ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

Scuola di Scienze

Corso di Laurea in Ingegneria e Scienze Informatiche

BENCHMARK DI GENERATORI DI NUMERI PSEUDOCASUALI

Elaborato in
Crittografia

Relatore
Prof. Luciano Margara

Presentata da
Simone Fabbri

Seconda Sessione di Laurea
Anno Accademico 2022 – 2023

KEYWORDS

Generatori di numeri pseudocasuali

Randomness

P-value

Metriche

NIST Statistical Test Suite

Science is what we understand well enough to explain to a computer. Art is everything else we do.
- Donald Knuth

If you don't know how, observe the phenomena of nature, they will give you clear answers and inspiration.
- Nikola Tesla

Sommario

Ogni applicazione crittografica si basa sulla generazione di numeri random. Pertanto, la sicurezza di queste applicazioni è strettamente dipendente dalle proprietà statistiche dei sottesi generatori di numeri casuali impiegati. Lo scopo di questa ricerca è prendere una vasta serie di generatori di numeri pseudocasuali (PRNG) considerati dalla comunità scientifica come validi fonti di randomness e valutarne le principali proprietà statistiche attraverso una serie di test utilizzando la NIST test suite. In particolare, andremo a stilare una classifica dei PRNG analizzati e saremo in grado di rispondere a domande come: *Quale è il numero più randomico tra Pigreco ed il numero di Nepero?*, *Quale è il miglior generatore di numeri pseudocasuali tra quelli analizzati?*. Andremo difatti a dimostrare come il Blum Blum Shub Generator, il Permuted Congruential Generator, il Linear Congruential Generator ed il Mersenne Twister Generator siano tra i migliori PRNG tra quelli analizzati.

Introduzione

Motivazioni e contributi

La sicurezza di molti sistemi crittografici si basa sulla generazione di numeri random. Per esempio sono casuali: la chiave segreta nel DES, i numeri primi p e q nel RSA ed il numero e $\in (0, 1)$ usato nei protocolli zero-knowledge. In tutti queste applicazioni la sicurezza del protocollo si basa sul fatto che la quantità generata sia sufficientemente grande e che sia random. In particolare, per numero random si intende una valore la cui probabilità di essere selezionato è sufficientemente bassa da precludere ad un possibile attacker di ottenere un vantaggio attraverso una strategia di ricerca basata su tale probabilità.
Data l'importanza dei numeri random in crittografia si è deciso di valutare la qualità dei generatori di numeri pseudocasuali oggi comunemente usati nelle applicazioni crittografiche. Per fare ciò si introdurranno le principali metriche di valutazione di tali generatori, se ne definiranno i principi alla base, e infine si riporteranno i risultati delle metriche su questi generatori, per realizzare così una panoramica di confronto.

Organizzazione della tesi

La tesi è organizzata come segue:

- **Capitolo 1** - Vengono introdotti i principali concetti matematici relativi alla statistica utilizzata. Inoltre, vengono presentati vari generatori di numeri pseudocasuali e le principali metriche di valutazione utilizzate;
- **Capitolo 2** - Viene chiarito il contesto e vengono riassunti lavori pre-esistenti correlati a questo progetto;
- **Capitolo 3** - Definisce la suite utilizzata per la realizzazione dei test ed i criteri di interpretazione dei risultati;
- **Capitolo 4** - Vengono riportati e discussi i risultati dei test realizzati;

- **Capitolo 5** - Presenta le conclusioni della tesi con le principali osservazioni e deduzioni che si possono realizzare sulla base dei risultati ottenuti.

Contents

Elenco delle immagini

Elenco dei programmi

Chapter 1

Preconcetti teorici

In questo capitolo si introducono i preconcetti teorici necessari per la piena comprensione dei test e dei risultati successivamente riportati. In particolare, viene fornita la necessaria conoscenza di statistica per poter comprendere le metriche, vengono introdotti gli algoritmi usati per la valutazione dei generatori di numeri pseudocasuali e viene presentata la logica di funzionamento o la diretta implementazione dei PRNG analizzati.

1.1 Statistica

Le metriche impiegate per la valutazione dei generatori di numeri pseudocasuali si basano sui concetti di distribuzione normale e distribuzione chi-quadro, che possono servire come riferimento di generatore random ideale.

1.1.1 Distribuzione normale

La distribuzione normale è un pattern che emerge quando un vasto numero di variabili random indipendenti con la stessa media e varianza sono sommate tra di loro.

Definizione. *Una variabile random continua X ha distribuzione normale di media μ e varianza σ^2 se la sua funzione di densità di probabilità può essere definita come:*

$$f(x) = \frac{1}{\sigma \cdot \sqrt{2\pi}} \cdot \exp \frac{-(x-\mu)^2}{2\sigma^2}$$

con $-\infty < x < \infty$.

In questo caso X viene definita come $N(\mu, \sigma^2)$. Se X è $N(0,1)$ allora viene detto che X ha una distribuzione normale standard. Si noti inoltre che, nel caso X sia $N(\mu, \sigma^2)$, allora la variabile random $Z = \frac{(X-\mu)}{\sigma}$ è $N(0,1)$.

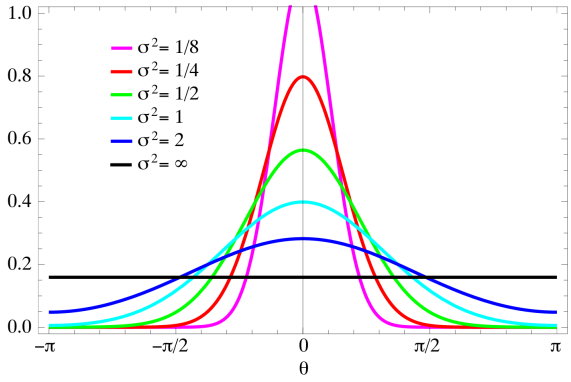

Figure 1.1: Distribuzione normale con $\mu = 0$ al variare del parametro σ

1.1.2 Distribuzione χ^2

La distribuzione χ^2 può essere usata per definire la qualità con la quale le frequenze osservate di determinati eventi seguono le frequenze definite da una distribuzione ipotetica. In pratica, la distribuzione χ^2 con v gradi di libertà emerge quando i quadrati di v variabili random indipendenti con distribuzione normale standard sono sommati tra di loro.

Definizione. *Sia $v \geq 1$ un intero. Una variabile random continua X ha distribuzione χ^2 con v gradi di libertà se la sua funzione di densità di probabilità può essere definita come:*

$$f(x) = \begin{cases} \frac{1}{\tau(\frac{v}{2}) \cdot 2^{\frac{v}{2}}} \cdot x^{\frac{v}{2}-1} \cdot e^{\frac{-x}{2}}, & \text{se } 0 \leq x < \infty \\ 0, & \text{se invece } x < 0 \end{cases}$$

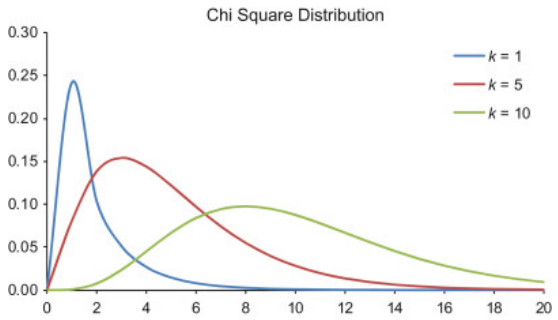

Figure 1.2: Distribuzione χ^2 al variare del grado di libertà k

Dove τ è la funzione gamma definita come: $\tau(t) = \int_0^\infty x^{t-1} \cdot e^{-x}\, dx$, per $t > 0$.
La media e la deviazione standard della distribuzione così definita sono: $\mu = v$ e $\sigma^2 = 2 \cdot v$.

1.2 Metriche di confronto

In questo capitolo andremo a definire le metriche di confronto dei generatori di numeri pseudocasuali, che verranno poi implementate come codice per poter realizzare dei veri e propri test. Prima però, introduciamo un concetto statistico alla base dell'analisi dei risultati ottenuti da tali metriche di confronto.

1.2.1 Il livello di significatività

I test esposti in questo capitolo permetteranno la definizione di statistiche X sulle sequenze binarie in input, ovvero funzioni degli elementi di tali sequenze (ad esempio: il numero di 0 presenti nella sequenza). Queste statistiche è stato dimostrato che seguono particolari andamenti di tipo $N(0,1)$ oppure χ^2 quando applicate a sequenze di input random. Pertanto, sulla base degli andamenti tipici di queste variabili statistiche X nel caso di input random, è possibile definire un valore di soglia x_a, che permette di delineare il confine tra sequenze considerate random e sequenze da considerarsi non random. In pratica, si determina un "livello di significatività" α al quale corrisponde un determinato valore di soglia x_a tale che $P(X > x_a) = \alpha$. Quando la statistica X di una determinata sequenza di input s supera il valore di soglia x_a il test fallisce e la sequenza viene considerata come non random. Mentre, quando la statistica X rimane al di sotto il valore di soglia x_a il test ha successo e la sequenza viene considerata frutto del caso. Pertanto, il parametro α permette di impostare il livello di certezza del predicato "la sequenza di input è random" e corrisponde infatti alla percentuale di sequenze scartate dal test sebbene randomiche. Per questo motivo, è opportuno impostare correttamente il livello di significatività α (generalmente pari 0.01). Infatti, prendendo un α troppo grande si ha uno scarto eccessivo di sequenze realmente randomiche (Type-1 error); mentre, con un α troppo piccolo si tende ad accettare come randomiche anche sequenze che non lo sono (Type-2 error). Nello specifico, nei test realizzati nella Capitolo 4, per ogni test con risultato $X_{risultante}$ viene valutato il corrispondente PValue, ovvero $P(X > X_{risultante})$. Questo parametro viene confrontato direttamente con α e viene quindi dedotto il successo ($PValue \geq \alpha$) o il fallimento ($PValue < \alpha$) del test.

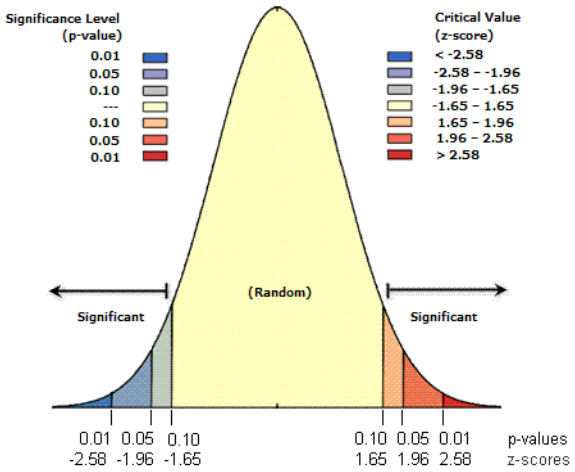

Figure 1.3: Grafico che mostra i valori di soglia x_a (qui riportati come z-score) corrispondenti a diversi valori di PValue, considerando come distribuzione di riferimento una distribuzione normale

Indichiamo con $s = s_0, s_1, s_2, ..., s_{n-1}$ una sequenza binaria di lunghezza n.

1.2.2 Cinque semplici test

I criteri di seguito riportati sono 5 semplici test probabilistici che sono necessari (e ovviamente non sufficienti) affinchè un generatore di numeri random possa essere considerato tale.

Frequency test (monobit test)

Definiamo con n_0 e n_1 rispettivamente il numero di '0' ed il numero di '1' che compaiono nella sequenza s. Viene valutato il parametro

$$X_1 = \frac{(n_0 - n_1)^2}{n}$$

che deve seguire approsimativamente una distribuzione χ^2 con grado di libertà 1 se $n >= 10$. Lo scopo di questo test è valutare se il numero di 0 ed il numero di 1 sia approsimativamente lo stesso.

Serial test (two bit test)

Definiamo con n_0 il numero di 0, con n_1 il numero di 1, con n_{00} il numero di occorrenze di 00, con n_{01} il numero di occorrenze di 01, con n_{10} il numero di occorrenze di 10 e con n_{11} il numero di occorrenze di 11. Dove $n_{00} + n_{01} +$

$n_{10} + n_{11} = (n-1)$ in quanto le sequenze possono sovrapporsi tra loro. Viene valutato il parametro

$$X_2 = \frac{4}{(n-1)} \cdot (n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{n} \cdot (n_0^2 + n_1^2) + 1$$

che deve seguire approsimativamente una distribuzione χ^2 con grado di libertà 2 se $n >= 21$. Lo scopo di questo test è valutare se il numero di occorrenze delle sequenze 00, 01, 10 e 11 sia approsimativamente lo stesso.

Poker test

Sia m un intero positivo tale che $\lfloor \frac{n}{m} \rfloor \geq 5 \cdot (2^m)$ e sia $k = \lfloor \frac{n}{m} \rfloor$. Si divida la sequenza s in k parti disgiunte di lunghezza m e si denoti con n_i il numero di occorrenze della i-esima tipologia di sequenza lunga m, $1 \leq i \leq 2^m$. Viene valutato il parametro

$$X_3 = \frac{2^m}{k} \cdot (\sum_{i=1}^{2^m} n_i^2) - k$$

che deve seguire approsimativamente una distribuzione χ^2 con grado di libertà $2^m - 1$. Lo scopo di questo test è valutare se il numero di occorrenze delle sequenze di lunghezza m sia approsimativamente lo stesso.

