



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 и системы управления»

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

РАСЧЕТНО-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

(Подпись, дата)

П. А. Калашков

(И.О. Фамилия)

Руководитель

(Подпись, дата)

Д. Е. Бекасов

(И.О. Фамилия)

2022 г.

СПИСОК ИСПОЛНИТЕЛЕЙ

Расчетно-пояснительная записка 37 с., 10 рис., 0 табл., X ист., X прил.

КЛЮЧЕВЫЕ СЛОВА

РЕФЕРАТ

Расчетно-пояснительная записка 37 с., 10 рис., 0 табл., X ист., X прил.

КЛЮЧЕВЫЕ СЛОВА

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	10
1 Аналитическая часть	11
1.1 Гипертекстовые документы	11
1.2 Объектная модель документа	11
1.2.1 Алгоритм построения документа с использованием DOM	15
1.2.2 Алгоритм обновления документа с использованием DOM	15
1.2.3 Алгоритм отображения документа с использованием DOM	15
1.3 Виртуальная объектная модель документа	16
1.3.1 Алгоритм построения документа с использованием VDOM	19
1.3.2 Алгоритм обновления документа с использованием VDOM	19
1.3.2.1 Алгоритм согласования	20
1.3.3 Алгоритм отображения документа с использованием VDOM	25
2 Конструкторская часть	26
2.1 Разработка алгоритмов	26
2.2 Модель вычислений для проведения оценки трудоёмкости	31
2.3 Трудоёмкость алгоритмов	31
3 Технологическая часть	32
4 Исследовательская часть	33
ЗАКЛЮЧЕНИЕ	34
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35
ПРИЛОЖЕНИЕ А	37

НОРМАТИВНЫЕ ССЫЛКИ

В настоящем отчете о НИР использованы ссылки на следующие стандарты:

- 1) DOM Living standart [1]
- 2) HTML Living standart [4]

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

HTML — HyperText Markup Language — язык гипертекстовой разметки

DOM — Document Object Model — объектная модель документа

ВВЕДЕНИЕ

Работа с гипертекстовыми документами является неотъемлемой частью жизни каждого человека, пользующегося Всемирной сетью. Часто возникает потребность в просмотре различных гипертекстовых документов, а также в выполнении операций, приводящих к их изменению. 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. Преимущественная часть браузеров использует стандарт [1], обеспечивающий использование объектной модели документа [2].

Целью данной работы является анализ алгоритмов построения, обновления и отображения гипертекстового документа при помощи 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 с использованием объектной модели документа и виртуальной объектной модели документа 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 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) [3], а именно документы, соблюдающие стандарт HTML5 [4].

Данный выбор обусловлен тем, что использование HTML5 получило широкое распространение в Всемирной сети благодаря рекомендации [5] к использованию от Консорциума Всемирной паутины (англ. *World Wide Web Consortium* — W3C). 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) [2] — 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.

Стандарт 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 родитель-

ского элемента. Дочерние элементы не просто находятся внутри родительских, но и наследуют различные свойства от них. Рассмотрим, как выглядит DOM-представление следующего HTML документа.

Листинг 1: Пример простого HTML документа

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4      <link/>
5      <meta/>
6      <title/>
7  </head>
8  <body>
9      <header/>
10     <section>
11         <div/>
12         <p/>
13         <a/>
14     </section>
15     <footer/>
16 </body>
17 </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:

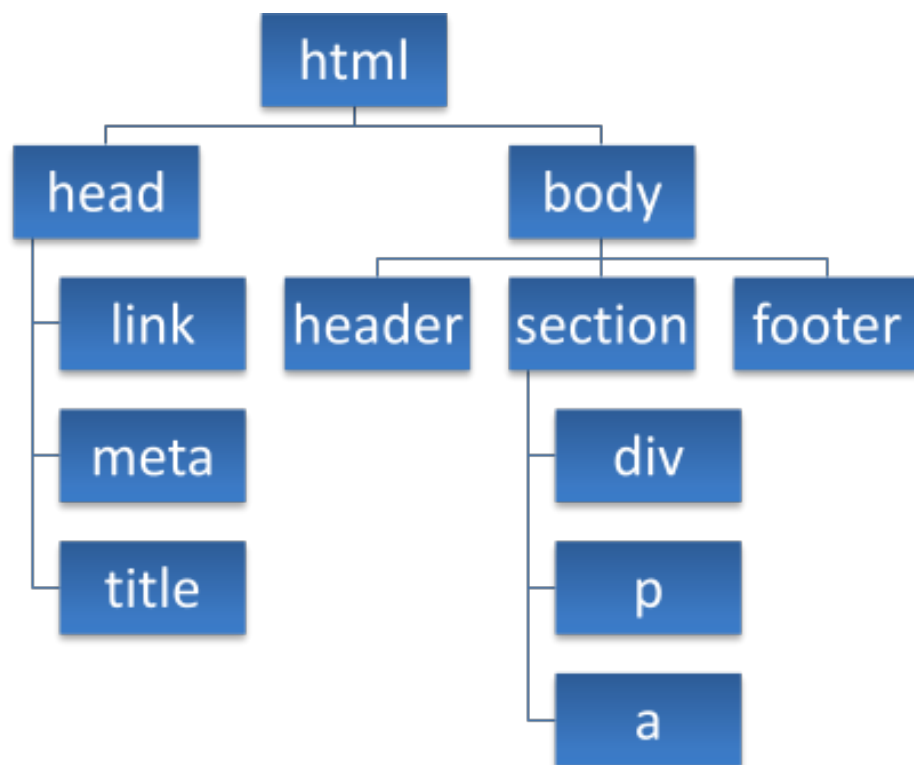


Рисунок 1 – Пример DOM-представления для простого HTML документа

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. Это можно описать так:

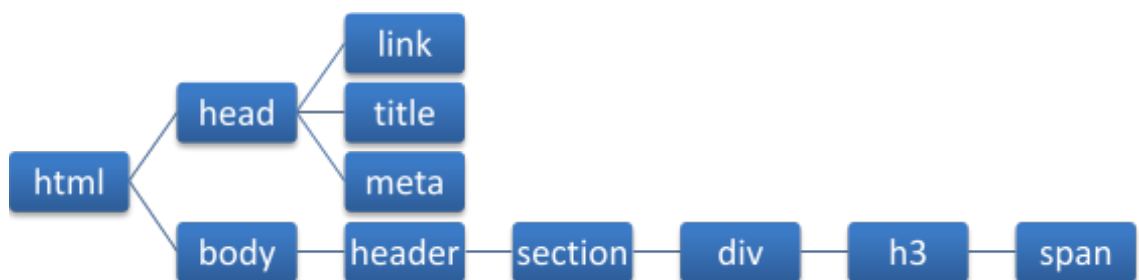


Рисунок 2 – Пример DOM-представления для чуть более сложного HTML документа

А если бы система была бы более разветвлённая и с большим количеством вложений — так:

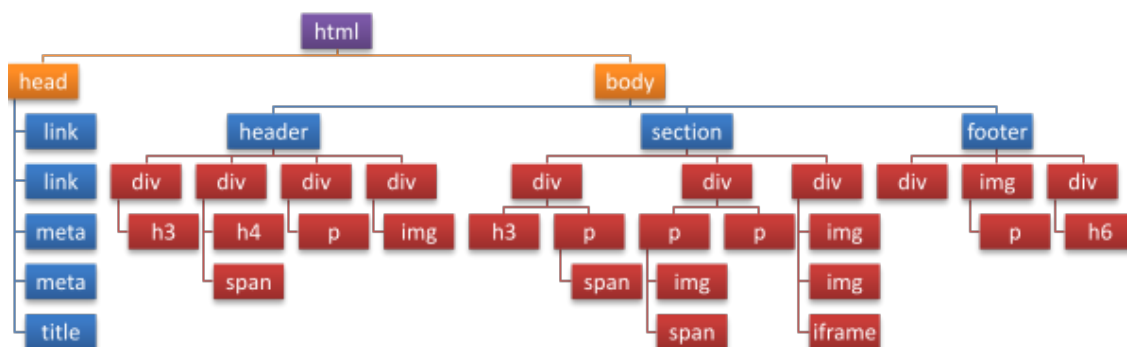


Рисунок 3 – Пример DOM-представления для сложного HTML документа

На схеме изображено довольно большое DOM-дерево, и его сложно воспринимать из-за его размера. Для удобства часто используется система многоуровневых списков. Например, предыдущее дерево можно преобразовать в такой список:

