Pseudo random number generators

Leonardo Toffalini

2026-02-19

Outline

1. Intuition

1.1 What we expect from a PRNG

1. Given a short input (seed) it produces a long *seemingly random* sequence.
2. Generation should be really fast.
3. The sequence must be reproducible just from the seed.

1.2 The sad truth

What does *seemingly random* mean?

We cannot use Kolmogorov complexity to measure randomness, because that would not satisfy 1.

We cannot use physical methods like radioactive radiation as they would not satisfy 2. and 3.

1.3 Reconciliation

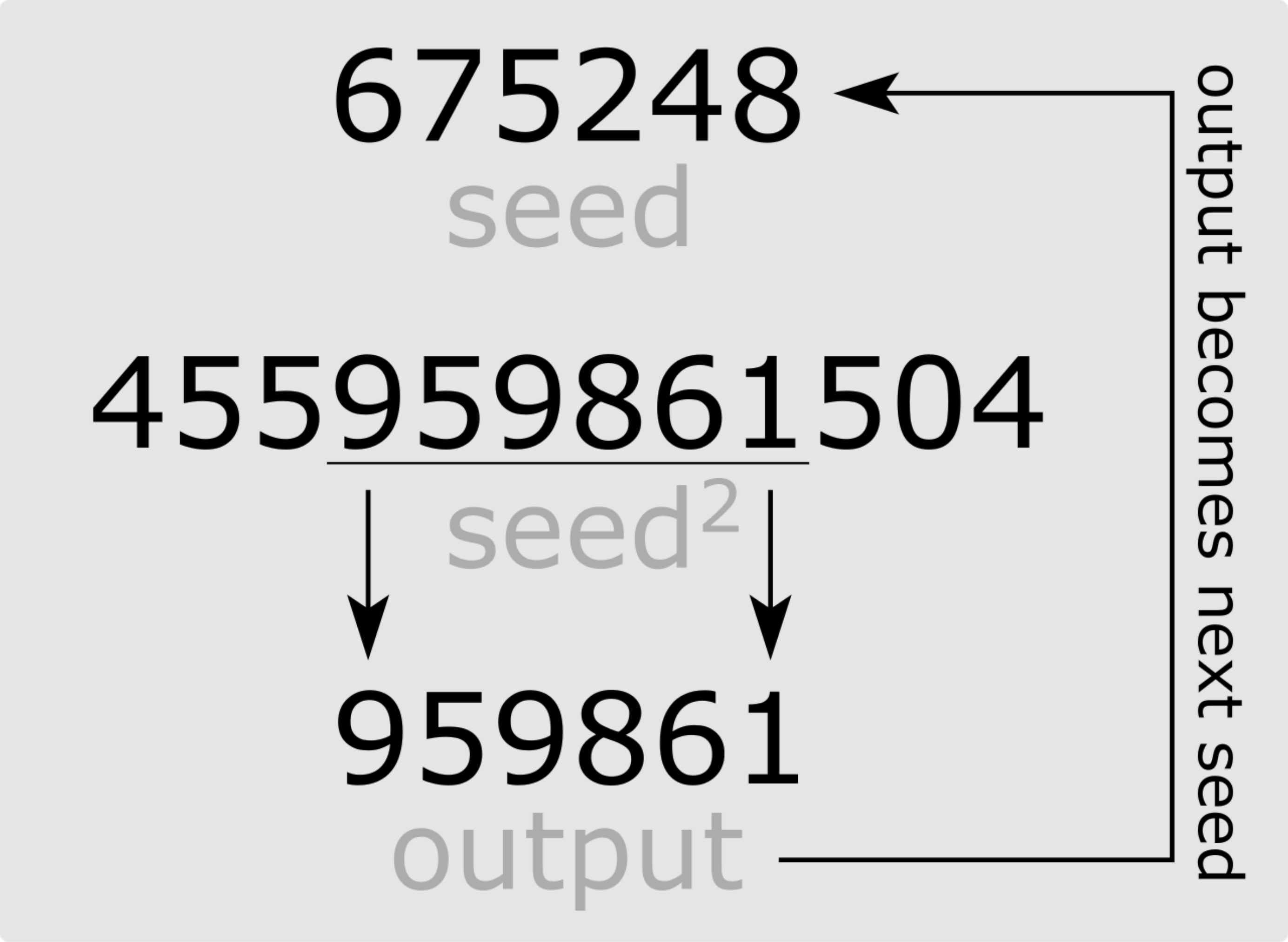
We need to redefine what *seemingly random* means.

