SBVORIN: Organização e Recuperação da Informação

Aula 10: Algoritmos de Busca de Substrings

Bacharelado em Ciência da Computação
Prof. Dr. David Buzatto

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

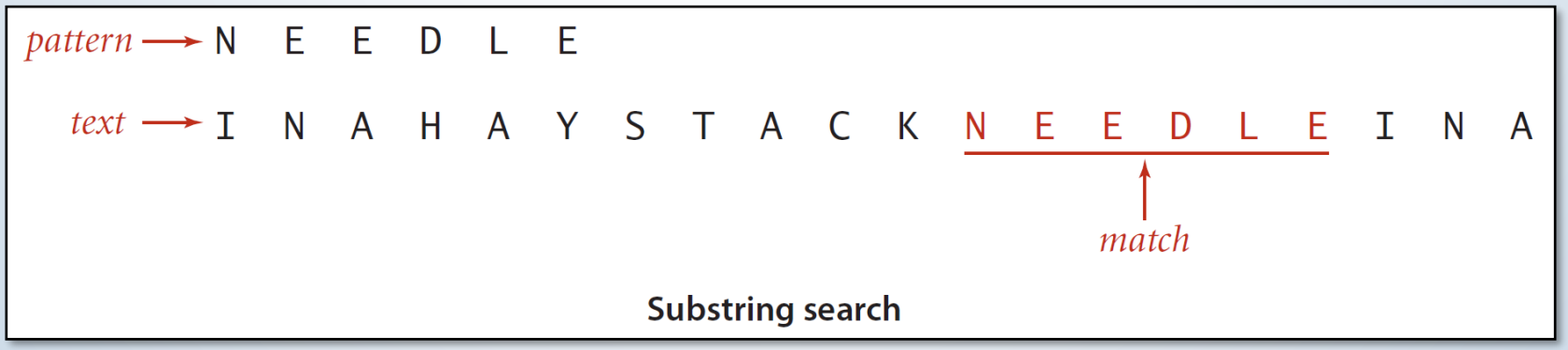
Introdução

- **Problema de Busca de Substrings (*Substring Search*):** Dada uma String `pat` e uma String `txt`, encontrar uma ocorrência de `pat` em `txt`;
- **Exemplo:** Encontre `ATTGG` na String abaixo, ignorando as quebras de linha:

```
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
CTGACCTATTGCATCTCCCGCCGTGCACAGGGTGTCACGTTGCAAGACCTGCCTGAAACCGAACTGCCCGCTGTTCTGC
AGCCGGTCGCGGAGGCCATGGATGCGATCGCTGCGGCCGATCTTAGCCAGACGAGCGGGTTCGGCCCATTCGGACCGCA
AGGAATCGGTCAATACACTACATGGCGTGATTTCATATGCGCGATTGCTGATCCCCATGTGTATCACTGGCAAACTGTG
```

Introdução

- A String `pat` é conhecida como padrão (*pattern*) e `txt` é conhecida como texto:
  - Busca de Padrões em Texto;
- **Exemplo:** Procurando um padrão de tamanho M em um texto de tamanho N onde, tipicamente, $N \gg M$;
- **Aplicações:**
  - Mecanismo de busca em um editor de texto;
  - Busca na Web por páginas que têm uma certa palavra;
  - Busca de padrões que indicam spam em email;
  - Hackers procurando pela palavra password no seu computador;



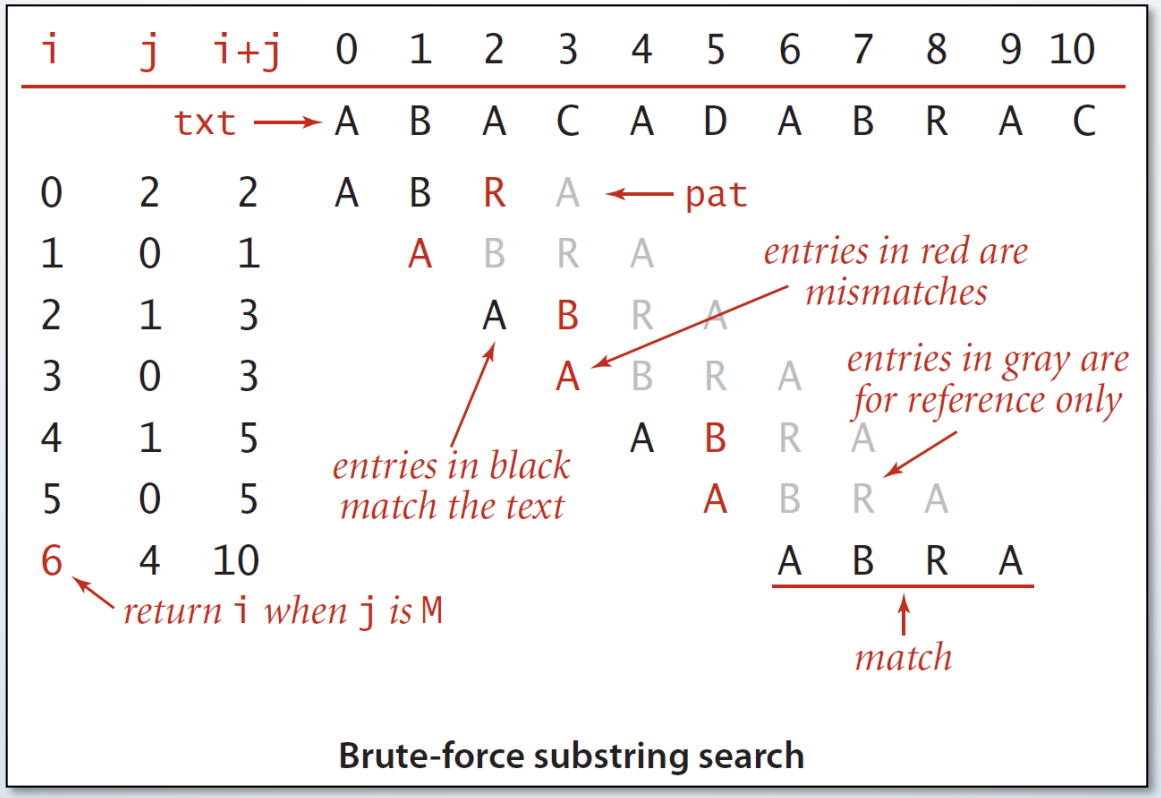
Substring search

Introdução
Regras do Jogo

- **Parâmetros:** M (tamanho do padrão `pat`) e N (tamanho do texto `txt`):
  - Em geral, M é pequeno e N é grande;
- **Notação:** Nas discussões informais, imagine que as Strings são arrays de caracteres. Assim, pode-se escrever:
  - `pat[j]` em lugar de `pat.charAt(j)`;
  - `pat[a..]` em lugar de `pat.substring(a)`;
  - `pat[a..b-1]` em lugar de `pat.substring(a,b)`;
- Pode-se dizer que o padrão é `pat[0..M-1]` e o texto é `txt[0..N-1]`;
- Diz que o padrão `pat` ocorre em `txt`, ou casa com `txt`, a partir de uma posição `i` se `pat` é igual a `txt[i..i+M-1]`. Isso implica que $0 \leq i \leq N - M$;
- **Nosso problema:** encontrar um índice a partir do qual o padrão casa com texto, ou constatar que tal índice não existe.

