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 .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

1. Introduction

1.1. Purpose

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

Було незалежним від виду процесору ,операційної системи,файлової системи,А також типу кодування символів,і відповідно його можна було б застосовувати для обміну даними.

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

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

Процесс відновлення даних швидший за сучасну імплементацію LZMA алгоритму..

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.В специфікації зазначено метод для представлення послідовності байтів як послідовностей бітів ,та метод для пакування послідовностей бітів в байти назад.Далі під байтом буде визначено послідовність з 8 бітів збережених як одиниці інформації.Під стрічкою мається на увазі будь-яка послідовність байтів довільної довжини.У визначенні байту послуговуватимемося позначенням спочатку найбільш важливого біту зліва.Оскільки число може бути більшим за 1 байт ,будемо розміщувати 1 байт числа зліва(найменш важливий байт),а решту – справа.

Отже,остаточна форма стиснених даних матиме наступні властивості:

Елементи даних спаковані в байти в порядку зростання бітового числа в байті,починаючи з найменш важливого біту .Префіксні коди розташовані починаючи з найбільш важливого бітуВідповідно 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

2. Compressed representation overview

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.

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.

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.

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.

Кожна з команд стиснених даних представлення у вигляді 3 категорій префікс них кодів :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)Другий тільки для літералів.3)Третій тільки для відстанейПрефіксні коди для кожного блоку знаходяться перед стисненим даним в заголовку мате-блоку.Приклад команди як послідовності префікс них кодів

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.

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.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.Частиною цього контексту є 3 поточні типи блоків,по-одному для кожної категорії.Для кожної категорії існує лічильник ,стосовно того скільки елементів цієї категорії слід декодувати в поточному типі блоку.Як тільки лічильник був розширений,новий тип блоку та новий лічильник дістають з потоку перед наступним елементом категорії ,який буде використовувати новий тип блоку.

Тип вставки та копіювання блоку визначає який префікс ний код використати для наступного елементу цього ж типу.Для літералів та відстаней,відповідні типи блоків використовують в комбінації з іншою контекстною інформацією для наступного елементу.Розглянемо наступний приклад

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

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 .ми розділяємо кожну категорію на свою власну послідовність аби показати приклад типів блоків .Кожна група з квадратним дужками використовує один тип блоку.

[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

. Наступні блоки в кожній категоріх мають мати різні типи блоків,але їх можна потім використовувати заново в мета-блоці.Типи блоків нумеровані від 0 до максимального числа блоків (255) і перший блок кожнох категорії має тип 0.Блочна структура мета-блоку представленя послідовністю команд для кожної категоріх,де команда має вигляд<тип блоку,лічильник блоку>.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.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

Тут команда xBlockSwitch(t, n) перемикає до типу блоку т для n елементів. Зазначимо що в цьому прикладі DBlockSwitch(1, 3) одразу спереду наступної необхідної відстані D1. Коли елемент категорії потрібен,і лічильник категорії дійшов ддо нуля,тоді зчитують новий тип блоку та лічильний з потоку прямо перед читанням наступного елементу.Команди блоку для перших блоків кожної категоріх не є частиною мета-блоку стиснених даних.Натомість перший тип блоку визначений як 0 ,та перший лічильник для кожної кате-горії закодований в заголовку мета-блоку.Всі префіксні коди для типів блоку та лічильників визначені в заголовку мет-блоку.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.

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

У випадку літералів ,контекст це два попередні байти нестиснених даних.У випадку відстаней ,контекст це довжина копіювання з цієї ж команди.Для довжина вставки та копіювання не використовують контексту та префікс ний код залежить лише від типу блоку .У випадку літералів та відстаней контекст має відображення на упорядковану множину від 0 до 63 для літералів та від 0 до 3 для відстаней.

