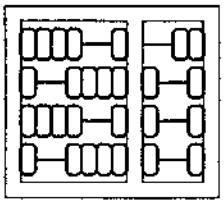


Métodos Eficientes para Reconhecimento de Padrões em Texto[1]

Marcus Vinícius Alvim Andrade[2]

Departamento de Ciência da Computação

IMECC – UNICAMP

Banca Examinadora:

- Cláudio Leonardo Lucchesi (Orientador)[3]
- Imre Simon[4]
- João Meidanis[3]
- Tomasz Kowaltowski (suplente)[3]

[1]Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação da UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

[2]O autor é Bacharel em Matemática pela Universidade Federal do Viçosa (MG) e atualmente é professor do Departamento de Informática daquela universidade.

[3]Professor do Departamento de Ciência da Computação - IMECC - UNICAMP.

[4]Professor do Departamento de Ciência da Computação - IME - USP.

Métodos Eficientes para Reconhecimento de Padrões em Texto

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Marcus Vinícius Alvim Andrade e aprovada pela Comissão Julgadora.

Campinas, 03 de outubro de 1993.

Prof. Dr. C L Lucchesi

Cláudio Leonardo Lucchesi

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do Título de MESTRE em Ciência da Computação.

À memória de meu pai.

Agradecimentos

Um agradecimento muito especial a toda a minha família. Ao meu saudoso pai, Pedro Arnaldo D. Andrade agradeço pelo constante apoio, incentivo e amor que sempre me foi dedicado. À minha mãe, Annabela F. A. Andrade, agradeço pelas infinitas demonstrações de amor e carinho recebidas e também pelas incontáveis orações que sem dúvida se converteram em força e perseverença de que tanto necessitamos. Aos meus irmãos Pedro Ricardo e Rodrigo que são meus eternos amigos e companheiros. À minha adorada amiga e esposa dedico os meus mais sinceros agradecimentos, pois a sua contribuição a este trabalho foi inestimável, pois além de ser o meu principal fator de motivação e equilíbrio ela é a personificação dos meus sonhos, desejos e vontades. Gostaria de agradecer também aos meus avós, tios, primos, sogro, sogra, cunhadas etc.

Ao meu amigo e orientador Lucchesi agradeço a sua dedicação profissional que permitiu a realização deste trabalho, mas agradeço principalmente o apoio pessoal recebido nas mais variadas situações e também as nossas alegres e descontraídas reuniões.

A todos os colegas de curso (Atta, Inês, Heitor, Carrard, Raimundo, $\cdots$) agradeço pela agradável convivência e pela ajuda profissional e pessoal. Em particular, gostaria de destacar a contribuição dos colegas Herbert Baier, Marcelo Carvalho e Mário Harada que apresentaram diversas sugestões importantes ao trabalho. Além disso, agradeço aos amigos Fábio Lucena, Maurício Fernandez, Mário Harada e Thierson Couto que foram companhias constantes em infindáveis reuniões sobre Teoria da Computação.

Aos eternos amigos de Viçosa e de Belo Horizonte presto um agradecimento especial pelas cervejadas e pelas longas conversas e risadas que sempre nos reabasteciam de animação para continuar o trabalho.

Aos colegas de profissão do Departamento de Informática da Universidade Federal de Viçosa agradeço pelo apoio e pelo estímulo recebidos.

A todos os professores e funcionários do Departamento de Ciência da Computação da UNICAMP agradeço pela cordialidade e carinho com que sempre fui tratado, sendo que aqui vai um agradecimento especial ao nosso secretário Luiz Cláudio, meu "assessor para assuntos aleatórios".

Finalmente, agradeço à Universidade Federal de Viçosa e à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) que através do Programa Institucional de Capacitação de Docentes (PICD) deram o suporte financeiro à realização deste trabalho.

Abstract

The pattern matching problem arises very frequently in several areas of knowledge and basically consists in determining if a given object (pattern) occurs in any place of another object (usually bigger). There are many variations of this problem, for example, objects with one or more dimensions, approximate pattern matching etc.

In this work we approach the pattern matching problem on one dimension that in the literature is normally named the string matching problem. More precisely, we confined ourselves to the exact pattern matching problem. Our main objective is to present (describe and analyse) clearly and precisely the most important algorithms to solve this problem.

In chapter 2 we describe the Knuth, Morris and Pratt algorithm through automata. It is worth mentioning that, although the association between this algorithm and automata is cited in the literature quite often, in general, automata are not effectively used in the description of the algorithm. This approach made the description of the algorithm very simple. Moreover, in the analysis of the algorithm, the proofs of some of the results were accomplished in a clearer way.

In chapter 3 we present the Boyer and Moore algorithm, which is extremelly efficient in practice and the majority of the algorithms found in the literature are based on the ideas of this algorithm. Actually, we present a little variation of this algorithm that is simpler and, in some cases, more efficient than the original algorithm. Moreover, in this chapter, we deal with the complexity analysis of the Boyer and Moore algorithm. We also present several variations of this algorithm.

In appendix A we describe other pattern matching algorithms that were recently developed and finally, in the appendix B, we analise theoretically the average behaviour of some algorithms and also describe the results of some empirical analyses made by other authors.

Sumário

O problema de reconhecimento de padrões surge muito freqüentemente em diversas áreas e consiste basicamente em determinar se um dado objeto (padrão) ocorre em alguma parte de um outro objeto (geralmente bem maior). Existem diversas variações sobre o tema, por exemplo, objetos com uma ou mais dimensões, reconhecimento aproximado de padrões etc.

Neste trabalho abordaremos a questão do reconhecimento de padrões unidimensionais que na literatura normalmente é citado como reconhecimento de padrões em texto. Além disso, nos concentramos no problema de reconhecimento exato de padrões. Nosso objetivo principal é apresentar (descrever e analisar) de forma clara e precisa os principais algoritmos que solucionam o problema em questão.

No capítulo 2 descrevemos o algoritmo de Knuth, Morris e Pratt através de autômatos, sendo que vale destacar que, embora a associação entre este algoritmo e autômatos seja citada na literatura com bastante freqüência, normalmente ela não é efetivamente utilizada na descrição do algoritmo. Esta abordagem tornou a descrição do algoritmo bastante simples. Além disso, na análise do algoritmo, a demonstração de alguns resultados foram realizadas de forma bem mais clara do que a originalmente proposta.

No capítulo 3 apresentamos o algoritmo de Boyer e Moore que é um algoritmo extremamente eficiente na prática e no qual se baseiam a maioria dos outros algoritmos existentes. Inclusive, nós apresentamos uma variação deste algoritmo que pode ser descrita de forma mais simples do que o algoritmo original e, em alguns casos, é mais eficiente do que ele. Além disso, neste capítulo tratamos da questão da análise de complexidade do algoritmo de Boyer e Moore que é um problema razoavelmente complexo e apresentamos ainda as principais variações deste algoritmo.

No apêndice A descrevemos outros algoritmos propostos recentemente que solucionam o problema de reconhecimento de padrões em textos e finalmente, no apêndice B, analisamos teoricamente o comportamento médio de alguns algoritmos e também descrevemos os resultados de algumas análises empíricas realizadas por outros autores.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O problema de reconhecimento de padrões, há algum tempo, vem recebendo uma atenção considerável por parte de vários pesquisadores da área de ciência de computação. Este interesse se justifica porque o reconhecimento de padrões é uma componente importante de vários outros problemas que surgem nas mais diversas áreas de conhecimento, dentre os quais podemos destacar:

- análises meteorológicas, agrícolas, geológicas etc, de imagens geradas por satélites;
- análises de exames médicos como, por exemplo, tomografias, em que se deseja verificar a existência de tumores;
- biologia computacional que estuda o DNA para reconhecer as subseqüências de nucleotídeos que o compõem;
- processamento de textos, quando se deseja encontrar as ocorrências de determinadas palavras;
- pesquisas bibliográficas em que se deseja obter todas as publicações que façam referência a um determinado assunto.

Estas várias aplicações dão origem a diversas variações para o problema, tais como: reconhecimento aproximado de padrões, reconhecimento de padrões em duas ou três dimensões, reconhecimento da maior subseqüência repetida etc. Para cada uma destas variações, existem inúmeras soluções que visam explorar as diferentes características de cada situação. Desta forma,

podemos perceber que uma discussão detalhada das diversas variações deste problema é um assunto bastante longo. Assim, nesta dissertação, vamos nos restringir ao problema de reconhecimento exato de padrões unidimensionais, sendo que de agora em diante, referenciaremos este problema apenas como reconhecimento de padrões em texto. Vale salientar que decidimos abordar esta variação do problema porque muitas das soluções existentes para as outras variações se baseiam em soluções originalmente propostas para este tipo de problema. O leitor interessado pode obter uma discussão sobre as outras variações do problema em [Ste92] e [Sed83].

1.1 Objetivos e Organização da Dissertação

Conforme relatamos acima, a descrição detalhada de soluções para todas as variações do problema de reconhecimento de padrões tornaria esta dissertação extremamente longa. Portanto, nos permitimos concentrar na questão do reconhecimento de padrões em texto.

É interessante ressaltar que existem inúmeras soluções para este problema, mas muitas destas soluções podem ser vistas como pequenas variações de alguns algoritmos que são considerados marcos nesta área, pois eles introduziram alguns conceitos fundamentais. Na realidade, atualmente alguns destes algoritmos (por exemplo, o algoritmo de Knuth, Morris e Pratt) são considerados marcos de uma área mais geral que é a área de desenvolvimento e análise de algoritmos.

Assim, o objetivo inicial desta dissertação é descrever e analisar os principais algoritmos de reconhecimento de padrões em texto. Na figura 1.1 relacionamos cronologicamente estes algoritmos e além disso, estabelecemos também uma relação de dependência cuja interpretação pode ser exemplificada da seguinte forma: o algoritmo de Galil incorpora novos conceitos ao algoritmo de Boyer e Moore, sendo que estes novos conceitos se baseiam em observações de cunho teórico; por outro lado, o algoritmo de Horspool também incorpora novos conceitos ao algoritmo de Boyer e Moore, porém estes conceitos se baseiam em observações de cunho prático. Vale frisar que esta figura produz um melhor resultado quando o leitor conhece pelo menos as idéias centrais de cada um dos algoritmos. Portanto, convidamos ao leitor a analisá-la novamente após a leitura completa desta dissertação.

Ainda com relação à figura 1.1, vale acrescentar que os algoritmos que aparecem no ramo em linha tracejada não se baseiam exclusivamente em

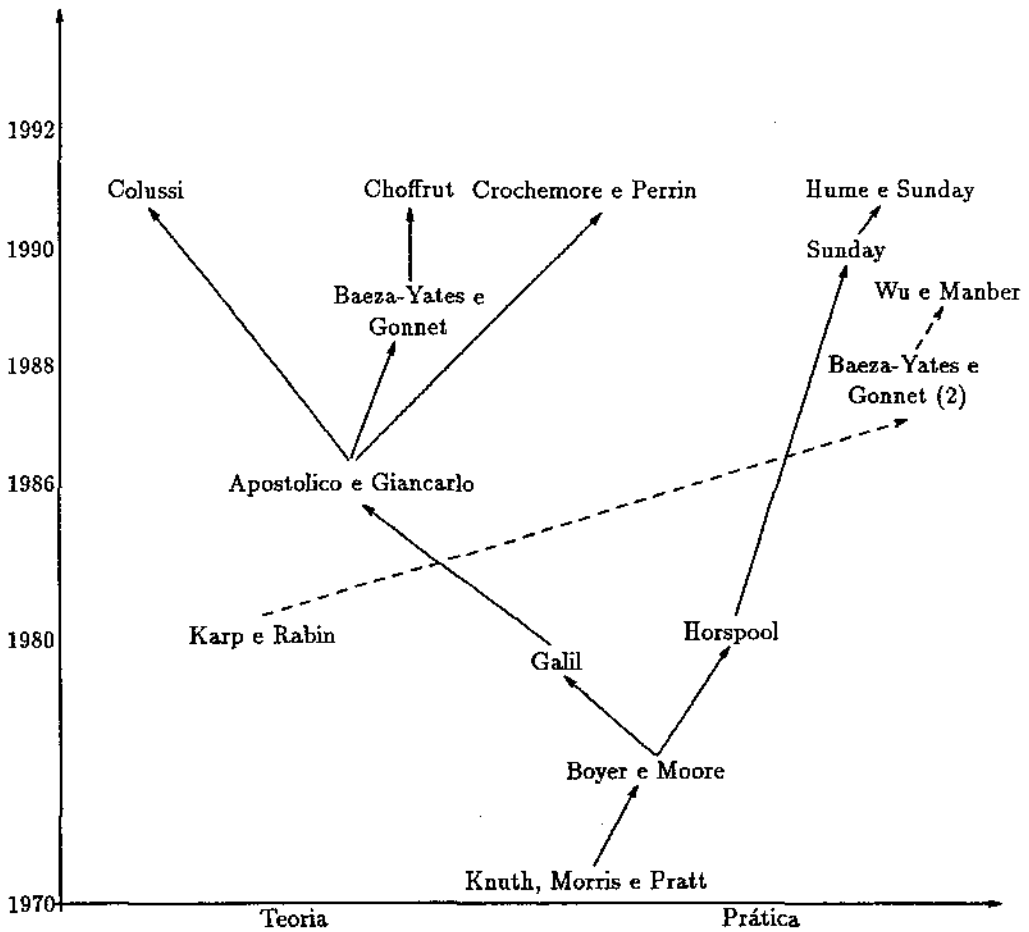


Figura 1.1: Principais algoritmos para o reconhecimento de padrões em texto.

comparações. Por exemplo, o algoritmo de Karp e Rabin utiliza conceitos de funções de "hashing", o algoritmo de Baeza-Yates e Gonnet (2) utiliza operações em bits e assim por diante. Estes algoritmos, embora solucionem o problema de reconhecimento de padrões em texto, são aplicados com mais freqüência em outras variações do problema, como por exemplo, no reconhecimento de padrões em mais de uma dimensão ou no reconhecimento aproximado de padrões.

Nesta dissertação, nos concentraremos naqueles algoritmos para reconhecimento de padrões em texto que baseiam fundamentalmente em comparações para realizar este reconhecimento, ou seja, consideraremos apenas aqueles algoritmos que aparecem no ramo em linha contínua na figura 1.1. Uma exceção será feita ao algoritmo de Karp e Rabin que será apresentado na seção A.5.

Concluindo, podemos estabelecer que o objetivo desta dissertação é apresentar (descrever e analisar) os principais algoritmos de reconhecimento de padrões em texto que se baseiam exclusivamente em comparações entre o padrão e o texto.

Agora, vamos estabelecer quais são os objetivos da descrição e da análise de cada algoritmo. Inicialmente, conforme relatado por vários autores, a utilização efetiva dos principais algoritmos de reconhecimento de padrões em texto tem sido enormemente prejudicada porque muitas pessoas não compreendem o funcionamento destes algoritmos. Dentre estes relatos podemos destacar: *"muitos programadores às vezes não acreditam que o algoritmo de Boyer e Moore (se é que eles ao menos já ouviram falar nele) é realmente uma abordagem prática"* [Hor80]; *"tanto o algoritmo de Knuth, Morris e Pratt como o algoritmo de Boyer e Moore requerem algum tipo de pré-processamento no padrão cujo funcionamento é difícil de entender e isto tem limitado a utilização destes algoritmos"* [Sed83]; *"em parte porque os melhores algoritmos apresentados na literatura são difíceis de entender e implementar, o conhecimento de algoritmos rápidos e práticos não é muito comum"* [HS91].

Em nossa opinião, uma parte da culpa por esta situação deve ser creditada aos autores dos algoritmos, pois geralmente eles não se preocupam em descrever de uma forma clara os algoritmos que eles estão propondo (é verdade que muitas vezes esta situação é causada pela restrição de espaço que a publicação lhes impõe). Assim, pretendemos minimizar este problema descrevendo os algoritmos da forma mais clara e didática possível.

Visto que restringiremos a nossa investigação a algoritmos de reconhecimento de padrões em texto que efetivamente realizam comparações entre o padrão e o texto então vamos estabelecer que o comportamento dos algoritmos será determinado pelo número de comparações que eles efetuam, ou seja, vamos definir que a *função de complexidade* dos algoritmos é dada pelo número de comparações efetuadas. É importante ressaltar que vamos considerar apenas as comparações que envolvem símbolos pertencentes ao padrão e ao texto, isto é, vamos desprezar comparações que envolvam, por exemplo, "variáveis de controle" do algoritmo e limites numéricos.

Observe que a função estabelecida acima implicitamente considera apenas a questão da complexidade de tempo dos algoritmos e ignora a questão da complexidade de espaço. Isto porque a complexidade de espaço, no caso dos algoritmos de reconhecimento de padrões em texto, geralmente é negligenciada, pois os principais algoritmos para solucionar este problema requerem espaço proporcional ao tamanho do padrão e este, normalmente é bem menor do que o do texto.

É importante destacar que, por definição, as funções de complexidade têm como argumento o tamanho do problema, que no caso é dado pelo tamanho do padrão e do texto e, como é de praxe, nas análises dos algoritmos utilizaremos a notação O ("O grande") cuja descrição pode ser obtida em várias publicações, como por exemplo, [AHU74], [GKP90] e [Ziv86].

Finalmente, o nosso objetivo nas análises dos algoritmos é avaliar o comportamento destes algoritmos considerando situações de pior caso e de caso médio.

Estabelecidos os objetivos principais deste trabalho, vejamos agora como ele foi organizado. Inicialmente, no restante deste capítulo, apresentamos algumas definições e conceitos que serão utilizados nas descrições e análises dos algoritmos, sendo que na seção 1.2 definimos o problema de reconhecimento de padrões em texto e adicionalmente estabelecemos alguns conceitos sobre seqüências de caracteres que serão amplamente utilizados ao longo desta dissertação. Além disso, apresentamos também uma primeira solução para o problema, juntamente com a sua análise de pior caso. Na seção 1.3 tratamos da questão do limite inferior para o problema de reconhecimento de padrões em texto, ou seja, estabelecemos o número mínimo de comparações que devem ser efetuadas por qualquer algoritmo que solucione este problema. Finalmente, na seção 1.4 apresentamos alguns dos principais resultados sobre periodicidade em seqüências de caracteres.

O capítulo 2 é dedicado à descrição e análise do algoritmo de Knuth, Morris e Pratt. É interessante ressaltar que em quase todas as publicações sobre este algoritmo, é citado que ele pode ser modelado através de autômatos. Entretanto, na imensa maioria dos casos, esta abordagem não é efetivamente utilizada para descrevê-lo. Em nossa opinião (com a qual certamente o leitor concordará), a compreensão deste algoritmo se torna bastante simples quando o descrevemos através de um autômato, conforme apresentamos nas seções 2.1 e 2.2. Agora, o objetivo de utilizar autômatos na descrição deste algoritmo era apenas facilitar a sua compreensão, porém, esta abordagem permitiu que a análise do algoritmo (seção 2.3) fosse realizada de forma bem mais simples do que a originalmente apresentada pelos autores. Encerramos esse capítulo, descrevendo algumas variações do algoritmo (seção 2.4), onde destacamos as variações que permitem a sua execução "on-line", ou seja, que permitem que a busca seja realizada à medida que os caracteres do padrão sejam fornecidos (como ocorre em vários editores de texto).

No capítulo 3, apresentamos o algoritmo de Boyer e Moore que conforme veremos, na prática, é um algoritmo mais eficiente do que o algoritmo anterior. Vale destacar que visando facilitar a descrição, obtivemos uma variação do algoritmo original que, em vários casos, se mostra mais eficiente do que aquele. Na seção 3.2, novamente lançamos mão de autômatos, sendo que neste caso, os empregamos para descrever como algumas funções utilizadas pelo algoritmo podem ser computadas. Na seção 3.3, apresentamos uma análise de complexidade do algoritmo de Boyer e Moore, onde utilizamos conceitos sobre análise amortizada. É importante salientar que a questão da complexidade deste algoritmo é um problema bastante complexo com algumas questões ainda em aberto. Na seção 3.4, apresentamos diversas variações deste algoritmo e, em particular, ressaltamos que, na seção 3.4.2, obtivemos um algoritmo para computar uma das funções utilizadas por esta variação do algoritmo de Boyer e Moore que é bem mais simples do que o algoritmo originalmente proposto.

Agora, no apêndice A, apresentamos outros algoritmos para reconhecimento de padrões em texto e desta forma, esgotamos os algoritmos citados na figura 1.1. Vale salientar que neste apêndice acrescentamos uma descrição do algoritmo de Karp e Rabin, embora ele não se inclua no ramo dos algoritmos que se baseiam exclusivamente em comparações. Decidimos incluir uma descrição deste algoritmo devido ao seu valor histórico, pois ele

foi um dos primeiros algoritmos para reconhecimento de padrões em texto que não se baseia exclusivamente em comparações.

Finalmente, no apêndice B, abordamos a questão da análise de caso médio dos algoritmos Ingênuo, KMP e BM. Neste apêndice apresentamos uma análise teórica para cada um destes algoritmos e além disso, na seção B.4, descrevemos os resultados de algumas análises empíricas (realizadas por outros autores) que ilustram o comportamento dos principais algoritmos na prática. O objetivo da apresentação deste tipo de análise é permitir que o leitor possa avaliar comparativamente ou sob determinadas circunstâncias os principais algoritmos descritos neste trabalho.

1.2 A Definição do Problema e uma Primeira Solução

Antes de definir o problema de reconhecimento de padrões em textos será necessário estabelecer alguns conceitos importantes que serão amplamente utilizados ao longo desta dissertação.

Definição 1.1 Um *alfabeto* Σ é um conjunto finito de símbolos, os quais denominamos *caracteres*. Uma *seqüência de caracteres* é obtida concatenando-se um número finito de caracteres do alfabeto, sendo que o *comprimento* de uma seqüência corresponde ao número de caracteres que a compõem. Denotaremos o comprimento de uma seqüência α por $|\alpha|$. Um caso particular é a seqüência de comprimento zero que denominaremos *seqüência vazia* e que representaremos por ϵ.

Vamos representar uma seqüência α com n caracteres por $\alpha_1 \ldots \alpha_n$. Assim, α_i indica o i-ésimo caractere da seqüência contado a partir de sua extremidade esquerda. Além disso, adotaremos que $\alpha_i \ldots \alpha_j$, onde $1 \leq i \leq j+1 \leq m+1$, é uma *subseqüência* de α, sendo que se $i = j+1$ então a subseqüência é vazia.

Definição 1.2 Seja α uma seqüência com n caracteres, onde $n \geq 0$. Assim, para todo $j : 0 \leq j \leq n$, $\alpha_1 \ldots \alpha_j$ é um *prefixo* de α. Se $j < n$ então temos um *prefixo próprio* de α. Por outro lado, para todo $i : 1 \leq i \leq n+1$, $\alpha_i \ldots \alpha_n$ é um *sufixo* de α. Se $i > 1$ então temos um *sufixo próprio* de α.

Definição 1.3 Seja $l \geq 0$. Então α^l corresponde à seqüência dada por:

$$\alpha^l := \begin{cases} \alpha\alpha^{(l-1)} & \text{se } l > 0 \\ \epsilon & \text{caso contrário.} \end{cases}$$

Agora, sejam α e β seqüências de caracteres tais que $|\beta| \geq |\alpha|$ e sejam $n := |\beta|$ e $m := |\alpha|$. Se existe $k : 0 \leq k \leq n - m$ tal que

$$\alpha_1 \ldots \alpha_m = \beta_{k+1} \ldots \beta_{k+m} \tag{1.1}$$

então dizemos que há uma *ocorrência* de α na posição $k + 1$ de β.

Observe que a equação (1.1) equivale a

$$\alpha_j = \beta_{k+j} \quad \forall j : 1 \leq j \leq m.$$

Finalmente, podemos definir o problema de reconhecimento de padrões em texto.

Definição 1.4 Dado um *padrão* $p = p_1 \ldots p_m$, com $m > 0$, e um *texto* $t = t_1 \ldots t_n$, com $n \geq m$, o problema de *reconhecimento de padrões em texto* consiste em obter todas as posições no texto t onde o padrão p ocorre.

Vale salientar que este problema também aparece em sua forma reduzida, onde se deseja obter apenas a primeira (a mais à esquerda) ocorrência do padrão no texto.

Ao longo desta dissertação vamos adotar que p representa o padrão, sendo que $m := |p|$, e que t representa o texto, sendo que $n := |t|$. Observe que não é imposta nenhuma restrição ao alfabeto Σ e portanto, as seqüências p e t podem ser compostas por quaisquer tipos de símbolos (letras, dígitos, etc).

Uma solução simples para o problema definido acima consiste em verificar se o padrão p ocorre em cada posição $k + 1$ do texto, sendo que k é tal que $0 \leq k \leq n - m$, ou seja, nesta solução o padrão é verificado com todas as subseqüências $t_{k+1} \ldots t_{k+m}$. Cada uma destas verificações consiste em comparar p_j com t_{k+j}, onde $j : 1 \leq j \leq m$, até que uma diferença seja obtida ou até que a extremidade direita do padrão seja ultrapassada (isto é, quando $j > m$). No primeiro caso temos que o padrão não ocorre na posição $k + 1$ do texto e no segundo, o padrão ocorre na posição $k + 1$. Como desejamos obter todas as ocorrências do padrão então este processo

deve ser repetido até que a extremidade direita do texto seja ultrapassada. Denominaremos este método de *algoritmo Ingênuo.*

O algoritmo descrito acima é bastante simples e dispensa a apresentação de uma implementação. Entretanto, aproveitaremos a oportunidade para colocar o leitor em contato com a pseudo-linguagem que utilizaremos na apresentação de implementações dos próximos algoritmos. Assim, na figura 1.2 mostramos uma implementação do algoritmo Ingênuo que poderia ser um pouco mais eficiente se estendêssemos o padrão com uma posição e acrescentássemos nesta posição uma "sentinela" (isto evitaria a comparação $j \leq m$ no laço **while** mais interno). Porém, como nosso objetivo central é a clareza na descrição dos algoritmos e como, em nossa opinião, geralmente estas técnicas de otimização (principalmente nos próximos algoritmos) tornam obscuros alguns detalhes essenciais do algoritmo então evitaremos utilizá-las. Naqueles casos em que acharmos realmente interessante adotá-las alertaremos o leitor para o fato.

Agora, observe que o algoritmo Ingênuo, no pior caso, efetua $m(n-m+1)$ comparações, pois ele verifica se o padrão ocorre em cada posição $k+1$ do texto, sendo que k é tal que $0 \leq k \leq n-m$. Assim, o algoritmo verifica $n-m+1$ possíveis ocorrências do padrão. Em cada uma destas verificações são efetuadas, no máximo, m comparações (quando a verificação resulta numa ocorrência do padrão ou quando uma diferença é obtida na posição m do padrão). Portanto, temos que o número de comparações, no pior caso, é $m(n-m+1)$, o que nos leva a concluir que o comportamento do algoritmo Ingênuo no pior caso é $O(mn)$. Para ilustrar o pior caso do algoritmo Ingênuo considere

$$\begin{aligned} p &:= a^{m-1}b \\ t &:= a^n. \end{aligned}$$

Desta forma, nestes casos o algoritmo Ingênuo é bastante ineficiente. Porém, na prática estas situações são pouco prováveis de acontecer e, conforme veremos na seção B.1, o comportamento do algoritmo Ingênuo no caso médio é linear em $m+n$.

Nos próximos capítulos apresentaremos outros algoritmos que são consideravelmente mais eficientes do que o algoritmo Ingênuo, principalmente quando consideramos o pior caso.

Algoritmo *Ingênuo*;

Entrada : *O padrão p e o texto t.*
Saída : *Lista das posições no texto onde o padrão ocorre.*

```
begin
   k := 0;
   while k ≤ n − m do
   begin
      j := 1;
      while j ≤ m and p_j = t_{k+j} do j := j + 1;
      if j > m then
         write('O padrão ocorre na posição ', k + 1);
      k := k + 1
   end
end.
```

Figura 1.2: Uma implementação do algoritmo Ingênuo; no conectivo **and** a segunda condição somente é avaliada caso a primeira condição seja verdadeira.

1.3 Um Limite Inferior para o Problema

Nesta seção vamos estabelecer um limite inferior para o problema de reconhecimento de padrões em textos. Conforme salientamos anteriormente, uma boa maneira de avaliar os algoritmos que solucionam este problema é considerar o número de caracteres comparados até que a solução seja obtida. Assim, considerando apenas as comparações entre caracteres, vamos estabelecer um número mínimo de comparações que devem ser efetuadas, no pior caso, por qualquer algoritmo de reconhecimento de padrões em textos. Na análise a seguir vamos adotar que o alfabeto Σ consiste de pelo menos 2 caracteres, ou seja, $|\Sigma| \geq 2$.

Um dos primeiros resultados apresentados sobre o limite inferior para o problema de reconhecimento de padrões em texto foi obtido por Rivest ([Riv77]) que demonstrou que, no pior caso, qualquer algoritmo de reconhecimento de padrões em texto deve "examinar" no mínimo $n - m + 1$ caracteres do texto para obter a primeira ocorrência do padrão. É importante ressaltar que neste resultado o "exame" de um caractere do texto pode corresponder a mais de uma comparação e além disso, é considerada apenas a primeira ocorrência do padrão no texto. A seguir, apresentaremos um resultado recentemente proposto por Galil e Giancarlo ([GG91]) onde os autores estabelecem o número mínimo de comparações (efetivas) que devem ser realizadas, no pior caso, por um algoritmo de reconhecimento de padrões em texto para obter todas as ocorrências do padrão.

Inicialmente, é fácil perceber que se o texto tem comprimento n então, para obter todas as ocorrências de um padrão neste texto, um algoritmo deve efetuar, no pior caso, pelo menos n comparações. Por exemplo, considere $p = a$. Então, para obter todas as ocorrências do padrão no texto, um algoritmo deve verificar todos os caracteres deste texto pelo menos um vez.

Agora, conforme veremos na seção 2.4 (lema 2.20), se o alfabeto Σ é tal que $|\Sigma| = 2$ então existe uma variação do algoritmo de Knuth, Morris e Pratt que obtém todas as ocorrências do padrão no texto efetuando exatamente n comparações. Isto implica que, para alfabetos binários, n é um limite inferior exato. Portanto, resta considerar a questão do limite inferior para o caso em que $|\Sigma| \geq 3$.

Assim, considerando que $|\Sigma| \geq 3$ então vamos estabelecer um limite inferior para o número de comparações efetuadas pelos algoritmos de reconhecimento de padrões em textos seguindo a abordagem proposta em [GG91].

Nesta abordagem adotou-se um modelo de computação que se baseia em uma memória de acesso aleatório com custo de acesso uniforme. Além disso, adotou-se que o texto está armazenado em um dispositivo que permite o acesso aleatório e que este texto somente pode ser acessado através de um servidor que não é controlado pelo algoritmo a ser analisado. O servidor aceita consultas (comparações) apenas do tipo "$t_k = p_j$?" ou "$t_k = t_i$?" e fornece a resposta a este tipo de consulta em uma unidade de tempo.

Neste modelo, um algoritmo pode ter qualquer conhecimento sobre o padrãc; mas as informações sobre o texto somente podem ser obtidas através das comparações estabelecidas acima.

Nesta análise foram considerados dois tipos de algoritmos: *on-line* e *off-line*. Um algoritmo é dito *on-line* se, em cada instante, as comparações (consultas) estão restritas a uma parte do texto de comprimento m. Além disso, somente podemos efetuar comparações sobre a parte do texto dada por $t_{k+1} \ldots t_{k+m}$ se o algoritmo já determinou todas as ocorrências do padrão nas posições i, onde $i : 1 \leq i \leq k$. Por outro lado, um algoritmo é dito *off-line* se nenhuma destas restrições é imposta.

Agora, seja $C(n, m)$ o número de comparações efetuadas, no pior caso, por um algoritmo *on-line* que obtém todas as ocorrências de um padrão de comprimento m num texto de comprimento n. Se o algoritmo é *off-line* então indicaremos o número de comparações efetuadas por $C_{\text{off}}(n, m)$ e se o algoritmo obtém apenas a primeira ocorrência do padrão no texto então o número de comparações efetuadas por algoritmos *on-line* e *off-line* será indicado respectivamente por $C^1(n, m)$ e $C^1_{\text{off}}(n, m)$. Vale lembrar que estamos considerando que $|\Sigma| \geq 3$.

Teorema 1.5 Se $0 < m \leq 2$ então

$$C(n, m) \geq n;$$

se $m \geq 3$ então, para m ímpar

$$C(n, m) \geq n + \left\lfloor \frac{2(n-m)}{m+3} \right\rfloor$$

e, para m par

$$C(n, m) \geq n + \left\lfloor \frac{2(n-m)}{m+4} \right\rfloor.$$

Demonstração : Nesta demonstração utilizaremos a estratégia do adversário ([Baa91]), ou seja, dado um algoritmo *on-line* que obtém todas as ocorrências de um padrão num texto então, para cada caso estabelecido acima, vamos construir um texto t e mostrar que para este texto o número de comparações efetuadas pelo algoritmo não é menor do que o limite inferior estabelecido para aquele caso.

Inicialmente, para $0 < m \leq 2$, seja $t := a^n$. Se $m = 1$ então considere $p := a$ e se $m = 2$ considere $p := aa$. Assim, em ambos os casos, o algoritmo deve efetuar, no mínimo, n comparações.

Agora, considere $m \geq 3$. Neste caso, a demonstração será obtida por indução no comprimento do texto. Inicialmente, vamos considerar o caso em que m é ímpar. Assim, para m ímpar, seja $p := a^l b a^l$, onde $l := \frac{m-1}{2} \geq 1$. Como $n \geq m$ então, para $n = m$, seja $t := p$. Portanto, o algoritmo deve efetuar m comparações para concluir que o padrão ocorre na posição 1 do texto. Isto implica que, neste caso, o limite inferior é válido.

Agora, por hipótese de indução, suponha que t' é um texto de comprimento n_0, que tem p como sufixo e para o qual o limite inferior é válido, ou seja,

$$C(n_0, m) \geq n_0 + \left\lfloor \frac{2(n_0 - m)}{m+3} \right\rfloor .$$

A partir de t' vamos construir um texto t de comprimento n_1, com $n_1 > n_0$, e vamos mostrar que o limite inferior também é válido para t.

Seja t um texto de comprimento n_1, com $n_1 > n_0$, tal que t' seja um prefixo de t, isto é, $t_1 \ldots t_{n_0} = t'_1 \ldots t'_{n_0}$. Os outros caracteres de t, dados por $t_{n_0+1} \ldots t_{n_1}$, serão convenientemente estabelecidos à medida que o algoritmo efetuar as comparações (consultas).

Assim, o algoritmo deve efetuar $C(n_0, m)$ comparações para obter todas as ocorrências de p em $t_1 \ldots t_{n_0}$. Agora, falta estabelecer o número mínimo de comparações que devem ser efetuadas para obter as ocorrências de p considerando os caracteres adicionados $(t_{n_0+1} \ldots t_{n_1})$. Inicialmente, observe que $n_0 - 2l$ é a posição mais à direita em $t_1 \ldots t_{n_0}$ onde o padrão pode ocorrer. Agora, se considerarmos os caracteres adicionados $t_{n_0+1}, t_{n_0+2} \ldots t_{n_1}$ então torna-se possível a ocorrência do padrão nas posições $n_0 - 2l + 1, n_0 - 2l + 2, \ldots, n_1 - 2l$. Portanto, falta estabelecer o número mínimo de comparações que devem ser efetuadas para obter todas as ocorrências de p em $t_{n_0-2l+1} \ldots t_{n_1}$.

Como p é um sufixo de t' e t' é um prefixo de t de comprimento n_0

então p é um sufixo de $t_1 \ldots t_{n_0}$, ou seja, $t_{n_0-2l} \ldots t_{n_0} = p = a^l b a^l$ e portanto, há uma ocorrência do padrão na posição $n_0 - 2l$ do texto. Em particular, isto implica que $t_{n_0-l+1} \ldots t_{n_0} = a^l$ e $t_{n_0-l} = b$. Como b não ocorre no prefixo do padrão de comprimento l então, após obter esta ocorrência, o algoritmo pode deslocar o padrão de modo que $t_{n_0-l} = b$ não fique alinhado com nenhum caractere do prefixo do padrão de comprimento l, ou seja, o padrão pode ser deslocado de modo que p_1 fique alinhado com t_{n_0-l+1}. Assim, após este deslocamento, o algoritmo, que é *on-line*, pode consultar os caracteres do texto que estão entre as posições $n_0 - l + 1$ e $n_0 + l + 1$, inclusive. Visto que estamos interessados em estabelecer o número mínimo de comparações efetuadas pelo algoritmo e visto que o algoritmo já sabe que $t_{n_0-l+1} \ldots t_{n_0} = a^l$ então podemos supor que o algoritmo não comparará novamente os caracteres $t_{n_0-l+1} \ldots t_{n_0}$, ou seja, as comparações ficarão restritas a $t_{n_0+1} \ldots t_{n_0+l+1}$.

Agora, vamos estabelecer os valores de $t_{n_0+1} \ldots t_{n_1}$. Primeiro, vamos fazer $t_{n_0+3} \ldots t_{n_0+l+1} := a^{l-1}$ (note que se $l = 1$ então $t_{n_0+3} \ldots t_{n_0+l+1}$ é a seqüência vazia). Assim, fica faltando definir apenas os valores de t_{n_0+1}, t_{n_0+2} e de n_1. Observe que, de acordo com as considerações acima, n_1 deve ser tal que $n_1 \geq n_0 + l + 1$. Portanto, dependendo do valor de n_1, também devemos definir os valores de $t_{n_0+l+2} \ldots t_{n_1}$.

Como as consultas efetuadas pelo algoritmo estão restritas à parte do texto dada por $t_{n_0+1} \ldots t_{n_0+l+1}$ então todas estas consultas já podem ser precisamente respondidas exceto quando as consultas envolverem os caracteres t_{n_0+1} e t_{n_0+2}, pois os valores destes caracteres ainda não foram definidos. Para defini-los vamos analisar como o algoritmo poderia consultar estes caracteres pela primeira vez. A partir desta análise, definiremos também os valores de n_1 e de $t_{n_0+l+2} \ldots t_{n_1}$.

Inicialmente, vale lembrar que os tipos de consultas permitidas são "$t_i = p_j$?" ou "$t_i = t_k$?", para i, j e k válidos. Assim, visto que o padrão é conhecido e que t_k, para $k : n_0+3 \leq k \leq n_0+l+1$, já está definido então uma consulta envolvendo t_{n_0+1} ou t_{n_0+2} relaciona um destes caracteres com um caractere já conhecido, exceto no caso em que a consulta é "$t_{n_0+1} = t_{n_0+2}$?" (note que consultas do tipo "$t_{n_0+1} = t_{n_0+1}$?" não são consideradas, pois tais consultas são no mínimo inúteis). Desta forma, podemos dividir a análise em dois casos:

Caso 1: A primeira consulta envolvendo t_{n_0+1} ou t_{n_0+2} é "$t_{n_0+1} = b$?" ou "$t_{n_0+2} = a$?" ou equivalente a uma destas duas (por exemplo, "$t_{n_0+1} = t_k$?"

com $t_k = b$). Neste caso, estabelecemos $t_{n_0+1} := a$ e $t_{n_0+2} := b$. Além disso, tomamos $n_1 := n_0 + l + 2$ e $t_{n_1} := a$. Desta forma, a resposta a qualquer das duas consultas é *não*. Note que pelos valores estabelecidos para o texto t o padrão não ocorre na posição $n_0 - l + 1$ do texto. Porém,

$$\begin{aligned} t_{n_0-l+2} \ldots t_{n_0} &= a^{l-1} \\ t_{n_0+1} &= a \\ t_{n_0+2} &= b \\ t_{n_0+3} \ldots t_{n_1} &= a^l, \end{aligned}$$

o que implica que $t_{n_0-l+2} \ldots t_{n_1} = a^l b a^l = p$. Portanto, há uma ocorrência do padrão na posição $n_0 - l + 2$ do texto. Para obter esta ocorrência, o algoritmo poderia utilizar o fato de que ele já sabe que $t_{n_0-l+2} \ldots t_{n_0} = a^{l-1}$ e assim, seria necessário consultar apenas $t_{n_0+1} \ldots t_{n_1}$, ou seja, seriam necessárias pelo menos $n_1 - n_0 = l + 2$ comparações. Isto implica que para obter todas as ocorrências do padrão em t, o algoritmo deve efetuar $C(n_0, m)$ comparações para obter todas as ocorrências em $t_1 \ldots t_{n_0}$ mais uma das comparações estabelecidas na hipótese deste caso e mais pelo menos as $l+2$ comparações descritas acima. Assim,

$$\begin{aligned} C(n_1, m) &\geq C(n_0, m) + 1 + l + 2 \\ &\overset{\text{H.I.}}{\geq} n_0 + \left\lfloor \frac{2(n_0 - m)}{m+3} \right\rfloor + l + 3 \\ &\overset{n_1 = n_0 + l + 2}{=} n_1 + \left\lfloor \frac{2(n_0 - m)}{m+3} \right\rfloor + 1 \\ &= n_1 + \left\lfloor \frac{2(n_0 - m)}{m+3} + 1 \right\rfloor. \end{aligned}$$

Como $n_1 = n_0 + l + 2$ e como $m = 2l + 1$ então podemos concluir que

$$C(n_1, m) \geq n_1 + \left\lfloor \frac{2(n_1 - m)}{m+3} \right\rfloor.$$

Assim, temos que, neste caso, o limite inferior é válido. Além disso, temos que p é um sufixo de t e portanto, após a realização do passo de indução, as hipóteses iniciais permanecem válidas.

Caso 2: A primeira consulta envolvendo t_{n_0+1} ou t_{n_0+2} não se enquadra no *caso 1*. Observe que este caso inclui a consulta "$t_{n_0+1} = t_{n_0+2}$?" e

também consultas do tipo "$t_{n_0+1} = t_k$?", com $t_k \neq b$ ou "$t_{n_0+2} = t_k$?", com $t_k \neq a$. Neste caso, estabelecemos $t_{n_0+1} := b$ e $t_{n_0+2} := a$. Além disso, tomamos $n_1 := n_0 + l + 1$, o que implica que o valor de t_{n_1} já está estabelecido ($t_{n_1} = t_{n_0+l+1} = a$). Desta forma, a resposta a qualquer das consultas é *não* e, pelos valores estabelecidos para o texto t, temos que

$$\begin{aligned} t_{n_0-l+1} \ldots t_{n_0} &= a^l \\ t_{n_0+1} &= b \\ t_{n_0+2} &= a \\ t_{n_0+3} \ldots t_{n_1} &= a^{l-1}, \end{aligned}$$

o que implica que $t_{n_0-l+1} \ldots t_{n_1} = a^l b a^l{=}p$. Portanto, há uma ocorrência do padrão na posição $n_0{-}l{+}1$ do texto e para obter esta ocorrência, o algoritmo poderia utilizar o fato de que ele já sabe que $t_{n_0-l+1} \ldots t_{n_0} = a^l$ e assim, seria necessário consultar apenas $t_{n_0+1} \ldots t_{n_1}$, ou seja, seriam necessárias pelo menos $n_1 - n_0 = l + 1$ comparações. Isto implica que, de maneira análoga ao caso anterior, para obter todas as ocorrências do padrão em t, o algoritmo deve efetuar $C(n_1, m)$ comparações, onde

$$\begin{aligned} C(n_1,m) &\geq C(n_0,m) + 1 + l + 1 \\ &\overset{\text{H.I.}}{\geq} n_0 + \left\lfloor \frac{2(n_0 - m)}{m+3} \right\rfloor + l + 2. \end{aligned}$$

Como $n_1 = n_0 + l + 1$ e como $m = 2l + 1$ então podemos efetuar um desenvolvimento análogo ao do caso anterior e obtemos que

$$C(n_1,m) \geq n_1 + \left\lfloor \frac{2(n_1 - m)}{m+3} \right\rfloor.$$

Assim, também neste caso, o limite inferior é válido. Além disso, temos que p é um sufixo de t e portanto, após a realização do passo de indução, as hipóteses iniciais permanecem válidas.

Para concluir a demonstração, devemos analisar o caso em que m é par. Para isto, basta tomar $p := a^l b a^{l+1}$ e efetuar um desenvolvimento semelhante ao realizado para o caso m ímpar observando que, neste caso, os textos obtidos possuirão um caractere a mais do que os textos determinados para m ímpar. □

Além deste resultado, em [GG91] são estabelecidos limites inferiores para $C^1(n,m)$ (número de comparações efetuadas por algoritmos *on-line* que

obtêm apenas a primeira ocorrência do padrão no texto) e para $C_{\text{off}}(n, m)$ (número de comparações efetuadas por algoritmos *off-line*). A seguir apresentamos sem demonstração estes resultados. O leitor interessado pode obter tais demonstrações em [GG91].

