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 Лемпеля-Зива (LZ77, LZ78)

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

Алгоритмы LZ77 и LZ78

LZ77 и LZ78 — алгоритмы сжатия без потерь, опубликованные в статьях Абрахама Лемпеля и Якоба Зива в 1977 и 1978 годах. Эти алгоритмы наиболее известны в семействе LZ, которое включает в себя также LZW, LZSS, LZMA и другие алгоритмы. Оба алгоритма относятся к алгоритмам со словарным подходом. LZ77 использует, так называемое, «скользящее окно», что эквивалентно неявному использованию словарного подхода, впервые предложенного в LZ78.*

Принцип работы LZ77

Основная идея алгоритма это замена повторного вхождения строки ссылкой на одну из предыдущих позиций вхождения. Для этого используют метод скользящего окна. Скользящее окно можно представить в виде динамической структуры данных, которая организована так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Таким образом, сам процесс сжимающего кодирования согласно LZ77 напоминает написание программы, команды которой позволяют обращаться к элементам скользящего окна, и вместо значений сжимаемой последовательности вставлять ссылки на эти значения в скользящем окне. В стандартном алгоритме LZ77 совпадения строки кодируются парой:

- длина совпадения (match length)
- смещение (offset) или дистанция (distance)

Кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в словаре на значение смещения символов и скопировать значение длины символов, начиная с текущей позиции». Особенность данного алгоритма сжатия заключается в том, что использование кодируемой пары длина-смещение является не только приемлемым, но и эффективным в тех случаях, когда значение длины превышает значение смещения. Пример с командой копирования не совсем очевиден: «Вернуться на 1 символ назад в буфере и скопировать 7 символов, начиная с текущей позиции». Каким образом можно скопировать 7 символов из буфера, когда в настоящий момент в буфере находится только 1 символ? Однако следующая интерпретация кодирующей пары может прояснить ситуацию: каждые 7 последующих символов совпадают (эквивалентны) с 1 символом перед ними. Это означает, что каждый символ можно однозначно определить переместившись назад в буфере, даже если данный символ еще отсутствует в буфере на момент декодирования текущей пары длина-смещение.

Описание алгоритма

LZ77 использует скользящее по сообщению окно. Допустим, на текущей итерации окно зафиксировано. С правой стороны окна наращиваем подстроку, пока она есть в строке <скользящее окно + наращиваемая строка> и начинается в скользящем окне. Назовем наращиваемую строку буфером. После наращивания алгоритм выдает код состоящий из трех элементов:

- смещение в окне;
- длина буфера;
- последний символ буфера.

В конце итерации алгоритм сдвигает окно на длину равную длине буфера+1.
Пример kababababab

Содержимое окна	Содержимое буфера	КОД
$kababababab$	k	<0,0,k>
$kababababab$	a	<0,0,a>
$kababababab$	b	<0,0,b>
k ab $ababab$	aba	<2,2,a>
$kaba$ babab ab	$bababz$	<2,5,z>

Описание алгоритма LZ78

В отличие от LZ77, работающего с уже полученными данными, LZ78 ориентируется на данные, которые только будут получены (LZ78 не использует скользящее окно, он хранит словарь из уже просмотренных фраз). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда мы выдаем код в виде (индекс строки в словаре без последнего символа, последний символ).

Пример kababababab

Содержимое словаря	Содержимое считываемой строки	КОД
	k	<0,k>
k	a	<0,a>
k, a	b	<0,b>
k, $\boxed{\text{a}}$, b	ab	<2,b>
k, a, b, $\boxed{\text{ab}}$	aba	<4,a>
k, a, $\boxed{\text{b}}$, ab, aba	ba	<3,a>
k, a, b, ab, $\boxed{\text{aba}}$, ba	$abab$	<5,b>
k, a, b, ab, aba, ba, $abab$	z	<0,z>

