УДК 8.74

АССОЦИАТИВНОЕ КОДИРОВАНИЕ: РЕАЛИЗАЦИЯ И ПРИМЕНЕНИЕ

В.М.Величко, В.Д.Гусев, Ю.Г.Косарев, В.С.Лозовский, Т.Н.Титкова

Специалистам – разработчикам ЭВМ хорошо знаком термин "ассоциативная память". Информация в ассоциативных устройствах памяти упорядочивается не по адресам, а в соответствии со значениями имен (признаков, ключей), которыми сопровождается порция информации, заносимая в память. В ответ на запрос по ключу выдается вся информация, которая в процессе запоминания (возможно, в несколько этапов) была снабжена этим ключом. При этом с пользователя снимаются заботы по распределению памяти под массивы и составлению алгоритмов поиска информации с нужным именем.

Идея занесения и поиска информации по ключу широко используется и системными программистами, которые разработали ряд приемов, позволяющих на ЭВМ с адресными ОЗУ достаточно эффективно моделировать работу устройств ассоциативной памяти.

Первая практическая реализация этой идеи, по-видимому, имела место при разработке ассемблера для IBM-70I в I954 г. Первая публикация в иностранной печати появилась в I956 г. [I]. Независимо Л.Н.Королев [2] в I956 г. предложил использовать аналогичную идею для кодирования слов некоторого словаря. А.П.Ершов [3] в I957 г. выдвинул идею адресации с использованием датчика псевдослучайных чисел и привел эмпирические оценки трудоемкости метода.

В работах иностранных авторов для обозначения такого рода процедур обычно используются термины: " hash coding ", "scatter storage", "random-access storage"

Не меньшим разнообразием терминологии отличаются и публикации в этой области на русском языке, несмотря на их относительную немногочисленность. Фигурируют следующие термины: метод хранения и поиска информации с использованием расстановочного поля (А.П.Ершов [4]), функции расстановки (С.С.Лавров, Л.И.Гончарова [5]), перемешанные таблицы, таблицы с вычисляемым входом, разбросанные таблицы, рандомизированные таблицы, таблицы с преобразованием ключа (Ф.Хопгуд [28]).

В данной работе авторы постарались сохранить более или менее установившиеся термины, такие, как "функция расстановки" и "расстановочное поле", используя их в том смысле, в каком они определены в [4,5]. Вместе с тем предлагается использовать термин "а с с о ц и а т и в н о е к о д и р о в а н и е" для самого подхода, реализующего идею занесения и поиска информации в соответствии с некоторой функцией $h(\rho(x))$, указывающей по значению признака ρ элемента массива x (либо по самому x, если $\rho(x)=x$) адрес участка памяти, где размещается данный элемент. Это название, на наш взгляд, в достаточной степени отражает суть обсуждаемых методов и менее обще по сравнению с термином "ассоциативный поиск", используемым С.С.Лавровым и Л.И.Гончаровой как в интересующем нас смысле, так и в смысле поиска информации в линейно упорядоченном массиве, поиска на дереве и т.д.

Настоящая работа преследует две цели. С одной стороны, делается попытка систематизировать основные идеи и методы ассоциативного кодирования. С другой стороны, на ряде примеров показывается, что метод ассоциативного кодирования не следует ограничивать узкими рамками системного программирования и что он с успехом может быть применен, скажем, при решении задач распознавания образов, при выявлении статистических закономерностей в информационных массивах, при построении информационно-поисковых систем, АСУ и т.д.

Порядок изложения и подбор материала продиктован сформулированной целью. В первом параграфе проводится параллель между методом нумерации и методом ассоциативного кодирования и делается естественный, на наш взгляд, переход от первого метода ко второму. Во втором параграфе рассмотрена вероятностная модель, имитирующая процедуру ассоциативного кодирования, и приведены некоторые соотношения, полезные для оценки трудоемкости метода и решения ряда оптимизационных задач, с ним связанных. Третий, четвертый и пятый параграфы носят в основном обзорный характер и посвящены описанию различных процедур ассоциативного кодирования, методов устранения неоднозначности и оценке трудоемкости этих методов. И, наконец, в шестом параграфе рассмотрены примеры использования методов ассоциативного кодирования как в традиционных областях, связанных с системным программированием, так и для решения ряда новых задач, о которых упоминалось выше.

I. М е т о д н у м е р а ц и и и а с с о ц и а т и в н о е к о д и р о в а н и е. Задачу быстрого поиска информации нельзя решать в отрыве от задачи размещения информации в памяти машины, которая предшествует задаче поиска. Известно, что среднее число сравнений при поиске нужного элемента в массиве, не упорядоченном по интересующему нас признаку, равно $n/2$ (n – число элементов массива). В упорядоченном массиве затраты на поиск снижаются до $\log_2 n$ сравнений [5]. Еще бóльшей экономии времени поиска можно достичь, если размещение элементов информационного массива в памяти производить независимо друг от друга, руководствуясь лишь значением признака, характеризующего каждый элемент (или значением самого элемента). В этом состоит основная идея метода ассоциативного кодирования.

Под затратами на поиск в описанных выше примерах мы понимали среднее число сравнений признака интересующего нас элемента с другими. В принципе такого рода затраты могут быть вообще сведены к нулю, т.е. сравнений не понадобится, если установить взаимно-однозначное соответствие между значением признака, характеризующего данный элемент, и адресом, по которому записывается этот элемент.

Фактически речь идет о построении однозначной нумерации для элементов информационного массива. Под нумерацией мы понимаем отображение исследуемого класса объектов на некоторое подмножество множества натуральных чисел.

Рассмотрим в качестве примера задачу размещения и поиска в памяти слов какого-либо словаря с сопутствующей каждому сло-

ву информацией, например, переводом этого слова на другой язык, его синонимами и т.д. Само слово выступает здесь в качестве признака, по которому извлекается сопутствующая ему информация.

Пусть $\{a_0, a_1, \ldots, a_{\tau-1}\}$ – алфавит символов, из которых составлены слова словаря, а $\{0,1,\ldots,\tau-1\}$ – отрезок натурального ряда, элементы которого поставлены во взаимно-однозначное соответствие элементам алфавита. Тогда любому слову $x = a_{k_1} a_{k_2} \ldots a_{k_\rho}$, составленному из элементов алфавита, может быть поставлено во взаимно-однозначное соответствие число

$$W_\rho(x) = k_1 \tau^{\rho-1} + k_2 \tau^{\rho-2} + \ldots + k_\rho \tau^0 = \sum_{i=1}^{\rho} k_i \tau^{\rho-i}, \qquad (I.1)$$

являющееся представлением этого слова в τ-ичной системе счисления.

Выражение (I.I) представляет пример простейшей нумерации, а число W_ρ может рассматриваться как адрес участка памяти, куда заносится информация, связанная с анализируемым словом. Однако уже в этом простейшем случае отчетливо выявляется существенный недостаток такой нумерации – нерациональное использование ресурсов памяти. Действительно, диапазон возможных изменений чисел W_ρ, как правило, очень велик (порядка τ^ρ), в то время как реально используемый словарь существенно ограничен по своему объему. Таким образом, мы получаем очень разреженный информационный массив, требующий для своего размещения громадного объема памяти, хотя отличные от нуля элементы, будучи выписанными подряд, могли бы уместиться в имеющемся объеме памяти. Однако поиск нужного элемента в таком массиве уже не являлся бы таким быстрым и был бы эквивалентен по трудоемкости поиску в упорядоченном массиве. Аналогичный вывод, как правило, справедлив и по отношению к другим попыткам построения взаимно-однозначной нумерации, учитывающей реальные ограничения по памяти и в то же время описываемой достаточно простым а н а л и т и ч е с к и м выражением. Сложность вычисления нумерующей функции в лучшем случае оказывается сравнимой с трудоемкостью поиска одним из вышеупомянутых методов.

Итак, для того чтобы процедура нумерации могла служить эффективным средством быстрого поиска информации, разумно предъявить к ней такие требования, как:

I) экономичность по времени вычисления;
2) экономичность по памяти;
3) универсальность;
4) однозначность прямого и обратного соответствий.

Первое условие подразумевает, что рост трудоемкости поиска в зависимости от длины массива n должен происходить медленнее, чем по закону $c \cdot \log_2 n$. Второе условие означает, что если фиксирован число нумеруемых элементов M, взятых произвольным образом из множества, содержащего N элементов, то процедура нумерации должна поставить в соответствие этим элементам номера в интервале от I до M, что позволяет записать их без пробелов в заданном объеме памяти. Третье условие означает, что алгоритм нумерации не должен зависеть от того, какое конкретное подмножество исходного множества предъявлено для нумерации.

В данной работе речь идет об алгоритмическом построении процедуры нумерации, удовлетворяющей всем сформулированным требованиям. При этом мы не затрагиваем вопроса о возможности представления такой нумерации некоторым аналитическим выражением, что было бы удобно по ряду соображений практического характера. Покажем лишь на некоторых примерах, что снятие любого из ограничений I–4 в принципе позволяет осуществить такое представление и удовлетворить всем оставшимся требованиям.

