605.202: Introduction to Data Structures (Spring 2022)

Mukul Sherekar

Lab-2 Analysis

Due Date: March 29, 2022

Date Turned In: March 29, 2022

Justification of data structure/design:

Lists are the only data structure used in this lab. I used lists to store the given prefix string in order to manipulate it. Manipulation included reading the string to find the pattern of operator-operand-operand and store the new elements of post-fix string. I could not implement a better way of returning (operand-1 + operand-2 + operator) string so using lists became mandatory for me. I realize it is not the most efficient way of solving this problem but got the job done. Also, the list data structure helped in error handling. Once the pre-fix string was stored in the list, I could loop over to find invalid characters.

This package has 2.py files (lab2.py and convert.py), a __main__ and a __init__ file. convert.py is the main part of the package. It contains a function to convert prefix to postfix using recursion. lab2.py is the script that gets executed at command line. It contains a function whose arguments are the name of input and output files. It reads the input file, converts each line into post fix, handles errors and prints postfix string onto an output file. The __main__.py is the script that executes import of arguments of for process_files method (in lab2.py). __init__.py exexutes import of process files method everytime module gets executed.

Since conversion using recursion is in a separate file, it can be reused later or modified which provides flexibility. As an example, recursion can be changed without affecting other parts of package and compare performances. Finally, lab2 script which converts at file level, can be changed to accommodate any error handling at file level Overall, design of this package is very flexible.

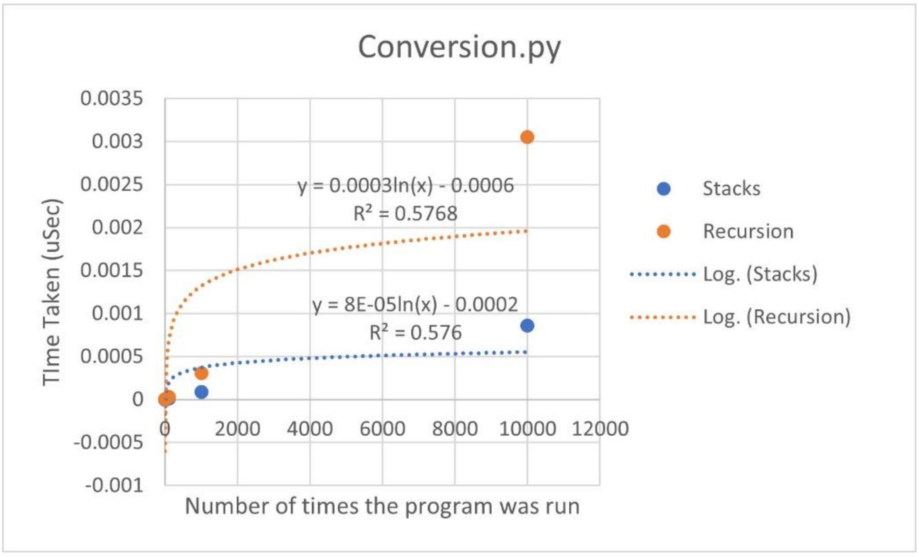
Description of recursion

- Each line in the file was read one character at a time (convert module)
- Recursion meant modifying string if there were a operator followed by two operands
- Modification meant changing operator-operand1-operand2 into operand1-operand2-operator keeping rest of the string as it is
- Base case was to return the string if the first character was not an operator
- Recursive call was to keep modifying the string

Does recursion makes sense?

