УДК 004

Реализация алгоритма сжатия данных Лемпеля – Зива LZ77 на языке программирования Python

Эрдман Александр Алексеевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Научный руководитель:
Лучанинов Дмитрий Васильевич
Приамурский государственный университет имени Шолом-Алейхема
Старший преподаватель кафедры информационных систем, математики и правовой информатики

Аннотация
В статье рассмотрен алгоритм сжатия данных Лемпеля – Зива LZ77 и его реализация на практике. Исполнение алгоритма выполнено на языке программирования Python. Результатом статьи является объяснение работы алгоритма и его практическая реализация.
Ключевые слова: python, сжатие данных, алгоритм, LZ77

I Implementation of the Lempel – Ziv LZ77 data compression algorithm in the Python programming language

Erdman Alexander Alekseevich
Sholom-Aleichem Priamursky State University
Student

Scientific supervisor: Luchaninov Dmitry Vasilyevich
Sholom-Aleichem Priamursky State University
Senior lecturer of the Department of Information Systems, Mathematics and Legal Informatics

Abstract
The article discusses the Lempel–Ziv LZ77 data compression algorithm and its implementation in practice. The algorithm is executed in the Python programming language. The result of the article is an explanation of the algorithm and its practical implementation.
Keywords: python, data compression, algorithm, LZ77

1 Введение

1.1 Актуальность

На начало 21 века технологии сделали огромный шаг вперёд по сравнению с концом предыдущего века. Удивительно, что компьютер был

изобретён всего несколько десятков лет назад, но как сильно он изменился в плане вычислительных возможностей. На ровне с ростом вычислительной мощности, активно развивались новые виды данных такие, как аудио и видео данные, а также изображения. Эти данные становились с каждым годом всё более продвинутыми, у них у всех улучшались характеристики – от чёрно-белых картинок и видео, до цветных с огромным количеством передаваемых цветов. На фоне роста качества данных, соответственно рос их размер – когда-то нормальным занимаемым местом файлов считались биты, потом байты, а на сегодняшний день нормой считаются уже гигабайты данных. Можно предположить, что чем дальше идёт развитие индустрия технологий, а значит качество данных, и как следствие занимаемое ими пространство на носителях. Чтобы избежать чрезмерно большого размера файла, были созданы алгоритмы сжатия данных, которые позволяет уменьшать размер файлов как незначительно, так и довольно существенно. Алгоритмов сжатия данных существует множество и они активно развиваются по сегодняшний день. Одним из таких алгоритмов сжатия данных является алгоритм Лемпеля – Зива LZ77. О данном алгоритме пойдёт речь в статье.

1.2 Обзор исследований

Д.В. Фатеенко описал работу и реализацую преобразования Барроуза-Уилера (BWT) и Run-Length Encode алгоритма на языке программирования C++ [1]. И.В. Ковалева, А.Н. Размахнина и Д.В. Лучанинов описали реализацию алгоритма преобразования Барроуза-Уиллера на языке программирования C++ [2]. А.О. Кизянов показал процесс реализации алгоритма Хаффмана на языке программирования Python [3]. А.В. Ленкин и Д.В. Лучанинов описали реализацию алгоритма «Стопка книг» с помощью языка программирования C++ [4]. Б.Я. Рябко описала новую схему организации сжатия данных с помощью «мнимого скользящего окна» [5]. Н.М. Беспалова, Л.А. Сологубовал и Ф.Н. Байбекова рассмотрели вопросы об основных алгоритмах сжатия данных [6].

1.3 Цель исследования

Целью исследование является изучение алгоритма Лемпеля – Зива LZ77 и его реализация на языке программирования Python.

2 Материалы и методы

Для изучения алгоритма Лемпеля – Зива LZ77 использовалась соответствующая литература. Для практической реализации алгоритма использовалась среда программирования PyCharm и язык программирования Python.

