Вестник КРАУНЦ. Физ.-мат. науки. 2023. Т. 43. №2. С. 44-54. ISSN 2079-6641

ИНФОРМАЦИОННЫЕ И ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ

https://doi.org/10.26117/2079-6641-2023-43-2-44-54

Научная статья

Полный текст на русском языке

УДК 004.6

Бинарное кодирование иерархических структур

В. С. Кириллов*

Кабардино-Балкарский научный центр Российской академии наук,
360010, г. Нальчик, ул. Балкарова, 2, Россия

Аннотация. В данной статье представлен алгоритм, который дает расширенные возможности представления ключей для иерархических структур. Использование бинарного представления материализованного пути позволяет эффективно сортировать узлы путем побитного сравнения и быстро вычислять верхний и нижний пределы для всех ключей элементов поддерева. Эта методика широко применяется в проектировании баз данных и в задачах фильтрации информации. В работе проведено сравнение данного алгоритма с различными подходами, используемыми в известных серверах баз данных. Результаты исследования подтверждают эффективность предложенного метода и его преимущества по сравнению с альтернативными подходами. Он обеспечивает более быстрое выполнение операций сортировки и вычисления пределов ключей, что является критически важным для эффективного функционирования баз данных и обработки больших объемов информации. Таким образом, представленный алгоритм имеет значительное практическое применение и может быть полезным инструментом при разработке и оптимизации баз данных, а также в других задачах, связанных с обработкой и фильтрацией информации.

Ключевые слова: деревья данных, иерархии, реляционные базы данных и модели

Получение: 03.04.2023; Исправление: 12.04.2023; Принятие: 16.04.2023; Публикация онлайн: 30.06.2023

Для цитирования. Кириллов В. С. Бинарное кодирование иерархических структур // *Вестник КРАУНЦ. Физ.-мат. науки.* 2023. Т. 43. № 2. С. 44-54. EDN: XUMKPG. https://doi.org/10.26117/2079-6641-2023-43-2-44-54.

Финансирование. Работа выполнялась без поддержки фондов.

Конкурирующие интересы. Конфликтов интересов в отношении авторства и публикации нет.

Авторский вклад и ответственность. Автор участвовал в написании статьи и полностью несет ответственность за предоставление окончательной версии статьи в печать.

***Корреспонденция:** E-mail: vkirillov74@gmail.com

INFORMATION AND COMPUTATION TECHNOLOGIES

https://doi.org/10.26117/2079-6641-2023-43-2-44-54

Research Article

Full text in Russian

MSC 68T99

Binary Coding of Hierarchical Structures

V. S. Kirillov*

Kabardino-Balkarian Scientific Center of the Russian Academy of Sciences
360010, Nalchik, 2 Balkarov street, Russia

Abstract. This article presents an algorithm that provides enhanced capabilities for representing keys in hierarchical structures. By using a binary representation of the materialized path, it allows efficient sorting of nodes through bitwise comparison and rapid computation of upper and lower bounds for all keys within the subtree. This methodology finds widespread application in database design and information filtering tasks. The study compares this algorithm with various approaches used in well-known database servers. The research findings confirm the effectiveness of the proposed method and its advantages over alternative approaches. It enables faster execution of sorting operations and computation of key bounds, which are critical for the efficient functioning of databases and processing large volumes of information. Therefore, the presented algorithm holds significant practical relevance and can serve as a valuable tool in the development and optimization of databases, as well as in other tasks related to information processing and filtering.

Key words: trees data, hierarchies, relational database & models

Received: 03.04.2023; Revised: 12.04.2023; Accepted: 16.04.2023; First online: 30.06.2023

For citation. Kirillov V.S. Binary coding of hierarchical structures. *Vestnik KRAUNC. Fiz.-mat. nauki.* 2023, **43**: 2, 44-54. EDN: XUMKPG. https://doi.org/10.26117/2079-6641-2023-43-2-44-54.

Funding. The work was carried out without the support of funds.

Competing interests. There are no conflicts of interest regarding authorship and publication.

Contribution and Responsibility. The author participated in the writing of the article and is fully responsible for submitting the final version of the article to the press.

***Correspondence:** E-mail: vkirillov74@gmail.com

Введение

При проектировании информационных систем, разработчик часто сталкивается с необходимостью фильтрации данных, образующих иерархические структуры. Классическим примером таких структур является почтовый адрес.

