CIS 677 - Complexity, Randomness, and Algebra in Counting and Cryptography

Lectures by Prof. Sampath Kannan
Notes by Zach Schutzman

University of Pennsylvania, Spring 2019

Introduction

CIS 677 is a doctoral-level seminar in the theory and applications of Complexity, Randomness, and Algebra in Counting and Cryptography.

These notes are being live-TeXed, though I edit for typos and add diagrams requiring the TikZ package separately. I am using the editor TeXstudio. The template for these notes was created by Zev Chonoles and is made available (and being used here) under a Creative Commons License.

I am responsible for all faults in this document, mathematical or otherwise; any merits of the material here should be credited to the lecturer, not to me.

Please email any corrections or suggestions to ianzach+notes@seas.upenn.edu.

Lecture 1: Algebra Basics and Some Applications (2019-01-23)

1 Groups

What is a group?

A group is a set G together with an operation $\cdot : G \times G \to G$ such that:

- There is an identity element e such that $eg = ge = g$ for all $g \in G$
- $\cdot$ is associative. That is, $g(hk) = (gh)k$ for all $g, h, k \in G$
- For any $g \in G$, there exists a unique g^{-1} such that $gg^{-1} = g^{-1}g = e$

Definition. A group is called **abelian** or **commutative** if $gh = gh$ for all $g, h \in G$.

Definition. A group G is **finite** if $|G|$ is finite. It is **infinite** otherwise.

Some examples of groups:

1. The integers $\mathbb{Z}$ with the operation $+$ is an abelian group
2. The integers modulo n ($\mathbb{Z}_n$) is a finite abelian group under addition modulo n
3. $\mathbb{Z}_p^*$, the elements of $\mathbb{Z}_n$ which are relatively prime to p is a finite abelian group under multiplication modulo p. If p is prime, its elements are $\{1, 2, \ldots, p-1\}$.
4. The set of permutations of n letters is the *symmetric group* S_n under the operation of composition. It has $n!$ elements and is non-abelian (unless $n = 1$ or $n = 2$).
5. Sets of square invertible matrices (of a fixed size) under matrix multiplication are non-abelian, sometimes infinite groups.
6. The group of symmetries of a regular polygon with n vertices is the dihedral group D_{2n} and it is non-abelian (it's the semidirect product of $\mathbb{Z}_2$ with $\mathbb{Z}_n$)

Definition. A **subgroup** H of a group G is a set of elements $H \subset G$ which themselves form a group with respect to the operation of G. The identity element $e \in G$ is the identity in H. We denote this as $H < G$

We can prove our first theorem about groups!

Theorem (Lagrange's Theorem)**.** *If G is a finite group and $H < G$, then $|H|$ divides $|G|$.*

Proof. We begin by constructing an equivalence relation $\sim$ on G where $g \sim h$ if $gh^{-1} \in H$. We can easily verify this is a proper equivalence relation. It is reflexive because $gg^{-1} = e \in H$. It is symmetric because $gh^{-1} \in H$ implies $(gh^{-1})^{-1} \in H$, but this is equal to hg^{-1}. Transitivity follows from the cancellation property. If $gh^{-1} \in H$ and $hk^{-1} \in H$, then their product is also in H. The product is $gh^{-1}hk^{-1} = gk^{-1}$, which shows transitivity.

This equivalence relation partitions the group into disjoint subsets (called cosets). The set of things equivalent to a (written $[a]$) is exactly the set Ha, which is the set of things which are the product of something in H with a. Then, we can see that $|Ha| = |H|$, since for distinct elements $h_1, h_2 \in H$, $h_1 a \neq h_2 a$, meaning that each element in Ha can be written as the product of a unique element in H with a. Thus, $|G| = |H|$number of equivalence classes and $|H|$ divides $|G|$. This number of equivalence classes is called the *index of H in G*. □

This has a corollary:

Corollary. *If $H \subsetneq G$, $|H|$ is at most half of $|G|$, since $|H|$ is a proper factor of $|G|$ (or 1...).*

We can use this to do a randomized thingy!

