Инженерный вестник Дона, №2 (2017)
ivdon.ru/ru/magazine/archive/n2y2017/4166

Представление графовых моделей данных в виде n-арных деревьев

Г.Е. Засядко

Московский политехнический университет

Аннотация: в данной статье предложен и рассмотрен новый способ представления произвольных графовых моделей данных в виде n-арных деревьев во внешней памяти, обеспечивающий выполнение операций помещения, извлечения и поиска элементов данных за логарифмическое время. Такой подход позволяет за наименьшее время осуществлять поиск не только свойств и связей, принадлежащих объектам, но также и самих объектов по значениям их свойств, при этом обеспечивая линейный рост сложности по памяти.
Ключевые слова: модель данных, граф, представление графа, база данных, структура данных, индексация.

В последнее время, в связи с активным развитием таких областей науки как Data Science и Big Data, встает вопрос о поиске новых и эффективных методов хранения и обработки огромных массивов структурированных и неструктурированных данных. [1 — 3]. Для решения определенного круга задач, таких как моделирование социальных графов, исследования в биоинформатике, представление семантической паутины [4], особую популярность получили графовые базы данных [5]. Для задач с естественной графовой структурой данных [6] графовые СУБД могут существенно превосходить реляционные по производительности, а также иметь преимущества в наглядности представления и простоте внесения изменений, благодаря отсутствию жесткой схемы данных [7].

Графовые базы данных строятся на модели данных, представляющей собой граф и его обобщения. Обычно основой модели данных графовых БД является параметризованный гиперграф, в котором как вершины, так и ребра могут иметь различные наборы индивидуальных свойств, а ребра соединять произвольное количество вершин. Соответственно, как и любую логическую модель данных, так же и графовую, необходимо представлять в виде физической модели, пригодной для хранения и обработки.

Существующие способы представления графов, пригодных для компьютерной обработки [8 – 10]: матрица смежности, матрица инцидентности, список смежности, список ребер и их модификации, обладают одним общим существенным недостатком – они не обеспечивают возможности хранения и быстрого поиска значений свойств узлов и ребер графа.

Таким образом, встает задача о поиске нового способа представления графа в виде, пригодном для компьютерной обработки, позволяющем осуществлять хранение и поиск значений свойств вершин и ребер графа, а также прямой доступ к любому элементу структуры, обеспечивая при этом минимальную избыточность данных. [11].

Исходя из постановки задачи, предлагается использовать для представления графа, вместо матриц и списков, сбалансированные n-арные деревья поиска: B-дерево, B*-дерево, B+-дерево, B++-дерево, T-дерево и др., выбор определенного вида будет зависеть от дополнительных требований, предъявляемых на стадии реализации. Применение сбалансированных n-арных деревьев позволит минимизировать избыточность данных и количество операций чтения из внешней памяти, осуществляя вставку, удаление и поиск данных за логарифмическое время. [8, 12 — 14].

Таким образом, требуется разбить граф на некоторые пары вида <Ключ> – <Значение>, которые будут помещены в структуру дерева поиска.

Как известно, произвольный граф G является совокупностью, или, другими словами, множеством, состоящим из непустого множества вершин V и множества ребер E, таким образом, что:

$G = \{V, E\}$, $G = \{V_G = \{v_1, v_2, .., v_i\}, E_G = \{e_1, e_2, .., e_j\}\}$, где v_i и e_j – элементы множеств вершин и ребер соответственно, $i >= 1, j >= 0$.

Исходя из задачи построения логической модели данных на графах, можно сказать, что каждая вершина v_i или ребро e_j фактически представляют

собой множество некоторых свойств *p*, при помощи которых будет описана информация об объектах:

$v_i = \{p_1, p_2, .., p_n\}$, $e_j = \{p_1, p_2, .., p_m\}$, где p_n, p_m – элементы множеств свойств вершины v_i и ребра e_j соответственно, $n >= 0$, $m >= 0$.

