Complementando a aula passada

AULA 24

Tabelas de dispersão (*hash tables*)

Uma **tabela de dispensão** (= *hash table*) é uma maneira de organizar uma tabela de símbolos.

Inventadas para funcionar bem em média.

universo de chaves = conjunto de **todas** as possíveis chaves

chaves *realmente usadas* são, em geral, uma *parte pequena* do universo.

A tabela terá a forma `tab`[0 . . `M`-1], onde `M` é o tamanho da tabela.

Funções de dispersão

Uma **função de dispersão** (= *hash function*) é uma maneira de mapear o universo de chaves no conjunto de índices da tabela.

A função de dispersão recebe uma chave e retorna um número inteiro `h` no intervalo 0 . . `M`-1.

O número `h` é o **código de dispensão** (= *hash code*) da chave.

Exemplo de uma função de dispersão modular:

```
int hash(Chave chave, int M) {
   return chave % M;
}
```

Função de dispersão básica

Para evitar *overflow* usamos o método de Horne e tomamos o resto da divisão após cada multiplicação:

```
int hash(Chave chave, int M)
{
   int i, h = 0;
   int primo = 127;
   for (i = 0; chave[i] != '\0'; i++)
       h = (h * primo + chave[i]) % M;
   return h;
}
```

Colisões

Como o número de chaves é em geral maior que `M`, é inevitável que a função de dispersão leve várias chaves diferentes no mesmo índice.

Dizemos que há uma **colisão** quando duas chaves diferentes são levadas no mesmo índice.

Algumas maneiras de tratar colisões:

- lista encadeadas (=*separating chaining*);
- sondagem linear (=*linear probing (open addressing)*);
- *double hashing (open addressing)*;

Colisões por sondagem linear

Um outro método de resolução de colisões é conhecido como sondagem linear (=*linear probing*).

Todos os objetos são armazenados em um vetor `tab`[0 . . `M`-1].

Quando ocorre uma colisão, procuramos a próxima posição vaga do vetor.

Digamos que a tabela tem `M` posições e
contém `N` chaves num dado instante.
O fator de carga $\alpha = \texttt{N}/\texttt{M}$ da tabela é menor do que 1.
Quanto maior o fator de carga, mais tempo
as funções de busca e inserção vão consumir.

Interface

```
typedef char *String;
typedef String Chave;
typedef int Valor;

void stInit(int);
void stInsert(Chave, Valor);
Valor stSearch(Chave);
void stFree();
void stDelete(Chave);
```

Implementação

```
#define LIVRE(h) (tab[h].chave == NULL)
#define INCR(h) (h = h == M-1?  0:  h+1)

static int hash(Chave chave, int M);

typedef struct celTS CelTS;
struct celTS {
   Chave chave;
   Valor valor;
};

static CelTS *tab = NULL;
static int nChaves = 0;
static int M; /* tam. da tabela */
```

stInit

```
void stInit(int max)
{
   int h;
   M = max;
   nChaves = 0;
   tab = mallocSafe(M * sizeof(CelTS));
   for (h = 0; h < M; h++)
       tab[h].chave = NULL;
}
```

stInsert

```
void stInsert(Chave chave, Valor valor) {
   CelTS *p;    int h = hash(chave,M);
   while(!LIVRE(h)&&strcmp(tab[h].chave,chave))
       INCR(h);
   if (LIVRE(h)) {
       if (nChaves == M-1) {
           printf("Tabela cheia\"); return;
       }
       tab[h].chave = copyString(chave);
       nChaves += 1;
   }
   tab[h].valor = valor;
}
```

stSearch

```
Valor stSearch(Chave chave) {
   CelTS *p = NULL;
   int h;

   /* procure chave na tabela */
   h = hash(chave,M); /* codigo de hash */
   while(!LIVRE(h)&&strcmp(tab[h].chave,chave))
       INCR(h);

   if (LIVRE(h)) return 0;
   return tab[h].valor;
}
```

stDelete

```
void stDelete(Chave chave) {
   int h;

   /* procure chave na tabela */
   h = hash(chave,M); /*codigo de hash*/
   while(!LIVRE(h)&&strcmp(tab[h].chave,chave))
       INCR(h);
   if (LIVRE(h)) return;

   /* remova chave da tabela */
   nChaves -= 1;
   free(tab[h].chave);
   tab[h].chave = NULL;
```

