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:

«Анализ алгоритмов построения, обновления и отображения гипертекстового документа при помощи виртуальной объектной модели»

Студент группы **ИУ7-56Б** _____ **П. А. Калашков**
(Подпись, дата) (И.О. Фамилия)

Руководитель _____ **Д. Е. Бекасов**
(Подпись, дата) (И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчётно-пояснительная записка содержит 38 с., 10 рис., 1 табл., 19 ист.

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

Целью работы: анализ алгоритмов построения, обновления и отображения гипертекстового документа при помощи объектной модели и виртуальной объектной модели.

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.

Результаты: алгоритм обновления гипертекстового документа с использованием VDOM и алгоритма согласования имеет меньшую трудоёмкость при соблюдении эвриситки агоритма согласования, за счёт перерисовки только необходимых улов. В противном случае, когда эвристика алгоритма согласования не соблюдена, использование VDOM может иметь большую трудоёмкость, чем просто использование DOM.

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.

DOM — Document Object Model — объектная модель документа.

UI — User Interface — пользовательский интерфейс.

ВВЕДЕНИЕ

Работа с гипертекстовыми документами является неотъемлемой частью жизни каждого человека, пользующегося Всемирной сетью и часто появляется потребность в просмотре различных гипертекстовых документов, а также в выполнении операций, приводящих к их изменению [4]. Возникает вопрос: каким образом стоит отображать документ и производить операции его обновления и построения?

Для взаимодействия с гипертекстовыми документами, входящими в сеть Интернет, существуют программы-браузеры. Преимущественнная часть браузеров использует стандарт [1], обеспечивающий использование объектной модели документа [5].

Целью данной работы является анализ алгоритмов построения, обновления и отображения гипертекстового документа при помощи объектной модели и виртуальной объектной модели. Для достижения поставленной цели необходимо выполнить следующие задачи:

1) изучить принципы работы объектной модели документа;
2) изучить принципы работы виртуальной объектной модели документа;
3) сравнить и проанализировать трудоёмкости алгоритмов обновления документа с использованием объектной модели и виртуальной объектной модели на основе теоретических расчётов.

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 и Safari.

1.2 Объектная модель документа

Объектная модель документа (англ. *Document Object Model* — DOM) [5] — программный интерфейс для HTML, XML и CSV документов. Он обеспечивает структурированное представление документа (дерева), 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.

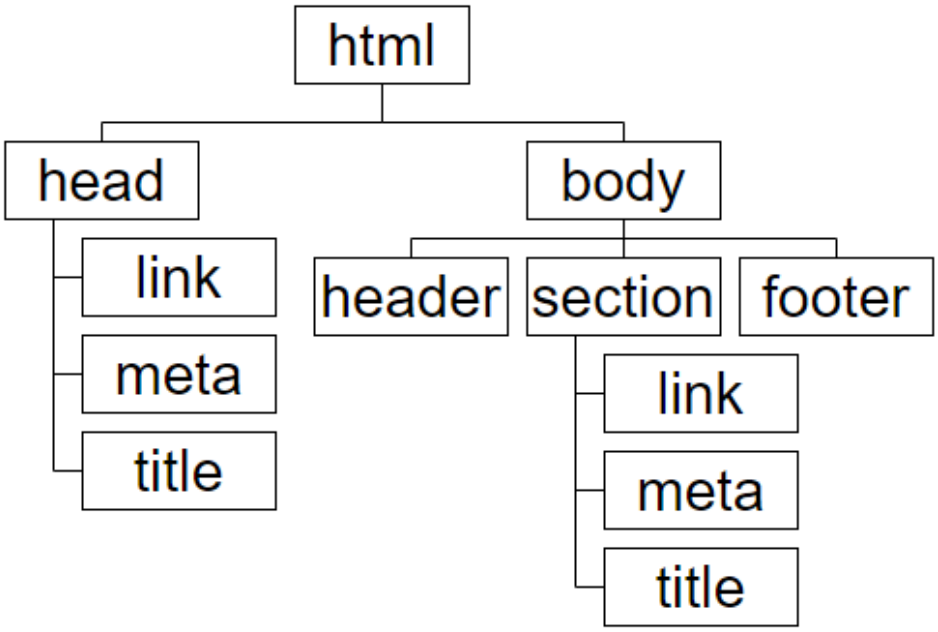

Рисунок 1 – Пример DOM-представления для простого HTML документа

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.

