

Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ»

Защита информации и надёжность информационных
систем

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2024

Лабораторная работа № 5

ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. ИТЕРАТИВНЫЕ КОДЫ

Цель: приобретение практических навыков кодирования/декодирования двоичных данных при использовании итеративных кодов.

Задачи:

1. Закрепить теоретические знания по использованию итеративных кодов для повышения надежности передачи и хранения в памяти компьютера двоичных данных.
2. Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.
3. 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 (k_1) и столбцами кода (k_2).

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.

Принято считать рассматриваемый код многомерным, если количество измерений, по которым вычисляются и анализируются паритеты, не менее 3. Таким образом, простейшим многомерным линейным итеративным кодом является код трехмерный.

Практическое задание

Разработать собственное приложение, которое позволяет выполнять следующие операции:

1) вписывать произвольное двоичное представление информационного слова X_k (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);

Для начала составим функцию для заполнения матрицы, строки и столбцы которой зависят от выбранных значений размерности:

```
public 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
    {
        for (int k = 0; k < _z.Value; k++)
        {
            for (int i = 0; i < _k1; i++)
            {
                for (int j = 0; j < _k2; j++)
                {
                    _matrix3D[i, j, k] = data[index++];
                }
            }
        }
    }
}

public void FillMatrix2DFromData(int[] data, int[,] matrix)
{
    int index = 0;
    for (int i = 0; i < _k1; i++)
        for (int j = 0; j < _k2; j++)
            matrix[i, j] = data[index++];
}

public void FillMatrix3DFromData(int[] data, int[,,] matrix)
```

```

{
    int index = 0;
    for (int k = 0; k < _z.Value; k++)
        for (int i = 0; i < _k1; i++)
            for (int j = 0; j < _k2; j++)
                matrix[i, j, k] = data[index++];
}

```

Листинг 1.1 – Функции заполнения матриц.

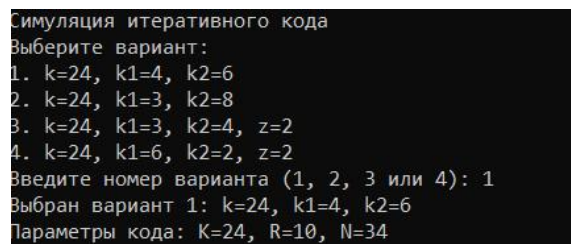
Варианты заполнения матрицы:

```

switch (choice)
{
    case "1":
        k1 = 4; k2 = 6; z = null; numParityGroups = 2;
        Console.WriteLine("Выбран вариант 1: k=24, k1=4,
k2=6");
        break;
    case "2":
        k1 = 3; k2 = 8; z = null; numParityGroups = 2;
        Console.WriteLine("Выбран вариант 2: k=24, k1=3,
k2=8");
        break;
    case "3":
        k1 = 3; k2 = 4; z = 2; numParityGroups = 4;
        Console.WriteLine("Выбран вариант 3: k=24, k1=3,
k2=4, z=2");
        break;
    case "4":
        k1 = 6; k2 = 2; z = 2; numParityGroups = 4;
        Console.WriteLine("Выбран вариант 3: k=24, k1=6,
k2=2, z=2");
        break;
    default:
        Console.WriteLine("Неверный выбор. Выход.");
        return;
}

```

Листинг 1.1 – Ввод значений k1, k2, z по варианту.



```

Симуляция итеративного кода
Выберите вариант:
1. k=24, k1=4, k2=6
2. k=24, k1=3, k2=8
3. k=24, k1=3, k2=4, z=2
4. k=24, k1=6, k2=2, z=2
Введите номер варианта (1, 2, 3 или 4): 1
Выбран вариант 1: k=24, k1=4, k2=6
Параметры кода: K=24, R=10, N=34

```

Рисунок 1.1 – Результат работы приложения.

```
Матрица данных (2D):  
[0, 0, 1, 0, 1, 0]  
[1, 0, 0, 0, 1, 0]  
[1, 1, 0, 0, 1, 1]  
[1, 1, 0, 1, 1, 1]
```

Рисунок 1.2 – Результат работы приложения.

