Fingerprinting and primality

Curs 2019

Checking matrix multiplication

Problem: Given 3 square matrices ($n \times n$), A, B and C, we want to see if $A \times B = C$.
Easy solution: compute $A \times B$ and compare with C.

$n \times n$ matrix multiplication:

1. Naive algorithm: $O(n^3)$
2. Strassen (1969): $O(n^{2.81})$
3. Coppersmith-Winograd (1987): $O(n^{2.376})$
4. Vassilevska (2015): $O(n^{2.373})$

Can we check in $O(n^2)$ if $A \times B = C$?

Freivalds algorithm for checking if $A \times B = C$ (1977)

Given $n \times n$ matrices A, B, C

Freivald A, B, C
choose u.a.r. $\vec{r} \in \{0, 1\}^n$
if $A(B\vec{r}) = C\vec{r}$ **then**
  output true
else
  output false
end if

Choosing u.a.r. the n dimensional Boolean vector $\vec{r}$ is choosing independently with probability $1/2$ each of the n bits. Therefore the probability of any given $\vec{r}$ is $1/2^n$, and the cost of generate one vector in n.

The time complexity of Freivalds is $\Theta(n^2)$.
Notice: if $AB = C$ the algorithm yields always the correct answer.
It could be that $AB \neq C$ the algorithms may yield the wrong answer ($AB = C$) with a certain probability
(ex: with prob.$=1/2^n$, $\vec{r} = (0, 0, \ldots, 0)$)

Example

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 1 & 4 \\ 1 & 5 & 9 \\ 2 & 6 & 5 \end{pmatrix} \quad C = \begin{pmatrix} 11 & 29 & 37 \\ 29 & 65 & 91 \\ 47 & 99 & 45 \end{pmatrix}$$

Pick with probability $= \frac{1}{2^3}$ $\vec{r} = (1, 1, 0)$:

$$A(Br) = \begin{pmatrix} 40 \\ 94 \\ 148 \end{pmatrix} \quad \neq Cr = \begin{pmatrix} 40 \\ 94 \\ 146 \end{pmatrix}$$

Therefore $AB \neq C$.

Error probability

Theorem If $AB \neq C$ then $\mathbf{Pr}\left[A(B(\vec{r})) = C\vec{r}\right] \leq \frac{1}{2}$
Proof
Neat trick: As $AB \neq C$ taking $D = AB - C$, then $D \neq (0)$.
$\Rightarrow \exists d_{ij} \in D$ s.t. $d_{ij} \neq 0$. W.l.o.g. assume $d_{11} \neq 0$.
If $\exists \vec{r}$ s.t. $A(B\vec{r})) = C\vec{r}$ then $D\vec{r} = 0$.
$D\vec{r} = 0 \Rightarrow \sum_{j=1}^{n} d_{1j} r_j = 0$, but as $d_{11} \neq 0$ then

$$r_1 = \frac{-\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}.$$

Second trick: Choose $\vec{r} = (r_1, \ldots, r_n)$ from r_n to r_1 and stop at r_2, just before choosing r_1, which could be only 0 or 1.
Then $r_1 = \frac{-\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}$ the equality holds with prob. $= 1/2$

Notice that by considering $r_n, \ldots, r_2$ to be fixed, we reduce the sample space to $r_1 \in \{0, 1\}$

Error probability: Proof

Using the law of Total Probability:

$$\mathbf{Pr}\left[A(B\vec{r}) = C\vec{r}\right] = \sum_{(x_2,\ldots,x_n)\in\{0,1\}^{n-1}} \mathbf{Pr}\left[(A(B\vec{r}) = C\vec{r}) \cap (\underbrace{(r_2,\ldots,r_n) = (x_2,\ldots,x_n)}_{\text{event } E})\right]$$

$$\leq \sum_{(x_2,\ldots,x_n)} \mathbf{Pr}\left[\left(r_1 = \frac{-\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}\right) \cap E\right] \text{ (independence of } x_1\text{)}$$

$$= \sum_{(x_2,\ldots,x_n)} \mathbf{Pr}\left[r_1 = \frac{-\sum_{j=2}^{n} d_{1j} r_j}{d_{11}}\right] \times \mathbf{Pr}\left[E\right]$$

$$\leq \sum_{(x_2,\ldots,x_n)} \frac{1}{2} \times \mathbf{Pr}\left[(r_2,\ldots,r_n) = (x_2,\ldots,x_n)\right]$$

$$= \frac{1}{2}\left(\sum_{(x_2,\ldots,x_n)} \underbrace{\mathbf{Pr}\left[(r_2,\ldots,r_n) = (x_2,\ldots,x_n)\right]}_{=1/2^{n-1}}\right) = \frac{1}{2}\frac{2^{n-1}}{2^{n-1}} = \frac{1}{2}$$

$\square$

Randomize algorithms and amplification

Notice Freivald's algorithm finish always in finite time ($\Theta(n^2)$) but may output the wrong answer. That type of randomized algorithms are called Monte-Carlo algorithms.
Moreover Freivald also is a one side error, if $AB = C$ we always get the correct answer, but if $AB \neq C$ we may get the wrong answer with a "small" probability.

