CS 441: Applications of Number Theory

PhD. Nils Murrugarra-Llerena
nem177@pitt.edu

University of Pittsburgh

Today's topics

- Hashing non-numeric data
  - Horner’s method for efficient computation

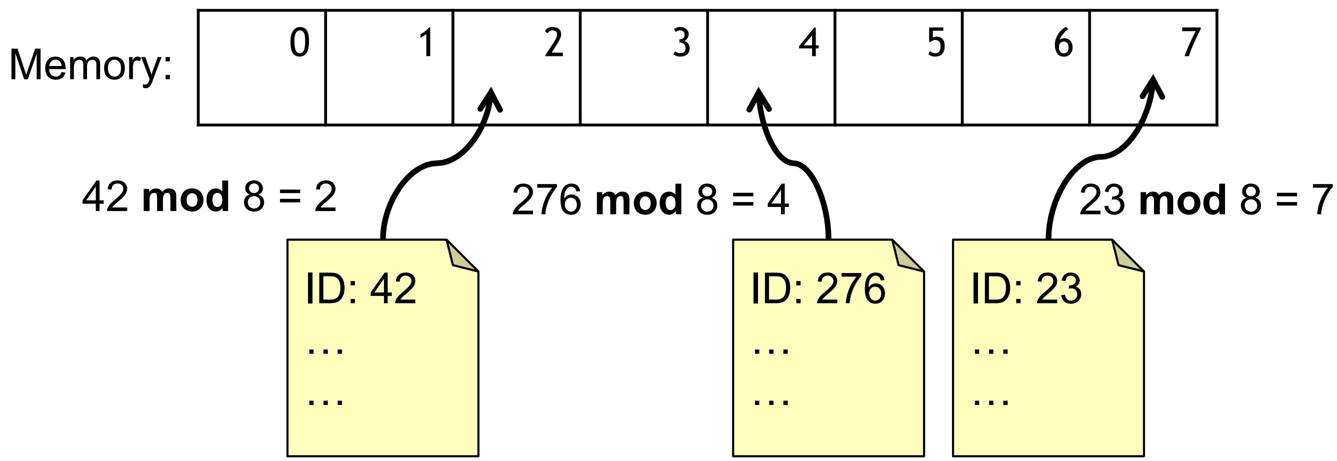
- Check digits and error-correcting codes
  - ISBN, Luhn, Reed–Solomon

- Cryptography
  - Block ciphers
  - Public-key cryptography
  - RSA

Hash functions, recap

Problem: Given a large collection of records, how can we find the one we want quickly?

Solution: Apply a hash function that determines the storage location of the record based on the record's ID. A common hash function is $h(k) = k \textbf{ mod } n$, where n is the number of available storage locations.



What if we want to use non-numeric IDs?

For example, say your IDs are alphanumeric strings

Thanks to base-b expansion, we can interpret these as integers!

- Let $b = 36$, with digits 0–9 then A–Z
- If case sensitive, $b = 62$ to include 0–9, A–Z, a–z
- Base64 is a standard where $b = 64$ using A–Z, a–z, 0–9, +, /
  - Why might this be preferred over $b = 62$?

We can even hash arbitrary binary data

- Any binary data can be interpreted as a base-b integer! Let $b = 2$
- Or, we can read k bits at a time (say, 1 byte = 8 bits) and let $b = 2^k$, interpreting each "block" as an integer in $\mathbf{Z}_b$

To calculate these more efficiently, we can use Horner's method

An k-digit string in base b:

$$a_{k-1}b^{k-1} + a_{k-2}b^{k-2} + \cdots + a_1 b^1 + a_0 b^0$$

- If k and b are large, values like b^{k-1} are very time-consuming to calculate

Instead, we can use Horner's method:

$$((\ldots(a_{k-1} * b + a_{k-2}) * b + \cdots + a_2) * b + a_1) * b + a_0$$

procedure *Horner*(b, a_0, a_1, …, a_{k-1})

$y := a_{k-1}$

for i : k–2 **to** 0

$y := y * b + a_i$

return y

Each a_i is multiplied by b a total of i times

We can add "mod n" here if using this for hashing!

