Algoritmo de KMP

Vanessa de Souza Câmara
Fernanda Pinto Lopes

ROTEIRO DA APRESENTAÇÃO

1. MOTIVAÇÃO (F)
2. ALGUNS CONCEITOS (V)
3. ALGORITMO DE KMP (V)
4. FUNÇÃO PI (V)
5. IMPLEMENTAÇÃO (V)
6. COMPLEXIDADE (F)
7. VANTAGENS E DESVANTAGENS (F)
8. AUTÔMATO FINITO EM KMP (F)
9. APLICAÇÕES COM EXEMPLOS REAIS (V)
10. USO DO KMP (F)
11. REFERÊNCIAS

MOTIVAÇÃO

MOTIVAÇÃO

- Busca em Substrings ou *String Matching*

MOTIVAÇÃO

- Busca em Substrings ou *String Matching*
- Esse problema consiste em encontrar uma dada string dentro de outra maior

MOTIVAÇÃO

- Busca em Substrings ou *String Matching*
- Esse problema consiste em encontrar uma dada string dentro de outra maior
- Exemplo: Dado um texto t com m caracteres e um padrão s com n caracteres, em quais posições de t s ocorre como substring?

MOTIVAÇÃO

- Busca em Substrings ou *String Matching*
- Esse problema consiste em encontrar uma dada string dentro de outra maior
- Exemplo: Dado um texto t com m caracteres e um padrão s com n caracteres, em quais posições de t s ocorre como substring?

t = aabaacaadaabaaba

s = aaba

- Matching nas posições 0, 9 e 12

MOTIVAÇÃO

- Busca em Substrings ou *String Matching*
- Esse problema consiste em encontrar uma dada string dentro de outra maior
- Exemplo: Dado um texto t com m caracteres e um padrão s com n caracteres, em quais posições de t s ocorre como substring?

t = aabaacaadaabaaba
s = aaba

- Matching nas posições 0, 9 e 12

MOTIVAÇÃO

- Busca em Substrings ou *String Matching*
- Esse problema consiste em encontrar uma dada string dentro de outra maior
- Exemplo: Dado um texto t com m caracteres e um padrão s com n caracteres, em quais posições de t s ocorre como substring?

t = aabaacaadaabaaba
s = aaba

- Matching nas posições 0, 9 e 12

ALGUNS CONCEITOS

- Prefixo/sufixo **próprio** de uma string s é um prefixo/sufixo diferente de s.
  - “ab” é prefixo próprio de “abcd”
  - “abcd” não é prefixo próprio de “abcd”
  - “cd” é sufixo próprio de “abcd”
  - “abcd” não é sufixo próprio de “abcd”

PROBLEMA

- Dado um texto t com m caracteres e um padrão s com n caracteres, em quais posições de t s ocorre como substring?

t = abababababcd

s = abababcd

PROBLEMA

t = abababcd

s = ababab cd

- Vamos comparar e verificar a maior substring de s e t

PROBLEMA

t = abababcd

s = ababab**cd**

- Vamos comparar e verificar a maior substring de s e t (não queremos descartar nosso progresso)

PROBLEMA

t = abababcd

s = abababcd

ababab**c**d

- Vamos comparar e verificar a maior substring de s e t
- Depois disso, vamos tentar reaproveitar alguns caracteres comparados (o que é possível reaproveitar?)

PROBLEMA

t = ababababcd

s = abababcd

ababababcd

- Vamos comparar e verificar a maior substring de s e t
- Depois disso, vamos tentar reaproveitar alguns caracteres comparados
- Podemos observar, então, que existe uma simetria do padrão com ele mesmo

PROBLEMA

t = ababababcd

s = abababcd

ababababcd

- Vamos comparar e verificar a maior substring de s e t
- Depois disso, vamos tentar reaproveitar alguns caracteres comparados
- Podemos observar, então, que existe uma simetria do padrão com ele mesmo
- vamos reaproveitar carcteres que já comparamos. Para isso, deve haver um prefixo de s que seja igual ao sufixo dele mesmo

DEFININDO A FUNÇÃO PI

- Conceito: Para cada prefixo s1 de s, a função pi nos diz o tamanho do maior prefixo próprio de s que também é sufixo próprio de s1.

