Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №4 «Избыточное кодирование данных в информационных систем в информационных системах. Код Хемминга»

Студент: Водчиц Анастасия

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О. А.

Минск 2025

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

**Задачи**:

* Закрепить теоретические знания по использованию методов помехоустойчивого кодирования для повышения надежности передачи и хранения в памяти компьютера двоичных данных.
* Разработать приложение для кодирования/декодирования двоичной информации кодом Хемминга с минимальным кодовым расстоянием 3 или 4.
* Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

# **Теоретические сведения**

Надежность системы – характеристика способности программного, аппаратного, аппаратно-программного средства выполнить при определенных условиях требуемые функции в течение конкретного периода времени.

Достоверность работы системы (устройства) – свойство, характеризующее истинность конечного (выходного) результата работы (выполнения программы), определяемое способностью средств контроля фиксировать правильность или ошибочность работы.

Ошибка устройства – неправильное значение сигнала (бита – в цифровом устройстве) на внешних выходах устройства или отдельного его узла, вызванное технической неисправностью, или воздействующими на него помехами (преднамеренными либо непреднамеренными), или иным способом.

Ошибка программы – проявляется в не соответствующем реальному (требуемому) промежуточном или конечном значении(результате) вследствие неправильно запрограммированного алгоритма или неправильно составленной программы.

Надежность является комплексным свойством, включающим в себя единичные свойства: безотказность, ремонтопригодность, сохраняемость, долговечность.

Безотказность – это свойство технического объекта непрерывно сохранять работоспособное состояние в течение некоторого времени (или наработки).

Ремонтопригодность – это свойство технического объекта, заключающееся в приспособленности к поддержанию и восстановлению работоспособного состояния путем технического обслуживания, ремонта (или с помощью дополнительных, избыточных технических средств, функционирующих параллельно с объектом).

Изначальной причиной нарушения нормальной работы цифрового устройства являются технические дефекты (неисправности), возникающие внутри узлов или блоков устройства либо в каналах связи между ними.

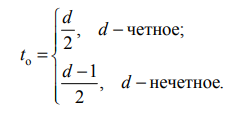
Дефекты или неисправности могут приводить либо к кратковременному нарушению достоверности работы устройства (сбой), либо к полной и окончательной потере достоверности (отказ).

В каждом из этих случаев следствием неисправности являются ошибки в информации (информационные ошибки). Чаще всего причиной ошибок бывают внешние помехи. Количество таких ошибок (количество ошибочных двоичных символов) принято называть кратностью ошибки. Обнаружение и/или исправление подобных ошибок как раз и призваны обеспечить кодер и декодер.

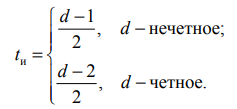
При использовании избыточных кодов исходные данные делятся на блоки из k битов (называются информационными битами). В процессе кодирования каждый k-битный блок данных преобразуется, как было отмечено выше, в блок из n битов (кодовое слово). Число k часто называется размерностью кода. Таким образом, к каждому блоку данных в процессе кодирования присоединяются r = n – k битов, которые называют избыточными битами (redundant bits), битами четности (parity bits) или контрольными битами (check bits); новой информации они не несут.

Для обозначения описанного кода обычно пользуются записью (n, k) и говорят, что данный код использует n символов для передачи (хранения) k символов сообщения. Отношение числа битов данных к общему числу битов k/n именуется степенью кодирования (code rate) – доля кода, которая приходится на полезную информацию. Еще одним важным параметром кода является расстояние Хемминга (d), которое показывает, что два кодовых слова различаются по крайней мере в d позициях.

В общем случае код позволяет обнаруживать ошибок:



Количество исправляемых кодом ошибок иt определяется следующим образом:



К. Шеннон сформулировал теорему для случая передачи дискретной информации по каналу связи с помехами, утверждающую, что вероятность ошибочного декодирования принимаемых сигналов может быть обеспечена сколь угодно малой путем выбора соответствующего способа кодирования сигналов. В теореме Шеннона не говорится о том, как нужно строить необходимые помехоустойчивые коды. Однако в ней указывается на принципиальную возможность кодирования, при котором может быть обеспечена сколь угодно высокая надежность передачи.

Код Хемминга относится к классу линейных блочных кодов. Линейные блочные коды – это класс кодов с контролем четности, которые можно описать парой чисел (n, k).