Пример, при котором система была бы более разветвлённая и с большим количеством вложений, представлен на рисунке 3.

На схеме изображено довольно большое DOM-дерево, и его сложно воспринимать из-за его размера. Для удобства часто используется система многоуровневых списков. Пример того, как предыдущее дерево можно преобразовать в такой список, представлен на рисунке 4.

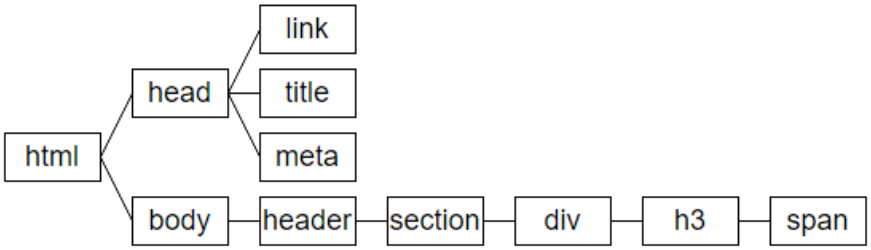

Рисунок 2 – Пример DOM-представления для чуть более сложного HTML документа

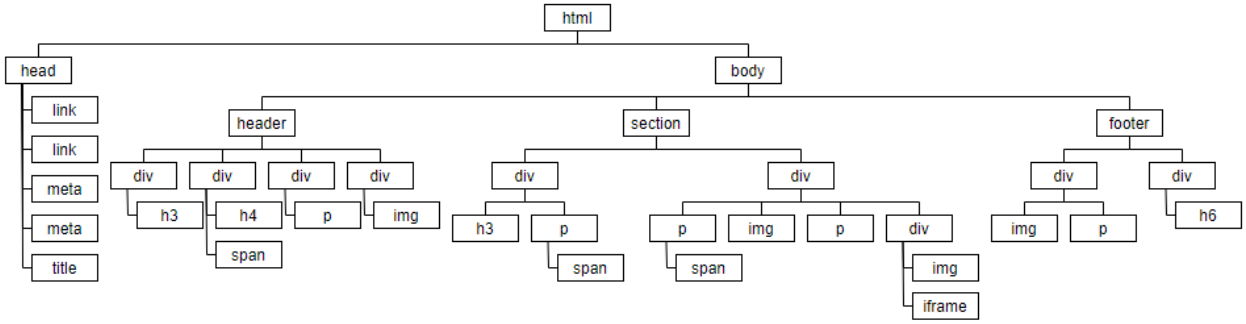

Рисунок 3 – Пример DOM-представления для сложного HTML документа

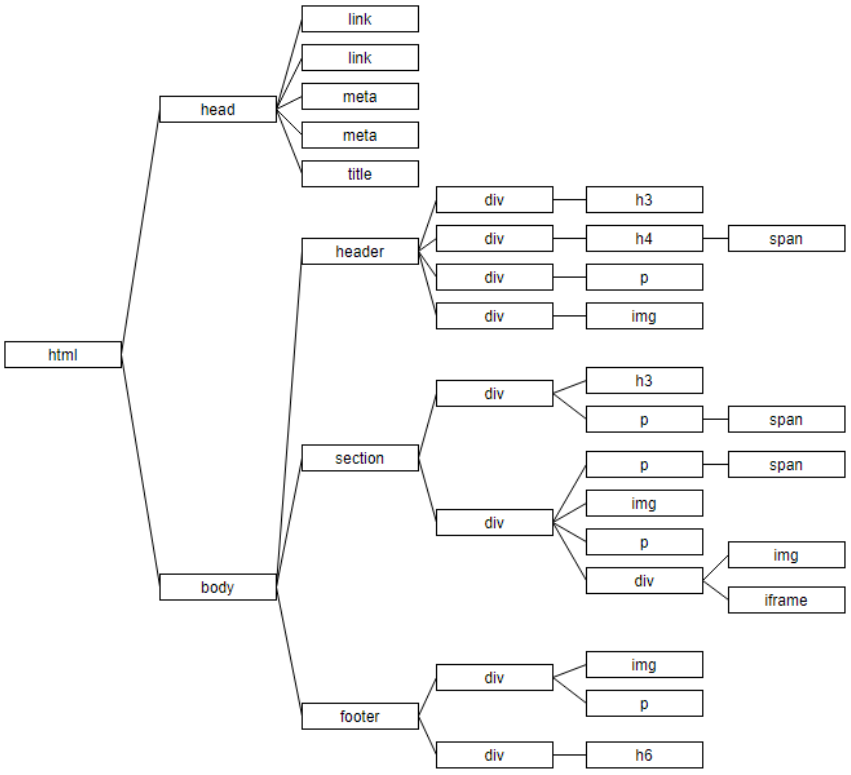

Рисунок 4 – Пример DOM-представления для сложного HTML документа в виде списка

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 страницы. DOM-узлы содержат гораздо больше информации, чем просто данные о дочерних узлах [13]. 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 [14].

1.2.1 Алгоритм построения документа с использованием DOM

Рассмотрим алгоритм построения DOM-дерева по имеющемуся HTML документу.

Для того, чтобы построить дерево объектной модели, требуется обработать документ и произвести операцию вставки в получающееся дерево n раз, где n - количество элементов.

1.2.2 Алгоритм 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).

Рекоммендация W3C в таком случае призывает повторно отрисовать обновлённое дерево с нуля — то есть, каждый раз, когда необходимо обновить дерево, будет использоваться алгоритм отображения.

1.2.3 Алгоритм отображения документа с использованием DOM

Полученное DOM-дерево необходимо отобразить, чтобы пользователь получил возможность увидеть результаты отображения у себя на экране.

Чтобы это сделать, придётся сделать полный обход DOM-дерева и отобразить каждый из имеющихся элементов.

1.3 Виртуальная объектная 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 [15] 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) [16] — 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 5) и VDOM- дерева (рисунок 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 5 – 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 [13], 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 содержится гораздо меньше информации, чем в узлах DOM за счёт того, что нет необходимости хранить данные,

помогающие визуализировать элемент (за это отвечает DOM).

1.3.1 Алгоритм построения документа с использованием VDOM

Рассмотрим, как происходит построение VDOM-дерева по гипертекстовому документу.

Чтобы построить виртуальное дерево, требуется обработать документ и осуществить операцию вставки в виртуальное представление n раз, где n — количество элементов. После этого необходимо сделать полный обход виртуального дерева и для каждого элемента создать соответствующие ему узлы DOM-дерева.

1.3.2 Алгоритм обновления документа с использованием VDOM

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 6 показано виртуальное дерево DOM и процесс сравнения.

Красными кружками обозначены узлы, которые изменились — эти узлы представляют собой элементы пользовательского интерфейса, состояние котоых 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.

