Gems of TCS

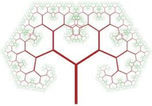
Data Structures

Sasha Golovnev

September 6, 2022

Data Structures

Stack, Queue, List, Heap



Search Trees

```
hash(unsigned x) {
   x ^= x >> (w-m);
   return (a*x) >> (w-m);
}
```

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

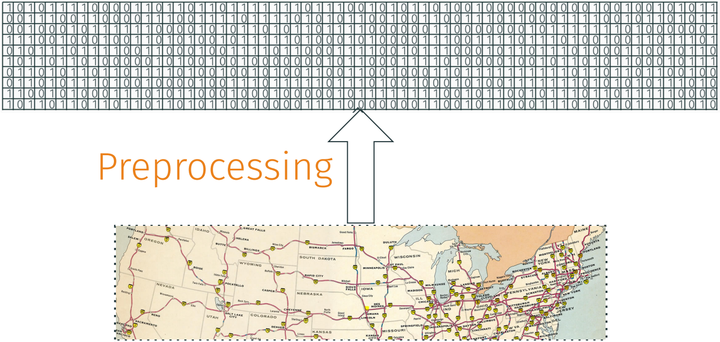
Examples

- 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



Data Structures

Queries

Preprocessing

Data Structures

Queries

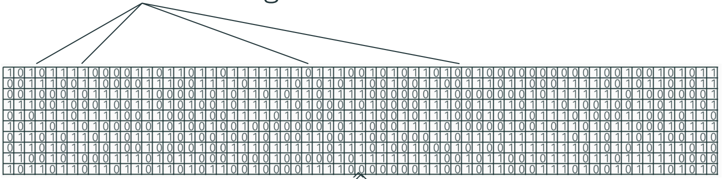
New York — Washington

Preprocessing

Data Structures

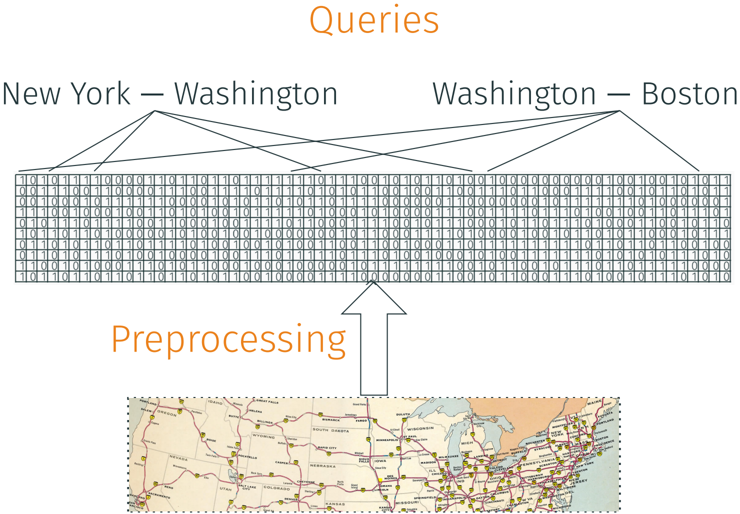
Queries

New York — Washington



Preprocessing

Data Structures



Stealing Passwords

Password Hashing

Password Hashing

haveibeenpwned.com:
Your account has
been compromised

Password Hashing

Password Hashing

hash(qwerty)=1xe4ht

hash(111111)=nh83l0

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

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)

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 $f\colon \{1, \ldots, N\} \to \{1, \ldots, N\}$ that are bijections

Inverting a Bijection

- Let $f\colon \{1, \ldots, N\} \to \{1, \ldots, N\}$ be a bijection

Inverting a Bijection

- Let $f\colon \{1,\ldots,N\} \to \{1,\ldots,N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$

Inverting a Bijection

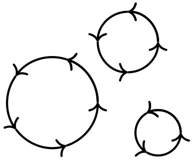
- Let $f\colon \{1,\ldots,N\} \to \{1,\ldots,N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$

Inverting a Bijection

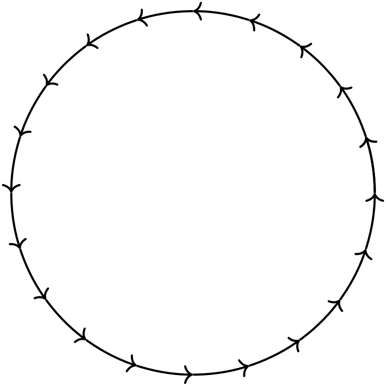
- Let $f\colon \{1,\dots,N\} \to \{1,\dots,N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- In- and out-degrees of all vertices are 1

