Студенников М.Р. Реализация версионности с помощью древовидно графовой структуры данных // Международный журнал информационных технологий и энергоэффективности. – 2024. – Т. 9 № 7(45) с. 84–96

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.4

РЕАЛИЗАЦИЯ ВЕРСИОННОСТИ С ПОМОЩЬЮ ДРЕВОВИДНО ГРАФОВОЙ СТРУКТУРЫ ДАННЫХ

Студенников М.Р.
ФГБОУ ВО «МИРЭА – РОССИЙСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ», Москва, Россия (119454, г. Москва, просп. Вернадского, 78, стр. 4), e-mail: mr.mixan01@mail.ru

В рамках проекта по созданию многопользовательской системы управления литературным контентом была поставлена задача по реализации/имитации версионности литературных произведений. В данной работе описан один из возможных подходов к хранению информации в системе, позволяющий решить поставленную задачу. Также в данной работе дано описание и теоретическая оценка временной сложности основных алгоритмов обработки данных при использовании представленного подхода. Представленные алгоритмы обработки данных для описанного подхода к хранению учитывают специфику проекта, но могут быть легко адаптированы и оптимизированы под другие предметные области.

Ключевые слова: Граф, деревья, версионность, структуры данных, алгоритмы обработки структуры данных, сложность алгоритмов.

IMPLEMENTATION OF VERSIONING USING A TREE-GRAPH DATA STRUCTURE

Studennikov M.R.
"MIREA - RUSSIAN TECHNOLOGICAL UNIVERSITY", Moscow, Russia (119454, Moscow, Vernadsky Avenue, 78, building 4), e-mail: mr.mixan01@mail.ru

As part of the project to create a multi-user literary content management system, the task was set to implement/simulate the versioning of literary works. This paper describes one of the possible approaches to storing information in the system, which allows solving the task. Also in this paper, a description and theoretical assessment of the time complexity of the main data processing algorithms using the presented approach are given. The presented data processing algorithms for the described storage approach take into account the specifics of the project, but can be easily adapted and optimized for other subject areas.

Keywords: Graph, trees, versioning, data structures, algorithms for processing data structures, complexity of algorithms..

В рамках проекта по созданию многопользовательской системы управления литературным контентом была поставлена задача по реализации/имитации версионности литературных произведений.

Описание задачи

В системе определено три сущности: Страница, Ветка и Блок.

Страницы организуются в древовидную/иерархичную структуру. Каждая Страница представляет собой дерево Веток. Ветки наследуются друг от друга в рамках Страницы. Каждая Ветка состоит из массива Блоков. Блоки внутри Ветки располагаются в определенной последовательности.

Страницам и Веткам присваиваются имена. Блоки хранят в себе некоторую полезную информацию, контент, к примеру текстовое описание чего-либо.

Представление Страницы в виде дерева Веток реализует версионность контента в пределах Страницы.

На Рисунке 1 схематично изображены взаимосвязи между описанными сущностями в нотации UML[1].

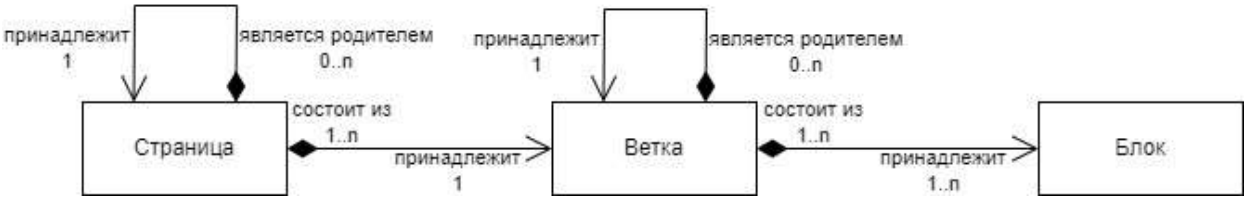

Рисунок 1 – UML-диаграмма взаимосвязи сущностей

Задача

1. Спроектировать и описать минимально необходимую структуру таблиц в реляционной базе данных, в которой могут храниться описанные сущности.
2. Описать алгоритмы выше указанных операций, с учетом того, что они не должны нарушать целостность контента Страницы и принадлежность Блоков их Веткам.
3. Провести теоретическую оценку временной сложности предложенных алгоритмов обработки данных в нотации О-большое[2-4].

Условия

Предложенный подход к хранению должен поддерживать следующие операции:

- чтение Ветки – выдача массива Блоков, относящихся к запрашиваемой Ветке;
- слияние двух родственных Веток;
- создание/вставка Блока на Ветке;
- удаление Блока;
- изменение Блока – изменение контента Блока, перемещение Блока в пределах Ветки.

Предложенный подход к хранению должен минимизировать количество хранимой в БД информации.

Изменение контента Блоков на одной Ветке должно приводить к изменению контента этих Блоков на всех дочерних Ветках, в которых они присутствуют.

Решение

В качестве решения поставленной задачи было предложено хранение Блоков в виде неполного невзвешенного ориентированного ациклического графа[5]. На Рисунке 2 представлена ER-диаграмма БД с минимально необходимыми полями сущностей для реализации такого подхода, а на Рисунке 3 представлен схематичный пример такого хранения[5].

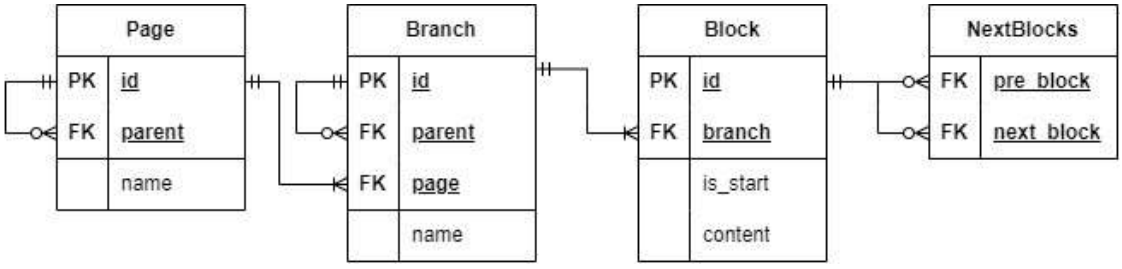

Рисунок 2 – ER-диаграмма, описывающая подход к хранению данных в БД

Атрибут is_start сущности Block хранит логическую переменную, которая указывает является ли Блок началом Линии или нет.

