

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

Курсовая работа

Дисциплина: Объектно-ориентированное программирование

Тема: Запоздывающие генераторы Фибоначчи

Студент гр. 3331506/80401

А. П. Винкельман

Преподаватель

Е. М. Кузнецова

<<__>>_____ 2021 г.

Санкт-Петербург

2021

Оглавление

| | |
|-------------------------------------|----------|
| Введение | 3 |
| История | 4 |
| Описание алгоритма..... | 5 |
| Исследование алгоритма | 6 |
| Список литературы | 6 |
| Приложение 1..... | 7 |

Введение

Запаздывающие генераторы Фибоначчи (*lagged Fibonacci generator*)

— генераторы псевдослучайных чисел, также называемые аддитивными генераторами.

В отличие от генераторов, использующих линейный конгруэнтный алгоритм, фибоначчиевы генераторы можно использовать в статистических алгоритмах, требующих высокого разрешения. В связи с этим 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:

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

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:

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

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:

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

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 $(X_n \bmod(2))$ 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:

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

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

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

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

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

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

Список литературы

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

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

Приложение 1

```
1  #include <iostream>
2  #include <ctime>
3  #include <vector>
4  #include <iterator>
5  #include <stdlib.h>
6
7  using namespace std;
8
9  typedef vector <int> Mass;
10
11  int main() {
12      unsigned int a = 0, b = 0;
13      int Amount = 0;
14
15      // Создание итераторов
16      Mass::iterator LagA;
17      Mass::iterator LagB;
18      Mass::iterator Lag;
19
20      // Запрос у пользователя на ввод параметров
21      cout << "Lag a = ";
22      cin >> a;
23      cout << "Lag b = ";
24      cin >> b;
25      cout << "Amount of numbers ";
26      cin >> Amount;
27
28      // Создание массива
29      Mass massive(1);
30      massive.resize( new_size: max(a,b) + Amount + 1);
31
32      // Инициализация лагов и сдвиг
33      // на значения a и b
34      LagA = massive.begin();
35      advance( &: LagA, a);
36      LagB = massive.begin();
37      advance( &: LagB, b);
38
```

```

39 // Инициализация дополнительного итератора
40 // для корректной работы цикла
41 if (LagA > LagB) {
42     Lag = LagA;
43 } else {
44     Lag = LagB;
45 }
46
47 // Основной итератор для движения по массиву
48 Mass::iterator index;
49 index = massive.begin();
50
51 // Заполнение первой части массива
52 // случайными значениями
53 srand(_Seed: time(_Time: nullptr));
54 while (index <= Lag) {
55     *index = rand();
56     cout << "Number = " << *index << endl;
57     index++;
58 }
59
60 // Получение последовательности методом
61 // Фибоначчи с запаздыванием
62 for (; index != massive.end(); index++, LagA++, LagB++) {
63     if (*LagA >= *LagB) {
64         *index = *LagA - *LagB;
65     } else {
66         *index = *LagB - *LagA;
67     }
68     cout << *index << endl;
69 }
70
71 return 0;
72 }
73

```