МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

ЛАБОРАТОРНАЯ РАБОТА

Алгоритм LZW

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

Научный руководитель

доцент, к. п. н. _____ А. С. Гераськин

подпись, дата

Саратов 2022

Применение

Опубликование алгоритма LZW произвело большое впечатление на всех специалистов по сжатию информации. За этим последовало большое количество программ и приложений с различными вариантами этого метода.

Этот метод позволяет достичь одну из наилучших степеней сжатия среди других существующих методов сжатия графических данных, при полном отсутствии потерь или искажений в исходных файлах. В настоящее время используется в файлах формата TIFF, PDF, GIF, PostScript и других, а также отчасти во многих популярных программах сжатия данных (ZIP, ARJ, LHA).

Описание

Процесс сжатия выглядит следующим образом: последовательно считываются символы входного потока и происходит проверка, существует ли в созданной таблице строк такая строка. Если такая строка существует, считывается следующий символ, а если строка не существует, в поток заносится код для предыдущей найденной строки, строка заносится в таблицу, а поиск начинается снова.

Например, если сжимают байтовые данные (текст), то строк в таблице окажется 256 (от " 0 " до " 255 "). Если используется 10-битный код, то под коды для строк остаются значения в диапазоне от 256 до 1023. Новые строки формируют таблицу последовательно, т. е. можно считать индекс строки ее кодом.

Для декодирования на вход подается только закодированный текст, поскольку алгоритм LZW может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту. Алгоритм генерирует однозначно декодируемый код за счет того, что каждый раз, когда генерируется новый код, новая строка добавляется в таблицу строк. LZW постоянно проверяет, является ли строка уже известной, и, если так, выводит существующий код без генерации нового. Таким образом, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер. Следовательно, при декодировании во время получения нового кода генерируется новая строка, а при получении уже известного, строка извлекается из словаря.

Алгоритм

Кодирование

- Начало.
- **Шаг 1.** Все возможные символы заносятся в словарь. Во входную фразу X заносится первый символ сообщения.
- **Шаг 2.** Считать очередной символ Y из сообщения.
- **Шаг 3.** Если Y — это символ конца сообщения, то выдать код для X, иначе:
  - Если фраза XY уже имеется в словаре, то присвоить входной фразе значение XY и перейти к **Шагу 2** ,
  - Иначе выдать код для входной фразы X, добавить XY в словарь и присвоить входной фразе значение Y. Перейти к **Шагу 2.**
- Конец.

Декодирование

- Начало.
- **Шаг 1.** Все возможные символы заносятся в словарь. Во входную фразу X заносится первый код декодируемого сообщения.
- **Шаг 2.** Считать очередной код Y из сообщения.
- **Шаг 3.** Если Y — это конец сообщения, то выдать символ, соответствующий коду X, иначе:
  - Если фразы под кодом XY нет в словаре, вывести фразу, соответствующую коду X, а фразу с кодом XY занести в словарь.
  - Иначе присвоить входной фразе код XY и перейти к **Шагу 2** .
- Конец.

Пример

Рассмотрим пример сжатия и декодирования сообщения. Сначала создадим начальный словарь единичных символов. В стандартной кодировке ASCII имеется 256 различных символов, поэтому, для того, чтобы все они были корректно закодированы (если нам неизвестно, какие символы будут присутствовать в исходном файле, а какие — нет), начальный размер кода будет равен 8 битам. Если нам заранее известно, что в исходном файле будет меньшее количество различных символов, то вполне разумно уменьшить количество бит. Чтобы инициализировать таблицу, мы установим соответствие кода 0 соответствующему символу с битовым кодом 00000000, тогда 1 соответствует символу с кодом 00000001, и т.д., до кода 255.

Больше в таблице не будет других кодов, обладающих этим свойством.
По мере роста словаря, размер групп должен расти, с тем чтобы учесть новые элементы. 8-битные группы дают 256 возможных комбинации бит, поэтому, когда в словаре появится 256-е слово, алгоритм должен перейти к 9-битным группам. При появлении 512-ого слова произойдет переход к 10-битным группам, что дает возможность запоминать уже 1024 слова и т.д.

В нашем примере алгоритму заранее известно о том, что будет использоваться всего 5 различных символов, следовательно, для их хранения будет использоваться минимальное количество бит, позволяющее нам их запомнить, то есть 3 (8 различных комбинаций).

Символ	Битовый код	Код
a	000	0
b	001	1
c	010	2
d	011	3
e	100	4

Кодирование

Пусть мы сжимаем последовательность $abacabadabacabae$.

- **Шаг 1:** Тогда, согласно изложенному выше алгоритму, мы добавим к изначально пустой строке a и проверим, есть ли строка a в таблице. Поскольку мы при инициализации занесли в таблицу все строки из одного символа, то строка a есть в таблице.
- **Шаг 2:** Далее мы читаем следующий символ b из входного потока и проверяем, есть ли строка ab в таблице. Такой строки в таблице пока нет.

