CS 473: Algorithms

Ruta Mehta

University of Illinois, Urbana-Champaign

Spring 2021
Fingerprinting

Lecture 11
March 2, 2021

Most slides are courtesy Prof. Chekuri
Bloom Filters

Hashing:
1. To insert $x$ in dictionary store $x$ in table in location $h(x)$
2. To lookup $y$ in dictionary check contents of location $h(y)$

Issue: False positives due to collisions.
Hashing:
1. To insert $x$ in dictionary store $x$ in table in location $h(x)$
2. To lookup $y$ in dictionary check contents of location $h(y)$

Bloom Filter: tradeoff space for false positives
1. What if elements ($x$) are unwieldy objects such as long strings, images, etc with non-uniform sizes.
2. To insert $x$ in dictionary, set bit at location $h(x)$ to 1 (initially all bits are set to 0)
3. To lookup $y$ if bit in location $h(y)$ is 1 say yes, else no.
Bloom Filters

Hashing:
1. To insert \(x\) in dictionary store \(x\) in table in location \(h(x)\)
2. To lookup \(y\) in dictionary check contents of location \(h(y)\)

Bloom Filter: tradeoff space for false positives
1. What if elements \((x)\) are unwieldy objects such as long strings, images, etc with non-uniform sizes.
2. To insert \(x\) in dictionary, set bit at location \(h(x)\) to 1 (initially all bits are set to 0)
3. To lookup \(y\) if bit in location \(h(y)\) is 1 say yes, else no.

Issue: False positives due to collisions.
Bloom Filters

**Bloom Filter:** tradeoff space for false positives

**Reducing false positives:**

1. Pick \( k \) hash functions \( h_1, h_2, \ldots, h_k \) *independently*

2. Insert \( x \): for \( 1 \leq i \leq k \) set bit in location \( h_i(x) \) in table \( i \) to 1
Bloom Filters

Bloom Filter: tradeoff space for false positives

Reducing false positives:

1. Pick $k$ hash functions $h_1, h_2, \ldots, h_k$ independently
2. Insert $x$: for $1 \leq i \leq k$ set bit in location $h_i(x)$ in table $i$ to 1
3. Lookup $y$: compute $h_i(y)$ for $1 \leq i \leq k$ and say yes only if each bit in the corresponding location is 1, otherwise say no. If probability of false positive for one hash function is $\alpha < 1$ then with $k$ independent hash function it is
Bloom Filters

Bloom Filter: tradeoff space for false positives

Reducing false positives:

1. Pick $k$ hash functions $h_1, h_2, \ldots, h_k$ independently
2. Insert $x$: for $1 \leq i \leq k$ set bit in location $h_i(x)$ in table $i$ to 1
3. Lookup $y$: compute $h_i(y)$ for $1 \leq i \leq k$ and say yes only if each bit in the corresponding location is 1, otherwise say no. If probability of false positive for one hash function is $\alpha < 1$ then with $k$ independent hash function it is $\alpha^k$.
Take away points

1. Hashing is a powerful and important technique for dictionaries. Many practical applications.
2. Randomization fundamental to understand hashing.
3. Good and efficient hashing possible in theory and practice with proper definitions (universal, perfect, etc).
4. Related ideas of creating a compact fingerprint/sketch for objects is very powerful in theory and practice.
Process of mapping a large data item to a much shorter bit string, called its fingerprint.

Fingerprints uniquely identifies data “for all practical purposes”.
Process of mapping a large data item to a much shorter bit string, called its fingerprint.

Fingerprints uniquely identifies data “for all practical purposes”.

Typically used to avoid comparison and transmission of bulky data.

Eg: Web browser can store/fetch file fingerprints to check if it is changed.
Fingerprinting


Process of mapping a large data item to a much shorter bit string, called its fingerprint.

Fingerprints uniquely identifies data “for all practical purposes”.

Typically used to avoid comparison and transmission of bulky data.

Eg: Web browser can store/fetch file fingerprints to check if it is changed.

As you may have guessed, fingerprint functions are hash functions.
Use of hash functions for designing fast algorithms

Problem
Given a text $T$ of length $m$ and pattern $P$ of length $n$, $m \gg n$, find all occurrences of $P$ in $T$. 

Karp-Rabin Randomized Algorithm
It involves:
- Sampling a prime
- String equality via mod $p$ arithmetic
- Rabin’s fingerprinting scheme – rolling hash
- Karp-Rabin pattern matching algorithm: $O(m + n)$ time.
Use of hash functions for designing fast algorithms

Problem
Given a text $T$ of length $m$ and pattern $P$ of length $n$, $m \gg n$, find all occurrences of $P$ in $T$.

Karp-Rabin Randomized Algorithm
Use of hash functions for designing fast algorithms

Problem
Given a text $T$ of length $m$ and pattern $P$ of length $n$, $m \gg n$, find all occurrences of $P$ in $T$.

Karp-Rabin Randomized Algorithm
It involves:
- Sampling a prime
- String equality via $mod\ p$ arithmetic
- Rabin’s fingerprinting scheme – rolling hash
- Karp-Rabin pattern matching algorithm: $O(m + n)$ time.
Part I

Sampling a Prime
Problem
Given an integer $x > 0$, sample a prime uniformly at random from all the primes between 1 and $x$. 
Sampling a prime

**Problem**

Given an integer $x > 0$, sample a prime uniformly at random from all the primes between $1$ and $x$.

**Procedure**

1. Sample a number $p$ uniformly at random from $\{1, \ldots, x\}$.
2. If $p$ is a prime, then output $p$. Else go to Step (1).
Sampling a prime

Problem
Given an integer $x > 0$, sample a prime uniformly at random from all the primes between 1 and $x$.

Procedure
1. Sample a number $p$ uniformly at random from $\{1, \ldots, x\}$.
2. If $p$ is a prime, then output $p$. Else go to Step (1).

Checking if $p$ is prime
- Agrawal-Kayal-Saxena primality test: deterministic but slow
- Miller-Rabin randomized primality test: fast but randomized outputs ‘prime’ when it is not with very low probability.
Sampling a Prime: Analysis

Is the returned prime \textit{sampled uniformly at random}?
Sampling a Prime: Analysis

Is the returned prime \textit{sampled uniformly at random}? 

\( \pi(x) \): number of primes in \( \{1, \ldots, x\} \),

Lemma

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = \frac{1}{\pi(x)} \).
Is the returned prime sampled uniformly at random?