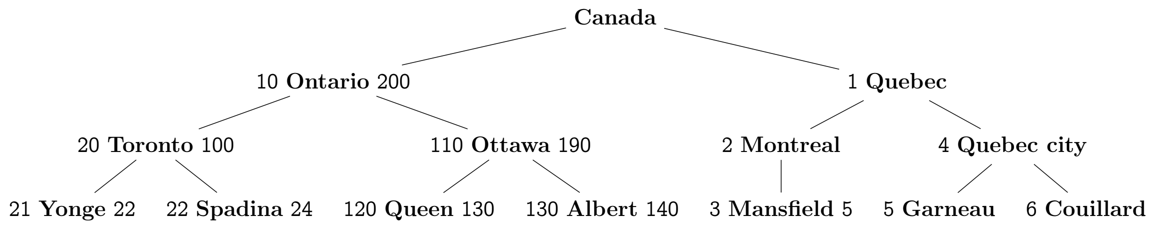

Рис. 1. Иерархическая структура.
[Figure 1. Hierarchical structure.]

В настоящее время существует несколько подходов для кодирования информации в структуре иерархии. В частности, большое распространение получили следующие подходы: подход основанный на списке смежности [12], [13], таблице замыкания [4], интервалах [11], [14], материализированный путь [11], [12], [13], [14], [15], [1], [6]. Также, широко используются некоторые другие проприетарные решения [5].

Список смежности - самый простой и легко исполняемый способ. Он основан на записи информации о родительском узле в записи дочернего узла. Однако, при выборке поддеревьев для некоторой родительской записи, мы сталкиваемся с рекурсивным запросом к данным, что значительно сказывается на производительности системы. Кроме этого, к недостаткам можно отнести сложность сохранения целостности информации. Для хранения иерархических данных - это самый интуитивно понятный и неудачный способ. В некоторых разновидностях систем используют отдельную таблицу для хранения информации родительский узел - узел потомок. Однако, при этом все недостатки первоначального метода остаются и здесь. При использовании таблицы замыкания, мы в отдельную таблицу сохраняем все потомки первого поколения для данного узла. Это также влечет за собой рекурсию.

Метод основанный на таблице замыкания базируется на том, что в отдельную таблицу мы записываем пары родительский узел - узел потомок. При этом, узлом потомком считается любой узел входящий в поддерево данного родительского узла. Этот подход не ведет к рекурсивной выборке, однако количество записей в таблице оценивается $O(n^2)$ в худшем случае. Тем не менее, в реальных системах это значение значительно меньше. К недостаткам этого алгоритма можно отнести сложность обновления информации при изменении иерархии, а также большой размер таблицы. В алгоритме, основанном на интервалах, каждому узлу сопоставляется 2 числа, которые характеризуют наименьшее и наибольшее значение для всех нисходящих узлов данного родительского узла. Это решение дает достаточно хорошие результаты для поиска поддеревьев, однако не лишено недостатков. При добавлении нисходящего узла к некоторому родительскому, у которого разница между минимальным и максимальным числом равна единице, приходится к перенумеровывать некоторое количество узлов. На рис 1., этот подход показан для поддерева Ontario. Как легко можно заметить, добавление листа к Yonge затруднительно.

Достаточно широкое распространение получило решение, основанное на материализированном пути. Подобные решения были применены в PostgreSQL [2], MSSQL [1]. Решение `ltree`, предложенное в PostgreSQL, основано на записи всех предков данной вершины через разделительный знак. Например: Quibec.Montreal.Mansfield. Алгоритмы работы со строками позволяют найти потомков, родителей и т.д. для данной записи. Интересное решение предложили в MSSQL - `ordpath`. В данном решении строится последовательность битов, где номера узлов кодируются последовательностью похожей на применяемую в архиваторах, когда у нас есть прификс и значащие биты. При этом, для наиболее часто используемых номеров применяются последовательности битов меньшей длины. Номера узлов разделяются битом-флагом, который сигнализирует, что мы переходим на следующий уровень иерархии или данный узел был вставлен между двумя существующими узлами. К недостатку данного подхода можно отнести то, что уровень иерархии и номера узлов можно определить только после парсинга всех префиксов и флагов, а также, достаточно большую сложность кодирования информации об иерархии.