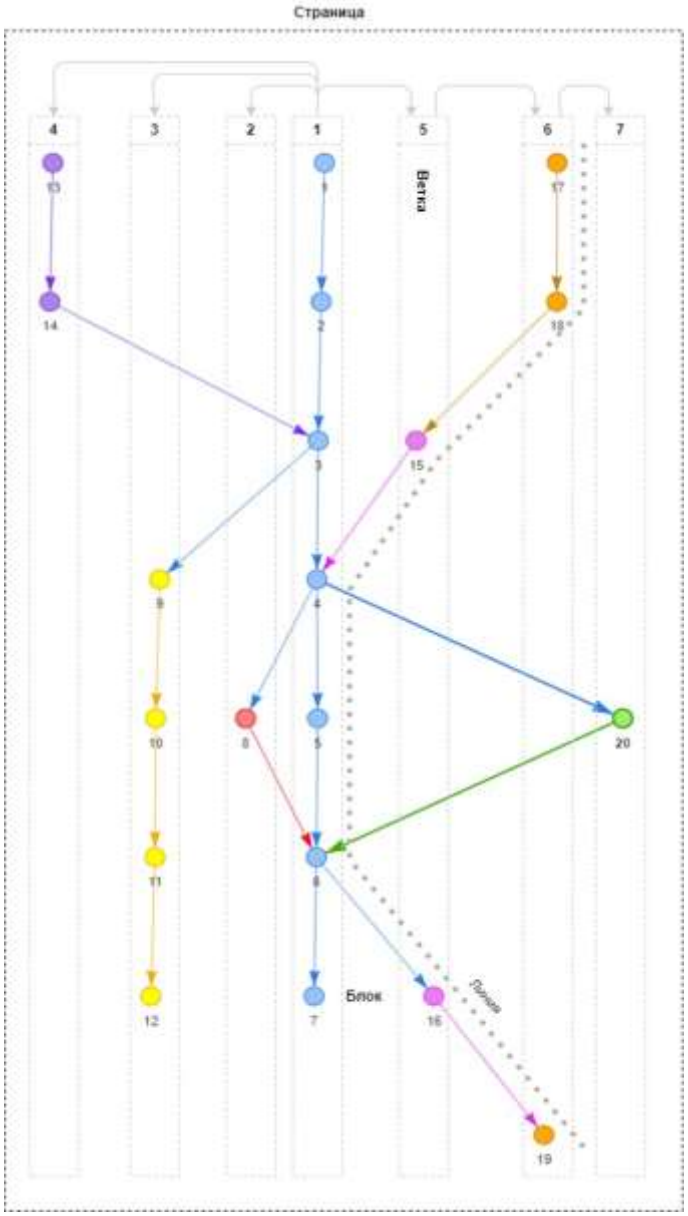

Рисунок 3 – Пример хранения Блоков в виде графа

Такой подход позволяет для каждой дочерней Ветки хранить в основном те Блоки, которые отличаются от родительской.

