**УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет математики, информационных и авиационных технологий

Кафедра информационной безопасности и теории управления

**КУРСОВАЯ РАБОТА**

На тему «КРИПТОАНАЛИЗ ПОТОКОВОГО ШИФРА НА ОСНОВЕ АЛГОРИТМА БЕРЛЕКЭМПА-МЕССИ»

Специальность 10.05.01 «Компьютерная безопасность»

Специализация: «Математические методы защиты информации»

Студент (ка) 3 курса

Группа \_КБ-О-19/1\_\_\_\_

Лавриненко Анна Дмитриевна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО полностью подпись

Руководитель:

Рацеев Сергей Михайлович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ФИО полностью подпись

УЛЬЯНОВСК **–** 2022

**ОГЛАВЛЕНИЕ**

[Список используемых сокращений 3](#_Toc102038407)

[Введение 4](#_Toc102038408)

[1. Описание и обоснование алгоритма Берлекэмпа—Месси 5](#_Toc102038409)

[1.1. Регистры сдвига с линейной обратной связью 5](#_Toc102038410)

[1.2. Алгоритм Берлекэмпа-Месси 6](#_Toc102038411)

[1.3. Примеры 10](#_Toc102038412)

[1.3.1. Примеры для алгоритма 1. 10](#_Toc102038413)

[1.3.2. Примеры для алгоритма 2. 14](#_Toc102038414)

[2. Криптоанализ потокового шифра на основе алгоритма Берлекэмпа-Месси 18](#_Toc102038415)

[2.1. Потоковые шифры 18](#_Toc102038416)

[2.2. Криптоанализ потокового шифра 19](#_Toc102038417)

[3. Реализация алгоритма Берлекэмпа-Месси 20](#_Toc102038418)

[Заключение 21](#_Toc102038419)

[Список использованной литературы 22](#_Toc102038420)

[ПРИЛОЖЕНИЕ А 23](#_Toc102038421)

[ПРИЛОЖЕНИЕ Б 29](#_Toc102038422)

# **Список используемых сокращений**

Алгоритм Берлекэмпа-Месси – алгоритм БМ;

Linear Feedback Shift Register (регистр сдвига с обратной линейной связью) — LFSR;

# **Введение**

Потоковый шифр – это симметричный тип шифра, где каждый элемент открытого текста переводится в зашифрованный вид, в зависимости от применяемого ключа и его позиции в текстовом потоке. Каждый бит данных зашифровывается посредством гаммирования. Процесс гаммирования подразумевает наложение на информацию гамм кода по строго определенным правилам. Чтобы расшифровать данные, требуется наложение той же гаммы на зашифрованный текст [1]. Наиболее популярный способ генерации гаммы — линейные сдвиговые регистры с обратной связью.

Алгоритм Берлекэмпа-Месси предназначен для оценки двоичных кодов BCH. Берлекэмп опубликовал свой алгоритм в 1968 году в качестве элемента конструкции декодера кодов Боуза-Чоудхудри-Хоквингема над конечным полем [2], а вскоре за ним в 1969 году Месси предложил свою интерпретацию алгоритма [3], который позволяет строить линейный регистр сдвига минимальной длины. Этот алгоритм позволяет, используя лишь небольшую часть закодированного сообщения, декодировать его.

Рассматриваемый алгоритм находит применение при декодировании различных классов кодов: кодов Рида–Соломона, кодов БЧХ, циклических и обобщенных циклических кодов, кодов Гоппы, и алгебро-геометрических кодов (вернее, некоторых их подклассов). Также он используется в криптографии, при тестировании псевдослучайных последовательностей и для быстрого вычисления в конечных полях.

Целью данной курсовой работы является исследование алгоритма Берлекэмпа—Месси, его вариаций, а также криптоанализ потокового шифра на основе алгоритма Берлекэмпа-Месси.

