Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надёжность информационных систем»

**Лабораторная работа №6**

**Тема «ПЕРЕМЕЖЕНИЕ/ДЕПЕРЕМЕЖЕНИЕ ДАННЫХ В ИНФОРМАЦИОННО-ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ»**

Выполнил:

Студент 3 курса 7 группы ФИТ

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

Минск 2024

**Цель работы:** приобретение практических навыков использования методов перемежения/деперемежения двоичных данных в информационных системах.

**Задание на лабораторную работу**

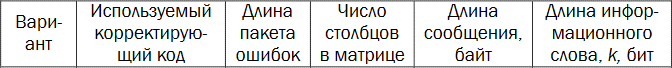
1. 1. Необходимо разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель. По желанию студент может использовать иной. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы.

2. За основу разрабатываемого приложения может быть взято приложение из выполненной лабораторной работы, соответствующей заданному корректирующему коду.

2. Местоположение заданной группы ошибок выбирается (генерируется) случайным образом. Необходимо для группы ошибок каждой длины сгенерировать 30−40 случайных ситуаций. После деперемежения и исправления ошибок в сообщении сравнить передаваемую последовательность и полученную после исправления ошибок. Проанализировать эффективность перемежения/ деперемежения.

3. Результаты оформить в виде отчета по установленным правилам.

Вариант 11





Довольно часто распределение ошибок носит взаимозависимый характер. В таких случаях говорят о группах(или пакетах) ошибок.

Существуют специальные коды, корректирующие пакетные ошибки, однако на практике чаще используют перемежение/деперемежение совместно с традиционными кодами.

Идея перемежения/деперемежения состоит в следующем. Если биты каждого кодового слова Хn передаются не в обычной последовательности, а через интервалы, превышающие ожидаемую длину пакета ошибок (в промежутки между битами одного слова вставляются биты других кодовых слов), то при возникновении такого типа ошибки обратная перемежению операция – деперемежение – разнесет («размажет») группу ошибок по всей совокупности кодовых слов, составляющих данное сообщение.

Длина пакета в нашем случае – это число рядом расположенных ошибочных битов.

Предложено много алгоритмов перемежения/деперемежения. Наиболее простыми являются блочные. При блочном перемежении входные биты делятся на блоки, которые последовательно записываются в строки некоторой таблицы.

Передаваемая последовательность делится на блоки по 5 битов. Каждый блок записывается в отдельную строку таблицы по порядку. Сообщение для передачи или хранения формируется при считывании символов из таблицы по столбцам. Деперемежение производится в обратной последовательности. Особенностью является неизменная позиция первого символа.

Глубина перемежения - разница между позициями одного и того же символа до и после перемежения.

В общем случае выбор глубины перемежения зависит от двух факторов. С одной стороны, чем больше расстояние между соседними символами, тем большей длины пакет ошибок может быть исправлен. С другой стороны, чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше задержка сигнала.

Для борьбы с длинными пакетами ошибок желательно увеличивать размеры таблицы. Однако это приводит к увеличению задержки в отправке и декодировании сообщения.

