

# Relatório EP4 - MAC0323 Algoritmos e Estruturas de Dados II

---

Sabrina Araújo - nºUSP 12566182

Programa que recebe uma expressão regular, que pode conter concatenações, alternativas(`|`), fechos(`*`), coringas(`.`), um ou mais(`+`), conjunto(`[ ]`), intervalo (`[ - ]`) ou complementos (`[^]`) e um conjunto de palavras, e, para cada palavra, verifica se é ou não reconhecida pela expressão regular.

## Algoritmo

---

### Função `regex()`

A função `regex()` recebe um objeto da classe Grafo e um array de char que armazena a expressão regular. Essa função constrói um grafo que tem um vértice para cada caractere da expressão regular e as arestas correspondem às  $\epsilon$  transições. Temos as seguintes operações:

- Concatenação: se é um caractere, guarda que o estado pode ler o caractere, caso contrário inclui arco para o estado seguinte.
- Parêntesis: uma pilha é utilizada para lembrar onde começa o parêntesis, e inclui  $\epsilon$ -transição para o próximo.
- Alternativa: temos  $\epsilon$ -transição do 1º parêntesis para o caractere depois do `|`,  $\epsilon$ -transição do `|` para o fecha parêntesis, e o `|` é inserido na pilha para saber onde apontar.
- Fecho: se ocorre depois de caractere tem arco de e para o caractere, se ocorre depois de `)` deve ter arco de e para o abre parêntesis correspondente, e o arco para frente.
- Um ou mais: se ocorre depois de caractere tem arco para o caractere, se ocorre depois de `)` deve ter arco para o abre parêntesis correspondente, e o arco para frente.
- Colchete: se é um `[`, o `]` correspondente tem arco para frente.

### Função `reconhece()`

A função `reconhece()` recebe um objeto da classe Grafo, um array de char que armazena a expressão regular (`char *letra`) e um array de char que armazena a palavra (`char *pal`). Essa função verifica se a partir da palavra é possível chegar no final do autômato construído na função `regex()`. Ao andar pelo autômato temos os seguintes casos quando um vértice é atingido:

- O vértice é igual ao caractere, então marcamos que o próximo vértice será verificado pelo ``dfsR()`.
- O vértice é `'\'`, então se o caractere seguinte for igual ao caractere da palavra que estamos verificando, o vértice seguinte à esse caractere seguinte será verificado.
- O vértice é `'.'`, então para qualquer caractere que estamos verificando o vértice seguinte será verificado.

- O vértice é '[', então temos 4 casos: conjunto, intervalo, conjunto com complemento e intervalo com complemento. Caso o caractere que está sendo verificado corresponda às condições desses casos, o vértice seguinte ao '[' será verificado pelo `dfsR()`. No final das verificações, a função devolve se o último vértice, que determina o final e não tem caractere, foi alcançado.

## grafo.h

Nesse programa, a classe Grafo está implementada. O objeto é iniciado com uma lista ligada com tamanho igual ao da expressão regular + 1. Ainda mais, a função `dfsR()` implementa uma busca em profundidade, no qual visita todos os vértices do grafo andando pelos arcos de um vértice a outro.

Observação: nos testes abaixo as letras do alfabeto não são case-sensitive nos casos dos intervalos (`[A-Z] = [a-z]`).

## Testes

---

### Testes usando os exemplos do enunciado do EP4

#### Exemplo 1

Expressão regular:

- `(([a-z])*|([0-9]))*@[([a-z])+\.]+br`

Palavras:

- `cef1999@ime.usp.br`
- `thilio@bbb.com`

Saída:

```
(([a-z])*|([0-9]))*@[([a-z])+\.]+br
2
cef1999@ime.usp.br
thilio@bbb.com
S
N
```

```
real    0m19.317s
user    0m0.015s
sys     0m0.000s
```

#### Exemplo 2

Expressão regular:

- `(.)*A(.)*`

Palavras:

- AAAAAAAAAA
- BCA
- AAAAABBBBBB
- BBB

Saída:

```
(.)*A(.)*
```

```
4
```

```
AAAAAAAAAA
```

```
BCA
```

```
AAAAABBBBBB
```

```
BBB
```

```
S
```

```
S
```

```
S
```

```
N
```

```
real      0m27.304s
```

```
user      0m0.015s
```

```
sys       0m0.000s
```

## Exemplo 3

Expressão regular:

- (A\*CG|A\*TA|AAG\*T)\*

Palavras:

- AACGTAAATA
- CAAGA
- ACGTA
- AAAGT

Saída:

```
(A*CG|A*TA|AAG*T)*
```

```
4
```

```
AACGTAAATA
```

```
CAAGA
```

```
ACGTA
```

```
AAAGT
```

```
S
```

```
N
```

```
S
```

```
N
```

```
real    0m28.803s
```

```
user    0m0.000s
```

```
sys     0m0.015s
```

## Exemplo 4

Expressão regular:

- `[^AEIOU][AEIOU][^AEIOU][AEIOU]`

Palavras:

- GATO
- FINO
- OLHO
- BELO
- RUSSO

Saída:

```
[^AEIOU][AEIOU][^AEIOU][AEIOU]
```

```
5
```

```
GATO
```

```
FINO
```

```
OLHO
```

```
BELO
```

```
RUSSO
```

```
S
```

```
S
```

```
N
```

```
S
```

```
N
```

```
real    0m14.097s
```

```
user    0m0.000s
```

```
sys     0m0.015s
```

## Outros Testes

## Teste 1

Expressão regular:

- `[0-9]*\-[0-9]*`

Palavras:

- 347823-213
- 472A-23
- 3434-
- -324
- 9-A

Saída:

```
[0-9]*\-[0-9]*
5
347823-213
472A-23
3434-
-324
9-A
S
N
S
S
N

real    0m33.151s
user    0m0.000s
sys     0m0.015s
```

