Gems of TCS

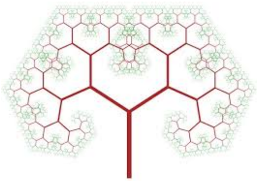
Data Structures

Sasha Golovnev

February 9, 2021

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)

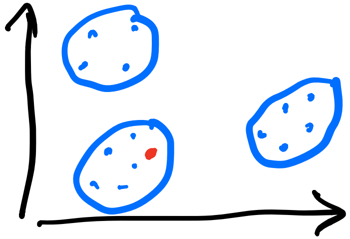
Preprocessing takes forever

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

I. Preprocessing

a lot time

Data Structures

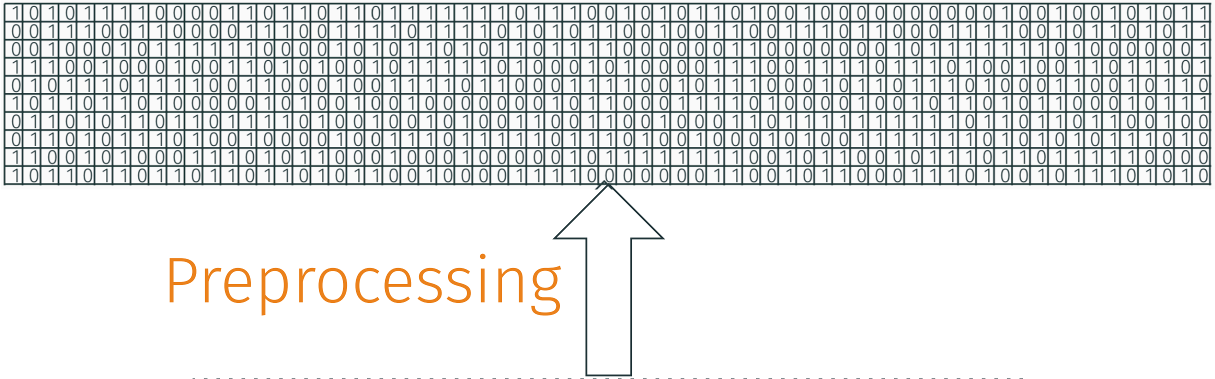
Queries

Preprocessing

Data Structures

Queries

New York — Washington

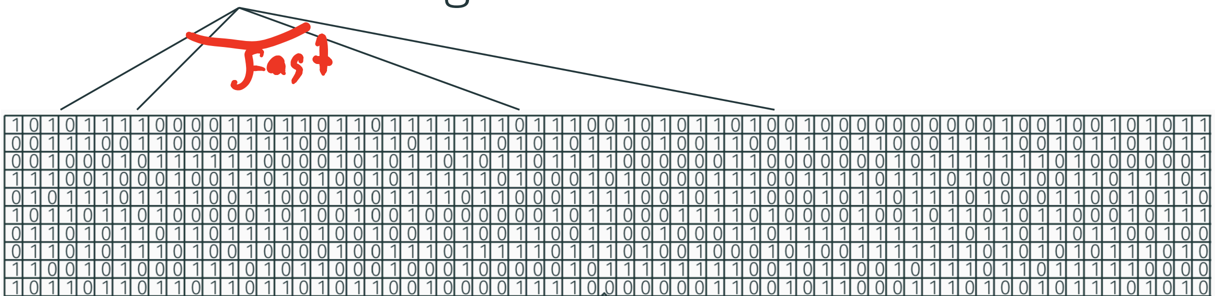


Preprocessing

Data Structures

Queries

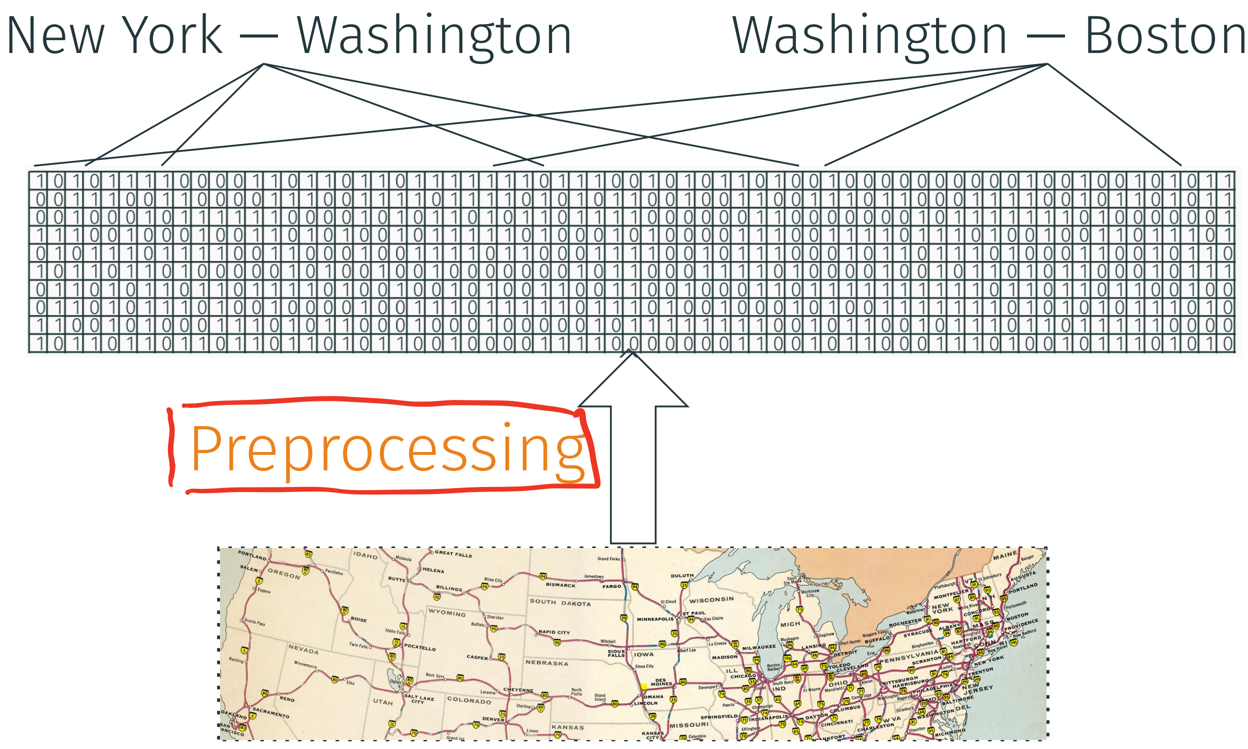
New York — Washington



Preprocessing

Data Structures

Queries

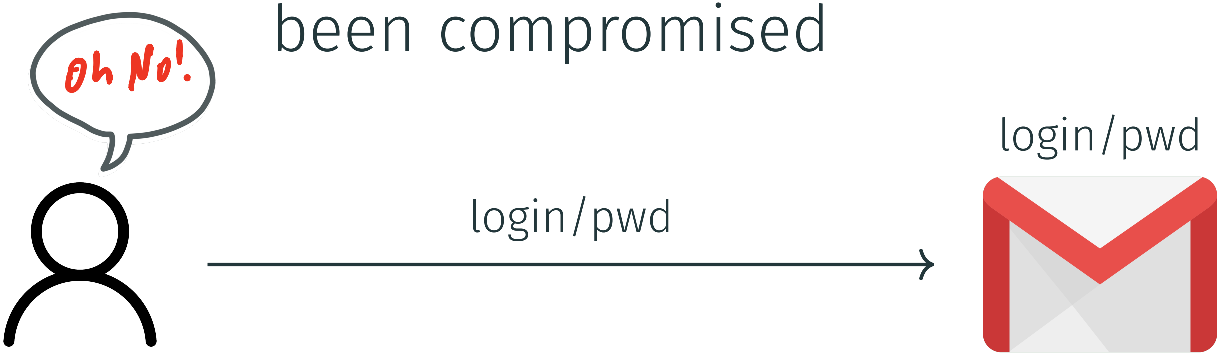


Stealing Passwords

Password Hashing

Password Hashing

haveibeenpwned.com:
Your account has
been compromised



Password Hashing

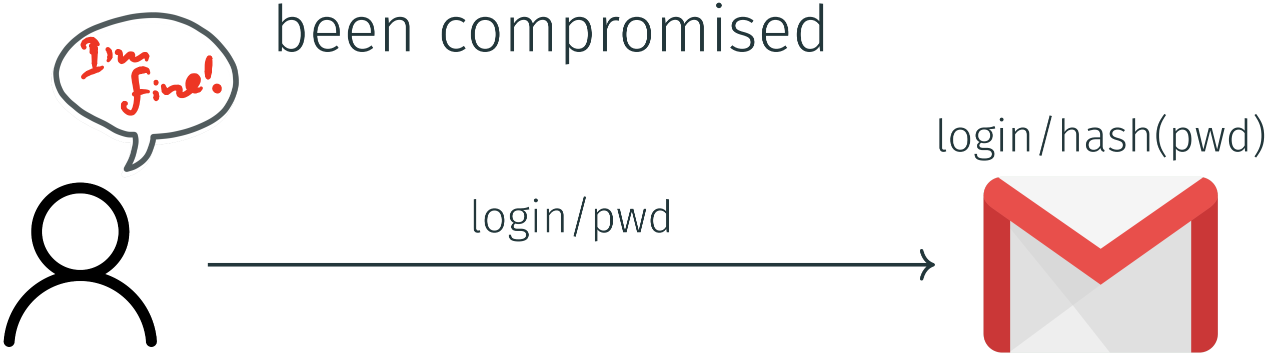
Password Hashing

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

pwd

hash

Ideally, in order to find pwd → 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

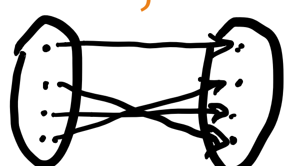
- (Cryptographic) hash function maps strings to strings such that it's hard to invert
- Ideally, to find a password that leads to a fixed hash value, one needs to brute force all possible passwords
- Hash functions are publicly known (SHA-3)

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

