Abstract

This specification defines a lossless compressed data format that

compresses data using a combination of the LZ77 algorithm and Huffman

coding, with efficiency comparable to the best currently available

general-purpose compression methods.

Специфікація визначає алгоритим стиснення без втрат завдяк якого можна стискати дані використовуючи комбінацію алгоритму LZ77 та кодування Хаффмана.При цьому,даний алгоритм можна назвати не менш ефективним за інші загальні доступні алгоритми стиснення .

Status of this Memo

This Internet-Draft is submitted in full conformance with the

provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering

Task Force (IETF). Note that other groups may also distribute

working documents as Internet-Drafts. The list of current Internet-

Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months

and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 4, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the

document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents

(http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

Alakuijala & Szabadka Expires November 4, 2016 [Page 1]

Internet-Draft Brotli May 2016

Table of Contents

1. Introduction . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.1. Purpose . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.2. Intended audience . . . . . . . . . . . . . . . . . . . . 3

1.3. Scope . . . . . . . . . . . . . . . . . . . . . . . . . . 3

1.4. Compliance . . . . . . . . . . . . . . . . . . . . . . . . 4

1.5. Definitions of terms and conventions used . . . . . . . . 4

1.5.1. Packing into bytes . . . . . . . . . . . . . . . . . 4

2. Compressed representation overview . . . . . . . . . . . . . . 5

3. Compressed representation of prefix codes . . . . . . . . . . . 9

3.1. Introduction to prefix coding . . . . . . . . . . . . . . 9

3.2. Use of prefix coding in the brotli format . . . . . . . . 10

3.3. Alphabet sizes . . . . . . . . . . . . . . . . . . . . . 12

3.4. Simple prefix codes . . . . . . . . . . . . . . . . . . . 13

3.5. Complex prefix codes . . . . . . . . . . . . . . . . . . 14

4. Encoding of distances . . . . . . . . . . . . . . . . . . . . 16

5. Encoding of literal insertion lengths and copy lengths . . . . 18

6. Encoding of block switch commands . . . . . . . . . . . . . . 20

7. Context modeling . . . . . . . . . . . . . . . . . . . . . . . 22

7.1. Context modes and context ID lookup for literals . . . . 22

7.2. Context ID for distances . . . . . . . . . . . . . . . . 24

7.3. Encoding of the context map . . . . . . . . . . . . . . . 24

8. Static dictionary . . . . . . . . . . . . . . . . . . . . . . 26

9. Compressed data format . . . . . . . . . . . . . . . . . . . . 29

9.1. Format of the stream header . . . . . . . . . . . . . . . 29

9.2. Format of the meta-block header . . . . . . . . . . . . . 30

9.3. Format of the meta-block data . . . . . . . . . . . . . . 32

10. Decoding algorithm . . . . . . . . . . . . . . . . . . . . . 34

11. Considerations for compressor implementations . . . . . . . . 36

11.1. Trivial compressor . . . . . . . . . . . . . . . . . . . 36

11.2. Aligning compressed meta-blocks to byte boundaries . . . 37

11.3. Creating self-contained parts within the compressed data 37

12. Security Considerations . . . . . . . . . . . . . . . . . . . 38

13. IANA Considerations . . . . . . . . . . . . . . . . . . . . . 40

14. Informative References . . . . . . . . . . . . . . . . . . . 40

15. Source code . . . . . . . . . . . . . . . . . . . . . . . . . 40

16. Acknowledgments . . . . . . . . . . . . . . . . . . . . . . . 41

Appendix A. Static dictionary data . . . . . . . . . . . . . . . 41

Appendix B. List of word transformations . . . . . . . . . . . . 121

Appendix C. Computing CRC-32 check values . . . . . . . . . . . 124

Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . 124

Internet-Draft Brotli May 2016

1. Introduction

1.1. Purpose

The purpose of this specification is to define a lossless compressed

data format that:

Мета специфікації визначити формат даних для стиснення без втрат яке б :

\* Is independent of CPU type, operating system, file system,

and character set, and hence can be used for interchange;

Було незалежним від виду процесору ,операційної системи,файлової системи,

А також типу кодування символів,і відповідно його можна було б застосовувати для обміну даними.

\* Can be produced or consumed, even for an arbitrarily long

sequentially presented input data stream, using only an a

priori bounded amount of intermediate storage, and hence

can be used in data communications or similar structures,

such as Unix filters;

Його можна застосувати до довільно великого обсягу даних,які послідовно записані у вхідному потоці ,використовуючи лише обмежене а пріорі сховище для проміжних даних.

\* Compresses data with a compression ratio comparable to the

best currently available general-purpose compression methods,

and in particular, considerably better than the gzip program;

Стискує дані з відношенням стискання яке порівнюване з аналогічними відношеннями інших алгоритмів стиснення.Зокрема,значно кращий за gzip алгоритм стисення даних.

\* Decompresses much faster than current LZMA implementations.

Процесс відновлення даних швидший за сучасну імплементацію LZMA алгоритму.

The data format defined by this specification does not attempt to:

\* Allow random access to compressed data;

\* Compress specialized data (e.g., raster graphics) as well

as the best currently available specialized algorithms.

Проте ,даний алгоритм не призначений для випадкового доступу до будь-якої частини стиснених даних.Також він не здатний стискати спеціалізовані формати даних ,як-от растрову графіку тощо.

1.2. Intended audience

This specification is intended for use by software implementers to

compress data into and/or decompress data from the brotli format.

The text of the specification assumes a basic background in

programming at the level of bits and other primitive data

representations. Familiarity with the technique of Huffman coding is

helpful but not required.

This specification uses heavily the notations and terminology

introduced in the DEFLATE format specification [RFC 1951]. For the

sake of completeness, we always include the whole text of the

relevant parts of RFC 1951, therefore familiarity with the DEFLATE

format is helpful but not required.

The compressed data format defined in this specification is an

integral part of the WOFF 2.0 web font file format [WOFF2], therefore

this specification is also intended for implementers of WOFF 2.0

compressors and decompressors.

1.3. Scope

Alakuijala & Szabadka Expires November 4, 2016 [Page 3]

Internet-Draft Brotli May 2016

The specification specifies a method for representing a sequence of

bytes as a (usually shorter) sequence of bits, and a method for

packing the latter bit sequence into bytes.

В специфікації зазначено метод для представлення послідовності байтів як послідовностей бітів ,та метод для пакування послідовностей бітів в байти назад.

1.4. Compliance

Unless otherwise indicated below, a compliant decompressor must be

able to accept and decompress any data set that conforms to all the

specifications presented here. A compliant compressor must produce

data sets that conform to all the specifications presented here.

1.5. Definitions of terms and conventions used

Byte: 8 bits stored or transmitted as a unit (same as an octet). For

this specification, a byte is exactly 8 bits, even on machines that

store a character on a number of bits different from eight. See

below for the numbering of bits within a byte.

Далі під байтом буде визначено послідовність з 8 бітів збережених як одиниці інформації.

String: a sequence of arbitrary bytes.

Під стрічкою мається на увазі будь-яка послідовність байтів довільної довжини.

Bytes stored within a computer do not have a "bit order", since they

are always treated as a unit. However, a byte considered as an

integer between 0 and 255 does have a most- and least-significant

bit, and since we write numbers with the most-significant digit on

the left, we also write bytes with the most-significant bit on the

left. In the diagrams below, we number the bits of a byte so that bit

0 is the least-significant bit, i.e., the bits are numbered:

У визначенні байту послуговуватимемося позначенням спочатку найбільш важливого біту зліва.

+--------+

|76543210|

+--------+

Within a computer, a number may occupy multiple bytes. All multi-byte

numbers in the format described here are stored with the least-

significant byte first (at the lower memory address). For example,

the decimal number 520 is stored as:

Оскільки число може бути більшим за 1 байт ,будемо розміщувати 1 байт числа зліва(найменш важливий байт),а решту – справа.

0 1

+--------+--------+

|00001000|00000010|

+--------+--------+

^ ^

| |

| + more significant byte = 2 x 256

+ less significant byte = 8

1.5.1. Packing into bytes

Alakuijala & Szabadka Expires November 4, 2016 [Page 4]

Internet-Draft Brotli May 2016

This document does not address the issue of the order in which bits

of a byte are transmitted on a bit-sequential medium, since the final

data format described here is byte- rather than bit-oriented.

However, we describe the compressed block format below as a sequence

of data elements of various bit lengths, not a sequence of bytes. We

must therefore specify how to pack these data elements into bytes to

form the final compressed byte sequence:

Отже,остаточна форма стиснених даних матиме наступні властивості:

\* Data elements are packed into bytes in order of

increasing bit number within the byte, i.e., starting

with the least-significant bit of the byte.

Елементи даних спаковані в байти в порядку зростання бітового числа в байті,починаючи з найменш важливого біту .

\* Data elements other than prefix codes are packed

starting with the least-significant bit of the data

element. These are referred to here as integer values

and are considered unsigned.

\* Prefix codes are packed starting with the most-significant

bit of the code.

Префіксні коди розташовані починаючи з найбільш важливого біту

In other words, if one were to print out the compressed data as a

sequence of bytes, starting with the first byte at the \*right\* margin

and proceeding to the \*left\*, with the most-significant bit of each

byte on the left as usual, one would be able to parse the result from

right to left, with fixed-width elements in the correct msb-to-lsb

order and prefix codes in bit-reversed order (i.e., with the first

bit of the code in the relative lsb position).

Відповідно при прочитанні стиснених даних «справа –наліво» фіксовані елементи стиснення будуть прочитані спочатку від найголовнішого до найменш важливого біту,для прфексних кодів-навпаки.

Приклад:значення 3-бітного числа -6;4-бітного – 2; префіксний код – 110;ще один префікс ний код -10;12 –бітного числа- 3628;

байт 2 байт 1 байт 0

+--------+--------+--------+

|11100010|11000101|10010110|

+--------+--------+--------+

^ ^ ^ ^ ^

| | | | |

| | | | +------ значення 6

| | | +---------- значення 2

| | +-------------- префіксний код 110

| +---------------- префіксний код 10

+----------------------------- значення 3628

As an example, consider packing the following data elements into a

sequence of 3 bytes: 3-bit integer value 6, 4-bit integer value 2,

prefix code 110, prefix code 10, 12-bit integer value 3628.

byte 2 byte 1 byte 0

+--------+--------+--------+

|11100010|11000101|10010110|

+--------+--------+--------+

^ ^ ^ ^ ^

| | | | |

| | | | +------ integer value 6

| | | +---------- integer value 2

| | +-------------- prefix code 110

| +---------------- prefix code 10

+----------------------------- integer value 3628

2. Compressed representation overview

A compressed data set consists of a header and a series of meta-

blocks. Each meta-block decompresses to a sequence of 0 to 16,777,216

(16 MiB) uncompressed bytes. The final uncompressed data is the

concatenation of the uncompressed sequences from each meta-block.

Стисненні дані містять в собі заголовок та послідовність мета-блоків. Кожен мета-блок відновлює від 0 до 16 777 216 нестиснених байтів. Остаточні відновлені дані є конкатенацією всіх відновлених послідовностей з кожного мета-блоку.

Alakuijala & Szabadka Expires November 4, 2016 [Page 5]

Internet-Draft Brotli May 2016

The header contains the size of the sliding window that was used

during compression. The decompressor must retain at least that

amount of uncompressed data prior to the current position in the

stream, in order to be able to decompress what follows. The sliding

window size is a power of two, minus 16, where the power is in the

range of 10 to 24. The possible sliding window sizes range from 1 KiB

- 16 B to 16 MiB - 16 B.

Заголовок містить розмір розсувного вікна яке було використанне під час стиснення .При відновленні це вікно має бути заповнене нестисненим даним для того щоб відновити наступні байти.Розмір розсувного вікна є степеню двійки і може бути зміненим від 1 кілобайта до 16 мегабайт.

Each meta-block is compressed using a combination of the LZ77

algorithm (Lempel-Ziv 1977, [LZ77]) and Huffman coding. The result of

Huffman coding is referred to here as a prefix code. The prefix

codes for each meta-block are independent of those for previous or

subsequent meta-blocks; the LZ77 algorithm may use a reference to a

duplicated string occurring in a previous meta-block, up to the

sliding window size of uncompressed bytes before. In addition, in

the brotli format, a string reference may instead refer to a static

dictionary entry.

Кожен мета-блок стиснений використовуючи комбінацію зLZ77 та алгоритму кодування Хаффмана.

Результатом кодування Хаффмана є префіксні коди.префікні коди для кожного мета-блоку є незалежними від інших кодів з попереднього чи наступного блоку.Алгоритм LZ77 ,в свою чергу,використовує посилання до стрічок які повторилися в попередніх мета-блоках.Також,посилання на стріску може звертатися до статисного словника вбудованого в реалізації словника.

Each meta-block consists of two parts: a meta-block header that

describes the representation of the compressed data part, and a

compressed data part. The compressed data consists of a series of

commands. Each command consists of two parts: a sequence of literal

bytes (of strings that have not been detected as duplicated within

the sliding window), and a pointer to a duplicated string,

represented as a pair <length, backward distance>. There can be zero

literal bytes in the command. The minimum length of the string to be

duplicated is two, but the last command in the meta-block is

permitted to have only literals and no pointer to a string to

duplicate.

Кожен мета-блок складається з двох частин :заголовку мета-блоку який описує представлення частини даних яка стиснута,та цю ж частину стиснених даних. Частина стиснених даних складається з послідовності команд. В свою чергу кожна з команд складається з двох частин :послідовності байтів ,або ж стрічок(які до того не були розпізнані розсувним вікном як такі,що мають повторення),та указник на стрічку з повторенням. ЇЇ представлено у вигляді пари <довжина,відстань>.В команді може бути 0 літеральних байтів(стрічка порожня).Мінімальна довжина стрічки для копіювання -2.Остання команда в блоці може містити лише літерали(байти) та не містить указник на стрічку з повторенням.

