MAC0323 Algoritmos e Estruturas de Dados II

Edição 2020 – 2

AULA 29

Mais busca de palavras

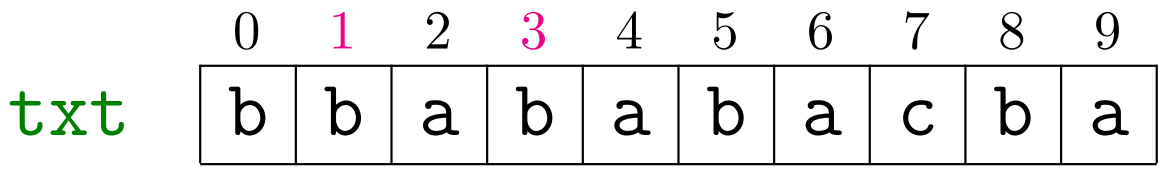
PF 13

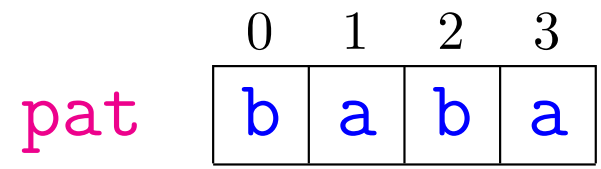
http://www.ime.usp.br/~pf/algoritmos/aulas/strma.html

Busca de palavras em um texto

Problema: Dados `pat`$[0\,..\,\mathtt{m}-1]$ e `txt`$[0\,..\,\mathtt{n}-1]$, encontrar o número de ocorrências de `pat` em `txt`.

Exemplo: Para $\mathtt{n} = 10$, $\mathtt{m} = 4$, e

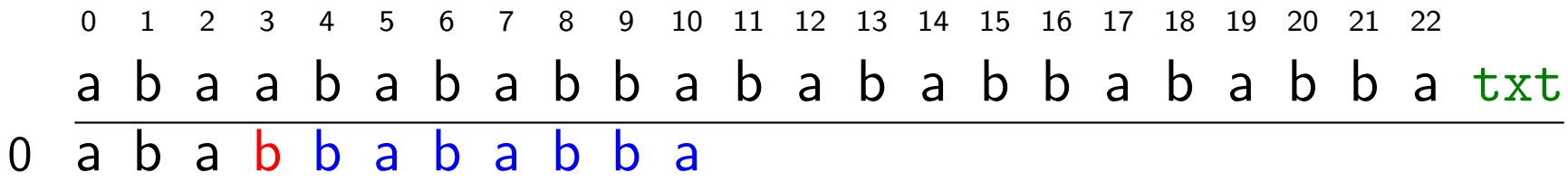




`pat` ocorre 2 vezes em `txt`.

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a



Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
1		a	b	a	b	b	a	b	a	b	b	a												

Algoritmo força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
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 força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
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 força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
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 força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
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 força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
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 força bruta: direita para esquerda

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
0	a	b	a	b	b	a	b	a	b	b	a													
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	

Força bruta: direita para esquerda

Devolve a primeira de ocorrências de `pat` em `txt`.

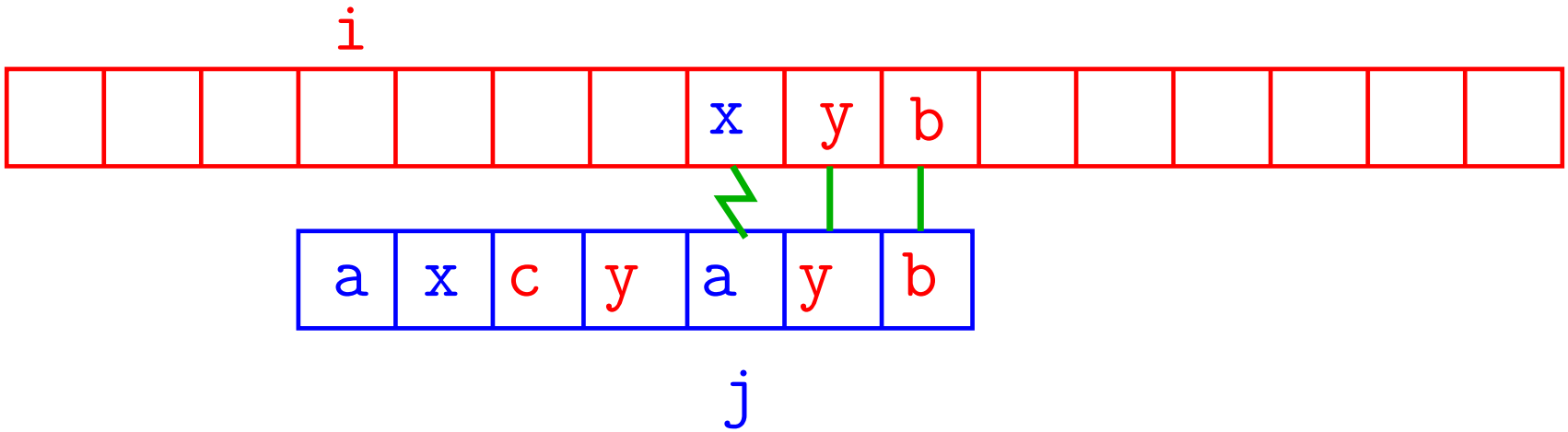
```
int search(char *pat, char *txt) {
   int i, n = strlen(txt);
   int j, m = strlen(pat);
   for (i = 0; i <= n-m; i+=1 /* skip */) {
      for (j = m-1; j >= 0; j--)
         if(txt[i+j] != pat[j])
            break;
      if (j == -1) return i;
   }
   return n;
}
```