stDelete

```
    /* faca rehash das chaves seguintes */
    for (INCR(h); !LIVRE(h); INCR(h)) {
        Chave chave = tab[h].chave;
        Valor valor = tab[h].valor;
        tab[h].chave = NULL;
        stInsert(chave,valor); /* rehash */
        free(chave);
    }
}
```

stFree

```
void stFree() {
   int h;
   for (h = 0; h < M; h++)
       if (!LIVRE(h))
           free(tab[h].chave);
   free(tab);
   tab = NULL
   nChaves = 0;
   M = 0;
}
```

AULA 25

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 $\mathtt{p}[1\,..\,\mathtt{m}]$ e $\mathtt{t}[1\,..\,\mathtt{n}]$, encontrar o número de ocorrências de $\mathtt{p}$ em $\mathtt{t}$.

Exemplo: Para $\mathtt{n} = 10$, $\mathtt{m} = 4$, 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) $\mathtt{p}[1\,..\,1{+}\mathtt{r}{-}1] = \mathtt{t}[\mathtt{k}\,..\,\mathtt{k}{+}\mathtt{r}{-}1]$

Consumo de tempo

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

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

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 $\mathrm{O}((\mathtt{n}-\mathtt{m}+1)\mathtt{m})$.

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

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a $\mathtt{mn}$.

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

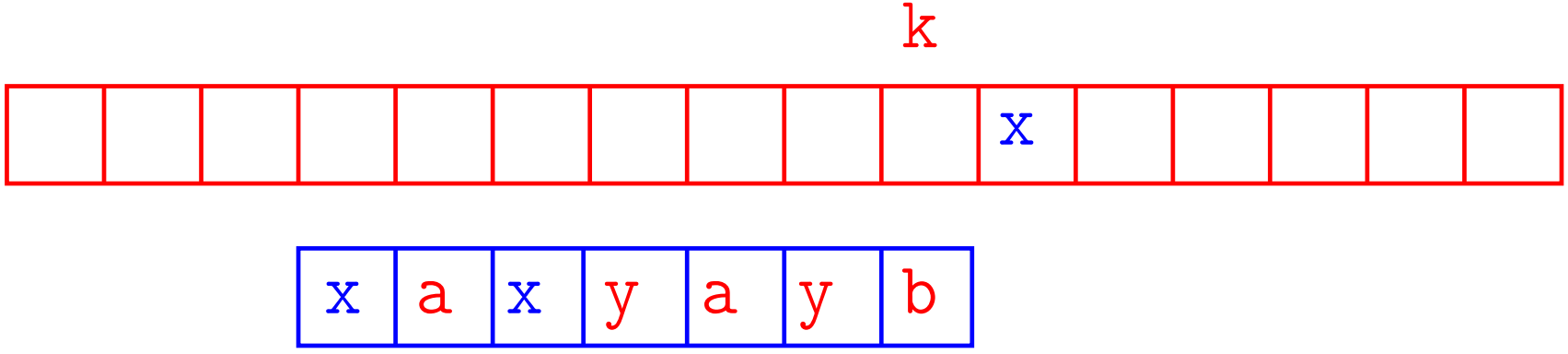
(i0) $\mathtt{p}[\mathtt{m}-\mathtt{r}+1\,..\,\mathtt{m}] = \mathtt{t}[\mathtt{k}-\mathtt{r}+1\,..\,\mathtt{k}]$

