Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Избыточное кодирование данных в информационных систем в информационных системах. Циклические коды

Студент: Водчиц Анастасия
ФИТ 3 курс 1 группа
Преподаватель: Нистюк О.А.

Минск 2025

Цель: приобретение практических навыков кодирования/декодирования двоичных данных при использовании циклических кодов (ЦК).

Задачи:

– Закрепить теоретические знания по алгебраическому описанию и использованию ЦК для повышения надежности передачи и хранения в памяти компьютера двоичных данных, для контроля интегральности файлов информации.

– Разработать приложение для кодирования/декодирования двоичной информации циклическим кодом.

– Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

Теоретические сведения

Циклические коды − это семейство помехоустойчивых кодов, одной из разновидностей которых являются коды Хемминга. Основные свойства ЦК: • относятся к классу линейных, систематических;

– сумма по модулю 2 двух разрешенных кодовых комбинаций дает также разрешенную кодовую комбинацию;

– каждый вектор (кодовое слово), получаемый из исходного кодового вектора путем циклической перестановки его символов, также является разрешенным кодовым вектором; к примеру, если кодовое слово имеет следующий вид: 1101100, то разрешенной кодовой комбинацией будет и такая: 0110110;

– при простейшей циклической перестановке символы кодового слова перемещаются слева направо на одну позицию, как в приведенном примере;

– поскольку к числу разрешенных кодовых комбинаций ЦК относится нулевая комбинация 000...00, то минимальное кодовое расстояние dmin для ЦК определяется минимальным весом разрешенной кодовой комбинации;

– циклический код не обнаруживает только такие искаженные помехами кодовые комбинации, которые приводят к появлению на стороне приема других разрешенных комбинаций этого кода;

– в основе описания и использования ЦК лежит полином или многочлен некоторой переменной (обычно X).

Порождающие полиномы циклических кодов. Характеризуя ЦК в общем случае, обычно отмечают следующее: ЦК составляют множество многочленов {Bj(X)} степени r (r − число проверочных символов в кодовом слове), кратных порождающему (образующему) полиному G(X) степени r, который должен быть делителем бинома Xn + 1, т. е. остаток после деления бинома на G(X) должен быть нулевым.

Формирование разрешенных кодовых комбинаций ЦК Bj(X) основано на предварительном выборе порождающего (генераторного или образующего)

полинома G(X), который обладает важным отличительным признаком: все комбинации Bj(X) делятся на порождающий полином G(X) без остатка.

Степень порождающего полинома определяет число проверочных символов: r = n – k. Из этого свойства следует простой способ формирования разрешенных кодовых слов ЦК − умножение информационного слова A(X) на порождающий полином G(X): Bj(X) = Aj(X) · G(X)

Порождающими могут быть только такие полиномы, которые являются делителями двучлена (бинома) X^z + 1: (X^z + 1) / G(X) = H(X) при нулевом остатке: R(X) = 0.

Таким образом, в основе построения ЦК лежит операция деления передаваемой кодовой комбинации на порождающий неприводимый полином степени r в соответствии с выражением (6.2). Остаток R(X) от деления используется при формировании проверочных разрядов. При декодировании принятой n-разрядной кодовой комбинации (Yn) опять производится ее деление на порождающий (производящий, образующий) полином.

Кодирование информационного слова. Деление полиномов позволяет представить кодовые слова в виде блочного кода, т. е. информационных Xk (Ai(X)) и проверочных Xr (Ri(X)) символов. Поскольку число последних равно r, то для компактной их записи в младшие разряды кодового слова надо предварительно к кодируемому (информационному) слову Ai(X) справа дописать r нулевых символов.

Декодирование принятого сообщения по синдрому. Основная операция: принятое кодовое слово (Yn) нужно поделить на порождающий полином, который использовался при кодировании.

Декодирование синдрома и исправление ошибки в принятом сообщении. Декодирование ненулевого синдрома имеет целью определение ошибочного бита в принятом сообщении или, иначе говоря, определение вектора En.

