Algorithm Design and Analysis

Hashing: Fingerprinting (String Matching)

Roadmap for today

- Learn some properties about ***random primes*** and how to pick one
- Design random hashing schemes for ***strings***
- Apply this idea (which we call ***Fingerprinting***) to ***string matching***

Formal model of computation

Model (word-RAM):

- We have unlimited constant-time addressable memory (“registers”)
- Each register can store a w-bit integer (a “word”)
- Reading/writing, arithmetic, logic, bitwise operations on a constant number of words takes constant time
- With input size n, we need $w \geq \log n$.

Random prime numbers

How do we pick a random prime?

- Suppose we want to pick a random prime in the range $\{0, \dots, M-1\}$

Algorithm (rejection sampling):

Pick a random integer x in the range $\{2, \dots, M-1\}$
Check if x is prime. Is so output it, else go back to the first step

- Two important follow-up questions:
  - How do we do step 2 (check if x is prime)?
  - How many iterations will this algorithm take (in expectation)?

How to check if x is a prime?

Simple trial division:

- Try every integer greater than 2, less than x, and check if it divides x
- Takes $O(x)$ iterations. That's a lot for large values of x

Better trial division:

- Try every integer greater than 2, at most $\sqrt{x}$, and check if it divides x
- Takes $O(\sqrt{x})$ iterations. That's much better

Even better (but not covered in this class)

- Miller-Rabin algorithm. Takes $O(\text{polylog } x)$ time and is randomized.
- AKS algorithm. $O(\text{polylog } x)$ time, but a higher exponent and deterministic!

How many iterations of sampling?

- This is asking about the ***density of primes***
- Let $\pi(n)$ be the number of primes between 1 and n

Prime Number Theorem:

$$\pi(n) \sim \frac{n}{\ln n}$$

The $\sim$ notation means "is asymptotic to":

$$\lim_{n\to\infty}\left(\frac{\pi(n)}{n/\ln n}\right) = 1$$

Tighter bounds

Chebyshev's Theorem:

$$\pi(n) \geq \frac{7}{8}\frac{n}{\ln n} > \frac{n}{\ln n} \qquad \text{for all } n \geq 2$$

Dusart's Theorem:

$$\frac{n}{\ln n - 1.1} \leq \pi(n) \leq \frac{n}{\ln n - 1} \qquad \text{for all } n \geq 60184$$

So, how many iterations of sampling?

- Since $\pi(n) \geq \frac{n}{\ln n}$ (Chebyshev), this means that

$$\Pr[\text{Random number in}\{2, \ldots, M-1\} \text{ is prime}] \geq \frac{\frac{M}{\ln M}}{M} = \frac{1}{\ln M}$$

- So, we should expect our rejection sampling algorithm to take about **$\ln M$ iterations**.

Super useful corollary: If we want there to be at least k possible primes, then we should pick a random prime from $\{2, \ldots, M\}$ where $M \geq 2k \lg k$

The String Equality Problem

The String Equality Problem

Alice

x

Is $x = y$?

Bob

y

- x and y are n-bit strings (i.e., written in binary, 0 and 1)
- Alice and Bob want to exchange messages to decide if $x = y$
- **Simplest solution**: Alice sends x to Bob and Bob checks if $x = y$

Probabilistic approach

- A simple information-theory argument shows that no scheme can do better than just sending n bits (there are 2^n possible strings, so we must communicate n bits to be able to distinguish them).
- We can relax our requirement and aim for a probabilistic guarantee!
  - If $x = y$ then $\Pr[\text{Bob says } \mathbf{equal}] = 1$
  - If $x \neq y$ then $\Pr[\text{Bob says } \mathbf{equal}] \leq \delta$ for a very small delta
- E.g., if we pick $\delta = 0.01$, we are saying we are okay with a 1% probability of a false positive. We never allow false negatives.