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

|  |
| --- |
| import \* as readline from "readline";  async function getUserInput(): Promise<string> {  const rl = readline.createInterface({  input: process.stdin,  output: process.stdout,  });  return new Promise((resolve) =>  rl.question("", (ans) => {  rl.close();  resolve(ans);  })  );  }  function RedBitsCount(k: number): number {  let r = Math.log(k) / Math.log(2) + 1.99;  return Math.floor(r);  }  function RandomArr2(mas: number[]): number[] {  for (let i = 0; i < mas.length; i++) {  mas[i] = Math.floor(Math.random() \* 2);  }  return mas;  }  function printMatrix(matrix: number[][], k: number, n: number): void {  for (let i = 0; i < k; i++) {  console.log(matrix[i].join(" "));  }  }  function printTransMatrix(matrix: number[][], k: number, n: number): void {  for (let j = 0; j < n; j++) {  let row = "";  for (let i = 0; i < k; i++) {  row += matrix[i][j] + " ";  }  console.log(row);  }  }  function checkMatrix(k: number): number[][] {  let r = RedBitsCount(k);  let n = r + k;  let rDouble = r - 1;  let rPow = Math.pow(2, rDouble);  let mas = new Array(n).fill(0).map(() => new Array(r).fill(0));  let combinations = new Array(rPow).fill(0).map(() => new Array(r).fill(0));  for (let i = 0; i < rPow; i++)  for (let j = 0; j < r; j++) combinations[i][j] = 0;  for (let segmentLength = 0; segmentLength < r - 2; segmentLength++) {  if (segmentLength \* r > k) break;  for (let i = 0; i < segmentLength + 2; i++) {  combinations[segmentLength \* r][i] = 1;  }  for (let segmentPosition = 1; segmentPosition < r; segmentPosition++) {  for (let i = 0; i < r - 1; i++) {  combinations[segmentLength \* r + segmentPosition][i + 1] =  combinations[segmentLength \* r + segmentPosition - 1][i];  }  combinations[segmentLength \* r + segmentPosition][0] =  combinations[segmentLength \* r + segmentPosition - 1][r - 1];  }  if (segmentLength == r - 3) {  for (let i = 0; i < r; i++) {  combinations[rPow - 1][i] = 1;  }  }  }  for (let i = 0; i < k; i++)  for (let j = 0; j < r; j++) mas[i][j] = combinations[i][j];  for (let i = 0; i < r; i++) mas[i + k][i] = 1;  return mas;  }  function addCheckBits(  masK: number[],  masN: number[],  checkMatrix: number[][]  ): number[] {  let lengthK = masK.length; // Should be equal to 2^n  let lengthN = masN.length;  let k = Math.sqrt(lengthK);  let r = RedBitsCount(k);  let n = k + r;  let matrix = new Array(k).fill(0).map(() => new Array(n).fill(0));  for (let i = 0; i < k; i++) {  let temp = new Array(n).fill(0);  for (let j = 0; j < k; j++) {  temp[j] = masK[k \* i + j];  }  sindrom(checkMatrix, temp, k);  for (let j = 0; j < n; j++) {  masN[i \* n + j] = temp[j];  }  }  return masN;  }  function interleaving(masN: number[], k: number): number[] {  let r = RedBitsCount(k);  let n = k + r;  let matrix = new Array(k).fill(0).map(() => new Array(n).fill(0));  for (let i = 0, m = 0; i < k; i++) {  for (let j = 0; j < n; j++, m++) {  matrix[i][j] = masN[m];  }  }  console.log("\n\nПолученая матрица");  printMatrix(matrix, k, n);  for (let i = 0, m = 0; i < n; i++) {  for (let j = 0; j < k; j++, m++) {  masN[m] = matrix[j][i];  }  }  return masN;  }  function reInterleaving(masN: number[], k: number): number[] {  let r = RedBitsCount(k);  let n = k + r;  let matrix = new Array(k).fill(0).map(() => new Array(n).fill(0));  for (let j = 0, m = 0; j < n; j++) {  for (let i = 0; i < k; i++, m++) {  matrix[i][j] = masN[m];  }  }  console.log("\n\nПолученая матрица");  printMatrix(matrix, k, n);  for (let j = 0, m = 0; j < k; j++) {  for (let i = 0; i < n; i++, m++) {  masN[m] = matrix[j][i];  }  }  return masN;  }  function searchErrorLong(  masN: number[],  checkMatrix: number[][],  k: number  ): number[] {  let r = RedBitsCount(k);  let n = r + k;  for (let i = 0; i < k; i++) {  let temp = new Array(n).fill(0);  for (let j = 0; j < n; j++) {  temp[j] = masN[n \* i + j];  }  searchError(temp, checkMatrix, k);  for (let j = 0; j < n; j++) {  masN[i \* n + j] = temp[j];  }  }  return masN;  }  function removeCheckBits(  masK: number[],  masN: number[],  checkMatrix: number[][]  ): number[] {  let lengthK = masK.length; // Should be equal to 2^n  let lengthN = masN.length;  let k = Math.sqrt(lengthK);  let r = RedBitsCount(k);  let n = k + r;  let matrix = new Array(k).fill(0).map(() => new Array(n).fill(0));  for (let i = 0; i < k; i++) {  let temp = new Array(n).fill(0);  for (let j = 0; j < n; j++) {  temp[j] = masN[n \* i + j];  }  for (let j = 0; j < k; j++) {  masK[i \* k + j] = temp[j];  }  }  return masK;  }  function sindrom(checkMatrix: number[][], mas: number[], k: number): number[] {  let r = RedBitsCount(k);  let n = r + k;  let sindrom = new Array(r).fill(0);  for (let i = 0, l = 0; i < r; i++, l = 0) {  for (let j = 0; j < k; j++) {  if (checkMatrix[j][i] == 1 && mas[j] == 1) l++;  else sindrom[i] = 0;  }  if (l % 2 == 1) sindrom[i] = 1;  else sindrom[i] = 0;  }  for (let i = 0; i < r; i++) {  mas[i + k] = sindrom[i];  }  return mas;  }  function searchError(  mas: number[],  checkMatrix: number[][],  k: number  ): number[] {  let r = RedBitsCount(k);  let n = r + k;  let beforeSindrom = new Array(r).fill(0);  for (let i = k; i < n; i++) {  beforeSindrom[i - k] = mas[i];  }  mas = sindrom(checkMatrix, mas, k);  for (let i = k, j = 0; i < n; i++) {  if (beforeSindrom[i - k] == mas[i]) {  mas[i] = 0;  j++;  if (j == r) {  for (let l = k; l < n; l++) {  mas[l] = beforeSindrom[l - k];  }  return mas;  }  } else {  mas[i] = 1;  }  }  for (let i = 0; i < n; i++) {  let l = 0;  for (let j = 0; j < r; j++) {  if (checkMatrix[i][j] == mas[j + k]) l++;  }  if (l == r) {  mas[i] = (mas[i] + 1) % 2;  }  }  mas = sindrom(checkMatrix, mas, k);  return mas;  } |

