Practice Assignment 3

Due: Tuesday, October 27 at 10AM 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 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?

```
PRIM(G = (V, E), w)
    pick an arbitrary vertex s
    mark s
    initialize T = ∅
    initialize a priority queue Q
    add edges incident to s to Q with priority given by their weight
    while Q is not empty:
        pop uv from Q
        if u and v are not both marked
            mark the unmarked endpoint of uv
            add uv to T
            add the edges incident to uv's unmarked endpoint to Q with weight priority
    return T
```

This pseudocode makes it clear that each edge is added to the priority queue at most twice (once for each endpoint). The priority queue has at most m elements and there are $\Theta(m)$ priority queue operations for a total run time of $\Theta(m) \times O(\log m) = O(m \log m) = O(m \log n)$.

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?

There are multiple ways to do this, here is one:

```
BORUVKA(G = (V, E), w)
    disjoint sets D = {{v_1}, {v_2}, ..., {v_n}}
    initialize F = (V, ∅)
    while the number of sets in D > 1:
        for each set S in D:
            for each vertex v in S:
                for each edge uv incident to v:
                    if find(D, u) = find(D, v)
                        delete uv (useless)
                    else
                        replace uv as saved edge for S if cheaper than current saved edge for S
        for each saved edge uv
            union(find(D, u), find(D, v))
    return the set of edges that were ever saved
```

The number of phases (while loops) is $O(\log n)$ as argued in class. For each phase, there are $O(m)$ edges to test and for each, there is a union-find operation, taking $O(\log n)$ time. The total is $O(m \log^2 n)$. This is not as good as contracting edges.

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.

   (a) We assume that the weights of the edges of G are unique. For a contradiction, suppose there are two MSTs T_1 and T_2 that have different sets of edges. Let $F = T_1 \cap T_2$ be the set of edges that are common to T_1 and T_2. Consider the safe edge uv for a component C of F. Since the edge weights are unique, there is a unique edge uv with minimum weight with one endpoint in C. By construction of F, though, uv is not in one of T_1 or T_2. However, by the safe edge lemma, we know that uv is in the MST. So one of T_1 or T_2 cannot be an MST, contradicting our supposition that both T_1 and T_2 are MSTs.

   (b) The following graph has a unique MST but its edge weights are not unique: a graph that is a path with 2 edges, both of weight 1.

4. Consider the following game for an odd number of school children. Each child stands at a point such that *no two distances between children are equal*. Each child has a water gun. Someone yells "go" and every child (simultaneously) sprays the person who is nearest to them. Prove that at least one person stays dry. Do this by proving that the statement *there is a dry survivor in every odd water-gun fight with $2n-1$ children* is true by induction on n. In the inductive step, how can you build a solution for $n = k+1$ from $n = k$? *The base case is when $n = 1$ and there is just one child playing. Of course this child stays dry!*

   Assume the statement is true for $n = k$, or $2k-1$ children. Using this inductive hypothesis, we show that the statement is true for $n = k+1$ or $2k+1$ children.

   Let C be the set of $2k+1$ children. Let Alice and Bob be the two children in C that are closest to each other (that is, no other pair of children are closer). Certainly Alice and Bob both get wet because they fire on eachother. Consider the rest of the children C'. $|C'| = 2k-1$. By the inductive hypothesis, we know that if the children in C' play amongst themselves, there is a survivor, Sue. In the game where Alice and Bob are now playing, Sue is still a survivor: each child has remained in place and Alice and Bob are closer to each other than either of them are to Sue (because distances are unique). No child, in the game with the set C of children, will fire on Sue.

5. 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.

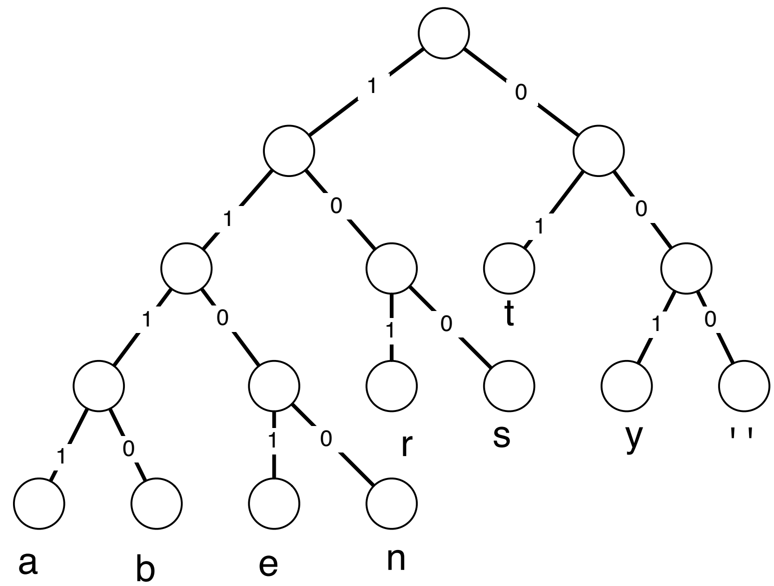
   Suppose we allowed for different length codewords such as:

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

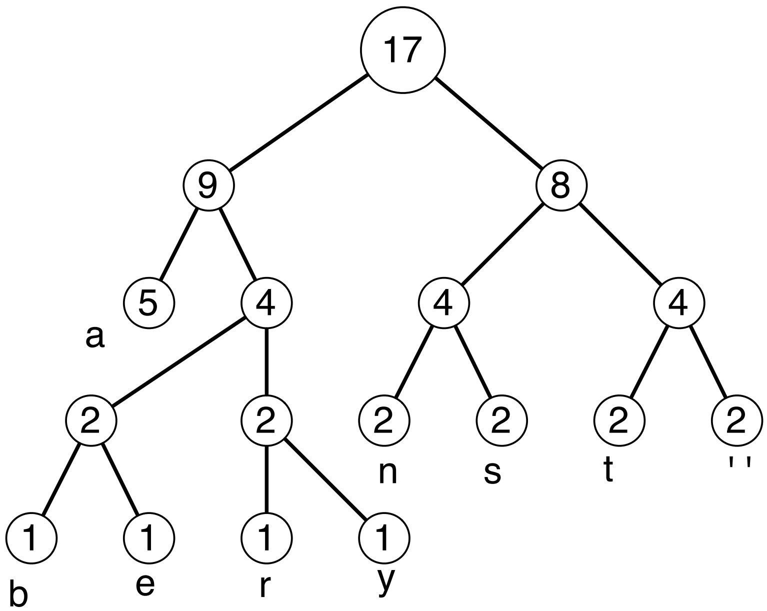
Note that while it is not obvious where one code-word starts and another ends, we can still decode this phrase because *no code-word is the prefix of another code-word.* This is known as a prefix-free code.

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. *Here is another encoding:*

a	*11*	*5*
b	*1011*	*1*
e	*1010*	*1*
n	*011*	*2*
r	*1001*	*1*
s	*010*	*2*
t	*001*	*2*
y	*1000*	*1*
' '	*000*	*2*



That encodes "bananas are tasty" in 50 bits:

1011 11 011 11 011 11 010 000 11 1001 1010 000 001 11 010 001 1000