A probabilistic algorithm

- Alice picks a ***random prime number*** p in $\{2, \ldots, M\}$ for some value of M
- Alice computes the hash value

$$\mathrm{h}_p(x) = x \mod p$$

- Alice sends p and the hash value $\mathrm{h}_p(x)$ to Bob
- Bob checks if $\mathrm{h}_p(\mathrm{x}) = \mathrm{h}_p(y)$, and if so, says **equal**, else says **not equal**

(remember that x is a string of n zeros and ones, so we can interpret it as an n-bit integer written in binary, in $\{0, \ldots, 2^n - 1\}$)

Analysis

- If $x = y$ then Bob always says equal, so no false negatives
- Let's pick $M = 200n \log(100n)$ *(our super useful corollary from before says that this gives us* $100n$ *possible primes)*
- When do we get a false positive? Suppose Bob says **equal**, then…

$x \equiv y \mod p \Rightarrow |x-y| \equiv 0 \mod p$

$\Rightarrow p$ is a divisor of $|x-y|$

Analysis

- How many prime divisors can $|x - y|$ have? *(remember they are n bits long)*

$\leq n$ (worst case $= 2 \cdot 2 \cdot 2 \cdots 2 = 2^n$)

- p is a random prime among $\{2, \ldots, 200n \log(100n)\}$, among which there are $100n$ primes, so…

$$\Pr[\text{False positive}] = \Pr[p \text{ divides } |x - y|]$$

$$= \frac{\#\text{divisors (bad)}}{\#\text{possible primes}} \leq \frac{n}{100n} = 1\%$$

Reducing the error probability

- We picked p as a random prime among $\{2, \ldots, 200n\log(100n)\}$, which gave us an error probability of $\frac{1}{100}$

In general, to get a probability of $\frac{1}{s} = \delta$, we pick p from the range…

$\{2, 2sn\log(sn)\}$

Cost analysis

- The naïve solution (send the whole n bits to Bob) requires sending n bits. How much better is the probabilistic solution?
- Alice must send the prime and the hash, which are both integers in the range $\{2, \dots, M\}$, so this is $O(\log M)$ bits
- Remember $M = 2sn \log (sn)$

$$2 \log_2 (2sn \log sn)$$

$$= O(\log s + \log n) \text{ bits!}$$

If $s = O(\text{poly } n) \longrightarrow O(\log n)$ bits!

The String-Matching Problem

The String-Matching Problem

Problem (String Matching): Given a text string T of n bits and a pattern P of m bits, output all positions in T where the substring P occurs

- For example, $P = 100$,

Naive alg = $O(nm)$

$$T = 10100110011100$$

Key idea (Karp-Rabin algorithm): Compute the hash of the pattern and compare the hash to the hash of every length-m substring.

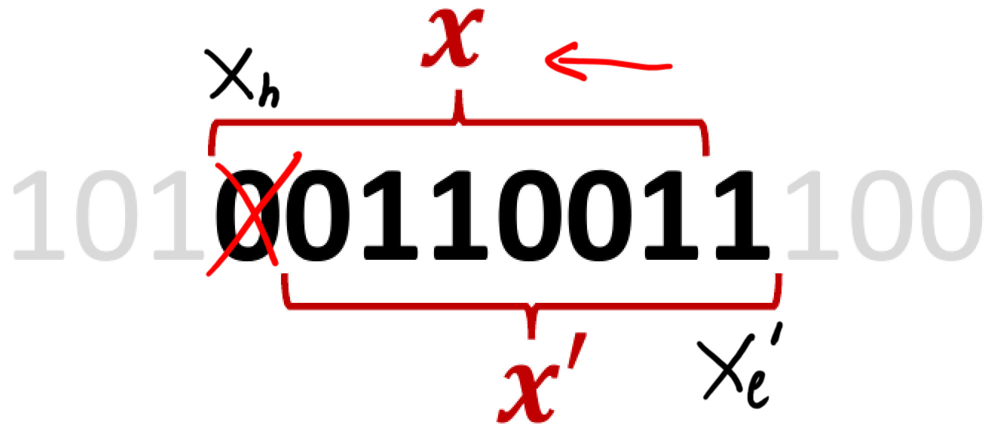
How to make it fast?

