

# Casamento de Padrões

Análise de Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira  
Nunes

Instituto Federal de Brasília,  
Câmpus Taguatinga



# Sumário

---

- 1 Introdução
- 2 Casamento Exato



# Sumário

---

## 1 Introdução



# Introdução

---

- O Problema de Casamento de Padrões está entre um dos mais estudados em Computação
- Sua solução proporciona Aplicações nas mais diversas áreas.



# Aplicações

---

## Biologia Computacional

- Alinhamento Múltiplo.
- Mapeamento de Sequências.
- Montagem.



# Aplicações

---

## Compiladores

- Identificação de Tokens.
- Unificação.



# Aplicações

---

## Editores de Texto

- Identificar padrões no arquivo.
- Substituir padrões por outros.
- Casamento de expressões regulares.



# Aplicações

---

## Recuperação de Informação

- Retornar os documentos que contém o padrão buscado.
  - ▶ Buscadores!
  - ▶ Big Data? Por trás disso tudo temos algoritmos. . .
  - ▶ A essência da Computação são algoritmos.
  - ▶ Tecnologia  $\neq$  Computação.





# Introdução

---

- O Casamento de Padrões pode ser:
  - ① Exato: ocorrências do Padrão no Texto sem nenhum erro.
  - ② Aproximado: ocorrências do Padrão no Texto permitindo até  $k$  erros segundo uma métrica de distância.
    - Casamento Exato = Casamento Aproximado quando  $k = 0$ .
  - ③ Focaremos em casamento exato de padrões.



# Introdução

---

- Antes de definir os problemas, precisamos de alguns conceitos iniciais.



# Conceitos Básicos

---

## Definição (Palavras)

- Palavras (*strings*) são sequências sobre o alfabeto  $\Sigma$ .
- O conjunto de todas as possíveis palavras é denotada por  $\Sigma^*$ .
- Em especial, temos a palavra vazia  $\varepsilon \in \Sigma^*$ .
- O  $i$ -ésimo símbolo de uma palavra  $S$ , é denotado por  $S[i]$ .



# Conceitos Básicos

---

## Definição (Tamanho de Palavras)

- O tamanho de uma palavra  $S$  é denotado por  $|S|$ , e contém a quantidade de símbolos em  $S$ .
- Em especial,  $|\varepsilon| = 0$ .



# Conceitos Básicos

---

## Definição (Subpalavras)

- Uma subpalavra de  $S$  é denotada por  $S[i, j]$ , e é constituída pelos símbolos  $S[i]S[i + 1] \dots S[j]$  desde que  $0 \leq i \leq j < |S|$ . Caso contrário,  $S[i, j] = \varepsilon$ .



# Conceitos Básicos

---

## Notação (Texto e Padrão)

- Em especial, o texto é uma palavra denotada por  $T$ , sendo  $|T| = n$ .
- Já o padrão é denotado por  $P$ , tendo tamanho  $|P| = m$ .



# Sumário

---

## 2 Casamento Exato



# Casamento Exato

---

- O casamento exato de padrões procura identificar as ocorrências de um padrão em um dado texto.





# Casamento Exato

---

## Casamento Exato

- Entrada:  $P$  e  $T$ .
- Saída:  $occ = \{k | P[0, m - 1] = T[k, k + m - 1]\}$ .



# Sumário

---

- 2 Casamento Exato
  - Algoritmo Ingênuo
  - Rabin-Karp
  - KMP



# Algoritmo Ingênuo

---

- O algoritmo ingênuo é capaz de resolver o problema do casamento exato de padrões.
- Ideia: força-bruta. Comparamos o padrão com todas as posições possíveis do texto.
- Você é capaz de delinear o algoritmo subjacente?