"There must not exist a randomized polynomial time algorithm that can differentiate between truly random and generated sequences with non-negligible advantage."

"There must not exists a polynomial time algorithm that can guess the next bit given the previous bits."

2. The classics

2.1 Middle square method (1946)



2.2 Problem with middle square

- short period
- often enters a short cycle
- if middle digits are 00…0 it no longer produces meaningful numbers

> "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."
>
> — John von Neumann

2.3 Linear congruential generator (1951)

$$X_i = aX_{i-1} + b \pmod{m}$$

2.4 RANDU (1960-1970)

$$X_i = 65539 \cdot X_{i-1} \pmod{2^{31}}$$

https://github.com/leonardo-toffalini/typsting/blob/main/kript/randu_planes.gif

Outputs fall into only 15 planes.

2.5 Park–Miller (1988)

$$X_i = 7^5 \cdot X_{i-1} \pmod{2^{31} - 1}$$

"Give me something I can understand, implement and port… it needn't be state-of-the-art, just make sure it's reasonably good and efficient."

— Park–Miller

2.6 Problem with LCG

The elements of LCG sequences can be guessed in polynomial time with polynomial many known elements of the sequence with a sufficiently complicated algorithm.

2.7 Shift register (1965)

$$a_k = f(a_{k-1}, a_{k-2}, \ldots, a_{k-n})$$

2.8 Linear shift register

$$f(x_0, ..., x_{n-1}) = b_0 x_0 + b_1 x_1 + ... + b_{n-1} x_{n-1}$$

$$b_i \in \{0, 1\}$$

2.9 Xorshift

$$y_1 = x_n \oplus (x_n \ll 13)$$
$$y_2 = y_1 \oplus (y_1 \gg 17)$$
$$x_{n+1} = x_2 \oplus (x_2 \ll 5)$$

Alternatively, in $\mathbb{F}_2^{32}$

$$x_{n+1} = (1 \oplus 2^5)(1 \oplus 2^{32-17})(1 \oplus 2^{13})x_n$$

2.10 Problem with shift registers

$$\begin{aligned}
b_0 a_0 + b_1 a_1 + \dots + b_{n-1} a_{n-1} &= a_n \\
b_0 a_1 + b_1 a_2 + \dots + b_{n-1} a_n &= a_n \\
\vdots \qquad\qquad \vdots \quad & \\
b_0 a_{n-1} + b_1 a_n + \dots + b_{n-1} a_{2-2} &= a_n
\end{aligned}$$

The coefficients can be recovered by solving the linear system.

2.11 Square root generator

$$\sqrt{5} = 10.\overbrace{0011100011011}^{f(5)}\ldots$$

$$f(a) = \sqrt{a} - \left\lfloor \sqrt{a} \right\rfloor$$

2.12 Problem with square root generator

Seems random but is still *breakable* with a sufficiently complicated number theoretic approach.

2.13 Mersenne twister

asd

3. Formalism

3.1 Definitions

Definition 3.1.1 A function $f : \mathbb{Z}^+ \to \mathbb{R}$ is negligible if for all fixed k $\lim_{n\to\infty} n^k f(n) \to 0$.

$f(n) = \mathrm{NEGL}(n)$

Definition 3.1.2 A function $G : \{0,1\}^* \to \{0,1\}^*$ is a generator if $|G(x)|$ only depends on $|x|$ and $|x| < |G(X)| < |x|^c$ for some constant c.

3.1 Definitions

Definition 3.1.3 Let $\mathcal{A}$ be a randomized polynomial time algorithm that for each $z \in \{0, 1\}^*$ input it outputs a bit $\mathcal{A}(z) \in \{0, 1\}$ meaning the input was random (1) or not (0). $\mathcal{A}$ is called a test.

3.1 Definitions

Definition 3.1.4 For a fixed $n \geq 1$ chose uniformly at random x from $\{0, 1\}^n$ and y from $\{0, 1\}^N$ where $N = |G(x)|$. With equal probability give either $G(x)$ or y as input to $\mathcal{A}$. We say that $\mathcal{A}$ was a successful test if it correctly determined whether it had a random input or not.

3.1 Definitions

Definition 3.1.5 We say that a generator G is secure if for all randomized polynomial time algorithms $\mathcal{A}$ the probability of $\mathcal{A}$ being successful is $\frac{1}{2} + \mathrm{NEGL}(n)$.

Remark

In essence, G passes all *meaningful* tests, that is, the best test is to guess at random.

