MAC0323 Algoritmos e Estruturas de Dados II

Edição 2020 – 2

AULA 28

Busca de Substrings

Substring search

Referências: Busca de substring (PF),
Substring Searching (SW), slides (SW), vídeo (SW).

Introdução

Problema: Dada uma string `pat` e uma string `txt`, encontrar uma ocorrência de `pat` em `txt`.

Exemplo: encontre `ATTGG` em:

```
TGGTAAGCGGTTCCTGCCCCGGCTCAGGGCCAAGAACAGATGAGACAGCTGAGTGATGGGCCAAACAGGATATCTGTGG
TAAGCAGTTCCTGCCCCGGCTCGGGGCCAAGAACAGATGGTCCCCAGATGCGGTCCAGCCCTCAGCAGTTTCTAGTGAA
TCATCAGATGTTTCCAGGGTGCCCCAAGGACCTGAAAATGACCCTGTACCTTATTTGAACTAACCAATCAGTTCGCTTC
TCGCTTCTGTTCGCGCGCTTCCGCTCTCCGAGCTCAATAAAAGAGCCCACAACCCCTCACTCGGCGCGCCAGTCTTCCG
ATAGACTGCGTCGCCCGGGTACCCGTATTCCCAATAAAGCCTCTTGCTGTTTGCATCCGAATCGTGGTCTCGCTGTTCC
TTGGGAGGGTCTCCTCTGAGTGATTGACTACCCACGACGGGGGTCTTTCATTTGGGGGCTCGTCCGGGATTTGGAGACC
CCTGCCCAGGGACCACCGACCCACCACCGGGAGGTAAGCTGGCCAGCAACTTATCTGTGTCTGTCCGATTGTCTAGTGT
CTATGTTTGATGTTATGCGCCTGCGTCTGTACTAGTTAGCTAACTAGCTCTGTATCTGGCGGACCCGTGGTGGAACTGA
CGAGTTCTGAACACCCGGCCGCAACCCTGGGAGACGTCCCAGGGACTTTGGGGGCCGTTTTTGTGGCCCGACCTGAGGA
AGGGAGTCGATGTGGAATCCGACCCCGTCAGGATATGTGGTTCTGGTAGGAGACGAGAACCTAAAACAGTTCCCGCCTC
CGTCTGAATTTTTGCTTTCGGTTTGGAACCGAAGCCGCGCGTCTTGTCTGCTGCAGCATCGTTCTGTGTTGTCTCTGTC
TGACTGTGTTTCTGTATTTGTCTGAAAATTAGGGCCAGACTGTTACCACTCCCTTAAGTTTGACCTTAGGTCACTGGAA
AGATGTCGAGCGGATCGCTCACAACCAGTCGGTAGATGTCAAGAAGAGACGTTGGGTTACCTTCTGCTCTGCAGAATGG
CCAACCTTTAACGTCGGATGGCCGCGAGACGGCACCTTTAACCGAGACCTCATCACCCAGGTTAAGATCAAGGTCTTTT
CACCTGGCCCGCATGGACACCCAGACCAGGTCCCCTACATCGTGACCTGGGAAGCCTTGGCTTTTGACCCCCCTCCCTG
GGTCAAGCCCTTTGTACACCCTAAGCCTCCGCCTCCTCTTCCTCCATCCGCCCCGTCTCTCCCCCTTGAACCTCCTCGT
TCGACCCCGCCTCGATCCTCCCTTTATCCAGCCCTCACTCCTTCTCTAGGCGCCGGAATTCGTTAACTCGAGGATCCGG
CTGTGGAATGTGTGTCAGTTAGGGTGTGGAAAGTCCCCAGGCTCCCCAGCAGGCAGAAGTATGCAAAGCATGCATCTCA
ATTAGTCAGCAACCAGGTGTGGAAAGTCCCCAGGCTCCCCAGCAGGCAGAAGTATGCAAAGCATGCATCTCAATTAGTC
AGCAACCATAGTCCCGCCCCTAACTCCGCCCATCCCGCCCCTAACTCCGCCCAGTTCCGCCCATTCTCCGCCCCATGGC
TGACTAATTTTTTTTATTTATGCAGAGGCCGAGGCCGCCTCGGCCTCTGAGCTATTCCAGAAGTAGTGAGGAGGCTTTT
TTGGAGGCCTAGGCTTTTGCAAAAAGCTGCCCAAGCTGATCCCCGGGGGCAATGAGATATGAAAAAGCCTGAACTCACC
GCGACGTCTGTCGAGAAGTTTCTGATCGAAAAGTTCGACAGCGTCTCCGACCTGATGCAGCTCTCGGAGGGCGAAGAAT
CTCGTGCTTTCAGCTTCGATGTAGGAGGGCGTGGATATGTCCTGCGGGTAAATAGCTGCGCCGATGGTTTCTACAAAGA
TCGTTATGTTTATCGGCACTTTGCATCGGCCGCGCTCCCGATTCCGGAAGTGCTTGACATTGGGGAATTCAGCGAGAGC
```

Introdução

Dizemos que um vetor $\mathtt{pat}[0\,..\,\mathtt{m}{-}1]$ **casa com** $\mathtt{txt}[0\,..\,\mathtt{n}{-}1]$ **a partir de** $\mathtt{i}$ se

$$\mathtt{pat}[0\,..\,\mathtt{m}-1] = \mathtt{txt}[\mathtt{i}\,..\,\mathtt{i}{+}\mathtt{m}{-}1]$$

para algum $\mathtt{i}$ em $[0\,..\,\mathtt{n}{-}\mathtt{m}]$.