$f: \{0,1\}^n \to \{0,1\}^n$ $\quad N = 2^n$



Inverting a Bijection

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

Preprocess: years

Query: hash value → pwd

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

find x s.t. $f(x) = Y$

2 Naive solution

$N = 2^{256} \approx 10^{77}$

I. No preprocessing

Space $= 0$

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

$< 10^{20}$ operations per second

the age of Universe 10^{15} seconds

II. Preprocess: store

hash $\rightarrow$ pwd

$0 \ldots 0 \rightarrow pwd_1$

$0 \ldots 01 \rightarrow pwd_2$

$\vdots$

$1 1 1 1 1 \rightarrow pwd_N$

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

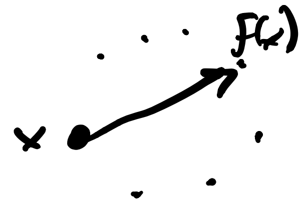
Time $= \log N$

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

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 $S = \sqrt{N}$

Directed Graph with N vertices



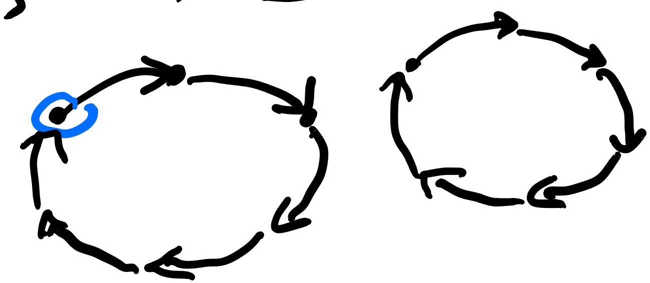
N directed edges

$x \in \{1, \ldots, N\}$

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 $x \to f(x)$

out-degree $(x) = 1$
in-degree $(x) = 1$

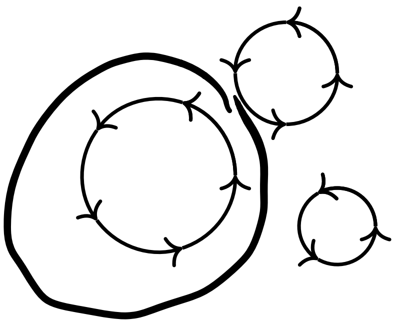


Inverting a Bijection

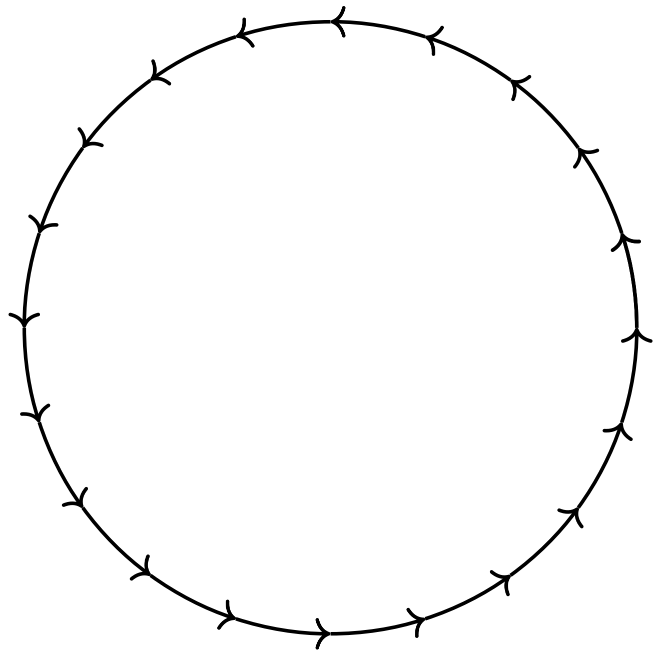
- Let $f\colon \{1,\ldots,N\} \to \{1,\ldots,N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- In- and out-degrees of all vertices are 1