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

```
private void CalculateParityBits()  
{  
    if (_matrix2D != null)  
    {  
        _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] =  
_parityCalculator.CalculateParity(row);  
        }  
  
        _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] =  
_parityCalculator.CalculateParity(col);  
        }  
        _parityGroup3 = null;  
        _parityGroup4 = null;  
    }  
    else  
    {  
        _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++] =  
_parityCalculator.CalculateParity(Enumerable.Range(0,  
K1).Select(i => _matrix3D[i, j, k]));  
            }  
        }  
    }  
}
```

```

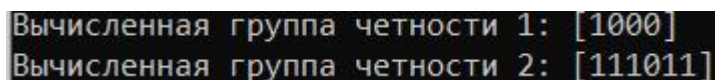
        _parityGroup2 = new int[K1 * Z.Value];
        int p2Idx = 0;
        for (int k = 0; k < Z.Value; k++)
        {
            for (int i = 0; i < K1; i++)
            {
                _parityGroup2[p2Idx++] =
                _parityCalculator.CalculateParity(Enumerable.Range(0,
                K2).Select(j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

        _parityGroup3 = new int[K1 * K2];
        int p3Idx = 0;
        for (int i = 0; i < K1; i++)
        {
            for (int j = 0; j < K2; j++)
            {
                _parityGroup3[p3Idx++] =
                _parityCalculator.CalculateParity(Enumerable.Range(0,
                Z.Value).Select(k => _matrix3D[i, j, k]));
            }
        }

        _parityGroup4 = new int[1];
        _parityGroup4[0] =
        _parityCalculator.CalculateParity(_informationWord);
    }
}

```

Листинг 2.1 – Функция вычисления битов паритетов.



```

Вычисленная группа четности 1: [1000]
Вычисленная группа четности 2: [111011]

```

Рисунок 2.1 – Результат работы приложения.

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

```

private void FormCodewordXn()
{
    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();

        if (_codewordXn.Length != N)
        {
            Console.WriteLine($"Предупреждение: Вычисленная
длина кодового слова ({_codewordXn.Length}) не соответствует
ожидаемой N ({N}). R={R}");
            N = _codewordXn.Length;
        }
    }
}

```

Листинг 3.1 – Функция формирования кодового слова.

```

Запуск единичной демонстрации (Количество ошибок = 1)
Xk (Информационное слово): [001010100010110011110111]
Xn (Исходное кодовое слово): [0010101000101100111101110001101100]

```

Рисунок 3.1 – Результат работы приложения.

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

```

public int[] IntroduceErrors(int errorCount)
{
    if (_codewordXn == null)
    {
        throw new InvalidOperationException("Кодирование
должно быть выполнено до внесения ошибок.");
    }
    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 добавления ошибок.

```

Вычисленная группа четности 1: [0001]
Вычисленная группа четности 2: [101100]
Yn (Принятое слово с ошибками): [0010101010101100111101110001101100]
(Ошибки внесены в позиции: 8)

```

Рисунок 4.1 – Результат работы приложения.

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’);

```

public int[] Decode()
{
    if (_receivedWordYn == null)
    {
        throw new InvalidOperationException("Ошибки должны
        быть внесены (или Yn установлен) до декодирования.");
    }

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

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

    int[] currentData =
    _correctedWordYnPrime.Take(K).ToArray();
    int[] receivedP1 =
    _correctedWordYnPrime.Skip(K).Take(_parityGroup1?.Length ??
    0).ToArray();
    int[] receivedP2 = _correctedWordYnPrime.Skip(K +
    (_parityGroup1?.Length ?? 0)).Take(_parityGroup2?.Length ??
    0).ToArray();
    int[] receivedP3 = _correctedWordYnPrime.Skip(K +
    (_parityGroup1?.Length ?? 0) + (_parityGroup2?.Length ??
    0)).Take(_parityGroup3?.Length ?? 0).ToArray();
    int[] receivedP4 = _correctedWordYnPrime.Skip(K +
    (_parityGroup1?.Length ?? 0) + (_parityGroup2?.Length ?? 0) +
    (_parityGroup3?.Length ?? 0)).Take(_parityGroup4?.Length ??

```



```

0).ToArray();

        if (!Z.HasValue)
        {
            currentMatrix2D = new int[K1, K2];

            _matrixOperations.FillMatrix2DFromData(currentData,
            currentMatrix2D);
        }
        else
        {
            currentMatrix3D = new int[K1, K2, Z.Value];

            _matrixOperations.FillMatrix3DFromData(currentData,
            currentMatrix3D);
        }

        for (int iter = 0; iter < MAX_DECODING_ITERATIONS;
        iter++)
        {
            bool correctionMade = false;

            int[] syndrome1 = null, syndrome2 = null,
            syndrome3 = null, syndrome4 = null;
            if (!Z.HasValue)
            {
                syndrome1 =
                _syndromeCalculator.CalculateSyndrome2D(currentMatrix2D,
                receivedP1, 1);
                syndrome2 =
                _syndromeCalculator.CalculateSyndrome2D(currentMatrix2D,
                receivedP2, 2);
            }
            else
            {
                syndrome1 =
                _syndromeCalculator.CalculateSyndrome3D(currentMatrix3D,
                receivedP1, 1);
                syndrome2 =
                _syndromeCalculator.CalculateSyndrome3D(currentMatrix3D,
                receivedP2, 2);
                syndrome3 =
                _syndromeCalculator.CalculateSyndrome3D(currentMatrix3D,
                receivedP3, 3);
                syndrome4 =
                _syndromeCalculator.CalculateSyndrome3D(currentMatrix3D,
                receivedP4, 4);
            }
        }
    }
}

```