В данной работе будут рассмотрены задачи:

1) описание и обоснование алгоритма Берлекэмпа—Месси;

2) на конкретных примерах рассмотреть работу алгоритма Берлекэмпа-Месси;

3) программно реализовать данный алгоритм на языке С++;

4) криптоанализ потокового шифра на основе алгоритма Берлекэмпа-Месси.

# **1. Описание и обоснование алгоритма Берлекэмпа—Месси**

## **1.1. Регистры сдвига с линейной обратной связью**

Регистр сдвига с линейной обратной связью (LFSR) — это сдвиговый регистр битовых слов, у которого значение входного бита однозначно задается некоторой функцией, исходя из значений остальных битов регистра до сдвига.

Регистр, изображенный на рисунке 1, состоит из ряда ячеек, которые устанавливаются вектором инициализации (чаще всего секретным ключом). Работа регистра определяется наличием или отсутствием связи от каждого разряда к обратной связи. Отводы регистра с обратной связью не фиксированы, а являются частью ключа. На очередном шаге содержимое ячеек регистра сдвигается на одну позицию вправо, а один бит, оставшийся в результате операции XOR свободным, помещается в крайнюю левую ячейку [4].

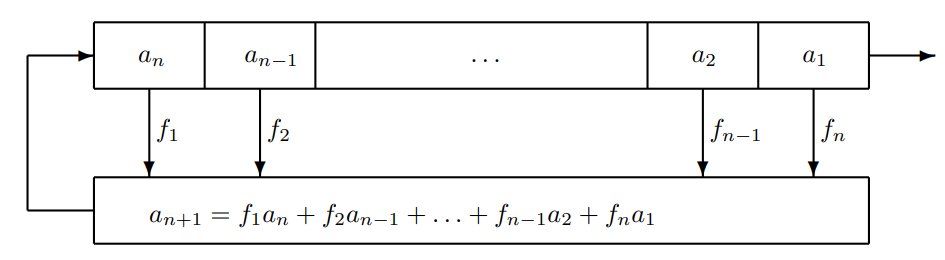


Рисунок 1 – Регистр сдвига с линейной обратной связью

Регистры сдвига с линейной обратной связью используются в криптографии как инструменты для повышения эффективности шифров.

Наиболее криптостойкие последовательности дает сжимающий генератор (shrinking generator), основанный на простом взаимодействии между результатами двух регистров LFSR. Биты одного результата определяют, будут ли соответствующие биты второго результата использоваться как часть полного «потокового ключа». Сжимающий генератор прост, масштабируем и имеет хорошие защитные свойства, однако скорость генерации «потокового ключа» не будет постоянной, если не принять некоторых предосторожностей.

## **1.2. Алгоритм Берлекэмпа-Месси**

Задача алгоритма - найти регистр сдвига с линейной обратной связью, который при соответствующих начальных условиях порождает заданную последовательность и является при этом кратчайшим.

Лучшим подходом к алгоритму Берлекэмпа-Месси является интерпретация матричного уравнения

как описания рекурсивной процедуры. Предположим, что вектор известен. Члены линейной рекуррентной последовательности вычисляются с помощью уравнения:

Изначально на элементы последовательности каких-либо ограничений не накладывается. Произвольный LFSR можно задать многочленом обратных связей и длиной регистра. Длина регистра может быть больше степени многочлена [5].

Пусть произвольный LFSR задается многочленом обратной связи, с длиной регистра . Тогда для построения регистра сдвига надо определить две величины, обозначаемые как пара , при условии . Данный LFSR должен порождать заданную последовательность и иметь минимальную длину .

Рассматриваемая процедура построения является рекурсивной. Для каждого он строит LFSR, порождающий последовательность . LFSR минимальной длины, порождающий последовательность обозначим через пару . Такой регистр не обязательно должен определяться однозначно, так как возможно существование нескольких LFSR минимальной длины. К началу -го шага имеется совокупность LFSR:

.

