**CS 6322.501**

**Information Retrieval**

**By – Navaneeth Venugopala Rao**

**NET ID – nbv140130**

**Program Description:**

# **Algorithm and Design Decisions**

As discussed in the problem statement (Harabagiu, Homework1, 2016), the following algorithm is proposed for the solution:-

1. The main function validates the command-line arguments using a third party library, Apache CLI (Apache, 2015) and then passes the path to the Cranfield (Cranfield, 2016) data and stopwords file (Harabagiu, stopwords, 2016) to the Parser class.
2. The parse() function of the Parser class is then invoked with this path and then following actions are performed:-
   1. Each entry in that path is traversed.
   2. If the entry is a directory, the parse () function is called recursively on the entries of that directory.
   3. If the entry is a file, step 3 is executed.
3. The parse() function calls the readFile() function which performs the following actions:-
   1. Creates an object of the StorageManager class which is responsible for storing the dictionary and inverted lists.
   2. Passes this object to the tokenize() method of the Tokenizer class which reads each text one line at a time.
   3. A line of text is read and the transformText() function is called which handles and transforms this text as:-
      1. All the SGML tags are replaced by a space.
      2. All the digits in the text are removed.
      3. All the special characters are removed.
      4. If the word is of abbreviated form (ex: U.S.A), the dots are removed (USA)
      5. If the word is of possession form (ex: University’s), the possessives is removed (University).
      6. If the word contains only ‘ such as in the case of their’middle’class then the ‘ is replaced by space and taken as different words.
      7. All the hyphens are replaced by space and the word is treated as two separate words.
      8. All the multiple consecutive spaces are replaced by a single space.
      9. The text is converted to lower case.
   4. After the text is transformed, the line is split into words by using space as delimiter.
   5. Each word is lemmatized using the Stanford Lemmatizer (Github, 2016)
   6. Each word is stemmed using the Porter Stemmer (UCSD, 2010)
   7. Both of these words are passed to the StorageManager class where dictionaries and posting files for both lemma and stem tokens are created
      1. Maps the frequency of stems and lemmas
      2. Stores the max\_tf and doclen for each document
4. The StorageManager object is passed to the Dictionary class where the uncompressed index is created as:-
   1. The stems are stored in a separate dictionary object with document frequency and a reference to the Properties class which represents the structure of the posting file
   2. The same is repeated for the lemmas
5. Step 2 to 4 are executed until all the files are processed. After step 4, the object of Parser class has the uncompessed dictionaries in the memory.
6. This object of Parser class is used to call displayIndexResults() function, which performs the following actions:-
   1. Writes the uncompressed indexes to files
   2. Time taken to create those files
   3. Reports the sizes of those files and number of inverted indexes
7. After step is executed, we have two files for each uncompressed index. Then the Compressor class is called for Index 1 to apply block compression for dictionary and gamma encoding for postings file. Similarly, the same class is called for Index 2 to apply front coding compression and delta encoding for postings file. After the indexes are compressed, 4 files are generated – 2 for each index (1 compressed file for dictionary and another for postings)
   1. The Index 1 is compressed as follows: -
      1. For the dictionary, each word is kept in a buffer and a count is incremented. When the count is k=8, the buffer is flushed out to a file appended with its length and the count is set to 0.
      2. If count is not 8, buffers is appended by the current dictionary term
      3. For the postings file, gamma code is calculated for the each of the 4 integers – tf, df, max\_tf and doclen and it is converted into bytes and written into a separate file.
   2. The Index 2 is compressed as follows: -
      1. For the dictionary, all the words are sorted and then minimum length for the words is obtained. This minimum length is the length of the prefix and then each term is trimmed by this prefix and appended to the front code.
      2. This front code is then written to a file.
      3. For the postings file, delta code is calculated for the each of the 4 integers – tf, df, max\_tf and doclen and it is converted into bytes and written into a separate file.
8. After the above step we have 4 compressed files, and the program reports the size of each by adding up sizes of dictionary and postings compressed files for each index and also the time taken to compress those indexes
9. Then the method displayTermCharacteristics() is called to display the df, tf, and inverted list length (in bytes) for the terms: "Reynolds", "NASA", "Prandtl", "flow", "pressure", "boundary", "shock"
   1. For each term, the inverted list size is calculated as: - Sum of (Integer size for each of docID, max\_tf, df, doclen)
10. Then results for “NASA” displayed along with peak and lowest terms for each index
11. Lastly, the documents with largest max\_tf and doclen are displayed.

# 

# **Program Overview**

## **Running time**

The program took 27500 milliseconds to acquire text characteristics during the test run.

## **Handling special cases in tokenization**

1. Upper and lower case words - All the text is converted to lower case words, so that words “People”, “people” and “pEople” are same token (“people”).
2. Words with dashes - Words having dash in the middle (“middle-class”) are separated and taken as two words. So, “middle” and “class” are taken as separate tokens.
3. Possessives: Possessive words (“chandler’s”) are transformed to non-possessive words by truncating “‘s” at the end (“chandler”).
4. Acronyms: Acronyms (“U.S.A”) are transformed by removing dots (“USA”)

## **Major Algorithms and Data Structures**

The algorithms used are described in the section Algorithms. The data structures used are described as follows:-

1. HashMap to store tokens and stems, where the key is of type String and value is of type Properties class
   1. Insertion takes linear time if collisions are assumed to not occur
   2. Retrieval takes constant time
   3. To compare and retrieve the peak and lowest tokens, a linear search is done on the keys of a map based on their values. This takes O(n) time.
2. The value Properties contains the following:-
   1. Document frequency
   2. HashMap with key as documentID of postings file and value as max\_tf and doclen of that document
   3. HashMap with key as documentID of postings file and value as term frequency in that document

# **References**

# 

Apache. (2015, June 17). *Downloads*. Retrieved from Apache Commons: https://commons.apache.org/proper/commons-cli/download\_cli.cgi

Apache. (2015, May 18). *Lang*. Retrieved from Commons: http://commons.apache.org/proper/commons-lang/download\_lang.cgi

Cranfield. (2016, January 27). *Cranfield collection*. Retrieved from UTD csgrads1: /people/cs/s/sanda/cs6322/Cranfield

Github. (2016). *Stanford NLP 3.4.1.* Retrieved from Github: http://stanfordnlp.github.io/CoreNLP/

Harabagiu, S. (2016, Jan 27). *Homework1.* Retrieved from CS 6322: http://www.hlt.utdallas.edu/~sanda/courses/IR/Hm1\_s2016.pdf

Harabagiu, S. (2016, Feb 25). *stopwords.* Retrieved from csgrads1: /people/cs/s/sanda/cs6322/Cranfield/resourcesIR

UCSD. (2010). *Stemmer.java.* Retrieved from SVN: http://chianti.ucsd.edu/svn/csplugins/trunk/soc/layla/WordCloudPlugin/trunk/WordCloud/src/cytoscape/csplugins/wordcloud/Stemmer.java

NOTE: The output file will increase the number of pages to 130. Hence it is not included in the Program Description. It is included separately as Output.txt File.