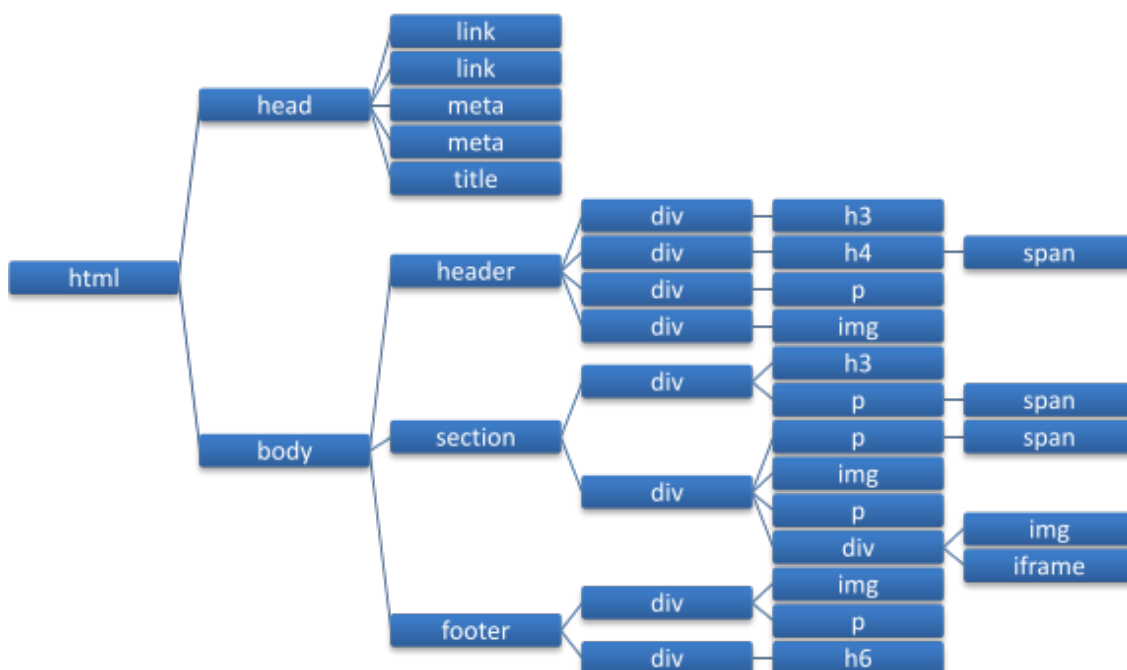


Рисунок 4 – Пример DOM-представления для сложного HTML документа в виде списка

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.

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).

Рекомендация 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 (UI). 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.

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) [8] — концепт программирования, в которой идеальное («виртуальное») представление пользовательского интерфейса хранится в памяти и синхронизируется 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 осуществляется представление DOM- и VDOM- дерева на примере следующего простого HTML документа:

Листинг 2: Простой HTML документ

```
1  <!DOCTYPE HTML>
2  <html lang="en">
3  <head>
4    <link/>
5  </head>
6  <body>
7    <ul class="list">
8      <li class="list__item">List item</li>
9    </ul>
10 </body>
11 </html>
```



Рисунок 5 – DOM-дерево для простого HTML документа

Листинг 3: VDOM-дерево для простого HTML документа

```
1  const vdom = {
2    tagName: "html",
3    children: [
4      { tagName: "head" },
5      {
6        tagName: "body",
7        children: [
8          {
9            tagName: "ul",
10           attributes: { "class": "list" },
11           children: [
12             {
13               tagName: "li",
14               attributes: { "class": "list__item" },
15               textContent: "List item"
16             } // end li
17           ]
18         } // end ul
19       ]
20     } // end body
21   ]
22 } // end html
```

Данный пример служит лишь для демонстрации того, как может осуществляться представление VDOM. Реальные деревья содержат намного большее число узлов, а элементы DOM имеют гораздо больше полей (классы, идентификаторы, стили, обработчики событий, поля данных, типы и значения, вспомогательные поля и т.п.).

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 снижению стоимости обновления реальной модели.