Inverting a Bijection

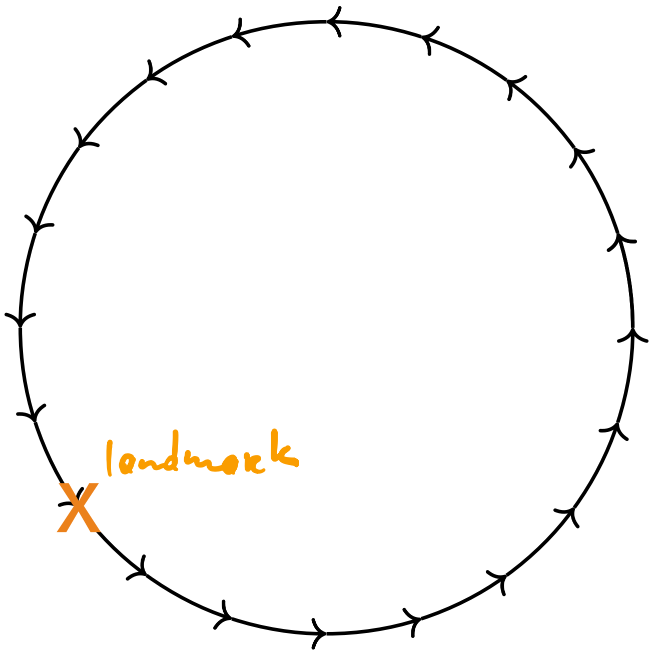
- Let $f\colon \{1, \ldots, N\} \to \{1, \ldots, N\}$ be a bijection
- Invert it in time $T = \sqrt{N}$ and space $S = \sqrt{N}$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- In- and out-degrees of all vertices are 1
- Thus, this graph is a union of cycles



