Generatori pseudo-nasumičnih brojeva

Matijević, Blanka

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2022./2023.

Blanka Matijević

Generatori pseudo-nasumičnih brojeva

Završni rad

Mentor: doc. dr. sc. Ivan Dunđer

Zagreb, rujan 2023.

Izjava o akademskoj čestitosti

Izjavljujem i svojim potpisom potvrđujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

_____

(potpis)

Sadržaj

1. Uvod

Donald Knuth, otac analize algoritama, rekao je kako se slučajni brojevi ne bi trebali generirati slučajno izabranim postupkom (Urbiha, 2010). Potaknuta pitanjem što čini dobar generator nasumičnih brojeva, u ovom radu ću prikazati razne metode generiranja pseudo-nasumičnih brojeva. Sama nasumičnost potrebna je u raznim domenama, a to kreće od drevnih civilizacija koje su tražile božansko vodstvo kroz bacanje kocke pa sve do kriptografske sigurnosti. Međutim, postizanje prave nasumičnosti može biti izazovan zadatak u području računarstva, gdje determinizam dominira. Kako bi se premostio ovaj jaz, pseudo-slučajni brojevi i generatori pseudo-slučajnih brojeva (PRNG) pojavili su se kao moćni alati koji nude iluziju slučajnosti. Iako se rad bavi generatorima pseudo-slučajnih brojeva, ukratko će biti opisani i generatori istinski slučajnih brojeva (TRNG). TRNG generira brojeve stohastičkim procesima te se njihova izlazna vrijednost često koristi kao ulazna vrijednost u generatore pseudo-slučajnih brojeva. Pseudo-slučajni brojevi generiraju se deterministčkim algoritmima koji imaju za cilj oponašati svojstva istinski slučajnih brojeva. Iako ti brojevi mogu pokazivati statistička svojstva slična onima istinski slučajnih nizova, oni su zapravo potpuno predvidljivi ukoliko netko ima uvid u algoritam, parametre i slično. Rad istražuje inherentna ograničenja pseudo-slučajnih brojeva kao i izazove povezane s njihovom primjenom u različitim područjima. Valja napomenuti kako izrazi „slučajni“ te „nasumični“ (brojevi) označavaju isti fenomen.

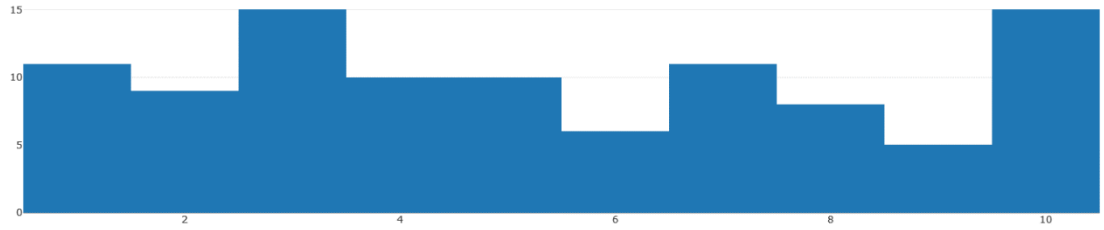
U radu će prije svega biti opisano što to čini niz brojeva slučajnim te kako su se slučajni brojevi generirali kroz povijest. Rad se ukratko osvrće na neke od prvih generatora kao što su ERNIE te John von Neumannov generator. Potom je objašnjena primjena slučajnih brojeva koja seže od simulacija i igrica do kriptografije. Nadalje, opisane su različite vrste PRNG-ova, uključujući linearne kongruentne generatore (LCG-ove), Lagged Fibonaccijeve generatore (LFG-ove) i Mersenne Twister, analizirajući njihove prednosti, slabosti i prikladnost za različite primjene. PRNG-ovi koriste matematičke formule i početnu vrijednost kao ulazne podatke za generiranje niza brojeva koji samo izgledaju statistički nasumično. Takva nasumičnost je dosta dobra za mnoge primjene. Detaljno su opisani LCG i LFG algoritmi što uključuje programsku implementaciju formula, testiranje rezultata te

opis ishoda mogućih parametara. Detaljnije je opisan i xorshift algoritam koji se često koristi u izradi generatora pseudo-nasumičnih brojeva. Dodatno, u radu su opisani i kriptografski sigurni generatori pseudo-slučajnih brojeva koji, za razliku od većine PRNG generatora, pružaju bitnu osnovu za kriptografske sustave. Kriptografija zahtijeva viši standard nasumičnosti, gdje predvidljivost može značiti lakše probijanje sustava. Opisani generatori uključuju Blum Blum Shub, CryptMT te RSA generator. RSA generator nije kriptografski siguran, ali je uključen jer počiva na DES-ovom algoritmu i dovoljno je siguran za mnoge primjene.

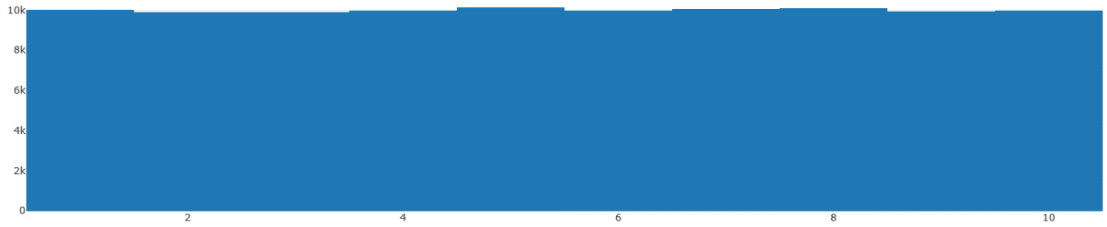
2. Slučajni brojevi

O nasumičnim brojevima govorimo u kontekstu niza brojeva. Ukoliko generatorom slučajnih brojeva dobijemo jedan broj, na primjer broj pet, ne možemo dokazati da je nasumičan. S druge strane na nizu brojeva možemo primijetiti određene karakteristike koje ukazuju na nasumičnost ili na manjak iste. Prva karakteristika slučajnih brojeva je nepredvidljivost te je vrlo intuitivna. U nizu X_0, X_1, X_2, X_3, …, X_n, ne možemo predvidjeti X_{n+1}. Praktičan primjer nepredvidljivosti je loto. Naime, možemo kontinuirano pratiti i zapisivati izvučene brojeve lota, ali na temelju tih podataka nikada nećemo moći predvidjeti koji će brojevi biti izvučeni u budućnosti (Urbiha, 2010). Slučajni događaji su nepredvidivi do te mjere da ne možemo raspoznati nikakav uzorak u nizu (Skočić, 2017). Drugi uvjet niza slučajnih brojeva je uniformiranost, što znači da se svi brojevi u nizu moraju podjednako često pojavljivati (Urbiha, 2010). Uniformiranost se odnosi na cijeli niz, ali i na sve podnizove. Kod podnizova generiranih brojeva bitno je naglasiti da ne smiju imati korelaciju, što znači da ne smiju međusobno ovisiti jedni o drugima (Jurić, 2001).

Provjera uniformnosti ne može se izvršiti na kratkim (manjim) nizovima nasumičnih brojeva (Urbiha, 2010). Navedeno je prikazano na grafikonu 2.1. gdje u generiranom nizu nije došlo do pojave uniformnosti. Os *x* predstavlja brojeve od jedan do deset, a os *y* količinu pojavljivanja u nizu. Isto se odnosi na grafikon 2.2. koji prikazuje raspodjelu pojave brojeva na mnogo većem nizu te je jasno vidljivo da su odstupanja u pojavljivanju pojedinih brojeva minimalna.



Grafički prikaz 2.1. Prikaz uniformnosti kratkog niza pseudo-nasumičnih brojeva



Grafički prikaz 2.2. Prikaz uniformnosti dugog niza pseudo-nasumičnih brojeva

Pseudo-nasumični brojevi iz grafikona (grafički prikazi 2.1. i 2.2.) generirani su u skriptnom jeziku JavaScript preko metode *random* biblioteke Math, a podatci su vizualizirani uz pomoć biblioteke Plotly. Grafikoni se generiraju nasumično pritiskom gumba „Generiraj graf“ na HTML stranici (programski kod 2.1.). Pritiskom gumba se ne generira novi grafikon nego se novi niz pseudo-nasumičnih brojeva nadodaje na već postojeći grafikon.

```
<body>
    <button id="graf">Generiraj graf</button>
    <div id="uni_graf"></div>

    <script>

        $(document).ready(function() {
            var array = [],
                length = 1e2;

            function nasumicni_brojevi() {
                for ( var i = 0; i < length; i++ ) {
                    array.push( Math.floor(Math.random() * 10) + 1 );
                }
            }

            function Graf() {

               nasumicni_brojevi();

               var density_data = {
                   x: array,
                   name: 'x density',
                   type: 'histogram'
               };

                Plotly.newPlot('uni_graf', [ density_data ] );
            }

            $("#graf").click(function() {
                Graf();
            });
        })

    </script>
</body>
```

Programski kod 2.1. Vizualizacija uniformnosti niza brojeva pomoću metode random

2.1. Povijest generatora slučajnih brojeva

Slučajni brojevi pojavljuju se isključivo u prirodnim situacijama čiji ishod nije moguće predvidjeti kao na primjer bacanjem simetričnog novčića (Jurić, 2001). Navedenu metodu su koristili i Rimljani te su dva moguća ishoda naglašavali izrazom „*navia aut caput*“ što znači brod ili glava prema motivima na svakoj strani novčića. Prije novčića ljudi su koristili kocke za nasumično odlučivanje, ali često nisu vjerovali u nasumičnost pa su tako rezultati bacanja kocke bili poruke od strane bogova. Kocke su bile popularne prije najmanje četiri tisuće godina na području današnje Kine, Indije i na području oko Mediteranskog mora, a najstarije kocke su pronađene na području današnjeg Iraka i Irana. Prve kocke nisu imale naznačene brojeve jer simboli za brojeve još nisu postojali. Ljudi su izumili generatore nasumičnih brojeva prije simbola za brojeve (L'Ecuyer, 2017).

Prvi „stroj“ za generiranje nasumičnih brojeva nije bio u potpunosti automatiziran. Kendall i Babington-Smith osmislili su stroj za koji je bila potrebna prethodno napravljena tablica od 100.000 nasumičnih brojeva. Brojevi su se nalazili na kartonskoj disketi te se ona okretala 250 puta u minuti uz oscilacije u brzini okretaja. Otprilike svake dvije sekunde bi svjetiljka osvijetlila nasumičan broj, a osoba bi zapisala taj broj na komad papira. Ovakva metoda je spora, ali moderniji generatori slučajnih brojeva uvelike su ubrzali rad na mnogim područjima ljudskog djelovanja. Prije njihove pojave statističari su nerijetko pisali znanstvene radove, čak i knjige, ispunjene gotovo isključivo nasumičnim brojevima. Engleski statističar Leonard Henry Caleb Tippett prvi je objavio takvu tablicu te je ona sadržala 41.600 brojeva (L'Ecuyer, 2017).

