*Generatori pseudoslučajnih brojeva*  
Tehnička dokumentacija

Verzija <1.3>

Studentski tim: Tanja Bertalanić

Josip Koprivnjak

Leo Marušić

Vice Sladoljev

Nastavnik: prof. dr. sc. Marin Golub

**Sadržaj**

[1. Sažetak projekta 4](#_Toc187233639)

[2. Generatori pseudoslučajnih brojeva 4](#_Toc187233640)

[2.1 Linear congruential generator (LCG) 5](#_Toc187233641)

[2.1.1 Uvod i definicija 5](#_Toc187233642)

[2.1.2 Primjeri parametara , i 5](#_Toc187233643)

[2.1.3 Primjeri perioda 6](#_Toc187233644)

[2.2 Hash\_DRBG 7](#_Toc187233645)

[2.2.1 Uvod i definicija 7](#_Toc187233646)

[2.2.2 Proces generiranja 7](#_Toc187233647)

[2.2.3 Primjer rada algoritma 8](#_Toc187233648)

[2.3 Mersenne twister 9](#_Toc187233649)

[2.3.1 Uvod i definicija 9](#_Toc187233650)

[2.3.2 Rad algoritma 9](#_Toc187233651)

[2.4 Wichmann Hill generator 11](#_Toc187233652)

[2.4.1 Uvod i definicija 11](#_Toc187233653)

[2.4.2 Opis 11](#_Toc187233654)

[2.4.3 Implementacija 11](#_Toc187233655)

[2.5 Lagged fibonacci generator 12](#_Toc187233656)

[2.5.1 Uvod i definicija 12](#_Toc187233657)

[2.5.2 Primjer rada algoritma 12](#_Toc187233658)

[2.5.3 Prednosti i nedostatci 12](#_Toc187233659)

[2.6 Inversive congruential generator 13](#_Toc187233660)

[2.6.1 Osnovna formula 13](#_Toc187233661)

[2.6.2 Period 13](#_Toc187233662)

[2.6.3 Primjer rada 13](#_Toc187233663)

[2.7 Xorshift 14](#_Toc187233664)

[2.7.1 Uvod i definicija 14](#_Toc187233665)

[2.7.2 Rad algoritma 14](#_Toc187233666)

[2.7.3 Primjer rada algoritma 14](#_Toc187233667)

[2.8 Pravilo 30 pseudoslučajan generator brojeva 15](#_Toc187233668)

[2.8.1 Uvod i definicija 15](#_Toc187233669)

[2.8.2 Način upotrebe 15](#_Toc187233670)

[2.8.3 Primjer rada algoritma 15](#_Toc187233671)

[2.9 Rezultati generiranja pseudoslučajnih brojeva 17](#_Toc187233672)

[2.9.1 Histogrami generiranih brojeva 17](#_Toc187233673)

[2.9.2 Zaključak 23](#_Toc187233674)

[3. Testovi kvalitete pseudoslučajnih brojeva 24](#_Toc187233675)

[3.1 Wald-Wolfowitz runs test 24](#_Toc187233676)

[3.1.1 Uvod i definicija 24](#_Toc187233677)

[3.1.2 Primjeri 25](#_Toc187233678)

[3.2 Autocorrelation test 26](#_Toc187233679)

[3.2.1 Uvod i definicija 26](#_Toc187233680)

[3.2.2 Ljung-Box test 27](#_Toc187233681)

[3.3 Chi-square test 27](#_Toc187233682)

[3.3.1 Koraci izvođenja 27](#_Toc187233683)

[3.4 Test diskretnom fourierovom transformacijom 28](#_Toc187233684)

[3.4.1 Uvod i definicija 28](#_Toc187233685)

[3.4.2 Koraci izvođenja 28](#_Toc187233686)

[3.4.3 Zaključak 29](#_Toc187233687)

[4. Korisničko sučelje 29](#_Toc187233688)

[5. Testiranje generatora u programu 32](#_Toc187233689)

[5.1 LCG 32](#_Toc187233690)

[5.2 Hash DRBG 33](#_Toc187233691)

[5.3 Mersenne twister i Wichmann Hill generator 33](#_Toc187233692)

[5.4 Lagged fibonacci generator 33](#_Toc187233693)

[5.5 Inversive congruential generator 35](#_Toc187233694)

[5.6 Xorshift 37](#_Toc187233695)

[5.7 Rule 30 37](#_Toc187233696)

[6. Literatura 39](#_Toc187233697)

# Sažetak projekta

U ovom projektu, istražuje se kvaliteta pseudoslučajnih generiranih brojeva dobivenih različitim algoritmima. Analiziraju se njihova statistička svojstva i uspoređuje ih se s idealnim slučajnim brojevima. Cilj je identificirati algoritme koji generiraju brojeve s najvećom razinom slučajnosti i prikladnosti za primjenu u simulacijama, kriptografiji i drugim područjima.

# Generatori pseudoslučajnih brojeva

Autor: Tanja Bertalanić, Literatura: <https://en.wikipedia.org/wiki/Pseudorandom_number_generator>, <https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/>

Generatori pseudoslučajnih brojeva (PRNG, eng. Pseudo-Random Number Generators) su algoritmi koji generiraju niz brojeva koji imaju svojstva slučajnosti. Za razliku od stvarnih slučajnih brojeva (koji se generiraju pomoću fizičkih metoda, poput radioaktivnog raspada), pseudoslučajni brojevi proizlaze iz determinističkih matematičkih pravila i ovise o početnim uvjetima, poznatima kao seed. Pseudoslučajni brojevi su različiti od stvarnih slučajnih brojeva po tome što su generirani algoritmima, a ne slučajnim procesima. Generatori pseudoslučajnih brojeva ključni su u različitim simulacijama (poput Monte Carlo simulacije), računalnim igrama i kriptografiji.

Kvalitetni PRNG-evi moraju zadovoljiti stroge statističke kriterije kako bi njihovi izlazi bili dovoljno "nasumični" za predviđenu primjenu.

Neki od mogućih nedostataka PRNG-eva zbog kojih generirani brojevi ne zadovoljavaju statističke kriterije:

* kratki periodi za određeni seed
* neujednačenost distribucije za veliku količinu generiranih brojeva
* korelacija između uzastopnih vrijednosti
* loša prostorna distribucija

## Linear congruential generator (LCG)

Autor: Leo Marušić, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Linear_congruential_generator), [MATHEMATICS OF COMPUTATION Volume 68, Number 225, Pages 249–260](https://www.ams.org/journals/mcom/1999-68-225/S0025-5718-99-00996-5/S0025-5718-99-00996-5.pdf), [Numerical Recipes](https://en.wikipedia.org/wiki/Numerical_Recipes)

### Uvod i definicija

Linearni kongruencijalni generator (LCG) je algoritam koji daje niz pseudoslučajnih brojeva izračunatih s diskontinuiranom djelomičnom linearnom jednadžbom(engl. *discontinuous piecewise linear equation*).  
Djelomično linearna funkcija je funkcija definirana na (moguće neograničenom) intervalu realnih brojeva, tako da postoji zbirka intervala na svakom od kojih je funkcija afina funkcija.  
Afina funkcija je funkcija koja preslikava afini prostor na sebe, a pritom zadržava i dimenziju svih afinih podprostora (što znači da šalje točke u točke, pravce u pravce, ravnine u ravnine i tako dalje) i omjere duljina paralelnih segmenti linije.  
Linearni kongurencijalni generator jednostavan je za izvesti i lako se izvršava na strojevima koji imaju mogućnost “modulo” operacije, sama jednadžba LCG-a glasi:

Gdje je X sekvenca pseudoslučajnih brojeva, te:

* - modulo
* - množitelj
* - inkrement
* - početna vrijednost (engl. *seed*)

su cjelobrojne vrijednosti koje definiraju svojstva generatora.

