Applications for Hashing

checksumming

hard disk/file system
Applications for Hashing

checksumming

duplicate detection

similarity search $LSH = \text{locality-sensitive hashing}$
Applications for Hashing

checksumming
duplicate detection
similarity search
encryption/signatures

SHA

HMAC
Applications for Hashing

checksumming
duplicate detection
similarity search
encryption/signatures
pseudo-randomization

\[ \text{seed} = 42 \]
Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures

pseudo-randomization

building block for query processing

point queries
Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures

pseudo-randomization

building block for query processing

point queries

range queries

\[ 42 \leq T \leq 7000 \]
Applications for Hashing

checksumming
duplicate detection
similarity search
encryption/signatures
pseudo-randomization
building block for query processing
point queries
range queries
...

Point Queries

given:

an integer domain 0, ..., N-1

N is potentially large

point queries asking for data items having a key in 0, ..., N-1

Example:

What is the address of the student having ID=424342?
Naïve Solution: Array

\[ \text{Data Item } [i] \text{ now } \text{Data Item } [i+N] \]

\[ 8 \times 1 \]
Array Methods

`insert(Integer key, DataItem value):` aka `put(..)` or `set(..)`

`array[key] = value;`

`delete(Integer key):` aka `remove(..)`

`array[key] = null;`

`pointQuery(Integer key):` aka `get(..)`

`return array[key];`
Space Usage

\[ N = 100,000,000 \]

\[ X = 10,000 \]

\[ N/X = 10,000 \]
Core Idea of Static Hashing

pick an $M \ll N$

allocate an array of size $M$ only

transform any incoming key from

$\text{range } 0, \ldots, N-1$ to $\text{range } 0, \ldots, M-1$
Core Idea of Static Hashing

pick an $M << N$

allocate an array of size $M$ only

transform any incoming key from

range $0, \ldots, N-1$ to range $0, \ldots, M-1$

this is achieved by a so-called hash function $h(key)$

$$h(key) \% M$$
Insert 424342

Data Item
ID=424342
...

\[ h(424342) = 77 \]

K \mapsto V
ID \mapsto Data Item
insert 3917564

slot 0  slot 424342  slot 3917564  slot N-1

bucket 0  bucket 6  bucket 77  bucket M-1

DataItem
ID= 3917564  ...

DataItem
ID=424342  ...

h(3917564) = 6
HashMap Methods (Arbitrary Type)

Note: this is ignoring collisions for the moment, see next slide for collisions.

**insert(Key key, DataItem value):**

array[ h(key) ] = value;

**delete(Key key):**

array[ h(key) ] = null;

**pointQuery(Key key):**

return array[ h(key) ];

aka put(..) or set(..)

aka remove(..)

aka get(..)

K → V
String → DataItem
h: String → Integer
Collisions? insert 3456789

slot 0 → slot 424342 → slot 3456789 → slot N-1

h(424342) = 77

h(3456789) = 77

collision

bucket 0 → bucket 6 → bucket 77 → bucket M-1

DataItem
ID= 3917564
...

DataItem
ID= 424342
...

PK

No Duplicates
Rehash!

pick an $M' > M$

create new hash table with $M'$

re-insert all entries from old hash table into new hash table