Capítulo 1

Processamento de Cadeias de Caracteres

Uma cadeia de caracteres é uma seqüência qualquer de elementos. As cadeias aparecem no processamento de textos em linguagem natural, códigos, dicionários, seqüenciamento de DNA em bilogia computacional, representação de imagens através de *bitmaps*, dentre outros. Este capítulo apresenta algoritmos para duas classes de problemas que envolvem a manipulação de cadeias. Os problemas da primeira classe envolvem algoritmos para casamento de cadeias, para os quais apresentamos algoritmos para pesquisa exata e pesquisa aproximada. Os problemas da segunda classe estão relacionados com algoritmos para compressão de cadeias.

1.1 Casamento de Cadeias

Uma **cadeia** corresponde a uma seqüência de elementos denominados caracteres. Os caracteres são escolhidos de um conjunto denominado **alfabeto**. Por exemplo, em uma cadeia de *bits* o alfabeto é $\{0, 1\}$. A pesquisa em cadeias de caracteres é um componente importante em diversos problemas computacionais tais como edição de texto, recuperação de informação e estudo de seqüências de DNA em biologia computacional. No caso de programas editores de texto o usuário pode estar interessado em buscar todas as ocorrências de um padrão (uma palavra particular) no texto que está sendo editado. Este problema é conhecido como casamento de cadeias de caracteres ou casamento de padrão (do inglês *pattern matching*).

O problema de **casamento de cadeias** ou **casamento de padrão** pode ser formalizado como se segue. O texto é um arranjo $T[1..n]$ de tamanho n e o padrão é um arranjo $P[1..m]$ de tamanho $m \leq n$. Os elementos de P e T são escolhidos de um alfabeto finito Σ de tamanho c. Por exemplo, podemos ter $\Sigma = \{0, 1\}$ ou $\Sigma = \{a, b, \ldots, z\}$. Dados duas cadeias P (padrão) de comprimento $|P| = m$ e T

(texto) de comprimento $|T| = n$, onde $n \gg m$, deseja-se saber as ocorrências de P em T.

Existem diversas categorias de algoritmos para resolver o problema de casamento de cadeias. Sob o ponto de vista algoritmico, considerando os dados de entrada como sendo o texto T e o padrão P, temos as seguintes categorias de algoritmos:

- Padrão e texto não são pré-processados: os algoritmos são do tipo seqüencial, on-line e de tempo-real, pois tanto o padrão quanto o texto não são conhecidos *a priori*. Os algoritmos têm complexidade de tempo $O(mn)$ e de espaço $O(1)$, para o pior caso.
- Padrão pré-processado: os algoritmos são do tipo seqüencial e o padrão é conhecido anteriormente, o que permite o seu pré-processamento. Os algoritmos têm complexidade de tempo $O(n)$ e complexidade de espaço $O(m + c)$, no pior caso. Os algoritmos nesta categoria são usados em diversas aplicações como, por exemplo, em programas para edição de textos.
- Padrão e texto são pré-processados: os algoritmos constroem um índice para permitir uma complexidade de tempo $O(\log n)$ ou menos, mas a complexidade de espaço é $O(n)$. O tempo de pré-processamento do texto para obter o índice pode ser tão grande quanto $O(n)$ ou $O(n \log n)$, mas este tempo é compensado por muitas operações de pesquisa no texto. Neste caso existe um compromisso entre espaço e tempo. Os tipos de índices mais conhecidos são:
  - Arquivos invertidos
  - Árvores *trie* e árvores Patricia
  - Arranjos de sufixos

1.1.1 Casamento Exato

No casamento de padrão o problema básico consiste em obter todas as ocorrências **exatas** do padrão no texto. A Figura 1.1 mostra uma ocorrência exata do padrão `teste` em uma texto exemplo.

```
   teste
os testes testam estes alunos . . .
```

Figura 1.1 *Exemplo de casamento exato*

Os algoritmos para casamento exato de cadeias serão apresentados de acordo com dois enfoques, dependendo da forma como o padrão é pesquisado no texto. O primeiro enfoque consiste em ler os caracteres do texto um a um e, a cada passo, algumas variáveis são atualizadas de forma a identificar uma ocorrência possível. Os algoritmos neste enfoque que serão estudados nas próximas seções são o de força bruta, Knuth-Morris-Pratt e Shift-And. O segundo enfoque consiste em pesquisar o padrão P em uma janela que desliza ao alongo do texto T. Para cada posição desta janela, o algoritmo faz uma pesquisa por um sufixo da janela que casa com um sufixo de P, através de comparações realizadas no sentido da direita para a esquerda. Os algoritmos a serem estudados neste enfoque são o de Boyer-Moore e uma de suas versões simplificadas, o de Boyer-Moore-Horspool.

Algoritmo Força Bruta

O algoritmo **fôrça bruta** é o algoritmo mais simples para casamento de cadeias. A idéia consiste em tentar casar qualquer subcadeia no texto de comprimento m com o padrão. O Programa 1.1 mostra a implementação do algoritmo fôrça bruta.

```
procedure ForcaBruta (var T: TipoTexto; var n: integer;
                      var P: TipoPadrao; var m: integer);
{-- Pesquisa P[1..m] em T[1..n] --}
var i, j, k: Integer;
begin
  for i := 1 to n - m + 1 do
    begin
    k := i;  j:= 1;
    while T[k] = P[j] do
      begin j := j + 1;  k := k + 1; end;
    if j>m then writeln ('Casamento na posicao', i:3);
    end;
end;
```

Programa 1.1 *Algoritmo fôrça bruta*

Análise

O pior caso do algoritmo fôrça bruta é

$$C_n = m \times n,$$

situação que ocorre, por exemplo, quando $P =$aab e $T =$aaaaaaaaaa.

