Generiranje pseudoslučajnih brojeva i testovi slučajnosti

Skočić, Mario

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Mario Skočić

GENERIRANJE PSEUDOSLUČAJNIH BROJEVA I TESTOVI SLUČAJNOSTI

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Saša Singer

Zagreb, lipanj, 2017.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

Tema ovog rada su pseudoslučajni brojevi, odnosno slučajni brojevi koje generiraju računalni algoritmi.

Slučajni brojevi imaju široku primjenu u raznim područjima, od simulacija, numeričke analize, programiranja, pa sve do upotrebe u rekreacijske svrhe, odnosno igrara na sreću. Tradicionalne metode za generiranje slučajnih brojeva, kao što su bacanje kocke, ili pomoću kola ruleta, su jako spore i nepraktične. Stoga ih mijenjaju računala. Međutim, kako su računala deterministički strojevi, brojevi koje oni generiraju ne mogu biti stvarno slučajni.

Za početak dat ću osnovni pogled na slučajnost i generiranje slučajnih brojeva na svima razumljivom jeziku. Opisat ću i prve primitivne algoritme za generiranje slučajnih brojeva.

Rad se sastoji od dva dijela. U prvom dijelu ću dati detaljnu matematičku analizu jednog od najkvalitetnijih generatora pseudoslučajnih brojeva, poznatog pod imenom xorshift generator. Također, izložit ću i osnove generatora koji se, vjerojatno, najviše koristi, Mersenne Twister.

Drugi dio rada su statistički testovi slučajnosti koji ispituju kvalitetu generatora slučajnih brojeva. Slučajni brojevi koje generiraju generatori moraju biti nepredvidivi i nezavisni. Cilj testova je otkriti pravilnosti u dobivenim nizovima. Dat ću detaljan pregled testova sadržanih u paketu TestU01, softverskoj biblioteci koja sadrži skup različitih empirijskih testova slučajnosti za generatore slučajnih brojeva. Također, dat ću samo kratak pregled skupa testova iz paketa Diehard.

Poglavlje 1

Slučajnost i slučajni brojevi

1.1 Slučajnost

Slučajnost je nedostatak uzorka, pravila ili predvidivosti događaja. Slučajan niz događaja, simbola ili koraka nema redoslijeda i ne slijedi nikakav razuman uzorak. Individualni slučajni događaji su po definiciji nepredvidivi, ali u dosta slučajeva frekvencija različitih ishoda nakon velikog broja događaja može biti predviđena. Na primjer, kada bacamo dvije igrače kocke, rezultat svakog pojedinog bacanja je nepredvidiv, ali suma brojeva na obje kocke jednaka 7 će se pojavljivati dva puta češće nego suma brojeva na obje kocke jednaka 4.

Koncept slučajnosti potječe još iz antičkih vremena kada su ljudi bacali kocku kako bi prorekli sudbinu, a to je s vremenom evoluiralo u igre slučajnosti. Kinezi su prije 3000 godina vjerojatno bili prvi ljudi koji su formalizirali izglede i šanse. Grčki filozofi su diskutirali o slučajnosti, ali ne u kvantitativnoj formi. Tek u 16. stoljeću talijanski matematičari započinju sa formaliziranjem šansi u različitim igrama na sreću. Otkriće infinitezimalnog računa donosi pozitivan utjecaj na formalno proučavanje slučajnosti. 1888. godine John Venn je u svojoj knjizi *The Logic of Chance* napisao poglavlje *The conception of randomness* u kojem je dao svoj pogled na slučajnost znamenki broja π, koristeći ih kako bi konstruirao slučajnu šetnju u dvije dimenzije.

1.2 Pseudoslučajnost

Pseudoslučajan proces je proces koji izgleda kao slučajan, no on to nije. Pseudoslučajni nizovi uglavnom posjeduju statističku slučajnost, a generirani su u potpunosti determinističkim procesima. Takav proces možemo lakše izvršiti nego stvarni slučajni proces i možemo ga ponovno koristiti kako bismo dobili potpuno isti niz vrijednosti, što je korisno za testiranja.

Generiranje slučajnih brojeva ima velike primjene, no o tome više kasnije. Prije modernog računarstva, znanstvenici koji su trebali slučajne brojeve generirali su ih pomoću kocke, karata, kola ruleta i slično, ili pomoću tablica slučajnih brojeva. Prvi pokušaj pružanja „zalihe“ slučajnih brojeva bio je 1927. godine, kada je Cambridge University Press objavio tablicu 41,600 slučajnih brojeva koje je dobio L. H. C. Tippett. Kasnije, 1947. godine, RAND Corporation je generirao slučajne brojeve elektroničkom simulacijom kola ruleta, a rezultati su objavljeni 1955. pod nazivom *A Million Random Digits with 100,000 Normal Deviates*. John von Neumann je pionir kompjuterskih generiranja slučajnih brojeva. 1949. godine, Derrick Henry Lehmer je izmislio linearni kongruentni generator koji je dugo vremena bio najviše korišten generator pseudoslučajnih brojeva. Danas, većina generatora u upotrebi je zasnovana na linearnoj rekurziji. Napretkom računarstva, algoritmi koji generiraju pseudoslučajne brojeve zamijenili su tablice slučajnih brojeva, a generatori pravih slučajnih brojeva se koriste vrlo rijetko.

1.3 Generiranje slučajnih brojeva

Generiranje slučajnih brojeva je generiranje niza brojeva ili simbola koji ne mogu unaprijed biti predviđeni. Različite potrebe za slučajnošću dovele su do nekoliko različitih metoda generiranja slučajnih podataka. Neke od metoda postoje još iz antičkih vremena, poput bacanja kocke ili novčića, miješanje karata i mnoge druge. Zbog mehaničke prirode ovih metoda, generiranje velikog broja slučajnih brojeva zahtijeva puno posla i vremena. Stoga su se nekada rezultati spremali i objavljivali u obliku tablica slučajnih brojeva. Danas, s napretkom računarstva i algoritama za generiranje slučajnih brojeva, upravo oni postaju glavni izvor slučajnih brojeva.

Postoji više različitih računalnih metoda generiranja slučajnih brojeva. Međutim, mnogima nedostaje svojstvo stvarne slučajnosti dobivenih nizova, iako neki mogu uspješno proći statističke testove slučajnosti koji mjere koliko su rezultati predvidivi. S druge strane, postoje i posebno dizajnirani, kriptografski sigurni, algoritmi bazirani na Yarrow algoritmu i sličnima.

Dvije su osnovne metode generiranja slučajnih brojeva. Prva metoda koristi neke fizičke fenomene za koje znamo da su slučajni i mjeri moguća odstupanja u procesu. Takve algoritme nazivamo pravi generatori slučajnih brojeva (eng. True Random Number Generators, ili kraće TRNG). Druga metoda koristi računalne algoritme koji mogu producirati duge nizove naizgled slučajnih rezultata, koji su zapravo u potpunosti određeni početnom vrijednošću koju nazivamo ključ ili sjeme. Ovakav tip algoritama nazivamo generatori pseudoslučajnih brojeva (eng. Pseudo-Random Number Generators, ili kraće PRNG).

1.3.1 Generatori pseudoslučajnih brojeva (PRNGs)

PRNG su algoritmi koji koriste matematičke formule kako bi generirali niz naizgled slučajnih brojeva. Dobiveni nizovi nisu slučajni jer su dobiveni determinističkim procesom i u potpunosti su određeni početnog vrijednosti algoritma, tzv. sjemenom ili ključem. PNRG algoritam pokrećemo njegovim početnim stanjem, sjemenom. Dobiveni niz će svaki put biti identičan kada algoritam inicijaliziramo istim sjemenom. Većina algoritama generira nizove s dobrim statističkim svojstvima, no ovako dobiveni nizovi su periodički, tj. nakon nekog vremena brojevi se ponavljaju.

John von Neumann je 1946. godine dao prvi PRNG poznat kao metoda srednjeg kvadrata. Algoritam funkcionira vrlo jednostavno: uzmi neki broj, kvadriraj ga, te uzmi srednje znamenke kao slučajan broj. Taj broj koristimo kao sjeme za sljedeću iteraciju algoritma. Na primjer, kvadriranjem broja 1111 dobivamo broj 1234321 kojeg možemo zapisati kao 01234321. Kvadriranjem 4-bitnog broja smo dobili 8-bitni broj, te je naš slučajan broj 2343. Koristeći 2343 kao sjeme za sljedeću iteraciju, dobivamo broj 4896 i tako dalje. Von Neumann je koristio desetoznamenkaste brojeve, no princip je isti. Problem ove metode je da se sve sekvence nakon nekog vremena ponavljaju, a neke čak i jako brzo, poput 0000. Von Neumann je bio svjestan ovog problema, ali postupak je bio dovoljno dobar za njegove potrebe.

Jedan od najstarijih i najpoznatijih generatora pseudoslučajnih brojeva je linearni kongruentni generator (LCG). Generator je definiran rekurzivnom relacijom:

$$X_{n+1} = (aX_n + c) \bmod m$$

pri čemu je X niz pseudoslučajnih vrijednosti, a:

m, $m > 0$ – modul

a, $0 < a < m$ – multiplikator

c, $0 \leq c < m$ – inkrement

X_0, $0 \leq X_0 < m$ – sjeme, početna vrijednost

su vrijednosti koje specificiraju generator.

Period linearno kongruentnih generatora je najviše m, a za neke izbore početnih vrijednosti period je i puno kraći.

1997. godine dolazi do izuma Mersenne Twister algoritma koji je riješio većinu mana prijašnjih generatora. Mersenne Twister ima period od $2^{19937} - 1$ iteracija i radio je brže nego svi dotadašnji generatori. S vremenom su razvijeni i WELL generatori koji su poboljšanja Mersenne Twister algoritma. 2003. godine George Marsaglia uvodi xorshift generatore koji su također zasnovani na linearnog rekurziji. Ovakvi algoritmi su jako brzi i uspješno prolaze komplicirane statističke testove.

Mane pseudoslučajnih generatora su determinističnost i periodičnost, no njihova velika prednost je brzina generiranja slučajnih brojeva.

1.3.2 Pravi generatori slučajnih brojeva (TRNGs)

Generatori stvarno slučajnih brojeva izvode slučajnost iz fizičkih fenomena i uvode je u računalo. Fizički fenomeni mogu biti vrlo jednostavni, poput varijacije u pokretima računalnog miša ili vrijeme između pritiska dvije tipke na tipkovnici. U praksi, ovakvi izvori slučajnosti pokazali su se kao loši. Atmosferski šum, termalni šum, mnogi elektromagnetski i kvantni fenomeni, poput kozmičke radijacije ili radioaktivnog raspadanja mjerenog u kratkim vremenskim okvirima su dobri izvori prirodne entropije.

Ovakvi generatori su nedeterministički i nisu periodički. No, generiranje slučajnih brojeva je dosta sporije obzirom na generiranje pomoću PRNG-a.

1.4 Primjene slučajnih brojeva

Slučajni brojevi imaju primjene u različitim područjima.

- Simulacije. Kada se računalo koristi za simuliranje prirodnih fenomena, slučajni brojevi su potrebni da stvari učine realnijima. Simulacije mogu biti različite, od proučavanja nuklearne fizike (slučajno sudaranje čestica) do različitih operacijskih istraživanja (na primjer, ljudi dolaze na aerodrom u slučajnim intervalima).

- Uzimanje uzoraka. Često je nepraktično proći kroz sve uzorke, ali korištenje slučajnih uzoraka pruža uvid u tipično ponašanje.

- Numerička analiza. Različite napredne tehnike rješavanja složenih numeričkih problema osmišljene su pomoću slučajnih brojeva. Na ovu temu napisano je nekoliko knjiga.

- Programiranje. Slučajne vrijednosti su dobar izvor podataka za testiranje efikasnosti računalnih algoritama.

- Odlučivanje. Navodno, mnogi menadžeri donose odluke bacanjem novčića ili pikado strelica. Također, neki profesori odabiru zadatke za ispite na slučajan način. Ponekad je potrebno donijeti potpuno slučajnu odluku.

- Estetika. Malo slučajnosti u računalno generiranoj grafici i glazbi ponekad daje ugodniji kontekst.

- Rekreacija. Bacanje kocke, miješanje karata, okretanje kola ruleta i slično. Iz ovih tradicionalnih korištenja slučajnih brojeva došlo je do naziva Monte Carlo algoritam, izraza koji označava algoritam koji koristi slučajne brojeve.

Poglavlje 2

Algoritmi za generiranje slučajnih brojeva

2.1 xorshift

Ovo je klasa jednostavnih, jako brzih generatora slučajnih brojeva iz [3], s periodima $2^k - 1$ za $k = 32, 64, 96, 128, 160, 192$, koji prilično uspješno prolaze testove slučajnosti.

2.1.1 Uvod

xorshift generator slučajnih brojeva (xorshift RNG) producira niz od $2^{32} - 1$ cijelih brojeva x ili niz od $2^{64} - 1$ parova x, y, ili niz od $2^{96} - 1$ trojki x, y, z, itd. Slučajni brojevi se generiraju po principu ponavljanja jednostavne operacije: ekskluzivno-ili (xor) računalne riječi s pomakom te iste riječi. U C-u, to je jednostavna operacija `y^(y<<a)` za pomak ulijevo, odnosno `y^(y>>a)` za pomak udesno. U Fortranu to možemo zapisati kao

```
ieor(y, ishft(y,a))
```

s negativnim `a` za pomak udesno, odnosno pozitivnim `a` za pomak ulijevo.

Kombiniranjem ovakvih xorshift operacija s različitim pomacima i argumentima dobivamo jako brze i jednostavne generatore pseudoslučajnih brojeva koji dosta uspješno prolaze testove slučajnosti. Da bismo dobili ideju o brzini i efikasnosti xorshift operacija, pogledajmo kako izgleda osnovni dio C procedure koja, sa samo tri xorshift operacije po pozivu, generira $2^{128} - 1$ slučajnih 32-bitnih cijelih brojeva, koristeći 4 vrijednosti x, y, z, w kao sjeme algoritma:

```
tmp=(x^(x<<15));
x=y;  y=z;  z=w;
return w=(w^(w>>21))^(tmp^(tmp>>4));
```

Ovakva procedura je ekstremno brza, tipično 200 milijuna brojeva po sekundi, a dobiveni niz slučajnih cijelih brojeva prolazi sve testove slučajnosti, posebno zahtjevne testove (vidjeti u [4]) i testove iz skupa testova Diehard, koji će biti opisan kasnije.