Алгоритм Берлекэмпа-Месси вычисляет новый LFSR минимальной длины, генерирующий последовательность . Для этого используется самый последний из вычисленных LFSR, в котором, при необходимости, модифицируется длина и весовые множители в отводах. На -м шаге алгоритма вычисляется элемент с номером на выходе -го регистра сдвига:

Если , то полагаем и завершаем этим самым -й шаг алгоритма.

Если , то изменим весовые множители в LFSR по правилу:

где число удовлетворяет условию .

Теорема 1 [6]. Пусть — последовательность LFSR минимальной длины, таких, что генерирует последовательность . Если для некоторого , выполнено то

.

Замечание 1. Пусть для некоторого выполнено неравенство . Тогда возможны два случая.

1. . Тогда из теоремы 1 следует, что . Поэтому .

2. .. Тогда . Поэтому .

Учтем данное замечание при описании алгоритма Берлекэмпа-Месси. В следующем алгоритме многочлен отвечает за многочлен LFSR , а многочлен  — за многочлен LFSR .

Алгоритм 1.

Вход: последовательность над некоторым полем .

Выход: LFSR минимальной длины , для которого:

1. Определить , , , .
2. Цикл :
   1. Определить
   2. Если , то .
   3. Если :
      1. Если :
      2. Иначе (т.е. выполнено ):

В алгоритме 1 можно минимизировать число вычислений обратных элементов в поле , т.е. вычислений вида . Если для некоторого унитарного многочлена выполнено равенство

то для любого выполнено равенство

где Поэтому алгоритм 1 можно записать в следующем виде.

Алгоритм 2.

Вход: последовательность над некоторым полем .

Выход: LFSR минимальной длины , для которого:

1. Определить , , , , .
2. Цикл :
   1. Определить
   2. Если , то .
   3. Если :
      1. Если :
      2. Иначе (т.е. выполнено ):
3. .

## **1.3. Примеры**

### **1.3.1. Примеры для алгоритма 1.**

Пример 1. Решим систему линейных уравнений над полем GF (5):

Алгоритм решения системы приведен в таблице 1 на основе алгоритма 1:

Таблица 1 — Алгоритм решения системы уравнений над полем GF (5)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 |  | 1 | 1 | 0 |
| 1 | 2 |  | 3 | 1 |
| 2 | 2 |  |  | 1 |
| 3 | 0 |  |  | 1 |
| 4 | 4 |  |  | 3 |
| 5 | 0 |  |  | 3 |
| 6 | 0 |  |  | 3 |
| 7 | 4 |  |  | 4 |
| 8 | 1 |  |  | 4 |

Решение системы линейных алгебраических уравнений с помощью реализованного алгоритма Берлекэмпа-Месси на С++ приведено на рисунке 2.

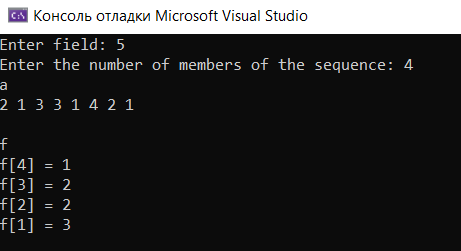


Рисунок 2 ­– Решение системы линейных уравнений

Пример 2. Решим систему линейных уравнений над полем GF (2):

Алгоритм решения системы приведен в таблице 2 на основе алгоритма 1:

Таблица 2 — Алгоритм решения системы уравнений над полем GF (2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 |  | 1 | 1 | 0 |
| 1 | 1 |  | 1 | 1 |
| 2 | 0 |  |  | 1 |
| 3 | 0 |  |  | 1 |
| 4 | 0 |  |  | 1 |
| 5 | 1 |  |  | 4 |
| 6 | 1 |  |  | 4 |
| 7 | 0 |  |  | 4 |
| 8 | 1 |  |  | 4 |
| 9 | 1 |  |  | 5 |
| 10 | 1 |  |  | 5 |

Решение системы линейных алгебраических уравнений с помощью реализованного алгоритма Берлекэмпа-Месси на С++ приведено на рисунке 3.

