СТАТИСТИЧЕСКИЙ АНАЛИЗ СТАНДАРТНЫХ ГЕНЕРАТОРОВ СЛУЧАЙНЫХ ЧИСЕЛ, ИСПОЛЬЗУЕМЫХ В ОПЕРАЦИОННЫХ СИСТЕМАХ MICROSOFT WINDOWS

Ю. С. Хамидулин

Стандартные генераторы случайных чисел, поставляемые вместе с операционными системами Windows (функции rand и rand_s, входящие в библиотеку C Runtime), находят широкое применение в различных прикладных задачах. В данной работе приведен статистический анализ генерируемых ими случайных последовательностей. Показано неудовлетворительное поведение генератора, реализованного функцией rand, и не выявлено серьезных отклонений у генератора, реализованного функцией rand_s.

1. Введение

В настоящее время, в связи с широким распространением компьютеров и их развитием, повсеместно используются различные методы и технологии, использующие генераторы случайных чисел, начиная от моделирования природных явлений и заканчивая имитацией процессов, происходящих в макроэкономике. Но наиболее широкое применение генераторы случайных чисел получили в криптографии и в системах защиты информации. В связи с этим необычайно остро стоит вопрос о качестве получаемых последовательностей случайных чисел, то есть, насколько «случайными» оказываются полученные последовательности. Абсолютно случайная последовательность – это такая последовательность, в которой появление любого из возможных чисел имеет одинаковую вероятность. В связи с этим, возникает необходимость в каких-либо инструментах, позволяющих оценить, насколько последовательность случайных чисел, производимая неким генератором, удовлетворяет понятию абсолютно случайной. Широкое распространение компьютеров и возросшие запросы на генераторы случайных чисел привели к тому, что было разработано бесчисленное множество детерминистических алгоритмов, производящих на первый взгляд достаточно случайную последовательность чисел. Одновременно с этим разрабатывались методы и тесты, позволяющие отсечь «плохие» генераторы от «хороших». Таких тестов и методов существует великое множество, некоторые из которых весьма мощны и рекомендованы для использования при тестировании генераторов. В этой работе используется тест «Стопка книг», предложенный в [1] и показавший свою эффективность. Этот тест был применён для обработки результатов, полученных при использовании двух генераторов псевдослучайных чисел, поставляемых вместе с операционными системами Microsoft Windows в составе стандартной библиотеки CRT (C Runtime). Выбор этих генераторов обусловлен также практическим интересом, так как эти генераторы широко используются в различных прикладных задачах.

Статья поступила в редакцию 30.11.2007.

2. Схема исследования

2.1. Описание метода «Стопка книг»

Более полное изложение алгоритма теста «Стопка книг» может быть найдено, например, в [1]. Тем не менее, дадим краткое описание этого метода, дабы ввести читателя в курс дела и дать необходимые объяснения и обозначения.

Пусть имеется алфавит $A \quad \{a_1,...,a_S\}$, источник, генерирующий символы из этого алфавита, и две гипотезы: $p(a_1) = p(a_2) = ... = p(a_S) = \frac{1}{S}$ (H_0) и $H_1 = \neg H_0$. Для проверки гипотез необходимо протестировать последовательность $x_1x_2...x_n, n \geq 1$, генерируемую источником. Изначально упорядочим все символы алфавита в порядке их определения в алфавите, по номерам индексов от 1 до S. При применении теста «Стопка книг» данный порядок после извлечении каждого символа x_t из генерируемой последовательности меняется в соответствии со следующим правилом:

$$v^{t\ 1}(a) \quad v^t(a) \quad \begin{array}{l} 1, \text{ если } x_t \quad a \\ 1, \text{ если } v^t(a) \quad v^t(x_t) \\ v^t(a), \text{ если } v^t(a) \quad v^t(x_t) \end{array} \qquad (1)$$

где v^t — это порядок расположения символов после обработки последовательности $x_1x_2...x_t$, $t = 1,2,...,n$, $v^t(a)$ — номер элемента a в определённом порядке v^t. Порядок v^1 был определён заранее, до начала теста (например, мы можем определить v^1 следующим образом: $v^1 = \{a_1,...,a_S\}$). Также