Для формирования r проверочных символов (кодирования), т. е. вычисления проверочного слова Xr, используется порождающая матрица G: совокупность базисных векторов будем далее записывать в виде матрицы G размерностью k×n с единичной подматрицей (I) в первых k строках и столбцах: G = [P|I].

Более точно матрица G называется порождающей матрицей линейного корректирующего кода в приведенно-ступенчатой форме. Кодовые слова являются линейными комбинациями строк матрицы G (кроме слова, состоящего из нулевых символов).

Кодирование заключается в умножении вектора сообщения Хk длиной k на порождающую матрицу по правилам матричного умножения (все операции выполняются по модулю 2). Очевидно, что при этом первые k символов кодового слова равны соответствующим символам сообщения, а последние r символов образуются как линейные комбинации первых.

Для всякой порождающей матрицы G существует матрица Н размерности r×n, задающая базис нулевого пространства кода и удовлетворяющая равенству . Справедливо также . Матрица Н, называемая проверочной, равна .

В коде Хемминга с минимальным кодовым расстоянием dmin = 3 проверочная матрица Н имеет классический вид и состоит из двух подматриц: P’ размером k×r и I размером r×r соответственно.

Общее число всех возможных комбинаций 2r должно удовлетворять неравенству .

Результат умножения сообщения на выходе канала передачи (Yn) или (что равнозначно) сообщения, считываемого из памяти, на проверочную матрицу (Н) называется синдромом (вектором ошибки) S: .

Синдром – это результат проверки четности, выполняемой над сообщением Yn для определения его принадлежности заданному набору кодовых слов. При положительном результате проверки синдром S равен 0, т. е. Yn = Хn. Если Yn содержит ошибки, которые можно исправить, то синдром имеет определенное ненулевое значение, что позволяет обнаружить и исправить конкретную ошибочную комбинацию.

Важно запомнить, что ненулевой синдром всегда равен сумме по модулю 2 тех векторстолбцов матрицы Н, номера которых соответствуют номерам ошибочных битов в слове Yn.

# **Практические задания**

Задание 1. На основе информационного сообщения, представленного символами русского/английского алфавитов, служебными символами и цифрами, содержащегося в некотором текстовом файле, сформировать информационное сообщение в двоичном виде; длина сообщения в бинарном виде должна быть не менее 16 символов. Для выполнения этого задания можно использовать коды ASCII символов алфавита либо результаты лабораторной работы № 3.

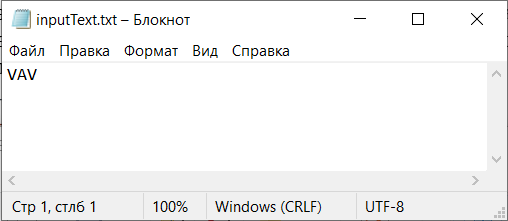


Рисунок 1.1 – Исходный файл

|  |
| --- |
| **class FileReader**  **{**  **public static string ReadTextFromFile(string filePath)**  **{**  **try**  **{**  **FileInfo fileInfo = new FileInfo(filePath);**  **return File.ReadAllText(filePath);**  **}**  **catch (Exception ex)**  **{**  **Console.WriteLine($"Ошибка при чтении файла: {ex.Message}");**  **return string.Empty;**  **}**  **}**  **public static int[] TextToBinaryArray(string text)**  **{**  **List<int> binaryArray = new List<int>();**  **foreach (char c in text)**  **{**  **int asciiValue = (int)c;**  **string binaryValue = Convert.ToString(asciiValue, 2).PadLeft(8, '0');**  **// Добавляем каждый бит как отдельный элемент массива**  **foreach (char bit in binaryValue)**  **{**  **binaryArray.Add(int.Parse(bit.ToString())); // Преобразуем '0'/'1' в 0/1**  **}**  **}**  **return binaryArray.ToArray();**  **}**  **}** |

Листинг 1.1 – Класс для работы с файлом

|  |
| --- |
| **string inputText = FileReader.ReadTextFromFile("inputText.txt");**  **int[] Xk = FileReader.TextToBinaryArray(inputText);**  **foreach (var x in Xk) Console.Write(x);**  **Console.WriteLine();**  **int k = Xk.Length;**  **Console.WriteLine("=========================================");** |

Листинг 1.2 – Обработка текстового файла

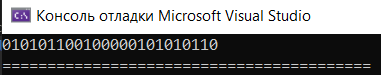


Рисунок 1.2 – Результат работы

Для примера рассмотрим следующие данные:

* Исходные данные (двоичные): Xk = 1011.
* Количество информационных бит: k = 4.

