Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА ИУ-7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЁТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Формализация алгоритма обновления гипертекстового документа Fiber и анализ эффективности его использования в современных web-приложениях»

Студент группы **ИУ7-76Б** _____ **П. А. Калашков**

(Подпись, дата) (И.О. Фамилия)

Руководитель _____ **Д. Е. Бекасов**

(Подпись, дата) (И.О. Фамилия)

***2023** г.*

РЕФЕРАТ

Расчётно-пояснительная записка содержит 53 с., 13 рис., 36 ист.

ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА, ВИРТУАЛЬНАЯ ОБЪЕКТНАЯ МОДЕЛЬ ДОКУМЕНТА, ОБНОВЛЕНИЕ ГИПЕРТЕКСТОВОГО ДОКУМЕНТА, DOM, VDOM, АЛГОРИТМ СОГЛАСОВАНИЯ, FIBER

Целью работы: анализ существующих методов обновления гипертекстовых документов, формализация алгоритма обновления гипертекстового документа Fiber, его анализ и сравнение его эффективности по сравнению с другими существующими алгоритмами.

В данной работе проводится изучение принципов работы объектной модели документа (DOM), виртуальной объектной модели документа (VDOM), а также сравнение и анализ трудёмкостей алгоритмов обновления документа с использованием объектной модели документа и виртуальной объектной модели документа, а также алгоритма Fiber.

Результаты: из перечисленных алгоритмов именно алгоритм Fiber позволяет обеспечивать плавность обновления изображения на экране и при большом размере DOM-дерева, и при большом количестве операций обновления, в то время как алгоритм обновления с использованием алгоритма согласования выполняется от начала до конца и при большом количестве операций обновления может приводить к уменьшению количества кадров в секунду.

СОДЕРЖАНИЕ

НОРМАТИВНЫЕ ССЫЛКИ

В настоящем отчете о НИР использованы ссылки на следующие стандарты:

1) DOM Living standart [1];
2) HTML Living standart [2];
3) DOM Level 3 Core Specification [3].

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

HTML — HyperText Markup Language — язык гипертекстовой разметки.

CSS — Cascading Style Sheets — каскадные таблицы стилей.

HTTP — HyperText Transfer Protocol — протол передачи гипертекстовых документов.

DOM — Document Object Model — объектная модель документа.

API — Application Programming Interface — программный интерфейс приложения.

UI — User Interface — пользовательский интерфейс.

SSR — Server Side Rendering — рендеринг на стороне сервера.

ВВЕДЕНИЕ

Работа с гипертекстовыми документами является неотъемлемой частью жизни каждого человека, пользующегося Всемирной сетью, и часто появляется потребность в просмотре различных гипертекстовых документов и в выполнении операций, приводящих к их изменению [4]. Существует необходимость решить, каким образом стоит производить операции обновления гипертекстовых документов.

Для взаимодействия с гипертекстовыми документами, входящими в сеть Интернет, существуют программы-браузеры. Преимущественнная часть браузеров использует стандарт DOM [1], обеспечивающий использование объектной модели документа [5]. Существуют несколько способов использоовать объектную модель документа, например, при помощи алгоритма Fiber и виртуальной объектной модели документа.

Целью данной работы является формализация алгоритма обновления гипертекстового документа Fiber, использующего виртуальную объектную модель документа, а также анализ эффективности его использования в современных web-приложениях. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) проанализировать существующие методы обновления гипертекстовых документов;
2) провести анализ предметной области, сформулировать критерии сравнения методов обновления гипертекстовых документов.
3) формализовать алгоритм Fiber обновления гипертекстового документа;
4) сравнить и проанализировать трудоёмкости изученных алгоритмов;
5) сделать выводы об эффективности использования алгоритма Fiber по сравнению с другими существующими алгоритмами.

1 Аналитическая часть

В данном разделе будут рассмотрены принципы работы с гипертекстовыми документами согласно объектной модели документа и виртуальной объектной модели документа.

1.1 Гипертекстовые документы

Гипертекстовым [6] документом является документ, состоящий из текстовых страниц, имеющих перекрёстные ссылки. В данной работе под гипертекстовыми документами будут подразумеваться документы, написанные при помощи языка гипертекстовой разметки (англ. *HyperText Markup Language* — HTML) [7], а именно документы, соблюдающие стандарт HTML5 [2].

Данный выбор обусловлен тем, что использование HTML5 получило широкое распространение в Всемирной сети [8] благодаря рекомендации [9] к использованию от Консорциума Всемирной паутины (англ. *World Wide Web Consortium* — W3C) [10]. Вследствие данной рекомендации HTML документы поддерживают большинство самых распространённых браузеров в России, такие, как Google Chrome, Яндекс.Браузер, Microsoft Edge, Opera и Firefox [11].

1.2 Web-приложение

Web-приложение (англ. *Web application*) [12] — web-страница (HTML или её вариант и CSS) или набор web-страниц, доставляемых про протоколу HTTP, которые используют обработку на стороне сервера или клиента (например, JavaScript) для обеспечения прикладного взаимодействия в браузере.

Web-приложения отличаются от простого web-контента наличием интерактивных элементов.

1.3 Существующие алгоритмы обновления гипертекстового документа

Из самых используемых фреймворков для разработки web-приложений на июнь 2023 года [13] можно выделить шесть, предоставляющих готовые решения в области обновления гипертекстовых документов: React (JavaScript), Angular (JavaScript), Vue.js (JavaScript), Flask (Python), Django (Python), Svelte (JavaScript).

В задаче отображения страницы можно выделить три следующие стадии:

1) подготовка HTML содержимого страницы;
2) построение объектной модели документа по HTML страницы;
3) отображение объектной модели документа на экран.

В данной работе рассматриваются алгоритмы, относящиеся к последним двум стадиям, т. е. построению и отображению DOM. Из-за этого будут рассмотрены готовые решения, относящиеся к этим двум стадиям.

Алгоритмы этих готовых решений можно разделить на две крупных категории: алгоритмы, использующие объектную модель документа и алгоритмы, использующие виртуальную объектную модель документа. Рассмотрим подробнее каждую из этих категорий.

1.4 Алгоритмы, использующие объектную модель документа

Объектная модель документа (англ. *Document Object Model* — DOM) [5] — программный интерфейс для HTML, XML и CSV документов. Он обеспечивает структурированное представление документа в виде дерева [14], и определяет способ, по которому структура может быть доступна для программы, для изменения структуры документа, его стиля и содержания. Представление DOM состоит из структурированной группы узлов и объектов, которые имеют свойства и методы.

Стадарт W3C DOM [1] формирует основы DOM, реализованные в большинстве современных браузеров. Для описания структуры DOM потребуются следующие термины: корневой, родительские и дочерние элементы. Корневой элемент находится в основании всей структуры и не имеет родительского элемента. Дочерние элементы не просто находятся внутри родительских, но и наследуют различные свойства от них. Рассмотрим, как выглядит DOM-представление HTML документа, представленного на листинге 1.

Листинг 1 – Пример простого HTML документа

```
<!DOCTYPE HTML>
<html>
<head>
  <link/>
  <meta/>
  <title/>
</head>
<body>
  <header/>
  <section>
    <div/>
    <p/>
    <a/>
  </section>
  <footer/>
</body>
</html>
```

Корневым элементом здесь является `html`, он не имеет родительского элемента и имеет два дочерних — `head` и `body`. По отношению друг к другу элементы `head` и `body` являются сиблингами (братьями и сестрами). В каждый из них можно вложить еще много дочерних элементов. Например, в `head`

обычно находятся `link`, `meta`, `script` или `title`.

Данный HTML документ будет иметь следующее DOM-представление, изображённое на рисунке 1.

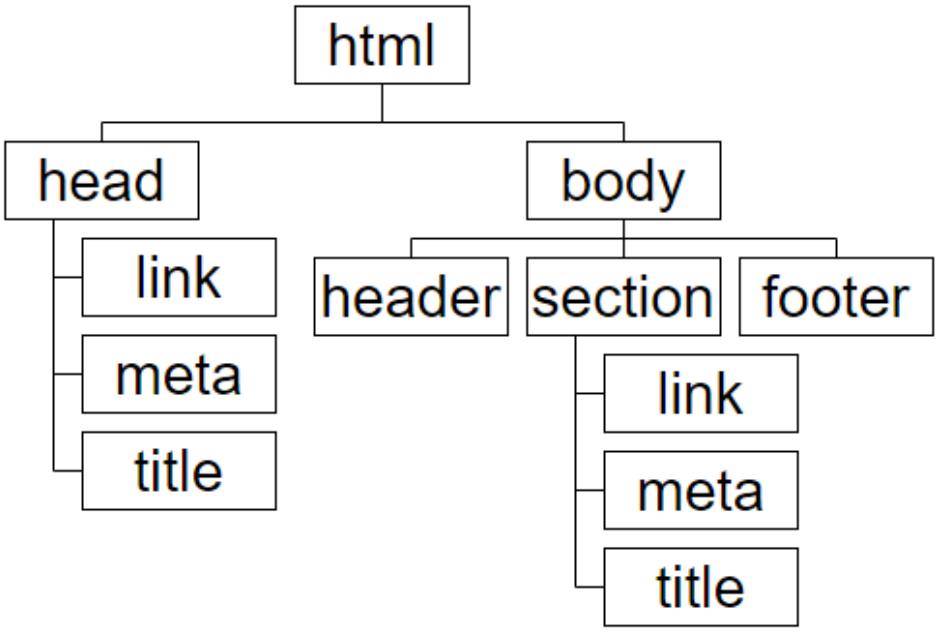

Рисунок 1 – Пример DOM-представления для простого HTML документа

Все эти теги не являются уникальными, и в одном документе может быть по несколько экземпляров каждого из них.

В `body` могут находиться разнообразные элементы. Например, в родительском `body` — дочерний элемент `header`, в элементе `header` — дочерний элемент `section`, в родительском `section` — дочерний `div`, в `div` — элемент `h3`, и, наконец, в `h3` — элемент `span`. В этом случае `span` не имеет дочерних элементов, но их можно добавить в любой момент. Пример того, как можно описать добавление данных элементов, представлен на рисунке 2.

Элементы могут наследовать не все, но многие свойства своих родителей — например, цвет, шрифт, видимость, размеры и т. д.

Таким образом, чтобы задать стиль шрифта на всей странице, потребуется не прописывать цвет для каждого элемента, а задать его только для `body`. А чтобы изменить наследуемое свойство у дочернего элемента, нужно прописать только ему новые свойства. Наследование удобно для создания единообразной

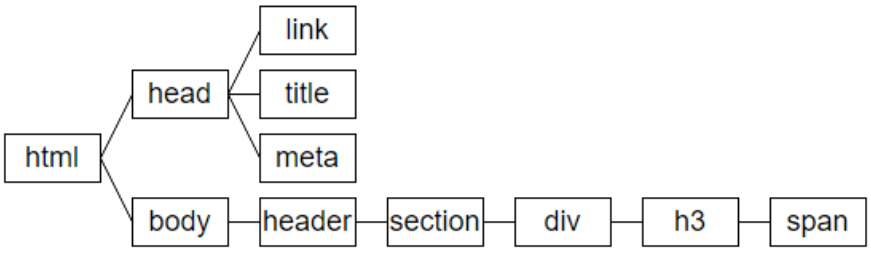

Рисунок 2 – Пример DOM-представления для чуть более сложного HTML документа

страницы. DOM-узлы содержат гораздо больше информации, чем просто данные о дочерних узлах [15]. Они также содержат информацию о родительском узле, стилях, обработчиках событий и свойствах элемента, его исходном коде и т. д. В таких браузерах, как Google Chrome, DOM-дерево страницы можно посмотреть, например, при помощи инструментов разработчика [16].

