

Lista de Exercícios 2 - AEDS 3

Henrique Oliveira da Cunha Franco

1 Compressão de dados

1.1 Q1

Uma compressão sem perdas é caracterizada pelo processo de redução do tamanho de um arquivo digital sem comprometer a qualidade ou a integridade dos dados contidos nele. Isso significa que, após a compressão e descompressão, o arquivo resultante é idêntico ao original.

1.2 Q2

Primeiro passo: As probabilidades de cada símbolo são:

- $P(A) = \frac{8}{20} = 0.4$
- $P(R) = \frac{4}{20} = 0.2$
- $P(N) = \frac{2}{20} = 0.1$
- $P(H) = \frac{2}{20} = 0.1$
- $P(\text{espaço}) = \frac{3}{20} = 0.15$
- $P(\tilde{A}) = \frac{1}{20}$

Segundo passo: Continuando com as probabilidades

- $P(A) = \frac{8}{20} = 0.4$
- $P(R) = \frac{4}{20} = 0.2$
- $P(N) = \frac{2}{20} = 0.1$
- $P(H) = \frac{2}{20} = 0.1$
- $P(\text{espaço}) = \frac{3}{20} = 0.15$
- $P(\tilde{A}) = \frac{1}{20} = 0.05$

Terceiro passo: Calcular a entropia

A fórmula da entropia de Shannon é:

$$H(X) = - \sum_i P(x_i) \log_2 P(x_i)$$

Cálculo da entropia passo a passo para cada símbolo:

- Para A :

$$P(A) \log_2 P(A) = 0.4 \log_2 0.4 \approx 0.4 \times (-1.32193) \approx -0.528772$$

- Para R :

$$P(R) \log_2 P(R) = 0.2 \log_2 0.2 \approx 0.2 \times (-2.32193) \approx -0.464386$$

- Para N :

$$P(N) \log_2 P(N) = 0.1 \log_2 0.1 \approx 0.1 \times (-3.32193) \approx -0.332193$$

- Para H :

$$P(H) \log_2 P(H) = 0.1 \log_2 0.1 \approx 0.1 \times (-3.32193) \approx -0.332193$$

- Para o espaço:

$$P(\text{espaço}) \log_2 P(\text{espaço}) = 0.15 \log_2 0.15 \approx 0.15 \times (-2.73697) \approx -0.410545$$

- Para \tilde{A} :

$$P(\tilde{A}) \log_2 P(\tilde{A}) = 0.05 \log_2 0.05 \approx 0.05 \times (-4.32193) \approx -0.216097$$

Somando todas as parcelas

$$H(X) = -(-0.528772 - 0.464386 - 0.332193 - 0.332193 - 0.410545 - 0.216097)$$

$$H(X) = 0.528772 + 0.464386 + 0.332193 + 0.332193 + 0.410545 + 0.216097$$

$$H(X) = 2.284186$$

Logo, a entropia da mensagem "A ARANHA ARRANHA A RÃ" é aproximadamente 2.28 bits por símbolo.

1.3 Q3

Para calcular a quantidade de bits necessária para representar a mensagem "A ARANHA ARRANHA A RÃ" usando diferentes métodos de compressão, precisamos analisar cada método individualmente. Vamos revisar cada técnica:

1. Codificação de Huffman A codificação de Huffman é um método de compressão sem perdas que utiliza uma árvore binária para atribuir códigos de comprimento variável a cada símbolo com base em suas frequências. Os símbolos mais frequentes recebem códigos mais curtos.

Frequências dos símbolos:

- A: 8
- R: 4
- N: 2
- H: 2
- (espaço): 3
- \tilde{A} : 1

Passos para construção da árvore de Huffman:

1. Listar todos os símbolos com suas frequências.
2. Combinar os dois símbolos de menor frequência em um novo nó.
3. Repetir até que reste apenas um nó.

