УДК 621.391.251

Л.О. КОВТУН, Д.М. МЕДЗАТИЙ
Хмельницький національний університет

ОСНОВНІ ІДЕЇ ТА АЛГОРИТМИ АДАПТИВНОГО СЛОВНИКОВОГО КОДУВАННЯ МЕТОДОМ ЛЕМПЕЛЯ-ЗІВА

У статті наведено дослідження основної ідеї адаптивного словникового кодування методом Лемпеля-Зіва. Викладено хронологічний та порівняльний аналіз вдосконалених реалізацій методу. Наведено приклад та алгоритм виконання методу адаптивного кодування LZW.

Ключові слова: метод Лемпеля-Зіва, адаптивне кодування, словникове кодування.

L.O. KOVTUN, D.M. MEDZATYI
Khmelnytskyi National University

BASIC IDEAS AND ALGORITHMS OF ADAPTIVE DICTIONARY USING METHODS ENCODE LEMPEL-ZIV

The article presents a research of the basic idea of adaptive dictionary coding method Lempel-Ziv. Expounded chronological and comparative analysis of advanced implementations of the method. An example of the method and algorithm performance adaptive coding LZW. Compression is important in the transmission, processing and storing information. The use of effective compression methods allow fully digital telephone networks radically change the nature of the final movement and the general scope of local networks. Compression efficiency is difficult to assess, but these improvements will certainly give the advantage in speed of implementation. Compression methods used for encrypting and protecting data. Squeeze provides stored and transmitted to some degree of secrecy. First, it protects them from the casual observer. Second, by removing excess it makes it impossible to set a cryptanalyst inherent statistical natural language order. Thirdly, and most importantly, the model has a very big key, decryption is not possible. The use of adaptive model means that all key depends on the text transmitted system encoding/decoding during its initialization.

Keywords: Lempel-Ziv methods, adaptive coding, dictionary coding.

Вступ. Протягом останніх років спостерігається стрімке збільшення кількості комп'ютерів, які використовуються практично у всіх сферах життя і діяльності людини. Це зумовлюється розвитком телекомунікаційних систем, ростом пропускної здатності та загальної кількості ліній зв'язку, появою і розвитком глобальних комп'ютерних мереж, які стали доступні більшості з користувачів. Невпинно збільшується і кількість користувачів ЕОМ, а також і сфери застосування комп'ютерів. Це в свою чергу викликало збільшення об'ємів інформації, що зберігається, використовується та передається. У зв'язку з розширенням спектру послуг, які може надавати глобальна мережа та постійний її розвиток, успішне виконання проектів по виконанню переводу у цифровий вигляд інформації, що зберігається (наприклад, бібліотечні архіви, модульне середовище для навчання і т.п.), слід очікувати, що об'єми інформації, яка зберігається і передається, будуть продовжувати збільшуватись з тією ж, або навіть більшою швидкістю.

Тому задача пошуку методів, які дозволять зберігати та передавати текстову, графічну, звукову та інші види інформації у найбільш компактному вигляді, є актуальною.

Мета статті. Розкрити основну ідею та навести приклади і алгоритм виконання адаптивного словникового кодування методом Лемпеля-Зіва як засобу оборотного стиску текстової інформації і послідовності дискретних даних. Провести хронологічний і порівняльний аналіз вдосконалених реалізацій даного методу.

Актуальність досліджень. Оборотний стиск особливо важливий для текстів, які записані на природних і на штучних мовах, оскільки в цьому випадку помилки неприпустимі. Хоча першочерговою областю застосування розглянутих методів є стиск текстів, ця техніка може знайти застосування і в інших випадках, включаючи оборотне кодування послідовностей дискретних даних.

Існує багато причин виділяти ресурси ЕОМ в розрахунку на стиснений вигляд, оскільки більш швидка передача даних і зменшення місця для їх зберігання дозволяють зменшити витрати і часто поліпшити показники ЕОМ. Стиск ймовірно буде залишатися в сфері уваги через постійно зростаючі обсяги даних в ЕОМ, які потрібно зберігати та передавати. Крім того, його можна використовувати для подолання деяких фізичних обмежень, таких, наприклад, як порівняно низька ширина смуги пропускання телефонних каналів.

Одним з найбільш ранніх і добре відомих методів стиску інформації є алгоритм Хаффмана, який був і залишається предметом багатьох досліджень. Однак в кінці 70-х років завдяки двом важливим переломним ідеям він був витіснений. Одна полягала у відкритті методу арифметичного кодування, який має схожу з кодуванням Хаффмана функцію, але який володіє декількома важливими властивостями, що дають можливість досягти значної переваги в стиску. Іншим нововведенням був метод Лемпеля-Зіва, який дає ефективний стиск і застосовує підхід, зовсім відмінний від хаффманівського і арифметичного. Обидві ці техніки з часу своєї першої публікації значно вдосконалились, розвинулися і лягли в основу практичних високоефективних алгоритмів.

Існують два основних способи проведення стиску: статистичний і словниковий. Кращі статистичні методи застосовують арифметичне кодування, кращі словникові – метод Лемпеля-Зіва. У словникових методах групи послідовних символів або «фраз» замінюються кодом. Замінена фраза може бути знайдена в деякому «словнику».

