GEMS OF TCS

DATA STRUCTURES

Sasha Golovnev
February 9, 2021
DATA STRUCTURES

Stack, Queue, List, Heap

Search Trees

Hash Tables

hash(unsigned x) {
    x ^= x >> (w-m);
    return (a*x) >> (w-m);
}
Coping with Hard Problems

• Some problems are too hard to solve exactly
Coping with Hard Problems

- Some problems are too hard to solve exactly
- Approximation
COPING WITH HARD PROBLEMS

- Some problems are too hard to solve exactly
- Approximation
- Randomness
COPING WITH HARD PROBLEMS

• Some problems are too hard to solve exactly

• Approximation

• Randomness

• Today: Preprocessing
**Examples**

- **Graph Distances**: Preprocess a road network in order to efficiently compute distance queries between cities (Google Maps)

  Preprocessing takes forever

  Query: Fast
Examples

- **Graph Distances:** Preprocess a road network in order to efficiently compute distance queries between cities (Google Maps)

- **Clustering:** Preprocess a set of movies in order to efficiently find closest movie to a query movie (Netflix recommendations)
DATA STRUCTURES

1. Preprocessing

a lot of time
DATA STRUCTURES

Queries

Preprocessing
DATA STRUCTURES

Queries

New York — Washington
DATA STRUCTURES

Queries

New York — Washington

Preprocessing
DATA STRUCTURES

Queries

New York — Washington
Washington — Boston

Preprocessing
Stealing Passwords
PASSWORD HASHING

User → login/pwd → Gmail
PASSWORD HASHING

haveibeenpwned.com: Your account has been compromised

Oh No!

login/pwd

login/pwd

Oh No!
PASSWORD HASHING

User sends login/pwd

Login/Hash(pwd)
Password Hashing

Password Hashing:

- **User** sends `login/pwd` to the server.
- The server hashes the password and sends `login/hash(pwd)` back to the user.

Hash examples:
- `hash(qwerty)=1xe4ht`
- `hash(111111)=nh83l0`
I'm fine!

PASSWORD HASHING

haveibeenpwned.com: Your account has been compromised

hash(qwerty)=1xe4ht
hash(111111)=nh83l0
HASHING

• (Cryptographic) hash function maps strings to strings such that it’s hard to invert

Ideally, in order to find pwd $\rightarrow$ fixed hash, you have to brute force all pwds
Hashing

- (Cryptographic) hash function maps strings to strings such that it’s hard to invert
- Ideally, to find a password that leads to a fixed hash value, one needs to brute force all possible passwords
HASHING

- (Cryptographic) hash function maps strings to strings such that it’s hard to invert
- Ideally, to find a password that leads to a fixed hash value, one needs to brute force all possible passwords
- Hash functions are publicly known (SHA-3)

[Wiki page]
Hashing

- (Cryptographic) hash function maps strings to strings such that it’s hard to invert
- Ideally, to find a password that leads to a fixed hash value, one needs to brute force all possible passwords
- Hash functions are publicly known (SHA-3)
- For now, consider hash functions $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ that are bijections

$$f: \{0,1\}^n \rightarrow \{0,1\}^n$$

$N = 2^n$
Inverting a Bijection

- Let $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be a bijection

  **Preprocess:** years

  **Query:** hash value $\rightarrow$ pwd

  Given $Y \in \{1, \ldots, N\}$

  find $x$ s.t. $f(x) = Y$
2 Naive solution

I. No preprocessing
   Space = 0
   Time = $N \approx 10^{77}$
   $\leq 10^{20}$ operations per second
   the age of Universe $10^{15}$ seconds

II. Preprocess: stone
    
    $\text{hash} \rightarrow \text{pwd}$
    $0 ... 0 \rightarrow \text{pwd}_1$
    $0 ... 01 \rightarrow \text{pwd}_2$
    ...
    $1 ... 1 \rightarrow \text{pwd}_N$

   Space = $N \approx 10^{77}$
   Time = $\log N$

   # of el. particles in observable Universe $\approx 10^{86}$
Inverting a Bijection

- Let $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
**Inverting a Bijection**

- Let $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be a **bijection**
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
Inverting a Bijection

- Let $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
- In- and out-degrees of all vertices are 1
INVERTING A BIJECTION

• Let $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be a bijection

• Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$

• Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$

• In- and out-degrees of all vertices are 1

• Thus, this graph is a union of cycles
INVERTING A BIJECTION
INVERTING A BIJECTION
Inverting a Bijection