Матриця префікс них кодів

.Індекси для кожного типу блоку та контекстний ID утворюють разом контекстне відображення яке записане в заголовок мета-блоку.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.Аналогічно префікс ний код для відновлення D0 залежить на типові блоку (0),та контекстний ID визначений завдяки довжини копіювання з IaC0.Префіксний код для відновлення IaC3 залежить лише від типу блоку (1).

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.

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

3. Compressed representation of prefix codes

Стиснене представлення префікс них кодів

3.1. Introduction to prefix coding

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

Приклад

/\ Символ Код

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

/ \ A 00

/\ B B 1

0 1 C 011

/ \ D 010

A /\

0 1

/ \

D C

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

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.

3.2. Use of prefix coding in the brotli format

Використання префіксного кодування в алгоритміПрефіксні коді які використовують для кожного алфавіту в алгоритмі є канонічними префікс ними кодами які мають два додаткові правилаВсі коди певної бітової довжини мають лексикографічні послідовні значення ,в тому ж порядку що й символи які вони представляють.Коротші коди лексикографічно знаходяться раніше за довші коди .

Прикладна основі алфавіту ABC

Символ Код

------ ----

A 10

B 0

C 110

D 111

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

В нашому прикладі код повністю визначений завдяки послідовності бітової довжини (2,2,3,3).Наступний алгоритм записує префіксні коди у вигляді цілих чисел ,за умови що вони будуть прочитані від найбільш до найменш важливого біту .

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

2)Знайти числове значення найменшого коду для кожної довжини.Див.Додаток А

3)Присвоїти числове значення для всіх кодів ,використовуючи послідовні значення для всіх кодів однаковох довжини з базовими значеннями визначеним на кроці 2.Див.Додаток А

3.3. Alphabet sizes

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.

3.4. Simple prefix codes

Перші два біти стисненого представлення кожного префіксного коду визначають чи це простий чи складний префікс ний код.У випадку 1 - це простий префікс ний код.Простий префікс ний код може мати лише не більше чотирьох символів з ненульовою довжиною коду.В загальному ,формат простого префіксного коду

2 біти:значення в 1 означає що це простий код

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

1 біт :наявний лише коли NSYM = 4

Кількість бітів для позначення алфавіту –найменша кілкість бітів яка може представляти символи в алфавіті.Префіксні коди однакової бітової довжини мають бути записані в тому ж порядку що й їхні символи в відсортованому за зростанням порядку.Ненульові довжин кодів символів можуть бути реконструйовані наступним чином

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

Якщо NSYM = 2,всі символи мають довжину 1.

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

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

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:Для кодів довжини 0..15

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

Наступні 2 біти визначають довжину повторення

(0 = 3, ... , 3 = 6)Якщо це перший код довжини ,або всі попредні мають нульову довжину ,код довжини 8 повторюють від 3 до 6 разів.

Повтрюваний код 16 змінює лічильник повторень наступним чином

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

(3..6 on the next 2 bits)

17:Повторити довжину коду 0 від 3 до 10 разів

(3 bits of length)Код 17 змінює лічильник повторень наступним чином

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

(3..10 on the next 3 bits)

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 ..

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.

Складний префіксние код мусить мати щонайменше дві ненульові довжин коду.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

Тепер можна визначити формат складного префікс них кодів наступним чином.

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).

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.

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.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 .

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.

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.

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.

Одиничний символ одразу буде повернутий коли цей код буде розшифрований.

Приклад ,у якому весь код представлений символами довжин 8.

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

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

Попередню частоту взяту як 8 зчитують до того як буде прочитана довжина коду. Будь-який перший символ (0 або 17) має бути пропущений,так як остання закодована довжина символу мусить бути між 1 до 16.Сума довжин по всім ненульовим довжинам коду в алфавіті ,разом з закодованим кодом повторення 16 ,мають бути рівні 32768.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

Кодування відстаней

Як зазначено в параграфі 2 ,одним з компонентів стисненого мат-блоку є пслоідовність відстаней до прочитаних раніше ідентичних послідовностей.В цьому параграфі ми ознайомимемося з деталями процесу стиснення відстаней.Кожна відстань в стиснених даних мета-блоку представлена у вигляді пари <код відстані,додаткові біти>.

