INSTITUTO FEDERAL
Norte de Minas Gerais

Projeto II - Técnicas de Busca e Ordenação

Equipe:
André Felipe
João Kennedy

Desafios escolhidos

As seguintes tarefas que a equipe escolheu para executar

- KMP: Knuth-Morris-Pratt
- Aho-Corasick
- Análise de Desempenho

KMP

Leitura e Busca de Padrões - KMP

1. Os padrões são colocados dentro de um vetor de sequência de caracteres;
2. Um laço permite iterar por cada padrão;
3. Dentro desse laço, o arquivo de texto é lido e o padrão atual é buscado.

- O arquivo de texto é lido na estrutura de Bufferização:
  - Um Buffer é declarado como uma sequência de caracteres, de tamanho pré-definido;
  - O arquivo é lido completamente na mesma quantidade de padrões que estão sendo buscados.

- A cada iteração de leitura, um “bloco” de texto é lido;

- Nesse bloco, o padrão atual da iteração é buscado.

Criação da tabela de prefixos:

```
29  vector<int> LPS(tamPattern, 0);
30  int j = 0;
31  int i = 1;
32
33  while (i < tamPattern) {
34
35      if (pattern[i] == pattern[j]) {
36          j++;
37          LPS[i] = j;
38          i++;
39      } else {
40          if (j != 0) {
41              j = LPS[j - 1];
42          } else {
43              LPS[i] = 0;
44              i++;
45          }
46      }
47  } //fim do while
48
49  return LPS; //retorna o vetor (tabela) de prefixos
```

Padrão: **ABCDABCA**

j = 0

i = 1

A	B	C	D	A	B	C	A
j	i						
0	1	2	3	4	5	6	7
0							

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
j		i					
0	1	2	3	4	5	6	7
0	0						

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
j			i				
0	1	2	3	4	5	6	7
0	0	0					

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
j				i			
0	1	2	3	4	5	6	7
0	0	0	0				

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
	j				i		
0	1	2	3	4	5	6	7
0	0	0	0	1			

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
		j				i	
0	1	2	3	4	5	6	7
0	0	0	0	1	2		

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
			j				i
0	1	2	3	4	5	6	7
0	0	0	0	1	2	3	

Padrão: **ABCDABCA**

j = LPS [j - 1]

i Não Incrementa

A	B	C	D	A	B	C	A
j							i
0	1	2	3	4	5	6	7
0	0	0	0	1	2	3	

Padrão: **ABCDABCA**

A	B	C	D	A	B	C	A
	j						i
0	1	2	3	4	5	6	7
0	0	0	0	1	2	3	1

Busca de Padrões no Texto

1. Passagem por referência do texto, padrão e linha;
2. Chamada do LPS passando o padrão e o seu tamanho;
3. Laço for que percorre todo o texto pelo seu tamanho;
4. Dentro do laço verifica se o padrão é igual ao texto e de mesmo tamanho do padrão procurado ;
5. Variável posição inicial e final percorre o padrão encontrado até o espaço;
6. Função que imprime passando posição inicial, final e a linha;
7. Caso o padrão seja diferente do texto, verifica se o “ j “ é diferente de 0:
    - j = LPS [j - 1] ;
8. Caso contrário, (j == 0), incrementa “ i ”.

Aho-Corasick

Estrutura utilizada

```cpp
//estrutura dos nós da Árvore Trie
struct Node {
    unordered_map<char, Node*> filho;
    Node* falha;
    vector<int> saidas;

    //construtor que inicia todos os nós com falha = null
    Node() : falha(nullptr) {}

};
```

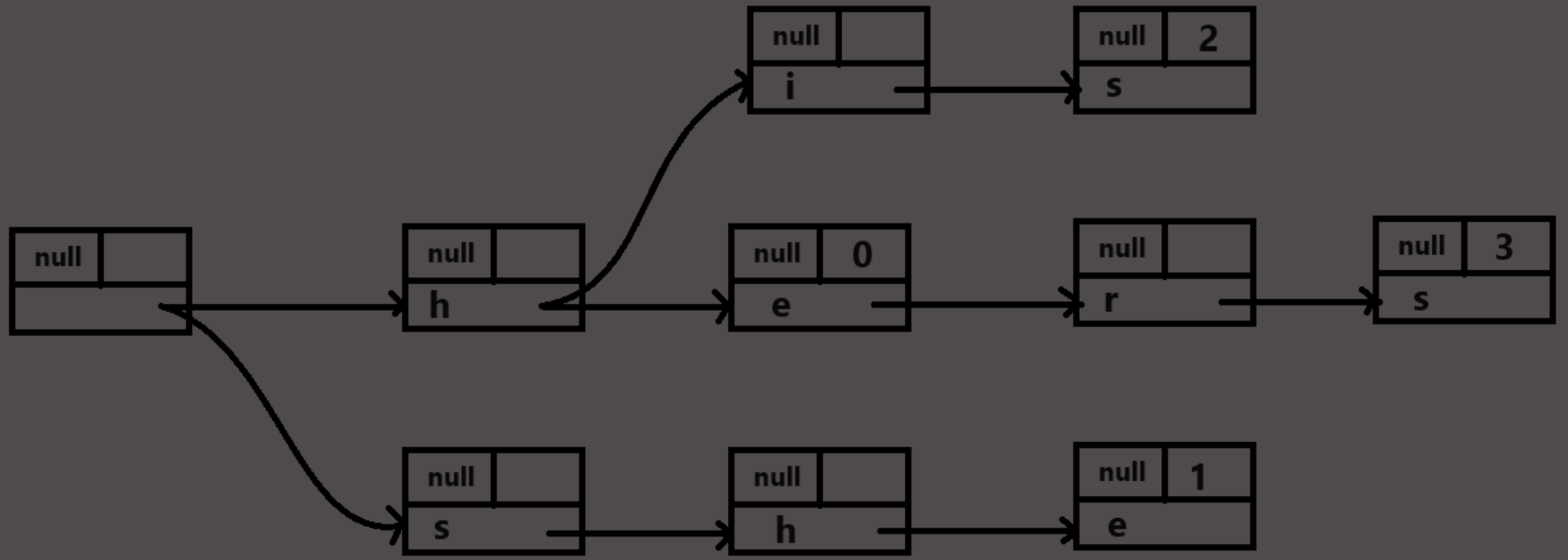
Leitura e Busca de Padrões - Aho-Corasick

1. Os padrões são colocados dentro de um vetor de strings;
2. A árvore de prefixos é criada a partir deles;
3. A função de falha, que usa BFS (Busca por Largura), é invocada;
4. O arquivo é lido sequencialmente, linha a linha, e os padrões são buscados;

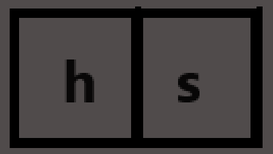
- O arquivo de texto é lido na estrutura de Bufferização:
  - Um Buffer é declarado como uma estrutura deque<string>;
  - Cada linha do arquivo texto é adicionada separadamente nesse deque;
  - O arquivo é lido somente uma única vez.

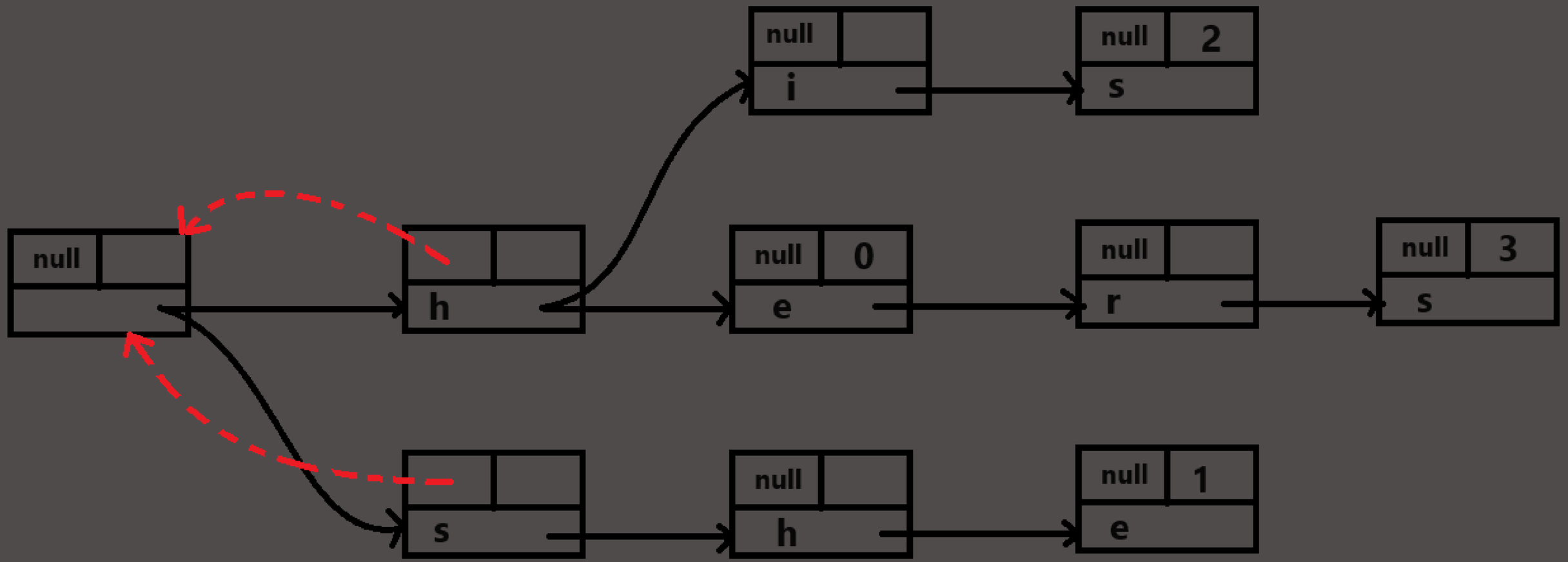
- A cada iteração de leitura de linha, o algoritmo realiza a busca de padrões simultaneamente.

```
                                0       1       2       3
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```



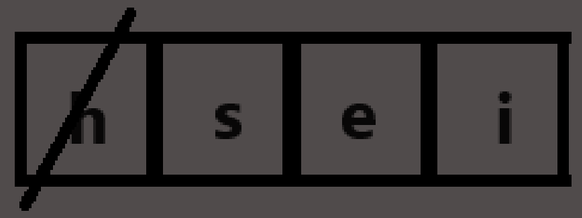
```
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```