Словникові методи Лемпеля-Зіва. В основі цих методів лежить ідея, абсолютно відмінна від ідеї

статистичного стиску. За допомогою словникового кодера відбувається стиск заміною груп послідовних символів (фрази) індексами деякого словника. Словником є список таких фраз, які, як очікується, будуть часто використовуватися. Індекси влаштовані так, що в середньому займають менше місця, ніж закодовані ними фрази, за рахунок чого і досягається стиск. Цей тип стиску ще відомий як «макро»-кодування або метод «книги кодів».

Словникові методи зазвичай швидкі з тих причин, що один код на виході відповідає кільком вхідним символам і розмір коду зазвичай відповідає машинним словам. Словникові моделі дають досить гарний стиск, хоча і не такий гарний, як у контекстно-обмежених моделях. Можна показати, що більшість словникових кодерів можуть бути відтворені за допомогою контекстно-обмежених моделей, тому їх головною перевагою є не якість стиску, а економія машинних ресурсів.

Головним моментом при проектуванні словникової схеми є вибір розміру запису кодового словника. Деякі розробники накладають обмеження на довжину збережених фраз, наприклад, при кодуванні діадами вони не можуть бути більше двох символів. Щодо цього обмеження вибір фраз може здійснюватися статичним, напівадаптивним або адаптивним способом. Найпростіші словникові схеми застосовують статичні словники, які містять тільки короткі фрази. Вони особливо годяться для стиску записів файлу, такого, наприклад, як бібліографічна база даних, де записи повинні декодуватись випадковим чином, але при цьому одна і та ж фраза часто з'являється в різних записах.

Статичні, адаптивні і неадаптивні моделі. Залежно від доступу кодера і декодера до моделі кодування виділяють три способи моделювання: статичне, напівадаптивне і адаптивне.

Статичне моделювання використовує для всіх текстів одну і ту ж модель. Вона задається при запуску кодера, можливо на підставі зразків типу очікуваного тексту. Така ж копія моделі зберігається разом з декодером. Недолік полягає в тому, що схема буде давати необмежено поганий стиск щоразу, коли текст, який кодується, не вписується в обрану модель, тому статичне моделювання використовують тільки тоді, коли важливі в першу чергу швидкість і простота реалізації.

Напівадаптивне моделювання вирішує цю проблему, використовуючи для кожного тексту свою модель, яка будується ще до самого стиску на підставі результатів попереднього перегляду тексту (або його зразка). Перед тим, як завершено формування стисненого тексту, модель повинна бути передана декодеру. Незважаючи на додаткові витрати на передачу моделі, ця стратегія в загальному випадку окупається завдяки кращій відповідності моделі тексту.

Адаптивне (або динамічне) моделювання звільняється від пов'язаного з цією передачею витратами. Спочатку і кодер, і декодер привласнюють собі деяку порожню модель, так, ніби всі символи були б рівноімовірними. Кодер використовує цю модель для стиску чергового символу, а декодер – для його розгортання. Потім вони обидва змінюють свої моделі однаковим чином (наприклад, нарощуючи ймовірність розглянутого символу). Наступний символ кодується і отримується на основі нової моделі, а потім знову змінює модель. Кодування триває аналогічним декодуванню чином, воно підтримує ідентичну модель застосуванням такого ж алгоритму її зміни, який забезпечений відсутністю помилок під час кодування. Використовувана модель, яку до того ж не потрібно передавати явно, буде добре відповідати стисненому тексту.

У адаптивних моделях, на відміну від напівадаптивних, не відбувається їх попередній перегляд. Тому вони є більш привабливими і краще стискаючими. Таким чином, алгоритми моделей, які описуються у статті, при кодуванні і декодуванні будуть виконуватися однаково. Модель ніколи не передається явно, тому збій відбувається тільки в разі нестачі під неї пам'яті.

Важливо, щоб значення ймовірностей, які присвоєні моделлю, не були рівними 0, оскільки якщо символи кодуються $-\log(p)$ бітами, то при наближенні ймовірності до 0 довжина коду буде прагнути до нескінченності. Нульова ймовірність має місце, якщо в зразку тексту символ не зустрівся жодного разу – розповсюджена ситуація для адаптивних моделей на початковій стадії стиску. Ця проблема відома як проблема нульової ймовірності, яку можна вирішити кількома способами. Один з них полягає в тому, щоб додавати 1 до лічильника кожного символу. Альтернативні підходи основані на ідеї виділення одного лічильника для всіх нових (з нульовою частотою) символів, для подальшого використання його значення. Адаптивні схеми, які допускають великі фрази, забезпечують кращий стиск.

Опис методу і його вдосконалені реалізації. Майже всі практичні словникові кодери належать сімейству алгоритмів, що походять з роботи Зіва і Лемпеля. Сутність полягає в тому, що фрази замінюються вказівником на те місце, де вони в тексті вже раніше з'являлися. Це сімейство алгоритмів називається методом Лемпеля-Зіва і позначається як LZ-стиск. Цей метод швидко пристосовується до структури тексту і може кодувати короткі функціональні слова, оскільки вони дуже часто в ньому з'являються. Нові слова і фрази можуть також формуватися з частин слів, які раніше зустрічались.