Força Bruta (*Brute Force*)

- O Algoritmo de Força Bruta compara o padrão `pat` (tamanho M) com todas as substrings do texto `txt` (tamanho N);
- Devolve um índice i tal que `pat` casa com `txt` a partir de i ou devolve N se tal índice não existe;
- **Exemplo:**



Brute-force substring search

Força Bruta (*Brute Force*)
Implementação

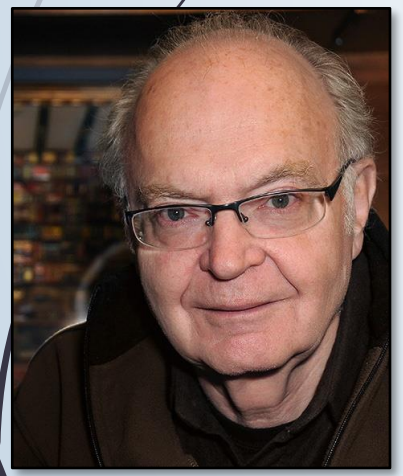
- O Algoritmo de Força Bruta compara o padrão `pat` (tamanho M) com todas as substrings do texto `txt` (tamanho N);
- Devolve um índice i tal que `pat` casa com `txt` a partir de i ou devolve N se tal índice não existe.

i	j	i+j	0	1	2	3	4	5	6	7	8	9	10
			A	B	A	C	A	D	A	B	R	A	C
4	3	7					A	D	A	C	R		
5	0	5						A	D	A	C	R	

```
public static int search(String pat, String txt)
{
   int M = pat.length();
   int N = txt.length();
   for (int i = 0; i <= N - M; i++)
   {
      int j;
      for (j = 0; j < M; j++)
         if (txt.charAt(i+j) != pat.charAt(j))
            break;
      if (j == M) return i;   <-- index in text where pattern starts
   }
   return N;                  <-- not found
}
```

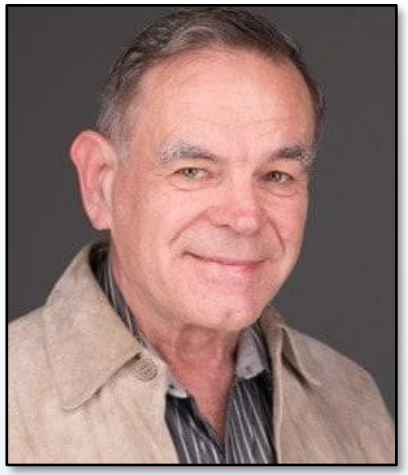
Knuth-Morris-Pratt (KMP)

- Algoritmo descoberto por Knuth, Morris e Pratt para o problema da Busca de Substring (Busca de Padrões)
  - Esse algoritmo introduz o conceito de autômato finito, que é fundamental em várias áreas da Computação.



Donald Knuth

Jim Morris



Vaughan Pratt

SIAM J. COMPUT.
Vol. 6, No. 2, June 1977

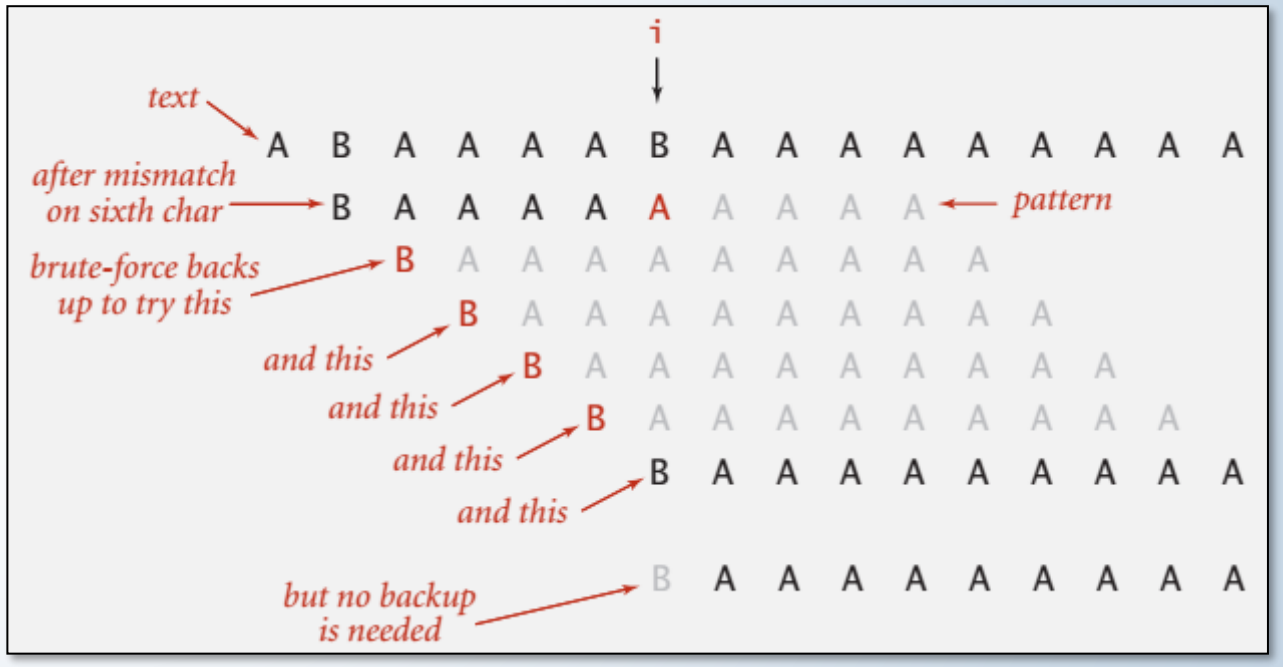
FAST PATTERN MATCHING IN STRINGS*

DONALD E. KNUTH†, JAMES H. MORRIS, JR.‡ AND VAUGHAN R. PRATT¶

Abstract. An algorithm is presented which finds all occurrences of one given string within another, in running time proportional to the sum of the lengths of the strings. The constant of proportionality is low enough to make this algorithm of practical use, and the procedure can also be extended to deal with some more general pattern-matching problems. A theoretical application of the algorithm shows that the set of concatenations of even palindromes, i.e., the language $\{\alpha\alpha^R\}^*$, can be recognized in linear time. Other algorithms which run even faster on the average are also considered.

Knuth-Morris-Pratt (KMP)

- Considere o padrão BAAAAAAAAA e o texto ABAAAABAAAAAAAAA:
  - Depois de detectar um conflito na posição i do texto, é desnecessário retroceder i:
    - Não produziria casamento algum
  - Suponha que combinamos 5 caracteres do padrão, com incompatibilidade no 6º caractere;
  - Sabe-se que os 6 caracteres anteriores no texto são BAAAAB.



