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 №6 НА ТЕМУ:**

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

Ф.И.О.

Трофимчук Михаил Витальевич

Минск 2022

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

**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). Для более глубокого изучения параметров и свойств ЦК, равно как и других корректирующих кодов, полезно ознакомиться с классическими книгами, например [16−18].

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 порождающий полином, который использовался при кодировании.

**Ход работы**

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.

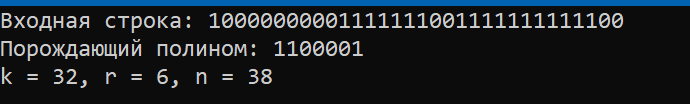
****

Рисунок 1 – Параметры кода и полином

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

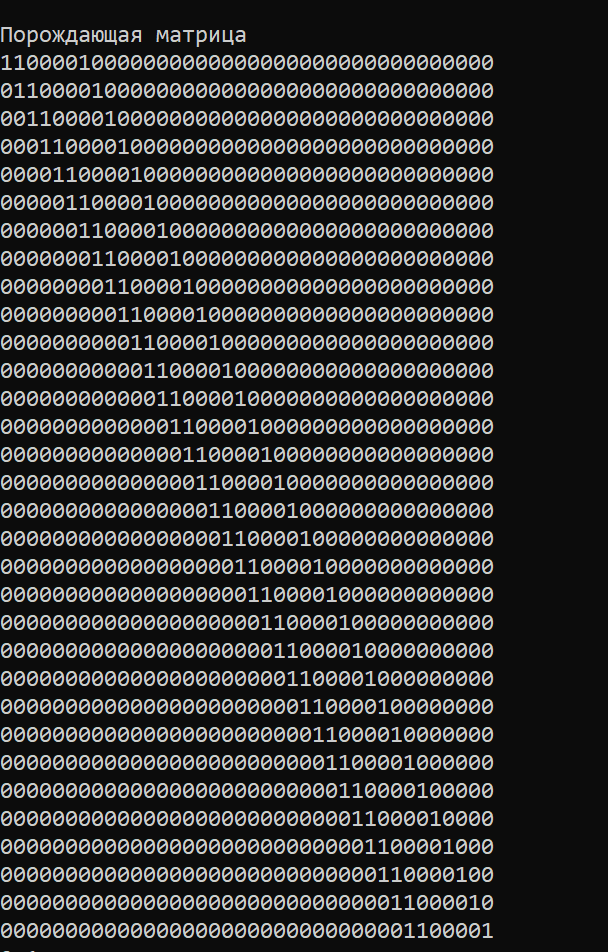


Рисунок 2 – Порождающая матрица

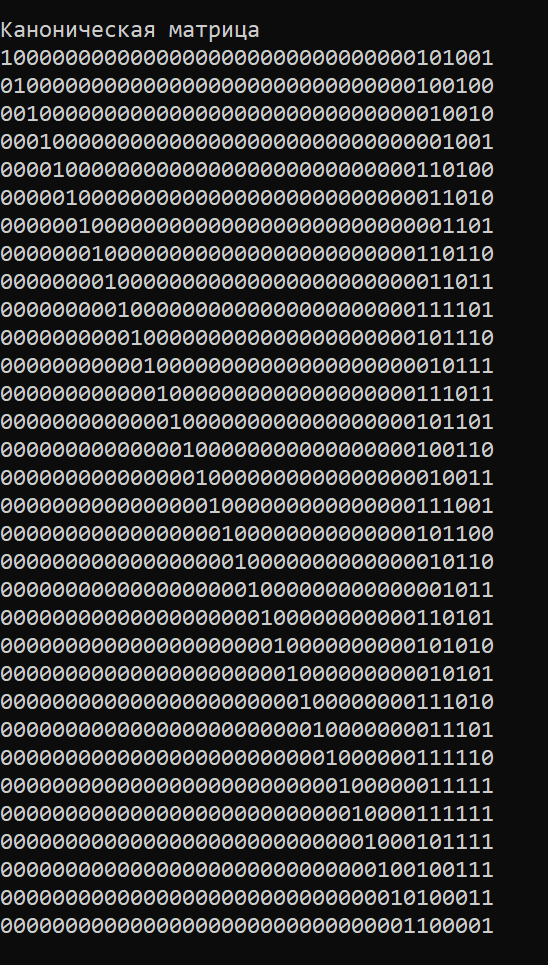


Рисунок 3 – Каноническая матрица

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

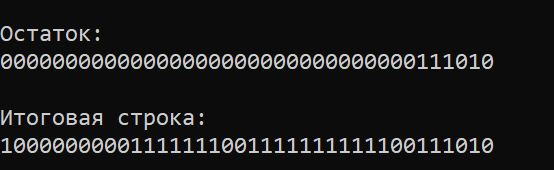


Рисунок 3 – Сформированные избыточные символы с кодовым словом

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

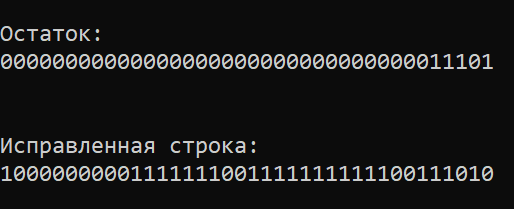


Рисунок 4 – Остаток и справленная трока при одной ошибке

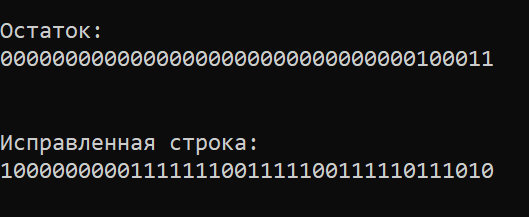


Рисунок 5 – Остаток и справленная трока при двух ошибках

|  |
| --- |
| from numpy.polynomial import polynomial as P  import math  import numpy as np  # вычисляем длину строки  def str\_len(X\_k):  k = len(X\_k)  print('Длина сообщения = "',k,'"')  return k  # вычисляем r  R\_Culc = {  'test':3,  '4':5  }  # добавочный полином  def Edition\_R\_Polinom(r):  r\_poly = []  for i in range(r):  r\_poly.append(0)  r\_poly.append(1)  print('Полином для умножения = ',r\_poly)  return r\_poly  # получаем начальную строку в виде массива  def str\_to\_arr(string):  X\_k\_list = list(string)  X\_k\_list = list(map(int,X\_k\_list))  return X\_k\_list  # полином  Poly = {  'test':[1,1,0,1],  '4':[1,0,0,1,0,1]  }  # умножение полиномов  def multiply\_result(poly\_1,poly\_2):  return P.polymul(poly\_1,poly\_2)  # деление полиномов  def division\_result(poly\_1,poly\_2):  res = P.polydiv(poly\_1,poly\_2)  res = [abs(elem) for elem in res]  return res  # добавление остатка к исходному сообщению  def FinalMessage(poly\_1,poly\_2):  res = []  res.extend(poly\_2)  res.extend(poly\_1)  return res  # проверка ошибки  def Mistake\_Check(Message, EncodePoly):  res = division\_result(Message,EncodePoly)  print(res)  for i in res[1]:  if int(i)==1 or int(i)==-1:  return 0  return 1  # генерация матрицы  def