Ling 473 Project 3
Due 11:59pm on Thursday, August 26, 2021

For this project you will implement a finite state automaton (FSA) that **identifies syllables in Thai text**. Thai uses a phonetic alphabet of consonants, vowels, and tone marks, but does not use space to separate words or syllables. Therefore, a fundamental problem in text processing for Thai—and many other East Asian languages—is to automatically identify syllable and word boundaries in text.

For this project, we will consider a simplified syllable-breaker for Thai which allows for a (mostly) greedy, deterministic FSA and use it to process **selected** Thai text. Note that, a real syllable-breaker for Thai is much more complicated than the model used in this project. The FSA used in this project is a simplified model which can work on the selected data.

In this project, the correct FSA is given, and you need to implement the given FSA and use it to process Thai text.

Syllables structure in Thai and the Correct FSA

In this project, a syllable in Thai text has the following structure, where square brackets show optional elements.

$$[V_1]\ C_1\ [C_2]\ [V_2]\ [T]\ [V_3]\ [C_3]$$

Category		Members	Unicode (UTF-8)
V_1	Preposed vowel	เ แ โ ใ ไ	0E40, 0E41, 0E42, 0E43, 0E44
C_1	Initial consonant	ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ฤ ล ฦ ว ศ ษ ส ห ฬ อ ฮ	0E01 - 0E2E (inclusive)
C_2	Clustered consonant	ร ล ว น ม	0E19, 0E21, 0E23, 0E25, 0E27
V_2	Super- or subscript vowel	◌ิ ◌ี ◌ึ ◌ื ◌ุ ◌ู ◌ั ◌็	0E31, 0E34, 0E35, 0E36, 0E37, 0E38, 0E39, 0E47
T	Superscript tone mark	◌่ ◌้ ◌๊ ◌๋	0E48, 0E49, 0E4A, 0E4B
V_3	Postposed vowel or glide	า อ ย ว	0E22, 0E27, 0E2D, 0E32
C_3	Final consonant	ง น ม ด บ ก ย ว	0E01, 0E07, 0E14, 0E19, 0E1A, 0E21, 0E22, 0E27

Additional constraints are encoded in the FSA state transitions, below.

States and transitions of the finite state automaton (FSA)

State #	Action	Transition
0	Accept V1/C1	V1 → 1 C1 → 2
1	Accept C1	C1 → 2
2	Accept C2/V2/T/V3/C3/V1/C1	C2 → 3 V2 → 4 T → 5 V3 → 6 C3 → 9

		V1 → 7 C1 → 8
3	Accept V2/T/V3/C3	V2 → 4 T → 5 V3 → 6 C3 → 9
4	Accept T/V3/C3/V1/C1	T → 5 V3 → 6 C3 → 9 V1 → 7 C1 → 8
5	Accept V3/C3/V1/C1	V3 → 6 C3 → 9 V1 → 7 C1 → 8
6	Accept C3/V1/C1	C3 → 9 V1 → 7 C1 → 8
7	Break before current character	→ 1
8	Break before current character	→ 2
9	Break now	→ 0

```
Start state: 0
Accept (final) states: 9
```

In this project, **you will implement the above FSA and use it to process Thai text.** If there is no valid transition out of a state based on the current character, then the input would be rejected. However, *the test data for this project has been selected so that a correct FSA will not reject any lines*, and will correctly identify all Thai syllables.

In the FSA, states 7, 8, and 9 are the cases where a syllable break has been detected. For states 7 and 8, the character that triggered the state actually belongs to the *next* syllable, so you will need to emit a space *before* the current character that caused the transition to state 7 and 8. The easier case is for state 9, where the character that triggered the state was the last character of a syllable. Here, you will simply emit a space after it.

Because some characters appear in more than one category, the order in which the current character is matched within each state is important. For example, for the state "Accept V3/C3/V1/C1," you will take the first category that matches, reading from left-to-right.

Hint: the pseudo code of this process is given at the end of this document. You can refer to it if you don't know where to start.

Input

In this project, you will have two input files, namely, "***category.txt***" and "***input.txt***". You can find the two files under "**/dropbox/20-21/473/project3/**" on Patas. **The encoding of both files is UTF-8.** You need to specify the encoding of the two files when reading them. Otherwise, you may read incorrect content. For example, in python, you can use the following code to read a file with UTF-8 encoding:

```
with open(file, 'r', encoding='utf8') as f:
        lines = f.readlines()
```

The following is the detailed illustration of the two input files.

category.txt

This file contains 7 lines of texts, where each line is a category followed by the characters that belong to it. If you open the file with a text editor (on your local machine) with UTF-8 encoding, you might see the following.

V1 เ แ โ ใ ไ
C1 ก ข ฃ ค ฅ ฆ ง จ ฉ ช ซ ฌ ญ ฎ ฏ ฐ ฑ ฒ ณ ด ต ถ ท ธ น บ ป ผ ฝ พ ฟ ภ ม ย ร ฤ ล ฦ ว ศ ษ ส ห ฬ อ ฮ
C2 ร ล ว น ม
V2 ิ ี ึ ื ุ ู ั ็
T ่ ้ ๊ ๋
V3 า อ ย ว
C3 ง น ม ด บ ก ย ว

In the 7 lines, elements are segmented by a single space. The first element of each line is the category name and the following elements are the characters that belong to the category. For example, in the first line, “V1” is the category name and that category has 5 characters, namely, “เ” “แ” “โ” “ใ” “ไ”.

