УДК 512.715:512.772.1:688.321

Р.В. Скуратовський, О.І. Трембовецька

ЗАСТОСУВАННЯ ДИСКРЕТНИХ СТРУКТУР І ЧИСЛОВИХ ПОСЛІДОВНОСТЕЙ ДО БЛОЧНИХ КОДІВ

The purpose to compress information using composition of universal codes with the recursive algorithm of original data recovery was achieved in this work. It obtains compression coefficient higher than in symbolic coding. Proposed method of time coding has reasonable values of compression coefficient and its purpose is coding with compression. For this purpose, entirely new kind of universal coding with the polybasic numeral system was created. The presented method is subtype of universal codes and has an advantage over the Huffman coding for compression, because there is no need to know the exact probability distribution that gives us the characters in the initial data stream and it is a subspecies of the universal coding. The Huffman coding requires exact probability distribution. But when we talk about universal coding it is sufficient to know only the relative order of these probabilities (symbol, are more often, the second of the most common symbol, etc.) withal. Created coding method can be applied in mobile communication and in means of closed communication, if it will be used with block codes, which doesn't scatter symbol frequencies, because it meets modern requirements for cyphering.

Keywords: code, algorithm, archiving, encryption.

Вступ

Сьогодні на практиці стиснення даних поділяється на дві основні групи: 1) стиснення з втратами, в якому дозволяється деградація данних і яке, як правило, використовують для зображень; 2) стиснення даних без втрат, яке повністю зберігає дані і яке, як правило, використовують для текстів, виконуваного коду тощо.

Якщо розглядати стиснення без втрат [1], то більшість методів, що використовуються, – адаптивні, а це означає, що метод адаптується до даних, які підлягають стисненню через кодування. Наприклад, словник з фрагментами даних будується під час початкового проходу по даних. Словники, як правило, побудовані таким чином, що найбільш поширеним фрагментам ставляться у відповідність найкоротші кодові слова.

Такі методи показують найкращий результат, коли про вхідні дані невідомо нічого, і тому неможливо встановити будь-яку попередню домовленість про протокол між сторонами. Саме тоді надлишкова інформація про словник стає необхідною і передається як частина самого повідомлення. При цьому саме повідомлення може бути стиснене найефективніше, оскільки вся інформація і статистика є унікальними для кожного повідомлення і можуть бути використані для побудови словника. Для певних наборів даних показник стиснення k: $k = l(X)/l(Y)\cdot 100\,\%$, де $l(X)$ – довжина вхідного текста, $l(Y)$ – довжина вихідного текста, досягає майже $50\,\%$ [2], що є значним результатом.

До основних недоліків таких методів можна віднести надлишковість даних (треба відправити дані-словник, необхідні для розкодування текста реципієнтом); і, що більш критично, ці методи є не такими потужними для стискуючого кодування коротких порцій тексту, оскільки немає достатньої кількості даних, щоб дозволити значну адаптацію.

Іншою групою дослідження для сфери стиснення даних є *неадаптивні методи*. Як правило, в таких методах замість використання динамічно побудованого словника застосовують *статичний* або *попередньо визначений* словник. Таким чином, необхідність передавати словник відпадає, бо він може бути попередньо створенний на обох сторонах. А оскільки фаза адаптації також не є необхідною, то ефективність і швидкість методу вже не залежать від довжини повідомлення. Основу для багатьох із методів, заснованих на статичному словнику, становлять універсальні коди, що ставлять натуральним числам у відповідність бітові слова, які можуть використовуватися для передачі даних на мережевому рівні моделей OSI та TCP/IP. Як і багато інших галузей, що належать до компю'терних наук, вивчення універсальних кодів ще досі є актуальним. На сьогодні існує багато різних кодів: унарне кодування, гамма-код Еліаса, дельта-код Еліаса [1], кодування Фібоначчі. Потрібно зауважити, що кодування Хаффмана і арифметичне кодування (коли вони можуть бути використані) дають для текстів зі змінним (зміна частот при переході до нового розділу) розподілом частот символів не гірше, а часто і краще, стиснення, ніж будь-який універсальний код.

Постановка задачі

Метою роботи є побудова композиції універсальних кодів, де застосовується рекурсивний алгоритм відновлення початкових даних. Це повинно дати можливість використовувати метод у мобільному зв'язку.