Each command in the compressed data is represented using three

categories of prefix codes:

Кожна з команд стиснених даних представлення у вигляді 3 категорій префікс них кодів :

1) One set of prefix codes are for the literal sequence lengths

(also referred to as literal insertion lengths) and

backward copy lengths. That is, a single code word represents

two lengths, one of the literal sequence and one of the

backward copy.

1)Перша множина префікс них кодів призначена для довжин послідовностей літералів та відстаней до копій.таким чином один префікс ний код визначає дві довжини,довжину самої послідовності та відстань до копії.

2)

2) One set of prefix codes are for literals.

2)Другий тільки для літералів

3) One set of prefix codes are for distances.

3)Третій тільки для відстаней

The prefix code descriptions for each meta-block appear in a compact

form just before the compressed data in the meta-block header. The

insert-and-copy length and distance prefix codes may be followed by

extra bits that are added to the base values determined by the codes.

The number of extra bits is determined by the code.

Префіксні коди для кожного блоку знаходяться перед стисненим даним в заголовку мате-блоку.

Alakuijala & Szabadka Expires November 4, 2016 [Page 6]

Internet-Draft Brotli May 2016

One meta-block command then appears as a sequence of prefix codes:

Приклад команди як послідовності префікс них кодів

Insert-and-copy length, literal, literal, ..., literal, distance

Довжина вставки та копії ,літерал,літерал,….,відстань

Довжина вставки визначає кількість літералів які йдуть одразу після нього.

Відстань визначає як далеко слід звернутися назад аби повернутися до копії.

Довжина копії визначає кількість байтів в копії.

В результаті нестиснені дані мають вигляд :

Літерал,літерал,…..,копія,копія.

where the insert-and-copy defines an insertion length and a copy

length. The insertion length determines the number of literals that

immediately follow. The distance defines how far back to go for the

copy and the copy length determines the number of bytes to copy. The

resulting uncompressed data is the sequence of bytes:

literal, literal, ..., literal, copy, copy, ..., copy

where the number of literal bytes and copy bytes are determined by

the insert-and-copy length code. (The number of bytes copied for a

static dictionary entry can vary from the copy length.)

The last command in the meta-block may end with the last literal if

the total uncompressed length of the meta-block has been satisfied.

In that case there is no distance in the last command, and the copy

length is ignored.

Остання команда в мета-блоці може закінчуватися останнім літералом якщо загальна довжина нестиснених даних була повністю заповнена.В цьому випадку немає відстані в цій команді,і довжина копії не потрібна.

There can be more than one prefix code for each category, where the

prefix code to use for the next element of that category is

determined by the context of the compressed stream that precedes that

element. Такоє може бути більше ніж один префікс ний код для кожної категорії,де префікс ний код для наступного лемента цієї категорії визначений в контексті стиснених даних які передують цьому елементові.

Part of that context is three current block types, one for

each category.

Частиною цього контексту є 3 поточні типи блоків,по-одному для кожної категорії.

A block type is in the range of 0..255.Діапазон типу блоку від 0 до 255.

For each

category there is a count of how many elements of that category

remain to be decoded using the current block type.

Для кожної категорії існує лічильник ,стосовно того скільки елементів цієї категорії слід декодувати в поточному типі блоку.

Once that count is

expended, a new block type and block count is read from the stream

immediately preceding the next element of that category, which will

use the new block type.

Як тільки лічильник був розширений,новий тип блоку та новий лічильник дістають з потоку перед наступним елементом категорії ,який буде використовувати новий тип блоку.

The insert-and-copy block type directly determines which prefix code

to use for the next insert-and-copy element.

Тип вставки та копіювання блоку визначає який префікс ний код використати для наступного елементу цього ж типу.

For the literal and

distance elements, the respective block type is used in combination

with other context information to determine which prefix code to use

for the next element.

Для літералів та відстаней,відповідні типи блоків використовують в комбінації з іншою контекстною інформацією для наступного елементу.

Consider the following example:Розглянемо наступний приклад

(IaC0, L0, L1, L2, D0)(IaC1, D1)(IaC2, L3, L4, D2)(IaC3, L5, D3)

The meta-block here has four commands, contained in parentheses for

clarity, where each of the three categories of symbols within these

commands can be interpreted using different block types.

Цей мета-блок має чотири команди ,записані в дужках для ясності,де кожна з 3 категорій символів з цими командам може бути інтерпретована використовуючи різні типи блоків .

Here we

separate out each category as its own sequence to show an example of

block types assigned to those elements.Тут ми розділяємо кожну категорію на свою власну послідовність аби показати приклад типів блоків .

Each square-bracketed group

is a block that uses the same block type:

Кожна група з квадратним дужками використовує один тип блоку.

Alakuijala & Szabadka Expires November 4, 2016 [Page 7]

Internet-Draft Brotli May 2016

[IaC0, IaC1][IaC2, IaC3] <-- insert-and-copy: block types 0 and 1

[L0, L1][L2, L3, L4][L5] <-- literals: block types 0, 1, and 0

[D0][D1, D2, D3] <-- distances: block types 0 and 1

[IaC0, IaC1][IaC2, IaC3] <-- вставка та видалення : типи блоків 0 та 1

[L0, L1][L2, L3, L4][L5] <-- літерали : типи блоків 0 ,1 та 0

[D0][D1, D2, D3] <-- відстані :типи блоків 0 та 1

The subsequent blocks within each block category must have different

block types, but we see that block types can be reused later in the

meta-block. Наступні блоки в кожній категоріх мають мати різні типи блоків,але їх можна потім використовувати заново в мета-блоці.

The block types are numbered from 0 to the maximum block

type number of 255 and the first block of each block category is type

0.Типи блоків нумеровані від 0 до максимального числа блоків (255) і перший блок кожнох категорії має тип 0.

The block structure of a meta-block is represented by the sequence

of block-switch commands for each block category, where a block-

switch command is a pair <block type, block count>. Блочна структура мета-блоку представленя послідовністю команд для кожної категоріх,де команда має вигляд<тип блоку,лічильник блоку>.

The block-switch

commands are represented in the compressed data before the start of

each new block using a prefix code for block types and a separate

prefix code for block counts for each block category.

Команди блоку представлены в стиснених даних до початку кожного нового блоку використовуючи префікс ний код для типів блоків та окремий префікс ний код для лічильників блоків кожної категорії.

For the above

example the physical layout of the meta-block is then:Для вищеназваного прикладу фізичним макетом мета-блоку є:

IaC0 L0 L1 LBlockSwitch(1, 3) L2 D0 IaC1 DBlockSwitch(1, 3) D1

IaCBlockSwitch(1, 2) IaC2 L3 L4 D2 IaC3 LBlockSwitch(0, 1) L5 D3

where xBlockSwitch(t, n) switches to block type t for a count of n

elements.

Тут команда xBlockSwitch(t, n) перемикає до типу блоку т для n елементів. Note that in this example DBlockSwitch(1, 3) immediately

precedes the next required distance D1. Зазначимо що в цьому прикладі DBlockSwitch(1, 3) одразу спереду наступної необхідної відстані D1. It does not follow the last

distance of the previous block, D0.

Whenever an element of a category

is needed, and the block count for that category has reached zero,

then a new block type and count is read from the stream just before

reading that next element.

Коли елемент категорії потрібен,і лічильник категорії дійшов ддо нуля,тоді зчитують новий тип блоку та лічильний з потоку прямо перед читанням наступного елементу.

The block-switch commands for the first blocks of each category are

not part of the meta-block compressed data.

Команди блоку для перших блоків кожної категоріх не є частиною мета-блоку стиснених даних.

Instead the first block

type is defined to be 0, and the first block count for each category

is encoded in the meta-block header.

Натомість перший тип блоку визначений як 0 ,та перший лічильник для кожної коте горіх закодований в заголовку мета-блоку.

The prefix codes for the block

types and counts, a total of six prefix codes over the three

categories, are defined in a compact form in the meta-block header.

Всі префіксні коди для типів блоку та лічильників визначені в заголовку мет-блоку.

Each category of value (insert-and-copy lengths, literals, and

distances) can be encoded with any prefix code from a collection of

prefix codes belonging to the same category appearing in the meta-

block header.

Кожне значення категорії(вставка та копіювання ,літерали,відстані) можуть бути закодовані з одним з префікс них кодів в колекції префікс них кодів які належать до тієї ж категорії що й в заголовку мета-блоку.

The particular prefix code used can depend on two

factors: the block type of the block the value appears in, and the

context of the value.

Кожен конкретний префікс ний код залежить від типу блоку та контексту значення блоку.

In the case of the literals, the context is

the previous two bytes in the uncompressed data, and in the case of

distances, the context is the copy length from the same command.

У випадку літералів ,контекст це два попередні байти нестиснених даних.У випадку відстаней ,контекст це довжина копіювання з цієї ж команди.

For

insert-and-copy lengths, no context is used and the prefix code

depends only on the block type

Для довжина вставки та копіювання не використовують контексту та префікс ний код залежить лише від типу блоку .

. In the case of literals and

distances, the context is mapped to a context ID in the range 0..63

for literals and 0..3 for distances.

У випадку літералів та відстаней контекст має відображення на упорядковану множину від 0 до 63 для літералів та від 0 до 3 для відстаней.

The matrix of the prefix code

Матриця префікс них кодів

Alakuijala & Szabadka Expires November 4, 2016 [Page 8]

Internet-Draft Brotli May 2016

indexes for each block type and context ID, called the context map,

is encoded in a compact form in the meta-block header.

Індекси для кожного типу блоку та контекстний ID утворюють разом контекстне відображення яке записане в заголовок мета-блоку.

For example, the prefix code to use to decode L2 depends on the block

type (1), and the literal context ID determined by the two

uncompressed bytes that were decoded from L0 and L1.

Для прикладу префікс ний код який використовують для відновлення L2 залежить від типу блоку (1),і контекстний ID для літералів визначений завдяки 2 нестисненими байтами якими були відновлені L0 та L1.

Similarly, the

prefix code to use to decode D0 depends on the block type (0), and

the distance context ID determined by the copy length decoded from

IaC0.Аналогічно префікс ний код для відновлення D0 залежить на типові блоку (0),та контекстний ID визначений завдяки довжини копіювання з IaC0.

The prefix code to use to decode IaC3 depends only on the block

type (1).

Префіксний код для відновлення IaC3 залежить лише від типу блоку (1).

In addition to the parts listed above (prefix code for insert-and-

copy lengths, literals, distances, block types and block counts, and

the context map), the meta-block header contains the number of

uncompressed bytes coded in the meta-block and two additional

parameters used in the representation of match distances: the number

of postfix bits and the number of direct distance codes.

Заголовок мета-блоку також містить кількість нестиснених байтів закодованими в мета-блок та два додаткові параметри для представлення відстаней між однаковими стрічками:кількість постфікс них бітів ,та кількість прямих відстаней.

A compressed meta-block may be marked in the header as the last meta-

block, which terminates the compressed stream.

Стиснений мета-блок може бути відзначеним у заголовку як останній мета-блок ,який закінчується в стисненому потоці.

A meta-block may instead simply store the uncompressed data directly

as bytes on byte boundaries with no coding or matching strings. In

this case the meta-block header information only contains the number

of uncompressed bytes and the indication that the meta-block is

uncompressed. An uncompressed meta-block cannot be the last meta-

block.

Мета-блок також може просто зберігати нестиснені дані прямо як байти без кодування чи стрічок з копіями.В цьому випадку заголовок мета-блок містить лише інформацію стосовно кількості нестиснених байтів та позначки що мета-блок є нестисненним.Нестиснений мета-блок не може бути останнім мета-блоком.

A meta-block may also be empty, which generates no uncompressed data

at all. An empty meta-block may contain metadata information as

bytes starting on byte boundaries, which are not part of either the

sliding window or the uncompressed data. Thus, these metadata bytes

cannot be used to create matching strings in subsequent meta-blocks

and are not used as context bytes for literals.

3. Compressed representation of prefix codes

Стиснене представлення префікс них кодів

3.1. Introduction to prefix coding

Prefix coding represents symbols from an a priori known alphabet by

bit sequences (codes), one code for each symbol, in a manner such

that different symbols may be represented by bit sequences of

different lengths, but a parser can always parse an encoded string

unambiguously symbol-by-symbol.

Префіксне кодування представляє символи з відомого алфавітуу бітові послідовності,кожен код на один символ.Таким чином,різні символи можуть бути представлені бітовим послідовностями різної довжини ,але програма-парсер може однозначно прочитати закодовану стрічку символ за символом.

We define a prefix code in terms of a binary tree in which the two

edges descending from each non-leaf node are labeled 0 and 1 and in

which the leaf nodes correspond one-for-one with (are labeled with)

Зазвичай префікс ний код позначають завдяки бінарному дереву в якого два ребра які спускаються в кожного не термінальної ноди позначені 0 та 1 та в якому останні ноди(листки)поставлені у відповідність до кожного символу алфавіту.

Alakuijala & Szabadka Expires November 4, 2016 [Page 9]

Internet-Draft Brotli May 2016

the symbols of the alphabet; then the code for a symbol is the

sequence of 0's and 1's on the edges leading from the root to the

leaf labeled with that symbol.

Від префікс ним кодом для символу є послідовність 0 та 1 для ребер що йдуть з кореня дл литска позначеного цим символом.

For example:

/\ Symbol Code

0 1 ------ ----

/ \ A 00

/\ B B 1

0 1 C 011

/ \ D 010

A /\

0 1

/ \

D C

Приклад

/\ Символ Код

0 1 ------ ----

/ \ A 00

/\ B B 1

0 1 C 011

/ \ D 010

A /\

0 1

/ \

D C

A parser can decode the next symbol from the compressed stream by

walking down the tree from the root, at each step choosing the edge

corresponding to the next compressed data bit.

