УДК 519.72

Завадський І. О.

ВИКОРИСТАННЯ СКІНЧЕННИХ АВТОМАТІВ У ЗАВАДОСТІЙКОМУ КОДУВАННІ

У статті описано новий метод завадостійкого кодування, у якому повідомлення, що кодуються, розглядаються як цілі числа та подаються у двобазисній системі числення, а потім кодуються за допомогою скінченного автомата. Запропонований код вирізняється вищою завадостійкістю порівняно з широковідомими згортковими та блоковими кодами.

Ключові слова: скінченний автомат, завадостійке кодування, двобазисна система числення, (2,3)-код.

Хоча в теорії завадостійкого кодування скінченні автомати використовуються насамперед як засіб подання діаграм станів згорткових кодів, насправді можливості їх застосування значно ширші. Загальний підхід до кодування за допомогою скінченних автоматів полягає в тому, що автомат здійснює переходи з одного стану в інший залежно від зчитаних символів вхідного повідомлення та записує під час цих переходів певні символи у вихідний код. Завадостійкість отримуваного в такий спосіб коду може забезпечуватися, по-перше, завдяки тому, що автомат записує більше символів, ніж зчитує, а по-друге, завдяки тому, що автомат може містити «пастки» – стани, що недосяжні в разі відсутності помилок у кодовому слові, але в які автомат може потрапити, якщо такі помилки є. Можуть бути й складніші пастки – умови, що пов'язують стан автомата, зчитувані символи та здійснюваний перехід у певному предикаті, який в разі відсутності помилок буде істинним, і хибність якого свідчитиме про наявність помилок. У методі завадостійкого кодування, що описаний у даній статті, поєднано ці підходи. Крім того, перш ніж бути обробленим скінченним автоматом, вхідне повідомлення перетворюється на так званий ***нижній (2,3)-код***, математичною основою якого є подання чисел у двобазисній системі числення з основами 2 і 3. Це надає коду додаткової завадостійкості.

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. Обчислювальний експеримент показує, що в середньому довжина (2,3)-коду числа перевищує довжину його двійкового подання в 1,13 раза. 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 [1] 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 в 1,16 раза у середньому.

Основна ідея методу кодування така. Нехай $\mathrm{N}_{2,3}$ – множина натуральних чисел, які є взаємно простими із 2 і 3, $x \in \mathrm{N}_{2,3}$, $x > 1$, $n = \lfloor \log_2 x \rfloor$. Тоді x можна подати у формі $2^{n-1} + 3^k x_1$ або $2^{n-2} + 3^k x_1$, де $k \in \mathrm{N}$, $x_1 \in \mathrm{N}_{2,3} \cup \{2\}$, $x_1 < x$ 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 $\Delta_i = \lfloor \log_2 3^{k_i} x_{i+1} \rfloor - n_i$ і k_i, які отримуємо на кожній ітерації. Справді, якщо відомі значення x_{i+1}, Δ_i і k_i, можна обчислити величину $m_i = \lfloor \log_2 3^{k_i} x_{i+1} \rfloor$, 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: $x_t, \ldots, x_0$, де $x_t = 1$ або $x_t = 2$. Для вибору одного з двох початкових значень слід зауважити, що коли $x_{t-1} = 7 = 2^0 + 3^1 \cdot 2$, то $x_t = 2$, $b_i = 0$, $k_{t-1} = 1$, $m = \lfloor \log_2 3^k x_1 \rfloor = 2$, $\Delta_{t-1} = m - b = 2$. 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 $\Delta_{t-1} = 2$, то $x_t = 2$, а інакше – $x_t = 1$.

Нехай $x = 2^b + 3^k x_i$, $m = \lfloor \log_2 3^k x_i \rfloor$, $\Delta = m - b$. Величина Δ має важливу властивість: вона може набувати лише значень 0, 1 або 2, якщо

$$2^m < 3^k x_1 \le \frac{7}{8} 2^{m+1} \qquad (1)$$

і лише значень 0 або 1, якщо

$$\frac{7}{8} 2^{m+1} < 3^k x_1 < 2^{m+1} \qquad (2)$$

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 обрано з метою скоротити середню довжину коду.