Vamos calcular a árvore e os códigos de Huffman:

1. Combine " \tilde{A} "(1) e "N"(2): [R, A, H, espaço, (\tilde{A} N:3)]
2. Combine "H"(2) e "espaço"(3): [R, A, (Hespaço:5), (\tilde{A} N:3)]
3. Combine "(\tilde{A} N:3)" e "R"(4): [A, (\tilde{A} NR:7), (Hespaço:5)]
4. Combine "(Hespaço:5)" e "A"(8): [(\tilde{A} NR:7), (HespaçoA:13)]
5. Combine "(\tilde{A} NR:7)" e "(HespaçoA:13)": [(\tilde{A} NRHespaçoA:20)]

Construindo a árvore, temos os códigos:

- A: 0
- R: 10
- N: 110
- H: 1110
- (espaço): 11110
- \tilde{A} : 11111

Codificação da mensagem:

0 11110 0 10 0 110 1110 0 11110 0 10 0 110 1110 0 10 0 11111

Total de bits = $1 + 5 + 1 + 2 + 1 + 3 + 4 + 1 + 5 + 1 + 2 + 1 + 3 + 4 + 1 + 2 + 1 + 5 = 44$ bits

2. Codificação Shannon-Fano A codificação Shannon-Fano é semelhante a Huffman, mas a árvore de codificação é construída de maneira diferente, dividindo os símbolos em grupos com somas de probabilidades aproximadamente iguais.

Frequências ordenadas:

- A: 8
- (espaço): 3
- R: 4
- N: 2
- H: 2
- \tilde{A} : 1

1. A: 0
2. Espaço, R, N, H, \tilde{A} : 1
3. Espaço, R: 10; N, H, \tilde{A} : 11
4. N, H: 110; \tilde{A} : 111
5. N: 1100; H: 1101

Codificação da mensagem:

0 100 0 101 0 1100 1101 0 100 0 101 0 1100 1101 0 101 0 111

Total de bits = $1 + 3 + 1 + 3 + 1 + 4 + 4 + 1 + 3 + 1 + 3 + 1 + 4 + 4 + 1 + 3 + 1 + 3 = 42$ bits

3. Codificação LZ77 LZ77 é uma técnica de compressão que utiliza janelas deslizantes para encontrar duplicatas em dados. Ela não é diretamente representada em número de bits sem simulação, mas podemos descrever o processo.

Na nossa mensagem, LZ77 encontraria padrões repetidos, mas o tamanho exato da codificação depende da implementação específica, incluindo o tamanho da janela e como os apontadores são codificados.

4. Codificação LZ78 LZ78 cria uma tabela de substrings à medida que processa a mensagem. Ao encontrar substrings repetidas, ela as substitui por referências à tabela.

Sem simulação, não é trivial calcular exatamente o número de bits, mas a eficiência dependeria da presença de substrings repetidas, como "AR", "ANHA".

5. Codificação LZW LZW é uma variante do LZ78 que também cria uma tabela de substrings.

Simulação LZW para nossa mensagem:

- A: adiciona A ao dicionário.
- (espaço): adiciona espaço ao dicionário.
- AR: adiciona AR ao dicionário.
- AN: adiciona AN ao dicionário.
- AH: adiciona AH ao dicionário.
- AR: referência ao dicionário.
- NH: adiciona NH ao dicionário.
- A: referência ao dicionário.
- RÃ: adiciona RÃ ao dicionário.

LZW geralmente é eficiente para mensagens maiores.
Resumo dos bits necessários

- **Huffman:** 44 bits
- **Shannon-Fano:** 42 bits
- **LZ77, LZ78 e LZW:** a eficiência depende da implementação, mas geralmente são competitivos com Huffman e Shannon-Fano para pequenas mensagens repetitivas.

Portanto, sem simulação detalhada para LZ77, LZ78 e LZW, podemos ver que Huffman e Shannon-Fano fornecem uma boa estimativa da compressão para a mensagem dada.

1.4 Q4

Para calcular a quantidade de bits necessária para representar a mensagem "BOTE A BOTA NO BOTE E TIRE O POTE DO BOTE" usando diferentes métodos de compressão, vamos analisar cada método individualmente.

