CSE 200 Spring 2008
Calibration Homework Solutions

Chris Calabro and Russell Impagliazzo

April 9, 2008

1 Counting strings

Let X_n be the set of binary strings of length n with no two consecutive zeroes. Let Y_n be the subset of such strings whose first bit is one, and let Z_n be the subset whose first bit is zero. So $|X_n| = |Y_n| + |Z_n|$. I claim that $|Y_n| = |X_{n-1}|$ and $|Z_n| = |X_n - 2|$ for $n \geq 2$.

Consider the map that takes a member of Y_n and deletes the first bit. Since the remaining bits still do not have any two consecutive ones, the map has range X_{n-1}. Since appending a 1 to the beginning of the string gives you back the original member of Y_n, the map is invertible, hence 1-1. Appending a 1 to any element of X_{n-1} gives you an element of Y_n, since there are no two consecutive zeroes in the last $n-1$ bits by definition of X_{n-1} and the first bit is a 1 (so the first two bits are not consecutive zeroes). Thus, the map is onto as well, so we have a 1-1 onto correspondence between Y_n and X_{n-1} so $|Y_n| = |X_{n-1}|$.

Note that any element of Z_n has to start with a 0 (by definition), followed by a 1 (or else the string has two consecutive 0's). Consider the map that takes an element of Z_n to an $n-2$ bit string by deleting the first two bits. As before the range of the map is within X_{n-2} (since deleting bits cannot create two consecutive zeroes), and the map is 1-1 because the inverse is to append a 01 in front of the string. Finally, the map is onto because, if we append 01 in front of any string in X_{n-2}, we do not create two 0's in the first and second positions, nor in the second and third positions (which are the only consecutive pairs that change.) Thus, $|Z_n| = |X_{n-2}|$.

Therefore, $|X_n| = |X_{n-1}| + |X_{n-2}|$. This is the same recurrence as the Fibonnaci numbers defined by $Fib_0 = Fib_1 = 1$, and $Fib_n = Fib_{n-1} + Fib_{n-2}$ for $n \geq 2$. However, $|X_0| = 1 = Fib_1$ (the empty sting has no consecutive zeroes), and $|X_1| = 2 = Fib_2$ (both stings of length 1 cannot contain consecutive zeroes).

Then what follows by induction is that $|X_n| = Fib_{n+1}$. (We've confirmed this for the base cases, and assuming it true for all $0 \leq n' \leq n-1$, $|X_n| = |X_{n-1}| + |X_{n-2}| = Fib_n + Fib_{n-1} = Fib_{n+1}$.)

Fib_n is approximately $((1+\sqrt{5})/2)^n$, but this is not required for the solution.

2 closure properties

Assume L is a language closed under concatenation, i.e. that $L^2 \subseteq L$, and define $L+ = \cup_{k\geq 1} L^k$.

Claim 1. *$L^+ = L$.*

Proof. ($\subseteq$) We claim by induction on k that $\forall k \geq 1\ L^k \subseteq L$. If $k = 1$, then $L = L^1 \subseteq L+$, and so the base case holds. If $k \geq 2$, then by the induction hypothesis, $L^k = L \cdot L^{k-1} \subseteq L \cdot L \subseteq L$. So $L+ = \cup_{k\geq 1} L^k \subseteq L$.

($\supseteq$) This is the base case of the above claim, which was already shown. □

Claim 2. *$L^* = L$ iff $\lambda \in L$. (here $\lambda =$ empty string)*

Proof. ($\Rightarrow$) $\lambda \in L^* = L$.

($\Leftarrow$) $L^* = L^+ \cup \{\lambda\} = L \cup \{\lambda\} = L$, the 2nd equality coming from claim 1. □

3 matching degrees

Claim 3. *Every simple graph with at least 2 nodes contains 2 nodes of equal degree.*

Proof. Suppose indirectly that $G = (V, E)$ is a graph with $n \geq 2$ nodes that fails to contain 2 nodes of equal degree.

Define $f : V \to \{0, \ldots n-1\}$ by $f(x) = \deg_G(x)$ be the degree map. If f were not surjective (i.e. onto), then by the pigeonhole principle, some degree would be shared by 2 nodes, contradicting assumption. (the pigeons are the nodes and the holes are the degrees.) So the degrees of the nodes of G must be exactly the numbers $0, \ldots, n-1$. In particular, this implies that some node x has degree $n-1$ and some other node y has degree 0. (x, y are distinct since $n \geq 2$) But then x must be adjacent to y, a contradiction. □

4 Primes

a

Claim 4. *$\forall n \in \mathbb{Z}$, if $n \equiv 3 \pmod 4$, then n has a prime factor $p \equiv 3 \pmod 4$.*

Proof. We show the contrapositive. Suppose the prime factors of n are each congruent mod 4 to an element of $R = \{0, 1, 2\}$. Since R is closed under multiplication mod 4, it follows by induction on the size of the prime factorization of n that n also is congruent mod 4 to an element of R. (We are implicitly using that the modulus operation is a ring homomorphism: i.e. that the mod of a sum (product) is the sum (product) of the mods.) □

b

Claim 5. $|\{\textit{primes } p \mid p \equiv 3 \pmod 4\}| = \infty$.

Proof. Let p_i be the ith prime and suppose indirectly that p_k is the largest prime congruent to 3 mod 4. Let $n = p_1 \cdots p_k + 1$. Since the prime 2 occurs exactly once in the list $p_1, \ldots, p_k$, we have $n \equiv 3 \pmod 4$. ¿From part (a), $\exists$ prime factor p of n with $p \equiv 3 \pmod 4$. Since none of $p_1, \ldots, p_k$ divide n, we have $p > p_k$, a contradiction. □

5 quine

A *quine* is a program that takes no input and produces its own code as output. The name comes from the philosopher W.V.Quine who crafted the phrase

> 'yields falsehood when preceded by its quotation' yields falsehood when preceded by its quotation,

an assertion in english without a truth value and yet which is not explicitly self referential, thus showing that the problem with the liar paradox ('this sentence is false') is not the allowance of explicit self reference. (We may see more of this idea in the recursion theorem later.)

A simple quine in C, adapted from code on the quine page (`http://www.nyx.net/~gthompso/quine.htm`) follows.

```
#include <stdio.h>
char*f="#include <stdio.h>%cchar*f=%c%s%c;%cmain(){printf(f,10,34,f,34,10,10);}%c";
main(){printf(f,10,34,f,34,10,10);}
```

To see why quines are possible in nearly any programming language (and to see how to generate them somewhat automatically, albeit inelegantly), consider a programming language in which it is possible to construct a program A that takes as input a program B and produces a program C that runs the program B on B. Then A with A hardcoded as input is a quine. I.e. A has the property that for any B

$$\text{run hardcode}(A, B) = \text{hardcode}(B, B),$$

and so

$$\text{run hardcode}(A, A) = \text{hardcode}(A, A).$$