

Доктор Регекс или как я перестал бояться и полюбил регулярные выражения

Марат Ахин

30 октября 2020 г.

спбпу

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

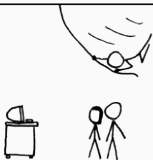


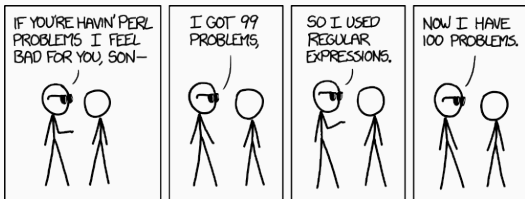
IT'S HOPELESS!

EVERYBODY STAND BACK.



I KNOW REGULAR EXPRESSIONS.





With regular expressions comes great responsibility (c)

- Помогают искать какой-либо текст в другом тексте
- Описывают интересующий нас текст
- Работают достаточно эффективно

Почему бы просто не перечислить все варианты текста?

- a@a.com
- b@a.com
- ...
- a@b.com
- ...

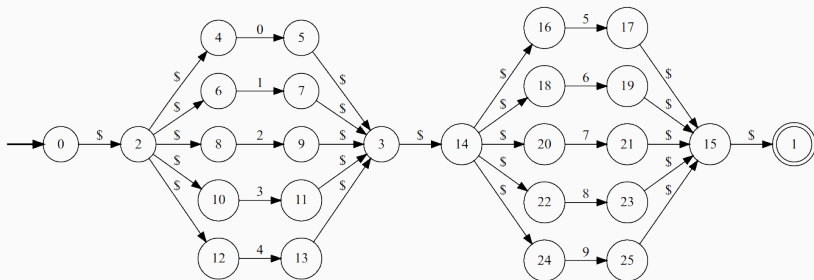
- Регулярные выражения описывают *регулярные строки*
 - KotlinAsFirst
 - `[A-Z0-9._%+~]+@[A-Z0-9.-]+\.[A-Z]{2,}`
 - `^4[0-9]{12}(?:[0-9]{3})?$$`
 - `[-+]?[0-9]*\.[0-9]+`
 - `<([a-z+)]^<]*(?:>(.*<V1>|V>)`

Что такое *регулярная строка*?

Регулярные выражения!

- Регулярное выражение может быть распознано *конечным автоматом*

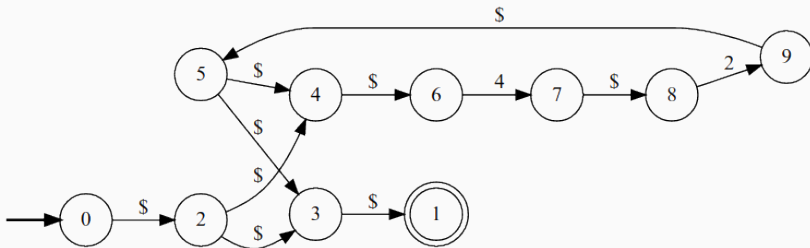
[0-4][5-9]



Конечный автомат

- Состояния
- Переходы между состояниями
- Условия переходов

$(42)^*$



- Я распознаю эту строку с 7 символов!

a{n}b{n}



- КА не умеют *считать*
- КА не умеют рекурсию
- Нужна память
- Нужна зависимость от памяти

Регулярные выражения **не являются** универсальным способом анализа строк

Студенты часто думают иначе... =(

Обычный символ ищет сам себя...

- KotlinAsFirst
- Трансмогрификация
- Мама мыла раму
- 42

...кроме ряда специальных символов (stay tuned!)

Класс символов ищет любой символ из определенного множества

- [0123456789]
- [aeiouy]
- [~!@#\$%^&*+~]

Отрицание класса символов ищет любой символ НЕ из заданного
множества

- [^0123456789]
- [^a-z]
- [^-az]

Якоря ищут начало или конец всей строки...

- `^fun`
- `\.$`
- `^Kotlin is great as a first language!$`

...и не учитывают переводы строки

Особые символы ищут символы по специальным правилам

- .
- `\t\n\r\f\v`
- `|s|S`
- `|d|D`
- `|w|W`
- `||`

Экранированные символы ищут сами себя

- `\^`
- `\$`
- `\.`
- `\[\]`
- `\\`

Шаблон выбора ищет по ИЛИ

- Марат|Михаил
- `^[\|]\$`
- `for.*(val|var).*`

Шаблон количества ищет определенное число совпадений

- `.*`
- `(Марат)+`
- `(Михаил)?`
- `([0-9]{4}){4}`
- `\w{8,16}`
- `Kotlin(?:As)+First`

Группы поиска объединяют несколько элементов вместе...

- `(Kotlin)+AsFirst`
- `(?:\$\$)+`

...и позволяют на них ссылаться

- `(\w+)\s\1`
- `fun\s+(\w+)\s*\{.*\1.*\}`

Группы особого поиска ищут по-особому

- Марат(?=\sАхин)
- (?<=Михаил\s)Глухих
- \d+(?!\\$)
- (?<!root\s)beer

Quiz time!



- `Regex("KotlinAsFirst")`
- `"KotlinAsFirst".toRegex()`
- `RegexOption`
 - `IGNORE_CASE`
 - `MULTILINE`
 - `LITERAL`
 - `UNIX_LINES`
 - `COMMENTS`
 - `DOT_MATCHES_ALL`
 - `CANON_EQ`

```
interface MatchResult {  
    val groupValues: List<String>  
    val groups: MatchGroupCollection  
    val range: IntRange  
    val value: String  
}
```

```
fun Regex.find(  
    input: CharSequence,  
    startIndex: Int = 0  
): MatchResult?
```

```
fun Regex.findAll(  
    input: CharSequence,  
    startIndex: Int = 0  
): Sequence<MatchResult>
```

```
fun Regex.replace(  
    input: CharSequence,  
    replacement: String  
): String
```

```
fun CharSequence.replace(  
    regex: Regex,  
    replacement: String  
): String
```

```
fun Regex.replaceFirst(  
    input: CharSequence,  
    replacement: String  
): String
```

```
fun CharSequence.replaceFirst(  
    regex: Regex,  
    replacement: String  
): String
```



```
fun Regex.containsMatchIn(input: CharSequence): Boolean
operator fun CharSequence.contains(regex: Regex): Boolean

fun Regex.matches(input: CharSequence): Boolean
fun CharSequence.matches(regex: Regex): Boolean

fun Regex.matchEntire(input: CharSequence): MatchResult?
```

Больше экранирования богу экранирования!

- В языке Kotlin `\\` в строке — экранированный символ `\` в итоговой строке
- В строке регулярного выражения `\\` — экранированный символ `\` в итоговом регулярном выражении

Чтобы получить `\\` в итоговом регулярном выражении, надо написать `\\\\` в исходной строке на языке Kotlin

Альтернативный способ — воспользоваться raw-строками

```
("""\\""")
```

- Регулярные выражения подходят для поиска *регулярных* строк
- Если вам нужно...
 - подсчитать количество скобок
 - учитывать одни группы при поиске других групп
 - делать что-то другое, что не умеет делать КА
- ...то лучше воспользоваться чем-то другим

Что дальше?

- <http://regexr.com/>
- <https://regex101.com/>
- ...and many more!

<https://docs.oracle.com/javase/tutorial/essential/regex/>