COMP6714 2023T3 Project: Ranked Retrieval

In this project, you are going to implement (using Python3 in CSE linux machine) a simple search engine that ranks the output documents based on the proximity of the matching terms. A search query in this project is a list of space-separated search terms and each search term may contain any numeric digits or uppercase/lowercase letters, and will not contain any punctuations. You will need to implement an indexer (to index the files) and a search program (to search the index generated by your indexer). As a core requirement for this project, you must implement an inverted index with positional information (for example, the positional index described in you may also implement any additional indexes as appropriate, if you wish.

Given a search query, each matching document from the search result must contains terms that match all the search terms following the below matching rules:

- Search is case insensitive.
- Full stops for abbreviations are ignored. e.g., U.S., US are the same.
- Singular/Plural is ignored. e.g., cat, cats, cat's, cats are all the same.
- Tense is ignored. e.g., breaches, breach, breached, breaching are all the same.
- A sentence can only end with a full stop, a question mark, or an exclamation mark.
- Numeric tokens such as years, integers should be indexed accordingly and searchable.
- Commas in numeric tokens are ignored. e.g., 1,000,000 and 1000000 are the same.
- Except the above, all other punctuation should be treated as token dividers.
- Numbers with decimal places can be ignored from the index, if you wish, as a decimal number is not a valid search term (since '.' is not allowed).

As a requirement of this project, the matching documents in a search result are ranked according to the distances between the matching terms in these documents, such that matching terms closer to each other will be ranked higher than those further apart. Further details are described below in the Ranking section.

You are provided with approximately 1000 small documents (named with their document IDs) available in *~cs6714/reuters/data*. You can find these files by logging into CSE machines and going to folder *~cs6714/reuters/data*. Your submitted project will be tested against a similar collection of up to 1000 documents (i.e., we may replace some of these documents to avoid any hard-coded solutions).

Your submission must include 2 main programs: index.py and search.py as described below. It is your responsibility to submit any other auxiliary Python files if they are needed for the 2 main programs to work properly.

This project will be marked based on auto marking, and will then be checked manually for other requirements described in this specification (for example, if a positional index is implemented). To ensure your project satisfies the input and output formatting requirements, a simple sanity test script is available in *~cs6714/reuters/sanity*. You should run the sanity test script before you submit your solution. To run the sanity test script on a CSE linux machine, simply go inside the folder that contains your index.py and search.py and type: *~cs6714/reuters/sanity* that will run tests based on examples presented below. Note that it is only a sanity test primarily for formatting, and you are expected to test your project more thoroughly.

The Indexer

Your indexer is run by

```
python3 index.py [folder-of-documents] [folder-of-indexes]
```

where [folder-of-documents] is the path to the directory for the collection of documents to be indexed and [folder-of-indexes] is the path to the directory where the index file(s) should be created. All the files in [folder-of-documents] should be opened as read-only, as you may not have the write permission for these files. If [folder-of-indexes] does not exist, create a new directory as specified. You may create multiple index files although too many index files may slow down your performance. The total size of all our generated index files shall not exceed 20MB (which should be plenty for this project).

After the indexing is completed, it will output the total number of documents, the total number of tokens (after any preprocessing and filtering), and the total number of terms to be indexed. The following example illustrates the required formats:

```
$ python3 index.py ~cs67              /MyTestIndex
Total number of document
Total number of tokens:
Total number of terms: 9
$
```

Note: Each line of the output of index.py ends with one newline ('\n') character; and except for the total number of documents, your indexer may have different numbers of tokens and terms depending on your preprocessing choices.

The Search

Your search program is run by

```
python3 search.py [folder-of-indexes]
```

where [folder-of-indexes] is the path to the directory containing the index file(s) that are generated by the indexer. After the above command is executed, it will accept a search query from the standard input and output the result to the standard output as a sequence of document names (the same as their document IDs), one per line and sorted according to their ranking as described in the Ranking section below. It will then continue to accept the search queries from the standard input and output the results to the standard output until the end (i.e., a Ctrl-D). The following example illustrates the required input and output formats:

```
$ python3 search.py ~/Proj/MyTestIndex
Apple
1361
Malaysia
311
908
1356
1675
2956
3051
3438
5169
5195
5216
5258
5285
5382
Australia Technology
3454
271 billions
880
$
```

