# Chinese Word Segmentation

Team Name: TIPO
Team Leader: 谢雨波

January 2, 2012

## 1 Prototype System Introduction

### 1.1 Functions

This is a system dealing with Chinese word segmentation. With such a system, you can:

- Load a lexicon which fits our format
- Load a rule file which fits our format
- Open texts in files
- Segment sentences
- Choose one sentence to segment
- Add a new word to the lexicon
- Add a new rule to the rule file
- Save the result of sentence and word segmentation

### 1.2 Running Environment

We test this system in Windows 7 with Service Pack 1. However, since the system is developed with Python, it can, theoretically, also run in environments supporting Python, such as Mac OS X, Linux and so on.

### 1.3 Developing Environment

We develop the system in Python IDLE (Python GUI).

# 2 Task Allocation

## 2.1 Task Allocation

Here are the tasks for each team member:

Table 1: Task Allocation

| Position | Name | No. | Task |
|---|---|---|---|
| Leader | 谢雨波 | 5110309766 | Algorithm and Implementation |
| Member 1 | 李斌彬 | 5110309769 | User Interface |
| Member 2 | 李瑛恺 | 5110309765 | Implementation and Debugging |
| Member 3 | 赵小波 | 5110309768 | Lexicon and Testing |

# 3 System Architecture

## 3.1 User Interface Components

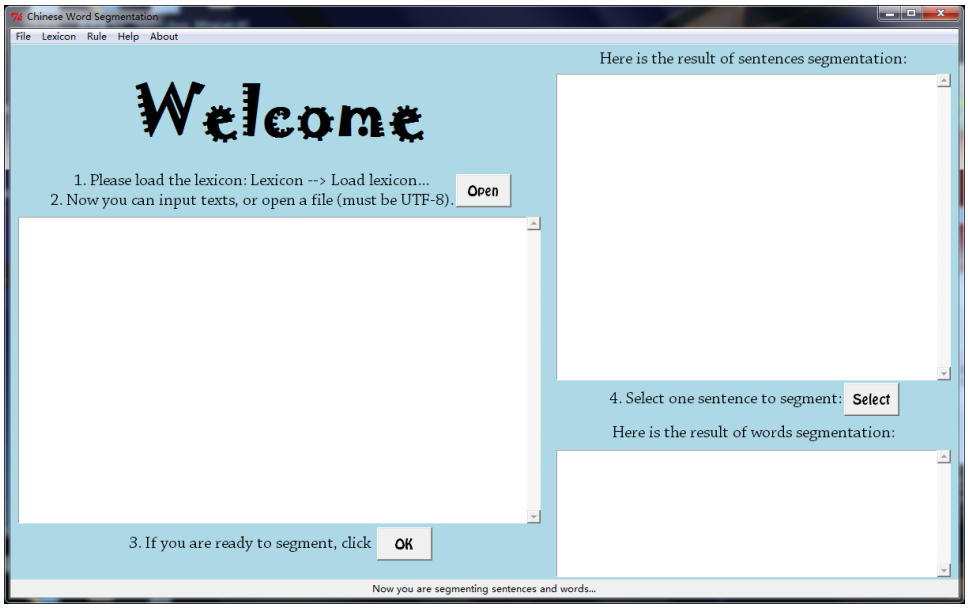


Figure 1: User Interface

The main function contains the most major part of the user interface component. And other functions defined above are all the commanding functions that will be called in the main function. Since these functions won't matter much about the system architecture, we don' t specify them.

In the main function, the following statements create the root on which all components are based:

```
root = Tk()
root.geometry('1200x700')
root.title('Chinese Word Segmentation')
```

And next we are going to construct the main menu:

```
menu = Menu(root)
root.config(menu = menu)
```

Together come some submenus created by some statements like these:

```
# ...
filemenu = Menu(menu)
menu.add_cascade(label = 'File', menu = filemenu)
filemenu.add_command(label = 'Open...', command = OpenFile)
filemenu.add_separator()
filemenu.add_command(label = 'Save sentence as...', command = save1)
filemenu.add_command(label = 'Save word as...', command = save2)
filemenu.add_separator()
filemenu.add_command(label = 'Exit', command = root.destroy)
# ...
```

which look like something following:

The status bar of the root window is created by the following statements:

```
status = Label(root, text = 'text...', anchor = S)
status.pack(side = BOTTOM, fill = X)
```

In this program, we put all components such as texts, buttons into the `Frame` which is easy to used to adjust their position. So we create quite a lot frames, and some frames are nested in others.

```
framex = Frame(root, height = 700, width = 1200, bg = 'lightblue')
frame = Frame(framex, height = 700, width = 300, bg = 'lightblue')
frame0 = Frame(frame, height = 150, width = 600)
# ...
```


Figure 2: Menu and Submenu

We also create 3 buttons: "Open", "OK", and "Select":

1. "Open" can open a dialog box that suggests you should choose a text file to open.
2. "OK" can be clicked when you're ready to segment the text into sentences.
3. "Select" will produce a dialog box for you to select one sentence to segment.

Take the "Open" button for example, writing the following statements will produce an "Open" button:

```
button1 = Button(frame1, font = ft3, text = 'Open',
                 width = 6, command = OpenFile)
button1.pack(side = LEFT)
```

and here is what the three buttons look like:


Figure 3: The Three Buttons

What's more, the most important component is text. We create three texts and corresponding scroll bars, using the following statements:

```
text = Text(frame2)
scrollbar = Scrollbar(frame2)
scrollbar.pack(side = RIGHT, fill = Y)
text['yscrollcommand'] = scrollbar.set
text.pack(side = LEFT)
scrollbar['command'] = text.yview
```

Besides, we also create three types of fonts that can make our program more luscious.

```
ft1 = tkFont.Font(family = 'Jokerman', size=60,
                  weight = tkFont.BOLD)
ft2 = tkFont.Font(family = 'Chaparral Pro', size = 16,
                  weight = tkFont.NORMAL)
ft3 = tkFont.Font(family = 'Hobo Std', size=13,
                  weight = tkFont.NORMAL)
```

# 4 Algorithm Description

## 4.1 User Interface

Since there are not many important algorithms in the interface part, we will specify the way of constructing the interface with some flowcharts.

In fact, we just construct many frames that can be easily placed. And others such as buttons and text are all put into them.

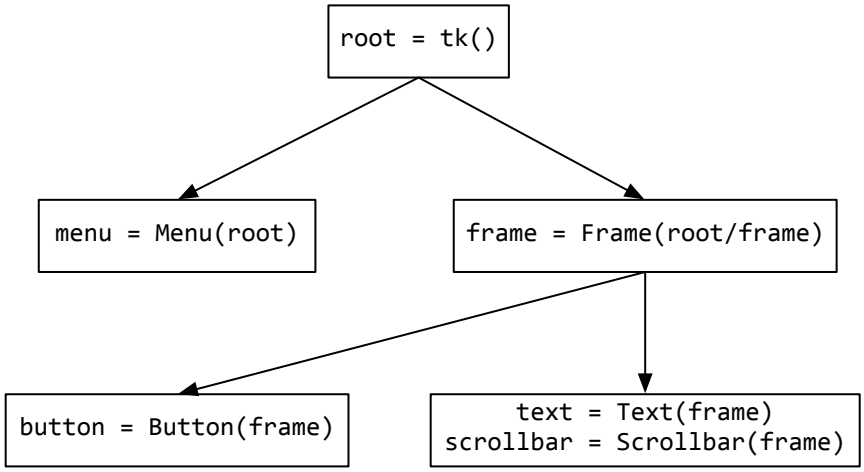


Figure 4: Construct Frames

As for some other functions such as `Instruction()`, `Copyright()`, we just create a toplevel with a label.

However, for some functions which need open or save files, we should be more careful with them. First, we use `askopenfilename()` to get the path where a file is stored or will be saved. Then we can use this path to save or read the corresponding file. However, if the user clicks "Cancel", there will be no path and error will take place. So we use an `if` statement to overcome this problem.