Означення і основні поняття

Елементарними випадковими подіями вважатимемо a_i – символи тексту, які набувають значень з алфавіту *A*, а оскільки добуток, сума і композиція випадкових величин є випадковими величинами, то розглянемо випадкову величину, що відповідає появі тексту:

$$T = \prod_{i_j=1}^{n} \xi_{i_j} = \prod_{j=1}^{n} a_j .$$

Випадкова величина – це вимірна функція, яка діє з $\Omega \to \mathbb{R}$, тобто обов'язково набуває дійсних значень. Тому природно ототожнити весь алфавіт X з $\mathrm{M} \subset \mathbb{N}$.

Означення 1. Таймерний генератор G_T, у загальному сенсі, – це пристрій, який реалізує відлік (інкрементацію) у вибраній системі числення (с.ч.) з постійною частотою і дискретизацією. Як с.ч. може бути непозиційна с.ч., а для таймерної мітки може бути використаний унарний код [3], подібний до коду Райса, в основі якого використана непозиційна одинична с.ч.

G_T буде використовуватися для зупинки генератора коду тексту в потрібний момент – коли на його виході буде відновлено початковий текст.

Універсальним кодом у теорії кодування називається код, що має такі властивості:

1) префіксність;

2) відображає весь ряд натуральних чисел на множину бінарних слів $\{0,1\}^*$;

3) для будь-якого монотонного розподілу (ймовірність того, що число буде наступним у потоці даних спадає при зростанні самого числа) математичне сподівання довжини кодового слова відрізняється не більше ніж на константу від математичного подівання довжини слова для оптимального коду при заданному розподілі частот.

Використання універсального коду для ентропійного кодування

Оскільки використання універсальних кодів пов'язане з апріорним знанням статистичного ранжування символів вхідного потоку (або хоча б приблизного ранжування), яке пов'язане зі знанням функції розподілу ймовірностей $F(x)$, де x – випадкова величина, рівна кількості входжень символу $\alpha_i \in X$ до довільного тексту T. Для створення цього коду використаємо унарний код [3] як допоміжний, за допомогою якого створюється таймерна мітка. При фіксованому ранжуванні універсальний код завжди показує однаково хороший результат в асимптотичному розумінні. Нагадаємо, що вірогідність появи T – це кумулятивна вірогідність $P(T) = p_1 p_2 \dots p_n$, де p_i – вірогідність появи *i*-ї букви у тексті з певної області знань. Самі ж тексти впорядковуються лексикографічно згідно з ранжуванням їх символів $\wp$. Таким чином, існує взаємно однозначна відповідність між множиною текстів T і множиною їх цілочислових рангів $\rho(T), T \in \mathrm{T}$ при фіксованому $\wp$. Сама ж множина T є лексикографічно впорядкованою. У випадку застосування статистично орієнтованого генератора середньостатистичний ранг (вага) тексту рівний математичному сподіванню від рангів усіх символів з урахуванням їх позиційності [4]:

$$M(Wh(W)) = \sum_{i=1}^{k} P(W_i) W_i ,$$

де k – число символів у тексті.

Означення 2. Статистично орієнтований генератор текстів *Gs* – це генератор, який реалізує необхідний ряд розподілення частот $\wp$ символів $n_i < n_j$.

Виходячи з означення *Gs*, його можна застосовувати для серії повідомлень, не передаючи кожний раз словник, що використовується для кодування.

Найкраще стиснення [2, 3] досягається тоді і тільки тоді, коли розподіл частот появи слів з тексту T збігається з розподілом частот кодових слів з коду цього ж тексту – $c(T)$.

Властивість 1. Код, що базується на послідовності Фібоначчі (як і (2, 3)-код), є префіксним.

Дійсно, в коді Фібоначчі не існує підслова 111, тобто воно заборонене. Тому якщо воно чомусь трапляється в цьому коді, то воно є не початком якогось слова, а спеціально введеною міткою закінчення блока цього коду. Таким чином, виконується умова префіксноті, і чітко визначено початок нового блока *F*-коду.