\( \pi(x) : \) number of primes in \( \{1, \ldots, x\} \),

**Lemma**

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).

**Proof.**

Event \( A : \) a prime is picked in a round. \( \Pr[A] = \)
Sampling a Prime: Analysis

Is the returned prime \textit{sampled uniformly at random}?

\[ \pi(x) : \text{number of primes in } \{1, \ldots, x\}, \]

Lemma

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).

Proof.

Event \( A \): a prime is picked in a round. \( \Pr[A] = \pi(x)/x \).
Sampling a Prime: Analysis

Is the returned prime sampled uniformly at random?

\( \pi(x) \): number of primes in \( \{1, \ldots, x\} \).

Lemma

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = \frac{1}{\pi(x)} \).

Proof.

Event \( A \): a prime is picked in a round. \( \Pr[A] = \frac{\pi(x)}{x} \).
Event \( B \): number (prime) \( p^* \) is picked. \( \Pr[B] = \)}
Sampling a Prime: Analysis

Is the returned prime sampled uniformly at random?

\( \pi(x) : \) number of primes in \( \{1, \ldots, x\} \),

**Lemma**

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).

**Proof.**

Event \( A \): a prime is picked in a round. \( \Pr[A] = \pi(x)/x \).
Event \( B \): number (prime) \( p^* \) is picked. \( \Pr[B] = 1/x \).
Sampling a Prime: Analysis

Is the returned prime \textit{sampled uniformly at random}?

\[ \pi(x) : \text{number of primes in } \{1, \ldots, x\}, \]

Lemma

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).

Proof.

Event \( A \): a prime is picked in a round. \( \Pr[A] = \pi(x)/x \).

Event \( B \): number (prime) \( p^* \) is picked. \( \Pr[B] = 1/x \).

\[ \Pr[A \cap B] = \]
Sampling a Prime: Analysis

Is the returned prime sampled uniformly at random?

\( \pi(x) \): number of primes in \( \{1, \ldots, x\} \),

**Lemma**

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).

**Proof.**

Event \( A \): a prime is picked in a round. \( \Pr[A] = \pi(x)/x \).

Event \( B \): number (prime) \( p^* \) is picked. \( \Pr[B] = 1/x \).

\( \Pr[A \cap B] = \Pr[B] = 1/x \). Why?
Is the returned prime *sampled uniformly at random*?

\[ \pi(x) : \text{number of primes in } \{1, \ldots, x\}, \]

**Lemma**

For a fixed prime \(p^* \leq x\), \(\Pr[\text{algorithm outputs } p^*] = \frac{1}{\pi(x)} \).

**Proof.**

Event \(A\): a prime is picked in a round. \(\Pr[A] = \frac{\pi(x)}{x}.\)

Event \(B\): number (prime) \(p^*\) is picked. \(\Pr[B] = \frac{1}{x}.\)

\[ \Pr[A \cap B] = \Pr[B] = \frac{1}{x}. \text{ Why? Because } B \subset A. \]
Sampling a Prime: Analysis

Is the returned prime *sampled uniformly at random*?

\( \pi(x) \) : number of primes in \( \{1, \ldots, x \} \).

**Lemma**

*For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).*

**Proof.**

Event \( A \): a prime is picked in a round. \( \Pr[A] = \pi(x)/x \).

Event \( B \): number (prime) \( p^* \) is picked. \( \Pr[B] = 1/x \).

\( \Pr[A \cap B] = \Pr[B] = 1/x \). **Why?** Because \( B \subset A \).

\( \Pr[B|A] = \)
Sampling a Prime: Analysis

Is the returned prime sampled uniformly at random?

\( \pi(x) : \) number of primes in \( \{1, \ldots, x\} \),

**Lemma**

For a fixed prime \( p^* \leq x \), \( \Pr[\text{algorithm outputs } p^*] = 1/\pi(x) \).

**Proof.**

Event \( A \): a prime is picked in a round. \( \Pr[A] = \pi(x)/x \).

Event \( B \): number (prime) \( p^* \) is picked. \( \Pr[B] = 1/x \).

\( \Pr[A \cap B] = \Pr[B] = 1/x \). Why? Because \( B \subset A \).

\[
\Pr[B|A] = \frac{\Pr[A \cap B]}{\Pr[A]} = \frac{\Pr[B]}{\Pr[A]} = \frac{1/x}{\pi(x)/x} = \frac{1}{\pi(x)}
\]
Sampling a prime: Expected number of samples

Procedure

1. Sample a number $p$ uniformly at random from $\{1, \ldots, x\}$.
2. If $p$ is a prime, then output $p$. Else go to Step (1).

Running time in expectation

Q: How many samples in expectation before termination?
A: $\frac{x}{\pi(x)}$. Exercise.
How many primes between 0 and x

\( \pi(x) \): Number of primes between 0 and x.

J. Hadamard and C. J. de la Vallée-Poussin (1896)

Prime Number Theorem: \( \lim_{x \to \infty} \frac{\pi(x)}{x / \ln x} = 1 \)
How many primes between 0 and \( x \)

\( \pi(x) \): Number of primes between 0 and \( x \).

**J. Hadamard and C. J. de la Vallée-Poussin (1896)**

**Prime Number Theorem:**

\[
\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1
\]

**Chebyshev (from 1848)**

\[
\pi(x) \geq \frac{7}{8} \frac{x}{\ln x} = (1.262..) \frac{x}{\lg x} > \frac{x}{\lg x}
\]
How many primes between 0 and $x$

$\pi(x)$: Number of primes between 0 and $x$.

J. Hadamard and C. J. de la Vallée-Poussin (1896)

Prime Number Theorem: $\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1$

Chebyshev (from 1848)

$$\pi(x) \geq \frac{7}{8} \frac{x}{\ln x} = \left(1.262..\right) \frac{x}{\lg x} > \frac{x}{\lg x}$$

• $y \sim \{1, \ldots, x\}$ u.a.r., then $y$ is a prime w.p. $\frac{\pi(x)}{x} > \frac{1}{\lg x}$. 
How many primes between 0 and \( x \)

\( \pi(x) \): Number of primes between 0 and \( x \).

