Instructions:

- The input for books is the bucket at gs://jack_book_base .
- The output is located in the bucket at gs://jack-result-base as a single txt file.
- All 100 books for the coursework are already in the book input bucket.
- All functions for the program itself are stored on the cloud.
- All functions coded in python.
- The program is ran via google workflow.
- The manual trigger for the google workflow, jack-execute-workflow.py , can be activated by visiting https://jack-execute-workflow-iyvvmy7hfa-nw.a.run.app . This allows you to bypass the need for my credentials to run the workflow. Upon visiting the page, it will appear to be endlessly buffering, however that is simply the website waiting for a response from the workflow, eventually it will display "Anagrams file generated to result bucket", signalling that it has finished, which should take just over a minute usually. Part of this code is taken from https://cloud.google.com/workflows/docs/samples/workflows-api-quickstart#workflows_api_quickstart-python .

The mapper part of the program is represented in jack-map-anagrams.py .

The shuffler and reducer part of the program are both implemented in jack-merge-dictionaries.py .

Jack-clear-data clears out processing data and result data from the buckets before attempting another process.

Jack-get-blobs is used to get the names of the books in the input bucket.

The mapping part of the program is done in parallel. It was decided through trial and error that a suitable system involved 100 max instances, each with a cpu limit of 4 and a memory of 4gib.

For the shuffling and reducing part, we will likely only ever need 1 instance, as it is not done in parallel. We found a cpu limit of 1 with 2 gib of memory to be more than capable.

Due to the small presence of accented characters in the books, utf-8 was found to be an unsuitable encoder. Therefore, we opted to use latin-1, which seems successful in handling all the texts without error.

We have four used buckets:

- Jack-anagram-base for storing the anagrams in individual books after the mapping phase.
- Jack-result-base for storing the result txt file.
- Jack-stop-words for storing stop words.
- Jack_book_base for storing input.

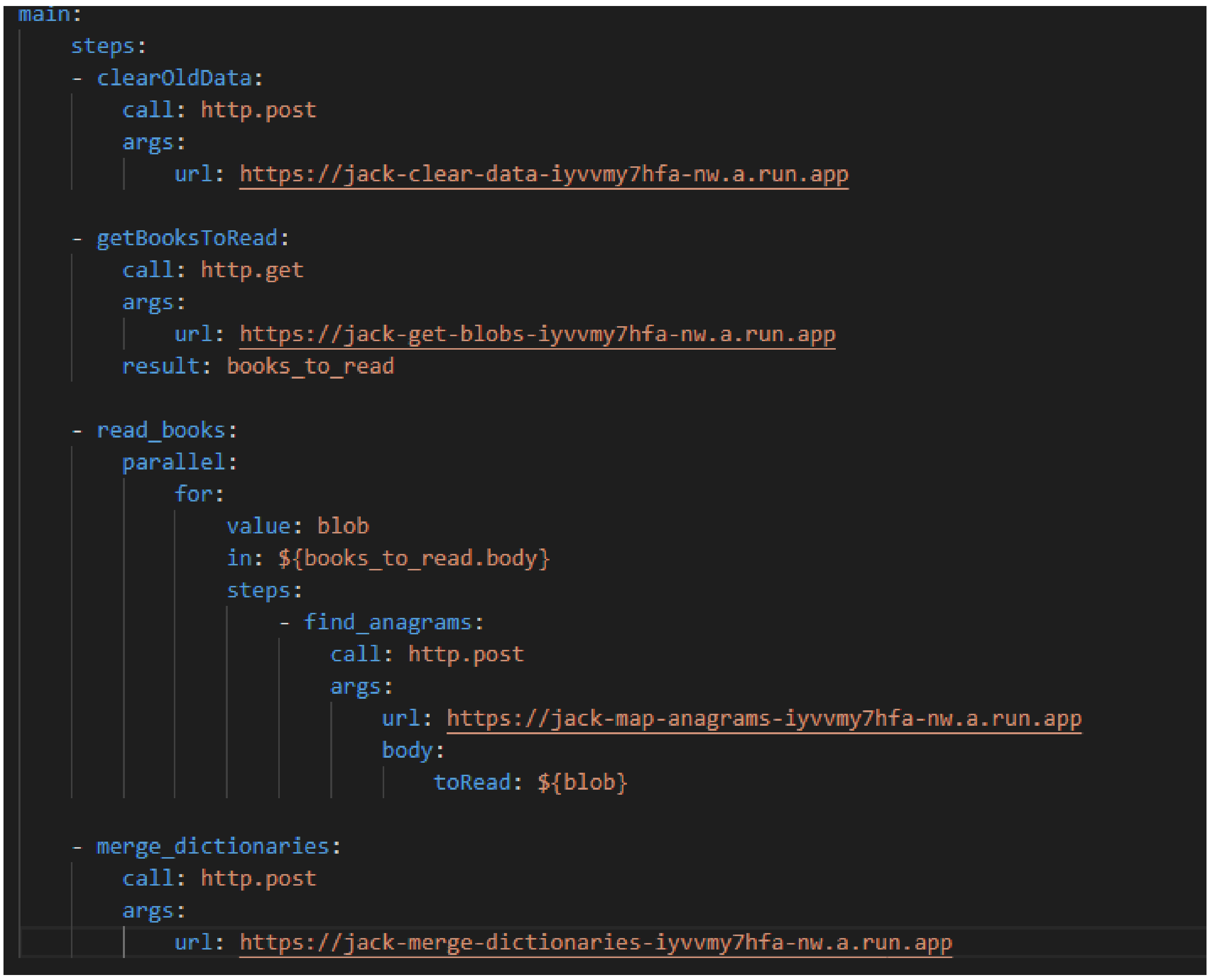
These are all publicly accessible read only.

I have attempted to explain how each function works in their code, as comments. This full commenting is not present in the code in the google cloud, only python files submitted on learn. The actual code is identical in both however.

It was decided that "-" will be interpreted as separating two separate words, rather than joining a single word.

We remove all grammar before scanning for anagrams. With the exception of " ' ", all grammar is replaced with a space as this seemed to give the most sensible results. Some strange non sensical words still appear however due to the nature of some of the texts, as well as multiple languages.

Workflow:



```yaml
main:
    steps:
    - clearOldData:
        call: http.post
        args:
            url: https://jack-clear-data-iyvvmy7hfa-nw.a.run.app

    - getBooksToRead:
        call: http.get
        args:
            url: https://jack-get-blobs-iyvvmy7hfa-nw.a.run.app
        result: books_to_read

    - read_books:
        parallel:
            for:
                value: blob
                in: ${books_to_read.body}
                steps:
                    - find_anagrams:
                        call: http.post
                        args:
                            url: https://jack-map-anagrams-iyvvmy7hfa-nw.a.run.app
                            body:
                                toRead: ${blob}

    - merge_dictionaries:
        call: http.post
        args:
            url: https://jack-merge-dictionaries-iyvvmy7hfa-nw.a.run.app
```

This is our workflow.

We first clear old data.

Then get the names of books in the input.

Then get the anagrams of each book in parallel in order to save huge amounts of time, as there is no need to do them sequentially.

Then we merge the dictionaries sequentially, as we cannot do it in parallel.