Завдяки цьому досягається можливість пошуку всередині архіву, для чого ставиться мітка на дереві Хаффмана. Таким чином, йому надається властивість префіксності і не потрібно розгортати все дерево, тобто весь архів. Тому достатньо лише, орієнтуючись на мітки, переглянути шуканий блок, де є потрібне входження.

Для виконання стиснення за допомогою універсального коду можна використати таку схему:

1) пронумеруємо вхідні символи в порядку незростання частоти (ранжування відоме, тому така нумерація можлива);

2) кодовим словом для кожного символа вважатимемо закодоване за допомогою універсального коду натуральне число, що відповідає цьому символу (визначене в нумерації на першому кроці);

3) проведемо стиснення за допомогою словника, що визначений на другому кроці алгоритму.

При такому алгоритмі необхідність передавати інформацію, що дає змогу відновити словник, зникає, поки обидві сторони попередньо знають приблизне ранжування символів потоку.

Означення 3. Послідовності U і V називають ортогональними, якщо для довільного номера n виконується умова найбільшого спільного дільника $((u_n, v_n) = 1$. У цьому випадку використаємо позначення $u_n \perp v_n$ для елементів послідовності і $U \perp V$ для відповідних послідовностей.

Означення 4. Зображення натурального числа x у вигляді лінійної комбінації

$$x = a u_n + b v_n \qquad (1)$$

назвемо *лінійною формою* в базисі (U, V).

Означення 5. Лінійну форму (1) назвемо додатно визначеною, якщо $x > 0$ і a, $b \geq 0$.

Означення 6. Лінійну форму (1) назвемо додатно визначеною, якщо $x > 0$ і a, $b \geq 0$.

Для певних чисел існує додатно визначений розклад $x = \lambda_1 u^{n_1} + b y_1,\ 0 < \lambda_1 < b$. Далі

$$x = \lambda_1 u^{n_1} + x_1 v^{k_1} = \lambda_1 u^{n_1} + \\ + b^{k_1}(\lambda_2 u^{n_2} + x_2 v^{k_2}) = \ldots, \qquad (2)$$
$$x_1 = (\lambda_2 u^{n_2} + x_2 v^{k_2}),$$

де $0 < \lambda_1,\ \lambda_2 < b,\ n_1 > 0,\ k_1 > 0,\ k_1 = s_1 + 1,\ x_1$ – залишкове число $(x_1, a) = 1$, $(x_1, b) = 1$ і s_1 – максимальний степінь, такий що $x_1 : v^{s_1}$ і x_1 не діляться на v^{s_1+1}.

Означення 7. Індекс n у зображенні (1) назвемо *рангом* лінійного зображення.

Добре відомо, що якщо $u_n \perp v_n$, то довільне натуральне число x можна подати у вигляді лінійної комбінації типу (1). В дійсній роботі нас цікавлять тільки додатно визначені форми, тому в подальшому покладаємо подання натуральних чисел у вигляді формули (1) додатно визначеним.

Означення 8. Зображення (1) називаємо *канонічним*, якщо $0 \leq a < v_n$.

Очевидно, що застосовуючи тотожне перетворення $x = (a - k v_n) + (b + k u_n) v_n$, завжди можна досягти канонічності зображення (1).

Серед усіх зображень числа x типу (1) нас цікавить зображення максимального рангу. Очевидно, що число x має *єдине канонічне зображення максимального рангу.*

Властивість 2. Зауважимо, що якщо зображення (1) має максимальний ранг, то $x < u_{n+1} \cdot v_{n+1}$.

Ортогональні лінійні форми максимального рангу при рекурсивному застосуванні дають змогу будувати різноманітні змішані системи числення.

Звичайне числення за основою степенів базового числа M отримується рекурсивним застосуванням розкладу чисел у максимальні лінійні форми по степенях числа M.

Більш точно, нехай M – базис системи числення. Розглянемо дві послідовності $U = \{M, M^2, M^3\}$ і $V = \{1, 1, 1, \ldots\}$.

