МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»
Кафедра програмної інженерії та інформаційних технологій управління

Звіт з лабораторної роботи № 5
з дисципліни «Основи теорії алгоритмів»

Виконав:
ст. гр. КН-221в
Шулюпов Є.Р.
Перевірила:
доцент каф. ПІТУ
Солонська С.В.

Харків
2022

ТЕМА: КОМБІНАТОРНІ АЛГОРИТМИ

ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Розробити програму, яка читає з клавіатури число N (1 < N < 256) та парам1етри генератору випадкових чисел та виводить на екран послідовність з N згенерованих чисел. Програма зберігає до файлу графічну характеристику послідовності згідно завдання та виводить на екран результат одного з тестів NIST (згідно варіанту завдання).

Варіанти генераторів випадкових чисел.

1 Лінійний конгруентний метод.

2 Метод Фібоначчі із затримуванням.

Варіанти графічних характеристик.

1 Гістограма розподілу елементів послідовності.

2 Розподіл на площині (елементи попарно обробляються як координати точок (x, y)).

3 Автокореляція (користувач задає зсув для копії послідовності).

Варіанти тестів NIST.

1 Частотний побітовий тест.

2 Тест на послідовність однакових бітів.

3 Тест на найдовшу послідовність одиниць.

МЕТА РОБОТИ

Ознайомлення з алгоритмами псевдовипадкових послідовностей.

1 ОСНОВНІ ТЕОРЕТИЧНІ ПОЛОЖЕННЯ

Лінійний конгруентний метод застосовується в простих випадках і не володіє криптографічного стійкістю.

Лінійний конгруентний метод полягає у обчисленні членів лінійної рекурентної послідовності за модулем деякого натурального числа m, що задається наступною формулою: $Xk+1=(aXk+c)\mathrm{mod}m$, де aі c- деякі цілочисельні коефіцієнти. Отримана послідовність залежить від вибору стартового числа і при різних його значеннях виходять різні послідовності випадкових чисел. У той же час, багато властивостей цієї послідовності визначаються вибором коефіцієнтів у формулі і не залежать від вибору стартового числа.

Особливості розподілу випадкових чисел, що генеруються лінійним конгруентним алгоритмом, роблять неможливим їх використання в статистичних алгоритмах, що вимагають високого дозволу.

Один з широко поширених датчиків Фібоначчі. При реалізації через цілі числа досить формули. Для роботи датчику Фібоначчі потрібно знати $\max\{a,b\}$попередніх згенерованих випадкових чисел, які можуть бути згенеровані простим конгруентним датчиком.

Лаги a і b - «магічні» і їх не слід вибирати довільно. Рекомендуються наступні значення лагів: (a, b) = (55,24), (17,5) або (97,33).

Статистичні тести видають чисельну характеристику послідовності і дозволяють однозначно сказати, чи пройдено тест. Розглянемо кілька тестів пакету NIST.

Частотний побітовий тест полягає у визначенні співвідношення між нулями і одиницями у всій двійковій послідовності. Мета - з'ясувати, чи дійсно число нулів і одиниць у послідовності приблизно однакові, як це можна було б припустити у випадку істинно випадкової бінарної послідовності. Тест оцінює, наскільки близька частка одиниць до 0.5. Таким чином, число нулів і одиниць має бути приблизно однаковим. Якщо обчислена в ході тесту значення ймовірності p<0.01, то дана двійкова послідовність не є істинно випадковою. В іншому випадку, послідовність носить

випадковий характер. Варто відзначити, що всі наступні тести проводяться за умови, що пройдено даний тест.

Тест на послідовність однакових бітів полягає в підрахунку повного числа рядів у вихідній послідовності, де під словом «ряд» мається на увазі безперервна підпослідовність однакових бітів. Ряд довжиною kбіт складається з kабсолютно ідентичних бітів, починається і закінчується з біта, що містить протилежне значення. Мета даного тесту - зробити висновок про те, чи дійсно кількість рядів, що складаються з одиниць і нулів з різними довжинами, відповідає їх кількості у довільній послідовності. Зокрема, визначається швидко або повільно чергуються одиниці і нулі у вихідній послідовності. Якщо обчислена в ході тесту значення ймовірності p<0.01, то дана двійкова послідовність не є істинно випадковою. В іншому випадку, вона носить випадковий характер.