Предположим, что снято ограничение на время вычисления. Вновь возвращаясь к примеру со словарем, пронумеруем в соответствии с (I.I) все элементы словаря и упорядочим вдоль числовой оси в порядке возрастания их кодов: $W_0, W_1, \ldots, W_M$ ($M+1$ – объем словаря). Тогда задача отображения кодов W_i ($i = 0,1,\ldots,M$) в последовательность адресов $(A+i)$ сводится к построению интерполяционного многочлена

$$\varphi(x) = \alpha_0 x^M + \alpha_1 x^{M-1} + \ldots + \alpha_M, \qquad (I.2)$$

удовлетворяющего условию

$$\varphi(W_i) = A + i, \quad i = 0,1,\ldots,M. \qquad (I.3)$$

Такая постановка задачи рассматривалась в работе [7]. Для адреса участка памяти, где хранится информация о слове X, получено выражение

$$y = \varphi(x) = A + \sum_{k=0}^{M} \left\{ k \prod_{\substack{j=0 \\ j \neq k}}^{M} \left(\frac{W(x) - W_j}{W_k - W_j} \right) \right\}. \qquad (I.4)$$

Очевидно,что нумерация слов словаря,задаваемая выражением (I.4), безизбыточная по памяти,не зависит от конкретного словаря и взаимно-однозначна, что позволяет не записывать в памяти само слово, а записывать лишь относящуюся к нему словарную информацию.

Вместе с тем очевидно, что прямое вычисление адреса в соответствии с (I.4),как это предполагается в работе [7], потребует затрат даже больших, чем это необходимо для поиска слова в неупорядоченном словаре. Легко видеть,однако, что если $W(x) = W_\ell$, т.е. числовой код слова x занимает $(\ell+1)$ -е место в упорядоченном наборе $W_0,\dots, W_M$, то все слагаемые суммы в выражении (I.4) равны нулю за исключением члена, соответствующего $k = \ell$. Если теперь мы попытаемся оптимизировать вычисление выражения (I.4) и откажемся от вычисления заведомо нулевых членов суммы, то мы придем к задаче отыскания единственного ненулевого члена, что эквивалентно задаче поиска номера элемента $W(x)$ в упорядоченном наборе $W_0,\dots, W_M$.

Таким образом,в лучшем случае процедура нумерации в соответствии с (I.4) по своей трудоемкости может сравниться с процедурой поиска элемента в упорядоченном массиве и не обладает никакими дополнительными преимуществами по сравнению с ней.

Сложность процедуры нумерации в разобранном примере частично объясняется универсальностью процедуры, т.е. применимостью её к произвольному словарю. Если отказаться от этого требования, т.е. рассматривать не произвольные подмножества исходного множества, а в некотором смысле "регулярные", выделенные в соответствии с какими-либо ограничениями, то иногда удается,используя информацию о структуре выделенного подмножества, предложить более эффективные процедуры нумерации.

Такой подход, к примеру,реализован в работе [8], где исходное множество представлено всевозможными n -разрядными двоичными векторами (всего 2^n элементов),а подмножества выделяются таким образом, чтобы каждый из входящих в них двоичных векторов содержал одинаковое число единиц - k (всего C_n^k элементов в каждом подмножестве; $k = 0,1,\dots, n$). Полученная нумерация удовлетворяет требованиям I,2,4, но не является универсальной.

Примером нумерации, удовлетворяющей требованиям I,3,4, но не удовлетворяющей требованию 2, является, как уже упоминалось выше, нумерация, задаваемая соотношением (I.I).

И наконец, если отказаться от требования взаимной однозначности нумерации, мы можем использовать для отображения практически любую достаточно простую функцию (функцию расстановки) с множеством значений,соответствующим интересующему нас диапазону чисел, потребовав от неё лишь относительной равномерности распределения значений в указанном диапазоне. Таким образом,мы приходим к процедуре типа " hash coding " [I3].

Неоднозначность нумерации означает, что два (и более) различных объекта могут получить один и тот же адрес в выделенном под информационный массив участке памяти (расстановочном поле). Отказ от однозначности позволяет очень просто организовать процедуру нумерации,поскольку не следует заботиться о том,чтобы каждый новый объект обязательно снабжался адресом,по которому еще не произошло ни одного обращения.

Вместе с тем число наложений следует минимизировать, поскольку в большинстве приложений пользователя интересует все-таки однозначная нумерация и он вынужден принимать специальные меры для различения объектов, попадающих по одному адресу (см. п.4, стр. I7). При большом числе наложений увеличиваются затраты на поиск нужного элемента, т.е. растет трудоемкость метода.

В предположении равномерности распределения значений признака $\rho(x)$ число наложений в среднем окажется небольшим, если в качестве нумерующей функции $h(\rho(x))$ выбрать функцию, значения которой равномерно распределены внутри заданного отрезка чисел натурального ряда, определяемого размером расстановочного поля. Вводимое иногда дополнительно требование о том, чтобы малому изменению признака соответствовало большое изменение значения функции, вообще говоря, не является необходимым и не выполняется для некоторых широко используемых на практике функций расстановки (например, для метода деления)*). Нежелательные последствия группировки объектов,которая может при этом иметь место, легко обходятся путем выбора соответствующего метода устранения наложений.

*) Указанное свойство метода деления может быть использовано для построения эффективной процедуры сортировки.

В свою очередь, требование равномерности распределения значений функции расстановки ещё не является достаточным условием получения минимального числа наложений. Известно, что если k из N позиций расстановочного поля заполнены независимо случайным образом, то ожидаемое число попыток, которые потребуются для нахождения пустой позиции, составляет величину

$$C_0(k, N) = 1 + \frac{k}{N-k+1} \quad . \qquad (I.5)$$

Ульман [6] показал, что для определенных значений k и N оценка $C_0(k,N)$ может быть улучшена за счет выбора детерминированной стратегии. Тем не менее в среднем эта оценка все же дает нижнюю границу числа наложений в следующем смысле: если мы имеем произвольный алгоритм размещения, который эффективнее случайного для некоторого значения k, то существует $k' < k$, такое, что для него данный алгоритм будет менее эффективен, чем случайный.

Итак, отсутствие взаимно-однозначного соответствия между признаком объекта и адресом, по которому заносится информация об объекте, приводит к необходимости дополнения описанной выше процедуры нумерации некоторой процедурой устранения возникающих при нумерации наложений объектов. Если выбрать вторую процедуру так, что её трудоемкость удовлетворяет требованию I, то в результате объединения этих двух процедур мы получаем нумерацию, удовлетворяющую всем сформулированным требованиям.

Таким образом, под ассоциативным кодированием будем понимать удовлетворяющую свойствам I–4 двухступенчатую процедуру нумерации, первым этапом которой является неоднозначная нумерация с использованием функции расстановки, а вторым – устранение неоднозначности.

2. Вероятностная модель процедуры ассоциативного кодирования.

Схема формирования адреса, по которому информация об объекте

заносится в нужную позицию расстановочного поля, достаточно хорошо согласуется с вероятностной моделью, лежащей в основе известной "задачи о дробинках" [9].

Задача формулируется следующим образом. Дано N ящиков, в которые бросают наудачу n дробинок, так что вероятность каждой дробинке попасть в каждый ящик одна и та же, независимо от того, сколько дробинок уже попало в данный ящик. Требуется определить вероятность попадания в данный ящик ровно τ дробинок ($\tau = 0, 1, 2, \ldots, n$). Соответствующая вероятность ρ_τ имеет вид

$$\rho_\tau = C_n^\tau \left(1 - \frac{1}{N}\right)^{n-\tau} \left(\frac{1}{N}\right)^\tau, \qquad (2.1)$$

т.е. представляет частный случай биномиального распределения для τ успехов в n независимых испытаниях с вероятностью успеха $\rho = \frac{1}{N}$ в каждом испытании.