Ovisno o vrijednostima parametra linearni kongruencijalni generator se naziva:

* Multiplikativni kongruencijalni generator (MCG), za
* Mješoviti kongruencijalni generator, za

Uz dani početni broj (engl. *seed*), linearni kongruencijalni generator će uvijek proizvoditi isti niz brojeva.

Linearni kongruencijalni generatori imaju period, što znači da će se slijed na kraju ponoviti. Duljina perioda ovisi o parametrima (, i ).  
Dok su linearni kongruencijalni generatori sposobni proizvesti pseudoslučajne brojeve koji mogu proći formalne testove slučajnosti, kvaliteta izlaza je izuzetno osjetljiva na izbor parametara i . Zbog svoje determinističke prirode, linearni kongruencijalni generatori nisu prikladni za kriptografske svrhe gdje je potrebna velika razina slučajnosti.

### Primjeri parametara , i

Neki od parametara koji se koriste u širokoj upotrebi su npr.:

* glibc (standardna biblioteka za programski jezik C)
  + , ,
* java.util.Random
  + , ,
* Apple CarbonLib
  + , ,
* [*Numerical Reicpes*](https://en.wikipedia.org/wiki/Numerical_Recipes)- parametri od Knuth i H. W. Lewisa
  + , ,

Linearni kongruencijalni generatori se koriste u mnogim tehnologijama zbog njihovih jednostavnosti, ne samo u gore navedenima. Iako mnogi od njih su u širokoj upotrebi, ne znači da su njihovi parametri dobri ako uzmemo u obzir kvalitetu slučajno generiranih brojeva.

Tablica 1.1: Primjeri dobrih parametara multiplikativnih kongruencijalnih generatora

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Gdje je , te jedinstveni multiplikativni inverz modulo , broj takav da vrijedi . Jednostavnije, ako se zamijeni s u jednadžbi, generira se ista sekvenca brojeva, ali u obrnutom redoslijedu. Što efektivno čini i ekvivalentnim jer generiraju istu strukturu.  
, i predstavljaju mjeru kvalitete generiranog skupa. , veći broj znači bolje.

### Primjeri perioda

Parametri:

* , s početnom vrijednosti 0
  + …
  + Sažeti niz: 0 1 6 7 4 5 2 3
* , s početnom vrijednosti 15
  + 1
  + Sažeti niz: 15 1 19 5 23 9 27 13 31 17 3 21 7 25 11 29

## Hash\_DRBG

Autor: Leo Marušić, Literatura: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>

### Uvod i definicija

Hash\_DRBG je kriptografski siguran generator pseudoslučajnih brojeva standardiziran od strane NIST-a u SP 800-90A. Koristi hash funkciju (poput SHA-256 ili SHA-512) za generiranje pseudoslučajnih bitova. Ista hash funkcija se treba koristiti u cijelom procesu generiranja pseudoslučajnog broja.

A diagram of a block diagram

Description automatically generatedVrijednosti V i C su kritične vrijednosti unutarnjeg stanja, o kojima ovisi sigurnost za ovaj DRBG mehanizam.

### Proces generiranja

Proces generiranja pseudoslučajnog broja započinje inicijalizacijom početnog stanja preko „Hash\_DRBG Instantiate Process“:

1. seed\_material = entropy\_input || nonce || personalization\_string
2. seed = Hash\_df (seed\_material, seedlen)
3. V = seed
4. C = Hash\_df ((0x00 || V), seedlen)
5. reseed\_counter = 1
6. Return (V, C, reseed\_counter)

Gdje „||“ označava konkatenaciju, a „Hash\_df“ derivacijsku funkciju koja hashira dani niz znakova i vraća traženi broj bitova.

Završetkom tog procesa se dobiva početno stanje (vrijednosti V, C i *reseed\_count*)

Dobivene vrijednosti daju se „Hashgen“ funkciji koja stvara novi hash, ako je odabrano da se želi generirati više brojeva, onda se treba ići u *reseed* process kojim se određuju nove vrijednosti V, C i *reseed\_count*. U nastavku se nalazi jednostavni prikaz rada generatora, gdje je većina internih procesa izostavljena.

Slika 1: Hash\_DRBG - NIST SP 800-90A str. 39.

### Primjer rada algoritma

Generator započinje odabirom početne vrijednosti (engl. *seed*), za ovaj primjer će se koristiti riječ „bok“, čija je *hash* vrijednost (SHA256): „393e806553e469ba810883d19de96e2fe118a0146beed80905a2eee10ccc7e3a“

Ta hash vrijednost koristiti će se kao vrijednost od V.

Vrijednost C se isto treba izračunati iz početne vrijednosti, no za svrhu primjera, uzima se fiksna konstanta: „c18fcb516e54792cd70a1ebaf6853202“

*Hashgen* funkcija koja će se koristiti, definirana je: Hashgen(V, *Counter*) = SHA256(V || 1), gdje „||“ označava konkatinaciju.

Početno stanje izgleda ovako:

* V = 393e806553e469ba810883d19de96e2fe118a0146beed80905a2eee10ccc7e3a
* C = c18fcb516e54792cd70a1ebaf6853202
* *Counter* = 1

Generira se prvih 128 bitova:

1. Hashgen(V, *1*) = SHA256(„393e806553e469ba810883d19de96e2fe118a0146beed80905a2eee10ccc7e3a“ || 1)
2. Iz koje se dobivaju bitovi: 0be67570244b193bf281874a4d0b569b
3. Ponovno se generira vrijednost V: V = SHA256(„393e806553e469ba810883d19de96e2fe118a0146beed80905a2eee10ccc7e3a“ || „c18fcb516e54792cd70a1ebaf6853202“ || 1) = 2ef92fb56c698d3e9bedf42565c229a71a9eb8b2e06518a01bd9e7056e4a9d19
4. Te se poveća *Counter: Counter* = 2

Generiranje sljedećih 128 bitova:

1. Hashgen(V, 2) = SHA256(„2ef92fb56c698d3e9bedf42565c229a71a9eb8b2e06518a01bd9e7056e4a9d19“ || 2)
2. Iz koje se dobivaju bitovi: ea2190e8b1530d7e1d9055177eabd47d
3. Ponovno se generira vrijednost V: V = SHA256(„2ef92fb56c698d3e9bedf42565c229a71a9eb8b2e06518a01bd9e7056e4a9d19“ || „c18fcb516e54792cd70a1ebaf6853202“ || 2) = 813698ba7ce47b9bc370d7940129a3e4dd58010a153173eb5ed61c00fb0ca665
4. Te se povećava *Counter: Counter* = 3

Proces se nastavlja sve do definiranog broja ponavljanja.