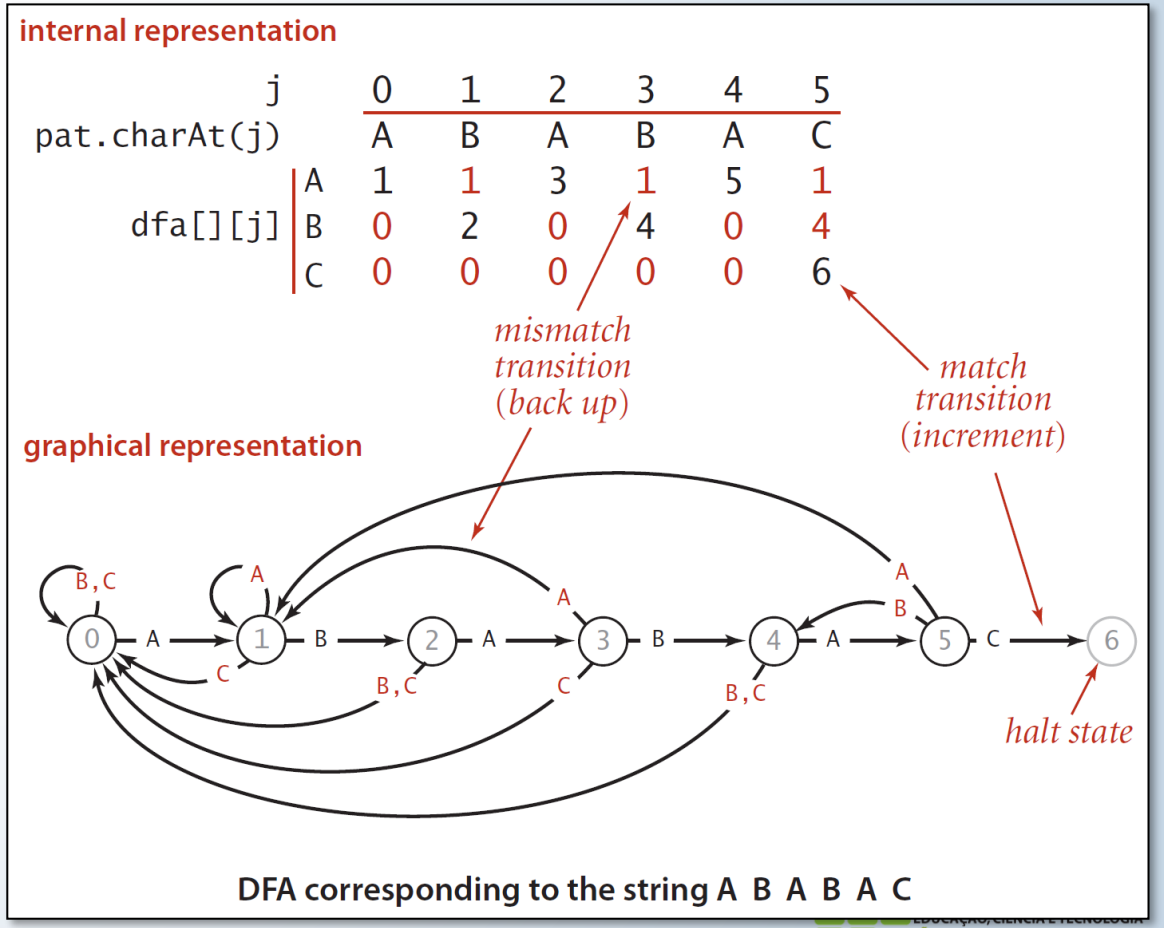
Knuth-Morris-Pratt (KMP)

- **Ideia geral:** Quando encontra-se um conflito entre `txt[i]` e `pat[j]`, não é necessário retroceder i e 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]` e passar a comparar `txt[i+1..]` com `pat[k..]`
- **Exemplo:** padrão AABAAA e texto CAABAABAAAA
  - Depois do conflito entre `pat[5]` e `txt[6]`, não precisa-se retroceder no texto;
  - Pode continuar e comparar `pat[3..]` com `txt[7..];`

```
                    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
```

Knuth-Morris-Pratt (KMP)
Autômato Finito Determinístico (DFA)

- Autômato Finito Determinístico (DFA) representado como uma máquina abstrata para pesquisa de Strings:
  - Número finito de estados;
  - Exatamente uma transição para cada caractere do alfabeto;
  - Aceita se a sequência de transições levar ao estado final (*halt*);
- A tabela `dfa[][]` é uma representação interna do DFA:
  - Os estados correspondem aos índices $0..M-1$ do padrão `pat`;
  - Para cada estado e cada caractere do alfabeto, há uma transição que leva desse estado a um outro;
- Exemplo para alfabeto `A`, `B` e `C` e padrão `ABABAC`:

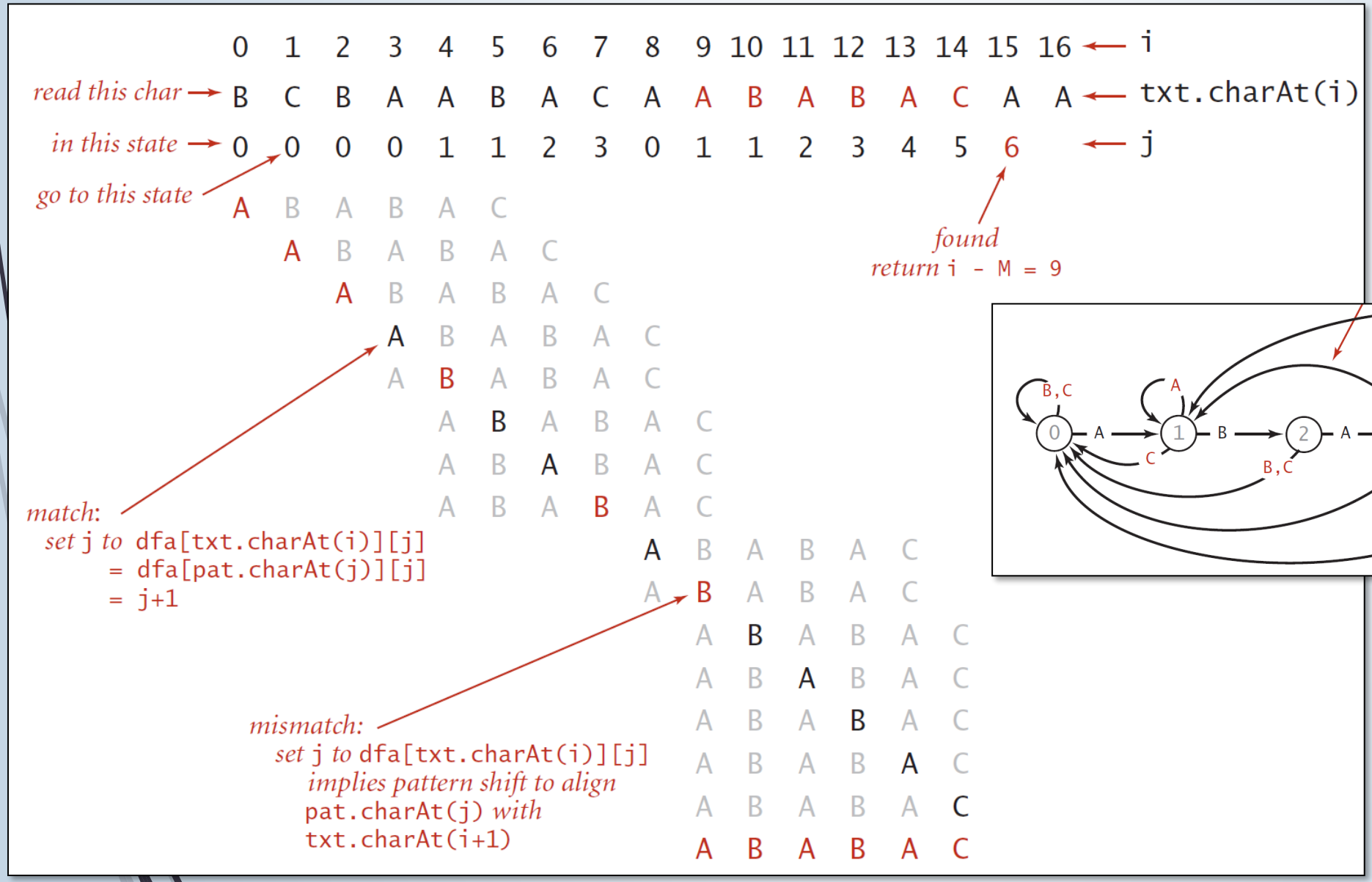


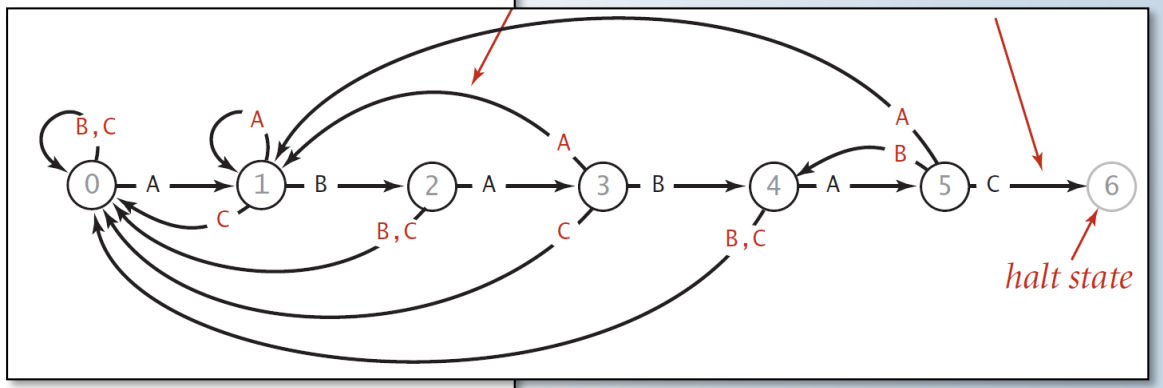
Knuth-Morris-Pratt (KMP)
Autômato Finito Determinístico (DFA)

