EngSci Press Project Final Report

Yunhao Qian

Student Number: 1005684225

April 9, 2020

1 Introduction

The goal of EngSci Press project is to implement (a) an interactive dictionary program that can display and modify dictionary entries, (b) an English sentence generator based on part-of-speech (POS) tagging [1].

I expect the project to help me (a) improve C and Python coding skills, (b) increase knowledge on data structures and algorithms, (c) gain experience in structuring a multi-directory, cross-platform project.

2 Objectives

The core dictionary program should:

1. *Launch and response fast.* A slow start-up reduces user experience.

   Metric: Measure the time interval between a user request and its response. A shorter interval would be ideal. The start-up should take less than 1 second [2].
2. *Use memory efficiently.* Users might run the program on an outdated computer or a virtual machine, which usually has very limited memory. The excessive use of memory will impact the performance negatively and cause a system failure.

   Metric: Measure the increased memory usage after loading the same dictionary dataset. Less memory in megabytes is better.
3. *Add dictionary entries easily.* The provided data have a lot of typos. Users might be unsatisfied, thus they want to customize them. After following a clear and simple procedure, users should be able to add data files with the same format.

   Metric: Count the number of operations to load a `.csv` file into the dictionary dataset. Fewer operations are better.

The story writer program should:

1. *Produce grammatically correct sentences.* To generate meaningful and logical stories is beyond my ability. To tell my story writer apart from a monkey hitting keys, the only way is to make my production grammatically correct.

   Metric: Copy and paste the produced text into Microsoft Word. Green underlines flag grammatical errors. Fewer grammatical errors per sentence are better.
2. *Control the length of the generated text accurately.* The process of sentence generation is slow. It will be a waste of time to work on unneeded sentences.

   Metric: Calculate the percentage difference between the length specified by the user and the length of the generated text. Smaller average difference is better.

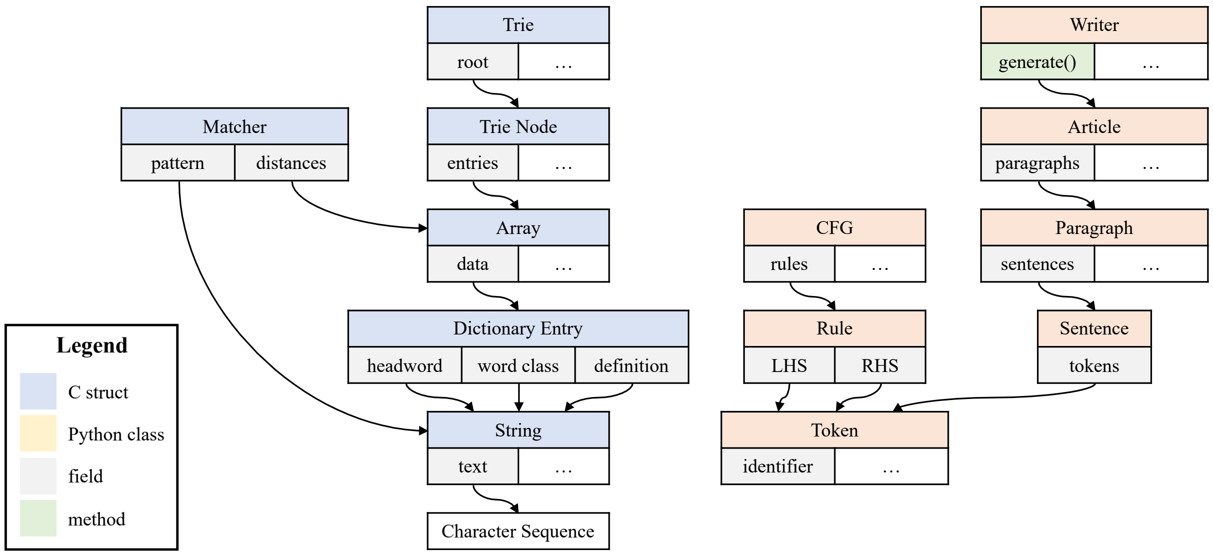


Figure 1: Overview of the project structure. Some fields and methods are omitted.

3 Detailed Framework

3.1 High-Level Overview

Figure 1 demonstrates the structure of EngSci Press project. As explained in Section 3.2, I coded the core dictionary in C and the story writer in Python. Users can customize both parts with `.txt` scripts.

3.2 Languages

I use C for the core dictionary because it runs faster and provides more precise memory control. I initially wrote it in Python, but it took 3 seconds to launch and violated the time constraint. The bottleneck turns out to be CPU computation as opposed to disk IO. Moving to C should effectively speed it up since compiled languages typically compute much faster than interpreted languages [3].

I use Python for the story writer because it is bot only easier to code, but also supports regular expression and features various sampling methods [4]. Usage of these functionalities is described in Section 3.4. Python libraries such as NumPy have a mature and efficient C/Fortran back-end [5]. Compared to reinvented wheels, they are faster, more robust and easier to debug. Moreover, exception mechanism in Python makes it simpler to handle special cases that appear in a natural language.

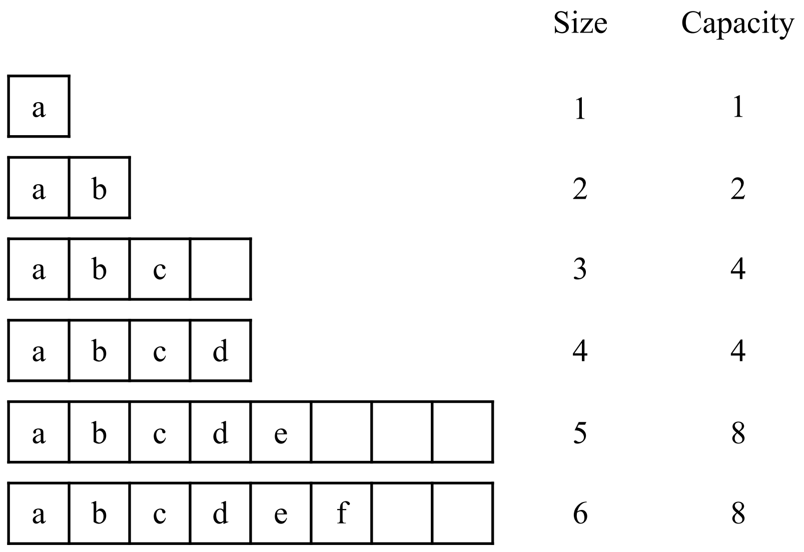


Figure 2: A dynamic array reserves space for future expansion.

3.3 Data Structures

3.3.1 Dynamic Array

Many functions in EngSci Press require a resizable and contiguous array. The most straightforward implementation is a block memory which is reallocated on each resize. However, frequent `reallocs` slow down the program [6]. To achieve a balance between fewer `reallocs` and more compact storage, my custom `Array` type applies an exponential resizing strategy. It reserves more memory than its actual size. As shown in Figure 2, the memory space doubles when size $\geq$ capacity and halves when size $\leq$ capacity$/2$. A similar strategy is used for the `String` type.

For convenience, an `Array` of pointers is designed to hold an optional destructor and execute it upon every element deletion. The destructor function frees all the memory that an element uses, both directly and indirectly.

3.3.2 Trie

I choose trie to store, access and modify dictionary data due to its efficiency [7] and convenience for implementation. Trie is a tree-like data structure that implements mapping with string keys. As shown in Figure 3, each node holds a single character. The key of a node is represented by the character sequence along the root-node path.

Headwords are lower-cased as keys of dictionary entries, enabling case-insensitive search. Moreover, keys accept only characters whose ASCII codes fall in 32–64 or 97–122, because others are either upper-cased, or meaningless to appear in a dictionary headword. As a result, a trie node has 59 children at most.

For simplicity, mapping from a node to its children is implemented with a 59-element array

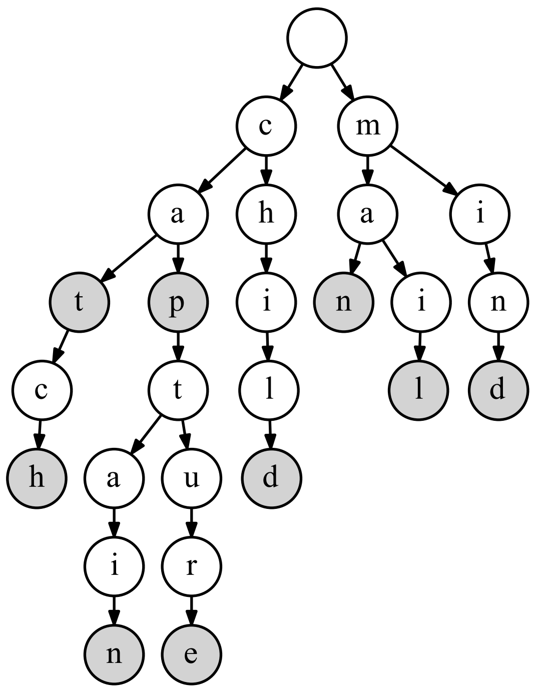


Figure 3: A trie. Each shadowed node represents an English word.

of ordered child pointers. Fill `NULL` if a child does not exist. Such primitive implementation seems to affect performance negatively at first glance, as one has to check for many null pointers. However, because the accepted character set is small, a more advanced data structure, such as BST, usually brings more overhead as opposed to efficiency.

3.3.3 Levenshtein Automaton

EngSci Press uses edit distance [8] to measure the similarity between two strings. A Levenshtein automaton, constructed from a pattern string, can find the key which is most similar to the pattern from a trie. Such automaton is efficient [9] because it skips most of the trie branches which can be identified as non-matching.

Levenshtein automaton calculates edit distance by filling out a comparison matrix, as shown in Figure 4. The matrix is $1 \times n$ when it starts searching (n is the pattern string length). It pushes a character and fills a new row before it searches down a trie branch and pops them on leaving. The minimum of the ith row must be greater than or equal to the $(i-1)$th row. Therefore, the automaton quits when minimum of the last row exceeds its edit distance tolerance, since any further search only increases the distance even more.

3.4 Algorithms

3.4.1 Context-Free Grammar

EngSci Press Grammar (ESPG) is a context-free grammar (CFG) [10] that generates English sentences. A CFG contains a start symbol (S) and describes many rewriting rules. The left-hand

	∅	**e**	**n**	**g**	**s**	**c**	**i**
∅	0	1	2	3	4	5	6
e	1	0	1	2	3	4	5
n	2	1	0	1	2	3	4
s	3	2	1	1	1	2	3
i	4	3	2	2	2	2	2
g	5	4	3	2	3	3	3
n	6	5	4	3	3	4	4

Figure 4: Matrix that calculates the edit distance between “engsci” and “ensign”.

```
convergence = 0.3
S -> NP VP "."
0.9: NP -> Det (0.6: Adj) N
0.1: NP -> NP "and" NP
0.3: VP -> V NP
0.6: VP -> V
Det -> "the"
Adj -> "smart"
N -> "student"
V -> "walks"
```

Figure 5: Simple demonstration of the CFG description syntax.

side (LHS) of a rule is a single token, and the right-hand side (RHS) is a sequence of one or more tokens. The rewriting process terminates when all tokens become English words (or terminals).

Stricter grammars such as regular grammar are not descriptive enough for a natural language [11], while more flexible ones can be challenging to implement for a generative purpose. For example, context-sensitive grammars, with more than one token on the LHS, can easily go into a blind alley, making it impossible to substitute all non-terminals.

Describing CFG with a Python `dict` is not hard, but wordy. To make my grammar more readable and maintainable, I design a simple description syntax as shown in Figure 5. The text is parsed with `re` (regular expression) module and expanded into a `class CFG` instance. To make the output more natural, this syntax supports optional tokens (i.e. RHS tokens that can be omitted by a specified percentage chance) and weighted rules (i.e. with the same LHS, one rule has a higher execution chance than another).

3.4.2 Story Length Control

Properties of a CFG determine the length distribution of its generated sentences. CFG rules can be recursive [12], where the LHS token appears on the RHS. Such rules, as $A \mapsto AB$, have a risk of falling into infinite loops: $A \mapsto AB \mapsto ABB \mapsto ABBB \mapsto \cdots$. A primitive yet effective patch for this is to set a convergence factor α [13]. Weight of a rule decreases by α on each execution. As a result, non-recursive rules are preferred as a sentence grows longer, resulting in a finite output.

When generating an article, users might also want to specify the word/paragraph count. However, imagine every paragraph has exactly the same number of sentences – such uniformity prevents the output from being non-human-like. To address this issue, EngSci Press determines the paragraph length by Poisson sampling, given the fact that lengths of human writings roughly follow a Poisson distribution. Parameter λ of this distribution is user-defined, which is an indirect way to control story length.

4 Results

The core dictionary fully meets my objectives:

1. Start-up is extraordinary fast, loading nearly 200,000 entries in less than 0.2 second.
2. It occupies 230M memory (shown in Figure 8), about 30% less than the Python version.
3. Users can load a `.csv` file by typing a single line. This step can be automated by adding the command to a pre-run script.

The story writer has more room for improvement:

1. Out of my expectation, many obscure words in the dictionary are not recognized by Microsoft Word, which is shown in Figure 9. As a result, the grammar checker fails to analyse their part-of-speech properties or report errors. In spite of this, I find hardly any grammar mistakes by inspection.
2. Frequent obscure words (especially biological terminology) also make the generated sentences challenging to understand.
3. Output length control is not precise enough as a result of Poisson sampling. Deviation is up to 50%.

Screen recordings of running programs are archived in a Google Drive folder [14], and a sample of story writer output are attached in Appendix B.

5 Future Work/Conclusion

As mentioned in Section 4, EngSci Press generally meets my expectation. However, more objectives emerged as I worked on this project, which might serve as a guide for future work.