**J. Hadamard and C. J. de la Vallée-Poussin (1896)**

**Prime Number Theorem:**

\[
\lim_{x \to \infty} \frac{\pi(x)}{x / \ln x} = 1
\]

**Chebyshev (from 1848)**

\[
\pi(x) \geq 7 \frac{x}{8 \ln x} = (1.262..) \frac{x}{\lg x} > \frac{x}{\lg x}
\]

- \( y \sim \{1, \ldots, x\} \) u.a.r., then \( y \) is a prime w.p. \( \frac{\pi(x)}{x} > \frac{1}{\lg x} \).
- If we want \( k \geq 4 \) primes then \( x \geq 2k \lg k \) suffices.

\[
\pi(x) \geq \pi(2k \lg k) = \frac{2k \lg k}{\lg 2 + \lg k + \lg \lg k} \geq \frac{k(2 \lg k)}{2 \lg k} = k
\]
Part II

String Equality
String Equality

Problem

Alice, the captain of a Mars lander, receives an N-bit string $x$, and Bob, back at mission control, receives a string $y$. They know nothing about each other's strings, but want to check if $x = y$.
Problem

Alice, the captain of a Mars lander, receives an N-bit string $x$, and Bob, back at mission control, receives a string $y$. They know nothing about each others strings, but want to check if $x = y$.

Alice sends Bob $x$, and Bob confirms if $x = y$. But sending $N$ bits is costly! *Can they share less communication and check equality?*
String Equality

Problem
Alice, the captain of a Mars lander, receives an N-bit string $x$, and Bob, back at mission control, receives a string $y$. They know nothing about each others strings, but want to check if $x = y$.

Alice sends Bob $x$, and Bob confirms if $x = y$. But sending $N$ bits is costly! *Can they share less communication and check equality?*

Possibilities:
- If want 100% surety then NO.
- If OK with 99.99% surety then $O(\lg N)$ may suffice!!!
String Equality

Problem
Alice, the captain of a Mars lander, receives an N-bit string $x$, and Bob, back at mission control, receives a string $y$. They know nothing about each other’s strings, but want to check if $x = y$.

Alice sends Bob $x$, and Bob confirms if $x = y$. But sending $N$ bits is costly! *Can they share less communication and check equality?*

Possibilities:
- If want 100% surety then NO.
- If OK with 99.99% surety then $O(\lg N)$ may suffice!!!
  - If $x = y$, then $\Pr[\text{Bob says equal}] = 1$.
  - If $x \neq y$, then $\Pr[\text{Bob says un-equal}] = 0.9999$. 
String Equality

Problem

Alice, the captain of a Mars lander, receives an N-bit string $x$, and Bob, back at mission control, receives a string $y$. They know nothing about each others strings, but want to check if $x = y$.

Alice sends Bob $x$, and Bob confirms if $x = y$. But sending $N$ bits is costly! *Can they share less communication and check equality?*

Possibilities:

- If want 100% surety then **NO**.
- If OK with 99.99% surety then $O(lg N)$ may suffice!!!
  - If $x = y$, then $Pr[\text{Bob says equal}] = 1$.
  - If $x \neq y$, then $Pr[\text{Bob says un-equal}] = 0.9999$.

**HOW?**
String Equality: Randomized Algorithm

\(x, y\): N-bit strings.
String Equality: Randomized Algorithm

\( \mathbf{x, y} : \mathbf{N}\text{-bit strings.} \)

(Recall) If \( M = \lceil 2(5N) \lg 5N \rceil \), then \( 5N \) primes in \( \{1, \ldots, M\} \).
String Equality: Randomized Algorithm

\( x, y : \) N-bit strings.

(Recall) If \( M = \lceil 2(5N) \lg 5N \rceil \), then \( 5N \) primes in \( \{1, \ldots, M\} \).

**Procedure**

1. Alice picks a random prime \( p \) from \( \{1, \ldots, M\} \).
2. Define \( h_p(x) = x \mod p \).
3. Bob checks if \( h_p(y) = h_p(x) \). If so, he says equal, else unequal.

Lemma

If \( x = y \) then Bob always says equal.
String Equality: Randomized Algorithm

$x, y$: N-bit strings.

(Recall) If $M = \lceil 2(5N) \lg 5N \rceil$, then $5N$ primes in $\{1, \ldots, M\}$.

**Procedure**

1. Alice picks a random prime $p$ from $\{1, \ldots, M\}$.
2. She sends Bob prime $p$, and also $h_p(x) = x \mod p$.
3. Bob checks if $h_p(y) = h_p(x)$. If so, he says equal else un-equal.

**Lemma** If $x = y$ then Bob always says equal.
String Equality: Randomized Algorithm

\( x, y \): N-bit strings.

(Recall) If \( M = \lceil 2(5N) \lg 5N \rceil \), then \( 5N \) primes in \( \{1, \ldots, M\} \).

Procedure

- Define \( h_p(x) = x \mod p \)
- Alice picks a random prime \( p \) from \( \{1, \ldots, M\} \).
- She sends Bob prime \( p \), and also \( h_p(x) = x \mod p \).
- Bob checks if \( h_p(y) = h_p(x) \). If so, he says equal else un-equal.

Lemma

If \( x = y \) then Bob always says equal.
String Equality: Randomized Algorithm

$x, y$ : N-bit strings.

(Recall) If $M = \lceil 2(5^N) \lg 5^N \rceil$, then $5^N$ primes in \{1, \ldots, M\}.

**Procedure**

1. Alice picks a random prime $p$ from \{1, \ldots, $M$\}.
2. She sends Bob prime $p$, and also $h_p(x) = x \mod p$.
3. Bob checks if $h_p(y) = h_p(x)$. If so, he says *equal* else *un-equal*.

**Lemma**

If $x \neq y$ then, $\Pr[Bob \ says \ equal] \leq 1/5$ (error probability).
String Equality: Randomized Algorithm

\(x, y: N\)-bit strings.

(Recall) If \(M = \lceil 2(sN) \log sN \rceil\), then \(sN\) primes in \(\{1, \ldots, M\}\).

Procedure

Define \(h_p(x) = x \mod p\)

1. Alice picks a random prime \(p\) from \(\{1, \ldots, M\}\).
2. She sends Bob prime \(p\), and also \(h_p(x) = x \mod p\).
3. Bob checks if \(h_p(y) = h_p(x)\). If so, he says equal else un-equal.