1.4.1 Обновления документа с использованием DOM

Обновление структуры DOM — распространённая и часто используемая операция, производимая, например, в случае, когда документ должен меняться в ответ на действия пользователя (любая активность на странице).

Рекомендация W3C в таком случае призывает повторно отрисовать обновлённое дерево с нуля — то есть, каждый раз, когда необходимо обновить дерево, будет использоваться алгоритм отображения [17]: DOM-дерево строится заново и производится операция вставки элемента n раз, где n - количество элементов.

Благодаря тому, что современные браузеры реализуют DOM и предоставляют API для взаимодействия с DOM-деревом, фреймворки, такие, как Angular, используют именно этот API в своих готовых решениях [18].

1.5 Алгоритмы, использующие виртуальную объектную модель документа

Основной проблемой DOM является то, что он никогда не был приспособлен для динамического интерфейса [3]. Он предоставляет удобный программный интерфейс, позволяющий взаимодействовать с ним из кода, но это не решает проблем с производительностью. Современные социальные сети, такие, как ВКонтакте, Twitter или Facebook будут использовать тысячи DOM узлов, взаимодействие с которыми может занимать ощутимое для пользователя время, в то время как создатели браузера Google Chrome, к примеру, рекомендуют [20] не использовать в одной странице более 800 узлов.

Одним из решений данной проблемы служит технология виртуальной объектной модели, которая, хоть и не является стандартом, позволяет по-прежнему взаимодействовать с DOM, но делать это как можно реже.

Виртуальная объектная модель документа (англ. *Virtual Document Object Model* — VDOM) [21] — концепт программирования, в которой идеальное («виртуальное») представление пользовательского интерфейса хранится в памяти и синхронизуется с «настоящим» DOM при помощи алгоритмов согласования.

Вместо того, чтобы работать с DOM напрямую, согласно концепции VDOM работа происходит с его легковесной копией, хранящийся непосредственно в памяти компьютера. За счёт этого операции вставки, сравнения, удаления и обхода узлов дерева происходит быстрее благодаря отсутствию необходимости отрисовывать изменения после каждой операции.

Когда в пользовательский интерфейс добавляются новые элементы, создаётся виртуальная модель DOM, представленная в виде дерева. Если состояние любого из элементов изменяется, создаётся новое виртуальное дерево DOM. Затем это дерево сравнивается с предыдущим виртуальным деревом.

Рассмотрим, как для простого HTML документа, представленного на листинге 2, осуществляется представление DOM- (рисунок 3) и VDOM-дерева

(листинг 3).

Листинг 2 – Простой HTML документ

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <link/>
</head>
<body>
  <ul class="list">
    <li class="list__item">List item</li>
  </ul>
</body>
</html>
```

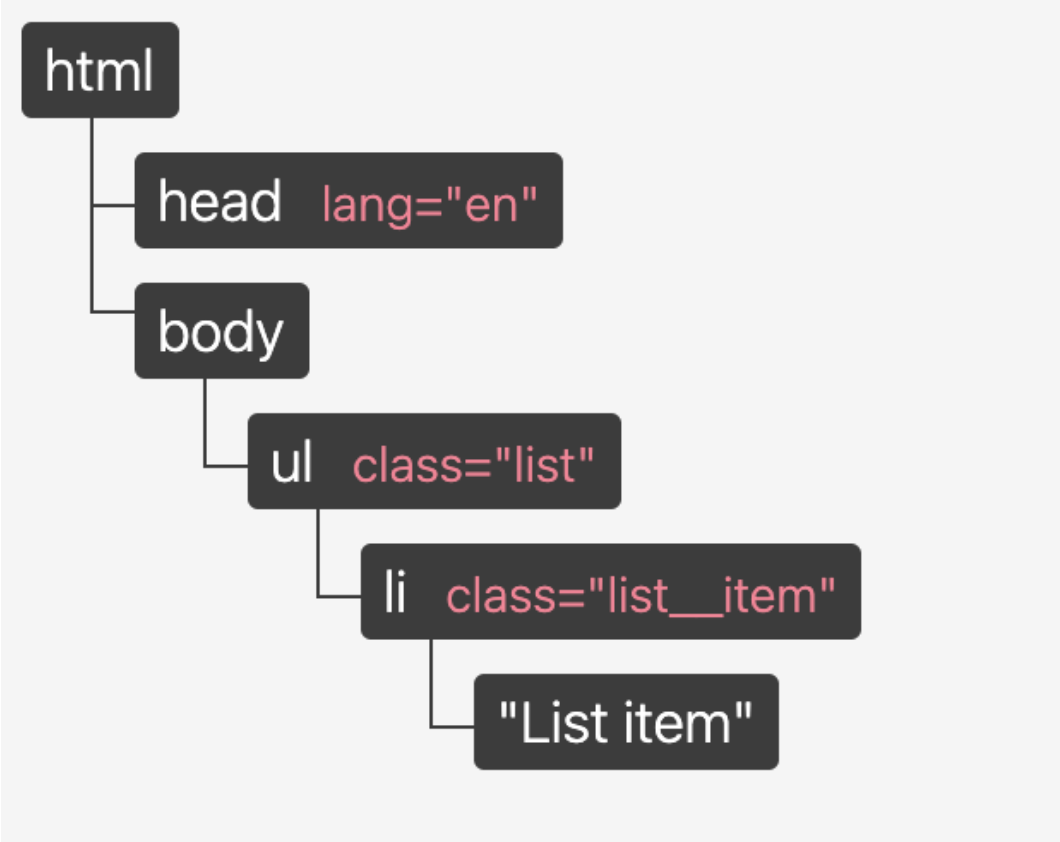

Рисунок 3 – DOM-дерево для простого HTML документа

Листинг 3 – VDOM-дерево для простого HTML документа

```
const vdom = {
  tagName: "html",
  children: [
    { tagName: "head" },
    {
      tagName: "body",
      children: [
        {
            tagName: "ul",
            attributes: { "class": "list" },
            children: [
                {
                    tagName: "li",
                    attributes: { "class": "list__item" },
                    textContent: "List item"
                } // end li
            ]
        } // end ul
      ]
    } // end body
  ]
} // end html
```

Данный пример служит лишь для демонстрации того, как может осуществляться представление VDOM. Реальные деревья содержат намного большее число узлов [15], а элементы DOM имеют гораздо больше полей (классы, идентификаторы, стили, обработчики событий, поля данных, типы и значения, вспомогательные поля и т. п.), и тем не менее содержат гораздо меньше информации, чем узлы DOM за счёт того, что нет необходимости хранить данные, помогающие визуализировать элемент (за это отвечает DOM).

Существует несколько способов использовать VDOM для оптимизации алгоритма отображения гипертекстового документа. с использованием алгоритма согласования и с использованием Fiber алгоритма.

1.5.1 Обновление документа с использованием VDOM и алгоритма согласования

Если состояние любого из элементов необходимо изменить, создаётся новое виртуальное дерево. Затем получившееся дерево сравнивается с предыдущим виртуальным деревом объектной модели документа и происходит вычисления наилучшего из возможных методов внесения изменений в реальной DOM. Это гарантирует минимальное количество операций с реальной DOM, что приводит к снижении стоимости обновления реальной модели.

На рисунке 4 показано виртуальное дерево DOM и процесс сравнения.

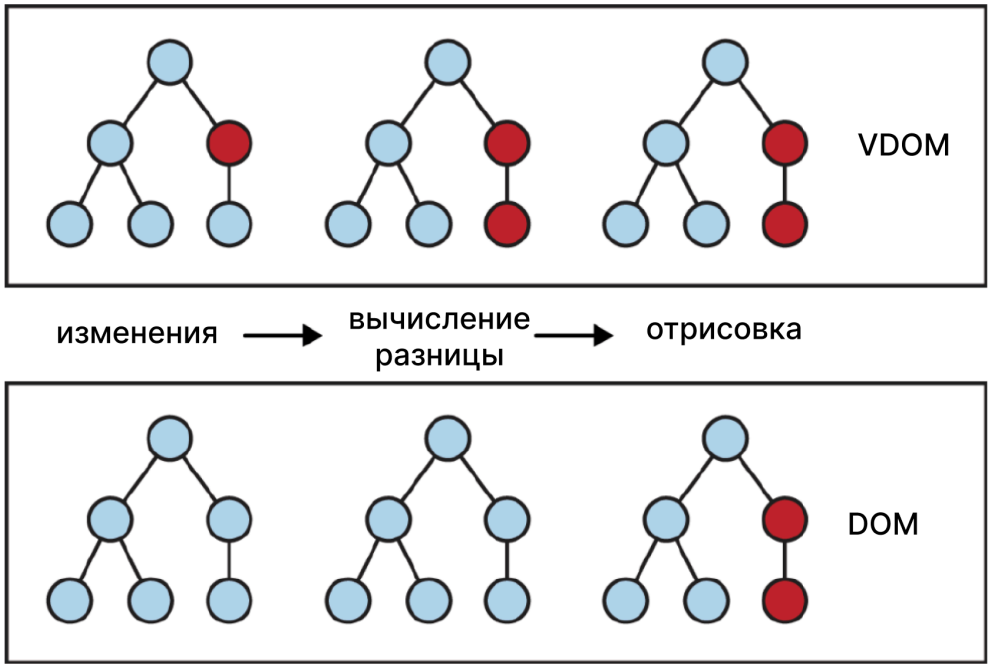

Рисунок 4 – Пример работы алгоритма обновления документа при помощи VDOM

Красными кружками обозначены узлы, которые изменились — эти узлы представляют собой элементы пользовательского интерфейса, состояние ко-

тоых изменилось. Затем вычисляется разница между предыдущей версией VDOM-дерева и текущей (посредством алгоритма согласования), после чего всё родительское поддерево повторно визуализируется для получения обновлённого пользовательского интерфейса.

1.5.1.1 Алгоритм согласования

Алгоритм согласования (англ. *reconciliation*) [22] — эвристический алгоритм решения проблемы трансформации одного дерева в другое за минимальное число операций. Он основан на следующих двух предположениях:

Во-первых, два элемента с разными типами произведут разные деревья.

Во-вторых, можно указать, какие дочерние элементы могут оставаться стабильными между разными отображениями при помощи специального параметра `key`.

При сравнении двух деревьев первым делом производится сравнение родительских элементов, начиная с корневого. Дальнейшее поведение различается в зависимости от типов родительских элементов.

Всякий раз, когда родительские элементы имеют различные типы, старое VDOM-дерево уничтожается и с нуля строится новое. Так, переходы от `<div>` к `<a>` или от `<img>` к `<div>` приведут к полному перестроению дерева, старое DOM-дерево также уничтожается.

Любые элементы, лежащие ниже родительского, будут уничтожены, а их состояние уничтожится. На листиге 4 приведён пример изменения узла, при котором меняется его тип и всё поддерево уничтожается.

Листинг 4 – Пример HTML-узлов разного типа, при сравнении которых деревья будут полностью перестроены

```
<!-- Старый элемент -->
<div>
  <a href="example.com" />
```

```
  </div>

  <!-- Новый элемент -->
  <span>
    <a href="example.com" />
  </span>
```

При сравнении DOM-элементов одного типа, алгоритм согласования проверяет их атрибуты, сохраняет лежащий в основе элементов DOM-узел и обновляет только изменённые атрибуты (пример таких элементов приведён на листинге 5).

Листинг 5 – Пример HTML-узлов одного типа, при сравнении которых будет сохранена лишь разница содержимого атрибутов

```
<!-- Старый элемент -->
<div class="before" title="stuff" />

<!-- Новый элемент -->
<div class="after" title="stuff" />
```

При рекурсивном обходе дочерних элементов DOM-узла алгоритм согласования проходит по спискам потомков одновременно и находит отличие. Например, при добавлении элемента в конец дочерних элементов, преобразование между этими деревьями работает хорошо, определяя минимальный набор операций.

Листинг 6 – Пример HTML-узлов при добавлении элемента в конец дочернего

