Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение
высшего профессионального образования «Национальный исследовательский
университет «Высшая школа экономики»

Московский институт электроники и математики Национального исследовательского
университета "Высшая школа экономики"

Экзаминационный проект

по дисциплине "Программирование алгоритмов защиты информации

«Корреляционные тесты»

Выполнили

Кузьмина Ксения
Монахова Мария
Родионов Филипп
Япаров Матвей

Проверил

Нестеренко А.Ю.
Преподаватель дисциплины

Москва, 2024 г.

Содержание

1 Постановка задачи

Провести корреляционный анализ алгоритма выработки производных ключей, в основе которого лежит функция хеширования Стрибог512. Для данной работы были выбраны два теста – Знаковый корреляционный критерий Нелсона и Сериальный критерий Шведа-Эйзенхарта.

Далее будет рассмотрена сначала теоретическая часть, необходимая для применения этих тестов. Далее приведён код на языке C++ для анализа выборок различного размера из файлов.

2 Теоретическая часть

2.1 Знаковый корреляционный критерий Нелсона

2.1.1 Базовая информация

Критерий, предложенный Нелсоном позволяет установить наличие корреляции, непрерывно анализируя совместное поведение пар (x_i, y_i) по мере их появления в эксперименте. Критерий основан на числе знаков последовательного изменения величин пар (x_i, y_i).

Если $x_i > x_{i-1},\ y_i > y_{i-1}$ или $x_i < x_{i1},\ y_i < y_{i-1}$, то паре (x_i, y_i) приписывается знак +, в ином случае знак -. Другими словами, если значения пар (x_i, y_i) изменилось в одном направлении, то это отображается знаком +, в разных направлениях – знаком -. Если в паре одно или оба значения не изменилось, то ей приписывается значение 0.Статистикой критерия является наименьшее количество S знаков одного вида (+ или -). Корреляция признаётся значимой при $S > S_\alpha$ (S_α – критическое значение из справочной таблицы).

При $n > 90$ сумма S распределена асимптотически нормально и $S_\alpha \equiv \frac{n}{2} + u_{1-\alpha} \cdot \sqrt{\frac{11n-2}{36}}$.

2.1.2 Подробное обоснование

1. Гипотеза H_0 (независимость):
   - Предполагается, что x_i и y_i независимы.
   - Следовательно, направления изменений $x_i > x_{i-1},\ x_i < x_{i-1}$ и $y_i > y_{i-1},\ y_i < y_{i-1}$ также независимы.
2. Распределение знаков при H_0:
   - Для каждой пары (x_i, y_i) вероятность получить:
     - $+:\ \Pr(+|H_0) = 0.25$: оба числа растут или убывают;
     - $-:\ \Pr(-|H_0) = 0.25$: одно число растёт, второе убывает;
     - $0:\ \Pr(0|H_0) = 0.25$: любое неизменение равно нулю.
3. Пусть n – общее число значимых наблюдений (опускаем 0, рассматриваем только + и -). S_+ – число знаков +, S_- – число знаков -, где $S_+ + S_- = n$. Предполагается, что знаки + и - равновероятны при H_0, то есть вероятность получить знак + или - равна $p = 0.5$.
   - **Математическое ожидание**. S_+ следует биномиальному распределению
   $$S_+ \sim Binomial(n, p = 0.5)$$
   - **Среднее значение** для S_+:
   $$\mathbb{E}[S_+] = n \cdot p = \frac{n}{2}$$
   $$\mathbb{E}[S_-] = n \cdot (p-1) = \frac{n}{2}$$

- **Дисперсия**. Для биномиального распределения:

$$\mathrm{Var}[S_+] = n \cdot p \cdot (1-p) = n \cdot 0.5 \cdot 0.5 = \frac{n}{4}$$

$$\mathrm{Var}[S_-] = n \cdot p \cdot (1-p) = n \cdot 0.5 \cdot 0.5 = \frac{n}{4}$$

4. Статистика критерия $S = \min(S_+, S_-)$
   - Теперь вводим статистику $S = \min(S_+, S_-)$, которая используется для оценки минимального количества знаков одного типа. Так как было установлено, что $S_+ + S_- = n$, то статистика S зависит от распределения S_+.