Парсер може відновити наступний символ з стисненого потоку спускаючись по дереву з кореня вибираючи на кожному кроці відповідне до наступного біту значення .

Given an alphabet with known symbol frequencies, the Huffman

algorithm allows the construction of an optimal prefix code (one

which represents strings with those symbol frequencies using the

fewest bits of any possible prefix codes for that alphabet). Such a

prefix code is called a Huffman code. (See [HUFFMAN] in Chapter 5,

references for additional information on Huffman codes.)

За даним алфавітом з відомими частотами ,алгоритм Хаффмана дозволяє будувати оптимальне префікс не кодування в якому для кожного символу з відомою частотою виділено нейменшу можливу кількість бітів.

Note that in the brotli format, the prefix codes for the various

alphabets must not exceed certain maximum code lengths. This

constraint complicates the algorithm for computing code lengths from

symbol frequencies. Again, see Chapter 5, references for details.

3.2. Use of prefix coding in the brotli format

Використання префіксного кодування в алгоритмі

The prefix codes used for each alphabet in the brotli format are

canonical prefix codes, which have two additional rules:

Префіксні коді які використовують для кожного алфавіту в алгоритмі є канонічними префікс ними кодами які мають два додаткові правила

\* All codes of a given bit length have lexicographically

consecutive values, in the same order as the symbols they

represent;

Всі коди певної бітової довжини мають лексикографічні послідовні значення ,в тому ж порядку що й символи які вони представляють.

\* Shorter codes lexicographically precede longer codes.

Коротші коди лексикографічно знаходяться раніше за довші коди .

We could recode the example above to follow this rule as follows,

assuming that the order of the alphabet is ABCD:

Прикладна основі алфавіту ABCD

Alakuijala & Szabadka Expires November 4, 2016 [Page 10]

Internet-Draft Brotli May 2016

Символ Кож

------ ----

A 10

B 0

C 110

D 111

I.e., 0 precedes 10, which precedes 11x, and 110 and 111 are

lexicographically consecutive.

0 перед 10,який перед 11,і далі 110 та 111 .Всі вони є лексикографічно послідовними.

Given this rule, we can define the canonical prefix code for an

alphabet just by giving the bit lengths of the codes for each symbol

of the alphabet in order; this is sufficient to determine the actual

codes.

Завдяки цьому правилу можна визначити канонічний префікс ний код для алфавіту лише знаючи бітові довжини кодів для кожного символу в алфавіті по порядку.Цього достатньо для дійсних кодів .

In our example, the code is completely defined by the sequence

of bit lengths (2, 1, 3, 3).

В нашому прикладі код повністю визначений завдяки послідовності бітової довжини (2,2,3,3).

The following algorithm generates the

codes as integers, intended to be read from most- to least-

significant bit.

Наступний алгоритм записує префіксні коди у вигляді цілих чисел ,за умови що вони будуть прочитані від найбільш до найменш важливого біту .

The code lengths are initially in tree[I].Len; the

codes are produced in tree[I].Code.

1) Count the number of codes for each code length. Let

bl\_count[N] be the number of codes of length N, N >= 1.

1)Порахувати кількість кожів для кожної довжини коду.Нехай bl\_count[N] – кількість кодів довжини N ,при N>=1.

2) Find the numerical value of the smallest code for each

code length:

Знайти числове значення найменшого коду для кожноъ довжини

code = 0;

bl\_count[0] = 0;

for (bits = 1; bits <= MAX\_BITS; bits++) {

code = (code + bl\_count[bits-1]) << 1;

next\_code[bits] = code;

}

3) Assign numerical values to all codes, using consecutive

values for all codes of the same length with the base

values determined at step 2. Codes that are never used

(which have a bit length of zero) must not be assigned a

value.

Присвоїти числове значення для всіх кодів ,використовуючи послідовні значення для всіх кодів однаковох довжини з базовими значеннями визначеним на кроці 2.

for (n = 0; n <= max\_code; n++) {

len = tree[n].Len;

if (len != 0) {

tree[n].Code = next\_code[len];

next\_code[len]++;

}

}

Example:

приклад

Alakuijala & Szabadka Expires November 4, 2016 [Page 11]

Internet-Draft Brotli May 2016

Consider the alphabet ABCDEFGH, with bit lengths (3, 3, 3, 3, 3, 2,

4, 4).Розглянемо алфавіт ABCDEFGH з відповідними довжинами (3,3,3,3,2,4,4).

Крок 1 :

After step 1, we have:

N bl\_count[N]

- -----------

2 1

3 5

4 2

Step 2 computes the following next\_code values:

Крок 2:

N next\_code[N]

- ------------

1 0

2 0

3 2

4 14

Step 3 produces the following code values:

Крок 3 :

Символ Довжина Код

------ ------ ----

A 3 010

B 3 011

C 3 100

D 3 101

E 3 110

F 2 00

G 4 1110

H 4 1111

3.3. Alphabet sizes

Prefix codes are used for different purposes in the brotli format,

and each purpose has a different alphabet size. For literal codes the

alphabet size is 256. For insert-and-copy length codes the alphabet

size is 704. For block count codes, the alphabet size is 26. For

distance codes, block type codes, and the prefix codes used in

compressing the context map, the alphabet size is dynamic and is

based on parameters defined in later sections. The following table

summarizes the alphabet sizes for the various prefix codes and the

sections where they are defined.

Префіксні коди використовують для різних цілей в алгоритмі,і для кожної цілі використовують різний розмір алфавіту.Для літеральних кодів -256.Для кодів що позначають вставку та копіювання -704.Для лічильників блоків -26.Для кодів відстаней ,кодів типів блоків,префіксні коди є динамічними.В наступній таблиці зазначені розміри.

Alakuijala & Szabadka Expires November 4, 2016 [Page 12]

Internet-Draft Brotli May 2016

+-----------------+-------------------------+------------+

| Prefix code | Alphabet size | Definition |

+-----------------+-------------------------+------------+

| literal | 256 | |

+-----------------+-------------------------+------------+

| distance | 16 + NDIRECT + | Section 4. |

| | (48 << NPOSTFIX) | |

+-----------------+-------------------------+------------+

| insert-and-copy | 704 | Section 5. |

| length | | |

+-----------------+-------------------------+------------+

| block count | 26 | Section 6. |

+-----------------+-------------------------+------------+

| block type | NBLTYPESx + 2, | Section 6. |

| | (where x is I, L, or D) | |

+-----------------+-------------------------+------------+

| context map | NTREESx + RLEMAXx | Section 7. |

| | (where x is L or D) | |

+-----------------+-------------------------+------------+

3.4. Simple prefix codes

The first two bits of the compressed representation of each prefix

code distinguish between simple and complex prefix codes.

Перші два біти стисненого представлення кожного префіксного коду визначають чи це простий чи складний префікс ний код.

If this

value is 1, then a simple prefix code follows as described in this

section.

У випадку 1 - це простий префікс ний код.

Otherwise, a complex prefix code follows as described in

Section 3.5.

A simple prefix code can have only up to four symbols with non-zero

code length.

Простий префікс ний код може мати лише не більше чотирьох символів з ненульовою довжиною коду.

The format of the simple prefix code is as follows:

В загальному ,формат простого префіксного коду

2 bits: value of 1 indicates a simple prefix code

2 біти:зна=чення в 1 означає що це простий код

2 bits: NSYM - 1, where NSYM = # of symbols coded

2 біти : NSYM - 1,де NSYM = # символів закодованих

NSYM symbols, each encoded using ALPHABET\_BITS bits

1 bit: tree-select, present only for NSYM = 4

1 біт :наявний лише коли NSYM = 4

The value of ALPHABET\_BITS depends on the alphabet of the prefix

code: it is the smallest number of bits that can represent all

symbols in the alphabet.

Кількість бітів для позначення алфавіту –найменша кілкість бітів яка може представляти символи в алфавіті.

E.g. for the alphabet of literal bytes,

ALPHABET\_BITS is 8. The value of each of the NSYM symbols above is

the value of the ALPHABETS\_BITS width integer value. If the integer

value is greater than or equal to the alphabet size, or the value is

identical to a previous value, then the stream should be rejected as

invalid.

Note that the NSYM symbols may not be presented in sorted order.

Alakuijala & Szabadka Expires November 4, 2016 [Page 13]

Internet-Draft Brotli May 2016

Prefix codes of the same bit length must be assigned to the symbols

in sorted order.

Префіксні коди однакової бітової довжини мають бути записані в тому ж порядку що й їхні символи в відсортованому за зростанням порядку.

The (non-zero) code lengths of the symbols can be reconstructed as

follows:

Ненульові довжин кодів символів можуть бути реконструйовані наступним чином

\* if NSYM = 1, the code length for the one symbol is zero --

when encoding this symbol in the

compressed data stream using this prefix code, no

actual bits are emitted.

Якщо NSYM=1то довжина коду для 1 символу -0.при кодуванні цього символу в стиснених даних ніяких додаткових бітів не додають.

Similarly, when decoding a symbol

using this prefix code, no bits are read and the one symbol

is returned.

\* if NSYM = 2, both symbols have code length 1.

Якщо NSYM = 2,всі символи мають довжину 1.

\* if NSYM = 3, the code lengths for the symbols are 1, 2, 2 in

the order they appear in the representation of the simple

prefix code.

Якщо NSYM = 3,довжини кодів для символів наступні :1,2,2 в порядку в якому вони з’являються в представленні простого префіксного коду.

\* if NSYM = 4, the code lengths (in order of symbols decoded)

depend on the tree-select bit: 2, 2, 2, 2 (tree-select bit 0),

or 1, 2, 3, 3 (tree-select bit 1).

Якщо NSYM = 4,довжини кодів (в порядку символи закодовані)залежить від двох комбінацій довжин :2,2,2,2 або 1,2,3,3.

3.5. Complex prefix codes

Склдані префіксні коди

A complex prefix code is a canonical prefix code, defined by the

sequence of code lengths, as discussed in Section 3.2., above. For

even greater compactness, the code length sequences themselves are

compressed using a prefix code. The alphabet for code lengths is as

follows:

Складним префікс ним кодом називають канонічний префікс ний код визначений як послідовність довжин кодів визначених в секції 3.2.Для більшої зручності довжини послідовностей стискають використовуючи префіксні коди.Алфавіт для довжин кодів наступний :

0..15: Represent code lengths of 0..15

0..15:Для кодів довжини 0..15

16: Copy the previous non-zero code length 3..6 times

16:Скопіювати попередню ненульову довжину коду 3..6 разів

The next 2 bits indicate repeat length

(0 = 3, ... , 3 = 6)

Наступні 2 біти визначають довжину повторення

(0 = 3, ... , 3 = 6)

If this is the first code length, or all previous

code lengths are zero, a code length of 8 is

repeated 3..6 times

Якщо це перший код довжини ,або всі попредні мають нульову довжину ,код довжини 8 повторюють від 3 до 6 разів.

A repeated code length code of 16 modifies the

repeat count of the previous one as follows:

Повтрюваний код 16 змінює лічильник повторень наступним чином

repeat count = (4 \* (repeat count - 2)) +

(3..6 on the next 2 bits)

Example: Codes 7, 16 (+2 bits 11), 16 (+2 bits 10)

will expand to 22 code lengths of 7

буде розширений до 22 кодів довжини 7

(1 + 4 \* (6 - 2) + 5)

17: Repeat a code length of 0 for 3..10 times.

17:Повторити довжину коду 0 від 3 до 10 разів

(3 bits of length)

A repeated code length code of 17 modifies the

Alakuijala & Szabadka Expires November 4, 2016 [Page 14]

Internet-Draft Brotli May 2016

Код 17 змінює лічильник повторень наступним чином

repeat count of the previous one as follows:

repeat count = (8 \* (repeat count - 2)) +

(3..10 on the next 3 bits)

Note that a code of 16 that follows an immediately preceding 16

modifies the previous repeat count, which becomes the new repeat

count. The same is true for a 17 following a 17. A sequence of three

or more 16 codes in a row or three of more 17 codes in a row is

possible, modifying the count each time. Only the final repeat count

is used. The modification only applies if the same code follows. A 16

repeat does not modify an immediately preceding 17 count nor vice

versa.

Варто зазначити що код 16 який передує змінює попередній лічильник повторень ,який стає новим лічильником повтрень.Аналогічно для 17.Послідовність з кількох кодів 16 або ж для 17 також можлива,якщо постійно будуть відбуватися зміни лічильника.Відповідно лише останній лічильнки буде остаточно використаний .

A code length of 0 indicates that the corresponding symbol in the

alphabet will not occur in the compressed data, and should not

participate in the prefix code construction algorithm given earlier.

Довжина коду в 0 означає що відповідний символ в алфавіті не з’явиться в стиснених даних ,і він не бере участі в обчисленнях префіксного коду в алгоритмі описаного вище.

A complex prefix code must have at least two non-zero code lengths.

Складний префіксние код мусить мати щонайменше дві ненульові довжин коду.

The bit lengths of the prefix code over the code length alphabet are

compressed with the following variable length code (as it appears in

the compressed data, where the bits are parsed from right to left):

Бітові довжини префіксного коду стиснені використовуючи наступне кодування змінної довжини(variable length coding).

Symbol Code

------ ----

0 00

1 0111

2 011

3 10

4 01

5 1111

We can now define the format of the complex prefix code as follows:

Тепер можна визначити формат складного префікс них кодів наступним чином.

2 bits: HSKIP, values of 0, 2, or 3 represent the respective

number of skipped code lengths. The skipped lengths

are taken to be zero. (An HSKIP of 1 indicates a

Simple prefix code.)

2 біти:HSKIP,значення 0,2,3 представляють відповідне число пропущених довжин кодів.пропущені довжини всі за замовчуванням нульові.(HSKIP -1 .означає що наявний прости префіксний код).

Code lengths for symbols in the code length alphabet given

just above, in the order: 1, 2, 3, 4, 0, 5, 17, 6, 16, 7,

8, 9, 10, 11, 12, 13, 14, 15.

