Simplesearch

Python version: 3.7
Testrun: Macbook Pro OS 10.15.1

Introduction and instructions:

This script prompts the user for a directory and sets of words or phrases, and prints the percentage of the given words or phrases that are found in each .txt file in the given directory. To run from terminal, write

```
> python simplesearch.py directory
```

where *directory* indicates the desired directory. Alternatively, if you simply write

```
> python simplesearch.py
```

without giving a directory, the program will prompt you for one, and also give you the opportunity to view and change the standard settings, or load an earlier session (see below). When a directory is given, the program will read the words from all .txt files in the directory and store them in a tree structure (see section Data Structure below). The program will then prompt you for a set of words (the query) separated by spaces, and print the percentage of the given words that are found in each .txt file in the given directory. For example, if your directory contain the files file1.txt, file2.txt and file3.txt, and you enter the following query

```
> my neighbour had a horse in her house
```

then if file3 contains all eight words, file2 contains none of the words, and file1 contains the words "horse", "in", “her” and “house” but not the remaining ones, the program will print the following

```
Scores:
file3.txt: 100.00%
file1.txt: 50.00%
file2.txt: 0.00%
```

and prompt you for a new query. This cycle will then repeat until you enter the query `:quit` to exit the program.

You can use the program to search for full phrases rather than individual words by using the flag `-p`. For example, if you want to search for the phrase “horse in her house”, i.e. you want to know if the words *horse, in, her* and *house* occurs sequentially in any of the files, you can give the query

```
> horse in her house -p
```

Then, if file3 contains this phrase and file2 contains the individual words but not sequentially, you will get the following output:

```
Scores:
file3.txt: 100.00%
file2.txt: 0.00%
file1.txt: 0.00%
```

You can search for several phrases by separating them with a comma, like this:

```
> where did she get a horse, must have nicked it -p
```

then if file3 contains the phrase "where did she get a horse" and "must have nicked it", and file2 contain the second phrase but not the first, and file contain none of the phrases, you will get the following output:

```
Scores:
file3.txt: 100.00%
file2.txt: 50.00%
file1.txt: 0.00%
```

Phrases can also be individual words, so

```
> he, used, to, chase, cars -p
```

is equivalent to

```
> he used to chase cars
```

The current session, including the tree structure (see section *Data structure* below) can at any time be saved by typing

```
> :save
```

and following the instructions. The session can then be loaded again when restarting the program later. This allows you to skip over the process of generating the data structure from scratch for every session, which may be convenient for large files.

Settings:

You can access the settings options by first running the program from terminal without including a directory, like this

```
> python simplesearch.py
```

before typing

```
> :settings
```

when prompted. The program will then display the following:

```
Default settings:
```

```
case_sensitive = False
end_indicator = $
mamixum_report = 10
ignore_punctuation = True
```

These are *keywords* and *default values* that can be changed by the users. The set of changeable keywords are as follows:

- *case_sensitive (bool)* indicates whether or not the storage and searches should be case sensitive.
- *end_indicator (str)* is the character that indicates the end of a word (see section *Data structure* below). If you know that the default option (*$*) is found in one or more of your files, you must change the *end_indicator* to a different character (if you don't, the program will tell you to restart and do so when encountering the character in your file)
- *maximum_report (int)* is the maximum number of files that will be reported to the screen, so that e.g. if maximum_report = 10, only the names of the ten files with the highest score will be reported.
- *ignore_punctuation (bool)* indicates whether punctuation in the files should be ignored. For example, if you search for *> developer* when *ignore_punctuation* is set to False, you will not get a match with the word developer if it is associated with a full stop or comma or other punctuation marks in one of the files (say, when it occurs at the end of a sentence). You will probably want to keep this setting to *True* when searching articles, but it might be useful to turn it off when searching other types of documents, where you might for instance be interested in knowing whether the number *22.3* or the link *github.com* occurs.

To change a keyword, simply write

```
> keyword = new_value
```

e.g.