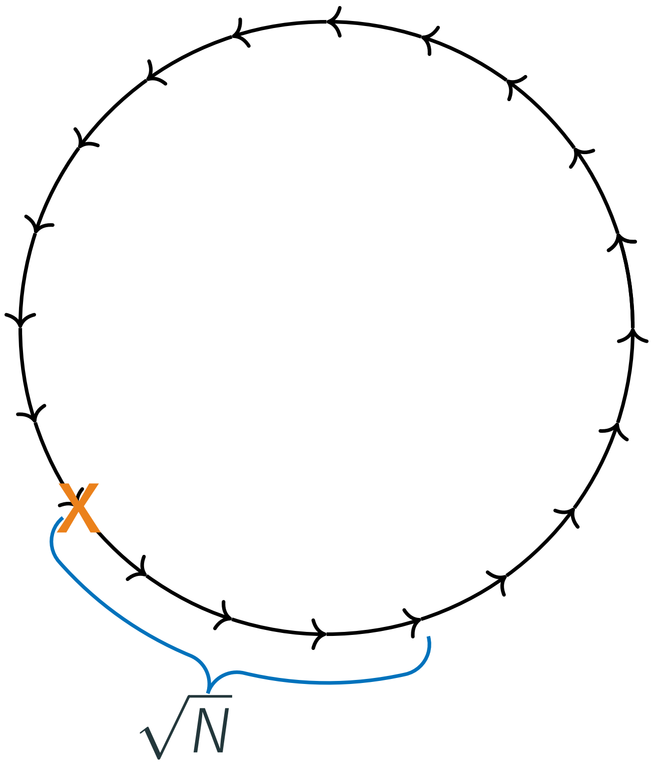
Inverting a Bijection



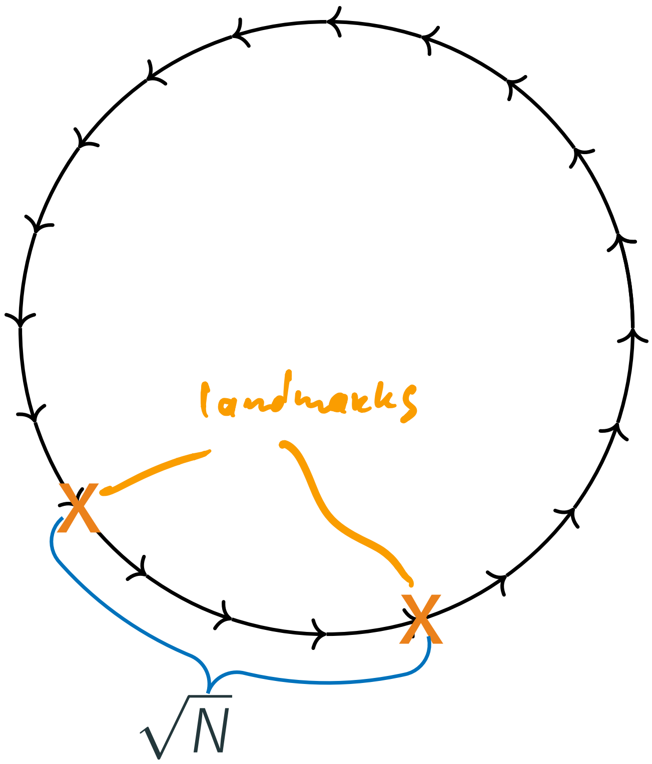
Inverting a Bijection



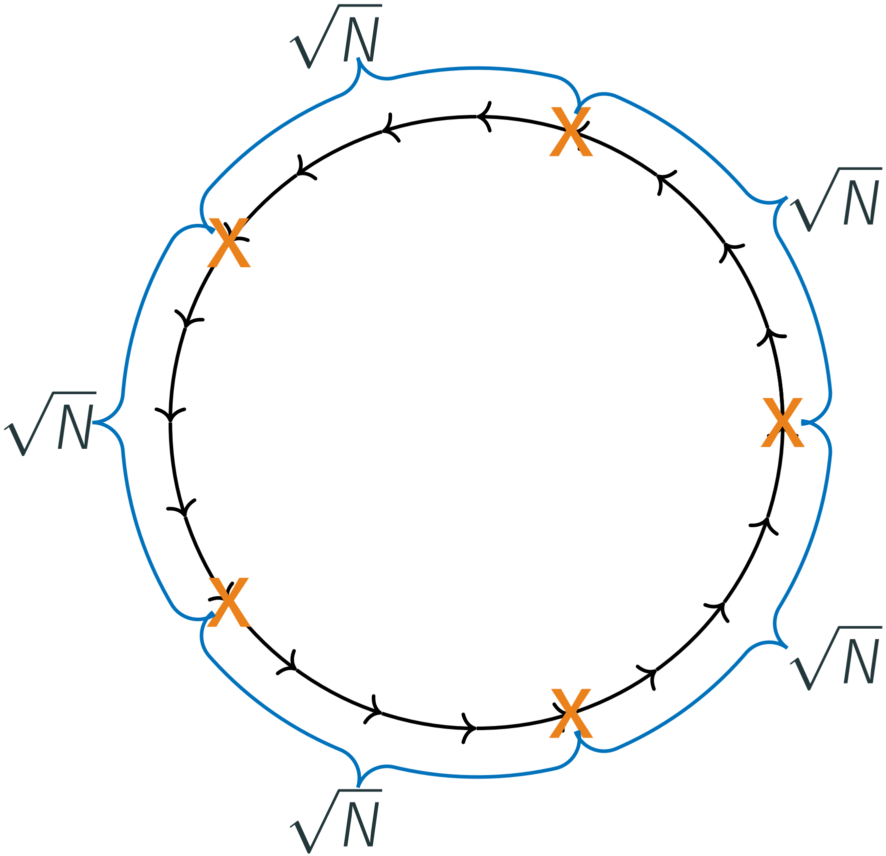
Inverting a Bijection



Inverting a Bijection

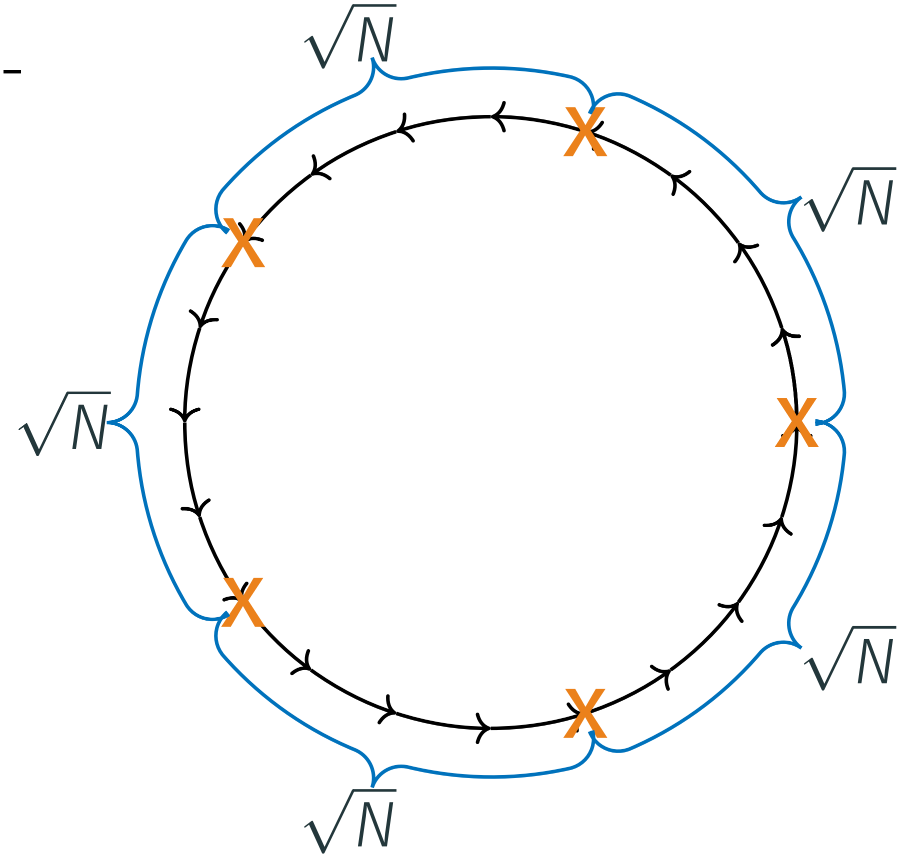


Inverting a Bijection



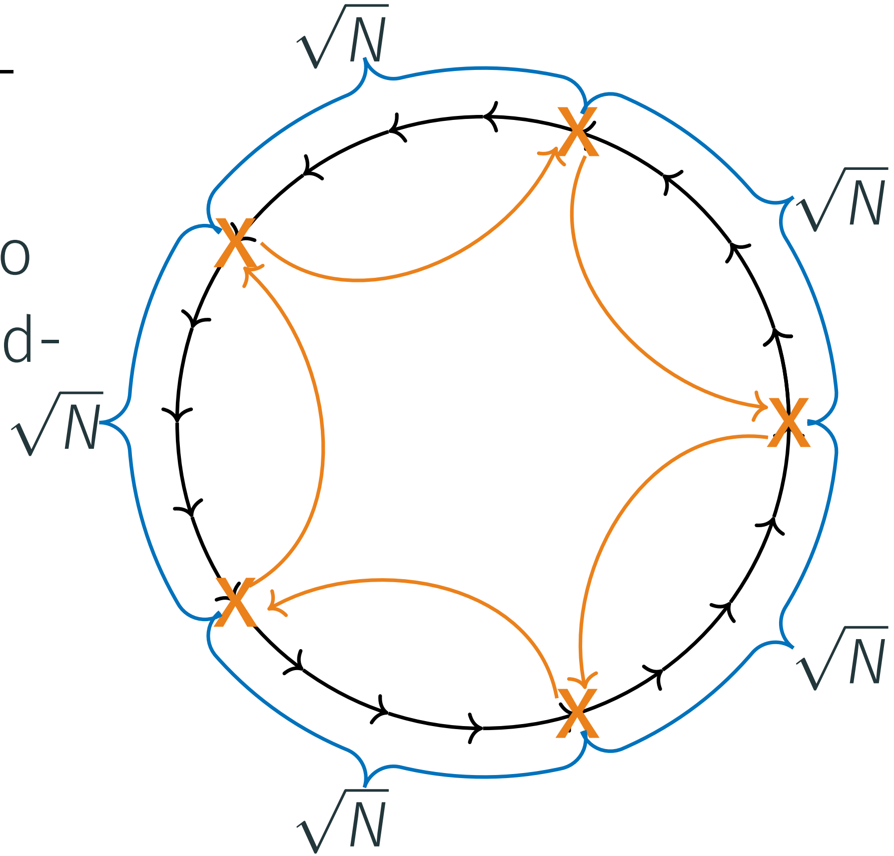
Inverting a Bijection

Store x land-marks,



Inverting a Bijection

Store x landmarks,
and links ↶ to previous landmarks



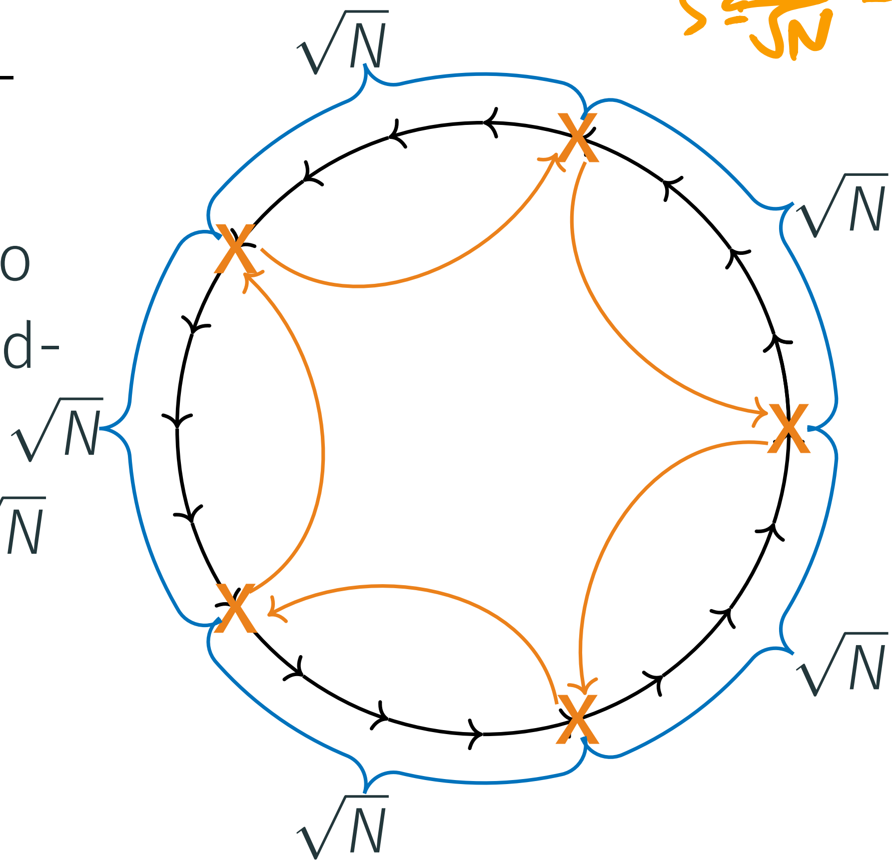
Inverting a Bijection

$S \leq \frac{N}{\sqrt{N}} = \sqrt{N}$

Store x landmarks,

and links ↶ to previous landmarks

space $S \approx \sqrt{N}$

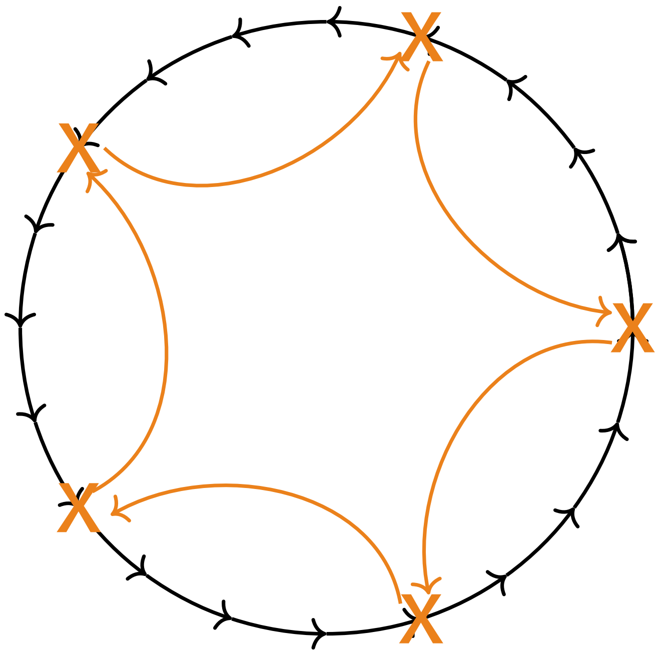


Inverting a Bijection

Store x landmarks,

and links ↺ to previous landmarks

space $S \approx \sqrt{N}$



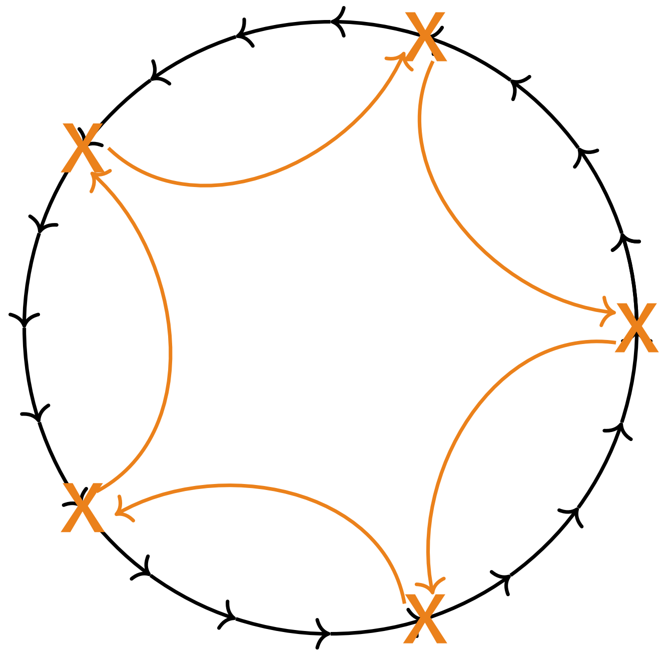
Inverting a Bijection

Store x landmarks,

and links to previous landmarks

space $S \approx \sqrt{N}$