     Так как $S_+ \sim Binomial(n, p = 0.5)$ при достаточно большом n биномиальное распределение можно приближать нормальным:

$$S_+ \sim N\left(\frac{n}{2}, \frac{n}{4}\right)$$

     Среднее значение $\mu = \frac{n}{2}$, дисперсия $\sigma^2 = \frac{n}{4}$.

     Соответственно, стандартное отклонение будет равно:

$$\sigma = \sqrt{\frac{n}{4}} = \frac{\sqrt{n}}{4}$$

5. Поскольку $S = \min(S_+, S_-)$ и нам известно, что S не может превышать $\frac{n}{2}$. При этом S меньше половины выборки только в случая, когда S_+ отклоняется от $\frac{n}{2}$. Чтобы определить критическое значение S_α нужно использовать нормальное приближение. На уровне значимости α, вероятность отклонения определяется через квантиль стандартного нормального распределения $u_{1-\alpha}$.
6. Вычисление S_α Для определения S_α используем теоретическое распределение знаков:

$$S_\alpha = \mathbb{E}[S] + u_{1-\alpha} \cdot \sigma_S$$

   - **Математическое ожидание**. Среднее значение S:

$$\mathbb{E}[S] = \frac{n}{2}$$

   - **Стандартное отклонение**. По теоретической модели Нелсона:

$$\mathrm{Var}[S] = \frac{11n-2}{36}$$

     Откуда стандартное отклонение

$$\sigma_S = \sqrt{\frac{11n-2}{36}}$$

   - **Учёт квантиля**.
     - Квантиль $u_{1-\alpha}$ зависит от уровня значимости α.
       - Для $\alpha = 0.05,\ u_{1-\alpha} = 1.645$;
       - Для $\alpha = 0.01,\ u_{1-\alpha} = 2.33$;
7. **Проверка гипотезы**.
   - Если $S \leq S_\alpha$, принимаем нулевую гипотезу H_0, предполагающую независимость.
   - Если $S \geq S_\alpha$, отвергаем нулевую гипотезу H_0, что свидетельствует о значимой корреляции между x_i и y_i.

2.2 Сериальный критерий Шведа-Эйзенхарта

2.2.1 Базовая информация

Совокупность пар (x_i, y_i) разбивается на две равные совокупности, отвечающие условиям $y_i > \tilde{y}$ и $y_i < \tilde{y}$ (где $\tilde{y}$ – медиана ряда y_i, при нечётном n значение $y_i = \tilde{y}$ исключается). Затем наблюдения ранжируются по возрастающим значениям y_i. Для последовательных пар значений (x_i, y_i) с $y_i > \tilde{y}$ будем применять символ a, для последовательных пар (x_i, y_i) с $y_i < \tilde{y}$ – символ b. В результате получим последовательность вида a, b, a, a, b. Последовательность элементов одного вида, ограниченная с двух сторон элементами другого вида (замыкающие интервал последовательности одного вида ограничены с одной стороны последовательностями другого вида), называется *серией*. Количество m серий является статистикой рассматриваемого критерия.

Корреляция признаётся значимой, если $m \leq m_\alpha$ (критические значения приведены в таблицах). При чётных $n > 40$ можно использовать приближение:

$$m_{0.95} = \left[\frac{n+1}{2} - 0.82 \cdot \sqrt{n-1}\right],$$

$$m_{0.99} = \left[\frac{n+1}{2} - 1.16 \cdot \sqrt{n-1}\right],$$

2.2.2 Подробное обоснование

Сериальный критерий Шведа-Эйзенхарта используется для проверки гипотезы о независимости последовательности знаков $\{+, -\}$. Основная идея — анализ числа серий в последовательности. Серия определяется как непрерывный участок одинаковых знаков. Например:

- Для последовательности $+++---+++$ количество серий $m = 3$.
- Для последовательности $+++++++$ количество серий $m = 1$.

1. Гипотеза H_0 (независимость):
   - Знаки + и - независимы и равновероятны.

   Гипотеза H_1 (независимость):
   - Знаки + и - не независимы (наблюдается корреляция).

   Если H_0 верна, число серий m должно быть близко к математическому ожиданию $\mathbb{E}[m]$, которое вычисляется на основе теории вероятностей. Отклонение от этого значения позволяет судить о корреляции.

2. **Математическое ожидание и дисперсия числа серий** m.

