МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет-приложений)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ:

по дисциплине «Информационная безопасность»

Исполнитель

студент (ка) 3 курса группы 6 Розель Станислав Александрович

(Ф.И.О.)

Минск 2025

Исходный текст

Зашифрованный текст (маршрутные перестановки)

Зашифрованный текст (метод множественных перестановок)

Время шифрования метод маршрутной перестановки: 0.977ms

Время расшифровки: 0.597ms

Время шифрования метод множественной перестановки: 1.766ms

Время расшифровки: 1.512ms

|  |
| --- |
| const fs = require('fs');  const BELARUSIAN\_ALPHABET = 'абвгдеёжзійклмнопрстуўфхцчшщъыьэюя';  function cleanText(text, removePunctuation = true) {  if (removePunctuation) {  return text  .toLowerCase()  .replace(/[^\sа-яёіўA-ЯЁІЎa-zA-Z]/g, '')  .replace(/\s+/g, '')  .trim();  } else {  return text  .toLowerCase()  .replace(/\s+/g, '')  .trim();  }  }  function countCharacters(text, label) {  const charCount = {};  for (const char of text) {  charCount[char] = (charCount[char] || 0) + 1;  }  const charArray = Object.entries(charCount).sort((a, b) => b[1] - a[1]);  console.log(`\n=== Количество символов в тексте (${label}) ===`);  console.log('Символ | Количество');  console.log('-----------------');  charArray.forEach(([char, count]) => {  const displayChar = char === ' ' ? '[пробел]' : char;  console.log(`'${displayChar}' | ${count}`);  });  console.log(`\nВсего уникальных символов: ${charArray.length}`);  console.log(`Общая длина текста: ${text.length} символов`);  }  function readTextFromFile(filePath) {  return new Promise((resolve, reject) => {  fs.readFile(filePath, 'utf8', (err, data) => {  if (err) reject(new Error(`Ошибка чтения файла: ${err.message}`));  else resolve(data);  });  });  }  function writeTextToFile(filePath, text) {  return new Promise((resolve, reject) => {  fs.writeFile(filePath, text, 'utf8', (err) => {  if (err) reject(new Error(`Ошибка записи в файл: ${err.message}`));  else resolve();  });  });  }  function routeTranspositionEncrypt(text, rows, cols) {  let cipherText = '';  const blockSize = rows \* cols;    for (let start = 0; start < text.length; start += blockSize) {  let block = text.substring(start, start + blockSize);  while (block.length < blockSize) {  block += ' '; // дополнение пробелами  }    const matrix = [];  for (let i = 0; i < rows; i++) {  matrix[i] = block.slice(i \* cols, (i + 1) \* cols).split('');  }    for (let j = 0; j < cols; j++) {  for (let i = 0; i < rows; i++) {  cipherText += matrix[i][j];  }  }  }    return cipherText;  }    function routeTranspositionDecrypt(cipherText, rows, cols) {  let plainText = '';  const blockSize = rows \* cols;    for (let start = 0; start < cipherText.length; start += blockSize) {  let block = cipherText.substring(start, start + blockSize);  const matrix = [];    for (let j = 0; j < cols; j++) {  for (let i = 0; i < rows; i++) {  if (!matrix[i]) matrix[i] = [];  matrix[i][j] = block[j \* rows + i] || ' ';  }  }    for (let i = 0; i < rows; i++) {  plainText += matrix[i].join('');  }  }    return plainText.trim();  }  function createKeyFromKeyword(keyword, alphabet) {  const uniqueChars = [...new Set(keyword.toLowerCase())].filter((char) =>  alphabet.includes(char)  );  const sortedChars = [...uniqueChars].sort();  return uniqueChars.map((char) => sortedChars.indexOf(char));  }    function multipleTranspositionEncrypt(text, key1, key2, alphabet) {  const perm1 = createKeyFromKeyword(key1, alphabet);  const perm2 = createKeyFromKeyword(key2, alphabet);    const cols1 = perm1.length;  const rows1 = Math.ceil(text.length / cols1);  while (text.length < rows1 \* cols1) {  text += ' ';  }    const matrix1 = [];  for (let i = 0; i < rows1; i++) {  matrix1.push(text.slice(i \* cols1, (i + 1) \* cols1).split(''));  }    const intermediateText = perm1  .map((col) => matrix1.map((row) => row[col]).join(''))  .join('');    const cols2 = perm2.length;  const rows2 = Math.ceil(intermediateText.length / cols2);  while (intermediateText.length < rows2 \* cols2) {  intermediateText += ' ';  }    const matrix2 = [];  for (let i = 0; i < rows2; i++) {  matrix2.push(intermediateText.slice(i \* cols2, (i + 1) \* cols2).split(''));  }    