
Лекция 7

Генерация псевдослучайных чисел



Вопросы

1. Принципы генерирования случайных и псевдослучайных чисел
2. Линейные конгруэнтные последовательности
3. Числовые последовательности максимальной длины



Возможные сферы применения случайных чисел

- ❑ **Моделирование** (приближают модель к реальности)
- ❑ **Выборка** (если проверка всех вариантов нереальна)
- ❑ **Численный анализ** (интегрирование, методы Монте-Карло)
- ❑ **Программирование** (источник тестовых данных)
- ❑ **Принятие решений** (если детерминированность приводит к замедлению процесса)
- ❑ **Развлечение** (карты, рулетка, ...)



«Случайные» числа

- «2» - случайное число или нет?
- «Случайность» проявляется в «последовательностях независимых случайных чисел»:
 - Каждое число получено самым произвольным образом;
 - Без связи с другими членами последовательности;
 - Может оказаться в последовательности на любом месте



Старые методы получения С.Ч.П.

- ❑ 1927 г. Использование таблиц С.Ч. (произвольно взятых из отчетов)
- ❑ 1939 г. Машина, механически вырабатывающая С.Ч. – 100 000 С.Ч.
- ❑ 1955 г. RAND Corporation – таблицы из 1 000 000 С.Ч., полученных машиной
- ❑ 1946 г. Алгоритмы. Дж. фон Нейман. Метод «середины квадрата»



Метод «середины квадрата»

- ❑ С.Ч. - возводится в квадрат
- ❑ Из результата извлекаются средние цифры
- ❑ 10-значные числа:
5772156649 -> 33317**7923805949**09201
- ❑ ??!! Неслучайный характер последовательности
- ❑ **Проблема** - появление коротких циклов
- ❑ **Псевдослучайные** или **квазислучайные** числа



-
- Обычно интересуют С.Ч., равномерно распределенные на $[0,1)$
 - Проблемы работы с **дробными** числами
 - \Rightarrow генерируют **целые** числа X_n на $[0, m)$
 $U_n = X_n / m$ попадает в $[0,1)$
(обычно m – размер машинного слова)



Линейный конгруэнтный метод (схема предложена Д.Лехмером (1949))

□ Задаются числа:

- m , модуль, $m > 0$
- a , множитель, $0 \leq a < m$
- c , приращение, $0 \leq c < m$
- X_0 , начальное значение, $0 \leq X_0 < m$

□ Желаемая последовательность случайных чисел $\{X_n\}$:

- $X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0 -$

□ Линейная конгруэнтная последовательность



Пример и проблемы

- $m = 10, X_0 = a = c = 7$
- $7, 6, 9, 0, 7, 6, 9, 0, \dots$
- «Плохой» набор m, X_0, a, c ??? Да !!!
- Конгруэнтная последовательность ВСЕГДА образует «петли» - \exists цикл, повторяющийся ∞ число раз. (Свойство всех последовательностей вида $X_{n+1} = f(X_n)$).
- Задача – сделать это цикл – **max** длины
- Т.е. задача выбора параметров m, X_0, a, c



$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0$$

- При $c=0$ – мультипликативный датчик:
 $X_{n+1} = (aX_n) \bmod m$
- При $c=1$ – смешанный датчик

- При $a=1$ – последовательность не случайная: $X_n = (X_0 + nc) \bmod m$
- При $a=0$ – еще хуже :)



$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0$$

□ При $a \geq 2$

$$X_{n+k} = (a^k X_n + (a^k - 1)c/b) \bmod m, \\ k \geq 0, n \geq 0$$

□ Для каждого k -го члена $\{X_{i+k}\}, i=1,2,3, \dots$ – линейная конгруэнтная последовательность с $a = a^k, c = (a^k - 1)c/b$



Выбор модуля m



Выбор модуля m

- m – должно быть достаточно большим, т.к. период не больше m
 - Даже для последовательности из 0,1 нельзя брать $m=2$!!!!
- Вычисления $(aX_0+c) \bmod m$ должны проходить быстро.
- Удобно использовать $m =$ (длине слова компьютера): 2^e для e -разрядной машины
- Однако лучше $2^e \pm 1$: $2^e - 1$ или $2^e + 1$



$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0$$

m = длина машинного слова

Почему для m лучше $2^e \pm 1$, а не 2^e ?