O caso esperado é dado por Baeza-Yates (1992) como sendo

$$\overline{C_n} = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1) + O(1),$$

que é muito melhor do que o pior caso. Em um experimento realizado por Baeza-Yates (1992), para um texto randômico e um alfabeto de tamanho $c = 4$, o número esperado de comparações por caractere do texto é aproximadamente igual a 1,3.

Uso de Autômato

Um autômato é um modelo de computação muito simples. Um **autômato finito** é definido por uma tupla $(Q, I, F, \Sigma, \mathcal{T})$, onde: Q é um conjunto finito de estados, entre os quais existe um estado inicial $I \in Q$, e alguns são estados finais ou estados de término $F \subseteq Q$. As transições entre estados são rotuladas por elementos de $\Sigma \cup \{\epsilon\}$, onde Σ é o alfabeto finito de entrada e ϵ é a transição vazia. As transições são formalmente definidas por uma função de transição $\mathcal{T}$, a qual associa a cada estado $q \in Q$ um conjunto $\{q_1, q_2, \ldots, q_k\}$ de estados de Q para cada $\alpha \in \Sigma \cup \{\epsilon\}$.

Na prática existem dois tipos de autômatos, dependendo da forma da função de transição $\mathcal{T}$. Se $\mathcal{T}$ é tal que existe um estado q associado a um dado caractere α para mais de um estado, digamos $\mathcal{T}(q, \alpha) = \{q_1, q_2, \ldots, q_k\}$, $k > 1$, ou existe alguma transição rotulada por ϵ, então o autômato é chamado de **autômato finito não-determinístico**. Neste caso, a função de transição $\mathcal{T}$ é definida pelo conjunto de triplas $\Delta = \{(q, \alpha, q')$, onde $q \in Q$, $\alpha \in \Sigma \cup \{\epsilon\}$, e $q' \in \mathcal{T}(q, \alpha)$. Senão, o autômato é chamado de **autômato finito determinístico**, e a função de transição $\mathcal{T}$ é definida pela função $\delta = Q \times \Sigma \cup \epsilon \rightarrow Q$. Neste caso, se $\mathcal{T}(q, \alpha) = \{q'\}$, então $\delta(q, \alpha) = q'$.

A Figura 1.2 mostra um exemplo dos dois tipos de autômatos. O autômato da Figura 1.2(a) é um autômato finito não-determinístico, uma vez que a partir do estado 0, através do caractere de transição a é possível atingir os estados 2 e 3. O autômato da Figura 1.2(b) é um autômato finito determinístico, uma vez que para cada caractere de transição todos os estados levam a um único estado.

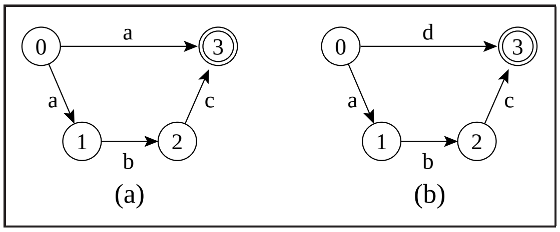

Figura 1.2 Dois autômatos, onde o estado 0 é inicial e o estado 3 com dois cículos concêntricos é final. (a) Autômato finito não-determinístico. (b) Autômato finito determinístico

Uma cadeia é **reconhecida** por $(Q, I, F, \Sigma, \Delta)$ ou $(Q, I, F, \Sigma, \delta)$ se qualquer um dos autômatos rotula um caminho que vai de um estado inicial até um estado final. A **linguagem reconhecida** por um autômato é o conjunto de cadeias que o autômato é capaz de reconher. Por exemplo, a linguagem reconhecida pelo autômato da Figura 1.2(a) é o conjunto de cadeias $\{a\}$ e $\{aba\}$ no estado 3.

Em autômatos não-determinísticos transições podem ser rotuladas com uma cadeia vazia ϵ, e são chamadas de **transições-ϵ** ou **transições vazias**. O significado de uma transição vazia é que não há necessidade de se ler um caractere para caminhar através da transição, isto é, se estamos no estado origem da transição-ϵ simplesmente saltamos para o estado destino. Estas transições muitas vezes simplificam a construção do autômato, mas sempre existe um autômato equivalente que reconhece a mesma linguagem sem transições-ϵ.

Em um autômato, seja ele determinístico ou não-determinístico, se uma cadeia x rotula um caminho de I até um estado q então o estado q é considerado ativo depois de ler x. Um autômato finito determinístico tem no máximo um estado ativo em um determinado instante, enquanto um autômato finito não-determinístico pode ter vários. O algoritmo para casamento aproximado de cadeias a ser estudado na Seção 1.2 é baseado em autômatos finitos não-determinísticos.