Следующий подход представляет собой дальнейшее развитие метода интервалов в [7], [8], [9], [10]. При кодировании информации используется техника продолжающегося деления и дроби Фарей. К недостаткам данного метода можно отнести сложность вычисления индекса.

Также существует ряд разработок, связанных со специфическим способом хранения информации в базах данных. В частности, в PostgreSQL реализован способ хранения иерархической информации в `rtree` деревьях. Однако, подобные структуры данных выходят за рамки данной работы [3].

Теория

Основными требованиями к индексу, реализующему материализованный путь, являются: простота вычисления верхнего и нижнего пределов, используемых для фильтрации узлов поддеревьев, простота вычисления самого индекса, малый размер, а следовательно и малое время сравнения двух узлов иерархии. В данной работе рассмотрены только древовидные иерархии, в которых наследник имеет только один родительский узел. Рассмотрим случай, когда узлы иерархии имеют сквозную нумерацию. Например, поддерево Регион 2 (рисунок 1) имеет сквозную нумерацию узлов. Кодирование информации будем осуществлять в бинарной форме. Запишем материализованный путь для узла улица Carneau: 1.4.5. Для кодирования информации, запишем данные числа в бинарной форме:$1_{10} = 1_2$, $4_{10} = 100_2$, $5_{10} = 101_2$. Запишем биты данных чисел в нечетные биты индекса, начиная со старшего. Несмотря на то, что номер узла может иметь 64 битное и более значение мы будем записывать только биты, начиная со старшего значащего бита в номере индекса слева направо в биты индекса тоже слева направо. В четные биты запишем 0, если слева и справа у нас записаны биты номера одной вершины и 1 в противном случае (рисунок 2). Как нетрудно заметить, 1 в четном бите является разделителем между номерами узлов различных слоев иерархии. В дальнейшем, этот факт позволит достаточно просто узнавать глубину иерархии данного узла.

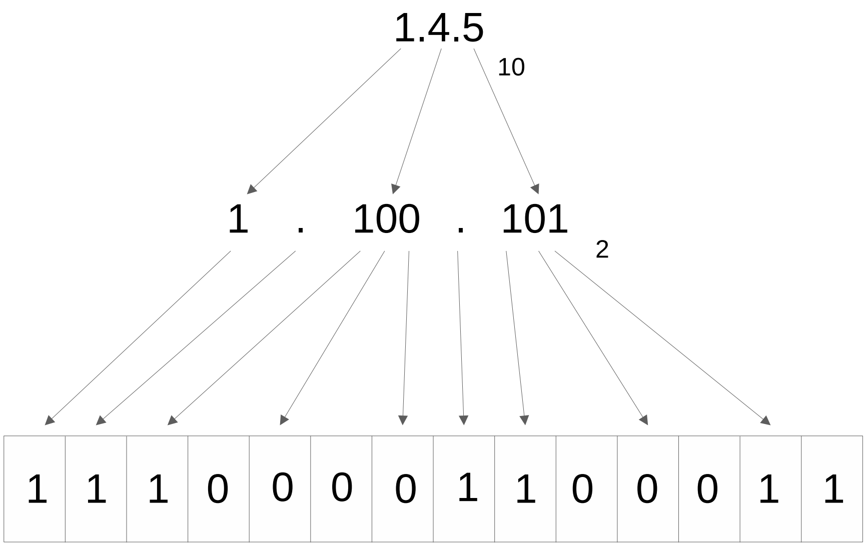

Рис. 2. Бинарное кодирование материализованного пути.
[Figure 2.Binary encoding of materialized path]

Легко заметить, что у нас получается структура переменной длины. Современные процессоры в основном являются 64 битными, поэтому длину материализованного пути сделаем кратной 64 битам. Сравнение двух материализованных путей производим слева направо, беря соответствующие 64 битные слова. Алгоритм сравнения на языке C++ приведен ниже:

```
int HTreeHelper::hrtree_cmp(
 const shared_ptr<vector<uint64_t>> a,
 const shared_ptr<vector<uint64_t>> b){
        size_t keysizea = a->size();
        size_t keysizeb = b->size();
        size_t minkey = (keysizea < keysizeb ? keysizea : keysizeb);
        int i;
        auto it1 = a->begin();
        auto it2 = b->begin();
        for (i = 0; i < minkey; ++i, it1++, it2++) {
               if (*it1 < *it2) return -1;
               if (*it1 > *it2) return 1;
        }
        // it is defenetelly one key child of another
        if (keysizea < keysizeb) return 1;
```

```
        if (keysizea > keysizeb) return -1;

        return 0;
}
```