«При $m = 2^e$ младшие цифры числа X_n намного менее случайны, чем его старшие цифры»

???



$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0$$

- Если d – делитель m и $Y_n = X_n \bmod d$, то
 $Y_{n+1} = (aY_n + c) \bmod d$

- Если $m = 2^e$, то 4-е младших бита X_n :
 $Y_n = X_n \bmod 2^4 = X_n \bmod 16$,
и образуют конгруэнтную последовательность с
периодом не больше 16 (!!!!)

- Старшие же биты X_n ведут себя случайно



Выбор модуля m

- $m = 2^e \pm 1$
- $2^{32} - 1 = 3 * 5 * 17 * 257 * 65537$:
младшие члены последовательности не совсем случайны по mod 3, 5, 17, 257
- $2^{32} + 1 = 641 * 6700417$

- $m =$ наибольшее простое, меньше 2^e :
неплохо !!!



Выбор множителя a

- Цель – получать период последовательности **Max** длины
- Пример. Неслучайная последовательность с очень большим периодом:
 $a=c=1$.
 $X_{n+1}=(X_n+1) \bmod m$. Период = m
- Период длины m – достижим. И в этом случае выбор X_0 не влияет на длину периода.



Выбор множителя a

- Теорема. Длина периода Линейной Конгруэнтной Последовательности равна m т. и тт., когда
 - s и m – взаимно простые числа;
 - $b = a - 1$ кратно p для любого простого p , являющегося делителем m ;
 - b кратно 4, если m кратно 4



Выбор множителя a

- Пример. $m=5, c=7, a=6$

$$X_{n+1} = (6X_n + 7) \bmod 5, \quad n \geq 0$$

- $X_0 = \mathbf{1}$ - любое из $[0, 4]$

$$X_1 = 13 \bmod 5 = 3$$

$$X_2 = 25 \bmod 5 = 0$$

$$X_3 = 7 \bmod 5 = 2$$

$$X_4 = 19 \bmod 5 = 4$$

$$X_5 = 31 \bmod 5 = \mathbf{1}$$



Выбор множителя a

- Максимальный период – при $b=a-1$, кратном всем простым делителям m (также b кратно 4, если m кратно 4)
- Если \mathbf{z} – основание системы счисления, которое используется в машине, \mathbf{e} – разрядность, (\mathbf{m} – размер слова, $m = z^e$), то

$$a = z^k + 1, 2 \leq k < e$$

$$X_{n+1} = ((z^k + 1) X_n + 1) \bmod z^e$$



Реализация метода ЛКМ в ANSI-C

```
#define RAND_MAX 32767
unsigned long next=1;
int rand(void)
{ next=next*1103515245+12345;
  return((unsigned int) (next/65536)%32768);
}
void srand(unsigned int seed)
{ next=seed; }
//-----
D.Knuth, H.Lewis: a= 1664525, c= 1013904223,
  m=232: next=1664525*next+1013904223;
```



Реализация на Python

```
import math
```

```
def lkm():
```

```
    global x
```

```
    x = (1664525*x+1013904223) % math.pow(2, 32)
```

```
    return int(x)
```

```
x=1 # start meaning of X
```

```
for i in range(0,10):
```

```
    print lkm()
```



Реализация на Pascal

```
Const x :LongWord = 10;
Function LCM:LongWord;
Begin
  x = (1664525*x+1013904223) mod exp(32*ln(2))
  LCM = x
End;
Begin
  writeln(LCM)
End.
```



Другие методы

Различные модификации Линейного Конгруэнтного Метода



-
- Расширенный конгруэнтный генератор

$$x_n = 2^{13}(x_{n-1} + x_{n-2} + x_{n-3}) \bmod 2^{32} - 5 \quad \text{период } 2^{96}$$

- Инверсивный конгруэнтный генератор

$$y_n = a((y_{n-1})^{-1} + c) \bmod m \quad \text{период } m/2$$

- Системный генератор MS Fortran

$$x_n = 48271x_{n-1} \bmod 2^{31} - 1$$



Генераторы на основе умножения с переносом

□ Период 2^{160}

$$x_n = 2111111111x_{n-4} + 1492x_{n-3} + 1776x_{n-2} + 5115x_{n-1} + \text{carry} \mod 2^{32}$$