Os dois autômatos da Figura 1.2 são **acíclicos** porque as transições não formam ciclos. Entretanto, os **autômatos finitos cíclicos**, sejam determinísticos ou não-determinísticos, são úteis para **casamento de expressões regulares**. A linguagem reconhecida por um autômato cíclico pode ser infinita. O autômato da Figura 1.3 reconhece `ba`, mas também reconhece `bba`, `bbba`, `bbbba`, e assim por diante.

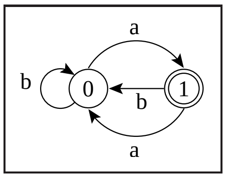

Figura 1.3 *Autômato finito determinístico cíclico*

A Figura 1.4 mostra o autômato que reconhece o padrão P ={`aabc`}. A pesquisa de P sobre um texto T com alfabeto Σ ={`a, b, c`} pode ser vista como a simulação do autômato na pesquisa de P sobre T. No início da computação o estado inicial ativa o estado 1. Para cada caractere lido do texto a aresta correspondente é seguida, ativando o estado destino. Se o estado 4 estiver ativo e um caractere `c` é lido o estado final se torna ativo, resultando em um casamento de `aabc` com o texto. Como cada caractere do texto é lido uma vez o algoritmo tem complexidade de tempo $O(n)$, e complexidade de espaço $m + 2$ para vértices e $|\Sigma| \times m$ para arestas.

Algoritmo Knuth-Morris-Pratt

O algoritmo Knuth-Morris-Pratt (KMP) é o primeiro algoritmo cujo pior caso tem complexidade de tempo linear no tamanho do texto ($O(n)$). O KMP, um dos algoritmos mais famosos para resolver o problema de casamento de cadeias, é devido a Knuth, Morris e Pratt (1977). Como este algoritmo tem uma implemen-

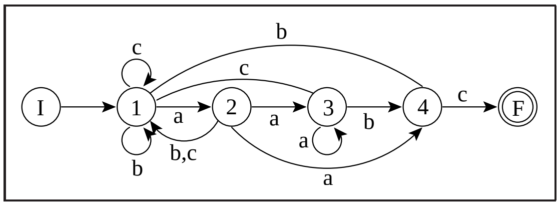

Figura 1.4 *Autômato finito determinístico cíclico para reconhecer* aabc

tação complicada e na prática perde em eficiência para os outros dois algoritmos que estudaremos nas seções seguintes, o Shift-And e o Boyer-Moore-Horspool, não vamos apresentar a sua implementaão.

Entretanto, sob o ponto de vista histórico, existe um fato interessante sobre como surgiu o algoritmo KMP. Até 1971, o limite inferior conhecido para busca exata de padrões era $O(mn)$. Em 1971, Cook provou que qualquer problema que puder ser resolvido por um autômato determinístico de dois caminhos com memória de pilha (do inglês *Two-way Deterministic Pushdown Store Automaton* (2DPDA)) pode ser resolvido em tempo linear por uma máquina RAM. O 2DPDA é constituído de uma fita apenas para leitura, uma pilha de dados (memória temporária), e um controle de estado que permite mover a fita para esquerda ou direita, empilhar ou desempilhar símbolos, e mudar de estado. A Figura 1.5 mostra o casamento de cadeias no 2DPDA.

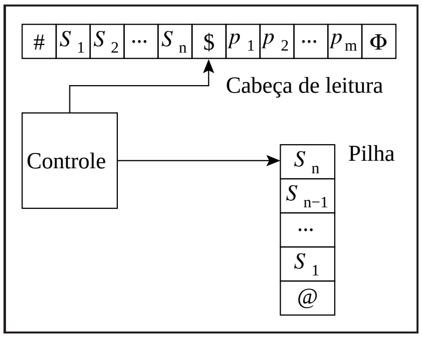

Figura 1.5 *Casamento de cadeias no 2DPDA*

No autômato da Figura 1.5 a entrada é constituída da cadeia

$$\#c_1c_2\cdots c_n\$p_1p_2\cdots p_n\phi.$$

A partir de # todos os caracteres são empilhados até encontrar o caractere $. A leitura cotinua até encontrar o caractere ϕ. A seguir a leitura é realizada no sentido contrário, iniciando por p_n, o qual deve ser comparado com o último caractere empilhado, no caso c_n. Esta operação é repetida para os caracteres seguintes, e se o caractere $ for atingido então as duas cadeias são iguais.

Knuth, Morris e Pratt trabalharam na simulação linear que Cook obteve para o 2DPDA para obterem a primeira versão do algoritmo KMP. O algoritmo obtém um mecanismo para computar o sufixo mais longo no texto que é também prefixo de P. Quando o comprimento do sufixo no texto é igual a $|P|$ ocorre um casamento. O pré-processamento de P permite que nenhum caractere seja reexaminado e o apontador para o texto nunca é decrementado. Na realidade o pré-processamento de P pode ser visto como a construção econômica de um autômato determinístico que depois é usado para pesquisar pelo padrão no texto.

Algoritmo Shift-And

O algoritmo Shift-And foi proposto por Baeza-Yates e Gonnet (1989). Ele é aproximadamente duas vezes mais rápido e muito mais simples do que o algoritmo KMP. Além disso, na Seção 1.2 veremos como estender o algoritmo Shift-And para permitir casamento aproximado de cadeias de caracteres.

