Pseudo random number generators

Leonardo Toffalini

2026-02-18

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 fast, 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

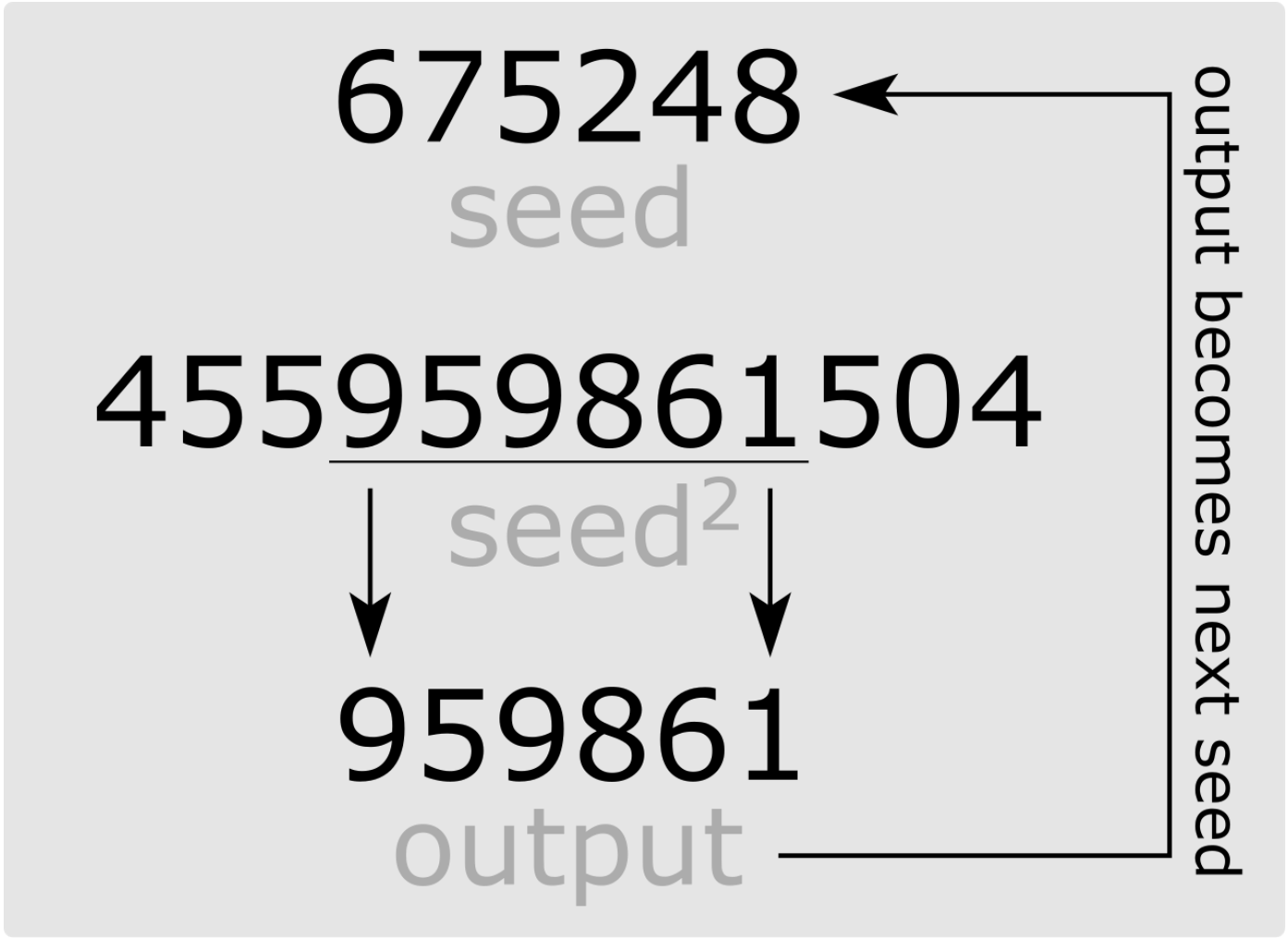
We need to redefine what *seemingly random* means.

"There must not exist a polynomial time algorithm that can differentiate between truly random and generated sequences."

"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

> “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)

$$X_i = 65539 \cdot X_{i-1} \quad \left(\mathrm{mod}\, 2^{31}\right)$$

file:///Users/tleo/Downloads/git/school/typsting/kript/randu_planes.gif

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 $\mathbb{F}_2^{32}$

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

2.10 Problem with shift registers

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

2.11 Square root generator

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

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

2.12 Problem with square root generator

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

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.

3.1 Definitions

Definition 3.1.6 unpredictable...

Theorem 3.1.7 (Yao) A generator G is secure if and only if it is unpredictable.

Proposition 3.1.8 If $\mathbf{P} = \mathbf{NP}$, then there is no secure generator.

3.1 Definitions

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 $\mathbf{P} = \mathbf{NP}$ we also have $L \in \mathbf{P}$, meaning that $\exists \mathcal{A}$ polynomial time algorithm 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.

□