В интересующем нас случае n относительно велико, ρ мало, а произведение $\alpha = n\rho$ не мало, но и не велико. Поэтому для величины ρ_τ удобно воспользоваться известным пуассоновским приближением для биномиального распределения:

$$\rho_\tau \approx \frac{\alpha^\tau}{\tau!} e^{-\alpha}. \qquad (2.2)$$

Непосредственный интерес представляет оценка числа ящиков, в которые не попадет ни одной дробинки, попадет одна, две или более дробинок. Если обозначить через μ_τ случайную величину, равную числу ящиков, в которых содержится ровно по τ дробинок ($\tau = 0, 1, 2, \ldots, n$), то для любых факториальных моментов этих случайных величин справедливы соотношения [10]:

$$M\mu_\tau = N C_n^\tau \frac{1}{N^\tau} \left(1 - \frac{1}{N}\right)^{n-\tau}, \qquad (2.3)$$

$$M\mu_\tau(\mu_\tau - 1) = N(N-1) \frac{n!}{(\tau!)^2 (n-2\tau)!} \frac{1}{N^{2\tau}} \left(1 - \frac{2}{N}\right)^{n-2\tau}, \qquad (2.4)$$

$$M\mu_\tau \mu_t = N(N-1) \frac{n!}{\tau! t! (n-\tau-t)!} \frac{1}{N^{\tau+t}} \left(1 - \frac{2}{N}\right)^{n-\tau-t}, \quad \tau \neq t. \qquad (2.5)$$

В практически интересном случае, если $n, N \to \infty$, а $\alpha = \frac{n}{N}$ ограничено, выражения (2.3) – (2.4) переходят соответственно в

$$M\mu_r \sim N\rho_r, \quad \mathcal{D}\mu_r \sim N\sigma_{rr}, \quad COV(\mu_r,\mu_t) \sim N\sigma_{rt}, \qquad (2.6)$$

где

$$\left.\begin{aligned} \rho_r &= \frac{\alpha^r}{r!}\, e^{-\alpha}, \\ \sigma_{rr} &= \rho_r\left[1-\rho_r-\frac{\rho_r}{\alpha}(\alpha-r)^2\right], \\ \sigma_{rt} &= -\rho_r\rho_t\left[1+\frac{(\alpha-r)(\alpha-t)}{\alpha}\right]. \end{aligned}\right\} \qquad (2.7)$$

В схеме с неравными вероятностями попадания $\alpha_1, \alpha_2, \ldots, \alpha_N$ выражения (2.2) – (2.4) заменяются на следующие:

$$M\mu_r = \sum_{k=1}^{N} C_n^r\, \alpha_k^r (1-\alpha_k)^{n-r}, \qquad (2.8)$$

$$M\mu_r(\mu_r-1) = \sum_{\substack{k,\ell=1 \\ k\neq\ell}}^{N} \frac{n!}{(r!)^2(n-2r)!}\, \alpha_k^r\, \alpha_\ell^r\, (1-\alpha_k-\alpha_\ell)^{n-2r}, \qquad (2.9)$$

$$M\mu_r\mu_t = \sum_{\substack{k,\ell=1 \\ k\neq\ell}}^{N} \frac{n!}{r!t!(n-r-t)!}\, \alpha_k^r\, \alpha_\ell^t\, (1-\alpha_k-\alpha_\ell)^{n-r-t}. \qquad (2.10)$$

Выражение (2.3) определяет математическое ожидание числа ящиков, в которые попало ровно r дробинок ($r = 0, 1, 2, \ldots, n$) при фиксированном общем числе дробинок n. Представляет интерес рассмотреть также в некотором смысле обратную задачу, результаты которой понадобятся нам в дальнейшем для оценки трудоемкости метода ассоциативного кодирования. А именно, каково должно быть математическое ожидание $M n_S$ числа брошенных дробинок n_S, для того чтобы получить ровно S занятых ящиков, т.е. ящиков, в каждом из которых находится хотя бы по одной дробинке?

Следуя Феллеру [II], обозначим через x_S случайную величину, равную числу дробинок, которые необходимо бросить в ящики для того, чтобы увеличить число занятых ящиков от значения S до $(S+1)$. Тогда $n_S = 1 + x_1 + x_2 + \ldots + x_{S-1}$. Если k ящиков уже заняты, то вероятность выбора свободного ящика равна

при каждом обращении $\rho = \frac{N-k}{N}$. Распределение случайной величины x_k совпадает с распределением номера первого успеха в последовательности испытаний Бернулли $\rho = (N-k)/N$. Отсюда $Mx_k = 1+(1-\rho)/\rho = N/(N-k)$ (см. выражение (I.5)) и

$$M n_S = N\left(\frac{1}{N} + \frac{1}{N-1} + \frac{1}{N-2} + \ldots + \frac{1}{N-S+1}\right). \qquad (2.11)$$

При больших N выражение (2.II) можно приближенно заменить на

$$M n_S \approx N \ln \frac{N}{N-S}, \qquad (2.12)$$

откуда видно, что в среднем на каждый из S занятых ящиков приходится

$$T \approx \frac{N}{S} \ln \frac{N}{N-S} = -\frac{1}{\alpha_1} \ln (1-\alpha_1) \qquad (2.13)$$

дробинок, где $\alpha_1 = \frac{S}{N}$ – отношение числа занятых ящиков к общему числу ящиков.

В дальнейшем применительно к процедуре ассоциативного кодирования под N будем понимать число позиций в расстановочном поле, т.е. максимальное число объектов, которые могут быть размещены в этом поле при условии, что в каждую позицию помещается по одному объекту. Величине n соответствует реальное число объектов, размещаемых в расстановочном поле.

3. Методы построения функций расстановки. Требования, предъявляемые к функции расстановки, были сформулированы в п.I, стр. 5. В данном разделе приведены примеры построения некоторых широко используемых на практике функций расстановки.

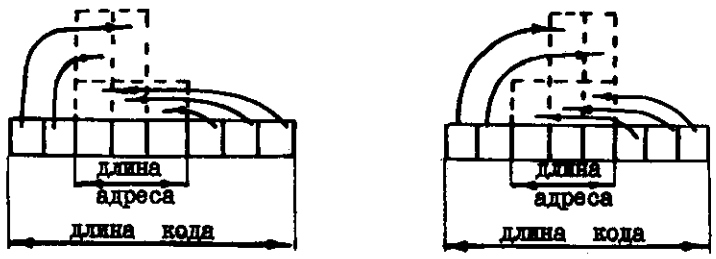
3.I. Метод анализа разрядов [I2]. Значения признаков, которыми обладают элементы информационного массива и в соответствии с которыми происходит их адресация, представляются в виде числового кода, после чего анализируется распределение содержимого каждой позиции кода по всему множеству этих кодов.

Для целей адресации используются те позиции признаков, которые характеризуются наиболее равномерным распределением значений по всем кодам. Позиции, характеризующиеся малой дисперсией своих значений или аномальными отклонениями математического

ожидания от середины диапазона возможных изменений признака в данной позиции, вычеркиваются. Предполагается, что вычеркиваются из всех кодов одни и те же разряды, пока число оставшихся разрядов не станет равным желаемой длине адреса.

К недостаткам метода следует отнести необходимость предъявления всех элементов информационного массива одновременно. Метод непригоден в случае, если предъявляется по одному элементу информационного массива, причем сразу требуется указать адрес элемента. Кроме того, знание распределений значений признака в каждой из позиций еще не гарантирует нам в описанной модификации метода получения идеально равномерного распределения адресов. Существуют более совершенные преобразования, которые отличаются, однако, значительно большей трудоемкостью.

3.2. Метод свертки [2, I2]. Числовой код, характеризующий признак элемента информационного массива, делится на несколько частей, каждая из которых, за исключением, быть может, последней, имеет длину, равную требуемой длине адреса. Выделенные части накладываются друг на друга путем совмещения сдвинутых частей кода (fold - shifting), либо с помощью зеркального отражения (fold - boundary). Разряды, попадающие в одно и то же место, складываются по модулю ρ, где ρ - основание используемой системы счисления. Рис. I иллюстрирует действия над позициями в обоих методах свертки.



Зеркальное отражение Сдвиг

Рис. I

3.3. Метод середины квадрата [I3]. Числовой код признака умножается на константу или сам на себя и из произведения вырезаются k средних разрядов, которые образуют адрес. Величина k определяется количеством объектов n, размещаемых в памяти.

Метод середины квадрата неприменим, если коды содержат много нулей. К недостаткам методов свертки и середины квадрата следует отнести не совсем рациональное использование памяти, которая выделяется блоками размером 2^k. Может случиться так, что незначительное увеличение количества n размещаемых объектов потребует двойного увеличения памяти.

3.4. Метод деления [I4]. Код признака делится на положительное целое число M. Остаток от деления принимается в качестве адреса элемента информационного массива, снабженного признаком k: $h(K) = K \bmod M$.

В качестве делителя M выбирается число, близкое к объему памяти S, отводимому под размещаемый информационный массив ($M \leq S$). При этом M должно быть таким, чтобы обеспечивалось по возможности наиболее равномерное распределение адресов внутри расстановочного поля. Например, не рекомендуется брать четное M, поскольку при этом четным K соответствуют лишь четные $h(K)$ и нечетным K - нечетные $h(K)$. Число M не рекомендуется брать кратным 3, поскольку $10^n \bmod 3 = 4 \bmod 3 = 1$ и т.д. На практике обычно в качестве M используется наибольшее простое число, меньшее или равное S. В [I2] указывается, что делитель может быть и составным числом, но сомножители должны быть достаточно велики (скажем, порядка 20 и выше).

Вместо деления можно воспользоваться умножением на обратное число. Эта модификация метода удобна для реализации на ЭВМ, у которых умножение выполняется существенно быстрее, чем деление.

К достоинствам метода следует отнести его простоту и рациональное использование памяти. В качестве недостатка иногда указывают на тот факт, что малому изменению признака в этом методе соответствует малое изменение адреса. Однако при более внимательном рассмотрении этот факт оказывается несущественным, если выбран подходящий метод устранения наложений.

3.5. Метод Л и н а [I5]. Десятичный код признака по-разрядно переводится в двоичную форму. Разряды полученной двоичной цепочки группируются в ρ-ичные числа. Результат выражается десятичным числом, которое, взятое по модулю q^m, дает адрес. Последняя операция является, очевидно, аналогом метода деления для случая $M = q^m$, отсюда q и m выбираются так, чтобы $q^m \approx S$. Первые две операции представляют, по-видимому, попытку сделать дополнительный случайный шаг, нацеленный на получение более равномерного распределения адресов. Для упрощения выбора ρ рекомендуется брать $\rho = q + 1$.

