Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Запаздывающие генераторы Фибоначчи

Студент гр. 3331506/80401 А. П. Винкельман

Преподаватель Е. М. Кузнецова

<<___>>_____ 2021 г.

Санкт-Петербург

2021

Оглавление

Введение

Запаздывающие генераторы Фибоначчи (*lagged Fibonacci generator*) — генераторы псевдослучайных чисел. Также эти генераторы называются аддитивными генераторами.

Генераторы Фибоначи в отличие от других генераторов, которые используют линейный конгруэнтный алгоритм, можно использовать в статистических алгоритмах, требующих высокого разрешения. По этой причине линейный конгруэнтный алгоритм постепенно потерял свою популярность и его место заняло семейство фибоначчиевых алгоритмов, которые могут быть рекомендованы для использования в алгоритмах, критичных к качеству случайных чисел.

Наибольшую популярность фибоначчиевы датчики получили в связи с тем, что скорость выполнения арифметических операций с вещественными числами сравнялась со скоростью целочисленной арифметики, а фибоначчиевы датчики естественно реализуются в вещественной арифметике.

История

Последовательность, в которой число X_{n+1} зависит более, чем от одного из предшествующих значений, и которая определяется следующей формулой:

$$X_{n+1} = (X_n + X_{n-1}) mod(2^m), \quad (1)$$

носит название последовательность Фибоначчи.

В начале 50-х годов изучался данный алгоритм, однако, исследования показали, что этот генератор, как источник случайных чисел, был неэффективен. Грин (Green), Смит (Smith) и Клем (Klem) предложили доработанную формулу последовательности Фибоначчи в виде:

$$X_{n+1} = (X_n + X_{n-k}) mod(2^m). \quad (2)$$

Однако, положительный результат получался лишь при $k \geq 16$.

В 1958 году Митчел Дж. Ж. (Mitchell G. J.) и Мур Д. Ф. (Moore D. P.) вывели последовательность:

$$X_n = (X_{n-24} + X_{n-55}) mod(2^m), \quad (3)$$

где $n \geq 55$, m – чётное число, $X_0, X_1, \dots, X_{55}$ – произвольные целые не все чётные числа. Числа 24 и 55 выбраны так, чтобы определялась последовательность, младшие значащие двоичные разряды ($X_n mod(2)$) которой имеют длину периода $2^{55} - 1$.

Очевидными преимуществами данного алгоритма являются его быстрота, поскольку он не требует умножения чисел, а также, длина периода.

Числа 24 и 55 обычно называют *запаздыванием*, а числа X_n, определённые по уравнению (3) – последовательностью Фибоначчи с запаздыванием.

Описание алгоритма

Требуется получить псевдослучайные значения. При чём, если нужна последовательность случайных чисел, то она должна обладать хорошими статистическими свойствами.

Известны разные схемы использования метода Фибоначчи с запаздыванием. Один из широко распространённых фибоначчиевых датчиков основан на следующей рекуррентной формуле:

$$X_k = \begin{cases} X_{k-a} - X_{k-b}, \text{если } X_{k-a} \geq X_{k-b} \\ X_{k-a} - X_{k-b} + 1, \text{если } X_{k-a} < X_{k-b} \end{cases}, \quad (4)$$

где X_k – вещественные числа из диапазона [0, 1), a, b – целые положительные числа, называемые лагами.

При реализации через целые числа достаточно формулы $X_k = X_{k-a} - X_{k-b}$, при этом будут происходить арифметические переполнения. Для работы фибоначчиеву датчику требуется знать $max\{a,\ b\}$ предыдущих сгенерированных случайных чисел. При программной реализации для хранения сгенерированных случайных чисел используется конечная циклическая очередь на базе массива. Предыдущие случайные числа могут быть сгенерированы простым конгруэнтным методом.

