last word of a line if the line ends with a lower-case letter, and if the last word consists of lower-case letters only.
- The result for searching `^[A-Za-z, ]*$` is given below.



Quoting the special characters

- Use `\.`, `\[`, `\]`, `\*`, `\^`, `\$`, `\\`, and `\/`. The last one is used during substitution.
- `-` need not be quoted.
- `[a-z]*\.` matches the last word with the period in a sentence if the word consists of lower-case letters only. If the last word contains characters other than the lower-case letters, then the match starts after the last such character.
- The pattern `[a-z]*-[a-z-]*.*\.` matches as follows.

```
Abstract

This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method, Dixon's
method uses random squares, but illustrates the basic concepts of the
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

As the third module, I formally define the discrete-logarithm problem (DLP)
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
Various stages of the basic ICM are explained both for prime fields and
for extension fields of characteristic two.
```

CS29206 Systems Programming Laboratory
Spring 2024

Introduction to grep

Abhijit Das
Pralay Mitra

The Unix command grep

- Abbreviation of *Global Regular Expression Print*.
- Locates lines that contains matches of regular expression(s).
- May or may not highlight the match.
- Options of grep enable you to do a lot of tasks with the matched lines.
- You run grep as:

```
grep <OPTIONS> <PATTERN> <FILE(S)>
```

- The pattern is a regular expression.
- A regular expression may contain characters (like *) having special meanings to the shell, so it is preferable to quote the pattern.
- Single (forward) quotes are recommended.
- Quoting also enables you to search for patterns containing space.

The input file used in the examples

The file textfile.txt

```
1   Abstract
2
3   This tutorial focuses on algorithms for factoring large composite integers
4   and for computing discrete logarithms in large finite fields. In order to
5   make the exposition self-sufficient, I start with some common and popular
6   public-key algorithms for encryption, key exchange, and digital signatures.
7   These algorithms highlight the roles played by the apparent difficulty of
8   solving the factoring and discrete-logarithm problems, for designing
9   public-key protocols.
10
11  Two exponential-time integer-factoring algorithms are first covered:
12  trial division and Pollard's rho method. This is followed by two
13  sub-exponential algorithms based upon Fermat's factoring method. Dixon's
14  method uses random squares, but illustrates the basic concepts of the
15  relation-collection and the linear-algebra stages. Next, I introduce the
16  Quadratic Sieve Method (QSM) which brings the benefits of using small
17  candidates for smoothness testing and of sieving.
18
19  As the third module, I formally define the discrete-logarithm problem (DLP)
20  and its variants. As a representative of the square-root methods for solving
21  the DLP, the baby-step-giant-step method is explained. Next, I introduce the
22  index calculus method (ICM) as a general paradigm for solving the DLP.
23  Various stages of the basic ICM are explained both for prime fields and
24  for extension fields of characteristic two.
25
```

Search examples

```
$ grep method textfile.txt
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
$ grep 'method ' textfile.txt
method uses random squares, but illustrates the basic concepts of the
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
$ grep method[ \.] textfile.txt
grep: Invalid regular expression
$ grep 'method[ \.]' textfile.txt
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
$ grep '[a-z]*-[a-z-]*.*\.' textfile.txt
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
relation-collection and the linear-algebra stages. Next, I introduce the
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
$
```

Making searches for multiple patterns

- Use the option `-e` multiple times.

```
$ grep -e 'method' -e 'algorithm' textfile.txt
This tutorial focuses on algorithms for factoring large composite integers
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
$
```

- This option also helps you specify patterns starting with `-`.

```
$ grep '-key' textfile.txt
grep: invalid option -- 'k'
Usage: grep [OPTION]... PATTERNS [FILE]...
Try 'grep --help' for more information.
$ grep -e '-key' textfile.txt
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
$
```

The inverted search

- The option `-v` prints the lines that do not contain matches.

Lines not containing upper-case letters

```
$ grep -v '[A-Z]' textfile.txt

public-key algorithms for encryption, key exchange, and digital signatures.
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

method uses random squares, but illustrates the basic concepts of the
candidates for smoothness testing and of sieving.

for extension fields of characteristic two.

$
```

Case-insensitive search using the option –i

```
$ grep 'method' textfile.txt
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
$ grep -i 'method' textfile.txt
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
Quadratic Sieve Method (QSM) which brings the benefits of using small
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
$
```

Word-based search using the option –w

Lines containing upper-case letters

```
$ grep '[A-Z]' textfile.txt
Abstract
This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
These algorithms highlight the roles played by the apparent difficulty of
Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
As the third module, I formally define the discrete-logarithm problem (DLP)
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
Various stages of the basic ICM are explained both for prime fields and
$
```

Lines containing single-letter upper-case words

```
$ grep -w '[A-Z]' textfile.txt
make the exposition self-sufficient, I start with some common and popular
relation-collection and the linear-algebra stages. Next, I introduce the
As the third module, I formally define the discrete-logarithm problem (DLP)
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
$
```

Printing line numbers and counting

- Use the option `-n` to print the line numbers before the printed lines.