В свою очередь, каждое свойство *p также* является множеством, состоящим из значений *val* данного свойства:

$p_n = \{val_1, val_2, .., val_k\}$, где val_k – значение свойства p_n, $k > 0$.

Стоит заметить, что сами значения свойств *val* вполне могут представлять собой не только значения простых типов, но и некоторые коллекции значений.

Таким образом, граф представляет собой объект, состоящий из наборов рекурсивно вложенных множеств.

Подобные структуры данных легко описываются при помощи нотации JavaScript Object Notation (JSON). [15] Именно эта нотация и ее термины будут применяться далее для записи структур данных в наглядной и удобной для восприятия форме.

Общий вид графа в нотации JSON:

```
{
    "G": {
        "V": [
            {
                "p1": [
                    "val1",
                    ...,
                    "valk"
                ],
                ...,
                "pn": [
                    ...
                ]
            },
```

```
            ...,
            {
                ...
            }
        ],
        "E": [
            {
                ...
            },
            ...,
            {
                ...
            }
        ]
    }
}
```

В общем случае такая структура служит источником данных, но сейчас данные в ней еще не пригодны для помещения в дерево поиска. Связано это с тем, что в пределах всей структуры могут встречаться одинаковые ключи коллекций, это противоречит условию однозначной идентификации элементов внутри структуры.

Для решения поставленной задачи и сохранения при этом единообразия описания всех коллекций и их элементов необходимо ввести два дополнительных понятия: уникальный идентификатор элемента и ассоциированное с элементом значение.

Уникальный идентификатор — уникальная в пределах всей структуры последовательность байт, соответствующая конкретному элементу коллекции, и позволяющая однозначно определять его среди других, в зависимости от варианта реализации может быть представлена в виде числа или строки, выступает в качестве ключа элемента, не образуется на основе каких-либо данных структуры, а генерируется искусственно.

Ассоциированное значение — обязательный элемент каждой коллекции, который представлен единственным значением одного из простых типов, таких как число, строка, булево или *null* (по умолчанию), и доступный по ключу, однозначно соответствующему уникальному идентификатору коллекции.

Чтобы преобразовать исходное описание графа к виду пригодному для последующего помещения в дерево поиска необходимо рекурсивно обойти все элементы структуры и:

1. каждый элемент исходной структуры, являющийся массивом или значением простого типа, преобразовать в объект;
2. для каждого объекта новой структуры установить в качестве ключа уникальный идентификатор, сгенерированный по некоторому заранее определенному правилу. В качестве уникального идентификатора можно использовать, например, сквозное для всей структуры автоинкрементное значение, или значение типа Universally Unique Identifier (UUID) [16];
3. в состав каждого объекта новой структуры добавить элемент, представляющий собой ассоциированное с объектом значение. Ключ нового элемента состоит из уникального идентификатора объекта и флага, показывающего, что это ключ ассоциированного значения. А значением нового элемента будет:
   - значение ключа коллекции, которой являлся объект в исходной структуре, для элементов массивов ключ считать равным *null*;
   - иначе, значение элемента простого типа, которым являлся объект в исходной структуре.

   Преобразованный общий вид графа в нотации JSON:

```
{
    "uidnull":{
        "uidnull*":null,
```

```
"uid0": {
    "uid0*": "G",
    "uid1": {
        "uid1*": "V",
        "uid2": {
            "uid2*": null,
            "uid3": {
                "uid3*": "p1",
                "uid4": {
                    "uid4*": null,
                    "uid5": {
                        "uid5*": "val1"
                    }
                },
                ...,
                "uid6": {
                    "uid6*": null,
                    "uid7": {
                        "uid7*": "valk"
                    }
                }
            },
            ...,
            "uid8": {
                "uid8*": "pn",
                ...,
            }
        },
        ...,
        "uid9": {
            "uid9*": null,
            ...,
        }
    },
    "uid10": {
        "uid10*": "E",
```

```
                "uid11": {
                    "uid11*": null,
                    ...,
                },
                ...,
                "uid12": {
                    "uid12*": null,
                    ...,
                }
            }
        }
    }
}
```