Teorema 1.6 Se o algoritmo pode efetuar apenas consultas do tipo "$t_k = p_j$?" temos que se $0 < m \leq 3$ então

$$C^1(n, m) \geq n;$$

se $m \geq 4$ então

$$C^1(n, m) \geq n + \left\lfloor \frac{n-m}{m} \right\rfloor.$$

□

Teorema 1.7 Se $0 < m \leq 2$ então

$$C_{\text{off}}(n, m) \geq n;$$

se $m \geq 3$ então

$$C_{\text{off}}(n, m) \geq n + \left\lfloor \frac{n}{2m} \right\rfloor.$$

□

Para encerrar esta seção vale comentar que Galil e Giancarlo elaboraram um outro estudo sobre a complexidade de algoritmos para reconhecimento de padrões em textos. Neste estudo, apresentado em [GG92], eles analisam a questão do limite superior para o número de comparações efetuadas. Ao final, eles utilizam os resultados obtidos nas duas análises para estimar o valor da constante de proporcionalidade para o número de comparações efetuadas. Além disso, eles também listam algumas questões em aberto relacionadas aos limites inferior e superior para algoritmos de reconhecimento de padrões em textos. Em particular, dentre estas questões temos que não se conhece um limite inferior para $C^1_{\text{off}}(n, m)$.

1.4 Periodicidade em Seqüências de Caracteres

O objetivo desta seção é estabelecer alguns resultados sobre periodicidade em seqüências de caracteres, sendo que apresentaremos apenas aqueles resultados que serão necessários nos próximos capítulos. O leitor interessado numa discussão mais detalhada sobre este assunto pode obtê-la em [GO81].

Definição 1.8 Uma *borda* de α é um prefixo próprio de α que também é sufixo de α, isto é, β é uma borda de α se existem seqüências não vazias γ e δ tais que $\beta\gamma = \alpha = \delta\beta$. Além disso, para $0 < j < |\alpha|$, se $\alpha_1 \ldots \alpha_i$ é uma borda de $\alpha_1 \ldots \alpha_j$ e $\alpha_{i+1} \neq \alpha_{j+1}$ então dizemos que $\alpha_1 \ldots \alpha_i$ é uma *borda disjunta* de $\alpha_1 \ldots \alpha_j$.

Definição 1.9 Seja α uma seqüência de caracteres tal que $|\alpha| > 1$. Sejam β e γ seqüências não vazias tais que $\alpha = \beta\gamma$. Então a seqüência $\gamma\beta$ é denominada uma *rotação* de α.

Definição 1.10 Um inteiro positivo $q \leq |\alpha|$ é um *período* de α se

$$\alpha_i = \alpha_{i+q} \text{ para todo } i : 1 \leq i \leq |\alpha| - q.$$

Proposição 1.11 Um inteiro positivo q é um período de α se, e somente se, uma das propriedades abaixo é satisfeita:

(i) existem β e γ tais que $|\beta| = q$ e $\alpha = \beta^i\gamma$ para algum $i \geq 1$, sendo que γ é um prefixo próprio de β;

(ii) existem θ, θ' e ψ tais que $|\theta| = q = |\theta'|$ e $\theta\psi = \alpha = \psi\theta'$.

Demonstração: Vamos mostrar que a definição 1.10 implica (i), que por sua vez implica (ii) e este finalmente implica a definição 1.10, ou seja, vamos mostrar que a definição 1.10, (i) e (ii) são equivalentes.

Seja q um período de α. Para mostrar que a definição 1.10 implica (i), sejam k e r tais que $k := \left\lfloor \frac{|\alpha|}{q} \right\rfloor$ e $r := |\alpha| \bmod q$. Assim, $r < q$ e $k \geq 1$ pois, $q \leq |\alpha|$. Agora, sejam $\beta := \alpha_1 \ldots \alpha_q$ e $\gamma := \alpha_1 \ldots \alpha_r$. Então, $|\beta| = q$ e γ é um prefixo próprio de β. Além disso, pela definição 1.10, temos que $\alpha_i = \alpha_{i+q}$, para todo $i : 1 \leq i \leq |\alpha| - q$. Daí, $\alpha = \beta^k\gamma$, o que implica que (i) é satisfeito.

Suponhamos agora que (i) é válido e vamos mostrar a veracidade de (ii). Como γ é um prefixo próprio de β então existe β', com $|\beta'| > 0$, tal que $\beta = \gamma\beta'$. Isto implica que $\alpha = (\gamma\beta')^k\gamma$. Daí, sejam $\theta := \gamma\beta'$, $\theta' := \beta'\gamma$ e $\psi := (\gamma\beta')^{(k-1)}\gamma$. Então $|\theta| = q = |\theta'|$ e $\theta\psi = \alpha = \psi\theta'$, o que mostra que (i) implica (ii).

Finalmente, vamos mostrar que (ii) implica a definição 1.10. Por (ii) existem θ, θ' e ψ tais que

$$\theta\psi = \alpha = \psi\theta' \tag{1.2}$$

onde, $|\theta| = q = |\theta'|$. Da primeira igualdade em (1.2) temos que

$$\psi_i = \alpha_{i+q} \qquad \forall i : 1 \leq i \leq |\alpha| - q.$$

Da segunda igualdade em (1.2) temos que

$$\alpha_i = \psi_i \qquad \forall i : 1 \leq i \leq |\alpha| - q.$$

Assim, $\alpha_i = \alpha_{i+q}$ para $i : 1 \leq i \leq |\alpha| - q$. □

Corolário 1.12 Um inteiro positivo q é um período de α se, e somente se, existe uma borda de α cujo comprimento é $|\alpha| - q$. □

Agora, observe que a definição 1.10 pode ser reescrita da seguinte forma: um inteiro positivo q é um período de α se $\alpha_{i-q} = \alpha_i$ para todo $i : q+1 \leq i \leq |\alpha|$. Isto equivale dizer que ao invés de analisarmos o padrão da esquerda para a direita podemos fazê-lo em sentido reverso, isto é, da direita para a esquerda. Com o intuito de ressaltar esta característica, vamos a seguir estabelecer a forma equivalente do item (i) da proposição 1.11.

Proposição 1.13 Um inteiro positivo q é um período de α se, e somente se, existe ω tal que $|\omega| = q$ e $\alpha = \rho\omega^i$ onde, $i \geq 1$ e ρ é um sufixo próprio de ω. □

É importante observar que as seqüências β e ω das proposições 1.11 e 1.13 respectivamente, podem não ser iguais. Por exemplo, considere $\alpha = abcabcab$. Então $q = 3$ é um período de α. Neste caso, $\beta = abc$, $\gamma = ab$, $\omega = cab$ e $\rho = ab$. Na verdade, demonstra-se facilmente que se $|\gamma| > 0$ então β é uma rotação de ω.

Definição 1.14 Uma seqüência α é *periódica* se existe um período q de α tal que $q \leq \frac{|\alpha|}{2}$, ou seja, se existe β tal que $\alpha = \beta^i\gamma$ para algum $i \geq 2$, sendo que γ é um prefixo próprio de β. Além disso, se $|\gamma| = 0$ então dizemos que α é *periódica perfeita.*

É claro que, na definição acima, também podemos considerar a ordem reversa e, neste caso, dizemos que α é periódica se existe ω tal que $\alpha = \rho\omega^i$ para algum $i \geq 2$, sendo que ρ é um sufixo próprio de ω. Além disso, se $|\rho| = 0$ então α é periódica perfeita.

Agora, a partir da definição 1.14 e do corolário 1.12 podemos estabelecer a seguinte afirmação:

Corolário 1.15 Uma seqüência α é periódica se, e somente se, existe uma borda b de α tal que $|b| \geq \frac{|\alpha|}{2}$. □

Definição 1.16 Seja α uma seqüência periódica e seja q um período de α tal que $q \leq \frac{|\alpha|}{2}$. Então o prefixo de α cujo comprimento é igual a q é denominado *elemento periódico de α à esquerda.* Analogamente, o sufixo de α cujo comprimento é igual a q é denominado *elemento periódico de α à direita.*

Para ilustrar as várias definições estabelecidas acima considere a seguinte tabela:

α	Períodos	Periódica?	Elementos Periódicos	
			à esq.	à dir.
abababa	2,4,6,8 e 9	Sim	*ab* e *abab*	*ba* e *baba*
abcdxyzwab	8 e 10	Não	-	-
aaaaaa	1,2,3,4,5 e 6	Sim	*a*, *aa* e *aaa*	*a*, *aa* e *aaa*
abcdabc	4 e 7	Não	-	-
ababab	2, 4 e 6	Sim	*ab*	*ab*

Teorema 1.17 Se q_1 e q_2 são períodos de α e $q_1 + q_2 \leq |\alpha| + mdc(q_1, q_2)$ então $mdc(q_1, q_2)$ também é um período de α.

Demonstração : Por indução em $q_1 + q_2$. Se $q_1 = q_2$ então a asserção é trivialmente verdadeira. Se $q_1 \neq q_2$ então ajuste a notação, permutando q_1 e q_2 se necessário, de forma que $q_1 < q_2$.

Sejam $r := q_2 - q_1$ e $d := mdc(q_1, q_2) = mdc(q_1, r)$. Por hipótese,

$$q_1 + q_2 \leq |\alpha| + d$$

e, além disso,

$$d \leq q_1 < q_2.$$

Portanto,

$$q_1 < |\alpha|.$$

Assim, sejam θ e ψ tais que $|\theta| = q_1$ e

$$\alpha = \theta\psi. \tag{1.3}$$

Vamos agora mostrar que q_1 e r são ambos períodos de ψ e que satisfazem

$$q_1 + r \leq |\psi| + d. \tag{1.4}$$

Para mostrar a validade de (1.4), observe que

$$q_1 + r = q_2 \leq |\alpha| + d - q_1 = |\psi| + d.$$

Para mostrar que tanto $q_1 \leq |\psi|$ quanto $r \leq |\psi|$, basta usar (1.4), lembrando que $d \leq q_1$ e $d \leq r$.

Certamente, $\psi_i = \psi_{i+q_1}$, para todo $i : 1 \leq i \leq |\psi| - q_1$, pois ψ é um sufixo de α e q_1 é um período de α. Além disso, para todo $i : 1 \leq i \leq |\psi| - r$, temos que

$$\psi_i = \alpha_{i+q_1} = \alpha_i = \alpha_{i+q_2} = \psi_{i+r},$$

onde a primeira e a última igualdades seguem de (1.3) e as demais do fato que tanto q_1 quanto q_2 são períodos de α. Portanto, de fato, q_1 e r são ambos períodos de ψ e satisfazem (1.4). Daí, por hipótese de indução, d é um período de ψ.

Vamos mostrar agora que d é também um período de α. Para isto, basta mostrarmos que para todo $i : 1 \leq i \leq |\alpha|$

$$\alpha_i = \alpha_{f(i)},$$

onde

$$f(i) := 1 + (i - 1) \bmod d.$$

Consideremos inicialmente o caso em que $i > q_1$. Então

$$\alpha_i = \psi_{i-q_1} = \psi_{f(i-q_1)} = \psi_{f(i)},$$

onde a igualdade entre $f(i-q_1)$ e $f(i)$ segue do fato que d divide q_1 e a igualdade entre ψ_{i-q_1} e $\psi_{f(i-q_1)}$ segue do fato que d é um período de ψ.

Consideremos agora o caso em que $i \leq q_1$. Visto que $q_1 \leq |\psi|$ então $i + q_1 \leq |\alpha|$ e portanto,

$$\alpha_i = \alpha_{i+q_1} = \psi_i = \psi_{f(i)}.$$

Assim, em ambos os casos concluímos que $\alpha_i = \psi_{f(i)}$ e finalmente,

$$\psi_{f(i)} = \alpha_{f(i)+q_1} = \alpha_{f(i)}.$$

Logo, $\alpha_i = \alpha_{f(i)}$ o que nos leva a concluir que, de fato, d é período de α. □

Corolário 1.18 Se q_1 e q_2 são períodos de α tais que $q_1 \neq q_2$ e $q_1 + q_2 \leq |\alpha| + mdc(q_1, q_2)$ então $|q_2 - q_1|$ também é um período de α. □

Corolário 1.19 Se q_1 e q_2 são períodos de α tais que $q_1 + q_2 \leq |\alpha| + mdc(q_1, q_2)$ e q_1 é o menor período de α então q_1 divide q_2. □

Vale ressaltar que o teorema 1.17 foi apresentado em [FW65] e é um refinamento de um resultado estabelecido em [LS62] cuja hipótese era de que $q_1+q_2 \leq |\alpha|$. Na realidade, o limite para q_1+q_2, imposto pelo teorema 1.17, é o limite exato, pois conforme mostra o exemplo a seguir, q_1 e q_2 são períodos de α, $q_1 + q_2 = |\alpha| + mdc(q_1, q_2) + 1$ e $mdc(q_1, q_2)$ não é um período de α. Considere $\alpha = a\ a\ b\ a\ a$. Então $q_1 = 3$ e $q_2 = 4$ são períodos de α mas $mdc(3, 4) = 1$ não é um período de α.

Proposição 1.20 Sejam ψ e γ seqüências não vazias tais que $\alpha = \psi\gamma$ e seja $\beta := \gamma\psi$ (β é uma rotação de α). Se $\alpha = \beta$ então o menor período de α divide $|\psi|$ e $|\gamma|$ e portanto α é periódica perfeita.

Demonstração : Por indução em $|\alpha| = |\psi| + |\gamma|$. Inicialmente, considere que $|\psi| = |\gamma|$. Assim, se $\psi\gamma = \gamma\psi$ então $\psi = \gamma$ o que implica que $\alpha = \psi^2$ e portanto, $|\psi|$ é um período de α. Daí, pelo corolário 1.19, o menor período de α divide $|\psi| = |\gamma|$.

Agora, sem perda de generalidade, podemos considerar que $|\gamma| < |\psi|$. Suponha por hipótese de indução que, para todo ψ' tal que $|\psi'| < |\psi|$, a asserção seja verdadeira. Assim, se $\psi\gamma = \gamma\psi$ então γ é um prefixo próprio de ψ, o que implica que existe ψ', com $|\psi'| > 0$, tal que $\psi = \gamma\psi'$. Portanto, $\gamma\psi'\gamma = \gamma\gamma\psi'$ e daí, $\psi'\gamma = \gamma\psi'$.

Seja q_1 o menor período de $\gamma\psi'$. Pela hipótese de indução q_1 divide $|\gamma|$ e $|\psi'|$ o que implica que existe ρ, com $|\rho| = q_1$, tal que $\gamma = \rho^j$ e $\psi' = \rho^k$, onde $j \geq 1$ e $k \geq 1$. Assim,

$$\alpha = \gamma\gamma\psi' = \rho^{2j+k},$$

o que implica que α é periódica perfeita e q_1 é um período de α.

Agora, seja q_2 o menor período de α. Então $q_2 \leq q_1$.

Por outro lado, q_1 divide $|\gamma|$ e $|\gamma| < |\psi|$. Daí,

$$q_2 \leq q_1 \leq |\gamma| < |\psi|$$

e portanto $q_1 + q_2 < |\alpha|$. Pelo corolário 1.19, q_2 divide q_1. Assim, q_2 divide $|\gamma|$ e $|\psi'|$ e conseqüentemente q_2 divide $|\gamma| + |\psi'| = |\psi|$. Logo, o menor período de α divide $|\gamma|$ e $|\psi|$. □

Proposição 1.21 Sejam a e $b \in \Sigma$ e α, β e γ cadeias de caracteres tais que $\beta a\gamma = \alpha = \gamma b\beta$. Então $a = b$.

Demonstração : Por indução em $|\alpha|$. Sem perda de generalidade, podemos supor que $|\beta| \leq |\gamma|$. Se $|\beta| = |\gamma|$ então a asserção é trivialmente verdadeira. Suponhamos então que $|\beta| < |\gamma|$.

Seja $q := |\beta a| = |b\beta| \leq |\gamma|$. Por hipótese, γ é uma borda de α e além disso, $|\gamma| = |\alpha| - q$. Assim, pelo corolário 1.12, q é um período de α. Dado que $q \leq |\gamma|$, então q é também um período de γ. Novamente, aplicando o corolário 1.12, temos que γ tem uma borda, digamos δ, de comprimento $|\gamma| - q$. Pela hipótese, o prefixo de γ de comprimento q é βa e o sufixo de γ de comprimento q é $b\beta$. Portanto,

$$\beta a\delta = \gamma = \delta b\beta.$$

Logo, por hipótese de indução, $a = b$. □

Capítulo 2

O Algoritmo de Knuth, Morris e Pratt (KMP)

Como foi salientado no capítulo 1, existem diversos problemas, nas mais variadas áreas, cujas soluções dependem fundamentalmente de métodos para o reconhecimento de padrões. Na maioria destes casos, o volume de dados a ser processado é relativamente grande e portanto, a eficiência do método utilizado para o reconhecimento de padrões é um fator de extrema importância. Além disso, em algumas situações, é conveniente que durante o processo de reconhecimento do padrão não ocorram retrocessos no texto pois, se considerarmos as operações de gerenciamento de *buffer*, tais retrocessos podem significar um aumento substancial no tempo de execução do algoritmo.

Motivados por estas questões, Knuth, Morris e Pratt [KMP77] propuseram um algoritmo para o reconhecimento de padrões em texto que, no pior caso, efetua $O(m+n)$ comparações sem realizar retrocessos no texto. Além disso, cada caractere do texto é comparado, no máximo, $O(\log m)$ vezes e, supondo que o texto é armazenado externamente, este algoritmo utiliza apenas $O(m)$ posições de memória. A história do desenvolvimento deste algoritmo é bastante interessante valendo a pena lê-la em [KMP77].

A seguir, na seção 2.1 faremos uma descrição do autômato utilizado pelo algoritmo KMP para reconhecer um padrão no texto, enquanto na seção 2.2 apresentaremos como este autômato pode ser construído. Na seção 2.3 analisaremos a complexidade do algoritmo KMP e na seção 2.4 descreveremos algumas variações deste algoritmo.

2.1 O Autômato Reconhecedor de Padrão

O algoritmo Ingênuo descrito na seção 1.2, possui o inconveniente de realizar, no pior caso, um número de comparações que é da ordem do produto entre o comprimento do texto e o comprimento do padrão. Podemos notar que várias comparações efetuadas por aquele algoritmo poderiam ser evitadas, se nos "lembrássemos" dos caracteres do padrão que foram comparados com o texto até o ponto onde ocorreu a diferença. Isto permitiria, em alguns casos, deslocar o padrão mais de uma posição em relação ao texto. É exatamente esta a idéia básica do algoritmo KMP, e para concretizá-la, utilizaremos um autômato, que denominaremos *autômato reconhecedor de padrão.* Este autômato será construído a partir de uma análise do padrão e, posteriormente, o texto será processado através dele. Durante este processamento, a cada caractere do texto analisado haverá uma mudança de estado no autômato e, no caso de ser verificada uma diferença entre o texto e o padrão então o autômato "deslocará" adequadamente o padrão em relação ao texto. Este processamento prosseguirá até que se atinja ou o estado final do autômato, o que indica que uma ocorrência do padrão no texto foi obtida, ou até que o texto se encerre. Portanto, a parte central do algoritmo KMP é a construção do autômato reconhecedor de padrão. Antes de descrever como este autômato será construído vamos estabelecer algumas definições e ressaltar algumas características particulares deste autômato.

Definição 2.1 Um *autômato finito determinístico* é dado pela quíntupla $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ onde Q é um conjunto finito de *estados*, Σ é um *alfabeto* finito, $q_0 \in Q$ é o *estado inicial*, $F \subseteq Q$ é o conjunto de *estados finais* e $\delta : Q \times \Sigma \mapsto Q$ é a *função de transição.*

A entrada de um autômato é uma palavra em Σ^* que está armazenada em uma fita de entrada. Inicialmente, o autômato está no estado q_0 e a cabeça de leitura está sobre o primeiro caractere armazenado na fita, ou seja, o caractere corrente no texto é aquele que está na primeira posição deste texto. Se, em determinado instante, o autômato está no estado q e a cabeça de leitura está sobre o caractere a do texto então ocorre uma transição que modifica o estado corrente do autômato para $\delta(q, a)$ e avança a cabeça de leitura para o caractere seguinte na fita. Um processamento efetuado pelo autômato equivale à execução de uma seqüência de transições, ou seja, se a fita contém a cadeia de caracteres $\alpha = \alpha_1 \ldots \alpha_n$ então o processamento de α

pelo autômato $\mathcal{A}$ corresponde ao posicionamento da cabeça de leitura à direita de α_n e à entrada do autômato no estado $\delta(\ldots\delta((\delta(q_0,\alpha_1),\alpha_2)\ldots,\alpha_n)$; se este estado pertence a F, isto é, se ele for um estado final, então $\mathcal{A}$ *aceita* α, caso contrário $\mathcal{A}$ *rejeita* α. Para facilitar a descrição de um processamento, vamos estender a função de transição δ para que ela opere sobre $Q \times \Sigma^*$ aceitando como entrada o par (q,α) onde $\alpha \in \Sigma^*$. Assim, quando utilizarmos o termo $\delta(q,\alpha)$ estaremos nos referindo ao estado alcançado pelo autômato após o processamento de α a partir do estado q. Esta extensão de δ pode então ser definida formalmente da seguinte forma:

$$\delta(q,\alpha) := \begin{cases} q & \text{se } \alpha = \epsilon \\ \delta(\delta(q,a),\beta) & \text{se } \alpha = a\beta,\ a \in \Sigma \text{ e } \beta \in \Sigma^*. \end{cases}$$

Para representar um autômato $\mathcal{A}$ utilizaremos um grafo orientado, denominado *diagrama de estados*, no qual os vértices representarão os estados do autômato e as arestas representarão as transições. Se, no autômato, $\delta(q,a) = s$, onde $a \in \Sigma$, então, no grafo, haverá uma aresta do vértice q para o vértice s rotulada com o caractere a. Além disso, no grafo, o estado inicial será representado pelo vértice que é o destino de uma aresta cuja origem não é nenhum outro vértice. De modo análogo, um vértice que é origem de uma aresta que não se destina a nenhum outro vértice indica um estado final. As arestas que são utilizadas para indicar o estado inicial e os estados finais não serão rotuladas. Supondo $\Sigma = \{a,b,c\}$ então a figura 2.1 contém o diagrama de estados representando o autômato reconhecedor de $p = abcaabcaba$. Este autômato aceita cadeias em Σ^*p, isto é, para cada ocorrência de p no texto fornecido ao autômato o estado final será alcançado.

Uma outra forma de representar um autômato é através de uma tabela bidimensional onde os estados são indicados nas linhas e os caracteres que compõem o alfabeto são colocados nas colunas. Nesta tabela, cada entrada *(l,c)* contém o valor resultante da transição a partir do estado l quando o caractere corrente é c. Assim, as figuras 2.1 e 2.2 descrevem o mesmo autômato.

Na introdução deste capítulo, foi dito que o algoritmo KMP soluciona o problema de reconhecimento de padrões em $O(m+n)$ unidades de tempo utilizando $O(m)$ posições de memória. Entretanto, podemos notar claramente que se o algoritmo representar o autômato exatamente como descrito acima, ele não poderá utilizar apenas $O(m)$ posições de memória e mais, ele

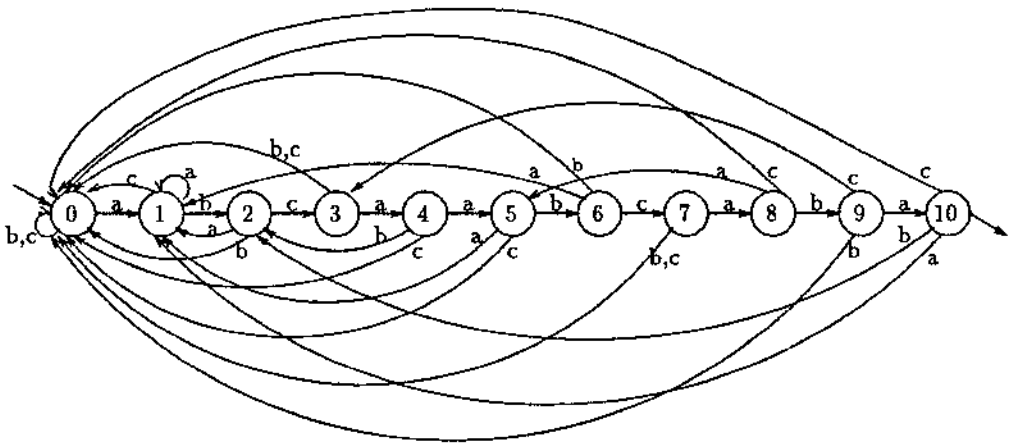


Figura 2.1: Diagrama de estados do autômato reconhecedor de *abcaabcaba*.

δ	a	b	c
0	1	0	0
1	1	2	0
2	1	0	3
3	4	0	0
4	5	2	0
5	1	6	0
6	1	0	7
7	8	0	0
8	5	9	0
9	10	0	3
10	1	2	0

Figura 2.2: Tabela representando o autômato reconhecedor de *abcaabcaba*.

não gastará apenas $O(m+n)$ unidades de tempo pois, o número de posições na tabela, e conseqüentemente o tempo para preenchê-las, é dependente do tamanho do alfabeto. Para contornar este problema, o algoritmo adota uma *representação abreviada* para o autômato reconhecedor de padrão. Basicamente, esta representação consiste em estabelecer, independentemente do tamanho do alfabeto, um valor fixo para o número de transições a partir de um estado, o que é feito da seguinte forma: analisando a figura 2.1 podemos perceber que, naquele autômato, existem dois tipos de transições, as que levam a um estado mais distante do estado inicial e as que reduzem ou mantêm esta distância[1]. Na realidade, a partir de cada estado, existe apenas uma transição que aumenta esta distância pois, seguindo a ordem de percurso, se concatenarmos os rótulos das arestas do menor caminho orientado entre o estado inicial e um estado q obteremos exatamente o prefixo do padrão que foi reconhecido até o estado q. Este prefixo, que denominaremos *palavra reconhecida* pelo estado q, só pode ser aumentado através de um caractere igual ao próximo caractere do padrão, então apenas uma transição pode aumentar a distância em relação ao estado inicial.

Assim, de acordo com este critério, para cada estado existem apenas dois tipos de transições e portanto, na representação abreviada, podemos indicar apenas duas transições em cada estado. A primeira, que denominaremos *transição desejável*, será aquela que aumenta o prefixo já reconhecido. A segunda, a *transição de falha*, será a representante de todas as outras transições de um estado. A figura 2.3 mostra o diagrama de estados da representação abreviada do autômato reconhecedor de *abcaabcaba* correspondente ao autômato completo das figuras 2.1 e 2.2. As arestas pontilhadas indicam as *transições de falha* e as arestas contínuas indicam as *transições desejáveis*. Vale ressaltar que este tipo de representação implicará em algumas modificações na interpretação do autômato que passará a ser a seguinte: suponha que o estado corrente do autômato seja j e que a cabeça de leitura da fita esteja sobre o caractere a. Se a for igual ao rótulo da transição desejável do estado j, isto é, se a é o *caractere desejável pelo estado* j, então o estado corrente do autômato passa a ser $j+1$ e a cabeça de leitura é avançada para o caractere seguinte da fita. Entretanto, se a não for o caractere desejável pelo estado j então a transição de falha é efetuada e, neste caso, a cabeça de leitura não é avançada porque o estado indicado

[1]Consideramos a distância entre dois vértices (estados) como sendo o comprimento, em número de arestas, do menor caminho orientado ligando estes dois vértices.

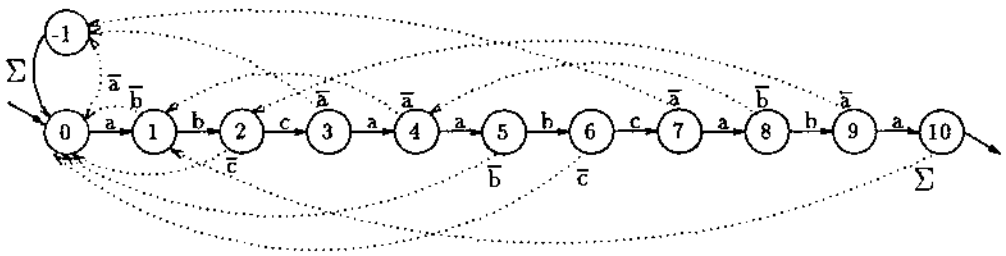


Figura 2.3: Diagrama de estados da representação abreviada do autômato reconhecedor de *abcaabcaba*.

pela transição de falha deve novamente analisar o caractere a, verificando se ele é igual ao seu caractere desejável. Daí, ou a transição desejável ou a transição de falha, deste novo estado será efetuada. Isto implica que o caractere a será analisado até que se obtenha um estado onde seja possível a realização da transição desejável.

Através da descrição acima, fica claro que uma transição de falha representa um tipo especial de transição, na qual a cabeça de leitura da fita não é avançada. Por isso, indicaremos as transições de falha através de arestas pontilhadas. Além disso, não é possível definir precisamente o rótulo de uma transição de falha pois, este tipo de transição é realizada caso o caractere sob a cabeça de leitura seja diferente do caractere desejável pelo estado corrente. Assim, estabeleceremos o rótulo de uma transição de falha da seguinte forma: se o rótulo da transição desejável é a então o rótulo da transição de falha será $\overline{a}$.

Na figura 2.4 a representação abreviada do autômato da figura 2.1 é dada através de uma tabela. É fácil ver que o espaço necessário para representar o autômato agora é independente do tamanho do alfabeto e depende apenas do tamanho do padrão, ou seja, o espaço utilizado pelo algoritmo é realmente $O(m)$. Na próxima seção descreveremos como a representação abreviada pode ser obtida em $O(m)$ unidades de tempo.

Pela figura 2.3, notamos que o estado final na representação abreviada não possui transição desejável. Isto ocorre porque se este estado for al-

Estado	Caractere Desejável	Transição Desejável	Transição de Falha
-1	Σ	0	–
0	a	1	-1
1	b	2	0
2	c	3	0
3	a	4	-1
4	a	5	1
5	b	6	0
6	c	7	0
7	a	8	-1
8	b	9	4
9	a	10	2
10	–	–	1

Figura 2.4: Tabela da representação abreviada do autômato reconhecedor de *abcaabcaba*.

cançado então uma ocorrência do padrão no texto foi obtida. Portanto, o processamento do texto visando buscar a próxima ocorrência do padrão deve prosseguir a partir do estado indicado pela transição de falha.

Além disso, comparando as figuras 2.1 e 2.3, verificamos que na representação abreviada foi incluído o estado -1 que não existia no autômato completo. Este estado foi incluído para manter preservada a propriedade de que a transição de falha de um estado j leva a um estado $i < j$. Note que esta propriedade é inerente ao processo de construção da representação abreviada e ela assume um papel importante neste tipo de representação porque ela impede a ocorrência de circuitos formados apenas por transições de falha. A violação desta propriedade poderia tornar o processamento de um texto interminável. Para ilustrar esta situação suponha que o texto *badaabada* fosse processado pela representação dada na figura 2.5. Quando a cabeça de leitura atingir a quinta posição do texto (caractere a) o estado corrente será o estado 4 e, neste caso, o processamento se torna interminável, passando continuamente pelos estados 0, 2 e 4 através de transições de falha. Este problema ocorre porque, na representação dada na figura 2.5, existem

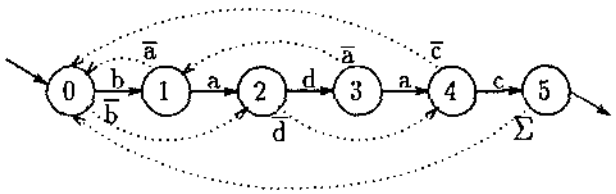


Figura 2.5: Exemplo onde é violada a propriedade de que a transição de falha de um estado j sempre indica um estado menor do que j.

transições de falha formando um circuito (estados 0, 2 e 4) cujos estados não têm, nenhum deles, o caractere a como desejável.

Vale ressaltar que, por definição, situações como a descrita acima (circuitos) não ocorrem na representação abreviada de um autômato reconhecedor de padrão. Além disso, a inclusão do estado -1 permite a realização de uma transição desejável qualquer que seja o caractere corrente no texto. Desta forma, a realização de várias transições de falha consecutivas, causadas por um mesmo caractere do texto, termina, na pior das hipóteses, no estado -1. Isto equivale a dizer que todas as tentativas de aproveitamento de prefixos da palavra reconhecida até o momento foram infrutíferas. Para ilustrar esta situação vamos processar o texto *abcaabcacabcabcabc* através do autômato representado na figura 2.3. Quando a cabeça de leitura atingir a posição 9 (caractere c) o autômato estará no estado 8 cujo caractere desejável é b. Portanto, a transição de falha do estado 8 é efetuada, o que nos leva ao estado 4. Neste estado ocorre uma nova transição de falha que nos leva ao estado 1. Novamente, mais uma transição de falha que nos leva ao estado 0 e finalmente, a transição de falha do estado 0 nos leva ao estado -1 onde ocorre a transição desejável. Isto significa que após havermos reconhecido a palavra *abcaabca* no texto, ou seja, após atingirmos o estado 8, a "chegada" do caractere c faz com que não seja possível aproveitar nenhum prefixo daquela palavra. Portanto, o caractere corrente do texto (c) pode ser desprezado, o que é feito através da transição desejável do estado -1 para o estado 0. Esta transição é efetuada qualquer que seja o caractere corrente no texto, isto é, ela tem como único objetivo avançar a cabeça de leitura no texto.

Resumindo, a representação abreviada do autômato reconhecedor do padrão $p = p_1 \dots p_m$ é composta de $m + 2$ estados cujos índices variam de -1 a m, sendo que o estado inicial é o estado 0 e o estado final é m. Além disso, para cada estado j tal que $0 \leq j \leq m - 1$ existem dois tipos de transições: a transição desejável que vai do estado j para o estado $j + 1$ quando o caractere sob a cabeça de leitura é igual a p_{j+1} (esta transição avança a cabeça de leitura) e a transição de falha que vai do estado j para um estado $i < j$ quando o caractere sob a cabeça de leitura é diferente de p_{j+1} (esta transição não avança a cabeça de leitura). O estado -1 é um estado especial uma vez que a sua transição desejável é efetuada qualquer que seja o caractere sob a cabeça de leitura e portanto, este estado não tem transição de falha. O estado m também é especial, pois só tem transição de falha, que vai de m para um estado $i < m$. Logo, a representação abreviada do autômato reconhecedor do padrão $p_1 \dots p_m$, denotada por $\mathcal{A}_m$, pode ser descrita da seguinte forma:

- $Q := \{j \mid -1 \leq j \leq m\}$
- $\delta(j, a) := \begin{cases} 0 & \text{se } j = -1 \\ j + 1 & \text{se } 0 \leq j < m \text{ e } a = p_{j+1} \\ \delta(f(j), a) & \text{caso contrário} \end{cases}$
  onde $f(j)$ é o estado indicado pela transição de falha do estado j.
- $q_0 := 0$
- $F := \{m\}$

Convém ressaltar que δ está bem definida, uma vez que $f(j) < j$ para todo $j : 0 \leq j \leq m$.

Desta forma, fica claro que para obtermos $\mathcal{A}_m$ basta calcularmos o valor da transição de falha dos estados de 0 a m, ou seja, na realidade, para representarmos $\mathcal{A}_m$ através de uma tabela semelhante àquela da figura 2.4, basta armazenarmos a coluna *transição de falha*, conforme mostra a figura 2.6.

Uma outra maneira de contornar o problema de que o número de transições no autômato completo depende do tamanho do alfabeto foi proposta por Simon [Sim93]. Neste trabalho o autor analisa o autômato reduzido que reconhece $\Sigma^* p$ e demonstra que neste autômato são necessárias, no máximo, m transições que diminuem a distância em relação ao estado inicial (transições correspondentes às transições de falha). A partir deste

Estado	Transição de Falha
0	-1
1	0
2	0
3	-1
4	1
5	0
6	0
7	-1
8	4
9	2
10	1

Figura 2.6: Tabela da representação abreviada do autômato reconhecedor de *abcaabcaba*.

resultado o autor propõe uma outra representação para o autômato reconhecedor de $\Sigma^* p$ que permite a obtenção de uma variação mais eficiente para o algoritmo KMP. Maiores detalhes sobre estes resultados também podem ser encontrados em [Han93a] e [Han93b].

2.2 Um Algoritmo para Obter $\mathcal{A}_m$

Nosso objetivo, agora, é descrever como obter a representação abreviada do autômato reconhecedor do padrão $p_1 \ldots p_m$ que, conforme vimos na seção anterior, é indicada por $\mathcal{A}_m$. Para tornar esta descrição mais clara, vamos inicialmente estabelecer como obter uma representação abreviada preliminar que basicamente fornece as mesmas informações que $\mathcal{A}_m$, mas que é mais simples (e menos eficiente) do que $\mathcal{A}_m$. Vamos indicar a representação abreviada preliminar por $\mathcal{B}_m$, sendo que nesta representação a transição de falha de um estado j será denotada por $g_m(j)$. Depois de estabelecido como obter $\mathcal{B}_m$, descreveremos como podemos "modificar" $\mathcal{B}_m$ para obter $\mathcal{A}_m$.

Na seção anterior, vimos que o valor da transição de falha do estado 0 é -1. Assim, a representação abreviada (preliminar) do autômato reconhecedor do padrão vazio, que é indicada por $\mathcal{B}_0$, é dada por:

j	$g_0(j)$
0	-1

Agora, observe que o autômato reconhecedor de $p_1 \ldots p_m$ pode ser obtido estendendo-se o autômato reconhecedor de $p_1 \ldots p_{m-1}$ incluindo-se o estado m juntamente com as suas transições. Então $\mathcal{B}_m$ será uma extensão de $\mathcal{B}_{m-1}$ onde devemos incluir o valor de $g_m(m)$. Isto implica que para obter $\mathcal{B}_m$ a partir de $\mathcal{B}_{m-1}$, precisamos calcular apenas o valor de $g_m(m)$.

Inicialmente sabemos que a transição de falha de um estado j, onde $0 \leq j \leq m$, é efetuada em uma das duas situações: se $j < m$ e o caractere corrente do texto é diferente do caractere desejável pelo estado j ou se $j = m$. Além disso, podemos perceber que a transição de falha do estado j deve levar a um estado $i < j$ tal que a maior parte possível da palavra já reconhecida até o estado j seja aproveitada.

Assim, imaginando que o padrão está alinhado com o texto, então a realização de uma transição de falha equivale a deslocar o padrão em relação ao texto para verificar uma possível ocorrência deste padrão numa outra posição do texto. A figura 2.7 ilustra este deslocamento quando ocorre uma diferença entre p_{j+1} (caractere desejável pelo estado $j : j < m$) e o caractere do texto alinhado com ele, indicado por t_{k+1}. Através desta figura, podemos perceber que uma ocorrência do padrão, após o deslocamento, somente será possível se as partes já verificadas do padrão que ficaram sobrepostas forem iguais. Além disso, o deslocamento deve ser o menor

dentre todos aqueles que satisfazem esta condição, pois assim impedimos que possíveis ocorrências do padrão sejam "esquecidas".

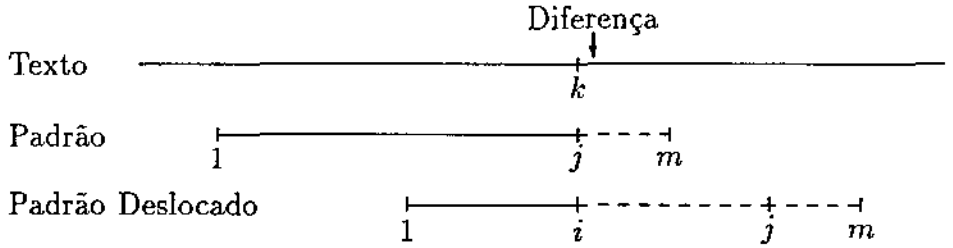


Figura 2.7: Após o deslocamento, as partes do padrão indicadas em linha contínua que ficaram sobrepostas devem ser iguais.

Portanto, para estabelecer o deslocamento devemos determinar o comprimento do maior prefixo próprio de $p_1 \dots p_j$ que também é sufixo de $p_1 \dots p_j$, ou seja, o deslocamento a ser efetuado no padrão pode ser determinado calculando-se o maior i, com $0 \leq i \leq j-1$, tal que $p_1 \dots p_i$ seja um sufixo próprio de $p_1 \dots p_j$. Isto significa que o padrão deve ser deslocado $j - i$ posições.

Agora, observe que desta forma as comparações podem prosseguir a partir da posição $k+1$ no texto e $i+1$ no padrão, pois

$$p_1 \cdots p_i = p_{j-i+1} \cdots p_j = t_{k-i+1} \dots t_k.$$

Isto equivale a dizer que $g_m(j)$ deve indicar que as comparações devem prosseguir a partir do estado i. Assim, denotando por $W(r)$ a palavra $p_1 \dots p_r$, podemos concluir que $W(g_m(j))$ deve ser o maior sufixo próprio de $W(j)$.

Deste modo, para obter o valor de $g_m(m)$ devemos determinar o estado i tal que $W(i)$ seja o maior sufixo próprio de $W(m)$, o que pode ser feito da seguinte forma: suponha que B_{m-1} já foi calculada; então temos que $W(g_{m-1}(m-1))$ é o maior sufixo próprio de $W(m-1)$. Lembrando que

$$W(m) = W(m-1) \cdot p_m,$$

então i deve ser tal que $W(i-1)$ seja o maior sufixo próprio de $W(m-1)$ e $p_i = p_m$, isto é, i deve ser tal que $W(i-1)$ seja o maior sufixo próprio de $W(m-1)$ que seja seguido de um caractere igual a p_m. Conforme mostraremos a seguir, o estado i pode ser obtido efetuando a transição a partir do estado $g_{m-1}(m-1)$ com o caractere p_m, ou seja,

$$g_m(m) = \delta_{\mathcal{B}_{m-1}}(g_{m-1}(m-1), p_m) = \delta_{\mathcal{B}_{m-1}}(m-1, p_m),$$

onde $\delta_{\mathcal{B}_{m-1}}$ é a função de transição em $\mathcal{B}_{m-1}$. Assim, podemos estabelecer a seguinte definição:

Definição 2.2 Seja $m \geq 0$. Então $\mathcal{B}_m$ é descrita através dos valores de $g_m(j)$, para $0 \leq j \leq m$, que são dados por:

(i) $g_0(0) := -1$

(ii) Para $m \geq j > 0$,

$$g_m(j) := \begin{cases} \delta_{\mathcal{B}_{m-1}}(m-1, p_m) & \text{se } j = m \\ g_{m-1}(j) & \text{caso contrário.} \end{cases}$$

Visto que g_m é uma extensão de g_{m-1} então a partir de agora omitiremos o índice de $g_m(j)$ indicando a transição de falha do estado j apenas por $g(j)$. Entretanto, em alguns casos será necessário especificar este índice para evitar ambigüidades.

Agora, vamos mostrar que através de $\mathcal{B}_m$ é possível obter todas as ocorrências de $p_1 \ldots p_m$ num texto t. Vamos indicar por $\mathcal{B}_m(w)$ o estado alcançado em $\mathcal{B}_m$ ao final do processamento de $w \in \Sigma^*$, isto é, $\mathcal{B}_m(w) = \delta_{\mathcal{B}_m}(0, w)$.

Lema 2.3 Seja $a \in \Sigma$ e sejam $m, q \in \mathcal{Z}$ tais que $0 \leq q \leq m$. Então temos que:

(i) $g_m(q) < q$.

(ii) $\delta_{\mathcal{B}_m}(q, a) \leq q + 1$ com igualdade somente se $q < m$ e $a = p_{1+q}$.

(iii) Para $m > 0$, se $q < m-1$ ou se $q = m-1$ e $a \neq p_m$ então $\delta_{\mathcal{B}_m}(q, a) = \delta_{\mathcal{B}_{m-1}}(q, a)$.

Demonstração: facilmente obtida por indução dupla em m e em q, com maior prioridade em m. □

Teorema 2.4 Sejam $m \geq 0$, $w \in \Sigma^*$ e $i := \mathcal{B}_m(w)$. Então as seguintes afirmações são verdadeiras:

(i) $w \in \Sigma^* p_1 \ldots p_i$

(ii) $w \notin \Sigma^* p_1 \ldots p_j \qquad \forall j : i < j \leq m.$

Isto equivale a dizer que $p_1 \ldots p_i$ é o maior prefixo de $p_1 \ldots p_m$ que é também sufixo de w e assim, podemos estabelecer o seguinte corolário:

Corolário 2.5 A linguagem aceita por $\mathcal{B}_m$ é $\Sigma^* p_1 \ldots p_m$. □

Demonstração do Teorema 2.4: Se $m = 0$ então $i = \mathcal{B}_0(w) = 0$; neste caso, (i) é trivialmente verdadeira e (ii) é vacuosamente verdadeira. Por outro lado, se $w = \epsilon$ então $i = \mathcal{B}_m(\epsilon) = 0$; neste caso, (i) e (ii) são trivialmente verdadeiras.