Example. Freivald's Matrix Multiplication Checker is an algorithm which does the following. Suppose, for the sake of ease of exposition, everything here is done over $\mathbb{F}_2$. It takes in 3 $n \times n$ matrices A, B, C such that we are trying to verify $AB \stackrel{?}{=} C$. We can do this efficiently by taking a uniformly random binary vector v and checking that $ABv = Cv$. If the multiplication was correct, the test will always pass. If there is an error in C, what is the probability we detect it? Call v a *witness* if $ABv \neq Cv$ and a *non-witness* otherwise. We can observe that the set of non-witnesses forms a group under componentwise addition (mod 2), since for two non-witnesses v_1, v_2, then $(AB-C)v_1 = (AB-C)v_2 = 0$, so $(AB-C)(v_1+v_2) = 0$ (closure), and the inverse of a non-witness is a non-witness since everything here is self-inverse.[1] Since this is a subgroup of the group of all binary vectors, the set of non-witnesses has set at most half of 2^n as long as a witness exists. This means that if we pick a random v, with probability at least .5 we can detect an error.

To see that a witness must exist, consider again $AB - C$, which is non-zero if $AB \neq C$. Let ij be a non-zero entry in $AB - C$ and let v be a non-witness. Then $(AB-C)v = 0$ and the ith position of $(AB-C)v$ is the ith row of $AB - C$ times v. Therefore, the vector which is all zeroes except a 1 in the jth position constitutes a witness.

The test is efficient because we can compute $A(Bv)$ in n^2 time rather than $(AB)v$ which takes more.

The idea of this, and many other applications, is that the set of bad examples forms a group and a proper subgroup of the set of all examples, so there aren't too many of them.

Definition. A mapping $\varphi : G \to H$ from a group G to a group H is a **homomorphism** if $\varphi(xy) = \varphi(x)\varphi(y)$ for all $x, y \in G$, where the operation on the left is with respect to G and on the right with respect to H.

In particular $\varphi(e_G) = e_H$ and $\varphi(x^{-1}) = \varphi(x)^{-1}$.

Definition. The **kernel** of a homomorphism φ is the set $K \subset G$ such that $\varphi(k) = e$ for all $k \in K$.

Example. Take $\varphi : \mathbb{Z}_2 \to Z_7$ where $x \mapsto x \mod 7$. The kernel is the set of all multiples of 7 in $\mathbb{Z}$, i.e. $7\mathbb{Z}$.

Lemma. *The kernel of a homomorphism K forms a subgroup of G. By definition, $e \in K$, and if $x, y \in K$, then $\varphi(x) = \varphi(y) = e$ and so $\varphi(ab) = \varphi(a)\varphi(b) = e$.*

Definition. Let G be an abelian group and K a subgroup. Then the quotient G/K is a group whose elements are in correspondence with the equivalence classes (cosets) of K as defined in the proof of Lagrange's theorem.

Example. Take $G = \mathbb{Z}$ and $K = 7\mathbb{Z}$. Then the elements of G/K are equivalence classes of $[0], [1], \ldots, [6]$ which is isomorphic to $\mathbb{Z}_7$, where we mapped each element to its remainder modulo 7.

Theorem (First Isomorphism Theorem)**.** *If φ is a surjective homomorphism from G to H with kernel K, then $G/K \cong H$.*

We can apply this in proving the Chinese Remainder Theorem:

[1] This is true in general, too.

Theorem (Chinese Remainder Theorem)**.** *Let m_1, m_2 be two coprime numbers. Then $\mathbb{Z}_{m_1 m_2} \cong \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2}$.*

Proof. First, it's clear that $\mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2}$ is a group, where everything just happens componentwise. We'll prove this by showing that there is a homomorphism $\varphi : \mathbb{Z}_{m_1 m_2} \to \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2}$ with trivial kernel. We check that $x \mapsto (x \mod m_1, x \mod m_2)$ suffices. It's a homomorphism by checking it across the factors, and $0 \mapsto (0,0)$ is the entirety of the kernel. □

The Chinese Remainder Theorem is a special case of the following statement:

Lemma. *Let $\varphi : G \to H$ be a surjective homomorphism with trivial kernel. Then $G \cong H$.*

Here's an application to communication complexity:

Example (The Man on the Moon Problem)**.** Let Alice be on the Moon and Bob on Earth. Alice has a string x of length n and Bob has a copy y, and they want to make sure that they have the same string. We want a communication protocol to check $x \stackrel{?}{=} y$. We can assume without loss that y is also of length n, since checking this only requires $\log n$ bits of communication. We'll assume as well that the communication channel is reliable.