Congruences are also used for check bits and check digits

The textbook describes parity bits and UPC/ISBN check digits

The Luhn algorithm is used for credit cards

- Calculates the 16th digit based on the first 15
- Double the rightmost digit and every second digit, moving right to left
- Let s be the sum of the resulting digits
- Set the check digit to $10 - (s \bmod 10)$

1234 5678 4321 555

2 6 10 14 8 4 10 10

2 4 6 8 3 1 5

- $s = 2 + 2 + 6 + 4 + (1 + 0) + 6 + (1 + 4) + 8 + 8 + 3 + 4 + 1 + (1 + 0) + 5 + (1 + 0) = 57$

Error-correcting codes are used to store data when reading is error-prone

Examples: Optical media, QR codes

Goal: Store k digits* with extra check digits so that erasures/errors can be detected/corrected

- Store $n > k$ digits, where $t = n - k$ are check digits
- If any t digits are lost, they can be recovered
- If any t digits are changed, it can be detected
- If any $\lfloor t/2 \rfloor$ digits are changed, it can be corrected

Reed–Solomon: Let each digit represent a point on a curve

- Any k points identify a $(k-1)$-degree polynomial
- Extend the curve to generate check digits, **interpolate to recover lost digits**

Recall the Caesar cipher

To encode a message using the Caesar cipher:

- Choose a shift index *s*
- Convert each letter A-Z into a number 0-25
- Compute $f(p) = (p + s)\ \mathbf{mod}\ 26$

Example: Let *s* = 9. Encode "ATTACK".

- ATTACK = 0 19 19 0 2 10
- *f*(0) = 9, *f*(19) = 2, *f*(2) = 11, *f*(10) = 19
- Encrypted message: 9 2 2 9 11 19 = JCCJLT

Affine ciphers use bijections of the form $f(p) = (ap + b)\ \mathbf{mod}\ 26$

- To be a bijection, requires $\gcd(a, 26) = 1$

These simple ciphers are not secure!

Patterns become obvious even though the letters are replaced

- Frequencies of letters, digraphs, trigraphs, etc.

However, modern secure block ciphers might look similar at first!

- Replace each k-bit block with another k-bit block
  - Say, $k = 128$, as with Advanced Encryption Standard (AES)
- Use a key (e.g., also 128 bits) to determine the substitution
- There are 2^k possible blocks, similar number of keys, so there are way more combinations
- Even so, to be used securely, a block cipher needs to be combined with a secure mode of operation (block mode)
- A secure block mode ensures that the same block encrypts differently depending on its location

These ciphers, including modern block ciphers, are examples of secret-key cryptography

Symmetric-key (or private-key, or secret-key) cryptography uses the same key to encrypt and decrypt

- The ability to encrypt is the same as the ability to decrypt
- This means we need to share a key with each other party we want to communicate with!

Public-key cryptography: The encryption and decryption key are distinct

- The two keys are paired, but the decryption key is hard to get from the encryption key
- I can generate a key pair, **keep the decryption key**, and **distribute the encryption key** to multiple parties!

The RSA cryptosystem

To generate an RSA key pair:

- Choose two primes, p and q, and let $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$ 2
  - The count of integers less than n that are coprime with n
- Choose an integer e such that is coprime with $\varphi(n)$
  - How can we ensure this?
- Calculate d such that $ed \equiv 1 \pmod{\varphi(n)}$ 1
  - How can we do this?
- Let (n, e) and be the public key
- Let d be the private key

To use an RSA key pair:

- Encrypt a message $m \in \mathbf{Z}_n$ to public key (n, e): $c = m^e \ \mathbf{mod}\ n$
- Decrypt a ciphertext to (n, e): $r = c^d \ \mathbf{mod}\ n = m^{ed} \ \mathbf{mod}\ n = m$

3

Why does RSA decryption work?