□ Период $a \cdot 2^{31} - 1$

$$x_n = ax_{n-1} + \text{carry} \mod 2^{32}$$



Комбинированные генераторы

Генератор на основе объединения путем сложения по $\text{mod } 2^{32}$ двух генераторов:

- запаздывающего генератора Фибоначчи и
- генератора на основе умножения с переносом

$$x_n = x_{n-99}x_{n-33} \text{ mod } 2^{32}$$

$$y_n = 30903y_{n-1} + \text{carry} \text{ mod } 2^{16}$$

Период 2^{127}



-
- Генератор на основе объединения путем конкатенации двух 16-битных генераторов на основе умножения с переносом. Период 2^{16}

$$x_n = ax_{n-1} + \text{carry} \bmod 2^{16}$$
$$y_n = by_{n-1} + \text{carry} \bmod 2^{16}$$





Другие методы

- Квадратичный конгруэнтный метод
$$X_{n+1} = (dX_n^2 + aX_n + c) \pmod{m}$$

- Период длины m т. и т.т., когда
 - c и m взаимно простые числа;
 - d и $a-1$ кратны всем нечетным простым делителям m ;
 - d четное и $d \equiv a-1 \pmod{4}$, если m кратно 4;
 - $d \equiv a-1 \pmod{2}$, если m кратно 2;
 - или $d \equiv 0$ или $a \equiv 1$ и $cd \equiv 6 \pmod{9}$, если m кратно 9



Другие методы

□ Если $m = 2^e$ – квадратичный метод Конвэя.

□ Пусть $X_0 \bmod 4 = 2$

$$X_{n+1} = X_n (X_n + 1) \bmod 2^e \quad n \geq 0$$

□ Почти идентичен методу «середины квадрата»



Последовательностью Фибоначчи

- Зависимость X_{n+1} от более чем одного из предыдущих значений

$$X_{n+1} = (X_n + X_{n-1}) \bmod m$$

$$X_{n+1} = (X_n + X_{n-k}) \bmod m$$

Длина периода больше m

Но числа «недостаточно случайны»





Некоторые выводы. «Псевдослучайные» числа

- «Псевдослучайные» - из того, что если известно i -е число, то по формуле мы однозначно вычислим $(i+1)$ -й элемент.
- Из рекуррентной природы формулы следует, что при одном и том же x_0 мы получим при повторной генерации ту же самую последовательность.
- Все арифметические алгоритмы генерации псевдослучайных последовательностей периодичны, то есть существует некоторое $p \in \mathbb{N}$, для которого выполняется $x_i = x_{i + pn}$ при любом натуральном n .



Некоторые выводы. «Псевдослучайные» числа

- Любой генератор псевдослучайных последовательностей должен быть инициализирован случайной величиной, неким внешним источником случайных значений.
- Такими «аппаратными» генераторами в ПК могут выступать, например, электрические шумы в полупроводниковых устройствах, а также текущее количество выполненных процессором тактов или текущее значение времени.



Недостатки классических генераторов псевдослучайных чисел

- ❑ сравнительно короткий период генерируемой последовательности
- ❑ зависимость между соседними последовательными значениями
- ❑ неравномерность распределения значений



Комбинированные генераторы

- {данная функция использует три генератора для возврата одного случайного числа}

```
function CombRandom: real;
var
  f: real;

begin
  f := Ran2;
  if f>0.5 then CombRandom := Random
  else CombRandom := Ran1; { случайный выбор
                             генератора }
end; {CombRandom}
```



```
Var a2,a1: integer;
```

```
function Ran1: real;  
  var  
    t: real;  
  begin  
    t := (a1*32749+3)  
          mod 32749;  
    a1 := Trunc(t);  
    Ran1 := Abs(t/32749);  
  end; {Ran1}
```

```
function Ran2: real;  
  var  
    t: real;  
  begin  
    t := (a2*10001+3)  
          mod 17417;  
    a2 := Trunc(t);  
    Ran2 := Abs(t/17417);  
  end; {Ran2}
```





Инициализация генератора ПСП

Чтение текущего значения миллисекунд

- Вызвать функцию «2Ch» прерывания 21h операционной системы DOS и получить «почти» случайное число на отрезке [0..99].

