Monona — A Poor Man's DIY Dictionary Kit

Wuu Yang
Computer and Information Science
National Chiao-Tung University
Hsin-Chu, Taiwan
Repulbic of China
wuuyang@cis.nctu.edu.tw

Pin-Chia Feng
Foreign Languages and Literatures
National Chiao-Tung University
Hsin-Chu, Taiwan
Repulbic of China
pcfeng@cc.nctu.edu.tw

Abstract

When learning a new foreign language, a non-native speaker met more problems in using a word than in understanding the meaning of the word. The Monona system is a useful tool for learners of English that provides a user with many sample sentences containing a queried word. Monona builds a database of the high-quality articles that are available freely in the internet and an accompanied index. Since building the index is done automatically by programs, it is cheap to add more articles to the database. Finally, a user interface is built with the Java programming language that allows a user to enter queries. The results are presented in a web browser.

keywords: *composition, concordance, dictionary, word processing*

1 Introduction

Dictionaries have been a great help for non-native speakers to learn a foreign languages, but only at the early stages. What a more advanced non-native speaker wants most are sample sentences, that is, she/he wants to know how to use a given word. For instance, even given the definition *"a situation in which a desired outcome or solution is impossible to attain because of a set of inherently illogical rules or conditions"*, it is not easy to see how the word *catch-22* is usually used in a sentence. On the other hand, with a sample sentence *"the catch-22 of a closed repertoire, only music that is already familiar is thought to deserve familiarity"* [1], a user sees not only the meaning but also the usage of the word *catch-22*.

Many modern dictionaries [3, 4] have added helpful information, such as idioms and short phrases, in addition to the definitions of words. But, due to the sizes of dictionaries, few contain sufficient examples as far as a non-native speaker would like to see.

It is too easy for a non-native speaker to write a sentence that agrees with all the grammar rules but that is completely unnatural to the native speakers (you probably could find such sentences in this article). The root of this problem lies in the fact that grammar rules are never complete. Though grammars can help a non-native speaker to learn a new language quickly, there are always exceptions to rules listed in a grammar book. The best way to learn a natural language is to use it and to see how it is actually used.

The Monona system is similar to an electronic English concordance that provides a user with many sample sentences that contains a queried word. It aims at helping non-native speakers to write and speak natural English.

The first question in designing Monona is finding sufficient sample sentences. It is out of the question to ask professional editors to compile sentences for all the English words due to the budget problem. In Monona, the sources are taken from the internet where there are many organizations that publish free, high-quality articles, such as *Times* and *Scientific American*. These high-quality articles are good enough to serve as models for non-native speakers to learn English.

After an article is retrieved from the internet (manually), a program, called *sindex*, builds an index file for the words in the article and a database of English sentences. A second program, called *smerge*, merges multiple index files into a single one. The third program, called *ssearch*, provides a user interface to search the database with a queried word and to present the results of searching to the user. The third program is written in Java and hence should be able to run on most platforms.

An advantage of the Monona system is that new sentences can be easily added to our collection. Only a new index needs to be built when new articles are added to the database. Monona can also be used to study the writing style of a specific writer, such as Jane Austen's work, or of a particular academic field, such as chemistry. It is also easy to add a dictionary facility to Monona: by arranging the index properly, the definition of a word could be viewed as a quotation containing the word. Monona can also help editors to compile a dictionary.

This paper presents the design and implementation of the Monona system. The remainder of this paper is organized as follows: The next section discusses how to decide the boundaries of words and sentences. The third section discusses the mechanism for merging the index files. The fourth section discusses the user interface of the query system. The last section concludes this paper.

2 Building the Index

The first stage of the *Monona* system is the indexing program *sindex*. *Sindex* divides the input article into words and sentences. Individual sentences are added to the database separately. The words are used to build an index, which are used in the searching program.

The input article is a text file. Because a user might want to view the entire article with a browser, the input article is marked with a few simple html tags, such as `<header>`, `<title>`, and `<p>`. Because we are only interested in the primary text of the article, certain information, such as the copyright notice, publication date, author(s), etc., is skipped. Monona provodes the pair of tags `<!s>` and `<!q>` to mark the interesting texts. Only the text inside the pair of tags will be indexed.

