УДК 512.64

Р.В. Скуратовський, Є.О. Осадчий

Методи таймерного шифрування

Разработаны статистически ориентированные методы сжатия данных с применением нетрадиционного таймерного шифрования, используемые для текстов определенной отрасли знаний совместно со скоростными таймерными счетчиками.

The statistically oriented methods of a data compression using a non-traditional timer encryption are developed. The methods are used for the texts of a particular branch of knowledges with speed timer counters.

Розроблено статистично орієнтовані методи стиснення даних із застосуванням нетрадиційного таймерного шифрування, які використовуються для текстів визначеної галузі знань сумісно з швидкісними таймерними лічильниками.

Вступ. При архівуванні даних очікування результату іноді затримується. Наприклад, якщо обсяг даних перевищує 1 Гб, то методи стиснення, що застосовуються в програмі *RAR*, будуть працювати на доступному персональному комп'ютері не менш, ніж одну годину. Важливою характеристикою архівування вважається обсяг стислої інформації. Тому цільовою функцією може бути $F = k_1x + k_2y \to \min$, де Y – час, X – обсяг (адитивна форма) $X \le Q_1$, $Y \le t_1$.

Маємо лінійне зростання обсягу виділеної пам'яті в залежності від кількості таймерних (часових) міток x, оскільки кожній мітці відповідає 8 біт пам'яті, а $y = t(\mu_\Sigma)$ є системою міток даного тексту.

Об'єднання текстів, які відповідають розкодованим міткам, ідентичним всьому тексту, дає швидкість розархівування міток більше, ніж s_0. Для порівняння меж стиснення нагадаємо, що при символьному кодуванні з однозначним декодуванням, тобто в префіксному кодуванні, існує код з найменшою довжиною коду символу $L(c,\xi)$: $H(\xi) \le L(c,\xi) \le H(\xi)+1$, де – $H(\xi)$ – це ентропія джерела, тобто його нижня межа стиснення. $H(\xi) = \sum_{i=1}^{n} p_i \log_2\left(1/p_i\right)$, або середньо очікувана кількість інформації, яку несе один символ. Зрозуміло, що зі зростанням довжини тексту T довжина коду, наприклад *арифметичного*, буде лінійно зростати. Метод, що пропонується (у разі використання однієї мітки) використовує для неї 8 біт і пам'ять для зберігання алфавітного впорядкування символів відкритого тексту. При цьому довжина коду не росте зі збільшенням довжини тексту, а коли використовується система міток з періодом 3 Б, то навіть після досягнення нижньої межі для символьного кодування стиснення за Шеноном, можна досягти стиснення в 24 рази.

Саме тому в даній статті вперше запропоновано нові методи таймерного кодування, доведено їх ефективність, розроблено відповідне математичне забезпечення для оптимізації.

Основи таймерного шифрування

Символи тексту є відносні частоти f_i, за якими вони впорядковуються за спаданням частоти, чим забезпечують лінійний порядок. Нагадаємо, що генератор генерує символи в заданому порядку послідовно за розрядами. Час і результат його роботи однозначно визначаються таймерною міткою.

Визначення. Номером символу при заданому впорядкуванні $\wp$ є його порядковий номер n_i в цьому впорядкуванні, здійснюваному для статистичного генератора *Gs* згідно з ймовірностями, характерними для символів виділеного тексту. Зауважимо, що вагою символу буде саме його номер в частотному впорядкуванні, такому, що коли, $f_i > f_j$, то ну $n_i < n_j$. Позначимо символом A множисимволів c.

Визначення. Вагою слова W або тексту $T = a_1, a_2, a_3, \ldots, a_n$ назвемо величину

$$Wh(T) = \sum_{i=1}^{n} n_i \left|A\right|^{n-i}, \qquad (1)$$

де n_i – номер символу, що знаходиться на i-му місці тексту в заданому для генератора впорядкуванні, N – довжина тексту.

Поставивши помилково першу літеру, генератор змушений ставити на всіх позиціях після нього всі букви з A – множини символів, поки не дійде до величини часу, більшої, ніж таймерна мітка для T. Наприклад, якщо мітка для числа $T = 1567$ і генератор поставив перший символ 2 (що відповідає впорядкуванню 2, 3, 4, 5, 6, 7, 8, 9, 0, 1), то він перебиратиме всі цифри, спочатку послідовно збільшуючи символ у другому розряді, потім всі комбінації в наступних розрядах – так до набору $a_1 a_n a_n \ldots a_n$, поки не повернеться до першої позиції, де далі поставить наступну за порядком цифру *три* (яка теж помилкова) і знову почне послідовний перебір в молодших розрядах. Так буде, поки він не дійде в першому розряді до одиниці. Отже, в кожному розряді від *другого* до *четвертого* було використано від дев'яти до десяти цифр, а в *першому* розряді – всі 10 цифр. Тому верхня межа перебраних комбінацій при неправильно вибраному a_1 буде $n_1 \cdot 10^3$ для T. Якщо неправильно вибрано a_2 $n_2 \cdot 10^2$, то це $n_2 \cdot 10^2$ комбінацій і т.д. В результаті, вага дорівнює

$$Wh(1567) = \sum_{i=1}^{4} n_i |A|^{4-i},$$

$$n_1 = 10,\ n_2 = 4,\ n_3 = 5,\ n_4 = 6.$$

Оцінна величина перебору в разі неправильно поставленого a_1 становить $n_1 \cdot |A|^{N-1}$. При цьому величина $|A|^{N-1}$ є вагою або розрядністю позиції i-го символу. Взагалі величина $|A|^{N-i}$ дорівнює розрядності i-го символу. Відповідно $Wh(T)$ є комбінаторна складність гіперслова по $\wp$. Вона відповідає потужності множини гіперслів, що генеруються при обраному впорядкуванні символів, після якої слідує шукане T. А ймовірність виникнення слова T – це кумулятивна ймовірність

$$P(T) = p_1 p_2 \ldots p_n, \qquad (2)$$

де p_i – ймовірність появи i-го символу в даному тексті з даної *предметної галузі* знань (ПГ).

