CS515

Assignment 8P

The purpose of this assignment is to give you practice on how to encode a text file using a Huffman code and implement the algorithm using a priority queue.

Download all files from `~cs515/public/8P`

In the early 1980s, personal computers had hard disks that were no larger than 10MB. Now, the smallest of disks are still measured in hundreds of gigabytes. Even though hard drives are getting bigger, the files we want to store (funny pictures, videos, music and so on) seem to keep pace with that growth, which makes even today's gargantuan disks seem too small to hold everything.

One technique to use our storage more optimally is to compress the files. By taking advantage of redundancy or patterns, we are able to "abbreviate" the contents so they take up less space, but can be reconstructed into a full version of the original when needed. Such compression could be useful when trying to cram more things on a disk or to shorten the time needed to send a file over a network. In this assignment you will produce a Huffman code suitable for compressing a text file.

Part 1 (80%):

You will be given a table of frequencies for the occurrences of each character in the original text. You are asked to build a Huffman tree using a priority queue, and produce the Huffman code for all the characters. Since a priority queue is needed in the algorithm, you should use the template version you finished in Lab 6.

The starter files include the header **`HuffTree.h`** You need to provide the implementation in a source file named **`HuffTree.cpp`**. You may modify the header file by adding new methods but you should **not** make changes to or remove the existing declarations for methods and instance variables. You should **not** use any STL containers in your program.

A test driver `testHuff1.cpp` is given to test your program. Its output is as follows:

```
1
00
010
0110
0111
```

The Huffman algorithm guarantees that the code is optimal, even though the actual encoding of the character might not be unique. In this assignment, however, you need to make sure your encoding algorithm produces output that matches the sample output.

Part 2 (20%):

Write a program named **`HuffmanCoder.cpp`** that takes the name of an input text file as a command line argument, and outputs the rate of compression using Huffman encoding algorithm. The program does not actually produce the compressed file; it only calculates the rate of compression as

100% * (*total bits in original file – total bits after compression*) */ total bits in original file*

For simplicity, you only need to consider encoding **all 26 alphabetic characters**. Moreover, all upper case alphabetic letters are treated as lower case ones so they will be decoded the same. This way, you can just calculate the frequency of each letter (if 'a' and "A" appear total 50 times in the file, value 50 is used as the frequency for 'a'), then pass an array of size 26 to the buildTree() method of HuffTree. The array should contain zero-entries for letters that do not appear in the file—this will still cause codes to be generated even if the character does not appear in the text. Non-alphabetic characters should be considered as taking up 8 bits in both the original and compressed files.

The output of a sample run on input file `input1` is:

```
$ ./HuffmanCoder input1
original bits: 6400
bits after compression: 3991
compression rate: 37.64%
```

Note your output should match the above format exactly. (Hint: use `std::setprecision` to format the floating number output.)

Submission:

- Submit **`HuffTree.cpp`**, **`HuffTree.h`** for part 1, and **`HuffmanCoder.cpp`** for part 2, as well as your priority queue files **`PQueue.cpp`** and **`PQueue.h`**.
- You should also fill out the README file and submit it as well. Submit the following files to the appropriate assignment on Mimir:

  `HuffTree.cpp HuffTree.h HuffmanCoder.cpp PQueue.cpp PQueue.h README`

- Do not turn in executables or object code. Programs that produce compile time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer).
- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
/**    CS515 Assignment 8
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
Collaboration Declaration: assistance received from TA, PAC etc.
*/
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style.
- To successfully pass the provided sample tests is not an indication of a potential good grade; to fail one or more of these tests is an indication of a potential bad grade.
- You must test thoroughly your program with your own test data/cases to ensure all the requirements are fulfilled. We will use additional test data/cases other than the sample tests to grade your program.
- Here is a tentative grading scheme.

Huffman code test runs 1 through 4	18 each
Huffman Compression Rate tests 1 through 3	5 each
Valgrind Huffman code test run 4	8
Valgrind Huffman Compression Rate test 3	5
Use STL container	-80