## Teste 2

Expressão regular:

- `[^0-9][0-9]*`

Palavras:

- 12345
- A4905
- 96BS
- -980
- K

Saída:

```
[^0-9][0-9]*
```

```
5
```

```
12345
```

```
A4905
```

```
96BS
```

```
-980
```

```
K
```

```
S
```

```
S
```

```
N
```

```
S
```

```
S
```

```
real    0m56.657s
```

```
user    0m0.000s
```

```
sys     0m0.015s
```

## Teste 3

Expressão regular:

- `\(([0-9])*\)[0-9]*\-[0-9]*`

Palavras:

- (11)91234-5678
- (1234)1234-1234
- (AB)91234-2334
- ()9-
- (1)9-1

Saída:

```
\(([0-9])*\)[0-9]*\-[0-9]*
```

```
5
```

```
(11)91234-5678
```

```
(1234)1234-1234
```

```
(AB)91234-2334
```

```
()9-
```

```
(1)9-1
```

```
S
```

```
N
```

```
N
```

```
S
```

```
S
```

```
real    1m2.150s
```

```
user    0m0.000s
```

```
sys     0m0.000s
```

## Teste 4

Expressão regular:

- `([A-G])+([0-5])*`

Palavras:

- AEDG023
- ABHJZ09
- E
- E901
- ABC234

Saída:

```
([A-G])+([0-5])*
```

```
5
```

```
AEDG023
```

```
ABHJZ09
```

```
E
```

```
E901
```

```
ABC234
```

```
S
```

```
N
```

```
S
```

```
N
```

```
S
```

```
real      0m56.210s
```

```
user      0m0.000s
```

```
sys       0m0.031s
```

## Teste 5

Expressão regular:

- `g(oog)+le`

Palavras:

- google
- googoogle
- gooooogle
- googoooogle
- googoogoogoogoogo

Saída:

```

g(oog)+le
5
google
googoogle
gooooogle
googooooogle
googooogoogoogoogle
S
S
N
N
S

real    0m31.031s
user    0m0.000s
sys     0m0.015s

```

## Teste 6

Expressão regular:

- `([0-9])*\.([0-9])*\.([0-9]*\.([0-9]))*`

Palavras:

- 192.168.1.255
- 345.233.9.314
- 1234.1234.1234.1234
- ab346.123a.34
- 1.1.1.1

Saída:

```

([0-9])*\.([0-9])*\.([0-9]*\.([0-9]))*
5
192.168.1.255
345.233.9.314
1234.1234.1234.1234
ab346.123a.34
1.1.1.1
S
S
N
N
S

real    1m19.726s
user    0m0.000s
sys     0m0.015s

```

## Teste 7



Expressão regular:

- $([a-z])\backslash-([a-b])^*\backslash-([a-c])^+$

Palavras:

- z-ababab-abcabcabc
- a-b-c
- x-y-z
- abc
- aaaaaaaa-bbbbbbbccccc-a

Saída:

```
([a-z])\-([a-b])*\-([a-c])^+
5
z-ababab-abcabcabc
a-b-c
x-y-z
abc
aaaaaaa-bbbbbbbccccc-a
S
S
N
N
N

real    0m27.673s
user    0m0.000s
sys     0m0.015s
```

## Teste 8

Expressão regular:

- $(a|b)^*c$

Palavras:

- aaaaaaabbabbbbc
- aaababababc
- cababab
- abc
- aaaac

Saída:

```

(a|b)*c
5
aaaaaaaaabbbbbbc
aaababababc
cababab
abc
aaaaac
S
S
N
S
S

real      0m24.895s
user      0m0.000s
sys       0m0.000s

```

## Teste 9

Expressão regular:

- $(.)*([0-2])+.$

Palavras:

- abcdefg
- aaaaa020202
- bbbb0303
- cccc0a
- x0yz9

Saída:

```

(.)*([0-2])+.
5
abcdefg
aaaaa020202
bbbb0303
cccc0a
x0yz9
N
S
S
S
S
N

real      0m46.071s
user      0m0.015s
sys       0m0.000s

```

## Teste 10

Expressão regular:

- $(AA^*B)$

Palavras:

- A
- AAAAA
- AB
- B
- BA

Saída:

```
(AA*B)
```

```
5
```

```
A
```

```
AAAAA
```

```
AB
```

```
B
```

```
BA
```

```
N
```

```
N
```

```
S
```

```
N
```

```
N
```

```
real      0m21.978s
```

```
user      0m0.000s
```

```
sys       0m0.015s
```

## Teste 11

Expressão regular:

- $((a|b)^*c)$

Palavras:

- a
- ab
- abc
- ac
- bc

Saída:

```

((a|b)*c)
5
a
ab
abc
ac
bc

N
N
S
S
S

real    0m25.833s
user    0m0.000s
sys     0m0.015s

```

## Teste 12

Expressão regular:

- $A^*B$

Palavras:

- A
- B
- AAB
- AAABBB
- AB

Saída:

```

A*B
5
A
B
AAB
AAABBB
AB
N
S
S
N
S

real    0m15.310s
user    0m0.000s
sys     0m0.000s

```

## Teste 13

Expressão regular:

- `A([A-Z])`

Palavras:

- `AA`
- `aA`
- `aa`
- `Aa`
- `Za`

Saída:

```
A([A-Z])
```

```
5
```

```
AA
```

```
aA
```

```
aa
```

```
Aa
```

```
Za
```

```
S
```

```
N
```

```
N
```

```
S
```

```
N
```

```
real    0m52.790s
```

```
user    0m0.000s
```

```
sys     0m0.015s
```

Observação: em relação aos tempos de execução, é possível notar que são mínimos.