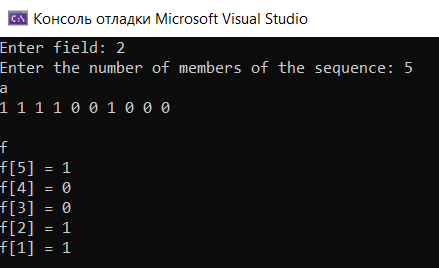


Рисунок 3 ­– Решение системы линейных уравнений

Пример 3. Решим систему линейных алгебраических уравнений над полем GF (7):

Алгоритм решения системы приведен в таблице 3 на основе алгоритма 1:

Таблица 3 — Алгоритм решения системы уравнений над полем GF (7)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 |  | 1 | 1 | 0 |
| 1 | 5 |  | 3 | 1 |
| 2 | 5 |  |  | 1 |
| 3 | 5 |  |  | 2 |
| 4 | 3 |  |  | 2 |
| 5 | 6 |  |  | 3 |
| 6 | 2 |  |  | 3 |

Решение системы линейных алгебраических уравнений с помощью реализованного алгоритма Берлекэмпа-Месси на С++ приведено на рисунке 4.

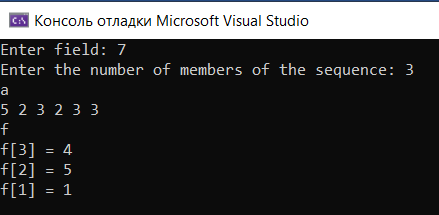


Рисунок 4 ­– Решение системы линейных уравнений

### **1.3.2. Примеры для алгоритма 2.**

Пример 4.Решим систему линейных алгебраических уравнений над полем GF (5):

Алгоритм решения системы приведен в таблице 4 на основе алгоритма 2:

Таблица 4 — Алгоритм решения системы уравнений над полем GF (5)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 0 |  | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 |  | 1 | 1 |
| 2 | 3 | 1 |  |  | 1 |
| 3 | 1 | 1 |  |  | 2 |
| 4 | 3 | 1 |  |  | 2 |
| 5 | 4 | 4 |  |  | 3 |
| 6 | 3 | 4 |  |  | 3 |
| 7 | 4 | 4 |  |  | 4 |
| 8 | 3 | 4 |  |  | 4 |
| 9 | 0 | 4 |  |  | 4 |
| 10 | 0 | 4 |  |  | 4 |

Решение системы линейных алгебраических уравнений с помощью реализованного алгоритма Берлекэмпа-Месси на С++ приведено на рисунке 5.

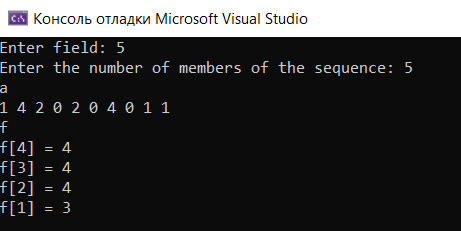
****

Рисунок 5 ­– Решение системы линейных уравнений

Пример 5.Решим систему линейных алгебраических уравнений над полем GF (2):

Алгоритм решения системы приведен в таблице 5 на основе алгоритма 2:

Таблица 5 — Алгоритм решения системы уравнений над полем GF (2)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 0 |  | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |  | 0 |
| 2 | 0 | 1 | 1 |  | 0 |
| 3 | 1 | 1 |  | 1 | 3 |
| 4 | 1 | 1 |  |  | 3 |
| 5 | 1 | 1 |  |  | 3 |
| 6 | 1 | 1 |  |  | 3 |
| 7 | 1 | 1 |  |  | 4 |
| 8 | 0 | 1 |  |  | 4 |

Решение системы линейных алгебраических уравнений с помощью реализованного алгоритма Берлекэмпа-Месси на С++ приведено на рисунке 6.

