CS302 - Problem Set 9

1. Suppose you would like to use Huffman's Algorithm to encode 2^n different letters. Suppose you also know that the probability of the least frequent letter is at least half the probability of the most frequent letter. (Note: a perfect binary tree is a binary tree where all leaves have the same depth, so there are 2^k leaves for some $k \in \{0, 1, 2, \dots\}$).

   (a) The following example has the property that the probability of the least frequent letter is at least half the probability of the most frequent letter, and there are 2^2 letters. Please use Huffman's algorithm to create a tree for this input, and describe the structure of the tree.

   $$A : .18, \quad B : ..2, \quad C : .26, \quad D : .36 \tag{1}$$

   (b) In a future problem set (if we have time to learn the technique involved), you will prove that if the least frequent letter is at least half the probability of the most frequent letter, then the resulting tree is a perfect binary tree, which means all leaves have the same depth. For now, please explain why it makes sense that such a condition would result in such a tree.

2. Finish the proof of the Lemma from class:

   Lemma: Given an input Σ and probabilities p_i for each $i \in \Sigma$, let a and b be the letters with the smallest probabilities, there is an optimal tree (a tree with smallest average letter length) where a and b are siblings.

   Proof: Assume for contradiction that there is no optimal tree with a and b siblings. Let T be any optimal tree. By our assumption, in T, a and b are not siblings but must exist as leaves somewhere in T. Let x and y be sibling leafs in T at maximum depth.

   If we exchange the positions of x and a in T, and we exchange the positions of y and b in T, then we get a new tree T_{ex} with a and b siblings. We will show this new tree has the same average length as T, and thus is optimal, which contradicts our assumption that no optimal tree has a and b as siblings.

   To show that T_{ex} has the same average length as T, ...

3. Explain why any problem in NP (using our definition from class) can be solved in exponential (i.e. $O\left(2^{n^a}\right)$ time where n is the size of the input to the problem, and a is a positive constant.

4. Here are some problem definitions:

   - DOUBLE-SAT: Given a CNF formula involving the variables $x_1, x_2, \ldots, x_n$ and their negations, with at most m clauses, are there at least two different satisfying assignments? For example, $(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_3)$ has two valid assignments, $x_1 = 1, x_2 = 1, x_3 = 0$ and $x_1 = 0, x_2 = 1, x_3 = 0$.
   - k-INDSET: Given an undirected, unweighted graph $G = (V, E)$, is there a set $V' \subseteq V$ such that $|V'| \geq k$, and for all $v, u \in V'$, there is no edge $\{u, v\} \in E$?
   - k-CLIQUE: Given an undirected, unweighted graph $G = (V, E)$, is there a set $V' \subseteq V$ such that $|V'| \geq k$, and for all $v, u \in V'$, there is an edge $\{u, v\} \in E$?

   (a) Prove DOUBLE-SAT is in NP.

   (b) Prove k-INDSET is in NP.

   (c) Prove k-CLIQUE is in NP.

5. Consider the problem whose input is an unsorted array A of unique integers of length n (you may assume n is a power of 2), and which should output the *second* largest value in the array. (This is a challenging, "aha!" kind of problem. I will give a succession of hints providing more and more information, but you will get the most out of this if you use the next hint only when you are really stuck.)

   (a) Describe an algorithm that uses exactly $n + \log_2 n - 2$ comparison operations, where a comparison is an operation that tests whether one element is less than another element.

   (b) Justify why your algorithm is correct. (You can do this as a proof, or as a slightly less formal explanation.)

6. Approximately how long did you spend on this assignment (round to the nearest hour)?

Hints:
2. Think about the relationship between the difference in the depth of x between T and T_{ex} vs. the difference in the depth of a between T and T_{ex}. How does depth relate to length of encoding?

Also, key points are that a and b have minimum weight, and x and y have maximum depth.
3. Think brute force!

5. Hint 1: The solution uses techniques from both divide and conquer and dynamic programming, and your choice of data structure to store previous solutions is important.

Hint 2: The data structure you should use to store previous solutions is a binary tree of depth $\log_2 n$.

Hint 3: Start by storing all of the values of the array at the leaf nodes.

Hint 4: Fill in the rest of the array as if you are trying to find the maximum value in the array.

Hint 5: How can the array you create be used to find the next largest value?