Huffman Warmup (skip if you're feeling comfortable with Huffman coding). How many bits are in the Huffman encoding of the following message?

a b a a b a c a b a a b a c d a b a a b a c a b a a b a c d e

The frequencies for this message are given for the table below:

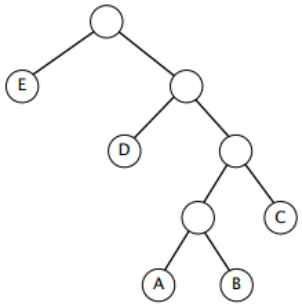
a	b	c	d	e
16	8	4	2	1

The first step is to draw the Huffman coding trie. Doing this we find that a is one bit, b is two bits, c is three bits, and d and e are four bits. This results in a length of 16+16+12+8+4=56 bits.

Huffman Best/Worst Case. Assuming the input is a sequence of 8-bit characters, give a best case input for Huffman coding. What is the compression ratio? What is the compression ratio for a worst case input?

Best case: All the same character. In this case, the compression ratio will be 1/8. In the worst case, we have an equal number of every key (out of 256 possible keys), resulting in a compression ratio of 1. We ignore the coding trie because for large N this trie is negligible.

Tougher Huffman Analysis. Consider the following Huffman trie of a message over the 5-character alphabet {A, B, C, D, E}. You may assume every character is used at least once.



For each statement below, determine which of the three conditions applies: true for all messages, false for all messages, or its veracity depends on the message.

1. The frequency of A is strictly less (i.e. not equal) than the frequency of B.
   Depends. All we know is that $F(A) \leq F(B)$.

2. The frequency of C is greater than or equal to the frequency of A.
   True. Since A and B were merged first, they must have the smallest frequencies.

3. The frequency of D is strictly greater than the frequency of A.
   True. If the frequency of D were equal to that of A, then it would also be equal to that of B and C. In that case, C and D would have merged.

4. The frequency of D is greater than or equal to that of A, B, and C combined.
   Depends. If A, B, and C have frequency 1, then D could have frequency 2 or 3 and produce the same subtree.

5. The frequency of E is strictly less than that of A, B, and C combined.
   False. If the frequency of E is strictly less than that of A, B, and C combined, then so is the frequency of D. Hence D and E would be merged.

LZW Encoding Warmup (skip if you feel comfortable with LZW basics). Use LZW to compress the string ABAAABABA. Assume 8 bit codewords and that the first new codeword starts at 81. A corresponds to 41, and B corresponds to 42.
41 42 41 83 82 82 80

LZW Decoding Warmup (skip if you feel comfortable with LZW basics). Use LZW to decompress 42 41 42 83 82 82 80. Assume 8 bit codewords and that the first new codeword starts at 81.
BABBBABAB

LZW. What is the best-case compression ratio for N characters using 12-bit codewords? Assume our alphabet is R=256 (8 bit characters). Recall that no new codewords are added to the table if it already has $2^{12} = 4096$ entries.
12/(8*3840). The best case is a single character repeated N times. The table contains 12-bit codewords for A, AA, AAA and so forth, all the way up to $2^{12} - 256 = 3840$ As when the table gets full. After this point, each sequence of 3840 As is encoded using only 12 bits.

What is the compression ratio in the worst case?
12/8. The worst case is when the table gets filled with useless codewords and then the rest of the message cannot use any additional codewords. For example, 3,840 As followed by a very long string of Bs would have this property. In this case, each B requires a 12 bit codeword.

Extra: Given an input bistream of length L and a policy where the most-recently-used-codeword is replaced, what is the best case order of growth of the compression ratio as a function of L?
The best case is a repeated sequence all of the same character. In this case, the Kth codeword added to the table represents K instances. The Kth codeword is added after processing the $1 + 2 + \cdots + K = \frac{K^2}{2}$th character of the input. For example, if L = 66, then we have codewords representing A, AA, AAA, AAAA, …, up until AAAAAAAAAAA (11 As). In this case, the order of growth of the compression ratio would be K^2/L. Since

Tries. The problem with tries is that they are very memory hungry. Suppose we wanted to address this by replacing the `next[]` array with an LLRB. What would be the worst-case order of growth for a search hit, assuming a query of length L, an alphabet of size R, and that there are N string keys in the Trie. Give your answer in tilde notation.

In the worst case, following each link would take time $\sim 2 \lg R$. This would result in $\sim 2 L \lg R$ time.

Prefix-Free Codes. In data compression, a set of binary strings is prefix free if not string is a prefix of another. For example {01,10,0010,1111} is prefix free, but {01,10,0010,10100} is not because 10 is a prefix of 10100. Design an efficient algorithm to determine if a set of binary strings is prefix-free. The running time of your algorithm should be proportional the number of bits in all of the binary stings.

Build a binary trie (R=2) with all the keys. If the key being inserted is a prefix of any other key, or if any other key is the prefix of the key being inserted, then we know that the code is not prefix free.

Extra problem: Autocomplete Enhanced. On assignment 3, we designed an autocomplete data structure that used binary search to return the list of words starting with a given prefix. For an array of length N and a number of matches M, our algorithm took M + log N time to return all matching strings. Suppose we want to return only the top Q matches. Design a Q + log N data structure.

Just steal the idea from the Fall 2013 assignment. Use a TST, but when inserting keys, at each node cache the maximum value contained in the current node's subtree. When searching, first find the node that matches the prefix (in L time, where L is the length of the query). Then search its subtrees in order of decreasing cached-subtree-value. Stop once you've collected Q keys.

Extra problem #2: Kolmogorov Complexity. Given a string S, the Java-Kolmogorov complexity $K_J(S)$ is defined by the number of bytes in the shortest Java program that prints S to the screen. Let $|S|$ be the length of the string.

a. Prove that $K_J(S) \leq |S| + c$ for some constant c for all strings.
   System.out.println(S) is a trivial way to print a string.
b. Give an example of an S such that $K_J(S)$ is much less than$|S|$.
   A long sequence of a million As (use a for loop)
c. Suppose we use LZW compression on S and get an output stream B. Explain how B and $K_J(S)$ are related.
   We can bound $K_J(S)$ using B, since we know that $K_J(S) \leq |Z| + |B|$ where Z is the length of the code needed to implement the LZW decoding algorithm.
d. Intuitively explain why $K_J(S) \geq |S|$ for most strings.
   We can intuitively think of $K_J(S)$ as being the compressibility of S. In class, we discussed a counting argument (to be covered again in precept) that most bitstreams cannot be compressed.
e. Let the Python-Kolmogorov complexity $K_P(S)$ be defined as the number of bytes in the shortest Java program that prints S to the screen. Prove that for all S, $K_J(S) \leq K_P(S) + c$ for some constant c. Why is this result interesting?
   We can simply write a python interpreter in Java, the length of which is c. This is neat because it means that generally speaking, the complexity of a bitstream is independent of the way that you try to cleverly represent it (i.e. there are some easy bitstreams and some hard ones).