MOV AH, 2Ch ;	номер функции получения времени
INT 21h ;	получаем время
MOV HUNDREDS, DL ;	получаем сотые доли текущей секунды из DL



Инициализация генератора ПСП

Чтение значения счетчика тактов процессора

- Начиная с линейки процессоров Pentium в архитектуре x86 появилась инструкция, позволяющая прочитать счетчик тактов процессора с момента последнего сброса.
- Для инструкции *rdtsc* (Read Time Stamp Counter) задан машинный код `0F 31`. 64-битное значение счетчика возвращается в паре регистров `<EDX:EAX>`.
- Если для инициализации генератора достаточно 32-битного значения, то необходимо использовать наиболее «чувствительные» младшие 32 бита счетчика тактов.
- При использовании старого компилятора языка ассемблера для обращения к 32-битному регистру *EAX* понадобится вручную проставить префикс с кодом `66h`.



Инициализация генератора ПСП

Чтение значения счетчика тактов процессора

; Чтение значения младших 32 битов счетчика тактов процессора в пару <DX:AX>

DB 0Fh, 31h ;	машинный код инструкции RDTSC, результат в <EDX:EAX>
MOV CL, 16 ;	32-битное значение EAX помещаем в пару <DX:AX>
DB 66h ;	= ROR EAX, CL - циклический сдвиг старшей части EAX в младшую
ROR AX, CL	
MOV DX, AX ;	старшие 16 бит копируются в DX
DB 66h ;	= ROR EAX, CL - возвращаем младшие биты на место
ROL AX, CL	



Современные генераторы ПСП



Алгоритм MT19937. Виток Мерсена (Вихрь Мерсена, Mersenne Twister)

- ❑ Макото Мацумото и Такуджи Нишимура (1997 год)
- ❑ Числа Мерсена: простые вида $2^p - 1$
- ❑ Цель алгоритма – иметь огромный период ($2^{19937} - 1$)
- ❑ Имеет высокий порядок пространственного эквираспространения, равный 623, т.е. корреляция между последовательными значениями в выходной последовательности пренебрежимо мала
- ❑ Алгоритм статистически случаен во всех выходных битах, и проходит строгие Тесты **DIEHARD** (<http://stat.fsu.edu/pub/diehard/>).



Mersenne Twister Home Page

- A very fast random number generator of period $2^{19937}-1$
- <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html>



Текст программы

Посмотрим текст программы



VBS. Алгоритм Блюма-Блюма-Шуба (1986)

Ленор Блум, Мануэль Блум и Майкл Шуб алгоритм генерации ПСП, стойкий к обратным преобразованиям

- Основная рекуррентная формула алгоритма: $x_n = (x_{n-1})^2 \bmod pq$, где p и q — два больших простых числа. $z_n = \text{parity}(x_n)$
- Два простых числа, p и q , должны быть оба сравнимы с 3 по модулю 4 ($p \equiv 3 \pmod{4}$, $q \equiv 3 \pmod{4}$) и НОД($\varphi(p-1)$, $\varphi(q-1)$) должен быть мал (это увеличивает длину цикла).
- Для повышения качества получаемой последовательности на очередном шаге выбираются не все биты x_n , а только $\log(\log(M))$ младшие, или даже только бит четности. Из полученных «случайных битов» формируются двоичные псевдослучайные числа произвольной разрядности.
- Имеется возможность вычислить x_n без генерации предыдущих членов последовательности: $x_n = (x_0)^{2^n \bmod (p-1)(q-1)} \bmod pq$,
- Данный алгоритм более требователен к вычислительным ресурсам, но, с другой стороны, обладает хорошими статистическими характеристиками.



