УДК 519.72

ЗАВАДОСТІЙКИЙ КОД НА ОСНОВІ СКІНЧЕННОГО АВТОМАТА ТА ПОДАННЯ ЧИСЕЛ У ДВОБАЗИСНІЙ СИСТЕМІ ЧИСЛЕННЯ

І.О. Завадський

КНУ ім. Т. Шевченка, 03680, проспект Академіка Глушкова, 4д.
Тел.: (044) 252 0927, ihorza@gmail.com

Запропоновано новий метод завадостійкого кодування, що поєднує кілька підходів до побудови завадостійких кодів: з використанням арифметичних властивостей чисел, що подаються вхідними бітовими послідовностями, кодування скінченним автоматом та простим блоковим кодом. Наведено як алгоритм побудови коду, так і алгоритм декодування.

A new error correcting code is introduced. It combines several approaches to constructing of error correcting codes: utilizing the arithmetic properties of numbers, represented by input bit sequences, encoding by finite automaton and by simple block code. Either encoding and decoding algorithms are presented.

Вступ

У теорії завадостійкого кодування скінченні автомати насамперед використовуються як засіб подання діаграм станів згорткових кодів [1]. Згортковий код швидкості l/n і з пам'яттю обсягом m бітів визначається скінченним автоматом, що має 2^m станів, і з кожного стану є 2^l переходів. Перехід визначається l двійковими вхідними символами, які зчитує автомат. У результаті переходу автомат генерує n вихідних символів, що визначаються за n формулами, однаковими для всіх станів. Кожна з таких формул є сумою за модулем 2 деякого піднабору вхідних бітів. Як бачимо, діаграма станів згорткового коду є дуже специфічним різновидом автомату, що має задовольняти жорстким обмеженням. Головною перевагою такої структури є простота її апаратної реалізації, однак у разі програмної реалізації ця перевага не є надто суттєвою.

У цій доповіді буде показано, що завадостійкі коди можна будувати на основі значно ширшого кола різновидів скінченних автоматів. Буде розглянуто приклади коду, що генерується автоматом, який не задовольняє зазначеним вище обмеженням. Зокрема вихідні символи обчислюються не як суми за модулем 2 наборів вхідних символів, а за значно складнішими формулами, у яких вхідна послідовність бітів розглядається як набір цілих чисел і над цими числами виконуються певні арифметичні операції. Якщо довжина таких чисел не перевищує 64 біти, то такі операції, як додавання чи множення фактично виконуватимуться не довше, ніж сумування певних бітів за модулем 2, оскільки сучасні комп'ютери є переважно 64-розрядними. Математичною основою розглянутих кодів є подання чисел у двобазисній системі числення з основним базисом 2 та додатковим базисом 3, яке було вперше досліджено А.В. Анісімовим [2].

1. Нижній (2,3)-код

Метод кодування на основі подання чисел у двобазисній двійково-трійковій системі описано в [2]. Йдеться про префіксні коди змінної довжини, у яких кодові слова розділяються спеціальними послідовностями-роздільниками. Цей метод має переваги порівняно з іншими методами префіксного кодування, зокрема довжина коду буде меншою, ніж у кодах Фібоначі. Обчислювальний експеримент показує, що в середньому довжина (2,3)-коду числа перевищує довжину його двійкового подання у 1,13 разів. Однак для деяких чисел довжина (2,3)-коду буде меншою за довжину двійкового подання, що дає змогу використовувати такий код для стискання даних. З іншого боку, надлишковість, яку має (2, 3)-код у середньому, дає можливість використовувати його з метою виправлення помилок, що виникають у каналах зв'язку та пристроях збереження даних.

З метою підвищення завадостійкості у [3] введено модифікацію (2, 3)-коду – так званий ***нижній (2,3)-код***, який має дещо більшу довжину, ніж код [2], але й вищу завадостійкість. Довжина нижнього (2,3)-коду перевищує довжину двійкового подання числа у 1,16 разів у середньому.