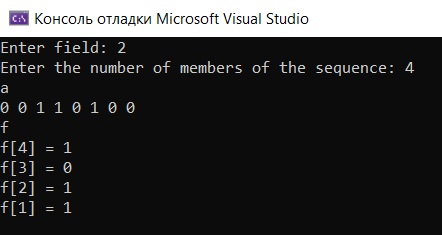
****

Рисунок 6 ­– Решение системы линейных уравнений

Пример 6.Решим систему линейных алгебраических уравнений над полем GF (7):

Алгоритм решения системы приведен в таблице 6 на основе алгоритма 2:

Таблица 6 — Алгоритм решения системы уравнений над полем GF (7)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 0 |  | 1 | 1 | 1 | 0 |
| 1 | 6 | 6 |  | 1 | 1 |
| 2 | 0 | 6 |  |  | 1 |
| 3 | 1 | 1 |  |  | 2 |
| 4 | 0 | 1 |  |  | 2 |
| 5 | 1 | 1 |  |  | 3 |
| 6 | 2 | 1 |  |  | 3 |

Решение системы линейных алгебраических уравнений с помощью реализованного алгоритма Берлекэмпа-Месси на С++ приведено на рисунке 7.

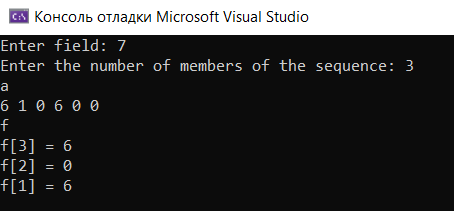
****

Рисунок 7 ­– Решение системы линейных уравнений

# **Криптоанализ потокового шифра на основе алгоритма Берлекэмпа-Месси**

## **2.1. Потоковые шифры**

Потоковый шифр – это симметричный шифр, преобразующий каждый символ открытого текста в символ шифрованного, зависящий от ключа и расположения символа в тексте. Потоковые шифры преобразуют открытый текст в шифротекст по одному биту за операцию.

Важнейшее достоинство потоковых шифров – высокая скорость шифрования, примерно равная скорости поступления входной информации. Это позволяет обеспечить шифрование практически в реальном масштабе времени вне зависимости от объема информации и разрядности потока данных [7].

Простейшая реализация потокового шифра показана на рисунке 8. Генератор потока ключей выдает поток битов: . Этот поток ключей и поток битов открытого текста, , подвергаются операции "исключающее или" (XOR), и в результате получается поток битов шифротекста.

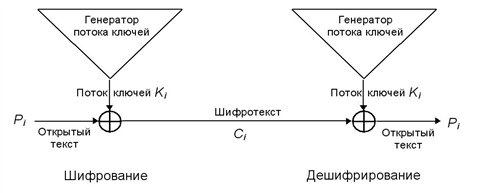


Рисунок 8 ­– Реализация потокового шифра

## **2.2. Криптоанализ потокового шифра**

Известно множество атак на потоковые шифры, основными из которых является атака на основе использования алгоритма БМ. Сущность атаки заключается в том, что, не зная структуры шифра, но зная, что он основан на использовании LFSR и что линейная эквивалентная сложность шифрующей гаммы является обозримой величиной, находится отрезок шифрующей гаммы достаточно большой длины (для чего должно быть известно N бит сообщениями соответствующих им бит криптограммы Е). Далее к этой гамме применяется алгоритм БМ, что позволяет найти длину эквивалентного LFSR, а также его отводы и начальное заполнение и, вычислить произвольное продолжение этой гаммы. Знание продолжения гаммы позволяет, в свою очередь, дешифровать сообщение за пределами ее известных пока бит. Сложность выполнения БМ составляет примерно операций [8].

Для предотвращения этой атаки необходимо использовать потоковые шифры с нелинейными узлами усложнения, которые обеспечивают такую линейную эквивалентную сложность гаммы, что нахождение элементов этой гаммы или выполнение операций оказывается невозможным. Примером такого шифра является генератор гаммы с управляемым тактированием, показанный на рисунке 9, при выборе его параметров.