Ranking

A proximity distance is defined as the number of terms between each pair of terms that match two search terms. The search result is sorted by its minimum sum of the proximity distances between the matching terms (left-to-right by query terms) in an ascending order, then by the number of matching terms occurring in the same order as the search terms, then by the numeric values of the documentIDs (e.g., 72 will be output before 125). To elaborate this further, consider an example of 4 documents with documentIDs 1-4:

```
DocID   Content
1       apple durian cherry bread egg fennel garlic ham
2       bread garlic ham
3       egg bread cherry ... ... ham garlic bread
4       ham garlic bread
```

Given a search query of "garlic bread", ... documents contain these two search terms. The minimum proximity distance between the matching ... terms are calculated as below:

```
Query: garlic bread
Expected output:
DocID   Distance
3       0
4       0
2       0
1       2
```

For example, in Document 3, *bread* appears twice. We pick the second *bread* to calculate the proximity distance as it will result in a minimum value since it is next to *garlic*, i.e., with a distance 0. We define the matching term that is used to calculate the minimum proximity distance as *the closest matching term*. Although Documents 2, 3, 4 are all having the minimum distance of 0, Documents 3 and 4 rank higher than Document 2 since two matching terms "garlic bread" appear in the same order as the search query, while Document 2 has a different order (i.e., "bread garlic"). Furthermore, since Documents 3 and 4 have the same minimum distance and the matching terms are in the same order, they are then sorted by their documentIDs. Finally, Document 1 has the largest minimum distance (2) and hence has the lowest ranking.

Consider another search query "egg ham bread" for the example documents above. Only two documents contain terms matching all three search terms:

```
Query: egg ham bread
Expected output:
DocID   Distance
3       1+1=2
1       2+3=5
```

For Document 3, there is more than one term matching *egg* and *bread* respectively. The second *egg* and the second *bread* are chosen as the closest matching terms to achieve the minimum sum of the proximity distances between the 3 matching terms for the 3 search terms. As a result, Document 3 has a smaller minimum distance than Document 1 and hence it ranks higher.

Using the real documents available in *~cs6714/reuters/data*, the following examples present their expected ranked output from your search.py and you can study their ranking further by inspecting the content of their corresponding documents.

```
$ python3 search.py ~/Proj/MyTestIndex
australia technology
3454
bank expect distribution
3077
4019
4367
875
$
```

Since the input is from the standard input, your search engine should be able to accept input redirected from a file as shown below:

```
$ cat test1.txt
US finance COMPANY investor
$ python3 search.py ~/Proj/MyTestIndex < test1.txt
5171
3023
3396
5778
1682
$
```

Displaying Lines Containing Matching Terms

Given a search query starting with '>', in addition to the displaying the matching document IDs, the lines of text that contain the closest matching terms are also displayed according to the following rules:

- Each matching document ID is displayed with '>' and a space in front, followed by lines of text that containing the closest matching terms.
- For each matching document, only one line of text that contains its corresponding closest matching term is displayed for each search term.
- Lines of text are displayed according to the order of lines in a document, i.e., output line 1 before line 2.
- In case more than one closest matching term is determined for a search term and they are at different lines, only the first line of these lines is displayed. This includes the case when a search query only consists of one search term such that matching terms can be found in multiple lines (refer to the **apple** example below).

The examples below illustrate some of these rules and the display format.

```
$ python3 search.py ~/Proj/MyTestIndex
> bank expect distribution
> 3077
     The bank said it expects the distribution will be made in
> 4019
  It said it did not know when distributions would be made.
     The bank said it expected to report positive earnings in
> 4367
     Closing is expected to take place in early April and the
     The partnership will acquire the refining and distribution
  facility with U.S. and foreign banks to finance inventories and
> 875
  prepared to negotiate a new distribution based on objective
  bank debt, have increased political pressure on the country to
     The expected drop in prices could result in losses of as
> AUStralia Technology
> 3454
  marketing of high-technology smelting processes invented in
  Australia, notably the Siromelt Zinc Fuming Process.
> apple
> 1361
     The department said stocks of fresh apples in cold storage
$
```

Marking

This project is worth **40 points**. Your submission will be tested and marked on CSE linux machines using Python3. Therefore, please make sure you have tested your solution on these machines using **Python3** before you submit. You will not receive any marks if your program does not work on CSE linux machines and only

works in other environment such as your own laptop. Full marks will be awarded to submissions that follow this specification and pass all the test cases.

