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 №7 НА ТЕМУ:**

**Перемежение/деперемежение данных в информационно-вычислительных системах**

Ф.И.О.

Божко Денис Владимирович

Преподаватель

асс. Берников Владислав Олегович

Минск 2021

**Цель:** приобретение практических навыков использования методов перемежения/деперемежения двоичных данных в информационных системах.

**Теоретические сведения**

Проанализированные и исследованные нами коды, как и большинство других кодов, которые были разработаны для увеличения надежности каналов передачи и хранения информации, наиболее эффективны, когда возникающие ошибки статистически независимы, т. е. вероятность передачи (хранения в памяти) любого символа одинакова. Однако довольно часто распределение ошибок носит взаимозависимый характер. В таких случаях говорят о группах (или пакетах) ошибок. Такие ошибки характерны и для каналов передачи, и для устройств памяти (главным образом магнитной и полупроводниковой; cм., например, [9−10, 19−22]).

Существуют специальные коды, корректирующие пакетные ошибки, однако на практике чаще используют перемежение/деперемежение совместно с традиционными кодами

Идея перемежения/деперемежения состоит в следующем. Если биты каждого кодового слова Хn передаются не в обычной последовательности, а через интервалы, превышающие ожидаемую длину пакета ошибок (в промежутки между битами одного слова вставляются биты других кодовых слов), то при возникновении такого типа ошибки обратная перемежению операция – деперемежение – разнесет («размажет») группу ошибок по всей совокупности кодовых слов, составляющих данное сообщение.

Длина пакета в нашем случае – это число рядом расположенных ошибочных битов.

Рассмотренный метод блочного перемежения применяется в SM. К числу других используемых на практике относятся следующие методы перемежения/деперемежения: псевдослучайный, S-типа (применяется в турбо-кодировании: CDMA (Codе Division Multiple Access) – стандарт беспроводной связи множественного доступа с кодовым разделением каналов и др.); циклически-сдвиговый; сверточный; случайный; диагональный; многошаговый [5, 23, 24].

Перемежение (перестановка) символов также является основой некоторых классов криптографических методов [1], которые мы будем анализировать с практической точки зрения в другой части курса.

**Листинг кода**

|  |
| --- |
| import math  import cycliccode  import numpy as np  g = np.array([1, 1, 1])  cc = cycliccode.cycliccode(g, 5)  r = 2  def shuffle(msg, columns):  rows = math.ceil(msg.\_\_len\_\_()/columns)  matrix = np.zeros((rows, columns)).astype(int)  k = 0;  for i in range(rows):  for j in range(columns):  matrix[i,j] = msg[k]  k+=1  print('msg fit into matrix')  print(matrix)  end\_msg = []  transposed = matrix.transpose();  for i in range(columns):  for j in range(rows):  end\_msg.append(transposed[i,j])  return np.array(end\_msg), rows  def split\_and\_encode(x, split\_by: int, times: int):  result = []  for i in range(times):  word = []  for j in range(split\_by):  word.append(x[i\*split\_by+j])  word = np.array(word)  encoded\_msg = cc.c(word)  result.append(encoded\_msg)  return np.array(result)  def split\_and\_decode(x, split\_by: int, times: int):  result = []  for i in range(times):  word = []  for j in range(split\_by):  word.append(x[i\*split\_by+j])  word = np.array(word)  decoded\_msg = cc.syndromeDecode(word)  result.append(decoded\_msg)  return np.array(result) |
| import itertools as perms  import math  import numpy as np  def str\_to\_int\_arr(x: str):  return np.array([int(symbol) for symbol in x])  def generate\_square\_matrix(r):  matrix = np.zeros((r, r))  np.fill\_diagonal(matrix, 1)  return matrix  def binary\_permutations(lst):  for comb in perms.combinations(range(len(lst)), lst.count(1)):  result = [0] \* len(lst)  for i in comb:  result[i] = 1  yield result  def search\_2D(matrix, B):  column\_num = 0  for column in matrix:  if np.array\_equal(B, column):  return column\_num  column\_num += 1  def to\_binary(a):  l, m = [], []  for i in a:  l.append(ord(i))  for i in l:  m.append(int(bin(i)[2:]))  return m  def int\_array\_to\_str(arr):  res = ""  for num in arr:  res += str(num)  return res  def char\_and(a, b):  if (a == '1' or a == 1) and (b == '1' or b == 1):  return True  else:  return False  def xor(a, b):  if (a == b):  return True  else:  return False  # def sum\_xor(items\_list):  # items\_list\_copy = list(items\_list)  # xor\_result = items\_list\_copy.pop()  # while items\_list\_copy.\_\_len\_\_() > 0:  # xor\_result = xor(xor\_result, items\_list\_copy.pop())  # # or just filter(result == 1) and check if length % 2 == 0 or 1  # return xor\_result  def sum\_xor(word: np.ndarray):  ones\_count = np.count\_nonzero(word)  return ones\_count % 2 == 1  def invert(x):  if x == 0 or x == '0':  return 1  else:  return 0  def get\_xr(matrix\_perms, x):  xr = []  for row in matrix\_perms:  seq = zip(row, x)  result = (list(char\_and(x, y) for x, y in seq))  xr.append(sum\_xor(result))  return np.array(xr).astype(int)  def message\_to\_binary\_array(msg: str):  msg = np.array(to\_binary(msg)).flatten()  return int\_array\_to\_str(msg)  def count\_r(xlen: int):  r = np.round\_(math.log2(xlen), 0) + 1  r = int(r)  return r  def generate\_h\_matrix(xlen: int):  permCount = 0  matrixPerms = [[]]  onesCounter = 2  r = count\_r(xlen)  rMatrix = generate\_square\_matrix(r)  while permCount < xlen:  permsSeq = np.concatenate((np.ones(onesCounter), np.zeros(r - onesCounter)))  currentPerms = list(binary\_permutations(permsSeq.tolist()))  permCount += currentPerms.\_\_len\_\_()  onesCounter += 1  matrixPerms.append(currentPerms)  matrixPerms = matrixPerms[1:]  matrixPerms = list(perms.chain.from\_iterable(matrixPerms))  matrixPerms = np.transpose(np.array(matrixPerms[:xlen]))  matrix = np.concatenate((matrixPerms, rMatrix), axis=1).astype(int)  return matrix, r  def syndrome(yn: str, xlen: int, matrix):  y = yn[:xlen]  yr = yn[xlen:]  yrnew = get\_xr(matrix, y)  return calc\_syndrome(yr, yrnew, xlen)  def calc\_syndrome(yr: str, yrnew: str, xlen: int):  return np.logical\_xor(yrnew, yr).astype(int) |

**Вывод**: в данной работе было рассмотрено перемежение и деперемежение как варианты кодирования информации. Было установлено, что такой способ должен использоваться в совокупности с другими корректирующими кодами, например кодом Хемминга. Перемежение/деперемежение позволяет разбить исходную строку, на строки нужной длины корректирующего кода, так чтобы можно было исправить как можно больше ошибок.