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 №10 НА ТЕМУ:**

**Сжатие/распаковка данных методом Лемпеля-Зива**

Ф.И.О.

Божко Денис Владимирович

Преподаватель

асс. Берников Владислав Олегович

Минск 2021

**Цель:** приобретение практических навыков использования метод Лемпеля − Зива (Lempel-Ziv) для сжатия/распаковки данных

**Теоретические сведения**

Известный метод сжатия RLE, который заключается в записи вместо последовательности одинаковых символов одного символа и их количества, является подклассом LZ77

Суть метода LZ77 (как и последующих его модификаций) состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также скользящим словарем (англ. sliding dictionary), варьируются в разных реализациях систем сжатия. Скользящее окно имеет длину n, т. е. в него помещается n символов, и состоит из двух частей:

• последовательности длины n1 = n − n2 уже закодированных

символов (словарь);

• упреждающего буфера (буфера предварительного просмотра, lookahead) длиной n2 – буфера кодирования.

Пусть к текущему моменту времени закодировано t символов:

S1, S2, ..., St. Тогда словарем будут являться n1 предшествующих символов: St − (n1 − 1), St − (n1 − 1)+1, …, St.

В буфере находятся ожидающие кодирования (сжатия) символы St+1, St+2, …, St+n2. Если n2 ≥ t, то словарем будет являться вся уже обработанная часть входной последовательности.

Нужно найти самое длинное совпадение между строкой буфера кодирования, начинающейся с символа St + 1, и всеми фразами словаря. После каждого шага окно смещается на q + 1 символов вправо и осуществляется переход к новому циклу кодирования. Величина сдвига объясняется тем, что мы реально закодировали именно q + 1 символов: q – с помощью указателя и 1 − с помощью тривиального копирования.

Передача одного символа в явном виде (s) позволяет разрешить проблему обработки еще ни разу не встречавшихся символов, но существенно увеличивает размер сжатого блока.

**Листинг кода**

|  |
| --- |
| import math  from bitarray import bitarray  class LZ77Compressor:  MAX\_WINDOW\_SIZE = 15  def \_\_init\_\_(self, window\_size):  self.window\_size = min(window\_size, self.MAX\_WINDOW\_SIZE)  self.lookahead\_buffer\_size = 5  def compress(self, input\_file\_path, output\_file\_path, verbose=True):  data = None  i = 0  output\_buffer = bitarray(endian='big')  try:  with open(input\_file\_path, 'rb') as input\_file:  data = input\_file.read()  except IOError:  print('Could not open input file ...')  raise  while i < len(data):  #print(i)  match = self.findLongestMatch(data, i)  #print(match)  if match:  (bestMatchDistance, bestMatchLength) = match  output\_buffer.append(True)  output\_buffer.frombytes(bytes([bestMatchDistance >> 4]))  output\_buffer.frombytes(bytes([((bestMatchDistance & 0xf) << 4) | bestMatchLength]))  if verbose:  print("<1, %i, %i>" % (bestMatchDistance, bestMatchLength), end='\n')  i += bestMatchLength  else:  output\_buffer.append(False)  output\_buffer.frombytes(bytes([data[i]]))    if verbose:  print("<0, %s>" % data[i], end='\n')  i += 1  output\_buffer.fill()  if output\_file\_path:  try:  with open(output\_file\_path, 'wb') as output\_file:  output\_file.write(output\_buffer.tobytes())  print("File was compressed successfully and saved to output path ...")  return None  except IOError:  print('Could not write to output file path. Please check if the path is correct ...')  raise  # an output file path was not provided, return the compressed data  return output\_buffer  def decompress(self, input\_file\_path, output\_file\_path=None,verbose = True):  data = bitarray(endian='big')  output\_buffer = []  # read the input file  try:  with open(input\_file\_path, 'rb') as input\_file:  data.fromfile(input\_file)  except IOError:  print('Could not open input file ...')  raise  while len(data) >= 9:  flag = data.pop(0)  if not flag:  byte = data[0:8].tobytes()  output\_buffer.append(byte)  del data[0:8]  else:  byte1 = ord(data[0:8].tobytes())  byte2 = ord(data[8:16].tobytes())  del data[0:16]  distance = (byte1 << 4) | (byte2 >> 4)  length = (byte2 & 0xf)  for i in range(length):  output\_buffer.append(output\_buffer[-distance])  out\_data = b''.join(output\_buffer)  if output\_file\_path:  try:  with open(output\_file\_path, 'wb') as output\_file:  output\_file.write(out\_data)  print('File was decompressed successfully and saved to output path ...')  return None  except IOError:  print('Could not write to output file path. Please check if the path is correct ...')  raise  if verbose:  print (out\_data)  return out\_data  def findLongestMatch(self, data, current\_position):  end\_of\_buffer = min(current\_position + self.lookahead\_buffer\_size, len(data) + 1)  best\_match\_distance = -1  best\_match\_length = -1    for j in range(current\_position + 2, end\_of\_buffer):  start\_index = max(0, current\_position - self.window\_size)  substring = data[current\_position:j]  for i in range(start\_index, current\_position):  repetitions = len(substring) // (current\_position - i)  last = len(substring) % (current\_position - i)  matched\_string = data[i:current\_position] \* repetitions + data[i:i+last]  if matched\_string == substring and len(substring) > best\_match\_length:  best\_match\_distance = current\_position - i  best\_match\_length = len(substring)  if best\_match\_distance > 0 and best\_match\_length > 0:  return (best\_match\_distance, best\_match\_length)  return None  compressor = LZ77Compressor(window\_size=8)  input\_file\_path = 'X:/BSTU/3 year/5 семестр/Защита инфы/лаб10/lab10/input.txt'  output\_file\_path = 'X:/BSTU/3 year/5 семестр/Защита инфы/лаб10/lab10/output.txt'  decompressed\_file\_path = 'X:/BSTU/3 year/5 семестр/Защита инфы/лаб10/lab10/decompressed.txt'  compressor.compress(input\_file\_path, output\_file\_path)  compressor.decompress(output\_file\_path,decompressed\_file\_path) |

**Вывод**: в данной работе был рассмотрен способ сжатия данных – метод Лемпеля – Зива. Данный метод позволяет представлять исходные данные на основе другого алфавита. Метод можно оптимизировать, варьируя параметры буфера и алфавита кодирования.