Zagadnienia maturalne z informatyki. Wydanie II. Tom II

Autorzy: Tomasz Francuz, Marcin Szeliga
ISBN: 83-246-0298-4
Format: B5, stron: 232

Przystąp do matury odpowiednio przygotowany

- Opanuj wszystkie wymagane zagadnienia
- Rozwiąż przykładowe zadania
- Poznaj zasady działania komputera

Jeśli przygotowujesz się do egzaminu maturalnego z informatyki, chcesz pogłębić wiedzę informatyczną, którą zdobywasz w szkole, lub poznać budowę komputera i zasady programowania – zajrzyj do tej książki. Znajdziesz tu wszystkie informacje, jakich możesz do tego potrzebować. Przeczytasz o różnych aspektach programowania, językach programowania, szyfrowaniu danych i kryptografii oraz metodach numerycznych.

Opracowując „Zagadnienia maturalne z informatyki. Wydanie II", autorzy wykorzystywali materiały udostępnione przez Ministerstwo Edukacji Narodowej, zadania z olimpiad informatycznych oraz podręczniki szkolne. Dzięki temu przedstawione w książce zagadnienia są dostosowane do zakresu tematycznego zadań maturalnych.

- Programowanie strukturalne i obiektowe
- Podstawowe elementy języków programowania
- Sterowanie przebiegiem działania programu
- Pobierane danych ze źródeł zewnętrznych
- System binarny, ósemkowy i szesnastkowy
- Algebra Boole'a
- Podstawy algorytmiki
- Algorytmy sortowania i przeszukiwania
- Szyfrowanie danych
- Metody numeryczne
- Analiza złożoności algorytmów

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Tom I

Tom II

Rozdział 5.

Liczby pseudolosowe

Liczby losowe mają ogromne zastosowanie w informatyce. **Przede wszystkim są używane do generowania kluczy i haseł, a więc to od nich zależy bezpieczeństwo systemów komputerowych i przechowywanych w nich danych**. Klucz wygenerowany na podstawie ciągu niebędącego ciągiem losowym może zostać w prosty sposób złamany — znając wady generatora, możemy je odtworzyć i wygenerować podobny ciąg, a co za tym idzie — odtworzyć klucz. Oprócz kryptografii i problemów bezpieczeństwa liczby losowe znajdują zastosowanie w symulacjach różnych zjawisk fizycznych, a także w… grach. Innym ich zastosowaniem są symulacje, np. Monte Carlo. Służą one do numerycznego rozwiązywania różnych problemów. Wiele układów lub zjawisk jest zbyt skomplikowanych, aby przetestować wszystkie możliwe kombinacje i sprawdzić zachowanie badanego układu. W takich przypadkach wykorzystuje się generatory liczb losowych do sprawdzenia układu w przypadkowych sytuacjach — testując odpowiednią liczbę kombinacji metodami statystycznymi, możemy udowodnić prawidłowe lub wadliwe działanie układu, możemy też ocenić typowe zachowanie badanego układu.

Liczby losowe możemy stosunkowo łatwo wygenerować — wyobraźmy sobie, że rzucamy monetą. Prawdopodobieństwo, że wypadnie reszka lub orzeł wynosi dokładnie $^1/_2$ i z góry nie daje się przewidzieć co wypadnie w kolejnym rzucie. Na tym prostym przykładzie łatwo zauważyć cechy, jakimi powinien charakteryzować się idealny generator liczb losowych:

- powinien dawać losowe wyniki;
- kolejny wynik nie powinien być uzależniony od poprzednio uzyskanych wyników.

Niektóre definicje liczb losowych i generatorów liczb losowych obejmują warunek równości rozkładu uzyskanych liczb losowych w podanym przedziale. Jednak znane są generatory liczb losowych, których wynikiem jest rozkład inny niż równomierny — np. rozkład normalny lub Poissona.

Czy wynik rzutu monetą rzeczywiście jest losowy? Czy podczas gry w ruletkę prawdopodobieństwo wypadnięcia czarnych lub czerwonych rzeczywiście wynosi $^1/_2$? Na pierwszy rzut oka mogłoby się tak wydawać. Jednak sami łatwo jesteśmy w stanie nauczyć się tak rzucać monetą, aby uzyskać z góry ustalony rezultat, podobnie jak doświadczony krupier potrafi tak rzucić kulką, aby rozkład uzyskanych wartości wcale nie był losowy. Potrzebne są testy, które określą jak dobry jest dany generator i czy otrzymywane sekwencje rzeczywiście są losowe. Takimi testami zajmiemy się pod koniec tej części rozdziału.

