CS 180 - Hash Tables (part 2)

Announcements

- Checkpoint is tomorrow
- Review session: Dec 10
  Not: 8am, 10am, 2-6
  10am - noon
Hashing

An array is not very space efficient. We would like to take the key and make it smaller.

A hash function \( h \) maps each key in our dictionary to an integer in the range \([0, N-1]\).

\( N \) should be much smaller than the number of keys.

Then we store \((k,e)\) in \( A[h(k)]\)
Good hash functions:
- Are fast (goal: $O(1)$ time)

- Don't have collisions.
  \[ \text{Collisions are unavoidable.} \]
First: map key to a number
Say we want keys to fit in an int.
What can we do for int, char, and short types?
Now what about long or float?

\[ a \gg 32 \]
\[ a + b \leftarrow \text{simplest way to hash} \]
This can backfire. Remember ASCII?

128-bit (full newest version)

9, 9, 9, 9, 9, 9, 9, 9,

\[ temp0! \] and \[ temp10 \] or pm0-te 1

\[ a_0 + a_1 + a_2 + a_3 + a_4 + a_5 \]

all will go to same #

Goal:

Better way to avoid collisions between "similar" keys.
A better idea: Polynomial Hash codes

Pick $a 
eq 1$ and split data into $k$ 32-bit parts
$(x_0, x_1, ..., x_{k-1}) = x$

Let $h(x) = x_0 a^{k-1} + x_1 a^{k-2} + ... + x_{k-2} a + x_{k-1}$

```
// temp 0
a = 37

// "f" * 37^5 + "e" * 37^4 + "m" * 37^3 + "p" * 37^2 +
// "0" * 37 + "1" * 37^0

// temp 10

// ["1" * 37 + "0" * 37^0]
```
Aside: Efficiency

Hornor's rule:
\[ x_{n-1} + a(x_{n-2} + a(x_{n-3} + \ldots + \ldots (x_3 + a(x_2 + a(x_1 + a(x_0))))) \]

\[ \sum_{i=0}^{n} c_i = O(n^2) \]
This strategy makes it less likely that “similar” words/data will collide.

What about overflow? (Remember, we want only 32 bits in key.)

Chop it at 32-bit

(not so great)
Cyclic Shift Hash codes
shift bits in representation somehow
\[10100010 \ldots 010001\]
Compression Map:

2. Once we have an integer key representation:
   Need to make sure it is between 0 and N-1, so is in our array.

Idea? map everything to 0 lots of collisions
want to spread things out evenly
- modular arithmetic

\[ h(k) \mod N \]
Compressing number down to something between 0 and \( N-1 \).

Ideas:

- Modular arithmetic

\[
\text{int } h(x) \mod N \\
\]

\[
x \mod n
\]
Example

A: 0 1 2 3 4 5 6 7 8 9 10

- Map is \( h(k) = k \mod 11 \)

- \( \text{insert}(12, E) \) \( h(12) = 1 \) \( E \) goes in \( A[1] \)
- \( \text{insert}(21, R) \) \( h(21) = 10 \)
- \( \text{insert}(37, I) \) \( h(37) = 4 \) \( I \) goes in \( A[4] \)
- \( \text{insert}(26, N) \) \( h(26) = 4 \) The hash bucket is full
- \( \text{insert}(16, C) \) \( h(16) = 5 \)
- \( \text{insert}(5, H) \) \( h(5) = 5 \) The hash bucket is full

Strategy 1: go to next open spot
Strategy 2: secondary hash
Strategy 3: Chaining
Some comments:
This works better if size of table is a prime number.

Why?
Go take number theory
(check book)
Before: $x \mod N$

The MAD (multiply add + divide) method is a bit better:

$$h(x) = |ax + b| \mod N$$

where $a$ and $b$ are

- not equal
- relatively prime
  $$\gcd(a, b) = 1$$
- even better if $a$ and $b$ are prime numbers
- less than $N$

Collisions may occur
Set more frequent

3 \rightarrow 2, 4, 5, 10
Goal: Simple Uniform Hashing Assumption:

For all \( k \in \text{keyspace} \),

\[
\Pr [ h(k) = i ] = \frac{1}{N}
\]

(Essentially, elements are “thrown randomly” into the buckets)
Collisions

Can we ever totally avoid collisions?

No – goal was to minimize them.

How to deal with them?

(had 3 strategies)
Strategy #3

How can we handle collisions?
(Do we have data structures to store more than one thing??)

- list — bad search
- vectors — insert can be bad
- trees — \(O(\log n)\)
  (in between lists & vectors)