team members: Ruineng Li, rl3315; Frank Zhang, hz2716.

api key: AIzaSyDm0Wcp0OqqLiWypz0ijCRMjkHZ2mYisJs

engine id: 2d6a0e6f605702952

Files:

- install.sh: please install the packages listed in this .sh file before running the code
- test.py: code file
- readme_handin.md: readme
- transcript_brin.txt
- transcript_cases.txt
- transcript_per_se.txt

Description for Program Running:

- Prepare the enviroment:
  - In project1 folder, give execution right for install.sh by using: sudo chmod +x install.sh
  - It will install all required packages and set up the enviroment.
- Executing test.py:
  - in the project1 folder, it contains the test.py file.
  - Run: python3 test.py AIzaSyDm0Wcp0OqqLiWypz0ijCRMjkHZ2mYisJs 2d6a0e6f605702952 0.9 cases
  - API key: AIzaSyDm0Wcp0OqqLiWypz0ijCRMjkHZ2mYisJs
  - Engine id: 2d6a0e6f605702952

Description for Project Design:

- This project contain the following functions:
  - processUserInput: handle inputs from the command line
  - fetch_relevant_pages: fetch pages that contain targeted information based on user feedbacks
  - compute_word_scores: compute scores for words occurred in relevant pages using keyBERT
  - fetch_new_words: use the query dictionary, which is updated by *compute_word_scores*, to get new words for the next iteration
  - reorder_new_words: reorder new search words generated by *fetch_new_words*
- The workflow of this project can be described as follow:
  - While the current precision is smaller than the target precision, we first retrieve search results using Google Search API. Only the url, title and the snippet of each page will be processed. Then, user feedback for each page is asked to decide whether the given page is relevant to the requested information. Then scores for words in relevant pages are computed to obtain a query dict, which is the reference for the choice of new search words. Then, new search words are extracted from the query dict. And finally, we reorder the newly extracted words to perform the search for the next iteration.
- External libraries used in the code
  - This project uses a key word extraction tool, keyBERT (https://github.com/MaartenGr/KeyBERT). We use this package to assign a score for every word in relevant pages for

further keyword selection.

Description of the query-modification method:

- **Outline:** Our proposed query-modification method first leverage user feedbacks to obtain relevant pages. Then the titles and snippets of all relevant pages will be processed by the method of computing scores for words within all contents, denoting the importance of the corresponding word. Then, the scores will be sorted and the two words with the highest importance scores will be added to the search words for the next iteration. Details for word score computation, new words fetching and reordering will be given below.
- **Word score computation:**
  - We leverage keyBERT to extract the most possible 2-gram and 1-gram key words of the contents. For every keyword, we record the importance (which is the value given by keyBERT), and the occurrence of it among all the retrieved texts using a dictionary.
  - We handle 2-gram keywords and 1-gram keywords in different ways. For 2-gram keywords, we update information (i.e. their importance scores) for the extracted 2-gram, the first word in the 2-gram and the second word in the 2-gram simultaneously. For 1-gram keywords, we directly update information for this 1-gram. Both the importance scores and the number of occurrences are updated cumulatively during the entire search process. Besides, importance scores are added every time it appears in a 2-gram keyword or a 1-gram keyword, while the number of occurrence is updated by pages. To put it in another way, if a word simultaneously appears as part of a 2-gram keyword and independently as a 1-gram keyword for a specific relevant page, then its importance scores will be added twice, while it's occurrence will only be counted once.
- **New words fetching:**
  - By the above word score computation we obtain a dictionary containing all the information for words in relevant pages. This dictionary will first be sorted by

    $$0.25 * importance + 0.75 * occurrence$$

    then we add the first two **new** keywords (notice that in this part we distinguish 2-gram keywords from 1-gram keywords, different from the above computation process.), that is, keywords that are not in the previous search words, to the search words for the next iteration.
- **New words reordering:**
  - The search words will be again sorted by their values in the dictionary using the same expression above. We do this again because this time both previous keywords and new keywords will be counted.
  - In case that the original search words have some interruptive information (i.e. useless for the targeted information based on the user feedbacks) and that such words won't appear in the keyword dictionary, we will assign a 0-value for such words in the reordering process, which aims to give them the least importance in the future queries.
- **Non-HTML FIle:**
  - In the search algorith, we decided to ignore the non-html files as suggested in https://edstem.org/us/courses/34785/discussion/2623177. The search algorithm analyze the snippet/title from the query as the keywords.