Декодування стисненого тексту здійснюється безпосередньо – відбувається проста заміна вказівника готовою фразою зі словника, на яку той вказує. Однією з форм такого вказівника є пара (m,l), яка замінює фразу з першого символу, що починається зі зміщенням *m* у вхідному потоці. Наприклад, вказівник (7,2) адресує 7-й і 8-й символи початкового рядка. Використовуючи це позначення, рядок «abbaabbbabab» буде закодований як «abba (1,3) (3,2) (8,3)». Зауважимо, що незважаючи на рекурсію в останньому вказівнику, створене кодування не буде двозначним.

Поширено невірне уявлення, що за поняттям LZ-методу стоїть єдиний алгоритм. Спочатку це був

варіант для вимірювання «складності» рядків, що призвів до двох різних алгоритмів стиску. Ці перші статті були глибоко теоретичними і лише наступні переклади інших авторів дали більш доступне уявлення. Ці тлумачення містять в собі багато нововведень, що створюють туманне уявлення про те, що таке LZ-стиск насправді. Через велику кількість варіантів цього методу кращий опис можна здійснити тільки через його зростаюче сімейство, де кожен член відображає своє рішення розробника. Ці версії відрізняються одна від одної в двох головних чинниках: чи є межа зворотного ходу вказівника, і на які підрядки з цієї множини він може посилатися. Просування вказівника в раніше переглянуту частину тексту може бути необмеженим (розширюється вікно) або обмежено вікном постійного розміру з N попередніх символів, де N зазвичай становить кілька тисяч. Обрані підрядки також можуть бути необмеженими або обмеженими безліччю фраз, які обрані згідно деякому задуму.

Кожна комбінація цих умов є компромісом між швидкістю виконання, об'ємом необхідної оперативної пам'яті і якістю стиску. Вікно, яке розширюється, пропонує кращий стиск за рахунок організації доступу до більшої кількості підрядків.

Але по мірі збільшення вікна кодер може сповільнити свою роботу через зростання часу пошуку відповідних підрядків, а стиск може погіршитись через збільшення розмірів вказівників. Якщо пам'яті для вікна буде не вистачати, стається збій процесу, що також погіршить стиск доки не відбудеться нового збільшення вікна. Вікно постійного розміру позбавлено цієї проблеми, але містить менше підрядків, які доступні вказівнику. Обмеження множини доступних підрядків розмірами фіксованого вікна зменшує розмір вказівників і пришвидшує кодування.

Відзначено найголовніші варіанти LZ-методу, які нижче будуть розглянуті більш детально. Всі вони походять від одного з двох різних підходів, які описані Зівом і Лемпелем [1–3, 5–7] і помічені відповідно як LZ77 і LZ78. Ці два підходи зовсім різні, хоча деякі автори вносять плутанину твердженнями про їх ідентичність. Термін «LZ-схеми» походить від імен їх винахідників. Зазвичай кожен наступний розглянутий варіант є поліпшенням більш раннього, і в наступних описах ми відзначимо їх попередників.

LZ77. Це перша опублікована версія LZ-методу. У ній вказівники позначають фрази в вікні постійного розміру, які є попередніми позиціями коду. Максимальна довжина вказівника підрядків, що замінюються, визначаються параметром F (зазвичай 10-20). Ці обмеження дозволяють LZ77 використовувати «змінне вікно» з N символів. З них перші $(N-F)$ були вже закодовані, а останні F складають попереджуючий буфер.

При кодуванні символу в перших $(N-F)$ символах вікна визначається найдовший рядок, що збігається з цим буфером. Він може частково перекривати буфер, але не може бути самим буфером.

Знайдена найбільша відповідність потім кодується тріадою <*i,j,a*>, де *i* – його зміщення від початку буфера, *j* – довжина відповідності, *a* – перший символ, який не відповідає підрядку вікна. Потім вікно зсувається вправо на (*j*+1) символ і готове до нового кроку алгоритму. Прив'язка певного символу до кожного вказівника гарантує, що кодування буде виконуватися навіть в тому випадку, якщо для першого символу попереджувального буфера не буде знайдено відповідності.

Об'єм пам'яті, необхідний кодеру і декодеру, обмежується розміром вікна. Зміщення (*i*) в тріаді може бути представлено $\lceil \log(N-F) \rceil$ бітами, а кількість символів, які замінюються тріадою, $j - \lceil \log F \rceil$ бітами.

Декодування здійснюється дуже просто і швидко. При цьому підтримується той же порядок роботи з вікном, що і при кодуванні, але на відміну від пошуку однакових рядків він, навпаки, копіює їх з вікна відповідно до чергової тріади.

Зів і Лемпель показали, що, при досить великому N LZ77 може стиснути текст не гірше, ніж будь-який, спеціально на нього налаштований напівадаптивний словниковий метод. Цей факт інтуїтивно підтверджується тим міркуванням, що напівадаптивна схема повинна мати крім самого кодованого тексту ще й словник, тоді як для LZ77 словник і текст – це одне і те ж. А розмір елемента напівадаптивного словника не меншого розміру відповідної йому фрази в кодованому LZ77 тексті.

