Melhores momentos

AULA 21

Resumo

função	consumo de tempo	observação
bubble	$O(n^2)$	todos os casos
insercao	$O(n^2)$ $O(n)$	pior caso melhor caso
insercaoBinaria	$O(n^2)$ $O(n \lg n)$	pior caso melhor caso
selecao	$O(n^2)$	todos os casos
mergeSort	$O(n \lg n)$	todos os casos
quickSort	$O(n^2)$ $O(n \lg n)$	pior caso melhor caso
heapSort	$O(n \lg n)$	todos os casos

Divisão e conquista

Algoritmos por divisão-e-conquista têm três passos em cada nível da recursão:

Dividir: o problema é dividido em subproblemas de tamanho menor;

Conquistar: os subproblemas são resolvidos recursivamente e subproblemas "pequenos" são resolvidos diretamente;

Combinar: as soluções dos subproblemas são combinadas para obter uma solução do problema original.

Exemplo: ordenação por intercalação (mergeSort).

AULA 22

Busca de palavras (string matching)

PF 13

http://www.ime.usp.br/~pf/algoritmos/aulas/strma.html

Busca de palavras em um texto

Dizemos que um vetor $p[1..m]$ **ocorre em** um vetor $t[1..n]$ se

$$p[1..m] = t[s+1..s+m]$$

para algum s em $[0..n-m]$.

Exemplo:

	1	2	3	4	5	6	7	8	9	10
t	x	c	b	a	b	b	c	b	a	x

	1	2	3	4
p	b	c	b	a

$p[1..4]$ ocorre em $t[1..10]$ com deslocamento 5.

Busca de palavras em um texto

Problema: Dados $p[1\,..\,m]$ e $t[1\,..\,n]$, encontrar o número de ocorrências de p em t.

Exemplo: Para $n = 10$, $m = 4$, e

	1	2	3	4	5	6	7	8	9	10
t	b	b	a	b	a	b	a	c	b	a

	1	2	3	4
p	b	a	b	a

p ocorre 2 vezes em t.

Algoritmo trivial

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													

Algoritmo trivial

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												

Algoritmo trivial

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											

Algoritmo trivial

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										

Algoritmo trivial

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								
7							a	b	a	b	b	a	b	a	b	b	a							
8								a	b	a	b	b	a	b	a	b	b	a						
9									a	b	a	b	b	a	b	a	b	b	a					
10										a	b	a	b	b	a	b	a	b	b	a				
11											a	b	a	b	b	a	b	a	b	b	a			
12												a	b	a	b	b	a	b	a	b	b	a		
13													a	b	a	b	b	a	b	a	b	b	a	

Algoritmo trivial

Devolve o número de ocorrências de p em t.

```
int trivial (unsigned char p[], int m,
             unsigned char t[], int n) {
    int r, k, ocorrs = 0;
1   for (k = 1; k <= n-m+1; k++) {
2       r = 0;
3       while (r < m && p[1+r] == t[k+r])
4           r += 1;
5       if (r == m) ocorrs += 1;
    }
6   return ocorrs;
 }
```

Algoritmo trivial

Relação invariante: no início da linha 3 vale que

(i0) $p[1\,..\,1{+}r{-}1] = t[k\,..\,k{+}r{-}1]$

Consumo de tempo

Consumo de tempo da função `trivial`, versão direita para a esquerda.

linha	todas as execuções da linha
1	$= n - m + 2$
2	$= n - m + 1$
3	$\leq (n - m + 1)(m + 1)$
4	$\leq (n - m + 1)m$
5	$= n - m + 1$
6	$= 1$
total	$< 3\,(n - m + 2) + 2\,(n - m + 1)(m + 1)$ $= O((n - m + 1)m)$