Из приведенной записи видно, что после преобразования каждому элементу структуры в качестве ключа был присвоен некоторый сгенерированный уникальный идентификатор *uid*. Если в исходной структуре элемент являлся массивом или значением простого типа, то он был преобразован в объект, а исходное значение ключа элемента теперь доступно в качестве ассоциированного значения элемента по ключу *uid**. Так же в качестве единого корневого элемента для всей структуры добавлен элемент с пустым идентификатором uid_{null}, он будет выполнять вспомогательную роль для нахождения всех элементов верхнего уровня.

Теперь для заполнения дерева поиска необходимо рекурсивно обойти всю структуру и поместить в дерево поиска набор пар <Ключ> - <Значение>, где:

- для каждой коллекции и каждого элемента, являющегося непосредственным потомком этой коллекции, в качестве ключа узла дерева будет выступать уникальный идентификатор коллекции, а в качестве значения узла дерева — уникальный идентификатор элемента;
- для каждого элемента, представляющего собой значение простого типа, в качестве ключа узла дерева будет выступать уникальный

идентификатор коллекции, а в качестве значения узла дерева — непосредственное значение элемента структуры.

Такой набор даст возможность получать всю информацию о любом объекте по его уникальному идентификатору. Помимо этого, для реализации поиска по значениям свойств объектов требуется также поместить в дерево инвертированные варианты вышеописанных пар, добавляя к ключу специальный флаг, обозначающий обратный порядок. Таким образом, все элементы исходной структуры данных графа будут помещены в дерево поиска, то есть проиндексированы, и готовы для последующей обработки.

Графовая модель данных по своей сути является объектной моделью, поэтому можно сделать вывод о том, что описанный способ подходит также и для представления произвольных объектных моделей данных во внешней памяти.

Данный способ обеспечивает атомарность хранения данных и выполнение операций вставки, изменения, удаления и поиска данных за логарифмическое время. Способ может быть применен при разработке различных прикладных решений, систем управления базами данных, хранилищ данных, программного обеспечения для распределенных систем и "облачной" инфраструктуры.

Литература

1. Min Chen, Shiwen Mao, Yin Zhang, Victor C.M. Leung. Big Data. Related Technologies, Challenges, and Future Prospects. Springer, 2014. 89 p.
2. EMC Education Services. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. Wiley, 2015. 432 p.
3. Jaiteg Singh, Saravjeet Singh. Using Big Data for business perspectives. LAP Lambert Academic Publishing, 2014.108 p.
4. Todd Hoff. Paper: Graph Databases and the Future of Large-Scale Knowledge Management // High Scalability URL:

highscalability.com/blog/2009/6/10/paper-graph-databases-and-the-future-of-large-scale-knowledg.html (Accessed: 22.04.2017).

5. Renzo Angles, Claudio Gutierrez. Survey of Graph Database Models // DCC. Docencia del Departamento de Ciencias de la Computacion. Universidad de Chile. URL: users.dcc.uchile.cl/~cgutierr/papers/surveyGDB.pdf (Accessed: 22.04.2017).

6. С.В. Астанин, Н.В. Драгныш, Н.К. Жуковская. Вложенные метаграфы как модели сложных объектов // Инженерный вестник Дона, 2012, №4. URL: ivdon.ru/ru/magazine/archive/n4p2y2012/1434.

7. Ian Robinson, Jim Webber, Emil Eifrem. Graph Databases 1st edition. O'Reilly, 2013. 224 p.

8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Introduction to Algorithms, Third Edition. The MIT Press, 2010. 1313 p.

9. Niklaus Wirth. Algorithms and Data Structures. Prentice-Hall, Inc, 1986. 366 p.

10. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы. Вильямс, 2016. 400 с.

11. Г.Е. Засядко, А.В. Карпов. Проблемы разработки графовых баз данных // Инженерный вестник Дона, 2017, №1. URL: ivdon.ru/ru/magazine/archive/n1y2017/3994.

