A SIMPLE SEARCH SYSTEM FOR PHRASE AND PROXIMITY QUERIES

Karahan Sarıtaş - 2018400174
Department of Computer Engineering, Boğaziçi University
karahan.saritas@boun.edu.tr

ABSTRACT

In this assignment, we are expected to implement a document retrieval system for phrase and proximity queries using the *positional inverted indexing scheme*. We will use the Reuters-21578 data set for this purpose. Reuters-21578 contains 21578 news stories from Reuters newswire. There are 22 SGML files, each containing 1000 news articles, except the last file, which contains 578 articles. We are expected to preprocess the data set, build the inverted index and implement a query processor.

1 Data Preprocessing

Firstly, we start our data preprocessing procedure by parsing the news from the Reuters dataset. We extract **<TITLE>** and **<BODY>** tags of each news. *NEWID* fields are used as document IDs. A set of regex patterns are used to extract relevant information. Documents are stored in a dictionary where *key* corresponds to document IDs and *value* corresponds to smaller dictionaries composed of a title and body for each text. After collecting all the textual information, we feed these documents into the tokenization pipeline. Our pipeline roughly consists of five steps: HTML unescaping, punctuation removal, splitting, case-folding and stemming.

In HTML, escaping involves substituting certain special characters with alternative ones, typically <, >, ",$, and &. These characters hold distinct purposes within HTML documents. The fundamental idea of escaping is to encrypt the characters < and > and apostrophes to make them less identifiable as tags. Escaping HTML has a variety of uses, the most obvious of which is that it can be inserted into an HTML document without rendering to show code. It can be observed that our dataset contains examples of HTML escaping. We have to eliminate escaping characters before proceeding with tokenization. For this purpose, the built-in html (HyperText Markup Language support) library can be used in Python.

After removing the escape characters, we can remove the punctuation and digits. To remove the punctuation, we simply used the list in string.punctuation provided within the standard library. Upon removal of punctuation and digits, we split the text into tokens by whitespaces. This operation changes our data structure from *string* to *list*. Now we have a list of tokens in our hands.

Now, we apply *case-folding* to each token in our list, by reducing all letters to lowercase. Case-folding helps us to match instances of different words at the beginning of a sentence with the query of those words. On the other hand, such case folding can equate words that might better be kept apart (such as *Fed* and *fed*) [1, p. 30]. Case-folding doesn't reduce the number of tokens we have - but it is very likely to reduce the number of *unique* tokens in the corpus.

Lastly, we apply *stemming* to each token in the list. We used a slightly modified version of the Porter Stemming algorithm, which has proved to be one of the most effective stemming algorithms empirically, provided here [2].The goal of both stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form [1, p. 32]. After the stemming procedure, the number of unique tokens decreases since some of the different tokens will be reduced to the same root.

