КІБЕРНЕТИКА

УДК 51.681.3

А.В. АНІСІМОВ
Київський національний університет імені Тараса Шевченка, Київ, Україна
e-mail: *a.v.anisimov@knu.ua.*

І.О. ЗАВАДСЬКИЙ
Київський національний університет імені Тараса Шевченка, Київ, Україна
e-mail: *ihorzavadskyi@knu.ua.*

Т.С. ЧУДАКОВ
Київський національний університет імені Тараса Шевченка, Київ, Україна
e-mail: *timofey.chudakov@gmail.com.*

СТИСНЕННЯ ПРИРОДНОМОВНИХ ТЕКСТІВ РЕВЕРСНИМИ МУЛЬТИРОЗДІЛЬНИКОВИМИ КОДАМИ

Анотація. У статті досліджено бінарні реверсні мультироздільникові (РМР) стискальні коди. РМР-коди мають низку корисних властивостей, як-от: однозначна декодовність, повнота, універсальність, синхронізовність, розпізнавання за допомогою скінченного автомата, а також можливість швидкого пошуку даних у закодованому файлі. Побудовано просте монотонне відображення з множини цілих невід'ємних чисел на множину кодових слів, а на його основі — швидкий побайтовий декодувальний алгоритм. Комп'ютерні експерименти демонструють, що РМР-код можна декодувати майже з тією самою швидкістю, що й код SCDC й у рази швидше, ніж код Фібоначчі. Якщо порівняти з відомими кодами подібного типу, РМР-коди демонструють кращий коефіцієнт стиснення природномовних текстів (більш ніж у 4 рази ближче до ентропійної межі, ніж SCDC). Також описано технологію передоброблення природномовних текстів, яка в поєднанні з кодуванням РМР-кодами підвищує ефективність потужних сучасних архіваторів.

Ключові слова: стиснення, архіватор, код, мультироздільниковий.

ВСТУП

Стиснення великих текстових корпусів є одним із ключових елементів сучасних інформаційно-пошукових систем. Методи стиснення тексту можна розділити на дві групи: методи, що використовують окремі символи як елементи алфавіту, і методи, що використовують слова як атомарні символи. Загалом методи другої групи мають кращі коефіцієнти стиснення, тому зосередимо увагу саме на них. Добре відомі класичні методи статистичного кодування можна застосувати до стиснення тексту на рівні слів та забезпечити коефіцієнти стиснення, близькі до теоретичної межі, визначеної ентропією Шеннона. Йдеться про арифметичне кодування, коди на основі асиметричних систем числення [1] і певною мірою коди Хаффмана [2]. Однак велике значення має не тільки ступінь стиснення, але й такі характеристики, як можливість пошуку у стиснутому потоці даних, висока швидкість декодування та обмеження можливого поширення помилок. Як відомо, зазначені коди недостатньо задовольняють перелічені вимоги.

Альтернативний підхід полягає у використанні кодів змінної довжини із суфіксами-роздільниками, відомих як патерн-коди. Роздільники — це спе-

ціальні бітові послідовності, що позначають початок або кінець кодового слова. Правильний вибір роздільників гарантує, що будь-яку частину бітового потоку коду можна в унікальний спосіб декодувати незалежно від контексту. Це забезпечує здатність коду до синхронізації, тобто обмеження області впливу можливої помилки лише до наступного роздільника. До того ж це дає можливість швидко шукати рядок у стиснутому файлі алгоритмом типу Боєра–Мура. Інші властивості патерн-кодів, зокрема повноту й універсальність (у сенсі Еліаса [3]), доведено в [4–6], де також визначено асимптотичну щільність патерн-кодів.

Зазвичай зазначені властивості патерн-кодів досягаються за рахунок зниження коефіцієнта стиснення. У разі використання символьного алфавіту це зниження є дуже суттєвим. Однак алфавіт зі слів містить значно більше елементів, а розподіл їхніх частот є більш плоским, що вирівнює коефіцієнт стиснення різноманітних кодів. Як показано у [7, 8], розмір текстів природною мовою, стиснутих патерн-кодами на алфавіті зі слів, може перевищувати рівень ентропії H0 Шеннона лише на 3–5%.

Ймовірно, найвідомішим класом патерн-кодів є коди Фібоначчі, що ґрунтуються на використанні чисел Фібоначчі вищих порядків. Математичні властивості цих кодів вперше досліджено у статті [9], де, зокрема, доведено їхню повноту та універсальність. Однак для коду Фібоначчі третього порядку Р. Капочеллі описав випадок, коли декодер повністю розсинхронізовується [10]. Як альтернативу він запропонував патерн-код з роздільником 01^r. Цей код є синхронізованим, повним і універсальним. До того ж він містить більше коротких кодових слів, ніж відповідний код Фібоначчі порядку $r+1$, тоді як його асимптотична щільність нижча.

Ідея патерн-кодів була розвинута з винайденням байтових кодів, у яких кодові слова складаються з цілого числа байтів. Це тегові коди Хаффмана [11], щільні коди з кінцевими тегами (ETDC) [12], (s, c)-щільні коди (SCDC) [13] і байтові коди з обмеженими префіксними властивостями (RPBC) [14]. Завдяки байтовій структурі кодування та декодування цих кодів можна виконати дуже швидко, чим і пояснюється їхня практична привабливість. З-поміж байтових кодів найкраще стиснення текстових даних забезпечують RPBC, хоча і тут їхні показники ще далекі від ентропійної межі (обсяги стиснутих файлів перевищують її приблизно на 14–16%). До того ж кодові слова в RPBC не відокремлюються роздільниками, тому ці коди не синхронізовні та унеможливлюють швидкий пошук у стиснутому файлі.

Поява байтових кодів стимулювала розроблення побайтових алгоритмів декодування бітових кодів змінної довжини. Вперше такі алгоритми були розроблені для кодів Фібоначчі [7]. Автори розкрили потенціал коду Fib3 як найкращого коду Фібоначчі для стиснення тексту, а також розробили швидкі алгоритми декодування та пошуку в стиснених текстах, що опрацьовують цілі байти коду. Однак ці алгоритми залишаються значно повільнішими (більш ніж удвічі), ніж декодувальні алгоритми для байтових кодів. Водночас середня довжина кодового слова коду Fib3 перевищує значення ентропії на 6–8%, що набагато краще, ніж для кодів SCDC/RPBC, але все ще залишає простір для вдосконалення.

Щоб краще припасувати стискальний код до розподілу ймовірностей символів алфавіту, неважко узагальнити визначення патерн-кодів, допустивши використання кількох суфіксів-роздільників. Однак безпосередня імплементація цієї ідеї потребує ретельного дослідження питань, пов’язаних із синхронізовністю та асимптотичною щільністю коду. Запропонований Р. Капочеллі роздільник вигляду 01^k забезпечує синхронізовність. Проте в коді можна використати тільки один

роздільник цього типу, оскільки коротші роздільники будуть префіксами довших (це справедливо також для роздільників кодів Фібоначчі вигляду 1^k). Вдалими кандидатами на роль кількох роздільників в одному коді видаються роздільники вигляду 01^k0. Проте можливі й подальші удосконалення.

Розглянемо код із кількома суфіксними роздільниками вигляду 01...10, що містить також слова вигляду 1...10, утворені з роздільників відтинанням їхніх перших бітів. Ці слова утворюють роздільники разом з кінцевими нульовими бітами попередніх кодових слів у стиснутому файлі. Різновид цих кодів з множинними роздільниками (MP-кодів) означено та всебічно досліджено у [8]. Заданій зростаючій послідовності натуральних чисел m_1, m_2, ..., m_t, відповідає мультироздільниковий код $D_{m_1,\ldots,m_t}$, що містить t слів вигляду $1^{m_i}0$, а також всі інші можливі двійкові слова з суфіксами $1^{m_i}0$. При цьому всі послідовності вигляду $1^{m_i}0$, $i=1,\ldots,t$, можуть входити до складу кодових слів тільки як суфікси.

