МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

ОТЧЕТ
по производственной практике

по теме:
АЛГОРИТМЫ СЖАТИЯ С ПОТЕРЯМИ

Студент:
Группа № 5140901/31501 *Д.С. Кузик*

Руководитель:
доцент *К.К. Семенов*

Санкт-Петербург
2024

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Задача 9.

Выполнить анализ алгоритмов сжатия с потерями KRLE (K-Run Length Encoding), LTC (Lightweight Coding) и метрологического JPEG. Выполнить описание данных алгоритмов. Реализовать данные алгоритмы в пакете Matlab и на языке Python. Выполнить тестирование. Выполнить сравнительный анализ.

2 ОПИСАНИЕ АЛГОРИТМОВ

2.1 K-RLE

В контексте использования технологии беспроводной сенсорной сети (WSN) для мониторинга окружающей среды двумя основными элементарными функциями WSN являются сбор и передача данных. Однако передача/прием данных требует больших затрат энергии. Чтобы снизить энергопотребление, связанное с передачей, используется сжатие данных с помощью локальной обработки информации.

Рассмотрим новый алгоритм сжатия данных, основанный на кодировании длины выполнения (RLE), который называется K-RLE.

Идея, лежащая в основе этого нового алгоритма, заключается в следующем: пусть K - число, если элемент данных d, $d+K$ или $d-K$ встречается n раз подряд во входном потоке, мы заменяем эти n вхождений одной парой nd [1].

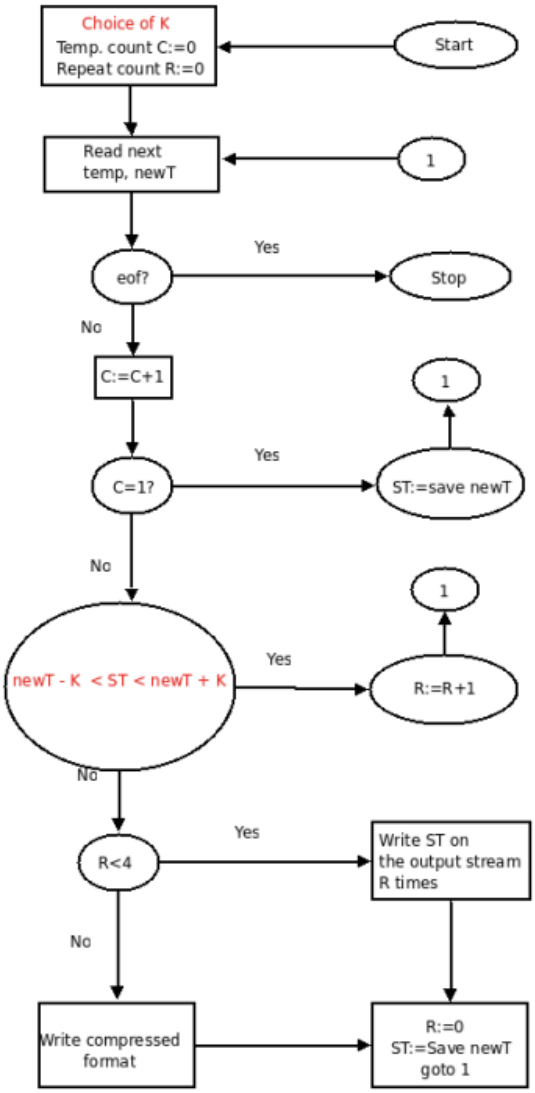

Рисунок 1 — K-RLE Алгоритм

3 РАЗРАБОТАННЫЙ КОД

3.1 K-RLE

Представим разработанные алгоритмы кодирования и декодирования K-RLE на языке программирования Python:

```
1   from typing import List
2
3   def k_rle_code(stream: List[int], K: int) -> List[int]:
4       # Функция для кодирования потока чисел с использованием модифицированного RLE алгоритма.
5       #
6       # Алгоритм:
7       # 1. Проходим по входному потоку чисел.
8       # 2. Если элемент повторяется (учитывая допустимое отклонение K), увеличиваем счетчик повторов.
9       # 3. Если элемент не повторяется, вставляем текущие накопленные повторы в результат.
10      # 4. В конце вставляем оставшиеся элементы после завершения цикла.
11      #
12      # Аргументы:
13      # stream – входной поток чисел
14      # K – допустимое отклонение для повторяющихся чисел
15      #
16      # Возвращает:
17      # Закодированный поток чисел
18
19      result_stream = []
20      count = 0
21      repeat_count = 0
22      ST = 0
23
24      def insert_func():
25          if repeat_count < 4:
26              result_stream.extend([ST] * (repeat_count + 1))
27          else:
28              result_stream.extend([repeat_count, ST])
29
30      for i, newT in enumerate(stream):
31          count += 1
32          if count == 1:
33              ST = newT
34              continue
35          if (newT + K > ST and newT - K < ST) or newT == ST:
36              repeat_count += 1
37              continue
38          insert_func()
39          repeat_count = 0
40          ST = newT
41      insert_func()
42
43      return result_stream
44
45  def k_rle_decode(stream: List[int], threshold: int) -> List[int]:
46      # Функция для декодирования потока чисел, закодированного модифицированным RLE алгоритмом.
47      #
48      # Алгоритм:
49      # 1. Проходим по входному закодированному потоку чисел.
```

```
50          # 2. Если элемент меньше порога и больше 0, считаем его как количество повторов и добавляем соответствующие элементы в результат.
51          # 3. Если элемент не является счетчиком повторов, просто добавляем его в результат.
52          # 4. Переходим к следующему элементу.
53          #
54          # Аргументы:
55          # stream – входной закодированный поток чисел
56          # threshold – порог для определения счетчиков повторов
57          #
58          # Возвращает:
59          # Декодированный поток чисел
60
61          result_stream = []
62          i = 0
63          while i < len(stream):
64              if stream[i] < threshold and stream[i] > 0:
65                  result_stream.extend([stream[i+1]] * (stream[i] + 1))
66                  i += 2
67              else:
68                  current = stream[i]
69                  result_stream.append(current)
70                  i += 1
71          return result_stream
```

Листинг 3.1 — K-RLE реализация на языке Python

4 ТЕСТИРОВАНИЕ АЛГОРИТМОВ

5 АНАЛИЗ РЕЗУЛЬТАТОВ И ВЫВОДЫ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Pamba Capo-Chichi E.*, *Guyennet H.*, *Friedt J.* K-RLE: A new Data Compression Algorithm for Wireless Sensor Network //. — 07/2009. — P. 502–507.