Для определения нижней границы поддерева, для данного родительского ключа, возьмем либо родительский ключ, если 1 символ обозначающий окончание индекса родительского узла не приходится на 0 бит, либо добавим еще одно 64 битное слово заполненное 0. Алгоритм генерации нижнего предела для узлов поддерева приведен ниже:

```
shared_ptr<vector<uint64_t>> HTreeHelper::getDownLimitForChild(
const shared_ptr<vector<uint64_t>> currentNode)
{
        size_t downSize = currentNode->size();
        if (currentNode->back() & 1) downSize++;
        shared_ptr<std::vector<uint64_t>> ret =
        make_shared<vector<uint64_t>>(downSize);
        copy(currentNode->begin(), currentNode->end(),
        ret->begin());
        if (downSize > currentNode->size()) ret->back() = 0;
        return ret;
}
```

Верхняя граница для поддерева может быть найдена следующим образом: если родительский ключ оканчивается по границе 64 битного слова, то возьмем этот ключ, в противном случае дополним слово до границы единичными битами. Алгоритм генерации верхнего предела для узлов поддерева приведен ниже:

```
shared_ptr<vector<uint64_t>> HTreeHelper::getUpperLimitForChild(
const shared_ptr<vector<uint64_t>> currentNode)
{
        shared_ptr<vector<uint64_t>> ret =
        make_shared<vector<uint64_t>>(currentNode->size());
        copy(currentNode->begin(), currentNode->end(), ret->begin());
        int n = ctz(ret->back());
        ret->back() |= zerrosMask[ctz(ret->back())];
        return ret;
}
```

В данных алгоритмах использованы процедура `ctz - count tailing zeros` [16].

Утверждение 1: Для кажного узла поддерева все ключи `childKey>getDownLimitForChild(parentKey)`. На рисунке 2, приведены примеры формирования верхнего и нижнего пределов.

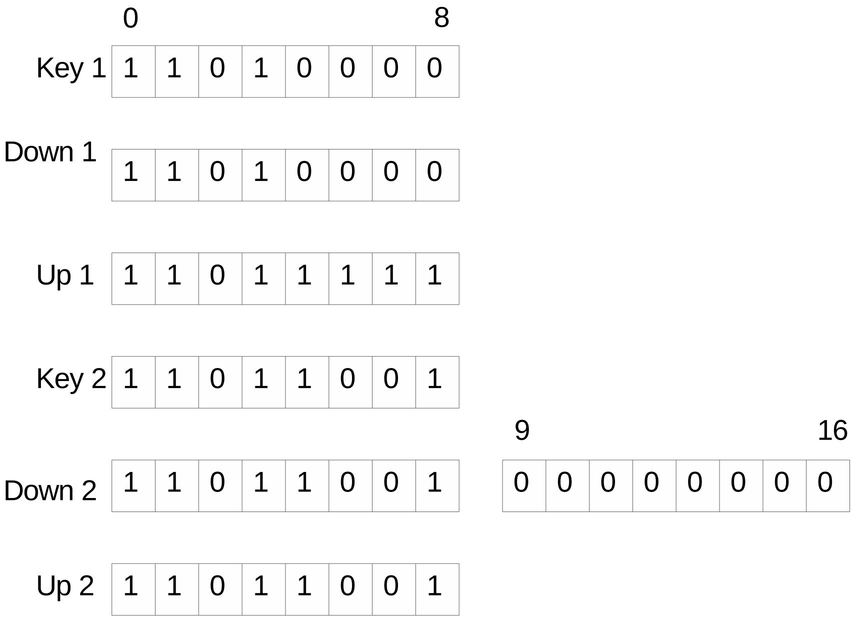

Рис. 3. Вычисленные верхние и нижние пределы для ключей.
[Figure 3. Computed upper and lower limits for keys.]