З цього визначення випливає, що роздільниками коду $D_{m_1,\ldots,m_t}$ є послідовності вигляду $1^{m_i}0$. Проте код містить також коротші слова вигляду $1^{m_i}0$, що є важливим, оскільки наявність коротких кодових слів покращує коефіцієнт стиснення даних, які часто трапляються на практиці.

Варто зауважити, що хоча MP-коди не є патерн-кодами, їх можна розглядати як узагальнення певних різновидів патерн-кодів. Зокрема код $D_{k-\infty}$, тобто код з усіма можливими роздільниками, що містять k або більше одиниць, являє собою множину всіх двійкових рядків, де послідовності бітів вигляду 1^i0, $i \geq k$ є суфіксами і тільки суфіксами. Такий код є ізоморфним коду Капочеллі $S(k+1, 01^k)$, який своєю чергою можна розглядати як узагальнення кодів Фібоначчі, що продемонстровано в [10].

Очевидно, будь-який мультироздільниковий код є префіксним, а отже однозначно декодовним. Повноту та універсальність MP-кодів доведено у роботі [8]. Також у ній визначено асимптотичну щільність MP-кодів та наведено результати експериментів зі стиснення природномовних текстів. Порівняно з іншими кодами з роздільниками, MP-коди мають кращий коефіцієнт стиснення (зокрема, більше ніж у 4 рази ближчий до ентропійної межі, ніж коди SCDC). Це можна пояснити насамперед двома факторами. По-перше, найкоротший (m_1+2)-бітовий роздільник $01^{m_1}0$ відіграє ту саму роль, що й (m_1+2)-бітовий роздільник у патерн-коді, однак водночас MP-код містить (m_1+1)-бітове слово $1^{m_1}0$, яке є коротшим за цей роздільник. По-друге, в MP-кодах можна використовувати багато роздільників i, варіюючи їхні довжини, припасовувати код до розподілу частот символів вхідного алфавіту. Зауважимо також, що роздільники вигляду $01^{m_1}0$ забезпечують синхронізовність коду. Зрештою, MP-коди мають просту структуру регулярних мов. Це дає можливість створити ефективні кодувальні, декодувальні та пошукові алгоритми, які оперують цілими байтами коду, за допомогою «квантифікації» таблиць переходів скінчених автоматів, що розпізнають коди біт за бітом. Розроблений у [8] швидкий побайтовий алгоритм декодування MP-кодів працює майже вдвічі швидше порівняно зі швидким методом декодування коду Fib3, запропонованим у [10], проте є повільнішим за швидкі методи декодування байтових кодів.

Головною нерозв'язаною проблемою для MP-кодів є проблема ефективної індексації словника. Усі кодувальні / декодувальні алгоритми, які ми розглядаємо, можна описати такою схемою. Під час кодування застосовують

відображення *слово тексту* $\rightarrow$ *кодове слово*, де слова тексту відсортовані за спаданням їхніх частот, а кодові слова — за зростанням довжин. Декодувальний процес є зворотним: потрібно сконструювати відображення з множини кодових слів на множину слів тексту. Щоб прискорити декодування, для зберігання слів тексту слід використати структуру даних зі швидким доступом до її елементів. Найефективнішою структурою з цього погляду є масив з цілочисловими індексами. Однак тоді слід побудувати бієктивне відображення з множини невід'ємних цілих чисел на множину кодових слів. Для кодів Фібоначчі це відображення можна ефективно реалізувати, використовуючи специфічні властивості чисел Фібоначчі [7]. У байтових кодах SCDC та RPBC для цього використовують арифметичні властивості кодових слів та їхніх частин.

Кодувальне відображення ψ для МР-кодів описано у [8]. Проте воно має суттєвий недолік: кодові слова $\psi(0)$, $\psi(1)$, ... не відсортовано за зростанням їхніх довжин. Тому кодові слова $\psi(0)$, ... , $\psi(n-1)$ не утворюють множини з n найкоротших кодових слів. І якщо для досягнення найвищого коефіцієнту стиснення використовувати n найкоротших кодових слів $\psi(i_1)$, ... , $\psi(i_n)$, то i_n може бути значно більшим за n. Отже, масив з ненульовими елементами з індексами i_1 , ... , i_n може бути дуже розрідженим і непрактичним для кодування навіть відносно коротких текстів, що містять 2000–3000 різних слів.

Хоча деякі обхідні шляхи розв'язання цієї проблеми описано в [15], найкращим способом є конструювання монотонного кодувального відображення ψ (тобто такого, що $i > j \Rightarrow |\psi(i)| \geq |\psi(j)|$). Його можна задати масивом значень $\psi(i)$, однак для швидкого декодування потрібно вказати також спосіб ефективного обчислення зворотного відображення ψ^{-1}. Основною проблемою тут є опрацювання довгих кодових слів. Для реалістичних обсягів даних їхня довжина може сягати 32–34 біт, а отже, якщо ψ^{-1} подавати у вигляді масиву, цей масив міститиме $2^{32} - 2^{34}$ елементів, що перевищує межу практичної застосовності. Тому кодові слова слід декодувати частинами, бажано байт за байтом. Для МР-кодів побудова такого відображення ψ^{-1}, що опрацьовує код зліва направо, видається проблематичною, адже опрацювання частини кодового слова не дає суттєвої інформації про позицію всього слова у словнику. Ймовірно, таке відображення можна було б побудувати для непрефіксного коду, у якому коротші кодові слова є префіксами довших. Тоді декодувати слово uv, частина якого u є також кодовим словом, можна було б так: знаходимо індекс u у впорядкованому за зростанням довжин масиві усіх кодових слів, а потім збільшуємо цей індекс на певну величину, що залежить лише від v. Зокрема, цей підхід дає можливість виконати побайтове декодування довгих кодових слів.

Зауважимо, що якщо записати біти кодових слів МР-коду у зворотному порядку, можна отримати непрефіксний, але однозначно декодовний код. Справді, закодований файл можна буде декодувати справа наліво як МР-код. Його також можна буде однозначно декодувати зліва направо, оскільки у ньому всі роздільники залишаються такими самими, як і в початковому МР-коді. Однак вони є префіксами, а не суфіксами кодових слів. Такі «реверсні» мультироздільникові (РМР) коди і є головним об'єктом цього дослідження. Найважливішою властивістю модифікованих у такий спосіб мультироздільникових кодів є наявність монотонного кодувального відображення з множини невід'ємних чисел на множину кодових слів, для якого легко сконструювати зворотне декодувальне відображення, що опрацьовує біти кодових слів зліва направо.

У разі використання алфавітів, що складаються зі слів, разом із закодованим файлом потрібно зберігати великий словник, і це суттєво знижує коефіцієнт стиснення. Тому запропоновано здійснити кілька перетворень тексту та словника, які можна застосувати на етапі передоброблення тексту, підвищивши підсумковий коефіцієнт стиснення. Подальше РМР-кодування теж можна розглядати як другий етап передоброблення, виконуючи основне стиснення за допомогою універсальних архіваторів. Експерименти свідчать, що завдяки застосуванню цього підходу зростає ефективність стиснення природномовних текстів з використанням відомих потужних архіваторів.

ОЗНАЧЕННЯ МНОЖИНИ КОДОВИХ СЛІВ

Нехай $M = \{m_1, \ldots, m_t\}$ — зростаюча послідовність натуральних чисел.