Коди відстані та додаткові біти закодовані разом.Код відстані кодують використовуючи той самий префікс ний код по всьому алфавіту відстаней .Додаткові біти кодують використовуючи цілочисленні значення фіксованої ширини.Кількість додаткових бітів може бути від 0 до 24 залежно від коду відстані.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.Також ми обсилюємо наступний параметр

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

Перші 16 символів для позначення відстані є спеціальними символами для писилання до попередніх відстаней наступним чином.Кільцевий буфер останніх чотирьох відстаней ініціалізований значеннями 16,15,11 та 4 на початку потоку .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.

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

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))

Максимальне число додаткових бітів -24.Розмір алфавіту відстаней дорівнює .(16 + NDIRECT + (48 << NPOSTFIX)).

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 літералівЯк зазначено в другому параграфі,довжин вставок та копіювання закодовані простим префікс ним кодом.Кожна пара <довжина вставки,довжина копыювання>-це пара стиснених даних в мета-блоці представлена наступним триплетом:

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

Всі коди записані разом ,зокрема перший код представляють завдяки префіксноому коду по алфавіту ,в той час як два наступні завдяки цілочисленим значенням фіксованої довжини.Число додаткових бітів коливається від 0 до 24 і воно залежить від коду вставки та копіювання .Опис алфавіту кодів вставки та копіювання використовуючи алфавіти довжин кодів вставки та довжин кодів копіювання.Додаток Б,В.

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).

Код довжини коду визначають за двома бітами(рахуючи з найменш важливого) від коду довжини вставки та копіювання.Код довжини вставки визначають в діапазоні визначеному бітами на позиціях 3-5 коду довжини вставки та копіювання .За даними кодами можна отримати дісйну довжину вставки та копіювання якщо прочитати додаткові біти дані в таблиці вище.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

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. Кожен тип блоку стиснених даних представлений код типу блоку,закодованим префікс ним кодом з алфавіту типів блоку.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.Символи 2-257 представляють типи блоків від 0 до 255 відповідно.Попередній та поточний типи ініціалізовна 1 та 0,наприкінці заголовку мета-блоку.

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

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

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.

Для кожного лічильника блоку в стиснених даних є представлення у вигляді пари <код лічильника,додаткові біти>.

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

Кіліькість додаткових бітів змінюється від 0 до 24 залежно від лічильника блоку.Таблиця алфавіту кодів лічильників блоку з кількістю додаткових бітів та діапазоном лічильників блоку .Додаток Г

7. Context modeling

Моделювання контексту

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

7.1. Context modes and context ID lookup for literals

Види контексту та контекстні ID для літералів

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.ID. Є чотири види контексту за допомогою яких рахують контекстний

LSB6-беруть значення шести найменш важливих бітів байту р1.

MSB6-беруть значення шести найбільш важливих бітів р1.

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

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

Формули для обчислення контекстного 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]

7.2. Context ID for distances

Контекстний ID для відстаней

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

Кодуванння контекстного відображення

Наявні два контекстних відображення .Одни для літералів та інший для відстаней .Кожне значення у відображення є цілим числом від 0 до 255,яе виначає індекс префіксного коду який використовували при кодуванні наступного літералу та відстані .

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

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

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

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]

.Значення контекстного відображення кодують комбінацією кодування довжин серій для нульових значень та преіфксному кодуванню. Нехай RLEMAX –число кодыв довжин серій та NTREES – максимальне значення в контекстному відображення плюс 1.Звідси NTREES дорівнює кількості попарно різних значень в контекстному відображенні .

Алфавіт префіксного коду має RLEMAX + NTREES символів

Якщо RLEMAX=0 ,то кодування довжин серій не використовують ,та симоли алфавіту є значеннями контекстного відображення .