3 Результаты и обсуждения

Сжатие данных – это обратимые алгоритмические преобразования данных, результатом которых является уменьшение занимаемого объёма данных на носителе. Существует несколько видов сжатия данных - сжатие

без потерь или полностью обратимое и сжатие с потерями, кода данные не сохраняют свою целостность после сжатия. Алгоритм LZ77 относится к первому типу, то есть не нарушает целостность данных после сжатия. Также алгоритм LZ77 относится к алгоритмам корреляционного анализа для поиска корреляций, а именно точных повтор между участками данных и их заменой на коды, которые позволяют восстановить данные на основе предшествующих данных. LZ77 является частью группы методов Лемпеля-Зива (словарные алгоритмы), которые имеют общую особенность сжатия – кодирование не количество повторов символов, а ранее встречавшиеся последовательности символов.

Принцип работы алгоритма LZ77 следующий - это замена повторяющей части строки вхождения ссылкой(кодом) на одну из предыдущих позиций вхождения. Для реализации алгоритма используется метод скользящего окна. Скользящее окно представляется в виде динамической структуры данных - сохраняет повторяющуюся информацию и предоставлять к ней доступ. Окно имеет свой заданный размер - 16 или 32 кБайт. Это обусловлено очевидными ограничениями по размерам ОЗУ и необходимостью ограничить время поиска фрагмента в буфере. Само окно представлено в виде двух частей. Первая часть большая по размеру, включает уже просмотренную часть сообщения. Вторая часть намного меньше, является буфером, содержащим еще незакодированные символы входного потока. Алгоритм пытается найти в словаре (большей части окна) фрагмент, совпадающий с содержимым буфера. В результате работы алгоритма на выходе получаются сжатые данные за счёт того, что части элементов данных представлены кодом алгоритма LZ77. Декодирование кодов LZ77 проще их получения, т.к. не нужно осуществлять поиск в словаре.

После разбора принципа работы алгоритма, необходимо воссоздать данный алгоритм сжатия LZ77 на практике. Практические результаты должны будут показать, как выглядят сжатые данные и насколько они будут сжаты. Данные представлены текстом.

Текст выбран так, чтобы в нём присутствовали повторяющие слова – «two beer or not two beer (not Hamlet)». Данные слова будут занесены в переменную src. Создаётся пустая переменная pack, а также i = 0. Создаётся шаблон упаковщика при помощи команды цикла while. Шаблон упаковщика получает один символ из исходного текста и без изменений записывает в переменную pack. Переменная i будет указывать на номер символа в исходной строке. Данная переменная будет инкрементироваться, пока не превысит размер строки src. Для наглядности необходимо выводить исходный текст, сжатый и распакованный. Для начала будет выводиться исходный текст и сжатый при помощи команды print:

```
src = "two beer or not two beer (not Hamlet)"
print('src:    ' + src)
pack = ''
i = 0
```

```
while i < len(src):
        pack += src[i]
        i += 1
print('pack:   ' + pack)
```

По такому же принципу создаётся распаковщик, то есть каждый символ без изменений отправляется в переменную unpack:

```
while i < len(pack):
        pack += pack[i]
        i += 1
print('unpack:   ' + unpack)
```

В результате сделаны шаблоны для упаковщика и распаковщика текста. На данный момент они одинаковый исходный текст. Для продвижения дальше знать, как получать фрагмент строки. Для получения фрагмента строки используется определённая команда. Допустим в строке нужно взять фрагмент, который начинается с i =16 элемента и после нужно выбрать фрагмент длиной ln = 4 элемента. То есть фрагмент состоит из 4 символом после 16 элемента строки (рис. 1).

```
>>> src = "two beer or not two beer (not Hamlet)"
>>> i = 16
>>> ln = 4
>>> src[i:i+4]
'two '
>>> src[i:i+ln]
'two '
```

