**Report**

邓苏鸣 1730026014

邹罗瑾 1730026167

周寒雨 1730014088

# Text files collection

I’m mainly responsible for text files collection in our project.

My dataset has two part, one is for test and another is bigger than 200MB.

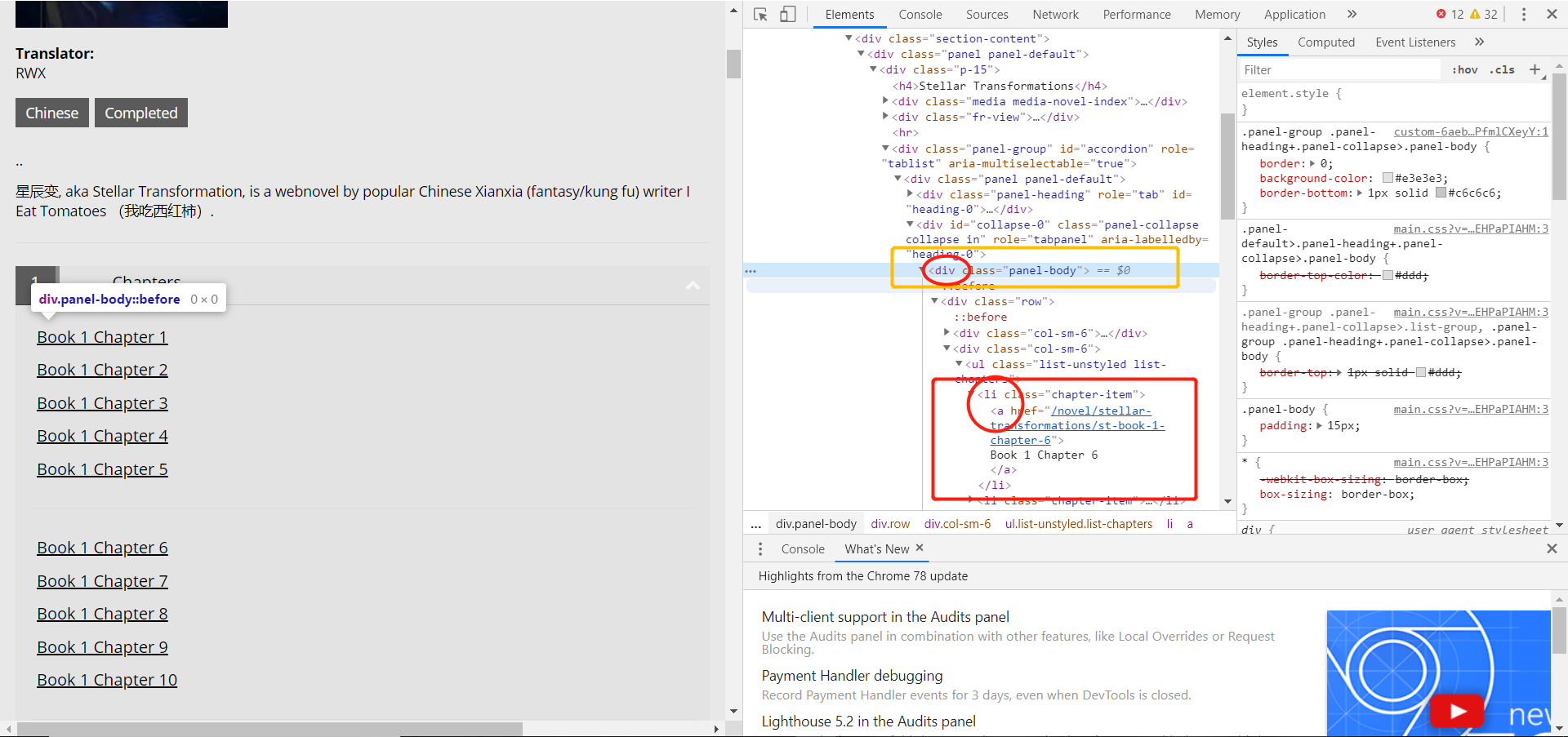
## Test part

Firstly, we find our target novel：stellar-transformations.

The ‘URL’ is https://www.wuxiaworld.com/novel/stellar-transformations

### FetchUrl(url)

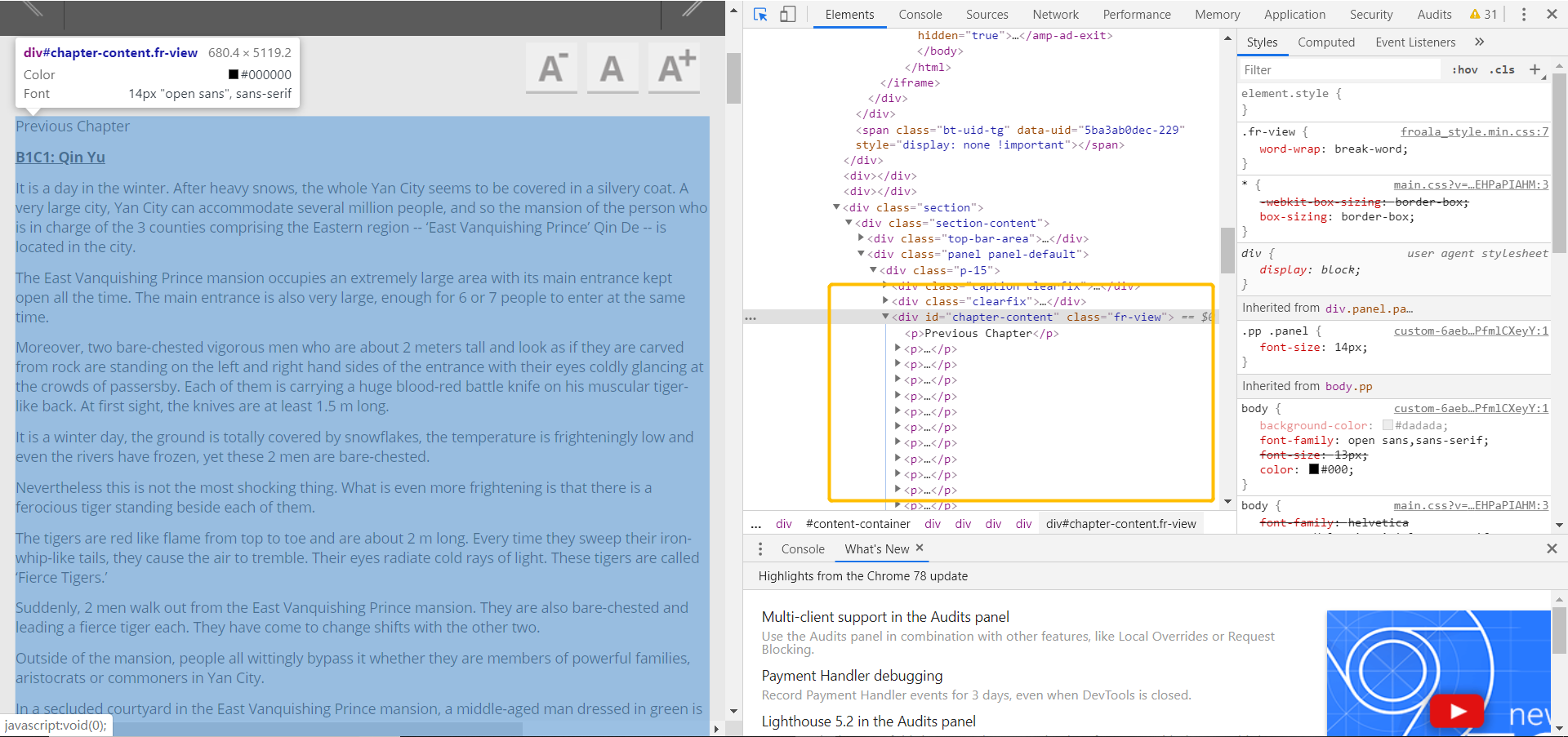
Right click and check, we can locate the corresponding element in the source code. In this way, we follow the path: div class: panel body, we know that the link is in the class: chapter-item. The tag is “a”.