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;
  }
```

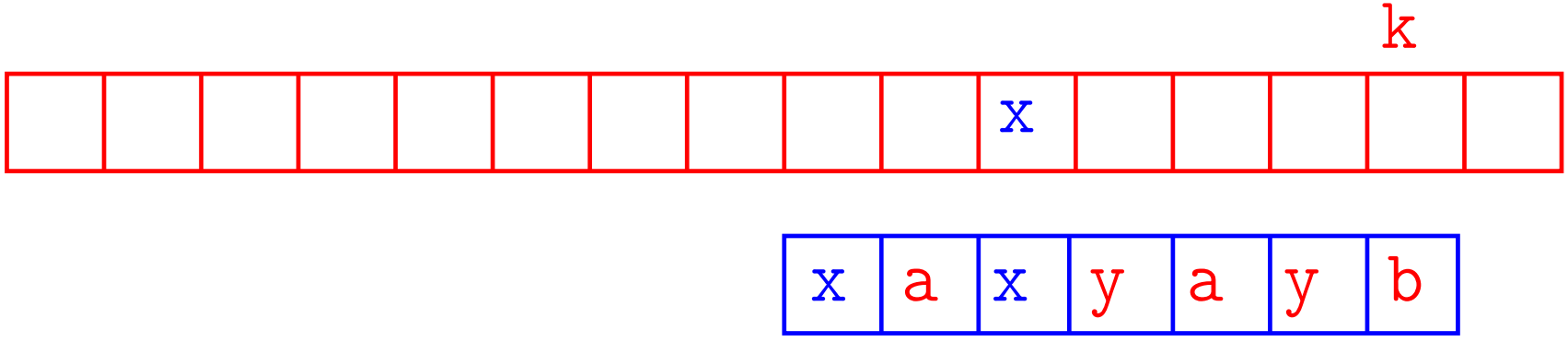
Primeiro algoritmo de Boyer-Moore

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 = 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

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

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

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
```

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
```

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 */
1   for (i=0; i < 256; i++) ult[i] = 0;
2   for (i=1; i <= m; i++) ult[p[i]] = i;
```

Primeiro algoritmo de Boyer-Moore

```
   /* busca da palavra p no texto t */
 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       if (k == n) k += 1;
10       else k += m - ult[t[k+1]] + 1;
     }
11   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	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

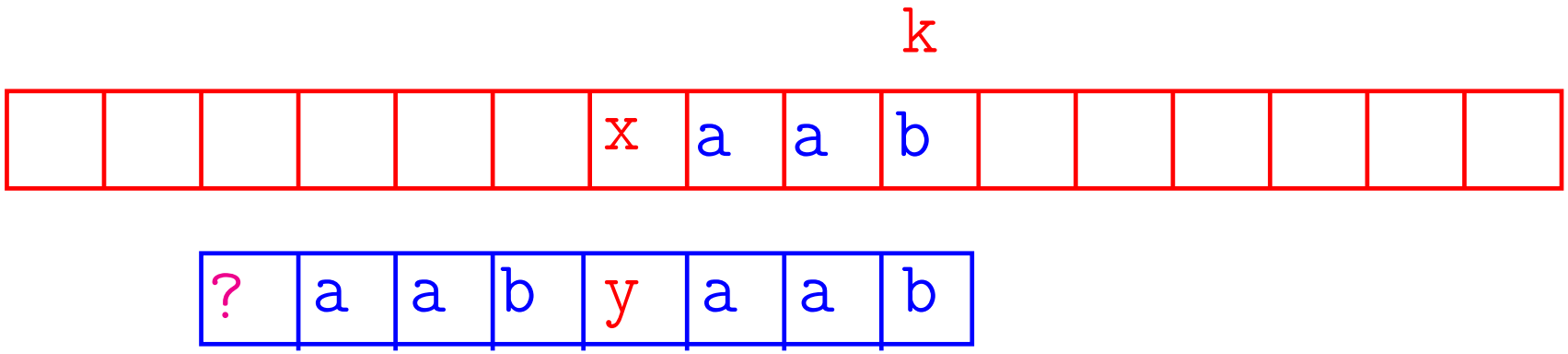
O consumo de tempo da função `BoyerMoore` no pior caso é $O((n - m + 1)m)$.

O consumo de tempo da função `BoyerMoore` 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**.

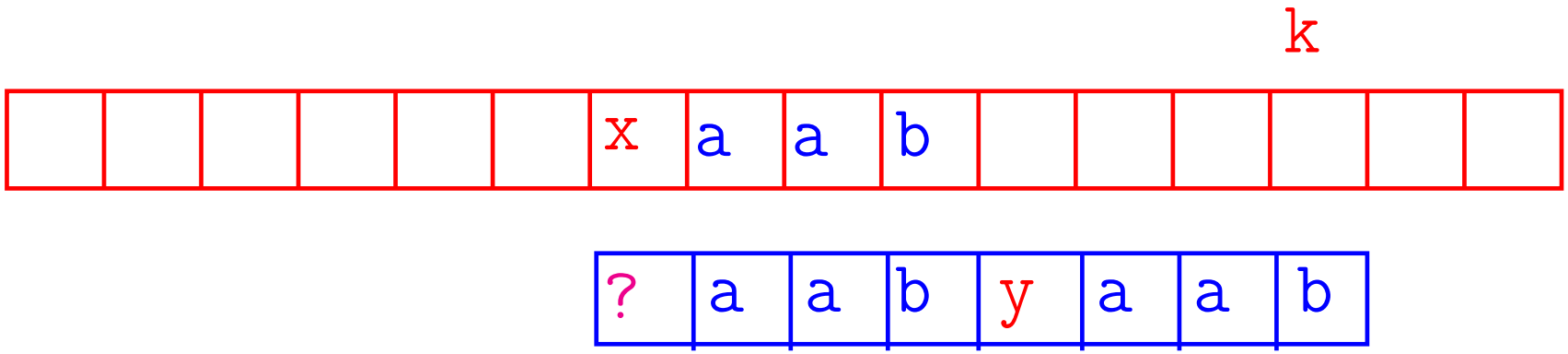
Segundo algoritmo de Boyer-Moore

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



Segundo algoritmo de Boyer-Moore

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



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
   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
```