Рисунок 1. Метод получения фрагмента строки

Метод получения фрагмента строки необходим, так как алгоритм LZ77 основан на поиске похожих фрагментов. Для упрощения будет сначала использоваться длина фрагмента ln = 4. В цикл while помещается конструкция if, в которой сравнивается два фрагмента в позициях i и j. Позиция j указывает на фрагмент, который находится до фрагмента i. Таким образом вместо фрагментов позиции i будут вставляться ссылку на позицию j. Позиция j устанавливается на ln = 4 ранее i, что предотвращает пересечение фрагментов. Добавляется новый цикл while в имеющийся. В условие прописывается неравенство j > = 0, так как в первые несколько итераций переменная j будет отрицательной. Если совпадений не будет найдено, то идёт уменьшение j на единицу, то есть идёт поиск похожего фрагмента ещё ранее до тех пор, пока не будет упор в начало текста. Создаётся переменная pack, которая содержит вид кодировки ссылки(кода). Кодировка имеет вид «’#$2d’ % i - j», то есть ссылка(код) обозначится решёткой, после которой

идёт ещё цифры, которые указывают расстояние от ссылки до текущей позиции, где было найдено совпадение, то есть разница между i и j. Добавляется переменная found, которая будет сообщать о том, было ли найдено совпадение. Если в конце поиска found = false, то на выходе посылается символ о текущей позиции в переменной i (рис. 2).

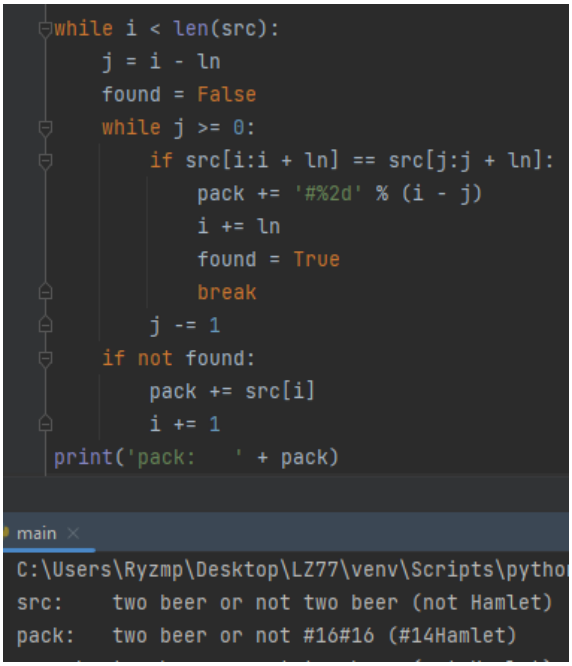

Рисунок 2. Работа упаковщика. Кодирование совпадений ссылкой на схожий фрагмент

Дорабатывание распаковщика происходит следующим образом. Если распаковщик встречает символ решётку, то идёт определение ссылки. Используется оператор continue, так как позволяет начинать следующий проход цикла, минуя оставшееся тело цикла. После решётки распаковщику необходимо извлечь два символа, сконвертировать в int. Это будет расстояние до фрагмента (хранится в переменной dist), который необходимо скопировать. Для избежания бесконечного цикла, позиция курсора увеличивается на i += 3 (один символ — это решётка и два символа расстояние). В случае, если распаковщик выполнит распаковку неправильно, то есть распакованный текст будет отличаться от исходного, тогда будет предупреждение (рис. 3). Алгоритм работает с длинного фрагмента ln =4. Лучше найти максимально длинный фрагмент, так как сжатие будет лучше. Начнётся поиск с ln = 9 символов, если не найдётся такой длины, то будет искаться с длиной ln = 8 символом и так далее. Совпадение ln <4 символом не имеет смысла, так как уменьшится коэффициент сжатия. Изменяется кодирование – добавляется один символ для указания длины фрагмента «1d» (меняется параметры переменной pack).

