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 2025

**Лабораторная работа № 6**

**ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХВ ИНФОРМАЦИОННЫХ СИСТЕМАХ. ЦИКЛИЧЕСКИЕ КОДЫ**

**Цель**: приобретение практических навыков кодирования/декодирования двоичных данных при использовании циклических

кодов (ЦК).

**Задачи**:

1. 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.
2. Разработать приложение для кодирования/декодирования двоичной информации циклическим кодом.
3. 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 Х).

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

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

Возможные порождающие полиномы для различных длин k информационного слова, найденные с помощью ЭВМ, сведены в обширные таблицы.

Синдромом ошибки в этих кодах является наличие остатка от деления принятой кодовой комбинации на порождающий полином. Если синдром равен нулю, то считается, что ошибок нет. В противном случае с помощью полученного синдрома можно определить номер разряда принятой кодовой комбинации, в котором произошла ошибка, и исправить ее примерно по той же схеме, которую мы использовали для кода Хемминга. При этом следует обратить внимание на важную деталь: умножение полинома на х приводит к сдвигу членов полинома на один разряд влево, а при умножении на х^r – на r разрядов влево с заменой r младших разрядов полинома на нули. Деление же полинома на х приводит к соответствующему сдвигу членов полинома вправо с уменьшением показателей членов на 1. Такой сдвиг требует дописать справа r проверочных символов к исходной кодовой комбинации Аi(Х) после умножения ее на х^r.

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

Основная операция: принятое кодовое слово (Yn) нужно поделить на по рождающий полином, который использовался при кодировании. Если Yn принадлежит коду, т. е. слово не искажено помехами, то остаток от деления (синдром) будет нулевым. Ненулевой ток свидетельствует о наличии ошибок в принятой кодовой комбинации: Yn ≠ Хn.

Декодирование ненулевого синдрома имеет целью определение ошибочного бита в принятом сообщении или, иначе говоря, определение вектора Еn. Наряду с полиномиальным способом задания кода, структуру построения кода можно определить с помощью матричного представления. При этом в ряде случаев проще реализуется построение кодирующих и декодирующих устройств ЦК. Здесь нам следует вернуться к упомянутым матрицам и представлению ЦК с их помощью.

**Практическое задание**

1. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы 1.1, из которого 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.

Таблица 1.1

|  |  |  |
| --- | --- | --- |
| Вариант | Количество избыточных символов кода, r | Полином |
| 8 | 5 | х5 + х4 + х3 + х2 + 1 |

Задаём параметры для выполнения:

|  |
| --- |
| int r = 5;  int n = 10;  int k = n - r;  int[] g = { 1, 1, 1, 1, 0, 1 }; |

Листинг 1.1 – Параметры для выполнения.

Возьмём для примера Xk = 1101. Для k = 4 берём n = 7, r = 3.  
 Порождающий полином G(x) = x^3 + x + 1, что соответствует двоичной строке 1011.

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

|  |
| --- |
| static int[,] GenerateGeneratorMatrix(int k, int r, int[] g)  {  int n = k + r;  int[,] G = new int[k, n];  for (int i = 0; i < k; i++)  {  for (int j = 0; j <= r; j++)  {  G[i, i + j] = g[j];  }  }  return G;  } |

Листинг 2.1 – Функция для составления порождающей матрицы.

|  |
| --- |
| static int[,] GenerateHMatrix(int[,] G, 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[j, k + i];  H[i, k + i] = 1;  }  return H;  } |

Листинг 2.3 – Функции для составления проверочной матрицы.

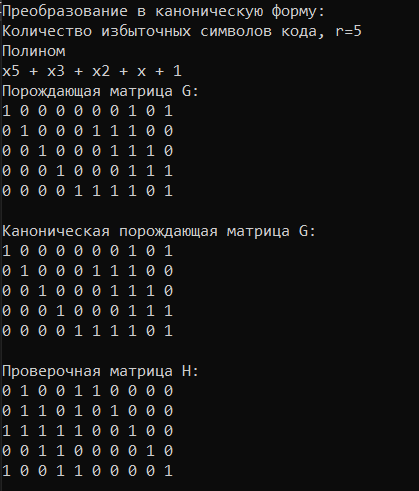


Рисунок 2.1 – Результат работы программы

Строим порождающую матрицу. Эта матрица строится путём циклического сдвига порождающего полинома G(x). Каждая строка матрицы получается сдвигом порождающего полинома вправо на количество позиций, соответствующее номеру строки (начиная с 0), с дополнением нулями до длины n.