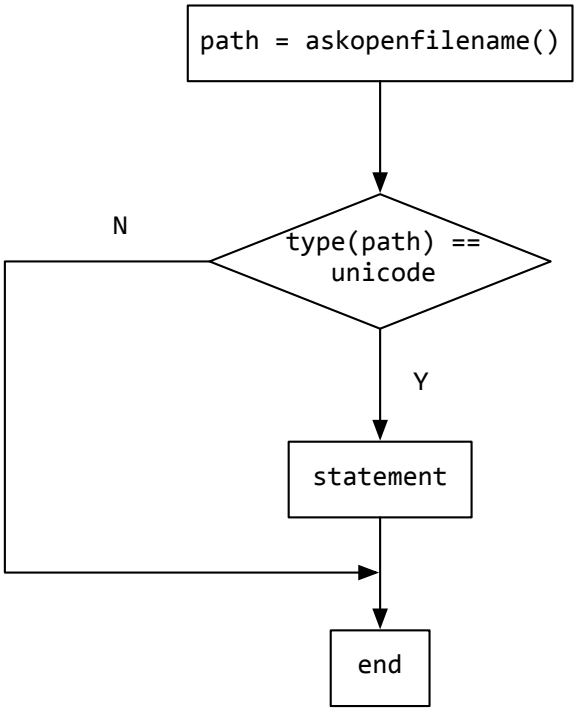


Figure 5: Dealing with Path

This idea can be implemented using the following code:

```
# Open a text file.
def OpenFile():
    global text
    path = askopenfilename(title = 'Sentences Input')
    if type(path) == unicode:
        text.delete('1.0', END)
        file_in = open(path, 'r')
```

```
s = file_in.read().decode('utf-8', 'ignore')
text.insert('1.0', s)
file_in.close()
```

What's more, we also build a function that let users input integer. However, if users input inappropriate number or click "Cancel", it will show error as well. So we also need to put an `if` statement in it.

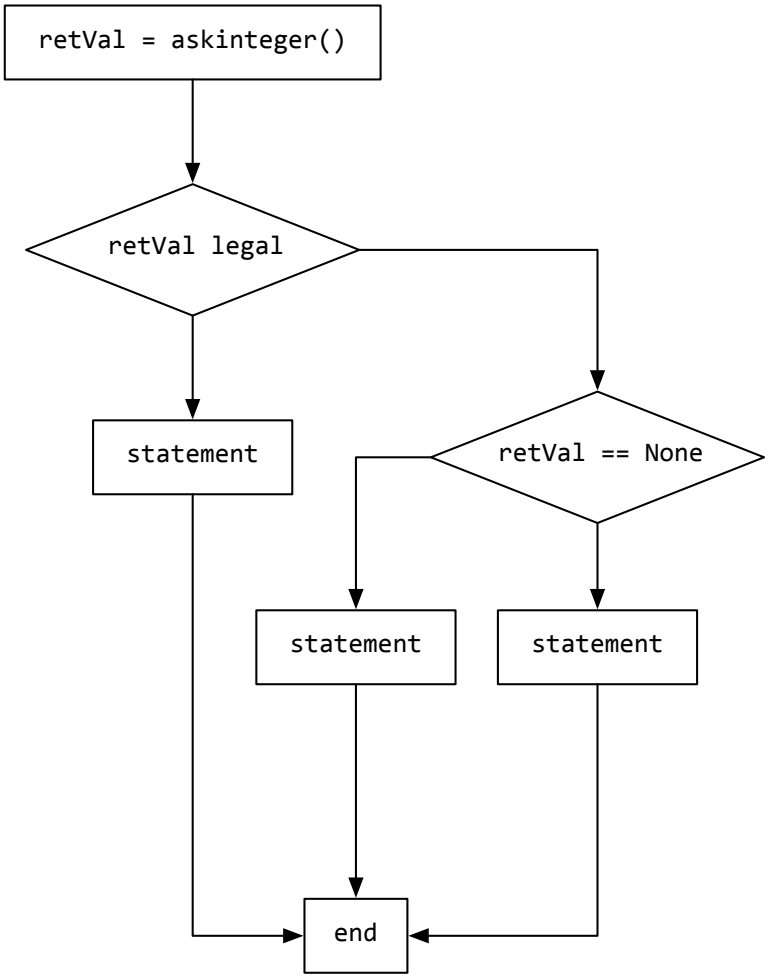


Figure 6: Ask an Integer

### 4.2 Word Segmentation

First of all, we need to separate the input file into sentences. We make a list of all the punctures, and then we search the whole paragraph. If the character is in the list, we split it as a sentence.