- O algoritmo KMP simula o funcionamento do DFA:
  - O autômato começa no estado 0 e lê 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, diz-se que o autômato reconheceu ou aceitou o padrão;
  - Se chegar ao fim do texto sem atingir o estado M, sabe-se que o padrão não ocorre no texto;
- 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 menor ou igual a j.

Knuth-Morris-Pratt (KMP)

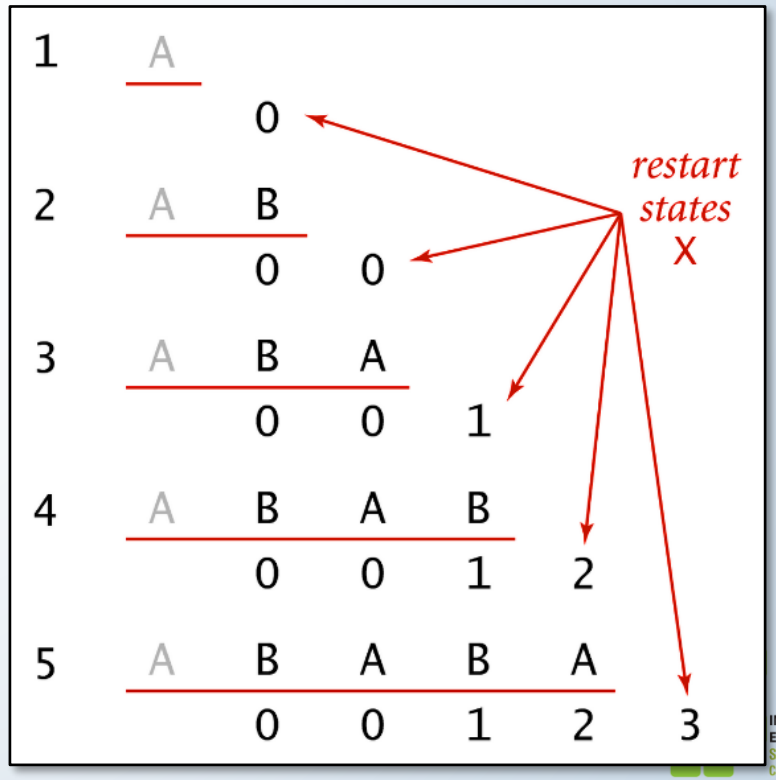
Simulação





Knuth-Morris-Pratt (KMP)
Construção do DFA

- Suponha que `pat[0..j]` é igual a `txt[i-j..i]` exceto no último caractere:
  - Deseja-se saber em que estado o DFA estaria se retrocedêssemos o índice i para $i - j + 1$, ou seja, se o autômato partisse do estado 0 e lesse o texto `txt[i-j+1..i]`;
  - Uma vez calculado esse estado X, pode-se reiniciar o DFA nesse estado como se i tivesse retrocedido;
- Como `txt[i-j+1..i-1]` é igual a `pat[1..j-1]`,pode-se calcular o estado de reinício X sem conhecer o texto;
- **Exemplo:** padrão `ABABAC` e alfabeto `A`, `B` e `C`:

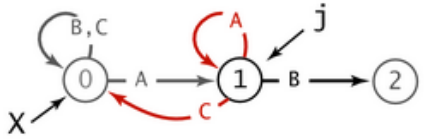


Knuth-Morris-Pratt (KMP)
Construção do DFA

- **Exemplo:** Construção o autômato para o padrão ABABAC e o alfabeto A, B e C:

	j	0
pat.charAt(j)		A
dfa[][j]	A	1
	B	0
	C	0

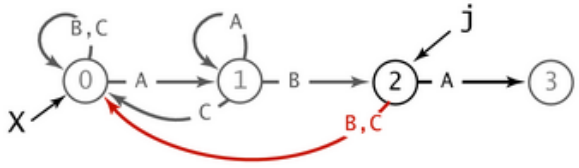
		X ↓	
	j	0	1
pat.charAt(j)		A	B
dfa[][j]	A	1	1
	B	0	2
	C	0	0



copy `dfa[][X]` *to* `dfa[][j]`

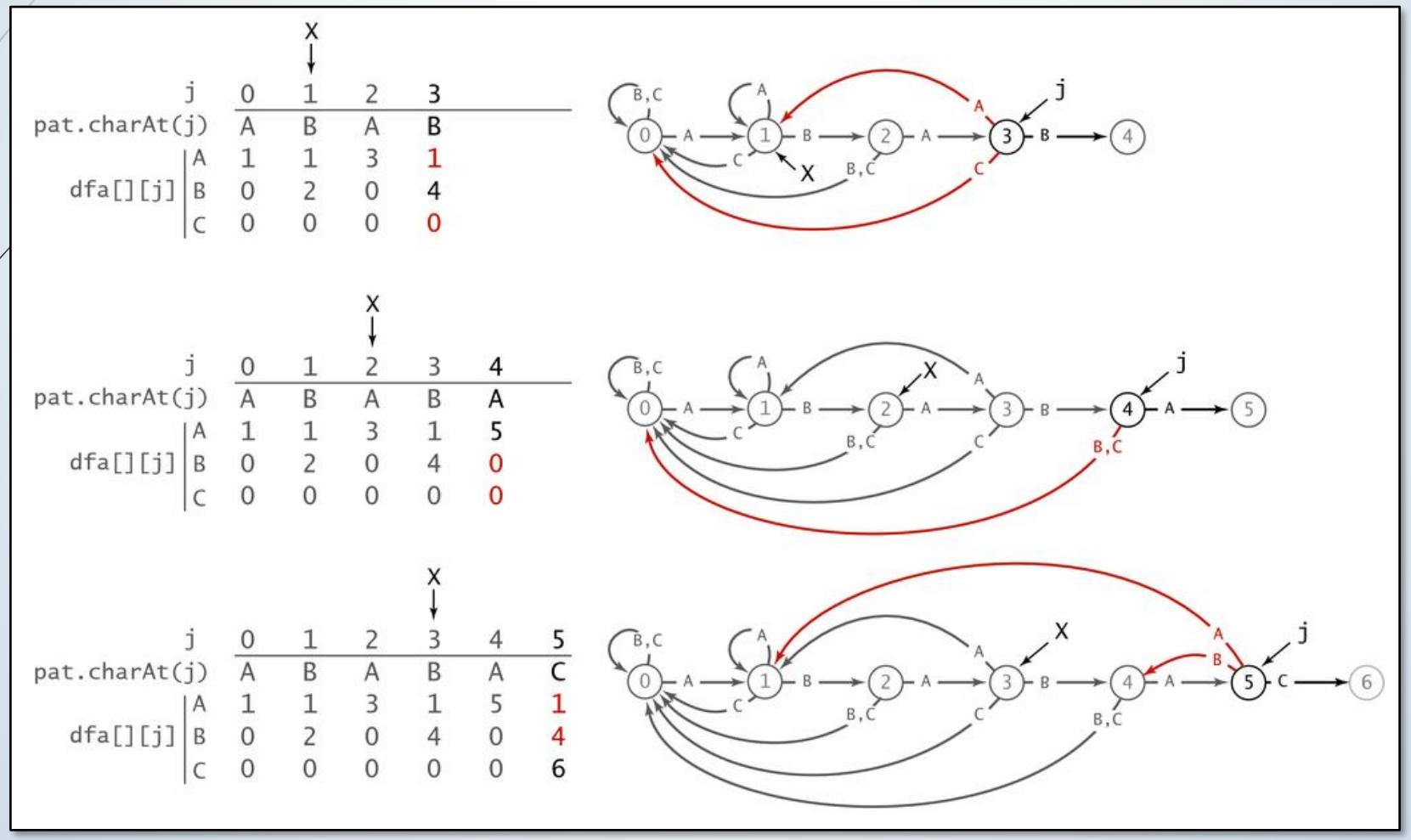
```
dfa[pat.charAt(j)][j] = j+1;
X = dfa[pat.charAt(j)][X];
```

