Ch 02. Regular Expressions, Tokenization, Edit Distance

Hanseul Kim

January 2025

1 Summary

1.1 Regular Expression

1.1.1 Basic patterns

Regular expression is a text pre-processing tool for pattern matching.

- **Concatenation**: Matching characters sequence.
  ex) /woodchuncks/ → "interesting links to <u>woodchuncks</u> and lumers"
- **Disjunction**: Match using 'or' case.
  ex) / [wW]oodchuck / → "<u>Woodchuck</u> and <u>woodchuck</u>."
- **Range**: Matching range of items.
  ex) /[A-Z][a-z][1-9]/ → "Cs6"
- **Negation**: Matching except.
  ex) /[ˆ A-Z]/ → not an upper case letter.
- **The preceding character or nothing**: /?/
  ex) /colou?r/ → "<u>color</u> and <u>colour</u>."
- **Kleene ***: Zero or more occurrences of immediately previous character or regular expression.
  ex) /[0-9][0-9]*/ → "<u>1</u>, <u>100</u>"
- **Kleene +**: One or more occurrences of immediately previous character or regular expression.
  ex) /[0-9]$^+$/ → "<u>1</u>, <u>100</u>"
- **Wildcard**: Any single character.
  ex) /beg.n/ → "<u>begin</u>, <u>beg'n</u>, <u>begun</u>"
- **Anchors**: Start(ˆ), end of a line($), word boundary (\\$b$), and non-word boundary (\\$B$)

- **Start** (ˆ): The pattern ˆ `cat` matches "cat" only if it is at the beginning of the string (e.g., "catnap", not "napcat").
- **End ($)**: The pattern `cat$` matches "cat" only if it is at the end of the string (e.g., "napcat", not "catnap").
- **Word boundary** ($\backslash b$): The pattern $\backslash b$`cat`$\backslash b$ matches "cat" as a whole word (e.g., "a cat", but not "cater").
- **Non-word boundary** ($\backslash B$): The pattern $\backslash B$`cat`$\backslash B$ matches "cat" within a larger word (e.g., "cater", but not "a cat").

1.1.2 Disjunction, Grouping, and Precedence

- **Disjunction**: Matching 'or' operator using word level.
  ex) [cat | dog] → cat or dog
- **Precedence**: Using operator hierarchy.
  ex) /guppy(y | ies)/ → "guppy or guppies"

Some of the **operator precedence hierarchy** given from highest to lowest is following.

Operator name	Operator symbol
Parenthesis	()
Counters	$*$ + ? {}
Sequences and anchors	the ˆ my end$
Disjunction	|

Table 1: Operator precedence hierarchy

Some of frequently used regex.

Regex	**Expansion**	**Match Example**
`\d`	`[0-9]` any digit	`Party of 5`
`\D`	`[ˆ0-9]` any non-digit	`Blue moon`
`\w`	`[a-zA-Z0-9_]` any alphanumeric/underscore	`Daiyu`
`\W`	`[ˆ\w]` a non-alphanumeric	`!!!!`
`\s`	`[ \r\t\n\f]` whitespace (space, tab)	`in Concord`
`\S`	`[ˆ\s]` Non-whitespace	`in Concord`

Table 2: Common Regex Patterns and Their Matches

Regex	**Match Description**
`*`	Zero or more occurrences of the previous character or expression
`+`	One or more occurrences of the previous character or expression
`?`	Zero or one occurrence of the previous character or expression
`{n}`	Exactly `n` occurrences of the previous character or expression
`{n,m}`	From `n` to `m` occurrences of the previous character or expression
`{n,}`	At least `n` occurrences of the previous character or expression
`{,m}`	Up to `m` occurrences of the previous character or expression

Table 3: Regex Quantifiers and Their Descriptions

Regex	**Match Description**	**First Patterns Matched**
`\*`	An asterisk `"*"`	`"K*A*P*L*A*N"`
`\.`	A period `"."`	`"Dr.  Livingston, I presume"`
`\?`	A question mark `"?"`	`"Why don't they come and lend a hand?"`
`\n`	A newline	
`\t`	A tab	

Table 4: Some Characters That Need to Be Escaped (via Backslash).

1.1.3 ELIZA

Regular expression can be used to make simple natural language processing system such as **ELIZA**. Example is shown below.

User:	I feel sad.
Regex:	`I feel (.+)`
ELIZA:	Why do you feel `$1`?
User:	My mother doesn't care about me.
Regex:	`My (.+) doesn't care about me`
ELIZA:	Why do you think your `$1` doesn't care about you?
User:	I can't sleep.
Regex:	`I can't (.+)`
ELIZA:	What makes it difficult for you to `$1`?
User:	I need help.
Regex:	`I need (.+)`
ELIZA:	How will getting `$1` help you?

Explanation

The conversation is based on ELIZA's classic pattern-matching approach using regular expressions:

- ELIZA matches user input to predefined regex patterns.
- Placeholders like `$1` capture specific parts of the input for dynamic responses.
- For example, in `I feel (.+)`, `$1` captures the emotion or state the user mentions (e.g., "sad").

