# SEARCH ENGINE GUIDE

**NAME: VENKATA SAI VARUN UPPALA**

**COSC 488**

**PROJECT:** Search Engine (Part 1)

**STATUS:** Completed

**THINGS I WISH I KNEW  BEFOREHAND:** I was a little confused about the indexes, in the beginning, I wasn't sure about an optimal approach to the problem. Would I have reached the TA beforehand, things would have been a lot easier.

## ENGINE DESIGN

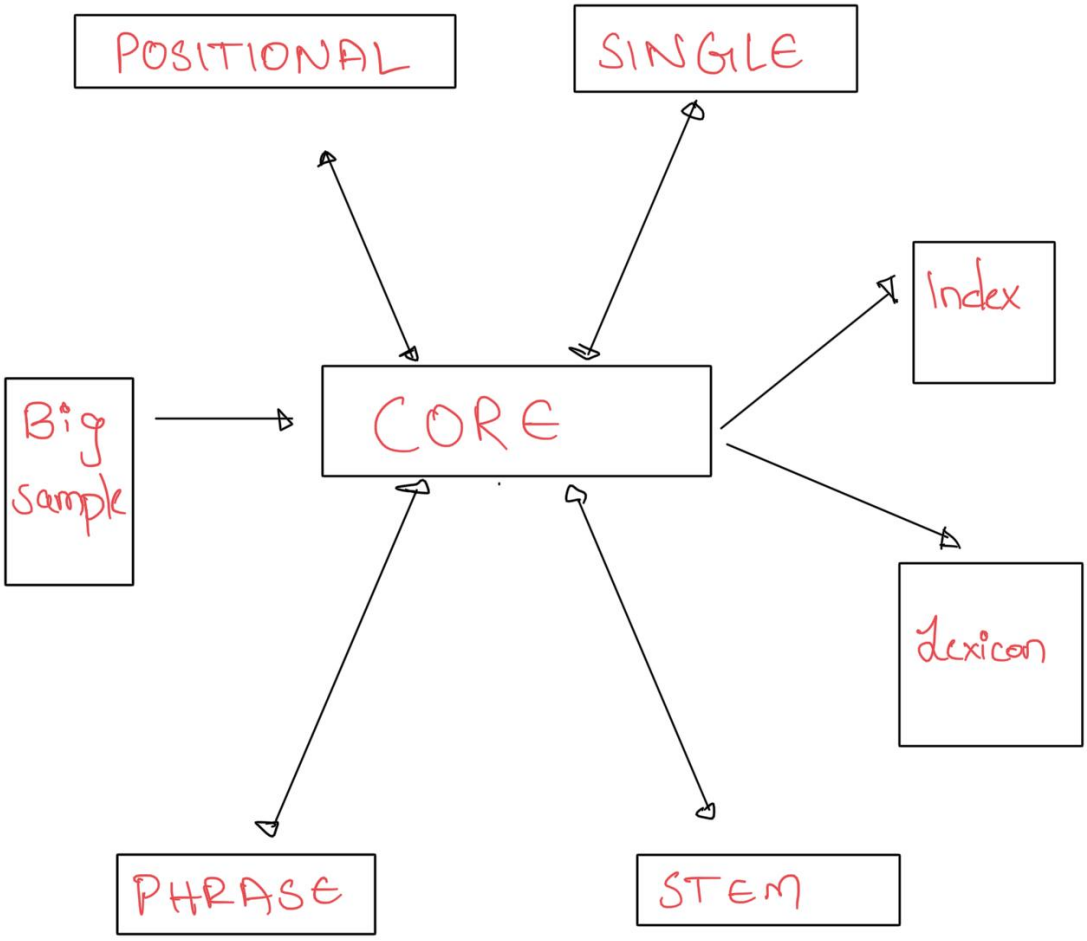


**Figure 1: Engine Structure**

The above image depicts the structure of the engine. The engine is run through the file core.py, A Bigsample is uploaded to the engine and the required type of indexed is queried through command-line argument. A memory constraint has to be specified. The output folder has to be specified.

**Language**: Python
**Libraries** :

- RE
- JSON
- TIME
- OS
- STATISTICS
- STRING
- NLTK (For Porter Stemmer)
- WARNINGS (To ignore few warnings)

**Note**: Please make sure all the packages are installed before the program is run.

**Input Syntax**:

Python core.py [-Input directory] [-index type] [-memory constraint] [-Output directory]

Index options : single,pos,stem,phrase
Memory constraints : **1000 / 10,000 / 100,000 / 0**

**Note**: Please mention memory as '0' for unlimited memory.
The output directory cannot be changed, it has to be 'output' only.

**Examples**:

- Python core.py BigSample stem 1000 output
- Python core.py BigSample phrase 0 output
- Python core.py BigSample single 10000 output
- Python core.py BigSample pos 100000 output

## The flow of the Program

The initial flow of the program is the same for all the indexes, which then changes for each during the tokenization.