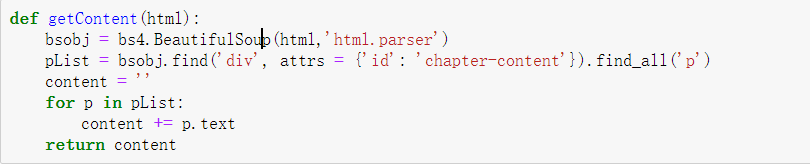




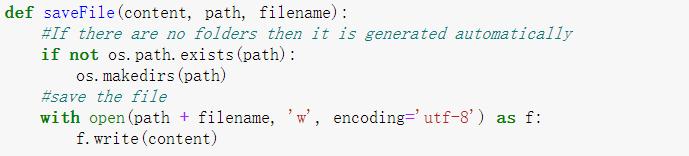
### getContent(html)

Now that I have found the links for each chapter. What I need to do is to extract the text elements from the page and save them in a TXT file. As you can see, every piece of text exists in <p>, so we just need to read all the text information under <p>, and then we can save it as a section.

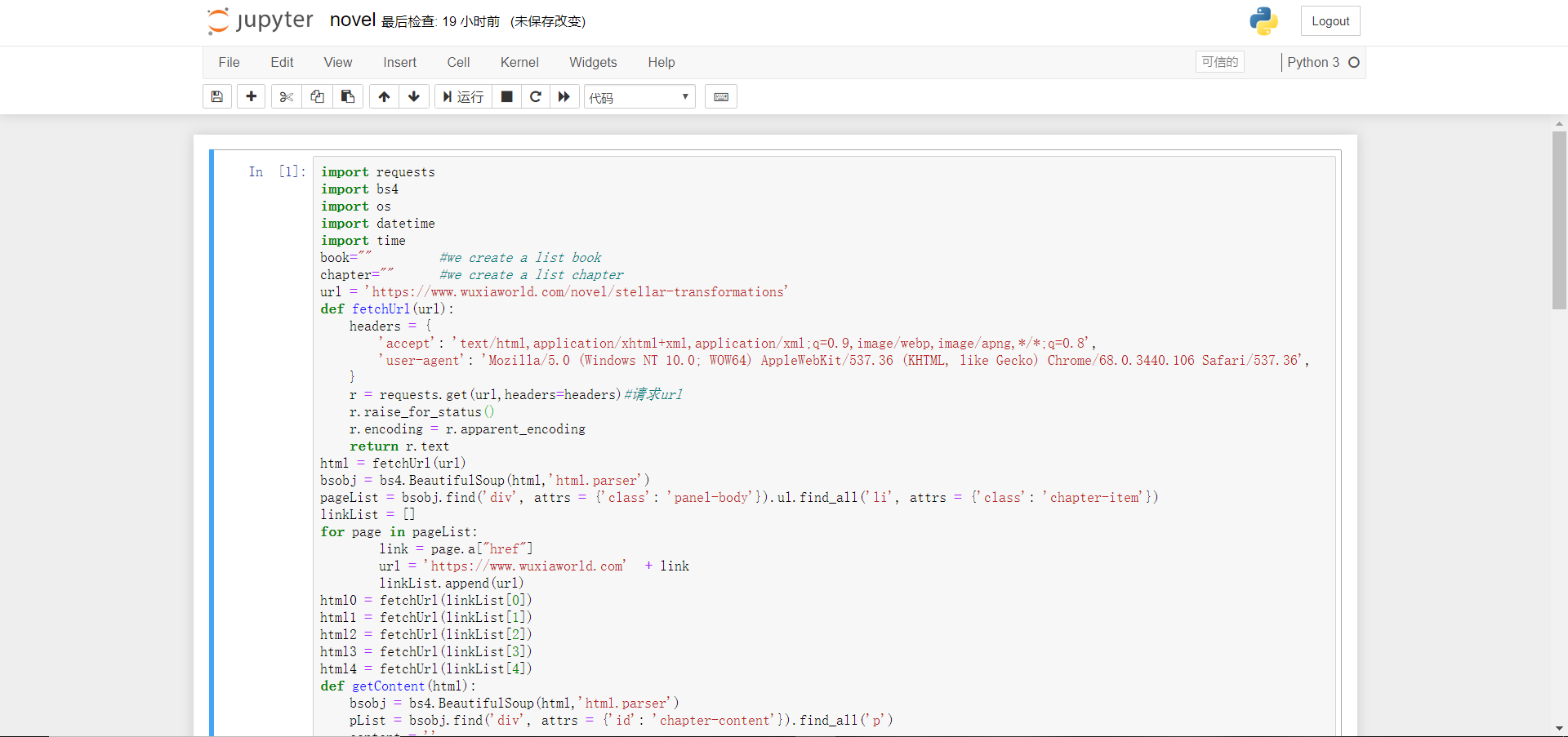