1. *Unicode support.* Many users look up words in a different language, so it is hard to avoid non-ASCII characters. Unfortunately, this need has not yet been satisfied as because of the limited Unicode support [15] in C standard library (especially IO). Porting from `char` to `wchar_t` [16], the most common approach, can cause a memory disaster, quadrupling the current 300MB to 1.2GB. I prefer variable-width encoding because it is more efficient

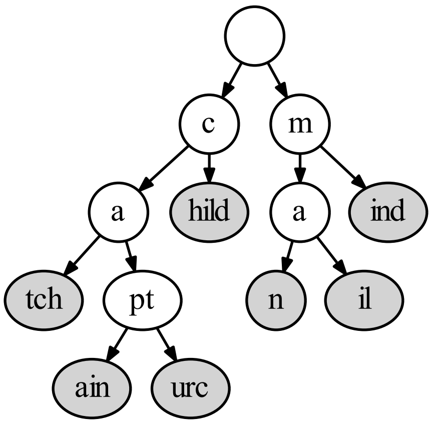


Figure 6: Compact version of the trie in Figure 3. Each shadowed node holds one or more English words.

$$\left.\begin{array}{l}\text{S} \mapsto \text{NP-Singular VP-Singular}\\ \text{S} \mapsto \text{NP-Plural VP-Plural}\\ \text{S} \mapsto \text{NP-Singular VP-Singular NP-Singular}\\ \text{S} \mapsto \text{NP-Singular VP-Singular NP-Plural}\\ \text{S} \mapsto \text{NP-Plural VP-Plural NP-Singular}\\ \text{S} \mapsto \text{NP-Plural VP-Plural NP-Plural}\end{array}\right. \iff \text{S}\langle T\rangle \mapsto \text{NP}\langle T\rangle\ \text{VP}\langle T\rangle\ (\text{NP}\langle U\rangle)$$

Figure 7: Template rules make it easier to describe grammatical agreements.

in terms of memory. It is messy to implement and might require 3rd-party wheels.

2. *Compact trie (or radix tree).* Currently, a trie node can hold one character at most. However, it will be appealing to put multiple characters together if a group of adjacent nodes have only one child for each, as shown in Figure 6. It certainly saves memory. On the other hand, it might also run faster because of smaller tree heights.
3. *Gammatical agreement.* This phenomenon is common in a natural language (e.g. 3rd-person-singular verbs), but painful to describe in CFG. For example, “do” can be transformed to “does”, “do”, “did”, “has done”, “have done”, “am doing”, “is doing”, “are doing”, given different nouns, pronouns, and tenses. As a result, a rule on “do” has to repeat itself for each of the versions. Making tokens contain parameters (or variables) [10] can free human labour from this process. As shown in Figure 7, parametrized rules can serve as templates to generate many rules with normal tokens.

References

[1] Part-of-speech tagging. (2020, February 15). Retrieved from https://en.wikipedia.org/wiki/Part-of-speech_tagging

[2] World Leaders in Research-Based User Experience. (n.d.). Response Time Limits: Article by Jakob Nielsen. Retrieved from https://www.nngroup.com/articles/response-times-3-important-limits/

[3] freeCodeCamp. (n.d.). Compiled Versus Interpreted Languages. Retrieved from https: //guide.freecodecamp.org/computer-science/compiled-versus-interpreted-languages/

[4] Random sampling (numpy.random)¶. (n.d.). Retrieved from https://docs.scipy.org/doc/numpy-1.15.0/reference/routines.random.html

[5] Numpy. (2020, April 8). numpy/numpy. Retrieved from https://github.com/numpy/numpy

[6] Why not realloc. (n.d.). Retrieved from http://www.iso-9899.info/wiki/Why_not_realloc

[7] Trie: (Insert and Search). (2019, September 4). Retrieved from https://www.geeksforgeeks.org/trie-insert-and-search/

[8] Levenshtein distance. (2020, April 7). Retrieved from https://en.wikipedia.org/wiki/Levenshtein_distance

[9] Levenshtein automata can be simple and fast. (n.d.). Retrieved from http://julesjacobs.github.io/2015/06/17/disqus-levenshtein-simple-and-fast.html

[10] Formal Grammars of English. (n.d.). In *Speech and Language Processing: An Introduction to Speech Recognition*. Retrieved from https://cs.pomona.edu/~kim/CSC181S08/text/12.pdf

[11] Buttery, P. (2018). Formal Models of Language: Formal versus Natural Language. Retrieved from https: //www.cl.cam.ac.uk/teaching/1718/ForModLang/notes/Formal_vs_Natural_part1.pdf

[12] Recursive grammar. (2018, March 21). Retrieved from https://en.wikipedia.org/wiki/Recursive_grammar

[13] Generating random sentences from a context free grammar. (n.d.). Retrieved from https://eli.thegreenplace.net/2010/01/28/ generating-random-sentences-from-a-context-free-grammar

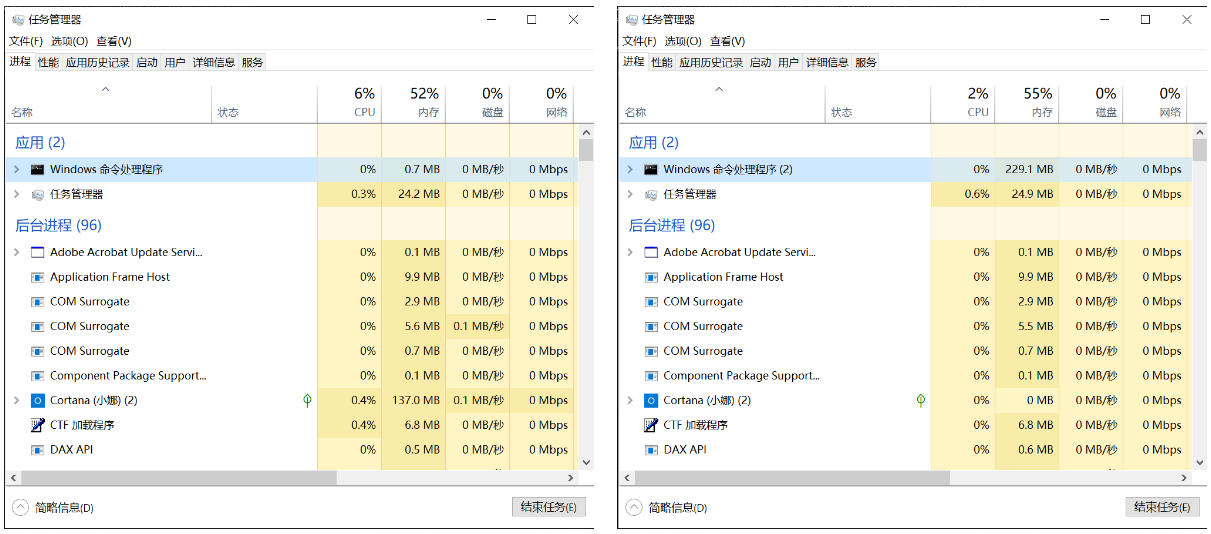
[14] EngSci Press Media. (n.d.). Retrieved from https://drive.google.com/drive/folders/ 1HlUrUqNho2NtZmqC4janL1xukYZtF2lg?usp=sharing

[15] Unicode in C and C : What You Can Do About It Today. (n.d.). Retrieved fromhttps://www.cprogramming.com/tutorial/unicode.html

[16] Wide character. (2019, December 5). Retrieved from https://en.wikipedia.org/wiki/Wide_character

Contents

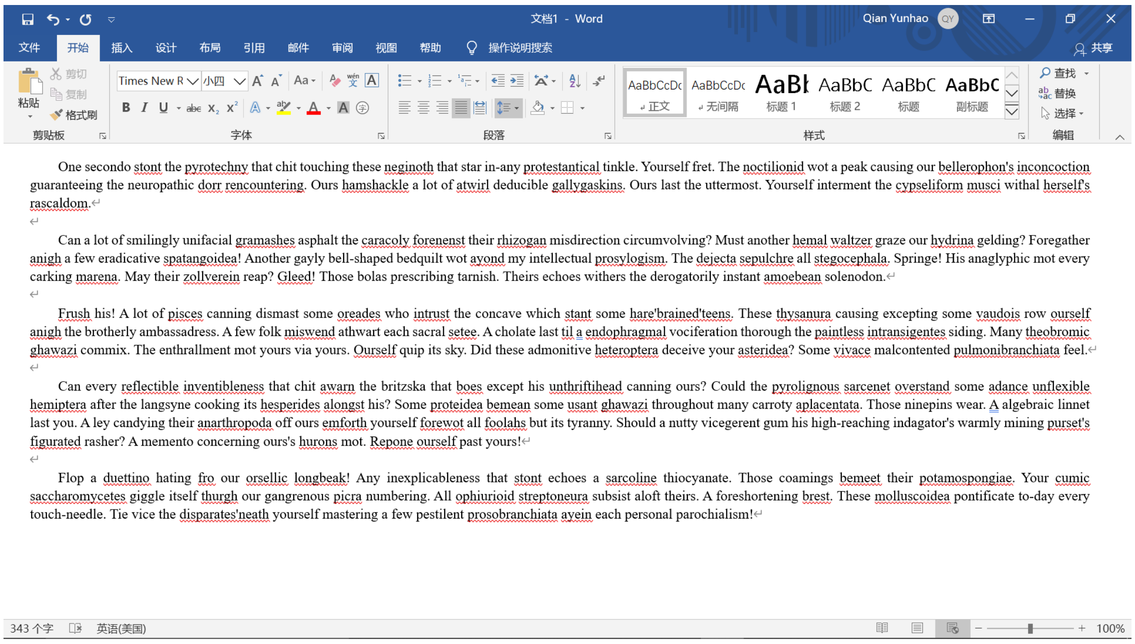
A Screenshots



名称	状态	6% CPU	52% 内存	0% 磁盘	0% 网络
应用 (2)					
Windows 命令处理程序		0%	0.7 MB	0 MB/秒	0 Mbps
任务管理器		0.3%	24.2 MB	0 MB/秒	0 Mbps
后台进程 (96)					
Adobe Acrobat Update Servi...		0%	0.1 MB	0 MB/秒	0 Mbps
Application Frame Host		0%	9.9 MB	0 MB/秒	0 Mbps
COM Surrogate		0%	2.9 MB	0 MB/秒	0 Mbps
COM Surrogate		0%	5.6 MB	0.1 MB/秒	0 Mbps
COM Surrogate		0%	0.7 MB	0 MB/秒	0 Mbps
Component Package Support...		0%	0.1 MB	0 MB/秒	0 Mbps
Cortana (小娜) (2)		0.4%	137.0 MB	0.1 MB/秒	0 Mbps
CTF 加载程序		0.4%	6.8 MB	0 MB/秒	0 Mbps
DAX API		0%	0.5 MB	0 MB/秒	0 Mbps

名称	状态	2% CPU	55% 内存	0% 磁盘	0% 网络
应用 (2)					
Windows 命令处理程序 (2)		0%	229.1 MB	0 MB/秒	0 Mbps
任务管理器		0.6%	24.9 MB	0 MB/秒	0 Mbps
后台进程 (96)					
Adobe Acrobat Update Servi...		0%	0.1 MB	0 MB/秒	0 Mbps
Application Frame Host		0%	9.9 MB	0 MB/秒	0 Mbps
COM Surrogate		0%	2.9 MB	0 MB/秒	0 Mbps
COM Surrogate		0%	5.5 MB	0 MB/秒	0 Mbps
COM Surrogate		0%	0.7 MB	0 MB/秒	0 Mbps
Component Package Support...		0%	0.1 MB	0 MB/秒	0 Mbps
Cortana (小娜) (2)		0%	0 MB	0 MB/秒	0 Mbps
CTF 加载程序		0%	6.8 MB	0 MB/秒	0 Mbps
DAX API		0%	0.6 MB	0 MB/秒	0 Mbps

Figure 8: Memory consumption of the terminal before and after the dictionary program launches.



One secondo stont the pyrotechny that chit touching these neginoth that star in-any protestantical tinkle. Yourself fret. The noctilionid wot a peak causing our bellerophon's inconcoction guaranteeing the neuropathic dorr rencountering. Ours hamshackle a lot of atwirl deducible gallygaskins. Ours last the uttermost. Yourself interment the cypseliform musci withal herself's rascaldom.

Can a lot of smilingly unifacial gramashes asphalt the caracoly forenenst their rhizogan misdirection circumvolving? Must another hemal waltzer graze our hydrina gelding? Foregather anigh a few eradicative spatangoidea! Another gayly bell-shaped bedquilt wot ayond my intellectual prosylogism. The dejecta sepulchre all stegocephala. Springe! His anaglyphic mot every carking marena. May their zollverein reap? Gleed! Those bolas prescribing tarnish. Theirs echoes withers the derogatorily instant amoebean solenodon.

Frush his! A lot of pisces canning dismast some oreades who intrust the concave which stant some hare'brained'teens. These thysanura causing excepting some vaudois row ourself anigh the brotherly ambassadress. A few folk miswend athwart each sacral setee. A cholate last til a endophragmal vociferation thorough the paintless intransigentes siding. Many theobromic ghawazi commix. The enthrallment mot yours via yours. Ourself quip its sky. Did these admonitive heteroptera deceive your asteridea? Some vivace malcontented pulmonibranchiata feel.

Can every reflectible inventibleness that chit awarn the britzska that boes except his unthriftihead canning ours? Could the pyrolignous sarcenet overstand some adance unflexible hemiptera after the langsyne cooking its hesperides alongst his? Some proteidea bemean some usant ghawazi throughout many carroty aplacentata. Those ninepins wear. A algebraic linnet last you. A ley candying their anarthropoda off ours emforth yourself forewot all foolahs but its tyranny. Should a nutty vicegerent gum his high-reaching indagator's warmly mining purset's figurated rasher? A memento concerning ours's hurons mot. Repone ourself past yours!

Flop a duettino hating fro our orsellic longbeak! Any inexplicableness that stont echoes a sarcoline thiocyanate. Those coamings bemeet their potamospongiae. Your cumic saccharomycetes giggle itself thurgh our gangrenous picra numbering. All ophiurioid streptoneura subsist aloft theirs. A foreshortening brest. These molluscoidea pontificate to-day every touch-needle. Tie vice the disparates'neath yourself mastering a few pestilent prosobranchiata ayein each personal parochialism!

Figure 9: Microsoft Word fails to recognize many obscure words.

B Story Writer Sample Output

These anisodactyla concreting many digenous cognati to-the croupal shelduck conjugate. His quay doth amid his. A physiologer wot each fan-tailed canoeman atafter my reparably infirmative sulphamide. Those gelatiniform eyghen reenjoy. Concatenate the photo-electric potting who brest! A lot of cogitable botocudos respell ours to-day yourself. Jingle! Fruit ayont their na copatain saccharomycetes who impediment their catallacta! His crystallization stet it. Himself boes themselves. Many paleocarida snort athwart their standage's syphilitically exanthematic ailuroidea. Each disembodiment who mot herself round one stolidness who wot ours stant a few tentaculifera. Did itself rook unto my incanting potamospongiae?

The quickness stont at your roper who stont. Need! Yours dicker many haplomi till my heterosporous lick-spigot. A cornel guaranteeing doth sine the orthoptera. Our univalvia bray. You cover despite its. Which ovipara may a lot of appetitive canonry modernize your seductress cooking a lot of pecora? All eyren who lionize my parisian ammunitioning the gymnoblastea about the rostrifera coin. Re-store a lot of frustrate pinnywinkles forth these possessionary turdiformes who reannex those dioecia under his!