Множинна система міток

Зрозуміло, що перші символи мають більшу вагу. Саме тому доцільно сформувати окрему мітку (мітки) для початку тексту, але якщо мітки будуть розподілені нерівномірно, залишок тексту буде довгим і його перші символи (префікс) матимуть ваги з високими ступенями. Саме тому природною є гіпотеза про рівномірний розподіл міток. Але спочатку доведемо ефективність використання множинної системи міток за певних обмежень на їх кількість. Кожна мітка займає 1 Б, а сукупна їх кількість може бути обмежена потребами користувача. Підрахуємо виграш в числі операцій від застосування двох міток, відповідних префіксній множині з двох гіперслів. Окремі мітки розділяють текст рівномірно порівну. Нехай $|W| = N$. Якщо спочатку знаходимо префікс $Pr(W) = W_1$, де W_1 – початок гіперслова W, довжини $L(w_1) \approx \left[N/2 \right]$, відповідної таймерній мітці μ_0. Але якщо наступна частина слова, що генерується, не збігається з W, продовжимо генерувати її з $L+1$-го місця до кінця слова. Відповідно, наступну частину W_2 шукатимемо окремо, зафіксувавши вже знайдений префікс $Pr(W) = w_1$. Для них частотні ваги будуть наступні: $Wh(w_1) = \sum_{i=1}^{N-L} n_i |A|^{n-i}$, а для другої частини тексту $Wh(w_2) = \sum_{i=L+1}^{N-L} n_i |A|^{N-i}$. Тоді частотна вага всього слова при використанні переборного генератора буде $Wh(w) = \sum_{i=1}^{N} n_i |A|^{n-i}$, де $w = Pr(w) \cdot w_2$. Це скоротить перебір як мінімум в $|A|^{n/2-2}$ раз, оскільки ступінь знаменника менша в два рази

$$k = \frac{Wh(W)}{Wh(W_1) + Wh(W_2)} = \frac{\sum_{i=1}^{N} n_i |A|^{n-i}}{\sum_{i=1}^{N/2} n_i |A|^{n-i} + \sum_{i=N/2}^{N} n_i |A|^{n-i}}.$$

Лінгвістична аналітика

Якщо для даної ПГ знань ввести статистику не тільки частот символів, а ще й імовірностей

появи символу в слові, після того як визначено вже I символів в цьому слові, будемо мати умовні ймовірності. Саме ці ймовірності дозволять зменшити перебір. Наприклад, є кілька прикметників з однаковими закінченнями: статистичний, середньозважений, кореляційний, математичний, інтегральний. Тому при $l > 5$, при виникненні поєднання букв *ні*, логічно почати підбір наступного символу з букви *й*, не враховуючи, що його частота порівняно мала. Також при $l > 3$ доцільно враховувати можливість появи не існуючих комбінацій символів у мові даної галузі знань. Наприклад, не можуть з'являтися три голосних символи поспіль і інші поєднання. Це все формалізується за умовних імовірнісних переходів до наступного символу.

Префіксно-таймерне кодування. Методика обчислення ймовірностей

Розглянемо ймовірності і швидкості породження текстів статистично налаштованим генератором. Нехай Gs – статистично навчений генератор. Слід знайти ймовірність генерування слова. Якщо *aaa* – потрібне слово раніше, ніж *abab* – довільне слово. Розглянемо помічений граф виду, такого як на рис. 1 і 2 та відповідні ймовірності переходів.

$$\begin{cases} x_{\varnothing} = p(a)x(a) + p(b)x_{\varnothing} \\ x_{a} = p(a)x_{aa} + p(a)x_{aa} \\ x_{ab} = p(a)x_{aba} + p(b)x_{\varnothing} \\ x_{aba} = p(a)\cdot x_{aa} + p(b)x_{abab}, x_{abab} = 0 \\ x_{aa} = p(a)\cdot x_{aaa} + p(b)x_{ab}, \quad x_{aaa} = 1 \end{cases},$$

де x_{aa} – ймовірність прийти з одного стану *aa* шуканого слова *aaa* раніше, ніж до стану слова *abab*.

Визначення. Префікс-функцією по S від рядка L назвемо найдовший префікс з $Pr(S)$, яким закінчується рядок L, $L \in W$. Позначимо цю функцію як $PrfS(L)$.

Якщо є заборонені слова (що не є початком жодного з шуканих слів), то їх префікси додають у множину $Pr(S)$. Потрапляння під заборонене слово приводить до вершини $\varnothing$. Наприклад, в цьому ж наборі {*aaa*, *abab*} не має зустрічатися *bbb*, оскільки воно не є початком слів набору.

Якщо прийняти, що *abab* – текст з даної галузі знань і має мітки в створеній префіксній множині M, то є дійсним *паралельне розархівування двох текстів*. При цьому упорядкування здійснюється за середньостатистичними даними для множини текстів з даної ПГ.

Метод обчислення ймовірності

Імовірність переходу до шуканого слова при генеруванні дорівнює сумі добутків ймовірностей шляхів, за якими можна прийти в стан *aaa* з початкового стану $\varnothing$.

$P_{aaa} = \sum_{i=1}^{k} p_i(\varnothing,\ldots,aaa)$, де $p_i(\varnothing,\ldots,aaa)$ – ймовірність переходу i-м шляхом з $\varnothing$ в *aaa*.

Якщо *abab* – тестове середньостатистичне слово, для якого немає спеціальних міток підслів, то статистичний порядок букв виконується не для шуканого гіперслова, а для *aaa*. Зауважимо, що перехід з дописуванням букви *a* відбувається частіше, оскільки він є більш пріоритетним, тому ймовірність отримання слова *aaa* раніше *abab* буде більше половини, що доводить ефективність статистично зорієнтованого генератора. Середній очікуваний час $M(t(aaa)) << M(t(abab))$.

За великої довжини тексту в такий спосіб обчислювати ймовірність переходу нераціонально. Але з системи (1), що відповідає марківським ланцюгам, можна відносно легко обчислити ліву частину передостаннього рівняння, тобто x_{aba}, оскільки в його правій частині $x_{abab} = 0$ і $x_{aa} = p(a)$. Далі, підставляючи знайдене x_{aba} в інші рівняння, можна легко знайти ймовірності у всіх станах.

Твердження. Нехай порядок символів зафіксовано, тоді ймовірність генерації *методом спільного генерування* одного з слів більше, ніж *методом роздільного генерування*.

Це так тому, що кожне з w_0, w_1 має свою систему міток дільників слова і в процесі генерації більшого слова w_0 виникатимуть підслова, відповідні міткам для w_1 і навпаки. Тому недоцільно генерувати окремо слова w_1, оскільки їх побудова відбуватиметься одночасно. При цьому порядок букв для w_0, w_1 однаковий.

Архівування за множиною префіксів

Визначення. Дільником слова називається підслово u: $w = p \cdot u \cdot r$, де $p, r -$ підслово слова.

