Practice Assignment 3

Due: Tuesday, February 3 at 2PM to TEACH

To get credit, each student must submit their own solutions (which need not be typeset) to TEACH by the due date above – no late submissions are allowed. Note that while you must each submit solutions individually, you are free to work through the problem sets in groups. Solutions will be posted shortly after class and may be discussed in class. These will be not be graded on the basis of *completion* alone, not *correctness.*

In the following questions, you are given a graph G with n vertices and m edges and positive weights on the edges. The questions at the end of the MST lecture notes are good practice for the midterm (in particular, questions 1-3).

1. Give pseudocode for Prim's MST algorithm that makes it clear what the running time is using a priority queue data structure. What is the asymptotic running time in terms of n and m?
2. Give pseudocode for Borůvka's MST algorithm that makes it clear what the running time is using a *union find* data structure (as for Kruskal's algorithm) without contracting edges of G. What is the asymptotic running time in terms of n and m?
3. (a) Prove that if the weights of the edges of G are unique then there is a unique MST of G.

   (b) Note that the converse is not true. Give a counterexample.
4. Suppose we want to encode the following phrase "`bananas are tasty`" in binary There are 9 characters that need to be represented (including the space); since k-bit code-words can represent 2^k letters, we need 4-bit code-words. Assigning one of these 4-bit code words to each letter $\{a, b, e, n, r, s, t, y, '\ \ '\}$ allow us to encode the phrase in $17 \times 4 = 68$ bits. Having each code-word be the same length allows us to easily know the start and end positions of each letter.

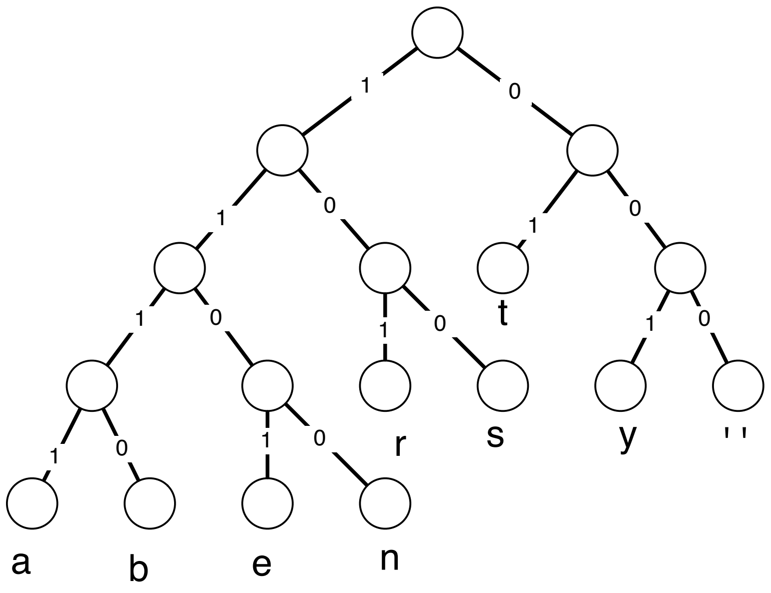
   Suppose we allowed for different length codewords such as:

a	1111
b	1110
e	1101
n	1100
r	101
s	100
t	01
y	001
' '	000

   The phrase encodes in 58 bits to: 1110111111001111110011111000001111101110100001111110001001

   Note that while it is not obvious where one code-word starts and another ends, we can still decode this phrase because *no code-word is the prefix of another code-word.* This is known as a prefix-free code.

   It seems quite cumbersome to decode until we notice that we can represent the code with a tree:



Develop another code that encodes "`bananas are tasty`" in fewer than 58 bits.