|  |  |
| --- | --- |
|  | ы |
| МИНОБРНАУКИ РОССИИ |  |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |  |

**Институт информационных технологий**

КАФЕДРА ИНСТРУМЕТНАЛЬНОГО И ПРИКЛАДНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (ИППО)

пРАКТИЧЕСКАЯ РАБОТА № 2

по дисциплине «Теория автоматов и формальных языков»

«РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ»

Выполнил студент группы ИКБО-12-18 Копотов М.А.

Принял Алпатов А.Н.

Практическая работа выполнена «29» сентября 2020г.

«Зачтено» «\_\_\_»\_\_\_\_\_\_\_2020г.

Москва 2020

**Оглавление**

[Цель практической работы 3](#_Toc52228313)

[Теоретические сведения 3](#_Toc52228314)

[Порядок выполнения работы 4](#_Toc52228315)

[Вывод 4](#_Toc52228316)

# Цель практической работы

Целью данной практической работы является ознакомления с конечными автоматами, и построение на их базе алгоритмов работы с регулярными выражениями.

# Теоретические сведения

1. Конечный автомат

Коне́чный автома́т — абстрактный автомат, число возможных внутренних состояний которого конечно.

Существуют различные способы задания алгоритма функционирования конечного автомата. Например, конечный автомат может быть задан в виде упорядоченной пятерки элементов некоторых множеств:

где

— входной алфавит (конечное множество входных символов), из

которого формируются входные слова, воспринимаемые конечным автоматом;

— множество внутренних состояний;

— начальное состояние

— множество заключительных, или конечных состояний

— функция переходов, определенная как отображение, такое, что, то есть значение функции переходов на упорядоченной паре (состояние, входной символ или пустая цепочка) есть множество всех состояний, в которые из данного состояния возможен переход по данному входному символу или пустой цепочке (ε).

Принято полагать, что конечный автомат начинает работу в состоянии, последовательно считывая по одному символу входного слова (цепочки входных символов). Считанный символ переводит автомат в новое состояние

в соответствии с функцией переходов.

Читая входную цепочку символов и делая переходы из состояния в

состояние, автомат после прочтения последнего символа входного слова

окажется в некотором состоянии.

Если это состояние является заключительным, то говорят, что автомат

допустил слово.

Конечные автоматы широко используются на практике, например, в

синтаксических и лексических анализаторах, тестировании программного обеспечения на основе моделей.

2. Регулярные выражения

Регуля́рные выраже́ния (англ. regular expressions) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters). Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

Обычные символы (литералы) и специальные символы (метасимволы)

Большинство символов в регулярном выражении представляют сами себя за исключением специальных символов [ ] \ / ^ $ . | ? \* + ( ) { }, которые могут быть экранированы символом \ (обратная косая черта) для представления самих себя в качестве символов текста. Можно экранировать целую последовательность символов, заключив её между \Q и \E.

Аналогично могут быть представлены другие специальные символы (набор символов, требующих экранирования, может отличаться в зависимости от конкретной реализации). Часть символов, которые в той или иной реализации не требуют экранирования (например, угловые скобки < >), могут быть экранированы из соображений удобочитаемости.

Любой символ

Метасимвол . (точка) означает один любой символ, но в некоторых реализациях исключая символ новой строки.

Вместо символа . можно использовать [\s\S] (все пробельные и непробельные символы, включая символ новой строки).

Символьные классы (наборы символов) Набор символов в квадратных скобках [ ] именуется символьным классом и позволяет указать интерпретатору регулярных выражений, что на данном месте в строке может стоять один из перечисленных символов. В частности, [абв] задаёт возможность появления в тексте одного из трёх указанных символов, а [1234567890] задаёт соответствие одной из цифр. Возможно указание диапазонов символов: например, [А-Яа-я] соответствует всем буквам русского алфавита, за исключением букв «Ё» и «ё»

# Порядок выполнения работы

# Задание 2. Поиск заглавных букв в тексте

# *Листинг 1. класс Main*

|  |
| --- |
| package com.company;  import java.util.Arrays;  import java.util.Collections;  import java.util.List;  import java.util.Set;  public class Main {  public static void main(String[] args) {  final int START = 0;  final int UPPER = 1;  String str = "QWERTYUIOPASDFGHJKLZXCVBNM";  List<Character> chars = new java.util.ArrayList<>(Collections.emptyList());  for (int i = 0; i < str.length(); i++)  chars.add(str.charAt(i));  Set<Integer> endStates = Collections.singleton(UPPER);  StateMachine sm = new StateMachine(START, endStates);  sm.add(START, chars, UPPER);  sm.add(UPPER, chars, UPPER);  // Поиск всех вхождений в строке:  String inputString = "\"/Applications/IntelliJ IDEA CE.app/Contents/jbr/Contents/Home/bin/java\" \"-javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea\_rt.jar=63457:/Applications/IntelliJ IDEA CE.app/";  sm.findAll(inputString);  }  } |

