Розділ VI. Теорія кодування

Питання кодування здавна відігравало значну роль у математиці. Наприклад, десяткова позиційна система числення – це спосіб кодування натуральних чисел. Римські цифри – інший спосіб кодування натуральних чисел, до того ж більш наглядний та природній: палець – I, п'ятірня – V, дві п'ятірні – X. Але при цьому способі кодування складніше виконувати арифметичні дії над великими числами, тому він був витиснений позиційною десятковою системою.

Раніше засоби кодування відігравали допоміжну роль і не розглядались як окремий предмет математичного вивчення, але з появою комп'ютерів ситуація радикально змінилась. Теорія кодування виникла у 40-х роках XX століття після появи робіт К. Шенона. У цій теорії досліджуються методи кодування, пов'язані з основними математичними моделями, які відображають істотні риси реальних інформаційних систем.

Кодування буквально пронизує інформаційні технології і є центральним питанням при розв'язанні самих різних (практично усіх задач) програмування:

- представлення даних довільної природи (наприклад, чисел, тексту, графіки) у пам'яті комп'ютера;
- захист інформації від несанкціонованого доступу;
- забезпечення перешкодозахищеності при передачі даних по каналам зв'язку;
- стиснення інформації у базах даних.

Властивості, які вимагаються від кодування, бувають різної природи:

- існування декодування – це дуже природна властивість, але навіть вона потрібна не завжди. Наприклад, трансляція програми на мові високого рівня у машинні команди – це кодування, для якого не потрібно однозначного декодування;
- перешкодозахищеність, або виправлення помилок – коли від кодування вимагається можливість відновлення інформації в разі її пошкодження;
- задана складність (або простота) кодування та декодування. Наприклад, у криптографії вивчаються такі способи кодування, при яких є просто обчислювальна функція F, але визначення зворотної функції F^{-1} потребує дуже складних обчислень.

В цьому розділі буде розглянуто деякі найбільш важливі задачі теорії кодування та продемонстровано застосування більшої частини згаданих тут методів.

Тема 21. Теорія кодування

21.1. Алфавітне й рівномірне кодування

Без утрати загальності можна сформулювати задачу кодування так.

Означення 21.1. Нехай задано алфавіт $A = \{a_1, \ldots, a_n\}$ зі скінченної кількості символів, які називають **буквами**. Скінченну послідовність букв алфавіту A називають **словом** у цьому алфавіті. Для позначення слів будемо використовувати малі грецькі літери: $\alpha = a_{i1}a_{i2}\ldots a_{il}$. Число l – кількість букв у слові α - називають **довжиною** слова α і позначають $l(\alpha)$ або $|\alpha|$.

Множину всіх слів у алфавіті A позначають як A^*. Порожнє слово позначають ε; зазначимо, що $\varepsilon \notin A$, $\varepsilon \in A^*$, $|\varepsilon|=0$.

Означення 21.2. Якщо слово α має вигляд $\alpha = \alpha_1\alpha_2$, то α_1 називають **початком** або **префіксом** слова α, а α_2 – його **закінченням** або **постфіксом**. Якщо при цьому $\alpha_1 \neq \varepsilon$ (відповідно, $\alpha_2 \neq \varepsilon$), то α_1 називають **власним початком** (відповідно **власним закінченням**) слова α.

Для слів визначена операція конкатенації або зчеплення.

Означення 21.3. **Конкатенація** слів α та β є слово $\alpha\beta$, отримане виписуванням підряд спочатку всіх букв слова α, а потім всіх букв слова β. Зазвичай операція зчеплення ніяк не позначається.

Означення 21.4. Довільна множина L слів у деякому алфавіті A називається **мовою** в цьому алфавіті, $L \subset A^*$.

Нехай задано алфавіти $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_m\}$ і функція F: $S \to B^*$, де S – деяка множина слів у алфавіті A, $S \subset A^*$. Тоді функція F називається **кодуванням**, елементи множини S – **повідомленнями**, а елементи $\beta = F(\alpha)$, $\alpha \in S$, $\beta \in B^*$ – **кодами** (відповідних повідомлень). Обернена функція F^{-1}, якщо вона існує, називається **декодуванням**.

