УДК 004.422.639

Ю.В. Шабля, А.В. Токарева

Способы хранения структур деревьев И/ИЛИ и их вариантов в оперативной и постоянной компьютерной памяти

Древовидные структуры данных активно используются для представления информационных объектов, содержащих в себе иерархические отношения между их составными частями. Примером таких древовидных структур являются деревья И/ИЛИ, приложение которых можно найти в области разработки алгоритмов комбинаторной генерации и связанных с нею задач. Исследуются возможные способы хранения древовидных структур в оперативной и постоянной памяти устройств, обрабатывающих эти структуры. Также рассматривается адаптация данных способов к задаче хранения структур деревьев И/ИЛИ и их вариантов. Кроме того, с целью организации оперативной работы с хранящимися вариантами дерева И/ИЛИ авторами предлагается биективное отображение структуры соответствующего дерева И/ИЛИ на схему реляционной базы данных.

Ключевые слова: древовидная структура, вариант дерева И/ИЛИ, оперативная память, постоянная память, структура данных, связный список, реляционная база данных.
DOI: 10.21293/1818-0442-2024-27-2-44-50

Хранение информации является одним из основных процессов любой информационной системы. Выделяются следующие две задачи, связанные с хранением данных в компьютерной памяти, в зависимости от типа используемого запоминающего устройства: хранение данных в оперативной и постоянной памяти.

Оперативная память представляет собой тип энергозависимой памяти в электронных устройствах, который предназначен для хранения обрабатываемых данных в рамках текущего сеанса информационной системы. Постоянная память играет ключевую роль в длительном хранении данных, в том числе при отсутствии электропитания.

Для разных типов структур данных существуют свои способы организации их хранения в оперативной и постоянной памяти. При этом можно оценивать эффективность способов хранения данных по таким критериям, как удобство формы представления для последующей обработки, скорость работы алгоритмов обработки, объем занимаемой памяти.

Древовидные структуры являются широко используемой структурой данных для представления информации, особенно при наличии в ней иерархических отношений. Например, деревья могут применяться при решении задач в управленческих целях, когда требуется хранение большого объема данных, способствующих оперативному принятию решения по ретроспективной информации [1, 2]. Древовидные структуры используются для анализа электрических цепей, формирования генеалогического древа или представления информации в системах управления базами данных [3], в системах поддержки принятия решений как карты возможных результатов вариантов [4], для создания каталогов в программном обеспечении [5], а также во многих других теоретических и практических задачах. Распространенность иерархических структур в реальном мире обосновывает актуальность поиска подходящих способов представления древовидных структур в оперативной и постоянной памяти.

Существует большое количество разнообразных типов древовидных структур: двоичное дерево, B-дерево, красно-черное дерево, АВЛ-дерево и многие другие [6]. Данная работа посвящена исследованию структур деревьев И/ИЛИ и их вариантов, приложение которых можно найти в области разработки алгоритмов комбинаторной генерации и связанных с нею задач [7]. Примеры прикладных задач, решение которых может быть реализовано с помощью структур деревьев И/ИЛИ: построение многовариантных тестовых заданий [8], сжатие журналов событий [9], идентификация и прослеживаемость сложных технических изделий [10]. Таким образом, применение методологии разработки алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ в прикладных задачах в конечном итоге сопровождается разработкой программного обеспечения, автоматизирующего соответствующие информационные процессы. Следовательно, возникает потребность в наличии оптимальных способов хранения структур деревьев И/ИЛИ и их вариантов как в оперативной, так и постоянной компьютерной памяти.

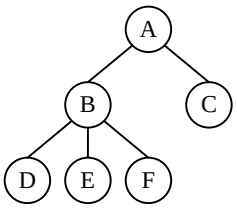
Целью данной работы является исследование возможных способов хранения древовидных структур в оперативной и постоянной памяти устройств, обрабатывающих эти структуры, а также адаптация данных способов к задаче хранения структур деревьев И/ИЛИ и их вариантов.

Способы хранения древовидных структур

Важной задачей управления данными является организация их хранения. Особенность древовидных структур данных заключается в том, что для работы с ними необходимо знать корневой узел и его связи со всеми узлами-потомками, а также аналогичную информацию о каждом узле-потомке. Существуют различные способы организации процесса хранения древовидных структур как в оперативной, так и в постоянной памяти. Далее рассмотрим возможные варианты решения данной задачи.