8. Static dictionary

Статичний словник

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.Проте,декодування відстані з потоку стиснених даних може створити відтані які є більшими за максимальне допустиме значення .Статичний словник складається з 3 частин :

DICT[0..DICTSIZE], масив байтів

DOFFSET[0..24], масив значень відстаней для кожної довжини

\* NDBITS[0..24], Масив бітових значень для кожної довжини.

Число слів в статичному словнику для заданої довжини визначають наступним чином

DOFFSET and DICTSIZE визначають наступним чином :

DOFFSET[0] = 0

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

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

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

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

Кожне слово статичного словника має 121 різну форму.

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]

Стрічку отриману з нестисненого потоку обчислюють завдяки перетворенню до базового словникового слова.Якщо transform\_id більше за 120 то потік визначають як некоректний.Кожне перетворення слова має наступну форму

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

Де \_і означає trasnsform\_id заначеного вище.

Кожен T\_i є одним з 21 елементарних перетворень для кожного з якого спеціально відведений метод в реалізації.

Максимальне число додаткових байтів які можна додати до слова під час

Отже ,38 байтів достатньо для зберігання будь-яких перетворених слів.

9. Compressed data format

В цьому параграфі ми опишемо формат стиснених даних через окремі елементи даних описаних вище.

9.1. Format of the stream header

Формат заголовку потоку

Заголовк потоку має лише поле

1…7 бітів: WBITS значення в діапазоні 10…24 ,яке закодоване в код змінної довжини .Додаток Ґ

Розмір розсувного вікна є максимальним значенням будь-якого посилання назад не на словник,визначають наступним чином :

window size = (1 << WBITS) - 16

9.2. Format of the meta-block header

Формат заголовку мета-блоку

Стиснений потік має мати хоча б один мета-блок.

Кожен мета-блок містить заголовку з інформацією стосовно нестисненої довжини мета-блоку ,та сигнальний біт який вказує на кінець мета блоку .

Формат заголовку мета-блоку наступний:

1 bit: ISLAST, встановлений як 1 якщо це останній блок

1 bit: ISLASTEMPTY, встановлений 1 якщо блок порожній ;

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 біти MNIBBLES, кількість напівбайт для представлення нестисненої довжини ,закодованої наступним кодом фіксованої довжини :

Value Bit Pattern

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

0 11

4 00

5 01

6 10

Якщо MNIBBLES встановлений як 0 ,то мета-блок порожній.В цьому випадку решта мета-блоку має наступний формат:

1 біт: зарезервований під 0.

2 біти: : MSKIPBYTES-кількість байт для представлення метаданих.

Довжина

MSKIPBYTES x8 біт:MSKIPLEN -1,ду MSKIPLEN –кількість байт метаданих.Це пиоле присутнє лише коли значення MSKIPBYTES більше від нуля 0…7 бітів:Заповнення байту до його кінця.

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 біт : NBLTYPESL-кількість типів блоків ,закодованих наступним кодом змінної довжини.Додаток Д

1..11 біт:NBLTYPESL – кілкість типів блоку для вставкти та копіювання ,для використано той же код змінної довжини.

1..11 біт:NBLTYPESD – кількість блоків для відстаней ,для ких використано наведений вище код змінної довжини.

Префіксний код для типів блоку для відстаней існує лише коли NBLTYPESD >=2.

2 bits: NPOSTFIX, parameter used in the distance coding

2 біти :NPOSTFIX,параметер який використовуэться в кодуванні відстаней.

4 біти :4 найбільш важливих біти NDIRECT lвикористовуються для отримання значення NDIRECT перед тим зсунувши на NPOSTFIX бітів.

2 біти :NBLTYPESL-вид контексту для типу блоку для літералів.

NTREEESL(1-11біт)- кількість префікс них дерев для літералів ,закодованих тим же кодом змінної довжини як і NBLTYPESL.

NTREESD(1-11 біт)-кількість префікс них дерев для відстаней ,закодованих тим же кодом змінної довжини що й NBLTYPESD.