Доказательство. Допустим родительский ключ занимает n бит, включая завершающий 1 бит. Следовательно, ID потомка будет располагаться начиная с $n+1$ бита. Рассмотрим 2 случая:

1. n mod 64 != 0 (`key1`). В этом случае, в последнем 64-битном слове будет добавленно либо 1 в $n-1$ разряд (если `childId` > 0), либо 01, если `childId`$==0$. В обоих случаях, мы получаем `childId>parentId`.

2. n mod 64 == 0. В этом случае, `ChildId` будет распологаться в следующем 64 битном слове (`key2`). В соответсвии с алгоритмом генерации верхнего предела, мы добавим еще одно нулевое 64 битное слово и таким образом получим `childId>parentId`.

Утверждение доказано. □

Утверждение 2: Для кажного узла поддерева все ключи `childKey<=getUpperLimitForChild(parentKey)`.

Доказательство. Соглано алгоритму формирования верхней границы номеров поддерева, мы заполняем остаток 64 битного слова единицами. В этом случае, если у нас первые n бит совпадают и все биты, считая слева, $k > n$ равны 1, то в случае, если у узла потомка номер больше 1, мы получаем 0 в бите $n+2$, если номер дочернего узла равен 0 то мы получаем 0 в бите $n+1$. В случае, если номер узла потомка равен 1, то к родительскому узлу добавляется 11 и таким образом возможна ситуация когда `childKey==getUpperLimitForChild(parentKey)`, однако она не превышает этого значения. Если у нас возникает ситуация, когда биты `childKey` совпадают с битами `getUpperLimitForChild(parentKey)` в пределах длины верхней границы и при этом длина childKey больше мы считаем, что `childKey<getUpperLimitForChild(parentKey)`. Таким образом, комбинация

операции сравнения и формирования верхнего предела дает необходимый результат.

Утверждение доказано. □

Утверждение 3: Не существует ни одного ключа, который не является потомком данного, для которого `getDownLimitForChild(parentKey)<key<=getUpperLimitForChild(parentKey)`

Доказательство. Пусть некоторый parentKey имеет длину n. Для ключа узла, который не является потомком данного, существует вершина в пути к нему из корня, номер которой отличается от номера parentKey. Следовательно, в коде Key будет другая комбинация старших n бит и значит key не попадет в заданный диапазон.

Утверждение доказано. □

Анализ

Определим сложность генерации и размер ключа. Очевидно, что размер ключа равен удвоенному значению количества значащих битов номеров узлов. Среди рассматриваемых ключей только ordpath для маленьких номеров вершин дает более маленький ключ. При генерации ключа можно использовать, помимо битовых сдвигов, алгоритмы разработанные для работы с графикой. Как уже упоминалось ранее, при генерации верхнего и нижнего пределов используется алгоритм ctz [16], [17]. Данные алгоритмы используют 4 и 5 операций процессора. Для генерации самого ключа, также используется алгоритм ctz [18] и [21]. Данные алгоритмы используют 12 и 11 операций процессора.

Для определения глубины вершины использовуется битовая маска (операция and) и алгоритм подсчета единичных битов [19], [20]. Однако, у многих современных процессоров имеется соответствующая машинная команда и, в этом случае, эти алгоритмы не используются.

При реализации алгоритма генерации данного ключа, количество операций несколько выше вследствие того, что нам необходимо сдвигать полученные слова влево и вправо при объединении информации родительского ключа с номером дочернего. Однако, это не вызывает существенного увелечения длительности генерации ключа. При объединении родительского ключа со сгенерированным дочерним, используется до 6 операций побитового сдвига.

Задача изменения родительского ключа для некоторого поддерева также решается достаточно просто. В начале выполнения операции мы должны сгенерировать маску и убрать биты, принадлежащие убираемому родительскому ключу, сдвинуть биты влево или вправо, для того чтобы при копировании и ключ поддерева был скопирован в необходимое положение и произвести операцию побитового или. Следует отметить, что на каждое 64 битное слово у нас потребуется до двух операций битового сдвига.

При сравнении [2] можно отметить, что ltree структура более проста в исполнении, легче воспринимается человеком, позволяет сохранять текстовые ключи, одноко при этом генерируются более длинные ключи, в которых есть ограничения на используемые в ключах символы. Задачи поиска определенной вершины в индексе более интуитивно понятны, однако требуют больших вычислительных затрат.