   Пусть $\{s_1, s_2, ..., s_n\}$ – последовательность знаков + и −. Число серий m определяется как:

   $$m = 1 + \sum_{i=1}^{n-1} I(s_i \neq s_{i+1})$$

   , где $I(s_i \neq s_{i+1})$ – индикатор смены знака:

   $$I(s_i \neq s_{i+1}) = \begin{cases} 1, & \text{если } s_i \neq s_{i+1}, \\ 0, & \text{если } s_i = s_{i+1} \end{cases}. \quad (1)$$

   (a) **Математическое ожидание** $\mathbb{E}[m]$.

   При гипотезе независимости H_0 вероятность смены знака между s_i и s_{i+1} равна 0.5, так как $\Pr(s_i = +) = \Pr(s_i = -) = 0.5$. Тогда математическое ожидание смены знаков:

   $$\mathbb{E}\left[\sum_{i=1}^{n-1} I(s_i \neq s_{i+1})\right] = (n-1) \cdot \Pr(s_i \neq s_{i+1}) = \frac{n-1}{2}$$

Добавим первую серию, которая есть всегда, для получения математического ожидания числа серий:

$$\mathbb{E}[m] = 1 + \frac{n-1}{2} = \frac{n+1}{2}.$$

(b) **Дисперсия** $\mathrm{Var}[m]$.

Для индикаторной функции $I(s_i \neq s_{i+1})$ дисперсия равна:

$$\mathrm{Var}[I(s_i \neq s_{i+1})] = \Pr(s_i \neq s_{i+1}) \cdot (1 - \Pr(s_i \neq s_{i+1})) = 0.5 \cot 0.5 = 0.25.$$

Так как индикаторы независимы при H_0 дисперсия суммы:

$$\mathrm{Var}\left[\sum_{i=1}^{n-1} I(s_i \neq s_{i+1})\right] = (n-1) \cdot 0.25 = \frac{n-1}{4}.$$

Добавляем первую серию, которая не вносит дисперсию, и получаем дисперсию числа серий:

$$\mathrm{Var}[m] = \frac{n-1}{4}$$

3. **Стандартное отклонение** σ_m.

Стандартное отклонение числа среий:

$$\sigma_m = \sqrt{\mathrm{Var}[m]} = \sqrt{\frac{n-1}{4}} = \frac{\sqrt{n-1}}{2}.$$

4. **Приближённое распределение числа серий** m.

При $n > 40$ распределение числа серий m приближается нормальным распределением:

$$m \sim N\left(\frac{n+1}{2}, \frac{\sqrt{n-1}}{2}\right)$$

5. **Критическое значение** m_α.

Для проверки гипотезы H_0 используется критическое значение m_α, которое зависит от уровня значимости α. При $n > 40$ значение m_α вычисляется как:

$$m_\alpha = \left[\frac{n+1}{2} - z_\alpha, \frac{\sqrt{n-1}}{2}\right]$$

, где Z_α – квантиль стандартного нормального распределения:

- $z_{0.05} = 1.645$ (для уровня значимости $\alpha = 0.05$);
- $z_{0.01} = 2.33$ (для уровня значимости $\alpha = 0.01$);

6. **Формулы для** m_α

- Для $\alpha = 0.05$:

$$m_{0.95} = \left[\frac{n+1}{2} - 0.82 \cdot \sqrt{n-1}\right]$$

.

- Для $\alpha = 0.01$:

$$m_{0.99} = \left[\frac{n+1}{2} - 1.16 \cdot \sqrt{n-1}\right]$$

.

7. **Валидность критерия**

- При независимости H_0:
  - Знаки $+$ и $-$ распределены случайно, и число серий m будет подчиняться теоретическому распределению $\mathbb{E} = \frac{n+1}{2}$ и $\mathrm{Var}[m] = \frac{n-1}{4}$.
  - При больших n это распределение приближается к нормальному.
- При зависимости H_1:
  - Если знаки зависимы, то m отклоняется от ожидаемого значения:
    - Кластеризация знаков $(+++---)$ уменьшает m.
    - Слишком частое чередование $(+-+-)$ уменьшает m.