Таймерний лічильник запускається на початку генерації, але його значення, отримані в процесі генерації фіксуються в ОЗУ після кожної генерованої текстової одиниці дискретизації, яка може бути словом або реченням. Це необхідно для зіставлення з елементами множини $Pr(S)$. Вони є не ті й не інші, а відповідають префіксам і є дільниками тексту T.

У даному дослідженні з метою архівування методом таймерного кодування введемо спеціально позначену *таймерними мітками множину префіксів S*, відповідну підсловам тексту W. Серед цих міток обов'язково є повна мітка, відповідна всьому тексту.

Цілком природно, що закінчення гіперслова (узагальнений суфікс) або довільний дільник цього гіперслова може збігатися з підсловом шуканого гіперслова W, а перевірити це можна порівнянням міток цих підтекстів. Тому, маючи мітки певного набору дільників гіперслова W, можна за їх появи в процесі статистично зорієнтованого перебору істотно скоротити час роботи (кількість перебраних текстів до отримання закодованого тексту). Крім того, цілком природно для архівування брати кілька текстів з родинних ПГ.

Доведемо, що їх спільне розархівування є більш швидким. Нехай дано множину міток архівованого набору рядків S. Розглянемо множину можливих *префіксів* $Pr(S)$ – множини початків деяких підрядків (виділені префікси) для всіх рядків з S, включаючи власне рядки, а також префікс нульової довжини. Нехай кожному префіксу відповідає деяка вершина кореневого орієнтованого дерева, причому від вершини A йде стрілка в вершину B, тоді і тільки тоді, коли префікс, відповідний B, можна отримати з префікса, відповідного A, шляхом приписування до нього одного символу. Наприклад, для слів *abb* і *abab* можна взяти $Pr(S) = \{a, abb, abab\}$ (рис. 1). На дереві квадратами позначено префікси – члени $Pr(S)$. Біля стрілок вказано символи, які потрібно додати, щоб перейти до зазначеного префіксу. Тепер, якщо задано деяку мітку, відповідну загальній частині двох архівованих текстів, то слід окремо генерувати відповідне цій мітці гіперслово для обох текстів, що пришвидшує спільне розархівування.

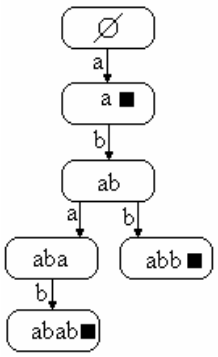

Рис. 1. Дерево пошуку префіксів

Якщо згенеровано гіперслово, то, щоб перевірити його наявність в S, потрібно, починаючи з мітки порожнього префікса, «пройти» по дереву пошуку (це траєкторія паралельного генерування з таймерним пошуком), кожного разу рухаючись по ребру, позначеному символами заданого набору S. Задане гіперслово буде частиною S тоді і тільки тоді, коли знаходиться відповідне ребро. В результаті опинимося на префіксі, позначеному квадратиком. За наявності набору S замість одного тексту, що генерується, можливих переходів по ребрам з тієї ж вершини (за умови, що ці переходи ведуть до одного з елементів з $Pr(S)$), зрозуміло, буде не менше, тому швидкість спільного розархівування буде більшою. Введемо наступну функцію.

Визначення. *Префікс-функцією* по S від рядка L назвемо довгий префікс з $Pr(S)$, яким закінчується рядок L, $L \in W$. Позначимо цю функцію як $PrfS(L)$. Оскільки порожній префікс належить $Pr(S)$, то для довільного рядка він буде «кандидатом» на роль значення префікс-функції. Тому для довільного рядка $PrfS(L)$ визначена.

У загальному випадку міткам відповідають довільні, заздалегідь обрані подільники гіперслова W. Вони утворюють множину S. Тому в процесі генерації, як тільки виникатиме підслово, відповідне елементу з множини S, воно перетворюватиметься в двійковий запис підслова. Після цього здійснюватиметься пошук продовжень отриманого підслова шляхом такої ж генерації символів і символів з урахуванням

ймовірності їх появи. Ці ймовірності враховано при визначенні порядку генерації. За подальшого дописування символів в $PrfS(p)$ можлива поява наступного префікса з S. Для цього слід вирахувати $PrfS(PC)$, який формально буде суфіксом нового підслова pc. Максимум $PrfS(PC) = pc$. Опишемо цей процес шляхом побудови спеціального графа переходів по префіксах p. У цьому графі стани – це множина виявлених дільників гіперслова W і початковий стан $\emptyset$.

Граф переходів за префіксами. Дерево пошуку підслів можна розвинути в нову структуру даних – граф переходів за префіксами з W, що дозволяє швидко відповісти на питання: *які гіперслова з S входять в генерований G, на яких місцях і скільки разів*? Ця структура також самоконсолідована. Її перебудова відбувається приблизно так, як і для дерева пошуку, та є більш довготривалою. Щоб краще уявити собі роботу з графом переходів за префіксами, нагадаємо визначення. Будуємо граф всіх переходів між префіксами, тобто для кожного префікса зазначаємо, яким він буде, коли до нього приписати взятий символ. Під символом розумітимемо дільник гіперслова, відповідний одиниці дискретизації таймерного лічильника. Граф будується так, щоб для довільного рядка L виконувалося наступне: якщо починати маршрут за графом від порожнього префікса і на кожному кроці слідувати за стрілкою з черговим символом з L, то в кінці маршруту отримаємо префікс, рівний $PrfS(L)$. Існування такого графа можливе, оскільки для довільних рядків x і y виконується

$$Prf_s(x) = Pr_s(y) \Rightarrow Prf_s(xc) = Pr_s(yc) \qquad (6)$$

для довільного символу c (або частини тексту, відповідної одиниці дискретизації даного тексту таймерним лічильником). Розглянутий символ приписано до рядків x і y. Для нашого набору S зобразимо граф переходів за префіксами (рис. 2).

Приймемо речення за одиницю дискретизації. На початку кожного з них запускається таймерний лічильник, а в кінці – зчитуються значення часу і порівнюються зі значеннями міток з $Pr(S)$. У разі знаходження відповідної мітки відшукується речення з тексту і запам'ятовується адреса знайденої мітки в ОЗУ. Опишемо процес побудови такого графа.

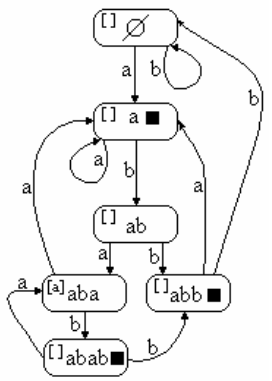

Рис. 2. Граф переходів за префіксами

Спочатку зафіксуємо на етапі, коли знайдено мітку для тексту довжиною n, дерево переходів або ж відзначимо його ребра як *основні*. Далі в дереві переходів, що відповідає формуванню тексту, з кожної знайденої вершини продовжимо визначати переходи по всіх можливих символах, а не виконувати побудову цього графа заново, як це було б у разі генератора з однією міткою. При такому генеруванні першим буде виконуватися перехід, відповідний більш пріоритетному символу.