Runs test

Si definisce "corsa" di una sequenza s una sottosequenza di consecutivi 0 o consecutivi 1 che non sia preceduta o seguita dallo stesso simbolo. In particolare, una corsa di 0 si definisce "buco", mentre una corsa di 1 si definisce "blocco". Il numero di buchi (o blocchi) di lunghezza i in una sequenza random di lunghezza n dovrebbe essere $e_i = \frac{(n-i+3)}{2^{(i+2)}}$. Sia k il più grande intero i per il quale vale la relazione $e_i \geq 5$. Sia B_i e G_i il numero di blocchi e buchi, rispettivamente, di lunghezza i nella sequenza s per ogni i, $1 \leq i \leq k$. Viene valutato il parametro

$$X_4 = \sum_{i=1}^{k} \frac{(B_i - e_i)^2}{e_i} + \sum_{i=1}^{k} \frac{(G_i - e_i)^2}{e_i}$$

che deve seguire approsimativamente una distribuzione χ^2 con grado di libertà $2 \cdot k - 2$. Lo scopo di questo test è valutare se il numero di corse (sia buchi che blocchi) di diverse lunghezze nella sequenza s sia quello tipico di una sequenza random.

Autocorrelation test

Sia d un intero prefissato tale che $1 \leq d \leq \lfloor \frac{n}{2} \rfloor$. Il numero di bits in s diverso dalla sequenza shiftata di d bits vale $A(d) = \sum_{i=0}^{n-d-1}(s_i \oplus s_{i+d})$. Viene valutato il parametro

$$X_5 = 2 \cdot \frac{(A(d) - \frac{n-d}{2})}{\sqrt{n-d}}$$

che deve seguire approsimativamente una distribuzione normale $N(0,1)$ se $n - d \geq 10$. Lo scopo di questo test è valutare la correlazione tra la sequenza s e sue versioni shiftate (non cicliche).

1.2.3 Altri test statistici

Test for the Longest Run of Ones in a Block

Il criterio alla base di questo test è la valutazione della sequenza di 1 più lunga presente in blocchi di M-bit presi dalla sequenza originaria. Lo scopo è vedere se la sequenza di 1 più lunga individuata è simile a quella tipica di una sequenza random. Si noti che una irregolarità individuata da questo test corrisponde ad una irregolarità anche nella sequenza di 0 più lunga in un blocco.

Discrete Fourier Transform (Spectral) Test

Il criterio alla base di questo test è l'analisi spettrale della sequenza binaria in input usando la trasformata di Fourier discreta. Analizzando quindi il segnale nel dominio delle frequenze si cercano di individuare eventuali pattern periodici che darebbero prova della non casualità della sequenza di input. In particolare, si valuta se il numero di picchi (in ampiezza) che superano la soglia del 95% del picco massimo sono significativamente differenti dal 5% del numero di picchi totale. Risulta appurato come per condurre al meglio questo test sia necessario considerare sequenze di input molto grandi ($>10^6$), al fine di ridurre al minimo gli errori di tipologia 2 [2].

Linear Complexity Test

Il criterio alla base di questo test è la valutazione della lunghezza di un registro a scorrimento a retroazione lineare (LFSR), ovvero un registro i cui bit sono derivati da una funzione lineare dello stato precedente, che possa rappresentare la sequenza di input s. In particolare, si suddivide la sequenza di input in N blocchi di M-bit ciascuno e si valuta con l'algoritmo di Berlekamp-Massey [3] la complessità lineare L_i di ciascun blocco. Quindi, per complessità lineare L_i si intende la lunghezza della più corta sequenza LFSR che genera

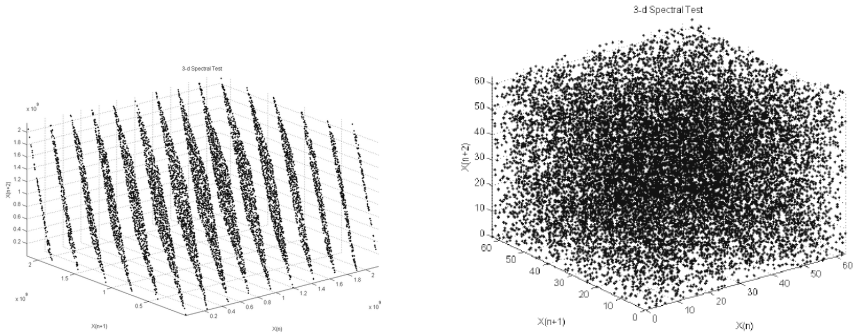

Figure 1.4: LCG spectral test (left) vs random generator spectral test (right) [1]

tutti i bit del blocco N_i. Ovviamente, più la sequenza è randomica più ci si aspetta di avere L_i grandi.

Approximate Entropy Test

Il criterio alla base di questo test è la valutazione delle frequenze dei possibili pattern di sottosequenze sovrapposte di m-bit lungo la sequenza originaria s. Lo scopo è comparare le frequenze delle sottosequenze sovrapposte di m-bit rispetto alle frequenze delle sottosequenze sovrapposte di $(m+1)$-bit. In pratica, data una sequenza di input $s = s_1, ..., s_n$ e definito un parametro m, si considerano le n sottosequenze $S_1, ..., S_n$ di m-bit andando ad aggiungere in coda ad s gli $m-1$ bit iniziali. Per esempio, con $m = 3$ si avrà: $S_1 = s_1, s_2, s_3$, $S_2 = s_2, s_3, s_4$, ..., $S_N = s_n, s_1, s_2$. Quindi, si considerano le n sottosequenze $P_1, ..., P_n$ di $m+1$-bit andando ad aggiungere in coda ad s gli m bit iniziali. Infine, si confrontano le frequenze dei pattern delle sequenze S_i con le frequenze dei pattern delle sequenze P_i.

Cumulative Sums (Cumsum) Test

Il criterio alla base di questo test è la valutazione dell'escursione massima (rispetto allo 0) delle somme cumulate dei bit della sequenze, aggiustati all'intervallo (-1, +1). Lo scopo è vedere quanto la somma cumulata dei valori aggiustati di sequenze parziali della sequenza originaria si discosta dal valore di riferimento di una random walk, che dovrebbe essere pari a zero. In particolare, l'algoritmo è il seguente:

1. Data una sequenza $s = s_1, ..., s_n$ in input, normalizza i bit di s a valori X nell'intervallo (-1, +1), attraverso la sequente trasformazione: $X_i = 2 \cdot s_i - 1$;

2. Calcola le somme parziali S_i di sottosequenze progressivamente larghe a partire da X_1: $S_k = S_{k-1} + X_k$;

3. Calcola la statistica $z = \max_{1 \leq k \leq n}(|S_k|)$, ovvero la somma parziale più grande in valore assoluto;

4. Calcola il $PValue$ come
$1 - \sum_{k=\frac{\frac{-n}{z}+1}{4}}^{\frac{\frac{n}{z}-1}{4}} [\phi(\frac{(4 \cdot k+1) \cdot z}{\sqrt{n}}) - \phi(\frac{(4 \cdot k-1) \cdot z}{\sqrt{n}}] + \sum_{k=\frac{\frac{-n}{z}-3}{4}}^{\frac{\frac{n}{z}-1}{4}} [\phi(\frac{(4 \cdot k+3) \cdot z}{\sqrt{n}}) - \phi(\frac{(4 \cdot k+1) \cdot z}{\sqrt{n}}]$,
dove $\phi(z) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot \int_{-\infty}^{z} e^{\frac{-u^2}{2}} \, du$

Random Excursions Test

Il criterio alla base di questo test è l'analisi dei cicli definiti dalla somma cumulata della sequenza di input s. In particolare, si definisce ciclo di una somma cumulata $S_1, ..., S_n$ una sequenza di passi di lunghezza unitaria $C_1, ..., C_k$ che partono e tornano all'origine, ovvero tali che $C_1 = C_k = 0$. Lo scopo di questo test è valutare il numero di occorrenze degli stati $[-4, -3, -2, -1, +1, +2, +3, +4]$ all'interno di ogni ciclo C e paragonarlo ad una distribuzione tipica di una sequenza randomica.

Random Excursions Variant

Test uguale alla variante 1, che però considera come possibili stati i 18 valori:

$$[-9, -8, -7, -6, -5, -4, -3, -2, -1, +1, +2, +3, +4, +5, +6, +7, +8, +9]$$

Non Overlapping Template Matching Test

Il criterio alla base di questo test è l'analisi delle frequenze di specifiche sequenze di bit prefissate nella sequenza s. In particolare, si analizza la sequenza s attraverso una finestra di m-bit a scorrimento. Se il pattern non viene trovato, la sequenza scorre di un bit verso destra. Invece, se il pattern viene trovato, la finestra viene impostata al bit successivo a quello del pattern trovato, e la ricerca ricomincia. Lo scopo di questo test è individuare quei generatori che producono troppe occorrenze di un partiolcare pattern scelto a priori, non periodico. Ad esempio, nei test riportati nella sezione 4 la finestra è settatat a m = 9, e quindi si possono individuare fino a 148 template non periodici.

Overlapping Template Matching Test

Come il test sopra riportato, il criterio alla base di questo test è l'analisi delle frequenze di specifiche sequenze di bit prefissate nella sequenza s. Anche

in questo test si analizza la sequenza s attraverso una finestra di m-bit a scorrimento. Differentemente dal Non Overlapping Template Matching Test, se il pattern viene trovato, la finestra di ricerca viene fatta shiftare di un bit verso destra e non impostata al bit successivo a quello del pattern trovato.

Binary Matrix Rank Test

Il criterio alla base di questo test è l'analisi del rango di sottomatrici della sequenza di input s. Lo scopo di questo test è verificare eventuali dipendenze lineari tra sottosequenze di lunghezza prefissata della sequenza originaria.

1.2.4 Criterio di Maurer

Il criterio di Maurer [4], anche detto "Test statistico universale di Maurer", valuta quanto una sequenza binaria possa venire compressa senza perdita di informazione. L'idea alla base è che non dovrebbe essere possibile comprimere in modo significativo una sequenza binaria random.
Per fare ciò la sequenza binaria s viene partizionata in $Q + K$ blocchi disgiunti di lunghezza L-bit, con L scelto tra [6, 16]. Indichiamo con b_i il numero intero la cui rappresentazione binaria corrisponde all'i-esimo blocco. Viene quindi applicato il seguente algoritmo:
INPUT: Una sequenza binaria $s_0, ..., s_{n-1}$ di lunghezza n ed i parametri L, Q e K.
OUTPUT: Il valore della statistica X_u relativa alla sequenza s.

1. Azzera la tabella T: per ogni j da 0 a $2^L - 1$: $T[j] \leftarrow 0$;
2. Inizializza la tabella T: per ogni i da 1 a Q: $T[b_i] \leftarrow i$. In modo tale che $T[b_i]$ contenga l'indice dell'ultimo blocco di Q in cui è presente b_i;
3. $sum \leftarrow 0$;
4. Per i compreso tra $Q + 1$ e $Q + K$ esegui:
   (a) $sum \leftarrow sum + \log(i - T[b_i])$;
   (b) $T[b_i] \leftarrow i$.
5. $X_u \leftarrow \frac{sum}{K}$;
6. Return X_u.

Definiti L e K esistono tabelle che riportano la media μ e la varianza σ^2 caratteristiche delle sequenze casuali. Pertanto, affinche la sequenza in input s possa superare il test di Maurer è necessario che la statistica $Z_u = \frac{(X_u - \mu)}{\sigma}$

segua approsimativamente una distribuzione normale $N(0, 1)$.
Questo test viene definito "universale" perchè permette di identificare una grande varietà di errori che un generatore di numeri pseudocasuali può avere, inclusi i cinque semplici test definiti nel paragrafo precedente. Però, un lato negativo del criterio di Maurer rispetto a test statistici sopra riportati è che Maurer richiede una sequenza di lunghezza molto superiore per essere effettivo.

1.3 Generatori di numeri pseudocasuali

In questo capitolo andremo a definire in modo formale cosa sono i generatori di numeri casuali e pseudocasuali, per poi elencare quelli comunemente utilizzati e quindi oggetto dei nostri test.

1.3.1 Definizioni

Definizione. *Un generatore di bit casuali è un dispositivo hardware o un algoritmo software che genera in output una sequenza di digit statisticamente indipendenti.*

Ovviamente, un generatore di bit casuali può essere impiegato per generare numeri casuali. Infatti, un numero intero nell'intervallo [0, n] può essere ottenuto generando una sequenza di [lg n] +1 bits.
Un vero generatore di numeri casuali richiede in input una qualche fonte di casualità tipica della natura. Per esempio un generatore hardware-based può sfruttare la casualità tipica di alcuni fenomeni fisici, come il suono di un microfono oppure il video di una videocamera. Mentre, un generatore software è più complesso e può sfruttare ad esempio il clock di sistema, le statistiche di rete o il tempo che intercorre tra due successivi movimenti del mouse. Ovviamente, creare un generatore di bit casuali è un compito difficile e soggetto ad influenze esterne, come malfunzionamenti. Per questo motivo è sempre preferibile non affidarsi ad una sola fonte di casualità ma cercare di combinare attraverso funzioni complesse diverse fonti.
In particolare, la qualità di un generatore di bit casuali si misura in termini di:

- imparzialità: la probabilità di generare un 1 piuttosto che uno 0, che deve essere il più possibile pari a 0.5;
- correlazione: la probabilità che l'emissione di un 1 piuttosto che uno 0 dipenda dai bits emessi precedentemente, che deve essere il più possibile pari a 0.

