|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

«Сравнение табличного и алгоритмического способа получения последовательности псевдослучайных чисел»

Студент ИУ7-68Б(В) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Н. Бахолдин

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель темы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.В. Рудаков

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

Табличные ГСЧ в качестве источника случайных чисел используют специальным образом составленные таблицы, содержащие проверенные некоррелированные, то есть никак не зависящие друг от друга, цифры. В табл. 1 приведен небольшой фрагмент такой таблицы. Обходя таблицу слева направо сверху вниз, можно получать равномерно распределенные от 0 до 1 случайные числа с нужным числом знаков после запятой. Так как цифры в таблице не зависят друг от друга, то таблицу можно обходить разными способами, например, сверху вниз, или справа налево, или, скажем, можно выбирать цифры, находящиеся на четных позициях.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 1. Случайные цифры. Равномерно распределенные от 0 до 1 случайные числа   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Случайные цифры** | | | | | | | | **Равномерно распределенные от 0 до 1 случайные числа** | | 9 | 2 | 9 | 2 | 0 | 4 | 2 | 6 | 0.929 | | 9 | 5 | 7 | 3 | 4 | 9 | 0 | 3 | 0.204 | | 5 | 9 | 1 | 6 | 6 | 5 | 7 | 6 | 0.269 | | … | | | | | | | | … | |
|  |

Достоинство данного метода в том, что он дает действительно случайные числа, так как таблица содержит проверенные некоррелированные цифры. Недостатки метода: для хранения большого количества цифр требуется много памяти; большие трудности порождения и проверки такого рода таблиц, повторы при использовании таблицы уже не гарантируют случайности числовой последовательности, а значит, и надежности результата.

Числа, генерируемые с помощью алгоритмически составленых ГСЧ, всегда являются псевдослучайными (или квазислучайными), то есть каждое последующее сгенерированное число зависит от предыдущего:

*ri* + 1 = *f*(*ri*).

Последовательности, составленные из таких чисел, образуют петли, то есть обязательно существует цикл, повторяющийся бесконечное число раз. Повторяющиеся циклы называются периодами.

Достоинством данных ГСЧ является быстродействие; генераторы практически не требуют ресурсов памяти, компактны. Недостатки: числа нельзя в полной мере назвать случайными, поскольку между ними имеется зависимость, а также наличие периодов в последовательности квазислучайных чисел. Рассмотрим линейный конгруэнтный метод получения ГСЧ.

Линейный конгруэнтный метод является одной из простейших и наиболее употребительных в настоящее время процедур, имитирующих случайные числа. В этом методе используется операция mod(*x*, *y*), возвращающая остаток от деления первого аргумента на второй. Каждое последующее случайное число рассчитывается на основе предыдущего случайного числа по следующей формуле:

*ri* + 1 = mod(*k* · *ri* + *b*, *M*).

|  |  |
| --- | --- |
| |  | | --- | | *M* — модуль (0 < *M*);  *k* — множитель (0 ≤ *k* < *M*);  *b* — приращение (0 ≤ *b* < *M*);  *r*0 — начальное значение (0 ≤ *r*0 < *M*). | |

Последовательность случайных чисел, полученных с помощью данной формулы, называется линейной конгруэнтной последовательностью. Многие авторы называют линейную конгруэнтную последовательность при *b* = 0 мультипликативным конгруэнтным методом, а при *b* ≠ 0 — смешанным конгруэнтным методом. Для качественного генератора требуется подобрать подходящие коэффициенты. Необходимо, чтобы число *M* было довольно большим, так как период не может иметь больше *M* элементов. С другой стороны, деление, использующееся в этом методе, является довольно медленной операцией, поэтому для двоичной вычислительной машины логичным будет выбор *M* = 2*N*, поскольку в этом случае нахождение остатка от деления сводится внутри ЭВМ к двоичной логической операции «AND». Также широко распространен выбор наибольшего простого числа *M*, меньшего, чем 2*N*: в специальной литературе доказывается, что в этом случае младшие разряды получаемого случайного числа *ri* + 1 ведут себя так же случайно, как и старшие, что положительно сказывается на всей последовательности случайных чисел в целом. В качестве примера можно привести одно из чисел Мерсенна, равное 231 – 1, и таким образом, *M* = 231 – 1.