Ordinary English words are made up of English letters. In everyday English we also use abbreviations such as `Mr.`, `etc.`, etc., special words, such as `catch-22`, and many scientific words, such as `RU486`. In Monona, only ordinary words will be considered for indexing. Abbreviations, special words, and scientific words are not indexed. However, Monona also provides the pair of tags `<!w>` and `<!e>` to group a sequence of characters that will be indexed as ordinary English words. For instance, we could use the following tags to index the word `catch-22`:

```
The <!w>catch-22<!w> of a
closed repertoire, only music
that is already familiar is
```

```
thought to deserve
familiarity.
```

Compound words, such as `brother-in-law`, are treated as three independent words, as is done in most dictionaries. An editor can also choose to use the pair of tags `<!w>brother-in-law<!e>` to force Monona to index on the whole word `brother-in-law`. Almost all characters could be enclosed in the pair of tags `<!w>` and `<!e>`.

In order to reduce the size of the index file, certain common words are not indexed. These words include the pronouns (such as `I`, `you`, etc.), prepositions (such as `in`, `at`, etc.), abbreviations (such as `A.`, `Mr.`, `Jr.`, etc.), and common verbs (such as `have`, `is`, etc.). These avoided words are explicitly listed in a file, which can be edited as necessary.

In Monona, upper-case letters are treated as their lower-case counterparts.

Numbers, such as `1098`, `3.1416`, etc., are also not indexed because their usages are very uniform and can be found in many popular dictionaries.

As a rule, Monona usually indexes on all words in the regions that are explicitly marked by the pair of tags `<!s>` and `<!q>` in an article. However, Monona also provides a pair of tags `<!y>` and `<!z>` that can be used to mark a region in the article in which all words will *not* be indexed.

A difficulty in processing English words intelligently is that there are many inflections of a word. Nouns have singular and plural forms. Verbs have many tenses. Adjectives and adverbs have comparative and superlative forms. For instance, `fantasy` and `fantasies` are essentially the same word. In Monona, every word taken from an article is therefore transformed to its canonical form. Plural nouns are turned into their singular form; verbs in the past tense, part participle, present participle, and the third-person singular form are turned to the original form; adjectives and adverbs in the comparative and superlative form are translated to their normal form.

The transformation is not easy because, in English, there are many exceptions to the rules of inflection. Such irregular examples include `criterion` and `criteria`. In Monona, we use the following rules to transform a word to its canonical form. These rules are applied in the order listed below.

1. Words ending with `ables`, `ages`, `ances`, `ants`, `ars`, `ates`, `cals`, `ences`, `ents`, `ers`, `eses`, `graphs`, `hoods`, `ians`, `ings`, `isms`, `ists`, `ments`, `ors`, `scopes`, `ships`, `sions`, `sives`, `tions`, `tives`, `tons`, `tudes`, `tures` are considered the plural form of a word. Monona simply deletes the last character `s` and assumes the transformed word is in the canonical form.

2. If a word ends with `aries` (such as `ovaries`), its suffix is replaced with `ary`. If a word ends with `ating` (such as `creating`), its suffix is replaced with `ate`. If a word ends with `encies` (such as `currencies`), its suffix is replaced with `ency`. If a word ends with `eries` (such as `arteries`), its suffix is replaced with `ery`. If a word ends with `ifies` (such as `justifies`), its suffix is replaced with `ify`. If a word ends with `ified` (such as `notified`), its suffix is replaced with `ify`. If a word ends with `ifying` (such as `qualifying`), its suffix is replaced with `ify`. If a word ends with `ities` (such as `activities`), its suffix is replaced with `ity`. If a word ends with `ories` (such as `observatories`), its suffix is replaced with `ory`. Monona assumes the transformed word is in the canonical form.

3. If a word ends with `s` (such as `gifts`), Monona first deletes its suffix `s`. If the

transformed word appears a table `Sword`, then Monona assumes it is in the canonical form.

4. If a word ends with `es` (such as `matches`), Monona first deletes its suffix `es`. If the transformed word appears a table `ESword`, then Monona assumes it is in the canonical form.

5. If a word ends with `ies` (such as `countries`), Monona first changes its suffix to `y`. If the transformed word appears a table `IESword`, then Monona assumes it is in the canonical form.