Invece:

Definizione. *Un generatore di bit pseudocasuali (PRBG) è un algoritmo deterministico che, data in input una sequenza binaria realmente casuale di lunghezza k, genera in output una sequenza binaria di lunghezza $l >> k$, che "sembra" casuale. L'input al PRBG viene detto "seme", mentre l'output del PRBG viene detto "sequenza di bit pseudocasuali".*

Certamente, l'output di un PRBG non è random. Infatti, il numero di possibili sequenze di output casuali è una frazione molto ridotta ($\frac{2^k}{2^l}$) di tutte le

possibili sequenze binarie di lunghezza l. Pertanto, l'obiettivo di un generatore di numeri pseudocasuali è quello di prendere in input una breve sequenza binaria casuale ed espanderla in una sequenza più lunga, che però appaia comunque casuale e sia indistiguibile da una vera sequenza casuale di lunghezza l agli occhi di un attacker.
Per definire un PRBG come "crittograficamente sicuro" (CSPRBG) è necessario che il generatore passi il next-bit test sotto definito.

Definizione. *Un generatore di bit pseudocasuali passa il next-bit test se non esiste algoritmo polinomiale che, prendendo in input i primi l bits di una sequenza di output s, riesce a predire il* $(l+1)$*-esimo bit di s con una probabilità maggiore di 0.5.*

Passare il next-bit test equivale a passare il polynomial-time statistical test sotto definito.

Definizione. *Un generatore di bit pseudocasuali passa il polynomial-time statistical test se non esiste algoritmo polinomiale che riesce a distinguere correttamente tra l'ouput del generatore e una reale sequenza casuale della stessa lunghezza con probabilità significativamente maggiore di 0.5.*

1.3.2 Linear Congruential Generator (LCG)

Basato sugli studi di Fishman e Moore [5], il generatore LCG prende in input un seme z_0 e genera una sequenza di bit usando la formula

$$z_{i+1} = a \cdot z_i \pmod{2^{31} - 1} \text{ , per } i \geq 0$$

dove a è una funzione relativa allo stato corrente. In particolare, ad ogni step z_i viene convertito ad un valore nel campo [0, 1] e quindi riportato in binario: '0' se ≤ 0.5, altrimenti '1'.

1.3.3 Quadratic Congruential Generator I (QCG-I)

Il generatore QCG-I prende in input un numero primo p di 512 bit ed un seme random x_0 anch'esso di 512 bit, creando in output una sequenza di elementi (numeri a 512 bit) secondo la formula

$$x_{i+1} = x_i^2 \pmod{p} \text{ , per } i \geq 0$$

.

1.3.4 Quadratic Congruential Generator II (QCG-II)

Il generatore QCG-II prende in input un seme random x_0 di 512 bit, creando in output una sequenza di elementi (numeri a 512 bit) secondo la formula

$$x_{i+1} = 2 \cdot x_i^2 + 3 \cdot x_i + 1 \pmod{2^{512}} \text{ , per } i \geq 0$$

.

1.3.5 Cubic Congruential Generator II (CCG)

Il generatore CCG prende in input un seme random x_0 di 512 bit, creando in output una sequenza di elementi (numeri a 512 bit) secondo la formula

$$x_{i+1} = x_i^3 \pmod{2^{512}} \text{ , per } i \geq 0$$

.

1.3.6 Exclusive OR Generator (XORG)

Il generatore CCG prende in input un seme random di 127 bit $x_1, x2, ..., x_{127}$, creando in output una sequenza di bit secondo la formula

$$x_i = x_{i-1} \oplus x_{i-127} \text{ , per } i \geq 128$$

.

1.3.7 Modular Exponentiation Generator (MODEXPG)

Il generatore CCG prende in input un numero primo p di 512 bit, una base g di 512 bit e un seme y di 160 bit, creando in output una sequenza di numeri a 512 bit secondo le formula

$$x_1 = g^y \pmod{p}$$

$$x_{i+1} = g^{y_i} \pmod{p} \text{ , per } i \geq 1$$

dove y_i sono i 160 bit meno significativi di x_i.

1.3.8 Secure Hash Generator (G-SHA1)

Il generatore G-SHA1, anche chiamato FIPS-186, è un generatore di numeri pseudocasuali approvato appunto dalla FIPS (Federal Information Processing Standards) che utilizza la funzione hash SHA1. Questo generatore prende in

input un seme s di 160 bit ed una chiave k di lunghezza compresa tra [160, 512] bit, generando in output un numero pseudocasuale di 160 bit secondo la seguente procedura: [6]
INPUT:

- Una stringa di 160-bit s;
- Una stringa di b-bit k, $160 \leq b \leq 512$.

OUTPUT: Una stringa di 160-bit definita G(s, k).

1. Suddividi s in cinque blocchi da 32 bit ciascuno: $s = H_1|H_2|H_3|H_4|H_5$;
2. Riempi k con degli 0 per ottenere un blocco di lunghezza 512: $X \leftarrow k|0^{512-b}$;
3. Dividi X in 16 parole da 32 bit: $x_0, .., x_15$, e imposta $m \leftarrow 1$;
4. Esegui 4 volte l'algoritmo SHA-1. (Questo modifica i H_i's.);
5. L'output è la concatenazione di: G(s, k) = $H_1|H_2|H_3|H_4|H_5$.

1.3.9 Blum-Blum-Shub (BBSG)

Il generatore BBSG [7], anche chiamato generatore $x^2 \pmod n$, è un generatore di numeri pseudocasuali che prende in input due numeri primi p e q ed un seme s, creando in output una sequenza $z_1, ..., z_l$ di l bit pseudocasuali seguendo la seguente procedura: [6]
INPUT:

- Due numeri primi distinti p e q generati casualmente, ciascuno pari a 3 modulo 4. Si denota con $n = p \cdot q$;
- Un numero intero s nell'intervallo [1, n-1] tale che gcd(s,n)=1. Si denota con $x_0 = s^2 \pmod n$.

OUTPUT: Una sequenza $z_1, ..., z_l$ di l bit pseudocasuali.

1. Per i compreso tra 1 e l esegui:
   (a) $x_i = x_{i-1}^2 \pmod n$;
   (b) $z_i \leftarrow$ il bit meno significativo di x_i.
2. La sequenza di output è $z_1, z_2, ..., z_l$.

1.3.10 Micali-Schnorr Generator (MSG)

Il generatore MSG [8], anche chiamato generatore RSA, è un generatore di numeri pseudocasuali che prende in input due numeri primi p e q, un intero e ed una sequenza random inziale x_0, creando in output una sequenza $z_1, ..., z_l$ di bit pseudocasuali seguendo la seguente procedura: [6]
INPUT:

- Due numeri primi distinti p e q generati casualmente, come nel protocollo RSA. Si denota con $n = p \cdot q$, con $\phi = (p-1) \cdot (q-1)$ e con $N = \lfloor \ln n + 1 \rfloor$;
- Un numero intero e nell'intervallo $]1, \phi[$, tale che gcd(e,ϕ)=1 e $80 \cdot e \leq N$. Si denota con $k = \lfloor N \cdot (1 - \frac{2}{e}) \rfloor$ e con $r = N - k$;
- Una sequenza random x_0 (seme) di lunghezza r bit.

OUTPUT: Una sequenza $z_1, ..., z_l$ di $k \cdot l$ bit pseudocasuali.

1. Per i compreso tra 1 e l esegui:
   (a) $y_i \leftarrow x_{i-1}^e \pmod n$;
   (b) $x_i \leftarrow$ gli r bit più significativi di y_i;
   (c) $z_i \leftarrow$ i k bit meno significativi di y_i.
2. La sequenza di output è $z_1|z_2|...|z_l$.

1.3.11 Permuted Congruential Generator (PCG)

Il generatore PCG è un algoritmo per la generazione di numeri pseudo-casuali molto compatto e di rapida esecuzione ideato da Melissa E. O'Neill [9]. In particolare, questo generatore è progettato per fornire buone proprietà statistiche, come periodi più lunghi e migliore casualità, rimanendo efficiente dal punto di vista computazionale. Esso combina tecniche provenienti da diversi tipi di generatori di numeri casuali (RNG), come gli spostamenti bit-a-bit e le permutazioni. Nello specifico, il PCG è parametrizzato da diversi valori, tra cui:

- la dimensione dello stato del generatore ("PCG_STATE_SIZE");
- la costante di moltiplicazione ("PCG_MULTIPLIER");
- la costante di incremento ("PCG_INCREMENT");
- le costanti di permutazione.

Questi parametri possono essere regolati per creare diverse istanze del PCG con proprietà variabili.
Nella sua implementazione minimale in C (da noi utilizzata per la realizzazione dei test statistici) si evidenziano le fasi di transizione di stato e di permutazione tipiche di questo generatore di numeri pseudocasuali:

```c
//Define the PCG parameters
#define PCG_MULTIPLIER 6364136223846793005ULL
#define PCG_INCREMENT 0x14057b7ef767814fULL
#define PCG_STATE_SIZE 64
//Define the PCG state struct
typedef struct {
    uint64_t state;
    uint64_t inc;
} pcg_state;
//Initialize the PCG state
void pcg_init(pcg_state* rng_state, uint64_t seed) {
    rng_state->state = seed;
    rng_state->inc = PCG_INCREMENT;
}
// Generate a random bit using PCG
uint32_t pcg_rand(pcg_state* rng_state) {
    uint64_t oldstate = rng_state->state;
    rng_state->state = oldstate * PCG_MULTIPLIER + (rng_state->inc|1);
    uint32_t xorshifted = ((oldstate >> 18u) ^ oldstate) >> 27u;
    uint32_t rot = oldstate >> 59u;
    return ((xorshifted >> rot) | (xorshifted << ((-rot) & 31)));
}
```

Code 1.1: Implementazione in C dell'algoritmo del Permuted Congruential Generator

In particolare, nella nostra implementazione in C, viene restituito un numero unsigned int a 32 bit, i cui singoli bit vengono aggiunti alla sequenza sulla quale vengono poi eseguiti i test.

1.3.12 Mersenne-Twister Generator (METW)

Il generatore METW è un algoritmo di generazione di numeri pseudocasuali molto diffuso, ideato nel 1997 da Makoto Matsumoto e Takuji Nishimura [10]. Questo è l'algoritmo usato ad esempio come standard PRNG per molti linguaggi di programmazione, tra i quali: C++11, Python, Matlab, R e Julia. Il nome "Mersenne" deriva dal fatto che il periodo del generatore è un numero primo di Mersenne, ovvero un numero primo della forma $2^p - 1$. Infatti, il METW ha un periodo di $2^{19937} - 1$, il che significa che può produrre $2^{19937} - 1$ distinti valori pseudo-casuali prima di ripetersi.
Il funzionamento di questo generatore si basa sul mantenimento di uno stato interno composto da 624 numeri a 32 bit, che vengono combinati opportunamente per realizzare l'output. Nella sua implementazione minimale in C (da noi utilizzata per la realizzazione dei test statistici) si evidenziano le fasi di inizializzazione e di transizione di stato e la funzione combinatoria di generazione del numero pseudocasuale:

```c
//Define the Marsenne Twister parameters
#define MT_SIZE 624
//Define the PCG state
static uint32_t mt[MT_SIZE];
static int index_mt = 0;
// Initialize the Mersenne Twister with a seed
void mt_seed(uint32_t seed) {
    mt[0] = seed;
    for (int i = 1; i < MT_SIZE; i++) {
        mt[i] = 1812433253UL * (mt[i - 1] ^ (mt[i - 1] >> 30)) + i;
    }
}
// Generate a random number using the Mersenne Twister
uint32_t mt_rand() {
    if (index_mt == 0) {
        // State transition function
        for (int i = 0; i < MT_SIZE; i++) {
            uint32_t y = (mt[i] & 0x80000000UL) + (mt[(i + 1) %
                MT_SIZE] & 0x7fffffffUL);
            mt[i] = mt[(i + 397) % MT_SIZE] ^ (y >> 1);
            if (y % 2 != 0) {
                mt[i] ^= 0x9908b0dfUL;
            }
        }
    }
    // Pseudo-random bit generation
    uint32_t y = mt[index_mt];
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);
    index_mt = (index_mt + 1) % MT_SIZE;
    return y;
}
```

Code 1.2: Implementazione in C dell'algoritmo di Mersenne Twister

Anche in questo caso, nella nostra implementazione in C, viene restituito un numero unsigned int a 32 bit, i cui singoli bit vengono aggiunti alla sequenza sulla quale vengono poi eseguiti i test.