O algoritmo usa o conceito de **paralelismo de bit** (do inglês *bit parallelism*), uma técnica que tira proveito do paralelismo intrínseco das operações sobre *bits* dentro de uma palavra de computador. Neste caso, é possível empacotar muitos valores em uma única palavra e atualizar todos eles em uma única operação. Pelo fato de tirar proveito do paralelismo de *bit*, o número de operações que um algoritmo realiza pode ser reduzido por um fator de até w, onde w é o número de *bits* da palavra do computador. Considerando que nas arquiteturas atuais w é 32 ou 64, o ganho na prática pode ser muito grande.

Antes de continuar a descrever o algoritmo Shift-And vamos utilizar a notação usada em Navarro e Raffinot (2002) para descrever as operações usando paralelismo de *bit*. Para denotar **repetição de bit** é usado exponenciação: $01^3 = 0111$. Uma seqüência de *bits* $b_1 \dots b_c$ é chamada de **máscara de bits** de comprimento c, a qual é armazenada em alguma posição de uma palavra w do computador. Para as operações sobre os *bits* da palavra do computador, "|" é a operação *or*, "&" é a operação *and*, "$\sim$" complementa todos os *bits*, e "$>>$" move os *bits* para a direita e entra com zeros à esquerda (por exemplo, $b_1, b_2, \dots, b_{c-1}, b_c >> 2 = 00b_3, \dots, b_{c-2}$).

O algoritmo mantém um conjunto de todos os prefixos de P que casam com o texto já lido e utiliza o paralelismo de *bit* para atualizar o conjunto a cada caractere lido do texto. Este conjunto é representado por uma máscara de *bits* $R = (b_1, b_2, \dots, b_m)$. Assim como o algoritmo KMP, o algoritmo Shift-And pode ser visto como a simulação de um autômato que pesquisa pelo padrão no texto. A diferença entre o KMP e o Shift-And é que o KMP usa um autômato determinístico enquanto o Shift-And usa um autômato não-determinístico para simular o paralelismo de *bit*.

A Figura 1.6 mostra um autômato não-determinístico capaz de reconhecer todos os prefixos do padrão P =$\{$`teste`$\}$. O *self-loop* no vértice 0 significa que o estado permanece ativo durante todo o processamento, permitindo que um casamento possa se iniciar na posição corrente do texto. Neste caso, mais de um estado pode estar ativo em um determinado instante.

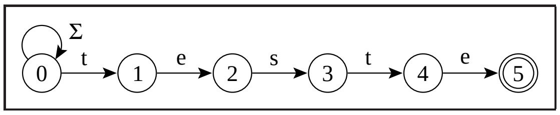

Figura 1.6 *Autômato não-determinístico que reconhece todos os prefixos de P ={`teste`}*

O funcionamento do algoritmo Shift-And é explicado a seguir. O valor 1 é colocado na j-ésima posição de $R = (b_1, b_2, \dots, b_m)$ se e somente se $p_1 \dots p_j$ é um sufixo de $t_1 \dots t_i$, onde i corresponde à posição corrente no texto. A j-ésima posição de R é dita estar *ativa*. Um casamento é relatado sempre que b_m fica ativo.

Na leitura do próximo caractere t_{i+1}, o novo valor do conjunto R é calculado, o qual chamaremos de R'. A posição $j+1$ no conjunto R' ficará ativa se e somente se a posição j estava ativa em R. Em outras palavras, $p_1 \dots p_j$ era um sufixo de $t_1 \dots t_i$ e t_{i+1} casa com p_{j+1}. Com o uso de paralelismo de *bit* é possível computar o novo conjunto com custo $O(1)$ em uma linguagem de programação que realize com eficiência as operações *and*, *or*, deslocamento à direita e complemento, como no caso da linguagem C. Além disso, se o tamanho do padrão m for menor do que a palavra w do computador então o conjunto R pode ser implementado em um arranjo de *bits* que cabe em um registrador do computador.

O primeiro passo do algoritmo é a construção de uma tabela M para armazenar uma máscara de *bits* $b_1 \dots, b_m$ para cada caractere. A Tabela 1.1 apresenta as máscaras de *bits* para os caracteres presentes em P ={`teste`}. A máscara em M[`t`] é 10010, pois o caractere `t` aparece nas posições 1 e 4 de P.

	1	2	3	4	5
M[t]	1	0	0	1	0
M[e]	0	1	0	0	1
M[s]	0	0	1	0	0

Tabela 1.1 *Máscara relativa a P ={`teste`}*

Na fase de pesquisa de P em T o valor do conjunto é inicializado como $R = 0^m$ (0^m significa 0 repetido m vezes). Para cada novo caractere t_{i+1} lido do texto o valor do conjunto R' é atualizado de acordo com a seguinte fórmula:

$$R' = ((R >> 1) \mid 10^{m-1}) \ \& \ M[T[i]]$$

Intuitivamente, a operação “>>” desloca todas as posições para a direita no passo $i+1$ para marcar quais posições de P eram sufixos no passo i. A cadeia vazia ϵ também é marcada como um sufixo por meio da operação *or* entre o conjunto obtido após a operação ">>" e 10^{m-1}. Essa operação permite que um casamento possa iniciar na posição corrente do texto, o que corresponde ao *self-loop* no início do autômato da Figura 1.6. Do conjunto obtido até o momento são mantidas apenas as posições que t_{i+1} casa com p_{j+1}, o que é obtido por meio da operação *and* desse conjunto de posições com o conjunto $M[t_{i+1}]$ de posições de t_{i+1} em P.