U ovom poglavlju dana je teorija koja je u pozadini različitih xorshift generatora slučajnih brojeva s periodima do 2^{160}. Moguće je postići i dosta veće periode drugim metodama koje ćemo samo spomenuti na kraju poglavlja.

2.1.2 Teorija

Matematički model za većinu generatora slučajnih brojeva može se svesti na sljedeću formu: imamo skup sjemena Z koji se sastoji od uređenih m-torki $(x_1, x_2, \ldots, x_m)$ i bijektivne funkcije $f()$ na Z. Najčešće je Z skup cijelih brojeva, ali za bolje generatore on može biti i skup uređenih parova, trojki, itd. Ako z odaberemo uniformno i slučajno iz skupa Z, tada je izlaz algoritma niz $f(z), f^2(z), f^3(z), \ldots$, pri čemu $f^2(z)$ znači $f(f(z))$, itd. Kako je f bijekcija nad Z, tada je i vrijednost $f(z)$ uniformna na Z, kao i $f^2(z)$. Naravno, svi elementi niza $f(z), f^2(z), \ldots$ su uniformno distribuirani nad Z, ali nisu neovisni.

Za xorshift generatore slučajnih brojeva, skup sjemena Z je skup $1 \times n$ binarnih vektora $\beta = (b_1, b_2, \ldots, b_n)$, isključujući nul-vektor. Najčešće je n jednak 32, 64, 96 i slično, tako da elementi mogu biti dobiveni koristeći 32-bitne računalne riječi. Elementi vektora β su iz polja $0, 1$. Za naš xorshift RNG, trebamo invertibilnu funkciju nad Z, a za to ćemo koristiti linearan operator nad prostorom binarnih vektora, karakteriziran nesingularnom $n \times n$ binarnom matricom T. Ako je vektor β izabran uniformno slučajno iz skupa sjemena Z, tada je i niz $\beta T, \beta T^2, \beta T^3, \ldots$, također uniformno distribuiran nad Z.

Teorem 2.1.1. *Da bi nesingularna $n \times n$ binarna matrica T davala sve moguće ne-nul $1 \times n$ binarne vektore u nizu $\beta T, \beta T^2, \ldots$, za svaki početni ne-nul $1 \times n$ binarni vektor β, nužan i dovoljan uvjet je da matrica T ima red $2^n - 1$, u grupi svih nesingularnih $n \times n$ binarnih matrica.*

Dokaz. Nužnost: Ako je period od $\beta T, \beta T^2, \ldots$ jednak $k = 2^n - 1$, tada je $\beta T^k = \beta$ za svaki $1 \times n$ binarni vektor β, pa mora biti $T^k = I$. Ako je $T^j = I$ za neki $j < k$, tada je period od $\beta T, \beta T^2, \ldots$, strogo manji od $2^n - 1$.

Dovoljnost: Ako je red matrice T jednak $k = 2^n - 1$, tada su matrice $T, T^2, T^3, \ldots, T^k$ nesingularne i različite. Pomoću karakterističnog polinoma od T i Euklidovog algoritma, svaka od njih može biti reprezentirana polinomom u T stupnja strogo manjeg od n. Kako ima točno $k = 2^n - 1$ ne-nul polinoma u T stupnja strogo manjeg od n, tada oni svi moraju biti različite nesingularne matrice $T, T^2, \ldots, T^k$. Posebno, ako je polinom u T singularna matrica, tada se može reducirati, pomoću karakterističnog polinoma od T, do nul matrice. Slijedi da period niza $\beta T, \beta T^2, \ldots$ mora biti $k = 2^n - 1$, jer da je $\beta T^j = \beta$ za neki ne-nul vektor β i za $j < k$, to bi značilo da je $T^j + I$ singularna. □

2.1.3 Primjena na xorshift RNG

Za binarni vektor y, operacija yT je u računalnom smislu skupa, osim u slučaju ako matrica T ima posebnu formu. Neka je L $n \times n$ binarna matrica koja odgovara pomaku ulijevo za jedno mjesto vektora y, tj. L ima sve vrijednosti nula, osim jedinica na donjoj sporednoj dijagonali. Tada, uz

$$T = I + L^a,$$

linearna transformacija yT odgovara xorshift operaciji u C-u, `y^(y<<a)`, tj. zbrajanju modulo 2, binarnog vektora y s a puta ulijevo pomaknutom verzijom samoga sebe. Slično, ako je R matrica koja odgovara pomaku udesno za jedno mjesto (L transponirano), tada xorshift operaciju `y^(y>>b)` možemo zapisati kao yT, za

$$T = I + R^b.$$

Matrice $I + L^a$ i $I + R^b$ dimenzija $n \times n$ su nesingularne. Npr., očito je $L^n = 0$ i stoga je konačan niz $I + L^a + L^{2a} + L^{3a} + \cdots$ jednak inverzu od $(I + L^a)$. Očiti kandidat za matricu reda $2^n - 1$ je

$$T = (I + L^a)(I + R^b).$$

Nažalost, kada je n jednak 32 ili 64, ni za jedan izbor brojeva a i b matrica T neće imati traženi red. (Preliminarni test za kandidate je kvadrirati matricu T n puta. Ako rezultat nije T, tada T ne može imati red $2^n - 1$.)

No, ako je matrica

$$T = (I + L^a)(I + R^b)(I + L^c),$$

postoji mnogo odabira parametara a, b, c tako da matrica T ima red $2^n - 1$, za $n = 32$ ili $n = 64$. Imamo 81 uređenih trojki (a, b, c), $a < c$, za koje je 32×32 binarna matrica $T = (I + L^a)(I + R^b)(I + R^c)$ reda $2^{32} - 1$. To su trojke:

(1, 3,10)	(1, 5,16)	(1, 5,19)	(1, 9,29)	(1,11, 6)	(1,11,16)
(1,19, 3)	(1,21,20)	(1,27,27)	(2, 5,15)	(2, 5,21)	(2, 7, 7)
(2, 7, 9)	(2, 7,25)	(2, 9,15)	(2,15,17)	(2,15,25)	(2,21, 9)
(3, 1,14)	(3, 3,26)	(3, 3,28)	(3, 3,29)	(3, 5,20)	(3, 5,22)
(3, 5,25)	(3, 7,29)	(3,13, 7)	(3,23,25)	(3,25,24)	(3,27,11)
(4, 3,17)	(4, 3,27)	(4, 5,15)	(5, 3,21)	(5, 7,22)	(5, 9, 7)
(5, 9,28)	(5, 9,31)	(5,13, 6)	(5,15,17)	(5,17,13)	(5,21,12)
(5,27, 8)	(5,27,21)	(5,27,25)	(5,27,28)	(6, 1,11)	(6, 3,17)
(6,17, 9)	(6,21, 7)	(6,21,13)	(7, 1, 9)	(7, 1,18)	(7, 1,25)

(7, 13, 25)	(7, 17, 21)	(7, 25, 12)	(7, 25, 20)	(8, 7, 23)	(8, 9, 23)
(9, 5, 1)	(9, 5, 25)	(9, 11, 19)	(9, 21, 16)	(10, 9, 21)	(10, 9, 25)
(11, 7, 12)	(11, 7, 16)	(11, 17, 13)	(11, 21, 13)	(12, 9, 23)	(13, 3, 17)
(13, 3, 27)	(13, 5, 19)	(13, 17, 15)	(14, 1, 15)	(14, 13, 15)	(15, 1, 29)
(17, 15, 20)	(17, 15, 23)	(17, 15, 26)			

Za ovih 81 uređenih trojki, trojke (c, b, a), također, daju puni red za T. Dakle, ima ih ukupno 162. Nadalje, možemo dobiti još 162 puna perioda ako T transponiramo. Koristeći trojke (a, b, c) za formiranje matrice

$$T = (I + R^a)(I + L^b)(I + R^c),$$

dobivamo ukupno 324 različite mogućnosti. Na kraju, za svaku od 324 matrice T oblika

$$T = (I + L^a)(I + R^b)(I + L^c) \qquad \text{ili} \qquad T = (I + R^a)(I + L^b)(I + R^c),$$

odgovarajuće matrice

$$T = (I + L^a)(I + L^c)(I + R^b) \qquad \text{i} \qquad T = (I + R^a)(I + R^c)(I + L^b),$$

također, imaju period $2^{32} - 1$. Sve skupa imamo 648 različitih izbora.

Ukratko, za svaku od 81 uređenih trojki (a, b, c), $a < c$, svaka od ovih 8 linija C koda može poslužiti kao baza za 32-bitni xorshift RNG s periodom $2^{32} - 1$:

```
y^=y<<a;  y^=y>>b;  y^=y<<c;
y^=y<<c;  y^=y>>b;  y^=y<<a;
y^=y>>a;  y^=y<<b;  y^=y>>c;
y^=y>>c;  y^=y<<b;  y^=y>>a;
y^=y<<a;  y^=y<<c;  y^=y>>b;
y^=y<<c;  y^=y<<a;  y^=y>>b;
y^=y>>a;  y^=y>>c;  y^=y<<b;
y^=y>>c;  y^=y>>a;  y^=y<<b;
```

Za 64-bitne cijele brojeve, sljedećih 275 uređenih trojki (a, b, c), $a < c$, daju matrice

$$T = (I + L^a)(I + R^b)(I + L^c)$$

reda $2^{64} - 1$:

(1, 1,54)	(1, 1,55)	(1, 3,45)	(1, 7, 9)	(1, 7,44)	(1, 7,46)
(1, 9,50)	(1,11,35)	(1,11,50)	(1,13,45)	(1,15, 4)	(1,15,63)
(1,19, 6)	(1,19,16)	(1,23,14)	(1,23,29)	(1,29,34)	(1,35, 5)
(1,35,11)	(1,35,34)	(1,45,37)	(1,51,13)	(1,53, 3)	(1,59,14)
(2,13,23)	(2,31,51)	(2,31,53)	(2,43,27)	(2,47,49)	(3, 1,11)
(3, 5,21)	(3,13,59)	(3,21,31)	(3,25,20)	(3,25,31)	(3,25,56)
(3,29,40)	(3,29,47)	(3,29,49)	(3,35,14)	(3,37,17)	(3,43, 4)
(3,43, 6)	(3,43,11)	(3,51,16)	(3,53, 7)	(3,61,17)	(3,61,26)
(4, 7,19)	(4, 9,13)	(4,15,51)	(4,15,53)	(4,29,45)	(4,29,49)
(4,31,33)	(4,35,15)	(4,35,21)	(4,37,11)	(4,37,21)	(4,41,19)
(4,41,45)	(4,43,21)	(4,43,31)	(4,53, 7)	(5, 9,23)	(5,11,54)
(5,15,27)	(5,17,11)	(5,23,36)	(5,33,29)	(5,41,20)	(5,45,16)
(5,47,23)	(5,53,20)	(5,59,33)	(5,59,35)	(5,59,63)	(6, 1,17)
(6, 3,49)	(6,17,47)	(6,23,27)	(6,27, 7)	(6,43,21)	(6,49,29)
(6,55,17)	(7, 5,41)	(7, 5,47)	(7, 5,55)	(7, 7,20)	(7, 9,38)
(7,11,10)	(7,11,35)	(7,13,58)	(7,19,17)	(7,19,54)	(7,23, 8)
(7,25,58)	(7,27,59)	(7,33, 8)	(7,41,40)	(7,43,28)	(7,51,24)
(7,57,12)	(8, 5,59)	(8, 9,25)	(8,13,25)	(8,13,61)	(8,15,21)
(8,25,59)	(8,29,19)	(8,31,17)	(8,37,21)	(8,51,21)	(9, 1,27)
(9, 5,36)	(9, 5,43)	(9, 7,18)	(9,19,18)	(9,21,11)	(9,21,20)
(9,21,40)	(9,23,57)	(9,27,10)	(9,29,12)	(9,29,37)	(9,37,31)
(9,41,45)	(10, 7,33)	(10,27,59)	(10,53,13)	(11, 5,32)	(11, 5,34)
(11, 5,43)	(11, 5,45)	(11, 9,14)	(11, 9,34)	(11,13,40)	(11,15,37)
(11,23,42)	(11,23,56)	(11,25,48)	(11,27,26)	(11,29,14)	(11,31,18)
(11,53,23)	(12, 1,31)	(12, 3,13)	(12, 3,49)	(12, 7,13)	(12,11,47)
(12,25,27)	(12,39,49)	(12,43,19)	(13, 3,40)	(13, 3,53)	(13, 7,17)
(13, 9,15)	(13, 9,50)	(13,13,19)	(13,17,43)	(13,19,28)	(13,19,47)
(13,21,18)	(13,21,49)	(13,29,35)	(13,35,30)	(13,35,38)	(13,47,23)
(13,51,21)	(14,13,17)	(14,15,19)	(14,23,33)	(14,31,45)	(14,47,15)
(15, 1,19)	(15, 5,37)	(15,13,28)	(15,13,52)	(15,17,27)	(15,19,63)
(15,21,46)	(15,23,23)	(15,45,17)	(15,47,16)	(15,49,26)	(16, 5,17)

(16, 7, 39)	(16, 11, 19)	(16, 11, 27)	(16, 13, 55)	(16, 21, 35)	(16, 25, 43)
(16, 27, 53)	(16, 47, 17)	(17, 15, 58)	(17, 23, 29)	(17, 23, 51)	(17, 23, 52)
(17, 27, 22)	(17, 45, 22)	(17, 47, 28)	(17, 47, 29)	(17, 47, 54)	(18, 1, 25)
(18, 3, 43)	(18, 19, 19)	(18, 25, 21)	(18, 41, 23)	(19, 7, 36)	(19, 7, 55)
(19, 13, 37)	(19, 15, 46)	(19, 21, 52)	(19, 25, 20)	(19, 41, 21)	(19, 43, 27)
(20, 1, 31)	(20, 5, 29)	(21, 1, 27)	(21, 9, 29)	(21, 13, 52)	(21, 15, 28)
(21, 15, 29)	(21, 17, 24)	(21, 17, 30)	(21, 17, 48)	(21, 21, 32)	(21, 21, 34)
(21, 21, 37)	(21, 21, 38)	(21, 21, 40)	(21, 21, 41)	(21, 21, 43)	(21, 41, 23)
(22, 3, 39)	(23, 9, 38)	(23, 9, 48)	(23, 9, 57)	(23, 13, 38)	(23, 13, 58)
(23, 13, 61)	(23, 17, 25)	(23, 17, 54)	(23, 17, 56)	(23, 17, 62)	(23, 41, 34)
(23, 41, 51)	(24, 9, 35)	(24, 11, 29)	(24, 25, 25)	(24, 31, 35)	(25, 7, 46)
(25, 7, 49)	(25, 9, 39)	(25, 11, 57)	(25, 13, 29)	(25, 13, 39)	(25, 13, 62)
(25, 15, 47)	(25, 21, 44)	(25, 27, 27)	(25, 27, 53)	(25, 33, 36)	(25, 39, 54)
(28, 9, 55)	(28, 11, 53)	(29, 27, 37)	(31, 1, 51)	(31, 25, 37)	(31, 27, 35)
(33, 31, 43)	(33, 31, 55)	(43, 21, 46)	(49, 15, 61)	(55, 9, 56)	

Kao i u slučaju 32-bitnih cijelih brojeva, odabir bilo koje od 275 uređenih trojki brojeva (a, b, c) za 64-bitne nizove i odabir bilo koje od ranije spomenutih 8 linija C koda daju xorshift RNG s periodom $2^{64} - 1$, tj. imamo ukupno $8 \times 275 = 2200$ različitih izbora.

