Number Theory: Applications

Slides by Christopher M. Bourke
Instructor: Berthe Y. Choueiry

Fall 2007

Computer Science & Engineering 235
Introduction to Discrete Mathematics
Sections 3.4–3.7 of Rosen
cse235@cse.unl.edu

Notes

Number Theory: Applications

Results from Number Theory have *countless* applications in mathematics as well as in practical applications including security, memory management, authentication, coding theory, etc. We will only examine (in breadth) a few here.

- Hash Functions (Sect. 3.4, p. 205, Example 7)
- Pseudorandom Numbers (Sect. 3.4, p. 208, Example 8)
- Fast Arithmetic Operations (Sect. 3.6, p. 223)
- Linear congruences, C.R.T., Cryptography (Sect. 3.6 & 3.7)

Notes

Hash Functions I

Some notation: $\mathbb{Z}_m = \{0, 1, 2, \ldots, m-2, m-1\}$

Define a *hash function* $h : \mathbb{Z} \to \mathbb{Z}_m$ as

$$h(k) = k \bmod m$$

That is, h maps all integers into a subset of size m by computing the remainder of k/m.

Notes

Hash Functions II

In general, a hash function should have the following properties

- It must be easily computable.
- It should distribute items as evenly as possible among all values addresses. To this end, m is usually chosen to be a prime number. It is also common practice to define a hash function that is dependent on each bit of a key
- It must be an onto function (surjective).

Hashing is so useful that many languages have support for hashing (perl, Lisp, Python).

Notes

Hash Functions III

However, the function is clearly not one-to-one. When two elements, $x_1 \neq x_2$ hash to the same value, we call it a *collision*.

There are many methods to resolve collisions, here are just a few.

- Open Hashing (aka separate chaining) – each hash address is the head of a linked list. When collisions occur, the new key is appended to the end of the list.
- Closed Hashing (aka open addressing) – when collisions occur, we attempt to hash the item into an adjacent hash address. This is known as *linear probing*.

Notes

Pseudorandom Numbers

Many applications, such as randomized algorithms, require that we have access to a random source of information (random numbers).

However, there is not *truly random* source in existence, only *weak random sources*: sources that appear random, but for which we do not know the probability distribution of events.

Pseudorandom numbers are numbers that are generated from weak random sources such that their distribution is "random enough".

Notes

Pseudorandom Numbers I

Linear Congruence Method

One method for generating pseudorandom numbers is the *linear congruential method*.

Choose four integers:

- m, the modulus,
- a, the multiplier,
- c the increment and
- x_0 the seed.

Such that the following hold:

- $2 \leq a < m$
- $0 \leq c < m$
- $0 \leq x_0 < m$

Notes

Pseudorandom Numbers II

Linear Congruence Method

Our goal will be to generate a sequence of pseudorandom numbers,

$$\{x_n\}_{n=1}^{\infty}$$

with $0 \leq x_n \leq m$ by using the congruence

$$x_{n+1} = (ax_n + c) \bmod m$$

For certain choices of m, a, c, x_0, the sequence $\{x_n\}$ becomes *periodic*. That is, after a certain point, the sequence begins to repeat. Low periods lead to poor generators.

Furthermore, some choices are better than others; a generator that creates a sequence $0, 5, 0, 5, 0, 5, \ldots$ is obvious bad—its not uniformly distributed.

For these reasons, very large numbers are used in practice.

Notes

Linear Congruence Method

Example

Example

Let $m = 17, a = 5, c = 2, x_0 = 3$. Then the sequence is as follows.

- $x_{n+1} = (ax_n + c) \bmod m$
- $x_1 = (5 \cdot x_0 + 2) \bmod 17 = 0$
- $x_2 = (5 \cdot x_1 + 2) \bmod 17 = 2$
- $x_3 = (5 \cdot x_2 + 2) \bmod 17 = 12$
- $x_4 = (5 \cdot x_3 + 2) \bmod 17 = 11$
- $x_5 = (5 \cdot x_4 + 2) \bmod 17 = 6$
- $x_6 = (5 \cdot x_5 + 2) \bmod 17 = 15$
- $x_7 = (5 \cdot x_6 + 2) \bmod 17 = 9$
- $x_8 = (5 \cdot x_7 + 2) \bmod 17 = 13$ etc.

Notes