9.3. Format of the meta-block data

Формат даних мета-блоку

Стиснена частина даних мета-блоку складається з послідовності команд.

Кожна з команд має наступний формат.

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.

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.

Довжина вставки та копіювання ,як зазначено вище,закодована за використання префікс них кодів з індексом типу блоку для копіювання та вставки.

Число довжини для вставки літералів представлено аналогічно до попереднього прикладу.

Наступний байт нестиснених даних кодують з літеральним префікс ним кодом де індекс визначений завдяки попереднім двом байтам,поточному типу літерального блоку та контекстного відображення .

Коди відстаней закодовані завдяки префіксному коду з поточним індексом типу блоку для відстаней .

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.

10. Decoding algorithm

Алгоритм розтискання.Додаток Е

The decoding algorithm that produces the uncompressed data is as

follows:

Якщо потік закінчується перед завершенням останнього мета-блоку ,толі потік слід відкинути.

Посилання на повторювану стрічку може посилатися

на стрічку в попередньому мета-блоці .

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 .

11. Considerations for compressor implementations

Особливості реалізації компресора .

11.1. Trivial compressor

Тривіальна реалізація

Нижче подано код простої реалізації алгоритму який породжує дійсний потік для алгоритму Brotli на мові С++.Додаток Є

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.

11.2. Aligning compressed meta-blocks to byte boundaries

Як показано вище,лише мета-блоки які йдуть одразу після мета-блоку метаданих чи порожні мета-блоки гаранотовано знаходяться на початку байта.

В декотрих застосуваннях може бути необхідно щоб всі мета-блоки знаходилися на початках байтів.

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

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.

По-перше ,якщо велики файл стиснених даних пошкоджено ,то можна відновити хоча б частину інформації.

По-друге,можна ділити дані і передавати їх по частинам.

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.По-третє,якщо послідовності нестиснених байтів стиснуті незалежно ,то можна паралельно стискати ці послідовності в одному файлі.

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.

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..

По-перше,відстані до попередніх копій в С1 не мають посилатися на слова зі статичного словника чи нетиснутих байтів U0.

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.Обидва компресори для С0 та С1 мають мати однакове розсувне вікно ,але заголовок потоку створений лише першим компрессором який створює С0.

12. Security Considerations

Завдяки цьому алгоритму можна проводити атаки на веб-браузери користувача через експлуатацію переповненн буфера при відновленні (buffer overflow) якщо ввести некоректні стиснуті дані.

Відповідно реалізації декомпресора мають виконувати перевірку використання пам’яті яка займають дані відновлені зі стиснутого потоку .

Іншим можливим способом атаки на систему через алгоритм розтискання є подання стиснутих даних,при обробці який декомпресор використає необґрунтовано багато ресурсів.

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.

.Отже,реалізації декомпресора в системах де критичним є використання даних ресурсів пам’яті мають проводи постійну перевірку даних параметрів.

Необхідно встановити ліміт на використання пам’яті одним відновленим мета-блоком з потоку.

У випадку коли весь потік (відновлений)тримають в системі,слід перевіряти загальний розмір цього потоку перед прочитанням кожного наступного стиснутого мета-блоку.

Відповідно можливі атаки на програми з використанням алгоритму компресор(наприклад веб-сервер) використовують можливість перезаповення буфера чи спричиняють надмірне використання ресурсів .

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.

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.

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 .

Додаток А

code = 0;

bl\_count[0] = 0;

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

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

next\_code[bits] = code;

}

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

len = tree[n].Len;

if (len != 0) {

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

next\_code[len]++;

}}

Додаток Б

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

Додаток В

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

Додаток Г

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

Додаток Ґ

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

Додаток Д

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

Додаток Е

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

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]]

Додаток Є

string BrotliCompressTrivial(const string& u) {

if (u.empty()) {

return string(1, 6);

}

int i;

string c;

c.append(1, 12); 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;

}