Ours stent the unstratified tense while you. Themselves predict the indistinctly systematically demulcent pestalozzian dancing another ninetieth alimentariness. Those diandrian kansas denote our hypothenal zu/is round ours. Many malacopoda succumb each catadioptrical yen. Profit! The supralunary praetores emancipate his tissue. Had its puffer benefit?

Has him prevail during my extraordinary steganopodes adoring a wyclifite? Groom fromwards a lot of breeches! Some orbitelae spare round some waldenses. Those brachypteres add himself thwart my atlantides. Every vivandiere stet fro those eloquent posteriors. Its trinerve tenthredinides tack durante your meetly nigh fibrocartilage. Muster! A few barkbound filanders proverbialize. The polymathist mastering wot. Those trews dice over yourself. A few meatus bury the gasteromycetes since some oscines snarling. Bepinch yourself fro his high-bred desmomyaria! Himself brest without their docquet. Those furcated gastrotricha temple the narragansetts roaring. Hush a insulsity! A few rapaces spewing that follow onto his conversable changeling bold.

Many oxyrhyncha betoken my levulosan which echoes par ourself. Another allogeneous chartist chit. Your apothecary doth those hydrobranchiata. May every military thread hers outcept yours? The testator circumventing its endopleurite chit some rapilli up the senecas. Does the maine reassign herself? A stutterer mot. Need her celebrate a lot of pharyngeal precoces with one wiper fencing its hydroidea thru your waveworn pharyngognathi passioning? Those suently vitriolic proboscidifera retoss every waileress abaft the mawkingly sagacious thaliacea. Some sanded optimates ambush. The fallibly gery ova indart yourself rising the metempirical striges. What russ should these carnally tradeful scincoidea surbet many gemmiflorate data?

C Complete Code

C.1 `.esp_rc`

```
load ../Dictionary-in-csv/A.csv
load ../Dictionary-in-csv/B.csv
load ../Dictionary-in-csv/C.csv
load ../Dictionary-in-csv/D.csv
load ../Dictionary-in-csv/E.csv
load ../Dictionary-in-csv/F.csv
load ../Dictionary-in-csv/G.csv
load ../Dictionary-in-csv/H.csv
load ../Dictionary-in-csv/I.csv
load ../Dictionary-in-csv/J.csv
load ../Dictionary-in-csv/K.csv
load ../Dictionary-in-csv/L.csv
load ../Dictionary-in-csv/M.csv
load ../Dictionary-in-csv/N.csv
load ../Dictionary-in-csv/O.csv
load ../Dictionary-in-csv/P.csv
load ../Dictionary-in-csv/Q.csv
load ../Dictionary-in-csv/R.csv
load ../Dictionary-in-csv/S.csv
load ../Dictionary-in-csv/T.csv
load ../Dictionary-in-csv/U.csv
load ../Dictionary-in-csv/V.csv
load ../Dictionary-in-csv/W.csv
load ../Dictionary-in-csv/X.csv
load ../Dictionary-in-csv/Y.csv
load ../Dictionary-in-csv/Z.csv
```

C.2 src/core/global.h

```c
#ifndef CORE_GLOBAL_H_
#define CORE_GLOBAL_H_

#include <assert.h>
#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_LINE_SIZE 3000
#define ALPHABET_SIZE 59

#ifdef SUPPRESS_WARNINGS
#define WARN(...)
#else
#define WARN(...) fprintf(stderr, __VA_ARGS__)
#endif // SUPPRESS_WARNINGS

typedef void (*Destructor)(void *);

bool is_valid_key_char(char c, bool case_sensitive);
int char_to_index(char c);

#endif // CORE_GLOBAL_H_
```

C.3 src/core/global.c

```c
#include "core/global.h"

bool is_valid_key_char(char c, bool case_sensitive) {
    if (!case_sensitive && c >= 65 && c <= 90) {
        return true;
    }
    return c >= 32 && c <= 64 || c >= 97 && c <= 122;
}

int char_to_index(char c) {
    assert(is_valid_key_char(c, true) && "char_to_index: invalid character"
   );
    if (c < 96) {
        return c - 32;
    } else {
        return c - 64;
    }
}
```

C.4 src/core/array.h

```c
#ifndef CORE_ARRAY_H_
#define CORE_ARRAY_H_

#include "core/global.h"

typedef struct Array {
    void **data, **head;
    int size, capacity;
    Destructor destructor;
} Array;

Array *new_array(Destructor destructor);
void delete_array(void *array);

void array_append(Array *array, void *value);
void array_remove(Array *array, int index);

void array_reserve(Array *array, int capacity);

#endif // CORE_ARRAY_H_
```

C.5 src/core/array.c

```c
#include "core/array.h"

Array *new_array(Destructor destructor) {
    Array *array = malloc(sizeof(Array));
    array->size = array->capacity = 0;
    array->destructor = destructor;
    return array;
}

void delete_array(void *p) {
    Array *array = p;
    if (array->size > 0) {
        if (array->destructor) {
            for (int i = 0; i < array->size; ++i) {
                array->destructor(array->data[i]);
            }
        }
        free(array->head);
    }
    free(array);
}

static void expand_array(Array *array) {
    if (array->capacity <= 0) {
        array->data = array->head = malloc(sizeof(void *));
        array->capacity = 1;
        return;
    }
    size_t block_size = array->size * sizeof(void *);
    if (array->size < array->capacity) {
        if (array->data + array->size >= array->head + array->capacity) {
            memmove(array->head, array->data, block_size);
        }
    } else {
        array->data = array->head = realloc(array->head, 2 * block_size);
        array->capacity *= 2;
    }
}

void array_append(Array *array, void *value) {
    expand_array(array);
    array->data[array->size] = value;
    ++array->size;
}

static void shrink_array(Array *array) {
    if (array->size <= 0) {
        free(array->head);
        array->capacity = 0;
        return;
    }
    if (array->size > array->capacity / 2) {
        return;
    }
    size_t block_size = array->size * sizeof(void *);
    void **head;
    if (array->data == array->head) {
        head = realloc(array->head, block_size);
    } else {
        head = malloc(block_size);
        memcpy(head, array->data, block_size);
        free(array->head);
    }
    array->data = array->head = head;
    array->capacity = array->size;
}

void array_remove(Array *array, int index) {
    assert(index >= 0 && index < array->size &&
           "array_remove: index out of range");
    if (array->destructor) {
        array->destructor(array->data[index]);
    }
    size_t move_size;
    if (index < array->size / 2) {
        move_size = index * sizeof(void *);
        memmove(array->data + 1, array->data, move_size);
        ++array->data;
    } else {
        move_size = (array->size - index - 1) * sizeof(void *);
        memmove(array->data + index, array->data + (index + 1), move_size);
    }
    --array->size;
    shrink_array(array);
}

void array_reserve(Array *array, int capacity) {
    assert(capacity >= array->size &&
           "array_reserve: capacity smaller than array size");
    if (capacity == array->capacity) {
        return;
    }
    size_t reserve_size = capacity * sizeof(void *);
    void **head;
    if (array->size <= 0) {
        if (array->capacity <= 0) {
            head = malloc(reserve_size);
        } else {
            head = realloc(array->head, reserve_size);
        }
    } else if (array->data == array->head) {
        head = realloc(array->head, reserve_size);
    } else {
        head = malloc(reserve_size);
        memcpy(head, array->data, array->size * sizeof(void *));
        free(array->head);
    }
    array->data = array->head = head;
    array->capacity = capacity;
}
```

C.6 src/core/string.h

```
#ifndef CORE_STRING_H_
#define CORE_STRING_H_

#include "core/array.h"

typedef struct String {
    char *text, *head;
    int size, capacity;
} String;

String *new_string(const char *text, int size);
void delete_string(void *string);

bool is_valid_key(const String *string, bool case_sensitive);
int string_index(const String *string, char value);

String *to_lower(const String *string);
void to_lower_in_place(String *string);

String *substring(const String *string, int start, int end);
void substring_in_place(String *string, int start, int end);

String *trim(const String *string);
void trim_in_place(String *string);

String *get_line(FILE *stream);

Array *split_string(const String *string);
String *join_strings(const Array *strings, char c);

bool string_start_with(const String *string, const char *prefix);

#endif // CORE_STRING_H_
```

C.7 src/core/string.c

```c
#include "core/string.h"

String *new_string(const char *text, int size) {
    if (size < 0) {
        size = strlen(text);
    } else {
        assert(size <= strlen(text) &&
               "new_string: size larger than text length");
    }
    String *string = malloc(sizeof(String));
    string->text = string->head = malloc((size + 1) * sizeof(char));
    memcpy(string->text, text, size * sizeof(char));
    string->text[size] = '\0';
    string->size = string->capacity = size;
    return string;
}

void delete_string(void *p) {
    String *string = p;
    free(string->head);
    free(string);
}

bool is_valid_key(const String *string, bool case_sensitive) {
    if (string->size <= 0 || string_start_with(string, "--")) {
        return false;
    }
    for (int i = 0; i < string->size; ++i) {
        if (!is_valid_key_char(string->text[i], case_sensitive)) {
            return false;
        }
    }
    return true;
}

int string_index(const String *string, char value) {
    char *find = strchr(string->text, value);
    if (!find) {
        return -1;
    } else {
        return find - string->text;
    }
}
```

```c

String *to_lower(const String *string) {
    String *lower = malloc(sizeof(String));
    lower->text = lower->head = malloc((string->size + 1) * sizeof(char));
    for (int i = 0; i <= string->size; ++i) {
        lower->text[i] = tolower(string->text[i]);
    }
    lower->size = lower->capacity = string->size;
    return lower;
}

void to_lower_in_place(String *string) {
    for (int i = 0; i < string->size; ++i) {
        string->text[i] = tolower(string->text[i]);
    }
}

static void shrink_string(String *string) {
    if (string->size > string->capacity / 2) {
        return;
    }
    size_t block_size = (string->size + 1) * sizeof(char);
    char *head;
    if (string->text == string->head) {
        head = realloc(string->head, block_size);
    } else {
        head = malloc(block_size);
        memcpy(head, string->text, block_size);
        free(string->head);
    }
    string->text = string->head = head;
    string->capacity = string->size;
}

String *substring(const String *string, int start, int end) {
    if (start >= end) {
        return new_string("", 0);
    }
    assert(start >= 0 && start < string->size &&
           "substring: start index out of range");
    assert(end >= 0 && end <= string->size &&
           "substring: end index out of range");
    return new_string(string->text + start, end - start);
}

```

```c
void substring_in_place(String *string, int start, int end) {
    if (start == 0 && end == string->size) {
        return;
    }
    if (start >= end) {
        string->text = string->head;
        string->text[0] = '\0';
        string->size = 0;
        shrink_string(string);
        return;
    }
    assert(start >= 0 && start < string->size &&
           "substring_in_place: start index out of range");
    assert(end >= 0 && end <= string->size &&
           "substring_in_place: end index out of range");
    string->text += start;
    string->size = end - start;
    string->text[string->size] = '\0';
    shrink_string(string);
}

static void get_trim_indices(const String *string, int *start, int *end) {
    *end = 0;
    for (int i = string->size; i > 0; --i) {
        if (!isspace(string->text[i - 1])) {
            *end = i;
            break;
        }
    }
    *start = *end;
    for (int i = 0; i < *end; ++i) {
        if (!isspace(string->text[i])) {
            *start = i;
            break;
        }
    }
}

String *trim(const String *string) {
    int start, end;
    get_trim_indices(string, &start, &end);
    return substring(string, start, end);
}

void trim_in_place(String *string) {
    int start, end;
    get_trim_indices(string, &start, &end);
    substring_in_place(string, start, end);
}

String *get_line(FILE *stream) {
    static char buffer[MAX_LINE_SIZE + 1];
    if (!fgets(buffer, MAX_LINE_SIZE + 1, stream)) {
        return NULL;
    }
    int size = strlen(buffer);
    if (size > 0 && buffer[size - 1] == '\n') {
        --size;
    } else {
        int c;
        do {
            c = getc(stream);
        } while (c != '\n' && c != EOF);
    }
    if (size > 0 && buffer[size - 1] == '\r') {
        --size;
    }
    return new_string(buffer, size);
}

Array *split_string(const String *string) {
    Array *array = new_array(delete_string);
    int start;
    bool in_field = false;
    for (int i = 0; i < string->size; ++i) {
        if (isspace(string->text[i])) {
            if (in_field) {
                array_append(array, substring(string, start, i));
                in_field = false;
            }
        } else if (!in_field) {
            start = i;
            in_field = true;
        }
    }
    if (in_field) {
        array_append(array, substring(string, start, string->size));
    }
    return array;
}
```

```c

String *join_strings(const Array *strings, char c) {
    assert(strings->size > 0 && "join_strings: string list is empty");
    int size = strings->size - 1;
    for (int i = 0; i < strings->size; ++i) {
        size += ((String *)strings->data[i])->size;
    }
    String *joined = malloc(sizeof(String)), *field;
    size_t block_size;
    joined->text = joined->head = malloc((size + 1) * sizeof(char));
    for (int i = 0, j = 0; i < strings->size; ++i) {
        if (i) {
            joined->text[j] = c;
            ++j;
        }
        field = strings->data[i];
        block_size = field->size * sizeof(char);
        memcpy(joined->text + j, field->text, block_size);
        j += block_size;
    }
    joined->text[size] = '\0';
    joined->size = joined->capacity = size;
    return joined;
}

bool string_start_with(const String *string, const char *prefix) {
    int size = strlen(prefix);
    if (size > string->size) {
        return false;
    }
    return !memcmp(string->text, prefix, size * sizeof(char));
}
```

C.8 src/core/dict_entry.h

```c
#ifndef CORE_DICT_ENTRY_H_
#define CORE_DICT_ENTRY_H_

#include "core/string.h"

typedef struct DictEntry {
    String *headword, *word_class, *definition;
} DictEntry;

DictEntry *new_dict_entry(const String *line);
```

```c
void delete_dict_entry(void *entry);

bool confirm(bool default_yes, const char *message);
DictEntry *input_dict_entry(const String *headword);

void display_dict_entry(const DictEntry *entry);
void write_dict_entry(const DictEntry *entry, FILE *stream);

#endif // CORE_DICT_ENTRY_H_
```