We can have a dumb algorithm where Alice sends all of x to Bob, which requires n bits of communication. If the potential corruption is adversarial, then this is the best deterministic procedure. To see this, observe that Alice and Bob are trying to compute a function together which can be viewed as mapping $X \times Y \to \{0,1\}$, where $|X| = |Y| = 2^n$. We can write this down in a table which looks like the $2^n \times 2^n$ identity. Suppose further, without loss, that the protocol is alternating. That is, the transcript of the communication will have things sent by Alice in odd positions and Bob in even positions.

If for any two pairs (x^i, y^i) and (x^j, y^j) we demand that the transcripts differ, in order for the 2^n matching pairs to have different transcripts, we need to communicate n bits. To see this, suppose that (x^i, y^i) and (x^j, y^j) do have the same transcript, but represent different pairs of matching strings. We'll show that this implies that the transcripts of (x^i, y^j) and (x^i, y^j) are identical, leading Alice and Bob to reach the incorrect decision. We can prove this by induction. Alice has x^i and sends the first bit accordingly. Bob has y^j and sees this first bit, which is consistent with the case where he has y^j and Alice has x^j, so he continues accordingly. Alice proceeds, given that everything she sees is consistent with Bob having y^i.

We can give a randomized algorithm which does better. A witness here will be a prime p such that $x \mod p \neq y \mod p$. First, note that if two primes p_1 and p_2 both divide a number $x - y$, then $p_1 p_2$ does as well. Since the product of n distinct primes is at least 2^n, since x, y, and $x - y$ are all at most 2^n, then there are at most n primes which divide $x - y$ (i.e. are non-witnesses). Alice can write down the first $2n$ primes and picks one of them p at random. She can then send Bob p $x \mod p$. Since there are at most n non-witnesses, the probability that p is a non-witness is at most .5. It takes $\log n$ bits to communicate p and another $\log n$ bits to send $x \mod p$.

Definition. Let G be a group and $g, h, \cdots \in G$. We denote by $\langle g, h, \dots \rangle$ the **group generated by** $g, h, \dots$. That is, the set of all elements of G which can be written as a finite product of g. Equivalently, $\langle g, h, \dots \rangle$ is the smallest subset of G which is closed under the operation and contains the **generators** $g, h, \dots$. We sometimes replace the elements with a set $S \subset G$ and write $\langle S \rangle$.

Definition. Thee **order** of an element g is the size of the group generated by g. Equivalently, it is the smallest number k such that $g^k = e$.

Example. In $\mathbb{Z}_7^*$, the order of 2 is 3, since $2 \times 2 = 4$, $4 \times 2 = 2^3 = 1$.

Definition. A group is **cyclic** if it can be generated by a single element.

Lemma. *If G is a group with $|G| = n$, then G can be generated by at most $\log n$ generators.*

Proof. We proceed inductively. Let $|G| \geq 2$ and choose $a_1 \neq e$ as the first generator. We look at $|\langle a_1 \rangle| \geq 2$. Choose some $a_2 \notin \langle a_1 \rangle$. This is a strict supergroup of $\langle a_1 \rangle$, so $|\langle a_1, a_2 \rangle| \geq 2|\langle a_1 \rangle|$. Since adding each generator at least doubles the size of the resultant group, we only need $\log n$ generators to reach order n. □

Example. The *group isomorphism problem* asks (algorithmically) whether two group G and H are isomorphic. One way to present a group to a computer is to list out the n^2 entries in the multiplication table of a group, so we can phrase the problem as asking whether the table for G is a permutation of the table of H.

There is an $O(n^{\log n})$ algorithm for this which works by finding a set of $\log n$ generators for G and match against every set of the same size in H to check if there is an isomorphic mapping among the generators.

A consequence of this and Lagrange's theorem is Fermat's Little Theorem.