Podemos então supor que $m > 0$ e $w \neq \epsilon$. Assim, adotaremos como hipótese de indução que a asserção é verdadeira para todo natural m' e todo prefixo w' de w tais que ou $m' < m$ ou $m' = m$ e w' é um prefixo próprio de w, isto é, $|w'| < |w|$.

Como $w \neq \epsilon$ então existem $x \in \Sigma^*$ e $a \in \Sigma$ tais que $w = x \cdot a$. Seja $r := \mathcal{B}_{m-1}(x)$ e $s := \mathcal{B}_m(x)$. É claro que:

$$0 \leq r \leq m-1 \tag{2.1}$$
$$0 \leq s \leq m \tag{2.2}$$

Daí, pela hipótese de indução temos que:

$$x \in \Sigma^* p_1 \ldots p_r \tag{2.3}$$
$$x \notin \Sigma^* p_1 \ldots p_j \qquad \forall j : r < j \leq m-1 \tag{2.4}$$
$$x \in \Sigma^* p_1 \ldots p_s \tag{2.5}$$
$$x \notin \Sigma^* p_1 \ldots p_j \qquad \forall j : s < j \leq m \tag{2.6}$$

o que permite estabelecer as seguintes proposições:

Proposição 2.6 Ou $r = s$ ou $s = m$.

Demonstração : Vamos provar inicialmente que $r \leq s$. Suponhamos, por absurdo, que $r > s$. De (2.6) obtemos que $x \notin \Sigma^* p_1 \dots p_r$, o que contradiz (2.3). Portanto, realmente $r \leq s$.

Agora, suponhamos por absurdo que $r < s$ e $s < m$. De (2.4) obtemos que $x \notin \Sigma^* p_1 \dots p_s$, o que contradiz (2.5). Logo, ou $r = s$ ou $s = m$. □

Proposição 2.7 Para todo $a \in \Sigma$, $i = \delta_{\mathcal{B}_m}(s, a) = \delta_{\mathcal{B}_m}(r, a)$.

Demonstração : Temos que $i = \mathcal{B}_m(w) = \mathcal{B}_m(x \cdot a) = \delta_{\mathcal{B}_m}(\mathcal{B}_m(x), a) = \delta_{\mathcal{B}_m}(s, a)$. Portanto, fica demonstrada a primeira igualdade. Agora, pela proposição 2.6 temos que ou $r = s$ ou $s = m$. Se $r = s$ então a segunda igualdade é trivialmente verdadeira. Por outro lado, se $s = m$ então por (2.5) temos que $x \in \Sigma^* p_1 \dots p_m$ e daí, existe $y \in \Sigma^* p_1 \dots p_{m-1}$ tal que $x = y \cdot p_m$. Por hipótese de indução (corolário 2.5), $\mathcal{B}_{m-1}(y) = m - 1$. Assim, $g(m) = \delta_{\mathcal{B}_{m-1}}(m - 1, p_m) = r$ e portanto, $\delta_{\mathcal{B}_m}(s, a) = \delta_{\mathcal{B}_m}(m, a) = \delta_{\mathcal{B}_m}(g(m), a) = \delta_{\mathcal{B}_m}(r, a)$. □

Finalmente, para concluir a demonstração do teorema vamos considerar dois casos:

Caso 1 : $r = m - 1$ e $a = p_m$. De (2.3) $x \in \Sigma^* p_1 \dots p_{m-1}$ o que implica que $w = x \cdot a \in \Sigma^* p_1 \dots p_m$. Por outro lado, pela proposição 2.7, $i = \delta_{\mathcal{B}_m}(r, a) = \delta_{\mathcal{B}_m}(m - 1, p_m) = m$. Assim, provamos que $w \in \Sigma^* p_1 \dots p_m$ e que $i = m$, o que torna a afirmação (i) verdadeira e (ii) vacuosamente verdadeira.

Caso 2 : $r < m - 1$ ou $a \neq p_m$. Pelo lema 2.3 (iii), $\delta_{\mathcal{B}_m}(r, a) = \delta_{\mathcal{B}_{m-1}}(r, a)$ e, pela proposição 2.7, $i = \delta_{\mathcal{B}_m}(r, a)$ o que implica que $i = \delta_{\mathcal{B}_{m-1}}(r, a) = \mathcal{B}_{m-1}(w)$. Daí, por hipótese de indução, $w \in \Sigma^* p_1 \dots p_i$ e $\forall j : i < j \leq m-1$, $w \notin \Sigma^* p_1 \dots p_j$. Deste modo, para obter que as afirmações (i) e (ii) são verdadeiras, fica faltando mostrar apenas que $w \notin \Sigma^* p_1 \dots p_m$, o que pode ser facilmente obtido observando que: se $r < m - 1$ então $x \notin \Sigma^* p_1 \dots p_{m-1}$ e isto implica que $w = x \cdot a$ não pode pertencer a $\Sigma^* p_1 \dots p_m$; por outro lado, se $r = m - 1$ então pela hipótese do caso, $a \neq p_m$ e portanto, $w = x \cdot a \notin \Sigma^* p_1 \dots p_m$. Desta forma, concluímos a demonstração do teorema. □

A partir deste teorema fica estabelecido que $\mathcal{B}_m$ processa corretamente um texto $t \in \Sigma^*$ obtendo todas as ocorrências do padrão $p_1 \dots p_m$ em t. Mais

precisamente, $\mathcal{B}_m$ reconhece a linguagem $\Sigma^* p$. Obviamente, todo autômato determinístico que reconhece $\Sigma^* p$ tem pelo menos $m+1$ estados e portanto, o autômato representado por $\mathcal{B}_m$ é mínimo.

Como salientamos no início desta seção, $\mathcal{B}_m$ é uma representação abreviada preliminar que pode ser "transformada" na representação abreviada $\mathcal{A}_m$ que é mais eficiente para o processamento de um texto. Para "transformar" $\mathcal{B}_m$ em $\mathcal{A}_m$ vamos considerar a seguinte observação: se $0 < j < m$ e se $p_{1+j} = p_{1+g(j)}$ então uma transição de falha no estado j implica numa transição de falha no estado $g(j)$, ou seja, se $a \neq p_{1+j}$ então $\delta_{\mathcal{B}_m}(j,a) = \delta_{\mathcal{B}_m}(g(j),a) = \delta_{\mathcal{B}_m}(g(g(j)),a)$. Portanto, o valor de $g(j)$ pode ser atualizado para $g(g(j))$. Entretanto, pode ser que $p_{1+g(g(j))}$ também seja igual a p_{1+j} e daí, o valor de $g(j)$ poderia ser atualizado para $g(g(g(j)))$. Prosseguindo com este raciocínio concluímos que a atualização do valor de $g(j)$ consiste em encontrar o menor inteiro positivo l tal que: ou $g^l(j) = -1$ ou $g^l(j) \geq 0$ e $p_{1+g^l(j)} \neq p_{1+j}$. A "transformação" de $\mathcal{B}_m$ em $\mathcal{A}_m$ estará concluída após atualizarmos o valor de $g(j)$ para todo $j : 0 < j < m$.

Assim, denotando por $\delta_{\mathcal{A}_m}$ a função de transição de $\mathcal{A}_m$ e por $f_m(j)$ o valor da transição de falha de um estado j em $\mathcal{A}_m$ então podemos estabelecer que:

Definição 2.8 Seja $m \geq 0$. Então $\mathcal{A}_m$ é descrita através dos valores de $f_m(j)$, para $0 \leq j \leq m$, que são dados por:

(i) $f_0(0) := -1$

(ii) Para $m \geq j > 0$, seja $s := f_{m-1}(m-1)$; então:

$$f_m(j) := \begin{cases} \delta_{\mathcal{A}_{m-1}}(m-1, p_m) & \text{se } j = m \\ f_{m-1}(s) & \text{se } j = m-1,\ s \geq 0 \text{ e } p_m = p_{1+s} \\ f_{m-1}(j) & \text{caso contrário.} \end{cases}$$

Corolário 2.9 Para todo $j : 0 < j < m$, $p_1 \ldots p_{f_m(j)}$ é a maior borda disjunta de $p_1 \ldots p_j$. □

Conforme mostraremos a seguir, a representação abreviada $\mathcal{A}_m$ obtida através da definição acima é exatamente igual àquela obtida através da "transformação" de $\mathcal{B}_m$. Porém, o método de construção estabelecido pela definição é mais eficiente. Para ilustrar este ganho em eficiência, considere

a figura 2.8, onde temos $\mathcal{A}_5$ e $\mathcal{B}_5$ para o padrão *aaaaa*, e suponha que a partir destas representações queremos obter $\mathcal{A}_6$ para o padrão *aaaaab*. Utilizando a definição 2.8, são necessárias apenas 2 comparações para obter $\mathcal{A}_6$ enquanto que se utilizarmos o processo de transformação, somente para obter $\mathcal{B}_6$ são necessárias 5 comparações.

j	$f(j)$
0	-1
1	-1
2	-1
3	-1
4	-1
5	4

(a)

j	$g(j)$
0	-1
1	0
2	1
3	2
4	3
5	4

(b)

Figura 2.8: (a) $\mathcal{A}_5$ do padrão *aaaaa* (b) $\mathcal{B}_5$ do padrão *aaaaa*.

Agora, vamos mostrar que a definição 2.8 e o método de "transformação" de $\mathcal{B}_m$ produzem a mesma representação abreviada $\mathcal{A}_m$, ou seja, vamos mostrar que $f_m(j)$ é igual ao valor atualizado de $g_m(j)$. Além disso, vamos mostrar também que $\mathcal{A}_m$ e $\mathcal{B}_m$ são representações abreviadas de um mesmo autômato, isto é, $\mathcal{A}_m$ e $\mathcal{B}_m$ são equivalentes.

Proposição 2.10 Dado $m \geq 0$ temos que:

(i) para $0 \leq j \leq m$,

$$f_m(j) = \begin{cases} g_m(j) & \text{se } j = m \\ g_m^l(j) & \text{se } 0 \leq j < m, \end{cases}$$

onde l é o menor inteiro positivo tal que: ou $g_m^l(j) = -1$ ou $g_m^l(j) \geq 0$ e $p_{1+g_m^l(j)} \neq p_{1+j}$.

(ii) para $-1 \leq j \leq m$ e $a \in \Sigma$, $\delta_{\mathcal{A}_m}(j,a) = \delta_{\mathcal{B}_m}(j,a)$.

Demonstração : Por indução em m. Se $m = 0$ então (i) é verdadeira pois, para $j = 0$ temos que $f_0(0) = -1 = g_0(0)$ e (ii) também é verdadeira porque,

para $-1 \leq j \leq 0$, $\delta_{\mathcal{A}_0}(j,a) = 0 = \delta_{\mathcal{B}_0}(j,a)$. Vamos supor que as asserções (i) e (ii) sejam verdadeiras para todo m' tal que $0 \leq m' < m$ e vamos provar que elas são verdadeiras para m. Para provar que (i) é verdadeira vamos considerar os seguintes casos:

Caso 1: $0 \leq j < m-1$. Pela definição 2.8, $f_m(j) = f_{m-1}(j)$. Agora, pela hipótese de indução de (i), $f_{m-1}(j) = g^l_{m-1}(j)$ onde l é o menor inteiro positivo tal que: ou $g^l_{m-1}(j) = -1$ ou $g^l_{m-1}(j) \geq 0$ e $p_{1+g^l_{m-1}(j)} \neq p_{1+j}$. Mas, pela definição 2.2, g_m é uma extensão de g_{m-1} e portanto, (i) é verdadeira.

Caso 2: $j = m-1$. Seja $s := f_{m-1}(m-1)$. Pela hipótese de indução de (i), $f_{m-1}(m-1) = g_{m-1}(m-1)$ e, pela definição 2.2, $g_m(m-1) = g_{m-1}(m-1)$. Daí, $s = g_m(m-1)$ e $-1 \leq s < m-1$; agora vamos considerar dois subcasos:

- $s \geq 0$ e $p_m = p_{1+s}$. Pela definição 2.8, $f_m(m-1) = f_{m-1}(s) = f_m(s)$. Além disso, pelo caso 1, $f_m(s) = g^l_m(s)$ onde l é o menor inteiro positivo tal que: ou $g^l_m(s) = -1$ ou $g^l_m(s) \geq 0$ e $p_{1+g^l_m(s)} \neq p_{1+s}$. Mas, $s = g_m(m-1)$ e portanto, $f_m(m-1) = g^{l+1}_m(m-1)$, sendo que, pela hipótese deste subcaso, $l+1$ é o menor inteiro positivo tal que: ou $g^{l+1}_m(m-1) = -1$ ou $g^{l+1}_m(m-1) \geq 0$ e $p_{1+g^{l+1}_m(m-1)} \neq p_m$.

- ou $s = -1$ ou $s \geq 0$ e $p_m \neq p_{1+s}$. Daí, $f_m(m-1) = f_{m-1}(m-1) = s = g_m(m-1)$, sendo que, pela hipótese deste subcaso, ou $g_m(m-1) = -1$ ou $g_m(m-1) \geq 0$ e $p_{1+g_m(m-1)} \neq p_m$ e portanto, (i) é verdadeira com $l = 1$.

Caso 3: $j = m$. Neste caso, temos que mostrar que $f_m(m) = g_m(m)$. Sabemos que

$$\begin{aligned} f_m(m) &= \delta_{\mathcal{A}_{m-1}}(m-1, p_m) \\ &= \delta_{\mathcal{B}_{m-1}}(m-1, p_m) \\ &= g_m(m) \end{aligned}$$

onde, a primeira igualdade decorre da definição 2.8, a segunda vem da hipótese de indução de (ii) e a terceira segue da definição 2.2.

Assim, concluímos que a asserção (i) é verdadeira e agora, vamos mostrar, por indução em j, que a asserção (ii) também é verdadeira. Se $j = -1$ então $\delta_{\mathcal{A}_m}(-1,a) = 0 = \delta_{\mathcal{B}_m}(-1,a)$. Agora, suponha que, para $-1 \leq j' < j$, $\delta_{\mathcal{A}_m}(j',a) = \delta_{\mathcal{B}_m}(j',a)$. Vamos provar que $\delta_{\mathcal{A}_m}(j,a) = \delta_{\mathcal{B}_m}(j,a)$. Se $j < m$ e $a = p_{1+j}$ então $\delta_{\mathcal{A}_m}(j,a) = 1 + j = \delta_{\mathcal{B}_m}(j,a)$. Assim, resta-nos verificar

que (ii) é verdadeira se $j = m$ ou se $j < m$ e $a \neq p_{1+j}$. Em ambos os casos temos, pela definição 2.8, que:

$$\delta_{\mathcal{A}_m}(j, a) = \delta_{\mathcal{A}_m}(f_m(j), a).$$

Agora, pelo item (i) desta proposição, $f_m(j) = g_m^l(j)$ onde, se $j = m$ então $l = 1$ ou se $j < m$ então l é o menor inteiro positivo tal que: $g_m^l = -1$ ou $g_m^{(l)}(j) \geq 0$ e $p_{1+g_m^l(j)} \neq p_{1+j}$. Além disso, como $l \geq 1$ então, pelo lema 2.3 (i), $f_m(j) = g_m^l(j) < j$. Daí, pela hipótese de indução em j,

$$\delta_{\mathcal{A}_m}(j, a) = \delta_{\mathcal{B}_m}(g_m^l(j), a).$$

Mas, pela hipótese do caso, se $j < m$ então $a \neq p_{1+j}$ e portanto, para todo r : $1 \leq r < l$, $p_{1+g_m^r(j)} = p_{1+j} \neq a$, o que implica que $\delta_{\mathcal{B}_m}(g_m l(j), a) = \delta_{\mathcal{B}_m}(j, a)$. Por outro lado, se $j = m$ então $l = 1$ e daí, $\delta_{\mathcal{B}_m}(g_m(m), a) = \delta_{\mathcal{B}_m}(m, a)$, ou seja,

$$\delta_{\mathcal{A}_m}(j, a) = \delta_{\mathcal{B}_m}(j, a).$$

Assim, fica demonstrada a veracidade das asserções (i) e (ii). □

Corolário 2.11 A linguagem aceita por $\mathcal{A}_m$ é $\Sigma^* p_1 \ldots p_m$. □

Conforme salientamos no início desta seção, $\mathcal{A}_m$ processa um texto de forma mais eficiente do que $\mathcal{B}_m$. Para ilustrar este fato, vamos considerar a figura 2.8 e o texto *aaaabaaaaa*. Se processarmos este texto através de $\mathcal{B}_5$ então serão necessárias 14 comparações para obtermos a primeira e única ocorrência do padrão no texto enquanto que se o processamento for efetuado através de $\mathcal{A}_5$, este mesmo resultado será obtido após 10 comparações. De um modo geral, se o processamento de um texto for efetuado utilizando $\mathcal{B}_m$ então um mesmo caractere do texto pode vir a ser comparado m vezes. Por outro lado, conforme mostraremos na próxima seção, se utilizarmos $\mathcal{A}_m$ então um mesmo caractere do texto pode ser comparado, no máximo, $O(\log m)$ vezes.

Assim, concluímos a descrição de como obter a representação abreviada de um autômato reconhecedor de padrão e, para encerrar esta seção, apresentamos na figura 2.9 uma implementação do algoritmo para construir $\mathcal{A}_m$ e na figura 2.10 apresentamos uma implementação do algoritmo KMP. Ambos os algoritmos utilizam uma função *delta* cuja implementação é dada juntamente com o algoritmo *Constroi_*$\mathcal{A}_m$. É importante ressaltar que nesta

função utilizamos uma sentinela na "posição" 0 (fictícia) do padrão, pois desta forma a implementação do algoritmo se torna mais simples uma vez que sem a sentinela seria necessário incluir o teste $s \geq 0$ no "loop" **while**. Entretanto, na análise de complexidade, devemos considerar uma implementação que não faz uso da sentinela, pois, como contamos apenas as comparações que envolvem caracteres, então o uso da sentinela substitui uma comparação que não é contada ($s \geq 0$) por uma que é ($a \neq p_0$).

Algoritmo delta(s,a);

Entrada : *O estado s e o caractere a.*
Saída : *O estado* $\delta(s,a)$.

begin
  $p_0 := a$; $\{sentinela\}$
  if $s = m$ **then** $s := f(s)$;
  while $a \neq p_{s+1}$ **do** $s := f(s)$;
  $delta := s + 1$;
end;

Algoritmo Constroi_$\mathcal{A}_m$;

Entrada : *O padrão p.*
Saída : *Tabela com os valores de f.*

begin
  $f(0) := -1$;
  for $j := 1$ **to** m **do**
  begin
    $f(j) := delta(f(j-1), p_j)$;
    if $f(j-1) \geq 0$ **and** $p_{1+f(j-1)} = p_j$ **then** $f(j-1) := f(f(j-1))$
  end
end;

Figura 2.9: Uma implementação do algoritmo para construir $\mathcal{A}_m$.

Algoritmo KMP;

Entrada : *O texto t e o padrão p.*
Saída : *Lista de posições em que p ocorre em t.*

```
begin
   {Construa A_m para o padrão p.}
   j := 0;
   for k := 1 to n do
   begin
      j := delta(j, t_k);
      if j = m then write ('O padrão ocorre na posição ', k - m + 1);
   end
end;
```

Figura 2.10: Uma implementação do algoritmo KMP.

2.3 Análise de Complexidade do Algoritmo KMP

Nesta seção analisaremos o comportamento do algoritmo KMP mostrando que através dele é possível realizar a busca do padrão p no texto t, efetuando $O(n)$ transições[2], ou seja, mostraremos que através da representação abreviada é possível processar o texto t em tempo linear no comprimento de t. É importante salientar que este resultado é válido para todas as representações abreviadas em que o valor da transição de falha de um estado j é menor do que j (por exemplo, $\mathcal{A}_m$ e $\mathcal{B}_m$).

Além disso, vamos mostrar também que a representação abreviada do autômato reconhecedor do padrão p pode ser construída efetuando-se $O(m)$ transições. Assim, teremos mostrado que o algoritmo KMP efetua, no máximo, $O(m+n)$ transições para realizar a busca do padrão p no texto t.

Finalmente, mostraremos que se utilizarmos a representação abreviada $\mathcal{A}_m$ então um único caractere do texto t pode causar, no máximo, $O(\log m)$ transições de falha consecutivas, ou seja, isto equivale a dizer que o processamento de t através de $\mathcal{A}_m$ consome, no máximo, $O(\log m)$ unidades de tempo entre a leitura de dois caracteres consecutivos deste texto. Como podemos observar através da figura 2.8, esta propriedade não se verifica se o texto for processado através de $\mathcal{B}_m$.

Na demonstração dos resultados acima vamos utilizar a seguinte notação adicional. Seja $\mathcal{C}_m$ uma representação abreviada do autômato reconhecedor do padrão p onde a transição de falha do estado j $(0 \leq j \leq m)$, que indicaremos por $h(j)$, é menor do que j, isto é, em $\mathcal{C}_m$ temos que $h(j) < j$. Além disso, seja $N_h(j,a)$ o número de transições de falha que são efetuadas para calcular $\delta_{\mathcal{C}_m}(j,a)$, onde $j : -1 \leq j \leq m$ é um estado de $\mathcal{C}_m$ e $a \in \Sigma$. Assim,

$$N_h(j,a) := \begin{cases} 0 & \text{se } 0 \leq j < m \text{ e } a = p_{1+j} \text{ ou se } j = -1 \\ 1 + N_h(h(j),a) & \text{caso contrário.} \end{cases}$$

Daí, podemos estabelecer o seguinte lema:

[2]Note que o número de comparações entre caracteres do texto e do padrão é menor do que ou igual ao número de transições pois, a transição desejável do estado -1 e a transição de falha do estado final são efetuadas sem que ocorram comparações.

Lema 2.12 Para $-1 \leq j \leq m$ e para $a \in \Sigma$ temos que $N_h(j,a) \leq j - \delta_{\mathcal{C}_m}(j,a) + 1$.

Demonstração: Por indução em j. Se $j = -1$ então $N_h(-1,a) = 0$ e $\delta(-1,a) = 0$, o que implica que a desigualdade é verdadeira. Agora, suponha que, para $-1 \leq j' < j$ temos $N_h(j',a) \leq j' - \delta(j',a) + 1$. Daí, vamos analisar duas situações. Se $N_h(j,a) = 0$ então $a = p_{1+j}$ o que implica que $\delta(j,a) = 1 + j$ e portanto, $N_h(j,a) = j - \delta(j,a) + 1$. Por outro lado, se $N_h(j,a) = 1 + N_h(h(j),a)$ então,

$$\begin{aligned} N_h(j,a) &\leq 1 + h(j) - \delta(h(j),a) + 1 \\ &= 1 + h(j) - \delta(j,a) + 1 \\ &\leq j - \delta(j,a) + 1 \end{aligned}$$

onde, a primeira desigualdade segue da hipótese de indução, a igualdade segue da hipótese do caso e a segunda desigualdade segue de $h(j) < j$. □

Agora, vamos estabelecer o teorema que nos permite concluir que o algoritmo KMP realiza, no máximo, $O(n)$ transições para efetuar a busca do padrão p no texto t.

Teorema 2.13 No processamento de t através de $\mathcal{C}_m$ ocorrem, no máximo, $2n - q_n$ transições (desejáveis e de falha), onde $q_n := \delta(0,t)$.

Demonstração: Para $0 \leq k \leq n$ seja $q_k := \delta(0, t_1 \ldots t_k)$ e seja i_k o número de transições realizadas em $\mathcal{C}_m$ durante o processamento de $t_1 \ldots t_k$. Vamos provar, por indução em k, que $i_k \leq 2k - q_k$. Se $k = 0$ então $i_k = 0 = q_k$ o que implica que a desigualdade é trivialmente verdadeira. Agora, suponha que $i_k \leq 2k - q_k$. Assim,

$$\begin{aligned} i_{k+1} &= i_k + N_h(q_k, t_{k+1}) + 1 \\ &\leq 2k - q_k + N_h(q_k, t_{k+1}) + 1 \\ &\leq 2k - \delta(q_k, t_{k+1}) + 2 \\ &= 2(k+1) - q_{k+1}, \end{aligned}$$

sendo que a primeira igualdade decorre da definição de i_{k+1}, as duas desigualdades seguintes seguem respectivamente da hipótese de indução e do lema 2.12 e a última igualdade vem do fato de que $\delta(q_k, t_{k+1}) = q_{k+1}$. □

A demonstração de que na construção da representação abreviada ocorrem, no máximo, $O(m)$ comparações se baseia no teorema anterior, sendo que inicialmente vamos estabelecer o resultado para $\mathcal{B}_m$ e depois mostraremos que este resultado também vale para $\mathcal{A}_m$. Lembre-se de que a transição de falha de um estado j em $\mathcal{B}_m$ é dada por $g(j)$ e em $\mathcal{A}_m$ por $f(j)$.

Corolário 2.14 Seja i_m o número de transições efetuadas para construir $\mathcal{B}_m$. Então $i_m < 2m - g(m)$.

Demonstração: Sabemos que a construção de $\mathcal{B}_m$ é feita calculando-se os valores de $g(j)$, sendo que $g(0) = -1$ e para $1 \leq j \leq m$, $g(j) = \delta_{\mathcal{B}_{j-1}}(g(j-1), p_j)$. Assim, $g(1) = \delta_{\mathcal{B}_0}(-1, p_1) = 0$, $g(2) = \delta_{\mathcal{B}_1}(0, p_2)$, $g(3) = \delta_{\mathcal{B}_2}(g(2), p_3)$, $\cdots$, $g(m) = \delta_{\mathcal{B}_{m-1}}(g(m-1), p_m)$. Isto implica que, para $2 \leq j \leq m$, $g(j) = \delta_{\mathcal{B}_{j-1}}(0, p_2 \ldots p_j)$, ou seja, i_m é igual ao número de transições efetuadas para calcular $g(1)$ mais o número de transições efetuadas para processar $p_2 \ldots p_m$ através de $\mathcal{B}_{m-1}$. Daí, pelo teorema 2.13, temos que $i_m \leq 1 + 2(m-1) - g(m) < 2m - g(m)$. □

Assim, temos que na construção de $\mathcal{B}_m$ são efetuadas, no máximo, $O(m)$ transições. Agora, pela definição 2.8, notamos que no cálculo de cada $f(j)$, para $0 \leq j \leq m$, pode ocorrer no máximo uma transição a mais do que no cálculo de $g(j)$. Portanto, na construção de $\mathcal{A}_m$ também são efetuadas, no máximo, $O(m)$ transições.

Finalmente, vamos mostrar que se utilizarmos $\mathcal{A}_m$ para processar o texto, então entre a leitura de dois caracteres consecutivos do texto, o algoritmo consome, no máximo, $O(\log m)$ unidades de tempo. Para isto, vamos inicialmente estabelecer a seguinte proposição:

Proposição 2.15 Seja $j : 0 < j < m$ um estado de $\mathcal{A}_m$ tal que $f(j) \neq -1$. Então $j > f(j) + f^2(j) + 1$.

Demonstração: Seja $q := f(j)$ e $s := f(q) = f^2(j)$. Se $s = -1$ então a proposição é verdadeira pois, para todo $j \geq 0$, temos que $j > f(j)$. Podemos então supor que $s \geq 0$. Neste caso, suponha por absurdo que existe um estado $j > 0$ tal que $j \leq q + s + 1$.

Inicialmente, pela definição da transição de falha f, temos que $W(s)$ é a maior borda disjunta de $W(q)$ e portanto, $p_{s+1} \neq p_{q+1}$. Além disso, $W(q)$ é a maior borda disjunta de $W(j)$ e portanto, $p_{q+1} \neq p_{j+1}$. Vamos agora analisar dois casos:

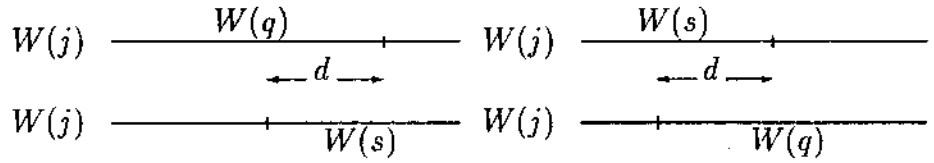


Figura 2.11: Sobreposições entre $W(q)$ e $W(s)$.

Caso 1 : $j < q+s+1$. Neste caso, há uma sobreposição entre $W(q)$ e $W(s)$ de comprimento $d = q + s - j$ conforme mostra a figura 2.11. Sabemos que o caractere imediatamente após $W(q)$ é p_{q+1} e, pela parte (a) da figura 2.11, notamos que este caractere está alinhado com o caractere p_{d+1} de $W(s)$, ou seja, $p_{q+1} = p_{d+1}$. Na parte (b) desta mesma figura, vemos que o caractere imediatamente após $W(s)$, que é p_{s+1}, está alinhado com o caractere p_{d+1} de $W(q)$, ou seja, $p_{s+1} = p_{d+1}$. Portanto, $p_{q+1} = p_{d+1} = p_{s+1}$ o que contradiz o fato de que $p_{s+1} \neq p_{q+1}$.

Caso 2 : $j = q + s + 1$. Neste caso, temos que

$$W(s)p_{s+1}W(q) = W(j) = W(q)p_{q+1}W(s).$$

Mas, pela proposição 1.21, temos que $p_{s+1} = p_{q+1}$ o que novamente nos leva a uma contradição.

Logo, $j > q + s + 1$. □

A partir da proposição acima vamos relacionar o comprimento de um padrão com os termos da seqüência de Fibonacci, denotados por S_i, onde $S_0 := 0$, $S_1 := 1$ e $S_i := S_{i-1} + S_{i-2}$ para $i \geq 2$.

Proposição 2.16 Seja m_r o comprimento mínimo de um padrão que permite a realização de r comparações com um mesmo caractere de um texto. Então $m_r \geq S_{r+2} - 1$.

Demonstração: Por indução em r. Para $r = 1$ ($r = 2$) temos que $m_1 = 1 = S_3 - 1$ ($m_2 = 2 = S_4 - 1$) e portanto, nestes casos a asserção é verdadeira. Para $r > 2$, suponha por hipótese de indução que a asserção seja verdadeira para todo $r' < r$.

Pela definição de m_r, temos que, em $\mathcal{A}_{m_r}$, o estado que permite a realização de r comparações com um mesmo caractere de um texto é $m_r - 1$

(senão haveria um padrão de comprimento menor do que m_r que também permite a realização destas r comparações). Conseqüentemente, os estados $f(m_r - 1)$ e $f^2(m_r - 1)$ permitem, respectivamente, a realização de $r - 1$ e $r - 2$ comparações com um mesmo caractere de um texto e portanto, $m_{r-1} \leq f(m_r - 1) + 1$ e $m_{r-2} \leq f^2(m_r - 1) + 1$.

Visto que, pela proposição 2.15,

$$m_r - 1 \geq f(m_r - 1) + f^2(m_r - 1) + 2$$

então

$$\begin{aligned} m_r - 1 &\geq m_{r-1} + m_{r-2} \\ &\overset{H.I.}{\geq} S_{r+1} - 1 + S_r - 1 \\ &= S_{r+2} - 2. \end{aligned}$$

Logo, $m_r \geq S_{r+2} - 1$. □

Finalmente, podemos estabelecer o seguinte teorema:

Teorema 2.17 Um mesmo caractere do texto pode ser comparado no máximo $\log_\phi(m + 1)$ vezes, onde $\phi := \frac{1+\sqrt{5}}{2}$.

Demonstração: Dado um padrão de comprimento m, seja r o número máximo de comparações que podem ser realizadas com um mesmo caractere de um texto, quando este texto é processado através de $\mathcal{A}_m$. Assim,

$$\begin{aligned} m &\geq m_r \\ &\geq S_{r+2} - 1 \\ &\geq \phi^r - 1, \end{aligned}$$

onde a primeira desigualdade segue da minimalidade de m_r, a segunda é devida à proposição 2.16 e a última decorre do fato de que, para todo $i \geq 1$, $S_i \geq \phi^{i-2}$, conforme estabelecido em [Knu69, seção 1.2, exercício 4]. Portanto, $m \geq \phi^r - 1$ o que implica que $r \leq \log_\phi(m + 1)$. □

Agora vamos mostrar que o limite imposto pelo teorema 2.17 é justo. Para isto vamos exibir uma família de padrões para os quais, na representação abreviada $\mathcal{A}_m$, existem estados que permitem a realização de exatamente $\lfloor \log_\phi(m + 1) \rfloor$ comparações.

Inicialmente, seja $\Psi_{[i]}$ a *palavra de Fibonacci* de comprimento S_i que é definida da seguinte forma:

$$\Psi_{[1]} := b$$
$$\Psi_{[2]} := a$$
$$\Psi_{[i]} := \Psi_{[i-1]}\Psi_{[i-2]} \text{ para } i \geq 3.$$

Portanto, $\Psi_{[3]} = ab$, $\Psi_{[4]} = aba$, $\Psi_{[5]} = abaab$ e assim por diante. Denotaremos por $\Psi_{[i],j}$ o j-ésimo caractere da palavra de Fibonacci $\Psi_{[i]}$, ou seja,

$$\Psi_{[i]} = \Psi_{[i],1} \ldots \Psi_{[i],S_i}.$$

A seguir apresentamos os valores da transição de falha para a representação abreviada do autômato reconhecedor de $\Psi_{[8]}$:

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$\Psi_{[8],j+1}$	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*a*	*b*	*a*	-
$f(j)$	-1	0	-1	1	0	-1	3	-1	1	0	-1	6	0	-1	3	-1	1	0	-1	11	-1	3

Na tabela abaixo apresentamos o comprimento mínimo m_r da palavra de Fibonacci que permite a realização de r comparações com um mesmo caractere de um texto. Apresentamos também os valores correspondentes de $\log_\phi(m_r+1)$ e através destes valores, juntamente com os da tabela acima, observamos que o limite estabelecido pelo teorema 2.17 é alcançado.

r	m_r	$\log_\phi(m_r+1)$
2	2	2.2831
3	4	3.3447
4	7	4.3214
5	12	5.3304
6	20	6.3270

Na verdade, para as palavras de Fibonacci temos que:

Proposição 2.18 Dada $\Psi_{[n]}$ então, para todo $i : 3 \leq i \leq S_n$, $f(S_i - 2) = S_{i-1} - 2$.

Demonstração : Por indução em i. Para $i = 3$, $f(S_3 - 2) = f(0) = -1 = S_2 - 2$ e portanto, a asserção é verdadeira. Agora, suponha por hipótese de indução que a asserção é verdadeira para todo $i' : 3 \leq i' < i$.

Podemos demonstrar facilmente por indução que as palavras de Fibonacci apresentam a seguinte propriedade:

$$\Psi_{[i-1]}\Psi_{[i-2]} = \wp(\Psi_{[i-2]}\Psi_{[i-1]}) \text{ para } i \geq 3, \tag{2.7}$$

onde $\wp(\alpha)$ corresponde a permutar os dois caracteres mais à direita em α (por exemplo, $\Psi_{[5]} = abaab$ e $\wp(\Psi_{[5]}) = ababa$).

Por definição, $\Psi_{[i]} = \Psi_{[i-1]}\Psi_{[i-2]}$ o que implica que $\Psi_{[i-1]}$ é um prefixo próprio de $\Psi_{[i]}$. Daí, $\Psi_{[i-1],1} \ldots \Psi_{[i-1],S_{i-1}-2}$ é um prefixo próprio de $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$. Por outro lado, aplicando a propriedade (2.7), temos que $\Psi_{[i]} = \wp(\Psi_{[i-2]}\Psi_{[i-1]})$ e portanto, $\Psi_{[i-1],1} \ldots \Psi_{[i-1],S_{i-1}-2}$ é também um sufixo de $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$ e daí $\Psi_{[i-1],1} \ldots \Psi_{[i-1],S_{i-1}-2}$ é uma borda de $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$.

Para mostrar que $\Psi_{[i-1],1} \ldots \Psi_{[i-1],S_{i-1}-2}$ é a maior borda de $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$ basta aplicar indução em i observando que

$$\Psi_{[i-2],1} \ldots \Psi_{[i-2],S_{i-2}-2} \quad \text{e} \tag{2.8}$$

$$\Psi_{[i-3],1} \ldots \Psi_{[i-3],S_{i-3}-2} \tag{2.9}$$

são bordas de $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$ e além disso, (2.9) é a maior borda de (2.8). Assim, se $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$ possuísse uma borda maior do que $\Psi_{[i-1],1} \ldots \Psi_{[i-1],S_{i-1}-2}$ então, pela definição de Ψ, (2.8) também teria uma borda maior do que (2.8) o que nos levaria a uma contradição.

Finalmente, pela propriedade (2.7), temos que $\Psi_{[i],S_i-1} \neq \Psi_{[i],S_{i-1}-1}$ e portanto, $\Psi_{[i-1],1} \ldots \Psi_{[i-1],S_{i-1}-2}$ é a maior borda disjunta de $\Psi_{[i],1} \ldots \Psi_{[i],S_i-2}$. Logo, $f(S_i - 2) = S_{i-1} - 2$. □

Corolário 2.19 Para $n \geq 3$, seja $\Pi_{[n]}$ o prefixo de $\Psi_{[n]}$ de comprimento $S_n - 1$ e seja $\Gamma_{[n]}$ uma seqüência de caracteres cujo prefixo de comprimento $S_n - 2$ coincide com o de $\Pi_{[n]}$ e $\Gamma_{[n],S_n-1} = c$, onde $c \notin \{a, b\}$. Então, tomando $\Pi_{[n]}$ como padrão e $\Gamma_{[n]}$ como texto, temos que o caractere $\Gamma_{[n],S_n-1}$ é comparado $\lfloor \log_\phi S_n \rfloor$ vezes.

Demonstração : Seja r o número de comparações realizadas com um mesmo caractere do texto. Pelo teorema 2.17,

$$r \leq \lfloor \log_\phi S_n \rfloor .$$

Por outro lado, por indução em n, é fácil demonstrar que a proposição 2.18 implica que $r = n - 2$.

Finalmente, conforme demonstrado em [Knu69, seção 1.2]

$$\left\lfloor \log_\phi S_n \right\rfloor < n - 1 = r + 1.$$

Logo, $r = \left\lfloor \log_\phi S_n \right\rfloor$. □

Para encerrar esta seção, vale a pena ressaltar que embora o algoritmo KMP apresente um comportamento, no pior caso, bem superior ao do algoritmo Ingênuo (respectivamente $O(m+n)$ e $O(mn)$), na prática, conforme veremos na seção B.4, o comportamento de ambos é bastante semelhante. Isto se deve ao fato de que o comportamento do algoritmo KMP é baseado fundamentalmente nas igualdades entre sufixos e prefixos do padrão e, na prática, tais igualdades não são muito freqüentes.

2.4 Variações do Algoritmo KMP

Uma primeira variação do algoritmo KMP surge da observação de que, no início do processamento do texto, não precisamos ter $\mathcal{A}_m$ totalmente calculada, ou seja, não é necessário ter o valor da transição de falha de todos os estados pois, uma vez que a construção de $\mathcal{A}_m$ é efetuada de modo incremental e dado que $\mathcal{A}_j$ é uma extensão de $\mathcal{A}_{j-1}$, então podemos proceder esta construção sob demanda, da seguinte forma: iniciamos o processamento do texto, comparando os caracteres do padrão com os caracteres do texto até obtermos a primeira diferença ou até atingirmos o final do padrão. Supondo que a etapa anterior se encerrou na posição j do padrão então, a partir de $\mathcal{A}_0$, obtemos $\mathcal{A}_1, \ldots, \mathcal{A}_j$, e esta última informa através de $f(j)$ em que estado devemos prosseguir com o processamento do texto. Se $j = m$ então $\mathcal{A}_m$ já foi totalmente obtida e portanto, o processamento prossegue de modo análogo ao descrito na seção 2.2. Porém, se $j < m$ e se posteriormente, durante o processamento do texto, atingirmos o estado final de $\mathcal{A}_j$ então calculamos $\mathcal{A}_{j+1} \ldots \mathcal{A}_k$ onde k é tal que $k = m$ ou $k < m$ e p_{k+1} é diferente do caractere corrente no texto. Em ambos os casos, o valor de $f(k)$ indica em que estado o processamento do texto deve prosseguir sendo que se $k = m$ então uma ocorrência do padrão no texto foi obtida.

A implementação destas idéias está descrita em [Bar81] e em [Tak86], sendo que em [Bar81] também são apresentados alguns resultados de ex-

perimentos realizados com este algoritmo que mostram que, em média, é necessário obter $\mathcal{A}_k$ apenas para pequenos valores de k, ou seja, em média as diferenças ocorrem no início do padrão. Esta variação do KMP, além de permitir, na maioria dos casos, uma economia no tempo necessário para construir a representação abreviada também permite que o processamento do texto se inicie antes mesmo de se conhecer todo o padrão. Assim, em determinadas situações, como por exemplo, quando o padrão é fornecido via teclado, o processamento do texto pode ser efetuado à medida que o usuário fornece o padrão.

Uma outra variação para o algoritmo KMP poderia ser motivada pela seguinte observação: se considerarmos apenas o tempo de processamento do texto então uma maneira eficiente de reconhecer um padrão num texto é processar este texto através de uma representação completa do autômato reconhecedor de padrão. Porém, o grande inconveniente da representação completa, além do espaço necessário para armazená-la, é o tempo gasto para obtê-la.

Observe que estes inconvenientes não ocorrem no caso em que o alfabeto é binário, pois neste caso, cada estado possui apenas duas transições e estas transições podem ser facilmente obtidas a partir de $\mathcal{A}_m$, procedendo da seguinte forma: sabemos que, para todo $j : 1 \leq j \leq m-1$, $p_{1+f(j)} \neq p_{1+j}$. Como Σ é um alfabeto binário então, para todo $a \in \Sigma$, se $a \neq p_{1+j}$ então $a = p_{1+f(j)}$, ou seja, se a causa uma transição de falha num estado $j : 1 \leq j \leq m-1$ então a é o caractere desejável do estado $f(j)$. Portanto, para todo $a \in \Sigma$, temos que se $1 \leq j \leq m-1$ então

$$\delta(j,a) = \begin{cases} j+1 & \text{se } a = p_{1+j} \\ f(j)+1 & \text{caso contrário.} \end{cases}$$

Agora, se $j = m$ então é fácil ver que

$$\delta(m,a) = \begin{cases} f(m)+1 & \text{se } a = p_{1+f(m)} \\ f(f(m))+1 & \text{caso contrário.} \end{cases}$$

Vale ressaltar que, no autômato completo, também deixa de ser necessário o estado -1. Para ilustrar o autômato completo obtido através do método descrito acima, considere $\Sigma = \{a,b\}$ e $p = ababba$. Na figura 2.12 mostramos a representação abreviada $\mathcal{A}_m$ e o autômato completo para este padrão.

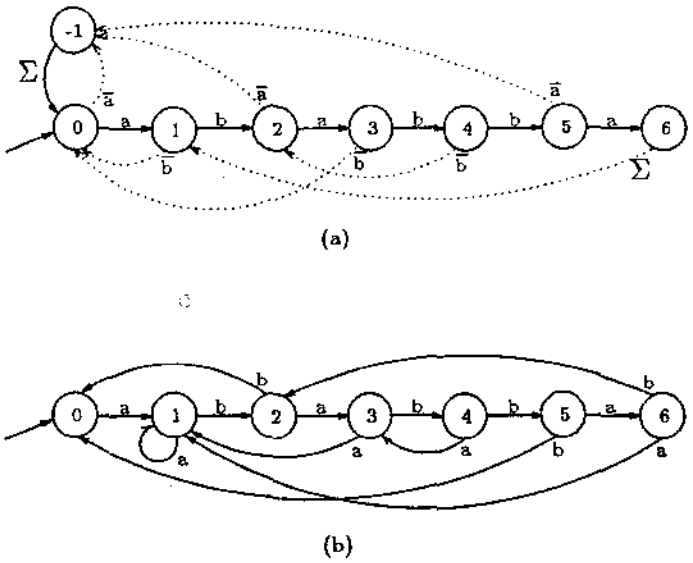


Figura 2.12: (a) Representação abreviada $\mathcal{A}_m$ e (b) autômato completo para $p = ababba$ considerando $\Sigma = \{a, b\}$.

Visto que $\mathcal{A}_m$ pode ser construída em tempo $O(m)$ então o autômato completo também pode ser construído em tempo $O(m)$. Além disso, é fácil ver que se um texto (composto de caracteres de Σ, com $|\Sigma| = 2$) é processado através do autômato completo então o número de comparações (transições) efetuadas é exatamente n.

Lema 2.20 A variação do algoritmo KMP para alfabetos binários descrita acima realiza n comparações durante o processamento do texto t. □

Agora, se o alfabeto Σ é tal que $|\Sigma| > 2$ então podemos adotar uma nova representação que possua as vantagens da representação completa mas, cuja construção não seja tão dispendiosa. A característica básica desta nova representação é incorporar o conceito de transição de falha à representação completa do autômato. Assim, cada estado na nova representação, que denominaremos de *representação estendida*, possuirá $c + 1$ transições, onde c é o tamanho do alfabeto.