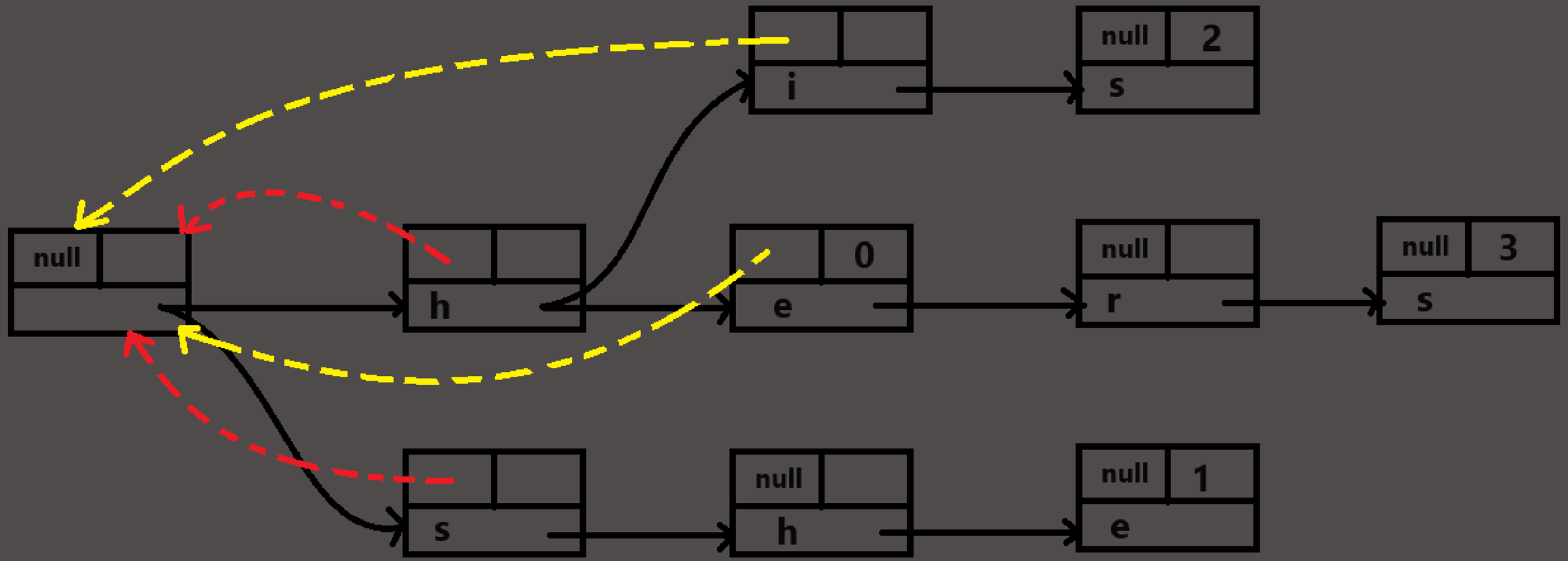
queue<Node*> fila = 



```
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```

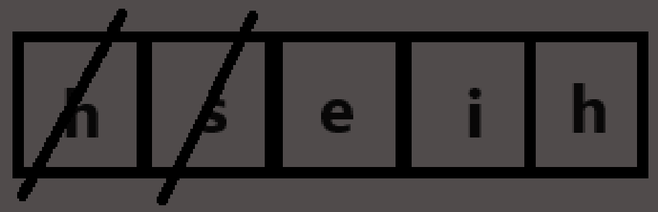
queue<Node*> fila =

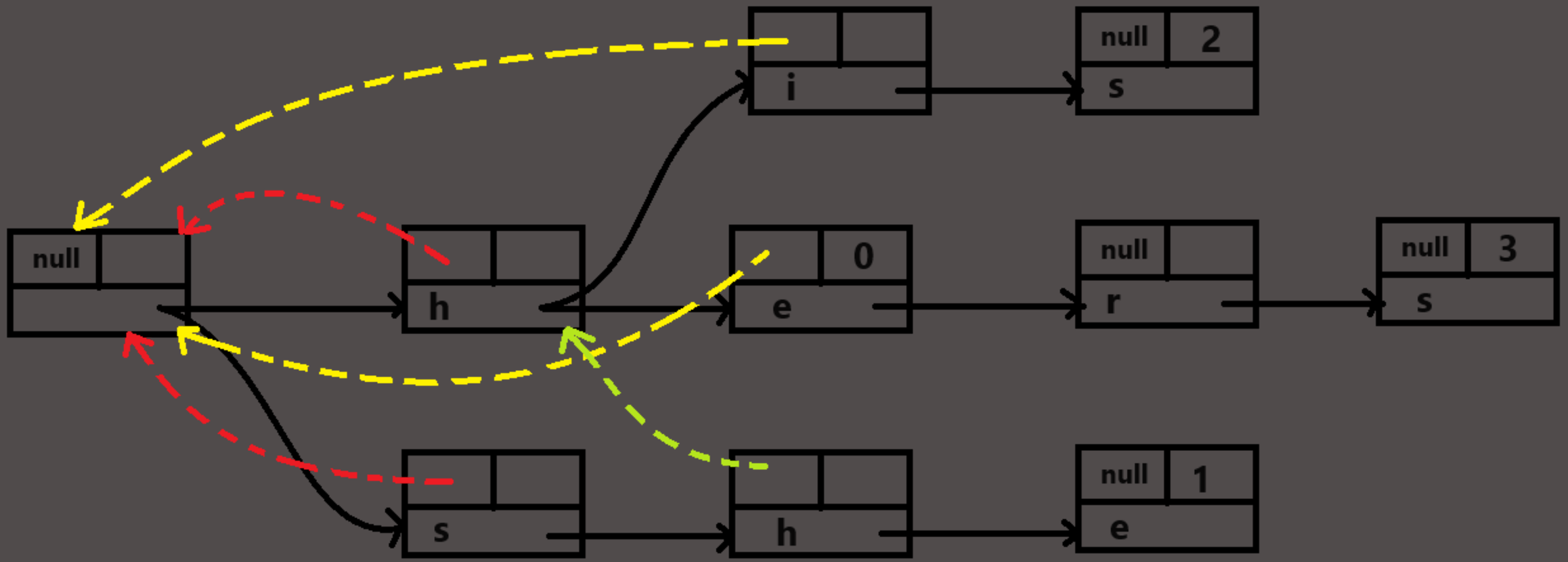




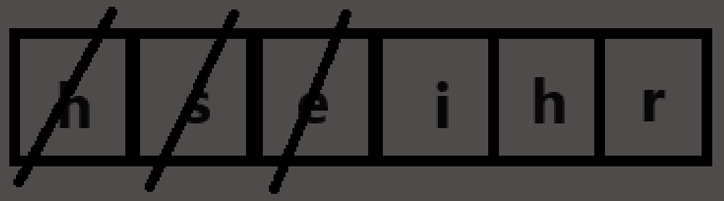
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados

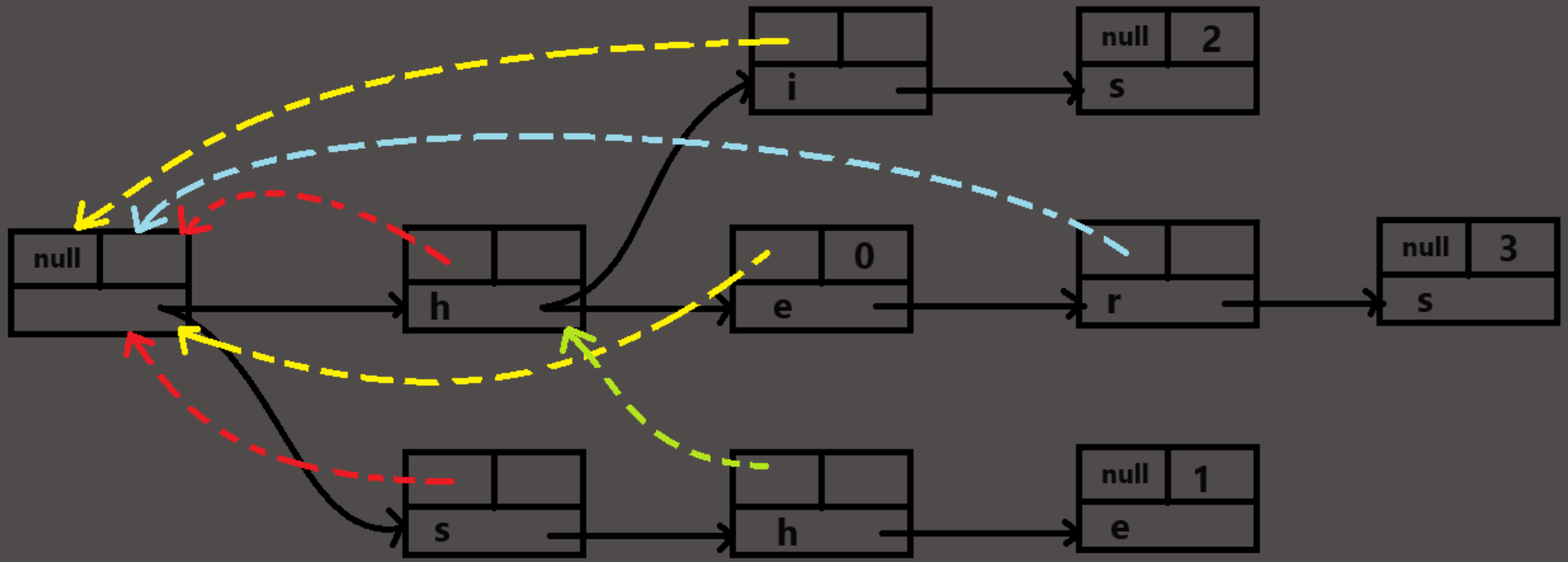
queue<Node*> fila =





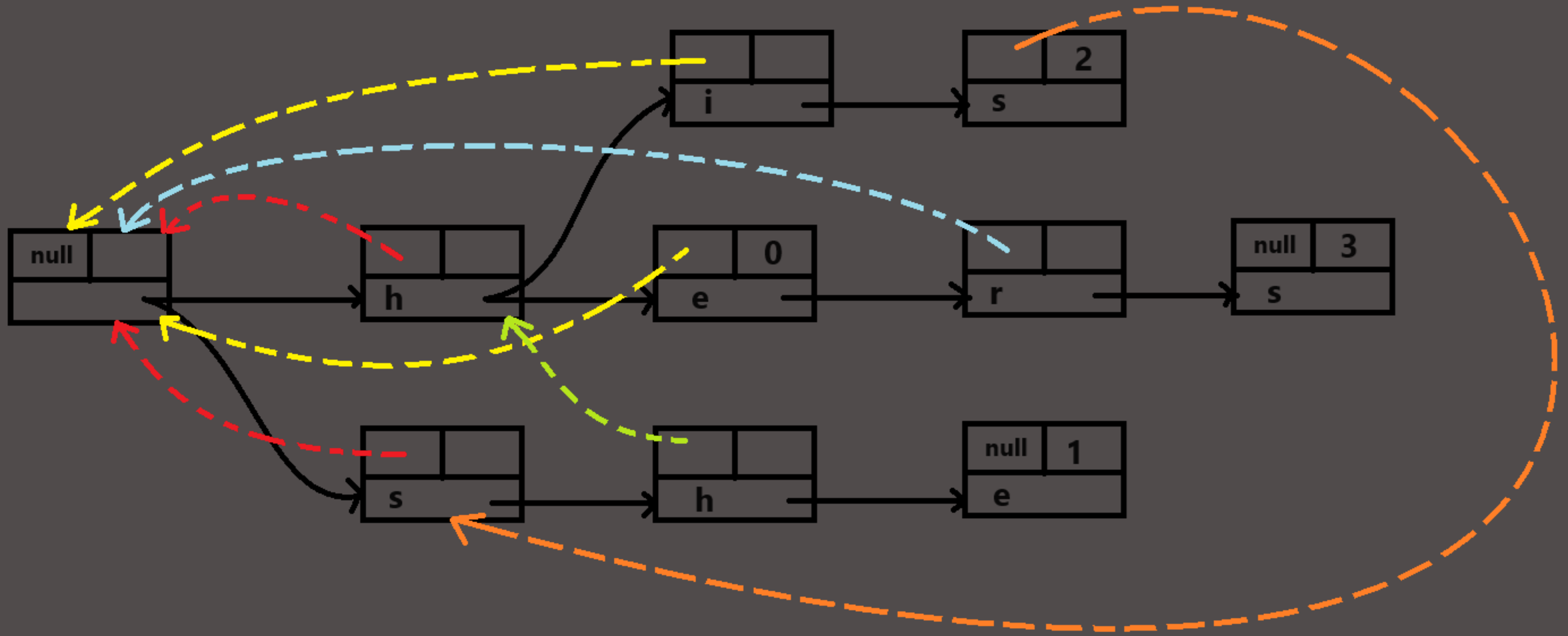
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados

queue<Node*> fila = 

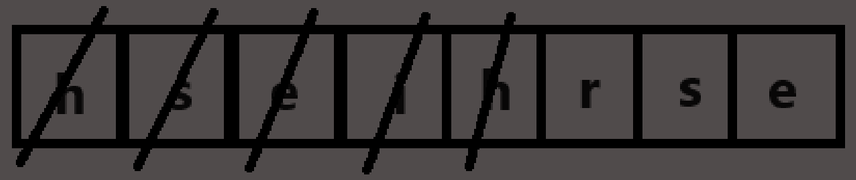


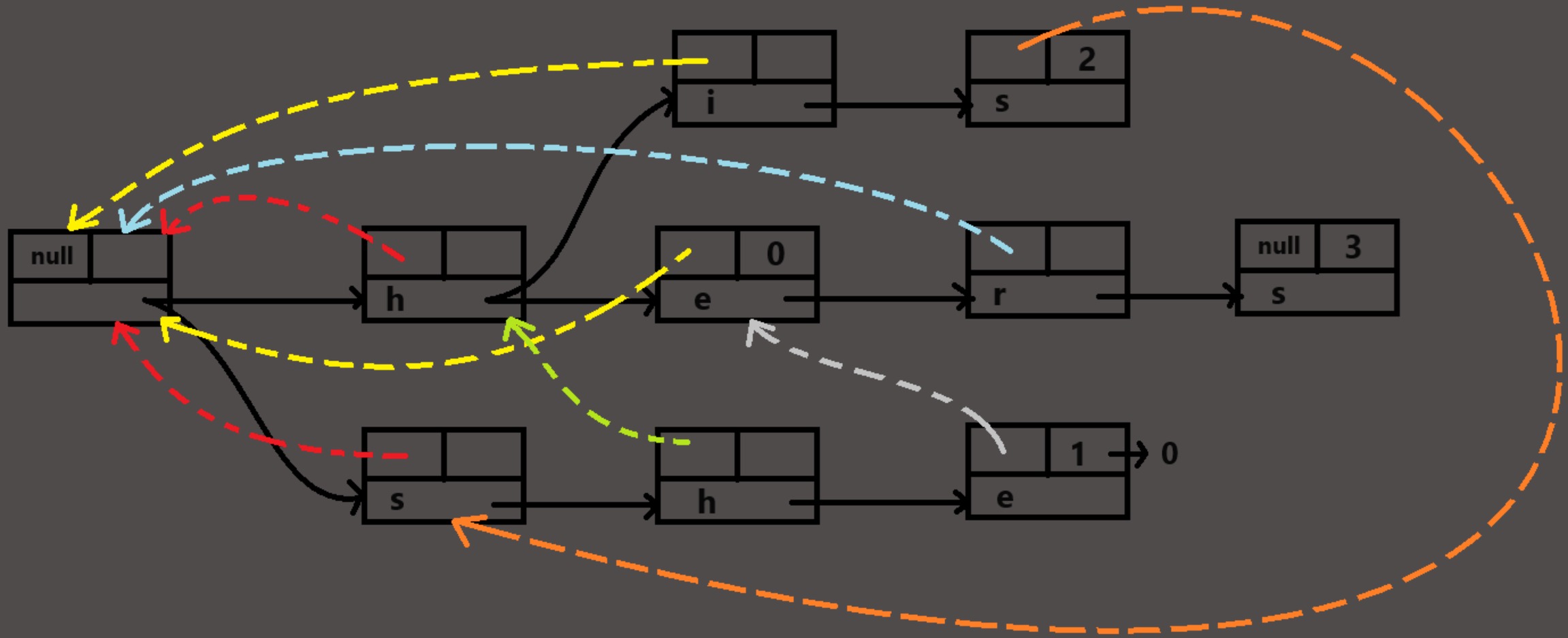
```
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```

queue<Node*> fila =

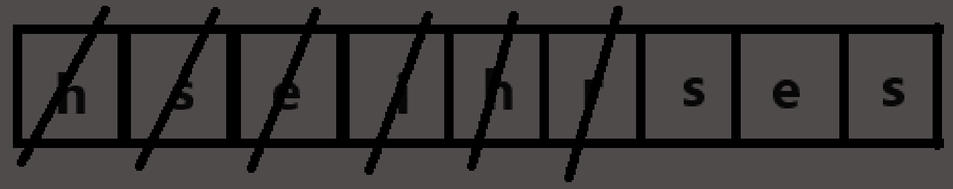


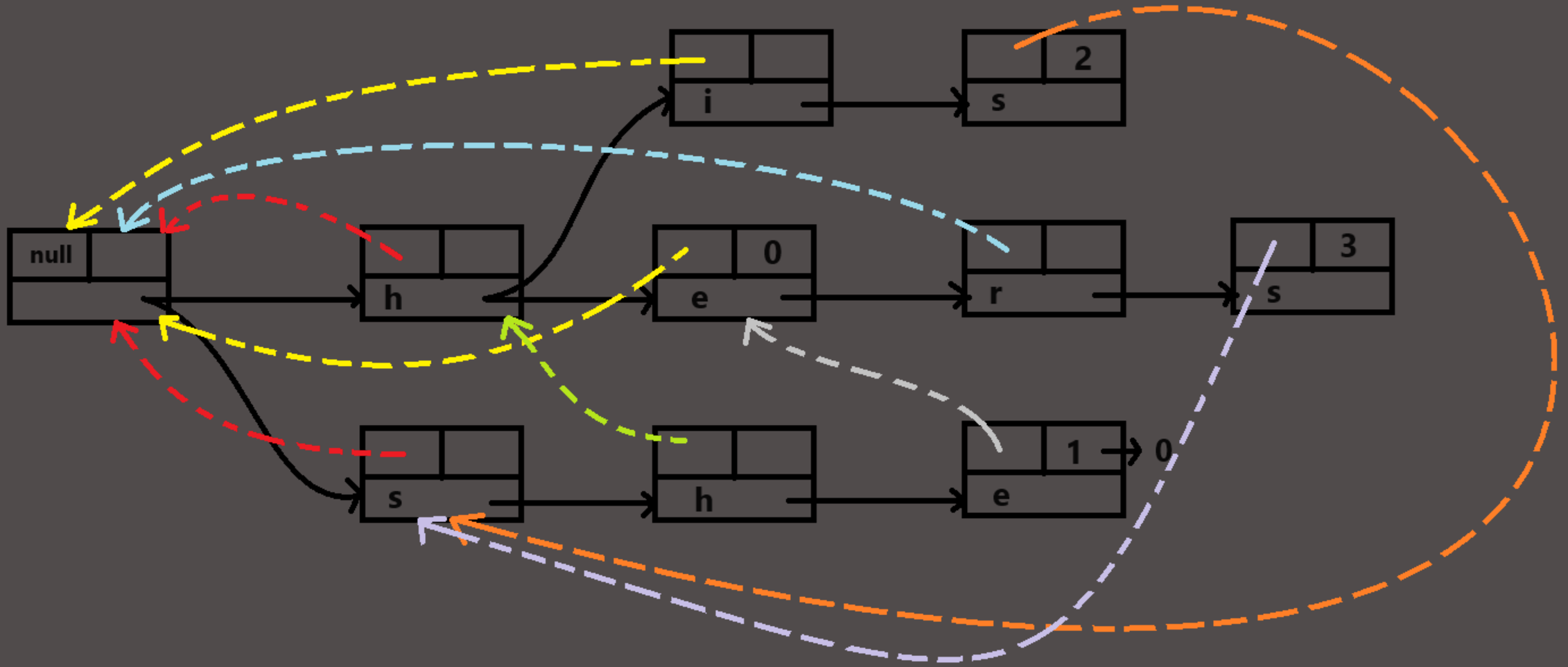
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados

queue<Node*> fila = 



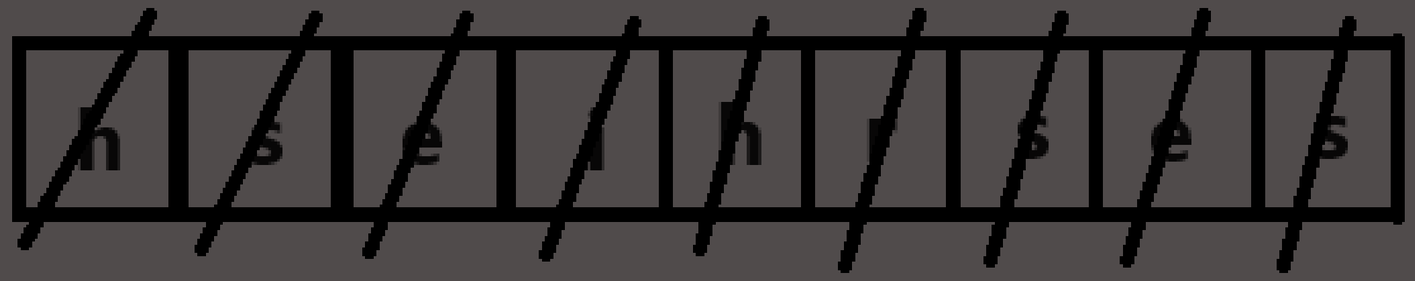
```
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```

queue<Node*> fila = 



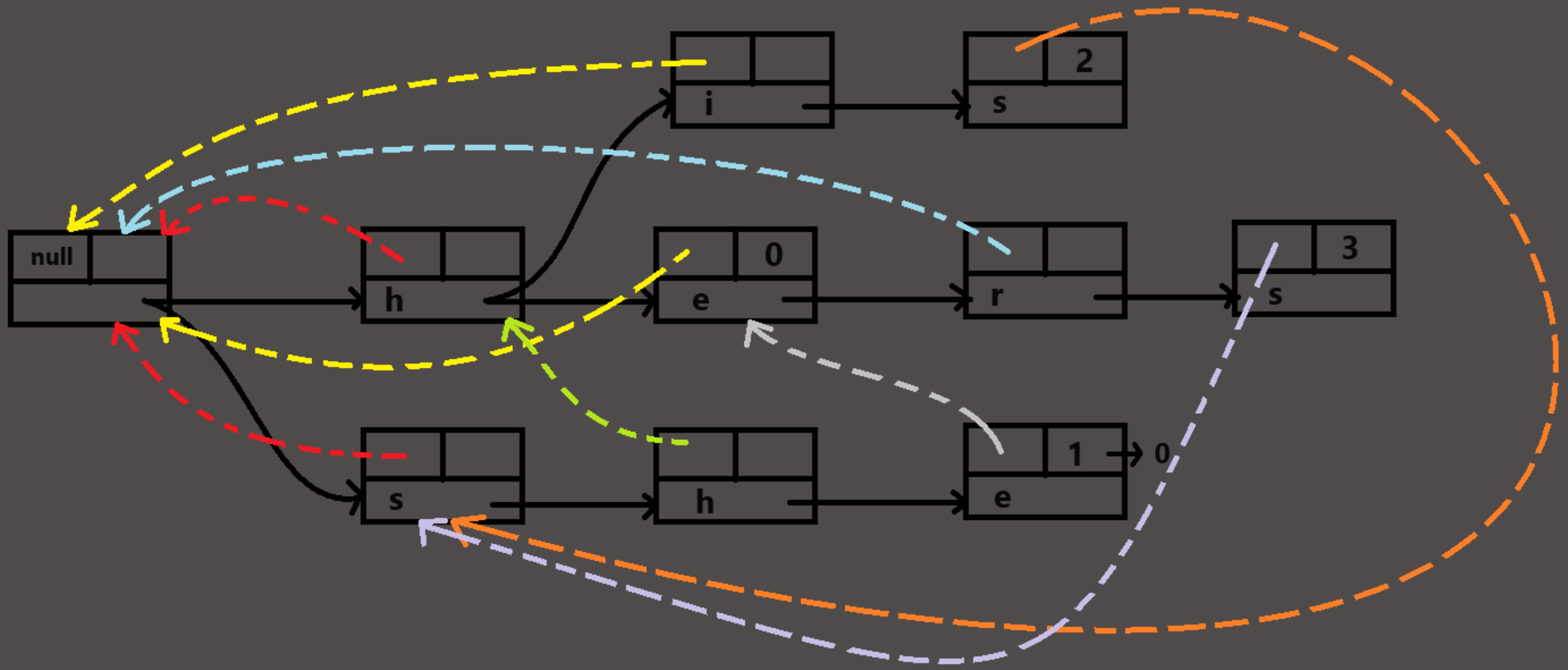
```
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```

queue<Node*> fila =



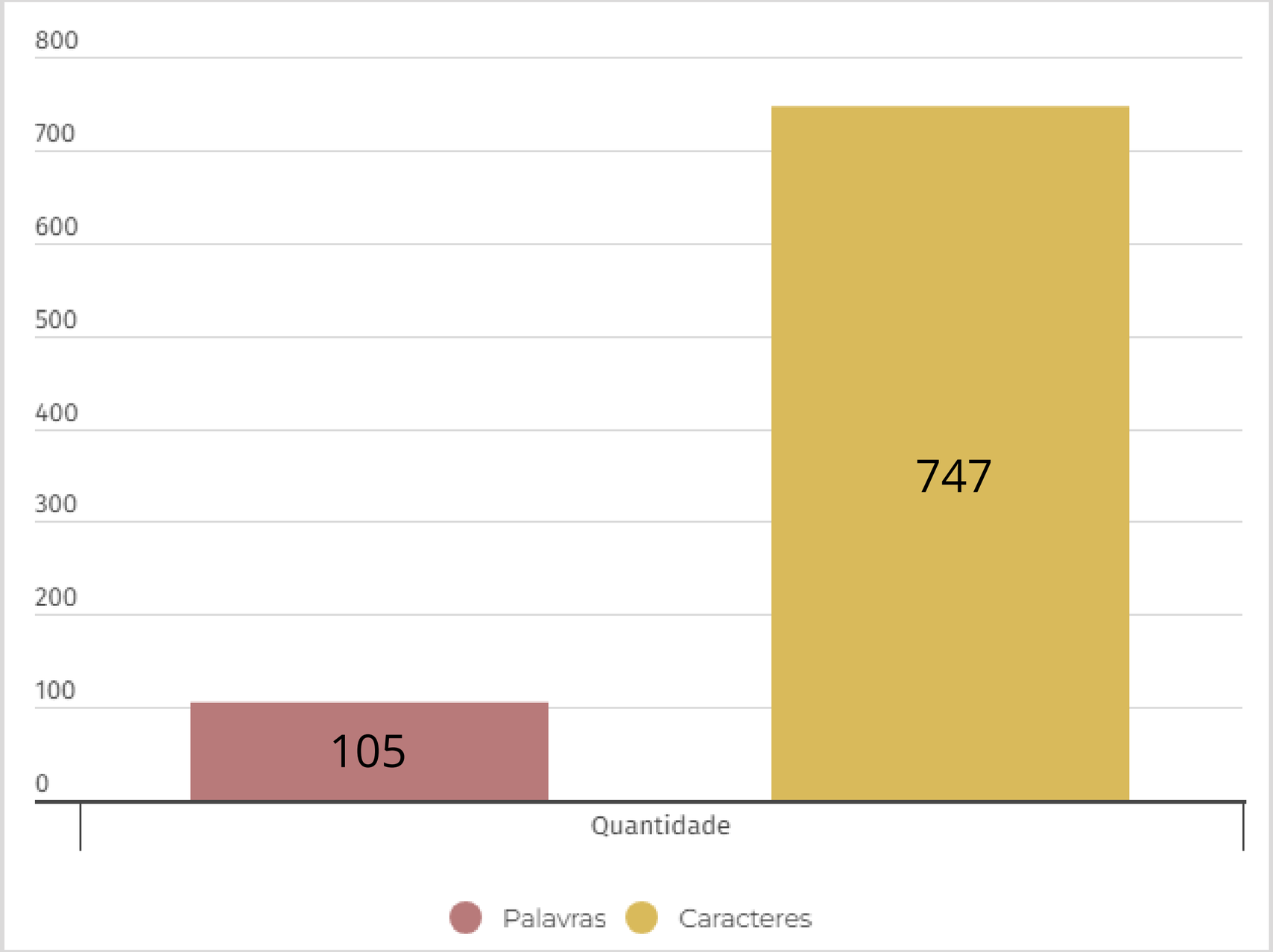
0 1 2 3

```
vector<string> patterns = {"he", ”she”, ”his”, ”hers”}; //padrões buscados
```

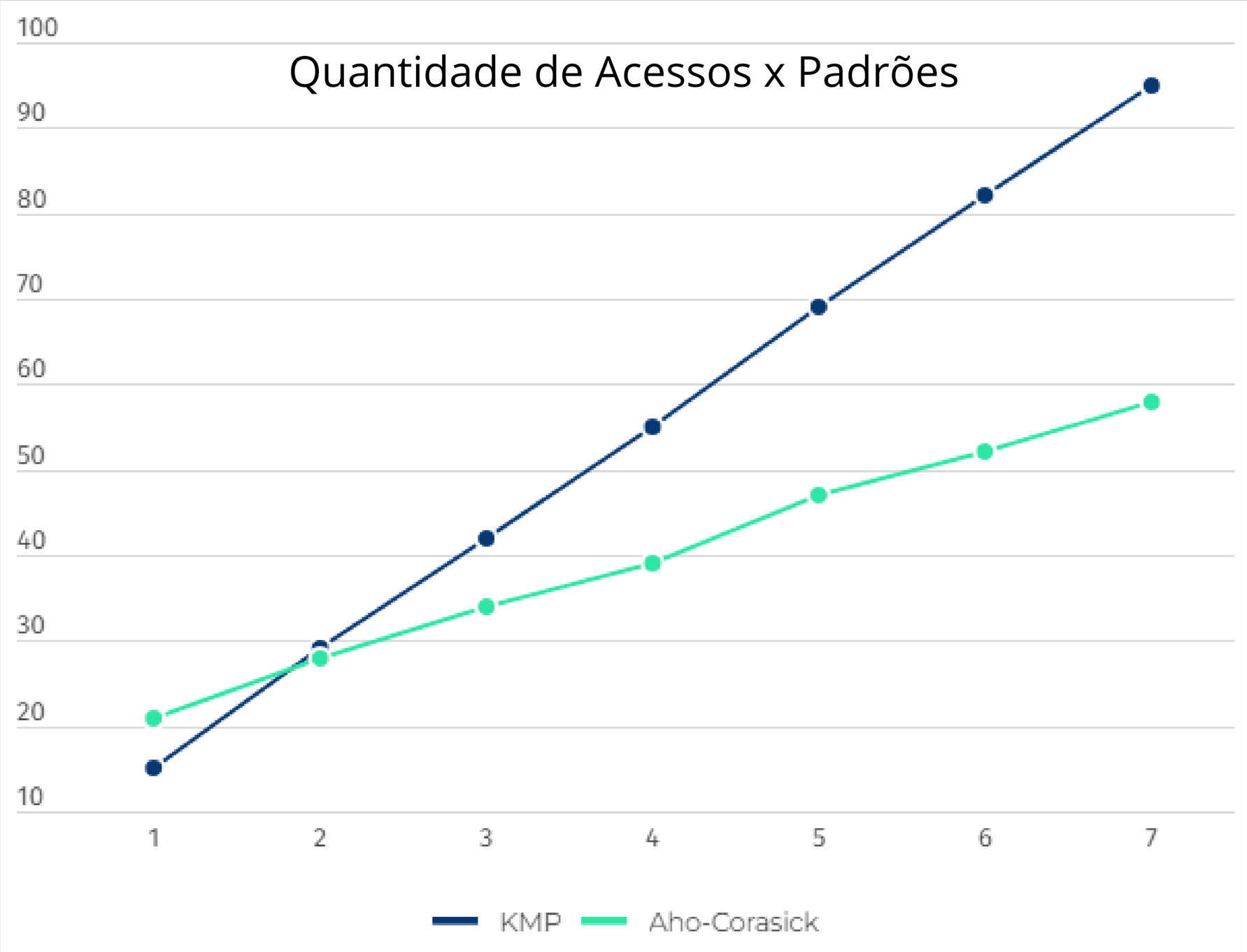


Análise de Desempenho

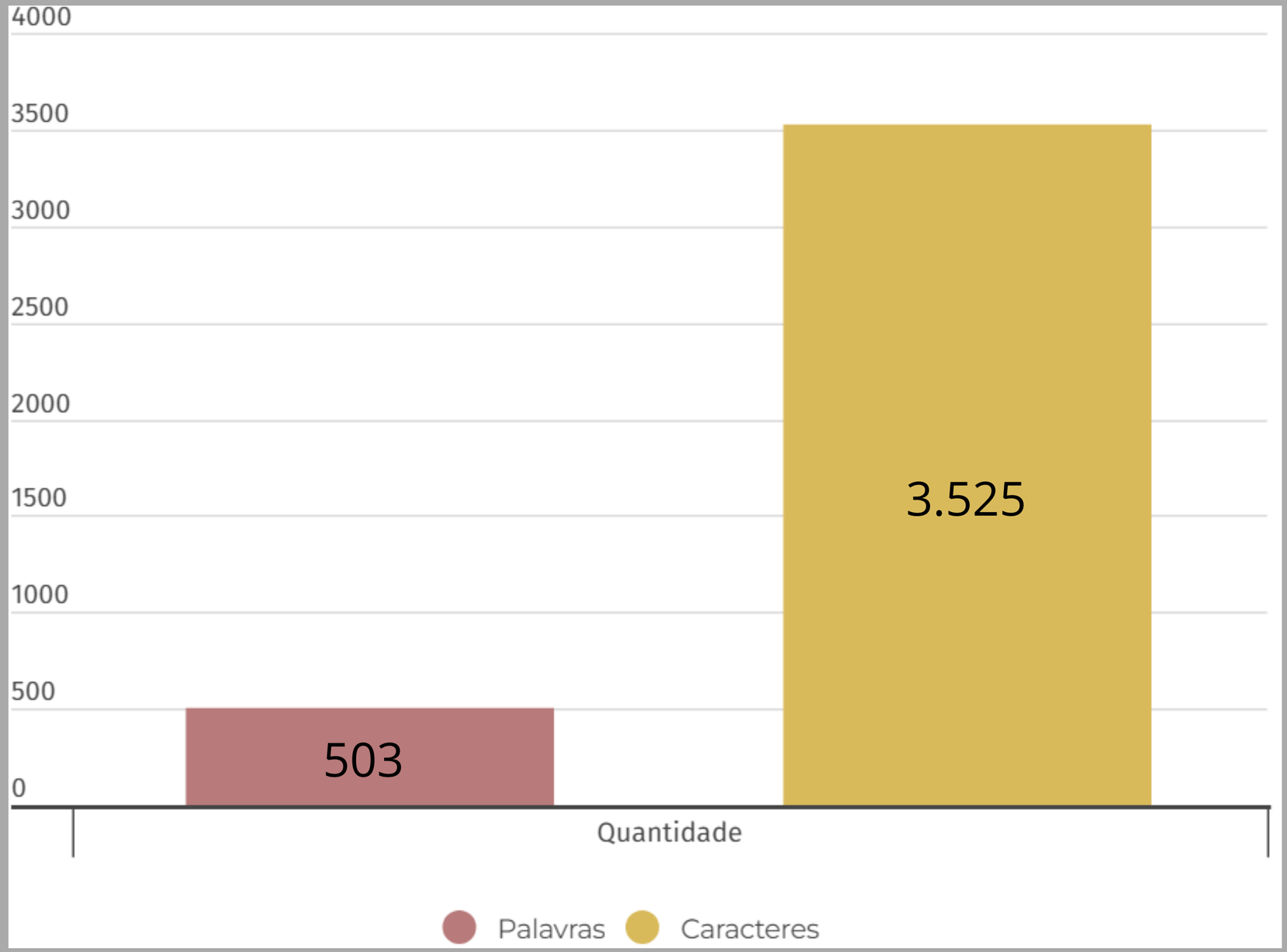
Teste 01:


800
700
600
500
400
300
200
100
0
747
105
Quantidade
Palavras
Caracteres

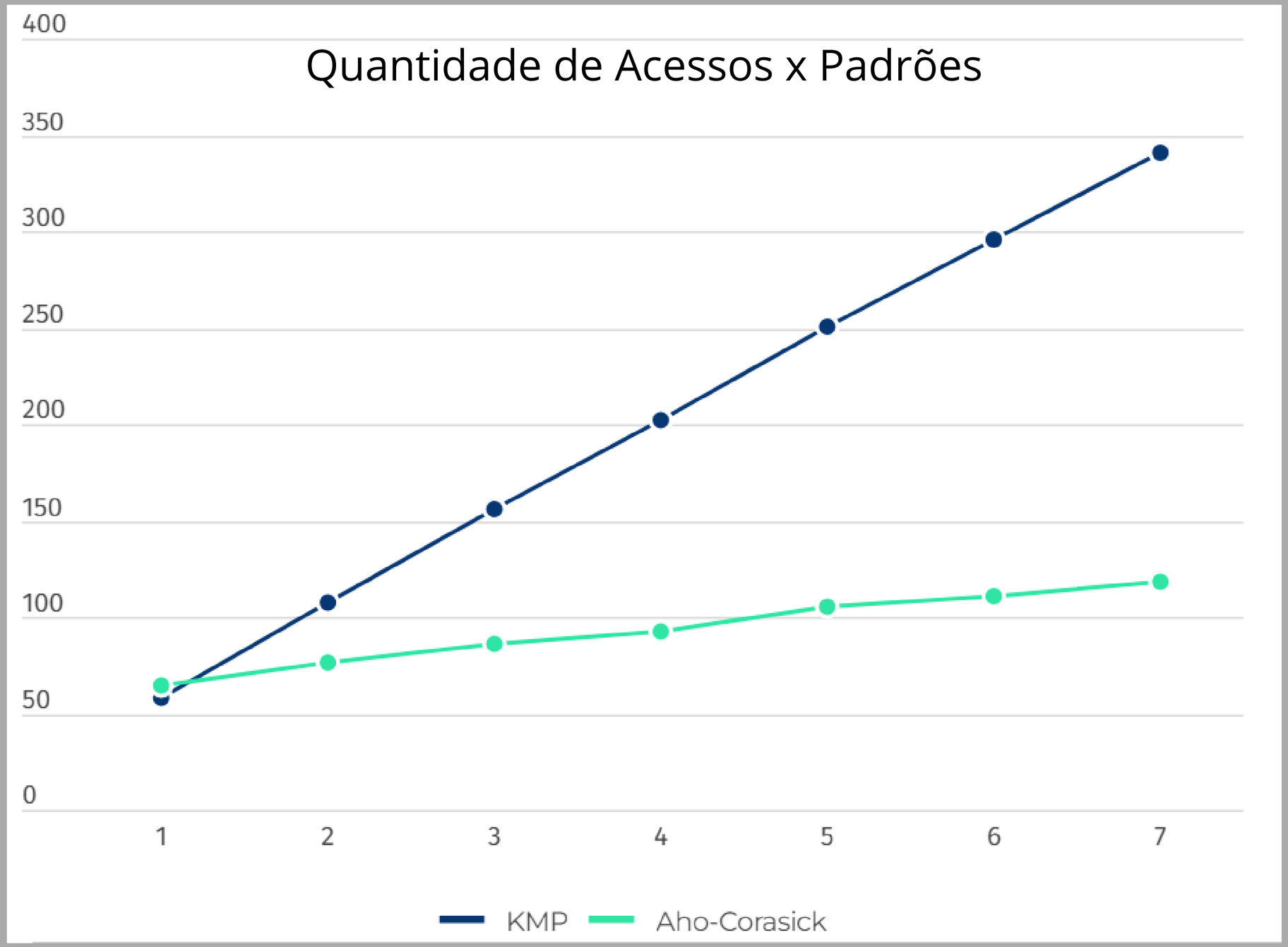
Teste 01:


Quantidade de Acessos x Padrões
100
90
80
70
60
50
40
30
20
10
1
2
3
4
5
6
7
KMP
Aho-Corasick

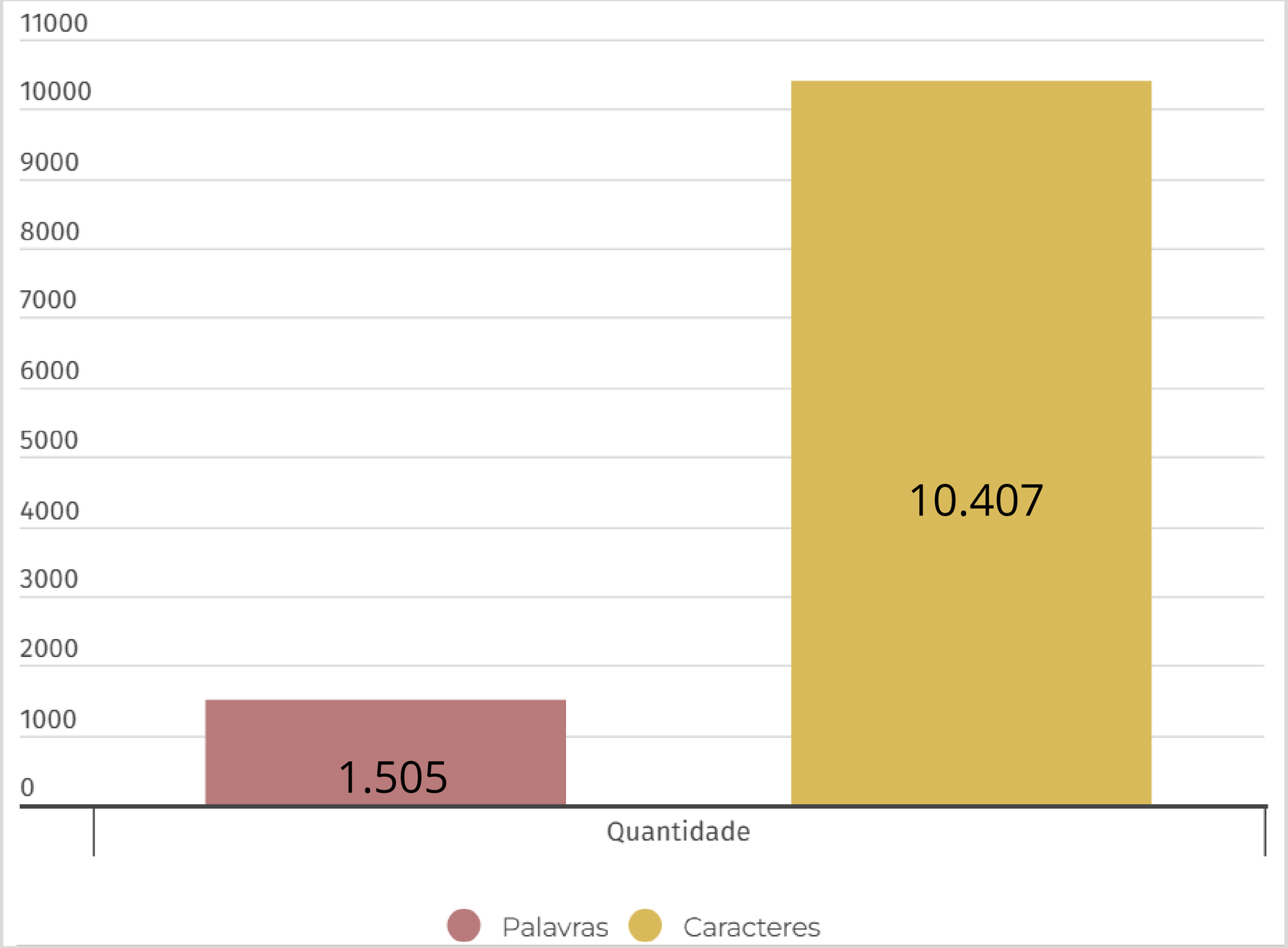
Teste 02:


4000
3500
3000
2500
2000
1500
1000
500
0
503
3.525
Quantidade
Palavras
Caracteres

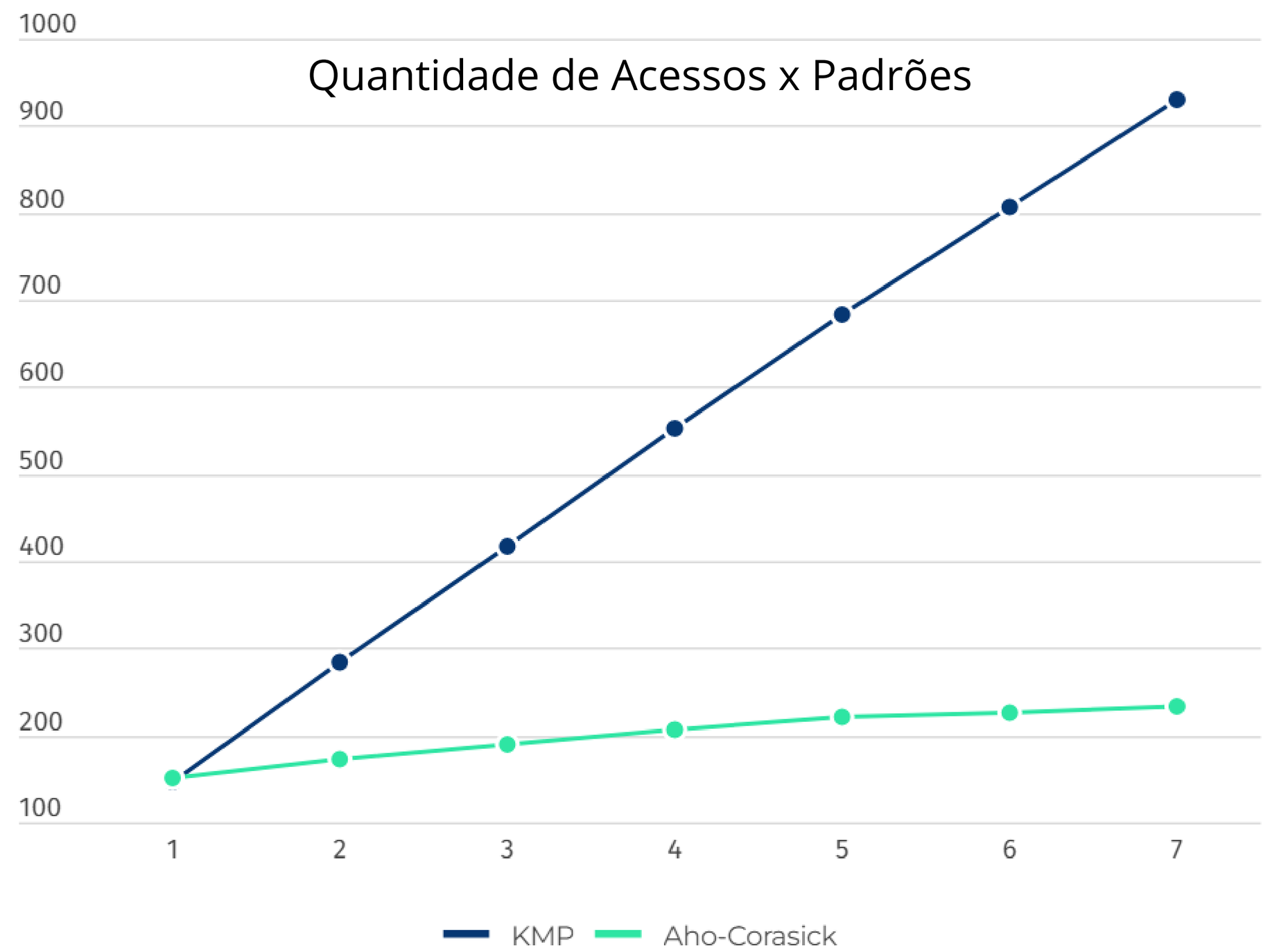
Teste 02:


Quantidade de Acessos x Padrões
400
350
300
250
200
150
100
50
0
1
2
3
4
5
6
7
KMP
Aho-Corasick

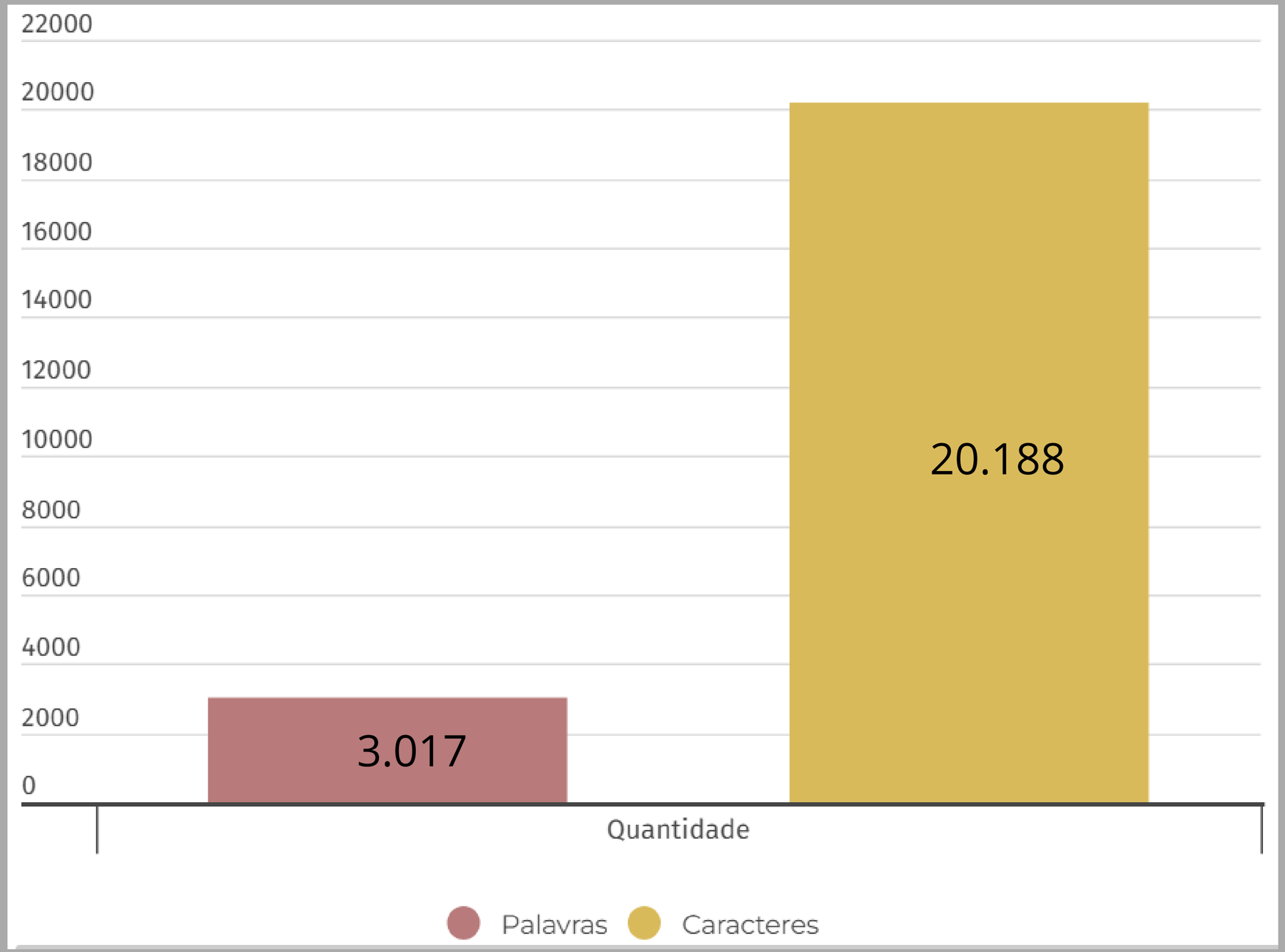
Teste 03:


11000
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
1.505
10.407
Quantidade
Palavras
Caracteres

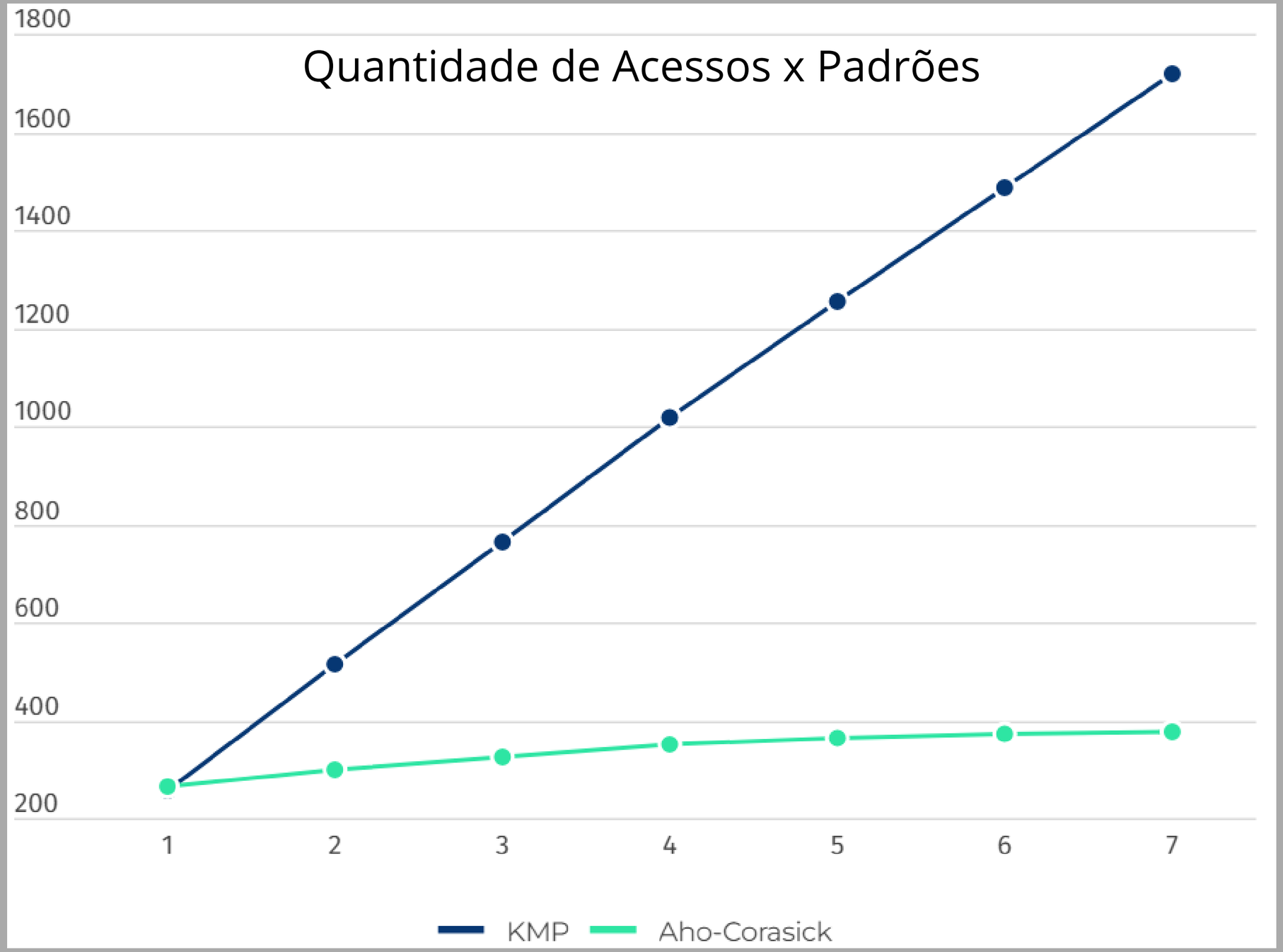
Teste 03:


Quantidade de Acessos x Padrões
1000
900
800
700
600
500
400
300
200
100
1
2
3
4
5
6
7
KMP
Aho-Corasick

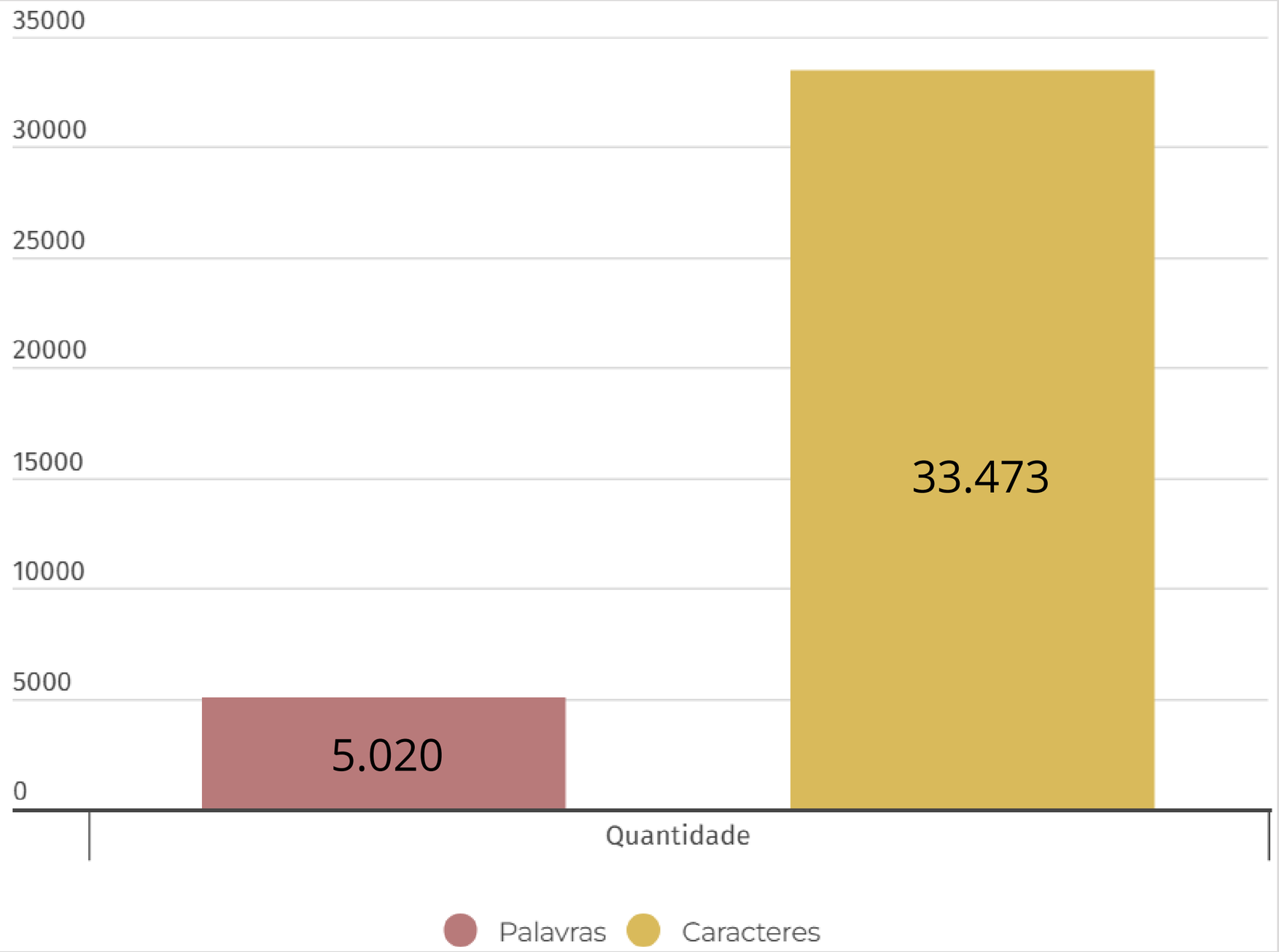
Teste 04:


22000
20000
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
3.017
20.188
Quantidade
Palavras
Caracteres

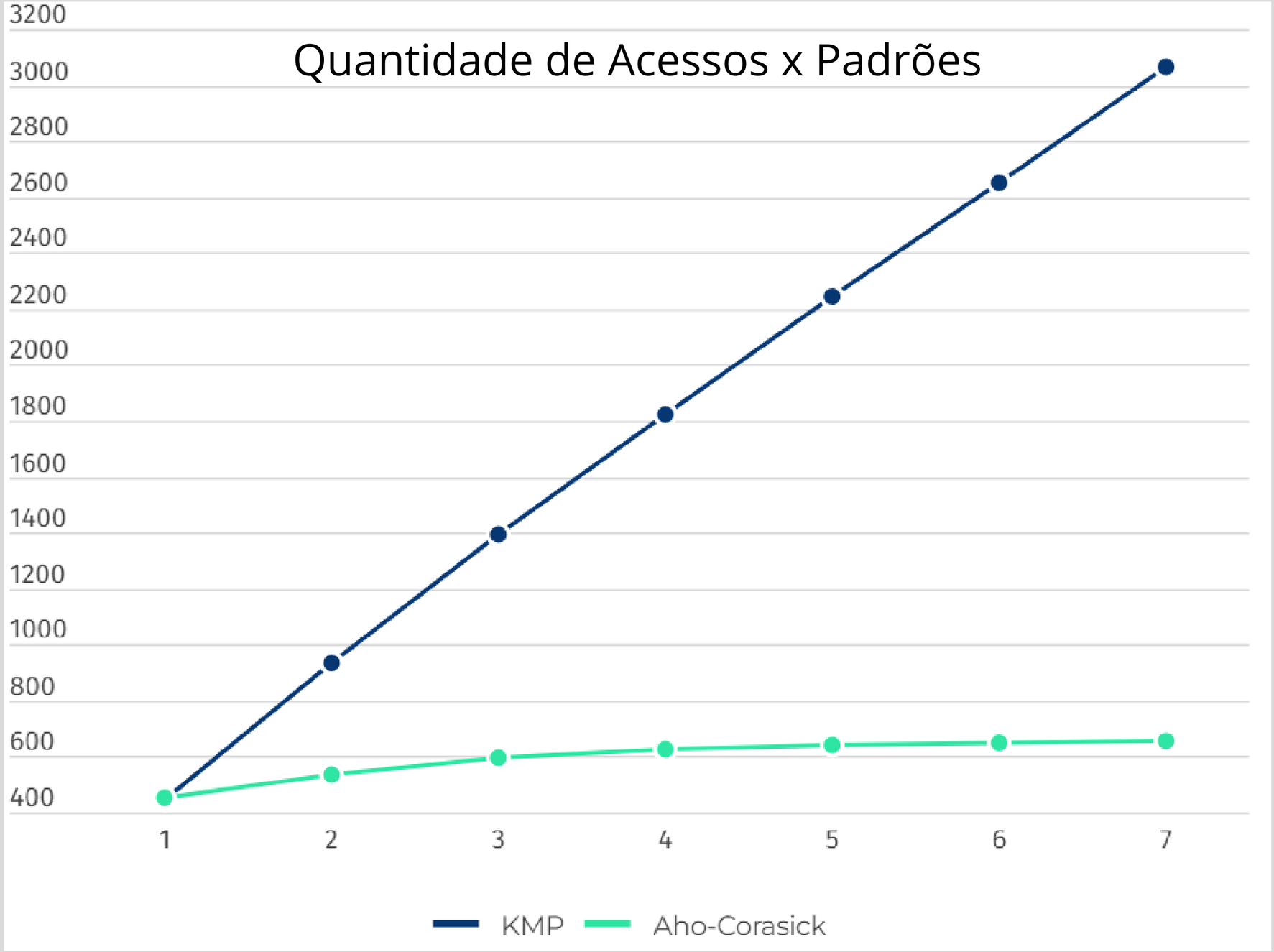
Teste 04:


Quantidade de Acessos x Padrões
1800
1600
1400
1200
1000
800
600
400
200
1
2
3
4
5
6
7
KMP
Aho-Corasick

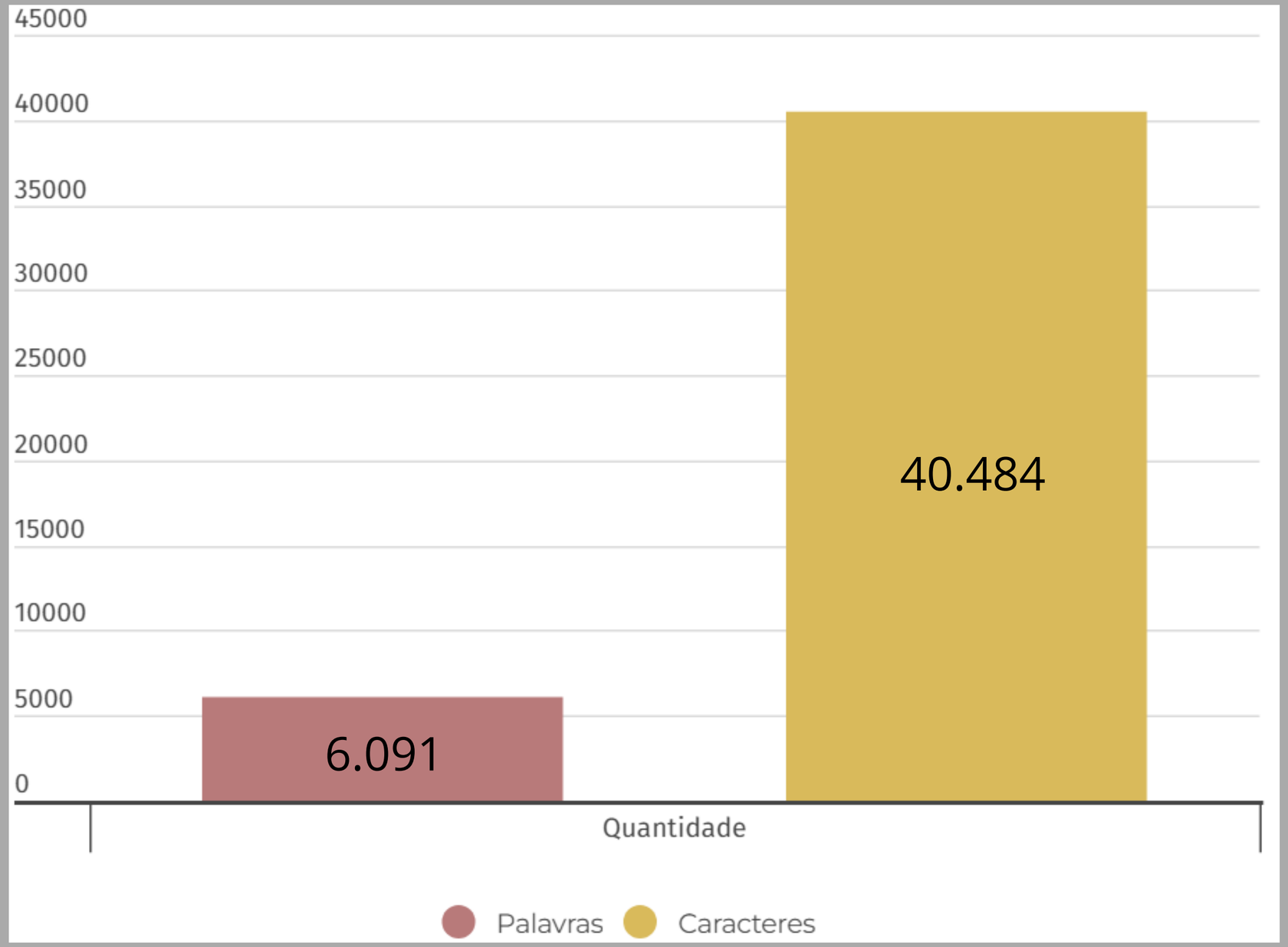
Teste 05:


35000
30000
25000
20000
15000
10000
5000
0
5.020
33.473
Quantidade
Palavras
Caracteres

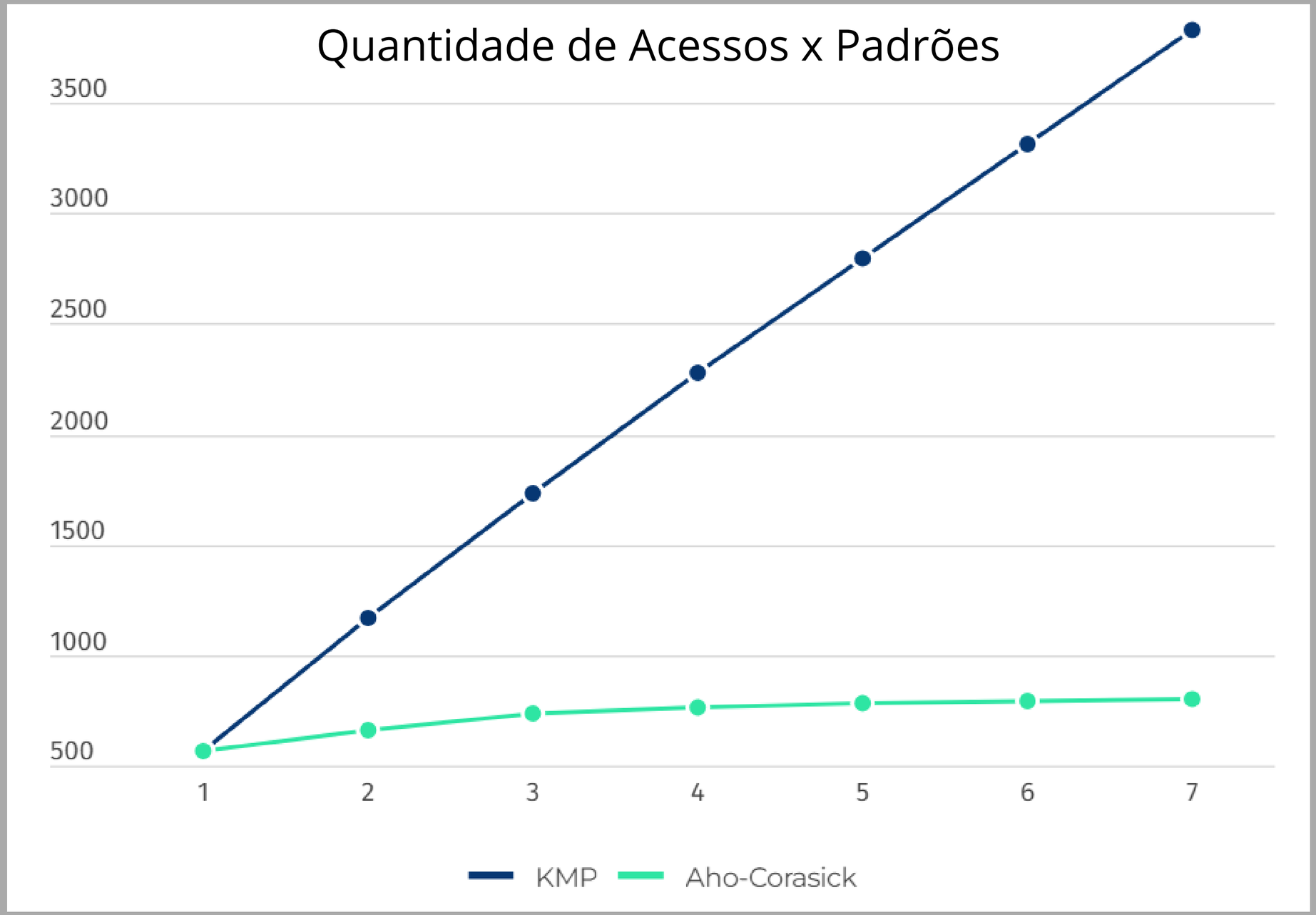
Teste 05:


Quantidade de Acessos x Padrões
3200
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
1
2
3
4
5
6
7
KMP
Aho-Corasick

Teste 06:


45000
40000
35000
30000
25000
20000
15000
10000
5000
0
6.091
40.484
Quantidade
Palavras
Caracteres

Teste 06:


Quantidade de Acessos x Padrões
3500
3000
2500
2000
1500
1000
500
1
2
3
4
5
6
7
KMP
Aho-Corasick

Conclusões

Conclusões

- Eficiência:
  - O algoritmo KMP é eficiente para a busca de apenas 1 padrão;
  - O algoritmo Aho-Corasick é eficiente para busca de diversos padrões, pois busca de forma simultânea.

- Acessos:
  - À medida que aumenta a quantidade de padrões buscados, o KMP mantém um crescimento constante, enquanto o Aho-Corasick começa a adquirir uma estabilidade a partir de buscas com 5 padrões.

- Tamanho dos textos:
  - Independente da quantidade de caracteres, ou de palavras, as buscas mantiveram um caráter de acessos semelhante, o que é evidenciado pelos gráficos.

Referências

1. AED3 12 07 Casamento de padrões por Aho Corasick. Disponível em: <https://www.youtube.com/watch?v=kKQLjWFf4nE&t=397s>. Acesso em: 29 nov. 2023.
2. Aho-Corasick Algorithm | Implementation | Advanced DS | Text Processing. Disponível em: <https://www.youtube.com/watch?v=8T8N2fDgzoc&t=1s>. Acesso em: 29 nov. 2023.
3. BFS - Algoritmo de Busca em Largura - Algoritmos em Grafos. Disponível em: <https://www.youtube.com/watch?v=b7guo8KcJ_w>. Acesso em: 29 nov. 2023.
4. Aho-Corasick Algorithm. Disponível em: <https://www.youtube.com/watch?v=alG4qAJdd4o>. Acesso em: 29 nov. 2023.
5. ÁRVORES TRIES Disciplina Estrutura de Dados - ppt carregar. Disponível em: <https://slideplayer.com.br/slide/2502254/>. Acesso em: 29 nov. 2023.

Obrigado!!!