- There are $O(n)$ substrings, and computing their hash takes $O(m)$ time each, so this is $O(nm)$ time. The same as just manually comparing each substring!
- How can we make it faster?

h

$$T = 10100110011100$$

"Rolling" the hash function



To go from x to x': (mod p)

- Remove the high-order bit by **subtracting $x_h \cdot 2^{m-1}$**
- Shift every remaining bit one position by **multiplying by 2**
- Append the new low-order bit by **adding x_l'**

“Rolling” the hash function

- So, we can write x' in terms of x as

$$x' = 2(x - x_h \cdot 2^{m-1}) + x_l' \quad \text{mod } p$$

- Therefore, we can write $h_p(x')$ in terms of $h_p(x)$ as…

$$h_p(x') = \Big(2\, \underbrace{h_p(x)}_{\text{old hash}} - x_h \cdot \underbrace{h_p(2^m)}_{\text{precompute}} + x_l' \Big) \text{ mod } p$$

- This is just a constant number of arithmetic operations mod p!

The Karp-Rabin Algorithm

1. Pick a random prime p in $\{2, \ldots, M\}$ for $M = 2sm \log_2 (sm)$
2. Compute $h_p(P)$ and $h_p(2^m)$
3. Compute $h_p(T_{0 \ldots m-1})$ and check if it matches $h_p(P)$
   If so, output **match at position 0**
4. For each $i \in \{0, \ldots, n-m-1\}$
   i. Compute $h_p\left(T_{i+1, \ldots, i+m}\right)$ using $h_p(T_{i \ldots i+m-1})$ as per the previous slide
   ii. If $h_p(T_{i+1 \ldots i+m}) = h_p(P)$, output **match at position $\boldsymbol{i} + 1$**

Error analysis

Theorem: We can achieve an error probability of δ using Karp-Rabin with a prime p that is $O\left(\log\frac{1}{\delta} + \log m + \log n\right)$ bits.

$n-m+1$

- We do [$\leq n$] comparisons, each with a probability [$1/s$] of failure
- By a ***union bound***, the probability of encountering at least one failure is

[n/s]

$\log(\ell) = O\left(\log\frac{1}{\delta} + \log n + \log m\right)$

- So, we pick a prime from the range $M =$ [$2\frac{1}{\delta}nm\log(\frac{1}{\delta}nm)$]

Cost analysis

- We are still working in the word-RAM model
- Since the text string has n characters/bits in it, we have $w \geq \log n$
- Same for the pattern length m, we have $w \geq \log m$
- Say we want polynomial error probability, i.e.,

$$\delta = \frac{1}{O(\text{poly}(n, m))}$$

- Therefore, $\log M = O(\log(\text{poly}(n, m))) = O(\log n + \log m)$
- Since $p < M$, all arithmetic mod p is constant time!

Cost analysis

- Computing $h_p(x)$ for an m-bit x can be done in $O(m)$ time
  - Hashes of powers of 2 can be computed iteratively by multiplying the previous power of 2 by 2 then taking mod p
- So, the initial hashes of Karp-Rabin $\mathrm{h}_p(P)$, $\mathrm{h}_p(2^m)$, $\mathrm{h}_p(T_{0\ldots m-1})$ can be computed in $O(m)$ time
- Rolling from $h_p(T_{i+1\ldots i+m})$ from $h_p(T_{i\ldots i+m-1})$ takes a constant number of arithmetic operations, each of which takes $O(1)$ time!

Total runtime = $O(n+m)$

Summary of fingerprinting

- We rely heavily on randomness, specifically ***picking a random prime***!
- We can check whether two n-bit strings are equal with ***low failure probability*** by comparing a $O(\log n)$-bit hash, which is very cheap compared to $O(n)$ bits!
- Applied to the pattern matching problem, this gives us the ***Karp-Rabin algorithm***, which finds the locations of a pattern in a text in $O(n + m)$ time with small failure probability