```
<!-- Старый элемент -->
```

```
  <ul>
    <li>первый</li>
    <li>второй</li>
  </ul>

  <!-- Новый элемент -->
  <ul>
    <li>первый</li>
    <li>второй</li>
    <li>третий</li>
  </ul>
```

Алгоритм согласования сравнит два дерева `<li>первый</li>`, сравнит два дерева `<li>второй</li>`, а затем вставит дерево `<li>третий</li>`.

При вставке элемента в начало, прямолинейная реализация такого алгоритма будет работать не эффективно. Например, преобразование между этими деревьями работает плохо, минимальный набор операций для преобразованя одного дерева в другое не будет найден.

Алгоритм согласования будет преобразовывать каждого потомка, вместо того, чтобы оставить элементы `<li>Санкт-Петербург</li>` и `<li>Москва</li>` . Пример таких узлов продемонстрирован на листинге 7.

Листинг 7 – Пример HTML-узлов при добавлении элемента в начало дочернего

```
  <!-- Старый элемент -->
  <ul>
    <li>Санкт-Петербург</li>
    <li>Москва</li>
  </ul>


```

```
  <!-- Новый элемент -->
  <ul>
    <li>Ростов-на-Дону</li>
    <li>Санкт-Петербург</li>
    <li>Москва</li>
  </ul>
```

Для решения этой проблемы существуют вспомогательные атрибуты `key` — ключи [23]. Когда у дочерних элементов есть ключи, алгоритм согласования использует их, чтобы сопоставить потомков исходного виртуального дерева с потомками полученного виртуального дерева.

Значение атрибута `key` при этом должно быть уникальным. Так, как значение ключа можно использовать уникальный идентификатор элемента, если он есть, или же можно добавить новое свойство идентификатора или прохешировать данные, чтобы сгенерировать ключ. Достаточно, чтобы ключ элемента был уникальным среди его соседей, а не глобально.

Например, если добавить `key` к примеру выше, преобразование дерева будет верным.

Листинг 8 – Пример HTML-узлов при добавлении элемента в начало дочернего с использованием ключей

```
  <!-- Старый элемент -->
  <ul>
    <li key="2015">Санкт-Петербург</li>
    <li key="2016">Москва</li>
  </ul>

  <!-- Новый элемент -->
  <ul>
```

```
9       <li key="2014">Ростов-на-Дону</li>
10      <li key="2015">Санкт-Петербург</li>
11      <li key="2016">Москва</li>
12    </ul>
```

Алгоритм согласования является эвристическим [24] алгоритмом, и если предположения, на которых он основан, не соблюдены, пострадает производительность.

Так, алгоритм не будет пытаться сопоставить поддеревья элементов разных типов, а ключи должны быть стабильными (например, ключ, произведённый случайно, не является стабильным), предсказуемым и уникальным. Нестабильные ключи вызовут необязательное пересоздание многих экземпляров элемента и DOM-узлов, что может вызывать ухудшение производительности и потерю состояния у дочерних элементов.

1.5.2 Обновление документа с использованием Fiber алгоритма

Fiber — основной алгоритм сравнения библиотеки React, начиная с 16 версии конкурирующий с алгоритмом согласования [25]. Для пользовательского интерфейса не важно, чтобы каждое обновление было применено сразу; фактически такое поведение будет лишним, оно будет способствовать падению количества кадров в секунду и ухудшению пользовательского опыта [26].

Волокно (англ. *fiber*) — объект, являющийся абстракцией работы, которую необходимо выполнить [26]. Работой являются любые вычисления, которые должны быть выполнены для того, чтобы определить, какие объекты необходимо обновлять, а какие можно оставить в прежнем состоянии. Такая работа должна иметь возможность быть остановленной и возобновлённой. Рассмотрим следующий пример: пусть имеются HTML-элементы, представленные на листинге 9.

Листинг 9 – Пример HTML-элементов для демонстрации структуры волокна

```
<div class="HostRoot">
    <li class="list">
        <button class="btn">^2</button>
        <div>1</div>
        <div>2</div>
        <div>3</div>
    </li>
</div>
```

Волокно, отвечающее за вычисления, связанные с кнопкой, имеющий класс btn, будет иметь следующий вид, представленный на листинге 10.

Листинг 10 – Fiber объект для элемента кнопки