DEFININDO A FUNÇÃO PI

- Conceito: Para cada prefixo s1 de s, a função pi nos diz o tamanho do maior prefixo próprio de s que também é sufixo próprio de s1.

$$\pi[i] = \max_{0 \leq k \leq i} \{k : s[0 : k-1] = s[i-k+1 : i]\}$$

DEFININDO A FUNÇÃO PI

- Exemplo:

i	0	1	2	3	4	5	6
pi[i]	0	0	0	1	2	3	0
s	a	b	c	a	b	c	d

- calculando pi por partes

DEFININDO A FUNÇÃO PI

- Exemplo:

i	0	1	2	3	4	5	6
pi[i]	0	0	0	1	**2**	3	0
s	a	b	c	a	b	c	d

DEFININDO A FUNÇÃO PI

- Exemplo:

i	0	1	2	3	4	5	6
pi[i]	0	0	0	1	**2**	3	0
s	a	b	c	a	b	c	d

- O maior prefixo (próprio) que é sufixo (próprio) da string até a posição 4 é 2

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s = abacab
```

$$i = 0\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s = abacab
```

$$i = 1\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

t = abadaabaccabacabacab

s = abacab

$$i = 2\,;\, j = 2$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s = abacab
```

$$i = 3\,;\, j = 3$$

- sendo i o índice do texto
- sendo j o índice do padrão

- C e D são diferentes e temos que lidar com uma pergunta:

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s = abacab
```

$$i = 3\,;\, j = 3$$

- sendo i o índice do texto
- sendo j o índice do padrão

- quantos caracteres podemos aproveitar para não perder todo o progresso e começar do zero?

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	**1**	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s = abacab
```

$$i = 3\,;\, j = 3$$

- sendo i o índice do texto
- sendo j o índice do padrão

- a resposta está em pi[j-1]

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =   abacab
```

$$i = 3\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

- feito isso, o j recebe pi[j-1]

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =   abacab
```

$$i = 3\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

- D e B são diferentes, olhamos para pi[j-1]

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	**0**	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =   abacab
```

$$i = 3\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

- novamente, esse é o tamanho da string que é prefixo e sufixo próprio

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =    abacab
```

$$i = 3\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

- como não podemos salvar nada, seguimos do zero

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =    abacab
```

$$i = 3\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

- como não podemos salvar nada, seguimos do zero
- j == 0, então só passamos

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 4\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 5\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	**0**	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 5\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 5\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 6\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 7\,;\, j = 2$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 8\,;\, j = 3$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 9\,;\, j = 4$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	**0**	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =     abacab
```

$$i = 9\,;\, j = 4$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab
```

$$i = 9\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab
```

$$i = 10\,;\, j = 0$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab
```

$$i = 11\,;\, j = 1$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =         abacab
```

$$i = 12\,;\, j = 2$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab
```

$$i = 13\,;\, j = 3$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab
```

$$i = 14\,;\; j = 4$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab
```

$$i = 15\,;\, j = 5$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab_
```

$$i = 16\,;\; j = 6$$

- sendo i o índice do texto
- sendo j o índice do padrão

- o que é possível reaproveitar até agora?

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	**2**
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =          abacab_
```

$$i = 16\,;\; j = 6$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =               abacab
```

$$i = 16\,;\; j = 2$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =                 abacab
```

$$i = 17\,;\, j = 3$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =                abacab
```

$$i = 18\,;\; j = 4$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab
s =                abacab
```

$$i = 19\,;\, j = 5$$

- sendo i o índice do texto
- sendo j o índice do padrão

O ALGORITMO KMP

i	0	1	2	3	4	5
pi[i]	0	0	1	0	1	2
s	a	b	a	c	a	b

```
t = abadaabaccabacabacab_
s =                abacab_
```

$$i = 20\,;\; j = 6$$

- sendo i o índice do texto
- sendo j o índice do padrão

IMPLEMENTAÇÃO E COMPLEXIDADE

```
1  vector<int> matching(string& t, string& s) {
2      vector<int> p = pi(s+'$'), match;
3      for (int i = 0, j = 0; i < t.size(); i++) {
4          while (j > 0 and s[j] != t[i]) j = p[j-1];
5          if (s[j] == t[i]) j++;
6          if (j == s.size()) match.push_back(i-j+1);
7      }
8      return match;
9  }
```

