CSE 521
Assignment 6
Due Tuesday, May 13, 2003

1. There are a number of matrix multiplication algorithms that use less than $O(n^3)$ operations where the matrices are $n \times n$. The most famous is Strassen's algorithm that is based on being able to do 2×2 matrix multiplication in 7 multiplications instead of the usual 8 (see 28.2 of CLRS). Strassen's method and many of the others only require that the elements of the matrices be members of a *ring*. Rings have addition, subtraction (additive inverses), and multiplication. The operations are associative and the distributive law holds. The addition is commutative, but the multiplication does not have to be commutative. The integers with addition and multiplication, $(Z, +, \cdot)$, is ring where the multiplication is commutative. Even more $(Z_n, +_n, \cdot_n)$ is a commutative ring. The 2×2 matrices over the reals also form a ring that is noncommutative.

   Now consider Boolean matrix multiplication defined as follows. Let A and B be $n \times n$ matrices of Boolean values $\{0, 1\}$. The Boolean operators are $\vee$ (or) and $\wedge$ (and). Define $C = AB$ by

   $$C_{ij} = (A_{i1} \wedge B_{1j}) \vee (A_{i2} \wedge B_{2j}) \vee \cdots \vee (A_{in} \wedge B_{nj})$$

   for $1 \leq i, j \leq n$. Boolean matrix multiplication can be computed in $O(n^3)$ operations and $O(n^3)$ bit operations. Strassen's algorithm does not directly apply to Boolean matrix multiplication because the $\vee$ operator does not have inverses.

   (a) Show how to efficiently reduce Boolean matrix multiplication to a matrix multiplication problem in $(Z_m, +_m, \cdot_m)$ for some m. You'll want to choose m as small a possible to achieve your result.

   (b) Suppose matrix multiplication over a ring takes $O(n^\alpha)$ operations. Assume binary addition and subtraction take linear bit operations and binary multiplication takes quadratic bit operations, how many bit operations, as a function of n, does it take to do Boolean matrix multiplication using your reduction.

2. Consider polynomials over Z_2, defined as $Z_2[x]$. These polynomials have coefficients that are either 0 or 1. An example is $x^5 + x^3 + x + 1$. The $Z_2[x]$ polynomials can be added or multiplied in the usual way, except that the coefficients are always added and multiplied mod 2. As examples we have: $(x^5 + x^3 + x + 1) + (x^6 + x^5 + x^2 + 1) = x^6 + x^3 + x^2 + 1$ and $(x + 1)(x + 1) = x^2 + 1$. There is also a division algorithm, where if $B(x)$ and $D(x)$ are two polynomials in $Z_2[x]$ then there are polynomials $Q(x)$ and $R(x)$ such that $B(x) = Q(x)D(x) + R(x)$ and $\deg(R(x)) < \deg(D(x))$. There is a one-to-one correspondence between $Z_2[x]$ polynomial with degree-bound n and bit strings of length n given

   $$\sum_{i=0}^{n-1} b_i x^i \leftrightarrow b_{n-1} b_{n-2} \cdots b_0.$$

Error detecting codes called *cyclic redundancy codes* use this correspondence in the following way. Suppose we want to transmit a bit string $b_{n-1}b_{n-2}\cdots b_0$. Think of this as transmitting a $Z_2[x]$ degree-bound m polynomial $T(x) = \sum_{i=0}^{m-1} t_i x^i$. Think of an error in transmission as another $Z_2[x]$ degree-bound m polynomial $E(x) = \sum_{i=0}^{m-1} e_i x^i$. What is received is $T(x) + E(x) = \sum_{i=0}^{m-1}(t_i + e_i)x^i$, where addition is mod 2. For example, an isolated single bit error is represented by $E(x) = x^i$.

A cyclic redundancy code (CRC) is defined by a degree k polynomial $G(x) = \sum_{i=0}^{k} g_i x^i$ where $g_k = 1$. To apply this code we transmit $n + k$ bits, instead of the original n bits, computed in the following fashion. Let $B(x)$ be a degree-bound n polynomial representing the n bits to be transmitted and let $G(x)$ be the degree k polynomial of the code. Let $R(x)$ be the remainder of $x^k B(x)$ divided by $G(x)$. The remainder $R(x)$ is has degree-bound k so it can be represented in k bits. We transmit the original n bits followed by the k bits representing the remainder. In terms of $Z_2[x]$ polynomials we transmit $T(x) = x^k B(x) + R(x)$ which has degree bound $n + k$. Suppose $n + k$ bits, represented by $T'(x) = T(x) + E(x)$, are received. Divide $T'(x)$ by $G(x)$. If the remainder is not zero then an error is detected, otherwise accept the transmission (although there may be an undetected error).

As an example, consider $G(x) = x + 1$ and $B(x) = x^6 + x^5 + 1$ of degree-bound 8 representing the bit string 01100001. Dividing $x + 1$ into $x^7 + x^6 + x$ yields a remainder of 1. We transmit the 9 bit string 011000011, represented by the polynomial $x^7 + x^6 + x + 1$. Suppose the string 011100011, represented by $T'(x) = x^7 + x^6 + x^5 + x + 1$, is received. Dividing $T'(x)$ by $x + 1$ yields a remainder of 1. Thus, we have detected a one bit error. Standard 16 bit CRC's are $G(x) = x^{16} + x^{15} + x^2 + 1$ and $G(x) = x^{16} + x^{15} + x^5 + 1$

(a) Let $x^k B(x) = Q(x)G(x) + R(x)$ where $\deg(R(x)) < \deg(G(x)) = k$. Show that $G(x)$ is a factor of $x^k B(x) + R(x)$.

(b) Show that if $x + 1$ divides $G(x)$ then all errors with an odd number of bits are detected. These errors are represented by $E(x)$ with an odd number of non-zero terms.

(c) Show that if $G(x)$ does not divide $x^m + 1$ for all $m \leq n + k$ (n is the original number of bits to be transmitted and $k = \deg(G(x))$) and x does not divide $G(x)$, then all isolated two bit errors represented by a polynomial $E(x) = x^j + x^i$ with $0 \leq i < j < n$ and $j - i = m$ are detected.

(d) Show that if $G(x)$ has the constant term 1, then all burst errors of length $\leq k$ are detected where $\deg(G(x)) = k$. A burst error of length m is represented by $E(x) = x^i(x^{m-1} + \cdots + x + 1)$.

3. An alternative GCD algorithm that does not use division is based on the following:

   a. If a and b are even then $\gcd(a, b) = 2\gcd(a/2, b/2)$.

   b. If a is odd and b is even then $\gcd(a, b) = \gcd(a, b/2)$.

   c. If a and b are odd then $\gcd(a, b) = \gcd((a - b)/2, b)$.

   (a) Prove a, b, and c above.

   (b) Use these to design a recursive algorithm for the GCD.

   (c) Analyze the number of bit operations your algorithm takes assuming that testing for parity, addition/subtraction, and division by 2 take a linear number of bit operations.