Lines ending with non-alphabetic letters

```
$ grep '[^a-zA-Z]$' textfile.txt
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
Two exponential-time integer-factoring algorithms are first covered:
candidates for smoothness testing and of sieving.
As the third module, I formally define the discrete-logarithm problem (DLP)
index calculus method (ICM) as a general paradigm for solving the DLP.
for extension fields of characteristic two.
$ grep -n '[^a-zA-Z]$' textfile.txt
6:public-key algorithms for encryption, key exchange, and digital signatures.
9:public-key protocols.
11:Two exponential-time integer-factoring algorithms are first covered:
17:candidates for smoothness testing and of sieving.
19:As the third module, I formally define the discrete-logarithm problem (DLP)
22:index calculus method (ICM) as a general paradigm for solving the DLP.
24:for extension fields of characteristic two.
$
```

- Use the option `-c` to print only the number of lines.

```
$ grep -c '[^a-zA-Z]$' textfile.txt
7
$
```

Search recursively in subdirectories

- Use the option `-r` or `-R`.

Recursive search for nodep in the current directory (.)

```
$ grep -r 'nodep' .
./libstaque/static/defs.h:typedef node *nodep;
./libstaque/static/stack.h:typedef nodep stack;
./libstaque/static/queue.h:   nodep front;
./libstaque/static/queue.h:   nodep back;
./libstaque/shared/defs.h:typedef node *nodep;
./libstaque/shared/stack.h:typedef nodep stack;
./libstaque/shared/queue.h:   nodep front;
./libstaque/shared/queue.h:   nodep back;
$
```

- Use the option `-l` only to print the names of the files that match. This is valid without the flag `-r` or `-R` as well.

```
$ grep -r -l 'nodep' .
./libstaque/static/defs.h
./libstaque/static/stack.h
./libstaque/static/queue.h
./libstaque/shared/defs.h
./libstaque/shared/stack.h
./libstaque/shared/queue.h
$
```

Practice exercises

1. Write regular expressions for the following patterns.

   (a) A consonant (in upper or lower case).
   (b) A sequence of alphabetic letters containing no vowels.
   (c) A sequence of alphabetic letters containing exactly one vowel.
   (d) A sequence of alphabetic letters containing exactly two vowels.
   (e) A sequence of alphabetic letters containing at least one vowel.
   (f) A sequence containing no non-alphabetic characters.
   (g) An entire line containing no non-alphabetic characters.
   (h) An integer in the hexadecimal notation.
   (i) A line containing at least ten spaces.
   (j) A line containing at least ten non-space characters.

2. Write a grep command to find each of the following patterns in a text file.

   (a) A line starting with a tab.
   (b) The string `foo` or the string `bar`.
   (c) Both the strings `foo` and `bar`.
   (d) The string `foo` followed by the string `bar`.
   (e) The string `foo` followed but not immediately by the string `bar`. Should the line `"foobar bar"` match? Or the line `"foo foobar"`? What are the matches (if any)?

Practice exercises

3. Write a grep command to locate all the lines in a C file, that contain `printf`. Note that this `printf` should not come from `fprintf` or `sprintf`.

4. You have a C file `program.c` in which the strings (if any) do not contain the characters `{` and `}`. These characters are used solely to indicate the beginnings and the ends of blocks. Assume also there are no nested blocks in the same line, that is, situations like `{{}` or `{}}` or `{{}{}}` do not occur in a line. Write grep commands by which you can identify the following types of lines in `program.c`.

   (a) Lines in which a block is opened using `{` but not closed by `}` in the same line.
   (b) Lines in which a block is closed using `}` but not opened by `{` in the same line.
   (c) Lines in which a block is both opened by `{` and closed by `}` in the same line.

5. Professor Foojit has a text file `foonums.txt` in which each line contains five positive integers separated by single spaces. There is no extra space at the beginning or at the end of any line. The integers may be arbitrarily large, but do not contain leading zero digits. Professor Foojit considers all integers in the range 500-5000 unlucky. Out of these unlucky numbers, the number 876 is considered super-unlucky. Write grep commands by which you can identify the following types of lines in `foonums.txt`.

   (a) Lines that do not contain the super-unlucky number.
   (b) Lines that do not contain any unlucky number.
   (c) Lines that contain one or more unlucky numbers, neither of which is super-unlucky.
   (d) Lines that contains the super-unlucky number along with (at least) another unlucky number (which may again be super-unlucky).

Practice exercises

6. Write a command by which you can find all the files in the current directory, with execute permission of the owner. Use `ls -l` in tandem with `grep`. What about only the non-directory files with the same permission?

7. As in the last exercise, write a command to find all the files in the current directory, whose sizes are at least 1 MB (take this as 10^6 bytes) each.

8. Study the format of the file `/etc/passwd`. Write grep commands to find the users with the following restrictions.

   (a) Users with 4-digit user IDs.
   (b) Users having `bash` as the login shell. Users with names like *Foobashki Barlov* or with login shells like `rbash` must be excluded in the search.

9. Study what the option `-o` does for grep. What if you run grep with both the options `-o` and `-n`?

10. As some of the previous exercises illustrate, grep may need to be invoked with a lot of search patterns, each with its `-e` option. Editing a long command line is sometimes very clumsy. Investigate how you can write the individual search patterns in a text file, and let grep read from that file.