Data Structures

Hash Tables: Some Implementation Details

Design and Analysis of Algorithms I

Hash Table: Supported Operations

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

Insert: add new record

Delete: delete existing record

Lookup: check for a particular record

(a "dictionary")

using a "key"

AMAZING GUARANTEE

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 chess board 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 $n \approx |S|$ (for simplicity assume $|S|$ doesn't vary too much)

② choose a hash function $h: U \to \{0, 1, 2, \ldots, n-1\}$

③ use array A of length n, store x in $A[h(x)]$

Naive 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

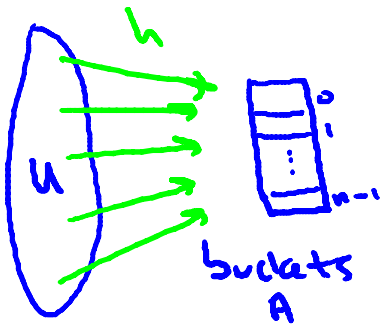
Tim Roughgarden

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"

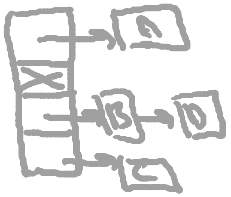
Resolving Collisions



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 $A[h(x)]$ (bucket for x; linked list for x)



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

- hash function now specifies probe sequence $h_1(x), h_2(x), \ldots$ (keep trying til 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 $O(1)$ [insert new object x at front of list in $A[h(x)]$]

$O(\text{list length})$ for Insert / Delete.

→ could be anywhere from $\frac{m}{n}$ to m for m objects

$\frac{m}{n}$ → equal-length lists

m → all objects in same bucket

Point: performance depends on the choice of hash function!

(analogous situation with open addressing)

Properties of a "Good" Hash function

① should lead to good performance ⇒ 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: $h(x)$ = last 3 digits of x (i.e., area code) Choose $n = 10^3$
- mediocre hash function: $h(x)$ = last 3 digits of x

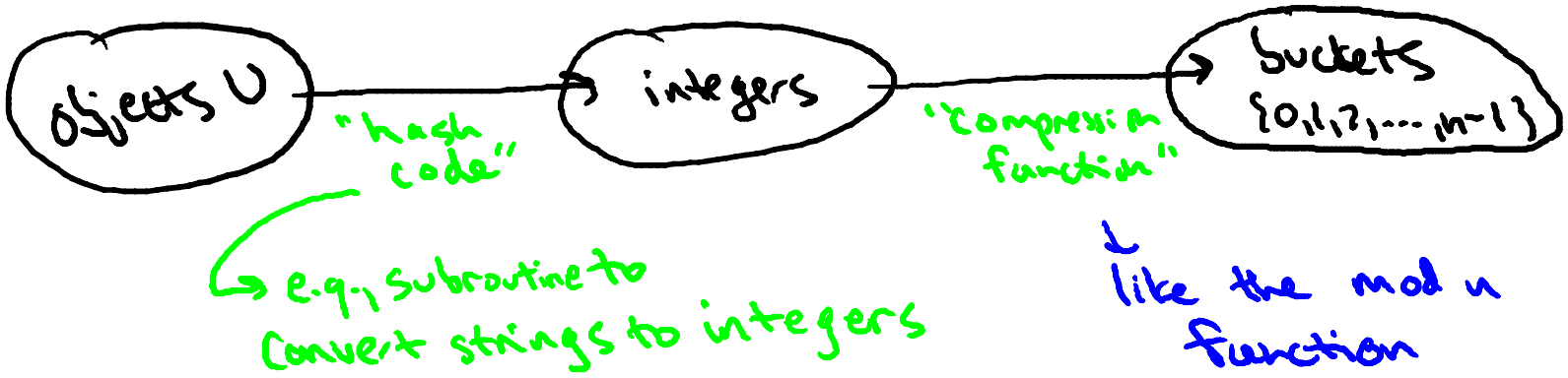
  [still vulnerable to patterns in last 3 digits]

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