CSE143X Lecture Questions for Monday, 11/30/20

Time (e.g., 12:45)	Question	Answer
	Where is Unicode/ASCII included? Is it 'baked in' with the OS? Programming language? In this sense these are standards for reading bits right? Let's say I'm making a programming language from scratch, do I have to do work to make sure it's ASCII/unicode compatible, or does the standard include some sort of library that everyone uses.	You could think of it as being part of the operating system. It knows how to read text files. But individual programs like editors also need to know how to read ASCII files. Yes (how to interpret the bits you read). In general you would rely on libraries provided by the operating system.
	Assuming the Node of each tree has a field for the character, in those nodes that you create by combining the two lowest nodes, would it just not have a value?	The internal nodes created when you combine two subtrees don't need to keep track of a character.
	Are ASCII a set length? Always 8 bits? I'm quite confused as to how ASCII and huffman trees store the characters in bit: For example for the letter C in the example: In ASCII: it would a bunch of 0 & 1 taking up 8 bits In Huffman: it is just 01? So why then would the compressed file be longer? Oh I see, thanks!	ASCII is a 7-bit code with 128 characters. As I mentioned, the 8th bit is used for different things. Yes, the letter C in our example would have the code 01, which is just 2 bits long. If you give a command like this: `System.out.print(0);` Java will output the ASCII representation of the character "0". That will require 8 bits of storage. So using the example of the code 01 for C, writing it to a file as text will require 16 bits, which is very wasteful. That's why you want a compact binary file, not a normal text file.
	In general, we are supposed to remove debugging code before turning in our assignments. Would it be ok to leave HuffmanNode.toString()? (this method is not required by the spec, but is useful for reference/debugging)	You can comment out debugging code and leave it in your program as long as you comment that it's debugging code.

	For the Huffman tree, each character would start in different positions within a byte, so would that be inconvenient, like with only using 7 bits with the ASCII, or would it not matter? I see now, thank you!	It is inconvenient in the sense that you can no longer view the file as a simple text file. I showed this when I tried to display the contents of hamlet.short. It was incomprehensible. But compression is something you do to make a smaller copy. I think people understand that you need to undo the compression (unzip) to get back a readable file.
35:39	Can you clarify 4x as big? When you say big what are you referring to? Since the compressed/zip file does take up less bytes, I'm unsure what the 4x is referring to. Ohhh I see now, Thanks for the clarification.	I was saying that if you use simple commands like System.out.print for the 0s and 1s, then you will be producing output in text format (standard ASCII). In that format, each 0 and 1 will require 8 bits of storage. If your compression is giving you a 50% reduction, then the new file will require n * 8 * 0.5 amount of space, or 4n.
38:49	When there are 803 bits, I don't know why it needs to read 5 more bits to recognize a byte? I see.	My BitInputStream and BitOutputStream give you bits in multiples of 8. You can't get just 803 bits. You'd get 808 bits. So there are going to be 5 stray bits at the end that can cause a problem.
	^ I'm assuming that only when the compressed version will have a x number of bits that is not a multiple of 8 right? (Sorry I'm confused with how the bits work) What output or character does the pseudo eof character output when there's the extra bits? Does it just make a certain letter have extra 0/1's? Oh I see, thanks that clarifies things. Just to make sure the spec will clarify how the pseudo eof will be identified right? Thanks! Sorry so many qns.	I think what you said is right. Using the Huffman code, we're likely to get a sequence of bits that is not a multiple of 8, like 803. But my BitInputStream and BitOutputStream are limited in how they work. They can only give you multiples of 8. So once you get above 800, it goes to 808. That's a mismatch between the way the Huffman code works and the storage options available to you with the bit input stream classes. No. We'll be on the lookout for the pseudo-eof when we are decoding and we'll stop as soon as we see it. We won't write out anything, we'll just stop processing the file. Yes, the spec is very clear about the pseudo-eof.