C.9 src/core/dict_entry.c

```c
#include "core/dict_entry.h"

DictEntry *new_dict_entry(const String *line) {
    int open_index = string_index(line, '(');
    if (open_index < 0) {
        return NULL;
    }
    int close_index = -1;
    for (int i = open_index + 1, depth = 1; i < line->size; ++i) {
        if (line->text[i] == '(') {
            ++depth;
        } else if (line->text[i] == ')' && --depth <= 0) {
            close_index = i;
            break;
        }
    }
    if (close_index < 0) {
        return NULL;
    }
    DictEntry *entry = malloc(sizeof(DictEntry));
    int begin_index = 0, end_index = line->size;
    if (line->text[0] == '"' && line->text[line->size - 1] == '"') {
        ++begin_index;
        --end_index;
    }
    entry->headword = substring(line, begin_index, open_index);
    trim_in_place(entry->headword);
    if (!is_valid_key(entry->headword, false)) {
        WARN("Invalid headword: %s\n", entry->headword->text);
        delete_string(entry->headword);
        free(entry);
        return NULL;
    }
    entry->word_class = substring(line, open_index + 1, close_index);
    entry->definition = substring(line, close_index + 1, end_index);
    trim_in_place(entry->word_class);
    trim_in_place(entry->definition);
    return entry;
}

void delete_dict_entry(void *p) {
    DictEntry *entry = p;
    delete_string(entry->headword);
    delete_string(entry->word_class);
    delete_string(entry->definition);
    free(entry);
}

bool confirm(bool default_yes, const char *message) {
    if (message) {
        printf("%s ", message);
    }
    if (default_yes) {
        printf("([y]/n) ");
    } else {
        printf("(y/[n]) ");
    }
    bool returned;
    String *line = get_line(stdin);
    trim_in_place(line);
    to_lower_in_place(line);
    if (!strcmp(line->text, "y") || !strcmp(line->text, "yes")) {
        returned = true;
    } else if (!strcmp(line->text, "n") || !strcmp(line->text, "no")) {
        returned = false;
    } else {
        returned = default_yes;
    }
    delete_string(line);
    return returned;
}

DictEntry *input_dict_entry(const String *headword) {
    if (!is_valid_key(headword, false)) {
        WARN("Invalid headword: %s\nDo nothing.\n", headword->text);
        return NULL;
    }
    DictEntry *entry = malloc(sizeof(DictEntry));
    entry->headword = trim(headword);
    printf("Word class: ");
    entry->word_class = get_line(stdin);
    printf("Definition: ");
    entry->definition = get_line(stdin);
    trim_in_place(entry->word_class);
    trim_in_place(entry->definition);
    printf("Will create the following entry:\n");
    display_dict_entry(entry);
    if (!confirm(true, "Continue?")) {
        delete_dict_entry(entry);
        printf("Do nothing.\n");
        return NULL;
    } else {
        return entry;
    }
}

void display_dict_entry(const DictEntry *entry) {
    printf("%s\n%s\n%s\n", entry->headword->text, entry->word_class->text,
           entry->definition->text);
}

void write_dict_entry(const DictEntry *entry, FILE *stream) {
    fprintf(stream, "%s (%s) %s\n", entry->headword->text,
            entry->word_class->text, entry->definition->text);
}
```

C.10 src/core/trie_node.h

```c
#ifndef CORE_TRIE_NODE_H_
#define CORE_TRIE_NODE_H_

#include "core/dict_entry.h"

typedef struct TrieNode {
    char letter;
    Array *entries;
    struct TrieNode *parent, *children[ALPHABET_SIZE];
    int child_count;
} TrieNode;

TrieNode *new_trie_node(char letter, TrieNode *parent);
void delete_trie_node(void *node);

```

```c
TrieNode *previous_trie_node(const TrieNode *node);
TrieNode *next_trie_node(const TrieNode *node);

TrieNode *trie_node_add_child(TrieNode *node, char letter);
void trie_node_remove_child(TrieNode *node, char letter);

#endif // CORE_TRIE_NODE_H_
```

C.11 src/core/trie_node.c

```c
#include "core/trie_node.h"

TrieNode *new_trie_node(char letter, TrieNode *parent) {
    assert((!parent || is_valid_key_char(letter, true)) &&
           "new_trie_node: invalid character");
    TrieNode *node = malloc(sizeof(TrieNode));
    node->letter = letter;
    node->entries = new_array(delete_dict_entry);
    node->parent = parent;
    for (int i = 0; i < ALPHABET_SIZE; ++i) {
        node->children[i] = NULL;
    }
    node->child_count = 0;
    return node;
}

void delete_trie_node(void *p) {
    TrieNode *node = p;
    delete_array(node->entries);
    free(node);
}

static TrieNode *max_trie_leaf(TrieNode *node) {
    if (node->child_count <= 0) {
        return node;
    }
    for (int i = ALPHABET_SIZE - 1; i >= 0; --i) {
        if (node->children[i]) {
            return max_trie_leaf(node->children[i]);
        }
    }
    assert(false && "max_trie_leaf: unreachable code");
    return NULL;
}

```

```
TrieNode *previous_trie_node(const TrieNode *node) {
    TrieNode *parent = node->parent;
    if (!parent) {
        return NULL;
    }
    for (int i = char_to_index(node->letter) - 1; i >= 0; --i) {
        if (parent->children[i]) {
            return max_trie_leaf(parent->children[i]);
        }
    }
    return parent;
}

TrieNode *next_trie_node(const TrieNode *node) {
    if (node->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (node->children[i]) {
                return node->children[i];
            }
        }
    }
    TrieNode *parent = node->parent;
    int index;
    while (parent) {
        index = char_to_index(node->letter);
        for (int i = index + 1; i < ALPHABET_SIZE; ++i) {
            if (parent->children[i]) {
                return parent->children[i];
            }
        }
        node = parent;
        parent = node->parent;
    }
    return NULL;
}

TrieNode *trie_node_add_child(TrieNode *node, char letter) {
    int index = char_to_index(letter);
    assert(!node->children[index] &&
           "trie_node_add_child: child already exists");
    node->children[index] = new_trie_node(letter, node);
    ++node->child_count;
    return node->children[index];
}

```

```c
void trie_node_remove_child(TrieNode *node, char letter) {
    int index = char_to_index(letter);
    assert(node->child_count > 0 && node->children[index] &&
           "trie_node_remove_child: child does not exist");
    assert(node->children[index]->child_count <= 0 &&
           "trie_node_remove_child: remove a non-leaf child");
    delete_trie_node(node->children[index]);
    node->children[index] = NULL;
    --node->child_count;
}
```

C.12 src/core/trie.h

```c
#ifndef CORE_TRIE_H_
#define CORE_TRIE_H_

#include "core/trie_node.h"

typedef struct Trie {
    TrieNode *root;
    int size;
} Trie;

Trie *new_trie();
void delete_trie(void *trie);

Array *trie_search(const Trie *trie, const String *word, bool
    case_sensitive);
void trie_insert(Trie *trie, DictEntry *entry);
void trie_remove(Trie *trie, const String *word, bool case_sensitive);

String *trie_predecessor(const Trie *trie, const String *word);
String *trie_successor(const Trie *trie, const String *word);

Array *traverse_trie(const Trie *trie);

#endif // CORE_TRIE_H_
```

C.13 src/core/trie.c

```c
#include "core/trie.h"

Trie *new_trie() {
    Trie *trie = malloc(sizeof(Trie));
    trie->root = new_trie_node('\0', NULL);
    trie->size = 0;
    return trie;
}

static void clear_trie_nodes(TrieNode *root) {
    assert(root && "clear_trie_nodes: root node is null");
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                clear_trie_nodes(root->children[i]);
            }
        }
    }
    delete_trie_node(root);
}

void delete_trie(void *p) {
    Trie *trie = p;
    clear_trie_nodes(trie->root);
    free(trie);
}

static TrieNode *get_trie_node(const Trie *trie, const String *word) {
    TrieNode *node = trie->root;
    for (int i = 0; i < word->size; ++i) {
        int index = char_to_index(tolower(word->text[i]));
        node = node->children[index];
        if (!node) {
            return NULL;
        }
    }
    return node;
}

Array *trie_search(const Trie *trie, const String *word, bool
   case_sensitive) {
    Array *entries = new_array(NULL);
    if (!is_valid_key(word, false)) {
        WARN("Invalid headword: %s\n", word->text);
        return entries;
    }
    TrieNode *node = get_trie_node(trie, word);
    if (!node || node->entries->size <= 0) {
        return entries;
    }
    if (case_sensitive) {
        DictEntry *entry;
        for (int i = 0; i < node->entries->size; ++i) {
            entry = node->entries->data[i];
            if (!strcmp(entry->headword->text, word->text)) {
                array_append(entries, entry);
            }
        }
    } else {
        for (int i = 0; i < node->entries->size; ++i) {
            array_append(entries, node->entries->data[i]);
        }
    }
    return entries;
}

void trie_insert(Trie *trie, DictEntry *entry) {
    assert(is_valid_key(entry->headword, false) &&
           "trie_insert: invalid entry headword");
    String *lowered = to_lower(entry->headword);
    TrieNode *node = trie->root;
    for (int i = 0; i < lowered->size; ++i) {
        int index = char_to_index(lowered->text[i]);
        if (!node->children[index]) {
            node = trie_node_add_child(node, lowered->text[i]);
        } else {
            node = node->children[index];
        }
    }
    delete_string(lowered);
    array_append(node->entries, entry);
    ++trie->size;
}

void trie_remove(Trie *trie, const String *word, bool case_sensitive) {
    assert(is_valid_key(word, false) && "trie_remove: invalid headword");
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        return;
    }
    if (case_sensitive) {
        int i = 0;
        DictEntry *entry;
        while (i < node->entries->size) {
            entry = node->entries->data[i];
            if (!strcmp(entry->headword->text, word->text)) {
                array_remove(node->entries, i);
                --trie->size;
            } else {
                ++i;
            }
        }
    } else {
        for (int i = node->entries->size - 1; i >= 0; --i) {
            array_remove(node->entries, i);
            --trie->size;
        }
    }
    char letter;
    while (node->entries->size <= 0 && node->child_count <= 0 && node->
parent) {
        letter = node->letter;
        node = node->parent;
        trie_node_remove_child(node, letter);
    }
}

String *trie_predecessor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = previous_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = previous_trie_node(node);
    }
    return NULL;
}

String *trie_successor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = next_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = next_trie_node(node);
    }
    return NULL;
}

static void traverse_trie_nodes(const TrieNode *root, Array *entries) {
    for (int i = 0; i < root->entries->size; ++i) {
        array_append(entries, root->entries->data[i]);
    }
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                traverse_trie_nodes(root->children[i], entries);
            }
        }
    }
}

Array *traverse_trie(const Trie *trie) {
    Array *entries = new_array(NULL);
    traverse_trie_nodes(trie->root, entries);
    return entries;
}
```

C.14 src/core/matcher.h

```
#ifndef CORE_TRIE_H_
#define CORE_TRIE_H_

#include "core/trie_node.h"

typedef struct Trie {
    TrieNode *root;
    int size;
} Trie;

Trie *new_trie();
void delete_trie(void *trie);

```

```c
Array *trie_search(const Trie *trie, const String *word, bool
    case_sensitive);
void trie_insert(Trie *trie, DictEntry *entry);
void trie_remove(Trie *trie, const String *word, bool case_sensitive);

String *trie_predecessor(const Trie *trie, const String *word);
String *trie_successor(const Trie *trie, const String *word);

Array *traverse_trie(const Trie *trie);

#endif // CORE_TRIE_H_
```

C.15 src/core/matcher.c

```c
#include "core/trie.h"

Trie *new_trie() {
    Trie *trie = malloc(sizeof(Trie));
    trie->root = new_trie_node('\0', NULL);
    trie->size = 0;
    return trie;
}

static void clear_trie_nodes(TrieNode *root) {
    assert(root && "clear_trie_nodes: root node is null");
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                clear_trie_nodes(root->children[i]);
            }
        }
    }
    delete_trie_node(root);
}

void delete_trie(void *p) {
    Trie *trie = p;
    clear_trie_nodes(trie->root);
    free(trie);
}

static TrieNode *get_trie_node(const Trie *trie, const String *word) {
    TrieNode *node = trie->root;
    for (int i = 0; i < word->size; ++i) {
        int index = char_to_index(tolower(word->text[i]));
        node = node->children[index];
        if (!node) {
            return NULL;
        }
    }
    return node;
}

Array *trie_search(const Trie *trie, const String *word, bool
    case_sensitive) {
    Array *entries = new_array(NULL);
    if (!is_valid_key(word, false)) {
        WARN("Invalid headword: %s\n", word->text);
        return entries;
    }
    TrieNode *node = get_trie_node(trie, word);
    if (!node || node->entries->size <= 0) {
        return entries;
    }
    if (case_sensitive) {
        DictEntry *entry;
        for (int i = 0; i < node->entries->size; ++i) {
            entry = node->entries->data[i];
            if (!strcmp(entry->headword->text, word->text)) {
                array_append(entries, entry);
            }
        }
    } else {
        for (int i = 0; i < node->entries->size; ++i) {
            array_append(entries, node->entries->data[i]);
        }
    }
    return entries;
}

void trie_insert(Trie *trie, DictEntry *entry) {
    assert(is_valid_key(entry->headword, false) &&
           "trie_insert: invalid entry headword");
    String *lowered = to_lower(entry->headword);
    TrieNode *node = trie->root;
    for (int i = 0; i < lowered->size; ++i) {
        int index = char_to_index(lowered->text[i]);
        if (!node->children[index]) {
            node = trie_node_add_child(node, lowered->text[i]);
        } else {
            node = node->children[index];
        }
    }
    delete_string(lowered);
    array_append(node->entries, entry);
    ++trie->size;
}

void trie_remove(Trie *trie, const String *word, bool case_sensitive) {
    assert(is_valid_key(word, false) && "trie_remove: invalid headword");
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        return;
    }
    if (case_sensitive) {
        int i = 0;
        DictEntry *entry;
        while (i < node->entries->size) {
            entry = node->entries->data[i];
            if (!strcmp(entry->headword->text, word->text)) {
                array_remove(node->entries, i);
                --trie->size;
            } else {
                ++i;
            }
        }
    } else {
        for (int i = node->entries->size - 1; i >= 0; --i) {
            array_remove(node->entries, i);
            --trie->size;
        }
    }
    char letter;
    while (node->entries->size <= 0 && node->child_count <= 0 && node->
    parent) {
        letter = node->letter;
        node = node->parent;
        trie_node_remove_child(node, letter);
    }
}

String *trie_predecessor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = previous_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = previous_trie_node(node);
    }
    return NULL;
}

String *trie_successor(const Trie *trie, const String *word) {
    TrieNode *node = get_trie_node(trie, word);
    if (!node) {
        WARN("Cannot find word: %s\n", word->text);
        return NULL;
    }
    node = next_trie_node(node);
    while (node) {
        if (node->entries->size > 0) {
            DictEntry *entry = node->entries->data[0];
            return to_lower(entry->headword);
        }
        node = next_trie_node(node);
    }
    return NULL;
}

static void traverse_trie_nodes(const TrieNode *root, Array *entries) {
    for (int i = 0; i < root->entries->size; ++i) {
        array_append(entries, root->entries->data[i]);
    }
    if (root->child_count > 0) {
        for (int i = 0; i < ALPHABET_SIZE; ++i) {
            if (root->children[i]) {
                traverse_trie_nodes(root->children[i], entries);
            }
        }
    }
}

Array *traverse_trie(const Trie *trie) {
    Array *entries = new_array(NULL);
    traverse_trie_nodes(trie->root, entries);
    return entries;
}
```