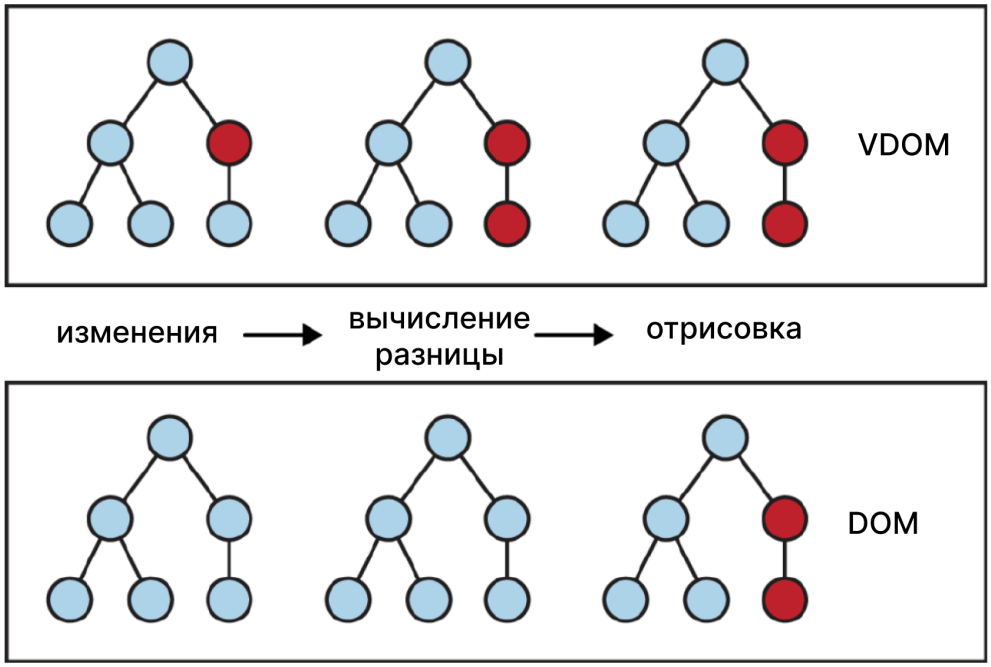

Рисунок 6 – Пример работы алгоритма обновления документа при помощи VDOM

1.3.2.1 Алгоритм согласования

Алгоритм согласования (англ. *reconciliation*) [17] — эвристический алгоритм решения проблемы трансформации одного дерева в другое за минимальное число операций. Он основан на следующих двух предположениях:

Во-первых, два элемента с разыми 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 [18]. 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>
  <li key="2014">Ростов-на-Дону</li>
  <li key="2015">Санкт-Петербург</li>
  <li key="2016">Москва</li>
</ul>
```

Алгоритм согласования является эвристическим [19] 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.

1.3.3 Алгоритм отображения документа с использованием VDOM

Отображение гипертекстового документа с использованием VDOM происходит аналогично алгоритму отображения с использованием DOM, за исключением того, что DOM-дерево получается благодаря VDOM-дереву.

1.4 Сравнение DOM и VDOM

Проведём сравнение объектной модели документа и виртуальной объектной модели документа по следующим критериям:

1) память — количество используемой памяти относительно друг друга;
2) поведение — действия моделей при обновлении элемента;
3) доступ к HTML — может ли изменять HTML напрямую;
4) смысл — что представляет модель.

Таблица 1 – Сравнительная таблица DOM и VDOM

Критерий	DOM	VDOM
Память	Больше, чем у VDOM	Меньше, чем у DOM
Поведение	Обновение DOM-дерева	Обновление изменяемого элемента
Доступ к HTML	Может менять HTML напрямую	Не может менять HTML напрямую
Смысл	Представляет собой UI документа	Является виртуальным представлением DOM

Вывод

В данном разделе были рассмотрены принципы работы с гипертекстовыми документами согласно объектной модели документа (DOM) и витруальной

