**General Advice:**

- Get started early; it's more work than you probably think.

- This is the foundation of the following projects, so think about design right away. Consider design patterns, e.g. the strategy pattern. Make it easily configurable.

- If you get stuck implementing some component, come to office hours for help, but in the meantime make progress on another component. The same goes for optimization: get everything working first, then tackle your pain points.

- Don't neglect your design document, analysis of results, etc. Having a working system is critical, but communication and evaluation of your system is important as well.

**Q: Can I use external libraries?**

A: We would prefer you to use the standard libraries included in the default installation of your programming language of choice. If you need to use an external library, please contact us via email first.

**Q: What should I submit?**

A: Please submit the source code of your system, a readme file containing instructions on how to run it, and the report (PDF preferred). Do no submit the document collection or any output file.

**Q: Is there a desired interface?**

A: Your program needs to be run from a shell interface (e.g., terminal, command prompt) and it needs to accept arguments. The arguments are the inputs and the configuration to your code. Please follow the exact following instructions when preparing the submission. Your code needs to run regardless of the underlying platform (if you have trouble achieving this, let us know). Please also include all the resources that your code uses in your submission.

To build the index, please follow this format:

build [trec-files-directory-path] [index-type] [output-dir]

- [trec-files-directory-path] is the directory containing the raw documents (e.g. fr940104.0)

- [index-type] can be one of the following: “*single*”, “*stem*”, “*phrase*”, “*positional*”

- [output-dir] is the directory where your index and lexicon files will be written (please make your program create it if doesn’t exist).

example command for building the index (in python):

python build.py /tmp/Mini-Trec/BigSample/ phrase /tmp/my-indexes/

**Q: What format does the document collection follow?**

A: The format of each file is as follows: each file contains multiple documents. Each document has the following structure:

<DOC>

<DOCNO>[document id]</DOCNO>

<PARENT>[parent id]</PARENT>

<TEXT>

...

</TEXT>

</DOC>

For each document, you should extract the [document id] and preprocess/tokenize whatever is enclosed between the <TEXT> tags. Lines starting with <!-- and ending with -➔ are comment lines; you should ignore those.

**Q: Should I do case folding before or after tokenization?**

A: Because you need case information to process some tokens (e.g., “Ph.D” or “U.S.A.”), you should perform case folding after tokenization

**Q: I found this strange email address: BARNES.Don@EPAMAIL.EPA.GOV@IN**

A: The document collection contains typos. Your choice on how to address them.

**Q: Should I use a database for the index?**

A: No database is a great fit to store the inverted index; we suggest you to store it in text files (e.g., csv format); you will be allowed to load the entire index in memory before processing queries.

**Q: How fast should my code be?**

A: Performance of your system is not a factor in grading the project. However, we expect your code to run in less than 2-3 minutes (you’re processing just 10MB of data!)

**Q: Will I be graded based on the style of my code?**

A: No, but please be a responsible programmer and follow the best conventions of the programming language you chose. Remember that we will look over your code while grading, so if your code is not clear, we might end misinterpreting some parts.

**Q: Do you have a test file for the tokenization step?**

A: No, we do not.

**Q: How do I identify special tokens (e.g., “Ph.D”, “U.S.A”, etc.)? Is there a list a can use?**

A: You should try to extract those programmatically. In other words, try to come up with one or more regular expressions to capture such tokens. For example, it could be “all sequences of letters where uppercase and lowercase characters are mixed”.

**Q: Is there a set of common prefixes my system should be able to identify?**

A: No; you should either come up on your own or use some external resource. If you do the latter, please include a citation in the design document.

**Q: I’m having troubles with regular expressions!**

A: Please come to office hours, contact the TAs, or ask on the forum (make sure not to share code!). If you are having troubles testing your regular expressions, we recommend to you use this website: https://regex101.com/

**Q: How should I process escape sequences such as &blank; &hyph; etc. ?**

A: You can replace those SGML escape sequences as follows (this has to be done before tokenization):

&blank; ➔ &

&hyph; ➔ -

&sect; ➔ §

&times ➔ ×

You can find a full list of escape sequences here: <http://turner.faculty.swau.edu/webstuff/htmlsymbols.html>

**Q: How should I process date ranges (e.g. "January 11-14, 2016")?**

A: The assignment contains no instruction on how to handle date ranges; therefore, you are free to choose how to process them. In any case, make sure to report your choice in the design portion of the project report.

**Q: Should I include special tokens in stem index, phrase index and positional index?**

A: No, you are not required to.

**Q: What is the input and the expected output of the engine?**

A: Your system should take in a directory that contains the TREC documents, and output lexicon/index files.

**Q: If a document has more than a 1,000 unique terms, how can I prevent violating the lowest memory constrain?**

A: You can assume that you can always fit at least one full document into memory.

**Q: How should m-way merging work so that space constrains (i.e., limit in the number of triples that can be held in memory) is not violated?**

A: Sort the partial files by term\_id and doc\_id. m-way merging is conceptually similar to mergesort: you have a set of sorted files, and you want to combine them in a single sorted file. Because all partial files are sorted, you can compare the first line of each and write the smallest to the final index. you then load the second line from the file the just written triple came from, and compare the m triples you have in memory again. You continue until all temporary files have been merged.

**Q: Can I use a named entity recognition (NER) tool to extract phrases?**

A: The choice is us to you; however, keep in mind that most NER tools are trained to only extract three types of ngrams: location, people, and organization names. Thus, your entity coverage might be very low and hurt your performance a lot. In any case, make sure to analyze the outcome of your choice in the analysis section of your report.

**Q: For the positional index do you still use <term\_id, doc\_id, term\_frequency> or should you use something like <term\_id, doc\_id, position> to keep track of the locations for each term?**

A: You need to store the position information in the positional index. So, term\_frequency alone will not suffice.

You need to have a set or list of positions for each term in each document.

**Q: After we have merged all of our partial files of triples (with term, document name, and term frequency), we end up with one I file of these triples (which is sorted) on the disk. Would this be the final form of the index?**

A: It is perfectly fine to save the posting lists as <term\_id, document\_id, term\_frequency> triples.

**Q: Will we have to observe memory limitation during query processing? If so, does that influence how we output the index to disk?**

A: For querying, you can assume to have infinite memory; i.e., you have no memory constrains. Thus, the output for project 1 can be as simple as a pair of text files: one containing your posting lists in triplet format, the other containing the lexicon. The lexicon can either be in represented by <term, term\_id> tuples, or by <term, term\_id, document frequency> triplets.

**Q: For the analysis, we need to report the index size, as lexicon + PL (byte). What does this mean?**

A: Just include the size in bytes of the text files containing your posting lists and lexicon.

**Q: What is the best way to obtain term ids?**

A: Using the hash is fine, but we would suggest to use progressive integers to represent term\_ids. That is, initialize a counter while indexing documents; each time you see a new term, increase the counter by 1 and use the new counter as term id.

**Q: Based on the project requirements hyphenated terms like “black-tie” should be stored as “blacktie”, “black”, and “tie”. Where should all of these new tokens be stored after you process them? Do they need to be placed in a separate list in order to not mess up the positional index later on?**

A: You should store the compounded terms separately only when building single terms index. In other words, "part-of-speech" should be stored as "part", "speech", "partofspeech" only in the single term index ("of" is a stopword). For the other indices, you can decide whether to consider the compound word as a single token or split it in multiple tokens. Make sure to report and motivate your choice in the design document of your report.