Exemplo:

	0	1	2	3	4	5	6	7	8	9
txt	x	c	b	a	b	b	c	b	a	x

	0	1	2	3
pat	b	c	b	a

$\mathtt{pat}[0\,..\,3]$ casa com $\mathtt{txt}[0\,..\,9]$ a partir de 5.

Busca de substrings

Problema alternativo: 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

	0	1	2	3	4	5	6	7	8	9
`txt`	b	b	a	b	a	b	a	c	b	a

	0	1	2	3
`pat`	b	a	b	a

`pat` ocorre 2 vezes em `txt`.

Algoritmo de força bruta

`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													

Algoritmo de força bruta

`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 de força bruta

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 de força bruta

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 de força bruta

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	

Algoritmo de força bruta

Devolve a primeira das 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++) {
    for (j = 0; j < m; j++)
      if(txt[i+j] != pat[j])
        break;
    if (j == m) return i;
  }
  return n;
}
```

Algoritmo de força bruta

Relação invariante: no início de cada iteração do "`for (j = 0; ...)`" vale que

(i0) $\texttt{pat}[0\,.\,.\,\texttt{j}{-}1] = \texttt{txt}[\texttt{i}\,.\,.\,\texttt{i}{+}\texttt{j}{-}1]$

Consumo de tempo

Consumo de tempo da função `search`().

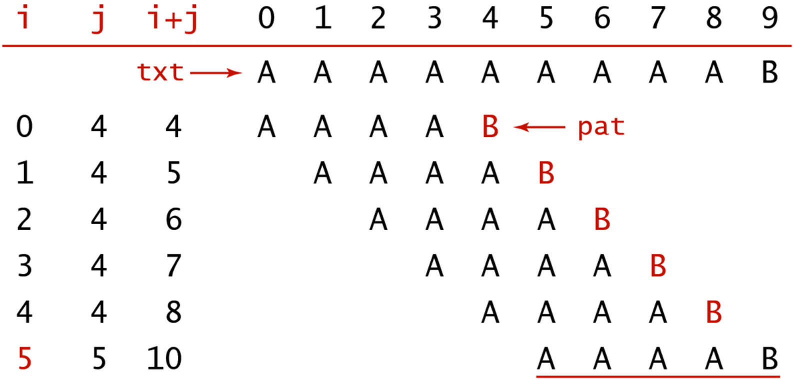
linha	todas as execuções da linha
1-2	$= 1$
3	$= n - m + 1$
4	$\leq (n - m + 1)(m + 1)$
5	$\leq (n - m + 1)\, m$
6	$\leq (n - m + 1)$
7	$= n - m$
8-9	$= 1$
total	$< 3\,(n - m + 2) + 2\,(n - m + 1)(m + 1)$
	$= \mathrm{O}((n - m + 1)m)$

Pior caso

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

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

Pior caso



i	j	i+j	0	1	2	3	4	5	6	7	8	9
		txt →	A	A	A	A	A	A	A	A	A	B
0	4	4	A	A	A	A	B	← pat				
1	4	5		A	A	A	A	B				
2	4	6			A	A	A	A	B			
3	4	7				A	A	A	A	B		
4	4	8					A	A	A	A	B	
5	5	10						A	A	A	A	B

Brute-force substring search (worst case)

Melhor caso

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

Conclusões

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

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

Isto significa que no pior caso o consumo de tempo é essencialmente proporcional a $m\,n$.

Em geral o algoritmo é rápido e
faz não mais que $1.1 \times n$ comparações.

Próximos passos

Existe algoritmo mais rápido que o força bruta?

Existe algoritmo que faz apenas `n` comparações entre caracteres?

Existe algoritmo que faz menos que `n` comparações?

Algoritmo KMP para busca de substring

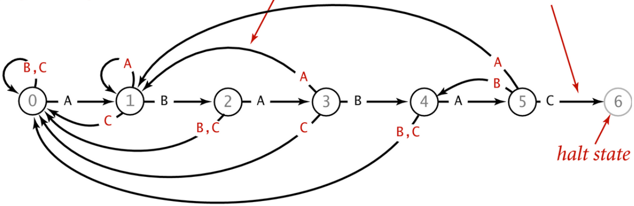
internal representation

