Homework #2: CMPT-413

Due in class on Feb 4, 2003

Anoop Sarkar – `anoop@cs.sfu.ca`

(1) In some languages, the written or textual script does not have whitespace characters between the words. The task of taking an input sentence and inserting legitimate word boundaries is called *word segmentation*. In this question, we will use finite state transducers (FSTs) for word segmentation in Chinese.

The lexicon we will use to construct Chinese sentences is given in Table 1. The finite state machine in Figure 1 is an extremely simple grammar for sentences in Chinese using this lexicon.

a. (20pts) Extending Figure 1, construct a FST T_{1a} that will insert a space after each word (map the empty string ϵ in the input after each word to a space character in the output). Denote the space character as ␣ in your FST.

b. (10pts) Use the following sentence as an input to T_{1a} and produce at least two different output sentences (that is: two sentences with different word segmentations): *wo3wang4bu4liao3jie3fang4jie1zai4na3li3*

c. (15pts) Construct a FST T_{1c} that takes as input the output of T_{1a} and produces an output English translation based on the English entries in Table 1. Write down the output of T_{1c} for the two output sentences from question 1b.

d. (5pts) Provide one output from FST T_{1c} that is closest in meaning to the following English sentence: *I was unable to forget where Liberation Avenue is.*

Chinese word (pinyin)	English translation
da4	big
da4jie1	avenue
wo3	I
fang4	place
jie1	avenue
jie3fang4	liberation
bu4	not
liao3jie3	understand
wang4	forget
wang4bu4liao3	unable to forget
fang4da4	enlarge
na3li3	where
zai4	at

Table 1: Lexicon for Question 1.

(2) Perl programming warm-up exercises: For this question, use the text file called `cmpt413-hw2-2.txt` as the input to each of the sub-parts to this question. This file is available from the course web page (look under the *Assignments* section). Each program should be run from the command line as follows:

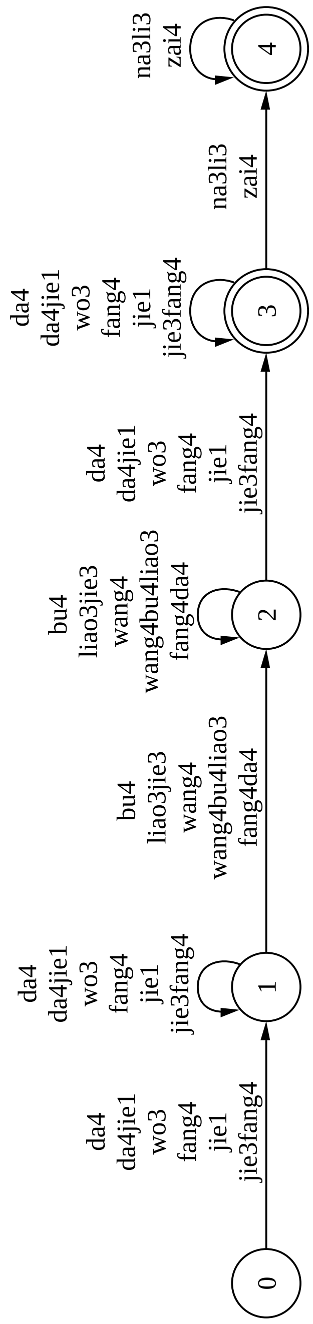

Figure 1: Finite-state machine for Question 1.