Although we do not measure the runtime speed, your indexing program will be terminated if it does not end after **one minute**, and you will receive zero marks for the project (since we cannot get the index generated successfully for further testing), and your search program will be terminated if it does not end after **10 seconds** per search query, and you will receive zero marks for that search query.

Partial Marks

For this project, a search result from the search engine is only considered correct if it contains exactly the same set of document names as the expected answer, and their order must be exactly the same as well. We will grant you some partial marks based on F-measure to evaluate how close your result is to the expected answer (in which any correct document but in a wrong order will be treated as wrong documents). *F-measure = 2 * (precision * recall) / (precision + recall)* will be used to calculate the final score for each test when your search results differ from the expected output. The final score for each test will be calculated by: [F-measure round to 2 decimal places] * [fullMarks of that test]. The following examples further illustrate how this scheme works. Given the expected answer Ans, suppose that there are 6 results returned from 6 different search programs.

```
Ans      Search1    Search2    Search3    Search4    Search5    Search6
1        1          3          2          1          1          1
2        2          2          1          2          2          2
3        3          1          3          3          3          3
4                              3          4          4          5
                                          5          4          6
```

Their F-measure based on their precision and recall are calculated as below:

```
Search1 - recall: 0.75 precision: 1.00 fmeasure: 0.86
Search2 - recall: 0.25 precision: 0.33 fmeasure: 0.28
Search3 - recall: 0.50 precision: 0.50 fmeasure: 0.50
Search4 - recall: 1.00 precision: 0.80 fmeasure: 0.89
Search5 - recall: 1.00 precision: 0.80 fmeasure: 0.89
Search6 - recall: 0.75 precision: 0.60 fmeasure: 0.67
```

If this search query is worth 2 pts, Search6 will get 0.67 * 2 = 1.34 pts.

For search queries starting with '>' (the queries with matching lines displayed), no partial marks will be awarded. In order to get full marks, a search result has to exactly match the output of the expected answer (including the document IDs, the matching lines and their order). Therefore, please check your solution with the provided sanity test before submitting the project (and do not leave this till last minute). For your information, search queries starting with '>' will contribute in total less than 10 points (out of 40 points) of this project.

Python3 Libraries

As a requirement of this project, you are not allowed to use any Python libraries other than the Python Standard Library (https://docs.python.org/3.9/library/index.html) and NLTK.

For NLTK, you may assume the entire collection (using "all") has been downloaded before the marking process starts. i.e., you should **delete/comment all nltk.download statements**, if any, in your code before your project submission. Note that during your project development, you may want to download only individual required packages rather than "all", as your CSE account has limited space. Please also set the quiet parameter to True in case you forgot to remove the nltk.download statements and the download output affects the auto-marking. For example,

```
nltk.download('package_name', quiet=True)
```

Submission

Deadline: Tuesday 7th November 12:00pm AEST (noon).

The penalty for late submission of assignments will be 5% (of the worth of the assignment) subtracted from the raw mark per day of being late. ~~earned~~ marks will be lost. **No assignments will be accepted later than 5 days after the d**

Use the give command below ment:

```
give cs6714 proj *.p
```

It is **your responsibility** to u your submission to make sure that you have submitted all the required files for index.py and search.py to run properly.

```
6714 classrun –check proj
```

Plagiarism

The work you submit must be your own work. Group submissions will not be allowed. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assignments in previous years, if applicable) and serious penalties will be applied, particularly in the case of repeat offences. Your submissions will also be checked against any relevant code available on the Internet. In particular:

- Do not copy ideas or code from others.
- Do not use a publicly accessible repository or allow anyone to see your code.

Please refer to the Academic Honesty and Plagiarism section under Other Useful Information in <u>the course outline</u> to help you understand what plagiarism is and how it is dealt with at UNSW.

Finally, reproducing, publishing, posting, distributing or translating this assignment is an infringement of copyright and will be referred to UNSW Student Conduct and Integrity for action.