Якщо $|B| = m$, то F називають m-ковим кодуванням. Найчастіше використовується алфавіт $B = \{0, 1\}$, тобто **двійкове кодування**. Саме його ми й розглядатимемо в наступних підрозділах, випускаючи слово «двійкове».

Відображення F задають якимось алгоритмом. Є два способи задати відображення F.

Алфавітне кодування задають схемою (або таблицею кодів) σ:

$$a_1 \to \beta_1,$$
$$a_2 \to \beta_2,$$
$$\ldots$$
$$a_n \to \beta_n,$$

де $a_i \in A$, $\beta_I \in B^*$, $i = 1,\ldots,n$. Схема σ задає відповідність між буквами алфавіту A та деякими словами в алфавіті B. Вона визначає алфавітне кодування так: кожному слову $\alpha = a_{i1}a_{i2}\ldots a_{il}$ із повідомлення S поставлено у відповідність слово $\beta = \beta_{i1}\beta_{i2}\ldots\beta_{il}$ – його код. Слова $\beta_1, \ldots, \beta_n$ називають **елементарними кодами**.

Наприклад, розглянемо алфавіт $A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $B = \{0, 1\}$ і схему:
$\sigma_1 = \langle 0 \to 0, 1 \to 1, 2 \to 10, 3 \to 11, 4 \to 100, 5 \to 101, 6 \to 110, 7 \to 111, 8 \to 1000, 9 \to 1001\rangle$.

Ця схема однозначна, але кодування не є взаємно однозначним:

$$F_{\sigma 1}(333) = 111111 = F_\sigma(77),$$

а значить, неможливе декодування. З іншого боку, схема

$$\sigma_1 = \langle 0 \to 0000, 1 \to 0001, 2 \to 0010, 3 \to 0011, 4 \to 0100, 5 \to 0101,$$
$$6 \to 0110, 7 \to 0111, 8 \to 1000, 9 \to 1001\rangle,$$

відома під назвою «двійково-десяткове кодування», допускає однозначне декодування.

Рівномірне кодування з параметрами k й r задають так. Повідомлення α розбивають на блоки довжиною k:

$$\alpha = (x_1\ldots x_k)(x_{k+1}\ldots x_{2k})\ldots(x_{mk+1}\ldots x_{pk+s}),$$

де $s \le k$ (останній блок може бути коротшим, у такому разі спосіб його кодування спеціально обумовлюють), $x_i \in A$, $i = 1,\ldots, pk + s$. Блоки довжиною k розглядають як «букви» якогось алфавіту (таких блоків, очевидно, n^k, бо алфавіт A складається з n букв) і кодують словами в алфавіті B довжиною r за **схемою рівномірного кодування** $\sigma_{k,r}$:

$$\alpha_1 \to \beta_1,$$
$$\alpha_2 \to \beta_2,$$
$$\ldots$$
$$\alpha_{n^k} \to \beta_{n^k}.$$

Надлишковістю схеми $\sigma_{k,r}$ **на символ повідомлення** називають величину $R = (n-k)/k = (n/k) - 1$.

21.2. Роздільні схеми

Розглянемо схему алфавітного кодування σ та різні слова, складені з елементарних кодів.

Означення 21.5. Схему σ називають **роздільною**, якщо з рівності $\beta_{i_1}\ldots\beta_{i_k} = \beta_{j_1}\ldots\beta_{j_l}$ випливає, що $k = l$ та $i_t = j_t$ для кожного $t = 1,\ldots, k$, тобто будь-яке слово, складене з елементарних кодів, можна єдиним способом розкласти на елементарні коди.

Алфавітне кодування з роздільною схемою вможливлює однозначне декодування.

Означення 21.6. Схему σ називають **префіксною**, якщо для будь-яких i та j ($i, j = 1,\ldots,n, i \neq j$) елементарний код β_i не є префіксом елементарного коду β_j.

Теорема 21.1 (без доведення). Префіксна схема є роздільною.

Властивість префіксності достатня, але не необхідна умова роздільності схеми. Наприклад, нехай $A = \{a, b\}$, $B = \{0, 1\}$, $\sigma = \langle a \to 0, b \to 01 \rangle$. Схема σ не префіксна, але роздільна. Справді, перед кожним входженням 1 в слові β_2 безпосередньо є 0. Це дає змогу виділити всі пари (01). Частина слова, що залишилась, складається із символів 0.