Pojavom računala nastala je težnja za računalnim generatorima nasumičnih brojeva. Ovdje se javlja problem jer su računala deterministički strojevi kod kojih ne postoje slučajni događaji. Iako slučajni brojevi nastali računalnim putem nisu „istinski“ slučajni, moguće je napraviti da ti brojevi izgledaju slučajno. Takvi brojevi nazivaju se pseudo-nasumičnim brojevima, a generatori su generatori pseudo-nasumičnih brojeva (Urbiha, 2010). Pionir generatora pseudo-nasumičnih brojeva bio je John von Neumann koji je 1946. godine izumio metodu srednjeg kvadrata. Ovaj jednostavan algoritam uzima početni uvjet, kvadrira ga, a srednje znamenke

kvadriranog broja su ujedno i slučajan broj i početni uvjet za sljedeću iteraciju (Skočić, 2017). Ukoliko je znamenka kvadriranog broja manje nego što je potrebno, dodaju se nule s lijeve strane zapisa (Urbiha, 2010). John von Neumann koristio je deseteroznamenkaste brojeve, ali ovdje će biti prikazan jednostavniji primjer. Broj 1111 kvadrira se te je rezultat 1234321 kojem se dodaje nula na početak te nastaje broj 01234321. Time je od 4-bitnog broja dobiven 8-bitni broj iz kojeg je cilj izvući novi 4-bitni broj. U ovom slučaju taj je broj 2343 i on se ponovno može kvadrirati te će se ponavljanjem postupka dobiti novi pseudo-slučajan broj. Kod ove metode postoji problem jer se sve sekvence nakon nekog vremena počnu ponavljati, a to se događa izrazito brzo kod sekvence 0000 (Skočić, 2017). Konkretan primjer takvog problema je: 62, 62^2 = 3844, 84^2 = 7056, 5^2 = 0025, 2^2 = 0004, 0^2 = 0000. Svaki sljedeći broj opet je nula (Urbiha, 2010). John von Neumann bio je svjestan tog problema, no ovaj generator je bio dosta dobar za njegove potrebe (Skočić, 2017).

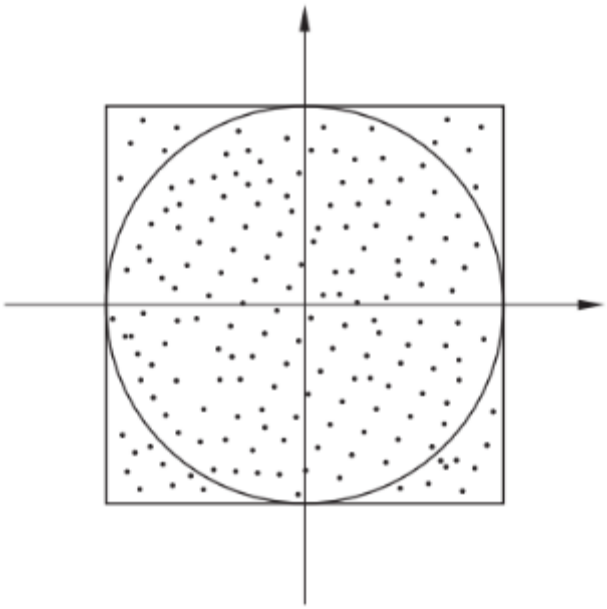
Strojevi koji generiraju „istinski“ nasumične brojeve nastavili su se razvijati paralelno sa pseudo-nasumičnim generatorima pa je 1957. godine po prvi puta korišten stroj ERNIE (*Electronic Random Number Indicator Equipment*). Nastao je za potrebe lutrije u Velikoj Britaniji jer se tada izvlačilo nekoliko tisuća dobitnih brojeva dnevno. Nasumični brojevi su nastajali od prikupljenog šuma nastalog spajanjem plina neona s elektronima. Proces se odvijao u staklenim cijevima gdje se nalazio neon te se na kraj svake cijevi dovodio visok napon. Taj stroj je korišten do 1972. godine, a od 2004. godine se koristi ERNIE 4 koji nasumične bitove izvlači iz toplinskog šuma u tranzistorima (L’Ecuyer, 2017).

2.2. Primjena slučajnih brojeva

Primjenu generatora nasumičnih brojeva možemo podijeliti u dvije osnovne grane: kriptografija i računalne simulacije. Kod obje kategorije poželjno je da generirani nizovi slučajnih bitova budu što sličniji “stvarnim” slučajnim brojevima. Generatori slučajnih bitova se u svojim svojstvima ne razlikuju od generatora slučajnih brojeva, a generator slučajnih brojeva lako može biti dobiven grupiranjem bitova iz generatora slučajnih bitova. Slučajni bitovi su posebno potrebni u kriptografske svrhe. Kriptografskim algoritmima i protokolima nasumični bitovi su

najčešće potrebni u svrhu generiranja javnog ključa, generiranja slučajnih bitova u postupku autentifikacije te za ključeve u digitalnom potpisu i pečatu. Druga primjena je u računalnim simulacijama gdje se varijablama u sustavu dodjeljuju nasumični brojevi koji prolaze kroz simulacije te se rezultati tih simulacija statistički obrađuju. Takva obrada može dovesti do zaključaka kao što su vjerojatnost pojave pojedinog događaja ili očekivana vrijednost neke varijable. Monte Carlo metoda primjer je računalne simulacije koja koristi nasumične brojeve kako bi riješila probleme koje nije moguće riješiti analitičkim putem (Jurić, 2001). Metoda se koristi od područja financija za predviđanje cijene dionica pa sve do fizike elementarnih čestica (Pease, 2018). Konkretan primjer vidljiv je u nuklearnoj fizici gdje se simulacije koriste jer se raspadanje radioaktivne tvari događa na slučajan način (Urbiha, 2010).

Nasumični brojevi koriste se za numeričke analize. Postoji kompleksan numerički problem koji se može riješiti približno točno uz pomoć nasumičnih brojeva. Zamislimo da ne znamo kako izračunati površinu kruga. Krug je potrebno postaviti u koordinatni sustav tako da mu je središte u ishodištu koordinatnog sustava. Krug se postavlja u sredinu kvadrata tako da su stranice kvadrata paralelne s koordinatnim osima kao što je vidljivo na slici 2.1. i zatim se uz pomoć dva nasumična broja postavljaju nasumične točke unutar kvadrata i kruga. Potrebna su dva broja kako bi se odredile koordinate točke. Neka je m točaka koje su pale unutar kruga, a n ukupan broj točaka u kvadratu. Velik broj ravnomjerno raspoređenih točaka daje omjer m/n koji je otprilike jednak omjeru površine kvadrata i kruga. Površina kvadrata jednostavnija je od površine kruga te je zato ona poznata u ovom primjeru. Stoga vrijedi formula $m/n \approx P_{kr}/P_{kv}$ koju je moguće proširiti u $P_{kr} \approx mP_{kv}/n = 4mr^2/n$. Sličan princip se koristi kao test za provjeru generatora nasumičnih brojeva. Poznata je površina kruga $P_{kr} = r^2\pi$ iz koje se može približno točno izračunati broj π preko formule $\pi \approx 4m/n$. Što je aproksimacija bliža stvarnom broju π, to je generator nasumičnih brojeva bolji (Urbiha, 2010).



Slika 2.1. Računanje površine kruga uz pomoć nasumično raspoređenih točaka (Urbiha, 2010)

Tokom godina primjena generatora nasumičnih brojeva se proširila i u mnoge druge sfere. Biranje reprezentativnog uzorka također može biti napravljeno uz pomoć nasumičnih brojeva. Primjer svojevrsnog uzorka građana su porotnici koje pravosudni sustav SAD-a odabire uz pomoć nasumičnih brojeva. Nadalje, u programiranju se slučajni brojevi mogu koristiti za testiranje programa ili događaja, ali se mogu koristiti i kao dio programa. Program koji igra šah može doći do situacija kada ima više jednako dobrih (ili loših) mogućih odabira. Program bi se smatrao lošijim da uvijek u istim situacijama odabire iste poteze ili da uvijek ima isto otvaranje pa je tu dobro implementirati nasumičnost. Šah manje ovisi o nasumičnim brojevima nego mnogi sadržaji namijenjeni zabavi. Računalne igrice, kockanje, miješanje karata koriste nasumične brojeve (Urbiha, 2010). Uzmimo računalne igrice u kojima korisnik igra nekog lika i cilj mu je prijeći sve razine igrice. Korisnik kod svakog ponovnog igranja neke razine može dobiti malo drugačiji doživljaj ukoliko se razina nasumično generira. To je moguće uz već pripremljeni set različitih razina pa nasumični brojevi služe samo za odabir gotove razine. Drugi način je da se svi elementi razine nasumično generiraju. Takav algoritam treba biti pažljivo napravljen kako ne bi došlo do nelogičnosti te se najčešće koristi kod generiranja krajolika, a rjeđe za generiranje soba, zgrada i sličnih objekata. Postoje i igrice koje su

napravljene uz pomoć takvih algoritama za nasumično generiranje razina, ali su autori postavili početni uvjet pa je svim korisnicima uvijek svaka razina identična (Anonymus, 2023).

Još jedna modernija tehnologija koja koristi nasumične brojeve je *blockchain*. *Blokchain* je vrsta baze podataka u kojoj se podatci nalaze u podatkovnim blokovima. Blokovi se nadovezuju jedan na drugi kriptografskim metodama, a kriptografska funkcija *hash* je bitan čimbenik u procesu. Naime, svaki *blockchain* entitet identificira se zasebnom adresom koja se sastoji od nasumično generiranih ključeva i *hash* funkcija koje služe kako bi bilo koju dužinu znakova pretvorili u fiksnu veličinu niza znakova. Time se *blockchain* osigurava od krivotvorenja što je vrlo bitno s obzirom da se koristi za osjetljive zadatke kao što su transakcije kriptovalutama. Zbog toga *blockchain* zahtijeva vrlo dobre generatore nasumičnih brojeva (Babić, 2019).

3. Generatori istinski slučajnih brojeva

Generatori istinski ili stvarno slučajnih brojeva (engl. *True random number generators* ili skraćeno TRNG) nasumične brojeve dobivaju iz fizičkih fenomena što znači da su nedeterministički i da nisu periodički (Skočić, 2017). Karakteristika istinski slučajnih brojeva je da u nizu slučajnih brojeva niti jedan broj ne ovisi o svome prethodniku. Takvi generatori su obično sklopovske naravi ili većim dijelom sklopovske naravi. Kod izvora temeljenih na sklopovlju često bude potrebna programska obrada dobivenih podataka pa se takvi generatori moraju povezati s računalom. Programska obrada je potrebna jer su brojevi često nebalansirani što znači da je u nizu češće javljanje pojedinih brojeva. Navedeno nije jedini problem kod izvora temeljenih na sklopova, naime oni su jako osjetljivi te su skloni kvarovima. Takvi strojevi su skupi, primjena im je ograničena, a generiranje brojeva je sporo pa i sporije od generatora pseudo-slučajnih brojeva. Primjeri izvora temeljenih na sklopovlju uključuju vrijeme između emisija čestica za vrijeme radioaktivnog raspada, šum u poluvodičkoj diodi, frekvencijsku nestabilnost oscilatora te zvuk iz mikrofona (Jurić, 2001). Postoji još mnogo primjera (atmosferski šum, termalni šum, kozmička radijacija i tako dalje), a zajednička karakteristika im je to što su iznimno dobri izvori prirodne entropije (Skočić, 2017).

Kod programski orijentiranih izvora prirodni procesi mogu biti iznimno jednostavni procesi nastali interakcijom korisnika i računala. Takvi izvori prirodne entropije nisu optimalni jer interakcije od strane korisnika slijede pravilnosti. Primjeri uključuju varijaciju u pokretima računalnog miša, vrijeme između pritiska dvije tipke na tipkovnici i sadržaj ulaznog ili izlaznog spremnika. Postoje i programski izvori za koje nije potreban korisnik kao što je generiranje brojeva uz pomoć specifičnih varijabli operacijskog sustava ili uz pomoć sistemskog sata, no njegovo ponašanje je također lako predvidivo. Dobar generator slučajnih brojeva dobiva se kombinacijom različitih načina generiranja (Jurić, 2001).

Konkretan primjer generatora stvarno slučajnih brojeva je Quantis. Razvila ga je švicarska tvrtka ID Quantique, a generator koristi kvantnu optiku kao izvor slučajnosti. To je moguće jer se svjetlost sastoji od elementarnih čestica koje nazivamo „fotoni“. Fotoni su dobar izvor nasumičnosti samo u određenim situacijama

kao što je refleksija. Quantis koristi polupropusno ogledalo na koje šalje fotone te iz toga dobiva nasumične brojeve (Babić, 2019). Još jedan primjer generatora istinski slučajnih brojeva je generator koji kao ulaznu vrijednost uzima fotografije iz digitalne kamere. Algoritam promatra intenzitet boje svakog piksela promatrajući cijeli *RGB* sustav boja te iz toga proizvodi nasumične vrijednosti. Nužno je da se dobivene vrijednosti kasnije dodatno ispremiješaju. Naime, fotografije često imaju uzorke koji bi proizveli loš niz nasumičnih bitova pa je potrebna naknadna programska obrada rezultata. Ova metoda generiranja brojeva je praktična te se može implementirati i kao aplikacija koju je moguće lako koristiti na mobitelu ili računalu (Li, 2015).

4. Generatori pseudo-slučajnih brojeva

Pseudo-nasumični nizovi brojeva statistički su slučajni, ali su nastali deterministčkim procesima. Iz navedenog se izvlači zaključak da takav niz brojeva izgleda kao slučajan niz, no zapravo nije slučajan (Skočić, 2017). Niz brojeva: *21, 28, 21, 19, 13, 14, 9, 24, 16, 27, 4, 1, 14, 19, 13, 19, 13, 29* izgleda u potpunosti slučajno, ali zapravo nije. Nije sasvim slučajan jer brojevi zapravo predstavljaju pozicije slova u hrvatskoj abecedi te je njihovo značenje „*ovonijeslučajniniz*“ (Petrović, 2018).

Bitno svojstvo generatora pseudo-nasumičnih brojeva (engl. *Pseudorandom number generators* ili skraćeno *PRNG*) je period generiranja, a to označava broj generiranih brojeva u nizu prije prvog ponavljanja. Naime, ovakvi generatori generiraju brojeve deterministčkim putem te prolaze kroz stanja, a ta stanja nisu beskonačna. Zbog toga će generator nakon nekog vremena naići na stanje u kojem je već bio te će doći do periodičkog ili cikličkog ponavljanja nekog podniza. Iz tog razloga je poželjno da period generiranja bude što veći. Nakon tog osnovnog svojstva je važna i brzina. Velika brzina je posebno poželjna kod simulacija jer one generiraju jako veliku količinu slučajnih brojeva. Potom je poželjna prenosivost koja uključuje različite strojeve, operacijske sustave i implementaciju u različitim programskim jezicima. Sve navedene karakteristike se očekuju uz minimalne memorijske zahtjeve. Iako je optimizacija generatora poželjna, ona ne smije smanjivati kvalitetu niza slučajnih brojeva (Jurić, 2001).

Funkcija koja daje niz pseudo-slučajnih brojeva radi to na temelju neke ulazne vrijednosti. Ulazna vrijednost ili početni uvjet (engl. *seed*) često je istinski slučajan broj dobiven iz generatora istinski slučajnih brojeva (Jurić, 2001). Početni uvjet se još naziva i ključ ili sjeme (Skočoć, 2017). Glavna karakteristika početnog uvjeta je da će uvijek davati isti niz pseudo-slučajnih brojeva te se svaki idući broj u nizu može predvidjeti. Zbog toga su ovakvi generatori deterministčke prirode, a brojevi nisu slučajni već pseudo-slučajni. Istraživanja su pokazala da su generatori pseudo-slučajnih brojeva ponekad uspješniji od generatora koji slučajne brojeve dobivaju iz fizičkih izvora. Za to je potreban dobro osmišljen generator i početni uvjet koji mogu dati niz brojeva koji će poprimiti karakteristike slučajnih brojeva te će se takav

generator smatrati vrlo dobrim. Nadalje, kombinacija različitih generatora će nerijetko dati zadovoljavajuće rezultate. Kombinirati se mogu istinski i pseudo-slučajni generatori brojeva, ali i razni pseudo-slučajni generatori. Primjeri osnovnih generatora slučajnih brojeva su LCG i LFG te se i oni mogu međusobno kombinirati, kao na primjer dva LCG generatora ili kombinacijom LCG i LFG generatora. Dobar odabir kombinacija generatora, u kombinaciji s dobrim izborom njihovih parametara, može značajno povećati period generiranja. Treba imati na umu da to dolazi uz moguće smanjenje brzine i povećanje memorijskih zahtjeva (Jurić, 2001).

4.1. Linearno kongruentni generator – LCG

Linearni kongruentni generator (engl. *Linear Congruential Generator* ili skraćeno LCG) dugo je bio najkorišteniji generator pseudo-slučajnih brojeva (Skočić, 2017). Generator je popularan zbog svoje jednostavnosti i jer je vrlo brz uz male memorijske zahtjeve. Brzinu postiže jer se svaki sljedeći broj računa prema njegovu prethodniku. Ipak, linearni kongruentni generator nije pogodan za veće pothvate kao što su računalne simulacije. Generirani brojevi dovoljno su nasumični za puno upotreba, ali niz pokazuje mnoge pravilnosti. Prvi broj u nizu se nerijetko dobiva iz generatora istinski slučajnih brojeva, a prvi broj je ujedno i početni uvjet. Valja naglasiti i kako je početni uvjet uvijek cijeli broj (Jurić, 2001). Izumio ga je američki matematičar Derrick Henry Lehmer 1948. godine (Babić, 2019). LCG se nerijetko naziva i linearni slijedni generator (Petrović, 2018).

Formula kojom LCG generira niz nasumičnih brojeva glasi:

$$X_{n+1} = (a \cdot X_n + c) \pmod{m}$$

gdje X_{n+1} predstavlja *n+1* slučajni cijeli broj, X_n je n-ti slučajni cijeli broj, *a* je koeficijent (multiplikator), *c* je konstanta (inkrement), *m* je gornja granica brojeva (modul za modulsku aritmetiku), a X_0 je početni uvjet ili sjeme. Početna vrijednost (X_0) koristi se za početak niza, a svaki sljedeći slučajni broj (X_{n+1}) generira se uz pomoć nasumičnog broja (X_n) tako što se množi s konstantom *a* te se na umnožak dodaje konstanta *c*. Na dobiven broj se primjenjuje modulo aritmetika uz pomoć parametra *m*. Proces se ponavlja kako bi nastao niz pseudo-nasumičnih brojeva

(Petrović, 2018). S obzirom da konstante *a*, *c* i *m* te početni uvjet određuju sve brojeve u generiranom nizu, LCG možemo definirati kao uređenu četvorku: *LCG(a, c, m, X_0)*. Izostavljena konstanta *m* služi za definiranje perioda generiranja, a to znači da period može biti jednak ili manji od konstante *m* (Jurić, 2001). Konkretan primjer može biti $m = 10$ i $X_0 = a = c = 7$, pa je potrebno te brojeve uvrstiti u formulu koja će sada glasiti:

$$X_{n+1} = (7 \cdot X_n + 7) \pmod{10}.$$

Iz toga je lako izračunati prvih nekoliko brojeva niza:

$$X_1 = 7 \cdot 7 + 7 = 56 \equiv 6 \pmod{10},$$

$$X_2 = 7 \cdot 6 + 7 = 49 \equiv 9 \pmod{10},$$

$$X_3 = 7 \cdot 9 + 7 = 70 \equiv 0 \pmod{10},$$

$$X_4 = 7 \cdot 0 + 7 \equiv 7 \pmod{10}.$$

Niz počinje brojevima *6*, *9*, *0* i *7*. Već na četvrtom broju se ponovno dolazi do broja *7* što znači da je ovo primjer niza čiji je period četiri. Navedeni period je iznimno malen za potrebe generatora pseudo-nasumičnih brojeva. Stoga je jasno da su parametri loše odabrani te da je takav generator loš (Urbiha, 2010).

LCG se može podijeliti u tri vrste prema odabiru parametara:

1. $m = 2^M$, $c > 0$

   U prvom slučaju konstanta *m* je potencija broja 2, konstanta *c* je pozitivni cijeli broj. Ovaj način pruža brz generator, a brzina proizlazi iz sklopovske podrške operacija brojeva koji su potencija broju dva. Puni period (2^M) je moguće postići ukoliko vrijedi $a = 1 \pmod 4$ te ukoliko je konstanta *c* neparna.

2. $m = 2^M$, $c = 0$

   Druga se vrsta razlikuje od prve po tome što joj je konstanta *c* jednaka nuli. Takav se generator naziva MCG (engl. *Multiplicative Congruential Generator*). Najveći mogući period je 2^{M-2} ili četvrtina modula *m*. Taj je period ostvariv ukoliko je konstanta $a = 3 \pmod 8$ ili $a = 5 \pmod 8$, a općenito se preferira druga opcija.

3. $m = p$ (prost broj)

Ako je konstanta *a* prost broj, moguće je postići najveći mogući period. Generalno ovim generatorom je moguće doći do maksimalnog perioda od *p – 1*. Preporuča se generator s parametrima: $m = 2^{31}-1$, $a = 16807$, $c = 0$ (Jurić, 2001).

Generalno vrijedi da je $n \geq 0,\ m > 0,\ 0 \leq a < m$ te $0 \leq c < m$ (Herrero-Collantes i Garcia-Escartin, 2017).

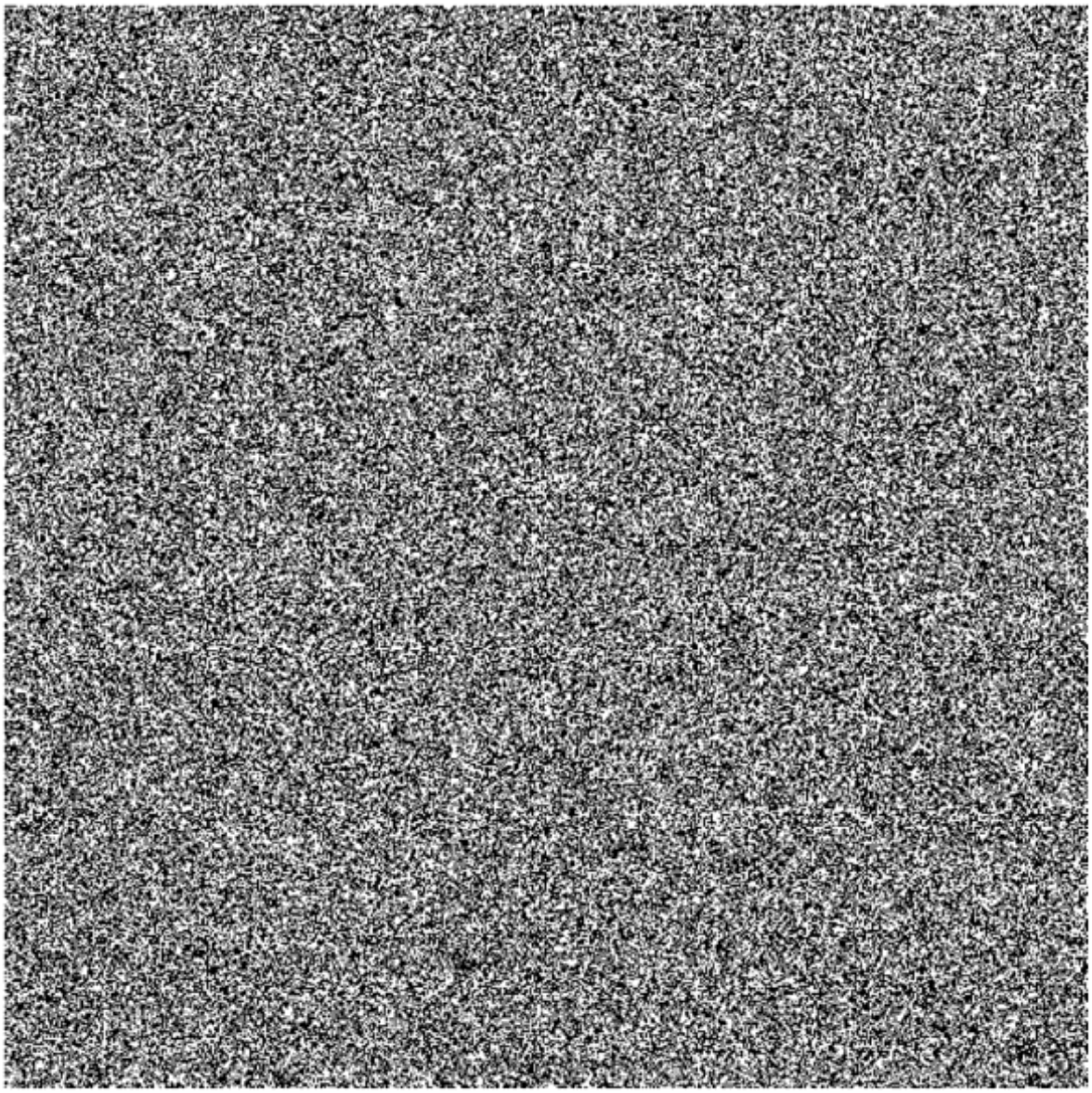
Testiranje te izrada LCG-a i LFG-a napravljeni su u skriptnom jeziku JavaScript (programski kod 4.1). Prvo su deklarirane i inicijalizirane varijable *a*, *c*, *m*, *seed* (X_0). Funkcija *Rand* primjenjuje formulu: $X_{n+1} = (a \cdot X_n + c) \ (mod\ m)$, a funkcija RandFloat služi kako bi nasumični brojevi bili u rasponu 0.0 do 1.0 (Babić, 2019). Korišteni su parametri poznatog *RANDU* LCG-a izrađenog od strane *IBM-a* 1968. godine. Konstante u pitanju su: $a = 65539,\ c = 0,\ m = 2^{31}$, a početni uvjet je 1 (Petrović, 2018).

```
8   <body>
9       <canvas id="canvas" width="600" height="600"></canvas>
10      <script>
11
12          var a = 65539,
13              c = 0,
14              m = Math.pow(2, 31),
15              seed = 1;
16
17          function Rand() {
18              seed = (a * seed + c) % m;
19              return seed;
20          }
21
22          function RandFloat(){
23              return Rand() / m;
24          }
25
26          var canvas = document.getElementById("canvas");
27          var context = canvas.getContext("2d");
28
29          var y = 0;
30          Ispis();
31          function Ispis() {
32              for(var x = 0; x < 600; x++){
33                  if(RandFloat() < 0.5){
34                      context.fillRect(x, y, 1, 1);
35                  }
36              }
37              y++
38              if(y < 600){
39                  requestAnimationFrame(Ispis);
40              }
41          }
42
43      </script>
44  </body>
```

Programski kod 4.1. LCG generator RANDU u jeziku JavaScript

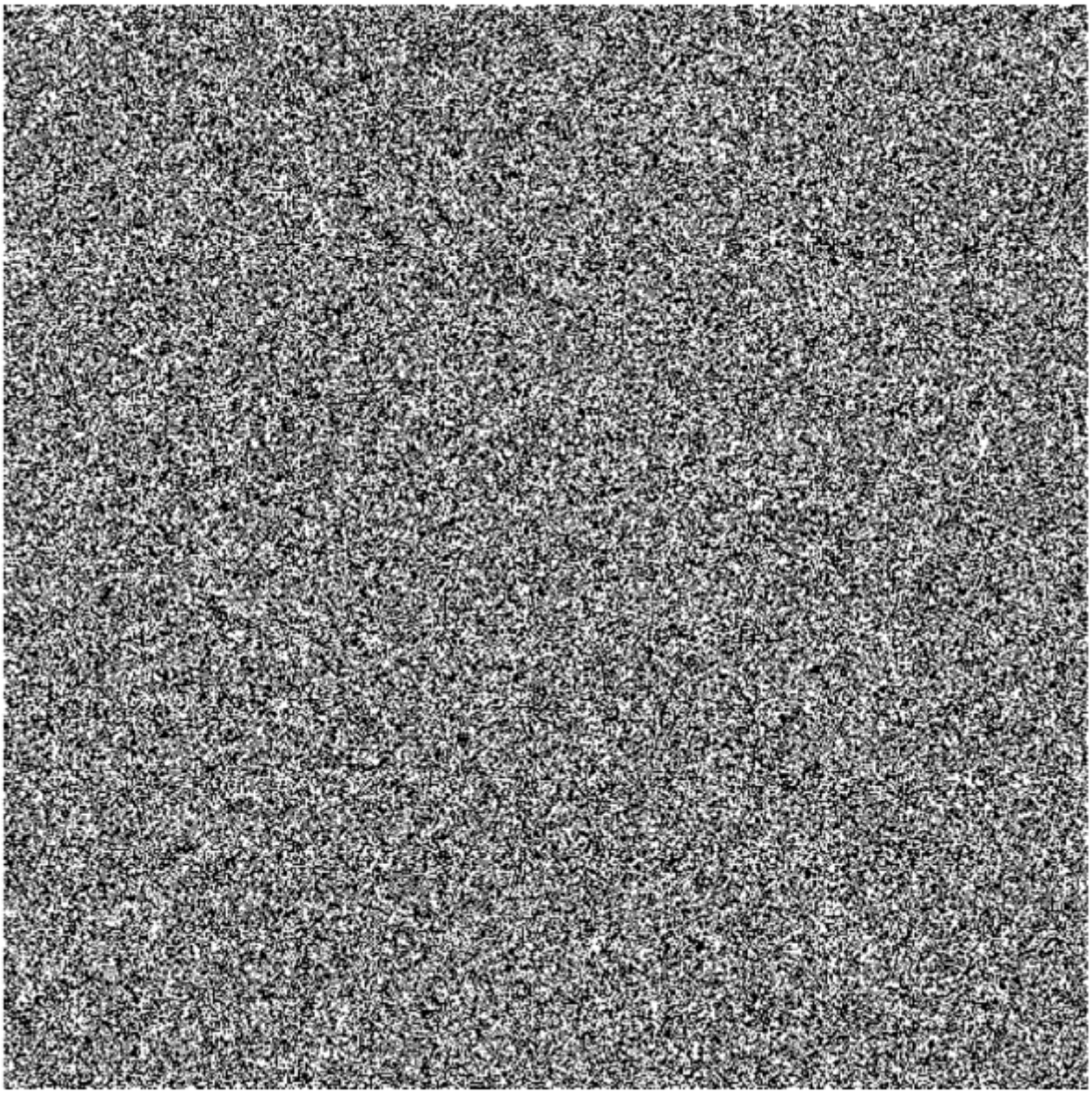
Za potrebe testiranja izrađena je mrežna stranica koja sadrži element *canvas* dimenzija 600x600 piksela te je u JavaScriptu deklarirana istoimena varijabla.

Funkcija *Ispis()* „crta“ na elementu *canvas*. Navedeno „crtanje“ obavlja se tako da se svaki piksel može popuniti crnom bojom ili ostaviti prazan. Ako je rezultat generatora manji od 0.5, piksel se popunjava crnom bojom (Babić, 2019). Isti princip je iskorišten za testiranje svih generatora u ovome radu. Naime, vizualni testovi su korisni jer ljudsko oko često može primijetiti uzorke ili anomalije. Najčešći način za takvo testiranje je putem grafikona (grafički prikazi 2.1. i 2.2.). Dobro obavljeno testiranje uključuje statističke testove i vizualne testove. Prednosti vizualnih testova su u tome što su lagani za shvatiti ili objasniti, mogu otkriti uzorke ili anomalije te ukoliko model prođe vizualni test, najvjerojatnije će proći i statističke testove. Ovakvi testovi nisu savršeni. Subjektivni su, ne pomažu pri odlučivanju između opcija sa sličnim rezultatima, a vizualni dokazi nisu dostatni kada je potrebno odbaciti model za koji se prije vjerovalo da je točan model (Tobias, 2012). Vizualni test pokazuje da linearno kongruentni generator *RANDU* daje dobre pseudo-nasumične brojeve. Na slici 4.1. golim okom nije vidljiv uzorak.

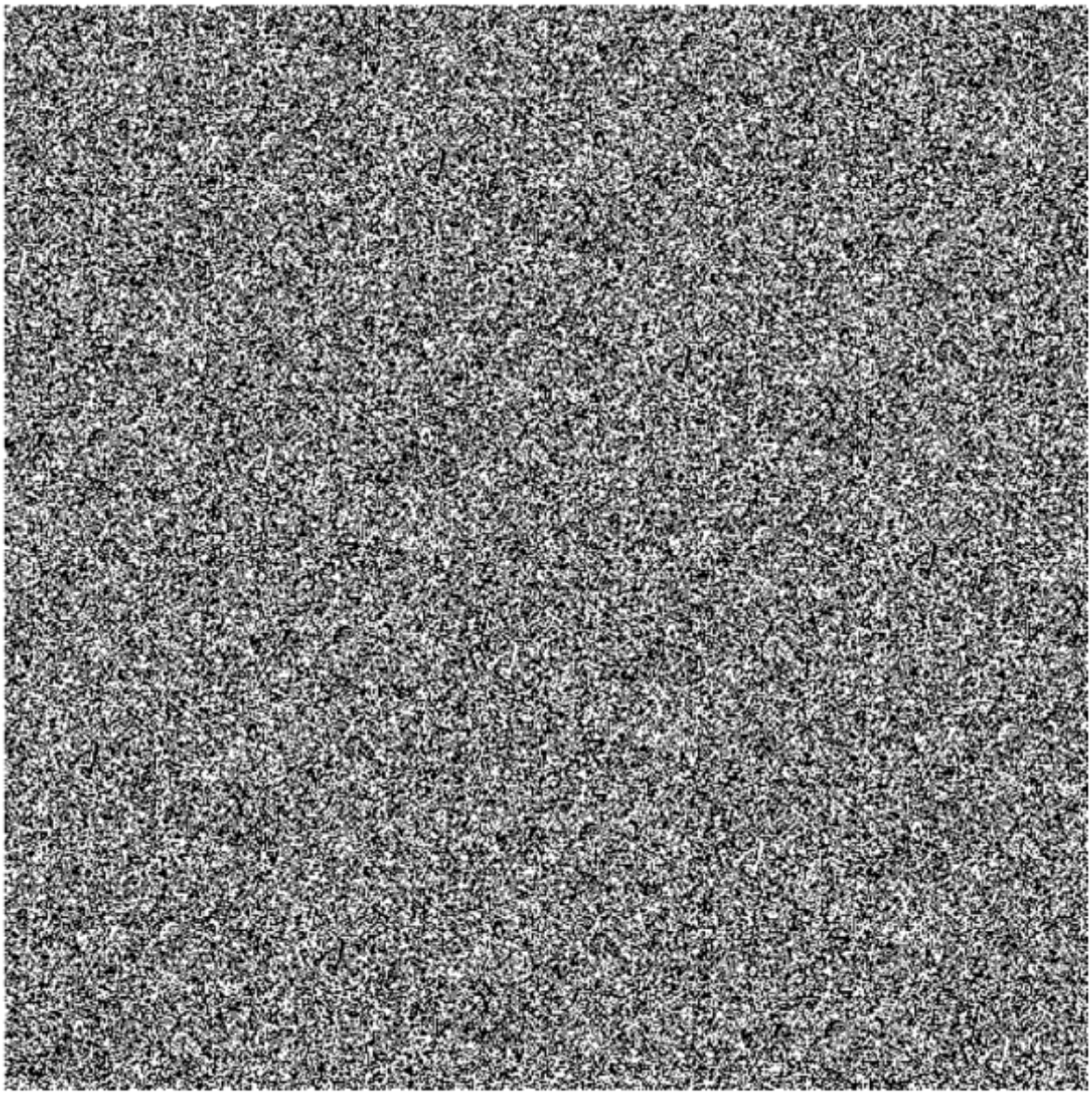


Slika 4.1. Vizualni prikaz rezultata *RANDU* LCG-a

Neki od poznatijih LCG generatora uključuju *„Minimalni standard“* LCG (a = 16807, c = 0, m = 2^{31}-1, sjeme = 1), *SIMSCRIPT LCG* (a = 630360016, c = 0, m = 2^{31}-1, sjeme = 1), *APPLE LCG* (a = 1220703125, c = 0, m = 2^{35}, sjeme = 1), *CRAY LCG* (a = 44485709377909, c = 0, m = 2^{48}, sjeme = 1), *Super-Duper* (a = 69069, c = 0, m = 2^{32}, sjeme = 1) te *DRAND48* (a = 25214903917, c = 11, m = 2^{48}, sjeme = 0) (Petrović, 2018). Slika 4.2. prikazuje vizualne rezultate generatora *Super-Duper*, a slika 4.3. generatora *DRAND48.* Na slikama je vidljivo da generatori nisu proizveli nikakav jasno vidljiv uzorak.

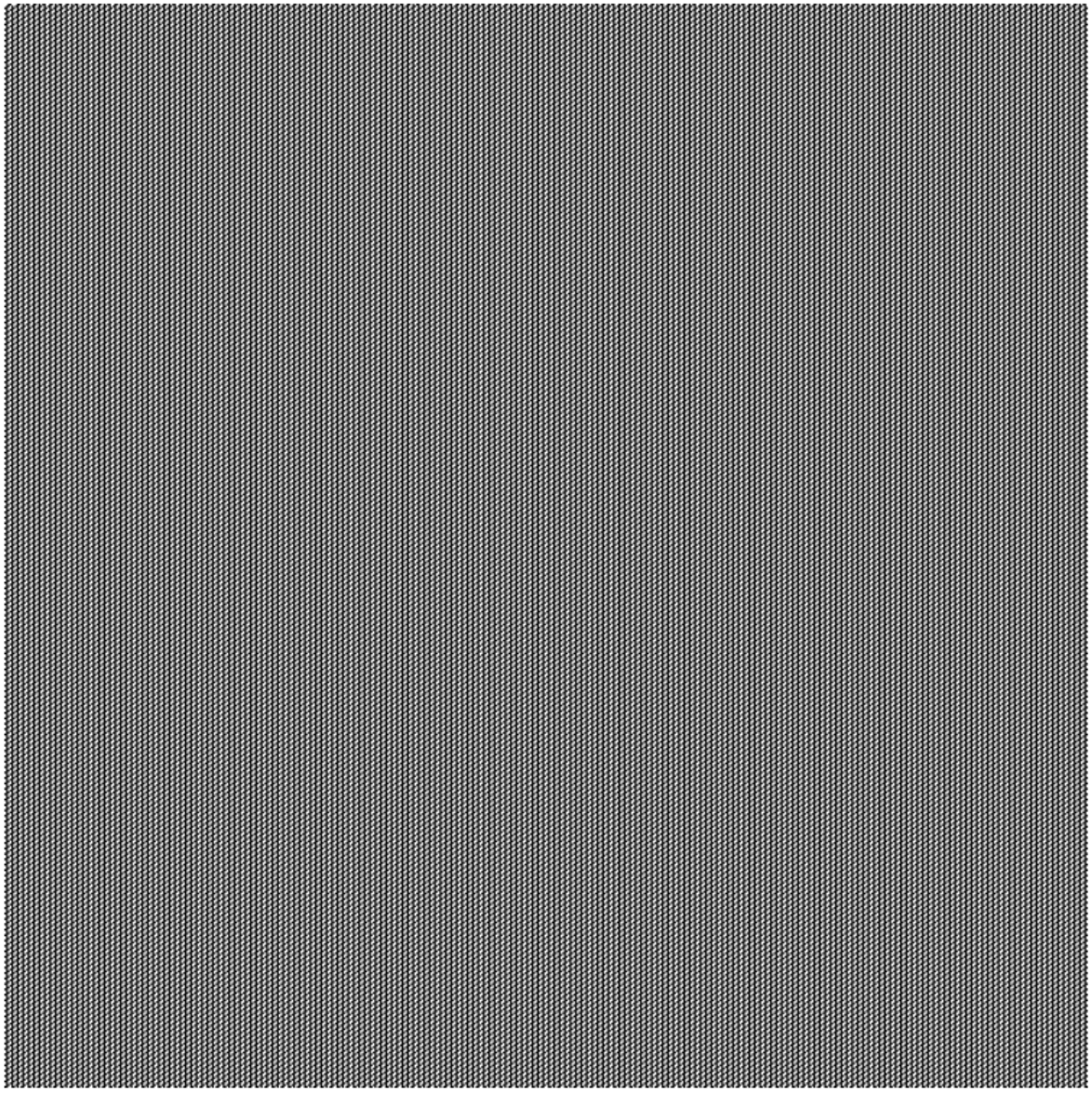


Slika 4.2. Vizualni prikaz rezultata parametra *Super-Duper* LCG-a

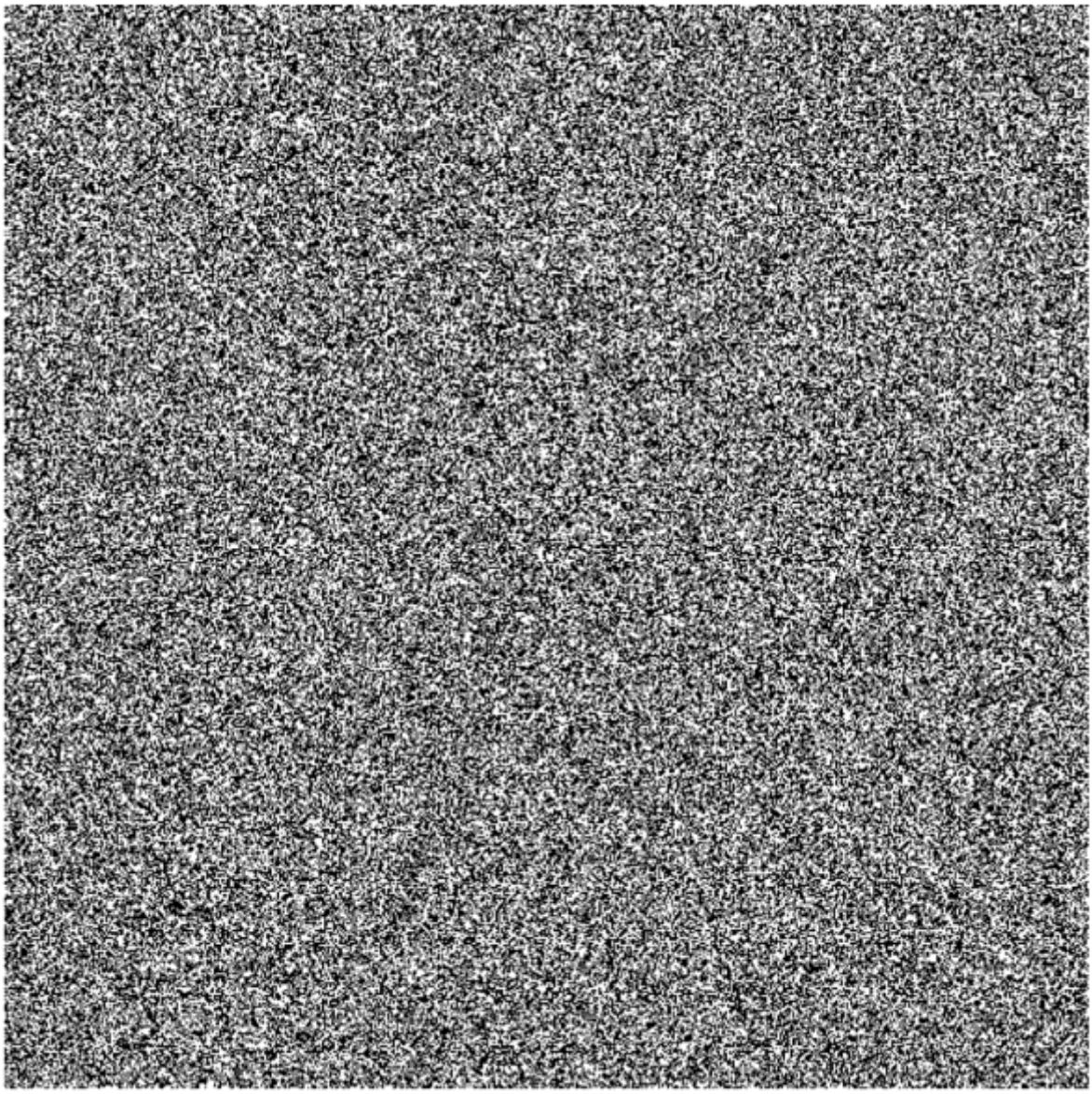


Slika 4.3. Vizualni prikaz rezultata parametra *DRAND48* LCG-a

Parametri korišteni za posljednja dva vizualna testa LCG-a odabrani su proizvoljno. Slika 4.4. prikazuje rezultate generatora čiji su parametri: *a = 569845648, c = 7, m = 2^{32}, sjeme = 4846583*. Navedeni parametri proizveli su loše rezultate jer je vidljiv jasan uzorak na slici. Uzorak nije vidljiv na slici 4.5. što znači da su odabrani dobri parametri (a = 1784639, c = 0, m = 2^{32}, sjeme = 9275183729).



Slika 4.4. Vizualni prikaz lošeg rezultata proizvoljno odabranih parametra LCG-a



Slika 4.5. Vizualni prikaz dobrog rezultata proizvoljno odabranih parametra LCG-a

4.2. Lagged Fibonacci Generator – LFG

Fibonaccijevi brojevi su sveprisutni oko nas, a Leonardo Fibonacci je prvi primijetio tu pravilnost u prirodi. Niz kreće od 0 i 1 te se do svakog idućeg broja dolazi zbrajanjem dva broja koji mu prethode. Stoga će prvih nekoliko članova niza glasiti: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55 (Buchanan, 2018). *Lagged Fibonacci Generator* (LFG) dobio je ime prema Fibonaccijevom nizu jer generator koristi

jednadžbu koja je vrlo slična rekurzivnoj jednadžbi koja se koristi za računanje Fibonaccijevog niza. Formula glasi:

$$X_n = X_{n-p} \cdot X_{n-q}$$

U formuli su *p* i *q* cijeli brojevi, gdje *p* mora biti veći od *q* te *q* mora biti veći od nule ili jednostavno iskazano kao $p > q > 0$. Znak · predstavlja binarnu operaciju (Petrović, 2018). Stoga · može biti zbrajanje (+), oduzimanje (-), množenje (x), *XOR* ili ekskluzivno-ili (⊕) i tako dalje (Buchanan, 2018). Najčešće se koristi operacija zbrajanje modulo *m*, a *m* se uzima kao potencija broja dva ($m = 2^M$). Stoga se formula LFG-a može zapisati i kao:

$$X_n = X_{n-p} + X_{n-q} \pmod{2^M}$$

Takav generator označujemo s uređenom trojkom LFG(*p, q, M*) (Jurić, 2001).

Za razliku od LCG-a, LFG se može koristiti u računalnim simulacijama jer se pokazao kao generator dobrog omjera brzine i kvalitete. Iako je brzina sekundarni zahtjev za generator nasumičnih brojeva, kod računalnih simulacija je nerijetko važan parametar generatora zbog ogromne količine nasumičnih brojeva s kojima simulacije uobičajeno raspolažu. Potencijalni problem u kontekstu računalnih simulacija predstavlja to što LGF zahtijeva veći memorijski prostor. Ipak, LFG se pokazao boljim od LCG-a i izvan konteksta računalnih simulacija. Naime, istraživanja su dokazala da niz generiran LFG-om posjeduje bolja nasumična svojstva, a to je posebno vidljivo kod perioda generiranja. Dobro odabrani parametri *p* i *q* mogu dostići duže periode u odnosu na periode LCG-a (Jurić, 2001).

Generator A je napravljen u programskom jeziku Python (programski kod 4.2.). Uzeti su parametri: *p = 7, q = 3, M = 10*. Varijabla *x* je sjeme u obliku liste te mora sadržavati minimalno sedam brojeva od kojih barem jedan mora biti neparan. Sedam brojeva je zbog toga što je potrebno imati barem jednako elemenata vrijednosti parametra *p* (u protivnom program javlja grešku), a potrebna parnost brojeva ovisi o korištenoj operaciji. Tako će zbrajanje zahtijevati minimalno jedan neparan član, a za množenje je broj potrebnih neparnih članova jednak vrijednosti parametra *p*. Parametri generatora A će generirati niz: 6, 1, 4, 4, 3, 9, 0, 4, 8, 1. Niz se generira tako što iz početne vrijednosti x_1 = 8, 6, [7], 5, 3, 0, [9] izdvojimo brojeve 7 i 9, a potom računamo 7+9 = 6 (mod 10). Ostali se brojevi dobivaju na sljedeći način (Toponce, 2015):

x_2 = 6 7 [5] 3 0 9 [6] = 5+6 ≡ 1 (mod 10),

x_3 = 7 5 [3] 0 9 6 [1] = 3+1 ≡ 4 (mod 10),

x_4 = 5 3 [0] 9 6 1 [4] = 0+4 ≡ 4 (mod 10),

x_5 = 3 0 [9] 6 1 4 [4] = 9+4 ≡ 3 (mod 10),

x_6 = 0 9 [6] 1 4 4 [3] = 6+3 ≡ 9 (mod 10),

x_7= 9 6 [1] 4 4 3 [9] = 1+9 ≡ 0 (mod 10),

x_8 = 6 1 [4] 4 3 9 [0] = 4+0 ≡ 4 (mod 10),

x_9 = 1 4 [4] 3 9 0 [4] = 4+4 ≡ 8 (mod 10),

x_{10} = 4 4 [3] 9 0 4 [8] = 3+8 ≡ 1 (mod 10)

```
p = 7
q = 3
x = [8, 6, 7, 5, 3, 0, 9]
print(type(x))
for n in range(10):
    for i in range(len(x)):
        if i == 0:
            out = (x[p-1] + x[q-1]) % 10
        elif 0 < i < 6:
            x[i] = x[i+1]
        else:
            x[i] = out
            print (x[i])
```

Programski kod 4.2. LFG generator A

Postoje dobri i često korišteni parametri LFG-a: LFG (*p = 17, q = 5, M = 31*), LFG (*p = 55, q = 24, M =31*) (Jurić, 2001). Parametri su implementirani u Python program (programski kod 4.3. i 4.4.). Iznos sjemena je ponovno jednak ili veći od parametra *p* te su dobiveni rezultati generatora B: 0, 7, 28, 15, 23, 16, 22, 16, 4, 4, a rezultati generatora C su: 6, 27, 12, 16, 4, 15, 14, 15, 18, 27.

```
p = 17
q = 5
x = [8, 6, 7, 5, 3, 0, 9, 2, 11, 10, 16, 15,
     19, 20, 12, 22, 25, 31, 29, 26, 4, 13]
for n in range(10):
    for i in range(len(x)):
        if i == 0:
            out = (x[p-1] + x[q-1]) % 31
        elif 0 < i < 21:
            x[i] = x[i+1]
        else:
            x[i] = out
            print (x[i])
```

Programski kod 4.3. LFG generator B

```
p = 55
q = 24
x = [41,5,21,28,60,10,26,38,50,2,15,27,22,57,16,0,
     4,56,3,52,35,14,29,32,40,7,34,20,44,19,46,8,
     51,13,47,45,58,1,36,59,33,31,18,55,23,42,25,
     12,54,30,53,17,49,48,37,9,6,39,11,43]
for n in range(10):
    for i in range(len(x)):
        if i == 0:
            out = (x[p-1] + x[q-1]) % 31
        elif 0 < i < 59:
            x[i] = x[i+1]
        else:
            x[i] = out
            print (x[i])
```

Programski kod 4.4. LFG generator C

4.3. Mersenne Twister

Makoto Matsumoto i Takuji Nishimura 1998. godine osmislili su generator pseudo-slučajnih brojeva Mersenne Twister. Ime algoritma dolazi od Mersenneovih prostih brojeva. Naime, dužina perioda generatora je Mersennov prosti broj koji možemo zapisati kao $M_n = 2^n-1$. Stoga, uz dobar odabir parametara, ovaj generator može doseći period generiranja od 2^{19937}-1 što je vrlo dugačak period (Babić, 2019). Riječ „*Twister*“ proizlazi iz toga što generator pripada u granu pseudo-nasumičnih

generatora brojeva koja se skraćeno naziva TGFSR (engl. Twisted Generalized Feedback Shift Register) (Jagannatam, 2010).

Postoji više varijacija Mersenne Twister algoritma, ali najčešće se koristi MT19937 kojemu je period 2^{19937}-1 te on koristi 32-bitne računalne riječi. MT19937-64 je verzija koja koristi 64-bitne riječi (Babić, 2019). Mutsuo Saito i Makoto Matsumoto su 2006. godine osmislili dva puta bržu verziju originalnog algoritma. SIMD-Oriented Fast Mersenne Twister (SFMT) iskorištava napretke u razvoju procesora kao što je SIMD (engl. *Single instruction, multiple data*). SFMT ima širok raspon mogućih perioda generiranja, od 2^{607}-1 do 2^{216091}-1 (Jagannatam, 2010).

Nedostatci Mersenne Twister algoritma su relativno veliki međuspremnik i to što nije kriptografski siguran, ali generator prolazi mnoge statističke testove slučajnosti. Diehard testovi i većinu TestU01 testova su dokazali da algoritam generira dobre pseudo-slučajne brojeve. Metoda se iskazala i na području brzine generiranja brojeva zato što je brža od ostalih metoda. Stoga ne čudi da je Mersenne Twister jedan od najčešće korištenih algoritama za generiranje pseudo-nasumičnih brojeva. Korišten je u mnogim programskim jezicima i računalnim sustavima kao što su Microsoft Excel, MATLAB, Python, R i drugi (Babić, 2019).