Основна ідея методу кодування така. Нехай $\mathbf{N}_{2,3}$ – множина натуральних чисел, які є взаємно простими із 2 і 3, $x \in \mathbf{N}_{2,3}$, $x > 1$, $n = [\log_2 x]$. Тоді x можна подати у формі $2^{n-1} + 3^k x_1$ або $2^{n-2} + 3^k x_1$, де $k \in \mathbf{N}_{2,3}$, $x_1 \in \mathrm{N}_{2,3} \cup \{2\}$, $x_1 < x$ і тільки в одній із цих форм. Застосовуючи аналогічний розклад до x_1, отримаємо x_2 і далі будемо обчислювати x_{i+1} з рівності $x_i = 2^{b_i} + 3^{k_i} x_{i+1}$, поки на певній ітерації t не отримаємо $x_t = 1$ або $x_t = 2$. Для однозначного декодування чисел достатньо зберігати величини $\Delta_i = [\log_2 3^{k_i} x_{i+1}] - n_i$ і k_i, які отримуємо на кожній ітерації. Справді, якщо відомі значення x_{i+1}, Δ_i і k_i, можна обчислити величину

$m_i = [\log_2 3^{k_i} x_{i+1}]$, потім $-b_i = m_i - \Delta_i$, а потім і $x_i = 2^{b_i} + 3^{k_i} x_{i+1}$. Таким чином, під час декодування послідовність значень x_i відновлюється у зворотному порядку: x_t, … , x_0, де $x_t = 1$ або $x_t = 2$. Для вибору одного з двох початкових значень слід взяти до уваги, що якщо $x_{t-1} = 7 = 2^0 + 3^1 \cdot 2$, то $x_t = 2$, $b_i = 0$, $k_{t-1} = 1$, $m = [\log_2 3^k x_1] = 2$, $\Delta_{t-1} = m - b = 2$. Легко показати, що $x_{t-1} = 7$ – це єдиний випадок, коли $k_{t-1} = 1$ і $\Delta_{t-1} = 2$, і єдиний випадок, коли $x_t = 2$. Тому, якщо $k_{t-1} = = 1$ і $\Delta_{t-1} = 2$, покладаємо $x_t = 2$, а інакше – $x_t = 1$.

Нехай $2^b + 3^k x_i$ – нижнє (2, 3)-подання числа x, $m = [\log_2 3^k x_i]$, $\Delta = m - b$. Величина Δ має важливу властивість: вона може набувати лише значень 0, 1 або 2, якщо

$$2^m < 3^k x_i \le \frac{7}{8} 2^{m+1} \tag{1}$$

і лише значень 0 або 1, якщо

$$\frac{7}{8} 2^{m+1} < 3^k x_i \le 2^{m+1}. \tag{2}$$

Якщо припустити, що x є рівномірно розподіленою на відрізку $[2^n; 2^{n+1}]$ випадковою величиною, то різні значення Δ траплятимуться з такими частотами: у випадку (1) $P(\Delta_i = 0) = 3/14$, $P(\Delta = 1) = 5/14$ і $P(\Delta = 2) = 3/7$, а у випадку (2) $P(\Delta = 0) = P(\Delta = 1) = 1/2$. Серед значень *k* з ймовірністю 2/3 траплятиметься 1, а ймовірність кожного наступного значення *k* буде втричі меншою за ймовірність попереднього.

Нижній (2, 3)-код числа x – це послідовність блоків бітів вигляду 0…01…1, перший з яких кодує подання числа *x,* а наступні – подання x_i на подальших ітераціях. Кількість одиниць у блоці дорівнює k_i, а кількість нулів визначається величиною Δ_i. У випадку (1) значення $\Delta_i = 0$ кодується трьома нулями, $\Delta_i = 1$ – двома, а $\Delta_i = 2$ – одним. У випадку (2) $\Delta_i = 0$ кодується двома нулями, а $\Delta_i = 1$ – одним нулем. Такий спосіб кодування обраний з огляду на вищезазначені ймовірності різних значень k_i та Δ_i з метою скоротити середню довжину коду. Також з метою скорочення змінюється кодування числа 5: замість 0001 записуємо 001 (це не призводить до двозначності, адже за стандартного кодування кінцівка коду 001 ніколи не трапляється).

Приклад. Побудуємо нижній (2, 3)-код числа *x* = 1387.

1) $1387 = 2^8 + 3^1 \cdot 377$, а отже, $b_0 = 8$, $k_0 = 1$, $x_1 = 377$. Тут $3^k x_1 = 3^1 \cdot 377 = 1131$, $m_0 = [\log_2 3^k x_1] = 10$, $\Delta_0 = m_0 - b_0 = 2$.
2) $377 = 2^7 + 3^1 \cdot 83$, а отже, $b_1 = 7$, $k_1 = 1$, $x_2 = 83$. Тут $3^k x_2 = 3^1 \cdot 83 = 249$, $m_1 = [\log_2 3^k x_2] = 7$, $\Delta_1 = m_1 - b_1 = 0$.
3) $83 = 2^5 + 3^1 \cdot 17$, а отже, $b_2 = 5$, $k_2 = 1$, $x_3 = 17$. Тут $3^k x_3 = 3^1 \cdot 17 = 51$, $m_2 = [\log_2 3^k x_3] = 5$, $\Delta_2 = m_2 - b_2 = 0$.
4) $17 = 2^3 + 3^2 \cdot 1$, а отже, $b_3 = 3$, $k_3 = 2$, $x_4 = 1$. Тут $3^k x_4 = 3^2 \cdot 1 = 9$, $m_3 = [\log_2 3^k x_3] = 3$, $\Delta_3 = m_3 - b_3 = 0$.