Задание 2. Для полученного информационного слова построить проверочную матрицу Хемминга.

|  |
| --- |
| **public static int[,]? GenerateMatrix(int k)**  **{**  **Console.WriteLine("Длина информационного слова: " + k);**  **int r = CalculateRedundantBits(k);**  **Console.WriteLine("Длина избыточного слова: " + r);**  **int n = k + r;**  **Console.WriteLine("Длина кодового слова: " + n);**  **int[,] iMatrix = CreateIMatrix(r);**  **int[,]? pMatrix = CreatePMatrix(k, r);**  **if (pMatrix == null) return null;**  **// Соединяем identityMatrix и pMatrix**  **int[,] Matrix = ConcatenateMatrices(pMatrix, iMatrix);**  **return Matrix;**  **}**  **private static int[,]? CreatePMatrix(int k, int r)**  **{**  **// Шаг 1: Посчитать общее количество возможных столбцов**  **int totalColumns = 0;**  **for (int p = 2; p <= r; p++)**  **{**  **totalColumns += Combinations(r, p);**  **}**  **Console.WriteLine($"Общее количество возможных столбцов: {totalColumns}");**  **// Шаг 2: Проверить, что количество столбцов больше k**  **if (totalColumns < k) // Corrected condition: <= changed to <**  **{**  **Console.WriteLine("Ошибка: количество возможных столбцов меньше k");**  **return null;**  **}**  **// Шаг 3: Построить матрицу**  **// Генерация всех возможных столбцов с весом от двух**  **var allColumns = GenerateAllColumns(r);**  **// Выбор k случайных уникальных столбцов**  **var selectedColumns = SelectRandomColumns(allColumns, k);**  **// Построение матрицы**  **int[,] matrix = new int[r, k];**  **Console.WriteLine($"Матрица P:");**  **for (int i = 0; i < r; i++)**  **{**  **for (int j = 0; j < k; j++)**  **{**  **matrix[i, j] = selectedColumns[j][i]; // Corrected indexing: selectedColumns[j][i]**  **Console.Write(matrix[i, j] + " ");**  **}**  **Console.WriteLine();**  **}**  **return matrix;**  **}**  **public static int[,] CreateIMatrix(int r)**  **{**  **if (r <= 0)**  **{**  **throw new ArgumentException("Размер матрицы должен быть положительным числом");**  **}**  **int[,] identityMatrix = new int[r, r];**  **Console.WriteLine($"Матрица I:");**  **for (int i = 0; i < r; i++)**  **{**  **for (int j = 0; j < r; j++)**  **{**  **if (i == j) identityMatrix[i, j] = 1;**  **else identityMatrix[i, j] = 0;**  **Console.Write(identityMatrix[i, j] + " ");**  **}**  **Console.WriteLine();**  **}**  **return identityMatrix;**  **}** |

Листинг 2.1 – Методы для вычисления проверочной матрицы

|  |
| --- |
| **int[,]? H = Hemming.GenerateMatrix(k);**  **Console.WriteLine("=========================================");**  **Console.WriteLine("Проверочная матрица H:");**  **Hemming.PrintMatrix(H);**  **Console.WriteLine("=========================================");** |

Листинг 2.2 – Вычисление проверочной матрицы

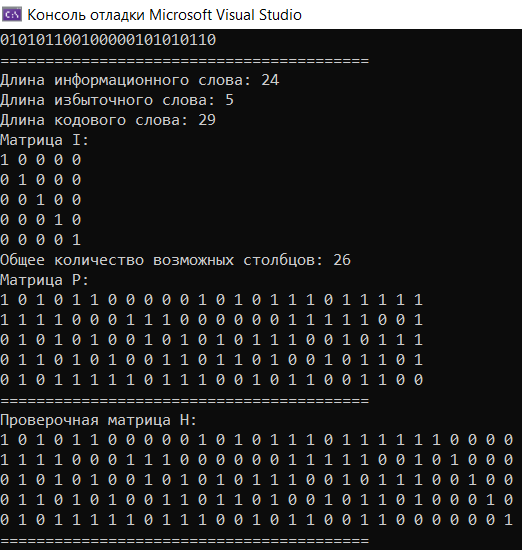


Рисунок 2.1 – Результат работы

Нам нужно достаточное количество проверочных бит (r), чтобы определить позицию одиночной ошибки во всем кодовом слове (n бит) или указать на отсутствие ошибки. Общее количество бит в кодовом слове равно n = k + r.