Проиллюстрируем метод на примере. Предположим, что числовой код признака K = 975. Перекодируя каждую цифру четырьмя разрядами (наименьшее число бит, требующееся для представления десятичной цифры), получаем двоичный ряд I00I0III0I0I. Теперь, если число имеющихся адресов равно 48, выбор $\rho = 8$, $q = 7$, $m = 2$ соответствует описанному правилу. Группируя по 3 бита, имеем $4565_8 = 2421_{10}$. Вычисляя $2421 \bmod 49$, получаем адрес 20.

Следует отметить, что влияние первого шага на окончательный результат не исследовано. Выбор незначительно изменяющихся ρ может резко изменить результаты. Выбор же $M = q^m$ противоречит требованию о неделимости M, сформулированному в п. 3.4.

3.6. Метод а л г е б р а и ч е с к о г о к о д и р о в а н и я [I6,I7]. В этом методе каждый разряд числового кода признака $K = (k_{n-1} \dots k_1 k_0)$ рассматривается как коэффициент полинома $K(x) = \sum_{i=0}^{n-1} k_i x^i$. Полученный таким образом полином делится на подобранный соответствующим образом другой полином $\mathcal{Y}(x) = \sum_{i=0}^{W-1} q_i x^i$. Коэффициенты α_i полинома $A(x) = \sum_{i=0}^{\alpha-2} \alpha_i x^i$, получающегося в остатке, образуют адрес $A = (\alpha_{W-2} \dots \alpha_1 \alpha_0)$, по которому помещается информационный элемент, снабженный признаком K. Делитель $\mathcal{Y}(x)$ берется одинаковым для всех элементов информационного массива.

Метод алгебраического кодирования основан на теории кодов, исправляющих ошибки [I8]. Предполагается, что если сконструировать функцию, отображающую множество ключей (признаков) во множество адресов таким образом, что при этом близким, скажем, в хэмминговой метрике, значениям ключей будут соответствовать достаточно далекие значения адресов, то такая функция будет обладать хорошими рандомизирующими свойствами.

Если определять расстояние α между двумя ключами числом позиций, по которым они отличаются, то соответствующая задача в теории кодирования формулируется следующим образом. Найти максимальное расстояние W и функцию T, отображающую множество ключей во множество адресов, так, чтобы для любых различных ключей U и V равенство $T(U) = T(V)$ было бы возможно лишь при условии $\alpha(U, V) \geq W$. Таким образом, преобразование T должно разделять любые два ключа, отличающиеся не более чем на $(W-1)$ позицию.

Сформулированную задачу приходится решать при построении кодов Боуза-Чоудхури. Коэффициенты делимого $K(x)$ и делителя $\mathcal{Y}(x)$ должны являться при этом элементами поля Галуа $GF(q)$, состоящего из $q = \rho^m$ элементов, где ρ – простое число. В частном случае $q = 2$ многочлен $\mathcal{Y}(x)$ над $GF(2)$ представим в виде

$$\mathcal{Y}(x) = (x - \alpha)(x - \alpha^2) \dots (x - \alpha^{W-1}) = \sum_{i=0}^{W-1} q_i x^i,$$

где α – примитивный элемент поля $GF(2^m)$ (т.е. $\alpha^{2^m - 1} = 1$, но $\alpha^i \neq 1$ для $i < 2^m - 1$). Теорема Боуза-Чоудхури утверждает при этом, что если $A(x)$ есть остаток от деления $K(x)$ на $\mathcal{Y}(x)$ (используется полиномиальная арифметика по модулю 2 [I9]), то минимальное расстояние между двумя ключами, дающими одинаковое значение A, не меньше, чем W.

Процедура определения делителя $\mathcal{Y}(x)$ в общем случае довольно громоздка, правда, эта операция разовая. Деление полинома на полином можно осуществлять на ЭВМ или с использованием специального сдвигового регистра [I8].

4. М е т о д ы у с т р а н е н и я н а л о ж е н и й. Как уже указывалось ранее, принципиальной особенностью метода ассоциативного кодирования является его неоднозначность, т.е. возможность попадания в одну позицию расстановочного поля двух и более различных объектов. В большинстве приложений, однако, пользователю необходимо как-то отличать между собой наложившиеся объекты, в результате чего возникает задача устранения наложений. Эта задача, так же как и задача формирования адреса, является неотъемлемой частью процедуры ассоциативного кодирования, и её неудачное решение может существенно повлиять на трудоемкость этой процедуры.

Существующие методы устранения наложений можно разбить на две группы. I-я группа методов, получившая в американской литературе название "открытой адресации" (open addressing), предполагает, что при появлении наложения осуществляется поиск свободной позиции в пределах того же поля, к которому происходит обращение в соответствии с используемой функцией расстановки. Наложившийся объект заносится затем в обнаруженную свободную позицию. Различные методы этой группы отличаются друг от друга алгоритмами поиска свободной позиции. Принципиальной особенностью методов данной группы является идентичность процедур заполнения расстановочного поля и поиска информации об объекте в этом поле.

Вторая группа методов использует списковую организацию памяти для устранения наложений. С этой целью каждая позиция расстановочного поля делится на две части - информационную и адресную. В первую часть заносится информация о самом первом объекте, который был распределен по данному адресу. Во второй части при наличии повторного обращения по данному адресу в явном виде указывается адрес позиции, куда помещается наложившийся объект. При наличии трехкратного наложения второй объект в свою очередь, снабжается указателем перехода на третий объект и так далее.

Поиск свободной позиции для записи наложившегося объекта может осуществляться любым из методов I-й группы, либо путем использования дополнительной памяти, о чем пойдет речь ниже. Извлечение же нужного объекта из памяти осуществляется всегда путем просмотра сформированного списка. Таким образом, в методах данной группы процедуры заполнения расстановочного поля и поиска информации в нем не являются идентичными. Учитывая разовый характер первой процедуры, некоторым усложнением её добиваются максимальной простоты второй процедуры за счет относительно небольших издержек памяти.

4.I. М е т о д ы, и с п о л ь з у ю щ и е о т к р ы т у ю а д р е с а ц и ю.

4.I.I. Простейший л и н е й н ы й метод [20] отыска - ния свободной ячейки заключается в следующем: при наложении идем в любую сторону от вычисленного адреса, последовательно проверяя все позиции расстановочного поля, пока не дойдем до

ожидаемого кода (при поиске) или до пустой позиции (при размещении). Память при этом рассматривается как замкнутая цепочка, т.е. после просмотра последней позиции осуществляется переход на первую позицию, или наоборот.

Рассмотренная стратегия поиска является наименее эффективной с точки зрения среднего числа проверок, требуемых для извлечения объекта. Это объясняется тем, что устранение наложений таким способом приводит к группировке объектов внутри расстановочного поля, и если новый объект попадает в начало группы, то требуется осуществить много проверок, чтобы дойти до пустой позиции.

Можно осуществлять линейный метод с шагом i , т.е. просматривать ячейки $l+i$, $l+2i$,... (l - первоначальный адрес, вычисленный по функции расстановки). В этом случае также происходит группировка объектов, но гораздо медленнее.

Трудоемкость линейного метода, определяемая средним числом проверок, требующихся для поиска объекта, в случае больших N и n может быть представлена в виде [2I]:

$$T = \left(1 - \frac{\alpha}{2}\right) / (1 - \alpha) , \qquad (4.1)$$

где $\alpha = \frac{n}{N}$ - коэффициент заполнения расстановочного поля.

4.I.2. В к в а д р а т и ч н о м методе [I3] шаг i , с которым осуществляется поиск свободной позиции в расстановочном поле, является уже квадратичной функцией номера попытки k (a и b константы):

$$i = a \cdot k + b \cdot k^2 , \quad k = 1, 2, 3, \ldots . \qquad (4.2)$$

В основе следующих трех методов лежит одна и та же идея, обобщающая линейную и квадратичную процедуру поиска, а именно: для устранения возможности группировки объектов следует вносить элемент случайности в процедуру поиска свободной позиции. Фактически это эквивалентно тому, что наложившиеся объекты вновь случайным образом распределяются внутри расстановочного поля, пока для каждого из них не будет обнаружена свободная позиция.

4.3.I. М е т о д В ы с о ц к о г о [22] состоит в использовании набора функций расстановки для устранения наложений.

Алгоритм имеет следующий вид:

- вычисляется адрес $h(K)$ объекта K использованием какой-либо функции расстановки;

- если по вычисленному адресу уже расположен искомый объект или соответствующая позиция пуста, то процедура закончена;

- если по вычисленному адресу расположен другой объект, то определяется новый адрес с использованием следующей функции набора и так далее, пока запас функций расстановки не будет исчерпан;

- если запас функций расстановки исчерпан, то осуществляется линейный поиск по всей памяти, пока не будет найден искомый объект или пустая позиция.

