УДК 378.14:004

МЕТОДИКА ОБУЧЕНИЯ СТУДЕНТОВ РАБОТЕ С РЕГУЛЯРНЫМИ ВЫРАЖЕНИЯМИ НА ЗАНЯТИЯХ ПО ПРОГРАММИРОВАНИЮ

Вильданов А.Н.[1]

[1] *ФГБОУ ВО «Башкирский государственный университет», Нефтекамский филиал, Нефтекамск, e-mail: alvild@mail.ru*

Работа посвящена проблеме эффективного овладения учащимися достаточно сложной и обширной темой «Регулярные выражения». Несмотря на то что синтаксис регулярных выражений достаточно простой и понятный, в реальных задачах размер используемых регулярных выражений может показаться начинающему программисту очень большим, запутанным и громоздким и «напугать» его, заставляя избегать использования регулярных выражений в своих программах. Поэтому важно показать применение регулярных выражений сначала для простых задач, а потом и для сложных. В статье сначала представлен подробный конспект вводного занятия, в котором рассматриваются основные элементы и конструкции регулярных выражений. Далее идет описание разработанного тренажера для тренировки навыков написания регулярных выражений. Тренажер способен генерировать многовариантные задания, и поэтому может использоваться преподавателями для тестирования без угрозы списывания. Студенты могут использовать многовариантные задания для тренировки на дому навыков написания регулярных выражений. Тренажер также послужит хорошим инструментом при дистанционном обучении. Результаты, изложенные в статье, могут оказаться полезными для преподавателей, ведущих занятия по дисциплине «Дискретная математика» в теме «Конечные автоматы» или «Языки программирования» в теме «Регулярные выражения» и т.д. Студенты же смогут ознакомиться с важным инструментом обработки строковых выражений и получить первичные навыки написания регулярных выражений.

Ключевые слова: JavaScript, регулярные выражения, конечные автоматы, тренажер, многовариантные задания.

DEVELOPMENT OF AN ONLINE SIMULATOR FOR TRAINING REGULAR EXPRESSION WRITING SKILLS

Vildanov A.N.[1]

[1]*FGBOU VO «Bashkir State University», Neftekamsk branch, Neftekamsk, e-mail: alvild@mail.ru*

The work is devoted to the problem of effective mastering by students of a rather complex and extensive topic «Regular expressions». Despite the fact that the syntax of regular expressions is quite simple and straightforward, in real tasks the size of the regular expressions used may seem very large, confusing and cumbersome to a novice programmer, and «scare» him, forcing him to avoid using regular expressions in his programs. Therefore, it is important to show the use of regular expressions, first for simple tasks, and then for complex ones. This article first provides a detailed outline of an introductory lesson that covers the basic elements and constructs of regular expressions. The following is a description of the developed simulator for training the skills of writing regular expressions. The simulator is capable of generating multivariate assignments, and therefore can be used by teachers for testing without the threat of cheating. Students can use multivariate assignments to practice regex writing skills at home. The simulator will also serve as a good help for distance learning. The results presented in the article may be useful for teachers teaching in the discipline «Discrete mathematics» in the topic «Finite state machines», or «Programming languages» in the topic «Regular expressions», etc. Students will also be able to familiarize themselves with an important string expression processing tool and gain basic skills in writing regular expressions.

Keywords: JavaScript, regular expressions, state machines, simulator, multivariant task.

В рутинной работе программиста широкий круг задач связан с анализом и обработкой текстовых выражений. Например, часто бывает нужно найти какие-либо фрагменты текста, провести с ними какие-либо манипуляции, подсчитать их и т.д. Современные языки программирования уже снабжены методами поиска фрагментов текста. Например, в языке Javascript есть методы indexOf для поиска начального индекса вхождения, метод includes, который проверяет, содержит ли данная строка заданную подстроку, и пр.

Для «сложного» поиска этих методов бывает недостаточно или реализация поиска будет слишком трудоемкой. Тогда на помощь приходят регулярные выражения.

Например, задача поиска всех e-mail на странице без регулярных выражений может выглядеть, например, так:

1. Находим символ @.
2. Начинаем перебирать все символы слева от @, пока не уткнемся в пробел или начало строки.
3. Начинаем перебирать все символы справа от @, пока не уткнемся в пробел или конец предложения и т.д.
4. Переходим к следующему символу @.