```
const fiber = {
    stateNode: HTMLButtonElement,
    child: null,
    return: HTMLLIElement,
    sibling: HTMLDivElement,
    props: {
        className: "btn",
    },
    changed: 0,
    tag: 0
}
```

Поля объекта, представленного на листинге 10, имеют следующий смысл:

1) stateNode — указатель на объект DOM-дерева, соответствующего элементу волокна (в данном случае кнопке);
2) child — указатель на волокно первого дочернего элемента (в данном случае дочерних элементов нет);
3) return — указатель на волокно родительского элемента (в данном слечае список);
4) sibling — указатель на первый соседний элемент (в данном случае элемент div);
5) props — объект, хранящий вспомогательные свойства элемента (в данном случае, класс элемента);
6) changed — флаг, изменяющийся при всплытии события, инициирующего изменение (в данном случае изменение появилось не в данном элементе);
7) tag — флаг, отмечающий данный элемент как требующий обновления (в данном случае элемент не требует обновления).

В процессе работы алгоритма строится два дерева fiber объектов: дерево, соответствующее текущему состоянию элементов на странице (назовём его current) и дерево, в котором производятся вычисления при обновлении (назовём его workInProgress). При обновлении происходит разделение вычислений на две фазы: фаза рендера (англ. *render phase*), выполняемая при помощи функции requestIdleCallback и фаза закрепления (англ. *commit phase*), переносящая изменения в DOM (см. рисунок 5).

Прежде, чем рассмотреть подробнее каждую из двух фаз, необходимо познакомиться с функцией requestIdleCallback и тем, как она должна работать согласно предложению W3C от 28 июня 2022 года [27].

1.5.2.1 Функция requestIdleCallback

requestIdleCallback — функция, впервые представленная в черновом варианте [28] W3C 17 сентября 2015 года и предназначенная для предоставле-

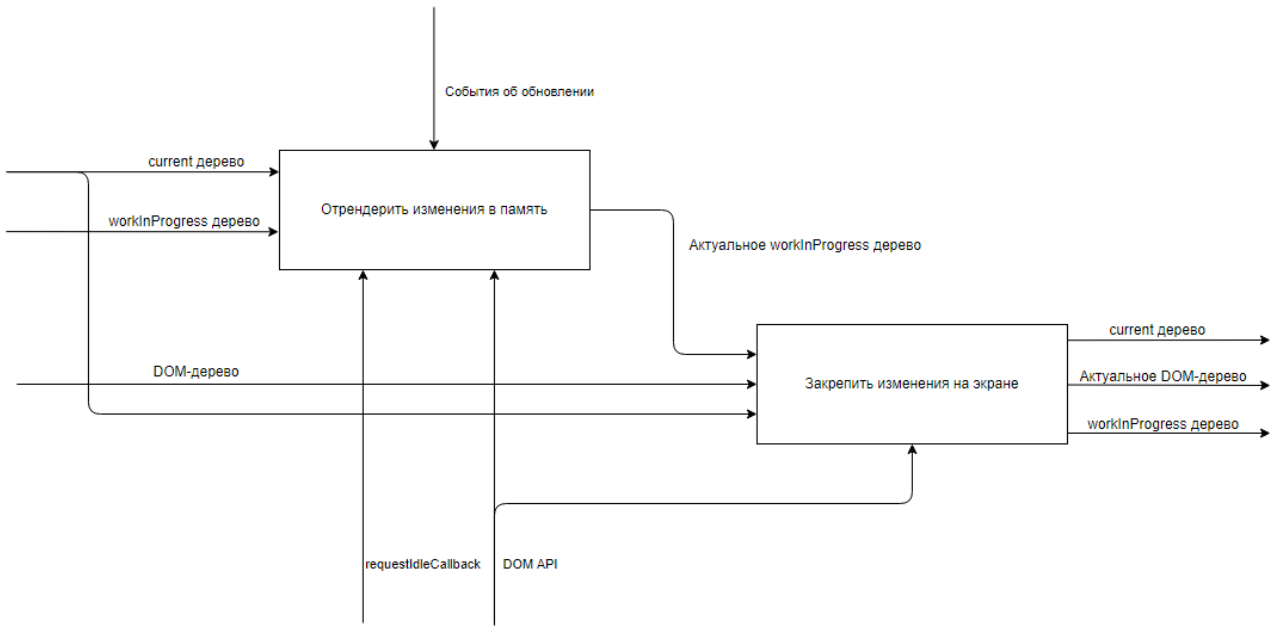

Рисунок 5 – Связь фаз рендера и закрепления в алгоритме Fiber

ния разработчикам возможности использовать периоды простоя браузера для выполнения фоновой и низкоприоритетной работы. 10 октября 2017 года данная функция приобрела статус официально рекомендованной для реализации фунции благодаря выходу соответствующей рекомендации W3C [29], поэтому сейчас она реализована такими браузерами, как Google Chrome, Microsoft Edge, Firefox и Opera [30], т. е. самыми популярными браузерами в России [11].

Данная функция имеет один обязательный параметр callback — ссылка на функцию, которая должна быть вызвана в ближайшем будущем, когда цикл событий будет в режиме ожидания. Функции callback при вызове передаётся объект, позволяющий ей при помощи вызова функции timeRemaining определить, сколько времени цикл событий будет в режиме ожидания, т. е. когда необходимо вернуть управление циклу основных событий [31].

Функция requestIdleCallback также имеет один опциональный параметр options, который позволяет задать значение timeout для браузера. Если timeout задан и и имеет положительное значение, а callback ещё не был вызван по времени наступления миллисекундного timeout, callback будет вызван в течение следующего периода простоя, даже если это может привести к негативному

влиянию на производительность [30].

Возвращает requestIdleCallback значение идентификатора, который, будучи переданным в функцию cancelIdleCallback, позволит отменить вызов функции callback [29].

1.5.2.2 Фаза рендера

Фаза рендера (англ. *render phase*) алгоритма Fiber предназначена для определения необходимых для обновления документа операций в фоновом режиме, таки образом, чтобы работу можно было выполнять во время простоя браузера при помощи функции requestIdleCallback. Цель данной фазы — привести workInProgress дерево fiber-элементов в целостное состояние, т. е. такое состояние, которое можно отобразить на web-страницы без нарушения целостности внешнего вида страницы.

Во время фазы рендера используются две функции: beginWork и completeWork. beginWork принимает на вход текущий элемент в workInProgress дереве, после чего проверяет, необходимо ли данному элементу обновление. В случае, если оно необходимо, поле tag соответствующего текущему fiber-элемента устанавливается равным 1, и возвращается следующий элемент workInProgress дерева, подлежащий обработке (дочерний, или, если дочерние элементы обработаны, соседний).

Функция completeWork принимает на вход current дерево fiber-элеметов, workInProgress дерево fiber-элементов и current, текущий элемент workInProgress дерева. Она проверяет значение поля tag и, если оно установлено, осуществляет рендеринг элемента current в памяти (иными словами, создаёт необходимые DOM-элементы и хранит их в памяти, не отображая их на web-странице и не внося изменений в гипертекстовый документ). После чего, если ещё существуют элементы, подлежащие рендеру, возвращается fiber-элемент для первого такого элемента (дочернего, или, если дочерние элементы отрисованы, соседне-

го).

1.5.2.3 Фаза закрепления

Фаза закрепления (англ. *commit phase*) предназначена для изменения гипертекстового документа и переноса созданных в фазе рендера DOM-элементов в текущее DOM-дерево с использованием предаставленного браузером API. В фазе закрепления не производится дополнительных вычислений, связанных с тем, необходимо менять элемент или нет. Вместо этого осуществляется обход workInProgress дерева Fiber-элементов и происходит отображение DOM-элементов.

Отметим, что основное отличие фазы закрепления от фазы рендера заключается в том, что она должна быть выполнена от начала до конца, без прерываний. Это необходимо для соблюдения целостности внешнего вида web-страницы: при передаче управления основному циклу событий возможны изменения внешнего вида web-страницы, расходящиеся с произведёнными в фазе рендера вычислениями. Фаза рендера же не выполняется от начала до конца, вычисления происходят лишь в время, указанное функцией requestIdleCallback.

1.5.3 Сравнение алгоритмов с использованием VDOM

Сравним два рассмотренных выше алгоритма: алгоритм с использованием согласования и Fiber-алгоритм. Обновление документа с использованием алгоритма соогласования осуществляется от начала до конца, что может быть рассмотренно и как преимущество, и как недостаток. Как преимущество, потому что несмотря на пользовательские вычисления (т. е. вычисления в основном цикле событий) обновление документа будет произведено в сооответствии с обусловленной эвристикой. Пользовательские вычисления будут произведены лишь после исполнения алгоритма согласования, что гарантирует корректное обновление документа.

