УДК 004

# Разработка алгоритма сжатия данных RLE на языке программирования Python

*Андриенко Иван Сергеевич*
*Приамурский государственный университет имени Шолом-Алейхема*
*Студент*

**Аннотация**
В статье описывается работа и реализация преобразования Run-Length алгоритма на языке программирования Python. Продемонстрирован пример кодирования и декодирования строки. Результатом статьи является объяснение работы алгоритма и его практическая реализация.
**Ключевые слова:** Python, сжатие данных, алгоритм, Run-Length.

# Development of the RLE data compression algorithm in the Python programming language

*Andrienko Ivan Sergeevich*
*Sholom-Aleichem Priamursky State University*
*Student*

**Abstract**
The article describes the operation and implementation of the Run-Length transformation algorithm in the Python programming language. An example of encoding and decoding a string is demonstrated. The result of the article is an explanation of the algorithm and its practical implementation.
**Keywords:** Python, data compression, Algorithm, Run-Length.

## 1 Введение

### 1.1 Актуальность

На сегодняшний день сжатие файлов – актуальная проблема из-за все большего потребления информации пользователями. Сжатие файла позволяет уменьшить его размер, при этом, сохранив исходные данные. При использовании сжатия файл будет занимать намного меньше места на устройстве. Это также облегчает хранение и передачу сжатых файлов через интернет или другим способом.

### 1.2 Обзор исследований

Д.А. Автушко, А.Я. Скляр дали краткий обзор существующих алгоритмов сжатия без потерь. Предложена модель арифметического кодирования и алгоритм сжатия данных на основе прогнозирования поступления очередного символа во входном потоке средствами нейронной сети. Приведены результаты работы алгоритма [1]. И.В. Ковалёва и

А.Н.Размахнина описали процесс реализации алгоритма преобразования Барроуза-Уилера на языке программирования C++ [2]. Н.М. Беспалова, Л.А.Сологубова, Ф.Н. Байбекова в своей работе рассмотрели вопросы об основных алгоритмах сжатия данных [3]. В статье Х.М. Котиева описала как работает сжатие данных без потерь. Также был рассмотрен самый распространенный метод сжатия без потерь - алгоритм Хаффмана [4]. А.Р.Муртазина, В.П. Миронов, И.Б. Разин рассмотрели основные требования к алгоритмам сжатия изображений. Математически описан и создан алгоритм компрессии контуров, заданных в виде последовательности точек. Проверена работа алгоритма на разных деталях обуви и сделана оценка результатов сжатия [5].

### 1.3 Цель исследования

Цель исследования – создать программу, которая будет кодировать и декодировать строку с помощью алгоритма Run-Length.

## 2 Материалы и методы

Для практической реализации алгоритма использовалась среда программирования PyCharm и язык программирования Python.

## 3 Результаты и обсуждения

Кодирование длины выполнения в python – это алгоритм, который заменяет значения внутри повторяющейся строки, подсчитывает количество похожих символов и вместо того, чтобы отправлять их несколько раз, отправляет их вместе с количеством символов. Алгоритм RLE хорошо работает с изображениями, в которых есть большие одноцветные области. Для каждого символа с аналогичным значением его представляют только два символа – его значение и его количество.

Создадим функцию для кодирования. Имя определенной функции – run_length_encoding. Она принимает один аргумент, который является последовательностью, подлежащей сжатию. Создаем пустой список «compressed», к которому будут добавляться вложенные списки. Затем берем переменную счетчика с именем «count», которая будет подсчитывать вхождения символов. Ее значение равно 1. Строка с именем «char» будет использоваться для хранения предыдущего символа, и будет сравнивать текущий символ с «char». Изначально значение переменной «char» является первым символом последовательности. Используя условие if, сравним, равен ли текущий символ символу, хранящемуся в переменной «char». Если равен, то это означает, что два последовательных символа имеют одинаковое значение, поэтому увеличиваем значение «count». Количество будет увеличиваться до тех пор, пока два последовательных символа не станут равными. Как только это условие не выполняется, программа перейдет к другой части. Таким образом, он добавит вложенный список к основному списку. Вложенный список должен содержать символ и его количество. В конце возвращаем список (рис. 1).

```python
def run_length_encoding(seq):
    compressed = []
    count = 1
    char = seq[0]
    for i in range(1, len(seq)):
        if seq[i] == char:
            count = count + 1
        else:
            compressed.append([char, count])
            char = seq[i]
            count = 1
    compressed.append([char, count])
    return compressed
```