2 Exercises

2.1

Write regular expression for the following languages. a) the set of all alphabetic strings;

$$\hat{}\,[\text{a-zA-Z}]+$$

b) the set of all lower case alphabetic strings ending in a b;

$$\hat{}[\text{a-z}] * \text{b}\$$$

c) the set of all strings from the alphabet a,b such that a is immediately preceded by and immediately followed by a b;

$$\hat{}(ba)+b\$$$

2.2

Write regular expression for the following languages. By "word", we mean an alphabetic string separated from other words by whitespace, any relevant punctuation, line break and so forth.

a) the set of all strings with two consecutive repeated words (e.g., "Humbert Humbert" and "the the" bug not "the bug" or "the big bug")

$$/(\backslash\text{b}[\text{a-zA-Z}]+\backslash\text{b})(\backslash 1)/$$

b) all strings that start at the beginning of the line with an integer and that end at the end of the line with a word;

$$\hat{}[\text{0-9}]+\backslash\text{s}.*(\backslash\text{b}[\text{a-zA-Z}]+)\$$$

c) all strings that have both the word *grotto* and the word *raven* in them (but not, e.g., words like *grottos* that merely contain the word *grotto*);

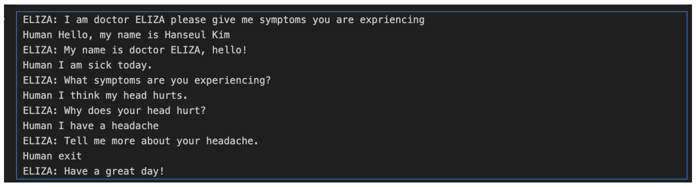
$$(.*(?=\text{grotto}).*(?=\text{raven})*.)$$

d) write a pattern that places the first word of fan English sentence in a register. Deal with punctuation.

$$\hat{}\,\backslash\text{s}*(\backslash\text{w}+)$$

2.3

Implement an ELIZA-like program, using substitutions such as those described on page 13. You might want to choose a different domain than Rogerian psychologist, although keep im ind that you would need a domain in which your program can legitmately engage in a lot of simple repetition.
Codes at here.



```
ELIZA: I am doctor ELIZA please give me symptoms you are expriencing
Human Hello, my name is Hanseul Kim
ELIZA: My name is doctor ELIZA, hello!
Human I am sick today.
ELIZA: What symptoms are you experiencing?
Human I think my head hurts.
ELIZA: Why does your head hurt?
Human I have a headache
ELIZA: Tell me more about your headache.
Human exit
ELIZA: Have a great day!
```

Figure 1: Doctor ELIZA

2.4

Compute the edit distance (using insertion cost 1, deletion cost 1, substitution cost 1) of ”leda” to ”deal”.

edit distance	#	d	e	a	l
#	0	1	2	3	4
l	1	1	2	3	3
e	2	2	1	2	3
d	3	2	2	2	3
a	4	3	3	2	3

Table 5: Edit distance grid

By reverse selection from 3, the minimal edit distance is 3.

$$\text{"leda" (substitution)} \rightarrow \text{"deda"}$$

$$\text{"deda" (substitution)} \rightarrow \text{"deaa"}$$

$$\text{"deaa" (substitution)} \rightarrow \text{"deal"}$$

2.5

Figure out whether *drive* is closer to *brief* or to *divers* and what the edit distance to each. You may use any version of distance that you like.
edit distance (using insertion cost 1, deletion cost 1, substitution cost 2)

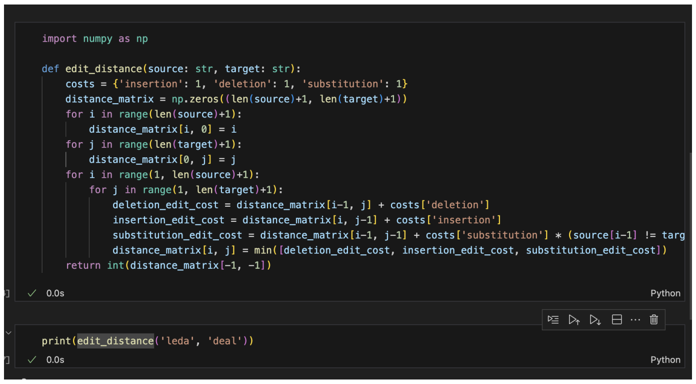
edit distance	#	b	r	i	e	f
#	0	1	2	3	4	5
d	1	2	3	4	5	6
r	2	3	2	5	6	7
i	3	4	3	2	3	4
v	4	5	4	3	4	5
e	5	6	5	4	3	4

Table 6: Edit distance grid

Edit distance between *drive* and *drivers* is just 1(insertion). Which is closer than *drive* and *brief* (4).

2.6

Now implement a minimum edit distance algorithm and use hand-computed results to check your code.
Codes at here.

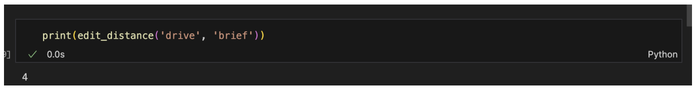


```python
import numpy as np

def edit_distance(source: str, target: str):
    costs = {'insertion': 1, 'deletion': 1, 'substitution': 1}
    distance_matrix = np.zeros((len(source)+1, len(target)+1))
    for i in range(len(source)+1):
        distance_matrix[i, 0] = i
    for j in range(len(target)+1):
        distance_matrix[0, j] = j
    for i in range(1, len(source)+1):
        for j in range(1, len(target)+1):
            deletion_edit_cost = distance_matrix[i-1, j] + costs['deletion']
            insertion_edit_cost = distance_matrix[i, j-1] + costs['insertion']
            substitution_edit_cost = distance_matrix[i-1, j-1] + costs['substitution'] * (source[i-1] != targ
            distance_matrix[i, j] = min([deletion_edit_cost, insertion_edit_cost, substitution_edit_cost])
    return int(distance_matrix[-1, -1])
```

```python
print(edit_distance('leda', 'deal'))
```

Figure 2: Edit distance from ’leda’ to ’deal’



```python
print(edit_distance('drive', 'brief'))
```

```
4
```

Figure 3: Edit distance between ’drive’ and ’brief’ using substitution cost: 2

2.7

Augment the minimum edit distance algorithm to output an alignment; ypu will need to store pointers and add a stage to compute the backtrace.

Codes at here.