УДК 004.7

Способ представления множеств последовательностей

Г.В. Дорохина, В.Н. Павлыш
Государственное учреждение «Институт проблем искусственного интеллекта»,
Донецкий национальный технический университет
sgv_iai@mail.ru, pavlyshvn@mail.ru

Дорохина Г.В., Павлыш В.Н. Способ представления множеств последовательностей. *Предложен способ представления множеств последовательностей однотипных элементов путём их обозначения, что позволяет оперировать последовательностями как строковыми величинами. Разработан метод представления словаря строк для хранения и поиска, позволяющий выполнять анализ общих начальных частей строк словаря единожды, а также выполнять операцию «активация» для части строк словаря. Показано применение разработанного метода представления словаря строк для хранения и поиска в компьютерных программах.*

Введение

Данные во многих информационно-вычислительных системах представимы в виде последовательностей – последовательности состояний, событий, команд, анализируемых условий, образов, классов, знаков.

Вопрос представления последовательности образов, условий, сложных объектов или сущностей покажется гораздо более простым, если ввести их обозначения с помощью уникальных идентификаторов.

Последовательности – это удобный способ описания явлений, ситуаций и предметных областей. В интеллектуальных системах предметные области принято представлять в виде древовидных структур (например, дерево классификации, дерево решений, дерево фактов). В данной работе древовидная структура рассматривается как способ представления последовательностей, позволяющий единожды анализировать их общие начальные части. Отметим, что большие древовидные структуры не всегда обозримы для человека. Поэтому для целей проверки отдельных фрагментов описаний предметных областей человеком наглядным будет представление некого поддерева с помощью множества последовательностей. В связи с этим актуальной является разработка такого унифицированного способа представления последовательностей однотипных элементов, который будет пригоден для их скоростной обработки и компактного хранения.

За основу возьмём деревья цифрового поиска, а именно их разновидность – лучевую память Рене де ла Брианде [1, 2]. Данную древовидную структуру используют для скоростного поиска строковых величин. Она организована следующим образом: узлы уровня 1 иерархической структуры соответствуют значениям первых символов строковых величин, элементы иерархии второго уровня соответствуют значениям вторых символов строковых величин и т.д. Так происходит ветвление «дерева», которое продолжается, представляя каждую строку, символ за символом, до достижения конца строки, о чем сигнализирует признак окончания строки, представленный специальным дочерним узлом. Поиск в этой иерархической структуре осуществляется путем нахождения элемента иерархии уровня 1, соответствующего первому символу строковой величины, затем дочернего узла этого элемента, соответствующего второму символу, и т.д.

Недостатком данного метода является необходимость хранения самих данных (множества строк) в специально выделенной области, что дополнительно приводит к значительным затратам памяти.

Цель работы – разработка и обоснование способа представления последовательностей однотипных элементов, пригодного для их скоростной обработки и компактного хранения.

Для достижения цели решены следующие задачи. Введены обозначения однотипных элементов и их последовательностей, что позволило представление последовательностей свести к задаче представления строковых величин. Формализованы представления для хранения строковых величин и для поиска строковых величин, соответствующие [1, 2]. Предложено представление для хранения и поиска строковых величин, а также его модификация, обеспечивающая удобный способ хранения строк с помощью массивов. Показаны примеры его применения, в т. ч., при распознавании последовательности образов. Введён механизм активации подмножества строк словаря, позволяющий задавать ветви, участвующие в поиске, анализе, распознавании.

Обозначение однотипных элементов и их последовательностей

Пусть имеется ограниченное количество элементов одной природы (образов, классов, условий и т.д.).

Назовём упорядоченное множество этих элементов *алфавитом элементов*. Введя для каждого элемента его обозначение с помощью уникального идентификатора (символ или число), получим упорядоченное множество (или алфавит) идентификаторов.

Обозначим последовательность элементов с помощью строки, а множество последовательностей с помощью словаря строк (табл. 1). Это позволит применять ко множествам последовательностей элементов (сложных объектов), как операции, являющиеся обычными для словарей строк (сравнение, хранение, поиск, поиск вхождений последовательностей), так и специфические (например, активация некоторых подмножеств последовательностей).

Таблица 1 – Обозначение элементов и последовательностей

Объект	Обозначение
Элемент последовательности	Идентификатор – уникальный символ (или число)
Алфавит элементов	Упорядоченное множество идентификаторов
Последовательность элементов	Строка или последовательность чисел
Множество последовательностей	Словарь строк (или множество последовательностей чисел)

Формализация понятия строки

Определение 1. Алфавитом A будем называть упорядоченное множество символов a_i, внутри которого символы с меньшим номером предшествуют символам с большим номером, т.е. справедливо условие:

$$A = \{a_i\}, i \in [1,t], \quad (1)$$

$$\forall i, j \in [1,t] \;\; i < j \Rightarrow a_i \prec a_i .$$

Здесь t – размер алфавита.

Определение 2. Строкой u длины L назовем вектор, состоящий из L символов алфавита A:

$$u = (c_1, \dots c_k, \dots c_L), c_k \in A . \quad (2)$$

Длину строки u обозначим через $|u|$.

Утверждение 1. Любой символ c алфавита A будем рассматривать как строку длины один, т.е. $u = (c)$.

Определение 2. *Пустой строкой* назовем строку нулевой длины.

Обозначим строку нулевой длины λ.

Универсальное множество строк U можно представить в виде:

$$U = \{u_j\}, j \in N . \quad (3)$$

Здесь N – множество натуральных чисел.

Введем отношения и функции, определенные на универсальном множестве строк U, а также операции, которые можно выполнять над строками.

Рассмотрим строки $u_1 = (c_1, \dots c_k, \dots c_n)$ длины n и $u_2 = (d_1, \dots d_k, \dots d_m)$ длины m, где $c_k, d_k \in A$, а также целое число $i \in [0; l]$, где $l = \min(n, m)$.

Определение 3. Две строки u_1 и u_2 назовем *эквивалентными по i первым компонентам*, если каждый из символов c_k строки u_1 равен символу d_k строки u_2 при целых $k \in [1; i]$.

Зададим это отношение с помощью функции $\tau(i, u_1, u_2)$, принимающей значение 1, когда u_1 и u_2 эквивалентны по i первым компонентам и ноль в противном случае:

$$\tau(i, u_1, u_2) = \begin{cases} 1 \mid \forall k \in [1, i], c_k = d_k \\ 0 \mid \textit{иначе} \end{cases} . \quad (4)$$

Определение 4. Назовем строку $q = (b_1, \dots b_k, \dots b_i)$ длины $i = |q|$ *маркером эквивалентности по i первым компонентам* строк $u_1 = (c_1, \dots c_k, \dots c_n)$ и $u_2 = (d_1, \dots d_k, \dots d_m)$, таких что справедливо $\tau(i, u_1, u_2) = 1$, если для любого $k \in [1, i]$ справедливо $b_k = d_k, b_k = c_k$.

Отметим, что две любые строки являются эквивалентными по 0 первым компонентам, а маркером эквивалентности по 0 первым компонентам выступает пустая строка λ.

$$\forall u_1, u_2 \in U \;\; \tau(0, u1, u2) \equiv 1 .$$

Для сравнения строк понадобится отношение равенства.

Определение 5. Две строки $u_1 = (c_1, \dots c_k, \dots c_n)$ и $u_2 = (d_1, \dots d_k, \dots d_m)$ назовем *равными* и будем записывать $u_1 = u_2$, если длины этих строк n и m равны и строки u_1, u_2 эквивалентны по n первым компонентам.

С целью упорядочения множества строк по алфавиту введем для строк u_1, u_2 отношение “меньше”.

Определение 6. Будем говорить, что строка $u_1 = (c_1, \dots c_k, \dots c_n)$ длины n *меньше* строки $u_2 = (d_1, \dots d_k, \dots d_m)$ длины m и обозначим $u_1 \prec u_2$, если для наибольшего числа *i,* такого что строки u_1 и u_2 эквивалентны по i первым компонентам, либо i равно длине первой и меньше длины второй строки, либо i меньше длин обеих строк и (i+1)-й символ первой строки c_{i+1} имеет меньший номер в алфавите, чем (i+1)-й символ второй строки d_{i+1}.

$$\forall u_1,u_2 : |u_1| = n, |u_2| = m,\ u_1 \prec u_2 \Rightarrow \begin{cases} n < m, \tau(n,u_1,u_2) = 1 \\ \exists i \in [0,l-1] : (\tau(i,u_1,u_2) = 1) \cap (c_{i+1} \prec d_{i+1}) \end{cases}. \quad (5)$$

Здесь $l = \min(n,m)$.

Используя отношение “меньше”, упорядочим множество строк.

Определение 7. Множество строк $F = \{u_i\}$ будем называть упорядоченным по алфавиту, если любая строка с меньшим номером меньше строки с большим номером, то есть справедливо:

$$u_i \prec u_j\ \forall u_i,u_j \in F : i < j\ . \quad (6)$$

Введем для двух строк $u_1 = (c_1, \dots c_k, \dots c_n)$ длины n и $u_2 = (d_1, \dots d_k, \dots d_m)$ длины m операцию конкатенации ◊, результатом которой является строка $u_3 = (b_1, \dots b_i, \dots b_{n+m})$ длины $n+m$, у которой каждый из первых n символов равен символу строки u_1, имеющему такой же порядковый номер, а каждый из последующих m символов с порядковым номером $(n+j)$ равен j-му символу строки u_2.

$$\forall u_1 = (c_1, \dots c_k, \dots c_n), u_2 = (d_1, \dots d_k, \dots d_m)$$
$$u_3 = u_1 \lozenge u_2 \Rightarrow u_3 = (b_1, \dots b_i, \dots b_{n+m}), \quad (7)$$
$$b_i = \begin{cases} c_i \mid \ i \in [1,n] \\ d_{i-n} \mid i \in [n+1, n+m] \end{cases}.$$

При разработке правил представления словаря строк для обеспечения хранения и поиска строк нам понадобится функция инверсии строки $I(u)$, которая функция определена на множестве U и принимает значения из этого множества. Она изменяет порядок символов строки на обратный:

$$\forall u = (c_1, \dots c_k, \dots c_n)$$
$$I(u) = (c_n, \dots c_k, \dots c_1). \quad (8)$$

Формализация представления словаря строк для хранения

Определение 8. Словарем строк G назовем конечное упорядоченное по алфавиту множество строк, то есть:

$$G = \{g_k\}, \forall i > j\ g_i \prec g_j\ , \quad (9)$$

где $i,k \in [1,z]$, $z = |G|$ – размер словаря.

Определение 9. Строку g_k, принадлежащую словарю G, назовем словом словаря G.

Определение 10. Идентификатором слова g_k в словаре G будем считать порядковый номер k слова g_k в словаре G.

Чтобы определить, является ли строка u словом g_i словаря G, и определить идентификатор i этого слова в словаре G, введем функцию $N(u,G)$, принимающую значение i, если в словаре найдется слово g_i, равное строке u, и 0, иначе:

$$N(u,G) = \begin{cases} i \in [1;z] \mid \exists g_i \in G : u = g_i \\ 0 \mid \quad иначе \end{cases}. \quad (10)$$

Обратная функция $N^{-1}(i,G)$ позволяет получить слово g_i словаря по его идентификатору i в словаре G. При этом $i \in [1, z]$, а принимаемое значение $g \in G$:

$$N^{-1}(i,G) = g : i = N(g,G)\ . \quad (11)$$

Утверждение 2. Некоторое представление словаря строк применимо для хранения строк, если для него определены функция получения идентификатора строки и функция получения строки по идентификатору.

Формализация представление словаря строк для поиска

Формализуем представление словаря строк для поиска методом деревьев цифрового поиска. В этом методе сначала весь словарь, а затем его подмножества разбивают на группы. На каждом шаге таких разбиений в одну и ту же группу попадают слова, у которых 1, 2 и т.д. первых символов совпадают. Согласно введенной ранее терминологии, на каждом шаге разбиений слова, попавшие в одну группу, эквивалентны друг другу по 1, 2 и т.д. первым компонентам, соответственно.

Для формального описания этих операций введем подмножества словаря, состоящие из слов, длина которых не меньше некоторой величины i. Разобьем каждое из этих подмножеств на классы эквивалентности по i первым компонентам.

Пусть y_i – подмножество словаря G, состоящее из всех слов длины больше или равной i и пусть $Y = \{y_i\}, i \in [0,h]$, где h – максимальная длина слова словаря:

$$Y = \{y_i\}_{i=0}^{h},$$
$$y_i \subseteq G : \forall g \in G, \mid g \mid \geq i \Rightarrow g \in y_i. \quad (12)$$

Каждое из множеств y_i разобьем на классы эквивалентности G_k^i по i первым компонентам (4). Разбиение $\Re_i$ будет включать в себя классы эквивалентности G_k^i:

$$\Re_i = \{G_k^i\}_{k=1}^{r_i}, \quad (13)$$

где r_i – количество классов эквивалентности в разбиении множества y_i. Соберём во множество $\aleph$ разбиения $\Re_i$, $i \in [0,h]$, где h – максимальная длина слова в словаре G:

$$\aleph = \{\Re_i\}_{i=0}^{h}.$$

Исходя из свойств разбиений на классы эквивалентности, разбиение $\Re_i$ должно удовлетворять следующему условию:

$$\forall i \in [1,h] \forall G_k^i, G_l^i \in \Re_i$$
$$G_k^i \cap G_l^i = \varnothing, \bigcup_{k=1}^{r_i} G_k^i = y_i. \quad (14)$$

Все строки, попавшие в один класс разбиения G_k^i эквивалентны по i первым компонентам. Поставим в соответствие каждому классу G_k^i строку q_k^i длины i, которая будет маркером эквивалентности любой пары строк, принадлежащих этому классу. Назовём строку q_k^i маркером эквивалентности класса G_k^i. Соответствие между классами G_k^i разбиения $\Re_i$ и маркерами эквивалентности q_k^i этих классов является взаимно-однозначным.

Введем отображение $\alpha(G_k^i)$ классов разбиения $\Re_i$ на универсальное множество строк U, по позволяющее определить маркер эквивалентности q_k^i класса G_k^i:

$$\forall i \in [0,h] \forall G_k^i \in \Re_i \forall g \in G_k^i$$
$$\alpha(G_k^i) = q \in U : | q | = i, \tau(i,q,g) = 1,$$

где h – максимальная длина слова словаря G.

Объединим строки $q_k^i = \alpha(G_k^i)$ во множество $S_i = \{q_k^i\}_{k=1}^{r_i}$, где $r_i = |\Re_i|$. Потребуем, чтобы множество строк S_i было упорядочено по алфавиту и номера соответствующих друг другу элементов во множествах $\Re_i$ и S_i совпадали.

Утверждение 3. Любой класс G_k^i разбиения $\Re_i$ содержит не более одного слова длины i, причем это слово равно маркеру эквивалентности $q = \alpha(G_k^i)$ класса G_k^i.

Пользуясь Утверждением 3 и отображением (11), можно определить, является ли маркер эквивалентности некоторого класса G_k^i словом словаря, и получить идентификатор этого слова.

Пусть $q = \alpha(G_k^i)$. Допустим, $q \in G_k^i$. Так как $G_k^i \subseteq G$, то $q \in G$. Тогда, согласно (11), $N(q,G)$ принимает положительное значение. Если же $q \notin G_k^i$, то $q \notin G$. А значит $N(q,G) = 0$.

Введем отображение $\beta(G_k^i, G)$, позволяющее определить, является ли маркер эквивалентности класса G_k^i словом словаря, и получить идентификатор этого слова. Отображение $\beta(G_k^i, G)$ принимает значения из множества целых [0; z], где z – количество строк в словаре G.

$$\beta(G_k^i, G) = N\left(\alpha(G_k^i), G\right) \quad (15)$$

Рассмотрим, как связаны между собой классы разбиений $\Re_i$ и $\Re_{i+1}$.

Утверждение 4. Каждый класс G_l^{i+1} разбиения $\Re_{i+1}$ является подмножеством некоторого класса G_k^i разбиения $\Re_i$, причём только одного.

Введем отображение связывающее классы разбиений $\Re_i$ и $\Re_{i+1}$. Рассмотрим классы G_k^i и класс G_l^{i+1}, для которых $G_l^{i+1} \subseteq G_k^i$. Их маркерами эквивалентности являются q_l^{i+1} и q_k^i, соответственно. Первые i символов строк q_k^i и q_l^{i+1} совпадают, и строка q_l^{i+1} длиннее строки q_k^i на один символ, который можем обозначить через c, откуда:

$$q_l^{i+1} = q_k^i \Diamond (c).$$

Преобразуем это выражение к виду:

$$\alpha(G_l^{i+1}) = \alpha(G_k^i) \Diamond (c). \quad (16)$$

То есть, каждому классу G_l^{i+1} можно поставить в соответствие пару (G_k^i, c), причём только одну. Аналогично, любой паре (G_k^i, c), где $G_k^i \in \Re_i$ и $c \in A$ соответствует не более одного класса разбиения $\Re_{i+1}$. Введём отображение $\eta(G_k^i, c)$.

$$\eta(G_k^i, c) = \begin{cases} G_l^{i+1} \mid \exists G_l^{i+1} : \alpha(G_l^{i+1}) = \alpha(G_k^i) \Diamond (c) \\ \varnothing \mid \textit{иначе} \end{cases}. \quad (17)$$

Здесь $G_l^{i+1} \subseteq G_k^i$.

Маркер эквивалентности q_l^{i+1} каждого класса $G_l^{i+1} = \eta(G_k^i, c)$ единственным образом формируется из маркера эквивалентности q_k^i класса G_k^i и символа c. Введем отображение φ, позволяющее определить для некоторого класса G_l^{i+1} этот символ c.

$$\forall c \in A \forall G_l^{i+1} \in \Re_{i+1}, \forall G_k^i \in \Re_i : G_l^{i+1} = \eta\left(G_k^i, c\right) \quad (18)$$
$$\varphi(G_l^{i+1}) = c$$

Оценка количества r_i классов в разбиении $\Re_i$. необходима при оценке количества памяти, необходимой для хранения представлений словаря.

Утверждение 5. Величина r_i зависит от содержимого словаря G и размера алфавита A и удовлетворяет неравенству:

$$\max\left(z_i, \left\lceil \frac{r_{i+1}}{t} \right\rceil\right) \le r_i \le \min\left(|y_i|, t^i\right), \quad (20)$$

где i-номер разбиения, $r_i = |\Re_i|$, $r_{i+1} = |\Re_{i+1}|$, t –

размер алфавита A, z – количество слов словаря G, z_i – количество слов словаря длины i, $\rceil\lfloor$ – операция округления вверх до целого числа. Неравенство (20) легло в основу работы [3].

Утверждение 6. Любые два различных класса G_l^{i+1} и G_m^{i+1} разбиения $\Re_{i+1}$ являющиеся подмножествами некоторого класса G_k^i разбиения $\Re_i$ не имеют общих элементов:

$$\begin{matrix}\forall G_k^i : \exists G_l^{i+1}, G_m^{i+1} \subseteq G_k^i, G_l^{i+1} \neq G_m^{i+1} \\ G_l^{i+1} \cap G_m^{i+1} = \varnothing\end{matrix}. \quad (21)$$

Данное утверждение следует из свойств разбиения множества на классы эквивалентности. Из утверждения 6 следует, что классы G_k^i разбиения $\Re_i$ связаны с классами и разбиения $\Re_{i+1}$ в отношении один ко многим. То есть взаимосвязь между классами разбиений можно описать с помощью ориентированного графа, который является деревом. Вершинами этого графа являются классы разбиений. Дуги графа соединяют классы $G_k^i \in \Re_i$ и $G_l^{i+1} \in \Re_{i+1}$ и направлены от класса разбиения с меньшим номером $\Re_i$ к классу разбиения с большим номером $\Re_{i+1}$.

Обозначим через W множество вершин w_j, каждая из которых является классом G_k^i одного из разбиений $\Re_i$. Множество W мощности v можно получить путем объединения всех разбиений $\Re_i$, где $i \in [0;h]$.

$$W = \bigcup_{i=0}^{h} \Re_i \ , \ |W| = \sum_{i=0}^{h} \Re_i = v \ ,$$
$$W = \{w_j\}, \forall j \in [1,v] \ \exists i \in [0,h] : w_j \in \Re_i \ . \quad (22)$$

Пусть классы во множестве W пронумерованы в соответствии с порядком по алфавиту маркеров эквивалентности этих классов. Тогда $w_1 = G$ – единственный класс разбиения $\Re_0$, т.к. $\alpha(w_1)$ – пустая строка находится в отношении меньше с любой непустой строкой.

Исходя из (22), каждая функция, определенная на множестве классов разбиения $\Re_i$, где $i \in [0,h]$, определена и на множестве W, а отображения принимающие значения из множества классов разбиения $\Re_i$ также принадлежат множеству W.

Перепишем отображения (14), (15), (17), (18) с учетом (22).

Отображение α, ставящее в соответствие каждому классу эквивалентности маркер эквивалентности, (14) примет вид:

$$\begin{matrix}\forall w_j \in \Re_i \forall g \in w_j \\ \exists q \in U, q = \alpha(w_j) : |q| = i, \tau(i,q,g) = 1\end{matrix}, \quad (23)$$

где $j \in [1; v]$, $i \in [0; h]$, q – маркер эквивалентности класса w_j.

Отображение $\beta(w_j, G)$ (15) позволяет определить, принадлежит ли словарю G строка g, равная маркеру эквивалентности класса w_j, и каков ее идентификатор в словаре. С учетом (22) оно примет вид:

$$\beta(w_j, G) = N(\alpha(w_j), G) \ . \quad (24)$$

Отображение η (17), определяющее для любого класса $w_j \in \Re_i$ и символа c, существует ли класс $w_k \in \Re_{i+1}$, чей маркер эквивалентности $\alpha(w_k)$ формируется путём конкатенации маркера эквивалентности класса $\alpha(w_j)$ и символа c, примет вид:

$$\eta(w_j, c) = \begin{cases} w_k \mid \exists w_k : \alpha(w_k) = \alpha(w_j) \Diamond (c) \\ \varnothing \mid иначе \end{cases} \quad (25)$$

Отображение φ (18), позволяющее определить для некоторого класса $w_k = \eta(w_j, c)$ символ c, примет вид:

$$\begin{matrix}\forall c \forall w_k, w_j : w_k = \eta(w_j, c) \\ \varphi(w_k) = c\end{matrix} \quad (26)$$

Таким образом, получим представление словаря строк в виде ориентированного графа, вершины которого представлены множеством W. Вершины w_j и w_k соединены ребром (w_j, w_k), если выполняется условие:

$$w_k = \eta(w_j, \varphi(w_k)) \ . \quad (27)$$

Рёбра графа можно задать с помощью матрицы смежности M размерности $v \times v$, где $v = |W|$ (22). Значения элементов m_{jk} матрицы смежности M определим по формуле:

$$m_{jk} = \begin{cases} 1 \mid w_k = \eta(w_j, \varphi(w_k)) \\ 0 \mid иначе \end{cases} . \quad (28)$$

Множество вершин W и матрица смежности M образуют представление словаря для поиска. Это представление требует больших затрат памяти для хранения множества вершин, так как каждая из вершин – множество строк. Покажем, что оно избыточно.

Каждое слово длины i принадлежит словарю, а также одному из классов разбиения $\Re_1$, одному классу разбиения $\Re_2$, ... , одному классу разбиения $\Re_{i-1}$, одному классу разбиения $\Re_i$. То есть, одно и то же слово хранится несколько раз. Значит, нужно более экономно задать множество вершин.

Подавляющее большинство элементов m_{jk} матрицы смежности M будут нулевыми. То есть, матрица М – разряженная. Следовательно, нужно найти иной метод хранения рёбер ориентированного графа.

Утверждение 7. Каждый класс w_j

однозначно определяется значениями отображений $\varphi(w_k)$, $\beta(w_j,G)$, а также множеством номеров классов $\wp_j=\{k\}$, при которых справедливо $w_k=\eta(w_j,\varphi(w_k))$.

По Утверждению 7 каждая вершина w_j, может быть заменена парой значений $\left(\varphi(w_j),\beta(w_j,G)\right)$. Обозначим через o_{j1} значение $\varphi(w_j)$ и o_{j2} значение $\beta(w_j,G)$.

Утверждение 7 также позволяет найти более экономный метод хранения рёбер ориентированного графа в сравнении с матрицей смежности *M*. Множество $\wp_j=\{k\}$ есть не что иное, как множество всех номеров *k*, при которых для значений справедливо $m_{jk}=1$. То есть, $\wp_j=\{k\}$ описывает множество рёбер, исходящих из вершины w_j. Обозначим через o_{j3} множество $\wp_j$. Для ускорения поиска элемента во множестве o_{j3} пронумеруем его элементы таким образом, чтобы элемент k_i с меньшим номером *i* ссылался на вершину w_{k_i}, для которой символ $o_{k_i}=\varphi(w_{k_i})$ предшествует в алфавите аналогичному символу вершины, на которую ссылается элемент k_{i+x} с большим номером (*i*+*x*):

$$\begin{array}{l} o_{k_i 1} \prec o_{k_{(i+x)}1} \\ \forall x>0\ \forall k_i,k_{i+x}\in o_{j3} \end{array}. \qquad (29)$$

Множество *O* векторов $o_j=(o_{j1},o_{j2},o_{j3})$ является более компактным заданием орграфа, определяемого множеством вершин $W=\{w_j\}_{j=1}^{v}$, и множеством рёбер, которое задано матрицей смежности $M=\{m_{jk}\}_{j,k=1}^{v}$:

$$\begin{array}{l} O=\{o_j\}_{j=1}^{v},\ o_j=(o_{j1},o_{j2},o_{j3}), \\ o_{j1}=\varphi(w_j), o_{j2}=\beta(w_j,G), \\ o_{j3}=\{k_i\}: w_{k_i}=\eta(w_j,\varphi(w_{k_i})). \end{array} \qquad (31)$$

Множество векторов *O* (31) является представлением словаря строк *G*. Оно описывает древовидную структуру, которая хранит данные о строках в ветви: вершина древовидной структуры описывается элементом o_1; вершина высоты *j* соответствует *j*-му символу строки. Вектор o_j содержит: o_{j1} – символ строки; o_{j2} – идентификатор строки, оканчивающейся в данной вершине или 0, если в данной вершине строка не оканчивается; o_{j3} – множество номеров вершин-потомков данной вершины.

Представление (31) не позволяет задать функцию получения строки *g* по её идентификатору в словаре *G*, но даёт возможность определить идентификатор строки g в словаре *G*.

Введём функцию $N(g,O)$ для определения идентификатора строки $g=(c_1,..c_l,..c_L)$ словаря *G* в представлении *O* (31).

Алгоритм определения $N(g,O)$.

1. *Инициализация:*
   l=0, b_0 = 1.
2. *Итерация:*
   Пока $(l<L)\wedge(b_l\neq 0)$ выполнять
   $$b_{l+1}=\begin{cases} k, \exists k\in o_{b_l 3}: o_{k1}=c_{l+1} \\ 0, \textit{иначе} \end{cases}.$$
3. *Результат:*
   $$N(g,O)=\begin{cases} 0 \mid & \exists l\in[1,L]: b_l=0 \\ o_{k_L 2} \mid \textit{иначе} \end{cases}.$$

В приведенном алгоритме *l* – номер шага и рассматриваемого символа строки *g*, а b_l – номер текущей вершины древовидной структуры, которая описана вектором $o_{b_l}=(o_{b_l 1},o_{b_l 2},o_{b_l 3})$. На каждом шаге итерации выполняют поиск номера вершины-потомка, которая хранит символ $o_{b_l 1}$, совпадающий со следующим символом строки c_{l+1}. Если такая вершина-потомок существует, её номер будет присвоен b_{l+1} – номеру вершины, рассматриваемому на следующем шаге. Итерация прекращается при достижении последнего символа строки c_L или в случае, если у вершины с номером b_l нет вершины-потомка, которая хранит символ, совпадающий со следующим символом строки. Если для одного из b_l, $l\in[0, L\text{-}1]$, не найдено вершины-потомка, то результатом функции будет 0, что означает отсутствие строки *g* в словаре *G*. При обнаружении вершины, чей маркер эквивалентности совпадает с искомой строкой, результатом будет $o_{b_l 2}$ – идентификатор маркера эквивалентности класса w_{b_l} в словаре *G*.

Утверждение 8. Некоторое представление словаря строк применимо для поиска строк, если для него определена функция получения идентификатора строки.

Разработка представления словаря строк для хранения и поиска

Модифицируем представление словаря для поиска с тем, чтобы обеспечить также хранение строк.

Утверждение 9. Для любого класса $w_k\in W$ с номером $k\in[2,v]$ существует класс $w_j\in W$ с номером $j\in[1,v]$, для которого справедливо $w_k=\eta(w_j,\varphi(w_k))$, причём только один.

$$\begin{array}{l} \forall k\in[2,v]: w_k\in W \\ \exists!\, w_j\in W, j\in[1,v]: w_k=\eta(w_j,\varphi(w_k)) \end{array} \qquad (32)$$

Пользуясь Утверждением 9, введем функцию $\chi(w_k)$, определенную на множестве

$W' = W \setminus w_1$ и принимающую целочисленные значения j из диапазона [1; v], где $v = |W|$.

$$\chi(w_k) = j : w_k = \eta(w_j, \varphi(w_k)) \qquad (33)$$

Заменим в представлении (31), множество вершин O множеством вершин O', элементы o'_j которого формируются из элементов o_j множества O путём введения в него дополнительного элемента $o_{j4} = \chi(w_j)$, являющегося номером родительской вершины по отношению к вершине w_j:

$$\begin{aligned} &O' = \{o'_j\}_{j=1}^{v},\ o'_j = (o_{j1}, o_{j2}, o_{j3}, o_{j34}), \\ &o_{j1} = \varphi(w_j), o_{j2} = \beta(w_j, G), \\ &o_{j3} = \{k_i\} : w_{k_i} = \eta(w_j, \varphi(w_{k_i})), \\ &o_{j4} = \chi(w_j). \end{aligned} \qquad (34)$$

Поскольку представление O' полностью включает в себя данные представления O, то оно обеспечивает поиск строк аналогично описанному в алгоритме определения $N(g, O)$.

Пользуясь представлением (34) можем для каждого элемента o'_j множества O' получить маркер эквивалентности соответствующего ему класса эквивалентности w_j.

Алгоритм получения маркера эквивалентности q класса w_j по представлению O' словаря G.

1. *Инициализация:*
   $r = j$, $u = \lambda$.
2. *Итерация:*
   Пока $(r \neq 1)$ выполнять
   $u = u \lozenge o_{r1}$,
   $r = o_{r4}$.
3. *Результат:*
   $q = I(u)$.

Здесь переменная r принимает целочисленные значения и обозначает номер рассматриваемого элемента множества O'; u – строка (изначально – пустая). В строку u, используя операцию конкатенации (7), добавляют символ o_{r1}, который хранится в рассматриваемом элементе. Эта операция выполняется итеративно по мене продвижения вверх по ветви древовидной структуры (напомним, o_{r4} – номер вершины, являющейся родительской по отношен) до тех пор пока не достигнем корневой вершины дерева w_1.

Запомним в некотором множестве $X = \{x_i\}_{i=1}^{z}$, где $z = |G|$, номера j всех элементов o'_j, у которых $o'_{j2} \neq 0$. Вспомним, что o'_{j2} хранит идентификатор в словаре G строки q, являющейся маркером эквивалентности вершины w_j, если она принадлежит словарю G и ноль в противном случае. Таким образом, множество X будет хранить номера всех вершин, чьи маркеры эквивалентности являются словами словаря G.