`% perl your-program.pl cmpt413-hw2-2.txt` ⟨*cr*⟩

For each question print out your Perl source code as well as the output of your program.

a. (5pts) Write a Perl program to read in the text file and print each line to standard output with the line number of each line as a prefix. The line numbers should start from 1. e.g. it would take each line of text from the input:

```
some text here
some more text here
```

and print each line with it's line number:

```
1 some text here
2 some more text here
```

Also print out the number of lines to `STDERR`. e.g. if the file had ten lines then print to `STDERR`: `number of lines=10`

b. (5pts) Extend the program for Q.2a to print only the odd-numbered lines.

c. (5pts) Write a Perl program that takes the text file and strips away any initial positive integer followed by a space if it exists, e.g. it would take a line of text:

```
1 some text here
2 some more text here
```

and print

```
some text here
some more text here
```

d. (10pts) Write a Perl program to randomly shuffle the lines in the text file. Use the following function `fisher_yates_shuffle` which randomly shuffles an array in place:

```
sub fisher_yates_shuffle {
    my ($array) = @_;
    for (my $i = @$array; --$i; ) {
        my $j = int(rand($i+1));
        next if ($i == $j);
        @$array[$i,$j] = @$array[$j,$i];
    }
}
```

e. (10pts) Use the function `fisher_yates_shuffle`, but this time use it in a Perl program to randomly shuffle the words in each input line. Each line of text should appear in the same order as the input but the words in each line should be randomly permuted.

f. (15pts) Write a Perl program to eliminate duplicate lines from the input file. Provide the number of lines that remain after all duplicate lines are eliminated from `cmpt413-hw2-2.txt`.

(3) Text processing using Perl: For all sub-parts of this question, use the text file called `cmpt413-hw2-3.txt` available from the course web page. Each program should be run from the command line as follows:
`% perl your-program.pl cmpt413-hw2-3.txt` ⟨*cr*⟩
For this question, provide the source code but **not** the output of the program on the input file.

a. (10pts) Each word in the input file is followed by an underscore and then a part of speech tag, e.g. `plants_NN`. Write a Perl program to remove the underscore and part of speech tag after each word.

b. (10pts) Write a Perl program to print out only the lines which contain the word ␣*plants* (e.g. it should not match *plant* or *implants* or even *plantocracy*), where ␣ is a whitespace character. Provide the number of lines printed out by your program. The underscore and part of speech does not play any role here.

c. (15pts) Write a Perl program that takes the input file and prints out those lines which contain the word *plant* with any kind of suffix (e.g. it should match ␣*plant* and ␣*plants* but not *implants*) **and** any word containing the string *flower* (e.g. it should match *flowers* and *wildflowers*). Similarly, print out the lines matching the word *plant* with any suffix **and** any word containing the string *manufacturing*. The underscore and part of speech does not play any role here.

   Notice that the two subsets of the original text provided by your Perl program correspond in a very crude way to two different meanings of the word *plant*, one meaning is synonymous to *plant-life* and the other to *factory* .

d. (15pts) Each word in the input file is followed by an underscore and a tag that represents the word's part of speech. Write a Perl program that finds for each word starting with ␣*plant* (i.e. the word *plant* with any suffix) how many times the word was observed with each part of speech tag. For the text file given the output is:

```
      plant   NN  2310
 plantation   NN     2
    planted  VBN     4
plantocracy   NN     1
     plants  NNS    45
```

(4) Download the Perl program `fsm-nogen.pl` from the course web page. This file contains an implementation of the algorithm `NDRecognize` from Jurafsky and Martin (see Figure 2.21 on page 44) minus the function `generateNewStates`. You have to add that function.

The program takes as input finite-state machine (FSM) described as the following Perl data (available as the file `fsm-ex1.pl` from the course web page):

```
$startstate = 1;
$finalstate[3] = 'true';
$edges[1] = [2];
$edges[2] = [2,3];
$input[1][2] = ['a'];
$input[2][2] = ['a','b'];
$input[2][3] = ['b'];
```

This FSM corresponds to the usual graphical representation shown in Figure 2. Note that the `transition-table` data structure used in the Jurafsky and Martin textbook has been replaced here with two arrays: `@edges` provides a list of possible transitions from a state, and `@input` provides the possible inputs to match from the tape on each transition from one state to another.

