Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Алгоритми та структури даних

Лабораторна робота №4

Тема: «ТЕСТУВАННЯ ГЕНЕРАТОРІВ ПСЕВДОВИПАДКОВИХ ЧИСЕЛ»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: викладач
Константинова Л. В.

Кропивницький 2026

Мета: Навчитися реалізовувати алгоритми тестування генераторів псевдовипадкових чисел.

Завдання: Протестувати отримані за допомогою розроблених у попередній лабораторній роботі генераторів випадкових чисел послідовності на однорідність.

Повний лістинг програми міститься в git репозиторії: https://github.com/movavok/Algorithms-labs/tree/main/lab_4

Лістинг *uniformitytest.h*:

```
#ifndef UNIFORMITYTEST_H
#define UNIFORMITYTEST_H

#include <vector>

#include "randomgenerator.h"

class UniformityTest {
private:
    int m_numbers;
    int m_intervals;

public:
    UniformityTest(int numbers, int intervals) : m_numbers(numbers),
m_intervals(intervals) {}

    double runTest(RandomGenerator& generator) {
        vector<int> bucket(m_intervals, 0);

        for (int count = 0; count < m_numbers; count++) {
            double random = generator.nextDouble();
            int index = (int)(random * m_intervals);
            bucket[index]++;
        }

        double expected = (double)m_numbers / m_intervals;
        double chiSquared = 0.0;

        for (int index = 0; index < m_intervals; index++)
            chiSquared += (bucket[index] - expected) * (bucket[index] -
expected) / expected;

        return chiSquared;
    }
};

#endif // UNIFORMITYTEST_H
```

Лістинг *main.cpp*:

```
#include <iostream>
#include <ctime>

#include "parkmiller.h"
#include "lecuyer.h"
#include "bbs.h"
#include "uniformitytest.h"

void printSequence(RandomGenerator& gen, int numbersAmount, const string&
title) {
    cout << title;
    for (int count = 0; count < numbersAmount; ++count)
        cout << gen.next() << " ";
    cout << endl;
}

void runUniformity(RandomGenerator& gen, const string& title, int numbers =
10000, int intervals = 10) {
    UniformityTest test(numbers, intervals);
    double chi2 = test.runTest(gen);
    cout << title << "X^2 = " << chi2 << endl << endl;
}

int main() {
    long long seed = time(nullptr);

    ParkMiller pmGen(seed);
    printSequence(pmGen, 5, "Мінімальний генератор Парка-Міллера: ");
    runUniformity(pmGen, "Тест рівномірності Парка-Міллера: ");

    LEcuyer lecGen(seed, seed / 2);
    printSequence(lecGen, 5, "Алгоритм Л'Екюера: ");
    runUniformity(lecGen, "Тест рівномірності Л'Екюера: ");

    const long long firstPrime = 10007;
    const long long secondPrime = 10009;

    BBS bbsGen(seed % (firstPrime * secondPrime - 2) + 2, firstPrime,
secondPrime);

    cout << "Алгоритм Блюма-Блюма-Шуба (біти): ";
    for (int count = 0; count < 32; ++count) {
        cout << bbsGen.next();
        if ((count + 1) % 8 == 0) cout << " ";
    }
    cout << endl;

    cout << "                              (як байти): ";
    for (int count = 0; count < 4; ++count)
        cout << bbsGen.nextByte() << " ";
    cout << endl;

    runUniformity(bbsGen, "Тест рівномірності Блюма-Блюма-Шуба: ");

    return 0;
}
```

Результати виконання:

```
Мінімальний генератор Парка-Міллера: 1999827421 832905550 1345167704 1673249159 1000257848
Тест рівномірності Парка-Міллера: X^2 = 7.792

Алгоритм Л'Екюера: 2022368284 1330381689 78427050 1982792944 1125045486
Тест рівномірності Л'Екюера: X^2 = 9.964

Алгоритм Блюма-Блюма-Шуба (біти): 00000011 01100011 01111110 10010001
                        (як байти): 238 65 140 125
Тест рівномірності Блюма-Блюма-Шуба: X^2 = 9.092
```

Висновки: У попередній лабораторній роботі були реалізовані три генератори псевдовипадкових чисел: мінімальний Парка-Міллера, Л’Екюера та Блюма-Блюма-Шуба. Для кожного створено методи отримання чисел і бітів, що дозволило підготувати дані для тесту рівномірності. Проведене χ^2-тестування показало, що всі генератори дають рівномірний розподіл: Лінійний $\chi^2 = 7.792$, комбінований $\chi^2 = 9.964$, криптографічний $\chi^2 = 9.092$. Теоретично для заданих 10 інтервалів очікуване $\chi^2 \approx k-1 = 9$, тому ці результати підтверджують правильну роботу генераторів, а використання великих простих чисел для ББШ забезпечує більш рівномірний розподіл.