Для довільного символу c і для довільного префікса p з $Pr(S)$ алгоритм шукає $PrfS(pc)$ і з префікса p направляє стрілку з символом c в префікс $PrfS(pc)$. Напрямок стрілок визначається динамічно, тобто з результатів для менших префіксів слідують результати для більших префіксів.

Індукція за довжиною префікса. Для порожнього префікса переходи обчислюються на основі дерева пошуку. У разі успіху відповіддю буде односимвольний префікс c, а в разі неуспіху – порожній префікс.

Припустимо, що для префіксів, довжиною, меншою, ніж визначена, всі переходи генеруються і підраховуються. На цьому етапі можна застосувати розпаралелювання після знаходження префікса $p \in Pr(S)$. Підрахуємо переходи для всіх префіксів даної довжини. Перевіримо присутність pc в $Pr(S)$ за допомогою дерева пошуку у тексті G. Для цього генератор дописує символи c до наявного префіксу. Це можна

виконати за один крок, коли зберігається місце знаходження таймерної мітки, що встановлюється таймерним лічильником, який працює паралельно початку префікса p в тексті. При паралельній роботі кількох генераторів зберігаємо місце знаходження таймерних міток всіх префіксів в дереві породжених послідовностей. У разі успіху направляємо стрілку в pc. У разі неуспіху, в силу (1) буде виконано $Prf_s(pc) = Prf_s(p'c)$, де p' виходить з p відкиданням першого символу (далі використовується це позначення і для інших рядків). $Prf_s(p'c)$ може бути отримано на основі вже частково побудованого графа переходів. Якщо зберегти всі адреси префіксів $Prf_s(p'c)$ (їх число лінійно залежить від обсягу S), то наступний $Prf_s(p'c)$ можна отримати за один крок, починаючи з $Prf_s(p'c)$. Текст, відповідний мітці $Prf_s(p')$, буде збережений, оскільки $p'=q's$, де q – префікс, попередній p в дереві пошуку, а s – останній символ префікса p. І нарешті, щоб не пропустити жодного слова, для кожного префікса p з $Pr(S)$, необхідно вказати $LwS(p)$ – найбільш довге слово з S, яке не збігається з p в разі його наявності. Це теж можна зробити динамічно. Для порожнього префікса LwS – відсутня. Оскільки $LwS(p)$ – кандидат на роль $PrfS(p')$, то $LwS(p)$ служить закінченням останнього. Тому потрібно перевірити належність $PrfS(p')$ множині S. Це виконується в один крок, оскільки на попередньому етапі збережено значення $Prf_s(p')$. Якщо $Prf_s(p')$ – в S, то $Lw_s(p)=Prf_s(p')$. В іншому випадку (див. рис. 2) – перехід за стрілкою a з aba, тобто нове вкладення префікса в отримане слово без дописування нового символу $Lw_s(p) = Lw_s(Prf_s(p'))$.

Останній вираз вже збережено, оскільки $Prf_s(p')$ коротший, ніж p. Алгоритм завершено. Він виконується за кількість кроків, лінійно залежних від обсягу S.

На рис. 3 в дужках для приставок вказані їх значення Lw_s. Насправді, потрібно зберігати посилання на них в графі, оскільки за цими посиланнями будуть здійснюватися переходи. Рухаючись графом, відзначаємо всі префікси, позначені квадратиками. При попаданні на префікси, які мають Lw_s, робимо перехід за посиланнями на Lw_s, поки це можливо, кожного разу відзначаючи нове входження деякого слова з S. Після переходу по таких посиланнях повернемося до того префіксу, на який потрапили останнім переходом за символом, а не за посиланнями Lw_S.

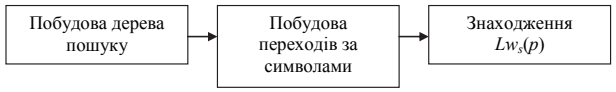

Рис. 3. План побудови графа переходів за префіксами

Проілюструємо дію алгоритму на прикладі. Нехай вводиться рядок *ababb*. Для неї черговість операцій алгоритму буде наступна:

0. Знаходимося на порожньому префіксі.

1. Перехід по *'a'* на *"a"*; фіксація входження *"a"* з кінцем в п. 1.

2. Перехід по *'b'* на *"ab"*.

3. Перехід по *'a'* на *"aba"*. Для цього здійснюється:

– перехід по Lw_s на *"a"*, фіксація входження *"a"* з кінцем в п. 3;

– повернення на *"aba"*.

4. Перехід по *'a'* на *"abab"*, фіксація входження *"abab"* з кінцем в п. 4.

5. Перехід по *'b'* на *"abb"*, фіксація входження *"abb"* з кінцем в п. 5.

За великої кількості варіантів символів можна виконати перехід по меншій одиниці дискретизації. Це дає зменшення необхідної пам'яті і числа кроків приблизно в $m/(2\log_2 m)$ раз, де m – кількість різних можливих підтекстів, які мають таку ж довжину, що і одиниця дискретизації d. Якщо взяти одиницю дискретизації $d = 3$ Б, то навіть після застосування методів символьного кодування отримаємо стиснення в $8*3 - 1 = 23$ рази. Це так тому, що один біт, йде власне на мітку. Але при цьому зі збільшенням величини d зростає час роботи. Алгоритм, який використовує граф переходів за префіксами, виконує кількість кроків, лінійно пропорційниу сумі розмірів введеного тексту. При цьому досягається теоретично передбачена максимальна ефективність таймерного кодування.

1. *Осадчий Є.*, *Осадчий О.* Проблеми створення, інтеграції і використання науково-технічної інформації на сучасному етапі // Тез. доп. VI міжнар. наук.-практ. конф. – Київ, 16–17 груд. 1999. – С. 354–355.