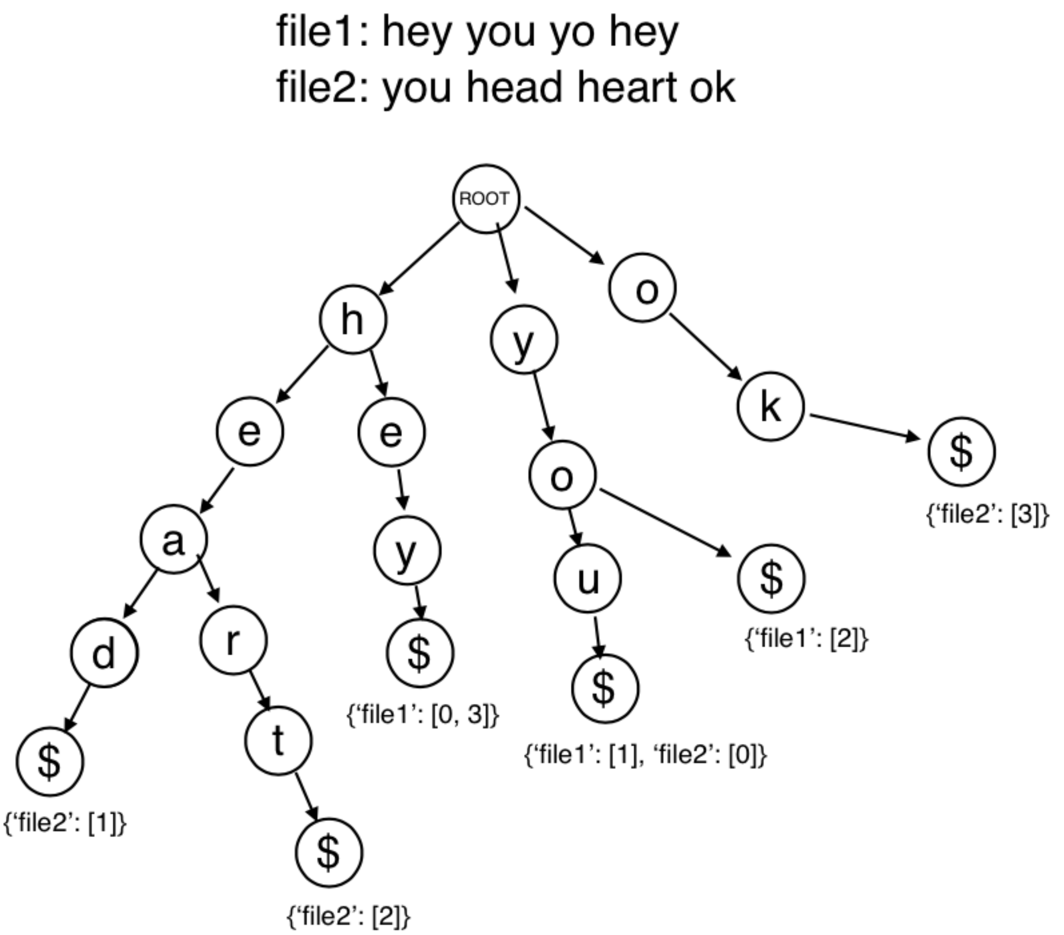
```
> case_sensitive = True
```

When you are done changing the settings, type

```
> :done
```

Data structure:

The words from the files are stored in a tree structure to minimize memory usage, and to maximize efficiency when searching. Each letter is stored as an attribute of an instance of class Node, and each such node instance also has an attribute called children, which is a list of that node's children nodes. The characters of a path from the root node via subsequent generations of children spells out a word found in one or more of the files, and the set of all such paths spells out all of the words in all of the files. All words are given an *end_indicator* character (by default *$*) that indicates the end of the word. This character is the *character* attribute of a special kind of node called a *leaf node*, which has no children. Instead, the leaf node has an attribute called *word_indices*, which is a dictionary for which the keys are the names of files that contain the word spelled out by the path from the root node to that leaf node, and the values are lists of indices indicating at which position(s) in each file the word is found. For example, *{"file1.txt": [2, 5, 15], "file2.txt": [3]}* indicate that the associated word occurs as word number 2, 5 and 15 in file1, and as word number 3 in file2 (zero-indexed). See the tree below for an example tree for two simple files.



The *word_indices* attribute allows me to store all the words from all the files in a single tree structure rather than creating one tree for each file, which would have been more expensive in terms of both memory usage and time spent searching. The method *retrieve_words* reverses the process, and returns a dictionary with lists of the words of each original file in the original order (including punctuation if *ignore_punctuation = False*). In the current version of the program this method is used only for testing purposes (see section *Testing* below). At present, the only information that is lost when converting to and from a tree structure (given that *ignore_punctuation = False* and *case_sensitive = True*) is that spaces, newlines, and tabs are not differentiated, but it would not be difficult to modify the program so as to also (optionally) store this information. Hence, by also including a simple additional method to write the retrieve word lists to files and combining with the :save and :load options, the program can quite easily be converted into a lossless compressor for multiple text files.

Testing

I have included a file *simplesearch_test.py,* which aims to test whether the information stored in the tree structure is representative of the original files. It achieves this in four steps. Firstly, it creates a tree structure of the files in the desired directory. Secondly, it loads the individual words from the same files directly (without generating a tree structure). Thirdly, it translates the tree structure back into lists of words in the original order. Finally, it checks whether the lists loaded directly are identical to the corresponding lists (file by file) translated from the tree structure. To run a test from terminal, quit the program (if you haven't already) and type

```
> python simplesearch_test.py directory
```

where *directory* indicates the desired directory.

Øystein Kapperud, December 2019