Означення 1. Реверсний мультироздільниковий код $R_{m_1,\ldots,m_t}$ містить усі слова вигляду 01^{m_i}, $i = 1, \ldots, t$, а також усі інші слова, що задовольняють такі вимоги:

(i) слово починається з префікса 01^{m_i} для деякого $m_i \in M$;

(ii) для будь-якого $m_i \in M$ слово не закінчується послідовністю 01^{m_i};

(iii) для будь-якого $m_i \in M$ слово може містити послідовність $01^{m_i}0$ тільки як префікс.

Із цього означення випливає, що роздільники коду $R_{m_1,\ldots,m_t}$ є бітовими послідовностями вигляду 01...10. Однак код містить також коротші слова вигляду 01^{m_i}, які утворюють роздільники разом із першим нулем наступного кодового слова.

Для РМР-кодів існує монотонне кодувальне відображення з множини цілих невід'ємних чисел на множину кодових слів. Вперше його означено в [16]. Тут також опишемо його як базу ефективних методів декодування. Нехай $K = \{k_1, \ldots, k_q\}$ — зростаюча послідовність всіх цілих чисел на відрізку $[0, m_t + 1]$, які не належать множині M. Наприклад, $K = \{0, 1, 3, 6\}$ для коду $R_{2,4,5}$. Зауважимо, що кожне кодове слово коду $R_{m_1,\ldots,m_t}$ має таку структуру: воно починається з префікса 01^{m_i} для певного $m_i \in M$, за яким слідують групи бітів вигляду 01^s, де $s \in K$ або $s > k_q$. Зворотне твердження також справедливе: будь-яка бітова послідовність описаної структури є кодовим словом.

Отже, у коді $R_{m_1,\ldots,m_t}$ будь-яке кодове слово довжиною L біт можна сконструювати в один і тільки один із таких способів:

- воно утворене зі слова довжиною $L - s - 1$ дописуванням бітової послідовності 01^s, $s \in K$;
- воно утворене зі слова довжиною $L - 1$ із суфіксом 01^r, $r > m_t$, дописуванням біта '1';
- це слово вигляду 01^{m_i}, $i = 1, \ldots, t$.

Ґрунтуючись на цих фактах, сформулюємо принцип конструювання впорядкованої множини кодових слів довжиною L, якщо множини коротших кодових слів вже побудовано. Він містить такі правила:

(а) Змінюючи i від 1 до q, повторюємо множини кодових слів довжиною $L - k_i - 1$, додаючи до них послідовності вигляду 01^k, $k_i \in K$.

(б) Повторюємо множину кодових слів довжиною $L - 1$ із суфіксом 01^r, $r > m_t$, додаючи до них біт '1'.

(в) Якщо $L = m_i + 1$, $i \in \{1, \ldots, t\}$ додаємо слово 01^{m_i} до множини кодових слів.

Застосовуючи крок за кроком цей підхід для формування множин кодових слів довжиною m_1+1, m_1+2, ..., отримуємо множину всіх слів коду $R_{m_1,\ldots,m_t}$, впорядковану за зростанням їхніх довжин. Наприклад, множину слів коду $R_{1,4,5}$ довжиною не більше 8 зображено на рис. 1. Також на рис.1 показано, як слова довжиною 8 утворено зі слів довжин 7, 6, і 4 додаванням бітових послідовностей 0, 01 і 0111 відповідно (правило (а)). Три кодових слова, позначені сірим кольором, сконструйовано за правилом (в). Найкоротше кодове слово, утворене із застосуванням правила (б), має довжину 11 біт; його зображено в лівому нижньому куті.

Будь-який реверсний мультироздільниковий код $R_{m_1,\ldots,m_t}$ містить таку саму кількість кодових слів певної довжини, як і «прямий» мультироздільниковий код $D_{m_1,\ldots,m_t}$. Тому реверсні мультироздільникові коди успадковують всі властивості МР-кодів, зокрема повноту та універсальність, а також мають таку саму асимптотичну щільність. У [8] для МР-кодів ці властивості доведено, а також обчислено асимптотичні щільності та кількості коротких кодових слів.

Далі розглянемо «нескінченні» версії РМР-кодів, а саме $R_{2-\infty}$ і $R_{2,4-\infty}$, оскільки вони мають найкращі коефіцієнти стиснення на розглядуваних нами даних. У них використано всі роздільники з 2, 3, ... або 2, 4, 5, ... одиницями відповідно. Зазначимо, що на практиці достатньо обмежити довжини роздільників певним відносно великим числом, наприклад 27, оскільки відмінність коефіцієнтів стиснення кодів R_{x-27} і R_{x-28} є незначною.

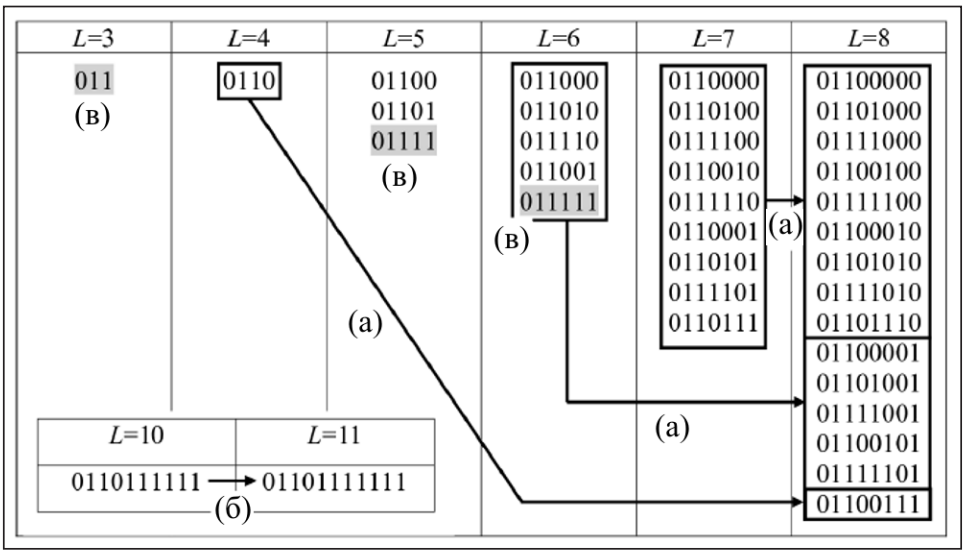

Рис. 1. Конструювання кодових слів коду $R_{2,4,5}$

КОДУВАННЯ ТА ДЕКОДУВАННЯ

Щойно множину кодових слів c_1, c_2, ..., впорядкованих за зростанням довжин, побудовано, кодування тексту стає тривіальним. Залишається тільки впорядкувати слова тексту в порядку спадання частот і поставити у відповідність кожному слову w_i у тексті кодове слово c_i.

Декодування — значно складніший процес. Фактично він полягає у розпізнанні в закодованому файлі кодового слова c_i і визначенні його індексу i у множині всіх кодових слів, побудованій за описаним у попередньому розділі

принципом. Нагадаємо, що коди $R_{m_1,\ldots,m_t}$, як і $R_{m_1,\ldots,m_t-\infty}$, є регулярними мовами, що розпізнаються скінченними автоматами, які опрацьовують кодові слова зліва направо біт за бітом. Ці автомати можна застосувати для обчислення індексів кодових слів у їхній впорядкованій множині.