На рисунке 6 показано виртуальное дерево DOM и процесс сравнения:

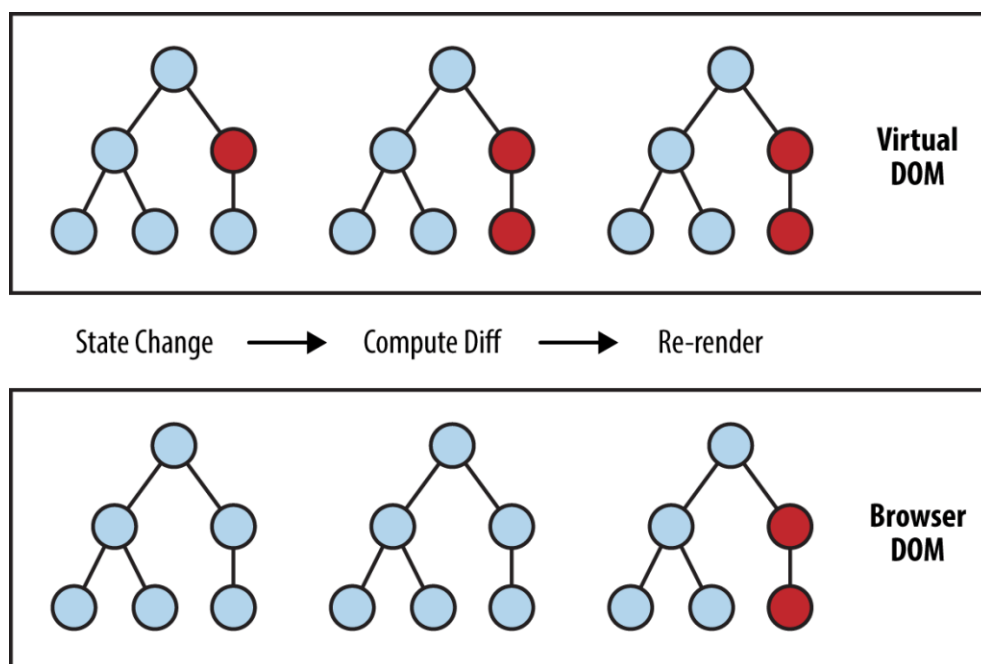


Рисунок 6 – Пример работы алгоритма обновления документа при помощи VDOM

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

1.3.2.1 Алгоритм согласования

Алгоритм согласования (англ. *reconciliation*) [9] — 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 производят 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 `` к `<div>` приведут к полному перестроению дерева, старое DOM-дерево также уничтожается.

Любые элементы, лежащие ниже родительского, будут уничтожены, а их состояние уничтожится.

Рассмотрим 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

```
1  <!-- Старый элемент -->
2  <div>
3    <a href="example.com" />
4  </div>
5
6  <!-- Новый элемент -->
7  <span>
8    <a href="example.com" />
9  </span>
```

При сравнении DOM-элементов одного типа, алгоритм согласования проверяет их атрибуты, сохраняет лежащий в основе элементов DOM-узел и обновляет только изменённые атрибуты. Например:

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

```
1  <!-- Старый элемент -->
2  <div class="before" title="stuff" />
3
4  <!-- Новый элемент -->
5  <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

```
1  <!-- Старый элемент -->
2  <ul>
3      <li>первый</li>
4      <li>второй</li>
5  </ul>
6
7  <!-- Новый элемент -->
8  <ul>
9      <li>первый</li>
10     <li>второй</li>
11     <li>третий</li>
12 </ul>
```

Алгоритм согласования сравнит два дерева `первый`, сравнит два дерева `второй`, а затем вставит дерево `третий`.

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

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

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

```
1  <!-- Старый элемент -->
2  <ul>
3      <li>Санкт-Петербург</li>
4      <li>Москва</li>
5  </ul>
6
7
8  <!-- Новый элемент -->
9  <ul>
10     <li>Ростов-на-Дону</li>
11     <li>Санкт-Петербург</li>
12     <li>Москва</li>
13 </ul>
```

Для решения этой проблемы существуют вспомогательные атрибуты `key` — ключи. 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

```
1  <!-- Старый элемент -->
2  <ul>
3      <li key="2015">Санкт-Петербург</li>
4      <li key="2016">Москва</li>
5  </ul>
6
7  <!-- Новый элемент -->
8  <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 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.