Кожен крок кодування LZ77 вимагає однакової кількості часу, що є його головним недоліком в разі, якщо воно буде великим. Тоді пряма реалізація може зажадати до $(N-F)*F$ операцій порівнянь символів в фрагменті, який аналізується. Ця властивість повільного кодування і швидкого декодування властива для багатьох LZ-схем. Швидкість кодування може бути збільшена за рахунок використання двійкових дерев, дерева цифрового пошуку або хеш-таблиці, але обсяг необхідної пам'яті при цьому також зросте. Тому цей тип стиску є найкращим для випадків, коли одного разу закодований файл (переважно на швидкій ЕОМ з достатньою кількістю пам'яті) багато разів розгортається і, можливо, на повільній машині. Це часто трапляється на практиці при роботі, наприклад, з діалоговими довідковими файлами, посібниками, новинами, телетекстами і електронними книгами.

LZR. Подібний LZ77, за винятком того, що він дозволяє вказівникам в уже переглянутій частині тексту адресувати будь-яку позицію. Для LZ77 це аналогічно встановленню параметра N більше розміру вхідного тексту. Оскільки *i* та *j* в тріаді <*i,j,a*> можуть зростати на довільно велике значення, вони представляються цілими кодами змінної довжини. Цей метод використаний Еліасом і позначений як $C(w')$. При кодуванні цілого позитивного числа довжина коду зростає в логарифмічній залежності від його розміру. Наприклад, коди для чисел 1, 8 і 16 відповідно становитимуть 0010,10010000 та 101100000.

Через відсутність обмеження на зростання словника, LZR не часто застосовується на практиці, оскільки при цьому процесі кодування потрібно все більше пам'яті для розміщення тексту, в якому визначаються відповідності. При використанні лінійного пошуку *n*-символьний текст буде закодований за час $O(n^2)$.

LZSS. Результатом роботи LZ77 і LZR є серія тріад, що представляють собою символи, які строго чергуються. Використання явного символу слідом за кожним вказівником є на практиці марнотратним, так як часто його можна зробити частиною такого вказівника. LZSS працює над цією проблемою, застосовуючи вільну послідовність вказівників і символів, причому останні включаються у випадку, якщо необхідно, щоб вказівник мав більший розмір, ніж кодований ним символ. Вікно з *N* символів застосовується так само, як і в LZ77, тому розмір вказівників постійний. До кожного вказівника або символу додається додатковий біт для відмінності їх між собою, а для усунення невикористовуваних бітів вихідна послідовність пакується LZSS.

LZB. Незалежно від довжини фрази, яка ним адресується, кожен вказівник в LZSS має сталий розмір. На практиці фрази з однаковою довжиною зустрічаються набагато частіше за інші, тому з вказівниками різної довжини можна досягнути кращого стиску. LZB став результатом експериментів по оцінці різних методів кодування вказівників, теж як явних символів, так і прапорців, які розрізняють їх. Метод дає набагато кращий, ніж LZSS, стиск і має додаткову перевагу в меншій чутливості до вибору параметрів. Першою складовою вказівника є позиція початку фрази від початку вікна. LZB працює відносно цієї компоненти. Спочатку, коли символів у вікні 2, розмір дорівнює 1 біту, потім, при 4-х символах в вікні, зростає до 2 бітів, і т.д., поки вікно не стане містити *N* символів. Для кодування другої складової (довжини фрази) вказівника LZB застосовують схему кодів змінної довжини Еліаса-С (gamma). Оскільки цей код може представляти фразу будь-якої довжини, то ніяких обмежень на неї не накладається.

LZH. Для подання вказівника LZB застосовують кілька простих кодів, але краще представлення може бути здійснено на підставі ймовірності їх розподілу за допомогою арифметичного кодування або кодування Хаффмана. LZH-система подібна LZSS, але застосовує для вказівників і символів кодування Хаффмана. При застосуванні одного з цих статистичних кодерів до LZ-вказівників, через витрати на передачу великої кількості кодів (навіть в адаптивному режимі) виявилося важко поліпшити стиск. Крім того, кінцевій схемі не вистачає швидкості і простоти LZ-методу.

LZ78. Новий підхід до адаптивного словникового стиску важливий як з теоретичної, так і з практичної точок зору. Він був першим в сімействі схем, що розвиваються паралельно (і в плутанині) з LZ77. Незалежно від можливості вказівників звертатися до будь-якого вже переглянутого рядка, переглянутий текст розбирається на фрази, де кожна нова фраза є найдовша з уже переглянутих плюс один символ. Вона кодується як індекс її префікса плюс додатковий символ. Після цього нова фраза додається до списку фраз, на які можна посилатися.

Наприклад, рядок "aaabbabaabaaabab", як показано в таблиці, ділиться на 7 фраз. Кожна з них кодується як вже фраза, яка раніше зустрічалась, плюс поточний символ. Наприклад, останні три символи кодуються як фраза номер 4 ("ba"), за якою слідує символ "b". Фраза номер 0 – порожній рядок.

Ввід	a	aa	b	ba	baa	baaa	bab
Номер фрази	1	2	3	4	5	6	7
Вивід	(0,a)	(1,a)	(0,b)	(3,a)	(4,a)	(5,a)	(4,b)

Дальність просування вперед вказівника необмежена (тобто немає вікна), тому в міру виконання кодування накопичується все більше фраз. Допущення довільно великої їх кількості вимагає по мірі розбору збільшення розміру вказівника. Коли розібрано p фраз, вказівник представляється $\lceil \log p \rceil$ бітами. На практиці словник не може продовжувати рости нескінченно. При вичерпанні доступної пам'яті вона очищається і кодування триває як би з початку нового тексту.

