Project 4: Huffman Encoding

Due: Mon, May 1

In this assignment, you will be creating Huffman Coding trees in order to compress strings. While normally the results of this encoding would be written in binary to a file, for this project you will simply output the string representation of the binary result. Using a priority queue, insert nodes containing each character and its respective frequency. While there are at least two nodes in the queue, remove them them and group them under a parent node containing the total of the nodes' frequencies. Enqueue the new node. When only one node remains, the tree is complete. Using 0 for left and 1 for right, determine the encoding for each letter used and output both the serialized tree and the encoded text.

Getting started

Clone the project stub into the `Projects` subdirectory of your SVN working directory.

```
cd ~/ID/Projects
svn export https://dev.cs.uakron.edu/svn/cs316sp17/shared/Projects/huffman/
```

Be sure to `svn add` the newly exported `huffman` directory and then commit.

Investigate the contents of that directory. You should have the following file:

- `test_huffman` — A small binary you can use to test the output of your project

You are responsible for creating all the files for this project.

Class requirements

The `Huffman` class must have the following:

1. A default constructor.
2. A destructor — memory leaks will cost you points
3. A `build_tree` function that takes a string and creates a huffman coding tree from it.
4. An `encode` function that takes a string and uses the tree created with `build_tree` to encode the message. This function returns a string representation of the encoded binary.
5. A `serialize` function that returns a string representing the huffman tree in a serialized manner.

Define the `Huffman` class and define all associated functions in `huffman.hpp`.

The `main` function should utilize the Huffman class and continually loop asking for input and displaying the encoded message and serialized tree. Loop until the string input is "q" and then end the program.

You must compile the program yourself. You can look at the CMakeLists.txt file from previous projects and modify it to this project or you can compile using g++ by itself. Make sure if you use g++ to use `-std=c++11` if it is needed.

Testing

You can test your project on knuth by running the test_huffman program. Input the tree serialization and the encoded output and the test program should return the original string.

```
$ ./huffman.out

Input String: MISSISSIPPI
10001110111101011010
*I//**M//P//S//

Input String: q

$ ./test_huffman
Tree: *I//**M//P//S//

Code: 10001110111101011010
processing:  * I / / * * M / / P / / S / / tree complete.
1 -> 0 -> 0 -> added M
0 -> added I
1 -> 1 -> added S
1 -> 1 -> added S
0 -> added I
1 -> 1 -> added S
1 -> 1 -> added S
0 -> added I
1 -> 0 -> 1 -> added P
1 -> 0 -> 1 -> added P
0 -> added I
MISSISSIPPI
```

Submission

1. Committing it to your SVN repository.

Grading basis

If your homework is not in subversion you will get a 0 on your assignment.

The total is out of 100 points.

- 100 If you write your own priority queue class.
- 80 If you use the standard library priority_queue class.
- You lose 10 points for having memory leaks. You can use valgrind to check for memory leaks on the knuth server.