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



Inverting a Bijection

Store x landmarks,

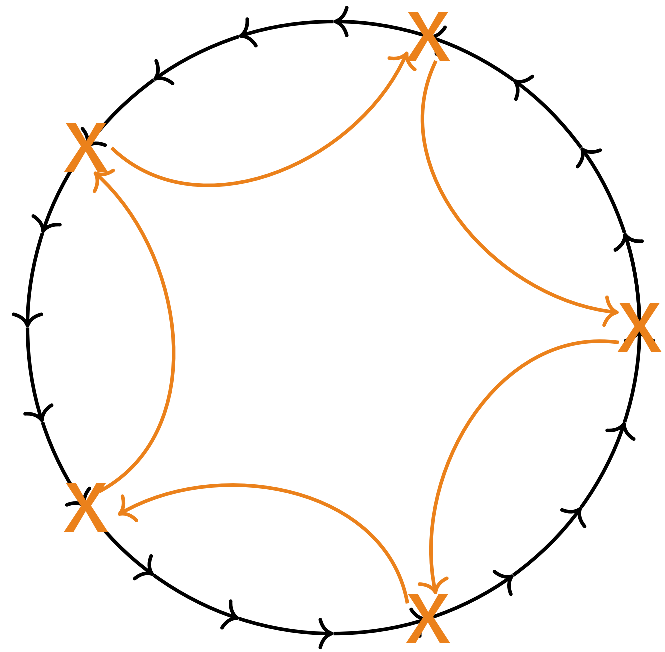
and links ↶ to previous landmarks

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$

given hash, find pwd

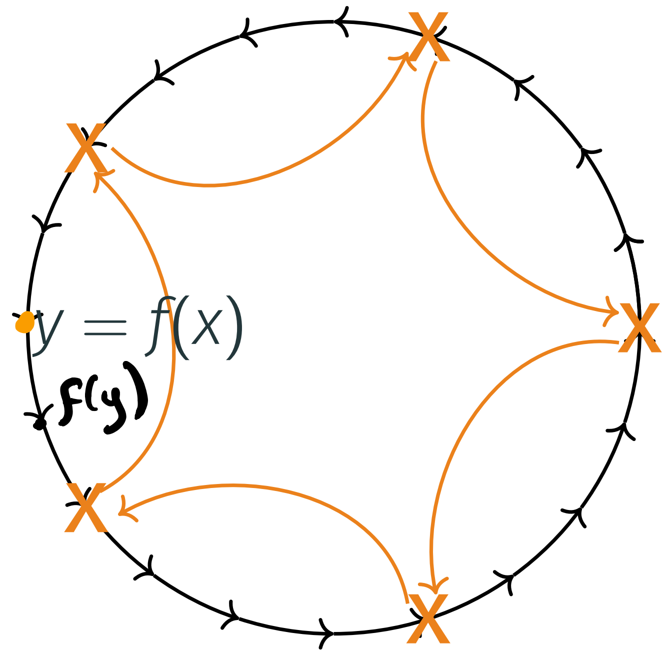


Inverting a Bijection

Recall F is publicly known
F(y) $y \to f(y)$

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

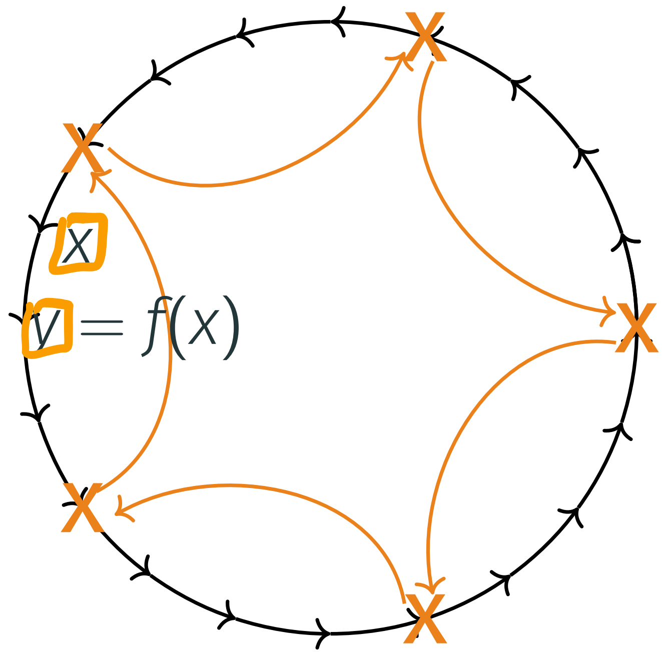
Store x land-marks,

and links ↰ to previous land-marks

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$



Inverting a Bijection

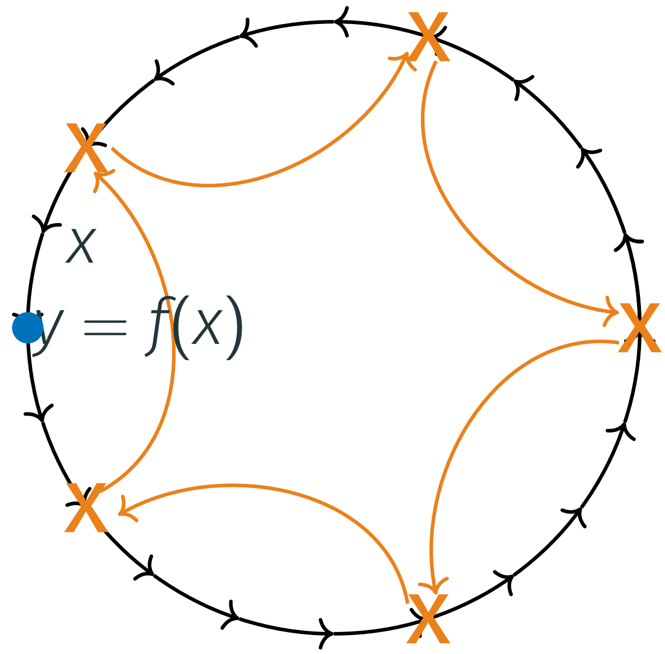
Store x land-marks,

and links ⤴ to previous land-marks

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$



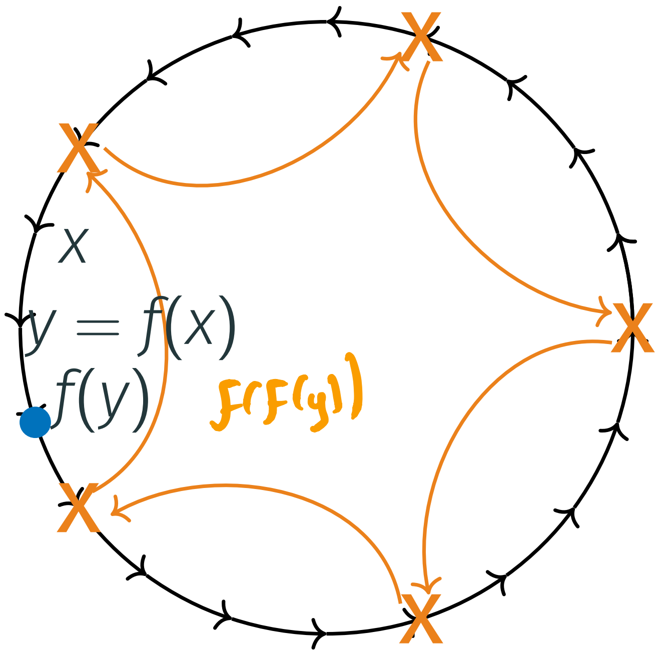
Inverting a Bijection

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

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$



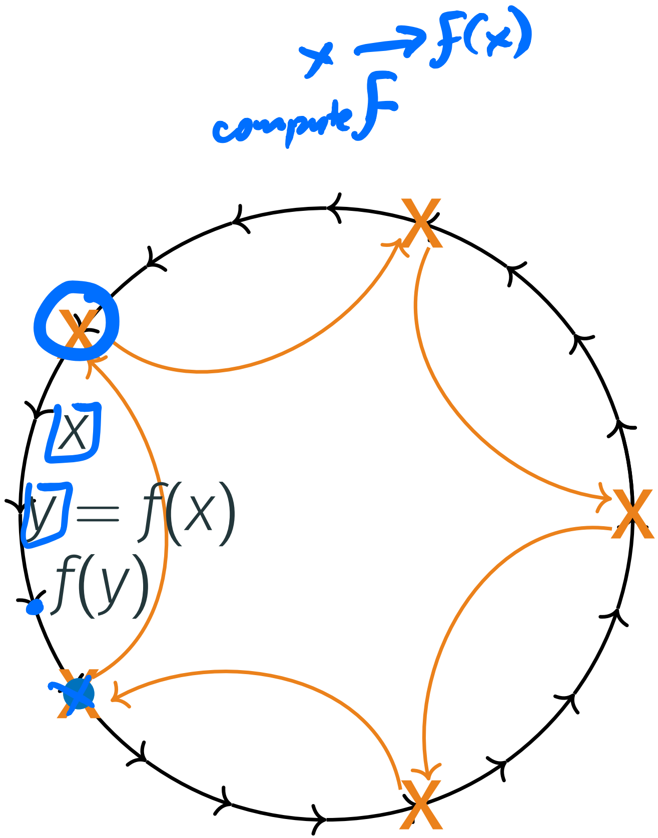