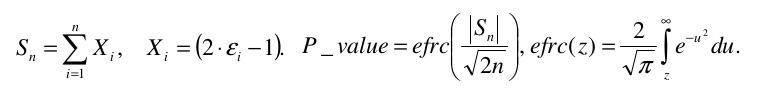
Одним из требований к линейным конгруэнтным последовательностям является как можно большая длина периода. Длина периода зависит от значений *M*, *k* и *b*. Теорема, которую мы приведем ниже, позволяет определить, возможно ли достижение периода максимальной длины для конкретных значений *M*, *k* и *b*.

**Теорема**. Линейная конгруэнтная последовательность, определенная числами *M*, *k*, *b* и *r*0, имеет период длиной *M* тогда и только тогда, когда:

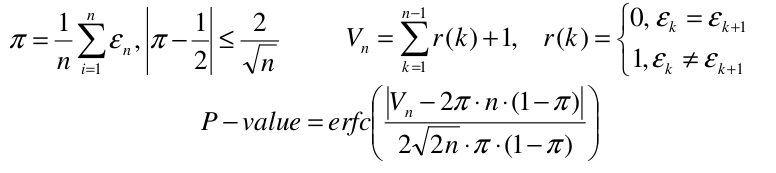
* числа *b* и *M* взаимно простые;
* *k* – 1 кратно *p* для каждого простого *p*, являющегося делителем *M*;
* *k* – 1 кратно 4, если *M* кратно 4.

Для выполнения задачи по оценке случайности последовательности были программно реализованы следующие тесты:

1. Частотный побитовый тест. Суть данного теста заключается в определении соотношения между нулями и единицами во всей двоичной последовательности. Цель — выяснить, действительно ли число нулей и единиц в последовательности приблизительно одинаковы, как это можно было бы предположить в случае истинно случайной бинарной последовательности. Тест оценивает, насколько близка доля единиц к 0,5. Таким образом, число нулей и единиц должно быть примерно одинаковым. Если вычисленное в ходе теста значение вероятности *p* < 0,01, то данная двоичная последовательность не является истинно случайной. В противном случае последовательность носит случайный характер. Стоит отметить, что все последующие тесты проводятся при условии, что пройден данный тест.
2. Частотный побитовый тест. Вероятность ошибки первого рода называют уровнем статистической значимости и обозначают как α. Т.е. α — это вероятность отбраковать «хорошую» случайную последовательность. Это значение определяется областью применения. В криптографии принято α брать от 0.001 до 0.01. В каждом тесте вычисляется P-значение: это вероятность того, что подопытный генератор произведет последовательность не хуже, чем гипотетический истинный. Если P - значение = 1, то наша последовательность идеально случайна, а если оно = 0, то последовательность полностью предсказуема. В дальнейшем P-значение сравнивается с α, и если она больше α, то нулевая гипотеза принимается и последовательность признается случайной. В противном случае — отбраковывается. В тестах берется α = 0.01. Из этого следует, что: Если P-значение ≥ 0.01, то последовательность признается случайной с уровнем доверия 99%. Если P-значение < 0.01, то последовательность отбраковывается с уровнем доверия 99%. Очевидно, что чем более случайна последовательность, тем ближе это соотношение к 1. Данный тест оценивает, насколько P близко к 1.



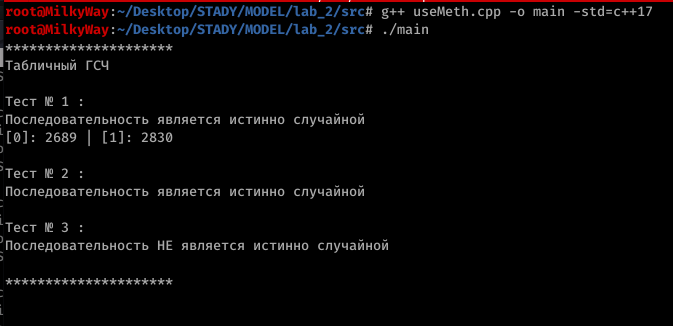
1. Тест на одинаковые идущие подряд биты. В тесте ищутся все последовательности одинаковых битов, а затем анализируется, насколько количество и размеры этих последовательностей соответствуют количеству и размерам истинно случайной последовательности. Смысл в том, что если смена 0 на 1 (и обратно) происходит слишом редко, то такая последовательность не случайная. Сначала вычисляем долю единиц в общей массе. Вычисляем суммарное число знакоперемен. Вычисляем P-значение через функцию ошибок.