Условие: 2^r >= n + 1 (что равносильно 2^r >= k + r + 1).

Проверим значения для r:

Если r = 3: 2^3 >= 4 + 3 + 1 => 8 >= 8 (Истина)

Следовательно, нам нужно r = 3 проверочных бита. Общая длина кодового слова будет n = k + r = 4 + 3 = 7 бит.

Проверочная матрица имеет вид H = [P | Ir], где Ir — это единичная матрица размера r x r (в нашем случае 3 x 3), P — это матрица размера r x k (в нашем случае 3 x 4). Столбцы матрицы P должны быть выбраны из всех возможных r-битных (3-битных) векторов с весом (количеством единиц) не менее 2. Их должно быть ровно k=4.

Возможные 3-битные векторы с весом >= 2: [011], [101], [110], [111]. Их как раз 4.

Составим матрицу P, используя эти столбцы.

[ 0 1 1 1 ]

[ 1 0 1 1 ]

[ 1 1 0 1 ]

Единичная матрица Ir (размер 3 x 3):

[ 1 0 0 ]

[ 0 1 0 ]

[ 0 0 1 ]

Итоговая проверочная матрица H = [P | I]:

[ 0 1 1 1 | 1 0 0 ]

[ 1 0 1 1 | 0 1 0 ]

[ 1 1 0 1 | 0 0 1 ]

Задание 3. Используя построенную матрицу, вычислить избыточные символы (слово Xr).

|  |
| --- |
| **public static int[] SolveHamming(int[,] h, int[] knownBits, out int[] unknownBits)**  **{**  **int k = knownBits.Length;**  **int r = CalculateRedundantBits(k);**  **unknownBits = new int[r];**  **// Проверка размерностей**  **if (h.GetLength(0) != r || h.GetLength(1) != r + k || knownBits.Length != k)**  **{**  **throw new ArgumentException("Неверные размерности матриц или вектора известных битов.");**  **}**  **// Создаем полный вектор сообщения x, добавляя нули для неизвестных**  **int[] x = new int[r + k];**  **for (int i = 0; i < k; i++)**  **{**  **x[i] = knownBits[i];**  **}**  **// Остальные элементы (неизвестные) уже инициализированы нулями при создании массива**  **// Вычисляем неизвестные элементы (теперь работаем с полным вектором x)**  **for (int i = 0; i < r; i++)**  **{**  **int x\_ri = 0;**  **for (int j = 0; j < k; j++)**  **{**  **x\_ri ^= h[i, j] \* x[j];**  **}**  **unknownBits[i] = x\_ri; // Сохраняем неизвестный бит**  **x[k + i] = x\_ri;**  **}**  **return x;**  **}** |

Листинг 3.1 – Метод для вычисления избыточного слова матрицы

|  |
| --- |
| **int[] Xr;**  **int[] Xn = Hemming.SolveHamming(H, Xk, out Xr);**  **Console.WriteLine("Избыточное слово:");**  **foreach (var x in Xr) Console.Write(x);**  **Console.WriteLine();**  **Console.WriteLine("Кодовое слово:");**  **foreach (var x in Xn) Console.Write(x);**  **Console.WriteLine();** |

Листинг 3.2 – Вычисление избыточного слова матрицы

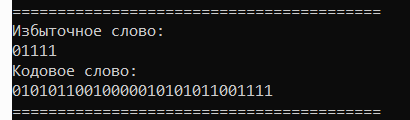


Рисунок 3.1 – Результат работы

Для систематического кода, где H = [P | I], кодовое слово Xn имеет структуру Xn = [Xk | Xr]. То есть, информационные биты идут первыми, за ними проверочные.

Основное уравнение линейного кода: H \* Xn^T = 0, где Xn^T — это кодовое слово как вектор-столбец.

[ 0 1 1 1 | 1 0 0 ] [ 1]

[ 1 0 1 1 | 0 1 0 ] \* [ 0] = 0

[ 1 1 0 1 | 0 0 1 ] [ 1]

[ 1]

[ xr1]

[ xr2]

[ xr3]

Получили Xr = [0 1 0].