Inverting a Bijection

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

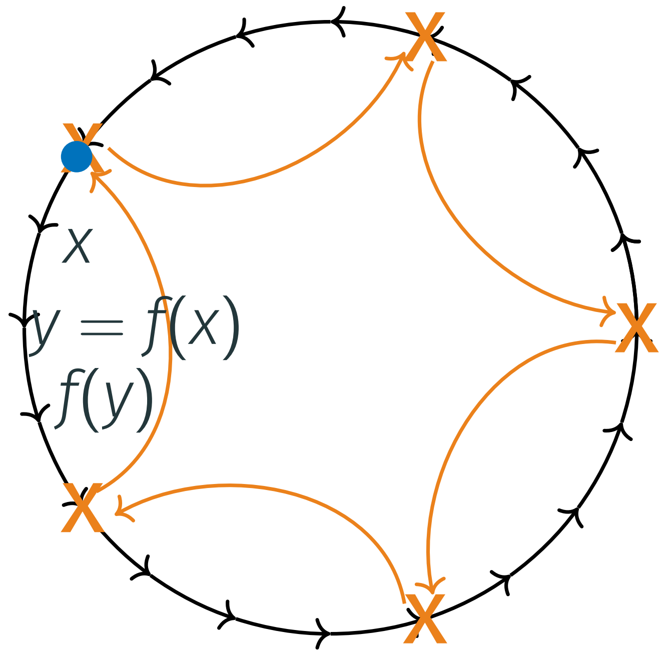
Store x land-marks,

and links ↰ to previous land-marks

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$



Inverting a Bijection

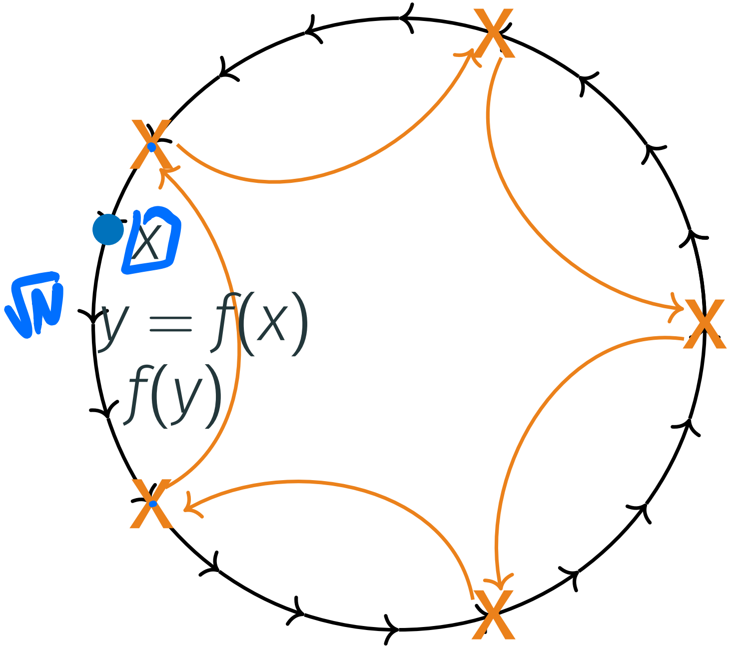
Store x landmarks,

and links to previous landmarks

space $S \approx \sqrt{N}$

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

Invert $y = f(x)$



Data Structure

space — query time

- Let $ST = N$

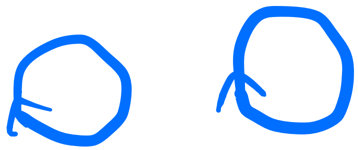
Ex. $S = N^{1/3}$ $T = N^{2/3}$

Data Structure

- Let $ST = N$
- Let's define a directed graph on N vertices with edges $x \rightarrow f(x)$

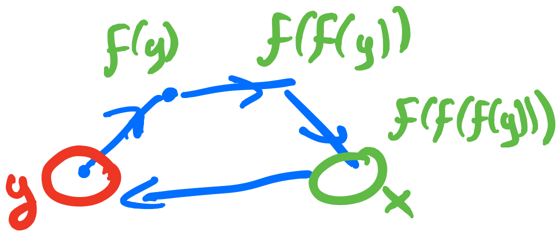
Data Structure

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



Data Structure

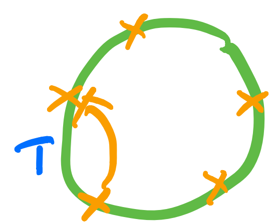
- Let $ST = N$
- Let's define a directed graph on N vertices with edges $x \to f(x)$
- Partition the graph into cycles
- Ignore cycles of length $\leq T$



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

$S \leq N/T$



Data Structure

someone publishes $\sqrt{N}$ carefully chosen hash value

- 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

Rainbow tables attacks

Now use hash functions: big N

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

losing information

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

Data Structure

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

prohibited pwd

Data Structure

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

Naive: list of prohibited pwd

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

we're storing $\approx m$ bits

store $\approx m \log m$ bits

Data Structure

- We want a data structure that supports two functions
    - 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 functions
  - 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