6. If a word ends with `ves` (such as `calves`), Monona first changes its suffix to `f`. If the transformed word appears a table `VESword`, then Monona assumes it is in the canonical form.

7. If a word ends with `ves` (such as `wives`), Monona first changes its suffix to `fe`. If the transformed word appears a table `EVESword`, then Monona assumes it is in the canonical form.

8. If a word ends with `d` (such as `achieved`), Monona first deletes its suffix `d`. If the transformed word appears a table `Dword`, then Monona assumes it is in the canonical form.

9. If a word ends with `ed` (such as `accepted`), Monona first deletes its suffix `ed`. If the transformed word appears a table `EDword`, then Monona assumes it is in the canonical form.

10. If a word ends with `ied` (such as `carried`), Monona first ahanges its suffix to `y`. If the transformed word appears a table `IEDword`, then Monona assumes it is in the canonical form.

11. If a word ends with `ed` and the third-to-the-last and the fourth-to-the-last characters are identical (such as `mapped`), Monona first deletes its last three characters. If the transformed word appears a table `REDword`, then Monona assumes it is in the canonical form.

12. If a word ends with `ing` (such as `boiling`), Monona first deletes its suffix `ing`. If the transformed word appears a table `INGword`, then Monona assumes it is in the canonical form.

13. If a word ends with `ing` and the fourth-to-the-last and the fifth-to-the-last characters are identical (such as `robbing`), Monona first deletes its last four characters. If the transformed word appears a table `RINGword`, then Monona assumes it is in the canonical form.

14. If a word ends with `ing` (such as `automating`), Monona first changes its suffix to `e`. If the transformed word appears a table `EINGword`, then Monona assumes it is in the canonical form.

15. If a word ends with `r` (such as `finer`), Monona first deletes its suffix `r`. If the transformed word appears a table `Rword`, then Monona assumes it is in the canonical form.

16. If a word ends with `er` (such as `clearer`), Monona first deletes its suffix `er`. If the transformed word appears a table `ERword`, then Monona assumes it is in the canonical form.

17. If a word ends with `ier` (such as `wearier`), Monona first changes its suffix to `y`. If the transformed word appears a table `IERword`, then Monona assumes it is in the canonical form.

18. If a word ends with `er` and the third-to-the-last and the fourth-to-the-last characters are identical (such as `hotter`), Monona first deletes its last three characters. If the transformed word appears a table `RERword`, then Monona assumes it is in the canonical form.

19. If a word ends with `st` (such as `crudest`), Monona first deletes its suffix `st`. If the transformed word appears a table `STword`, then Monona assumes it is in the canonical form.

20. If a word ends with `est` (such as `deepest`), Monona first deletes its suffix `est`. If the transformed word appears a table `ESTword`, then Monona assumes it is in the canonical form.

21. If a word ends with `iest` (such as `chilliest`), Monona first changes its suffix to `y`. If the transformed word appears a table `IESTword`, then Monona assumes it is in the canonical form.

22. If a word ends with `est` and the fourth-to-the-last and the fifth-to-the-last characters are identical (such as `hottest`), Monona first deletes its last four characters. If the transformed word appears a table `RESTword`, then Monona assumes it is in the canonical form.

23. Monona keeps a table `SPECIALword` of irregular inflections, such as `go` and `went`. A word not matching any of the above rules will be checked again this table to see if it is an irregular inflection.

24. A word not matching any of the above rules are considered to be in its canonical form.

The first and the second rules do not require any tables. However, the other 21 rules rely on tables.

The table `EDword` contains the words that have inflections by adding the suffix `ed`. Other tables contain similar information. The 21 tables are compiled manually. When a new article is about to be added to the Monona database, we check all the words that are not already in the database. Each word is added to appropriate tables. It turns out that compiling these tables constitutes a major part of the efforts in developing the Monona system.

When a user submits a query, his query must also go through the same transformation. This means that the query interface `ssearch` must also reserve memory space for the 21 tables. We are trying to find more rules similar to the first and the second rules in order to reduce the size of the 21 tables.

