CPSC 221: Project `proj2` due Monday, Nov. 14, 2016 at 9pm via `handin`

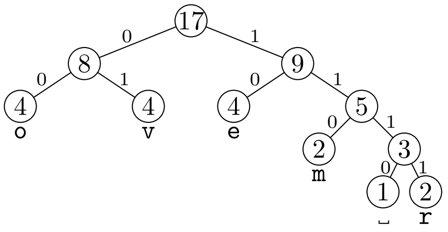
Out: October 26, 2016 Last Updated: October 26, 2016

Compressing Text

Huffman coding is a character-based compression technique invented by David Huffman in 1952. It encodes a string like `moveover moveover` using a sequence of bits, a *code*, for each character. ASCII (the American Standard Code for Information Interchange) is a character-based code but it assigns the same number of bits (seven) to each character. So `moveover moveover` is encoded in ASCII as 17 × 7 = 119 bits: (the spaces are not part of the encoding)

```
1101101 1101111 1110110 1100101 1101111 1110110 1100101 1110010 0100000 1101101 1101111 1110110 1100101 1101111 1110110 1100101 1110010
```

Huffman coding chooses the code for a character so that frequently occurring characters have shorter codes than rare characters. The resulting code also has the property that no character's code is the prefix of another's code (such codes are called *prefix codes*; ASCII is also a prefix code); that way a decompression algorithm can read the encoding bit-by-bit and know immediately when it has seen the code for a character. Huffman coding constructs an optimal character-based prefix code given a text to compress by counting the number of occurrences of each character in the text and then building a prefix code using these frequencies. For example, `moveover moveover` could be encoded using Huffman's technique as 42 bits: `110 00 01 10 00 01 10 1111 1110 110 00 01 10 00 01 10 1111` using the following *code tree*.



character	code
`o`	00
`v`	01
`e`	10
`m`	110
	1110
`r`	1111

A code tree is a binary tree where each character with a code appears at a leaf, every node has zero or two children, each left branch is labeled '0' and each right branch is labeled '1'. The code for a character is the sequence of edge labels from the root to the leaf for that character. The list of characters and their codes are shown beside the code tree in the example.

Huffman's Algorithm You might have noticed that the nodes in the code tree are labeled with integers. Each integer is the sum of the frequencies of the characters whose leaves are in the node's subtree. Huffman's algorithm builds the code tree by starting with a *priority queue* containing separate nodes, one for each distinct character in the input, whose priorities are the frequencies of their corresponding characters. It picks two nodes with the lowest frequencies (the nodes `r` and ␣ in our example) and joins them with a new common parent node whose frequency is the sum of the two frequencies (the node labeled 3). The parent node replaces its two children in the queue. The algorithm repeats this process until only one node, the root, remains.

Outline For this project, you need to hand in a program that will produce an optimal character-based prefix code using Huffman's algorithm for a given text file in both code-tree form and as a list of characters and their codes. For example, on typing `./huftree moveover.txt`, my program produces (using `std::cout`) the output shown to the right.

You must write your own priority queue implementation, **using a heap implemented with a resizeable array or a vector**, and code tree implementation. (I have them in separate files.) You may use the heap code from class and labs. Your code tree and code list should look the same as mine but characters with the same frequencies might be interchanged and codes might be different (as long as they have the same length).

```
.
|`1-.
|   |`1-.
|   |   |`1-.
|   |   |   |`1-"r"
|   |   |   |
|   |   |   `-0-" "
|   |   |
|   |   `-0-"m"
|   |
|   `-0-"e"
|
`-0-.
    |`1-"v"
    |
    `-0-"o"
"o":00
"v":01
"e":10
"m":110
" ":1110
"r":1111
```

I should be able to type "`./huftree text.txt`" at a Unix prompt to produce a Huffman code tree for the text in the file `text.txt` using your program.

Provided Code So that we all agree on how to print characters, I've included a `printChar( int ch )` function. (Characters are represented as integers to make our lives easier.) Use this function to print the characters in the code tree and in the list of codes. It prints a hexadecimal representation for characters outside of the "normal range".

```
void printChar( int ch ) {
  if( ch < 16 ) {
    std::cout << "x0" << std::hex << ch;
  } else if( ch < 32 || ch > 126 ) {
    std::cout << "x" << std::hex << ch;
  } else {
    std::cout << "\"" << (char)ch << "\"";
  }
}
```

I've also provided my `huftree.cc` file. It contains a loop that reads characters from a file and calculates their frequencies. Feel free to use it and modify it, but you should be certain to obtain the same sequence of characters from the input file as I do.

Deliverables

Using `handin proj2`, you should submit:

- Your source code (.cc and .h files).
- A `Makefile` so that typing `make` in your handin directory on an undergrad Unix server will produce an executable called `huftree`.

- A `README` file containing:
    1. Your name or, if a team submission, both your names.
    2. Approximately how long the project took you to complete.
    3. Acknowledgment of any assistance you received from anyone but your team members, the 221 staff, or the 221 textbooks, but please cite code quoted or adapted directly from the texts (per the course's Collaboration policy).
    4. A list of the files in your submission with a brief description of each file.
    5. Any special instructions for the marker.
- DO NOT HAND IN: .o files, executables, core dumps, irrelevant stuff.

How to `handin` this assignment

1. Create a directory called `~/cs221/proj2` (i.e., create directory `cs221` in your home directory, and then create a subdirectory within `cs221` called `proj2`).

2. Move or copy all of the files that you wish to hand in, to the `proj2` directory that you created in Step 1.

3. Before the deadline, hand in your directory electronically, as follows: `handin cs221 proj2`

   Note that you will receive a set of confirmation messages. If you don't get any kind of an acknowledgment, then something went wrong. Please re-read the instructions and try again.

4. You can overwrite an earlier submission by including the `-o` flag, and re-submitting, as follows: `handin -o cs221 proj2`

   You can hand in your files electronically as many times as you want, up to the deadline.

5. Additional instructions about `handin`, if you need them, are listed in the man pages (type: `man handin`). At any time, you can see what files you have already handed in (and their sizes) by typing the command: `handin -c cs221 proj2`

   If your files have zero bytes, then something went wrong and you should run the original `handin` command again.