Pior caso

p = a a a a a a a a a a a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  t
1   a  a  a  a  a  a  a  a  a  a  a
2      a  a  a  a  a  a  a  a  a  a  a
3         a  a  a  a  a  a  a  a  a  a  a
4            a  a  a  a  a  a  a  a  a  a  a
5               a  a  a  a  a  a  a  a  a  a  a
6                  a  a  a  a  a  a  a  a  a  a  a
7                     a  a  a  a  a  a  a  a  a  a  a
8                        a  a  a  a  a  a  a  a  a  a  a
9                           a  a  a  a  a  a  a  a  a  a  a
10                             a  a  a  a  a  a  a  a  a  a  a
11                                a  a  a  a  a  a  a  a  a  a  a
12                                   a  a  a  a  a  a  a  a  a  a  a
13                                      a  a  a  a  a  a  a  a  a  a  a
```

Melhor caso

p = b a a a a a a a a a a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  a  t
1   b  a  a  a  a  a  a  a  a  a  a
2      b  a  a  a  a  a  a  a  a  a  a
3         b  a  a  a  a  a  a  a  a  a  a
4            b  a  a  a  a  a  a  a  a  a  a
5               b  a  a  a  a  a  a  a  a  a  a
6                  b  a  a  a  a  a  a  a  a  a  a
7                     b  a  a  a  a  a  a  a  a  a  a
8                        b  a  a  a  a  a  a  a  a  a  a
9                           b  a  a  a  a  a  a  a  a  a  a
10                             b  a  a  a  a  a  a  a  a  a  a
11                                b  a  a  a  a  a  a  a  a  a  a
12                                   b  a  a  a  a  a  a  a  a  a  a
13                                      b  a  a  a  a  a  a  a  a  a  a
```

Conclusões

O consumo de tempo da função `trivial` no pior caso é $O((n - m + 1)m)$.

O consumo de tempo da função `trivial` no melhor caso é $O(n - m + 1)$.

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a mn.

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2		a	b	a	b	b	a	b	a	b	b	a												
3			a	b	a	b	b	a	b	a	b	b	a											
4				a	b	a	b	b	a	b	a	b	b	a										
5					a	b	a	b	b	a	b	a	b	b	a									
6						a	b	a	b	b	a	b	a	b	b	a								

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

```
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
     a b a a b a b a b b  a  b  a  b  a  b  b  a  b  a  b  b  a   t
  1  a b a b b a b a b b  a
  2    a b a b b a b a b  b  a
  3      a b a b b a b a  b  b  a
  4        a b a b b a b  a  b  b  a
  5          a b a b b a  b  a  b  b  a
  6            a b a b b  a  b  a  b  b  a
  7              a b a b  b  a  b  a  b  b  a
```

Algoritmo trivial: direita para esquerda

p = a b a b b a b a b b a

```
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
     a b a a b a b a b b  a  b  a  b  a  b  b  a  b  a  b  b  a   t
  1  a b a b b a b a b b  a
  2    a b a b b a b a b  b  a
  3      a b a b b a b a  b  b  a
  4        a b a b b a b  a  b  b  a
  5          a b a b b a  b  a  b  b  a
  6            a b a b b  a  b  a  b  b  a
  7              a b a b  b  a  b  a  b  b  a
  8                a b a  b  b  a  b  a  b  b  a
  9                  a b  a  b  b  a  b  a  b  b  a
 10                    a  b  a  b  b  a  b  a  b  b  a
 11                       a  b  a  b  b  a  b  a  b  b  a
 12                          a  b  a  b  b  a  b  a  b  b  a
 13                             a  b  a  b  b  a  b  a  b  b  a
```

Algoritmo trivial: direita para esquerda

Devolve o número de ocorrências de p em t.

```
int trivial (unsigned char p[], int m,
             unsigned char t[], int n) {
   int r, k, ocorrs = 0;
1  for (k = m; k <= n; k++) {
2     r = 0;
3     while (r < m && p[m-r] == t[k-r])
4        r += 1;
5     if (r == m) ocorrs += 1;
   }
6  return ocorrs;
}
```

Algoritmo trivial: direita para esquerda

Relação invariante: no início da linha 3 vale que

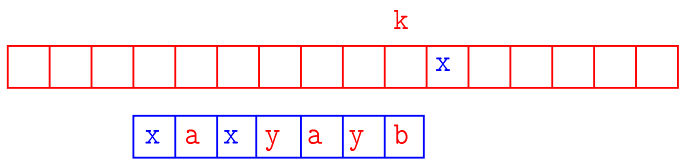
(i0) $p[m-r+1\,..\,m] = t[k-r+1\,..\,k]$