Хранение древовидных структур в оперативной памяти может быть организовано:

1. *На основе массивов*. В данном случае количество элементов массива определяется количеством узлов соответствующего дерева, т.е. каждому элементу массива сопоставляется конкретный узел дерева. При этом каждый элемент массива хранит указатель на другой элемент массива, соответствующий родительскому узлу рассматриваемого узла дерева. Указанный способ позволяет хранить только структуру дерева (связи между узлами). Если требуется хранить дополнительную информацию о каждом узле (например, метка или ключ), то необходимо создать второй массив, элементы которого будут хранить такую информацию о соответствующих узлах дерева. На рис. 1 приведены пример структуры дерева и соответствующее ему представление в форме массивов.



Индексы массива:	0	1	2	3	4	5
Элементы массива № 1:	0	0	0	1	1	1
Элементы массива № 2:	A	B	C	D	E	F

Рис. 1. Пример структуры дерева
и соответствующее ему представление в форме массивов

Использование массивов для хранения древовидных структур эффективно с точки зрения требуемого объема памяти. Однако такой способ хранения неэффективен с точки зрения удобства формы представления для последующей обработки (обнаружение связи между узлами требует многократного обхода элементов массива);

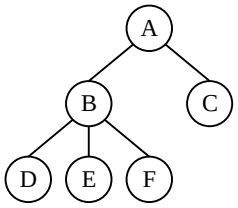
2. *На основе связных списков*. В данном случае каждому узлу дерева сопоставляется указатель на связный список узлов-потомков. Если требуется хранить дополнительную информацию о каждом узле (например, метка или ключ), то необходимо дополнить структуру данных соответствующим атрибутом. Тогда каждый узел дерева представляется комбинацией атрибута «Метка» и указателя на начало связного списка узлов-потомков. При этом элементы списка узлов-потомков содержат указатель на один из узлов-потомков, а также указатель на следующий элемент списка. На рис. 2 приведен пример структуры дерева и соответствующее ему представление в форме связных списков.

Использование собственной структуры данных на основе связных списков является более естественной формой хранения древовидных структур за счет сохранения связей родитель–потомок. В свою очередь, это значительно упрощает реализацию алгоритмов обработки таких структур данных.

Хранение древовидных структур в постоянной памяти может быть организовано:

1. Если древовидная структура представлена в форме массивов, то ее хранение в постоянной памяти можно организовать в виде бинарного файла, содержащего последовательность элементов массива в байтовом представлении. В качестве альтернативного варианта хранения можно представить последовательность элементов массива в виде строки символов и записать ее в текстовый файл. Например, представленная на рис. 1 структура дерева в форме массивов определяется следующими двумя строками символов: «[0,0,0,1,1,1]» и «[A,B,C,D,E,F]».

2. Если древовидная структура представлена в форме связных списков, то ее хранение в постоянной памяти можно организовать в виде текстового файла, содержащего строку символов со скобочным представлением (правильная скобочная последовательность) [11]. В таком случае каждому узлу дерева сопоставляется пара открывающей и закрывающей скобок, при этом внутреннее содержание скобок соответствует описанию узлов-потомков. Например, представленная на рис. 2 структура дерева в форме связных списков определяется следующей строкой символов со скобочным представлением: «((( )( )( ))( ))». Указанный способ позволяет хранить только структуру дерева (связи между узлами). Если требуется хранить дополнительную информацию о каждом узле (например, метка или ключ), то можно разместить такую информацию перед открывающей скобкой: «A(B(D( )E( )F( ))C( ))».



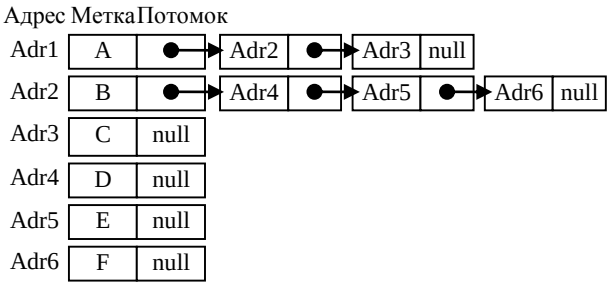


Рис. 2. Пример структуры дерева и соответствующее
ему представление в форме связных списков

Кроме того, могут быть организованы и другие способы сериализации объектов, представляющих собой древовидную структуру на основе связных списков. Например, одним из таких способов является применение JSON (JavaScript Object Notation) – текстового формата обмена данными на основе языка программирования JavaScript [12]. Данный инструмент позволяет хранить массив неупорядоченных пар «ключ:значение», определяющих структуру и содержание рассматриваемого объекта с возможностью последующего восстановления.

3. Наиболее распространенным способом для хранения большого объема данных, требующего эффективной реализации оперативного изменения и поиска необходимой информации, является приме-

нение баз данных и функциональных возможностей соответствующих им систем управления. Выбор конкретного способа представления древовидных структур в реляционных базах данных формирует требования к реализации программного обеспечения для работы с такими структурами данных, при этом сложность их обработки будет напрямую зависеть от глубины узлов [13].

В работе [14] представлены следующие возможные способы представления древовидных структур в реляционных базах данных: список смежных вершин, вложенное множество, материализованные пути. Например, идея хранения древовидной структуры с использованием списка смежных вершин заключается в записи одного кортежа для каждого узла дерева, содержащего атрибут с идентификатором кортежа родительского узла. Такой способ аналогичен описанному выше представлению структуры дерева с помощью массивов, эффективен с точки зрения требуемого объема памяти, но плохо применим для последующей обработки. Если же хранить больше информации о связях между узлами в дереве (например, добавив в кортеж атрибут с информацией о пути от корневого узла до рассматриваемого узла), то можно повысить скорость обработки древовидных структур.

Для решения задачи хранения больших объемов данных о древовидной структуре с множеством связей более подходящим способом является использование графовых баз данных. Графовые базы данных относятся к нереляционным и базируются на теории графов [15–17]. Само по себе хранение иерархических структур в графовых базах данных интуитивно понятно, так как за основу организации информации уже взята древовидная структура [18]. Графовые базы данных обладают более высокой скоростью обработки запросов для масштабных древовидных структур с большим количеством связей.

Способы хранения структур деревьев И/ИЛИ и их вариантов

Дерево И/ИЛИ представляет собой древовидную структуру, которая позволяет оперировать двумя типами внутренних узлов: И-узел (соответствует операции декартова произведения множеств и используется для объединения узлов-потомков) и ИЛИ-узел (соответствует операции объединения непересекающихся множеств и используется для выбора одного из узлов-потомков) [7]. Графическое представление И-узла отличается наличием дополнительной линии, объединяющей его ребра, ведущие к узлам-потомкам. Кроме того, структура дерева И/ИЛИ порождает конечное множество соответствующих ей древовидных структур, каждая из которых называется вариантом дерева И/ИЛИ. Чтобы получить один возможный вариант заданной структуры дерева И/ИЛИ, необходимо для каждого его ИЛИ-узла оставить только один узел-потомок и его поддерево.

На рис. 3 представлен пример помеченной структуры дерева И/ИЛИ, а также все 6 соответствующих ему вариантов.

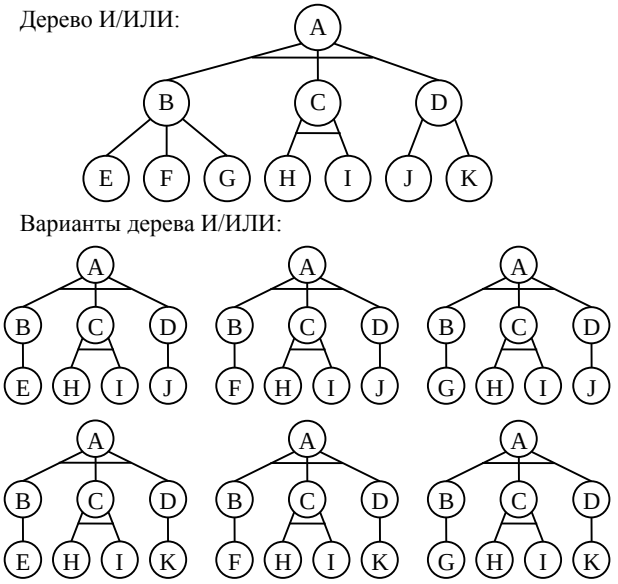


Рис. 3. Пример структуры дерева И/ИЛИ и его варианты

Для решения задачи хранения структуры дерева И/ИЛИ в оперативной памяти можно воспользоваться описанными выше способами хранения древовидных структур: на основе массивов или на основе связных списков. Однако для случая деревьев И/ИЛИ структуру данных необходимо дополнить информацией о типе узла (И-узел или ИЛИ-узел). При использовании представления на основе массивов это отразится в наличии дополнительного массива, аналогичного массиву с информацией о метке каждого узла. При использовании представления на основе связных списков это отразится в наличии дополнительного атрибута «Тип узла».

В отличие от хранения структуры дерева И/ИЛИ, обработка конкретного хранимого его варианта требует явного указания порядкового номера выбранного узла-потомка для каждого ИЛИ-узла. В случае отсутствия такой информации в используемом представлении варианта дерева И/ИЛИ нужно предварительно реализовать сопоставление рассматриваемого варианта с исходной структурой дерева И/ИЛИ. Таким образом, для ускорения процесса обработки необходимо введение еще одного дополнительного массива (при использовании представления на основе массивов) или атрибута (при использовании представления на основе связных списков). Заметим, что указанная дополнительная информация о типе узла и о порядковом номере выбранного узла-потомка для ИЛИ-узла может быть описана единым массивом (или атрибутом). Например, значение элемента массива (или атрибута), равное нулю, соответствует типу И-узла или конечному узлу, а значение больше нуля соответствует типу ИЛИ-узла и одновременно показывает порядковый номер выбранного узла-потомка для ИЛИ-узла.

Далее рассмотрим способы хранения структуры дерева И/ИЛИ и его вариантов в постоянной памяти.

Если структура дерева И/ИЛИ либо его вариантов представлена в форме массивов, хранящих информацию о родительских узлах, о метках узлов и о

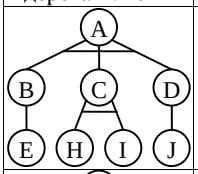
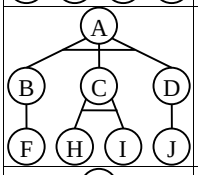
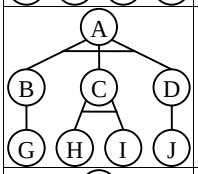
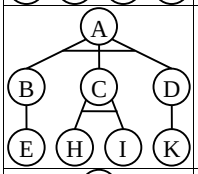
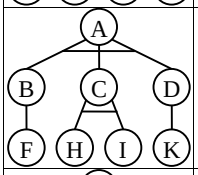
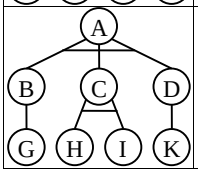
типах узлов, то последовательности элементов таких массивов можно записать в виде строк символов текстового файла. В таблице приведен пример представления вариантов дерева И/ИЛИ, изображенных на рис. 3, в форме массивов.

Если использовать скобочное представление древовидных структур для деревьев И/ИЛИ, то разный тип узлов может быть отражен путем применения скобок разного вида. Например, можно использовать круглые скобки вида «(» и «)» для И-узлов и конечных узлов вместе с фигурными скобками вида «{» и «}» для ИЛИ-узлов. При этом для варианта дерева И/ИЛИ информация о порядковом номере выбранного узла-потомка для ИЛИ-узла может быть записана внутри соответствующих фигурных скобок. Пример скобочного представления о каждом варианте дерева И/ИЛИ, изображенного на рис. 3, также приведен в крайнем справа столбце таблицы.

Кроме того, могут быть организованы и другие способы сериализации объектов, представляющих собой структуру дерева И/ИЛИ либо его вариантов на основе связных списков (например, через JSON).

Однако самым эффективным с точки зрения требуемого объема памяти для хранения набора вариантов дерева И/ИЛИ является использование алгоритмов комбинаторной генерации, а именно алгоритма ранжирования [7]. Указанный класс алгоритмов комбинаторной генерации позволяет для заданной структуры варианта дерева И/ИЛИ вычислить уникальное числовое значение, называемое рангом. Для восстановления исходной структуры варианта дерева И/ИЛИ необходимо выполнить обратный алгоритм, называемый алгоритмом генерации по рангу. Следовательно, для хранения множества конкретных вариантов дерева И/ИЛИ достаточно знать структуру соответствующего дерева И/ИЛИ, а также значения рангов хранимых вариантов.

Пример представления вариантов дерева И/ИЛИ в форме массивов и через скобочную запись

Вариант дерева И/ИЛИ	Ранг	Представление в форме массивов			Скобочное представление
		№ 1 (родительский узел)	№ 2 (метка узла)	№ 3 (тип узла)	
	0	[0,0,0,0,1,2,2,3]	[A,B,C,D,E,H,I,J]	[0,1,0,1,0,0,0,0]	A(B{1,E( )}C(H( )I( ))D{1,J( )})
	1	[0,0,0,0,1,2,2,3]	[A,B,C,D,F,H,I,J]	[0,2,0,1,0,0,0,0]	A(B{2,F( )}C(H( )I( ))D{1,J( )})
	2	[0,0,0,0,1,2,2,3]	[A,B,C,D,G,H,I,J]	[0,3,0,1,0,0,0,0]	A(B{3,G( )}C(H( )I( ))D{1,J( )})
	3	[0,0,0,0,1,2,2,3]	[A,B,C,D,E,H,I,K]	[0,1,0,2,0,0,0,0]	A(B{1,E( )}C(H( )I( ))D{2,K( )})
	4	[0,0,0,0,1,2,2,3]	[A,B,C,D,F,H,I,K]	[0,2,0,2,0,0,0,0]	A(B{2,F( )}C(H( )I( ))D{2,K( )})
	5	[0,0,0,0,1,2,2,3]	[A,B,C,D,G,H,I,K]	[0,3,0,2,0,0,0,0]	A(B{3,G( )}C(H( )I( ))D{2,K( )})

Например, если для хранения вариантов дерева И/ИЛИ использовать представление на основе массивов, то количество требуемых бит памяти определяется следующим выражением:

$$N_{\text{nodes}} \cdot \left(B_1 + B_2 + B_3\right), \qquad (1)$$

где N_{nodes} показывает количество узлов в хранимом варианте дерева И/ИЛИ, $B_1 = \lceil \log_2 N_{\text{nodes}} \rceil$ определяет количество бит для хранения одного элемента массива № 1 (родительский узел), B_2 определяет количество бит для хранения одного элемента мас-

сива № 2 (метка узла), B_3 определяет количество бит для хранения одного элемента массива № 3 (тип узла). Кроме того, необходимо учесть округление до целого числа для количества байтов, которое требуется при программной реализации одномерных массивов. В таком случае для хранения каждого варианта дерева И/ИЛИ, изображенного на рис. 3, потребуется $8\cdot(8+8+8)=192$ бит памяти.

Если для каждого варианта дерева И/ИЛИ выполнить алгоритм ранжирования, то количество требуемых бит памяти для их хранения вычисляется по формуле

$$\left\lceil \log_2 |W(D)| \right\rceil, \quad (2)$$

где $|W(D)|$ показывает мощность множества вариантов заданной структуры дерева И/ИЛИ D. В таком случае для хранения каждого варианта дерева И/ИЛИ, изображенного на рис. 3, потребуется $\lceil \log_2 6 \rceil = 3$ бит памяти.

Указанные выше способы хранения структуры дерева И/ИЛИ и его вариантов в постоянной памяти направлены на сокращение объема занимаемой памяти. Однако такой подход усложняет последующую обработку древовидных структур, так как отсутствуют эффективные реализации соответствующих алгоритмов и появляется необходимость предварительной записи из постоянной в оперативную память.

Для решения данной проблемы можно организовать хранение древовидных структур в базе данных и реализовывать их обработку с помощью функционала соответствующей системы управления базой данных. Это также применимо и для хранения структуры дерева И/ИЛИ, но для случая хранения вариантов дерева И/ИЛИ потребуется наличие собственной базы данных для каждого варианта дерева И/ИЛИ, либо можно хранить в одной базе данных множество разных древовидных структур. Если хранить все варианты дерева И/ИЛИ в одной базе данных, то для установки связей родитель–потомок конкретного варианта потребуется просмотреть все содержимое базы данных, что является вычислительно сложной операцией при хранении множества вариантов дерева И/ИЛИ. Следовательно, данный способ хранения вариантов дерева И/ИЛИ также является неэффективным с точки зрения их последующей обработки.

Чтобы решить такую задачу, предлагается рассмотреть биективное отображение структуры дерева И/ИЛИ на схему реляционной базы данных. В основе реляционной модели данных лежит представление данных в виде таблиц и связей между ними. Тогда каждому узлу варианта дерева И/ИЛИ можно сопоставить одну таблицу, при этом связь родитель–потомок для узлов дерева реализуется за счет связи между соответствующими таблицами. Для произвольной структуры дерева И/ИЛИ соответствующие ей таблицы в схеме реляционной базы данных можно поделить на следующие три типа:

1. Таблица «Root» для корня дерева И/ИЛИ. Структура данной таблицы представляет собой совокупность атрибутов:

– атрибут «ID_Root» – идентификатор записи таблицы (PK – первичный ключ);

– атрибут «Data» – дополнительная информация об узле дерева И/ИЛИ;

– набор атрибутов вида «Child_i» – связь узла дерева И/ИЛИ с таблицами узлов-потомков (FK – внешний ключ).

Каждая запись таблицы «Root» показывает информацию о корне конкретного варианта дерева И/ИЛИ. При этом тип узла определяется наличием или отсутствием значения «null» для атрибутов вида «Child_i»:

– если запись таблицы не содержит значения «null» для атрибутов вида «Child_i», то рассматриваемый узел является И-узлом;

– если запись таблицы содержит значение «null» для всех атрибутов вида «Child_i», кроме одного, то рассматриваемый узел является ИЛИ-узлом и выбранный в варианте дерева И/ИЛИ узел-потомок определяется атрибутом с отличным значением от «null».

2. Таблицы «Node_j» для внутренних узлов дерева И/ИЛИ, структура которых аналогична структуре таблицы «Root».

3. Таблица «Leaves» для всех конечных узлов дерева И/ИЛИ. Данная таблица отличается от предыдущих отсутствием внешних ключей, так как у таких узлов отсутствуют узлы-потомки.

На рис. 4 приведен пример представления структуры дерева И/ИЛИ, изображенного на рис. 3, в форме схемы реляционной базы данных.

Таким образом, конкретный вариант дерева И/ИЛИ определяется одной записью в таблице «Root» реляционной базы данных, а также совокупностью всех связанных с нею через внешние ключи записей остальных таблиц. Следовательно, это избавляет от необходимости просмотра всего содержимого базы данных для установки связей родитель–потомок конкретного варианта дерева И/ИЛИ. Кроме того, разные варианты дерева И/ИЛИ могут содержать одинаковое поддерево. В таком случае они могут быть связаны внешним ключом с одной и той же записью таблицы базы данных, соответствующей корню одинакового поддерева, что уменьшит избыточность хранимой информации.

Заключение

Решение вопросов, связанных с организацией хранения древовидных структур, является важной задачей для информационных систем, обрабатывающих такие данные. В данной работе представлены широко используемые способы хранения древовидных структур как в оперативной, так и постоянной компьютерной памяти.

Основным результатом представленного исследования является адаптация данных способов к задаче хранения структур деревьев И/ИЛИ и их вариантов. Наиболее эффективным с точки зрения

требуемого объема памяти для хранения набора вариантов дерева И/ИЛИ является использование алгоритма ранжирования, так как каждый хранимый вариант дерева И/ИЛИ кодируется всего лишь одним числовым значением.

Однако организация последующей обработки хранимых древовидных структур требует полного восстановления структуры варианта дерева И/ИЛИ. Для оперативной работы с вариантами структуры дерева И/ИЛИ можно воспользоваться предложенным биективным отображением структуры дерева И/ИЛИ на схему реляционной базы данных. Тогда функциональные возможности соответствующей системы управления базой данных позволят эффективно обрабатывать хранимые данные.

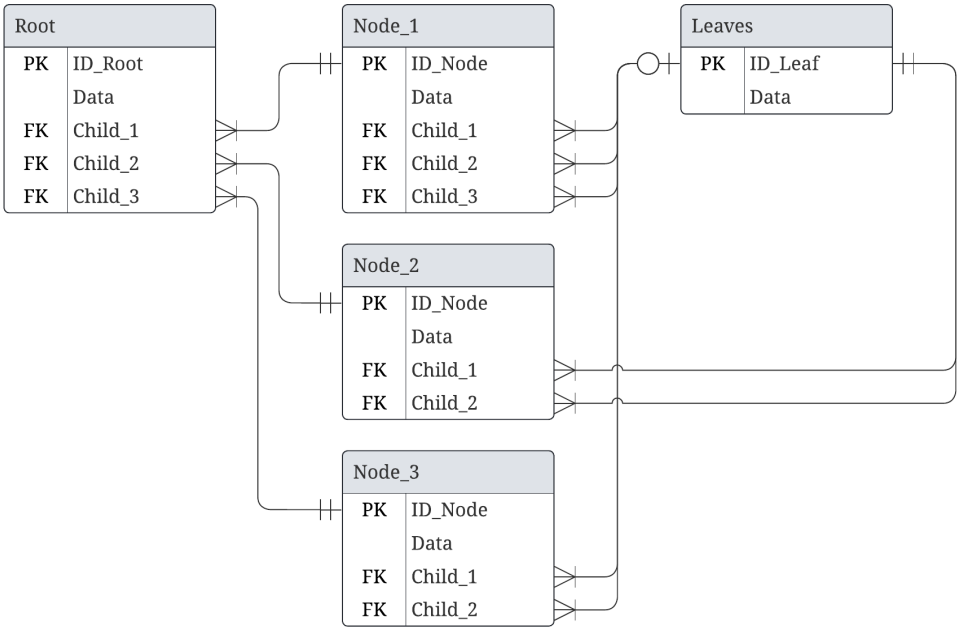


Рис. 4. Пример представления структуры дерева И/ИЛИ в форме схемы реляционной базы данных (ER-диаграмма)

Работа выполнена при финансовой поддержке Российского научного фонда в рамках научного проекта № 22-71-10052.