Для полученной формулы существует алгоритм работы, представленный ниже:

1. Запрашиваем у пользователя параметры a, b и количество желаемых случайных величин (*Amount*).
2. Создаём массив (условно назовём его *Arr*[]), размер которого имеет величину $max(a, b) + Amount + 1$.
3. Заполняем участок массива от 0 до $max(a,\ b)$ случайными числами. Не имеет значения, как эти величины были получены. Здесь можно использовать встроенную функцию *rand*().
4. Выполняем цикл от $i = 0$ до $i = \max(a, b) + Amount$, i увеличивается на единицу.
5. При каждой итерации цикла выполняем проверку:

Если $Arr[i-a] \geq Arr[i-b]$, тогда

- следующий элемент массива равен $Arr[i-a] - Arr[i-b]$,
- иначе $Arr[i-b] - Arr[i-a]$.

Для лучшего понимания рассмотрим следующий пример:

Требуется получить 5 случайных положительных чисел. Известны лаги $a = 4, b = 7$.

Пусть первые случайные числа имеют следующие значения: $X_0 = 5, X_1 = 15, X_2 = 20, X_3 = 13, X_4 = 8, X_5 = 2, X_6 = 3, X_7 = 17$. Получим тогда:

$X_8 = X_{8-7} - X_{8-4} = 15 - 8 = 7$ $(X_{i-b} > X_{i-a})$;

$X_9 = X_{9-7} - X_{9-4} = 20 - 2 = 18$ $(X_{i-b} > X_{i-a})$;

$X_{10} = X_{10-7} - X_{10-4} = 13 - 3 = 10$ $(X_{i-b} > X_{i-a})$;

$X_{11} = X_{11-4} - X_{11-7} = 17 - 8 = 9$ $(X_{i-a} \geq X_{i-b})$;

$X_{12} = X_{12-4} - X_{12-7} = 7 - 2 = 5$ $(X_{i-a} \geq X_{i-b})$;

В результате были получены числа 7, 18, 10, 9, 5. Видно, генерируемая последовательность чисел внешне похожа на случайную. Также видно, что диапазон выдаваемых чисел лежит от 0 до максимального из имеющихся в начале случайных чисел.

Сам код представлен в приложении 1.

Исследование алгоритма

Для изучения зависимости времени работы алгоритма от количества вычисляемых «случайных» величин проводился подсчёт времени, необходимого алгоритму для вычисления требуемого количества новых значений. Во всех измерениях лаги имели значения $a = 1$ и $b = 3$. Менялось количество вычисляемых значений *Amount*: 5, 10, 15, 20, 25, 30, 35. Для каждого значения *Amount* проводилось по 10 измерений.

Это было сделано для того, чтобы вычислить среднее время работы алгоритма при различных количествах находимых «случайных» величин, поскольку время работы зависит не только от *Amount*, но и от порядка первоначально сгенерированных чисел и последовательно вычисленных значений. Подсчёт времени проводился при помощи встроенной функции *clock*(), которая высчитывает время в миллисекундах.

В качестве примера ниже представлена таблица с замерами для *Amount* = 15. Лаги $a = 1$, $b = 3$, как говорилось выше.

Таблица 1 – Значения замеренных величин

№	Первоначально сгенерированные числа	Последующие вычисленные значения	Время, мс
1	20837, 9851, 17544, 6372	3479, 14065, 7693, 4214, 9851, 2158, 2056, 7795, 5637, 3581, 4214, 1423, 2158, 2056, 633	11
2	21226, 10959, 13460, 19091	8132, 5328, 13763, 5631, 303,	7

		13460, 7829, 7526, 5934, 1895, 5631, 303, 1592, 4039, 3736	
3	21595, 13113, 496, 18501	5388, 4892, 13609, 8221, 3329, 10280, 2059, 1270, 9010, 6951, 5681, 3329, 3622, 2059, 1270	9
4	21931, 6087, 5497, 6654	567, 4930, 1724, 1157, 3773, 2049, 892, 2881, 832, 60, 2821, 1989, 1929, 892, 1097	8
5	22199, 2720, 28568, 13766	11046, 17522, 3756, 7290, 10232, 6476, 814, 9418, 2942, 2128, 7290, 4348, 2220, 5070, 722	7
6	22548, 5920, 6722, 32637	26717, 19995, 12642, 14075, 5920, 6722, 7353, 1433, 5289, 2064, 631, 4658, 2594, 1963, 2695	7
7	22849, 11733, 11826, 18238	6505, 5321, 12917, 6412,	6