One-side Monte-Carlo algorithms have the nice characteristic that can be amplified: Each run of the algorithm can be considered as an independent "experiment", so they can be repeated, at each run we generate a new random choice, and by independence, each run decreases the probability of error.

If we repeat k times Freivald's algo. and each time we generate a new $\vec{v}$, the answer keep being true, the probability of error is $\leq 1/2^k$.

Fingerprinting technique

Freivalds algorithm is an example of algorithmic fingerprinting technique, we do not want to compute, but just to check.
Suppose we want to compare two items, A_1 and A_2, instead of comparing them directly, we compute random fingerprints $\phi(A_1)$ and $\phi(A_2)$ and compare these.
We require that the fingerprint function $\phi()$ has the following properties:

- If $A_1 = A_2$ then whp $\mathbf{Pr}[\phi(A_1) = \phi(A_2)]$.
- If $A_1 \neq A_2$ then $\mathbf{Pr}[\phi(A_1) = \phi(A_2)]$.
- It is a lot more efficient to compute and compare $\phi(A_1)$ and $\phi(A_2)$, than directly computing and comparing A_1 and A_2.

Notice that for Freivalds' algorithm, if A is $n \times n$ matrix, then $\phi(A) = \vec{r}A$, for $\vec{r}$ a random n-dimensional Boolean vector.

Database consistency

Alice and Bob are in different continents. Each has a copy of a huge database with n bits.
Alice maintain its large N-bit database $X = \{x_{N-1}, \ldots, x_0\}$ of information, while Bob maintains a second copy $Y = \{x_{N-1}, \ldots, x_0\}$ of the same database.

Periodically they want to check consistency of their copies, i.e., to check that both are the same.

Alice could send X to Bob, and he could compare it to Y. But this requires transmission of n bits, which is costly and error-prone.

Instead, suppose Alice first computes a much smaller fingerprint $\phi(X)$ and sends this to Bob. He then computes $\phi(Y)$ and compares it with $\phi(X)$. If the fingerprints are equal, he announces that the copies are identical.

What kind of fingerprint function should we use here?
How many bits do we need to send?

Which is the error in the fingerprint test?

Review of Algebra 1

Given $a, b, n \in \mathbb{Z}$, a congruent with b modulo n ($a \equiv b \mod n$) if $n|(a-b)$.

1. $a \mod n = b \Rightarrow a \equiv b \mod n$.
2. $(a+b) \mod n \equiv ((a \mod n) + (b \mod n)) \mod n$.
3. $(a \cdot b) \mod n \equiv ((a \mod n) \cdot (b \mod n)) \mod n$.
4. $a + (b+c) \equiv (a+b) + c \mod n$ (associativity)
5. $ab \equiv ba \mod n$ (commutativity)
6. $a(b+c) \equiv ab + ac \mod n$ (distributivity)

n partition $\mathbb{Z}$ in n equivalence classes: $\mathbb{Z}_n = \{0, 1 \dots, n-1\}$.
For any $m \in \mathbb{Z}$, $m \mod n \in \mathbb{Z}_n$.

Define $\mathbb{Z}_n^+ = \{1 \dots, n-1\}$.
$(\mathbb{Z}_n, +_n, \cdot_n)$ form a commutative ring,

Review of Algebra 2

Theorem (Prime number Theorem)

Let $n \in \mathbb{Z}$ and let $\pi(n)$ be the number of primes $\leq n$, then

$$\pi(n) \sim \frac{n}{\ln n}, \text{ as } n \to \infty.$$

The frequency of primes slowly decay as the integers increase in length.

For ex. if $n = 10^4$, $\pi(n) = 1929$ and $\frac{n}{\ln n} = 1086$,
while, if $n = 10^7$, $\pi(n) = 664579$ and $\frac{n}{\ln n} = 620420$.