Привабливою практичною властивістю LZ78 є ефективний пошук в дереві цифрового пошуку за допомогою вставки. Кожен вузол містить номер фрази, яка ним представляється. Оскільки фраза, яка вставляється, буде лише на один символ довше однієї з тих, які їй передують, то для здійснення цієї операції кодеру потрібно буде тільки спуститися вниз по дереву на одну дугу.

Важливою теоретичною властивістю LZ78 є те, що при створені вихідного тексту стаціонарним ергодичним джерелом стиску є приблизно оптимальним по мірі зростання введення. Це означає, що LZ78 створить нескінченно довгий рядок до мінімального розміру, визначеного ентропією джерела. Лише деякі методи стиску володіють цією властивістю. Джерело є ергодичним, якщо будь-яка вироблена їм послідовність все точніше характеризує його по мірі зростання своєї довжини. Оскільки це досить слабке обмеження, то може здатися, що LZ78 є вирішенням проблеми стиску текстів. Однак оптимальність з'являється, коли розмір введення прямує до нескінченності, а більшість текстів значно коротші. Вона заснована на розмірі явного символу, який значно менший за розмір всього коду фрази. Оскільки його довжина 8 біт, він буде займати всього 20% виведення при створенні 2^{40} фраз. Навіть якщо можливе тривале введення, то пам'ять вичерпається задовго до того, як стиск стане оптимальним.

Реальна задача – подивитися, як швидко LZ78 сходиться до цієї межі. Як показує практика, збіжність ця відносно повільна, в цьому метод можна порівняти з LZ77. Причина великої популярності LZ-техніки на практиці не в її наближенні до оптимальності, а по тій причині, що деякі варіанти дозволяють здійснювати високоефективну реалізацію.

LZW. Перехід від LZ78 до LZW паралельний переходу від LZ77 до LZSS. Включення явного символу в вихідну послідовність після кожної фрази часто є марнотратним. LZW управляє повсюдним виключенням цих символів, тому вихідна послідовність містить тільки вказівники. Це досягається ініціалізацією списку фраз, що включає всі символи вихідного алфавіту. Останній символ кожної нової фрази кодується як перший символ наступної фрази. Особливої уваги потребує ситуація, що виникає при декодуванні, якщо фраза кодувалася за допомогою іншої, яка безпосередньо їй передувала, але це не є непереборною проблемою.

LZW був спочатку запропонований як метод стиску даних при записі їх на диск за допомогою спеціального обладнання каналу диска. Через високу вартість інформації, при такому підході важливо, щоб стиск відбувався дуже швидко. Передача вказівників може бути спрощена і прискорена при використанні для них постійного розміру в (як правило) 12 бітів. Після розбору 4096 фраз нових до списку додати вже не можна, і кодування стає статичним. Незалежно від цього, на практиці LZW досягає прийнятного стиску і для адаптивної схеми є дуже швидким. Перший варіант Міллера і Вегмана з [4] є незалежним винаходом LZW.

LZC. Схема, що застосовується програмою COMPRESS, яка використовується в системі UNIX. Вона починалася як реалізація LZW, але потім кілька разів змінювалася з метою досягнення кращого і швидшого стиску. Результатом стала схема з високими характеристиками, яка в даний час є однією з найбільш корисних.

Рання модифікація працювала з вказівниками змінної довжини як в LZ78. Розділ програми, що працює з вказівниками, для ефективності був написаний на асемблері. Щоб уникнути переповнення пам'яті словником у якості параметра повинна передаватися максимальна довжина вказівника (зазвичай 16 біт, але для невеликих машин менше). Перш ніж очистити пам'ять після заповнення словника, LZC стежить за коефіцієнтом стиску. Тільки після початку його погіршення словник очищається і знову будується з самого початку.

LZT. Заснований на LZC. Головна відмінність полягає в тому, що коли словник заповнюється, місце для нових фраз створюється скиданням найменш використовуваної останнім часом фрази (LRU-заміщення). Це ефективно здійснюється підтриманням саморегульованого списку фраз, організованого у вигляді хеш-таблиці. Список спроектований так, що фраза може бути замінена за рахунок невеликого числа операцій з вказівниками. Цей алгоритм трохи повільніший LZC, але більш продуманий вибір фраз в словнику забезпечує досягнення такого ж коефіцієнта стиску з меншими витратами пам'яті.

LZT також кодує номера фраз більш ефективно, ніж LZC за допомогою трохи кращого методу розбиття на фрази двійкового кодування (його можна застосувати також і до деяких інших LZ-методів). При цьому кодеру і декодеру потрібні невеликі додаткові витрати, які є незначними в порівнянні з завданням пошуку та підтримки LRU-списку. Другий варіант Міллера і Вегмана є незалежним винаходом LZT.