1.3.13 Generatori basati sull'espansione binaria di numeri

Come già anticipato, in questo articolo andremo anche a confrontare la randomicità di numeri come Pigreco, il numero di Nepero, la radice quadrata di 2 e la radice quadrata di 3. Per fare ciò, si è realizzato uno script in Wolfram Mathematica al fine di creare dei file ASCII contenenti le espansioni binarie delle cifre decimali di questi numeri sotto forma di stringhe di '0' e '1'. Si riporta quindi lo script utilizzato per la generazione dei file "data.pi", "data.e", "data.sqrt2" e "data.sqr3", che contengono l'espansione binaria dei primi 150000000 bit dei rispettivi numeri.

```
binaryExpansion = RealDigits[, 2, 150000000][[1]];
asciiData = StringJoin[ToString /@ binaryExpansion];
Export["data.pi", asciiData, "Text"]

binaryExpansion = RealDigits[E, 2, 150000000][[1]];
asciiData = StringJoin[ToString /@ binaryExpansion];
Export["data.e", asciiData, "Text"];

binaryExpansion = RealDigits[Sqrt[2], 2, 150000000][[1]];
asciiData = StringJoin[ToString /@ binaryExpansion];
Export["data.sqrt2", asciiData, "Text"];

binaryExpansion = RealDigits[Sqrt[3], 2, 150000000][[1]];
asciiData = StringJoin[ToString /@ binaryExpansion];
Export["data.sqrt3", asciiData, "Text"];
```

Code 1.3: Espansione binaria numeri in Wolfram Mathematica

Nella capitolo 4 andremo quindi ad analizzare i risultati dei test statistici su 4 ulteriori generatori: Pi-greco-Gen, Nepero-Gen, Sqrt2-Gen e Sqrt3-Gen, basati sull'espansione binaria delle cifre decimali degli omonimi numeri.

Chapter 2

Lavori simili

Dalla data del suo rilascio nel 2001, la NIST Test Suite [11] ha rappresentato il punto di riferimento per la validazione dei generatori di numeri casuali. Pertanto, questo software è stato oggetto di molti studi e successive modifiche, che l'hanno migliorato fino alla versione da noi oggi utilizzata, ovvero la NIST SP 800-22 Rev.1a. In particolare, sono stati realizzati studi relativi alla correlazione esistente tra i diversi test che la suite implementa. Nello specifico, Emil Simion e Paul Burciu in [12] hanno dimostrato l'esistenza di una forte correlazione tra ben 5 coppie di test statistici (frequency-cumulative sums forward, frequency-cumulative sums reverse, cumulative sums forward-cumulative sums reverse, random excursions-random excursions variant e serial1-serial2) e pertanto consigliano l'esecuzione di un solo test per ognuna di queste coppie, al fine di risparmiare tempo di calcolo. Inoltre, Liua et.Al in [13] hanno condotto studi relativi alla percentuale di test della suite soddisfatti che un generatore di numeri casuali ideale dovrebbe ottenere, ovvero circa il 60-70%. Infine, un lavoro simile a quello da noi condotto è stato realizzato da Hegadi et.Al in [14], che hanno elaborato un confronto tra i generatori di numeri casuali built-in nei diversi linguaggi di programmazione. La loro ricerca ha portato a dimostrare come in termini di randomness l'algoritmo migliore sia quello implementato da Java e C#, mentre il peggiore quello di Perl.
Non sono presenti, a nostra conoscenza, lavori che si impegnino a classificare la bontà dei diversi generatori di numeri pseudocasuali riportati nel capitolo 1.3 e che si impegnino a verificare la randomicità di numeri come: il numero di Nepero, Pi-greco, la radice di 2 e la radice di 3.

Chapter 3

Metodo

Per verificare la randomicità dei generatori di numeri pseudocasuali riportati attraverso la valutazione delle varie metriche definite si è utilizzato un pacchetto software molto diffuso: la suite di test statistici del NIST(National Institute of Standards and Technology) [11].

3.1 NIST statistical test suite

La suite di test statistici del NIST (NSTS) è un importante sistema di valutazione statistica di randomicità, spesso usato per approvazioni e certificazioni formali di generatori di numeri casuali. In particolare, NSTS è stata utilizzata per scegliere il generatore delle chiavi del protocollo AES (Advanced Encription Standard). Questa test suite implementa di default tutte le 15 metriche di valutazione viste nel paragrafo 1.2 e tutti i generatori di numeri pseudocasuali analizzati nel paragrafo 1.3. Si riporta quindi una tabella che riassume i vari test realizzati dalla suite, consigliando il numero minimo di sequenze n, la dimensione dei singoli blocchi m o M da analizzare (nel caso di test che agiscono per blocchi) e il numero di subtest *subtests* tipico del test in questione con i parametri consigliati:

Test #	Test name	n	m or M	# sub-tests
1.	Frequency	$n \geq 100$	-	1
2.	Frequency within a Block	$n \geq 100$	$20 \leq M \leq n/100$	1
3.	Runs	$n \geq 100$	-	1
4.	Longest run of ones	$n \geq 128$		1
5.	Rank	$n > 38\,912$	-	1
6.	Spectral	$n \geq 1000$	-	1
7.	Non-overlapping T. M.	$n \geq 8m - 8$	$2 \leq m \leq 21$	148*
8.	Overlapping T.M.	$n \geq 10^6$		1
9.	Maurer's Universal	$n > 387\,840$		1
10.	Linear complexity	$n \geq 10^6$	$500 \leq M \leq 5000$	1
11.	Serial		$2 < m < \lfloor \log_2 n \rfloor - 2$	2
12.	Approximate Entropy		$m < \lfloor \log_2 n \rfloor - 5$	1
13.	Cumulative sums	$n \geq 100$		2
14.	Random Excursions	$n \geq 10^6$		8
15.	Random Excursions Variant	$n \geq 10^6$		18

Figure 3.1: Test realizzati dalla suite NIST, con i valori consigliati n, m o M ed il numero di subtest subtests tipico del test in questione

[15]

Una volta lanciato il software ed eseguiti i test su un particolare generatore scegliendo la lunghezza n delle sequenze generate ed il numero *bitstreams* di sequenze da analizzare, la suite riporta in un file ("finalAnalysisReportXXX.txt") una tabella riassuntiva dei risultati dei vari test sul generatore di numeri pseudocasuali scelto. In particolare, l'intervallo di PValue $[0, 1)$ viene suddiviso in 10 sottointervalli C_i, in cui viene appunto riportato il numero di volte in cui la sequenza analizzata ha avuto un PValue compreso tra $0.i - 1$ e $0.i$. Vengono poi riportate due ulteriori colonne:

- PValue: riporta la distribuzione statistica dei vari PValue ottenuti dalle *bitstreams* sequenze analizzate. In particolare, valuta se i PValue sono distribuiti uniformemente nell'intervallo [0, 1), come ci si aspetterebbe da una sequenza perfettamente randomica. Nello specifico, dividendo l'intervallo in 10 sottointervalli C_i ci si aspetta che approsimativamente ricadano $\frac{bitstreams}{10}$ valori in ogni sottointervallo;

- Proportion: riporta la frazione di sequenze testate che hanno superato il test con il livello di significatività α impostato. Ovvero viene valutato il rapporto $\frac{N_{sequenze_{OK}}}{bitstreams}$, dove $N_{sequenze_{OK}} \leftarrow$ numero di sequenze tali che $PValue \geq \alpha$.

Al termine del file "finalAnalysisReportXXX.txt" la suite riporta anche i valori soglia di PValue (β_{PValue}) e Proportion ($\beta_{Proportion}$) che permettono di determinare quindi se il generatore scelto ha superato un determinato test statistico. Formalmente, si considera superato il test se $PValue \geq \beta_{PValue} \wedge$

$Proportion \geq \beta_{Proportion}$. Si riporta quindi una tabella tipica dei risultati della test suite su 1000 sequenze, ciascuna di 10^6 bit, di un generatore di numeri casuali:

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	Proportion	Test
99	108	91	105	109	104	92	101	93	98	0.920383	0.9910	Frequency
90	89	103	101	111	105	100	94	108	99	0.853049	0.9970	BlockFrequency
90	114	93	114	96	90	102	96	101	104	0.643366	0.9910	CumulativeSums
103	91	101	99	113	97	87	88	114	107	0.506194	0.9930	CumulativeSums
41	44	44	45	50	568	51	48	51	58	0.000000*	0.9970	NonOverlapping
41	44	49	46	47	589	54	41	51	38	0.000000*	1.0000*	NonOverlapping
99	107	99	113	94	100	110	87	91	100	0.733899	0.9940	Serial
104	116	103	96	94	95	101	102	84	105	0.695200	0.9890	Serial
97	107	101	111	115	90	100	94	98	87	0.622546	0.9900	LinearComplexity

Figure 3.2: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 1000 sequenze binarie prodotte da un generatore di numeri casuali.

[15]

3.2 Setup

Per analizzare la qualità e determinare la randomicità dei vari generatori di numeri pseudocasuali sono stati valutati i risultati della NSTS considerando i seguenti parametri:

- α: livello di significatività del test pari a 0.01;
- α_{PValue}: livello di significatività del test relativo alla distribuzione dei PValue. Sebbene il NIST consigli valori di 0.0001, si è deciso di seguire le indicazione riportate in [15] e dimensionare questo parametro come 0.01. In questo modo aumenta la probabilità che il test dia un risultato veritiero;
- n: lunghezza delle singole sequenze pari a 10^6 bit;
- *bitstreams*: numero di sequenze da valutare per ogni singolo test pari a 100;
- Block frequency test con lunghezza del blocco pari a $M = 128$ (default);
- Non overlapping template test con lunghezza del blocco pari a $m = 9$ (default);
- Overlapping template test con lunghezza del blocco pari a $m = 9$ (default);

- Approximate entropy test con lunghezza del blocco pari a $m = 10$ (default);
- Serial test con lunghezza del blocco pari a $m = 16$ (default);
- Linear complexity test con lunghezza del blocco pari a $M = 500$ (default).

Chapter 4

Risultati

Per semplicità di rappresentazione su tabella, i test che prevedono molti subtests (Non Overlapping Template Matching Test, Random Excursions Test e Random Excursions Variant Test) vengono riportati come singola riga i cui valori sono le medie dei valori dei relativi subtests. In particolare, le tabelle di seguito riportate avranno le sequenti colonne:

- "0.i": in cui viene riportato il conteggio dei test il cui PValue ricade nell'intervallo $0.(i-1) \leq PValue \leq 0.i$;
- "PV": in cui viene riportato il PValue come valutazione della distribuzione dei PValue dei singoli test;
- "PROP": proporzione di test superati rispetto al numero totale di test eseguiti;
- "TEST": test in questione;
- "OK": variabile booleana che indica se il test in questione è da considerarsi come soddisfatto (True) o violato (False);
- "PV-OK": variabile booleana che indica se il test relativo alla distribuzione dei PValue è da considerarsi come soddisfatto (True) o violato (False).

Per ogni generatore si è valutata la percentuale relativa alla quantità di test superati rispetto al numero totale di test e la percentuale relativa alla quantità di test di uniformità dei PValue superati, sempre rispetto al numero totale di test. Queste due percentuali sono state moltiplicate per fornire un punteggio $Q \in [0, 1]$ che indichi la qualità complessiva di ciascun generatore, utilizzabile per realizzare confronti di qualità.

4.1 Risultati LCG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	6.00	15.00	8.00	9.00	11.00	7.00	10.00	11.00	13.00	10.00	0.68	0.99	Frequency	True	True
1	11.00	7.00	11.00	6.00	10.00	13.00	15.00	13.00	6.00	8.00	0.44	0.99	BlockFrequency	True	True
2	6.00	12.00	17.00	10.00	3.00	11.00	10.00	15.00	9.00	7.00	0.08	0.99	CumulativeSums	True	True
3	7.00	13.00	11.00	10.00	9.00	10.00	11.00	13.00	11.00	5.00	0.78	0.99	CumulativeSums	True	True
4	12.00	10.00	11.00	8.00	9.00	8.00	14.00	11.00	9.00	8.00	0.94	0.97	Runs	True	True
5	8.00	8.00	10.00	18.00	7.00	9.00	13.00	10.00	11.00	6.00	0.29	1.00	LongestRun	True	True
6	12.00	9.00	9.00	8.00	8.00	7.00	18.00	10.00	10.00	9.00	0.46	0.99	Rank	True	True
7	5.00	11.00	12.00	11.00	13.00	5.00	15.00	13.00	8.00	7.00	0.26	1.00	FFT	True	True
34	4.94	5.78	6.11	5.78	5.94	6.11	5.61	4.94	4.94	5.83	0.41	0.99	RandomExcursions...	True	True
41	4.25	5.38	5.88	4.50	6.00	5.00	6.62	3.88	7.25	7.25	0.30	0.99	RandomExcursions...	True	True
156	12.00	16.00	13.00	10.00	10.00	8.00	7.00	10.00	7.00	7.00	0.53	0.99	OverlappingTemplate	True	True
157	8.00	9.00	8.00	9.00	8.00	12.00	9.00	15.00	15.00	7.00	0.55	0.99	Universal	True	True
158	10.00	9.00	8.00	14.00	14.00	14.00	7.00	9.00	5.00	10.00	0.46	0.99	ApproximateEntropy	True	True
185	7.00	8.00	11.00	17.00	12.00	9.00	14.00	6.00	9.00	7.00	0.28	1.00	Serial	True	True
186	13.00	8.00	7.00	9.00	17.00	8.00	14.00	8.00	9.00	7.00	0.30	0.99	Serial	True	True
187	9.00	12.00	6.00	10.00	15.00	13.00	7.00	11.00	9.00	8.00	0.64	0.99	LinearComplexity	True	True
188	10.59	10.28	10.02	10.00	10.23	9.85	10.09	10.03	9.28	9.64	0.53	0.99	NonOverlappingTe...	False	True

Figure 4.1: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Linear Congruential Generator.*

Come si nota dal file riportato, il LCG passa 186 test su 188, ottenendo una percentuale di successo pari al 98,94%. In particolare, non vengono superati 2 test di NonOverlappingTemplate. Inoltre, i vari valori di P-Value (colonna "PV") sono tutti maggiore dell' α_{PValue} dimensionato e il success rate per questo test è quindi pari al 100%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{98,936}{100}$.

