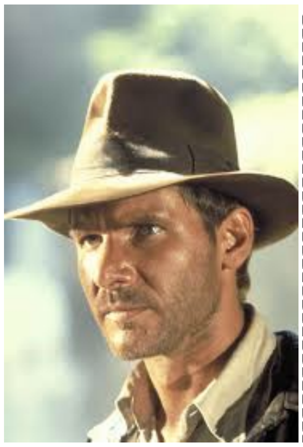
PS3 – Ancient Texts Word Histogram



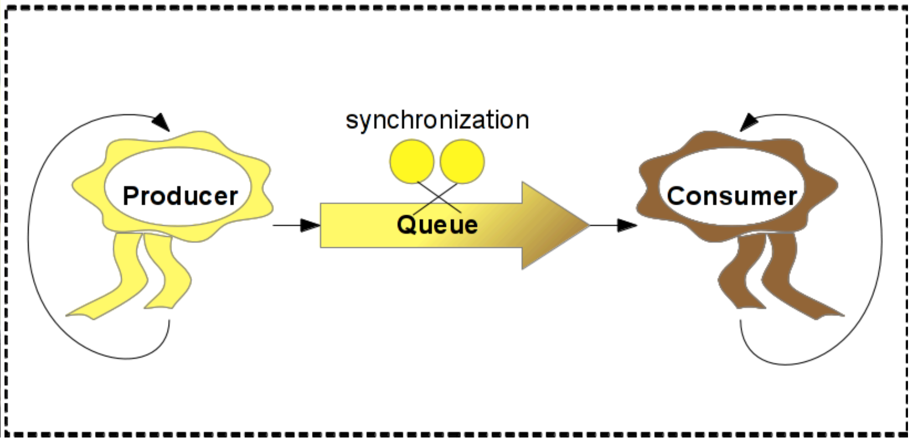


Image sources: https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcQiZpSx9NMeCZNSZJPbgTVb-vGlvk6uXFzSe2b3_oynMsC-l7Q_
http://4.bp.blogspot.com/-9UpZKfJbugY/UFYe4MpX7II/AAAAAAAAAIw/57jLbqLQn6c/s1600/producer_consumer.png

Description

Your employer, Indiana Jones has given you a job in the study of philology – that is –the study of language in written historical texts. One way of studying historical texts, is to create a histogram of the word frequency of each, so we can understand how language has changed over time. In this assignment you will be implementing a word histogram in Python using the producer-consumer model.

Your Task

It is recommend you get started early on this programming project. We have broken up the problem into two sections to help you plan accordingly. You will be working individually on this project.

Task 1 – Python File Parsing

Write a Python function or class that can open a text file and read a list of words (*tokens*) separated by whitespace. Any sequence of non-whitespace characters counts as a token. Then have a data structure that can output the frequency of each word found in that file. Your histogram should be case insensitive; convert everything to lower case.

Example File (words1.txt):	Example Output:
Cat dog fish cat dog dog.	cat 2 dog 2 dog. 1 fish 1

Task 2 – Python File Reader

Write a python program that makes use of Task 1:

word_histogram [-m max_threads] [-o outfile]

`word_histogram` computes a combined histogram for multiple files specified on standard input.

`-m` specifies the maximum number of threads that can be running at one time, see below. Default is 10.

`-o` specifies the file in which the resulting histogram will be stored. The histogram is exactly like the one from Task 1, but it is based on occurrences in all the input files. `-o -` means this result goes to standard output (after all other output to standard output has happened, see below). Default is standard output.

The program reads its standard input (a text file) line-by-line. Each line contains a path name of a file. The histogram for that file is sent to standard output in the following format:

```
file1.txt:	cat 2
file1.txt:	dog 2
file2.txt:	fish 1
```

The name of the file is followed by a colon, a tab character, a word, a space, and the word's frequency. It is possible that when outputting these histograms, lines from different files will be interleaved, which is fine for this assignment. If the same file is given twice on the input, then you can treat them as two files with identical contents, and you can specify what is correct so long as there is no crash or deadlock.

Finally, the combined histogram for all the files is sent to the designated output file (from the command line).

All histograms should be printed in sorted alphabetical. For example, running

`word_histogram` –o words-output.text < test1.text

might go like this:

test1.text contains:	Example output in words-output.text):
words1.text words2.text	cat 4 dog 4 fish 2

Standard output would get a histogram for words1.text and words2.text as shown above.

Key Features

- You will need to decide when it is safe to write to words-output (Avoid race conditions). In particular, per-file histograms and the final histogram should not be interleaved if all go to standard output.
- If a pathname can't be read, there should be a helpful error message on standard error, and the program should keep going. It must not terminate due to such an error, whether by crashing or exiting, and there must be no deadlock.
- You should not have duplicate entries in any histogram (i.e. the word 'cat' should only display in any one histogram once).
- You must not exceed the number of threads (represented by the argument –M)
- Use the Producer-Consumer model
- The maximum number of threads should default to 10. You have (at least) two compliance strategies:
    - You may use a thread pool: Allocate max_threads threads at the start, or
    - You may generate threads on demand, being careful not to exceed the maximum specified on the command line
- You will want one thread to act as the producer.
- The minimum number of threads is 2 (one producer and one worker). If the program is called specifying fewer than 2 thread, the program should announce on standard error that it's running with 2 threads and then proceed to process the data.
- You should be careful to avoid deadlock. We will be testing this on very large samples.
- Ignore any empty lines in the input files.
- Ignore empty lines on standard input
- It is up to you to decide the data structure in which you'll store each word and its corresponding frequency.
- At your discretion, you may create other modules in Python that your program imports. Submit all such modules. We'll run your `word_histogram` program in a directory with all your files available. We'll accept up to 6 files total for the assignment (if you need more, please come and discuss your strategy with us).

Evaluation

- We will compare your output against several test text files to make sure the statistics match up
    - You should make or generate your own files (words1.txt, words2.txt, etc.) to test your solution.
- We will run large tests with different combinations of processes (the –m flag)

How to Submit

We will be using the provide system for this assignment.

Create a file called README. In it, write

- your name
- the date
- approximate amount of time the assignment took you
- a list of anyone with whom you collaborated

Be sure your files don't contain lines longer than 80 characters, and do not use tabs for alignment (use spaces). Follow the Python PEP advice of indents of 4 characters.

```
provide comp50cp ps03 README *.py
```

Going Further

Finished everything in this assignment? Try these items just for fun and report results back to class.

- Implement your solution in C/C++ and report back on the performance (time to completion)
- Have your output update in realtime.
- Update your data structure so that new word files can be added at any time

Resources

- If you need additional resources on Python, please refer to the documentation here: https://www.python.org/doc/
- Specific documentation on threading in Python is here: https://docs.python.org/2/library/threading.html
- Of course, Piazza: https://piazza.com/class
- Python course: https://developers.google.com/edu/python/