Нехай x – натуральне число, відмінне від нуля. Покладемо $x_0 = x$. Існує таке зображення максимального рангу:

$$x = a_1 M^{n_1} + x_1 \cdot 1. \qquad (3)$$

Серед зображень x типу (3) вибираємо зображення з максимальним значенням a_1. В силу *максимальності рангу* n_1 виконуються нерівності $a_1 < M$ і $x_1 < M^{n_1}$. Потім, аналогічно, розкладемо x_1 в лінійну форму: $x_1 = a_2 M^{n_2} + x_2 \cdot 1$, $a_2 < M$, $n_2 < n_1$. Повторюємо процедуру розкладу для x_2 і наступних чисел. У результаті отримуємо традиційне степеневе зображення x у системі числення з основою M:

$$x = a_1 M^{n_1} + a_2 M^{n_2} + a_m .$$

Зазначимо, що існує необмежена кількість можливих базисних ортогональних послідовностей U і V з різними цікавими властивостями. Для розв'язання різних задач можна підбирати відповідні U і V. У [5–7] вивчено загальні властивості зображення чисел лінійними формами. Також у [5] досліджувались лінійні форми Фібоначчі, тобто випадок, коли ортогональний базис будувався за допомогою пар послідовних чисел Фібоначчі $u_n = F_{n-1}$, $v_n = F_n$. Система числення Фібоначчі має базисну послідовність, яка є ідеальною послідовністю і породжена рекурсивною функцією $F_{n+2} = F_{n+1} + F_n$.

Приклад. Нехай $x = 15411$, базова послідовність задається послідовністю чисел Фібоначчі. Тоді $15411 = 123F_{11} + 31F_{12}$, $123 = 2F_9 + F_{10}$, $31 = 3F_5 + 2F_6$, $3 = F_4$, $2 = F_3$.

Означення 9. Позначимо $\overline{n}_i$ – максимальний степінь 2 у позиційній двійковій с.ч. залишкового числа з (2). Блок $2^{n_i} + 3^{k_i} 2^{n_{i+1}}$ назвемо максимальним, якщо $n_i = \overline{n}_i$.

Теорема 1. Для будь-якого i-го блоку зображення (2), $i = 1, \ldots, t$, виконуються такі властивості:

1) або $n_i = \overline{n}_i$, або $n_i = \overline{n}_i - 1$;

2) виконується нерівність $k_i(1\log_2 3 - 1) - 1\log_2 3 < n_i - \overline{n_{i+1}} - k_i$.

Доведення. Будь-яке $s \in \mathbb{N}$, $2^s \bmod 3 \neq 2^{s-1} \bmod 3$. Тому оскільки $(x_i, 3) = 1$, то має місце $x_i \equiv 2^{\overline{n}_i} \bmod 3$ або $x_i \equiv 2^{\overline{n}_i - 1} \bmod 3$. Тому для будь-якого $s \in \mathbb{N}$ можна однозначно визначити n_i.

З означення $\overline{n}_i$ слідує, що $x_i < 2^{\overline{n}_i + 1}$. Згідно з властивістю 1 теореми 1 маємо $\overline{n}_i + 1 < n_i + 2$. Тому виконується $2^{n_i} + 3 \cdot 2^{\overline{n}_i} < 2^{n_i + 2}$.

$3^{k_i} < 3 \cdot 2^{n_i + \overline{n}_{i+1}}$. Логарифмуючи за основою 2, отримуємо результат $k_i 1\log_2 3 - 1\log_2 3 < n_i - \overline{n_{i+1}}$.

В основу (2, 3)-кодування покладено новий метод зображення цілих чисел у поліосновній системі числення, що використовує основу 2, а також допоміжну основу – 3. Такий вибір зображення натуральних чисел має більш широкий спектр властивостей порівняно з класичною системою числення з єдиною основою – 2 або 3. Ці властивості використовуються в побудові (2, 3)-кодів, і вони також визначають структуру роздільників для кодових слів. Також використане зображення дає змогу представити покращену варіацію для гамма-кодів та дельта-кодів Еліаса порівняно з їх базовим варіантом.

Обґрунтування основних результатів

Поліосновне (2, 3)-кодування являє собою нове сімейство універсальних самостійно синхронізованих префіксних кодів з непостійною довжиною кодового слова. Це сімейство не є узагальненням існуючих префіксних кодів.

Основна частина коду являє собою послідовне чергування груп $0^k 1^l$, кожна з яких, по суті, кодує пару (k, l). Основній частині передує префікс із двох бітів, а в кінці стоїть роздільник, що може бути різним для різних кодів із сімейства.