Ниже мы покажем, как такая задача решается достаточно просто и быстро с использованием регулярного выражения.

Регулярные выражения представляют собой формальный язык для поиска и произведений манипуляций с подстроками в тексте, основанном на метасимволах. Для описания шаблонов регулярных выражений используются конечные автоматы [1].

Несмотря на то что синтаксис регулярных выражений достаточно простой и понятный, в реальных задачах размер используемых регулярных выражений может показаться начинающему программисту очень большим и громоздким и «напугать» его, заставляя старательно избегать использования регулярных выражений в своих программах. Поэтому важно показать применение регулярных выражений сначала для простых задач, а потом и для сложных, реальных практических заданий.

Для отработки навыков написания регулярных выражений удобно использовать тренажер, разработанный автором. Цель данной работы – описать методический подход к преподаванию регулярных выражений студентам на дисциплинах, связанных с программированием, и ознакомить преподавателей с разработанным тренажером, чтобы они могли использовать его на своих занятиях.

1. Конспект занятия с регулярными выражениями. Рассмотрим примеры написания регулярных выражений на языке JavaScript. Поскольку наш тренажер работает в браузере, удобнее начинать именно с этого языка. Браузер сам же и будет находить правильное решение, не нужен отдельный сервер с установленным ПО для проверки результата.

1. Начнем с задачи поиска первого вхождения слова в тексте, например **«слон»**. Для наглядности представим, что у нас есть предложение, например:

Слон со слоненком пошли на водопой.

Регулярное выражение можно создать следующим способом:

```
var re = /слон/;
```

Теперь учащимся можно объяснить, что именно найдет команда поиска с таким регулярным выражением. Можно использовать любой способ выделения текста, например:

Слон со слоненком заслонили солнце.

Такой поиск находит только одно, первое вхождение, причем первое слово **«Слон»** - игнорируется. И действительно, ведь, строго говоря, **«Слон»** и **«слон»** - разные тексты. Далее можно сказать, что для расширенного поиска используются флаги.

2. Найдем теперь все вхождения слова **«слон».** Используется флаг **g** (от слова **global**):

var re = /слон/g;

Результат поиска:

Слон со слоненком заслонили солнце.

3. Наконец, скажем, что для поиска без учета регистра используется флаг **i**:

var re = /слон/gi;

Результат поиска:

Слон со слоненком заслонили солнце.

4. Следующим шагом рассмотрим поиск двух и более выражений. Например, нам нужно найти все вхождения двух слов - **«слон»** и **«жираф»**. Тогда используется скобка |:

var re = /слон|жираф/gi;

Результат поиска:

Слон и жираф слонялись по саванне.

Еще пример:

var re = /слон|жираф|тигр/gi;

Результат поиска:

Слон и жираф слонялись по саванне и встретили тигра.

5. Теперь рассмотрим похожую ситуацию, когда мы ищем не несколько слов, а любой из символов, например буквы «л» и «п». Тогда их достаточно разместить в квадратных скобках (без скобки |):

var re = /[лп]/gi;

Результат поиска:

Слон и жираф слонялись по саванне.

6. Если между символами поставить дефис, то будет искаться целый диапазон:

var re = /[л-п]/gi;

Результат поиска (искомые буквы **«л»**, **«м»**, **«н»**, **«о»**):

Слон и жираф слонялись по саванне.

Аналогично работает поиск с цифрами:

var re = /[5-7]/gi;

Результат поиска (искомые цифры **«5»**, **«6»**, **«7»**):

Северная семилетняя война шла с 1563 по 1570 г.

7. Теперь пришла пора рассказать про метасимволы – это, например, **\d**, **\s**. Метасимволы являются символами с особым значением. Метасимвол **\d** означает любую цифру (**«d»** от английского **«digit»** означает «цифра»). Тогда, например, аналогичны друг другу следующие регулярные выражения:

```
var re1 = /[0-9]/gi;
var re2 = /\d/gi;
```

Например, поиск всех дат (годов) в тексте (состоящих из четырёх цифр) может иметь вид:

```
var re = /[\d\d\d\d]/gi;
```

Результат поиска:

Северная семилетняя война шла с 1563 по 1570 г.