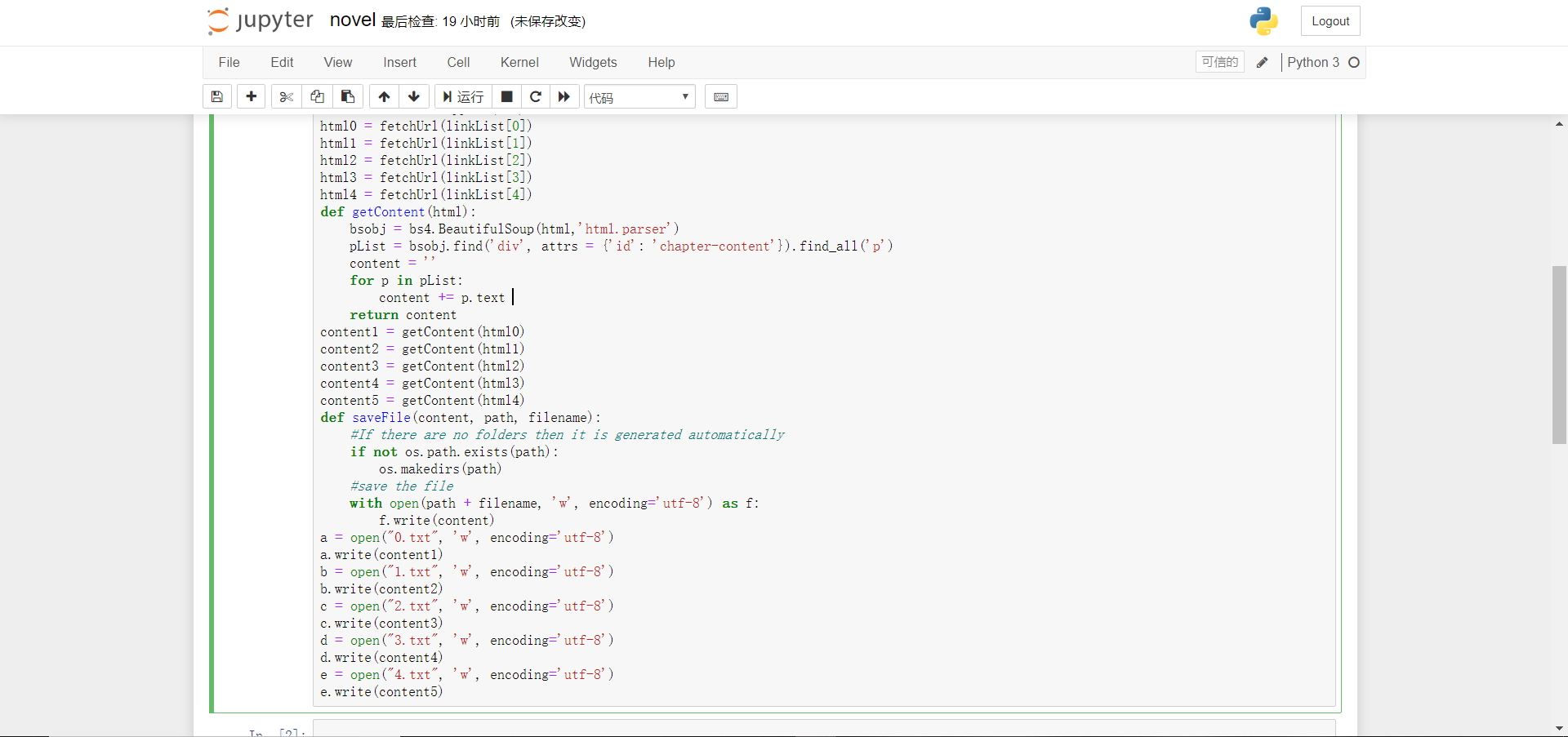


### saveFile

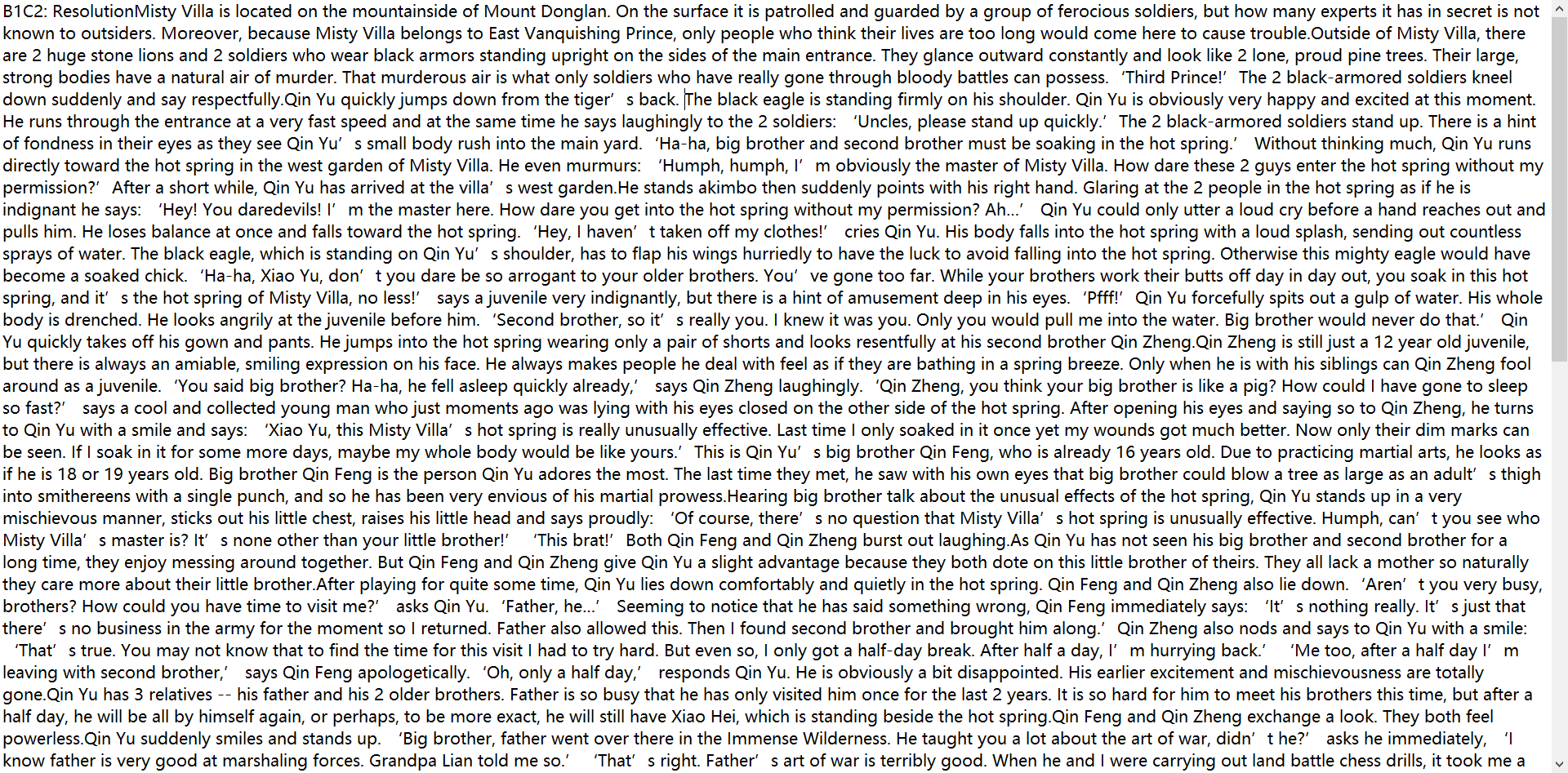


### All code and some results:







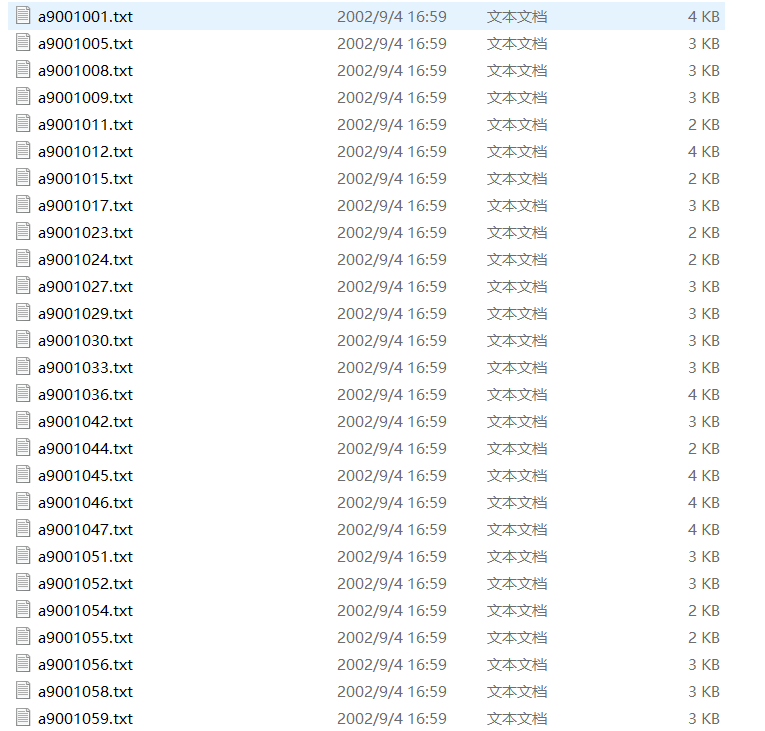
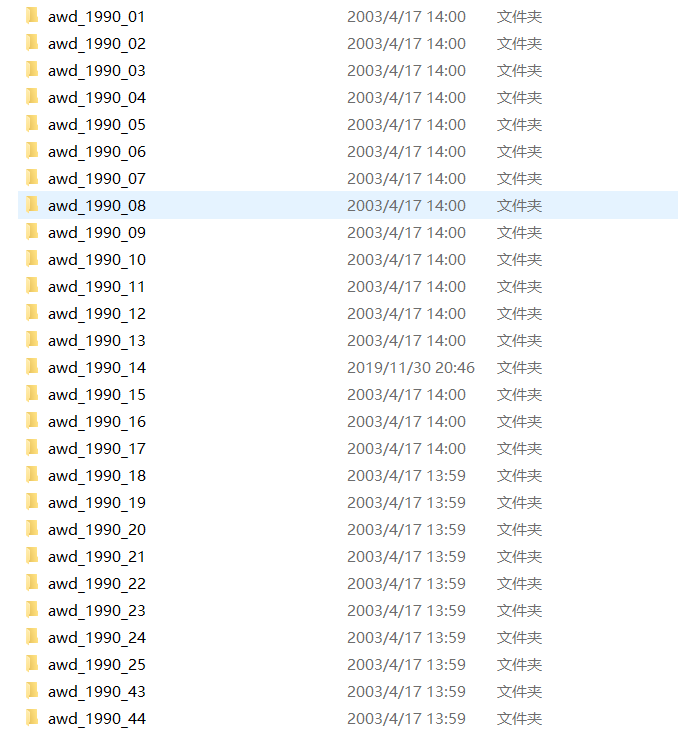
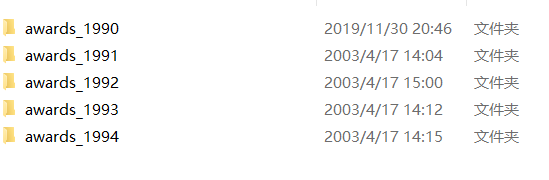


I think my code is a little stupid because I basically craw the article one by one. I can write some for loop to make my code more effective.

