Mini Search Engine

Date: 2019-03-20

Chapter 1: Introduction

1.1 Problem Description

In this project, we created our own mini search engine which can handle inquiries over "The Complete Works of William Shakespeare" (http://shakespeare.mit.edu/).

This project has the following features

(1) Run a word count over the Shakespeare set and try to identify the stop words (also called the **noisy** words)

(2) Create an inverted index over the Shakespeare set with word stemming. The stop words identified in part (1) must not be included.

(3) There is a query program on top of our inverted file index, which will accept a user-specified word (or phrase) and return the IDs of the documents that contain that word.

1.2 Test Environment

1.2.1 Hardware

A laptop with 8th generation i7 quad-core CPU.

1.2.2 Network Connection

During the local test, due to the GFW, it is almost impossible to get connected directly.

```
[nltk_data] Error loading punkt: <urlopen error [Errno 60] Operation
[nltk_data]     timed out>
```

To minimize the influence of the network, I connected my laptop to a ShadowSocks server in Los Angeles. (ping: 65.397ms) And as the test starts, it works very well. And as we find it out later that, the size of every node is roughly 100KB, so the connection speed won't affect much. The slowest step of the test is the algorithm itself, indicating a stable connection is enough regardless of the connection speed.

```
[nltk_data] Downloading package punkt to /Users/fenghexu/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/fenghexu/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
allswell/
['allswell/full.html', 'allswell/allswell.1.1.html', 'allswell/allswell.1.2.html', 'allswell/allswell.1.3.html', 'allswell/allswell.2.1.html',
```

1.3 Peer Review Instructions

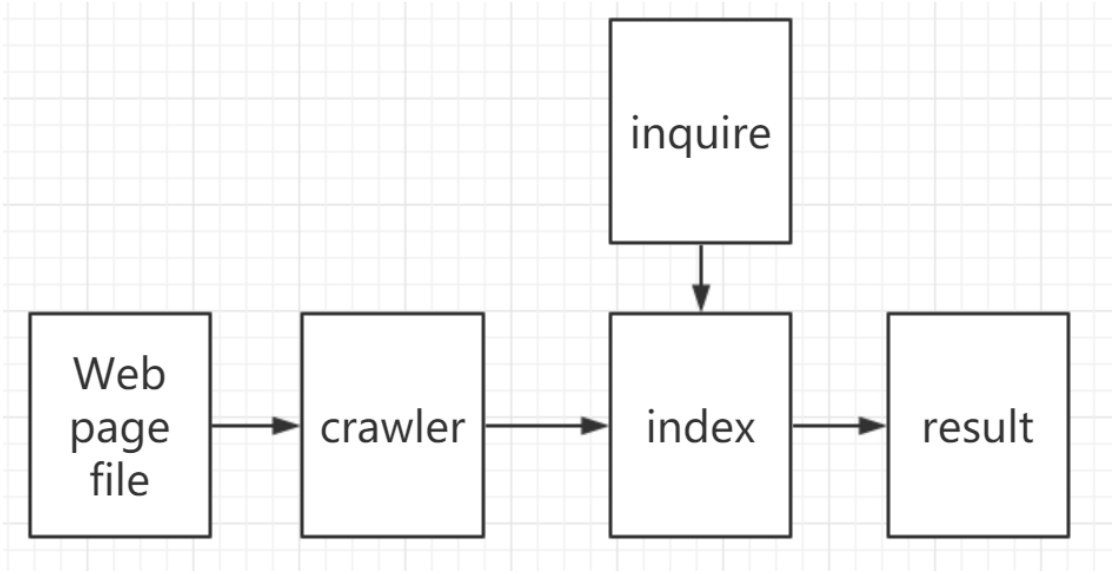
Since the index file are too big to be uploaded to PTA, we had got to

delete all the index before submission. Nevertheless, the venv file is provided, so you are safe to generate the index again by your own, simply by running the command **python shakecrawler.py** in bash or terminal. And keep it in mind that you have to have all the dependencies installed rather than only the source code.

Go to http://shakespeare.fenghe.info if it is available now since reconstructing the index can take a long time.

In command line, **flask run** will enable the local web server.

Chapter 2: Data Structure / Algorithm Specification



The crawler crawls the webpage for content extraction to generate the text file, and then process the index to form the index library. After the user enters a query, the search module searches from the index library and returns the result.