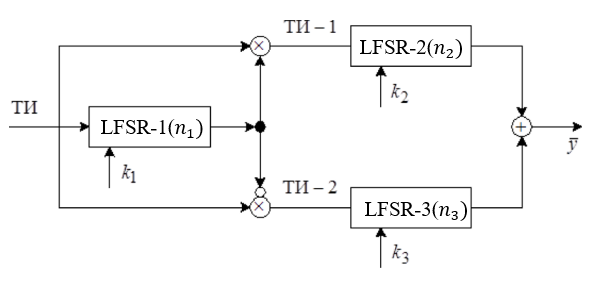
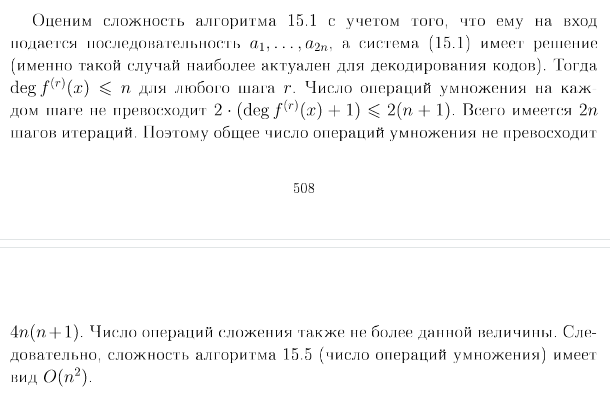


Рисунок 9 ­– Генератор гаммы с управляемым тактированием

# **Реализация алгоритма Берлекэмпа-Месси**



# **Заключение**

# **Список использованной литературы**

1. Потоковый шифр [Электронный ресурс]: база данных. Режим доступа: <https://utmagazine.ru/posts/14950-potokovyy-shifr>
2. Berlekamp E. R. Algebraic Coding Theory. – New York: McGrow Hill, 1968 (перевод: Берлекэмп Э. Алгебраическая теория кодирования. – М.: Мир, 1971).
3. Massey J.L., Shift Register Synthesis and BCH Decoding, // IEEE Trans. Inform. Theory. — vol. IT-15, no. 1, 1969.
4. Регистр линейной обратной связью пример. Регистры сдвига с обратной линейной связью [Электронный ресурс]: база данных. Режим доступа: <https://pervayakyzyl.ru/registr-lineinoi-obratnoi-svyazyu-primer-registry-sdviga-s/>
5. Рацеев С.М., Лавриненко А.Д., Степанова Е.А. ­­Об алгоритме Берлекэмпа-Месси и его применении в алгоритмах декодирования // Вестник Самарского университета. Естественнонаучная серия. 2021 Т.27 №1 С. 44-60.
6. Massey J.L. Shift-register synthesis and BCH decoding // IEEE Trans. Inf. Theory. – 1969. Vol. IT. 15, № 1. – P. 122–12.
7. Сушко, С. А. Практическая криптология. Лекция 10 [Электронный ресурс]: Учеб. пособие Национальный технический университет «Днепровская политехника».
8. Основные способы криптоанализа потоковых шифров [Электронный ресурс]: база данных. Режим доступа:

<https://bstudy.net/839284/tehnika/postroenie_datchika_shifruyuschey_gammy>

# ПРИЛОЖЕНИЕ А

Реализация алгоритма 1.

#include <iostream>

#include <string>

#include <vector>

#include <numeric>

#include <algorithm>

#include <time.h>

using namespace std;

int Sravnenie(int a, int m) {

int y0 = 0, y1 = 1, q, r, buf, x0;

int zn;

if (a > 0) zn = 1;

else zn = -1;

a = abs(a);

q = m / a;

r = m - q \* a;

while (r != 0) {

m = a;

a = r;

buf = y0 - q \* y1;

y0 = y1;

y1 = buf;

q = m / a;

r = m - q \* a;

}

x0 = zn \* y1;

return x0;