Предложенная Microsoft структура [1], представляет меньшую длину ключа. Так, например, для диапазона значений от 16 до 79 здесь потребуется 12 бит и один бит для перехода на следующий уровень иерархии. В нашем случае, потребуется от 10 до 14 бит. При значениях номера вершины в диапазоне 5 200 - 4 294 972 495 для ordpath потребуется 43 бита и один бит для флага перехода на следующий уровень иерархии. В нашем случае, потребуется 26 - 50 бит. Таким образом, длина ключа получается примерно одинаковая. К достоинствам ordpath можно отнести легкость включения новой вершины между двумя существующими, однако при этом длина ключа удваивается для данного слоя иерархии. К недостаткам ordpath, можно отнести сложность вычисления уровня иерархии. Для этого требуется провести парсинг всего ключа. В отличии от ordpath, предлагаемый в этой статье ключ лишен данной проблемы.

Заключение

В настоящей работе предложен высокоэффективный алгоритм кодирования иерархических стурктур. Данный алгоритм позволяет быстро и эффективно вычислять матереализированный путь до узла, находить пределы для всех узлов поддеревьев и быстро находить уровень иерархии данной вершины. Алгоритм может быть использован в базах данных, а также во всех алгоритмах требующих быстрой фильтрации иерархических данных.У алгоритма отсутствуют ограничения на объем обрабатываемых данных. При значительном увеличении количества данных предложенный метод кодирования ключа приводит к незначительному увеличению временных затрат и ресурсов памяти, что является основным показателем эффективности работы алгоритма. Отсутствие проблем, которые существуют в других популярных программах таких как ltree и ordpath, делает данный подход более эффективным, и дает возможность использования данного метода в высоконагруженных системах при фильтрации информации.

Список литературы

1. O'Neil P., O'Neil E., Pal S., Cseri I., Schaller G., Westbury N. ORDPATHs: Insert-Friendly XML Node Labels / *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, 2004, pp. 903–908 http://www.cse.iitb.ac.in/infolab/Data/Courses/CS632/2014/2007/Papers/ordpath.pdf.
2. PostgreSQL index - ltree URL: https://www.postgresql.org.
3. Guttman A. R-trees: A dynamic index structure for spatial searching / *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, 1984, pp. 47-57.
4. Zabavskyy A. *Closure Table Pattern to Model Hierarchies in NoSQL URL: https://towardsdatascience.com.*
5. *Oracle - Hierarchical Queries URL: https://docs.oracle.com.*
6. Abdelali E., Mazouz S. et al. Different approaches to modeling the trees data in relational database, *International Journal of Computer Applications*, 2012. vol. 53, no. 18.

7. Tropashko V. Nested Intervals Tree Encoding with Continued Fraction, 2004 arXiv preprint cs/0402051.
8. Tropashko V. Nested intervals tree encoding in SQL, *ACM SIGMOD Record*, 2005. vol. 34, no. 2, pp. 47–52.
9. Tropashko V. *Trees in SQL: Nested Sets and Materialized Path*, 2002 URL: www.dbazine.com/tropashko4.shtml.
10. Tropashko V. *Nested Intervals with Farey Fractions*, 2004 URL: arXiv preprint cs.DB/0401014.
11. Celko J. *Joe Celko's SQL for Smarties: Trees and Hierarchies*. San Francisco: Morgan Kaufmann Publishers Inc, 2004. 436 pp.
12. Celko J. *SQL for Smarties: Advanced SQL Programming*. San Francisco: Morgan Kaufmann Publishers Inc, 1999. 576 pp.
13. Celko J. *Thinking in Auxiliary Sets, Temporal, and Virtual Tables in SQL*. Burlington: Morgan Kaufmann Publishers Inc, 2008. 576 pp.
14. Celko J. *SQL for Smarties: Advanced SQL Programming, third edition*. San Francisco: Morgan Kaufmann Publishers Inc, 2005. 576 pp.
15. Celko J. *SQL for Smarties: Advanced SQL Programming, fourth edition*. Burlington: Morgan Kaufmann Publishers Inc, 2011.
16. Leiserson Ch. E., Prokop H., Randall K. H. Using de Bruijn sequences to index a 1 in a computer word, *Available on the Internet from http://supertech.csail.mit.edu/papers.html*, 1998. vol. 3, no. 5 `http://supertech.csail.mit.edu/papers/debruijn.pdf`.
17. Anderson S. *Count the consecutive zero bits (trailing) on the right with modulus division and lookup*, 2005 URL: https://graphics.stanford.edu/ seander/bithacks.html#ZerosOnRightModLookup.
18. Anderson S. *Reverse bits in word by lookup table*, 2005 URL: https://graphics.stanford.edu/ seander/bithacks.html#BitReverseTable.
19. Kernighan W. B., Ritchie D. *The C programming language*. Burlington: Pearson Education Asia, 2002.
20. Beeler M., Gosper R.W., Schroeppel R. H. *MIT Artificial intelligence memo*, vol. 239. Burlington: Feb, 1972.
21. Anderson S. *Interleave bits by table lookup*, 2005 InterleaveTableLookup.