2. *Осадчий Є.О.* Підхід до покращення таймерних методів захисту інформації // Вісн. ТАНГ, Економіко-математичне моделювання. – 1999. – № 1. – С. 30–35.
3. *Осадчий Є.*, *Осадчий О.* Таймерні методи та засоби захисту інформації // Тез. доп. Міжнар. наук.-практ. конф. «Проблеми впровадження інформаційних технологій в економіці та бізнесі», Ірпінь, 2000. – С. 354–355.
4. *Методи* та засоби захисту інформації в часі / Є. Осадчий, В. Курченко, Д. Гриценко та ін. // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні: Зб. пр. – НТУУ «КПІ», 2000. – С. 73–76.
5. *Осадчий Є.О.* Трансформерні технології побудови машин та механізмів. – К.: Наук. світ, 2004. – 167 с.
6. *Таймерне* кодування в прогресивних інформаційних технологіях / Є.О. Осадчий, О.Є. Осадчий, В.В. Ткаченко та ін. // Тез. доп. 12 міжнар. конф. з автоматизації керування «Автоматика 2005» – К.: НТУ «КПІ», 2005. – Т. 3. – С. 55.
7. *Патент* Российской Федерации № 2128878, МКИ 6 Н 03 К 23/00. *N*-разрядный счетчик / Е.А. Осадчий, А.Е. Осадчий, (Украина), Опубл. 10.04.99; Бюл. № 10. – 16 с.
8. *Авт.* свид. СССР № 1275471, НКИ *G* 06 *F* 15/38, 9/44. Устройство для преобразования кодов с одного языка на другой / В.И. Корнейчук, А.П. Марковский, Е.А. Осадчий, В.С. Бабак (СССР); НИИПиП (СССР). – № 3926126/24–24; Заяв. 05.07.85; Опубл. 07.12.86, Бюл. № 45. – 8 с.
9. *Осадчий Е.А.* Об одном подходе к определению эффективности методов формального сжатия информации их информационной совместимости // УСиМ. – 1996. – № 5. – С. 43–56.
10. *Осадчий Є.О.*, *Осадчий В.Є.* Трансформерна технологія як приклад високої технології, що здатна продукувати конкурентоздатну продукцію // Рынок технологий, проблемы и пути решения: Тез. докл. междунар. науч.-практ. конф. – К.: УкрИНТИ. – 2002. – С. 204–209.
11. *Рішення* про видачу патенту України на винахід по заявці № 99095303 від 27.09.99, МКИ 6 Н 03 К 23/00. Асоціативний інерційний лічильник / Осадчий О.Є., Осадчий В.Є. – 14 с.

Поступила 07.02.2011
(после доработки 27.07.2012)
Тел. для справок: +380 63 203-1025, +380 99 327-0661, +380 67 501-7213 (Киев)
E-mail: *skeleton@unicyb.kiev.ua*

Р.В. Скуратовский, Е.А. Осадчий

Методы таймерного шифрования

Введение. При архивировании данных ожидание результата иногда затягивается. Например, если объем данных превышает 1 Гб, то методы сжатия, применяемые в программе *RAR*, будут работать на доступном персональном компьютере не менее одного часа. Важной характеристикой архивирования считается объем сжатой информации. Поэтому целевой функцией может быть

$F = k_1x + k_2y \to \min$, где Y – время, X – объем (аддитивная форма) при $X \le Q_1$, $Y \le t_1$.

Имеем линейный рост объема выделенной памяти в зависимости от количества таймерных (временных) меток x, так как каждой метке соответствует 8 бит памяти, а $y = t(\mu_\Sigma)$ – система меток данного текста. Объединение текстов, соответствующее раскодированным меткам, идентичным всему тексту, дает скорость разархивирования меток более чем s_0. Для сравнения пределов сжатия напомним, что при символьном кодировании с однозначным декодированием, т.е. в префиксной кодировке, существует код с наименьшей длиной кода символа $L(c, \xi)$: $H(\xi) \le L(c, \xi) \le H(\xi) + 1$, где $H(\xi)$ – энтропия источника, т.е. его нижняя граница сжатия.

$H(\xi) = \sum_{i=1}^{n} p_i \log_2\left(1/p_i\right)$, или среднее ожидаемое количество информации, которую несет один символ. Понятно, что с ростом длины текста T длина кода, например *арифметического,* будет линейно возрастать. Предложенный метод (в случае использования одной метки) использует для нее 8 бит и память для хранения алфавитного упорядочивания символов открытого текста. При этом длина кода не возрастает с увеличением длины текста, а когда используется система меток с периодом 3 Б, то даже после достижения нижней границы для символьного кодирования сжатия по Шеннону, можно достичь сжатия в 24 раза.

Именно поэтому в данной статье впервые предложены новые методы таймерного кодирования, доказана их эффективность, разработано соответствующее математическое обеспечение для оптимизации.

Основы таймерного шифрования

Символы текста имеют относительные частоты f_i, по которым они упорядочиваются по убыванию частоты, чем обеспечивают линейный порядок. Напомним, что генератор генерирует символы в заданном порядке последовательно по разрядам. Время и результат его работы однозначно определяются таймерной меткой.

Определение. Номером символа при заданном упорядочении $\wp$ есть его порядковый номер n_i в этом упорядочении, осуществляемом для статистического гене-

ратора *Gs* согласно вероятностям, характерным для символов выделенного текста. Заметим, что весом символа будет именно его номер в частотном упорядочении, таком, что если $f_i > f_j$, то $n_i > n_j$. Обозначим символом A множество символов с T.

Определение. Весом слова W или текста $T = a_1, a_2, a_3, ..., a_n$ назовем величину

$$Wh(T) = \sum_{i=1}^{n} n_i |A|^{n-i}, \quad (1)$$

где n_i – номер символа, который находится на i-м месте текста и в заданном для генератора упорядочении, N – длина текста.

Поставив ошибочно первую букву, генератор вынужден ставить на всех позициях после него все буквы с A – множества символов, пока не дойдет до величины времени, большей, чем таймерная метка для T. Например, если метка для числа $T = 1567$ и генератор поставил первый символ 2 (что соответствует упорядочению 2, 3, 4, 5, 6, 7, 8, 9, 0, 1), то он будет перебирать все цифры, сначала последовательно увеличивая символ во втором разряде, потом все комбинации в следующих разрядах – так до набора $a_1 a_n a_n \ldots a_n$, пока не вернется к первой позиции, где далее поставит следующую по порядку цифру *три* (которая тоже ошибочна) и снова начнет последовательный перебор в младших разрядах. Так будет, пока он не дойдет в первом разряде до единицы. Таким образом, в каждом разряде от второго до четвертого было использовано от девяти до десяти цифр, а в первом разряде – все 10 цифр. Поэтому верхняя граница перебранных комбинаций при неправильно выбранном a_1 будет $n_1 \cdot 10^3$ для T. Если неправильно выбрано a_2, $n_2 \cdot 10^2$, то это $n_2 \cdot 10^2$ комбинаций и т.д. В итоге, вес равен

$$Wh(1567) = \sum_{i=1}^{4} n_i |A|^{4-i},\ n_1 = 10,\ n_2 = 4,\ n_3 = 5,\ n_4 = 6.$$

Оценочная величина перебора в случае неправильно поставленного a_1 составляет $n_1 \cdot |A|^{N-1}$. При этом величина $|A|^{N-1}$ будет весом или разрядностью позиции i-го символа. Вообще, величина $|A|^{N-i}$ равна разрядности i-го символа. Соответственно $Wh(T)$ – комбинаторная сложность гиперслова по $\wp$. Она соответствует мощности множества генерируемых гиперслов при выбранном упорядочении символов, после которой следует искомое T. А вероятность возникновения слова T есть кумулятивной вероятностью

$$P(T) = p_1 p_2 \ldots p_n, \quad (2)$$

где p_i – вероятность появления i-го символа в данном тексте по данной предметной области (ПО) знаний.