LZMV. Всі похідні від LZ78 алгоритми створюють для словника нову фразу шляхом додавання до вже існуючої фрази одного символу. Цей метод досить довільний, хоча, безсумнівно, робить реалізацію простою. LZMV використовує інший підхід для формування записів словника. Нова фраза створюється за допомогою конкатенації останніх двох кодованих фраз. Це означає, що фрази будуть швидко зростати, і не всі їх префікси будуть знаходитися в словнику. Фрази, які застосовуються нечасто, як і в LZT, при обмеженому розмірі словника будуть видалятися, щоб забезпечити адаптивний режим роботи. Взагалі, стратегією швидкого конструювання фрази LZMV досягається кращий стиск в порівнянні з нарощуванням фрази на один символ за раз.

LZJ. Являє собою новий підхід до LZ-стиску. Спочатку передбачуваний словник LZJ містить кожний унікальний рядок з уже переглянутої частини тексту, обмежений по довжині деяким максимальним значенням h. Кожній фразі словника присвоюється порядковий номер фіксованої довжини в межах від 0 до H-1. Для гарантії, що кожен рядок буде закодований, в словник включається безліч вихідних символів. Коли словник повний, він скорочується видаленням рядка, що з'являвся на вході тільки один раз.

Кодування і декодування **LZJ** виконується на основі структури дерева цифрового пошуку для зберігання підрядків з уже закодованої частини тексту. Висота дерева обмежена ***h*** символами і воно не може містити більше H вузлів. Рядок розпізнається за унікальним номером, який присвоєно відповідному йому вузлу. Процес декодування повинен підтримувати таке ж дерево методом перетворення номера вузла назад до підстроки, здійснюючи шлях вгору по дереву.

LZFG. Запропонований Фіалою і Гріні – це одні з найбільш практичних **LZ**-варіантів. Він дає швидке кодування і декодування, гарний стиск, не вимагаючи при цьому надмірної пам'яті. Він схожий з **LZJ** в тому, що втрати від можливості кодування однієї і тієї ж фрази двома різними вказівниками усуваються зберіганням кодованого тексту в вигляді дерева цифрового пошуку та розміщенням в вихідний файл позиції в дереві.

З використанням LZFG досягається більш швидкий стиск, ніж **LZJ**, за допомогою техніки з **LZ78**, де вказівники можуть починатися тільки за межами попередньої розібраної фрази. Це означає, що для кожної фрази, яка кодується, в словник вставляється одна фраза. На відміну від **LZ78**, вказівники включають компоненту по суті необмеженої довжини, яка вказує, як багато символів має бути скопійовано. Закодовані символи поміщені в вікні (в стилі **LZ77**), і фрази, які покидають вікно, видаляються з дерева цифрового пошуку. Для ефективного представлення кодів використовуються коди змінної довжини. Нові

фрази кодуються за допомогою лічильника символів, які слідують за символами.

Алгоритм методу LZW та дослідження ефективності його використання.

Алгоритм запропонованого методу наведено на рис. 1. Він полягає в наступному: на першому етапі відбувається ініціалізація словника всіма можливими односимвольними фразами та ініціалізація вхідної фрази W першим символом повідомлення (1). Далі відбувається зчитування наступного символу K (2) з повідомлення, яке кодується. Якщо настає кінець повідомлення і надходження символів більше не очікується, то необхідно забезпечити вивід коду для W (3-7-8-9).

В іншому випадку, якщо фраза W(K) вже є у словнику, то необхідно вхідній фразі надати значення W(K) і перейти до кроку №2, інакше видати код W, додати W(K) у словник, надати вхідній фразі значення K і перейти кроку №2 (4-5-6).

