Вісник Київського національного університету імені Тараса Шевченка
Серія фізико-математичні науки

2015, 4

Bulletin of Taras Shevchenko National University of Kyiv
Series Physics & Mathematics

УДК 519.72

І.О. Завадський, к. ф.-м. н., доцент

Швидкий табличний метод декодування мультироздільникових кодів

Київський національний університет імені Тараса Шевченка, 03680, м. Київ, пр-т. Глушкова 4д, e-mail: zava@ukr.net

Igor O. Zavadskyi, Associate Professor, Ph. D.

Fast tabular decoding method for multi-delimiter codes

Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova av., 4d

Описано метод декодування для родини префіксних стискальних кодів, що мають кілька роздільників. Метод базується на побайтовій обробці коду за допомогою таблиць, що визначені на стадії передобчислень. Метод є ефективнішим за аналогічний метод для відомих кодів Фібоначчі як за ємнісними, так і за часовими витратами.
Ключові слова: стискальні коди,мультироздільникові коди, декодування, табличний метод.

This article presents the further investigation of multi-delimiter compressing codes introduced in one of the previous issues of Bulletin of Taras Shevchenko National University of Kyiv. The distinctive feature of this family is the use of multiple codeword delimiters in the same code. The conventional bitwise encoding and decoding algorithms are considered. The fast bytewise decoding algorithm is presented in details for one representative of multi-delimiter codes family. This algorithm utilizes the pre-computed table, which allows to process one byte of encoded text at each iteration. This table serves as two-dimensional array, where bytes of encoded text are indicies and element values are represented by the words of the source text that can be obained from the encoded byte. The time and space complexity of algorithm is assessed and compared with the complexity of similar algorithm for known Fibonacci code Fib3. The space consumption of our algorithm is 3.7 times less than for the Fib3 one and time complexity is 20% lower.
Key words: compression codes, multi-delimiter, decoding, tabular method.

Статтю представив д.ф.-м.н., проф. Анісімов А.В.

Вступ

У роботі [1] було вперше розглянуто родину стискальних префіксних кодів, основною відмінною рисою яких є можливість розділення кодових слів за допомогою кількох спеціальних послідовностей бітів. Це так звані (Δ,k)-коди, або мультироздільникові коди. Представник цієї родини кодів позначається як $C_{m_1,\ldots,m_t}$. Цей код містить усі слова, що задовольняють таким умовам:

- 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.

Тут і надалі ми припускаємо, що $0 < m_1 < \ldots < m_t$.

У [1] доведено такі важливі властивості мультироздільникових кодів, як повноту та універсальність. Цю роботу ми присвятимо розробці алгоритмів кодування та декодування зазначених кодів. Спочатку буде наведено алгоритми, що опрацьовують вхідні числа або кодові слова біт за бітом. Такі алгорими є досить неефективними за обчислювальними витратами, оскільки складність обробки машинного слова (як правило, 4- чи 8-байтового) не вища за складність обробки одного біта, тож опрацьовувати інформацію біт за бітом недоцільно.

Потім ми наведемо значно швидший метод декодування, що дає змогу обробляти за одну ітерацію один байт закодованого тексту. Подібний підхід застосовувався у [2] для відомих кодів Фібоначчі, що були введені в [3]. Однак запропонований нами метод має нижчі як часові, так і ємнісні витрати.

© І.О. Завадський, 2015

Стандартне кодування та декодування

На вхід кодувального алгоритму надходить натуральне число x, двійковий вигляд якого – $x_n x_{n-1} \ldots x_0$, а на виході отримуємо відповідне слово з коду $C_{m_1,\ldots,m_t}$. 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 – 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 є порожнім, дописуємо справа послідовність 1…10, що містить m_1 одиниць. 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, шукане кодове число – це $x_{n-1} \ldots x_0$. Кінець алгоритму.
4) Замінюємо кожну бітову послідовність вигляду 01…10, що містить d одиниць і розташована не в кінці слова, послідовністю 01…10, що містить j_d одиниць.
5) Якщо кінець слова – це рядок вигляду 01…10 із m_1 одиницями, замінюємо його рядком подібної структури 01…10, що містить j_{m_1} одиниць.
6) Якщо слово починається з послідовності 1…10, що містить d одиниць, замінюємо її полідовністю подібної структури 1…10, що містить j_d одиниць.
7) Якщо слово закінчується послідовністю 01…10, яка містить m_2 або … або m_t одиниць, це шукане слово. Кінець алгоритму.
8) 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 $x - 2^n = 0$, останні m+1 бітів вигляду 1…10 мають бути видалені.