Inverting a Bijection

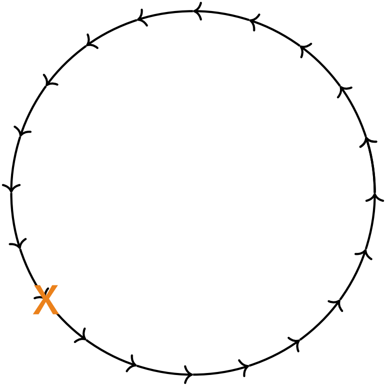
- Let $f\colon \{1,\ldots,N\} \to \{1,\ldots,N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- In- and out-degrees of all vertices are 1
- Thus, this graph is a union of cycles



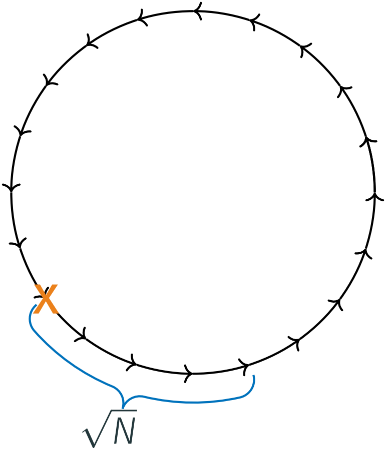
Inverting a Bijection



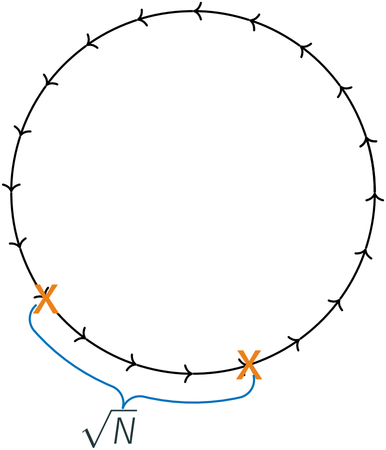
Inverting a Bijection



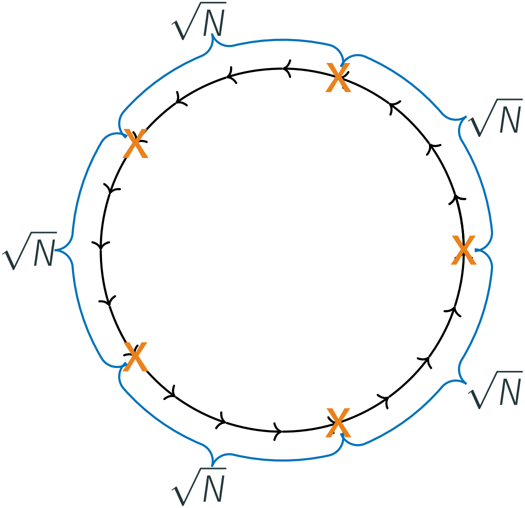
Inverting a Bijection



Inverting a Bijection

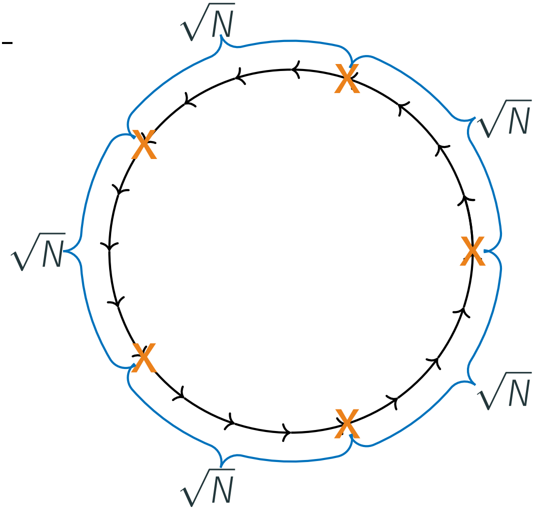


Inverting a Bijection



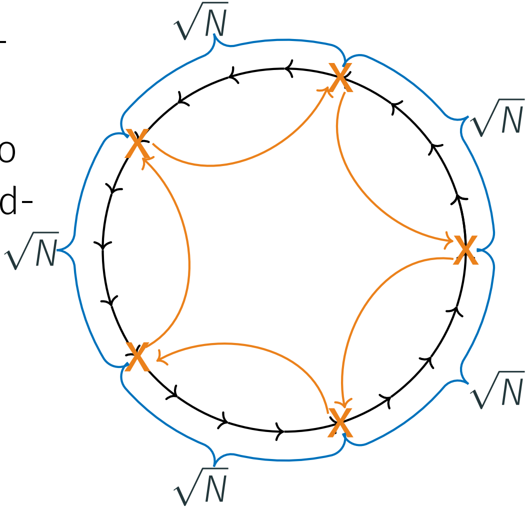
Inverting a Bijection

Store x land-
marks,



Inverting a Bijection

Store x land-marks,
and links to previous land-marks

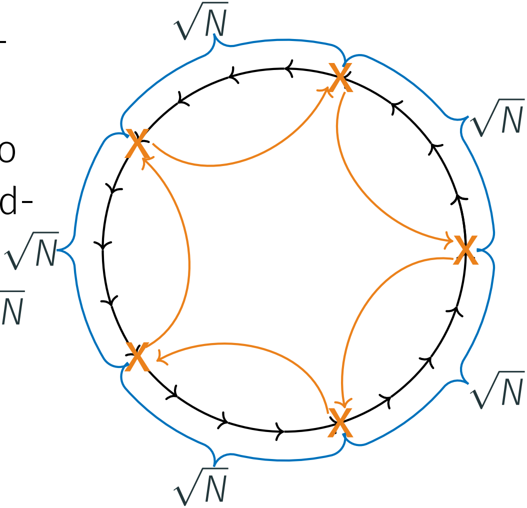


Inverting a Bijection

Store x land-marks,

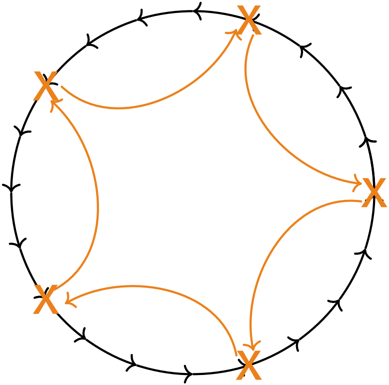
