6.006 R05

Agenda

- division
- multiplication
- python hash
- hash design

Reminders

- still holding office hours Monday
- PSET part A due TUESDAY!
- PSET2 released Tuesday.

---

want: simple uniform hashing – each key is equally likely to hash to any of the m slots, independantly of where other keys hash to.

Constructing Hash Functions

Division Method

size of table

$$\boxed{h(k) = k \bmod m}$$

collision: $k_1 \equiv k_2 \pmod{m}$ when m divides $|k_1 - k_2|$

- OK when keys uniform random over the integers.
- BAD when keys have a regularity like $(x, 2x, 3x, \dots)$ and x & m have common divisor d. then use $1/d$ of the table.
  - likely if m has a small divisor (like 2)

  [ex: $m = 12$, $x = 3, 6, 9 \dots$

- BAD if $m = 2^r$ then only consider r bits of key (lower bits)
- GOOD m is a prime
  - not close to power of 2 or 10 (to avoid common regularities)
  - !! finding primes is generally hard/inconvenient.

Multiplication Method

$$\boxed{h(k) = [(a \cdot k) \% 2^w] \gg (w - r)}$$

where $m = 2^r$ and w-bit machine words.
a = odd int $\in [2^{w-1}, 2^w]$

notes

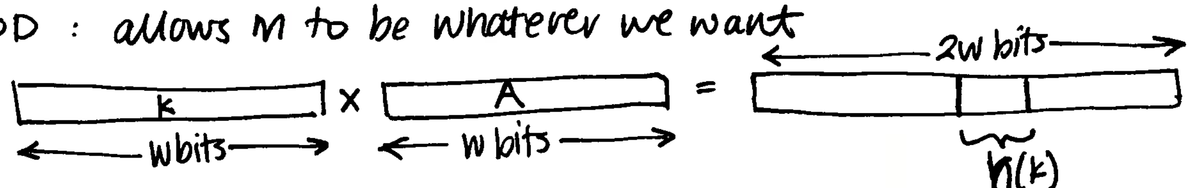
- GOOD: not too close to extremities of range

$$\boxed{h(k) = \lfloor m\,(k \cdot A \% 1) \rfloor}$$

$0 < A < 1$ extract fractional portion.

book

- GOOD: allows m to be whatever we want



3rd method in lecture

hash(key) method in Python

returns 32-bit int (may be negative)

(-1) $\Leftrightarrow$ NOT hashable.

a = hash(a) if a is an integer

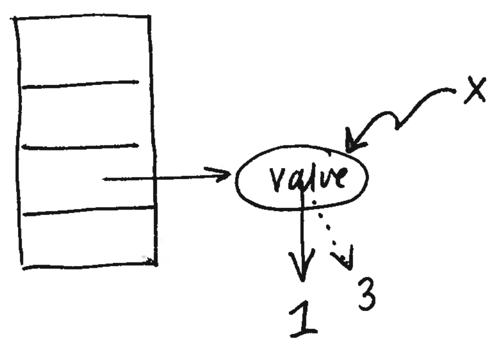
hash(x) = id(x) if x is an object

address of x

... and many more.

cannot use mutable objects

why?



insert x into dict.
gets hashed to slot 3
mutate x
now may hash elsewhen
but it's still located in 3!

Hashing non-integers

previously assumed $k \in \mathbb{N} = \{0\ 1\ 2\ 3\ \dots\}$

STRINGS : brainstorming / interactive

- treat it as an integer
  - using what representation? ASCII, Unicode, ...
- h(sum(h(string[i])))
- sum = 0
  for i in string: sum = A·sum + h(i)
  - where A is some well chosen # (prime usually)
- primes = [2 3 5 7 11 ...], prod = 1.
  for i in range(len(string)):
  prod *= pow(primes[i], string[i])
- ...

check for
- obvious non-uniformities
- inefficiencies.
- false assumptions.

- common idea in many.

```
h=0
for i in string:
    modify h using new input i
```

CRC PJW BUZ

link: http://www.cs.hmc.edu/~geoff/classes/hmc.cs070.200101/homework10/hashfuncs.html

SEQUENCES

order matters $h(1,2,3) \neq h(3,2,1)$

- $(p_1^{s_1})(p_2^{s_2})(p_3^{s_3}) \cdots$ p_i = ith prime, s_i = ith element of sequence.

  % m?

- hash(1111 , 2112 , 3113) or hash (1113 , 2112 , 3111)
  set hash: aka these the order doesn't matter, b/c we encoded it in the new keys.
- concate list in order and use methods like those for strings.
- use non-commutative operations

  $({1^2})^3$ and 3^{2^1} 2^{3^1} 2^{1^3} ...

  bad example... 1 9, 8, 2

check
- non-uniform
- order matters.

SETS

- sort and then use sequence hashes
- use commutative operations

  hash ($s_1 * s_2 * s_3 \cdots$)

  hash($s_1 + s_2 + s_3 \cdots$)

Food for thought: why is using an object's address as its hash value a good idea? (1) it address is immutable (2) completely independant of content so the 2nd portion of uniform hashing is satisfied.