Lemma

If \(x \neq y\) then, \(\Pr[Bob\ says\ equal] \leq 1/s\) (error probability).
Let $x = 6 = 2 \times 3$. If we draw a $p$ u.a.r. from $\{2, 3, 5, 7\}$, then what is the probability that $x \mod p = 0$?

(A) 0.
(B) 1.
(C) 1/4.
(D) 1/2.
(E) none of the above.

Now, let $y = 21$. What is the probability that $(y - x) \mod p = 15 \mod p = 0$?

(A) 0.
(B) 1.
(C) 1/4.
(D) 1/2.
(E) none of the above.
Question.

Let \( x = 6 = 2 \times 3 \). If we draw a \( p \) u.a.r. from \( \{2, 3, 5, 7\} \), then what is the probability that \( x \mod p = 0 \)?

(A) 0.
(B) 1.
(C) 1/4.
(D) 1/2.
(E) none of the above.

Now, let \( y = 21 \). What is the probability that \( (y - x) \mod p = 15 \mod p = 0 \)?

(A) 0.
(B) 1.
(C) 1/4.
(D) 1/2.
String Equality: Randomized Algorithm

Error probability

Let $x, y$ be N-bit strings, $M = \lceil 2(sN) \log sN \rceil$, and $h_p(x) = x \mod p$.

**Lemma**

If $x \neq y$ then, $\Pr[Bob\ says\ equal] = \Pr[h_p(x) = h_p(y)] \leq 1/s$.

**Proof.**

Given $x \neq y$, $h_p(x) = h_p(y) \implies x \mod p = y \mod p$. 

Probability that a random prime $p$ from $\{1, \ldots, M\}$ is a divisor of $D$ is

$$P = \frac{k \pi(M)}{M} \leq \frac{N \pi(M)}{\log M} \\ \leq \frac{N^2 (sN) \log sN}{\log M} \leq \frac{1}{s}.$$
String Equality: Randomized Algorithm

Error probability

$x, y$ N-bit string, $M = \lceil 2(sN) \lg sN \rceil$, and $h_p(x) = x \mod p$

**Lemma**

*If* $x \neq y$ *then*, $\Pr[Bob\ says\ equal] = \Pr[h_p(x) = h_p(y)] \leq 1/s$

**Proof.**

*Given* $x \neq y$, $h_p(x) = h_p(y)$ $\Rightarrow$ $x \mod p = y \mod p$.

- $D = |x - y|$, then $D \mod p = 0$, and $D \leq 2^N$. 

Ruta (UIUC) CS473 19 Spring 2021 19 / 30
String Equality: Randomized Algorithm

Error probability

\[ x, y \text{ N-bit string, } M = \lceil 2(sN) \lg sN \rceil, \text{ and } h_p(x) = x \mod p \]

**Lemma**

*If* \( x \neq y \) *then,* \( \Pr[\text{Bob says equal}] = \Pr[h_p(x) = h_p(y)] \leq 1/s \)

**Proof.**

Given \( x \neq y \), \( h_p(x) = h_p(y) \Rightarrow x \mod p = y \mod p \).

- \( D = |x - y| \), then \( D \mod p = 0 \), and \( D \leq 2^N \).
- \( D = p_1 \ldots p_k \) prime factorization.
String Equality: Randomized Algorithm

Error probability

$x, y$ N-bit string, $M = \lceil 2(sN) \lg sN \rceil$, and $h_p(x) = x \mod p$

Lemma

If $x \neq y$ then, $\Pr[Bob \ says\ equal] = \Pr[h_p(x) = h_p(y)] \leq 1/s$

Proof.

Given $x \neq y$, $h_p(x) = h_p(y) \Rightarrow x \mod p = y \mod p$.

- $D = |x - y|$, then $D \ mod\ p = 0$, and $D \leq 2^N$.
- $D = p_1 \ldots p_k$ prime factorization. All $p_i \geq 2 \Rightarrow D \geq 2^k$. 
String Equality: Randomized Algorithm

Error probability

$x, y$ N-bit string, $M = \lceil 2(sN) \log sN \rceil$, and $h_p(x) = x \mod p$

**Lemma**

If $x \neq y$ then, \( \Pr[Bob \ says \ equal] = \Pr[h_p(x) = h_p(y)] \leq 1/s \)

**Proof.**

Given $x \neq y$, $h_p(x) = h_p(y) \Rightarrow x \mod p = y \mod p$.

1. $D = |x - y|$, then $D \ mod \ p = 0$, and $D \leq 2^N$.
2. $D = p_1 \ldots p_k$ prime factorization. All $p_i \geq 2 \Rightarrow D \geq 2^k$.
3. $2^k \leq D \leq 2^N \Rightarrow k \leq N$. $D$ has at most $N$ divisors.
String Equality: Randomized Algorithm

Error probability

\(x, y\) N-bit string, \(M = \lceil 2(sN) \lg sN \rceil\), and \(h_p(x) = x \mod p\)

Lemma

If \(x \neq y\) then, \(\Pr[Bob\ says\ equal] = \Pr[h_p(x) = h_p(y)] \leq \frac{1}{s}\)

Proof.

Given \(x \neq y\), \(h_p(x) = h_p(y) \Rightarrow x \mod p = y \mod p\).

- \(D = |x - y|\), then \(D \mod p = 0\), and \(D \leq 2^N\).
- \(D = p_1 \ldots p_k\) prime factorization. All \(p_i \geq 2 \Rightarrow D \geq 2^k\).
- \(2^k \leq D \leq 2^N \Rightarrow k \leq N\). \(D\) has at most \(N\) divisors.
- Probability that a random prime \(p\) from \(\{1, \ldots, M\}\) is a divisor 
  \(= \frac{k}{\pi(M)} \leq \frac{N}{\pi(M)}\)
String Equality: Randomized Algorithm

Error probability

Let $x, y$ be $N$-bit strings, $M = \lceil 2(sN) \log sN \rceil$, and $h_p(x) = x \mod p$.

**Lemma**

If $x \neq y$ then, $\Pr[Bob \ says\ equal] = \Pr[h_p(x) = h_p(y)] \leq 1/s$

**Proof.**

Given $x \neq y$, $h_p(x) = h_p(y) \Rightarrow x \mod p = y \mod p$.