Код программы

Lz77

```
import math
import struct
import time


def compression(text, file, max_buffer):
    x = 16
    max_l = int(math.pow(2, (x - math.log(max_buffer, 2))))
    buffer_pointer = 0
    l_pointer = 0
    while l_pointer < len(text):
        buffer = text[buffer_pointer:l_pointer]
        next = text[l_pointer:l_pointer + max_l]
        turple = make_turple(buffer, next)
        offset = turple[0]
        length = turple[1]
        char = bytes(turple[2], "ansi")
        shifted_offset = offset << 6
        off_length = shifted_offset + length

        byte = struct.pack(">Hc", off_length, char)
        file.write(byte)
        l_pointer += length + 1
        buffer_pointer = l_pointer - max_buffer
        if buffer_pointer < 0:
            buffer_pointer = 0


def make_turple(buffer, next):
    if len(buffer) == 0:
        return 0, 0, next[0]
    if len(next) == 0:
        return -1, -1, ""
    length = 0
    offset = 0
    tmp_buffer = buffer + next
    buffer_pointer = len(buffer)
    for i in range(len(buffer)):
        tmp_length = 0
        while tmp_buffer[i + tmp_length] == tmp_buffer[buffer_pointer + tmp_length]:
            tmp_length += 1
            if buffer_pointer + tmp_length == len(tmp_buffer):
                tmp_length -= 1
                break
            if i + tmp_length >= buffer_pointer:
                break
        if tmp_length > length:
            offset = i
            length = tmp_length
    return offset, length, tmp_buffer[buffer_pointer + length]


def decompression(input, output, max_buffer):
    i = 0
    text = ""
    while i < len(input):
        off_length, char = struct.unpack(">Hc", input[i:i+3])
```

```
        char = chr(ord(char.decode('ansi')))
        offset = off_length >> 6
        length = off_length - (offset << 6)
        i += 3
        if offset == 0 and length == 0:  # (0, 0, char)
            text += char
        else:
            pointer = len(text) - max_buffer
            if pointer < 0:
                pointer = offset
            else:
                pointer += offset
            for j in range(length):
                text += text[pointer + j]
            text += char
    output.write(text)


def main():
    print("Введите 0 для кодирования, 1 для раскодирования")
    mode = input()
    if mode == "0":
        text = (open("studies\\tkisi\\Тест_8.txt", mode='r', encoding='cp1251')).read()
        comp = open("res7.bin", mode='wb')
        start = int(round(time.time() * 1000))
        compression(text, comp, 1024)
        end = int(round(time.time() * 1000))
        print("Текст закодирован и помещен в файл Coding")
        print("Скорость кодирования: " + str((end - start)) + " мс")
    if mode == "1":
        decomp = open("restore7.txt", mode='w')
        copm_text = open("res7.bin", mode='rb').read()
        start = int(round(time.time() * 1000))
        decompression(copm_text, decomp, 1024)
        end = int(round(time.time() * 1000))
        print("Текст раскодирован и помещен в файл Decoding")
        print("Скорость декодирования: " + str((end - start)) + " мс")


if __name__ == "__main__":
    main()
```

Lz78

```
import struct
import time


def compression(text, file):
    tree = create_tree(text)
    for node in tree:
        pair = node[0]
        first = pair[0]
        second = bytes(pair[1], "ansi")
        byte = struct.pack(">Hc", first, second)
        file.write(byte)


def create_tree(text):
    tree = []
```

```
    not_changed = True
    first_in_pair = 0
    new_entry = True
    index = 2
    i = 1
    tree.append([[0, text[0]], 1, text[0]])
    while i < len(text):
        char = text[i]
        while new_entry:
            for j in range(len(tree)):
                if tree[j][2] == char:
                    not_changed = False
                    if i < len(text) - 1:
                        i += 1
                        char += text[i]
                        first_in_pair = j + 1
                        break
                    else:
                        char = ''
                        new_entry = False
                if j == len(tree) - 1:
                    new_entry = False
                    if not_changed:
                        char = text[i]
                        first_in_pair = 0
        pair = [first_in_pair, text[i]]
        tree.append([pair, index, char])
        index += 1
        new_entry = True
        i += 1
        not_changed = True
    return tree


def decompression(input, output):
    i = 0
    idx = 0
    tree = []
    while i < len(input):
        index, char = struct.unpack(">Hc", input[i:i+3])
        char = chr(ord(char.decode('ansi')))
        i += 3
        idx += 1
        if index == 0:
            tree.append([[index, char], idx, char])
        else:
            tree.append([[index, char], idx, (tree[index - 1][2] + char)])
    for node in tree:
        output.write(node[2])


def main():
    mode = input()  # 0 - Архивация, 1 - Разархивация
    if mode == "0":
        input_t = parse('studies\\tkisi\\Тест_3.txt')
        comp = open("res8.bin", mode='wb')
        start = int(round(time.time() * 1000))
        compression(input_t, comp)
        end = int(round(time.time() * 1000))
        print("Скорость сжатия: " + str((end - start)) + " мс")
    if mode == "1":
        decompressed = open('restore8.txt', mode='w', encoding='ansi')
```

```
        compressed = open("res8.bin", mode='rb').read()
        decompression(compressed, decompressed)


def parse(file):
    r = []
    f = open(file, "r", encoding='ansi')
    text = f.read()
    return text


if __name__ == "__main__":
    main()
```