Для получения набора функций расстановки вычисляется n - разрядная функция, у которой только m первых разрядов используются под адрес. Последующие функции расстановки образуются путем циклического сдвига исходной функции и использования тех же самых m первых разрядов. После n сдвигов набор исчерпывается и осуществляется переход к линейному поиску.

4.I.4. Стратегия поиска, основанная на использовании датчика псевдослучайных чисел [23], такая же, как и в методе линейного поиска с шагом i, но i - псевдослучайное число. Генератор псевдослучайных чисел должен по разу пробежать все целочисленные значения в диапазоне от I до $N-1$. Если пустая позиция не найдена и после этого, то это означает, что таблица полна и вводить объект некуда.

Возможный датчик псевдослучайных чисел для $N=2^n$ имеет вид:

$$\begin{cases} -\ k := 1\ ; \\ -\ k := k \times 5\ ; \\ -\ k := k \bmod 2^{n+2}\ ; \\ -\ i := \mathit{entier}\ \frac{k}{4}\ . \end{cases}$$

4.I.5. При модифицированном квадратичном методе [24] новый адрес для наложившегося объекта предлагается вычислять по формуле

$$h(K) = h_0(K) + a \times k + b(K) \times k^2, \qquad (4.3)$$

т.е. в отличие от выражения (4.2) коэффициент b уже не является константой, а определяется числовым кодом признака.

Первоначальный адрес объекта $h_0(K)$ предлагается вычислять, используя метод деления. В качестве $b(K)$ рекомендуется использовать $\mathit{entier}\,(K/S)$, где S - используемый объем памяти.

4.2. Методы, использующие списковую организацию памяти. Типичная схема реализации метода, не предусматривающая использования дополнительной памяти, описана в работе [23].

Процедура поиска нужного элемента сводится к вычислению его функции расстановки $h(K)$ и проверке содержимого соответствующей позиции. Если позиция свободна, то данного элемента нет в памяти. Если позиция занята другим кодом K', то просматривается список, начинающийся с этой позиции, и в случае отсутствия в нем нужного элемента поиск прекращается.

Процедура заполнения расстановочного поля также начинается с вычисления функции расстановки для размещаемого в этом поле объекта, а затем:

- если соответствующая позиция пуста, то объект вводится туда;

- если позиция занята заголовками списка, то просматривается весь список, и в случае отсутствия в нем данного объекта определяется свободная позиция, например, любым из методов, рассмотренных в разделе 4.I, и в эту позицию заносится объект, который и становится последним элементом списка (одновременно в предпоследний элемент списка заносится адрес последнего элемента);

- если в позиции записан объект из другого списка, не являющийся заглавным, то старый объект должен быть передвинут на другую свободную позицию, а новый помещен на его место. Это требует нахождения свободной позиции, записи туда объекта и корректировки списка.

Последнее обстоятельство представляет принципиальное неудобство. В работе [25] описан метод, в котором предлагается в аналогичной ситуации не производить перемещения первоначально записанного объекта, а включить новый объект в этот же список.

Гораздо эффективнее можно организовать процедуру заполнения расстановочного поля при наличии некоторой дополнительной

памяти, выделяемой еще до того, как заполнена основная память. Дополнительная память отводится только под наложившиеся объекты, которые заполняют её последовательно, позиция за позицией, по мере возникновения наложений в основной памяти. При этом отпадает задача поиска свободной позиции в случае наложения, поскольку роль таковой играет очередная по порядку незанятая позиция дополнительного поля. Все, что говорилось о списковой организации памяти, при этом остается в силе.

Оценка дополнительной и общей памяти, требующейся для реализации данной процедуры, может быть получена на основании соотношения (2.3). При $\tau = 0$ из него получаем математическое ожидание числа позиций основного поля, оказавшихся свободными в результате n обращений к этому полю:

$$M\mu_0 = N\left(1 - \frac{1}{N}\right)^n \approx N e^{-\alpha}, \qquad (4.4)$$

где $\alpha = \frac{n}{N}$ – коэффициент загрузки памяти (при наличии дополнительной (сверх N единиц) памяти α может быть больше I). Тогда оставшиеся $N(1 - e^{-\alpha})$ позиций основного поля будут заняты, и поскольку в каждой занятой позиции может располагаться лишь по одному объекту, то для $n - N(1 - e^{-\alpha})$ объектов понадобится дополнительная память. Таким образом, ожидаемая величина общей памяти равна

$$S \approx N + (n - N(1 - e^{-\alpha})) = N(\alpha + e^{-\alpha}).$$

Полученная оценка совпадает с оценкой, приведенной у Морриса [23].

Зачастую количество объектов n, которые требуется разместить в памяти, известно заранее. Коэффициент же загрузки α выбирается на основе разумного компромисса между имеющимися ресурсами памяти S ($S > n$) и допустимым временем поиска. Действительно, с одной стороны, увеличение α при фиксированном n приводит к некоторому уменьшению потребной памяти S; с другой стороны, это эквивалентно уменьшению размеров основного поля N, что при фиксированном n приводит к увеличению средней длины списка, а следовательно, и времени поиска.

Организация списковой структуры, несколько отличная от рассмотренной выше, описана в работах [23, 26, 27] (см. также

пример, приведенный в п.6.I. стр. 28). Она характеризуется тем, что информация об объектах отделена от вспомогательной адресной информации, используемой для построения списка, что представляет известное удобство при отсутствии единого стандарта на формат элементов информационного массива. Эта вспомогательная информация, к примеру, в [23] фигурирует в виде специальной расстановочной индексной таблицы, а в [26] в виде трех векторов c, α, W. В α записаны адреса объектов, являющихся заглавными в списках, в c записаны адреса переходов при наложениях, а W получен вторичным применением к признаку (или объекту) новой функции расстановки и содержит адреса, по которым происходит обращение к вектору α.

5. Теоретические и экспериментальные оценки трудоемкости методов ассоциативного кодирования. Трудоемкость методов ассоциативного кодирования определяется в основном средним числом проверок (сравнений), требующихся для записи и извлечения объекта из памяти. При этом предполагается, что сама функция расстановки вычисляется достаточно просто, а затраты на занесение объекта в память, когда они носят разовый характер, в расчет не принимаются.

Оценка трудоемкости каждого метода существенным образом определяется способом устранения наложений, используемым в данном методе. Ранее все методы были разделены на 2 группы, одна из которых использует открытую адресацию, а другая – списковую организацию памяти. Для каждой из этих групп методов можно получить единую оценку трудоемкости. Исключение из I-й группы составляет лишь линейный метод, оценка трудоемкости которого приведена выше без вывода (выражение (4.I)), а из 2-й группы – косвенный списочный метод [26], являющийся, так же как и линейный, несколько более трудоемким сравнительно с другими методами своей группы.

Переходя к оценке трудоемкости методов первой группы, замечаем, что, собственно говоря, эта оценка нами уже получена (выражение (2.I3)). Это следует, во-первых, из того, что устранение наложения посредством нового случайного перераспределения объекта, характерное для методов этой группы, эквивалентно

бросанию очередной дробинки в описанной выше модели явления, а, во-вторых, из идентичности процедур заполнения расстановочного поля и поиска объекта в нем, являющейся отличительной особенностью методов данной группы.

Таким образом, параметру S в модели соответствует число объектов n, размещаемых в расстановочном поле; числу бросаемых дробинок n_S соответствует число попыток, необходимых для размещения n объектов; среднему числу дробинок на ящик соответствует среднее число проверок на объект, т.е. трудоемкость метода T_1:

$$T_1 = -\frac{1}{\alpha}\ln(1-\alpha), \qquad (5.1)$$

где $\alpha = \frac{n}{N}$ – коэффициент загрузки памяти.

В методах второй группы поиск нужного объекта осуществляется по списку, поэтому в качестве меры трудоемкости обычно используется отношение среднего по всем спискам числа проверок, необходимых для извлечения информации обо всех объектах списка, к среднему числу объектов в списке [26]:

$$T_2 = \frac{\sum_r t_r \rho_r}{\sum_r r \rho_r}, \qquad (5.2)$$

где ρ_r – вероятность того, что список содержит ровно r объектов, а t_r – число проверок, необходимых для извлечения информации об этих r объектах.

Нетрудно видеть, что для извлечения информации об i-м объекте списка ($i = 1,2,\ldots,r$) потребуется i проверок, следовательно, затраты на весь список, содержащий r объектов, составят

$$t_r = \frac{r(r+1)}{2} \qquad (5.3)$$

проверок. В соответствии с используемой нами моделью вероятность ρ_r того, что список содержит r объектов, может интерпретироваться как приведенная в п.2 (формула (2)) вероятность попадания в ящик ровно r дробинок. Воспользовавшись известными соотношениями для первых двух моментов пуассоновского распределения,

$$\sum_r r \rho_r = \alpha, \qquad (5.4)$$

$$\sum_r r^2 \rho_r = \alpha(\alpha+1), \qquad (5.5)$$

для трудоемкости T_2 методов второй группы окончательно получаем следующее выражение:

$$T_2 = \frac{1}{2\alpha}\sum_r (r^2+r)\rho_r = \frac{1}{2\alpha}\left[\alpha(\alpha+1)+\alpha\right] = 1+\frac{\alpha}{2}. \qquad (5.6)$$

Полученное выражение справедливо и при $\alpha > 1$ (при этом подразумевается использование дополнительной памяти). Сравнение формул (5.1) и (5.6) показывает, что списочные методы устранения наложений гораздо эффективнее методов, использующих открытую адресацию. Так, например, при коэффициенте загрузки $\alpha = 1$ списочный метод требует в среднем около 1,5 проверок для извлечения объекта, в то время как при открытой адресации наблюдается резкое снижение эффективности поиска при $\alpha \to 1$. Для иллюстрации ниже приведена таблица теоретических оценок трудоемкости для различных методов устранения наложений.

