Reminders

- Still holding office hours Monday
- PSET part A due TUESDAY!
- PSER released Tuesday.

Want: Simple uniform hashing - each key is equally likely to hash to any of the m slots, independently of where other keys hash to.

Constructing Hash Functions

**Division Method**

\[ h(k) = k \mod m \]

**Collision:** \( k_1 \equiv k_2 \pmod{m} \) when m divides \(|k_1 - k_2|\)

- OK when keys uniformly random over the integers.
- BAD when keys have a regularity like \((x, 2x, 3x, \ldots)\) and \(x, m\) have a common divisor \(d\).
  - Then use \(d\) of the table.
  - Likely if \(m\) has a small divisor (like 2)
- 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**

\[ h(k) = \left\lfloor (a \cdot k) \% 2^w \right\rfloor \gg (w-r) \]

where \(m = 2^r\) and \(w\)-bit machine words.

- GOOD: not too close to extremities of range

**Book**

- GOOD: allows \(m\) to be whatever we want

Agenda

- division
- multiplication
- python hash
- hash design
**hash(key) method in Python**

- returns 32-bit int (may be negative)
- \((-1) \iff \text{Not hashable.}\)
- \(a = \text{hash}(a) \) if \(a\) is an integer
- \(\text{hash}(x) = \text{id}(x) \) if \(x\) is an object
  \(\leftarrow\) address of \(x\)

... and many more.

cannot use mutable objects

Why?

![Diagram](dia.png)

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

---

**Hashing non-integers**

previously assumed \(k \in \mathbb{N} = \{0, 1, 2, 3, \ldots, 3\}\)

**STRINGS**: brainstorming/interactive

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

Check for
- obvious non-uniformities
- inefficiencies
- false assumptions
common idea in many.

\[
\begin{array}{c}
h = 0 \\
\text{for } i \text{ in string:} \\
\text{modify } h \text{ using new input } i
\end{array}
\]

CRC PJW BUZ

link: http://www.cs.hmc.edu/~geoff/classes/hmc.cs 090. 2001 01/homework10/applet<Hashfuncs.html

---

**SEQUENCES**

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

- \((p_1^{S_1})(p_2^{S_2})(p_3^{S_3}) \ldots\) \( p_i \) = \( i \)th prime \( S_i \) = \( i \)th element of sequence.

- \% m?

- hash( 111, 2112, 3113 ) or hash( 113, 2112, 3111 )

  set hash: aka these the order doesn't matter, blc we encoded it in the new keys.

- concat list in order and use methods like those for strings.

- use non-commutative operations

\[
\begin{array}{c}
\binom{3}{2}^3 \\
\text{bad example...} \\
1
\end{array}
\]

\[
\begin{array}{c}
9 \\
8 \\
2
\end{array}
\]

check
- non-uniform
- order matters

---

**SETS**

- sort and then use sequence hashes

- use commutative operations

\[
\begin{array}{c}
\text{hash } ( S_1 \times S_2 \times S_3 \ldots ) \\
\text{hash } ( S_1 + S_2 + S_3 \ldots )
\end{array}
\]

---

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