Верхні оцінки довжини для таких кодів показують перевагу цього сімейства над кодами Фібоначчі. Доведено, що коефіцієнт середньої точкової надлишковості для (2, 3)-коду обмежений зверху величиною 0,16. Для звичайного коду Фібоначчі цей коефіцієнт дорівнює 0,44.

(2, 3)-зображення цілих чисел є частковим випадком більш загальної групи (g, p)-зображення, в якому g є генератором мультиплікативної групи лишків за $\bmod p$, $p \in \mathrm{P}$. Кожне (g, p)-зображення цілих чисел утворює групу префіксних кодів з відповідними роздільниками. Проте кодові слова таких узагальнених кодів обтяжені більшою постійною частиною коду. Жодних переваг у використанні таких кодів порівняно з (2, 3)-кодами ще не виявлено.

Розглянемо (2, 3)-зображення натуральних чисел. Нехай дано число $x \in \mathbb{N}$, $x > 1$, тоді (2, 3)-зображення цього числа можна отримати за таким алгоритмом.

1. Нехай максимальний степінь входження числел 2 та 3 у x – це n_0 та k_0 відповідно. Тоді покладемо $x = 2^{n_0} 3^{k_0} x_1$.

2. Якщо $x_1 = 1$, то процес закінчується. Інакше – перейдемо до кроку 3, поклавши $x = x_1$.

3. Запишемо $x_i = 2^{n_i} + 3^{k_i} x_{i+1}$, де n_i – максимальний степінь 2, такий що $x_i = 2^{n_i} (\bmod 3)$.

4. Якщо $x_{i+1} = 1$, то процес закінчується. Інакше переходимо знову до кроку 3.

На виході цього алгоритму отримуємо набір пар-блоків $(n_0, k_0),(n_1,k_1),\ldots,(n_t,k_t)$ – цей запис і називається (2, 3)-зображенням числа $x \in \mathbb{N}$. Для числа 1 покладемо його рівним $(0,0)$. Очевидно, що для такого зображення виконується $k \geq 1, n_i > n_{i+1}$ при $i \geq 1$. Ця властивість є важливою, і вона використовується при подальшому спрощенні цього зображення.

Через $\overline{n}_i$ позначимо номер найстаршого розряду в зображенні відповідного числа x_i. Тоді $\overline{n}_i \geq n_i$, а оскільки 2^k та 2^{k+1} дають різні залишки при діленні на 3, то виконується одне з двох тверджень: або $\overline{n}_i = n_i$, або $\overline{n}_i = n_i + 1$. В першому випадку зображення називається максимальним, в другому – мінімальним.

Для прикладу розглянемо зображення числа 2014: $2014 = 2 \cdot 1007 = 2^1 \cdot 3^0(512 + 495) = 2^1 \cdot 3^0(2^9 + 159) = 2^1 \cdot 3^0(2^9 + 159) = 2^1 \cdot 3^0(2^9 + 3^2 \cdot 53) = 2^1 \cdot 3^0(2^9 + 3^2 \cdot (2^5 + 21)) = 2^1 \cdot 3^0(2^9 + 3^2 \cdot (2^5 + 3 \cdot 7)) = 2^1 \cdot 3^0(2^9 + 3^2 \cdot (2^5 + 3 \cdot (2^2 + 3)))$.

Тому маємо зображення $2014 = (1,0)(9,2)(5,1)(2,1)$.

Кодування чисел за допомогою (2, 3)-зображення. В наведеному вище зображенні є недолік, якого, власне кажучи, хотілося б позбутися, а саме: перший блок зображення може бути довільно великим і не пов'язаний жодною закономірністю з наступними блоками. Згідно з принципом кодування, перший блок являє собою степені входження 2 та 3 у початкове число. Він є ненульовим лише для тих чисел, які діляться або на 2, або на 3. Тобто якщо початкове число не буде ділитися на жодне з них, то блок буде мати постійний вигляд (0, 0) і його можна буде опустити.

Введемо префікс для (2, 3)-кодування таким чином.

1) Якщо початкове число x не ділиться ні на 2, ні на 3, то префікс – 00, а саме число залишається незмінним.

2) Якщо число ділиться на 2 і при цьому взаємно просте з 3, то покладемо префікс рівним 11, а $x = x + 3$.