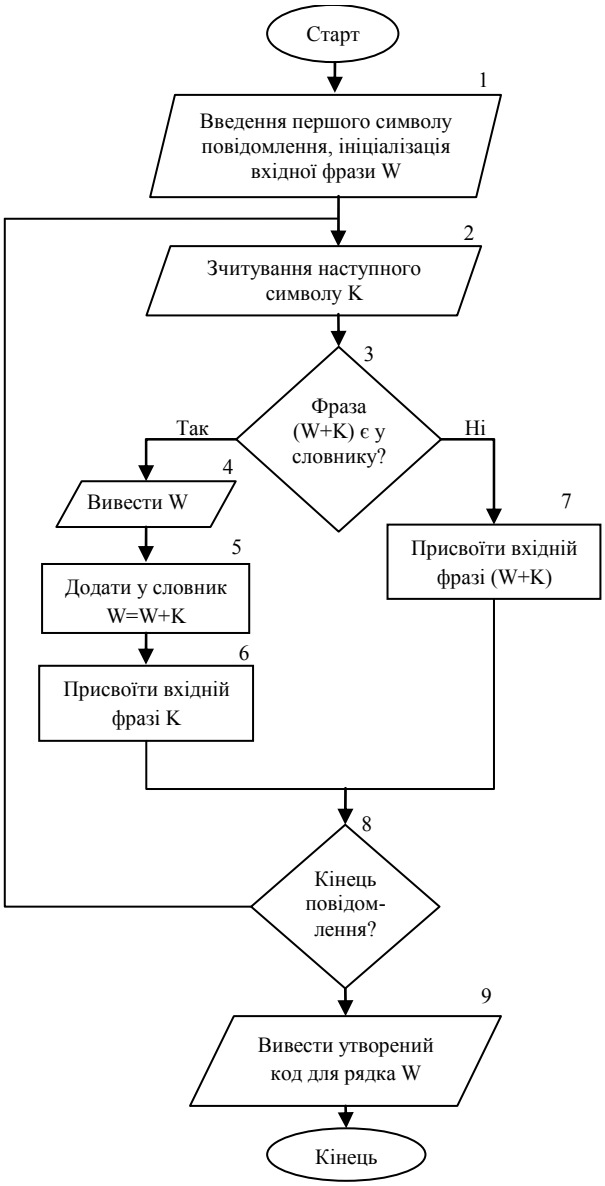

Рис. 1. Алгоритм виконання стиску повідомлень за методом LZW

У таблиці 1 наведено приклад стиску послідовності QWEQWEQQWWEEWQWEQWE символів зі словником, який містить 512 символів. Основною особливістю є те, що словник формується шляхом розширення таблиці ASCII додаванням необхідної кількості бітів. Якщо словник містить 512 символів, то додаватись буде 1 біт, якщо 1024 – то 2, 2048 – 3 і т.д. Оскільки словник в наведеному прикладі містить 512 символів, то для заданої послідовності біт «0» буде додаватись, якщо символ кодується з таблиці ASCII, а «1» – якщо новий символ є розширенням її.

Для порівняння визначимо довжину повідомлення до стиску $l_{\text{пов.}} = N_{In} \cdot l_{ASCII}$ та після стиску $l_{LZW} = N_{Out} \cdot l_{ASCII+}$, де $l_{\text{пов.}}$, l_{LZW} – довжина повідомлення до стиску та після відповідно, N_{In} – кількість символів на вході кодера, N_{Out} – кількість символів на виході кодера, l_{ASCII} – довжина двійкової комбінації символу згідно таблиці ASCII ($l_{ASCII} = 8$ біт), l_{ASCII+} – довжина двійкової комбінації символу

згідно розширеної таблиці (словника, що створений під час кодування методом LZW) ASCII+ ($l_{ASCII+} = 9$ біт). Отже $l_{\text{пов.}} = 19 \cdot 8 = 152$ біт , $l_{LZW} = 11 \cdot 9 = 99$ біт.

Таблиця 1

Результати стиску повідомлення методом LZW

Вхідне повідомлення	Символ з таблиці ASCII		Рядок символів, який порівнюємо з символами у словнику	Чи є співпадіння з символа-ми у словнику	Словник	Вихідні символи	Символ з таблиці ASCII+	
	Порядковий номер	Двійкове представлення символів					Порядковий номер	Двійкове представлення символів
Q	81	01010001						
W	87	01010111	QW	–	QW=256	Q	81	001010001
E	69	01000101	WE	–	WE=257	W	87	001010111
Q	81	01010001	EQ	–	EQ=258	E	69	001000101
W	87	01010111	QW	+ (256)				
E	69	01000101	QWE	–	QWE=259	QW	256	100000000
Q	81	01010001	EQ	+ (258)				
Q	81	01010001	EQQ	–	EQQ=260	EQ	258	100000010
W	87	01010111	QW	+ (256)				
W	87	01010111	QWW	–	QWW=261	QW	256	100000000
E	69	01000101	WE	+ (257)				
E	69	01000101	WEE	–	WEE=262	WE	257	100000001
W	87	01010111	EW	–	EW=263	E	69	001000101
Q	81	01010001	WQ	–	WQ=264	W	87	001010111
W	87	01010111	QW	+ (256)				
E	69	01000101	QWE	+ (259)				
Q	81	01010001	QWEQ	–	QWEQ=265	QWE	259	100000011
W	87	01010111	QW	+ (256)				
E	69	01000101	QWE	+ (259)		QWE	259	100000011

Таким чином, використовуючи метод стиску LZW вхідне повідомлення стало коротшим на 53 біт, а це майже 35%. Чим вхідне повідомлення довше та у ньому більше однакових послідовностей символів, тим ці цифри можуть сягати більших значень.

Висновки. Існують 3 напрямки досліджень в даній області: підвищення ефективності стиску, прискорення роботи алгоритму і здійснення стиску на підставі нової системи контекстів. Зараз кращі схеми досягають стиску в 2,3-2,5 бітів/символ для англійського тексту. Показник іноді може бути трохи поліпшений шляхом використання великих обсягів пам'яті.

Одним напрямком для досліджень є підлаштування схем стиску до вітчизняних мов. Сьогоднішні системи працюють повністю на лексичному рівні. Використання великих словників з синтаксичною і семантичною інформацією може дозволити машинам отримати перевагу від наявної в тексті високорівневої зв'язності. Однак необхідно мати на увазі, що дуже велика кількість слів звичайного англійського тексту в звичайному англійському словнику може бути не знайдена. Більшість словників не містять імен людей, назв місць, інститутів, торгових марок і т.д., хоча вони становлять основну частину майже всіх документів. Тому спеціальні алгоритми стиску, взяті в розрахунку на лінгвістичну інформацію вищого рівня, будуть безсумнівно залежати від системної сфери. Ймовірно, що пошуки методів поліпшення характеристик стиску в даному напрямку будуть об'єднуватися з дослідженнями в області контекстуального аналізу з таких проблем як витяг ключових слів і автоматичне абстрагування.

Другий підхід діаметрально протилежний описаному вище наукомісткому напрямку, і полягає в підтримці повної адаптивності системи і пошуку поліпшень в наявних алгоритмах. Необхідні кращі шляхи організації контекстів і словників. Іншою темою, яка варта уваги, для досліджень є методи виділення кодів уходів в частково відповідних контекстуальних моделях.

