МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**РГЗ**

по дисциплине: Теория информации

Тема: «Моделирование канала передачи данных для источника Хартли с использованием кодера LZW в условиях воздействия помех»

Выполнил: студент группы ПВ-223

Романов М.

Проверил:

Твердохлеб В.В.

Белгород 2024 г.

**Ход работы:**

Источник. Был выбран источник Бернулли (источник с неравновероятными и статистически независимыми символами), что вероятности появления различных символов в сообщениях сильно отличаются, и появление одного символа не зависит от появления другого. Примером такого сообщения может быть последовательность символов, где один символ встречается значительно чаще, чем остальные, и их появление независимо друг от друга.

Ниже приведено математическое описание источников Бернулли, как алфавит в виде вероятностного ансамбля, а также условие монотонности:

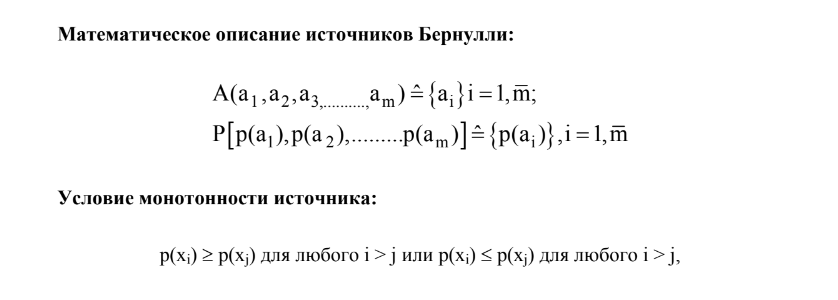


Рисунок 1. Математическое описание источников Бернулли.

Кодер. Используется алгоритм сжатия LZW, который работает без потери данных. Алгоритм LZW заменяет повторяющиеся последовательности символов ссылками, позволяя сокращать размер передаваемых данных. Другими словами, вместо отправки всех символов исходной последовательности кодер отправляет ссылки на те символы, которые уже встречались ранее.

Поскольку алгоритм LZW основан на использовании ссылок, он не требует передачи отдельной информации об используемом алфавите. Это делает его более эффективным для сжатия данных, особенно для текстовых файлов и изображений с повторяющимися элементами.

**Реализация:**

Код написан на языке Python. Описаны две функции lzw\_compress и lzw\_decompress.

Функция lzw\_compress (Листинг 1) представляет собой сжатие LZW для ASCII символов, после чего приводит массив чисел полученного от алгоритма сжатия к последовательности бит.

LZW алгоритм принимает символы ASCII – первоначальный словарь формируется из 128 символов. В процессе формируется словарь из повторяющихся отрезков символов. Словарь не ограничен по количеству полей поэтому особо большие сообщения следует кодировать с осторожностью. Для каждого символа c в сообщении строится строка wc путем конкатенации буфера buf и символа c. Проверяется, существует ли в словаре запись с ключом wc. Если да, буфер buf обновляется значением wc. Если нет, добавляется новая запись в словарь с ключом wc и значением, равным размеру словаря. Буфер buf обновляется первым символом сообщения. Результат сжатия пополняется кодом символа из словаря, соответствующим буферу buf.

def lzw\_compress(message):  
 dictionary = {chr(i): i for i in range(256)}  
 result = []  
 buf = ""  
 for c in message:  
 wc = buf + c  
 if wc in dictionary:  
 buf = wc  
 else:  
 result.append(dictionary[buf])  
 dictionary[wc] = len(dictionary)  
 buf = c  
 if buf:  
 result.append(dictionary[buf])  
  
 return result

Листинг 1. Функция lzw\_compress.

Для принимающей стороны реализован метод lzw\_decompress (Листинг 2), который совершает обратные действия. Принимая размер символа равным 7 и динамически его увеличивая по ходу формирования новых слов. Для каждого следующего сжатого кода k проверяется, существует ли в словаре запись с ключом k. Если да, буфер buf пополняется соответствующим символом из словаря. Если нет, и k равен размеру словаря, буфер buf пополняется конкатенацией prev\_buf и первого символа prev\_buf. Результат декомпрессии пополняется символами из buf. В словаре создается новая запись с ключом, равным текущему размеру словаря, и значением, равным конкатенации prev\_buf и первого символа buf. Буфер prev\_buf обновляется значением buf.

def lzw\_decompress(compressed):  
 dictionary = {i: chr(i) for i in range(256)}  
 result = ""  
 prev\_buf = chr(compressed.pop(0))  
 result += prev\_buf  
 for k in compressed:  
 buf = ""  
 if k in dictionary:  
 buf = dictionary[k]  
 elif k == len(dictionary):  
 buf = prev\_buf + prev\_buf[0]  
 result += buf  
 dictionary[len(dictionary)] = prev\_buf + buf[0]  
 prev\_buf = buf  
 return result

Листинг 2. Функция lzw\_decompress.

Ниже представлен пример работы алгоритма (Рисунок 2).

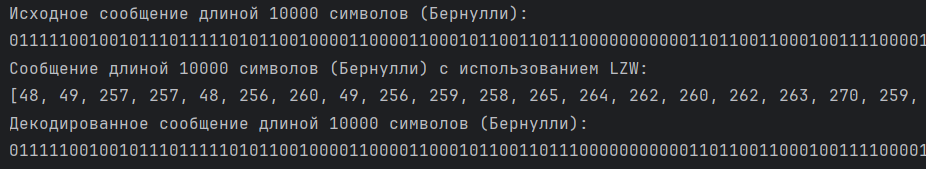


Рисунок 2. Пример работы алгоритма.

Данный пример имеет следующие параметры: исходное сообщение длинной 10000.

Кодированное сообщение получилось размером 7315 символов, что получилось меньше исходного.

После кодирования можно увидеть как закодированное сообщение было успешно перекодированно в исходное.

**Список литературы:**

1. [Алгоритм Лемпеля — Зива — Велча — Википедия [Электронный ресурс]. Режим доступа: wikipedia.org](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D0%B5%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F_%E2%80%94_%D0%97%D0%B8%D0%B2%D0%B0_%E2%80%94_%D0%92%D0%B5%D0%BB%D1%87%D0%B0#:~:text=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%CC%81%D1%82%D0%BC%20%D0%9B%D0%B5%CC%81%D0%BC%D0%BF%D0%B5%D0%BB%D1%8F%20%E2%80%94%20%D0%97%D0%B8%CC%81%D0%B2%D0%B0%20%E2%80%94%20%D0%A3%D1%8D%D0%BB%D1%87%D0%B0%20(,%D0%B8%20%D0%97%D0%B8%D0%B2%D0%BE%D0%BC%20%D0%B2%201978%20%D0%B3%D0%BE%D0%B4%D1%83.) (Дата обращения 01.06.2024)
2. [Распределение Бернулли — Википедия [Электронный ресурс]. Режим доступа: wikipedia.org](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%91%D0%B5%D1%80%D0%BD%D1%83%D0%BB%D0%BB%D0%B8) (Дата обращения 02.06.2024)

**Вывод**: в ходе работы исследованы источник Бернулли и кодирование Лемпеля-Зива-Велча. Построена математическая модель источника Бернулли. Разработана и протестированна программная реализация алгоритма кодирования и декодирования Лемпеля-Зива-Велча.