Вывод

В данном разделе были рассмотрены принципы работы с гипертекстовыми документами согласно объектной модели документа (DOM) и виртуальной объектной модели документа (VDOM). Также был рассмотрен алгоритм согласования, находящий минимальный набор операций, необходимых для преобразования одного дерева в другое.

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.

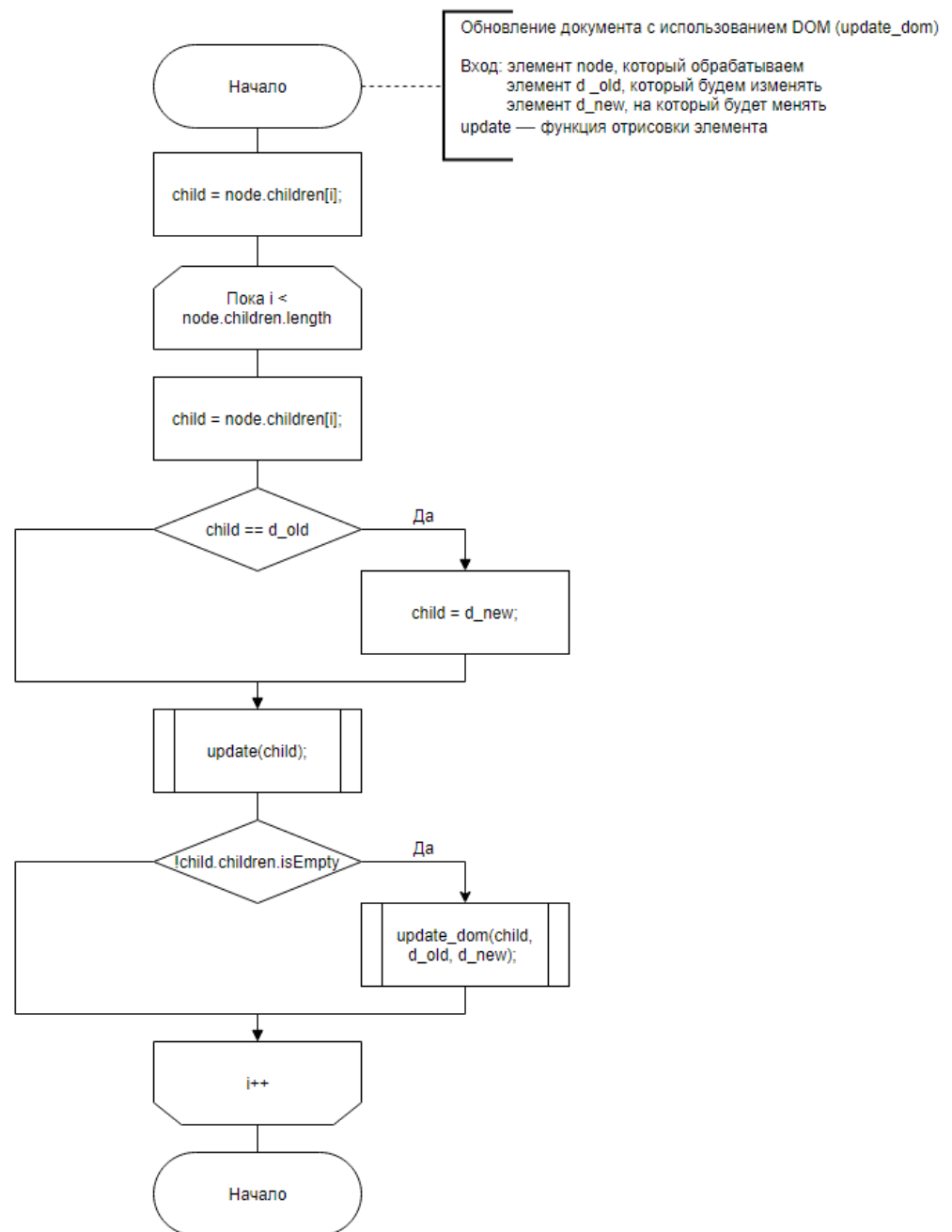


Рисунок 7 – Схема алгоритма обновления документа с использованием DOM

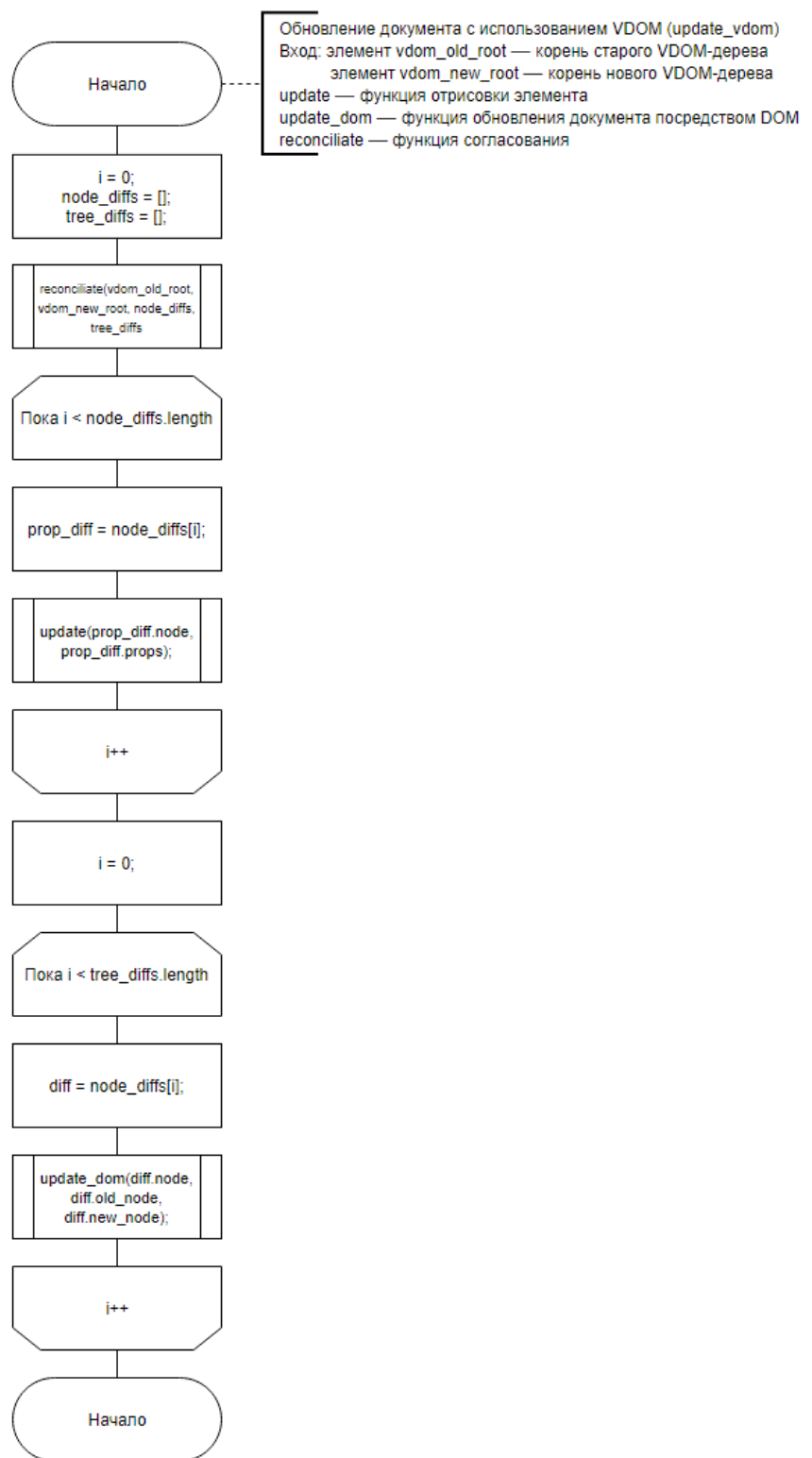


Рисунок 8 – Схема алгоритма обновления документа с использованием VDOM