объектной модели документа (VDOM) и проведено их сравнение. Также был рассмотрен алгоритм согласования, находящий минимальный набор операций, необходимых для преобразования одного дерева в другое. В соответствии с поставленной целью, необходимо разработать алгоритмы обновления документа с использованием DOM и VDOM, а также алгоритма согласования.

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,

2 Конструкторская часть

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

2.1 Разработка алгоритмов

На рисунках 7–10, представлены схемы алгоритмов обновления документа с использованием DOM, VDOM, а также алгоритма согласования соответственно, разработанных на основании приведённых в части 1 описаниях.

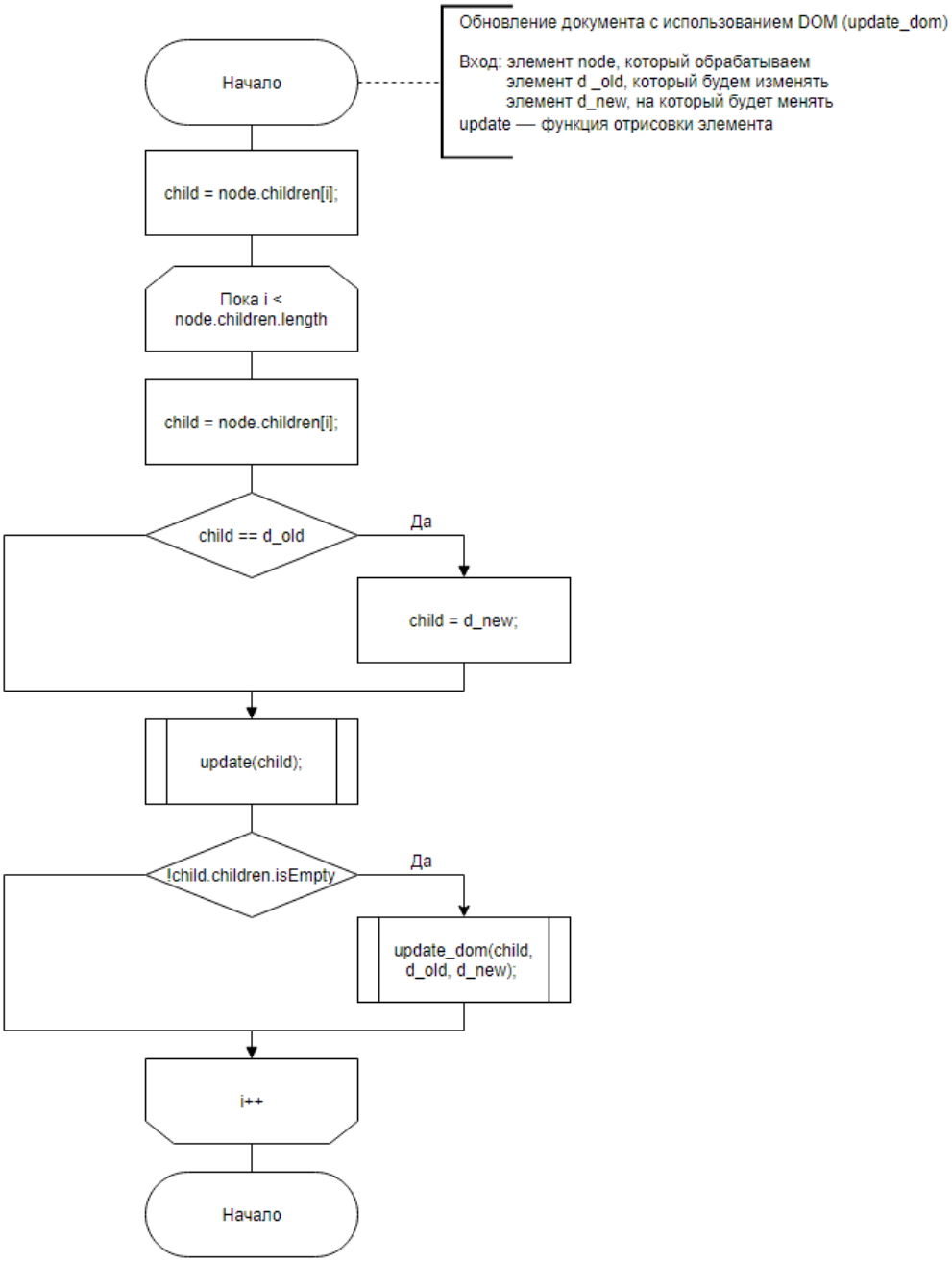

Рисунок 7 – Схема алгоритма обновления документа с использованием DOM

Рисунок 8 – Схема алгоритма обновления документа с использованием VDOM

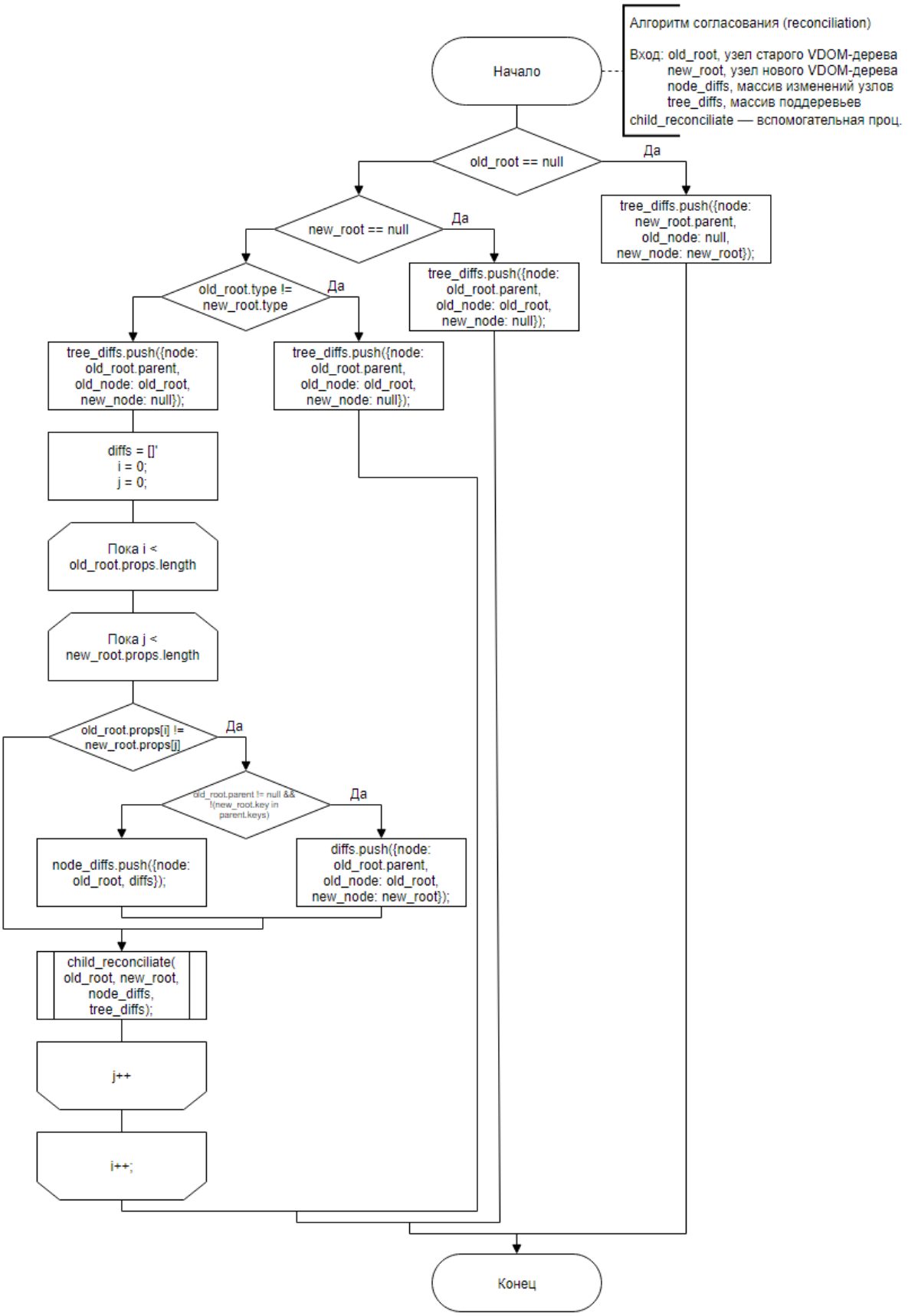

Рисунок 9 – Схема алгоритма обновления документа с использованием DOM

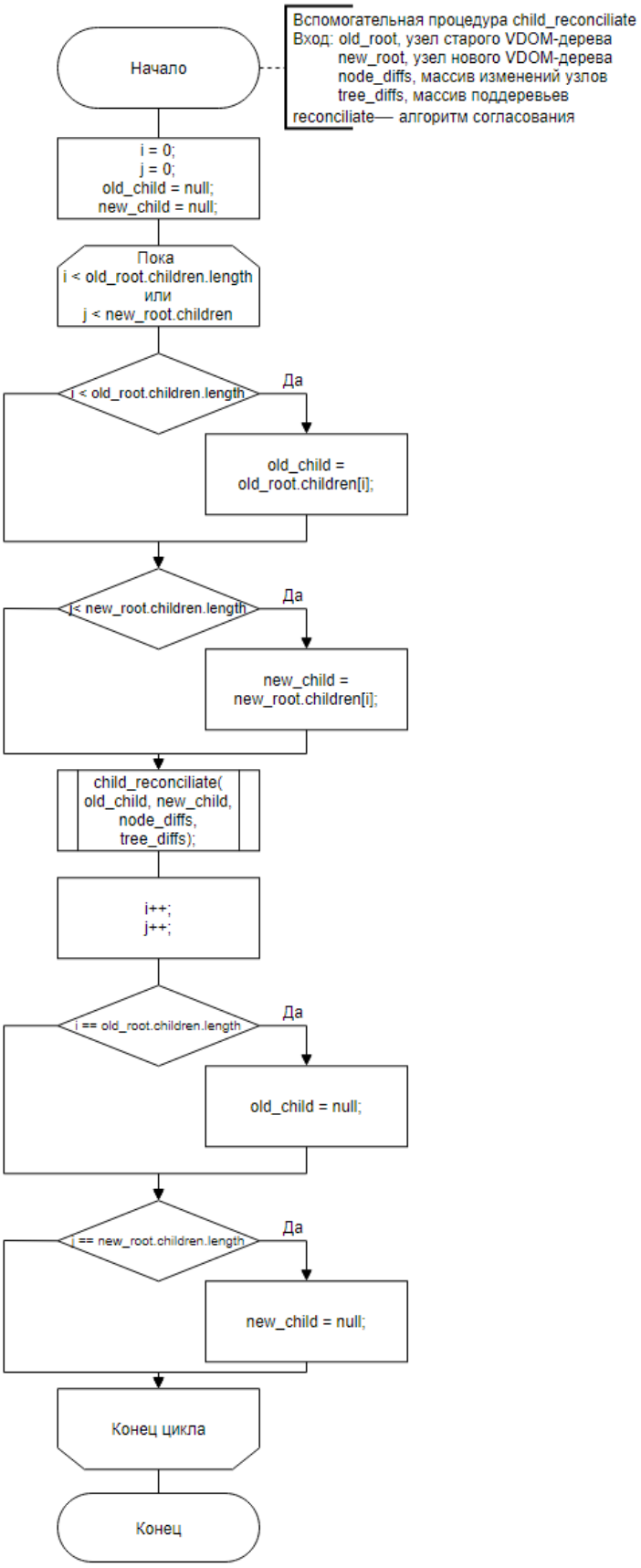

Рисунок 10 – Схема алгоритма обновления документа с использованием DOM

2.2 Модель вычислений для проведения оценки трудоёмкости

Введем модель вычислений [20], которая потребуется для определния трудоёмкости каждого отдельно взятого алгоритма сортировки.

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 схемам алгоритмов.