You may want to read this file (with UTF-8 encoding) and split each line by space, which will give you a list of elements, where the first element is the category and the followings are the member characters. Then use a hash table (e.g., the dictionary in python) to store the category and its members (i.e., characters), where the categories are keys and the lists of their members are values.

For example, the hash table may look like:

cat2chars = {'V1': ['เ', 'แ', 'โ', 'ใ', 'ไ'], ‘C1’: […], ‘C2’: […], …}

input.txt

This file contains 27 lines of selected Thai text to be processed. If you open the file with a text editor (on your local machine) with UTF-8 encoding, you might see the following.

คู่แข่งขันต่างก็คุมเชิงกัน
เขาเงียบไปครู่หนึ่งแล้วพูดขึ้น
นกเล็กมีหน้าอกสีแดง
ขาของเขาสั่นเทาเมื่อต้องลงมาจากที่สูง
หนึ่งหมื่นสองพันห้าร้อยหกสิบสาม
สองหมื่นสี่พันหนึ่งร้อยห้าสิบสอง
……

In the code, you may want to read the file with UTF-8 encoding and process the text line by line.

Running the code and the output format

The program should accept exactly **three argument**s where

- the **first** is the absolute path to the **category** file (i.e., /dropbox/20-21/473/project3/category.txt) that contains the categories and the members;
- the **second** is the absolute path to the **input** file (i.e., /dropbox/20-21/473/project3/input.txt) that contains 27 lines of text to be processed;
- the **third** is the path to the output file. Note that, in this project, **your program needs to write the segmented Thai text into a file**, rather than printing it to the standard output. This will allow us to specify the encoding of the output file. When you write the results into a file, use UTF-8 encoding. For example, in python, you can specify the encoding by

```
with open(file, 'w', encoding='utf8') as f:
    f.write(string)
```

The output file should look like the following, where each line corresponds to a line in the input That text following the same order.

คู่ แข่ง ขัน ต่าง ก็ คุม เชิง กัน
เขา เงียบ ไป ครู่ หนึ่ง แล้ว พูด ขึ้น
นก เล็ก มี หน้า อก สี แดง
ขา ของ เขา สั่น เทา เมื่อ ต้อง ลง มา จาก ที่ สูง
หนึ่ง หมื่น สอง พัน ห้า ร้อย หก สิบ สาม
สอง หมื่น สี่ พัน หนึ่ง ร้อย ห้า สิบ สอง
……

Run your code on Patas with the following command line:

./run.sh /dropbox/20-21/473/project3/category.txt /dropbox/20-21/473/project3/input.txt ./output

The first 6 lines in the output file should be the same as the above figure. You can download the output file and open it on your local machine. If you open it on Patas, you may not be able to see the correct content due to the encoding issues.

Submission

hw.tar.gz

Include the following files in your submission:

`run.sh`	The command(s) that run your program. Your code should take three arguments. The first is the category file (i.e., **category.txt** illustrated above) which contains the categories and the members; the second is the input Thai file (i.e., **input.txt** illustrated above); the third is the output file with segmented Thai text.
`output`	The output file which is produced by **./run.sh /dropbox/20-21/473/project3/category.txt /dropbox/20-21/473/project3/input.txt ./output**
`(source code and binary files)`	All source code and binary files (jar, a.out, etc., if any) required to run and compile your program

Similar to project 1 and project 2, gather together all the required files and use the following command to package your files for submission.

```
tar -czf hw.tar.gz ./*
```

Notice that this command packages all files in the current directory; do not include any top-level directories.

Check whether your submission contains all required file using

`/dropbox/20-21/473/project3/check_project3.sh`

`readme.{pdf, txt}`

Your write-up of the project. Describe your approach, any problems or special features, or anything else you'd like me to review. If you could not complete some or all of the project's goals, please explain what you were able to complete.

Submit **hw.tar.gz** and **readme.{pdf, txt}** to Canvas.

The pseudo code is on the **NEXT** page.

The pseudo code of processing a line with the given FSA:

```
# define the FSA
fsa = {
        0: [('V1', 1), ('C1', 2)],
        1: [('C1', 2)],
        2: [('C2', 3), ('V2', 4), ('T', 5), ('V3', 6), ('C3', 9)],
        …
        7: 1,
        8: 2,
        9: 0
}

# get the categories and their members.
# cat2chars is a dictionary that can map the category to a list of its members
cat2chars = get_cat2chars(category_file)

# define a varible to store the segmented text
segmented = ''

current_state_index = 0
# the process to segment one line
for each char in line:
        for each (category, to_state_index) in fsa[current_state_index]:
                if char in cat2chars[category]:
                        # set the current state index to the to-state index
                        current_state_index = to_state_index
                        # we find the first category that contains the current character
                        end the for-loop

        if current_state_index == 7 or current_state_index == 8:
                # we need to break the syllable before current character
                segmented = segmented + ' ' + char
                # move to the next state for free
                current_state_index = fsa[current_state_index]
        else if current_state_index == 9:
                # we need to break now
                segmented = segmented + char + ' '
                # move to the next state for free
                current_state_index = fsa[current_state_index]
        else:
                # in other cases, we don't add space to the segmented text
                segmented = segmented + char

write segmented to the output file
```