Для примера Xk = 1101 и порождающего полинома G(x) = x³ + x + 1, что соответствует в двоичной форме 1011, порождающая матрица будет выглядеть так:

Первая строка — это сам порождающий полином: [1, 0, 1, 1 | 0, 0, 0].  
Вторая строка — это сдвиг порождающего полинома на 1 вправо: [0, 1, 0, 1 | 1, 0, 0].  
Третья строка — сдвиг на 2: [0, 0, 1, 0 | 1, 1, 0].  
Четвёртая строка — сдвиг на 3: [0, 0, 0, 1 | 0, 1, 1].

Затем мы приводим матрицу к каноническому виду. Для этого с помощью элементарных строковых операций преобразуем её так, чтобы первые k столбцов образовали единичную матрицу.

После преобразования канонический вид матрицы будет таким:

Первая строка: [1, 0, 0, 0 | 1, 0, 1].  
Вторая строка: [0, 1, 0, 0 | 1, 1, 1].  
Третья строка: [0, 0, 1, 0 | 1, 1, 0].  
Четвёртая строка: [0, 0, 0, 1 | 0, 1, 1].

Для проверки канонической формы строим проверочную матрицу H. Структура проверочной матрицы для канонического кода выглядит как H=[PT∣Ir]H = [P^T | I\_r]H=[PT∣Ir​], где PTP^TPT — транспонированная матрица P, а IrI\_rIr​ — единичная матрица размером r × r.

Проверочная матрица для нашего примера будет такой:

Первая строка: [1, 1, 1, 0 | 1, 0, 0].  
Вторая строка: [0, 1, 1, 1 | 0, 1, 0].  
Третья строка: [1, 1, 0, 1 | 0, 0, 1].

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

|  |
| --- |
| static int[] MultiplyVectorByMatrix(int[] vector, int[,] matrix)  {  int[] result = new int[matrix.GetLength(1)];  for (int j = 0; j < matrix.GetLength(1); j++)  {  int sum = 0;  for (int i = 0; i < vector.Length; i++)  sum ^= vector[i] \* matrix[i, j];  result[j] = sum;  }  return result;  } |

Листинг 3.1 – Функция для вычисления избыточных символов



Рисунок 3.1 – Результат работы программы

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

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

Листинг 4.1 – Функция для генерации ошибок

Примем:

Yn1=1011000

Yn2 = 0011000

Yn3 = 0011000

1. 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 Еn = е1, е2, …, еn и исправить одиночную ошибку, используя выражение (6.5); проанализировать ситуацию при возникновении ошибки в 2 битах.

Синдром определяется как остаток от деления YnY\_nYn​ на порождающий полином G(x)G(x)G(x).

Для Yn1=1011000Y\_{n1} = 1011000Yn1​=1011000, делим x6+x4+x3x^6 + x^4 + x^3x6+x4+x3 на x3+x+1x^3 + x + 1x3+x+1. Получаем остаток 0, что означает отсутствие ошибок.

Для Yn2=0011000Y\_{n2} = 0011000Yn2​=0011000, делим x4+x3x^4 + x^3x4+x3 на x3+x+1x^3 + x + 1x3+x+1. Остаток равен x2+1x^2 + 1x2+1, что соответствует синдрому 101. Это означает, что ошибка произошла в 1 бите. Вектор ошибок EnE\_nEn​ равен 1000000.