Codificação de Huffman

Frequências dos símbolos: - B: 4 - O: 6 - T: 7 - E: 6 - A: 3 - espaço: 9 - N: 1 - I: 1 - R: 1 - P: 1 - D: 1

Passos para construção da árvore de Huffman:

1. Listar todos os símbolos com suas frequências.
2. Combinar os dois símbolos de menor frequência em um novo nó.
3. Repetir até que reste apenas um nó.
1. Combine N (1) e I (1): [(2, N/I)]
2. Combine R (1) e P (1): [(2, R/P)]
3. Combine D (1) e N/I (2): [(3, D/N/I)]
4. Combine A (3) e D/N/I (3): [(6, A/D/N/I)]
5. Combine B (4) e R/P (2): [(6, B/R/P)]
6. Combine O (6) e A/D/N/I (6): [(12, O/A/D/N/I)]
7. Combine E (6) e espaço (9): [(15, E/espaço)]
8. Combine T (7) e B/R/P (6): [(13, T/B/R/P)]
9. Combine O/A/D/N/I (12) e T/B/R/P (13): [(25, O/A/D/N/I/T/B/R/P)]
10. Combine O/A/D/N/I/T/B/R/P (25) e E/espaço (15): [(40, O/A/D/N/I/T/B/R/P/E/espaço)]

Construindo a árvore, temos os códigos:

- B: 010
- O: 011
- T: 10
- E: 110
- A: 0110
- espaço: 111
- N: 0000
- I: 0001
- R: 0010
- P: 0011
- D: 0111

Codificação da mensagem:

010 011 10 110 111 0110 111 010 011 10 011 0110 111 0000 011 111 010 011 10 110 111 10 0001 0010 110 111 011 0011 011 111 0

Total de bits = $3 + 3 + 2 + 3 + 3 + 4 + 3 + 3 + 3 + 2 + 3 + 4 + 3 + 4 + 3 + 3 + 3 + 2 + 3 + 3 + 3 + 2 + 4 + 4 + 3 + 3 + 3 + 4 + 3 + 3 + 2 + 3 = 91$ bits

Codificação Shannon-Fano

Frequências dos símbolos:

- B: 4
- O: 6
- T: 7
- E: 6
- A: 3
- espaço: 9
- N: 1
- I: 1
- I: 1R: 1
- I: 1P: 1
- I: 1D: 1

Ordenando os símbolos por frequência:

- espaço: 9
- T: 7
- E: 6
- O: 6
- B: 4
- A: 3

- N: 1
- I: 1
- R: 1
- P: 1
- D: 1

Dividindo e construindo a árvore de Shannon-Fano:

1. Espaço (9), T (7), E (6), O (6), B (4), A (3)
2. Dividindo em dois grupos:
 - Espaço, T: 0
 - E, O, B, A: 1
3. Espaço (9): 0
 - T (7): 10
 - E, O, B, A: 11
4. E (6), O (6): 110
 - B (4), A (3): 111
5. E (6): 1100
 - O (6): 1101
 - B (4): 1110
 - A (3): 1111
6. N (1), I (1), R (1), P (1), D (1): 1111
 - N (1): 11110
 - I (1): 11111
 - R (1): 111111
 - P (1): 111110
 - D (1): 111111

Codificação da mensagem:

10 0110 11110 110 0 1100 0 10 0110 1100 0110 0 0010 10 1100 110 0010 0011 10 0 1100 0 10 0110 110 0010 1110 0011 10 0 1100

Total de bits = $3 + 4 + 5 + 3 + 1 + 4 + 1 + 3 + 4 + 4 + 4 + 1 + 4 + 3 + 4 + 3 + 4 + 4 + 3 + 1 + 4 + 1 + 3 + 4 + 3 + 4 + 4 + 3 + 1 + 4 = 91$ bits

Codificação LZ77 Para calcular a codificação LZ77, precisamos fazer uma simulação:

1. BOTE (4 bits)
2. A (1 bit)
3. BOTA (4 bits)
4. NO (2 bits)
5. BOTE (4 bits)
6. E (1 bit)
7. TIRE (4 bits)

8. O (1 bit)
9. POTE (4 bits)
10. DO (2 bits)
11. BOTE (4 bits)

Cada substring repetida pode ser referenciada pela posição e comprimento (no mínimo 9 bits, com 4 bits para posição e 5 bits para comprimento).