j		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

mismatch transition (back up)

match transition (increment)

graphical representation



DFA corresponding to the string A B A B A C

S&W 5.3

~pf/estruturas-de-dados/aulas/kmp.html

Ideia básica do algoritmo

$P =$ 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	T
0	a																							

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									
15													a	b	a	b								

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									
15													a	b	a	b								
16													a	b	a	b	b							

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									
15													a	b	a	b								
16													a	b	a	b	b							
17													a	b	a	b	b	a						

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									
15													a	b	a	b								
16													a	b	a	b	b							
17													a	b	a	b	b	a						
18													a	b	a	b	b	a	b					

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									
15													a	b	a	b								
16													a	b	a	b	b							
17													a	b	a	b	b	a						
18													a	b	a	b	b	a	b					
19													a	b	a	b	b	a	b	a				

Ideia básica do algoritmo

$P =$ 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	T
0	a																							
1	a	b																						
2	a	b	a																					
3				a																				
4				a	b																			
5				a	b	a																		
6				a	b	a	b																	
7						a	b	a																
8						a	b	a	b															
9						a	b	a	b	b														
10						a	b	a	b	b	a													
11						a	b	a	b	b	a	b												
12						a	b	a	b	b	a	b	a											
13						a	b	a	b	b	a	b	a	b										
14													a	b	a									
15													a	b	a	b								
16													a	b	a	b	b							
17													a	b	a	b	b	a						
18													a	b	a	b	b	a	b					
19													a	b	a	b	b	a	b	a				
20													a	b	a	b	b	a	b	a	b			

Ideia básica do algoritmo

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

Ideia básica do algoritmo

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

Ideia geral

Quando encontramos um conflito entre `txt[i]` e `pat[j]`, **não** é necessário passar a comparar `txt[i-j+1 . . ]` com `pat[0 . . ]`.

Basta:

> *encontrar o comprimento do maior prefixo de $pat[0\,.\,.\,]$ que é sufixo de $txt[\,.\,.\,i]$,*

ou seja,

> *encontrar o maior k tal que $pat[0\,.\,.\,k-1]$ é igual a $txt[i-k+1\,.\,.\,i]$ que é igual a $pat[j-k+1\,.\,.\,j-1]+txt[i]$,*

e passar a comparar `txt[i+1 . . ]` com `pat[k..]`.

Ideia geral