- Em cada iteração do while, o j diminui em pelo menos 1
- O j só aumenta quando o i também aumenta
- Portanto, o while executa no máximo m =|t| iterações.
- A complexidade do algoritmo é a complexidade de construir a função de prefixo pi + O(m)

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0															
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	**0**															
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0														
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	**0**														
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0													
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	**0**													
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- para este caso, vemos que a == a, então ele incrementa o padrão até aquele momento

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1												
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	**1**												
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- novamente, precisamos que o char na posição 1 seja igual ao char atual

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2											
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	**2**											
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- neste caso a != c, então temos que saber quantos chars é possível reaproveitar
- vamos olhar para a posição pi[2 - 1] e olhar para s[pi[2-1]]

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	**0**	0	1	2											
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1										
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- assim, conseguimos estender o 0 para 1

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1										
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- como a o char no indice 1 é b, conseguimos estender para 2

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	**4**							
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5						
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	**5**						
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- agora nos perguntamos, quanto de tamanho podemos aproveitar?

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	**5**						
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

- agora nos perguntamos, quanto de tamanho podemos aproveitar?
- olhamos para a última posição do "pai" que gerou essa repetição

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	**5**						
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3					
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	**3**					
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4				
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	**4**				
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	**1**	2	1	2	3	4	5	3	4				
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	**0**	0	0	1	2	1	2	3	4	5	3	4				
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	1			
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	**1**			
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	1	2		
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	1	**2**		
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	**0**	0	1	2	1	2	3	4	5	3	4	1	2		
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	1	2	1	
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	1	2	**1**	
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	**0**	0	0	1	2	1	2	3	4	5	3	4	1	2	1	
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

COMO COMPUTAR A FUNÇÃO PI

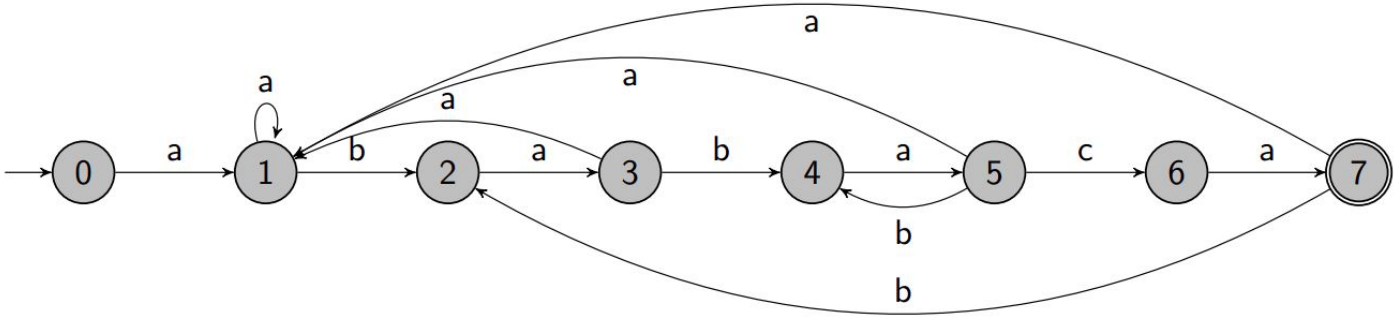
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi[i]$	0	0	0	1	2	1	2	3	4	5	3	4	1	2	1	0
s	a	b	c	a	b	a	b	c	a	b	c	a	a	b	a	c