a. (50pts) Copy `fsm-nogen.pl` to a new file `fsm.pl` and add the function `generateNewStates` based on the description given in the textbook. Once you add this function you should be able to run the following command:

   `% perl fsm.pl fsm-ex1.pl` ⟨*cr*⟩

   It takes as input any string from it's standard input. Enter a string and then press ⟨*cr*⟩ and then `ctrl-D`. If the string is accepted the algorithm will display a trace of its execution and say `yes` if the string is accepted by the FSM or `no` otherwise. Given below is a sample trace of the execution of `fsm.pl` (with the missing function `generateNewStates`) for the input string `abab`.

```
% echo "abab" | perl fsm.pl fsm-ex1.pl
```

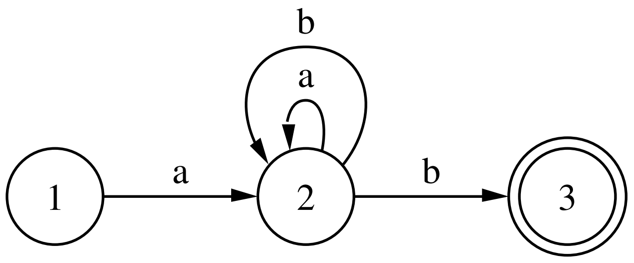

Figure 2: Finite-state machine for Question 4.

```
currentItem=1, index=-1 tape=abab
agenda=
currentItem=2, index=0 tape=bab
agenda=
currentItem=2, index=1 tape=ab
agenda=3 1
currentItem=3, index=1 tape=ab
agenda=2 2
currentItem=2, index=2 tape=b
agenda=
currentItem=2, index=3 tape=
agenda=3 3
currentItem=3, index=3 tape=
accept state=3
yes
```

b. (25pts) *Optional*: For extra credit, modify `fsm.pl` so that it can handle transitions on multi-character strings. Use the FSM in `fsm-ex2.pl` to test your implementation.

(5) (75pts) Download the perl program `edit-distance.pl` from the course web page. It implements a function `minEditDistance` which computes the minimum edit distance between two strings. Here are two examples of the output produced by `edit-distance.pl`:

```
% perl edit-distance.pl gamble gumbo
levenshtein distance = 5

% perl edit-distance.pl "recognize speech" "wreck a nice beach"
levenshtein distance = 14
```

Copy `edit-distance.pl` to a new file `edit-distance-align.pl` and then extend `edit-distance-align.pl` by modifying the function `minEditDistance` so that you can memorize or trace back your steps from the final score for the minimum distance alignment (for each substring record whether it was an insertion, deletion or substitution that provided the best alignment). Then, pass this information to a single new function `printAlignment` which produces the following visual display of the best (minimum distance) alignment:

```
% perl edit-distance-align.pl gamble gumbo
levenshtein distance = 5
g a m b l e
|   | |
g u m b _ o

% perl edit-distance-align.pl "recognize speech" "wreck a nice beach"
levenshtein distance = 14
_ r e c _ _ o g n i z e   s p e e c h
  | | |         | |   | |     |   | |
w r e c k   a   n i c e   _ b e a c h

% perl edit-distance-align.pl execution intention
levenshtein distance = 8
_ e x e c u t i o n
      |     | | | |
i n t e _ n t i o n
```

The 1st line of the visual display shows the *target* word or phrase and the 3rd line shows the *source* word or phrase. An insertion is represented as an underscore '_' in the 1st line with the letter being inserted in the 3rd line (aligned with the underscore char). A deletion is represented as an underscore in the 3rd line aligned with the deleted letter in the 1st line. Finally, if a letter is unchanged between target and source then a vertical bar (the pipe symbol '|') is printed aligned with the letter in the 2nd line.

Apart from the use of references (used to pass multiple arrays to a function in `edit-distance.pl`), this program does not require any advanced knowledge of Perl. There is a hint on how to proceed in the source for `edit-distance.pl`.