заранее, до начала теста, множество всех индексов $\{1,...,S\}$ разбивается на $r, r \geq 2$, групп:

$A_1 \quad \{1,2,...,k_1\},$
$A_2 \quad \{k_1 \quad 1,...,k_2\},$
...
$A_r \quad \{k_{r\ 1} \quad 1,...,k_r\}.$

Затем, тестируя последовательность $x_1x_2...x_n$, мы подсчитываем, сколько номеров $v^t(x_t), t = 1,...,n,$ попало в группу $A_k, k = 1,...,r$. Обозначим это число как n_k или, более формально:

$$n_k = \left|\{t : v^t(x_t) \in A_k; t = 1,...,n\}\right|, k = 1,...,r$$

Итак, если гипотеза H_0 истинна, то вероятность события $v^t(x_t) \in A_k$ равна $\frac{|A_k|}{S}$. Затем, используя критерий хи-квадрат, мы проверим равноценную гипотезу $\widehat{H}_0 : P\{v^t(x_t) \in A_k\} = \frac{|A_k|}{S}$, основываясь на последовательности полученных $n_1,...,n_r$, а также противоположную гипотезу $\widehat{H}_1 = \neg\widehat{H}_0$. Используя полученную последовательность $n_1,...,n_r$, вычислим следующее значение:

$$x^2 = \sum_{i=1}^{r} \frac{\left(n_i - n \cdot \frac{|A_i|}{S}\right)^2}{n \cdot \frac{|A_i|}{S}}.$$

Известно, что величина x^2 асимптотически приближается к распределению хи-квадрат с $(k-1)$ степенью свободы, если гипотеза $\widehat{H}_0$ истинна.

2.2. Реализация алгоритма

Так как последовательность, производимая генератором, может быть достаточно большой (необходимо по крайней мере $10^6 - 10^9$ символов), то операции поиска и перестановки символов в «стопке» могут занять огромное время. Поэтому был разработан «быстрый» вариант реализации алгоритма на основе массива указателей, который не требует:

1. Поиска элемента в последовательности. Скорость доступа к элементу постоянна, ибо индекс извлекаемого символа в алфавите есть индекс в массиве элементов, а доступ к элементам массива по индексу есть величина постоянная и не зависит от размера массива.
2. Удаления и создания объектов или какой-либо перестройки массива. Эти операции – одни из самых дорогостоящих по времени, а частое удаление и создание объектов (особенно таких, которые малы по объёму занимаемой памяти) приводит к дефрагментации памяти и ещё большему снижению производительности. При обработке извлеченного элемента изменяются только значения указателей, а это является простейшей атомарной операцией и не требует никаких затрат памяти.

Однако у данного варианта реализации есть и недостатки. Один из самых главных – большие требования к используемой памяти. При тестировании последовательность случайных данных разбивается на блоки длиной s бит и рассматривается как выборка из алфавита размером $S = 2^s$. Если размер алфавита достаточно большой, порядка 2^{32} и больше, то данный способ не применим из-за нехватки доступной памяти. Как известно, на 32-х разрядных компьютерах под управлением операционной системы семейства Microsoft Windows на каждый процесс отводится 2^{32} байт памяти и поэтому при размере алфавита, большем или равном 2^{32}, не удастся выделить необходимый объём памяти. Перейдём же теперь к краткому описанию данного варианта реализации алгоритма.