and links to previous land-marks

space $S \approx \sqrt{N}$



Inverting a Bijection

Store x land-
marks,
and links to
previous land-
marks
space $S \approx \sqrt{N}$



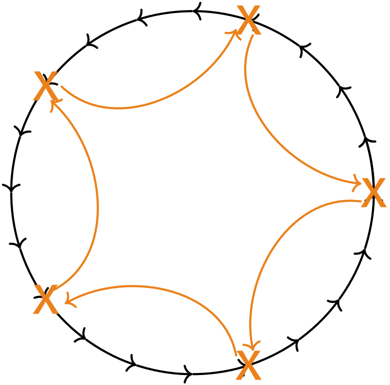
Inverting a Bijection

Store x land-marks,

and links to previous land-marks

space $S \approx \sqrt{N}$

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

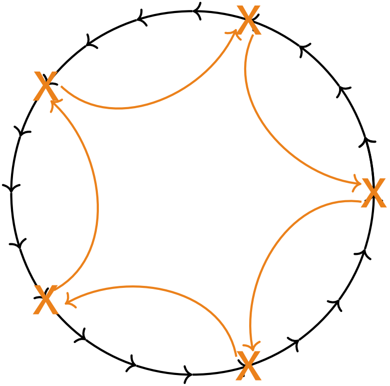


Inverting a Bijection

Store x land-marks,
and links to previous land-marks

space $S \approx \sqrt{N}$

time $T \approx \sqrt{N}$: Invert $y = f(x)$



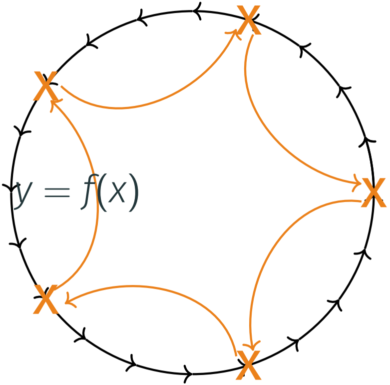
Inverting a Bijection

Store x landmarks,

and links to previous landmarks

space $S \approx \sqrt{N}$

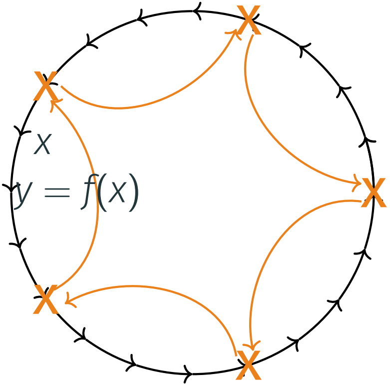
time $T \approx \sqrt{N}$: Invert $y = f(x)$