}

int BinAlgEuclid(int a, int b) {

if (a == 0 && b == 0)

return -1;

a = abs(a);

b = abs(b);

int d = 1;

if (a == 0 || b == 0)

return a + b;

while (!(a & 1) && !(b & 1)) {

a = a >> 1;

b = b >> 1;

d = d \* 2;

}

while (!(a & 1)) {

a = a >> 1;

}

while (!(b & 1)) {

b = b >> 1;

}

while (a != 0 && b != 0) {

while (!(a & 1)) {

a = a >> 1;

}

while (!(b & 1)) {

b = b >> 1;

}

if (a >= b) a = (a - b) >> 1;

else b = (b - a) >> 1;

}

return((a + b) \* d);

}

int obratnoe(int a, int m) {

int d, x0, k = 0;

d = BinAlgEuclid(a, m); //находим нод

if (1 % d != 0) {

return -1; //решений нет

}

x0 = Sravnenie(a, m);

if (x0 > 0) x0 = x0 % m;

else {

int f = (x0 / m) \* (-1) + 1;

x0 = f \* m + x0;

}

return x0;

}

int main() {

setlocale(LC\_ALL, "Russian");

int m, n;

cout << "Enter field: ";

cin >> m;

cout << "Enter the number of members of the sequence: ";

cin >> n;

int l = n-1;

int r = 0, L = 0, d=0, k=0, obr;

vector<int> a(2 \* n, 0);

srand(time(NULL));

m--;

for (int i = 0; i < n; i++) {

a[i] = 0 + rand() % (m - 0 + 1);

}

vector<int> f\_answer(n, 0);

for (int i = 0; i < n; i++) {

f\_answer[i] = 0 + rand() % (m - 0 + 1);

}

n = 2 \* n;

int g = 0;

m++;

for (int i = n / 2; i < n; i++) {

for (int j = 0; j < n / 2; j++) {

a[i] += a[l-j+g] \* f\_answer[j];

}

a[i] = ((a[i] % m) - m ) % m;

a[i] = -a[i];

g++;

}

cout << "a" << endl;

for (int i = 0; i<n; i++) {

cout << a[i] << " " ;

}

cout << endl;

cout << "f\_ans" << endl;

for (int i = 0; i <= l; i++) {

cout << f\_answer[i] << " ";

}

cout << endl;

vector<int> f(n, 0);

vector<int> b(n, 0);

vector<int> buf(n, 0);

f[0] = b[0] = buf[0] = 1;

for (r = 0; r < n; r++) {

int k = 0;

for (int i = 1; i < L+1; i++)

k += f[i] \* a[r - i];

d = (f[0] \* a[r] + k) % m;

if (d == 0) {

for (int i = n - 1; i > 0 ; i--)

b[i] = b[i - 1];

b[0] = 0;

}

else {

buf[0] = f[0];

for (int i = 1; i < n; i++) {

buf[i] = (f[i] - d \* b[i - 1]) % m;

if (buf[i] < 0) buf[i] += m;

}

if (2 \* L < r + 1) {

obr = obratnoe(d, m);

for (int i = 0; i < n; i++) {

b[i] = (obr \* f[i]) % m;

f[i] = buf[i];

if (b[i] < 0) b[i] += m;

}

L = r - L + 1;

}

else {

for (int i = n - 1; i > 0; i--) {

b[i] = b[i - 1];

f[i] = buf[i];

}

f[0] = buf[0];

b[0] = 0;

}

}

}

cout << "f" << endl;

int i=n-1;

while (f[i] == 0 && i >= 0) {

i--;

}

for (; i > 0; i--) {

cout << "f[" << i << "] = " << f[i] << endl;

}

}

# ПРИЛОЖЕНИЕ Б

Реализация алгоритма 2.