Тем не менее, в случае, когда изменений на странице много [26], они могут занимать время, большее чем 1 кадр (при частоте обновления экрана 60 раз в секунду 1 кадр должен занимать не более, чем 16.67 миллисекунд). Такая задержка приводит к тому, что изображение обновляется ”скачками”, теряет плавность или даже приводит к переходу web-страницы в состояния, приводящие к ошибкам [25].

Данная проблема решается переходом от алгоритма согласования к Fiber-алгоритму, согласно которому преимущественная часть вычислений осуществляется во время между обновлением кадров на экране [27]. Несмотря на это отсутствует информация о том, какой из этих алгоритмов является наиболее эффективным с точки зрения вычисления. Чтобы ответить на этот вопрос, необходимо рассмотреть схемы изученных алгоритмов и оценить их трудоёмкость, что позволят оценить эффективность использования алгоритмов как с точки зрения пользовательского опыта и плавности изображения, так и с точки зрения производительности.

1.6 Критерии сравнения алгоритмов обновления гипертекстовых документов

В данной работе осуществляется сравнение изученных выше алгоритмов по трём критериям: пользовательский опыт, трудоёмкость алгоритма и затраты по памяти. Рассмотрим каждый из этих критериев подробнее.

Пользовательский опыт (англ. *User Experience* — UI) — критерий, основывающийся на ощущениях пользователя от использования конкретноого продукта [32]. Эмпатичный подход (англ. *empathic approach*) — способ оценки пользовательского опыта, берущий за основу эмоциональный опыт человека. Данный подход позволяет, при наличии информации о необходимостях, желаниях и мотавициях человека оценить пользовательские ощущения, что позволяет разрабатывать идеи и продукты, удоволетворяющие желаниям пользовате-

ля. Отметим, что выбор пользовательского опыта от использования алгоритмов как критерия для сравнения обусловлен тем, что количественные характерики, необходимые для оценки плавности изображения при работе алгоритма (например, количество кадров в секунду), не были измерены и зафиксированы в достоверных источниках, а произвести измерения в данной работе не представляется возможным. Тем не менее, существует возможность оценить трудоёмкость изученных алгооритмов.

Трудоёмкость алгоритма — количество операций, использованных в алгоритме для решения поставленной задачи [33]. Используется при наличии формализованной модели вычислений и формального описания алгоритма.

Затраты по памяти — зависимость необходимой для работы алгоритма памяти. Разделяют такие виды затрат по памяти, как память входа, память выхода и дополнительная память — именно по данным видам памяти необходимо сравнить изученные алгоритмы.

Вывод

В данном разделе были рассмотрены существующие методы обновления гипертекстовых документов: алгоритмы, использующие объектную модель документа, рендеринг на стороне сервера и алгоритмы, использующие виртуальную объектную модель документа. Как один из алгоритмов, использующих VDOM, был рассмотрен алгоритм согласования, находящий минимальный набор операций, необходимых для преобразования одного дерева в другое. Также был рассмотрен Fiber алгоритм и изучены две его составляющие: фаза рендера и фаза закрепления. В соответствии с поставленной целью, необходимо разработать алгоритмы обновления документа с использованием DOM и VDOM, а также алгоритма согласования и Fiber-алгоритма.

На вход алгоритма обновления с использованием DOM будут подаваться обрабатываемый элемент, изменяемый элемент и элемент, который нужно

отрисовать вместо него. На вход алгоритма обновления документа с использованием VDOM будут подаваться корни старого VDOM-дерева и нового VDOM-дерева, а на вход алгоритма согласования - обрабатываемые узлы старого и нового VDOM-деревьев, а также указатели на массивы изменений узлов и поддеревьев. На вход Fiber-алгоритма будут подаваться корни текущего Fiber-дерева и workInProgress-дерева.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов обновления гипертекстового документа с использованием DOM, VDOM и алгоритма согласования, а также алгоритма Fiber. Также будут найдены их трудоёмкости и произведено их сравнение на основе полученных результатов.

2.1 Разработка алгоритмов

На рисунках 6–9, представлены схемы алгоритмов обновления документа с использованием DOM, VDOM, а также алгоритма согласования соответственно, разработанных на основании приведённых в части 1 описаниях.

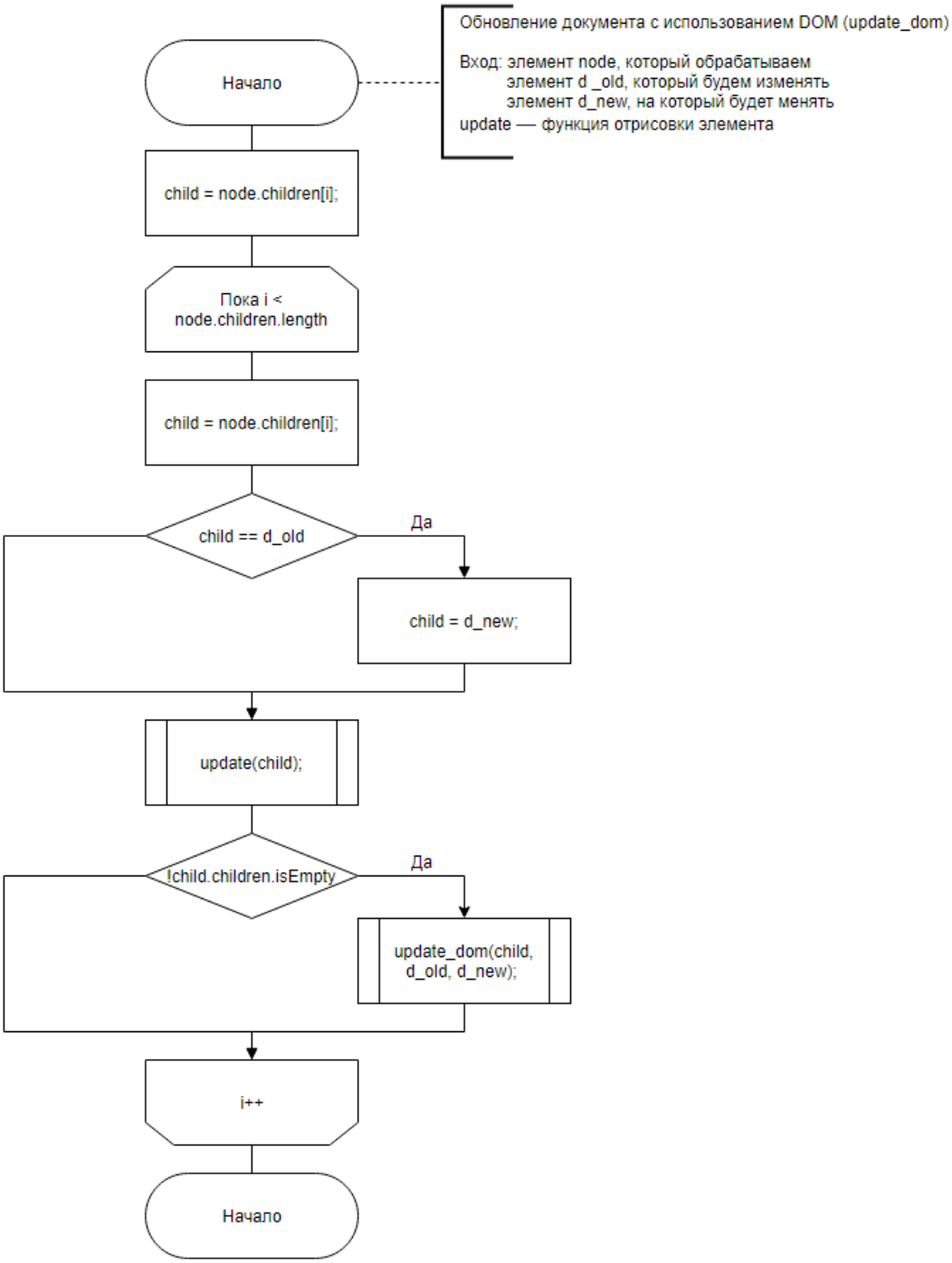

Рисунок 6 – Схема алгоритма обновления документа с использованием DOM

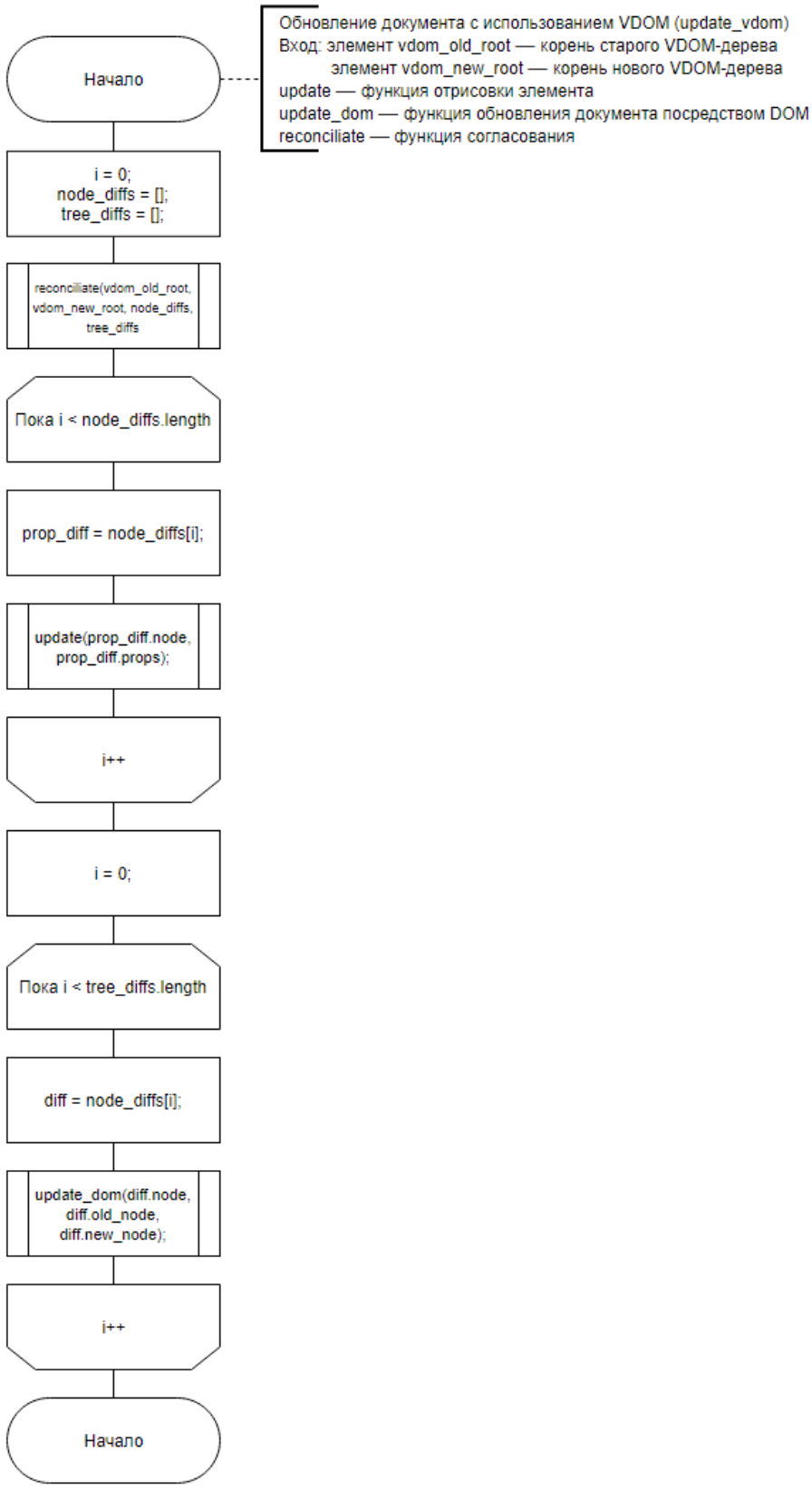

Рисунок 7 – Схема алгоритма обновления документа с использованием VDOM

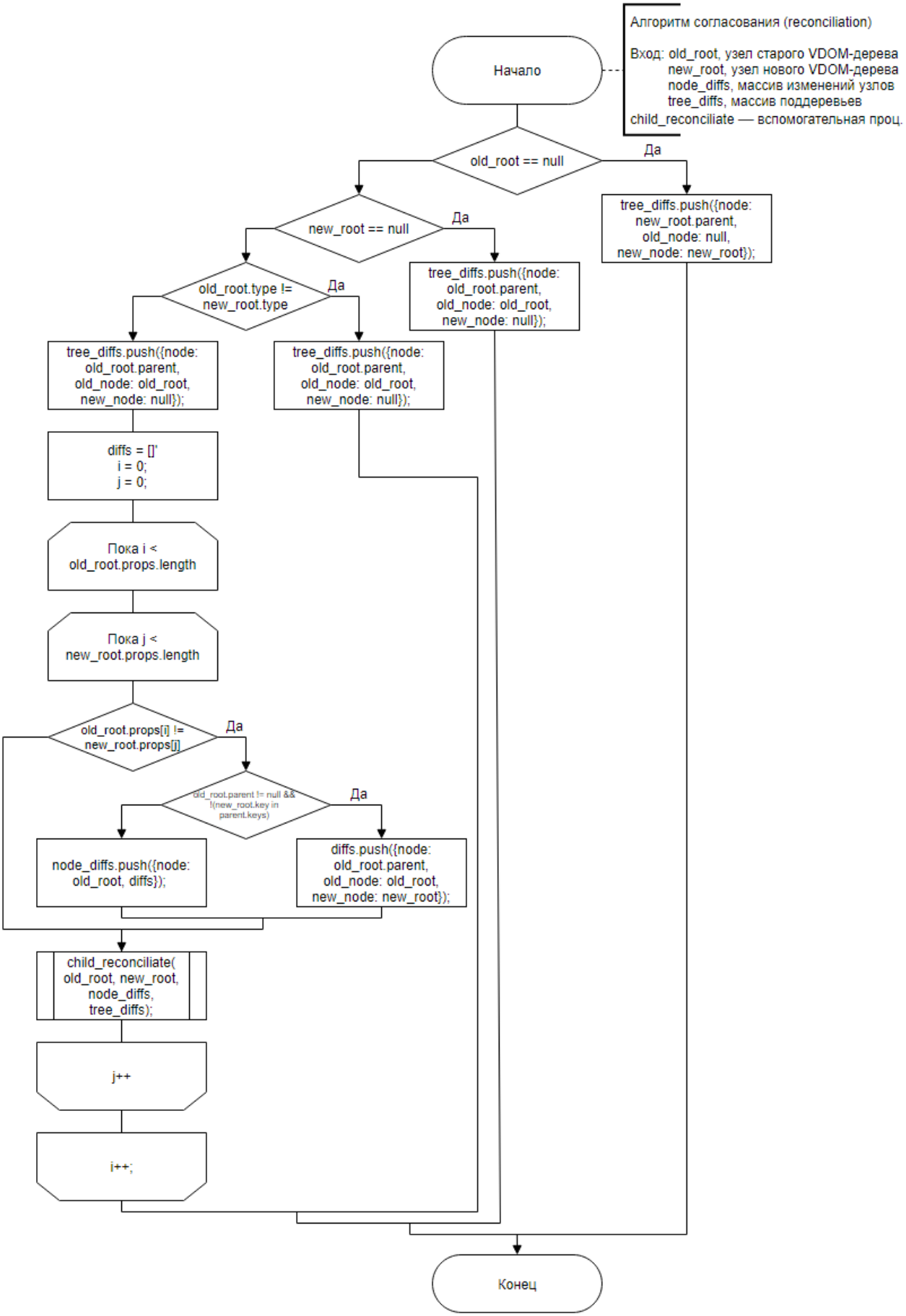

Рисунок 8 – Схема алгоритма согласования, часть 1

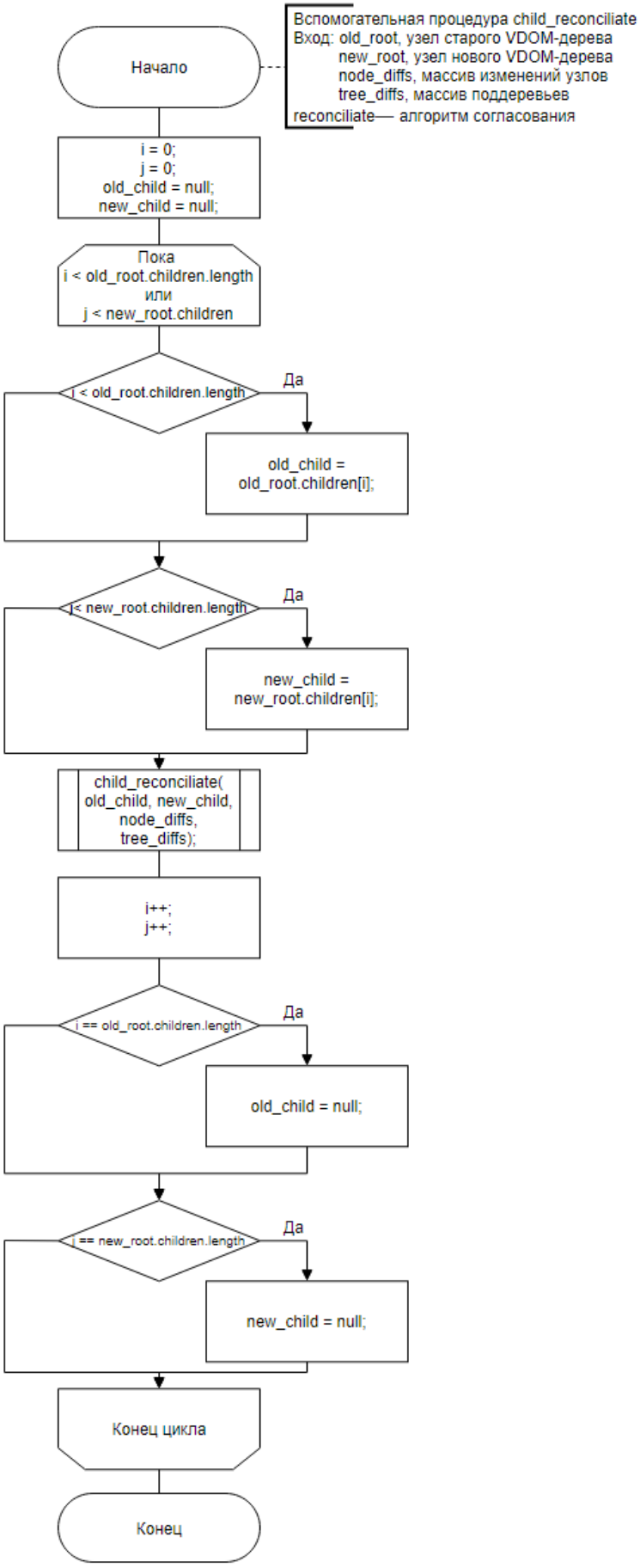

Рисунок 9 – Схема алгоритма согласования, часть 2

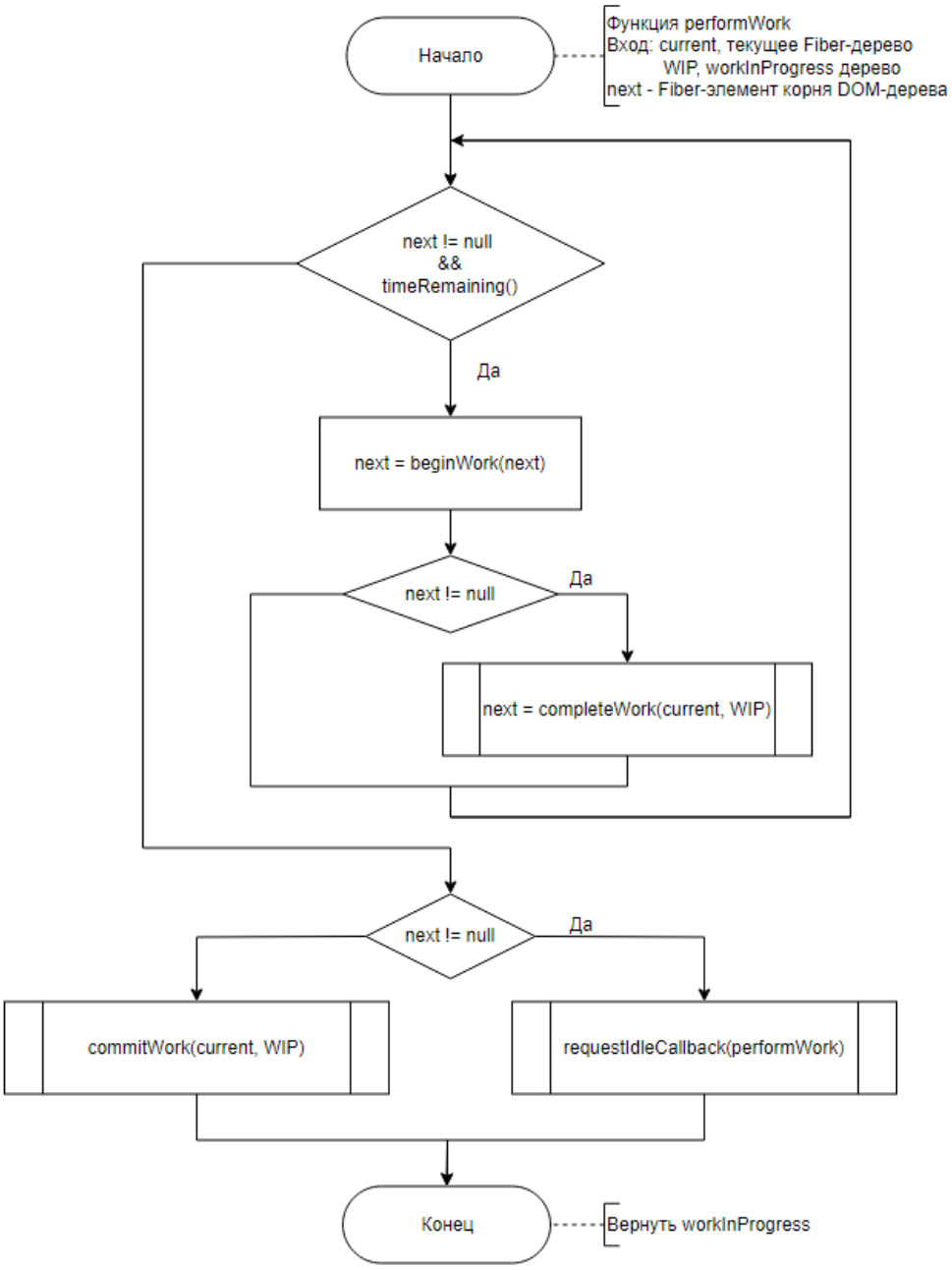

Рисунок 10 – Схема алгоритма Fiber

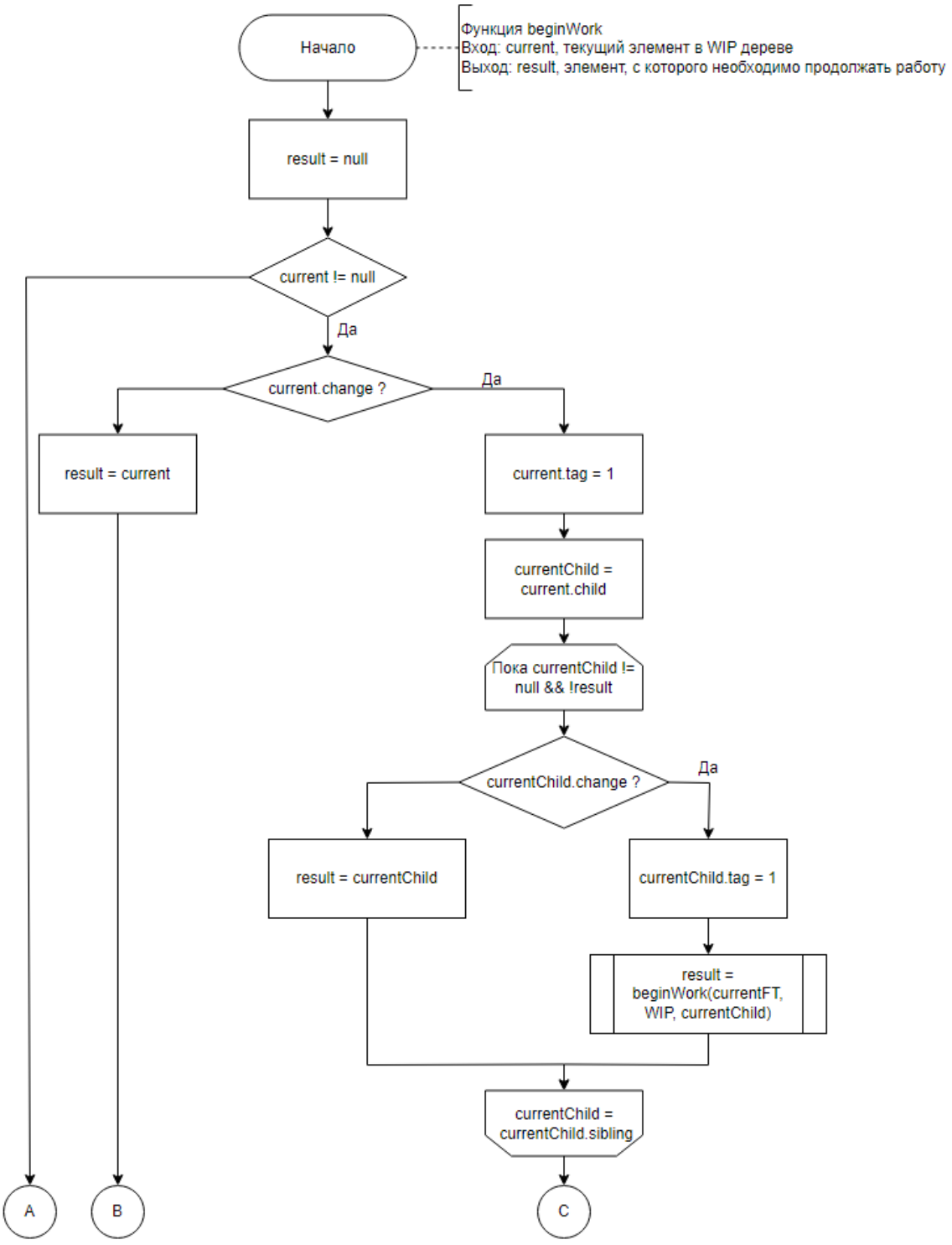

Рисунок 11 – Схема алгоритма функции beginWork, часть 1

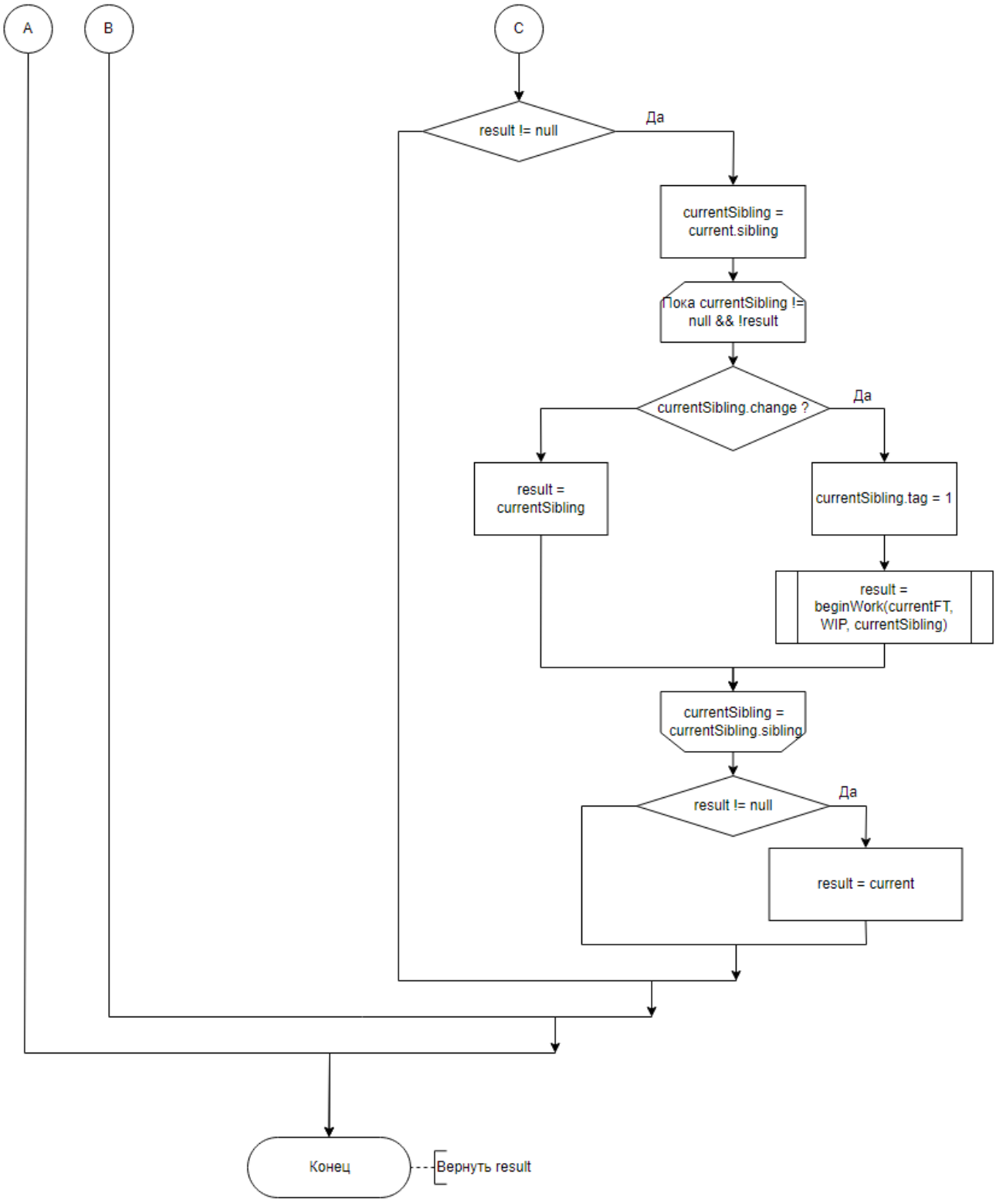

Рисунок 12 – Схема алгоритма функции beginWork, часть 2

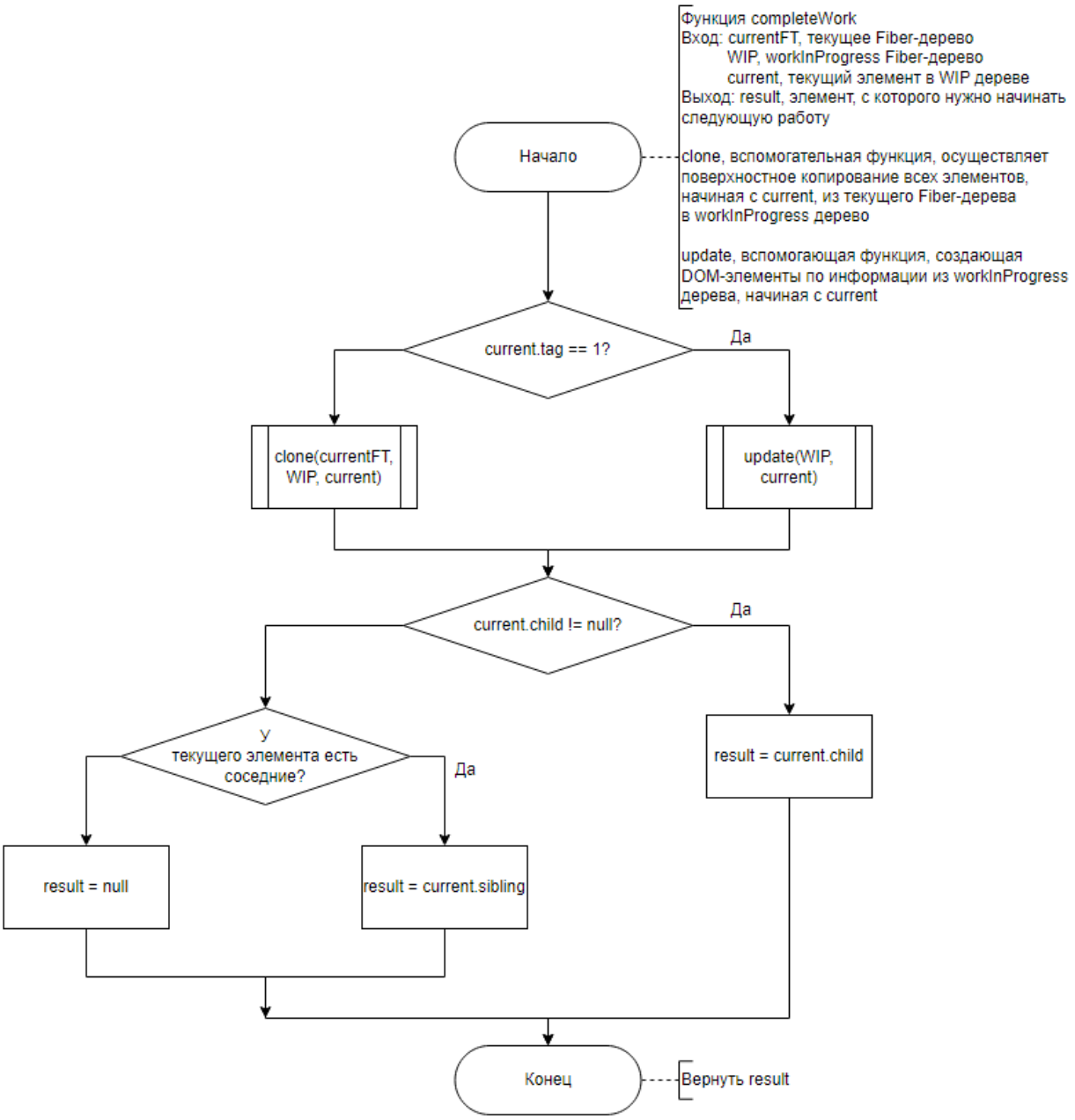

Рисунок 13 – Схема алгоритма функции completeWork

2.2 Модель вычислений для проведения оценки трудоёмкости

Введем модель вычислений [33], которая потребуется для определния трудоёмкости каждого отдельно взятого алгоритма.

1) Операции из списка (1) имеют трудоёмкость, равную 1.

$$+, -, /, *, \%, =, +=, -=, *=, ==, !=, <, >, <=, >=, [], ++, --, . \quad (1)$$

2) Трудоёмкость оператора выбора `if условие then A else B` рассчитывается по формуле (2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \quad (2)$$

3) Трудоёмкость цикла рассчитывается по формуле (3).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремент}} + f_{\text{сравнения}}) \quad (3)$$

4) Трудоёмкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Определим трудоёмкость выбранных алгоритмов по их схемам согласно выбранной модели вычислений для оценки.

2.3.1 Алгоритм обновления документа с использованием DOM

Введём следующие обозначения:

1) n — количество узлов в DOM-дереве;
2) old — случайная величина, значение которой может быть 1 или 0 в зависимости от того, нужно ли менять значение узла на новое;
3) $E(old)$ — математическое ожидание случайной величины old;
4) x — трудоёмкость операции `update`, $x >> 1$.

Поскольку величина old может принимать значение 1 только в случае одного узла и 0 во всех остальных случаях, можно найти значение её математического ожидания по формуле (4).

$$E(old) = 1 \cdot \frac{1}{n} + 0 \cdot \frac{n-1}{n} = \frac{1}{n} \tag{4}$$

Трудоёмкость алгоритма обновления документа с использованием DOM будет вычисляться по формуле (5).

