Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Избыточное кодирование данных в информационных систем в информационных системах. Итеративные 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 итеративных кодов.

Задачи:

– Закрепить теоретические знания по использованию итеративных кодов для повышения надежности передачи и хранения в памяти компьютера двоичных данных.

– Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.

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

Теоретические сведения

Итеративные коды относятся к классу кодов произведения.

Кодом произведения двух исходных (базовых) помехоустойчивых кодов называется такой многомерный помехоустойчивый код, кодовыми последовательностями которого являются все двумерные таблицы со строками кода (k1) и столбцами кода (k2).

Итеративные коды могут строиться на основе использования дву-, трехмерных матриц (таблиц) и более высоких размерностей. Каждая из отдельных последовательностей информационных символов кодируется определенным линейным кодом (групповым или циклическим). Получаемый таким образом итеративный код также является линейным.

Простейшим из итеративных кодов является двумерный код с проверкой на четность по строкам и столбцам. Итеративные коды, иногда называемые прямоугольными кодами (англ. rectangular code) либо композиционными (англ. product code), являются одними из самых простых (с точки зрения аппаратной реализации) избыточных кодов, позволяющих исправлять ошибки в информационных словах.

Основное достоинство рассматриваемых кодов – простота как аппаратной, так и программной реализации.

Основной недостаток – сравнительно высокая избыточность. В упомянутой двумерной матрице кодовые слова записываются в виде таблицы. Проверочные символы вычисляются исходя из того, что строки и столбцы должны содержать четное (нечетное) число единиц.

Практические задания

Разработать собственное приложение, которое позволяет выполнять следующие операции:

Задание 1. вписывать произвольное двоичное представление информационного слова Xk (кодируемой информации) длиной k битов в двумерную матрицу размерностью в соответствии с вариантом либо в трехмерную матрицу в соответствии с вариантом.

Вариант	Длина информационного слова	k1	k2	z	Количество групп паритетов
3	24	4 3 3 6	6 8 3 2	- - 4 2	2;3 2;3 2;3;4;5 2;3;4;5

К примеру возьмём Xk: [10111011]

Строим матрицу из Xk:

[1, 0, 1, 1]

[1, 0, 1, 1]

```
// 1. Генерация информационного слова
int[] infoWord = GenerateRandomBinaryWord(coder.K);
PrintUtils.PrintVector(infoWord, "Xk");
```

Листинг 1.1 – Генерация информацинного слова

```
private void FillMatrix(int[] data)
{
    int index = 0;
    if (_matrix2D != null)
    {
        for (int i = 0; i < K1; i++)
        {
            for (int j = 0; j < K2; j++)
            {
                _matrix2D[i, j] = data[index++];
            }
        }
    }
    else // 3D
    {
        for (int k = 0; k < Z.Value; k++) // Layer (z)
        {
            for (int i = 0; i < K1; i++)  // Row (k1)
            {
                for (int j = 0; j < K2; j++) // Column (k2)
                {
                    _matrix3D[i, j, k] = data[index++];
                }
```

```
            }
        }
    }
}
```

Листинг 1.2 – Заполнение матрицы

Задание 2. вычислять проверочные биты (биты паритетов): а) по двум; б) по трем; в) по четырем направлениям (группам паритетов);

