

02 de junho

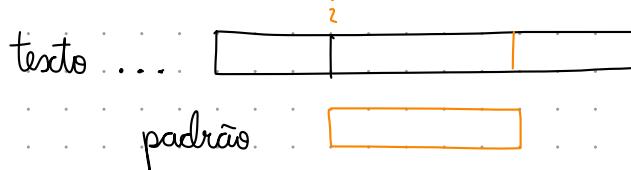
## Parte 03 - ALGORITMOS DE PROCESSAMENTO DE TEXTOS

### ① BUSCA DE PADRÕES

Considere um texto com  $n$  caracteres e um padrão com  $m$  caracteres e encontre uma (ou todas) ocorrência do padrão no texto.

Nas aplicações interessantes  $m \ll n$  e  $n$  pode ser muito grande ( $n = 10^8$ ,  $m = 10^4$ )

O algoritmo ingênuo testa todos os possíveis inícios para procurar o padrão.



```
int ingenuo(char * texto, char * pad){  
    int m = strlen(pad);  
    int n = strlen(texto);  
    for(int i = 0; i <= n-m; i++){  
        int j;  
        for(j = m-1; j > 0; j--)  
            if(texto[i+j] != pad[j])  
                break;  
        if(j == -1) return i;  
    }  
    return n; /* padrao nao ocorre */  
}
```

O algoritmo trabalha em tempo  $(n-m) * m = nm - m^2$

Um exemplo de pior caso é:

texto: AAAAAA ... A

padrão: BA ... A

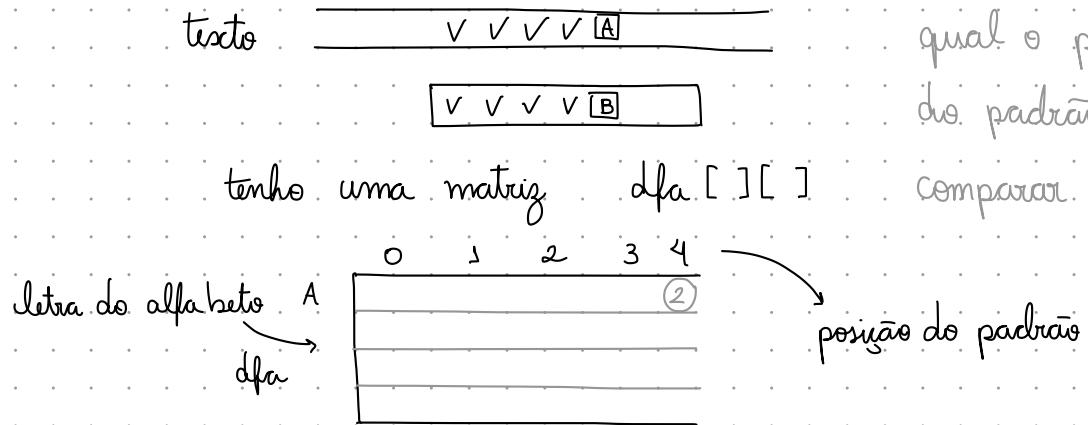
Um algoritmo mais eficiente é devido a Knuth, Morris e Pratt (KMP - 1977). Esse algoritmo encontra o padrão em tempo  $O(n)$ .

07 de junho

## ALGORITMO KMP

É um algoritmo de busca de padrões em textos que faz  $O(n)$  comparações (onde  $n$  é o tamanho do texto).

A ideia é pré-processar o padrão de forma a aproveitar o que já foi comparado.



### EXEMPLO

	A	B	A	B	A	C	A
A	1	1	3	1	5	1	7
B	0	2	0	4	0	4	0
C	0	0	0	0	0	6	0

A matriz  $dfa[c][c]$  contém o índice do padrão com que a próxima letra do texto deverá ser comparada.

