Lab 11: Regular Expressions

Comp Sci 1585
Data Structures Lab:
Tools for Computer Scientists

Outline

1 Introduction

2 grep
- Basic patters
- Variable-length patterns
- DIY character classes
- Anchors
- Groups
- Greedy vs. Polite matching

3 sed
- sed print command
- sed substitute command

4 C++ regex

What are Regular Expressions?

Regex is a language for describing patterns in strings.

Use regex for:

- Finding needles in haystacks.
- Changing one string to another.
- Pulling data out of strings.
- Finding lines or content in large text files

Outline

Looking for stuff with `grep`

- `$ grep` is Global Regular Expression Print
- `$ grep 'REGEX' FILES`: Search `FILES` for `REGEX` and print matches.
- If you don't specify `FILES`, `grep` will read `STDIN` (so you can pipe stuff into it).
  `$ echo 'text with REGEX in it' | grep 'REGEX'`
- `-C LINES` prints `LINES` lines of context around the match.
- `-v` prints every line that doesn't match (in**v**ert).
- `-i` Ignore case when matching.
- `-P` Use Perl-style regular expressions.
- `-o` Only print the part of the line the regex matches.

Outline

1 Introduction

2 grep
Basic patters
Variable-length patterns
DIY character classes
Anchors
Groups
Greedy vs. Polite matching

3 sed
sed print command
sed substitute command

4 C++ regex

Basic Patterns

- `.` Matches one of any character.
- `\w` Matches a word character (letters, numbers, and _).
- `\W` Matches everything `\w` doesn't.
- `\d` Matches a digit.
- `\D` Matches anything that isn't a digit.
- `\s` Matches whitespace (space, tab, newline, carriage return, etc.).
- `\S` Matches non-whitespace (everything `\s` doesn't match).
- `\` is also the escape character.

Outline

1 Introduction

2 grep
- Basic patters
- Variable-length patterns
- DIY character classes
- Anchors
- Groups
- Greedy vs. Polite matching

3 sed
- sed print command
- sed substitute command

4 C++ regex

Variable-length Patterns

- `{n}` matches *n* of the previous character.
- `{n,m}` matches between *n* and *m* of the previous character (inclusive).
- `{n,}` matches at least *n* of the previous character.
- `*` matches 0 or more of the previous character (`{0,}`).
- `+` matches 1 or more of the previous character (`{1,}`).
- `?` matches 0 or 1 of the previous character (`{0,1}`).

Outline

DIY character classes

- `[abc\d]` matches a character that is either a, b, c, or a digit.
- `[a-z]` matches characters between a and z.
- `^` negates a character class: `[^abc]` matches everthing except a, b, and c.

Outline

Anchors

- `^` forces the pattern to start matching at the beginning of the line.
- `$` forces the pattern to finish matching at the end of the line.
- `\b` forces the next character to be a word boundary.
- `\B` forces the next character to not be a word boundary.

Outline

Groups

- `(ab|c)` matches either 'ab' or 'c'.
- You can use length modifiers on groups, too: `(abc)+` matches one or more 'abc'
- One benefit of grouping is back-references. You can refer to the thing matched by the 1st group, etc.
- For example, `(ab|cd)\1` matches 'abab' or 'cdcd' but not 'abcd' or 'cdab'. Useful for repeats.

Outline

Greedy vs. Polite matching

- Regular expressions are greedy by default: they match as large of a block of string as they possibly can.
- Usually this is what you want, but sometimes it isn't.
- You can make a variable-length match non-greedy by putting a `?` after it.
- For example: `.+\.` vs. `.+?\.`
  (the former quits as late as possible,
  and latter is quits as early as possible)
- This means it will quit at the minimum length fitting criteria

Outline

sed: Editing with regex

- `$ sed` is a stream editor-use it for editing files or STDIN.
- It uses regular expressions to perform edits to text.
- `-r` enables extended regular expressions.
- `-n` makes `sed` only print the lines it matches.

Outline

The Print Command

- `$ sed -n '/regex/ p' f.txt` works pretty much exactly like `$ grep`.
- Use this to make sure your regex is matching what you want it to.
- You can also use `p` in conjunction with `s`, which we'll talk about immediately.

Outline

The Substitute Command

- `$ echo 'sometext...' | sed -i s/PATTERN/REPLACEMENT/` replaces the thing matched by PATTERN with REPLACEMENT (-i means in-place).
- Patterns can be any regular expression that we've talked about so far.
- Replacements can be plain text and/or back-references!
- `s/ / /g` makes the substitution global (every match on each line), e.g.,
  ```
  $ sed -i s/PATTERN/REPLACEMENT/g infile.txt
  ```
- `s/ / /i` makes the match case-insensitive.

Outline

C++ regex

- `http://en.cppreference.com/w/cpp/regex`
- `http://www.cplusplus.com/reference/regex/`

C++ regex


```
#include <iostream>
#include <iterator>
#include <string>
#include <regex>
int main(){
    std::string s = "Some people, when confronted
        with a problem, think I know, I'll use
        regular expressions.
        Now they have two problems.";

    std::regex self_regex("regular");
    if (std::regex_search(s, self_regex)){
        std::cout << "Text contains 'regular'\n";
    }

    std::regex long_regex("(\\w{7,})");
    std::string new_s = std::regex_replace(s,
                                           long_regex,
                                           "[$&]");

    std::cout << new_s << '\n';
}
```