Prof. Friedrich Eisenbrand
Alfonso Cevallos

Discussions from: March 17, 2015

Computer Algebra

Spring 2015

Assignment Sheet 3

Note: These are just notes and not necessarily full solutions to each exercise. Please report any mistakes you may find.

Exercise 1

Let N be an odd, composite number. We will prove that

$$N \text{ is Carmichael} \Leftrightarrow \forall \text{ prime } p|N:\ p^2 \nmid N \text{ and } p-1|N-1.$$

Backward implication. No prime power divides N, so its prime factorization is $N = p_1 \cdots p_k$, where all primes are distinct. So the Chinese remainder theorem (CRT) states that there is a ring isomorphism between $\mathbb{Z}_N^*$ and $\mathbb{Z}_{p_1}^* \times \cdots \times \mathbb{Z}_{p_k}^*$. On the other hand, we need to prove that for any a coprime with N, $a^{N-1} \equiv 1 \mod N$. But by the previous isomorphism, we have that a is coprime with N iff it is coprime with each p_i; and similarly $a^{N-1} \equiv 1 \mod N$ iff $a^{N-1} \equiv 1 \mod p_i$ for each p_i. Thus it is enough to prove the Carmichael property locally for each prime: Let a be coprime with p_i; from Fermat's little theorem, $a^{p_i-1} \equiv 1 \mod p_i$. Now, we know that $p_i - 1|N-1$, so there is an integer t such that $N-1 = t(p_i-1)$. Then, $a^{N-1} \equiv a^{(p_i-1)t} \equiv 1^t \equiv 1 \mod p$. This completes the proof.

Forward implication. Let p be a prime factor of N, and let p^k be the highest power of p dividing N (we will prove that $k = 1$). As p is odd, we know that $\mathbb{Z}_{p^k}^*$ is cyclic, i.e. there is a number $c \in \mathbb{Z}_{p^k}^*$ whose order is $|\mathbb{Z}_{p^k}^*| = (p-1)p^{k-1}$. Now we define the number $C \in \mathbb{Z}_N^*$ by the system $C \equiv c \mod p^k$ and $C \equiv 1 \mod N/p^k$ (the CRT ensures that C is well defined). Since N is Carmichael, we know that $C^{N-1} \equiv 1 \mod N$, which means that $c^{N-1} \equiv 1 \mod p^k$. This last congruence implies that $N-1$ must be a multiple of the order of c, i.e. $(p-1)p^{k-1}|N-1$. As p does not divide $N-1$, it must be that $p^{k-1} = 1$ and $k = 1$; and also $p-1|N-1$.

Exercise 2

We know that the Fermat test fails for Carmichael numbers with high probability. In this exercise we convince ourselves of this fact by computing this probability. If $N = \prod_{i=1}^k p_i$ is a Carmichael number, then the Fermat test outputs "composite" iff it picks a number $a \in \{1, \cdots, N-1\}$ that is not coprime with N. Then $P := P[\text{Fermat outputs "composite"}] = 1 - P[gcd(a,N) = 1] = 1 - \frac{\phi(N)}{N-1}$. For large N, the following upper bound gives a pretty accurate estimate:

$$P \leq 1 - \frac{\phi(N)}{N} = 1 - \frac{\prod_i (p_i - 1)}{\prod_i p_i} = 1 - \prod_i \left(1 - \frac{1}{p_i}\right) \leq 1 - \left(1 - \sum_i \frac{1}{p_i}\right) = \sum_{i=1}^k \frac{1}{p_i}$$

Exercise 3

Assume that $\pi(x) \sim \frac{x}{\ln x}$, and fix any $\epsilon > 0$, which is henceforth considered as a constant. Define $g(x) = \frac{\pi(x)\ln x}{x}$, and $h(x) = (1+\epsilon)g(x(1+\epsilon))$. From these definitions, it's clear that $g(x) \to 1$ and $h(x) \to 1+\epsilon$, so $h(x) - g(x) \to \epsilon$, as $x \to \infty$. Now, $h(x) = \frac{\pi(x(1+\epsilon))\ln x(1+\epsilon)}{x} = \frac{\pi(x(1+\epsilon))\ln x}{x} + \frac{\pi(x(1+\epsilon))\ln(1+\epsilon)}{x}$, where the second term is clearly $o(1)$ (so it is safe to ignore it). Therefore $\lim_{x\to\infty} \frac{[\pi(x(1+\epsilon))-\pi(x)]\log x}{x} = \lim_{x\to\infty}(h(x)-g(x)) = \epsilon$. And the result follows.

In particular, if $\epsilon = 1$ and $x = 2^n$, we get $\frac{\pi(2^{n+1})-\pi(2^n)}{2^n} \to \frac{1}{\ln 2^n} = \frac{1}{n\ln 2}$, as $n \to \infty$. And the second result follows.

Exercise 4

1. Define the following events
   A: You have the winning ticket; F_A: John predicts that you have the winning ticket.

   The probabilities that we know are: $P[A] = 10^{-6}$; $P[F_A|\bar{A}] = 1-p$; and $P[F_A|A] = p$. And what we want is: $P[A|F_A] \geq 0.9$. So now we have to apply Bayes formula

   $$P[A|F_A] = \frac{P[F_A|A]P[A]}{P[F_A]} = \frac{P[F_A|A]P[A]}{P[F_A|A]P[A] + P[F_A|\bar{A}]P[\bar{A}]}.$$

   Which can be conveniently rewritten as the formula below. It gives $p \geq 1 - 1.11 \times 10^{-7}$.

   $$\frac{P[F_A|\bar{A}]}{P[F_A|A]} = \frac{P[A]P[\bar{A}|F_A]}{P[\bar{A}]P[A|F_A]}.$$

2. This problem has the same flavor as the one above. Fix n, k and $\epsilon > 0$, and let N be an n-bit odd number chosen uniformly at random. Define the events
   A: N is prime; F_A: k consecutive runs of MR over N output "probably prime".