Щоби схема алфавітного кодування була роздільною, необхідно, щоби довжини елементарних кодів задовольняли певному співвідношенню, відомому як нерівність Мак-Міллана.

Теорема 21.2 (нерівність Мак-Міллана, без доведення). Якщо схема алфавітного кодування $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ роздільна, то

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leq 1, \text{ де } l_i = |\beta_i|.$$

Теорема 21.3 (без доведення). Якщо числа $l_1,\ldots, l_n$ задовольняють нерівність

$$\sum_{i=1}^{n} \frac{1}{2^{l_i}} \leq 1 \text{ (нерівність Мак-Міллана),}$$

то існує префіксна схема алфавітного кодування σ:

$$a_1 \to \beta_1,$$
$$\ldots$$
$$a_n \to \beta_n,$$

де $l_i = |\beta_i|$.

Наслідок 1. Нерівність Мак-Міллана – необхідна й достатня умова існування алфавітного кодування з префіксною схемою та довжинами елементарних кодів, що дорівнюють $l_1,\ldots, l_n$.

Наслідок 2. Якщо існує роздільна схема алфавітного кодування із заданими довжинами елементарних кодів, то існує й префіксна схема з тими самими довжинами елементарних кодів.

21.3. Оптимальне кодування

Для практики важливо, щоби коди повідомлень мали по можливості найменшу довжину. Алфавітне кодування придатне для довільних повідомлень, тобто $S = A^*$. Якщо більше про множину S не відомо нічого, то точно сформулювати задачу оптимізації важко. Однак на практиці часто доступна додаткова інформація. Наприклад, для текстів на природних мовах відомо розподілення ймовірностей (частот) появи букв у повідомленні. Використання такої інформації дозволяє строго поставити та розв'язати задачу побудови оптимального алфавітного кодування.

Нехай задано алфавіт $A = \{a_1, \ldots, a_n\}$ і ймовірності появи букв у повідомленні $P = (p_1, \ldots, p_n)$; тут p_i – ймовірність появи букви a_i (ймовірність можна розглядати як частоту і обчислювати у вигляді формули $f(a_i)/l$, де $f(a_i)$ – кількість разів появи букви a_i у повідомленні, а l – загальна кількість букв у повідомленні, тобто довжина повідомлення). Не втрачаючи загальності, можна вважати, що $p_1 \geq p_2 \geq \ldots \geq p_n > 0$, тобто можна одразу вилучити букви, які не можуть з'явитись у повідомленні, і впорядкувати букви за зменшенням ймовірностей їх появи. Крім того, $p_1 + p_2 + \ldots + p_n = 1$.

Означення 21.7. Для кожної роздільної схеми алфавітного кодування $\sigma = \langle a_i \to \beta_i \rangle_{i=1}^{n}$ величина $l_\sigma(P) = \sum_{i=1}^{n} p_i l_i$, де $l_i = |\beta_i|$, $i = 1, \ldots, n$ називають **середньою довжиною або ціною** кодування за схемою σ для розподілу ймовірностей P.

Наприклад, для алфавітів $A = \{a, b\}$, $B = \{0, 1\}$ та роздільної схеми $\sigma = \langle a \to 0, b \to 01 \rangle$ при розподілі ймовірностей (0,5, 0,5) ціна кодування складає $0{,}5\times1 + 0{,}5\times2 = 1{,}5$, а при розподілі ймовірностей (0,9, 0,1) вона дорівнює $0{,}9\times1 + 0{,}1\times2 = 1{,}1$.

Уведемо величину $l_*(P) = \inf\limits_{\sigma} l_\sigma(P)$, де нижню грань беруть за всіма роздільними схемами σ. Легко довести, що

$$1 \le l_*(P) \le \lceil \log_2 n \rceil,$$

де верхню оцінку дає схема з елементарними кодами з однаковою довжиною $\lceil \log_2 n \rceil$. Звідси випливає, що для побудови кодів, у яких величина l_σ близька до l_*, можна не враховувати коди з більшим l_σ, ніж $\lceil \log_2 n \rceil$. Отже, будемо розглядати лише схеми з $p_i l_i \le \lceil \log_2 n \rceil$. Позначимо $p_* = \min(p_1, \ldots, p_n)$, тоді

$$l_i \le \lceil \log_2 n \rceil / p_*$$

для всіх $i = 1,\ldots, n$.