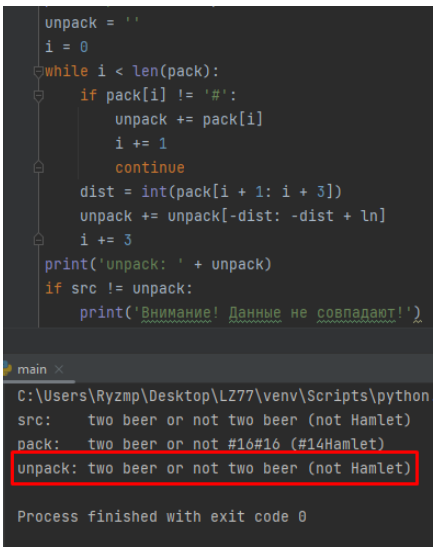

Рисунок 3. Работа распаковщика. Обнаружение решётки и переход по ссылкам, для восстановления фрагмента

Чтобы избежать бесконечного цикла, необходимо уменьшить поисковую длину фрагмента ln -=1. Проверка работы (рис. 4).

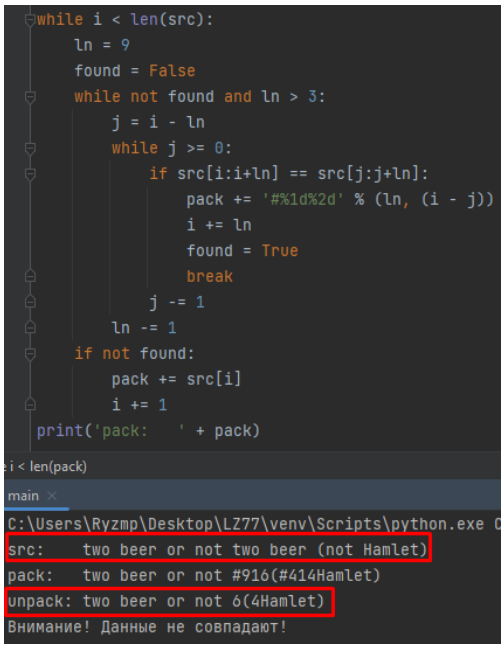

Рисунок 4. Проверка работы упаковщика

Видно, что упаковщик нашёл фрагмент текст длинной ln = 9 на расстояние 16 (слова «two beer » с пробелами). Но работа распаковщика некорректна. В распаковщике добавляется новая переменная ln, которая схожа с переменной dist – в ln хранится сконвертированная в целочисленное значение информация о длине фрагмента. Длина ссылки меняется на i += 4. На расстояние до фрагмента было выделено 2 символа, а код может найти фрагменты на больших расстояниях. Вводятся ограничения (i - j) < 100. Реализация алгоритма LZ77 завершена. Для сравнения, как сильно сжимается исходный файл, прописывается в конце вывод длины исходной строки и

сжатой. Проверим алгоритм ZL77 на произвольном тексте из интернета (рис. 5).

```
src:    Развитие формата рок-групп связано с возрастанием доли собственного творчества в
pack:   Развитие формата рок-групп связано с возрастанием доли собственного творчества в
unpack: Развитие формата рок-групп связано с возрастанием доли собственного творчества в
699 723
```

Рисунок 5. Проверка алгоритма

Алгоритм сжал текст и распаковал.

Выводы

Таким образом был описан алгоритм сжатия данных Лемпеля – Зива LZ77, а также проведена его практическая реализация на языке программирования Python.

Библиографический список

1. Фатеенков Д. В. Сжатие данных с применением алгоритмов BWT и RLE на языке программирования C++ // Постулат. 2022. №. 2.
2. Ковалева И. В., Размахнина А. Н., Лучанинов Д. В. Реализация преобразования Барроуза-Уилера с помощью языка C++ // Постулат. 2016. №. 12.
3. Кизянов А. О. Реализация алгоритма Хаффмана на языке программирования Python // Постулат. 2017. №. 5.
4. Ленкин А. В., Лучанинов Д. В. Реализация алгоритма преобразования «Стопка книг» с помощью языка C++ // Постулат. 2017. №. 1.
5. Рябко Б. Я. Сжатие данных с помощью “мнимого скользящего окна” // Проблемы передачи информации. 1996. Т. 32. №. 2. С. 22-30.
6. 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.