   The probabilities we know are: $P[A] \geq c/n$, for a constant c independent from n (see exercise 3 above); $P[F_A|\bar{A}] \leq 2^{-k}$; and $P[F_A|A] = 1$ (these results were seen in class). And what we want is $P[A|F_A] \geq 1-\epsilon$, or $P[\bar{A}|F_A] \leq \epsilon$. We reuse the formula above.

   $$P[F_A|\bar{A}] = \frac{P[A]P[\bar{A}|F_A]}{P[\bar{A}]P[A|F_A]} > P[A]P[\bar{A}|F_A], \text{ so } P[\bar{A}|F_A] < \frac{P[F_A|\bar{A}]}{P[A]} \leq \frac{2^{-k}}{c/n} = \frac{n}{c2^k} \leq \epsilon.$$

   So it is enough to pick $k \geq \log_2 \frac{n}{c\epsilon}$. Remark: If ϵ is a constant, we get $k = O(\log n)$; and if ϵ is exponentially small in n, i.e. $1/\epsilon = 2^{O(n)}$, we get $k = O(n)$.

3. If N is composite, we can think of each iteration of MR over N as an independent Bernoulli trial, with outcome either success ("composite"), or failure ("prob. prime"); and the success probability p depends on N, but in any case $1/2 \leq p \leq 1$. We will run as many iterations as necessary until we get a "composite", so what we are interested in is the probability distribution of the number X of such Bernoulli trials needed to get one success. Variable X has what is called a *geometric distribution*, with parameter p, and it is an easy exercise to prove that its expected value is $\mathbb{E}[X] = 1/p$. Thus $1 \leq \mathbb{E}[X] \leq 2$, the important point being that $\mathbb{E}[X] = \Theta(1)$, independent from the size of N.

4. See the code. We chose to run up to $k = 1.1n$ iterations of MR over each candidate number, which corresponds to an exponentially small error parameter ϵ. See part 7 for an explanation for this choice.

5. Remember that the complexity of fast mod. exponentiation (FME), for an exponent of bit-size e and a modulo of bit-size n, is $O(eM(n))$, where $M(n)$ is the complexity of multiplying two number of size n (if we use Karatsuba, $M(n) = O(n^{log_2 3}) = O(n^{1.585})$). In its core, MR is basically the same as FME, where the exponent is of similar size as the modulo, so its complexity is $O(nM(n))$.

   Now, since the probability for a random n-bit number N' to be prime is $\Omega(1/n)$, the expected number of trials until we get a prime is $O(n)$. (Here we are dealing again with a geometric distribution, just as in part 3.) Now we put all the previous results together: in expectation, we will generate $O(n)$ composite numbers N', and run $O(1)$ iterations of MR on each of them, and then we will find a prime number N, and run k iterations on it (see part 2). This gives a total of $O(n+k) = O(n)$ iterations of MR (we obtain the same complexity whether ϵ is a constant, or exponentially small in n). Therefore the expected complexity of our prime generator is $O(n^2 M(n))$.

6. See the code. We implement first the extended Euclidean algorithm. For the given example, the original message is ln

7. To simplify the analysis, suppose that p and q have the same bit-size n, and we obtain them with the help of our prime generator, with security parameter $\epsilon \leq 2^{-n}$. Now, if P is the probability that the scheme fails, because either p or q is not prime, then $P \leq 2\epsilon \leq 2 \times 2^{-n}$. We consider such a probability negligible, because that's the same probability that an adversary has to break our scheme simply by guessing either p or q. As seen in part 5, we can afford such an overwhelming level of security without increasing the overall complexity of the prime generator algorithm.

Exercise 5

1. Suppose there is an algorithm $A(l, N)$ that solves problem (P), with a running time $f(l, N)$ that is polynomial in the input size, i.e. polynomial in $\log N$. We use it to construct the following algorithm $B(N)$ for problem (Q).

Data: $N \in \mathbb{N}$.
Result: Printed list of prime factors of N, with multiplicities.
while $N > 1$ **do**
    Find l' = largest proper factor of N, by executing a binary search with variable l over $\{1, \cdots, N-1\}$, that calls on $A(l, N)$;
    Print $p = N/l'$. It must be the smallest prime factor of N;
    Update $N = l'$;
end

The algorithm is clearly correct. Suppose that N has k prime factors, counting multiplicities, then it is easy to see that $k \leq \log_2 N$. Algorithm $B(N)$ executes k iterations, each iteration makes $O(\log N)$ calls to algorithm A, and each call takes at most time $f(N,N)$. So the running time of $B(N)$ is $O(\log^2 N f(N,N))$, which is polynomial in the input size.

Remark: Prime factorization (problem (Q)) is believed to be intractable, i.e. no polynomial time algorithm is known, or is expected to be found in the future. This presumed intractability provides the security base of several crytographic schemes, such as RSA. *Polynomial reductions*, like the one above, prove that other problems (such as (P)) must also be intractable, and thus can also be used for cryptographic applications.

2. The prime factorization of $N = \prod p_i$ can be used to build both YES- and NO-verifiers for problem (P), as follows. Notice first that any number N has at most $\log N$ prime factors, each of size at most $\log N$, so the size of the factorization is polynomial in $\log N$. Also, in polynomial time we can check the correctness of the factorization, by checking that each term is indeed a prime, and that the product equals N. Finally, if p is the smallest prime factor of N, then the largest proper factor of N is N/p. If this is greater than l, then the answer is YES; otherwise the answer is NO.