A vantagem da utilização da representação estendida é que podemos iniciar o processamento do texto sem que todas as transições estejam previ-

amente calculadas e, durante o processamento do texto, à medida que novas transições forem determinadas elas serão incorporadas à representação estendida. Assim, às custas de um pouco mais de espaço para armazenamento, eliminamos a necessidade de obter o autômato completo, antes do início do processamento de um texto e, ao final deste processamento, apenas as transições que foram efetivamente utilizadas terão sido calculadas.

Agora, vamos descrever mais detalhadamente como construir e utilizar a representação estendida do autômato reconhecedor do padrão $p_1 \ldots p_m$, que indicaremos por $\mathcal{E}_m$. Em $\mathcal{E}_m$ teremos $m+2$ estados, que serão numerados de -1 a m. Em cada estado j, para $0 \leq j \leq m$, teremos $c+1$ transições e no estado -1 teremos apenas uma transição cujas características serão as mesmas de sua correspondente em $\mathcal{A}_m$. No início do processamento do texto, para cada estado j, com $0 \leq j \leq m$, apenas duas transições estarão definidas. Uma dessas transições será rotulada com p_{j+1} e seu valor será $j+1$. A outra será a transição de falha, cujo rótulo indicaremos por @, onde $@ \notin \Sigma$, sendo que seu valor será $f(j)$. As demais $c-1$ transições do estado j, no momento, ficam indefinidas (por exemplo com o valor -2). Ou seja, no início do processamento do texto, $\mathcal{E}_m$ estaria reduzida a $\mathcal{A}_m$. À medida que o texto é processado através de $\mathcal{E}_m$, algumas das transições indefinidas podem vir a ser determinadas e assim, esta transição seria incorporada à representação estendida.

Portanto, no início do processamento do texto, a função de transição de $\mathcal{E}_m$, indicada por $\delta_{\mathcal{E}_m}$, teria os seguintes valores (por conveniência de notação, vamos supor que $\delta_{\mathcal{E}_m}$ está definida para valores (j,a), onde $-1 \leq j \leq m$ e $a \in \Sigma \cup \{@\}$):

$$\delta_{\mathcal{E}_m}(j,a) := \begin{cases} j+1 & \text{se } j = -1 \text{ ou } j \geq 0 \text{ e } a = p_{1+j} \\ f(j) & \text{se } j \geq 0 \text{ e } a = @ \\ -2 & \text{caso contrário.} \end{cases}$$

O processamento do texto t através de $\mathcal{E}_m$ é realizado da seguinte forma: seja j o estado corrente em $\mathcal{E}_m$, seja k a posição corrente no texto e seja $q := \delta_{\mathcal{E}_m}(j, t_k)$. Se $q \neq -2$ então a transição está definida e pode ser normalmente efetuada. Por outro lado, se $q = -2$ então a transição a partir do estado j com o caractere t_k ainda não foi definida e assim, devemos efetuar a transição de falha cujo valor é dado por $\delta_{\mathcal{E}_m}(j, @)$. Neste caso, a cabeça de leitura no texto não deve ser avançada e além disso, antes de atualizar o estado corrente, ou seja, antes de fazer $j := \delta_{\mathcal{E}_m}(j, @)$, devemos

armazenar o valor de j para que posteriormente, possamos definir o valor de $\delta_{\varepsilon_m}(j, t_k)$. Assim, quando uma transição não estiver definida, devemos efetuar os seguintes passos,

```
while q = −2 do
begin
    empilhe j;
    j := δ_{ε_m}(j, @);
    q := δ_{ε_m}(j, t_k);
end;
while pilha não vazia do
begin
    desempilhe j;
    δ_{ε_m}(j, t_k) := q;
end;
```

Depois de efetuados os passos acima, a transição $\delta_{\varepsilon_m}(j, t_k)$ é realizada normalmente avançando a cabeça de leitura no texto, ou seja, podemos fazer $j := q$ e $k := k + 1$.

Supondo que a representação estendida é armazenada através de uma tabela então a figura 2.13 ilustra o processo descrito acima, para $\Sigma = \{a, b, c\}$, $p = abcaabcaba$ e $t = aacabcaabcaaabcaabcabab$.

Como podemos perceber, na maioria das vezes, a representação estendida possuirá várias transições que continuarão indefinidas ao final do processamento do texto. Embora, isso possa sugerir uma má utilização do espaço reservado para armazená-la, podem ocorrer situações em que a utilização deste tipo de representação torna-se vantajosa. Tais situações ocorrem principalmente quando o texto a ser processado possui várias cadeias que se repetem com freqüência. Neste caso, obtém-se uma certa economia no tempo de processamento do texto pois, uma vez percorrida uma seqüência de transições de falha então, em cada estado visitado, passamos a ter uma transição definida para aquele caractere do texto. Assim, evitamos que uma outra ocorrência daquele caractere cause a realização de uma mesma seqüência de transições de falha.

No entanto, vale ressaltar que tais situações não são muito comuns. Além disso, o espaço necessário para armazenar a representação estendida, que é dependente do tamanho do alfabeto, pode vir a ser consideravelmente maior do que o utilizado para armazenar a representação abreviada. Portanto,

δ	a	b	c	@
-1	0	0	0	-2
0	1	-2	-2	-1
1	-2	2	-2	0
2	-2	-2	3	0
3	4	-2	-2	-1
4	5	-2	-2	1
5	-2	6	-2	0
6	-2	-2	7	0
7	8	-2	-2	-1
8	-2	9	-2	4
9	10	-2	-2	2
10	-2	-2	-2	1

(a)

δ	a	b	c	@
-1	0	0	0	-2
0	1	-2	0	-1
1	1	2	0	0
2	-2	-2	3	0
3	4	-2	-2	-1
4	5	-2	-2	1
5	1	6	-2	0
6	-2	-2	7	0
7	8	-2	-2	-1
8	5	9	-2	4
9	10	-2	-2	2
10	-2	2	-2	1

(b)

Figura 2.13: Tabela da representação estendida do autômato reconhecedor de *abcaabcaba*; (a) antes do processamento do texto e (b) depois do processamento do texto t = *aacabcaabcaaabcaabcabab*.

este tipo de variação do algoritmo KMP tem utilidade limitada a algumas situações particulares em que o tamanho do alfabeto é pequeno e/ou o texto a ser processado é composto de várias repetições.

Finalmente, vale citar que existe uma variação do algoritmo KMP, proposta por Aho e Corasick em [AC75], que permite a busca simultânea de vários padrões no texto. A idéia central deste algoritmo é construir uma representação abreviada semelhante a $\mathcal{A}_m$ que engloba todos os padrões aproveitando os prefixos comuns dos padrões.

Capítulo 3

O Algoritmo de Boyer e Moore (BM)

Neste capítulo apresentaremos o método proposto por R. Boyer e S. Moore [BM77] para reconhecimento de padrões em textos. Este algoritmo, que indicaremos por BM, também se baseia em inicialmente processar o padrão para estabelecer como deslocá-lo em relação ao texto durante o processo de busca. Uma outra semelhança entre os dois algoritmos é que ambos inicialmente posicionam o padrão à esquerda no texto e o deslocam para a direita à medida que diferenças entre o padrão e o texto são detectadas. Porém, existe uma diferença fundamental entre o algoritmo de BM e o algoritmo KMP que é a ordem em que os caracteres do padrão e do texto são comparados. Enquanto, no algoritmo KMP, as comparações ocorrem a partir do início do padrão e prosseguem em direção ao seu final, ou seja, da esquerda para a direita, no algoritmo BM estas comparações ocorrem em sentido contrário iniciando no final do padrão e prosseguindo em direção ao seu início, ou seja, da direita para esquerda. Conforme veremos no próximo capítulo, isto permite que o algoritmo BM seja, em média, mais eficiente do que o algoritmo KMP, apresentando um comportamento sublinear no comprimento do texto. Entretanto, estes dois algoritmos apresentam, no pior caso[1], um comportamento linear semelhante.

A seguir, na seção 3.1, descreveremos como o algoritmo BM efetua a

[1]Em algumas situações o algoritmo BM não apresenta um comportamento linear. Porém, estas situações podem ser contornadas através de pequenas alterações no algoritmo original.

busca do padrão no texto e principalmente, como o padrão é deslocado em relação ao texto durante esta busca. Na seção 3.2, discutiremos um método para obter este deslocamento e finalmente, na seção 3.3 apresentaremos um estudo sobre a complexidade do algoritmo BM.

3.1 Uma Descrição do Algoritmo BM

Vale lembrar que o padrão p é dado por $p_1 \ldots p_m$ e que o texto t é dado por $t_1 \ldots t_n$ e que ambos são seqüências de caracteres em Σ^* cujos comprimentos são respectivamente m e n.

Para obter as ocorrências do padrão p no texto t, o algoritmo BM realiza uma série de *rodadas de verificações* sendo que o objetivo de cada rodada é verificar se uma determinada subseqüência do texto, de comprimento m, é igual ao padrão. A primeira rodada de verificações é realizada com os m primeiros caracteres do texto (aqueles mais à esquerda). Quando uma rodada de verificações se encerra, o padrão é deslocado para a direita, em relação ao texto, e este deslocamento será determinado a partir das informações obtidas na rodada que se encerrou. Após deslocar o padrão, uma nova rodada de verificações é realizada. Este processo continua até que o padrão seja deslocado além da extremidade direita do texto.

Uma rodada de verificações consiste em comparar seqüencialmente, da direita para a esquerda, os caracteres do padrão com os caracteres da subseqüência do texto que está sendo analisada, até obtermos uma diferença ou até ultrapassarmos a extremidade esquerda do padrão, o que equivale a dizer que encontramos uma ocorrência do padrão no texto. Assim, o resultado de uma rodada de verificações pode ser descrito através de um par de valores (j, k) tais que:

- $0 \leq j \leq m$ e $j \leq k \leq n - (m - j)$;
- $p_{j+1} \ldots p_m = t_{k+1} \ldots t_{k+m-j}$;
- se $j > 0$ então $p_j \neq t_k$.

Desta forma, se $j = 0$ então encontramos uma ocorrência do padrão a partir da posição $k + 1$ do texto e, neste caso, diremos que a rodada se encerrou com *sucesso*. Por outro lado, se $j > 0$ então a subseqüência testada não

é igual ao padrão e portanto, diremos que a rodada se encerrou com uma *falha*.

Observe que a essência do algoritmo BM, assim como a do KMP, está em determinar o deslocamento a ser efetuado no padrão após uma rodada de verificações. Vale ressaltar que este deslocamento deve garantir a obtenção de todas as ocorrências do padrão no texto e portanto, um deslocamento mínimo óbvio seria 1. Porém, podemos utilizar as informações obtidas durante a última rodada de verificações para estabelecer alguns limites inferiores que geralmente implicarão num deslocamento maior do que 1.

Nosso objetivo agora será descrever como estabelecer estes limites inferiores. Para tornar a descrição mais clara vamos dividi-la em duas etapas. Na primeira, determinaremos o deslocamento mínimo a ser efetuado após uma rodada de verificações que se encerrou com uma falha e na segunda analisaremos o caso em que a rodada se encerrou com sucesso.

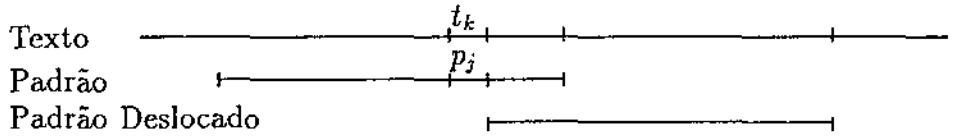
Assim, inicialmente suponha que uma rodada de verificações se encerre com uma falha, ou seja, $j > 0$ e $p_j \neq t_k$. Daí, podemos realizar a seguinte análise: se o caractere t_k não ocorre no padrão então o deslocamento poderia ser de, no mínimo, j posições para a direita, veja figura 3.1 parte (a). Por outro lado, se t_k ocorre no padrão, seja s_1 a posição mais à direita em p tal que $t_k = p_{s_1}$. Isto implica que se $s_1 < j$ então o deslocamento poderia ser de, no mínimo, $j - s_1$ posições, de modo a alinhar o caractere p_{s_1} com t_k, veja figura 3.1 parte (b); mas, se $s_1 > j$ então, pelo menos por enquanto, o deslocamento continuaria a ser igual a 1.

Na realidade, poderíamos estabelecer que s_1 é a maior posição à esquerda de j tal que $p_{s_1} = t_k$. Isto sempre nos permitiria realizar um deslocamento de, no mínimo, $j - s_1$ posições. Mas, para utilizarmos s_1 definido desta forma teríamos que determinar o valor de s_1 para cada $j : 1 \leq j \leq m$ e para cada $a \in \Sigma$ o que torna esta abordagem inviável.

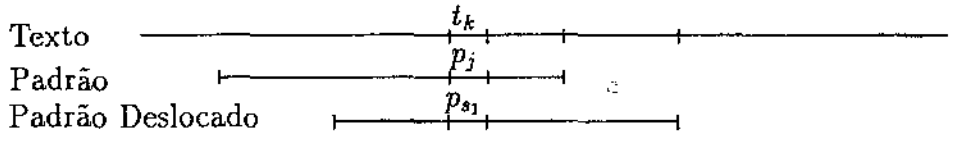
Denotando por $\Delta(j, t_k)$ o deslocamento mínimo a ser efetuado após uma rodada de verificações que se encerra com uma falha, concluímos que um primeiro limite inferior para $\Delta(j, t_k)$ poderia ser obtido em função da posição mais à direita no padrão onde t_k ocorre. Então, seja $s_1 : \Sigma \mapsto \{0, \ldots, m\}$ a função definida por, para todo $a \in \Sigma$,

$$s_1(a) := \max\{s \mid (s = 0) \text{ ou } (1 \leq s \leq m \text{ e } p_s = a)\}.$$

A partir desta definição podemos demonstrar facilmente a afirmação que



(a) $t_k \neq p_j$ e $t_k \notin \{p_1, \ldots, p_m\}$



(b) $p_j \neq t_k$ e $p_{s_1} = t_k$

Figura 3.1: Deslocamento mínimo obtido em função de t_k, sendo que em (a) t_k não ocorre no padrão e em (b) s_1 é a posição mais à direita no padrão onde há uma ocorrência de t_k.

se segue.

Proposição 3.1 Para todo $j : 1 \leq j \leq m$ e para todo $k : j \leq k \leq n-(m-j)$, $\Delta(j, t_k) \geq j - s_1(t_k)$. □

Até este momento, estabelecemos um limite inferior para $\Delta(j, t_k)$ utilizando apenas uma parte das informações fornecidas pela rodada de verificações que se encerrou com uma falha. Agora, vamos obter um outro limite inferior para $\Delta(j, t_k)$ utilizando o fato de que após uma rodada de verificações que se encerra com uma falha, temos que $t_{k+1} \ldots t_{k+m-j} = p_{j+1} \ldots p_m$ e $t_k \neq p_j$, ou seja, vamos obter um outro limite inferior para o deslocamento utilizando o conhecimento adquirido na última rodada de verificações sobre a parte do texto composta por $t_k \ldots t_{k+m-j}$.

Considere a figura 3.2 onde mostramos um deslocamento efetuado após uma rodada de verificações que se encerrou com uma falha. Para que este novo alinhamento possa levar à obtenção de uma ocorrência do padrão no texto é necessário que a parte do padrão deslocado que ficou alinhada com o sufixo $p_{j+1} \ldots p_m$ (no padrão não deslocado) seja igual a este sufixo. Além disso, o caractere p_i no padrão deslocado deve ser diferente de p_j no padrão não deslocado pois já sabemos que $p_j \neq t_k$.

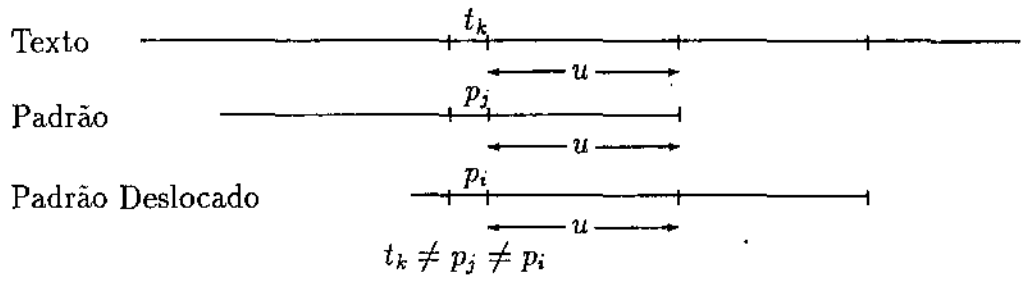


Figura 3.2: Sobreposição entre o padrão não deslocado e o padrão deslocado.

Isto nos permite realizar a seguinte análise: suponha que exista no padrão, uma outra ocorrência do sufixo $p_{j+1} \dots p_m$ e que esta ocorrência seja precedida de um caractere diferente de p_j. Seja i a posição mais à direita no padrão tal que $p_{i+1} \dots p_{i+m-j} = p_{j+1} \dots p_m$ e $p_i \neq p_j$. Então o deslocamento poderia ser de, no mínimo, $j - i$ posições.

Portanto, um segundo limite inferior para $\Delta(j, t_k)$ pode ser obtido conhecendo-se a outra ocorrência do sufixo $p_{j+1} \dots p_m$ mais à direita no padrão que é precedida por um caractere diferente de p_j. Daí, seja $s_2 : \{1, \dots, m\} \mapsto \{0, \dots, m-1\}$ a função definida por, para todo $j : 1 \leq j \leq m$,

$$s_2(j) := \max\{i \mid (i = 0) \text{ ou } (1 \leq i < j \text{ e } p_i \neq p_j \text{ e } p_{i+1} \dots p_{i+m-j} = p_{j+1} \dots p_m)\}.$$

Assim, podemos demonstrar facilmente a afirmação a seguir.

Proposição 3.2 Para todo $j : 1 \leq j \leq m$ e para todo $k : j \leq k \leq n - (m - j)$, $\Delta(j, t_k) \geq j - s_2(j)$. □

A partir das proposições 3.1 e 3.2 podemos estabelecer um limite inferior para $\Delta(j, t_k)$, o qual denotaremos por $d(j, t_k)$, cujo valor é dado por:

$$d(j, t_k) := \max\{j - s_1(t_k), j - s_2(j)\}.$$

Observe que $d(j, t_k)$ é no mínimo 1 (pois, $s_2(j) < j$) e no máximo j. O valor máximo é obtido quando $s_1(t_k) = 0$ ou $s_2(j) = 0$, ou seja, quando t_k não ocorre no padrão ou quando, no padrão, não há uma outra ocorrência

do sufixo $p_{j+1} \ldots p_m$ precedida de um caractere diferente de p_j. Em qualquer uma destas situações o deslocamento estabelecido por $d(j, t_k)$ é de j posições. Após este deslocamento, o prefixo do padrão de comprimento $m - j$ estará alinhado com a parte do texto formada por $t_{k+1} \ldots t_{k+m-j}$. Mas, já sabemos que esta parte do texto é igual a $p_{j+1} \ldots p_m$. Portanto, se $d(j, t_k) = j$ então podemos estabelecer mais um limite inferior para o deslocamento pois, neste caso, o prefixo do padrão alinhado com o texto deve ser um sufixo de $p_{j+1} \ldots p_m$. Veja a figura 3.3. Assim, seja $p_1 \ldots p_s$ o maior prefixo do padrão que é sufixo de $p_{j+1} \ldots p_m$ então o deslocamento pode ser de, no mínimo, $m - s$ posições.

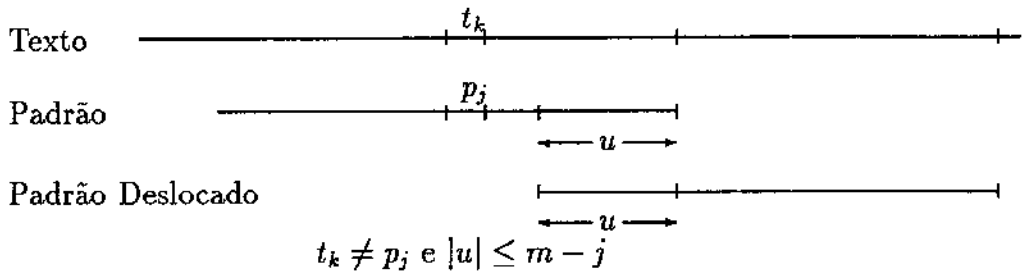


Figura 3.3: Sobreposição entre um prefixo do padrão deslocado e $p_{j+1} \ldots p_m$ no padrão não deslocado quando o deslocamento é de, no mínimo, j posições.

Desta forma, se $d(j, t_k)$ indica um deslocamento mínimo de j posições então um outro limite inferior para $\Delta(j, t_k)$ pode ser obtido a partir do maior prefixo do padrão que é sufixo de $p_{j+1} \ldots p_m$. Assim, seja $s_3 : \{1, \ldots, m\} \mapsto \{0, \ldots, m-1\}$ a função definida por, para todo $j : 1 \leq j \leq m$,

$$s_3(j) := \max\{s \mid s \leq m - j \text{ e } p_1 \ldots p_s = p_{m-s+1} \ldots p_m\}.$$

Isto nos permite demonstrar facilmente a seguinte afirmação.

Proposição 3.3 Para todo $j : 1 \leq j \leq m$ e para todo $k : j \leq k \leq n - (m - j)$, se $d(j, t_k) = j$ então $\Delta(j, t_k) \geq m - s_3(j)$. □

Agora, reunindo os resultados obtidos até o momento e observando que $m - s_3(j) \geq j$, podemos estabelecer que após uma rodada de verificações

que se encerra com uma falha, o deslocamento $\Delta(j, t_k)$ é dado por:

$$\Delta(j, t_k) := \begin{cases} d(j, t_k) & \text{se } d(j, t_k) < j \\ m - s_3(j) & \text{caso contrário.} \end{cases} \tag{3.1}$$

Para finalizar a descrição de como estabelecer o deslocamento a ser efetuado no padrão após uma rodada de verificações, vamos analisar o caso em que uma rodada se encerra com sucesso, ou seja, quando ao final de uma rodada temos $j = 0$, $0 \leq k \leq n - m$ e $p_1 \dots p_m = t_{k+1} \dots t_{k+m}$.

Observe que, neste caso, para obtermos um limite inferior para o deslocamento, não podemos levar em consideração o valor de t_k pois, como não houve uma diferença entre o texto e o padrão então o valor de t_k utilizado nas análises anteriores é irrelevante e mais, pode ser que t_k nem mesmo esteja definido (se $k = 0$). Além disso, como em caso de sucesso temos que $j = 0$ então vamos denotar apenas por Δ_0 o deslocamento a ser efetuado neste caso.

De modo análogo ao algoritmo KMP, quando uma ocorrência do padrão é obtida no texto, o deslocamento pode ser determinado em função da maior borda deste padrão, isto é, do maior prefixo próprio do padrão que também é sufixo deste padrão. Assim, se $p_1 \dots p_s$ é a maior borda do padrão então o deslocamento pode ser de, no mínimo, $m - s$ posições.

Pela definição de s_3, verificamos que o valor de $s_3(1)$ nos fornece o maior prefixo de $p_1 \dots p_{m-1}$ que também é sufixo de $p_2 \dots p_m$. Isto implica que $p_1 \dots p_{s_3(1)}$ é a maior borda do padrão. Portanto, o deslocamento mínimo após uma rodada de verificações que se encerra com sucesso pode ser de $m - s_3(1)$ posições, ou seja,

$$\Delta_0 := m - s_3(1).$$

Até este momento, por motivos de clareza, tratamos separadamente o caso em que uma rodada de verificações se encerra com uma falha do caso em que uma rodada se encerra com sucesso. Agora, podemos adaptar o processo de determinação do deslocamento de modo que, em ambos os casos, o deslocamento seja dado por uma única expressão.

Inicialmente, supondo que os valores de s_1, s_2 e s_3 já foram obtidos para o padrão p, vamos definir que $s_2(0) := s_2(1)$ e $s_3(0) := s_3(1)$. Agora, vamos criar a posição 0 no padrão e no texto e acrescentar uma "sentinela" nestas posições. Em p_0 colocamos um caractere que não ocorre no texto

Algoritmo *BM*;

Entrada : *o padrão p e o texto t.*

Saída : *Lista das posições em t onde o padrão p ocorre.*

```
begin
    {Obtenha s1, s2 e s3.}
    p0 :='#';   {onde # é um caractere que não ocorre no texto}
    t0 := t1;
    s2(0) := s2(1);    s3(0) := s3(1);
    k := m;
    while k ≤ n do
    begin
        j := m;
        while pj = tk-m+j do j := j - 1;
        if j = 0 then write ('O padrão ocorre na posição ',k - m + 1);
        Δ := max{j - s1(tk-m+j), j - s2(j)};
        if Δ = j then Δ := m - s3(j);
        k := k + Δ;
    end
end.
```

Figura 3.4: Uma implementação do algoritmo BM.

e em t_0, um caractere que ocorre no texto, por exemplo t_1. Desta forma, estamos simulando que uma rodada de verificações sempre se encerra com uma "falha", sendo que se esta "falha" é na posição 0 do padrão então na verdade temos um sucesso. Além disso, se $j = 0$ então $j - s_1(t_k) \leq 0$ e $j - s_2(j) = 0$, pois $s_2(0) = s_2(1) = 0$. Assim, por 3.1, temos que $\Delta(j, t_k) = m - s_3(0) = m - s_3(1)$, que é o valor estabelecido na descrição anterior. Uma implementação do algoritmo BM que incorpora estas idéias é dada na figura 3.4.

Desta forma, encerramos a descrição do algoritmo BM e na próxima seção descreveremos como os valores das funções s_1, s_2 e s_3 podem ser

obtidos. Entretanto, vale ressaltar que a descrição do algoritmo BM apresentada nesta seção é uma pequena variação do algoritmo BM original, conforme pode ser verificado através das descrições apresentadas em [BM77], [KMP77], [GO80], [Ryt80], [Baa91] e outros. Isto ocorre porque, no algoritmo BM original, as funções s_2 e s_3 estão unificadas numa única função, que denominaremos s_{23}. Esta unificação pode ser realizada da seguinte forma:

$$s_{23}(j) := \begin{cases} s_2(j) & \text{se } s_2(j) \neq 0 \\ s_3(j) - (m-j) & \text{caso contrário.} \end{cases}$$

Daí, utilizando a estratégia das sentinelas, descrita acima, o deslocamento é dado por

$$\Delta(j, t_k) := \max\{j - s_1(t_k), j - s_{23}(j)\}.$$

Isto implica que, no algoritmo original, o valor de $s_3(j)$ somente é levado em consideração para o cálculo do deslocamento caso $s_2(j) = 0$. Mas, conforme vimos na descrição dada nesta seção, o valor de $s_3(j)$ também pode ser considerado quando $s_1(t_k) = 0$. Esta observação permite que a variação do algoritmo BM descrita nesta seção seja um pouco mais eficiente (efetue menos comparações) do que o algoritmo original.

Para ilustrar a melhoria alcançada pelo método por nós proposto vamos lançar mão de um exemplo extraído de [KMP77] onde $p = badbacbacba$. Seja $\Sigma = \{a, b, c, d, e\}$ e $t = (bacbae)^5$. Agora, considere a figura 3.5 onde mostramos as rodadas de verificações efetuadas pelo algoritmo BM; na parte (a) o deslocamento foi estabelecido utilizando o método original e na parte (b) utilizamos o nosso método. Nesta figura, uma linha contendo apenas o padrão corresponde a uma rodada de verificações e os caracteres comparados durante cada rodada aparecem sublinhados.

Através da figura 3.5 percebemos que após as três primeiras rodadas da parte (a) e após as duas primeiras rodadas da parte (b) o alinhamento entre o padrão e o texto será o mesmo, ou seja, após estas rodadas os deslocamentos totais obtidos nas duas partes serão iguais. Entretanto, na parte (a) foram efetuadas 18 comparações até atingirmos este alinhamento enquanto na parte (b) foram efetuadas apenas 9 comparações. Assim, neste caso particular, o nosso método para estabelecer o deslocamento produz um resultado duas vezes mais eficiente do que o obtido utilizando o método original. No entanto, é necessário um estudo mais detalhado para avaliar mais precisamente o efeito prático desta melhoria nas diversas situações possíveis.

```
b a c b a e b a c b a e b a c b a e b a c b a e b a c b a e
b a d b a c b a c b a
          b a d b a c b a c b a
                    b a d b a c b a c b a
                              b a d b a c b a c b a
```

(a)

```
b a c b a e b a c b a e b a c b a e b a c b a e b a c b a e
b a d b a c b a c b a
              b a d b a c b a c b a
                              b a d b a c b a c b a
```

(b)

Figura 3.5: Rodadas de verificações efetuadas pelo algoritmo BM para procurar o padrão *badbacbacba* no texto $(bacbae)^5$. Na parte (a) o deslocamento é obtido pelo método original e na parte (b) o deslocamento é obtido pelo método por nós proposto.

Vale salientar que o método original nunca produzirá um resultado melhor do que o nosso.

É importante observar que a melhoria a que nos referimos é com relação ao número de comparações efetuadas, pois, a utilização da nossa abordagem implica em um pequeno aumento no espaço requerido para armazenar as tabelas que fornecerão o deslocamento. Enquanto a nossa descrição necessita de três tabelas, uma de tamanho $|\Sigma|$ para armazenar s_1 e duas de tamanhos m para armazenar s_2 e s_3, o algoritmo original necessita de apenas duas tabelas, uma de tamanho $|\Sigma|$ (para s_1) e outra de tamanho m (para s_{23}).

3.2 Um Método para Determinar o Deslocamento

Conforme vimos na seção anterior, o valor do deslocamento a ser efetuado após uma rodada de verificações pode ser determinado através de $s_1(t_k)$, $s_2(j)$ e $s_3(j)$. Vale relembrar que j e k são os valores que descrevem o resultado de uma rodada de verificações, de modo que se $j = 0$ então a rodada se encerrou com sucesso, ou seja, uma ocorrência do padrão foi obtida a partir da posição $k + 1$ do texto. Caso contrário, se $j > 0$ então a rodada se encerrou com uma falha, ou seja, $p_j \neq t_k$ e $p_{j+1} \cdots p_m = t_{k+1} \cdots t_{k+m-j}$.

A função s_1 deve ser obtida para todos os caracteres do alfabeto Σ e o valor de $s_1(a)$ deve indicar a posição mais à direita no padrão onde a ocorre; se a não ocorre no padrão então $s_1(a) = 0$. Assim, os valores de s_1 podem ser facilmente determinados do seguinte modo: inicialmente, fazemos $s_1(a) := 0$ para todo $a \in \Sigma$. Depois, para j variando de 1 a m fazemos $s_1(p_j) := j$.

Para obter os valores da função s_2 vale lembrar que para todo $j : 1 \leq j \leq m$,

$$s_2(j) := \max\{i \mid (i = 0) \text{ ou } (1 \leq i < j \text{ e } p_i \neq p_j \text{ e } p_{i+1} \cdots p_{i+m-j} = p_{j+1} \cdots p_m)\},$$

o que pode ser representado pela figura 3.6.

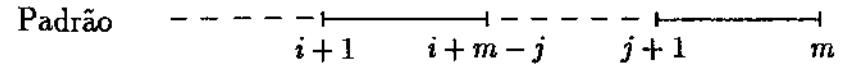


Figura 3.6: O valor de $s_2(j)$ indica o maior i tal que ou $(i = 0)$ ou $(1 \leq i < j$ e $p_i \neq p_j$ e $p_{i+1} \cdots p_{i+m-j} = p_{j+1} \cdots p_m)$.

Vamos agora verificar que as informações de que necessitamos para determinar s_2 podem ser obtidas através da representação abreviada do autômato reconhecedor de p^R (no algoritmo KMP), onde p^R é o reverso do padrão p.

Inicialmente, seja r_2 a função reversa de s_2, isto é, dado $j : 1 \leq j \leq m$,

$$r_2(j) := \min\{i \mid (i = m + 1) \text{ ou } (j < i \leq m \text{ e } p_1 \cdots p_{j-1} = p_{i-j+1} \cdots p_{i-1} \text{ e } p_i \neq p_j)\}.$$

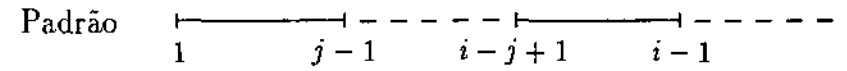


Figura 3.7: O valor de $r_2(j)$ indica o menor i tal que ou $(i = m+1)$ ou $(j < i \leq m$ e $p_1 \ldots p_{j-1} = p_{i-j+1} \ldots p_{i-1}$ e $p_i \neq p_j)$.

A figura 3.7 ilustra a definição de r_2.

Conforme mostraremos a seguir, os valores de s_2 podem ser obtidos facilmente a partir dos valores de r_2, utilizando-se o padrão reverso p^R no lugar de p. Seja

$$p^R := p_m \ldots p_2 p_1$$

o padrão reverso. Assim, para todo $i : 1 \leq i \leq m$,

$$p_i^R = p_{m+1-i}.$$

Denotando por $r_2^R(j)$ o valor de $r_2(j)$ com p^R no lugar de p, temos obviamente que

$$s_2(j) = m + 1 - r_2^R(m + 1 - j).$$

A seguir descreveremos como obter os valores da função r_2.

No capítulo anterior apresentamos a representação abreviada do autômato reconhecedor de padrão, denotada por $\mathcal{B}_m$. Conforme demonstramos no teorema 2.4, para todo $i : 2 \leq i \leq m+1$, $p_1 \ldots p_{g(i-1)}$ é a maior borda de $p_1 \ldots p_{i-1}$. Na realidade, como mostraremos a seguir, $\mathcal{B}_m$ nos fornece todas[2] as bordas de $p_1 \ldots p_{i-1}$.

Proposição 3.4 Seja $i : 2 \leq i \leq m+1$. Então o conjunto das bordas de $p_1 \ldots p_{i-1}$ é dado por $\{p_1 \ldots p_{g^l(i-1)} \mid l \geq 1$ e $g^l(i-1) \geq 0\}$.

Demonstração : Pelo teorema 2.4, $b_0 := p_1 \ldots p_{g(i-1)}$ é a maior borda de $p_1 \ldots p_{i-1}$. Agora, seja b uma borda de $p_1 \ldots p_{i-1}$. Então, conforme podemos verificar facilmente através da figura 3.8, ou $b = b_0$ ou b é uma borda de b_0. Portanto, a afirmação segue por indução. □

[2]Observe que esta propriedade não é válida para $\mathcal{A}_m$. Por isso, para determinar r_2 utilizaremos $\mathcal{B}_m$ e não $\mathcal{A}_m$.

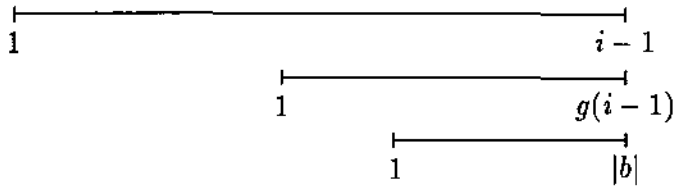


Figura 3.8: b é uma borda de $p_1 \dots p_{i-1}$ se, e somente se, $b = p_1 \dots p_{g(i-1)}$ ou b é uma borda de $p_1 \dots p_{g(i-1)}$.

A partir desta proposição concluímos que os valores de r_2 podem ser obtidos a partir de $\mathcal{B}_m$, pois para todo $l \geq 1$, se $g^l(i-1) \geq 0$ então $p_1 \dots p_{g^l(i-1)} = p_{i-g^l(i-1)+1} \dots p_{i-1}$. Portanto, se $p_{g^l(i-1)+1} \neq p_i$ então $r_2(g^l(i-1)+1) \leq i$. Daí, podemos estabelecer o seguinte método para obter r_2:

```
{Construa B_m para o padrão p}
for j := 1 to m do r2(j) := m + 1;
for i := 2 to m do
begin
   j := g(i − 1) + 1;
   while j ≥ 1 do
   begin
      if p_i ≠ p_j then r2(j) := min{r2(j), i};
      j := g(j − 1) + 1
   end
end
```

Porém se analisarmos mais cuidadosamente este algoritmo verificaremos que ele poderia ser um pouco mais eficiente se considerarmos a seguinte propriedade:

Proposição 3.5 Seja $i : 2 \leq i \leq m$. Se $p_1 \dots p_{k-1}$ é uma borda de $p_1 \dots p_{i-1}$ e $p_k = p_i$ então para todo $j : 1 \leq j \leq k$ temos que $r_2(j) \neq i$.

Demonstração : Pela definição de r_2, se $j = k$ então a afirmação é trivialmente verdadeira. Por outro lado, se $1 \leq j < k$ então $p_1 \dots p_{j-1}$ é uma

borda de $p_1 \ldots p_{i-1}$ se, e somente se, $p_1 \ldots p_{j-1}$ é uma borda de $p_1 \ldots p_{k-1}$; veja figura 3.9. Além disso, $p_j \neq p_i$ se, e somente se, $p_j \neq p_k$.

Portanto, pela definição de r_2, temos que $r_2(j) = m + 1$ ou $r_2(j) \leq k$. Como $m \geq i$ e como $k < i$ então, em ambos os casos, $r_2(j) \neq i$. □

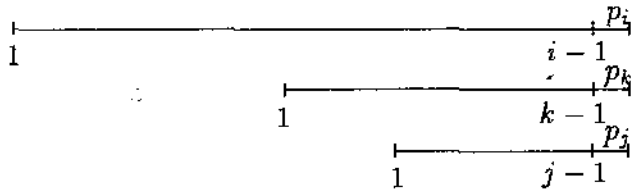


Figura 3.9: $p_1 \ldots p_{k-1}$ e $p_1 \ldots p_{j-1}$ são bordas de $p_1 \ldots p_{i-1}$ e $p_j \neq p_i = p_k$.

Em outras palavras, a proposição acima estabelece que, para um dado $i : 2 \leq i \leq m$, as atualizações em $r_2(g^l(i-1)+1)$ podem ser encerradas assim que obtivermos um $l \geq 1$ tal que ou $g^l(i-1) < 0$ ou $p_{g^l(i-1)+1} = p_i$. Portanto, podemos reescrever o laço repetitivo mais interno, onde os valores de r_2 são atualizados, da seguinte forma (assumimos que a segunda condição no conectivo **and** somente é avaliada se a primeira condição for verdadeira):

```
while j ≥ 1 and p_i ≠ p_j do
begin
    r_2(j) := min{r_2(j), i};
    j := g(j − 1) + 1
end
```

Observe que para um dado $i : 2 \leq i \leq m$, ao final da execução deste trecho do algoritmo, o valor de j é tal que $j = \delta_{\mathcal{B}_m}(g(i-1), p_i) = g(i)$. Como, durante uma iteração i do comando **for**, somente são utilizados valores de g menores do que ou iguais a $g(i-1)$ então a construção de $\mathcal{B}_m$ (obtenção dos valores de g) poderia ser realizada simultaneamente ao cálculo de r_2.

Além disso, vale lembrar que na realidade estamos interessados em calcular s_2 e, conforme vimos anteriormente, esta função pode ser obtida facilmente a partir dos valores de r_2 calculados para o padrão em ordem reversa. Assim, ao invés de referenciarmos as posições i e j no padrão reverso p^R referenciaremos respectivamente as posições $m+1-i$ e $m+1-j$ no padrão

p. Desta forma, o algoritmo definitivo para determinar s_2 é dado na figura 3.10. Este algoritmo também fornece a representação abreviada do autômato reconhecedor de p^R, que denotaremos por $\mathcal{B}_m^R$, sendo que, nesta representação, indicaremos a transição de falha por g_R.

Algoritmo s_2;

Entrada : o padrão p.
Saída : uma tabela com os valores de s_2 e
uma tabela com os valores de g_R $(\mathcal{B}_m^R)$.

begin
  for $j := 1$ **to** m **do** $s_2(j) := 0$;
  $j := 0$; $g_R(0) := -1$;
  for $i := 1$ **to** m **do**
  begin
    while $j \geq 1$ **and** $p_{m+1-j} \neq p_{m+1-i}$ **do**
    begin
      $s_2(m+1-j) := \max\{s_2(m+1-j), m+1-i\}$;
      $j := g_R(j-1)+1$
    end;
    $g_R(i) := j$;
    $j := j+1$
  end
end

Figura 3.10: Uma implementação do algoritmo para calcular s_2 e para obter $\mathcal{B}_m^R$.

Agora vamos descrever como os valores da função s_3 podem ser determinados. Vale lembrar que para $1 \leq j \leq m$ o valor de $s_3(j)$ deve indicar o comprimento do maior prefixo do padrão que é sufixo de $p_{j+1} \ldots p_m$, ou seja, $s_3(j)$ deve indicar o comprimento da maior borda de p cujo comprimento é no máximo $m-j$.

Dado que uma cadeia a é uma borda de uma outra cadeia b se, e somente se, a^R é uma borda de b^R, onde a^R e b^R denotam as cadeias reversas de a e de b respectivamente, então para obter s_3 podemos aproveitar a disponibilidade de $\mathcal{B}_m^R$, que reconhece p^R, lembrando que as bordas de p^R, em ordem decrescente de comprimento, são dadas pela seqüência $g_R(m), g_R^2(m), \cdots$.

Assim, temos facilmente um algoritmo para obter s_3, conforme mostra a figura 3.11.

Algoritmo s_3;

Entrada : uma tabela com os valores de g_R $(\mathcal{B}_m^R)$.
Saída : uma tabela com os valores de s_3.

```
begin
  q := g_R(m);
  for j := 1 to m do
  begin
    if q > m - j then q := g_R(q);
    s_3(j) := q
  end
end
```

Figura 3.11: Uma implementação do algoritmo para calcular s_3.

Como salientamos no final da seção 3.1, no algoritmo BM original, as funções s_2 e s_3 são unificadas na função s_{23}. A partir das definições de s_2 e de s_3 e da descrição de como a unificação é efetuada (definição 3.1) podemos concluir que s_{23} pode ser definida da seguinte forma: para todo $j : 1 \leq j \leq m$,

$$s_{23}(j) := \max\{i \mid (i < j) \text{ e } (i < 1 \text{ ou } p_i \neq p_j) \text{ e } ((i + q < 1) \text{ ou } (p_{i+q} = p_{j+q}) \text{ para } 1 \leq q \leq m - j)\}.$$

Um algoritmo para determinar s_{23} pode ser obtido combinando-se os algoritmos das figuras 3.10 e 3.11 em um único algoritmo. Vale lembrar que

a unificação de s_2 e s_3 permite uma pequena economia no espaço utilizado pelo algoritmo BM (uma tabela de comprimento m a menos) mas, por outro lado, impede a utilização da abordagem apresentada na seção anterior, que proporciona uma pequena melhoria em relação ao algoritmo BM original.

Para finalizar esta seção vamos analisar a complexidade dos algoritmos apresentados. O algoritmo para calcular s_1 é bastante simples sendo que ele requer um espaço $O(|\Sigma|)$ (uma tabela de comprimento $|\Sigma|$) para armazenar os valores de s_1 e obtém estes valores em tempo $O(|\Sigma| + m)$.

O algoritmo para determinar s_2 requer um espaço $O(m)$ (duas tabelas de comprimento m; uma para armazenar $\mathcal{B}_m^R$ e outra para s_2) e conforme vimos nesta seção, este algoritmo, exceto pelos comandos de atualização de s_2, é exatamente igual ao algoritmo para obter $\mathcal{B}_m$ que, de acordo com o estabelecido no capítulo anterior, tem complexidade de tempo $O(m)$. Assim, o algoritmo da figura 3.10 obtém os valores de s_2 em tempo $O(m)$.

Finalmente, o algoritmo da figura 3.11 requer um espaço $O(m)$ (duas tabelas de comprimento m, uma contendo $\mathcal{B}_m^R$ e uma para armazenar s_3). Como podemos perceber facilmente através do algoritmo, os valores de s_3 também são obtidos em tempo $O(m)$.

Isto mostra que o processamento inicial do padrão, realizado para determinar como o padrão deve ser deslocado durante a busca no texto, pode ser efetuado em tempo linear no comprimento do padrão.

A análise de complexidade do algoritmo BM propriamente dito será apresentada na seção a seguir.

3.3 Análise de Complexidade do Algoritmo BM

A análise de complexidade do algoritmo BM é uma questão relativamente complicada, principalmente quando comparada com a análise de complexidade do algoritmo KMP. Isto ocorre porque, durante uma rodada de verificações, a subseqüência do texto que está sendo analisada é comparada com o padrão de forma consecutiva e mesmo que algumas partes desta subseqüência já sejam conhecidas (devido a igualdades obtidas em rodadas anteriores) elas serão novamente comparadas, ou seja, no algoritmo BM, as informações obtidas sobre o texto durante uma rodada de verificações são utilizadas apenas para determinar o deslocamento e depois são "es-

quecidas". Esta característica pode ser facilmente observada quando, por exemplo, desejamos obter todas as ocorrências do padrão a^m no texto a^n. Este exemplo também mostra que, no pior caso, o número de comparações efetuadas pelo algoritmo BM é $O(mn)$, ou seja, no pior caso, o algoritmo BM não é linear no comprimento do texto.