- Firstly, after the program is run, the program checks whether the output directory exists. If there are files already in the directory, they are deleted.
- Then, the index queried is identified and the following function is called.
- After reaching out for the respective program, firstly the directory is opened and each file is read one by one.
- For each file, a generator is created and it is read line by line.
- Each document is identified by the opening tag <DOC> and closing tag </DOC> and stored into a string and a document number is assigned to it by the counter. Later, the text is identified by the regular expression by using <TEXT> and </TEXT>.

- During this process, HTML entities and comment lines are removed with the help of regular expressions.
- This is further passed for tokenization which varies for each index.

**Single term Index**

- After the string has been created for the following document, then it is passed to a function to capture the special cases and store them as in the required format.
- Once these have been performed they are passed for removing punctuations, stopwords and then tokenized and stored in a dictionary.

**Single term positional Index**

- After the string has been created, then it is passed to a function to remove the punctuations.
- Later, for each term, their frequency,docid and respective positions are stored.

**Stem Index**

- After the string has been created, then it is passed to a function to remove the punctuations.
- Then, Porter stemmer is used to identify the stem word for each of them.
- Later, each term is stored in a dictionary.

**Phrase Index**

- After the string has been created, then it is passed to a function to remove the punctuations.
- Then two-term phrases are identified and stored in a dictionary.

Later all the indexes follow the procedure below.

- All the output files are later merged into a single file.
- Then the analysis is provided.

## Results and Analysis

The outputs for each index are as follows.

**Single Term index**

1. Lexicon {term:term frequency}
2. Final {term:{docid:freq}}

| **Single Term Stats** | **Lexicon(# of terms)** | **Index size Lexicon+PL (bytes)** | **Max df** | **Min df** | **Mean df** | **Median df** |
|---|---|---|---|---|---|---|
| | **32486** | **4645476** | **5529** | **8** | **203.76** | **194** |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |

| | 1000 | 10,000 | 100,000 | UNLIMITED |
|---|---|---|---|---|
| LOADING FILES | 14528 | 14046 | 12687 | 13058 |
| MERGING | 54339 | 2408 | 552 | 555 |
| TOTAL | 68868 | 16455 | 13239 | 13613 |

**Single Term Positional index**

3. Lexicon {term:term frequency}
4. Final {term:[frequency,{ docid :[ positions ]}}

| **Positional Stats** | **Lexicon(# of terms)** | **Index size Lexicon+PL (bytes)** | **Max df** | **Min df** | **Mean df** | **Median df** |
|---|---|---|---|---|---|---|
| | **29305** | **17033850** | **5556** | **11** | **267** | **257** |

| | 1000 | 10,000 | 100,000 | UNLIMITED |
|---|---|---|---|---|
| LOADING FILES | 5748 | 4755 | 4478 | 4506 |
| MERGING | 339267 | 12326 | 2471 | 2469 |
| TOTAL | 345016 | 17082 | 6950 | 6975 |

**Stem index**

5. Lexicon {Stem term:term frequency}

6. Final {Stem term:{docid:freq}}

| **Stem Term Stats** | **Lexicon(# of terms)** | **Index size Lexicon+PL (bytes)** | **Max df** | **Min df** | **Mean df** | **Median df** |
|---|---|---|---|---|---|---|
| | **22784** | **5040425** | **3846** | **11** | **243** | **238** |

| | 1000 | 10,000 | 100,000 | UNLIMITED |
|---|---|---|---|---|
| LOADING FILES | 18452 | 17914 | 18093 | 17933 |
| MERGING | 55899 | 2221 | 596 | 620 |
| TOTAL | 74352 | 20135 | 18689 | 18544 |

**Phrase index**

7. Lexicon {Phrase: phrase frequency}
8. Final {Phrase:{docid:freq}}

| **Phrase Stats** | **Lexicon(# of terms)** | **Index size Lexicon+PL (bytes)** | **Max df** | **Min df** | **Mean df** | **Median df** |
|---|---|---|---|---|---|---|
| | **293966** | **21821946** | **36528** | **21** | **587** | **545** |

| | 1000 | 10,000 | 100,000 | UNLIMITED |
|---|---|---|---|---|
| LOADING FILES | 5361 | 4486 | 4069 | 3964 |
| MERGING | more than 3 mins | 59720 | 6763 | 2303 |
| TOTAL | more than 3 mins | 64206 | 10832 | 6267 |

**DRAWBACKS:** Time taken to merge phrase index when memory is 1000 triples is high. (10 mins approx)

**Conclusion:** The system is efficiently generating indexes. The frequency of each term, the document in which it appears, and the number of times it appears in the subsequent document are all saved during the indexing process. The primary goal of storing in this manner is to improve the retrieval process' efficiency. Since then, two-term phrases have been utilized, with a lower count than three-term phrases, which would increase the likelihood of capturing the correct information.