#include <iostream>

#include <string>

#include <vector>

#include <numeric>

#include <algorithm>

#include <time.h>

using namespace std;

int Sravnenie(int a, int m) {

int y0 = 0, y1 = 1, q, r, buf, x0;

int zn;

if (a > 0) zn = 1;

else zn = -1;

a = abs(a);

q = m / a;

r = m - q \* a;

while (r != 0) {

m = a;

a = r;

buf = y0 - q \* y1;

y0 = y1;

y1 = buf;

q = m / a;

r = m - q \* a;

}

x0 = zn \* y1;

return x0;

}

int BinAlgEuclid(int a, int b) {

if (a == 0 && b == 0)

return -1;

a = abs(a);

b = abs(b);

int d = 1;

if (a == 0 || b == 0)

return a + b;

while (!(a & 1) && !(b & 1)) {

a = a >> 1;

b = b >> 1;

d = d \* 2;

}

while (!(a & 1)) {

a = a >> 1;

}

while (!(b & 1)) {

b = b >> 1;

}

while (a != 0 && b != 0) {

while (!(a & 1)) {

a = a >> 1;

}

while (!(b & 1)) {

b = b >> 1;

}

if (a >= b) a = (a - b) >> 1;

else b = (b - a) >> 1;

}

return((a + b) \* d);

}

int obratnoe(int a, int m) {

int d, x0, k = 0;

d = BinAlgEuclid(a, m);

if (1 % d != 0) {

return -1;

}

x0 = Sravnenie(a, m);

if (x0 > 0) x0 = x0 % m;

else {

int f = (x0 / m) \* (-1) + 1;

x0 = f \* m + x0;

}

return x0;

}

int main() {

setlocale(LC\_ALL, "Russian");

int m, n;

cout << "Enter field: ";

cin >> m;

cout << "Enter the number of members of the sequence: ";

cin >> n;

int l = n - 1;

int r = 0, L = 0, d = 0, k = 0, obr;

vector<int> a(2 \* n, 0);

srand(time(NULL));

m--;

for (int i = 0; i < n; i++) {

a[i] = 0 + rand() % (m - 0 + 1);

}

vector<int> f\_answer(n, 0);

for (int i = 0; i < n; i++) {

f\_answer[i] = 0 + rand() % (m - 0 + 1);

}

int g = 0;

m++;

n = 2 \* n;

for (int i = n / 2; i < n; i++) {

for (int j = 0; j < n / 2; j++) {

a[i] += a[l - j + g] \* f\_answer[j];

}

a[i] = ((a[i] % m) - m) % m;

a[i] = -a[i];

g++;

}

cout << "a" << endl;

for (int i = 0; i < n; i++) {

cout << a[i] << " ";

}

cout << endl;

cout << "f\_ans" << endl;

for (int i = 0; i <= l; i++) {

cout << f\_answer[i] << " ";

}

cout << endl;

vector<int> f(n, 0);

vector<int> b(n, 0);

vector<int> buf(n, 0);

f[0] = b[0] = buf[0] = 1;

int delta = 1;

for (r = 0; r < n; r++) {

int k = 0;

for (int i = 1; i < L + 1; i++)

k += f[i] \* a[r - i];

d = (f[0]\*a[r] + k) % m;

if (d == 0) {

for (int i = n - 1; i > 0; i--)

b[i] = b[i - 1];

b[0] = 0;

}

else {

buf[0] = (delta \* f[0]) % m;

for (int i = 1; i < n; i++) {

buf[i] = (delta\*f[i] - d \* b[i - 1]) % m;

if (buf[i] < 0) buf[i] += m;

}

if (2 \* L < r + 1) {

for (int i = 0; i < n; i++) {

b[i] = f[i];

f[i] = buf[i];

}

L = r - L + 1;

delta = d;

}

else {

for (int i = n - 1; i > 0; i--) {

b[i] = b[i - 1];

f[i] = buf[i];

}

f[0] = buf[0];

b[0] = 0;

}

}

}

for (int i = n - 1; i > 0; i--) {

obr = obratnoe(f[0], m);

f[i] = (obr \* f[i]) % m;

}

cout << "f" << endl;

int i = n - 1;

while (f[i] == 0 && i >= 0) {

i--;

}

for (; i > 0; i--) {

cout << "f[" << i << "] = " << f[i] << endl;

}

}