Информация об авторе

Кириллов Владимир Святославович – кандидат физико-математических наук, доцент, научный сотрудник КВНЦ РАН, Россия, ORCID 0009-0007-3996-1844.

References

[1] O'Neil P. et al. ORDPATHs: Insert-friendly XML node labels. Proceedings of the 2004 ACM SIGMOD international conference on Management of data, 2004, 903-908. http://www.cse.iitb.ac.in/infolab/Data/Courses/CS632/2014/2007/Papers/ordpath.pdf

[2] PostgreSQL index - ltree https://www.postgresql.org/docs/current/ltree.html

[3] Guttman A. R-trees: A dynamic index structure for spatial searching http://www.sai.msu.su/~megera/postgres/gist/papers/gutman-rtree.pdf

[4] Zabavskyy A. Closure Table Pattern to Model Hierarchies in NoSQL urlhttps://towardsdatascience.com/closure-table-pattern-to-model-hierarchies-in-nosql-c1be6a87e05b

[5] Oracle-Hierarchical Queries https://docs.oracle.com/database/121/SQLRF/queries003.htm#SQLRF52332

[6] Abdelali E., Mazouz S. et al. Different Approaches to Modeling the Trees Data in Relational Database. International Journal of Computer Applications. 2012, 53, 18.

[7] Tropashko V. Nested Intervals Tree Encoding with Continued Fraction.

[8] Tropashko V. Encoding in SQL, SIGMOD. Nested Intervals Tree. 2005.

[9] Tropashko V. Trees in SQL: Nested Sets and Materialized Path. 2003.

[10] Tropashko V. Nested Intervals with Farey Fractions. 2004.

[11] Celko J. Trees & Hierarchy in SQL for Smarties. Morgan Kaufmann Publishers, San Francisco, 2004, CA 94111.

[12] Celko J. SQL for Smarties: Advanced SQL Programming. Morgan Kaufmann Publishers, San Francisco, 1999, CA, p. 576.

[13] Celko J. Thinking in Auxiliary Sets, Temporal and Virtual Tables in SQL. Morgan Kaufmann Publishers, Burlington, 2008. MA 01803-4255.

[14] Celko J. SQL for Smarties: Advanced SQL Programming third edition. Morgan Kaufmann Publishers, San Francisco, continuation 400, 2005, CA 94111.

[15] Celko J. SQL for Smarties: Advanced SQL Programming fourth edition. Morgan Kaufmann Publishers, Burlington, 2011, MA 01803.

[16] Leiserson Ch.E., Prokop H., Randall K.H. Using de Bruijn Sequences to Index a 1 in a Computer Word http://supertech.csail.mit.edu/papers/debruijn.pdf

[17] Count the consecutive zero bits (trailing) on the right with modulus division and lookup https://graphics.stanford.edu/~seander/bithacks.html#ZerosOnRightModLookup

[18] Reverse bits in word by lookup table https://graphics.stanford.edu/~seander/bithacks.html#BitReverseTable

[19] Kernighan B. W., Ritchie D. M. C Programming Language 2nd Ed.

[20] Beeler M., Gosper R.W., Schroeppel R. H. MIT Artificial intelligence memo, 239, 1972

[21] Interleave bits by table lookup https://graphics.stanford.edu/~seander/bithacks.html#InterleaveTableLookup

Information about author

Kirillov Vladimir Svjatoslavovich – Ph. D. (Phys. & Math.), Researcher, Kabardino-Balkarian Scientific Center of the Russian Academy of Sciences, Russia, ORCID 0009-0007-3996-1844.