Потребуем, чтобы элементы каждый номер i элемента x_i=j множества X совпадал с идентификатором в словаре G слова, совпадающего с маркером эквивалентности класса w_j:

$$\begin{aligned} &\forall o'_j \in O' : o_{j2} \neq 0 \\ &\exists x_i \in X : (x_i = j) \wedge (o'_{j2} = i) \end{aligned}. \qquad (35)$$

Пара множеств

$$(O', X) \qquad (36)$$

образует представление словаря G, которое можно использовать для хранения. Оно позволяет получать слово g по его идентификатору $B \in [1, z]$ без использования словаря G. Искомую строку получаем как маркер эквивалентности класса w_j, где $j = x_B$ по выше приведенному алгоритму.

Таким образом, разработанное представление (36) словаря G является представлением для хранения и поиска строк.

Применение представления словаря строк для хранения и поиска в компьютерных программах

Разработанное представление словаря строк для хранения и поиска апробировано в качестве средства для хранения и скоростного поиска строк в программных библиотеках «Модуль декларативного морфологического анализа слов русского языка» [4] и «Модуль бессловарного морфологического анализа слов русского языка» [5], а также как средство для хранения и анализа последовательностей образов в задаче пофонемного распознавания [6] слов русского языка. В последнем случае последовательность образов – последовательность аллофонов (разновидностей звучания фонем), образующих слово словаря. Для решения задачи пофонемного распознавания слов параллельно с алфавитом образов (аллофонов) введён алфавит их идентификаторов (транскрипционных символов), а множество слов словаря обозначено через множество транскрипций, которое представлено согласно (38). Распознавание выполняется в процессе обхода древовидной структуры O', при котором в каждой вершине происходит сопоставление фрагмента речевого сигнала с образом аллофона.

В упомянутых программных реализациях представление (38) физически организовано с помощью нескольких массивов, хранящих структуры фиксированной длины:

- массив потомков;
- массив вершин;
- массив номеров вершин, в которых оканчиваются строки.

То есть, для удобства программной реализации представление (38) преобразовано так, что его удобно хранить с помощью массивов.

Допустим, O' содержит ω элементов

$o'_j = (o_{j1}, o_{j2}, o_{j3}, o_{j4})$ для которых $o_{j3} \neq \varnothing$. Соберём все элементы $o_{j3} \neq \varnothing$ во множестве *P*:

$$P = \{p_i\}_{i=1}^{\omega}$$
$$\forall i \in [1, \omega] \exists! j : o'_j = (o_{j1}, o_{j2}, o_{j3}, o_{j4}) \in O', o_{j3} \neq \varnothing.$$
$$\forall o'_j = (o_{j1}, o_{j2}, o_{j3}, o_{j4}) \in O' : o_{j3} \neq \varnothing \quad o_{j3} \in P$$

Множество *P* содержит по одному вхождению непустых множеств o_{j3} и все непустые множества o_{j3} обязательно принадлежат *P*.

На основе вектора $o'_j = (o_{j1}, o_{j2}, o_{j3}, o_{j4})$ сформируем вектор o''_j путём замены вектора o_{j3} на число o'_{j3}, принимающее значение, равное номеру i элемента p_i множества *P*, значение которого совпадает со множеством o_{j3}, или 0, если $o_{j3} = \varnothing$.

$$O'' = \{o''_j\}_{j=1}^{v}, o''_j = (o_{j1}, o_{j2}, o'_{j3}, o_{j4}),$$
$$o'_{j3} = \begin{cases} i \mid \exists p_i \in P : p_i = o'_{j3} \\ 0 \mid иначе \end{cases}$$

В результате представление для хранения и поиска (*O', X*) будет преобразовано к виду:

(O'', P, X).

Данное представление также позволяет выполнять хранение и поиск. Его преимуществом по сравнению с представлением (38) является то, что вектора o''_j могут быть описаны структурой данных фиксированного размера, и, следовательно, множество O'' представимо в виде массива таких структур. Это делает удобным работу с ними даже при хранении O'' в файле (прямого доступа). Упорядоченное множество O'' – основной элемент, описывающий вершины древовидной структуры, являющейся представлением множества последовательностей. Номер j вершины однозначно определяет вершину o''_j древовидной структуры, ветвь из корня o''_1 до вершины o''_j, характеризующую общую начальную часть множества последовательностей, принадлежащих классу разбиения w_j. Это использовано в методе пофонемного распознавания слов большого словаря, где распознавание выполняется в процессе обхода древовидной структуры, заданной парой множеств (O'', P). Каждому элементу o''_j множества O'' поставлены в соответствие b_j и d_j. При работе с вершиной o''_j значения b_j известно к моменту начала обработки образа аллофона, связанного с вершиной o''_j, и обозначает границу начала этого аллофона в распознаваемом сигнале, а d_j определяют по результату сопоставления образа аллофона и фрагмента речевого сигнала.

Связав с каждой вершиной o''_j представления (O'', P, X) «флаг» активации ac_i, принимающий ненулевое значение для активированных вершин и 0 для остальных. Объединим элементы ac_i во множестве *AC*.

Алгоритм активации вершин, строки с идентификатором *Id*.

1. *Инициализация:*
   $r = x_{Id}$.
2. *Итерация:*
   Пока $(r \neq 1) \wedge (ac_r = 0)$ выполнять
   $ac_r = 1$,
   $r = o_{r4}$.

Здесь *r* – номер текущей вершины; $x_{Id} \in X$ (35) – номер вершины, в которой оканчивается строка с идентификатором *Id*.