Множественная система меток

Понятно, что первые символы имеют больший вес, именно поэтому целесообразно сформировать отдельную метку (метки) для начала текста, но если метки будут распределены неравномерно, то остаток текста будет длинным и его первые символы (префикс) будут иметь веса с высокими степенями. Именно поэтому естественна гипотеза о равномерном распределении меток. Но сначала докажем эффективность использования множественной системы меток при определенных ограничениях на их количество. Каждая метка занимает 1 Б, а совокупное их количество может быть ограничено потребностями пользователя.

Подсчитаем выигрыш в числе операций от применения двух меток, соответствующих префиксному множеству из двух гиперслов. Отдельные метки разделяют текст равномерно и на равные части. Пусть $|W| = N$. Если сначала находим префикс $Pr(w) = w_1$, где w_1 – начало гиперслова w длины $L(w_1) \approx \left[\frac{N}{2}\right]$, соответствующей таймерной метке μ_0. Но если следующая часть генерируемого слова не совпадает с W, продолжаем генерировать ее с $L + 1$-го места до конца слова. Соответственно следующую часть w_2 ищем отдельно, зафиксировав уже найденный префикс $Pr(w) = w_1$. Для них частотные веса будут следующие:

$Wh(w_1) = \sum_{i=1}^{N-L} n_i |A|^{n-i}$, а для второй части текста $Wh(w_2) = \sum_{i=L+1}^{N-L} n_i |A|^{N-i}$. Тогда частотный вес всего слова при использовании переборного генератора $Wh(w) = \sum_{i=1}^{N} n_i |A|^{n-i}$, где $w = Pr(w) \cdot w_2$. Это сократит перебор как минимум в $|A|^{N/2-2}$ раз, поскольку степени знаменателя меньше в два раза

$$k = \frac{Wh(W)}{Wh(W_1) + Wh(W_2)} = \frac{\sum_{i=1}^{N} n_i |A|^{n-i}}{\sum_{i=1}^{N/2} n_i |A|^{n-i} + \sum_{i=N/2}^{N} n_i |A|^{n-i}}.$$

Лингвистическая аналитика

Если для данной ПО знаний ввести статистику не только частот символов, а еще и вероятностей появления символа в слове, после того как определены уже I символов в этом слове, то будем иметь условные вероятности. Именно эти вероятности позволят уменьшить перебор. Например, есть ряд прилагательных с одинаковыми окончаниями: статистический, средневзвешенный, корреляционный, математический, интегральный.

Поэтому при $I > 5$, при возникновении сочетания букв *ни*, логично начать подбор следующего символа с буквы *й*, не учитывая, что его частота сравнительно мала. Также при $I > 3$ целесообразно учитывать возможность появления не существующих комбинаций символов в языке данной области знаний. Например, не могут появляться три гласных символа подряд и другие сочетания. Это все формализуется в условных вероятностных переходов к следующему символу.

Префиксно-таймерное кодирование. Методика вычисления вероятностей

Рассмотрим вероятности и скорости порождения текстов статистически настроенным генератором. Пусть Gs – статистически обученный генератор. Следует найти вероятность генерирования слова. Если *aaa* – нужное слово раньше, чем *abab* – произвольное слово. Рассмотрим помеченный граф вида такого, как на рис. 1 и 2, и соответствующие вероятности переходов.

$$\begin{cases} x_{\varnothing} = p(a)x(a) + p(b)x_{\varnothing} \\ x_a = p(a)x_{aa} + p(a)x_{aa} \\ x_{ab} = p(a)x_{aba} + p(b)x_{\varnothing} \\ x_{aba} = p(a)\cdot x_{aa} + p(b)x_{abab}, \quad x_{abab} = 0 \\ x_{aa} = p(a)\cdot x_{aaa} + p(b)x_{ab}, \quad x_{aaa} = 1 \end{cases},$$

где x_{aa} – вероятность прийти из одного состояния *aa* искомого слова *aaa* раньше, чем к состоянию слова *abab*.

Определение. Префикс-функцией по S от строки L назовем самый длинный префикс с $Pr(S)$, которым заканчивается строка L, $L \in W$. Обозначим эту функцию как $Prf_S(L)$.

Если есть запрещенные слова (что не есть началом ни одного из искомых слов), то их префиксы добавляются в множество $Pr(S)$. Попадание в запрещенное слово приводит в вершину ∅. Например, в этом же наборе {*aaa*, *abab*} не должно встречаться *bbb*, так как оно не есть началом слов набора.

Если положить, что *abab* – текст из данной области знаний и имеет метки в созданном префиксном множестве *M*, то будем иметь *параллельное разархивирование двух текстов.* При этом упорядочение осуществляется по среднестатистическим данным для множества текстов из данной ПО.

Метод вычисления вероятности

Вероятность перехода к искомому слову при генерировании равна сумме произведений вероятностей путей, по которым можно прийти в состояние *aaa* из начального состояния ∅.

$$P_{aaa} = \sum_{i=1}^{k} p_i(\varnothing, ..., aaa),$$

где $p_i(\varnothing, ..., aaa)$ – вероятность перехода по *i*-му пути из ∅ в *aaa*.

Если *abab* – тестовое среднестатистическое слово, для которого нет специальных меток подслов, то статистический порядок букв выполняется не для искомого гиперслова, а для *aaa*. Заметим, что переход с дописыванием буквы *a* происходит чаще, так как он более приоритетен, поэтому вероятность получения слова *aaa* раньше, чем *abab*, будет больше половины, что доказывает эффективность статистически ориентированного генератора. Среднее ожидаемое время $M(t(aaa)) < M(t(abab))$.