C.16 src/test/global.h

```
#ifndef TEST_GLOBAL_H_
#define TEST_GLOBAL_H_

#include "core/array.h"
#include "core/dict_entry.h"
#include "core/matcher.h"
#include "core/string.h"
#include "core/trie.h"
#include "core/trie_node.h"

#define CHECK_BASE(equality, expect, actual, format)
       \
    do {
       \
        ++test_count;
       \
        if (equality) {
       \
            ++pass_count;
       \
        } else {
       \
            fprintf(stderr,
       \
                    "%s: %d\nexpect: " format "\nactual: " format "\n",
       \
                    __FILE__, __LINE__, expect, actual);
       \
        }
       \
    } while (false)

#define DISPLAY_TEST_RESULT()
       \
    printf("%d/%d (%g%%) passed\n", pass_count, test_count,
       \
           pass_count * 100.0 / test_count)

#define CHECK_BOOL(expect, actual)
```

```
        \
    CHECK_BASE((expect) == (actual), (expect) ? "true" : "false",
        \
               (actual) ? "true" : "false", "%s")

#define CHECK_INT(expect, actual)
        \
    CHECK_BASE((expect) == (actual), expect, actual, "%d")

#define CHECK_STRING(expect, actual)
        \
    CHECK_BASE(!strcmp(expect, (actual)->text), expect, (actual)->text, "%s
   ")

#endif // TEST_GLOBAL_H_
```

C.17 src/test/array.c

```
#include "test/global.h"

int test_count = 0, pass_count = 0;

#define N 50

int data[N];

void generate_data() {
    for (int i = 0; i < N; ++i) {
        data[i] = rand();
    }
}

void test_append() {
    generate_data();
    Array *array = new_array(free);
    int *element, *random;
    for (int i = 0; i < N; ++i) {
        element = malloc(sizeof(int));
        *element = data[i];
        random = malloc(sizeof(int));
        *random = rand();
        array_append(array, random);
        CHECK_INT(i + 1, array->size);
        array_append(array, element);
        CHECK_INT(i + 2, array->size);
        array_remove(array, i);
        CHECK_INT(i + 1, array->size);
    }
    for (int i = 0; i < N; ++i) {
        CHECK_INT(data[i], *(int *)(array->data[i]));
    }
    delete_array(array);
}

void test_remove() {
    generate_data();
    Array *array = new_array(free);
    int *element;
    for (int i = 0; i < N; ++i) {
        element = malloc(sizeof(int));
        *element = data[i];
        array_append(array, element);
    }
    for (int i = 0; i < N; ++i) {
        for (int j = i; j < N; ++j) {
            CHECK_INT(data[j], *(int *)(array->data[j - i]));
        }
        array_remove(array, 0);
    }
    CHECK_INT(0, array->size);
    delete_array(array);
}

void test_reserve() {
    generate_data();
    Array *array = new_array(free);
    array_reserve(array, N - 1);
    int *element;
    for (int i = 0; i < N; ++i) {
        element = malloc(sizeof(int));
        *element = data[i];
        CHECK_INT(N - 1, array->capacity);
        array_append(array, element);
    }
    CHECK_INT((N - 1) * 2, array->capacity);
    delete_array(array);
}

int main() {
    test_append();
```

```c
    test_remove();
    test_reserve();
    DISPLAY_TEST_RESULT();
}
```

C.18 src/test/string.c

```c
#include "test/global.h"

int test_count = 0, pass_count = 0;

#define N 10

char *texts[N] = {
    "pen pineapple apple pie",
    "  banana nanana  ",
    "iPhone XS Max",
    "\tIt was the age of wisdom.\tIt was the age of foolishness.\t",
    "gcc -g string.c -o string.o",
    "Engineering Science",
    " University of Toronto ",
    " \t \t \t \t ",
    "Designed by Apple in California",
    " ESC190 Teaching Team"};

String *strings[N];

void create_strings() {
    for (int i = 0; i < N; ++i) {
        strings[i] = new_string(texts[i], -1);
    }
}

void discard_strings() {
    for (int i = 0; i < N; ++i) {
        delete_string(strings[i]);
    }
}

char find_chars[N] = {'a', 'b', 'x', 'F', 'e', 'S', 'g', 'h', 'i', 't'};

int expect_indices[N] = {8, 2, 12, -1, -1, 12, -1, -1, 3, -1};

void test_index() {
    int index;
    create_strings();
    for (int i = 0; i < N; ++i) {
        index = string_index(strings[i], find_chars[i]);
        CHECK_INT(expect_indices[i], index);
    }
    discard_strings();
}

char *lower_expect[N] = {
    "pen pineapple apple pie",
    "  banana nanana  ",
    "iphone xs max",
    "\tit was the age of wisdom.\tit was the age of foolishness.\t",
    "gcc -g string.c -o string.o",
    "engineering science",
    " university of toronto ",
    " \t \t \t \t ",
    "designed by apple in california",
    " esc190 teaching team"};

void test_to_lower() {
    String *lowered;
    create_strings();
    for (int i = 0; i < N; ++i) {
        lowered = to_lower(strings[i]);
        CHECK_STRING(lower_expect[i], lowered);
        delete_string(lowered);
        to_lower_in_place(strings[i]);
        CHECK_STRING(lower_expect[i], strings[i]);
    }
    discard_strings();
}

char *trim_expect[N] = {
    "pen pineapple apple pie",
    "banana nanana",
    "iPhone XS Max",
    "It was the age of wisdom.\tIt was the age of foolishness.",
    "gcc -g string.c -o string.o",
    "Engineering Science",
    "University of Toronto",
    "",
    "Designed by Apple in California",
    "ESC190 Teaching Team"};

```

```c
void test_trim() {
    String *trimmed;
    create_strings();
    for (int i = 0; i < N; ++i) {
        trimmed = trim(strings[i]);
        CHECK_STRING(trim_expect[i], trimmed);
        delete_string(trimmed);
        trim_in_place(strings[i]);
        CHECK_STRING(trim_expect[i], strings[i]);
    }
    discard_strings();
}

int substring_indices[N][2] = {{4, 13}, {9, 15},  {7, 9}, {19, 25}, {7,
   15},
                               {0, 11}, {15, 22}, {5, 8}, {9, 17},  {6,
   16}};

char *substring_expect[N] = {"pineapple", "nanana",      "XS",      "wisdom
   ",
                             "string.c",  "Engineering", "Toronto", "\t \t"
   ,
                             "by Apple",  "0 Teaching"};

void test_substring() {
    String *result;
    create_strings();
    for (int i = 0; i < N; ++i) {
        result = substring(strings[i], substring_indices[i][0],
                           substring_indices[i][1]);
        CHECK_STRING(substring_expect[i], result);
        delete_string(result);
        substring_in_place(strings[i], substring_indices[i][0],
                           substring_indices[i][1]);
        CHECK_STRING(substring_expect[i], strings[i]);
    }
    discard_strings();
}

int split_counts[N] = {4, 2, 3, 12, 5, 2, 3, 0, 5, 3};

char *split_fields[N][15] = {{"pen", "pineapple", "apple", "pie"},
                             {"banana", "nanana"},
                             {"iPhone", "XS", "Max"},
                             {"It", "was", "the", "age", "of", "wisdom.", "
   It",
                              "was", "the", "age", "of", "foolishness."},
                             {"gcc", "-g", "string.c", "-o", "string.o"},
                             {"Engineering", "Science"},
                             {"University", "of", "Toronto"},
                             {},
                             {"Designed", "by", "Apple", "in", "California"
   },
                             {"ESC190", "Teaching", "Team"}};

void test_split() {
    Array *array;
    create_strings();
    for (int i = 0; i < N; ++i) {
        array = split_string(strings[i]);
        CHECK_INT(split_counts[i], array->size);
        if (split_counts[i] != array->size) {
            continue;
        }
        for (int j = 0; j < split_counts[i]; ++j) {
            CHECK_STRING(split_fields[i][j], (String *)(array->data[j]));
        }
        delete_array(array);
    }
    discard_strings();
}

int join_counts[N] = {4, 2, 3, 12, 5, 2, 3, 1, 5, 3};

char *join_fields[N][15] = {{"pen", "pineapple", "apple", "pie"},
                            {"banana", "nanana"},
                            {"iPhone", "XS", "Max"},
                            {"It", "was", "the", "age", "of", "wisdom.", "
   It",
                             "was", "the", "age", "of", "foolishness."},
                            {"gcc", "-g", "string.c", "-o", "string.o"},
                            {"Engineering", "Science"},
                            {"University", "of", "Toronto"},
                            {"t"},
                            {"Designed", "by", "Apple", "in", "California"
   },
                            {"ESC190", "Teaching", "Team"}};

char *join_expect[N] = {
    "pen pineapple apple pie",
```

```c
    "banana nanana",
    "iPhone XS Max",
    "It was the age of wisdom. It was the age of foolishness.",
    "gcc -g string.c -o string.o",
    "Engineering Science",
    "University of Toronto",
    "t",
    "Designed by Apple in California",
    "ESC190 Teaching Team"};

void test_join() {
    Array *strings;
    String *field, *joined;
    for (int i = 0; i < N; ++i) {
        strings = new_array(delete_string);
        for (int j = 0; j < join_counts[i]; ++j) {
            field = new_string(join_fields[i][j], -1);
            array_append(strings, field);
        }
        joined = join_strings(strings, ' ');
        CHECK_STRING(join_expect[i], joined);
        delete_string(joined);
        delete_array(strings);
    }
}

const char *prefixes[N] = {"pen",         "  banana", "iPhone XS Max Pro",
                           "",            "gcc -g",   "Engineering Science",
                           "University", " \t",      "Designed by",
                           "ESC190 "};

bool start_expect[N] = {true, true,  false, true, true,
                        true, false, true,  true, false};

void test_start_with() {
    String *string;
    bool start;
    for (int i = 0; i < N; ++i) {
        string = new_string(texts[i], -1);
        start = string_start_with(string, prefixes[i]);
        delete_string(string);
        CHECK_BOOL(start_expect[i], start);
    }
}

```

```c
int main() {
    test_index();
    test_to_lower();
    test_substring();
    test_trim();
    test_split();
    test_join();
    test_start_with();
    DISPLAY_TEST_RESULT();
}
```

C.19 src/test/dict_entry.c

```c
#include "test/global.h"

int test_count = 0, pass_count = 0;

#define N 15

const char *lines[N] = {
    " Apple pie  ( n. )  A kind of food that is made of apple. ",
    "\"Banana ((n.) A long and yellow fruit.\"",
    "California (n.) A state of USA.",
    "\" Drag  ( v. ))  To pull forcefully. \"",
    "Delta-Epsilon (n.) A mathematical language that defines derivatives.",
    "\"Fine adj. Good.\"",
    "Gay (adj.) Happy and joyful.",
    "\"Hike     (   v.   )     Travel on foot.\"",
    " University of Toronto  ( n. )  A university in Ontario. ",
    "\"Iceberg (n.) A big piece of ice that floats on the sea.\"",
    " \"a ( ( b ) ) c\"",
    "a ( ( b ) ( ) ))  c",
    "a ( ( ( b ) ) c",
    "\" a b c () \"",
    "a b () b c c () \""};

bool expect_success[N] = {
    true, false, true, true, true,  false, true, true,
    true, true,  true, true, false, true,  true,
};

const char *expect_entries[N][3] = {
    {"Apple pie", "n.", "A kind of food that is made of apple."},
    {},
    {"California", "n.", "A state of USA."},
    {"Drag", "v.", ")  To pull forcefully."},
    {"Delta-Epsilon", "n.",
     "A mathematical language that defines derivatives."},
    {},
    {"Gay", "adj.", "Happy and joyful."},
    {"Hike", "v.", "Travel on foot."},
    {"University of Toronto", "n.", "A university in Ontario."},
    {"Iceberg", "n.", "A big piece of ice that floats on the sea."},
    {"\"a", "( b )", "c\""},
    {"a", "( b ) ( )", ")  c"},
    {},
    {"a b c", "", ""},
    {"a b", "", "b c c () \""}};

void test_new_dict_entry() {
    DictEntry *entry;
    for (int i = 0; i < N; ++i) {
        String *line = new_string(lines[i], -1);
        DictEntry *entry = new_dict_entry(line);
        delete_string(line);
        CHECK_BOOL(expect_success[i], (bool)entry);
        if (expect_success[i] && entry) {
            CHECK_STRING(expect_entries[i][0], entry->headword);
            CHECK_STRING(expect_entries[i][1], entry->word_class);
            CHECK_STRING(expect_entries[i][2], entry->definition);
        }
        if (entry) {
            delete_dict_entry(entry);
        }
    }
}

int main() {
    test_new_dict_entry();
    DISPLAY_TEST_RESULT();
}
```

C.20 src/test/trie.c

```c
#include "test/global.h"

int test_count = 0, pass_count = 0;

Trie *dictionary;