We specify the algorithm in the following flowchart:

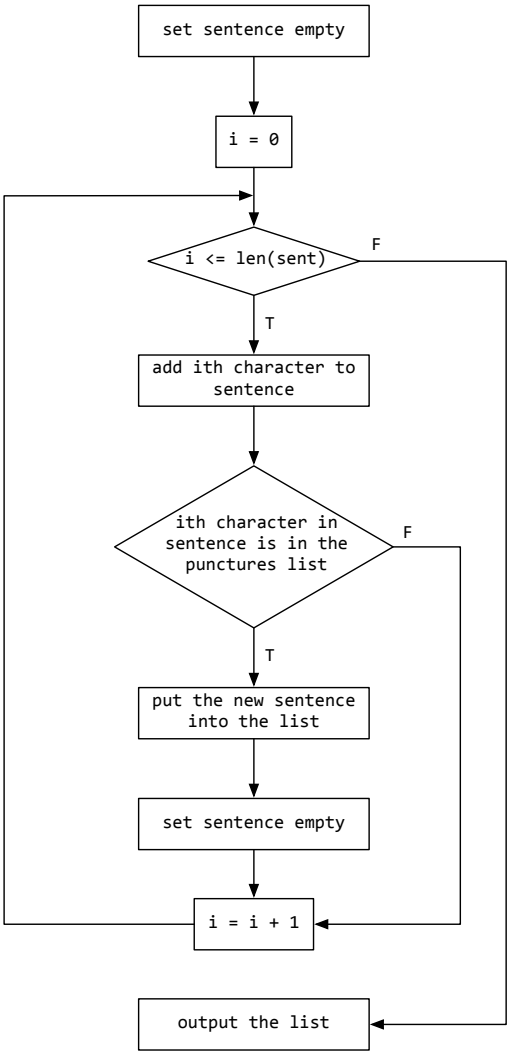


Figure 7: Sentence Segmentation

Secondly, we process with the sentence chosen by the users.

At first, we consider about matching backwards with the largest word

length. But this kind of segmentation exists a great inaccuracy error. For example, if the input sentence is "在青少年中", the output will be "在|青少|年中". Clearly it's wrong. There are a great numbers of sentence having the same situation.

So, we consider about using the word frequency. However, if we list all the possibility of all the segmentation, the time complexity will be amazing. Our solution is to find out some segmentation with greater possibility and use the word frequency to compare and decide which one is the best.

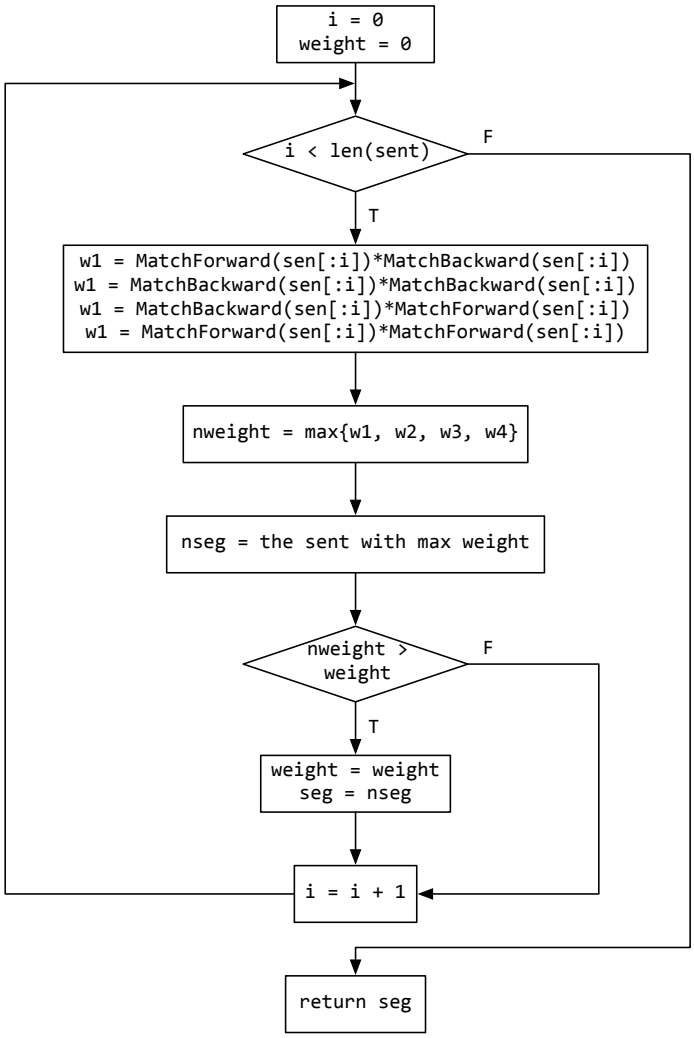


Figure 8: Word Segmentation

Initially, we just think about matching forward or backward with the largest word length. It's not perfect enough, so the final decision is that we lay a point in the sentence. This point moves from the beginning of the sentence to the end of the sentence, and then, matching the part before or after the point with both matching forward and backward with the largest word length. We combine those two parts and get four possibilities. We find out the segmentation with the maximum weight. Then we compare all the weights of all the point position, and this is the final result.

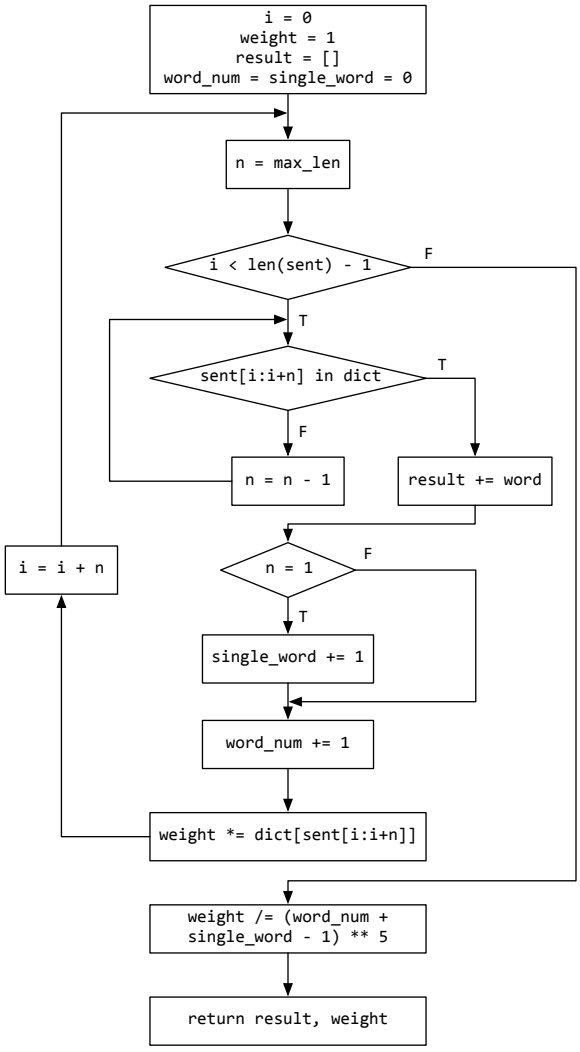


Figure 9: Calculate the Weight

To give more details about calculating weight in Figure 9, we use the following formula to specify the method:

$$W = \frac{\prod_{i=1}^{N_1} \mathrm{tf}(w_i) \prod_{i=N_2+1}^{N_1} [\mathrm{len}(w_i) - 1]^4}{(N_1 + N_2 - 1)^5}$$

where $\{w_i\}$ is a reordered sequence of the result of segmenting a sentence, with word number $N_1$, and $w_1, w_2, \ldots, w_{N_2}$ are single words with number $N_2$, and $w_{N_2+1}, w_{N_2+2}, \ldots, w_{N_1}$ are non-single words.

As most of the Chinese names don' t exist in the dictionary, we need to find a way to improve the program to find names out.

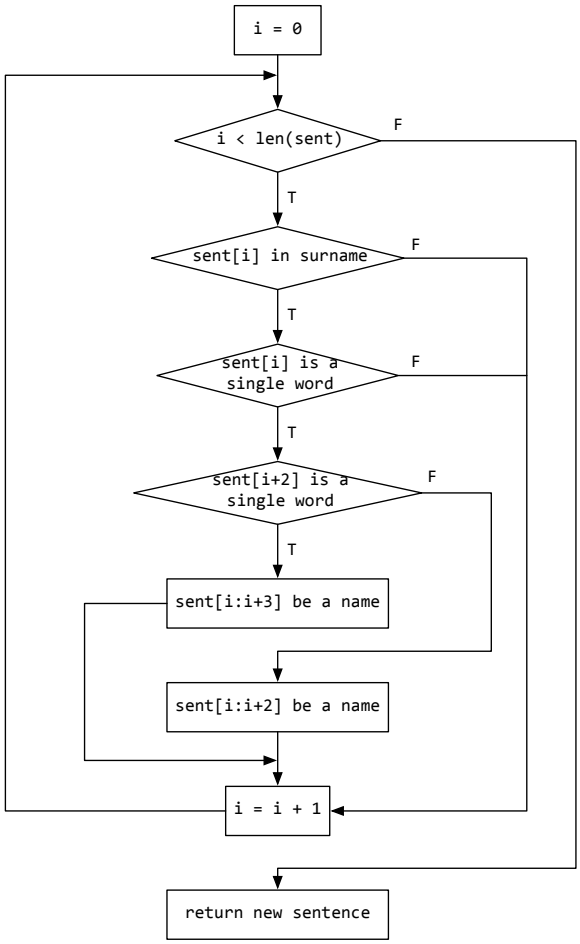


Figure 10: Deal with Single Surname

First, we need to find all surnames in Chinese. The number is not so great that we can list all of them into a file. The surnames are classified in two categories. Then, for both of them, we find out all surnames in the input file and specify whether it can't form a word with other letters. If they can't, we can consider it as a real surname and form a name for it. We skip the next character and judge the singularity of the one after that. If true, we combine it into the name. Otherwise we just combine the former two.

For the difference of those two categories locates at when we judge the singularity of the surname. For compound surname, it's more possible to form a name. So, we just match in backward sequence. While for the single surname, we exam all the possibility of it to make sure it's exactly single. After that, we process the above progress and get the final result.

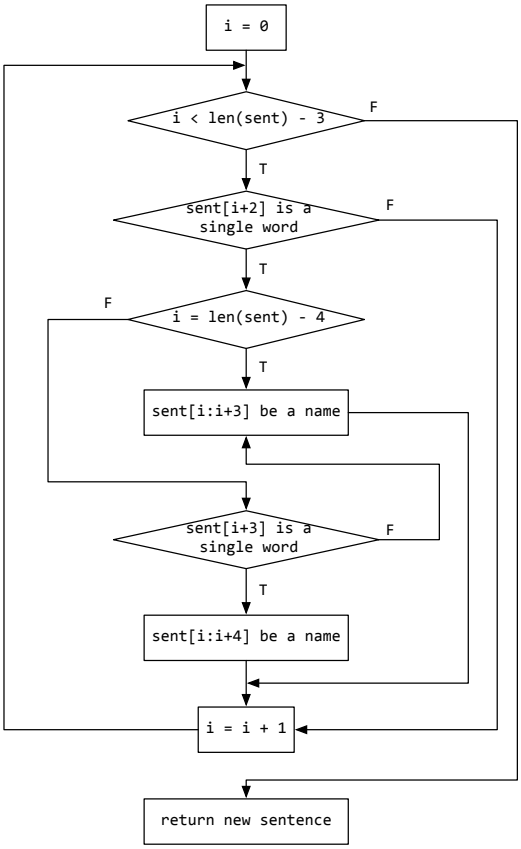


Figure 11: Dealing with Compound Surname

Of course there are various details to be mentioned such as we need to consider whether the last character is a puncture. It will interference the program and get error easily. And when the surname occurs in the first character or the last character, they all need to be specified.

# 5 Demo and Testing Result

## 5.1 Sreenshots

First, you should load the lexicon.


Figure 12: Load Lexicon

After choosing a lexicon, you can input texts, or just open a file.

