POIS-5

17/01/2023 (Tue)

Special Mathematical Objects

- One-way Functions
- Pseudo-Random Generators

In the last lecture we described what a negligible function is. Essentially, A function $f(n)$ is said to be negligible if $\forall$ polynomials p, $\exists n_0$ st $n > n_0$

$$f(n) < \frac{1}{p(n)}$$

Consider the constant function given by $f(n) = 10^{-6}$. Although it seems that the function value is extremely low for all values of n. It is still NOT a negligible function, because if we take the function $p(n) = n$ then for $n \geq 10^{-6}$ the above inequality breaks, therefore the function is not negligible.

Similarly we can also show that inverses of any polynomial function CANNOT be negligible, there will always exist another polynomial which grows faster than it (consider a polynomial of higher degree).

Then which functions are negligible exactly? We notice that functions of the form $f(n) = a^{-n}$ where $a > 1$ are negligible, hence functions such as $e^{-n}, 2^{-n}$ etc. will be negligible. We can see that it is obvious, since if we consider the taylor expansion of any exponential function, it consists of all powers of x. Therefore it rises faster than any polynomial function, since it also has terms of higher degree in it, however a more formal proof goes as follows.

Adding this part since it might come in quiz ig

Say we want to prove that e^{-n} is negligible. Then we need to show that for all $n > n_0$ the following inequality holds

$$e^{-n} < n^{-k}$$

Where k was the degree of the polynomial we are comparing with.
Taking $\log$ on both sides we have

$$\begin{aligned} -n \log e &< -k \log n \\ \implies n &> k \log n \\ \implies \frac{n}{\log n} &> k \end{aligned}$$

Consider the expansion of e^x

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} \dots$$

Consider only the first three terms

$$e^x > 1 + x + \frac{x^2}{2!}$$

divide by x on both sides, and take $x = \ln n$

$$\frac{e^x}{x} > \frac{1}{x} + 1 + \frac{x}{2}$$

$$\frac{e^x}{x} > \frac{x}{2}$$

$$\frac{n}{\ln n} > \frac{\ln n}{2}$$

Therefore it becomes very clear now that the value of n_0 to be chosen is such that $\frac{\ln n}{2} > k$ which gives us

$$n_0 = e^{2k}$$

Which completes our proof.
In general, for the function $f(n) = a^{-n}$ we need to choose $n_0 = e^{2k/\ln a}$ to make it smaller than the inverse of any polynomial of degree k.

There are other functions such as $\frac{1}{(\log n)!}$ which are negligible but note that $\frac{1}{(\log \log n)!}$ is not negligible

Proof that $\frac{1}{\log n!}$ is negligible

We just need to show that $\log n!$ rises faster than any polynomial function, assume that the polynomial was of degree k, in that case we have to prove that for all $n > n_0$ we have

$$(\log n)! > n^k$$

Take $n = e^x$ on both sides and take logarithm on both sides

$$\begin{aligned} x! \ ? \ (e^x)^k = e^{kx} \\ \log x! \ ? \ kx \end{aligned}$$

We can use Stirling's approximation here, which states that

$$\ln n! = n \ln n - n + O(\ln n)$$

Therefore we have $\ln x! > x \ln x - x$
plugging that into the equation gives us

$$\begin{aligned} x \ln x &> kx \\ \ln x &> k \\ x &> e^k \end{aligned}$$

Therefore for a critical value of $n_0 = e^{e^k}$ we have the required inequality.

(No clue about the $(\log \log n)!$) proof.

Special Mathematical Objects (One Way Functions and Psuedo Random Generators)

Pseudo Random Generators

It is basically a function which takes input of size n and maps it to a string of size $l(n)$. Therefore we have

$$G : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$$

Pseudo Random Generators satisfy the following two properties.

- Expansion: $l(n) > n$ meaning the size of the output string is should be larger than the size of the input string
- Pseudo Randomness: Assume that we have a probabilistic polynomial time Distinguisher D. A Distinguisher is defined as a machine, which on input as string s, outputs 1 if and only if, there exists a string w such that $G(w) = s$ and outputs 0 otherwise. Then we have

  $$|P\left[D(U_{l(n)}) = 1\right] - P\left[D(G(U_n)) = 1\right]| \leq negl(n)$$

  This says that the probability of the distinguisher to figure out the decoded form of the input string, when given a perfect random generator (basically a uniform distribution) and a pseudo random generator is negligible.
  Now, consider a pseudo random generator which maps a string of length 100 bits, to a string of length 1000 bits, we realize that for a truly random generator , the output string is uniformly distributed amongst 2^{1000} strings. However, for the pseudo random generator, the output of the generator depends on the input string, which is only of 100 bits, therefore we can have only 2^{100} possible output strings for a pseudo random generator, therefore the distributions of a truly random generator and a pseudo random generator are very different.
  One way to distinguish between a pseudo random generator and a truly random generator would be to take around $2^n + 1$ samples from each, if the generator is truly random, then we should not see any repeats (repeats with a very low probability), however if it is pseudo random then we should see some string repeat twice (repeat probability is exactly 1).

Pseudo random generators make One-Time Pads practical

One of the primary concerns with the one-time pad, was the fact that the size of the keyspace, and the size of the messagespace are equal therefore sending the key through the secure channel is going to be problematic, since the speed of the secure channel is much slower than the speed of the unsecure channel, through which the message passes.
This can be solved using Pseudo-random generators. The idea is as follows

- Assume that the key is of size n and the message is of size $l(n)$.
- Apply the pseudo random generator on the key, and find the output string, which will be used to encode the message (note that now the message, and the new key, which is the output of the pseudo random generator is of equal length)
- Pass the encrypted message through the unsecure channel
- Pass the key (of length n) through the secure channel
- Again use the pseudo random generator on the other end to generate an output string which will be used to decode the message.
  Notice that this solves the problem regarding the speeds of transferring the messages being bottlenecked by the secure channel, the key is now much smaller, and hence will be much easier to transfer.