Na verdade, a não linearidade do algoritmo BM somente é verificada quando se deseja obter todas as ocorrências do padrão no texto, pois, conforme veremos nesta seção (corolário 3.13), o número de comparações efetuadas por este algoritmo é $O(n + vm)$, onde v é o número de ocorrências do padrão obtidas no texto. Assim, no caso em que se deseja obter apenas a primeira ocorrência do padrão temos que $v \leq 1$ e portanto, neste caso, o algoritmo é linear no comprimento do texto.

Embora o comportamento do algoritmo BM, no pior caso, não seja linear no comprimento do texto, é interessante analisar o comportamento deste algoritmo em determinadas situações que são mais próximas daquelas que ocorrem na prática. Análises empíricas apresentadas em [BM77], [Smi82] e [HS91], e análises teóricas probabilísticas apresentadas em [Sch88] e [BYGR90], mostram que, no caso médio, o comportamento do algoritmo BM é linear no comprimento do texto e em média este algoritmo é melhor do que o algoritmo KMP. Esta questão será abordada mais detalhadamente no apêndice B.

Além disso, conforme veremos nesta seção, se o padrão não é periódico então o algoritmo BM é linear no comprimento do texto. Assim, podemos concluir que o único caso em que o algoritmo BM não é linear é quando o padrão é periódico e se deseja obter todas as ocorrências deste padrão no texto (veja exemplo acima). Entretanto, conforme apresentado em [Gal79], é possível modificar ligeiramente o algoritmo BM de modo a contornar esta situação, ou seja, com uma pequena modificação o comportamento do algoritmo BM passa a ser linear no comprimento do texto mesmo que o padrão seja periódico. Esta variação do algoritmo BM será descrita na próxima seção.

Uma outra forma de contornar o problema da não linearidade do algoritmo BM é eliminar o "esquecimento" descrito acima fazendo com que o algoritmo seja capaz de se "lembrar" de quais partes do texto foram igualadas a sufixos do padrão. Uma variação do algoritmo BM que incorpora estas idéias foi apresentada em [AG86] e também será descrita na próxima seção.

Nosso objetivo agora é analisar o comportamento do algoritmo BM no pior caso. O primeiro resultado obtido neste sentido foi apresentado por Knuth em [KMP77] onde foi demonstrado que se o padrão não ocorre no texto então o número de comparações efetuadas pelo algoritmo BM não excede $7n$. Na realidade, o resultado demonstrado por Knuth estabelece que o número máximo de comparações é $O(n + vm)$ onde v é o número de ocorrências do padrão no texto. A demonstração elaborada por Knuth é bastante complexa e ele sugere que possa existir outra demonstração que seja mais simples e/ou que forneça um limite superior mais justo, isto é, mais próximo do valor real. Nesta seção apresentaremos uma demonstração que confirma esta suspeita.

Posteriormente, Guibas e Odlyzko em [GO80] conseguiram refinar o resultado obtido por Knuth demonstrando que se o padrão não ocorre no texto então o número de comparações efetuadas não excede a $4n$. Embora a demonstração apresentada em [GO80] seja descrita de forma mais clara que a de Knuth, ela é bastante longa. Neste mesmo artigo os autores conjeturam que o limite superior para o número de comparações deve ser $2n$.

Recentemente, Cole em [Col90] demonstrou que se o padrão não é periódico então o número de comparações efetuadas pelo algoritmo BM não excede $3n - \frac{n}{m}$. A demonstração deste resultado, embora seja um pouco longa, se baseia em argumentos relativamente simples. Estes mesmos argumentos nos permitem demonstrar que se o padrão não é periódico então o número de comparações efetuadas pelo algoritmo BM não excede $4n$. Observe que este resultado é um pouco mais abrangente do que aquele estabelecido por Guibas e Odlyzko, pois, neste caso a hipótese é de que o padrão não é periódico e naquele caso a hipótese é de que o padrão não ocorre no texto.

Nesta seção, descreveremos a demonstração do resultado mais simples ($4n$) apresentado em [Col90]. A nossa escolha se deve ao fato de que a demonstração do resultado mais refinado ($3n - \frac{n}{m}$) é efetuada utilizando-se basicamente as mesmas idéias da demonstração do resultado mais simples. Porém, é realizado um tratamento mais cuidadoso que gera um maior número de casos a serem analisados.

A análise que apresentaremos a seguir se baseia em uma análise de complexidade amortizada. Por isso, inicialmente descreveremos resumidamente os princípios básicos deste tipo de análise que, de acordo com [Tar85], consiste em avaliar o tempo médio consumido pelo algoritmo, no pior caso,

para efetuar uma seqüência de operações. Vale ressaltar que existe uma considerável diferença entre análise de caso médio e análise amortizada. A primeira consiste em estabelecer uma distribuição de probabilidade sobre os dados e operações a serem efetuadas e a partir daí, obtém-se o tempo médio consumido para executar as operações. Por outro lado, a segunda visa avaliar o tempo de execução das operações levando-se em consideração uma seqüência de operações e as possíveis correlações entre estas operações.

Ainda segundo [Tar85], a análise de complexidade amortizada pode ser efetuada seguindo uma das duas abordagens: *visão do banqueiro* ou *visão do físico*. No sentido figurado, a visão do banqueiro corresponde a imaginar que o computador é uma máquina caça-níqueis, sendo que a inserção de uma moeda (*crédito*) permite que o computador funcione durante um determinado tempo. Assim, esta abordagem consiste em alocar um certo número de créditos para cada operação e a partir daí, a meta é mostrar que o total de créditos alocados é suficiente para que todas as operações sejam realizadas. Para isto utilizamos a seguinte estratégia: se uma operação gasta menos do que o crédito alocado para ela então o saldo é depositado numa poupança comum a todas as operações. Por outro lado, se o crédito alocado para uma operação não foi suficiente então esta operação recorre à poupança para quitar sua dívida. Vale ressaltar que é permitido que o saldo da poupança fique temporariamente negativo. Entretanto, ao final da execução de todas as operações este saldo deve ser no mínimo zero.

No caso da visão do físico, que será a abordagem utilizada na análise do algoritmo BM, nós estabelecemos quais elementos (estruturas e operações) do algoritmo serão analisados. Durante a execução do algoritmo, a cada passo, as estruturas selecionadas assumem uma determinada configuração, sendo que uma configuração é alterada quando uma dentre as operações selecionadas é realizada sobre as estruturas.

Assim, a visão do físico consiste em definir uma *função potencial* $\mathcal{P}$ que mapeia uma configuração D em um número real $\mathcal{P}(D)$, que denominaremos o *potencial* de D. Agora, seja D uma configuração assumida em um instante qualquer durante a execução do algoritmo e seja D' a configuração obtida, a partir de D, após a realização de uma ou mais operações. Daí, estas operações causam uma variação no potencial cujo valor é dado por, $\mathcal{P}(D') - \mathcal{P}(D)$. Esta variação no potencial corresponde à diferença entre o tempo amortizado e o tempo real necessários para a realização de tais operações.

Em outras palavras, denotando por a o *tempo amortizado* de um con-

junto de operações que transformam uma configuração D em uma configuração D', temos que:

$$a := \tau + \mathcal{P}(D') - \mathcal{P}(D), \tag{3.2}$$

onde τ é o tempo real gasto para a execução do conjunto de operações.

Portanto, se estimarmos o tempo amortizado consumido pelo algoritmo e a variação do potencial entre a configuração inicial e a configuração final obtida após a execução do algoritmo então teremos uma estimativa do tempo real consumido por este algoritmo.

Passemos agora à análise do algoritmo BM. Como salientamos anteriormente, utilizaremos os conceitos de análise amortizada (visão do físico) para avaliar o comportamento do algoritmo BM, no pior caso, quando o padrão não é periódico. Portanto, a partir de agora assumiremos que o padrão p não é periódico. Além disso, vamos supor que o algoritmo BM efetua z rodadas de verificações e que o tempo real consumido por uma rodada é dado pelo número de comparações efetuadas durante aquela rodada.

Inicialmente, vamos estabelecer que uma configuração D é identificada através de um par de valores l e c, onde l é a posição no texto tal que p_m está alinhado com t_l e c é o número de caracteres do texto que já foram comparados com algum caractere do padrão. Vale ressaltar que, para obter o valor de c, cada caractere do texto que já foi comparado é contado uma única vez, independentemente do número de vezes que ele foi comparado. Assim, definiremos que o potencial de D é dado por:

$$\mathcal{P}(D) := (n - c) + 3(n - l),$$

ou seja, o potencial de D é dado pelo número de caracteres do texto que ainda não foram comparados mais três vezes o número de posições do texto que ainda não foram sobrepostas pelo padrão. Reescrevendo a função potencial temos que:

$$\mathcal{P}(D) = 4n - 3l - c.$$

Agora, seja r, onde $1 \leq r \leq z$, uma rodada de verificações e seja d_r o deslocamento efetuado no padrão após a rodada r. Então o potencial da configuração obtida após o deslocamento d_r, que denotaremos apenas por $\mathcal{P}_r$, é dado por:

$$\mathcal{P}_r = 4n - 3m - 3\sum_{i=1}^{r} d_i - \sum_{i=1}^{r} c_i, \tag{3.3}$$

onde, d_i é o deslocamento após a rodada i e c_i é o número de caracteres do texto que, na rodada i, foram comparados pela primeira vez.

No algoritmo BM, a configuração inicial (antes da primeira rodada de verificações) é tal que p_m está alinhado com t_m ($l = m$) e nenhum caractere do texto ainda foi comparado ($c = 0$). Isto implica que o potencial da configuração inicial, que denotaremos por $\mathcal{P}_0$, é $4n - 3m$.

Por outro lado, a configuração final (após o deslocamento d_z) não pode ser rigorosamente determinada. Entretanto, para qualquer configuração durante a execução do algoritmo, $c \leq n$ e $l \leq n + m$ (na última rodada, p_m pode estar alinhado com t_n e o deslocamento pode ser igual a m). Logo, o potencial associado a qualquer rodada nunca é inferior a $-3m$. Em particular,

$$\mathcal{P}_z \geq -3m,$$

o que nos leva a concluir que

$$\mathcal{P}_0 - \mathcal{P}_z \leq 4n.$$

Agora, como o algoritmo BM efetua z rodadas de verificações então $\tau = \sum_{i=1}^{z} \tau_i$ e $a = \sum_{i=1}^{z} a_i$ onde, τ_i e a_i são respectivamente o tempo real e o tempo amortizado consumidos na execução da rodada i.

Logo, a partir da equação (3.2), temos que

$$\sum_{i=1}^{z} \tau_i \leq 4n + \sum_{i=1}^{z} a_i.$$

Portanto, se mostrarmos que o tempo amortizado total é no máximo zero então obteremos que o tempo real total é no máximo $4n$, ou seja, obteremos que o número de comparações efetuadas pelo algoritmo BM não excede $4n$.

Para provar que $\sum_{i=1}^{z} a_i \leq 0$ mostraremos que $a_r \leq 0$, para todo $r : 1 \leq r \leq z$.

Pela equação (3.2) podemos deduzir que

$$a_r = \tau_r + \mathcal{P}_r - \mathcal{P}_{r-1}.$$

Daí, pela equação (3.3)

$$a_r = \tau_r - 3d_r - c_r,$$

o que implica que $a_r \leq 0$ se, e somente se,

$$\tau_r \leq 3d_r + c_r. \tag{3.4}$$

Inicialmente, vamos mostrar que esta inequação é verdadeira no caso em que a rodada r se encerra com sucesso. Conforme vimos na seção 3.2, se a rodada r se encerra com sucesso então $d_r = m - s_3(1)$. Como o padrão não é periódico então, pelo corolário 1.15, o comprimento da maior borda de p é menor do que $\frac{m}{2}$, o que implica que $s_3(1) < \frac{m}{2}$. Portanto,

$$d_r > \frac{m}{2}.$$

Além disso, temos que $\tau_r = m$ e $c_r \geq 1$ (pelo menos o caractere do texto alinhado com p_m foi comparado pela primeira vez). Daí, $\tau_r < 3d_r + c_r$, o que implica que, neste caso, a inequação 3.4 é verdadeira e portanto, se a rodada se encerra com sucesso então o tempo amortizado da rodada é, no máximo, zero.

Agora, vamos mostrar que a inequação (3.4) é verdadeira também no caso em que a rodada r se encerra com uma falha. Suponha que a rodada r se encerra com uma falha. Assim, sejam j, k e α tais que

$$\begin{aligned} 1 \leq \ j \ &\leq m \\ p_j \ &\neq \ t_k \\ \alpha := t_{k+1} \ldots t_{k+m-j} \ &= \ p_{j+1} \ldots p_m. \end{aligned}$$

Nestas condições, $c_r \geq 1$, pois t_k foi comparado pela primeira vez na rodada r. Além disso, obviamente, $\tau_r = m - j + 1 = |\alpha| + 1$.

Assim, se $3d_r \geq |\alpha|$ então (3.4) é válida. Podemos, portanto supor que

$$3d_r < |\alpha|. \tag{3.5}$$

Seja β o sufixo do padrão cujo comprimento é igual ao deslocamento que seria obtido na rodada r considerando-se apenas as funções s_2 e s_3, isto é,

$$|\beta| := \begin{cases} j - s_2(j) & \text{se } s_2(j) > 0 \\ m - s_3(j) & \text{caso contrário.} \end{cases}$$

Por definição, $d_r \geq |\beta|$. Desta desigualdade e de (3.5) segue que

$$|\alpha| > 3\,|\beta|. \tag{3.6}$$

Tanto α quanto β são sufixos do padrão. Daí, de (3.6) temos que β é também um sufixo de α. Então, seja λ o prefixo de α tal que

$$\alpha = \lambda\beta. \tag{3.7}$$

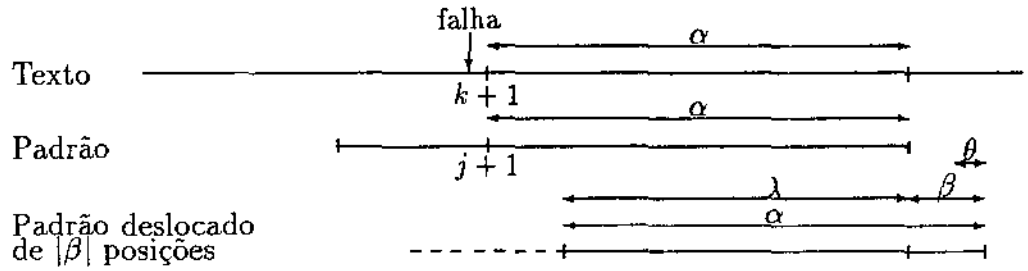


Figura 3.12: Ilustração de α, β, λ e θ.

Finalmente, seja θ o sufixo de comprimento mínimo de β tal que $\beta = \theta^i$ para algum inteiro $i \geq 1$. Obviamente, θ não é periódica perfeita. A figura 3.12 ilustra as cadeias definidas acima.

Lema 3.6 Seja $\gamma\alpha$ um sufixo do padrão tal que $0 \leq |\gamma| \leq |\beta|$. Então $\gamma\alpha$ é periódica e $|\theta|$ é um período de $\gamma\alpha$. Mais precisamente, existe um sufixo δ de θ e um inteiro $l \geq 3$ tal que $\gamma\alpha = \delta\theta^l$.

Demonstração : Vamos primeiramente mostrar que $\gamma\lambda$ é uma borda de $\gamma\alpha$.

Certamente, $\gamma\lambda$ é um prefixo de $\gamma\alpha$, por (3.7). Para provar que $\gamma\lambda$ é também um sufixo de $\gamma\alpha$ vamos considerar dois casos (que são ilustrados pela figura 3.13):

Caso 1 : $|\beta| = j - s_2(j) < j$.

Por definição de s_2,

$$p_{s_2(j)+1} \cdots p_{s_2(j)+|\alpha|} = \alpha,$$

ou seja,

$$p_{j+1-|\beta|} \cdots p_{m-|\beta|} = \alpha.$$

Assim, o sufixo de $\gamma\alpha$ de comprimento $|\gamma\lambda|$ é

$$p_i \cdots p_{m-|\beta|},$$

onde $m - |\beta| - i + 1 = |\gamma\lambda|$. Dado que $|\lambda| + |\beta| = |\alpha| = m - j$, então $i = j + 1 - |\gamma|$. Assim, o sufixo de $\gamma\alpha$ de comprimento $|\gamma\lambda|$ é

$$p_{j+1-|\gamma|} \cdots p_j p_{j+1} \cdots p_{m-|\beta|} = \gamma\lambda.$$

Caso 2: $|\beta| = m - s_3(j)$.

Por definição de s_3,

$$p_1 \cdots p_{s_3(j)} = p_{m-s_3(j)+1} \cdots p_m,$$

ou seja,

$$p_1 \cdots p_{m-|\beta|} = p_{1+|\beta|} \cdots p_m.$$

O sufixo de $p_1 \ldots p_{m-|\beta|}$ de comprimento $|\gamma\lambda|$ é, obviamente, $\gamma\lambda$. Portanto, pela igualdade acima, o sufixo do padrão, de comprimento $|\gamma\lambda|$, é $\gamma\lambda$. Mas, $\gamma\alpha$ é um sufixo do padrão e o seu comprimento é $|\gamma\lambda| + |\beta|$. Logo, $\gamma\lambda$ é um sufixo de $\gamma\alpha$.

Completamos assim a demonstração de que $\gamma\lambda$ é uma borda de $\gamma\alpha$. Além disso, $|\gamma\lambda| = |\gamma\alpha| - |\beta|$. Pela proposição 1.11 (item (ii)), $|\beta|$ é um período de $\gamma\alpha$. Dado que β é um sufixo de $\gamma\alpha$ e que $\beta = \theta^i$, então

$$\gamma\alpha = \delta\theta^l,$$

para algum sufixo próprio δ de θ e para algum inteiro l. Finalmente, de (3.6), $l \geq 3$. □

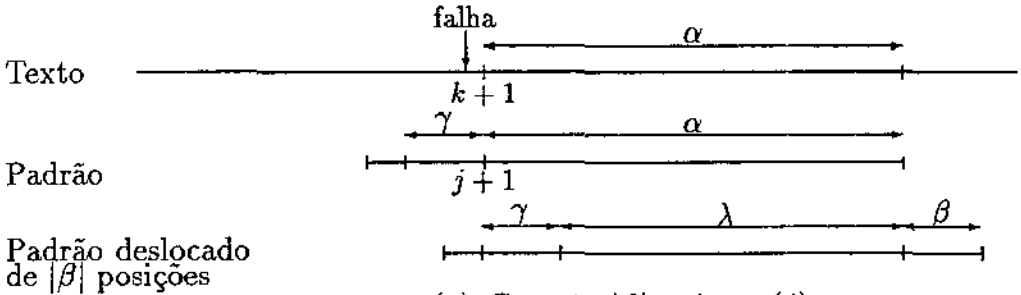
Vamos agora considerar rodadas de verificações r', anteriores à rodada r, em que a posição m do padrão esteja alinhada com uma posição u do texto tal que

$$k + 1 \leq u < k + |\alpha|.$$

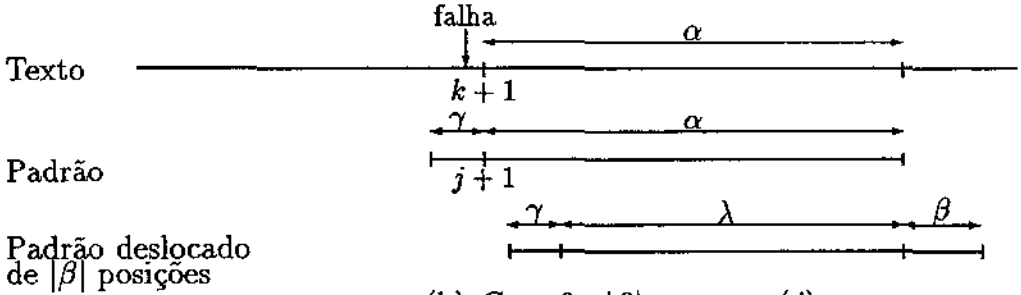
Ou seja, o caractere p_m do padrão está alinhado com o caractere $t_u = \alpha_{u-k}$ de α. Veja figura 3.14.

Denotaremos por j' a posição do padrão alinhada com a posição k do texto.

Vamos agora mostrar três propriedades fundamentais da rodada r'. Inicialmente, lembrando (lema 3.6) que $\alpha = \delta\theta^l$, onde δ é um sufixo de θ, vamos mostrar que se $u \geq k + |\theta|$ então, na rodada r' a posição m do padrão não pode estar alinhada com a última posição de θ, para nenhum θ em α. A figura 3.15 ilustra este fato.



(a) *Caso 1* : $|\beta| = j - s_2(j)$.



(b) *Caso 2* : $|\beta| = m - s_3(j)$.

Figura 3.13: Ilustração da demonstração do lema 3.6.

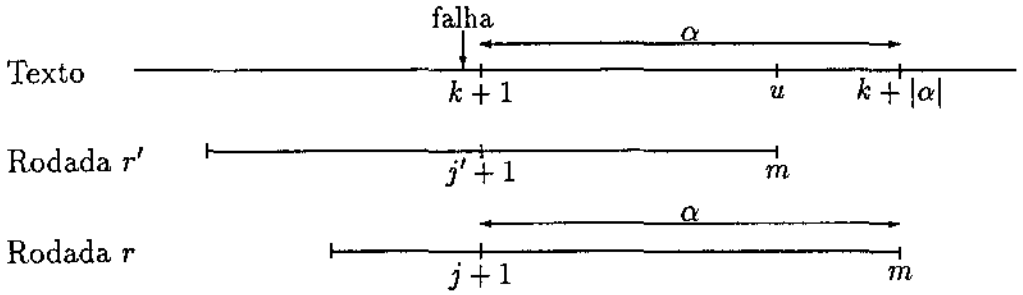


Figura 3.14: Ilustração do alinhamento entre o padrão e o texto na rodada r'.

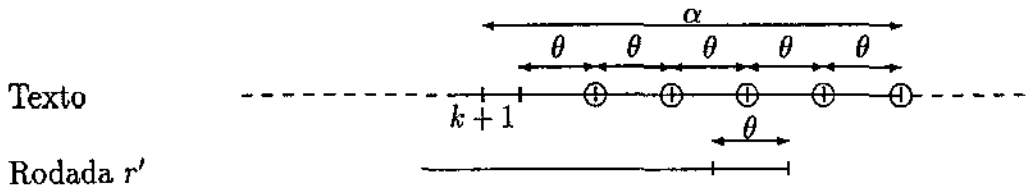


Figura 3.15: As posições marcadas com um círculo indicam as posições do texto com as quais p_m não poderia estar alinhado na rodada r'.

Lema 3.7 Se $u \geq k + |\theta|$ então $u \not\equiv k + |\alpha| \,(\mathrm{mod}\, |\theta|)$.

Demonstração : Suponha, por absurdo, que $u \geq k + |\theta|$ e que $u \equiv k + |\alpha| \,(\mathrm{mod}\, |\theta|)$.

Dado que as posições j e j' do padrão estão alinhadas com a posição k do texto nas rodadas r e r', respectivamente, então

$$j' - j = (k + |\alpha|) - u,$$

pois as posições $k + |\alpha|$ e u do texto estão alinhadas com a posição m do padrão nas rodadas r e r', respectivamente. Pela hipótese de absurdo, o lado direito da igualdade acima é um múltiplo de $|\theta|$. Pelo lema 3.6, com $\gamma := p_j$,

$$p_{j'} \cdots p_m = p_j \cdots p_{m-j'+j}.$$

Isto implica que a rodada r' também se encerra com uma falha na mesma posição k do texto onde ocorre a falha na rodada r.

Dado que r' é anterior a r e que as falhas nas duas rodadas ocorrem na mesma posição k do texto, alinhada, respectivamente, com as posições j' e j do padrão, temos que $d_{r'} \leq j' - j$. Mas, $j' - s_2(j') \leq d_{r'}$, portanto

$$s_2(j') \geq j. \tag{3.8}$$

Logo, por definição de s_2,

$$p_{j'+1} \cdots p_m = p_{s_2(j')+1} \cdots p_{m-j'+s_2(j')} \tag{3.9}$$

e

$$p_{j'} \neq p_{s_2(j')}. \tag{3.10}$$

Pela hipótese de absurdo, $u \geq k + |\theta|$ e portanto, $m \geq j' + |\theta|$. Assim, as cadeias envolvidas na igualdade (3.9) têm comprimento no mínimo $|\theta|$. Levando em conta (3.8) e aplicando o lema 3.6 com $\gamma := p_{j'}$ em (3.9) temos que

ou (i) $j' \equiv s_2(j') (\mathrm{mod}\, |\theta|)$

ou (ii) há uma rotação de θ que é igual a θ.

Por outro lado, aplicando o lema 3.6 (com $\gamma := p_{j'}$) em (3.10) temos que

$$j' \not\equiv s_2(j') (\mathrm{mod}\ |\theta|).$$

Assim, há uma rotação de θ que é igual a θ. Então, pela proposição 1.20, θ é periódica perfeita. Mas, por definição, θ é o menor sufixo de β tal que $\beta = \theta^i$, para algum inteiro i. Portanto, a periodicidade perfeita de θ contradiz a sua escolha. □

Lema 3.8 Na rodada r', no máximo $|\theta|$ comparações entre caracteres do texto e do padrão envolvem posições do texto entre $k+1$ e u. Ademais, se ocorrerem $|\theta|$ dessas comparações, então a $|\theta|$-ésima resulta numa diferença.

Demonstração : Suponha, por absurdo, que

$$u \geq k + |\theta|$$

e que

$$t_{u-|\theta|+1} \ldots t_u = p_{m-|\theta|+1} \ldots p_m = \theta,$$

ou seja, na rodada r' ocorreram $|\theta|$ comparações envolvendo caracteres do texto que estão entre as posições $k+1$ e u e todas resultaram em igualdades. Por definição de θ,

$$t_{k+|\alpha|-|\theta|+1} \ldots t_{k+|\alpha|} = \theta.$$

Pelo lema 3.7,

$$u \not\equiv k + |\alpha| (\mathrm{mod}\ |\theta|).$$

Assim, pelo lema 3.6, com $\gamma := \epsilon$, podemos concluir que há uma rotação de θ que é igual a θ e portanto, θ é periódica perfeita o que contradiz a minimalidade de θ. □

Agora, vamos estabelecer a terceira propriedade da rodada r' mostrando que, nesta rodada, a posição m do padrão está alinhada com uma posição do texto que pertence ou ao prefixo de α de comprimento $|\theta|-1$ ou ao sufixo de α de comprimento $|\theta|$.

Lema 3.9 Ou $u < k + |\theta|$ ou $u > k + |\alpha| - |\theta|$.

Demonstração : Suponha, por absurdo, que existem rodadas r', anteriores a r, tais que $k+|\theta| \leq u \leq k+|\alpha|-|\theta|$. Dentre estas rodadas, escolha aquela que maximiza u.

Dado que $k + |\theta| \leq u$ então pelo lema 3.8 a rodada r' se encerra com uma falha numa posição l do padrão tal que

$$m - l + 1 \leq |\theta| . \tag{3.11}$$

Assim,

$$t_{u-(m-l)+1} \cdots t_u = p_{l+1} \cdots p_m \tag{3.12}$$

e

$$t_{u-(m-l)} \neq p_l. \tag{3.13}$$

Seja

$$d := (k + |\alpha| - u) \bmod |\theta| .$$

Intuitivamente, considerando a cadeia θ que contém t_u, d é a distância entre u e a última posição desta cadeia θ. Veja figura 3.16.

Pelo lema 3.7, $d \neq 0$ o que implica que $1 \leq d \leq |\theta| - 1$. Assim,

$$u < u + d \leq k + |\alpha| - |\theta| \tag{3.14}$$

e

$$u + d \equiv k + |\alpha| \,(\mathrm{mod}\, |\theta|).$$

Além disso,

$$l - d \geq m + 1 - |\theta| - (|\theta| - 1) > j, \tag{3.15}$$

onde a primeira desigualdade segue de (3.11) e segunda segue de $m - j = |\alpha| > 3\,|\beta| \geq 3\,|\theta|$.

Assim, pelo lema 3.6, com $\gamma := \epsilon$, temos que

$$t_{u-(m-l)} \cdots t_{u+d} = p_{l-d} \cdots p_m.$$

Desta igualdade e de (3.12) e (3.13),

$$p_{l+1} \cdots p_m = p_{l-d+1} \cdots p_{m-d} \tag{3.16}$$

e

$$p_l \neq p_{l-d}. \tag{3.17}$$

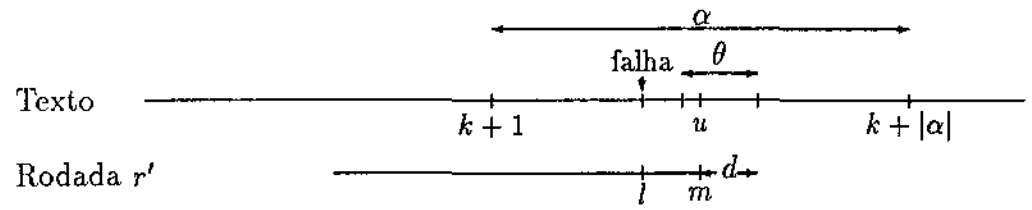


Figura 3.16: Ilustração da demonstração do lema 3.9.

Ademais,

$$t_{u-(m-l)} = p_{l-d}. \tag{3.18}$$

De (3.16) e (3.17) concluímos que $s_2(l) \geq l - d$. De (3.18) concluímos que $s_1(t_{u-(m-l)}) \geq l - d$. Portanto, $d_{r'} = \max\{l - s_1(t_{u-(m-l)}), l - s_2(l)\}$. Por (3.15) temos que $d < l$. Então, podemos afirmar que $d_{r'} \leq d$. Assim, de (3.14), podemos afirmar que na rodada $r' + 1$ a posição m do padrão estará alinhada com uma posição u' do texto tal que $u < u' \leq k + |\alpha| - |\theta|$, em contradição à escolha de r'. □

Lema 3.10 O número de posições do texto entre $k + 1$ e $k + |\alpha|$ cujos caracteres foram comparados com caracteres do padrão antes da rodada r é, no máximo, $3|\theta| - 3$.

Demonstração : Considere uma rodada r' anterior a r que compara caracteres do texto cujas posições estão entre $k+1$ e $k+|\alpha|$. Então o alinhamento entre as posições u do texto e m do padrão satisfaz

$$k + 1 \leq u < k + |\alpha|.$$

Nos casos em que $u < k + |\theta|$ então, dentre as posições que estamos levando em consideração, somente as posições $k + 1, \cdots, k + |\theta| - 1$ podem participar de comparações, isto é, neste caso temos no máximo $|\theta| - 1$ comparações.

Nos casos em que $u \geq k + |\theta|$, pelo lema 3.9, temos que $u > k + |\alpha| - |\theta|$. Nestes casos, pelo lema 3.8, somente as posições de $k + |\alpha| - 2|\theta| + 2$ até $k + |\alpha| - 1$ poderiam vir a ser comparadas, o que fornece um total de $2|\theta| - 2$ posições.

O total combinado de posições do texto entre $k+1$ e $k+|\alpha|$ que podem vir a ser comparadas antes da rodada r não excede portanto a $3|\theta|-3$, e corresponde às posições de $k+1$ a $k+|\theta|-1$ e de $k+|\alpha|-2|\theta|+2$ a $k+|\alpha|-1$. □

Finalmente, podemos demonstrar a validade de (3.4). Na rodada r, foram comparadas $\tau_r = |\alpha|+1$ posições de k a $k+|\alpha|$. No intervalo de $k+1$ a $k+|\alpha|$, pelo menos $|\alpha|-3(|\theta|-1)$ posições foram comparadas pela primeira vez. Assim,

$$c_r \geq \tau_r - 1 - 3(|\theta|-1).$$

Lembrando que $|\theta| \leq |\beta| \leq d_r$, então

$$c_r \geq \tau_r - 3d_r + 2,$$

o que implica que

$$\tau_r < 3d_r + c_r,$$

como queríamos demonstrar.

Portanto, a inequação (3.4) também é verdadeira no caso em que a rodada r se encerra com uma falha. Logo, agrupando os resultados estabelecidos para o caso de sucesso e de falha temos o seguinte resultado:

Teorema 3.11 Se o padrão não é periódico então o número de comparações efetuadas pelo algoritmo BM não excede $4n$. □

Observe que a hipótese de que o padrão não é periódico é utilizada apenas no caso em que a rodada de verificações se encerra com sucesso. Isto nos permite concluir que, qualquer que seja o padrão, se ele não ocorre no texto então o número de comparações efetuadas pelo algoritmo BM não excede $4n$. Assim, podemos estabelecer as seguintes afirmações:

Corolário 3.12 Se o padrão não ocorre no texto então o número de comparações efetuadas pelo algoritmo BM não excede $4n$. □

Corolário 3.13 Se o padrão ocorre v vezes no texto então o número de comparações efetuadas pelo algoritmo BM não excede $4n + 5vm - 8v$.

Demonstração : Por indução em v. Seja $C(n, v)$ o número de comparações efetuadas pelo algoritmo BM para obter as v ocorrências do padrão num texto de comprimento n.

Se $v = 0$ então o padrão não ocorre no texto e portanto, pelo corolário 3.12, $C(n, 0) \leq 4n$.

Suponha por hipótese de indução que se o padrão ocorre $v-1$ vezes num texto de comprimento n então $C(n, v-1) \leq 4n + 5(v-1)m - 8(v-1)$. Agora, considere um texto de comprimento n no qual há v ocorrências do padrão e seja k a posição neste texto onde se inicia a rodada de verificações que obtém a v-ésima ocorrência deste padrão. Assim, no prefixo do texto de comprimento $k-1$, ou seja, em $t_1 \ldots t_{k-1}$, existem $v-1$ ocorrências do padrão. Além disso, a v-ésima ocorrência é obtida na posição $k-m+1$, ou seja, $t_{k-m+1} \ldots t_k = p_1 \ldots p_m$ e portanto, em $t_{k-m+2} \ldots t_n$ não há mais nenhuma ocorrência do padrão, isto é, o padrão não ocorre no sufixo do texto de comprimento $n-k+m-1$. Assim, temos que

$$\begin{aligned} C(n,v) &\leq C(k-1, v-1) + m + C(n-k+m-1, 0) \\ &\overset{H.I.}{\leq} 4(k-1) + 5(v-1)m - 8(v-1) + m + 4(n-k+m-1) \\ &= 4n + 5vm - 8v. \end{aligned}$$

□

3.4 Variações do Algoritmo BM

Desde que foi apresentado em [BM77], o algoritmo BM tem despertado o interesse de vários pesquisadores que o investigam ou com o intuito de analisar a sua complexidade que, conforme vimos na seção 3.3, é um problema intrincado ou com o intuito de obter "novos" algoritmos que apresentem um melhor desempenho. Em função disto, têm surgido diversas variações do algoritmo BM que geralmente apresentam uma ou mais dentre as seguintes características:

- são mais simples do que o algoritmo original e, em média, apresentam um desempenho semelhante ao do algoritmo original;
- a análise de complexidade é bem mais simples do que a do algoritmo original;

- na prática, apresentam um desempenho melhor do que o do algoritmo original.

Nesta seção descreveremos algumas das variações do algoritmo BM, sendo que devido ao grande número de variações existentes, vamos restringir a nossa discussão àquelas que possuem um valor histórico, ou seja, àquelas que contribuíram com novos conceitos que foram utilizados por muitas outras variações.

É importante ressaltar que as variações a serem descritas foram baseadas no algoritmo BM original que, conforme vimos nas seções 3.1 e 3.2, determina o deslocamento através das funções s_1 e s_{23} (unificação de s_2 e s_3). Entretanto, a maioria das modificações propostas nas diversas variações também se aplicam à versão do algoritmo BM apresentada na seção 3.1, onde o deslocamento é determinado através das funções s_1, s_2 e s_3. Como, em nossa opinião, esta versão do algoritmo BM é mais clara do que o algoritmo original então descreveremos as variações do algoritmo BM nos baseando na nossa versão, que não unifica s_2 e s_3.

3.4.1 O Algoritmo de Galil

Este algoritmo, que indicaremos por BM_G, foi uma das primeiras variações do algoritmo BM e foi desenvolvida por Galil [Gal79]. A sua principal característica é que ele contorna em parte o "esquecimento" do algoritmo BM utilizando uma observação muito simples que é a seguinte: no algoritmo BM, quando uma rodada de verificações se encerra com sucesso, o deslocamento do padrão é determinado através da maior borda deste padrão. Vale lembrar que, neste caso, o deslocamento é de $j - s_3(1)$. Como $s_3(1)$ é a maior borda do padrão então $m - s_3(1)$ é o menor período deste padrão. Assim, na próxima rodada de verificações, bastaria compararmos o sufixo do padrão de comprimento $m - s_3(1)$, pois, já sabemos que o prefixo do padrão de comprimento $s_3(1)$ é igual à parte do texto alinhada com ele. Veja figura 3.17. Isto nos permite concluir que numa rodada de verificações efetuada imediatamente após uma rodada que se encerra com sucesso, se o sufixo do padrão de comprimento $m - s_3(1)$ for igual à parte do texto alinhada com ele então esta rodada também se encerra com sucesso.

Conforme veremos a seguir, esta pequena modificação no algoritmo original faz com que o algoritmo BM_G seja linear qualquer que seja o padrão.

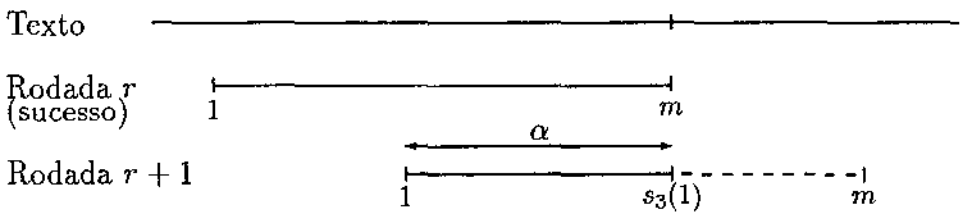


Figura 3.17: α é a maior borda do padrão. Então α é igual à parte do texto alinhada com ele.

Inicialmente, observe que o número de comparações efetuadas pelo algoritmo $\mathrm{BM_G}$ é menor do que ou igual ao número de comparações efetuadas pelo algoritmo BM. Pelo teorema 3.11 e pelo corolário 3.12 temos que se o padrão não é periódico ou se o padrão não ocorre no texto então o número máximo de comparações efetuadas pelo algoritmo BM é $4n$. Conseqüentemente, neste caso, o algoritmo $\mathrm{BM_G}$ também efetua no máximo $4n$ comparações.

Agora, vamos analisar o comportamento do algoritmo $\mathrm{BM_G}$ no caso em que o padrão é periódico e ocorre no texto. Assim, assumindo que o padrão é periódico e que ele ocorre no texto, seja θ o menor elemento periódico do padrão e seja $B := \{r, \ldots, r+u\}$ um conjunto de rodadas de verificações consecutivas que se encerram com sucesso tais que se existe a rodada $r-1$ ou a rodada $r+u+1$ então estas rodadas se encerram com uma falha. Denominaremos o conjunto B de *bloco de ocorrências*. Vamos mostrar que, como o padrão é periódico então duas ocorrências deste padrão no texto, que não pertencem a um mesmo bloco de ocorrências, estão distantes mais do que $\frac{m}{2}$ posições no texto.

Lema 3.14 Sejam $B_1 := \{r_1, \ldots, r_1+u_1\}$ e $B_2 := \{r_2, \ldots, r_2+u_2\}$ dois blocos de ocorrências tais que $r_1+u_1 < r_2$ e sejam k e k' as respectivas posições no texto onde as rodadas r_1+u_1 e r_2 se iniciaram. Se o padrão é periódico então $k'-k > \frac{m}{2}$.

Demonstração : Sabemos que no algoritmo $\mathrm{BM_G}$ o deslocamento é obtido exatamente como no algoritmo BM, ou seja, através das funções s_1, s_2 e s_3.

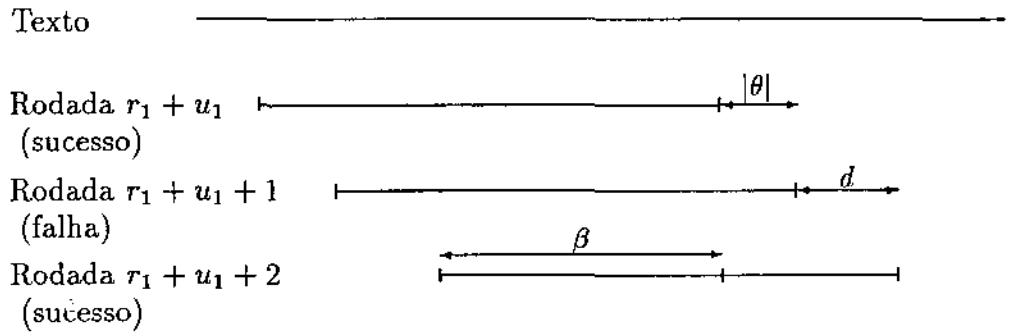


Figura 3.18: Se a rodada $r_1 + u_1 + 2$ se encerra com sucesso então β, onde $|\beta| = m - (|\theta| + d)$, é uma borda do padrão.

Assim, como a rodada $r_1 + u_1$ se encerra com sucesso então o deslocamento do padrão após esta rodada será de $m - s_3(1)$. Seja θ o sufixo do padrão de comprimento $m - s_3(1)$.

Agora, pela definição de bloco de ocorrências, a rodada $r_1 + u_1 + 1$ se encerra com uma falha. Daí, seja t_i o caractere do texto onde houve a falha na rodada $r_1 + u_1 + 1$ e seja d o deslocamento efetuado após esta rodada. Vamos mostrar que se $|\theta| + d \leq \frac{m}{2}$ então a rodada $r_1 + u_1 + 2$ também se encerra com uma falha.

Suponha por absurdo que $|\theta| + d \leq \frac{m}{2}$ e que a rodada $r_1 + u_1 + 2$ se encerra com sucesso. Então, podemos concluir que (veja figura 3.18) existe uma borda no padrão cujo comprimento é $m - (|\theta| + d)$. Daí, pela proposição 1.11 item (ii), temos que $|\theta| + d$ é um período do padrão.

Agora, como $|\theta|$ e $|\theta| + d$ são períodos do padrão e como $|\theta| + |\theta| + d \leq m$ então, pelo corolário 1.18, d também é um período do padrão. Daí, como d é o deslocamento efetuado após a rodada $r_1 + u_1 + 1$ então, na rodada $r_1 + u_1 + 2$, teremos t_i alinhado com um caractere do padrão que é igual àquele alinhado com t_i na rodada $r_1 + u_1 + 1$. Isto implica que a rodada $r_1 + u_1 + 2$ se encerra com uma falha, o que nos leva a uma contradição.

Portanto, uma rodada de verificações que é realizada após a rodada $r_1 + u_1$ e que se encerra com sucesso deve se iniciar numa posição do texto cuja distância em relação ao inicio de $r_1 + u_1$ é maior do que $\frac{m}{2}$. □

Corolário 3.15 Se o padrão é periódico então o número de blocos de ocorrências existentes no texto é no máximo $\frac{2n}{m}$. □

Agora, conforme vimos anteriormente, o algoritmo BM_G é exatamente idêntico ao algoritmo BM original até que a primeira ocorrência do padrão no texto seja obtida, isto é, até atingirmos a primeira rodada de verificações de um bloco de ocorrências. A partir daí, em cada rodada de verificações que pertence àquele bloco de ocorrências, são efetuadas exatamente $m - s_3(1)$ comparações. Visto que o deslocamento, após uma rodada que se encerra com sucesso, é de $m - s_3(1)$ posições então, durante a realização de um bloco de ocorrências, nenhum caractere do texto é comparado mais de uma vez. Isto é, supondo que um bloco de ocorrências é composto por $u+1$ rodadas de verificações então, durante a realização deste bloco de ocorrências ocorrem exatamente $m + u \cdot (m - s_3(1))$ comparações. Depois de encerrado um bloco de ocorrências, o algoritmo BM_G volta a ser idêntico ao algoritmo original até que a próxima ocorrência do padrão no texto seja obtida, isto é, até que o próximo bloco de ocorrências seja atingido.

Assim, temos que o número de comparações efetuadas pelo algoritmo BM_G é igual à soma entre o número de comparações efetuadas pelo algoritmo BM original para obter a primeira ocorrência de cada bloco de ocorrências mais o número de comparações efetuadas na realização de cada bloco.

Como, pelo corolário 3.15, existem no máximo $\frac{2n}{m}$ blocos de ocorrências então, pelo corolário 3.13, o número de comparações efetuadas pelo algoritmo BM original para obter a primeira ocorrência de cada bloco é $O(n + \frac{2n}{m}m) = O(n)$.