3) Якщо число ділиться на 3 і при цьому взаємно просте з 2, то покладемо префікс рівним 10.

4) Інакше, якщо число ділиться і на 2, і на 3, то префікс – 01, а $x = x + 1$.

Після такого перетворення число вже не є кратним 2 або 3. Для нього необхідно знайти (2, 3)-зображення і вилучити перший блок. Отримали: $x = (n_1,k_1),\ldots,(n_t,k_t)$.

Для подальшого спрощення розглянемо теорему, що дасть змогу перейти до різниці між елементами сусідніх блоків.

Теорема 2. Якщо блок i з (2, 3)-зображення є максимальним, то виконується така нерівність: $n_i - \overline{n_{i+1}} - k_i \log_2 3 > 0$. Якщо i-ий блок є мінімальним, то $1 > n_i - \overline{n_{i+1}} - k_i \log_2 3 > -\log_2 3$.

Доведення. Якщо блок з номером i є максимальним, то виконується нерівність $x_i < 2^{\overline{n}_i+1}$, з цього отримуємо $2^{n_i} + 3^{k_i} 2^{\overline{n}_i+1} < 2^{n_i+1}$, а в логарифмічній формі з цього і випливає шукана нерівність. Якщо ж блок є мінімальним, то $x_i < 2^{n_i+2}$, тобто $2^{n_i} + 3^{k_i} \cdot 2^{\overline{n}_i+1} < 2^{n_i+2}$, що еквівалентно $-\log_2 3 < n_i - \overline{n}_{i+1} - k_i \log_2 3$. З цього отримали нижню границю шуканої нерівності.

Оскільки $\overline{n}_i < n_i + 1$, то $x_i > 2^{n_i+1}$. Також виконується $x_{i+1} < 2^{n_{i+1}+1}$. З цього слідує $2^{n_i+1} < x_i < 2^{n_i} + 3^{k_i} + 2^{\overline{n}_{i+1}+1}$. А в логарифмічній формі це дає верхню границю нерівності, що й потрібно було довести.

Означення 10. Вагою i-го блока назвемо величину $\Delta_i = n_i - \overline{n_{i+1}} - k_i$.

Теорема 2 дає можливість при розгляді (2, 3)-зображення $x = (n_1,k_1),\ldots,(n_t,k_t)$ перейти до розгляду зображення

$$x = (\Delta_1,k_1),\ldots,(\Delta_t,k_t).$$

Залежно від вибору Δ_i отримуємо різні коди з цього сімейства. Розглянемо вибір $\Delta_i = n_i - \overline{n}_{i+1}$, що також забезпечить додаткову властивість $\Delta_i \geq k_i \geq 1$.

Далі наші пари чисел будемо кодувати групою бітів $0^{\Delta_i}1^{k_i}$. Основна частина кодового слова матиме вигляд $0^{\Delta_1}1^{k_1}, 0^{\Delta_2}1^{k_2},\ldots,0^{\Delta_t}1^{k_t}$. Залишається вибрати роздільник для кодових слів, що забезпечить властивість префіксності. Таким роздільником можна вибрати послідовність 01^l, де $l \geq 2$. Будемо розглядати лише найкоротший роздільник – 011. Саме такий блок не може зустрітися у кодовому слові для вибраного Δ_i.

Таким чином, отримали кодування для натуральних чисел. Для $x \in \mathbb{N}$ воно має такий вигляд: $x = \alpha 0^{\Delta_1} 1^{k_1}, 0^{\Delta_2} 1^{k_2}, \ldots, 0^{\Delta_t} 1^{k_t} 011$, де α – префікс, а Δ_i, k_i – відповідні коефіцієнти стисненого (2, 3)-зображення.

Таблиця кодів для перших 16 натуральних чисел має такий вигляд:

x	(2, 3)-кодування x	x	(2, 3)-кодування x
1	001011	9	100001011
2	1101011	10	110011011
3	1001011	11	000001011
4	11001011	12	010011011
5	0001011	13	000011011
6	01001011	14	1100011011
7	00001011	15	1000011011
8	110001011	16	1100001011

Стиснення даних з використанням (2, 3)-кодування. Розглянемо подальші оптимізації, що дають змогу використати (2, 3)-кодування для більш ефективного стиснення даних.