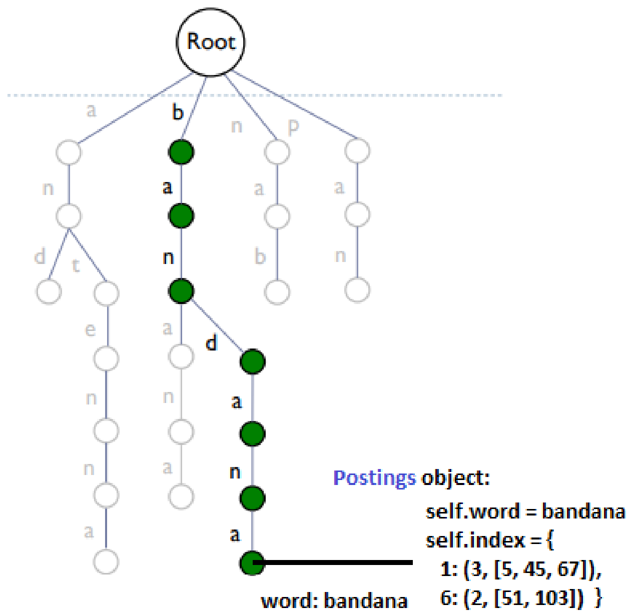
- *How many tokens does the corpus contain before and after case-folding?*: We don't expect the number of tokens to change upon case-folding. We had 2,644,987 tokens both before and after case-folding.
- *How many terms (unique tokens) are there before and after case-folding?*: We expected the number of tokens to change upon case-folding. Before the case folding we had 58,617 unique tokens, whereas after the case folding we had 42,574 unique tokens in the corpus. After case-folding, we apply stemming on the words. As a result of the stemming operation, we get 31,760 unique tokens in our corpus.
- List the top 100 most frequent terms after case-folding: Below, you can find the top 100 most frequent words both after case-folding (before stemming) and after stemming side by side. As you can see, most frequent words are usually stopwords in English:

```
After case-folding (before stemming)                    After stemming
        the: 144423                         |            the: 144423
        of: 73605                           |            of: 73615
        to: 73073                           |            to: 73074
        in: 54939                           |            in: 54942
        and: 54545                          |            and: 54549
        said: 53096                         |            said: 53096
        a: 52169                            |            a: 52169
        s: 32871                            |            it: 37881
        for: 27306                          |            s: 32871
        mln: 26732                          |            for: 27306
        it: 22882                           |            mln: 26745
        dlrs: 21273                         |            on: 25309
        on: 19392                           |            dlr: 24681
        reuter: 18966                       |            reuter: 20036
        pct: 18046                          |            pct: 18046
        is: 16875                           |            is: 16875
        that: 15527                         |            be: 15923
        from: 15277                         |            that: 15527
        by: 15154                           |            year: 15386
        its: 14995                          |            from: 15277
        will: 14855                         |            by: 15154
        vs: 14836                           |            will: 15042
        be: 14738                           |            vs: 14836
        at: 14517                           |            at: 14525
        with: 13706                         |            with: 13706
        year: 13109                         |            bank: 12332
        was: 11938                          |            wa: 11938
        u: 11326                            |            compani: 11520
        billion: 10726                      |            u: 11326
        he: 10676                           |            billion: 10756
        has: 10185                          |            he: 10676
        company: 9699                       |            ha: 10224
        as: 9694                            |            share: 10220
        an: 9590                            |            as: 9695
        cts: 9219                           |            an: 9594
        would: 9200                         |            ct: 9457
        not: 8308                           |            would: 9201
        inc: 8161                           |            market: 8485
        bank: 8018                          |            not: 8309
        net: 7698                           |            inc: 8161
        which: 7556                         |            new: 8159
        new: 7182                           |            net: 7712
        corp: 7171                          |            which: 7556
        but: 7141                           |            trade: 7498
        are: 7043                           |            price: 7469
        this: 6855                          |            corp: 7192
        have: 6758                          |            but: 7142
        were: 6262                          |            ar: 7052
        market: 5959                        |            have: 6989
        last: 5915                          |            thi: 6856
        one: 5873                           |            stock: 6713
        stock: 5705                         |            loss: 6398
```

```
        had: 5685                    |             were: 6262
        loss: 5626                   |             rate: 5961
        or: 5510                     |             last: 5955
        shares: 5238                 |             or: 5720
        also: 5174                   |             had: 5692
        up: 5160                     |             sale: 5680
        about: 5118                  |             offer: 5431
        they: 5098                   |             shr: 5391
        two: 5073                    |             up: 5249
        share: 4865                  |             also: 5174
        trade: 4746                  |             about: 5118
        co: 4736                     |             thei: 5099
        been: 4501                   |             two: 5073
        shr: 4293                    |             unit: 4914
        oil: 4272                    |             oper: 4894
        may: 4258                    |             co: 4840
        debt: 4094                   |             product: 4704
        sales: 4070                  |             oil: 4506
        government: 4034             |             expect: 4505
        first: 4014                  |             been: 4501
        more: 3958                   |             profit: 4405
        april: 3804                  |             month: 4369
        after: 3738                  |             issu: 4344
        march: 3640                  |             govern: 4326
        exchange: 3607               |             debt: 4290
        group: 3459                  |             mai: 4274
        over: 3457                   |             industri: 4249
        than: 3442                   |             report: 4216
        dlr: 3408                    |             plan: 4126
        japan: 3381                  |             offici: 4096
        other: 3356                  |             first: 4014
        profit: 3329                 |             increas: 4010
        prices: 3315                 |             exchang: 3988
        three: 3253                  |             more: 3958
        we: 3249                     |             end: 3877
        price: 3249                  |             note: 3821
        banks: 3242                  |             april: 3804
        per: 3151                    |             after: 3738
        no: 3134                     |             group: 3734
        rate: 3099                   |             export: 3718
        international: 3094          |             week: 3675
        their: 3088                  |             march: 3650
        ltd: 3064                    |             other: 3633
        week: 3014                   |             interest: 3544
        interest: 3004               |             secur: 3504
        foreign: 2987                |             over: 3457
        some: 2945                   |             than: 3442
        told: 2913                   |             includ: 3417
```