```

    }

    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;
    }

    if (!Z.HasValue && currentMatrix2D != null)
    {
        correctionMade =
_errorCorrector.CorrectErrors2D(currentMatrix2D, syndrome1,
syndrome2);
    }
    else if (Z.HasValue && currentMatrix3D != null)
    {
        correctionMade =
_errorCorrector.CorrectErrors3D(currentMatrix3D, syndrome1,
syndrome2, syndrome3);
    }

    if (!correctionMade && !allZero)
    {
        break;
    }
    if (iter == MAX_DECODING_ITERATIONS - 1
&& !allZero)
    {
    }

    }

    if (!Z.HasValue)
    {
        CalculateParityBitsFromMatrix(currentMatrix2D);
        currentData =
_matrixOperations.FlattenMatrix(currentMatrix2D);
    }
    else
    {

```

```

        CalculateParityBitsFromMatrix(currentMatrix3D);
        currentData =
        _matrixOperations.FlattenMatrix(currentMatrix3D);
    }

    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();

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

```

Листинг 5.1 – Функция декодирования.

```

Начало процесса декодирования...
Процесс декодирования завершен.
Yn' (Исправленное слово): [0010101000101100111101110001101100]

Исправление успешно: True

```

Рисунок 5.1 – Результат работы приложения.

б) выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов X_n 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.

```

static void RunAnalysis(IterativeCode coder, int
errorMultiplicity, int numTrials)
{
    Console.WriteLine($"{ "\n Запуск анализа (Кратность ошибок
= {errorMultiplicity}, Количество испытаний = {numTrials}) ");
    if (numTrials <= 0) return;

    int correctedCount = 0;

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

```

```

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

Console.Write("  Проверка: ");
for (int i = 0; i < numTrials; i++)
{
    coder.IntroduceErrors(errorMultiplicity);
    coder.Decode();
    if (coder.AnalyzeCorrection())
    {
        correctedCount++;
    }

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

double correctionRate = (double)correctedCount /
numTrials;