# >200MB dataset(steps and problems)

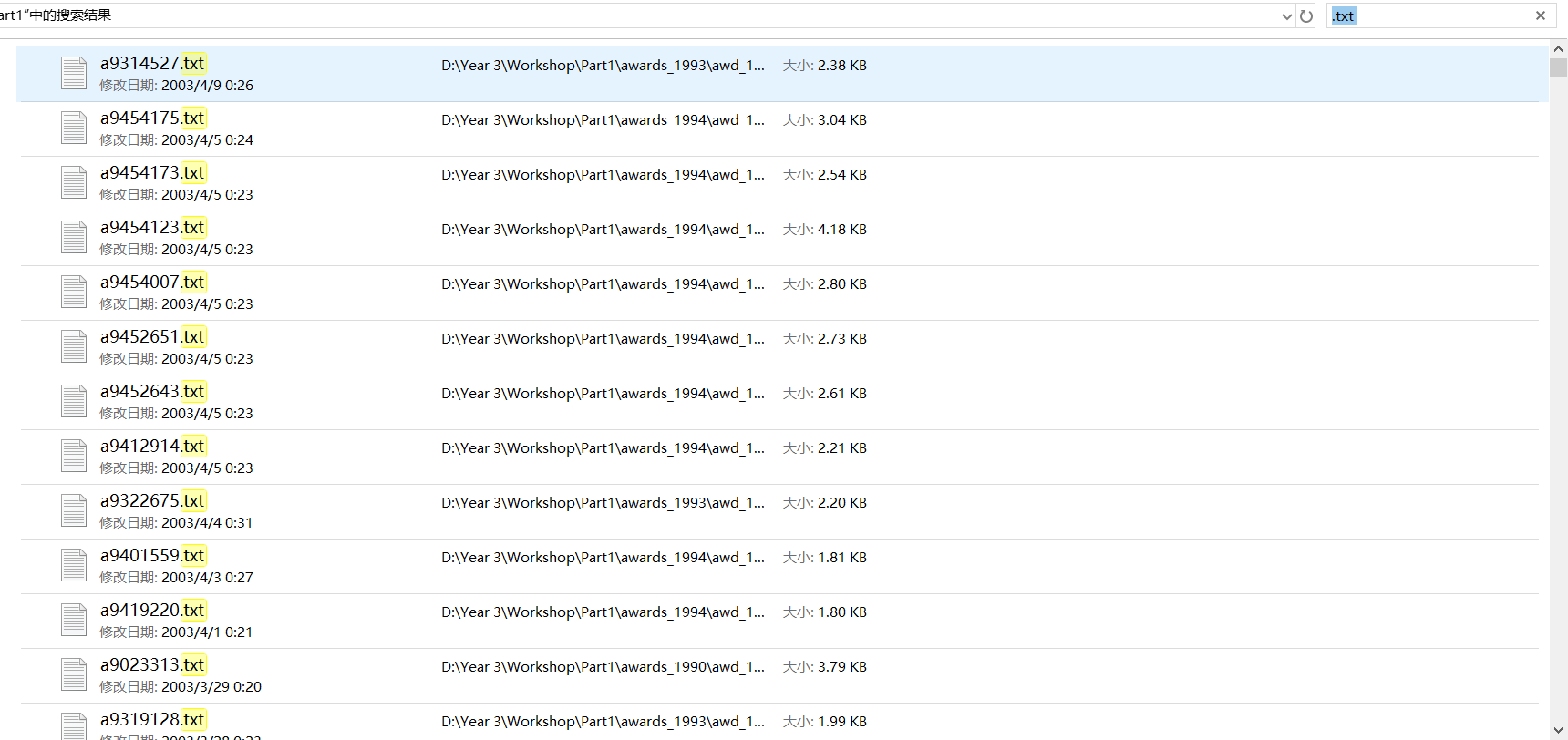
Firstly, I try to find data from two links of Wikipedia. However, the school VPN is not work. Therefore, I can only try the third links. This link has a bunch of text datasets such as the “Reuters-21578 Text Categorization Collection” and “Twenty Newsgroups” in the following lists.

**Then I find my first dataset in regard to award winning projects in the field of science:**



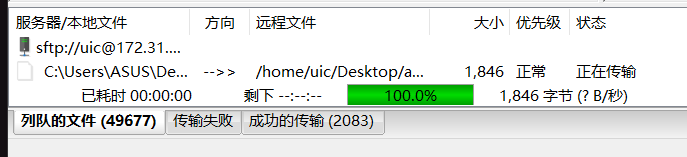
## The folder has a bunch of subfolders

Here comes the first problem. I have to get a file, which can only have the txt files. However, there are a bunch of subfiles, which is not suit for our project. I got an idea, I type the “.txt” in the search box and ctrl-A, ctrl-C to copy them to another file. Finally, I got a file, which has a bunch of txt files:



## 50000+ TXT Files number is too big

Here come the second problems, even the dataset satisfies the requirements, which is bigger than 200MB size, but it has 50000+ text files. In this case, at least fifty thousand rows will be built. The transfer is slow and the index values can appear in the tens of thousands row. The local server will explode. Just like this:

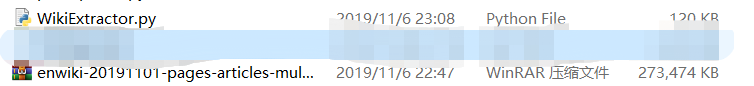


## Cannot extract the RAR from Wikipedia

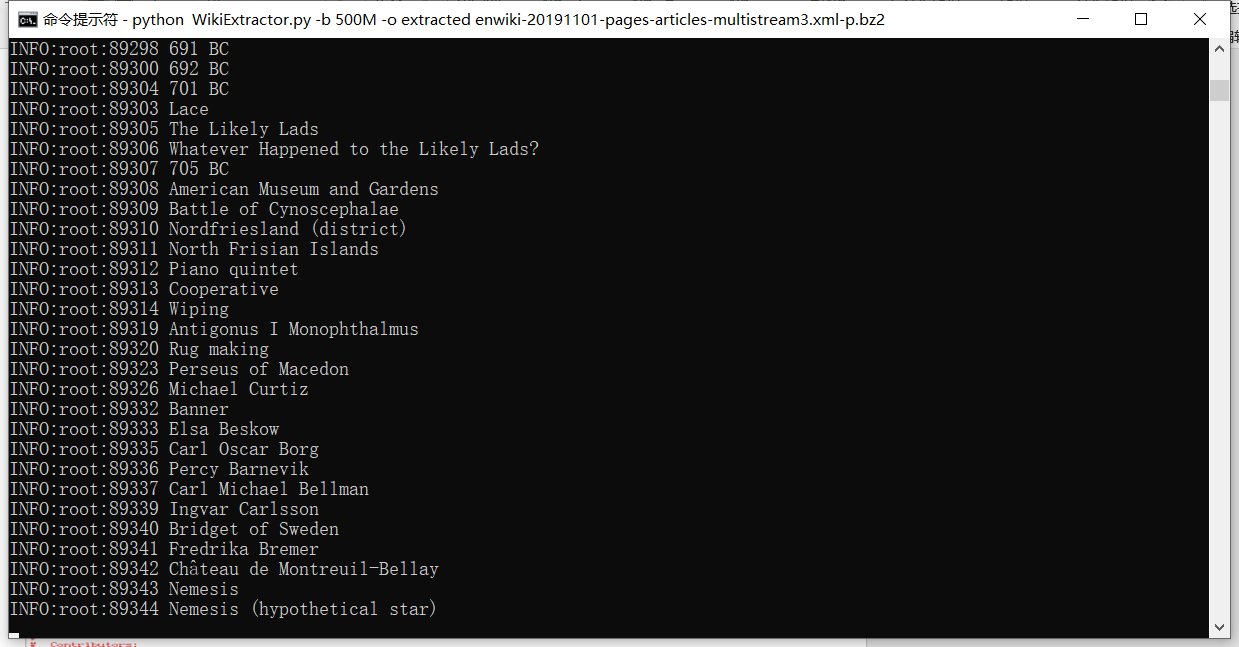
**Then I have to find the second data.**