2.1 Main Data Structures – Index

```
class Node:
    def __init__(self, is_leaf=True, file_index=-1):
        self.file_index = file_index
        self.is_leaf = is_leaf
        self.n_keys = 0
        self.keys = []
        self.child = []
        self.child_index = []
```

Index – B Tree with LRU Buffer

During the initial attempt, I set the degree of B Tree to 30, which turned out to be a bad idea. As you can see later, in a much better situations, when the degree of the B tree is set to 300, it takes about 1.5 hours to index the Shakespeare website.

2.2 crawler

By observing the website before designing the Crawler, we found that full.html is a full replication of separate documents in a sub-directory. So we were able to cut down the cost of time by half by skipping full.html during indexing the webpage while showing it when the user searches a keyword.

```
1.  class Crawler:
2.      def __init__(self, base_uri):
3.          self.base_uri = base_uri.strip('/')
4.      def crawl_html(self, path):
5.          ctx = ssl.create_default_context()
6.          ctx.check_hostname = False
7.          ctx.verify_mode = ssl.CERT_NONE
8.          html = urllib.request.urlopen('http://{0}/{1}'.format(self.base_uri,
    path.strip('/')), context=ctx).read()
9.          return html
10.     def get_links(self, html):
11.         if type(html) is bytes:
12.             soup = BeautifulSoup(html, 'html.parser')
13.             links = []
14.             tags = soup('a')
15.             for tag in tags:
16.                 links.append(tag.get('href', None))
17.             return links
18.         if type(html) is str:
19.             return self.get_links(self.crawl_html(html))
20.         return 'Error Relative Links & Bytes Supported Only'
21.     def get_relative_links(self, html):
22.         if type(html) is bytes:
23.             soup = BeautifulSoup(html, 'html.parser')
24.             links = []
25.             tags = soup('a')
26.             if not tags:
27.                 return None
28.             for tag in tags:
29.                 href = tag.get('href', None)
```

```
                if href and 'http' not in href:
                    links.append(href)
            return links
        if type(html) is str:
            return self.get_relative_links(self.crawl_html(html))
        return 'Error Relative Links & Bytes Supported Only'
```

2.3 Function read_index & write_index:

Keeps track of the LRU(Least Recently Used Algorithm) list, when the buffer list is full, oldest node in the list is written to disk and a new one is retrieved from the disk. These disk operations together with the LRU buffer guarantees the both the read and write efficiency of the search engine.

2.4 Function Search

Find the node vertically and then horizontally(logarithmically and then linearly)

If not found, return NULL(None in Python)

If found, return the Node and ith_child to identify uniquely

2.5 Function Insert

Search vertically and horizontally

If a node_size is (2 * degree – 1), split the node, then insert

Else insert directly

2.6 Function crawl_all

Crawl and analyze the first page and get relative links(1) on this page

Crawl relative links(1) and analyze each page, get relative links(2) on new pages

Crawl relative links(2)

2.7 Function Analyze Page

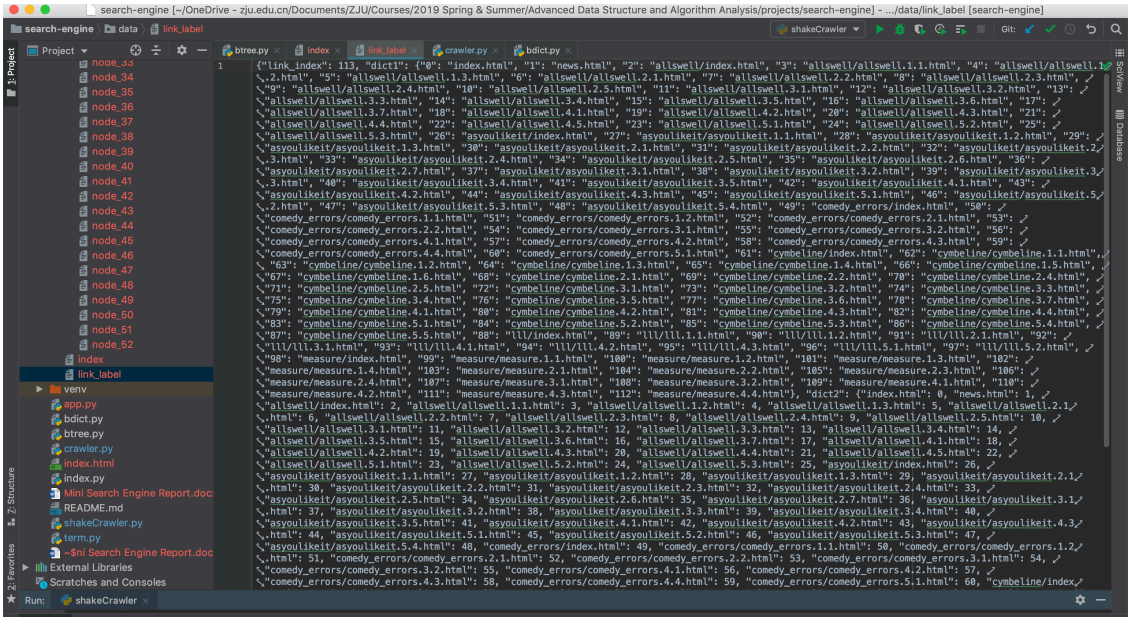
Get the contents of <a> and <title> tag of each html document