4.2 Risultati QCG-I

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	71.00	10.00	3.00	6.00	1.00	3.00	4.00	0.00	1.00	1.00	0.00	0.55	Frequency	False	False
1	8.00	5.00	9.00	13.00	11.00	11.00	16.00	17.00	7.00	3.00	0.03	0.99	BlockFrequency	True	True
2	71.00	7.00	8.00	4.00	2.00	2.00	1.00	1.00	3.00	1.00	0.00	0.61	CumulativeSums	False	False
3	69.00	13.00	6.00	0.00	3.00	2.00	4.00	3.00	0.00	0.00	0.00	0.62	CumulativeSums	False	False
4	14.00	8.00	13.00	10.00	7.00	8.00	14.00	9.00	9.00	8.00	0.70	0.91	Runs	False	True
5	9.00	12.00	9.00	9.00	9.00	12.00	11.00	12.00	9.00	8.00	0.99	0.98	LongestRun	True	True
6	13.00	5.00	11.00	13.00	11.00	7.00	11.00	11.00	7.00	11.00	0.68	0.99	Rank	True	True
7	17.00	8.00	18.00	15.00	8.00	4.00	8.00	9.00	2.00	11.00	0.00	0.96	FFT	False	False
34	3.28	4.06	3.89	3.33	3.78	3.44	3.33	3.28	3.06	3.56	0.47	0.99	RandomExcursions...	True	False
41	3.50	3.12	3.62	3.25	3.75	3.50	2.88	3.75	4.00	3.62	0.22	0.99	RandomExcursions...	False	True
156	8.00	11.00	16.00	9.00	8.00	11.00	6.00	15.00	7.00	9.00	0.37	0.99	OverlappingTemplate	True	True
157	10.00	6.00	12.00	7.00	16.00	8.00	9.00	10.00	12.00	10.00	0.60	0.99	Universal	True	True
158	15.00	12.00	11.00	8.00	12.00	13.00	7.00	9.00	5.00	8.00	0.47	0.99	ApproximateEntropy	True	True
185	4.00	11.00	16.00	11.00	12.00	4.00	9.00	13.00	10.00	10.00	0.19	1.00	Serial	True	True
186	8.00	10.00	8.00	7.00	11.00	14.00	18.00	5.00	9.00	10.00	0.19	0.99	Serial	True	True
187	10.00	8.00	8.00	5.00	9.00	13.00	10.00	13.00	9.00	15.00	0.55	0.99	LinearComplexity	True	True
188	10.82	10.26	10.20	9.84	9.86	9.74	9.90	9.84	9.86	9.66	0.50	0.99	NonOverlappingTe...	False	True

Figure 4.2: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Quadratic Congruential Generator ver.1 .*

Come si nota dal file riportato, il QCG-I passa 178 test su 188, ottenendo una percentuale di successo pari al 94,68%. In particolare, non vengono superati: il Frequency Test, 2 CumulativeSums Tests, il Runs Test, il FFT (Spectral Test), 4 NonOverlappingTemplate Tests e un RandomExcursion Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 183 test su 188, con un success rate pari al 97,34%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{92{,}163}{100}$

4.3 Risultati QCG-II

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	63.00	12.00	7.00	7.00	2.00	3.00	1.00	1.00	2.00	2.00	0.00	0.74	Frequency	False	False
1	7.00	7.00	14.00	9.00	11.00	6.00	12.00	12.00	11.00	11.00	0.72	0.99	BlockFrequency	True	True
2	63.00	13.00	6.00	2.00	5.00	5.00	3.00	2.00	0.00	1.00	0.00	0.76	CumulativeSums	False	False
3	59.00	14.00	9.00	1.00	7.00	3.00	3.00	2.00	2.00	0.00	0.00	0.72	CumulativeSums	False	False
4	9.00	8.00	9.00	14.00	12.00	10.00	3.00	11.00	9.00	15.00	0.33	0.96	Runs	False	True
5	10.00	9.00	7.00	11.00	7.00	10.00	11.00	10.00	13.00	12.00	0.95	0.99	LongestRun	True	True
6	7.00	12.00	12.00	12.00	5.00	13.00	10.00	5.00	12.00	12.00	0.46	1.00	Rank	True	True
7	33.00	17.00	14.00	8.00	4.00	4.00	7.00	6.00	5.00	2.00	0.00	0.94	FFT	False	False
34	2.94	1.78	4.06	4.28	3.33	3.56	3.39	2.22	2.50	1.94	0.36	0.98	RandomExcursions...	True	True
41	3.00	3.50	3.00	3.50	2.62	3.00	2.38	3.12	2.88	3.00	0.52	0.99	RandomExcursions...	False	True
156	5.00	12.00	13.00	11.00	7.00	12.00	11.00	10.00	9.00	10.00	0.80	1.00	OverlappingTemplate	True	True
157	18.00	11.00	8.00	10.00	8.00	9.00	10.00	4.00	12.00	10.00	0.25	0.98	Universal	True	True
158	8.00	9.00	11.00	10.00	10.00	10.00	5.00	14.00	8.00	15.00	0.57	0.99	ApproximateEntropy	True	True
185	8.00	12.00	12.00	5.00	8.00	9.00	9.00	15.00	7.00	15.00	0.33	0.98	Serial	True	True
186	6.00	10.00	6.00	14.00	12.00	6.00	15.00	8.00	10.00	13.00	0.30	0.99	Serial	True	True
187	16.00	15.00	9.00	11.00	9.00	11.00	6.00	7.00	8.00	8.00	0.37	0.98	LinearComplexity	True	True
188	9.98	10.28	9.80	10.12	10.14	10.40	9.66	9.82	9.91	9.89	0.52	0.99	NonOverlappingTe...	False	False

Figure 4.3: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Quadratic Congruential Generator ver.2 .*

Come si nota dal file riportato, il QCG-II passa 177 test su 188, ottenendo una percentuale di successo pari al 94,15%. In particolare, non vengono superati: il Frequency Test, 2 CumulativeSums Tests, il Runs Test, il FFT (Spectral Test), 4 NonOverlappingTemplate Tests e 2 RandomExcursion Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 182 test su 188, con un success rate pari al 96,81%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{91{,}144}{100}$.

4.4 Risultati CCG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	64.00	12.00	8.00	6.00	5.00	1.00	1.00	1.00	2.00	0.00	0.00	0.67	Frequency	False	False
1	5.00	4.00	10.00	1.00	13.00	7.00	11.00	8.00	16.00	25.00	0.00	1.00	BlockFrequency	True	False
2	61.00	14.00	8.00	4.00	1.00	5.00	3.00	0.00	3.00	1.00	0.00	0.66	CumulativeSums	False	False
3	65.00	6.00	10.00	1.00	6.00	3.00	6.00	0.00	2.00	1.00	0.00	0.71	CumulativeSums	False	False
4	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	Runs	False	False
5	5.00	16.00	9.00	10.00	7.00	13.00	7.00	13.00	9.00	11.00	0.35	0.99	LongestRun	True	True
6	7.00	15.00	9.00	10.00	7.00	10.00	16.00	7.00	13.00	6.00	0.25	0.99	Rank	True	True
7	57.00	9.00	10.00	5.00	5.00	1.00	4.00	3.00	3.00	3.00	0.00	0.85	FFT	False	False
34	2.83	3.83	3.11	3.67	3.44	2.89	3.33	2.56	2.28	2.06	0.41	0.99	RandomExcursions...	True	True
41	2.88	2.62	3.12	2.00	3.38	3.62	3.25	3.38	3.12	2.62	0.63	0.99	RandomExcursions...	False	True
156	8.00	15.00	6.00	7.00	11.00	19.00	9.00	11.00	9.00	5.00	0.06	0.99	OverlappingTemplate	True	True
157	11.00	11.00	8.00	14.00	9.00	11.00	8.00	9.00	10.00	9.00	0.96	0.99	Universal	True	True
158	64.00	12.00	7.00	6.00	5.00	3.00	1.00	2.00	0.00	0.00	0.00	0.77	ApproximateEntropy	False	False
185	14.00	12.00	9.00	8.00	8.00	9.00	8.00	10.00	5.00	17.00	0.29	0.99	Serial	True	True
186	9.00	9.00	12.00	7.00	11.00	7.00	10.00	13.00	9.00	13.00	0.88	0.98	Serial	True	True
187	12.00	7.00	7.00	9.00	12.00	5.00	12.00	16.00	12.00	8.00	0.35	0.97	LinearComplexity	True	True
188	14.82	11.64	10.53	10.24	9.36	9.32	9.26	8.54	8.61	7.68	0.36	0.97	NonOverlappingTe...	False	False

Figure 4.4: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Cubic Congruential Generator.*

Come si nota dal file riportato, il CCG passa 163 test su 188, ottenendo una percentuale di successo pari al 86,70%. In particolare, non vengono superati: il Frequency Test, 2 CumulativeSums Tests, il Runs Test, il FFT (Spectral Test), 18 NonOverlappingTemplate Tests, un RandomExcursion Test e l'ApproximateEntropy Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 160 test su 188, con un success rate pari al 85,11%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{73{,}789}{100}$.

4.5 Risultati XORG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	26.00	15.00	7.00	10.00	8.00	11.00	6.00	9.00	6.00	2.00	0.00	0.89	Frequency	False	False
1	33.00	7.00	2.00	3.00	3.00	2.00	4.00	3.00	13.00	30.00	0.00	0.76	BlockFrequency	False	False
2	29.00	10.00	9.00	11.00	9.00	7.00	12.00	5.00	4.00	4.00	0.00	0.87	CumulativeSums	False	False
3	28.00	15.00	10.00	8.00	5.00	7.00	5.00	8.00	10.00	4.00	0.00	0.86	CumulativeSums	False	False
4	25.00	16.00	8.00	10.00	8.00	9.00	5.00	11.00	6.00	2.00	0.00	0.89	Runs	False	False
5	45.00	9.00	11.00	8.00	6.00	5.00	4.00	3.00	5.00	4.00	0.00	0.69	LongestRun	False	False
6	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	Rank	False	False
7	95.00	1.00	0.00	0.00	2.00	0.00	1.00	0.00	0.00	1.00	0.00	0.08	FFT	False	False
34	4.00	4.44	5.39	5.44	5.61	5.78	5.44	6.00	6.06	5.83	0.36	0.99	RandomExcursions...	True	False
41	6.00	6.12	5.38	5.62	5.00	5.25	4.62	5.00	5.75	5.25	0.43	0.98	RandomExcursions...	True	False
156	50.00	10.00	9.00	11.00	3.00	6.00	4.00	3.00	2.00	2.00	0.00	0.71	OverlappingTemplate	False	False
157	79.00	4.00	3.00	3.00	2.00	3.00	3.00	3.00	0.00	0.00	0.00	0.36	Universal	False	False
158	94.00	1.00	2.00	1.00	0.00	0.00	1.00	0.00	1.00	0.00	0.00	0.07	ApproximateEntropy	False	False
185	98.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.03	Serial	False	False
186	95.00	2.00	2.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.07	Serial	False	False
187	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	LinearComplexity	False	False
188	40.95	7.98	6.22	6.26	5.86	6.08	6.27	6.28	6.40	7.71	0.00	0.72	NonOverlappingTe...	False	False

Figure 4.5: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal XOR Generator.*

Come si nota dal file riportato, il XORG passa 26 test su 188, ottenendo una percentuale di successo pari al 13,83%. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 24 test su 188, con un success rate pari al 12,77%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{1{,}766}{100}$.

