

Lecture 15: Randomness

CS442: Great Insights in Computer Science
Michael L. Littman, Spring 2006

Average of List

- Let's say we've got a list of n small integers ($n = 10,000,000,000$, for example).
- We want to know the average value of the integers.
- How can we calculate this value?
- What running time would you expect?

Straightforward Algorithm

```
def average(l):  
    total = 0  
    for i in l:  
        total = total + i  
    return (total + 0.0)/len(l)
```

- Totals up all the elements in the list.
- Divides by the length of the list.
- Running time proportional to the list length ($O(n)$), which could be quite long...

Sampling

- What if we are content with 2% error?
- To estimate the mean of a population (of bounded variance), the mean of a random sample approaches the mean of the population proportionally to the square root of the sample size.
- Error depends on variance, confidence, and sample size: Not the list size!

Demo: Oldest?

- Michael G.
- Ken
- Ray
- Steven
- Stephen
- Uyoata
- Michael S.
- Gretchen
- Vishan
- Krina
- Jawad
- Zach
- Krithika
- Caleb

Why Random?

```
def averageSample(l):
```

```
    m = 100
```

```
    total = 0
```

```
    for i in range(m):
```

```
        total = total + l[randint(0,len(l))]
```

```
    return (total + 0.0)/m
```

```
def averageFirst(l):
```

```
    m = 100
```

```
    total = 0
```

```
    for i in range(m):
```

```
        total = total + l[i]
```

```
    return (total + 0.0)/m
```

- What can go wrong if sample not random?
- $l = [0, \dots, 0, 1, \dots, 1]$ (600 0s, then 400 1s)
- `averageSample(l): 0.44`; `averageFirst(l): 0.0`;
`average(l): 0.40`.

Another Random Example

- **quicksort:** Another sorting algorithm.
- Idea: Break the list of $n+1$ elements into the median and two lists of $n/2$. The two lists are those smaller than the median and those larger than the median.
- Sort the two lists separately.
- Glue them together: All n are sorted.

Quicksort Example

- Original list:
 - [56, 80, 66, 64, 37, 36, 91, 48, 17, 20, 86, 89, 41, 1, 96, 12, 74]
- Median is 56; smaller: [37, 36, 48, 17, 20, 41, 1, 12]
 - bigger: [80, 66, 64, 91, 86, 89, 96, 74]
- Sort each; smaller: [1, 12, 17, 20, 36, 37, 41, 48]
 - bigger: [64, 66, 74, 80, 86, 89, 91, 96]
- Glue:
 - [1, 12, 17, 20, 36, 37, 41, 48, 56, 64, 66, 74, 80, 86, 89, 91, 96]

But...

- If we could find the median, the whole sorting process would be pretty easy.
- Sufficient to split anywhere in the middle half at least half the time: Still $O(n \log n)$.
- Pick a random list element. 25% of the time, it will be in the 1st quarter of the sorted list, 25% of the time in the last quarter, and 50% in the middle half.



Solve Hard Problems?

- Many randomized algorithms have better running time or are more to program than their deterministic counterparts.
- Is random computation somehow more powerful? Can it solve NP-complete problems fast?
 - Guess a “yes” proof: Check if it’s right.
- Not sure, but probably doesn’t help. Probability of guessing right can be $1/2^n$.

Using Random Bits

- Since numbers are made of bits, we can generate a random number using random bits.
- If there's a way to create random bits (coin flips), how make a random number from 0 to 3 (dreidel)?
- How about 0 to 15?
- Tricky: How about 0 to 2?

Examples

```
def rand4():  
    return randbit()* 2 + randbit()
```

```
def rand16():  
    return randbit()* 8 + randbit()*4 + randbit()* 2 + randbit()
```

```
def rand3():  
    x = 3  
    while x == 3:  
        x = randbit()* 2 + randbit()  
    return x
```

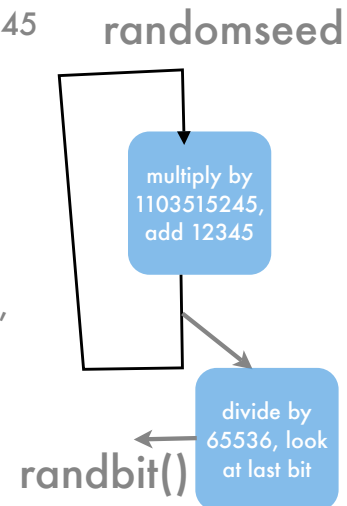
(1 1/3 calls per random number. 0, 1, 2 equally likely.)