Theorem (Fermat's Little Theorem)**.** *If p is a prime then $a^{p-1} \cong 1 \mod p$ for all a relatively prime to p.*

Proof. Consider $\mathbb{Z}_p^*$, which has size $p - 1$. Then the order of $\langle a \rangle$ is ℓ, which divides $p - 1$. Thus $a^{p-1} = (a^\ell)^q = e$. □

Example. This was proposed as a (wrong) primality test. To test if n is prime, pick a at random from $1, 2, \ldots, n$ and compute $a^{n-1} \mod n$. If this is equal to 1, say that n is prime. Otherwise, say that it isn't. This is good because it will always be 1 if n is prime. This algorithm actually works on most inputs, but not all. The ones that fail are called the *Carmichael numbers*. These are composite numbers which satisfy Fermat's Little Theorem.

Example. Given $a \in \mathbb{Z}_p^*$, how can we find a^{-1}. We know that a is relatively prime to p, so we know that $\gcd(a, p) = 1$. To compute a GCD, we can use the *extended Euclid's algorithm*, which takes in an x and y and produces the GCD d as well as the two integers q_1 and q_2 such that $xq_1 + yq_2 = d$. Using this, we can get the q_1, q_2 such that $aq_1 + pq_2 = 1$. Since modulo p, $pq_1 = 0$, $q_1 = a^{-1}$.

Lecture 2: ((I missed this)) (2019-01-30)

Lecture 3: tdb (2019-02-06)

2 Square Roots (modulo p)

Consider a prime $p = 3 \mod 4$, and the equation $x^2 - a = 0 \mod p$, i.e. a is a *square root* of p.

By Fermat's Little Theorem, $a^{p-1} = 1$, so $a^p = a$, and $a^{p+1} = a^2$, so $a^{\frac{p+1}{2}} = \pm a$. We know it's $+a$ because $-a = -1 \times a$, and -1 is not a quadratic residue in $\mathbb{Z}_p^*$ when $p = 3 \mod 4$ because $(x)^{\frac{p-1}{2}}$ will be 1 if and only if x is a quadratic residue (i.e. is an even power of the generator g). This is not the case for -1 because $\frac{p-1}{2}$ is odd since $p - 1 = 2 \mod 4$. Therefore, $-a$ is not a quadratic residue. Therefore $a^{\frac{p+1}{4}}$ is a square root of a.

Furthermore, the set of quadratic residues in $\mathbb{Z}_p^*$ when $p = 3 \mod 4$ forms a group Q_p, since for quadratic residues a, b, since the product of the square roots of a and b is the square root of ab, and demonstrating closure suffices to prove that this is a group, since it's finite.

However, if $p = 1 \mod 4$, then we don't have a good formula for finding square roots. Take such a p and consider $\mathbb{Z}_p^*$ and an element a. We have $a^{p-1} = 1$ so $a^{\frac{p-1}{2}} = 1$ and $a^{p+1} = a^2$, but $p+1$ is not divisible by 4.

Let's write $p = 2^s \times m$ where m is odd (this is easy). We want to solve $x^2 - a = 0$ and we know $a^{\frac{p-1}{2}} = 1$. So we can write $a^{m(2^{s-1})} = 1$. Let $a^m = u_0$. We know that the order of u_0 divides 2^{s-1} and is therefore a power of 2. If the order is 1, i.e. $u_0 = 1$, then $a^{m+1} = u_0 a = a$. Since $m + 1$ is even, $a^{\frac{m+1}{2}}$ is a square root of a. Otherwise, $a^{\frac{m+1}{2}}$ is a square root of $u_0 a$, which we will call v_0.

Construct two sequences, $u_0, u_1, \ldots$ and $v_0, v_1, \ldots$, such that $v_i^2 = u_i a$. The u_i will have orders which are powers of 2 and decreasing, so we eventually get a $u_j = 1$, which will imply v_j is a square root of a.

We'll need some randomness to construct the u_i, since we'll need to find an element of $\mathbb{Z}_p^*$ which is not a residue. We can do this by randomly testing, since half of the elements are non-residues. Let b be a randomly chosen non-residue in $\mathbb{Z}_p^*$, and let $c = b^m$. By Fermat's Little Theorem, $b^{p-1} = b^{m2^s} = 1$. Furthermore, $c^{2^{s-1}} = b^{\frac{p-1}{2}} = -1$ because b is a non-residue.