У другому розкладі величина $3^k x_i$ задовольняє нерівностям (2), а в усіх інших – нерівностям (1). Тому коди нижніх (2, 3)-розкладів 1)–4) будуть такими: 01, 001, 0001, 00011. Їх конкатенація і є нижнім (2, 3)-кодом числа 1387: 01001000100011.

2. Завадостійкий нижній (2, 3)-код

Якщо в каналі зв'язку, яким передається нижній (2, 3)-код, виникли перешкоди, що призвели до інвертування одного чи кількох бітів цього коду, наявність помилок у деяких випадках можна виявити за такими ознаками:

1) у випадку (2) значення Δ кодується більш ніж двома нулями;
2) у випадку (1) значення Δ кодується більш ніж трьома нулями;
3) величина $m - \Delta$ від'ємна.

Чим довшим є кодове слово, тим вищою є ймовірність, що в разі наявності помилок виконається одна з цих ознак, а отже, факт наявності помилок буде виявлено. Також ймовірність виявлення помилок зростатиме зі зростанням їхньої кількості. У роботі [3] описано алгоритм, що дає змогу виправляти одну помилку в кодовому слові нижнього (2, 3)-коду з ймовірністю 100% і 2 помилки з ймовірністю 87,5%. Це досить низькі показники. Однак завадостійкість нижнього (2, 3)-коду можна суттєво підвищити завдяки його обробці показаним на рис. 1 скінченним автоматом. Назвемо результуючий код на виході цього автомату ***завадостійким нижнім* (2, 3)-*кодом***.

Автомат має головку, яка зчитує символи нижнього (2, 3)-коду зліва направо. У автомата є 5 станів. Коли головка зчитує символи, стан автомату змінюється відповідно до зображеної на рис. 1 діаграми і у код на виході записуються символи, що відповідають виконаному переходу; ці символи вказано над стрілками переходів після скісних рисок. Стани позначено кругами. Перша цифра у кругу – це номер стану, а після неї в дужках записано символи, що розташовані перед головкою автомату, коли він перебуває у цьому стані.

Автомат переходить у той чи інший стан залежно від символів, зчитаних головкою, а також деяких інших умов; ці символи та умови зазначено над стрілками переходів перед скісними рисками. Умови пов'язані з тим, що деякі переходи автомат виконує залежно від значення певної хеш-функції, застосовної до величини x_i, отримуваної на кожній ітерації під час побудови нижнього (2, 3)-коду. Цією хеш-функцією може бути, наприклад $G(x_i) = x_i \bmod 20$. Оскільки x_i – непарне число, то ця функція може набувати 10 значень. Якщо розподілити ці значення за двома множинами G_1 і G_2 так, що G_1={1, 3, 5, 7, 11}, G_2 ={9, 13, 15, 17, 19}, то ймовірності потрапляння значень $G(x_i)$ у кожну з цих множин будуть приблизно однаковими. Символ G_1 або G_2 над стрілкою переходу означає, що перехід здійснюється лише якщо поточне значення $G(x_i)$ належить множині G_1 або G_2 відповідно (та/або виконуються інші умови переходу). Таким чином, під час обробки нижнього коду автоматом мають обчислюватися й значення x_i (або можуть використовуватися значення, отримані під час побудови нижнього (2, 3)-коду). Кожне наступне таке значення обчислюватиметься після обробки чергової групи символів 0…01…1, що кодує пару значень (Δ_i, k_i) – її ми назвемо **(Δ, *k*)-*групою***. З причин, які стануть зрозумілі з методу декодування, (Δ, *k*)-групи нижнього (2, 3)-коду мають оброблятися зображеним на рис. 1 автоматом справа наліво (проте всередині груп змінювати порядок бітів не потрібно).

Щоб запобігти захаращенню зображення, переходи у стан 5 на рис. 1 не позначено. Це всі ті самі переходи, що й у стан 1, але здійснюються вони в разі виконання нерівностей (2), в той час як переходи у стан 1 – у разі виконання нерівностей (1). Оскільки і нерівності (1), і нерівності (2) не можуть виконуватися водночас, то як під час кодування, так і під час декодування завжди можна визначити однозначно, у який стан – 5 чи 1 – слід переходити.