Exemplo: texto `CAABAABAAAA` e padrão `AABAAA`: depois do conflito entre `txt[6]` e `pat[5]`, não precisamos retroceder no texto: podemos continuar e comparar `txt[7 . . ]` com `pat[3 . . ]`:

	C	A	A	B	A	A	B	A	A	A	A
uma tentativa:		A	A	B	A	A	A				
não precisa tentar:			A	A	B	A	A	A			
não precisa tentar:				A	A	B	A	A	A		
próxima tentativa:					A	A	B	A	A	A	

Algoritmo KMP

Examina os caracteres de `txt` um a um, da esquerda para a direita, sem nunca retroceder.

Em cada iteração, o algoritmo sabe qual posição `k` de `pat` deve ser emparelhada com a próxima posição `i`+1 de `txt`.

Ou seja, no fim de cada iteração, o algoritmo sabe qual índice `k` deve fazer o papel de `j` na próxima iteração.

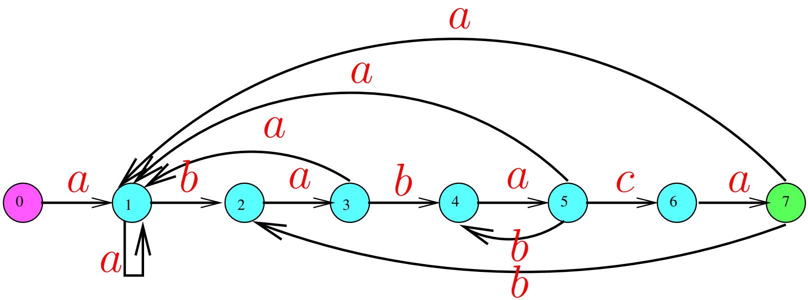
Algoritmo `KMP`

O algoritmo `KMP` usa uma tabela `dfa[][]` que armazena os `índices mágicos` `k`.

O nome da tabela deriva da expressão *deterministic finite-state automaton*.

As colunas da tabela são indexadas pelos índices `0..m-1` do padrão e as linhas são indexadas pelo alfabeto, que é o conjunto de todos os caracteres do texto e do padrão.

Autômato de estados determinístico (DFA)



$0 \,..\, 7$ = conjunto de estados

Σ = {a, b, c} = alfabeto

δ = função de transição

0 é estado inicial e 7 é estado final

Exemplo: pat = ABABAC

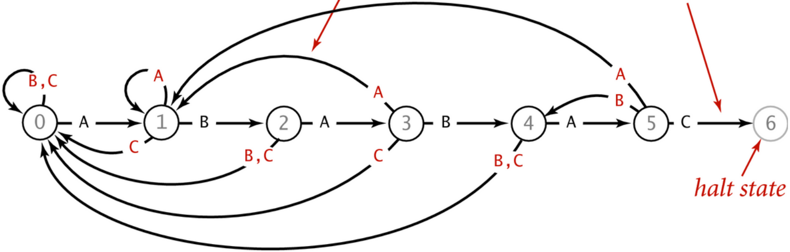
internal representation

j		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

mismatch transition (back up)

match transition (increment)

graphical representation



halt state

Autômato finito determinístico (DFA)

O algoritmo KMP simula o funcionamento do autômato de estados.

O autômato começa no estado 0 e examina os caracteres do texto, um de cada vez, da esquerda para a direita, mudando para um novo estado cada vez que lê um caractere do texto.

Se atingir o estado `m`, dizemos que o autômato reconheceu ou aceitou o padrão.

Se chegar ao fim do texto sem atingir o estado `m`, sabemos que o padrão não ocorre no texto.

Autômato finito determinístico (DFA)

O autômato está no estado `j` se acabou de casar os `j` primeiros caracteres do padrão com um segmento do texto, ou seja, se acabou de casar `pat[0 . . j-1]` com `txt[i-j . . i-1]`.

Para cada estado `j`, a transição que corresponde ao caractere `pat[j]` é de casamento e leva ao estado `j+1`.

Todas as outras transições que começam no estado `j` são de conflito e levam a um estado $\leq$ `j`.

O autômato de estados é uma ideia **muito importante** em compilação, na teoria da computação, etc.

Autômato finito determinístico (DFA)

