Regular Expressions

CS14 - S26

Regular Expression

A regular expression (often abbreviated regex) is a string that specifies a particular pattern of text in other strings.

- Regex are derived from the concept of regular languages – take Theory of Computation for more info.
- Regex are regularly used to search and filter data.
- Many programs and programming languages support using regex. grep, sed, awk, python, javascript, vi, less, gmail, etc.
- grep == global, **regular expression**, print

Practical Usage

While different regex engines support a variety of features, the most basic usage is to report all strings or lines of text from an input that contain a specific pattern.

Examples:

```
apples?
```

- match strings containing "apple" or "apples"

```
^.*-.*$
```

- match strings containing a hyphen

```
\([0-9]{3}\)[0-9]{3}-[0-9]{4}
```

- match phone numbers of the format (XXX)XXX-XXXX

Complex regular expressions look like gobbledy gook, but they are made up of small building blocks!

Building Block 1 – Normal Text and Escaped Characters

Most characters like letters and numbers will be interpreted literally.

E.g., the regex `pears` matches text containing “pears”

Some characters have special meanings and in order to search for them literally, they must be “escaped” by putting a backslash before them.

The list of special characters varies between languages, but typically contains:

```
[]\^$.|?*+()
```

E.g., the regex `\?` matches text containing “?”, but `?` alone does not.

Building Block 2 – Character Classes

Un-escaped square brackets `[ ]` are used to define a set of characters that can be matched. Only one of the characters in the “class” needs to be present in order for the text to match.

`c[ua]t` matches “cat” and “cut”, but not “cit” or “cuat”

You can also specify ranges

`[a-zA-Z]` matches any lowercase or uppercase letter

And also specify that a class should be excluded rather than matched using `^` after the opening bracket.

`[^B]ot` matches “Tots”, “tots”, and “bots”, but not “Bots”

Predefined Character Classes

- `\w` matches any “word” characters, which include alphanumerics and underscores
- `\d` matches any digit
- `\s` matches any whitespace
- `.` matches any single character except newline

Building Block 3 – Quantifiers

After a literal character or a character class (collectively called tokens), you can specify that the token may or must appear a certain number of times using a quantifier.

`?` matches 0 or 1 of the previous token, making it optional

`de?r` matches "drop" and "powdery", but not "reindeer"

`+` matches 1 or more of the previous token

`de+r` matches "powdery" and "reindeer", but not "drop"

Quantifiers cont.

`*` matches 0 or more of the previous token

`de*r` matches all of “powdery”, “reindeer”, and “drop”

`{}` let you specify a number or range of times the token must appear

`ya{3}y` matches “yaaay”, but not “yay”

`\^_{1,3}\^` matches “^_^” and “^__^”, but not “^^” or “^_______^”

`br{2,}` matches “brr”, “brrrusque”, but not “bright”

Quantifiers cont.

All of the quantifiers can be used with character classes

`[wo]+w` matches "wow" and "wowow" as well as "oooooooooow"

`.*` matches everything except newline, including the empty string

Building Block 4 – Anchors

Anchors specify that the match must appear either at the beginning or end of a string. `^` anchors the beginning of a pattern, and `$` anchors the end.

> `^r` matches “rhino”, but not “gorrilla”
>
> `t$` matches “cat”, but not “ostrich”
>
> `^banana$` matches “banana” but not “bananas” or “bananananana”

Using `^` and `$` together is really useful for forcing exact matches on entire strings!

We now have all of the syntax needed to break down the first three examples!

```
apples?
```

```
^.*-.*$
```

```
\([0-9]{3}\)[0-9]{3}-[0-9]{4}
```

Building Block 5 – Capture Groups

Parentheses `()` are used to define capture groups. Capture groups can be used to group multiple tokens that must be matched together.

> `(wo)+w` matches “wow” and “wowowow”, <u>but not</u> “ooooooow”

Capture groups can contain character classes and quantifiers.

> `(\d+[\.:])+` matches both “10:” and “5.2:”

Capture groups have many other applications such as lookahead/behind and substitution, but this is outside the scope of this lecture.

Building Block 6 – Alternation

The pipe symbol `|` allows for one of multiple tokens or groups to match, kind of like logical OR. They can be used both in and outside of capture groups.

`(bye|hi){2}` matches “byebye” and “hihi” as well as “hibye”

grep and regex

grep’s default regex syntax is non-standard compared to a lot of other languages. When using grep to match regex, you should use the `-E` flag, which uses “extended” regex syntax. E.g.,

```
    $ grep -E “apples?” foods.txt
```

Practice using regex following the instructions in:

`/home/ckazer/public/cs14/inclass/regex_exercise.adoc`