Литература

1. Subero A. Codeless data structures and algorithms. – USA, CA, Berkeley: Apress, 2020. – 159 p.

2. Celko J. Trees and hierarchies in SQL for smarties. – USA: Morgan Kaufmann Publishers, 2012. – 296 p.

3. Aho A.V. Data structures and algorithms / A.V. Aho, J.E. Hopcroft, J.D. Ullman. – USA: Addison-Wesley. – 1983. – 427 p.

4. Михеев М.Ю. Древовидные карты для повышения качества поддержки решений / М.Ю. Михеев, О.В. Прокофьев, И.Б. Семочкина // Надежность и качество сложных систем. – 2021. – № 1. – С. 76–86.

5. Шаров В.Ю. Сохранение древовидных структур в базе данных и управление ими / В.Ю. Шаров, Ю.Ю. Горшкова, И.Н. Филоненко // Комплексные проблемы развития науки, образования и экономики региона. – 2015. – № 2. – С. 223–231.

6. Knuth D. The art of computer programming: Vol. 1: Fundamental algorithms. – USA: Addison-Wesley. – 1997. – 654 p.

7. Кручинин В.В. Метод кодирования информационных объектов на основе деревьев И/ИЛИ / В.В. Кручинин, Б.А. Люкшин // Доклады ТУСУР. – 2010. – № 1 (21). – С. 170–172.

8. Зорин Ю.А. Интерпретатор языка построения генераторов тестовых заданий на основе деревьев И/ИЛИ // Доклады ТУСУР. – 2013. – № 1 (27). – С. 75–79.

9. Шабля Ю.В. Метод сжатия данных журналов событий на основе теории комбинаторной генерации с применением структур деревьев И/ИЛИ // Современные информационные технологии и ИТ-образование. – 2023. – Т. 19, № 3.

10. Токарева А.В. Модификация метода идентификации и прослеживаемости сложных технических изделий с применением алгоритмов комбинаторной генерации на основе деревьев И/ИЛИ / А.В. Токарева, Д.В. Кручинин // Вестник СибГУТИ. – 2024. – Т. 18, № 3 (в печати).

11. Бабенко М.А. Введение в теорию алгоритмов и структур данных / М.А. Бабенко, М.В. Левин. – М.: ФМОП, МЦНМО. – 2012. – 144 с.

12. Wang Z. Exploiting common patterns for tree-structured data / Z. Wang, S. Chen // Proceedings of the 2017 ACM International Conference on Management of Data. – 2017. – P. 883–896.

13. Гребенщиков Н.Н. Представление древовидной зависимости в реляционной базе данных // Программные продукты и системы. – 2008. – № 1. – С. 41–44.

14. Богданов Д.В. Оптимальный способ хранения и обработки древовидных структур в базах данных // Программные продукты и системы. – 2009. – № 1. – С. 140–142.

15. Засядко Г.Е. Проблемы разработки графовых баз данных / Г.Е. Засядко, А.В. Карпов // Инженерный вестник Дона. – 2017. – № 1.

16. Renzo A. Survey of graph database models / A. Renzo, C. Gutierrez // ACM Computing Surveys. – 2008. – Vol. 40, No. 1. – Article 1.

17. Гуральник Р.И. Некоторые задачи на графовых базах данных // Труды ИСП РАН. – 2016. – Т. 28, № 4. – С. 193–216.

18. Monteiro J. Experimental evaluation of graph databases: JanusGraph, Nebula Graph, Neo4j, and TigerGraph / J. Monteiro, F. Sa, J. Bernardino // Applied Sciences. – 2023. – Vol. 13, No. 9. – Article 5770.

Шабля Юрий Васильевич
Канд. техн. наук, с.н.с. лаборатории алгоритмов и технологий исследования дискретных структур (ЛАТИДС) Томского государственного ун-та систем управления и радиоэлектроники (ТУСУР)
Ленина пр-т, 40, г. Томск, Россия, 634050
ORCID: 0000-0002-9695-7493
Тел.: +7-906-949-03-07
Эл. почта: syv@fb.tusur.ru

Токарева Алина Вячеславовна
Аспирант каф. технологий электронного обучения ТУСУРа
Ленина пр-т, 40, г. Томск, Россия, 634050
ORCID: 0000-0003-1353-0177
Тел.: +7-953-928-94-13
Эл. почта: alina.v.tokareva@tusur.ru

Shablya Y.V., Tokareva A.V.
Methods for storing AND/OR tree structures and their variants in RAM and ROM