Стан 3 – особливий. Кожен із двох переходів у цей стан може виконуватися у разі зчитування не однієї певної групи символів, а двох різних груп, позначених на рис. 1 як A і B. Щоб розрізняти випадки A і B під час декодування, спосіб виходу зі стану (3) залежить від того, за якою групою символів було здійснено перехід у цей стан: A чи B. Наприклад, якщо автомат перейшов у стан 3 зі стану 2, зчитавши символи 11 (випадок A), а потім головка прочитала 0, автомат перейде зі стану 3 до стану 1 за стрілкою A & 0 і запише 00.

Автомат починає роботу у стані 1, якщо величина $3^{k_{t-1}} x_t$ задовольняє нерівності (1), і у стані 5, якщо ця величина задовольняє нерівності (2). Перший символ коду, який завжди дорівнює 0, автомат пропускає і починає декодування з другого символу. На кожному переході автомат записує у вихідний код 3 двійкові символи, хоча з кожного стану є 4 переходи і для їхнього кодування вистачило б і двох бітів. Це зроблено з метою підвищення завадостійкості. Трійки двійкових символів, що позначають переходи з певного стану, утворюють множину слів з Хемінговою відстанню 2: {000, 011, 101, 110}. Таким чином, якщо у вихідному коді в трійку бітів з номерами 3*k*+1, 3*k*+2 і 3*k*+3 буде внесено 1 або 3 помилки, ця трійка вже не належатиме вказаній вище допустимій множині слів, а отже, такі трійки можна буде відразу виявити. Тоді питання полягатиме лише в тому, на якій саме позиції (чи позиціях) розташовані помилкові біти в кожній помилковій трійці. З'ясувати це дозволяє описаний у наступному розділі алгоритм ефективного перебору можливих варіантів розташування помилок.

Коли автомат досягає кінця кодового слова нижнього (2, 3)-коду, можливі такі особливі випадки:

1) кодове слово закінчується символами 101, головку розміщено після символу 0, автомат перебуває у стані 1 і для останнього біту 1 переходу зі стану 1 немає. У цьому випадку до вихідного коду дописуємо справа біти 011, що означатиме символи 10 у вхідному коді і дасть змогу коректно відтворити вхідне число згідно з наведеним нижче алгоритмом декодування;

2) кодове слово закінчується символами 1001, головку розміщено після символів 10, автомат перебуває у стані 1 і для останніх символів 01 переходу зі стану 1 немає. У цьому випадку до вихідного коду дописуємо справа біти 000, що означатиме символи 010 у вхідному коді;

3) кодове слово закінчується символами 11, головку розміщено перед останнім символом 1, автомат перебуває у стані 2 і для останнього біта 1 переходу зі стану 2 немає. Тоді до вихідного коду дописуємо справа біти 011, що означатиме символи 10 у вхідному коді і перехід у стан 1;

4) автомат зупиняється у стані 3. Щоб визначити, який перехід було здійснено в цей стан: A чи B, до кодового слова дописується послідовність 000, якщо відбувся перехід A, і 011, якщо B. Ці послідовності відповідають переходам у стан 1 і, таким чином, завжди, коли автомат мав би завершувати роботу стані 3, він її завершуватиме у стані 1.

Отже, фактично всі особливі випадки обробляються через дописування фіктивного нуля до нижнього (2,3)-коду. Насправді кодове слово нижнього (2, 3)-коду не може закінчуватися символом 0 і, згідно з наведеним нижче алгоритмом декодування, цей фіктивний 0 ігноруватиметься.

За достатньо великої довжини вхідного кодового слова (починаючи від кількох сотень бітів) довжина завадостійкого нижнього (2, 3)-коду перевищує довжину вхідного двійкового повідомлення в 1,75 разів у

середньому. Якщо замість трійок кожен перехід кодувати парами бітів, то таке перевищення становитиме лише 1,165 разів у середньому, що майже не відрізняється від аналогічного показника для нижнього (2, 3)-коду за значно вищої завадостійкості.