Boyer-Moore

Fonte: ADS: Boyer Moore String Search

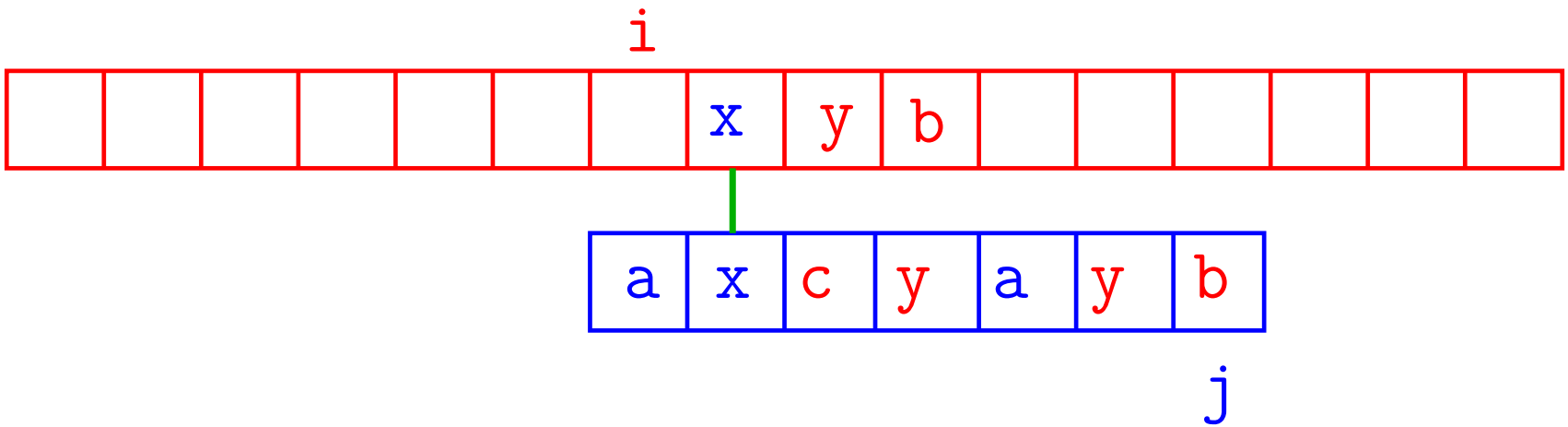
Primeiro algoritmo de Boyer-Moore

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

pat = a n d a n d o

```
  0 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
  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 txt
  ---------------------------------------------------------------
1 a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 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
  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 txt
1 a n d a n d o
2       a n d a n d o
```

Boyer-Moore

pat = a n d a n d o

```
  0 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
  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 txt
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

pat = a n d a n d o

```
  0 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
  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 txt
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