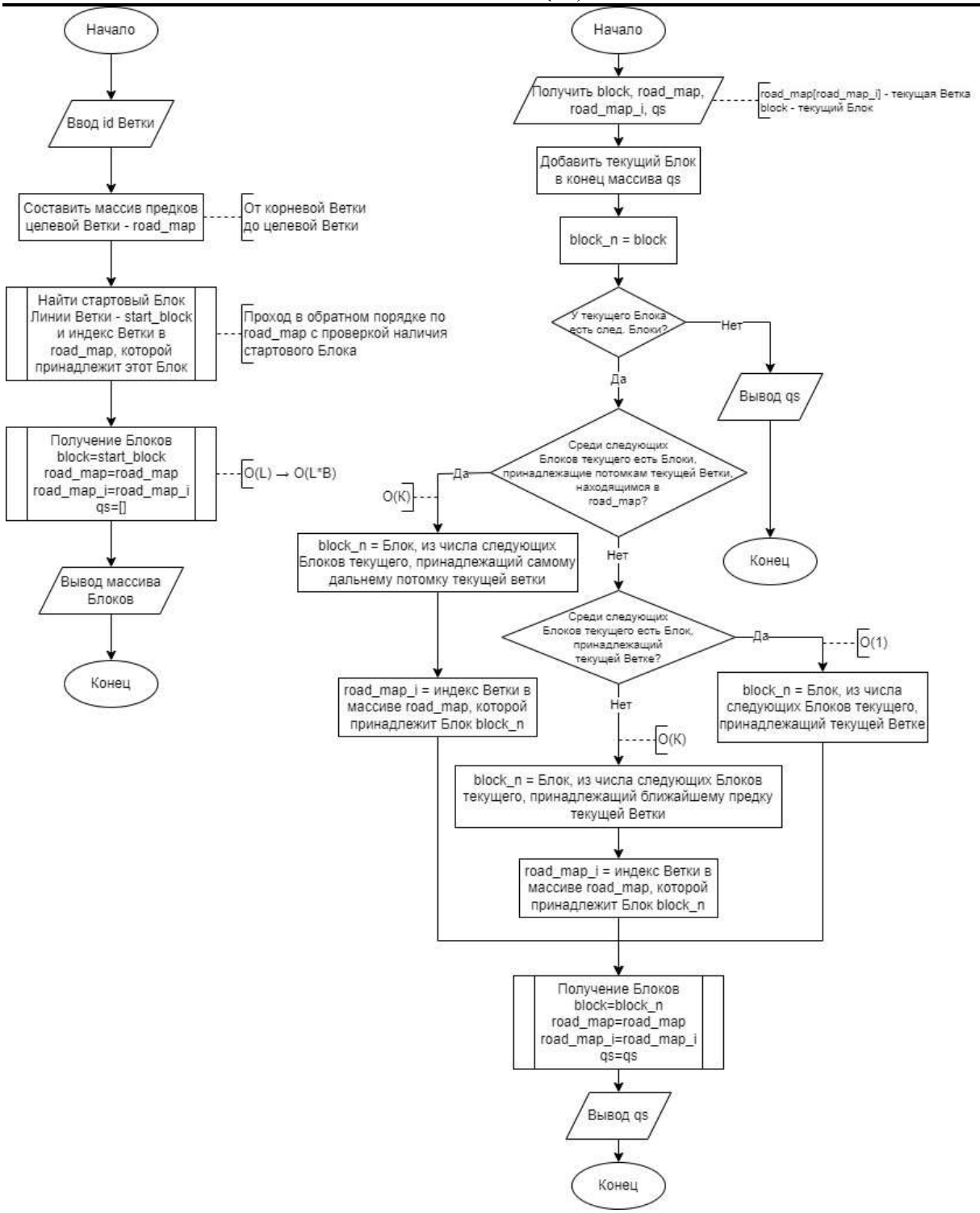

Рисунок 4 – Блок-схема алгоритма чтения Ветки

Немаловажным понятием в рамках данной реализации является понятие Линии. Линия – это массив Блоков, построенный в соответствии со связями в графе Блоков Страницы и иерархией Веток Страницы. Линия является результатом чтения Ветки.

Такой подход также позволяет, внося изменения в одну Линию (версию), изменять сразу несколько других Линий, к примеру изменение контента в Блоке 3 Ветки 1 будет сразу же отображаться во всех Линиях, в которые входит Блок 3 (Ветки 1-4) (Рисунок 3).

Алгоритм чтения Ветки создает массив Блоков Линии Ветки, рекурсивно проходя по связям между Блоками в соответствии с составленной картой Веток. В случае «развилки», то есть когда у текущего Блока есть несколько следующих Блоков с разных Ветках, следующий Блок выбирается согласно «карте». Подробнее алгоритм чтения Ветки описан в блок-схемах Рисунка 4.

Временная сложность алгоритма чтения Ветки при оценке сверху в наихудшем случае (рисунок 5) может быть рассчитана по формуле 1:

$$O(K * L) = O(B^2) = O(n^2), \quad (1)$$

где *K* – поколение Ветки;

L – количество Блоков в Линии;

B – общее количество Блоков Страницы.

Наихудшим случаем для алгоритма чтения Ветки является ситуация, когда поколение Ветки равно общему количеству Блоков на Странице, то есть каждая Ветка состоит всего из одного Блока:

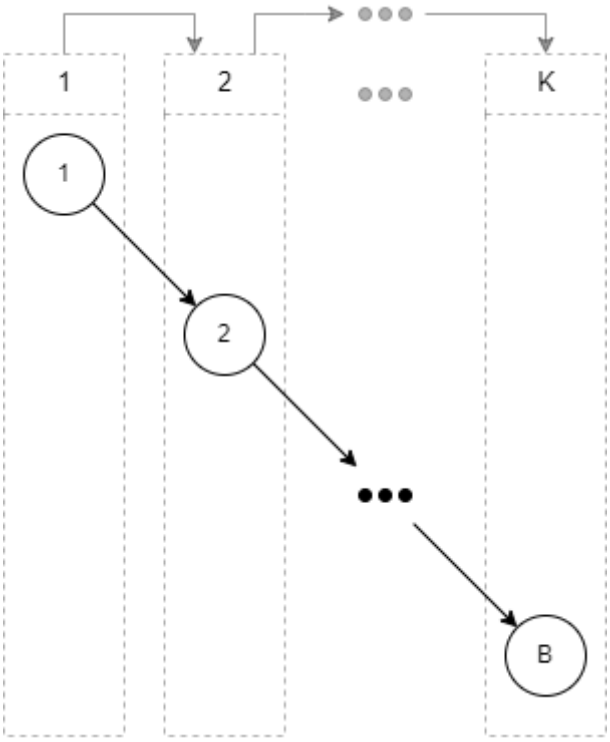

Рисунок 5 – Наихудший случай для алгоритма чтения Ветки

Основная временная сложность алгоритма создания Блока заключается во вставке Блока в массив (список) по индексу. Следовательно, сложность алгоритма создания Блока на Ветке (вставка Блока в Линию (Рисунок 6)) линейна, однако для работы алгоритма необходимо сначала получить Линию Ветки[6]. Следовательно, сложность алгоритма с учетом предварительного чтения Ветки возрастает до квадратичной.