Let r_i denote the order of u_i. Then $u_0^{2^{r_0}} = 1$ and $u_0^{2^{r_0-1}} = -1$, since it's the square root and not equal to 1. Given this, we'll construct u_1 by taking the product $(c^{2^{s-1}})(u_0^{2^{r_0-1}}) = (u_0 c^{2^{s-r_0}})^{2^{r_0-1}} = 1$ and we'll let the base $(u_0 c^{2^{s-r_0}}) = u_1$. Now, we want a v_1 such that $v_1^2 = u_1 a$.

We have $v_0^2 = u_0 a$, so let's try $v_1 = v_0 c^{2^{s-r_0-1}}$, and so

$$v_1^2 = v_0^2 c^{2^{s-r_0}} = u_0 c^{2^{s-r_0}} a = u_1 a$$

which works. We repeat this process until we find a u_j of order 1.

Definition. The **Legendre symbol** of a in $\mathbb{Z}_p^*$ is written as $\left(\frac{a}{p}\right)$ and is equal to 1 if $a \in Q_p$ (i.e. is a quadratic residue) and -1 otherwise, and 0 if $a = 0$.

What if we have $N = pq$ for primes p and q. Given $a \in \mathbb{Z}_N^*$, can we find a square root of a? If a is a quadratic residue, then a has exactly four square roots, by the Chinese Remainder Theorem, since we can write it as a pair which is a square modulo p and a square modulo q, so if (b_1, b_2) is a square root of a, then the four square roots are given by $(\pm b_1, \pm b_2)$.

It turns out that finding a square root in $\mathbb{Z}_N^*$ is equivalent to factoring in the ring, given that a randomized polynomial time algorithm for finding square roots can be used to factor. [[see exercise 6.1 from Michael's CIS625 problem set]]

This gives us a candidate *one-way function*. Take $N = pq$ and $f(x) = x^2 \mod n$ for $x \in \mathbb{Z}_n^*$. This gives an encryption scheme for messages over $\mathbb{Z}_n^*$. Alice can encode x with $f(x)$. Bob can decode it using the factors of N using the Chinese Remainder Theorem and finds the square roots with respect to p and q (quickly). This is actually a *trapdoor function* since it is easy to compute the forward direction and easy to compute the backward direction only if you know a certain secret.

As a cryptosystem, this is called the *Rabin* cryptosystem. It's relatively simple, but not widely used. One drawback is that it gives four possible decryptions (assuming N is the product of two primes). We can restrict the message space to numbers less than $N/2$.

Another cryptosystem, which is very well-known and used a lot in practice is the *Rivest-Shamir-Adelman* (RSA) cryptosystem. In this system, Bob picks $N = pq$ and an encryption exponent e. He then publishes N and e. The secret will be knowledge of p and q as well as a decryption exponent d. When Alice sends a message x, she sends $f(x) = m^e \mod N$ (observe that the Rabin cryptosystem has $e = 2$). Bob needs to choose d such that $m^{ed} = m$, so $m^{ed-1} = 1$, so the order of m divides $ed - 1$. It also divides $(p-1)(q-1) = \phi(N)$ by Lagrange's theorem. Then $m^{\phi(N)} = 1$, so we might want to pick d such that $ed - 1 = k\phi(N)$. We know the GCD of e and $\phi(N)$ must be 1, so Bob should pick e according to that. He should pick d by using the extended GCD on $e, \phi(N)$ which gives an $xe + y\phi(N) = 1$, and let $d = x$. This will guarantee the uniqueness of the decryption.

Interestingly, there is no result equating the difficulty of RSA decryption to the difficulty of factoring. It may be easier, though it is certainly not harder. If Eve can factor N, she can use the same process that Bob used to pick d and use the same process as Bob to decrypt the message. Even knowing $\phi(N)$ is sufficient to decrypt.

3 Primality Testing

The first randomized algorithms for primality testing were developed in the 1970s: Solovay-Strassen and Miller-Rabin. Both therefore show that COMPOSITES $\in$ RP (PRIMES $\in$ Co $-$ RP, equivalently). A paper in the early 2000s shows that PRIMES $\in P$, but it's a much more complicated algorithm than either of these.