# Algoritmo Ingênuo

---

---

**Algorithm 1:** NAIVE-MATCHER( $P, T$ )

---

**Input:**  $T, P$

**Output:**  $occ = \{k | P[0, m - 1] = T[k, k + m - 1]\}$

```
1 for(  $i \leftarrow 0; i < n - m + 1; i++$  )
2    $found \leftarrow true$ 
3   for(  $j \leftarrow 0; j < m; j++$  )
4     if(  $T[i + j] \neq P[j]$  )
5        $found \leftarrow false$ 
6   if(  $found$  )
7     REPORT-MATCH( $i$ )
```

---



# Algoritmo Ingênuo

---

- Qual a complexidade de tal algoritmo?
- $\Theta(n \cdot m)$



# Sumário

---

- 2 Casamento Exato
  - Algoritmo Ingênuo
  - Rabin-Karp
  - KMP



## Rabin-Karp

---

- Tome  $P = aaaaaab$  e  $T = aaaaaaaaaaaaaaaaaaaaaa$ .
- O Algoritmo Ingênuo é extremamente ineficiente, pois ele precisa inspecionar todos os símbolos do padrão para descobrir que  $P$  não ocorre naquela posição do texto.
- O algoritmo de Rabin-Karp perde um pouco de tempo ao **preprocessar** informação do padrão de modo a agilizar a etapa de busca.



## Rabin-Karp

---

- Para facilitar, suponha que o alfabeto subjacente seja  $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ .
- Computamos o valor do padrão módulo algum número  $q$ , com preferência de  $q$  primo.
- Exemplo, se  $P = 31415$ , então  $P \bmod 13 = 7$ .
- De maneira geral, o alfabeto tem base  $d$ . No caso do alfabeto de DNA,  $d = 4$ . Considerando proteínas  $d = 20$ . Considerando ASCII,  $d = 255$ .





# Rabin-Karp

---

---

## Algorithm 2: RK-PREPROCESS

---

**Input:**  $P, d, q$

**Output:**  $P \bmod q$

```
1  $sum \leftarrow 0$ 
2 for(  $i = 0; i < m; i++$  )
3    $sum = sum \cdot d$ 
4    $sum \leftarrow sum + P[i]$ 
5 return  $sum \bmod q$ 
```

---



# Rabin-Karp

---

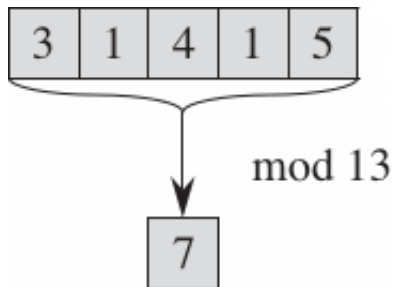


Figura: Preprocessando o padrão.



# Rabin-Karp

---

- A ideia do algoritmo é fazer várias comparações de caracteres através de uma única comparação de inteiros.
- Assim como o padrão, cada porção do texto a ser comparada com o padrão, é preprocessada.
- Se o valor do padrão  $P$  é  $k$  e o valor da porção do texto é  $l$ . O que podemos dizer quando:
  - ▶  $k \neq l$ ?
  - ▶  $k = l$ ?



# Rabin-Karp

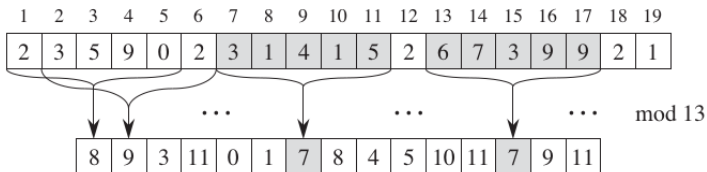


Figura: Valores das porções do texto.



## Processando o Texto

---

- Como converter o texto de maneira eficiente?
- Precisamos guardar o valor para cada posição do texto?



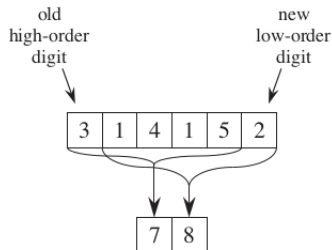
## Processando o Texto

---

- Como converter o texto de maneira eficiente?
- Precisamos guardar o valor para cada posição do texto?
- Na verdade não, só da porção que estamos analisando.



