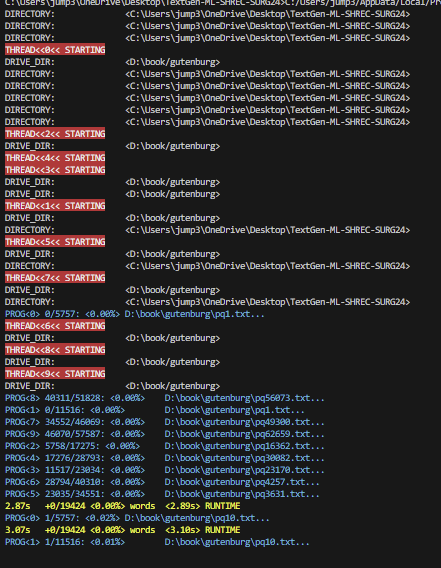
before (serial)

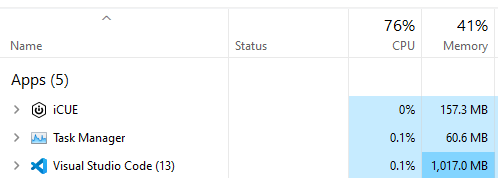




would’ve taken 50 days

**after (parallel)**10 threads (41 days)





Had the idea of making a Red-black tree with the dictionary but was afraid of RAM usage. Solution was to keep data in a text file and just load each line at a time into RAM. This was extremely slow, even with parallelization. Serial: 50 days for Gutenberg. Para: 42 days.

Also was going to originally make a separate ML model to detect if a word was “proper” but was able to sort out words so well that this wasn’t needed. If there was a small hiccup, the neuron connection between that word would be so low that it practically wouldn’t exist in training anyway.

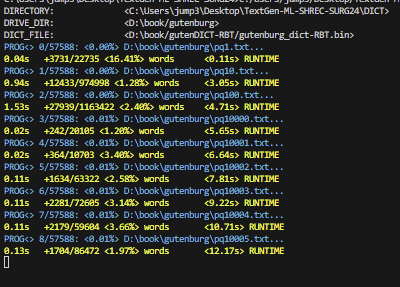
Dictionary types

* RAMmin txt file: word per line, load a line (word) into ram one at a time
* Too slow
* RAMmax Red-black Tree: key by string
* RAMmax Red-black Tree: key by int encoded string
* Tiktoken:Gpt2
* If a string is too long it has multiple tokens and is longer then just the string
* RAMmax enumerate sorted list: proper coding ‘dictionary’ term
* Would have to recreate object for every addition, doesn’t really work for this approach
* DOES absolutely make sense for **saving dictionary** (our term) as a nonRAM file
* Convert into sorted list, enumerate, save as ‘.bin’
* Load back into a new file
* Any dictionary would have to be converted to this anyway for the model to use

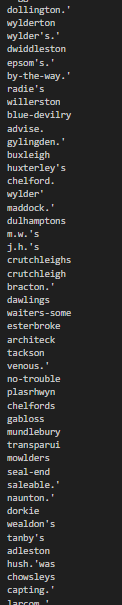
Dictionary Conclusion:

* Load whole data into RAM, convert into pure words
* RAM Red-Black Tree (key by string), save from RAM as enumerated sorted list ‘.bin’ file
* Load into a model from this file

After RedBlack Tree with maxRAM



unfathomably faster



Some books present challenges unheard of in other examples like this author using lots of " .' "s

Was thinking of putting the red-black tree into something like how the morse-code tree works: like a max heap of frequency. But then I realized that the search for that is no different then just the sorted list. Even if you made it a balanced tree, it would be more effective if characters were more evenly spread out with out of all characters (international included) but it’s a very uneven distribution. Kept a Red-Black Tree.

The prepare for by-char turned out fine, completed with a freq-sorted list in under an hour. The by-word had a very large initial upkeep before it started running the files and had to stored in the uint32 format instead of the uint16 format like by-char. Afterwards it ran at a similar rate to the by-char.

6/10/24

Working on nanogpt hack

Getting rid of the ‘val’ set does nothing at all in performance and isn’t used for anything. If anything removing it and making it all training data would make it train better.

A screenshot of a computer screen

Description automatically generated

Left: removed, right: original