Префікс, який використовується для того, щоб перевести число у множину чисел, які є взаємно простими з 2 та 3, не є необхідною чистиною коду, якщо розглядати лише ті числа, що вже належать цій множині. Отже, ми змінюємо нумерацію таким чином, що числу k відповідає кодування k-го числа, що є взаємно простим з 2 та 3 без префікса.

Далі, як видно, довжина коду не зростає монотонно при зростані самого числа, що кодується. А оскільки при ентропійному стисненні нумерація йде числами від найменшого і припускається, що меншому числу відповідає більш короткий код, то має сенс упорядкувати їх за зростанням довжини коду.

Таблиця перших 16-ти кодів, що можна використати для кодування, має такий вигляд:

x	(2, 3)-кодування x	x	(2, 3)-кодування x
1	1011	9	01001011
2	01011	10	00101011
3	001011	11	000011011
4	0001011	12	000001011
5	0011011	13	001001011
6	0101011	14	000101011
7	00011011	15	010001011
8	00001011	16	010011011

Програмна реалізація (2, 3)-кодування. Основною метою роботи було побудувати систему з консольним доступом, що вміє робити кодування/декодування чисел з допомогою коду Фібоначчі та (2, 3)-кодування для стиснення, а також розробити модуль, що дає змогу будувати словник для стиснення та виконувати саме стиснення і зворотній до нього процес.

Для користувача така система дає змогу побудувати словник на основі типово-шаблонних текстів, потім виконати поширення словника на комп'ютерах, що можуть бути використанні для прийому і декодування. Після цього з'являється можливість використовувати систему для передачі багатьох повідомлень на основі побудованого словника. Для цього система, приймаючи як аргументи вхідний текст та словник, створює стиснений файл, що готовий для передачі і подальшого декодування.

Бітові коди для створення таймерного коду вводу/виводу в унарній системі числення

Як зазначалося, одиницею інформації на файловій системі є байт, тому у випадку, коли кількість біт, що записуються, не кратна 8 (кількість бітів у одному байті), то в кінці неповного байту записується одиниця, за якою слідують нулі у кількості, що необхідна для "завершення" байту. Якщо ж кількість біт кратна 8, то для того щоб можна було відрізнити ситуацію при зчитуванні від попередньої, в кінець файлу додається нульовий байт, що індикує завершення потоку.

Наприклад, 0 → 01000000, 1000 → → 10001000, 1101010 → 11010101, 00100101 → → 00100101 000000000.

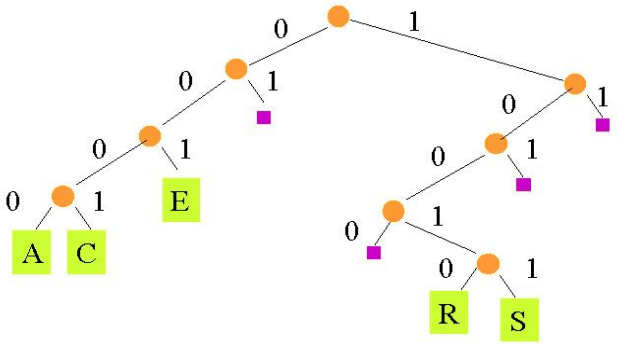
Основними функуціями є такі.

- Функцію *Reader* реалізовано на основі буферного *BReader*. Він забезпечує зчитування бітів з файлу, обробку останнього байту (відповідно до наведоної вище схеми), має індикатор, що показує завершення потоку і дає змогу "дивитися" наступний біт без його зчитування. Для цього у змінній об'єкта зберігається оброблюваний байт, флаги, що індикують, чи закінчився потік даних та чи є поточний байт останнім, а також деяка додаткова інформація.
- *BitWriter* призначений для запису у файл кодових бітових слів.

Префіксне дерево для обробки потоку кодових слів

Основною структурою для обробки вхідного потоку бітів і його перетворення на символи є префіксне дерево (рисунок). Така струк-

тура даних також називається асоціативним масивом, бо дає можливість поставити у відповідність впорядкованому набору даних (у нашому випадку – це набір бітів) певне значення (у нашому випадку – символ тексту). На рисунку показано приклад такого дерева для символів A, C, E, R, S та відповідних кодів 0000, 0001, 001, 10010, 10011.