Console.WriteLine($"{Environment.NewLine}Результаты для {errorMultiplicity}
ошибок:");
Console.WriteLine($"  Всего испытаний: {numTrials}");
Console.WriteLine($"  Успешно исправлено:
{correctedCount}");
Console.WriteLine($"  Процент успешных исправлений:
{correctionRate:P2}");
Console.WriteLine(" Конец анализа ");
}

```

Листинг 6.1 – Вывод испытаний.

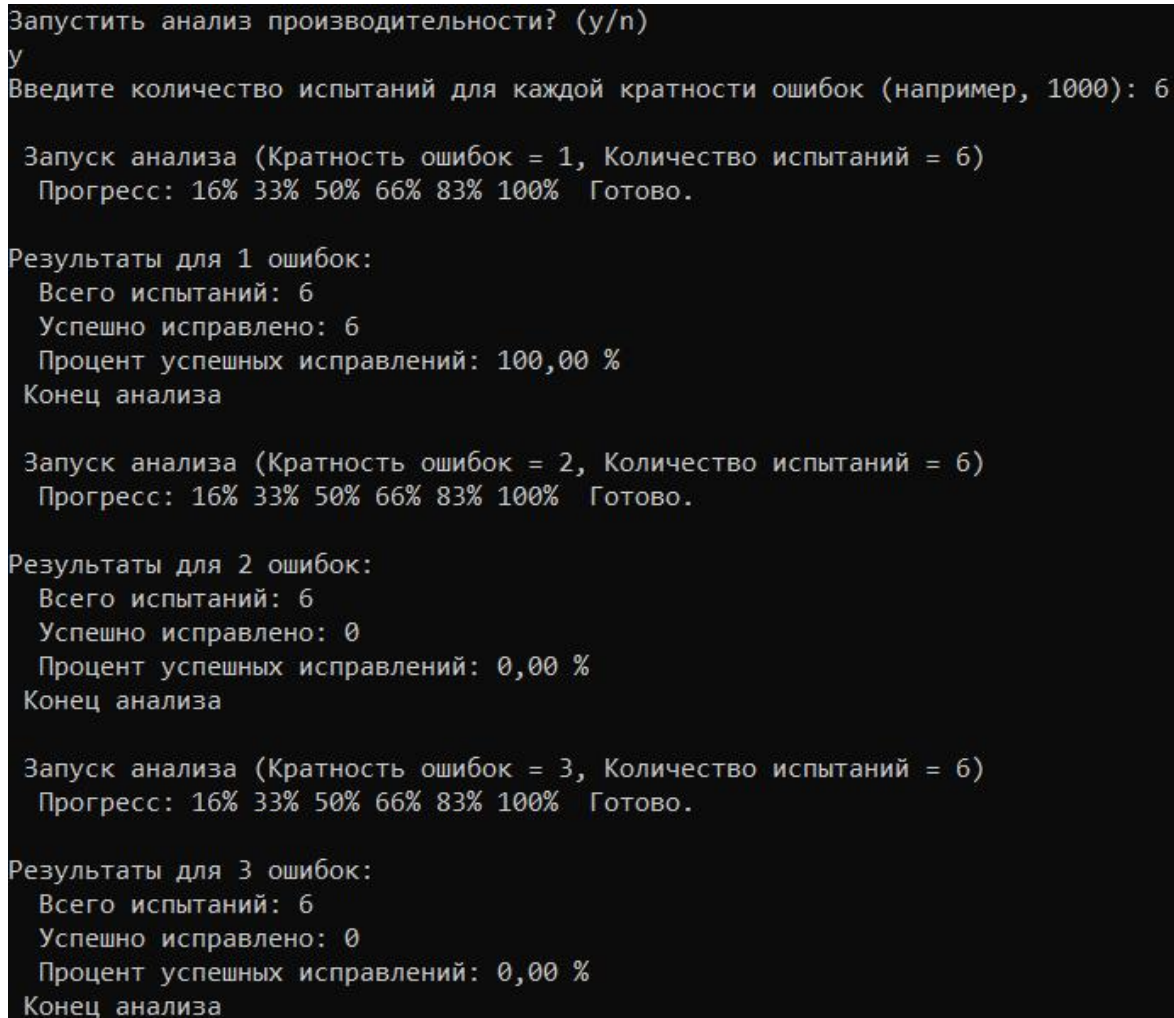
```

public bool AnalyzeCorrection()
{
    if (_codewordXn == null || _correctedWordYnPrime ==
null)
    {
        return false;
    }
    if (_codewordXn.Length != _correctedWordYnPrime.Length)
    {
        Console.WriteLine($"Невозможно проанализировать:
Длина не соответствует Xn={_codewordXn.Length},
Yn'={_correctedWordYnPrime.Length}");
        return false;
    }
}

```

```
    }  
    return  
_codewordXn.SequenceEqual(_correctedWordYnPrime);  
}
```

Листинг 6.1 – Функция анализа коррекций.



```
Запустить анализ производительности? (y/n)  
y  
Введите количество испытаний для каждой кратности ошибок (например, 1000): 6  
  
Запуск анализа (Кратность ошибок = 1, Количество испытаний = 6)  
Прогресс: 16% 33% 50% 66% 83% 100% Готово.  
  
Результаты для 1 ошибок:  
Всего испытаний: 6  
Успешно исправлено: 6  
Процент успешных исправлений: 100,00 %  
Конец анализа  
  
Запуск анализа (Кратность ошибок = 2, Количество испытаний = 6)  
Прогресс: 16% 33% 50% 66% 83% 100% Готово.  
  
Результаты для 2 ошибок:  
Всего испытаний: 6  
Успешно исправлено: 0  
Процент успешных исправлений: 0,00 %  
Конец анализа  
  
Запуск анализа (Кратность ошибок = 3, Количество испытаний = 6)  
Прогресс: 16% 33% 50% 66% 83% 100% Готово.  
  
Результаты для 3 ошибок:  
Всего испытаний: 6  
Успешно исправлено: 0  
Процент успешных исправлений: 0,00 %  
Конец анализа
```

Рисунок 6.1 – Результат работы приложения.

Вывод

Симуляция позволяет наглядно продемонстрировать и проанализировать работу итеративного кодирования для исправления ошибок. Полученные результаты могут быть использованы для оценки эффективности различных вариантов кода и выбора оптимального варианта для конкретных условий передачи данных. Изменяя параметры, можно увидеть как количество и расположение битов четности влияет на устойчивость кода к ошибкам. Эксперименты показывают, что 3D код обладает большей устойчивостью к ошибкам по сравнению с 2D кодами при равной длине кодового слова. Однако, это достигается за счет увеличения сложности декодирования.