		1091, 11826, 5414, 4323, 7503, 2089, 2234, 5269, 3180, 946, 4323	
8	23146, 6798, 32833, 12543	5745, 26088, 13545, 7800, 18288, 4743, 3057, 15231, 10488, 7431, 7800, 2688, 4743, 3057, 369	5
9	23466, 11566, 13050, 11452	114, 12936, 1484, 1370, 11566, 10082, 8712, 2854, 7228, 1484, 1370, 5858, 4374, 3004, 2854	7
10	23799, 26560, 188, 8310	18250, 18062, 9752, 8498, 9564, 188, 8310, 1254, 1066, 7244, 5990, 4924, 2320, 3670, 1254	6

Высчитаем среднее время работы алгоритма при a = 1, b = 3, *Amount* = 15:

$$t_{\text{ср}} = \frac{t_1 + t_2 + t_3 + t_4 + t_5 + t_6 + t_7 + t_8 + t_9 + t_{10}}{10}$$

$$= \frac{11 + 7 + 9 + 8 + 7 + 7 + 6 + 5 + 7 + 6}{10} = 7{,}3 \text{ мс}$$

График зависимости времени работы алгоритма от величины *Amount* представлена ниже на рисунке 1.

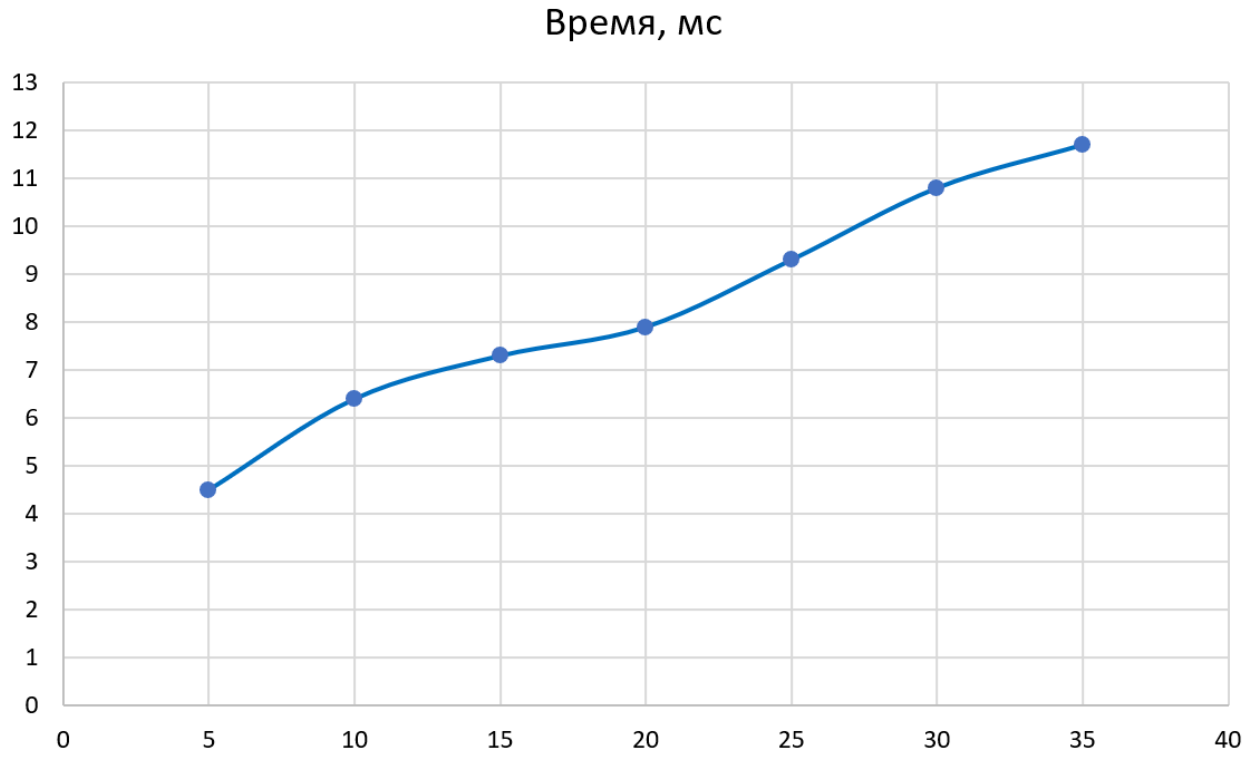

Рисунок 1 – Зависимость времени от *Amount*

Из графика видно, что зависимость можно считать линейной. Для более точного определения времени работы следует провести намного больше 10 измерений для каждого значения *Amount*.

При одинаковых значениях лагов время работы алгоритма при изменении *Amount* с 5 до 35 увеличилось на 7,2 мс. Исходя из этого, можно предположить, что время порядка 1 с будет достигнуто, если программа будет высчитывать более 5000 новых «случайных» значений.

В целом же сложность алгоритма можно оценить как O(n). Где $n = \max(a, b) + Amount + 1$.

Список литературы

Жельников В. Криптография от папируса до компьютера. — 1996. — 325 с.

https://ru.wikipedia.org/wiki/Метод_Фибоначчи_с_запаздываниями

https://intuit.ru/studies/courses/691/547/lecture/12383?page=2

Приложение 1

```cpp
#include <iostream>
#include <ctime>
#include <vector>
#include <iterator>
#include <stdlib.h>
#include <ctime>

using namespace std;

typedef vector <int> Mass;

int main() {
    unsigned int a = 0, b = 0;
    int Amount = 0;