Группа паритетов 1: [11] – по строкам

Группа паритетов 2: [0000] – по столбцам

```
private int CalculateParity(IEnumerable<int> bits) // Вспомогательный метод
для вычисления бита четности (по модулю 2)
{
    return bits.Aggregate(0, (acc, bit) => acc ^ bit); // XOR sum
}

private void CalculateParityBits()
{
    if (_matrix2D != null) // 2D Case
    {
        // Group 1: проверка по строкам
        _parityGroup1 = new int[K1];
        for (int i = 0; i < K1; i++)
        {
            int[] row = Enumerable.Range(0, K2).Select(j => _matrix2D[i,
j]).ToArray();
            _parityGroup1[i] = CalculateParity(row);
        }

        // Group 2: проверка по столбцам
        _parityGroup2 = new int[K2];
        for (int j = 0; j < K2; j++)
        {
            int[] col = Enumerable.Range(0, K1).Select(i => _matrix2D[i,
j]).ToArray();
            _parityGroup2[j] = CalculateParity(col);
        }
        _parityGroup3 = null; // не используются
        _parityGroup4 = null; // не используются
    }
    else // 3D Case
    {
        // Group 1: Parity along k1 (fixing k2, z)
        _parityGroup1 = new int[K2 * Z.Value];
        int p1Idx = 0;
        for (int k = 0; k < Z.Value; k++)
        {
            for (int j = 0; j < K2; j++)
            {
                _parityGroup1[p1Idx++] = CalculateParity(Enumerable.Range(0,
K1).Select(i => _matrix3D[i, j, k]));
            }
        }

        // Group 2: Parity along k2 (fixing k1, z)
        _parityGroup2 = new int[K1 * Z.Value];
        int p2Idx = 0;
```

```
        for (int k = 0; k < Z.Value; k++)
        {
            for (int i = 0; i < K1; i++)
            {
                _parityGroup2[p2Idx++] = CalculateParity(Enumerable.Range(0,
K2).Select(j => _matrix3D[i, j, k]));
            }
        }

        // Group 3: Parity along z (fixing k1, k2)
        _parityGroup3 = new int[K1 * K2];
        int p3Idx = 0;
        for (int i = 0; i < K1; i++)
        {
            for (int j = 0; j < K2; j++)
            {
                _parityGroup3[p3Idx++] = CalculateParity(Enumerable.Range(0,
Z.Value).Select(k => _matrix3D[i, j, k]));
            }
        }

        // Group 4: Overall parity of information bits
        _parityGroup4 = new int[1];
        _parityGroup4[0] = CalculateParity(_informationWord);
    }
}
```

Листинг 2.1 – Вычисление паритетов

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

Xn: [10111011110000]

```
private void FormCodewordXn() // Формирует полное кодовое слово
{
    var codewordList = new List<int>(_informationWord);
    if (_parityGroup1 != null) codewordList.AddRange(_parityGroup1);
    if (_parityGroup2 != null) codewordList.AddRange(_parityGroup2);
    if (_parityGroup3 != null) codewordList.AddRange(_parityGroup3);
    if (_parityGroup4 != null) codewordList.AddRange(_parityGroup4);
    _codewordXn = codewordList.ToArray();

    // Сравнивает фактическую длину _codewordXn.Length с ожидаемой N
    if (_codewordXn.Length != N)
    {
        Console.WriteLine($"Warning:     Calculated     codeword     length
({_codewordXn.Length}) doesn't match expected N ({N}). R={R}");
        N = _codewordXn.Length;
    }
}
```

Листинг 3.1 – Вычисление кодового слова

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

Генерация одной ошибки Yn: [00111011110000] на позиции 1.

Строим матрицу из Yn:

[0, 0, 1, 1]

[1, 0, 1, 1]

Группа паритетов 1: [01] – по строкам

Группа паритетов 2: [0001] – по столбцам

```
public int[] IntroduceErrors(int errorCount) // Вносит заданное количество
(errorCount) случайных ошибок в _codewordXn
{
    if (_codewordXn == null)
    {
        throw new InvalidOperationException("Encoding must be performed
before introducing errors.");
    }
    if (errorCount < 0) errorCount = 0;
    if (errorCount > N) errorCount = N;


    _receivedWordYn = (int[])_codewordXn.Clone();

    if (errorCount == 0) return (int[])_receivedWordYn.Clone();


    var indicesToFlip = new HashSet<int>();
    while (indicesToFlip.Count < errorCount)
    {
        indicesToFlip.Add(_random.Next(N));
    }

    foreach (int index in indicesToFlip)
    {
        _receivedWordYn[index] = 1 - _receivedWordYn[index];
    }

    return (int[])_receivedWordYn.Clone();
}
```

Листинг 4.1 – Функция для генерации ошибки

Задание 5. определять местоположение ошибочных символов итеративным кодом в слове Yn в соответствии с используемыми группами паритетов по пункту (2) и исправлять ошибочные символы (результат исправления – слово Yn');

Синдром определяется как сумма по модулю 2 между полученными и вычисленными паритетами.