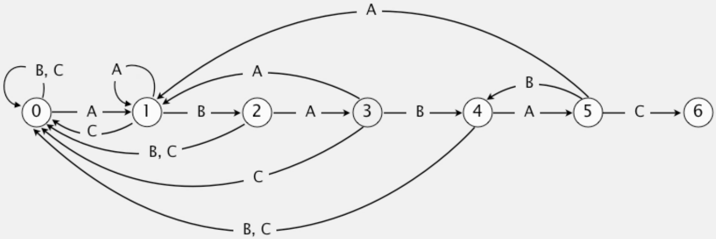
Um **autômato finito** é formado por uma 5-upla $(\mathtt{Q}, \Sigma, \delta, \mathtt{q}_0, \mathtt{F})$, onde

- $\mathtt{Q}$ é um conjunto finito de estados,
- Σ é um conjunto finito chamado alfabeto,
- $\delta : \mathtt{Q} \times \Sigma \to \mathtt{Q}$ é a função de transição,
- $\mathtt{q}_0 \in \mathtt{Q}$ é o estado inicial, e
- $\mathtt{F} \subseteq \mathtt{Q}$ é o conjunto de aceitação.

Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	1
dfa[][j]	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

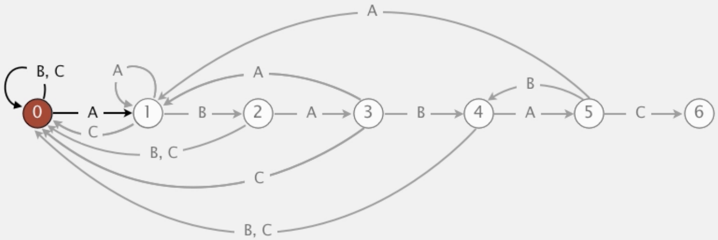


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	1
dfa[][j]	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

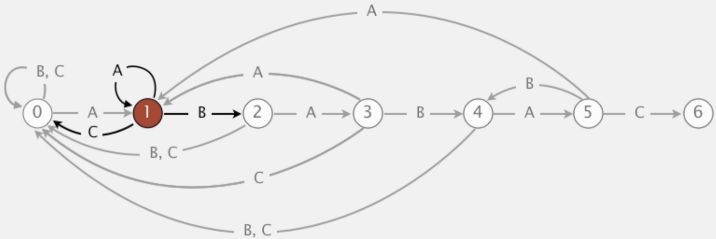


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	1
dfa[][j]	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

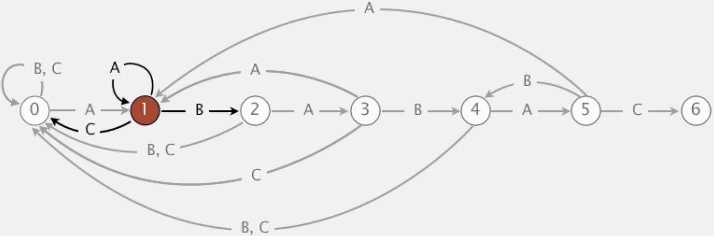


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

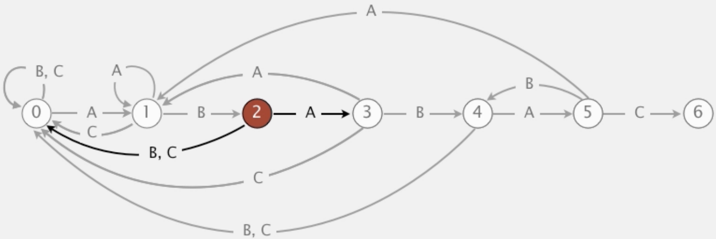


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

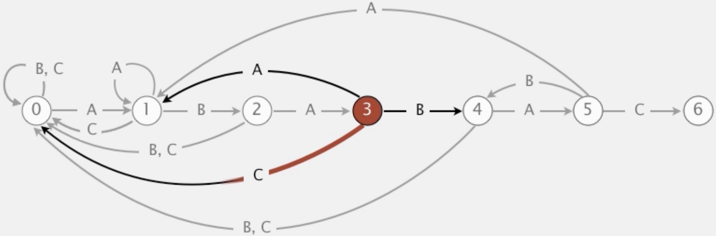


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

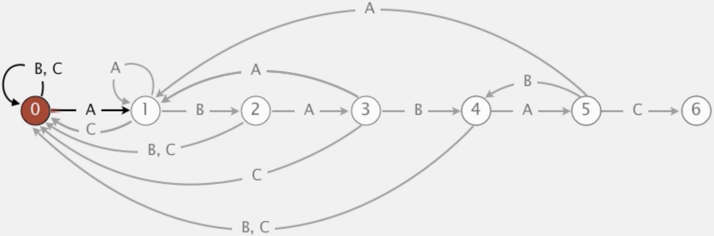
		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6



Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

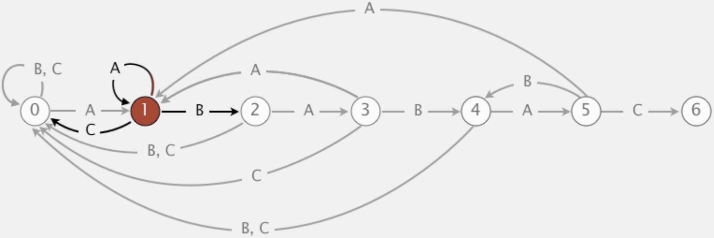


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

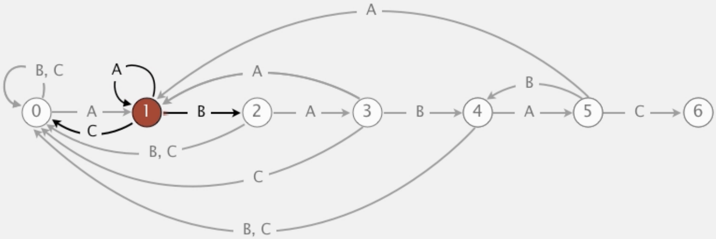
		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	1
dfa[][j]	B	0	2	0	4	0	4
	C	0	0	0	0	0	6



Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

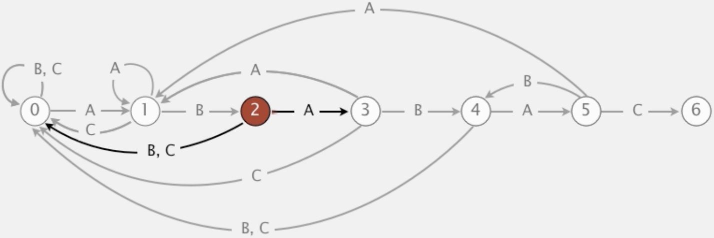


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1	1	3	1	5	1
`dfa[][j]`	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

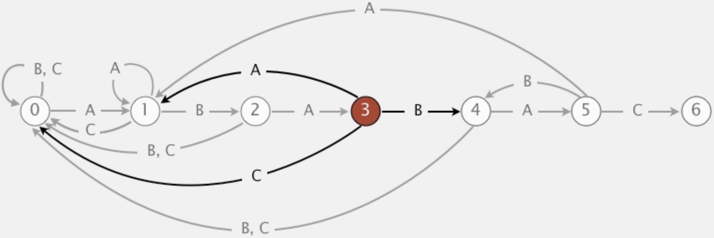


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

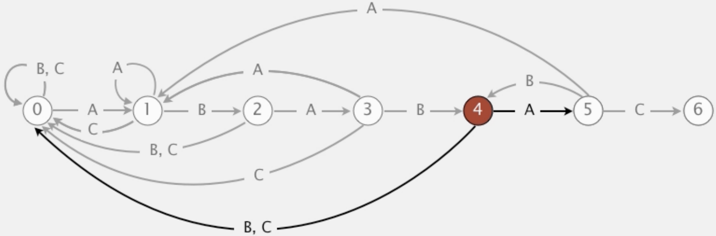


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	1
dfa[][j]	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

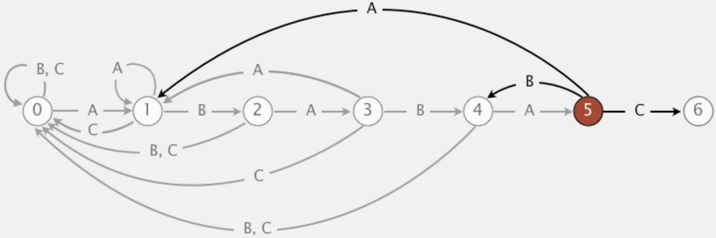


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	1
dfa[][j]	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

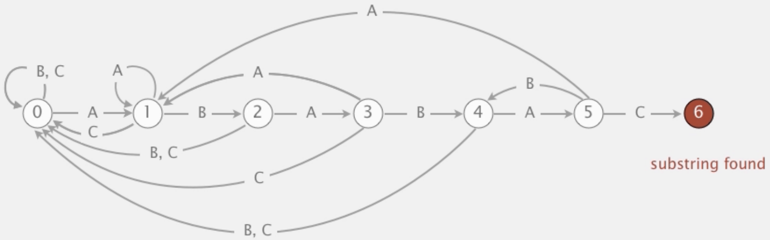


Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

↑

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6



substring found

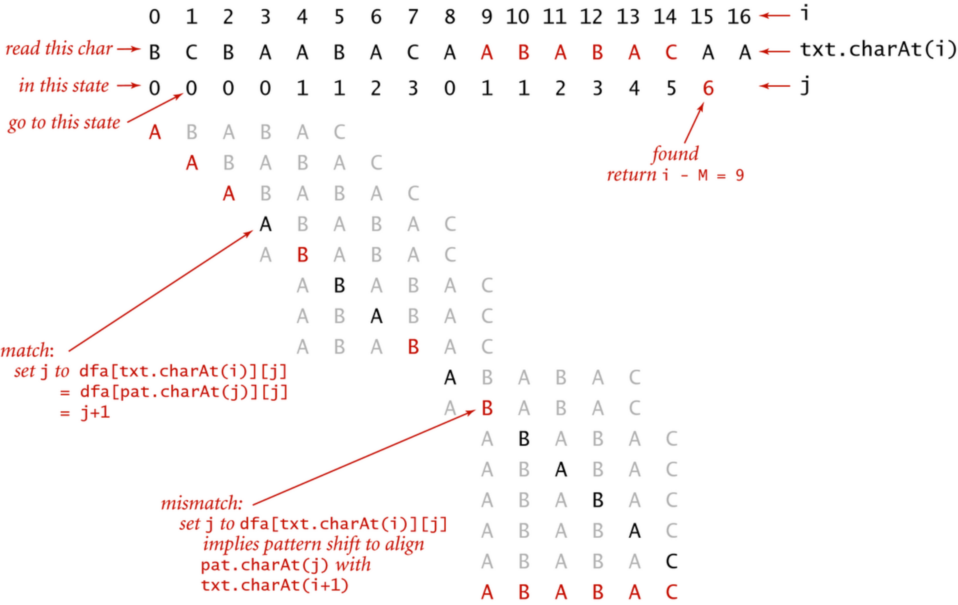
Algoritmo KMP

Retorna a posição a partir de onde pat ocorre em txt; se pat não ocorre em txt retorna n.