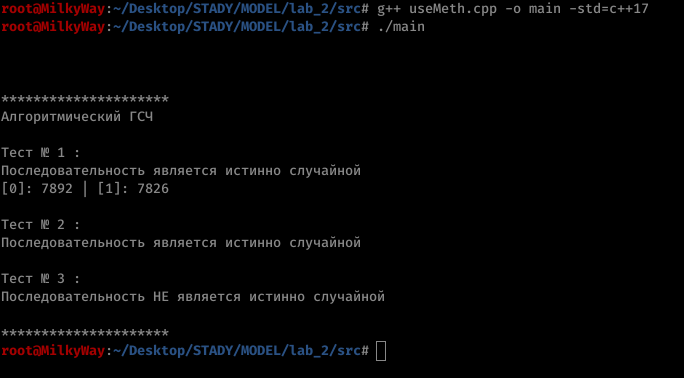
Для выполнения лабораторной работы использован язык програмирования C++ STL (стандарт языка C++17), для компиляции программы использован компилятор языка С++: g++( gcc version 9.2.1 20200123 (Debian 9.2.1-25) ). Ключ компиляции: g++ useMeth.cpp -o main -std=c++17. Программа скомпилирована под ОС Linux Debian( Debian 5.4.8-1kali1 (2020-01-06) x86\_64 GNU/Linux ).

Программа состоит из 3 классов: GenerateRandomNumericWithTables, GenerateRandomNumericWithAlgoritmic, RandomAnalysis. Первые два для генерации последовательностей табличным и алгоритмическим методом соответственно, третий для проведения тестов и анализа результатов. Для исследования табличного способа генерации случайной последовательности была взята таблица содержащая 500 абсолютно случайных проверенных чисел (взято из книги И. Г. Венецкого, В. И. Венецкой «Основные математико-статистические понятия и формулы в экономическом анализе»). Для исследования алгоритмического метода генерации случайной последовательности был программно реализован линейный конгруэнтный метод со следующими параметрами a10 = 1680710 ; c10 = 010 ; m10 = 214748364710 ( эти значения используются для генерации псевдослучайных чисел в C++11's minstd\_rand0(https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9\_%D0%BA%D0%BE%D0%BD%D0%B3%D1%80%D1%83%D1%8D%D0%BD%D1%82%D0%BD%D1%8B%D0%B9\_%D0%BC%D0%B5%D1%82%D0%BE%D0%B4) ).

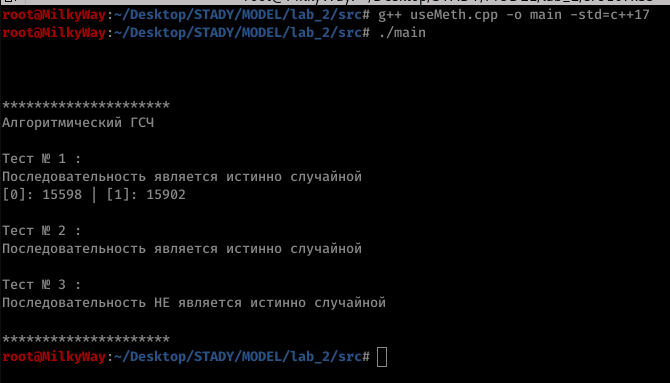
Для 500 чисел табличной генерации получены следующие результаты:



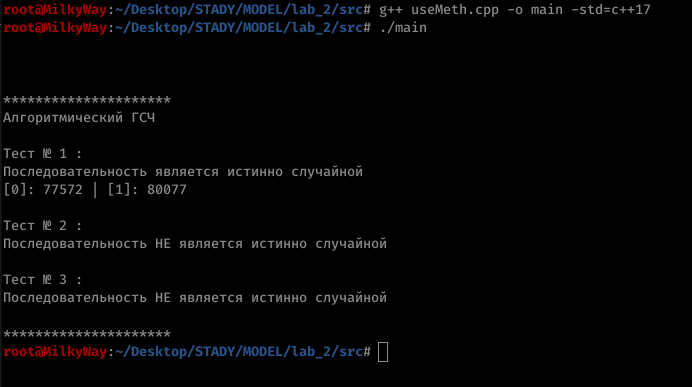
Для 500 чисел полученых алгоритмически :



Для 1000 чисел полученых алгоритмически :



Для 50000 чисел полученых алгоритмически :



Первый тест проходят все выборки чисел. Ни одна выборка чисел не проходит полностью все тесты. Алгоритмически сгенерированые числа начинают не проходить второй тест уже при объеме 50000 чисел. Добиться того чтобы алгоритмически сгенерированые числа не проходили первый тест не удалось.