pat = a n d a n d o

```
  0 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
  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 txt
-------------------------------------------------------------------
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

pat = a n d a n d o

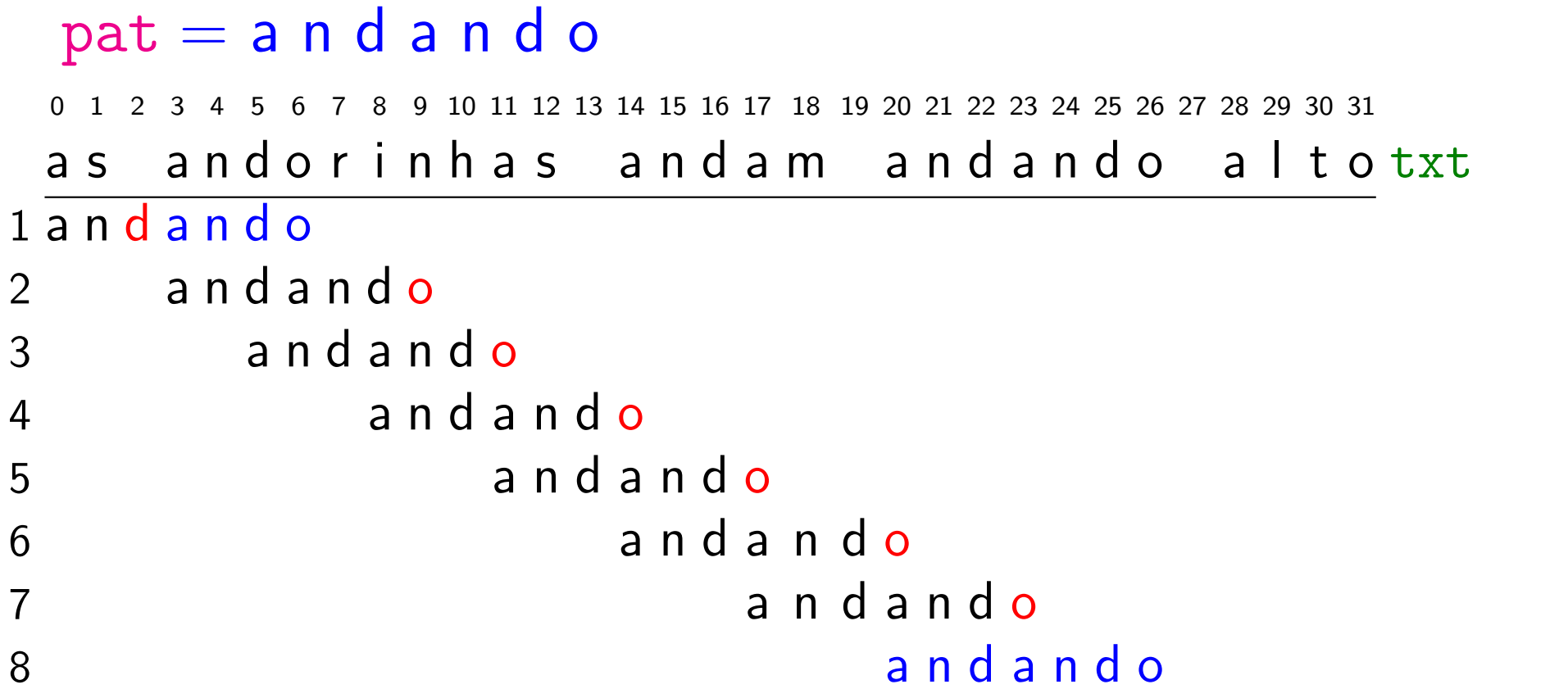
```
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
  a s   a n d o r i n h a s   a n d a m   a n d a n d o   a l t o txt
-------------------------------------------------------------------
1 a n d a n d o
2       a n d a n d o
3           a n d a n d o
4               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

pat = a n d a n d o

```
   0 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
   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 txt
  ---------------------------------------------------------------------------------------
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


pat = a n d a n d o
0 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
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 txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													

Boyer-Moore

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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	

Bad-character heuristic

Ideia ("*bad-character heuristic*"): calcular um deslocamento de modo que `txt[j]` fique emparelhado com a última ocorrência do caractere `txt[j]` em `pat`.

Bad-character heuristic

Ideia ("*bad-character heuristic*"): calcular um deslocamento de modo que `txt[j]` fique emparelhado com a última ocorrência do caractere `txt[j]` em `pat`.

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

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

Bad-character heuristic

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

`pat`

0	1	2	3	4	5	6
a	n	d	a	n	d	o

`right`

0	...	'a'	'b'	'c'	'd'	...	...	'n'	'o'	'p'	...	255
−1	...	3	−1	−1	5	...	...	4	6	−1	...	...

Classe BoyerMoore: esqueleto

```
static int R;                    /* tam. alfabeto */
static int  *right;   /* pulo bad-character */
static char *pat;
void BoyerMooreInit(char *pattern) {...}
int search(char *txt) {...}
```

BoyerMoore: pré-processamento

```
void BoyerMooreInit(char *pattern) {
   int m = strlen(pattern);
   R = 256;
   pat = mallocSafe((m+1)*sizeof(char));
   strcpy(pat, pattern);

   /* última ocorrência de c em pat */
   right = mallocSafe(R*sizeof(int));
   for (int c = 0; c < R; c++)
      right[c] = -1;

   for (int j = 0; j < m; j++)
      right[pat[j]] = j;
}
```

BoyerMoore: search()

Recebe strings pat e txt com $m \geq 1$ e $n \geq 0$,
e retorna o índice da primeira ocorrência de pat
em txt. Se pat não ocorre em txt, retorna n.

```
int search(char *txt) {
   int n = strlen(txt);
   int m = strlen(pat);
   int skip;
```

BoyerMoore: search()

```
    for (int i = 0; i <= n-m; i += skip) {
        skip = 0;
        for (int j = m-1; j >= 0; j--) {
            if (pat[j] != txt[i+j]) {
                int r = right[txt[i+j]]
                skip = MAX(1,j-r);
                break;
            }
        }
        if (skip == 0) return i; /* achou */
    }
    return n;                    /* não achou */
}
```

Pior caso

pat = 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	txt
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													a	b	a	a	a	a	a	a	a	a	a	a

Melhor caso

pat = a b c d e