# *Листинг 2, класс StateMachine*

|  |
| --- |
| package com.company;  import java.util.HashMap;  import java.util.List;  import java.util.Map;  import java.util.Set;  /\*\*  \* Конечный автомат, описывающий регулярное выражение \*/  public class StateMachine {  /\*\*  \* Начальное состояние  \*/  private final int startState;  /\*\*  \* Множество конечных состояний  \*/  private final Set<Integer> endStates;  /\*\*  \* Хранение таблицы переходов для конечного автомата:  \* исходное состояние -> (входной символ -> состояние, в которое КА переходит)  \*/  private final Map<Integer, Map<Character, Integer>> transitions = new HashMap<>();  /\*\*  \* @param startState начальное состояние  \* @param endStates множество успешных конечных состояний  \*/  public StateMachine(int startState, Set<Integer> endStates) {  this.startState = startState;  this.endStates = endStates;  }  /\*\*  \* Добавление возможного перехода в таблицу переходов. \*  \*  \* @param from исходное состояние  \* @param ch входной символ  \* @param to состояние, в которое КА переходит  \*/  public void add(Integer from, Character ch, Integer to) {  Map<Character, Integer> fromMap = transitions.computeIfAbsent(from, k -> new HashMap<>());  fromMap.put(ch, to);  }  /\*\*  \* Добавление нескольких возможных переходов в таблицу переходов. \*  \*  \* @param from исходное состояние  \* @param chars входные символы  \* @param to состояние, в которое КА переходит  \*/  public void add(Integer from, List<Character> chars, Integer to) {  for (Character ch : chars) {  add(from, ch.charValue(), to);  }  }  /\*\*  \* Получение следующего состояния по текущему состоянию и входному  \* символу. \*  \*  \* @param state текущее состояние  \* @param ch входной символ  \* @return новое состояние (null, если в таблице нет переходов, т.е.  \* сопоставление с регулярным выражением  \* не может быть осуществлено для такого входного символа)  \*/  public Integer getNext(Integer state, Character ch) {  Map<Character, Integer> stateTransitions = transitions.get(state);  return stateTransitions.get(ch);  }  /\*\*  \* Сопоставление подстроки с регулярным выражением.  \*  \* @param str строка  \* @param from индекс в строке, с которого начинается сопоставление  \* @return -1 если подстрока начиная с позиции from не сопоставлена с  \* регулярным выражением; неотрицательное  \* значение, если подстрока сопоставлена с регулярным выражением - индекс в  \* строке символа, следующего за  \* сопоставленной подстрокой  \*/  public int match(String str, int from) {  int state = startState;  int i = from;  while (i < str.length()) {  char ch = str.charAt(i);  Integer nextState = getNext(state, ch);  if (nextState == null)  break;  state = nextState.intValue();  i++;  }  if (endStates.contains(state)) {  return i;  } else {  return -1;  }  }  /\*\*  \* Поиск и печать всех сопоставлений в строке.  \*/  public void findAll(String str) {  int i = 0;  while (i < str.length()) {  int match = match(str, i);  if (match < 0) {  i++;  } else {  System.out.println(str.substring(i, match));  i = match;  }  }  }  } |

Изображение выглядит как снимок экрана, компьютер, монитор

Автоматически созданное описание

*Рисунок 1 - Выполнение готовой программы*

# 

# Вывод

В ходе выполнения практической работы были произведено ознакомление с понятием регулярного выражения, и универсального автомата. Также была создана программа, применяющая полученные навыки для нахождения в тексте заглавных букв.

# Список литературы

# А.А.Малявко. Формальные языки и компиляторы. Учебное пособие для вузов. – М.:ACT, 2017 – 433c.

# А.Саломаа. Жемчужины теории формальных языков. Учебное пособие для вузов. – М.:МИР, 1986 – 160c.

# Е.Н.Ешакова. Теория языков программирования и методов трансляции. – Спб.:БИБКОМ, 2007 – 137с.