CSE250 Assignment A4 – Huffman Coding

Due: 8/13/2019, 1:59PM

Last updated: 2019-08-01 23:24

Objectives

- Practice design and implementation of tree-based algorithms.
- Get some exposure to a priority queue.
- Implement a fun tool.

Introduction

Huffman coding is one of the most important and elegant techniques in information theory. For a given sequence of input symbols, and their counts, it builds a binary tree that can be used to generate the optimal binary encoding of each symbol. The basic algorithm to build a Huffman tree can be summarized as follows:

1. Create leaf node for every input symbol and its count.
2. Take two nodes with the lowest count.
3. Attach them to a new parent node that stores a placeholder symbol with the count equal to the sum of counts in children.
4. Repeat steps 2 and 3 until all nodes are connected.

Your task is to implement a function that will build a Huffman tree for a given input sequence of symbols, a function that will print code-words encoded in the tree, and function to release the memory occupied by the tree.

Instructions

1. Create directory `A4` where you will place your code.
2. Download `A4handout.tar` from piazza.
3. Untar handout, and move `a4.cpp`, `a4.hpp` and `symbol.hpp` to your `A4` directory. These files provide all functionality you will need.
4. Analyze `symbol.hpp`, which provides helper structures you will be using in your code.
5. In `a4.hpp` implement missing functions considering the following:
   (a) The `huffman_tree` function should return a pointer to a root node of the Huffman tree for a sequence of symbols in $[first, last)$.
   (b) Input symbols are represented via type `symbol` (inspect `symbol.hpp`).
   (c) You can safely assume that the input range is valid: it contains at least two symbols, and count of every symbol is greater than zero.
   (d) Tree consists of nodes of type `bnode<symbol>` (inspect `symbol.hpp`).

(e) For a given node in the resulting Huffman tree, the left child must store symbol that is "less than" the symbol in the right child as prescribed by the `operator<` for `symbol` (inspect `symbol.hpp`). The value of a symbol (i.e. specific character) stored in the node should be minimum between the values of its children.

(f) The `release_tree` function should release the entire memory used by tree `root`.

(g) The `print_dictionary` function should print to `os` all code-words encoded in the Huffman tree represented by `root`. Each code-word should be printed as shown below. Note that `std::ostream& os` defines reference that essentially behaves the same way as `std::cout`. So for example, this code `os « "A";` will have the same effect as `std::cout « "A";`.

(h) To test your code you can use two files provided in the handout: `dummy.txt` and `loremipsum.txt`. You can see correct answers for both files in `dummy.ans` and `loremipsum.ans`. Note that the order in which you print your tree is irrelevant, however, code-words must match exactly. So for example, if you run `./a4 dummy.txt` you should see (with lines possibly in a different order):

```
B 000
U 001
H 01
A 1
```

Note that symbol and code-word should be separated by a single space and code-word should be written as a string of `0` and `1`.

Submission

1. Remove your binary code and other unrelated files (e.g. your test files).
2. Create a tarball with your `A4` folder.
3. Follow to https://autograder.cse.buffalo.edu and submit `A4.tar` for grading.
4. You can make five submissions, and the last submission will be graded.
5. Any submission after the deadline will have 50% points deducted.

Grading

- 10pt: `a4.cpp` compiles with your `a4.hpp`, runs and has no memory leaks.
- 90pt: If you pass the initial test, there will be nine benchmark tests. You will get 10pt for each correctly completed test.
- If your code is **extremely** inefficient, for instance due to infinite loop, autograder will terminate your code and you will receive 0pt.

Remarks

- Make sure that you start working on the assignment early! If you wait to the very last moment, you may expect delays from autograder (overloaded by other students who like you waited with their submission).
- If you are having trouble, have a public discussion on piazza! Before posting your question, ask yourself: Am I giving enough information where someone who is not looking at my code can still help? Can you describe tests that you have already tested? Do not, of course, share your code.