I bought a VPN and access to the Wikipedia link. I found a proper dataset and try to download it. However, if I want to decompress the dataset of Wikipedia, I have to use python to write a program to extract it. Therefore, I found a python code of extractor to extract the data.

The steps: 1. The files are in the same folder as wikiextractor.py 2. CMD enters the file directory 3. Perform: python WikiExtractor.py -b 500M -o extracted enwiki-20191101-pages-articles-multistream3.xml-p.bz2









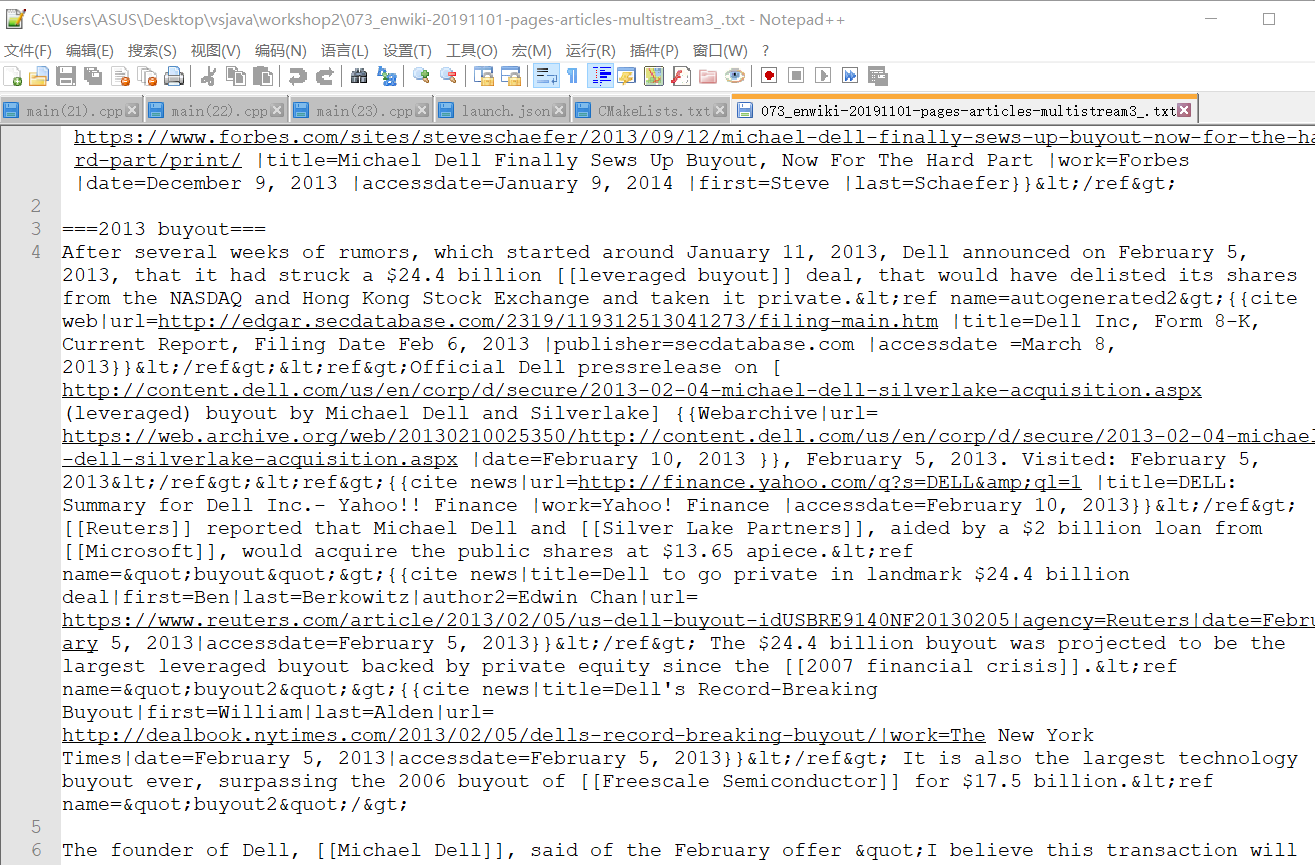
## How to divide a big TXTfile into some small parts

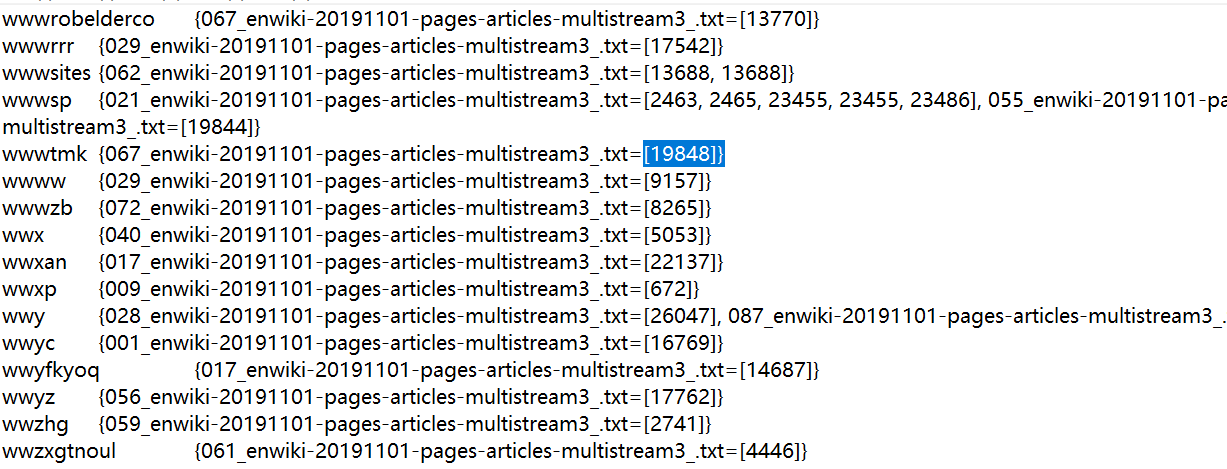
The text file is too large, which is bigger than 1GB, so I download an app called TXT divider to divide the whole txt file to some proper size.