return perm2  .map((col) => matrix2.map((row) => row[col]).join(''))  .join('');  }    function multipleTranspositionDecrypt(cipherText, key1, key2, alphabet) {  const perm1 = createKeyFromKeyword(key1, alphabet);  const perm2 = createKeyFromKeyword(key2, alphabet);    const cols2 = perm2.length;  const rows2 = Math.ceil(cipherText.length / cols2);  const matrix2 = Array.from({ length: rows2 }, () =>  new Array(cols2).fill('')  );    let index = 0;  for (const col of perm2) {  for (let i = 0; i < rows2; i++) {  matrix2[i][col] = cipherText[index++] || ' ';  }  }    const intermediateText = matrix2.map((row) => row.join('')).join('');    const cols1 = perm1.length;  const rows1 = Math.ceil(intermediateText.length / cols1);  const matrix1 = Array.from({ length: rows1 }, () =>  new Array(cols1).fill('')  );    index = 0;  for (const col of perm1) {  for (let i = 0; i < rows1; i++) {  matrix1[i][col] = intermediateText[index++] || ' ';  }  }    return matrix1.map((row) => row.join('')).join('').trim();  }  async function processFile(inputFilePath, outputFilePath, encryptionType, params) {  try {  let text = await readTextFromFile(inputFilePath);  console.log(`Исходный текст (${text.length} знаков):`);  console.log(text.substring(0, 100) + '...');    const removePunctuation = params.removePunctuation || false;  text = cleanText(text, removePunctuation);  console.log(`\nОчищенный текст (${text.length} знаков):`);  console.log(text.substring(0, 100) + '...');    countCharacters(text, "Исходный текст");    let processedText;    if (encryptionType === 'route') {  const { rows, cols, mode } = params;  if (mode === 'encrypt') {  console.time('Execution Time');    processedText = routeTranspositionEncrypt(text, rows, cols);  console.log(`\nЗашифрованный текст (маршрутная перестановка, ${rows}x${cols}):`);  console.timeEnd('Execution Time');  } else {  console.time('Execution Time');  processedText = routeTranspositionDecrypt(text, rows, cols);  console.log(`\nРасшифрованный текст (маршрутная перестановка, ${rows}x${cols}):`);  console.timeEnd('Execution Time');  }  } else if (encryptionType === 'multiple') {  const { key1, key2, mode } = params;  if (mode === 'encrypt') {  console.time('Execution Time');  processedText = multipleTranspositionEncrypt(text, key1, key2, BELARUSIAN\_ALPHABET);  console.log(`\nЗашифрованный текст (множественная перестановка, ключи: "${key1}", "${key2}"):`);  console.timeEnd('Execution Time');  } else {  console.time('Execution Time');  processedText = multipleTranspositionDecrypt(text, key1, key2, BELARUSIAN\_ALPHABET);  console.log(`\nРасшифрованный текст (множественная перестановка, ключи: "${key1}", "${key2}"):`);  console.timeEnd('Execution Time');  }  } else {  throw new Error(`Неизвестный тип шифрования: ${encryptionType}`);  }    console.log(processedText.substring(0, 100) + '...');  console.log(`\nДлина результата: ${processedText.length} знаков`);    const resultLabel = params.mode === 'encrypt' ? "Зашифрованный текст" : "Расшифрованный текст";  countCharacters(processedText, resultLabel);    await writeTextToFile(outputFilePath, processedText);  console.log(`\nРезультат успешно записан в файл: ${outputFilePath}`);    return processedText;  } catch (error) {  console.error('Ошибка при обработке файла:', error.message);  throw error;  }  }  async function main() {  const args = process.argv.slice(2);    try {  const removePunctuation = args[6] === 'true';  if (encryptionType === 'route') {  const rows = parseInt(args[4], 10);  const cols = parseInt(args[5], 10);  if (isNaN(rows) || isNaN(cols) || rows <= 0 || cols <= 0) {  throw new Error('Неверные параметры rows и cols! Должны быть положительными числами.');  }  await processFile(inputFile, outputFile, 'route', { rows, cols, mode, removePunctuation });  } else if (encryptionType === 'multiple') {  const key1 = args[4];  const key2 = args[5];  if (!key1 || !key2) {  throw new Error('Необходимо указать два ключевых слова!');  }  await processFile(inputFile, outputFile, 'multiple', { key1, key2, mode, removePunctuation });  } else {  throw new Error(`Неизвестный тип шифрования: ${encryptionType}`);  }  } catch (error) {  console.error('Ошибка:', error.message);  }  }  main().catch(console.error); |