Пошуки більш швидких алгоритмів, які сильно впливають на розвиток методів стиску текстів, будуть безсумнівно продовжені в майбутньому. Постійний компроміс між вартостями пам'яті і обчислень стимулюватиме подальшу роботу над більш складними системами даних, які вимагають великих обсягів пам'яті, але прискорюють доступ до інформації, що зберігається. Буде розвиватися апаратура, наприклад, дослідники експериментують з арифметичним кодуванням на мікросхемі, тоді як Гонзалес-Сміт і Сторер розробили для методу стиску Лемпеля-Зіва паралельні алгоритми пошуку. Загалом, збільшення варіантів апаратних реалізацій буде стимулювати і урізноманітнювати дослідження щодо поліпшення алгоритмів, які використовують їх.

Ще однією сферою застосування є шифрування і захист даних. Стиск надає збереженим і переданим повідомленням деяку ступінь секретності. По-перше, він захищає їх від випадкового спостерігача. По-друге, за допомогою видалення надмірності він не дає можливості криптоаналітику встановити властивий природній мові статистичний порядок. По-третє, що найважливіше, модель діє як дуже великий ключ, без якого розшифрування неможливе. Застосування адаптивної моделі означає, що ключ залежить від всього

тексту, переданого системі кодування/декодування під час її ініціалізації. Також в якості ключа може бути використаний деякий префікс стиснених даних, що визначає модель для подальшого декодування.

Література

1. Шульгин В. И. Основы теории передачи информации: Ч.I. Экономное кодирование : учебное пособие / Виталий Иванович Шульгин. – Харьков : Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2003. – 102 с.

2. Основи теорії інформації та кодування : підручник / І. В. Кузьмін, І. В. Троцишин, А. І. Кузьмін, В. О. Кедрус, В. Р. Любчик. – 3-є вид., переробл. та доповн. – Хмельницький : ХНУ, 2009. – 373 с.

3. Теорія інформації та кодування : підручник / В. І. Барсов, В. А. Краснобаєв, З. В. Барсова, О. І. Тиртишніков, І. В. Авдєєв ; ред. : В. І. Барсов ; МОНМС України, Укр. інж.-пед. акад., Полтав. нац. техн. ун-т ім. Ю. Кондратюка. – Полтава, 2011. – 321 с.

4. Данченков Я. В. Теорія інформації : навч. посіб. / Я. В. Данченков ; Нац. ун-т вод. госп-ва та природокористування. – Рівне : НУВГП, 2012. – 111 с.

5. Цымбал В.П. Теория информации и кодирование : учебник / Цымбал В.П. – 4-е изд., перераб. и доп. – К. : Вища школа, 1992. – 263 с.

6. Кохманюк Д. Сжатие данных: как это делается / Д. Кохманюк // Index Pro. – 1992. – № 1. – С. 18–29.

7. Жураковский Ю.П. Теорія інформації та кодування / Ю.П. Жураковский, В.П. Полоторак. – К. : Вища школа, 2001. – 255 с.

References

1. Shulhyn V. Y. Osnovy teoryy peredachy ynformatsyy: Ch.I. Ekonomnoe kodyrovanye : uchebnoe posobye / Vytalyi Yvanovych Shulhyn. – Kharkov : Nats. aerokosm. un-t «Khark. avyats. yn-t», 2003. – 102 s.

2. Osnovy teorii informatsii ta koduvannia : pidruchnyk / I. V. Kuzmin, I. V. Trotsyshyn, A. I. Kuzmin, V. O. Kedrus, V. R. Liubchyk. – 3-ye vyd., pererobl. ta dopovn. – Khmelnytskyi : KhNU, 2009. – 373 c.

3. Teoriia informatsii ta koduvannia : pidruchnyk / V. I. Barsov, V. A. Krasnobaiev, Z. V. Barsova, O. I. Tyrtyshnikov, I. V. Avdieiev ; red. : V. I. Barsov ; MONMS Ukrainy, Ukr. inzh.-ped. akad., Poltav. nats. tekhn. un-t im. Yu. Kondratiuka. – Poltava, 2011. – 321 c.

4. Danchenkov Ya. V. Teoriia informatsii : navch. posib. / Ya. V. Danchenkov ; Nats. un-t vod. hosp-va ta pryrodokorystuvannia. – Rivne : NUVHP, 2012. – 111 c.

5. Tsymbal V.P. Teoryia ynformatsyy y kodyrovanye : uchebnyk / Tsymbal V.P. – 4-e yzd., pererab. y dop. – K. : Vyshcha shkola, 1992. – 263 s.

6. Kokhmaniuk D. Szhatye dannykh: kak эto delaetsia / D. Kokhmaniuk // Index Pro. – 1992. – # 1. – S. 18–29.

7. Zhurakovskyi Yu.P. Teoriia informatsii ta koduvannia / Yu.P. Zhurakovskyi, V.P. Polotorak. – K. : Vyshcha shkola, 2001. – 255 s.

Рецензія/Peer review : 13.06.2017 р. Надрукована/Printed :09.09.2017 р.
Рецензент: д.т.н., проф. Мартинюк В.В.