BBS. Алгоритм Блюма-Блюма-Шуба (1986)

- ❑ Этот генератор подходит для [криптографии](#), но не для моделирования, потому что он недостаточно быстр.
- ❑ Однако, он имеет необычно высокую стойкость, которая обеспечивается качеством генератора исходя из [вычислительной сложности](#) задачи [факторизации](#) чисел.
- ❑ Когда простые числа выбраны аккуратно, и $O(\log\log M)$ бит каждого x_n являются выходными данными, тогда предел взятый как M быстро растёт, и вычисление выходных бит будет настолько же трудно, как и факторизация M .
- ❑ Если факторизация целых чисел так трудна (как предполагается), тогда BBS с большим M будет иметь выход, свободный от любых неслучайных узоров, которые могут быть выявлены при достаточном объёме вычислений. Однако, возможно появление быстрого алгоритма для факторизации, и вследствие этого BBS не является гарантированно надёжным.



Алгоритмы «xor-shift»

Здесь также демонстрируется техника
«умножения с переносом»



Алгоритм "Marsaglia-Multicarry» (Джордж Марсаглия, у-н Флорида)

Использует метод **умножения с переносом**. Достаточно быстрый и имеет различные периоды начиная с 2^{60} . Недостаток - использование умножения

```
unsigned long mwc() // период около  $2^{125}$ 
{
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);
}
```



Алгоритм «xor-shift». Очень быстрый генератор. Джордж Марсаглия

```
□ Const x : LongWord = 3456789; /* Паскаль */  
y : LongWord = 62436069;  
z : LongWord = 1288629;  
w : LongWord = 675123;
```

```
function xor128: LongWord;  
  var t : LongWord;  
  begin  
    t := (x xor (x shl 11));  
    x := y;  
    y := z;  
    z := w;  
    xor128 := w = (w xor (w shr 19)) xor (t xor (t shr 8));  
  end;
```



```
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)));
}
```



Комбинация «xor-shift» с линейным конгруэнтным алгоритмом и алгоритмом Фибоначчи с запаздываниями

```
program Sample;
  var x : LongWord = 123456789;
      y : LongWord = 362436000;
      z : LongWord = 521288629;
      c : LongWord = 7654321;
  function Random() : LongWord;
    var t : int64;
  begin
    x := int64(69069)*x + 12345;
    y := y xor (y shl 13);
    y := y xor (y shr 17);
    y := y xor (y shl 5);
    t := int64(698769069)*z + c;
    c := t shr 32;
    z := t;
    Random := x + y + z;
  end;
begin {...} writeln ( Random() ); end.
```



M-последовательности



M-последовательности

- ❑ **M-последовательности** или **последовательности максимальной длины** (англ. *Maximum length sequence*, **MLS**) — псевдослучайные последовательности, нашедшие широкое применение в широкополосных системах связи. Как правило, используются двоичные M-последовательности, члены которых состоят из чисел 1 и 0
- ❑ Могут быть легко реализованы аппаратно на основе регистров сдвига и XOR-элементов
- ❑ Применяются, в частности, в мобильной связи (стандарт CDMA)



Генерация M-последовательностей

□ Реализуются на основе
N-разрядных регистров сдвига

□ Задаются полиномом вида

$$G(X) = g_m X^m + g_{m-1} X^{m-1} + g_{m-2} X^{m-2} + \dots + g_2 X^2 + g_1 X + g_0$$