2 Inverted Index

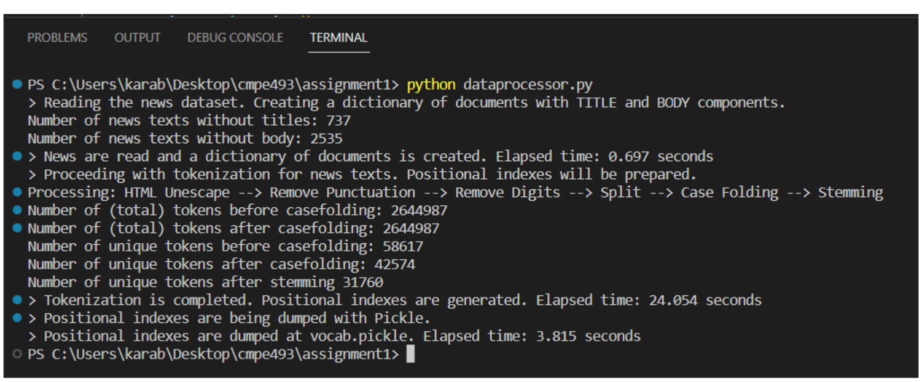
Positional inverted indexing scheme is used to store the words, document frequency, and positions. In the positional inverted index, for each term in the vocabulary, we store postings of the form `<docID: position1, position2, ...>` where each position is a token index in the document. Each posting will also usually record the term frequency. For the final version of the vocabulary, I've used Trie data structure. My *Trie* implementation consists of *TrieNode* objects, where each node corresponds to a character. For the posting lists, I implemented another data structure called *Posting*, which stores the word, document frequency and a **dictionary** of postings in the form given above. Only the nodes corresponding to the end of the words (last character), are associated with a *Posting* object. To exemplify, let's say the word "bandana" occurs in two documents with IDs 1 and 6. In the first document, it occurs at positions 5, 45, and 67. In the second document, it occurs at positions 51 and 103. In this scenario, we can visualize trie and inverted index scheme as follows:



We can easily calculate the term frequency by adding the lengths of the position lists and document frequency by getting the number of keys in the dictionary, if necessary.

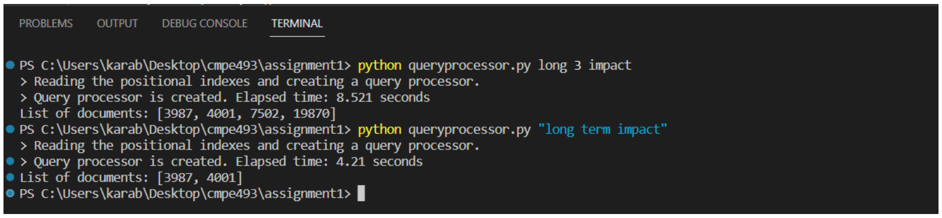
3 Screenshots

- *Provide a screenshot of running the indexing module of your system*

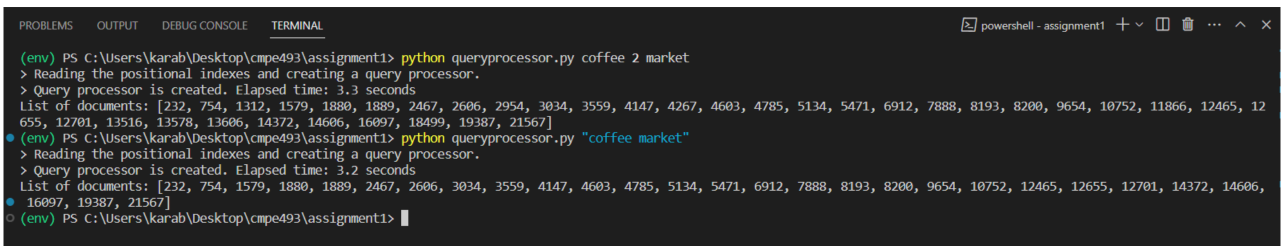


- *Provide two screenshots of running your system for each of the two types of queries.*

  (i) Proximity query: `coffee 2 market`
  Phrase query: `"coffee market"`



  (ii) Proximity query: `long 3 impact`
  Phrase query: `"long term impact"`



References

[1] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[2] Martin Porter. Python implementation of the porter stemming algorithm, 2002.