Тест на найдовшу послідовність одиниць полягає у визначенні найдовшого ряду одиниць всередині блоку довжиною mбіт. Мета - з'ясувати чи дійсно довжина такого ряду відповідає очікуванням довжини найпротяжнішого ряду одиниць у разі абсолютно довільній послідовності. Якщо вирахувана в ході тесту значення ймовірності p<0.01, то вихідна послідовність не є випадковою. В іншому випадку, робиться висновок про її випадковості. Слід зауважити, що з припущення про приблизно однаковою частотою появи одиниць і нулів (тест №1) випливає, що точно такі ж результати даного тесту будуть отримані при розгляді самого довгого ряду нулів. Тому вимірювання можна проводити тільки з одиницями.

2 ОПИСАННЯ РОЗРОБЛЕНОГО ЗАСТОСУНКУ

Програма реалізована в одному файлі source.cpp

```
include<iostream>
#include<fstream>
using namespace std;

double GammaFunct(double x, int k);
void Autocorelation(double* arr, int n)
{
	double result[300];
```

```
	int k;
	double sum1 = 0;
	double sum2 = 0;
	cout << "Введіть значення зсуву для автокореляційної функції"<<endl;
	cin >> k;
	if (k > 256)
	{
		cout << "Завеликий зсув" << endl;
		exit(0);
	}
	ofstream outFile("result.txt");
	for (int i = 0; i < n+k; i++)
	{
		for (int j = 0; j < n - 1; j++)
		{
			if (k + j + i < n)
				sum1 += arr[j] * arr[abs(k + i + j)];
			sum2 += arr[j] * arr[j];

		}
		result[i] = sum1 / sum2;
		sum1 = 0;
		sum2 = 0;
	}
	for (int i = 0; i < n; i++)
	{
		outFile << i+1 << " ітерація - " << result[i] << " " << endl;
	}

}
void Test(double* arr, int n)
{
	int secondArr[256];
	int result[256];
	const int K = 3;
	const int M = 8;

	for (int i = 0; i < n ; i++)
	{
		secondArr[i] = arr[i] * 10;
		result[i] = 0;
	}

	for (int i = 0; i < n; i++)
	{
		setlocale(LC_ALL, "Ukrainian");
		int k = 1;
		if (!secondArr[i])
		{
			break;
		}
		while (secondArr[i])
		{
			result[i] += (secondArr[i] % 2) * k;
			k *= 10;
			secondArr[i] /= 2;
		}

	}
	for (int i = 0; i < n; i++)
	{
		if (arr[i] < 0.5)
			result[i] = 0;
		else
```

```
                result[i] = 1;
        }
        int v[4];
        for (int i = 0; i < 4; i++)
        {
                v[i] = 0;
        }
        for (int i = 0; i < n; i++)
        {
                if (arr[i] == arr[i + 1] && arr[i + 1] == arr[i + 2] && arr[i + 2] == arr[i + 3] &&
arr[i + 3] == 1)
                        v[3]++;
                else if(arr[i] == arr[i + 1] && arr[i + 1] == arr[i + 2] && arr[i + 2] == 1)
                        v[2]++;
                else if (arr[i] == arr[i + 1] && arr[i + 1] == 1)
                        v[1]++;
                else if (arr[i] == 1)
                        v[0]++;
        }
        double p[4];
        p[0] = 0.2148;
        p[1] = 0.3672;
        p[2] = 0.2305;
        p[3] = 0.1875;
        double test = 0;
        for (int i = 0; i < 4; i++)
        {
                test += pow((v[i] - 16 * p[i]), 2) / (16 * p[i]);
        }
         cout << "Значення NIST тесту = "<<GammaFunct(test, K);

}
int main()
{
        setlocale(LC_ALL, "Ukr");
        int n, x1, x2;
        cout << "Введіть кількість чисел, що будуть згенеровані" << endl;
        cin >> n;
        if (n > 256)
        {
                cout << "Непрвильно введені дані" << endl;
                exit(0);
        }
        x1 = 5;
        x2 = 2;
        double arr[256];
        arr[0] = 0.2;
        arr[1] = 0.5;
        arr[2] = 0.1;
        arr[3] = 0.8;
        arr[4] = 0.7;
        for (int i = 5; i < n; i++)
        {
                if(arr[i - x1] >= arr[i - x2])
                        arr[i] = arr[i - x1] - arr[i - x2];
                else
                        arr[i] = arr[i - x1] - arr[i - x2] + 1;
                if (abs(arr[i] < 0.01))
                {
                        arr[i] = 0;
                }
        }
        for (int i = 0; i < n; i++)
        {
```

```
        cout <<"arr[" << i << "] = " << arr[i] << endl;
    }
    Autocorelation(arr, n);
    Test(arr, n);
    return 0;
}
```

Результат роботи представлений на рисунку 2.1.

