Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надёжность информационных систем»

**Лабораторная работа №4**

**Тема «ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. ИТЕРАТИВНЫЕ КОДЫ»**

Выполнил:

Студент 3 курса 7 группы ФИТ

Тимошенко Д. В.   
 Проверила:   
 асс. Николайчук А. С.

Минск 2024

**Цель работы:** приобретение практических навыков кодирования/декодирования двоичных данных при использовании кода Хемминга.

**Задание на лабораторную работу**

1. Разработать собственное приложение, которое позволяет выполнять следующие операции:

1) вписывать произвольное двоичное представление информационного слова Хk (кодируемой информации) длиной k битов в двумерную матрицу размерностью в соответствии с вариантом, либо в трехмерную матрицу в соответствии с вариантом (указаны в табл. 5.2);

2) вычислять проверочные биты (биты паритетов):

а) по двум;

б) по трем;

в) по четырем направлениям (группам паритетов);

3) формировать кодовое слово Xn присоединением избыточных символов к информационному слову;

4) генерировать ошибку произвольной кратности (i, i > 0), распределенную случайным образом среди символов слова Xn, в результате чего формируется кодовое слово Y n;

5) определять местоположение ошибочных символов итеративным кодом в слове Y n в соответствии с используемыми группами паритетов по пункту (2) и исправлять ошибочные символы

(результат исправления – слово Y n’);

6) выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов Xn и Y n’; результат анализа может быть представлен

в виде отношения общего числа сгенерированных кодовых слов с ошибками определенной одинаковой кратности (с одной ошибкой, с двумя ошибками и т. д.) к числу кодовых слов, содержащих ошибки этой кратности, которые правильно обнаружены и которые правильно скорректированы.

Вариант 1

Приложение написано на языке программирования JavaScript. Ниже представлен листинг функций, используемых во всех задания, листинг 1.1.

|  |
| --- |
| function insertInfoWordToMatrix(infoWord, k1, k2, z) {  const rows = Math.ceil(infoWord.length / k1);  const columns = k2 + 1;  const matrix = Array(rows).fill().map(() => Array(columns).fill().map(() => Array(z+1).fill(0)));  for (let i = 0; i < rows; i++) {  for (let j = 0; j < columns; j++) {  if (i \* k1 + j < infoWord.length) {  matrix[i][j][0] = infoWord[i \* k1 + j];  }  }  }  return matrix;  }  function calculateParityBits(matrix) {  const rows = matrix.length;  const columns = matrix[0].length;  const z = matrix[0][0].length;  const horizontalParity = Array(rows).fill(0);  const verticalParity = Array(columns).fill(0);  const diagonalParityTopLeftToBottomRight = Array(Math.min(rows, columns)).fill(0);  const diagonalParityBottomLeftToTopRight = Array(Math.min(rows, columns)).fill(0);  for (let i = 0; i < rows; i++) {  for (let j = 0; j < columns; j++) {  for (let k = 0; k < z; k++) {  horizontalParity[i] ^= matrix[i][j][k];  }  }  }  for (let j = 0; j < columns; j++) {  for (let i = 0; i < rows; i++) {  for (let k = 0; k < z; k++) {  verticalParity[j] ^= matrix[i][j][k];  }  }  }  for (let i = 0; i < Math.min(rows, columns); i++) {  for (let k = 0; k < z; k++) {  diagonalParityTopLeftToBottomRight[i] ^= matrix[i][i][k];  }  }  for (let i = 0; i < Math.min(rows, columns); i++) {  for (let k = 0; k < z; k++) {  diagonalParityBottomLeftToTopRight[i] ^= matrix[rows - 1 - i][i][k];  }  }  return {  horizontalParity,  verticalParity,  diagonalParityTopLeftToBottomRight,  diagonalParityBottomLeftToTopRight  };  }  function generateCodeWord(infoWord) {  const codeWord = [...infoWord];  parityBits.horizontalParity.forEach((el) => {  codeWord.push(el);  })  parityBits.verticalParity.forEach((el) => {  codeWord.push(el);  })  parityBits.diagonalParityTopLeftToBottomRight.forEach((el) => {  codeWord.push(el);  })  parityBits.diagonalParityBottomLeftToTopRight.forEach((el) => {  codeWord.push(el);  })  return codeWord;  }  function generateError(codeWord, errorCount) {  const erroneousCodeWord = [...codeWord];  errorCount = Math.min(errorCount, erroneousCodeWord.length);  for (let i = 0; i < errorCount; i++) {  let randomIndex;  do {  randomIndex = Math.floor(Math.random() \* erroneousCodeWord.length);  } while (erroneousCodeWord[randomIndex] === undefined);  erroneousCodeWord[randomIndex] = 1 - erroneousCodeWord[randomIndex];  }  return erroneousCodeWord;  }  function correctErrors(Y\_n, parityBits) {  const correctedCodeWord = [...Y\_n];  const parityPositions = {  diagonalParityBottomLeftToTopRight: Y\_n.length - parityBits.diagonalParityBottomLeftToTopRight.length,  diagonalParityTopLeftToBottomRight: Y\_n.length - parityBits.diagonalParityBottomLeftToTopRight.length - parityBits.diagonalParityTopLeftToBottomRight.length,  verticalParity: Y\_n.length - parityBits.diagonalParityBottomLeftToTopRight.length - parityBits.diagonalParityTopLeftToBottomRight.length - parityBits.verticalParity.length,  horizontalParity: Y\_n.length - parityBits.diagonalParityBottomLeftToTopRight.length - parityBits.diagonalParityTopLeftToBottomRight.length - parityBits.verticalParity.length - parityBits.horizontalParity.length  };    Object.keys(parityBits).forEach(parityType => {  const expectedParityBits = parityBits[parityType];  const receivedParityBits = correctedCodeWord.slice(parityPositions[parityType], parityPositions[parityType] + expectedParityBits.length);  console.log(`Expected parity bits ${parityType}, ${parityPositions[parityType] + expectedParityBits.length}`, expectedParityBits);  console.log("Received parity bits", receivedParityBits);  expectedParityBits.forEach((expectedBit, index) => {  if (expectedBit !== receivedParityBits[index]) {  correctedCodeWord[parityPositions[parityType] + index] = expectedBit;  }  });  });    return correctedCodeWord;  }  function analyzeCorrectionCapability(X\_n, Errors, generateError, correctErrors, N1) {  let N2 = 0; // Все найдены  let N3 = 0; // Все исправлены  for (let i = 0; i < N1; i++) {  const erroneousCodeWord = generateError(X\_n, Errors);  //console.log(X\_n, erroneousCodeWord);  const correctedCodeWord = correctErrors(erroneousCodeWord, parityBits);  //console.log(erroneousCodeWord);  //console.log(correctedCodeWord);  if (countErrors(erroneousCodeWord, X\_n) === Errors) {  N2++;  if (countErrors(correctedCodeWord, X\_n) === 0) {  N3++;  }  }  }  const ratioN2N1 = N2 / N1;  const ratioN3N1 = N3 / N1;  return { ratioN2N1, ratioN3N1 };  }  function countErrors(word1, word2) {  let errorCount = 0;  for (let i = 0; i < word1.length; i++) {  if (word1[i] !== word2[i]) {  errorCount++;  }  }  return errorCount;  } |