```
int search(char *txt) {
   int i, n = strlen(txt);
   int j, m = strlen(pat);

   for (i=0, j=0; i < n && j < m; i++)
      j = dfa[txt[i]][j];

   if (j == m) return i - m;
   return n;
}
```



Trace of KMP substring search (DFA simulation) for A B A B A C

Invariantes

A função `search()` de `KMP` tem os seguintes invariantes.

Imediatamente antes do teste `i < n && j < m` vale que:

- `pat` não ocorre em `txt[0 . . i-1]`;
- `pat[0 . . k]` é diferente de `txt[i-k . . i]` para todo `k` no conjunto `j+1 . . m-1`; e
- `pat[0 . . j-1]` é igual a `txt[i-j . . i-1]`.

Autômato de estados determinístico (DFA)

A tabela `dfa[][]` representa uma máquina imaginária conhecida como **autômato de estados** (*deterministic finite-state automaton, DFA*).

Os estados do autômato correspondem aos índices 0 . . `m`−1 de `pat`.

Também há um estado *final* `m`.

Para cada estado e cada caractere do alfabeto, há uma transição que leva desse estado a um outro.

Construção do DFA

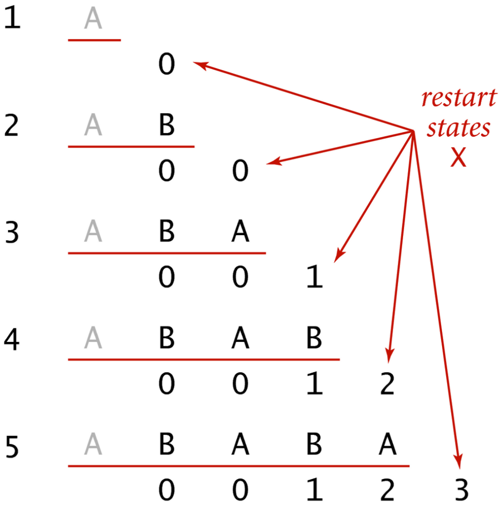
Para construir a tabela `dfa[][]` que representa o autômato, podemos pré-processar o padrão `pat` desde que o alfabeto de `txt` seja conhecido.

Para qualquer caractere `c` do alfabeto e qualquer `j` em 0 . . `m`−1, o valor de `dfa[c][j]` é

> *o comprimento do maior prefixo*
> *de `pat`[0 . . `j`] que é sufixo*
> *de `pat`[0 . . `j`−1]+`c`.*

Uma implementação literal dessa definição faria cerca de $\mathtt{Rm}^3$ comparações entre caracteres para calcular a tabela `dfa[][]`, sendo `R` o número de caracteres do alfabeto.

Exemplo: padrão ABABAC e alfabeto A B C



DFA simulations to compute restart states for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Include one state for each character in pattern (plus accept state).

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A						
dfa[][j]	B						
	C						

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Include one state for each character in pattern (plus accept state).

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A						
dfa[][j]	B						
	C						

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Match transition. If in state `j` and next char `c == pat.charAt(j)`, go to `j+1`.

first j characters of pattern have already been matched

next char matches

now first j+1 characters of pattern have been matched

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1		3		5	
	B		2		4		
	C						6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1		3		5	
`dfa[][j]`	B		2		4		
	C						6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1		3		5	
dfa[][j]	B	0	2		4		
	C	0					6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1		3		5	
`dfa[][j]`	B	0	2		4		
	C	0					6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1	1	3		5	
`dfa[][j]`	B	0	2		4		
	C	0	0				6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3		5	
	B	0	2		4		
	C	0	0				6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
`dfa[][j]`	A	1	1	3		5	
	B	0	2	0	4		
	C	0	0	0			6

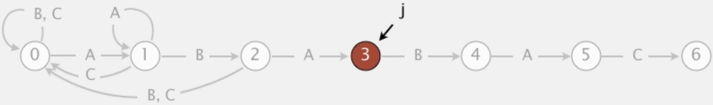
Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3		5	
dfa[][j]	B	0	2	0	4		
	C	0	0	0			6

Constructing the DFA for KMP substring search for A B A B A C

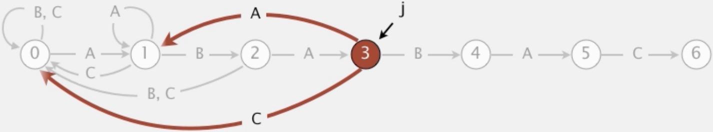


Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
`dfa[][j]`	A	1	1	3	1	5	
	B	0	2	0	4		
	C	0	0	0	0		6

Constructing the DFA for KMP substring search for A B A B A C

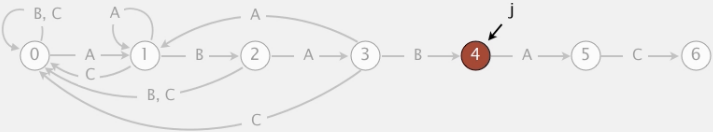


Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4		
	C	0	0	0	0		6

Constructing the DFA for KMP substring search for A B A B A C

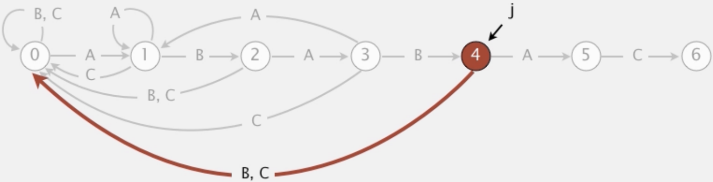


Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4	0	
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C

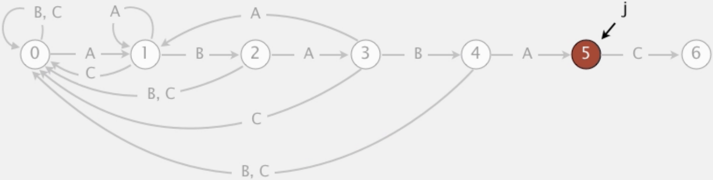


Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	
dfa[][j]	B	0	2	0	4	0	
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C

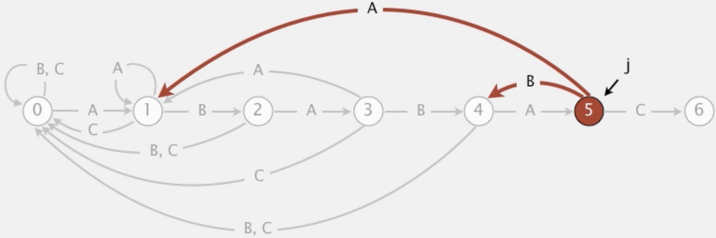


Knuth-Morris-Pratt demo: DFA construction

Mismatch transition: back up if `c != pat.charAt(j)`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
`dfa[][j]`	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

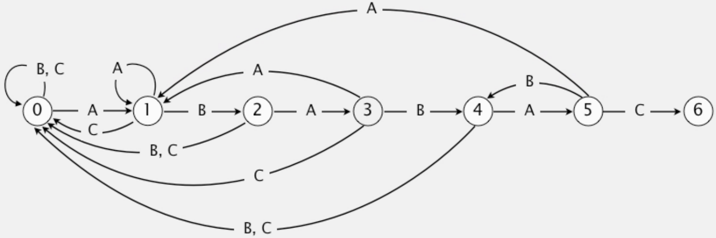
Constructing the DFA for KMP substring search for A B A B A C