$$f_{dom} = n \cdot (2 + 1 + E(old) + x + 2 + 3) \tag{5}$$

С учётом формулы (4) получим итоговую формулу (6) для расчёта трудоёмкости алгоритма обновления документа с использованием DOM.

$$f_{dom} = xn + 8n + 1 \tag{6}$$

Поскольку $x >> 1$, $f_{dom} \approx \Theta(xn)$.

2.3.2 Алгоритм обновления документа с использованием VDOM

Введём следующие обозначения:

1) n — количество узлов в изменяемом VDOM-дереве;
2) k — случайная величина, которая может принимать значения от 0 до n включительно в зависимости от того, в каком количестве узлов необходимо изменить содержимое;
3) $E(k)$ — математическое ожидание случайной величины k;
4) t — случайная величина, которая может принимать значения от 0 до $n-k$ включительно в зависимости от того, какое количество поддеревьев требует полной перерисовки;
5) $E(t)$ — математическое ожидание случайной величины t;
6) x — трудоёмкость операции `update`, $x >> 1$;
7) $f_{reconciliate}$ — трудоёмкость алгоритма согласования.

Поскольку случайная величина k принимает свои значения равновероятно, можно найти её математическое ожидание по формуле (7).

$$\begin{aligned} E(k) &= \frac{n}{n+1} + \frac{n-1}{n+1} + \ldots + \frac{1}{n+1} + \frac{0}{n+1} \\ &= \frac{1}{n+1} \cdot \frac{(n+0) \cdot (n+1)}{2} = \frac{n}{2} \end{aligned} \tag{7}$$

Аналогичным образом найдём математическое ожидание случайной величины t, формула (8).

$$\begin{aligned} E(t) &= \frac{n - E(k)}{n - E(k) + 1} + \frac{n - E(k) - 1}{n - E(k) + 1} + \ldots + \frac{1}{n - E(k) + 1} \\ &+ \frac{0}{n - E(k) + 1} = \frac{1}{n - E(k) + 1} \cdot \frac{(n - E(k) + 0) \cdot (n - E(k) + 1)}{2} \\ &= \frac{n - E(k)}{2} = \frac{n}{4} \end{aligned} \tag{8}$$

Трудоёмкость алгоритма обновления документа с использованием VDOM будет вычисляться по формуле (5).

$$\begin{aligned} f_{vdom} &= 2 + f_{reconciliate} + 2 + E(k) \cdot (3 + x) + 2 + E(t) \cdot (3 + f_{dom}) \\ &= E(k) \cdot (x + 3) + E(t) \cdot (f_{dom} + 3) + 6 \end{aligned} \tag{9}$$

С использованием формул (7) и (8) получим итоговую формулу (10) расчёта трудоёмкости алгоритма обновления документа с использованием VDOM.

$$f_{vdom} = \frac{n \cdot (3 + x)}{2} + \frac{n \cdot (3 + f_{dom})}{2} + f_{reconciliate} + 6 \tag{10}$$

2.3.3 Алгоритм согласования

Для алгоритма согласования рассмотрим худший и лучший случай.

Худшим случаем будем считать случай, при котором необходимо изменить все параметры всех узлов дерева, и добавить новые узлы в каждый из листовых узлов. Введём следующие обозначения:

1) n — количество узлов в изменяемом VDOM-дереве;

2) m — количество параметров в одном узле;

3) k — количество листов изменяемого VDOM-дерева;

4) λ — количество новых узлов, добавлемых в каждый лист изменяемого VDOM-дерева.

Тогда трудоёмкость алгоритма согласования для худшего случая может быть вычислена по формуле (11).

$$\begin{aligned} f_{reconciliate_{worst}} &= (n+\lambda)\cdot(5+1+2+m\cdot(3+2+m\cdot(3+1+3)) \\ &+ 2+7+4+4+7) + (n+\lambda)\cdot(2+2+2) + n\cdot 3 \\ &+ (n+\lambda)\cdot 3 + \lambda + 1 = \\ &= (n+\lambda)\cdot(41+m\cdot(5+7m)) + 3n + \lambda + 1 \\ &= (7m^2+5m)\cdot(n+\lambda) + 44n + 42\lambda + 1 \end{aligned} \tag{11}$$

Лучшим случаем для алгоритма согласования будем считать случай, при котором разница в типе узла находится в первом же узле, т. е. возникает необходимость перерисовать всё дерево целиком, что является худшим случаем для алгоритма обновления документа с использованием VDOM и алгоритма согласования.

Трудоёмкость алгоритма согласования для лучшего случая считается по формуле (12).

$$f_{reconciliate_{best}} = 1+2+2+4 = 9 \tag{12}$$

2.3.4 Алгоритм Fiber

Для алгоритма Fiber рассмотрим худший и лучший случай, аналогично худшему и лучшему случаю при анализе трудоёмкости алгоритма согласования.

Введём следующие обозначения:

1) n — количество узлов в изменяемом (workInProgress) дереве;

2) m — количество параметров в одном узле;
3) k — количество листов workInProgress-дерева;
4) λ — количество новых узлов, добавлемых в каждый лист изменяемого VDOM-дерева;
5) f_r — суммарная трудоёмкость вызовов функции requestIdleCallback и timeRemaining;
6) x — трудоёмкость функции update (для обновления DOM-элементов);
7) f_{begin} — суммарная трудоёмкость вызовов функции beginWork;
8) $f_{complete}$ — суммарная трудоёмкость вызовов функции completeWork;
9) f_{commit} — трудоёмкость алгоритма, выполняемого во время фазы закрепления;

Трудоёмкость алгоритма Fiber для худшего случая может быть вычислена по формуле 13.

$$f_{fiber} = 1 + f_r + f_{begin} + f_{complete} + f_{commit} \tag{13}$$

В худшем случае, суммарная трудоёмкость функций beginWork и completeWork может быть вычислена по формулам 14 и 15.

$$\begin{aligned} f_{beginWork_{worst}} &= n \cdot (1+1+1+2+2+1 \cdot (1+1+2)+2+1+1)+ \\ &+ \lambda \cdot (1+1+1+2+2+1+1+1+2+1+1+1) \\ &= 15n + 15\lambda \end{aligned} \tag{14}$$

$$\begin{aligned} f_{completeWork_{worst}} &= n \cdot (m+1+2+1) + k \cdot (x+2+1+1+2+1) \\ &= mn + 4n + kx + 7k \end{aligned} \tag{15}$$

В лучшем случае, суммарная трудоёмкость функций beginWork и completeWork может быть вычислена по формулам 16 и 17.

$$f_{beginWork_{best}} = n \cdot (1+2+1+1) = 5n \tag{16}$$

$$f_{completeWork_{best}} = n \cdot (2+x) \tag{17}$$

Трудоёмкость фазы закрепления f_{commit} будет одинаковой и в лучшем, и в худшем случае, поскольку она представляет из себя перенос значений ссылок на DOM-элементы, отрисованные и хранящиеся в workInProgress дереве, в реальное DOM-дерево. Считая, что n — количество элементов workInProgress-дерева, будем считать, что $f_{commit} = n$.

Трудоёмкость алгоритма Fiber для худшего случая считается по формуле 18.

$$\begin{aligned} f_{fiber_{worst}} &= 1 + f_r + 15n + 15\lambda + mn + 4n + kx + 7k + n \\ &= kx + mn + 20n + 7k + 15\lambda + 1 \end{aligned} \tag{18}$$

С учётом того, что вызовы функций requestIdleCallback и timeRemaining обязаны выполняться в течении 1 мс согласно стандарту W3C [29], будем считать, что $1 < f_r << x$. Тогда трудоёмкость алгоритма Fiber для худшего случая может быть оценена по формуле 19.

$$f_{fiber_{worst}} \approx \Theta(kx + mn) \tag{19}$$

Трудоёмкость алгоритма Fiber для лучшего случая считается по формуле 20 и может быть оценена по формуле

$$\begin{aligned} f_{fiber_{best}} &= 1 + f_r + f_{beginWork_{worst}} + f_{completeWork_{worst}} + f_{commit} = \\ &= 1 + f_r + 5n + n \cdot (2 + x) + n = n \cdot (x + 8) + 1 \end{aligned} \tag{20}$$

$$f_{fiber_{worst}} \approx \Theta(xn) \tag{21}$$

2.4 Сравнение трудоёмкостей алгоритмов

Сравним трудоёмкости алгоритмов обновления документа с использованием DOM и VDOM.

