МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5 НА ТЕМУ:**

**Избыточное кодирование данных в информационных системах. Итеративные коды**

Ф.И.О.

Божко Денис Владимирович

Преподаватель

асс. Берников Владислав Олегович

Минск 2021

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

**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.

Запись матрицы в таком виде не считается канонической, так как подматрица I не является единичной диагональной матрицей. Для преобразования такой записи к каноническому виду воспользуемся свойствами линейного кода: в дополнительную строку необходимо записать сумму по модулю 2 соответствующих символов матрицы кода с dmin = 3.

В общем случае любая проверочная матрица кода Хемминга с dmin = 4 имеет нечетный вес столбцов, т. е. вес любого из столбцов подматрицы А′ может быть равен 3, 5, 7, … .

Как и в предыдущем случае (при dmin = 3), равенство нулю синдрома означает отсутствие ошибок. Если же синдром не равен нулю и имеет нечетный вес, то это говорит о том, что произошла одиночная ошибка. Если же синдром не равен нулю и его вес четный, то произошла двойная ошибка, так как вес суммы любых двух столбцов всегда четный.

**Ход работы**

1. Вписывать произвольное двоичное представление информационного слова Хk (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);
2. Вычислять проверочные биты (биты паритетов) по а) двум, б) трем, в) четырем направлениям (группам паритетов);
3. Формировать кодовое слово Xn, присоединением избыточных символов к информационному слову;
4. Генерировать ошибку произвольной кратности (i, i>0), распределенную случайным образом среди символов слова Xn, в результате чего формируется кодовое слово Yn;
5. Определять местоположение ошибочных символов итеративным кодом в слове Yn в соответствии с используемыми группами паритетов по п. 1.2 и исправлять ошибочные символы (результат исправления – слово Yn’;
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.