Полученные: 110000, вычисленные: 010001

Синдром равен 100001, что соответствует ошибке в 1 строке 1 столбце матрицы.

```
public int[] Decode()
```

```
{
    if (_receivedWordYn == null)
    {
        throw new InvalidOperationException("Errors must be introduced (or Yn
set) before decoding.");
    }

    _correctedWordYnPrime = (int[])_receivedWordYn.Clone();

    int[,] currentMatrix2D = null;
    int[,,] currentMatrix3D = null;

    // Извлечение данных
    int[] currentData = _correctedWordYnPrime.Take(K).ToArray(); // Это Yk
(возможно, с ошибками)
    int[]                                receivedP1                                =
_correctedWordYnPrime.Skip(K).Take(_parityGroup1?.Length ?? 0).ToArray(); //
Это Yr (Received) - Группа 1
    int[]       receivedP2       =       _correctedWordYnPrime.Skip(K       +
(_parityGroup1?.Length ?? 0)).Take(_parityGroup2?.Length ?? 0).ToArray(); //
Yr (Received) - Группа 2
    int[]       receivedP3       =       _correctedWordYnPrime.Skip(K       +
(_parityGroup1?.Length     ??     0)     +     (_parityGroup2?.Length     ??
0)).Take(_parityGroup3?.Length ?? 0).ToArray(); // Yr (Received) - Группа 3
    int[]       receivedP4       =       _correctedWordYnPrime.Skip(K       +
(_parityGroup1?.Length   ??   0)   +   (_parityGroup2?.Length   ??   0)   +
(_parityGroup3?.Length ?? 0)).Take(_parityGroup4?.Length ?? 0).ToArray(); //
Yr (Received) - Группа 4

    // Собираем полное "Yr (Received)" для вывода
    var fullReceivedYrList = new List<int>();
    if (receivedP1 != null) fullReceivedYrList.AddRange(receivedP1);
    if (receivedP2 != null) fullReceivedYrList.AddRange(receivedP2);
    if (receivedP3 != null) fullReceivedYrList.AddRange(receivedP3);
    if (receivedP4 != null) fullReceivedYrList.AddRange(receivedP4);

    // Инициализация рабочей матрицы НЕИСПРАВЛЕННЫМИ данными currentData (Yk)
    if (!Z.HasValue)
    {
        currentMatrix2D = new int[K1, K2];
        FillMatrix2DFromData(currentData, currentMatrix2D);
    }
    else
    {
        currentMatrix3D = new int[K1, K2, Z.Value];
        FillMatrix3DFromData(currentData, currentMatrix3D);
    }

    // Вычисление и вывод Yr по НЕИСПРАВЛЕННЫМ данным (Yk) ДО начала итераций
    Console.WriteLine("\n------");
    PrintUtils.PrintVector(fullReceivedYrList, "Yr "); // Выводим полученное
Yr

    // Вычисляем, каким ДОЛЖЕН быть Yr, если бы данные были currentData (Yk)
    var calculatedYrFromYkList = new List<int>();
    // Используем те же методы, что и в CalculateParityBits, но на текущей
матрице
    if (!Z.HasValue) // 2D
    {
        if (NumParityGroups >= 1)
        {
            var tempP1 = new int[K1];
            for    (int    i    =    0;    i    <    K1;    i++)    tempP1[i]    =
CalculateParity(Enumerable.Range(0, K2).Select(j => currentMatrix2D[i, j]));
```

```
            calculatedYrFromYkList.AddRange(tempP1);
        }
        if (NumParityGroups >= 2)
        {
            var tempP2 = new int[K2];
            for (int j = 0; j < K2; j++) tempP2[j] =
CalculateParity(Enumerable.Range(0, K1).Select(i => currentMatrix2D[i, j]));
            calculatedYrFromYkList.AddRange(tempP2);
        }
    }
    else // 3D
    {
        if (NumParityGroups >= 1)
        {
            var tempP1 = new int[K2 * Z.Value];
            int p1Idx = 0; for (int k = 0; k < Z.Value; k++) for (int j = 0;
j < K2; j++) tempP1[p1Idx++] = CalculateParity(Enumerable.Range(0,
K1).Select(i => currentMatrix3D[i, j, k]));
            calculatedYrFromYkList.AddRange(tempP1);
        }
        if (NumParityGroups >= 2)
        {
            var tempP2 = new int[K1 * Z.Value];
            int p2Idx = 0; for (int k = 0; k < Z.Value; k++) for (int i = 0;
i < K1; i++) tempP2[p2Idx++] = CalculateParity(Enumerable.Range(0,
K2).Select(j => currentMatrix3D[i, j, k]));
            calculatedYrFromYkList.AddRange(tempP2);
        }
        if (NumParityGroups >= 3)
        {
            var tempP3 = new int[K1 * K2];
            int p3Idx = 0; for (int i = 0; i < K1; i++) for (int j = 0; j <
K2; j++) tempP3[p3Idx++] = CalculateParity(Enumerable.Range(0,
Z.Value).Select(k => currentMatrix3D[i, j, k]));
            calculatedYrFromYkList.AddRange(tempP3);
        }
        if (NumParityGroups >= 4)
        {
            var tempP4 = new int[1];
            tempP4[0] = CalculateParity(currentData); // Используем
извлеченные данные Yk