Листинг программы.

**tab\_met.h**

#ifndef TAB\_MET\_H

#define TAB\_MET\_H

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

template< class Tdata >

class GenerateRandomNumericWithTables {

private :

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::vector< Tdata > m\_array ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public :

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

GenerateRandomNumericWithTables() {}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

GenerateRandomNumericWithTables( const GenerateRandomNumericWithTables &\_copy ) = delete ;

GenerateRandomNumericWithTables& operator=(const GenerateRandomNumericWithTables &other ) = delete ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

~GenerateRandomNumericWithTables() {}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::vector< Tdata > GET\_ARRAY() const { return m\_array ; }

void ADD\_ARRAY( Tdata value ) { m\_array.push\_back( static\_cast< Tdata >( value ) ) ; }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void ReadFile( const std::string &file ) {

std::ifstream input( file ) ;

std::string line ;

if( input ) {

while( getline( input, line, ' ') )

ADD\_ARRAY( std::stoi ( line, nullptr, 10 ) ) ;

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void PrintVector() {

for( const auto &itm : m\_array ) std::cout << itm << " " ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

} ; //class GenerateRandomNumericWithTables

#endif // TAB\_MET\_H

**algo\_met.h**

#ifndef ALGO\_MET\_H

#define ALGO\_MET\_H

#include <iostream>

#include <vector>

template< class Tdata >

class GenerateRandomNumericWithAlgoritmic {

private :

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::vector< Tdata > m\_array ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

const int64\_t LCG\_a = 16807 ;

const int64\_t LCG\_c = 0 ;

const int64\_t LCG\_m = 2147483647;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public :

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

GenerateRandomNumericWithAlgoritmic() {}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

GenerateRandomNumericWithAlgoritmic(const GenerateRandomNumericWithAlgoritmic &\_copy ) = delete ;

GenerateRandomNumericWithAlgoritmic&operator=( const GenerateRandomNumericWithAlgoritmic &other ) = delete ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

~GenerateRandomNumericWithAlgoritmic() {}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::vector< Tdata > GET\_ARRAY() const { return m\_array ; }

void ADD\_ARRAY( Tdata value ) { m\_array.push\_back( static\_cast< Tdata >( value ) ) ; }

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void LinearCongruentialGenerator() {

if(!m\_array.empty()) m\_array.push\_back((LCG\_a \* m\_array.back() + LCG\_c ) % LCG\_m) ;

else m\_array.push\_back( 1 ) ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void PrintVector() {

for( const auto &itm : m\_array ) std::cout << itm << " " ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

} ; // class GenerateRandomNumericWithAlgoritmic

#endif // ALGO\_MET\_H

**random\_analysis.h**

#ifndef RANDOM\_ANALYSIS\_H

#define RANDOM\_ANALYSIS\_H

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

#include <map>

#include <cmath>

#include <sstream>

template< class Tdata>

class RandomAnalysis {

private :

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::vector< Tdata > m\_array ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::map< Tdata, int64\_t > m\_map ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

public :

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

RandomAnalysis() {}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

RandomAnalysis( const RandomAnalysis &\_copy ) = delete ;

RandomAnalysis& operator=( const RandomAnalysis &other ) = delete ;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

~RandomAnalysis() {}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void ADD\_ARRAY( Tdata value ) {

m\_array.push\_back( static\_cast< Tdata >( value ) ) ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void ADD\_MAP() {

for( const auto &itm : m\_array ) ++m\_map[itm] ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::vector<Tdata> ConvertToBinary( const std::vector< Tdata > &array ) {

Tdata x ;

for( const auto &itm : array ) {

bool flag = false ;

for ( int i = (sizeof( Tdata ) << 3) ; i >= 0 ; i-- ) {

x= ((itm >> i) & 1) ;

if( x == 1 ) flag = true ;

if( flag ) ADD\_ARRAY( x ) ;