How does it work? We discussed earlier that we can use Fermat's Little Theorem as an almost-test, because for all $a < p$, $a^{p-1} = 1 \mod p$. For any N, the set of numbers $a < N$, the set of things which satisfy $a^{N-1} = 1 \mod N$ form a group. For primes, this group is all of the numbers less than N. For a composite N, we'll consider $a \in \mathbb{Z}_N^*$. The group $G = \{a | a^{N-1} = 1 \mod N\} < \mathbb{Z}_N^*$. If this subgroup is proper, it's at most half the size.

This suggests an algorithm. Given N, pick $a \in_R \{1, 2, \ldots, N-1\}$ and check if $a \in G$. If no, say that a is composite and if yes, say that a is prime. If G is a proper subgroup, this has probability of success at least half. However, there are composite numbers N (the Carmichael numbers) for which G is not a proper subgroup. This means the algorithm is incorrect, since it is always wrong about Carmichael numbers (unless you pick a number not in $\mathbb{Z}_N^*$).

The Miller-Rabin algorithm works as follows, leveraging the property of primes that the sequence (modulo N) $x, x^2, x^4, x^8, \ldots, 1$ has as its second-last term -1 if N is prime and sometimes -1 and sometimes something else if N is composite. Given an input N, write $N - 1 = m(2^s)$ where m is

odd. Then, pick $a \in_R [N-1]$ If $a^{N-1} \neq 1 \mod N$, output composite. If $a^m = 1 \mod N$, output prime. Otherwise, let $b = a^m$ and compute the sequence $b, b^2, \ldots, b^{2^s}$ and look at the term before the first one. If this is not -1, output composite. Otherwise, output prime.

If the number is composite, it will return composite with probability at least one half. If the number is prime, it will return prime always.

Definition. A **strongly one-way (length-preserving) function** f is a function such for all x, $|x| = |f(x)|$ and for any probabilistic polynomial-time adversaries A, there is a sufficiently large n such that $\Pr_{x \in U_n}[f(A(f(x))) = f(x)]$ is negligible. I.e. the probability of an adversary successfully inverting f is exponentially small.

A **weakly one-way (length-preserving) function** if there exists a polynomial $p(n)$ such that for any probabilistic polynomial-time adversary A, $\Pr_{x \in U_n}[f(A(f(x))) \neq f(x)] \geq \frac{1}{p(n)}$. I.e. any adversary which finds a sort of inverse of f fails sufficiently often.

Lemma. *The existence of a weak one-way function implies the existence of a strong one.*

Proof. Suppose that f is weakly one-way with polynomial $p(n)$. Define $g(x_1, x_2, \ldots, x_{np(n)}) = (f(x_1), f(x_2), \ldots f(x_{np(n)}))$ where each x_i has n bits. We'll show that g is strongly one-way.

Assume, for the sake of contradiction, that g is not strongly one-way. Let B be an algorithm that inverts g with non-negligible probability, at least $\frac{1}{q(n)}$ for some polynomial $q(n)$. We'll use B to invert f.

This algorithm I takes a string y and finds a string x such that $f(x) = y$. Given a y, I chooses an x_i to be equal to y and sets the rest of the x_j to be uniformly random, will try y in each position i, and calls B on $(f(x_1), \ldots, y, \ldots, f(x_{np(n)}))$.

Let S_n be the set of n-bit strings such that I succeeds in inverting $f(x)$ with probability at least $\frac{n}{p(n)}$. For any x in this set, we can boost the success probability to be exponentially close to one by repeating I on it enough times. In order for f to be weakly one-way, S_n can't have more than $(1 - \frac{1}{p(n)})2^n$ strings in it. We'll show that $|S_n| \geq (1 - \frac{1}{2p(n)})2^n$, which is our contradiction.

