



Design and Analysis
of Algorithms I

Data Structures

Hash Tables: Some
Implementation Details

Hash Table: Supported Operations

Purpose : maintain a (possibly evolving) set of stuff.
(transactions, people + associated data, IP addresses, etc.)

Insert : add new record

Using a “key”

Delete : delete existing record

AMAZING
GUARANTEE

Lookup : check for a particular record
(a “dictionary”)

All operations in
 $O(1)$ time ! *

* 1. properly implemented 2. non-pathological data

High-Level Idea

Setup : universe U [e.g., all IP addresses, all names, all chessboard configurations, etc.]
[generally, REALLY BIG]

Goal : want to maintain evolving set $S \subseteq U$
[generally, of reasonable size]

Solution : 1.) pick $n = \#$ of “buckets” with
(for simplicity assume $|S|$ doesn't vary much)
2.) choose a hash function $h : U \rightarrow \{0, 1, 2, \dots, n - 1\}$
3.) use array A of length n , store x in $A[h(x)]$

Naïve Solutions

1. Array-based solution
[indexed by u]
- $O(1)$ operations
but $\theta(|U|)$ space
2. List-based solution
- $\theta(|S|)$ space but
 $\theta(|S|)$ Lookup

Consider n people with random birthdays (i.e., with each day of the year equally likely). How large does n need to be before there is at least a 50% chance that two people have the same birthday?

- ☒ 23 50 %
- ☐ 57 99 %
- ☐ 184 99.99....%
- ☐ 367 100%

BIRTHDAY
"PARADOX"

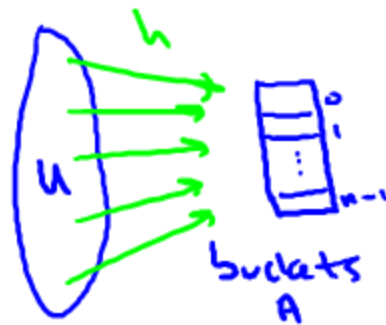
Resolving Collisions

Collision: distinct $x, y \in U$ such that $h(x) = h(y)$

Solution #1 : (separate) chaining

- keep linked list in each bucket
- given a key/object x , perform Insert/Delete/Lookup in the list in $A[h(x)]$

Linked list for x → Bucket for x



Solution #2 : open addressing. (only one object per bucket)

- Hash function now specifies probe sequence $h_1(x), h_2(x), \dots$

(keep trying till find open slot)

- Examples : linear probing (look consecutively), double hashing

Use 2 hash functions

What Makes a Good Hash Function?

Note : in hash table with chaining, Insert is $\theta(1)$
 $\theta(\text{list length})$ for Insert/Delete.

Insert new object x at
front of list in $A[h(x)]$

could be anywhere from m/n to m for m objects

Equal-length lists

Point : performance depends on the choice of hash function!
(analogous situation with open addressing)

All
objects in
same
bucket

Properties of a “Good” Hash function

1. Should lead to good performance \Rightarrow i.e., should “spread data out” (gold standard – completely random hashing)
2. Should be easy to store/ very fast to evaluate.

Bad Hash Functions

Example : keys = phone numbers (10-digits).

$$|u| = 10^{10}$$

-Terrible hash function : $h(x) = 1^{\text{st}} 3 \text{ digits of } x$
(i.e., area code)

$$\text{choose } n = 10^3$$

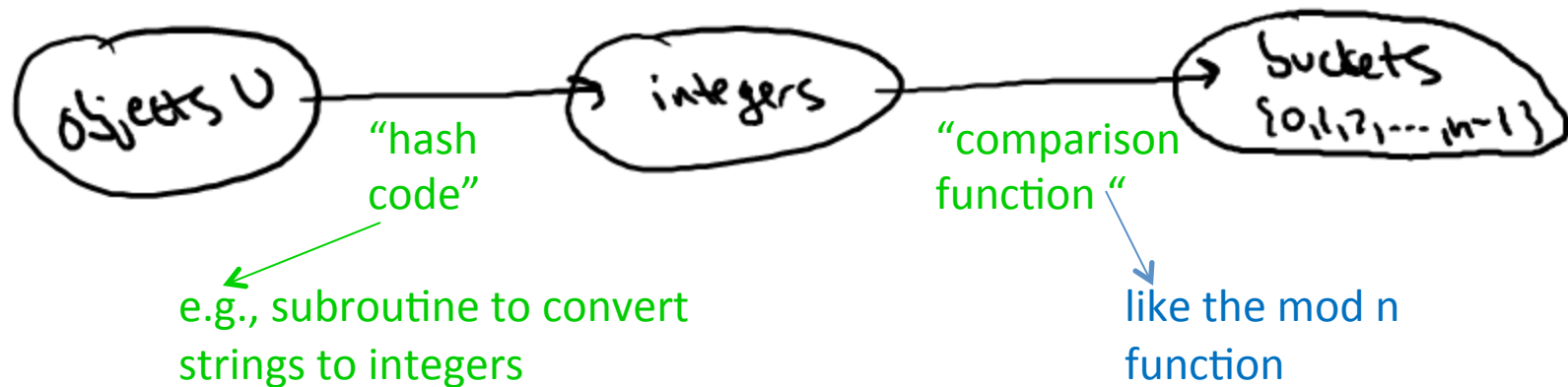
- mediocre hash function : $h(x) = \text{last 3 digits of } x$
[still vulnerable to patterns in last 3 digits]

Example : keys = memory locations. (will be multiples of a power of 2)

-Bad hash function : $h(x) = x \bmod 1000$ (again $n = 10^3$)

=> All odd buckets guaranteed to be empty.

Quick-and-Dirty Hash Functions



How to choose $n = \#$ of buckets

1. Choose n to be a prime (within constant factor of $\#$ of objects in table)
2. Not too close to a power of 2
3. Not too close to a power of 10