GenerateMatrix(K\_R,K,poly):    Matrix = [ [0]\*K\_R for i in range(K)]  for i in range(K):  Matrix[i]=poly  poly=[poly[-1]] + poly[:-1]  for i in range(K-1):  for j in range(i+1,K):  if Matrix[i][j]==1:  Matrix[i] = list(np.array(Matrix[i])+np.array(Matrix[j]))  for m in range(K):  for l in range(K\_R):  if Matrix[m][l]%2==0:  Matrix[m][l]=0    return Matrix  # подматрица H  def SubMatrixH(K,R,Matrix):  H = []  for i in range(K):  for j in range(K,K+R):  H.append(Matrix[i][j])    H = np.reshape(H,(K,R))  H = np.concatenate((H,np.eye(R,dtype=int).reshape((R,R))))  return H  # исправление ошибок  def ClearMistake(K\_R,K,Fin\_message,curr\_poly):  poly\_for\_matrix = curr\_poly.copy()  poly\_for\_matrix.reverse()  for i in range(K\_R-K):  poly\_for\_matrix.append(0)  matr = GenerateMatrix(K\_R,K,poly\_for\_matrix)  #print(matr)  H = SubMatrixH(K,K\_R-K,matr)  print(H)  divres = division\_result(Fin\_message,curr\_poly)  divres = list(map(int,divres[1]))  if len(divres)!=K:  for i in range(K-len(divres)):  divres.append(0)  print(divres)  E = []  index = 99999  for i in range(K\_R):  if divres==list(H[i]):  index = i;  print(index)  for i in range(K\_R):  if i==index:  E.append(1)  else:  E.append(0)  E.reverse()  print(E)  Corrected\_Message = list(np.array(Fin\_message)+np.array(E))  for i in range(len(Corrected\_Message)):  if Corrected\_Message[i]%2 == 0:  Corrected\_Message[i]=0  print(Corrected\_Message)  # сбор всего  def Cycle\_Code(X\_k):  X\_k\_list = str\_to\_arr(X\_k)  K = str\_len(X\_k)  curr\_r = R\_Culc['4']  r\_poly = Edition\_R\_Polinom(curr\_r)  curr\_poly = Poly['4']  X\_k\_mul\_X\_Pow\_R = list(map(int,multiply\_result(X\_k\_list,r\_poly)))  print('Строка с избыточными битами = ',X\_k\_mul\_X\_Pow\_R)  div\_res = division\_result(X\_k\_mul\_X\_Pow\_R,curr\_poly)  print(div\_res)  Fin\_message = FinalMessage(X\_k\_list,list(map(int,div\_res[1])))  print('Итоговая строка = ',Fin\_message)    Fin\_message = [1, 0, 1, 1, 1, 1, 0, 1, 0, 1]  if Mistake\_Check(Fin\_message,curr\_poly)==1:  print('Ошибок нет')  else:  print('Есть ошибка')  ClearMistake(K+curr\_r,K,Fin\_message,curr\_poly)    def main():  X\_k = '10101'  print('Сообщение = "',X\_k,'"')  Cycle\_Code(X\_k)  main() |

Рисунок 6 – Листинг кода

**Вывод**: в данной работе был рассмотрен циклический вид кодирования информации. Данный вид кодирования позволяет представлять информацию в другом формате, на основе деления двоичной информации на некоторый порождающий полином. Было установлено, что при декодировании у нас есть возможность исправления одной ошибки, как в предыдущих видах кодирования.