Довжини кодів для символів в алфавіті заданому вище,наступні: 1, 2, 3, 4, 0, 5, 17, 6, 16, 7,

8, 9, 10, 11, 12, 13, 14, 15.

If HSKIP is 2, then the

За умови що HSKIP дорівнює 2 ,тоді довжини кодів для символів 1 та 2 дорівнюють 0 ,та перша довжина коду встановлена для символу 3.За умови зщо HSKIPдорівнює 3 ,тоді довжина коду для символу 3 також нульова ,та перша довжина коду встановлена вже відповідно для символу 4.

code lengths for symbols 1 and 2 are zero, and the first

code length is for symbol 3. If HSKIP is 3, then the code

length for symbol 3 is also zero, and the first code length

is for symbol 4.

The code lengths of code length symbols are between 0 and

5, and they are represented with 2..4 bits according to

the variable length code above. A code length of 0 means

the corresponding code length symbol is not used.

Кодові довжини для символів між 0 та 5 представлені від 2 до 4 бітів залежно від коду зміннох довжини.Довжина коду 0 означає що відповідний кодовий символ не використаний .

If HSKIP is 2 or 3, a respective number of leading code

lengths are implicit zeros and are not present in the

code lengths sequence above.

Якщо HSKIP 2 або 3 ,відповідне число основних(перших у розташування при зчитуванні )дожвни коду є нульовими та не пристуні вд послідовності довжин коду визначеній вище.

If there are at least two non-zero code lengths, any

trailing zero code lengths are omitted, i.e. the last

code length in the sequence must be non-zero. In this

case, the sum of (32 >> code length) over all the non-zero

code lengths must equal to 32.

Якщо у нас наявно принаймні дві ненульові довжини коду,будя-яка нульова довжина коду опущена.Тобто остання дожвина коду мусить бути ненульовою.В цьому випадку ,сума всіх ненульових довжин коду мусить дорівнювати 32.

If the lengths have been read for the entire code length

alphabet and there was only one non-zero code length,

then the prefix code has one symbol whose code has zero

length. In this case, that symbol results in no bits

being emitted by the compressor, and no bits consumed by

the decompressor.

Якщо жовжини були прочитані для всього алфавіту довжин коду і є лише одна ненульова довжина коду ,тоді префіскний код має один символ чий префікс ний має один символ має нульову довжину.В цьому випадку,для цьому символу не визначають додаткових бітів для кодування компрессором ,та жодні біти не отримані декомпресором.

That single symbol is immediately

returned when this code is decoded.

Одиничний символ одразу буде повернутий коли цей код буде розшифрований.

An example of where this occurs is if the

entire code to be represented has symbols of length 8.

Приклад ,у якому весь код представлений символами довжин 8.

E.g. a literal code that represents all literal values

with equal probability.

Тобто ми маємо літеральний код який представляє значення цих літералів з однаковою ймовірністю.

In this case the single symbol

is 16, which repeats the previous length.

В цьому випадку єдиним символом є 16,який повторюють з попередньою частотою.The previous

length is taken to be 8 before any code length code

lengths are read.

Попередню частоту взяту як 8 зчитують до того як буде прочитана довжина коду.

Sequence of at most alphabet size code lengths symbols, encoded

using the code length prefix code.

Any trailing 0 or 17 must be

omitted, i.e. the last encoded code length symbol must be

between 1 and 16. Будь-який перший символ (0 або 17) має бути пропущений,так як остання закодована довжина символу мусить бути між 1 до 16.

The sum of (32768 >> code length) over

all the non-zero code lengths in the alphabet, including

those encoded using repeat code(s) of 16, must equal to

32768.Сума довжин по всім ненульовим довжинам коду в алфавіті ,разом з закодованим кодом повторення 16 ,мають бути рівні 32768.

If the number of times to repeat the previous length

or repeat a zero length would result in more lengths in

total than the number of symbols in the alphabet, then the

stream should be rejected as invalid.

Якщо кількість повторень попередньої довжини або повторення нульової довжини матомиуть в сумі більше число аніж кількість всіх символів в алфавіті,то потік має бути відкинутим .

4. Encoding of distances

Кодування відстаней

As described in Section 2., one component of a compressed meta-block

is a sequence of backward distances.

Як зазначено в параграфі 2 ,одним з компонентів стисненого мат-блоку є пслоідовність відстаней до прочитаних раніше ідентичних послідовностей.

In this section we provide the

details to the encoding of distances.

В цьому параграфі ми ознайомимемося з деталями процесу стиснення відстаней.

Each distance in the compressed data part of a meta-block is

represented with a pair <distance code, extra bits>.

Кожна відстань в стиснених даних мета-блоку представлена у вигляді пари <код відстані,додаткові біти>.

The distance

code and the extra bits are encoded back-to-back, the distance code

Коди відстані та додаткові біти закодовані разом.

Alakuijala & Szabadka Expires November 4, 2016 [Page 16]

Internet-Draft Brotli May 2016

is encoded using a prefix code over the distance alphabet, while the

extra bits value is encoded as a fixed-width integer value.

Код відстані кодують використовуючи той самий префікс ний код по всьому алфавіту відстаней .Додаткові біти кодують використовуючи цілочисленні значення фіксованої ширини.

The

number of extra bits can be 0..24, and it is dependent on the

distance code.

Кількість додаткових бітів може бути від 0 до 24 залежно від коду відстані.

To convert a distance code and associated extra bits to a backward

distance, we need the sequence of past distances and two additional

parameters, the number of "postfix bits", denoted by NPOSTFIX (0..3),

and the number of direct distance codes, denoted by NDIRECT (0..120).

Щоб перевести код відстані та відповідні додаткові біти до відстані до попереднього запису слід отримати послідовність з попередніх відстаней та два додаткові параметри .Перший-кількість постфікс них бітів ,позначених NPOSTFIX (0..3).Другий – кількість прямих кодів відстані позначених NDIRECT (0..120). Both of these parameters are encoded in the meta-block header.

Обдива параметри закодовані в заголовку мета-блоку.

We

will also use the following derived parameter:

Також ми обсилюємо наступний параметр

POSTFIX\_MASK = (1 << NPOSTFIX) - 1

The first 16 distance symbols are special symbols that reference past

distances as follows:

Перші 16 символів для позначення відстані є спеціальними символами для писилання до попередніх відстаней наступним чином:

0: last distance

1: second-to-last distance

2: third-to-last distance

3: fourth-to-last distance

4: last distance - 1

5: last distance + 1

6: last distance - 2

7: last distance + 2

8: last distance - 3

9: last distance + 3

10: second-to-last distance - 1

11: second-to-last distance + 1

12: second-to-last distance - 2

13: second-to-last distance + 2

14: second-to-last distance - 3

15: second-to-last distance + 3

0: остання відстань

1: драгу з кінця відстань

2: третя зкінця відстань

3: четверта з кінця відстань

4: остання відстань - 1

5 остання відстань + 1

6: остання відстань - 2

7: остання відстань + 2

8: остання відстань - 3

9: остання відстань + 3

10: остання з кінця - 1

11: second-to-last distance + 1

12: second-to-last distance - 2

13: second-to-last distance + 2

14: second-to-last distance - 3

15: second-to-last distance + 3

The ring buffer of four last distances is initialized by the values

16, 15, 11, and 4 (i.e. the fourth-to-last is set to 16, the third-

to-last to 15, the second-to-last to 11, and the last distance to 4)

at the beginning of the \*stream\* (as opposed to the beginning of the

meta-block) and it is not reset at meta-block boundaries.

Кільцевий буфер останніх чотирьох відстаней ініціалізований значеннями 16,15,11 та 4 на початку потоку .

When a

distance symbol 0 appears, the distance it represents (i.e. the last

distance in the sequence of distances) is not pushed to the ring

buffer of last distances, in other words, the expression "(second,

third, fourth)-to-last distance" means the (second, third,

fourth)-to-last distance that was not represented by a 0 distance

symbol. Коли трапляється символ відстані 0 ,відстань не записують до кільцевого буфера останніх відстаней ,іншими словами ,вираз(другий,третій,четвертий до останньої відстані означає що друга ,третя ,четверта з кінця відстань не представлені символом відстані 0.

Similarly, distances that represent static dictionary words

(see Section 8.) are not pushed to the ring buffer of last distances.

Подібно,відстані які представляють статичні словники слів не записують до кільцевого буфера останніх відстаней.

If a special distance symbol resolves to a zero or negative value,

Alakuijala & Szabadka Expires November 4, 2016 [Page 17]

Internet-Draft Brotli May 2016

the stream should be rejected as invalid.

If NDIRECT is greater than zero, then the next NDIRECT distance

symbols, from 16 to 15 + NDIRECT, represent distances from 1 to

NDIRECT. Neither the special distance symbols, nor the NDIRECT

direct distance symbols are followed by any extra bits.

Якщо NDIRECT>0,тоді наступні символи NDIRECT від 16 до 15+NDIRECT представляють відстані выд 1 до NDIRECT.Ані спеціальні символи для відстаней ,ані NDIRECT прямі символи відстаней не мають після себе додаткових бітів.

Distance symbols 16 + NDIRECT and greater all have extra bits, where

the number of extra bits for a distance symbol "dcode" is given by

the following formula:

Символи відстаней 16+NDIRECT та більше,мають додаткові біти для кількість додаткових бітів для символу "dcode" визначають наступною формулою.

ndistbits = 1 + ((dcode - NDIRECT - 16) >> (NPOSTFIX + 1))

The maximum number of extra bits is 24, therefore the size of the

distance symbol alphabet is (16 + NDIRECT + (48 << NPOSTFIX)).

Максимальне число додаткових бітів -24.Розмір алфавіту відстаней дорівнює .

(16 + NDIRECT + (48 << NPOSTFIX)).

Given a distance symbol "dcode" (>= 16 + NDIRECT), and extra bits

"dextra", the backward distance is given by the following formula:

За симолом dcode" та додатковими бітами "dextra",відстань визначають наступною формулою

hcode = (dcode - NDIRECT - 16) >> NPOSTFIX

lcode = (dcode - NDIRECT - 16) & POSTFIX\_MASK

offset = ((2 + (hcode & 1)) << ndistbits) - 4

distance = ((offset + dextra) << NPOSTFIX) + lcode + NDIRECT + 1

5. Encoding of literal insertion lengths and copy lengths

Кодування довжин вставок та копіювання літералів

As described in Section 2., the literal insertion lengths and

backward copy lengths are encoded using a single prefix code. This

section provides the details to this encoding.

Як зазначено в другому параграфі,довжин вставок та копіювання закодовані простим префікс ним кодом.

Each <insertion length, copy length> pair in the compressed data part

of a meta-block is represented with the following triplet:

Кожна пара <довжина вставки,довжина копыювання>-це пара стиснених даних в мета-блоці представлена наступним триплетом:

<insert-and-copy length code, insert extra bits, copy extra bits>

<код вставки та копіювання ,додаткові біти вставки ,додаткові біти копіювання >

The insert-and-copy length code, the insert extra bits, and the copy

extra bits are encoded back-to-back, the insert-and-copy length code

is encoded using a prefix code over the insert-and-copy length code

alphabet, while the extra bits values are encoded as fixed-width

integer values. Всі коди записані разом ,зокрема перший код представляють завдяки префіксноому коду по алфавіту ,в той час як два наступні завдяки цілочисленим значенням фіксованої довжини.

he number of insert and copy extra bits can be

0..24, and they are dependent on the insert-and-copy length code.

Число додаткових бітів коливається від 0 до 24 і воно залежить від коду вставки та копіювання .

Some of the insert-and-copy length codes also express the fact that

the distance symbol of the distance in the same command is 0, i.e.

the distance component of the command is the same as that of the

previous command.

In this case, the distance code and extra bits for

the distance are omitted from the compressed data stream.

Alakuijala & Szabadka Expires November 4, 2016 [Page 18]

Internet-Draft Brotli May 2016

We describe the insert-and-copy length code alphabet in terms of the

(not directly used) insert length code and copy length code

alphabets. The symbols of the insert length code alphabet, along with

the number of insert extra bits, and the range of the insert lengths

are as follows:

Опис алфавіту кодів вставки та копіювання використовуючи алфавіти довжин кодів вставки та довжин кодів копіювання.

Extra Extra Extra

Code Bits Lengths Code Bits Lengths Code Bits Lengths

---- ---- ------- ---- ---- ------- ---- ---- -------

0 0 0 8 2 10..13 16 6 130..193

1 0 1 9 2 14..17 17 7 194..321

2 0 2 10 3 18..25 18 8 322..577

3 0 3 11 3 26..33 19 9 578..1089

4 0 4 12 4 34..49 20 10 1090..2113

5 0 5 13 4 50..65 21 12 2114..6209

6 1 6,7 14 5 66..97 22 14 6210..22593

7 1 8,9 15 5 98..129 23 24 22594..16799809

The symbols of the copy length code alphabet, along with the number

of copy extra bits, and the range of copy lengths are as follows:

Extra Extra Extra

Code Bits Lengths Code Bits Lengths Code Bits Lengths

---- ---- ------- ---- ---- ------- ---- ---- -------

0 0 2 8 1 10,11 16 5 70..101

1 0 3 9 1 12,13 17 5 102..133

2 0 4 10 2 14..17 18 6 134..197

3 0 5 11 2 18..21 19 7 198..325

4 0 6 12 3 22..29 20 8 326..581

5 0 7 13 3 30..37 21 9 582..1093

6 0 8 14 4 38..53 22 10 1094..2117

7 0 9 15 4 54..69 23 24 2118..16779333

To convert an insert-and-copy length code to an insert length code

and a copy length code, the following table can be used:

Щоб ковертувати код довжини вставки та копіювання до коду довжини вставки та коду довжини копіювання ,використовують настпну таблицю:

Alakuijala & Szabadka Expires November 4, 2016 [Page 19]

Internet-Draft Brotli May 2016

Insert

length Copy length code

code 0..7 8..15 16..23

+----------+----------+

| | |

0..7 | 0..63 | 64..127 | <--- distance symbol 0

| | |

+----------+----------+----------+

| | | |

0..7 | 128..191 | 192..255 | 384..447 |

| | | |

+----------+----------+----------+

| | | |

8..15 | 256..319 | 320..383 | 512..575 |

| | | |

+----------+----------+----------+

| | | |

16..23 | 448..511 | 576..639 | 640..703 |

| | | |

+----------+----------+----------+

First, look up the cell with the 64 value range containing the

insert-and-copy length code, this gives the insert length code and

the copy length code ranges, both 8 values long. Спочатку відбувається пошук по таблиці за даними кодами вставки та копіювання ,потім отримують код довжини вставки та код довжини копіювання(кожне має 8 значень довжини).