Метасимвол **\s** служит для обозначения пробела (**«s»** от английского **«space»** – «пробел»).

```
var re = /он\s/gi;
```

Результат поиска:

Слон и Наполеон заканчиваются на ~~он~~.

Важно подчеркнуть, что самое последнее вхождение не ищется, поскольку оно заканчивается не на пробел, а на точку.

8. Метасимвол **\b** служит для обозначения границы слова. Например, найдем все вхождения слога **«сл»** в начале слова:

```
var re = /\bсл/gi;
```

Результат поиска:

Слон со слоненком заслонили слабое солнце.

В слове **«заслонили»** слог **«сл»** не ищется, так как он не находится в начале слова. Начало слова компилятор языка определяет самостоятельно, основываясь на простых правилах. Далее найдем все вхождения словосочетания **«он»** в конце слова:

```
var re = /он\b/gi;
```

Результат:

Слон и Наполеон заканчиваются на он.

К сожалению, для русских букв на языке Javascript этот метасимвол не работает. Поэтому в тренажере предлагаются примеры с английскими словами или с транслитом.

9. Кроме того, точка означает специальный символьный класс, который соответствует «любому символу, кроме новой строки». Например,

var re = /й.к/gi;

Результат:

Лейка, лей капель, на войско налей-ка.

10. Теперь предположим, что нужно найти все словосочетания типа **«Ура»**, **«ураа»**, **«урааа»** и т.д. Для этой задачи можно использовать так называемые квантификаторы: **+,*,?**

n+ строка содержит по крайней мере один **n**;

n* строка содержит 0 или более вхождений **n**;

n? строка содержит 0 или 1 вхождение **n**.

Для решения этой задачи можно использовать квантификатор +:

var re = /ура+/gi;

Результат:

Балагур кричал ура ураа урааа ураааа!

Если задать регулярное выражение как выражения в виде

var re = /ура*/gi;

результат будет немного другой:

Балагур кричал ура ураа урааа ураааа!

Если же задать регулярное выражение как выражения в виде

var re = /ура?/gi;

результат будет следующий:

Балагур кричал ура ураа урааа ураааа!

11. Теперь нужно сказать несколько слов об экранировании. Некоторые символы несут различные функции – это [,\,|,?,*,+,. Что, если нужно найти один из этих символов? Тогда их нужно экранировать с помощью слэша \. Например, найдем все многоточия:

var re = /\.\.\./gi;

и результат будет следующий:

Математика... Формулы... Синусы... Тангенсы...!

12. Разобранный аппарат регулярных выражений уже достаточен для решения кейсов. Первая задача – поиск всех слов (на русском языке) в тексте. Сначала можно предложить учащимся попробовать самим сформулировать решение, а потом уже дать правильный ответ:

var re = /[а-яё]+/gi;

Нужно иметь в виду, что буквы **«ё»** нет в списке **«а-я»** из-за особенности кодировки русских символов. Поэтому она добавляется в регулярное выражение.

Здесь следует отметить, что раньше, до этого примера, мы фактически искали не «отдельные» слова, а фрагменты текста, включающие подстроки внутри других слов. В данном же примере ищутся именно отдельные слова. Подобный подход будет использоваться

во многих кейсовых заданиях ниже.

13. Следующее задание – найти все слова, заканчивающиеся на **«r»**:

var re = /[a-z]+r\b/gi;

14. Теперь мы можем написать регулярное выражение, с помощью которого можно найти все почтовые адреса e-mail в тексте:

var re = /[a-z]+@[a-z]+\.[a-z]+/gi;

Будут находиться все почтовые адреса, не содержащие цифр. При желании их поиск можно тоже добавить. Как видим, найти все почтовые адреса e-mail в тексте не так уж и сложно.

Эту задачу (поиска e-mail адресов в тексте) не следует путать с более сложной задачей определения валидности введенного e-mail (там уже код действительно очень громоздкий).

2. Тренажер для тренировки навыков написания регулярных выражений. Для отработки владения регулярными выражениями разработан онлайн-тренажер (рис. 1). Тренажер доступен в сети Интернет и написан на языке Javascript с использованием HTML, css. Для его использования не нужно устанавливать дополнительное программное обеспечение, достаточно современного браузера.