Рисунок 1 – Функция для кодирования

Далее помещаем строку, которую необходимо закодировать, в функцию. Вызываем функцию, передаем ей строку и сохраняем выходные данные в переменную «list1» (рис.2).

```python
seq = 'DFFMMMAADF'
list1 = run_length_encoding(seq)
```

Рисунок 2 – Вызов функции

Создадим переменную compressed_seq с нулевым значением для вывода шифра. С помощью цикла for печатаем каждый вложенный список и сохраняем его значение в виде строки (рис. 3). Выводим полученный шифр (рис. 4).

```python
for i in range(0, len(list1)):
    for j in list1[i]:
        compressed_seq += str(j)

print(compressed_seq)
```

Рисунок 3 – Сохранение шифра

```
Изначальная строка:DDDDFFMMMAADDDFFFFEEEEBBBBCCCCCCAAA
Шифр:D4F2M3A2D3F4E4B4C6A3

Process finished with exit code 0
```

Рисунок 4 – Вывод шифра

Создадим код для декодирования. Определим функцию с именем «run_length_decoding» для выполнения декодирования. Функция примет один аргумент «compressed_seq», который будет последовательностью в ее сжатом виде. Берем локальную нулевую строку с именем «seq» и запускаем цикл for по длине сжатой последовательности. Также проверим, является ли текущий символ алфавитом или нет с помощью метода «isalpha».

Поскольку сжатое представление содержит сначала символ, а затем его количество, сначала проверяем наличие символа алфавита. Если это алфавит, то следующим символом будет его количество. Далее запускаем еще один цикл for до тех пор, пока не получим значение счетчика. В конце возвращаем строку «seq». Для вывода декодированного шифра вызовем функцию run_length_decoding() и передадим сжатую строку в качестве аргумента этой функции (рис. 6).

```
30  def run_length_decoding(compressed_seq):
31      seq = ''
32      for i in range(0, len(compressed_seq)):
33          if compressed_seq[i].isalpha() == True:
34              for j in range(int(compressed_seq[i + 1])):
35                  seq += compressed_seq[i]
36      return (seq)
37
38
39  print(run_length_decoding(compressed_seq))
40
```

Рисунок 6 – Код декодирования шифра

Запускаем программу и проверяем декодирование (рис. 7).

```
Изначальная строка:DDDDFFMMMAADDDFFFFEEEEBBBBCCCCCCAAA
Шифр:D4F2M3A2D3F4E4B4C6A3
Декодирование:DDDDFFMMMAADDDFFFFEEEEBBBBCCCCCCAAA

Process finished with exit code 0
```

Рисунок 7 – Вывод декодированного шифра

**Выводы**

В данной работе было описано создание Run-Length алгоритма. Продемонстрирован пример кодирования и декодирования строки.

**Библиографический список**

1. Автушко Д.А., Скляр А.Я. Алгоритм нейросетевого сжатия данных без потерь. // Тенденции развития науки и образования. 2022. № 87-1. С. 7-12.
2. Ковалёва И.В., Размахнина А.Н. Реализация преобразования БарроузаУилера с помощью языка С++ // Постулат. 2016. № 12. С. 56
3. Беспалова Н.М., Сологубова Л.А., Байбекова Ф.Н. Обзор основных алгоритмов сжатия данных. // Информационная безопасность - актуальная проблема современности. Совершенствование образовательных технологий подготовки специалистов в области информационной безопасности. 2019. № 1 (10). С. 145-148.
4. Котиева Х.М. Сжатие данных без потерь. Использование алгоритма Хаффмана. // Молодой ученый. 2020. № 35 (325). С. 12-14.
5. Муртазина А.Р., Миронов В.П., Разин И.Б. Алгоритм сжатия данных. // Дизайн и технологии. 2014. № 43 (85). С. 51-55.