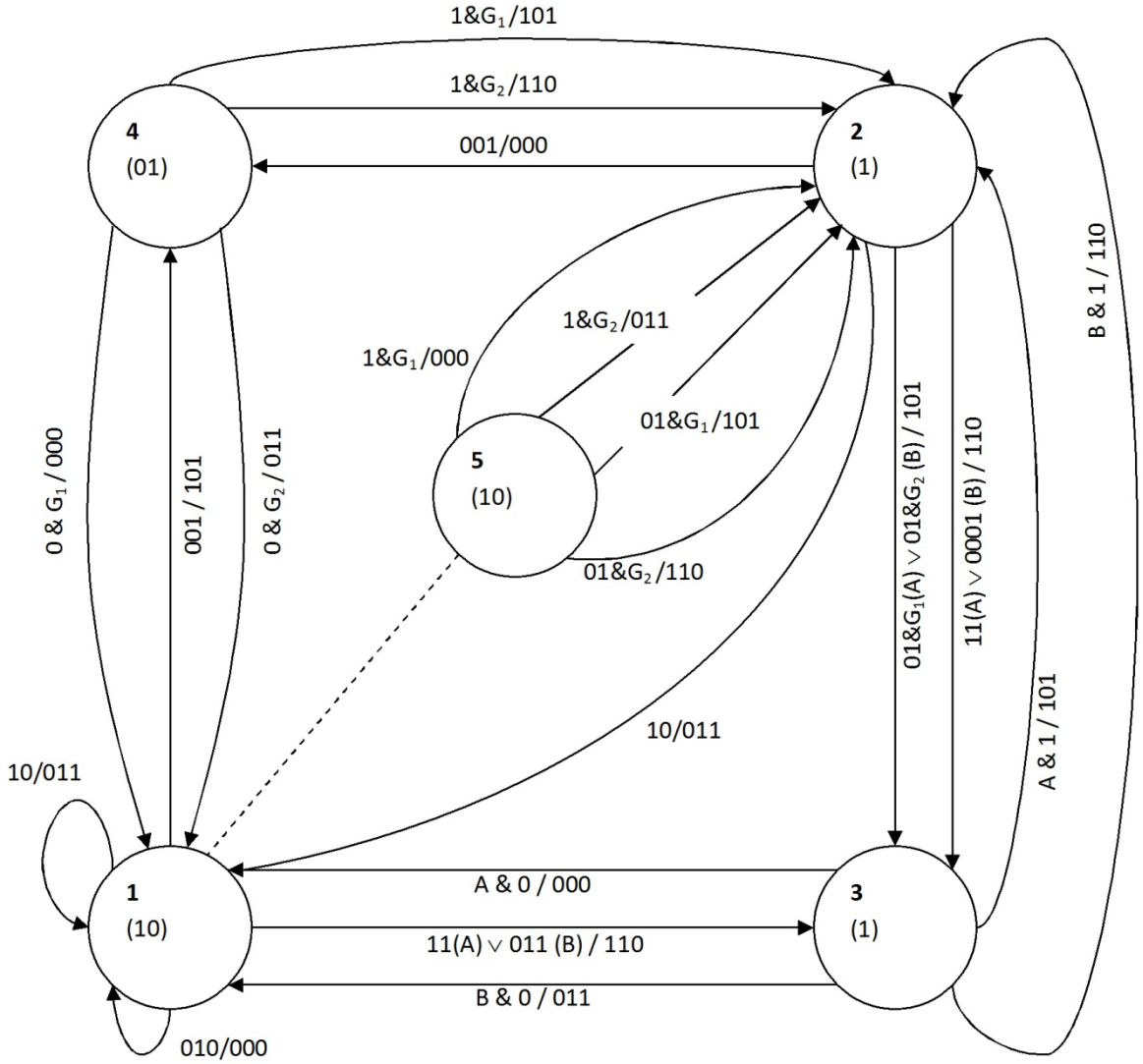

Рис. 1. Скінченний автомат для стискання модифікованого нижнього (2, 3)-коду

Приклад. Побудуємо завадостійкий (2, 3)-код числа 1387. Його нижній (2, 3)-код було побудовано у прикладі з розділу 2 і він становить 01001000100011. Отримана при цьому послідовність значень x_i становила 1387, 377, 83, 17, 1.

1. Переписуємо (Δ, k)-групи нижнього (2, 3)-коду справа наліво: 00011000100101.

2. Розглядаємо першу групу 00011. Тут $k = 2$, $\Delta = 0$, і декодування починаємо з числа $x = 1$. $3^k x = 3^2 \cdot 1 = 9$. Це число задовольняє нерівності (1), а отже автомат починає роботу зі стану 1.

3. Автомат пропускає перший символ коду і, починаючи з другого символу, зчитує групу 001, що означає перехід у стан 4. У вихідний код записуються символи 101.

4. Автомат зчитує п'ятий символ коду – 1 і переходить у стан 2. Оскільки $x \bmod 20 = 9 \in G_2$, то перехід відбувається за умовою 1 & G_2 і у вихідний код записуються символи 110.

5. Групу 00011 повністю оброблено, тому $x \leftarrow 17$.

6. Перебуваючи у стані 2, автомат зчитує символи 0001, переходить у стан 3 за умовою (B) і записує у вихідний код символи 110.

7. Групу 0001 повністю оброблено, тому $x \leftarrow 83$.