Звідси випливає, що є лише скінченна кількість варіантів значень l_σ, для яких $l_* \le l_\sigma \le \lceil \log_2 n \rceil$. Отже, значення l_* досягається на якійсь схемі σ; його можна визначити як

$$l_* = \min_{\sigma} l_\sigma .$$

Означення 21.8. Коди, визначені схемою σ з $l_\sigma = l_*$, називають кодами з **мінімальною надлишковістю** або **оптимальними кодами** для розподілу ймовірностей P.

За наслідком 2 з теорем 21.2 та 21.3 існує алфавітне кодування з префіксною схемою, яке дає оптимальні коди. У зв'язку з цим, будуючи коди, можна обмежитися префіксними схемами.

21.4. Алгоритм Шенона-Фано

Алгоритм Шеннона-Фано – один з перших алгоритмів кодування, який вперше сформулювали американські вчені Шенон та Фано. Даний метод кодування має велику подібність із алгоритмом Хаффмана, який з'явився на декілька років пізніше. Алгоритм заснований на кодах змінної довжини: символ, який зустрічається часто, кодується кодом меншої довжини і навпаки. Для того, щоби існувало декодування, коди Шенона-Фано мають володіти унікальністю, тобто, не дивлячись на їх змінну довжину, кожний код унікально визначає один закодований символ і не є префіксом будь-якого іншого коду. Тобто отриманий код є префіксним.

Код Шенона-Фано будується за допомогою дерева. Побудова цього дерева починається від кореня. Вся множина елементів, які кодуються, відповідає кореню дерева (вершині першого рівня). Вона (множина) розбивається на дві підмножини з приблизно однаковими сумарними ймовірностями. Ці підмножини відповідають двом вершинам другого рівня, які з'єднуються з коренем. Далі кожна з цих підмножин розбивається на дві підмножини з приблизно однаковими сумарними ймовірностями. Їм відповідають вершини третього рівня. Якщо підмножина містить один елемент, то йому відповідає кінцева вершина кодового дерева; така підмножина більше не розбивається. Подібним чином поступаємо до тих пір, поки не отримуємо всі кінцеві вершини. Гілки кодового дерева помічаємо символами 1 та 0.

При побудові коду Шенона-Фано розбиття множини елементів може бути виконано, взагалі кажучи, декількома способами. Вибір розбиття на рівні n може погіршити варіанти розбиття на наступному рівні (n+1) і привести до неоптимального коду в цілому. Іншими словами, оптимальна поведінка на кожному кроці шляху не гарантує оптимальності усієї сукупності дій. Тому код Шенона-Фано не є оптимальним в загальному сенсі, хоча й дає оптимальні результати при деяких розподілах ймовірностей. Для одного й того самого розподілу ймовірностей можна побудувати, взагалі кажучи, декілька кодів Шенона-Фано, і всі вони можуть давати різні результати.

Кодування Шенона-Фано є доволі старим методом і на сьогоднішній день воно не представляє собою практичного інтересу. В більшості випадків, довжина отриманої

послідовності, за даним методом, дорівнює довжині послідовності при використанні методу Хаффмана. Але на деяких послідовностях все ж таки формуються неоптимальні коди Шенона-Фано, тому метод Хаффмана прийнято вважати більш ефективним.

Розглянемо приклад кодового дерева. Нехай маємо алфавіт $A = \{a, b, c, d, e, f\}$, де для кожної букви відома частота появи у тексті – $f(\cdot)$, за якою можна визначити ймовірність $p(\cdot)$. Частоти, ймовірності та коди букв представлені в табл. 21.1.

Буква	Частота f	Ймовірність p	Код
a	50	0,23	11
b	39	0,18	101
c	18	0,08	100
d	50	0,23	01
e	33	0,16	001
f	26	0,12	000

Табл. 21.1.

Кодове дерево, яке отримуємо під час роботи алгоритму, зображено на рис. 21.1.