Simple randbit

```
randomseed = 1000
def randbit():
    global randomseed
    randomseed = randomseed * 1103515245 + 12345
    return int(randomseed / 65536) % 2 == 1
```

In 50 calls, 26 True, 24 False:

```
[True, True, True, True, False, True, False, True, False,
False, False, True, False, True, False, True, True, False,
True, False, True, True, True, True, False, True, True,
False, False, True, True, False, False, False, True, False,
True, False, False, True, True, True, False, True, False,
False, True, False, False, False]
```



State of the Art

- The ... Mersenne twister algorithm, by Makoto Matsumoto and Takuji Nishimura in 1997 ... has a colossal period of $2^{19937}-1$ iterations (probably more than the number of computations which can be performed in the future existence of the universe), is proven to be equidistributed in 623 dimensions (for 32-bit values), and runs faster than all but the least statistically desirable generators.
- Python has a package for this generator.

Letter Substitution

- Caesar rotate (rot13).

abcdefghijklm
↑↑↑↑↑↑↑↑↑↑
↓↑↓↑↓↑↓↑↓↑
nopqrstuvwxyz

```
def encode(c):  
    if c < 0 or c >= 26: return c  
    if c + 13 >= 26: return c-13  
    return c+13
```

```
def rot13(s):  
    return "".join([chr(encode(ord(i)-ord('a'))+ord('a')) for i in s])
```

```
rot13('michael littman') zvpunry yvggzna  
rot13('ravine')          enivar  
rot13('pbzchgrf oht zr') ???
```

Too Crackable

- rot13 is hard to read, but easy to decode.
- In fact, any letter-for-letter substitution code can be cracked given a long enough piece of text.
- Doesn't even need to be that long...

Cryptogram

- “I ARRIVED AT THE AIRPORT ONE HOUR EARLY SO THAT, IN ACCORDANCE WITH AIRLINE PROCEDURES, I COULD STAND AROUND.” - *DAVE *BARRY
- B > I, PVS > THE, GP > AT, JBPV > WITH, BH > IN, DPGHA > STAND, KHS > ONE, GBOZKOP > AIRPORT, VKEO > HOUR, GOOBWSA > ARRIVED, SGOTU > EARLY, FKETA > COULD, QGOOU > BARRY

XOR and Random Bits

- $0 \text{ xor } 0 = 0$, $1 \text{ xor } 0 = 1$, $0 \text{ xor } 1 = 1$, $1 \text{ xor } 1 = 0$
- (2nd bit says whether or not to flip 1st bit.)
- Let's say you and I have got two copies of a long sequence of random bits (called a “pad”): pad = 1100111110.
- I want to send you ten bits so no one can figure out what I sent but you: msg = 0110111010.

Message Received

We can use our secret bits to encrypt (encode) and decrypt (decode) the message we want to transmit:

$$\begin{aligned} \textit{send} &= \textit{msg} \text{ xor } \textit{pad} = 1100111110 \text{ xor } 0110111010 \\ &= 1010000100. \end{aligned}$$

$$\begin{aligned} \textit{decode} &= \textit{send} \text{ xor } \textit{pad} = 1010000100 \text{ xor } 0110111010 \\ &= 1100111110 = \textit{msg}! \end{aligned}$$

If we agree on a randomseed, can make a pseudorandom pad.