Формируем полное кодовое слово Xn = [Xk | Xr] = [1 0 1 1 | 0 1 0]

Задание 4. Принять исходное слово со следующим числом ошибок: 0, 1, 2. Позиция ошибки определяется (генерируется) случайным образом.

|  |
| --- |
| **public static void ChangeValue(int[] array)**  **{**  **int d = random.Next(0, array.Length + 1);**  **if (array[d] == 0) array[d] = 1;**  **else array[d] = 0;**  **}** |

Листинг 4.1 – Метод для генерации случайной ошибки

|  |
| --- |
| **int[] Yn1 = new int [Xn.Length];**  **int[] Yn2 = new int [Xn.Length];**  **int[] Yn3 = new int [Xn.Length];**  **Xn.CopyTo(Yn1, 0);**  **Xn.CopyTo(Yn2, 0);**  **Xn.CopyTo(Yn3, 0);**  **Hemming.ChangeValue(Yn2);**  **Hemming.ChangeValue(Yn3);**  **Hemming.ChangeValue(Yn3);** |

Листинг 4.2 – Генерация слов для декодирования

Пусть при передаче произошла ошибка в 3-й позиции кодового слова Xn.

Переданное слово Xn = 1011010.

Принятое слово Yn = 1001010 (3-й бит инвертирован с 1 на 0).

Задание 5. Для полученного слова Yn = Yk, Yr, используя уже известную проверочную матрицу Хемминга, вновь вычислить избыточные символы (обозначим их Yr’).

Задание 6. Вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки Еn = е1, е2, …, еn и исправить одиночную ошибку; проанализировать ситуацию при возникновении ошибки в 2 битах.

|  |
| --- |
| **public static (int[] firstPart, int[] secondPart) SplitArray(int[] array, int k)**  **{**  **// Проверка входных данных**  **if (array == null)**  **{**  **throw new ArgumentNullException(nameof(array), "Входной массив не может быть null.");**  **}**  **if (k < 0 || k > array.Length)**  **{**  **throw new ArgumentOutOfRangeException(nameof(k), "Значение k должно быть неотрицательным и не больше длины массива.");**  **}**  **// Создаем первый подмассив**  **int[] firstPart = new int[k];**  **Array.Copy(array, 0, firstPart, 0, k);**  **// Создаем второй подмассив**  **int[] secondPart = new int[array.Length - k];**  **Array.Copy(array, k, secondPart, 0, array.Length - k);**  **return (firstPart, secondPart);**  **}**  **public static int[] XorVectors(int[] vector1, int[] vector2)**  **{**  **if (vector1 == null || vector2 == null)**  **{**  **throw new ArgumentNullException("Один или оба входных вектора равны null.");**  **}**  **if (vector1.Length != vector2.Length)**  **{**  **throw new ArgumentException("Векторы должны иметь одинаковую длину.");**  **}**  **int[] result = new int[vector1.Length];**  **for (int i = 0; i < vector1.Length; i++)**  **{**  **result[i] = vector1[i] ^ vector2[i];**  **}**  **return result;**  **}**  **public static void CheckYn(int[] Yn, int k, int[,] H)**  **{**  **Console.WriteLine("Поступило слово:");**  **foreach (var x in Yn) Console.Write(x);**  **Console.WriteLine();**  **(int[] Yk, int[] Yr) = SplitArray(Yn, k);**  **Console.WriteLine("Yk:");**  **foreach (var x in Yk) Console.Write(x);**  **Console.WriteLine();**  **Console.WriteLine("Yr:");**  **foreach (var x in Yr) Console.Write(x);**  **Console.WriteLine();**  **Console.WriteLine("Вычисляем Yr':");**  **int[] Yrq;**  **int[] Ynq = SolveHamming(H, Yk, out Yrq);**  **foreach (var x in Yrq) Console.Write(x);**  **Console.WriteLine();**  **int[] S = XorVectors(Yr, Yrq);**  **Console.WriteLine("Синдром:");**  **foreach (var x in S) Console.Write(x);**  **Console.WriteLine();**  **// Проверка на нулевой вектор**  **bool isZeroVector = true;**  **foreach (int element in S)**  **{**  **if (element != 0)**  **{**  **isZeroVector = false;**  **break;**  **}**  **}**  **if (isZeroVector)**  **{**  **Console.WriteLine("Синдром нулевой, значит ошибок нет");**  **return;**  **}**  **if (S.Length != H.GetLength(0))**  **{**  **throw new ArgumentException("Длина вектора должна быть равна количеству строк матрицы.");**  **}**  **// Поиск совпадающего столбца**  **for (int j = 0; j < H.GetLength(1); j++) // Перебираем столбцы**  **{**  **bool match = true;**  **for (int i = 0; i < H.GetLength(0); i++) // Перебираем строки**  **{**  **if (H[i, j] != S[i])**  **{**  **match = false;**  **break;**  **}**  **}**  **if (match)**  **{**  **Console.WriteLine($"Найден совпадающий столбец: {j}");**  **int[] E = new int[Yn.Length];**  **for (var e = 0; e < E.Length; e++) E[e] = 0;**  **E[j] = 1;**  **Console.WriteLine($"Вектор ошибки:");**  **for (var e = 0; e < E.Length; e++) Console.Write(E[e]);**  **Console.WriteLine();**  **Console.WriteLine($"Исправленный Yn:");**  **int[] YnFIX = XorVectors(Yn, E);**  **for (var i = 0; i < YnFIX.Length; i++) Console.Write(YnFIX[i]);**  **Console.WriteLine();**  **return;**  **}**  **}**  **Console.WriteLine("Совпадающих столбцов не найдено")**  **}** |