Данный алгоритм выполняет проход вверх по ветви, представляющей ФТ, и выставляет флаг активации вершин ac_r до тех пор, пока не будет достигнута корневая вершина или уже активированная вершина. Второе из условий сокращает число операций, когда необходимо активировать более одной строки.

Операция, обратная активации, выполняется путём обнуления всех элементов множества *AC*. Если *AC* представлено массивом, то она выполняется путём заполнения области памяти нулевым значением, что не требует значительных временных затрат. Так же быстро могут быть выполнены операции инициализации элементов и структур данных, которые в рамках решаемой задачи связываются по номеру j с вершиной древовидной структуры o''_j.

Выводы

В работе предложено использовать для представления последовательностей сложных объектов их обозначения, что позволит хранить и обрабатывать множества таких последовательностей как словари строк.

Формализовано представления словаря строк для хранения строк. Показано, что некоторое представление применимо для хранения строк, если для него определены функция получения идентификатора строки и функция получения строки по идентификатору.

Формализовано представление словаря строк для поиска. Оно организовано с помощью древовидной структуры, хранящей строки вдоль ветви от корня к листу, и позволяет определять принадлежность строки словарю и её идентификатор.

На основе представления словаря строк для поиска, разработано представление словаря строк для хранения и поиска, которое в ряде случаев позволяет сократить затраты памяти. Предложена модификация разработанного представления,

обеспечивающая удобный способ хранения строк с помощью массивов. Указаны компьютерные программы, в которых оно использовано как представление словарей строк. Показано, что данное представление можно использовать при распознавании последовательности образов. Введён механизм активации подмножества строк словаря, позволяющий задавать ветви, участвующие в поиске, анализе, распознавании.

Предложенное в работе представление словаря строк для хранения и поиска, а также введённое обозначение множеств последовательностей может быть полезно при решении практических задач и проведении исследований в области искусственного интеллекта.

Литература

11. R. de la Briandais. File searching using variable-length keys // Proc. Western Joint Computer Conf., San Francisco, March 1959. – pp. 295–298.
12. Кнут, Д. Э. Искусство программирования, том 3. Сортировка и поиск, 2-изд.: Пер. с англ.: Уч. Пос. – М., 2000, С. 529–530.
13. Дорохина Г.В. Сравнение затрат памяти для метода деревьев цифрового поиска и его усовершенствования // Искусственный интеллект 2009. – № 4. – с. 338 – 343.
14. Свідотство про реєстрацію авторського права на твір Комп'ютерна програма «Модуль декларативного морфологічного аналізу слів російської мови» («РДМА_ІПШІ») №43188 від 09.04.2012. Автор: Г.В. Дорохіна, Власник: Інститут проблем штучного інтелекту.
15. Дорохина Г.В., Трунов В.Ю., Шилова Е.В. Модуль морфологического анализа без словаря слов русского языка // Искусственный интеллект. – 2010. – №2. – С.32-36.
16. Дорохина Г.В. Модификация алгоритма DTW для пофонемного распознавания слов // Проблемы искусственного интеллекта. – 2015. - № 0 (1). – С. 38 – 49.

Ключевые слова: *способы представления данных, хранение, скоростной поиск, усовершенствованное дерево цифрового поиска, последовательности, временные ряды.*

Дорохіна Г.В., Павлиш В.М. Спосіб представлення множин послідовностей. *Дані в багатьох інформаційно-обчислювальних системах подаютьюся у вигляді послідовностей. Запропоновано спосіб подання множин послідовностей однотипних елементів шляхом їх позначення, що дозволяє оперувати послідовностями як строковими величинами. Формально описано традиційний метод представлення словника рядків у вигляді впорядкованої множини (представлення словника рядків для зберігання) і метод представлення словника рядків у вигляді дерева цифрового пошуку (представлення словника рядків для пошуку). На їх основі розроблено метод представлення словника рядків для зберігання і пошуку. Він дозволяє виконувати аналіз загальних початкових частин рядків словника раз, а також виконувати операцію «активація» для частини рядків словника. Крім того, розроблений метод є альтернативою одночасному використанню двох формально описаних представлень, що для певних наборів даних дозволяє зменшити витрати пам'яті. Показано застосування розробленого методу представлення словника рядків для зберігання і пошуку в комп'ютерних програмах.*

Ключові слова: *способи представлення даних, зберігання, швидкісний пошук, удосконалене дерево цифрового пошуку, послідовності, часові ряди.*

Dorokhina G.V. Pavlysh V.N. A method of presenting sets of sequences. *The data in many data-processing systems can be represented as a series. We propose the way of presentation of sets of similar elements of the sequences by their designation. It allows the management of strings as string variables. The traditional method for representation of a dictionary of strings as an ordered set (the representation of the dictionary of strings for storing) and the method of representation of the dictionary of strings as a digital search tree (representation of the dictionary of strings for searching). On this basis, we developed a method of presenting a dictionary of strings for storing and searching. It enables to analyze the common initial parts of strings of dictionary once. It also performs the operation "activation" of strings of dictionary. In addition, the developed method is an alternative to the simultaneous using of two formally described representations, that allows to reduce memory costs. The application of the developed method of presentation of a dictionary of a strings for storing and searching in computer programs.*

Keywords: *data representation, storing of data, high-speed search, advanced digital search tree, sequences, time series.*

Статья поступила в редакцию 21.05.2016
Рекомендована к публикации д-ром техн. наук А.С. Миненко