Além disso, vimos acima que, em cada bloco de ocorrências, os caracteres do texto são comparados uma única vez. Então o número de comparações efetuadas na realização de todos os blocos é $O(n)$. Assim, temos que:

Teorema 3.16 No algoritmo BM_G ocorrem no máximo $O(n)$ comparações. □

3.4.2 O Algoritmo de Apostolico e Giancarlo

Conforme vimos anteriormente, o algoritmo BM, depois de deslocar o padrão, "esquece" todas as informações que foram obtidas a respeito do texto. Este esquecimento, além de dificultar a análise de complexidade deste algoritmo, também faz com que geralmente sejam efetuadas diversas comparações que poderiam ser evitadas.

O algoritmo BM_G, descrito na seção 3.4.1, aborda parcialmente esta questão, eliminando o esquecimento quando uma rodada de verificações se encerra com sucesso. Entretanto, não é adotado nenhum mecanismo para tentar eliminar ou diminuir o esquecimento caso uma rodada de verificações se encerre com uma falha.

A primeira tentativa de eliminar completamente o esquecimento do algoritmo BM foi proposta em [KMP77] e consiste em construir uma tabela de estados contendo todas as combinações de caracteres do padrão que poderiam vir a ocorrer durante a busca deste padrão num texto qualquer. Assim, cada combinação é representada por um estado da tabela e, durante o processamento do texto, a cada caractere "lido" no texto ocorre uma mudança de estado correspondente à nova combinação obtida. O problema é que o número de estados pode ser muito grande, o que inviabiliza a utilização desta variação do algoritmo BM. Para maiores detalhes sobre esta variação do algoritmo BM vide [KMP77].

Agora, uma outra variação do algoritmo BM que também visa eliminar o esquecimento do algoritmo original foi proposta por Apostolico e Giancarlo [AG86]. A principal característica desta variação, que denotaremos por BM_{AG}, é que, durante o processamento do texto, são armazenadas informações sobre as partes deste texto que se tornam conhecidas e estas informações são posteriormente utilizadas para evitar comparações desnecessárias. Como veremos mais adiante, o algoritmo BM_{AG} é capaz de realizar o reconhecimento do padrão no texto efetuando no máximo $2n - m + 1$ comparações. Além disso, a análise da complexidade deste algoritmo é bastante simples.

Inicialmente, para descrever o algoritmo BM_{AG}, considere a figura 3.19 que ilustra duas rodadas de verificações consecutivas realizadas pelo algoritmo BM. Suponha que na rodada de verificações r obtivemos que $t_{k-m+j+1} \ldots t_k = p_{j+1} \ldots p_m$ e que $t_{k-m+j} \neq p_j$ se $j > 0$. Além disso, suponha que após a rodada r houve um deslocamento de d posições, conforme mostra a figura 3.19. Assim, se na rodada $r + 1$ houverem d comparações

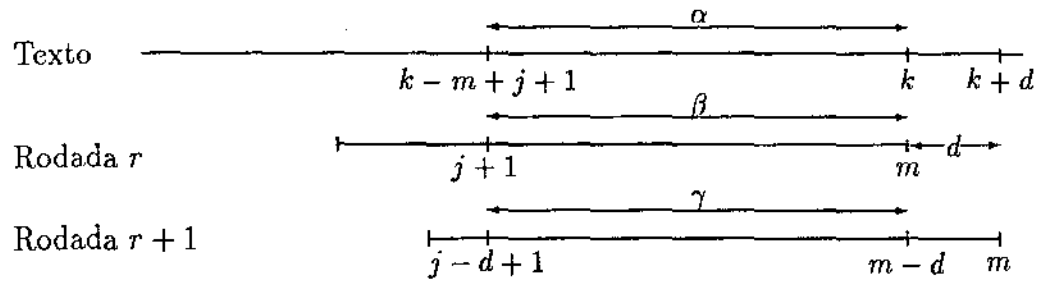


Figura 3.19: Da rodada r temos que $\alpha = \beta$. Daí, se γ é um sufixo do padrão então $\gamma = \beta$ e portanto, $\gamma = \alpha$. Assim, na rodada $r + 1$, não precisamos comparar γ com α.

entre o padrão e o texto e todas resultarem em igualdades então podemos concluir que:

- se $p_{j-d+1} \ldots p_{m-d}$ (γ na figura 3.19) é um sufixo do padrão, ou seja, se $\gamma = \beta$, então a parte do texto $t_{k-m+j+1} \ldots t_k$ (α na figura 3.19) não precisa ser novamente comparada, pois, da rodada r temos que α é igual ao sufixo do padrão de comprimento $m - j$, ou seja, $\alpha = \beta$, o que implica que $\alpha = \gamma$. Portanto, a rodada de verificações poderia prosseguir a partir da posição $j - d$ no padrão e $k - m + j$ no texto.

- se $p_{j-d+1} \ldots p_{m-d}$ (γ na figura 3.19) não é um sufixo do padrão então $\gamma \neq \beta = \alpha$ e portanto, a rodada $r + 1$ se encerraria com uma falha em alguma posição do texto entre $k - m + j + 1$ e k. Assim, a parte do texto $t_{k-m+j+1} \ldots t_k$ também não precisa ser novamente comparada e a rodada $r + 1$ pode ser encerrada com uma falha.

Observe que no segundo item acima teremos uma rodada de verificações se encerrando com uma falha, porém não teremos precisamente os caracteres do texto e do padrão que causaram esta falha. Isto faz com que seja necessária uma pequena modificação na maneira como o deslocamento do padrão é calculado. Posteriormente, descreveremos esta modificação.

Conforme vimos acima, se conhecermos as repetições dos sufixos do padrão no próprio padrão e as partes do texto que, em cada rodada de

verificações, foram igualadas a sufixos do padrão então podemos eliminar o esquecimento do algoritmo BM. Para armazenar as partes do texto que foram igualadas a sufixos do padrão vamos utilizar um vetor S que, para facilitar a descrição, possuirá um tamanho igual ao comprimento do texto, isto é, $S[1 \ldots n]$. Na realidade, conforme descrito em [AG86], o tamanho de S pode ser igual ao comprimento do padrão.

Daí, se numa rodada de verificações obtivermos que $t_{k-m+j+1} \ldots t_k = p_{j+1} \ldots p_m$, ou seja, se obtivermos que $t_{k-m+j+1} \ldots t_k$ é um sufixo do padrão de comprimento $m - j$ então armazenaremos em $S[k]$ o valor $m - j$.

Além disso, as repetições dos sufixos do padrão no próprio padrão serão descritas através de uma função lógica $Q : \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\} \mapsto \{verdade, falso\}$ que é dada por:

$$Q(u,v) := \begin{cases} verdade & \text{se } (u > v \text{ e } p_{u-v+1} \ldots p_u = p_{m-v+1} \ldots p_m) \text{ ou} \\ & \quad (u \leq v \text{ e } p_1 \ldots p_u = p_{m-u+1} \ldots p_m) \\ falso & \text{caso contrário.} \end{cases}$$

Em outras palavras, o valor de $Q(u, v)$, para $v < u$, indica se $p_{u-v+1} \ldots p_u$ é ou não um sufixo do padrão e, para $v \geq u$, $Q(u, v)$ indica se $p_1 \ldots p_u$ é ou não um sufixo do padrão.

Assim, para um dado $k : 1 \leq k \leq n$ e $j : 1 \leq j \leq m$, se $S[k] > 0$ e $Q(j, S[k]) = verdade$ então uma das situações ocorre:

- se $j > S[k]$ então

$$\begin{aligned} t_{k-S[k]+1} \ldots t_k &= p_{m-S[k]+1} \ldots p_m \\ &= p_{j-S[k]+1} \ldots p_j, \end{aligned}$$

  onde a primeira igualdade é devido à definição de S e a segunda é devido à definição de Q. Portanto, $t_{k-S[k]+1} \ldots t_k = p_{j-S[k]+1} \ldots p_j$.

- se $j \leq S[k]$ então, pela definição de S, temos que

$$t_{k-S[k]+1} \ldots t_k = p_{m-S[k]+1} \ldots p_m,$$

  o que implica que

$$t_{k-j+1} \ldots t_k = p_{m-j+1} \ldots p_m.$$

Agora, pela definição de Q, temos que

$$p_{m-j+1} \cdots p_m = p_1 \cdots p_j.$$

e portanto, $t_{k-j+1} \ldots t_k = p_1 \ldots p_j$.

Desta forma, assumindo que os valores de S e de Q já foram determinados então podemos concluir que, durante uma rodada de verificações, ao atingirmos uma posição k no texto e j no padrão tal que $S[k] > 0$ e $Q(j, S[k]) = \textit{verdade}$ então se $j > S[k]$ a rodada de verificações pode prosseguir a partir das posições $k - S[k]$ no texto e $j - S[k]$ no padrão. Por outro lado, se $j \leq S[k]$ então foi obtida uma ocorrência do padrão no texto a partir da posição $k - j + 1$ e a rodada de verificações se encerra com sucesso. A figura 3.20 ilustra estas situações.

Agora, se $S[k] > 0$ e $Q(j, S[k]) = \textit{falso}$ então a rodada de verificações se encerra com uma falha.

Finalmente, se $S[k] = 0$ então devemos comparar t_k com p_j e caso esta comparação resulte numa igualdade prosseguimos com a rodada de verificações, repetindo o processo descrito acima, nas posições $k - 1$ do texto e $j - 1$ do padrão.

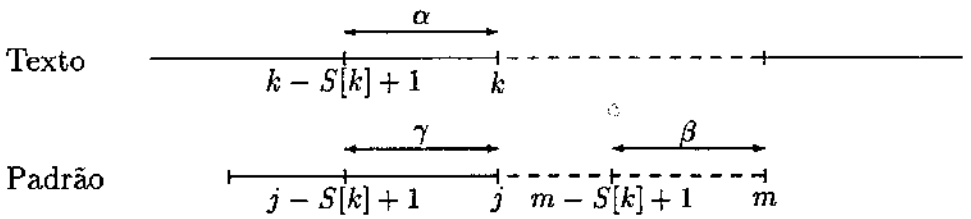
Conforme salientamos anteriormente, no algoritmo BM_{AG}, uma rodada de verificações pode se encerrar com uma falha sem que se conheça precisamente os caracteres do padrão e do texto que causaram esta falha. Este fato faz com que seja necessário um pequeno ajuste na função de deslocamento Δ que foi definida na seção 3.1.

Para adequar a função Δ, cujo valor é dado através de s_1, s_2 e s_3, à nova situação, vamos inicialmente modificar a função s_2. Indicaremos por s_2' a variação da função s_2, sendo que:

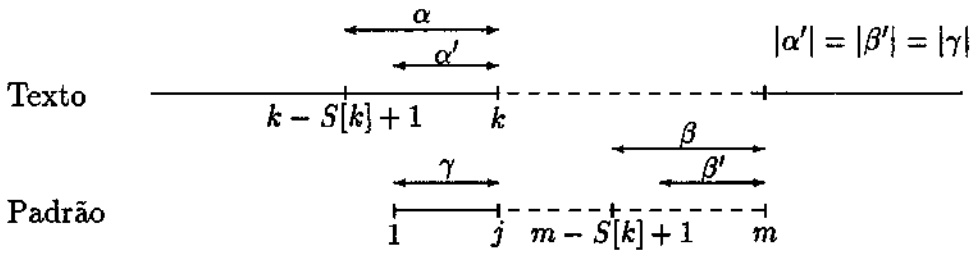
$$s_2'(j) := \max\{i | (i = 0) \text{ ou } (0 < i < j \text{ e } p_{i+1} \cdots p_{i+m-j} = p_{j+1} \cdots p_m)\}.$$

Note que, comparando as definições de s_2 e de s_2', verificamos que para obter s_2' apenas eliminamos a condição $p_i \neq p_j$ da definição de s_2 (lembre-se de que naquela função p_j indicava o caractere do padrão onde houve a falha).

Vale acrescentar que para obter os valores de s_2', podemos utilizar basicamente o mesmo algoritmo que fornece os valores de s_2 (figura 3.10) exceto que no laço repetitivo mais interno devemos eliminar a condição $p_{m-j} \neq p_{m-i}$.



(a) $j > S[k]$. Devido a $S[k]$, $\alpha = \beta$ e devido a $Q(j, S[k])$, $\beta = \gamma$. Daí, $\alpha = \gamma$.



(b) $j \leq S[k]$. Devido a $S[k]$, $\alpha = \beta$ então $\alpha' = \beta'$. Devido a $Q(j, S[k])$, $\beta' = \gamma$. Daí, $\alpha' = \gamma$.

Figura 3.20: Ilustração do algoritmo $\mathrm{BM_{AG}}$. As partes tracejadas no padrão e no texto indicam as posições comparadas durante uma rodada de verificações até que seja obtido k no texto tal que $S[k] > 0$.

Além da função s_2, deveríamos modificar também a função s_1, visto que o argumento desta função é o caractere do texto onde houve a falha. No entanto, no caso da função s_1, podemos contornar este problema da seguinte forma: quando atingimos uma posição k no texto e uma posição j no padrão tais que $S[k] > 0$ e $Q(j, S[k]) = falso$ então sabemos que há uma falha e que esta falha ocorre entre as posições $k - S[k] + 1$ e k do texto (veja figura 3.20). Como a rodada de verificações se processa da direita para a esquerda então k é a primeira posição no texto onde esta falha poderia ocorrer. Assim, se sempre utilizarmos t_k como argumento da função s_1 então não é necessário alterar a definição de s_1.

Vale ressaltar que desta forma podem ocorrer situações em que a função s_1 não contribui para o deslocamento (no caso em que $Q(j, S[k]) = falso$ e $t_k = p_j$, ou seja, a falha não é devido a t_k). Mas, nestes casos, as funções s_2' e s_3 garantem um deslocamento de, no mínimo, uma posição.

Finalmente, a função s_3 não necessita de nenhuma alteração, pois ela não depende do caractere do padrão onde houve a falha.

Na figura 3.21, apresentamos uma implementação do algoritmo $\mathrm{BM_{AG}}$ na qual utilizamos Δ' para representar a "nova" função deslocamento, cuja descrição foi dada acima.

Agora, vamos mostrar que o número de comparações efetuadas pelo algoritmo $\mathrm{BM_{AG}}$ é no máximo $2n - m + 1$. Observe que se uma comparação entre um caractere do texto e do padrão resulta numa igualdade então o valor de $S[l]$ (l é a posição do texto onde a rodada de verificações se iniciou) é atualizado ao final da rodada de verificações e engloba este caractere do texto. Desta forma, este caractere do texto não será mais comparado em nenhuma outra rodada de verificações. Assim, cada caractere do texto pode se envolver em, no máximo, uma comparação que resulte em igualdade. Por outro lado, se a comparação resulta em uma diferença então haverá um deslocamento no padrão que será de, no mínimo, uma posição. Isto implica que haverá no máximo $n - m + 1$ diferenças, pois este é o número máximo de deslocamentos possíveis. Portanto, o número máximo de comparações efetuadas pelo algoritmo $\mathrm{BM_{AG}}$ é $2n - m + 1$.

Para encerrar esta seção, vamos descrever como os valores de Q podem ser determinados. Observe que dados u e v tais que $1 \leq u \leq m$ e $1 \leq v \leq m$, $Q(u, v) = verdade$ se, e somente se, ou $v < u$ e $p_{u-v+1} \ldots p_u$ é um sufixo do padrão, isto é, $p_{u-v+1} \ldots p_u$ é uma borda de $p_{u-v+1} \ldots p_m$ ou $v \geq u$ e $p_1 \ldots p_u$ é um sufixo do padrão, isto é, $p_1 \ldots p_u$ é uma borda do padrão. Portanto,

Algoritmo $\mathrm{BM_{AG}}$;

Entrada : *O padrão p e o texto t.*

Saída : *Lista das posições onde o padrão ocorre no texto.*

```
begin
   l := m;
   {l é a posição no texto onde as rodadas de verificações se iniciam.}
   while l ≤ n do
   begin
      j := m;  k := l;
      while (j > 0) and (((S[k] > 0) and Q(j, S[k])) or (p_j = t_k)) do
      begin
         j := j − max(1, S[k]);
         k := k − max(1, S[k])
      end;
      if j ≤ 0 then
      begin
         write ('o padrão ocorre na posição ', l − m + 1);
         j := 0
      end;
      S[l] := m − j;
      l := l + Δ'(j, t_{l−m+j})
   end
end;
```

Figura 3.21: Uma implementação do algoritmo $\mathrm{BM_{AG}}$.

podemos determinar os valores de Q se conhecermos todas as bordas de cada sufixo do padrão.

Agora, através da descrição apresentada na seção 3.2 (proposição 3.4) sabemos que a representação abreviada de um autômato reconhecedor do padrão em ordem reversa ($\mathcal{B}_m^R$) fornece todas as bordas de cada sufixo do padrão.

Assim, para determinar os valores de Q vamos lançar mão de $\mathcal{B}_m^R$ sendo que para tal nos basearemos no lema a seguir. Vale lembrar que a transição de falha de $\mathcal{B}_m^R$ é indicada por g_R e que p^R indica o padrão em ordem reversa.

Lema 3.17 Dados $u : 1 \leq u \leq m$ e $v : 1 \leq v \leq m$, $Q(u,v) = \mathit{verdade}$ se, e somente se, existe $l \geq 1$ tal que

$$\begin{aligned} &u > v \quad \text{e} \quad v = g_R^l(m-u+v) \\ &\qquad \text{ou} \\ &u \leq v \quad \text{e} \quad u = g_R^l(m). \end{aligned}$$

Demonstração : Pela definição de Q temos que $Q(u,v) = \mathit{verdade}$ se, e somente se,

$$\begin{aligned} &u > v \quad \text{e} \quad p_{u-v+1} \cdots p_u = p_{m-v+1} \cdots p_m \\ &\qquad \text{ou} \\ &u \leq v \quad \text{e} \quad p_1 \cdots p_u = p_{m-u+1} \cdots p_m. \end{aligned}$$

Equivalentemente, temos

$$u > v \quad \text{e} \quad p_{m-u+1}^R \cdots p_{m-u+v}^R = p_1^R \cdots p_v^R \tag{3.19}$$

$$\text{ou}$$

$$u \leq v \quad \text{e} \quad p_{m-u+1}^R \cdots p_m^R = p_1^R \cdots p_u^R. \tag{3.20}$$

Assim, (3.19) equivale a dizer que $p_1^R \ldots p_v^R$ é uma borda de $p_1^R \ldots p_{m-u+v}^R$. Portanto, pela proposição 3.4, temos que (3.19) equivale a dizer que existe $l \geq 1$ tal que $v = g_R^l(m-u+v)$.

Por outro lado, (3.20) equivale a dizer que $p_1^R \ldots p_u^R$ é uma borda de $p_1^R \ldots p_m^R$ e portanto, pela proposição 3.4, temos que (3.20) equivale a dizer que existe $l \geq 1$ tal que $u = g_R^l(m)$. □

A partir deste lema, podemos elaborar facilmente um algoritmo que determina os valores de Q. Na figura 3.22, apresentamos uma implementação

Algoritmo *Obtém_Q*;

Entrada : *Tabela com os valores de* g_R, *ou seja,* B_m^R.

Saída : *Tabela com os valores de* Q.

```
begin
   for u := 1 to m do
      for v := 1 to m do Q(u,v) := false;
   for i := 1 to m - 1 do
   begin
      v := g_R(i);
      while v > 0 do
      begin
         Q(m - i + v, v) := true;
         v := g_R(v)
      end;
   end;
   u := g_R(m);
   while u > 0 do
   begin
      for v := u to m do Q(u,v) := true;
      u := g_R(u)
   end
end;
```

Figura 3.22: Uma implementação do algoritmo para determinar os valores de Q.

para este algoritmo. É fácil perceber que a complexidade do algoritmo é $O(m^2)$.

É importante ressaltar que o algoritmo para determinar os valores de Q apresentado acima é bem diferente do algoritmo originalmente proposto em [AG86]. Embora ambos possuam a mesma complexidade, o algoritmo por nós proposto é bem mais simples do que o algoritmo original.

3.4.3 O Algoritmo de Horspool

Nas seções anteriores, vimos variações do algoritmo BM que foram propostas com o intuito de diminuir ou eliminar o "esquecimento" do algoritmo original. Agora, veremos uma variação do algoritmo BM que não se baseia na questão do "esquecimento". Esta variação, que indicaremos por BM_H, foi proposta por Horspool [Hor80] e sua principal característica é que embora ela seja bem mais simples do que o algoritmo BM, na prática o desempenho destes dois algoritmos é muito próximo.

Conforme vimos na seção 3.1, no algoritmo BM, o deslocamento Δ, efetuado após uma rodada de verificações, é determinado através das funções s_1, s_2 e s_3 sendo que as funções s_2 e s_3 são altamente dependentes da ocorrência de repetições no padrão, ou seja, as funções s_2 e s_3 somente contribuem de forma substancial para o deslocamento quando os sufixos do padrão ocorrem em outras partes deste mesmo padrão. Como, na prática, a existência destas "reocorrências" é pouco freqüente então, nestes casos, as funções s_2 e s_3 pouco contribuem para o deslocamento do padrão.

Assim, a idéia central do algoritmo BM_H é obter o deslocamento Δ utilizando apenas a função s_1 sem levar em consideração as funções s_2 e s_3. Isto permite uma economia de $2m$ posições de memória que eram utilizadas para armazenar os valores de s_2 e s_3 (caso s_2 e s_3 estejam unificadas na função s_{23} esta economia é de m posições). Portanto, o espaço utilizado pelo algoritmo BM_H depende apenas do tamanho do alfabeto Σ. Mais importante do que esta economia, é que a etapa inicial do algoritmo (processamento do padrão) torna-se consideravelmente mais simples. Entretanto, é importante ressaltar que as funções s_2 e s_3 são de fundamental importância para garantir que o número de comparações efetuadas pelo algoritmo BM, no pior caso, seja linear no comprimento do texto (veja seção 3.3).

Agora, suponha que o deslocamento Δ a ser efetuado após uma rodada de verificações seja determinado apenas através da função s_1. Assim, se uma

rodada de verificações se encerra com uma falha nas posições k do texto e j do padrão então $\Delta(j, t_k) := j - s_1(t_k)$. Porém, se o deslocamento for definido apenas desta forma, isto pode causar certos problemas ao algoritmo pois, suponha que t_k seja igual a p_i para algum $i : j < i \leq m$. Isto implica que $s_1(t_k) \geq i$ e portanto, $\Delta(j, t_k) < 0$, pois, $\Delta(j, t_k) = j - s_1(t_k) \leq j - i < 0$. Como mostra a figura 3.23, isto pode tornar o algoritmo interminável.

Padrão	a d b a c
s_1	4 2 3 4 5

Texto	a b c a b a c a a b
	a d b a c
	a d b a c
	a d b a c
	⋮

Figura 3.23: Situação interminável que ocorreria se o deslocamento fosse definido por $\Delta(j, t_k) := j - s_1(t_k)$. Os caracteres sublinhados indicam os caracteres comparados em cada rodada de verificações.

Uma solução óbvia para este problema é definir $\Delta(j, t_k) = \max\{1, j - s_1(t_k)\}$. Porém, Horspool observou que poderíamos obter uma outra solução, que no geral é um pouco mais eficiente, se alterarmos ligeiramente a definição e a forma como s_1 é utilizada.

Inicialmente, observe que $s_1(p_m) = m$. Agora, suponha que houve uma falha nas posições j do padrão e k do texto, isto é, $p_j \neq t_k$, com $1 \leq j \leq m$. Além disso, suponha que $t_k = p_m$. Então $j - s_1(t_k) = j - m \leq 0$. Portanto, podemos concluir que o valor de $s_1(p_m)$ é inútil, pois este valor nunca contribui para o deslocamento do padrão.

Para dar utilidade ao valor de $s_1(p_m)$ podemos definir uma "nova" função s_1, que denotaremos por s_{1_H} sendo que: para todo $a \in \Sigma$,

$$s_{1_H}(a) := \max\{s | (s = 0) \text{ ou } (1 \leq s \leq m-1 \text{ e } p_s = a)\}.$$

Isto é, para obter os valores de s_{1_H} consideramos apenas o prefixo do padrão cujo comprimento é $m - 1$. Desta forma, temos que, para todo $a \in \Sigma$,

$s_{1_H}(a) < m$ e portanto, $m - s_{1_H}(a) \geq 1$. Observe ainda que, para todo $a \in (\Sigma - \{p_m\})$, $s_{1_H}(a) = s_1(a)$.

Assim, o problema descrito acima (figura 3.23) pode ser contornado através da função s_{1_H} da seguinte forma: suponha que uma rodada de verificações se inicia na posição l do texto, isto é, suponha que temos t_l alinhado com p_m. Como, para todo t_l, $s_{1_H}(t_l) < m$ então $m - s_{1_H}(t_l) \geq 1$. Portanto, podemos estabelecer que, após uma rodada de verificações que se inicia na posição l do texto, o padrão será deslocado de $m - s_{1_H}(t_l)$ posições.

Vale ressaltar que este deslocamento pode ser efetuado independentemente da rodada se encerrar com uma falha ou com sucesso pois, se $t_l \neq p_m$ então a rodada se encerra com uma falha e $s_{1_H}(t_l) = s_1(t_l)$. Neste caso, pela definição de s_1, sabemos que $m - s_{1_H}(t_l)$ é um deslocamento válido. Por outro lado, se $t_l = p_m$ então o valor de $s_{1_H}(t_l)$ indicará a posição no padrão (se existir) do segundo caractere mais à direita que é igual a t_l. Daí, é claro que para todo $i : s_{1_H}(t_l) < i < m$, $t_l \neq p_i$ e portanto, um deslocamento menor do que $m - s_{1_H}(t_l)$ não poderia levar a uma ocorrência do padrão no texto.

Concluindo, no algoritmo $\mathrm{BM_H}$, após uma rodada de verificações que se inicia com o caractere p_m do padrão alinhado com o caractere t_l do texto, o padrão é deslocado de Δ_H posições onde,

$$\Delta_H(t_l) := m - s_{1_H}(t_l).$$

Uma implementação do algoritmo $\mathrm{BM_H}$ é apresentada na figura 3.24.

Uma característica importante deste algoritmo é que nele não é fundamental que durante uma rodada de verificações as comparações sejam efetuadas em ordem decrescente (da direita para a esquerda), ou seja, a ordem em que as comparações são efetuadas durante uma rodada de verificações pode ser totalmente arbitrária. Esta observação deu origem a muitas outras variações do algoritmo BM, como por exemplo, [Sun90], [HS91], [Rai92]. Na próxima seção descreveremos algumas destas variações.

Como podemos perceber facilmente, o algoritmo $\mathrm{BM_H}$ efetua, no pior caso, $O(mn)$ comparações. Por exemplo, se $p = a^m$ e $t = a^n$.

Entretanto, conforme demonstrado através de análises teóricas e empíricas apresentadas em [Hor80], [BY89b], [BY89a], [HS91], [BYR92] e outros, o número de comparações efetuadas pelo algoritmo $\mathrm{BM_H}$ é, em média, muito próximo ao número de comparações efetuadas pelo algoritmo BM original. Na seção B.4 apresentaremos algumas destas análises que mos-

Algoritmo BM_H;

Entrada : *O padrão p e o texto t.*
Saída : *Lista das posições onde o padrão ocorre no texto.*

```
begin
   for a ∈ Σ do s_1H(a) := 0;
   for i := 1 to m − 1 do s_1H(p_i) := i;
   l := m;
   while l ≤ n do
   begin
      j := l;  k := l;
      while (j > 0) and (p_j = t_k) do
      begin
         j := j − 1;
         k := k − 1;
      end;
      if j = 0 then write (' o padrão ocorre na posição ', k + 1);
      l := l + Δ_H(t_l)
   end
end;
```

Figura 3.24: Uma implementação do algoritmo BM_H.

tram que o algoritmo BM_H assim como o algoritmo BM efetuam, em média, $O(\frac{n}{m})$ comparações, sendo que as constantes de proporcionalidade para estes dois algoritmos são consideravelmente próximas. Assim, levando-se em consideração a simplicidade e eficiência, podemos adiantar que estas análises nos permitem concluir que o algoritmo BM_H, juntamente com o algoritmo BM_S (descrito na próxima seção), são, em geral, os melhores algoritmos conhecidos para o reconhecimento de padrões em texto.

3.4.4 O Algoritmo de Sunday

Esta variação do algoritmo BM, que denominaremos BM_S, se baseia nas mesmas idéias da variação BM_H, ou seja, em BM_S o deslocamento do padrão também é determinado utilizando-se apenas a função s_1. O que difere BM_S de BM_H é o caractere do texto utilizado como argumento da função s_1. Enquanto no algoritmo BM_H o deslocamento do padrão é determinado através do caractere do texto alinhado com p_m, em BM_S este deslocamento é determinado utilizando o caractere do texto que está imediatamente à direita daquele alinhado com p_m. Isto provém da seguinte observação: após qualquer rodada de verificações, o padrão é deslocado de, no mínimo, uma posição e de, no máximo, m posições. Assim, se uma rodada r se inicia na posição l do texto então na rodada seguinte $(r+1)$ o caractere do texto t_{l+1} necessariamente estará alinhado com algum caractere do padrão. Portanto, a rodada $r+1$ somente poderia se encerrar com sucesso se t_{l+1} for igual ao caractere do padrão alinhado com ele. Logo, na rodada r, podemos utilizar o caractere t_{l+1} como argumento da função s_1 e desta forma, podemos deslocar o padrão de $(m+1) - s_1(t_{l+1})$ posições.

Concluindo, após uma rodada de verificações que se inicia na posição l do texto, o algoritmo BM_S desloca o padrão de $\Delta_s(t_{l+1})$ posições onde,

$$\Delta_s(t_{l+1}) := (m+1) - s_1(t_{l+1}).$$

Como, para todo t_{l+1}, $0 \leq s_1(t_{l+1}) \leq m$ então $1 \leq \Delta_s(t_{l+1}) \leq m+1$. Assim, o deslocamento no algoritmo BM_S pode ser ligeiramente maior do que o deslocamento no algoritmo BM_H. Além disso, em BM_S não é necessário modificar a função s_1.

Visto que uma implementação do algoritmo BM_S pode ser obtida facilmente a partir do algoritmo BM_H então não a apresentaremos aqui. Além

disso, em [Sun90] é apresentada uma implementação bastante detalhada deste algoritmo.

Agora, observe que no algoritmo BM_S, assim como no algoritmo BM_H, a ordem em que uma rodada de verificações é efetuada pode ser completamente arbitrária, isto é, não há mais a necessidade de que as comparações sejam efetuadas da direita para a esquerda. Baseado neste fato, em [Sun90] são apresentadas três variações do algoritmo BM_S, denominadas pelo autor de *quick search* (QS), *maximal shift* (MS) e *optimal mismatch* (OM). O que distingue estas variações é a ordem em que as rodadas de verificações são efetuadas.

Em QS, as comparações são efetuadas da esquerda para a direita (ao contrário do algoritmo BM original).

Em MS, as funções s_2 e s_3 são novamente incorporadas ao algoritmo BM_S e assim, a ordem das rodadas de verificações é estabelecida de modo a minimizar os valores de s_2 e de s_3. Observe que a minimização dos valores destas funções implica numa maximização do deslocamento. Para incorporar s_2 e s_3 ao algoritmo BM_S, ou seja, para permitir que s_2 e s_3 possam ser utilizadas independentemente da ordem em que comparações são efetuadas[3] procedemos da seguinte forma: seja π uma permutação de $\{1,\ldots,m\}$ tal que π estabeleça a ordem das comparações durante as rodadas de verificações, isto é, o i-ésimo caractere do padrão a ser comparado é dado por π_i. Assim, se uma rodada de verificações se inicia na posição k do texto e esta rodada se encerra com uma falha na j-ésima comparação então $p_{\pi_j} \neq t_{k-m+\pi_j}$. Além disso, temos que os caracteres p_{π_1}, p_{π_2} até $p_{\pi_{j-1}}$ são iguais aos respectivos caracteres do texto alinhados com eles, ou seja, para todo $i : 1 \leq i < j$, $p_{\pi_i} = t_{k-m+\pi_i}$.

Desta forma, as funções s_2 e s_3 devem ser redefinidas de modo a se adaptarem à nova ordem para as comparações. Como esta ordem é dada pela permutação π então devemos substituir os índices para o padrão, utilizados na definição de s_2 e s_3, pelos índices dados por π. Porém, ao efetuarmos tais substituições devemos incluir algumas restrições nas definições destas funções de modo a impedir a ocorrência de índices inválidos (menores do que 1 ou maiores do que m). Isto faz com que a "nova" definição de s_2 incorpore automaticamente a "nova" definição de s_3. Portanto, neste caso, é mais adequado utilizarmos a função s_{23} (unificação de s_2 e s_3). Assim,

[3]Lembre-se de que s_2 fornece a posição mais à direita das "reocorrências" dos sufixos do padrão e s_3 fornece a maior borda destes sufixos.

vamos redefinir s_{23} de modo a permitir que esta função possa ser utilizada com uma ordem de comparações arbitrária dada por π, ou seja,

$$s_{23}(\pi_j) := \max\{\pi_j - d \mid (d \geq 1) \text{ e } (\pi_j - d < 1 \text{ ou } p_{\pi_j - d} \neq p_{\pi_j}) \text{ e } ((\pi_i - d < 1) \text{ ou } (p_{\pi_i - d} = p_{\pi_i})\ \forall i : 1 \leq i < j)\}.$$

Daí, o deslocamento é dado por

$$\Delta(\pi_j, t_{k-m+\pi_j}) := \max\{\pi_j - s_1(t_{k-m+\pi_j}), \pi_j - s_{23}(\pi_j)\}.$$

Uma implementação de um algoritmo que obtém a função s_{23} para uma ordem arbitrária de comparações pode ser obtida em [Sun90].

Finalmente, em OM, a ordem das rodadas de verificações é determinada a partir da distribuição de probabilidade de que os caracteres do alfabeto ocorram no texto. Assim, conhecida esta distribuição de probabilidade, comparamos primeiro o caractere do padrão que possui a menor probabilidade de ocorrer no texto. Depois, comparamos o caractere do padrão com a segunda menor probabilidade e assim por diante. Desta forma, no caso de rodadas de verificações que se encerram com uma falha, há uma boa possibilidade de obtermos uma diferença entre o padrão e o texto efetuando menos comparações do que seriam necessárias caso a rodada adotasse a ordem original.

Em [Sun90] são apresentados os resultados de análises empíricas efetuadas com o intuito de comparar o desempenho das três variações descritas acima em relação ao algoritmo BM original. Estes resultados mostram que estas três variações são razoavelmente mais eficientes (realizam menos comparações) do que o algoritmo BM original, sendo que OM é a que apresenta um melhor desempenho. Porém, esta análise também mostra que a diferença de desempenho entre o algoritmo BM e estas três variações decresce à medida que o comprimento do padrão aumenta.

Uma outra análise empírica envolvendo as variações descritas acima foi apresentada em [Smi91]. Na realidade, os experimentos efetuados envolveram apenas a variação OM e foram realizados visando avaliar os efeitos da ordem de comparações sobre a eficiência desta variação, ou seja, foram estabelecidas diversas ordens de comparações e avaliou-se o desempenho do algoritmo OM para cada uma das ordens utilizadas. Para determinar a ordem em que as comparações seriam efetuadas foram estabelecidos dois modelos: um modelo estático e um modelo dinâmico.

No modelo estático, a ordem das rodadas de verificações foi determinada a partir de um conhecimento prévio da distribuição de probabilidade de que os caracteres do alfabeto ocorram numa categoria de textos (por exemplo, inglês, francês, código fonte em C). Neste modelo, a ordem é mantida fixa durante a execução do algoritmo OM.

No modelo dinâmico, o algoritmo OM inicia efetuando as rodadas de verificações numa ordem qualquer e durante a execução do algoritmo esta ordem é adaptada. Uma maneira de efetuar esta adaptação é contabilizar o número de falhas causadas por cada caractere do alfabeto e comparar primeiro os caracteres do padrão que mais causaram falhas até aquele momento durante a execução do algoritmo. Uma outra maneira se baseia no fato de que se uma rodada de verificações se encerra com uma falha então podemos supor que naquela posição do padrão há um caractere cuja freqüência no texto é baixa, ou seja, a probabilidade daquele caractere ocorrer no texto é pequena e portanto, esta posição do padrão passa a ocupar a primeira posição na lista de posições a serem comparadas na próxima rodada. Esta forma de adaptação foi a adotada nas avaliações descritas em [Smi91].

Os resultados apresentados em [Smi91] mostram que a eficiência do algoritmo OM é praticamente a mesma em ambos os modelos (estático e dinâmico) e portanto, o algoritmo OM também pode ser utilizado naquelas situações em que não se conhece a distribuição de probabilidade do alfabeto (neste casos, basta utilizar o modelo dinâmico).

Texto	$\cdots$	a	b	d	c	a	$\cdots$
Padrão		a	b	c			
Deslocamento via $s_{1_H}(t_l)$					a	b	c
Deslocamento via $s_1(t_{l+1})$			a	b	c		

Figura 3.25: Diferenças entre os deslocamentos obtidos através de $s_1(t_{l+1})$ e de $s_{1_H}(t_l)$.

Finalmente, neste mesmo artigo ([Smi91]), é apresentado um exemplo (veja figura 3.25), onde o deslocamento obtido através de $s_{1_H}(t_l)$ (algoritmo $\mathrm{BM_H}$) é maior do que o obtido através de $s_1(t_{l+1})$ (algoritmo $\mathrm{BM_S}$). Diante disto, Smith propôs que o deslocamento Δ_s seja dado por:

$$\Delta_s := \max\{(m+1) - s_1(t_{l+1}), m - s_{1_H}(t_l)\}.$$

3.4.5 O Algoritmo de Hume e Sunday

Quando Boyer e Moore apresentaram o algoritmo BM em [BM77], eles também propuseram uma pequena variação considerando questões de implementação. Nesta variação, que denominaremos BM', somente iniciamos uma rodada de verificações quando o caractere p_m do padrão estiver alinhado com um caractere do texto igual a ele. Portanto, antes de iniciarmos uma rodada de verificações, devemos obter um alinhamento entre o padrão e o texto em que esta condição seja satisfeita. É claro que para obter o alinhamento desejado não vamos comparar p_m com o caractere do texto alinhado com ele, pois senão obteremos um algoritmo exatamente idêntico ao algoritmo BM.

Para obter o alinhamento desejado sem efetuarmos comparações entre o padrão e o texto vamos utilizar uma função s_1', que é uma pequena variação da função s_1, sendo que, para todo $a \in \Sigma$,

$$s_1'(a) := \begin{cases} s_1(a) & \text{se } a \neq p_m \\ -(n+1) & \text{caso contrário.} \end{cases}$$

A função s_1' é utilizada apenas para obter o alinhamento desejado enquanto que o deslocamento continua sendo determinado através de s_1, s_2 e s_3. Porém, poderíamos unificar s_1 e s_1' em uma única função que seria utilizada para os dois propósitos.

Através da função s_1', o alinhamento desejado pode ser obtido da seguinte forma: suponha que, no algoritmo BM original, uma rodada de verificações deveria se iniciar na posição k do texto; daí, antes de efetivamente iniciar a rodada de verificações, poderíamos analisar o valor de $s_1'(t_k)$. Observe que, para todo $a \neq p_m$, $s_1'(a) = s_1(a) \geq 0$. Assim, se $s_1'(t_k) = -(n+1)$ então $t_k = p_m$ e este é um alinhamento desejado. Por outro lado, se $s_1'(t_k) \neq -(n+1)$ então $t_k \neq p_m$ o que implica que este não é um alinhamento desejado e, neste caso, podemos verificar outro alinhamento deslocando o padrão de $m - s_1'(t_k)$ posições, ou seja, podemos fazer $k := k + m - s_1'(t_k)$ e repetimos a análise acima até obtermos o alinhamento desejado ou até que $k > n$. No caso em que um alinhamento desejado é obtido então podemos iniciar a rodada de verificações a partir das posições $m-1$ no padrão e $k-1$ no texto. Por outro lado, se o alinhamento desejado não é obtido então podemos concluir que o padrão não ocorre nas próximas posições do texto.

Agora, na análise acima poderíamos evitar as comparações entre $s_1'(t_k)$ e $-(n+1)$ observando que: para todo k tal que $t_k \neq p_m$ temos que $k + m -$

$s'_1(t_k) = k + m - s_1(t_k)$ e, pela definição de s_1, $s_1(t_k) \geq 0$. Como $k \leq n$ então $k + m - s'_1(t_k) \leq k + m \leq n + m$. Por outro lado, se $t_k = p_m$ então $k+m-s'_1(t_k) = k+m+n+1 > n+m$. Assim, $k+m-s'_1(t_k) > n+m$ se, e somente se, o alinhamento desejado é encontrado. Isto nos permite obter o alinhamento desejado de uma forma bastante simples e eficiente, conforme mostra o trecho de programa dado a seguir:

```
{Neste ponto, k é a posição no texto
onde a rodada de verificações se iniciaria.}
while k ≤ n do k := k + s'1(tk);
if k ≤ n + m then
   write('O padrão não ocorre nas próximas posições do texto');
else begin
   k := k − (m + n + 1);
   {Como pm = tk então inicie a rodada de
   verificações com j = m − 1 e k = k − 1.}
```

Para obter o algoritmo BM', basta incluir este trecho no algoritmo BM, imediatamente antes de iniciar uma rodada de verificações.

Baseado no algoritmo BM', Hume e Sunday elaboraram uma taxonomia dos algoritmos mais importantes para reconhecimento de padrões em texto. Nesta taxonomia, apresentada em [HS91], os diversos algoritmos foram classificados em função de como estes algoritmos executam os três componentes básicos do algoritmo BM', que são: como o alinhamento desejado é obtido, em que ordem as rodadas de verificações são efetuadas e como o deslocamento é determinado.

Além disso, neste mesmo artigo, os autores propõem duas variações para o algoritmo BM' que foram denominadas *tuned Boyer Moore* (TBM) e *least cost* (LC). Em ambas, as comparações são efetuadas da esquerda para a direita (ao contrário do algoritmo BM) e o deslocamento é determinado através das função s_1, s_2 e s_3 sendo que s_2 e s_3 são adaptadas à nova ordem para a rodada de verificações (conforme descrito na seção 3.4.4). Entretanto, as duas variações diferem na forma como o alinhamento desejado é obtido. Em TBM o método utilizado é uma otimização do método estabelecido no algoritmo BM'. Em LC, ao invés de obter o alinhamento desejado baseando-se no caractere p_m, utiliza-se o caractere do padrão que possui o mais baixo custo de busca, sendo que este custo é calculado a partir da probabilidade do caractere ocorrer no texto e da posição do caractere no padrão. De

acordo com os resultados obtidos através de análises empíricas realizadas pelos autores, as variações TBM e LC foram cerca de 4.5 vezes mais eficiente do que o algoritmo BM original.

Para encerrar a seção 3.4 é importante ressaltar que na descrição de diversas variações do algoritmo BM referenciamos análises empíricas efetuadas com o intuito de comparar esta(s) variação(ões) com o algoritmo original. Nestas análises, as funções s_2 e s_3 não foram consideradas, pois conforme vimos anteriormente, na prática, elas pouco contribuem para o deslocamento do padrão. Entretanto, esta suposição não é verdadeira quando o alfabeto é pequeno. Por exemplo, no caso de reconhecimento de padrões em seqüências de DNA, onde $|\Sigma| = 4$. Nestes casos, as funções s_2 e s_3 e suas variações contribuem consideravelmente para o deslocamento do padrão (veja [HS91]) e portanto, não podem ser desprezadas.

Capítulo 4

Contribuições e Conclusões

Inicialmente, vale lembrar que, conforme estabelecemos na seção 1.1, os principais objetivos deste trabalho eram descrever e analisar de forma clara e precisa os principais algoritmos para reconhecimento de padrões em texto. Assim, em nossa opinião, os objetivos estabelecidos foram plenamente alcançados. Entretanto, é importante destacar que, além de atingir os objetivos propostos, conseguimos obter alguns resultados interessantes. A seguir relacionaremos aqueles que merecem um maior destaque.

No caso do algoritmo KMP, a utilização efetiva de autômatos na sua descrição (o que é uma contribuição relativamente original), realmente tornou esta descrição consideravelmente mais simples do que aquelas geralmente apresentadas em outras publicações. Além disso, a análise deste algoritmo foi apresentada de forma bem mais clara do que a originalmente proposta em [KMP77]. Em particular, destacamos a demonstração do teorema 2.17 que, na sua forma original, é mais longa e complexa (utiliza conceitos sobre periodicidade).

No caso do algoritmo BM, apresentamos uma variação inédita deste algoritmo que, além de permitir que a descrição se tornasse mais compreensível, em alguns casos, é mais eficiente do que o algoritmo original (conforme vimos através de um exemplo). É importante notar que esta variação nunca apresenta um desempenho pior do que o algoritmo original (futuramente, pretendemos analisar empiricamente a variação proposta para avaliar se, na prática, ela realmente tem um desempenho melhor do que o algoritmo BM).