Inverting a Bijection

Store x land-marks,
and links to previous land-marks

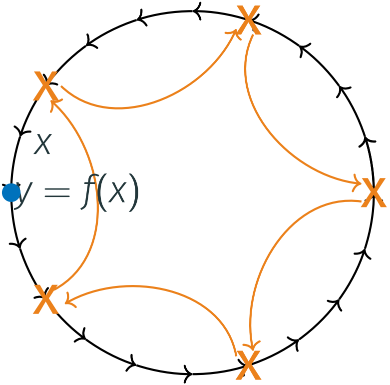
space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Invert $y = f(x)$



Inverting a Bijection

Store x land-
marks,
and links to
previous land-
marks

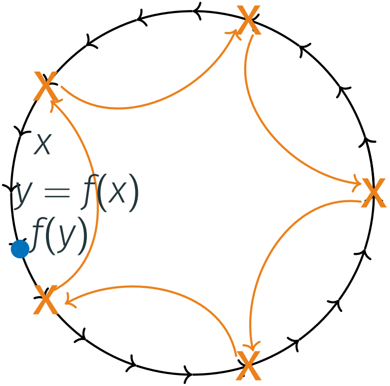
space $S \approx \sqrt{N}$
time $T \approx \sqrt{N}$:
Invert $y = f(x)$



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

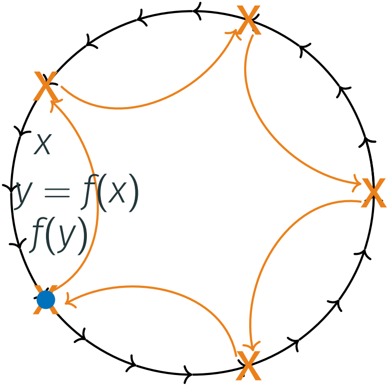
Store x landmarks,

and links to previous landmarks

space $S \approx \sqrt{N}$

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

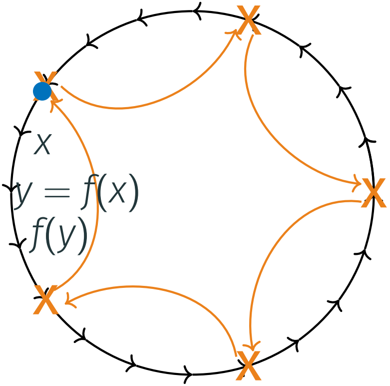
Invert $y = f(x)$



Inverting a Bijection

Store x land-
marks,
and links to
previous land-
marks

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



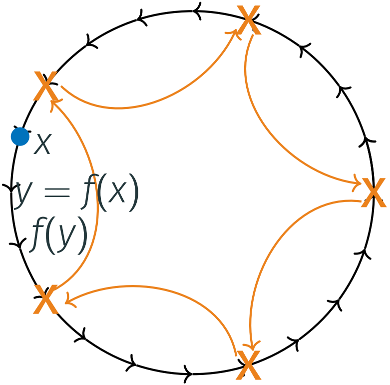
Inverting a Bijection

Store x land-marks,

and links to previous land-marks

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 $x \to f(x)$

Data Structure

- Let $ST = N$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- Partition the graph into cycles

Data Structure

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

Data Structure

- Let $ST = N$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- Partition the graph into cycles
- Ignore cycles of length $\leq T$
- In all other cycles store every Tth 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 operations

Data Structure

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

Data Structure

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

Data Structure

- We want a data structure that supports two operations
    - Insert(x)
    - Lookup(x)
- Hashtables: less efficient but don't make mistakes

Data Structure

- We want a data structure that supports two operations
  - Insert(x)
  - Lookup(x)
- Hashtables: less efficient but don't make mistakes
- Bloom fitler 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 operations
    - Insert(x)
    - Lookup(x)
- Hashtables: less efficient but don't make mistakes
- Bloom fitler 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 $f_1, \dots, f_k$ from strings to $\{0, \dots, n-1\}$
- Assume that functions are independent and uniform random

Bloom Fitler

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

Bloom Fitler

- 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