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

1) Якщо кодове число 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 число закінчується послідовністю 01…10, яка містить m_1 одиниць, видаляємо останні m_1+2 бітів.
3) Замінюємо кожну послідовність вигляду 01…10 із $j_d \in J$ одиницями послідовністю подібного вигляду 01…10, що містить d одиниць.
4) Якщо слово починається з рядка 1…10 із $j_d \in J$ одиницями, замінюємо його подібним рядком 1…10, що містить d одиниць.
5) Дописумо символ “1” до початку числа.

Прискорене побайтове декодування

Стандартні кодувальний та декодувальний алгоритми обробляють біт за бітом, а тому є достатньо повільними. У разі обробки неперервного потоку закодованого тексту їх можна значно прискорити, застосувавши побайтові табличні методи, основна ідея яких подібна до описаної в [2], [4], [5] для кодів Фібоначчі.

Оскільки декодування виконується в режимі реального часу частіше, ніж кодування, і в цілому триває довше, прискорення декодування є важливішою задачею, на якій ми і зосередимо увагу далі. На кожній ітерації у побайтовому методі декодування зчитується ціле число байтів закодованого тексту, і з таблиці, подібної до табл. 1, відразу визначаються всі числа, які можна отримати в результаті декодування цих байтів. 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 масив з індексами s_{prev} (біти, які на попередній ітерації не були оброблені) 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 побайтовий метод декодування коду C_2, що на кожній ітрації 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 C_2. Це легко побачити з огляду на те, що найкоротше кодове слово C_2 має вигляд 110. Єдиний варіант, коли байт може охоплювати повністю чи частково чотири кодових слова, – це 0110110x, де x – довільний останній біт. Тут 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 C_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. 1 проілюстровано декодування останнього байта з наведеного прикладу.

	s_{prev}	*поточний байт*							
…	1	1	0	0	1	1	0	0	0
	w_1=(1)		w_2=(1)0				w_3		

Рис. 1. Декодування байта 10011000

Визначимо множину можливих значень величини s. Насамперед зробимо такі зауваження.

- Якщо до складу байта входить повністю деяка (Δ,k)-група, то вона може бути декодована однозначно незалежно від вмісту наступного байта, а отже, її біти до складу s не входитимуть.
- Якщо байт закінчується $p \geq 3$ одиницями поспіль, то вони декодуватимуться як p–1 одиниць незалежно від вмісту наступного байта. У цьому випадку рядок s складатиметься з одного останнього одиничного біта, який під час декодування наступного байта служитиме ознакою того, що попередній байт не закінчився нулем.
- Рядок 10 або міститься в кінці (Δ,k)-групи, або кодує значення Δ. В обох випадках його можна декодувати незалежно від вмісту наступного байта: у першому випадку він декодуватиметься разом із (Δ,k)-групою, до складу якої входить, а в другому декодуватиметься як 10.

З першого із цих трьох тверджень випливає, що послідовність s не може містити двох нулів поспіль, адже така ситуація можлива лише якщо ці два нулі складають повну (Δ,k)-групу (тоді її біти не ввійдуть у s) або якщо перший "0" – це кінець однієї (Δ,k)-групи, а другий "0" – початок наступної (у такому разі в s ввійде тільки другий нуль). З другого і третього тверджень безпосередньо випливає, що послідовність s не може містити трьох одиниць поспіль та рядка 10. Таким чином, отримуємо всього 6 можливих значень s: порожній рядок, 0,1, 01, 11, 011.

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. 2), 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. У цьому випадку значення w_1 займатиме не більше 10 бітів. Дійсно, якщо s_{prev}=011 і f_1=0, то найстарший біт байта u не може бути нулем, оскільки інакше склалася б послідовність 0110, яка означає кінець кодового слова і f_1=1. Якщо всі біти u одиничні, то останній біт відноситься до s, а довжина декодованого значення w_1 становить 3+7=10 бітів. Якщо u містить нульовий біт, то під час декодування w_1 оброблятиметься послідовність вигляду 01…10 із $p\geq3$ одиницями, якій відповідатиме на один біт коротший фрагмент коду w_1, а отже, загальна бітова довжина w_1 не перевищуватиме 3+8–1=10 бітів. Якщо ж значення s містить менше трьох бітів, то довжина w_1, очевидно, не може бути більшою за 8+2=10 бітів. Таким чином, у випадку (f_1, f_2, f_3)=000 для зберігання величини $|w_1|$ достатньо чотирьох бітів, а загалом пакування рядка табл. 1 у чотирибайтове слово виглядає так, як на рис. 2а.