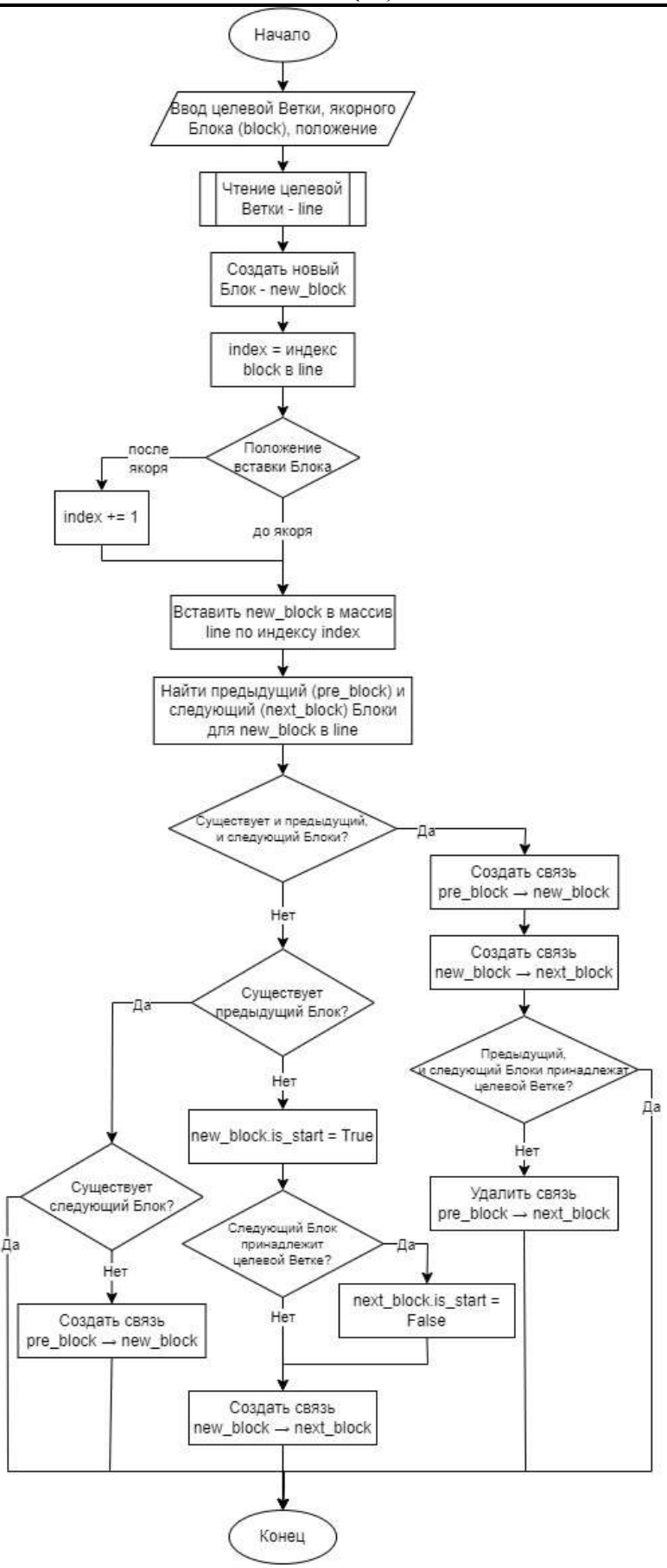

Рисунок 6 – Блок-схема алгоритма создания Блока на Ветке

К обновлению Блока относится как обновление контента в самом Блоке, так и перемещение Блока по Линии Ветки.

Если обновляется контент Блока для Ветки, которой он принадлежит изначально, то обновление сводится к простому обновлению одной записи в базе данных. Сложность такой операции константна[7]. Если же обновление контента Блока происходит для Ветки, которой он не принадлежит, но в чьей Линии находится, то для Ветки создается новый Блок и соответствующие связи (Рисунок 9). Для такого случая существует два наихудших взаимоисключающих сценария расположения Блоков в графе – Рисунки 7 и 8. Оба эти сценария имеют квадратичную сложность.