Tree structures are widely used to represent information objects containing hierarchical relations between their parts. An example of such tree structures are AND/OR trees, that have found their application in the development of combinatorial generation algorithms and related tasks. This paper explores possible ways to store tree structures in the random access memory (RAM) and read only memory (ROM) of devices that process these structures. The adaptation of these methods to the problem of storing AND/OR tree structures and their variants is also considered. In addition, to organize operational work with stored variants of the AND/OR tree, the authors propose the bijective mapping of an AND/OR tree structure to a relational database schema.
Keywords: tree structure, AND/OR tree variant, random access memory, read only memory, data structure, linked list, relational database.
DOI: 10.21293/1818-0442-2024-27-2-44-50

References

1. Subero A. *Codeless data structures and algorithms*. USA, CA, Berkeley, Apress, 2020, 143 p.
2. Celko J. *Trees and hierarchies in SQL for smarties*. USA, Morgan Kaufmann Publishers, 2012, 296 p.
3. Aho A.V., Hopcroft J.E., Ullman J.D. *Data structures and algorithms*. USA, Addison-Wesley, 1983, 427 p.
4. Miheev M.Y. Prokof'ev O.V., Semochkina I.B. [Treemaps to improve quality of support of decision]. *Reliability and Quality of Complex Systems,* 2021, no. 1, pp. 76–86 (in Russ.).
5. Sharov V.Y., Gorshkova Y.Y., Filonenko I.N. [Preservation of treelike structures in a database and management of them]. *Complex Problems of Development of Science, Education and Economics of the Region*, 2015, no. 2, pp. 223–231 (in Russ.).
6. Knuth D. *The art of computer programming. Volume 1: Fundamental algorithms*. USA, Addison-Wesley, 1997, 654 p.
7. Kruchinin V.V., Lukschin B.A. [Method of coding of information objects on the basis of trees and-or]. *Proceedings of TUSUR University,* 2010, no. 1(21), pp. 170–172 (in Russ.).
8. Zorin Y.A. [The interpreter of programming language for design generators of tests based on AND/OR trees]. *Proceedings of TUSUR University,* 2013, no. 1(27), pp. 75–79 (in Russ.).
9. Shablya Y.V. [A method for compressing event log data based on combinatorial generation using AND/OR tree structures]. *Modern Information Technologies and IT-education,* 2023, vol. 19, no. 3 (in Russ.).
10. Tokareva A.V., Kruchinin D.V. [Modification of the method for identifying and tracing complex technical products using combinatorial generation algorithms based on AND/OR trees]. *The Herald of the Siberian State University of Telecommunications and Information Science*, 2024, vol. 18, no. 3 (in Russ.).
11. Babenko M.A., M.V. *Vvedenie v teoriyu algoritmov i struktur dannyh* [Introduction to the theory of algorithms and data structures]. Moscow, FMOP, MCNMO Publ., 2012. 144 p. (in Russ.).
12. Wang Z., Chen S. Exploiting common patterns for tree-structured data. *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 883–896.
13. Grebenshchikov N.N. [Representation of tree dependency in a relational database]. *Software Products and Systems*, 2008, no. 1, pp. 41–44 (in Russ.).
14. Bogdanov D.V. [The optimal way to store and process tree structures in databases]. *Software Products and Systems*, 2009, no. 1, pp. 140–142.
15. Zasyadko G.E., Karpov A.V. [Problems of graph database development]. *Engineering Journal of Don*, 2017, no. 1 (in Russ.).
16. Renzo A., Gutierrez C. Survey of graph database models. *ACM Computing Surveys*, 2008, vol. 40, no. 1, Article 1.
17. Guralnik R.I. [Some problems on graph databases]. *Proc. ISP RAS*, 2016, vol. 28, no. 4, pp. 193–216 (in Russ.).
18. Monteiro J., Sa F., Bernardino J. Experimental evaluation of graph databases: JanusGraph, Nebula Graph, Neo4j, and TigerGraph. *Applied Sciences*, 2023, vol. 13, no. 9, Article 5770.

Yuriy V. Shablya
Candidate of Sciences in Engineering, Senior Researcher, Laboratory of Algorithms and Technologies for Discrete Structures Research, Tomsk State University of Control Systems and Radioelectronics (TUSUR)
40, Lenin pr., Tomsk, Russia, 634050
ORCID: 0000-0002-9695-7493
Phone: +7-906-949-03-07
Email: syv@fb.tusur.ru

Alina V. Tokareva
Postgraduate student, Department of e-Learning Technology, TUSUR
40, Lenin pr., Tomsk, Russia, 634050
ORCID: 0000-0003-1353-0177
Phone: +7-953-928-94-13
Email: alina.v.tokareva@tusur.ru