(f_1, f_2, f_3)=100. У цьому випадку, якщо $s_{prev} \in \{0,01,011\}$, то конкатенація рядків s_{prev} та u повинна містити роздільник 0110. Те саме справедливо і в тому випадку, якщо $s_{prev} \in \{1,11\}$ і роздільник міститься не на початку конкатенації s_{prev} та u. Значення w_1 буде найдовшим, якщо цей роздільник зсунуто до правої межі байта. Оскільки роздільник під час декодування не враховується, значення w_1 буде отримано в результаті декодування щонайбільше 7 бітів, а з міркувань, викладених для випадку (f_1, f_2, f_3)=000, найбільша можлива довжина w_1 буде на одиницю меншою, тобто $|w_1|\leq6$ і для зберігання значення $|w_1|$ достатньо трьох бітів. Якщо ж s_{prev} – порожній рядок чи s_{prev}=0, то роздільником для w_1 може бути й рядок 110, але нерівність $|w_1|\leq6$ у цьому разі є очевидною.

У випадку (f_1, f_2, f_3)=100 потрібно також зберігати значення w_2. Оскільки код w_1 займає принаймні один 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. 2б.

(f_1, f_2, f_3)=11*x*. У цьому випадку код w_1 задовольняє тим самим обмеженням, 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 трьох бітів. Таким чином, для значення w_2 достатньо чотирьох бітів, для $|w_2|$ – трьох. Оскільки код w_1 займає принаймні один біт байта u, а найкоротший код w_2 – це 110, довжина закодованого й декодованого значення w_3 не більша чотирьох, $|w_3|\leq3$, і в цілому маємо таке пакування, як на рис. 2в.

Тепер детально опишемо побайтовий алгоритм декодування для коду C_2. Через $x << c$ позначатимемо операцію зсуву значення x вліво, а через $x >> c$ – зсуву вправо на c бітів (зсув нециклічний, а нові біти заповнюються нулями). Символом «&» позначатиметься операція побітового «і», а символом «|» – побітового «або». Через $text_i$ позначатимемо 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 чи w_3, а у 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;                                        // ця “1” буде найстаршим бітом
                                              // декодованого числа