4.6 Risultati MODEXPG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	65.00	21.00	5.00	4.00	2.00	1.00	1.00	0.00	1.00	0.00	0.00	0.67	Frequency	False	False
1	10.00	6.00	10.00	10.00	9.00	17.00	10.00	6.00	18.00	4.00	0.03	0.98	BlockFrequency	True	True
2	66.00	15.00	9.00	3.00	1.00	2.00	1.00	1.00	0.00	2.00	0.00	0.72	CumulativeSums	False	False
3	62.00	15.00	13.00	2.00	1.00	2.00	1.00	2.00	0.00	2.00	0.00	0.69	CumulativeSums	False	False
4	10.00	11.00	8.00	9.00	6.00	8.00	12.00	11.00	14.00	11.00	0.85	0.96	Runs	False	True
5	8.00	10.00	6.00	7.00	15.00	10.00	9.00	14.00	9.00	12.00	0.57	0.99	LongestRun	True	True
6	8.00	7.00	12.00	10.00	13.00	8.00	9.00	8.00	11.00	14.00	0.82	0.97	Rank	True	True
7	8.00	10.00	14.00	12.00	10.00	12.00	13.00	7.00	8.00	6.00	0.68	1.00	FFT	True	True
34	2.11	3.61	3.44	3.44	3.89	3.72	4.83	3.83	3.17	3.94	0.30	0.99	RandomExcursions...	True	False
41	3.62	3.62	3.25	2.12	3.38	3.00	4.12	5.00	4.88	3.00	0.35	1.00	RandomExcursions...	True	True
156	12.00	8.00	9.00	11.00	7.00	15.00	6.00	3.00	16.00	13.00	0.08	1.00	OverlappingTemplate	True	True
157	6.00	17.00	14.00	10.00	11.00	6.00	11.00	12.00	4.00	9.00	0.12	1.00	Universal	True	True
158	9.00	14.00	13.00	10.00	14.00	13.00	8.00	8.00	3.00	8.00	0.26	1.00	ApproximateEntropy	True	True
185	13.00	18.00	5.00	8.00	11.00	9.00	8.00	12.00	8.00	8.00	0.21	0.99	Serial	True	True
186	17.00	5.00	12.00	15.00	9.00	10.00	3.00	5.00	12.00	12.00	0.03	0.97	Serial	True	True
187	8.00	10.00	11.00	16.00	5.00	5.00	10.00	13.00	8.00	14.00	0.21	0.99	LinearComplexity	True	True
188	10.34	10.01	10.17	10.09	10.20	10.05	9.66	9.62	9.91	9.95	0.54	0.99	NonOverlappingTe...	False	False

Figure 4.6: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Modular Exponential Generator.*

Come si nota dal file riportato, il MODEXP passa 181 test su 188, ottenendo una percentuale di successo pari al 96,28%. In particolare, non vengono superati: il Frequency Test, 2 CumulativeSums Tests, il Runs Test e 3 NonOverlapping-Template Tests. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 181 test su 188, con un success rate pari al 96,28%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{92{,}692}{100}$.

4.7 Risultati G-SHA1

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	6.00	10.00	21.00	9.00	7.00	9.00	10.00	15.00	7.00	6.00	0.02	0.99	Frequency	True	True
1	10.00	12.00	13.00	8.00	11.00	6.00	9.00	8.00	15.00	8.00	0.66	1.00	BlockFrequency	True	True
2	8.00	12.00	9.00	7.00	9.00	10.00	13.00	6.00	16.00	10.00	0.53	1.00	CumulativeSums	True	True
3	6.00	7.00	10.00	17.00	7.00	10.00	7.00	14.00	12.00	10.00	0.26	1.00	CumulativeSums	True	True
4	4.00	12.00	9.00	8.00	15.00	14.00	11.00	10.00	10.00	7.00	0.38	1.00	Runs	True	True
5	7.00	10.00	12.00	17.00	5.00	9.00	8.00	15.00	8.00	9.00	0.20	0.99	LongestRun	True	True
6	7.00	12.00	11.00	13.00	10.00	10.00	7.00	11.00	5.00	14.00	0.60	1.00	Rank	True	True
7	10.00	11.00	6.00	12.00	8.00	12.00	12.00	10.00	11.00	8.00	0.92	0.99	FFT	True	True
34	6.78	5.28	5.06	5.56	5.78	6.11	5.22	5.17	5.83	5.22	0.48	0.99	RandomExcursions...	True	False
41	6.25	5.88	4.88	5.25	5.12	5.12	6.12	6.00	4.38	7.00	0.34	0.99	RandomExcursions...	False	False
156	7.00	6.00	11.00	7.00	10.00	10.00	9.00	11.00	11.00	18.00	0.33	0.99	OverlappingTemplate	True	True
157	12.00	9.00	9.00	9.00	7.00	12.00	13.00	6.00	14.00	9.00	0.72	0.98	Universal	True	True
158	11.00	8.00	13.00	6.00	9.00	14.00	14.00	10.00	8.00	7.00	0.57	0.97	ApproximateEntropy	True	True
185	8.00	6.00	13.00	16.00	8.00	6.00	11.00	10.00	8.00	14.00	0.30	1.00	Serial	True	True
186	9.00	10.00	13.00	7.00	6.00	8.00	12.00	7.00	14.00	14.00	0.49	0.99	Serial	True	True
187	8.00	4.00	12.00	9.00	11.00	8.00	15.00	12.00	10.00	11.00	0.53	0.99	LinearComplexity	True	True
188	10.06	9.99	9.90	10.34	9.95	10.08	9.93	9.72	10.01	10.01	0.53	0.99	NonOverlappingTe...	False	False

Figure 4.7: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Secure Hash Generator.*

Come si nota dal file riportato, il G-SHA1 passa 186 test su 188, ottenendo una percentuale di successo pari al 98,93%. In particolare, non vengono superati un test di NonOverlappingTemplate ed uno di RandomExcursions. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 184 test su 188, con un success rate pari al 97,87%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{96{,}831}{100}$.

4.8 Risultati BBSG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	8.00	10.00	11.00	9.00	11.00	10.00	6.00	11.00	12.00	12.00	0.96	1.00	Frequency	True	True
1	6.00	12.00	7.00	7.00	10.00	9.00	12.00	14.00	11.00	12.00	0.70	0.99	BlockFrequency	True	True
2	4.00	15.00	8.00	11.00	15.00	6.00	10.00	11.00	9.00	11.00	0.28	1.00	CumulativeSums	True	True
3	8.00	10.00	8.00	12.00	11.00	17.00	9.00	9.00	6.00	10.00	0.53	1.00	CumulativeSums	True	True
4	11.00	9.00	11.00	15.00	4.00	10.00	9.00	8.00	10.00	13.00	0.55	0.99	Runs	True	True
5	16.00	10.00	9.00	14.00	10.00	11.00	9.00	7.00	6.00	8.00	0.49	0.98	LongestRun	True	True
6	13.00	8.00	6.00	6.00	14.00	8.00	12.00	10.00	13.00	10.00	0.55	0.98	Rank	True	True
7	7.00	11.00	9.00	9.00	8.00	14.00	9.00	10.00	13.00	10.00	0.90	0.99	FFT	True	True
34	4.89	6.56	5.44	5.94	7.67	6.67	6.22	7.06	6.33	6.22	0.46	1.00	RandomExcursions...	True	True
41	6.38	4.88	6.62	5.50	8.38	5.75	4.50	7.50	6.50	7.00	0.24	0.99	RandomExcursions...	True	True
156	14.00	4.00	13.00	14.00	11.00	11.00	10.00	7.00	8.00	8.00	0.38	0.98	OverlappingTemplate	True	True
157	7.00	12.00	13.00	13.00	13.00	9.00	7.00	11.00	6.00	9.00	0.66	0.99	Universal	True	True
158	12.00	12.00	5.00	12.00	9.00	11.00	8.00	15.00	11.00	5.00	0.40	1.00	ApproximateEntropy	True	True
185	9.00	12.00	9.00	10.00	12.00	4.00	11.00	14.00	10.00	9.00	0.70	0.99	Serial	True	True
186	5.00	8.00	7.00	9.00	19.00	12.00	9.00	13.00	13.00	5.00	0.05	0.99	Serial	True	True
187	13.00	7.00	11.00	9.00	13.00	7.00	9.00	9.00	11.00	11.00	0.90	1.00	LinearComplexity	True	True
188	10.20	10.14	10.26	9.84	9.91	9.72	10.09	10.07	9.78	10.00	0.49	0.99	NonOverlappingTe...	False	False

Figure 4.8: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Blum Blum Shub Generator.*

Come si nota dal file riportato, il BBSG passa 187 test su 188, ottenendo una percentuale di successo pari al 99,47%. In particolare, non viene superato un NonOverlappingTemplate Tests. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 187 test su 188, con un success rate pari al 99,47%. Possiamo attribuire quindi a questo generatore un punteggio $Q = \frac{98{,}938}{100}$.

4.9 Risultati MSG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	9.00	7.00	11.00	9.00	7.00	11.00	14.00	10.00	13.00	9.00	0.85	1.00	Frequency	True	True
1	14.00	9.00	10.00	8.00	11.00	10.00	9.00	12.00	7.00	10.00	0.94	1.00	BlockFrequency	True	True
2	9.00	8.00	4.00	13.00	12.00	9.00	11.00	17.00	9.00	8.00	0.28	0.99	CumulativeSums	True	True
3	7.00	10.00	9.00	8.00	10.00	9.00	11.00	10.00	10.00	16.00	0.82	1.00	CumulativeSums	True	True
4	10.00	6.00	6.00	14.00	13.00	11.00	8.00	9.00	12.00	11.00	0.66	0.99	Runs	True	True
5	9.00	12.00	12.00	14.00	9.00	7.00	4.00	15.00	11.00	7.00	0.30	0.98	LongestRun	True	True
6	12.00	8.00	8.00	11.00	10.00	10.00	11.00	10.00	10.00	10.00	1.00	0.98	Rank	True	True
7	7.00	6.00	10.00	11.00	12.00	15.00	7.00	11.00	10.00	11.00	0.68	1.00	FFT	True	True
34	4.44	5.78	6.17	5.89	7.28	6.78	5.89	7.17	6.33	7.28	0.47	0.98	RandomExcursions...	True	False
41	7.12	6.38	6.50	6.50	5.62	5.00	6.25	7.00	5.88	6.75	0.50	0.99	RandomExcursions...	False	True
156	15.00	10.00	9.00	11.00	5.00	12.00	12.00	11.00	11.00	4.00	0.37	0.98	OverlappingTemplate	True	True
157	12.00	8.00	11.00	5.00	10.00	11.00	15.00	8.00	8.00	12.00	0.62	0.98	Universal	True	True
158	11.00	11.00	11.00	5.00	8.00	15.00	11.00	7.00	14.00	7.00	0.42	0.97	ApproximateEntropy	True	True
185	12.00	13.00	7.00	10.00	8.00	18.00	6.00	7.00	12.00	7.00	0.17	0.99	Serial	True	True
186	14.00	14.00	15.00	11.00	6.00	8.00	2.00	8.00	16.00	6.00	0.02	0.97	Serial	True	True
187	14.00	5.00	5.00	13.00	10.00	7.00	19.00	10.00	6.00	11.00	0.03	0.97	LinearComplexity	True	True
188	10.01	9.97	9.89	10.11	10.25	10.05	10.09	9.73	10.03	9.86	0.49	0.99	NonOverlappingTe...	False	False

Figure 4.9: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Blum Micali Schnorr Generator.*

Come si nota dal file riportato, il MSG passa 185 test su 188, ottenendo una percentuale di successo pari al 98,40%. In particolare, non vengono superati: 2 NonOverlappingTemplate Tests ed un RandomExcursions Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 185 test su 188, con un success rate pari al 98,40%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{96{,}834}{100}$.

4.10 Risultati PCG

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	14.00	8.00	15.00	10.00	11.00	12.00	10.00	4.00	15.00	1.00	0.02	0.99	Frequency	True	True
1	6.00	13.00	13.00	13.00	8.00	11.00	11.00	12.00	8.00	5.00	0.51	0.99	BlockFrequency	True	True
2	11.00	13.00	9.00	12.00	10.00	16.00	9.00	5.00	7.00	8.00	0.44	0.99	CumulativeSums	True	True
3	15.00	8.00	11.00	9.00	11.00	6.00	15.00	8.00	10.00	7.00	0.47	1.00	CumulativeSums	True	True
4	9.00	7.00	12.00	6.00	14.00	10.00	7.00	9.00	10.00	16.00	0.42	0.99	Runs	True	True
5	10.00	9.00	9.00	9.00	10.00	10.00	10.00	14.00	9.00	10.00	0.99	0.99	LongestRun	True	True
6	8.00	12.00	15.00	16.00	15.00	9.00	7.00	7.00	9.00	2.00	0.04	1.00	Rank	True	True
7	8.00	12.00	10.00	18.00	9.00	7.00	6.00	14.00	13.00	3.00	0.05	0.99	FFT	True	True
34	5.67	6.06	6.22	5.94	5.78	6.78	6.22	6.94	6.06	5.33	0.44	0.99	RandomExcursions...	True	True
41	4.88	6.62	6.00	5.50	6.75	6.25	6.25	6.00	6.12	6.62	0.52	0.99	RandomExcursions...	True	True
156	13.00	10.00	7.00	11.00	17.00	8.00	5.00	11.00	6.00	12.00	0.22	1.00	OverlappingTemplate	True	True
157	10.00	4.00	13.00	9.00	10.00	8.00	8.00	10.00	13.00	15.00	0.46	0.99	Universal	True	True
158	8.00	13.00	10.00	11.00	11.00	6.00	11.00	13.00	10.00	7.00	0.83	1.00	ApproximateEntropy	True	True
185	6.00	11.00	6.00	8.00	8.00	9.00	13.00	14.00	15.00	10.00	0.42	1.00	Serial	True	True
186	7.00	7.00	9.00	5.00	8.00	10.00	15.00	14.00	13.00	12.00	0.33	0.99	Serial	True	True
187	10.00	13.00	9.00	8.00	10.00	10.00	13.00	6.00	11.00	10.00	0.91	0.99	LinearComplexity	True	True
188	10.02	9.89	9.88	10.16	9.72	9.96	10.27	9.91	10.18	10.01	0.43	0.99	NonOverlappingTe...	False	False

Figure 4.10: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Permuted Congruential Generator.*

Come si nota dal file riportato, il PCG passa 187 test su 188, ottenendo una percentuale di successo pari al 99,47%. In particolare, non viene superato un NonOverlappingTemplate Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 187 test su 188, con un success rate pari al 99,47%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{98{,}938}{100}$.