# Processando o Texto



$$\begin{aligned}
 14152 &\equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \\
 &\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13} \\
 &\equiv 8 \pmod{13}
 \end{aligned}$$

**Figura:** Processando o valor do texto em  $O(1)$  de tempo e espaço.



# Rabin-Karp

---

---

## Algorithm 3: COMPARE

---

**Input:**  $P, T, i$

**Output:**  $true \Leftrightarrow P = T[i, i + m - 1]$

```
1 for(  $k \leftarrow 0; k < m; k++$  )
2   |   if(  $P[k] = T[i + k]$  )
3   |   |   return false
4 return true
```

---





# Rabin-Karp

---

**Algorithm 4:** RK-MATCHER

---

**Input:**  $P, T, d, q$

**Output:**  $occ = \{k \mid P[0, m-1] = T[k][k+m-1]\}$

```
1  $h \leftarrow d^{m-1} \bmod q$ 
2  $P_v = \text{RK-PREPROCESS}(P, d, q)$ 
3  $T_v = \text{RK-PREPROCESS}(T[0, m-1], d, q)$ 
4 for ( $i \leftarrow 0; i < n - m + 1; i++$ )
5   if ( $P_v = T_v$ )
6     if ( $\text{COMPARE}(P, T, i)$ )
7       REPORT-MATCH( $i$ )
8   if ( $i < n - m$ )
9      $T_v \leftarrow (d \cdot (T_v - T[i] \cdot h) + T[i+m]) \bmod q$ 
```

---



# Complexidade

---

- Qual a complexidade do algoritmo de Rabin-Karp?



# Complexidade

---

- Qual a complexidade do algoritmo de Rabin-Karp?
- $O(n \cdot m)$ .



# Sumário

---

- 2 Casamento Exato
  - Algoritmo Ingênuo
  - Rabin-Karp
  - KMP



# KMP

---

- Qual o problema dos algoritmos anteriores?



# KMP

---

- Qual o problema dos algoritmos anteriores?
- Eles esquecem tudo! As comparações sempre são dadas do início.



# KMP

---

- O algoritmo KMP, descoberto por Vaughan Pratt e Donald Knuth e de maneira independente por James Morris, contorna esse problema. O algoritmo utiliza a informação aprendida durante o casamento para evitar comparações supérfluas nas próximas etapas.
- Os três se juntaram e publicaram a descoberta.
- Um dos artigos mais clássicos da área de “Stringology”.
- Knuth, Donald; Morris, James H.; Pratt, Vaughan (1977). “Fast pattern matching in strings”. SIAM Journal on Computing 6 (2): 323–350. doi:10.1137/0206024



# Algoritmo Ingênuo

---

## Exemplo

- Tome  $T = xyxxyxyxyxyxyxyxyxyxyxyx$
- Considere  $P = xyxyxyxyxyx$ .





# Algoritmo Ingênuo

Tabela: Casamento do Padrão com o Texto.

		0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2
	<i>T</i> :	<i>x</i>	<i>y</i>	<i>x</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>x</i>
0	<i>P</i> :	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>																			
1	<i>P</i> :		<i>x</i>																					
2	<i>P</i> :			<i>x</i>	<i>y</i>																			
3	<i>P</i> :				<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>y</i>															
4	<i>P</i> :					<i>x</i>																		
5	<i>P</i> :						<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>x</i>							
6	<i>P</i> :							<i>x</i>																
7	<i>P</i> :								<i>x</i>	<i>y</i>	<i>x</i>													
8	<i>P</i> :									<i>x</i>														
9	<i>P</i> :										<i>x</i>													
10	<i>P</i> :											<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>y</i>								
11	<i>P</i> :												<i>x</i>											
12	<i>P</i> :														<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>	<i>y</i>	<i>x</i>



# KMP

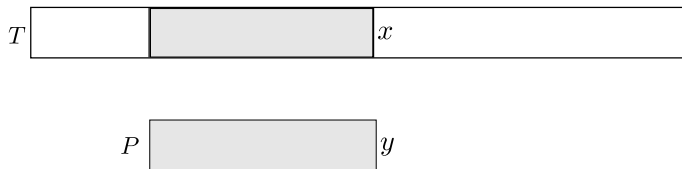
---

- O algoritmo KMP usa a informação da porção de  $P$  que causou para deslizar o padrão de maneira eficiente.
- Ele utiliza o conhecimento durante o processo de casamento para evitar comparações desnecessária.
- Vamos examinar como ele faz isso.



# KMP

---



**Figura:** Casamento de uma porção do padrão com o texto.

- Suponha que  $x \neq y$ .