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 тією ознакою, що значення величини Δ не відповідають нерівностям $0 \le \Delta \le 2$ у випадку (1) або $0 \le \Delta \le 1$ у випадку (2). У [1] описано алгоритм, що дає змогу виправляти одну помилку в кодовому слові нижнього (2,3)-коду з імовірністю 100 % і 2 помилки з ймовірністю 87,5 %. Це достатньо низькі показники. Однак завадостійкість нижнього (2,3)-коду можна суттєво підвищити завдяки його обробці зображеним на рис. 1 скінченним автоматом. Назвемо результуючий код на виході цього автомата ***завадостійким нижнім (2,3)-кодом***.

Автомат має головку, яка зчитує символи нижнього (2,3)-коду зліва направо. У автомата є 3 стани, позначених кружками. Перша цифра у кружку – це номер стану, а після неї в дужках записано символи, що розташовані перед головкою автомата, якщо він перебуває в цьому стані. Коли головка зчитує символи, стан автомата змінюється залежно від символів, зчитаних головкою, а також деяких інших умов; ці символи та умови зазначено над стрілками переходів перед скісними рисками. Під час переходу в код на виході записуються символи, які вказано над стрілками переходів після скісних рисок.

Умови переходів можуть бути двох типів. По-перше, це умови, пов'язані з тим, що деякі переходи автомат виконує залежно від значення певної хеш-функції, застосовної до величини x_i, 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). По-друге, умовою може бути виконання нерівностей (1) або (2) для величини $3^k x_i$. Ці умови над стрілками переходів позначено як (1) і (2).

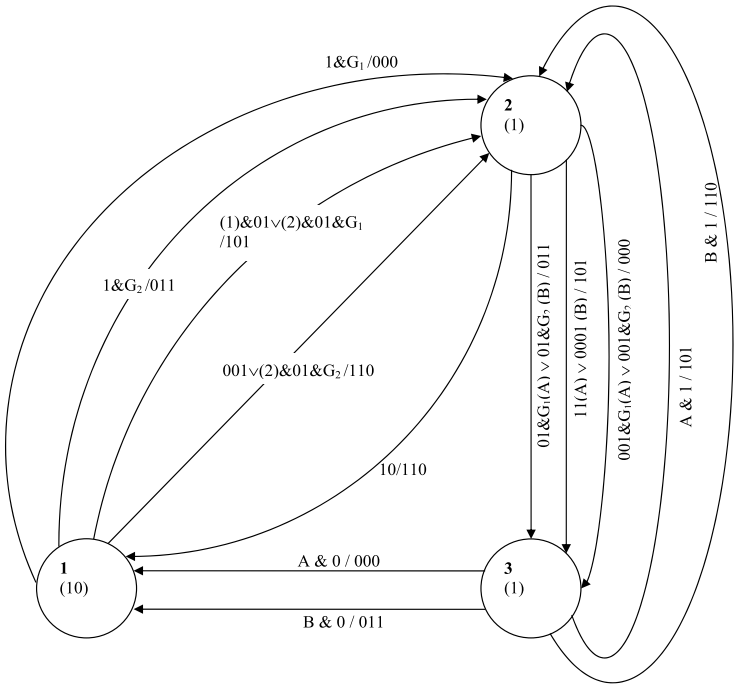

Рис. 1. Скінченний автомат, що генерує завадостійкий код

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***. Через причини, 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).

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 000.

Автомат починає роботу у стані 1. 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 із Хемінговою відстанню 2: {000, 011, 101, 110}. Таким чином, якщо у вихідному коді в трійку бітів з номерами $3k$+1, $3k$+2 і $3k$+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) Автомат зупинився у стані 3. Тоді він має дописати до кодового слова справа один біт, що визначатиме, за якою умовою, A чи B, було виконано перехід в останній стан.

2) 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. У цьому випадку до вихідного коду дописуємо справа біти 110, що відповідатимуть символам 10 у вхідному коді й переходу в стан 1. Останній символ 0 під час декодування 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 1,875 раза у середньому.

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 бітів.

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

3) Якщо отримана $(L + c)$-бітна послідовність не є числом з множини $\mathbf{N}_{2,3}$, перетворюємо її на таке число. Для цього дописуємо спочатку символ 1 зліва. Якщо число з $\mathbf{N}_{2,3}$ не отримано, дописуємо символ 1 справа. Якщо отримане число все ще не є взаємно простим із 2 і 3, дописуємо справа ще один символ 1.