Тренажер устроен следующим образом. Учащийся видит задание и специальное поле для ввода ответа, с подсказкой формы ответа (рис. 1). Он заполняет это поле, и нажимает «Проверить». В зависимости от того, правилен или нет ответ, учащийся получает сообщение «Верно» или «Неверно».

С помощью cookies на языке JavaScript реализовано сохранение в браузере как номеров правильно выполненных заданий, так и ответов на них (рис. 1). Это поможет, например, в такой нештатной ситуации, когда студент случайно закроет браузер. Он сможет открыть заново тест по ссылке и продолжить прохождение теста.

Также учащийся может закрыть браузер и продолжить решение на следующий день, если он тренируется дома. Преподаватель визуально может оценить количество выполненных заданий.

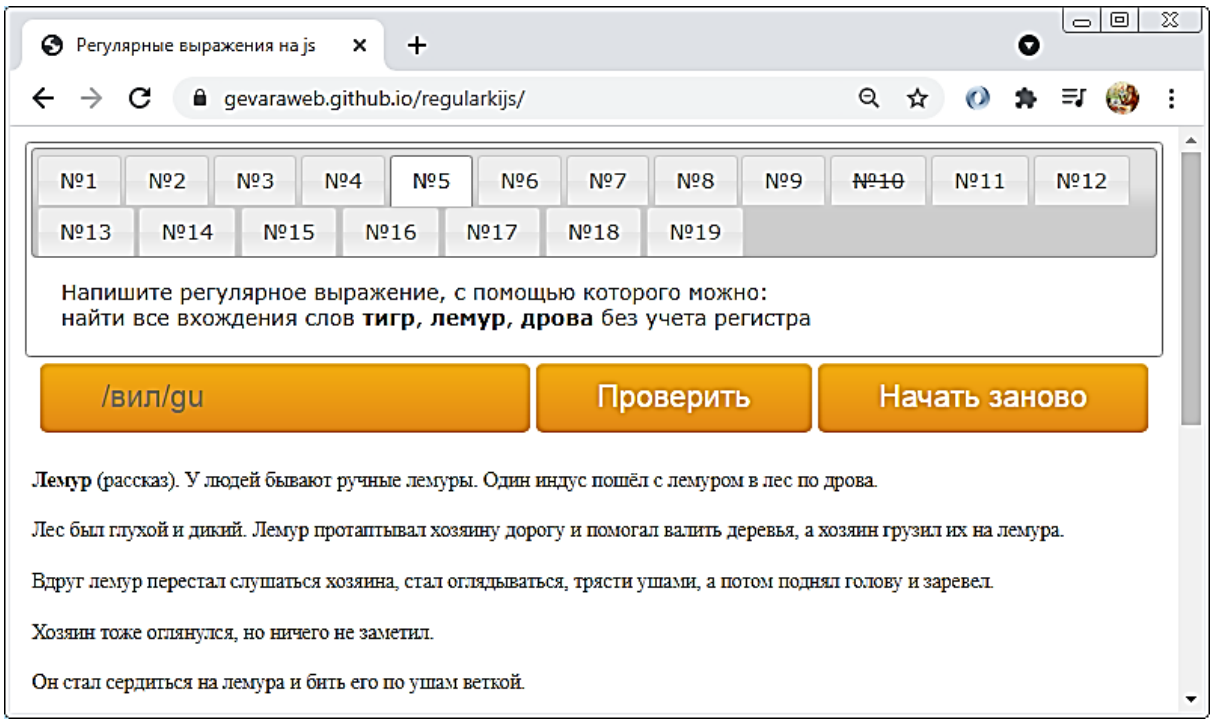

Рис. 1. Пример задания на знание регулярных выражений

Тренажер способен генерировать разные задачи. Понятно, что многовариантные задания являются одним из наиболее эффективных способов защиты от списывания [2]. Наш тренажер содержит генератор случайных заданий, поэтому другой учащийся увидит немного другое задание (рис. 2).