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

Практические задания

Разработать собственное приложение, которое позволяет выполнять следующие операции:

Задание 1. Выбирается порождающий полином ЦК, а по значению соответствующего ему значения r – длина k информационного слова Xk. Полагаем, что каждый полином соответствует коду, обнаруживающему и исправляющему одиночные ошибки в кодовых словах. Определить параметры (n, k)-кода для своего варианта. Основой задания является разработка приложения.

Вариант	r	Полином
3	5	x^5+x^2+1

n	k	r	Полином	d_{min}
7	4	3	$x^3 + x + 1$	3
15	11	4	$x^4 + x + 1$	3
15	7	8	$x^8 + x^7 + x^6 + 1$	5
15	5	10	$x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$	7
31	26	5	$x^5 + x^2 + 1$	3
31	21	10	$x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$	5
31	16	15	$x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$	7
31	11	20	$x^{20} + x^{18} + x^{17} + x^{13} + x^{10} + x^9 + x^7 + x^6 + x^4 + + x^2 + 1$	11
31	6	25	$x^{25} + x^{24} + x^{21} + x^{19} + x^{18} + x^{16} + x^{15} + x^{14} + + x^{13} + x^{11} + x^9 + x^5 + x^2 + x + 1$	15
63	57	6	$x^6 + x + 1$	3
63	51	12	$x^{12} + x^{10} + x^8 + x^5 + x^4 + x^3 + 1$	5
63	45	18	$x^{18} + x^{17} + x^{16} + x^{15} + x^9 + x^7 + x^6 + x^3 + x^2 + + x + 1$	7
63	39	24	$x^{24} + x^{23} + x^{22} + x^{20} + x^{19} + x^{17} + x^{16} + x^{13} + + x^{10} + x^9 + x^8 + x^5 + x^4 + x^2 + x + 1$	9

Рисунок 1.1 – Параметры некоторых циклических кодов

Возьмём для примера Xk = 1011, как мы видим из матрицы, для k = 4 берём n = 7, r = 3.

Порождающий полином G(x) = x^3+x+1~1011

Задание 2. Составить порождающую матрицу (n, k)-кода в соответствии с формулой (6.7), трансформировать ее в каноническую форму и далее – в проверочную матрицу канонической формы.

Строим порождающую матрицу:

[1, 0, 1, 1 | 0, 0, 0] + 3 + 4

[0, 1, 0, 1 | 1, 0, 0] + 4

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

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

Приводим к каноническому виду:

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

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

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

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

Проверочная матрица канонической формы:

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

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

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

```
static int[,] GenerateGeneratorMatrixNonCanonical(int k, int n, int[] g)
{
    int r = g.Length - 1;
    if (n != k + r) throw new ArgumentException("Несоответствие размеров n,
k, r");
    int[,] G_noncanonical = new int[k, n];
    for (int i = 0; i < k; i++)
    {
        for (int gi = 0; gi < g.Length; gi++)
        {
            int colIndex = i + gi;
            if (colIndex < n)
            {
                G_noncanonical[i, colIndex] = g[gi];
            }
        }
    }
    return G_noncanonical;
}
```