При большой длине текста так вычислять вероятность перехода нерационально. Но из системы (1), соответствующей марковским цепям, можно относительно легко вычислить левую часть предпоследнего уравнения, т.е. x_{aba}, так как в его правой части $x_{abab} = 0$ и $x_{aa} = p(a)$. Далее, подставляя найденное x_{aba} в другие уравнения, можно легко найти вероятности во всех состояниях.

Утверждение. Пусть порядок символов зафиксирован, $t(\mu_0) > t(\mu_1)$, тогда вероятность генерации *методом совместного генерирования* одного из слов w_0, w_1 больше, чем *методом раздельного генерирования*.

Это так потому, что каждое из w_0, w_1 имеет свою систему меток делителей слова, и в процессе генерации большего слова w_0 будут возникать подслова, соответствующие меткам для w_1 и наоборот. Поэтому нецелесообразно генерировать отдельно слова w_1, так как построение этих слов будет происходить одновременно. При этом порядок букв для w_0, w_1 одинаков.

Архивирование по множеству префиксов

Определение. Делителем слова *w* называется подслово *u*: $w = p \cdot u \cdot r$, где p, r – подслово слова.

Таймерный счетчик запускается в начале генерации, но его значения, полученные в процессе генерации, фиксируются в ОЗУ после каждой сгенерированной текстовой единицы дискретизации, которая может быть словом или предложением. Это необходимо для сопоставления с элементами множества $Pr(S)$. Они – не те и не другие, а соответствуют префиксам и служат делителями текста *T*.

В данном исследовании с целью архивирования методом таймерного кодирования введем специально помеченное *таймерными метками множество префиксов S*, соответствующее подсловам текста *W*. Среди этих меток обязательно есть полная метка, соответствующая всему тексту.

Вполне естественно, что окончание гиперслова (обобщенный суффикс) или произвольный делитель этого гиперслова может совпадать с подсловом искомого гиперслова *W*, а проверить это можно путем сравнения меток этих подтекстов. Поэтому, имея метки некоторого набора делителей гиперслова *W*, можно при их появлении в процессе статистически ориентированного перебора существенно сократить время работы (количество перебранных текстов до получения закодированного текста). Кроме того, вполне естественно для архивирования брать несколько текстов из родственных ПО.

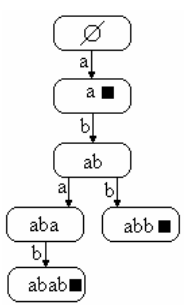

Рис. 1. Дерево поиска префиксов

Докажем, что их совместное разархивирование будет более быстрым. Пусть дано множество меток архивированного набора строк S. Рассмотрим множество возможных *префиксов* $Pr(S)$ – множества начал некоторых подстрок (выделенные префиксы) для всех строк из S, включая собственно строки, а также префикс нулевой длины. Пусть каждому префиксу соответствует некоторая вершина корневого ориентированного дерева, причем от вершины A ведет стрелка в вершину B тогда и только тогда, когда префикс, соответствующий B, можно получить из префикса, соответствующего A, путем приписывания к нему одного символа. Например, для слов *abb* и *abab* можно взять $Pr(S) = \{a, abb, abab\}$. Дерево поиска для них представлено на рис. 1. На дереве квадратами обозначены префиксы – члены $Pr(S)$. Возле стрелок указаны символы, которые нужно добавить, чтобы перейти к указанному префиксу. Теперь, если задана некоторая метка, соответствующая общей части двух архивированных текстов, то следует отдельно генерировать соответствующее этой метке гиперслово для обоих текстов, что повышает скорость совместного разархивирования.

Если сгенерировано гиперслово, то, чтобы проверить его наличие в S, нужно, начиная с метки пустого префикса, «пройти» по дереву поиска (это траектория параллельного генерирования с таймерным поиском), каждый раз двигаясь по ребру, помеченному символами заданного набора S. Заданное гиперслово будет частью S тогда и только тогда, когда находится соответствующее ребро. В итоге, окажемся на префиксе, помеченном квадратиком. При наличии набора S вместо одного генерируемого текста, возможных переходов по ребрам из той же вершины (при условии, что эти переходы ведут к одному из элементов с $Pr(S)$) понятно будет не меньше, поэтому скорость совместного разархивирования больше. Введем следующую функцию.

Определение. *Префикс-функцией по* S от строки L назовем длинный префикс с $Pr(S)$, которым заканчивается строка L, $L \in W$. Обозначим эту функцию как $Prf_s(L)$. Так как пустой префикс принадлежит $Pr(S)$, то для произвольной строки он будет «кандидатом» на роль значения префикс-функции. Поэтому для произвольной строки $Prf_s(L)$ определена.

В общем случае меткам соответствуют произвольные, заранее выбранные делители гиперслова W. Они образуют множество S. Поэтому в процессе генерации, как только будет возникать подслово, соответствующее элементу из множества S, оно будет превращаться в двоичную запись подслова. После этого будет осуществляться поиск продолжений полученного подслова путем такой же генерации символов и символов с учетом вероятности их появления. Эти вероятности учтены при определении порядка генерации. При дальнейшем дописывания символов в $Prf_s(p)$ возможно появление следующего префикса с S. Для этого следует вычислить $Prf_s(pc)$, который формально будет суффиксом нового подслова pc. Максимум $Prf_s(pc) = pc$.

Опишем этот процесс путем построения специального графа переходов по префиксам p. В этом графе состояния – множество выявленных делителей гиперслова W и исходное состояние $\emptyset$.

Граф переходов по префиксам. Дерево поиска подслов можно развить в новую структуру данных – *граф переходов по префиксам* с W, что позволяет быстро ответить на вопрос: *какие гиперслова с S входят в генерируемый G, на каких местах и сколько раз*? Эта структура также самоконсолидируема. Ее перестройка происходит примерно так же, как для дерева поиска, но более длительна. Чтобы лучше представить себе работу с графом переходов по префиксам, напомним определения.

Строим граф всех переходов между префиксами, т.е. для каждого префикса указываем, каким он будет, если к нему приписать взятый символ. Под символом будем понимать делитель гиперслова, соответствующий единице дискретизации таймерного счетчика. Граф строится так, чтобы для произвольной строки L выполнялось следующее. Если начинать маршрут по графу от пустого префикса и на каждом шаге следовать по стрелке с очередным символом из L, то в конце маршрута получим префикс, равный $Prf_S(L)$. Существование такого графа возможно, так как для произвольных строк x и y выполняется

$$Prf_s(x) = Prf_s(y) \Rightarrow Prf_s(xc) = Prf_s(yc) \qquad (6)$$

для произвольного символа c (или части текста, соответствующей единице дискретизации данного текста таймерным счетчиком). Рассматриваемый символ приписан к строкам x и y. Для нашего набора S изобразим граф переходов по префиксам (рис. 2).