- $D = |x - y|$, then $D \mod p = 0$, and $D \leq 2^N$.
- $D = p_1 \cdots p_k$ prime factorization. All $p_i \geq 2 \Rightarrow D \geq 2^k$.
- $2^k \leq D \leq 2^N \Rightarrow k \leq N$. $D$ has at most $N$ divisors.
- Probability that a random prime $p$ from $\{1, \ldots, M\}$ is a divisor $= \frac{k}{\pi(M)} \leq \frac{N}{\pi(M)} \leq \frac{N}{M/\log M} = \frac{N}{2(sN) \log sN} \log M \leq \frac{1}{s}$.
Choose large enough \( s \). Error prob: \( 1/s \).
Low Error Probability

1. Choose large enough $s$. Error prob: $1/s$.
2. Alice repeats the process $R$ times, and Bob says *equal* only if he gets equal all $R$ times.
Low Error Probability

1. Choose large enough $s$. Error prob: $\frac{1}{s}$.

2. Alice repeats the process $R$ times, and Bob says *equal* only if he gets equal all $R$ times.

Error probability: $\frac{1}{s^R}$. 
Low Error Probability

1. Choose large enough $s$. Error prob: $1/s$.
2. Alice repeats the process $R$ times, and Bob says equal only if he gets equal all $R$ times.

Error probability: $\frac{1}{s^R}$. For $s = 5$, $R = 10$, $\frac{1}{5^{10}} \leq 0.000001$. 

Amount of Communication

Each round sends $2$ integers $\leq M$. # bits: $2 \log_2 M \leq 4(\log_2 s + \log_2 N)$.

If $x$ and $y$ are copies of Wikipedia, about $25$ billion characters. If $8$ bits per character, then $N \approx 2^{38}$ bits.

Second approach will send $10(2 \log_10(2N \log_5 N)) \leq 1280$ bits.
## Error Probability and Communication

### Low Error Probability

1. Choose large enough $s$. Error prob: $1/s$.
2. Alice repeats the process $R$ times, and Bob says *equal* only if he gets equal all $R$ times.

   Error probability: $\frac{1}{s^R}$. For $s = 5$, $R = 10$, $\frac{1}{5^{10}} \leq 0.000001$.

   $$M = \lceil 2(sN) \log sN \rceil$$

### Amount of Communication

Each round sends 2 integers $\leq M$. # bits: $2 \log M \leq 4(\log s + \log N)$. 

- If $x$ and $y$ are copies of Wikipedia, about 25 billion characters. If 8 bits per character, then $N \approx 2^{38}$ bits.
- Second approach will send $10(2 \log (10N \log 5N)) \leq 1280$ bits.
Error Probability and Communication

Low Error Probability

1. Choose large enough $s$. Error prob: $1/s$.
2. Alice repeats the process $R$ times, and Bob says equal only if he gets equal all $R$ times.
   Error probability: $\frac{1}{s^R}$. For $s = 5, R = 10$, $\frac{1}{5^{10}} \leq 0.000001$.

$$M = \lceil 2(sN) \lg sN \rceil$$

Amount of Communication

Each round sends 2 integers $\leq M$. # bits: $2 \lg M \leq 4(\lg s + \lg N)$.

If $x$ and $y$ are copies of Wikipedia, about 25 billion characters. If 8 bits per character, then $N \approx 2^{38}$ bits.
Low Error Probability

1. Choose large enough $s$. Error prob: $1/s$.
2. Alice repeats the process $R$ times, and Bob says *equal* only if he gets equal all $R$ times.

Error probability: $\frac{1}{s^R}$. For $s = 5$, $R = 10$, $\frac{1}{5^{10}} \leq 0.000001$.

$$M = \lceil 2(sN) \log sN \rceil$$

Amount of Communication

Each round sends 2 integers $\leq M$. # bits: $2 \log M \leq 4(\log s + \log N)$.

If $x$ and $y$ are copies of Wikipedia, about 25 billion characters. If 8 bits per character, then $N \approx 2^{38}$ bits.

Second approach will send $10(2 \log (10N \log 5N)) \leq 1280$ bits.
Part III

Karp-Rabin Pattern Matching Algorithm
Pattern Matching

Given a string $T$ of length $m$ and pattern $P$ of length $n$, s.t. $m \gg n$, find all occurrences of $P$ in $T$.

Example

$T=$abracadabra, $P=$ab.
Pattern Matching

Given a string $T$ of length $m$ and pattern $P$ of length $n$, s.t. $m \gg n$, find all occurrences of $P$ in $T$.

**Example**

$T =$ abracadabra, $P =$ ab.

Solution $S = \{1, 8\}$. 
Pattern Matching

Given a string $T$ of length $m$ and pattern $P$ of length $n$, s.t. $m \gg n$, find all occurrences of $P$ in $T$.

Example

$T = \text{abracadabra}$, $P = \text{ab}$.

Solution $S = \{1, 8\}$.

For $j > i$, let $T_{i...j} = T[i]T[i + 1] \ldots T[j]$. 
Pattern Matching

Given a string $T$ of length $m$ and pattern $P$ of length $n$, s.t. $m \gg n$, find all occurrences of $P$ in $T$.

Example

$T=$abracadabra, $P=$ab.

Solution $S = \{1, 8\}$.

For $j > i$, let $T_{i...j} = T[i]T[i+1]...T[j]$.

Brute force algorithm

$S = \emptyset$. For each $i = 1...m - n + 1$

- If $T_{i...i+n-1} = P$ then $S = S \cup \{i\}$. 
Pattern Matching

Given a string $T$ of length $m$ and pattern $P$ of length $n$, s.t. $m \gg n$, find all occurrences of $P$ in $T$.

Example

$T =$ abracadabra, $P =$ ab.

Solution $S = \{1, 8\}$.

For $j > i$, let $T_{i...j} = T[i]T[i+1]...T[j]$.

Brute force algorithm

$S = \emptyset$. For each $i = 1...m - n + 1$

- If $T_{i...i+n-1} = P$ then $S = S \cup \{i\}$.

$O(mn)$ run-time.
Using Hash Function

Pick a prime $p$ u.a.r. from $\{1, \ldots, M\}$. $h_p(x) = x \mod p$.

**Brute force algorithm using hash function**

$S = \emptyset$. For each $i = 1 \ldots m - n + 1$

- If $h_p(T_{i \ldots i+n-1}) = h_p(P)$ then $S = S \cup \{i\}$.
Using Hash Function

Pick a prime $p$ u.a.r. from $\{1, \ldots, M\}$. $h_p(x) = x \ mod \ p$.

