Assignment 2
IN711 Algorithms and Data Structures

Introduction

Select problems from the list below and implelment solutions to them in Python. Be sure to read each problem carefully and be sure you understand what you are asked to do before starting work on it. Ask for clarification if you have any questions about the problems. It is impossible to fairly mark submissions that do not correctly address the problem so they cannot be accepted.

Your goal is to solve the problems with elegant and efficient code. In particular, pay attention to your choices of data structures and be sure that your algorithms are implemented cleanly and efficiently. Your code should also be easy to read, since obfuscation does nothing to improve efficiency.

You are encouraged to research and discuss these problems. The text contains relevant information about some of them and you should use it as a reference. There is useful and interesting information about some of the algorithms involved on the Internet. You are free to reuse any code that you wrote earlier in the semester if it is useful. Be sure that the code you submit, however, is your own.

Many of the problems direct you to implement a particular class or method. You should also include appropriate "main" code that demonstrates how the class or method works.

Note that the point values of all of the problems together exceed 100, but your goal is to earn 100 marks. You are not necessarily expect to do every problem. You may also opt to submit an unfinished problem for partial credit.

Place all your work in a directory named `assignment2` that you will commit and push to your `ads` repository on GitHub. Each problem's code must be placed in a clearly labeled subdirectory of this directory.

1 Problems

Problem 1: FizzBuzz (5 marks)

Write a program that writes the numbers from 1 to 100 (each on a separate line). But for multiples of three print "Fizz" instead of the number and for multiples of five print "Buzz" instead of the number. For numbers which are multiples of both three and five print "FizzBuzz". Think carefully about the most efficient and elegant way to code this problem.

Problem 2: Reverse a linked list (10 marks)

Write a simple singly linked list class and include a method that reverses the order of the nodes in the list. Do not use any auxiliary structures like a second list. Simply traverse the list, modifying the nodes as you go. Be sure that the reversed list is a correct singly linked list, i.e., all other list operations still work, and reversing the list twice should give you exactly the original list.

Problem 3: Breadth-first tree traversal(10 marks)

Implement a basic binary tree class and provide a method that performs a breadth-first traversal of the tree. That is, it shall access every node on a particular level of the tree (from left to right) before continuing to the next level.

Problem 4: Postfix calculator (15 marks)

A *postfix* arithmetic expression is one in which we supply the two operands followed by the operator. Thus, the infix expression 1 + 2 is written as 1 2 + in postfix notation. The advantage of this is that brackets are not needed to show order of evaluation. For example, the infix expression (3 + 7) * 4 is simply written 3 7 + 4 *.

Write a program that accepts postfix artrithmetic expressions (You need only support the binary operations addition, subtraction, multiplication, and division) and prints the evaluated result.

Problem 5: In place heap sort (20 marks)

Write an in-place heapsort function as described in Chapter 9, section 4.2 of the text.

Problem 6: Inverted file (15 Marks)

An *inverted file* is a critical data structure for implementing a search en- gine or the index of a book. Given a document D, which can be viewed as an unordered, numbered list of words, an inverted file is an ordered list of words, L, such that, for each word w in L, we store the indices of the places in D where w appears. Implement an efficient algorithm for construct- ing L from D.

Problem 7: Quicksort (20 marks)

Implement a nonrecursive, in-place version of the quick-sort algorithm, as described at the end of Chapter 12, Section 3.2.

Problem 8: Word Ladders (35 marks)

Word Ladder is a puzzle game reportedly invented by Lewis Carroll (the author of Alice in Wonderland). The goal is to take two words - for example "fool" and "wise" - and progress from the first word to the second by sequentially changing exactly one letter of the word. All intermediate character strings must be real words. For example, a solution to the above word ladder is: fool, pool, poll, pill, will, wile, wise.

Build a program that computes word ladders. Your application should accept two words of **exactly** five letters and compute and display a word ladder between the two, if there is one, and appropriate feedback if there is not. For this exercise, please use the file âĂŸdictionary.txtâĂŹ, available in this repository. The words contained in that file should be viewed as the set of all possible legal 5-letter words in the universe

2 Marking schedule

Each problem submission will be marked accodring the to the following schedule:

- Complete and correct solution: 20%
- Coding style, commenting, readability: 10%
- Data structure choice and implementation: 20%
- Algorithm elegance, clarity, and efficiency: 40%
- Unit tests: 10%

3 Submission

Commit and push your code submissions to your GitHub repository by 5:00 PM on Tuesday, 17 November.