Practice Assignment 3

Due: Tuesday, February 2 at 2PM to Canvas

To get credit, each student must submit their own solutions (which need not be typeset) to Canvas 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.

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 $T = \emptyset$
   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 $D = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$
   initialize $F = (V, \emptyset)$
   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. The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counterexample (if it isn't correct). Always assume that the graph $G = (V; E)$ is undirected. Do not assume that edge weights are distinct unless this is specifically stated.

   a. If graph G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.
   b. If G has a cycle with a unique heaviest edge e, then e cannot be part of any MST.
   c. Let e be any edge of minimum weight in G. Then e must be part of some MST.
   d. If the lightest edge in a graph is unique, then it must be part of every MST.

   a. False, consider the case where the heaviest edge is a bridge (is the only edge connecting two connected components of G.
   b. True. Assuming false, removing e from the MST will break the tree into two parts, and adding another edge across the two parts will re-connect the tree and we get a new tree with less total weight, a contradiction.
   c. True, e will belong to the MST produced by Kruskal.
   d. True. Assuming there is a MST that does not include this edge (e), adding e to the tree will cause a cycle, removing another edge in the cycle will produce a lighter tree.

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.

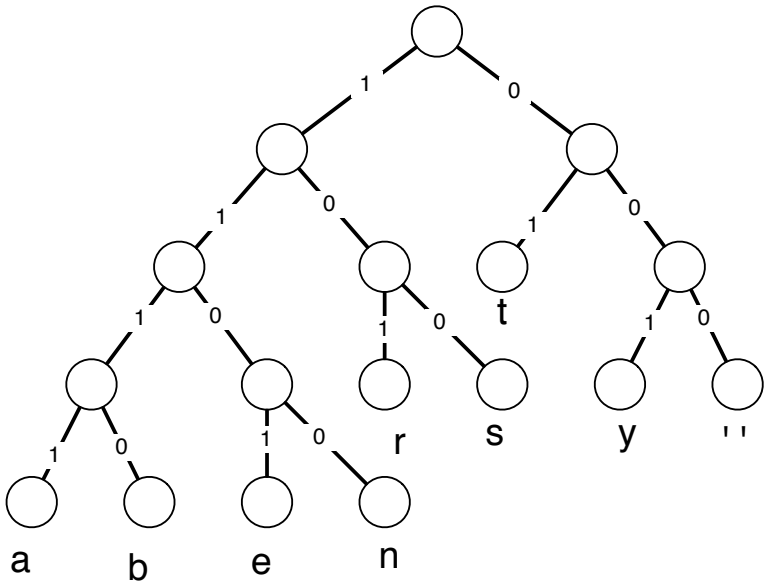
   Suppose we allowed for different length codewords such as:

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

The phrase encodes in 58 bits to: 11101111110011111100111110000011111011101000011111110001001

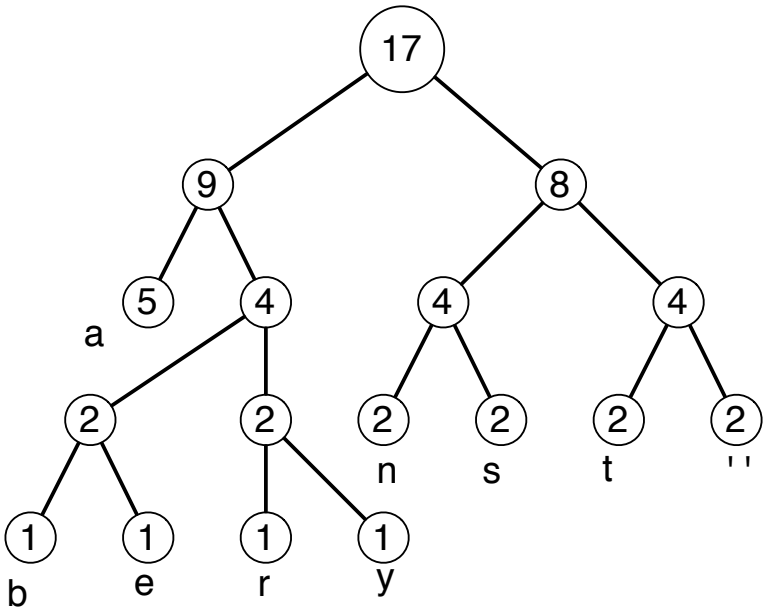
Note that while it is not obvious where one code-word starts and another ends, we can still decode this phrase because *no code-word is the prefix of another code-word*. This is known as a prefix-free code.

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