**Brute force algorithm using hash function**

$S = \emptyset$. For each $i = 1 \ldots m - n + 1$

- If $h_p(T_{i \ldots i+n-1}) = h_p(P)$ then $S = S \cup \{i\}$.

If $x$ is of length $n$, then computing $h_p(x)$ takes $O(n)$ running time.

Overall $O(mn)$ running time.
Using Hash Function

Pick a prime $p$ u.a.r. from $\{1, \ldots, M\}$. $h_p(x) = x \mod p$.

**Brute force algorithm using hash function**

$S = \emptyset$. For each $i = 1 \ldots m - n + 1$

- If $h_p(T_{i \ldots i+n-1}) = h_p(P)$ then $S = S \cup \{i\}$.

If $x$ is of length $n$, then computing $h_p(x)$ takes $O(n)$ running time.

Overall $O(mn)$ running time.

Can we compute $h_p(T_{i+1 \ldots i+n})$ using $h_p(T_{i \ldots i+n-1})$ fast?
Let $a$ and $b$ be (non-negative) integers.

$$(a + b) \mod p = ((a \mod p) + (b \mod p)) \mod p$$
Let $a$ and $b$ be (non-negative) integers.

\[(a + b) \mod p = ((a \mod p) + (b \mod p)) \mod p\]

\[(a \cdot b) \mod p = ((a \mod p) \cdot (b \mod p)) \mod p\]
Rolling Hash

\[ x = T_{i...i+n-1} \text{ and } x' = T_{i+1...i+n}. \]

**Example**

\[ x = 1011001, \text{ and } x' = 0110010 \text{ (or } x' = 0110011). \]
Rolling Hash

\[ x = T_{i...i+n-1} \text{ and } x' = T_{i+1...i+n}. \]

**Example**

\[ x = 1011001, \text{ and } x' = 0110010 \text{ (or } x' = 0110011). \]

\[ x' = 2(x - x_{hb}2^{n-1}) + x'_{lb} \]
Rolling Hash

\[ x = T_{i...i+n-1} \text{ and } x' = T_{i+1...i+n}. \]

**Example**

\[ x = 1011001, \text{ and } x' = 0110010 \text{ (or } x' = 0110011). \]

\[
x' = 2(x - x_{hb}2^{n-1}) + x'_{lb} \\
= 2x - x_{hb}2^n + x'_{lb}
\]
Rolling Hash

\[ x = T_{i...i+n-1} \text{ and } x' = T_{i+1...i+n}. \]

Example

\[ x = 1011001, \text{ and } x' = 0110010 \text{ (or } x' = 0110011). \]

\[
x' = 2(x - x_{hb}2^{n-1}) + x'_{lb}
= 2x - x_{hb}2^n + x'_{lb}
\]

\[
h_p(x') = x' \mod p
= (2(x \mod p) - x_{hb}(2^n \mod p) + x'_{lb}) \mod p
= (2h_p(x) - x_{hb}h_p(2^n) + x'_{lb}) \mod p
\]
Rolling Hash

\[ x = T_{i...i+n−1} \text{ and } x' = T_{i+1...i+n}. \]

**Example**

\[ x = 1011001, \text{ and } x' = 0110010 \text{ (or } x' = 0110011). \]

\[
x' = 2(x - x_{hb}2^{n−1}) + x'_{lb}
= 2x - x_{hb}2^n + x'_{lb}
\]

\[
h_p(x') = x' \mod p
= (2(x \mod p) - x_{hb}(2^n \mod p) + x'_{lb}) \mod p
= (2h_p(x) - x_{hb}h_p(2^n) + x'_{lb}) \mod p
= (2h_p(T_{i...i+n−1}) - T_ih_p(2^n) + T_{i+n}) \mod p
\]
Karp-Rabin Algorithm

\( p \): a random prime from \( \{1, \ldots, M\} \).

Rolling hash: 
\[
h_p(T_{i+1} \ldots i+n) = (2h_p(T_{i \ldots i+n-1}) - T_i h_p(2^n) + T_{i+n}) \mod p.
\]
Karp-Rabin Algorithm

$p$: a random prime from $\{1, \ldots, M\}$.

Rolling hash: $h_p(T_{i+1}\ldots i+n) = (2h_p(T_{i}\ldots i+n-1) - T_i h_p(2^n) + T_{i+n}) \mod p$.

1. Set $S = \emptyset$. Compute $h_p(T_1\ldots n)$, $h_p(2^n)$, and $h_p(P)$.

2. For each $i = 1, \ldots, m - n + 1$
   
   1. If $h_p(T_{i}\ldots i+n-1) = h_p(P)$, then $S = S \cup \{i\}$.
   2. Compute $h_p(T_{i+1}\ldots i+n)$ using $h_p(T_{i}\ldots i+n-1)$ and $h_p(2^n)$ by applying rolling hash.
Karp-Rabin Algorithm

\( p \): a random prime from \( \{1, \ldots, M\} \).

Rolling hash: \( h_p(T_{i+1}\ldots i+n) = (2h_p(T_{i}\ldots i+n-1) - T_i h_p(2^n) + T_{i+n}) \mod p \).

1. Set \( S = \emptyset \). Compute \( h_p(T_1\ldots n), h_p(2^n), \text{ and } h_p(P) \).

2. For each \( i = 1, \ldots, m-n+1 \)
   - If \( h_p(T_{i}\ldots i+n-1) = h_p(P) \), then \( S = S \cup \{i\} \).
   - Compute \( h_p(T_{i+1}\ldots i+n) \) using \( h_p(T_{i}\ldots i+n-1) \) and \( h_p(2^n) \) by applying rolling hash.

Running Time

- In Step 1, computing \( h_p(x) \) for an \( n \) bit \( x \) is in \( O(n) \) time.
Karp-Rabin Algorithm

\( p \): a random prime from \( \{1, \ldots, M\} \).

Rolling hash:  
\[
h_p(T_{i+1} \ldots i+n) = (2h_p(T_i \ldots i+n-1) - T_i h_p(2^n) + T_{i+n}) \mod p.
\]

1. Set \( S = \emptyset \). Compute \( h_p(T_{1 \ldots n}), h_p(2^n), \) and \( h_p(P) \).

2. For each \( i = 1, \ldots, m - n + 1 \)
   
   1. If \( h_p(T_i \ldots i+n-1) = h_p(P) \), then \( S = S \cup \{i\} \).
   
   2. Compute \( h_p(T_{i+1} \ldots i+n) \) using \( h_p(T_i \ldots i+n-1) \) and \( h_p(2^n) \) by applying rolling hash.

**Running Time**

- In Step 1, computing \( h_p(x) \) for an \( n \) bit \( x \) is in \( O(n) \) time.