```
  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
  ? ? ? ? x ? ? ? ? x  ?  ?  ?  ?  x  ?  ?  ?  ?  x  ?  ?  ?  txt
---------------------------------------------------------------
1 a b c d e
```

Melhor caso

pat = a b c d e

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

Melhor caso

pat = a b c d e

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

Melhor caso

pat = a b c d e

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

Melhor caso

pat = a b c d e

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

Conclusões

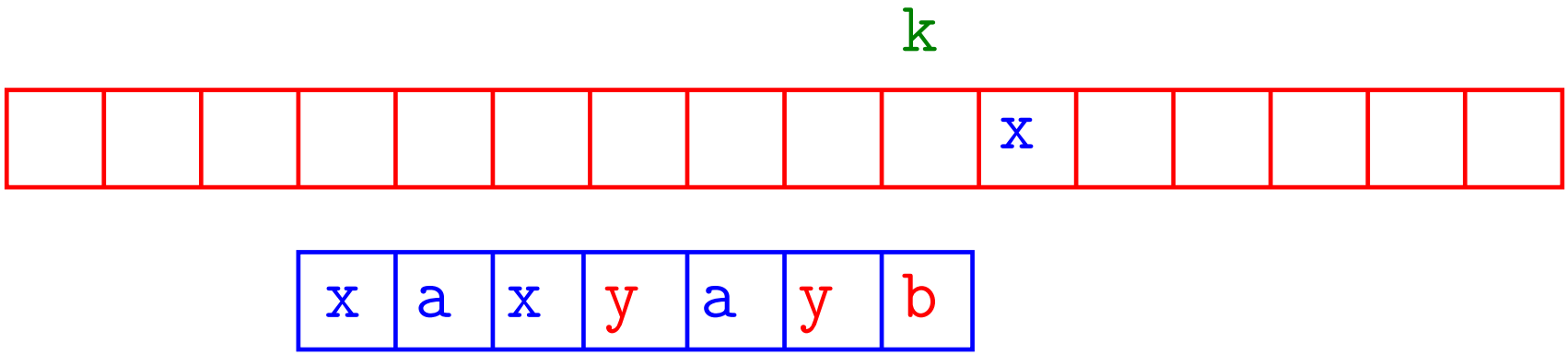
O consumo de tempo do algoritmo
`BoyerMoore` no pior caso é $O((n - m + 1)m)$.

O consumo de tempo do algoritmo
`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**.

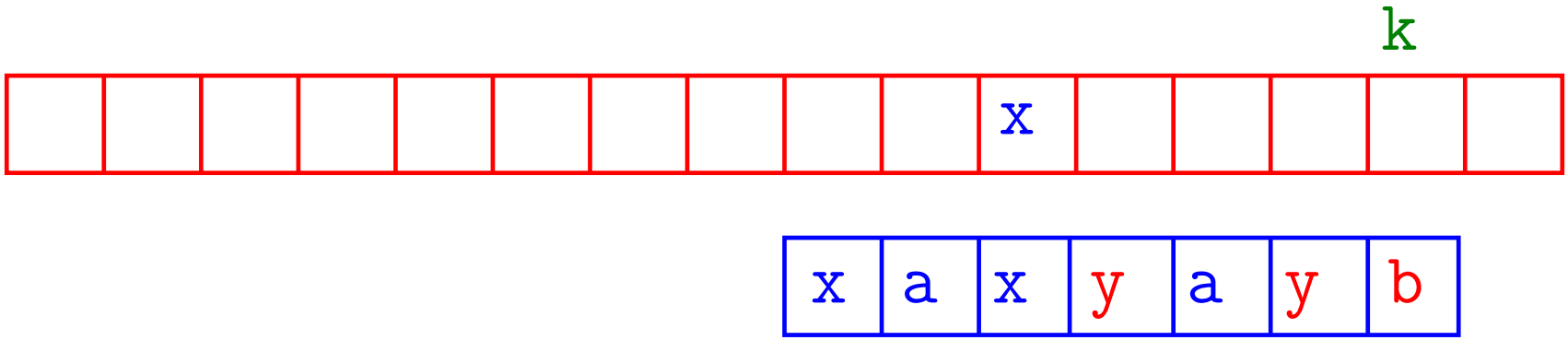
Bad-character heuristic: variante

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



Bad-character heuristic: variante

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



Bad-character heuristic: variante

pat = a n d a n d o

```
  0 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
  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 txt
1 a n d a n d o
```

Bad-character heuristic: variante

pat = a n d a n d o

```
  0 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
  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 txt
1 a n d a n d o
2             a n d a n d o
```

Bad-character heuristic: variante

pat = a n d a n d o

```
  0 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
  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 txt
1 a n d a n d o
2             a n d a n d o
3                 a n d a n d o
```

Bad-character heuristic: variante

pat = a n d a n d o

```
   0 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
   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 txt
  ---------------------------------------------------------------------------------------
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
```

Bad-character heuristic: variante

pat = a n d a n d o

