CS418
Homework 01

Hoang Khoi
Student ID: 1351026

February 24, 2016

Contents

1 Question 1

Write regular expressions by using Perl notation for the following languages. By ‘word’, we mean an alphabetic string separated from other words by white space, any relevant punctuation, line breaks, etc.

1. the set of all alphabetic strings;

2. the set of all lower case alphabetic strings ending in a *b*;

3. the set of all strings with two consecutive repeated words (for example “Humbert Humbert” and “the the” but not “the bug” or “the big bug”);

4. the set of all strings from the alphabet *a,b* such that each *a* is immediately preceded by and immediately followed by a *b*;

5. all strings which start at the beginning of the line with an integer (i.e. 1,2,3...10...10000...) and which end at the end of the line with a word;

6. all strings which have both the word *grotto* and the word *raven* in them (but not, for example, words like *grottos* that merely contain the word *grotto*);

7. write a pattern that places the first word of an English sentence in a register. Deal with punctuation.

Your answer:

1. `[A-Za-z]+`

2. `[a-z]*b`

3. `(\b\w+\b)\s+\b\1\b`

4. `(b+(ab)+)+`

5. `^\d+.*[A-Za-z]+$`

6. `(.*grotto.*raven.*|.*raven.*grotto.*)`

7. `^[^A-Za-z]*([A-Za-z])+`

2 Question 2

Implement an ELIZA-like program, using substitutions such as those described in slide (01 Regular Expressions). You may choose a different domain than a Rogerian psychologist, if you wish, although keep in mind that you would need a domain in which your program can legitimately engage in a lot of simple repetition.

Your answer:

```
# Name/ID = Hoang Khoi/1351026

import string, re, random

SAD_PATTERN = r'.*\b(depressed|sad|not␣happy)\b.*'
ALL_PATTERN = r'.*\b(all|every)\b.*'
ALWAYS_PATTERN = r'.*\b(always|often)\b.*'
FINE_PATTERN = r'.*\b(good|nice|happy|ok|fine)\b.*'

NO_IDEAS = (
        'I␣have␣no␣ideas␣what␣you␣are␣talking␣about',
        'I␣am␣a␣bot,␣not␣a␣human!',
        'Say␣something␣easier␣to␣understand!')
RES_FINE = (
        'Nice␣to␣hear␣that,␣now,␣gimme␣a␣good␣grade␣please!',
        'OK,␣good␣to␣know␣that',
        'Sweet,␣now␣do␣something␣to␣damage␣your␣healthy␣life!'
)
RES_ALWAYS = (
        'What␣makes␣you␣say␣that?',
        r'You␣said␣always...␣Really?',
        'For␣real?'
)
RES_ALL = (
        'You␣are␣confident?␣aren\'t␣you?'
        'Prove␣what␣you␣say!'
        'I␣don\'t␣think␣so'
)
RES_SAD = (
        'Yeah,␣serve␣you␣right',
        'Awww!␣What␣\'s␣up?',
        'Huehuehuehuehue!!!'
)

user_str = ''    # user input

def is_sad():
        return re.match(SAD_PATTERN, user_str)
def is_all():
        return re.match(ALL_PATTERN, user_str)
def is_always():
        return re.match(ALWAYS_PATTERN, user_str)
def is_fine():
        return re.match(FINE_PATTERN, user_str)
print 'Welcome␣to␣ELIZA␣talk␣show!␣Press␣Ctrl-C␣(Linux)␣to␣exit␣:)'

exit_flag = False
while not exit_flag:
        user_str = raw_input('>user:␣')
        user_str = user_str.lower()
```

```
if is_sad():
        print '>ELIZA:␣',random.choice(RES_SAD)
elif is_all():
        print '>ELIZA:␣',random.choice(RES_ALL)
elif is_always():
        print '>ELIZA:␣',random.choice(RES_ALWAYS)
elif is_fine():
        print '>ELIZA:␣',random.choice(RES_FINE)
else:
        print '>ELIZA:␣',random.choice(NO_IDEAS)
```

3 Question 3

(Thanks to Pauline Welby; this problem probably requires the ability to knit.) Write a regular expression that matches all knitting patterns for scarves with the following specification: *32 stitches wide, K1P1 ribbing on both ends, stockinette stitch body, exactly two raised stripes.* All knitting patterns must include a cast-on row (to put the correct number of stitches on the needle) and a bind-off row (to end the pattern and prevent unraveling). Heres a sample pattern for one possible scarf matching the above description [1] :

1. Cast on 32 stitches.	*cast on; puts stitches on needle*
2. K1 P1 across row (i.e., do (K1 P1) 16 times).	*K1P1 ribbing*
3. Repeat instruction 2 seven more times.	*adds length*
4. K32, P32.	*stockinette stitch*
5. Repeat instruction 4 an additional 13 times.	*adds length*
6. P32, P32.	*raised stripe stitch*
7. K32, P32.	*stockinette stitch*
8. Repeat instruction 7 an additional 251 times.	*adds length*
9. P32, P32.	*raised stripe stitch*
10. K32, P32.	*stockinette stitch*
11. Repeat instruction 10 an additional 13 times.	*adds length*
12. K1 P1 across row.	*K1P1 ribbing*
13. Repeat instruction 12 an additional 7 times.	*adds length*
14. Bind off 32 stitches.	*binds off row: ends pattern*