8. Перебуваючи у стані 3, автомат зчитує символ 0 і тому має перейти у стан 1 або 5. Поточна (Δ, k)-група – це 001, звідки отримуємо $k = 1$. Значення $3^k x = 3^1 \cdot 83 = 249$ задовольняє нерівностям (2), і тому автомат переходить у стан 5. Цей перехід виконується за умовою B & 0, оскільки перехід у стан 3 відбувся за умовою (B). Тому автомат записує у вихідний код символи 011.

9. Перебуваючи у стані 5, автомат зчитує символи 01 і переходить у стан 2. Оскільки $x \bmod 20 = 3 \in G_1$, то перехід відбувається за умовою 01 & G_1 і у вихідний код записуються символи 101.

10. Групу 001 повністю оброблено, тому $x \leftarrow 377$.

11. Автомат зчитує символи 01 і переходить у стан 3. Оскільки $x \bmod 20 = 17 \in G_2$, то перехід відбувається за умовою 01 & G_2 (В) і у вихідний код записуються символи 101.

12. Нижній (2, 3)-код повністю оброблено, але автомат закінчив роботу у стані 3. Оскільки перехід у цей стан відбувся за умовою (В), до коду додаються символи 011.

Отже, завадостійким нижнім (2, 3)-кодом числа 1387 буде послідовність 101 110 110 011 101 101 011.

3. Алгоритм кодування

На вході маємо послідовність бітів, а на виході потрібно отримати завадостійкий (2, 3)-код. Вхідну двійкову послідовність будемо ділити на блоки довжиною L бітів. До кожного блоку дописуватиметься контрольна сума довжиною c бітів, потім блок перетворюватиметься на число з множини $\mathbf{N}_{2,3}$ і для нього будуватиметься завадостійкий (2, 3)-код. Контрольна сума використовуватиметься під час декодування як критерій відбору серед кількох варіантів декодованих слів потрібного.

1. Ділимо вхідну послідовність бітів на блоки довжиною L.

2. Додаємо до кожного блоку c-бітну контрольну суму, i-й біт якої може бути обчислено як суму за модулем 2 всіх бітів блоку, номери яких дорівнюють $i \pmod q$.

3. Отриману $(L+c)$-бітну послідовність перетворюємо на число з множини $\mathbf{N}_{2,3}$ за описаним далі алгоритмом.

4. Для отриманого числа будуємо нижній (2, 3)-код, переписуємо послідовність його блоків справа наліво і за нею будуємо завадостійкий (2, 3)-код.

5. Отримані завадостійкі коди блоків конкатенуються без додавання жодних роздільників.

Тепер опишемо детально, як виконується крок 3 цього алгоритму: перетворення довільної послідовності довжиною M бітів на ціле число, що є взаємно простим із 2 і 3.

1. Якщо послідовність починається із символу 1 і являє собою взаємно просте з 2 і 3 двійкове число, зупиняємось.

2. Додаємо символ 1 до послідовності зліва. Якщо отримано число, яке взаємно просте з 2 і 3, зупиняємось.

3. Додаємо символ 1 до послідовності справа. Якщо отримано число, яке взаємно просте з 2 і 3, зупиняємось.

4. Додаємо символ 1 до послідовності справа.

Зворотна процедура перетворює число з множини $\mathbf{N}_{2,3}$ на вхідну послідовність бітів. Вона може завершитися помилкою, яка означатиме, що це число не може бути отримане в результаті прямого перетворення M-бітної послідовності.

1. Якщо бітова довжина числа менша за M або більша за $M+3$, завершуємо процедуру з помилкою.

2. Якщо бітова довжина числа дорівнює $M+3$, видаляємо найменш значущий біт. Якщо отримане число взаємно просте з 2 і 3, завершуємо процедуру з помилкою.

3. Якщо бітова довжина отриманого на попередньому кроці числа дорівнює $M+2$, видаляємо найменш значущий біт. Якщо отримане число взаємно просте з 2 і 3, завершуємо процедуру з помилкою.

4. Якщо бітова довжина отриманого на попередньому кроці числа дорівнює $M+1$, видаляємо найбільш значущий біт.

4. Алгоритм декодування

Декодувальний алгоритм застосовний до завадостійкого (2, 3)-коду, отриманого за наведеним у попередньому розділі алгоритмом. Якщо у кожну трійку бітів із номерами $3m+1$, $3m+2$ і $3m+3$ внесено не більше однієї помилки, цей алгоритм теоретично дає змогу виправити всі такі помилки. Однак якщо помилок буде забагато, або вони будуть скупчені, час роботи алгоритму може виявитися завеликим або може зрости ймовірність хибного виправлення – ситуації, коли за результат виправлення прийматиметься послідовність, що насправді містить помилки.