Добавляем в таблицу $\langle 5 \rangle\ ab$. В поток: $\langle 0 \rangle$;

- **Шаг 3:** ba — нет. В таблицу: $\langle 6 \rangle\ ba$. В поток: $\langle 1 \rangle$;
- **Шаг 4:** ac — нет. В таблицу: $\langle 7 \rangle\ ac$. В поток: $\langle 0 \rangle$;
- **Шаг 5:** ca — нет. В таблицу: $\langle 8 \rangle\ ca$. В поток: $\langle 2 \rangle$;
- **Шаг 6:** ab — есть в таблице; aba — нет. В таблицу: $\langle 9 \rangle\ aba$. В поток: $\langle 5 \rangle$;
- **Шаг 7:** ad — нет. В таблицу: $\langle 10 \rangle\ ad$. В поток: $\langle 0 \rangle$;
- **Шаг 8:** da — нет. В таблицу: $\langle 11 \rangle\ da$. В поток: $\langle 3 \rangle$;
- **Шаг 9:** aba — есть в таблице; $abac$ — нет. В таблицу: $\langle 12 \rangle\ abac$. В поток: $\langle 9 \rangle$;
- **Шаг 10:** ca — есть в таблице; cab — нет. В таблицу: $\langle 13 \rangle\ cab$. В поток: $\langle 8 \rangle$;
- **Шаг 11:** ba — есть в таблице; bae — нет. В таблицу: $\langle 14 \rangle\ bae$. В поток: $\langle 6 \rangle$;
- **Шаг 12:** И, наконец последняя строка e, за ней идет конец сообщения, поэтому мы просто выводим в поток $\langle 4 \rangle$.

Текущая строка	Текущий символ	Следующий символ	Вывод		Словарь
			Код	Биты	
ab	a	b	0	000	5: ab
ba	b	a	1	001	6: ba
ac	a	c	0	000	7: ac
ca	c	a	2	010	8: ca
ab	a	b	-	-	- -
aba	b	a	5	0101	9: aba
ad	a	d	0	0000	10: ad
da	d	a	3	0011	11: da
ab	a	b	-	-	- -
aba	b	a	-	-	- -
abac	a	c	9	1001	12: abac
ca	c	a	-	-	- -
cab	a	b	8	1000	13: cab
ba	b	a	-	-	- -
bae	a	e	6	0110	14: bae
e	e	-	4	0100	- -

Итак, мы получаем закодированное сообщение 01025039864 и его битовый эквивалент $000001000010010100000011100110000011100100$. Каждый символ исходного сообщения был закодирован группой из трех бит, сообщение содержало 16 символов, следовательно длина сообщения составляла $3 \cdot 16 = 48$ бит.

Закодированное же сообщение так же сначала кодировалось трехбитными группами, а при появлении в словаре восьмого слова — четырехбитными, итого длина сообщения составила $4 \cdot 3 + 7 \cdot 4 = 40$ бит, что на 8 бит короче исходного.

Декодирование

Особенность LZW заключается в том, что для декомпрессии нам не надо сохранять таблицу строк в файл для распаковки. Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов.

Теперь представим, что мы получили закодированное сообщение, приведённое выше, и нам нужно его декодировать. Прежде всего нам нужно знать начальный словарь, а последующие записи словаря мы можем реконструировать уже на ходу, поскольку они являются просто конкатенацией предыдущих записей. Кроме того, в процессе кодировании и декодировании коды в словарь добавляются во время обработки одного и того же символа, т.е. это происходит “синхронно”.

Данные		На выходе	Новая запись	
Биты	Код		Полная	Частичная
000	0	a	- -	5: a?
001	1	b	5: ab	6: b?
000	0	a	6: ba	7: a?
010	2	c	7: ac	8: c?
0101	5	ab	8: ca	9: ab?
0000	0	a	9: aba	10: a?
0011	3	d	10: ad	11: d?
1001	9	aba	11: da	12: aba?
1000	8	ca	12: abac	13: ca?
0110	6	ba	13: cab	14: ba?
0100	4	e	14: bae	- -

Код программы

```
from lzw import LZW
import os
import sys
import numpy as np
import matplotlib.image as mpimg

print("0 - Сжатие, 1 - Расжатие")
oper = input()
if oper == "0":
    compressor = LZW('studies\\tkisi\\sample.tif')
    compressor.compress()
    img = mpimg.imread('studies\\tkisi\\sample.tif')
    size_origin = os.stat('studies\\tkisi\\sample.tif').st_size
    size_compress = os.stat('Compressed/sample_compressed.lzw').st_size
    print("Коэффициент сжатия: ")
    print(round(size_compress / size_origin, 3))
    width = np.size(img, 1)
    height = np.size(img, 0)
    quality = (101 - ((width * height) * 3) / size_compress)
    print("Качество сжатия: " + str(quality))
if oper == "1":
    decompressor = LZW('Compressed\sample_compressed.lzw')
    decompressor.decompress()
```