Листинг 6.1 – Методы для декодирования

|  |
| --- |
| **Console.WriteLine("=========================================");**  **Hemming.CheckYn(Yn1, k, H);**  **Console.WriteLine("=========================================");**  **Hemming.CheckYn(Yn2, k, H);**  **Console.WriteLine("=========================================");**  **Hemming.CheckYn(Yn3, k, H);**  **Console.WriteLine("=========================================");** |

Листинг 6.2 – Декодирование

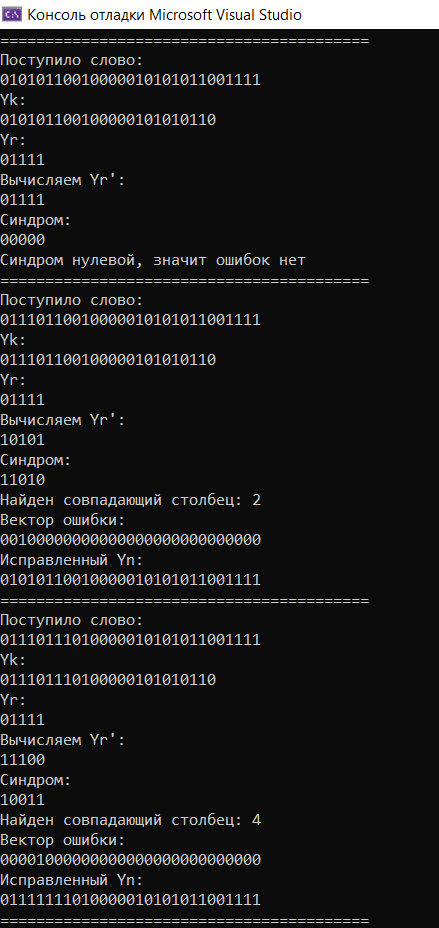


Рисунок 6.1 – Результат работы

Получаем слово Yn = 1001010.

Вычисляем синдром (S). Синдром вычисляется по формуле S = H \* Yn^T (mod 2).

Yn = [1 0 0 1 0 1 0], значит Yn = [1 0 0 1 0 1 0].

S = [1 1 0].

Синдром S не равен нулевому вектору [0 0 0], значит, в принятом слове Yn есть ошибка. При одиночной ошибке синдром S точно совпадает со столбцом матрицы H, номер которого соответствует позиции ошибки.

Сравниваем S = [1 1 0] со столбцами матрицы H. Видно, что S = [1 1 0] совпадает с третьим столбцом матрицы H. Это означает, что ошибка произошла в 3-й позиции слова Yn.

Нужно инвертировать бит на 3-й позиции в Yn. Yn = 1 0 0 1 0 1 0 Инвертируем 3-й бит (0 -> 1). Исправленное слово Yn' = 1 0 1 1 0 1 0.

**Вывод**: в ходе лабораторной работы были закреплены теоретические знания по основам теории информации, а также разработано программное средство для расчета энтропии по Шеннону и определения количества информации для латышского, таджикского и бинарного алфавитов.

Выполнены вычисления, позволяющие проанализировать объем информации в зависимости от используемого алфавита. Дополнительно исследовано влияние вероятности ошибок на передаваемую информацию. Установлено, что формула эффективной энтропии применима исключительно к бинарному алфавиту, так как только в двоичной системе представления информации возможно однозначное изменение символа на противоположный при наличии ошибки, что делает данный метод неприменимым для многосимвольных алфавитов.