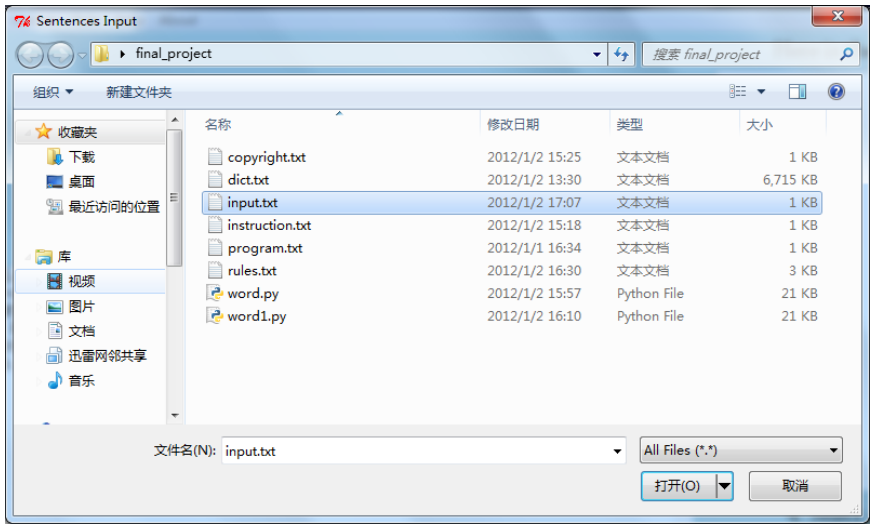


Figure 13: Open a Text File

When you are ready to segment, just click the “OK” button.

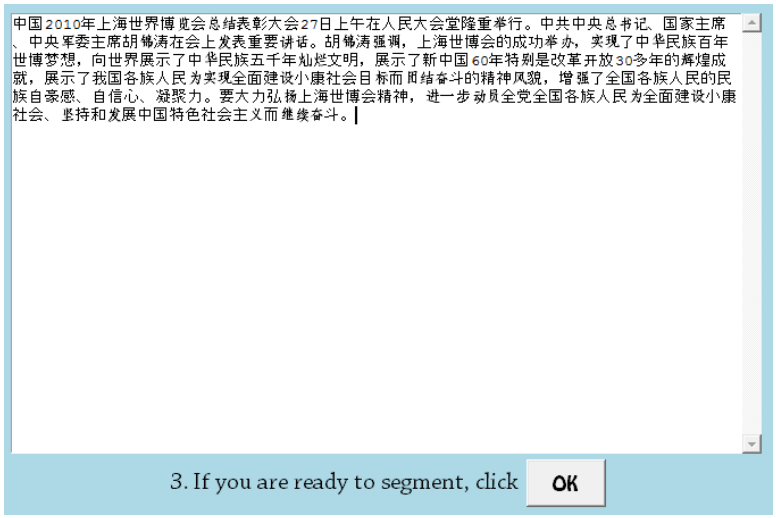


Figure 14: Ready to Segment

And you can see the result of sentence segmentation.

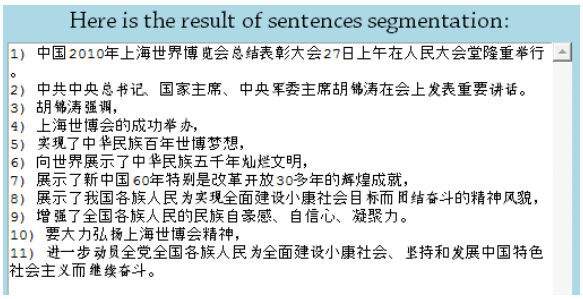


Figure 15: Sentence Segmentation

Then you can click the "Select" button to choose one sentence to segment. Please pay attention to not entering an invalid number (such as float number, or a number out of range).

After clicking the "OK" button, you can see the result of word segmentation in the text box below.

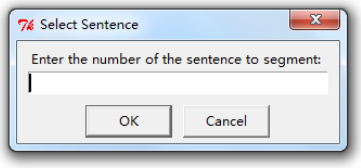


Figure 16: Select a Sentence

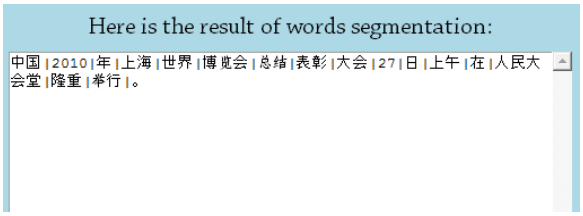


Figure 17: Word Segmentation

## 5.2 Testing Procedure, Data and Result

Since the system is mainly aimed at dealing with the texts extracted from the Expo2010 website (`http://www.expo2010.cn/`), first of all, we choose some articles on the website.