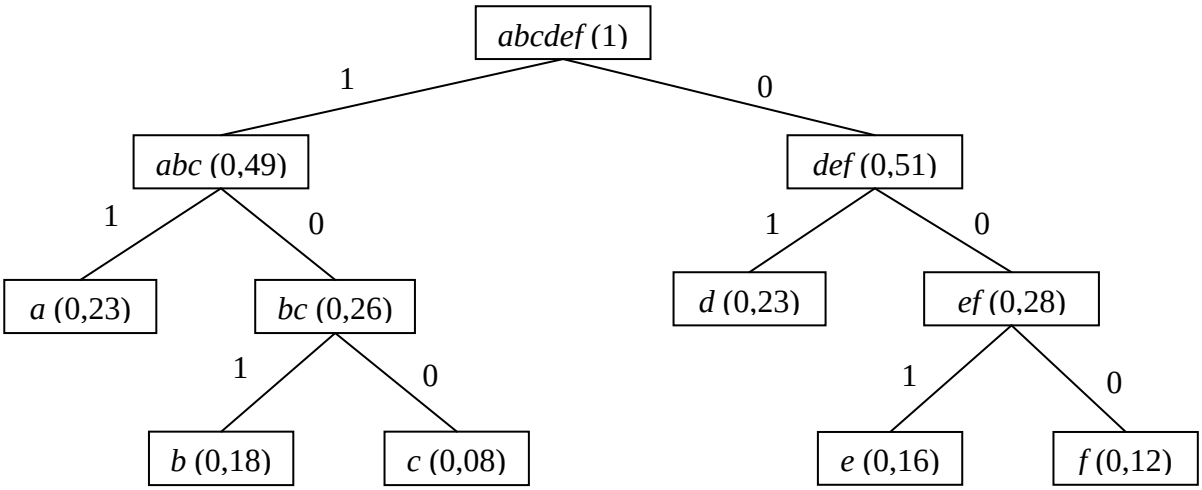

Рис. 21.1.

Підрахуємо l_σ: 0,23×2 + 0,18×3 + 0,08×3 + 0,23×2 + 0,16×3 + 0,12×3 = 2,54.

21.4. Алгоритм Хаффмана

Розглянемо декілька властивостей оптимальних кодів.

Теорема 21.4. В оптимальному коді букву з меншою ймовірністю її появи не можна закодувати коротшим словом. Інакше кажучи, для оптимального коду з того, що $p_i < p_j$ випливає, що $l_i \geq l_j$.

Доведення. Припустимо протилежне: нехай є дві букви a_i та a_j такі, що $p_i < p_j$ і $l_i < l_j$. Поміняємо місцями β_i та β_j у схемі кодування. Тоді середня довжина елементарних кодів зміниться на

$$p_i l_j + p_j l_i - p_i l_i - p_j l_j = (p_i - p_j) l_j - (p_i - p_j) l_i = (p_i - p_j)(l_j - l_i) < 0,$$

тобто зменшиться, що суперечить оптимальності коду. ►

Очевидно, що якщо код оптимальний, то можна перенумерувати букви алфавіту A і відповідні елементарні коди β_i, так що $p_1 \geq p_2 \geq \ldots \geq p_n$ та $l_1 \leq l_2 \leq \ldots \leq l_n$. Далі ми будемо розглядати схеми кодування, де коди впорядковані таким чином.

Теорема 21.5. В оптимальному коді є два елементарні коди з найбільшою довжиною l_n, які відрізняються лише останніми символами.

Доведення. Припустимо, що це не так. Тоді можна відкинути останній символ елементарного коду β_n, не порушуючи властивості префіксності. При цьому, очевидно,

зменшиться середня довжина елементарного коду. Це суперечить твердженню, що код оптимальний. ►

Теорема 21.6. Існує такий оптимальний код, у якому елементарні коди двох найменш імовірних букв a_n та a_{n-1} відрізняються лише останніми символами.

Доведення. За теоремою 21.5 знайдеться елементарний код β_t, який має ту саму довжину, що й β_n, і відрізняється від нього лише останнім символом. Із теореми 21.4 випливає, що $l_t = l_{t+1} = \ldots = l_n$. Якщо $t \neq n-1$, то можна поміняти місцями β_t та β_{n-1}, не порушуючи нерівності $l_1 \le l_2 \le \ldots \le l_n$. ►

Теорема 21.6 дає змогу розглядати лише такі схеми алфавітного кодування, у яких елементарні коди β_{n-1} та β_n (для двох найменш імовірних букв a_{n-1} та a_n) мають найбільшу довжину й відрізняються тільки останніми символами. Це означає, що листки β_{n-1} та β_n кодового дерева оптимального коду мають бути з'єднані в одній вершині попереднього рівня.