Листинг 2.1 – Создание порождающей матрицы (n, k)-кода

```
static void ToCanonicalForm(int[,] G, int k, int n)
{
    Console.WriteLine(STEP_PREFIX + "Выполнение приведения к каноническому
виду [Ik | P]...");
    bool possible = true;
    Dictionary<int, List<int>> rowModifications = new Dictionary<int,
List<int>>();

    for (int i = 0; i < k; i++)
    {
        int pivotRow = i;
        while (pivotRow < k && G[pivotRow, i] == 0) pivotRow++;

        if (pivotRow == k)
        {
            Console.WriteLine($"{DEBUG_PREFIX}{WARN_STATUS} Столбец {i}: Не
найден опорный '1'. Пропуск.");
            possible = false;
            continue;
        }

        if (pivotRow != i)
        {
            Console.WriteLine($"{DEBUG_PREFIX}Обмен:    Строка {i} <-> Строка
```

```
{pivotRow}");
            List<int>   history_i   =   rowModifications.ContainsKey(i)   ?
rowModifications[i] : null;
            List<int>                        history_pivot                        =
rowModifications.ContainsKey(pivotRow) ? rowModifications[pivotRow] : null;
            if (history_i != null) rowModifications[pivotRow] = history_i;
else rowModifications.Remove(pivotRow);
            if (history_pivot != null) rowModifications[i] = history_pivot;
else rowModifications.Remove(i);
            for (int t = 0; t < n; t++) { int temp = G[i, t]; G[i, t] =
G[pivotRow, t]; G[pivotRow, t] = temp; }
        }

        for (int j = i + 1; j < k; j++)
        {
            if (G[j, i] == 1)
            {
                if (!rowModifications.ContainsKey(j)) rowModifications[j] =
new List<int>();
                rowModifications[j].Add(i);
                for (int t = i; t < n; t++) G[j, t] ^= G[i, t];
            }
        }
    }

    if (!possible)
    {
        Console.WriteLine($"{DEBUG_PREFIX}{ERROR_STATUS}   Прямой   ход   не
завершен из-за отсутствия опорных элементов.");
    }

    for (int i = k - 1; i >= 0; i--)
    {
        if (G[i, i] == 0)
        {
            if   (possible)   Console.WriteLine($"{DEBUG_PREFIX}{WARN_STATUS}
Столбец {i}: Опорный элемент G[{i},{i}] == 0. Пропуск.");
            continue;
        }

        for (int j = 0; j < i; j++)
        {
            if (G[j, i] == 1)
            {
                if (!rowModifications.ContainsKey(j)) rowModifications[j] =
new List<int>();
                rowModifications[j].Add(i);
                for (int t = i; t < n; t++) G[j, t] ^= G[i, t];
            }
        }
    }

    Console.WriteLine(DEBUG_PREFIX   +   "Итоговые   операции   сложения   для
строк:");
    bool modificationsFound = false;
    foreach (int targetRow in rowModifications.Keys.OrderBy(key => key))
    {
        List<int> sourceRows = rowModifications[targetRow];
        if (sourceRows != null && sourceRows.Count > 0)
        {
            sourceRows.Sort();
            Console.WriteLine($"{DEBUG_PREFIX}Строка    {targetRow,-2}    +=
Строка {string.Join(" + Строка ", sourceRows)}");
            modificationsFound = true;
```

```
        }
    }
    if (!modificationsFound)
    {
        Console.WriteLine(DEBUG_PREFIX + "(Операций сложения строк не зафиксировано)");
    }

    bool diagonalOk = true;
    for (int idx = 0; idx < k; ++idx) if (G[idx, idx] == 0) diagonalOk = false;

    if (possible && diagonalOk)
        Console.WriteLine(STEP_PREFIX + "Приведение к канонической форме завершено.");
    else
        Console.WriteLine($"{ERROR_STATUS} Приведение к канонической форме не удалось полностью завершить.");
}
```

Листинг 2.2 – Приведение матрицы в каноническую форму

```
static int[,] GenerateHMatrix(int[,] G_canonical, int k, int n)
{
    int r = n - k;
    int[,] H = new int[r, n];
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < k; j++)
            H[i, j] = G_canonical[j, k + i];
        H[i, k + i] = 1;
    }
    return H;
}
```

Листинг 2.3 – Создание проверочной матрицы

Задание 3.# Используя порождающую матрицу ЦК, вычислить избыточные символы (слово Xr) кодового слова Xn и сформировать это кодовое слово.

Избыточные биты вычисляем как Xk/G(x). x^6+x^4+x^3/x^3+x+1=x^3 с остатком 0. Остаток и есть избыточные биты.