3 Практическое применение

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>

#if defined(_WIN32) || defined(_WIN64)
#include <direct.h> // For Windows (_getcwd)
#define GetCWD _getcwd
#else
#include <unistd.h> // For Linux/macOS (getcwd)
#define GetCWD getcwd
#endif

using namespace std;

// Function to read nhmac values from the file
vector<string> readNHMACValues(const string &filename) {
    vector<string> nhmacValues;
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error: Failed to open file " << filename << endl;

        char current_path[FILENAME_MAX];
        if (GetCWD(current_path, sizeof(current_path))) {
            cout << "Current working directory: " << current_path << endl;
        } else {
            cerr << "Error: Unable to determine the current working directory." << endl;
        }
        return nhmacValues;
    }

    string line;
    while (getline(file, line)) {
        // Look for the line starting with "nhmac:"
        if (line.rfind("nhmac:", 0) == 0) { // Check if the line starts with "nhmac:"
            string nhmac = line.substr(6); // Extract the part after "nhmac:"
            nhmac.erase(remove_if(nhmac.begin(), nhmac.end(), ::isspace), nhmac.end());
    // Remove any whitespace
            nhmacValues.push_back(nhmac);
        }
    }

    file.close();
    return nhmacValues;
}

// Function to calculate median using lexicographical order
string calculateMedian(vector<string> &yValues) {
    sort(yValues.begin(), yValues.end()); // Sort lexicographically

```

```
    size_t n = yValues.size();
    if (n % 2 == 0) {
        return yValues[n / 2 - 1]; // For even n, return the lower middle value
    } else {
        return yValues[n / 2]; // For odd n, return the middle value
    }
}

// Function to generate the symbol sequence (a, b) based on the median
vector<char> generateSymbolSequence(vector<pair<string, string>> &keyPairs, const string
     &median) {
    vector<char> symbols;

    for (const auto &pair : keyPairs) {
        const string &yValue = pair.second;
        if (yValue < median) {
            symbols.push_back('a');
        } else if (yValue > median) {
            symbols.push_back('b');
        }
    }

    return symbols;
}

// Function to sort pairs by x-values
void sortPairsByX(vector<pair<string, string>> &keyPairs) {
    sort(keyPairs.begin(), keyPairs.end(), [](const pair<string, string> &a, const pair<
    string, string> &b) {
        return a.first < b.first;
    });
}

// Function to count the number of series in the symbol sequence
int countSeries(const vector<char> &symbols) {
    if (symbols.empty()) return 0;

    int seriesCount = 1; // Start with one series
    for (size_t i = 1; i < symbols.size(); ++i) {
        if (symbols[i] != symbols[i - 1]) {
            seriesCount++;
        }
    }
    return seriesCount;
}

// Calculation of critical series value m_alpha
int calculateCriticalSeries(int n, double alpha) {
    double criticalValue;
    if (alpha == 0.05) {
        criticalValue = (n + 1) / 2.0 - 0.82 * sqrt(n - 1);
    } else if (alpha == 0.01) {
        criticalValue = (n + 1) / 2.0 - 1.16 * sqrt(n - 1);
    } else {
        cerr << "Unsupported alpha = " << alpha << ". Defaulting to 0.05." << endl;
        criticalValue = (n + 1) / 2.0 - 0.82 * sqrt(n - 1);
    }
    return static_cast<int>(round(criticalValue)); // Round to nearest integer
}

// Nelson's Criterion
int nelsonTest(const vector<pair<string, string>> &keyPairs, int &n) {
    vector<int> signs;
    for (size_t i = 1; i < keyPairs.size(); ++i) {
        int deltaX = keyPairs[i].first.compare(keyPairs[i - 1].first);
        int deltaY = keyPairs[i].second.compare(keyPairs[i - 1].second);

        // Determine the sign of the pair
        if ((deltaX > 0 && deltaY > 0) || (deltaX < 0 && deltaY < 0)) {
            signs.push_back(1); // "+"
```

```
        } else if ((deltaX > 0 && deltaY < 0) || (deltaX < 0 && deltaY > 0)) {
            signs.push_back(-1); // "-"
        } else {
            signs.push_back(0); // "0"
        }
    }

    // Count the number of "+" and "-"
    int plusCount = count(signs.begin(), signs.end(), 1);
    int minusCount = count(signs.begin(), signs.end(), -1);

    // n - total number of "+" and "-"
    n = plusCount + minusCount;

    // Return the minimum of "+" and "-"
    return min(plusCount, minusCount);
}

// Function to write results to a CSV file
void writeResultsToCSV(const string &filename, const string &testFile, int n, int
    nelsonS, double criticalSAlpha,
                       int series, int criticalM, bool independenceS, bool independenceM
    ) {
    ofstream file(filename, ios::app); // Append mode
    if (!file.is_open()) {
        cerr << "Error: Unable to open file " << filename << " for writing results." <<
    endl;
        return;
    }