## Some problems of article format

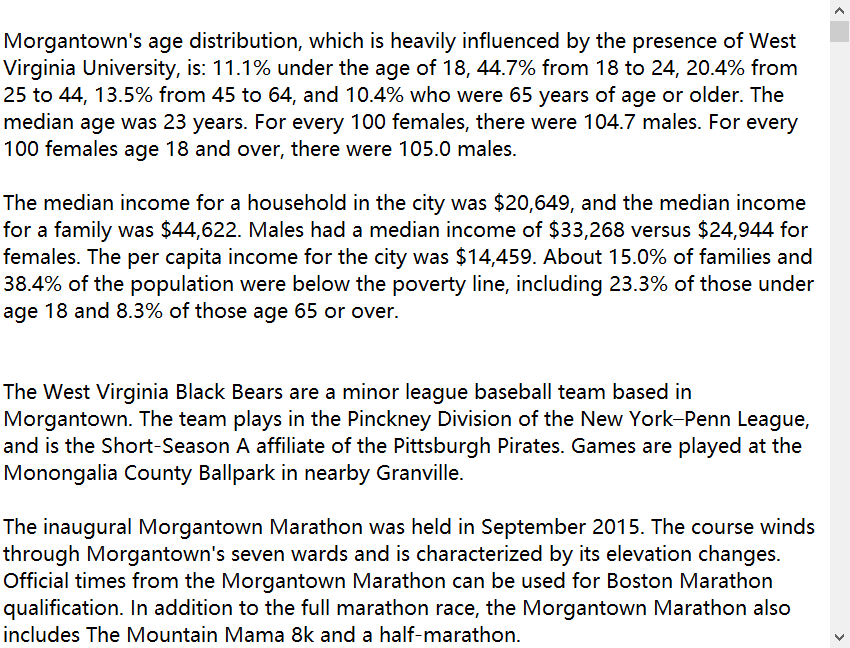
However, When I open some of the TXT file to have a look, I find that some article has a lot of website and messy code. By the way, there are some problems of article format, which will affect my partner’s work. My partner said: 1. There are a lot of strange index, which we don’t know. 2. A line in the page is a section in TXT, my TXT file is too long, so the page cannot be displayed



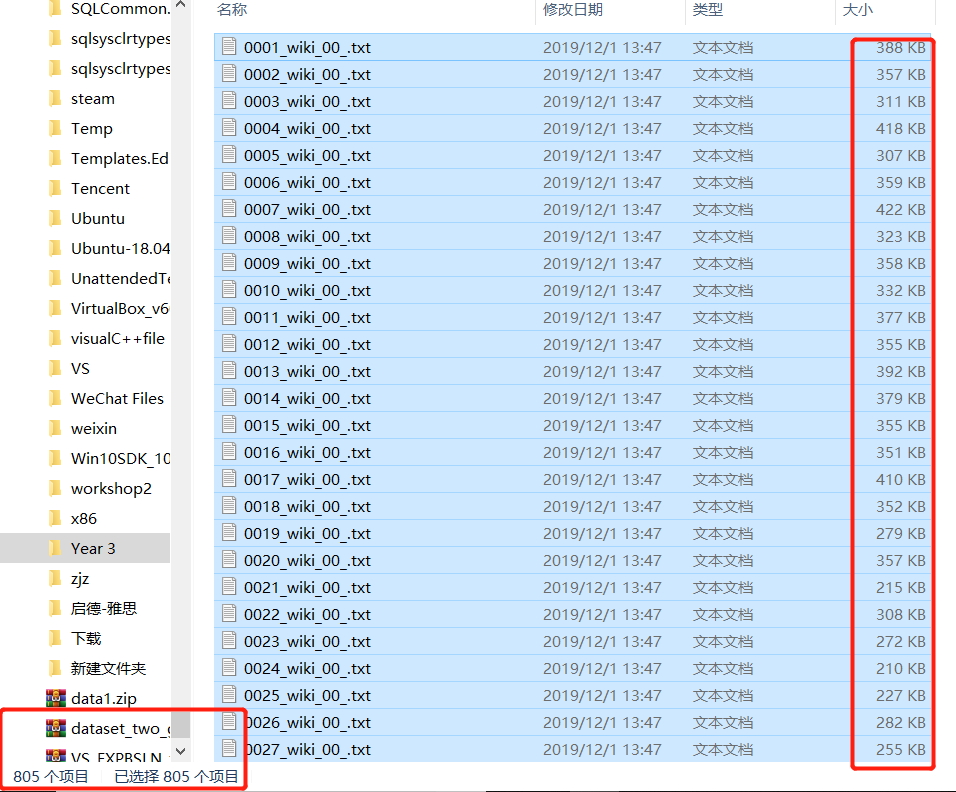


**That’s fine, I started to find my third dataset.**

I use the same way to acquire the dataset from Wikipedia, which is about the “Articles that often search for key words”. The difference is the division method. The last time my division method is according to the words number. This time my division way is based on the line number.



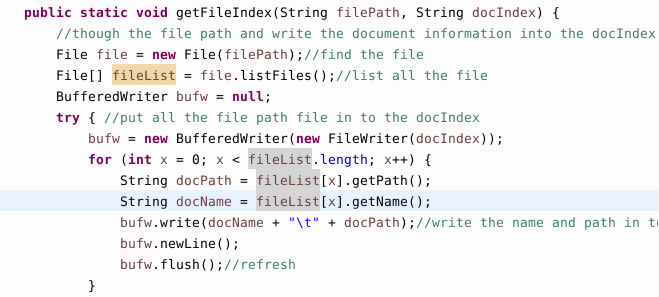
This time the TXT file is perfect. The size and the number are proper.



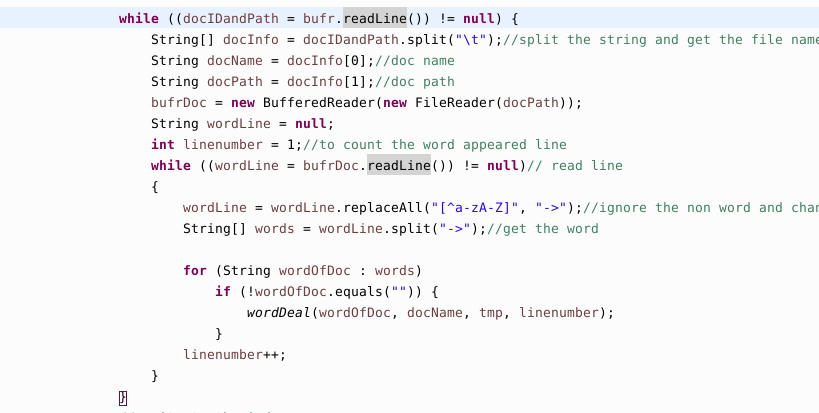
### Index

In this project, I do the building index job.

When I do the index which run in a singer computer. At first, I should find a structure to store the value. Then I search on the internet and find the java map value. It is like (key, value). Then I want to put the key as the word, and value is an Array list which to store the word appear line number. At first, I write a function to read all the file in a folder. And then build filename and document path in the “docIndex” file.



Then I know the location of the file and the file name, I can use the program to open this file and read by line. Use the regular expression to split each word.



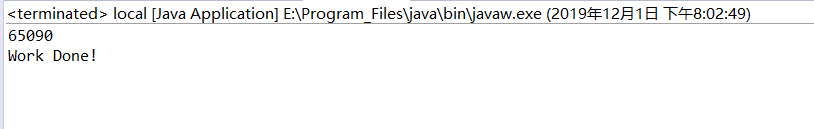
And finally, write it to the word Index document.