Agora, conforme o leitor deve ter notado, a análise de complexidade do algoritmo BM é, sem dúvida, um problema longo e complexo. Neste caso, a

nossa principal contribuição foi apresentar detalhadamente a demonstração dos vários resultados auxiliares (além de descrever o conceito de análise amortizada). Em nossa opinião, esta abordagem didática tornou a análise mais compreensível.

Na descrição das variações do algoritmo BM prosseguimos adotando a mesma estratégia de tentar manter a clareza da descrição evitando, ao máximo, utilizar truques de programação (que tornam o algoritmo um pouco mais eficiente), mas que geralmente obscurecem a sua compreensão. Neste caso, a nossa principal contribuição foi a descrição de um algoritmo inédito para obter os valores da função Q (utilizada pelo algoritmo $\mathrm{BM_{AG}}$, seção 3.4.2) que é consideravelmente mais simples do que o algoritmo original.

Agora, analisando o trabalho como um todo, verificamos que conseguimos descrever e analisar de maneira bastante didática os principais algoritmos para reconhecimento de padrões em texto. Isto, por si só, é uma contribuição considerável, principalmente se observarmos que diversos autores relatam que, de certa forma, o que impede uma maior utilização destes algoritmos é exatamente a dificuldade em compreendê-los (na seção 1.1, apresentamos comentários de alguns autores sobre este problema). Portanto, a organização e a descrição clara dos principais algoritmos para reconhecimento de padrões em texto contribui para amenizar um pouco esta situação.

Em nossa opinião, estas foram as principais contribuições oferecidas aos leitores.

Finalmente, o trabalho desenvolvido nos permite estabelecer dois tipos de conclusões, ambas relacionadas à questão da eficiência dos algoritmos. A primeira se refere às análises de pior caso, sendo que a partir delas podemos concluir que, nestas situações, os algoritmos que apresentam um melhor desempenho são os algoritmos KMP e $\mathrm{BM_{AG}}$ que realizam cerca de $2n - m$ comparações. Porém, como o algoritmo $\mathrm{BM_{AG}}$ utiliza algumas funções que são mais onerosas de computar do que a utilizada pelo algoritmo KMP então temos que, considerando o pior caso, o algoritmo KMP é o que apresenta o melhor desempenho.

Por outro lado, considerando as análises de caso médio (teóricas e empíricas) podemos concluir que, na prática, o algoritmo Ingênuo é (quase) tão bom quanto o algoritmo KMP e, mais importante do que isto, é que o algoritmo que invariavelmente (em média, é claro) apresenta o melhor de-

Algoritmo	Pior Caso	Caso Médio
Ingênuo	$(n-m+1)m$	$\frac{c}{c-1}\left(1-\frac{1}{c^m}\right)(n-m+1)$
KMP	$2n-m$	$\left(1+\frac{(c-\Lambda)(c-1)}{c^2(c-\Lambda)-c(c-1)}+O\left(\frac{1}{n}\right)\right)n$ onde $\Lambda=1+\log_\phi(m+1)$ e $\phi=\frac{1+\sqrt{5}}{2}$
BM	$(n-m+1)n^\dagger$	$\frac{1-\frac{1}{c^m}}{(c-1)\left[1-\left(1-\frac{1}{c}\right)^m\right]}(n-m+1)^\ddagger$

Figura 4.1: Complexidade dos algoritmos Ingênuo, KMP e BM.
† se o padrão não é periódico ou se ele não ocorre no texto temos $3n-\frac{n}{m}$;
‡ mais precisamente, este limite se refere ao algoritmo BM_H.

sempenho é o algoritmo BM_H (é importante salientar que o algoritmo BM_S apresenta um desempenho semelhante ao do algoritmo BM_H, porém ele é pouco citado na literatura principalmente pelo fato de ele ser bastante recente). Este fato aliado à simplicidade destes dois algoritmos (BM_H e BM_S) nos levam a concluir que em situações práticas (sem particularidades, como por exemplo quando o alfabeto é pequeno) devemos optar por utilizar um destes algoritmos.

Para encerrar este capítulo apresentamos na figura 4.1 uma tabela contendo a complexidade (número máximo de comparações efetuadas) no pior caso e no caso médio de alguns algoritmos descritos neste trabalho.

Apêndice A

Outros Algoritmos

O objetivo deste apêndice é apresentar de maneira resumida alguns algoritmos para reconhecimento de padrões em textos que foram propostos recentemente. Vale ressaltar que, de certa forma, os algoritmos que apresentaremos (exceto o algoritmo de Karp e Rabin) se baseiam nas idéias dos algoritmos KMP e/ou BM, sendo que estas idéias são combinadas ou refinadas utilizando-se alguns resultados da área de combinatória de palavras.

Na seção A.1 apresentamos o algoritmo de Baeza-Yates, a seção A.2 é dedicada à descrição de um autômato para o algoritmo BM, na seção A.3 descrevemos o algoritmo de Crochemore e Perrin e na seção A.4, o algoritmo de Colussi. Além disso, embora o algoritmo de Karp e Rabin não se baseie exclusivamente em comparações (o que o exclui do nosso centro de atenção), devido ao seu valor histórico decidimos apresentar uma descrição deste algoritmo na seção A.5.

A.1 O Algoritmo de Baeza-Yates

A idéia básica deste algoritmo consiste em combinar os algoritmos KMP e $\mathrm{BM_H}$ da seguinte forma: suponha que $p_1 \ldots p_m$ esteja alinhado com $t_{k+1} \ldots t_{k+m}$, para algum $k : 0 \leq k \leq n - m$. Para verificar se o padrão é igual a esta parte do texto, comparamos da esquerda para a direita (idem KMP) os caracteres destas duas seqüências. Se obtivermos uma diferença, digamos entre p_{j+1} e t_{k+j+1}, então o deslocamento a ser efetuado será determinado pelo máximo entre os deslocamentos fornecidos pelos algoritmos KMP e $\mathrm{BM_H}$, ou seja, lembrando que uma diferença no caractere p_{j+1} cor-

responde a uma falha no estado j em $\mathcal{A}_m$, então o deslocamento será dado por:

$$\max\{j - f(j), m - s_{1_H}(t_{k+m-j})\}.$$

Uma implementação deste algoritmo pode ser obtida em [BY89b] onde o autor ressalta que este algoritmo apresenta as seguintes características:

- no pior caso, são efetuadas $2n$ comparações (devido ao algoritmo KMP), o que é melhor do que o algoritmo BM,
- em média, são efetuadas menos do que n comparações (devido ao algoritmo $\mathrm{BM_H}$), o que é melhor do que o algoritmo KMP, e
- o processamento do padrão é mais simples do que aquele necessário no algoritmo BM.

A.2 Um autômato para o algoritmo BM

Adotando uma estratégia semelhante àquela utilizada no capítulo 2, onde apresentamos o algoritmo KMP, também podemos modelar o algoritmo BM através de um autômato. Esta abordagem foi inicialmente proposta em [KMP77] e posteriormente, Baeza-Yates e Gonnet em [BYGR90] a utilizaram para estabelecer um modelo que permitisse analisar o algoritmo BM e suas variações. Neste sentido, os autores apresentam uma descrição detalhada do autômato BM e inclusive propõem um algoritmo que o constrói.

No autômato BM, cada estado q é caracterizado por $\pi(q)$ que indica a posição do padrão que deve ser comparada com o caractere corrente do texto quando o autômato atinge o estado q. Implicitamente, o estado q informa que $p_{\pi(q)+1} \ldots p_m$ é igual à parte do texto alinhada com ele. Além disso, temos associado a cada estado q os valores $ok \in \{true, false\}$ e $s \in \{0, \ldots, m\}$. Mais precisamente, um autômato BM é descrito por $(Q, \Sigma, \pi, \delta, q_0)$, onde:

- Q é um conjunto finito de estados,
- Σ é um alfabeto finito,
- $\pi : Q \mapsto \{1 \ldots m\}$,
- q_0 é o estado inicial, e

- δ é a função de transição que mapeia $Q \times \Sigma$ numa tripla da forma $(ok, s, q') = \delta(q, a)$, sendo que se definirmos como k $(1 \leq k \leq n-m+1)$ o alinhamento do padrão no texto então,
  - a denota o caractere $t_{k-1+\pi(q)}$,
  - $ok = true$ indica que foi encontrada uma ocorrência do padrão na posição k do texto,
  - s indica o deslocamento a ser efetuado no padrão, e
  - q' indica o estado a ser alcançado.

Note que como estamos interessados em obter todas as ocorrências do padrão no texto então não há um estado final. Uma ocorrência é obtida sempre que $ok = true$.

Para ilustrar este tipo de autômato considere a figura A.1, onde apresentamos o autômato para o padrão $p = aab$. Nesta figura, $*$ indica que não há informação sobre o caractere do texto alinhado com aquele caractere no padrão. O rótulo de cada estado aparece no lado de fora do círculo que representa o estado. Dentro do círculo temos a posição $\pi(q)$ e a informação implícita associada ao estado q. Cada transição é rotulada com o(s) caractere(s) que a ativa(m) e com o valor do deslocamento s que ela provoca. Para facilitar a notação, utilizamos o símbolo x para indicar os caracteres do alfabeto que não ocorrem no padrão, ou seja, $x := \Sigma - \{p_1, \ldots, p_m\}$. Finalmente, se o caractere a leva a uma ocorrência do padrão então, no rótulo da transição $\delta(q, a)$, incluímos o caractere a acompanhado do símbolo !, ou seja, $a!$ indica que se o caractere corrente do texto é a então obtivemos uma ocorrência do padrão no texto.

Conforme descrito em [BYGR90], o autômato BM pode ser construído utilizando um procedimento recursivo que, para cada estado q e para cada transição $\delta(q, a)$, determina:

(i) se $\delta(q, a)$ leva à obtenção de uma ocorrência do padrão,

(ii) o deslocamento provocado por $\delta(q, a)$, e

(iii) o estado alcançado por $\delta(q, a)$.

A operação (i) pode ser realizada em tempo $O(m)$, a operação (ii) em tempo $O(m^2)$ utilizando um método Ingênuo e (iii) em tempo $O(m \log |Q|)$

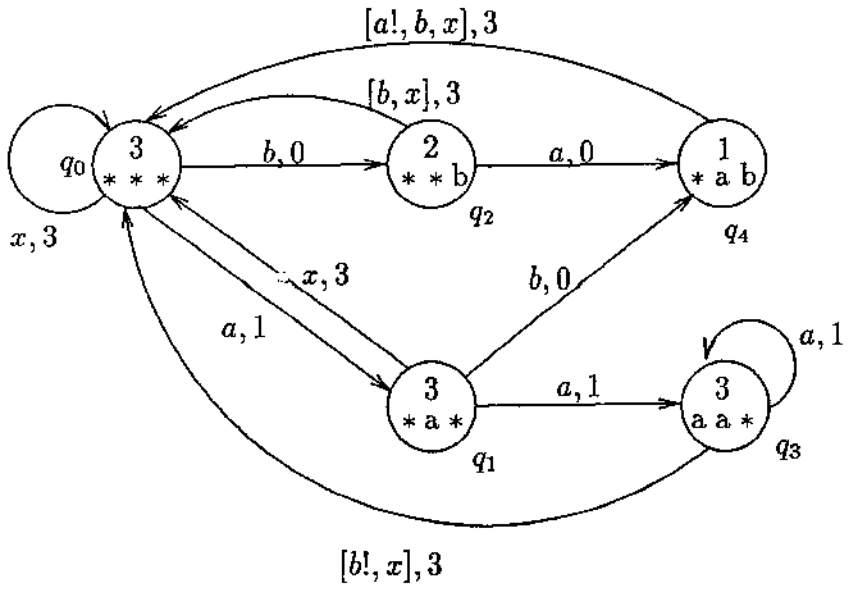


Figura A.1: Autômato que modela o algoritmo BM para o padrão *aab*.

utilizando uma árvore de busca balanceada. Visto que, para todo x (lembre-se de que x não ocorre no padrão), $\delta(q, x)$ leva de volta ao estado inicial então em cada estado podem haver no máximo $m + 1$ transições distintas. Assim, podemos concluir que o algoritmo de Baeza-Yates e Gonnet constrói o autômato BM em tempo

$$O(|Q| \min(m, |\Sigma|)(m^2 + m \log |Q|)).$$

Vale comentar que a questão do limite superior para $|Q|$ (número de estados no autômato BM) se encontra em aberto, ou seja, não se conhece um limite superior preciso para $|Q|$. Sabe-se apenas que

$$m \leq |Q| \leq 2^m - 1,$$

onde o limite superior é obtido calculando-se o número de possíveis subseqüências do padrão exceto o próprio padrão (observe que, neste caso, os caracteres do padrão não precisam aparecer em posições consecutivas da subseqüência; por exemplo $ab * *a * b$ é uma possível subseqüência de *ababaab*).

A questão do limite superior para $|Q|$ é discutida em [BYGR90] e em [Cho90], sendo que os autores demonstram que, para determinadas situações, $|Q|$ é polinomial em m. Em particular, Choffrut mostra que se cada caractere do alfabeto ocorre no padrão no máximo k vezes então o número de estados no autômato é $O(m^{k+1})$.

Assim, como $|Q| < 2^m$ então podemos concluir que o algoritmo de Baeza-Yates e Gonnet constrói o autômato BM em tempo

$$O(|Q| \min(m, |\Sigma|)m^2) \leq O(|Q| m^3).$$

Agora, Choffrut em [Cho90] propõe um algoritmo alternativo para efetuar a operação (ii) acima que permite obter o deslocamento provocado por cada transição em tempo $O(m)$. Desta forma, utilizando este algoritmo, a construção do autômato BM pode ser efetuada em tempo

$$O(|Q| \min(m, |\Sigma|)(m + m \log |Q|)).$$

A.3 O Algoritmo de Crochemore e Perrin

Este algoritmo foi apresentado em [CP91] e basicamente consiste em dividir cada rodada de verificações em duas etapas, sendo que na primeira aplicam-se as idéias do algoritmo KMP e na segunda as idéias do algoritmo BM.

Assim, inicialmente determina-se uma posição l no padrão e depois, se $p_1 \ldots p_m$ está alinhado com $t_{k+1} \ldots t_{k+m}$, para algum $k : 0 \leq k \leq n - m$, então primeiro verificamos se o sufixo do padrão $p_{l+1} \ldots p_m$ é igual a $t_{k+l+1} \ldots t_{k+m}$. Neste caso, as comparações são efetuadas da esquerda para a direita de modo análogo ao algoritmo KMP. Caso esta verificação se encerre com uma falha, digamos entre p_j e t_{k+j}, então o padrão pode ser deslocado de

$$\max\{j - l, s - q + 1\}$$

posições, onde s é o comprimento do maior prefixo do padrão que se igualou ao texto na rodada de verificações anterior e q é o período do padrão.

Por outro lado, se $p_{l+1} \ldots p_m$ é igual a $t_{k+l+1} \ldots t_{k+m}$ então passamos à segunda etapa onde verificamos se o prefixo do padrão $p_1 \ldots p_l$ é igual a $t_{k+1} \ldots t_{k+l}$. Neste caso, as comparações são efetuadas da direita para a esquerda de modo análogo ao algoritmo BM. Ao final desta etapa, o padrão pode ser deslocado de q (período) posições, independentemente da etapa se encerrar com sucesso ou falha.

É importante ressaltar que uma descrição detalhada, assim como uma implementação deste algoritmo, podem ser obtidas em [CP91]. Agora, a principal característica deste algoritmo é que, graças a algumas propriedades que a posição l deve satisfazer, o padrão pode ser deslocado "eficientemente" (evitando várias comparações desnecessárias) sem utilizar a função f do algoritmo KMP e nem as funções s_1, s_2 e s_3 do algoritmo BM.

A seguir estabeleceremos como a posição l é determinada. Inicialmente, seja $j : 1 \leq j \leq m$ uma posição do padrão. Um inteiro $r \geq 1$ é denominado um *período local* de p na posição j se

$$p_i = p_{i+r} \qquad \forall i : (j - r + 1 \leq i \leq j) \text{ e } (i \geq 1) \text{ e } (i + r \leq m).$$

Seja $r(p, j)$ o menor período local de p na posição j e seja q o menor período (global) de p. Assim, a escolha de l se baseia nos seguintes resultados:

Teorema A.1 (Fatoração Crítica) Dada uma palavra p existe no mínimo uma posição l com $1 \leq l < q$ tal que $r(p, l) = q$. □

Uma posição l tal que $r(p, l) = q$ é denominada uma *posição crítica* de p. Por exemplo, a palavra $p = abaabaa$ tem período $q = 3$ e tem as seguintes posições críticas: 2, 4 e 5.

Lema A.2 Seja l uma posição crítica de p e seja r um período local de p na posição l. Se $r \leq \max\{l, |p| - l\}$ então r é um múltiplo de q. □

A partir destes resultados pode-se mostrar (vide [CP91]) que a utilização do algoritmo descrito acima obtém todas as ocorrências do padrão no texto. Além disso, este algoritmo efetua, no pior caso, no máximo $2n + 5m$ comparações. Desta forma, este algoritmo também é linear, pois ele efetua $O(m + n)$ comparações. Mas, ao contrário dos algoritmos KMP e BM, ele requer um espaço de memória que é constante (a menos do armazenamento do padrão).

Finalmente, vale comentar que o algoritmo de Crochemore e Perrin é semelhante ao algoritmo proposto por Galil e Seiferas [GS83] que também efetua o reconhecimento do padrão no texto em tempo linear utilizando espaço constante. A principal diferença entre estes dois algoritmos é que o algoritmo de Galil e Seiferas se baseia numa versão mais fraca do teorema da fatoração crítica e além disso, aquele algoritmo é conceitualmente mais complexo do que o algoritmo descrito nesta seção.

A.4 O Algoritmo de Colussi

O algoritmo que descreveremos a seguir também é uma fusão dos algoritmos KMP e BM. Este algoritmo foi apresentado por Colussi em [Col91], sendo que o autor utiliza técnicas de prova formal de correção de programas para refinar o algoritmo Ingênuo até obter o algoritmo em questão.

Basicamente, este algoritmo consiste em inicialmente determinar os valores de f para o padrão p (construir $\mathcal{A}_m$) e depois o conjunto $\{0,\ldots,m-1\}$ (estados de $\mathcal{A}_m$ cujo caractere desejável é um caractere do padrão) é particionado em

$$\begin{aligned} \mathcal{H} &:= \{j \mid 0 \leq j < m-1 \text{ e } f(j) \geq 0\} \text{ e} \\ \overline{\mathcal{H}} &:= \{j \mid 0 \leq j < m-1 \text{ e } f(j) = -1\}. \end{aligned}$$

Sejam $N_{\mathcal{H}}$ e $N_{\overline{\mathcal{H}}}$ o número de elementos em $\mathcal{H}$ e $\overline{\mathcal{H}}$, respectivamente e sejam h e $\overline{h}$ enumerações de $\mathcal{H}$ e de $\overline{\mathcal{H}}$ respectivamente, tais que $h(i)$ denota o i-ésimo elemento de $\mathcal{H}$, ou seja, h ordena os elementos de $\mathcal{H}$ em ordem crescente, e $\overline{h}(i)$ denota o $(N_{\overline{\mathcal{H}}}+1-i)$-ésimo elemento de $\overline{\mathcal{H}}$, ou seja, $\overline{h}$ ordena os elementos de $\overline{\mathcal{H}}$ em ordem decrescente.

Agora, supondo que $p_1 \ldots p_m$ está alinhado com $t_{k+1} \ldots t_{k+m}$ então, para $i : 1 \leq i \leq N_{\mathcal{H}}$, comparamos $p_{h(i)+1}$ com $t_{k+h(i)+1}$ (note que, pela definição de h, estas comparações são efetuadas da esquerda para a direita). Se alguma destas comparações resulta numa diferença então, pelo algoritmo KMP, o padrão pode ser deslocado de $h(i) - f(h(i))$ posições e as comparações podem prosseguir a partir da posição $h(i')$ no padrão, sendo que $h(i')$ deve ser o menor elemento de $\mathcal{H}$ que é maior do que $f(h(i))$ (depois do deslocamento, $f(h(i))$ é a posição do padrão que estará alinhada com a posição do texto onde ocorreu a diferença).

Por outro lado, se todas as comparações resultam em igualdade então, para $i : 1 \leq i \leq N_{\overline{\mathcal{H}}}$, comparamos $p_{\overline{h}(i)+1}$ com $t_{k+\overline{h}(i)+1}$ (note que, pela definição de $\overline{h}$, estas comparações são efetuadas da direita para esquerda). Se todas estas comparações resultam em igualdade então há uma ocorrência do padrão na posição $k+1$ do texto e, pelo algoritmo KMP, o padrão pode ser deslocado de $f(m)$ posições. Depois deste deslocamento voltamos ao primeiro estágio, sendo que as comparações prosseguem a partir da posição $h(i')$ do padrão, sendo que $h(i')$ deve ser o menor elemento de $\mathcal{H}$ que é maior do que $f(m)$.

Finalmente, se para algum $i : 1 \leq i \leq N_{\overline{\mathcal{H}}}$, $p_{\overline{h}(i)+1} \neq t_{k+\overline{h}(i)+1}$ então, conforme demonstrado em [Col91], o padrão pode ser deslocado de $r(\overline{h}(i))$ posições, onde para todo $j : 1 \leq j \leq m$, $r(j)$ é definido como o menor período do padrão que é maior do j. Também neste caso, voltamos ao primeiro estágio, sendo que as comparações prosseguem a partir da posição $h(i')$ do padrão, sendo que $h(i')$ deve ser o menor elemento de $\mathcal{H}$ que é maior do que $m - r(\overline{h}(i))$.

Uma implementação detalhada deste algoritmo, assim como as demonstrações dos vários resultados utilizados, podem ser obtidas em [Col91]. Além disso, neste mesmo artigo, o autor demonstra que este algoritmo, no pior caso, efetua no máximo $\frac{3}{2}n + \frac{1}{2}(m-1)$ comparações. Vale ressaltar que, por motivos semelhantes àqueles que ocorrem no algoritmo BM ("esquecimento"), a análise deste algoritmo também é consideravelmente complexa.

A.5 O Algoritmo de Karp e Rabin

O algoritmo de Karp e Rabin, que denominaremos algoritmo KR, foi apresentado em [KR87] e consiste em analisar cada subseqüência do texto de comprimento m para verificar se esta subseqüência é igual ao padrão, ou seja, se $t_k \ldots t_{k+m-1}$, com $1 \leq k \leq n-m+1$, é igual a $p_1 \ldots p_m$. Assim, em princípio, este algoritmo procede de maneira similar ao algoritmo Ingênuo descrito na seção 1.2. Porém, ao invés de comparar diretamente os caracteres do padrão com os caracteres de cada subseqüência do texto, o algoritmo KR utiliza uma função "hash" para associar um número inteiro (uma *assinatura*) ao padrão e à subseqüência do texto a ser analisada. Através destas assinaturas o algoritmo determina as possíveis ocorrências do padrão no texto. A seguir descreveremos como a assinatura de uma subseqüência é determinada e como o algoritmo utiliza tais assinaturas.

Inicialmente, seja $c := |\Sigma|$, $\mathcal{S} := \{\alpha \mid \alpha \in \Sigma^* \text{ e } |\alpha| = m\}$ e seja $e : \Sigma \mapsto \{0, \ldots, c-1\}$ uma enumeração do alfabeto Σ. Além disso, seja $H : \mathcal{S} \mapsto \{0, \ldots, c^m - 1\}$ a função definida por:

$$H(\alpha_1 \ldots \alpha_m) := \sum_{i=1}^{m} e(\alpha_i) c^{(m-i)}.$$

Desta forma, a função H estabelece um isomorfismo entre o conjunto $\mathcal{S}$ e o conjunto de números inteiros na base c com m dígitos, ou seja, para

cada seqüência $\alpha = \alpha_1 \ldots \alpha_m$ há um único número de m dígitos na base c dado por $H(\alpha)$ e vice-versa.

O inconveniente da função H, definida acima, é que, dependendo dos valores de c e de m, o valor de $H(\alpha)$ pode ser consideravelmente elevado $(c^m - 1)$. Isto inviabiliza a utilização prática desta função. Para contornar este problema, vamos considerar que a assinatura de $\alpha = \alpha_1 \ldots \alpha_m$ é dada por:

$$h(\alpha) := H(\alpha) \bmod q,$$

onde q é um número primo convenientemente escolhido (posteriormente, estabeleceremos uma forma de escolher o valor de q). Assim, para todo α, $h(\alpha) \in \{0, \ldots, q-1\}$. O problema é que a função h, ao contrário de H, não é injetora, ou seja, existem α e α' tais que $\alpha \neq \alpha'$ e $h(\alpha) = h(\alpha')$. Nestes casos diremos que há uma *colisão* entre as assinaturas de α e α'.

Uma vez estabelecida a função h, o algoritmo KR procede da seguinte forma: digamos que queremos verificar se o padrão ocorre na posição k do texto, onde $1 \leq k \leq n - m + 1$, ou seja, queremos verificar se $t_k \ldots t_{k+m-1}$ é igual ao padrão. Inicialmente, comparamos $h(t_k \ldots t_{k+m-1})$ com $h(p_1 \ldots p_m)$. Se estes valores são diferentes então podemos concluir $t_k \ldots t_{k+m-1}$ é diferente do padrão e portanto, na posição k do texto não há uma ocorrência do padrão. Por outro lado, se $h(t_k \ldots t_{k+m-1}) = h(p_1 \ldots p_m)$ então, como a função h não é injetora, devemos comparar os caracteres da subseqüência do texto com os caracteres do padrão para confirmar se realmente há uma ocorrência do padrão naquela posição texto ou se há uma colisão de assinaturas. Este processo deve ser repetido até que $k > n-m+1$, ou seja, após a análise descrita acima, se $k < n-m+1$ então devemos analisar a próxima subseqüência do texto que é $t_{k+1} \ldots t_{k+m}$.

Observe que para analisar a próxima subseqüência do texto $(t_{k+1} \ldots t_{k+m})$ devemos inicialmente obter a sua assinatura. Como veremos a seguir, esta assinatura pode ser facilmente obtida a partir da assinatura da subseqüência analisada imediatamente antes, isto é, através de $h(t_k \ldots t_{k+m-1})$.

Sejam $\alpha := t_k \ldots t_{k+m-1}$ e $\beta := t_{k+1} \ldots t_{k+m}$. Pela definição de H temos que:

$$H(\beta) = \left[H(\alpha) - e(t_k)c^{(m-1)}\right] c + e(t_{k+m}).$$

Assim,

$$h(\beta) = \left\{\left[H(\alpha) - e(t_k)c^{(m-1)}\right] c + e(t_{k+m})\right\} \bmod q.$$

Como a função mod possui as seguintes propriedades: $\forall\, x, y, q \in \mathcal{Z}$,

(i) se $0 \leq x < q$ então $x \bmod q = x$;

(ii) $(x \bmod q) \bmod q = x \bmod q$;

(iii) $(x + y) \bmod q = ((x \bmod q) + (y \bmod q)) \bmod q$;

(iv) $(xy) \bmod q = ((x \bmod q) \cdot (y \bmod q)) \bmod q$,

então podemos concluir que:

$$h(\beta) = \left\{\left[h(\alpha) - e(t_k)c^{(m-1)}\right] c + e(t_{k+m})\right\} \bmod q. \qquad \text{(A.1)}$$

É claro que em (A.1), ao invés de primeiro efetuar todas as operações e somente depois aplicar a função mod, pelas propriedades (iii) e (iv) acima, podemos primeiro aplicar a função mod a cada parcela para depois efetuar as operações e finalmente aplicar de novo a função mod. Isto permite que o valor de cada parcela seja convenientemente pequeno. Utilizando este fato, na figura A.2, apresentamos uma implementação do algoritmo KR. Observe que estamos assumindo que na nossa pseudo-linguagem, para todo $x \in \mathcal{Z}$, $x \bmod q \geq 0$, por exemplo, $(-3) \bmod 4 = 1$.

É fácil perceber que, no pior caso, para obter todas as ocorrências do padrão no texto, o número de caracteres comparados pelo algoritmo KR é $O(mn)$. O limite superior para o número de comparações é alcançado quando há $O(n)$ ocorrências do padrão no texto e/ou quando ocorrem $O(n)$ colisões. Entretanto, conforme apresentado em [KR87], se o valor de q é determinado respeitando as condições impostas pelos autores então, na prática, a probabilidade de que ocorram $O(n)$ colisões é muito pequena. Além disso, os autores propõem uma pequena modificação no algoritmo de modo que o valor de q é modificado sempre que há uma colisão. Desta forma, a probabilidade de que ocorram $O(n)$ colisões se torna praticamente zero. Assim, eles concluem que, na prática, para obter a primeira ocorrência do padrão no texto, o número de caracteres comparados pelo algoritmo KR é $O(m + n)$.

Finalmente, podemos estabelecer uma maneira para escolher o valor do número primo q. Note que, pela forma como a função h é definida, quanto maior for o valor de q, melhor. Além disso, pelo algoritmo dado na figura A.2, podemos perceber que o valor da maior parcela envolvida no

Algoritmo *KR*;

Entrada : *o padrão p e o texto t.*

Saída : *Lista das posições em t onde o padrão p ocorre.*

```
begin
    {q é um número primo convenientemente escolhido.}
    {e é uma enumeração do alfabeto Σ.}
    {h_p é a assinatura do padrão; h_t é a assinatura da subseqüência do
    texto.}
    {d = c^(m-1) mod q.}
    d := 1;
    for k := 1 to m − 1 do d := (d * c) mod q;
    h_p := 0;
    h_t := 0;
    for k := 1 to m do
    begin
        h_p := (h_p * c + e(p_k)) mod q;
        h_t := (h_t * c + e(t_k)) mod q;
    end;
    k := 1;
    while k ≤ n − m + 1 do
    begin
        if (h_p = h_t) and (p_1 ... p_m = t_k ... t_{k+m-1}) then
            write('O padrão ocorre na posição ',k);
        h_t := ((h_t − e(t_k) * d) mod q * c + e(t_{k+m})) mod q;
        k := k + 1
    end
end;
```

Figura A.2: Uma implementação do algoritmo KR.

cálculo da assinatura h_t é menor do que $qc-1$. Portanto, q deve ser o maior número primo tal que $qc-1$ não cause "overflow". Assim, Sedgewick, em [Sed83], sugere que, para $c = 32$, devemos utilizar $q = 33554393$ e conforme descrito em [Ste92], Pirklbauer ([Pir92]) sugere que, para $c = 256$, devemos utilizar $q = 8355967$.

Apêndice B

Análises de Caso Médio

Neste apêndice vamos analisar o comportamento médio dos algoritmos Ingênuo, KMP e BM estabelecendo o valor esperado para o número de comparações ($\overline{C}_n$) efetuadas por cada um destes algoritmos. A importância deste tipo de análise é que ela nos permite avaliar o comportamento dos algoritmos levando-se em consideração situações que, na prática, ocorrem com maior freqüência.

Conforme vimos nos capítulos anteriores, se considerarmos o pior caso, o algoritmo KMP é o que apresenta o melhor desempenho. Entretanto, neste apêndice veremos que, em média, o algoritmo com melhor desempenho é o algoritmo BM, enquanto que os algoritmos KMP e Ingênuo apresentam desempenhos semelhantes.

Nas análises que descreveremos a seguir vamos supor que o alfabeto tem tamanho c (isto é, $c := |\Sigma|$) e que a probabilidade de que o i-ésimo caractere do alfabeto ocorra no padrão ou no texto é dada por ρ_i. Assim, se escolhermos independentemente dois caracteres do alfabeto então a probabilidade de que estes dois caracteres sejam iguais é:

$$\rho_{eq} := \sum_{i=1}^{c} \rho_i^2.$$

Definição B.1 Uma *seqüência aleatória* de comprimento m é obtida concatenando-se m caracteres de Σ escolhidos independente e uniformemente. Isto é, numa seqüência aleatória, $\rho_i = \frac{1}{c}$, para todo i.

De agora em diante, vamos supor que tanto o texto como o padrão

são seqüências aleatórias com uma distribuição de probabilidade uniforme. Assim, temos que:

Lema B.2 A probabilidade de que um determinado caractere do texto seja igual a um caractere do padrão é $\frac{1}{c}$. □

É importante ressaltar que os resultados que estabeleceremos a seguir também são válidos quando a distribuição de probabilidade não é uniforme. Nestes casos, basta fazer ρ_{eq} igual a $\sum_{i=1}^{c} \rho_i^2$ ao invés de utilizar o lema acima.

Lema B.3 Seja $\overline{C}_m$ o valor esperado para o número de comparações efetuadas para se verificar se o padrão é igual (ou não) a uma determinada subseqüência do texto de comprimento m. Então,

$$\overline{C}_m = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right).$$

Demonstração : Inicialmente, note que a assertiva independe da ordem em que as comparações são efetuadas. Assim, sem perda de generalidade, vamos supor que as comparações são efetuadas da esquerda para a direita. Seja $t_{k+1} \ldots t_{k+m}$, onde $0 \leq k \leq n-m$, uma subseqüência do texto de comprimento m. Então a probabilidade de que o padrão seja igual a $t_{k+1} \ldots t_{k+m}$ é:

$$\begin{aligned} \mathcal{P}\{p_1 \ldots p_m = t_{k+1} \ldots t_{k+m}\} &= \mathcal{P}\{p_1 = t_{k+1} \wedge \ldots \wedge p_m = t_{k+m}\} \\ &= \prod_{i=1}^{m} \mathcal{P}\{t_{k+i} = p_i\} \\ &= \frac{1}{c^m}, \end{aligned} \tag{B.1}$$

onde a última igualdade é devido ao lema B.2.

Para verificar se o padrão é igual a $t_{k+1} \ldots t_{k+m}$ efetuamos pelo menos uma comparação (entre p_1 e t_{k+1}) e para cada igualdade obtida ($p_i = t_{k+i}$) efetuamos a próxima comparação (se houver caracteres a comparar), ou seja, a cada igualdade obtida efetuamos mais uma comparação. Assim,

$$\overline{C}_m = 1 + \sum_{i=1}^{m-1} \mathcal{P}\{p_1 = t_{k+1} \wedge \ldots \wedge p_i = t_{k+i}\}$$

$$\stackrel{(B.1)}{=} 1 + \sum_{i=1}^{m-1} \frac{1}{c^i}$$
$$= \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right).$$

□

Agora, passemos às análises dos algoritmos Ingênuo, KMP e BM. Estas análises serão apresentadas respectivamente nas seções B.1, B.2 e B.3. Além disso, na seção B.4 apresentaremos os resultados de algumas análises empíricas que confirmam os resultados obtidos teoricamente.

B.1 Caso Médio do Algoritmo Ingênuo

Conforme vimos na seção 1.2, o algoritmo Ingênuo consiste em verificar o padrão com todas as subseqüências do texto de comprimento m, sendo que as verificações se iniciam na extremidade esquerda do texto e a próxima verificação é realizada deslocando-se o padrão de uma posição para a direita em relação ao texto. Assim, lembrando que o texto e o padrão são seqüências aleatórias temos que:

Teorema B.4 O valor esperado para o número de comparações efetuadas pelo algoritmo Ingênuo é:

$$\overline{C}_n = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1).$$

Demonstração : O algoritmo Ingênuo verifica todas as subseqüências $t_{k+1} \ldots t_{k+m}$ para $k : 0 \leq k \leq n - m$. Então serão verificadas $(n - m + 1)$ subseqüências de comprimento m o que implica que

$$\overline{C}_n = \overline{C}_m \cdot (n - m + 1).$$

Assim, pelo lema B.3, temos que

$$\overline{C}_n = \frac{c}{c-1}\left(1 - \frac{1}{c^m}\right)(n - m + 1).$$

□

É importante notar que o resultado acima mostra que o comportamento médio do algoritmo Ingênuo é bem melhor do que o seu comportamento no pior caso, que é $m(n-m+1)$, por exemplo, quando o texto é a^n e o padrão é $a^{m-1}b$.

A análise apresentada nesta seção foi proposta por Baeza-Yates em [BY89b]. Uma outra abordagem para a análise do comportamento médio do algoritmo Ingênuo foi proposta por Barth [Bar84] e consiste em modelar o algoritmo Ingênuo através de um autômato que depois é transformado numa cadeia de Markov (na próxima seção descreveremos este tipo de cadeia). Daí, aplicando-se alguns resultados estabelecidos para estas cadeias, obtém-se o valor esperado para o número de comparações efetuadas por este algoritmo.

B.2 Caso Médio do Algoritmo KMP

A análise que descreveremos a seguir foi apresentada por Baeza-Yates em [BY89b] e se utiliza de conceitos sobre cadeias de Markov. Portanto, inicialmente descreveremos brevemente este tipo de cadeia, sendo que um tratamento mais detalhado sobre tais cadeias pode ser obtido em vários livros, como por exemplo [IM76].

Uma *cadeia de Markov* finita é um processo estocástico que em um determinado instante está em, digamos, um dos estados $s_1, s_2, \ldots, s_r$. A probabilidade de que o processo se mova de um estado s_j para um estado s_k depende apenas dos estados s_j e s_k, ou seja, independe dos estados alcançados antes de s_j. Para cada par de estados (s_j, s_k), seja ρ_{jk} a probabilidade de que a transição de s_j para s_k ocorra. Desta forma, uma cadeia de Markov com r estados pode ser descrita através de uma matriz M, denominada *matriz de transição*, com $r \times r$ posições, sendo que

$$M(j,k) := \rho_{jk}.$$

É claro que

$$\begin{array}{cccc} & \forall\, j,k & & M(j,k) \geq 0 \\ e & \forall\, j & \sum_{k=1}^{r} & M(j,k) = 1. \end{array}$$

Um estado s_j é *absorvente* se $\rho_{jj} = 1$, o que implica que $\rho_{jk} = 0$, para todo $k \neq j$.

Agora, considerando o tempo como um elemento discreto, dizemos que numa cadeia de Markov a probabilidade de que a cadeia atinja o estado s_j no instante i depende apenas do estado no instante anterior $(i-1)$. Seja $\rho_j^{(i)}$ a probabilidade de que a cadeia atinja o estado s_j no instante i e seja $\rho^{(i)} = (\rho_1^{(i)}, \ldots, \rho_r^{(i)})$ o vetor de probabilidades para o instante i. Assim, dado que $\rho^{(0)}$ é o vetor das probabilidades dos estados iniciais, isto é, a probabilidade de que s_j seja o estado inicial é $\rho_j^{(0)}$, então para $i = 1, 2, \ldots$, temos que:

$$\rho^{(i)} = \rho^{(0)} \cdot M^i.$$

Por exemplo, se desejamos saber qual é a probabilidade de alcançar os estados $s_1, \ldots, s_r$ no instante $i = 5$ começando do estado s_1 então tomamos $\rho^{(0)} = (1, 0, 0, \ldots, 0)$ e calculamos $\rho^{(5)} = (1, 0, \ldots, 0) \cdot M^5$.

Considere uma cadeia de Markov que não possui um estado absorvente. Então, nesta cadeia, todos os estados possuem pelo menos uma transição para algum outro estado e, conforme demonstrado em [IM76, cap. III], para i grande, $\rho^{(i)}$ converge para um valor que é denominado *vetor de probabilidades estacionário* e que será denotado por π. Assim,

$$\lim_{i \to \infty} \rho^{(i)} = \pi,$$

e π é a solução do seguinte sistema de equações lineares:

$$\pi \cdot M = \pi \qquad \text{e} \qquad \sum_{j=1}^{r} \pi_j = 1. \tag{B.2}$$

Agora podemos iniciar a análise do algoritmo KMP. A tendência natural seria transformar a representação abreviada do autômato reconhecedor de padrão ($\mathcal{A}_m$) numa cadeia de Markov atribuindo às transições desejáveis a probabilidade $\frac{1}{c}$ e às transições de falha, $1 - \frac{1}{c}$. Entretanto há um problema com esta abordagem, pois desta forma diferentes padrões (que possuem $\mathcal{A}_m$ diferentes) geram cadeias de Markov distintas, o que dificulta uma análise genérica. Para contornar este problema Barth [Bar84] propôs um modelo aproximado que, para todos os padrões de comprimento m, gera uma cadeia de Markov única. Esta aproximação é obtida supondo que, para todo padrão, $f(j) = 1$, $\forall j : 2 \leq j \leq m$. Porém, conforme mencionado por Baeza-Yates [BY89b] e por Regnier [Reg89] este modelo é inválido, pois por definição, numa cadeia de Markov uma transição não pode depender dos

estados anteriormente alcançados e, na realidade, no algoritmo KMP, há uma "memória" implícita que se manifesta através da transição de falha. Vale ressaltar que Barth utiliza a mesma estratégia para analisar o algoritmo Ingênuo, mas neste caso, esta estratégia se mostra adequada pois o algoritmo Ingênuo não é dotado de "memória".

A seguir vamos descrever a análise apresentada por Baeza-Yates em [BY89b] cuja idéia básica é modelar $\mathcal{A}_m$ através de uma cadeia de Markov reduzida. Porém, é importante salientar que identificamos alguns problemas nesta abordagem e infelizmente não conseguimos contorná-los. Assim, apresentaremos a descrição originalmente proposta e durante esta descrição indicaremos os problemas detectados.

Intuitivamente, a idéia é inicialmente estender $\mathcal{A}_m$ incluindo estados fictícios e a partir daí, construir uma cadeia de Markov reduzida composta apenas de três "estados" que denominaremos: S_0, S_1 e S_2. O estado S_0 representa o estado 0 de $\mathcal{A}_m$ e o estado S_1 corresponde aos estados $1, \ldots, m$ de $\mathcal{A}_m$. Além disso, o estado S_2 representa os estados fictícios acrescentados em $\mathcal{A}_m$, sendo que para cada estado $j \geq 1$ tal que $f(j) \geq 0$ acrescentamos um estado fictício. Este estado fictício será alcançado toda vez que houver uma falha no estado j correspondente. Na verdade, os estados fictícios são cópias dos estados maiores do que -1 que são destinos de transições de falha e sempre que houver uma falha num estado $j \geq 1$ (estados que compõem S_1) com $f(j) \geq 0$ então haverá uma transição para um estado fictício (estados que compõem S_2). Note que ao atingirmos um estado fictício temos a informação adicional de que o caractere corrente do texto é diferente de p_{j+1} (j é o estado onde houve a falha).

Agora, vejamos como são estabelecidas as transições entre os estados S_0, S_1 e S_2 que formam a cadeia de Markov. Considere a figura B.1 (aqui temos o principal problema detectado, pois, em nossa opinião, faltam estabelecer duas transições nesta cadeia conforme mostramos na figura B.2).

A partir do estado S_0 temos duas transições. Uma para o próprio S_0, sendo que esta transição corresponde às transições (em $\mathcal{A}_m$) de 0 para -1 e deste de volta para 0. Como a transição de 0 para -1 é efetuada quando o caractere corrente do texto é diferente de p_1 então a probabilidade da transição de S_0 para S_0 é $1 - \frac{1}{c}$.

A outra transição a partir de S_0 é para S_1. Esta transição corresponde à transição do estado 0 para o estado 1 em $\mathcal{A}_m$, que é efetuada quando o caractere corrente do texto é igual a p_1. Portanto, a transição de S_0 para

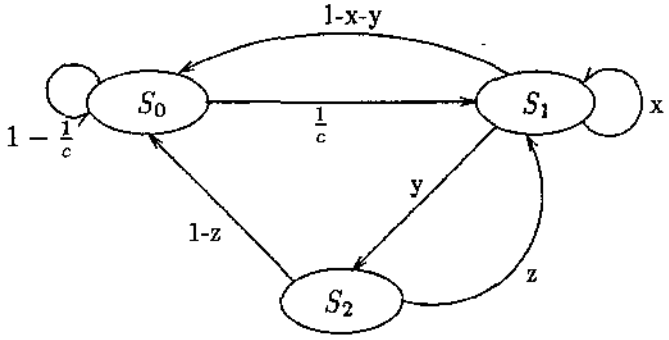


Figura B.1: Cadeia de Markov (incompleta) que modela $\mathcal{A}_m$.

S_1 é efetuada com uma probabilidade igual a $\frac{1}{c}$.

A partir do estado S_1 temos três transições. A transição de S_1 para S_2 ocorre sempre que houver uma falha em algum estado $j \geq 1$ com $f(j) \geq 0$. Seja y a probabilidade de que esta transição ocorra. Então, temos que

$$0 \leq y \leq 1 - \frac{1}{c},$$

pois a probabilidade de que ocorra uma falha em algum estado é $1 - \frac{1}{c}$ e a probabilidade de que a transição de falha deste estado seja maior do que -1 é um valor entre 0 e 1.