Slijedi jednostavna 32-bitna xorshift procedura u C-u koja uzima 32-bitno sjeme y:

```
unsigned long xor(){
static unsigned long y=2463534242;
y^=(y<<13); y=(y>>17); return (y^=(y<<5)); }
```

Ova procedura koristi izbor $(a, b, c) = (13, 17, 5)$ koji prolazi skoro sve testove slučajnosti, osim binarnog rang testa iz skupa Diehard.

Za C kompajlere koji koriste 64-bita, procedura za RNG s periodom $2^{64} - 1$ i 64-bitnim sjemenom x izgleda ovako:

```
unsigned long long xor64(){
static unsigned long long x=88172645463325252LL;
x^=(x<<13);  x^=(x>>7);  return (x^=(x<<17)); }
```

2.1.4 Binarni vektorski prostori dimenzija $n = 96, 128, 160, \ldots$

Dok je prikladno koristiti 32-bitne računalne riječi za reprezentiranje elemenata vektorskog prostora dimenzija 32, odnosno dimenzije 64 za kompajlere koji prihvaćaju 64-bitne

cijele brojeve, za dobivanje duljih xorshift perioda potrebna je metoda za reprezentiranje elemenata vektorskih prostora većih dimenzija. Dobar pristup ovom problemu je na vektore dimenzije 1×96 gledati kao 32-bitne komponente (x, y, z) ili vektore dimenzije 1×128 prikazati kao 32-bitne komponente (x, y, z, w), itd. Pitanje je kako odabrati matricu T koja definira linearne transformacije na takvom vektorskom prostoru.

Prirodno je odabrati T kao blok-matricu sljedećih oblika (za $n = 64, 96, 128$):

$$T = \begin{pmatrix} 0 & A \\ I & B \end{pmatrix}, \quad T = \begin{pmatrix} 0 & 0 & A \\ I & 0 & C \\ 0 & I & B \end{pmatrix}, \quad T = \begin{pmatrix} 0 & 0 & 0 & A \\ I & 0 & 0 & C \\ 0 & I & 0 & D \\ 0 & 0 & I & B \end{pmatrix}.$$

Tada je, na primjer,

$$(x, y, z)T = (y, z, xA + yC + zB)$$

i potrebno je pronaći 32×32 matrice A, B, C tako da su 32-bitne operacije xA, yC, zB jednostavne i da T ima red $2^{96} - 1$ u grupi 96×96 nesingularnih binarnih matrica. Pokazuje se da se puni period $2^{64} - 1$, $2^{96} - 1$, odnosno $2^{128} - 1$, ostvaruje jednostavnim odabirom

$$A = (I + L^a)(I + R^b) \qquad \text{i} \qquad B = (I + R^c),$$

dok su svi ostali blokovi 0. Tako bi, za prikladne odabire trojki (a, b, c), ove matrice reda $2^{64} - 1$, $2^{96} - 1$, odnosno $2^{128} - 1$, izgledale ovako:

$$\begin{pmatrix} 0 & (I + L^a)(I + R^b) \\ I & (I + R^c) \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & (I + L^a)(I + R^b) \\ I & 0 & 0 \\ 0 & I & (I + R^c) \end{pmatrix}, \quad \begin{pmatrix} 0 & 0 & 0 & (I + L^a)(I + R^b) \\ I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & (I + R^c) \end{pmatrix}.$$

Neki od mogućih izbora trojki (a, b, c) su:

za $n = 64$, $(10, 13, 10), (8, 9, 22), (2, 7, 3), (23, 3, 24)$;

za $n = 96$, $(10, 5, 26), (13, 19, 3), (1, 17, 2), (10, 1, 26)$;

za $n = 128$, $(5, 14, 1), (15, 4, 21), (23, 24, 3), (5, 12, 29)$.

U svakom slučaju, red blokovske matrice T je $2^n - 1$. Slijede osnovni dijelovi C procedura za generiranje nizova perioda $2^{64} - 1$, $2^{96} - 1$ i $2^{128} - 1$:

```
t=(x^(x<<a));  x=y;  return y=(y^(y>>c))^(t^(t>>b));
t=(x^(x<<a));  x=y;  y=z;  return z=(z^(z>>c))^(t^(t>>b));
t=(x^(x<<a));  x=y;  y=z;  z=w;  return w=(w^(w>>c))^(t^(t>>b));
```

Sljedeći primjer daje xorshift RNG s periodom $2^{160} - 1$, koristeći istu shemu pridruživanja, uz odabir parametara

$$(a, b, c) = (2, 1, 4), (7, 13, 6), (1, 1, 20).$$

```
t=(x^(x>>a));
x=y;  y=z;  z=w;  w=v;
return v=(v^(v>>c))^(t^(t>>b));
```

Po potrebi, duge periode možemo, također, ostvariti i ako za zadnji stupac blok-matrice odaberemo $I + L^a, I + R^b, I + L^c, I + R^d$. C kod za ovaj slučaj s periodima $2^{96} - 1$, odnosno $2^{128} - 1$ izgleda ovako:

```
t=(x^(x<<3))^(y^(y>>19))^(z^(z<<6));
x=y;  y=z;
return(z=t);
```

odnosno

```
t=(x^(x<<20))^(y^(y>>11))^(z^(z<<27))^(w^(w>>6));
x=y;  y=z;  z=w;
return(w=t);
```

Sve gornje procedure vraćaju 32-bitni cijeli broj, iako (stvarno) te procedure kreiraju niz parova (x, y), trojki (x, y, z) ili četvorki (x, y, z, w) s maksimalnim periodom, $2^n - 1$, nad binarnim poljem vektora dimenzije $n = 64, 96, 128$. Za svaki od gornjih izbora parametara, dobiveni xorshift generatori slučajnih brojeva prolaze sve Diehard testove i iznimno su brzi, mogu izbaciti oko 200 milijuna slučajnih brojeva u sekundi.

2.1.5 Zaključak

Mnoštvo jednostavnih i brzih generatora slučajnih brojeva može se dobiti kombiniranjem xorshift operacija na različite načine. Usprkos jednostavnosti ovih algoritama, dobiveni nizovi slučajnih brojeva jako uspješno prolaze testove slučajnosti. Od stotine ranije opisanih izbora vrijednosti (a, b, c), najjednostavnija je C operacija

```
x^=(x<<a);  x^=(x>>b);  x^=(x<<c);
```

za dobivanje nizova 32-bitnih vrijednosti perioda $2^{32} - 1$ ili nizova 64-bitnih vrijednosti perioda $2^{64} - 1$.

Prema modernim standardima, period $2^{32} - 1$ se smatra malim. S utroškom još jako malo računalnog vremena, xorshift operacije mogu davati nizove parova (x, y) s periodom

$2^{64} - 1$, trojki (x, y, z) s periodom $2^{96} - 1$, četvorki (x, y, z, w) s periodom $2^{128} - 1$ ili petorki (x, y, z, w, v) s periodom $2^{160} - 1$. Proširenje na veće k-torke brojeva je izvedivo, ali osnovna ideja — izračunaj novu vrijednost kao funkciju prethodnih k vrijednosti, te pridruži $x = y$; $y = z$; $z = w$; itd., postaje manje efikasna od metode kada k prethodno generiranih vrijednosti čuvamo u cirkularnoj tablici. Pomoću takvih tablica, pomnoži s prijenosom (eng. multiply-with-carry, ili kraće MWC) i dodatno pomnoži s prijenosom (eng. complimentary-multiply-with-carry, ili kraće CMWC) metode dostižu periode do 2^{131102}. Te metode su pobliže objašnjene u [2].

Usporedimo xorshift RNG s periodom $2^{128} - 1$ s MWC RNG algoritmom sličnog perioda. Prvo, xorshift algoritam:

```
unsigned long xor128(){
static unsigned long x=123456789,
                     y=362436069,
                     z=521288629,
                     w=88675123;
unsigned long t;
t=(x^(x<<11));  x=y;  y=z;  z=w;
return( w=(w^(w>>19))^(t^(t>>8)) ); }
```

MWC algoritam:

```
unsigned long mwc(){
static unsigned long x=123456789,
                     y=362436069,
                     z=77465321,
                     c=13579;
unsigned long long t;
t=916905990LL*x+c;  x=y;  y=z;  c=(t>>32);
return z=(t&0xffffffff); }
```

MWC generator slučajnih brojeva generira niz

$$x_n = ax_{n-3} + c \bmod b,$$

uz $a = 916905990, b = 2^{32}$. Algoritam čuva prethodne tri vrijednosti x, y, z i trenutnu vrijednost c, formira $t = ax + c$ u 64 bita, te na kraju pridružuje $x = y$, $y = z$. Nova vrijednost varijable c su gornja 32 bita od t, dok je nova varijabla z donja 32 bita. Period je $(ab^3 - 1)/2 \approx 2^{125}$.

Obje procedure prolaze sve testove iz skupa testova Diehard. Također, obje procedure koriste svega par instrukcija u C-u. xorshift RNG čuva zadnje 4 vrijednosti, dok MWC čuva zadnje tri vrijednosti i zadnju vrijednost varijable c.

Sjeme za xor128 su četiri 32-bitna cijela broja x, y, z, w, koji nisu svi 0, a sjeme za MWC su tri 32-bitna cijela broja x, y, z, uz početni $c < a$, isključujući $x = y = z = c = 0$ i $x = y = z = b - 1, c = a - 1$. Algoritam xor128() je puno brži od algoritma mwc(). Na računalu brzine 1800MHz, poziv xor128() treba 4.4 nanosekunde (više od 220 milijona brojeva po sekundi), dok mwc() treba 21 nanosekundu (48 milijuna brojeva po sekundi).

Ako želimo RNG s jako dugim periodima, to možemo ostvariti pomoću MWC ili CMWC metoda koje ostvaruju period do 2^{131102}, ali zahtijevaju održavanja tablice zadnjih k vrijednosti, s time da je veličina k u tisućama. No, ako nije potreban preveliki period, 2^{160}, 2^{128}, 2^{96} ili manje, onda je dobar jedan od xorshift RNG koji pamti svega par zadnjih vrijednosti.

2.2 Mersenne Twister (MT19937)

2.2.1 Uvod

Mersenne Twister je uniformni generator pseudoslučajnih brojeva objavljen u [5]. Za dobar izbor parametara, algoritam dostiže period veličine $2^{19937} - 1$ i 623-dimenzionalnu ekvidistribuciju do 32-bitne preciznosti, a koristi samo 624 riječi. Ovaj algoritam prolazi nekoliko strogih statističkih testova, uključujući Diehard testove. Njegova brzina je usporediva s ostalim modernim generatorima slučajnih brojeva.

Definicija 2.2.1. *Za niz realnih brojeva $\{s_1, s_2, s_3, \ldots\}$ kažemo da je ekvidistribuiran na intervalu $[a, b]$ ako za svaki podinterval $[c, d] \subset [a, b]$ vrijedi:*

$$\lim_{n\to\infty} \frac{|\{s_1, \ldots, s_n\} \cap [c, d]|}{n} = \frac{d - c}{b - a}.$$

Izraz $|\{s_1, \ldots, s_n\} \cap [c, d]|$ označava broj elemenata niza koji se nalaze u intervalu $[c, d]$.

Definicija 2.2.2. *Za pseudoslučajan niz x_i w-bitnih cijelih brojeva perioda P kažemo je k-distribuiran na v-bitnu preciznost ako vrijedi sljedeće. Neka je* $\mathrm{trunc}_v(x)$ *broj dobiven od vodećih v bitova od x i neka imamo P kv-bitnih vektora*

$$(\mathrm{trunc}_v(x_i), \mathrm{trunc}_v(x_{i+1}), \ldots, \mathrm{trunc}_v(x_{i+k-1})), \quad (0 \le i < P).$$

Tada se svaka od 2^{kv} mogućih kombinacija bitova pojavljuje jednak broj puta u periodu, osim kombinacije kada su sve vrijednosti nula, koja se pojavljuje jedan puta manje.

Za svaki $v = 1, 2, \ldots, w$ s $k(v)$ označimo maksimalan broj tako da je niz $k(v)$-distribuiran na v-bitnu preciznost.

2.2.2 MT algoritam

U ovom dijelu, masna slova poput $\boldsymbol{x}$ i $\boldsymbol{a}$, označavat će vektore računalnih riječi, w-dimenzionalne vektore nad poljem $F_2 = \{0, 1\}$, pri čemu je w je veličina računalne riječi.

MT algoritam generira niz vektora koji su uniformno distribuirani pseudoslučajni brojevi između 0 i $2^w - 1$. Dijeljenjem s $2^w - 1$, svaki vektor postaje realan broj iz intervala $[0, 1]$.