Ovo je pojednostavljena ilustracija generatora. Stvarni Hash\_DRBG uključuje specifičnije izračune i korake za izračunavanje vrijednosti V, C i ažuriranje stanja. Sigurnost generiranih bitova najviše se oslanja na tajnost i slučajnost početne vrijednosti, prikazani način generiranja brojeva nije kriptografski siguran i ne bi se trebao koristiti u stvarne svrhe.

## Mersenne twister

Autor: Tanja Bertalanić, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Mersenne_Twister), <http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/emt.html>, <http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/ARTICLES/mt.pdf>

### Uvod i definicija

Mersenne Twister je algoritam za generiranje pseudoslučajnih brojeva. Razvili su ga Makoto Matsumoto i Takuji Nishimura 1997. godine. Jedan je od najčešće korištenih generatora nasumičnih brojeva zbog svoje velike brzine, kvalitete generiranih brojeva i dugog perioda. Ime mu potječe od korištenja Mersenneovog prostog broja za određivanje duljine perioda. Mersenneov prosti broj je oblika za neki cijeli broj . Najčešće korištena verzija Mersenne twister algoritma je MT19937 koji se temelji na Mersenneovom prostom broju i koristi 32-bitnu duljinu riječi.

### Rad algoritma

Za generiranje pseudoslučajnih -bitnih cijelih brojeva u rasponu , Mersenne Twister koristi linearnu rekurziju nad binarnim poljem . Algoritam se temelji na twisted generalised feedback shift registru (twisted GFSR ili TGFSR) u "racionalnom normalnom obliku"(TGFSR(R)). Temeljna ideja je definirati niz putem jednostavne rekurzije i potom izračunati brojeve oblika ​, gdje je inverzna matrica zvana tempering matrica.

Algoritam je definiran sljedećim parametrima:

* w - duljina riječi (u bitovima),
* n - stupanj rekurzije,
* m - srednja riječ, pomak korišten u rekurziji pri definiranju niza ,
* r - točka razdvajanja bitova unutar riječi, odnosno broj bitova u doljnjoj masci,
* a - koeficijenti matrice u normalnom obliku,
* b, c - TGFSR(R) maske za tempering,
* s, t - TGFSR(R) pomaci za tempering,
* u, d, l - dodatni pomaci/maske za Mersenne Twister.

S odabirom vrijednosti gdje je Mersenneov prost broj.

#### Inicijalizacija

Stanje potrebno za implementaciju Mersenne twistera je niz od -bitnih vrijednosti. Niz inicijalizira se sljedećim izrazom:

* vrijednost za MT19937 je 1812433253,
* ⊕ predstavlja bitovnu XOR operaciju,
* predstavlja pomak udesno.

Niz je definiran rekurzijom:

gdje:

* simbol ∣ predstavlja spajanje, tj. konkatenaciju bitova (viši bitovi lijevo)
* ⊕ predstavlja bitovnu XOR operacija
* su viši bitovi od ​
* predstavljaju nižih bitova od ​.

#### Twist transformacija

Twist transformacija je definirana u racionalnom normalnom obliku kao:

gdje je jedinična matrica dimenzija . Pomnožimo li matricom , račun se svodi na:

#### Tempering transformacija

Na novo generirani broj još se primjenjuje i tempering transformacija koja se sastoji od sljedećih bitovnih operacija:

gdje označava sljedeću vrijednost iz niza, privremenu vrijednost, a konačni izlaz algoritma.

#### Koeficijenti za MT19937

## Wichmann Hill generator

Autor: Tanja Bertalanić, Literatura: <https://en.wikipedia.org/wiki/Wichmann%E2%80%93Hill>

### Uvod i definicija

Generator slučajnih brojeva Wichmann-Hill je algoritam za generiranje pseudoslučajnih brojeva kojeg su predstavili Brian Wichmann i David Hill 1982. godine. Ovaj generator kombinira tri linearna kongruencijalna generatora kako bi postigao dulji period i poboljšao statističke osobine u usporedbi s pojedinačnim generatorima.

### Opis

Algoritam koristi tri zasebna linearna kongruencijalna generatora s različitim modulima i multiplikatorima. Svaki generator proizvodi niz cijelih brojeva, koji se zatim kombiniraju kako bi se dobio konačni nasumični broj u intervalu (0,1).

Parametri:

* s1, s2, s3 - početne vrijednosti (seedovi), cijeli brojevi veći od nule
* m1 = 30269, m2 = 30307, m3 = 30323 - moduli
* a1 = 171, a2 = 172, a3 = 170 - multiplikatori

Koraci algoritma:

1. Ažuriraj vrijednosti sjemenki:
2. Izračunaj nasumični broj u intervalu (0,1):

### Implementacija

Pseudokod za generator glasi:

Inicijaliziraj s1, s2, s3 s početnim vrijednostima

Za svako generiranje nasumičnog broja:

s1 = (171 \* s1) mod 30269

s2 = (172 \* s2) mod 30307

s3 = (170 \* s3) mod 30323

u = (s1 / 30269.0 + s2 / 30307.0 + s3 / 30323.0) mod 1.0

Vrati u

## Lagged fibonacci generator

Autor: Josip Koprivnjak, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Lagged_Fibonacci_generator), <https://asecuritysite.com/encryption/fab>

### Uvod i definicija

Lagged Fibonacci generator (LFG) je metoda generiranja pseudoslučajnih brojeva koja se temelji na modifikaciji poznatog Fibonacci niza.

Fibonaccijev niz dan je izrazom:

Problem s korištenjem običnog Fibonaccijevog niza je što bi pseudoslučajni nizovi bili jako predvidivi. Zato se umjesto tog izraza, koristi sljedeći:

Umjesto da sljedeći u nizu računamo tako da zbrajamo posljednja dva, biramo brojeve j i k koji označavaju koje ćemo brojeve koristiti. Također, nije nužno koristiti operaciju množenja. Umjesto zvjezdice u izrazu možemo iskoristiti zbrajanje, oduzimanje, množenje ili XOR.

Na početku izvođenja, uz gore navedene parametre, generatoru se mora dati i seed. Seed su brojevi koji će se koristiti kao početni niz.