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



Knuth-Morris-Pratt (KMP)
Construção do DFA

- **Exemplo:** Construção o autômato para o padrão **ABABAC** e o alfabeto **A**, **B** e **C**:



Knuth-Morris-Pratt (KMP)
Implementação da Construção do DFA

- Copie `dfa[][X]` para `dfa[][j]` em caso de incompatibilidade;
- Defina `dfa[pat.charAt(j)][j]` para $j + 1$ para o caso de correspondência;
- Atualizar estado X;
- Templo de Execução:
  - $T(n) = M$ (proporcional a RM, mas R é constante, pois é o tamanho do alfabeto)

```
public KMP(String pat)
{  // Build DFA from pattern.
   this.pat = pat;
   int M = pat.length();
   int R = 256;
   dfa = new int[R][M];
   dfa[pat.charAt(0)][0] = 1;
   for (int X = 0, j = 1; j < M; j++)
   {  // Compute dfa[][j].
      for (int c = 0; c < R; c++)
         dfa[c][j] = dfa[c][X];            // Copy mismatch cases.
      dfa[pat.charAt(j)][j] = j+1;         // Set match case.
      X = dfa[pat.charAt(j)][X];           // Update restart state.
   }
}
```

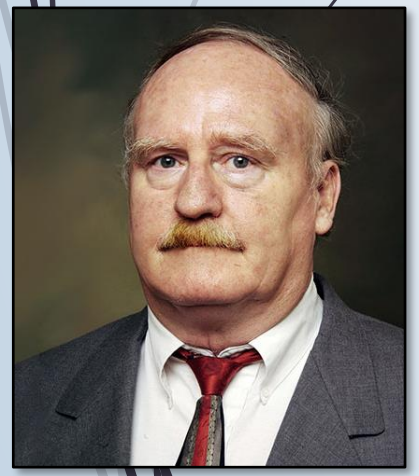
Knuth-Morris-Pratt (KMP)
Implementação da Busca

- Principais diferenças da implementação de Força Bruta:
  - É necessário pré-calcular o DFA a partir do padrão;
  - O ponteiro do texto não diminui, pois não existe retrocesso;
- Tempo de Execução:
  - $T(n) = N$ (no máximo N caracteres serão acessados)

```
public int search(String txt)
{
   int i, j, N = txt.length();
   for (i = 0, j = 0; i < N && j < M; i++)
      j = dfa[txt.charAt(i)][j];
   if (j == M) return i - M;
   else        return N;
}
```

Boyer-Moore (BM)

- **Proposta:**
  - Ler os caracteres no padrão da direita para a esquerda;
  - Pode pular (*skip*) até M caracteres do texto quando encontrar um que não está no padrão.



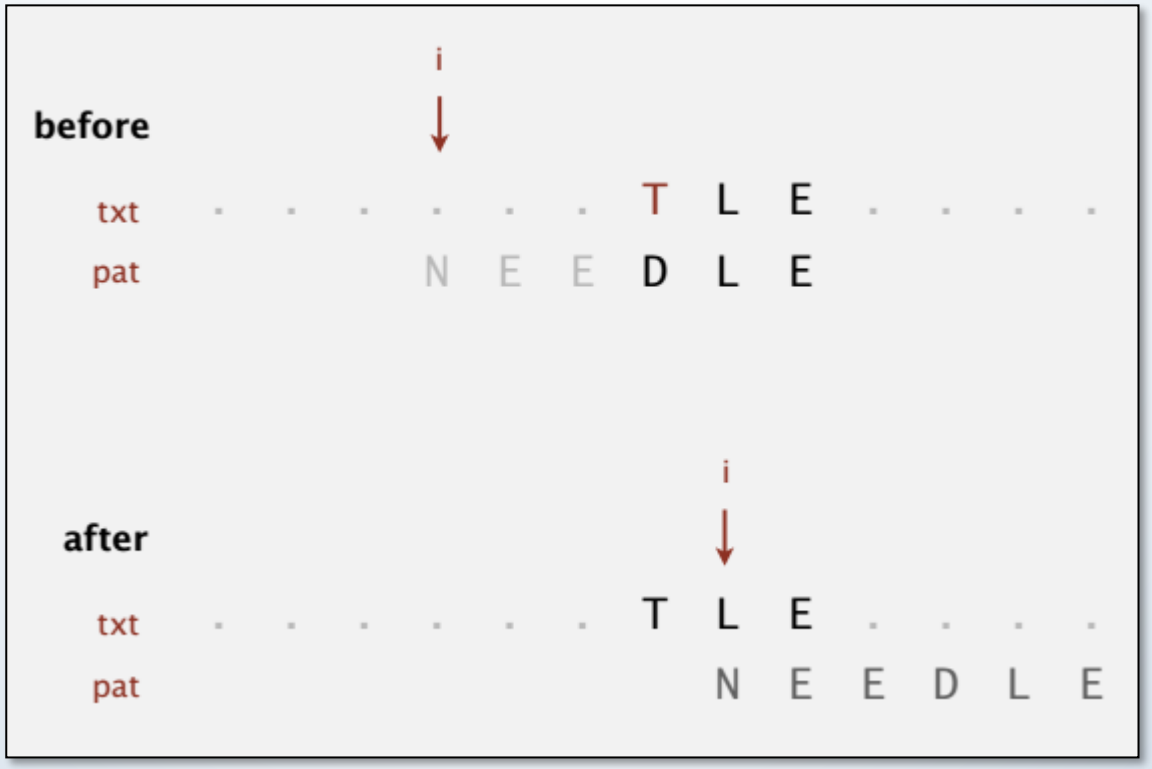
Robert Boyer

J Strother Moore

Boyer-Moore (BM)