# KMP

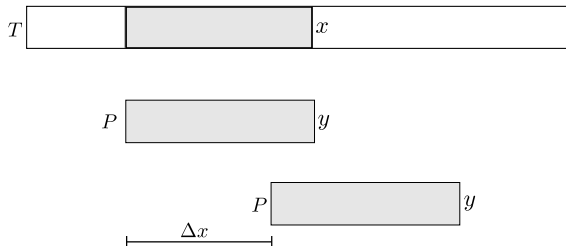


Figura: Deslizamento do padrão sobre o texto.

- Suponha que o algoritmo KMP deslize o padrão com um deslocamento  $\Delta x$ .
- O que podemos dizer sobre a sobreposição dos padrões para poder haver possibilidade de casamento?



# KMP

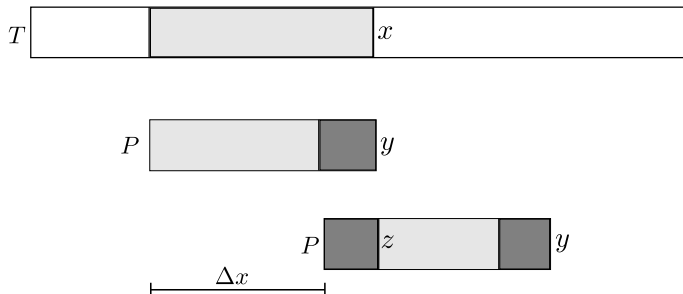
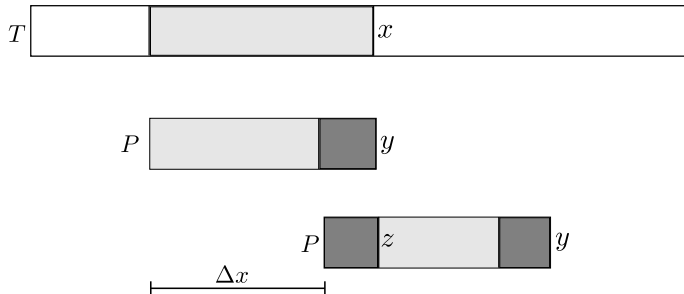


Figura: Deslizamento do padrão sobre o texto.

- Prefixo tem que ser igual a um sufixo da parte que casou.



# KMP



**Figura:** Deslizamento do padrão sobre o texto.

- Para não pular nenhuma ocorrência de  $P$  em  $T$ , o deslocamento deve ser o menor possível que atenda a propriedade.



# KMP

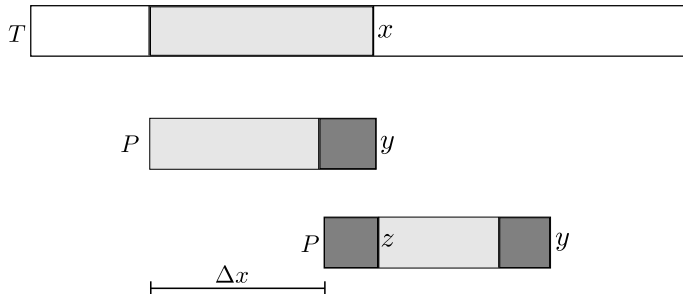


Figura: Deslizamento do padrão sobre o texto.

- Temos que achar o maior prefixo próprio que seja igual a um sufixo próprio da porção do Padrão que casou com o Texto.



# KMP

---

- Vamos definir nosso objeto de cálculo.
- Nossa função *next* vai nos dar o próximo caractere do padrão que se deve começar as comparações.

$$next(i) = \begin{cases} -1, & i = 0 \\ \max\{k | P[0, k - 1] = P[i - k, i - 1]\} \end{cases}$$





## Computando a Função *next*

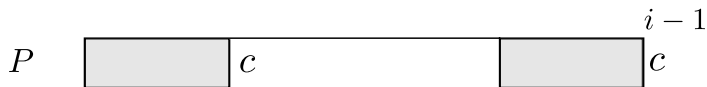
---

- Como computar *next*?
- Sabemos que  $next(0) = -1$ . Suponha agora que queiramos computar  $next(i)$  e já tenhamos computado  $next(j)$  com  $0 \leq j < i$ .
- Como computar  $next(i)$ ?