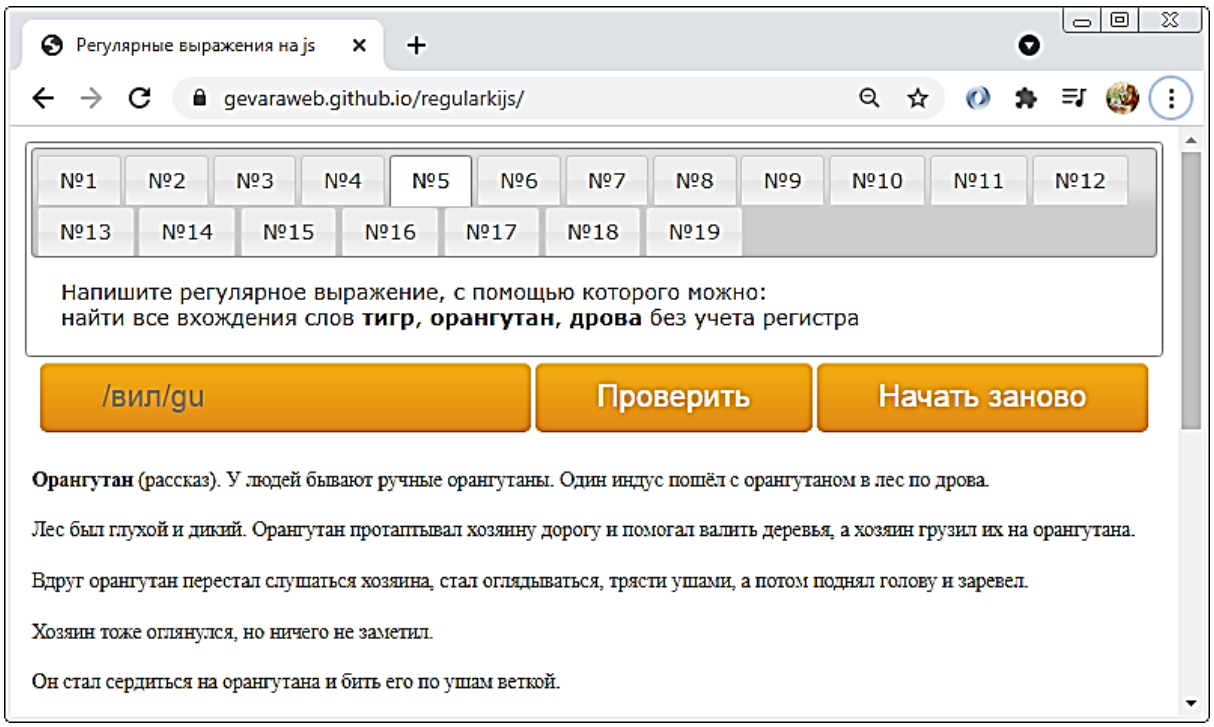

Рис. 2. Пример другого варианта

Также учащийся, при желании, может нажать кнопку «Начать заново» и пройти тест снова, с измененными заданиями.

Особенность тренажера еще и в том, что он выделяет цветом найденные по маске учащегося вхождения текста (рис. 3). Это поможет, во-первых, наглядно продемонстрировать учащемуся результат работы регулярного выражения. Во-вторых, если он допустил небольшую ошибку, скорректировать или уточнить свой ответ до правильного.