Припустимо, у деякі біти кодового слова внесено помилки, тобто ці біти інвертовані. Множину позицій бітів, які ми вважаємо помилковими, називатимемо ***маскою помилок***. Нехай $q = 3m+1$ – це номер першого біта деякої тріади кодового слова, $\nu = (\nu_1, \ldots, \nu_t)$, $\nu_1 < \ldots < \nu_t$ – певна маска помилок, а $x \in \mathbf{N}_{2,3}$ – результат декодування частини кодового слова, що починається з першого (найбільш значущого) біта та закінчується бітом q, у якій біти $\nu_1, \ldots, \nu_t$ інвертовано (інакше кажучи, x – це найбільше з чисел x_i, отримуваних після повної обробки (Δ, k)-груп в частині слова, що містить біти з 1-го по q-й). Крім того, через a позначимо номер

стану автомата, що відповідає положенню головки q. Четвірку (ν, x, q, a) називатиметься ***коригувальною конфігурацією***.

Якщо після застосування маски ν трійка бітів, що починається з біта q, не містить помилки, визначено операцію ***інкременту*** (ν, x, q, a)++, результатом якої є конфігурація $(\nu, x', q+3, a')$, де x' – результат декодування частини кодового слова, що починається з першого біта та закінчується бітом $q+3$, після інвертування бітів, номери яких належать масці ν, а a' – стан автомата, що відповідає положенню головки $q+3$. Якщо маска ν не відповідає справжньому положенню помилок у кодовому слові, операція (ν, x, q, a)++ може завершитися ***помилкою*** з таких причин:

1) $a=4$ або $a=5$, а частиною умови переходу, який визначається бітами $q, q+1, q+2$, є G_1, хоча $G(x)\in G_2$ або ж частиною умови переходу є G_2, хоча $G(x)\in G_1$;

2) $a=3$, а перехід у стан 3 відбувся зі стану 2 за умовою $01\&G_1$, хоча $G(x)\in G_2$, або ж за умовою $01\&G_2$, хоча $G(x)\in G_1$. Нагадаємо, що саме за значенням бітів з номерами $q, q+1, q+2$ визначається, за якою саме умовою відбувся перехід у стан 3, під час якого було переведено головку в позицію q.

Якщо для певної коригувальної конфігурації (ν, x, q, a) трійка бітів, що починається з позиції q, містить помилку, визначено операцію ***множення*** (ν, x, q, a)*. Її результатом є множина всіх можливих коригувальних конфігурацій (ν', x, q, a')++, де ν' – це маска, утворена з маски ν додаванням одного з бітів $q, q+1, q+2$, і операція інкременту не завершується помилкою. Тобто щонайбільше таких конфігурацій буде 3, а якщо деякі з операцій (ν', x, q, a)++ завершуються помилкою, у множині (ν, x, q, a)* буде менше трьох конфігурацій.

Операцію множення (ν, x, q, a)* визначимо також і для того випадку, коли трійка бітів, що починається з позиції q, не містить помилок. Вважатимемо, що в цьому разі результат множення збігається з результатом інкременту (ν, x, q, a)++, якщо він виконується коректно, та є порожньою множиною конфігурацій, якщо інкремент завершується помилкою.

Множина P всіх можливих коригувальних конфігурацій утворюється за наведеним далі ітеративним алгоритмом.

1. Утворюємо початкову множину коригувальних конфігурацій:

а) оскільки автомат може починати роботу або в стані 1, або в стані 5, покладаємо $P_1 \leftarrow (\{\}, x_t, 1, 1)$* і $P_5 \leftarrow (\{\}, x_t, 1, 5)$*, де $x_t = 1$ or $x_t = 2$. Як вибирати між цими двома початковими значеннями x_t, описано в розділі 2. З урахуванням особливостей кодувального автомату видно, що $x_t = 2$ тоді й тільки тоді, коли автомат починає роботу в стані 1 і першою трійкою бітів є 000. Якщо перша трійка бітів містить помилку, то залежно від того, який саме біт ми припускатимемо помилковим, значення x_t можуть відрізнятися;

б) переглядаємо всі конфігурації з множини P_1. Якщо першу (Δ, k)-групу для певної конфігурації c ще необроблено повністю, замінюємо конфігурацію p на множину конфігурацій p*. Якщо першу (Δ, k)-групу для конфігурації c оброблено повністю, видаляємо конфігурацію p із множини P_1 і, якщо величина $3^{k_t-1}x_t$ задовольняє нерівності (1), включаємо цю конфігурацію до результуючої множини конфігурацій P;

в) процедуру, аналогічну до описаної в п. б), застосовуємо до множини P_5. У цьому разі перевіряємо відповідність значення $3^{k_t-1}x_t$ нерівності (2);

г) повторюємо кроки б) і в) доти, доки множини P_1 і P_5 не стануть порожніми.

2. Переглядаємо всі конфігурації $p=(\nu, x, q, a)\in P$. Для кожної з них можливі такі варіанти:

а) бітова довжина числа x менша за $L+c$. Тут L – довжина блоків, на які ділиться вхідна послідовність, а c – бітова довжина контрольної суми. У цьому випадку замінюємо конфігурацію p на множину конфігурацій p *;

б) бітова довжина числа x дорівнює $L+c$. Тоді, можливо, x – це шукана вхідна послідовність. Щоб перевірити це, потрібно обчислити на перших L бітах x контрольну суму і порівняти її з останніми c бітами. Якщо вони збігаються, то вважаємо бітове подання x декодованою послідовністю і переходимо до декодування наступного кодового слова, що починається з біта q, якщо ні – то замінюємо p на p*. Крім того, $(L+c)$-бітне число x не може бути коректно декодованим, якщо воно не взаємно просте з 2 і 3 або якщо автомат перебуває у стані 1, а попереднім станом не був стан 3. У цих випадках також замінюємо p на p* і можемо навіть не перевіряти контрольну суму;

в) бітова довжина числа x більша за $L+c$, але менша за $L+c+4$. Тоді якщо a = 1, а попереднім станом не був стан 3, конфігурацію p з множини P видаляємо, інакше застосовуємо до числа $x \in \mathbf{N}_{2,3}$ описане в попередньому розділі зворотне перетворення на бітову послідовність і перевіряємо контрольну суму. Якщо всі ці процедури і перевірки завершилися успішно, вважаємо отриману послідовність шуканою і переходимо до декодування наступного числа, інакше конфігурацію p з множини P видаляємо;

г) бітова довжина числа x перевищує $L+c+3$. У цьому випадку конфігурацію p з множини P видаляємо.

3. Повертаємося на крок 2.

Цей алгоритм завжди завершуватиме свою роботу на кроці 2б) або 2в), коли ми припускатимемо, що певна коригувальна конфігурація відповідає справжньому розташуванню помилкових бітів.

Висновки

У запропонованому методі кодування поєднується три підходи до побудови завадостійких кодів, і всі вони мають принципово різну природу: нижній (2, 3)-код базується на арифметичних властивостях чисел, що подаються вхідними бітовими послідовностями, скінченний автомат дає змогу виявляти помилки завдяки наявності спеціальних «хибних» переходів між станами, у які можуть «завести» лише хибні значення бітів і, нарешті, точне місцезнаходження помилок визначається завдяки застосуванню одного з найпростіших блокових кодів, що кодує пари бітів трибітними кодовими словами із множини з Хемінговою відстанню 2: {000, 011, 101, 110}. Таке комбінування різнорідних підходів дає змогу досягти достатньо високої ефективності у виправленні помилок. Чисельний експеримент, у якому вхідна послідовність бітів ділилася на блоки довжиною 64 біти і довжина контрольної суми добиралася так, щоб середня швидкість коду становила 0,5 (тобто довжина кодового слова перевищувала довжину вхідної послідовності в середньому вдвічі), показав, що розглянутий вище алгоритм декодування завадостійкого (2, 3)-коду дає змогу з ймовірністю 99,8% виправляти 10 помилок у кодовому слові, у той час як, наприклад, широко відомий згортковий код NASA (171,133), що має ту саму швидкість, із зазначеною ймовірністю у кодовому слові зазначеної довжини дає змогу виправляти лише 4 помилки. Щоправда, завадостійкий (2, 3)-код має таке обмеження, що серед бітів із номерами 3m+1, 3m+2 і 3m+3 має бути не більше одного помилкового. Для усунення цього обмеження завадостійкий (2, 3)-код також може бути перетворено спеціальним скінченним автоматом, який зараз досліджується.

1. *Johannesson R., Zigangirov K.* Fundamentals of convolutional coding. – IEEE Press, 1999. – 400 p.
2. *Anisimov A.V.* Prefix Encoding by Means of the 2,3–Representation of Numbers // IEEE Transactions on Information Theory. – 2013. – Vol. 59. – N 4. – P. 2359–2374.
3. *Анисимов А.В., Завадский И.А.* Помехоустойчивое префиксное кодирование на основе нижнего (2,3)-представления чисел // Кибернетика и системный анализ. – 2014. – № 2. – С. 3–14.