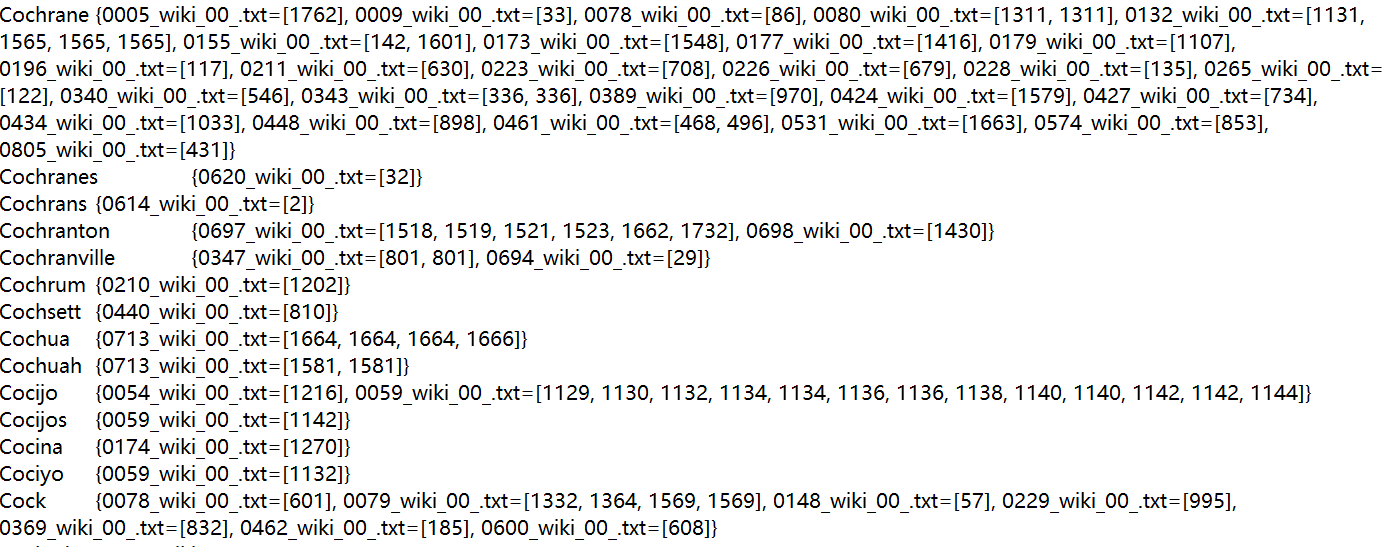
I can run successfully on my own computer, but when I upload on the Linux and run the 212m files. But it can’t run.



I think it is because of the Linux memory is small. I try the internet method to change the eclipse.ini file to increase the use of memory. But I also fail.

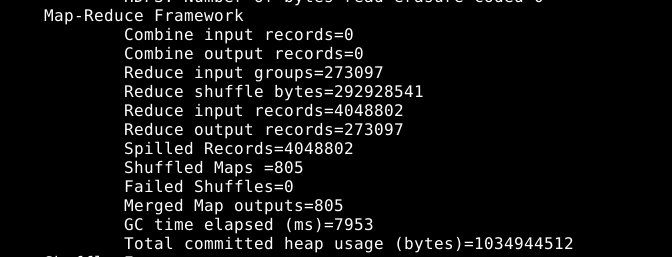
On my own computer, it cost 65090ms and it can run successfully.

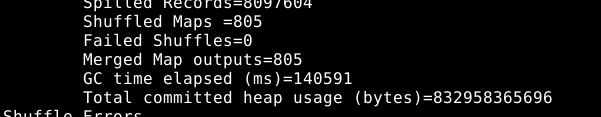




I have checked the line number on the file, and it is correct.

In the second task, it is use the Hadoop to create the word index. It is the same as the Assignment 4. Put the file+word and linenum as the (key, value). Then do the job2 to do the partition word and file+linenum as the (key, value) and write it to file.

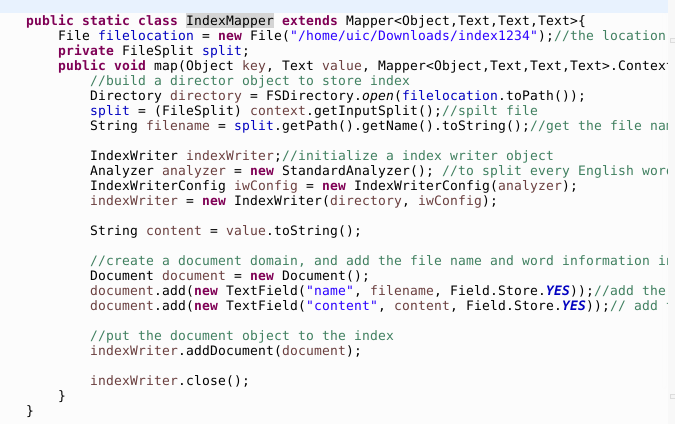




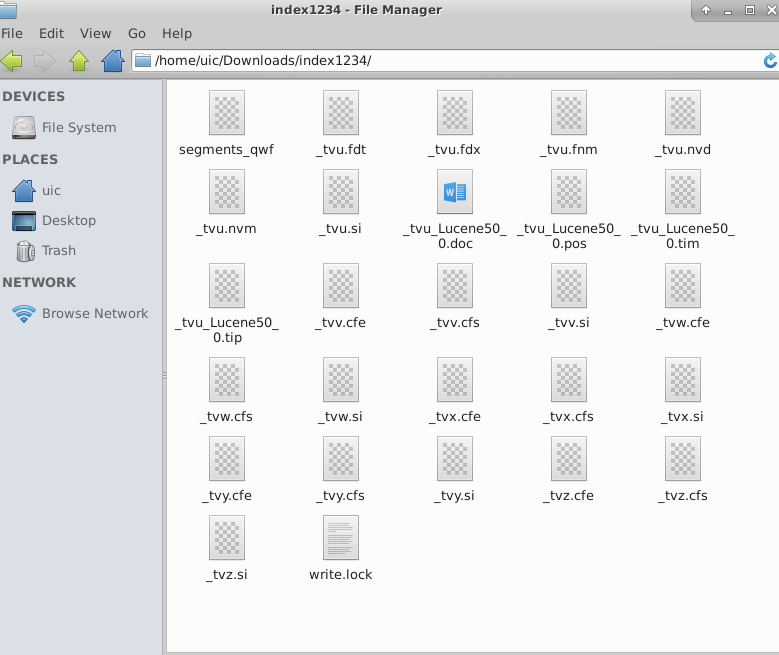
In the third task, I should use the Lucene + Hadoop to do the index job.

It is mainly having several steps, create an directory object and determine the location of the index library. Then create an Index Writer object. Read the file and create document object on every document. Then, add the field to the document object and finally, write the document object to the index library.

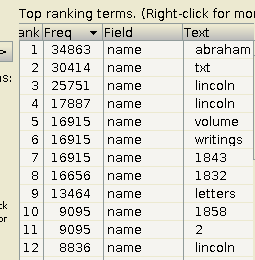
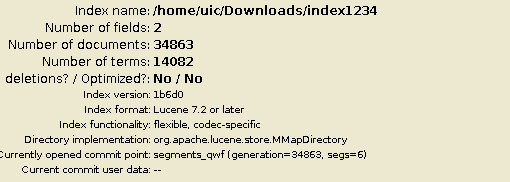
In the Hadoop, I write the lucence program in the map, at first, build the index library on the local place, then split the context to get the filename, then, use the Lucene standard analyzer to split the word and build every word index in the index library.



But it run very slow, I think it is maybe because the Hadoop put every word objective to the index library, but if on the local, it usually put the document objective to the index library. A document has many words, so it wastes many times. I have not use the 200m file to test but use the 2M file to test whether my program is correct.

