The University of Melbourne
School of Computing and Information Systems
COMP10002 Foundations of Algorithms
Semester 1, 2021
Assignment 2
Due: 4pm Tuesday 25th May 2021

Version 1.0

1 Learning Outcomes

In this assignment you will demonstrate your understanding of dynamic memory allocation, linked data structures, and search algorithms. You will further extend your skills in program design and implementation.

2 The Story...

Text analysis and understanding is becoming prevalent in our daily lives. For example, intelligent personal assistant apps such as Apple Siri and Google Now process user requests by first converting voice to text with a *speech recognition* algorithm, and then analysing the text and finding answers to the requests. Finally, the answers found are read out by a *text to speech* (TTS) algorithm.

To help computers understand a sentence, there are a few standard preprocessing steps. Two preprocessing steps of interest in this assignment are called *stemming* and *Part-Of-Speech* (POS) *tagging*.

Stemming reduces a word to its stem (that is, the root form). See the following example:

```
Sentence:         After stemming:
she               she
sells             sell
seashells         seashell
```

Here, "`sells`" is the third-person present form of "`sell`"; "`seashells`" is the plural form of "`seashell`". After stemming, the two words are reduced to their respective root forms. The word "`she`" is already in its root form, and hence is unchanged.

POS tagging assigns parts of speech to each word, such as noun, verb, etc. See the following example:

```
Sentence:          POS tag
she                pronoun
sells              verb
seashells          noun
```

There are various algorithms for stemming and POS tagging. The core of those algorithms are linguistic rules (such as "remove the ending '`s`' of a plural noun to obtain its singular form") and statistics (such as how often "`cook`" is used as a verb instead of a noun). Dictionaries are also used to support those algorithms.

In this assignment, you will implement an algorithm for stemming and POS tagging using a dictionary. *Note that you do not need to have any knowledge in linguistics to complete this assignment.*

3 Your Task

You are given a dictionary (with **at least one and up to 100 unique words**) and a sentence to be processed in the following format.

```
$allocate
vt
#0allocated1allocated2allocating
$sell
vt vi n
#0sold1sold2selling3sells
$she
n pron
#3they
$the
art
#
$zebra
n
#3zebra
**********
she sells seashells
```

The input starts with a list of unique dictionary words sorted alphabetically. Each word takes three lines. Line 1 starts with '`$`', which indicates the start of a word and is followed by the word itself (e.g., "`sell`"). There are **up to 22** lower-case English letters in each word, with no upper-case letters or special characters.

Line 2 contains the possible POS tags of the word separated by space (e.g., "`vt`", "`vi`", and "`n`" represent verb used with object, verb used without object, and noun, respectively). You may assume **up to 5** POS tags per word and **up to 4** lower-case English letters per POS tag. (*Hint: You may use either a single string or an array of strings to store the POS tags – the former is simpler.*)

Line 3 starts with '`#`', which indicates the ending line of a word in the dictionary. It is followed by the *variation forms* of the word. Each word can have **up to 4 forms** where each variation form may contain **up to 25** lower-case English letters:

1. Form '`0`' is the past tense form (e.g., "`sold`");
2. Form '`1`' is the past participle form (e.g, "`sold`");
3. Form '`2`' is the present participle form (e.g., "`selling`");
4. Form '`3`' is the plural form (e.g., "`sells`", note this is *not* the third-person present form).

The variation forms follow the rules below:

- A word may have (1) no variation forms, (2) the first three forms (such as a verb), (3) only the last form (such as a noun), or (4) all four forms (such as a word that can be either a verb or a noun). You do *not* need to check the POS tags to verify the variation forms that a word has.
- The forms of a word always appears in the ascending order, e.g., "`$0sold1sold2selling3sells`", not "`$3sells0sold2selling1sold`".
- The variation forms of a word can be the same (e.g., Forms 0 and 1 of "`sell`" are both "`sold`").
- A variation form of a word can be the word itself (e.g., Form 3 of "`zebra`" is still "`zebra`").
- You may assume that any two words in the dictionary will **not** share the same variation forms (e.g., the variation forms of "`allocate`" and "`sell`" are all different). You may also assume that any word will **not** be a variation form of another word (e.g., "`allocate`" is not a variation form of any other word in the dictionary).

(*Hint: You do not need to understand what these variation forms mean for the assignment. You may simply use a single string to store all variation forms of a word. For Stage 4, optionally, you can separate the different variation forms and store them in multiple strings, which will help make the search process faster.)*

The line "**********" indicates the end of the dictionary and the start of a given sentence.

The sentence given will occupy one line, where the words are separated by a single space. The sentence will contain at least one word, but **there is no upper limit on the number of words or the number of characters in the sentence**. Each word may contain **up to 25** lower-case English letters. **There will be no punctuation marks (e.g., ',' or '.')**.

You will perform stemming and POS tagging on the sentence by consulting the given dictionary.

3.1 Stage 1 - Read One Dictionary Word (Up to 3 Marks)

Your first task is to design a `struct` to represent one word. You will then read in one word from the input data (using standard input functions with input redirection; no file input operations needed), and output it in the following format.

```
==========Stage 1==========      /* 26 '='s each side */
Word 0: allocate
POS: vt
Form: 0allocated1allocated2allocating
```

3.2 Stage 2 - Read the Whole Dictionary (Up to 7 Marks)

Next, continue to finish reading the whole dictionary. You need to design a proper data structure to store the words read in. An array of `struct`'s will do the job nicely. When this stage is done, your program should output: the total number of words in the dictionary and the average number of variation forms per word (up to two digits after the decimal point, by using "`%.2f`" in the `printf()` function). The output of this stage based on the sample input is as follows.

```
==========Stage 2==========
Number of words: 5
Average number of variation forms per word: 1.80
```

3.3 Stage 3 - POS-Tag the Words in Sentence (Up to 13 Marks)

Your third task is to read in the given sentence, break it into words, store the words in a *linked data structure*, and output the words together with their POS tags one at a line. The output of your program in this stage for the example above should be:

```
==========Stage 3==========
she                       n pron
sells                     NOT_FOUND
seashells                 NOT_FOUND
```

Hint: You may used the `getword()` *function (Figure 7.13 in the textbook, link to source code available in lecture slides) to read the words in the sentence. You may modify the linked list in* `listops.c` *(link to source code available in lecture slides) to store the words.* You are free to use other linked data structures (e.g., queue, stack, etc) if you wish. Make sure to attribute the use of external code properly.