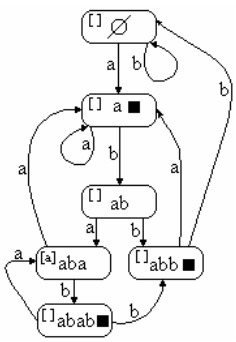

Рис. 2. Граф переходов по префиксам

Примем предложение за единицу дискретизации. В начале каждого из них запускается таймерный счетчик, а в конце – считываются значения времени и сравниваются со значениями меток с $Pr(S)$. В случае нахождения соответствующей метки отыскивается продолжение из текста и запоминается адрес найденной метки в ОЗУ. Опишем процесс построения такого графа.

Сначала зафиксируем на этапе, когда найдена метка для текста длиной n, дерево переходов или же отметим его ребра как *основные*. Далее в дереве переходов, что соответствует формированию текста, с каждой найденной вершины продолжим определять переходы по всем возможным символам, а не выполнять построение этого

графа заново, как это было бы в случае генератора с одной меткой. При таком генерировании, первым будет выполняться переход, соответствующий более приоритетному символу.

Для произвольного символа c и для произвольного префикса p с $Pr(S)$ алгоритм ищет $Prf_s(pc)$ и с префикса p направляет стрелку с символом c в префикс $Prf_s(pc)$. Направление стрелок определяется динамически, т.е. из результатов для меньших префиксов следуют результаты для больших префиксов.

Индукция по длине префикса. Для пустого префикса переходы вычисляются на основе дерева поиска. В случае успеха ответом будет односимвольный префикс c, а в случае неуспеха – пустой префикс.

Предположим, что для префиксов длиной, меньшей, чем данная, все переходы генерируются и подсчитываются. На этом этапе можно применить распараллеливание после нахождения префикса $p \in Pr(S)$. Подсчитаем переходы для всех префиксов данной длины. Проверим присутствие pc в $Pr(S)$ с помощью дерева поиска по тексту G. Для этого генератор дописывает символы c к имеющемуся префиксу. Это можно выполнить за один шаг, когда сохраняется место нахождения таймерной метки, что устанавливается таймерным счетчиком, работающим параллельно, начала префикса p в тексте. При параллельной работе нескольких генераторов сохраняем место нахождения таймерных меток всех префиксов в дереве порожденных последовательностей. В случае успеха направляем стрелку в pc. В случае неуспеха, в силу (1) будет выполнено $Prf_s(pc) = Prf_s(p'c)$ где p' получается из p отбрасыванием первого символа (дальше используется это обозначение и для других строк). $Prf_s(p'c)$ может быть получено на основе уже частично построенного графа переходов. Если сохранить все адреса префиксов $Prf_s(p'c)$ (их число линейно зависит от объема S), то следующий $Prf_s(p'c)$ можно получить за один шаг, начиная с $Prf_s(p'c)$. Текст, соответствующий метке $Prf_s(p')$, будет сохранен, поскольку $p' = q's$, где q – префикс, предыдущий p в дереве поиска, а s – последний символ префикса p. И наконец, чтобы не пропустить ни одного слова, для каждого префикса p из $Pr(S)$, необходимо указать $LwS(p)$ – наиболее длинное слово из S, не совпадающее с p в случае его наличия. Это тоже можно выполнить динамически. Для пустого префикса LwS отсутствует. Поскольку $Lw_s(p)$ – кандидат на роль $Prf_s(p')$, то $Lw_s(p)$ служит окончанием последнего. Поэтому нужно проверить принадлежность $Prf_s(p')$ множеству S. Это выполняется в один шаг, поскольку на предварительном этапе сохранено значение $Prf_s(p')$. Если $Prf_s(p')$ – в S, то $Lw_s(p) = Prf_s(p')$. В противном случае (см. рис. 2) – переход по стрелке a с aba, т.е. новое вложение префикса в полученное слово без дописывания нового символа: $Lw_s(p) = Lw_s(Prf_s(p'))$.

Последнее выражение уже сохранено, так как $Prf_S(p')$ короче, чем p. Алгоритм завершен. Он выполняется за количество шагов, линейно зависимых от объема S.

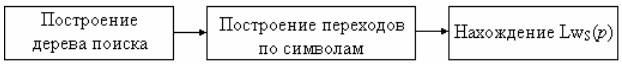

Рис. 3. План построения графа переходов по префиксам

На рис. 3 в скобках для приставок указаны их значения Lw_s. На самом деле, нужно хранить ссылки на них в графе, так как по этим ссылкам будут осуществляться переходы. Двигаясь по графу, отмечаем все префиксы, обозначенные квадратиками. При попадании на префиксы, имеющие Lw_s, переходим по ссылкам на Lw_s, пока это возможно, каждый раз отмечая новое вхождение некоторого слова из S. После перехода по таким ссылкам возвратимся к тому префиксу, на который попали последним переходом по символу, а не по ссылкам Lw_s.

Проиллюстрируем действие алгоритма на примере. Пусть вводится строка *ababb*. Для нее очередность операций алгоритма будет следующая:

0. Находимся на пустом префиксе.

1. Переход по '*a*' на "*a*", фиксация вхождения *"a"* с концом в п. 1.

2. Переход по '*b*' на "*ab*".

3. Переход по '*a*' на "*aba*", для чего осуществляется:
– переход по Lw_s на "*a*", фиксация вхождения "*a*" с концом в п. 3;
– возвращение на "*aba*".

4. Переход по '*a*' на "*abab*". Фиксация вхождения *"abab"* с концом в п. 4.

5. Переход по '*b*' на "*abb*". Фиксация вхождения "*abb*" с концом в п. 5.

При большом количестве вариантов символов можно выполнить переход по меньшей единице дискретизации. Это дает уменьшение требуемой памяти и числа шагов примерно в $m/(2\log_2 m)$ раз, где m – количество различных возможных подтекстов, имеющих такую же длину, что и единица дискретизаций d. Если взять единицу дискретизации d = 3 Б, то даже после применения методов символьного кодирования получим сжатие в 8*3 – 1 = 23 раза. Это так потому, что один бит идет на собственно метку. Но при этом с увеличением величины d возрастает время работы. Алгоритм, использующий граф переходов по префиксам, выполняет количество шагов, линейно пропорциональное сумме размеров введенного текста. При этом достигается теоретически предсказанная максимальная эффективность таймерного кодирования.