A Tabela 1.2 mostra o funcionamento do algoritmo Shift-And para pesquisar o padrão P ={`teste`} no texto T ={`os testes ...`}, parte inicial do exemplo da Figura 1.1.

Texto	$(R >> 1) \| 10^{m-1}$					R'				
`o`	1	0	0	0	0	0	0	0	0	0
`s`	1	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0
`t`	1	0	0	0	0	1	0	0	0	0
`e`	1	1	0	0	0	0	1	0	0	0
`s`	1	0	1	0	0	0	0	1	0	0
`t`	1	0	0	1	0	1	0	0	1	0
`e`	1	1	0	0	1	0	1	0	0	*1*
`s`	1	0	1	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0

Tabela 1.2 *Exemplo de funcionamento do algoritmo Shif-And. O 1 na última coluna significa que o estado final está ativo, indicando casamento de P ={`teste`} no texto*

O Programa 1.2 mostra um primeiro refinamento do algoritmo Shift-And.

Shift-And $(P = p_1p_2 \ldots p_m,\ T = t_1t_2 \ldots t_n)$;
  {–– Pré–processamento ––}
  for $c \in \Sigma$ **do** $M[c] := 0^m$;
  for j := 1 **to** m **do** $M[p_j] := M[p_j] \mid 0^{j-1}10^{m-j}$;
  {–– Pesquisa ––}
  $R := 0^m$;
  for i := 1 **to** n **do**
    $R = ((R >> 1) \mid 10^{m-1})\ \&\ M[T[i]]$;
    if $R\ \&\ 0^{m-1}1 \neq 0^m$ **then** 'Casamento na posicao $i - m + 1$';

Programa 1.2 *Primeiro refinamento do algoritmo Shift-And*

Infelizmente, as operações *and*, *or*, deslocamento à direita e complemento a serem realizadas sobre o conjunto R, não podem ser realizadas com eficiência na linguagem Pascal, o que compromete o conceito de paralelismo de *bit*. Por essa razão, não apresentamos a implementação do algoritmo Shift-And em Pascal, mas apenas a implementação em C. O Programa 1.3 mostra a implementação do algoritmo Shift-And na linguagem C.

Programa 1.3 *Implementação do algoritmo Shift-And para casamento exato de cadeias na linguagem C*

Análise

O custo do algoritmo Shift-And é $O(n)$, desde que as operações na Eq.(**??**) possam ser realizadas em $O(1)$ e o padrão caiba em umas poucas palavras do computador.

Algoritmo Boyer-Moore-Horspool

Assim como o algoritmo KMP apresentado em 1977, outro algoritmo clássico foi publicado por Boyer e Moore (1977), ficando conhecido como Boyer-Moore (BM). A idéia é pesquisar no padrão no sentido da direita para a esquerda, o que torna o algoritmo muito rápido. Em 1980, Horspool (1980) apresentou uma simplificação importante no algoritmo original, tão eficiente quanto o algoritmo original, ficando conhecida como Boyer-Moore-Horspool (BMH). Considerando sua extrema simplicidade de implementação, bem como sua comprovada eficiência, o algoritmo BMH deve ser o escolhido em aplicações de uso geral que necessitam realizar casamento exato de cadeias.

O enfoque dos algoritmos BM e BMH consiste em pesquisar o padrão P em uma janela que desliza ao alongo do texto T. Para cada posição desta janela, o algoritmo faz uma pesquisa por um sufixo da janela que casa com um sufixo de P, através de comparações realizadas no sentido da direita para a esquerda. Se não ocorrer uma desigualdade, então uma ocorrência de P em T ocorreu. Senão, o algoritmo calcula um deslocamento que o padrão deve ser deslizado para a direita antes que uma nova tentativa de casamento se inicie.

O algoritmo BM original propõe duas heurísticas para calcular o deslocamento:

- Heurística ocorrência (do inglês *ocurrence*): alinha a posição no texto que causou a colisão com o primeiro caractere no padrão que casa com ele;
- Heurística casamento (do inglês *match*): ao mover o padrão para a direita, faça-o casar com o pedaço do texto anteriormente casado.

A Figura 1.7 mostra o funcionamento da heurística ocorrência para o padrão P ={`cacbac`} no texto T ={`aabcaccacbac`}. A partir da posição 6, da direita para a esquerda, existe uma colisão na posição 4 de T, entre o caractere `b` do padrão e o caractere `c` do texto. Conseqüentemente, o padrão deve ser deslocado para a direita até o primeiro caractere no padrão que casa com `c`. O processo é repetido e o caractere no texto que causou a colisão é o caractere `c` na posição 6. Assim, o padrão deve ser novamente deslocado para a direita até o primeiro caractere no padrão que casa com `c`. O processo é repetido até encontrar um casamento a partir da posição 7 de T.

