PLH202 Exercise 1

Odysseas Stavrou 2018030199
Technical University of Crete
March 2020

Hierarchy:

The program consists of 3 classes and 1 main module.

main.py: Runnable from user. Entry point to the program

Classes:

Node.py: Has 3 attributes. Data, Next Node, Previous Node

Linked List.py: Has only 2 attributes, the head Node and the tail Node. However the Linked List object implements lots of methods and functionalities that will come in handy later as we approach each objective.

Index Creator.py: Has 3 attributes describing the words to be indexed in the output file and 1 method to create said file

Flow:

1.) When the user runs **main.py** it initializes to the default settings and then parses the argument passed to it(if not then it prompts the user for a filename after calling main()). It prompts the user for changing the defaults values.

2.) After creating the Linked List it populates it from the contents of the provided file line-by-line. If a certain line is more than a specified length then is takes ONLY the first n characters specified.

3.) Sets-Up the head and current node and then prompts the user with the menu for command input. Depending on what command the user has entered the correct method is called(either from the Linked_List Class or the Index_creator).

Usage:

Make sure you have the latest python3 on your system!
If you are on Linux run as such in terminal:

```
chmod +x main.py
./main.py filename
```

OR

```
python3 main.py filename
```

If you are on Windows run as such from cmd or ps:

python3 main.py filename

Note: filename argument is optional

Command Handling and Implementation:

- For basic moving around in the list (meaning changing the current Node)(see commands "^, $, -, +") we just change the current node (after checking) either to next/prev or we grab the head or tail from the list

- For insertion/deletion (commands "a,t,d")to and from the list we have 3 methods in the Linked_List Class. Since the list is Doubly Linked we do not need to traverse the list until we found that Node. We can cut/sew the new node in between them

- Printing commands("l,p,n,=,#") are implemented with 2 methods and 1 numbering Flag inside Linked_List Class. One method is for printing just the Node's data. It traverses the list until that Node counting the Nodes in between and prints its Data. If the numbered Flag is set then it bypasses the list and just prints the Data instead and returns. The other method is for printing the WHOLE list. Same logic as before. For the stats we follow the same notion but sum all characters as well

- For Writing and/or Exiting ("q,w,x"), depending on the command we OVERWRITE the whole list over the current file and if we should exit then the method exit() is called from the imported library 'sys'

- The command "c" invokes the create_idx_file() from the Index_Creator Class

- The leftover commands "s,v,b" use the above created file in combination with methods residing in main.py

Word Indexing:

- The program traverses through the list and takes each word that meets the requirements specified in the object, counting the line number along the way and appends the pair (w,l) into a list. It then sorts the list alphabetically

- Taking every pair of (w,l) the program pads the word with null characters to reach a specific length and then encodes the whole word into bytes. Integers in Python are 28 bytes long because it's a full object so we need to use its method to_bytes() to get a fixed sized byte chunk

- We append each entry into a bytearray() and each time we fill it up we write it to the file

- For viewing the created file we follow the reverse procedure. We read one sector from disk, split it into entries, grab word and line number from each entry, strip trailing null characters from word, decode the word and use from_bytes() to recreate our integer.

Word Searching:

- For the serial search the logic is exactly the same as with viewing the file with the difference that we append to a list the number of line that the word is found.
- For the binary search firstly we have to check the size of the file and we divide with the sector size so we get the number of pages in the file (always integer)
- We then can apply the binary search principle using our pages as indexes
- However as soon as we have a hit on 1 page that contains our word we need to go left and right sampling every line that the word appears as well

Bad Practices:

Python is a very powerful and simple language. However I should have split the program into more Classes and Methods, because due to its simplicity I have overcomplicated the main module with a lot of functionalities that were better off somewhere else. Also by having more Classes we minimize the parameters passed into methods.

Software Used:

- Visual Studio Code
- PyCharm Ultimate from JetBrain
- Sphinx
- Various Extensions for the above software

Used Libraries and Documentation:

- sys for system interaction mainly sys.exit()
- os for OS interaction (checking for file existence and getting file size)
- math for the ceiling function used in binary search
- punctuation from string for the special characters set
- Project's Documentation lies into html directory. Open index.html
- https://sphinx-doc.org/ for the sphinx documentation
- https://docs.python.org/ for the python documentation