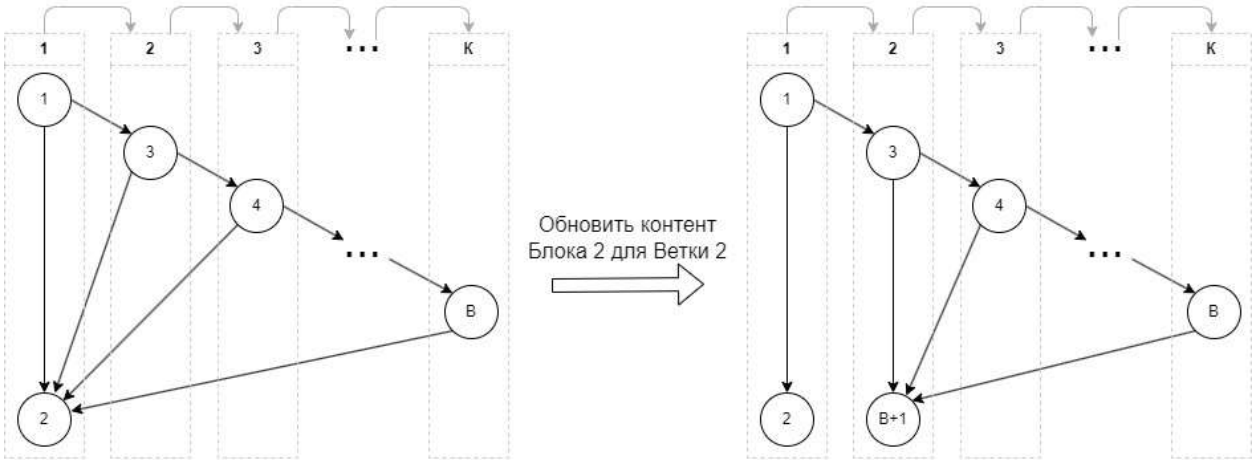

Рисунок 7 – Наихудший случай 1 для операции обновления контента Блока

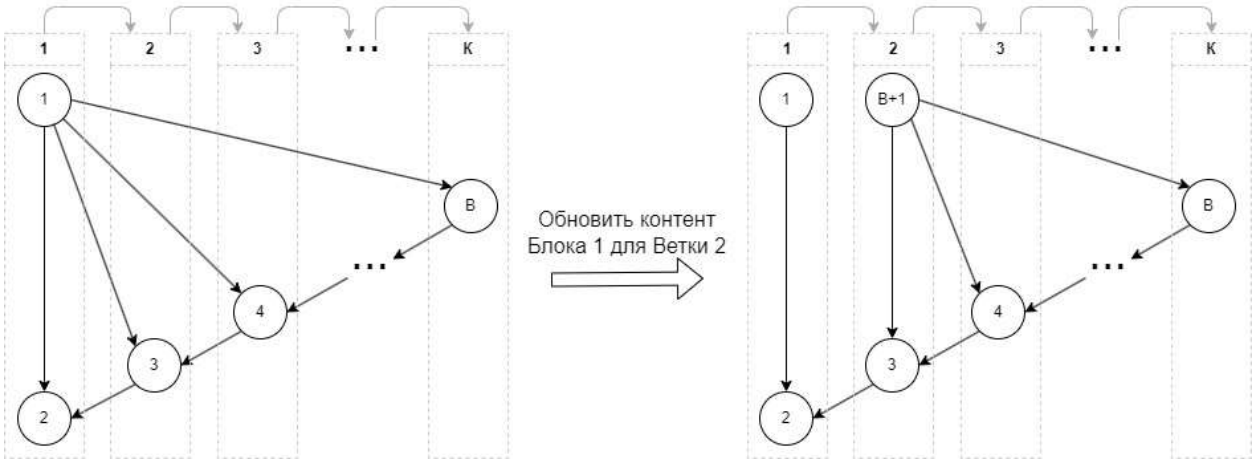

Рисунок 8 – Наихудший случай 2 для операции обновления контента Блока

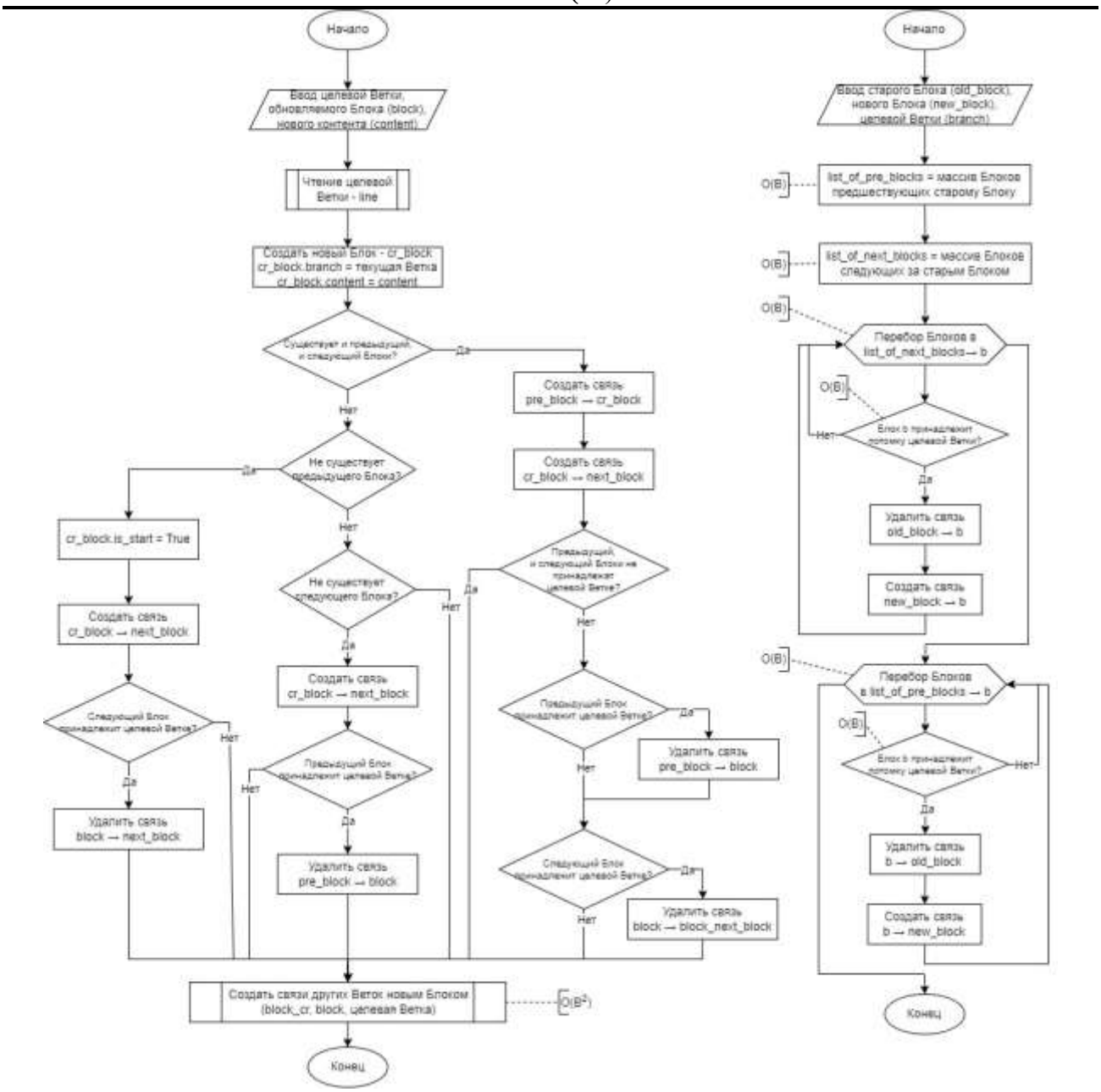

Рисунок 9 – Блок-схемы алгоритма Обновления контента Блока, не принадлежащего целевой Ветке

Операция перемещения Блока может быть реализована совмещением алгоритмов удаления (вырезания) Блока и создания (вставки) Блока на новое место. Временная сложность такого алгоритма будет кубической из-за повышенной сложности операции вырезания.

На Рисунках 10-11 представлены блок-схемы алгоритмов удаления и вырезания Блока. В данной реализации удаление Блока может приводить, как к уменьшению, так и к увеличению общего числа Блоков.