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

Figura 1.7 *Funcionamento da heurística ocorrência*

A Figura 1.8 mostra o funcionamento da heurística casamento para o mesmo padrão P ={`cacbac`} no texto T ={`aabcaccacbac`}. Novamente, a partir da posição 6, da direita para a esquerda, existe uma colisão na posição 4 de T, entre o caractere `b` do padrão e o caractere `c` do texto. Neste caso, o padrão deve ser deslocado para a direita até casar com o pedaço do texto anteriormente casado, no caso `ac`, deslocando o padrão 3 posições à direita. O processo é repetido mais uma vez e o casamento entre P e T ocorre.

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

Figura 1.8 *Funcionamento da heurística casamento*

O algoritmo BM decide qual das duas heurísticas deve usar escolhendo a que provoca o maior deslocamento do padrão. Entretanto, esta escolha implica em realizar uma comparação entre dois inteiros para cada caractere lido do texto, penalizando o desmpenho do algoritmo com relação a tempo de processamento. A partir dessa observação várias propostas de simplificação ocorreram ao longo dos anos, sendo as que produzem os melhores resultados são as que consideram apenas a heurística ocorrência. A simplificaç que estuadremos a seguir segue essa linha.

A simplificação mais importante é devida a Horspool (1980), conhecida como Boyer-Moore-Horspool)BMH), o qual executa mais rápido do que o algoritmo BM original. Horspool observou que qualquer caractere já lido do texto a partir do último deslocamento pode ser usado para endereçar a tabela de deslocamentos. Baseado nesse fato Horspool propôs endereçar a tabela com o caractere no texto correspondente ao último caractere do padrão.

Para pré-computar o padrão o valor inicial de todas as entradas na tabela de deslocamentos é feito igual a m. A seguir, apenas para os $m-1$ primeiros caracteres do padrão são usados para obter os outros valores da tabela. Formalmente

$$d[x] = min\{j talque j = m | (1 \leq j < m \& P[m-j] = x)\}.$$

Para o padrão $P = \{\texttt{teste}\}$, os valores de d são $d[\texttt{t}] = 1$, $d[\texttt{e}] = 3$, $d[\texttt{s}] = 2$, e todos os outros valores são iguais ao valor de $|P|$, nesse caso $m = 5$.

Uma implementação simples e eficiente para o algoritmo BMH pode ser vista no Programa 1.4. O pré-processamento do padrão para obter a tabela de deslocamentos d ocorre nas duas primeiras linhas do código. A fase de pesquisa é constituída por um anel em que i varia de m até n, com incrementos d[ord(T[i])], o que equivale ao endereço na tabela d do caractere que está na i-ésima posição no texto, a qual corresponde à posição do último caractere de P.

```
procedure BMH (var T: TipoTexto; var n: integer;
               var P: TipoPadrao; var m: integer);
{-- Pesquisa P[1..m] em T[1..n] --}
var i, j, k: Integer;
    d: array[0..MaxTamAlfabeto] of integer;
begin
  {-- Pre-processamento do padrao --}
  for j := 0 to MaxTamAlfabeto do d[j] := m;
  for j := 1 to m-1 do d[ord(P[j])] := m-j;
  i := m;
  while i <= n do {-- Pesquisa --}
    begin
    k := i;  j := m;
    while (T[k] = P[j]) and (j>0) do
      begin k := k-1; j := j-1; end;
    if j = 0 then writeln(' Casamento na posicao: ',k+1:3);
    i := i + d[ord(T[i])];
    end;
end;
```

Programa 1.4 *Algoritmo Boyer-Moore-Horspool*

Análise

Para o algoritmo original BM, os dois tipos de deslocamento (ocorrência e casamento) podem ser pré-computados com base apenas no padrão e no alfabeto. Assim, a complexidade de tempo e de espaço para esta fase é $O(m + c)$. Obtidas as duas funções de deslocamento A cada caractere lido o algoritmo escolhe o valor que proporciona o maior deslocamento. O pior caso do algoritmo é $O(n + rm)$, onde r é igual ao número total de casamentos, o que torna o algoritmo ineficente quando o número de casamentos é grande. O melhor caso e o caso médio para o algoritmo é $O(n/m)$, um resultado excelente pois executa em tempo sublinear.

Para o algoritmo BMH, o deslocamento ocorrência também pode ser pré-computado com base apenas no padrão e no alfabeto, e a complexidade de tempo e de espaço para essa fase é $O(c)$. Para a fase de pesquisa, o pior caso do algoritmo é $O(nm)$, o melhor caso é $O(n/m)$ e o caso esperado é $O(n/m)$, se c não é pequeno e m não é muito grande.

1.2 Casamento Aproximado

Existem variações com relação ao casamento exato de cadeias, sendo a mais importante aquela que permite operações de inserção, de substituição e de retirada de caracteres do padrão. A Figura 1.9 mostra três ocorrências do padrão `teste` em que os casos de inserção, de substituição, e de retirada de caracteres no padrão acontecem, a saber: (i) no primeiro, um espaço é inserido entre o terceiro e quarto caracteres do padrão, (ii) no segundo, o último caractere do padrão é substituído pelo caractere `a`, e (iii) no terceiro, o primeiro caractere do padrão é retirado.

```
    tes te
       testa
            este
os testes testam estes alunos . . .
```

Figura 1.9 *Exemplo de casamento aproximado do padrão* `teste`

O número k de operações de inserção, de substituição e de retirada de caracteres necessário para transformar uma cadeia x em outra cadeia y é conhecido na literatura como **distância de edição** (Levenshtein, 1965). Assim, a distância de edição entre duas cadeias x e y, $ed(x, y)$, é o menor número de operações necessárias para converter x em y, ou vice versa. Por exemplo, $ed(\texttt{t}este, \texttt{e}stende) = 4$, valor obtido por meio de uma retirada do primeiro `t` de P e a inserção dos 3 caracteres `nde` ao final de P. O problema do casamento aproximado de cadeias é o de encontrar todas as ocorrências em T de cada P' que satisfaz $ed(P, P') \leq k$.

