УДК 004

Создание алгоритма сжатия текстовых данных на языке программирования Python

Звайгзне Алексей Юрьевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье реализуется алгоритм, позволяющий сжимать объем конечного сообщения путем подсчета одинаковых символов по порядку аналогично алгоритму RLE (кодирование длин серий) на языке программирования Python, для этого создается анонимный класс, в котором используя логические операции для считывания ввода из строки, перевод, подсчитываются символы по порядку и в качестве выходных данных кодируется новая строка короче предыдущей или выводится изначальное значение ввода.
Ключевые слова: Python, алгоритм, кодирование, сжатие, цикл.

Creating a text data compression algorithm in the Python programming language

Zvaigzne Alexey Yurievich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article implements an algorithm that allows compressing the volume of the final message by counting identical characters in order, similar to the RLE algorithm (encoding of series lengths) in the Python programming language, for this an anonymous class is created in which, using logical operations to read input from a string, translation, characters are counted in order and a new string is encoded as output data shorter than the previous one, or the original input value is output.
Keywords: Python, algorithm, encoding, compression, cycle

1 Введение

1.1 Актуальность

В связи с постоянной передачей и обменом большого объема данных есть необходимость сжатия данных, позволяющая сохранить изначальное значение данных или сохранить при минимальной потере данных. Самым простым алгоритмом сжатия является – кодирование длин серий, он заменяет серию из двух и более повторяющихся чисел или символов числом, обозначающим количество повторений числа или символа, после

которого идет данный символ или число. Полезен пережатии большого числа повторений и до сих пор используется в комбинации с другими алгоритмами сжатия.

1.2 Обзор исследований

В. Голованов в своей статье рассказывает о некоторых алгоритмах сжатия с потерями и без них, их особенности [1]. В научной статье Д.Мастрюков, объясняет принципы работы различных методов сжатия данных, рассказывает о возможных проблемах, которые могут появиться при реализации [2]. Д. В. Дружинников в сборнике трудов научной конференции пытается улучшить имеющийся алгоритм кодирования данных UTF-8, путем сжатия различными реализациями популярных алгоритмов [3]. И.И. Киамов в научном журнале провел сравнительный анализ существующих алгоритмов, подобрав оптимальный для разного объема и содержания текста [4]. Е.А.Момот в своей статье отображает процесс разработки ПО выборочного сжатия рукописных данных с изображениями [5].

1.3 Цель исследования

Реализация данного алгоритма на Python может пригодиться для создания более мощного инструмента сжатия. Целью данной статьи является реализация алгоритма на языке программирования Python.

2 Материалы и методы

Алгоритм реализован на языке программирования Python, создан анонимный класс, проверяющий каждый символ в строке и выдающий на выходе новую закодированную строку.

3 Результаты и обсуждения

Для реализации собственного алгоритма сжатия данных требуется создать анонимный класс(def), присвоить ему имя - compress_string(), а в качестве аргумента указать line:

```
def compress_string(line):
```

После этого создать переменную и передать ей пустое значение, для подсчета количества символов необходимо сделать счетчик, указывающий количество повторяющихся рядом символов count и прировнять его 1:

```
new_string = “”
count = 1
```

Так как по умолчанию в строке должен содержаться хотя бы один символ. Для начала счета символов надо открыть цикл for, в качестве

переменной использовать i и повторять его пока не закончится строка. Просматривая каждый символ или индекс:

```
for i in range(len(line)):
```

В цикле нужно создать новую переменную j обозначающая новый мнимый символ, стоящий рядом с i, который существует пока есть символы в строке, его нужно поместить рядом с i приравняв его переменной i и прибавлять к нему единицу от текущего, так необходимо делать для того, чтобы получить конечное значения количества данной буквы:

```
j = i
j += 1
```

Чтобы после проверки последнего символа интерпретатор не выдавал ошибки, необходимо следующий цикл поместить в исключение try/except и уже в него закладывать проверку символов. Новый цикл также начинается с если(if) проверяется строка с переменной i, сравнивая её со строкой с переменной j. Если i равно j, то необходимо к счетчику count прибавить единицу:

```
try:
if line[i] == line [j]:
```

Иначе (else) нужно положить эту букву в новую строку присвоив текущее значение строки i. Теперь нужно отследить количество циклов, но так как это число, переменную необходимо перевести в число с помощью команды str указав в качестве аргумента счетчик (count), а в качестве значения передать 1, чтобы значение:

```
else:
new_string += line[i]
new_string += str(count)
count = 1
```

После обработки попытки, надо указать какое-либо исключение except, для этого необходимо поместить проверку последнего символа или индекса. Начать надо добавления строки i, а мнимую переменную j приравнять к i и вычесть 1:

```
except:
new_string += line[i]
j = i
j -= 1
```

Если (if) строка i равна строке j, то нужно приписать новой строке значение count используя команду str, иначе (else) к новой строке нужно приписать 1 вручную единицу:

```
else:
count += "1"
```

Осталось создать цикл проверяющий, если (if) оригинальная строка короче полученной, то алгоритм просто вернет оригинальную строку, не проводя с ней итераций с помощью команды len, иначе будет выводиться сжатая строка из переменной new_string:

```
if len(new_string) > len(line):
print(line)
else:
print(new_string)
```

Осталось открыть строку ввода и в случае, если пользователь введет строку выдать обратно сжатую или оригинальную строку.