Assuming \( O(lg M) \) bit arithmetic can be done in \( O(1) \) time,

- Since \( h_p(.) \) produces \( lg M \) bit numbers, both steps inside for loop can be done in \( O(1) \) time.
Karp-Rabin Algorithm

\( p \): a random prime from \( \{1, \ldots, M\} \).

Rolling hash: \( h_p(T_{i+1} \ldots i+n) = (2h_p(T_i \ldots i+n-1) - T_i h_p(2^n) + T_{i+n}) \mod p \).

1. Set \( S = \emptyset \). Compute \( h_p(T_1 \ldots n) \), \( h_p(2^n) \), and \( h_p(P) \).

2. For each \( i = 1, \ldots, m - n + 1 \)
   
   1. If \( h_p(T_{i \ldots i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
   
   2. Compute \( h_p(T_{i+1} \ldots i+n) \) using \( h_p(T_i \ldots i+n-1) \) and \( h_p(2^n) \) by applying rolling hash.

Running Time

- In Step 1, computing \( h_p(x) \) for an \( n \) bit \( x \) is in \( O(n) \) time.

Assuming \( O(\lg M) \) bit arithmetic can be done in \( O(1) \) time,

- Since \( h_p(.) \) produces \( \lg M \) bit numbers, both steps inside for loop can be done in \( O(1) \) time.

- Overall \( O(m + n) \) time.
Karp-Rabin Algorithm

\( p \): a random prime from \( \{1, \ldots, M\} \).

Rolling hash: \( h_p(T_{i+1\ldots i+n}) = (2h_p(T_{i\ldots i+n-1}) - T_i h_p(2^n) + T_{i+n}) \mod p \).

1. Set \( S = \emptyset \). Compute \( h_p(T_1\ldots n), h_p(2^n), \) and \( h_p(P) \).
2. For each \( i = 1, \ldots, m - n + 1 \)
   1. If \( h_p(T_{i\ldots i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
   2. Compute \( h_p(T_{i+1\ldots i+n}) \) using \( h_p(T_{i\ldots i+n-1}) \) and \( h_p(2^n) \) by applying rolling hash.

Running Time

- In Step 1, computing \( h_p(x) \) for an \( n \) bit \( x \) is in \( O(n) \) time.

Assuming \( O(\lg M) \) bit arithmetic can be done in \( O(1) \) time,

- Since \( h_p(.) \) produces \( \lg M \) bit numbers, both steps inside for loop can be done in \( O(1) \) time.

- Overall \( O(m + n) \) time. Can’t do better.
Karp-Rabin Algorithm: Error Probability

1. For each $i = 1, \ldots, m - n + 1$
   1. If $h_p(T_{i\ldots i+n-1}) = h_p(P)$, then $S = S \cup \{i\}$.
   2. Compute $h_p(T_{i+1\ldots i+n})$ using $h_p(T_{i\ldots i+n-1})$ and $h_p(2^n)$.

Lemma

If match at any position $i$ then $i \in S$. In otherwords if $T_{i\ldots i+n-1} = P$, then $i \in S$.

All matched positions are in $S$. 
For each $i = 1, \ldots, m - n + 1$

1. If $h_p(T_{i\ldots i+n-1}) = h_p(P)$, then $S = S \cup \{i\}$.
2. Compute $h_p(T_{i+1\ldots i+n})$ using $h_p(T_{i\ldots i+n-1})$ and $h_p(2^n)$.

**Lemma**

If match at any position $i$ then $i \in S$. In other words if $T_{i\ldots i+n-1} = P$, then $i \in S$.

All matched positions are in $S$.

Can it contain unmatched positions?
Karp-Rabin Algorithm: Error Probability

1. For each \( i = 1, \ldots, m - n + 1 \)
   1. If \( h_p(T_{i\ldots i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
   2. Compute \( h_p(T_{i+1\ldots i+n}) \) using \( h_p(T_{i\ldots i+n-1}) \) and \( h_p(2^n) \).

**Lemma**

*If match at any position \( i \) then \( i \in S \). In otherwords if \( T_{i\ldots i+n-1} = P \), then \( i \in S \).*

All matched positions are in \( S \).

Can it contain unmatched positions? YES!
Karp-Rabin Algorithm: Error Probability

1. For each $i = 1, \ldots, m - n + 1$
   1. If $h_p(T_{i...i+n-1}) = h_p(P)$, then $S = S \cup \{i\}$.
   2. Compute $h_p(T_{i+1...i+n})$ using $h_p(T_{i...i+n-1})$ and $h_p(2^n)$.

Lemma

If match at any position $i$ then $i \in S$. In other words if $T_{i...i+n-1} = P$, then $i \in S$.

All matched positions are in $S$.

Can it contain unmatched positions? YES! With what probability?
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

For each \( i = 1, \ldots, m - n + 1 \)

1. If \( h_p(T_{i \ldots i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
2. Compute \( h_p(T_{i+1 \ldots i+n}) \) using \( h_p(T_{i \ldots i+n-1}) \) and \( h_p(2^n) \).
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

1. For each $i = 1, \ldots, m - n + 1$
   1. If $h_p(T_{i\ldots i+n-1}) = h_p(P)$, then $S = S \cup \{i\}$.
   2. Compute $h_p(T_{i+1\ldots i+n})$ using $h_p(T_{i\ldots i+n-1})$ and $h_p(2^n)$.

Set $M = \lceil 2(sn) \lg sn \rceil$. Given $x \neq y$, $Pr[h_p(x) = h_p(y)] \leq 1/s$. 

---

Ruta (UIUC)
CS473
Spring 2021
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

1. For each \( i = 1, \ldots, m - n + 1 \)
   1. If \( h_p(T_{i...i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
   2. Compute \( h_p(T_{i+1...i+n}) \) using \( h_p(T_{i...i+n-1}) \) and \( h_p(2^n) \).

Set \( M = \lceil 2(sn) \lg sn \rceil \). Given \( x \neq y \), \( \Pr[h_p(x) = h_p(y)] \leq 1/s \).