Листинг 1.1 – используемые функции

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

|  |
| --- |
| async function main() {  let lengthK = 16; //2^3<15<=2^4 => 2^4  let k = Math.sqrt(lengthK);  let r = RedBitsCount(k);  let n = k + r;  let lengthN = lengthK + r \* k;  let masK = new Array(lengthK).fill(0);  let masK2 = new Array(lengthK).fill(0);  let masN = new Array(lengthK + r \* k).fill(0);  let checkM = new Array(n).fill(0).map(() => new Array(r).fill(0));  let error: number;  let errorLength: number;  RandomArr2(masK);  console.log("Входная строка: ");  console.log(masK.join(" "));  console.log("\n\nПроверочная матрица: ");  checkM = checkMatrix(k);  printTransMatrix(checkM, n, r);  addCheckBits(masK, masN, checkM);  console.log("\n\nСтрока с доб. проверочными битами: ");  console.log(masN.join(" "));  interleaving(masN, k);  console.log("\nСтрока после перемежения: ");  console.log(masN.join(" "));  console.log("Где ошибка");  error = parseInt((await getUserInput()) || "0");  console.log("Длина ошибки");  errorLength = parseInt((await getUserInput()) || "0");  for (let i = error; i < error + errorLength; i++) {  masN[i] = (masN[i] + 1) % 2;  }  console.log("\nСтрока с ошибками: ");  console.log(masN.join(" "));  reInterleaving(masN, k);  console.log("\nСтрока после re:перемежения: ");  console.log(masN.join(" "));  searchErrorLong(masN, checkM, k);  console.log("\n\nСтрока после исправления ошибок: ");  console.log(masN.join(" "));  removeCheckBits(masK2, masN, checkM);  console.log("\n\nСтрока после удаления проверочных бит: ");  console.log(masK2.join(" "));  console.log("");  console.log("\nИсходная строка: ");  console.log(masK.join(" "));  }  main(); |

Листинг 1.2 – выполнение задания лабораторной работы

Вывод представленного кода, рисунок 1.1.

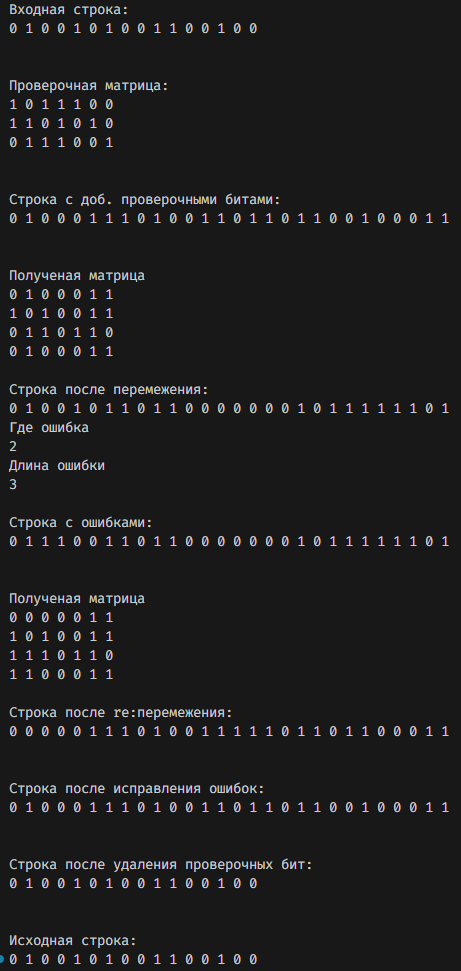


Рисунок 1.1

**Вывод:** в ходе выполнения лабораторной работы были изучены операции перемежения/деперемежения, с помощью которых можно исправлять пакеты ошибок, используя простые коды. Было изучено как зависит количество возможных исправлений от глубины перемежения.