Algoritam je baziran na sljedećoj linearnoj rekurzivnoj relaciji:

$$\boldsymbol{x}_{k+n} := \boldsymbol{x}_{k+m} \oplus (\boldsymbol{x}_k^u | \boldsymbol{x}_{k+1}^l)A, \quad k = 0, 1, \ldots, \tag{2.1}$$

pri čemu cijeli broj n označava stupanj rekurzije, r označava mjesto razdvajanja riječi ($0 \leq r \leq w - 1$), za m vrijedi $1 \leq m \leq n$ i A je matrica dimenzija $w \times w$ s elementima iz F_2. Zadaje se sjeme algoritma $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots \boldsymbol{x}_{n-1}$, a algoritam generira $\boldsymbol{x}_n$ pomoću gornje rekurzije s $k = 0$. Povećavanjem k na $1, 2, \ldots$, generiraju se $\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \ldots$. Na desnoj strani rekurzije, $\boldsymbol{x}_k^u$ znači gornjih $w - r$ bitova od $\boldsymbol{x}_k$, a $\boldsymbol{x}_{k+1}^l$ znači donjih r bitova od $\boldsymbol{x}_{k+1}$. Dakle, ako je $\boldsymbol{x} = (x_{w-1}, x_{w-2}, \ldots, x_0)$, tada je, po definiciji, $\boldsymbol{x}^u$ $(w - r)$-bitni vektor $(x_{w-1}, \ldots, x_r)$, a $\boldsymbol{x}^l$ je r-bitni vektor $(x_{r-1}, \ldots, x_0)$. Stoga $(\boldsymbol{x}_k^u | x_{k+1}^l)$ označava običnu konkatenaciju, vektor veličine w dobiven od $w - r$ gornjih bitova od $\boldsymbol{x}_k$ i r donjih bitova od $\boldsymbol{x}_{k+1}$. Nakon toga, dobiveni vektor pomnožimo matricom A. Za kraj, dobivenom vektoru dodamo vektor $\boldsymbol{x}_{k+m}$ i tako generiramo sljedeći vektor $\boldsymbol{x}_{k+n}$, s tim da $\oplus$ označava zbrajanje po bitovima modulo 2.

Matricu A treba odabrati tako da je množenje s A vrlo brzo. Kandidat za A je matrica:

$$\begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_{w-1} & a_{w-2} & \ldots & \ldots & a_0 \end{pmatrix}.$$

Za ovakvu matricu A, računanje $\boldsymbol{x}A$ možemo ostvariti korištenjem samo bitovnih operacija:

$$\boldsymbol{x}A = \begin{cases} \text{shiftright}(\boldsymbol{x}) & \text{ako je } x_0 = 0, \\ \text{shiftright}(\boldsymbol{x}) \oplus \boldsymbol{a} & \text{ako je } x_0 = 1, \end{cases}$$

pri čemu je

$$\boldsymbol{a} = (a_{w-1}, a_{w-2}, \ldots, a_0)$$

i

$$\boldsymbol{x} = (x_{w-1}, x_{w-2}, \ldots, x_0).$$

Također, $\boldsymbol{x}_k^u$ i $\boldsymbol{x}_{k+1}^l$ iz formule (2.1) mogu biti izračunate pomoću bitovnih AND operacija. Stoga, računanje cijele rekurzije (2.1) može biti realizirano pomoću bitovnog pomaka i bitovnih XOR, OR i AND operacija.

Da bismo poboljšali k-distribuiranost na v-bitnu preciznost, svaku generiranu riječ množimo s $w \times w$ invertibilnom matricom T (Matsumoto i Kurita je nazivaju „tempering matrix“). Za tempering matricu $\boldsymbol{x} \mapsto \boldsymbol{z} = \boldsymbol{x}T$, koristimo sljedeće transformacije:

$$\boldsymbol{y} := \boldsymbol{x} \oplus (\boldsymbol{x} \gg u) \tag{2.2}$$
$$\boldsymbol{y} := \boldsymbol{y} \oplus ((\boldsymbol{y} \ll s) \text{ AND } \boldsymbol{b}) \tag{2.3}$$
$$\boldsymbol{y} := \boldsymbol{y} \oplus ((\boldsymbol{y} \ll t) \text{ AND } \boldsymbol{c}) \tag{2.4}$$
$$\boldsymbol{z} := \boldsymbol{y} \oplus (\boldsymbol{y} \gg l) \tag{2.5}$$

Varijable l, s, t i u su cijeli brojevi, $\boldsymbol{b}$ i $\boldsymbol{c}$ su odgovarajuće bitovne maske veličine riječi, $\boldsymbol{x} \gg u$ označava pomak udesno za u bitova, a $\boldsymbol{x} \ll u$ pomak ulijevo za u bitova.

Za ostvarivanje rekurzije (2.1), dovoljno je uzeti polje od n riječi. Neka je $\boldsymbol{x}[0 : n-1]$ polje od n nenegativnih cijelih brojeva veličine računalne riječi, i cjelobrojna varijabla, te $\boldsymbol{u}, \boldsymbol{ll}$ i $\boldsymbol{a}$ nenegativne cjelobrojne konstante veličine riječi.

0. korak

$\boldsymbol{u} \leftarrow \underbrace{1\ldots1}_{w-r}\underbrace{0\ldots0}_{r}$ (bitovna maska za gornjih $w-r$ bitova)

$\boldsymbol{ll} \leftarrow \underbrace{0\ldots0}_{w-r}\underbrace{1\ldots1}_{r}$ (bitovna maska za donjih r bitova)

$\boldsymbol{a} \leftarrow a_{w-1}a_{w-2}\ldots a_1a_0$ (zadnji redak matrice A)

1. korak

$i \leftarrow 0$

$\boldsymbol{x}[0], \boldsymbol{x}[1], \ldots, \boldsymbol{x}[n-1] \leftarrow$ bilo koje ne-nul početne vrijednosti

2. korak

$\boldsymbol{y} \leftarrow (\boldsymbol{x}[i] \text{ AND } \boldsymbol{u}) \text{ OR } (\boldsymbol{x}[(i+1) \bmod n] \text{ AND } \boldsymbol{ll})$ (računanje $(\boldsymbol{x}_i^u | \boldsymbol{x}_{i+1}^l)$)

3. korak

$\boldsymbol{x}[i] \leftarrow \boldsymbol{x}[(i+m) \bmod n] \text{ XOR } (\boldsymbol{y} \gg 1)$

$$\text{XOR}\begin{cases} 0 & \text{ako je najmanje značajan bit od } \boldsymbol{y} = 0 \\ \boldsymbol{a} & \text{ako je najmanje značajan bit od } \boldsymbol{y} = 1 \end{cases}$$

U ovom koraku, druga operacija XOR predstavlja množenje vektora $\boldsymbol{y}$ matricom A, a prvi XOR je operacija $\oplus$ iz rekurzije (2.1).

4. korak (računanje $\boldsymbol{x}[i]T$)

$\boldsymbol{y} \leftarrow \boldsymbol{x}[i]$
$\boldsymbol{y} \leftarrow \boldsymbol{y}$ XOR $(\boldsymbol{y} \gg u)$
$\boldsymbol{y} \leftarrow \boldsymbol{y}$ XOR $((\boldsymbol{y} \ll s)$ AND $\boldsymbol{b})$
$\boldsymbol{y} \leftarrow \boldsymbol{y}$ XOR $((\boldsymbol{y} \ll t)$ AND $\boldsymbol{c})$
$\boldsymbol{y} \leftarrow \boldsymbol{y}$ XOR $(\boldsymbol{y} \gg l)$
rezultat je $\boldsymbol{y}$

5. korak

$i \leftarrow (i + 1) \bmod n$

6. korak

Vrati se na 2. korak.

U algoritmu postoje dvije vrste parametara:

(1) parametri perioda: cjelobrojni parametri w (veličina računalne riječi), n (stupanj rekurzivnosti), m (srednja riječ), r (mjesto razdvajanja riječi) i vektor parametara $\boldsymbol{a}$ (tj. matrica A);

(2) „tempering“ parametri za k-distribuciju do v-bitne preciznosti: cjelobrojni parametri l, u, s, t i vektorski parametri $\boldsymbol{b}, \boldsymbol{c}$.

Izbor parametara koji donosi najbolje rezultate, tj. algoritam MT19937, je sljedeći:

- $(w, n, m, r) = (32, 624, 397, 31)$
- $a = 9908B0DF_{16}$
- $u = 11$
- $s = 7, b = 9D2C5680_{16}$
- $t = 15, c = EFC60000_{16}$
- $l = 18$.

Poglavlje 3

Testovi slučajnosti

3.1 TestU01

TestU01 [1] je biblioteka softvera implementirana u ANSI C jeziku, koja nudi kolekciju metoda za empirijsko statističko testiranje uniformnih generatora slučajnih brojeva (RNG). Sadrži generalnu implementaciju klasičnih statističkih testova za RNG, kao i neke druge testove opisane u ovom poglavlju, uključujući i neke originalne. Preddefinirani testovi namijenjeni su za nizove uniformnih slučajnih brojeva iz intervala $(0, 1)$, te za nizove bitova.

3.1.1 Uvod

Kao što je već rečeno, brojevi koje generiraju algoritamski generatori su deterministički i periodički. Stoga je jasno da oni ne daju neovisne slučajne brojeve u matematičkom smislu i da oni ne mogu proći sve moguće statističke testove uniformnosti i nezavisnosti. Međutim, neki generatori imaju duge periode i pokazuju se vrlo dobrima na statističkim testovima.

Dobri RNG nisu dizajnirani tako da se isprobava proizvoljan algoritam koji se mijenja na temelju empirijskih testova, sve dok ne prođe sve testove. Dizajn algoritma mora sadržavati detaljnu matematičku analizu perioda i uniformnosti vektora uzastopnih vrijednosti koje algoritam generira. Međutim, nakon što je algoritam implementiran, on mora biti i empirijski testiran. Statistički testovi su, također, potrebni i za generatore pravih slučajnih brojeva (TRNG).

Osim TestU01, jedan od najpoznatijih javnih paketa za statističko testiranje RNGa je Diehard skup testova. Diehard testovi se neće detaljno obrađivati u ovome radu, ali ću dati pregled testova sadržanih u paketu.

3.1.2 Kriterij kvalitete generatora slučajnih brojeva

Generatori slučajnih brojeva za opću primjenu dizajnirani su tako da je njihov izlaz niz koji je dobra imitacija niza neovisnih uniformnih slučajnih varijabli, uglavnom nad realnim intervalom $(0, 1)$ ili nad binarnim skupom $\{0, 1\}$. U prvom slučaju, relevantna hipoteza $\mathcal{H}_0^A$ koju treba testirati kaže da su uzastopne izlazne varijable RNG-a, npr. $u_o, u_1, u_2, \ldots$, neovisne slučajne varijable s uniformnom distribucijom na intervalu $(0, 1)$, tj. $U(0, 1)$. U drugom slučaju, hipoteza $\mathcal{H}_0^B$ kaže da imamo niz neovisnih slučajnih bitova, od kojih svaki bit poprima vrijednost 0 ili 1 s jednakim vjerojatnostima i neovisno jedan o drugome.

Ova dva slučaja su jako povezana, jer svaki unaprijed zadan niz bitova (npr. niz bitova formiran uzastopnim bitovima od u_0, ili formiran koristeći svaki drugi bit, ili po prvih 5 bitova svakog u_i, itd.) mora biti niz neovisnih slučajnih bitova. Tako da statistički testovi za nizove bitova mogu biti indirektno korišteni za testiranje nul-hipoteze $\mathcal{H}_0^A$.

U slučaju $U(0, 1)$, hipoteza $\mathcal{H}_0^A$ je ekvivalentna tome da je za svaki cijeli broj $t > 0$, vektor $(u_0, \ldots, u_{t-1})$ uniformno distribuiran nad t-dimenzionalnom jediničnom kockom $(0, 1)^t$. To ne može biti ispunjeno za algoritamske RNG, jer ti vektori poprimaju svoje vrijednosti iz konačnog skupa ψ_t svih t-dimenzionalnih vektora od t uzastopnih vrijednosti koji taj generator daje, za sve moguće izbore sjemena. Kardinalnost ovoga skupa ne može prijeći broj dopustivih sjemena za RNG. Pretpostavljajući da je sjeme odabrano nasumično, vektori su generirani u ψ_t aproksimirajući uniformnu distribuciju na $(0, 1)^t$. Ovo sugerira da ψ_t mora biti ravnomjerno distribuiran na jediničnoj kocki.

Za niz bitova, nul-hipoteza $\mathcal{H}_0^B$ ne može biti formalno istinita čim duljina niza, t, prelazi broj bitova, b, generatorovog stanja, jer broj različitih nizova bitova koje generator izbacuje ne može prijeći 2^b. Za $t > b$, dio od ukupno 2^t nizova od t bitova koji možemo posjetiti je najviše 2^{b-t}. Cilj je biti siguran da su nizovi koje možemo posjetiti uniformno porazbacani u skupu svih 2^t mogućih nizova.

Drugačiji kriteriji koriste se za RNG u kriptografske i slične svrhe. Za te svrhe, bitna je i nepredvidljivost brojeva koji se generiraju.

3.1.3 Statističko testiranje

Statistički test za RNG je definiran testnom statistikom Y, koja je funkcija konačnog broja izlaznih realnih brojeva u_n (ili konačnog broja bitova, ako se radi o generatorima bitova), a čija je distribucija pod hipotezom $\mathcal{H}_0$ poznata ili može biti dobro aproksimirana ($\mathcal{H}_0$ može biti $\mathcal{H}_0^A$ ili $\mathcal{H}_0^B$). Broj različitih testova je praktički beskonačan i svaki test detektira drugačiji tip problema s RNG-om. Niti jedan univerzalan test, ili skup testova, ne može garantirati da je generator koji prođe test u potpunosti pouzdan, za sve vrste simulacija. Statistički testovi ne mogu dokazati da je RNG bez greške, ali oni mogu povećati naše povjerenje u njega. Sa sigurnošću možemo reći da niti jedan RNG ne može uspješno proći sve statističke testove. U suštini, razlika između dobrih i loših generatora slučajnih brojeva

je ta da loši generatori padaju na vrlo jednostavnim testovima, dok dobri generatori padnu samo na vrlo složenim testovima.

Idealno bi bilo, za testiranje generatora za neku simulaciju, da Y imitira slučajnu varijablu koja odgovara problemu, tako da na testu možemo lako uočiti loš odnos između RNG-a i simulacije problema. Međutim, to nije baš praktično, jer ne može biti primijenjeno za testiranje generatora koji će se koristiti u softverskim paketima šire namjene. Iskustvo s empirijskim testovima je pokazalo da generatori s jako dugim periodima, dobrom strukturom njihovog skupa ψ_t i koji su bazirani na rekurzijama koje nisu jednostavne, dobro prolaze zahtjevnije testove. S druge strane, generatori s kratkim periodima ili lošom strukturom padaju na standardnim statističkim testovima. Glavni razlog statističke slabosti je jednostavna struktura nekih generatora koja omogućava brzo generiranje brojeva. Potrebni su praktični alati za otkrivanje svih tih grešaka, a paket TestU01 sadrži mnoge empirijske testove upravo za te svrhe.

Klasični statistički udžbenici uglavnom govore da za testiranje hipoteze moramo odrediti područje odbijanja R čija vjerojatnost pod $\mathcal{H}_0$ odgovara nivou testiranja (npr. 0.05 ili 0.01), te odbijamo hipotezu $\mathcal{H}_0$ ako i samo ako $Y \in R$. Ova procedura je pogodna kada imamo malu fiksiranu veličinu uzorka, ali pri testiranju RNG veličina uzorka je jako velika i može se po potrebi i povećavati. Stoga, umjesto odabira nivoa testiranja i područja odbijanja, jednostavno računamo p-vrijednost testa definiranu s:

$$p = P[Y \geq y \mid \mathcal{H}_0]$$

Ako Y ima neprekidnu distribuciju, tada je p $U(0, 1)$ slučajna varijabla pod hipotezom $\mathcal{H}_0$. Za neke testove, na p možemo gledati kao na mjeru uniformnosti, u smislu da će vrijednost biti blizu 1 ako generator daje vrijednosti pretjerane uniformnosti, odnosno blizu 0 u suprotnoj situaciji.

Ako je p-vrijednost ekstremno mala (npr. manja od 10^{-10}), očito je da RNG pada na testu. S druge strane, ako vrijednost nije blizu 0 ili 1, test nije detektirao nikakav problem. Ako je p-vrijednost sumnjiva, ali ne kaže jasno da mora biti odbijena (npr. $p = 0.002$), tada se test može ponoviti više puta na disjunktnim izlaznim nizovima istog generatora, sve dok pogreška ne postane, ili očita, ili dok se sumnja ne otkloni. Ovaj pristup je moguć zato jer nema ograničenja na duljinu podataka koje generira RNG. Kada se primjenjuje nekoliko testova na zadanom generatoru, znaju se dogoditi p-vrijednosti manje od 0.01 ili veće od 0.99, iako RNG radi ispravno po testu (te vrijednosti se pojavljuju u 2% vremena). U tom slučaju, sumnjive vrijednosti se ne pojavljuju sistematično, osim ako stvarno nemamo sreće. Pad na testu uglavnom ovisi o strukturi skupa ψ_t, a rijetko o tome koji dio cijelog izlaznog niza se testira. Štoviše, kada generator odlučno pada na testu, p-vrijednost testa uglavnom konvergira prema 0 ili 1 eksponencijalno brzo, kao funkcija veličine testnog uzorka (veličina testnog uzorka se povećava). Stoga, sumnjive p-vrijednosti mogu biti odlučene povećavanjem uzorka kojeg se testira.

U slučaju kada Y ima diskretnu distribuciju pod hipotezom $\mathcal{H}_0$, treba biti pažljiv pri definiranju p-vrijednosti. U tom slučaju razlikuje se desna p-vrijednost

$$p_R = P[Y \geq y \mid \mathcal{H}_0]$$

i lijeva p-vrijednost

$$p_L = P[Y \leq y \mid \mathcal{H}_0].$$

Hipotezu $\mathcal{H}_0$ se odbija kada je jedna od ove dvije vrijednosti blizu 0.

Nekoliko autora se zalagalo za primjene testa na dva nivoa za testiranje RNG-a, (vidjeti [9, 13, 14, 16]). Ideja je generirati N nezavisnih kopija od Y, nazovimo ih $Y_1, \ldots, Y_N$, primjenjujući prvotni test N puta na disjunktnim podnizovima izlaza generatora. Neka je F teorijska funkcija distribucije od Y, pod hipotezom $\mathcal{H}_0$. Ako je F neprekidna, tada su transformacije $U_1 = F(Y_1), \ldots, U_N = F(Y_N)$ nezavisne i identično distribuirane $U(0,1)$ slučajne varijable pod hipotezom $\mathcal{H}_0$. Jedan način izvođenja testa na dva nivoa je usporediti empirijske distribucije dobivenih U_j s uniformnim distribucijama, pomoću tzv. *goodness-of-fit* (GOF) testa, poput Kolmogorov–Smirnovljeva testa, Anderson–Darlingova testa i slično.

Ako su $U_{(1)}, \ldots, U_{(N)}$ poredane u rastućem poretku, tada su Kolmogorov–Smirnovljeve (KS) testne statistike D_N^+, D_N^- i D_N definirane s:

$$\begin{aligned} D_N^+ &= \max_{1 \leq j \leq N} (j/N - U_{(j)}), \\ D_N^- &= \max_{1 \leq j \leq N} (U_{(j)} - (j-1)/N), \\ D_N &= \max(D_N^+, D_N^+), \end{aligned}$$

a Anderson–Darlingova (AD) testna statistika je:

$$A_N^2 = -N - \frac{1}{N} \sum_{j=1}^{N} \left\{ (2j-1) \ln(U_{(j)}) + (2N + 1 - 2j) \ln(1 - U_{(j)}) \right\}.$$

Kao i dosad, izračuna se p-vrijednost GOF testne statistike i hipoteza $\mathcal{H}_0$ se odbacuje ako je p-vrijednost previše ekstremna.

Postoji i mnogo testova koji moguće ishode particioniraju u konačan niz kategorija. Test neovisno generira n takvih ishoda, prebroji se koliko ishoda upada u koju od kategorija i primijeni χ^2 test za usporedbu dobivenih vrijednosti s očekivanim brojem za svaku kategoriju pod hipotezom $\mathcal{H}_0$.

Također, postoji beskonačno testova uniformnosti i nezavisnosti za RNG. Odabir testova koji će se koristiti je, uglavnom, subjektivna odluka. Međutim, neki od izbora su vrlo prirodni, a dio testova je motiviran važnim područjima primjene generatora.

U nastavku slijedi pregled glavnih empirijskih testova korištenih i implementiranih u TestU01. Podijeljeni su u dvije glavne kategorije: testovi koji testiraju hipotezu $\mathcal{H}_0^A$ za nizove realnih brojeva u intervalu $(0, 1)$ i testovi koji testiraju hipotezu $\mathcal{H}_0^B$ za nizove bitova. Svaka kategorija je dalje podijeljena na podkategorije.

3.1.4 Testovi za nizove realnih brojeva u $(0, 1)$

3.1.4.1 Testovi na nizu duljine n

Mjerenje globalne uniformnosti. Za niz $u_1, u_2, \ldots, u_n$ iz intervala $(0, 1)$ želimo testirati hipotezu $\mathcal{H}_0^A$. Prva ideja je izračunati empirijsku distribuciju niza $u_1, u_2, \ldots, u_n$ te usporediti je s $U(0, 1)$ distribucijom, koristeći Kolmogorov–Smirnovljev (KS) ili neki slični test. Još jednostavnije je izračunati srednju vrijednost i varijancu uzorka, te usporediti ih s teorijskim vrijednostima 1/2 i 1/12. Samo će jako loši RNG pasti na testovima koji mjere uniformnost na globalnom nivou. Testovi na dva nivoa s velikim N i malim n mogu otkriti mogućnost da generator na neko vrijeme zaglavi u području gdje je srednja vrijednost i varijanca veća, odnosno manja od prosječne.

Mjerenje grupiranja. Druga klasa testova mjeri grupiranje brojeva $u_1, u_2, \ldots, u_n$. Brojeve poredamo rastuće i računamo preklapajuće m-razmake

$$g_{m,i} = u_{(i+m)} - u_{(i)}$$

između članova sortiranog niza $u_{(1)}, u_{(2)}, \ldots, u_{(n)}$, pri čemu je $u_{(0)} = 0$ i

$$u_{(n+i)} = 1 + u_{(i-1)}$$

za $i > 0$. Testna statistika ima opću formu

$$H_{m,n}^{(c)} = \sum_{i=0}^{n} h(n g_{m,i}),$$

gdje je h neka glatka funkcija, npr. $h(x) = x^2$ ili $h(x) = \log(x)$. Ovako testiranje empirijske distribucije brojeva u_i je drugačije nego KS test, a testiranjem se pokušava utvrditi jesu li brojevi grupiraniji nego bi trebali biti. Na primjer, ako su brojevi grupirani u grupe od 3 do 5 vrijednosti koje su jako blizu jedna drugoj, KS test to neće otkriti, ali 3-razmak test hoće.

***Run* i *gap* testovi.** *Run* i *gap* testovi pobliže gledaju lokalnu anatomiju niza, pokušavajući detektirati ponavljajuće uzorke. Za *gap* test, odabiremo skup $A \in (0, 1)$ s Lebesgueovom mjerom p (najčešće je $A = [a, b]$ interval, pa je $p = b - a$). Test broji broj koraka (razmak, tzv. *gap size*) između dva uzastopna posjeta skupu A. Neka je X_j broj razmaka veličine

j, za $j \geq 0$. Test uspoređuje frekvencije $X_0, X_1, \ldots$ s njihovim očekivanim vrijednostima pod $\mathcal{H}_0^A$, koristeći χ^2 test. Generator čiji posjeti skupu A su grupirani u vremenu (npr. generator ulazi i izlazi iz A neko vrijeme, pa zatim odlazi iz A na dugi period, itd.) padaju na ovom testu. *Run* test provjera drugačiji tip uzoraka; on skuplja duljine svih rastućih (ili padajućih) podnizova, broji koliko ih ima za svaku duljinu podniza, te računa modificiranu χ^2 statistiku baziranu na tim brojačima.

3.1.4.2 Testovi bazirani na n podnizova duljine t

Particioniranje jedinične hiperkocke i brojanje pogodaka po ćeliji. Prisjetimo se da za RNG koji daje niz realnih brojeva u intervalu $(0, 1)$, hipoteza $\mathcal{H}_0^A$ je ekvivalentna tome da za sve $n \geq 0$ i $t > 0$, vektor izlaznih vrijednosti $(u_n, \ldots, u_{n+t-1})$ ima uniformnu distribuciju na t-dimenzionalnoj jediničnoj hiperkocki $(0, 1)^t$. Prirodni pristup testiranju uniformnosti je particionirati jediničnu hiperkocku $(0, 1)^t$ na k dijelova (ćelija) volumena $p_0, \ldots, p_{k-1}$. Najčešće je

$$p_0 = \ldots = p_{k-1} = \frac{1}{k},$$

ali ne uvijek. Zatim se generira n točaka

$$u_i = (u_{ti}, \ldots, u_{ti+t-1}) \in (0, 1)^t,$$

za $i = 0, \ldots, n-1$, i izračuna broj točaka X_j koje su upale u ćeliju j, za $j = 0, \ldots, k-1$. Pod $\mathcal{H}_0^A$, vektor $(X_0, \ldots, X_{k-1})$ ima multinomnu distribuciju s parametrima $(n, p_0, \ldots, p_{k-1})$. Bilo koja mjera udaljenosti između vrijednosti X_j i njihovih očekivanja $\lambda_j = np_j$ može poslužiti kao testna statistika Y. Opisat ćemo posebne slučajeve i varijante testova ovoga tipa.

Serijski testovi. Jednostavan način za particioniranje hiperkocke $(0, 1)^t$ na $k = d^t$ ćelija volumena $1/k$ je podjela intervala $(0, 1)$ na d jednakih dijelova, za neki cijeli broj $d > 1$. Tada je $p_j = 1/k$, za sve j. Testovi koji mjere ukupan raskorak između zbrojeva X_j i njihovih očekivanja $\lambda = n/k$ se općenito nazivaju serijski testovi uniformnosti.

Pri originalnom serijskom testu, udaljenost je mjerena pomoću χ^2 testne statistike

$$X^2 = \sum_{j=0}^{k-1} \frac{(X_j - \lambda)^2}{\lambda},$$

koja ima približno χ^2 distribuciju s $k-1$ stupnjeva slobode kada je n/k dovoljno velik.

Generalno, možemo koristiti statistiku oblika

$$Y = \sum_{j=0}^{k-1} f_{n,k}(X_j),$$

pri čemu je $f_{n,k}$ realna funkcija ovisna o n i k. Ove statistike imaju asimptotski χ^2 distribuciju s $k-1$ stupnjeva slobode kada $n \to \infty$, za fiksirani k, pod hipotezom $\mathcal{H}_0^A$.

Kolizija, prazne ćelije i slično. TestU01 podržava različite odabire za $f_{n,k}$. Na primjer, broj ćelija N_b koje sadrže točno b točaka (za $b \geq 0$), broj ćelija W_b koji sadrže barem b točaka (za $b \geq 1$) i broj kolizija C, tj. broj koji kaže koliko puta je točka upala u ćeliju koja već ima jednu ili više točaka. Ove statistike su povezane s

$$N_0 = k - W_1 = k - n + C, \quad W_b = N_b + \ldots + N_n \quad \text{i} \quad C = W_2 + \ldots + W_n.$$

Njih je pobliže proučavao i uspoređivao L'Ecuyer i dao je dobre aproksimacije njihovih distribucija pod hipotezom $\mathcal{H}_0^A$.

U svakom od slučajeva, testna statistika je mjera grupiranosti – smanjuje se kada su točke podjednako distribuirane između ćelija. Broj kolizija C često je najsnažnija testna statistika među spomenutima, za dobru detekciju tipičnih problema u RNG, prvenstveno zbog toga što dopušta $k \gg n$.

Preklapajuće verzije. Postoje i preklapajuće verzije serijskih testova, pri kojima su točke definirane s $\boldsymbol{u}_i = (u_i, \ldots, u_{i+t-1})$ za $i = 0, \ldots, n-1$. Marsaglia i Zaman u [18] preklapajuće serijske testove nazivaju „monkey tests" – oni gledaju na generator kao na majmuna koji tipka nasumične znakove iz abecede od d znakova. Test broji koliko se puta svaka riječ od t znakova pojavljuje u nizu kojeg je napisao majmun. Oni koriste statistiku N_0 i nazivaju odgovarajući test OPSO za $t = 2$, OTSO za $t = 3$ i OQSO za $t = 4$. Kada je $d = 4$ i t oko 10, test nazivaju DNA test.

Druge metode particioniranja jedinične hiperkocke. Osim podjela jedinične hiperkocke $(0,1)^t$ na $k = d^t$ ćelija jednakih veličina, kao pri serijskim testovima, postoje i drugi načini particioniranja hiperkocke na k dijelova (ćelija) i mapiranja vektora brojevima ćelija. Drugi bi način bio pronaći koja od $t!$ permutacija od t objekata će presložiti koordinate vektora $\boldsymbol{u}_i$ u rastućem poretku. Broj mogućih permutacija označimo s $k = t!$ i neka je X_j broj vektora $\boldsymbol{u}_i$ koji su presloženi permutacijom j, za $j = 0, \ldots, k-1$. Vektor $(X_0, \ldots, X_{k-1})$ ponovno ima multinomnu distribuciju s parametrima $(n, 1/k, \ldots, 1/k)$ i sve testne statistike Y definirane ranije se mogu primijeniti. Ovo okruženje pogodno je za testiranje RNG koji će se koristiti za miješanje špila karata za računalne igre ili kockarske aparate. Tada, kolizijski test primijenjen na permutacije, može detektirati ako se određena permutacija karata pojavljuje češće nego ostale. Knuth u [13] preporuča korištenje χ^2 statistike za testiranje jesu li sve permutacije jednako vjerojatne, ali broj kolizija C je, uglavnom, osjetljiviji i dozvoljava testove s većim t.

Test *argmax* koristi drugačije mapiranje. On definira X_j kao broj vektora $\boldsymbol{u}_i$ čija je najveća koordinata na j-tom mjestu. Na ovaj način testira se je li lokacija maksimuma podjednako distribuirana među koordinatama.

Alternative prebrojavanju

Fokusiranje na jednu ćeliju. CAT test je varijacija kolizijskog testa pri kojem se kolizije broje samo u jednoj ćeliji. U originalnoj verziji, ćelije su određene kao u preklapajućim serijskim testovima, ali princip funkcionira i bez preklapanja, kao i za proizvoljnu particiju. Pod hipotezom $\mathcal{H}_0^A$, za veliki n, broj točaka Y u ciljanoj ćeliji ima približno Poissonovu distribuciju sa srednjom vrijednosti λ, jednakoj broju generiranih točaka pomnoženom s volumenom ciljane ćelije, uz pretpostavku da su točke u parovima asimptotski neovisne kada $n \to \infty$. Kada su točke neovisne, Y ima binomnu distribuciju. Ovaj test može biti jak samo ako se ciljana ćelija često posjećuje ili ako je λ dovoljno velik, a ciljana ćelija je rijetko posjećena, zbog posebne slabosti generatora.

Matsumoto i Kurita su u [20] predložili vrlo sličan test s $t = 1$, ali na dva nivoa: treba generirati n_0 brojeva u intervalu $(0, 1)$ i prebrojiti koliko ih upada u dani interval $[\alpha, \beta]$, nazovimo ga Y. Postupak treba ponoviti n puta i usporediti distribucije n realizacija za Y s binomnom distribucijom s parametrima n_0 i $p = \beta - \alpha$, koristeći χ^2 test.

Vremenski razmaci između posjeta ćelijama. Umjesto prebrojavanja posjeta svakoj ćeliji u particiji, mogu se ispitati detaljnije informacije. Na primjer, može se gledati razmak (broj koraka) između dva uzastopna posjeta istoj ćeliji, raditi to za svaku ćeliju i na kraju izračunati odgovarajuću funkciju svih tih razmaka kao testnu statistiku. Ovo kombinira *gap* test i multinomni test baziran na particiji hiperkocke. Maurer [21] je uveo poseban slučaj ovog tipa testa i predlažio srednju vrijednost logaritama svih razmaka kao testnu statistiku.

***Birthday spacings*.** Interesantna verzija serijskih testova je tzv. *birthday spacing* test, opisan na sljedeći način. Generira se n točaka u k ćelija, kao i za serijski test. Neka su $I_1 \leq I_2 \leq \ldots \leq I_n$ brojevi ćelija u koje upadaju generirane točke, sortirani u rastućem poretku. Test računa razmake $I_{j+1} - I_j$, za $1 \leq j < n$, i prebrojava broj kolizija Y između tih razmaka. Pod hipotezom $\mathcal{H}_0^A$, Y je približno Poissonova slučajna varijabla sa srednjom vrijednosti $\lambda = n^3/(4k)$. Ako se test ponovi N puta, računa se suma N vrijednosti Y (ukupan broj kolizija) i uspoređuje se s Poissonovom slučajnom varijablom sa srednjom vrijednosti $N\lambda$, da bi se dobila p-vrijednost. Ova procedura je, uglavnom, osjetljivija od primjene χ^2 testa za brojanje kolizija.

Opravdanje za ovaj test je da neki tipovi RNG generiraju točke u ćelije s pravilnim razmacima. To se posebno događa za sve LCG (linearne kongurentne generatore) kada je $t \geq 2$.

Bliski parovi točaka u prostoru. Ponovno, n točaka $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ razbacuje se neovisno i uniformno po jediničnoj hiperkocki, ali sada se gledaju udaljenosti između točaka. Udaljenost se mjeri u $\mathcal{L}_p$ normi $\|.\|_p^o$ na jediničnom torusu $(0,1)^t$, dobivenom identificiranjem (u parovima) suprotnih strana jedinične hiperkocke, tako da su točke na suprotnim stranama blizu jedna drugoj.

Udaljenost između dvije točke $\boldsymbol{u}_i$ i $\boldsymbol{u}_j$ je definirana s $D_{n,i,j} = \|\boldsymbol{u}_i - \boldsymbol{u}_j\|_p^o$, pri čemu je:

$$\|\boldsymbol{x}\|_p^o = \begin{cases} [\min(|x_1|, 1-|x_1|)^p + \cdots + \min(|x_t|, 1-|x_t|)^p]^{1/p} & \text{ako je } 1 \leq p < \infty, \\ \max(\min(|x_1|, 1-|x_1|), \ldots, \min(|x_t|, 1-|x_t|)) & \text{ako je } p = \infty, \end{cases}$$

za $\boldsymbol{x} = (x_1, \ldots, x_t) \in [-1, 1]^t$. Razlog korištenja torusa je da bi se riješio problem graničnog efekta – lakše je dobiti dobru aproksimaciju relevantne distribucije pod hipotezom $\mathcal{H}_0^A$.

3.1.4.3 Testovi koji generiraju n podnizova slučajne duljine

Testovi u ovoj kategoriji stvaraju podnizove generirajući brojeve u_j sve dok se neki događaj ne dogodi, a potreban broj brojeva u_j je slučajan. Na primjer, u tzv. *coupon collector* testu particionira se interval $(0, 1)$ na d dijelova jednakih veličina i broji koliko slučajnih brojeva u_j treba generirati da bi se dobio barem po jedan u svakoj particiji. Računaju se frekvencije različitih brojača i uspoređuju se s teorijskim frekvencijama pomoću χ^2 testa. Ovo može biti ostvareno i s proizvoljnom particijom od $(0, 1)^t$ na k dijelova jednakih volumena.

3.1.5 Testovi za nizove slučajnih bitova

U ovom dijelu, pretpostavljamo da generator izbacuje bitove $b_0, b_1, b_2, \ldots$, te želimo testirati nul-hipotezu koja kaže da su bitovi b_i neovisni i da poprimaju vrijednosti 0 ili 1 s jednakom vjerojatnošću. Ovi bitovi mogu doći iz bilo kojeg izvora. Posebno, oni mogu biti izvučeni iz niza realnih brojeva u intervalu $(0, 1)$, uzimajući dio bitova svakog broja. Standardna procedura pri TestU01 je uzeti bitove $r + 1, \ldots, r + s$ iz svakog broja, tj. preskočiti prvih r bitova, te uzeti sljedećih s bitova, za neke cijele brojeve $r \geq 0$ i $s \geq 1$. Izvučeni bitovi spajaju se u dugi niz. Vrijednosti r i s su parametri testa.

3.1.5.1 Jedan dugi niz bitova duljine n

Testovi na binarnim nizovima su prvenstveno dizajnirani za područje kriptografije, gdje su visoka entropija i složenost ključni zahtjevi. Maurerov test, spomenut ranije, je entropij-

ski test. Isto tako, binarne verzije multinomnih testova u osnovi spadaju među entropijske testove.

Linearna složenost. Jedan od načina za testiranje složenosti je ispitati kako se linearna složenost L_l prvih l bitova niza povećava, kao funkcija varijable l. Linearna složenost L_l je definirana kao najmanji stupanj linearne rekurzije koju zadovoljava niz. L_l je neopadajuća funkcija od l i povećava se u cjelobrojnim skokovima za određene vrijednosti od l. TestU01 računa L_l Berlekamp–Masseyevim algoritmom (vidjeti [6, 19]) u vremenskoj složenosti $O(n^2 \log n)$. Ako cijeli binarni niz slijedi linearnu rekurziju stupnja $k \ll n$, tada će se L_l prestati povećavati kada je $l = k$, te će test pasti.

Proučavana su dva načina testiranja razvoja L_l. *Jump complexity* test računa broj J skokova linearne složenosti. Pod hipotezom $\mathcal{H}_0^B$ i za veliki n, J je otprilike normalno distribuirana sa srednjom vrijednosti i varijancom opisanom u [7, 22, 24]. *Jump size* test računa koliko ima skokova svakog raspona i uspoređuje te frekvencije s teorijskom distribucijom (geometrijska distribucija s parametrom 1/2) pomoću χ^2 testa.

Lempel–Ziv složenost. Složenost, također, može biti testirana računanjem broja W različitih uzoraka bitova koji se pojavljuju u stringu. Taj broj mjeri stlačljivost niza po Lempel–Ziv kompresijskom algoritmu [25]. Prema Kirschenhoferu i drugima [12], pod hipotezom $\mathcal{H}_0^B$ i za velike n, W je približno normalno distribuirana sa srednjom vrijednosti $n/\log_2 n$ i varijancom $0.266n/(\log_2 n)^3$. Međutim, ove aproksimacije srednje vrijednosti i varijance nisu previše precizne ni za n veličine 2^{24}. TestU01 implementacija koristi bolje aproksimacije, dobivene simulacijom i provjerene različitim tipovima pouzdanih RNG-ova. Ovaj test nije previše osjetljiv, generatori koji padnu na ovom testu, padaju i na mnogim drugim testovima.

Autokorelacije. Autokorelacija ili serijalna korelacija je naziv za korelaciju slučajnih varijabli unutar jednog stohastičkog procesa. Uzorak autokorelacije s kašnjenjem l u nizu bitova, definiran s

$$A_l = \sum_{i=0}^{n-l-1} b_i \oplus b_{i+l},$$

pri čemu $\oplus$ označava xor operaciju (ili zbrajanje modulo 2), definira zanimljiv statistički test [8]. Pod hipotezom $\mathcal{H}_0^B$, A_l ima binomnu distribuciju s parametrima $(n - l, 1/2)$, koja je približno normalna kada je $n - l$ velik.

***Run* i *gap* testovi.** Binarni verzija *run* testa može biti definirana na sljedeći način. Svaki binarni niz ima blok jedinica, pa blok nula, pa blok jedinica i tako dalje (analogno vrijedi ako niz započinje nulom). Pamtimo duljine svih blokova jedinica i svih blokova nula, dok

ne skupimo ukupno $2n$ blokova (n svakoga tipa). Potrebna duljina niza za dobivanje $2n$ blokova je slučajna. Računamo broj svih blokova jedinica duljine j i broj svih blokova nula duljine j, za $j = 1, \ldots, k$, za neki cijeli broj k i primijenimo χ^2 test na ovih $2k$ vrijednosti. Vjerojatnost da bilo koji blok ima duljinu j je 2^{-j}, tako da unaprijed znamo očekivani broj blokova svake duljine. Također, blokove nula možemo promatrati kao razmake između pojavljivanja jedinica (i obratno). Taj test je verzija *gap* testa za binarne nizove.

3.1.5.2 Testovi na n nizova bitova duljine m

Particioniranje skupa m-bitnih nizova i brojanje pogodaka u podskupovima. Sada razmatramo testove koji pokušavaju otkriti zavisnosti u nizovima bitova duljine m, u smislu jesu li neke od 2^m različitih mogućnosti više vjerojatne od drugih. Svi ovi testovi u osnovi (indirektno) sadrže sljedeći obrazac: treba regrupirati 2^m mogućnosti za sve nizove bitova u, recimo, k kategorija, prebrojiti koliko od n nizova upada u svaku kategoriju, te usporediti rezultate s očekivanjima.

Serijski testovi. Verzija serijskog testa za nizove bitova radi na sljedeći način: treba označiti moguće m-bitne nizove od 0 do $k - 1 = 2^m - 1$ i neka je X_j broj pojavljivanja niza j među n nizova. Može se koristiti isti skup testnih statistika kao i ranije (χ^2, entropija, kolizije, itd.). Za preklapajuću verziju, n bitova stavlja se u krug i svaki blok od m uzastopnih nizova na krugu određuje jednu ćeliju. U ovom slučaju, test treba samo n bitova. Teorijska distribucija je približno ista kao i za običan m-dimenzionalni preklapajući test sa $d = 2$ i $t = m$.

CAT test i testovi koji pamte razmake između posjeta stanjima, također, imaju svoje binarne verzije, pri kojima su ćelije zamijenjene m-bitnim nizovima, konstruirane sa ili bez preklapanja. Maurerov test, spomenut ranije, definiran je upravo za opisano okruženje. Maurer je dokazao da je njegov test univerzalan, u smislu da može detektirati bilo koji tip statističkog defekta na binarnim nizovima, kada testni parametri i veličina uzorka na prikladan način teže u beskonačnost. Međutim, test je u praksi uglavnom manje osjetljiv od kolizijskog testa na usporedivoj veličini uzorka.

Rang binarne matrice. Snažan test za detektiranje linearnih zavisnosti između blokova bitova je matrični rang test (vidjeti [16, 17]): treba napuniti binarnu matricu $k \times l$, red po red, nizom bitova duljine $m = k \times l$, te izračunati rang R dobivene matrice (broj linearno nezavisnih redaka). Postupak se ponavlja n puta i uspoređuje se empirijska distribucija n realizacija od R s njezinom teorijskom distribucijom pod hipotezom $\mathcal{H}_0^B$, pomoću χ^2 testa. Za dovoljno velik m, nizovi bitova koji slijede linearnu rekurziju past će na ovom testu, upravo zbog linearnih zavisnosti.

Najdulji tok jedinica. Test nazvan *longest head run* test [10, 11] je varijanta *run* testa koji gleda duljinu Y najduljeg podniza uzastopnih jedinica u nizu duljine m. Postupak se ponavlja n puta i empirijska distribucija n realizacija od Y se uspoređuje s teorijskom distribucijom pomoću χ^2 testa.

