

Mini Search Engine Report

Test Environment

Hardware

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

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, which means it takes a long time to index the pages.

```
[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',
```

Index – B Tree with LRU Buffer

During the first test, I set the degree of B Tree to 30, which turned out to be a bad decision. When the degree of the B+ tree (comparatively better) is set to 300, it takes about 1.5 hours to index the Shakespeare website.

Implementation

1. 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. 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

3. Function Insert:

Search vertically and horizontally

If a node_size is $(2 * \text{degree} - 1)$, split the node, then insert

Else insert directly

Algorithm Analysis

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 1 st 10 mins)

Storage

Since we used Python for the programming language, for convenience, we applied JSON to our on-disk files. And when the degree of the B+ tree is 300, we can expect high an average node to be of 200KB.

There are redundancies as 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.

node_14	Today at 11:04 AM	68 KB	TextEdit
node_15	Today at 11:04 AM	65 KB	TextEdit
node_16	Today at 11:04 AM	205 KB	TextEdit
node_17	Today at 11:04 AM	91 KB	TextEdit
node_18	Today at 11:04 AM	35 KB	TextEdit
node_19	Today at 11:04 AM	118 KB	TextEdit
node_20	Today at 11:04 AM	78 KB	TextEdit
node_21	Today at 11:04 AM	127 KB	TextEdit
node_22	Today at 11:04 AM	77 KB	TextEdit
node_23	Today at 11:04 AM	77 KB	TextEdit
node_24	Today at 11:04 AM	75 KB	TextEdit
node_25	Today at 11:04 AM	189 KB	TextEdit
node_26	Today at 11:04 AM	92 KB	TextEdit
node_27	Today at 11:04 AM	93 KB	TextEdit
node_28	Today at 11:04 AM	108 KB	TextEdit
node_29	Today at 11:04 AM	80 KB	TextEdit
node_30	Today at 11:04 AM	212 KB	TextEdit
node_31	Today at 11:02 AM	19 KB	TextEdit
node_32	Today at 11:04 AM	100 KB	TextEdit
node_33	Today at 11:03 AM	120 KB	TextEdit
node_34	Today at 11:04 AM	112 KB	TextEdit
node_35	Today at 11:04 AM	71 KB	TextEdit
node_36	Today at 11:04 AM	49 KB	TextEdit
node_37	Today at 11:04 AM	83 KB	TextEdit
node_38	Today at 11:04 AM	94 KB	TextEdit
node_39	Today at 11:04 AM	73 KB	TextEdit
node_40	Today at 11:04 AM	69 KB	TextEdit
node_41	Today at 10:50 AM	16 KB	TextEdit
node_42	Today at 11:04 AM	166 KB	TextEdit
node_43	Today at 11:04 AM	74 KB	TextEdit
node_44	Today at 11:04 AM	71 KB	TextEdit
node_45	Today at 11:04 AM	69 KB	TextEdit
node_46	Today at 11:04 AM	97 KB	TextEdit
node_47	Today at 11:04 AM	76 KB	TextEdit
node_48	Today at 11:04 AM	69 KB	TextEdit
node_49	Today at 11:04 AM	96 KB	TextEdit
node_50	Today at 10:36 AM	59 KB	TextEdit
node_51	Today at 10:36 AM	104 KB	TextEdit
node_52	Today at 10:36 AM	119 KB	TextEdit

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.

Pseudo-code:

1. Function crawl_all:

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

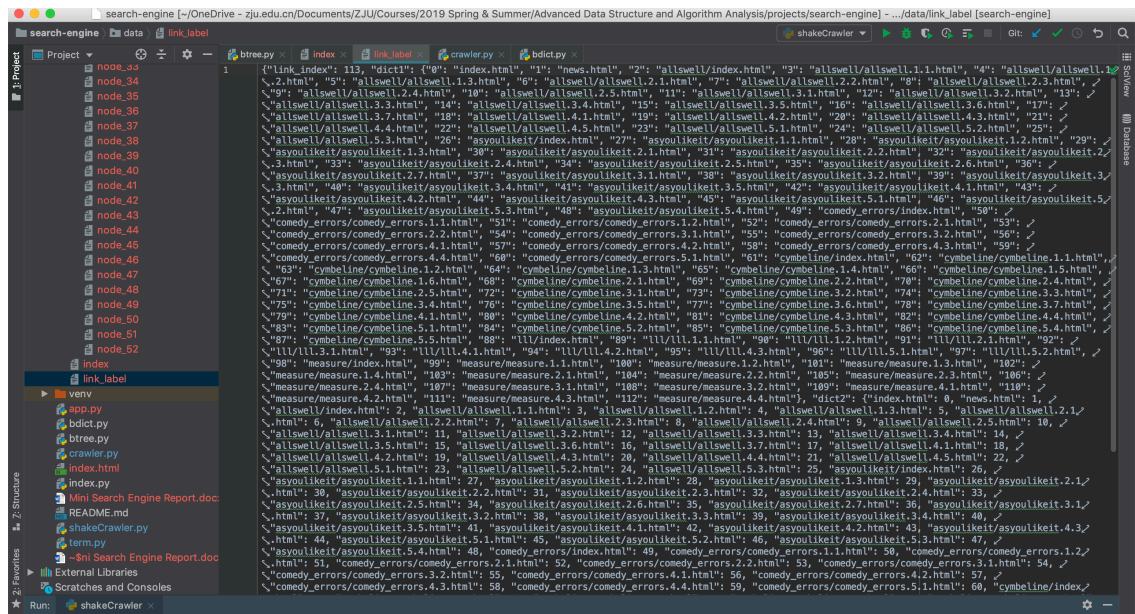
Crawl relative links(1) and analyze each page, get relative links(2) on new pages

Crawl relative links(2)

2. 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.

User Interface

We use command line currently.


We planned use Jinja and Flask Framework for our front-end search engine.




But due to limited time, it is not fully implemented yet. It will be a finished work in our final submission.

Peer Review Instructions

To generate an index, delete all the files(mind not to delete the folders) in data directory. Then run **python shakecrawler** in command line. And keep it in mind that you have to have all the dependencies installed.

Dependencies:

Project: search-engine > **Project Interpreter**  For current project

Project Interpreter:  Python 3.7 (search-engine) (2) ~/OneDrive - zju.edu.cn/Documents/ZJU/Courses/2019 Spring & Su  

Package	Version	Latest version
Click	7.0	7.0
Flask	1.0.2	1.0.2
Flask-WTF	0.14.2	0.14.2
Jinja2	2.10	2.10
MarkupSafe	1.1.1	1.1.1
WTForms	2.2.1	2.2.1
Werkzeug	0.15.0	0.15.0
beautifulsoup4	4.7.1	4.7.1
itsdangerous	1.1.0	1.1.0
nltk	3.4	3.4
pip	19.0.3	19.0.3
setuptools	40.8.0	40.8.0
singledispatch	3.4.0.3	3.4.0.3
six	1.12.0	1.12.0
soupsieve	1.8	1.8

Group Members