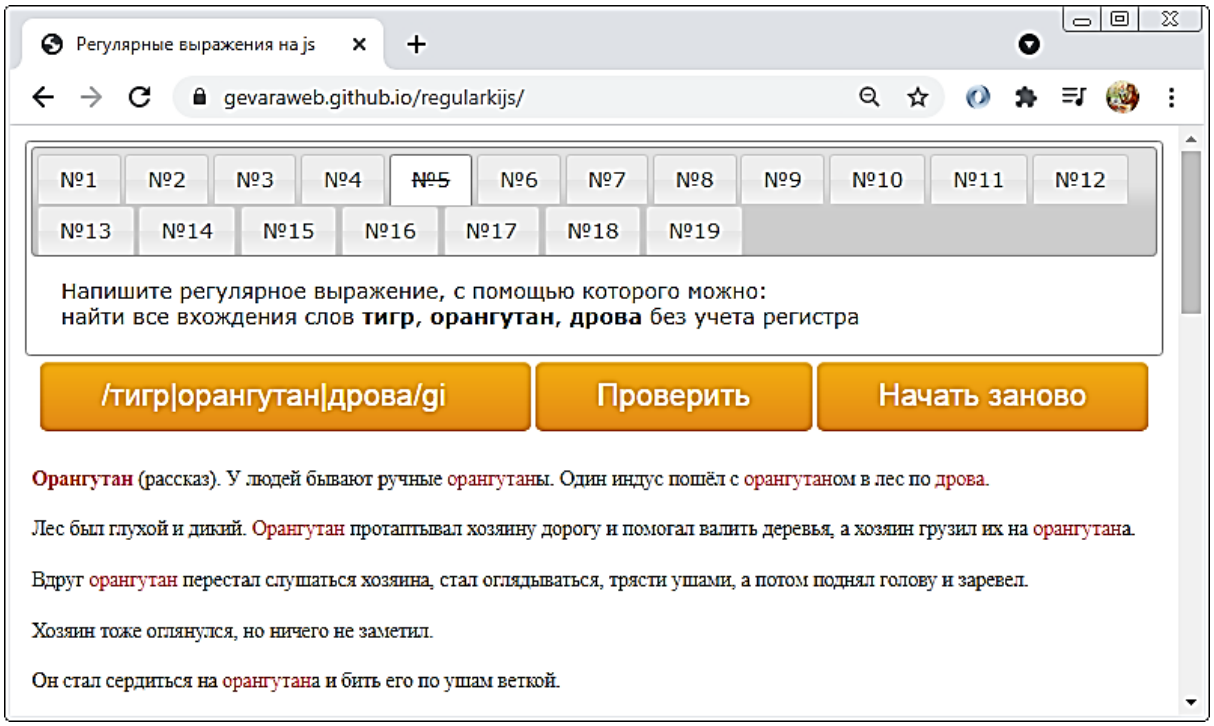

Рис. 3. Пример решенного задания

После тренировки на базовых регулярных выражениях, учащийся решает кейсовые задания (рис. 4).