A transição de S_1 para S_1 representa a transição em $\mathcal{A}_m$ que é efetuada quando o estado m é alcançado (isto é, uma ocorrência do padrão é obtida no texto) e $f(m) > 0$. (Neste ponto temos a primeira transição "esquecida" que seria a correspondente das transições em $\mathcal{A}_m$ que vão do estado j para o estado $j+1$, com $1 \leq j \leq m-1$, isto é, deveria existir uma "outra" transição de S_1 para S_1 de modo que as transições em $\mathcal{A}_m$, que são efetuadas quando o caractere corrente do texto é igual ao caractere desejável do estado j, fossem consideradas.) Seja x a probabilidade de que a transição (original) de S_1 para S_1 ocorra. Então, temos que

$$0 \leq x \leq \frac{1}{c},$$

pois o estado m é atingido apenas a partir do estado $m-1$ e a probabilidade de que esta transição (em $\mathcal{A}_m$) ocorra é $\frac{1}{c}$. Assim, se $f(m) \leq 0$ então $x = 0$. Por outro lado, se $f(m) > 0$ então $x = 1 \cdot \frac{1}{c} = \frac{1}{c}$.

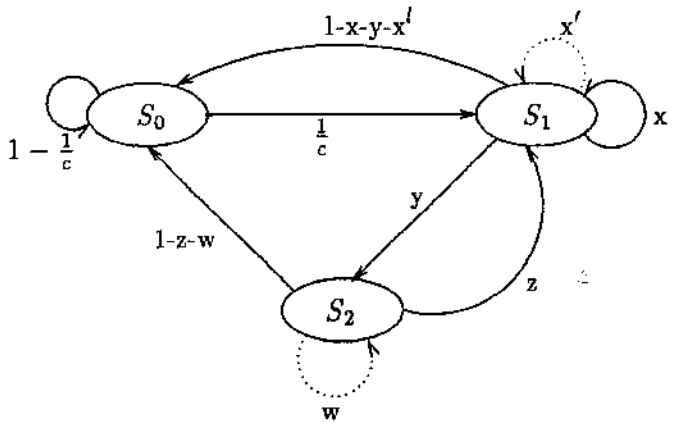


Figura B.2: Cadeia de Markov (completa) que modela $\mathcal{A}_m$. As transições indicadas em linhas pontilhadas representam as transições "esquecidas" na cadeia original.

A terceira transição a partir de S_1 vai para S_0 e corresponde às transições de falha que têm valor menor do que 0. É fácil ver que a probabilidade de que esta transição ocorra é $1-x-y$ (se considerarmos a transição "esquecida", cuja probabilidade é digamos x', então teremos $1-x-y-x'$).

Finalmente, a partir do estado S_2 temos duas transições. Uma que vai para o estado S_1 e que é efetuada quando, após alguma(s) transição(ções) de falha (consecutivas), conseguimos igualar o caractere corrente do texto com algum caractere do padrão. Seja z a probabilidade de que esta transição ocorra (a seguir estabeleceremos os limites para z). A outra transição a partir de S_2 vai para S_0. Esta transição é realizada quando, após algumas transições de falha atingimos o estado -1, ou seja, não conseguimos igualar o caractere corrente do texto com nenhum dos caracteres do padrão que foram verificados. A probabilidade de que esta transição ocorra é $1-z$. (Neste ponto temos a segunda transição "esquecida" que seria a representante das transições de falha em $\mathcal{A}_m$ efetuadas a partir de um estado que compõe S_2, isto é, deveria existir uma transição de S_2 para S_2 de modo que as transições de falha realizadas consecutivamente fossem consideradas. Se a transição de S_2 para S_2 fosse incorporada à cadeia de Markov com uma probabilidade igual a w então a probabilidade da transição de S_2 para S_0 seria $1-w-z$. Na figura B.2 apresentamos a cadeia de Markov que, em nossa opinião, é a

cadeia completa.)

Agora, para concluir a descrição da cadeia de Markov resta apenas estabelecer os limites para a variação de z. Inicialmente, como z é a probabilidade de que a transição de S_2 para S_1 seja efetuada então

$$0 \leq z \leq 1, \tag{B.3}$$

Mas, observe que esta transição somente é efetuada quando conseguimos igualar o caractere do texto com algum caractere do padrão. Além disso, para que o estado S_2 seja atingido é necessário que tenha ocorrido uma falha em $\mathcal{A}_m$ em algum estado $j \geq 1$ com $f(j) \geq 0$. Portanto, é necessário que o caractere corrente do texto, digamos t_k, seja diferente de p_{j+1}. Agora, se estivermos em S_2 então a probabilidade de que o caractere t_k seja igual a um caractere do padrão é no mínimo $\frac{1}{c-1}$ (não precisamos comparar t_k com p_{j+1} novamente). Por outro lado, conforme vimos na seção 2.3 (teorema 2.17), em $\mathcal{A}_m$ um mesmo caractere de um texto pode ser comparado no máximo $\log_\phi(m+1)$ vezes, onde $\phi = \frac{1+\sqrt{5}}{2}$. Seja

$$\Lambda := \left\lfloor \log_\phi(m+1) \right\rfloor - 1,$$

isto é, Λ corresponde ao número máximo de transições de falha consecutivas que podem ocorrer em $\mathcal{A}_m$. Assim, antes de obtermos uma igualdade entre t_k e um caractere do padrão, o máximo que podemos vir a saber sobre t_k é que este caractere é diferente de Λ caracteres. Daí, se estivermos no estado S_2 então a probabilidade de que t_k seja igual a um caractere do padrão não é maior do que $\frac{1}{c-\Lambda}$, o que implica que

$$\frac{1}{c-1} \leq z \leq \frac{1}{c-\Lambda}. \tag{B.4}$$

Agora, combinando os resultados de (B.3) e (B.4) temos que:

- se $c \leq \Lambda + 1$ então,
$$\frac{1}{c-1} \leq z \leq 1.$$
- se $c > \Lambda + 1$ então,
$$\frac{1}{c-1} \leq z \leq \frac{1}{c-\Lambda}.$$

Uma vez estabelecida a cadeia de Markov que modela o algoritmo KMP, vamos agora obter o vetor de probabilidades estacionário para esta cadeia. Inicialmente, note que a cadeia de Markov em questão não possui nenhum estado absorvente. Portanto, podemos obter este vetor através de (B.2).

A cadeia dada na figura B.1 é equivalente à seguinte matriz:

$$M = \begin{vmatrix} 1-\frac{1}{c} & \frac{1}{c} & 0 \\ 1-x-y & x & y \\ 1-z & z & 0 \end{vmatrix},$$

onde a primeira linha se refere ao estado S_0, a segunda ao estado S_1 e a terceira a S_2.

Seja $\pi = (\pi_0, \pi_1, \pi_2)$ o vetor de probabilidades estacionário. Então aplicando (B.2) obtemos o seguinte sistema de equações lineares:

$$\begin{cases} \left(1-\frac{1}{c}\right)\pi_0 + (1-x-y)\pi_1 + (1-z)\pi_2 &= \pi_0 \\ \frac{1}{c}\pi_0 + x\pi_1 + z\pi_2 &= \pi_1 \\ y\pi_1 &= \pi_2 \\ \pi_0 + \pi_1 + \pi_2 &= 1 \end{cases}$$

cuja solução é:

$$\begin{aligned} \pi_0 &= \frac{c(1-yz-x)}{c+1+y(1-cz)-cx}, \\ \pi_1 &= \frac{1}{c+1+y(1-cz)-cx}, \\ \pi_2 &= \frac{y}{c+1+y(1-cz)-cx}. \end{aligned}$$

A partir destes resultados, Baeza-Yates afirma que:

Teorema B.5 O valor esperado para o número de comparações efetuadas pelo algoritmo KMP é:

- se $c \leq \log_\phi(m+1)$ então

$$\frac{\overline{C}_n}{n} \leq 2 - \frac{1}{c} + O\left(\frac{1}{n}\right)$$

- se $c > \log_\phi(m+1)$ então

$$\frac{\overline{C}_n}{n} \leq 1 + \frac{(c-\Lambda)(c-1)}{c^2(c-\Lambda) - c(c-1)} + O\left(\frac{1}{n}\right),$$

onde $\Lambda = \left\lfloor \log_\phi(m+1) \right\rfloor - 1$ com $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

Demonstração : Baeza-Yates afirma que

$$\frac{\overline{C}_n}{n} = \frac{1}{(1-\pi_2)} + O\left(\frac{1}{n}\right). \tag{B.5}$$

(Os problemas indicados durante a descrição da cadeia de Markov se refletem basicamente neste ponto, pois a equação acima seria verdadeira, sem o termo $O\left(\frac{1}{n}\right)$, se cada comparação entre o padrão e o texto implicasse uma transição na cadeia e, é fácil ver que, na cadeia dada na figura B.1, esta propriedade não se verifica.)

Visto que

$$\pi_2 = \frac{y}{c+1+y(1-cz)-cx},$$

então, pelos limites estabelecidos para x, y e z, temos que o valor máximo para π_2 é alcançado quando $x = \frac{1}{c}$, $y = 1 - \frac{1}{c}$ e :

$$\begin{array}{ll} z = 1 & \text{se } c \leq \Lambda + 1 \quad \text{ou} \\ z = \frac{1}{c-\Lambda} & \text{se } c > \Lambda + 1. \end{array}$$

Substituindo estes valores na expressão acima (note que, $\Lambda + 1 = \left\lfloor \log_\phi(m+1) \right\rfloor$) temos que: se $c \leq \log_\phi(m+1)$ então

$$\pi_2 \leq \frac{c-1}{2c-1},$$

o que implica que

$$\frac{1}{1-\pi_2} \leq \frac{2c-1}{c} = 2 - \frac{1}{c}.$$

Logo,

$$\frac{\overline{C}_n}{n} \leq 2 - \frac{1}{c} + O\left(\frac{1}{n}\right)$$

o que demonstra a primeira parte do teorema.

Por outro lado, se $c > \log_\phi(m+1)$ então

$$\pi_2 \leq \frac{(c-\Lambda)(c-1)}{c^2(c-\Lambda)-\Lambda(c-1)},$$

o que implica que

$$\frac{1}{1-\pi_2} \leq \frac{c^2(c-\Lambda)-\Lambda(c-1)}{c^2(c-\Lambda)-c(c-1)}.$$

Logo,

$$\frac{\overline{C}_n}{n} \leq 1 + \frac{(c-\Lambda)(c-1)}{c^2(c-\Lambda)-c(c-1)} + O\left(\frac{1}{n}\right),$$

o que conclui a demonstração do teorema. □

Na figura B.3, apresentamos um gráfico que mostra a relação entre o limite teórico estabelecido pelo teorema acima e os valores para $\frac{\overline{C}_n}{n}$ obtidos empiricamente.

Conforme vimos, a abordagem proposta por Baeza-Yates apresenta alguns problemas que fundamentalmente se referem ao fato de que, na cadeia de Markov dada na figura B.1, algumas comparações entre o padrão e o texto não implicam em transições nesta cadeia e portanto, a equação B.5 não se aplica.

Na figura B.2 apresentamos a cadeia de Markov completa, onde incluímos as duas transições "esquecidas". Nesta cadeia temos que cada comparação entre o padrão e o texto corresponde a uma transição e portanto,

$$\frac{\overline{C}_n}{n} = \frac{1}{(1-\pi_2)}.$$

Agora, considerando a cadeia completa, se efetuarmos uma análise semelhante à descrita acima nos deparamos com o seguinte problema: para obter um limite superior para π_2 é necessário estabelecer um limite superior para w, mas infelizmente não obtivemos nenhum limite melhor do que 1 (este limite não nos leva a nenhum resultado considerável).

Para encerrar esta seção, vale salientar que Regnier em [Reg89] realiza uma análise de caso médio do algoritmo KMP utilizando uma outra estratégia que se baseia em combinatória de palavras e em funções geradoras.

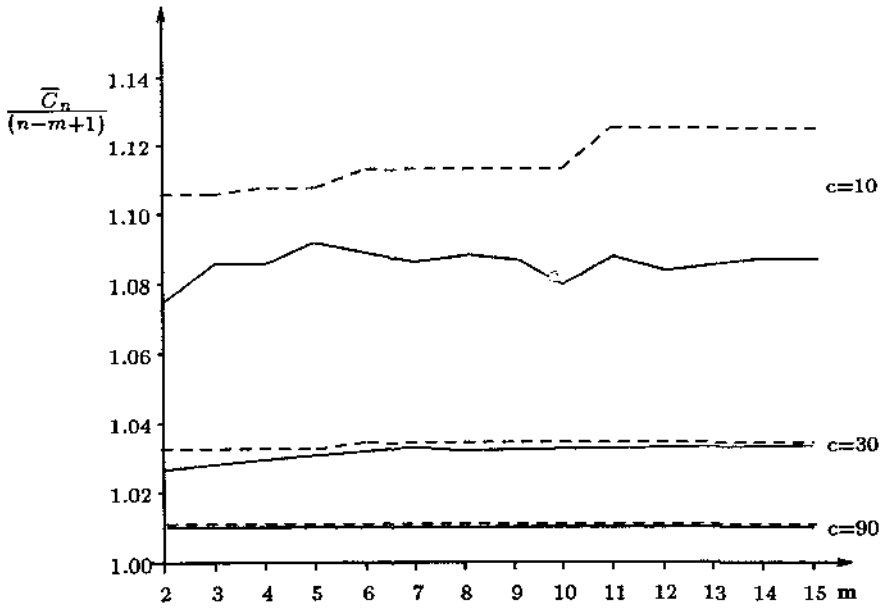


Figura B.3: Comparação entre o limite teórico e os resultados da análise empírica do algoritmo KMP. A linha contínua representa os resultados obtidos empiricamente enquanto a linha tracejada representa o limite teórico.

O resultado obtido através desta análise é que

$$\frac{\overline{C}_n}{n} \approx 1 + \frac{1}{c}.$$

O inconveniente desta análise é que, além de bastante longa, ela é consideravelmente complexa.

B.3 Caso Médio do Algoritmo BM

Como sabemos, o algoritmo BM determina o deslocamento a ser efetuado no padrão através das funções s_1, s_2 e s_3, sendo que s_2 e s_3 contribuem efetivamente para este deslocamento apenas nos casos em que existem repetições no padrão. Visto que, na prática, tais repetições são relativamente raras então, para avaliar o comportamento médio do algoritmo BM, podemos ignorar a contribuição destas funções. Assim, poderíamos analisar

uma versão simplificada do algoritmo BM onde o deslocamento seria determinado através do máximo entre 1 e $s_1(t_k)$ (t_k é o caractere do texto onde ocorreu uma falha). Porém, conforme vimos na seção 3.4.3, o algoritmo BM_H é uma variação do algoritmo BM que se baseia nestas idéias, mas as utiliza de modo mais simples, o que facilita a análise. Vale lembrar que, no algoritmo BM_H, o deslocamento é sempre igual a $m - s_1(t_l)$, sendo que t_l é o caractere do texto alinhado com p_m numa rodada de verificações.

A seguir vamos apresentar a análise de caso médio do algoritmo BM_H que foi proposta por Baeza-Yates em [BY89b], sendo que a estratégia adotada é estabelecer um limite inferior para $\overline{C}_n$, que é o número esperado de comparações efetuadas, e através de análises empíricas, mostrar que este limite é justo.

Lema B.6 No algoritmo BM_H, o valor esperado para o comprimento de cada deslocamento do padrão é

$$\overline{d} = c\left[1 - \left(1 - \frac{1}{c}\right)^m\right].$$

Demonstração : No algoritmo BM_H, o deslocamento máximo efetuado é de m posições e este deslocamento é obtido quando o caractere t_l não ocorre nas primeiras $m-1$ posições do padrão, ou seja, quando $t_l \neq p_i$, para $1 \leq i \leq m-1$. Portanto, a probabilidade de que se efetue um deslocamento de m posições é

$$\left(1 - \frac{1}{c}\right)^{m-1}.$$

Agora, se t_l ocorre em alguma das primeiras $m-1$ posições do padrão e p_j é o caractere mais à direita em $p_1 \ldots p_{m-1}$ tal que $p_j = t_l$ então o deslocamento será de $m-j$ posições. Portanto, para que seja efetuado um deslocamento de $m-j$ posições é necessário que $p_j = t_l$ e $p_i \neq t_l$ para todo $i : j+1 \leq i \leq m-1$, o que implica que a probabilidade de que seja efetuado um deslocamento igual a $m-j$ é

$$\frac{1}{c} \cdot \left(1 - \frac{1}{c}\right)^{m-j-1}.$$

Assim,

$$\overline{d} = m \cdot \left(1 - \frac{1}{c}\right)^{m-1} + \frac{1}{c} \cdot \sum_{j=1}^{m-1} (m-j)\left(1 - \frac{1}{c}\right)^{m-j-1}.$$

Visto que

$$\sum_{j=1}^{m-1}(m-j)\left(1-\frac{1}{c}\right)^{m-j-1} = c^2\left[1-\left(1-\frac{1}{c}\right)^m - \frac{m}{c}\left(1-\frac{1}{c}\right)^{m-1}\right]$$

então

$$\begin{aligned}\overline{d} &= m\cdot\left(1-\frac{1}{c}\right)^{m-1} + \frac{1}{c}\cdot c^2\left[1-\left(1-\frac{1}{c}\right)^m - \frac{m}{c}\left(1-\frac{1}{c}\right)^{m-1}\right] \\ &= c\cdot\left[1-\left(1-\frac{1}{c}\right)^m\right].\end{aligned}$$

□

Teorema B.7 No algoritmo $\mathrm{BM_H}$, o número esperado de comparações efetuadas é tal que

$$\overline{C}_n \geq \frac{1-\frac{1}{c^m}}{(c-1)\cdot\left[1-\left(1-\frac{1}{c}\right)^m\right]}\cdot(n-m+1).$$

Demonstração : Conforme demonstrado em [Cla82] (desigualdade de Kantorovich), temos que

$$\overline{X}\cdot\overline{X^{-1}} \geq 1,$$

onde $\overline{X}$ e $\overline{X^{-1}}$ são respectivamente os valores esperados das variáveis aleatórias X e X^{-1}.

Assim, denotando por $\overline{v}$ o número esperado de vezes que o padrão é deslocado, temos que

$$\overline{v} = (n-m+1)\cdot\overline{d^{-1}} \geq (n-m+1)\cdot\frac{1}{c\left[1-\left(1-\frac{1}{c}\right)^m\right]}.$$

Agora, cada deslocamento é efetuado após uma rodada de verificações e em cada uma destas rodadas, em média, são efetuadas $\overline{C}_m$ comparações. Visto que, pelo lema B.3,

$$\overline{C}_m = \frac{c}{c-1}\left(1-\frac{1}{c^m}\right)$$

então

$$\begin{aligned}\overline{C}_n &\geq \frac{c}{c-1}\left(1-\frac{1}{c^m}\right)\frac{n-m+1}{c\left[1-\left(1-\frac{1}{c}\right)^m\right]} \\ &= \frac{1-\frac{1}{c^m}}{(c-1)\cdot\left[1-\left(1-\frac{1}{c}\right)^m\right]}\cdot(n-m+1).\end{aligned}$$

□

Conforme salientamos anteriormente, o teorema acima estabelece um limite inferior para o valor esperado para o número de comparações efetuadas pelo algoritmo $\mathrm{BM_H}$. Através de uma análise empírica, cujos resultados são apresentados resumidamente na figura B.4, Baeza-Yates mostra que este limite é justo para $c \geq 10$. Nesta análise o autor avaliou o número médio de comparações efetuadas pelo algoritmo $\mathrm{BM_H}$ em função do tamanho do alfabeto. Maiores detalhes sobre esta análise empírica podem ser obtidos em [BY89a] e [BY89b].

Uma outra análise de caso médio do algoritmo $\mathrm{BM_H}$ é apresentada por Baeza-Yates e Regnier em [BYR92]. Nesta análise, os autores utilizam processos estocásticos e funções geradoras para realizar uma análise bem mais rigorosa e precisa do que a apresentada acima. Desta forma, eles demonstram que:

Teorema B.8 Seja $\overline{C}_{n,m}$ o número esperado de comparações efetuadas pelo algoritmo $\mathrm{BM_H}$ para obter todas as ocorrências de um padrão de comprimento $m+1$ num texto de comprimento n. Então

$$\frac{\overline{C}_{n,m}}{n} = \mathcal{H}(c,m)\left(1+\frac{1}{c}+\frac{2}{c^2}+O\left(\frac{1}{c^3}\right)\right),$$

onde

$$\mathcal{H}(c,m) = c\sum_{j=1}^{c}\binom{c}{j}\sum_{\substack{k_i \geq 1 \\ k_1+\ldots+k_{j-1}<m}}\frac{\prod_{i=1}^{j-1}\left(\frac{i}{j}\right)^{k_i}\left(\frac{j}{c}\right)^m}{j+\sum_{i=1}^{c-1}(j-i)k_i+(c-j)(m+1)}$$

□

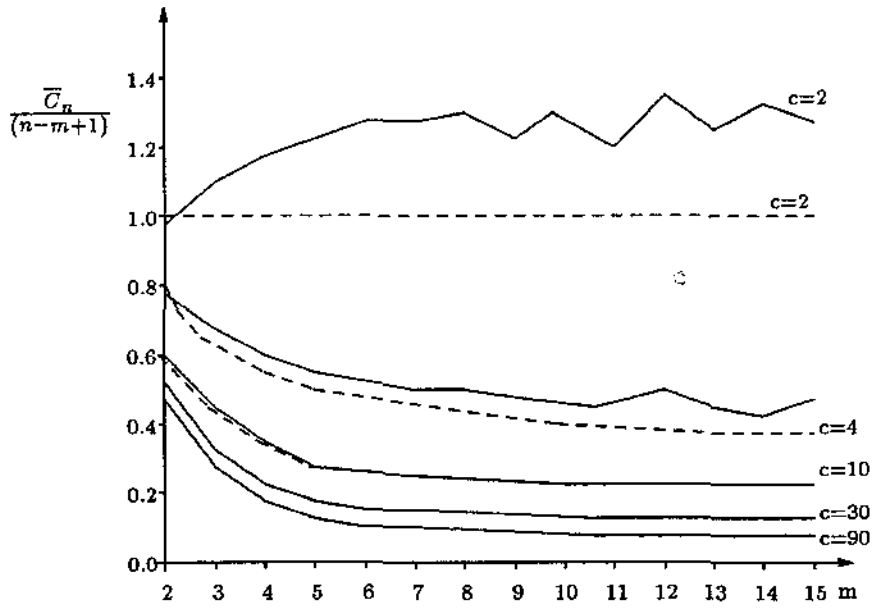


Figura B.4: Comparação entre o limite inferior teórico e os resultados da análise empírica do algoritmo BM_H. A linha contínua representa os resultados obtidos empiricamente enquanto a linha tracejada representa o limite inferior teórico.

Além disso, os autores demonstram que

$$\frac{1}{c} \leq \lim_{m\to\infty} \mathcal{H}(c,m) \leq \frac{2}{c+1}.$$

Finalmente, vale acrescentar que Schaback em [Sch88] também apresenta uma análise de caso médio do algoritmo BM. Neste caso, o autor analisa uma versão simplificada do algoritmo que é diferente do algoritmo BM_H. Nesta análise o autor demonstra que:

Teorema B.9 Se $2 \leq m \leq \frac{1}{q}$ então cada comparação implica num deslocamento esperado $\overline{d}$, sendo que

$$\overline{d} \geq (1-q)(1-q^2)\left(m - \frac{m^2 q}{2}\right) \geq (1-q)(1-q^2)\frac{m}{2},$$

onde q é a probabilidade de uma comparação resultar em igualdade. □

Através deste resultado, conclui-se que o comportamento médio da versão do algoritmo BM analisada é sublinear. Em particular, conforme apresentado naquele artigo, para textos e padrões em inglês, $\frac{1}{e} \approx 15.2$, o que implica que o teorema acima pode ser aplicado em padrões cujo comprimento é menor do que ou igual a 15 e, neste caso, o deslocamento esperado é de no mínimo $\frac{m}{2}$ posições.

B.4 Análises Empíricas

Os vários algoritmos descritos nos capítulos anteriores foram objetos de diversas análises empíricas, nas quais os autores avaliaram o comportamento destes algoritmos sob diferentes circunstâncias, por exemplo, a influência do comprimento do alfabeto no desempenho dos algoritmos. Dentre estas análises empíricas, podemos destacar as apresentadas em: [Smi82], [DB86], [Sch88], [BY89b], [BY89a], [HS91] e [Smi91].

Nesta seção apresentaremos resumidamente alguns dos resultados obtidos nestas análises, sendo que o principal objetivo desta apresentação é ilustrar o desempenho comparativo de alguns algoritmos descritos neste trabalho.

Inicialmente, na figura B.5, apresentamos um resultado obtido por Smit [Smi82], quando o autor avaliou o comportamento dos algoritmos Ingênuo, KMP e BM. Nesta análise, foi computada a razão entre o número de comparações efetuadas e o número de caracteres percorridos no texto até se obter a primeira ocorrência do padrão ou até que o final do texto fosse alcançado. Para cada m, $1 \leq m \leq 14$, foram efetuadas 20 buscas de padrões de comprimento m em um texto (escrito em *afrikaan*, língua da África do Sul) de comprimento 5000, sendo que para cada m foi gerado pelo menos um padrão que não ocorria no texto. No caso dos algoritmo KMP e BM também foram consideradas as comparações efetuadas durante o processamento do padrão.

Através do gráfico dado na figura B.5 podemos perceber que, na prática, o desempenho do algoritmo KMP é aproximadamente igual ao desempenho do algoritmo Ingênuo. Vale lembrar que, no pior caso, o comportamento destes algoritmos é bem diferente. Agora, no caso do algoritmo BM, temos que seu desempenho é ruim quando o comprimento do padrão é pequeno ($m \leq 3$). Uma justificativa para este fato é que, como o padrão é pequeno, então após cada rodada de verificações teremos um deslocamento também

pequeno, o que não compensa o custo de obtenção das funções s_1, s_{23} (unificação de s_2 e s_3). Por outro lado, se o padrão é um pouco maior ($m \geq 4$) então o desempenho do algoritmo BM é bem melhor do que o dos outros dois algoritmos. Vale notar que, para $m \geq 4$, o algoritmo BM não verifica todos os caracteres do texto, por exemplo, para $m = 10$, apenas cerca de 20 % dos caracteres percorridos no texto são verificados, ou seja, a cada 5 caracteres do texto apenas 1 é comparado, o que mostra a eficiência deste algoritmo.

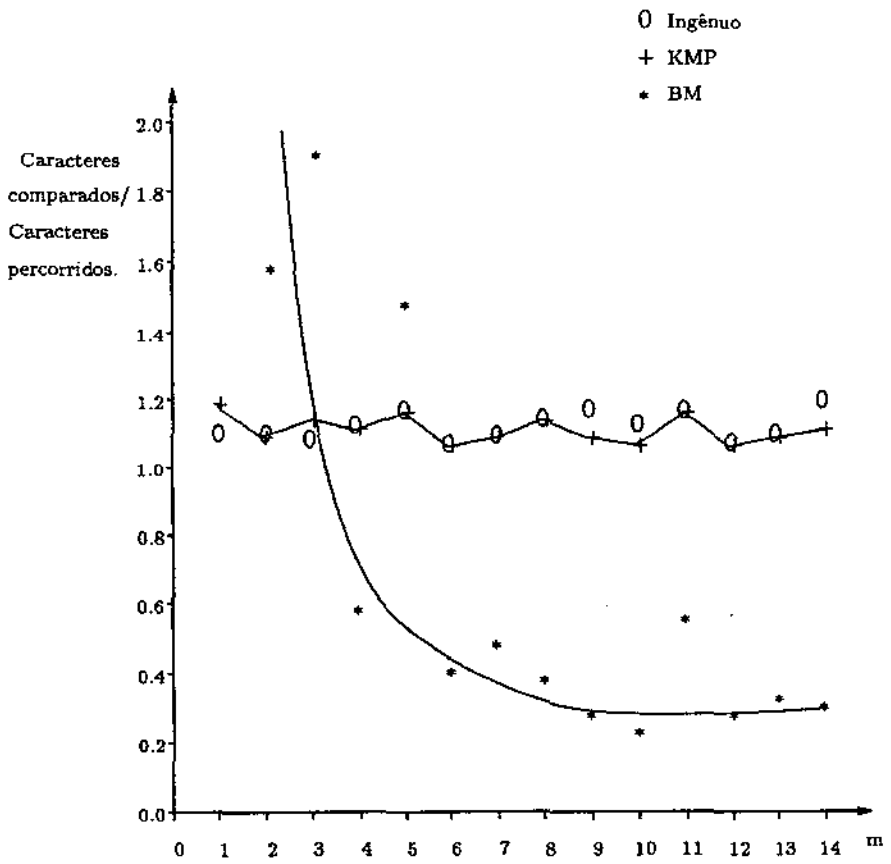


Figura B.5: Comportamento dos algoritmos Ingênuo, KMP e BM [Smi82].

Na figura B.6, apresentamos um resultado obtido por Davies e Bowsher [DB86], onde foi avaliado o comportamento dos algoritmos Ingênuo, KMP,

BM e KR (descrito na seção A.5). Nesta análise, os autores consideraram alfabetos de diversos comprimentos, sendo que no caso da figura aqui apresentada, mostramos os resultados considerando um alfabeto binário.

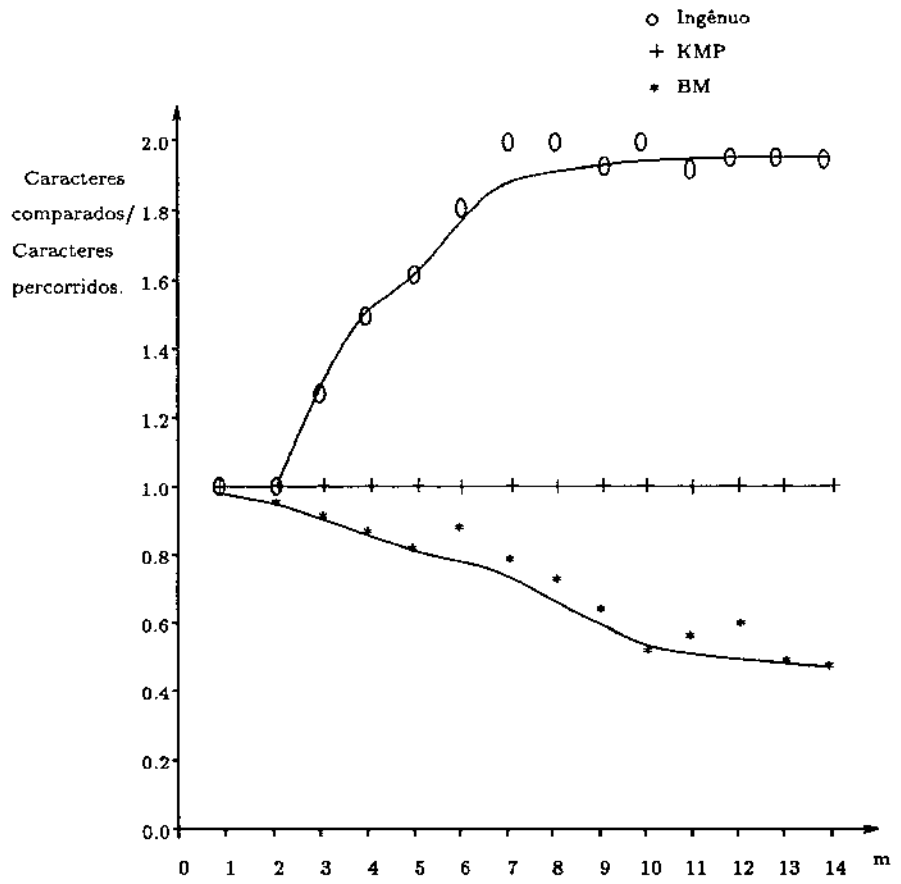


Figura B.6: Comportamento dos algoritmos Ingênuo, KMP, e BM considerando um alfabeto binário [DB86].

Comparando este gráfico com o anterior podemos perceber a influência do tamanho do alfabeto no desempenho dos algoritmos. No gráfico da figura B.5, onde $|\Sigma| \approx 30$, temos que os desempenhos dos algoritmos Ingênuo e KMP são bastante próximos. Por outro lado, no gráfico da figura B.6, percebemos que, para alfabetos binários, o algoritmo KMP é bem mais eficiente do que o algoritmo Ingênuo. Isto ocorre porque, como o alfabeto

é pequeno, então a probabilidade de que uma diferença ocorra no início do padrão é pequena e portanto, em média, uma diferença somente é obtida depois de várias comparações o que torna este caso parecido com o pior caso do algoritmo Ingênuo.

Observe que o desempenho do algoritmo BM também é pior do que aquele obtido com alfabetos maiores, mas ainda assim este algoritmo é sublinear. Neste caso, a justificativa para a queda na eficiência também se deve ao fato de que é pouco provável a obtenção de diferenças no início da rodada de verificações e portanto, quando uma diferença ocorre, em geral, já foram realizadas várias comparações e o deslocamento obtido (geralmente pequeno) não amortiza as comparações efetuadas.

A influência do comprimento do alfabeto no desempenho dos algoritmos KMP e BM pode ser verificada mais detalhadamente através das figuras B.3 e B.4.

Finalmente, na figura B.7 apresentamos os resultados de uma das análises efetuadas por Baeza-Yates [BY89b], na qual o autor compara o desempenho dos algoritmos Ingênuo, KMP, BM, BM simplificado (excluindo-se s_2 e s_3) e $\mathrm{BM_H}$. Nesta análise, foi avaliado o tempo de execução de cada algoritmo para efetuar a busca de 1000 padrões aleatórios em um texto também aleatório, sendo que $|\Sigma| = 30$.

As várias análises empíricas citadas no início desta seção, realmente confirmam que, em média, quanto maior o alfabeto e/ou o padrão melhor será o desempenho dos algoritmos (esta observação vale para todos os algoritmos que se baseiam em comparações). Isto se deve ao fato de que neste caso maior é a probabilidade de obtermos uma diferença após poucas comparações e, além disso, a probabilidade de "reocorrência" de prefixos ou de sufixos no padrão se torna menor, o que aumenta o deslocamento nos algoritmos (exceto no algoritmo Ingênuo). Adicionalmente, o algoritmo BM é mais sensível a este fato porque a probabilidade de que um caractere do texto ocorra no padrão se torna menor e isto faz com que a função s_1 forneça deslocamentos maiores.

Estas observações levaram Baeza-Yates a propor uma variação do algoritmo BM descrita em [BY89b] que em suma se baseia em expandir o alfabeto da seguinte forma: o padrão e o texto são transformados em "novas" seqüências onde o i-ésimo caractere é composto pela concatenação dos caracteres que estão nas posições $i, i+1, \ldots, i+k-1$ na seqüência original. Esta transformação produz um "novo" alfabeto cujo comprimento é c^k e,

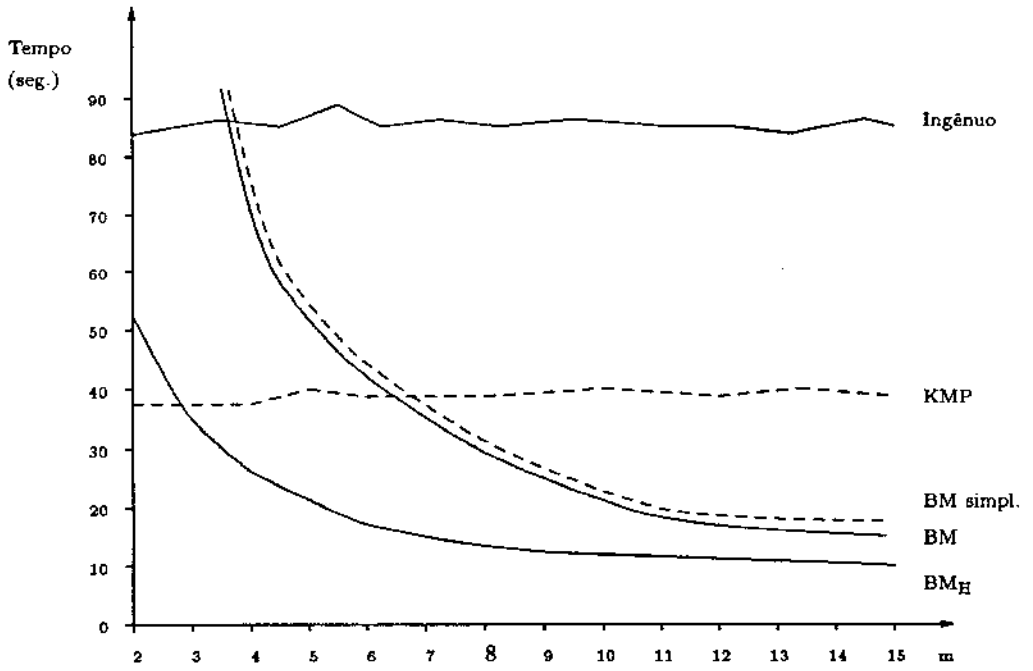


Figura B.7: Comportamento dos algoritmos Ingênuo, KMP, BM, BM simplificado e BM_H [BY89b].

por outro lado, reduz o comprimento do padrão para $m - k + 1$ e do texto para $n - k + 1$. Segundo o autor, para padrões de comprimentos razoáveis, isto torna o algoritmo, em média, cerca de 50 % mais eficiente.

Para encerrar esta seção, vale ressaltar que, através de inúmeras análises empíricas efetuadas, pode-se concluir que, em geral, o algoritmo que apresenta o melhor desempenho é o algoritmo BM_H. Entretanto, conforme apresentado em [HS91], em determinadas situações, por exemplo, quando o tamanho do alfabeto é pequeno (reconhecimento de DNA), outros algoritmos, como por exemplo o algoritmo de Giancarlo (descrito resumidamente em [HS91]), apresentam um melhor desempenho.

Além disso, embora não apresentemos aqui, através das análises descritas em [DB86] verifica-se que o tempo de execução do algoritmo KR, em todas as situações analisadas, é substancialmente maior do que o tempo de execução de todos os outros algoritmos considerados no gráfico da fi-

gura B.7. Isto ocorre porque o cálculo dos valores da função "hash" apresenta um custo computacional elevado, o que aumenta o tempo de execução do algoritmo.

Bibliografia

[AC75] A. V. Aho e M. J. Corasick. Efficient string matching : An aid to bibliography search. *Communications of the ACM*, 18(6):333–340, 1975.

[AG86] A. Apostolico e R. Giancarlo. The Boyer-Moore-Galil string searching strategies revisited. *SIAM Journal on Computing*, 15(1):98–105, 1986.

[AHU74] A. V. Aho, J. E. Hopcroft, e J. D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974.

[Baa91] S. Baase. *Computer Algorithms.* Addison-Wesley, 1991.

[Bar81] G. Barth. An alternative for the implementation of the Knuth-Morris-Pratt algorithm. *Information Processing Letters*, 13(5):134–137, 1981.

[Bar84] G. Barth. An analytical comparison of two string searching algorithms. *Information Processing Letters*, 18:249–256, 1984.

[BM77] R. S. Boyer e J. S. Moore. A fast string searching algorithms. *Communications of the ACM*, 20(10):762–772, 1977.

[BY89a] R. A. Baeza-Yates. Improved string searching. *Software-Practice and Experience*, 19(3):257–271, 1989.

[BY89b] R. A. Baeza-Yates. String searching algorithms revisited. *Lecture Notes in Computer Science*, 382:75–96, 1989.

[BYGR90] R. A. Baeza-Yates, G. H. Gonnet, e M. Regnier. Analysis of Boyer-Moore-type string searching algorithms. In *Proc. of the*

First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 328–343, 1990.

[BYR92] R. A. Baeza-Yates e M. Régnier. Average running time of the Boyer-Moore-Horspool algorithm. *Theoretical Computer Science*, 92:19–31, 1992.

[Cho90] C. Choffrut. An optimal algorithm for building the Boyer-Moore automaton. *Bulletin of European Assoc. Theoretical Computer Science*, 40:217–225, 1990.

[Cla82] A. Clausing. Kantorovich-type inequalities. *The American Mathematical Monthly*, 89:314–330, 1982.

[Col90] R. Cole. Tight bounds on the complexity of the Boyer-Moore pattern matching algorithm. Technical Report 512, Courant Institute, New York University, 1990.

[Col91] L. Colussi. Correctness and efficiency of pattern matching algorithms. *Information and Computation*, (95):225–251, 1991.

[CP91] M. Crochemore e D. Perrin. Two-way string-matching. *Journal of the ACM*, 38(3):651–675, 1991.

[DB86] G. Davies e S. Bowsher. Algorithms for pattern matching. *Software-Practice and Experience*, 16(6):575–601, 1986.

[FW65] N. J. Fine e H. S. Wilf. Uniqueness theorems for periodic Functions. In *Proc. Americ. Math. Soc.*, número 16, pp. 109–114, 1965.

[Gal79] Z. Galil. On improving the worst case running time of the Boyer-Moore string matching algorithm. *Communications of the ACM*, 22(9):505–508, 1979.

[GG91] Z. Galil e R. Giancarlo. On the exact complexity of string Matching: Lower bounds. *SIAM Journal on Computing*, 20(6):1008–1020, 1991.

[GG92] Z. Galil e R. Giancarlo. On the exact complexity of string Matching: Upper bounds. *SIAM Journal on Computing*, 21(3):407–437, 1992.

[GKP90] R. L. Graham, D. E. Knuth, e O. Patashnik. *Concrete Mathematics.* Addison-Wesley, Reading, Mass., 1990. (5^{th} printing with corrections).

[GO80] L. J. Guibas e A. M. Odlyzko. A new proof of the linearity of the Boyer-Moore string searching algorithm. *SIAM Journal on Computing*, 9(4):672–682, 1980.

[GO81] L. J. Guibas e A. M. Odlyzko. Periods in strings. *Journal of Combinatorial Theory*, Series A-30:19–42, 1981.

[GS83] Z. Galil e J. Seiferas. Time-space-optimal string matching. *Journal of Computer and System Sciences*, 26:280–294, 1983.

[Han93a] C. Hancart. *Analyse Exacte et en Moyenne d'Algorithmes de Recherche d'un Motif dans un Texte.* PhD thesis, Université Paris 7, Paris, 1993.

[Han93b] C. Hancart. On Simon's string searching algorithm. *Information Processing Letters*, 1993. a aparecer.

[Hor80] R. N. Horspool. Practical fast searching in strings. *Software-Practice and Experience*, 10:501–506, 1980.

[HS91] A. Hume e D. Sunday. Fast string searching. *Software-Practice and Experience*, 21(11):1221–1248, 1991.

[IM76] D. L. Isaacson e R. W. Madsen. *Markov Chains: Theory and Applications.* John Wiley and Sons, 1976.

[KMP77] D. E. Knuth, J. H. Morris, e V. R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[Knu69] D. E. Knuth. *The Art of Computer Programming*, volume 1. Addison-Wesley, Reading, Mass., 1969.

[KR87] R. M. Karp e M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*, 31(2):249–260, 1987.

[LS62] R. C. Lyndon e M. P. Schützenberger. The equation $a^M = b^N c^P$ in a free group. *Michigan Math. J.*, 9:289–298, 1962.

[Pir92] K. Pirklbauer. A study of pattern-matching algorithms. *Structured Programming*, 13:89–98, 1992.

[Rai92] T. Raita. Tuning the Boyer-Moore-Horspool string searching algorithm. *Software-Practice and Experience*, 22(10):879–884, 1992.

[Reg89] M. Regnier. Knuth-Morris-Pratt algorithm : An analysis. *Lecture Notes in Computer Science*, 379:431–444, 1989.

[Riv77] R. L. Rivest. On the worst-case behavior of string-searching algorithms. *SIAM Journal on Computing*, 6(4):669–674, 1977.

[Ryt80] W. Rytter. A correct preprocessing algorithm for Boyer-Moore string searching. *SIAM Journal on Computing*, 9(3):509–512, 1980.

[Sch88] R. Schaback. On the expected sublinearity of the Boyer-Moore algorithm. *SIAM Journal on Computing*, 17(4):648–658, 1988.

[Sed83] R. Sedgewick. *Algorithms*. Addison-Wesley, Reading, Mass., 1983.

[Sim93] I. Simon. String matching algorithms and automata. In *Proc. First South American Workshop on String Processing*, pp. 151–157, 1993.

[Smi82] G. De V. Smit. A comparison of three string matching Algorithms. *Software-Practice and Experience*, 12(1):57–66, 1982.

[Smi91] P.D. Smith. Experiments with a very fast substring search algorithm. *Software-Practice and Experience*, 21(10):1065–1074, 1991.

[Ste92] G. A. Stephen. String search. Technical Report 01, School of Eletronic Engineering Science, University College of North Wales, 1992.

[Sun90] D. M. Sunday. A very fast substring search algorithm. *Communications of the ACM*, 33(8):132–142, 1990.

[Tak86] T. Takaoka. An on-line pattern matching algorithm. *Information Processing Letters*, 22(6):329–330, 1986.

[Tar85] R. E. Tarjan. Amortized computational complexity. *SIAM J. Alg. Disc. Math.*, 6(2):306–318, 1985.

[Ziv86] N. Ziviani. *Projeto de Algoritmos e Estruturas de Dados.* I EBAI. Editora da UNICAMP, Campinas, SP, 1986.