Systems Programming Laboratory, Spring 2022

Text processing utilities

Abhijit Das
Arobinda Gupta

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

February 15, 2022

The roadmap

- Unix provides a lot of utilities to process text files (or stdin inputs).
  - **`sort`** sorts a file line by line.
  - **`uniq`** removes duplicate lines in a (sorted) file.
  - **`wc`** reports the counts of characters, words, and lines in one or more file(s).
  - ...
- Here, we will focus on a few of more sophisticated text-processing tools.
  - Pattern matching based on regular expressions is often very useful.
  - **`grep`** is a utility to do a lot of tasks on lines matching patterns.
  - **`sed`** selects lines in a file based upon line numbers or patterns, and can do a set of simple tasks on the selected lines.
  - **`awk`** is a full-fledged programming language targeted to handle text databases.

Regular Expressions

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

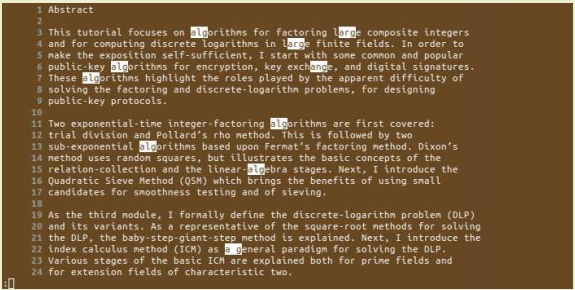
```
Abstract

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 algorithm 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.
:
```

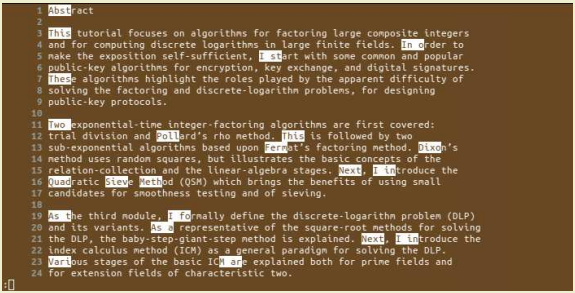
Matching any character

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



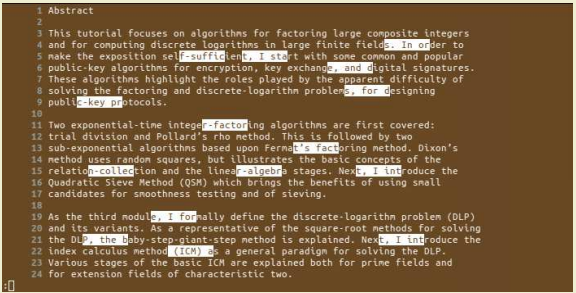
Matching a set of characters

- 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.



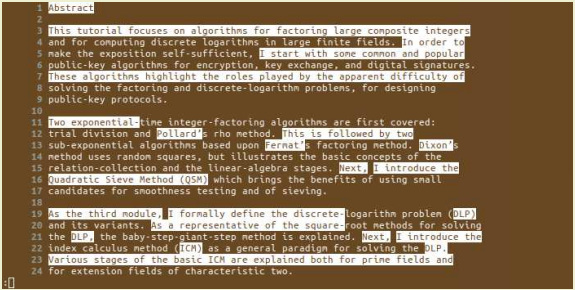
Negation of a set of characters

- 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.



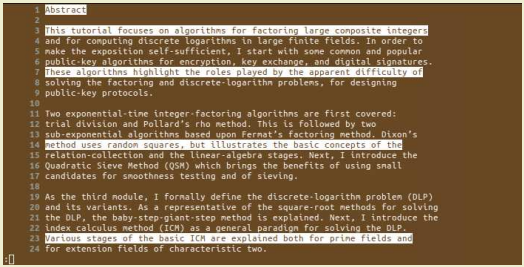
Matching zero or more characters

- 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-tine 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.
:
```