Consider the success probability of I on two cases. First, when all inputs are in S_n and second when at least one input is outside of S_n. The probability of all the inputs being in S_n is exponentially small. The probability that B succeeds on an input where at least one block is outside of S_n is also exponentially small, because otherwise the whole thing would have been in S_n. But the success probability of B is like the sum of these two probabilities, which is exponentially small, contradicting the assumption that B succeeds often enough. □

Lecture 4: Some Complexity Stuff (2019-02-13)

Definition. Suppose f is strongly one-way and $x = x_{n-1}x_{n-2}\dots x_0$ is a string of length n. We say that a bit $b(x_{n-1}x_{n-2}\dots x_0)$ (i.e. a boolean function) is a **hard-core bit** if any probabilistic polytime algorithm can't do more than negligibly better than random guessing $b(x)$ given $f(x)$.

How can we get a hard core bit?

Theorem. *Suppose we have such an f. Then there exists a strong one-way function g which has a hard core bit.*

Proof. Our function will be $g(x,r) = (f(x), r)$ where r is an n-bit string which gets 'passed through'. Clearly g is one-way if f is. Our hard core bit will be $b(x,r) = (x \odot r) \mod 2$, i.e. the inner product mod 2 of x and r.

We can show that b is hard core by showing that if an algorithm exists with a non-negligible advantage in computing b then we can invert f with non-negligible probability.

Suppose, as a warm-up to build intuition, that A is an algorithm which takes $(f(x), r)$ and always outputs $b(x,r) = (x \oplus r) \mod 2$ correctly. How can A use this to break g and find x? If we give A the sequence

1. $(f(x), 1000\dots) = x_{n-1}$
2. $(f(x), 0100\dots) = x_{n-2}$
3. $(f(x), 0010\dots) = x_{n-3}$
4. $\dots$

Then A can find which bits in x are equal to 1, and therefore recover x. Therefore such an A cannot exist, as otherwise g would not be strongly one-way.

Let's now suppose that A has a $\frac{3}{4} + \epsilon$ probability of correctly outputting $b(x,r)$, where ϵ is non-negligible. Call an x 'good' if the probability that A succeeds on it is at least $\frac{3}{4} + \frac{\epsilon}{2}$. The set of 'good' x's has size at least , which arises in the case where the set of good x's are those which succeed with probability 1 and those which succeed with just barely enough probability. I.e. the probability of success in this set is $1/p(n) \times 1 + (1 - 1/p(n))(3/4 \times \epsilon/2) \geq 3/4 + \epsilon$, and $1/p(n)$ is non-negligible. Therefore, if we can invert only the good x's, that's enough to violate the assumption of strong one-wayness.

Suppose x is good. We'll give a scheme to use A to go from $f((x), r)$ to (x, r). We can find the ith bit of x random choices of r. Choose $r \in_R \{0,1\}^n$ and let r_i be r but with the ith bit flipped. We then ask A for $b(x,r)$ and $b(x,r_i)$. Clearly both of these are independent. The probability of A failing on each is less than $1/4 - \epsilon/2$ so by a union bound the probability of failing on both is no more than $1/2 - \epsilon$. The probability that A is correct on both is at least $1/2 + \epsilon$, so it has a non-negligible advantage over guessing, so we can use amplification and majority voting to amplify this arbitrarily. We find x_i by taking the xor of the two answers that A gives for r and r_i.

We would finally like to construct a routine which can perform this task with only one call to A for each bit, thereby letting us have a procedure which works overall with probability at least $1/2 + \epsilon$. We can do this by adjusting the definition of 'good' and just making the query on r and just making

up an answer for the second query. If we redefine 'good' to mean the set of x's where the probability of A succeeding is greater than $1/2 + \epsilon/2$ and the same argument as before shows that at least $\epsilon/2$ of the x's are good.

Then, given a good x, we'll need polynomially many r, r_i as before, but we'll only ask the adversary about the (x, r_i) pairs. We only need the r's to be *pairwise* independent, rather than fully independent. To do this, let $\ell = \log p(n)$ and let $r_1, r_2, \ldots, r_\ell$ be uniformly random strings of length n. Then, for all $J \subset [\ell]$, let $r_J = \bigoplus_{i \in J} r_i$. There will be roughly $p(n)$ different r_J strings. The inner product of x with any r_J will be the xor of the inner product of x with all of the r_i used to construct r_J. We can then ask the algorithm to run on the r_i and this gives us some information about the r_Js. Once we have each bit, we can verify it's correct by passing it to f. We also need to use Chebyshev's inequality rather than Chernoff's to do the amplification by majority voting.

□