Algoritmo trivial: direita para esquerda

```
int trivial (unsigned char p[], int m,
             unsigned char t[], int n) {
    int r, k, ocorrs;
3   ocorrs = 0; k = m;
4   while (k <= n) {
5      r = 0;
6      while (r < m && p[m-r] == t[k-r])
7          r += 1;
8      if (r == m) ocorrs += 1;
9      k += 1;
    }
11  return ocorrs;
}
```

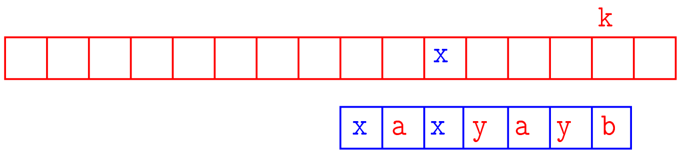
Primeiro algoritmo de Boyer-Moore

O primeiro algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Primeiro algoritmo de Boyer-Moore

O primeiro algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.



Boyer-Moore

```
p = andando
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
```

Boyer-Moore

```
p = andando
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2         andando
```

Boyer-Moore

```
p = andando
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2         andando
3            andando
```

Boyer-Moore

```
p = andando
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2         andando
3            andando
4                    andando
```

Boyer-Moore

```
p = andando
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2         andando
3            andando
4                    andando
5                     andando
```

Boyer-Moore

p = a n d a n d o

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
    a  s     a  n  d  o  r  i  n  h  a  s     a  n  d  a  m     a  n  d  a  n  d  o     a  l  t  o  t
1   a  n  d  a  n  d  o
2                        a  n  d  a  n  d  o
3                                 a  n  d  a  n  d  o
4                                                            a  n  d  a  n  d  o
5                                                               a  n  d  a  n  d  o
6                                                                                       a  n  d  a  ...
```

Boyer-Moore

p = a b a b b a b a b b a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1   a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1   a  b  a  b  b  a  b  a  b  b  a
2         a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1   a  b  a  b  b  a  b  a  b  b  a
2         a  b  a  b  b  a  b  a  b  b  a
3               a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1   a  b  a  b  b  a  b  a  b  b  a
2         a  b  a  b  b  a  b  a  b  b  a
3               a  b  a  b  b  a  b  a  b  b  a
4                     a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
    1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
    a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1   a  b  a  b  b  a  b  a  b  b  a
2         a  b  a  b  b  a  b  a  b  b  a
3               a  b  a  b  b  a  b  a  b  b  a
4                     a  b  a  b  b  a  b  a  b  b  a
5                        a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1  a  b  a  b  b  a  b  a  b  b  a
2        a  b  a  b  b  a  b  a  b  b  a
3              a  b  a  b  b  a  b  a  b  b  a
4                    a  b  a  b  b  a  b  a  b  b  a
5                       a  b  a  b  b  a  b  a  b  b  a
6                             a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1  a  b  a  b  b  a  b  a  b  b  a
2        a  b  a  b  b  a  b  a  b  b  a
3              a  b  a  b  b  a  b  a  b  b  a
4                    a  b  a  b  b  a  b  a  b  b  a
5                       a  b  a  b  b  a  b  a  b  b  a
6                             a  b  a  b  b  a  b  a  b  b  a
7                                   a  b  a  b  b  a  b  a  b  b  a
```

Boyer-Moore

p = a b a b b a b a b b a

```
   1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   a  b  a  a  b  a  b  a  b  b  a  b  a  b  a  b  b  a  b  a  b  b  a  t
1  a  b  a  b  b  a  b  a  b  b  a
2        a  b  a  b  b  a  b  a  b  b  a
3              a  b  a  b  b  a  b  a  b  b  a
4                    a  b  a  b  b  a  b  a  b  b  a
5                       a  b  a  b  b  a  b  a  b  b  a
6                             a  b  a  b  b  a  b  a  b  b  a
7                                   a  b  a  b  b  a  b  a  b  b  a
8                                      a  b  a  b  b  a  b  a  b  b  a
```

Primeiro algoritmo de Boyer-Moore

Ideia ("*bad-character heuristic*"): calcular um deslocamento de modo que t[k+1] fique emparelhado com a última ocorrência do caractere t[k+1] em p.