Декодувальний автомат для коду $R_{2-\infty}$ зображено на рис. 2. Він опрацьовує потік кодових слів, починаючи роботу у стані, позначеному зафарбованим кружком, і завершуючи її у стані 1 після опрацювання певного роздільника. Отже, для завершення декодування до закодованого файлу потрібно дописати будь-який роздільник, наприклад 0110. Опрацьовуючи кодові слова біт за бітом, автомат обчислює значення двох змінних: L — довжини кодового слова і p — власне результату, індексу кодового слова. На кожному переході вхідний біт записаний до вертикальної риски, а операції з L та p — після неї; порядок операцій є важливим.

Пояснимо дію скінченного автомата. Будь-яке слово коду $R_{2-\infty}$ можна подати як конкатенацію бітових послідовностей вигляду 01^t, де $t \geq 0$. Опрацювавши таку послідовність і '0' після неї, автомат переходить у стан 0. Якщо $t \geq 2$, послідовність 01^t може бути лише префіксом кодового слова. Отже, після оброблення останньої '1' у послідовності 011, автомат переходить зі стану 1 у стан 2, виводить індекс p попереднього кодового слова, та ініціалізує L новим значенням $3 = |011|$. На всіх інших переходах, крім переходу з початкового стану, L збільшується на 1. Коли автомат переходить зі стану 2 до стану 0, це означає, що префікс $01^{L-1}0$ кодового слова вже опрацьовано і p має бути ініціалізовано індексом кодового слова 01^{L-1} у списку кодових слів, пронумерованих з 0. Як видно з рис. 1, цей індекс дорівнює $n_L - 1$, де n_L — кількість кодових слів, не довших за L бітів.

Після опрацювання бітової послідовності 00 або 010 автомат виконує перехід до стану 0 зі станів 0 або 1 відповідно та додає до індексу p значення n_{L_0} або n_{L_1} відповідно. Значення n_{L_i} дорівнює зсуву кодового слова у впорядкованій послідовності всіх кодових слів внаслідок опрацювання бітової послідовності 01^i, $0 \leq i \leq 1$, якщо отримане в результаті слово має довжину L. Зауважимо, що описаний вище принцип побудови послідовності кодових слів гарантує, що додавання однакової групи бітів 01 ... 1 до будь-якого кодового слова заданої довжини зсуває це слово на однакову кількість позицій. Значення n_{L_0} та n_L можна визначити на стадії передобчислень, разом із побудовою множини кодових слів. Для певних експериментальних даних і методів стиснення (див. розд. «Обчислювальні експерименти»), коди $R_{3-\infty}$ та $R_{2,4-\infty}$ також можуть забезпечити найвищий коефіцієнт стиснення. Декодувальні автомати для цих кодів наведено в [17]. Вони подібні до автомата для коду $R_{2-\infty}$, хоча й мають більше станів та переходів.

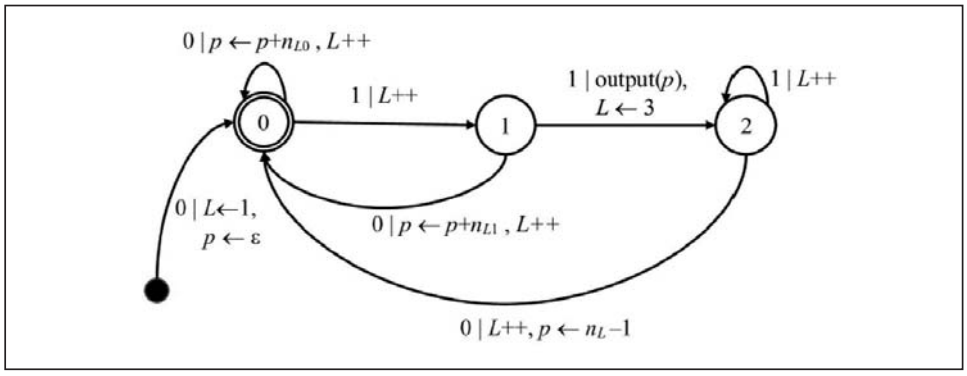

Рис. 2. Декодувальний автомат для коду $R_{2-\infty}$

ШВИДКЕ ПОБАЙТОВЕ ДЕКОДУВАННЯ

Як уже зазначено, будь-який реверсний мультироздільниковий код можна розглядати як регулярну мову, яку розпізнає скінченний автомат, що опрацьовує закодовані дані біт за бітом, тобто досить повільно. Головна ідея швидкого декодувального алгоритму полягає у «квантифікації» декодувального автомата так, щоб він зчитував цілі байти коду і виводив числа результату.

Розглянемо насамперед швидкий декодувальний алгоритм для РМР-кодів за припущення, що словник містить не більше 2^{16} елементів (алгоритм 1). Уведемо певні позначення. Припустимо, що оброблення деякого байта коду завершено. Вказівник *ptr* поєднує в собі поточний стан автомата a і кількість вже декодованих бітів останнього кодового слова l. Його значення можна обчислити так: $ptr = a \cdot l_{\max} + l$, де $l_{\max}$ — максимально можлива бітова довжина кодового слова. Якщо *ptr* помножити на 256 і додати поточний байт тексту, отримаємо індекс x пошукових таблиць (рядок 3 алгоритму 1). Використано такі пошукові таблиці:

- *Pointers*$[x]$ — вказівник на декодувальні таблиці для наступного байта тексту;
- *Numbers*$[x]$ — 64-бітне число, що складається з 4 16-бітних чисел, які отримують після декодування поточного байта;
- $c[x]$ — кількість кодових слів, декодування яких завершують під час оброблення поточного байта.

Отже, чотири послідовних декодованих числа можна вивести через одне присвоєння 64-бітного значення. З іншого боку, легко показати, що не більше чотирьох кодових слів можна повністю або частково декодувати під час опрацювання одного байта коду з найкоротшим роздільником 011. Враховуючи ці два факти, отримуємо алгоритм 1 декодування текстів з коротким словником.

Алгоритм 1. Побайтовий алгоритм декодування РМР-коду для коротких текстів

Вхід: бітовий потік РМР-коду, що складається з байтів *Code*$[1\ldots n]$.

Вихід: масив чисел *Out*.

1. $ptr \leftarrow 0$
2. For $i \leftarrow 1$ to n do
3. $\quad x \leftarrow 256 \cdot ptr + Code[i]$
4. $\quad ptr \leftarrow Pointers[x]$
5. $\quad Out\,[k,\ldots,k+3] \leftarrow Numbers[x] + tr$
6. $\quad k \leftarrow k + c[x]$
7. $\quad tr \leftarrow Out[k]$
8. End.

На кожній ітерації циклу в рядках 2–8 опрацьовується один байт коду. У рядку 3 за байтом коду *Code*$[i]$ та попереднім значенням вказівника *ptr* обчислюємо індекс x, а в рядку 4 — нове значення цього вказівника. У рядку 5 виводимо чотири декодованих числа, навіть якщо фактична кількість чисел, отримуваних у результаті опрацювання байта, буде меншою. Ця кількість міститься в масиві c, і в рядку 6 поточна позиція в масиві виведення *Out* збільшується на $c[x]$. Це означає, що певні елементи масиву *Out* можуть бути перезаписані на подальших ітераціях декодувального циклу.