IMPLEMENTAÇÃO E COMPLEXIDADE

```
3       for (int i = 1, j = 0; i < s.size(); i++) {
4           while (j > 0 and s[j] != s[i]) j = p[j-1];
5           if (s[j] == s[i]) j++;
6           p[i] = j;
7       }
```

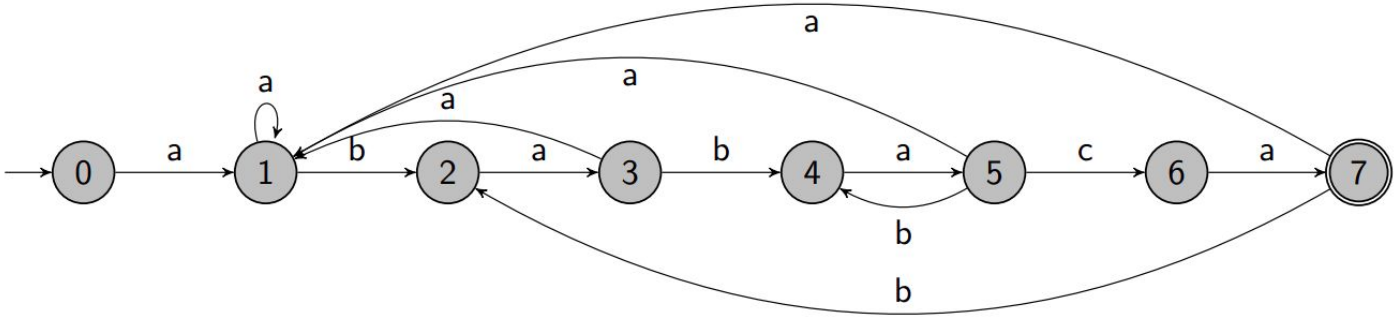
- Em cada iteração do while, o j diminui em pelo menos 1
- O j s´o aumenta quando o i também aumenta
- Portanto, o while executa no máximo n =|s| iterações.
- A complexidade do algoritmo é O(n)
- A complexidade do algoritmo de KMP é O(n + m).

REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

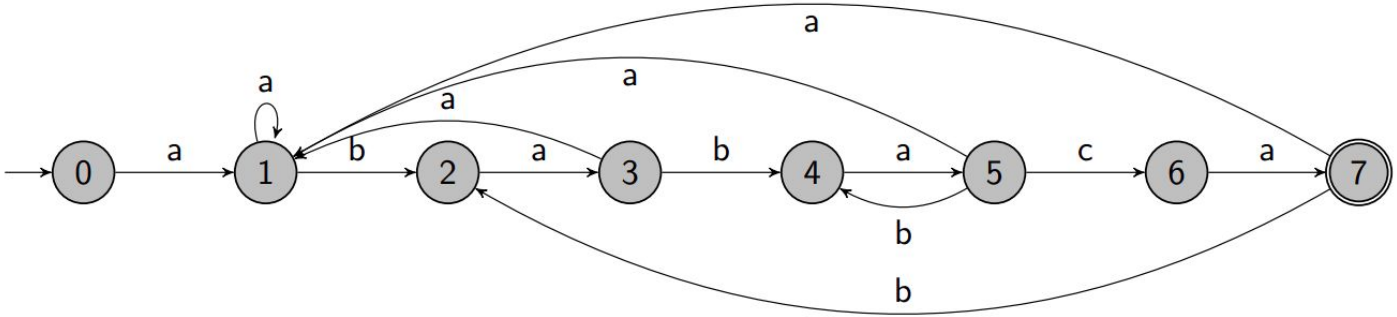
REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

- Podemos representar isso em um autômato (grafo)