4.4. Xorshift

Vrlo brzi i jednostavni generatori pseudo-nasumičnih brojeva mogu biti napravljeni kombinacijom *xorshift* operacija. *Xorshift* sastoji se od kombinacije operacije ekskluzivno-ili ($\oplus$) računalne riječi i pomaka te iste računalne riječi. Jednostavan prikaz operacije u programskom jeziku C će biti: *yˆ(y<<a)*, za pomak ulijevo te *yˆ(y>>a)*, za pomak udesno. Operacija bi se implementirala na sljedeći način:

```
tmp = (xˆ(x << 15)); x = y; y = z; z = w; return w = (wˆ(w>>21))ˆ(tmpˆ(tmp>>4));
```

pri čemu su korištene četiri početne vrijednosti ili sjemena (*x, y, z, w*). Sa samo tri *xorshift* operacije po pozivu dobiven je period od 2^{128}-1 32-bitnih cijelih brojeva, a brzina im je preko 200 milijuna generiranih brojeva po sekundi. Danas se period od 2^{32}-1 smatra iznimno malenim, a ovakvi generatori mogu povećati period uz

minimalne napore. Tako će period za nizove parova (*x, y*) biti 2^{64}-1, za nizove trojki (*x, y, z*) period će biti 2^{96}-1, za nizove četvorki (x, y, z, w) je period 2^{128}-1, a za nizove petorki (*x, y, z, w, v*) period će iznositi 2^{160}-1. Ovakvi generatori uspješno prolaze testove slučajnosti (Marsaglia, 2003).

5. Kriptografska sigurnost i slučajni brojevi

Kriptografija osobu koja šalje poruku naziva Alice, osobu koja prima poruku Bob te osobu koja prisluškuje njihovu komunikaciju Eve. Cilj kriptografije je da Eve ne može razumjeti poruke koje Alice šalje Bobu. U ovome kontekstu Eve je treća strana te kao neželjeni promatrač iskorištava nesigurne komunikacijske kanale poput telefonskih linija ili računalnih mreža. Alice može poslati sliku, tekst, numeričke podatke ili neku drugu vrstu podatka. Te podatke je potrebno kriptirati, a oni se nazivaju otvoreni tekst. Otvoreni tekst je potrebno transformirati preko kriptografskih algoritama, a oni koriste ključ i matematičke funkcije kako bi zakrili podatke. Upravo generatori nasumičnih brojeva generiraju ključ, a postupak koji otvoreni tekst transformira uz pomoć ključa i kriptografskih algoritama naziva se šifriranje. Rezultat šifriranja je šifrat ili kriptogram te se on šalje Bobu. Matematičke funkcije imaju cilj preslikati osnovne elemente otvorenog teksta u osnovne elemente šifrata. Eve ne posjeduje ključ i zato ne može otkriti što je poslano, a Bob ima ključ pa nesmetano može pristupiti dešifriranim podatcima. Eve može raskriti šifrat uz pomoć kriptoanalize. Kriptoanaliza služi za dešifriranje poruke bez poznavanja ključa, a pokušaj kriptoanalize smatra se napadom. Ključeva je više te se njihov skup naziva prostor ključeva (Petrović, 2018).

Kriptografija ima zadaću osigurati pouzdan i siguran prijenos otvorenog teksta. Postoje smjernice kojih se potrebno pridržavati kako bi to bilo moguće, a one su: integritet (engl. *Data integrity*), tajnost (engl. *Confidentiality*), autentifikacija (engl. *Authentification*) i odgovornost (engl. *Responsibility*). Integritet se održava tako što samo ovlašteni korisnici imaju mogućnost promjene informacija, a bitno je osigurati način na koji će se provjeriti je li neovlaštena osoba mijenjala informacije. Tajnost se postiže na sličan način. Samo ovlaštene osobe smiju imati pristup šifriranim informacijama. Sveprisutna autentifikacija se može izvršiti na dvije razine. Prva je razina korisnika gdje se utvrđuje identitet korisnika kako bi se uvidjelo ima li pravo pristupa. Druga razina naziva se razina informacije te je ovdje autentifikacija provjera izvora informacija (tko je pošiljatelj, odakle dolazi, kada je stigla i slično). Odgovornost je sve važniji aspekt jer se jako puno novčanih transakcija obavlja preko interneta (Petrović, 2018).

Postoje kriptografski sigurni generatori (engl. *Cryptographically secure pseudorandom bit generator* ili skraćeno CSPRBG) koji daju mnogo kvalitetnije pseudo-nasumične bitove. Bitovi kasnije tvore nasumične brojeve te mnogi generatori rade na takvom principu. Kvaliteta takvog niza odlična je i za računalne simulacije, ali se ne koristi u te svrhe zbog sporog generiranja brojeva. Naime, kriptografski sigurni generatori namijenjeni su za generiranje malog broja podataka (par desetaka ili stotina bitova) pa nije potrebna velika brzina. Generatori pseudo-nasumičnih brojeva opisani u prethodnim poglavljima okarakterizirani su niskom razinom sigurnosti zbog čega nisu primjenjivi u kriptografske svrhe. (Jurić, 2001). Izlazna vrijednost kongruentnih generatora može se predvidjeti i bez poznavanja korištenih parametara, a period Mersenne Twistera se može otkriti iz dovoljno dugačkog niza izlazne vrijednosti (Herrero-Collantes i Garcia-Escartin, 2017).

Moderna kriptografija većinski slijedi Kerckhoffsovo načelo, a to znači da se pretpostavlja da je sustav siguran i ako svi njegovi aspekti, osim ključa, dođu u neželjene ruke. Takvi otvoreni sustavi počivaju na tajnosti ključa što je praktično jer sve što je potrebno napraviti u slučaju ugroženog sustava je promjena ključa. Zato je dobar ključ iznimno bitan za svaki kriptografski sustav, a time je i slučajnost jedan od osnovnih aspekata kriptografije. Dobar ključ bira se nasumično iz cijelog prostora ključeva, a rezultat je n-bitan niz (Herrero-Collantes i Garcia-Escartin, 2017). Prilikom generiranja ključa, ulazna vrijednost mora biti odabrana na način da ju je gotovo nemoguće odgonetnuti pogađanjem ili pretragom svih mogućih kombinacija. Primjerice da je trenutno vrijeme ulazna vrijednost, ono mora biti odabrano što preciznije što znači da se uračunava trenutna godina, mjesec, dan, sati, minute, sekunde i milisekunde. Sigurnost ključa se osigurava i iznimno velikim periodom. Veći broj različitih nizova osigurava da ih je nemoguće sve pretražiti, a dobar generator neće imati očite elemente pravilnosti čime se rizik pogađanja pretragom znatno smanjuje (Jurić, 2001). Nadalje, nasumičnost se koristi i u drugim dijelovima kriptografskih sustava. Nasumični brojevi se koriste prilikom odabira brojeva koji se moraju koristiti samo jednom kao što je slučaj u vektorima inicijalizacije ili u sekvencijalnim brojevima (Herrero-Collantes i Garcia-Escartin, 2017).

5.1. Blum Blum Shub

Jedan od najboljih kriptografski sigurnih generatora pseudo-slučajnih brojeva je Blum Blum Shub. Napravljen je 1986. godine, a autori su Lenore Blum, Manuel Blum i Michael Shub (Buchanan, 2019). Izlazna vrijednost su bitovi dobiveni prema sljedećoj rekurzivnoj formuli:

$$\mathrm{X_{n+1}} = \mathrm{X_n^2} \bmod \mathrm{M}$$

pi čemu je *M* umnožak dvaju prostih brojeva (*p*, *q*), a X_n je n-ti broj koji se koristi kao period. Početna vrijednost je X_0 te se ona dobiva iz generatora istinski slučajnih brojeva. Generator ima mnoga kriptografski poželjna svojstva. Ukoliko je napadač naučio period X_n u fazi *n*, nepredvidljivost prethodnih bitova binarnog niza ostaje osigurana. Nadalje, pogađanje X_{n-1} iz X_n je iznimno zahtjevno izračunati. Generator je moguće uspješno napasti ukoliko napadač posjeduje kvantno računalo, zna vrijednost *M* te koristi Shorov algoritam za faktorizaciju cijelih brojeva (Herrero-Collantes i Garcia-Escartin, 2017). Mana generatora je to što je relativno spor, ali je i dalje koristan u kriptografske svrhe (Buchanan, 2019).

5.2. CryptMT

CryptMT je prva implementacija Mersenne Twistera kao kriptografskog sigurnog generatora. Sadrži kombinaciju algoritma Mersenne Twistera i filtra s 32-bitnom memorijom koji koristi izlaznu vrijednost Mersenne Twistera. Mersenne Twister generira niz cijelih brojeva duljine jedne riječi, a to je duljina memorije filtra (Matsumoto et al., 2007). Filtar koristi množenje i operaciju ili (engl. *OR*, |). Generator je iznimno brz, a njegova sigurnost uvelike ovisi o izlaznoj vrijednosti originalnog Mersenne Twistera. Trenutno najnovija verzija je CryptMT verzija 3. koja je 1,8 puta je brža od prve verzije, a iznimna duljina perioda čini najnoviju verziju otpornom na bilo kakve značajnije pokušaje napada. Postoji nepovjerenje u dizajn ovog generatora, a to uvelike proizlazi iz nerazumijevanja nelinearnog filtra generatora (Jagannatam, 2010).

5.3. ANSI X9.17 generator

Kriptografski algoritam *Data Encryption Standard* (DES) bio je vrlo popularan način šifriranja u prošlom stoljeću. DES otvoreni tekst pretvara u 64 bitne blokove i njih uz pomoć ključa pretvara u šifrat. Šifriranje obavlja uz pomoć simetričnog ključa što znači da se isti ključ koristi za šifriranje i dešifriranje (Thakkar, 2021). ANSI X9.17 generator koristi DES kako bi generirao pseudo-slučajne brojeve. Algoritam koristi trostruko E-D-E kriptiranje koje se definira preko izraza:

$$E(x) = E_{K3}(D_{K2}(E_{K1}(x)))$$

gdje $E_K(x)$ označava šifriranje pomoću ključa K, a $D_K(x)$ označava dešifriranje pomoću ključa K. Ukoliko vrijedi:

$$K_1 = K_3,$$

tada se u algoritmu koriste dva ključa te se radi o trostrukom E-D-E kriptiranju. Ulazni podatci uključuju nasumični 64-bitni podatak *s* koji je sjeme ovog algoritma, cijeli broj *m* i dva ključa kojima se kriptira otvoreni tekst (K_1 i K_2). Prvo se računa pomoćna vrijednost:

$$I = E(D),$$

gdje D predstavlja 64-bitni zapis trenutnog vremena. Potom se koriste ulazni podatci tako da se za i = 1 do *m* računa:

$$x_i = E(I \oplus s),$$

$$s = E(x_i \oplus I).$$

Računanje daje niz od 64-bitnih slučajnih brojeva: x_1, … , x_m. Rezultati ovog generatora nisu kriptografski sigurni, ali generator je dovoljno siguran za mnogo primjena. Jednostavniji je i mnogo brži od kriptografski sigurnih generatora (Jurić, 2001).

