Extended Introduction to Computer Science CS1001.py

Lecture 9: Randomness in Computing

Instructors: Benny Chor, Amir Rubinstein

Teaching Assistants: Michal Kleinbort, Amir Gilad

School of Computer Science

Tel-Aviv University

Spring Semester 2017

http://tau-cs1001-py.wikidot.com

Randomness in Computing

- What is randomness? According to Wikipedia:
  Randomness is the lack of pattern or predictability in events. A random sequence of events, symbols or steps has no order and does not follow an intelligible pattern or combination. Individual random events are by definition unpredictable, but in many cases the frequency of different outcomes over a large number of events (or "trials") is predictable.

- The programs we have written so far were almost all deterministic: given an input, the output is uniquely determined.

- In many cases it is useful to include randomness in computations. Algorithms that use randomness are called randomized or probabilistic or coin flipping algorithms. Unlike deterministic algorithms, their executions cannot be easily reproduced.

- Examples for using randomness in computation:
  - Simulation
  - Sampling (usually a large data set, *e.g.* testing a program)

  } The next two examples

  - Cryptography (we will see the an example soon, the Diffie-Hellman protocol)
  - Improving efficiency or avoiding worst case scenarios with high probability (we will see this in the Quicksort and the Primality Testing algorithms)

A Piece of History: Obtaining Random Sequences

From Wikipedia:

A Million Random Digits with 100,000 Normal Deviates is a random number book by the RAND Corporation, originally published in 1955.

The book, consisting primarily of a random number table, was an important 20th century work in the field of statistics and random numbers.

It was produced starting in 1947 by an electronic simulation of a roulette wheel attached to a computer, the results of which were then carefully filtered and tested before being used to generate the table.

The RAND table was an important breakthrough in delivering random numbers, because such a large and carefully prepared table had never before been available. In addition to being available in book form, one could also order the digits on a series of punched cards.

73735	45963	78134	63873
02965	58303	90708	20025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	20422	05720
15838	47174	76866	14330
89793	34378	08730	56522
78155	22466	81978	57323
16381	66207	11698	99314
75002	80827	53867	37797
99982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
30500	28220	12444	71840

A random sampling of 300 random digits from *A Million Random Digits with 100,000 Normal Deviates*

Obtaining Random Sequences (cont.)

- True Random Number Generators (TRNG)
  Extract randomness from physical phenomena such as cosmic radiation, radioactive decay, etc.

- A Philosophical question for you to ponder about:
  Are there any events in nature that are truly random?

  Recall Einstein's quote: "God does not play dice with the universe.",
  referring to his views about quantum mechanics
  (this quote is often misinterpreted:
  http://www.techinsider.io/god-does-not-play-dice-quote-meaning-2015-11)

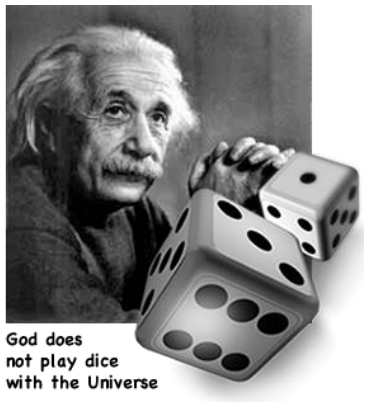


Image from:
http://www.smallplanet.us

- Pseudo-random Generators (PRG): An algorithm to generate a long sequence of numbers, deterministically, from a short, truly random random "seed". The long sequence appears random ("random enough") wrt different tests.

Example for PRG: Linear Congruential Generator

$$x_{i+1} = (a \cdot x_i + c) \bmod m$$

$a = 1, c = 7, m = 12$

$x_0 = 0$

$x_1 = 7$

$x_2 = 2$

$x_3 = 9$

…

The "seed".
Used to initialize the random number generator.
Picked at random (are we in a loop here?)
Can be picked *e.g.* by using the system's clock.

```
>>> a, c, m = 1, 7, 12
>>> r = 0
>>> for i in range(20):
        print(r)
        r = (a*r + c) % m
```

Note: this prg can easily be predicted, given sufficiently long prefix.

The output pasted as a single line:

```
0 7 2 9 4 11 6 1 8 3 10 5 0 7 2 9 4 11 6 1 8
```

- The numbers in the sequence must eventually enter a cycle. The length of the cycle is called the period of the random number generator.

- The choice of the parameters affects the period.
  Try, for example, c=8 instead of 7.

Randomness in Python

- Python employs a more sophisticated pseudo random generator, called Mersenne Twister.

