PRESEK

List za mlade matematike, fizike, astronome in računalnikarje

ISSN 0351-6652
Letnik **33** (2005/2006)
Številka 3
Strani 22–25

Aleksander Vesel:

GENERATORJI SLUČAJNIH ŠTEVIL

Ključne besede: računalništvo, slučajna števila, generiranje, Lehmerjeva metoda.

Elektronska verzija: http://www.presek.si/33/1625-Vesel.pdf

Generatorji slučajnih števil

Aleksander Vesel

Uvod

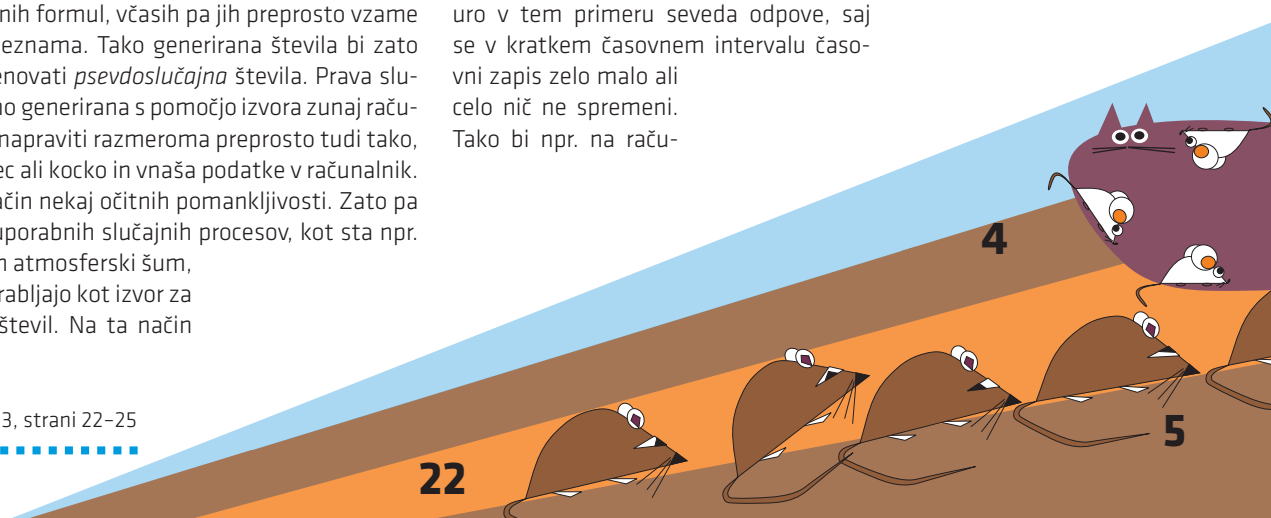
An ban, pet podgan. Tako se začne zelo znana *izštevanka.* Kot vemo, je to pesem oz. ritmizirano besedilo, s katerim se na začetku igre na »slučajen« način določijo udeležencem vloge v igri. Še bolj očitno se s slučajnim dogajanjem, velikokrat temelji to na zaporedju slučajnih števil, srečamo pri igrah na srečo. Igra *Človek ne jezi se* je seveda zelo znan in očiten primer.

S pojavom računalnikov se je zelo kmalu pojavila potreba, da se slučajnost prenese tudi v računalniški program. Pri tem ne gre samo za programe, ki oponašajo igre na srečo, ampak tudi za zelo resne programe. Programi za urejanje in programi za kriptografijo podatkov so samo majhen del sodobne programske opreme, pri kateri si lahko pomagamo z generiranjem slučajnih števil. Zanimivo je tudi, da so zaporedja slučajnih vrednosti zelo učinkovito orodje v programih, s katerimi rešujemo najtežje probleme. Običajno pravimo, da je težek problem tisti, za katerega ne znamo napisati uporabnega računalniškega programa, ki bi tak problem popolnoma rešil. Pri teh problemih se zato zadovoljimo s približkom rešitve, pri reševanju pa pogosto pomembno vlogo igrajo generatorji slučajnih števil.

Če se slučajna števila tvorijo znotraj računalniškega programa, seveda niso v resnici slučajna. Velikokrat jih program izračuna na podlagi matematičnih formul, včasih pa jih preprosto vzame iz prej izračunanega seznama. Tako generirana števila bi zato pravzaprav morali imenovati *psevdoslučajna* števila. Prava slučajna števila so običajno generirana s pomočjo izvora zunaj računalnika. To je mogoče napraviti razmeroma preprosto tudi tako, da nekdo meče kovanec ali kocko in vnaša podatke v računalnik. Seveda ima opisani način nekaj očitnih pomankljivosti. Zato pa obstaja nekaj drugih uporabnih slučajnih procesov, kot sta npr. radioaktivno sevanje in atmosferski šum, ki se tudi v resnici uporabljajo kot izvor za generiranje slučajnih števil. Na ta način pridobljena zaporedja slučajnih števil najdemo tudi na nekaterih specializiranih spletnih straneh (glej npr. www.random.org).

V praksi večinoma še vedno uporabljamo predvsem slučajna števila, ki jih tvori računalniški program oz. podprogram. Ker ne moremo generirati resnično slučajnega zaporedja, poskušamo generirati zaporedje, ki je po svojih lastnostih čim bližje pravemu slučajnemu zaporedju. Precej presenetljivo je, da je to zelo težaven problem, ki ga popolnoma verjetno ne bomo nikoli rešili. Precej težavna je že definicija lastnosti slučajnega zaporedja, saj intuicija pri tem marsikdaj odpove. Pri metanju kovanca tako ne pričakujemo, da se bo desetkrat zapored pojavila številka. Če vržemo kovanec desetkrat, je to v resnici zelo malo verjetno. Nasprotno pa je pri velikem številu metov zelo verjetno, da se nekje pojavi zaporedje desetih zaporednih številk, čeprav nam intuicija pravi drugače.

Generiranje posamičnega slučajnega števila v računalniškem programu ni posebno težavno. Ker se izvaja program na računalniku, lahko v tem primeru za generator uporabimo sistemsko uro in iz časovnega zapisa izluščimo »slučajno« zaporedje bitov. Do pravega problema pride, ko želimo pridobiti *zaporedje* slučajnih števil. Postopek s sistemsko uro v tem primeru seveda odpove, saj se v kratkem časovnem intervalu časovni zapis zelo malo ali celo nič ne spremeni. Tako bi npr. na raču-



nalniškem sistemu, ki meri čas v stotinkah sekunde, algoritem generiral dve isti zaporedni slučajni števili, če bi prišli zahtevi po generiranju dveh slučajnih števil znotraj tega časovnega intervala. Sistemska ura pa je zelo uporabna za določitev začetne vrednosti slučajnega zaporedja, ki ga imenujemo tudi *seme*.

Programski jeziki običajno generator slučajnih števil že vsebujejo. Še zdaleč pa ni nujno, da je v programski jezik vgrajeni generator tudi v resnici kakovosten. Leta 1988 sta tako Park in Miller objavila članek, v katerem ugotavljata, da je dobrih generatorjev malo, nekateri pa so celo zelo slabi. Tudi avtor tega prispevka ima zelo slabe izkušnje z enim od starejših pascalskih prevajalnikov. Kot bomo videli v nadaljevanju, pa lahko le z osnovnim znanjem programiranja sami napišemo spodoben generator.

Lehmerjeva metoda

Večina sodobnih programskih jezikov za generiranje slučajnih števil uporablja metodo, ki jo je leta 1951 predstavil Lehmer. V principu metoda deluje podobno kot izštevanka: neko (ponavadi veliko) število najprej pomnožimo z drugim številom. Dobimo vrednost, ki ustreza številu zlogov izštevanke. Dobljeni vrednosti nato prištejemo število, ki ustreza začetni točki izštevanja. Na koncu izračunamo ostanek pri deljenju s tretjo vrednostjo, ki ustreza številu otrok, ki sodelujejo v izštevanki.

Formalno Lehmerjevo metodo zapišemo na naslednji način. Zaporedje vrednosti $x_1, x_2, x_3, \ldots$ dobimo z enačbo

- $x_i+1 := (Ax_i+B) \bmod M$.

Pogosto postavimo kar $B=0$, kar olajša generiranje zaporedja. V nasprotnem primeru je generator zahtevnejši, zaradi česar mogoče intuitivno pričakujemo boljše rezultate. V resnici je prej obratno; tako definirani generatorji so zelo občutljivi, kar bomo v nadaljevanju ilustrirali s primerom.

Če velja $B=0$, je zaporedje generiranih števil odvisno od začetne vrednosti x_0 ter parametrov A in M. Začetna vrednost $x_0=0$ je seveda vedno slaba. Če pa sta A in M dobro izbrana, je lahko katerakoli vrednost z intervala med 1 in M dobra.

Ugodno je, če je M praštevilo. Na ta način zagotovimo, da so vse vrednosti x_i zanesljivo različne od 0.

Primer. Naj bo $M=7$, $A=5$ in $x_0=2$. Dobimo zaporedje

- 3, 1, 5, 4, 6, 2, 3, 1, 5, 4, 6, 2, ...

Števila v zaporedju se ponovijo po šestih generiranih številih. Pravimo, da ima generirano zaporedje *periodo* dolžine šest. Pravo slu-

čajno zaporedje števil v resnici nima periode, pri Lehmerjevi metodi pa se periodi ne moremo izogniti. Ali je zato Lehmerjeva metoda neuporabna? Teoretično seveda je, v praksi pa ne nujno. Pomagamo si tako, da izberemo čim večji M in definiramo zaporedje s periodo, ki je tako dolga, kot je to le mogoče. Ponavadi to pomeni, da smo zadovoljni s periodo, ki je blizu velikosti množice celih števil v danem računalniku oz. programskem jeziku.

V zgornjem primeru je dolžina periode enaka $M-1=6$, kar je največja možna dolžina za dani M. Če je M praštevilo, lahko vedno najdemo tak A, da dobimo zaporedje s periodo največje dolžine. Nekatere vrednosti konstante A pa povzročijo zaporedja s precej krajšo periodo, kar kaže naslednji primer.

Primer. Naj bo $M=7$, $A=4$ in $x_0=2$. Dobimo zaporedje

- 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, ...

Vrednosti v zaporedju se ponovijo že po treh generiranih številih, čeprav je v primerjavi s prejšnjim primerom edina razlika spremenjena vrednost konstante A.

Kot smo že povedali, si želimo generirati zaporedja s čim večjo periodo. Glede na zgornja primera je torej potrebno določiti čim večje praštevilo M in ustrezno vrednost konstante A. Zaradi zaželene čim večje hitrosti delovanja generatorja pa je smiselno, da izberemo praštevilo M v okviru velikostnega reda *besede* centralne procesne enote. (Spomnimo se, da velikost besede, poenostavljeno povedano, označuje največje število bitov, s katerimi lahko procesor naenkrat računa.)

Večina današnjih osebnih računalnikov in delovnih postaj ima 32 bitne procesorje. To nam omogoča, da izberemo praštevilo M iz intervala $0 \ldots 2^{32}-1$. Že Lehmer je predlagal izbiro $M=2^{31}-1$ oz. $M=2147483647$. Za to praštevilo je konstanta $A=48271$ ena od vrednosti, pri kateri dobimo zaporedje vrednosti z največjo dolžino periode. Nekateri avtorji priporočajo konstanto $A=16807$, ki naj bi tudi dala še sprejemljive rezultate.

Za veliko večino verjetnostnih problemov perioda z dolžino več kot dve milijardi povsem zadostuje, pri predlagani konstanti $A=48271$ pa zaporedje zadovoljuje tudi večino drugih kriterijev za uspešnost generatorja. Zato je Lehmerjeva metoda pri tako določenih parametrih v večini primerov najboljša izbira in jo je smiselno uporabljati, razen če ni zares nujno uporabiti generatorja z drugačnimi lastnostmi (npr. z daljšo periodo).

Marsikateri (posebej starejši) prevajalnik žal nima tako kakovostnega generatorja. Pogosto je vzrok neničelna konstanta B, zaradi česar postane generator zelo občutljiv.

Primer. Da bo primer bolj nazoren, vzemimo že znana parametra $M=2147483647$ in $A=48271$, namesto $B=0$ pa vzemimo $B=1$. Dobimo zaporedje določeno z enačbo

- $x_i+1 := (48271\, x_i+1) \bmod 2147483647.$

Konstantama, ki definirata dobro slučajno zaporedje, smo torej dodali še konstanto, ki naj bi omogočila, da bo zaporedje »še bolj« slučajno. Pa je to tudi res?

Če postavimo $x_0=179424105$, dobimo

- $x_1 := (48271\,(179424105)+1) \bmod 2147483647 = 179424105.$

Z majhno spremembo smo dober generator tako pokvarili, da lahko obtiči v ciklu s periodo ena.

Realizacija Lehmerjeve metode

Podprogram, ki temelji na tako preprosti enačbi, kot jo definira Lehmerjeva metoda, bi moral biti zelo enostaven. Ko pa gre za realizacijo na računalniškem sistemu, kar nas seveda tudi najbolj zanima, pa je potrebno precej previdnosti.

Običajno uporabimo za zaporedje slučajnih vrednost x_0, x_1, x_2, x_3, ... globalno spremenljivko. Tej spremenljivki je potrebno najprej določiti začetno vrednost x_0. V času pisanja oz. testiranja programa je ugodno, da je začetna vrednost vedno enaka, saj na ta način pri vsakem zagonu programa dobimo isto zaporedje vrednosti. Ko program deluje, pa začetno vrednost ponavadi slučajno določimo s pomočjo sistemske ure, lahko pa jo določi tudi sam uporabnik.

Kot smo že povedali, Lehmerjeva metoda za zgoraj izbrani vrednosti M in A, vrne vrednost med 1 in $M-1=2147483646$. Običajno je uporabnejše zaporedje vrednosti iz intervala med 0 in 1, ki ga lahko zelo hitro pretvorimo v zaporedje s poljubnega intervala. S tem razmislekom dobimo naslednji pascalski podprogram, ki pa je, kot bomo videli kasneje, napačen.

```
Function random (var x: Longint): real;
const
    M = 2147483647;
    A = 48271;
begin
    x := (A * x) mod M;
    random := x / M;
end;
```

Težava zgornjega podprograma je, da lahko izraz $A * x$ preseže največjo vrednost, ki jo lahko predstavimo z 32 biti oz. s podatkovnim tipom Longint (s katerim v pascalskih prevajalnikih običajno predstavimo 32 bitno celo število). V tem primeru se bo ta podprogram bodisi predčasno zaključil bodisi, če prevajalnik dovoli prekoračitev, bo zaporedje drugačno od željenega.

Rešitev ponuja spodnji podprogram. Podrobnosti izračuna zahtevajo nekaj matematične spretnosti in presegajo namen tega prispevka. Povejmo le, da si pomagamo z dvema novima konstantama:

- Q, ki je rezultat celoštevilskega deljenja M z A, in
- R, ki je ostanek celoštevilskega deljenja M z A.

Pri izračunu potem uporabimo dejstvo, da lahko ostanek pri deljenju celega števila z M izračunamo tudi tako, da najprej celoštevilsko delimo to število z M, nato rezultat pomnožimo z M in dobljeni produkt odštejemo od prvotnega števila.

```
Function random(var x: Longint): real;
{Funkcijski podprogram vrne slučajno
vrednost iz intervala (0,1)}
const
    M = 2147483647;
    A = 48271;
    Q = 44488;
    R = 3399;
var y: Longint;
begin
    y := A * (x mod Q) - R * (x div Q);
    if y > 0 then x := y else x := y + M;
    random := x / M;
end.
```