HOMEWORK 4:

Concurrent web-scrapping and data aggregation using goroutines

Implement a Golang application that receives from command line 1) a filename containing list of URLs of web pages, 2) a *MaxRoutines* number – the maximal number of goroutines, 3) *MaxIndexingTime* - the maximal allowed time for indexing, 4) base result files name (default "results.json"), 5) *MaxResultFileSize* number, and 6) *MaxResultFiles* number (explained below), and does the following:

PHASE I: Indexing

1. Traverses recursively the connected pages starting from URLs provided following the hyperlinks in these web pages. Separate goroutines should be used to speed up the traversal process. You could use breadth-first-search, or custom heuristics to prioritize the search pages.
2. The number of goroutines working in parallel should not exceed *MaxRoutines* number in any moment during the program execution.
3. Extracts information about the keywords, the names (human, places, etc.) and dates mentioned in each page using a custom extraction and weighting criteria (keywords and names found in headers and sub-headers should be more relevant than the others, as well as the keywords found in the first 5000 characters of the web page body). You should suggest your own criteria.
4. Aggregate the extracted information about the found web pages. Output the page urls and necessary metadata to search in them to a number of JSON files – up to *MaxResultFiles,* with maximal file size *MaxResultFileSize*, named using the provided *base result files name* (default "results.json"), that are cyclically rotated – e.g. *results01.json, results02.json, … results< MaxResultFiles>.json*
5. The maximal search time should not exceed the specified *MaxIndexingTime.*

PHASE II: Search

6. Input a list of search keywords from console and find the most relevant web pages to keywords. Output the sorted list of most relevant top 10 resources with relevance percentage and metadata to console in human readable form. You can repeat this step with multiple keywords.

* When indexing pages you can skip the so called stop words, such as (please extend the list as appropriate):

```
a,    is,    the,   an,   and,  are, as,  at,   be,   but,  by,   for,
if,   in,    into,  it,   no,   not, of,  on,   or,   such, that, their,
then, there, these, they, this, to,  was, will, with
```

** Advanced: You can use WordNet (https://wordnet.princeton.edu/) database to find only the *nouns* in the text and to filter out irrelevant keywords by their grammatical kind.

*** Hint: you can use https://godoc.org/golang.org/x/net/html or https://github.com/PuerkitoBio/goquery for parsing/querying the html documents if necessary.