Розглянемо новий алфавіт $A_1 = \{a_1, \ldots, a_{n-2}, a'\}$ із розподілом ймовірностей $P_1 = \{p_1, \ldots, p_{n-2}, p'\}$, де $p' = p_{n-1} + p_n$. Його одержано з алфавіту A об'єднанням двох найменших букв a_{n-1} та a_n в одну букву a' з ймовірністю $p' = p_{n-1} + p_n$. Говорять, що A_1 отримано стисненням алфавіту A.

Нехай для алфавіту A_1 побудовано схему σ_1 з елементарними кодами $\beta_1, \beta_2, \ldots, \beta_{n-2}, \beta'$. Схемі σ_1 можна поставити у відповідність схему σ з елементарними кодами $\beta_1, \beta_2, \ldots, \beta_{n-2}, \beta_{n-1}, \beta_n$ для початкового алфавіту A, узявши $\beta_{n-1} = \beta'0$, $\beta_n = \beta'1$ (тобто елементарні коди β_{n-1} та β_n одержують з елементарного коду β приписуванням справа відповідно 0 та 1). Процедуру переходу від σ_1 до σ називають **розщепленням**.

Теорема 21.7 (без доведення). Якщо схема σ_1 оптимальна для алфавіту A_1, то схема σ оптимальна для алфавіту A.

З цієї теореми випливає такий метод побудови оптимальної схеми алфавітного кодування. Спочатку послідовно стискають алфавіт A до отримання алфавіту з двох букв, оптимальна схема кодування для якого очевидна: першу букву кодують символом 0, другу – символом 1. Потім послідовно розщеплюють одержану схему. Очевидно, що отримана в результаті схема префіксна.

Цей метод кодування запропоновано 1952 р. американським математиком Д. Хаффманом.

Розглянемо використання цього алгоритму для попереднього прикладу з розділу 21.3. У процесі побудуємо так зване дерево Хаффмана. Це є бінарне дерево, що відповідає оптимальному коду, яке будується знизу вгору, починаючи з $|A| = n$ листків за $n - 1$ крок (злиття). Під час кожного кроку (злиття) дві вершини з найменшими ймовірностями об'єднуються однією вершиною вищого рівня, яка буде мати ймовірність, що дорівнює сумі ймовірності початкових двох вершин. При цьому нові ребра позначають 0 та 1.

Збудоване дерево зображено на рис. 21.2.

Після застосування алгоритму отримуємо таку схему кодування:

Буква	Код
a	00
b	010
c	111
d	10
e	011
f	110

Середня довжина побудованого коду, як і у випадку алгоритму Шенона-Фано, становить 0,23×2 + 0,18×3 + 0,08×3 + 0,23×2 + 0,16×3 + 0,12×3 = 2,54.

