CSCI205 Computer Science III
Lab Assignment
Binary Trees, Priority Queues, Heaps and HashTables: The Huffman Code

Create a C++ application to compress character data using Huffman Compression. The program input is a string of any length. The output of this program can simply be console output showing the compressed bit string. The compressed bit string can be created with a char array (or string) where each character is a 1 or 0. You are not required to write binary data to a file, but certainly explore if you'd like.

The following is a general layout. These signatures are not mandatory . . . only guidelines. Make appropriate design choices yourself and include ample comments explaining your choices. The only requirement for this project is that it must be object oriented with the following major classes.

Classes:

BinaryTree<T>: You can use the Binary Tree from this week's repo, from Runestone or write your own. To store both the character and the count as a single T you could define a HuffmanNode struct of: int, char

Heap<T>: Using a vector as the underlying memory component, build a "min" Heap class. Use bit shifting to calculate parent and child references. This will be the priority queue for building the Huffman Tree. This will end up being a ***PriorityQueue (Heap) of BinaryTrees***.

PriorityQueue<T>: If you want to be explicit, you can define a PriorityQueue class that uses composition or inheritance to use the Heap class mentioned above.

Huffman: Builds a Huffman Tree from string input. Include the following behaviors

- **histogram():** Creates the character frequency histogram.
    - Use your own HashTable from previous labs for the histogram
- **build_tree(histogram):** Builds the Huffman Tree
- **string compress(string):** Compresses the string input
- **string inflate(string):** Inflates the string input

main.cpp: Prove that you can compress and inflate string data using the classes mentioned above. All I'm looking for is the stream of compressed bits in terminal output.

Submission: All code must be error and leak free. Add a read me file if there is any special information the grader must know about building or running your program.