### Primjer rada algoritma

Uzimamo j= 3, k= 7 i seed 6421893:

6 4 [ 2] 1 8 9 [ 3] --> 5

4 2 [ 1] 8 9 3 [ 5] --> 6

2 1 [ 8] 9 3 5 [ 6] --> 4

1 8 [ 9] 3 5 6 [ 4] --> 3

8 9 [ 3] 5 6 4 [ 3] --> 6

9 3 [ 5] 6 4 3 [ 6] --> 1

3 5 [ 6] 4 3 6 [ 1] --> 7

5 6 [ 4] 3 6 1 [ 7] --> 1

6 4 [ 3] 6 1 7 [ 1] --> 4

4 3 [ 6] 1 7 1 [ 4] --> 0

### Prednosti i nedostatci

Prednosti generatora LFG su:

* Brzina: Generator LFG je općenito brži od ostalih generatora poput LCG.
* Dug period: Ako su vrijednosti j, k i m odabrane pažljivo, LFG može generirati dug niz nasumičnih brojeva bez ponavljanja.
* Jednostavna implementacija

Nedostaci:

* Ovisnost o početnim parametrima: Ako j, k, m i seed nisu dobro odabrani na početku, generator će generirati lošije nizove.
* Visoka autokorelacija: Iako je generator brz, njegova je loša strana što ne generira nizove koji su dovoljno nasumični i stoga se ne bi trebao koristiti u slučajevima gdje je sigurnost od velike važnosti.

## Inversive congruential generator

Autor: Josip Koprivnjak, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Inversive_congruential_generator)

### Osnovna formula

Generiranje pseudo-slučajnih brojeva u ICG-u definira se formulom:

Gdje su:

* - trenutno generirani broj,
* - sljedeći generirani broj,
* m - modul(cijeli broj),
* c - aditivna konstanta (cijeli broj),
* - multiplikativna inverzija ​ modulo m.

### Period

Nakon određenog broja koraka, algoritam će krenuti ponavljati brojeve. Najveći period ponavljanja je broj m. Kako bi algoritam generirao što više različitih brojeva prije nego što počne ispočetka, potrebo je odabrati što bolje parametre. Postoji nekoliko algoritama za odabir parametara, ali jedan od njih je odabrati brojeve takve da prsten polinoma

bude primitivan.

### Primjer rada

Za parametre m = 5, a = 2, c = 3 i seed = 1, algoritam daje niz: 1, 0, 3, 2, 4, 1, 0, 3, 2, 4, 1, 0...

## Xorshift

Autor: Vice Sladoljev, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Xorshift), <https://www.jstatsoft.org/article/view/v008i14/916>

### Uvod i definicija

Xorshift je jednostavan algoritam za generiranje pseudoslučajnih brojeva. Izumio ga je George Marsaglia 2003 godine. Jedan je od najbržih softverskih generatora pa se vrlo često koristi. Generiraju niz brojeva od početnih uvjeta koristeći logički pomak i operaciju isključivo ili. Sam generator ne može proći statističke testove pa se trebaju kombinirati s nekim ne linearnim funkcijama. Postoji perioda ponavljanja koja je povezana s duljinom riječi pa za generator s velikom periodom treba odvojiti dovoljno memorije.

Postoji 32, 64, 128 i 1024 bitni xorshift. Što je veći broj bita to je algoritam više slučajan. Verzije veće od 64 bita su sporije na današnjem hardveru.

Xorshift se danas koristi u simulacijama, igrama i ugrađenim sustavima gdje su resursi ograničeni i vrijeme bitno.

### Rad algoritma

Algoritam je vrlo jednostavan i učinkovit. Odabire se početno stanje i rade se operacije

Gdje je a b operacija logičkog pomaka broja a u lijevo za b mjesta, a b je ekvivalentna operacija za desno micanje, a a b je operacija logičkog isključivog ili.

### Primjer rada algoritma

Za 32 bitne brojeve, broj ponavljanja = 5:

početno stanje = 1

izlaz = 1082269761, 738268225, 1120282153, 2269446181, 1640309605

početno stanje = 123456

izlaz = 3745083780, 908452455, 5028696163, 8305958659, 4351679101

## Pravilo 30 pseudoslučajan generator brojeva

Autor: Vice Sladoljev, Literatura: <https://mathworld.wolfram.com/Rule30.html>

### Uvod i definicija

Pravilo 30 generatori su generatori koji koriste jednodimenzionalni stanični automat. Stanični automat je sličan Conwayevoj Igri života, ali u jednoj dimenziji, tj. pravilo stanja za svaku ćeliju je funkcija trenutnog stanja i stanja susjeda. Ćelije mogu biti predstavljene kao niz bitova. Generator je osmislio Stephen Wolfram, a koristi se u aplikaciji Wolfram Alpha, poznatoj aplikaciji za rješavanje matematičkih problema. Algoritam se često koristi jer je brz i relativno jednostavan za izradu. Također je zanimljiv zbog svoje povijesti i jer koristi zanimljive koncepte. Svoj naziv je dobio jer binarni broj 00011110 u dekadskom zapisu glasi 30. Pravilo se također može zapisati kao novo stanje = lijevi susjed XOR (staro stanje OR desni susjed).

Tablica 2.2: Pravila generatora Pravilo 30

|  |  |  |  |
| --- | --- | --- | --- |
| Lijevi susjed | Trenutno stanje | Desni susjed | Novo stanje |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 |

### Način upotrebe

Na početku automat treba inicijalizirati slučajnim seedom. Tada ga se pusti da se brojevi mijenjaju po zadanim pravilima (Tablica 2.2). Nakon svakog primjenjivanja pravila, uzme se broj u sredini kao slučajni bit. Ako je broj bitova paran, nije bitno uzima li se lijevi ili desni bit od sredine. Lista brojeva se vrti u krug (poslije zadnjeg elementa ide prvi) pa se tako rješava problem desnog susjeda zadnjeg elementa i lijevog susjeda prvog elementa.

### Primjer rada algoritma

Uzimamo broj ćelija 20.

Početno stanje = seed = 00000000011000000000

Stanje 1. 00000000100100000000

Stanje 2. 00000001111110000000

Stanje 3. 00000010000001000000

Stanje 4. 00000111000011100000

Stanje 5. 00001000100100010000

Stanje 6. 00011101111110111000

Stanje 7. 00100010000001000100

Stanje 8. 01110111000011101110

Stanje 9. 10001000100100010001

Stanje 10. 01011101111110111010

Stanje 11. 11100010000001000111

Stanje 12. 00010111000011101000

Stanje 13. 00111000100100011100

Stanje 14. 01000101111110100010

Stanje 15. 11101110000001110111

Stanje 16. 00010001000010001000

Stanje 17. 00111011100111011100

Stanje 18. 01000100011000100010

Stanje 19. 11101110100101110111

Stanje 20. 00010001111110001000

Izlaz iz generatora = 01000100010001000101

## Rezultati generiranja pseudoslučajnih brojeva

Za potrebe analize, generirane brojeve iz različitih generatora prikazat ćemo histogramima kako bismo vizualizirali njihovu učestalost. Histogram je stupčasti dijagram koji prikazuje podatke o učestalosti podataka. Pripadajuće grafove generirat ćemo pomoću Matplotlib biblioteke u jeziku Python.

Svakim generatorom ćemo generirati niz od 10000 brojeva te ćemo ih prikazati na grafu uz frekvenciju njihovog pojavljivanja. Parametre ćemo postaviti proizvoljno. Svi generatori implementirani su u jeziku Python.

### Histogrami generiranih brojeva

**Linear Congruential Generator (LCG)**

* Početna vrijednost (*seed*): 64
* Parametar a: 25907312
* Modul m: 33554393
* Normalizacija: Brojevi su skalirani u raspon [0,1].

A graph of blue lines

Description automatically generated

Slika 2: Histogram brojeva generiranih LCG-om

**Hash\_DRBG Generator**

Brojevi generirani ovim generatorom također su normalizirani u rasponu [0,1], a histogram prikazuje njihovu distribuciju.

A graph of blue lines

Description automatically generated

Slika 3: Histogram brojeva generiranih Hash\_DRBG

**Mersenne Twister**

* Početna vrijednost (*seed*): 5748
* Normalizacija: Brojevi su skalirani u raspon [0,1].

A graph of blue lines

Description automatically generated

Slika 4: Histogram brojeva generiranih Mersenne Twisterom

**Wichmann-Hill Generator**

* Početni parametri (*seeds*): 12345; 67890; 13579

A graph of blue lines

Description automatically generated

Slika 5: Histogram brojeva generiranih Wichmann-Hill generatorom

**Lagged Fibonacci Generator**

* Parametri: j=3, k=7, m=100
* Početna vrijednost (*seed*): 6421893
* Operacija: Zbrajanje (+).

A graph of blue lines

Description automatically generated

Slika 6: Histogram brojeva generiranih Lagged Fibonacci generatorom

**Inversive Congruential Generator**

* Parametri: a=2, c=3, m=101
* Početna vrijednost (*seed*): 50
* Normalizacija: Brojevi skalirani u raspon [0,1].

A graph of a number of blue bars

Description automatically generated with medium confidence

Slika 7: Histogram brojeva generiranih Inversive congruential generator

**Xorshift generator**

* Početna vrijednost (*seed*): 12345
* Iteracije: 100

A graph of blue lines

Description automatically generated

Slika 8: Histogram brojeva generiranih generatorom Xorshift

**Pravilo 30 generator**

* Početna vrijednost (*seed*): 12345
* Parametri: Blok veličine 32, izlaz s 32 bita

A graph of blue lines

Description automatically generated

Slika 9: Histogram brojeva generiranih generatorom Pravilo 30

### Zaključak

Dobri generatori slučajnih brojeva proizvode histogram s ravnomjernom raspodjelom visina stupaca odnosno svi intervali vrijednosti moraju imati približno jednaku frekvenciju, brojevi moraju biti ravnomjerno raspodjeljeni kroz cijeli definirani raspon te ne smiju biti vidljivi periodični obrazci ponašanja na grafu. Na temelju prikazanih histograma brojeva generiranih različitim generatorima slučajnih brojeva, možemo donijeti nekoliko zaključaka o distribuciji brojeva. LCG, Hash\_DRBG, Mersenne Twister, Xorshift, Pravilo 30 i Wichmann-Hill pokazuju gotovo ravnomjernu distribuciju brojeva u rasponu [0, 1]. To ukazuje na visoku kvalitetu generiranih brojeva. Lagged Fibonacci Generator ima blagu neravnomjernost u pojedinim segmentima, što moze ukazivati na slabosti u distribuciji. Inversive Congruential Generator pokazuje anomalije u distribuciji s izraženim vrhovima na određenim vrijednostima, što može ukazivati na problem u generiranju ili korištenim parametrima. Svi generatori osim Inversive Congruential Generatora pokazuju reznolikost distribucije koja odgovara očekivanjima za slučajne brojeve.

# Testovi kvalitete pseudoslučajnih brojeva

Autor: Tanja Bertalanić, Literaura: <https://en.wikipedia.org/wiki/Randomness_test>, <https://en.wikipedia.org/wiki/Statistical_randomness>

Testovi kvalitete pseudoslučajnih brojeva su statistički alati koji procjenjuju koliko niz generiranih brojeva zadovoljava svojstva slučajnosti. Cilj testova je otkriti obrasce ponašanja, ponavljanja ili korelacije koji bi ukazivali na neslučajnu prirodu brojeva.

Ključni koncepti:

* Slučajnost: Mjeri nepredvidljivost i odsutnost uzorka u generiranim brojevima.
* Distribucija: Generirani brojevi trebaju biti ravnomjerno raspoređeni u zadanom rasponu.
* Nezavisnost: Vrijednosti unutar niza ne smiju biti međusobno povezane.

Generatori pseudoslučajnih brojeva se koriste u mnogim osjetljivim područjima poput kriptografije, financijskih modela i simulacija. Neispravan generator može dovesti do predvidivih uzoraka, što može uzrokovati sigurnosne propuste ili netočne rezultate.

Ako generator prođe sve testove kvalitete, možemo smatrati da generirani niz brojeva dovoljno dobro simulira slučajnost za zadanu primjenu. Ako generator ne prođe određeni test, to može ukazivati na potrebu za prilagodbom parametara ili izborom drugačijeg algoritma.

## Wald-Wolfowitz runs test

Autor: Leo Marušić, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Wald%E2%80%93Wolfowitz_runs_test), [Runs Test for Detecting Non-randomness](https://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm) , [A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications](https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf)

### Uvod i definicija

Wald–Wolfowitzov test (engl. Wald–Wolfowitz runs test) je statistički test bez parametara koji provjera hipotezu o slučajnosti nad nizom podataka s dvije vrijednosti, odnosno, jesu li elementi niza međusobno neovisni. Nazvan je po statističarima Abrahamu Waldu i Jacobu Wolfowitzu.  
Test odgovara na pitanje “Jesu li ovi uzorci podataka generirani nasumičnim procesom?”.

Prvi korak u *runs* testu je brojanje broja izvoda u nizu podataka. Postoji nekoliko načina za definiranje izvoda ovisno o setu podataka nad kojim se provodi test. Najlakši primjer za to prikazati je bacanje novčića; gdje će sekvenca podataka izgledati npr.   
G G P G P G P P G G za 10 bacanja novčića (G - glava, P - pismo). Što se može isto prikazati kao niz ++-+-+--++, koji se sastoji od 7 izvoda duljina 2, 1, 1, 1, 1, 2, 2.

Test se temelji na nultoj hipotezi, odnosno, da je svaki element u nizu neovisno izvučen iz iste distribucije.

Pod nultom hipotezom, broj izvoda u nizu od N () elemenata je slučajna varijabla čija uvjetna distribucija, s obzirom na promatranje pozitivnih vrijednosti i negativnih vrijednosti, približno normalna s prosjekom i varijancom:

, te brojem izvoda

Očekivani broj izvoda, pretpostavljajući neovisnost događaja, računa se:

, što kada se raspišu, te pojednostavni se u:

Rezultat testa dobiva se uspoređivanjem dobivenog broja izvoda s očekivanim brojem izvoda. Ukoliko je dobiveni broj izvoda *mnogo* veći ili manji od očekivanog broja izvoda, hipoteza o nezavisnosti slučajnih događaja može biti odbačena.

Statistička značajnost kako bi se odredilo što je “mnogo veće” ili “mnogo manje”, izabire se prije prikupljanja podataka, i tipično iznosi oko 5%, što za Wald–Wolfowitzov test tipično nije slučaj, više o tome u primjerima.

Test se koristi za testiranje:

* Slučajnost distribucije - uzimanjem podataka po redu i označavanjem s “+” sve podatke veće od prosjeka, te s “-” sve podatke manje od prosjeka, ignorirajući podatke jednake prosjeku.
* Odgovara li funkcija dobro skupu podataka- podaci koji prelaze vrijednost funkcije, označuju se s “+”, dok ostali s “-”. U obzir se uzima samo je li podatak iznad ili ispod predviđene vrijednosti funkcije, ali ne i koliko je “udaljeno”.

### Primjeri

Bacanje novčića 10 puta:

* Dobivena sekvenca: GPGPGPGPGP
* Iz sekvence se može iščitati 10 izvoda
* Preko izraza se izračuna očekivani broj izvoda: 6
* Ovisno o postavljenoj statističkoj značajnosti, odbacuje se hipoteza o neovisnosti događaja

Nasumično generirani brojevi:

* Nasumično generirani brojevi između 1 i 10: 5, 2, 8, 9, 1, 6, 3, 7, 10, 4
* Izračuna se njihov prosjek:
* Sekvenca brojeva prevodi se u niz “+” i “-”, gdje se s “+” označavaju brojevi iznad prosjeka, te s “-”, brojevi ispod prosjeka: --++-+-++-
* Iz niza se može iščitati 7 izvoda
* Očekivani broj izvoda je:
* Pošto je broj 6 blizu broja 7, može se pretpostaviti da je generirana sekvenca slučajno generirana

Primjeri su prikazani na malim sekvencama, te njihovi rezultati se ne mogu uzeti kao jaki dokazi za slučajnost.

## Autocorrelation test

Autor: Tanja Bertalanić, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Autocorrelation), <https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4481.htm>

### Uvod i definicija

Autokorelacija je statistički koncept koji procjenjuje stupanj korelacije između vrijednosti varijable u različitim vremenskim točkama i temeljni je koncept u analizi vremenskih serija. Autokorelacija mjeri stupanj sličnosti između dane vremenske serije i njezine pomaknute verzije tijekom uzastopnih vremenskih perioda. Slično je računanju korelacije između dvije različite varijable, osim što u autokorelaciji računamo korelaciju između dvije različite verzije i iste vremenske serije.

Autokorelacija označava se s gdje k označava vremenski pomak ili broj intervala između opažanja. Izračunava se pomoću koeficijenta korelacije:

Za skup podataka vremenske serije, autokorelacija na pomaku k određuje se usporedbom vrijednosti varijable u trenutku s vrijednostima u trenutku .

* je kovarijanca
* je standardna devijacija
* je vrijednost varijable u trenutku

Pozitivna autokorelacija (ρ>0) ukazuje na tendenciju da vrijednosti u jednoj vremenskoj točki budu pozitivno povezane s vrijednostima u kasnijoj vremenskoj točki. Visoka autokorelacija na određenom pomaku sugerira snažnu linearnu vezu između trenutnih vrijednosti varijable i njezinih prošlih vrijednosti na tom pomaku.

Negativna autokorelacija (ρ<0) sugerira obrnuti odnos između vrijednosti u različitim vremenskim intervalima. Niska ili nulta autokorelacija ukazuje na odsutnost linearne ovisnosti između trenutnih i prošlih vrijednosti varijable na tom pomaku.

### Ljung-Box test

Ljung-Box test je statistički test koji se koristi za provjeru postojanja ukupne autokorelacije u vremenskoj seriji do određenog broja pomaka kk. Za razliku od testova koji ispituju autokorelaciju na pojedinačnim pomacima, Ljung-Box test istodobno provjerava postojanje autokorelacije na više pomaka, pružajući sveobuhvatan uvid u slučajnost serije.

Hipoteze testa:

* Nulta hipoteza : Nema autokorelacije u vremenskoj seriji do određenog reda k; podaci su nezavisni.
* Alternativna hipoteza : Postoji autokorelacija u vremenskoj seriji za barem jedan pomak do reda k.

Ljung-Boxova statistika izračunava se prema formuli:

gdje je:

* — ukupan broj opažanja u vremenskoj seriji,
* — maksimalni broj pomaka (lag) koji se testira,
* — procijenjena autokorelacija na pomaku .

Statistika prati (chi-kvadrat) raspodjelu s stupnjeva slobode.

Interpretacija rezultata:

* Visoka p-vrijednost (≥0.05): ne odbacuje se nulta hipoteza, nema statistički značajne autokorelacije u seriji do pomaka k, podaci mogu smatrati slučajnim u kontekstu testiranih pomaka
* Niska p-vrijednost (<0.05): odbacuje se nulta hipoteza, postoji značajna autokorelacija, podaci pokazuju obrazac koji nije slučajan

## Chi-square test

Autor: Josip Koprivnjak, Literatura: [Wikipedia](https://en.wikipedia.org/wiki/Chi-squared_test)

### Koraci izvođenja

1. Generiranje pseudo-nasumičnih brojeva  
   Prvo je potrebno generirati što veći niz pseudo-nasumičnih brojeva.
2. Definiranje intervala  
   Da bi se testirala ravnomjernost distribucije, opseg vrijednosti se dijeli na više intervala ili “binova”. Npr ako generiramo brojeve od 1 do 1000, možemo taj opseg podijeliti na 10 intervala, npr. 1-100, 101-200 itd. Pošto želimo da generator bude što nasumičniji, u svakom intervalu očekujemo otprilike jednak broj generiranih brojeva.
3. Brojanje stvarnih frekvencija  
   Brojimo koliko se puta pojavio broj u svakom od intervala.
4. Primjenjivanje chi-square izraza

Oi predstavlja stvarnu frekvencija u intervalu i  
Ei predstavlja očekivanu frekvencija u intervalu i. U prijašnjem primjeru svaki Ei za svaki interval će biti 100.  
k predstavlja broj intervala

1. Uspoređivanje s kritičnom vrijednošću   
   Izračunatu vrijednost uspoređujemo s kritičnom vrijednosti iz chi-square tablice za zadanu razinu značajnosti (npr. 5% ili 1%) i stupnjeve slobode k - 1.  
   Ako je izračunata vrijednost manja od kritične vrijednosti, ne možemo odbaciti hipotezu da su brojevi ravnomjerno raspodijeljeni, što ukazuje na to da generator vjerojatno dobro generira pseudo-nasumične brojeve.   
   Ako je izračunata vrijednost veća od kritične vrijednosti, odbacujemo hipotezu, što ukazuje na odstupanja od ravnomjerne distribucije.

## Test diskretnom fourierovom transformacijom

Autor: Vice Sladoljev, Literatura: [NIST (test 2.6)](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf)

### Uvod i definicija

Test diskretnom fourierovom transformacijom (DFT) je test koji napravi fourierovu transformaciju niza brojeva i tako ih prebaci iz vremenske u frekvencijsku domenu. Cilj toga je pronaći periodičnost niza. Da bi se pronašla periodičnost uzima se broj vrhove frekvencijske domene koja prelaze 95% svih frekvencija. Ako je broj prevelik generator nije uspio proći test.

### Koraci izvođenja

1. Stvori se određen broj slučajnih brojeva te se oni pretvore u binarni niz i spoje se zajedno.
2. Sve nule pretvorimo u -1
3. Napravimo DFT na nizu
4. Uzmemo magnitude gdje je prva polovica niza poslije DFT
5. Izračunamo što je 95%-tni vrh pod pretpostavkom da je niz nasumičan
6. Izračunamo

što je broj vrhova nižih od T ako je razdioba zaista nasumična

1. Nađemo K što je zapravo broj vrhova nižih od T
2. Izračunamo
3. Izračunamo

gdje je erfc funkcija pogreške

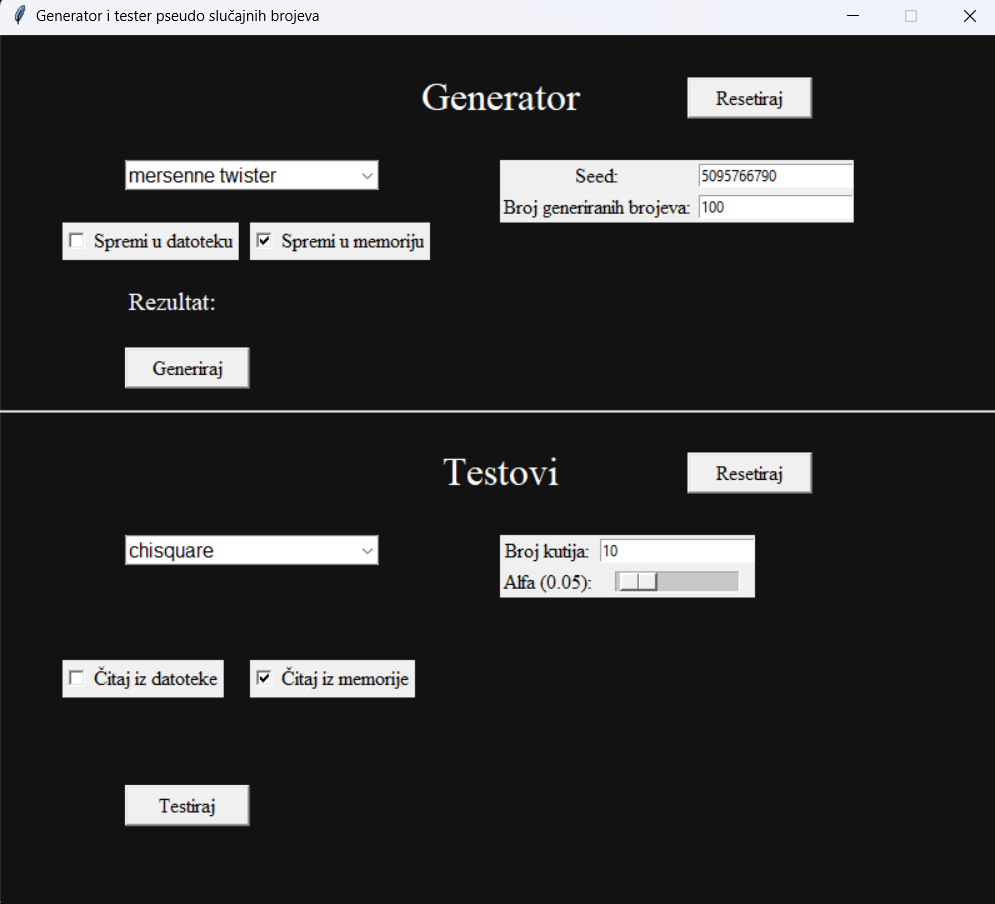
### Zaključak

Moramo odrediti koja nam granična p vrijednost. Standard je 1%, što znači ako je

p vrijednost < 0.01 generator nije slučajan. Preporuča se što veći broj bitova za testiranje, minimum je 1000.

# Korisničko sučelje

Aplikacija je implementirana koristeći Tkinter, biblioteku za grafička korisnička sučelja u Pythonu. Osmišljena je za upravljanje generatorima pseudoslučajnih brojeva i testiranje njihovih izlaza koristeći različite statističke testove. Glavni prozor aplikacije strukturiran je kako bi korisniku omogućio jednostavno odabiranje generatora, definiranje parametara, pokretanje generiranja brojeva, te njihovu evaluaciju putem odabranih testova.



Slika 10: Prikaz korisničkog sučelja

Za ispravan rad programa potrebno je:

1. Instalirati Python 3

2. U command promptu, u direktoriju projekta, potrebno je upisati naredbu: pip install -r requirements.txt

3. Pokrenuti main.pyw

Odabir generatora i testova omogućen je padajućim izbornicima. Svaki generator i test imaju parametre koji se mogu modificirati ili se pritiskom na gumb „Resetiraj“ postave na početno stanje. Pritiskom na tipku „Generiraj“ generiraju se brojevi te se pored teksta „Rezultat:“ ispiše samo zadnje generirani broj. Brojevi spremaju ili u datoteku ili u memoriju te se testovi mogu provoditi na brojevima iz datoteke ili iz memorije. Pritiskom na gumb „Testiraj“, na zaslonu se ispisuje zelena poruka „Generirani brojevi su nasumični po testu“ odnosno crvena poruka „Generirani brojevi nisu nasumični po testu“ ovisno o tome jesu li brojevi, tj. generator prošli statistički test.

A screenshot of a computer

Description automatically generated

Slika 11: Primjer korištenja aplikacije za generator Pravilo 30

A number on a white background

Description automatically generated

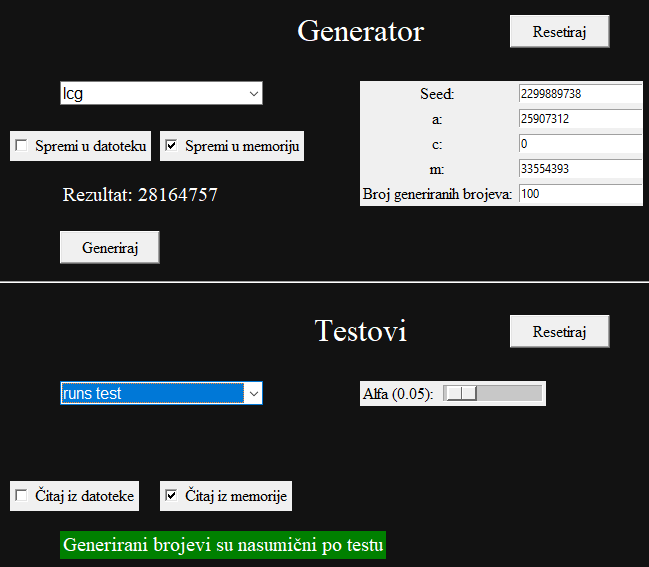
Slika 12: Sadržaj tekstualne datoteke „nasumični brojevi“

# Testiranje generatora u programu

## LCG

Generiranje nasumičnih brojeva LCG-om, koji prolaze razne testove, najviše ovisi o parametru *m*. Što je parametar *m* manji, to se generiraju lošiji i lošiji setovi podataka (u prosjeku). Parametar *a* isto igra važnu ulogu, ali ne na tolikoj razini kao i parametar *m.*



Ako uzmemo parametar *m* iz tablice 1.1, koja prikazuje dobro definirane MCG-ove, generirani brojevi će veliku većinu puta biti „nasumično“ generirani.  


## Hash DRBG

Hash DRBG generator koji je implementiran u programu, koristi SHA512 *hashing* algoritam, koji gotovo garantirano generira nasumične brojeve. Sama sigurnost i nasumičnost generatora upravo ovisi o odabiru *hashing* funkcije, manje sigurni algoritmi poput SHA1, MD5, MD4… generiraju „lošije“ rezultate.

Zbog korištenja *hashing* algoritma SHA512, čak i kod generiranje malog broja bitova i brojeva, dobiva se „nasumični“ set brojeva.



## Mersenne twister i Wichmann Hill generator

Generatori prolaze uglavnom sve statističke testove za bilo koje parametre

## Lagged fibonacci generator

Kako bi period bio maksimalan, polinom *y* = *xk* + *xj* + 1 mora biti primitivan.

Primjeri takvih parametara su:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| J | 7 | 5 | 24 | 65 | 128 | 6 | 31 |
| k | 10 | 17 | 55 | 71 | 159 | 31 | 63 |

Primjer koristeći parametre j=7 i k=10:

Screens screenshot of a computer

Description automatically generated

Primjer koristeći brojeve k=25, j = 26:

Screens screenshot of a computer

Description automatically generated

## Inversive congruential generator

Kako bi generator prošao kroz što više brojeva, potrebno je odabrati parametre tako da prsten polinoma bude primitivan.

Primjer dobrih parametara je: a = 926916007 i c = 297429857

Screens screenshot of a computer

Description automatically generated

## Xorshift

Xorshift je implementiran u programu je u 64 bitnoj verziji. Generator generira 64 bitne brojeve te za ulaz ima samo seed i broj brojeva koje želimo generirati. Za gotovo bilo koji seed i za relativno mali broj generiranih brojeva (10) generator prolazi testove sa alfom od 1%.

Slika na kojoj se prikazuje tekst, snimka zaslona, Font, softver

Opis je automatski generiran

## Rule 30

U generatoru rule 30 može se postaviti veličina jednog generiranog broja, te broj ćelija. Što je veći broj ćelija to će brojevi biti više slučajni, no generator prolazi razne testove i sa malim brojem ćelija. Zanimljivo je to što je najgori prolaz u chisquare testu što je dokazano Marco Tomassuni i Moshe Sipper.

Slika na kojoj se prikazuje tekst, snimka zaslona, softver, Font

Opis je automatski generiran

# Literatura

1. Pseudorandom Number Generator, Wikipedia (zadnje ažurirano 21.10.2024.), <https://en.wikipedia.org/wiki/Pseudorandom_number_generator>
2. Pseudo Random Number Generator (PRNG), Geeks for geeks, (zadnje ažurirano 30.12.2022.), <https://www.geeksforgeeks.org/pseudo-random-number-generator-prng/>
3. Linear Congruential generator, Wikipedia, (zadnje ažurirano 14.11.2024.), <https://en.wikipedia.org/wiki/Linear_congruential_generator>
4. L'Ecuyer, Pierre, Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure, Mathematics of Computation, (siječanj 1999.), <https://www.ams.org/journals/mcom/1999-68-225/S0025-5718-99-00996-5/S0025-5718-99-00996-5.pdf>
5. Numerical Recipes, Wikipedia, (zadnje ažurirano 23.7.2024.), <https://en.wikipedia.org/wiki/Numerical_Recipes>
6. Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST Special Publication 800-90A, (lipanj 2015.), <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>
7. Mersenne Twister, Wikipedia, (zadnje ažurirano 9.9.2024.), <https://en.wikipedia.org/wiki/Mersenne_Twister>
8. Mersenne Twister Home Page, <http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/emt.html>
9. Matsumoto, M.; Nishimura, T., Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator (1998.), <http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/ARTICLES/mt.pdf>
10. Wichmann-Hill, Wikipedia, (zadnje ažurirano 21.12.2022.), <https://en.wikipedia.org/wiki/Wichmann%E2%80%93Hill>
11. Lagged Fibonacci generator, Wikipedia, (zadnje ažurirano 8.5.2024.), <https://en.wikipedia.org/wiki/Lagged_Fibonacci_generator>
12. Lagged Fibonacci Generator, ASecuritySite, <https://asecuritysite.com/encryption/fab>
13. Inversive congruential generator, Wikipedia, (zadnje ažurirano 29.9.2024.), <https://en.wikipedia.org/wiki/Inversive_congruential_generator>
14. Xorshift, Wikipedia, (zadnje ažurirano 9.10.2024.), <https://en.wikipedia.org/wiki/Xorshift>
15. Marsaglia, George, Xorshift RNGs (srpanj 2003.), <https://www.jstatsoft.org/article/view/v008i14/916>
16. Randomness test, Wikipedia, (zadnje ažurirano 18.3.2024.), <https://en.wikipedia.org/wiki/Randomness_test>
17. Statistical randomness, Wikipedia, (zadnje ažurirano 15.8.2023.), <https://en.wikipedia.org/wiki/Statistical_randomness>
18. Wald-Wolfowitz runs test, Wikipedia, (zadnje ažurirano 6.4.2024.), <https://en.wikipedia.org/wiki/Wald%E2%80%93Wolfowitz_runs_test>
19. Runs Test for Detecting Non-randomness, NIST, <https://www.itl.nist.gov/div898/handbook/eda/section3/eda35d.htm>
20. A Statistical test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publication 800-22, (travanj 2010.), <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>
21. Autocorrelation, Wikipedia, (zadnje ažurirano 17.9.2024.), <https://en.wikipedia.org/wiki/Autocorrelation>
22. Box-Ljung Test, NIST, <https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4481.htm>
23. Chi-squared test, Wikipedia, (zadnje ažurirano 7.8.2024.), <https://en.wikipedia.org/wiki/Chi-squared_test>