Трудоёмкость алгоритма обновления документа с использованием DOM была вычислена по формуле (6) и не зависит только от двух параметров, являясь

пропорциональной $\Theta(xn)$, где x — трудоёмкость операции update отрисовки узла, а n — количество узлов в новом DOM-дереве.

Трудоёмкость алгоритма обновления документа с использованием VDOM и алгоритма согласования зависит от того, насколько предположения, заложенные в основу эвристики алгоритма согласования, выполняются.

Так, если процесс изменения производится с учётом данных предположений, будет некорректно считать результат работы алгоритма согласования равновероятным, и, следовательно, считать трудоёмкость использующего его алгоритма обновления посредством VDOM через формулу (10). Однако трудоёмкость, вычисленная посредством формулы (9) остаётся верной, поэтому дальнейшие выводы будут сделаны, основываясь на данной формуле.

Чем лучше соблюдаются предположения, положенные в основу алгоритма согласования, чем меньше будет величина t, т. е. количество поддеревьев, требующих полной перерисовки. То есть трудоёмкость алгоритма будет пропорциональна $\Theta(xk)$, где k — количество узлов, требующих перерисовки, $k << n$ в общем случае.

Иными словами, алгоритм обновления документа с использованием VDOM и алгоритма согласования будет иметь меньшую трудоёмкость, чем алгоритм обновления документа с использованием DOM, за счёт понимания того, какие узлы нужно перерисовывать, а какие нет.

Стоит отметить, что при несоблюдении предположений, на которых основывается алгоритм согласование, трудоёмкость алгоритма с использованием VDOM будет превышать трудоёмкость алгоритма, использующего DOM. Такая ситуация произойдёт, например, при постоянном изменении типа корневого элемента.

Эвристика, применимая для алгоритма согласования, работает и в случае алгоритма Fiber. Трудоёмкость алгоритма Fiber в случае изменения типа первого элемента DOM-дерева (корня) может быть вычислена через по формуле 21 и совпадает со значениям, полученными при рассмотрении аналогичных ситу-

аций для алгоритма обновления с использования DOM в формуле 6 и алгорима обновления, использующего согласование. В других случаях, в том числе при добавлении новых элементов в каждый листовой элемент, трудоёмкость алгоритма Fiber будет пропорциональна $\Theta(kx + mn)$, что превышает значение, полученное выше для алгоритма обновления с использованием согласования на значение линейного члена mn. Тем не менее, алгоритм Fiber обеспечивает обновление лишь тех узлов, которые необходимо перерисовывать, как и алгоритм с использованием согласования.

2.5 Сравнение алгоритмов по памяти

Стандарт W3C для объектной модели документа гласит, что управление памятью зависит полностью от реализации [5], поэтому в данной работе уделяется внимание лишь дополнительной памяти, требуемой алгоритмами.

В случае алгоритма обновления документа с использованием VDOM и алгоритм согласования, дополнительной памятью будем считать оперативную память, используемую под хранение виртуальной объектной модели документа. Как и DOM, VDOM хранится в памяти в виде дерева элементов, содержащих всю необходимую для рендера соответствующих виртуальным элемента реальных элементов DOM. В данной работе нет возможности исследовать точное потребление памяти алгоритмами, однако есть возможность сравнить потребление памяти относительно разных алгоритмов.

Использование VDOM может приводить к увеличению объёма дополнительную памяти в 5 раз по сравнению объёмом памяти, используемым в стандартном DOM алгоритме [34]. Тем не менее, данные затраты считаются приемлимыми при использовании таких готовых решений, как React или Vue [35].

В случае алгоритма Fiber, также относящегося к алгоритмам, использующим VDOM, память используется в меньших объёмах, чем при использовании алгоритма согласования [26]. Это связано с тем, что объекты Fiber являют-

ся основными составляющими дерева, характеризующего объектнную модель документа. В этом дереве содержится информация, необходимая для рендера элементов в реальный DOM, в том числе ссылки на элементы реального DOM, соответствующие объектам Fiber. Это позволяет экономить память в случаях, когда необходимую информацию можно найти в объектах DOM [36].

2.6 Сравнение алгоритмов по пользовательскому опыту

Основной недостаток алгоритма обновления документа с использованием DOM заключается в обновлении всех элементов DOM вне зависимости от того, является обновление необходимым или нет. Это приводит к тому, существует граничное значение количества элементов DOM, при превышении которого пользователь сможет заметить ”скачки” из-за уменьшения количества кадров в секунду. Так, в браузере Google Chrome это значение значение условно отмечается как 1400 [20], причём отмечается, что к таким же результатам может привести глубина DOM-дерева, превышающая значение 32.

Использование VDOM и алгоритма согласования решает эту проблему, обновляя лишь необходимые элементы. Тем не менее, при достаточно большом количестве обновляемых элементов данные вычисления будут занимать ощутимое для пользователя время, поскольку будут занимать время, превышающее 16.67 миллисекунд [26]. Это происходит из-за того, что алгоритм обновления документа с использованием VDOM и алгоритм согласования выполняется от начала до конца, блокируя тем самым выполнение других вычислений.

Данную проблему решает алгоритм Fiber. Фаза рендера, во время которой выполняется преимущественный объём вычислений, делится на множество частей при помощи функции requestIdleCallback. Благодаря этому даже при значительном объёме обновляемых элементов сохраняется плавность движения и количество кадров в секунду, большее 60 [26].

Вывод

Были разработаны схемы алгоритмов обновления гипертекстового документа с использованием DOM, VDOM и алгоритма согласования, а также алгоритма Fiber. Были найдены их трудоёмкости и произведено их сравнение по критериям, оописанным в аналитическом разделе.

Трудоёмкость алгоритма обновления документа с использованием DOM пропорциональна $\Theta(xn)$, где x — трудоёмкость операции update отрисовки узла, а n — количество узлов в новом DOM-дереве. В это время трудоёмкость обновления документа с использованием VDOM и алгоритма согласования в случае соблюдения предположений, лежащих в основе алгоритма согласования, пропорциональна $\Theta(xk)$, где k — количество узлов, требующих перерисовки, $k << n$ в общем случае. Трудоёмкость алгоритма Fiber при соблюдении аналогичных эвристических предположений пропроциональна $\Theta(xk + mn)$, где m — количество параметров, изменяемых при обновлении, n — количество узлов в DOM-дереве.