Пусть, как и в описании метода «Стопка книг», имеется алфавит $A = \{a_1,...,a_S\}$. В соответствии с этим мы создаём массив размером S элементов:

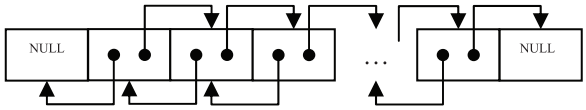

Рис. 1. Изначальное состояние массива

Данная схема изображает массив на начальном этапе тестирования, когда указатели указывают на соседние элементы. Пусть мы тестируем некую последовательность, произведённую генератором, и нам попадается элемент, индекс которого в алфавите равен 2. Тогда после обработки данного элемента массив объектов будет выглядеть следующим образом:

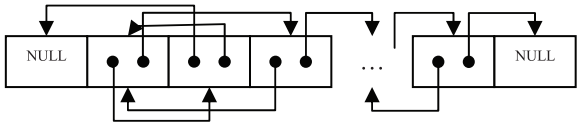

Рис. 2. После обработки элемента с индексом 2.

На рисунке 2 схематично показано, каким образом изменяются указатели. Таким образом, после обработки элемента, имеющего индекс 2 в алфавите, мы получили последовательность индексов алфавита следующего вида (если обходить массив по указателям): 2, 1, 3, … Аналогичным образом обрабатываются все элементы из алфавита S. Из приведенной выше схемы становится понятно, что для обработки одного элемента последовательности случайных данных необходимо всего лишь изменение значений шести указателей и не требуется перестройки структуры массива, то есть элементы массива остаются на своих местах. Читатели, желающие более подробно ознакомиться с деталями реализации алгоритма, могут обращаться к автору.

3. Экспериментальные результаты

Подробное описание генераторов, реализуемых функциями rand() и rand_s(), может быть найдено в

приложении. Сейчас же остановимся на экспериментальных результатах, полученных в ходе тестирования данных генераторов.

В процессе тестирования каждый генератор производил 100 последовательностей случайных данных. Эти генерируемые последовательности разбивались на блоки длиной s бит, и последовательность рассматривалась как выборка из алфавита A размером S 2^s символов. Размер каждой последовательности зависел от размера блока s и от того, какое количество символов алфавита A было необходимо для тестирования. В соответствии с алгоритмом теста «Стопка книг», все символы алфавита A разбивались на группы; количество групп принималось равным 3. Размер блока s постепенно увеличивался с целью выявления статистических закономерностей. Приведённые ниже таблицы показывают экспериментальные данные, полученные при применении теста «Стопка книг» к последовательностям, которые были сгенерированны функциями rand() и rand_s().

Таблица 1. Экспериментальные результаты, полученные при тестировании функции rand().

Размер блока (бит)	Превышение квантиля уровня 0.5	Превышение квантиля уровня 0.95	Размер последовательности в байтах
3	44	2	375000
4	0	100	500000
5	0	100	625000
6	0	100	750000
7	0	100	875000
8	0	100	10^6
12	0	100	$1\dot{p}.5 \quad 10^6$
16	0	100	$2 \cdot 10^6$
20	0	100	$2\dot{p}.5 \quad 10^7$
24	0	100	$3 \cdot 10^8$

Таблица 2. Экспериментальные результаты, полученные при тестировании функции rand_s()

Размер блока (бит)	Превышение квантиля уровня 0.5	Превышение квантиля уровня 0.95	Размер последовательности в байтах
3	49	6	375000
4	48	5	500000
5	43	4	625000
6	49	6	750000
7	44	6	875000
8	53	5	10^6
12	48	3	$1\dot{p}.5 \quad 10^6$
16	49	6	$2 \cdot 10^6$
20	49	7	$2\dot{p}.5 \quad 10^7$
24	51	4	$3 \cdot 10^8$

В идеальном случае, при тестировании абсолютно случайной последовательности, квантили уровня 0.5 и 0.95 превосходятся в среднем в 50 и в 5 случаях из ста соответственно. Как видно из таблиц, генератор, реализуемый функцией rand(), не выдерживает никакой критики. Уже при длине блока в 4 бита генератор не проходит тест. При бóльших длинах блока данный генератор также ведёт себя не лучшим образом. Даже использование более коротких последовательностей (от 100 до 10000 байт) не спасает этот генератор. И в этом случае превышение квантилей уровня 0.5 и 0.95 слишком далеко от идеальных величин. Из этого теста становится видно, что использование данного генератора в приложениях может быть чревато различного рода логическими ошибками и неточностями, вызванными неудовлетворительным качеством производимой генератором псевдослучайной последовательности.