4.11 Risultati METW

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	8.00	13.00	11.00	7.00	6.00	10.00	11.00	9.00	13.00	12.00	0.80	1.00	Frequency	True	True
1	6.00	4.00	11.00	7.00	14.00	12.00	10.00	14.00	7.00	15.00	0.15	1.00	BlockFrequency	True	True
2	7.00	9.00	11.00	10.00	7.00	14.00	10.00	10.00	14.00	8.00	0.78	1.00	CumulativeSums	True	True
3	15.00	7.00	7.00	10.00	5.00	10.00	12.00	14.00	9.00	11.00	0.44	0.99	CumulativeSums	True	True
4	12.00	15.00	15.00	7.00	8.00	11.00	6.00	11.00	10.00	5.00	0.28	0.99	Runs	True	True
5	16.00	9.00	3.00	13.00	11.00	8.00	10.00	4.00	14.00	12.00	0.08	0.99	LongestRun	True	True
6	5.00	15.00	12.00	5.00	13.00	9.00	10.00	14.00	10.00	7.00	0.25	1.00	Rank	True	True
7	14.00	5.00	11.00	9.00	13.00	8.00	12.00	11.00	9.00	8.00	0.68	1.00	FFT	True	True
34	7.56	9.28	6.89	6.50	6.50	6.44	7.17	7.78	6.11	5.78	0.45	0.99	RandomExcursions...	True	True
41	7.12	7.12	6.75	9.00	7.38	6.88	6.38	7.62	5.62	6.12	0.57	0.98	RandomExcursions...	True	True
156	15.00	11.00	11.00	15.00	6.00	7.00	6.00	5.00	12.00	12.00	0.18	1.00	OverlappingTemplate	True	True
157	11.00	11.00	9.00	12.00	8.00	16.00	10.00	6.00	12.00	5.00	0.42	0.97	Universal	True	True
158	9.00	7.00	8.00	11.00	18.00	9.00	8.00	10.00	8.00	12.00	0.42	1.00	ApproximateEntropy	True	True
185	12.00	7.00	7.00	12.00	9.00	10.00	14.00	8.00	10.00	11.00	0.85	0.98	Serial	True	True
186	12.00	14.00	15.00	8.00	7.00	10.00	4.00	7.00	10.00	13.00	0.26	1.00	Serial	True	True
187	11.00	12.00	9.00	9.00	10.00	11.00	13.00	7.00	11.00	7.00	0.94	1.00	LinearComplexity	True	True
188	9.88	9.51	10.08	9.93	10.01	10.01	10.11	10.01	10.18	10.28	0.46	0.99	NonOverlappingTe...	False	False

Figure 4.11: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi prodotte dal Mersenne-Twister Generator.*

Come si nota dal file riportato, il METW passa 187 test su 188, ottenendo una percentuale di successo pari al 99,47%. In particolare, non viene superato un NonOverlappingTemplate Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 186 test su 188, con un success rate pari al 98,94%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{98{,}410}{100}$.

4.12 Risultati Pi-greco-Gen

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	10.00	9.00	10.00	13.00	10.00	7.00	11.00	15.00	9.00	6.00	0.72	0.99	Frequency	True	True
1	10.00	13.00	11.00	10.00	6.00	12.00	15.00	10.00	8.00	5.00	0.49	0.98	BlockFrequency	True	True
2	9.00	9.00	8.00	11.00	15.00	7.00	12.00	6.00	13.00	10.00	0.64	0.99	CumulativeSums	True	True
3	9.00	8.00	8.00	11.00	17.00	9.00	11.00	9.00	9.00	9.00	0.70	0.99	CumulativeSums	True	True
4	12.00	8.00	6.00	8.00	12.00	9.00	10.00	11.00	16.00	8.00	0.60	0.98	Runs	True	True
5	9.00	7.00	9.00	8.00	13.00	17.00	6.00	7.00	13.00	11.00	0.29	1.00	LongestRun	True	True
6	14.00	6.00	9.00	5.00	18.00	11.00	11.00	9.00	7.00	10.00	0.15	0.98	Rank	True	True
7	16.00	13.00	7.00	5.00	12.00	7.00	9.00	10.00	8.00	13.00	0.30	0.97	FFT	True	True
34	3.72	4.94	5.94	6.83	6.83	5.61	7.17	8.33	7.11	6.50	0.37	1.00	RandomExcursions...	True	False
41	4.88	6.00	5.75	5.88	5.62	8.00	7.50	6.62	6.00	6.75	0.39	1.00	RandomExcursions...	True	True
156	11.00	12.00	13.00	10.00	11.00	10.00	7.00	10.00	7.00	9.00	0.95	0.98	OverlappingTemplate	True	True
157	8.00	8.00	10.00	10.00	9.00	10.00	14.00	13.00	11.00	7.00	0.88	0.99	Universal	True	True
158	12.00	10.00	9.00	13.00	8.00	6.00	9.00	14.00	11.00	8.00	0.78	0.98	ApproximateEntropy	True	True
185	13.00	6.00	9.00	7.00	13.00	13.00	12.00	9.00	9.00	9.00	0.74	1.00	Serial	True	True
186	13.00	10.00	7.00	8.00	12.00	14.00	9.00	8.00	6.00	13.00	0.62	1.00	Serial	True	True
187	8.00	15.00	6.00	9.00	15.00	9.00	13.00	6.00	12.00	7.00	0.28	0.98	LinearComplexity	True	True
188	9.85	10.34	9.77	10.56	10.06	10.03	10.10	9.90	9.88	9.50	0.51	0.99	NonOverlappingTe...	True	False

Figure 4.12: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi basate sull'espansione binaria delle cifre decimali di Pi-greco.*

Come si nota dal file riportato, il Pi-greco-Gen passa 188 test su 188, ottenendo una percentuale di successo pari al 100%. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 182 test su 188, con un success rate pari al 96,81%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{96{,}809}{100}$.

4.13 Risultati Nepero-Gen

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	9.00	9.00	7.00	8.00	13.00	11.00	14.00	11.00	7.00	11.00	0.82	0.97	Frequency	True	True
1	9.00	8.00	7.00	15.00	4.00	16.00	13.00	9.00	8.00	11.00	0.18	1.00	BlockFrequency	True	True
2	10.00	7.00	4.00	5.00	12.00	10.00	10.00	12.00	14.00	16.00	0.16	0.97	CumulativeSums	True	True
3	9.00	10.00	3.00	4.00	14.00	13.00	18.00	11.00	10.00	8.00	0.04	0.98	CumulativeSums	True	True
4	10.00	13.00	9.00	13.00	12.00	9.00	10.00	2.00	17.00	5.00	0.06	0.99	Runs	True	True
5	13.00	10.00	5.00	8.00	13.00	10.00	9.00	8.00	10.00	14.00	0.66	1.00	LongestRun	True	True
6	7.00	9.00	11.00	13.00	8.00	4.00	12.00	17.00	11.00	8.00	0.22	0.99	Rank	True	True
7	12.00	16.00	12.00	5.00	6.00	14.00	6.00	12.00	8.00	9.00	0.18	0.99	FFT	True	True
34	6.72	6.39	7.72	7.67	8.44	7.33	7.17	6.50	6.11	5.94	0.47	0.98	RandomExcursions...	True	True
41	9.12	8.25	7.62	7.25	8.38	5.25	6.75	6.50	5.62	5.25	0.31	0.98	RandomExcursions...	True	True
156	9.00	9.00	10.00	12.00	12.00	11.00	7.00	11.00	10.00	9.00	0.99	0.99	OverlappingTemplate	True	True
157	11.00	13.00	10.00	7.00	8.00	9.00	8.00	11.00	12.00	11.00	0.95	0.98	Universal	True	True
158	14.00	12.00	11.00	10.00	8.00	11.00	6.00	9.00	10.00	9.00	0.88	0.96	ApproximateEntropy	False	True
185	10.00	13.00	12.00	12.00	7.00	14.00	11.00	9.00	7.00	5.00	0.55	0.99	Serial	True	True
186	11.00	12.00	10.00	10.00	14.00	3.00	8.00	12.00	11.00	9.00	0.53	0.97	Serial	True	True
187	14.00	6.00	10.00	7.00	11.00	9.00	11.00	11.00	11.00	10.00	0.87	0.98	LinearComplexity	True	True
188	9.72	9.46	9.92	9.97	9.54	10.09	10.23	10.74	10.16	10.18	0.46	0.99	NonOverlappingTe...	False	False

Figure 4.13: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi basate sull'espansione binaria delle cifre decimali del numero di Nepero.*

Come si nota dal file riportato, il Nepero-Gen passa 184 test su 188, ottenendo una percentuale di successo pari al 97,87%. In particolare, non vengono superati: 3 NonOverlappingTemplate Tests e l'ApproximateEntropy Test. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 186 test su 188, con un success rate pari al 98,94%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{96{,}831}{100}$.

4.14 Risultati Sqrt2-Gen

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	13.00	8.00	6.00	10.00	13.00	10.00	11.00	10.00	8.00	11.00	0.88	0.98	Frequency	True	True
1	12.00	14.00	14.00	6.00	10.00	8.00	9.00	7.00	12.00	8.00	0.60	1.00	BlockFrequency	True	True
2	10.00	12.00	9.00	11.00	10.00	11.00	8.00	9.00	6.00	14.00	0.88	0.99	CumulativeSums	True	True
3	11.00	10.00	5.00	10.00	12.00	12.00	9.00	10.00	10.00	11.00	0.94	0.98	CumulativeSums	True	True
4	8.00	15.00	5.00	10.00	8.00	10.00	7.00	17.00	9.00	11.00	0.22	0.98	Runs	True	True
5	8.00	9.00	7.00	15.00	10.00	8.00	8.00	11.00	8.00	16.00	0.46	1.00	LongestRun	True	True
6	9.00	10.00	6.00	8.00	17.00	10.00	9.00	11.00	13.00	7.00	0.44	1.00	Rank	True	True
7	9.00	12.00	12.00	12.00	7.00	12.00	12.00	6.00	6.00	12.00	0.68	1.00	FFT	True	True
34	6.72	7.22	6.11	6.06	7.61	6.67	4.67	7.22	6.56	6.17	0.44	0.99	RandomExcursions...	True	False
41	6.50	6.62	7.12	6.75	6.38	6.62	7.25	6.12	6.12	5.50	0.40	0.99	RandomExcursions...	True	True
156	6.00	17.00	7.00	7.00	16.00	7.00	9.00	9.00	11.00	11.00	0.15	0.99	OverlappingTemplate	True	True
157	16.00	8.00	12.00	9.00	7.00	13.00	8.00	10.00	10.00	7.00	0.57	0.98	Universal	True	True
158	13.00	9.00	9.00	9.00	5.00	15.00	7.00	11.00	11.00	11.00	0.60	0.98	ApproximateEntropy	True	True
185	11.00	9.00	15.00	7.00	8.00	13.00	11.00	9.00	10.00	7.00	0.74	1.00	Serial	True	True
186	9.00	9.00	11.00	13.00	6.00	17.00	14.00	7.00	4.00	10.00	0.13	0.98	Serial	True	True
187	8.00	12.00	19.00	10.00	13.00	8.00	5.00	10.00	4.00	11.00	0.06	1.00	LinearComplexity	True	True
188	10.48	10.24	9.95	9.88	10.08	9.98	10.11	9.60	9.44	10.24	0.50	0.99	NonOverlappingTe...	False	False

Figure 4.14: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi basate sull'espansione binaria delle cifre decimali del numero* $\sqrt{2}$.

Come si nota dal file riportato, il Sqrt2-Gen passa 185 test su 188, ottenendo una percentuale di successo pari al 98,40%. In particolare, non vengono superati 3 NonOverlappingTemplate Tests. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 185 test su 188, con un success rate pari al 98,40%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{96{,}834}{100}$.