	A	A	A	A	B
A	1	2	3	4	1
B	0	0	0	0	5

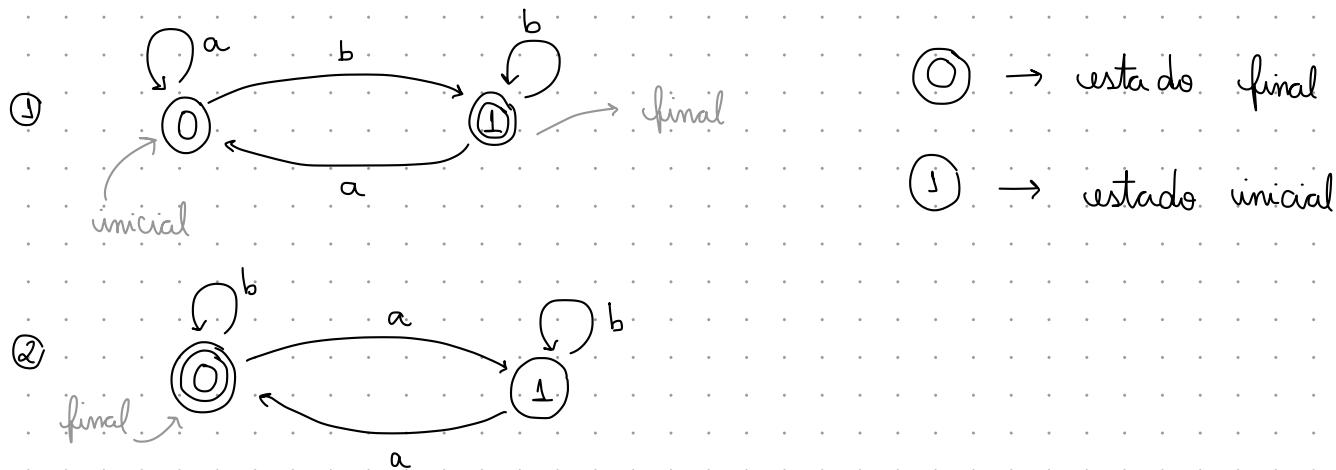
	A	A	B	A	B	A	A	B
A	1	2	2	4	2	6	7	2
B	0	0	3	0	5	0	0	8

07 de junho

A matriz  $dfa[][]$  é a matriz de transições de um autômato finito determinístico. [MAC 4.14].

Um autômato é constituído por estados e transições. Alguns estados podem ser finais (ou de aceitação) e um estado é inicial.

EXEMPLO.



Dizemos que uma palavra é aceita pelo autômato se a o letrá-la a partir do estado inicial termina em um estado final.

A linguagem aceita pelo autômato é o conjunto de palavras do alfabeto aceitas pelo autômato

$$L_{(1)} = \{x \in \{a,b\}^* \mid x \text{ termina em } b\}$$

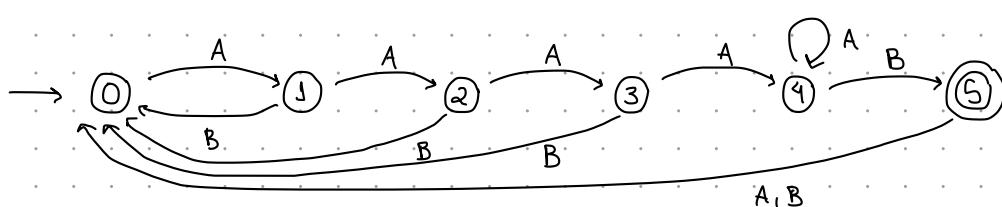
$$L_{(2)} = \{x \in \{a,b\}^* \mid x \text{ tem um número par de } (?)\}$$

↑  
(palavra vazia)  $\in L_{(2)}$

Autômatos finitos determinísticos conseguem expressar linguagens regulares (hierarquia de Chomsky). Há linguagens que não podem ser representadas por DFA:

$$L = \{w \in \{a,b,c\}^* \mid w = xcxc^R, \text{ onde } xc^R \text{ é o reverso de } x\}$$

EXEMPLO. A A A A B



07 de junho

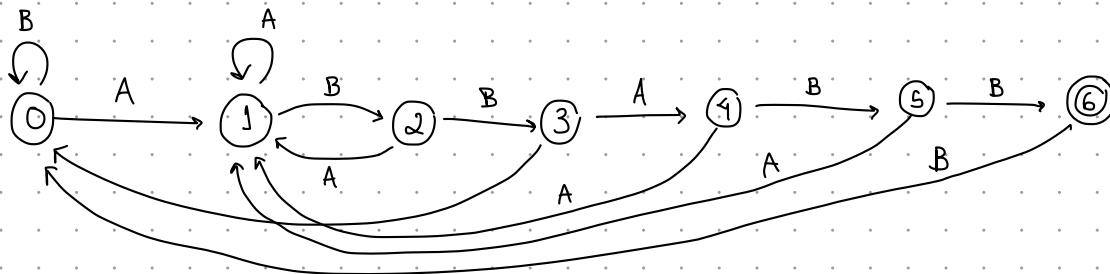
```
int KMP(char * texto, char * padrao){  
    int m = strlen(padrao);  
    int n = strlen(texto);  
    int i, j; /* i percorre o texto, j percorre o padrao */  
    for(i=0, j=0; i<n && j<m; i++)  
        j = dfa[texto[i]][j];  
    if(j==m) return i-m;  
    returnn; /*nao ocorre*/  
}
```