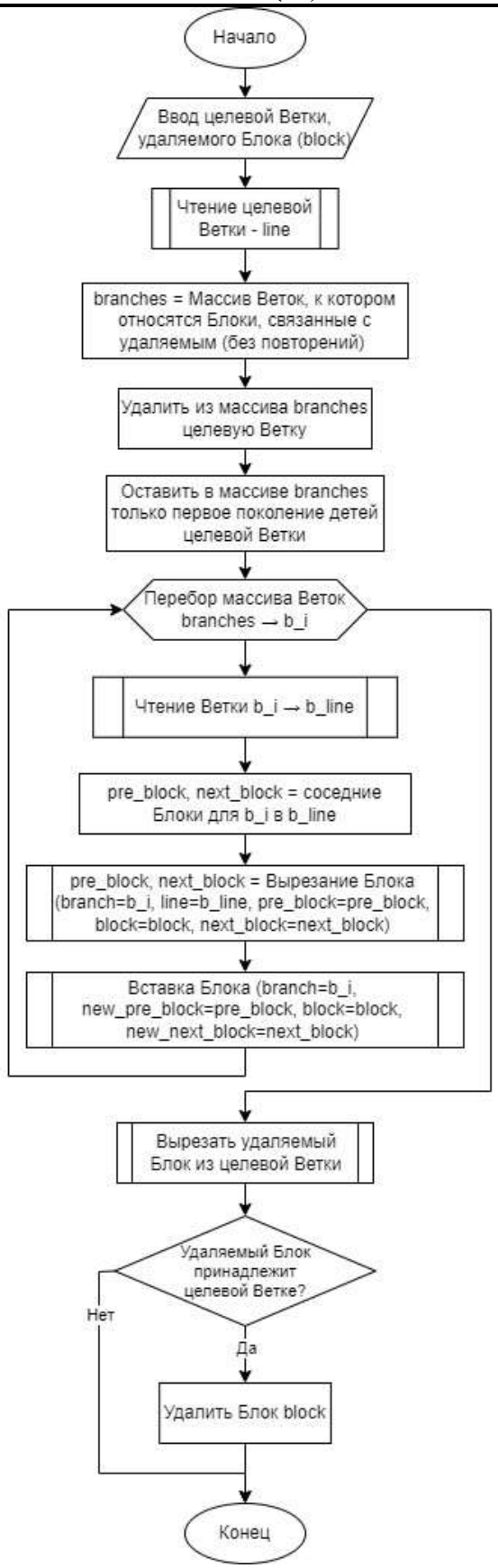

Рисунок 10 – Блок-схема алгоритма удаления Блока

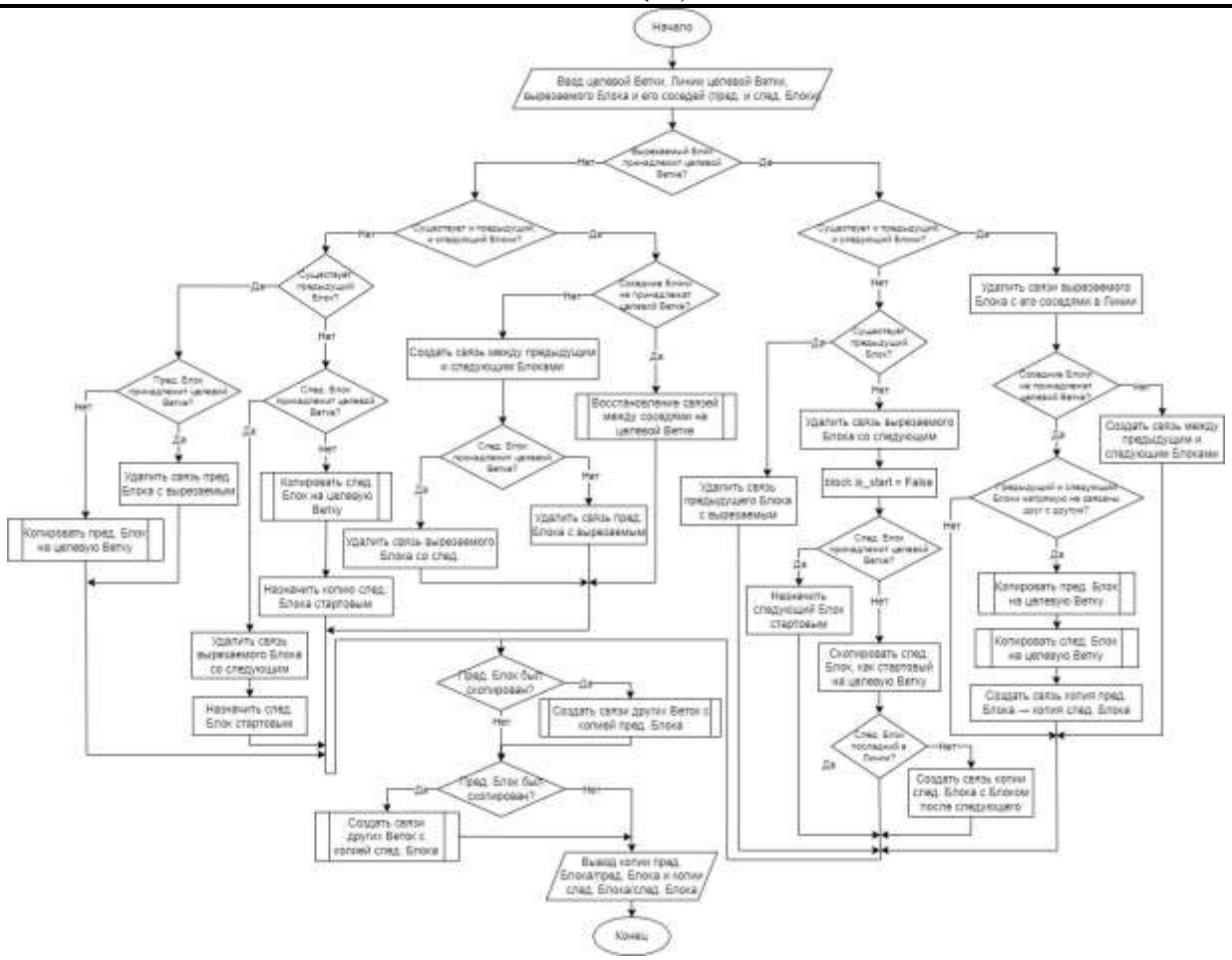

Рисунок 11 – Блок-схема алгоритма вырезания Блока

Одним из крайних наихудших случаев для алгоритма удаления Блока является случай, когда все Блоки для удаляемого являются предшествующими и имеют с ним прямую связь. При этом все связанные с удаляемым Блоки находятся на разных Ветках, которые расположены на одном уровне дерева Ветвей (Рисунок 12). Для такого случая временная сложность составляет $O(n^3)$.