```
   0 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
   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 txt
  ---------------------------------------------------------------------------------------
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
```

Bad-character heuristic: variante

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
1	a	b	a	b	b	a	b	a	b	b	a													
2			a	b	a	b	b	a	b	a	b	b	a											

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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									

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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							

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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						

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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				

Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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		

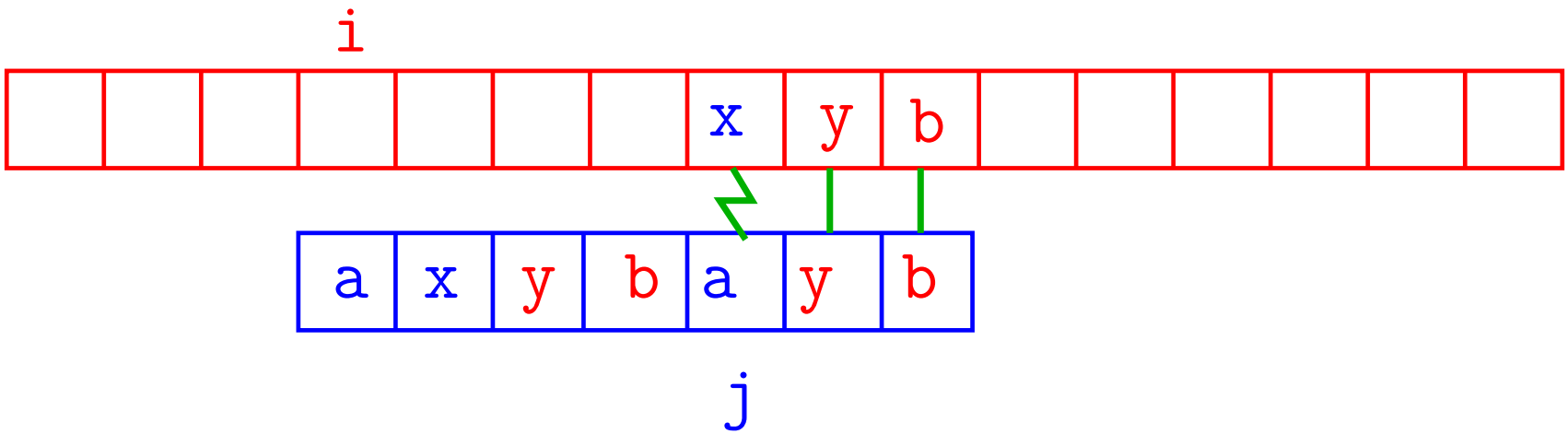
Bad-character heuristic: variante

pat = a b a b b a b a b b a

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	
	a	b	a	a	b	a	b	a	b	b	a	b	a	b	a	b	b	a	b	a	b	b	a	txt
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	

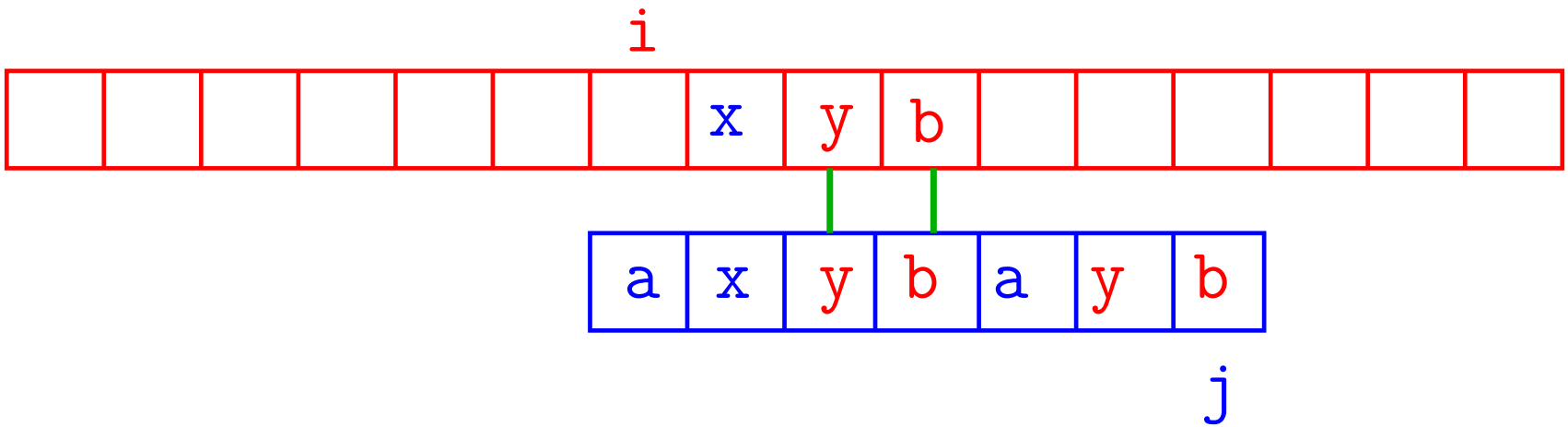
Segundo algoritmo de Boyer-Moore

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.



Good suffix heuristic

Não precisa conhecer o alfabeto explicitamente.

A implementação deve começar com
um pré-processamento de `pat`:
para cada `j` em $0, 1, \ldots,$ `m` $- 1$, devemos
calcular o maior `k` em $0, 1, \ldots,$ `m` $- 2$ tal que:

- `pat`[`j` . . `m`-1] é sufixo de `pat`[0 . . `k`] ou
- `pat`[0 . . `k`] é sufixo de `pat`[`j` . . `m`-1].

Chamemos de `bm[j]` esse valor `k`.

Good suffix heuristic

Exemplo 1:

	0	1	2	3	4	5
pat	c	a	a	b	a	a

	0	1	2	3	4	5
bm	−1	−1	−1	−1	2	4

Exemplo 2:

	0	1	2	3	4	5	6	7
pat	b	a	–	b	a	*	b	a

	0	1	2	3	4	5	6	7
bm	1	1	1	1	1	1	4	4

Good suffix heuristic

O vetor `bm[]` pode ser calculado facilmente em tempo $O(m^3)$.

Com mais trabalho, o vetor `bm[]` pode ser determinado em tempo $O(m)$.
Veja CLRS, seção 34.4.

Veja também a página do professor Paulo Feofiloff:
Busca de palavras em um texto

Apêndice: Rabin-Karp

Fonte: ADS: Boyer Moore String Search

Referência:
Algoritmo de Rabin-Karp para busca de substrings (PF)

Rabin-Karp

Criado por Richard M. Karp and Michael O. Rabin (1987).

O algoritmo também é conhecido como busca por impressão digital (*fingerprint search*).

Rabin-Karp

Criado por Richard M. Karp and Michael O. Rabin (1987).

O algoritmo também é conhecido como busca por impressão digital (*fingerprint search*).