Knuth-Morris-Pratt demo: DFA construction

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1	1	3	1	5	1
`dfa[][j]`	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



Knuth-Morris-Pratt demo: DFA construction in linear time

Include one state for each character in pattern (plus accept state).

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A						
dfa[][j]	B						
	C						

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Match transition. For each state `j`, `dfa[pat.charAt(j)][j] = j+1`.

first j characters of pattern have already been matched

now first j+1 characters of pattern have been matched

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1		3		5	
`dfa[][j]`	B		2		4		
	C						6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For state `0` and char `c != pat.charAt(j)`, set `dfa[c][0] = 0`.

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1		3		5	
`dfa[][j]`	B	0	2		4		
	C	0					6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For state `0` and char `c != pat.charAt(j)`, set `dfa[c][0] = 0`.

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1		3		5	
dfa[][j]	B	0	2		4		
	C	0					6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of empty string

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1		3		5	
	B	0	2		4		
	C	0					6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of empty string

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
`dfa[][j]`	A	1	1	3		5	
	B	0	2		4		
	C	0	0				6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of empty string

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3		5	
	B	0	2		4		
	C	0	0				6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3		5	
dfa[][j]	B	0	2		4		
	C	0	0				6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3		5	
dfa[][j]	B	0	2	0	4		
	C	0	0	0			6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3		5	
	B	0	2	0	4		
	C	0	0	0			6

Constructing the DFA for KMP substring search for A B A B A C

Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3		5	
dfa[][j]	B	0	2	0	4		
	C	0	0	0			6

Constructing the DFA for KMP substring search for A B A B A C

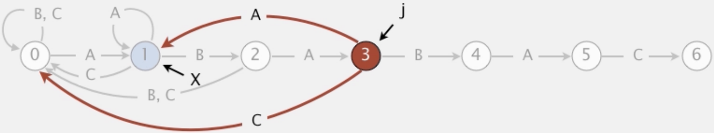
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4		
	C	0	0	0	0		6

Constructing the DFA for KMP substring search for A B A B A C



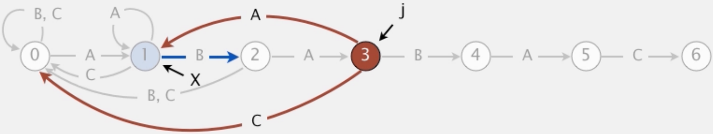
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4		
	C	0	0	0	0		6

Constructing the DFA for KMP substring search for A B A B A C



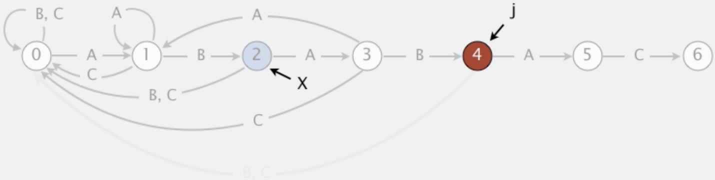
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A B

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
	A	1	1	3	1	5	
dfa[][j]	B	0	2	0	4		
	C	0	0	0	0		6

Constructing the DFA for KMP substring search for A B A B A C



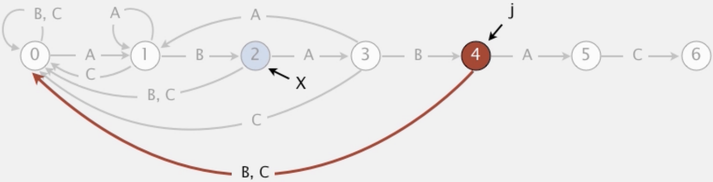
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A B

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4	0	
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



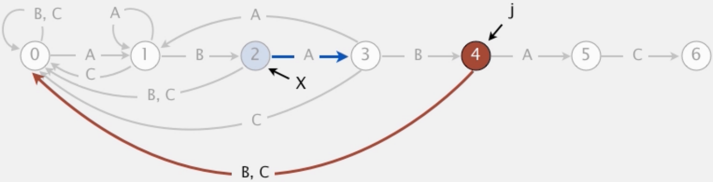
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A B

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4	0	
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



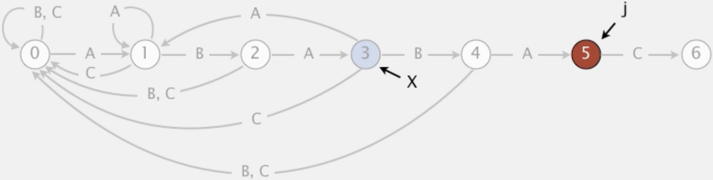
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A B A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	
	B	0	2	0	4	0	
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



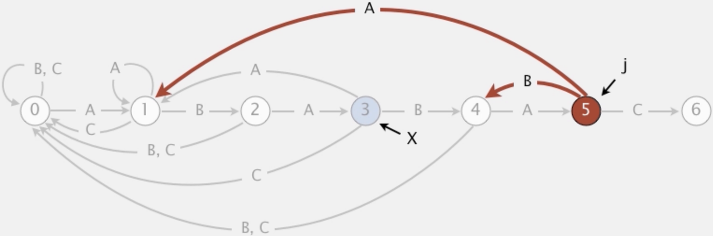
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A B A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



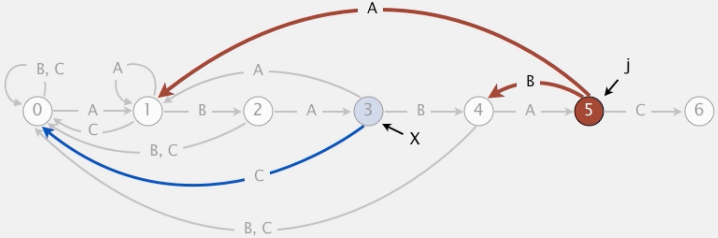
Knuth-Morris-Pratt demo: DFA construction in linear time

Mismatch transition. For each state `j` and char `c != pat.charAt(j)`, set `dfa[c][j] = dfa[c][X]`; then update `X = dfa[pat.charAt(j)][X]`.

X = simulation of B A B A

		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

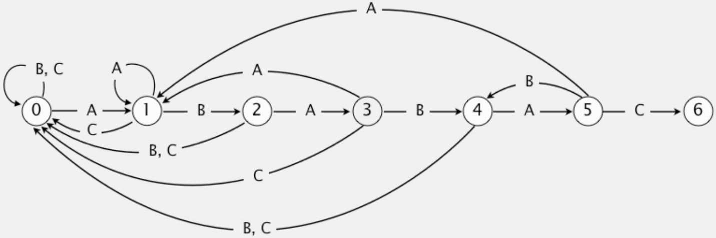
Constructing the DFA for KMP substring search for A B A B A C



Knuth-Morris-Pratt demo: DFA construction in linear time

		0	1	2	3	4	5
`pat.charAt(j)`		A	B	A	B	A	C
	A	1	1	3	1	5	1
`dfa[][j]`	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C



Construção da DFA

Trecho de código do KMP que constroi o DFA.