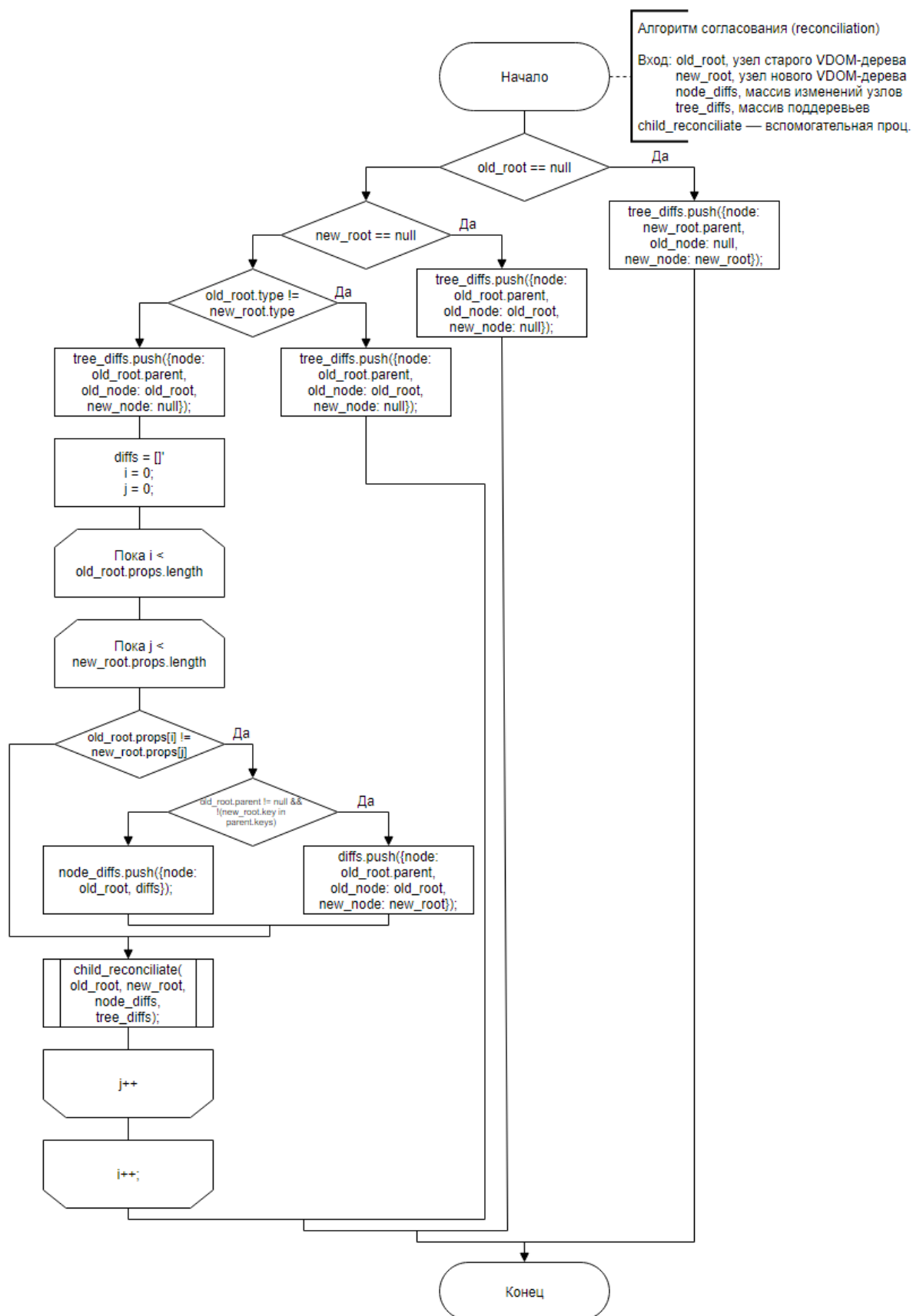


Рисунок 9 – Схема алгоритма обновления документа с использованием DOM

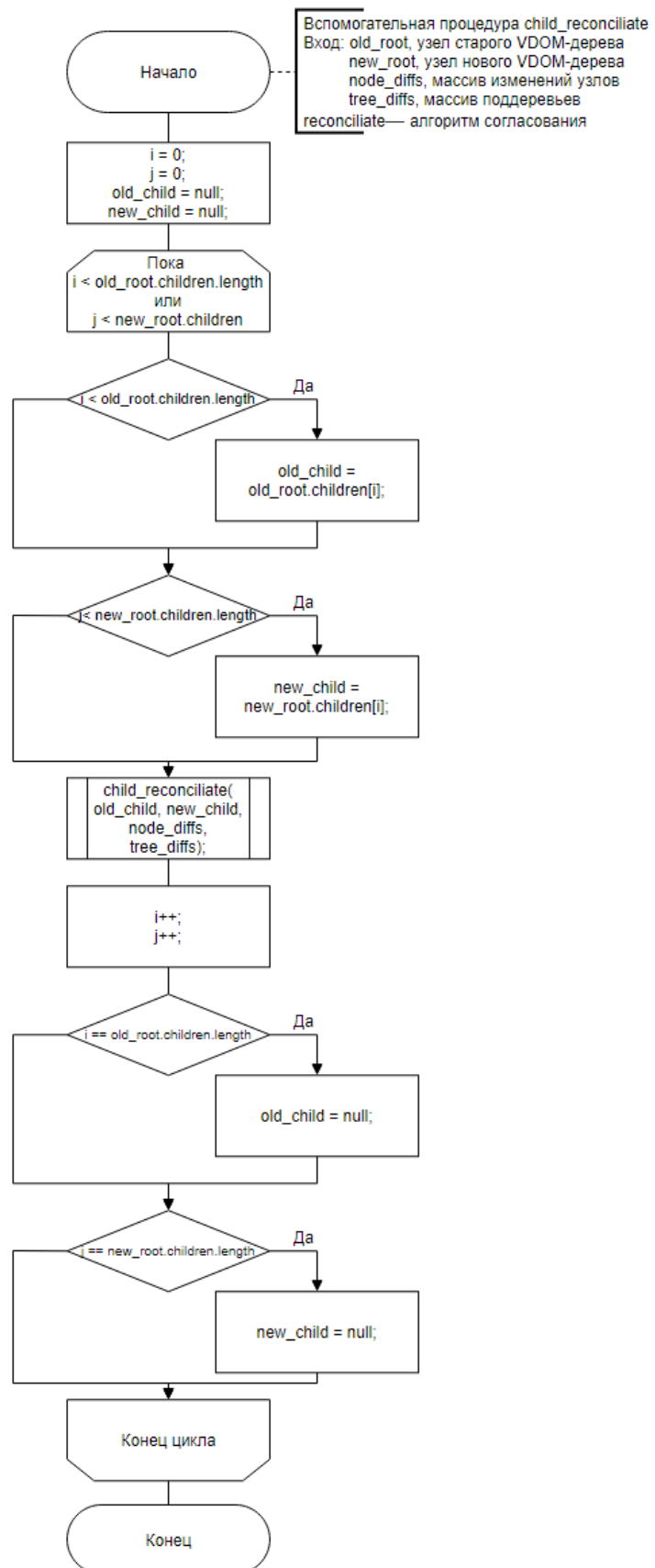


Рисунок 10 – Схема алгоритма обновления документа с использованием DOM

2.2 Модель вычислений для проведения оценки трудоёмкости

Введем модель вычислений [10], которая потребуется для определения 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 как

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

- 3) Трудоёмкость цикла рассчитывается как

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

- 4) Трудоёмкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

3 Технологическая часть

4 Исследовательская часть

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. DOM Living standart [Электронный ресурс]. — Режим доступа: <https://dom.spec.whatwg.org/>, свободный (дата обращения: 09.11.2022)
2. 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)
3. HTML [Электронный ресурс]. — Режим доступа: <https://developer.mozilla.org/ru/docs/Web/HTML>, свободный (дата обращения: 09.11.2022)
4. HTML Living standart [Электронный ресурс]. — Режим доступа: <https://html.spec.whatwg.org/multipage/>, свободный (дата обращения: 09.11.2022)
5. HTML5 is W3C recommendation [Электронный ресурс]. — Режим доступа: <https://www.w3.org/blog/news/archives/4167>, свободный (дата обращения: 09.11.2022)
6. Яндекс.Радар Браузер в России [Электронный ресурс]. — Режим доступа: <https://radar.yandex.ru/browsers>, свободный (дата обращения: 09.11.2022)
7. 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)
8. 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)

9. Согласование [Электронный ресурс]. — Режим доступа: `https://ru.reactjs.org/docs/reconciliation.html`, свободный (дата обращения: 12.11.2022)
10. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. — М. Наука, Физматлит, 2007. — 376 с.

ПРИЛОЖЕНИЕ А