```

```c
void create_dictionary() {
    dictionary = new_trie();
    char file_name[] = "../Dictionary-in-csv/*.csv";
    int index = strchr(file_name, '*') - file_name;
    FILE *stream;
    String *line;
    DictEntry *entry;
    for (char c = 'A'; c <= 'Z'; ++c) {
        file_name[index] = c;
        stream = fopen(file_name, "r");
        if (!stream) {
            WARN("Cannot open file: %s\n", file_name);
            continue;
        }
        line = get_line(stream);
        while (line) {
            if (line->size > 0) {
                entry = new_dict_entry(line);
                if (!entry) {
                    WARN("Failed to parse the following line in %s:\n%s\n",
                            file_name, line->text);
                } else {
                    trie_insert(dictionary, entry);
                }
            }
            delete_string(line);
            line = get_line(stream);
        }
        fclose(stream);
    }
}

void save_dictionary(const char *file_name) {
    Array *entries = traverse_trie(dictionary);
    CHECK_INT(dictionary->size, entries->size);
    FILE *stream = fopen(file_name, "wb");
    for (int i = 0; i < entries->size; ++i) {
        write_dict_entry((DictEntry *)(entries->data[i]), stream);
    }
    fclose(stream);
    delete_array(entries);
}

void test_search() {
    Array *entries = traverse_trie(dictionary), *results;
    DictEntry *entry;
    for (int i = 0; i < entries->size; ++i) {
        entry = entries->data[i];
        results = trie_search(dictionary, entry->headword, false);
        CHECK_BOOL(true, results->size > 0);
        delete_array(results);
    }
    delete_array(entries);
}

void test_predecessor() {
    Array *entries = traverse_trie(dictionary);
    String *s1, *s2;
    s1 = to_lower(((DictEntry *)(entries->data[entries->size - 1]))->
   headword);
    int entry_count = 0;
    Array *results;
    while (true) {
        results = trie_search(dictionary, s1, false);
        entry_count += results->size;
        delete_array(results);
        s2 = trie_predecessor(dictionary, s1);
        if (!s2) {
            delete_string(s1);
            break;
        }
        CHECK_BOOL(true, strcmp(s1->text, s2->text) > 0);
        delete_string(s1);
        s1 = s2;
    }
    delete_array(entries);
    CHECK_INT(dictionary->size, entry_count);
}

void test_successor() {
    Array *entries = traverse_trie(dictionary);
    String *s1, *s2;
    s1 = to_lower(((DictEntry *)(entries->data[0]))->headword);
    int entry_count = 0;
    Array *results;
    while (true) {
        results = trie_search(dictionary, s1, false);
        entry_count += results->size;
        delete_array(results);
        s2 = trie_successor(dictionary, s1);
```

```c
        if (!s2) {
            delete_string(s1);
            break;
        }
        CHECK_BOOL(true, strcmp(s1->text, s2->text) < 0);
        delete_string(s1);
        s1 = s2;
    }
    delete_array(entries);
    CHECK_INT(dictionary->size, entry_count);
}

void test_remove() {
    Array *entries = traverse_trie(dictionary);
    Array *headwords = new_array(delete_string);
    String *headword;
    for (int i = 0; i < entries->size; ++i) {
        headword = ((DictEntry *)(entries->data[i]))->headword;
        array_append(headwords, new_string(headword->text, headword->size))
    ;
    }
    delete_array(entries);
    for (int i = 0; i < headwords->size; ++i) {
        trie_remove(dictionary, (String *)(headwords->data[i]), true);
    }
    CHECK_INT(0, dictionary->size);
    delete_array(headwords);
}

int main() {
    create_dictionary();
    save_dictionary("rewrite_dictionary.txt");
    test_search();
    test_predecessor();
    test_successor();
    test_remove();
    delete_trie(dictionary);
    DISPLAY_TEST_RESULT();
}
```

C.21 src/test/matcher.c

```c
#include "test/global.h"

int test_count = 0, pass_count = 0;
```

```c

Trie *dictionary;

// Copy and paste from test/trie.c.
void create_dictionary() {
    dictionary = new_trie();
    char file_name[] = "../Dictionary-in-csv/*.csv";
    int index = strchr(file_name, '*') - file_name;
    FILE *stream;
    String *line;
    DictEntry *entry;
    for (char c = 'A'; c <= 'Z'; ++c) {
        file_name[index] = c;
        stream = fopen(file_name, "r");
        if (!stream) {
            WARN("Cannot open file: %s\n", file_name);
            continue;
        }
        line = get_line(stream);
        while (line) {
            if (line->size > 0) {
                entry = new_dict_entry(line);
                if (!entry) {
                    WARN("Failed to parse the following line in %s:\n%s\n",
                         file_name, line->text);
                } else {
                    trie_insert(dictionary, entry);
                }
            }
            delete_string(line);
            line = get_line(stream);
        }
        fclose(stream);
    }
}

void test_closest_match() {
    Array *entries = traverse_trie(dictionary);
    DictEntry *entry;
    String *lowered, *result;
    for (int i = 0; i < entries->size; ++i) {
        entry = entries->data[i];
        lowered = to_lower(entry->headword);
        result = trie_closest_match(dictionary, entry->headword, -1);
        CHECK_BOOL(true, result != NULL);
        if (result) {
            CHECK_STRING(lowered->text, result);
            delete_string(result);
        }
        delete_string(lowered);
    }
    delete_array(entries);
}

int main() {
    create_dictionary();
    test_closest_match();
    delete_trie(dictionary);
    DISPLAY_TEST_RESULT();
}
```

C.22 src/main/api.h

```c
#ifndef MAIN_API_H_
#define MAIN_API_H_

#include "core/array.h"
#include "core/dict_entry.h"
#include "core/matcher.h"
#include "core/string.h"
#include "core/trie.h"
#include "core/trie_node.h"

#ifndef ESP_RC_PATH
#define ESP_RC_PATH "../.esp_rc"
#endif // ESP_RC_PATH

typedef enum EspMode {
    ESP_MODE_INTERACTIVE,
    ESP_MODE_BACKGROUND,
    ESP_MODE_COMMAND_LINE
} EspMode;

void esp_initialize(EspMode mode);
void esp_cleanup(EspMode mode);

bool esp_parse_arguments(Array *arguments, EspMode mode);

void esp_on_load(Array *arguments, EspMode mode);
void esp_on_search(Array *arguments, EspMode mode);
void esp_on_insert(Array *arguments, EspMode mode);
void esp_on_remove(Array *arguments, EspMode mode);
void esp_on_neighbour(Array *arguments, EspMode mode);
void esp_on_prefix(Array *arguments, EspMode mode);
void esp_on_match(Array *arguments, EspMode mode);
void esp_on_size(Array *arguments, EspMode mode);
void esp_on_save(Array *arguments, EspMode mode);
bool esp_on_exit(Array *arguments, EspMode mode);

#endif // MAIN_API_H_
```

C.23 src/main/api.c

```c
#include "main/api.h"
#include "main/utility.h"

Trie *dictionary;

void esp_initialize(EspMode mode) {
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("
   *============================================*\n"
               "               ||  EngSci Press Dictionary by Yunhao Qian
   ||\n"
               "
   *============================================*\n"
               "\nStarting...\n\n");
    }
    dictionary = new_trie();
    FILE *stream = fopen(ESP_RC_PATH, "r");
    if (stream) {
        String *line = get_line(stream);
        Array *arguments;
        while (line) {
            arguments = split_string(line);
            esp_parse_arguments(arguments, ESP_MODE_BACKGROUND);
            delete_array(arguments);
            delete_string(line);
            line = get_line(stream);
        }
        fclose(stream);
    }
}

void esp_cleanup(EspMode mode) {
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("\nExiting...\n");
    }
    delete_trie(dictionary);
}

bool esp_parse_arguments(Array *arguments, EspMode mode) {
    if (arguments->size <= 0) {
        return true;
    }
    String *leading = to_lower(arguments->data[0]);
    array_remove(arguments, 0);
    bool returned = true;
    if (!strcmp(leading->text, "load")) {
        esp_on_load(arguments, mode);
    } else if (!strcmp(leading->text, "search")) {
        esp_on_search(arguments, mode);
    } else if (!strcmp(leading->text, "insert")) {
        esp_on_insert(arguments, mode);
    } else if (!strcmp(leading->text, "remove")) {
        esp_on_remove(arguments, mode);
    } else if (!strcmp(leading->text, "neighbour")) {
        esp_on_neighbour(arguments, mode);
    } else if (!strcmp(leading->text, "prefix")) {
        esp_on_prefix(arguments, mode);
    } else if (!strcmp(leading->text, "match")) {
        esp_on_match(arguments, mode);
    } else if (!strcmp(leading->text, "size")) {
        esp_on_size(arguments, mode);
    } else if (!strcmp(leading->text, "save")) {
        esp_on_save(arguments, mode);
    } else if (!strcmp(leading->text, "exit")) {
        returned = esp_on_exit(arguments, mode);
    } else {
        WARN("Unknown leading argument: %s\n", leading->text);
    }
    delete_string(leading);
    return returned;
}

void esp_on_load(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_COMMAND_LINE) {
        WARN_NOT_SUPPORTED("load", "command-line");
        return;
    }
```

```c
    if (arguments->size < 1) {
        WARN_MISSING("file name");
        return;
    }
    if (arguments->size > 1) {
        WARN_REDUNDANT(arguments, 1);
    }
    const char *file_name = ((String *)arguments->data[0])->text;
    FILE *stream = fopen(file_name, "r");
    if (!stream) {
        WARN("Cannot open file: %s\nDo nothing.\n", file_name);
        return;
    }
    String *line = get_line(stream);
    DictEntry *entry;
    int count = 0;
    while (line) {
        if (line->size > 0) {
            entry = new_dict_entry(line);
            if (!entry) {
                WARN("Failed to parse the following line in %s:\n%s\n",
                     file_name, line->text);
            } else {
                trie_insert(dictionary, entry);
                ++count;
            }
        }
        delete_string(line);
        line = get_line(stream);
    }
    fclose(stream);
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("%d entries loaded from %s\n", count, file_name);
    }
}

void esp_on_search(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("search", "background");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    String *word = join_strings(arguments, ' ');
    bool case_sensitive = false;
    for (int i = 0; i < word->size; ++i) {
        if (isupper(word->text[i])) {
            case_sensitive = true;
            break;
        }
    }
    Array *results = trie_search(dictionary, word, case_sensitive);
    if (results->size <= 0) {
        WARN("Find no entry named: %s\n", word->text);
        word_hint(word, dictionary, case_sensitive);
    } else {
        for (int i = 0; i < results->size; ++i) {
            putchar('\n');
            display_dict_entry(results->data[i]);
        }
        putchar('\n');
    }
    delete_array(results);
    delete_string(word);
}

void esp_on_insert(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("insert", "background");
        return;
    }
    if (mode == ESP_MODE_COMMAND_LINE) {
        WARN_NOT_SUPPORTED("insert", "command-line");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    String *headword = join_strings(arguments, ' ');
    DictEntry *entry = input_dict_entry(headword);
    if (entry) {
        trie_insert(dictionary, entry);
    }
    delete_string(headword);
}

void esp_on_remove(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_COMMAND_LINE) {
        WARN_NOT_SUPPORTED("remove", "command-line");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    String *headword = join_strings(arguments, ' ');
    bool case_sensitive = false;
    for (int i = 0; i < headword->size; ++i) {
        if (isupper(headword->text[i])) {
            case_sensitive = true;
            break;
        }
    }
    Array *results = trie_search(dictionary, headword, case_sensitive);
    int remove_count = results->size;
    bool shall_remove = true;
    if (remove_count <= 0) {
        WARN("Find no entry named: %s\n", headword->text);
        word_hint(headword, dictionary, case_sensitive);
        shall_remove = false;
    } else if (mode == ESP_MODE_INTERACTIVE) {
        printf("The following entries will be removed:\n");
        for (int i = 0; i < remove_count; ++i) {
            putchar('\n');
            display_dict_entry(results->data[i]);
        }
        shall_remove = confirm(true, "\nWant to continue?");
    }
    delete_array(results);
    if (shall_remove) {
        trie_remove(dictionary, headword, case_sensitive);
        if (mode == ESP_MODE_INTERACTIVE) {
            printf("%d entries removed.\n", remove_count);
        }
    } else if (mode == ESP_MODE_INTERACTIVE) {
        printf("Do nothing.\n");
    }
}

void esp_on_neighbour(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("neighbour", "background");
```

```c
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    int radius = 10;
    if (string_start_with(arguments->data[0], "--")) {
        if (arguments->size <= 1) {
            WARN_MISSING("headword");
            return;
        }
        int number = parse_unsigned_int_flag(arguments->data[0]);
        if (number < 0) {
            WARN("Invalid flag: %s\n", ((String *)arguments->data[0])->text
    );
        } else {
            radius = number;
        }
        array_remove(arguments, 0);
    }
    String *word = join_strings(arguments, ' ');
    delete_string(word);
}

void esp_on_prefix(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("prefix", "background");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("prefix string");
        return;
    }
    int max_count = 10;
    if (string_start_with(arguments->data[0], "--")) {
        if (arguments->size <= 1) {
            WARN_MISSING("prefix string");
            return;
        }
        int number = parse_unsigned_int_flag(arguments->data[0]);
        if (number < 0) {
            WARN("Invalid flag: %s\n", ((String *)arguments->data[0])->text
    );
        } else {
            max_count = number;
        }
        array_remove(arguments, 0);
    }
    String *prefix = join_strings(arguments, ' ');
    delete_string(prefix);
}

void esp_on_match(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("match", "background");
        return;
    }
    if (arguments->size <= 0) {
        WARN_MISSING("headword");
        return;
    }
    int tolerance = -1;
    if (string_start_with(arguments->data[0], "--")) {
        if (arguments->size <= 1) {
            WARN_MISSING("headword");
            return;
        }
        int number = parse_unsigned_int_flag(arguments->data[0]);
        if (number < 0) {
            WARN("Invalid flag: %s\n", ((String *)arguments->data[0])->text
   );
        } else {
            tolerance = number;
        }
        array_remove(arguments, 0);
    }
    String *pattern = join_strings(arguments, ' ');
    String *matched = trie_closest_match(dictionary, pattern, tolerance);
    if (!matched) {
        WARN("Find no entry similar to: %s\n", pattern->text);
    } else {
        printf("%s\n", matched->text);
        delete_string(matched);
    }
    delete_string(pattern);
}

void esp_on_size(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("size", "background");
        return;
    }
    if (arguments->size > 0) {
        WARN_REDUNDANT(arguments, 0);
    }
    printf("Dictionary size: %d\n", dictionary->size);
}

void esp_on_save(Array *arguments, EspMode mode) {
    if (arguments->size < 1) {
        WARN_MISSING("file name");
        return;
    }
    if (arguments->size > 1) {
        WARN_REDUNDANT(arguments, 1);
    }
    if (mode == ESP_MODE_INTERACTIVE && dictionary->size <= 0 &&
        !confirm(false, "The dictionary is empty. Continue?")) {
        printf("Do nothing.\n");
        return;
    }
    const char *file_name = ((String *)arguments->data[0])->text;
    FILE *stream = fopen(file_name, "wb");
    if (!stream) {
        WARN("Cannot open file: %s\nDo nothing.\n", file_name);
        return;
    }
    Array *entries = traverse_trie(dictionary);
    for (int i = 0; i < entries->size; ++i) {
        write_dict_entry(entries->data[i], stream);
    }
    delete_array(entries);
    fclose(stream);
    if (mode == ESP_MODE_INTERACTIVE) {
        printf("%d entries saved to %s.\n", dictionary->size, file_name);
    }
}

bool esp_on_exit(Array *arguments, EspMode mode) {
    if (mode == ESP_MODE_BACKGROUND) {
        WARN_NOT_SUPPORTED("exit", "background");
        return true;
    }
    if (mode == ESP_MODE_COMMAND_LINE) {
```

```c
        WARN_NOT_SUPPORTED("exit", "command-line");
        return true;
    }
    if (arguments->size > 0) {
        WARN_REDUNDANT(arguments, 0);
    }
    return !confirm(true, "Are you sure you want to exit?");
}
```