Procura um segmento do texto que tenha o mesmo valor hash do padrão `pat`.

Usa hashing modular: módulo `Q`.

Se hash de `pat` é diferente do hash de todos os segmentos do texto então o padrão não ocorre no texto. A recíproca não vale: pode haver colisão.

Exemplo 1

pat.charAt(j)

j	0	1	2	3	4	
	2	6	5	3	5	% 997 = 613

txt.charAt(i)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3	
0	3	1	4	1	5	% 997 = 508											
1		1	4	1	5	9	% 997 = 201										
2			4	1	5	9	2	% 997 = 715									
3				1	5	9	2	6	% 997 = 971								
4					5	9	2	6	5	% 997 = 442							
5						9	2	6	5	3	% 997 = 929						
6							2	6	5	3	5	% 997 = 613					

← *return* i = 6

match

Basis for Rabin-Karp substring search

Exemplo 2

Procurar o padrão 12345 nos primeiros 100 mil dígitos da expansão decimal de π.

3141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117067982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381964428810975665933446128475648233786783165271201909145648566923460348610454326648213393607260249141273724587006606315588174881520920962829254091715364367892590360011330530548820466521384146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190702179860943702770539217176293176752384674818467669405132000568127145263560827785771342757789609173637178721468440901224953430146549585371050792279689258923542019956112129021960864034418159813629774771309960518707211349999998372978049951059731732816096318595024459455346908302642522308253344685035261931188171010003137838752886587533208381420617177669147303598253490428755468731159562863882353787593751957781857780532171226806613001927876611195909216420198938095257201065485863278865936153381827968230301952035301852968995773622599413891249721775283479131515574857242454150695950829533116861727855889075098381754637464939319255060400927701671139009848824012858361603563707660104710181942955596198946767837449448255379774726847104047534646208046684259069491293313677028989152104752162056966024058038150193511253382430035587640247496473263914199272604269922796782354781636009341721641219924586315030286182974555706749838505494588586926995690927210797509302955321165344987202755960236480665499119881834797753566369807426542527862551818417574672890977772793800081647060016145249192173217214772350141441973568548161361157352552133475741849468438523323907394143334547762416862518983569485562099219222184272550254256887671790494601653466804988627232791786085784383827967976681454100953883786360950680064225125205117392984896084128488626945604241965285022210661186306

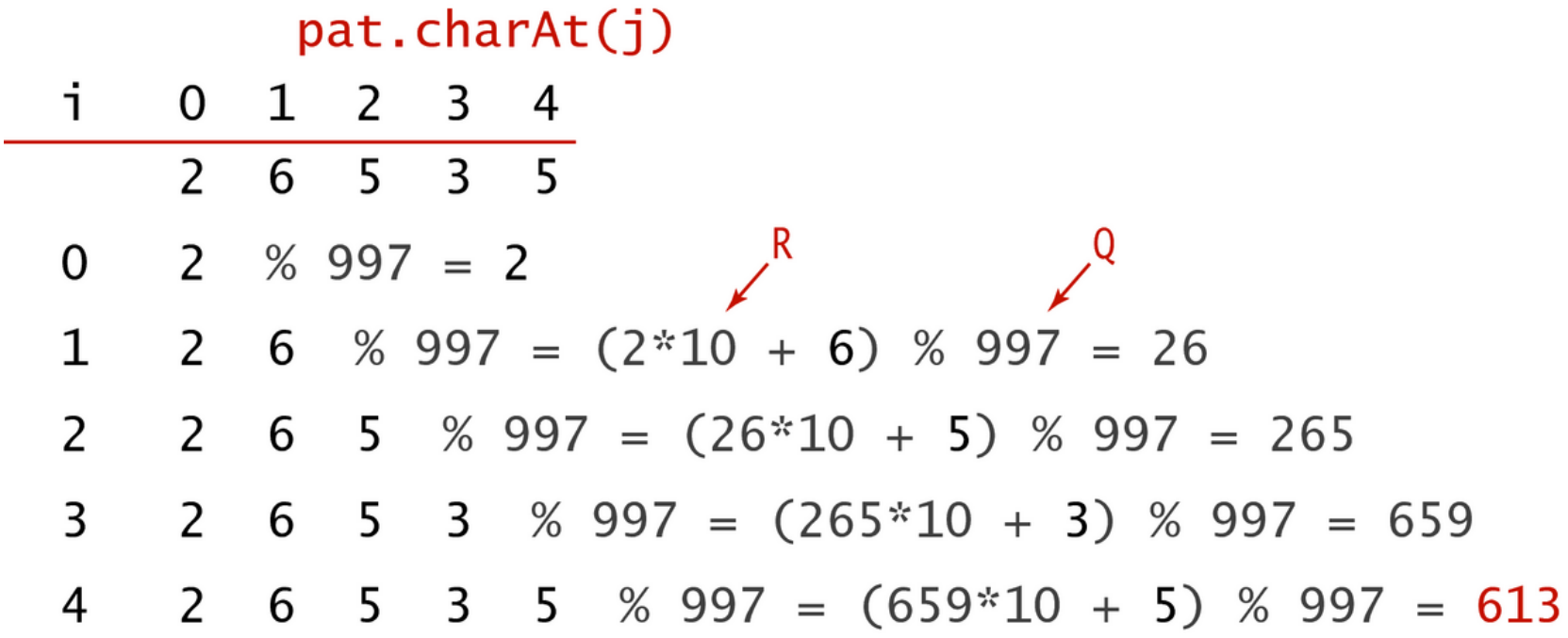
Algoritmo de Horner

Para calcular o valor de hash de uma string, usamos o algoritmo de Horner:

```
static long hash(char *key, int m) {
   long h = 0;
   for (int j = 0; j < m; j++)
      h = (h * R + key[j]) % Q;
   return h;
}
```

Escolha Q igual a um primo grande para evitar a chance de colisão.

Exemplo



Computing the hash value for the pattern with Horner's method

Ideia chave: hash de substrings consecutivas

Seja $t_i = \texttt{txt}[i]$ e $x_i =$ o inteiro $t_i t_{i-1} \ldots t_{i+m-1}$.

Assim,

$$x_{i-1} = t_{i-1}R^{m-1} + t_i R^{m-2} + \cdots + t_{i+m-2}$$
$$x_i = + t_i R^{m-1} + \cdots + t_{i+m-2}R + t_{i+m-1}.$$

Logo, $x_i = (x_{i-1} - t_{i-1}R^{m-1})R + t_{i+m-1}$.

Portanto, o valor $\texttt{hash}(x_i) = x_i \% Q$ pode ser obtido a partir do valor de $\texttt{hash}(x_{i-1}) = x_{i-1}\%Q$ em tempo constante:

$$\texttt{hash}(x_i) = ((\texttt{hash}(x_{i-1}) - t_{i-1}R^{m-1})R + t_{i+m-1})\%Q$$

Exemplo

i	...	2	3	4	5	6	7	...	
current value	1	4	1	5	9	2	6	5	*text*
new value		4	1	5	9	2	6	5	

	4	1	5	9	2	*current value*
–	4	0	0	0	0	
		1	5	9	2	*subtract leading digit*
			*	1	0	*multiply by radix*
	1	5	9	2	0	
				+	6	*add new trailing digit*
	1	5	9	2	6	*new value*

Key computation in Rabin-Karp substring search
(move right one position in the text)

Implementação

Evite *overflow* e números negativos:

- use um tipo-de-dados capaz de armazenar $\texttt{Q}^2$;
- na prática, escolha `Q` que caibe em um `int` ($2^{31} - 1$ é primo);
- faça as contas com `long`;
- tome o resto da divisão por `Q` depois de cada operação;
- some `Q` aos resultados intermediários quando necessário.

Exemplo: Q=997

$$\begin{aligned}(10000 + 535) \times 1000 &= (30 + 535) \times 3 \\ &= 565 \times 3 \\ &= 1695 \\ &= 698\end{aligned}$$

$$\begin{aligned}508 - 3 \times 10000 &= 508 - 3 \times (30) \\ &= 508 + 3 \times (-30) \\ &= 508 + 3 \times (997 - 30) \\ &= 508 + 3 \times 967 \\ &= 508 + 907 \\ &= 418\end{aligned}$$

Classe RabinKarp: esqueleto