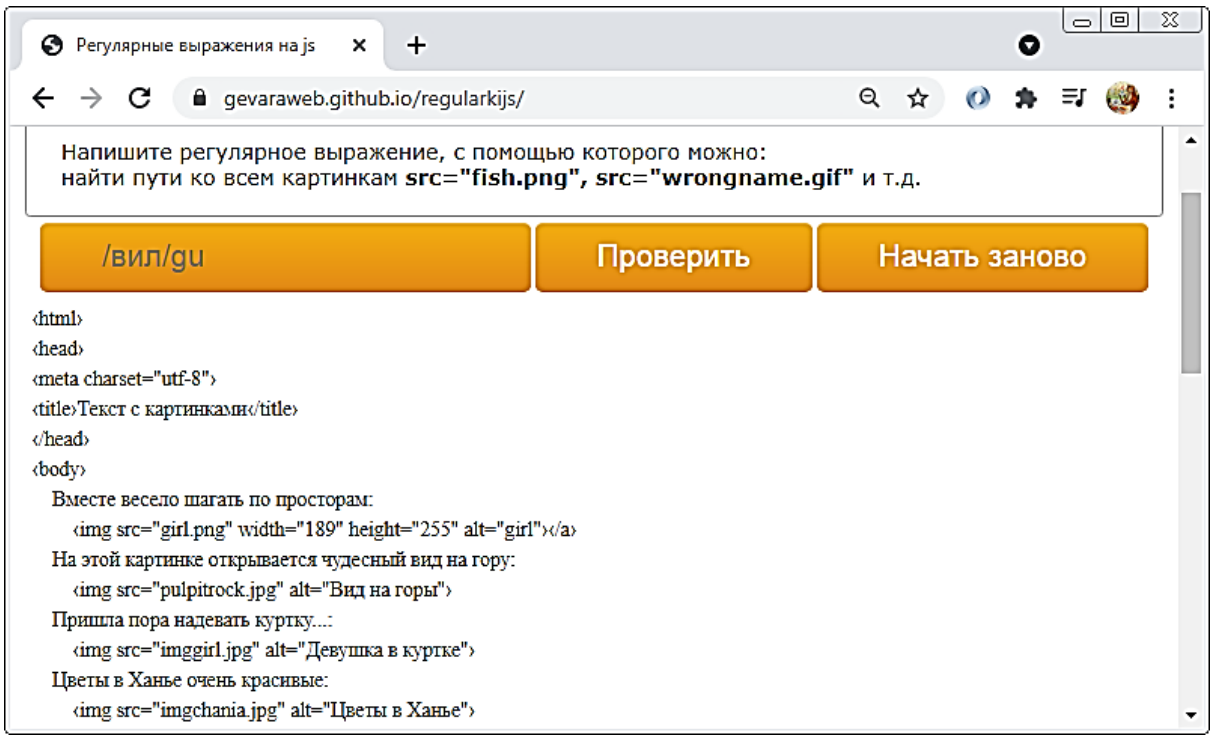

Рис. 4. Пример кейсовой задачи

Заключение. Регулярные выражения позволяют с помощью сравнительно небольшого по объему кода эффективно решать большое количество реальных задач, связанных с обработкой строк, таких, как, например, поиск всех e-mail на веб-странице, поиск всех ссылок и т.д.

Регулярные выражения находят широкое применение на практике. Сведения из теории автоматов и вообще языков регулярных выражений могут пригодиться любому программисту, иногда в самом неожиданном месте [3]. Например, регулярные выражения используются для автоматизации процесса сбора, сортировки и хранения нормативно-правовых актов [4], для автоматизация процесса метаразметки архивных документов [5], для обнаружения рекламы на сайтах [6], для разбора математических формул [7] и т.д.

Ознакомиться с разработанным тренажером и проверить свои навыки написания регулярных выражений можно на сайте GitHub: https://gevaraweb.github.io/regularkijs/. В дальнейшем планируется добавить в тренажер новые примеры, в первую очередь – новые кейсовые задания.

Материал статьи будет полезен для преподавателей, ведущих занятия по дисциплине «Дискретная математика», для самостоятельной работы учащихся по теме «Конечные автоматы», или же «Языки программирования», по теме «Регулярные выражения» и т.д. Студенты же смогут ознакомиться с важным инструментом обработки строковых выражений и получить первичные навыки написания регулярных выражений.

Список литературы

1. Алькаев Р.Р. Недетерминированные и детерминированные конечные автоматы в регулярных выражениях // Современные информационные технологии в образовании, науке и промышленности: сборник трудов XV Международной конференции, XIII Международного конкурса научных и научно-методических работ (Москва, 14–15 февраля 2020 года). М.: ООО «Издательство "Экон-Информ"», 2020. С. 5-10.
2. Вильданов А.Н. О генерации многовариантных задач по теории игр // Достижения и приложения современной информатики, математики и физики: материалы VI Всероссийской научно-практической заочной конференции (г. Нефтекамск, 01 ноября 2017 года). Нефтекамск: Башкирский государственный университет, 2017. С. 87-94.
3. Беляев В.Я. Дискретная математика. Грамматики и автоматы: учебное пособие. Мурманск: Мурманский арктический государственный университет, 2019. 104 с.
4. Латышев И.П., Лукьянченков И.Ю., Макеева И.Ю., Козлов А.Р., Климцов А.С., Лемза И.А. Программа синтаксического анализа нормативно-правовых актов // Свидетельство о регистрации программы для ЭВМ № 2021610674; заявл. 22.01.2021; опубл. 01.02.2021.
5. Филимонов Д.Ю., Светлов А.В., Горбань О.А., Косова М.В., Шептухина Е.М. Автоматизация процесса метаразметки архивных документов // Математическая физика и компьютерное моделирование. 2020. Т. 23. № 4. С. 56-68. DOI 10.15688/mpcm.jvolsu.2020.4.6.
6. Рианьо Д., Пиньон Р., Молеро-Кастильо Г., Барсенас Э., Веласкес-Мена А. Регулярные выражения для обнаружения Web-рекламы на основе автоматического скользящего алгоритма // Труды Института системного программирования РАН. 2021. Т. 33. № 2. С. 65-76. DOI 10.15514/ISPRAS-2020-33(2)-3.
7. Гусенков А.М., Биряльцев Е.В., Жибрик О.Н. GateMorphFormula: плагин разбора математических формул // Свидетельство о государственной регистрации программы для ЭВМ № 2020610079; заявл. 20.12.2019; опубл. 09.01.2020. Заявитель федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет» (ФГАОУ ВО КФУ).