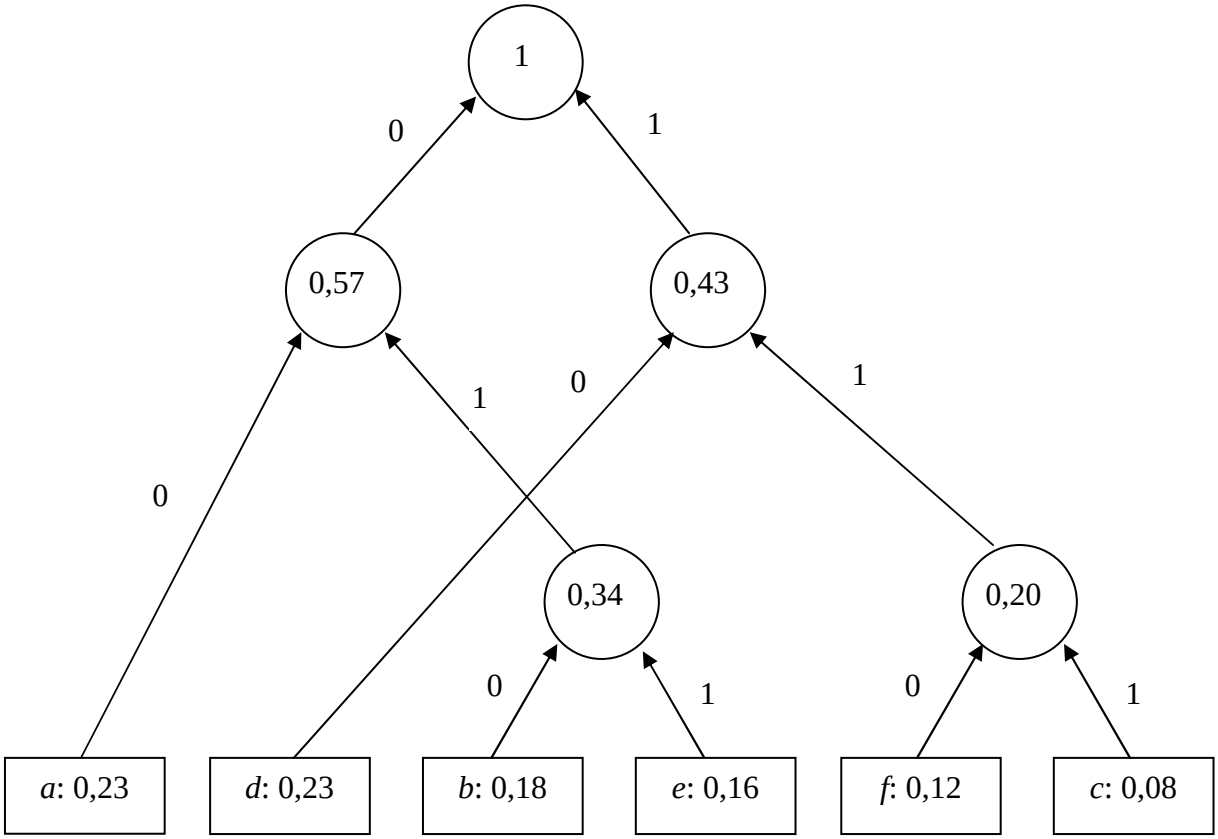

Рис. 21.2

21.5. Стиск даних

Припустимо, що маємо деяке повідомлення, яке закодовано деяким загально прийнятим способом (для текстів це, наприклад, код ASCII) і зберігається в пам'яті комп'ютера. Відмітимо, що рівномірне кодування не є оптимальним для текстів. Дійсно, в текстах зазвичай використовується значно менше, ніж 256 символів (в залежності від мови – приблизно 60-80 з урахуванням розділових знаків, цифр, малих та великих літер). Крім того, ймовірності появи букв різні і для кожної природної мови відомі (з деякою точністю) частоти появи букв у тексті. Таким чином, можна задатися деяким набором букв та частотами їх появи у тексті і за допомогою алгоритмів Шенона-Фано або Хаффмана побудувати оптимальне алфавітне кодування текстів (для заданих алфавіту та мови). Прості підрахунки показують, що таке кодування для розповсюджених природних мов буде мати ціну кодування дещо меншу за 6, тобто дає виграш у порівнянні з кодом ASCII на 25% або дещо більше.

Відомо також, що практичні програми стиснення даних мають значно більші показники: при стисненні текстових документів коефіцієнт стиску досягає 70% й більше. Це означає, що у таких програмах використовується не алфавітне кодування.

Розглянемо наступний спосіб кодування.

1. Висхідне повідомлення за деяким алгоритмом розбивається на послідовності символів, які називаються словами (слово може мати одно або декілька входжень у висхідний текст повідомлення).
2. Отримана множина слів вважається буквами нового алфавіту. Для цього алфавіту будується роздільна схема алфавітного кодування (рівномірного кодування або оптимального кодування, якщо для кожного слова підрахувати кількість входжень у текст). Отримана схема зазвичай називається **словником**, тому що вона співставляє слову код.
3. Далі код повідомлення будується як пара – код словника та послідовність кодів слів з цього словника.

4. При декодуванні висхідне повідомлення відновлюється шляхом заміни кодів слів на слова зі словника.

Наприклад, нехай потрібно кодувати тексти на українській мові. В якості алгоритму розділення на слова візьмемо природні правила мови: слова відокремлюються одне від одного пробілами або розділовими знаками. Можна прийняти припущення, що в кожному конкретному тексті мається не більше 2^{16} різних слів (зазвичай набагато менше). Таким чином, кожному слову можна співставити номер – ціле число з двох байтів (рівномірне кодування). Оскільки в середньому слова української мови складаються більше ніж з двох букв, то таке кодування дає суттєве стиснення тексту (близько 75% для звичайних текстів української мови). Якщо текст достатньо великий (сотні тисяч або мільйони букв), то додаткові витрати на збереження словника виявляються порівняно невеликими.