Таблица 1

Зависимость трудоемкости различных методов устранения наложений от коэффициента загрузки памяти

Коэффициент загрузки памяти	Метод устранения наложений			
	линейный	квадратичный	случайный, квадратичный, модифицированный метод Высоцкого	списочный с дополнительной памятью
0,1	1,06	–	1,05	1,05
0,5	1,5	1,44	1,38	1,25
0,75	2,5	1,9	1,83	1,38
0,9	5,5	2,79	2,55	1,45
1,5	–	–	–	1,75
2	–	–	–	2

Хороший сравнительный обзор различных методов устранения наложений содержится в работе [23]. В [I2] приведены результаты весьма детальных исследований по экспериментальной проверке эффективности различных методов построения функций расстановки. Критерием для сравнения при этом служил средний процент наложений для каждого преобразования и среднеквадратичный разброс результатов, характеризующий устойчивость метода к различным выборкам.

Среди методов построения функций расстановки на первое место поставлен метод деления. Почти на всех выборках он дал лучшие результаты, чем метод полной рандомизации. Автор не дает своей трактовки этого эффекта, ограничившись замечанием о том, что полная рандомизация не является желаемой целью и что она не является синонимом равномерности получающегося распределения адресов (см. п.I).

На втором месте стоит алгебраический метод, который хуже метода деления, но также дает несколько лучшие результаты, чем метод полной рандомизации. Однако этот метод довольно сложен с вычислительной точки зрения. Сторонники этого метода полагают, что хорошие характеристики метода связаны с оптимальным выбором делителя $\mathcal{Y}(x)$. Проведенный эксперимент [I2] показал, однако, что не существует никакой связи между параметром W и качеством преобразования. Выбор различных W давал примерно одинаковые результаты. Более того, близкие к теоретическим оценкам результаты дал случайный выбор делителя с единственным ограничением на то, чтобы коэффициент при высшей степени полинома и свободный член не были равны нулю. Выбор поля Галуа тоже не слишком принципиален. Использование полей $\mathcal{Y}F(2)$ и $\mathcal{Y}F(16)$ дало очень близкие результаты.

Метод середины квадрата дает хорошие результаты, близкие к теоретическим оценкам, при коэффициенте загрузки $\alpha < 0{,}75$. На некоторых выборках, однако, наблюдается резкое ухудшение результата. Обычно это соответствует случаю, когда код признака длинный, а адреса короткие и когда вариация средних разрядов кода незначительна.

Наименьшего времени вычисления требует обычно метод свертки. При длине кода признака, близкой к длине кода адреса, метод свертки ведет себя как метод деления. Однако в общем метод отличается большой неустойчивостью.

Метод поразрядного анализа применять не рекомендуется. Результаты по обоим показателям оказываются неудовлетворительными.

И, наконец, метод Лина по сравнительным показателям на одних и тех же выборках оказался самым худшим из 6 перечисленных выше методов. Возможные причины этого указаны в разделе 3.5.

Выбор метода устранения наложений зависит от наличия либо отсутствия дополнительной (сверх N единиц) памяти. При наличии таковой выбор очевиден – списочный метод с использованием дополнительной памяти под наложившиеся объекты.

Отсутствие дополнительной памяти означает, что при заполнении имеющейся памяти мы должны воспользоваться одним из методов открытой адресации для поиска свободной ячейки при наложении. При этом наложившиеся объекты либо увязываются в список и тогда в дальнейшем их поиск осуществляется по списку, либо не увязываются и тогда их поиск осуществляется тем же методом, который использовался для занесения объекта в память.

Поиск по списку, как было показано выше, эффективнее других методов, поэтому, как правило, даже при отсутствии дополнительной памяти имеет смысл наложившиеся объекты внутри основной памяти объединять в список. В ряде случаев, однако, это делать нецелесообразно, например, в случае малых по размеру объектов или при быстрой смене информационных массивов или в задачах, где допускается неоднозначность (см. пример в п.6.3). В первом случае память, требующаяся для организации списковой структуры, оказывается сравнимой с памятью, отводимой под сами объекты. Во втором случае основные затраты идут на размещение информационных массивов в памяти, а не на поиск в них. И, наконец, в третьем случае от процедуры ассоциативного кодирования используется лишь её первая часть – неоднозначная нумерация.

Сравнение конкретных методов открытой адресации показывает, что самый простой в программировании, но и наиболее трудоемкий из них – линейный метод. Трудоемкость остальных методов данной группы оказывается ниже, чем у линейного метода, но выше чем у списковых методов. Трудоемкость линейного метода можно заметно уменьшить, если рассматривать не отдельные объекты, а группы объектов ("buckets"). Такая группировка целесообразна при организации поиска с использованием внешней памяти (барабанов,

дисков), когда требуется минимизировать число обращений к этой памяти. Для групп большого размера (20-50 объектов) списочный и линейный методы оказываются равно удовлетворительными.

6. Примеры использования метода ассоциативного кодирования.

6.1. Системное программирование. В качестве первого примера рассмотрим организацию ассоциативного кодирования в интерпретаторе для БЭСМ-6 со спискового языка ЛИСП [29]. Ассоциативное кодирование используется здесь для нахождения внутреннего наименования "атомов" по их внешнему (посимвольному) обозначению. "Атомы" являются наиболее простыми объектами данного языка. Это - либо числа, либо имена, составленные из последовательности алфавитно-цифровых символов. Каждый атом представлен списком своих свойств (1 ячейка), содержащим, в частности, указатель внешнего наименования атома, хранящегося отдельно. Внутреннее наименование атома представляет адрес ячейки, содержащей список свойств атома.

Рассматриваемая система является "открытой", т.е. информация об общем числе и виде атомов, с которыми системе придется работать, отсутствует. Вначале в ней присутствуют только атомы, включенные разработчиком. В процессе работы к ним подключаются атомы, вводимые пользователем и кодируемые системой в том же формате. Использование ассоциативного кодирования позволяет при поиске нужного атома заменить линейный просмотр всех атомов просмотром относительно короткого списка наложившихся атомов. "Открытый" характер системы ведет к некоторой её неоптимальности, поскольку расширение системы осуществляется за счет привлечения дополнительной свободной памяти при фиксированной основной. Это приводит к увеличению средней длины списка и, следовательно, среднего времени поиска.

Функция расстановки формируется на основе логических операций. Аргументом функции является числовой код K, образованный из последовательности восьмиразрядных символов, составляющих внешнее наименование атома. Код $H(K)$ (начальное значение его равно 0) формируется за один цикл опроса этих символов. В теле цикла текущее значение кода $H(K)$ сдвигается влево на один разряд и циклически складывается с очередным восьмиразрядным

символом. К результату циклически прибавляется константа, вызывающая транспортировку избыточной сверх восьми разрядов единицы переноса (если таковая окажется) в младшие разряды сумматора. Лишняя (сверх восьми младших бит) информация обрезается, и полученный код служит текущим значением $H(K)$. По выходе из цикла накопленное значение сдвигается на один разряд вправо и выдается в качестве семиразрядного адреса рассматриваемого атома.

Для устранения наложений используется списочный метод с дополнительной памятью. Информация об объектах (список свойств атома и его посимвольное обозначение) отделена от вспомогательной информации, используемой для организации списка. Размер основной памяти фиксирован ($N = 128$ ячеек). В каждую из ячеек помещается по 2 указателя: на список свойств атома и на очередной из наложившихся атомов. В начальный момент все ячейки содержат указатель на атом NIL, обозначающий в системе понятие "пустой список".

Имя любого атома с помощью функции расстановки отображается в адрес c одной из N ячеек расстановочного поля. Если первым указателем этой ячейки является NIL, то по данному адресу еще не происходило обращений и необходимо пришедший атом включить в систему. Делается это путем обращений к свободной памяти системы.

Вся свободная память системы увязана в единый список (или несколько списков - для разнотипных объектов). Адрес первой ячейки списка хранится, например, в индекс-регистре. Каждая ячейка списка содержит указатель на очередную (её адрес). В последней ячейке списка содержится указатель NIL. При возникновении потребности в дополнительной памяти выбирается свободная ячейка по адресу, указанному в индекс-регистре (если там не стоит NIL), а находящийся в ней указатель на очередную ячейку свободной памяти передается в индекс-регистр. При наличии в индекс-регистре указателя NIL в работу включается программа, собирающая освободившиеся ячейки в новый список, либо происходит сигнализация об отсутствии свободной памяти и выход из интерпретатора.

