Beginning NP-completeness

CS255

Chris Pollett

Apr. 24, 2006.

Outline

- Finishing up Rabin-Miller Correctness
- Preliminaries for NP-completeness Chapter

Finishing up Error Rate Analysis of Miller Rabin

- Last Day, we began to show that if n is an odd composite number, then the number of witnesses to the compositeness is at least $(n-1)/2$.
- We did this by first showing any nonwitness must be in $\mathbf{Z}^*_n$ and these elements form a subgroup of $\mathbf{Z}^*_n$.
- We then wanted to show that they form a proper subgroup to give the result.
- The case where there was an x such that $x^{n-1} \not\equiv 1 \pmod{n}$ was handled last day.
- Suppose for all x in $\mathbf{Z}^*_n$, $x^{n-1} \equiv 1 \pmod{n}$.

More Miller-Rabin

- Then n is a Carmichael number.
- First, notice n can't be a prime power. To see this suppose $n = p^e$. Since n is odd, p must also be odd, so $\mathbf{Z}^*_n$ will be cyclic, so has a generator g and by assumption we have $g^{n-1} = 1 \bmod n$. On the other hand, $\mathrm{ord}(g) = \phi(n) = (p-1)p^{e-1}$ and the discrete logarithm theorem implies $n-1 \equiv 0 \pmod{\phi(n)}$. i.e., $(p-1)p^{e-1} \mid p^e - 1$, which is impossible.
- So suppose n is odd, not a prime power and composite. We can then decompose it as $n = n_1 n_2$ where n_1 and n_2 have different prime factors.
- Recall, t and u are defined so that $n-1 = 2^t u$ and u is odd.
- Recall Witness computes the sequence $X = \langle a^u, a^{2u}, a^{(2^2)u}, .., a^{(2^t)u} \rangle$ (all mod n)
- Call a pair (v, j) acceptable if v is in $\mathbf{Z}^*_n$ and $v^{(2^j)u} \equiv -1 \pmod{n}$.
- For example, $v = n-1$ and $j=0$ is acceptable.
- Pick an acceptable pair (v, j) with the largest possible value $j \leq t$.
- Can show $B = \{x \in \mathbf{Z}^*_n \mid x^{(2^j)u} \equiv \pm 1 \pmod{n}\}$ is a subgroup of $\mathbf{Z}^*_n$.

Even More Miller Rabin

- Every nonwitness must be a member of *B*, since the sequence *X* produced by a nonwitness must be all 1's or else have a -1 no later than the *j*th position, by the maximality of *j*.
- We now use the existence of v such that $v^{(2^j)u} \equiv -1 \pmod{n}$ to show there exists a w in $\mathbf{Z}^*_n - B$.
- Since $v^{(2^j)u} \equiv -1 \pmod{n}$ we have $v^{(2^j)u} \equiv -1 \pmod{n_1}$.
- So we can find by the Chinese Remainder Theorem a w such that $w \equiv v \pmod{n_1}$ and $w \equiv 1 \pmod{n_2}$.
- In which case, $w^{(2^j)u} \equiv -1 \pmod{n_1}$ and $w^{(2^j)u} \equiv 1 \pmod{n_2}$.
- So using Chinese Remainder theorem, we get $w^{(2^j)u}$ is not congruent to $\pm 1 \pmod{n}$.
- So w is not in *B*. Nevertheless, one can show its gcd(w, n) =1 using the Chinese Remainder Theorem together with the fact that v is in $\mathbf{Z}^*_n$. So w is in $\mathbf{Z}^*_n$ completing the proof.

Introduction to NP-Completeness

- Most algorithms we have studied run in polynomial time or some randomized variant.
- That is on all inputs of length n, the algorithms we've considered run in time at more $O(n^k)$ for some fixed k.
- We'll start looking today at some problems for which it is unknown if such efficient algorithms exist.
- First we make formal what it is we mean by polynomial times, then we'll consider variant which might be harder.

Abstract Problems

- We need a framework for describing problems and reasoning about their runtimes.
- We define an **abstract problem** Q to consist of a set of **instances** I and a set of **solutions** S.
- For example, for SHORTEST-PATH the instances might be triples consisting of graph and two vertices. A solution might be a sequence of vertices for a path between those two points in the graph of shortest distance.
- We will be interested in a subclass of problems called **decision problems**, where the answers are always yes or no.
- For example, does there exists a shortest path of size at most k?
- It is usually straightforward to binary search from a way to solve the decision problem to solve the associated o**ptimization problem.**
- Here optimization problems are where we want to find a largest or smallest value.

Encodings

- An encoding of a set S of abstract object is a mapping from S to binary strings.
- For example one can encode the natural numbers {0, 1, 2, ..} as strings {0, 1, 10,..}.
- One can encode legal English sentences using ASCII, etc.
- A computer algorithm "solves" some abstract decision problem by going from an encoding of a problem instance as an input to 0 or 1 as output.
- We call a problem whose instance set is the set of binary strings a **concrete problem**.
- We say an algorithm solve the problem in $O(T(n))$ time if when provided a problem instance i of length $n = |i|$, the algorithm can produce the solution using $O(T(n))$ steps.
- A concrete problem is called polynomial time decidable if there is an algorithm that solves it which runs in time $O(n^k)$ for some fixed k.
- We write **P** for the class of all such decision problems.
- Similarly, we can define the class of polynomial computed functions f:$\{0,1\}^*$-->$\{0,1\}^*$.

Formal Languages

- In order to study decision problems its useful to have an understanding of formal languages.
- An **alphabet** Σ is a finite set of symbols.
- A **language** is a set of strings over the symbols in an alphabet.
- Some common ways to create new languages from old ones is via unions, concatenation, and star.

NP Languages