4.15 Risultati Sqrt3-Gen

	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	PV	PROP	TEST	OK	PV-OK
0	5.00	15.00	7.00	13.00	11.00	10.00	8.00	10.00	12.00	9.00	0.55	1.00	Frequency	True	True
1	6.00	8.00	7.00	12.00	11.00	14.00	10.00	11.00	11.00	10.00	0.82	0.99	BlockFrequency	True	True
2	7.00	8.00	14.00	13.00	5.00	12.00	12.00	13.00	4.00	12.00	0.21	1.00	CumulativeSums	True	True
3	7.00	7.00	21.00	11.00	8.00	8.00	7.00	9.00	13.00	9.00	0.05	1.00	CumulativeSums	True	True
4	8.00	10.00	6.00	14.00	8.00	14.00	16.00	11.00	8.00	5.00	0.20	0.98	Runs	True	True
5	8.00	9.00	9.00	12.00	9.00	13.00	13.00	7.00	6.00	14.00	0.64	1.00	LongestRun	True	True
6	11.00	9.00	9.00	11.00	12.00	9.00	10.00	10.00	12.00	7.00	0.99	0.98	Rank	True	True
7	11.00	7.00	10.00	5.00	6.00	17.00	11.00	9.00	14.00	10.00	0.22	1.00	FFT	True	True
34	4.22	5.61	5.00	6.11	6.94	6.22	5.83	8.00	7.67	6.39	0.33	0.99	RandomExcursions...	True	False
41	5.12	5.12	5.25	6.12	5.25	5.75	7.50	6.75	6.62	8.50	0.46	0.99	RandomExcursions...	True	True
156	7.00	13.00	11.00	8.00	11.00	11.00	13.00	5.00	12.00	9.00	0.70	1.00	OverlappingTemplate	True	True
157	10.00	20.00	7.00	7.00	10.00	10.00	8.00	8.00	13.00	7.00	0.11	0.99	Universal	True	True
158	9.00	9.00	13.00	12.00	14.00	11.00	8.00	8.00	9.00	7.00	0.83	1.00	ApproximateEntropy	True	True
185	4.00	13.00	13.00	6.00	8.00	8.00	10.00	12.00	16.00	10.00	0.22	1.00	Serial	True	True
186	8.00	6.00	11.00	7.00	8.00	15.00	8.00	11.00	9.00	17.00	0.25	1.00	Serial	True	True
187	10.00	16.00	8.00	8.00	14.00	6.00	7.00	11.00	12.00	8.00	0.40	0.98	LinearComplexity	True	True
188	9.70	10.22	9.70	10.38	10.29	10.02	9.76	9.64	10.62	9.69	0.47	0.99	NonOverlappingTe...	False	True

Figure 4.15: Risultati parziali del file finalAnalysisReport.txt prodotto dalla suite NIST dopo aver processato 100 sequenze binarie di 10^6 *elementi basate sull'espansione binaria delle cifre decimali del numero* $\sqrt{3}$.

Come si nota dal file riportato, il Sqrt3-Gen passa 183 test su 188, ottenendo una percentuale di successo pari al 97,34%. In particolare, non vengono superati 5 NonOverlappingTemplate Tests. Inoltre, il test relativo alla distribuzione dei PValue (colonna PV) viene superato solo da 185 test su 188, con un success rate pari al 98,40%. Possiamo attribuire quindi a questo generatore un punteggio di $Q = \frac{95{,}787}{100}$.

Conclusioni e sfide future

TEST	PASSED	P-VALUE PASSED	Q
BBSG	99,47%	99,47%	**98,938**
PCG	99,47%	99,47%	**98,938**
LCG	98,93%	100%	**98,936**
METW	99,47%	98,94%	**98,410**
MSG	94,40%	94,40%	**96,834**
SQRT2	98,40%	98,40%	**96,834**
NEPERO	97,87%	98,94%	**96,831**
G-SHA1	98,93%	97,87%	**96,831**
PI-GRECO	100,00%	96,81%	**96,809**
SQRT3	97,34%	98,40%	**95,787**
MODEXPG	96,28%	96,28%	**92,692**
QCG-I	94,68%	97,34%	**92,163**
QCG-II	94,15%	96,81%	**91,114**
CCG	86,70%	85,11%	**73,789**
XORG	11,83%	13,83%	**1,766**

Figure 4.16: Statistiche complessive dei risultati dei test effettuati. Si riportano i dati in ordine decrescente, dal migliore generatore al peggiore. Si evidenzia con colore arancione i generatori da non considerarsi randomici; mentre, si evidenzia con un colore verde i generatori considerabili come randomici.

Il documento di specifiche della suite sviluppata dal NIST [11] dichiara che un generatore di numeri random può considerarsi tale se passa tutti i test della suite. Questo però è un evento altamente improbabile anche nel caso di un reale generatore di numeri casuali. Infatti, scegliendo, come nel nostro caso, un $\alpha = 0.01$ la probabilità che una sequenza passi tutti i 188 test è pari al $0.99^{188} = 0.15 = 15\%$. In realtà, i vari test della suite non sono del tutto indipendenti e questa probabilità è quindi leggermente più alta. Comunque, si è deciso di attenersi alle considerazioni riportate dal paper [15], in cui sono

stati analizzati i risultati della NSTS in corrispondenza di 100GB di dati prodotti da un vero generatore di numeri casuali. In particolare, secondo questo studio, sono da considerarsi come non random i generatori che falliscono 7 o più test. Pertanto, dai test da noi realizzati possiamo considerare come validi generatori di numeri pseudocasuali: il Linear Congruential Generator, il Secure Hash Algorithm Generator, il Blum Blum Shub Generator, il Micali Schnorr Generator, il Permuted Congruential Generator, il Mersenne Twister Generator ed i generatori basati su i numeri π, e, $\sqrt{2}$ ed $\sqrt{3}$. Nello specifico, secondo i risultati riportati, i 4 migliori generatori sono in sequenza: 1.BBSG, 2.PCG, 3.LCG e 4.METW.
Sebbene le notevoli proprietà statistiche dei generatori analizzati, la maggiorparte di questi non vengono considerati a tutti gli effetti dei Cryptographically Secure Pseudorandom Number Generators (CSPRNG). Infatti, per essere considerato Criptographically Secure un algoritmo deve produrre dei numeri assolutamente impredicibili e spesso questo viene realizzato mediante sistemi di raccolta di entropia da fonti esterne (es: movimento del muose, attività di rete, ecc...) e mediante l'utilizzo di funzioni hash riconosciute dalla comunità scientifica come sicure. In particolare, un CSPRNG deve essere in grado di resistere a potenziali attacchi da parte di terzi che conoscono sia i numeri fino a quel momento prodotti sia il funzionamento stesso dell'algoritmo. Per esempio, vengono considerati CSPRNG i seguenti algoritmi [16]: Fortuna, Yarrow, HMAC-DRBG, AES-CTR, ISAAC, WELL19937a e ChaCha20. L'implementazione di tali algoritmi è sicuramente molto più complessa di quella relativa agli algoritmi da noi analizzati ed inoltre i criteri di test vanno aldilà della sola analisi statistica delle sequenze prodotte. In particolare, per testare i CSPRNG ci si affida anche a valutazioni di performance, simulazioni Monte Carlo e test reali di utilizzo sul campo. Di fatto, sono considerati Criptographically Secure gli algoritmi oggi utilizzati che non sono ancora stati violati.

Ringraziamenti

Desidero ringraziare il mio relatore Luciano Margara, che mi ha permesso di intraprendere un percorso di approfondimento in questo campo.

Vorrei inoltre ringraziare i miei familiari e tutti i miei amici, la cui vicinanza mi ha sempre aiutato a dare il massimo di me stesso.

Simone
5 Ottombre 2023

Bibliography

[1] J Trein, Th As, Schwarzbacher, and B. Hoppe. The development of a binary pseudo random number generator with controllable output density. pages 13–14, 05 2023.

[2] Anghel Florin, Asandoaiei David, and Tabacaru Robert. A remark on the discrete fourier transform statistical test. Cryptology ePrint Archive, Paper 2022/066, 2022. `https://eprint.iacr.org/2022/066`.

[3] Anne Canteaut. Berlekamp-massey algorithm. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, page 80. Springer, 2011.

[4] Ueli M. Maurer. A universal statistical test for random bit generators. *J. Cryptol.*, 5(2):89–105, 1992.

[5] George S. Fishman and Louis R. Moore. Empirical testing of multiplicative congruential generators with modulus 231-1. In *WSC*, page 129. ACM, 1978.

[6] Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

[7] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Comput.*, 15(2):364–383, 1986.

[8] Silvio Micali and Claus-Peter Schnorr. Efficient, perfect random number generators. In *CRYPTO*, volume 403 of *Lecture Notes in Computer Science*, pages 173–198. Springer, 1988.

[9] Melissa E. O'Neill. Pcg: A family of simple fast space-efficient statistically good algorithms for random number generation. Technical Report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA, September 2014.

[10] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.

[11] NIST. MS Windows nist statistical test suite, 2010.

[12] Emil Simion and Paul Burciu. A note on the correlations between nist cryptographic statistical tests suite. Cryptology ePrint Archive, Paper 2019/551, 2019. https://eprint.iacr.org/2019/551.

[13] Dong Lihua, Zeng Yong, Ji Ligang, and Han Xucang. Study on the pass rate of nist sp800-22 statistical test suite. In *2014 Tenth International Conference on Computational Intelligence and Security*, pages 402–404, 2014.

[14] Rajendra Hegadi and Abhijit Prakash Patil. A statistical analysis on in-built pseudo random number generators using nist test suite. In *2020 5th International Conference on Computing, Communication and Security (ICCCS)*, pages 1–6, 2020.

[15] Marek Sýs, Zdenek Riha, Vashek Matyas, K. Marton, and Alin Suciu. On the interpretation of results from the nist statistical test suite. 18:18–32, 01 2015.

[16] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators, 2015-06-24 2015.

Appendice

Software di analisi

Di seguito si riporta il codice Python utilizzato per l'analisi del file "finalAnalysisReport.txt", prodotto dalla NIST test suite una volta terminati i test su un particolare PRNG. Nello specifico, il codice riportato prende in input il file .csv, ne manipola i dati per renderne una migliore rappresentazione sotto forma di tabella e poi salva i dati così manipolati sotto forma di tabella .png e dati .xlxs. Per la trasformazione dei file "finalAnalysisReport.txt" nel file "finalAnalysisReport.csv" si è utilizzato un semplice software di calcolo come LibreOfficeCalc.

```
import pandas as pd
import numpy as np
import dataframe_image as dfi
pd.set_option("max_colwidth", 20)
filename = "finalAnalysisReport"
data = pd.read_csv(filename+".csv", skipinitialspace = True)
data["PROPORTION"] = data["PROPORTION"].str.replace('*', '')
data["STATISTICAL TEST"] = data["STATISTICAL TEST"].str.replace('*',
    '')
data["STATISTICAL TEST"] = data["STATISTICAL TEST"].str.strip()
data.rename(columns = {'PROPORTION':'PROP', 'STATISTICAL TEST':
    'TEST'}, inplace = True)
threshold_tests = 96 #Soglia per l'accettazione dei test
threshold_variant = 53 #Soglia per l'accetazione dei test
    RandomExcursion
alpha_p_value = 0.01 #Soglia per l'accettazione della distribuzione
    dei P-Value
new_col = np.round(np.linspace(0.1,1,10),1)
new_col = np.concatenate((new_col, data.columns[-3:]))
data.columns = new_col
numeratori = [int(i) for i, j in data["PROP"].str.split('/')]
denumeratori = [int(j) for i, j in data["PROP"].str.split('/')]
```

```
frazioni = np.divide(numeratori, denumeratori)
data["PROP"] = frazioni
data["PASS"] = ((data["PROP"] > (threshold_tests/100)) &
    ((data["TEST"]!="RandomExcursionsVariant") |
    (data["TEST"]!="RandomExcursions"))) | ((data["PROP"] >
    (threshold_variant/62)) &
    ((data["TEST"]=="RandomExcursionsVariant") |
    (data["TEST"]=="RandomExcursions")))
data["P_VALUE_PASS"] = data["P-VALUE"] > alpha_p_value
new_row = data[data["TEST"]=="NonOverlappingTemplate"].mean()
data.loc[len(data)] = new_row
data.at[len(data)-1, "TEST"] = "NonOverlappingTemplate_mean"
data.at[len(data)-1, "PASS"] = data.at[len(data)-1, "PASS"]==1
data.at[len(data)-1, "P_VALUE_PASS"] = data.at[len(data)-1,
    "P_VALUE_PASS"]==1
data.drop(data[data["TEST"] == "NonOverlappingTemplate"].index,
    inplace = True)
new_row = data[data["TEST"]=="RandomExcursions"].mean()
data.loc[len(data)] = new_row
data.at[len(data)-1, "TEST"] = "RandomExcursions_mean"
data.at[len(data)-1, "PASS"] = data.at[len(data)-1, "PASS"]==1
data.at[len(data)-1, "P_VALUE_PASS"] = data.at[len(data)-1,
    "P_VALUE_PASS"]==1
data.drop(data[data["TEST"] == "RandomExcursions"].index, inplace =
    True)
new_row = data[data["TEST"]=="RandomExcursionsVariant"].mean()
data.loc[len(data)] = new_row
data.at[len(data)-1, "TEST"] = "RandomExcursionsVariant_mean"
data.at[len(data)-1, "PASS"] = data.at[len(data)-1, "PASS"]==1
data.at[len(data)-1, "P_VALUE_PASS"] = data.at[len(data)-1,
    "P_VALUE_PASS"]==1
data.drop(data[data["TEST"] == "RandomExcursionsVariant"].index,
    inplace = True)
data.rename(columns = {'PASS':'OK', 'P-VALUE':'PV',
    'P_VALUE_PASS':'PV-OK'}, inplace = True)
data=data.round(2).sort_index()
dfi.export(data,filename+".png", table_conversion = 'matplotlib')
data.to_excel(filename+".xlsx")
```

Code A.1: Software di analisi dati in Python per l'interpretazione del file finalAnalysisReport.txt