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.

Зазначимо, що перетворення числа з множини $\mathbf{N}_{2,3}$ на вхідну послідовність бітів, тобто зворотна процедура до тієї, що описана на кроці 3, виконується так.

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.

Завершення процедури з помилкою означає, що це число не може бути отримане в результаті прямого перетворення M-бітної послідовності.

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.

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, $v = (v_1, \dots, v_t)$, $v_1 < \dots < v_t$ – 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 $v_1, \dots, v_t$ 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. Четвірка (v, x, q, a) називатиметься ***коригувальною конфігурацією***.

Якщо після застосування маски v трійка бітів, що починається з біта q, не містить помилки, визначено операцію ***інкременту*** (v, x, q, a)++, результатом якої є конфігурація $(v, x', q+3, a')$, де x' – результат декодування частини кодового слова, що починається з першого біта та закінчується бітом $q+3$, після інвертування бітів, номери яких належать масці v, а a' – стан автомата, що відповідає положенню головки $q+3$. Якщо маска v не відповідає справжньому положенню помилок у кодовому слові, операція (v, x, q, a)++ може завершитися ***помилкою***, оскільки частиною умови переходу є G_1, хоча $G(x) \in G_2$ або навпаки.

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

Якщо трійка бітів, що починається з позиції q, не містить помилок, результатом операції множення (v, x, q, a)* вважатимемо результат інкременту (v, x, q, a)++, якщо він виконується коректно, та порожню множину конфігурацій, якщо інкремент завершується помилкою.

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

1. Утворюємо початкову коригувальну конфігурацію $p \leftarrow (\text{“”}, x_t, 1, 1)$ та початкову множину коригувальних конфігурацій $P \leftarrow \{p\}$. Тут $x_t = 1$ або $x_t = 2$ і принцип вибору одного з цих початкових значень описано в частині 1.

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

a) 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*;

b) бітова довжина числа x більша або дорівнює $L + c$, але менша за $L + c + 3$. Тоді, можливо, x – це шукане число. Щоб перевірити це, потрібно застосувати до числа $x \in \mathbf{N}_{2,3}$ описане в попередньому розділі зворотне перетворення на бітову послідовність, обчислити на перших L бітах x контрольну суму і порівняти її з останніми c бітами. Якщо згадане перетворення завершується коректно, а контрольні суми збігаються, то вважаємо двійкове подання x декодованою послідовністю і переходимо до декодування наступного кодового слова, що починається з біта q, якщо ні – то замінюємо p на p*;

c) бітова довжина числа x дорівнює $L + c + 3$. Тоді застосовуємо до числа $x \in \mathbf{N}_{2,3}$ описане в попередньому розділі зворотне перетворення на бітову послідовність і перевіряємо контрольну суму. Якщо ці дії завершилися успішно, вважаємо отриману послідовність шуканою і переходимо до декодування наступного числа, інакше конфігурацію p з множини P видаляємо;

d) 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 2b) або 2c), 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 Хеммінговою відстанню 2. 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, широковідомий згортковий код NASA (171,133), що має ту саму швидкість, із зазначеною ймовірністю у кодовому слові зазначеної довжини дає змогу виправляти лише 3 помилки. Однак завадостійкий (2,3)-код має й недолік: серед бітів із номерами $3m+1$, $3m+2$ і $3m+3$ повинно бути не більше одного помилкового. Для усунення цього недоліку переходи зображеного на рис. 1 автомата можна кодувати не трьома бітами, а двома, після чого до отриманого коду застосувати згортковий код зі швидкістю 2/3. Такий підхід наразі досліджується.

Список літератури

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

I. Zavadskyi

USING FINITE-STATE AUTOMATONS IN ERROR CORRECTION ENCODING

The new method of error correction encoding is introduced. Input message is considered as the whole number, represented in the two-base numeral system and encoded using finite-state automaton. The proposed code has better error correction capabilities than well known convolutional and block codes.

Keywords: finite automaton, error-correcting encoding, two-base numeral system, (2,3)-code.

Матеріал надійшов 04.04.2014