A second difficulty in processing English words intelligently is that there are different words that happen to be spelled identically. For instance, the word `wound` can mean *injury* or it can be considered as the past tense of the verb `wind`. Therefore, the canonical form of the word `wound` could be either `wound` or `wind`. For such words, Monona makes use of a meta-file that links together these different canonical forms of the same word.

An article is divided into individual sentences. Each sentence is stored in a single quotefile in the Monona database. Deciding the boundary of a sentence also causes a few minor problems. Usually, a sentence ends with a period, a question mark, or an exclamation mark . However, a period may also be used as the abbreviation symbol, such as `Mr.`. Furthermore, a sentence might interfere with quotations. A sentence may be divided into two separate quotations and a quotation may contain two or more sentences. In this case, the last sentence inside a quotation ends with `."`, `!"` or `?"`. Similarly, a sentence inside a pair of parentheses may end with `.)`, `!)`, or `?)`.

Monona keeps a list of common abbreviations. If a period together with the preceding word is one of the abbreviations, the period is taken as a part of the abbreviation. Otherwise, a sentence is delimited by one of the six punctuation signs: `.`, `!`, `?`, `."`, `!"`, and `?"`. Other quotation signs are treated as ordinary punctuation marks. An implication is that a long quotation might be partitioned into several sentences. For instance, the following quotation is divided into four sentences.

```
   The office is quiet this
morning.  "The plant grows,"
she talked to me, "pretty well
in the house.  Do you know how
to use the xerox machine?"
There are five xerox machines
in our office.
```

Sometimes, we would like a group of sentences to be stored as in a single quotefile, for instance, a poem. In this case, we can use the pair of tags `<!n>` and `<!q>` to indicate a region of text that should be treated as a single sentence. For instance,

```
<!n>Humpty Dumpty sat on a
   wall.
Humpty Dumpty had a great
   fall.
All the King's horses and all
   the King's men
Could not put Humpty Dumpty
   together again.<!q>
```

The tags defined in Monona are used to control the indexing operations. Note that Monona is designed to require as little human intervention as possible. For instance, after retrieving an article from the world-wide web, we add only a few html tags to highlight the title, author, date, and paragraphs in the article. Few other tags are added in an usual article. Because we totally trust the selected magazines concerning the quality of the articles, there is no extra editing work on the source articles. In this way, we can quickly accumulate a large database at very little cost. Currently, the database in Monona contains around 20 megabytes of pure text.

The output of the indexing program consists of the index file and a collection of quotefiles. The quotefiles are added to the Monona database. The format of the index file is similar to the database of the *bib* program in the Unix system. Below is a sample index file:

```
%A able
%F godel1999
%X 000000038
%T 000000858
%F money2000
%X 000000021
%T 000000503

%A abnormally
%F health1999
%X 000000025
%T 000000444

%A aborigine
%F walcott1992
%X 000000173
%T 000003555
```

An index file is divided into segments. Each segment, which is indicated by the `%A` label, corresponds to an indexed word. A segment is made up of one or more elements. Each element contains three labels: `%F` (which indicates the file name), `%X` (which indicates the extension of the file name) and `%T` (which indicates an anchor point in the article).

The Monona database consists of many quotefiles. The name of a quotefile is made up of the `%F` and `%X` labels. In the above example database, the first quotefile containing the word `able` is `godel1999.000000038`. All the quotefiles taken from an article is collected under a directory in the Monona database.

3 Merging the Indexes

The second stage of Monona is the `smerge` program. It merges and sorts the index files produced by `sindex` or `smerge`.

`Smerge` also prepares several header files for inclusion in the third stage `ssearch`. The header files could be prepared in the C language or in the Java language.

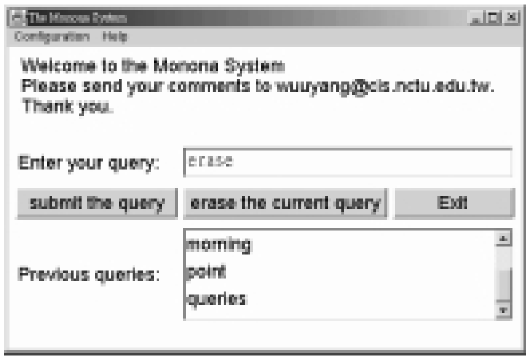
In the C language version, the header files define several data structures, their sizes, and their values. These header files will be compiled together with other part of the `ssearch` program. A sample C header file is as follows:

```
int anchorCNT=31956;
ANCHORNODETYPE
   anchorTab[31957] = {
    {3709,8,56},
   {3709,11,125},
   {2306,52,1115}, ... };
```

In the Java language version, the header files defines the data structures and their sizes. Their values are prepared in a separate file which will be an input file for the `ssearch` program.