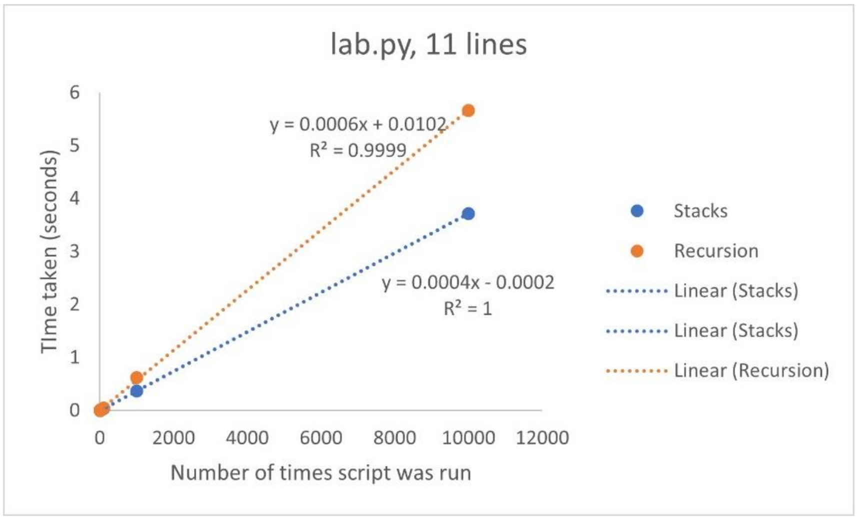
In order to show justification of why recursion makes sense or not, I calculated time taken by convert.py and lab.py of lab-1 (stacks) and lab-2(recursion). I used two types of input files – one had 11 pre-fix strings and other had 110 prefix string (10 x 11). I used timeit module whose arguments are code that gets executed and number of times for which code gets executed. Basically, number of time a script is run is akin to input size.

***** Comparison of convert.py from lab-1 (stacks) and lab-2(recursion)*****

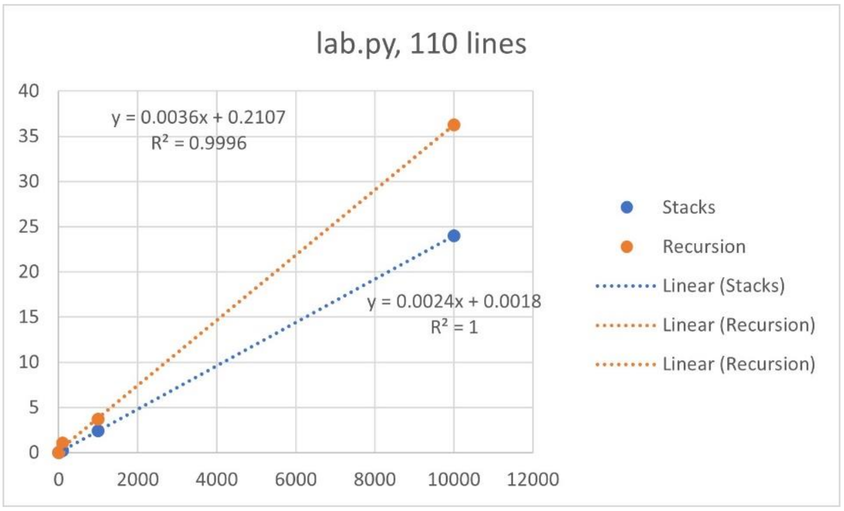


- Both scripts follow logarithmic time complexities i.e **O(logn**)
- Stacks take less time than recursion
- Graph shows that as the number of times the script has to be run i.e as number of files increase, stacks are faster

<u> Comparison of lab1.py (stacks) and lab2.py(recursion) to convert 11 prefix lines into postfix**</u>**



<u> Comparison of lab1.py (stacks) and lab2.py(recursion) to convert 110 prefix lines into postfix**</u>**



- Both scripts follow linear time complexities O(n)i.e time taken increases linearly with number of times the script has to be run
- Above two graphs show that stacks perform much better in reading lines from a file and converting them into postfix
- Number of lines only highlight efficiency of stacks

  Hence, one can draw the conclusion that, in terms of time, stacks perform much better than recursion or to be precise, the way I used stacks and recursion using lists, stacks make more sense than recursion.

Time and space efficiency

- As described in above section, my recursion code follows **logarithmic time** complexity
- As far as space complexity goes, my recursion convert code has 4 arrays out of which size of two is directly proportional to the length of prefix strings. Hence, it will follow **linear space complexity**.
- Recursion related space complexity will depend on recursion calls which is **linear dependence**.
- **So, space efficiency will have a space complexity linearly dependent on product of array size and recursion calls**.

What did I learn

- I learned the importance of writing the pseudo-code and ideas in general using pen and paper
- I got an idea of splitting a big problem into smaller problems
- I learnt how to use time and timeit module
- I learnt how to calculate and analyze space and time complexities

What would I do differently?

-Explore different data structures for recursion and keep an open mind about it

- Try avoiding arrays

-Read one character at a time for recursion