Листинг 1.1 – используемые функции

А также код выполнения самих заданий лабораторной работы, листинг 1.2.

|  |
| --- |
| const infoWord = [1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0];  const k1 = 2, k2 = 4, z = 2;  const dMin = k1 + k2 + z + 1;  const N1 = 5;  let matrix = insertInfoWordToMatrix(infoWord, k1, k2, z);  const parityBits = calculateParityBits(matrix);  console.log(`parity bits`, parityBits);  const codeWord = generateCodeWord(infoWord);  const errorCount = 3;  //const erroneousCodeWord = generateError(codeWord, errorCount);  //matrix.forEach(row => console.log(row.join(' ')));  //const correctedCodeWord = correctErrors(erroneousCodeWord, parityBits);  console.log(analyzeCorrectionCapability(codeWord, errorCount, generateError, correctErrors, N1)); |

Листинг 1.2 – выполнение задания лабораторной работы

Вывод представленного кода, рисунок 1.1.

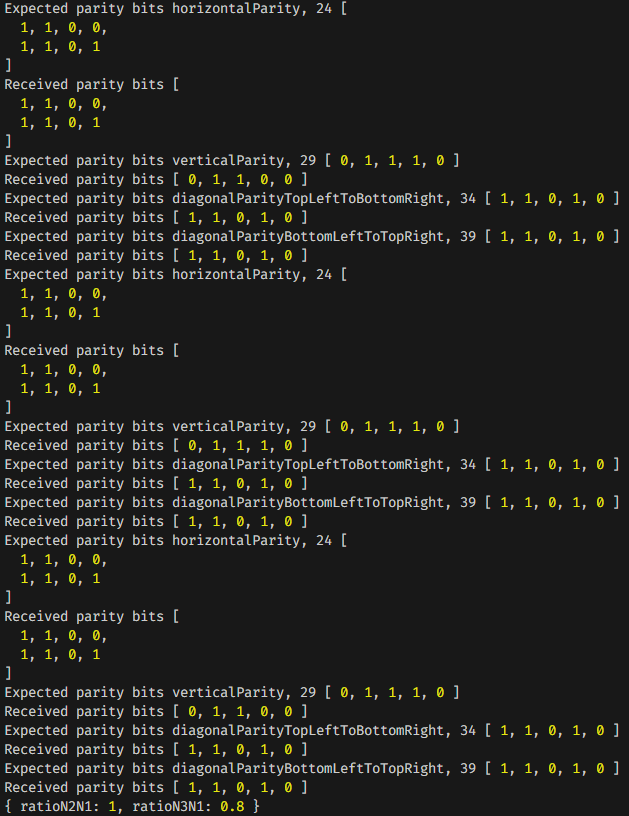


Рисунок 1.1

**Вывод:** в ходе выполнения лабораторной работы было выполнено вычисление битов паритетов, а также была реализована функция коррекции ошибок. Был проанализирован результат выполнения функции коррекции ошибок и построено отношения общего числа слов к правильно найденным ошибкам и общего числа слов к правильно скорректированным.