Generatory liczb losowych (1)

Mgr inż. Szymon Łukasik

szymonl@pk.edu.pl

Po co nam liczby losowe?

– symulacje (jeżeli używamy komputera to potrzebujemy liczb losowych by badane zjawisko urealnić) np. fizyka jądrowa (losowe zdarzenia cząstek) czy badania operacyjne (np. problemy optymalizacji – planowanie przydziału zadań w maszynie wieloprocesorowej)

– próbkowanie – wybór przypadków z dużej ilości danych

– analiza numeryczna

– programowanie – np. badanie efektywności algorytmów,

– podejmowanie decyzji – czasami zdajemy się tu na „ślepy los”, patrz: kryzys finansowy

– rozrywka – gry komputerowe, gry losowe („jednoręki bandyta”)

Jak można generować liczby losowe – z nieinformatycznej perspektywy

„Prawdziwie” losowym generatorem jest rozpad atomowy cząstek. Zachodzi on w losowych chwilach czasu, a jego zajście można zarejestrować licznikiem Geigera-Millera. W Internecie można znaleźć realizacje zmiennej losowej pobrane w ten sposób, a nawet istnieje strona na której dane takie można zamówić www.fourmilab.ch/hotbits.

Można również uzyskać liczby losowe analizując zjawiska zachodzące wewnątrz zwykłego komputera np. turbulencje powietrza wywołane pracą dysku twardego, szum termiczny w półprzewodnikach itp.

Generowanie liczb losowych – najprostszy generator liniowy

Najczęściej stosowanymi w zagadnieniach inżynierskich generatorami są generatory deterministyczne, pseudolosowe. Wyjściem takiego generatora jest sekwencja liczb która ma charakter losowy, jednak liczba otrzymywana jako *i*-ta zależy w pewien sposób od *k* liczb poprzednich tj. $x_i=f(x_{i-k}, x_{i-k+1}, ..., x_{i-1})$. Parametr k to tzw. rząd generatora.

Podstawowym generatorem tego typu jest generator liniowy. Najczęściej stosuje się tzw. **generator liniowy kongruencyjny**.

Kongruencja to relacja określona w zbiorze liczb całkowitych. Kongruencja modulo *n* nazywana jest też *przystawaniem liczb* "modulo *n*".

Liczby całkowite *a* i *b* przystają modulo *n* (pozostają w kongruencji modulo n), co zapisuje się: $a \equiv b (mod\ n)$ - [czasami oznacza się bez nawiasu] jeżeli ich różnica *a* − *b* dzieli się bez reszty przez *n*. Równoważnie: jeśli liczby *a* i *b* dają w dzieleniu przez *n* tę samą resztę.

Generator liniowy kongruencyjny generuje ciąg (tzw. ciąg Lehmera) :

$x_i = (a x_{i-1} + c) \bmod m$, gdzie: $0 \le x_i < m$

stan początkowy generatora x_0 to tzw. ziarno (seed). Pozostałe parametry specyfikują konfigurację generatora. Jeśli *c=0* to mamy do czynienia z tzw. **generatorem multiplikatywnym**.

Z generatora tego otrzymujemy liczby z przedziału [0, m), często zatem dokonujemy przeskalowania *($u_i = x_i / m$)* by uzyskać liczby losowe z przedziału [0,1). Z teoretycznych rozważań: maksymalny okres generatora liniowego kongruencyjnego to *m-1,* a minimalne i maksymalne wartości to odpowiednio $u_{min} = 0$, $u_{max} = 1-1/m$. Parametr m powinien być liczbą pierwszą. Jak łatwo zauważyć z generatora multiplikatywnego nie można uzyskać elementu $x_i=0$.

Bardziej wysublimowane generatory.

Opisany powyżej generator przy $a=7^5$, $c=0$, $m=2^{31}-1$ jest uznawany za „minimalny standard" jednak obecnie, ze względu na charakterystyki statystyczne częściej stosuje się generatory będące połączeniem kilku transformacji na bitach ciągu np.

A. 64 bitowy XOR z przesunięciem

B. mnożenie z przeniesieniem i bazą 2^{32}

C. generator liniowy kongruencyjny, z $m=2^{64}$

D. generator multiplikatywny, z $m=2^{64}$

E. generator multiplikatywny, z $m>>2^{32}$ (liczba pierwsza)

Zalecam Państwu również zapoznanie się z opisem generacji liczb losowych w książce Numerical Recipes. Pokazano tam między innymi bardzo interesujący (i prosty zarazem) generator który w kodzie w języku C przyjmuje postać:

```
struct Ran {
Ullong u,v,w;
Ran(Ullong j) : v(4101842887655102017LL), w(1)
{
        u = j ^ v;
        int64();
        v = u;
        int64();
        w = v;
        int64();
}
inline Ullong int64()
{
        u = u * 2862933555777941757LL + 7046029254386353087LL;
        v ^= v >> 17; v ^= v << 31; v ^= v >> 8;
        w = 4294957665U*(w & 0xffffffff) + (w >> 32);
        Ullong x = u ^ (u << 21); x ^= x >> 35; x ^= x << 4;
        return (x + v) ^ w;
}
inline Doub doub() { return 5.42101086242752217E-20 * int64(); }
```

```
inline Uint int32() { return (Uint)int64(); }
};
```

Stanowi on połączenie operacji A, B, C i posiada w zupełności zadowalający okres ok. $3{,}138 \cdot 10^{57}$ (niestety generator ten stosunkowo wolny). Obecnie w wielu zastosowaniach używa się generatora Mersenne-Twister (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html). Posiada on okres $2^{19937}-1$ i niezłe własności statystyczne (choć nie poleca się jego stosowania w zagadnieniach kryptografii).

Proszę zdawać sobie sprawę na przyszłość z roli ziarna generatora liczb pseudolosowych. Jeśli nie ustalimy go w sposób przypadkowy np. na podstawie daty czy ruchów myszki to wyniki działania algorytmu opartego o generator będą się reprodukować!