O problema da busca aproximada somente faz sentido para $0 < k < m$, porque no caso de $k = m$ toda subcadeia de comprimento m pode ser convertida em P por meio da substituição de m caracteres. O caso em que $k = 0$ corresponde ao casamento exato de cadeias. O nível de erro $\alpha = k/m$, fornece uma medida da fração do padrão que pode ser alterado. Em geral $\alpha < 1/2$ para a maioria dos casos de interesse.

O **casamento aproximado de cadeias**, também conhecido como **casamento de cadeias permitindo erros**, é o problema de encontrar um padrão P

em um texto T quando um número limitado k de operações (erros) de inserção, de substituição e de retirada é permitido entre P e suas ocorrências em T.

A seguir, vamos mostrar como lidar com o problema de casamento aproximado de cadeias por meio de um autômato não-determinístico.

Algoritmos Baseados em Autômatos

Conforme mostrado em Navarro e Raffinot (2002), uma maneira de tratar o problema do casamento aproximado de cadeias é modelar a pesquisa por um autômato não-determinístico. Assim como o algoritmo Shift-And para casamento exato de cadeias apresentado na Seção 1.1.1, o algoritmo que vamos apresentar também usa o **paralelismo de bit**, o qual simula o funcionamento de um autômato não-determinístico.

A Figura 1.10 apresenta três autômatos não-determinísticos para casamento aproximado do padrão P ={`teste`}. A primeira linha de cada autômato representa casamento exato e a segunda linha representa casamento aproximado com 1 erro. A Figura 1.10(a) mostra o autômato que permite a inserção de um caractere em qualquer posição de P, a Figura 1.10(b) mostra o autômato que permite a substituição de um caractere em qualquer posição de P, e a Figura 1.10(c) mostra o autômato que permite a retirada de um caractere de qualquer posição de P.

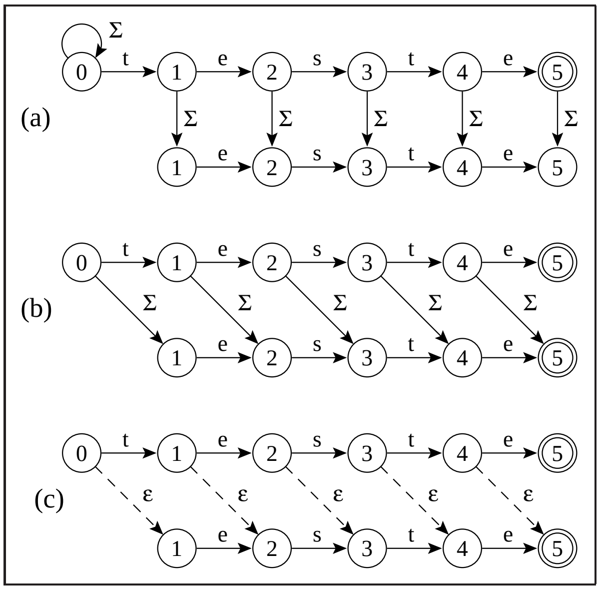

__Figura 1.10__ Autômatos para casamento aproximado permitindo um erro. (a) Erro de inserção de caractere. (b) Erro de substituição de um caractere. (c) Erro de retirada de um caractere

Em qualquer dos três autômatos da Figura 1.10, uma aresta horizontal representa o casamento de um caractere, isto é, se os caracteres do padrão e do texto casam, então avançamos em P e T. Uma aresta vertical na Figura 1.10(a) insere um caractere no padrão, o que significa que avançamos em T mas não em P. Uma aresta diagonal sólida na Figura 1.10(b) substitui um caractere, o que significa que avançamos em T e P. Uma aresta diagonal tracejada na Figura 1.10(c) retira um caractere de P, o que significa que avançamos em P mas não em T, o que equivale a uma **transições-ϵ**. O *self-loop* inicial em cada autômato permite que uma ocorrência se inicie em qualquer posição em T. Cada autômato sinaliza uma ocorrência sempre que o estado final mais à direita se torna ativo.

A Figura 1.11 apresenta um autômato que permite casamento aproximado com $k = 2$ erros para $P = \{\texttt{teste}\}$. Nesse caso, as três operações de distância de edição estão juntas em um único autômato. A linha 1 representa casamento exato ($k = 0$), a linha 2 representa casamento aproximado permitindo um erro ($k = 1$), e a linha 3 representa casamento aproximado permitindo dois erros ($k = 2$). Uma vez que um estado no autômato está ativo, todos os estados nas linhas seguintes na mesma coluna também estão ativos.