False positive: \( \Pr[S \text{ contains an } i, \text{ while no match at } i] \)
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

1. For each $i = 1, \ldots, m - n + 1$
   1. If $h_p(T_{i \ldots i+n-1}) = h_p(P)$, then $S = S \cup \{i\}$.
   2. Compute $h_p(T_{i+1 \ldots i+n})$ using $h_p(T_{i \ldots i+n-1})$ and $h_p(2^n)$.

Set $M = \lceil 2(sn) \log sn \rceil$. Given $x \neq y$, $\Pr[h_p(x) = h_p(y)] \leq 1/s$.

False positive: Pr[S contains an i, while no match at i]

- Given $T_{i \ldots i+n-1} \neq P$, $\Pr[i \in S] \leq 1/s$. 
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

For each $i = 1, \ldots, m - n + 1$
1. If $h_p(T_{i \ldots i+n-1}) = h_p(P)$, then $S = S \cup \{i\}$.
2. Compute $h_p(T_{i+1 \ldots i+n})$ using $h_p(T_{i \ldots i+n-1})$ and $h_p(2^n)$.

Set $M = \lceil 2(sn) \lg sn \rceil$. Given $x \neq y$, $\Pr[h_p(x) = h_p(y)] \leq 1/s$.

False positive: Pr[S contains an i, while no match at i]

- Given $T_{i \ldots i+n-1} \neq P$, $\Pr[i \in S] \leq 1/s$.
- $\Pr[\text{Any index in } S \text{ is wrong}]$
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

For each \( i = 1, \ldots, m - n + 1 \)

1. If \( h_p(T_{i \ldots i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
2. Compute \( h_p(T_{i+1 \ldots i+n}) \) using \( h_p(T_{i \ldots i+n-1}) \) and \( h_p(2^n) \).

Set \( M = \lceil 2(sn) \log sn \rceil \). Given \( x \neq y \), \( \Pr[h_p(x) = h_p(y)] \leq 1/s \).

False positive: \( \Pr[S \text{ contains an } i, \text{ while no match at } i] \)

- Given \( T_{i \ldots i+n-1} \neq P \), \( \Pr[i \in S] \leq 1/s \).
- \( \Pr[\text{Any index in } S \text{ is wrong}] \leq m/s \) (Union bound).
Karp-Rabin Algorithm: Error Probability

Pr[S contains an index i, while there is no match at i]

1. For each \( i = 1, \ldots, m - n + 1 \)
   1. If \( h_p(T_{i\ldots i+n-1}) = h_p(P) \), then \( S = S \cup \{i\} \).
   2. Compute \( h_p(T_{i+1\ldots i+n}) \) using \( h_p(T_{i\ldots i+n-1}) \) and \( h_p(2^n) \).

Set \( M = \lceil 2(sn) \log sn \rceil \). Given \( x \neq y \), \( \Pr[h_p(x) = h_p(y)] \leq 1/s \).

False positive: \( \Pr[S \text{ contains an } i, \text{ while no match at } i] \)

- Given \( T_{i\ldots i+n-1} \neq P \), \( \Pr[i \in S] \leq 1/s \).
- \( \Pr[\text{Any index in } S \text{ is wrong}] \leq m/s \) (Union bound).
- To ensure \( S \) is correct with at least 0.99 probability, we need

\[
1 - \frac{m}{s} = 0.99 \iff \frac{m}{s} = \frac{1}{100} \iff s = 100m
\]
Karp-Rabin Algorithm

Back to running time

Running Time

- In Step 1, computing \( h_p(x) \) for an \( n \) bit \( x \) is in \( O(n) \) time.

Assuming \( O(lg M) \) bit arithmetic can be done in \( O(1) \) time,

- Since \( h_p(.) \) produces \( lg M \) bit numbers, both steps inside \texttt{for loop} can be done in \( O(1) \) time.

- Overall \( O(m + n) \) time. Can’t do better.

\[
M = \lceil 200mn \lg 100mn \rceil \Rightarrow \lg M = O(lg m)
\]
Running Time

- In Step 1, computing $h_p(x)$ for an $n$ bit $x$ is in $O(n)$ time.
- Assuming $O(\lg M)$ bit arithmetic can be done in $O(1)$ time,
- Since $h_p(.)$ produces $\lg M$ bit numbers, both steps inside `for loop` can be done in $O(1)$ time.
- Overall $O(m + n)$ time. Can’t do better.

$$M = \lceil 200mn \lg 100mn \rceil \Rightarrow \lg M = O(\lg m)$$

Even if $T$ is entire Wikipedia, with bit length $m \approx 2^{38}$,
Karp-Rabin Algorithm
Back to running time

Running Time
- In Step 1, computing \( h_p(x) \) for an \( n \) bit \( x \) is in \( O(n) \) time.
- Assuming \( O(\lg M) \) bit arithmetic can be done in \( O(1) \) time,
  
  - Since \( h_p(.) \) produces \( \lg M \) bit numbers, both steps inside for loop can be done in \( O(1) \) time.
- Overall \( O(m + n) \) time. Can’t do better.

\[
M = \lceil 200mn \lg 100mn \rceil \Rightarrow \lg M = O(\lg m)
\]

Even if \( T \) is entire Wikipedia, with bit length \( m \approx 2^{38} \),

\[
\lg M \approx 64 \quad (\text{assuming bit-length of } n \leq 2^{16})
\]
In Step 1, computing $h_p(x)$ for an $n$ bit $x$ is in $O(n)$ time.

Assuming $O(\lg M)$ bit arithmetic can be done in $O(1)$ time,

Since $h_p(.)$ produces $\lg M$ bit numbers, both steps inside for loop can be done in $O(1)$ time.

Overall $O(m + n)$ time. Can’t do better.

$$M = \lceil 200mn \lg 100mn \rceil \Rightarrow \lg M = O(\lg m)$$

Even if $T$ is entire Wikipedia, with bit length $m \approx 2^{38}$,

$\lg M \approx 64$ (assuming bit-length of $n \leq 2^{16}$)

64-bit arithmetic is doable on laptops!
Take away points

1. Hashing is a powerful and important technique. Many practical applications.
2. Randomization fundamental to understand hashing.
3. Good and efficient hashing possible in theory and practice with proper definitions (universal, perfect, etc).
4. Related ideas of creating a compact fingerprint/sketch for objects is very powerful in theory and practice.