    // Создание итераторов
    Mass::iterator LagA;
    Mass::iterator LagB;
    Mass::iterator Lag;

    // Запрос у пользователя на ввод параметров
    cout << "Lag a = ";
    cin >> a;
    cout << "Lag b = ";
    cin >> b;
    cout << "Amount of numbers ";
    cin >> Amount;

    // Начало отсчёта времени работы
    unsigned int start_time = clock();

    // Создание массива
    Mass massive(1);
    massive.resize( new_size: max(a,b) + Amount + 1);
```

```
35
36      // Инициализация лагов и сдвиг
37      // на значения a и b
38      LagA = massive.begin();
39      advance( &: LagA, a);
40      LagB = massive.begin();
41      advance( &: LagB, b);
42
43      // Инициализация дополнительного итератора
44      // для коорректной работы цикла
45      if (LagA > LagB) {
46          Lag = LagA;
47      } else {
48          Lag = LagB;
49      }
50
51      // Основной итератор для движения по массиву
52      Mass::iterator index;
53      index = massive.begin();
54
55      // Заполнение первой части массива
56      // случайными значениями
57      srand( _Seed: time( _Time: nullptr));
58      while (index <= Lag) {
59          *index = rand();
60          cout << "Number = " << *index << endl;
61          index++;
62      }
```

```
63
64      // Получение последовательности методом
65      // Фибоначчи с запаздыванием
66      for (; index != massive.end(); index++, LagA++, LagB++) {
67          if (*LagA >= *LagB) {
68              *index = *LagA - *LagB;
69          } else {
70              *index = *LagB - *LagA;
71          }
72          cout << *index << endl;
73      }
74
75      unsigned int end_time = clock();   // Конец отсчёта времени работы
76      // Вычисление конечного времени работы
77      unsigned int search_time = end_time - start_time;
78      cout << "Run time = " << search_time << endl;
79      return 0;
80  }
81
```