Heurística dos Caracteres Incompatíveis

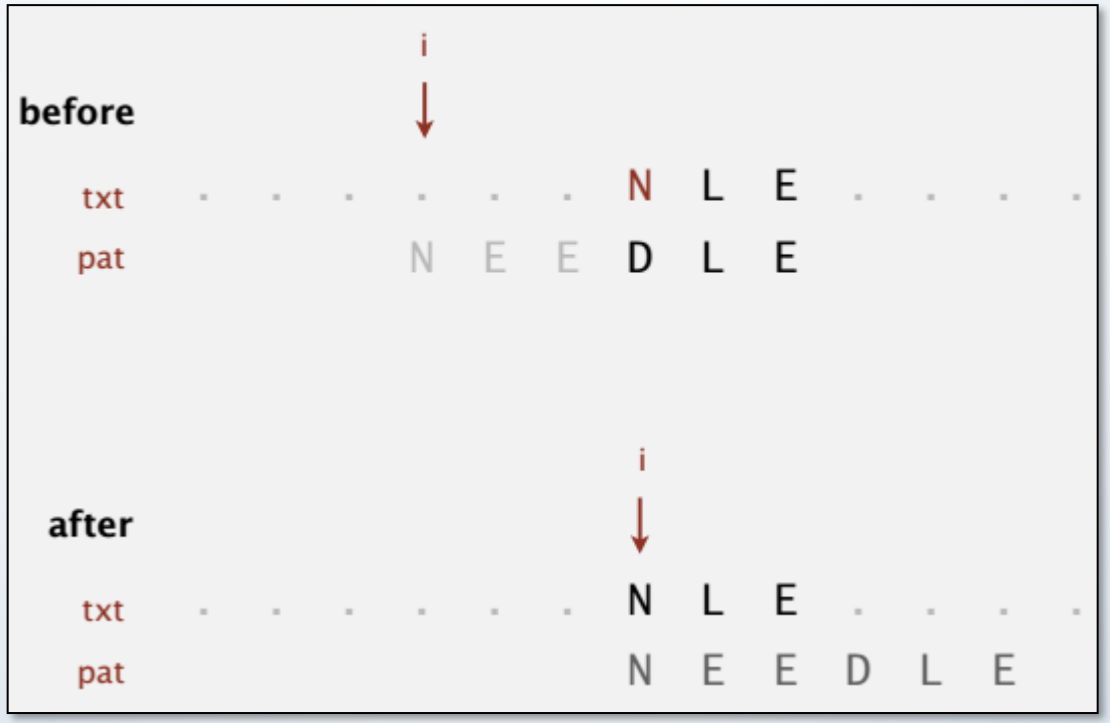
- Quanto caracteres devem ser pulados?
  - **Caso 01:** Caractere incompatível não pertence ao padrão (T não está em pat)
    - Caractere incompatível T fora do padrão: incrementar um caractere além de T



Boyer-Moore (BM)

Heurística dos Caracteres Incompatíveis

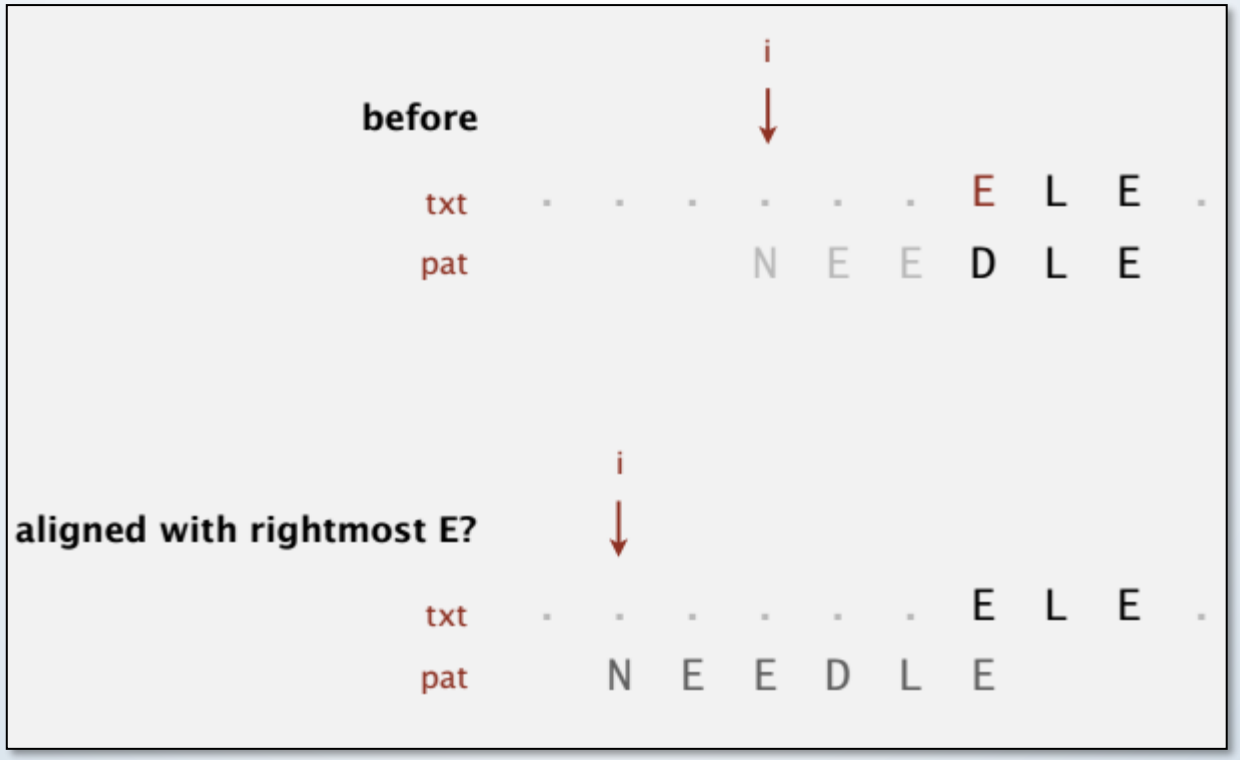
- Quanto caracteres devem ser pulados?
  - **Caso 02a:** Caractere incompatível pertence ao padrão (`N` está em `pat`)
    - Caractere incompatível `N` está do padrão: alinhar o texto `N` com o padrão mais à direita `N`



Boyer-Moore (BM)

Heurística dos Caracteres Incompatíveis

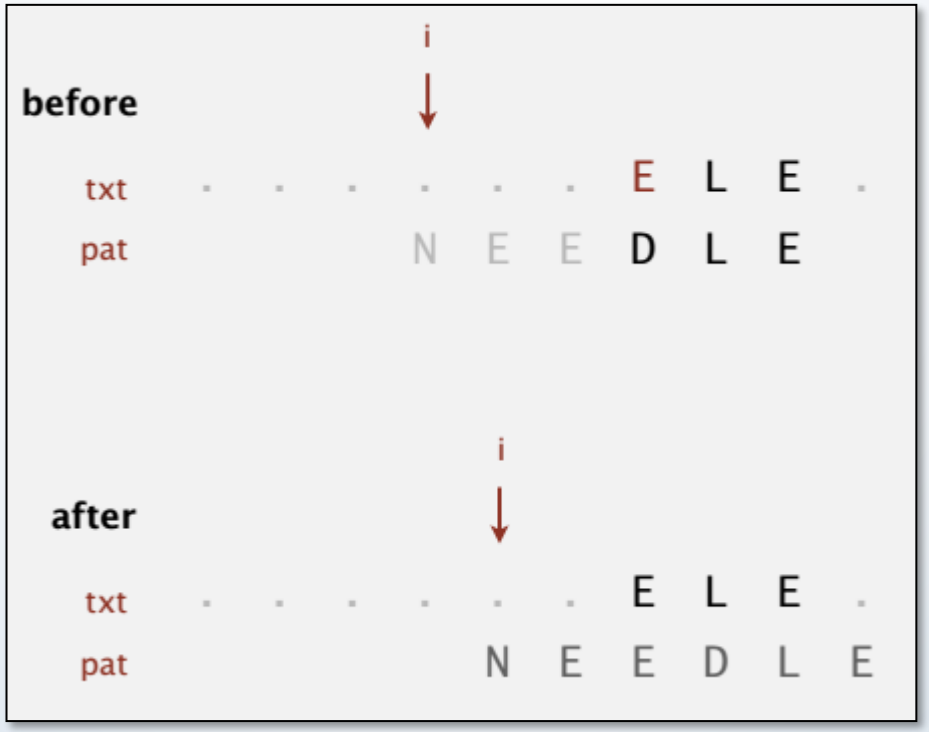
- Quanto caracteres devem ser pulados?
  - **Caso 02b:** Caractere incompatível pertence ao padrão, mas a heurística não ajuda
    - Caractere incompatível E está do padrão: alinhar o texto E com o padrão mais à direita E?