```
dfa[pat[0]][0] = 1;
for (int j = 1, X = 0; j < m; j++) {
   for (int c = 0; c < R; c++)
      /* copie casos de conflito */
      dfa[c][j] = dfa[c][X];

   /* defina casos de casamento */
   dfa[pat[j]][j] = j+1;

   /* atualize estado de reinício */
   X = dfa[pat[j]][X];
}
```

Construção da DFA: programação dinâmica

$\mathtt{dfa[c][j]} =$ maior $\mathtt{k}$ tal que

$\mathtt{pat}[0\,..\,\mathtt{k}\text{-}1] = \mathtt{pat}[\mathtt{j}\text{-}\mathtt{k}\text{+}1\,..\,\mathtt{j}\text{-}1] + \mathtt{c}$

Para $\mathtt{j}$ = 0:

$\mathtt{dfa[c][0]} = 1$, se $\mathtt{pat[0]} = \mathtt{c}$

0, se $\mathtt{pat[0]} \neq \mathtt{c}$

Para $\mathtt{j}$ > 0:

$\mathtt{dfa[c][j]} = \mathtt{dfa[c][j\text{-}1]\text{+}1}$, se $\mathtt{pat[j]} = \mathtt{c}$

$\mathtt{dfa[c][X]}$, se $\mathtt{pat[j]} \neq \mathtt{c}$,

onde $\mathtt{X}$ é o maior valor tal que

$\mathtt{pat}[0\,..\,\mathtt{X}] = \mathtt{pat}[\,..\,\mathtt{j}\text{-}1] + \mathtt{c}$.

Biblioteca KMP: esqueleto

```
static int R = 256;
static char *pat;
/* dfa[][] representa o autômato */
static int **dfa;
void KMPInit(char *pat) {...}
int search(char *txt) {...}
```

KMP: pré-processamento

```
void KMPInit(char *s) {
   int m = strlen(s);
   pat = mallocSafe((m+1)*sizeof(char));
   strcpy(pat, s);
   dfa = alocaMatriz(R, m);
   dfa[pat[0]][0] = 1;
   for (int j = 1, X = 0; j < m; j++) {
      /* calcule dfa[][j] */
      for (int c = 0; c < R; c++)
         dfa[c][j] = dfa[c][X];
      dfa[pat[jcor]][j] = j+1;
      X = dfa[pat[j]][X];
   }
}
```

KMP: search()

```
int search(char *txt) {
   int i, n = strlen(txt);
   int j, m = strlen(pat);

   for (i=0, j=0; i < n && j < m; i++)
      j = dfa[txt[i]][j];

   if (j == m) return i - m;
   return n;
}
```

Consumo de tempo

O consumo de tempo do
algoritmo KMP é $O(\mathtt{m} + \mathtt{n})$.

Proposição. O algoritmo KMP examina não mais que m + n caracteres.

Se levarmos em conta o tamanho do alfabeto, R, o consumo de tempo para construir o DFA é mR.