            calculatedYrFromYkList.AddRange(tempP4);
        }
    }
    PrintUtils.PrintVector(calculatedYrFromYkList, "Yr'"); // Выводим Yr,
вычисленное по Yk

    // Сравнение покажет разницу (это и есть синдром в развернутом виде)
    bool initialMatch =
fullReceivedYrList.SequenceEqual(calculatedYrFromYkList);
    Console.WriteLine($"Match: {initialMatch}");
    Console.WriteLine("------\n");


    // Итеративный Цикл Декодирования
    for (int iter = 0; iter < MAX_DECODING_ITERATIONS; iter++)
    {

        bool correctionMade = false;

        // --- Вычисление Синдромов ---
        int[] syndrome1 = null, syndrome2 = null, syndrome3 = null, syndrome4
= null;
        if (!Z.HasValue) // 2D Case
```

```
        {
            syndrome1 = CalculateSyndrome2D(currentMatrix2D, receivedP1, 1);
// Row Syndrome
            syndrome2 = CalculateSyndrome2D(currentMatrix2D, receivedP2, 2);
// Col Syndrome
        }
        else // 3D Case
        {
            syndrome1 = CalculateSyndrome3D(currentMatrix3D, receivedP1, 1);
// k1 dir
            syndrome2 = CalculateSyndrome3D(currentMatrix3D, receivedP2, 2);
// k2 dir
            syndrome3 = CalculateSyndrome3D(currentMatrix3D, receivedP3, 3);
// z dir
            syndrome4 = CalculateSyndrome3D(currentMatrix3D, receivedP4, 4);
// overall
        }

        // Проверка Сходимости
        bool allZero = (syndrome1?.All(s => s == 0) ?? true) &&
                       (syndrome2?.All(s => s == 0) ?? true) &&
                       (syndrome3?.All(s => s == 0) ?? true) &&
                       (syndrome4?.All(s => s == 0) ?? true);

        if (allZero)
        {
            break;
        }

        // --- Идентификация и Исправление Ошибок ---
        if (!Z.HasValue && currentMatrix2D != null) // 2D Correction
        {
            for (int i = 0; i < K1; i++)
            {
                for (int j = 0; j < K2; j++)
                {
                    int failingChecks = 0;
                    if   (syndrome1   !=   null   &&   syndrome1[i]   ==   1)
failingChecks++;
                    if   (syndrome2   !=   null   &&   syndrome2[j]   ==   1)
failingChecks++;

                    if (failingChecks >= 2)
                    {
                        currentMatrix2D[i, j] = 1 - currentMatrix2D[i, j];
                        correctionMade = true;
                    }
                }
            }
        }
        else if (Z.HasValue && currentMatrix3D != null)
        {
            for (int i = 0; i < K1; i++)
            {
                for (int j = 0; j < K2; j++)
                {
                    for (int k = 0; k < Z.Value; k++)
                    {
                        int failingChecks = 0;
                        if (syndrome1 != null && syndrome1[k * K2 + j] == 1)
failingChecks++;
                        if (syndrome2 != null && syndrome2[k * K1 + i] == 1)
failingChecks++;
                        if (syndrome3 != null && syndrome3[i * K2 + j] == 1)
```

```
failingChecks++;

                        if (failingChecks >= 2)
                        {
                            currentMatrix3D[i, j, k] = 1 - currentMatrix3D[i,
j, k];
                            correctionMade = true;
                        }
                    }
                }
            }
        }

        if (!correctionMade && !allZero)
        {
            break;
        }
        if (iter == MAX_DECODING_ITERATIONS - 1 && !allZero)
        {
            // Log message about reaching max iterations
        }
    }

    if (!Z.HasValue)
    {
        CalculateParityBitsFromMatrix(currentMatrix2D);   //   Пересчет   по
ИСПРАВЛЕННОЙ матрице
        currentData    =    FlattenMatrix(currentMatrix2D);    //    Получение
ИСПРАВЛЕННЫХ данных
    }
    else
    {
        CalculateParityBitsFromMatrix(currentMatrix3D);   //   Пересчет   по
ИСПРАВЛЕННОЙ матрице
        currentData    =    FlattenMatrix(currentMatrix3D);    //    Получение
ИСПРАВЛЕННЫХ данных
    }

    var correctedList = new List<int>(currentData);
    if (_parityGroup1 != null) correctedList.AddRange(_parityGroup1);
    if (_parityGroup2 != null) correctedList.AddRange(_parityGroup2);
    if (_parityGroup3 != null) correctedList.AddRange(_parityGroup3);
    if (_parityGroup4 != null) correctedList.AddRange(_parityGroup4);
    _correctedWordYnPrime = correctedList.ToArray(); // Финальное Yn'

    return (int[])_correctedWordYnPrime.Clone();
}
```