$\sqrt{N}$
INVERTING A BIJECTION
INVERTING A BIJECTION
Inverting a Bijection

Store $x$ landmarks,
Inverting a Bijection

Store $x$ landmarks, and links $\rightarrow$ to previous landmarks.
Inverting a Bijection

Store $x$ landmarks, and links $\hat{r}$ to previous landmarks $\sqrt{N}$

space $S \approx \sqrt{N}$

$S \leq \frac{N}{\sqrt{N}} = \sqrt{N}$
Inverting a Bijection

Store \( x \) landmarks, and links \( \tilde{\gamma} \) to previous landmarks
space \( S \approx \sqrt{N} \)
Inverting a Bijection

Store $x$ landmarks, and links $\xrightarrow{\sim}$ to previous landmarks
space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Inverting a Bijection

Store $x$ landmarks, and links $\tilde{\mathcal{F}}$ to previous landmarks.

Space $S \approx \sqrt{N}$

Time $T \approx \sqrt{N}$:

Invert $y = f(x)$ given hash, find pwd.
Inverting a Bijection

Recall $f$ is publicly known

$y = f(x)$

$\Rightarrow f(y)$

Store $x$ landmarks,
and links $F$ to previous landmarks
space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Invert $y = f(x)$
Inverting a Bijection

Store $x$ landmarks, and links $\mathcal{F}$ to previous landmarks. Space $S \approx \sqrt{N}$, time $T \approx \sqrt{N}$: Invert $y = f(x)$.
Inverting a Bijection

Store $x$ landmarks, and links $\bar{\gamma}$ to previous landmarks

space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Invert $y = f(x)$
Inverting a Bijection

Store \( x \) landmarks, and links \( \bar{f} \) to previous landmarks. Space \( S \approx \sqrt{N} \). Time \( T \approx \sqrt{N} \). Invert \( y = f(x) \).
Inverting a Bijection

Store $x$ landmarks, and links to previous landmarks

space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Invert $y = f(x)$
Inverting a Bijection

Store \( x \) landmarks, and links \( \tilde{\mathcal{L}} \) to previous landmarks

space \( S \approx \sqrt{N} \)
time \( T \approx \sqrt{N} \):
Invert \( y = f(x) \)
Inverting a Bijection

Store $x$ landmarks, and links $\Rightarrow$ to previous landmarks. Space $S \approx \sqrt{N}$, time $T \approx \sqrt{N}$. Invert $y = f(x)$. 

\[
y = f(x) \\
f(y)
\]
DATA STRUCTURE

- Let $ST = N$

-ex. $S = N^{1/3}$  
$T = N^{2/3}$
Data Structure

• Let $ST = N$

• Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
**Data Structure**

- Let $ST = N$
- Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
- Partition the graph into cycles
**Data Structure**

- Let $ST = N$
- Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
- Partition the graph into cycles
- Ignore cycles of length $\leq T$
DATA STRUCTURE

- Let $ST = N$
- Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
- Partition the graph into cycles
- Ignore cycles of length $\leq T$
- In all other cycles store every $T$th vertex as a landmark

\[ S \leq N/T \]
Data Structure

- Let $ST = N$
- Let’s define a directed graph on $N$ vertices with edges $x \rightarrow f(x)$
- Partition the graph into cycles
- Ignore cycles of length $\leq T$
- In all other cycles store every $T$th vertex as a landmark
- Space: $S$, query time: $T$
Prohibited Passwords
PROHIBITED PASSWORDS

- Check if entered password is in the list of \( m \) prohibited passwords
PROHIBITED PASSWORDS

• Check if entered password is in the list of $m$ prohibited passwords

• We can store $m$ strings, check in $\sim \log m$ time
PROHIBITED PASSWORDS

• Check if entered password is in the list of \( m \) prohibited passwords

• We can store \( m \) strings, check in \( \sim \log m \) time

• Bloom filters: store \( \sim m \) bits, check in \( O(1) \) time
PROHIBITED PASSWORDS

- Check if entered password is in the list of $m$ prohibited passwords
- We can store $m$ strings, check in $\sim \log m$ time
- **Bloom filters**: store $\sim m$ bits, check in $O(1)$ time
- We’ll be wrong with small probability
We want a data structure that supports two functions
Data Structure

• We want a data structure that supports two functions
  • Insert($x$)
Data Structure

- We want a data structure that supports two functions
  - Insert(x)
  - Lookup(x)

Naive: list of prohibited pool would use too much space

Lookup would be less efficient
DATA STRUCTURE

• We want a data structure that supports two functions
  • Insert(x)
  • Lookup(x)
• Hashtables: less efficient but don’t make mistakes
DATA STRUCTURE