Boyer-Moore (BM)

Heurística dos Caracteres Incompatíveis

- Quanto caracteres devem ser pulados?
  - **Caso 02b:** Caractere incompatível pertence ao padrão, mas a heurística não ajuda
    - Caractere incompatível E está do padrão: Incrementar i em 1



Boyer-Moore (BM)

Heurística dos Caracteres Incompatíveis

- Quanto caracteres devem ser pulados?
  - Índice pré-calculado da ocorrência mais à direita do caractere c no padrão (−1 se o caractere não estiver no padrão)

```
right = new int[R];
for (int c = 0; c < R; c++)
   right[c] = -1;
for (int j = 0; j < M; j++)
   right[pat.charAt(j)] = j;
```

		N	E	E	D	L	E	
c		0	1	2	3	4	5	right[c]
A	-1	-1	-1	-1	-1	-1	-1	-1
B	-1	-1	-1	-1	-1	-1	-1	-1
C	-1	-1	-1	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	(3)	3	3	3
E	-1	-1	(1)	(2)	2	2	(5)	5
...								-1
L	-1	-1	-1	-1	-1	(4)	4	4
M	-1	-1	-1	-1	-1	-1	-1	-1
N	-1	0	0	0	0	0	0	0
...								-1

Boyer-Moore (BM)
Implementação

```
public int search(String txt)
{
   int N = txt.length();
   int M = pat.length();
   int skip;
   for (int i = 0; i <= N-M; i += skip)
   {
      skip = 0;
      for (int j = M-1; j >= 0; j--)
      {
         if (pat.charAt(j) != txt.charAt(i+j))
         {
            skip = Math.max(1, j - right[txt.charAt(i+j)]);
            break;
         }
      }
      if (skip == 0) return i;
   }
   return N;
}
```

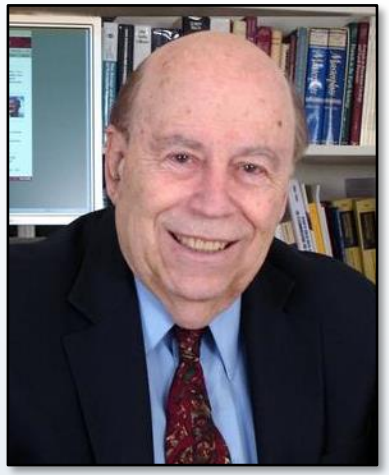
compute skip value

in case other term is nonpositive

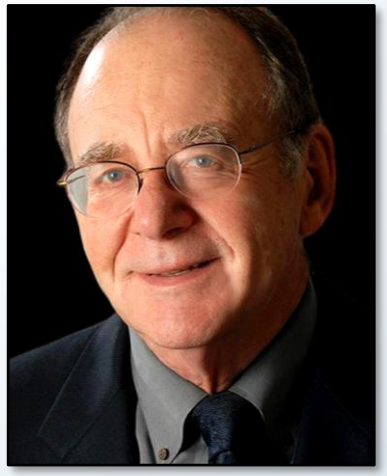
match

Rabin-Karp *Fingerprint Search*

- O algoritmo Rabin-Karp, inventado por Michael O. Rabin e Richard M. Karp, encontra um padrão num texto:
  - O algoritmo também é conhecido como Busca por Impressão Digital (*Fingerprint Search*)
- O algoritmo Rabin-Karp compara o padrão com o texto indiretamente: procura um segmento do texto que tenha o mesmo valor *hash* do padrão;
- O algoritmo usa *hashing* modular: Q



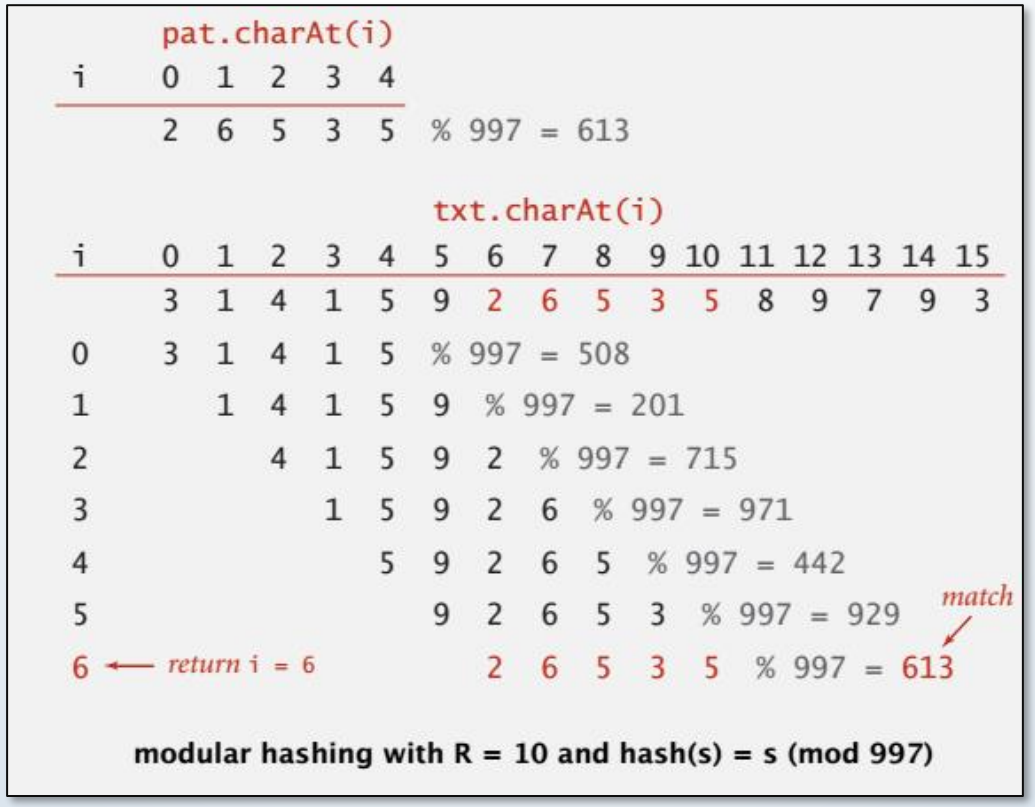
Michael O. Rabin



Richard M. Karp

Rabin-Karp *Fingerprint Search*

- **Proposta:**
  - Computa o *hash* do padrão `pat[0..M-1]`;
  - Para cada *i*, computa o *hash* do texto `txt[i..M+i-1]`;
  - Se o *hash* do padrão for igual ao *hash* do texto, verifica se encontrou o padrão.



modular hashing with R = 10 and hash(s) = s (mod 997)