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                                  // (f1, f2, f3)=110
                w ← (w << len)|((r >> 16)&0xf);       // 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 байту
}
```

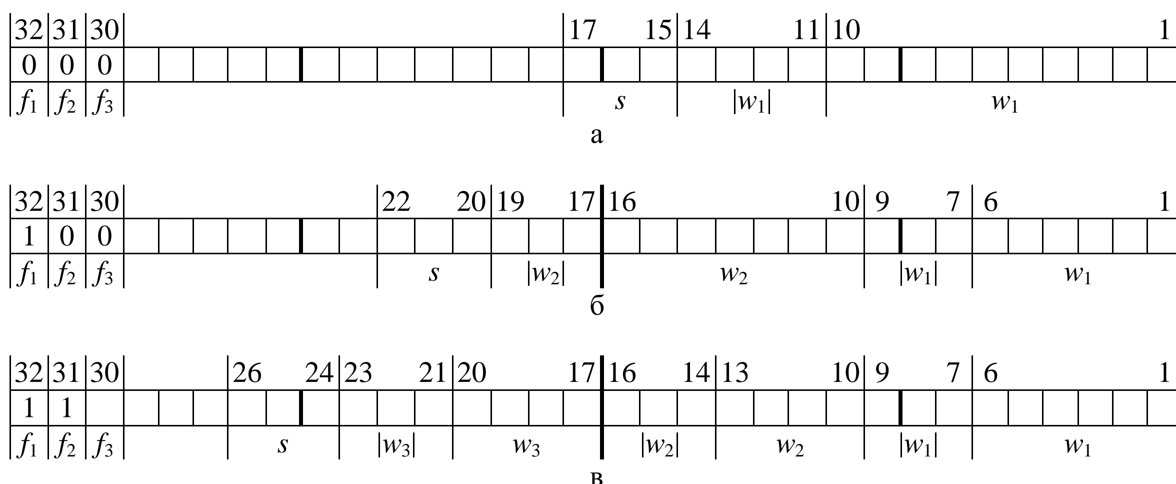

Рис. 2. Пакування рядка декодувальної таблиці у чотирибайтне машинне слово

Оцінка складності

Оцінимо ємнісну складність описаного методу. Для кожного з 6 можливих значень s_{prev} існує 256 значень u_i, а отже, табл. 1 містить 6 × 256 = 1,5 Кб рядків, кожен із яких займає 4 байти. Тому побайтове декодування потребує 6 Кб пам'яті. Зазначимо, що описаний у [2] найоптимальніший із побайтових методів декодування для коду Фібоначчі Fib3 (який, у свою чергу, є найефективнішим серед кодів Фібоначчі в застосуванні до стискання природномовних текстів) потребує 21,4 Кб пам'яті, тобто у 3,7 разів більше, ніж наведений нами метод.

Таблиця 2. Порівняння складнісних характеристик побайтових методів декодування

	Побайтове декодування C_2	Побайтове декодування Fib3
Час	0,255 с	0,321 с
Пам'ять	6 Кб	21,4 Кб

Часову складність побайтового декодування було визначено за допомогою чисельного експеримента. Для прикладу було взято код C_2 і декодовано 20 млн. випадкових кодових слів із частотами, відповідними розподілу Зіпфа (що близький до розподілу слів у природномовних текстах) і цю процедуру повторено 100 разів. Результати експеримента наведено в табл. 2. Як видно, побайтове декодування коду C_2 приблизно на 20% швидше за найоптимальніший з методів декодування коду Фібоначчі Fib3.

Насамперед таке прискорення пов'язане з тим, що у методі декодування для коду Fib3 кожної ітерації виконується 3 читання з таблиць, що зберігаються в пам'яті, в той час як у запропонованому нами методі – лише одне. Це найбільш затратна операція для сучасних процесорів. Решта операцій може бути виконана в регістрах процесора значно швидше.

Висновки

У статті описано швидкий метод декодування мультироздільникових кодів, що базується на обробці відразу одного чи кількох байтів закодованого тексту. Оцінено часову та ємнісну складність методу на прикладі представника сімейства мультироздільникових кодів – коду C_2. Зазначений метод декодування виявився ефективнішим за накращий з подібних методів для кодів Фібоначчі як за часом, так і за витратами пам'яті.

Список літератури

1. *Завадський І.О.* Префіксні (Δ,k)-коди / І.О. Завадський // Вісник Київського національного університету ім. Т. Шевченка. Серія: фіз.-мат. науки. – 2015. – №2. – с. 139–142.
2. *Klein S.T.* On the usefulness of Fibonacci compression codes / S.T. Klein, M.K. Ben-Nissan // Computer Journal. – 2010. – vol. 53, no. 6, pp. 701–716.
3. *Apostolico A.* Robust transmission of unbounded strings using Fibonacci representations / A. Apostolico, A. Fraenkel // IEEE Trans. on Inform. Theory. – 1987. – vol. IT-33, pp. 238–245.
4. *Klein S.T.* Fast decoding of Fibonacci encoded texts / S.T. Klein // Proc. of the Intern. 2007 Data Compression Conf., DCC 2007, IEEE Computer Society, p. 388, 2008.
5. *Walder J.* Fast Fibonacci Encoding Algorithm / J. Walder, M. Kratky, J. Platos // Proceedings of the Dateso 2010 Annual International Workshop on DAtabases, TExts, Specifications and Objects, Stedronin-Plazy, Czech Republic, April 21-23, 2010, p. 72–83.

References

1. ZAVADSKYI, I.O. (2015) Prefix (Δ,*k*)-codes, *Bulletin of Taras Shevchenko National University of Kyiv. Series Physics & Mathematics.* No 2. pp. 139–142.
2. KLEIN, S.T., BEN-NISSAN, M.K. (2010) On the usefulness of Fibonacci compression codes. *Computer Journal*. vol. 53, no. 6, pp. 701–716.
3. APOSTOLICO A., FRAENKEL A. (1987) Robust transmission of unbounded strings using Fibonacci representations. *IEEE Trans. on Inform. Theory*. vol. IT-33, pp. 238–245.
4. KLEIN, S.T. (2008) Fast decoding of Fibonacci encoded texts. *Proc. of the Intern. 2007 Data Compression Conf.*, DCC 2007, IEEE Computer Society, p. 388, 2008.
5. WALDER, J., KRATKY, M., PLATOS, J. (2010) Fast Fibonacci Encoding Algorithm. *Annual International Workshop on DAtabases, TExts, Specifications and Objects*, Stedronin-Plazy, Czech Republic, April 21-23, 2010, p. 72–83.

Надійшла до редколегії 25.11.2015