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
4               a n d a n d o
5                 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
4               a n d a n d o
5                 a n d a n d o
6                   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
4               a n d a n d o
5                 a n d a n d o
6                   a n d a n d o
7                     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
4               a n d a n d o
5                 a n d a n d o
6                   a n d a n d o
7                     a n d a n d o
8                       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
4               a n d a n d o
5                 a n d a n d o
6                   a n d a n d o
7                     a n d a n d o
8                       a n d a n d o
9                         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
4                a n d a n d o
5                  a n d a n d o
6                    a n d a n d o
7                      a n d a n d o
8                        a n d a n d o
9                          a n d a n d o
10                           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
4               a n d a n d o
5                 a n d a n d o
6                   a n d a n d o
7                     a n d a n d o
8                       a n d a n d o
9                         a n d a n d o
10                          a n d a n d o
11                            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
4               a n d a n d o
5                 a n d a n d o
6                   a n d a n d o
7                     a n d a n d o
8                       a n d a n d o
9                         a n d a n d o
10                          a n d a n d o
11                            a n d a n d o
15                                  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
4               a n d a n d o
5                 a n d a n d o
6                   a n d a n d o
7                     a n d a n d o
8                       a n d a n d o
9                         a n d a n d o
10                          a n d a n d o
11                            a n d a n d o
15                                  a n d a n d o
16                                        a n d 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													

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								

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 `i` calcular o maior `j` 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 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 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 `i` o índice `alcance[i]` 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 */
1   alcance = preProcessamento(p, m);
2   /* em branco */
```

Segundo algoritmo de Boyer-Moore

```
   /* busca da palavra p no texto t */
 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       if (r == 0) k += 1;
10       else k += m - alcance[m-r+1];
     }
11   free(alcance);
12   return ocorrs;
   }
```

Pré-processamento

```
int *
preProcessamento(unsigned char p[], int m) {
    int i, r, j, *alcance;
    alcance = malloc((m+1)*sizeof(int));
1   for (i = m; i >= 1; i--) {
2       j = m-1; r = 0
3       while (m-r >= i && j-r >= 1)
4           if (p[m-r] == p[j-r]) r += 1;
5           else j -= 1, r = 0;
6       alcance[i] = j;
    }
7   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	
	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 = 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 = 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 = 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 = 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 = 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 $\mathrm{O}((\mathtt{n}-\mathtt{m}+1)\mathtt{m})$.

O consumo de tempo da função `BoyerMoore2` no melhor caso é $\mathrm{O}(\mathtt{n}/\mathtt{m})$.

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a $\mathtt{mn}$ 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`*.*