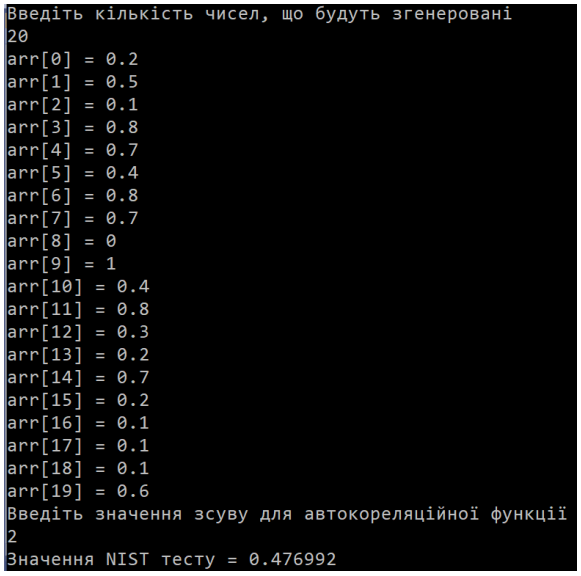

Рисунок 2.1 – Результат

Результат значень автокореляції записує до текстового файлу, а графік та ці значення зображені на рисунках 2.2 та 2.3.

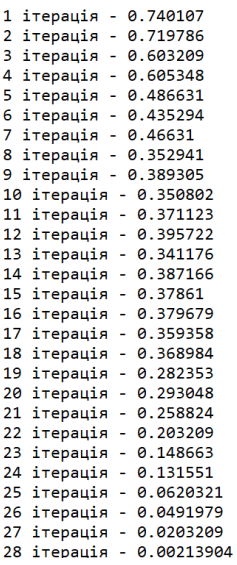

Рисунок 2.2 – Текстовий файл зі значеннями автокореляції

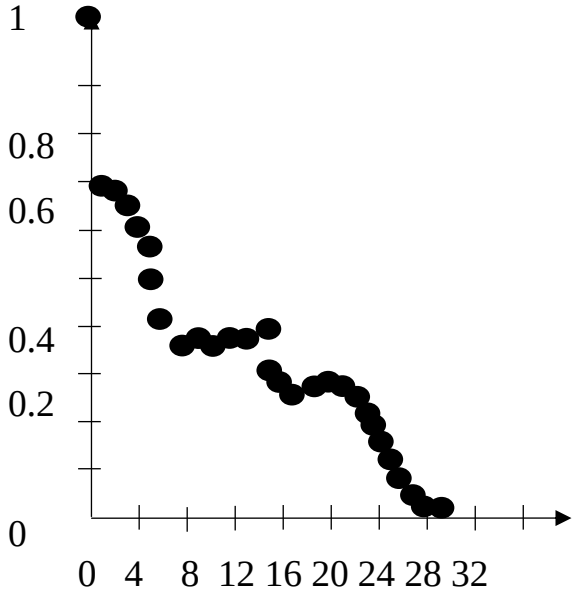

Рисунок 2.3 – Графік автокореляції

ВИСНОВКИ

Під час цієї лабораторної роботи була розроблена програма, яка генерує псевдорандомну послідовність, які після цього можна перевірити на випадковість за допомогою різних тестів та намалювати графік, що описує та показує випадковість даної послідовності. Ця програма показує всю суть псевдовипадкової послідовності та характеризує її з різних сторін.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Методичні вказівки до виконання лабораторних робіт з курсу "Алгоритми і структури даних": для студентів, які навчаються за спец. 121 "Інженерія програмного забезпечення" [Електронний ресурс] / уклад. Н. К. Стратієнко, І. О. Бородіна ; Харківський політехнічний інститут, національний технічний університет університет –

Електрон. текстові дані. – Харків, 2017. – 36 с. – Режим доступу: http://repository.kpi.kharkov.ua/handle/KhPI-Press/26426. (дата звернення: 25.03.2021)

2. Алгоритми і структури даних: практикум: навч. посіб./ Н.К. Стратієнко, М.Д. Годлевський, І.О. Бородіна.- Харьков: НТУ"ХПИ", 2017. - 224 с. (дата звернення: 25.03.2021)