В нашем примере полученная свободная ячейка занимается под список свойств вводимого атома, а адрес этой ячейки зано-

сится в качестве первого указателя в ячейку τ. Вторым указателем является NIL, свидетельствующий о том, что наложившихся атомов пока нет. Для внешнего наименования атома выделяется нужное количество подряд расположенных ячеек, указатель на первую из которых входит в список свойств.

Если по адресу τ происходит очередное обращение, то первым указателем в ячейке τ будет уже не NIL. По списку свойств уже присутствующего в системе атома извлекается его посимвольное наименование и сравнивается с именем вводимого атома. При совпадении имен новый атом не вводится в систему, а выдается лишь его внутреннее наименование. При несовпадении имен анализируется второй указатель ячейки τ. Если это NIL, то список наложившихся атомов исчерпан и его нужно продолжить, включив пришедший атом в систему и изменив соответствующим образом второй указатель ячейки τ. Если это не NIL, то список наложившихся атомов не исчерпан. Нужно обратиться по указанному адресу, проанализировать первый указатель (указатель списка свойств очередного атома), выявить его посимвольное наименование, осуществить сравнение и т.д.

6.2. Покажем возможность применения ассоциативного кодирования для построения решающего правила в задаче распознавания образов. Будем считать, что задан список образов S и признаковое пространство X, в котором эти образы следует отличать друг от друга, т.е. согласно терминологии, используемой в [30], мы имеем дело с задачей типа I. Перечислим некоторые методы построения решающих функций.

Метод линейных решающих функций сводится к разделению пространства X гиперплоскостями на части, каждая из которых содержит лишь один образ. Если граница между образами имеет сложную форму, то для разделения образов требуется большое число гиперплоскостей, которое в принципе невозможно сократить ниже порога $log_2 S$, где S - число образов. Рекомендации по сокращению числа гиперплоскостей содержатся в работах [30,31].

В методе дробящихся эталонов [30] решающее правило состоит в перечне простейших элементов (гиперофер, гиперпараллелепипедов), которые охватывают реализации каждого образа, и зазоров внутри них. Каждый образ может иметь при этом несколько эталонов, и чем сложнее граница между образами, тем большее число эталонов требуется для описания каждого образа.

Аналогично в методе, использующем функции принадлежности [32], число эталонов одного образа может быть больше единицы. В обоих последних методах контрольный объект относится к ближайшему (в смысле введенной метрики) эталону, т.е. для принятия решения необходимо вычислить расстояния до всех эталонов и определить минимальное из них. Увеличение числа эталонов улучшает представительность выборки и повышает качество распознавания, однако объем вычислений при этом сильно возрастает. Применение метода ассоциативного кодирования позволяет снять вопрос о вычислительных трудностях для данного класса задач.

Установим однозначный способ разбиения пространства X на параллелепипеды, зафиксировав положение их центров и длин сторон. Размер последних определяется степенью близости реализаций различных образов в данном участке пространства либо точностью измерения признаков. В результате разбиения каждой точке пространства оказывается поставлен в соответствие параллелепипед, и принадлежность контрольного объекта, попавшего в эту точку, к образу определяется эталонами, находящимися в данном параллелепипеде (предполагается, что геометрически близкие точки принадлежат реализациям одного образа - гипотеза "компактности").

Эта задача быстро решается с помощью метода ассоциативного кодирования. По набору числовых признаков контрольного объекта $\tilde{x}$ определяется соответствующий ему параллелепипед и вычисляется значение функции расстановки для координат его центра. По вычисленному адресу хранится информация о значении наборов признаков эталонов, лежащих в данном параллелепипеде (для идентификации параллелепипеда при возможной неоднозначности нумерации) и сведения о принадлежности эталонов к какому-либо образу.

Проиллюстрируем изложенное примером, изображенным на рис.2 (двумерный случай). На рисунке введены следующие обозначения: O - первый образ, x - второй образ, $?$ - контрольная реализация.

Зафиксируем разбиение плоскости на прямоугольники с центрами, координаты которых кратны 6 по оси X_1, и 4-по оси X_2. В качестве функции расстановки выберем остаток от деления на 37 по модулю I6 числа, составленного из координат центра соответствующего прямоугольника. Для точки с координатами $X_1 = I7$, $X_2 = I7$ имеем $H(I8I6) = 3$, для $X_1 = I2$, $X_2 = I7{,}5$ $H(I2I6) = 0$,

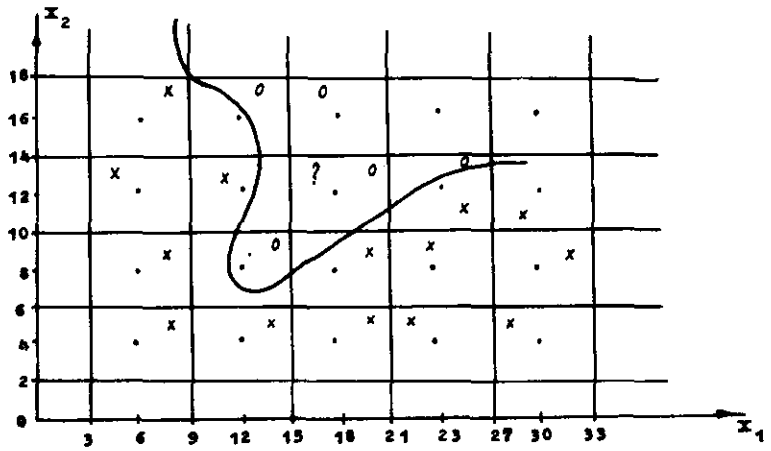


Рис. 2

для $X_1 = 20$, $X_2 = 13$ $H(1812) = 4$. По координатам контрольной реализации $X_1 = 16$, $X_2 = 12{,}5$ находим $H(1812) = 4$. В 4-й позиции расстановочного поля имеется информация о принадлежности эталона с признаками 1812 к первому образу.

Предложенный пример иллюстрирует простейший случай. Возможна более сложная конструкция – решения принимаются не только по параллелепипеду, включающему в себя контрольный объект (он может оказаться пустым, например (24,16)),но и по соседним, центры которых отстоят от контрольного объекта на расстояние, не большее ε [33,34]. Вычисление адресов кубиков,входящих в ε – окрестность контрольной реализации, в конкретном примере [33,34] по трудоемкости оказалось примерно эквивалентным вычислению трех евклидовых расстояний в рассматриваемом признаковом пространстве,причем трудоемкость этой процедуры не зависит ни от числа образов, ни от числа эталонов.

В ε – окрестности объекта могут находиться несколько,один или ни одного эталона. Поэтому принципиальной особенностью метода является возможность неоднозначных решений и отказов от принятия решения (неоднозначность самой нумерации устраняется методами, описанными выше). В некоторых задачах такая ситуация

является вполне допустимой.Это относится к использованию ассоциативного кодирования на промежуточных ступенях иерархической системы распознавания. Возникающая неоднозначность при этом может быть устранена на более высоких уровнях принятия решений. Для устранения неоднозначности в ситуациях, где она недопустима, можно рекомендовать вариацию величины ε или полный перебор эталонов в выделенной окрестности.

Описанный подход был применен в задаче распознавания фонем при обработке речевого сигнала [33,34]. Число образов (фонем) S составило 37, число их различных эталонов ~ 1100. Распознавание принадлежности участков речевого сигнала к фонемам занимало на ЭВМ БЭСМ-6 около 7% реальной длительности сигнала с надежностью около 75%.Среднее число фонемных решений на каждый участок с непустой ε – окрестностью – около 2. Полученные надежность и быстродействие оказались достаточными для требуемых приложений.

6.3. Рассмотрим задачу поиска **неповторяющихся** элементов. Имеется последовательность из N элементов,среди которых m_N элементов являются различными. Требуется построить эффективный алгоритм поиска элементов, входящих в эту последовательность ровно один раз (одиночных),для случая,когда

$$S < m_N << \mathcal{D} \qquad (6.1)$$

и

$$m_N \approx N. \qquad (6.2)$$

Здесь S – объем оперативной памяти в битах,отведенный под данную задачу, а $\mathcal{D}$ – диапазон изменения числовых значений кодов элементов последовательности.

Условие (6.1) означает, что ограничение на оперативную память не позволяет обработать последовательность путем однократного просмотра всех её членов. Более того, в силу ограничения $m_N << \mathcal{D}$ подобная задача не может быть решена и простым расчленением последовательности на части с использованием числового значения кода элемента в качестве адреса соответствующего счетчика. Условие (6.2) означает, что в подавляющем большинстве элементы в последовательности одиночные.

Подобная задача возникает, например, при определении частоты встречаемости ℓ-грамм в буквенных текстах для больших

значений ℓ [36]. Под эффективным алгоритмом решения задачи здесь понимается алгоритм, трудоемкость которого является квазилинейной функцией длины последовательности N при достаточно широком диапазоне изменения значений этого параметра.