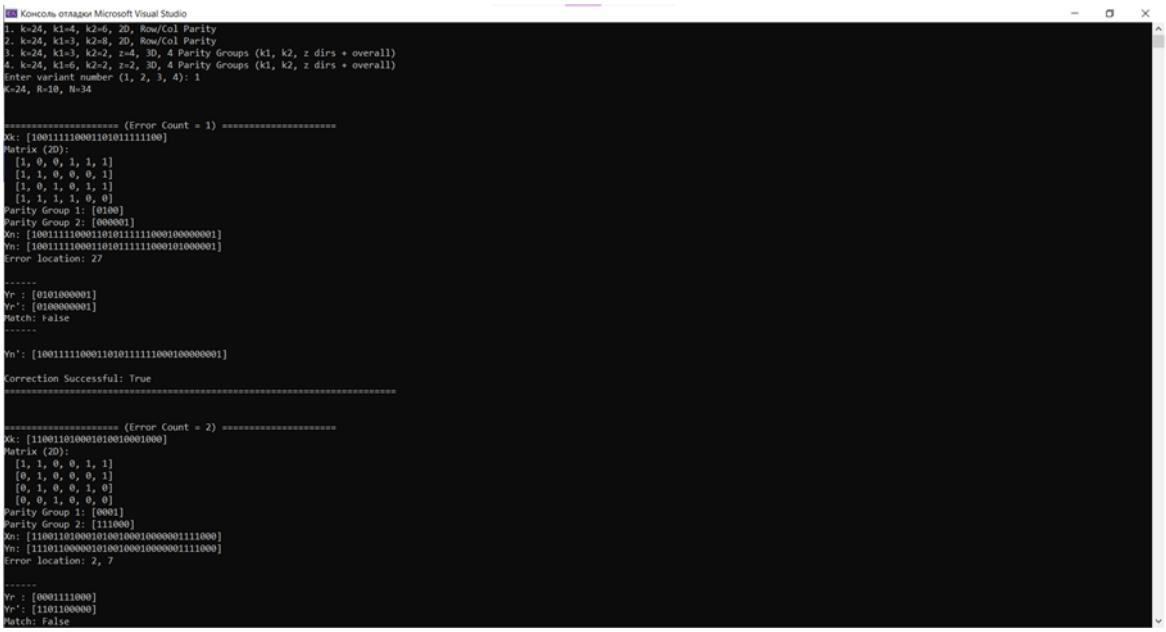
Листинг 5.1 – Функция для декодирования

Задание 6. выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов Xn и Yn’; 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.

```
static void RunAnalysis(IterativeCode coder, int errorMultiplicity, int
numTrials)
```

```
{
    Console.WriteLine($"\n============     (Analysis:     Error     Count     =
{errorMultiplicity}, Tests = {numTrials}) ============");

    if (numTrials <= 0) return;

    int correctedCount = 0;

    int[] infoWord = GenerateRandomBinaryWord(coder.K);
    int[] xn = coder.Encode(infoWord);

    for (int i = 0; i < numTrials; i++)
    {
        coder.IntroduceErrors(errorMultiplicity); // Yn
        coder.Decode(); // Yn'
        if (coder.AnalyzeCorrection()) // Compare Xn and Yn'
        {
            correctedCount++;
        }

        if ((i + 1) % (numTrials / 10 == 0 ? numTrials / 10 + 1 : numTrials
/ 10) == 0)
        {
            Console.Write($"{(int)(((double)(i + 1) / numTrials) * 100)}% ");
        }
    }

    // Calculate and display results (N3/N1)
    double correctionRate = (double)correctedCount / numTrials;

    Console.WriteLine($"Tests: {numTrials}");
    Console.WriteLine($"Correctly Corrected : {correctedCount}");
    Console.WriteLine($"Correction Rate: {correctionRate:P2}");

Console.WriteLine("================================================================
==================================");
}
```