O algoritmo roda um  $O(n)$  mais o tempo para construir a matriz dfa.  
Tempo para construir a matriz é  $O(m * |\text{alf}|)$

09 de junho

## CONSTRUÇÃO DO AUTÔMATO

Ex: A B B A B B



dfa	0	1	2	3	4	5
A	1	1	1	4	1	4
B	0	2	3	0	5	0

Supondo que o alfabeto é  $\{0, 1, 2, \dots, \text{tam} - 1\}$

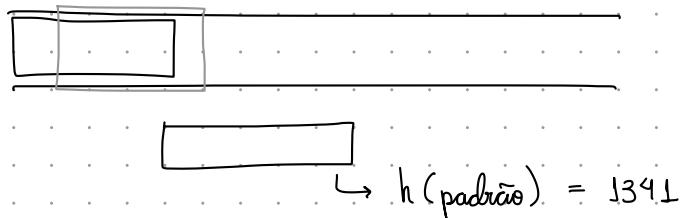
```
void constrói (int **dfa, int *padrão, int tam){  
    int x, c, j;  
    for(c = 0; c < tam; c++) dfa[c][0] = 0;  
    dfa[padrão[0]][0] = 1; x = 0;  
    for(j = 1; j < m; j++){  
        for(c=0; c < tam; c++)  
            dfa[c][j] = dfa[c][x];  
        dfa[padrão[j]][j] = j+1;  
        x = dfa[padrão[j]][x];  
    }  
    for(c=0; c < tam; c++)  
        dfa[c][m] = dfa[c][x];  
}
```

complexidade:  $O(m \cdot tam)$

09 de junho

## KARP - RABIN (~80's)

Este algoritmo é uma aplicação de hashing para busca de padrões:



Dá-se fácil obter o hash do próximo trecho do texto a partir do anterior.

Em um primeiro passo as letras do padrão são usadas em uma função de hashing. O mesmo cálculo é feito com as letras do texto, e só compararmos se o valor da função de hash é o mesmo.

$m + n$  poucas comparações quando  
↑ ↑ o hash é igual  
hashing do padrão hashing do texto

Os autores sugeriram duas implementações:

(1) checa quando a função de hash é igual ("Las Vegas")

(2) não checa, e pode devolver falsos positivos ("Monte carlo")

Também chamamos o algoritmo de busca por impressão digital.

09 de junho

## EXPRESSÕES REGULARES

Em muitas aplicações estaremos interessados em buscas por palavras que comecem de um jeito, terminem ou contenham algum padrão.

EXEMPLO: encontrar uma palavra que termine com ?ato

Uma expressão regular é composta por algumas operações:

- **Concatenação:** se  $x_1$  e  $x_2$  são exp. reg.,  $x_1x_2$  é uma exp. reg. Se a exp. reg.  $x_1$  reconhece  $p_1$  e  $x_2$  reconhece  $p_2$ ,  $x_1x_2$  reconhece  $p_1p_2$ .
- **Alternativa:** se  $x_1$  e  $x_2$  são exp. reg.,  $x_1|x_2$  é uma exp. reg. Se  $x_1$  reconheceu  $p_1$  e  $x_2$  reconheceu  $p_2$ ,  $x_1|x_2$  reconhece ambos.
- **Fecho:** se  $x_1$  é uma exp. reg. e  $x_1$  reconhece  $p_1$ , então  $x_1^*$  é uma expressão regular que reconhece 0 ou mais cópias de  $p_1$ .