Код:

```
def compress_string(line):
  new_string = ""
  count = 1
  for i in range(len(line)):
    j = i
    j += 1
    try:
      if line[i] == line [j]:
        count += 1
      else:
        new_string += line[i]
        new_string += str(count)
        count = 1
    except:
      new_string += line[i]
      j = i
      j -= 1
      if line[i] == line[j]:
        new_string += str(count)
      else:
          count += "1"

  if len(new_string) > len(line):
    print(line)
```

```
else:
  print(new_string)

  string = input("Введите строку с повторяющимися символами\n")
compress_string(string)
```

Так как Python чувствителен к положению написанию вложенного кода надо внимательно следить за его написанием, в противном случае при компиляции можно получить ошибку (рис. 1).

```
def compress_string(line):
  new_string = ""
  count = 1
  for i in range(len(line)):
    j = i
    j += 1
    try:
      if line[i] == line [j]:
        count += 1
      else:
        new_string += line[i]
        new_string += str(count)
        count = 1
    except:
      new_string += line[i]
      j = i
      j -= 1
      if line[i] == line[j]:
        new_string += str(count)
      else:
          count += "1"

  if len(new_string) > len(line):
    print(line)
  else:
    print(new_string)

string = input("Введите строку с повторяющимися символами\n")
compress_string(string)
```

Рисунок 1. Конечный вид кода

После компиляции, должна строка вывода должна быть доступна для ввода (рис. 2).

```
def compress_string(line):
  new_string = ""
  count = 1
  for i in range(len(line)):
    j = i
    j += 1
    try:
      if line[i] == line [j]:
        count += 1
      else:
        new_string += line[i]
        new_string += str(count)
        count = 1
    except:
      new_string += line[i]
      j = i
      j -= 1
      if line[i] == line[j]:
        new_string += str(count)
      else:
          count += "1"

  if len(new_string) > len(line):
    print(line)
  else:
    print(new_string)

string = input("Введите строку с повторяющимися символами\n")
compress_string(string)
...  Введите строку с повторяющимися символами
```

Рисунок 2. Вид скомпилированного кода

Для проверки, можно ввести строку с повторяющимися символами например, «ааааааабббббб» и посмотреть конечный вывод (рис. 3).

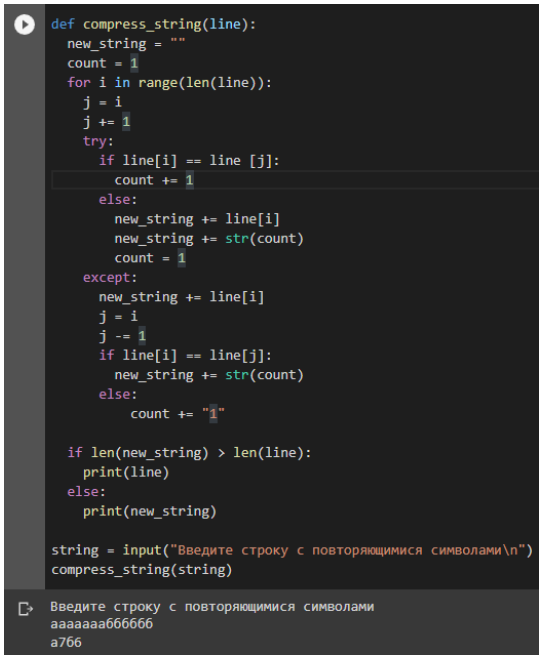

Рисунок 3. Конечный вывод кода.

Конечный вывод содержит введеную строку и сжатый код.

Выводы

Данный метод сжатия может спокойно работать с однородным вводом данных, например, буквы на латинице или кириллице в кодировке UTF-8, обрабатывать числа, но из-за проверки, если в поле ввода указать смешанный ввод из букв и чисел, то алгоритм выдаст ошибку.

Также алгоритм выдаст ошибку, если в качестве ввода указать определенное количество неуникальных чисел, а в конце этих чисел указать их суммарное число указанных символов, которое должен был указать алгоритм, так алгоритм проверяет есть ли это значение, то он его подставляет.

При этом алгоритм для подсчета однородных значений алгоритм не ограничен в длине ввода содержимого для проверки было введено сообщение длинною 406560 символов, а вычисление количества символов заняло порядка четырех секунд.

Библиографический список

1. Голованов В. Алгоритмы сжатия данных без потерь, часть 2 URL: https://habr.com/ru/post/251295/ (дата обращения 17.01.2023).
2. Мастрюков Д. Алгоритмы сжатия информации // Монитор. 1993. №. 7-8. С. 14-20.
3. Коркмасов Д. Д. Анализ алгоритмов сжатия текста в юникоде // Модели

инновационных решений повышения конкурентоспособности отечественной науки. 2020. С. 74-79.
4. Киамов И. И., Галимуллина Г. М. Сравнительный анализ алгоритмов сжатия // Тенденции и закономерности развития современного российского общества: экономика, политика, социально-культурная и правовая сферы. 2016. С. 29-29.
5. Момот Е. А. О некоторых алгоритмах сжатия с потерями фотографий печатного и рукописного текста (проект" kotspect") // Вестник факультета прикладной математики, информатики и механики: Сборник статей. Воронеж: Воронежский государственный университет, 2021. С. 124-132