где X^m – m-й бит регистра сдвига

$g_m = 0$ или 1

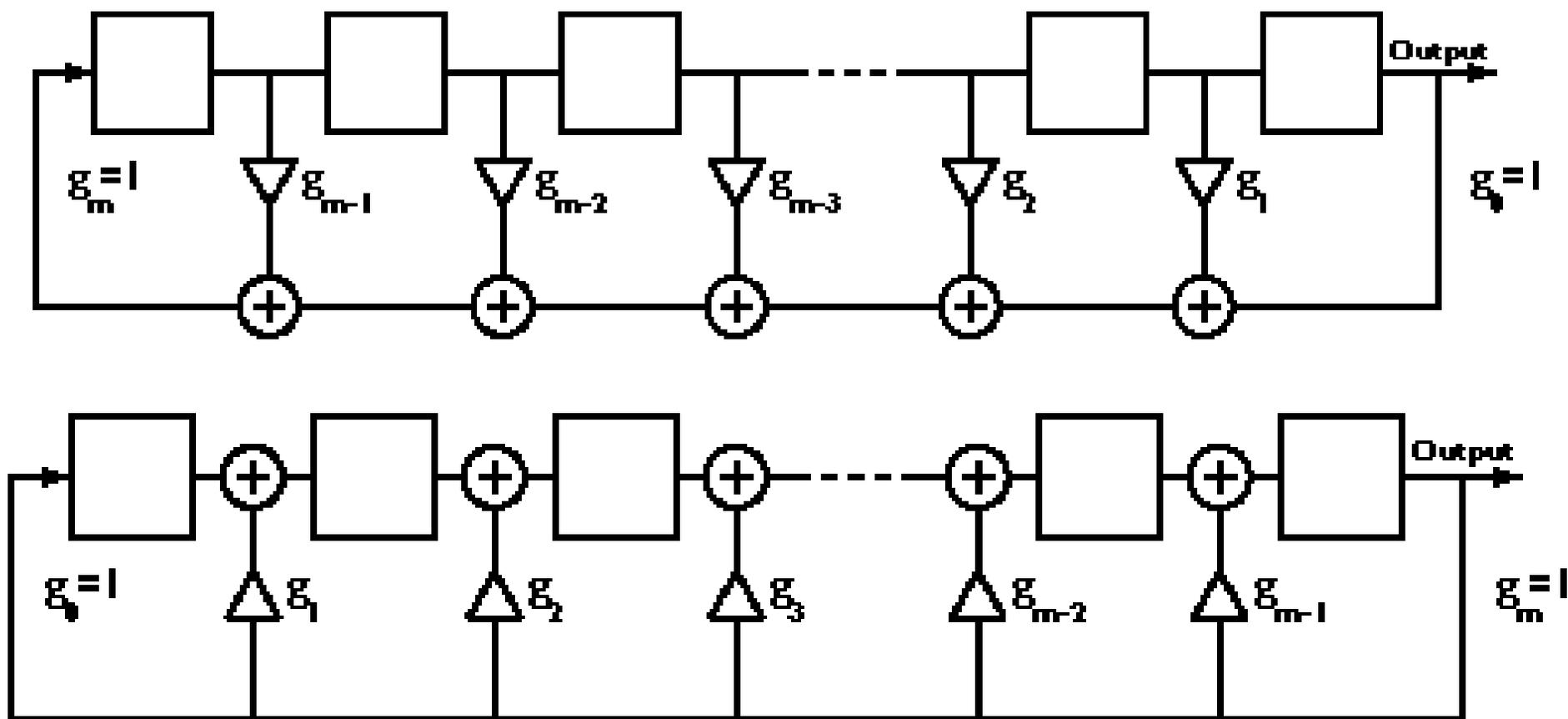


M-последовательности

- 1) M-последовательности являются периодическими с периодом $N = 2^n - 1$;
- 2) количество символов, принимающих значение 1, на длине одного периода M-последовательности на 1 больше, чем количество символов, принимающих значение 0;
- 3) любые комбинации символов длины n на длине одного периода M-последовательности за исключением комбинации из n нулей встречаются не более одного раза. Комбинация из n нулей является запрещённой: на её основе может генерироваться только последовательность из одних нулей;
- 4) сумма по mod 2 любой M-последовательности с её произвольным циклическим сдвигом также является M-последовательностью;



Генерация M-последовательностей (описания Фибоначи, Галуа)



Выбор коэффициентов

□ Файлы коэффициентов:

Stages – размер регистра	Taps – число отводов
5 stages, 2 taps: (1 set) [5, 3]	5 stages, 4 taps: (2 sets) [5, 4, 3, 2] [5, 4, 3, 1]

□ Переход от коэффициентов Галуа к Фиббоначи: N-g (кроме старшего)

[5, 4, 3, 1] -> [5, 1, 2, 4]



Программная реализация

5 stages, 4 taps: [5, 4, 3, 2]

1. Коэффициенты в биты: 00011110
2. 00011110 -> 10с/с -> 30 : полином
3. unsigned int polynom=30,rand=0;
....
 if(rand&1)
 {rand=(rand>>1)^polynom;}
 else {rand=(rand>>1);};
4. Повторять 2^5-1 раз (длина ПСП - 31)



Как бы все у нас
по этому поводу ...

