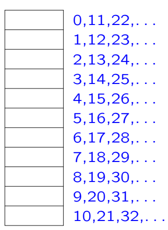
12—Hashing III

Hash Functions

CSE326 Spring 2002

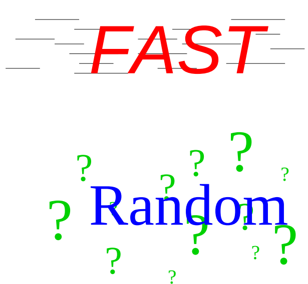
May 1, 2002

Hash Functions



- $h(k) = k$ mod TableSize works well for numerical keys
- Best if size of table is *prime*
- What if keys aren't numeric? Or really big?

Desiderata



Hashing Strings

```
int    x =  3765;
char *s = "3756";
```

A Number as a String

Conversions

Convert *A*lpha *to* *I*nteger

```
int atoi(char *s) {

  int i = 0;
  for (int idx = strlen(s)-1; idx >= 0; idx--) {







  }
  return i;
}
```


Why This Works

$$3765 = 5 \cdot 1 + 6 \cdot 10 + 7 \cdot 100 + 3 \cdot 1000$$

5	6	7	3
10^0	10^1	10^2	10^3

Strings

I	L	O	V	E	3	2	6

Business as Usual

$k =$ "Some big string" $= \sum_{i=0}^{\ell} k[i] \cdot 256^i$

$$h(k) = (\sum_{i=0}^{\ell} k[i] \cdot 256^i) \bmod m$$

Good or Bad

- $m = 256$?
- $m = 65536$?
- $m = 5683$?

Efficiency

- Horner's rule

$$\begin{aligned} p(x) &= a_3x^3 + a_2x^2 + a_1x + a_0 \\ &= ((a_3x + a_2)x + a_1)x + a_0 \end{aligned}$$

- Distributing the Mod

$$((a_3x + a_2)x + a_1)x + a_0 \bmod m = \begin{aligned} &((((a_3x + a_2) \bmod m) \cdot x \\ &\quad + a_1) \cdot x \bmod m \\ &\quad + a_0) \bmod m \end{aligned}$$

Some Code

```
int hash(char *s, int T)
{
  int l = strlen(s);
  int x = 256 % T;
  int h = s[l-1];
  for (int i = l-2; i >= 0; i--) {
    h = h*x + s[i];
    h %= T;
  }
  return h;
}
```


Slightly Better Code

```
int hash(char *s, int T)
{
  int x = 256 % T;
  int h = s[0];
  for (int i = 1; *s; i++) {
    h = h*x + s[i];
    h %= T;
  }
  return h;
}
```


Probing Efficiency

- *Linear* probing is easy

  Adds are *cheap*

- *Quadratic* probing seems to need multiply

  Mutiplies are *expensive*

Cheap Quadratic Probing

$$
\begin{aligned}
1 &= 1 \\
4 &= 1+3 \\
9 &= 1+3+5 \\
16 &= 1+3+5+7 \\
&\vdots \\
i^2 &= \sum_{j=1}^{i} 2i-1 \\
&= (i-1)^2 + 2i - 1
\end{aligned}
$$

Multiplying by the Base

- $4 \cdot 10 = 40$
- $563 \cdot 10 = 5630$
- $x \cdot 10 = x$ shifted left a digit
- $x \cdot 2 = x$ shifted left a digit, base 2
- In C++: $x \cdot 2 = x << 1$

Even Faster

```
probe_loc = h = hash(key, table_size );
probe_inc = probe_count = 0;
probe_max = table_size /2;
while ( table [ probe_loc ]. isEmpty()
        && probe_count < probe_max) {
  probe_inc = prob_inc + (probe_count << 1) - 1;
  probe_loc = h + probe_inc;
  while ( probe_loc >= table_size )
    probe_loc -= table_size ;
  probe_count++;
}
if (probe_count >= probe_max) fail ...
```

Hash Function Summary

- Quadratic Probing Effective in Practice
  - Faster than double hashing to probe
  - Need to handle table filling up prematurely
  - Limitation of quadratic probing not too bad practically
- Tables of Prime Size are Annoying
  - Pick a non-prime and hope for best
  - Powers of 2 bad for strings (just uses last few characters)
  - *odd* numbers not a bad start
  - The Internet is your Friend

Something Completely Different

```
int hash(char *s, int T)
{
  int h = *s;
  s++;
  for ( ; * s; s++)
   h = ( (h<<5)+(h>>27) ) ^ *s;

  h %= T;
}
```