For POS tagging, you will use the binary search algorithm (*Hint: You may use the* `binary_search()` *function, link to source code available in lecture slides, or* `bsearch()` *provided in* `stdlib.h`) to look up each word from the sentence in the dictionary. If the word is found, then output the word and *all* POS tags of the word. If the word is not found, then output the word and "`NOT_FOUND`".

In this sample output, "`sells`" has not been matched because we are not considering the root forms of the words in this stage. In real systems, stemming should happen before POS tagging. This is because, without stemming, a word (e.g., "`sells`") that is not in its root form may not be found in the dictionary. In this assignment, we perform POS tagging first because it is a simpler task.

If you are not confident with using linked data structures, you may use an array of strings to store the words, assuming a maximum of 20 words in the sentence. However, if you do so, the full mark of this stage will be reduced by 2 marks. Similarly, if you use sequential search for POS tagging instead of binary search, the full mark of this stage will also be reduced by 2 marks.

3.4 Stage 4 - Stem the Words in Sentence (Up to 15 Marks)

The final task is stemming, that is, for each word in the given sentence, you need to look up its root form from the dictionary, and output the root form. If a word is already in its root form, or the word cannot be found in the dictionary, then simply output the word itself as the root form. Further, you need to perform POS tagging on this root form. The output of your program in this stage for the example above should be as follows (note a final newline character '`\n`' at the end of the output).

```
=========================Stage 4=========================
she                       she                       n pron
sells                     sell                      vt vi n
seashells                 seashells                 NOT_FOUND
```

At the end of your submission file, you need to add a comment that states the worst-case time complexity of your root form search algorithm, and explains why it has that time complexity, assuming that there are `s` words in the sentence and `d` words in the dictionary, each dictionary word has up to `f` variation forms, and each form has a maximum length of `m` characters.

Hint: A simple approach is that, for every word in the sentence, apply sequential search on the dictionary words and variation forms to look up the word. This is allowed for Stage 4 and will *not* incur mark penalties.

As a challenge (no bonus marks), see if you can use binary search instead of sequential search to find the root form, for better efficiency. Note: if you use binary search in both Stages 3 and 4, you should *not write more than one* binary search function. You should use function pointers to customise the behaviour of the binary search function for different use cases.

4 Submission and Assessment

This assignment is worth 15% of the final mark. A detailed marking scheme will be provided on Canvas.

You need to submit your code in Grok Learning (`https://groklearning.com`) for assessment. Submission will NOT be done via Canvas. Instead, you will need to:

1. Log in to Grok Learning using your student login details.
2. Navigate to the Assignment 2 module of our subject COMP10002 2021 S1: Foundations of Algorithms.
3. Place *all* your code in the `program.c` tab window (including any function from `listops.c` used).
4. Compile your code by clicking on the `Compile` button.
5. Once the compilation is successful, click on the `Mark` button to submit your code. (You can submit as many times as you want to. *Only the last submission made before the deadline will be marked.* Submissions made after the deadline will be marked with late penalties as detailed at the end of this document. Do *not* press the `Mark` button after the deadline unless a late submission is intended.)
6. Two sample tests will be run automatically after you make a submission. Make sure that your submission passes these sample tests.
7. Two hidden tests will be run for marking purpose. Results of these tests will not be available until after the submissions are marked.

You can (and should) submit both **early and often** – to check that your program compiles correctly on our test system, which may have some different characteristics to the lab machines and your own machines.

You will be given a sample test file `test0.txt` and the sample output `test0-output.txt`. You can test your code on your own machine with the following command and compare the output with `test0-output.txt`:

```
mac: ./program < test0.txt    /* Here '<' feeds the data from test0.txt into program */
```

Note that we are using the following command to compile your code on the submission system (we name the source code file `program.c`).

```
gcc -Wall -std=c99 -o program program.c
```

The flag "`-std=c99`" enables the compiler to use a more modern standard of the `C` language – `C99`. To ensure that your submission works properly on the submission system, you should use this command to compile your code on your local machine as well.

You may discuss your work with others, but what gets typed into your program must be individual work, **not** from anyone else. Do **not** give a (hard or soft) copy of your work to anyone else; do **not** "lend" your memory stick to others; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "no" when they ask for a copy of, or to see, your program, pointing out that your "no", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode.* See `https://academichonesty.unimelb.edu.au` for more information.

Deadline: Programs not submitted by **4pm Tuesday 25th May 2021** will lose penalty marks at the rate of 2 marks per day or part day late. Late submissions after 4pm Friday 28th May 2021 will **not** be accepted. Students seeking extensions for medical or other "outside my control" reasons should email the lecturer at jianzhong.qi@unimelb.edu.au. If you attend a GP or other health care professional as a result of illness, be sure to take a Health Professional Report form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops into something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

Special consideration due to COVID-19: Please refer to the "Special Consideration" section in this page: `https://students.unimelb.edu.au/your-course/manage-your-course/exams-assessments-and-results/special-consideration`

And remember, *Algorithms are awesome!*

Prepared by Jianzhong Qi