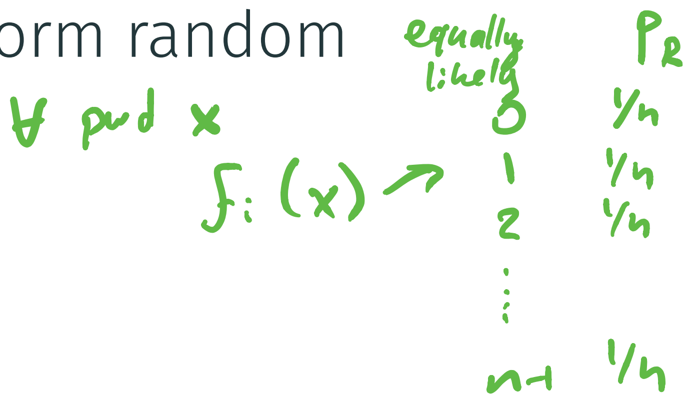
Hash Functions

prohibited pwds

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

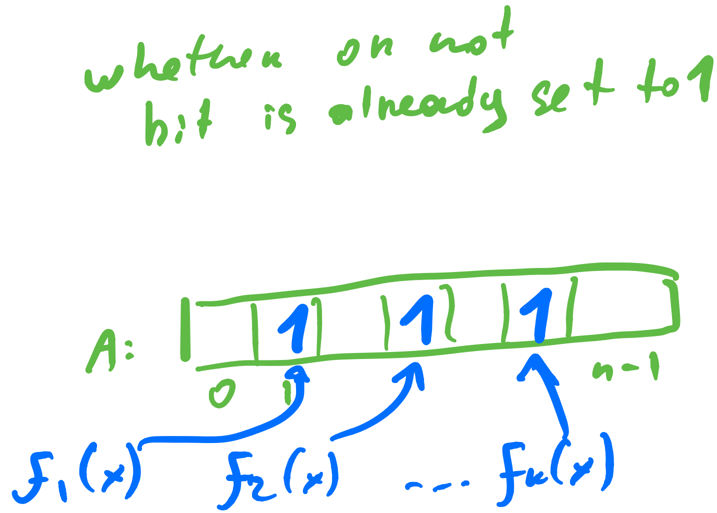
Hash Functions

- 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



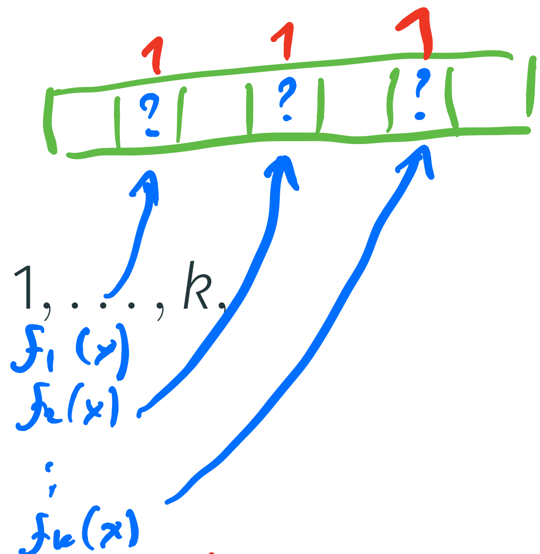
Bloom Fitler

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

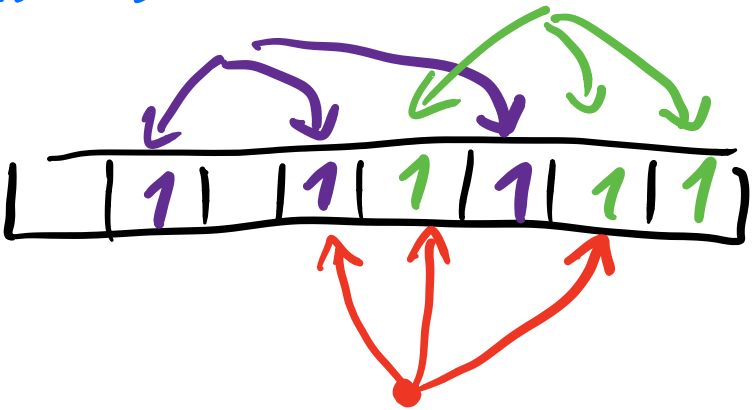


Obs: x was inserted, lookup(x) correctly says x is in data structure

Problem:

Analysis

Problem: x was not inserted
Lookup(x) may sometimes say that
x was inserted



Parameters

- m – # of prohibited pwds
- n – # of bits in array
- k – # of hash funs

Remains: analyze Pr of mistake

Analysis to set params n, k.

We've inserted all m strings

$$\Pr[A[0]=0] = ?$$

Insert 1st pwd:

$f_1 \leftarrow \Pr[A[0]=0] = 1 - 1/n$

⋮

$f_k \leftarrow \Pr[A[0]=0] = 1 - 1/n$

- - - - - - - -

mth pwd

f_1

⋮

f_k

$\Pr[A[0]=0] = 1 - 1/n$

$\Pr \ (1 - 1/n)^{m \cdot k}$ $A[0] = 0$ even after I inserted m els.

$\Pr[A[1] = 0] = (1 - 1/n)^{m \cdot k}$

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

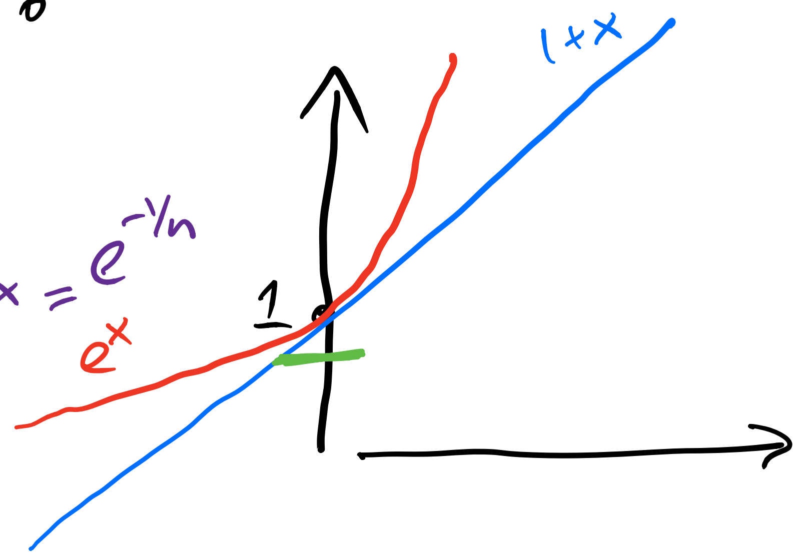
When x is small ($x = \pm 1/n$),

$\frac{x^2}{2} + \frac{x^3}{6} + \ldots$ don't matter

$e^x \approx 1 + x$

$x = -1/n$

$(1 - \frac{1}{n}) = (1 + x) \approx e^x = e^{-1/n}$

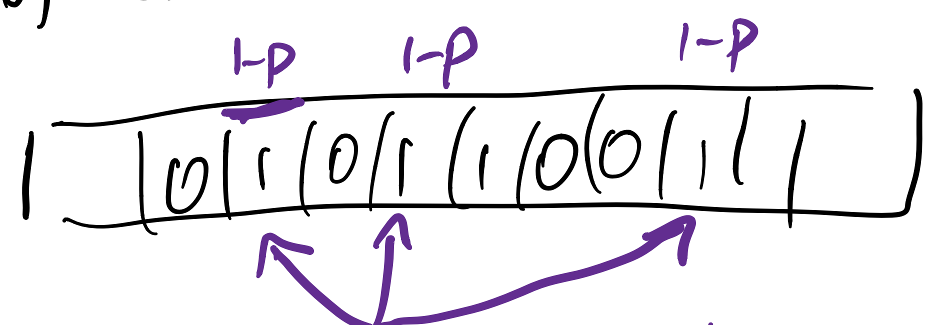


$$\Pr[A[0]=0] = (1-1/n)^{m \cdot k}$$

$$(1-1/n) \approx e^{-1/n}$$

$$p = (1-1/n)^{mk} \approx \left(e^{-1/n}\right)^{mk} = e^{-mk/n}$$

Pr of error?



all k hash values happen to be ones

$$\Pr[\text{error}] = (1-p)^k$$

Turns out, optimal $p = \frac{1}{2}$

$$\Pr[\text{error}] = (1-p)^k = \boxed{\frac{1}{2^k}}$$

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

$$\boxed{k} = \frac{n}{m} \ln 2$$

Ex. $n = 8m$ – 8 bits per string

$k = 8 \cdot \ln 2 \simeq 6$ hash func

$\Pr[\text{error}] \approx 1\%$

Ex. $n = 32m$ – 32 bits per string

$k \simeq 22$ hash funcs

$\Pr[\text{error}] \sim 10^{-7}$