CS 70 Discrete Mathematics and Probability Theory
Fall 2011 Rao Rough Outline Lecture 1

The reader shold be aware that thes notes have been subject to minimal if any editing and should not be distributed.

This course is about formal analysis in the context of discrete math and probability and its applications in computer science.

Today, we highlight some of the applications and hint at the mathematical ideas that we will use to derive these applications.

Secret sharing, coding theory.

Consider the problem of shaing (parts of) a secret number with three people, where any two can figure out the number, and any one person knows nothing about the number.

Let me describe a secret sharing scheme using a sequence of examples.

- Secret: 2. Shares: 2, 4,6.
- Secret: 3. Shares: 1000, 1003, 1006.
- Secret: 5. Shares: 41, 46,51.

This is an old "guess the pattern" problem, where the patterns consist of an arithmetic sequence: add the secret number every time.

Now, guess the secret given the following shares.

- Shares: 2, *,6. What is the secret?
- Shares: *, 8, 12. What is the secret?
- Shares: *,8,*. What is the secret?

From even the first example above, where the secret is 2, it becomes clear that the order of the shares matter. That is, the shares correspond to share 1, share 2, and share 3. With this, it is easy to see that the second example's secret is 4.

In the last example, with this scheme, one cannot determine the secret at all.

How can we generalize this scheme to share the secret among more people, have the minimum sized set of knowing people be larger?

Viewing the shares as a set of ordered pairs, E.g;, $(1,2);(2,4);(3,6)$, and associate a graph or function between share number and share value, one sees that the secret is encoded as the slope of a line. We have the very familiar notion that **points determine a line**. Moreover, one point tells is nothing about the slope of a line.

So, to generalize to having more shares, one can simply choose more points on the line. Any two suffice to reconstruct the line.

How about needing a larger group to collaborate to reconstruct the secret? Here, we will use the simply use functions which are higher degree polynomials. For example, three points uniquely determine a parabola, or any k points determine a degree $k-1$ polynomial. We can encode the secret as one of the coefficients of the the polynomial.

Later in this course, we will see how to do this computation using integers and small numbers.

This scheme is actually broadly used in communication. For example, if we have a k packet message that we want to send over a lossy communication channel, can we send n packets where any k of them allow us to reconstruct the original message. Here, each "sent" packet will correspond to a share of the original message.

A more challenging problem is a "noisy" channel, a channel that changes the contents of a packet. Here you wish to send a k packet message using n packets, and reconstruct the message of any g packets remain good. Here, we will need g to be larger than k, but can still do very well. Again, the constructions are based on properties of polynomials! Indeed, properties that were developed in the California 10th grade curriculum for real numbers. Here, we will use analagous properties over finite fields, as that is what computers do.

Cyrptography: Public Key encryption.

Since time immemorial: share codebook... secret key. Both ends need to share a codebook. A message is sent by encoding the message using the "codebook" and then decoding the message using the same "codebook".

Diffie and Hellman devised a public key system consisting of a public key, secret key pair, a method to encode a message using the public key, and a method to decode the message using the corresponding secret key. Rivest, Shamir, and Adleman later devised a public key scheme based on modular arithmetic. This is the basis of modern cryptography in practice.

In particular, a greatly simplified depiction of what happens when Bob wishes to send his credit card number to Amazon.

Amazon: I am Amazon; my public key is $K = (N, e)$!

This public key is known to the world (in particular to Bob's browser and everyone else's browser.)

Bob: $y = E(x = \text{"5422132217861111."}, N, e) = x^e \mod N$

Here, we use modular arithmetic. When we say $x \mod N$ we mean something akin to the remainder of x mod N, i.e., $13 \mod 7$. (In fact, we mean that the whole world only consists of $0, \ldots, N-1$. And $N+1$ mod N is simply another representation of $1 \mod N$. So, the $\mod N$ simply means we are in this world. The $\mod N$ will be at the end of an equation as above.)

Now, an evil eavesdropper (Eve) is snooping on the router.

Eve(il): See's y.. hopefully can't figure out x even though she knows N and e.

A tiny bit of intuition of how this might work. Let's consider an example where the encryption and decryption are done with one prime, 7. Here, we have.

Eve: for what x, is $x^5 = 5 \mod 7$?

In modular arithmetic, it is not clear how to take roots. For real numbers (or rationals) one can do some kind of binary search, but other than trying all of them it is not clear how to take the 5th root of something mod p.

But I know $(x^5)^5$ mod 7 = 1 by,

Fermat's (Little) Theorem: $a^{p-1} = 1 \mod p$ for prime p and for a non-multiple of p.

Now, with this theorem, we have that

$$(x^5)^5 = x^{25} = x^2 4x = (x^6)^4 x \mod 7$$

Now, by Fermats theorem, we have that $x^6 = 1 \mod 7$, and we have that $(x^5)^5 = 1 \mod 7$.

(This takes a bit of belief that basic rules of arithmetic; e.g., $(x^a)^b = x^{ab} \mod p$, apply to modular arithmetic.)

Thus, this mathematical fact called Fermat's Little Theorem gives us a somewhat counterintuitive decryption method. Unfortunately, Eve can as easily figure out the decryption method as Bob when N is a prime. But, the analagous decryption method when N is the product of two unknown primes along with the analagous theorem (Euler's Theorem) remains the state of the deployed technology.

Probability...

We will cover a fair bit of probability in this class. While primarily we will cover discrete probability we will touch on continuous probability and the connections should give you a better grasp of both.

Let's take a brief look at some examples which you will be able to formally reason about after this course.

You are offered a million dollars for winning 2 **consecutive** games of 3 1-on-1 games of Jeapardy against Watson (IBM's supercomputer which happens to be very good) and me (I am bad).

You get to choose from the following two orders.

Would you play? Watson-me-Watson or me-Watson-me.

Again, you are likely to lose agaist Watson where you are likely to win against me. You might think it better to play me twice, but the consecutive condition interferes with this intution. This and many other situations typically require us to carefully and formally reason to ensure we are acting correctly. We will provide the formalism and experience for doing so in this course.

Another big topic is dealing with understanding the behavior of large numbers of events.

For example, there are a 100 mutual funds, which trade once a day for a year. One fund makes the right choice 60% of the time, should you pay high fees to go with the fund manager?

This type of problem has a bit of modelling involved, what is the definition of "right choice", etc, and a bit of understanding of the computation of quantities like variance and their use in understanding performance. While some of you, in AP statistics, perhaps, could do this using various tables, we will start from the basics and derive concepts used to analyse these situations.

Self reference and undeciability..

Here, we consider the following sentence.

This statement is false.

This statement is neither true nor false.

This is possibly one of the great “disasters” of ..uh... a mathematics? philosophy? life? This kind of argument leads to problems in forming foundations of mathematics; for example, consider the set that of sets that don’t contain themselves. Does the set contain itself? The self reference here is the reference to itself.

For computer scientists, this reasoning results in a fundamental limit on whether programs completely "understand" other programs?

An example, of course, here is the metacircular evaluator that one writes in CS61A

For example, can we write a program, call it **HALT** that checks for another program, *P*, whether it is an infinite loop when run on itself?

Consider, a program program **FLIP-HALT** takes a program *P* and halts if P runs forever on P else it runs forever. This program can be written if **HALT** can.

What does **FLIP** do on **FLIP**?

Can the program **HAlT** exist?

(In class, a suggestion was made that recursion is the “self-reference” concept. It is not really the problem. The problem has more to do with the fact that the input to a program can be (a representation of) the program itself.)