Rabin-Karp *Fingerprint Search*
Truques Matemáticos

- **Notação:** o padrão tem M caracteres, o texto tem N caracteres, o alfabeto tem R caracteres $(0..R-1)$
- Por exemplo, $R = 10$ (alfabeto de dígitos decimais) ou $R = 256$ (alfabeto ASCII estendido)
- **Dificuldade 1:** Como calcular o *hash* de um padrão `pat` longo (que não cabe em um `int`, por exemplo)?
- Algoritmo de Horner para calcular o *hash* de uma String `s[0..M-1]`:

```
// Compute hash for M-digit key
private long hash(String key, int M)
{
   long h = 0;
   for (int j = 0; j < M; j++)
      h = (h * R + key.charAt(j)) % Q;
   return h;
}
```

	pat.charAt(j)				
i	0	1	2	3	4
	2	6	5	3	5
0	2	% 997 = 2			
1	2	6	% 997 = (2*10 + 6) % 997 = 26		
2	2	6	5	% 997 = (26*10 + 5) % 997 = 265	
3	2	6	5	3	% 997 = (265*10 + 3) % 997 = 659
4	2	6	5	3	5 % 997 = (659*10 + 5) % 997 = 613

R Q

Exemplo para o padrão 26535 com $R = 10$ e $Q = 997$

Rabin-Karp *Fingerprint Search*

Truques Matemáticos

- **Dificuldade 2:** Como calcular eficientemente o *hash* dos segmentos consecutivos do texto `txt`?
- Digamos que t_i é `txt[i]` e x_i é o inteiro representado por $t_i\ t_{i+1}\ \ldots\ t_{i+M-1}$:

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \ldots + t_{i+M-1} R^0$$

$$x_{i+1} = t_{i+1} R^{M-1} + t_{i+2} R^{M-2} + \ldots + t_{i+M} R^0$$

- Pode-se atualizar a função *hash* em tempo constante (*rolling hash*):

$$x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M}$$

current value — subtract leading digit — multiply by radix — add new trailing digit — (can precompute R^{M-1})

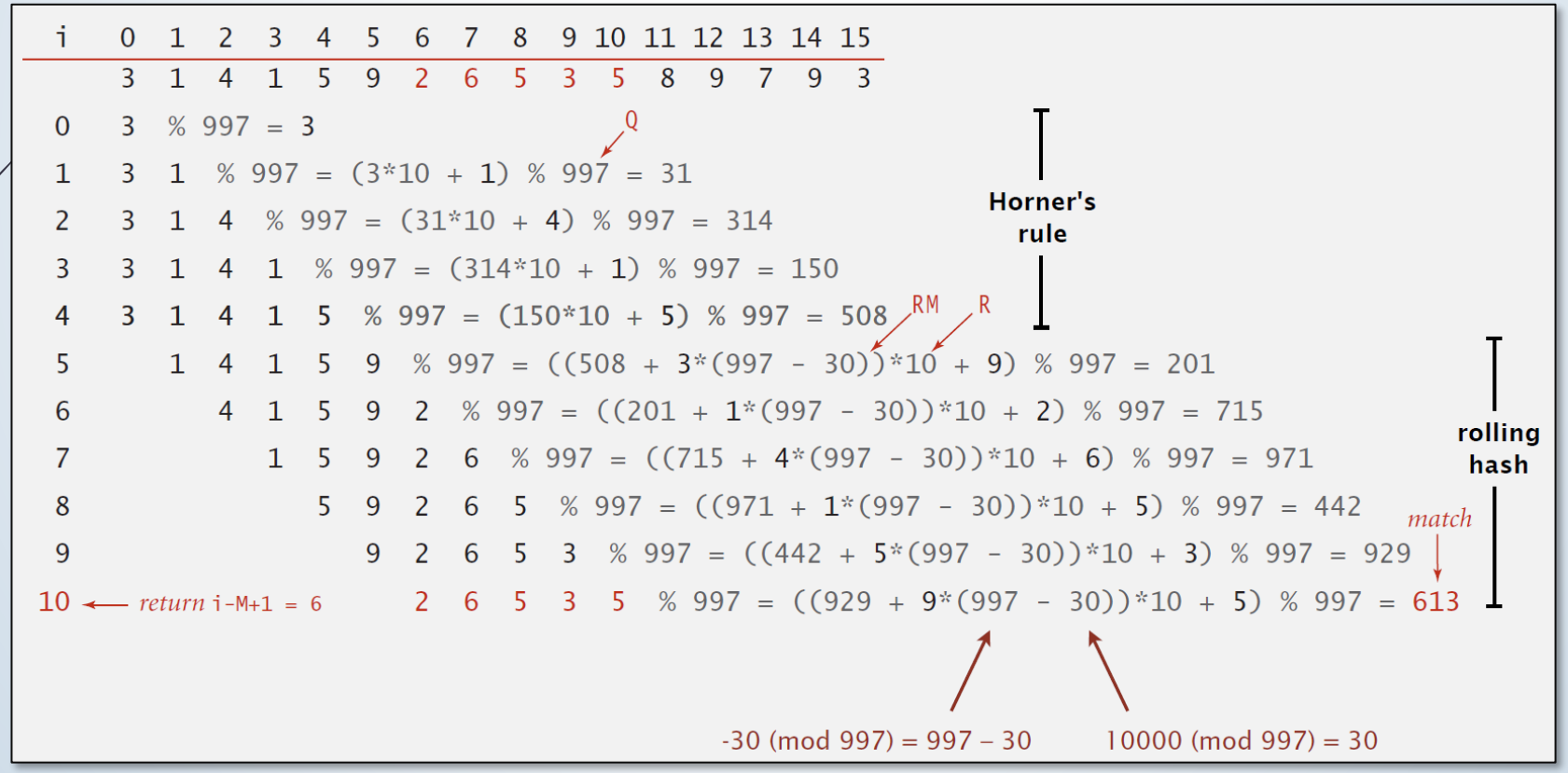
- **Exemplo:**

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*

Rabin-Karp *Fingerprint Search*
Exemplo

- **Primeiras R entradas:** utilizar o Algoritmo de Horner;
- **Entradas restantes:** utilizar o *rolling hash* (e um resto para evitar *overflow*)



Rabin-Karp *Fingerprint Search*
Implementação

```
public class RabinKarp
{
   private long patHash;    // pattern hash value
   private int M;           // pattern length
   private long Q;          // modulus
   private long RM1;        // R^(M-1) % Q

   public RabinKarp(String pat) {
      M = pat.length();
      R = 256;
      Q = longRandomPrime();

      RM1 = 1;
      for (int i = 1; i <= M-1; i++)
         RM1 = (R * RM1) % Q;
      patHash = hash(pat, M);
   }

   private long hash(String key, int M)
   {  /* as before */  }

   public int search(String txt)
   {  /* see next slide */  }
}
```

Um número primo grande para evitar *overflow*

Pré-computar o $R^{M-1} \% Q$

Resumo das Análises da Performance

algorithm	version	operation count guarantee	operation count typical	backup in input?	correct?	extra space
brute force	—	MN	$1.1\,N$	*yes*	*yes*	1
Knuth-Morris-Pratt	*full DFA (Algorithm 5.6)*	$2\,N$	$1.1\,N$	*no*	*yes*	MR
	mismatch transitions only	$3\,N$	$1.1\,N$	*no*	*yes*	M
Boyer-Moore	*full algorithm*	$3\,N$	N / M	*yes*	*yes*	R
	mismatched char heuristic only (Algorithm 5.7)	MN	N / M	*yes*	*yes*	R
Rabin-Karp†	*Monte Carlo (Algorithm 5.8)*	$7\,N$	$7\,N$	*no*	*yes*†	1
	Las Vegas	$7\,N$†	$7\,N$	*yes*	*yes*	1

† *probabilisitic guarantee, with uniform and independent hash function*

Bibliografia

SEDGEWICK, R.; WAYNE, K. **Algorithms**. 4. ed. Boston: Pearson Education, 2011. 955 p.

GOODRICHM M. T.; TAMASSIA, R. **Estruturas de Dados & Algoritmos em Java**. Porto Alegre: Bookman, 2013. 700 p.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos – Teoria e Prática**. 3. ed. São Paulo: GEN LTC, 2012. 1292 p.