- The name derives from the fact that the period of the generator is chosen to be a Mersenne prime: a prime number that is one less than a power of two. That is, it is a prime number that can be written in the form $2^n - 1$ for some positive integer n.

- Specifically, the PRNG in Python produces 53-bit precision floating point numbers, and has a period of 2^{19937}-1.

```
>>> import random

>>> random.random() #generate a uniform random number in [0, 1)
0.9724062711684623
>>> random.random()
0.9793789492766168
>>> random.random()
0.2880152915931866
```

Randomness in Python – Some Useful Functions

```
>>> random.uniform(3.2, 12.01) # random number in the range [a, b)
9.311113665186017


>>> random.randrange(0,1000) # random int from range(start, stop[, step])
929


>>> random.choice([1, 5, 6, -44, 9])
-44
>>> lst = [random.choice("a"*5 + "bcdef") for i in range(1000)]
>>> lst[:10]
['c', 'a', 'a', 'a', 'a', 'e', 'b', 'a', 'a', 'a']
>>> lst.count("a")/len(lst)
0.497
>>> lst.count("b")/len(lst)
0.101


>>> L = [1,2,3,4,5]
>>> random.shuffle(L) #in place, mutates L itself
>>> L
[2, 5, 1, 4, 3]
```

Randomness in Action – Example 1: Sampling
Estimating π by a Monte Carlo Method

- From Wikipedia:

  Monte Carlo methods (or Monte Carlo experiments) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other mathematical methods.

- The name "Monte Carlo" was coined by Nicholas Constantine Metropolis (1915-1999) and inspired by Stanslaw Ulam (1909-1986), because of the similarity of statistical simulation to games of chance, and because Monte Carlo is a center for gambling and games of chance. (http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html)

- Just to confuse you, there are also "Las Vegas algorithms": Randomized algorithms that never err; that is, they either produce the correct result (with high probability), or they halt without giving any outcome.

Randomness in Action – Example 1: Sampling
Estimating π by a Monte Carlo Method

- Randomly choose points (x,y) in the unit square (0<=x,y<=1)
- Count how many of them are located inside the quarter circle of radius 1 centered in the origin.
- The ratio is the area of the quarter circle (π/4) divided by the area of the square (1).

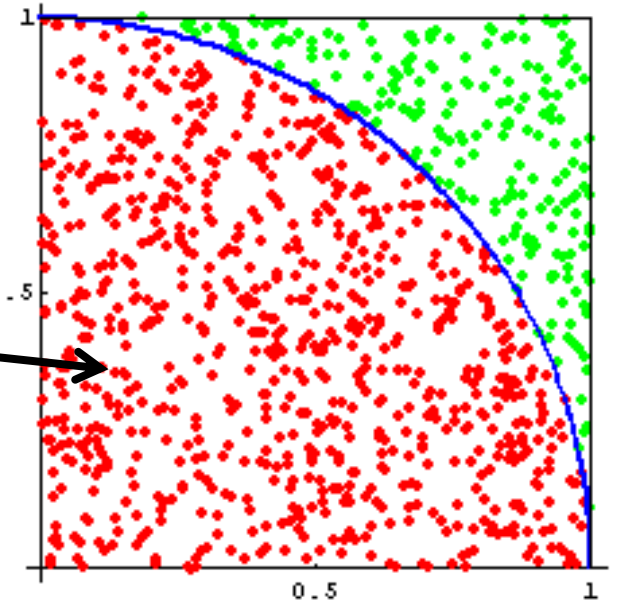


Figure taken from
http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html

Estimating π by a Monte Carlo Method in Python

```
import random

def estimate_Pi(num=1000):
    """ estimate pi by a monte carlo experiment """
    count=0
    for n in range(num):
        x = random.random()
        y = random.random()
        if x**2 + y**2 <= 1.0:
            count += 1
    return 4*count/num
```

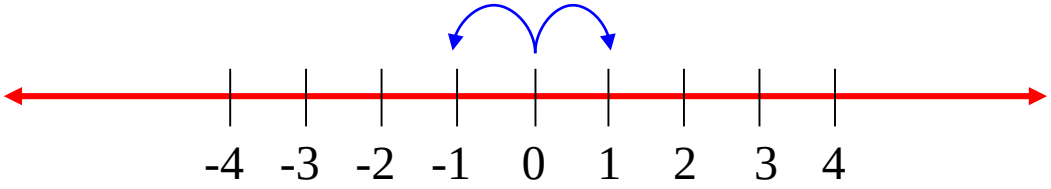
```
>>> estimatePi()
3.156
>>> estimatePi(100000)
3.12864
>>> estimatePi(10**8)
3.14175412 #took about a minute
```

```
>>> import math
>>> math.pi
3.141592653589793
```

