YEAH!

Huffman Encoding

Brendon Go / 11.10.2015
Adapted from SL Rishi Bedi's Slides

Compression

A way to represent information using less data:

- “aaabbbccd” -> “3a3b2c1d”
- 9 letters -> 8 letters

Huffman Encoding:

- Characters that occur often should take up less space to store instead of everything being 8 bits

Huffman Encoding

- “aaaaabbbbz”
  - Uncompressed:
    - 01100001 01100001 01100001 01100001 01100001 01100010 01100010 01100010 01100010 01111010
  - Let a = 0 b = 10 z = 11
    - 00000101 0101011

How do I do Huffman Encoding?

- Count Frequencies
- Make Encoding Tree
- Build Encoding Map
- Encode Text Data

Step 1: Count Frequencies

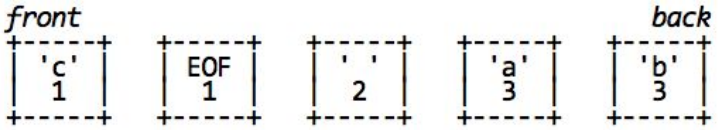
- Map<int, int> buildFrequencyTable(istream& input);
- example.txt: ab ab cab
- {‘ ’: 2, ‘a’:3, ‘b’:3, ‘c’:1, PSEUDO_EOF: 1}

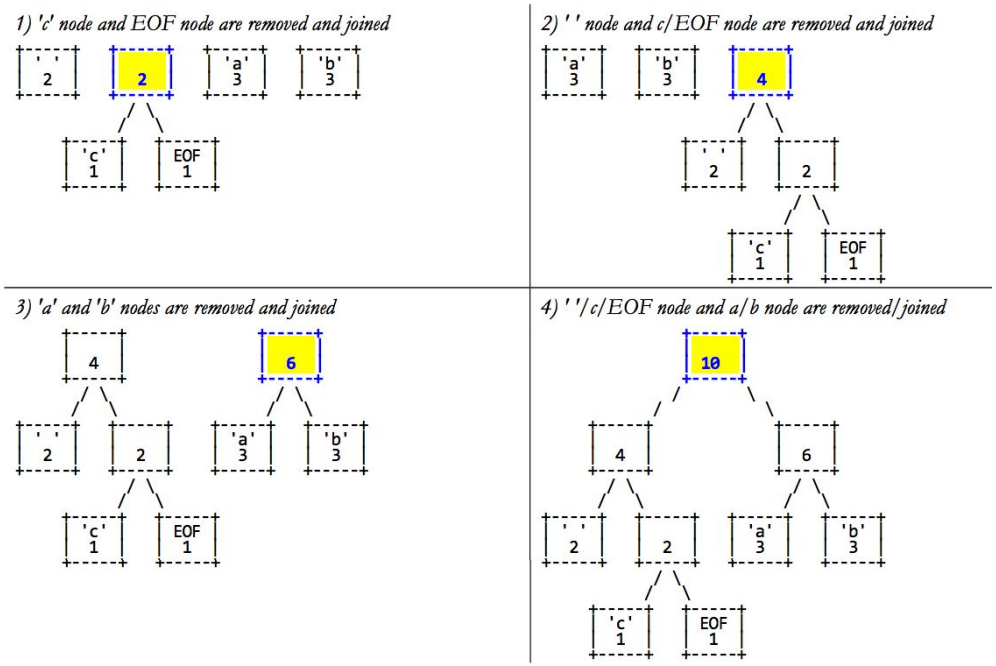
Relevant Code:

- PSEUDO_EOF
- int ch = input.get(); // reads single character. -1 if EOF

Step 2: Build an Encoding Tree

- Create PriorityQueue of HuffmanNode*'s with frequency as priority.
- While there's more than one thing in the PriorityQueue, dequeue two things
- Combine into one HuffmanNode* with Priority as sum of both things, and character NOT_A_CHAR