The data structures defined in these header files are the various tables (`Sword`, `EDword`, etc.) used in the transformation process. `Smerge` dumps these data structures in the alphabetical order.

`Smerge` makes use of the binary search tree [2] when it merges several index files. After merging the indexes, `smerge` dumps the combined index to a file in the sorted order. The combined index will be an input file to the `ssearch` stage of Monona.



Welcome to the Monona System
Please send your comments to wuuyang@cis.nctu.edu.tw.
Thank you.

Enter your query: erase

Previous queries:
morning
point
queries

Figure 1: Query interface

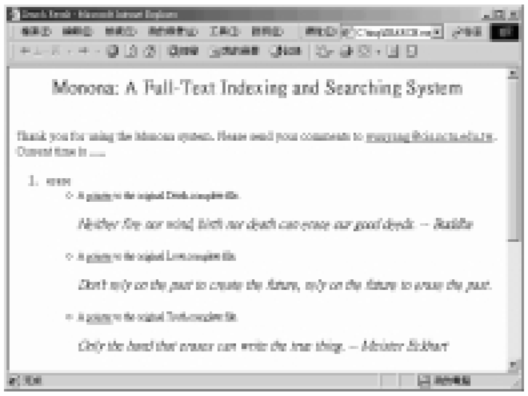
4 The Querying Interface

The third stage of Monona is the query interface. There are two versions of the query interface. One version is implemented as a server in the Unix environment. Other programs may communicate with the server through inter-process communication facilities. Based on this server, we built a web site that answers queries from all over the world-wide web.

The second version of the query interface is implemented in the Java language. This version is a stand-alone program. Due to the portability of Java programs, this version should be able to run on most platforms. A sample interface is shown in Figure 1.

The result of a query is prepared in the html format. Monona finally invokes an internet browser, such as Internet Explorer or Netscape, to present the result. A sample result is shown in the Figure 2.

The main data structures used in `ssearch` are the index and the transformation tables. All of these data structures are organized as ordered lists with the binary search operations. The ordered lists are actually built by the previous stage, `smerge`. `Ssearch` simply restores the ordered lists from the dump files


Monona: A Full-Text Indexing and Searching System

Thank you for using the Monona system. Please send your comments to [illegible].
Current time is ......

1. [illegible]
   - A pointer to the original [illegible] file.

     Neither fire nor wind, birth nor death can erase our good deeds. -- Buddha
   - A pointer to the original [illegible] file.

     Don't rely on the past to create the future, rely on the future to erase the past.
   - A pointer to the original [illegible] file.

     Only the hand that erases can write the true thing. -- Meister Eckhart

Figure 2: The result of a query

produced by `smerge`.

5 Conclusion

We have described the design and implementation of a useful tool for non-native speakers to learn English. Monona consists of three stages: `sindex` builds an index for an input article. `smerge` combines several index files into a single index. `ssearch` provides a query interface. Currently we are working on enriching the Monona database while reducing the size of the index and the transformation tables.

Acknowledgement

This work is supported, in part, by National Science Council, Taiwan, Republic of China, under contracts NSC 89-2213-E-009-014, NSC 89-2213-E-009-068, NSC 89-2213-E-009-146 and NSC 90-2213-E-009-142.

References

[1] The American Heritage Dictionary of the English Language, 3rd Edition, Houghton Mifflin Company, New York, 1992.

[2] E. Horowitz, E., S. Sahni and S. Anderson-Freed, “Fundamentals of Data Structures in C,” W.H. Freeman and Company, New York, 1993.

[3] Webster’s Ninth New Collegiate Dictionary, Merriam-Webster, Springfield, Massachusetts, 1985.

[4] Webster’s New World Dictionary of American English, Third College Edition, Prentice Hall, New York, 1994.