Побайтове декодування проілюстровано рис. 3. Під час декодування i-го байта останнє кодове слово в його складі може бути декодовано частково (як, наприклад, слово $k+2$ на рис. 3). Якщо перше декодоване слово байта

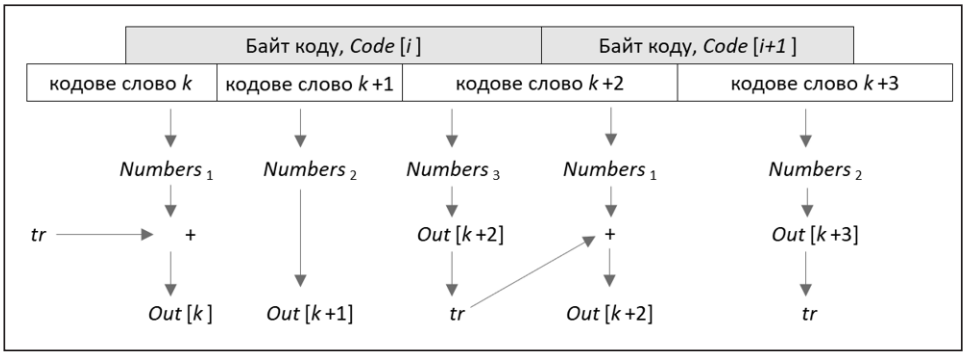

Рис. 3. Декодування двох байтів PMP-коду

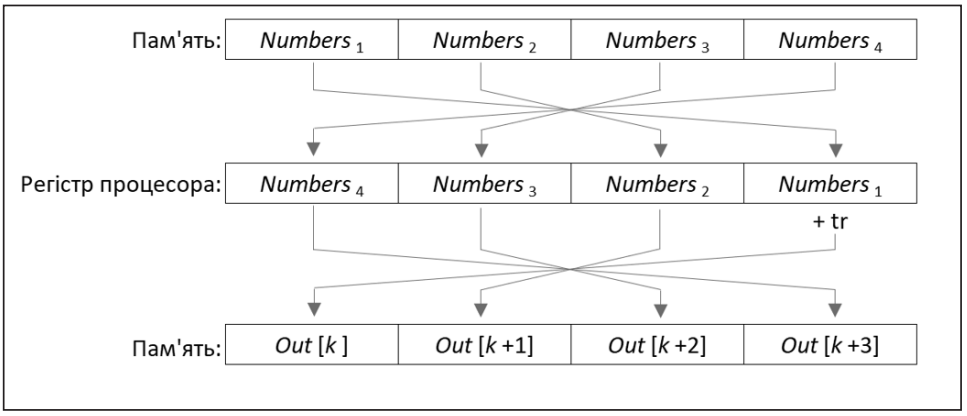

Рис. 4. Виконання рядка 5 алгоритму 1 на машині зі зворотним порядком завантаження байтів

зберігається в елементі $Out[k]$, згадане останнє слово зберігатиметься в елементі $Out[k + c[x]]$. Результат його часткового декодування присвоюється змінній *tr* у рядку 7. На машині зі зворотним порядком завантаження байтів (little endian machine) це значення додається до першого декодованого значення наступної ітерації декодувального циклу в рядку 5, оскільки порядок байтів у значенні, що завантажується як з пам'яті в регістр процесора, так і навпаки, є зворотним (рис. 4). Зауважимо, що на машині з прямим порядком завантаження байтів (big endian machine) в рядку 5 потрібна додаткова операція зсуву значення *tr* на 48 біт вліво.

Зауважимо також, що на відміну від швидкого алгоритму декодування PMP-кодів, наведеного в [4], автори цієї статті не аналізують кількість кодових слів, що декодуються на поточній ітерації циклу. Це зроблено для того, щоб уникнути використання непередбачуваного умовного оператора (тобто умова якого може бути як істинною, так і хибною з високою ймовірністю), адже виконання таких операторів є однією з головних причин уповільнення програм на сучасних процесорах.

Загальна ідея описаного алгоритму нагадує швидкий алгоритм декодування BCT-коду, наведений у [18]. Однак індекс пошукових таблиць BCT-коду не залежить від довжини вже декодованої частини кодового слова. Отже, ці таблиці мають значно менший розмір і тому BCT-код можна декодувати набагато швидше, хоча він забезпечує гірший коефіцієнт стиснення.

Якщо словник містить більше 2^{16} слів, у масиві *Out* зберігаються 32-розрядні числа і рядок 5 алгоритму 1 слід замінити такими двома рядками псевдокоду:

$$Out[k, k+1] \leftarrow Numbers[x] + tr$$
$$Out[k+2, k+3] \leftarrow ExtraNumbers[x]. \quad (1)$$

Тут *ExtraNumbers* — це спеціальний масив з 64-бітних значень, що містить третє і четверте числа, які можна отримати під час декодування поточного байта. Це дає змогу опрацьовувати тексти зі словником обсягом до 2^{32} елементів, що охоплює всі можливі на практиці випадки.

Оцінимо ємнісну складність алгоритму 1. Врахуємо, що декодувальний автомат $R_{2-\infty}$ має 3 стани. Довжина кодових слів у словнику, що містить не більше 2^{16} елементів, не перевищує 24 біт. Це дає $3 \cdot 24 = 72$ можливих значень вказівника *ptr* і 72·256=18432 можливих значень x, обчислюваних у рядку 3. Кожен елемент масиву *Numbers* займає 8 байтів, масивів *Pointers* і *c* — по 1 байту. Отже, загальний обсяг пошукових таблиць становить 184 KB, що вміщується в кеш-пам'ять рівня L2 на більшості комп'ютерів. Для довгих текстів обсяг пошукових таблиць буде менш ніж удвічі більшим, що не перевищує типового обсягу кешу рівня L3.

ПЕРЕДОБРОБЛЕННЯ ТЕКСТУ

Як правило, у практичному стисненні природномовних текстів коефіцієнт стиснення підвищують завдяки передобробленню даних. Найвідоміші засоби передоброблення, як-от: WRT [19], StarNT [20], LIPT [21] працюють на символьних алфавітах і використовують такі регулярності, як стиснення q-грам, інтерпретація великих літер як малих, окреме стиснення типових суфіксів і коренів тощо. Автори цієї статті сконструювали власний засіб передоброблення для алфавітів, що складаються зі слів. Він розрахований на зберігання словника разом із закодованим текстом і посткомпресію документа за допомогою кодів змінної довжини. Для подальшого стиснення тексту можна застосувати такі потужні архіватори, як 7zip. Розглянемо перетворення, що виконуються засобом передоброблення.

Згортання низькочастотних слів. Зі словами, що трапляються в тексті один раз, пов'язана певна надлишковість. Фактично, кожне слово зберігають двічі: перший раз у словнику, а другий — як кодове слово в закодованому тексті. Можна було б уникнути збереження кодового слова, замінивши його в тексті оригінальним словом, але тоді виникають проблеми з позначенням його початку та кінця. Натомість запропоновано таку схему:

1. Сортуємо частину словника, що складається зі слів частоти 1, у порядку їхньої появи в тексті.

2. Кодуємо всі слова частоти 1 у тексті однаковим коротким кодовим словом, скажімо $code_1$.

3. Під час декодування замінюємо кожне слово $code_1$ черговим елементом із цієї частини словника.

У результаті для збереження кожного слова частоти 1 використано $|word|+|code_1|$ бітів замість $|word|+|code(word)|$ бітів. Частота кодового слова $code_1$ дорівнює загальній кількості слів частоти 1, що визначає його позицію ближче до початку словника, ніж у будь-якого слова з частотою 1. Отже, $|code_1| \le |code(word)|$, і описана методика заощаджує простір.

Хоча схожа методика описана у [22], запропоновано узагальнити її, охопивши слова з частотою більше 1. Під час кодування, коли вперше трапляється слово з частотою $k > 1$, замінюємо його словом $code_k$, а всі наступні входження цього слова кодуємо відповідним йому початковим кодовим словом.