Your answer:

C{32}
((KP){16})+
(K{32}P{32})+
P{32}P{32}
(K{32}P{32})+
P{32}P{32}
(K{32}P{32})+

[1] *Knit* and *purl* are two different types of stitches. The notation Kn means do n knit stitches. Similarly for purl stitches. Ribbing has a striped texture — most sweaters have ribbing at the sleeves, bottom, and neck. Stockinette stitch is a series of knit and purl rows that produces a plain pattern — socks or stockings are knit with this basic pattern, hence the name.

((KP){16})+
B{32}

4 Question 4

Computing minimum edit distances by hand, figure out whether *drive* is closer to *brief* or to *divers* and what the edit distance is. You use 1–insertion, 1–deletion, 2–substitution costs.

Your answer:

- Distance between *drive* and *brief* is: 4
- Distance between *drive* and *divers* is: 2
- Thus, *drive* is closer to: *drivers*

5 Question 5

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

Your answer:

```
def med(str0, str1):
        len0 = len(str0)
        len1 = len(str1)

        result = [[0 for x in range(len1 + 1)] for x in range(len0 + 1)]
        for i in range(0, len0 + 1):
                for j in range(0, len1 + 1):
                        if i == 0:
                                result[i][j] = j
                        elif j == 0:
                                result[i][j] = i
                        else:
                                choice0 = result[i - 1][j] + 1
                                choice1 = result[i][j - 1] + 1
                                choice2 = result[i - 1][j - 1]
                                if str0[i - 1] != str1[j - 1]:
                                        choice2 += 2

                                result[i][j] = min(choice0, min(choice1, choice

        # Print out the matrix for testing:
#       for i in range(0, len0 + 1):
#               for j in range(0, len1 + 1):
#                       print result[i][j], ' ',
#               print
```

```
        return result[len0][len1]

str0 = raw_input('Input string0: ')
str1 = raw_input('Input string1: ')
print 'MED = ', med(str0, str1)
```

6 Question 6

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

Your answer:

```
import sys

def med(str0, str1):
        len0 = len(str0)
        len1 = len(str1)

        result = [[0 for x in range(len1 + 1)] for x in range(len0 + 1)]
        # Down, Diag, Right, DiagKeep
        trace = [[[False, False, False, False] for x in range(len1 + 1)] for x
        action = ['I', 'S', 'D', 'K']

        for i in range(0, len0 + 1):
                for j in range(0, len1 + 1):
                        if i == 0:
                                result[i][j] = j
                                trace[i][j][2] = True
                        elif j == 0:
                                result[i][j] = i
                                trace[i][j][0] = True
                        else:
                                choice0 = result[i - 1][j] + 1
                                choice1 = result[i][j - 1] + 1
                                choice2 = result[i - 1][j - 1]

                                sub_flag = False

                                if str0[i - 1] != str1[j - 1]:
                                        choice2 += 2
                                        sub_flag = True

                                min_value = min(choice0, min(choice1, choice2))

                                if min_value == choice0:
                                        trace[i][j][0] = True
                                if min_value == choice1:
```

```
                    trace[i][j][2] = True
                if min_value == choice2:
                    if sub_flag:
                        trace[i][j][1] = True
                    else:
                        trace[i][j][3] = True

                result[i][j] = min_value

    # Traceback
    i = len0
    j = len1

    stack_trace = []

    while not (i == 0 and j == 0):

        cur_trace = trace[i][j]

        if cur_trace[0]: down = result[i - 1][j]
        else: down = sys.maxint
        if cur_trace[1] or cur_trace[3]: diag = result[i - 1][j - 1]
        else: diag = sys.maxint
        if cur_trace[2]: right = result[i][j - 1]
        else:right = sys.maxint

        if diag <= right and diag <= down:
            if cur_trace[1]:
                stack_trace.append(1)
            else:
                stack_trace.append(3)
            i -= 1
            j -= 1
        elif right <= down and right <= diag:
            stack_trace.append(2)
            j -= 1
        elif down <= right and down <= diag:
            stack_trace.append(0)
            i -= 1
    print 'MED␣=␣', result[len0][len1]
    print 'Alignment␣instructions:␣',
    for i in stack_trace:
        print action[i],

print 'NOTE:␣This␣is␣the␣instruction␣to␣transform␣str0␣into␣str1'
print 'NOTE:␣the␣instruction␣is␣the␣TRACE␣STACK,␣so␣read␣it␣backward!'
print 'NOTE:␣K:␣Keep,␣D:␣Delete,␣I:␣Insert,␣S:␣Subtitute'
str0 = raw_input('str0␣=␣')
str1 = raw_input('str1␣=␣')
med(str0, str1)
```