Получаем Xn = 1011000

```
static int[] MultiplyVectorByGeneratorMatrix(int[] vector, int[,] G_canonical)
{
    int k = G_canonical.GetLength(0);
    int n = G_canonical.GetLength(1);
    if (vector.Length != k) throw new ArgumentException("Длина вектора должна быть равна k");
    int[] result = new int[n];
    for (int i = 0; i < k; i++)
        if (vector[i] == 1)
            for (int j = 0; j < n; j++)
                result[j] ^= G_canonical[i, j];
    return result;
}
```

Листинг 3.1 – Вычисление кодового слова

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

Примем:

Yn1=1011000

Yn2 = 0011000

Yn3 = 0011000

```
static int[] IntroduceErrors(int[] codeword, int errorCount, Random rand, out
List<int> flippedPositions)
{
    int n = codeword.Length;
    int[] corrupted = (int[])codeword.Clone();
    flippedPositions = new List<int>();
    if (errorCount <= 0 || errorCount > n) return corrupted;
    while (flippedPositions.Count < errorCount)
    {
        int pos = rand.Next(n);
        if (!flippedPositions.Contains(pos))
        {
            corrupted[pos] ^= 1;
            flippedPositions.Add(pos);
        }
    }
    flippedPositions.Sort();
    return corrupted;
}
```

Листинг 4.1 – Функция для генерации ошибки

Задание 5. Для полученного слова Yn вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки En = e1, e2, …, en и исправить одиночную ошибку, используя выражение (6.5); проанализировать ситуацию при возникновении ошибки в 2 битах.

Синдром определяется как остаток от деления Yn на G(x).

Для Yn1=1011000:

x^6+x^4+x^3/x^3+x+1=x^3 с остатком 0. Синдром 0, значит ошибок нет.

Для Yn2 = 0011000:

x^4+x^3/x^3+x+1=x с остатком x^2+1~101. Синдром 101, что соответствует 1 столбцу в проверочной матрице, значит ошибка в 1 бите. En=1000000.

Для Yn3 = 0011000:

x^4/x^3+x+1=x с остатком x^2+x~110. Ошибка в 1 и 4 битах, сложив 1 и 4 столбца проверочной матрицы получаем 110, что соответствует синдрому.

```
static int[] ComputeSyndrome(int[,] H, int[] received)
{
    int r = H.GetLength(0);
    int n = H.GetLength(1);
    if (received.Length != n) throw new ArgumentException("Длина принятого
слова должна быть равна n");
    int[] syndrome = new int[r];
    for (int i = 0; i < r; i++)
    {
        int sum = 0;
        for (int j = 0; j < n; j++)
            sum ^= H[i, j] * received[j];
        syndrome[i] = sum;
    }
    return syndrome;
}
```

Листинг 5.1 – Функция для вычисления синдрома

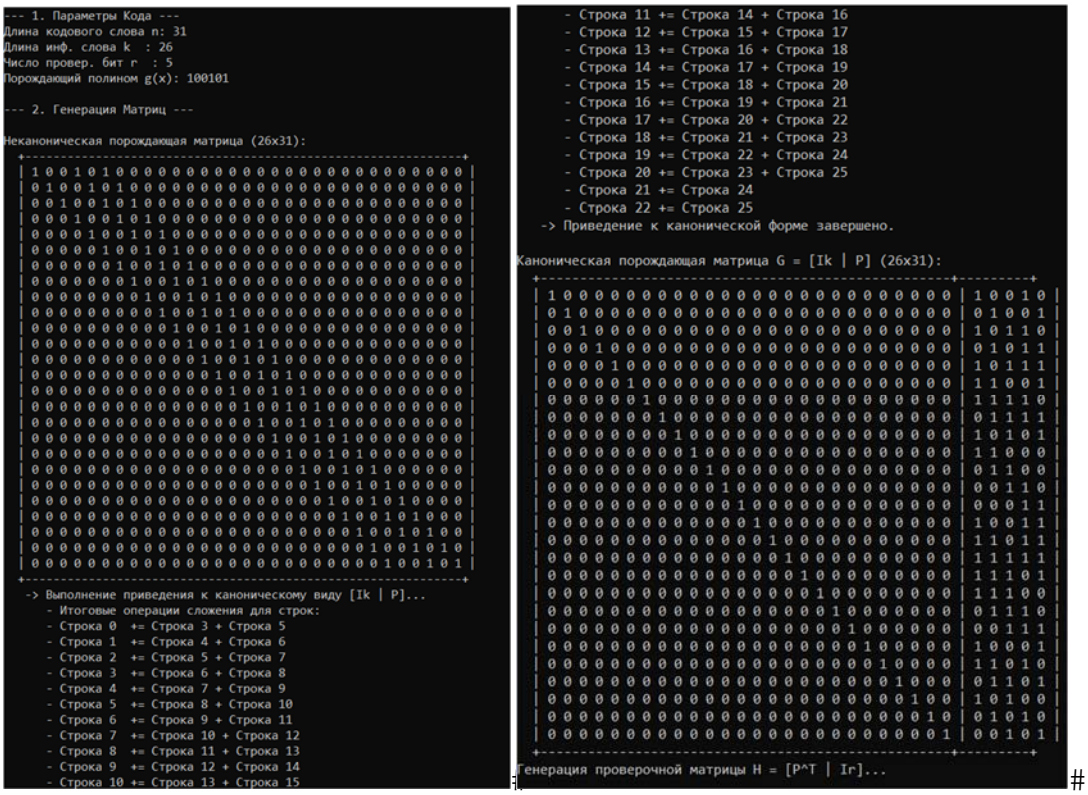
```
static int FindErrorPosition(int[] syndrome, int[,] H)
{
    int r = H.GetLength(0);
    int n = H.GetLength(1);
    if (syndrome.All(bit => bit == 0)) return -1;
    for (int j = 0; j < n; j++)
    {
        bool match = true;
        for (int i = 0; i < r; i++)
            if (H[i, j] != syndrome[i]) { match = false; break; }
        if (match) return j;
    }
    return -2;
}
```