Капіталізація. Це перетворення враховує регулярність природномовного тексту, пов'язану з тим, що майже завжди слова, що стоять після крапки, (і практично лише вони) починаються з великої літери. По-перше, до словника

додають спеціальний символ — індикатор аномалії в тексті. Потім опрацьовують усі пари послідовних слів у тексті. Якщо перше слово з пари закінчується символом «.», ймовірно, це кінець речення. У цьому випадку перевіряють, чи починається друге слово з великої літери. Якщо так, то порівнюють частоту другого слова, записаного з малої літери, з частотою його початкової версії. Якщо частота версії з малої літери вища, слово замінюють цією версією. В іншому разі жодних дій не вчиняють.

Під час декодування було б достатньо записати всі слова після «.» з великої літери, якби не траплялося аномальних випадків, коли слово після крапки в оригінальному тексті починається з малої літери. Такі випадки позначають спеціальним символом аномалії, який розміщують після слова з крапкою. Він сигналізує декодеру, що капіталізувати наступне слово не потрібно.

Нарешті, можливі випадки, коли слово після «.» починається не з літери взагалі. Тоді кодер і декодер не вчиняють жодних спеціальних дій.

Зауважимо, що це перетворення може записати з малої літери всі входження певного слова, яке починається з великої літери. Тоді словник міститиме слово з нульовою частотою, яке згодом можна буде з нього видалити.

Локальний словник. Ця техніка є спрощенням відомого методу прогнозування ймовірностей символів у контексті PPM [23], який у разі застосування до алфавітів зі слів є надто ресурсовитратним і недостатньо ефективним. Розглянемо всі слова тексту, які йдуть після певного вибраного слова. Цю підмножину тексту позначимо S і назвемо локальним словником, а вибране слово — ключовим. Розподіл частот слів у локальному словнику відрізняється від глобального. Загальну текстову ентропію можна зменшити, вибираючи серед високочастотних слів тексту ключові слова і перекодовуючи локальні словники.

Виходячи з цього, сформуємо L-елементну множину S_L, що містить коди елементів S з найвищими локальними частотами. Після ключового слова елементи S_L кодують кодовими словами $code(0), \ldots, code(L-1)$. Усі слова, які залишилися в $S \setminus S_L$, кодують так: якщо позиція слова в глобальному словнику є меншою L, перед відповідним кодовим словом вставляють спеціальний код аномалії, в іншому разі жодних дій не вчиняють. Код аномалії слід вибирати залежно від частоти аномалії, яка може бути більшою або меншою за L.

Декодування є достатньо очевидним. Якщо після ключового слова йде код аномалії, його просто видаляють і декодування продовжують з глобальним словником. Інакше припустимо, що після ключового слова записано код числа x. Якщо $x < L$, то слово тексту беремо з локального словника, інакше — з глобального.

Можна використовувати кілька локальних словників одночасно, по одному для кожного ключового слова. Кожен локальний словник разом із ключовим словом, параметром L і кодом аномалії мають зберігатися як частина закодованого файлу. Звичайно, що вищою є частота ключового слова, то ефективнішим є відповідний локальний словник. Для заданого локального словника існує певне оптимальне значення L, перевищення якого призводить до надмірної кількості аномальних кодів. Отже, кількість локальних словників та значення L для кожного словника мають бути збалансовані.

Використання q-грам. Хоча для алфавіту зі слів різниця між ентропією Hq та ентропією H0 значно менша, ніж для символьного алфавіту, окреме кодування деяких частих q-грам може зменшити загальну бітову довжину закодованого файлу.

Припустимо, що словник поділено на частини, які відповідають кодовим словам однакової довжини. Для PMP-кодів кодові слова в кожній частині

будуть на 1 біт довші за слова в попередній частині. Розглянемо q-граму слів і позначимо її частоту в тексті як l. Цій частоті відповідає певне кодове слово c. Отже, можна замінити кожну таку q-граму слів кодовим словом c.

Нехай $|c|$ — довжина кодового слова c, а C — сумарна довжина кодових слів, які замінюються словом c. Якщо словник вже побудовано, вставлення нового кодового слова зсуває решту словника. Нехай $q_1,\ldots, q_m$ — це частоти останніх слів у частинах словника, що містять кодові слова $|c|, |c|+1,\ldots, |c|+m$ бітів завдовжки. Після вставлення кодового слова c довжини цих останніх кодових слів збільшаться на 1, що додає $q_1+\ldots+q_m$ бітів до закодованого тексту. Тому виграш від кодування q-грами за допомогою одного кодового слова становить $l(C-|c|)-q_1-\ldots-q_m-a$, де a — розмір заархівованої інформації про кодування q-грами. Якщо це значення додатне, кодувати q-граму доцільно.

ОБЧИСЛЮВАЛЬНІ ЕКСПЕРИМЕНТИ

Виміряно коефіцієнти стиснення трьох англомовних текстів різного розміру різними кодами змінної довжини (для кожного з них вказано параметри оптимального коду SCDC):

- малий: Біблія англійською мовою, 3,83 МБ, 790 018 слів, 14 087 різних слів, SCDC(224,32).
- середній: статті, вибрані випадковим чином з Вікіпедії (116 МБ, 19 507 783 слів, 288 179 різних слів), SCDC(175,81).
- великий: перша половина найбільшого файлу з корпусу Pizza&Chili, (512 МБ, 92 424 896 слів, 1 686 371 різних слів), SCDC(164,92).

Результати наведено в табл. 1 як середню кількість бітів, що припадають на одне слово закодованого тексту. У відсотках також вказано перевищення рівня ентропії. Найкращі результати для кожного тексту виділено жирним.

Дані, наведені в табл. 1, свідчать про те, що з-поміж усіх досліджуваних кодів змінної довжини PMP-коди забезпечують найкращий коефіцієнт стиснення англомовних текстів різного розміру.

Автори перевірили на практиці застосування описаної техніки передоброблення тексту разом із PMP-кодами як засобу підвищення коефіцієнта стиснення популярних архіваторів. Експерименти проведено на тексті обсягом 1ГБ з корпусу Pizza&Chilie [24]. Його точний обсяг становить 1 073 741 824 байт, 189 528 100 слів, 2 523 827 різних слів. Ентропія H0 на алфавіті зі слів для цього файлу становить 273 284 721 байт, або 13,535 біт на 1 слово.

Результати стиснення з передобробленням з подальшим архівуванням та без нього наведено в табл. 2 у байтах. Застосовано архіватори 7z (64-бітну версію 16.02), RAR і gzip на максимальному рівні стиснення (рівень 9) до початкового і закодованого PMP-кодами текстів з усіма описаними вище перетвореннями. Словники та всю іншу метаінформацію включено до складу файлів.

Таблиця 1. Рівень стиснення тексту кодами змінної довжини

Текст/ Код	Ентропія (біт/слово)	Fib2	Fib3	SCDC	RPBC	$R_{2-\infty}$	$R_{2,4-\infty}$
Малий	8.88	10.341 (16.5%)	9.491 (6.9%)	10.373 (16.8%)	10.354 (16.6%)	**9.082** (2.3%)	9.2 (3.6%)
Середній	11.078	11.983 (8.2%)	11.564 (4.4%)	12.869 (16.2%)	12.685 (14.5%)	11.598 (4.7%)	**11.393** (2.8%)
Великий	11.548	12.547 (7.7%)	11.902 (3.1%)	13.063 (13.1%)	13.048 (13%)	12.014 (4%)	**11.77** (1.9%)

Таблиця 2. Стиснення з передобробленням, PMP-кодуванням і подальшим архівуванням

Архіватор	Лише архівування	$R_{2-\infty}$	$R_{2,4-\infty}$	$R_{3-\infty}$
–	–	295 561 517	289 577 183	292 954 265
7z	258 428 183	258 705 282	257 252 378	256 751 472
RAR	290 584 311	264 645 314	261 948 637	262 563 336
gzip	405 714 638	277 592 303	273 990 536	274 736 358

Як видно, попереднє кодування за допомогою PMP-кодів суттєво підвищує коефіцієнт стиснення архіваторів RAR або gzip, більше ніж на 10% і майже на 50% відповідно. Архіватор 7z, що ґрунтується на алгоритмі LZMA, стискає текст значно краще за RAR або gzip і є одним із найбільш потужних сучасних архіваторів. Однак навіть у цьому випадку PMP-коди залишають простір для вдосконалень. Наприклад, файл, отриманий у результаті 7z-архівування тексту, закодованого кодом $R_{3-\infty}$, є на 0.7% меншим за файл, отриманий у результаті 7z-архівування тексту без застосування PMP-кодів.