Suponha que o conjunto a que pertencem todos os elementos de p e de t é conhecido de antemão. Este conjunto é o **alfabeto** do problema.

Suponha que o alfabeto é o conjunto de todos os 256 caracteres.

Primeiro algoritmo de Boyer-Moore

Para implementar essa ideia fazemos um pré-processamento de p, determinando para cada símbolo x do alfabeto a posição de sua última ocorrência em p.

	1	2	3	4	5	6	7
p	a	n	d	a	n	d	o

	0	...	'a'	'b'	'c'	'd'	...	...	'n'	'o'	'p'	...	255
ult	0	...	4	0	0	6	...	...	5	7	0	...	...

Primeiro algoritmo de Boyer-Moore

Recebe vetores p[1 . . m] e t[1 . . n] de caracteres, com $m \geq 1$ e $n \geq 0$, e devolve o número de ocorrências de p em t.

```
int BoyerMoore (unsigned char p[], int m,
                unsigned char t[], int n) {
   int ult[256];
   int i, r, k, ocorrs;

   /* pre-processamento da palavra p */
   for (i=0; i < 256; i++) ult[i] = 0;
   for (i=1; i <= m; i++) ult[p[i]] = i;
```

Primeiro algoritmo de Boyer-Moore

```
/* busca da palavra p no texto t */
ocorrs = 0; k = m;
while (k <= n) {
    r = 0;
    while (r < m && p[m-r] == t[k-r])
        r += 1;
    if (r == m) ocorrs += 1;
    if (k == n) k += 1;
    else k += m - ult[t[k+1]] + 1;
}
return ocorrs;
}
```

Pior caso

p = a a a a a a a a a a a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	t
1	a	a	a	a	a	a	a	a	a	a	a												
2		a	a	a	a	a	a	a	a	a	a	a											
3			a	a	a	a	a	a	a	a	a	a	a										
4				a	a	a	a	a	a	a	a	a	a	a									
5					a	a	a	a	a	a	a	a	a	a	a								
6						a	a	a	a	a	a	a	a	a	a	a							
7							a	a	a	a	a	a	a	a	a	a	a						
8								a	a	a	a	a	a	a	a	a	a	a					
9									a	a	a	a	a	a	a	a	a	a	a				
10										a	a	a	a	a	a	a	a	a	a	a			
11											a	a	a	a	a	a	a	a	a	a	a		
12												a	a	a	a	a	a	a	a	a	a	a	
13													a	a	a	a	a	a	a	a	a	a	a

Melhor caso

p = a a a a b

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	t
1	a	a	a	a	b																		

Melhor caso

p = a a a a b

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	t
1	a	a	a	a	b																		
2							a	a	a	a	b												

Melhor caso

p = a a a a b

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	t
1	a	a	a	a	b																		
2							a	a	a	a	b												
3													a	a	a	a	b						

Melhor caso

p = a a a a b

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	c	?	?	?	?	a	t
1	a	a	a	a	b																		
2							a	a	a	a	b												
3													a	a	a	a	b						
4																			a	a	a	a	b

Conclusões

O consumo de tempo da função BoyerMoore no pior caso é $\mathrm{O}((n-m+1)m)$.

O consumo de tempo da função BoyerMoore no melhor caso é $\mathrm{O}(n/m)$.

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a mn e no melhor caso o algoritmo é **sublinear**.

Segundo algoritmo de Boyer-Moore

O segundo algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.

k

						x	a	a	b						

?	a	a	b	y	a	a	b

Segundo algoritmo de Boyer-Moore

O segundo algoritmo de R.S. Boyer e J.S. Moore (1977) é baseado na seguinte heurística.

k

						x	a	a	b						

?	a	a	b	y	a	a	b

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   a s   a n d o r i n  h  a  s     a  n  d  a  m     a  n  d  a  n  d  o     a  l  t  o  t
1  a n d a n d o
```

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   a s   a n d o r i n  h  a  s     a  n  d  a  m     a  n  d  a  n  d  o     a  l  t  o  t
1  a n d a n d o
2                a n d  a  n  d  o
```

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   a s   a n d o r i n  h  a  s     a  n  d  a  m     a  n  d  a  n  d  o     a  l  t  o  t
1  a n d a n d o
2                a n d  a  n  d  o
3                  a n  d  a  n  d  o
```

Boyer-Moore 2

p = a n d a n d o

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2        andando
3         andando
4          andando
```

Boyer-Moore 2

p = a n d a n d o

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2        andando
3         andando
4          andando
5           andando
```

Boyer-Moore 2

p = a n d a n d o

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2        andando
3         andando
4          andando
5           andando
6            andando
```

Boyer-Moore 2

p = a n d a n d o

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2        andando
3         andando
4          andando
5           andando
6            andando
7             andando
```

Boyer-Moore 2

p = a n d a n d o

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2        andando
3         andando
4          andando
5           andando
6            andando
7             andando
8              andando
```

Boyer-Moore 2

p = a n d a n d o

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
  as andorinhas andam andando alto t
1 andando
2        andando
3         andando
4          andando
5           andando
6            andando
7             andando
8              andando
9               andando
```

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   as andorinhas andam andando alto t
1  andando
2         andando
3          andando
4           andando
5            andando
6             andando
7              andando
8               andando
9                andando
10                andando
```

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   as andorinhas andam andando alto t
1  andando
2         andando
3          andando
4           andando
5            andando
6             andando
7              andando
8               andando
9                andando
10                andando
11                 andando
```

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   as andorinhas andam andando alto t
1  andando
2         andando
3          andando
4           andando
5            andando
6             andando
7              andando
8               andando
9                andando
10                andando
11                 andando
15                     andando
```

Boyer-Moore 2

p = a n d a n d o

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
   as andorinhas andam andando alto t
1  andando
2         andando
3          andando
4           andando
5            andando
6             andando
7              andando
8               andando
9                andando
10                andando
11                 andando
15                     andando
16                             anda...
```

Boyer-Moore 2

p = a b a b b a b a b b a

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   abaababababbababababbababba t
1  ababbababba
```

Boyer-Moore 2

p = a b a b b a b a b b a

```
   1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
   abaababababbababababbababba t
1  ababbababba
2       ababbababba
```

Boyer-Moore 2

$p = $ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2						a	b	a	b	b	a	b	a	b	b	a								
3							a	b	a	b	b	a	b	a	b	b	a							

Boyer-Moore 2

$p = $ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2						a	b	a	b	b	a	b	a	b	b	a								
3							a	b	a	b	b	a	b	a	b	b	a							
4								a	b	a	b	b	a	b	a	b	b	a						

Boyer-Moore 2

$p = $ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2						a	b	a	b	b	a	b	a	b	b	a								
3							a	b	a	b	b	a	b	a	b	b	a							
4								a	b	a	b	b	a	b	a	b	b	a						
5													a	b	a	b	b	a	b	a	b	b	a	

Boyer-Moore 2

$p = $ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2						a	b	a	b	b	a	b	a	b	b	a								
3							a	b	a	b	b	a	b	a	b	b	a							
4								a	b	a	b	b	a	b	a	b	b	a						
5													a	b	a	b	b	a	b	a	b	b	a	
6																		a	b	a	b	b	a	...

Boyer-Moore 2

$p = $ a b a b b a b a b b a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	t
1	a	b	a	b	b	a	b	a	b	b	a													
2						a	b	a	b	b	a	b	a	b	b	a								
3							a	b	a	b	b	a	b	a	b	b	a							
4								a	b	a	b	b	a	b	a	b	b	a						
5													a	b	a	b	b	a	b	a	b	b	a	
6																		a	b	a	b	b	a	...

Segundo algoritmo de Boyer-Moore

Ideia (*good-suffix heuristic*): para cada índice h calcular o maior i em $1\,..\,m$-1 tal que

- $p[1\,..\,j]$ é um sufixo de $p[i\,..\,m]$ ou
- $p[i\,..\,m]$ é um sufixo de $p[1\,..\,j]$.

Para implementar essa ideia basta um fazermos um pré-processamento que **só depende** de p.

Segundo algoritmo de Boyer-Moore

Para implementar essa ideia fazemos um pré-processamento de p, determinando para cada símbolo índice i o índice alcance[i] de um "sufixo bom".