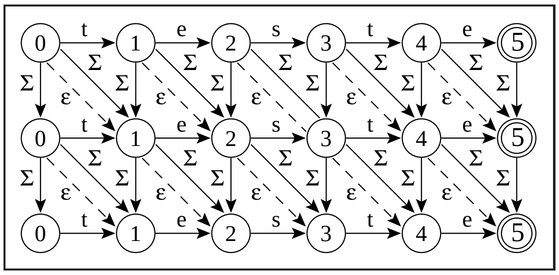

Figura 1.11 *Autômato para casamento aproximado permitindo até dois erros, podendo ser de inserção, de substituição, ou de retirada de um caractere*

Shift-And para Casamento Aproximado

Os melhores algoritmos para casamento aproximado de cadeias utilizam **paralelismo de bit**. O algoritmo Shif-And para casamento aproximado de cadeias também simula um autômato não-determinístico, como o autômato mostrado na Figura 1.11. O algoritmo que vamos estudar a seguir é uma extensão do algoritmo Shift-And para casamento exato de cadeias apresentado na Seção 1.1.1.

O algoritmo Shift-And para casamento aproximado de cadeias foi proposto por Wu e Manber (1992). O algoritmo empacota cada linha j ($0 < j \leq k$) do autômato não-determinístico em uma palavra R_j diferente do computador. Para cada novo caractere lido do texto todas as transições do autômato são simuladas usando operações entre as $k+1$ máscaras de *bits*. Todas as $k+1$ máscaras de *bits* têm a mesma estrutura e assim o mesmo *bit* é alinhado com a mesma posição no texto.

Na posição i do texto, os novos valores R'_j, $0 < j \leq k$, são obtidos a partir dos valores correntes R_j, a saber:

$$R'_0 = ((R_0 >> 1) \mid 10^{m-1}) \ \& \ M[T[i]]$$

e

$$R'_j = ((R_j >> 1) \ \& \ M[T[i]]) \mid R_{j-1} \mid (R_{j-1} >> 1) \mid (R'_{j-1} >> 1),$$

onde M é a tabela do algoritmo Shift-And para casamento exato da Seção 1.1.1. A pesquisa inicia com $R_j = 1^j 0_{m-j}$. Conforme esperado, R_0 equivale ao algoritmo Shift-And para casamento exato, e as outras linhas R_j recebem 1s (estados ativos) também de linhas anteriores. Considerando a Figura 1.11, a fórmula para R' expressa arestas horizontais indicando casamento de um caractere, arestas verticais indicando inserção, arestas diagonais cheias indicando substituição e arestas diagonais tracejadas indicando retirada, respectivamente.

O Programa 1.6 apresenta um primeiro refinamento do algoritmo Shif-And para casamento aproximado de cadeias.

```
Shift-And-Aproximado (P = p1p2 ... pm, T = t1t2 ... tn, k);
  {-- Pré-processamento --}
  for c ∈ Σ do M[c] := 0^m;
  for j := 1 to m do M[pj] := M[pj] | 0^(j-1)10^(m-j);
  {-- Pesquisa --}
  for j := 0 to k do Rj := 1^j0^(m-j);
  for i := 1 to n do
    Rant := R0;
    Rnovo := ((Rant >> 1) | 10^(m-1)) & M[T[i]];
    R0 := Rnovo;
    for j := 1 to k do
      Rnovo := ((Rj >> 1 & M[T[i]]) | Rant | ((Rant | Rnovo) >> 1);
      Rant := Rj;
      Rj := Rnovo;
    if Rnovo & 0^(m-1)1 ≠ 0^m then 'Casamento na posicao i';
```

***Programa 1.5** Primeiro refinamento do algoritmo Shift-And para casamento aproximado de cadeias*

Infelizmente, como no caso do algoritmo Shift-And para casamento exato de cadeias, as operações *and*, *or*, deslocamento à direita e complemento, a serem realizadas sobre o conjunto R, não podem ser realizadas com eficiência na linguagem Pascal, o que compromete o conceito de paralelismo de *bit*. Por essa razão, não apresentamos a implementação do algoritmo Shift-And em Pascal, mas apenas a implementação em C. O Programa 1.6 mostra a implementação do algoritmo Shift-And-Aproximado na linguagem C.

Programa 1.6 *Implementação do algoritmo Shift-And para casamento aproximado de cadeias na linguagem C*

1.3 Compressão de Cadeias de Caracteres

Ênfase em algoritmos para compressão de textos em linguagem natural e seqüências de inteiros.

Notas Bibliográficas

Baeza-Yates e Régnier (1992) Baeza-Yates (1992) Navarro e Raffinot (2002) é a melhor referência que existe atualmente sobre casamento de cadeias de caracteres. O livro enfatiza algoritmos e implementações que apresentam o melhor desempenho na prática, incluindo todos os principais desenvolvimentos recentes em casamento complexo de cadeias, desde o casamento simples, múltiplo e estendido de cadeias até o casamento exato e aproximado de expressões regulares. Boyer e Moore (1977)

O algoritmo Shift-And foi proposto por Baeza-Yates e Gonnet (1989)

Exercícios

1) Altere o procedimento *ForcaBruta* para pesquisar um padrão em um texto, mostrado no Programa 1.1, para permitir a pesquisa aproximada permitindo k erros. Compare o desempenho da sua implementação com a implementação do algoritmo *Shift-And* para busca aproximada.
2) Faça um estudo comparativo dos algoritmos força bruta, BMH e Shif-And para pesquisa exata em cadeias.