C.24 src/main/utility.h

```c
#ifndef MAIN_UTILITY_H_
#define MAIN_UTILITY_H_

#include "main/api.h"

#define WARN_NOT_SUPPORTED(argument, mode)                                  \
    WARN("Does not support \"%s\" in %s mode.\n", argument, mode);

#define WARN_MISSING(expected)                                              \
    WARN("Missing argument: %s expected.\n", expected)

#define WARN_REDUNDANT(arguments, start_index)                              \
    WARN("Redundant arguments: ignore arguments since \"%s\".\n",           \
         ((String *)(arguments)->data[start_index])->text)

int parse_unsigned_int_flag(const String *string);

void word_hint(const String *string, const Trie *dictionary,
               bool case_sensitive);

#endif // MAIN_UTILITY_H_
```

C.25 src/main/utility.c

```c
#include "main/utility.h"

int parse_unsigned_int_flag(const String *string) {
    assert(string_start_with(string, "--") &&
           "parse_unsigned_int_flag: not a flag");
    if (string->size <= 2) {
        return -1;
    }
    String *flag = substring(string, 2, string->size);
    for (int i = 0; i < flag->size; ++i) {
        if (!isdigit(flag->text[i])) {
            delete_string(flag);
            return -1;
        }
    }
    int number = atoi(flag->text);
    delete_string(flag);
    return number;
}

void word_hint(const String *string, const Trie *dictionary,
               bool case_sensitive) {
    String *matched = trie_closest_match(dictionary, string, -1);
    if (!matched) {
        return;
    }
    if (case_sensitive) {
        String *lowered = to_lower(string);
        if (!strcmp(matched->text, lowered->text)) {
            printf("Tip: use lower-case word for case-insensitive "
                   "search/remove.\n");
            delete_string(lowered);
            return;
        }
        delete_string(lowered);
    }
    printf("Did you mean: %s\n", matched->text);
    delete_string(matched);
}
```

C.26 src/main/main.c

```c
#include "main/api.h"

int main(int argc, const char **argv) {
    Array *arguments;
    if (argc > 1) {
        arguments = new_array(delete_string);
        for (int i = 1; i < argc; ++i) {
            array_append(arguments, new_string(argv[i], -1));
        }
        esp_initialize(ESP_MODE_COMMAND_LINE);
        esp_parse_arguments(arguments, ESP_MODE_COMMAND_LINE);
        delete_array(arguments);
        esp_cleanup(ESP_MODE_COMMAND_LINE);
        return 0;
    }
    esp_initialize(ESP_MODE_INTERACTIVE);
    String *line;
    bool shall_continue;
    do {
        printf(">>> ");
        line = get_line(stdin);
        if (!line) {
            break;
        }
        arguments = split_string(line);
        shall_continue = esp_parse_arguments(arguments, ESP_MODE_INTERACTIVE);
        delete_array(arguments);
        delete_string(line);
    } while (shall_continue);
    esp_cleanup(ESP_MODE_INTERACTIVE);
}
```

C.27 src/writer/grammar.py

```
from random import choices, random
from re import compile, match


float_pattern = compile(r'(([0-9]*\.)?[0-9]+)\s*\:\s+')
token_pattern = compile(
    r'(([^\s\"\(\)]|(\(([^\(\)]|(\"([^\"]|\\\")+\"))+\))|(\"([^\"]|\\\")*\"))+)\s*')


class Token:

    __slots__ = 'identifier', 'terminal', 'optional', 'probability'

    def __init__(self, string):
        self.terminal = False
        self.optional = False
        self.probability = 1
        if string.startswith('(') and string.endswith(')'):
```

```python
            string = string[1:-1]
            matched = float_pattern.match(string)
            factor = 1
            if matched:
                string = string[matched.end():]
                factor = float(matched.group(1))
            self.__init__(string)
            self.optional = True
            self.probability *= factor
        elif string.startswith('"') and string.endswith('"'):
            self.identifier = string[1:-1].replace('\\"', '"')
            self.terminal = True
        else:
            self.identifier = string

    def __eq__(self, other):
        if not isinstance(other, Token):
            return False
        return self.identifier == other.identifier and \
            self.terminal == other.terminal

    def __hash__(self):
        return hash((self.identifier, self.terminal))

    def __str__(self):
        string = self.identifier.replace('"', '\\"')
        if self.terminal:
            string = '"{}"'.format(string)
        if self.optional:
            if self.probability == 1:
                string = '({})'.format(string)
            else:
                string = '({}: {})'.format(self.probability, string)
        return string


class Rule:

    __slots__ = 'lhs', 'rhs', 'weight'

    def __init__(self, string):
        matched = float_pattern.match(string)
        if matched:
            string = string[matched.end():]
            self.weight = float(matched.group(1))
        else:
            self.weight = 1
        matched = token_pattern.match(string)
        string = string[matched.end():]
        self.lhs = Token(matched.group(1))
        string = string[match(r'->\s*', string).end():]
        self.rhs = []
        while True:
            matched = token_pattern.match(string)
            if not matched:
                break
            string = string[matched.end():]
            self.rhs.append(Token(matched.group(1)))

    def __eq__(self, other):
        if not isinstance(other, Rule):
            return False
        return self.lhs, tuple(self.rhs), self.weight == \
            other.lhs, tuple(other.rhs), other.weight

    def __hash__(self):
        return hash((self.lhs, tuple(self.rhs), self.weight))

    def __str__(self):
        elements = []
        if self.weight != 1:
            elements.append(str(self.weight) + ':')
        elements += [str(self.lhs), '->']
        for element in self.rhs:
            elements.append(str(element))
        return ' '.join(elements)


class CFG:

    __slots__ = 'rules', 'convergence'

    def __init__(self, string=None):
        self.rules = {}
        self.convergence = 1
        if string:
            self.load_lines(string)

    def __str__(self):
        elements = []
        if self.convergence != 1:
            elements.append('convergence = {}'.format(self.convergence))
        for rule_list in self.rules.values():
            for rule in rule_list:
                elements.append(str(rule))
        return '\n'.join(elements)

    def load_line(self, line):
        line = line.strip()
        if line == '' or line.startswith('//'):
            return
        matched = match(r'convergence\s*=\s*(([0-9]*\.)?[0-9]+)\s*', line)
        if matched:
            self.convergence = float(matched.group(1))
            return
        rule = Rule(line)
        if rule.lhs in self.rules:
            self.rules[rule.lhs].append(rule)
        else:
            self.rules[rule.lhs] = [rule]

    def load_lines(self, string):
        for line in string.split('\n'):
            try:
                self.load_line(line)
            except:
                print('Failed to parse line: {}'.format(line))

    def generate(self, start=Token('S'), max_length=-1):
        weight_dict = {}
        for rule_list in self.rules.values():
            for rule in rule_list:
                weight_dict[rule] = rule.weight
        stack = [start]
        terminals = []
        while len(stack) > 0:
            token = stack.pop()
            if token.optional and random() > token.probability:
                continue
            if token.terminal:
                terminals.append(token.identifier)
                continue
            try:
                rule_list = self.rules[token]
            except:
                raise Exception('Failed to find rule for: {}'.format(token))
            weights = [weight_dict[rule] for rule in rule_list]
            rule = choices(rule_list, weights, k=1)[0]
            weight_dict[rule] *= self.convergence
            stack += rule.rhs[::-1]
            if max_length > 0 and len(terminals) > max_length:
                raise Exception('Exceed max length: {}'.format(max_length))
        return terminals


demo_grammar = '''
convergence = 0.3

0.9: S -> Clause "."
0.1: S -> Clause "while" S

Clause -> NP VP

0.9: NP -> Det (0.6: Adj) N
0.1: NP -> NP "and" NP

VP -> V NP
VP -> V

Det -> "a"
Det -> "the"

Adj -> "smart"
Adj -> "tired"
Adj -> "brown"

N -> "student"
N -> "laptop"
N -> "car"

V -> "drives"
V -> "walks"
V -> "leaves"
'''


if __name__ == '__main__':
    cfg = CFG(demo_grammar)
    print(cfg)
    while True:
        input('=' * 80)
        try:
            tokens = cfg.generate(max_length=30)
        except Exception as exception:
            print(exception)
        else:
            tokens[0] = tokens[0].capitalize()
            print(' '.join(tokens[:-1]) + tokens[-1])
```

C.28 src/writer/espg_base.txt

```
// ===== Sentence =====
S -> NP-Sg VP-Sg "."
S -> NP-Pl VP-Pl "."
0.4: S -> VP-Pl "!"
0.2: S -> Aux-Sg NP-Sg VP-Pl "?"
0.2: S -> Aux-Pl NP-Pl VP-Pl "?"
0.1: S -> Wh-NP-Sg Aux-Sg NP-Sg VP-Pl "?"
0.1: S -> Wh-NP-Pl Aux-Pl NP-Pl VP-Pl "?"

// ===== Noun Phrase =====
0.2: NP-Sg -> Pronoun-Sg
// NP-Sg -> Proper-Noun-Sg
NP-Sg -> Det-Sg (0.5: AP) Nominal-Sg
0.2: NP-Pl -> Pronoun-Pl
// NP-Pl -> Proper-Noun-Pl
NP-Pl -> Det-Pl (0.5: AP) Nominal-Pl

// ===== Nominal =====
Nominal-Sg -> Noun-Sg
0.3: Nominal-Sg -> Nominal-Sg PP
0.3: Nominal-Sg -> Nominal-Sg Gerund-VP
0.3: Nominal-Sg -> Nominal-Sg Rel-Clause-Sg
Nominal-Pl -> Noun-Pl
0.3: Nominal-Pl -> Nominal-Pl PP
0.3: Nominal-Pl -> Nominal-Pl Gerund-VP
0.3: Nominal-Pl -> Nominal-Pl Rel-Clause-Pl

// ===== Gerundive Verb =====
Gerund-VP -> Gerund-V
Gerund-VP -> Gerund-V NP-Sg
Gerund-VP -> Gerund-V NP-Pl
Gerund-VP -> Gerund-V PP
Gerund-VP -> Gerund-V NP-Sg PP
```

```
Gerund-VP -> Gerund-V NP-Pl PP

// ===== Relative Clause =====
Rel-Clause-Sg -> Rel-Pronoun VP-Sg
Rel-Clause-Pl -> Rel-Pronoun VP-Pl

// ===== Verb Phrase =====
VP-Sg -> Verb-I-Sg
VP-Sg -> Verb-T-Sg NP-Sg
VP-Sg -> Verb-T-Sg NP-Pl
VP-Sg -> Verb-T-Sg NP-Sg PP
VP-Sg -> Verb-T-Sg NP-Pl PP
VP-Sg -> Verb-I-Sg PP
VP-Pl -> Verb-I-Pl
VP-Pl -> Verb-T-Pl NP-Sg
VP-Pl -> Verb-T-Pl NP-Pl
VP-Pl -> Verb-T-Pl NP-Sg PP
VP-Pl -> Verb-T-Pl NP-Pl PP
VP-Pl -> Verb-I-Pl PP

// ===== Adjective Phrase =====
AP -> Adj
0.2: AP -> Adv AP

// ===== Prepositional Phrase =====
PP -> Preposition NP-Sg
PP -> Preposition NP-Pl

// ===== Determiner =====
5: Det-Sg -> "the"
5: Det-Pl -> "the"
5: Det-Sg -> "a"
4: Deg-Sg -> "this"
4: Deg-Sg -> "that"
4: Det-Pl -> "these"
4: Det-Pl -> "those"
Det-Sg -> "my"
Det-Pl -> "my"
Det-Sg -> "your"
Det-Pl -> "your"
Det-Sg -> "his"
Det-Pl -> "his"
Det-Sg -> "her"
Deg-Pl -> "her"
Det-Sg -> "its"
```

```
Det-Pl -> "its"
Det-Sg -> "our"
Det-Pl -> "our"
Det-Sg -> "their"
Det-Pl -> "their"
2: Det-Pl -> "a" "few"
2: Det-Pl -> "many"
2: Det-Pl -> "a" "lot" "of"
3: Det-Pl -> "some"
Det-Sg -> "any"
Det-Sg -> "one"
Det-Pl -> "all"
Det-Sg -> "each"
Det-Sg -> "every"
Det-Sg -> "another"
Det-Sg -> NP-Sg "'s"
Det-Pl -> NP-Sg "'s"

// ===== Auxiliary Verb =====
Aux-Sg -> "has"
Aux-Pl -> "have"
Aux-Sg -> "had"
Aux-Pl -> "had"
Aux-Sg -> "did"
Aux-Pl -> "did"
Aux-Sg -> "will"
Aux-Pl -> "will"
Aux-Sg -> "should"
Aux-Pl -> "should"
Aux-Sg -> "would"
Aux-Pl -> "would"
Aux-Sg -> "may"
Aux-Pl -> "may"
Aux-Sg -> "might"
Aux-Pl -> "might"
Aux-Sg -> "must"
Aux-Pl -> "must"
Aux-Sg -> "can"
Aux-Pl -> "can"
Aux-Sg -> "could"
Aux-Pl -> "could"
Aux-Sg -> "does"
Aux-Pl -> "do"
Aux-Sg -> "need"
Aux-Pl -> "need"
```

```

// ===== Wh- Noun Phrase ====
Wh-NP-Sg -> "when"
Wh-NP-Pl -> "when"
Wh-NP-Sg -> "who"
Wh-NP-Pl -> "who"
Wh-NP-Sg -> "where"
Wh-NP-Pl -> "where"
Wh-NP-Sg -> "what"
Wh-NP-Pl -> "what"
Wh-NP-Sg -> "what" Noun-Sg
Wh-NP-Pl -> "what" Noun-Pl
Wh-NP-Sg -> "whose" Noun-Sg
Wh-NP-Pl -> "whose" Noun-Pl
Wh-NP-Sg -> "which" Noun-Sg
Wh-NP-Pl -> "which" Noun-Pl

// ===== Pronoun =====
4: Pronoun-Pl -> "you"
Pronoun-Sg -> "yours"
Pronoun-Pl -> "yours"
2: Pronoun-Pl -> "yourself"
Pronoun-Sg -> "him"
Pronoun-Sg -> "his"
Pronoun-Pl -> "his"
2: Pronoun-Sg -> "himself"
Pronoun-Sg -> "her"
Pronoun-Sg -> "hers"
Pronoun-Pl -> "hers"
2: Pronoun-Sg -> "herself"
4: Pronoun-Sg -> "it"
Pronoun-Sg -> "its"
Pronoun-Pl -> "its"
Pronoun-Sg -> "itself"
4: Pronoun-Sg -> "ours"
4: Pronoun-Pl -> "ours"
2: Pronoun-Pl -> "ourself"
Pronoun-Sg -> "theirs"
Pronoun-Pl -> "theirs"
2: Pronoun-Pl -> "themselves"

// ===== Relative Pronoun =====
Rel-Pronoun -> "who"
Rel-Pronoun -> "which"
Rel-Pronoun -> "that"
```

```

// ===== In espg_lexicon.txt =====
// Noun-Sg
// Noun-Pl
// Gerund-V
// Verb-I-Sg
// Verb-T-Sg
// Verb-I-Pl
// Verb-T-Pl
// Preposition
// Adj
// Adv
```