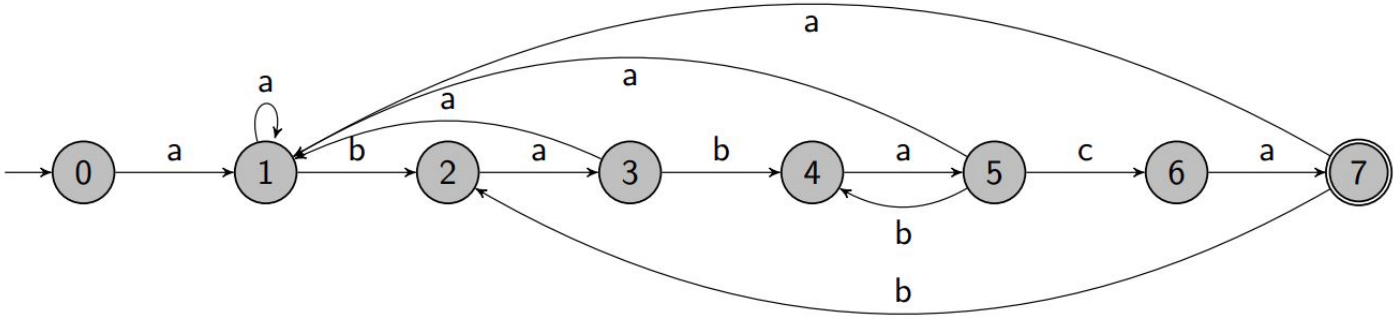
REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

- Cada estado é o j, que vai de 0 a 6

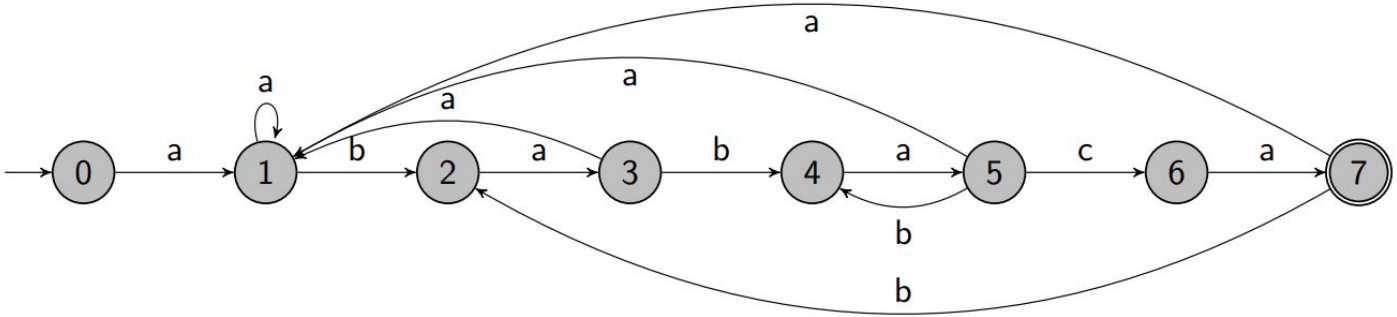
REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

- Se estamos no j == 3 e lemos "a", paramos no j = 1

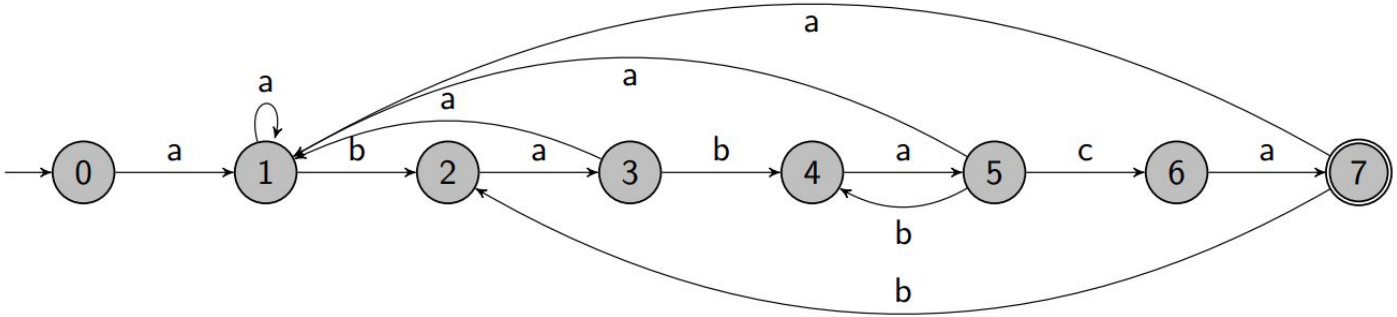
REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

