CSCI 230 -- Project 3 (Text Processing)

Due Monday, 05/11/2020 (MW Section)
Due Tuesday, 05/12/2020 (TTh Section)

This project is designed to help you understand text processing by implementing different text processing algorithms. Perform both Standard Trie and Huffman Coding problems as describe below.

Standard Trie: Implement a standard trie for a set of ASCII strings including a terminating character for each word. You might want to look at the trie in zyBook. Create a class that has a constructor that accepts the name of an input file as a parameter (a string), and the class should have an operation that test whether a given string is stored in the trie. The driver should allow user to specify the input data file, output number of words in the trie, and then use a loop to check for a few words (try the following words: honor, honour, government, computer). Output yes or no for each search word. Use the text file ***usdeclarPC.txt*** as an input file and you need to format the words to lowercase and remove extra characters like comma, periods, etc.

Huffman Coding: Implement a compression scheme that is based on Huffman coding. Write a program that allows user to compress a text file. For example, the two files, ***moneyIn.txt*** and ***moneyOut.txt***, are the normal text file and compressed file respectively (notice the compressed file is in text format so we can easily see what is going on). Given a normal input text file, you need to generate the compressed text file. You should utilize a class named ***HuffmanCoding*** with an appropriate interface.

File ***moneyIn.txt*** (enter key and then one line of text with no enter key at the end)

```
more money needed
```

File ***moneyOut.txt*** (enter key first and then space character)

```
 0110
  1011
d 100
e 11
m 001
n 000
o 010
r 0111
y 1010
*****
Number of characters: 18
Number of bits: 54
011000101001111111011001010000111010101100011111001110 0
```

Extra credit: Earn up to 5 additional points if you perform one of the following:

- **Improved Standard Trie**: Modify the standard trie so the search would return the number of times a word occurs in the text. Output something like "Found X 3 times" or "Found Y 0 times".

- **Decompression with Huffman Coding**: Given a compressed text file like moneyOut.txt you need to generate the normal text file by reconstruct the Huffman coding tree and then use it to decompress the bits to a string. Add this option to the regular Huffman coding program. Use input parameters for main as follow: inFileName outFileName C/D (i.e., set up arguments via MSVS, Eclipse, or your IDE to send data into the program so no input from keyboard; use C for compression and D for decompression. For example, use "moneyOut.txt moneyIn.txt D" for extra credit and "moneyIn.txt moneyOut.txt C" for regular part.

Team Project: It is not required but it is recommended that you work in a team of two members. You must sign up by Monday, 04/27/20, for MW class or Tuesday, 04/28/20, for TTh class if it is going to be a team project. Make sure to split the responsibilities equally and work together. You do need to provide a short description on who did what and your experience as a team project. It is possible that all team members may not receive the same score. **If it is a team project, you need to implement one item of extra credit options as a required component and you can optionally implement one more item for extra credit**.

Please provide documentation and applying good coding style because it is part of the grade. You must come up with enough test cases since the test cases are also part of the grade. Please submit the following items (items 1 – 4 as one pdf file on Canvas):

1. Title page with name, class, project number, and relevant information about your program (compiler and system used, file names).
2. Notes about your program (status of your program and lessons learned).
3. Printouts of any input/output.
4. A printout of the source code.
5. A copy of your source code on Canvas (.h /.cpp or .java files).

Your program will be graded as follow:

- Correctness/Efficiency: 40 points
- Test Cases: 5 points
- Documentation/Coding Style: 5 points