Fortunately, the articles are all categorized in several parts, such as Home, News, Encyclopedia, Pavilions, Activities, Forums, Services, Volunteer, Expo Online, Events, etc. Hence, it is really of great help to choose texts from each of these columns.

Table 2: Testing Data

| Category | Articles |
|---|---|
| News | Premier Wen Declares Shanghai World Expo Closed; China Bids Farewell to Successful Shanghai World Expo; Hu Jintao's Speech; Award Asembly for Expo; Introduction to Expo; Unforgettable Parts; Yu Zhengsheng's Interview |
| Encyclopedia | Rules and Regulations; Logo of Expo; Expo and Technology; Register Report |
| Pavilions | Theme Pavilion; China Pavilion; Jiangsu Pavilion; |

Continued on next page...

| Category | Articles |
|---|---|
| | Zhejiang Pavilion; Japan Pavilion; |
| | Taiwan Pavilion; Shanghai Pavilion |
| Activities | Opening Ceremony; Wang Qishan's Speech |
| Forums | Shanghai Declaration; Lan Feng's Speech |
| Services | Expo Delicious Food |
| Volunteer | Smiling Xiaobaicai; Fudan Volunteer; |
| | SJTU's Volunteer Say Good Night to Expo; |
| | Love Story among the Volunteers; |
| | It's Hard to Say Farewell |

We use these articles as our testing data. It's glad to see most sentences are segmented correctly. However, during the testing process, we still find some errors in our system, some of which are listed below:

Table 3: Error List

| Phrase | Result | Reason |
|---|---|---|
| 是以世博为核心的 | 是以\|世博\|为\|核心\|的 | Combination Ambiguity |
| 后世博时代 | 后\|世博\|时代 | Lexicon Problem |
| 个园之夏 | 个\|园\|之\|夏 | Lexicon Problem |
| 各大镇馆之宝 | 各\|大镇\|馆\|之宝 | Overlapping Ambiguity |
| 以物联网为契机 | 以\|物\|联网\|为\|契机 | Lexicon Problem |
| ... | ... | ... |

This is the disadvantage of the system, and that is to say, it can not always give the result.

We can solve some of the problems quickly. For example, the reason why "以物联网为契机" is segmented as "以|物|联网|为|契机" is that the lexicon doesn't conclude the word "物联网". Therefore, we can correct this error by adding the word "物联网" into the lexicon.

However, some other problems are not so easy to solve. For instance, if we are given the phrase "是以世博", there are probably two ways to segment it – "是以|世博" and "是|以|世博". We won't know which is the correct one until the context is given. To solve this, we need to design some new algorithms in our program.

# 6 Conclusion

Chinese word segmentation is a quite difficult problem in Chinese language processing. In spite of this, it is still necessary to try to solve the problem, for it is the first step of natural language processing in Chinese.

Many individuals, teams or even companies have put much effort in researching the Chinese word segmentation algorithms, and they have achieved great progress. Compared with them, our system looks a little weak, but our team treasures this lovely system because they are developed by ourselves and we have taken great pains. To us, it has significant meanings. We have been energized by applying some useful technologies in our "tiny" program and getting a "big" result, and that really makes all of us excited.

Also, we have learned a lot from this project. Matching strings, applying word frequency, using TKinter module, all these give us a deeper understanding on Python and the method of programming.

All of us have to admit that developing such a system is of no ease. Of course we have encountered many difficulties. Some are really big deals, which makes us deliberate a question again and again, and solving them finally inspires us a lot. Some are so called "peanut problem", but we don't agree with this facetious term. Small mistakes could eventually lead to serious results, which is enough to give a caution to us.

Without help we can have never done this job. Thanks to our professor, our classmates and anyone who supports us. Special thanks to SCWS, a free and open source Chinese word segmentation system. We construct our dictionary on the basis of yours, and this turns out an amazing effect in the final result.