## Computando a Função *next*

---

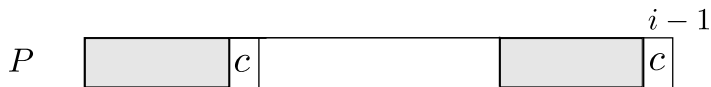


**Figura:** Caso 1:  $P[next(i - 1)] = P[i - 1]$

- Neste caso,  $next(i) = next(i - 1) + 1$ .



## Computando a Função *next*



**Figura:** Caso 1:  $P[\text{next}(i - 1)] = P[i - 1]$

- Neste caso,  $\text{next}(i) = \text{next}(i - 1) + 1$ .



## Computando a Função *next*

---

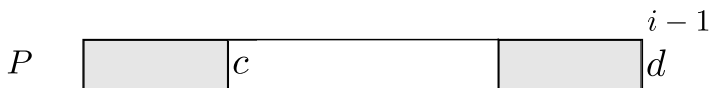
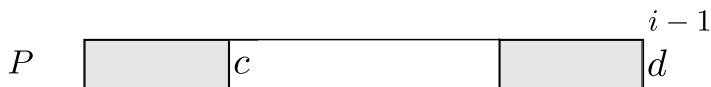


Figura: Caso 2:  $P[\text{next}(i - 1)] \neq P[i - 1]$

- Como não conseguimos estender e formar um sufixo. Precisamos pesquisar no próximo maior sufixo de  $P$  que é sufixo de  $P[0, i - 1]$ . Onde está esta informação?



## Computando a Função *next*



**Figura:** Caso 2:  $P[\text{next}(i - 1)] \neq P[i - 1]$

- Como não conseguimos estender e formar um sufixo. Precisamos pesquisar no próximo maior sufixo de  $P$  que é sufixo de  $P[0, i - 1]$ . Onde está esta informação?
- $\text{next}(\text{next}(i - 1))$ .



## Computando a Função *next*

---



**Figura:** Tentamos estender usando o próximo maior prefixo que também é sufixo.



## Pré-processamento

---

---

### Algorithm 5: KMP-PREPROCESS

---

**Input:**  $P$

**Output:**  $next$

```
1  $next[0] \leftarrow j \leftarrow -1$ 
2 for(  $i \leftarrow 1; i \leq m; i++$  )
3   while ( $j \geq 0 \wedge P[i-1] \neq P[j]$ ) do
4      $j \leftarrow next[j]$ 
5      $j++$ 
6    $next[i] \leftarrow j$ 
```

---



# Pré-processamento

---

## Análise

- Qual a complexidade de KMP-PREPROCESS?
- $\Theta(m^2)$ ?
- Quantas vezes o **while** interno pode executar?





## Pré-processamento

---

### Análise

- Qual a complexidade de KMP-PREPROCESS?
- $\Theta(m^2)$ ?
- Quantas vezes o **while** interno pode executar?
- Máximo de  $2m$  iterações agregadas no laço interno.
- $\Theta(m)$ .



# KMP

---

- Agora que temos a informação suficiente, podemos elaborar o algoritmo de busca.
- Toda vez que um caractere do texto diferir do  $i$ -ésimo caractere do padrão, não precisamos reinicializar a busca.
- Começamos de  $next[i]$ !



# Busca

---

## Algorithm 6: KMP-PREPROCESS

---

**Input:**  $T, P$

**Output:**  $occ\{k | P[0, m-1] = T[k, k+m-1]\}$

```
1  $next \leftarrow \text{KMP-PREPROCESS}(P)$ 
2  $i \leftarrow j \leftarrow 0$ 
3 while ( $i < n$ ) do
4   while ( $j \geq 0 \wedge T[i] \neq P[j]$ ) do
5      $j \leftarrow next[j]$ 
6    $j++$ 
7    $i++$ 
8   if ( $j = m$ )
9      $\text{REPORT-MATCH}(i-j)$ 
10     $j \leftarrow next[j]$ 
```



# KMP

---

## Exemplo

- Tome  $T = xyxxyxyxyxyxyxyxyxyxyx$
- Considere  $P = xyxyxyxyxx$ .
- Calcule *next* e aplique o algoritmo KMP.