Codificação LZ78 Para calcular a codificação LZ78, precisamos fazer uma simulação:

1. B (1 bit)
2. O (1 bit)
3. T (1 bit)
4. E (1 bit)
5. A (1 bit)
6. BOTA (4 bits)
7. NO (2 bits)
8. BOTE (4 bits)
9. E (1 bit)
10. TIRE (4 bits)
11. O (1 bit)
12. POTE (4 bits)
13. DO (2 bits)
14. BOTE (4 bits)

Cada substring repetida é codificada pela referência à tabela.

Codificação LZW Para calcular a codificação LZW, precisamos fazer uma simulação:

1. B (1 bit)
2. O (1 bit)
3. T (1 bit)
4. E (1 bit)
5. A (1 bit)
6. BOTA (4 bits)
7. NO (2 bits)
8. BOTE (4 bits)
9. E (1 bit)
10. TIRE (4 bits)
11. O (1 bit)
12. POTE (4 bits)
13. DO (2 bits)

14. BOTE (4 bits)

Cada substring repetida é codificada pela referência à tabela.
Resumo dos bits necessários

- **Huffman:** 91 bits
- **Shannon-Fano:** 91 bits
- **LZ77:** Aproximadamente $11 \times 9 = 99$ bits
- **LZ78:** Aproximadamente 47 bits
- **LZW:** Aproximadamente 47 bits

Assim, Huffman e Shannon-Fano têm um número muito próximo de bits necessários, enquanto LZ77, LZ78, e LZW são dependentes das implementações específicas e da estrutura das tabelas de substrings.

2 Casamento de Padrões

2.1 Q1

A vantagem do algoritmo de busca KMP sobre o algoritmo de força bruta é a reutilização de padrões dentro do trecho lido pelo buffer. A partir disso, é possível pular passos em seções que já foram analisadas e não contém o padrão desejado.

2.2 Q2

Reset Build failure function Search Text: ABCABCAC Pattern: ABCABCAC

Build comparisons: 9

Failure function built.

A	B	C	A	B	C	A	C
---	---	---	---	---	---	---	---

A	B	C	A	B	C	A	C
---	---	---	---	---	---	---	---

Failure function:

j	0	1	2	3	4	5	6	7
P[j]	A	B	C	A	B	C	A	C
f(j)	0	0	0	1	2	3	4	0

Animation Completed

Skip Back Step Back Pause Step Forward Skip Forward Animation Speed w: 1000 h: 500

Algorithm Visualizations

2.3 Q3

Execução passo a passo:

Texto: A B A B A B A B A B A C

Índices: 0 1 2 3 4 5 6 7 8 9 10 11 12 13

Padrão: A B A B A C

Índices: 0 1 2 3 4 5

LPS: 0 0 1 2 3 0

1. $i = 0, j = 0$: $\text{texto}[0] == \text{padrão}[0] \rightarrow i = 1, j = 1$
2. $i = 1, j = 1$: $\text{texto}[1] == \text{padrão}[1] \rightarrow i = 2, j = 2$
3. $i = 2, j = 2$: $\text{texto}[2] == \text{padrão}[2] \rightarrow i = 3, j = 3$
4. $i = 3, j = 3$: $\text{texto}[3] == \text{padrão}[3] \rightarrow i = 4, j = 4$
5. $i = 4, j = 4$: $\text{texto}[4] == \text{padrão}[4] \rightarrow i = 5, j = 5$
6. $i = 5, j = 5$: $\text{texto}[5] != \text{padrão}[5]$
 - $j = \text{LPS}[5 - 1] = \text{LPS}[4] = 3$
7. $i = 5, j = 3$: $\text{texto}[5] == \text{padrão}[3] \rightarrow i = 6, j = 4$
8. $i = 6, j = 4$: $\text{texto}[6] == \text{padrão}[4] \rightarrow i = 7, j = 5$
9. $i = 7, j = 5$: $\text{texto}[7] != \text{padrão}[5]$
 - $j = \text{LPS}[5 - 1] = \text{LPS}[4] = 3$
10. $i = 7, j = 3$: $\text{texto}[7] == \text{padrão}[3] \rightarrow i = 8, j = 4$
11. $i = 8, j = 4$: $\text{texto}[8] == \text{padrão}[4] \rightarrow i = 9, j = 5$
12. $i = 9, j = 5$: $\text{texto}[9] != \text{padrão}[5]$
 - $j = \text{LPS}[5 - 1] = \text{LPS}[4] = 3$
13. $i = 9, j = 3$: $\text{texto}[9] == \text{padrão}[3] \rightarrow i = 10, j = 4$
14. $i = 10, j = 4$: $\text{texto}[10] == \text{padrão}[4] \rightarrow i = 11, j = 5$
15. $i = 11, j = 5$: $\text{texto}[11] != \text{padrão}[5]$
 - $j = \text{LPS}[5 - 1] = \text{LPS}[4] = 3$
16. $i = 11, j = 3$: $\text{texto}[11] == \text{padrão}[3] \rightarrow i = 12, j = 4$
17. $i = 12, j = 4$: $\text{texto}[12] == \text{padrão}[4] \rightarrow i = 13, j = 5$
18. $i = 13, j = 5$: $\text{texto}[13] == \text{padrão}[5] \rightarrow i = 14, j = 6$ (padrão encontrado no índice 8 do texto)

Padrão encontrado no índice 8 do texto.

Resumo da Tabela de Movimentos

i	j	texto[i]	padrão[j]	Ação	Próximos i, j
0	0	A	A	i++, j++	1, 1
1	1	B	B	i++, j++	2, 2
2	2	A	A	i++, j++	3, 3
3	3	B	B	i++, j++	4, 4
4	4	A	A	i++, j++	5, 5
5	5	B	C	j = LPS[4]	5, 3
5	3	B	B	i++, j++	6, 4
6	4	A	A	i++, j++	7, 5
7	5	B	C	j = LPS[4]	7, 3
7	3	B	B	i++, j++	8, 4
8	4	A	A	i++, j++	9, 5
9	5	B	C	j = LPS[4]	9, 3
9	3	B	B	i++, j++	10, 4
10	4	A	A	i++, j++	11, 5
11	5	B	C	j = LPS[4]	11, 3
11	3	B	B	i++, j++	12, 4
12	4	A	A	i++, j++	13, 5
13	5	C	C	i++, j++	14, 6

Tabela 1: Execução do algoritmo KMP

2.4 Q4

Algoritmo Boyer-Moore - Heurística do Caráter Ruim

Para entender como o algoritmo Boyer-Moore trabalha no primeiro teste de busca do padrão CANOA no texto IA NA CANOA, RIO ABAIXO, é importante calcular o deslocamento usando a heurística de "caráter ruim".

1. Construção da Tabela de Caráter Ruim

A tabela de caráter ruim armazena a última ocorrência de cada caractere no padrão. Se o caractere não está no padrão, sua última ocorrência é -1 .

Caráter	Última ocorrência
C	0
A	4
N	2
O	3
(outros)	-1

2. Primeiro teste de busca

Alinhamos o padrão CANOA com o início do texto IA NA CANOA, RIO ABAIXO:

Texto: IA NA CANOA, RIO ABAIXO
 CANOA

3. Comparação de caracteres da direita para a esquerda

IA NA CANOA, RIO ABAIXO
CANOA

Comparando da direita para a esquerda: - A (padrão) comparado com N (texto) - Não há correspondência.

4. Deslocamento por Caráter Ruim

O caractere ruim é N no texto.

Consultamos a última ocorrência de N no padrão: - A última ocorrência de N em CANOA é no índice 2.

Calculamos o deslocamento:

$$\text{Deslocamento} = \max(1, j - \text{última ocorrência})$$

Onde j é o índice no padrão do caractere que causou a falha de correspondência.

Aqui, $j = 4$ (índice do caractere A no padrão CANOA) e a última ocorrência de N é 2.

$$\text{Deslocamento} = \max(1, 4 - 2) = \max(1, 2) = 2$$

Portanto, o padrão CANOA será deslocado por 2 posições para a direita.

