Huffman Code Generator

Instructions

For Programming Project 3, you will be generating Huffman codes to compress a given string. A Huffman code uses a set of prefix code to compress the string with no loss of data (lossless). David Huffman developed this algorithm in the paper “A Method for the Construction of Minimum-Redundancy Codes” (http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf)

A program can generate Huffman codes from a string using the following steps: · Generate a list of the frequency in which characters appear in the string using a map · Inserting the characters and their frequencies into a priority queue (sorted first by the lowest frequency and then lexicographically)

- Until there is one element left in the priority queue
  - Remove two characters/frequencies pairs from the priority queue
  - Turn them into leaf nodes on a binary tree
  - Create an intermediate node to be their parent using the sum of the frequencies for those children
  - Put that intermediate node back in the priority queue
- The last pair in the priority queue is the root node of the tree
- Using this new tree, encode the characters in the string using a map with their prefix code by traversing the tree to find where the character’s leaf is. When traversal goes left, add a 0 to the code, when it goes right, add a 1 to the code
- With this encoding, replace the characters in the string with their new variable-length prefix codes

In addition to the compress string, you will need to be able to serialize the tree. Without the serialized version of the Huffman tree, you will not be able to decompress the Huffman codes. Tree serialization will organize the characters associated with the nodes using post order. During the post order when you visit a node,

- if it the node is a leaf (external node) then you add a L plus the character to the serialize tree string
- if it is a branch (internal node) then you add a B to the serialize tree string

For decompression, two input arguments will be needed. The Huffman Code that was generated by your compress method and the serialized tree string from your serializeTree method. Your Huffman tree will have to be built by deserializing the tree string by using the leaves and branches indicators. After you have your tree back, you can decompress the Huffman Code by tracing the tree to figure out what variable length codes represent actual characters from the

original string.
So, for example, if we are compressing the string “if a machine is expected to be infallible it cannot also be intelligent”:

Our compress algorithm would generate the following codes for the characters:

Character	Frequency	Prefix Code
(space)	12	00
a	5	1011
b	3	10101
c	3	11000
d	1	010000
e	9	100
f	2	01011
g	1	010001
h	1	010010
i	8	011
l	6	1101
m	1	010011
n	6	1110
o	3	11001
p	1	010100
s	2	10100
t	6	1111
x	1	010101

Our code would be:
01101011001011000100111011111000010010011111010000011101000010001010101010010011000111110001000000111111001001010110000011111001011101111011101011101011011000001111111001100010111111011101100111111001011111011010011001001010110000001111101111001101110101101010001100111011111

And our serialize tree would look like:
L LdLgBLhLmBBLpLxBLfBBLiBBLeLsLbBLaBBLcLoBLlBLnLtBBBB

You will need to create one class for this project: HuffmanTree for the compression, decompression, and serialization that uses a linked binary tree. You are given a Heap-based
Priority Queue for the sorting. You are allowed to use the STL map, vector, and stack, but not the STL priority queue

Abstract Class Methods

`std::string compress(const std::string inputStr)`
Compress the input string using the method explained above. Note: Typically we would be returning a number of bits to represent the code, but for this project we are returning a string

`std::string serializeTree() const`
Serialize the tree using the above method. We do not need the frequency values to rebuild the tree, just the characters on the leaves and where the branches are in the post order.

`std::string decompress(const std::string inputCode, const std::string serializedTree)`
Given a string created with the compress method and a serialized version of the tree, return the decompressed original string

Other things in Huffman hpp/cpp

To simplify the process, I have given the full interface and implementation for a class called HuffmanNode. This class has all the basics for a tree node (leaf, branch, root, data members for linking, accessor) and also includes a comparator class for use with the heap. You should not need to alter any of the code for this node.

Examples

Below are some examples of how your code will run

```
HuffmanTree t;

string test = "It is time to unmask the computing community as
a Secret Society for the Creation and Preservation of
Artificial Complexity";

/*
1000101011110000101001100110000010111011001110111101111100011001001 01
```

```
1
0100100001111001110010011101101111010110010101010111110011000001110000
0
1011011110101100100010111110001100001111111111001011010011001011101001
1
1111101111001001110011111010011110111111000011100111111111101010111011
0
1001100111001001110110100110010011100101011000101100111100101001110000
0
0111010000000100111010100111001001000110010101100010110011110101110101
1
1110100010001000110001010110001111000001011001011101001101011001010101
1 0100101111010001110000111111111 */
string code = t.compress(test);
/*
LiLmLnBBLrLaBLtBBLPLdBLgLkBBLALIBLvLxBBBLhLlBLCLSBBBLsLpLfBBLoBBL
LeLcLuLyBBBBBB */
string tree = t.serializeTree();

/* It is time to unmask the computing community as a Secret
Society for the Creation and Preservation of Artificial Complexity
*/ string orig = t.decompress(code, tree);
```