The copy length

code within its range is determined by bits 0..2 (counted from the

lsb) of the insert-and-copy length code. Код довжини коду визначають за двома бітами(рахуючи з найменш важливого) від коду довжини вставки та копіювання.

The insert length code

within its range is determined by bits 3..5 (counted from the lsb) of

the insert-and-copy length code.

Код довжини вставки визначають в діапазоні визначеному бітами на позиціях 3-5 коду довжини вставки та копіювання .

Given the insert length and copy

length codes, the actual insert and copy lengths can be obtained by

reading the number of extra bits given by the tables above.

За даними кодами можна отримати дісйну довжину вставки та копіювання якщо прочитати додаткові біти дані в таблиці вище.

If the insert-and-copy length code is between 0 and 127, the distance

code of the command is set to zero (the last distance reused).

Для коду довжини вставки та видалення між 0 та 127,код відстані команд встановлений як 0 ,тобто використовують останню відстань.

6. Encoding of block switch commands

Кодування команд блоку

As described in Section 2., a block-switch command is a pair <block

type, block count>. These are encoded in the compressed data part of

the meta-block, right before the start of each new block of a

particular block category.

Як зазначено раніше,команди блоку є парою <тип блоку,лічильник блоку>.Їх кодують в стисненій частині мета-блоку ,перед початокм кожного нового блоку кожної блочної категорії.

Each block type in the compressed data is represented with a block

type code, encoded using a prefix code over the block type code

alphabet.Кожен тип блоку стиснених даних представлений код типу блоку,закодованим префікс ним кодом з алфавіту типів блоку.

A block type symbol 0 means that the new block type is the

same as the type of the previous block from the same block category,

i.e. the block type that preceded the current type, while a block

type symbol 1 means that the new block type equals the current block

type plus one. Символ 0 означає що новий тип блоку однаковий до попереднього з цієї ж категорії.Символ 1 означає що новий тип блоку дорівнює поточному блоку плюч ще один тип блоку.

If the current block type is the maximal possible,

Alakuijala & Szabadka Expires November 4, 2016 [Page 20]

Internet-Draft Brotli May 2016

then a block type symbol of 1 results in wrapping to a new block type

of 0. Block type symbols 2..257 represent block types 0..255

respectively.

Символи 2-257 представляють типи блоків від 0 до 255 відповідно.

The previous and current block types are initialized to

1 and 0, respectively, at the end of the meta-block header.

Попередній та поточний типи ініціалізовна 1 та 0,наприкінці заголовку мета-блоку.

Since the first block type of each block category is 0, the block

type of the first block-switch command is not encoded in the

compressed data. Оскільки кожен тип блоку кожної категоріх дорівнює 0,то тип блоку першої команди не закодований в стиснених даних.

If a block category has only one block type, the

block count of the first block-switch command is also omitted from

the compressed data, otherwise it is encoded in the meta-block

header.

Якщо кожна блочна категорія має лише один тип блоку ,то лічильник блоку першої команди пропускають,в іншому випадку його загодовують в заголовок мета-блоку.

Since the end of the meta-block is detected by the number of

uncompressed bytes produced, the block counts for any of the three

categories need not count down to exactly zero at the end of the

meta-block.

The number of different block types in each block category, denoted

by NBLTYPESL, NBLTYPESI, and NBLTYPESD for literals, insert-and-copy

lengths, and distances, respectively, is encoded in the meta-block

header, and it must equal to the largest block type plus one in that

block category. In other words, the set of literal, insert-and-copy

length, and distance block types must be [0..NBLTYPESL-1],

[0..NBLTYPESI-1], and [0..NBLTYPESD-1], respectively. From this it

follows that the alphabet size of literal, insert-and-copy length,

and distance block type codes is NBLTYPESL + 2, NBLTYPESI + 2, and

NBLTYPESD + 2, respectively.

Кількість різних типів блоку в кожній категорії блоку ,означений як NBLTYPESL, NBLTYPESI, and NBLTYPESD для літералів ,довжин та відстаней відповідно ,закодована в заголовк мета-блоку.Вона також має дорівнювати номеру найбільшого типу блоку плюч один в даній категорії блоку.Іншими словами,множина типів має бути впорядкован так : [0..NBLTYPESL-1],

[0..NBLTYPESI-1], and [0..NBLTYPESD-1].З цього випливає що розмір алфавіту коді типів блоку має дорівнювати NBLTYPESL + 2, NBLTYPESI + 2, and

NBLTYPESD + 2 відопвідно.

Each block count in the compressed data is represented with a pair

<block count code, extra bits>.Для кожного лічильника блоку в стиснених даних є представлення у вигляді пари <код лічильника,додаткові біти>.

The block count code and the extra

bits are encoded back-to-back, the block count code is encoded using

a prefix code over the block count code alphabet, while the extra

bits value is encoded as a fixed-width integer value.

Код лычильника та додаткові біти закодовані знову д таки разом.При цьому використовують алфавіт кодів лічильників блоку та цілочисленні значення для додаткових бітів.

The number of

extra bits can be 0..24, and it is dependent on the block count code.

Кіліькість додаткових бітів змінюється від 0 до 24 залежно від лічильника блоку.

The symbols of the block count code alphabet, along with the number

of extra bits, and the range of block counts are as follows:

Таблиця алфавіту кодів лічильників блоку з кількістю додаткових бітів та діапазоном лічильників блоку .

Extra Extra Extra

Code Bits Lengths Code Bits Lengths Code Bits Lengths

---- ---- ------- ---- ---- ------- ---- ---- -------

0 2 1..4 9 4 65..80 18 7 369..496

1 2 5..8 10 4 81..96 19 8 497..752

2 2 9..12 11 4 97..112 20 9 753..1264

3 2 13..16 12 5 113..144 21 10 1265..2288

4 3 17..24 13 5 145..176 22 11 2289..4336

5 3 25..32 14 5 177..208 23 12 4337..8432

6 3 33..40 15 5 209..240 24 13 8433..16624

7 3 41..48 16 6 241..304 25 24 16625..16793840

8 4 49..64 17 6 305..368

The first block-switch command of each block category is special in

the sense that it is encoded in the meta-block header, and as

described earlier, the block type code is omitted since it is an

implicit zero.

7. Context modeling

Моделювання контексту

As described in Section 2., the prefix tree used to encode a literal

byte or a distance code depends on the block type and the context ID.

Як зазначено раніше префікс не дерево використовують для кодування літеральних байтів та кодів відстаней залежно від типу блоку та контекстного ID.

This section specifies how to compute the context ID for a particular

literal and distance code, and how to encode the context map that

maps a <block type, context ID> pair to the index of a prefix code in

the array of literal and distance prefix codes.

7.1. Context modes and context ID lookup for literals

Види контексту та контекстні ID для літералів

The context for encoding the next literal is defined by the last two

bytes in the stream (p1, p2, where p1 is the most recent byte),

regardless of whether these bytes are produced by uncompressed meta-

blocks, backward references, static dictionary references, or by

literal insertions. At the start of the stream p1 and p2 are

initialized to zero.

Контексту для кодування кожного наступного літералу визначають за двома останніми байтами в потоці незалежно від того,чи ці байти створені завдяки посиланнями на нестиснені мета-блоки ,посилання на статичні словники чи посилання на літеральні вставки.На початку потоку ці значення очевидно нульові.

There are four methods, called context modes, to compute the Context

ID Є чотри види контексту за допомогою яких рахують контекстний ID

:

\* LSB6, where the Context ID is the value of six

least-significant bits of p1,

LSB6-беруть значення шести найменш важливих бітів байту р1.

\* MSB6, where the Context ID is the value of six

most-significant bits of p1,

MSB6-беруть значення шести найбільш важливих бітів р1.

\* UTF8, where the Context ID is a complex function of p1, p2,

optimized for text compression

UTF8-обраховують за допомогою функції від р1 та р1 для стискання даних.

, and

\* Signed, where Context ID is a complex function of p1, p2,

optimized for compressing sequences of signed integers.

Знаковий метод-використовують функцію від р1 та р2 для стиснених послідовностей цілих знакових чисел.

Alakuijala & Szabadka Expires November 4, 2016 [Page 22]

Internet-Draft Brotli May 2016

The Context ID for the UTF8 and Signed context modes is computed

using the following lookup tables Lut0, Lut1, and Lut2.

Lut0 :=

0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 0, 4, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

8, 12, 16, 12, 12, 20, 12, 16, 24, 28, 12, 12, 32, 12, 36, 12,

44, 44, 44, 44, 44, 44, 44, 44, 44, 44, 32, 32, 24, 40, 28, 12,

12, 48, 52, 52, 52, 48, 52, 52, 52, 48, 52, 52, 52, 52, 52, 48,

52, 52, 52, 52, 52, 48, 52, 52, 52, 52, 52, 24, 12, 28, 12, 12,

12, 56, 60, 60, 60, 56, 60, 60, 60, 56, 60, 60, 60, 60, 60, 56,

60, 60, 60, 60, 60, 56, 60, 60, 60, 60, 60, 24, 12, 28, 12, 0,

0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,

0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,

0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,

0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,

2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,

2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,

2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3,

2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3

Lut1 :=

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1,

1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1,

1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2

Lut2 :=

0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,

Alakuijala & Szabadka Expires November 4, 2016 [Page 23]

Internet-Draft Brotli May 2016

4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,

4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,

4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,

4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,

5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,

5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,

5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,

6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7

The lengths and the CRC-32 check values (see Appendix C.) of each of

these tables as a sequence of bytes are as follows:

Table Length CRC-32

----- ------ ------

Lut0 256 0x8e91efb7

Lut1 256 0xd01a32f4

Lut2 256 0x0dd7a0d6

Given p1 is the last uncompressed byte and p2 is the second-to-last

uncompressed byte, the context IDs can be computed as follows:

Формули для обчислення контекстний ID

For LSB6: Context ID = p1 & 0x3f

For MSB6: Context ID = p1 >> 2

For UTF8: Context ID = Lut0[p1] | Lut1[p2]

For Signed: Context ID = (Lut2[p1] << 3) | Lut2[p2]

From the lookup tables defined above and the operations to compute

the context IDs, we can see that context IDs for literals are in the

range of 0..63.

The context modes LSB6, MSB6, UTF8, and Signed are denoted by

integers 0, 1, 2, 3.

A context mode is defined for each literal block type and they are

stored in a consecutive array of bits in the meta-block header,

always two bits per block type.

7.2. Context ID for distances

Контекстний ID для відстаней

The context for encoding a distance code is defined by the copy

length corresponding to the distance. The context IDs are 0, 1, 2,

and 3 for copy lengths 2, 3, 4, and more than 4, respectively.

Контекст для кодування кодів відстані визначений завдяки довжині копіювання яка відповідає даній відстані. Контекстні ID -0,1,2,3 для відстаней копіювання и2.3.4 та тих які більші за 4.

7.3. Encoding of the context map

Кодуваняня контекстного відображення

There are two context maps, one for literals and one for distances.

Наявні два контекстних відображення .Одни для літералів та інший для відстаней .

The size of the context map is 64 \* NBLTYPESL for literals, and 4 \*

NBLTYPESD for distances.

Їхні розміри 64 \* NBLTYPESL для літералів , and 4 \*

NBLTYPESD для відстаней .

Each value in the context map is an integer.Кожне

Alakuijala & Szabadka Expires November 4, 2016 [Page 24]

Internet-Draft Brotli May 2016

between 0 and 255, indicating the index of the prefix code to be used

when encoding the next literal or distance.

Кожне значення у відображення є цілим числом від 0 до 255,яе виначає індекс префіксного коду який використовували при кодуванні наступного літералу та відстані .

The context maps are two-dimensional matrices, encoded as one-

dimensional arrays:

Контекстні відображення є насправді двовимірним матрицями ,закодованими як одновимірні масиви.

CMAPL[0..(64 \* NBLTYPESL - 1)]

CMAPD[0..(4 \* NBLTYPESD - 1)]

The index of the prefix code for encoding a literal or distance code

with block type, BTYPE\_x, and context ID, CIDx, is:

Індекс префіксного коду для кодування літералу та коду відстані для типу блоку BTYPE\_x та контекстногоID CID наступний :

index of literal prefix code = CMAPL[64 \* BTYPE\_L + CIDL]

index of distance prefix code = CMAPD[4 \* BTYPE\_D + CIDD]

The values of the context map are encoded with the combination of run

length encoding for zero values and prefix coding.Значення контекстного відображення кодують комбінацією кодування довжин серій для нульових значень та преіфксному кодуванню.

Let RLEMAX denote

the number of run length codes and NTREES denote the maximum value in

the context map plus one. Нехай RLEMAX –число кодыв довжин серій та NTREES – максимальне значення в контекстному відображення плюс 1.

NTREES must equal the number of different

values in the context map, in other words, the different values in

the context map must be the [0..NTREES-1] interval.

Звідси NTREES дорівнює кількості попарно різних значень в контекстному відображенні .

The alphabet of

the prefix code has the following RLEMAX + NTREES symbols:

Алфавіт префіксного коду має RLEMAX + NTREES символів

0: value zero

1: repeat a zero 2 to 3 times, read 1 bit for repeat length

2: repeat a zero 4 to 7 times, read 2 bits for repeat length

...

RLEMAX: repeat a zero (1 << RLEMAX) to (1 << (RLEMAX+1))-1

times, read RLEMAX bits for repeat length

RLEMAX + 1: value 1

...

RLEMAX + NTREES - 1: value NTREES - 1

If RLEMAX = 0, the run length coding is not used, and the symbols of

the alphabet are directly the values in the context map. Якщо RLEMAX=0 ,то кодування довжин серій не використовують ,та симоли алфавіту є значеннями контекстного відображення .

We can now

define the format of the context map (the same format is used for

literal and distance context maps):

Отже ,тепер можна визначити формат контекстного відображення :

1..5 bits: RLEMAX, 0 is encoded with one 0 bit, and values

1..16 are encoded with bit pattern xxxx1 (so 01001

is 5)З 1 по 5 біт :

Prefix code with alphabet size NTREES + RLEMAX

Context map size values encoded with the above prefix code

and run length coding for zero values.

If a run length

would result in more lengths in total than the size of

the context map, then the stream should be rejected as

Alakuijala & Szabadka Expires November 4, 2016 [Page 25]

Internet-Draft Brotli May 2016

invalid.

1 bit: IMTF bit, if set, we do an inverse move-to-front

transform on the values in the context map to get

the prefix code indexes

Note that RLEMAX may be larger than the value necessary to represent

the longest sequence of zero values. Also, the NTREES value is

encoded right before the context map as described in Section 9.2.

We define the inverse move-to-front transform used in this

specification by the following C language function:

void InverseMoveToFrontTransform(uint8\_t\* v, int v\_len) {

uint8\_t mtf[256];

int i;

for (i = 0; i < 256; ++i) {

mtf[i] = (uint8\_t)i;

}

for (i = 0; i < v\_len; ++i) {

uint8\_t index = v[i];

uint8\_t value = mtf[index];

v[i] = value;

for (; index; --index) {

mtf[index] = mtf[index - 1];

}

mtf[0] = value;

}

}

Note that the inverse move-to-front transform will not produce values

outside the [0..NTREES-1] interval.

8. Static dictionary

Статичний словник

At any given point during decoding the compressed data, a reference

to a duplicated string in the uncompressed data produced so far has a

maximum backward distance value, which is the minimum of the window

size and the number of uncompressed bytes produced.

В будь-який момент часу під час декодування стиснениз даних ,посилання на повторену стрічку в нестиснених даних має максимальне значення відстані ,яке є мінімальним розміром розсувного вікна та числа нестиснених даних які вже декодовані.

However, decoding

a distance from the compressed stream, as described in Section 4.,

can produce distances that are greater than this maximum allowed

value.

Проте,декодування відстані з потоку стиснених даних може створити відтані які є більшими за максимальне допустиме значення .

In this case, the distance is treated as a reference to a word

in the static dictionary given in Appendix A. В цьому випадку,за відстань вважають посилання на слово в статичному словнику .

The copy length for a

static dictionary reference must be between 4 and 24. Довжина копіювання для поислання на статичний словник мусить бути в межах від 4 до 24.

The static

dictionary has three parts:

Статичний словник складається з 3 частин :

\* DICT[0..DICTSIZE], an array of bytes

DICT[0..DICTSIZE], масив байтів

\* DOFFSET[0..24], an array of byte offset values for each length

DOFFSET[0..24], an array of byte offset values for each length

масив значень відстаней для кожної довжини

Alakuijala & Szabadka Expires November 4, 2016 [Page 26]

Internet-Draft Brotli May 2016

\* NDBITS[0..24], an array of bit-depth values for each length

Масив бітових значень для кожної довжини.

The number of static dictionary words for a given length is:

Число слів в статичному словнику для заданої довжини визначають наступним чином

NWORDS[length] = 0 (if length < 4)

NWORDS[length] = (1 << NDBITS[length]) (if length >= 4)

DOFFSET and DICTSIZE are defined by the following recursion:

DOFFSET and DICTSIZE визначають наступним чином :

DOFFSET[0] = 0

DOFFSET[length + 1] = DOFFSET[length] + length \* NWORDS[length]

DICTSIZE = DOFFSET[24] + 24 \* NWORDS[24]

The offset of a word within the DICT array for a given length and

index is:

відстань слова в масиві DIct визначають для заданої довжини та індексу так :

offset(length, index) = DOFFSET[length] + index \* length

Each static dictionary word has 121 different forms, given by

applying a word transformation to a base word in the DICT array.

Кожне слово статичного словника має 121 різну форму.

The

list of word transformations is given in Appendix B. The static

dictionary word for a <length, distance> pair can be reconstructed as

follows:Слово зі статичного словника для пари<довжина,відстань>реконструюють наступним чином :

word\_id = distance - (max allowed distance + 1)

index = word\_id % NWORDS[length]

base\_word = DICT[offset(length, index)..offset(length, index+1)-1]

transform\_id = word\_id >> NDBITS[length]

The string copied to the uncompressed stream is computed by applying

the transformation to the base dictionary word.

Стрічку отриману з нестисненого потоку обчислюють завдяки перетворенню до базового словникового слова.

If transform\_id is

greater than 120, or the length is smaller than 4 or greater than 24,

then the compressed stream should be rejected as invalid.

Якщо transform\_id більше за 120 то потік визначають як некоректний.

Each word transformation has the following form:

Кожне перетворення слова має наступну форму

transform\_i(word) = prefix\_i + T\_i(word) + suffix\_i

where the \_i subscript denotes the transform\_id above.

Де \_і означає trasnsform\_id заначеного вище.

Each T\_i is

one of the following 21 elementary transforms:

Кожен T\_i є одним з 21 елементарних перетворень :

Identity, FermentFirst, FermentAll,

OmitFirst1, ..., OmitFirst9, OmitLast1, ..., OmitLast9

The form of these elementary transforms is as follows:

Identity(word) = word

Alakuijala & Szabadka Expires November 4, 2016 [Page 27]

Internet-Draft Brotli May 2016

FermentFirst(word) = see below

FermentAll(word) = see below

OmitFirstk(word) = the last (length(word) - k) bytes of word, or

empty string if length(word) < k

OmitLastk(word) = the first (length(word) - k) bytes of word, or

empty string if length(word) < k

We define the FermentFirst and FermentAll transforms used in this

specification by the following C language functions:

Ми визначаємо ці два перетворення через спеціалізовані функції написані на мові С:

int Ferment(uint8\_t\* word, int word\_len, int pos) {

if (word[pos] < 192) {

if (word[pos] >= 97 and word[pos] <= 122) {

word[pos] = word[pos] ^ 32;

}

return 1;

} else if (word[pos] < 224) {

if (pos + 1 < word\_len) {

word[pos + 1] = word[pos + 1] ^ 32;

}

return 2;

} else {

if (pos + 2 < word\_len) {

word[pos + 2] = word[pos + 2] ^ 5;

}

return 3;

}

}

void FermentFirst(uint8\_t\* word, int word\_len) {

if (word\_len > 0) {

Ferment(word, word\_len, 0);

}

}

void FermentAll(uint8\_t\* word, int word\_len) {

int i = 0;

while (i < word\_len) {

i += Ferment(word, word\_len, i);

}

}

Appendix B. contains the list of transformations by specifying the

prefix, elementary transform and suffix components of each of them.

Note that the OmitFirst8 elementary transform is not used in the list

Alakuijala & Szabadka Expires November 4, 2016 [Page 28]

Internet-Draft Brotli May 2016

of transformations. The strings in Appendix B. are in C string format

with respect to escape (backslash) characters.

The maximum number of additional bytes that a transform may add to a

base word is 13.

Максимальне число додаткових байтів які можна додати до слова під час перетворення дорівнює 13.

Since the largest base word is 24 bytes long, a

buffer of 38 bytes is sufficient to store any transformed words

(counting a terminating zero byte).Отже ,38 байтів достатньо для зберігання будь-яких перетворених слів.

9. Compressed data format

In this section we describe the format of the compressed data set in

terms of the format of the individual data items described in the

previous sections.

В цьому параграфі ми опишемо формат стиснених даних через окремі елементи даних описаних вище.

9.1. Format of the stream header

Формат заголовку потоку

The stream header has only the following one field:

Заголовк потоку має лише поле

1..7 bits: WBITS, a value in the range 10..24, encoded with

the following variable length code (as it appears in

the compressed data, where the bits are parsed from

right to left):

1…7 бітів:значення в діапазоні 10…24 ,яке закодоване в код змінної довжини

Value Bit Pattern

----- -----------

10 0100001

11 0110001

12 1000001

13 1010001

14 1100001

15 1110001

16 0

17 0000001

18 0011

19 0101

20 0111

21 1001

22 1011

23 1101

24 1111

Note that bit pattern 0010001 is invalid and must not

be used.

The size of the sliding window, which is the maximum value of any

non-dictionary reference backward distance, is given by the following

formula:

Розмір розсувного вікна є максимальним значенням будь-якого посилання назад не на словник,визначають наступним чином :

Alakuijala & Szabadka Expires November 4, 2016 [Page 29]

Internet-Draft Brotli May 2016

window size = (1 << WBITS) - 16

9.2. Format of the meta-block header

Формат заголовку мета-блоку

A compliant compressed data set has at least one meta-block.

Стиснений потік має мати хоча б один мета-блок.

Each

meta-block contains a header with information about the uncompressed

length of the meta-block, and a bit signaling if the meta-block is

the last one. Кожен мета-блок містить заголовку з інформацією стосовно нестисненої довжини мета-блоку ,та сигнальний біт який вказує на кінець мета блоку .

The format of the meta-block header is the following:

Формат заголовку мета-блоку наступний:

1 bit: ISLAST, set to 1 if this is the last meta-block

1 bit: ISLAST, встановлений як 1 якщо це останній блок

1 bit: ISLASTEMPTY, встановлений 1 якщо блок порожній ;

this field is only present if ISLAST bit is set -- if

it is 1, then the meta-block and the brotli stream ends

at that bit, with any remaining bits in the last byte

of the compressed stream filled with zeros (if the

fill bits are not zero, then the stream should be

rejected as invalid)

Це поле існує лише якщо біт ISLAST встановлений як 1.Тоді мета-блок та потік Brotli закінчується на цьому біті,тоді всі решта бітів цього байту мають нульові значення .

2 bits: MNIBBLES, # of nibbles to represent the uncompressed

length, encoded with the following fixed-length code:

MNIBBLES, кількість напівбайт для представлення нестисненої довжини ,закодованої наступним кодом фіксованої довжини :

Value Bit Pattern

----- -----------

0 11

4 00

5 01

6 10

If MNIBBLES is 0, the meta-block is empty, i.e. it does

not generate any uncompressed data. In this case, the

rest of the meta-block has the following format:

Якщо MNIBBLES встановлений як 0 ,то мета-блок порожній.В цьому випадку решта мета-блоку має наступний формат:

1 біт: зарезервований під 0.

1 bit: reserved, must be zero

2 bits: MSKIPBYTES, # of bytes to represent metadata

Length

2 біти: : MSKIPBYTES-кількість байт для представлення метаданих.

Довжина

MSKIPBYTES x 8 bits: MSKIPLEN - 1, where MSKIPLEN is

the number of metadata bytes; this field is

only present if MSKIPBYTES is positive,

otherwise MSKIPLEN is 0 (if MSKIPBYTES is

greater than 1, and the last byte is all

zeros, then the stream should be rejected

as invalid)

MSKIPBYTES x8 біт:MSKIPLEN -1,ду MSKIPLEN –кількість байт метаданих.Це пиоле присутнє лише коли значення MSKIPBYTES більше від нуля .В іншому випдку MSKIPLEN дорівнює.

0…7 бітів:Заповнення байту до його кінця.

0..7 bits: fill bits until the next byte boundary,

must be all zeros

MSKIPLEN bytes of metadata, not part of the

Alakuijala & Szabadka Expires November 4, 2016 [Page 30] uncompressed data or the sliding window

MNIBBLES x 4 bits: MLEN - 1, where MLEN is the length

of the meta-block uncompressed data in bytes (if

MNIBBLES is greater than 4, and the last

nibble is all zeros, then the stream should be

rejected as invalid)

MNIBBLES x4 біти : MLEN -1 ,де MLEN –довжина мате-блоку нестиснених даних.Якщо MNIBBLES,більше ща 4 і останній напівбайт нульовий ,тоді потік відкидають як некоректний.

1 bit: ISUNCOMPRESSED, if set to 1, any bits of compressed

data up to the next byte boundary are ignored, and

the rest of the meta-block contains MLEN bytes of

literal data; this field is only present if the

ISLAST bit is not set (if the ignored bits are not

all zeros, the stream should be rejected as invalid)

1 біт:ISUNCOMPRESSED .Якщо значення 1 ,тоді всі біти до наступного байту відкидають.Це поле присутнє лише коли біт ISLAST не виставлений .

1..11 bits: NBLTYPESL, # of literal block types, encoded with

the following variable length code (as it appears in

the compressed data, where the bits are parsed from

right to left, so 0110111 has the value 12):

1…11 біт : NBLTYPESL-кількість типів блоків ,закодованих наступним кодом змінної довжини.

Value Bit Pattern

----- -----------

1 0

2 0001

3..4 x0011

5..8 xx0101

9..16 xxx0111

17..32 xxxx1001

33..64 xxxxx1011

65..128 xxxxxx1101

129..256 xxxxxxx1111

Prefix code over the block type code alphabet for

literal block types, appears only if NBLTYPESL >= 2

Префіксний код для типів блоку для літералів присутній якщо NBLTYPESL>=2/

Prefix code over the block count code alphabet for

literal block counts, appears only if NBLTYPESL >= 2

Block count code + extra bits for first literal

block count, appears only if NBLTYPESL >= 2

1..11 bits: NBLTYPESI, # of insert-and-copy block types, encoded

with the same variable length code as above

1..11 біт:NBLTYPESL – кілкість типів блоку для вставкти та копіювання ,для використано той же код змінної довжини.

Prefix code over the block type code alphabet for

insert-and-copy block types, appears only if NBLTYPESI >= 2

Prefix code over the block count code alphabet for

Alakuijala & Szabadka Expires November 4, 2016 [Page 31]

Internet-Draft Brotli May 2016

insert-and-copy block counts, appears only if NBLTYPESI >= 2

Block count code + extra bits for first insert-and-copy

block count, appears only if NBLTYPESI >= 2

1..11 bits: NBLTYPESD, # of distance block types, encoded

with the same variable length code as above

1..11 біт:NBLTYPESD – кількість блоків для відстаней ,для ких використано наведений вище код змінної довжини.

Prefix code over the block type code alphabet for

distance block types, appears only if NBLTYPESD >= 2

Префіксний код для типів блоку для відстаней існує лише коли NBLTYPESD >=2.

Prefix code over the block count code alphabet for

distance block counts, appears only if NBLTYPESD >= 2

Block count code + extra bits for first distance

block count, appears only if NBLTYPESD >= 2

2 bits: NPOSTFIX, parameter used in the distance coding

2 біти :NPOSTFIX,параметер який використовуэться в кодуванні відстаней.

4 bits: four most-significant bits of NDIRECT, to get the

actual value of the parameter NDIRECT, left-shift

this four-bit number by NPOSTFIX bits

4 біти :4 найбільш важливих біти NDIRECT lвикористовуються для отримання значення NDIRECT перед тим зсунувши на NPOSTFIX бітів.

NBLTYPESL x 2 bits: context mode for each literal block type

2 біти :NBLTYPESL-вид контексту для типу блоку для літералів.

1..11 bits: NTREESL, # of literal prefix trees, encoded

with the same variable length code as NBLTYPESL

NTREEESL(1-11біт)- кількість префікс них дерев для літералів ,закодованих тим же кодом змінної довжини як і NBLTYPESL.

Literal context map, encoded as described in Section 7.3.,

appears only if NTREESL >= 2, otherwise the context map

has only zero values

1..11 bits: NTREESD, # of distance prefix trees, encoded

with the same variable length code as NBLTYPESD

NTREESD(1-11 біт)-кількість префікс них дерев для відстаней ,закодованих тим же кодом змінної довжини що й NBLTYPESD.

Distance context map, encoded as described in Section 7.3.,

appears only if NTREESD >= 2, otherwise the context map

has only zero values

NTREESL prefix codes for literals

NBLTYPESI prefix codes for insert-and-copy lengths

NTREESD prefix codes for distances

9.3. Format of the meta-block data

Формат даних мета-блоку

The compressed data part of a meta-block consists of a series of

commands.

Стиснена частина даних мета-блоку складається з послідовності команд.

Each command has the following format:

Кожна з команд має наступний формат.

Block type code for next insert-and-copy block type, appears

only if NBLTYPESI >= 2 and the previous insert-and-copy

block count is zero

Код типу блоку для натупного типу блоку для вставки та копіювання з’явлється лише коли NBLTYPESL>=2 та коли попередій лічильни блоку встановлений як 0.

Block count code + extra bits for next insert-and-copy

block count, appears only if NBLTYPESI >= 2 and the

previous insert-and-copy block count is zero

Код лічильника блоку з додатковими бітами для наступного лічильника блоку для вставки та копіювання .Від пристуній лише коли NBLTYPESL>=2 та коли попередій лічильник блоку встановлений як 0.

Insert-and-copy length, encoded as in Section 5., using the

insert-and-copy length prefix code with the current

insert-and-copy block type index

Довжина вставки та копіювання ,як зазначено вище,закодована за використання префікс них кодів з індексом типу блоку для копіювання та вставки.

Insert length number of literals, with the following format:

Число довжини для вставки літералів представлено аналогічно до попереднього прикладу.

Block type code for next literal block type, appears

only if NBLTYPESL >= 2 and the previous literal

block count is zero

Block count code + extra bits for next literal

block count, appears only if NBLTYPESL >= 2 and the

previous literal block count is zero

Next byte of the uncompressed data, encoded with the

literal prefix code with the index determined by the

previous two bytes of the uncompressed data, the

current literal block type, and the context map, as

described in Section 7.3.

Наступний байт нестиснених даних кодують з літеральним префікс ним кодом де індекс визначений завдяки попереднім двом байтам,поточному типу літерального блоку та контекстного відображення .

Block type code for next distance block type, appears

only if NBLTYPESD >= 2 and the previous distance

block count is zero

Block count code + extra bits for next distance

block count, appears only if NBLTYPESD >= 2 and the

previous distance block count is zero

Distance code, encoded as in Section 4., using the distance

prefix code with the current distance block type index,

appears only if the distance code is not an implicit 0,

as indicated by the insert-and-copy length code

Коди відстаней закодовані завдяки префіксному коду з поточним індексом типу блоку для відстаней .

The number of commands in the meta-block is such that the sum of the

uncompressed bytes produced (i.e. the number of literals inserted

plus the number of bytes copied from past data or generated from the

static dictionary) over all the commands gives the uncompressed

length, MLEN encoded in the meta-block header.

Alakuijala & Szabadka Expires November 4, 2016 [Page 33]

Internet-Draft Brotli May 2016

If the total number of uncompressed bytes produced after the insert

part of the last command equals MLEN, then the copy length of the

last command is ignored and will not produce any uncompressed output.

In this case the copy length of the last command can have any value.

In any other case, if the number of literals to insert, the copy

length, or the resulting dictionary word length would cause MLEN to

be exceeded, then the stream should be rejected as invalid.

Якщо загальна кількість нестиснених даних після частини вставки останньої команди дорівнюєMLEN ?отже довжина копіювання останньої команди не беруть до уваги.В цьому випадку довжина копіювання останньої може мати будь-яке значення .В іншому випадку,потік відкидають.

If the last command of the last non-empty meta-block does not end on

a byte boundary, the unused bits in the last byte must be zeros.

10. Decoding algorithm

Алгоритм розтискання

The decoding algorithm that produces the uncompressed data is as

follows:

read window size

do

read ISLAST bit

if ISLAST

read ISLASTEMPTY bit

if ISLASTEMPTY

break from loop

read MNIBBLES

if MNIBBLES is zero

verify reserved bit is zero

read MSKIPLEN

skip any bits up to the next byte boundary

skip MSKIPLEN bytes

continue to the next meta-block

else

read MLEN

if not ISLAST

read ISUNCOMPRESSED bit

if ISUNCOMPRESSED

skip any bits up to the next byte boundary

copy MLEN bytes of compressed data as literals

continue to the next meta-block

loop for each three block categories (i = L, I, D)

read NBLTYPESi

if NBLTYPESi >= 2

read prefix code for block types, HTREE\_BTYPE\_i

read prefix code for block counts, HTREE\_BLEN\_i

read block count, BLEN\_i

set block type, BTYPE\_i to 0

initialize second-to-last and last block types to 0 and 1

else

set block type, BTYPE\_i to 0

Alakuijala & Szabadka Expires November 4, 2016 [Page 34]

Internet-Draft Brotli May 2016

set block count, BLEN\_i to 16777216

read NPOSTFIX and NDIRECT

read array of literal context modes, CMODE[]

read NTREESL

if NTREESL >= 2

read literal context map, CMAPL[]

else

fill CMAPL[] with zeros

read NTREESD

if NTREESD >= 2

read distance context map, CMAPD[]

else

fill CMAPD[] with zeros

read array of literal prefix codes, HTREEL[]

read array of insert-and-copy length prefix codes, HTREEI[]

read array of distance prefix codes, HTREED[]

do

if BLEN\_I is zero

read block type using HTREE\_BTYPE\_I and set BTYPE\_I

save previous block type

read block count using HTREE\_BLEN\_I and set BLEN\_I

decrement BLEN\_I

read insert-and-copy length symbol using HTREEI[BTYPE\_I]

compute insert length, ILEN, and copy length, CLEN

loop for ILEN

if BLEN\_L is zero

read block type using HTREE\_BTYPE\_L and set BTYPE\_L

save previous block type

read block count using HTREE\_BLEN\_L and set BLEN\_L

decrement BLEN\_L

look up context mode CMODE[BTYPE\_L]

compute context ID, CIDL from last two uncompressed bytes

read literal using HTREEL[CMAPL[64\*BTYPE\_L + CIDL]]

write literal to uncompressed stream

if number of uncompressed bytes produced in the loop for

this meta-block is MLEN, then break from loop (in this

case the copy length is ignored and can have any value)

if distance code is implicit zero from insert-and-copy code

set backward distance to the last distance

else

if BLEN\_D is zero

read block type using HTREE\_BTYPE\_D and set BTYPE\_D

save previous block type

read block count using HTREE\_BLEN\_D and set BLEN\_D

decrement BLEN\_D

compute context ID, CIDD from CLEN

read distance code using HTREED[CMAPD[4\*BTYPE\_D + CIDD]]

compute distance by distance short code substitution

Alakuijala & Szabadka Expires November 4, 2016 [Page 35]

Internet-Draft Brotli May 2016

if distance code is not zero,

and distance is not a static dictionary reference,

push distance to the ring buffer of last distances

if distance is less than the max allowed distance plus one

move backwards distance bytes in the uncompressed data,

and copy CLEN bytes from this position to

the uncompressed stream

else

look up the static dictionary word, transform the word as

directed, and copy the result to the uncompressed stream

while number of uncompressed bytes for this meta-block < MLEN

while not ISLAST

If the stream ends before the completion of the last meta-block, then

the stream should be rejected as invalid.

Якщо потік закінчується перед завершенням останнього мета-блоку ,толі потік слід відкинути.

Note that a duplicated string reference may refer to a string in a

previous meta-block, i.e. the backward distance may cross one or more

meta-block boundaries. Посилання на повторювану стрічку може посилатися

на стрічку в попередньому мета-блоці .

However a backward copy distance will not

refer past the beginning of the uncompressed stream or the window

size; any such distance is interpreted as a reference to a static

dictionary word.

Проте відстань копіювання не може посилатися до місця яке визначене перед початком нестисненого потоку чи розсувного вікна .В цьому випадку ,відбувається звертання до статичного словника .

Also note that the referenced string may overlap the

current position, for example, if the last 2 bytes decoded have

values X and Y, a string reference with <length = 5, distance = 2>

adds X,Y,X,Y,X to the uncompressed stream.

11. Considerations for compressor implementations

Особливості реалізації компресора .

Since the intent of this document is to define the brotli compressed

data format without reference to any particular compression

algorithm, the material in this section is not part of the definition

of the format, and a compressor need not follow it in order to be

compliant.

11.1. Trivial compressor

Тривіальна реалізація

In this section we present a very simple algorithm that produces a

valid brotli stream representing an arbitrary sequence of

uncompressed bytes in the form of the following C++ language

function.Нижче подано код простої реалізації алгоритму який породжує дійсний потік для алгоритму Brotli на мові С++.

string BrotliCompressTrivial(const string& u) {

if (u.empty()) {

return string(1, 6);

}

int i;

string c;

c.append(1, 12);

Alakuijala & Szabadka Expires November 4, 2016 [Page 36]

Internet-Draft Brotli May 2016

for (i = 0; i + 65535 < u.size(); i += 65536) {

c.append(1, 248);

c.append(1, 255);

c.append(1, 15);

c.append(&u[i], 65536);

}

if (i < u.size()) {

int r = u.size() - i - 1;

c.append(1, (r & 31) << 3);

c.append(1, r >> 5);

c.append(1, 8 + (r >> 13));

c.append(&u[i], r + 1);

}

c.append(1, 3);

return c;

}

Зауважимо що вищенаведений алгоритм не стискує дані.В даній примітивній версії потікBrotli буде більшим за оригінал .Проте,завдяки даному алгоритмові може з’ясувати що кожна послідовність з N нестиснених даних може бути представлена в потоцы Brotli який не більше ніж

N + (3 \* (N >> 16) + 5) байт.

Note that this simple algorithm does not actually compress data, that

is, the brotli representation will always be bigger than the

original, but it shows that every sequence of N uncompressed bytes

can be represented with a valid brotli stream that is not longer than

N + (3 \* (N >> 16) + 5) bytes.

11.2. Aligning compressed meta-blocks to byte boundaries

As described in Section 9., only those meta-blocks that immediately

follow an uncompressed meta-block or a metadata meta-block are

guaranteed to start on a byte boundary. Як показано вище,лише мета-блоки які йдуть одразу після мета-блоку метаданих чи порожні мета-блоки гаранотовано знаходяться на початку байта.

In some applications, it

might be required that every non-metadata meta-block starts on a byte

boundary.В декотрих застосуваннях може бути необхідно щоб всі мета-блоки знаходилися на початках байтів.

This can be achieved by appending an empty metadata meta-

block after every non-metadata meta-block that does not end on a byte

boundary.

Цього можна досягнути додаючи нульові ,а точніше,порожні мета-блоки метаданих після кожного мета-блоку зі стисненим даним .

11.3. Creating self-contained parts within the compressed data

Створення автономних частин в стиснених даних

In some encoder implementations it might be required to make a

sequence of bytes within a brotli stream self-contained, that is,

such that they can be decompressed independently from previous parts

of the compressed data. В декотрих випадках може бути необхідно створити послідовність байт з потоком Brotli який є автономним .Тобто такий,який може бути відновленим незалежно від попередніх частин відновлених даних.

This is a useful feature for three reasons.

Це дуже корисно з наступних причин.

First, if a large compressed file is damaged, it is possible to

recover some of the file after the damage.

По-перше ,якщо велики файл стиснених даних пошкоджено ,то можна відновити хоча б частину інформації.

Second, it is useful when

doing differential transfer of compressed data.