EXEMPLOS:

$$A(B|C)D \longrightarrow \{ABD, ACD\}$$

$$a^*(b|c)a^* \longrightarrow \{abaaa, ca, aba, \dots\}$$

$$bb^*acb^* \longrightarrow \{bacb, bbacb, bacbb, \dots\}$$

$$(B|A|C)^* \longrightarrow \{E, A, AB, CCAB, \dots\}$$

Mais informalmente, uma expressão regular sobre um alfabeto  $\Sigma$  é ou

- a palavra vazia  $\epsilon$
- um caractere do alfabeto
- uma exp. reg. entre parêntesis
- a concatenação de duas exp. reg.
- duas exp. reg. separadas por  $|$
- uma exp. reg. seguida por  $*$

09 de junho

Do ponto de vista usmântico, uma exp. reg. reconhece uma linguagem (strings de letras do alfabeto)

- a exp. reg.  $E$  reconhece a linguagem  $L = \{E\}$
- a exp. reg.  $C$  reconhece a linguagem  $L = \{C\}$
- a expressão reg. ( $e_1$ ) reconhece a mesma linguagem que  $e_1$
- se  $e_1$  reconhece  $L_1$ ,  $e_2$  reconhece  $L_2$
- $e_1 e_2$  reconhece  $L = \{p_1 p_2 \mid p_1 \in L_1, p_2 \in L_2\}$
- $e_1 | e_2$  reconhece  $L_1 \cup L_2$
- $e_1^*$  reconhece  $\{\epsilon\} \cup \{p = p_1 p_2 \dots p_k \mid t.q. p_i \in L_1\}$

Algumas abreviaturas são bem convenientes:

• : coringa

Ex:  $AB.c = \{ABA\bar{C}, ABBC, \dots\}$

$AB(A|B|C|\dots)\bar{C}$

+ : fecho com pelo menos uma cópia

? : zero ou uma cópia

$B(A?)B = BB, BAB$

{k} : número fixo de repetições

$A\{3\}CB\{2\}DA\{5\} = AACBBDA\bar{A}\bar{A}\bar{A}\bar{A}\bar{A}$

[ ] : conjunto

$R[AEI\bar{O}U]S[AEI\bar{O}U]^*$

[-] : intervalo

$[0-9]^* \setminus [0-9]^* : 4\bar{2}3 - 15, 32-7, -4, 4-$

[~] : complemento

$[\sim 0-9][0-9]^* : A\bar{2}3, B, A\bar{1}\dots$

Expressões regulares podem ser usadas como poder de expressão

EXEMPLO:  $([a-z] | [0-9] | \cdot | \_) + @ (([a-z]) + (\cdot)) + br$   
 $aaa\bar{2}345.f2@ime.usp.com.br$

$\backslash([0-9]\{2\}\backslash)9[0-9]\{4\}\bar{1}[0-9]\{4\}$   
 $(\dots)97\bar{1}43-\bar{1}234$

09 de junho

(011)\*0

binários pares

Será que existem linguagens que não podem ser expressas por exp. reg?

EXEMPLO:  $\{x \in \{a, b\}^* \mid x \text{ tem o mesmo n.º de } a's \text{ e } b's\}$

não pode ser expressa por uma exp. reg.

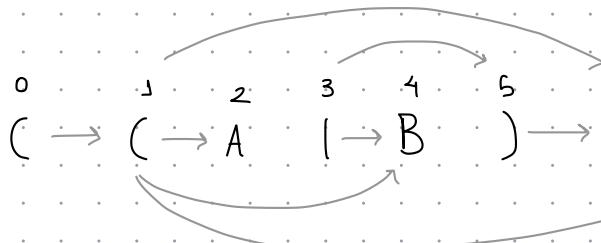
No caso de expressões regulares o operador | traz o risco de terminismo.

EXEMPLO:  $((A * B) | AC) * D$

21 de junho

## EXPRESSÕES REGULARES

$$((A \mid B) * (AC \mid BD))$$



A A B A A C

B A B D

Vamos construir um grafo que tem um vértice para cada caractere da expressão regular e as arestas correspondem às  $\epsilon$  transições