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 зависимсти 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 зависимсти 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 зависимсти 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 лудший 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 узлов 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).

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

2.4 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, требующиъ 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.

Вывод

Были разработаны схемы алгоритмов обновления гипертекстового документа с использованием DOM, VDOM и алгоритма согласования. Были найдены их трудоёмкости и произведено их сравнение.

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.

Таким образом, алгоритм обновления гипертекстового документа с использованием VDOM и алгоритма согласования имеет меньшую трудоёмкость при соблюдении эвриситки агоритма согласования, за счёт перерисовки только необходимых улов. В противном случае, когда эвристика алгоритма согласования не соблюдена, использование VDOM может иметь большую трудоёмкость, чем просто использование DOM.

ЗАКЛЮЧЕНИЕ

Была достингута цель работы: проанализированы алгоритмы построения, обновления и отображения гипертекстового документа при помощи объектной модели и виртуальной объектной модели. Также в ходе выполнения научно-исследовательской работы были решены следующие задачи:

1) были изучены принципы работы объектной модели документа;
2) были изучены принципы работы виртуальной объектной модели документа;
3) были проведены сравнение и анализ трудоёмкостей алгоритмов обновления документа с использованием объектной модели и виртуальной объектной модели на основе теоретических расчётов.

Исходя из полученных результатов, 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.

Таким образом, алгоритм обновления гипертекстового документа с использованием VDOM и алгоритма согласования имеет меньшую трудоёмкость при соблюдении эвриситки агоритма согласования, за счёт перерисовки только необходимых улов. В противном случае, когда эвристика алгоритма согласования не соблюдена, использование VDOM может иметь большую трудоёмкость, чем просто использование DOM.

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. 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).

13. 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).

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

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

16. 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).

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

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

19. 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.

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