Префіксне дерево

На префіксному дереві діють дві операції. Перша – додати асоціативну пару: кодове слово → символ. Друга – опрацювати наступний біт (якщо це спричиняє появу нового символа, то він буде результатом виконання описаної вище функції). Для ефективної підтримки таких операцій в дереві зберігаються: 1) допоміжна інформація про вершини, а саме чи є вона листом, значення якого асоціюється зі шляхом, що починається в корені і закінчується в цій вершині; 2) номера вершин, до яких потрібно перейти при обробці наступного біта.

Алгоритм додавання асоціації.

1) Ініціалізуємо поточну вершину коренем дерева.

2) Для кожного біта з кодового слова у порядку розрядності виконуємо кроки 3, 4.

3) Якщо не існує переходу з поточної вершини по заданому біту, то будуємо новий перехід, що веде з поточної вершини до нової вершини по заданому біту.

4) Робимо перехід: поточна вершина стає вершиною, до якої веде перехід по заданому біту.

5) Додаємо значення-асоціацію в поточну вершину, що була передана, як аргумент.

Для обробки тексту ініціалізуємо поточний стан кореневою вершиною, а при обробці наступного біта робимо перехід (відсутність переходу може бути лише у випадку невалідного тексту або непрефіксного кодування). Якщо вершина, в яку потрапили, містить асоціацію, то додавання асоціації робити вже не треба, тому повертаємо її (цю асоціацію), а поточний стан встановлюємо знову в кореневу вершину для наступного циклу роботи алгоритму.

Варто зауважити, що така структура працює коректно лише для префіксних кодів, бо інакше будуть з'являтися недосяжні вершини і не гарантується коректне розпізнавання тексту навіть у випадку однозначності.

Висновки

Розроблений метод таймерного кодування дає можливість спрощення процесів кодування і декодування. Універсальні коди можуть бути використані замість кодів Хаффмана, більш поширене їх використання – доповнення до динамічної схеми кодування. Головною його перевагою є можливість стиснення до об'єму меншого, ніж це може зробити традиційне символьне кодування. (2, 3)-кодування є простим для реалізації, а додаткова властивість самостійної синхронізації зумовлює підвищення стійкості та можливість ізолювати помилки, що виникають при передачі даних, таким чином зменшуючи кількість пошкодженої інформації.

Отже, для розробленого методу таймерного кодування як підвиду універсальних кодів над кодами Хаффмана для стиснення треба зберігати лише відносну шкалу частот символів і пересилати лише таймерну мітку. Кодування Хаффмана потребує точного розподілу ймовірностей, тоді як у випадку універсального кодування достатньо знати лише відносний порядок цих імовірностей.

Перспективою розвитку методу є створення способу кодування графічних файлів. Найбільш доцільним є застосування алгоритму для представлення числових зображень у кодуванні СМС-повідомлень, оскільки вони потребують найменшого обєму файлу.

Список літератури

1. *S. Ristov and E. Laporte,* "Ziv lempl compression of huge natural languadge data tries using suffix arrays", 10th Annual Symp. ACM. Experimentation, Combinatorial Pattern Matching, UK, Warwick University, M. Croche-

more and M. Paterson, eds. Berlin: Springer, 1999, pp. 196–211.

2. *Методы* сжатия без потерь / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. – М.: Диалог-МИФИ, 2002. – 384 с.
3. *K. Sayood*, Introduction to Data Compression, 3rd ed., Morgan Kaufman, 2006, p. 315.
4. *Скуратовский Р.В.* Метод быстрого таймерного кодирования текстов // Кибернетика и системный анализ. – 2013. – № 1. – С. 154–160.
5. *H. Yamamoto,* "A new recursive universal code of the positive integers", IEEE Trans. Inform. Theory, vol. 46, pp. 717–723, 2000.
6. *Скуратовський Р. В.* Комбінована λ,ν-адична система числення і деякі математичні об'єкти, які з нею пов'язані // Студентскі фіз.-мат етюди. – 2003. – С. 45–50.
7. *Стахов А.П.* Коды золотой пропорции. – М.: Радио и связь, 1994. – 152 с.

Рекомендована Радою
фізико-математичного факультету
НТУУ "КПІ"

Надійшла до редакції
3 квітня 2014 року