Совсем другое поведение показал следующий генератор, реализуемый функцией rand_s(). Фактически, при его тестировании не было обнаружено каких-либо существенных отклонений. Псевдослучайные последовательности, получаемые при использования данного генератора при различных размерах блока, вплоть до 24, хорошо проходят тест, что может служить достаточно веским основанием для использования этого генератора повсеместно вместо более широко известной (и используемой) функции rand(). Единственное, в чем rand() превосходит rand_s(), так это в скорости генерации последовательности псевдослучайных чисел. Эти скорости разнятся буквально на порядок. Тем не менее, даже если требования к скорости выполнения являются критичными, использование функции rand() не рекомендуется, гораздо более лучшим выходом в данной ситуации будет использование своей собственной либо сторонней реализации какого-либо генератора, более быстрого, чем rand_s(), и производящего более случайную последовательность, чем rand().

Приложение

I. Генератор псевдослучайных чисел, реализуемый функцией rand().

В стандарте ANSI-C имеется функция rand(), выдающая равномерно распределённое число от 0 до RAND_MAX, и связанная с ней функция srand(), производящая начальную установку счётчика. Описание данных функций можно найти в файле <stdlib.h>. Почти все подобные генераторы используют рекуррентную последовательность вида:

$$I(n+1) = [a \cdot I(n) + c] \bmod m,$$

где число *a* называется мультипликатором, число *c* инкрементом, а число *m* – модулем. Такой выбор может послужить серьёзным ограничением на статистические свойства последовательности случайных чисел. Кроме того, в большинстве случаев RAND_MAX = 35767, что значительно меньше, чем диапазон изменения целых чисел. В некоторых испытаниях теория рекомендует $10^6 - 10^9$ случайных

проб, но, пользуясь подобным счетчиком, можно получить не более RAND_MAX одинаковых случайных чисел, т.е. в 30 тыс.- 30 млн. раз меньше рекомендуемого. Генератор ANSI-C был опубликован комиссией как 'пример'.

```
/* (в модуле stdlib.h) */
#define RAND_MAX 32767

/* пример от комитета ANSI-C */
unsigned long next=1;

int rand(void) {
 next=next*1103515245+12345;
 return((unsigned int)(next/
65536)%32768);
}

void srand(unsigned int seed) {
 next=seed;
}
```

Мультипликатор и инкремент этого примера (который, скорее всего и поставляется со стандартной библиотекой C Runtime) не являются оптимальными, что подтверждается тестами.

II. Генератор псевдослучайных чисел, реализуемый функцией rand_s().

Если генератор, реализуемый функцией rand(), закреплён в стандарте ANSI-C и его использование не зависит от операционной системы, то генератор, реализуемый функцией rand_s(), является специфичной функцией операционной системы Windows и её использование не является платформенно-независимым. Функция rand_s() генерирует числа в диапазоне от 0 до UINT_MAX (на 32-х битных платформах эта константа равна 232) и не использует никаких функций инициализаций (наподобие srand() у функции rand()). Как сказано в документации, данная функция использует возможности операционной системы для генерации псевдослучайных чисел.

Литература

1. Рябко Б. Я., Фионов А. Н. Криптографические методы защиты информации: учебное пособие для вузов. – М.: Горячая линия–Телеком, 2005. – 229 с.
2. Рябко Б. Я., Пестунов А.И. «Стопка книг» как новый статистический тест для случайных чисел // Проблемы передачи информации. – 2004. – Т.40, №1. – с.73-78.
3. Кнут Д.Э. Искусство программирования. В 3 т.: Перевод с англ. Т.2: Получисленные алгоритмы – 3-е издание – М.: Издательский дом «Вильямс», 2003. – 828 с.

Хамидулин Юрий Сергеевич

аспирант кафедры ПМиК СибГУТИ, тел. (41136) 9-12-86, e-mail: tatarin@gmail.com.