$$A \rightarrow \{0, 1, 2, 4, 7, 8, 11\}$$

$$A \rightarrow \{3, 9\} \rightarrow \{3, 5, 6, 1, 2, 4, 7, 8, 11, 9\}$$

$$B \rightarrow \{3, 9\} \rightarrow \{3, 5, 6, 1, 2, 4, 7, 8, 11, 9\}$$

$$A \rightarrow \{5, 12\} \rightarrow \{5, 6, 1, 2, 4, 7, 8, 11, 12\}$$

$$A \rightarrow \{3, 9\} \rightarrow \{3, 5, 6, 1, 2, 4, 7, 8, 11, 9\}$$

$$A \rightarrow \{3, 9\} \rightarrow \{3, 5, 6, 1, 2, 4, 7, 8, 11, 9\}$$

$$C \rightarrow \{10\} \rightarrow \{13, 14, 15, 10\}$$

```
bool reconhece(Graph G, char * pal){  
    bool * atingidos = new bool[G.V];  
    for(int i=0; i<G.V; i++) atingidos[i]=false;  
    G.dfsR(0, atingidos);  
    for(int i=0; i<strlen(pal); i++){  
        bool * prox = new bool[G.V];  
        for(int j=0; j<G.V; j++) prox[j] = false;  
        for(int j=0; j<G.V; j++)  
            if(atingidos[j] && letra[j]==pal[i])  
                prox[j+1]=true;  
        bool * marked = new bool[G.V];  
        for(j=0; j<G.V; j++) autingidos[j] = false;  
        for(j=0; j<G.V; j++)  
            if(prox[j]){  
                for(k=0; k<G.V; k++) marked[k] = false;  
                G.dfsR(j, marked);  
                for(k=0; k<G.V; k++)  
                    if(marked[k])  
                        atingidos[k] = true;  
            }  
    }  
    return atingidos[G.V-1];  
}
```

21 de junho

Uma expressão regular é formada por uma das 4 operações:

#### ① Concatenação

Se é um caractere, guarda que o estado pode ler o caractere, caso contrário inclui arco para o estado seguinte.

#### ② Parentesis ( ) ~>

Vamos usar uma pilha para lembrar onde começa o parêntesis (pode vir \*)  
E inclui E-transições para o próximo.



Vamos ter:

- E-transição do 1º parêntesis para o caractere depois do |
- E-transição do | para o fechamento parêntesis

Guarda na pilha o | para saber para onde apontar



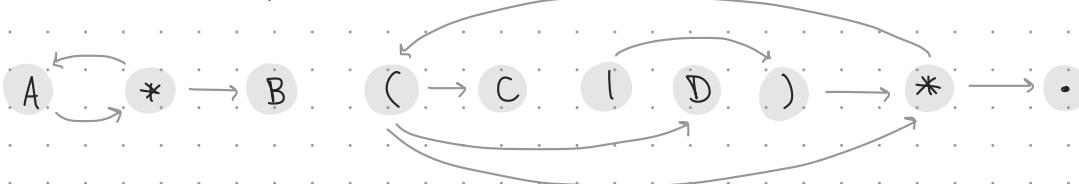
Se ocorre depois de caractere tem arco de \* para o caractere.

Se ocorre depois de ) deve ter arco de \* para o abre parêntesis correspondente

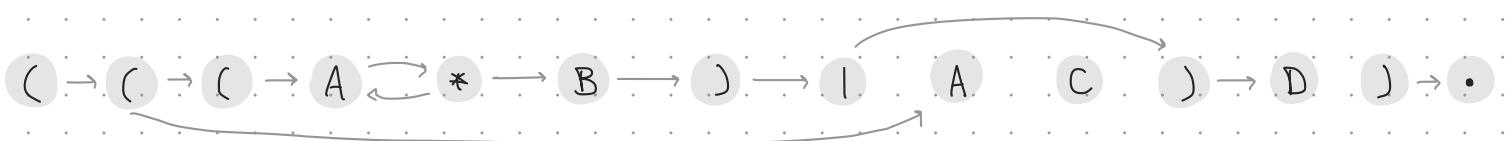
E o arco para frente.

### EXEMPLO

A \* B (C | D) \*



(( ((A \* B) | AC) D )



23 de abril

## EXPRESSÕES REGULARES

```
for(i=0; i<strlen(expreg); i++){
    int ant = i;
    if(expreg[i]=='(' || expreg[i]==')')
        pilha.push(i);
    else{ /*caractere ou */}
        if(expreg[i] == ')'){
            int otopo = pilha.pop();
            if(expreg[otopo] == '('){
                ant = pilha.pop();
                G.incluirco(ant, otopo+1);
                G.incluirco(otopo, i);
            }
            else ant = otopo; /* era '(' */
            ant = otopo;
        }
    if(i+1 < strlen(exp) && expreg[i+1] == '*'){
        G.incluirco(ant, i+1);
        G.incluirco(i+1, ant);
    }
    if(expreg[i]=='(' || expreg[i] == '*' || expreg[i] == ')')
        G.individuo(i, i+1);
}
```