Review of Algebra 3

Lemma: If $n \in \mathbb{Z}$ has N-bits, then $n \leq 2^N$, and at most N different primes can divide n.
(As prime numbers are ≥ 2, the $\#$ of distinct primes that divide n is $\leq N$, because if we multiply together more than n numbers that are at least 2, then we get a number greater than 2^n)

For ex. if $n = 33$, $(33_2 = 100001)$ $N = 6$ and $33 < 2^6 = 64$. So, $\pi(33) = 11$ of which 2 divide 33 $(2 < 6)$

Corollary: Let p_i be the i-th. prime number, then the value of $p_i \sim i \ln i$

For ex. if $i = 1000$, then $p_i \sim 1000 \ln(1000) = 6907$ and the exact value is $p_{1000} = 7919$

Solution to the database consistency problem

If Alice (A) has X and Bob (B) has Y, they use the following algorithm to check they are the same:

- Interpret the data as N-bit integers: $\mathbf{x} = \sum_{i=0}^{N-1} x_i 2^i$ and $\mathbf{y} = \sum_{i=0}^{N-1} x_i 2^i$.
- A Choose u.a.r. a prime $p \in [2, 3, 5, \dots, m]$, for suitable $m = cN \ln N$. (The number of primes in 2^N is N)
- A computes $\phi(\mathbf{x}) = \mathbf{x} \mod p$ and sends the result together with the value p to B.
- B computes a fingerprint $\phi(\mathbf{y}) = \mathbf{y} \mod p$ and compares with the quantity he got from A.
- If $\phi(\mathbf{x}) \neq \phi(\mathbf{y})$ for sure $X \neq Y$, but it is possible $\phi(\mathbf{x}) = \phi(\mathbf{y})$ and $X \neq Y$. (This happens if $\mathbf{x} \mod p = \mathbf{y} \mod p$, with $\mathbf{x} \neq \mathbf{y}$).

Bounding the probability of error

By the Prime Number Theorem $\pi(m) \sim \frac{m}{\ln m}$, so as we see below, we need to take $m = cN \ln N$, for constant $c > 1$.

We want to bound the probability that $\mathbf{x} \neq \mathbf{y}$ but $\phi(\mathbf{x}) = \phi(\mathbf{y})$, i.e.,

$$\begin{aligned}
\mathbf{Pr}\left[\mathbf{x} \bmod p = \mathbf{y} \bmod p | \mathbf{x} \neq \mathbf{y}\right] &= \mathbf{Pr}\left[p \mid |\mathbf{x} - \mathbf{y}|\right] \\
&= \mathbf{Pr}\left[\frac{\#\text{ of primes dividing } |\mathbf{x} - \mathbf{y}|}{\#\text{ primes } \leq m}\right] \\
&\leq \frac{N}{m/\ln m} = \frac{N \ln m}{cN \ln N} = \frac{\ln m}{c \ln N} \\
&= \frac{\ln(cN \ln N)}{c \ln N} = \frac{\ln N + \ln(c \ln N)}{c \ln N} \\
&= \frac{1}{c} + \frac{\ln(c \ln N)}{c \ln N} = \frac{1}{c} + o(1)
\end{aligned}$$

Lemma: Taking $c = 1/\epsilon$ for chosen $0 < \epsilon < 1$, the algorithm achieves an error probability of $\leq \epsilon$.

Choosing a large $m \Rightarrow$, i.e. a large c, we have a larger selection for p, so it is less likely that p divides $|\mathbf{x} - \mathbf{y}|$.

Communication bits

Lemma: The fingerprint algorithm to check the consistency of two databases uses $O(\lg N)$ bits of communication.
Proof: A sends to B p and $\mathbf{x} \bmod p$, which are $\leq m$. Since $m = cN \ln N$, then m is of size $\lg(cN \ln N) = \lg N + \lg(c \ln N) \sim O(lgN)$ bits, so the number of transmitted bits is $O(\lg n)$. $\square$

We proved that by using a more efficient representation of the data (modular), the randomized fingerprinting algorithm gives an exponential decrease in the amount of communication at a small cost in correctness.

How to pick a random prime number

Problem: Given an integer N we want to pick a random prime $p \in [2, \ldots, 2^N - 1]$.

Recall: if n has N bits $\Rightarrow n \leq 2^N - 1$ and $N \geq \lg n$.

Assume we have an efficient algorithm **Prime?** which tell us if an integer is a prime, or not.

Define the set $P = \{p \,|\, 1 < p \leq 2^N - 1, \text{ and } p \text{ is prime}\}$.

We want to pick u.a.r. $p \in P$ (i.e., with probability $\frac{1}{|P|}$)

Pickprime(p)
for $i = 0$ to t **do**
  $p =$ **Rand** $(2^N - 1)$
  if Prime?(p) $= T$ **then**
    return p
    STOP
  end if
end for

t will be fixed later

First analyze one iteration of the algorithm

After we analyze the probability of error after amplifying t times.

Analysis of the algorithm

Let A be the event that a random generate N-bit integer is a prime in P:

$$\mathbf{Pr}\left[A\right] = \frac{|P|}{2^N} = \frac{(2^N / \ln 2^N)}{2^N} = \frac{1}{N \ln 2} = \frac{1.442}{N}.$$

If $N = 2000$ then $\mathbf{Pr}\left[A\right] = 0.000721$, therefore the probability of failing is $\mathbf{Pr}\left[\bar{A}\right] = 0.999271$. Quite high !