}

}

return ( m\_array ) ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

std::pair<Tdata, int64\_t> Test\_1() {

ADD\_MAP() ;

if( double(std::max(m\_map[0], m\_map[1]) / (double)(m\_map[0] + m\_map[1])) < 0.01 )

std::cout << "Последовательность НЕ является истинно случайной\n" ;

else std::cout << "Последовательность является истинно случайной\n" ;

return std::make\_pair( m\_map[0], m\_map[1] ) ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Test\_2() {

int res = abs(m\_map[1] - m\_map[0]) / (pow( 2\*(m\_map[0]+m\_map[1]) ,0.5 ) ) ;

if( erfc( res ) > 0.001 ) std::cout << "Последовательность является истинно случайной\n" ;

else std::cout << "Последовательность НЕ является истинно случайной\n" ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void Test\_3() {

int pi = m\_map[1] / (m\_map[0] + m\_map[1]) ;

int64\_t count\_ones = 0 ;

if( abs(pi-0.5) <= 2/ (pow(m\_map[0] + m\_map[1], 0.5)) ) {

auto begin = m\_array.begin() ;

auto end = m\_array.end () ;

while( begin != end ) {

int64\_t c = 0 ;

while( begin != end && \*begin == 1 ) {

if( begin != end ) begin++ ; c++ ;

}

if( c > 1 ) count\_ones++ ;

if( begin != end ) begin++ ;

}

if( erfc( (count\_ones-2\*pi\*(m\_map[0]+m\_map[1])\*(1-pi)) /

(2\*(pow((2\*(m\_map[0]+m\_map[1])),0.5))\*pi\*(1-pi) ) ) > 0.001 )

std::cout << "Последовательность является истинно случайной\n" ;

else std::cout << "Последовательность НЕ является истинно случайной\n" ;

} else std::cout << "Последовательность НЕ является истинно случайной\n" ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void PrintVector() {

for( const auto &itm : m\_array ) std::cout << itm << " " ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void PrintMap() {

for( const auto &[ k, v ] : m\_map ) std::cout << k << " : " << v << "\n" ;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

} ; // class RandomAnalysis

#endif // RANDOM\_ANALYSIS\_H

**useMeth.cpp**

#include<iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <numeric>

#include <map>

#include <cmath>

#include "tab\_met.h"

#include "algo\_met.h"

#include "random\_analysis.h"

int main() {

RandomAnalysis< int16\_t > \*RA\_1 = new RandomAnalysis< int16\_t >() ;

RandomAnalysis< int32\_t > \*RA\_2 = new RandomAnalysis< int32\_t >() ;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" ;

//Табличный ГСЧ

std::cout << "Табличный ГСЧ" << "\n\n" ;

GenerateRandomNumericWithTables< int16\_t > \*GRNWT = new GenerateRandomNumericWithTables< int16\_t >() ;

GRNWT->ReadFile( "500\_random\_numeric" ) ;

//GRNWT->PrintVector() ;

RA\_1->ConvertToBinary( GRNWT->GET\_ARRAY() ) ;

std::cout << "Тест № 1 : " << "\n" ;

std::pair< int16\_t, int64\_t >res\_test1 = RA\_1->Test\_1();

std::cout << "[0]: "<< res\_test1.first << " | " << "[1]: " << res\_test1.second << "\n\n";

std::cout << "Тест № 2 : " << "\n" ;

RA\_1->Test\_2(); std::cout << "\n" ;

std::cout << "Тест № 3 : " << "\n" ;

RA\_1->Test\_3(); std::cout << "\n" ;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" ;

std::cout << "\n\n\n" ;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" ;

//Алгоритмический ГСЧ

std::cout << "Алгоритмический ГСЧ" << "\n\n" ;

GenerateRandomNumericWithAlgoritmic<int32\_t>\*GRNWA=new GenerateRandomNumericWithAlgoritmic< int32\_t >() ;

for( size\_t i = 1 ; i < 500000 ; i++ ) GRNWA->LinearCongruentialGenerator() ;

RA\_2->ConvertToBinary( GRNWA->GET\_ARRAY() ) ;

std::cout << "Тест № 1 : " << "\n" ;

std::pair< int32\_t, int64\_t >res\_test2 = RA\_2->Test\_1();

std::cout << "[0]: "<< res\_test2.first << " | " << "[1]: " << res\_test2.second << "\n\n";

std::cout << "Тест № 2 : " << "\n" ;

RA\_2->Test\_2(); std::cout << "\n" ;

std::cout << "Тест № 3 : " << "\n" ;

RA\_2->Test\_3(); std::cout << "\n" ;

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n" ;

delete GRNWT, RA\_1, RA\_2, GRNWA ;

return 0 ;

}