3.2 Equivalent definition of secure generator

Definition 3.2.1 A generator $g(x) = G_1 G_2 \ldots G_N$ is said to be unpredictable if

$$\mathbb{P}(\mathcal{B}(n, G_1 \ldots G_i) = G_{i+1}) = \frac{1}{2} + \mathrm{NEGL}(n) \quad \forall i \underset{R}{\in} \{1, \ldots, N\}$$

where $x \underset{R}{\in} \{0, 1\}^n$.

Theorem 3.2.2 (Yao) A generator G is secure if and only if it is unpredictable.

3.3 How hard could it be?

Proposition 3.3.1 If $\mathbf{P} = \mathbf{NP}$, then there is no secure generator.

Proof: Fix a generator G.

Define $L := \{y : \exists x \in \{0,1\}^* \text{ such that } y = G(x)\}$. Clearly $L \in \mathbf{NP}$, since x is a polynomial proof for $y \in L$.

By $\mathbf{P} = \mathbf{NP} \implies L \in \mathbf{P} \implies \exists \mathcal{A}$ PTA that decides $x \overset{?}{\in} L$.

From this we have that $\mathcal{A}$ is always successful in recognizing $G(x)$, that is G is not secure.

$\square$

3.3 How hard could it be?

Remark

If you find a secure generator you you can claim your \$1M, since you have proven $\mathbf{P} \neq \mathbf{NP}$.

3.4 One-way functions

Definition 3.4.1 We say that $f : \{0,1\}^* \to \{0,1\}^*$ is a one-way function, if

- $\exists c \geq 1$ such that $|x|^{\frac{1}{c}} < |f(x)| < |x|^c$
- $f(x)$ is polynomial time computable
- $\forall \mathcal{A} : \{0,1\}^* \to \{0,1\}^*$ RPA and $y \underset{R}{\in} \{0,1\}^*$ the following holds:

$$\mathbb{P}(f(\mathcal{A}(f(y))) = f(y)) = \mathrm{NEGL}(n).$$

Remark

Since f must not be invertible we could not write $\mathcal{A}(f(y)) = y$.

3.4 One-way functions

Definition 3.4.2 We say that f is a one-way permutation if it is a bijection and $|f(x)| = |x| \quad \forall x$.

Theorem 3.4.3 (Goldreich–Levin) If f is a one-way permutation then there is a secure generator created from f.

3.5 Construction of Goldreich–Levin generator

Choose a seed $(x, p) \underset{R}{\in} \{0,1\}^n \times \{0,1\}^n$

Define $y^{(t)} = f^t(x) \quad t = 1, \ldots, N$

Output $g(x, p) = \overline{G_1 G_2 \ldots G_N}$, where $G_t = p \cdot y^{(t)}$

Where

$$(a_i)_{i=1}^n \cdot (b_i)_{i=1}^n := \bigoplus_{i=1}^n a_i b_i$$

Even if someone knows p and $y^{(k)}$ they cannot predict

$$G_{k+1} = p \cdot f\left(y^{(k)}\right)$$

4. One-way function candidates

4.1 Factorization

Let p and q be two n long prime numbers.

Let $f(n, p, q) = pq$

Given $f(n, p, q)$ try to guess p and q.

> **Remark**
>
> Shor's algorithm shows that this function is reversible in polynomial time with a quantum computer.

4.2 Discrete logarithm

Given a prime p and a primitive root g and $k < p$, let $y = g^k \bmod p$ the output is (p, g, y).

Try to guess k, that is the discrete logarithm of y modulo p.

> **Remark**
>
> Shor's algorithm solves this problem too.

4.3 Discrete square root

Given m and $x < m$ the output is m and $y = x^2 \bmod m$.

Try to find x such that $x^2 \equiv y \pmod{m}$

5. Beyond random bits

5.1 Uniform distribution

Generate a random number $X \sim U(0, 1)$

Generate an n long binary sequence $(a_i)_{i=1}^{n}$

$$X = \sum_{i=1}^{n} \frac{a_i}{2^i}$$

$$X = \overline{0.a_1 a_2 a_3 \ldots a_n}_2$$

This has precision 2^{-n}.

5.2 Box–Muller transform

Suppose you have $U_1, U_2 \sim U(0,1)$.

Let

$$Z_1 = \sqrt{-2\log U_1}\cos(2\pi U_2)$$
$$Z_2 = \sqrt{-2\log U_1}\sin(2\pi U_2)$$

Then Z_1, Z_2 are independent standard normal variables.

https://upload.wikimedia.org/wikipedia/commons/1/1f/Box-Muller_transform_visualisation.svg