Aplicação do Deslocamento

Deslocamos o padrão 2 posições para a direita:

IA NA CANOA, RIO ABAIXO
CANOA

Agora, o padrão está alinhado a partir da posição 2 do texto. Podemos continuar a busca a partir dessa nova posição usando o mesmo processo.

Este foi o deslocamento específico por caráter ruim no primeiro teste ao buscar CANOA em IA NA CANOA, RIO ABAIXO.

2.5 Q5

Algoritmo Boyer-Moore - Heurística do Sufixo Bom

Para entender como o algoritmo Boyer-Moore trabalha no primeiro teste de busca do padrão CANOA no texto IA NA CANOA, RIO ABAIXO, é importante calcular o deslocamento usando a heurística de "sufixo bom".

1. Construção da Tabela de Sufixo Bom

A heurística de sufixo bom envolve a criação de uma tabela que indica quanto o padrão pode ser deslocado quando uma falha de correspondência ocorre. O deslocamento depende do maior sufixo do padrão que coincide com um prefixo do padrão.

Padrão: CANOA

Índices:	0	1	2	3	4
Caráter:	C	A	N	O	A

2. Primeiro teste de busca

Alinhamos o padrão CANOA com o início do texto IA NA CANOA, RIO ABAIXO:

Texto: IA NA CANOA, RIO ABAIXO
 CANOA

3. Comparação de caracteres da direita para a esquerda

IA NA CANOA, RIO ABAIXO
 CANOA

Comparando da direita para a esquerda: - A (padrão) comparado com N (texto) - Não há correspondência.

4. Deslocamento por Sufixo Bom

No caso da falha de correspondência no caractere A (padrão) com N (texto), precisamos determinar o maior sufixo do padrão que coincide com um prefixo do padrão.

No padrão CANOA, o maior sufixo que coincide com um prefixo ocorre no índice 4 (caractere A).

Para calcular o deslocamento, consideramos a posição da falha no padrão (índice 4) e a posição da última ocorrência do sufixo correspondente.

Neste caso, o sufixo A (índice 4) não coincide com um prefixo do padrão anterior. Então, consideramos a próxima maior correspondência.

5. Deslocamento

Como o maior sufixo A não coincide com nenhum prefixo anterior no padrão CANOA, o padrão será deslocado completamente.

Portanto, o deslocamento será igual ao comprimento do padrão, que é 5.

Aplicação do Deslocamento

Deslocamos o padrão 5 posições para a direita:

IA NA CANOA, RIO ABAIXO
 CANOA

Agora, o padrão está alinhado a partir da posição 5 do texto. Podemos continuar a busca a partir dessa nova posição usando o mesmo processo.

Este foi o deslocamento específico por sufixo bom no primeiro teste ao buscar CANOA em IA NA CANOA, RIO ABAIXO.

2.6 Q6

Diagrama de Estados para Aho-Corasick

Para construir o diagrama de estados da busca dos termos FACA, FOICE, CABO e CORTE usando o algoritmo Aho-Corasick, consideramos cada caractere de cada termo que leva a um novo estado.

Termos de Busca

- **FACA:**

- F
- FA
- FAC
- FACA

- **FOICE:**

- FO
- FOI
- FOIC
- FOICE

- **CABO:**

- C
- CA
- CAB
- CABO

- **CORTE:**

- CO
- COR
- CORT
- CORTE

Construção do Autômato

1. Iniciamos com o estado inicial, q_0 .

2. Adicionamos estados para **FACA**:

- F (q_1)
- FA (q_2)
- FAC (q_3)
- FACA (q_4)

3. Adicionamos estados para **FOICE**:

- FO (q_5)
- FOI (q_6)
- FOIC (q_7)
- FOICE (q_8)

4. Adicionamos estados para **CABO**:

- C (q_9)
- CA (q_{10})
- CAB (q_{11})
- CABO (q_{12})

5. Adicionamos estados para CORTE:

- CO (q_{13})
- COR (q_{14})
- CORT (q_{15})
- CORTE (q_{16})

Contagem Total de Estados

Reutilizando os estados comuns, a contagem final de estados é:

$$1(q_0) + 4(\text{para "FACA"}) + 4(\text{para "FOICE"}) + 4(\text{para "CABO"}) + 4(\text{para "CORTE"})$$

Portanto, o diagrama de estados conterá 17 estados distintos.