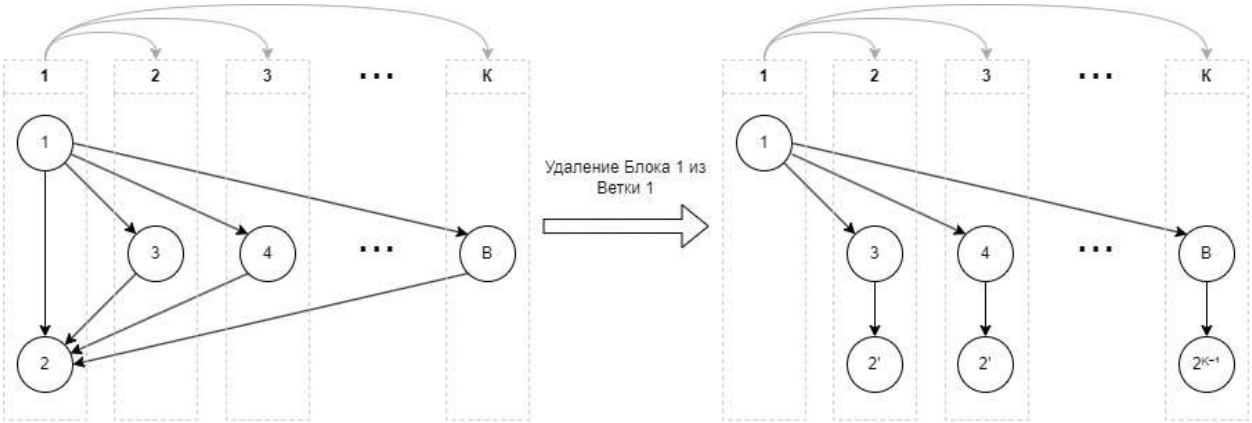

Рисунок 12 – Наихудший случай для алгоритма удаления Блока

Операция слияния (merge) (Рисунок 13) Веток происходит между двумя Ветками: первая (целевая), в которую происходит вливание новых Блоков, вторая (сливаемая) – Ветка-потомок первой чьи Блоки встраиваются в целевую. Целевая Ветка всегда должна быть предком

сливаемой Ветки, при этом если целевая и сливаемая Ветки разделены несколькими поколениями, то все промежуточные Ветки также вливаются в целевую.

Наихудший случай для алгоритма слияния веток совпадает с наихудшим случаем для алгоритма чтения Ветки при условии слияния Ветки 1 с Веткой К. Сложность алгоритма в таком случае является кубической.

