Applications for Hashing

checksumming

had disks/file systems
Applications for Hashing

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

checksumming
duplicate detection
similarity search
encryption/signatures
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 $I \Delta = 4 \pi$
Applications for Hashing

checksumming

duplicate detection

similarity search

encryption/signatures

pseudo-randomization

building block for query processing

point queries

range queries

$42 \leq \pi \leq 7000$
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

```
DataItem[424342] = new DataItem[11];
```

\[ 813 \times 11 \]
Array Methods

insert(Integer key, DataItem value):

array[key] = value;

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

delete(Integer key):

array[key] = null;

aka remove(..)

pointQuery(Integer key):

return array[key];

aka get(..)
Space Usage

\[ N = 100,000,000 \]
\[ X = 10,000 \]
\[ N/X = 10,000 \]
Core Idea of Static Hashing

pick an \( M < N \)

allocate an array of size \( M \) only

transform any incoming key from

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

pick an \( M < N \)

allocate an array of size \( M \) only

transform any incoming key from

\[ \text{range } 0, \ldots, N-1 \text{ to range } 0, \ldots, M-1 \]

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

\[ h(key) \% M \]
insert 3917564

h(3917564) = 6
HashMap Methods (Arbitrary Type)
Note: this is ignoring collisions for the moment, see next slide for collisions.

insert(Key key, Dataltem value):

array[ h(key) ] = value;

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

0(1)

delete(Key key):

array[ h(key) ] = null;

aka remove(..)

0(1)

pointQuery(Key key):

return array[ h(key) ];

aka get(..)

0(1)

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

`h(424342) = 77`
`h(3456789) = 77`

collision

Bucket 0
Bucket 6
Bucket 77
Bucket M-1

DataItem
ID = 3917564
...

DataItem
ID = 424342
...

No duplicates
Overflow Chains

slot 0  slot 424342  slot 3456789  slot N-1

\[ h(424342) = 77 \quad h(3456789) = 77 \]

collision

Bad Hash:

\[ \text{Hash} \]

\[ \frac{\text{streutabelle}}{\text{Gem} \quad 7} \]

\[ h(k) = 42 \]

\[ L_{77} = \text{null} \]

\[ (77) \]
Rehash!

pick an $M' > M$

create new hash table with $M'$

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