**Наумцев Антон**

**Домашняя работа**

**Второй курс | Первый модуль**

**Формальные языки # 3**

## Задание 1:

a) $L = \{uabv \mid u \in \{a, b\}*, v \in \{a, b\}*, |u| = |v|, u \neq v^R\}$

Предположим, что L - регулярный язык, тогда давайте рассмотрим слово вида: $b^n aba^n$, тогда по лемме о накачке его можно разбить на 3 кусочка, которые должны иметь вид:
$x = b^{n-l}, y = b^l, z = aba^n$, где $l > 0$, тогда заметим, что при $k = 0$ (средний кусок повторяется 0 раз) слово $b^{n-l}aba^n \notin L$, так как |u| \neq |v| \implies язык не является регулярным.

b) $L = \{a^k c^m e^n \mid k \geq 0, n \geq 0, m = k + n + 1\}$

Предположим, что L - регулярный язык, тогда давайте рассмотрим слово вида: $a^n m^{2n+1} e^n$, тогда по лемме о накачке его можно разбить на 3 кусочка, которые должны иметь вид:
$x = a^{n-l}, y = a^l, z = m^{2n+1}e^n$, где $l > 0$, тогда заметим, что при $k = 0$ (средний кусок повторяется 0 раз) слово $a^{n-l}m^{2n+1}e^n \notin L$, так как m = 2n + 1 \neq (n - l) + (n) + 1 \implies язык не является регулярным.

c) L = \{a^n \;|\; \exist p \geq n: p \; prime \; and \; p + 2 \}

Давайте заметим, что если слово $w = a^n \in L$, то и слова меньшей длины тоже входят в L, так как для них подходит то же $p$, что для $w$. Тогда есть 2 варианта:

1. $L$ - конечен, тогда $L$ - регулярный (можем перечислиь все слова)
2. $L$ - бесконечный, тогда $L$ включает в себя все слова $a^n$, и их можно описать как 'a*' \implies L - регулярный

Итого: мы показали, что внезависимости от существования $p$ для $n$, $L$ будет регурным языком.

## Задание 2:

В папке solution находится 2 версии парсера простых регулярных выражений: 1) Derivatives.py - без оптимизаций
2) DerivativesOptimized.py - с оптимизациями
Тесты можно посмотреть в файле tests.py
Чтобы запустить неоптимизированную версию поменяйте строчку (и поставьте флагу *with_big_tests* значение False, в противном случае вы не дождётесь завершения программы из-за больших тестов)

```
from DerivativesOptimized import match
```

на

```
from Derivatives import match
```

**Отчёт:**

При добавлении оптимизаций вычисления стали происходить в разы быстрее, но всё равно есть тесты, завершения которых можно не дождаться.

Приведу пример интересных результатов:

```
TEST #1 | Expression: 0(0|1)*0
0000000000 | Status: OK | Time:  10.290590s
------------------------------------------
0000000000 | Status: OK | Time:  0.000245s


TEST #2 | Expression: (111|000)*
000111000 | Status: OK | Time:  5.309656s
-----------------------------------------
000111000 | Status: OK | Time:  0.000096s

TEST #3 | Expression: (11)*01(00)*
111010000 | Status: OK | Time:  25.524381s
-----------------------------------------
111010000 | Status: OK | Time:  0.000080s


TEST #4 | Expression: 101010
10101011123 | Status: OK | Time:  2.666680s
------------------------------------------
10101011123 | Status: OK | Time:  0.000063s
```

Из тестов видно, что ускорение происходит в более чем 100 раз

Теперь хочется показать, сколько времени работает оптимизированный на "больших" выражениях

```
TEST #1 | Expression: ((1*1|00)|(111|000)*)*|(22|33*)
1100011100011100011100011100111 | Status: OK | Time:  4.334691s


1110001110001110001110001110011 | Status: OK | Time:  8.764154s
```

При увеличении входной строки, было замечено, что время всё равно растёт очень быстро: при длине строки n = 50, оптимизированный парсер уже не завершался за разумное время.