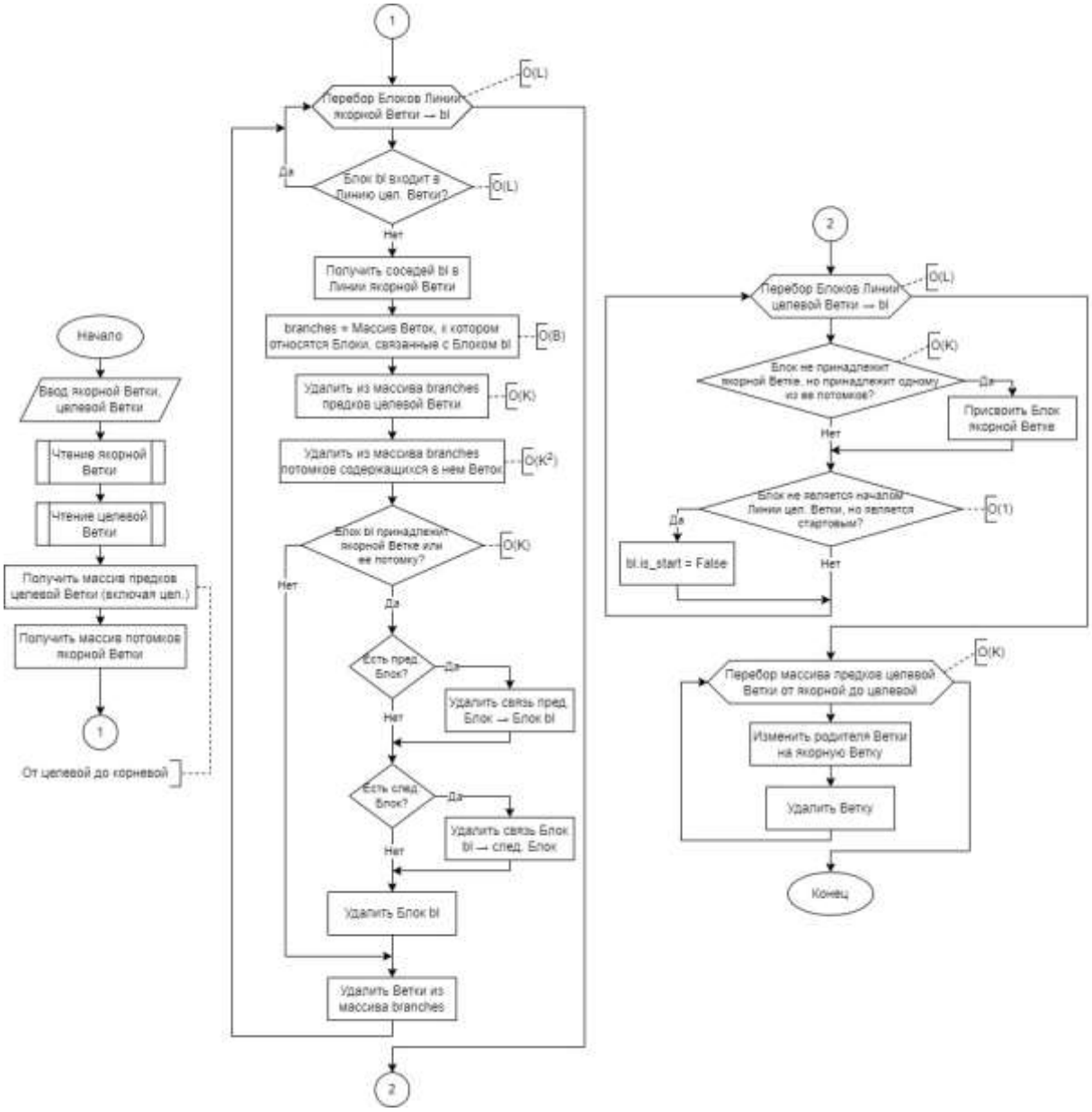

Рисунок 13 – Блок-схема алгоритма Слияния Веток

Заключение

В данной работе предложен один из возможных способов решения задачи реализации версионности в РСУБД с использованием трех взаимосвязанных сущностей и древовидно-графовой структуры хранения данных.

С учетом ограничений, введенных условиями задачи и контекста проекта в рамках, которого была поставлена данная задача, описанный подход является наиболее эффективным с точки зрения объема хранимой информации и минимизации дублирования данных, однако временная сложность алгоритмов обработки данных при этом подходе является достаточно

высокой, что может затруднить его использование в масштабных системах с высокими требованиями к быстродействию.

Использование базы данных не является необходимым условием для использования предложенного подхода. Он может быть изменен и адаптирован под другие технологии. Описанные алгоритмы обработки не являются единственно возможными вариантами реализации и могут быть концептуально изменены с учетом контекста проекта, в котором будет применяться данный подход. К примеру, возможен полный отказ от дублирования Блоков в операциях удаления и изменения контента.

Список литературы

1. Язык UML: что это такое и зачем он нужен. // Skillbox Media [Электронный ресурс] URL: https://skillbox.ru/media/code/yazyk-uml-chto-eto-takoe-i-zachem-on-nuzhen/?ysclid=lw2aqjufip920033510 (дата обращения 11.02.24).
2. Big O // Хабр [Электронный ресурс] URL: https://habr.com/ru/articles/444594/ (дата обращения 11.02.24).
3. Big O и временная сложность // Your to do [Электронный ресурс] URL: https://www.yourtodo.ru/posts/big-o-i-vremennaya-slozhnost/ (дата обращения 11.02.24).
4. 1.3 Анализ алгоритмов; время работы в лучшем, худшем случаях и в среднем // StudFiles [Электронный ресурс] URL: https://studfile.net/preview/915308/page:3/ (дата обращения 11.02.24).
5. ГРАФ И ЕГО ВИДЫ [Электронный ресурс] URL: https://informatikagmk.ucoz.ru/matem_praktika/grafy_teorija.pdf (дата обращения 11.02.24).
6. Что такое ER-диаграмма и как ее создать? // Lucidchart [Электронный ресурс] URL: https://www.lucidchart.com/pages/ru/erd-%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0 (дата обращения 11.02.24).
7. Какова сложность операции добавления элемента в список? // PYTHONIST [Электронный ресурс] URL: https://pythonist.ru/kakova-slozhnost-operaczii-dobavleniya-elementa-v-spisok/?ysclid=lw2b0x3kuv934372990 (дата обращения 11.02.24).
8. Запросы SQL для обновления данных (UPDATE) // SPACE BASE [Электронный ресурс] URL: https://space-base.ru/library/sql/zaprosyi-sql-dlya-obnovleniya-dannyix-update (дата обращения 11.02.24).

References

1. UML language: what is it and why is it needed. // Skillbox Media [Electronic resource] URL: https://skillbox.ru/media/code/yazyk-uml-chto-eto-takoe-i-zachem-on-nuzhen /?ysclid=lw2aqjufip920033510 (accessed 11.02.24).
2. Big O // Habr [Electronic resource] URL: https://habr.com/ru/articles/444594 / (accessed 11.02.24).
3. Big O and time complexity // Your to do [Electronic resource] URL: https://www.yourtodo.ru/posts/big-o-i-vremennaya-slozhnost / (accessed 11.02.24).
4. 1.3 Analysis of algorithms; working time in the best, worst cases and on average // StudFiles [Electronic resource] URL: https://studfile.net/preview/915308/page:3 / (accessed 11.02.24).

5. GRAPH AND ITS TYPES [Electronic resource] URL: https://informatikagmk.ucoz.ru/matem_praktika/grafy_teorija.pdf (accessed 11.02.24).
6. What is an ER diagram and how do I create it? // Lucidchart [Electronic resource] URL: https://www.lucidchart.com/pages/ru/erd-%D0%B4%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0 (accessed 11.02.24).
7. What is the complexity of the operation of adding an item to the list? // PYTHONIST [Electronic resource] URL: https://pythonist.ru/kakova-slozhnost-operaczii-dobavleniya-elementa-v-spisok /?ysclid=lw2b0x3kuv934372990 (accessed 11.02.24).
8. SQL queries for data update // SPACE BASE [Electronic resource] URL: https://space-base.ru/library/sql/zaprosyi-sql-dlya-obnovleniya-dannyix-update (accessed 11.02.24).