В [35] для решения данной задачи применяется автоматное поле. При этом в оперативной памяти выделяется $B = S/K$ перенумерованных от 0 до $B-1$ участков размером K бит. В каждом из участков запоминаются состояния соответствующего автомата. Функции этих автоматов состоят в выявлении ситуаций, возникающих при поступлении на их входы последовательности кодов $z^{(1)}, z^{(2)}, \ldots$, ассоциативно связанных со значениями элементов анализируемой последовательности. Простейший случай перехода автоматов для $K = 2$ иллюстрируется табл. 2,3 и рис.3.

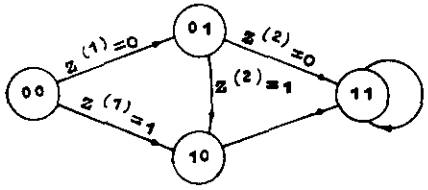


Рис. 3

Таблица 2

Таблица переходов автомата $\mathcal{A}$ - поля

Состояние автомата	Значение входного кода	
	$z = 0$	$z = 1$
00	01	10
01	11	10
10	11	11
11	11	11

Таблица 3

Таблица интерпретации состояния автоматов $\mathcal{A}$ -поля

Состояние автомата	Ситуация
00	Не было обращения
01	Одно обращение
10	Одно обращение или два различных обращения ($z^{(1)} \neq z^{(2)}$)
11	Все остальные случаи

В силу условия (6.1) задача решается путем последовательных итераций. На каждой итерации обрабатывается лишь часть элементов последовательности. При этом для каждого элемента выполняются следующие процедуры:

- устанавливается, принадлежит ли элемент к рассматриваемой сейчас части, и когда это так, то

- находится соответствующий этому элементу номер автомата и
- определяется код z, который подается на вход автомата.

Для всех трех процедур оказалось удобным применить ассоциативное кодирование. Поясним это.

1. В основе итеративного процесса лежит разбиение множества элементов, составляющих последовательность, на попарно непересекающиеся, близкие по количеству элементов подмножества. Желательно, чтобы процедура разбиения не требовала предварительного просмотра последовательности для определения границ между подмножествами. Этим требованиям как раз удовлетворяет процедура ассоциативного кодирования, дающая псевдослучайные коды, распределение которых близко к равновероятному, и гарантирующая попадание одинаковых элементов в одно и то же подмножество.

2. Для нахождения номера автомата достаточно преобразовать код элемента X в ассоциативный код y, заданный на отрезке $[0, B-1]$. Для разбиения последовательности на части, а также для нахождения номера автомата может быть использована одна и та же процедура ассоциативного кодирования, скажем метод деления.

Важно отметить, что в отличие от традиционной процедуры ассоциативного кодирования в рассматриваемом случае можно не расходовать память на запоминание кодов элементов с целью их последующего однозначного отождествления. Итеративность процедуры позволяет допускать некоторую неоднозначность в решении на каждом шаге, поскольку одиночные элементы, попавшие в зону неоднозначности на данной итерации, могут однозначно классифицироваться при последующих итерациях.

3. Использование дополнительного кода z позволяет в некоторых случаях выявить тот факт, что среди двух (или более) элементов с совпадающими кодами y нет одинаковых. Это дает возможность повысить число однозначно классифицируемых элементов на каждой итерации.

При формировании кода z важно, чтобы у наибольшего числа наложившихся элементов были бы различные значения z. Очевидно, это имеет место, когда нет зависимости между y и z и когда значения z распределены равномерно на отрезке его определения. Нетрудно видеть, что этим условиям и удовлетворяет процедура ас-

социативного кодирования. Более подробное изложение описанного выше подхода и оценки трудоемкости алгоритма содержатся в работах [35,36].

Литература

I. DUMEY A.J. Indexing for Rapid Random Access Memory Systems. - "Comput. and Automat.",1956,vol.5, N 12,p.6-9.

2. КОРОЛЕВ Л.Н. Кодирование и свертывание кодов. -"Доклады АН СССР, I957, т. II3, №4, с. 746-747.

3. ЕРШОВ А.П. О программировании арифметических операторов. -"Доклады АН СССР",I958, т.II8, № 3, с. 427-430.

4. ЕРШОВ А.П. Организация АЛЬФА-транслятора.-В кн.:АЛЬФА-система автоматизации программирования, Новосибирск, "Наука"СО, I967, с. 35-7I.

5. ЛАВРОВ С.С., ГОНЧАРОВА Л.И. Автоматическая обработка данных. Хранение информации в памяти ЭВМ. М., "Наука", I97I.

6. ULLMAN I.D. A note on the Efficiency of Hashing Functions. - "J.of the Association for Computing Machinery", 1972, vol.19, N 3,July, p.569-575.

7. ЯО ЧЖАО-ВЭЙ. Применение метода интерполяции для нахождения слов в словаре при машинном переводе.-В сб.: Проблемы кибернетики. Вып.9, I963, с. 307-3I3.

8. БАБКИН В.Ф. Простой метод универсального кодирования источника независимых сообщений. -В кн.: Доклады IУ симпозиума по избыточности в информационных системах. Л., I970.

9.БЕРНШТЕЙН С.Н. Теория вероятностей. М., "Гостехиздат", I946.

I0. СЕВАСТЬЯНОВ Б.А., ЧИСТЯКОВ В.А. Асимптотическая нормальность в классической задаче о дробинках. -"Теория вероятностей и её применения", I964, т.IX, вып. 2. с. 223-237.

II. ФЕЛЛЕР В. Введение в теорию вероятностей и её приложения, Т.I, М., Изд-во "Мир", I967.

I2. LUM V.Y., YUEN P.S.T., DODD M.M. Key-to-address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files. - "Comm.of the ACM",1971,vol.14,N 4.

I3. MAURER W.D. An improved hash code for scatter storage. - "Comm.ACM",1968, vol.11, N 1,p.35-38.

I4. BUCHHOLZ W. File organization and addressing. - "IBM System J.",1963, June, N 2,p.86-111.

I5. LIN A.D. Key addressing of random access memories by radix transformation. - Proc.AFIPS 1963 sice, vol.24, Spartan Books, New York, p.355-366.

I6. HANAN M., PALERMO F.P. An application of coding theory to a file addressing problem. - "IBM J.R.& D",1963,vol.7, N 2, April, p.127-129.

I7. SCHAY G., RAVER N. A method for key-to-address transformation.-"IBM J.R.& D",1963,vol.7, N 2,April, p.121-129.

I8. ПИТЕРСОН У. Коды,исправляющие ошибки. М.,Изд-во "Мир", I964.

I9. БЕРЛЕКЭМП. Алгебраическая теория кодирования. М., Изд-во "Мир", I97I.

20. PETERSON W.W. Addressing for random-access storage. - "IBM J.R.& D",1957,vol.1, N 2,April, p.130-146.

2I. SCHAY G., SPRUTH W.G. Analysis of a file addressing method."- "Comm.ACM",1962,vol.5, N 8,Aug.,p.459-462.

22. MCILROY M.D. A variant method of file searching.-"Comm. ACM",1963,vol.6, N 3,March,p.101.

23. MORRIS R. Scatter storage techniques. - "Comm. ACM", 1968,vol.11, N 1,p.38-43.

24. BELL J.R. The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering. - "Comm.ACM", 1970, vol.13, N 2,Feb.,p.107-109.

25. WILLIAMS F.A. Handling Identifiers as Internal Symbols in Language Processors.-"Comm.ACM",1959,vol.2,N6, June,p.21-24.

26. JOHNSON L.R. Indirect chaining method for addressing on secondary keys.-"Comm.ACM",1961,vol.4,N 5,May,p.218-222.

27. BAYS C. Some techniques for structuring chained hash tables. - "The computer journal",1973,vol.16, N 2,May,p.126-131.

28. ХОПУД. Методы компиляции. М., Изд-во "Мир", I972.

29. ЛАВРОВ С.С., СИЛАГАДЗЕ Г.С. Входной язык и интерпретатор системы программирования на базе языка ЛИСП для машины БЭСМ-6. М., I969, (ВЦ АН СССР).

30. ЗАГОРУЙКО Н.Г. Методы распознавания и их применения.М., Изд-во "Сов.радио", I972.

3I. МАЛКИНА Р.М., ПЕРВОЗВАНСКИЙ А.А. Построение последовательного решающего правила в задаче распознавания образов. -"Изв. АН СССР. Техническая кибернетика", I969, № I.

32. ТУРБОВИЧ И.Т. и др. Опознание образов. М., Изд-во "Наука", I968.

33. ВЕЛИЧКО В.М. Применение функций расстановки для обработки речи на ЭВМ. -Труды УП Всесоюзной школы-семинара "Автоматическое распознавания слуховых образов". Алма-Ата, I973.

34. ВЕЛИЧКО В.М. Обработка речи на ЭВМ методом ассоциативного кодирования. -Настоящий сборник, с.38-48.

35. КОСАРЕВ Ю.Г. Автоматные поля. -Настоящий сборник, с. 90-96.

36. ГУСЕВ В.Д., КОСАРЕВ Ю.Г., ТИТКОВА Т.Н.Отыскание статистических закономерностей текстов методом ассоциативного кодирования. -Настоящий сборник, с. 72-89.

Поступила в ред.-изд.отд.
24 декабря I973 года