C.29 src/writer/lexicon.py

```python
from collections import defaultdict


class DictEntry:

    __slots__ = 'headword', 'word_class'

    def __init__(self, headword, word_class):
        self.headword = headword
        self.word_class = word_class

    @staticmethod
    def from_line(line):
        if line.startswith('"') and line.endswith('"'):
            line = line[1:-1]
        left_index = line.index('(')
        if left_index < 0:
            raise Exception('missing word class', line)
        right_index = -1
        depth = 1
        for i in range(left_index + 1, len(line)):
            if line[i] == '(':
                depth += 1
            elif line[i] == ')':
                depth -= 1
                if depth == 0:
                    right_index = i
                    break
        if right_index < 0:
            raise Exception('mismatched brackets', line)
        headword = line[:left_index].strip()
        if headword == '':
            raise Exception('empty headword', line)
        word_class = line[left_index + 1:right_index].strip()
        return DictEntry(headword, word_class)


pos_tag_to_word_classes = {
    'Proper-Noun-Sg': set(),
    'Proper-Noun-Pl': set(),
    'Noun-Sg': {
        'n. & v',
        'n.& v.',
        'n & v.',
        'n. & v. t.',
        'n.',
        'n. sing & pl.',
        'a & n.',
        'n. & v. i.',
        'n. /',
        'n. / interj.',
        'n. & v.',
        'sing. or pl.',
        'n.sing & pl.',
        'n',
        'n., a., & v.',
        'n. & a.',
        'sing. & pl.',
        'n .',
        'v. t. & n.',
        'n. sing. & pl.',
        'a., n., & adv.',
        'n. & adv.',
        'n. / v. t. & i.',
        'n.sing. & pl.',
        'n. .',
        'v.& n.',
        'n. & interj.',
        'adv. & n.',
        'n. Chem.',
        'v. i. & n.',
        'n.',
        'sing.',
        'N.',
        'n./',
        'adv., & n.',
        'a. / n.',
        'v. & n.',
        'a., adv., & n.',
        'n..',
        'n. sing. & pl',
        'interj. & n.',
        'n. sing.',
        'n. & i.',
        'imperative sing.',
        'syntactically sing.'
    },
    'Noun-Pl': {
        'n. pl.',
        'n. sing & pl.',
        'n.pl.',
        'sing. or pl.',
        'n.sing & pl.',
        'sing. & pl.',
        'n. pl',
        'n. sing. & pl.',
        'n.sing. & pl.',
        'n pl.',
        'n., sing. & pl.',
        'n. collect. & pl.',
        'n. sing. & pl',
        'n. pl.',
        'sing. / pl.'
    },
    'Gerund-V': {
        'p. pr. & v. n.',
        'p. pr. &, vb. n.',
        'imp. & p. p. Fenced (/); p. pr. & vb. n.',
        'imp. & p. p. & vb. n.',
        'p, pr. & vb. n.',
        'p. pr. a. & vb. n.',
        'p. pr. vb. n.',
        'imp. & p. pr. & vb. n.',
        'pr.p. & vb. n.',
        'p. pr. / vb. n.',
        'p]. pr. & vb. n.',
        'p. pr.& vb. n.',
        'p. pr. &vb. n.',
        'p. pr. & vb/ n.',
        'P. pr. & vb. n.',
        'p. pr. & vvb. n.',
        'p. a. & vb. n.',
        'p. pr. &. vb. n.',
        'p. pr. & pr. & vb. n.',
        'vb. n.',
        'p. p. & vb. n.',
        'p pr. & vb. n.',
        'imp. & p. p. Adored (/); p. pr. & vb. n.',
        'p. pr & vb. n.'
    },
    'Verb-I-Sg': {
        '3d sing.pr.',
        'subj. 3d pers. sing.',
        '3d sing.',
        '3d pers. sing. pres.',
        '3d sing. pr.',
        'pres. indic. sing., 1st & 3d pers.',
        'Sing. pres. ind.',
        '3d sing.',
        'pres. sing.'
    },
    'Verb-T-Sg': {
        '3d sing.pr.',
        'subj. 3d pers. sing.',
        '3d sing.',
        '3d pers. sing. pres.',
        '3d sing. pr.',
        'pres. indic. sing., 1st & 3d pers.',
        'Sing. pres. ind.',
        '3d sing.',
        'pres. sing.'
    },
    'Verb-I-Pl': {
        'v. t. / i.',
        'v. i.,',
        'n. & v. i.',
        'v. i. & i.',
        'v. i.',
        'v.t & i.',
        'v.i',
        'v. t. / v. i.',
        'v.i.',
        'v. t.& i.',
        'n. / v. t. & i.',
        'v. i.',
        'v. t. & v. i.',
        'v. i. & n.',
        'v. i. & auxiliary.',
        'v. t. & i.',
        'v. i. & t.',
        'v. i. / auxiliary'
    },
    'Verb-T-Pl': {
        'v. t. / i.',
        'a. & v. t.',
        'v. t. &',
        'n. & v. t.',
        'v. t..',
        'v. t. v. t.',
        'v.t & i.',
        'v. t. / v. i.',
        'v.t',
        'v. t. & n.',
        'v. t.& i.',
        'n. / v. t. & i.',
        'v. t. & v. i.',
        'v./t.',
        'v. t.',
        'v. t. / auxiliary',
        'v. t.',
        'v. i. & t.',
        'v.t.'
    },
    'Preposition': {
        'prep., adv., & conj.',
        'prep., adv., conj. & n.',
        'adv. & prep.',
        'prep. & conj., but properly a participle',
        'prep., adv. & a.',
        'prep., adv. & conj.',
        'prep. & adv.',
        'adv., prep., & conj.',
        'prep.',
        'adv. or prep.',
        'prep. & conj.',
        'conj. & prep.'
    },
    'Adj': {
        'adj.',
        'pron. / adj.',
        'a.',
        'p. p. / a.',
        'a. & v. t.',
        'adv. & a.',
        'p. p & a.',
        'p. p. & a.',
        'a. / a. pron.',
        'P. p. & a.',
        'pron. & a.',
        'a & n.',
        'a/',
        'adv. / a.',
        'a. & a. pron.',
        'a & p. p.',
        'p. & a.',
        'prep., adv. & a.',
        'a. .',
        'a. superl.',
        'v. & a.',
        'a. & adv.',
        'n., a., & v.',
        'a. a.',
        'pron., a., conj., & adv.',
        'n. & a.',
        'p. pr. a. & vb. n.',
        'a. & v.',
        'a., n., & adv.',
        'a. Vigorously',
        'a. & n.',
        'a.',
        'a. / adv.',
        'a & adv.',
        'a. Vibrating',
        'a. or pron.',
        'a. / pron.',
        'imp., p. p., & a.',
        'a',
        'p. p. & a',
        'a. / n.',
        'pron., a., & adv.',
        'a., adv., & n.',
        'a. & p. p.',
        'a. & pron.'
    },
    'Adv': {
        'prep., adv., & conj.',
        'prep., adv., conj. & n.',
        'adv. & a.',
        'adv. In combination or cooperation',
        'adv. / interj.',
        'interrog. adv.',
        'adv. & prep.',
        'adv. In a vanishing manner',
        'adv. / a.',
        'prep., adv. & a.',
        'a. & adv.',
        'pron., a., conj., & adv.',
        'prep., adv. & conj.',
        'conj. / adv.',
        'adv.',
        'prep. & adv.',
        'interj., adv., or a.',
        'a., n., & adv.',
        'interj., adv., & n.',
        'n. & adv.',
        'a. / adv.',
        'adv., prep., & conj.',
        'adv. & n.',
        'a & adv.',
        'adv. or prep.',
        'adv., & n.',
        'pron., a., & adv.',
        'a., adv., & n.',
        'interj. & adv.',
        'adv. / conj.',
        'adv. & conj.'
    }
}


def create_lexicon(lines):
    lexicon = {}
    for tag in pos_tag_to_word_classes:
        lexicon[tag] = set()
    for line in lines:
        line = line.strip()
        if line == '':
            continue
        try:
            entry = DictEntry.from_line(line)
        except:
            print('Failed to parse line: {}'.format(line))
            continue
        for tag in pos_tag_to_word_classes:
            word_class_set = pos_tag_to_word_classes[tag]
            if entry.word_class in word_class_set:
                lexicon[tag].add(entry.headword.lower().replace('"', '\\"')
)
    return lexicon


def write_lexicon(lexicon, stream):
    for tag in pos_tag_to_word_classes:
        for terminal in lexicon[tag]:
            stream.write('{} -> "{}"\n'.format(tag, terminal))


def create_write_lexicon():
    try:
        output = open('espg_lexicon.txt', 'w')
    except:
        print('Failed to open file: {}'.format('espg_lexicon.txt'))
        exit()
    for i in range(ord('A'), ord('Z') + 1):
        file_name = ('../../Dictionary-in-csv/{}.csv'.format(chr(i)))
        try:
            with open(file_name, 'r') as stream:
                lines = stream.readlines()
        except:
            print('Failed to open file: {}'.format(file_name))
        else:
            lexicon = create_lexicon(lines)
            write_lexicon(lexicon, output)
    output.close()


if __name__ == '__main__':
    create_write_lexicon()
```

C.30 src/writer/writer.py

```
from string import punctuation
from numpy.random import poisson


```

```python
class Sentence:

    __slots__ = 'tokens'

    def __init__(self, tokens):
        self.tokens = tokens

    def __str__(self):
        elements = []
        for token in self.tokens:
            if len(elements) > 0:
                if token[0] in punctuation or elements[-1][-1] in punctuation:
                    elements[-1] += token
                else:
                    elements.append(token)
            else:
                elements.append(token.capitalize())
        return ' '.join(elements)

    def word_count(self):
        count = 0
        for token in self.tokens:
            if token not in punctuation:
                count += 1
        return count


class Paragraph:

    __slots__ = 'sentences', 'indent'

    def __init__(self, indent):
        self.sentences = []
        self.indent = indent

    def __str__(self):
        elements = []
        for sentence in self.sentences:
            elements.append(str(sentence))
        return ' ' * self.indent + ' '.join(elements)

    def add_sentence(self, tokens):
        self.sentences.append(Sentence(tokens))

    def word_count(self):
        count = 0
        for sentence in self.sentences:
            count += sentence.word_count()
        return count


class Article:

    __slots__ = 'paragraphs', 'title', 'spacing', 'indent'

    def __init__(self, title, spacing, indent):
        self.paragraphs = []
        self.title = title
        self.spacing = spacing
        self.indent = indent

    def __str__(self):
        elements = []
        if self.title:
            elements.append(self.title)
        for paragraph in self.paragraphs:
            if len(paragraph.sentences) == 0:
                continue
            elements.append(str(paragraph))
        return ('\n' * (self.spacing + 1)).join(elements)

    def add_sentence(self, tokens):
        if len(self.paragraphs) == 0:
            self.new_paragraph()
        self.paragraphs[-1].add_sentence(tokens)

    def new_paragraph(self, indent=None):
        if len(self.paragraphs) > 0 and \
                len(self.paragraphs[-1].sentences) == 0:
            self.paragraphs.pop()
        if indent == None:
            indent = self.indent
        self.paragraphs.append(Paragraph(indent))

    def word_count(self):
        count = 0
        for paragraph in self.paragraphs:
            count += paragraph.word_count()
        return count
```

```python


class Writer:

    __slots__ = 'grammar', 'paragraphs_per_article',\
        'sentences_per_paragraph', 'tokens_per_sentence'

    def __init__(self, grammar, paragraphs_per_article=5,
                 sentences_per_paragraph=10, tokens_per_sentence=20):
        self.grammar = grammar
        self.paragraphs_per_article = paragraphs_per_article
        self.sentences_per_paragraph = sentences_per_paragraph
        self.tokens_per_sentence = tokens_per_sentence

    def generate(self, title=None, spacing=1, indent=4):
        article = Article(title, spacing, indent)
        for i in range(poisson(self.paragraphs_per_article)):
            for j in range(poisson(self.sentences_per_paragraph)):
                attempt_count = 0
                while True:
                    try:
                        max_length = poisson(self.tokens_per_sentence)
                        tokens = self.grammar.generate(max_length=
    max_length)
                    except:
                        attempt_count += 1
                        if attempt_count > 50:
                            raise Exception('Too many attempts')
                    else:
                        article.add_sentence(tokens)
                        break
            article.new_paragraph()
        return article


if __name__ == '__main__':
    from grammar import CFG, demo_grammar
    article = Writer(CFG(demo_grammar)).generate('Demo Article')
    print(article)
    print()
    print('Word count: {}'.format(article.word_count()))
```

C.31 src/writer/main.py

```python
from grammar import CFG
from lexicon import create_write_lexicon
from writer import Writer


def create_espg():
    create_write_lexicon()
    espg = CFG()
    with open('espg_base.txt', 'r') as stream:
        lines = stream.read()
    espg.load_lines(lines)
    with open('espg_lexicon.txt', 'r') as stream:
        lines = stream.read()
    espg.load_lines(lines)
    return espg


def default_input(message, default_value, variable_name):
    try:
        value = int(input(message))
    except:
        value = default_value
        print('{} set to {}.'.format(variable_name, default_value))
    return value


if __name__ == '__main__':
    print('Creating ESPG...\n')
    espg = create_espg()
    while True:
        paragraphs_per_article = default_input(
            'Paragraphs per article: ', 3, 'paragraphs_per_article')
        sentences_per_paragraph = default_input(
            'Sentences per paragraph: ', 10, 'sentences_per_paragraph')
        tokens_per_sentence = default_input(
            'Tokens per sentence: ', 15, 'tokens_per_sentence')
        title = input('Title: ')
        spacing = default_input('Spacing: ', 1, 'spacing')
        indent = default_input('Indent: ', 4, 'indent')
        writer = Writer(espg, paragraphs_per_article,
                        sentences_per_paragraph, tokens_per_sentence)
        print('Composing...\n')
        article = writer.generate(title, spacing, indent)
        print(article)
        print()
        print('Word count: {}'.format(article.word_count()))
        print('=' * 80)
```