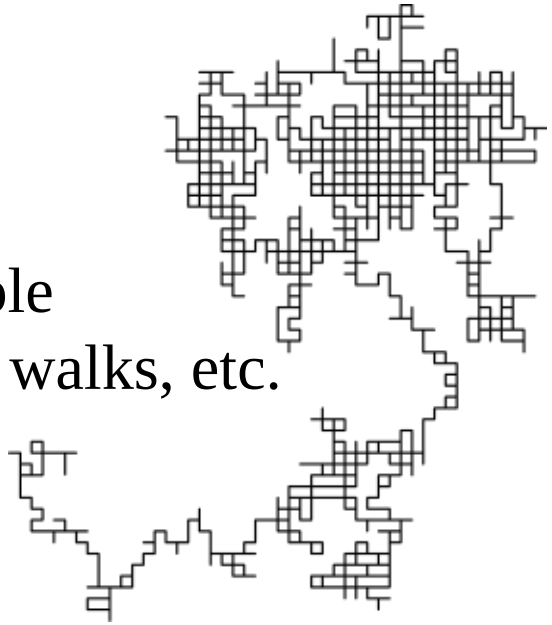
Randomness in Action – Example 2: Simulation
Random Walk

- A random walk is a mathematical formalization of a path that consists of a succession of random steps.

- For example, a simple unbiased 1-dimensional random walk: a marker is placed at zero on the number line, and at each step moves +1 or −1 with equal probability.



- There are numerous other versions, for example grids of higher dimensions, on graphs, biased walks, etc.

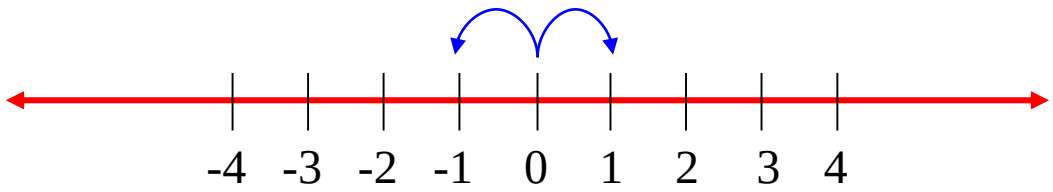


Random walk in two dimensions.
https://upload.wikimedia.org/wikipedia/commons/f/f3/Random_walk_2500_animated.svg

Randomness in Action – Example 2: Simulation
Random Walk

- Random walks are used to model various types of phenomena from a diverse range of fields, such as:
  - Economics: shares prices
  - Genetics: genetic drift (frequency of gene variants (allele) in a population)
  - Physics: Brownian motion and diffusion
  - Ecology: population dynamics
  - www : Twitter's WTF ("Who to Follow")
  - Gambling (e.g. the Gambler's ruin phenomenon)
  - …

Simple Symmetric 1D Random Walk



- Formally, a simple symmetric 1D random walk is defined as a series $\{S_1, S_2, \ldots\}$ where $S_n = \sum_{j=1}^{n} Z_j$ and $Z_j \in \{-1,1\}$.
- $|S_n|$ is the distance from 0 after *n* steps.
- It is known that: $\lim_{n\to\infty} \frac{E(|S_n|)}{\sqrt{n}} = \sqrt{\frac{2}{\pi}}$
- Let's examine this property by simulation.
  Note that $E(|S_n|)$ is the expectation of $|S_n|$, so we need to average the distance over several executions.

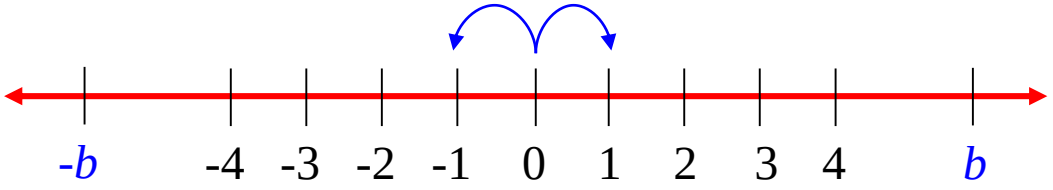
Simple Unbiased 1D Random Walk – in Python

```
def rand_walk(run_length, dimension=1, show=False):
      *
      *
      *
```

See homework 3

Simple Symmetric 1D Random Walk – Another Property

- What is the average number of steps needed to reach a distance of *b* from 0?



- What is the probability that a walk of length n, starting from the origin, will return to the origin.
- What is the average distance from origin for walks of length n?
- How do the last two questions depend on the dimension of the lattice?