Split the content to tokens

Filter out stop words, punctuations, get the stemmer of the word

If the key does not exist in index, insert it and update the word counter

Otherwise, update the word counter directly



As we can see from the illustration above, with a B-Tree of degree 300, a B Tree of 52 nodes is finally generated.

Chapter 3: Testing Results

3.1 Time

Degree / Result	30	100	200	300	400
Operas Crawled (indexed) (44 in all)	16	13	22	34	28
Time cost (min)	20	10	10	10	10
# nodes on disk (LRU nodes excluded)	412	80	44	38	0 (LRU not used in 1st 10 mins)

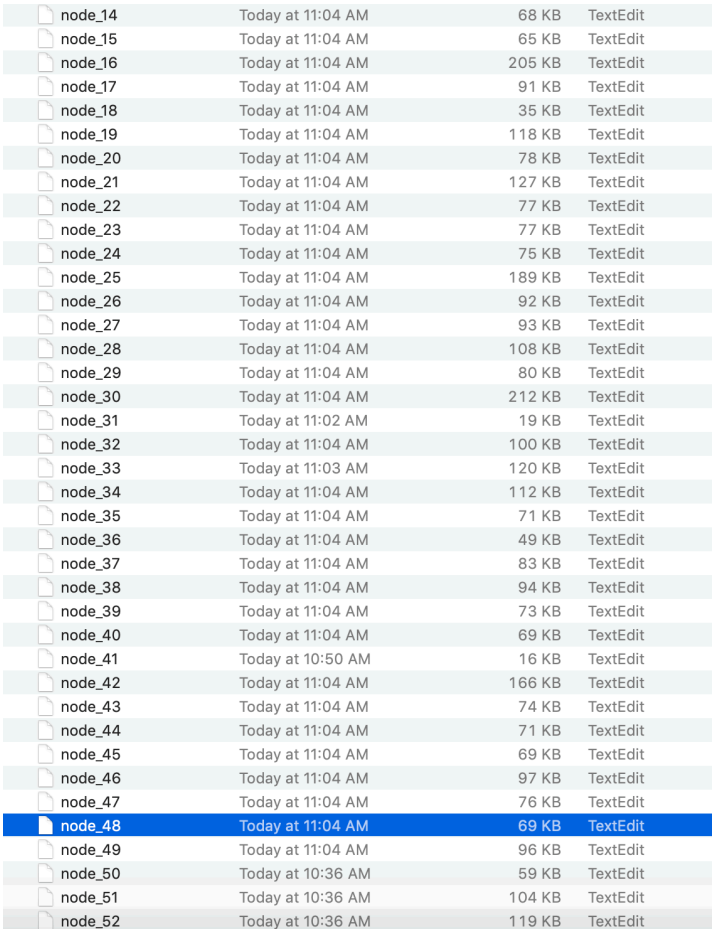
3.2 Storage

Since we used Python for the programming language, for convenience, JSON is applied to our on-disk files. And when the degree of the B+ tree is 300, we can expect an average node size to be of 200KB or so. To handle the input and output properly and a good correspondence with buffer and disk files, we implemented a standard, i.e., several **__jsonfy** and **__unjsonfy** functions. Following is an example:

```python
def display(self):
    print('is_leaf: ', self.is_leaf)
    print('keys: ', self.keys[0:self.n_keys])
    print('child: ', self.child)
    print("n_keys: ", self.n_keys)
    print("len(keys): ", len(self.keys))
    print("file_index:", self.file_index)
    print("child_index", self.child_index)
    print('\n')

# convert the node to json style to write on the disk
def __jsonfy(self):
    d = dict()
    d['file_index'] = self.file_index
    d['is_leaf'] = self.is_leaf
    d['n_keys'] = self.n_keys
    # should be maintained, or read from the children, type(child)
    # d['keys'] = self.keys[0:self.n_keys]
    d['keys'] = list(map(lambda key: key.jsonfy(), self.keys[0:self.n_keys]))
    d['child_index'] = self.child_index
    # print('d: ', d)
    return d
```

We do know that there are redundancies in our on-disk files because we write the field name of the nodes many times but for a problem in the size of the website http://shakespeare.mit.edu/ , it handles very well and successfully reduced the team workload.