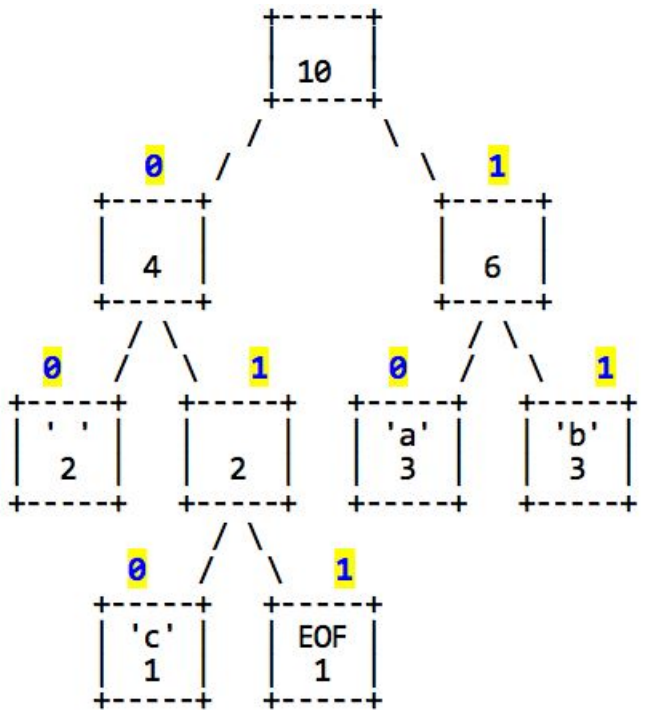
Step 2: Build an Encoding Tree

Relevant Code:

- #include “pqueue.h”
- pq.enqueue(node, priority), pq.dequeue(), pq.size()
- HuffmanNode
  - int character
  - int count
  - HuffmanNode* zero
  - HuffmanNode* one
  - isLeaf()
- NOT_A_CHAR

Step 3: Build an Encoding Map

- Map<int, string> buildEncodingMap(HuffmanNode* encodingTree);
- The code for each character is the path it took to get to the leaf.
- {' ':"00", 'a':"10", 'b':"11", 'c':"010", EOF:"011"}
- Note it's a map int:string



Step 4: Encode the Text

- {' ':"00", 'a':"10", 'b':"11", 'c':"010", EOF:"011"}
- "ab ab cab"
- -> 101100101100010101101100
- Prefix Property

Relevant Code:

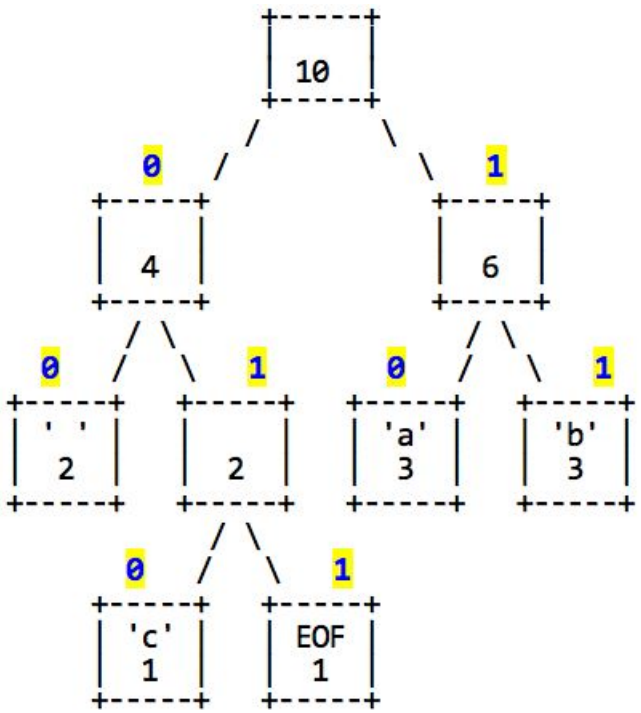
- obitstream
- output.writeBit(int bit) //0 or 1

Step 5: Decoding

- Read inputstream bit by bit.
- Go down the tree
- When you hit a leaf, you decoded that character. Repeat
- Stop when you decode PSEUDO_EOF
- not when you run out of bits to read
- Example: 10110010110001010110100

Relevant Code:

- ibitstream
- input.readBit() //reads 1 or 0 bit. -1 on EOF



Problem: We need the Tree to decode...

Solution: Put the Frequency Table in File

Relevant Code

- output << frequencyTable;
- input >> frequencyTable;
- Above handle printing and reading a frequency table
- rewindStream(input)

Compress:

Build Frequency Table, Build Tree, Build Map, Print Frequency Table to output, rewind, encode the file to output

Decompress:

Read the Frequency Table, Build Tree, Decode File

FreeTree: free memory used to make tree. Call when necessary