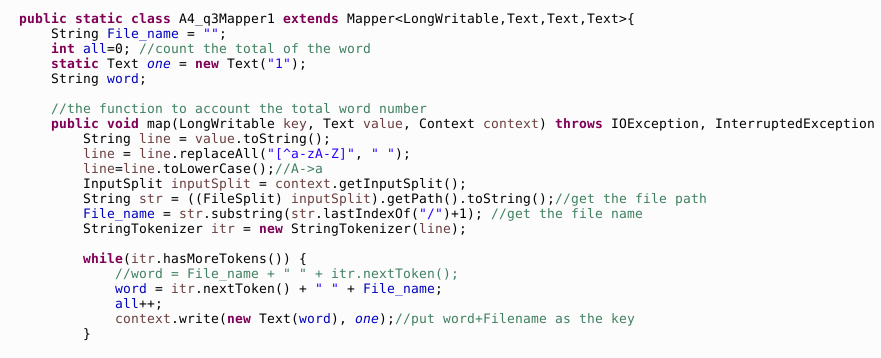


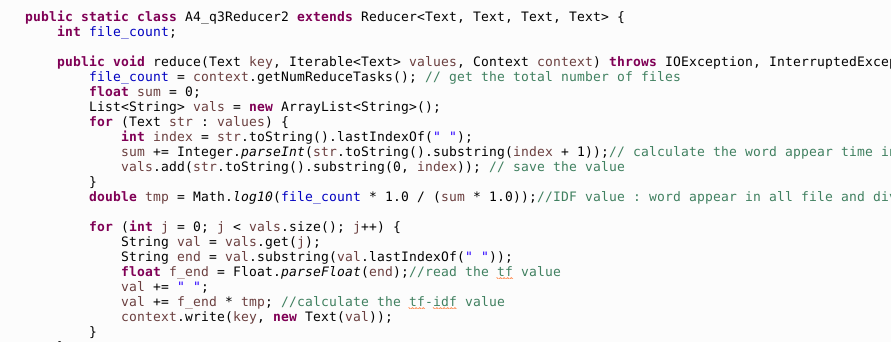
When open in the luke, it shows



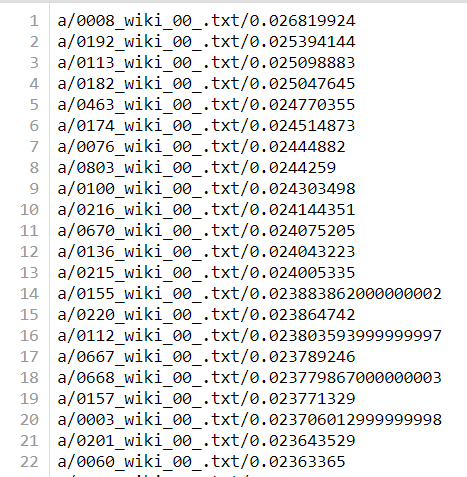
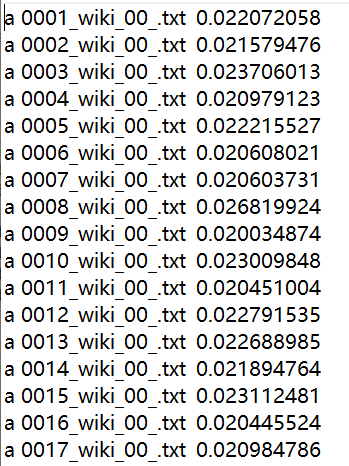
It runs correct.

In the ranking part, since our group data is txt file, so I use the TF-IDF to calculate the word weight. At first, for each input file set, calculate each word appeared times. Output is the <word, the word appeared time>. Then, calculate the number of all words. Out put is <word+filename, appeared time+all word number>. Finally, statistics the number and calculate the TF-IDF value.



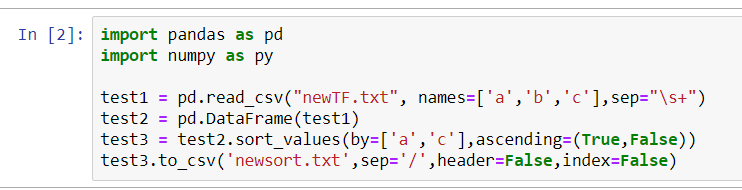


The final TF-IDF result is like this:

(right is sorted the value)

Since I only calculate the TF-IDF value, I use the python to do the second treatment. And select the largest TF-IDF value appear on the top.

The python code:

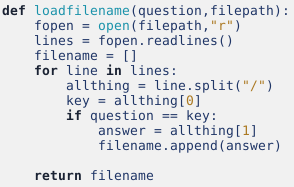


I select the three columns as ‘a’, ‘b’, ‘c’. and sort the word and TF-IDF columns. True is mean sort by the alphabetical order. And false is mean is value is descending.

### Django

After we have the TF-IDF result which already do the ranking process, we start to use Django to do the process of searching result and presenting the result in the interface.

1. **Find the name of the file**



**Function:** This function is used to find name of all the file that emerge the searching word.

**Parameters:**

* question: It is the searching word.
* filepath: It will input the TF-IDF result’s location in to it.

**Details:**

* for line in lines

It traverse all the line in the essay.

* allthing = line.split(“/”)

It split a line into three parts ---- keyword, filename and the value of TF-IDF, respectively.

The reason we use the “/” to split is that in our TF-IDF result, we use “/” to distinguish the keyword, filename and value of TF-IDF. The txt file’s template is like this:



* key = allthing[0]

The first elements in allthing is the keyword.

* if question == key:

If this line’s keyword is equal to the value we want to search, then filename in this line is what we want.

* answer = allthing[1]

As the picture of txt file’s template showed, the second element in allthing is filename.

* filename.append(answer)

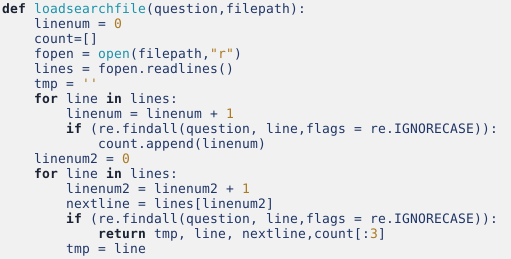
Since our result of TF-IDF already do the ranking process, so the order that append into filename is according to the number of TF-IDF. For example, if the filename called ‘a’ is the first one that append, then it means that ‘a’ has the maximum TF-IDF value.

**How to use this function:**



Note: The parameter of post\_data is the searching word.

1. **Find the line number, front, current and back context**



**Function:** This function is used to find line number, front, current and back context in one file that emerge the searching word.

**Parameters:**

* question: It is the searching word.
* filepath: It will input the location of the file that has the searching word.

**Details:**

* The first for loop: for line in lines

It is used to record all the number of the line that has the searching word.

1. re.findall(question, line, flags = re.IGNORECASE)

It is used to find whether there are some words that equal to the searching word in this line.

1. count.append(linenum)

If this line has the searching word, then the number of this line will be appended into count. Therefore, count list will record all the number of the line that has the searching word.

Note: Since some lines in our txt files contain too many words, the interface that we linked could not contain too many words in one lines. Therefore, the line number is not the line number in the hyperlink essay.

* The second for loop: for line in lines

It is used to record the front, current and back context that emerged the searching word in the first time. In this function, as long as we find one line that has searching word, it will not run the ‘for’ loop continually.

1. Linenum2

It is used to find the index number of back context.

Note: Since the count store all the number of the line that has the searching word, it will be really large, if we print all of it, it may occupy all of the screen.

**How to use this function:**

* Firstly, we try to use this way to run this function:

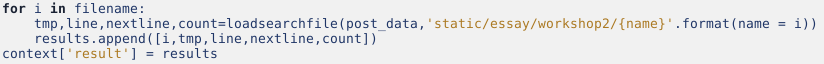
for i in filename:

context[‘tmp’], context[‘line’], context[‘nextline’], context[‘count’] = loadsearchfile(post\_data, “static/essay/{name}”.format(name = i))

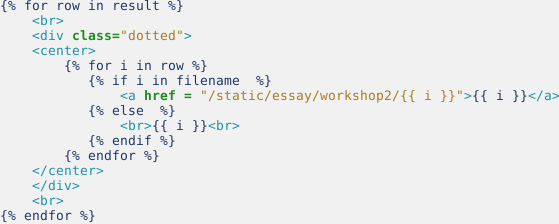
It just store the last file’s information about it.

* Secondly, we thought that we can append the filename, line number, front, current and back context into one list, and then we print the element in result in turn.

views.py



result.html



**Highlight:**

We try to use two ways to highlight the searching word, however we failed.

1. views.py

****

Reason: Index out of range.  
2. result.html



We found it in the Internet. (<https://www.cnblogs.com/dtdxrk/p/3507875.html>)

However, we found that it will highlight the front and back context.