Public Key Idea

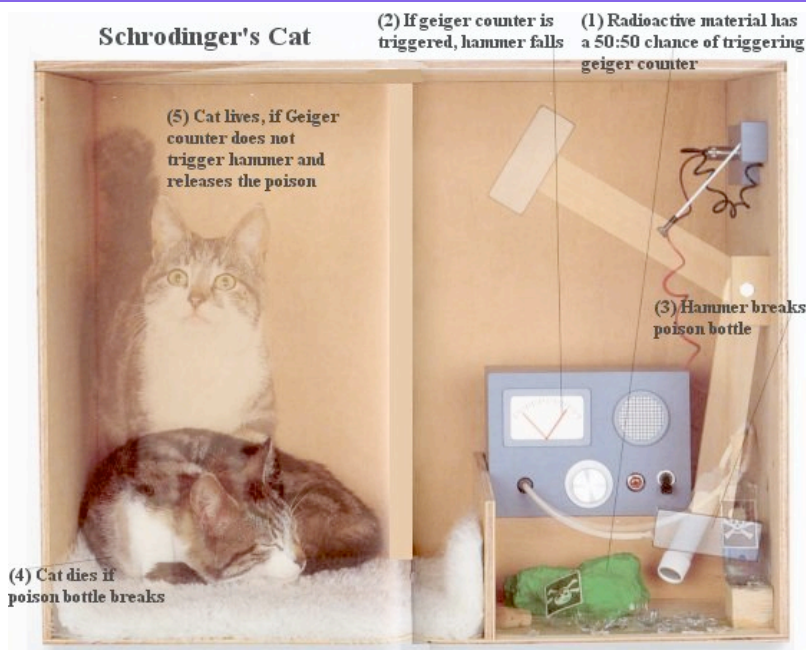
- I've got a number x that is the product of two big prime numbers.
- I tell everyone the product, which is what is needed to encrypt messages for me.
- Only *I* know the factors, which are what are needed to decrypt messages for me.
- Because factoring is hard, people can send me secret messages, even though I've publicized x .

Quantum Effects

- Pseudo-random-based encryption always has a chance of being cracked.
- Only source of true randomness: quantum mechanics (the rest of physics is deterministic, if chaotic).
- Einstein didn't like it. Tough.

Quantum Randomness

Schrodinger's Cat



Quantum Computer

- A quantum bit (qubit) is simultaneously zero and one (a superposition). n qubits can represent 2^n possibilities.
- When you look, one possibility presents itself, according to well understood probabilistic rules. A kind of parallel search.
- Shor: A computer with qubits can factor numbers in polynomial time!

If Factoring is Easy...

- quantum computers invalidate standard cryptosystems. No more secrets.
- However, they also open up some wild possibilities.
- quantum cryptography: qubits can be completely random and correlated at a distance. The perfect pad!

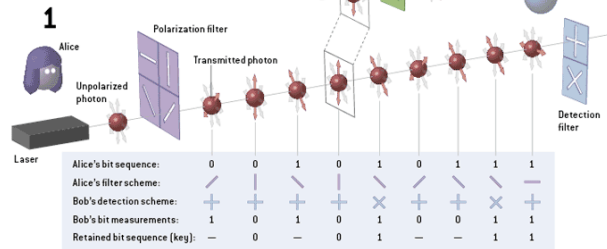
QUANTUM MECHANICS HIDES A SECRET CODE KEY

Alice and Bob try to keep a quantum-cryptographic key secret by transmitting it in the form of polarized photons, a scheme invented by Charles Bennett of IBM and Gilles Brassard of the University of Montreal during the 1980s and now implemented in a number of commercial products.

1 To begin creating a key, Alice sends a photon through either the 0 or 1 slot of the rectilinear or diagonal polarization filters, while making a record of the various orientations.

2 For each incoming bit, Bob chooses randomly which filter slot he uses for detection and writes down both the polarization and the bit value.

3 If Eve the eavesdropper tries to spy on the train of photons, quantum mechanics prohibits her from using both filters to detect the orientation of a photon. If she chooses the wrong filter, she may create errors by modifying their polarization.



4 After all the photons have reached Bob, he tells Alice over a public channel, perhaps by telephone or an e-mail, the sequence of filters he used for the incoming photons, but not the bit value of the photons.

5 Alice tells Bob during the same conversation which filters he chose correctly. Those instances constitute the bits that Alice and Bob will use to form the key that they will use to encrypt messages.

TOMMY MOORMAN; ADAPTED FROM THE CODE BOOK: THE SCIENCE OF SECRECY FROM ANCIENT EGYPT TO QUANTUM CRYPTOGRAPHY, BY SIMON SINGH (1999)

Next Time

- Image processing.
- We've completed Hillis 1-5. We'll start on Chapter 6 after the break/midterm.