По-друге,можна ділити дані і передавати їх по частинам.If a sequence of

uncompressed bytes is unchanged and compressed independently from

previous data, then the compressed representation may also be

unchanged and can therefore be transferred very cheaply. Якщо послідовність відновлених даних не була змінена та стиснена незалежно від попередніх даних то стиснене представлення також не буде зміненим ,а отже,передача даних буде дешевшою.

Third, if

sequences of uncompressed bytes are compressed independently, it

allows for parallel compression of these byte sequences within the

same file, in addition to parallel compression of multiple files.

По-третє,якщо послідовності нестиснених байтів стиснуті незалежно ,то можна паралельно стискати ці послідовності в одному файлі.

Alakuijala & Szabadka Expires November 4, 2016 [Page 37]

Internet-Draft Brotli May 2016

Given two sequences of uncompressed bytes, U0 and U1, we will now

describe how to create two sequences of compressed bytes, C0 and C1,

such that the concatenation of C0 and C1 is a valid brotli stream,

and that C0 and C1 (together with the first byte of C0 that contains

the window size) can be decompressed independently from each other to

U0 and U1.

За даним послідовностями U0та U1 ми можемо створити дві послідовності С0та С1 ,так що конкатенація С0 та С1 буде коректним потоком Brotli .До того ж,С0 та С1 можуть буди відновлені незалежно один від одного.

When compressing the byte sequence U0 to produce C0, we can use any

compressor that works on the complete set of uncompressed bytes U0,

with the following two changes. First, the ISLAST bit of the last

meta-block of C0 must not be set. Second, C0 must end at a byte-

boundary, which can be ensured by appending an empty metadata meta-

block to it, as in Section 11.2.Для цього слід внести зміни до останнього мета-блоку С0 ,так що ISLAST останнього блоку не маэ значення .Тоді слід забезпечити щоб останній блок С0 точно закінчувався на границі байта..

When compressing the byte sequence U1 to produce C1, we can use any

compressor that starts a new meta-block at the beginning of U1 within

the U0+U1 input stream, with the following two changes. Для створення С1 слід внести 2 зміни

First,

backward distances in C1 must not refer to static dictionary words or

uncompressed bytes in U0.

По-перше,відстані до попередніх копій в С1 не мають посилатися на слова зі статичного словника чи нетиснутих байтів U0.

Even if a sequence of bytes in U1 would

match a static dictionary word, or a sequence of bytes that overlaps

U0, the compressor must represent this sequence of bytes with a

combination of literal insertions and backward references to bytes in

U1 instead.Навіть якщо послідовність байт в U1 співпадає зі словом в статичному словнику ,чи з послідовністю байт в в U0,то компресор мусить представити послідовність байт як комбінацію з вставок літералів та посилань на копії в байтах U1.

Second, the ring buffer of last four distances must be

replenished first with distances in C1 before using it to encode

other distances in C1.

Note that both compressors producing C0 and C1

have to use the same window size, but the stream header is emitted

only by the compressor that produces C0.

Обидва компресори для С0 та С1 мають мати однакове розсувне вікно ,але заголовок потоку створений лише першим компрессором який створює С0.

Note that this method can be easily generalized to more than two

sequences of uncompressed bytes.

12. Security Considerations

As with any compressed file formats, decompressor implementations

should handle all compressed data byte sequences, not only those that

conform to this specification, where non-conformant compressed data

sequences should be rejected as invalid.

A possible attack against a system containing a decompressor

implementation (e.g. a web browser) is to exploit a buffer overflow

triggered by invalid compressed data.

Завдяки цьому алгоритму можна проводити атаки на веб-браузери користувача через експлуатацію переповненн буфера при відновленні (buffer overflow) якщо ввести некоректні стиснуті дані.

Therefore decompressor

implementations should perform bounds-checking for each memory access

that result from values decoded from the compressed stream and

derivatives therof.

Відповідно реалізації декомпресора мають виконувати перевірку використання пам’яті яка займають дані відновлені зі стиснутого потоку .

Another possible attack against a system containing a decompressor

implementation is to provide it (either valid or invalid) compressed

data that can make the decompressor system's resource consumptioт

(cpu, memory, or storage) to be disproportionately large compared to

the size of the compressed data.

Іншим можливим способом атаки на систему через алгоритм розтискання є подання стиснутих даних,при обробці який декомпресор використає необґрунтовано багато ресурсів.

In addition to the size of the

compressed data, the amount of cpu, memory and storage required to

decompress a single compressed meta-block within a brotli stream is

controlled by the following two paramters: the size of the

uncompressed meta-block, which is encoded at the start of the

compressed meta-block, and the size of the sliding window, which is

encoded at the start of the brotli stream.Кількість ресурсів яку використовує декомпресор для обробки одного мета-блоку на пряму залежить від розміру нестиненого мета-блоку ,який закодовано на початку стиснутого мета-блоку ,та розміру розсувного вікна ,який закодований на початку потоку brotli.

Decompressor

implementations in systems where memory or storage is constrained

should perform a sanity-check on these two parameters.Отже,реалізації декомпресора в системах де критичним є використання даних ресурсів пам’яті мають проводи постійну перевірку даних параметрів.

The

uncompressed meta-block size that was decoded from the compressed

stream should be compared against either a hard limit, given by the

system's constraints or some expectation about the uncompressed data,

or against a certain multiple of the size of the compressed data.

Необхідно встановити ліміт на використання пам’яті одним відновленим мета-блоком з потоку.

If

the uncompressed meta-block size is determined to be too high, the

compressed data should be rejected. Likewise, when the complete

uncompressed stream is kept in the system containing the decompressor

implementation, the total uncompressed size of the stream should be

checked before decompressing each additional meta-block. У випадку коли весь потік (відновлений)тримають в системі,слід перевіряти загальний розмір цього потоку перед прочитанням кожного наступного стиснутого мета-блоку.

If the size

of the sliding window that was decoded from the start of the

compressed stream is greater than a certain soft limit, then the

decompressor implementation should, at first, allocate a smaller

sliding window that fits the first uncompressed meta-block, and

afterwards, before decompressing each additional meta-block, it

should increase the size of the sliding window until the sliding

window size specified in the compressed data is reached.

Correspondingly, possible attacks against a system containing a

compressor implementation (e.g. a web server) are to exploit a buffer

overflow or cause disproportionately large resource consumption by

providing e.g. uncompressible data

Відповідно можливі атаки на програми з використанням алгоритму компресор(наприклад веб-сервер) використовують можливість перезаповення буфера чи спричиняють надмірне використання ресурсів .

As described in Section 11.1.,

an output buffer of

S(N) = N + (3 \* (N >> 16) + 5)

bytes is sufficient to hold a valid compressed brotli stream

representing an arbitrary sequence of N uncompressed bytes.

Як зазначено вище,для зберігання потоку слід мати

S(N) = N + (3 \* (N >> 16) + 5)

S(n) байт .Тобто,компресор має виділяти S(n) байт у вихідному буфері для розміщення даних .Якщо під час виконання алгоритму це кількість перевищено,алгоритм має повернутися до тривіального алгоритму описаного вище або ж відкинути потік.

Therefore compressor implementations should allocate at least S(N)

bytes of output buffer before compressing N bytes of data with

unknown compressibility and should perform bounds-checking for each

write into this output buffer. If their output buffer is full,

compresor implementations should revert to the trivial compression

algorithm described in Section 11.1. The resourse consumption of a

compressor implementation for a particular input data depends mostly

on the algorithm used to find backward matches and on the algorithm

used to construct context maps and prefix codes and only to a lesser

extent on the input data itself.Використання ресурсів компрессором для певного конкретного типу даних залежить від алгоритму пошуку співпадінь та алгоритму для контсурювання контекстних відображень та префікнсих кодів і в набагато меншій мірі від самих даних.

If the system containing a

compressor implementation is overloaded, a possible way to reduce

resource usage is to switch to more simple algorithms for backward

reference search and prefix code construction, or to fall back to the

trivial compression algorithm described in Section 11.1.

A possible attack against a system that sends compressed data over an

encrypted channel is the following. An attacker who can repeatedly

mix arbitrary (attacker-supplied) data with secret data (passwords,

cookies) and observe the length of the ciphertext can potentially

reconstruct the secret data. To protect against this kind of attack,

applications should not mix sensitive data with non-sensitive,

potentially attacker-supplied data in the same compressed stream.

13. IANA Considerations

The "HTTP Content Coding Registry" has been updated with the

registration below:

+-------+-------------------------------------+------------+

| Name | Description | Reference |

+-------+-------------------------------------+------------+

| br | Brotli Compressed Data Format | RFCXXXX |

+-------+-------------------------------------+------------+

14. Informative References

[HUFFMAN] Huffman, D. A., "A Method for the Construction of Minimum

Redundancy Codes", Proceedings of the Institute of Radio

Engineers, September 1952, Volume 40, Number 9, pp.

1098-1101.

[LZ77] Ziv J., Lempel A., "A Universal Algorithm for Sequential

Data Compression", IEEE Transactions on Information

Theory, Vol. 23, No. 3, pp. 337-343.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification

version 1.3", RFC 1951, Aladdin Enterprises, May 1996.

http://www.ietf.org/rfc/rfc1951.txt

[WOFF2] Levantovsky, V. (ed.), Levien, R. (ed.), "WOFF File Format

2.0", W3C WebFonts Working Group,

http://www.w3.org/TR/WOFF2/

15. Код застосування

Відкритий код застосування для реалізації на мові С декомпресора та компресора на мові C++ подаються як проект з відкритим програмни кодом .

Appendix B. List of word transformations

The string literals are in C format, with respect to the use of

backslash escape characters.

In order to generate a length and check value, the transforms can be

converted to a series of bytes, where each transform is the prefix

sequence of bytes plus a terminating zero byte, a single byte value

identifying the transform, and the suffix sequence of bytes plus a

terminating zero. The value for the transforms are 0 for Identity, 1 for

FermentFirst, 2 for FermentAll, 3 to 11 for OmitFirst1 to OmitFirst9,

and 12 to 20 for OmitLast1 to OmitLast9. The byte sequences that

represent the 121 transforms are then concatenated to a single sequence

of bytes. The length of that sequence is 648 bytes, and the CRC-32 is

0x3d965f81.

ID Prefix Transform Suffix

-- ------ --------- ------

0 "" Identity ""

1 "" Identity " "

2 " " Identity " "

3 "" OmitFirst1 ""

4 "" FermentFirst " "

5 "" Identity " the "

6 " " Identity ""

7 "s " Identity " "

8 "" Identity " of "

9 "" FermentFirst ""

10 "" Identity " and "

11 "" OmitFirst2 ""

12 "" OmitLast1 ""

13 ", " Identity " "

14 "" Identity ", "

15 " " FermentFirst " "

16 "" Identity " in "

17 "" Identity " to "

18 "e " Identity " "

19 "" Identity "\""

20 "" Identity "."

21 "" Identity "\">"

22 "" Identity "\n"

23 "" OmitLast3 ""

24 "" Identity "]"

25 "" Identity " for "

26 "" OmitFirst3 ""

27 "" OmitLast2 ""

28 "" Identity " a "

29 "" Identity " that "

30 " " FermentFirst ""

31 "" Identity ". "

32 "." Identity ""

33 " " Identity ", "

34 "" OmitFirst4 ""

35 "" Identity " with "

36 "" Identity "'"

37 "" Identity " from "

38 "" Identity " by "

39 "" OmitFirst5 ""

40 "" OmitFirst6 ""

41 " the " Identity ""

42 "" OmitLast4 ""

43 "" Identity ". The "

44 "" FermentAll ""

45 "" Identity " on "

46 "" Identity " as "

47 "" Identity " is "

48 "" OmitLast7 ""

49 "" OmitLast1 "ing "

50 "" Identity "\n\t"

51 "" Identity ":"

52 " " Identity ". "

53 "" Identity "ed "

54 "" OmitFirst9 ""

55 "" OmitFirst7 ""

56 "" OmitLast6 ""

57 "" Identity "("

58 "" FermentFirst ", "

59 "" OmitLast8 ""

60 "" Identity " at "

61 "" Identity "ly "

62 " the " Identity " of "

63 "" OmitLast5 ""

64 "" OmitLast9 ""

65 " " FermentFirst ", "

66 "" FermentFirst "\""

67 "." Identity "("

68 "" FermentAll " "

69 "" FermentFirst "\">"

70 "" Identity "=\""

71 " " Identity "."

72 ".com/" Identity ""

73 " the " Identity " of the "

74 "" FermentFirst "'"

75 "" Identity ". This "

76 "" Identity ","

77 "." Identity " "

78 "" FermentFirst "("

79 "" FermentFirst "."

80 "" Identity " not "

81 " " Identity "=\""

82 "" Identity "er "

83 " " FermentAll " "

84 "" Identity "al "

85 " " FermentAll ""

86 "" Identity "='"

87 "" FermentAll "\""

88 "" FermentFirst ". "

89 " " Identity "("

90 "" Identity "ful "

91 " " FermentFirst ". "

92 "" Identity "ive "

93 "" Identity "less "

94 "" FermentAll "'"

95 "" Identity "est "

96 " " FermentFirst "."

97 "" FermentAll "\">"

98 " " Identity "='"

99 "" FermentFirst ","

100 "" Identity "ize "

101 "" FermentAll "."

102 "\xc2\xa0" Identity ""

103 " " Identity ","

104 "" FermentFirst "=\""

105 "" FermentAll "=\""

106 "" Identity "ous "

107 "" FermentAll ", "

108 "" FermentFirst "='"

109 " " FermentFirst ","

110 " " FermentAll "=\""

111 " " FermentAll ", "

112 "" FermentAll ","

113 "" FermentAll "("

114 "" FermentAll ". "

115 " " FermentAll "."

116 "" FermentAll "='"

117 " " FermentAll ". "

118 " " FermentFirst "=\""

119 " " FermentAll "='"

120 " " FermentFirst "='"