	1	2	3	4	5	6	7	8	9	10	11
p	a	b	a	b	b	a	b	a	b	b	a

	1	2	3	4	5	6	7	8	9	10	11
alcance	6	6	6	6	6	6	6	6	6	8	8

Segundo algoritmo de Boyer-Moore

Para implementar essa ideia fazemos um pré-processamento de p, determinando para cada símbolo índice h o índice alcance[h] de um "sufixo bom".

	1	2	3	4	5	6
p	c	a	a	b	a	a

	1	2	3	4	5	6
alcance	0	0	0	0	3	5

Segundo algoritmo de Boyer-Moore

Recebe vetores p[1 . . m] e t[1 . . n] de caracteres, com $m \geq 1$ e $n \geq 0$, e devolve o número de ocorrências de p em t.

```
int BoyerMoore2 (unsigned char p[], int m,
                 unsigned char t[], int n) {
  int *alcance;
  int i, r, k, ocorrs;
  /* pre-processamento da palavra p */
  alcance = preProcessamento(p, m);
  /* em branco */
```

Segundo algoritmo de Boyer-Moore

```
  /* busca da palavra p no texto t */
  ocorrs = 0; k = m;
  while (k <= n) {
      r = 0;
      while (r < m && p[m-r] == t[k-r])
          r += 1;
      if (r == m) ocorrs += 1;
      if (r == 0) k += 1;
      else k += m - alcance[m-r+1];
  }
  free(alcance);
  return ocorrs;
}
```

Pré-processamento

```
int *
preProcessamento(unsigned char p[], int m)
{
   int i, r, j, *alcance;
   alcance = malloc((m+1)*sizeof(int));
   for (i = m; i >= 1; i--) {
       j = m-1; r = 0
       while (m-r >= i && j-r >= 1)
           if (p[m-r] == p[j-r]) r += 1;
           else j -= 1, r = 0;
       alcance[i] = j;
   }
   return alcance;
}
```

Pior caso

p = a a a a a a a a a a a

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
t	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
1	a	a	a	a	a	a	a	a	a	a	a												
2		a	a	a	a	a	a	a	a	a	a	a											
3			a	a	a	a	a	a	a	a	a	a	a										
4				a	a	a	a	a	a	a	a	a	a	a									
5					a	a	a	a	a	a	a	a	a	a	a								
6						a	a	a	a	a	a	a	a	a	a	a							
7							a	a	a	a	a	a	a	a	a	a	a						
8								a	a	a	a	a	a	a	a	a	a	a					
9									a	a	a	a	a	a	a	a	a	a	a				
10										a	a	a	a	a	a	a	a	a	a	a			
11											a	a	a	a	a	a	a	a	a	a	a		
12												a	a	a	a	a	a	a	a	a	a	a	
13													a	a	a	a	a	a	a	a	a	a	a

Melhor caso

$p = \mathtt{a\ a\ a\ a\ b}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	t
1	a	a	a	a	b																			

Melhor caso

$p = \mathtt{a\ a\ a\ a\ b}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	t
1	a	a	a	a	b																			
2						a	a	a	a	b														

Melhor caso

$p = \mathtt{a\ a\ a\ a\ b}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	t
1	a	a	a	a	b																			
2						a	a	a	a	b														
3											a	a	a	a	b									

Melhor caso

$p = \mathtt{a\ a\ a\ a\ b}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	t
1	a	a	a	a	b																			
2						a	a	a	a	b														
3											a	a	a	a	b									
4																a	a	a	a	b				

Melhor caso

$p = \mathtt{a\ a\ a\ a\ b}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	c	b	?	?	?	t
1	a	a	a	a	b																			
2						a	a	a	a	b														
3											a	a	a	a	b									
4																a	a	a	a	b				
5																					a	a	a	...

Conclusões

O consumo de tempo da função `BoyerMoore2` no pior caso é $O((n - m + 1)m)$.

O consumo de tempo da função `BoyerMoore2` no melhor caso é $O(n/m)$.

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a mn e no melhor caso o algoritmo é **sublinear**.

Terceiro algoritmo de Boyer-Moore

O algoritmo de Boyer-Moore propriamente dito é uma fusão dos dois anteriores:

> *a cada passo, o algoritmo escolhe o maior dos deslocamentos ditados pelas tabelas* ult *e* $alcance$.