CS 151
Randomization
Announcements

- Lab 10 due on Friday
- Final is next Thursday (Dec. 20) 2-4pm in King 123 (our classroom). Details on Friday.
- I cannot let people take the exam early unless they have 3 finals within 24 hours. (These are rules I cannot mess with, sorry!)
- Please check your grades on blackboard to make sure they are correct. Contact me ASAP if they are not.
Random numbers come in handy in many applications:

- program testing – to generate random inputs on which to test programs
- random ordering – to get an arbitrary order of items
- average-case running time analysis – can try RT on random input
- simulations – simulate a series of requests on random input
- randomized algorithms –
  - use random numbers to determine next step of algorithm
  - use random numbers to help search a space of solutions
Random Number Generators

It’s impossible for an algorithm to generate a truly random random numbers, because they, by nature, depend on the algorithm that generates them.

A pseudorandom number is a number that appears to be random by passing a bunch of “tests” that random numbers typically pass.

Ex. Suppose we want to simulate a coin flip, and suppose we use the number of seconds on the “system clock” mod 2. Seems random enough, right?

• it’s okay for one flip, but what if we want 100 flips?
• computers can easily request 100 flips in one sec so we’d have 100 heads
• (this isn’t very random)
• even if we used smaller units of time (microseconds), the time between calls to the clock would be evenly spaced and we’d repeat our sequences
• (still not very random)
Our goal is actually to produce a sequence of pseudorandom numbers. We want this sequence to be “random”, possibly in different ways.

In a uniform distribution, all numbers in the specified range are equally likely to occur. (ex. die roll, coin flip, hash functions, hopefully.)

In a normal/Gaussian distribution (bell-curve), numbers around the mean are much more likely to occur. (ex. course grades, sum of two die rolls.)

A Poisson distribution is used to model the number of occurrences of an event in a fixed period. (ex. number of lottery ticket wins.)

Different distributions will have different “tests” that we want them to pass.
Suppose I want to generate a sequence of pseudorandom numbers according to the uniform distribution, i.e. all numbers in the given range are equally likely.

What kind of tests would we want our number generator to pass?

• every number in the sequence is equally likely to be 0,1,...,R
• the expected average of all generated numbers is R/2

Okay, I got one:

• as the first number, use the system clock in milliseconds, mod R+1
  • (our first number is kinda sorta equally likely to be 0,1,...,R)
• now generate subsequent numbers by adding 1 to previous, mod R+1
  • (hm. since we started at an arbitrary point, any one number is equally likely to be 0,1,...,R. And the average will be R/2 b/c we’re cycling.)

Fine, add some more tests:

• the sum of two consecutive random numbers is equally likely to be even/odd
• out of R consecutive random numbers, some portion will be duplicates

Our first attempt fails these two tests. But any attempt will fail some tests!
Generating a Uniform Distribution

One uniform generator is called the **linear congruential generator**.

**idea:**

- generate $x[0] > 0$ “randomly”. We call $x[0]$ the “seed”
- for $i=1, 2, \ldots$ (until you don’t want no more darned random numbers)
  - set $x[i+1] = A \times x[i] \pmod{M}$ for very specific $A,M$
    (i.e. $x[i+1]$ is a linear multiple of the previous value, congruent to $M$)

**notes:**

- if $x[0]=0$, we’d just get a series of 0, 0, 0, 0... quite boring (and not random)
- if $M$ is prime, $x[i]$ is never 0
- repeating a number results in a repeating sequence (not great...)
- the period of a seq is the length of seq between repeats (in exs. it’s 10, 5)
  - (period can’t be $>M-1$ since there are at most $M-1$ distinct nums$>0$)
- if $M$ prime, can get full period of $M-1$ with good choice of $A$ (some bad.)
- $M=2^{31}-1=2,147,483,647$ and $A=48,271$ are often used

Ex. $M=11$, $A=7$, $x[0] = 1$.  
1, 7, 5, 2, 3, 10, 4, 6, 9, 8, 1, 7, 5, 2,...

Ex. $M=11$, $A=5$, $x[0] = 1$.  
1, 5, 3, 4, 9, 1, 5, 3,...
Generating a Uniform Distribution

Implementation notes:

• the choice of \( A, M \) are very important, and very sensitive to change
  • so, you should use \( A, M \) that other people have vetted
• If \( M=2^{31}-1, A=48,271 \) then you get a full-period generator. But then with 32-bit integers, \( x[i+1] = (A\times x[i]) \mod M \) is certain to “overflow”
  • so you need some algebraic tricks, no biggie.
• not all random generators are actually good.
  • many libraries use \( x[i+1] = (A\times x[i] + C) \mod 2^B \) where \( C \) is odd, and \( B=\#\text{bits to store an int on the given machine} \)
  • but these always generate sequences that alternate even-odd-even-odd
  • actually, lowest \( k \) bits cycle with period of \( 2^k \)
• Java and some C/C++ libraries use a 48-bit generator, but only return the highest 32 bits to avoid the cycling of the lowest 16 bits