Листинг 6.1 – Фунция для анализа



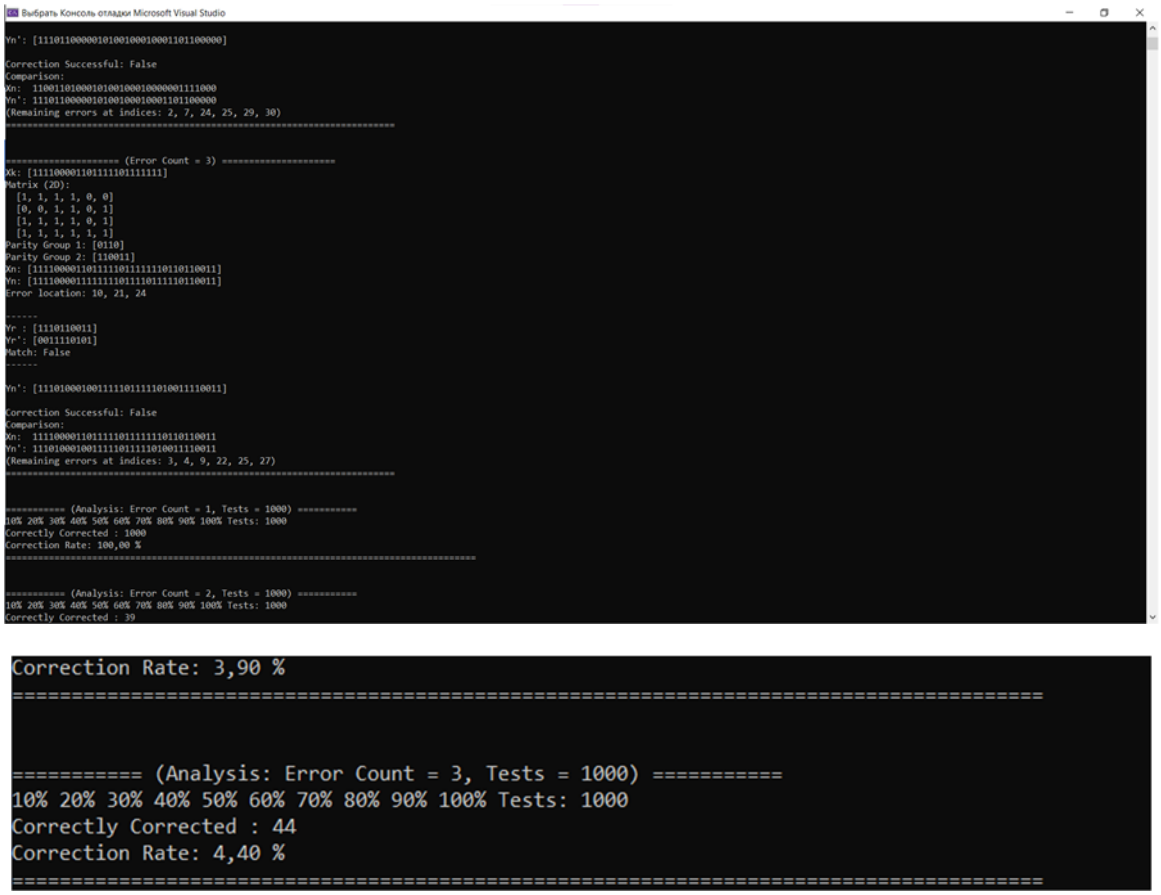


Рисунок 6.1 – Результат работы

Вывод: В ходе выполнения данной лабораторной работы были изучены принципы избыточного кодирования данных и рассмотрены особенности итеративных кодов как одного из мощных инструментов повышения помехоустойчивости информационных систем.