1. *Money Changing.* Fix a set of positive integers called *denominations* $x_1, x_2, \ldots, x_n$ (think of them as the integers 1, 5, 10, and 25). The problem you want to solve for these denominations is the following: Given an integer A, express it as

$$A = \sum_{i=1}^{n} a_i x_i$$

for some nonnegative integers $a_1, \ldots, a_n \geq 0$.

   (a) Under which conditions on the denominations x_i are you able to do this for all integers $A > 0$?

   (b) Given any set of denominations x_i, not necessarily satisfying the conditions in the first part, describe the set of *sufficiently large* integers A that you can express as $A = \sum_{i=1}^{n} a_i x_i$ with nonnegative a_i. In other words you should prove a statement like "If A exceeds X, then we can write $A = \sum_{i=1}^{n} a_i x_i$ for $a_i \geq 0$ if and only if A has the following simple property...." You do not have to give X explicitly, but prove it exists.

   (c) Suppose that you want, given A, to find the nonnegative a_i's that satisfy $A = \sum_{i=1}^{n} a_i x_i$, and such that the sum of all a_i's is minimal —that is, you use the smallest possible number of coins. Define a *greedy algorithm* for this problem.

   (d) Show that the greedy algorithm finds the optimum a_i's in the case of the denominations 1, 5, 10, and 25, and for any amount A.

   (e) Give an example of a denomination where the greedy algorithm fails to find the optimum a_i's for some A. Do you know of an actual country where such a set of denominations exists?

   (f) How far from the optimum number of coins can the output of the greedy algorithm be, as a function of the denominations?

2. *Shannon's Theorem.* In the lecture notes we defined the set F of all possible files such that

   - each file contains m characters
   - there are c distinct characters C_1,...,C_c.
   - C_i appears exactly f_i times in each file, where $\sum_{i=1}^{c} f_i = m$.

   Let $|F|$ be the number of members (files) of the set F. In class we said that at least $\log_2 |F|$ bits are needed to encode F by *any* algorithm, just to distinguish among all possible different files. In this question you will devise an encoding that actually uses $\log_2 |F| + O(\log m)$ bits.

   In particular you should devise a function f_{enc}=Encode$(f, C_1, C_2, ..., C_c)$ that takes a file f meeting the above criteria with the list of characters, and returns f_{end}, an encoded file at most $\log_2 |F| + O(\log m)$ bits long. You should also device a function Decode(f_{enc}) that takes f_{enc} and returns f. The point of $O(\log m)$ is that you can include a header in f_{enc} to include useful information such as m and possibly other data. You may assume for simplicity that the character set is either fixed length or has the prefix (free) property to make it easy to write Encode. You may assume that c and the length of each C_i is $O(1)$. Do not worry about trying to make Encode and Decode very efficient.

3. (From the Spring 1998 Midterm). In this question we will consider how much Huffman coding can compress a file F of m characters taken from an alphabet of $n = 2^s$ characters $x_0, x_2, \ldots, x_{n-1}$.

   - How many bits does it take to store F without using Huffman coding?
   - Suppose $m = 1000$ and $n = 8$, with characters 0,1,2,3,4,5,6, and 7. Give an example of a file F (a string of 1000 digits from 0 through 7) in which every character x_i appears at least once, which compresses *the most* under Huffman coding. How many bits does it take to store the compressed file?
   - Let $f(x_i)$ denote the frequency of x_i, i.e. the number of times x_i appears in F. Prove that there exist frequencies $f(x_i) > 0$ such that the number of bits needed to store F without Huffman coding is $\Omega(\log n)$ times the number of bits to store F when it is Huffman encoded. You can assume that the length of the file m, is much larger than n. Be sure to exhibit the bit patterns representing each character, both with and without Huffman coding, as well as explicit formulas for each $f(x_i)$.

4. CLR 16-2

5. CLR 16-3