**Лабораторная работа № 6**

**Изучение устройства и функциональных особенностей шифровальной машины «Энигма»**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации перестановочных шифров (рассчитана на 4 часа аудиторных занятий).

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости подстановочно-перестановочных шифров.

2. Изучить структуру, принципы функционирования, реализацию процедур зашифрования сообщений в машинах семейства Энигма.

3. Изучить и приобрести практические навыки выполнения криптопреобразований информации на платформе Энигма, реализованной в виде симуляторов.

4. Получить практические навыки оценки криптостойкости подстановочных и перестановочных шифров на платформе Энигма.

5. Результаты выполнения лабораторной работы оформить в виде отчета

проведенных исследованиях, методики выполнения практической части задания и оценки криптостойкости шифров.

**Вариант 15 (5)**.

**Основное** **задание**.

1. Ознакомиться с функционалом хотя бы одного (по согласованию с преподавателем) симулятора Энигмы:

1.1 Симулятор Энигмы М3 (М3 Enigma Simulator)

https://cryptocellar.org/simula/m3/index.html

1.2 Симулятор Энигмы М4 (М4 Enigma Simulator)

https://cryptocellar.org/simula/m4/index.html

1.3 Симулятор Энигмы Army/Air Force and the Railway

https://cryptocellar.org/simula/enigma/index.html

1.5 Симулятор Энигмы для Абвера (Abwehr Enigma Simulator)

https://cryptocellar.org/simula/abwehr/index.html

1.5 Симулятор Энигмы для Тирпица (Т (Tirpitz) Enigma Simulator)

https://cryptocellar.org/simula/tirpitz/index.html

Произвести зашифрование сообщения (собственные имя, отчество, фамилия) при 8-10 различных настройках машины-симулятора. Оценить частотные свойства символов в шифртекстах и сравнить этот параметр с частотными свойствами символов для исходного текста.

2. Разработать приложение-симулятор шифровальной машины, состоящей из клавиатуры, трех роторов и отражателя. Типы роторов (L-M-R) и отражателя Re следует выбрать из таблиц на рис. 4.5 и 4.6 в соответствии со своим вариантом, представленным в таблице 4.1. Крайний правый столбец этой таблицы показывает, на какое число шагов (букв, i) перемещается соответствующий ротор при зашифровании одного (текущего) символа; число 0 означает перемещение соответствующего ротора на один шаг при условии, что расположенный правее ротор совершит один оборот.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 15 | VII | Gamma | II | B Dunn | 1-2-2 |

С помощью разработанного приложения зашифровать сообщение в соответствии с п.1 практического задания, применив не менее 5 вариантов начальных установок роторов.

Оценить криптостойкость вашего варианта машины.

3. Результаты оформить в виде отчета по установленным правилам.

**Результат работы:**

Приложение написано на языке JS. Для запуска нужен Node.js свежей версии. Команда запуска “npm start”

Приложение

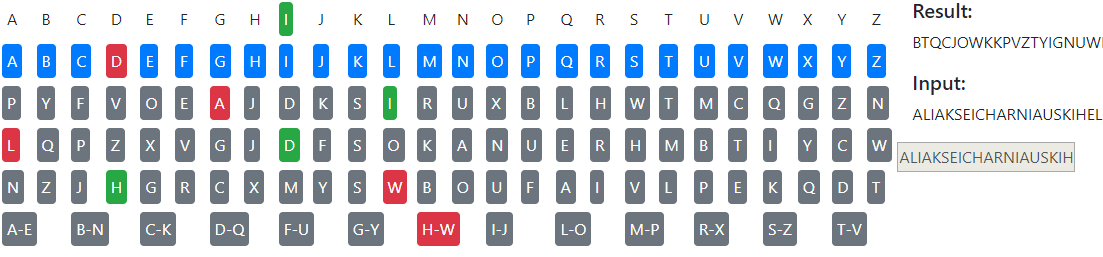


График встречаемости исходных данных:

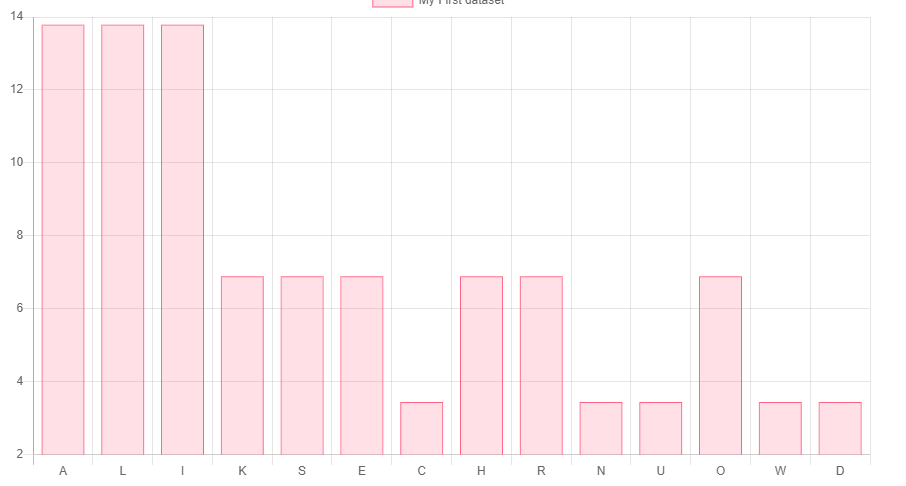
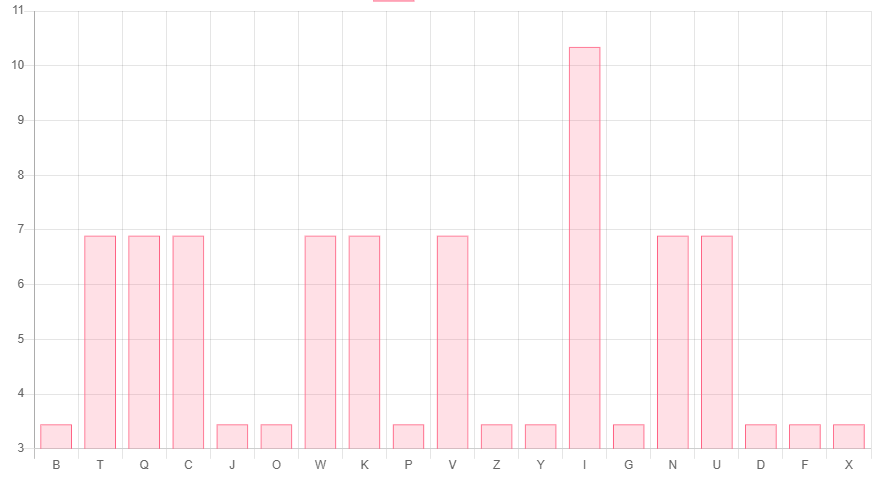
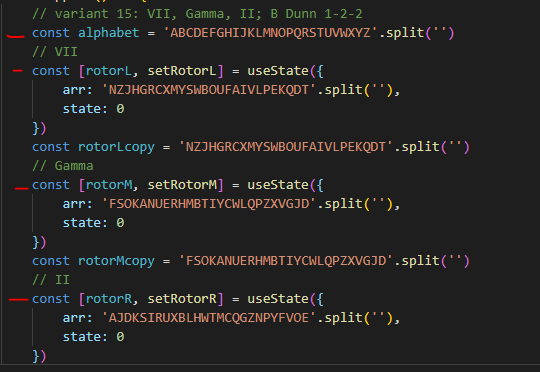


График частот зашифрованного сообщения:



Некоторые листинги:



Функция для вычисления зашифрованного символа:

    const calcResultSymbol = (key) => {

        const pressedIndex = alphabet.indexOf(key)

        setCurRotorR(rotorR.arr[pressedIndex])

        const cRotorR = rotorR.arr[pressedIndex]

        const cRotorM = rotorM.arr[alphabet.indexOf(rotorR.arr[pressedIndex])]

        const cRotorL = rotorL.arr[alphabet.indexOf(cRotorM)]

        setCurRotorM(cRotorM)

        setCurRotorL(cRotorL)

        let cRotorLBack = ''

        if ([...reflector.keys()].find(key => key === cRotorL)) {

            cRotorLBack = reflector.get(cRotorL)

        } else {

            cRotorLBack = [...reflector].find(([key, value]) => cRotorL === value)[0]

        }

        setCurRotorLBack(cRotorLBack)

        const cRotorMBack = alphabet[rotorL.arr.indexOf(cRotorLBack)]

        setCurRotorMBack(cRotorMBack)

        const cRotorRBack = alphabet[rotorM.arr.indexOf(cRotorMBack)]

        setCurRotorRBack(cRotorRBack)

        const resButton = alphabet[rotorR.arr.indexOf(cRotorRBack)]

        setResSym(resButton)

        setResultText(resultText + resButton)

        console.log(rotorLOffset + rotorL.state)

        console.log(rotorMOffset + rotorM.state)

        console.log(rotorROffset + rotorR.state)

        if (rotorR.state + rotorROffset < rotorR.arr.length) {

            setRotorR({ arr: rotorRcopy.splice(-rotorR.state - rotorROffset).concat(rotorRcopy), state: rotorR.state + rotorROffset })

            setRotorM({ ...rotorM })

            setRotorL({ ...rotorL })

        } else if (rotorM.state + rotorMOffset < rotorM.arr.length) {

            setRotorR({ arr: rotorRcopy, state: 0 })

            setRotorM({ arr: rotorMcopy.splice(-rotorM.state - rotorMOffset).concat(rotorMcopy), state: rotorM.state + rotorMOffset })

            setRotorL({ ...rotorL })

        } else if (rotorL.state + rotorLOffset < rotorL.arr.length) {

            setRotorR({ arr: rotorRcopy, state: 0 })

            setRotorM({ arr: rotorMcopy, state: 0 })

            setRotorL({ arr: rotorLcopy.splice(-rotorL.state - rotorLOffset).concat(rotorLcopy), state: rotorL.state + rotorLOffset })

        } else {

            setRotorR({ arr: rotorRcopy, state: 0 })

            setRotorM({ arr: rotorMcopy, state: 0 })

            setRotorL({ arr: rotorLcopy, state: 0 })

        }

        return resultText + resButton

    }