Niezależnie od naszych umiejętności, czy umiejętności krupiera, wszystkie zdarzenia podlegają ścisłym prawom fizyki. W takim razie **czy w ogóle możemy uzyskać losowe dane**? Na pewno **współczesne komputery, jako układy deterministyczne** (choć ich działanie może nam się wydawać przypadkowe) **nie mogą być źródłem liczb losowych.**

Jednak nawet proste układy, które doskonale potrafimy opisać prawami fizyki zachowują się w pewnych okolicznościach niby-losowo. Wyobraźmy sobie grę w snookera. Ruchem bil rządzą proste prawa, typu kąt padania równa się kątowi odbicia, zasada zachowania pędu itd. Powodują one, że w przypadku uderzenia jednej bili w drugą jesteśmy w stanie bardzo precyzyjnie przewidzieć zachowanie obu bil po zderzeniu. Jednak już w przypadku zderzenia trzech bil, mimo że posiadamy dokładne równania opisujące ich ruch, efekt tego zderzenia jest kompletnie nieprzewidywalny. Końcowa pozycja bil różni się diametralnie w zależności od drobnych różnic przed zderzeniem, np. niewielkiej różnicy w prędkości, czy minimalnych różnic w czasie pomiędzy kolejnymi zderzeniami. Pojawia się w układzie chaos, a my, choć potrafimy opisać go równaniami, to nie znamy z wyprzedzeniem konkretnych wyników. Nazywamy go ***chaosem deterministycznym***.

Komputery realizują dokładnie instrukcje programu, w związku z tym nie mają możliwości generowania liczb losowych — nie da się tak napisać programu, aby procesor wykonywał losowe instrukcje. Wynika z tego, że dla każdego algorytmu przy tych samych danych wejściowych zawsze uzyskamy te same dane wyjściowe. **Dlatego wszelkie generatory liczb losowych, jako oparte na algorytmach deterministycznych, umożliwiają wygenerowanie liczb pseudolosowych**, liczb które posiadają tylko pewne cechy liczb losowych. Liczby pseudolosowe zawsze posiadają pewien wzór, według którego są generowane, a osoba znająca algorytm generowania oraz wartość początkową generatora jest w stanie przewidzieć kolejną liczbę losową. Jednak w pewnych zastosowaniach nie jest to przeszkodą i takie liczby mimo wszystko są użyteczne.

Problem deterministycznych algorytmów generowania liczb jest omijany na różne sposoby. Komputery na potrzeby generowania liczb losowych rejestrują różne zdarzenia, np. naciskanie klawiszy przez użytkownika, dostęp do plików, liczbę transmitowanych pakietów przez sieć, a następnie tych przypadkowych danych używają do generowania liczb losowych za pomocą algorytmów deterministycznych. Dzięki temu nawet znając algorytm, nie jesteśmy w stanie przewidzieć kolejnej liczby losowej, gdyż nie znamy dokładnych parametrów początkowych generatora. Niektóre procesory, np. Pentium III i nowsze posiadają wbudowany generator liczb losowych oparty na zjawisku szumu termicznego. Jednak nawet taki, wydawałoby się idealny, generator nie gwarantuje, że uzyskamy losowe ciągi cyfr, gdyż nie ma dowodów na to, że fluktuacje temperatury w procesorze rzeczywiście są losowe[1].

Profesjonalne środowiska programistyczne udostępniają funkcje pseudolosowe o dużej zmienności. Nigdy nie używaj własnych albo słabo udokumentowanych funkcji pseudolosowych, jeśli istnieje odpowiednia funkcja systemowa — raczej nie uda Ci się uzyskać równie dobrych wyników. Na przykład, funkcja *Win API CryptGenrandom* zwraca dane obliczone na podstawie: identyfikatora procesu, aktualnego identyfikatora wątku, liczby taktów zegara od momentu uruchomienia systemu, czasu systemowego, kilkunastu liczników wydajności (np. czasu użycia procesora), dodatkowych informacji o systemie (np. liczby operacji porządkowania pamięci) i wewnętrznych licznikach procesora.

[1] W rzeczywistości są przekonujące dowody (np. teoria superstrun), że te fluktuacje podlegają deterministycznemu opisowi.

Problem ten jest bardziej ogólny i dotyczy różnych zjawisk. Na przykład zachowania ludzi jako jednostek są nieprzewidywalne, jednak zachowanie tłumu statystycznie daje się świetnie opisywać. Podobnie rzeczy, które wcale nie są losowe, w pewnych sytuacjach świetnie przybliżają generator losowy. Wyobraźmy sobie prostą sytuację, w której oczekujemy wygenerowania liczby losowej z zakresu 0 –100. Najprostszym takim generatorem będzie wykorzystanie zegara komputera i zwracanie liczby milisekund. Dopóki użytkownik nie pozna naszego triku, taki generator będzie się doskonale sprawował. Aby zrozumieć, jak działają używane w praktyce algorytmy generowania liczb losowych, musimy wprowadzić nieco teorii.

Generatory liczb pseudolosowych generują sekwencje liczb lub zbiory liczb na podstawie ściśle określonego algorytmu, dając tylko złudzenie ich losowości. Stopień złożoności algorytmu decyduje o jakości generowanych liczb pseudolosowych. Generatory tego typu wymagają pewnej początkowej liczby, **zalążka** (ang. *Seed*), na podstawie którego generują pseudolosowy ciąg (pseudolosowy, gdyż kolejne elementy ciągu determinuje wykorzystana funkcja matematyczna, a nie przypadek). Generowane liczby mogą być użyte do różnych celów, np. realizacji automatów do gier lub symulowania sztucznej inteligencji w programie. Ich podstawową wadą jest okresowość — po pewnym czasie powtarzają się te same sekwencje liczb. Ta cecha eliminuje tego typu generatory ze stosowania w kryptografii. Osoba znająca algorytm wykorzystywany do generowania liczby pseudolosowej jest w stanie przewidzieć generowany ciąg liczb (podobnie jak w przypadku naszego przykładu z zegarkiem — osoba znająca sposób zwracania wyniku i posiadająca stosowny refleks może uzyskać z góry założone wyniki).

W przeciwieństwie do generatorów liczb pseudolosowych**, generatory liczb losowych bazują na danych uzyskanych z otaczającego świata**. Generowane liczby zależą od takich czynników, jak ruchy myszą, sekwencje i czas naciskania klawiszy, i wiele innych. W profesjonalnych zastosowaniach używa się generatorów wykorzystujących niedeterministyczny proces rozpadu pierwiastków promieniotwórczych.

Dla wielu zastosowań potrzebne są ciągi liczb, które mają tylko niektóre własności prawdziwych ciągów liczb losowych. Najbardziej pożądaną cechą są właściwości statystyczne. Uzyskujemy je, wykorzystując generatory liczb pseudolosowych, których jedynym w pełni losowym składnikiem jest zalążek — wartość ustalana podczas uruchomienia generatora. W tym celu można posłużyć się np. odpowiednio zmodyfikowanym czasem zwracanym przez zegar komputera.

Prawie wszystkie stosowane generatory tworzą liczby pseudolosowe na podstawie funkcji rekurencyjnych wykorzystujących operator modulo. Dla dwóch liczb naturalnych n, m wynikiem działania n modulo m (co zapisujemy n mod m) jest reszta z dzielenia liczby n przez liczbę m. Przykładowo: 3 mod 2 = 1, 2015 mod 3 = 2. Obecnie najczęściej stosowanymi generatorami o rozkładzie równomiernym są:

- generatory liniowe,
- generatory rejestrów przesuwnych,
- generatory Fibonacciego,
- generatory oparte na nadmiarowym odejmowaniu,
- generatory nieliniowe.

Żaden generator nie jest idealny, niektóre nadają się lepiej niż inne do konkretnych zastosowań, dlatego generatory należy testować pod względem jakości otrzymywanych ciągów pseudolosowych. Testy takie wymagają wiedzy z zakresu statystyki i rachunku prawdopodobieństwa wykraczającej poza zakres szkoły średniej. Pod koniec rozdziału omówmy tylko jeden, ciekawy przypadek *testu π*.

W następnym podrozdziale poznamy przykłady generatora liniowego i nieliniowego. Osobom, które potrzebują „prawdziwych" liczb losowych, polecamy serwis internetowy *www.random.org*, z którego bezpłatnie można pobrać ciągi losowe.

Algorytmy generatorów liczb pseudolosowych

Generatory liniowe wykorzystują następującą zależność rekurencyjną:

$$x_{n+1} = (a_0 * x_n + a_1 * x_{n-1} + a_k * x_n - k + 1 + c) \bmod m,$$

gdzie a_0, a_1, ..., a_k, c, m są ustalonymi liczbami całkowitymi stanowiącymi parametry generatora, natomiast początkowe wartość x_0, x_0, ..., x_k stanowią zalążek generatora. Jeżeli $c = 0$, to generator taki nazywamy multiplikatywnym, w przeciwnym razie mówimy o generatorze mieszanym.

Zakres liczb, jaki otrzymujemy, stosując generator liniowy, wynosi $\{0...m\}$. Okres generatora liniowego wynosi maksymalnie m, co osiągalne jest jednak tylko dla generatorów mieszanych przy spełnieniu przez parametry wywołania dodatkowych założeń. Generatory liniowe nie dają dobrych wyników, obecnie używa się ich stosunkowo rzadko. Zaletą jest łatwość implementacji i szybkość działania. W typowych implementacjach wykorzystuje się najprostszą postać generatora liniowego:

$$x_{n+1} = (ax_n + c) \bmod m$$

Wszystkie przedstawiane funkcje implementujące generatory liczb pseudolosowych zadeklarujemy następująco: argumentami funkcji będą zalążek i parametry generatora. Zwracaną wartością będzie liczba pseudolosowa, będzie ona stanowiła jednocześnie zalążek dla kolejnej generowanej liczby pseudolosowej. Dodatkowo zakładamy, że operację modulo (n mod m) realizuje niezdefiniowana przez nas w tym rozdziale funkcja ***mod*** zwracająca resztę z dzielenia całkowitego x przez y.

Funkcję realizującą generator liniowy w najprostszej postaci zadeklarujemy następująco:

```
function generatoLiniowy(zalazek, a, c, m : Integer) : real;
```

Przyjmujemy, że funkcja ma zwracać liczby pseudolosowe z przedziału <0, m>. Zanim zdefiniujemy ciało funkcji, przekształcimy zależność rekurencyjną w postać iteracyjną. Rekurencja na kolejne liczby pseudolosowe z zakresu $\{0...m\}$ wygląda następująco:

$$gl(n) = \begin{cases} (a * zalazek + c) \bmod m & \text{dla } n = 0 \\ (a * gl(n-1) + c) \bmod m & \text{dla } n > 0 \end{cases}$$

Ostatecznie definicja funkcji *generatorLiniowy* mogłaby wyglądać jak poniżej:

```
function generatorLiniowy(zalazek, a, c, m : Integer ) : Integer;
 begin
 generatorLiniowy:=(a*zalazek+c) mod m;
 end;
```

Istnieje wiele nieliniowych generatorów liczb pseudolosowych, o których przewadze nad generatorami liniowymi świadczy to, że na podstawie ciągu wygenerowanych liczb pseudolosowych praktycznie nie można odgadnąć parametrów generatora, a co za tym idzie — przewidzieć kolejnych wartości ciągu. Własność taka jest niezbędna w przypadku zastosowań kryptograficznych.

Poniżej omówimy generator *BBS* (*Blum-Blum-Shuba* — nazwa pochodzi od nazwisk twórców). Jak w wielu zagadnieniach związanych z kryptografią, podstawą działania algorytmu są liczby pierwsze. W przypadku generatora BBS parametrami będą dwie różne liczby pierwsze p i q, na których podstawie wyliczane jest $n = p*q$, a wartości pseudolosowe wyliczane są rekurencyjnie według wzoru:

$$x_{k+1} = x_k^2 \bmod n.$$

Przekształcenie rekurencji na iterację zachodzi podobnie jak w generatorze liniowym. Definicja funkcji *generatorBBS* wygląda następująco:

```
function generatorBBS(zalazek, p, q : Integer) : Integer;
 begin
 generatorBBS:=(zalazek*zalazek) mod (p*q);
 end;
```

Spełnienie dodatkowego warunku: $p \bmod 4 = q \bmod 4 = 3$ oraz wybór zalążka innego niż p, q lub n powoduje, że generator ma odpowiednio silne własności kryptograficzne.

Jak wspomnieliśmy na początku rozdziału, generatory liczb pseudolosowych nie są doskonałe i wymagają testów. Jednym z nich jest ***test π***, którego idea polega na tym, że generuje się losowe pary punktów z kwadratu o boku 1, a następnie sprawdza, jaka liczba punktów leży wewnątrz koła wpisanego w ten kwadrat.

Wiemy, że powierzchnia koła wpisanego w kwadrat jednostkowy wynosi π/4, stąd stosunek liczby punktów leżących wewnątrz koła do liczby wszystkich wygenerowanych punktów powinien wynosić π/4. Wartość ta pomnożona przez 4 powinna dać przybliżoną wartość liczby π. Oczywiście, im lepszy generator i większa liczba generowanych punktów, tym przybliżenie π powinno być lepsze. Przypuśćmy, że interesujący nas kwadrat w układzie współrzędnych kartezjańskich będzie miał rogi w punktach o współrzędnych (0,0); (0,1); (1,0); (1,1), wówczas interesujące nas koło możemy opisać nierównością:

$$(x–0,5)2+(y–0,5)2 \le 0,52 = 0,25$$

Funkcja *testPi()* będzie pobierała ciągi liczb pseudolosowych w postaci dwóch tablic równej długości *X[]*, *Y[]*, losowy *i*-ty punkt będzie miał współrzędne *x[i]*, *y[i]*; funkcja powinna zwracać wyliczone przybliżenie liczby π:

```
function testPi (x, y : Array of Real; n : Integer) : Real;
var
i, ilosc : Integer;
temp,a,b : Real;
begin
for i:=0 to n do
begin
a:=x[i]-0.5; b:=y[i]-0.5;
temp:=a*a+b*b;
if temp<=0.25 then ilosc:=ilosc+1;
end;
testPi:=(ilosc/n)*4;
end;
```