Зауважимо також, що незаархівовані, але закодовані PMP-кодами файли на 32–34% менші за файли, заархівовані gzip. Якщо врахувати можливість швидкого декодування та пошуку у стиснутому файлі, можна стверджувати, що PMP-коди можна розглядати як пріоритетний формат для зберігання текстових корпусів порівняно з gzip.

Швидкість декодування виміряно на комп'ютері з процесором AMD Athlon 3000G, 32 КБ кеш-пам'яті рівня L1, 512 КБ кеш-пам'яті рівня L2, 4 МБ кеш-пам'яті рівня L3, 16 ГБ основної оперативної пам'яті, 64-бітною операційною системою Windows 10. Результати наведено в табл. 3. Найкращі результати для кожного тексту виділено жирним.

Для порівняння вибрано інші відомі коди, що дають можливість виконувати пошук у стиснутому файлі та/або швидке декодування: коди Фібоначчі Fib2 і Fib3, а також байтові (s, c)-щільні коди. Коди SCDC параметризовані, і для них вибрано значення параметра, що максимізує коефіцієнт стиснення.

Результати свідчать про те, що PMP-коди декодуються майже так само швидко, як і SCDC, і в рази швидше, ніж код Фібоначчі Fib3. Декодувальний алгоритм для коротких текстів (алгоритм 1) працює навіть швидше за декодування SCDC. Для довших текстів алгоритм 1 можна застосувати тільки зі зміною (1) і такий варіант є на 4–11% повільнішим за SCDC-декодування. Також, як видно, код $R_{2-\infty}$ можна декодувати на 4–7% швидше, ніж $R_{2,4-\infty}$. Це можна пояснити більшим розміром пошукових таблиць для коду $R_{2,4-\infty}$ (декодувальний автомат для $R_{2,4-\infty}$ містить п'ять станів, а для $R_{2-\infty}$ — три).

Також проведено експерименти з пошуку послідовності слів у закодованому тексті. Як SCDC, так і PMP-коди дають змогу виконувати швидкий пошук типу Боєра–Мура в закодованому файлі без його декодування. Зауважимо, що пошук в SCDC-файлі можна виконати на рівні байтів, в той час як у PMP-файлі потрібно аналізувати патерни бітів. Це означає, що пошук патернів в SCDC-файлах загалом має бути швидшим. Однак найшвидше з відомих на сьогодні сімейство методів пошуку патерну в потоці бітів [25] виконує пошук на рівні байтів і лише коли підрядок-кандидат знайдено, аналізуються окремі біти. Це дає можливість виконувати пошук бітового патерну зі швидкістю, зіставною зі швидкістю пошуку байтового патерну.

Таблиця 3. Емпіричне порівняння часу декодування (мс)

Текст	Fib3	SCDC	$R_{2-\infty}$ – короткий	$R_{2-\infty}$ – довгий	$R_{2,4-\infty}$ – довгий
Малий	7.38	2.73	**2.69**	3.08	-
Середній	240.3	**96.6**	-	100.6	108
Великий	984	**479**	-	507	525

Таблиця 4. Емпіричне порівняння часу пошуку (мс)

Код / Довжина патерну (слів)	2	4	8	16	32	64	128	256	512	1024
SCDC	21.38	17.46	17.53	17.02	11.87	6.23	3.78	2.68	2.11	1.61
$R_{2,4-\infty}$	120.91	37.54	20.6	16.73	15.88	15.3	8.81	4.74	2.5	1.67

Для пошуку патерну у зазначеному вище великому тексті застосовано один із таких методів, описаних у [25]. Для порівняння текст закодовано кодами SCDC та $R_{2,4-\infty}$. Для пошуку в SCDC-коді застосовано алгоритм RZkBit-wn, а в PMP-коді — алгоритм RZkBit-wn, де k — кількість значущих бітів у масці, а n — кількість ковзних вікон пошуку. Для пошуку всіх входжень закодованих послідовностей з 2–1024 слів, випадково взятих з тексту, перевірено різні значення параметрів k та n. Для кожної довжини патерну вибрано значення (k, n), які забезпечують найкращий середній час.

Середній час, виміряний за 1000 запусків кожного алгоритму, наведено в табл. 4. Як видно, для дуже коротких патернів (довжиною 2 слова) пошук патерну в SCDC-коді у 6 разів швидший. Це пояснюється тим, що патерни довжиною менше 16 біт неможливо шукати на рівні байтів. Для довших патернів певних довжин (4, 64 або 128 слів) пошук в SCDC більш ніж удвічі швидший. Проте є певні практично цікаві довжини патернів, для яких пошук в PMP-коді виконується майже з такою самою швидкістю, як і пошук в SCDC-коді чи навіть трохи швидше (8 або 16 слів). Коли патерни стають довшими, різниця між продуктивністю бітових і байтових пошукових методів нівелюється. Про це свідчить час пошуку патернів довжиною 512 і 1024 слів.

ВИСНОВКИ

Основним об'єктом дослідження є реверсні мультироздільникові стискальні коди. Їх можна використовувати як у статистичному стисканні природномовних текстів з алфавітом зі слів, так і для компактного подання необмежених послідовностей цілих чисел. Визначено монотонне бієктивне відображення з множини невід'ємних цілих чисел на множину кодових слів PMP-коду. Це відображення дає можливість реалізувати принцип «декодування за частинами», а отже й побудувати дуже швидкий побайтовий декодувальний алгоритм на основі пошукових таблиць, що є зіставним за часом виконання з методом декодування (s, c)-щільних кодів. Забезпечуючи високий коефіцієнт стиснення, PMP-коди є ефективним інструментом з точки зору співвідношення між рівнем стиснення та часом декодування природномовних текстів. Доповнені спеціальною методикою передоброблення тексту, PMP-коди стають засобом покращення рівня стиснення сучасних архіваторів. Також, будучи кодами з роздільниками, PMP-коди дають можливість виконувати швидкий пошук послідовності слів безпосередньо в закодованому файлі методами типу Боєра–Мура. Експерименти свідчать, що патерни різної довжини можна знаходити у файлах, закодованих SCDC- та PMP-кодами, із порівнянною швидкістю.

СПИСОК ЛІТЕРАТУРИ

1. Duda J., Tahboub K., Gadgil N., Delp E. The use of asymmetric numeral systems as an accurate replacement for Huffman coding. *Proc. 2015 Picture Coding Symposium (PCS 2015)* (31 May – 3 June 2015, Cairns, Australia). Cairns, 2015. P. 65–69. https://doi.org/10.1109/PCS.2015.7170048.
2. Huffman D.A. A method for the construction of minimum-redundancy codes. *Proc. IRE*. 1952. Vol. 40, N. 9. P. 1098–1101. https://doi.org/10.1109/JRPROC.1952.273898.
3. Elias P. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*. 1975. Vol. 21, N 2. P. 194–203. https://doi.org/10.1109/TIT.1975.1055349.
4. Lakshmanan K. On universal codeword sets. *IEEE Transactions on Information Theory*. 1981. Vol. 27, N 5. P. 659–662. https://doi.org/10.1109/TIT.1981.1056387.
5. Guibas L., Odlyzko A. Maximal prefix-synchronized codes. *SIAM Journal on Applied Mathematics*. 1978. Vol. 35, N 2. P. 401–418. URL: http://www.jstor.org/stable/2100678.
6. Capocelli R., de Santis A. Regular universal codeword sets (Corresp.). *IEEE Transactions on Information Theory.* 1986. Vol. 32, N 1. P. 129–133. https://doi.org/10.1109/TIT.1986.1057130.
7. Klein S.T., Ben-Nissan M.K. On the usefulness of Fibonacci compression codes. *The Computer Journal*. 2010. Vol. 53, N 6. P. 701–716. https://doi.org/10.1093/comjnl/bxp046.
8. Anisimov A., Zavadskyi I. Variable-length prefix codes with multiple delimiters. *IEEE Transactions on Information Theory*. 2017. Vol. 63, N 5. P. 2885–2895. https://doi.org/10.1109/TIT.2017.2674670.
9. Apostolico A., Fraenkel A.S. Robust transmission of unbounded strings using Fibonacci representations. *IEEE Transactions on Information Theory*. 1987. Vol. 33, N 2. P. 238–245. https://doi.org/10.1109/TIT.1987.1057284.
10. Capocelli R. Comments and additions to “Robust transmission of unbounded strings using Fibonacci representations”. *IEEE Transactions on Information Theory*. 1989. Vol. 35, N 1. P. 191–193. https://doi.org/10.1109/18.42193.
11. de Moura E.S., Navarro G., Ziviani N., Baeza-Yates R. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems*. 2000. Vol. 18, N 2. P. 113–119. URL: https://dl.acm.org/doi/pdf/10.1145/348751.348754.
12. Brisaboa N., Iglesias E., Navarro G., Parama J. An efficient compression code for text databases. *Proc. 25th European Conference on IR Research (ECIR 2003)* (14–16 April 2003, Pisa, Italy). Pisa, 2003. LNCS. 2003. Vol. 2633. P. 468–481. https://doi.org/10.1007/3-540-36618-0_33.
13. Brisaboa N., Fariña A., Navarro G., Esteller M. (s, c)-dense coding: an optimized compression code for natural language text databases. *Proc. 10th International Symposium on String Processing and Information Retrieval* (8–10 October 2003, Manaus, Brazil). Manaus, 2003. LNCS. 2003. Vol. 2857. P. 122–136. https://doi.org/10.1007/978-3-540-39984-1_10.
14. Culpepper J.S., Moffat A. Enhanced byte codes with restricted prefix properties. *Proc. 12th International Conference on String Processing and Information Retrieval (SPIRE 2005)* (2–4 November 2005, Buenos Aires, Argentina). Buenos Aires, 2005. LNCS. 2005. Vol. 3772. P. 1–12. https://doi.org/10.1007/11575832_1.
15. Zavadskyi I., Anisimov A. A family of data compression codes with multiple delimiters. *Proc. Prague Stringology Conference 2016* (29–31 August 2016, Prague, Czech Republic). Prague, 2016. P. 71–84.
16. Zavadskyi I., Anisimov A. Reverse multi-delimiter compression codes. *Proc. 2020 Data Compression Conference (DCC)* (24-27 March 2020, Snowbird, UT, USA). Snowbird, 2020. P. 173–182. https://doi.org/10.1109/DCC47342.2020.00025.
17. Zavadskyi I., Zavadska V. Reverse multi-delimiter codes in English and Ukrainian natural language text compression. *Proc. VIII International Scientific Conference “Information Technology and Implementation"* (IT&I-2021) (1–3 December 2021, Kyiv, Ukraine). Kyiv, 2021. CEUR Workshop Proceedings. 2022. Vol. 3132. P. 211–219. URL: https://ceur-ws.org/Vol-3132/Short_1.pdf.
18. Zavadskyi I. Binary-coded ternary number representation in natural language text compression. *Proc. 2022 Data Compression Conference (DCC)* (22-25 March 2022, Snowbird, UT, USA). Snowbird, 2022. P. 419–428. https://doi.org/10.1109/DCC52660.2022.00050.

19. Skibinski P., Grabowski S., Deorowicz S. Revisiting dictionary-based compression. *Software: Practice & Experience.* 2005. Vol. 35, N 15. P. 1455–1476. https://doi.org/10.1002/spe.678.

20. Sun W., Zhang N., Mukherjee A. A dictionary-based multi-corpora text compression system. *Proc. 2003 IEEE Data Compression Conference (DCC 2003)* (25-27 March 2003, Snowbird, UT, USA). Snowbird, 2003. p. 448. https://doi.org/10.1109/DCC.2003.1194067.

21. Awan F., Mukherjee A. LIPT: A lossless text transform to improve compression. *Proc. International Conference on Information Technology: Coding and Computing* (02–04 April 2001, Las Vegas, NV, USA). Las Vegas, 2001. P. 452–460. https://doi.org/10.1109/ITCC.2001.918838.

22. Adiego J., Martinez-Prieto M.A., de la Fuente P. High-performance word-codeword mapping algorithm on PPM. *Proc. 2009 Data Compression Conference (DCC 2009)* (16-18 March 2009, Snowbird, UT, USA). Snowbird, 2009. P. 23–32. https://doi.org/10.1109/DCC.2009.40.

23. Cleary J., Witten I. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications.* 1984. Vol. 32, Iss. 4. P. 396–402. https://doi.org/10.1109/TCOM.1984.1096090.

24. PIZZA&CHILI CORPUS — English texts. URL: http://pizzachili.dcc.uchile.cl/texts/nlang/.

25. Zavadskyi I. Fast exact pattern matching by the means of a character bit representation. *SN Computer Science*. 2022. Vol. 3, Iss. 3. Article number 181. P. 1–20. https://doi.org/10.1007/s42979-022-01052-w.

A.V. Anisimov, I.O. Zavadskyi, T.S. Chudakov

NATURAL-LANGUAGE TEXT COMPRESSION USING REVERSE MULTI-DELIMITER CODES

Abstract. We study a class of binary reverse multi-delimiter (RMD) data compression codes in application to natural language text compression. The RMD-codewords start with delimiters, i.e., prefixes of the form 01^m0 that cannot occur in other places of the codeword. The position of the delimiter in an RMD codeword differs from its position in "direct" multi-delimiter (MD) codes, where delimiters are codeword suffixes. RMD and MD codes possess many useful properties, such as unique decodability, completeness, universality, synchronizability, asymptotic densities, and finite automaton acceptability. For RMD-codes, we construct a monotonic mapping from the set of natural numbers to the set of codewords. For original MD-codes, hitherto, this was an open question. The discovered mapping and the byte quantification of a decoding automaton allow us to develop very fast byte-aligned algorithms for decoding and direct Boyer–Moore style search in compressed files. Compared with the known byte (SCDC) and Fibonacci codes, RMD codes demonstrate the best compression ratio on natural language texts (more than four times closer to the entropy bound than that of SCDC). Computer experiments demonstrate that RMD codes can be decoded almost as fast as SCDC and times faster than Fibonacci codes. In natural language text compression, we also practiced the RMD-encoding as a preprocessing tool, which improves the performance of the known modern powerful archivers.

Keywords: compression, archiver, code, multi-delimiter.

Надійшла до редакції 18.08.2023