- We want a data structure that supports two functions
  - Insert(x)
  - Lookup(x)
- Hashtables: less efficient but don’t make mistakes
- Bloom filter will use array of $n$ bits $A[0], \ldots, A[n-1]$, initialized with zeros
Data Structure

- We want a data structure that supports two functions
  - Insert($x$)
  - Lookup($x$)
- Hashables: less efficient but don’t make mistakes
- Bloom filter will use array of $n$ bits $A[0], \ldots, A[n - 1]$, initialized with zeros
- We’ll use $k = O(1)$ hash functions
Hash Functions

• We have $k$ hash functions $f_1, \ldots, f_k$ from strings to $\{0, \ldots, n - 1\}$ integers.
Hash Functions

- We have $k$ hash functions $f_1, \ldots, f_k$ from strings to $\{0, \ldots, n - 1\}$

- Assume that functions are independent and uniform random

  \[
  \forall \text{prob} \ x \to f_i(x) \xrightarrow{\text{equally likely}} 1 \frac{1}{n} \ldots \frac{1}{n}
  \]
BLOOM FITLER

- Insert(x):
  - for $i = 1, \ldots, k$,
  - $A[f_i(x)] \leftarrow 1$

whether or not bit is already set to 1
**Bloom Filter**

- **Insert(x):**
  - for $i = 1, \ldots, k$,
    - $A[f_i(x)] \leftarrow 1$
- **Lookup(x):**
  - return 1 iff for every $i = 1, \ldots, k$.
    - $A[f_i(x)] = 1$

**Obs:** $x$ was inserted, $\text{lookup}(x)$ correctly says $x$ is in data structure

**Problem:**
Problem: x was not inserted

Lookup(x) may sometimes say that x was inserted
Parameters

\[ m - \text{# of prohibited pwd}s \]
\[ n - \text{# of bits in array} \]
\[ k - \text{# of hash funs} \]

Remains, analyze \( P(K) \) of mistake

Analysis to set params \( n, k \).

We've inserted all \( m \) strings

\[ P(\text{CA} = 0) = ? \]

Insert 1st pwd:

\[ S_1 \leftarrow P(\text{CA} = 0) = 1 - \frac{1}{n} \]

\[ S_k \leftarrow P(\text{CA} = 0) = 1 - \frac{1}{n} \]

\( m \)th pwd

\[ S_1 \]

\[ S_k \]

\[ P(\text{CA} = 0) = 1 - \frac{1}{n} \]
\[ \Pr \left( 1 - \frac{1}{n} \right)^{m \cdot k} \]

\[ \Pr[\text{all } A_{i,j} = 0] = \left( 1 - \frac{1}{n} \right)^{m \cdot k} \]

\[ e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \ldots \]

When \( x \) is small (\( x = \pm \frac{1}{n} \)),

\[ \frac{x^2}{2} + \frac{x^3}{6} + \ldots \] don't matter.

\[ e^x \approx 1 + x \]

\( x = -\frac{1}{n} \)

\( (1 - \frac{1}{n}) = (1 + x) \approx e^x = e^{-\frac{1}{n}} \)
\[
\Pr \left[ A[0] = 0 \right] = (1 - \gamma_n)^m k
\]
\[
(1 - \gamma_n) \approx e^{-\gamma_n}
\]
\[
P = (1 - \gamma_n)^m k \approx (e^{-\gamma_n})^m k = e^{-mK/n}
\]

Pr of error?

\[
\begin{array}{cccc}
1-P & 1-P & 1-P \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1
\end{array}
\]

all \( k \) hash values happen to be ones

\[
\Pr \left[ \text{error} \right] = (1-P)^k
\]
Turns out, optimal \( p = \frac{1}{2} \)

\[
\Pr[\text{Error}] = (1 - p)^k = \left( \frac{1}{2} \right)^k
\]

\[
\frac{1}{2} = e^{-mk/n}
\]

\[
\Rightarrow k = \frac{n}{m} \ln 2
\]

\[\text{Ex.}\]

\[
\begin{align*}
    n &= 8m - 8 \text{ bits per string} \\
    k &= 8 \cdot \ln 2 \approx 6 \text{ hash functions} \\
    \Pr[\text{Error}] &\approx 1\% 
\end{align*}
\]

\[\text{Ex.}\]

\[
\begin{align*}
    n &= 32m - 32 \text{ bits per string} \\
    k &= 22 \text{ hash functions} \\
    \Pr[\text{Error}] &\approx 10^{-7}
\end{align*}
\]