Taking into consideration the t-amplification,

$$\mathbf{Pr}\left[\text{Failure after } t \text{ repetitions}\right] = (1 - \frac{1.442}{N})^t \leq e^{-\frac{1.442t}{N}},$$

so taking $t = 10N$ suffices to make small the prob. of failure.

Analysis of the algorithm: Numerical example

If $N = 2000$ taking $t = 10N = 20000$ yields $\mathbf{Pr}\left[\text{Failure}\right] = 0.00004539$ and $\mathbf{Pr}\left[\text{Success}\right] = 0.999955$. If $t = N = 2000$, $\mathbf{Pr}\left[\text{Success}\right] = 0.76425$.

In practice, most of the algorithms to generate a large prime, follows the previous scheme (see for ex. *https://asecuritysite.com/encryption/random3*)

The Primality problem

INPUT: $n \in \mathbb{N}$. QUESTION: Decide if n is prime.

Naive algorithm:

```
Is n ∈ ℕ prime?
for  a = 2, 3, ..., √n do
    if  a | n then
        return  "composite", and STOP
    end if
end for
return  prime and STOP
```

Recall that in arithmetic complexity, for large n ($n = 2^{2024}$), the input size is the number of bits N to express n
i.e., $n = 2^N$ and $N = \lg n$

Complexity of the algorithm: $T(N) = O(2^{N/2}N^2)$ Too slow!

Randomized algorithms for Primality Testing

Theorem (Fermat's Little Th.,XVII)

If n is prime, then for all $a \in \mathbb{Z}_n^+$, $a^{n-1} \equiv 1 \mod n$.

Fermat only works in one direction:
BUT $\exists n \in \mathbb{Z}$ s.t. for all a, $a^{n-1} \equiv 1 \mod n$ with n NOT prime.

The Carmichael numbers: Given a $n \in \mathbb{Z}$, n is a Carmichael number if for all $a \in \mathbb{Z}_n^*$ $a^{n-1} \equiv 1 \bmod n$.

Carmichael numbers are very rare (255 with value < 100000000)
561, 1105, 1729, $\cdots$
For example $561 = 3 \times 11 \times 17$

Test of pseudo-primality

(Assuming the non-existence of Carmichael numbers)

For any $n \in \mathbb{Z}$, n is a pseudo-prime if n is composite and $\forall a \in \mathbb{Z}_n^+$, $a^{n-1} \equiv 1 \mod n$.

Is $n \in \mathbb{N}$ prime?
$a :=$ **rand** $(1, n-1)$
if $a^{n-1} \equiv 1 \mod n$ **then**
  return pseudo-prime
else
  return not-prime
end if

Complexitat: $O(N^3)$.

Error probability

If we assume the non existence of Carmichael numbers:
if n is prime, the previous Monte-Carlo algorithm always give the correct answer, but if n is composite it errs with probability $\leq 1/2$.

The previous algorithm has one-side error, therefore amplifying t times the algorithm, the probability of error goes down to $\leq 1/2^t$.

```
Reapeated-Fermat n, t
for  i = 1 to t  do
    a_i = rand (1, n − 1)
    if  a^{n−1} ≢ 1  mod n then
        return  non-prime and STOP
    else
        return  prime
    end if
end for
```

Taking into consideration Carmichel numbers

Sketch of a Monte-Carlo algorithm for deciding of a given n is a prime: G. Miller (1976), M. Rabin (1980)

- If equation $x^2 \equiv 1 \mod n$ has exactly solutions $x = \pm 1$ that implies n is prime.
- If there is another solution different than ± 1, then n can not be prime.
- To see if n is prime: Randomly choose an integer $a < n$, if $a^2 \equiv 1 \mod n$, then a is a non-trivial root of $1 \mod n$, so n is not prime. Such an a is denoted a witness to the compositeness of n. Otherwise, n may be a prime.

The error of the resulting Monte-Carlo algorithm is $1/2^t$ and the complexity is $O(tN^3)$.

Deciding primality

- For a long time it was open to prove that primality$\in$ P. In 2002, Agrawal, Kayal, Saxena, (AKS) gave a deterministic polynomial time algorithm for Primality.
- If $n \leq 2^N$ the best implementation for the AKS is $\tilde{O}(N^6) = O(N \lg N)$.
- AKS has terrible running time, and it is not clear that it can be improved in the near future.
- From the computational point the Miller-Rabin's algorithm is the basis for existing efficient algorithms.
- However, the Fermat pseudo-primality test can also work fairly nicely, (if we are dealing with $N = 9$, the probability of hitting a Carmichel number is 0.000000255.