```
static char *pat;
static long patHash;  /* hash do padrão */
static int m;
static long Q;
static int R = 256;
static long RM;

void RabinKarpInit(char *pattern) {...}
static long hash(char *key, int m) {...}
int search(char *txt) {...}
bool check(char *txt, int i) {...}
```

RabinKarp: pré-processamento

```
void RabinKarpInit(char * pattern) {
   m = strlen(pattern);
   pat = mallocSafe((m+1)*sizeof(char));
   strcpy(pat, pattern);
   Q = longRandomPrime();
   RM = 1;
   /* calcula R^(m-1)%Q */
   for (int i = 1; i <= m-1; i++)
      RM = (R * RM) % Q;
   patHash = hash(pat, m);
}
```

RabinKarp: search()

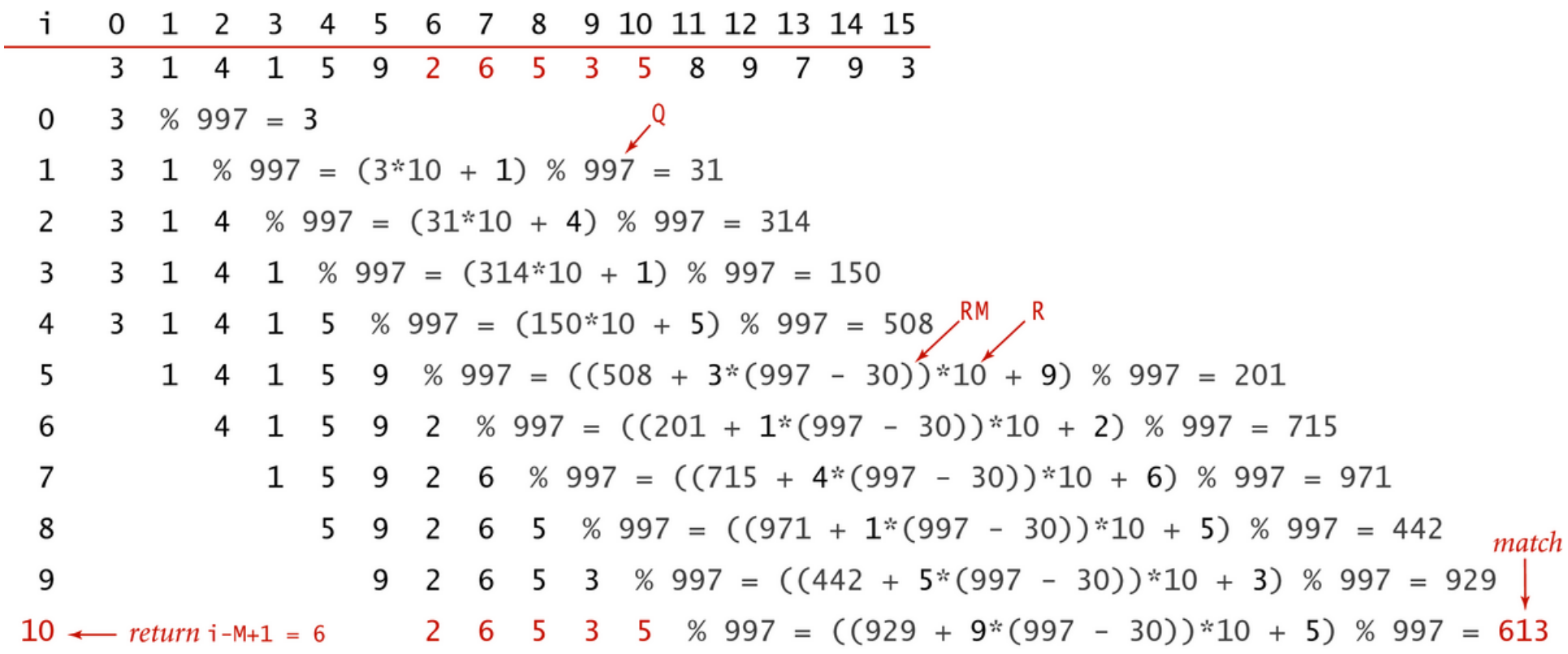
```
int search(char *txt) {
    int n = strlen(txt);
    long txtHash = hash(txt, m);
    if (patHash == txtHash && check(txt, 0))
        return 0;
    for (int i = 1; i <= n - m; i++) {
        txtHash = (txtHash + Q - RM*txt[i-1] % Q) % Q;
        txtHash = (txtHash * R + txt[i+m-1]) % Q;
        if (patHash == txtHash && check(txt, i))
            return i;                          /* achou */
    }
    return n;                                  /* não achou */
}
```

RabinKarp: check()

```
/* versão Las Vegas */
static bool check(char *txt, int i) {
   for (int j = 0; j < m; j++)
      if (pat[j] != txt[i+j])
         return false;
   return true;
}

/* versão Monte Carlo */
static bool check(char *txt, int i) {
   return true
}
```

Exemplo



Rabin-Karp substring search example

Monte Carlo *versus* Las Vegas

A versão Monte Carlo consome tempo linear,
mas pode dar uma resposta errada,
com baixíssima probabilidade.

A versão Las Vegas sempre dá a resposta certa,
mas pode consumir tempo não linear,
com baixíssima probabilidade.

Qual algoritmo é melhor

Força bruta é bom se o padrão e o texto não tiverem muitas auto-repetições.

`KMP` é rápido e tem a vantagem de nunca retroceder sobre o texto, o que é importante se o texto for dado como um fluxo contínuo (*streaming*).

`BoyerMoore` é provavelmente o mais rápido na prática.

`RabinKarp` é rápido mas pode dar resultados errados, com baixíssima probabilidade.

Implementações

Veja as implementações de busca de substrings:

- `glibc`: Implementation of strstr in glibc
- `cpython`: The stringlib Library
- Boyer-Moore-Horspool algorithm

Próximo passo

O que acontece se o padrão não é apenas uma string mas um conjunto de strings descrito por uma **expressão regular** como `A*|(A*BA*BA*)*` ou `((A*B|AC)D)`, por exemplo?

Essa generalização do problema de busca é muito importante. A solução envolve o conceito de autômato de estados não determinístico.