- O estado inicial é sempre 0 porque é impossível ter um tamanho de prefixo próprio que é sufixo próprio se a string tem tamanho 1

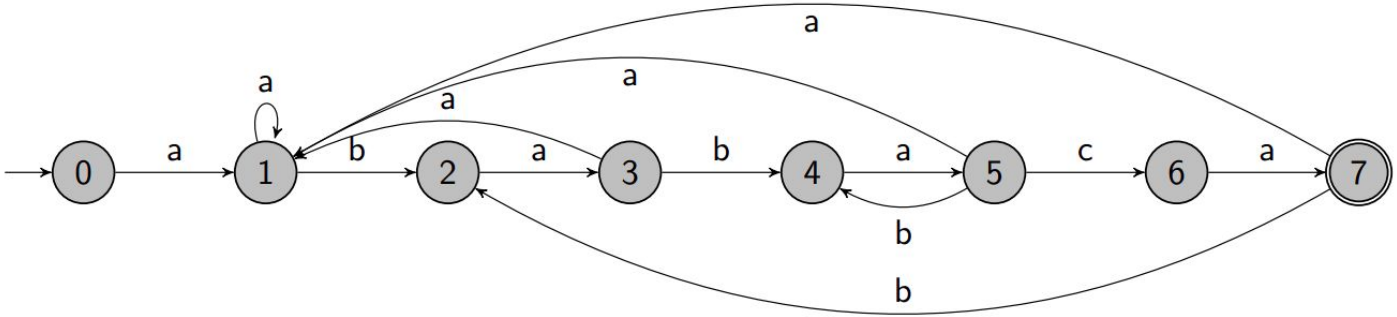
REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

- O estado final/ estado de aceitação é o tamanho da string, se chegamos nele, significa que achamos um match

REPRESENTAÇÃO DO KMP COMO AUTÔMATO FINITO



i	0	1	2	3	4	5	6
$\pi[i]$	0	0	1	2	3	0	1
s	a	b	a	b	a	c	a

- Todas as transições que não foram desenhadas levam o j atual para o estado inicial

VANTAGENS

- O algoritmo KMP é rápido e tem a vantagem de nunca retroceder sobre o texto, o que é importante se o texto for dado como um fluxo contínuo (streaming).
- Foi o primeiro algoritmo cujo pior caso tem complexidade de tempo linear no tamanho do texto: O(n)
- É um dos algoritmos mais famosos para resolver casamento de padrões

DESVANTAGENS

- Tem implementação complicada
- Na prática, perde em eficiência para outros algoritmos, como o de Boyer

APLICAÇÕES COM EXEMPLOS

Conjunto de Problemas CSES

String Matching

Limite de tempo: 1,00 s **Limite de memória:** 512 MB

Dado uma string e um padrão, sua tarefa é contar o número de posições onde o padrão ocorre na string.

Entrada

A primeira linha de entrada tem uma string de comprimento$n$, e a segunda linha de entrada tem um padrão de comprimento m. Ambos consistem em caracteres a – z.

Saída

Imprime um inteiro: o número de ocorrências.

Restrições

- $1 \leq n, m \leq 10^6$

Exemplo de

entrada: saída:
```
saippuakauppias
pp
```


Algoritmos de String

Combinações de palavras –
String Matching ✓
Encontrando Fronteiras –
Períodos de descoberta –
Rotação mínima –
Palíndromo Mais Longo –
Substring necessária –
Consultas Palíndromo –
...