Вказаний метод можна вдосконалити наступним чином. На кроці 2 слід застосовувати алгоритм Хаффмена або Шенона-Фано для побудови оптимального коду, а на кроці 1 – вирішувати екстремальну задачу розбиття тексту на слова таким чином, щоб серед всіх можливих розбиттів обрати те, яке дає найменшу ціну кодування на кроці 2. Таке кодування буде «абсолютно» оптимальним. Нажаль, вказана екстремальна задача дуже трудомістка, тому на практиці не використовується – час на попередню обробку великого тексту виявляється надто великим.

21.6. Алгоритм Лемпела-Зіва-Велча

Алгоритм Лемпела-Зіва-Велча (LZW) був опублікований Велчем у 1984 році в якості покращення реалізації алгоритму LZ78, який опублікували Лемпел та Зів у 1978 році. Він не є обов'язково оптимальним, тому що не проводить ніякого початкового аналізу даних.

Цей алгоритм використовує ідею **адаптивного стиску** – за один прохід тексту одночасно будується динамічно словник та кодується текст. При цьому словник не зберігається – за рахунок того, що при декодуванні використовується той самий алгоритм побудови словника, словник автоматично відновлюється.

Словник будується під час кодування наступним чином: певним послідовностям символів (словам) ставляться у відповідність групи бітів фіксованої довжини (звичайно 12-бітні). Словник ініціалізується усіма 1-символьними рядками (у випадку 8-бітових рядків – це 256 записів). Під час кодування, алгоритм переглядає текст символ за символом, та зберігає кожний новий, унікальний 2-символьний рядок у словник у вигляді пари код/символ, де код посилається у словнику на відповідний перший символ. Після того, як новий 2-символьний рядок збережений у словнику, на вихід передається код першого символу. Коли на вході зчитується черговий символ, для нього у словнику знаходиться рядок, який вже зустрічався і який має максимальну довжину, після чого у словнику зберігається код цього рядка з наступним символом на вході; на вихід видається код цього рядка, а наступний символ використовується в якості початку наступного рядка.

Розглянемо приклад. Нехай маємо алфавіт Δ, який складається тільки з великих літер латинського і допоміжного символу # - маркер кінця повідомлення: Δ = {A, B, C,…, Z, #}. Таким чином, в алфавіті є 27 символів. Для представлення кожного символу алфавіту Δ нам достатньо 5 бітів. 5-бітні групи дають $2^5 = 32$ можливих комбінації біт, тому, коли у словнику з'явиться 33-е слово, алгоритм має перейти до 6-бітових груп.

Розглянемо кодування рядку із символів алфавіту Δ:

TOBEORNOTTOBEORTOBEORNOT#

Початковий словник буде містити:

# = 00000
A = 00001
B = 00010
C = 00011
…
Z = 11010

Без використання алгоритму LZW повідомлення при передачі як воно є – 25 символів по 5 бітів кожний – воно займе 125 бітів. Порівняємо це з тим, що отримується при використанні алгоритму LZW (табл. 21.2).

Символ	Бітовий код (на виході)	Новий запис словнику
T	20 = 10100	27: TO
O	15 = 01111	28: OB
B	2 = 00010	29: BE
E	5 = 00101	30: EO
O	15 = 01111	31: OR
R	18 = 010010	32: RN
N	14 = 001110	33: NO
O	15 = 001111	34: OT
T	20 = 010100	35: TT
TO	27 = 011011	36: TOB
BE	29 = 011101	37: BEO
OR	31 = 011111	38: ORT
TOB	36 = 100100	39: TOBE
EO	30 = 011110	40: EOR
RN	32 = 100000	41: RNO
OT	34 = 100010	42: OT#
#	0 = 000000	

Табл. 21.2.

Таким чином, використовуючи LZW ми скоротили повідомлення на 28 біт з 125 – це майже 22%. Якщо повідомлення буде довшим, то елементи словнику будуть представляти все більш й більші довші частини тексту, завдяки чому слова, які повторюються, будуть представлені дуже компактно.