**Листинг кода**

|  |
| --- |
| import random as rand  import numpy as np  def RandSequence(k):  sequence = []  for i in range(k):  sequence.append(rand.randint(0,1))  return sequence  def Create2DMatrix(Xn, k1, k2):  count = 0  matrix = np.zeros((k1,k2), type(int))    submatrix = []  for i in range(k1):  submatrix.clear()  for k in range(k2):  matrix[i][k] = Xn[count]  count += 1  return matrix  def XORArray(array):  if(np.count\_nonzero(array == 1) % 2 == 0):  return 0  else:  return 1  def CreateParitetRedundancyArray(XMatrix):  k1 = XMatrix.\_\_len\_\_()  k2 = XMatrix[0].\_\_len\_\_()  rows = np.zeros(k1, type(int))  columns = np.zeros(k2, type(int))  for i in range(k1):  rows[i] = XORArray(XMatrix[i])  for i in range(k2):  columns[i] = XORArray(XMatrix[:,i])  return (rows, columns)  def CheckParitetsForErrors(xrr, xrc, yrr, yrc):  haveTwoErrors = False  rowErrorIndex = 0  columnErrorIndex = 0  for i in range(len(xrr)):  if(xrr[i] != yrr[i]):  if(rowErrorIndex != 0):  haveTwoErrors = True  rowErrorIndex = i  for i in range(len(xrc)):  if(xrc[i] != yrc[i]):  if(columnErrorIndex != 0):  haveTwoErrors = True  columnErrorIndex = i  return (haveTwoErrors, rowErrorIndex, columnErrorIndex)  def GenerateErrorsAtMatrix(matrix):  i = rand.randint(0, len(matrix) - 1)  j = rand.randint(0, len(matrix[0]) - 1)  if(matrix[i][j] == 0):  matrix[i][j] = 1  else:  matrix[i][j] = 0  return matrix  def ReverseElementAtMatrixIndex(matrix, i, j):  if(matrix[i][j] == 0):  matrix[i][j] = 1  else:  matrix[i][j] = 0  return matrix  def CreateHammingMatrix(k1, k2):  P = np.zeros((k1+k2, k1\*k2), int)  rowOffset = k2  rowCounter = 0  for i in range(k1):  for j in range(rowOffset \* i, (rowOffset \* i) + (rowOffset)):  P[i][j] = 1  rowCounter += 1  columnOffset = k2  for i in range(k2):  for j in range(k1):  P[rowCounter][(columnOffset \* j) + i] = 1  rowCounter += 1    I = np.zeros((k1 + k2, k1 + k2), int)  for i in range(k1 + k2):  for k in range(k1 + k2):  if(k == i):  I[i][k] = 1  else:  I[i][k] = 0  H = []  tempH = []  for i in range(k1 + k2):  tempH = list(np.concatenate((P[i], I[i]), axis=None))  H.append(tempH)  return np.array(H)    def RedundancySymbols(H, Xn):  Xr = []  r = len(H)  temp = []  for i in range(r):  temp.clear()  for k in range(len(Xn)):  temp.append(Xn[k] and H[i][k])  if(temp.count(1) % 2 == 0):  Xr.append(1)  else:  Xr.append(0)  return Xr    def XOR(a, b):  output = []    for i in range(0, len(a)):  if(a[i] == b[i]):  output.append(0)  else:  output.append(1)  return output  def checkErrors(H, Yn, En):  if(En.count(1) > 0):  print("Transfered message has errors")  errorind = findErrors(H, En)  print("Found error on " + str(errorind + 1) + " position")  fixedMessage = inverseAtIndex(Yn, errorind)  print("Fixed message = " + str(fixedMessage))  else:  print("Transfered message has no errors")  def findErrors(H, En):  k = 0  temp = []  while True:  if(k >= len(H[0])):  break  temp.clear()  for i in range(len(H)):  temp.append(H[i][k])  if(np.array\_equal(temp, En)):  return k  k += 1  return 0  def inverseAtIndex(array, index):  if index >= len(array):  return array  if(array[index] == 0):  array[index] = 1  else:  array[index] = 0  return array  #####################################################  Xn = RandSequence(24)  print(Xn)  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ k1 = 4 k2 = 6 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")  k1 = 4  k2 = 6  matrix46 = Create2DMatrix(Xn, k1, k2)  print(matrix46)  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Paritets ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")  Xrrows, Xrcolmnns = CreateParitetRedundancyArray(matrix46)  print('Rows paritets:')  print(Xrrows)  print('Columns paritets:')  print(Xrcolmnns)  print("~~~~~~~~~~~~~~~~~~~~~~~~~~ Transfer with 0 errors ~~~~~~~~~~~~~~~~~~~~~~~~~~~")  matrix = matrix46  print(matrix46)  print('Rows paritets:')  print(Xrrows)  print('Columns paritets:')  print(Xrcolmnns)  Yrrows, Yrcolumns = CreateParitetRedundancyArray(matrix)  print('Found rows paritets:')  print(Yrrows)  print('Foud columns paritets:')  print(Yrcolumns)  haveTwoErrors, RowError, ColumnError = CheckParitetsForErrors(Xrrows, Xrcolmnns, Yrrows, Yrcolumns)  if(RowError == 0 and ColumnError == 0 and not haveTwoErrors):  print('Message has no errors')  else:  if(haveTwoErrors):  print('Message has two errors. Cant indentify position')  else:  print('Found error on ' + str((RowError + 1) \* (ColumnError + 1)) + ' position')  matrix = ReverseElementAtMatrixIndex(matrix, RowError, ColumnError)  print(matrix)  print("~~~~~~~~~~~~~~~~~~~~~~~~~~ Transfer with 1 error ~~~~~~~~~~~~~~~~~~~~~~~~~~~~")  matrix = GenerateErrorsAtMatrix(matrix46)  print(matrix)  print('Rows paritets:')  print(Xrrows)  print('Columns paritets:')  print(Xrcolmnns)  Yrrows, Yrcolumns = CreateParitetRedundancyArray(matrix)  print('Found rows paritets:')  print(Yrrows)  print('Foud columns paritets:')  print(Yrcolumns)  haveTwoErrors, RowError, ColumnError = CheckParitetsForErrors(Xrrows, Xrcolmnns, Yrrows, Yrcolumns)  if(RowError == 0 and ColumnError == 0 and not haveTwoErrors):  print('Message has no errors')  else:  if(haveTwoErrors):  print('Message has two errors. Cant indentify position')  else:  print('Found error on ' + str((RowError + 1) \* (ColumnError + 1)) + ' position')  matrix = ReverseElementAtMatrixIndex(matrix, RowError, ColumnError)  print(matrix)  print("~~~~~~~~~~~~~~~~~~~~~~~~~~ Transfer with 2 error ~~~~~~~~~~~~~~~~~~~~~~~~~~~~")  matrix = GenerateErrorsAtMatrix(matrix46)  matrix = GenerateErrorsAtMatrix(matrix)  print(matrix)  print('Rows paritets:')  print(Xrrows)  print('Columns paritets:')  print(Xrcolmnns)  Yrrows, Yrcolumns = CreateParitetRedundancyArray(matrix)  print('Found rows paritets:')  print(Yrrows)  print('Foud columns paritets:')  print(Yrcolumns)  haveTwoErrors, RowError, ColumnError = CheckParitetsForErrors(Xrrows, Xrcolmnns, Yrrows, Yrcolumns)  if(RowError == 0 and ColumnError == 0 and not haveTwoErrors):  print('Message has no errors')  else:  if(haveTwoErrors):  print('Message has two errors. Cant indentify position')  else:  print('Found error on ' + str((RowError + 1) \* (ColumnError + 1)) + ' position')  matrix = ReverseElementAtMatrixIndex(matrix, RowError, ColumnError)  print(matrix)  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Encode with matrix ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")  H = CreateHammingMatrix(k1, k2)  print("Hamming matrix: ")  print(H)  print("Encoded message:")  print(Xn)  Xr = RedundancySymbols(H, Xn)  print("Redundancy symbols:")  print(Xr)  print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Transfer with 0 errors ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')  Yn = Xn  Yr = Xr  print('Yn = ' + str(Yn))  print('Yr = ' + str(Yr))  Yrr = RedundancySymbols(H, Yn)  En = XOR(Yr, Yrr)  print('En = ' + str(En))  if(En.count(1) > 0):  print("Transfered message has errors")  else:  print("Transfered message has no errors")  print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Transfer with 1 error ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')  Yn = Xn  errorIndex = rand.randint(0, len(Yn) - 1)  Yn = inverseAtIndex(Yn, errorIndex)  Yr = Xr  print('Yn = ' + str(Yn))  print('Yr = ' + str(Yr))  Yrr = RedundancySymbols(H, Yn)  En = XOR(Yr, Yrr)  print('En = ' + str(En))  checkErrors(H, Yn, En)  print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Transfer with 2 errors ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')  Yn = Xn  errorIndex = rand.randint(0, len(Yn) - 1)  Yn = inverseAtIndex(Yn, errorIndex)  errorIndex = rand.randint(0, len(Yn) - 1)  Yn = inverseAtIndex(Yn, errorIndex)  Yr = Xr  print('Yn = ' + str(Yn))  print('Yr = ' + str(Yr))  Yrr = RedundancySymbols(H, Yn)  En = XOR(Yr, Yrr)  print('En = ' + str(En))  checkErrors(H, Yn, En) |

**Вывод**: в данной работе был рассмотрен вариант итеративного кодирования информации, рассмотрены его плюсы и минусы. Было установлено, что итеративные коды могут корректировать несколько ошибок одинаковой четности.