Hammingove težine. Za mjerenje grupiranja nula i jedinica u nizu bitova, može se ispitati distribucija Hammingovih težina disjunktnih podnizova duljine m. Općenito, Hammingova težina niza je broj elemenata niza koji su različiti od nul simbola korištene abecede. Test Hammingovih težina u TestU01 generira n nepreklapajućih blokova od m bitova i računa Hammingovu težinu svakog bloka, nazovimo je H_i, za blok i. Pod hipotezom $\mathcal{H}_0^B$, vrijednosti H_i su nezavisne i identično distribuirane binomne slučajne varijable s parametrima $(m, 1/2)$. Neka je X_j broj blokova koji imaju Hammingovu težinu j, za $j = 0, \ldots, m$. Prvi test uspoređuje distribucije od X_j s njihovim teorijskim distribucijama. Drugi test (vidjeti [23]) računa χ^2 statistiku

$$X^2 = (4/m) \sum_{i=1}^{n} (H_i - m/2)^2,$$

koja, ako je m dovoljno velik, približno ima χ^2 distribuciju s n stupnjeva slobode pod hipotezom $\mathcal{H}_0^B$. Za $m = n$, ovaj test se degenerira do monobitnog testa [23], koji jednostavno broji proporciju jedinica u nizu od n bitova. Treći test računa linearnu korelaciju između uzastopnih H_i:

$$\hat{p}_1 = \frac{4}{(n-1)m} \sum_{i=1}^{n-1} (H_i - m/2)(H_{i+1} - m/2).$$

Pod hipotezom $\mathcal{H}_0^B$ i za veliki n, $\hat{p}_1 \sqrt{n-1}$ ima približno standardnu normalnu distribuciju. Ovaj način testira samo linearnu ovisnost između Hammingovih težina. L'Ecuyer i Simard u [15] predlažu drugačiji test neovisnosti koji koristi $2n$ blokova veličine m bitova. Ponovno, neka je H_i Hammingova težina bloka i. Parovi (H_i, H_{i+1}), za $i = 1, 3, \ldots, 2n-1$, mogu poprimiti $(m+1)^2$ mogućih vrijednosti. Ovaj test računa broj ponavljanja svake od mogućnosti i uspoređuje te brojeve s njihovim teorijskim očekivanjima pomoću χ^2 testa.

Test slučajne šetnje. Iz niza bitova duljine l definira se slučajna šetnja po cijelim brojevima na sljedeći način. Šetnja počinje na 0, a u koraku j ide za jedan ulijevo ako je $b_j = 0$, odnosno, za jedan udesno ako je $b_j = 1$. Ako definiramo $S_0 = 0$ i $S_k = \sum_{j=1}^{k} (2b_j - 1)$ za $k > 0$, tada je postupak $\{S_k, k \geq 0\}$ slučajna šetnja. Pod hipotezom $\mathcal{H}_0^B$, iz binomne distribucije dobivamo

$$p_{k,y} := P[S_k = y] = 2^{-k} \binom{k}{(k+y)/2} \quad \text{ako je } k + y \text{ paran broj}$$

i $p_{k,y} = 0$ inače. U nastavku pretpostavljamo da je l paran i definiramo statistike:

$$\begin{aligned}
H &= l/2 + S_l/2, \\
M &= \max\{S_k, 0 \le k \le l\}, \\
J &= 2\sum_{k=1}^{l/2} \mathbb{I}[S_{2k-1} > 0], \\
P_y &= \min\{k : S_k = y\} \quad \text{za } y > 0, \\
R &= \sum_{k=1}^{l} \mathbb{I}[S_k = 0], \\
C &= \sum_{k=3}^{l} \mathbb{I}[S_{k-2} S_k < 0],
\end{aligned}$$

pri čemu $\mathbb{I}$ označava karakterističnu funkciju. H je broj koraka udesno, M je maksimalna vrijednost dosegnuta šetnjom, J je dio vremena proveden na desnoj strani od početne, P_y je vrijeme prvog prolaska kroz y, R je broj povratka na 0 i C je broj promjena predznaka. Teorijske vjerojatnosti ovih statistika pod hipotezom $\mathcal{H}_0^B$ su sljedeće:

$$\begin{aligned}
P[H = k] &= P[S_l = 2k - l] = p_{l,2k-l} = 2^{-l}\binom{l}{k}, \quad 0 \le k \le l, \\
P[M = y] &= p_{l,y} + p_{l,y+1}, \quad 0 \le y \le l, \\
P[J = k] &= p_{k,0} p_{l-k,0}, \quad 0 \le k \le l,\ k \text{ paran}, \\
P\left[P_y = k\right] &= (y/k) p_{k,y}, \\
P[R = y] &= p_{l-y,y}, \quad 0 \le y \le l/2, \\
P[C = y] &= 2p_{l-1,2y+1}, \quad 0 \le y \le (l-1)/2.
\end{aligned}$$

Test slučajnih šetnji implementiran u TestU01 uzima dva parna cijela broja $m > m_0 > 0$ za parametre i generira n slučajnih šetnji duljine m. Za svaki l iz $\{m_0, m_0 + 2, \ldots, m\}$, test računa n vrijednosti statistika $H, \ldots, C$ definiranih iznad i uspoređuje njihove empirijske distribucije s odgovarajućim teorijskim distribucijama pomoću χ^2 testa.

Bliski parovi. Imamo sljedeću analogiju testa bliskih parova za binarne nizove. Svaka od n točaka u t dimenzija određena je nizom bitova duljine $m = st$, pri čemu svaka koordinata ima s bitova. Najčešće, svaki takav blok od s bitova se dobiva jednim pozivom generatora i ekstrahiranjem s bitova iz izlazne vrijednosti. Udaljenost između dvije točke X_i i X_j je definirana s $2^{-b_{i,j}}$, pri čemu je $b_{i,j}$ maksimalna vrijednost od b, a b je broj prvih bitova koji su isti u binarnoj ekspanziji svake koordinate za X_i i X_j. Ovo znači: ako je jedinična hiperkocka particionirana na 2^{tb} kubičnih kocki dijeljenjem svake osi na 2^b jednakih dijelova,

tada su dvije točke u istoj kocki za $b \leq b_{i,j}$, a ako je $b > b_{i,j}$ one su u različitim kockama. Neka je

$$D = \min_{1 \leq i < j \leq n} 2^{-b_{i,j}}$$

minimalna udaljenost između bilo koje dvije točke. Za dani par točaka je

$$P[b_{i,j} \geq b] = 2^{-tb}.$$

Imamo da vrijedi $D \leq 2^{-b}$, ako i samo ako je

$$-\log_2 D = \max_{1 \leq i < j \leq n} b_{i,j} \geq b,$$

tj. ako i samo ako se barem b bitova poklapa u barem jednom paru, a vjerojatnost da se to dogodi je približno

$$q_b \coloneqq P\left[D \leq 2^{-b}\right] \approx 1 - \left(1 - 2^{-tb}\right)^{n(n-1)/2}.$$

Ako se poklapaju u točno $b = -\log_2 D$ bitova, tada su lijeva i desna p-vrijednost $p_L = q_b$ i $p_R = 1 - q_{b-1}$. Ako je $N > 1$, test na dva nivoa računa minimum od N kopija od D i koristi to kao testnu statistiku, a p-vrijednost je dobivena iz

$$P\left[\min\{D_1, D_2, \ldots, D_N\} \leq 2^{-b}\right] \approx 1 - (1 - q_b)^N.$$

3.1.6 Primjer testiranja nekih poznatih generatora

U ovom dijelu dat ću rezultate primjene testova iz paketa TestU01 na neke poznate generatore slučajnih brojeva. U paketu testova TestU01 postoji 6 preddefiniranih skupova testova, tri za nizove slučajnih brojeva u intervalu $(0, 1)$, te tri za nizove bitova. Opisat ću samo skupove testova iz prve skupine.

To su skupovi testova *SmallCrush*, *Crush* i *BigCrush*, čija vremena izvršavanja za testiranje generatora poput MT19937, na procesoru AMD Athlon 64 (2.4 GHz), iznose 14 sekundi, 1 sat, odnosno 5.5 sati. Za testiranje generatora, preporučuje se započeti sa *SmallCrush* skupom. Ako je sve u redu, može se nastaviti s *Crush*, odnosno *BigCrush* skupom testova. U trenutnoj verziji, *Crush* koristi otprilike 2^{35} slučajnih brojeva i primjenjuje 96 statističkih testova. *BigCrush* koristi oko 2^{38} slučajnih brojeva i primjenjuje 106 testova. Testovi koji se koristi su ranije opisani testovi, uz neke dodatne.

Tablica 3.1.6 prikazuje rezultate testiranja odabranih generatora slučajnih brojeva. Za svaki generator, stupac $\log_2 \rho$ je vrijednost logaritma po bazi dva od perioda ρ. Stupci t-32 i t-64 su procesorska vremena potrebna za generiranje 10^8 slučajnih brojeva na 32-bitnom procesoru Intel Pentium (2.8 GHz), odnosno 64-bitnom procesoru AMD Athlon 64 (2.4 GHz). Zadnja tri stupca predstavljaju broj statističkih testova za koje je p-vrijednost izvan intervala $[10^{-10}, 1 - 10^{-10}]$, tj. broj testova na kojima je generator pao. Crtica (–) znači

da se skup testova nije primjenjivao na pojedini generator (uglavnom zato što je generator padao na prethodnim testovima). Također, neke vrijednosti imaju i broj u zagradama. Taj broj je broj testova čija p-vrijednost upada u interval $(10^{-10}, 10^{-4}] \cup [1 - 10^{-4}, 1 - 10^{-10})$. Njih možemo smatrati sumnjivim p-vrijednostima.

Rezultati testiranja poznatih RNG						
Generator	$\log_2 \rho$	t-32	t-64	*Small Crush*	*Crush*	*Big Crush*
LCG(2^{24}, 16598013, 12820163)	24	3.9	0.66	14	–	–
LCG(2^{31}, 65539, 0)	29	3.3	0.65	14	125 (6)	–
Java.util.Random	47	6.3	0.76	1	9 (3)	21 (1)
LCG(2^{48}, 44485709377909, 0)	46	4.1	0.65	5	24 (5)	–
LCG(2^{59}, 13^{13}, 0)	57	4.2	0.76	1	10 (1)	17 (5)
LCG(2^{63}, 5^{19}, 1)	63	4.2	0.75	0	5	8
LCG($2^{31} - 1$, 742938285, 0)	31	19.0	4.0	2	42 (5)	–
LCG($2^{31} - 1$, 950706376, 0)	31	20.0	4.0	2	42 (4)	–
LCG($10^{12} - 11$, 427419669081, 0)	39.9	87.0	25.0	1	22 (2)	34 (1)
Unix-random-32	37	4.7	1.6	5 (2)	101 (3)	–
Unix-random-64	45	4.7	1.5	4 (1)	57 (6)	–
Unix-random-128	61	4.7	1.5	2	13	19 (3)
Unix-random-256	93	4.7	1.5	1 (1)	8	11 (1)
MT19937	19937	4.3	1.6	0	2	2
WELL1024a	1024	4.0	1.1	0	4	4
WELL19937a	19937	4.3	1.3	0	2 (1)	2
LFSR113	113	4.0	1.0	0	6	6
LFSR258	258	6.0	1.2	0	6	6
Marsa-xor32 (13, 17, 5)	32	3.2	0.7	5	59 (10)	–
Marsa-xor64 (13, 7, 17)	64	4.0	0.8	1	8 (1)	7
Matlab-rand	1492	27.0	8.4	0	5	8 (1)
Matlab-LCG-Xor	64	3.7	0.8	0	3	5 (1)
KISS93	95	3.8	0.9	0	1	1
KISS99	123	4.0	1.1	0	0	0

Tablica 3.1: Rezultati testiranja poznatih RNG

3.2 Diehard testovi

Diehard testovi su skup statističkih testova za mjerenje kvalitete generatora slučajnih brojeva. Razvio ih je George Marsaglia i objavio prvi puta 1995. godine. Diehard testovi su se često spominjali kroz ovaj rad, tako da ću u ovom dijelu rada dati samo osnovni uvid u testove sadržane u Diehard skupu testova.

3.2.1 *Birthday spacing* test

Odabire se m rođendana u godini od n dana. Napravi se lista intervala između rođendana. Ako je j broj vrijednosti koje se pojavljuju više od jednom u toj listi, tada j ima asimptotsku Poissonovu distribuciju sa srednjom vrijednosti $m^3/(4n)$. Iskustvo je pokazalo da n mora biti poprilično velik, $n \geq 2^{18}$, da bi se rezultati mogli usporediti s Poissonovom distribucijom koja ima istu srednju vrijednost. Ovaj test koristi $n = 2^{24}$ i $m = 2^9$. Uzima se uzorak od 500 j-ova i pomoću χ^2 GOF testa dobiva p-vrijednost. Prvi test koristi bitove od 1 do 24, drugi bitove od 2 do 25, i tako dalje, do devetog testa koji koristi bitove od 9 do 32. Svaki skup bitova daje p-vrijednost, te se za usporedbu koristi KS test (Kolmogorov–Smirnovljev test).

3.2.2 Preklapajući 5-permutation test ili OPERM5 test

Test gleda niz od milijun 32-bitnih slučajnih cijelih brojeva. Svaki skup od 5 uzastopnih brojeva može biti u jednom od 120 stanja, za 5! mogućih rasporeda 5 brojeva. Promatra se na tisuće stanja, te se računaju kumulativne sume broja pojavljivanja svakog od stanja. Svako od 120 stanja trebalo bi se pojaviti sa statistički jednakom vjerojatnošću. Ova verzija koristi milijun cijelih brojeva, dva puta.

3.2.3 Binarni rang test 31×31 matrica

Pri ovom testu koristi se 31 bit od 31 slučajnog broja za formiranje 31×31 binarne matrice nad poljem $\{0, 1\}$, te se odredi rang matrice. Rang može biti od 0 do 31, ali rangovi manji od 28 su rijetki, te se brojevi njihova pojavljivanja zbrajaju. Računaju se rangovi 40,000 ovakvih slučajnih matrica, te se koristi χ^2 test na zbroju broja matrica ranga 31, 30, 29 i manjeg od 29.

3.2.4 Binarni rang test 32×32 matrica

Formira se 32×32 binarna matrica čiji retci su 32-bitni slučajni brojevi. Ponovno se određuje rang, koji može biti od 0 do 32. Rangovi manji od 29 su rijetki, te se njihova

pojavljivanja zbrajaju. Računaju se rangovi 40,000 ovakvih slučajnih matrica, te se koristi χ^2 test na zbroju matrica ranga 32, 31, 30 i manjeg od 30.