Для Yn3=0011000Y\_{n3} = 0011000Yn3​=0011000, делим x4x^4x4 на x3+x+1x^3 + x + 1x3+x+1. Остаток равен x2+xx^2 + xx2+x, что соответствует синдрому 110. Это указывает на ошибки в 1 и 4 битах. Сложив 1 и 4 столбцы проверочной матрицы, получаем синдром 110, что подтверждает наличие ошибок.

|  |
| --- |
| static int[] CorrectSingleError(int[] syndrome, int[,] H, int[] received, out int errorPosition)  {  int r = H.GetLength(0);  int n = H.GetLength(1);  errorPosition = -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)  {  errorPosition = j;  break;  }  }  int[] corrected = (int[])received.Clone();  if (errorPosition != -1)  corrected[errorPosition] ^= 1;  return corrected;  }  static void AnalyzeWithErrorCount(int[] codeword, int[,] H, int errorCount, int n)  {  Console.WriteLine($"\n--- Анализ с {errorCount} ошибкой(ами) ---");  List<int> errorPositions;  int[] received = IntroduceErrors(codeword, errorCount, out errorPositions);  Console.WriteLine("Yn: " + string.Join("", received));  if (errorCount > 0)  Console.WriteLine("Сгенерированные ошибки на позициях: " + string.Join(", ", errorPositions));  else  Console.WriteLine("Ошибка не была сгенерирована");  int[] syndrome = ComputeSyndrome(H, received);  Console.WriteLine("Синдром: " + string.Join("", syndrome));  if (syndrome.All(bit => bit == 0))  {  Console.WriteLine("Ошибок не обнаружено.");  }  else  {  int errorPos;  int[] corrected = CorrectSingleError(syndrome, H, received, out errorPos);  if (errorPos != -1)  {  Console.WriteLine($"Обнаружена ошибка в позиции: {errorPos}");  Console.WriteLine("En: " +  string.Join("", Enumerable.Range(0, n).Select(i => i == errorPos ? "1" : "0")));  Console.WriteLine("Исправленное слово: " + string.Join("", corrected));  }  else  {  Console.WriteLine("Синдром не соответствует одиночной ошибке — возможно, 2 ошибки.");  Console.WriteLine("Исправление невозможно.");  }  }  } |

Листинг 5.1 – Функции для обнаружения и исправления ошибок.

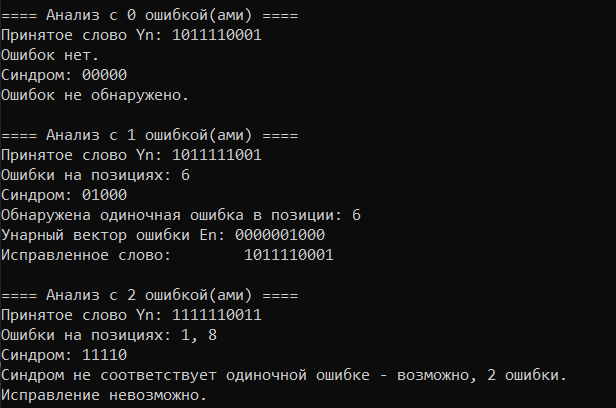


Рисунок 5.1 – Результат работы программы

**Вывод**

Эта программа демонстрирует основные принципы кодирования и декодирования с использованием линейного блочного циклического кода, где целью является добавление избыточности к информационному сообщению для обнаружения и исправления ошибок, возникающих при передаче по каналу с шумами; ключевыми элементами являются порождающий полином, задающий структуру кода, порождающая матрица G для кодирования информационного слова в кодовое, проверочная матрица H для вычисления синдрома, указывающего на наличие ошибок, и сам процесс кодирования, включающий добавление проверочных символов; при передаче по каналу кодовое слово может быть искажено, а декодирование включает вычисление синдрома, обнаружение ошибок и, в простых случаях, исправление одиночных ошибок путем инвертирования соответствующего бита, при этом программа демонстрирует исправление только одиночных ошибок, а более сложные коды способны исправлять большее количество, но требуют более сложных алгоритмов; суть программы заключается в демонстрации математических операций, позволяющих создать систему, устойчивую к ошибкам, выбор подходящего кода определяет эффективность обнаружения и исправления ошибок, а также сложность реализации.