12. Rudolf Bayer, Binary B-Trees for Virtual Memory, Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, 1971, pp. 219–235.

13. Дональд Э. Кнут. Искусство программирования. Том 3. Сортировка и поиск. Вильямс, 2014. 824 с.

14. Г.Е. Засядко. Применение префиксных деревьев для индексации информации в таблицах баз данных // Центр перспективных научных публикаций, II международная научно-практическая конференция

"Перспективы развития науки и образования". URL: co-nf.ru/wp-content/uploads/2016/03/Sbornik_29.02.2016.pdf.

15. Standard ECMA-404. The JSON Data Interchange Format. 1st Edition. October 2013. // Ecma International. URL: ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (Accessed 22.04.2017).

16. RFC 4122. A Universally Unique IDentifier (UUID) URN Namespace. July 2005. // The Internet Engineering Task Force (IETF®). URL: tools.ietf.org/html/rfc4122 (Accessed 22.04.2017).

References

1. Min Chen, Shiwen Mao, Yin Zhang, Victor C.M. Leung. Big Data. Related Technologies, Challenges, and Future Prospects. Springer, 2014. 89 p.

2. EMC Education Services. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. Wiley, 2015. 432 p.

3. Jaiteg Singh, Saravjeet Singh. Using Big Data for business perspectives. LAP Lambert Academic Publishing, 2014. 108 p.

4. Todd Hoff. Paper: Graph Databases And The Future Of Large-Scale Knowledge Management. High Scalability URL: highscalability.com/blog/2009/6/10/paper-graph-databases-and-the-future-of-large-scale-knowledg.html (Accessed 22.04.2017).

5. Renzo Angles, Claudio Gutierrez. Survey of Graph Database Models. DCC. Docencia del Departamento de Ciencias de la Computacion. Universidad de Chile. URL: users.dcc.uchile.cl/~cgutierr/papers/surveyGDB.pdf (Accessed 22.04.2017).

6. S.V. Astanin, N.V. Dragnysh, N.K. Zhukovskaja. Inzhenernyj vestnik Dona (Rus), 2012, №4. URL: ivdon.ru/ru/magazine/archive/n4p2y2012/1434.

7. Ian Robinson, Jim Webber, Emil Eifrem. Graph Databases 1st edition. O'Reilly, 2013. 224 p.

8. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein Introduction to Algorithms, Third Edition. The MIT Press, 2010. 1313 p.

9. Niklaus Wirth. Algorithms and Data Structures. Prentice-Hall, Inc, 1986. 366 p.

10. Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. Struktury dannyh i algoritmy [Data structures and algorithms]. Williams, 2016. 400 p.

11. G.E. Zasyadko, A.V. Karpov. Inzhenernyj vestnik Dona (Rus), 2017, №1. URL: ivdon.ru/ru/magazine/archive/n1y2017/3994.

12. Rudolf Bayer, Binary B-Trees for Virtual Memory, Proceedings of 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, 1971, pp. 219–235.

13. Donald E. Knut. Iskusstvo programmirovanija. Tom 3. Sortirovka i poisk [The Art of Computer Programming. Volume 3. Sorting and searching]. Williams, 2014. 824 p.

14. G.E. Zasyadko. Primenenie prefiksnyh derev'ev dlja indeksacii informacii v tablicah baz dannyh. Centr perspektivnyh nauchnyh publikacij, II mezhdunarodnaja nauchno-prakticheskaja konferencija "Perspektivy razvitija nauki i obrazovanija". URL: co-nf.ru/wp-content/uploads/2016/03/Sbornik_29.02.2016.pdf.

15. Standard ECMA-404. The JSON Data Interchange Format. 1st Edition. October 2013. Ecma International. URL: ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (Accessed 22.04.2017).

16. RFC 4122. A Universally Unique IDentifier (UUID) URN Namespace. July 2005. The Internet Engineering Task Force (IETF®). URL: tools.ietf.org/html/rfc4122 (Accessed 22.04.2017).