Lab 15: Number Theory

CSCI104

Why Number Theory

- Relevant for a lot of things but especially for hash tables
- If we want to maintain hash table efficiency and minimize collisions

$\Rightarrow$ We want to be able to prove certain properties of different hashing techniques

Definitions

$m|n$ implies an integer k such that $n = km$

$a \equiv b \pmod{m}$ means a is congruent to b mod m, which suggests $m|a - b$

$a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$ means:

$$ac \equiv bd \pmod{m}$$
$$a + c \equiv b + d \pmod{m}$$

$gcd(a, b)$ is the greatest common divisor of a,b an integer d such that $d|a$ and $d|b$

If gcd(a,b) = 1 then a and b are “co-prime”

Open Addressing (Probing)

If $h(k)$ is occupied with another key, then probe

Let i be number of failed probes, table size m

Linear Probing

$h(k,i) = (h(k)+i) \bmod m$

Quadratic Probing

$h(k,i) = (h(k)+i^2) \bmod m$

- If $h(k)$ occupied, then check $h(k)+1^2$, $h(k)+2^2$, $h(k)+3^2$, …

Double Hashing

Pick a second hash function $h_2(k)$ in addition to the primary hash function, $h_1(k)$

- $h(k,i) = [\, h_1(k) + i*h_2(k) \,] \bmod m$

Quadratic Probing

Property: If your hash table has a prime size m, the first $m/2$ probes are guaranteed to go to distinct locations.

Proof by contradiction: Suppose the i-th and j-th probe were to the same locations (where $j < i <= m/2$), then

$$\Rightarrow \quad (h(k)+i^2) \% m = (h(k)+j^2) \% m$$

$$(h(k)+i^2) - (h(k)+j^2) \% m = 0 \% m$$

$$(i^2 - j^2) = mq \text{ (for some q)}$$

$$\Rightarrow m \mid (i^2 - j^2)$$

$$m \mid (i+j)*(i-j)$$

Since m is prime it must divide one of $(i+j)$ or $(i-j)$

But $i,j \leq m/2$, and $i \neq j$, so $0 < i-j$, and $i+j < m$

Since m is prime, you can't divide m over $(i+j)$ and $(i-j)$

Double Hashing

With a table of prime size p, and a secondary hash function that returns a number k in the range $[1, p-1]$

Probe positions would be: h, h+k, h+2k, h+3k, …

Property: the first p locations will all be unique

Proof: (exercise) Try a proof by contradiction like the one for Quadratic probing

Fermat's Little Theorem

For prime number p and integer a that is not a multiple of p

$$a^{p-1} \equiv 1 \pmod{p}$$

Then for any number n, if we can find some number a such that

$$a^{n-1} \not\equiv 1 \pmod{n}$$

⇒ n is NOT a prime number

Fermat's test says tries a bunch of a's to see if a number n is prime

Fermat's Test and Modular Exponentiation

Because the numbers we want to test are possibly very big and would cause integer overflow, we take advantage of exponents

$$b^{23} = b^{16} \cdot b^{4} \cdot b^{2} \cdot b^{1} = b^{2^4} \cdot b^{2^2} \cdot b^{2^1} \cdot b^{\cdot 2^0}$$

So we will implement a function to do:

$$117^{27} mod\ 5 = 2^{27} mod\ 5 = (2^{16} \cdot 2^{8} \cdot 2^{2} \cdot 2^{1})\ mod\ 5$$

I	N[i]	x (after iter i)	r (after iter i)	
initially		1	117%5=2	$\equiv 2^1\ mod\ 5$
0	1 (a0 = LSB)	1*2=2 mod 5	2*2=4 mod 5	$\equiv 2^2\ mod\ 5$
1	1	2*4=3 mod 5	2*2=1 mod 5	$\equiv 2^4\ mod\ 5$
2	0	3 (no change)	1*1=1 mod 5	$\equiv 2^8\ mod\ 5$
3	1	3*1=3 mod 5	1*1=1 mod 5	$\equiv 2^{16}\ mod\ 5$
4	1 (a1 = MSB)	3*1=3 mod 5		

Modular Exponentiation

```
// suppose N is given in binary as a vector of bools
// in reverse order
int modularExponentiation(vector<bool> N, int b, int m) {
   int x = 1, r = b % m;
   for (int i = 0; i < N.size(); i++) {
     if ( N[i] == true )  x = (x * r) % m;
     r = (r * r) % m;
   }
   return x;
}
```

Checkoff

Two proofs:

1) If p is a prime number, b & c integers, and $p \mid bc \Rightarrow p \mid b$ or $p \mid c$
2) Double hashing with a table size P will guarantee P distinct values

One coding exercise:

1) Modular exponentiation function & Fermat's Primality Test