Chapter 4: Analysis and Comments

4.1 the time and space complexities

	Insertion	Query
B Tree Index	**O(h) + O(degree)**	**O(h) + O(degree)**
	Space: Proportional to the size of the webpage files	
Crawler	**Proportional to the number of new pages to crawl**	**NULL**

4.2 Comparison between data structures

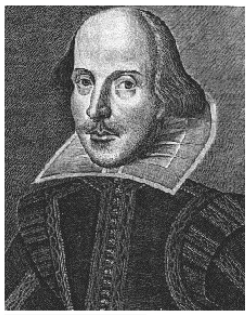
If it is an equivalence query, then the hash index obviously has an absolute advantage, because only one algorithm is needed to find the corresponding key value; of course, the premise is that the key value is unique. If the key value is not unique, you need to find the location of the key first, and then scan backward according to the linked list until you find the corresponding data.

The keyword retrieval efficiency of the B tree index is relatively average, and the fluctuation range is not as large as that of the B-tree. In the case of a large number of duplicate key values, the efficiency of the hash index is extremely low because of the so-called hash collision problem.

4.3 Further possible improvements.

We use Jinja and Flask Framework for our front-end search engine. But due to limited time, we only provide relative links but yet no quotes of the page. It's a bit ugly so I think we can try to show a quote of the destination link rather than just the plain link.

Reconstructing the index takes really long time, so for peer review convenience, I will deploy a cloud server. If it is deployed by the time you review our project, you can visit http://shakespeare.fenghe.info to view it.



What do you want? Search

- http://shakespeare.mit.edu/lll/lll.4.3.html
- http://shakespeare.mit.edu/midsummer/midsummer.3.2.html
- http://shakespeare.mit.edu/henryv/henryv.5.2.html
- http://shakespeare.mit.edu/two_gentlemen/two_gentlemen.2.4.html
- http://shakespeare.mit.edu/lll/lll.5.2.html
- http://shakespeare.mit.edu/midsummer/midsummer.1.1.html
- http://shakespeare.mit.edu/lll/lll.1.2.html
- http://shakespeare.mit.edu/romeo_juliet/romeo_juliet.2.2.html
- http://shakespeare.mit.edu/allswell/allswell.1.3.html
- http://shakespeare.mit.edu/much_ado/much_ado.2.3.html
- http://shakespeare.mit.edu/lll/lll.3.1.html
- http://shakespeare.mit.edu/romeo_juliet/romeo_juliet.1.1.html
- http://shakespeare.mit.edu/asyoulikeit/asyoulikeit.3.2.html
- http://shakespeare.mit.edu/asyoulikeit/asyoulikeit.3.5.html
- http://shakespeare.mit.edu/troilus_cressida/troilus_cressida.3.1.html
- http://shakespeare.mit.edu/twelfth_night/twelfth_night.2.4.html
- http://shakespeare.mit.edu/two_gentlemen/two_gentlemen.5.4.html
- http://shakespeare.mit.edu/asyoulikeit/asyoulikeit.5.2.html
- http://shakespeare.mit.edu/much_ado/much_ado.4.1.html
- http://shakespeare.mit.edu/two_gentlemen/two_gentlemen.1.1.html
- http://shakespeare.mit.edu/two_gentlemen/two_gentlemen.4.4.html
- http://shakespeare.mit.edu/winters_tale/winters_tale.4.4.html
- http://shakespeare.mit.edu/merchant/merchant.3.2.html
- http://shakespeare.mit.edu/midsummer/midsummer.2.1.html
- http://shakespeare.mit.edu/richardiii/richardiii.4.4.html
- http://shakespeare.mit.edu/midsummer/midsummer.2.2.html
- http://shakespeare.mit.edu/much_ado/much_ado.1.1.html
- http://shakespeare.mit.edu/two_gentlemen/two_gentlemen.1.2.html
- http://shakespeare.mit.edu/asyoulikeit/asyoulikeit.4.3.html

Appendix: Source Code (if required)

At least 30% of the lines must be commented. Otherwise the code will NOT be evaluated.

Author List

Specify ***who did what*** to show that particular contributors deserve to have their names printed in the cover page of your report.

Declaration

We hereby declare that all the work done in this project titled "Mini Search Engine" is of our independent effort as a group.