One-Way functions

We say that a function $f : \{0,1\}^n \rightarrow \{0,1\}^n$ is said to be a one way function if

- It is easy to compute, meaning that there exists a deterministic polynomial time turing machine M, which can calculate the value of $f(x)$ when given input x
- It is hard to invert, meaning that for all probabilistic polynomial time turing machines M, the following condition holds

  $$P[M(f(x)) = y, f(x) = f(y)] \leq negl(n)$$

  This states that the probability that the machine succesfully calculates the inverse of $f(x)$ is negligible. But inverse process can't be harder than NP.
  Therefore we have the a function, which is easy to calculate (in P) and whose inverse is hard to calculate (in NP but not in BPP)
  We consider one such problem, which is the Discrete Logarithm Problem (DLP)

Discrete Logarithm Problem

Assume that we are given a cyclic group G (A cyclic group is basically a group, which can be generated by a single element, called the generator of the group). A common example of such a group is Z_p^*, which consists of $p-1$ numbers between 1 and $p-1$. Notice that these numbers form a group under multiplication modulo p, with the identity being 1, and every number having a unique inverse.

Consider a very specific example for now, say Z_5^*, which consists of 1,2,3,4. We notice that 2 is a generator for this group, since (Note that we are doing power modulo p)

$$2^1 = 2$$
$$2^2 = 4$$
$$2^3 = 3$$
$$2^4 = 1$$

Therefore every element is generated just by repeatedly using 2, hence 2 is a generator of the group.
Now we know that every element of G can be written as a power of the generator g. That is

$$g^x \bmod p = y$$

Now the problem is, given g, p, y calculate the value of x.
Notice that we do have a relatively simple brute force solution to this, since the value of x cannot exceed p. Therefore we can just check every power between 0 and $p-1$. The issue is that exponentiating a number is an extremely expensive process, hence for extremely large groups, meaning for very large values of p, this problem becomes extremely hard.

Hard Core Predicate

In the above Discrete Logarithm Problem, we realized that finding x is extremely hard, however is there any info we can get about x ? The Answer is Yes, we can infact, find the least significant bit of x, that is, we can figure out whether x is even or odd. Hence One-Way Functions might not be perfect, they can leak some information about the input. This leads us to the concept of hard-core predicates, information about the input which cannot be leaked.

One can think of hard-core predicates as follows, consider the function $f(x) = x^2$ notice that the function can very well mask the sign of the input, since negative numbers will also have a positive square. A predicate is essentially a function p such that it outputs either 0 or 1, depending on some condition on the input. Hence it is basically a question, which just has a yes or no answer. We can define our predicate for the square function as "is the number positive", in other words.

$$p(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Notice that it is very easy to calculate the value of $p(x)$ given the value of x. However given the value of $f(x)$ it is impossible to predict whether the value of the input was positive, or negative (both outcomes are equally likely). Hence the best we can do, is just randomly guess the output of $p(x)$ as 1 or 0.

Formally a Hard-Core Predicate is defined as: For a family of One-Way Functions $f : \{0,1\}^n \to \{0,1\}^n$ a predicate $h : \{0,1\}^n \to \{0,1\}$ is said to be a hard-core predicate if

- It is easy to calculate the value of $h(x)$ form x
- It is impossible to do better than randomly guessing between the outputs of the hcp, that is $\forall PPTM\ M, P[M(1^n, f(x)) = h(x)] \leq \frac{1}{2} + negl(x)$

Algorithm for calculating the LSB of x

- Compute the value of $y^{\frac{p-1}{2}} \mod p$ in polynomial time. This can also be thought of as, $g^{\left(\frac{x.(p-1)}{2}\right)} \mod p$
- If x is even, i.e it is of form $2k$. then we are basically calculating $g^{k(p-1)}$ modulo p, which is equal to 1 by fermat's little theorem.
- If x is odd, then the final value is -1(apparently $y^{\frac{p-1}{2}}$ modulo p is either 0,1, or -1 only, with 0 being a possibility only when p divides y)

Fermat's Little Theorem

States that for a prime number p which does not divide the number a, we have

$$a^{p-1} \equiv 1 \mod p$$

Proof

We list the first $p-1$ positive multiples of a as

$$a, 2a, 3a \ldots (p-1)a$$

We notice that if $a \cdot i$ and $a \cdot j$ have the same remainder modulo p then we can write it as

$$\begin{aligned} a \cdot i &\equiv a \cdot j \mod p \\ a(i-j) &\equiv 0 \mod p \end{aligned}$$

This would imply that p either divides a or divides $(i-j)$ or both, since p is prime, now, we already assumed that p does not divide a, therefore p has to divide $i-j$. Hence we have $i-j$ to be a multiple of p.

Now notice that any of the first $p-1$ multiples of a do not have this property, therefore none of them have the same remainder modulo p. Since we have $p-1$ such numbers, and we can have $p-1$ possible remainders for each of them (Note that 0 cannot be a remainder). Therefore each of $a, 2a, 3a, \ldots (p-1)a$ must be congruent to 1,2,3,...p-1 in some order.

Multiplying all of these congruences together we have

$$\begin{aligned} a(2a)(3a)\ldots((p-1)a) &\equiv 1 \times 2 \cdots \times (p-1) \mod p \\ a^p(p-1)! &\equiv (p-1)! \mod p \\ a^p &\equiv 1 \mod p \end{aligned}$$

Which gives us the desired result