    // Write headers if the file is empty
    static bool headerWritten = false;
    if (!headerWritten) {
        file << "TestFile,n,NelsonS,CriticalSAlpha,Series,CriticalM,NelsonIndependence,
    SeriesIndependence\n";
        headerWritten = true;
    }

    // Write test data
    file << testFile << "," << n << "," << nelsonS << "," << criticalSAlpha << ","
         << series << "," << criticalM << "," << (independenceS ? "1" : "0") << ","
         << (independenceM ? "1" : "0") << "\n";

    file.close();
}


// Main function to perform the tests
void performTests(const string &filename, double alpha) {
    auto nhmacValues = readNHMACValues(filename);
    if (nhmacValues.size() < 2) {
        cerr << "Insufficient data for analysis in file: " << filename << endl;
        return;
    }

    // Convert nhmacValues to key pairs
    vector<pair<string, string>> keyPairs;
    for (size_t i = 0; i < nhmacValues.size() - 1; ++i) {
        keyPairs.emplace_back(nhmacValues[i], nhmacValues[i + 1]);
    }

    // Nelson's Test
    int n = 0;
    int nelsonS = nelsonTest(keyPairs, n);
    double criticalSAlpha = n / 2.0 + 1.645 * sqrt((11.0 * n - 2.0) / 36.0);
    bool independenceS = nelsonS <= criticalSAlpha;

    // Swed-Eisenhart Serial Criterion
    vector<string> yValues;
    for (const auto &pair : keyPairs) {
```

```
        yValues.push_back(pair.second);
    }

    string median = calculateMedian(yValues);
    sortPairsByX(keyPairs);
    auto symbols = generateSymbolSequence(keyPairs, median);
    int series = countSeries(symbols);
    int criticalM = calculateCriticalSeries(n, alpha);
    bool independenceM = series <= criticalM ;

    // Output results
    cout << "File: " << filename << endl;
    cout << "Sample size: " << n << " data points" << endl;
    cout << "Nelson's Criterion (S-statistic): " << nelsonS
         << " (critical value S_alpha = " << criticalSAlpha << ")" << endl;
    cout << "Nelson's result: " << (independenceS ? "Independence" : "Dependence") <<
    endl;
    cout << "Swed-Eisenhart Serial Criterion (m): " << series
         << " (critical value m for alpha = " << alpha << ": " << criticalM << ")" <<
    endl;
    cout << "Swed-Eisenhart result: " << (independenceM ? "Dependence" : "Independence")
     << endl;
    cout << endl;

    // Write results to CSV
    writeResultsToCSV("results.csv", filename, n, nelsonS, criticalSAlpha, series,
    criticalM, independenceS, independenceM);
}

int main() {
    vector<string> files = {
        "test_results_100_0.txt",
        "test_results_200_0.txt",
        "test_results_300_0.txt",
        "test_results_400_0.txt",
        "test_results_500_0.txt",
        "test_results_600_0.txt",
        "test_results_700_0.txt",
        "test_results_800_0.txt",
        "test_results_900_0.txt",
    };

    double alpha = 0.05; // Significance level

    for (const auto &file : files) {
        performTests(file, alpha);
    }

    return 0;
}
```

Листинг 1: Реализация обработки поданных на вход файлов и корреляционных тестов

Теперь рассмотрим вывод в **result.csv**:

TestFile	n	NelsonS	CriticalSAlpha	Series	CriticalM	Nelson Independence	SeriesIndependence
test_results_100_0.txt	98	32	57.9933	54	41	1	1
test_results_200_0.txt	198	71	111.789	106	88	1	1
test_results_300_0.txt	298	94	164.692	154	135	1	1
test_results_400_0.txt	398	120	217.136	198	183	1	1
test_results_500_0.txt	498	168	269.288	250	231	1	1
test_results_600_0.txt	598	182	321.233	316	279	1	1
test_results_700_0.txt	696	231	371.986	350	327	1	1
test_results_800_0.txt	798	278	424.684	392	376	1	1
test_results_900_0.txt	898	304	476.246	464	425	1	1

Таблица 1: Таблица result.csv

Ниже приведены графики, показывающие для каждого критерия зависимости S, S_α и m m_αот размеров выборки: