УДК 519.72

Завадський І. О.

МУЛЬТИРОЗДІЛЬНИКОВІ КОДИ

Означено та досліджено нове сімейство префіксних кодів, що можуть ефективно застосовуватися для стискання текстової інформації. Деякі з кодів цього сімейства мають на 10–30 % ближчий до границі Шеннона коефіцієнт стиснення, на 20 % вищу швидкість та в кілька разів нижчі витрати пам'яті, ніж найкращий із кодів Фібоначчі, які серед відомих класів стискальних кодів є найближчими до розглядуваних нами.

Ключові слова: стиснення інформації, кодування, роздільник, префіксний код, код Фібоначчі, мультироздільниковий код.

Вступ

Стискання інформації є достатньо добре дослідженою галуззю. Загальний підхід до розв'язання задачі стискання є таким: вхідний алфавіт являє собою множину слів, яким зіставляються за певними правилами кодові слова, що в багатьох випадках є коротшими за вхідні. Під час кодування кожне слово вхідного тексту замінюється відповідним йому кодовим словом, а під час декодування відбувається зворотна заміна. Найширший клас методів стискання базується на використанні частотних кодів, основна ідея яких полягає в тому, що словам вхідного алфавіту з більшими частотами зіставляються коротші кодові слова. Якщо кількість слів у вхідному алфавіті дорівнює степеню двійки, найвищий ступінь стискання забезпечують коди Хафмана, в іншому разі найефективнішими серед відомих на сьогодні є арифметичні коди. Ці коди дають дуже щільне наближення до границі Шеннона, яка визначає теоретичну межу можливостей стискальних методів.

Однак, крім коефіцієнта стискання, на практиці важливими є також інші характеристики зазначених методів, зокрема обчислювальна складність кодування, декодування та пошуку в стиснутих файлах, а також стійкість до перешкод, що можуть виникати в каналах зв'язку та на носіях, якими передається чи на яких зберігається стиснута інформація. Коди Хафмана та арифметичні коди не є оптимальними та вдалими з точки зору жодної характеристики, крім коефіцієнта стискання. Зокрема, вони є зовсім нестійкими до перешкод, оскільки помилка навіть в одному біті може призвести до викривлення всієї частини повідомлення, що йде за цим бітом, а також роблять неможливим пошук даних у стиснутих файлах без їхньої деархівації. Ці проблеми вирішують так звані тегові коди Хафмана (ETDC [3], SCDC [4]), які, проте, достатньо суттєво погіршують коефіцієнт стискання.

Компромісним варіантом є коди Фібоначчі, введені в [2] і детально досліджені в [5]. Як і тегові коди, коди Фібоначчі є синхронізованими, тобто кодові слова в них закінчуються спеціальними послідовностями-роздільниками (у кодах Фібоначчі вони мають вигляд 1…1), а отже, помилка в окремому біті з потоку закодованих слів може призвести до спотворення лише того слова, якому цей біт належить, або, якщо біт належить роздільникові, ще й наступного слова. Однак вплив помилки не може поширитися далі наступного роздільника. Найбільш поширеною сферою застосування кодів Фібоначчі є стискання природномовних текстів, під час якого кожному слову тексту зіставляється певне кодове слово. Як показано в [5], код Fib3 дає в 2–3 рази краще наближення до границі Шеннона, ніж коди ETDC та SCDC, однак має гіршу швидкість кодування, декодування та пошуку в стиснутих файлах.

Певним недоліком кодів Фібоначчі є також відсутність властивості негайного розділення. Це означає, що, якщо в закодованому тексті знайдена бітова послідовність, що відповідає певному кодовому слову, вона ще не гарантовано є цим кодовим словом, оскільки коди Фібоначчі не є суфіксними: одні кодові слова можуть бути суфіксами інших (хоча ці коди є префіксними). Більше того, найкоротші кодові слова в кодах Фібоначчі складаються лише з одного роздільника вигляду 1…1 і тому можуть «злипатися» в довгі послідовності з одиниць. Тому для відокремлення одного кодового слова від іншого потрібно перевіряти в загальному випадку необмежену кількість бітів (аж поки не знайдемо початок серії одиниць).

У цій статті ми розглянемо новий клас префіксних кодів – мультироздільникові коди. Таку назву пов'язано з тим, що кодові слова можуть відокремлюватися не одним, а кількома роздільниками вигляду 01…10. Деякі представники цього сімейства кодів у разі застосування до стискання природномовних текстів мають коефіцієнт стискання на 10–30 % ближчий до границі Шеннона, ніж для коду Fib3 (його величина залежить від обсягу алфавіту), на 20 % вищу швидкість і в чотири рази менші витрати пам'яті декодувального алгоритму, при цьому є значно кращими з огляду на негайне розділення: для виявлення в стиснутому файлі точного положення кодового слова достатньо переглянути лише невелику фіксовану кількість бітів, які передують відповідній цьому слову бітовій послідовності.

Означення мультироздільникових кодів

Припустимо, що m_1 або… або m_t – деякі цілі числа, такі, що $0 < m_1 <\ldots< m_t$. Дамо означення мультироздільникового коду $D_{m_1,\ldots,m_t}$.

Означення 1. Код $D_{m_1,\ldots,m_t}$ містить усі слова вигляду 1…10 із m_1 або… або m_t одиницями, а також усі слова, що задовольняють такі умови:

- слово не починається з послідовності 1…10, що містить m_1 або m_2 або… або m_t одиниць;
- слово закінчується послідовністю 01…10, що містить m_1 або m_2 або… або m_t одиниць;
- слово не містить послідовності 01…10 із m_1 або m_2 або… або m_t одиницями ніде, крім кінця.

Роздільниками в такому коді є послідовності вигляду 01…10, що містять m_1 або m_2 або… або m_t одиниць, однак код містить також слова вигляду 1…10 із m_1 або m_2 або… або m_t одиницями, які утворюють роздільник разом із останнім нулем попереднього кодового слова в тексті.

Для прикладу розглянемо такі мультироздільникові коди, як D_2 (для нього t=1, m_1=2) та $D_{2,3}$ (t=2, m_1=2, m_2=3). Наведемо всі кодові слова довжини не більше 7 цих кодів, а також, для порівняння, коду Фібоначчі Fib3 (рис. 1). Як видно, код із двома роздільниками $D_{2,3}$ містить значно більше коротких слів, ніж код Фібоначчі, а також ніж код з одним роздільником D_2. Однак асимптотична щільність цього коду гірша за асимптотичну щільність як коду D_2, так і коду Fib3, а тому він буде ефективним у разі стискання текстів із порівняно невеликим обсягом алфавіту.

Загалом коди із більшою кількістю роздільників мають гіршу асимптотичну щільність,

Код D_2	Код $D_{2,3}$	Код Fib3
110	110	111
0110	0110	0111
00110	1110	00111
10110	00110	10111
000110	10110	000111
010110	01110	010111
100110	000110	100111
0000110	010110	110111
0010110	100110	0000111
0100110	001110	0010111
1000110	101110	0100111
1110110	0000110	1000111
	0010110	1010111
	0100110	1100111
	1000110	0110111
	0001110	
	0101110	

Рис. 1. Кодові слова довжини не більше 7 для кодів D_2, $D_{2,3}$ та Fib3

однак містять більше коротких слів. Подібну закономірність пов'язано і з довжиною роздільника: чим коротшою вона є, тим більше коротких слів містить код, але тим гіршу асимптотичну щільність він має. З огляду на задачу стискання текстової інформації, найбільш ефективними видаються коди з роздільниками довжини 2, які ми й досліджуватимемо найдетальніше.

Побітові алгоритми кодування й декодування

Опишемо алгоритми кодування й декодування для мультироздільникових кодів. Елементами вхідного алфавіту вважатимемо натуральні числа (а отже, для кодування інших алфавітів їхні елементи слід нумерувати). На вхід кодувального алгоритму надходить натуральне число x, двійковий вигляд якого – $x_n x_{n-1}\ldots x_0$, а на виході отримуємо слово з коду $D_{m_1,\ldots,m_t}$. Нехай $J=\{j_1, j_2, \ldots\}$ – така нескінченна зростаюча послідовність натуральних чисел, що j_1 – найменше число, яке не належить множині $\{m_1,\ldots,m_t\}$, а j_i, $i \geq 2$, – таке найменше число, що $j_i > j_{i-1}$ і $j_i \notin \{m_1,\ldots,m_t\}$. Стандартний алгоритм кодування є таким:

1) $x \leftarrow x - 2^n$, тобто видаляємо найстарший біт числа x, який завжди дорівнює 1.

2) Якщо x=0, до рядка $x_{n-1}\ldots x_0$, який містить лише нулі або порожній, дописуємо справа послідовність 1…10 з m_1 одиницями. Шукане кодове слово має вигляд $x_{n-1}\ldots x_0 1\ldots 10$. Кінець алгоритму.

3) Якщо двійкове подання x набуває вигляду рядка 1…10 або 0…01…10 із m_2 або… або m_t одиницями, це і є шукане кодове слово. Кінець алгоритму.

4) Розглядаємо кожну бітову послідовність вигляду 01…10, що не є розташованою в кінці слова послідовністю із m_2 або… або m_t одиницями. Якщо в цій послідовності d одиниць, замінюємо її подібною послідовністю 01…10, що містить j_d одиниць.

5) Якщо слово починається з послідовності 1…10, закінчується послідовністю 01…1 або цілком складається з послідовності вигляду 1…1, замінюємо кількість одиниць у цих послідовностях із d на j_d.

6) Якщо слово закінчується послідовністю 01…10, яка містить m_2 або… або m_t одиниць, це шукане слово. Кінець алгоритму.

7) Дописуємо до слова справа рядок 01…10, що містить m_1 одиниць.

Згідно з цим алгоритмом, якщо $x-2^n \neq 0$, роздільник 01…10 із m_1 одиницями дописується штучно і тому має бути видалений під час декодування, а роздільники вигляду 01…10 із m_2 або… або m_t одиницями є інформативними частинами кодових слів і тому під час декодування повинні оброблятися. Якщо ж $x-2^n = 0$, мають бути видалені останні m_1+1 бітів вигляду 1…10.

Далі описано стандартний алгоритм декодування, на вхід якого подається кодове слово, а на виході отримуємо число.

1) Якщо кодове слово має вигляд 0…01…10 або 1…10 і містить m_1 одиниць, видаляємо останні m_1+1 бітів і переходимо на крок 5.

2) Якщо слово закінчується послідовністю 01…10, яка містить m_1 одиниць, видаляємо останні m_1+2 бітів.

3) Усі послідовності вигляду 01…10, розташовані не в кінці кодового слова, вигляду 1…10 на початку слова та 01…1 в кінці слова, якщо вони містять $j_d \in J$ одиниць, замінюються послідовностями подібного вигляду 01…10, 1…10 або 01…1 відповідно, які містять d одиниць.

4) Якщо слово цілком складається з рядка 1…1 із $j_d \in J$ одиницями, замінюємо його подібним рядком 1…1, що містить d одиниць.

5) Дописуємо символ «1» до початку числа.

Прискорений побайтовий метод декодування коду D_2

Наведені вище алгоритми кодування й декодування обробляють вхідні числа та кодові слова біт за бітом і тому є достатньо повільними. Далі опишемо прискорений алгоритм декодування, що обробляє відразу цілі байти кодових слів, і основна ідея якого подібна до описаної в [5] для кодів Фібоначчі. Ми розглядатимемо саме декодування, оскільки воно виконується в режимі реального часу частіше, ніж кодування. На кожній ітерації в побайтовому методі декодування зчитується ціла кількість байтів закодованого тексту, і з таблиці, подібної до табл. 1, відразу визначаються всі числа, які можна отримати в результаті декодування цих байтів, без урахування найстаршого біта, який завжди є одиничним. У табл. 1 ці числа наведено без старших одиничних бітів і позначено як w_1, w_2, w_3, а їхні довжини – як $|w_1|$, $|w_2|$, $|w_3|$. Можливо, на певних ітераціях вдасться отримати не повний двійковий запис останнього з цих чисел, а лише його старші біти. «Прапорці» f_i вказують, чи повністю декодовано число w_i. Крім того, певну кількість наймолодших бітів у байтах, що обробляються на певній ітерації, можливо, не вдасться декодувати однозначно (оскільки для цього потрібно знати початок наступної порції байтів), або не потрібно обробляти внаслідок особливостей алгоритму. Такі біти позначено через s. Значення цих бітів впливають на спосіб декодування наступної порції байтів. Тому табл. 1 можна розглядати як сукупність двовимірних масивів з індексами s_{prev} (необроблені біти s з попередньої ітерації) та u (чергова порція байтів закодованого тексту). Ці індекси записано в перших двох стовпцях таблиці, а всі інші стовпці містять значення елементів двовимірних масивів.

Для прикладу дослідимо детально метод декодування коду D_2, що на кожній ітерації обробляє по одному байту. Заголовок табл. 1 відповідає саме цьому методу, оскільки, за невеликим винятком, в одному байті може вміститися щонайбільше 3 повних чи неповних кодових слова D_2. Це легко побачити з огляду на те, що найкоротше кодове слово D_2 має вигляд 110. Єдиний варіант, коли байт може охоплювати повністю чи частково чотири кодових слова, – це 0110110x, де x – довільний останній біт, перший біт відповідає останньому біту коду w_1, потім записано коди w_2 і w_3 – двічі 110, а останній біт – це перший біт коду четвертого числа, який можна віднести до необробленого залишку s і обмежитись, таким чином, трьома результуючими числами. Під заголовком у табл. 1 записано зверху вниз ті рядки, що використовуються для декодування тексту 11000111 01101011 11001011 11101101 10011000.

Таблиця 1. **Таблиця для побайтового методу декодування коду D_2**

s_{prev}	u	w_1	$\|w_1\|$	f_1	w_2	$\|w_2\|$	f_2	w_3	$\|w_3\|$	f_3	s
	11000111		0	1	0011	4	0				1
1	01101011		0	1	1	1	0				011
011	11001011	0111001	7	0							011
011	11101101	01111	5	1		0	0				1
1	10011000		0	1	0	1	1	00	2	0	

На рис. 2 проілюстровано декодування другого байта з наведеного прикладу. Неважко показати, що існує всього 6 можливих значень s: порожній рядок, 0, 1, 01, 11, 011. Тому якщо s ='1' (як у першому байті), то ця одиниця не може бути кінцем послідовності 01 чи 011 (оскільки тоді значення s становило б 01 чи 011), а отже, вона має бути кінцем послідовності вигляду 01…1, що містить більше 2 одиниць. Під час декодування коду D_2 у таких послідовностях кількість одиничних бітів зменшується на 1. Усі біти такої послідовності, крім останнього, враховуються під час декодування попереднього байта, а тому рядок s_{prev} ='1' під час декодування наступного (у нашому прикладі – другого) байта не створює жодних бітів у декодованому числі w_1. Те саме стосується і початку другого байта (0110), що, згідно з алгоритмом кодування, є штучно доданим суфіксом і вказує на кінець числа (тому f_1=1), однак не перетворюється на біти самого числа – тому w_1 є порожнім рядком.

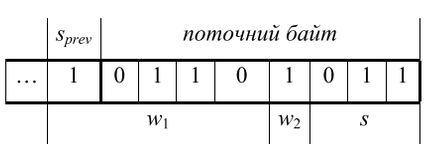

Рис. 2. Декодування байта 10011000

Тепер покажемо, що будь-який рядок табл. 1 може бути «запаковано» в одне 32-розрядне машинне слово. Усі можливі значення s пронумеруємо двійковими числами від 0 до 5, а отже, для зберігання будь-якого такого значення буде достатньо трьох бітів. Зауважимо, що якщо певний прапорець f_i є нульовим (це означає, що слово w_i декодовано не повністю), то слова w_{i+1}, w_{i+2},…, як і прапорці f_{i+1}, f_{i+2},…, можна не розглядати, адже код w_i поширюватиметься до початку рядка s або до правої межі байта. Позначивши ті значення f_i, які можна не враховувати, нулями, отримаємо такі можливі комбінації значень прапорців (f_1,f_2,f_3): 000, 100 та 11x, де x – довільне двійкове значення. Для кожного з цих випадків опишемо свій спосіб пакування рядка табл. 1 у чотирибайтове слово (рис. 3), однак за будь-якого способу значення (f_1, f_2, f_3) записуватимемо в 3 найстарші біти, а значення w_1, $|w_1|$; w_2, $|w_2|$ (якщо є); w_3, $|w_3|$ (якщо є) та s – від наймолодшого біта до старших, у зазначеній послідовності.

(f_1,f_2,f_3)=000. Легко показати, що в цьому випадку значення w_1 займатиме не більше 10 бітів, а отже, для зберігання величини $|w_1|$ достатньо 4 бітів, і загалом пакування рядка табл. 1 у чотирибайтове слово виглядає так, як на рис. 3*а*.

(f_1, f_2, f_3)=100. У цьому випадку значення w_1 буде отримано в результаті декодування щонайбільше 7 бітів, а найбільша можлива довжина w_1 буде на одиницю меншою, тобто $|w_1|\le 6$, і для зберігання значення $|w_1|$ достатньо 3 бітів.

У випадку (f_1,f_2,f_3)=100 потрібно також зберігати значення w_2. Оскільки код w_1 займає принаймні 1 біт байта u, для коду w_2 залишається не більше 7 бітів, що потребує 3 біти для значення $|w_2|$ і в цілому приводить до такого пакування, як на рис. 3*б*.

(f_1,f_2,f_3)=11x. У цьому випадку код w_1 задовольняє ті самі обмеження, що й у випадку (f_1,f_2,f_3)=100, а код w_2, загальна довжина якого не перевищує 7 бітів, повинен також містити роздільник із не менш ніж 3 бітів. Таким чином, для значення w_2 достатньо 4 бітів, для $|w_2|$ – 3 бітів. Оскільки код w_1 займає принаймні 1 біт байта u, а найкоротший код w_2 – це 110, довжина закодованого й декодованого значення w_3 не перевищує 4 бітів, для зберігання значення $|w_3|$ достатньо 3 бітів, і в цілому маємо таке пакування, як на рис. 3*в*.

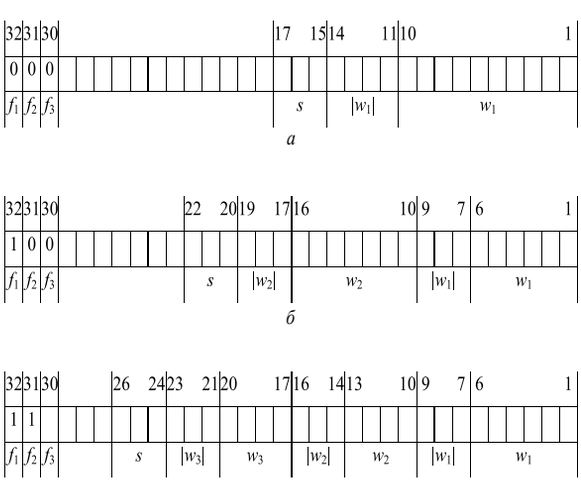

Рис. 3. Пакування рядка декодувальної таблиці в чотирибайтове машинне слово

Тепер побайтовий алгоритм декодування для коду D_2 опишемо детально. Через $x << c$ позначатимемо операцію зсуву значення x вліво, а через $x >> c$ – вправо на c бітів (зсув нециклічний, а нові біти заповнюються нулями). Символом «&» позначатимемо операцію побітового «і», а символом «|» – побітового «або». Через $text_i$ позначимо черговий байт закодованого тексту, через r – запакований у чотирибайтове слово рядок табл. 1, у змінній w формуватиметься декодоване число як конкатенація рядків w_1, w_2 та w_3, а в змінній *len* зберігатимуться довжини цих рядків. Спочатку значення w складається з одного одиничного біта, потім воно зсувається вліво, а праві біти замінюються значеннями w_1, w_2 або w_3 (з відповідних ділянок слова r) і, таким чином, найстарший біт w завжди залишається одиничним.

```
i ← 1;                                          // номер байта закодованого тексту
s ← 0;                                          // спочатку s – порожній рядок
w ← 1;                                          // найстарший біт результату
while (не досягнуто кінця тексту) {
    r ← TAB[s][text_i];                         // зчитуємо 4-байтовий рядок табл. 1
    if(r&0x80000000)  {                         // якщо f1 = 1
      len ← (r >> 6)&0x7;                       // len ← |w1|
      output (w << len)|(r&0x3f);               // декодоване число w із дописаними
                                                // справа 6 наймолодшими бітами r
      w ← 1;
      if(x&0x40000000)  {                       // якщо f2 = 1
        len ← (r >> 13)&0x7;                    // len ← |w2|
        output (w << len)|((r >> 9)&0xf);       // декодоване число: 1w2
        w ← 1;
        len ← (r >> 20)&0x7;                    // len ← |w3|
        if(r&0x20000000)  {                     // якщо f3 = 1
          output (w << len)|((r >> 16)&0xf);    // декодоване число: 1w3
          w ← 1;
        }  else
          w ← (w << len)|((r >> 16)&0xf);       // (f1, f2, f3)=110
                                                  // w ← 1w3
        s ← (r >> 23)&0x7;                      // s у бітах 24–26
      }  else  {                                // (f1, f2)=10
          len ← (r >> 16)&0x7;                  // len ← |w2|
          w ← (w << len)|((r >> 9)&0x7f);       // w ← 1w3
          s ← (r >> 19)&7;                      // s у бітах 20–22
      }
    }  else  {                                  // якщо f1 = 0
        len ← (r >> 10)&0xf;                    // len ← |w1|
        w ← (w << len)|(r&0x3ff);               // дописуємо до w справа w1
          s ← (r >> 14)&0x7;                    // s у бітах 15–17
    }
    i ← i+1;                                    // переходимо до наступного байта
}
```

Визначимо ємнісну складність описаного швидкого методу декодування. Для кожного з 6 можливих значень s_{prev} існує 256 значень u, тому повна табл. 1 містить 6×256 рядків, для зберігання кожного з яких потрібно 4 байти. Таким чином, обсяг пам'яті побайтового методу декодування становить 6 Кб.

Серед кількох класів кодів, що застосовуються для стискання інформації, до мультирозділникових кодів найближчими є коди Фібоначчі, оскільки вони також є префіксними кодами із підтримкою синхронізації, у яких довжина кодового слова не дорівнює цілому числу байтів. Як показано у [5], серед цього сімейства кодів найбільший коефіцієнт стискання в разі застосування до природномовних текстів має код Fib3. Порівняємо складність наведеного вище методу декодування зі швидкими методами декодування для коду Fib3. У [5] описано 3 таких методи, найшвидший із яких – це метод із використанням таблиць. Його ємнісна складність становить 21,4 Кб, тобто перевищує витрати пам'яті для запропонованого нами методу більш ніж у 3,5 рази. Для порівняння часової складності цих методів було проведено чисельний експеримент. Було згенеровано послідовність із 20 млн чисел, розподілених за законом Зіпфа (закон розподілу частот слів у природномовних текстах) у діапазоні від 1 до 100 000, яку було закодовано кодами D_2 та Fib3, а потім декодовано згаданими побайтовими методами й заміряно час декодування. Для підвищення достовірності експеримент було повторено 100 разів і результати усереднено. Ці результати наведено в табл. 2. Як видно, для коду D_2 часова складність декодування є приблизно на 20 % нижчою.

Таблиця 2. **Порівняння складності побайтових методів декодування кодів D_2 та Fib3**

	Побайтове декодування D_2	**Побайтове декодування Fib3**
Ємність	6 Кб	21,4 Кб
Час	0,255 с	0,321 с

Щоб обчислити асимптотичну щільність коду, слід вивести рекурентну формулу для визначення кількості слів довжини n (позначимо її f_n), а потім застосувати стандартну техніку твірних функцій для визначення порядку росту цієї величини. Як показано в [1], для коду D_2 кількість слів довжини n визначається за формулою $f_n=f_{n-1}+f_{n-2}+f_{n-3}+f_{n-6}$, в той час як для коду Fib3 вона становить $f_n=f_{n-1}+f_{n-2}+f_{n-3}$. З цього факту вже видно, що асимптотична щільність коду D_2 вища за асимптотичну щільність коду Fib3. Загалом асимптотичні щільності цих та інших кодів, а також кількості коротких кодових слів наведено в табл. 3.

Як уже зазначалося, для визначення ефективності стискання природномовних текстів припускають, що слова в мові розподілено за законом Зіпфа: якщо обсяг словника становить N і його впорядковано за спаданням частот слів, то ймовірність того, що певне слово в тексті буде займати i-ту позицію у словнику, становить $p_i=1/iH_N$, де $H_N=\sum_{j=1}^{N}\frac{1}{j}\approx \ln N+0,5772$ – N-е гармонійне

Таблиця 3. **Асимптотична щільність та кількість кодових слів для деяких кодів**

Код	**Асимптотична щільність**	**Кількість кодових слів довжини ≤ n**							
		n=2	n=3	n=4	n=5	n=6	n=7	n=8	n=15
Коди з найкоротшим словом довжини 2									
Fib2	$1,618^n$	1	2	4	7	12	20	33	986
D_1	$1,755^n$	1	2	3	5	9	16	28	1432
$D_{1,2}$	$1,618^n$	1	3	5	7	10	16	27	799
$D_{1,3}$	$1,674^n$	1	2	4	7	11	18	30	1106
Коди з найкоротшим словом довжини 3									
Fib3	$1,839^n$	0	1	2	4	8	15	28	2031
D_2	$1,867^n$	0	1	2	4	7	13	24	1906
$D_{2,3}$	$1,785^n$	0	1	3	6	11	19	33	1874
$D_{2,4}$	$1,823^n$	0	1	2	5	9	17	30	1998
$D_{2,5}$	$1,844^n$	0	1	2	4	8	15	28	1999
$D_{2,3,4}$	$1,731^n$	0	1	3	7	13	23	39	1721
$D_{2,4,5}$	$1,796^n$	0	1	2	5	10	19	34	2019
$D_{2,4,6}$	$1,809^n$	0	1	2	5	9	18	32	2032
Коди з найкоротшим словом довжини 4									
Fib4	$1,928^n$	0	0	1	2	4	8	16	1606
D_3	$1,933^n$	0	0	1	2	4	8	15	1510

число. Ефективність кодування текстів мовою, словник якої містить N слів, визначається середньою довжиною кодового слова, що дорівнює

$$\sum_{i=1}^{N} p_i |c_i| = \frac{1}{H_N} \sum_{i=1}^{N} \frac{1}{i} |c_i|. \qquad (1)$$

Нехай s_n – кількість кодових слів, довжина яких не перевищує n, а M_N дорівнює найменшому значенню n, за якого $s_n \geq N$. Покладемо також

$$q_n = \begin{cases} s_n, \text{ якщо } s_n \leq N \\ N, \text{ якщо } s_n > N \end{cases}.$$

Тоді формулу (1) можна переписати у вигляді

$$\frac{1}{H_N} \sum_{k=1}^{M_N} k \sum_{i=q_{k-1}+1}^{q_k} \frac{1}{i} \approx \frac{1}{H_N} \sum_{k=1}^{M_N} k \ln \frac{q_k}{q_{k-1}+1}. \qquad (2)$$

Розраховані за формулою (2) середні довжини кодових слів для різних кодів наведено в табл. 4. Як видно з табл. 4, у разі розподілу частот елементів алфавіту за законом Зіпфа код D_2 матиме вищу щільність, ніж Fib3, для обсягу алфавіту від 100 млн слів і вище. Для більш реалістичних для природних мов обсягів словників (від кількох десятків тисяч до кількох мільйонів слів) код D_2 має дещо нижчу щільність, ніж Fib3, однак погіршення є незначним і становить приблизно 1 % чи менше. Цікавим є код $D_{2,4,5}$ із трьома роздільниками: 0110, 011110 та 0111110. Для алфавітів, обсяг яких не перевищує 1 млн слів, він має кращу щільність за код Fib3. Асимптотична щільність коду $D_{2,4,5}$, проте, дещо нижча. Для коротких алфавітів, обсягом до 20 000–30 000 слів, ще кращий коефіцієнт стискання дає код $D_{2,3,4}$.

Таблиця 4. **Середня довжина кодового слова**

Обсяг алфавіту	Fib3	Fib3⁻	D_2	$D_{2,3,4}$	$D_{2,4,5}$
1000	8,21	8,88 (+8 %)	8,35 (+1,7 %)	7,88 (–4,2 %)	8,03 (–2,2 %)
10 000	10,1	10,61 (+5 %)	10,23 (+1,3 %)	9,92 (–1,8 %)	9,97 (–1,3 %)
100 000	11,99	12,4 (+3,4 %)	12,09 (+0,8 %)	11,99 (0 %)	11,92 (–0,6 %)
1 млн	13,87	14,22 (+2,5 %)	13,94 (+0,5 %)	14,07 (+1,4 %)	13,87 (0 %)
10 млн	15,76	16,06 (+1,9 %)	15,8 (+0,2 %)	16,09 (+2,1 %)	15,81 (+0,3 %)
100 млн	17,65	17,91 (+1,5 %)	17,65 (0 %)	18,17 (+2,9 %)	17,78 (+0,7 %)

У всіх кодів Фібоначчі, в порівнянні з мультироздільниковими кодами, є недолік, що стосується властивості негайного розділення, важливої для пошуку слів у стиснутому файлі без його розархівування. Обидва ці сімейства кодів, а також усі інші застосовувані для стискання текстів коди характеризуються тим, що якщо в стиснутому файлі знайдено входження певного слова w, то ми не можемо гарантувати, що це входження справді відповідає слову w, оскільки воно може бути суфіксом іншого слова. У коді D_2 для перевірки, чи є w справді окремим словом, достатньо переглянути 4 біти, які передують w: якщо це біти 0110, то w – окреме слово, інакше – ні. У всіх мультироздільникових кодах кількість бітів, які потрібно перевіряти для виділення кодового слова в стиснутому файлі, також є фіксованою. Проте в кодах Фібоначчі перевірки будь-якої фіксованої кількості бітів, що передують кодовому слову, недостатньо, оскільки роздільником і найкоротшим словом у цьому коді є послідовність вигляду 1…1, а кілька таких слів можуть «склеюватися», якщо записані одне за одним. Як один зі способів уникнення цієї проблеми у [5] пропонується вилучати з коду Fib3 найкоротше кодове слово 111 – такий код позначимо як Fib3⁻. Однак щільність коду Fib3⁻ є суттєво гіршою за щільність коду D_2 й більшості інших мультироздільникових кодів навіть за реалістичних обсягів алфавітів, елементи яких розподілено за законом Зіпфа. Це видно з табл. 4, де за базу порівняння відносних величин взято код Fib3.

Висновки

У статті означено префіксні стискальні мультироздільникові коди та досліджено деякі їхні властивості. Ці коди мають кілька роздільників вигляду 01…10 і являють собою ширше та гнучкіше сімейство кодів, ніж коди Фібоначчі, що мають роздільники вигляду 1…1. Деякі з мультироздільникових кодів у разі застосування до стискання текстової інформації дають кращий коефіцієнт стискання, характеризуються вищою швидкістю алгоритму негайного розділення, а також вищою швидкістю та меншими ємнісними витратами алгоритму декодування, ніж коди Фібоначчі.

Список літератури

1. Завадський І. О. Префіксний стискальний код на основі нижнього (2,3)-подання чисел / І. О. Завадський // Вісник КНУ ім. Т. Шевченка. Серія фіз.-мат. науки. – 2015. – № 1. – С. 124–129.
2. Apostolico A. Robust transmission of unbounded strings using Fibonacci representations / A. Apostolico, A. S. Fraenkel // IEEE Trans. Inform. Theory. – 1987. – Vol. 33. – P. 238–245.
3. Brisaboa N. An efficient compression code for text databases / N. R. Brisaboa, E. L. Iglesias, G. Navarro, J. R. Parama // Proc. European Conference on Information Retrieval ECIR'03, Pisa, Italy. – 2003. – LNCS 2633. – P. 468–481.
4. Brisaboa N. (S,C)-dense coding: an optimized compression code for natural language text databases / N. R. Brisaboa, A. Farina, G. Navarro, M. F. Esteller // Proc. Symposium on String Processing and Information Retrieval SPIRE'03, Manaus, Brazil. – 2003. – LNCS 2857. – P. 122–136.
5. Klein S. On the usefulness of Fibonacci compression codes / S. T. Klein, M. K. Ben-Nissan // Computer Journal. – 2010. – Vol. 53, № 6. – P. 701–716.

I. Zavadskyi

MULTIDELIMITER CODES

The new family of prefix codes is introduced and investigated. They can be efficiently applied for compressing of texts. Some of codes in this family are 10–30 % closer to Shannon entropy limit, they are faster by 20 % and use memory more efficiently in times than the best known code in Fibonacci codes family.

Keywords: data compressing, encoding, delimiter, prefix code, Fibonacci code, multidelimiter code.

Матеріал надійшов 06.04.2015

УДК 004.056.5

Гончар С. А.

КРИПТОГРАФІЯ З ЧАСОВИМ РОЗКРИТТЯМ: ПЕРСПЕКТИВИ РОЗВИТКУ ДЛЯ БАГАТОЯДЕРНИХ СИСТЕМ

У статті розглянуто можливість використання такого напряму криптографії з часовим розкриттям, як часові замки, для сучасних багатоядерних систем та наведено результати експериментальних досліджень, які показали, що кількість часових замків, процеси розкриття яких проходять одночасно, має бути на одиницю меншою за кількість ядер у процесорі. Новизною цього дослідження також є те, що разом з портативними системами дослідження проводились і на мобільній системі (смартфон).

Ключові слова: криптографія з часовим розкриттям, часові замки, багатоядерні процесори.

Вступ

Комп'ютерні системи з плином часу набувають усе більшого розвитку. Нині в них використовують високопродуктивні складові, про які ще зовсім недавно можна було тільки мріяти: багатоядерні процесори, оперативну пам'ять, яка забезпечує більш швидке оброблення операцій читання/запису процесором, а також більш продуктивно обробляє виконуваний машинний код, графічні