Why does $m^{ed}\ \mathbf{mod}\ n = m$? 2 1

- Recall: $ed \equiv 1 \pmod{\varphi(n)}$, and $\varphi(n) = (p-1)(q-1)$
- 3 $r = m^{ed} = m^{k\varphi(n)+1} = m^{k(p-1)(q-1)+1}$
- 4 By Fermat's Little Theorem, this means $r \equiv m \pmod{p}$ and $r \equiv m \pmod{q}$
- Thus, $p \mid (r-m)$ and $q \mid (r-m)$
- Since p and q are distinct and both prime, they are coprime
- Lemma: If $a \mid c$ and $b \mid c$ for coprime a and b, then $ab \mid c$
    - Good practice proof!
    - If $p \mid (r-m)$ and $q \mid (r-m)$ then $\mathrm{p} * q \mid (r-m)$
        - $n \mid (r-m)$
- This means that $n \mid (r-m)$, so $r \equiv m \pmod{n}$

Why does RSA decryption work?: Fermat's Little Theorem

- $r = m^{ed} = m^{k\varphi(n)+1} = m^{k(p-1)(q-1)+1}$
- 4 By Fermat's Little Theorem, this means $r \equiv m \pmod{p}$ and $r \equiv m \pmod{q}$
  - Let's assume gcd(m, p) = 1 and gcd(m,q) = 1. m and p are coprimes; m and q are coprimes [See slide 22 from congruences]
  - Since p is prime, and m is not divisible by p [gcd(m, p) = 1] → $m^{(p-1)}$ = 1 (mod p)
  - Since q is prime, and m is not divisible by p [gcd(m, q) = 1] → $m^{(q-1)}$ = 1 (mod q)
  - $r = m^{k(p-1)(q-1)+1} = m * m^{k(p-1)(q-1)}$
    $= m * 1\ (mod\ p)$
    $= m\ (mod\ p)$
  - $r = m^{k(p-1)(q-1)+1} = m * m^{k(p-1)(q-1)}$
    $= m * 1\ (mod\ q)$
    $= m\ (mod\ q)$
- Thus, $p \mid (r - m)$ and $q \mid (r - m)$

A brief practical security note…

The textbook includes examples and exercises where they encrypt a message using RSA, one “chunk” at a time

THIS IS INSECURE, DO NOT DO THIS

Remember what we learned from block ciphers!

- Done this way, if the plaintext chunk is the same, the ciphertext chunk is the same
- Even if the encryption approach is very sophisticated in isolation, encrypting piece-by-piece reveals patterns
- Best practice is to use RSA to encrypt a single-use symmetric key, then encrypt the message using a block cipher with a secure mode of operation
  - In part, because block ciphers are way faster than RSA

Why is RSA secure?

That is, if you know $c = m^e \textbf{ mod } n$, why can't you get m?

This is called the RSA problem, and the fastest known approach is to factor n

- In turn, the fastest factoring algorithm is slower than polynomial complexity ("hard")
- Factoring n reveals p and q, and thus $\varphi(n)$, and then d can be computed from e just like in key generation
- If you can get d, then you can get p and q
  - By contrapositive, if factoring is hard, then getting d is hard
- Similarly, if you can get $\varphi(n)$, you can get p and q

To be secure for the near future, n should be 2048 bits in size

- e.g., 28980031691694357068918562487659336178577290872139729240999721884150682654823846774504439389267921793843771740233811602035640310196929500591908624781 66152016032673099683618999980615311782821864256646973478297214481647222660269569400841134169754396451340590101145507012183878091040551030992366712077 51888612680781200445138803757546069773284441936327610981983867727670435168737551110881172718728253861892500326058954623805626985122349587194747221280 36031389620442812631321984742581817025098263901240154322179135628982031399236433383170589170534724928725807887253791412053381878561858347628938989347 5235786179508298 46264

Final thoughts

- Number theory has many applications in computing
  - Hashing for storage
  - Check digits and error-correcting codes
  - Cryptography
- Symmetric-key cryptography relies on complex substitutions, while public-key cryptography uses number theory
  - … and mathematical problems with no known efficient algorithms
- Next: Proof by induction! (Start reading Chapter 5)