Листинг 5.2 – Функция для анализа синдрома

```
static int[] CorrectSingleError(int[] received, int errorPosition)
{
    int[] corrected = (int[])received.Clone();
    if (errorPosition >= 0 && errorPosition < corrected.Length)
        corrected[errorPosition] ^= 1;
    return corrected;
}
```

Листинг 5.3 – Функция для исправления ошибок

Задание 6. Результаты оформить в виде отчета по установленным правилам.



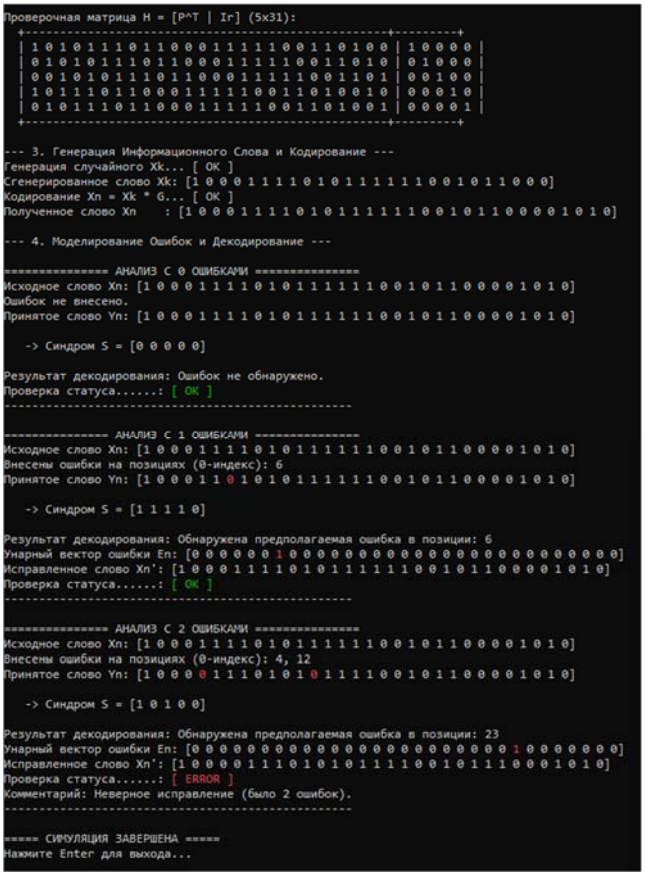


Рисунок 6.1 – Результат работы

Вывод: В ходе выполнения данной лабораторной работы были изучены и практически реализованы основные этапы работы с циклическими кодами

на примере (n=31, k=26)-кода, заданного порождающим полиномом g(x) = { 100101}.