5.4. RSA generator

Kriptografski siguran generator pseudo-nasumičnih brojeva je i RSA generator. Sigurnost ovog generatora temelji se na problemu faktorizacije broja *n*. Sjeme se uzima iz skupa cijelih brojeva te se iz toga radi niz elemenata. Generirani niz brojeva nastaje tako što se iz svakog elementa u nizu uzima najmanje značajan bit. Nadalje, taj niz se formira tako da svaki element niza postaje enkripcija svoga prethodnika, a enkripcija se izvršava RSA algoritmom (Petrović, 2018). Generator koristi algoritam u kojem je prvo potrebno izračunati izraze:

$$n = p \cdot q,$$

$$L(n) = (p - 1) \cdot (q - 1).$$

Za potrebe računanja umnožaka generiraju se dva prosta broja: *p* i *q*. Zatim se odabire nasumičan broj *e* gdje vrijedi $1 < e < L(n)$. Najveća zajednička mjera za *e* i *L(n)* iznosi 1 (Jurić, 2001). Parametri *n* i *e* su tajni, a *p* i *q* moraju biti tajni (Petrović, 2018). Potom se odabire nasumičan cijeli broj x_0 koji će biti sjeme generatora. Sjeme se mora nalaziti u intervalu [1, *n* - 1] (Jurić, 2001). Broj bitova koje ima sjeme je *k*, a *p* i *q* imaju *k/2* bitova. Na kraju za $i \geq 0$ vrijedi:

$$x_i = x^e_{i-1} \bmod n$$

Rezultat je niz nasumičnih bitova $z_1, \ldots, z_n$ (Petrović, 2018). Potrebno je napomenuti kako je ovaj generator spor u generiranju nasumičnih brojeva. Optimizacija je moguća tek uz veliki oprez. Naime, potrebno je izbjeći korelaciju između susjednih bitova jer to značajno povećava mogućnost provale (Jurić, 2001).

6. Zaključak

Generatori pseudo-nasumičnih brojeva primjenjivi su u mnogim domenama. Računalne simulacije, numeričke aproksimacije, računalne igre, kriptografski protokoli neke su od domena koje koriste pseudo-nasumične brojeve. Generatori pseudo-slučajnih brojeva predstavljaju jeftiniju, manje kvarljivu i bržu alternativu generatorima istinski slučajnim brojevima. Generatori istinski nasumičnih brojeva često se koriste samo kako bi generirali sjeme za generatore pseudo-slučajnih brojeva. Ipak, postoje ograničenja pseudo-nasumičnih brojeva. Deterministička priroda računalno generiranih nizova predstavlja problem u području kriptografije gdje je iznimno bitna nepredvidljivost. Neki od kriptografski sigurnih generatora pseudo-nasumičnih brojeva uključuju Blum Blum Shub, CryptMT te RSA generator.

Kvaliteta generatora pseudo-nasumičnih brojeva uvelike ovisi o njegovom periodu. Generatori kao što su LFG i LCG imaju malen period, ali su dobra polazišna točka kod istraživanja metoda računalno generiranih slučajnih brojeva. LFG je pokazao bolje rezultate u usporedbi s LCG-om, ali uz moguće povećanje memorijskih zahtjeva. Generator Mersenne Twister je najraširenij i najrazvijeniji od ove tri vrste generatora. Naime, postoje verzije Mersenne Twistera koje koriste razvoj novih tehnologija i postoje verzije koje su uz to i kriptografski sigurne.

7. Literatura

1) Anonymus, (2023), *Randomly Generated Levels*. Dostupno na: https://tvtropes.org/pmwiki/pmwiki.php/Main/RandomlyGeneratedLevels, [pristupljeno 24. lipnja, 2023.].
2) Babić, A., (2019). *IZVORI SLUČAJNIH BROJEVA I NJIHOVA UPOTREBA*, Završni rad, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Sveučilište Josipa Jurja Strossmayera u Osijeku. Dostupno na: https://zir.nsk.hr/islandora/object/etfos:2207, [pristupljeno 24. lipnja, 2023.].
3) Buchanan, B., (2018), *For The Love of Computing: The Lagged Fibonacci Generator — Where Nature Meet Random Numbers*. Dostupno na: https://medium.com/asecuritysite-when-bob-met-alice/for-the-love-of-computing-the-lagged-fibonacci-generator-where-nature-meet-random-numbers-f9fb5bd6c237, [pristupljeno 24. lipnja, 2023.].
4) Buchanan, B., (2019), *Go be rAnd0m with Blum Blum Shub*. Dostupno na: https://medium.com/asecuritysite-when-bob-met-alice/doo-wop-diddi-diddi-blum-blum-shub-96eebf44868d, [pristupljeno 24. lipnja, 2023.].
5) Herrero-Collantes, M., & Garcia-Escartin, J. C. (2017), *Quantum random number generators*, Reviews of Modern Physics, 89(1). Dostupno na: https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.89.015004, [pristupljeno 24. lipnja, 2023.].
6) Jagannatam, A., (2010), *Mersenne Twister–A Pseudo random number generator and its variants*, George Mason University, Department of Electrical and Computer Engineering. Dostupno na: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=285a65e11dbb6183a963489bc30b28ab04c6d7cf, [pristupljeno 24. lipnja, 2023.].
7) Jurić, S., (2001), *Generatori pseudo-slučajnih brojeva*, Seminarski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu. Dostupno na: http://sigurnost.zemris.fer.hr/random/2001_juric/, [pristupljeno 24. lipnja, 2023.].
8) L'Ecuyer, P., (2017), *History of uniform random number generation*, WSC 2017 - Winter Simulation Conference, Las Vegas, Sjedinjene Američke Države, prosinac 2017., Las Vegas: IEEE, pp. 202-230. Dostupno na:

https://inria.hal.science/hal-01561551/file/wsc17rng-history-report.pdf, [pristupljeno 24. lipnja, 2023.].

9) Li, R., (2015), *A true random number generator algorithm from digital camera image noise for varying lighting conditions*, SoutheastCon, Fort Lauderdale, Sjedinjene Američke Države, travanj 2015., Fort Lauderdale: IEEE, pp. 1-8. Dostupno na: https://www.researchgate.net/publication/283021854_A_True_Random_Number_Generator_algorithm_from_digital_camera_image_noise_for_varying_lighting_conditions, [pristupljeno 24. lipnja, 2023.].
10) Marsaglia, G., (2003), *Xorshift rngs*, Journal of Statistical software, 8(14), pp. 1-6. Dostupno na: https://www.jstatsoft.org/article/view/v008i14, [pristupljeno 24. lipnja, 2023.].
11) Matsumoto, M., Saito, M., Nishimura, T., & Hagita, M. (2007), *CryptMT stream cipher version 3. eSTREAM*, ECRYPT Stream Cipher Project. Dostupno na: https://www.semanticscholar.org/paper/CRYPTMT-STREAM-CIPHER-VERSION-3-Saito-Nishimura/c46dcbd2e08055fbf87438af4d8d3c6ac1193d67, [pristupljeno 24. lipnja, 2023.].
12) Pease, C., (2018), *An Overview of Monte Carlo Methods*, Towards Data Science. Dostupno na: https://towardsdatascience.com/an-overview-of-monte-carlo-methods-675384eb1694, [pristupljeno 24. lipnja, 2023.].
13) Petrović, N., (2018), *Alati za statističko testiranje nizova pseudoslučajnih brojeva*, Diplomski rad, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek, Sveučilište Josipa Jurja Strossmayera u Osijeku. Dostupno na: https://zir.nsk.hr/en/islandora/object/etfos%3A2055/datastream/, [pristupljeno 24. lipnja, 2023.].
14) Skočić, M. (2017), *Generiranje pseudoslučajnih brojeva i testovi slučajnosti*, Diplomski rad, Prirodoslovno-matematički fakultet, Sveučilište u Zagrebu. Dostupno na: https://repozitorij.pmf.unizg.hr/islandora/object/pmf%3A3861/datastream/PDF/view, [pristupljeno 24. lipnja, 2023.].
15) Thakkar, M., (2021), *What Is DES Encryption? A Look at the DES Algorithm*. Dostupno na: https://sectigostore.com/blog/what-is-des-encryption-a-look-at-the-des-algorithm/, [pristupljeno 24. lipnja, 2023.].

16) Tobias, P., (2012), *NIST/SEMATECH e-Handbook of Statistical Methods*. Dostupno na: https://www.itl.nist.gov/div898/handbook/apr/section2/apr231.htm, [pristupljeno 24. lipnja, 2023.].
17) Toponce, A., (2015), *The Lagged Fibonacci Generator*. Dostupno na: https://pthree.org/2015/05/29/the-lagged-fibonacci-generator/, [pristupljeno 24. lipnja, 2023.].
18) Urbiha, I., (2010), *Generiranje niza pseudoslucajnih brojeva*, Matematičko fizički list, pp. 75-82. Dostupno na: https://www.bib.irb.hr/554519, [pristupljeno 24. lipnja, 2023.].

Generatori pseudo-nasumičnih brojeva

Sažetak

U radu su opisane metode kojima se može generirati niz pseudo-slučajnih brojeva. Pojašnjeni generatori uključuju metodu sredine kvadrata, LCG, LFG, Mersenne Twister te ANSI X9.17 generator. Prikazana je i programska implementacija LCG i LGF generatora. Rezultati LCG programa testirani su vizualnim testovima. Korišteno je više setova parametara od kojih su neki uzeti od uvriježenih generatora, a neki su proizvoljno odabrani. Jedan set proizvoljno odabranih parametara se pokazao lošim, a drugi je dao zadovoljavajuće rezultate. Opisani su i neki kriptografski sigurni generatori (Blum Blum Shub, CryptMT te RSA generator). Pojašnjeni su problemi generatora pseudo-slučajnih brojeva u kontekstu kriptografske sigurnosti.

Rad pruža povijesni pregled slučajnih brojeva i njihovih generatora. Opisano je što čini niz brojeva slučajnim i koje su razlike između istinski slučajnih i pseudo-slučajnih brojeva. Za obje skupine brojeva pojašnjena su njihova ograničenja i prednosti. Isto je napravljeno i za generatore istinskih te pseudo-slučajnih brojeva. Nadalje, prikazane su i razne primjene generatora pseudo-nasumičnih brojeva te su dani konkretni primjeri tih primjena.

Ključne riječi: pseudo-slučajni brojevi, PRNG, LCG, LFG, kriptografska sigurnost

Pseudorandom number generators

Summary

This thesis describes methods that can be used to generate a sequence of pseudo-random numbers. Generators that are explained include middle square method, LCG, LFG, Mersenne Twister and ANSI X9.17 generator. The program implementation of the LCG and LGF generator is also shown. The results of the LCG program were tested with visual tests. Several sets of parameters were used, some of which were taken from established generators, and some of which were arbitrarily selected. One set of arbitrarily chosen parameters turned out to be bad, and the other gave satisfactory results. Some cryptographically secure generators (Blum Blum Shub, CryptMT and RSA generator) are also described. The problems of pseudo-random number generators in the context of cryptographic security are clarified.

The thesis provides a historical overview of random numbers and their generators. It describes what makes a sequence of numbers random and what the differences are between truly random and pseudorandom numbers. For both groups of numbers, their limitations and advantages are explained. The same was done for true and pseudorandom number generators. Furthermore, various applications of pseudo-random number generators are presented and the thesis provides specific examples of these applications.

Key words: pseudorandom numbers, PRNG, LCG, LFG, cryptographic security