Seus envios

2021-07-11 23:17:52 ✓
2021-07-11 23:16:32 ×
2021-07-11 23:15:13 ×

```cpp
#include <bits/stdc++.h>
using namespace std;
string s, t;
vector<int> p(1001233), match;
int matching(string t, string s)
{
    int n = t.size(), m = s.size();
    p[0] = 0;
    for (int i = 1, j = 0; i < m; i++)
    {
        while (j > 0 && s[j] != s[i])
        {
            j = p[j - 1];
        }
        if (s[j] == s[i])
            j++;
        p[i] = j;
    }

    for (int i = 0, j = 0; i < n; i++)
    {
        while (j > 0 && t[i] != s[j])
            j = p[j - 1];
        if (s[j] == t[i])
            j++;
        if (j == m)
            match.push_back(i - j + 1);
    }
    int ans = match.size()
    return ans;
}
```

Limite de tempo: 1,00 s **Limite de memória:** 512 MB

Dados uma string e padrões, encontre para cada padrão a primeira posição (indexada 1) onde ele aparece na string.

Entrada

A primeira linha de entrada tem uma string de comprimento$n$.

A próxima linha de entrada tem um inteiro$k$: o número de padrões. Finalmente, existem$k$ linhas que descrevem os padrões.

A string e os padrões consistem em caracteres a – z.

Saída

Imprime a primeira posição para cada padrão (ou- 1 se não aparecer).

Restrições

- $1 \leq n \leq 10^5$
- $1 \leq k \leq 5 \cdot 10^5$
- o comprimento total dos padrões é no máximo $5 \cdot 10^5$

Exemplo de

entrada: saída:

```
aybabtu
3
bab
abc
a
```

```
3
-1
1
```

Algoritmos de String

...
Consultas Palíndromo –
Encontrando Padrões –
Padrões de contagem –
Posições de padrão ×
Substrings distintos –
Substring repetitivo –
Funções de String –
Ordem de substring I –
...

Seus envios

2021-07-11 23:47:04 ×
2021-07-11 23:44:19 ×

Detalhes de envio

Tarefa:	Posições de padrão
Remetente:	Vanessadcamara
Horário de envio:	2021-07-11 23:47:04
Língua:	C ++ 17
Status:	PRONTO
Resultado:	LIMITE DE TEMPO EXCEDIDO

Resultado dos testes

teste	veredito	Tempo	
# 1	ACEITARAM	0,03 s	»
# 2	ACEITARAM	0,76 s	»
# 3	LIMITE DE TEMPO EXCEDIDO	-	»
# 4	LIMITE DE TEMPO EXCEDIDO	-	»
# 5	ACEITARAM	0,19 s	»
# 6	ACEITARAM	0,17 s	»
# 7	ACEITARAM	0,01 s	»

```cpp
#include <bits/stdc++.h>
using namespace std;
string s, t;
vector<int> p(500123), pattern;
int matching(string t, string s)
{
    int n = t.size(), m = s.size();
    p[0] = 0;
    for (int i = 1, j = 0; i < m; i++)
    {
        while (j > 0 && s[j] != s[i])
        {
            j = p[j - 1];
        }
        if (s[j] == s[i])
            j++;
        p[i] = j;
    }

    for (int i = 0, j = 0; i < n; i++)
    {
        while (j > 0 && t[i] != s[j])
            j = p[j - 1];
        if (s[j] == t[i])
            j++;
        if (j == m)
            return i - j + 2;
    }
    return -1;
}
int main()
{
    cin >> t;
    int k;
    cin >> k;
    while (k--)
    {
        cin >> s;
        cout << matching(t, s) << endl;
    }
    return 0;
}
```

Uso do KMP

Mecanismo de "busca/*find*" em um editor de texto

Pesquisa na web por arquivos com certa palavra

Busca de padrões que indicam *spam* em e-mail

Busca palavras password no computador por hackers

REFERÊNCIAS

- https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/kmp.html
- https://www.youtube.com/watch?v=RXISWaGmYW8&t=2308s
- https://www.ufjf.br/jairo_souza/files/2009/12/7-Strings-Casamento-de-padr%c3%b5es.pdf
- https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/kmp.html
- https://pt.slideshare.net/mcastrosouza/algoritmo-de-knuthmorrispratt-kmp
- https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/substring-search.html
- https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/