Data Structures

Stack, Queue, List, Heap

Search Trees

Hash Tables
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)
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

Preprocessing
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 → Server (SMTP)
Password Hashing

haveibeenpwned.com: Your account has been compromised
PASSWORD HASHING

users

login/pwd

login/hash(pwd)
PASSWORD HASHING

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

haveibeenpwned.com: Your account has been compromised

hash(qwerty)=1xe4ht
hash(111111)=nh83l0
(Cryptographic) hash function maps strings to strings such that it’s hard to invert
(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)
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
Let $f: \{1, \ldots, N\} \rightarrow \{1, \ldots, N\}$ be a bijection.
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
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

\[ \sqrt{N} \]
INVERTING A BIJECTION
Store $x$ landmarks,
INVERTING A BIJECTION

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

Store \( \times \) landmarks, and links \( \Rightarrow \) to previous landmarks.

Space \( S \approx \sqrt{N} \)
Store $x$ landmarks, and links $\Rightarrow$ to previous landmarks.

Space $S \approx \sqrt{N}$
INVERTING A BIJECTION

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

Space $S \approx \sqrt{N}$

Time $T \approx \sqrt{N}$:
Store $x$ landmarks, and links $\mapsto$ to previous landmarks. Space $S \approx \sqrt{N}$, time $T \approx \sqrt{N}$: Invert $y = f(x)$.
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 $\rightarrow$ to previous landmarks space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Invert $y = f(x)$
Store $x$ landmarks, and links $\rightsquigarrow$ 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 $\xrightarrow{\text{previous landmarks}}$ 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 $y = f(x)$
Inverting a Bijection

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

Store $x$ landmarks, and links $\sim$ to previous landmarks

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$
DATA STRUCTURE

• Let $ST = N$
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: $T$
Let ST = N

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

Partition the graph into cycles
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 $\leq T$
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
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
DATA STRUCTURE

• We want a data structure that supports two functions

  • Insert($x$)
  • Lookup($x$)

  • Hash tables: less efficient but don't make mistakes

  • Bloom filter uses array of $n$ bits $A[0], \ldots, A[n-1]$, initialized with zeros

  • We'll use $k = O(1)$ hash 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\((x)\)
• We want a data structure that supports two functions
  • Insert\(x\)
  • Lookup\(x\)
• Hashtables: less efficient but don’t make mistakes
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)
• 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
• We’ll use $k = O(1)$ hash functions
Hash Functions

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

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

- Lookup($x$):
  - return 1 iff for every $i = 1, \ldots, k$, $A[f_i(x)] = 1$
Bloom Filter

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

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