Из перечисленных алгоритмов именно алгоритм Fiber позволяет обеспечивать плавность обновления изображения на экране и при большом размере DOM-дерева, и при большом количестве операций обновления, в то время как алгоритм обновления с использованием алгоритма согласования выполняется от начала до конца и при большом количестве операций обновления может приводить к уменьшению количества кадров в секунду.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DOM Living standart [Электронный ресурс]. — Режим доступа: https://dom.spec.whatwg.org/, свободный (дата обращения: 09.11.2022).

2. HTML Living standart [Электронный ресурс]. — Режим доступа: https://html.spec.whatwg.org/multipage/, свободный (дата обращения: 09.11.2022).

3. Document Object Model (DOM) Level 3 Core Specification [Электронный ресурс]. — Режим доступа: https://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/, свободный (дата обращения: 25.11.2022).

4. Демин И. С. Проблемы развития гипертекстовых сред — М: Вестник ОГУ, 2004. — 79 с.

5. Document Object Model (DOM) [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model, свободный (дата обращения: 09.11.2022).

6. Гипертекст — определение, Большая российская энциклопедия [Электронный ресурс]. — Режим доступа: https://bigenc.ru/technology_and_technique/text/4426247, свободный (дата обращения: 25.11.2022).

7. HTML [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/ru/docs/Web/HTML, свободный (дата обращения: 09.11.2022).

8. Интернет — определение, Большая российская энциклопедия [Электронный ресурс]. — Режим доступа: https://bigenc.ru/technology_and_technique/text/2014701, свободный (дата обращения: 25.11.2022).

9. HTML5 is W3C recommendation [Электронный ресурс]. — Режим доступа: https://www.w3.org/blog/news/archives/4167, свободный (дата обращения: 09.11.2022).

10. World Wide Web Consortium (W3C) [Электронный ресурс]. — Режим доступа: https://www.w3.org/, свободный (дата обращения: 25.11.2022).

11. Яндекс.Радар Браузер в России [Электронный ресурс]. — Режим доступа: https://radar.yandex.ru/browsers, свободный (дата обращения: 09.11.2022).

12. W3C Recommendation 14 December 2010 [Электронный ресурс]. — Режим доступа: w3.org/TR/mwabp/#webapp-defined, свободный (дата обращения: 09.11.2022).

13. Most used web frameworks among developers worldwide, as of 2023 [Электронный ресурс]. — Режим доступа: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/, свободный (дата обращения: 09.11.2022).

14. Сбалансированные деревья, М.В. Губко. — М.: Институт проблем управления РАН [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/sbalansirovannye-derevya/viewer, свободный (дата обращения: 09.11.2022).

15. DOM Element, Web API Reference [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/Element#specifications, свободный (дата обращения: 25.11.2022).

16. Chrome DevTools [Электронный ресурс]. — Режим доступа: https://developer.chrome.com/docs/devtools/, свободный (дата обращения: 25.11.2022).

17. On Layout & Web Performance [Электронный ресурс]. — Режим доступа: https://kellegous.com/j/2013/01/26/layout-performance/, свободный (дата обращения: 11.10.2023).

18. Angular Core API: Renderer [Электронный ресурс]. — Режим доступа: https://angular.io/api/core/Renderer2, свободный (дата обращения: 11.10.2023).

19. Taufan F. I. Comparison between client-side and server-side rendering in the web development. — IOP Conf. Ser.: Mater. Sci., 2020.

20. Избегайте чрезмерного размера DOM [Электронный ресурс]. — Режим доступа: https://web.dev/i18n/ru/dom-size/, свободный (дата обращения: 25.11.2022).

21. Виртуальный DOM и детали его реализации в React [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/docs/faq-internals.html#gatsby-focus-wrapper, свободный (дата обращения: 12.11.2022).

22. Согласование [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/docs/reconciliation.html, свободный (дата обращения: 12.11.2022).

23. Списки и ключи [Электронный ресурс]. — Режим доступа: https://ru.reactjs.org/docs/lists-and-keys.html, свободный (дата обращения: 25.11.2022).

24. Большакова Е. И., Мальковский М. Г., Пильщиков В. Н. Искусственный интеллект. Алгоритмы эвристического поиска (учебное пособие) — М.: Издательский отдел факультета ВМК МГУ (лицензия ИД № 05899 от 24.09.01), 2002. – 83 с.

25. Прокофьев, А. П. FIBER–НОВЫЙ АЛГОРИТМ СРАВНЕНИЯ REACT. — Тольятти, Научный электронный журнал «Инновации. Наука. Образование № 51 (февраль), 2022. — 2149 с.

26. Fedosejev A, Boduch A. React 16 Essentials: A fast-paced, hands-on guide to designing and building scalable and maintainable web apps with React 16. — Birmingham, Packt Publishing Ltd, 2017. — 284 с.

27. requestIdleCallback() Cooperative Scheduling of Background Tasks, W3C 28 June 2022 [Электронный ресурс]. — Режим доступа: https://www.w3.org/TR/2022/WD-requestidlecallback-20220628/, свободный (дата обращения: 17.10.2023).

28. Cooperative Scheduling of Background Tasks W3C First Public Working Draft 17 September 2015 [Электронный ресурс]. — Режим доступа: https://www.w3.org/TR/2015/WD-requestidlecallback-20150917/, свободный (дата обращения: 17.10.2023).

29. Cooperative Scheduling of Background Tasks W3C Proposed Recommendation 10 October 2017 [Электронный ресурс]. — Режим доступа: https://www.w3.org/TR/2017/PR-requestidlecallback-20171010/, свободный (дата обращения: 17.10.2023).

30. MDN Web Docs window.requestIdleCallback() [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/ru/docs/Web/API/Window/requestIdleCallback, свободный (дата обращения: 17.10.2023).

31. MDN Web Docs IdleDeadline: timeRemaining() method [Электронный ресурс]. — Режим доступа: https://developer.mozilla.org/en-US/docs/Web/API/IdleDeadline/timeRemaining, свободный (дата обращения: 17.10.2023).

32. Sauer J, Sonderegger A, Schmutz S. Usability, user experience and accessibility: towards an integrative model. Ergonomics №63, Taylor & Francis, 2020. — 1207 с.

33. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. — М. Наука, Физматлит, 2007. — 376 с.

34. Kris Z., Memory Consumption: the Externality of Programming [Электронный ресурс]. — Режим доступа: https://www.sitepen.com/blog/memory-consumption-the-externality-of-programming, свободный (дата обращения: 19.10.2023).

35. Harris R., Virtual DOM is pure overhead [Электронный ресурс]. — Режим доступа: https://svelte.dev/blog/virtual-dom-is-pure-overhead, свободный (дата обращения: 19.10.2023).

36. Biradar D, An Introduction to React Fiber - The Algorithm Behind React [Электронный ресурс]. — Режим доступа: https://www.velotio.com/engineering-blog/react-fiber-algorithm, свободный (дата обращения: 19.10.2023).