3.2.5 Binarni rang test 6×8 matrica

Iz svakog od 6 slučajnih 32-bitnih cijelih brojeva odabiremo 1 byte, te od njih formiramo 6×8 binarnu matricu te računamo njezin rang. Rang može biti od 0 do 6, ali rangovi 0, 1, 2, 3 su rijetki te se njihova pojavljivanja zbrajaju s brojem matrica ranka 4. Računaju se rangovi 100,000 ovakvih matrica, te se koristi χ^2 test na zbrojeve matrica ranga 6, 5 i manjeg od 5.

3.2.6 *Bitstream* test

Na izlaz algoritma kojeg testiramo gledamo kao na tok (eng. stream) bitova. Nazovimo te bitove $b_1, b_2, \ldots$ Promotrimo abecedu od dva znaka, 0 i 1, te gledajmo na tok bitova kao na preklapajuće riječi od 20 znakova. Neka je $b_1 b_2 \ldots b_{20}$ prva riječ, $b_2 b_3 \ldots b_{21}$ druga riječ i tako dalje. *Bitstream* test prebrojava broj riječi od 20 znakova (20 bitova) koje nedostaju u nizu od 2^{21} preklapajućih riječi od 20 znakova. Postoji 2^{20} različitih riječi od 20 znakova. Za stvarno slučajan niz od $2^{21} + 19$ bitova, broj j riječi koje nedostaju bi trebao biti vrlo blizu normalne distribucije sa srednjom vrijednosti 141909 i standardnom devijacijom (σ) 428. Test se ponavlja 20 puta.

3.2.7 OPSO, OQSO i DNA testovi

OPSO je skraćenica za *overlapping-pairs-sparse-occupancy*. OPSO test razmatra riječi od 2 znaka nad abecedom od 1024 znaka. Svaki znak je određen pomoću 10 bitova 32-bitnog cijelog broja iz niza kojeg testiramo. OPSO generira 2^{21} preklapajućih riječi od 2 slova i računa broj riječi koje nedostaju, tj. riječi od dva slova koje se ne pojavljuju kroz cijeli niz. Taj broj bi trebao biti vrlo blizu normalne distribucije sa srednjom vrijednosti 141909 i $\sigma = 290$. Stoga bi varijabla (missingwords – 141909)/290 trebala biti standardna normalna slučajna varijabla.

OQSO znači *overlapping-quadruples-sparse-occupancy*. OQSO test je sličan prethodnom, samo što on razmatra riječi od 4 znaka iz abecede od 32 znaka. Svaki znak je određen pomoću 5 uzastopnih bitova 32-bitnog slučajnog broja. Srednja vrijednost broja riječi koje nedostaju u nizu od 2^{21} riječi od 4 znaka je ponovno 141909, sa $\sigma = 295$.

DNA test promatra abecedu od 4 znaka: C, G, A, T. Svaki znak je određen pomoću dva bita iz niza slučajnih cijelih brojeva koje testiramo. Test razmatra riječi od 10 znakova, imamo 2^{20} mogućih riječi, te je srednja vrijednost broja riječi koje nedostaju u nizu od 2^{21} preklapajućih riječi od 10 znakova opet jednaka 141909. Standardna devijacija (σ) iznosi 339, a ona je određena OQSO simulacijama.

3.2.8 Test brojanja jedinica u toku bytova

Izlaz algoritma kojeg testiramo promatramo kao tok (eng. stream) bytova (po 4 od svakog 32-bitnog cijelog broja). Svaki byte može sadržavati od 0 do 8 jedinica, sa vjerojatnostima 1, 8, 28, 56, 70, 56, 28, 8, 1 povrh 256. Neka tok bytova daje niz preklapajućih riječi od 5 znakova, pri čemu svaki znak može biti A, B, C, D ili E. Znakovi su određeni brojem jedinica u bytu: 0, 1 ili 2 daje A, 3 daje B, 4 daje C, 5 daje D, te 6, 7 ili 8 daje E. Na problem možemo gledati kao na majmuna (monkey tests) koji udara 5 tipaka na tipkovnici s različitim vjerojatnostima (37, 56, 70, 56, 37 povrh 256). Postoji 5^5 mogućih riječi od 5 znakova, a iz niza od 256,000 preklapajućih riječi od 5 znakova računamo frekvencije svake riječi.

3.2.9 Test brojanja jedinica u jednom bytu

Izlaz algoritma kojeg testiramo promatramo kao tok 32-bitnih cijelih brojeva. Iz svakog cijelog broja odaberemo jedan byte, npr. prvih 8 bitova. Test je sličan prethodnom.

Svaki byte može sadržavati od 0 do 8 jedinica, s vjerojatnostima 1, 8, 28, 56, 70, 56, 28, 8, 1 povrh 256. Odabrani bytovi uzastopnih cijelih brojeva čine niz preklapajućih riječi od 5 znakova, pri čemu svaki znak može biti A, B, C, D ili E. Znakovi su određeni brojem jedinica u bytu, na isti način kao u prethodnom testu, te je ostatak testa isti.

3.2.10 *Parking lot* test

Na kvadratno parkiralište sa 100×100 mjesta nasumično se parkira automobil u kvadratić veličine 1×1, pokušavajući pritom ne parkirati na već zauzeto mjesto. Ako se pokuša parkirati na već zauzeto mjesto, automobil se ponovno pokušava parkirati na novoj, nasumičnoj lokaciji. Svaki pokušaj rezultira, ili sudarom, ili uspjehom i povećanjem broja parkiranih automobila. Označimo s n broj pokušaja parkiranja, a s k broj uspješnih parkiranja. Za $n = 12000$ pokušaja, simulacije pokazuju da bi k trebao biti u prosjeku 3523, sa $\sigma = 21.9$ i vrlo blizu normalne distribucije. Stoga bi $(k - 3523)/21.9$ trebala biti standardna normalna slučajna varijabla, koja konverzijom u uniformnu varijablu služi kao ulaz za KS test baziran na uzorku od 10.

3.2.11 Test minimalne udaljenosti

Test 100 puta ponavlja sljedeće: odabere se $n = 8000$ slučajnih točaka u kvadratu stranice 10000. Pronađe se d, najmanja udaljenost između $(n^2 - n)/2$ parova točaka. Ako su točke uistinu nezavisne i uniformne, tada bi d^2, tj. kvadrat najmanje udaljenosti, trebao biti vrlo blizu eksponencijalne distribucije sa srednjom vrijednosti 0.995. Stoga bi

$1 - \exp(-d^2/0.995)$ trebala biti uniformna slučajna varijabla na $[0, 1)$ i KS test na 100 vrijednosti služi kao test uniformnosti slučajnih točaka u kvadratu.

3.2.12 Test 3D sfere

Odabere se 4000 slučajnih točaka iz kocke sa stranicom 1000. Oko svake točke napravi se sfera dovoljno velika da uhvati sljedeću najbližu točku. Tada bi volumen najmanje sfere trebao biti eksponencijalno distribuiran sa srednjom vrijednosti $120\pi/3$. Test 3D sfere generira 4000 ovakvih sfera 20 puta. Svaki najmanji radijus vodi do uniformne slučajne varijable sa srednjom vrijednosti $1 - \exp(r^3/30)$, te se radi KS test na 20 p-vrijednosti.

3.2.13 *Runs* test

Test broji uspone i padove uniformnih varijabli na intervalu $[0, 1)$. Na primjer, niz 0.123, 0.357, 0.789, 0.425, 0.224, 0.416, 0.95 ima uspon duljine 3, pad duljine 2, te ponovno uspon duljine (najmanje) 2. Poznata je matrica kovarijanci za uspone i padove. Usponi i padovi se računaju za nizove duljine 10000, a postupak se ponavlja 10 puta.

3.2.14 *Craps* test

Igra se 200000 partija craps-a, pronalazi se broj pobjeda i potreban broj bacanja da se završi svaka partija. Broj pobjeda bi trebao biti normalna slučajna varijabla sa srednjom vrijednosti $200000p$ i varijancom $200000p(1 - p)$, za $p = 244/495$. Broj bacanja potreban da se završi svaka partija može biti od 1 do beskonačno, ali zbroj slučajeva kada je potrebno više od 21 je pribrojen broju slučajeva kada je potrebno 21 bacanje. Radi se χ^2 test na potrebnom broju bacanja.

Bibliografija

[1] Pierre L'Ecuyer i Richard Simard, *TestU01: A C Library for Empirical Testing of Random Number Generators*, ACM Trans. Math. Soft. **23** (2007), br. 4, http://dx.doi.org/10.1145/1268776.1268777.

[2] George Marsaglia, *Random number generators*, J. Mod. App. Stat. Meth. **2** (2003), br. 1, http://dx.doi.org/10.22237/jmasm/1051747320.

[3] ———, *Xorshift RNGs*, J. Stat. Softw. **8** (2003), br. 14, http://dx.doi.org/10.18637/jss.v008.i14.

[4] George Marsaglia i Wai Wan Tsang, *Some difficult-to-pass tests of randomness*, J. Stat. Softw. **7** (2002), br. 3, http://dx.doi.org/10.18637/jss.v007.i03.

[5] Makoto Matsumoto i Takuji Nishimura, *Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator*, ACM T. Model. Comput. S. **8** (1998), br. 1, 3–30.

Dodatna literatura

[6] Elwyn R. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, Laguna Hills, CA, 1984.

[7] Glyn David Carter, *Aspects of local linear complexity*, Disertacija, 1989.

[8] Eva Diane Erdmann, *Empirical tests of binary keystreams*, 1992.

[9] George S. Fishman, *Monte Carlo: Concepts, Algorithms, and Applications*, Springer Series in Operations Research and Financial Engineering, Springer–Verlag, New York, NY, 1996.

[10] Antónia Földes, *The limit distribution of the length of the longest head run*, Period. Math. Hung. **10** (1979), br. 4, 301–310.

[11] Louis Gordon, Mark F. Schilling i Michael S. Waterman, *An extreme value theory for long head runs*, Probab. Theory Rel. **72** (1986), br. 2, 279–287.

[12] Peter Kirschenhofer, Helmut Prodinger i Wojciech Szpankowski, *Digital search trees again revisited: The internal path length perspective*, SIAM J. Comput. **23** (1994), br. 3, 598–616.

[13] Donald E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd edition*, Addison–Wesley, Reading, MA, 1998.

[14] Pierre L'Ecuyer, *Testing random number generators*, WSC'92: Proceedings of the 24th conference on Winter simulation (New York, NY), ACM, 1992, str. 305–313.

[15] Pierre L'Ecuyer i Richard Simard, *Beware of linear congruential generators with multipliers of the form* $a = \pm 2^q \pm 2^r$, ACM Trans. Math. Soft. **25** (1999), br. 3, 367–374.

[16] George Marsaglia, *A current view of random number generators*, Computer Science and Statistics: Proceedings of the Sixteenth Symposium on the Interface (Amsterdam) (L. Billard, ur.), Elsevier Science Publishers, North–Holland, 1985, str. 3–10.

[17] George Marsaglia i Liang Huei Tsay, *Matrices and the structure of random number sequences*, Linear Algebra Appl. **67** (1985), br. 3, 147–156.

[18] George Marsaglia i A. Zaman, *Monkey tests for random number generators*, Computers Math. Applic. **26** (1993), br. 9, 1–10.

[19] James L. Massey, *Shift-register synthesis and BCH decoding*, IEEE T. Inform. Theory **IT–15** (1969), br. 1, 122–127.

[20] Makoto Matsumoto i Yoshiharu Kurita, *Twisted GFSR generators II*, ACM T. Model. Comput. S. **4** (1994), br. 3, 254–266.

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

[22] Harald Niederreiter, *The linear complexity profile and the jump complexity of keystream sequences*, Advances in Cryptology: Proceedings of EUROCRYPT'90 (Berlin) (Ivan Bjerre Damgård, ur.), Springer–Verlag, 1991, str. 174–188.

[23] A. L. Rukhin, *Testing randomness: A suite of statistical procedures*, Theory Probab. Appl. **45** (2001), br. 1, 111–132.

[24] M. Z. Wang, *Linear complexity profiles and jump complexity*, Inform. Proces. Lett. **61** (1997), br. 3, 165–168.

[25] Jacob Ziv i Abraham Lempel, *Compression of individual sequences via variable-rate coding*, IEEE T. Inform. Theory **IT–24** (1978), br. 5, 530–536.

Sažetak

U današnje vrijeme algoritmi za generiranje slučajnih brojeva postaju glavni izvor slučajnih brojeva, no oni su deterministički i periodički. Da bi algoritam bio prihvatljiv u nekim osjetljivim područjima poput kriptografije, gdje je nepredvidivost generiranih brojeva ključni zahtjev, on mora proći stroge testove slučajnosti. Ne postoji algoritam koji može proći baš sve testove slučajnosti. Isto tako, ako algoritam prođe sve testove slučajnosti, to ne znači da je on bez greške. Prolazak testova slučajnost može samo pojačati naše povjerenje u pojedini algoritam.

Jedino generatori slučajnih brojeva koji koriste fenomene iz prirode mogu generirati stvarne slučajne brojeve, ali je njihova brzina generacije nedovoljna za današnje potrebe. Međutim, u praksi se najčešće koristi hibridni pristup, tj. koristi se pseudoslučajni generator čije se sjeme određuje pomoću generatora stvarnih slučajnih brojeva.

Summary

In modern times, algorithms for generating random numbers are becoming the main source of random numbers. However, they are deterministic and periodic. An algorithm must pass strict tests of randomness in order to be acceptable for cryptographic and similar purposes, as unpredictability of generated numbers is the main requirement. There is no such algorithm that can pass all tests of randomness, but passing a number of tests can boost our faith in a certain algorithm. Also, if an algorithm has passed all tests of randomness, that does not mean it is flawless.

Only random number generators that use some physical phenomenon can generate true random numbers. But, those generators are so slow that they cannot be used for modern purposes. In practice, hybrid approach has shown the best results, where the seed for a pseudorandom number generator is determined with a true random number generator.

Životopis

Rođen sam 20. listopada 1992. u Šibeniku. Završio sam Osnovnu školu „Vodice" u Vodicama, a Opću gimnaziju u Gimnaziji „Antun Vrančić" u Šibeniku. 2011. godine upisao sam Preddiplomski sveučilišni studij Matematika na Matematičkom odsjeku PMF-a u Zagrebu. Preddiplomski sveučilišni studij sam završio 2014. te stekao titulu univ.bacc.math. U listopadu 2014. godine upisujem Diplomski sveučilišni studij Računarstvo i matematika na PMF-u u Zagrebu kojeg trenutno završavam.