2.7 Q7

Complexidade de Tempo do Algoritmo Aho-Corasick

O algoritmo Aho-Corasick é utilizado para buscar múltiplos padrões em um texto de maneira eficiente. A análise de sua complexidade de tempo considera duas fases principais: a construção do autômato de Aho-Corasick e a busca no texto utilizando esse autômato.

1. Construção do Autômato de Aho-Corasick

- **Construção do trie (árvore de prefixos):**
 - Para m padrões de comprimento total P (a soma dos comprimentos de todos os padrões), a construção do trie requer $O(P)$ tempo.
- **Adição das falhas (funcionalidade de fallback):**
 - A construção das falhas do autômato também leva $O(P)$ tempo.

Portanto, a construção do autômato de Aho-Corasick tem uma complexidade de tempo de $O(P)$.

2. Busca no Texto Utilizando o Autômato

A busca no texto de comprimento n utilizando o autômato processa cada caractere do texto exatamente uma vez, resultando em uma complexidade de tempo de $O(n)$.

3. Complexidade Total

A complexidade total do algoritmo Aho-Corasick é a soma das complexidades de construção do autômato e de busca no texto. Portanto, a complexidade de tempo do algoritmo Aho-Corasick para buscar m padrões em um texto de n caracteres é:

$$O(P + n)$$

onde P é o comprimento total de todos os padrões.

2.8 Q8

		S	U	T	U	R	A	S
	0	1	2	3	4	5	6	7
C	1	1	2	3	4	5	6	7
U	2	2	1	2	3	4	5	6
L	3	3	2	2	3	4	5	6
T	4	4	3	2	3	4	5	6
U	5	5	4	3	2	3	4	5
R	6	6	5	4	3	2	3	4
A	7	7	6	5	4	3	2	3

2.9 Q9

		S	A	C	I
	0	1	2	3	4
C	1	1	2	2	3
A	2	2	1	2	3
P	3	3	2	2	3
I	4	4	3	3	2
M	5	5	4	4	3

2.10 Q10

Busca de "IARA" em "A IARA AMARRA A ARARA DE ARARAQUARA" usando Boyer-Moore

Padrão e Texto

Padrão: IARA

Texto: A IARA AMARRA A ARARA DE ARARAQUARA

Tabela de Caráter Ruim (DCR)

Caractere	DCR
A	1
R	2
I	3

Tabela de Sufixo Bom (DSB)

Sufixo	DSB
A	1
RA	3
ARA	4
IARA	5

Passos da Busca

1. Alinhamento inicial:

```
A IARA AMARRA A ARARA DE ARARAQUARA
IARA
```

Comparação falha no caractere 'A' do texto com 'A' do padrão.

$DCR = 1, \quad DSB = N/A \rightarrow \text{Deslocamento: } \boxed{1}$

2. Alinhamento após deslocamento:

```
A IARA AMARRA A ARARA DE ARARAQUARA
IARA
```

Padrão encontrado.

Deslocamento: $\boxed{1}$

3. Alinhamento após deslocamento:

```
A IARA AMARRA A ARARA DE ARARAQUARA
IARA
```

Comparação falha no caractere 'A' do texto com 'I' do padrão.

$DCR = 1, \quad DSB = N/A \rightarrow \text{Deslocamento: } \boxed{1}$

4. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'A' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

5. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'T' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

6. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'A' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

7. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'T' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

8. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'A' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

9. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'I' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

10. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'A' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

11. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'I' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1

12. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Padrão encontrado.

Deslocamento: 1

13. Alinhamento após deslocamento:

A IARA AMARRA A ARARA DE ARARAQUARA
IARA

Comparação falha no caractere 'A' do texto com 'I' do padrão.

DCR = 1, DSB = N/A → Deslocamento: 1