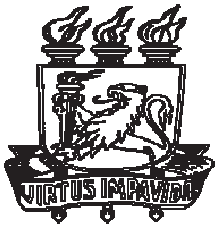


Universidade Federal de Pernambuco
Centro de Informática

Pós-graduação em Ciência da Computação

ÍNDICES COMPLETOS PARA CASAMENTO DE PADRÕES E INFERÊNCIA DE MOTIFS

Paulo Gustavo Soares da Fonseca

DISSERTAÇÃO DE MESTRADO

Recife
19 de Março de 2003

Universidade Federal de Pernambuco
Centro de Informática

Paulo Gustavo Soares da Fonseca

ÍNDICES COMPLETOS PARA CASAMENTO DE PADRÕES E INFERÊNCIA DE MOTIFS

Trabalho apresentado à Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: *Profa. Dra. Katia Silva Guimarães*

Recife
19 de Março de 2003

Ao meu pai pelo exemplo de caráter, fortaleza, disciplina e obstinação.

AGRADECIMENTOS

Valeu a pena? Tudo vale a pena
Se a alma não é pequena.
— FERNANDO PESSOA

Finda a jornada, a satisfação e o orgulho pelo dever cumprido equiparam-se ao sentimento de gratidão em relação a todos os co-responsáveis pelo sucesso do trabalho. A esses, cabe-me, agora, registrar meus profundos e sinceros agradecimentos.

Gostaria de agradecer aos meus professores do Centro de Informática, em especial a Geber Ramalho, Augusto Sampaio e Teresa Ludermir por terem me acolhido de volta a esta pós-graduação de maneira tão generosa. Devo também mencionar os demais funcionários do CCEN e do Centro de Informática da UFPE pela manutenção de um ambiente sempre tão amistoso e produtivo, especialmente as secretárias Lília e Neide e as bibliotecárias Jane e Raquel pelas idas e vindas e pelas prestimosas consultas ao COMUT.

Sou grato aos meus queridos mestres do Departamento de Matemática, André Rocha, Israel Vainsencher e Sérgio Santa Cruz não só por terem me apresentado aos verdadeiros caminhos da ciência mas, principalmente, pela amizade franca. Dificilmente serei capaz de retribuir à altura o que os Srs. têm feito por mim.

Agradeço aos membros da banca examinadora, Prof. André Santos, pela sua disposição e boa vontade em julgar este trabalho, e Profa. Marie-France Sagot que, além da sua inestimável contribuição a esta obra, concedeu-me também, a despeito dos seus inúmeros afazeres, a satisfação de tê-la presente à minha defesa.

Devo manifestar minha especial gratidão à Profa. Katia que, muito mais do que me orientar, soube me animar com seu espírito manso e paciente, ao mesmo tempo que alegre e determinado. Obrigado, entre outras coisas, por tantas vezes me fazer recobrar a confiança.

Por fim, e sobretudo, gostaria de agradecer aos meus pais, a quem devo não apenas esta conquista mas tudo o que sou, à minha esposa, Grácia, pelo amor, pelo companheirismo, pelo apoio e pela confiança inabaláveis, e ao meu filho, o pequeno Pedro, que, mesmo sem saber, ajudou-me tanto com seu sorriso largo e inocente. Custou mas conseguimos!

Era mais importante que eu aprendesse a usar
as minhas mãos do que a minha cabeça.
Na minha terra, as mãos produzem comida,
e a cabeça só produz confusão.

— MESTRE VITALINO, Artesão (Caruaru-PE)

RESUMO

Uma das maneiras mais eficientes (notadamente do ponto de vista computacional) empregada pela humanidade para a representação da informação tem sido através da forma de *texto*, ou seja, através de cadeias unidimensionais de símbolos (ou *caracteres*) tomados sobre conjuntos discretos finitos (ou *alfabetos*). As fecundas teorias, técnicas e algoritmos destinados ao processamento de texto têm ocupado um papel central em diversos âmbitos da Ciência da Computação, constituindo-se, sobretudo ao longo das últimas três décadas, em um campo de particular interesse no seio da grande área de Algoritmos e Estruturas de Dados.

Grande parte do recente interesse com respeito ao processamento de texto deve-se ao emergente ramo da ciência denominado *Biologia Molecular Computacional*, que, a grosso modo, comporta o estudo, através de técnicas matemáticas e computacionais, da estrutura e da função dos artefatos bio-moleculares responsáveis pela conformação e pelas atividades fisiológicas dos organismos vivos. A confluência dos problemas de Biologia Molecular e de processamento de textos dá-se na medida em que as estruturas macromoleculares fundamentais (DNA, RNA e proteínas) podem ser representadas através de cadeias (muito longas) de caracteres tomados sobre alfabetos (curtos) específicos.

O problema fundamental relacionado ao processamento de cadeias corresponde à determinação das ocorrências, exatas ou aproximadas, de um determinado *padrão* em um dado texto—*problema do casamento de padrões*—problema esse que admite inúmeras variações. Os problemas de casamento de padrões podem ser particionados em duas grandes categorias com respeito ao conhecimento prévio ou não do texto a ser examinado. Os algoritmos clássicos destinados à resolução do problema do casamento de padrões dizem respeito ao caso no qual o texto não é conhecido previamente. Nesse caso, cada um dos seus caracteres deve ser examinado pelo menos uma vez, o que resulta em soluções de custo, no mínimo, linearmente proporcional ao tamanho do texto. Se, todavia, o texto a ser examinado é conhecido *a priori*, então ele pode ser pré-processado (tipicamente em tempo linear), dando origem a uma estrutura auxiliar (tipicamente de tamanho linear) denominada *índice*, contra a qual os padrões podem então ser confrontados para que as suas ocorrências sejam determinadas. Nesse caso, o custo da solução ótima do problema é linearmente proporcional ao comprimento do padrão (em geral, muito menor do que o texto).

Em Biologia Molecular Computacional, freqüentemente estamos interessados em localizar as ocorrências de uma determinada subseqüência molecular (ou *motif*) dentro de estruturas maiores. Esses *motifs* representam, em geral, regiões altamente conservadas, *i.e.*, pouco afetadas por mutações, que desempenham funções biológicas específicas. Esse problema de *localização de motifs* limita-se com o problema do casamento de padrões e pode ser abordado através das mesmas técnicas. Em outras situações, todavia, estamos interessados não em localizar *motifs* mas sim em inferí-los. Isto é, dado um conjunto de seqüências moleculares, queremos descobrir que subseqüências aparecem repetidas em

uma quantidade significativa dessas seqüências de maneira suficientemente conservada e que, portanto, possuem uma boa probabilidade de representar um objeto biológico de particular interesse.

Neste trabalho, nos propomos a reunir em uma obra única, boa parte da informação fundamental dispersa na literatura acerca dos principais índices completos conhecidos, com ênfase nas suas propriedades estruturais. Nossa apresentação não intenciona ser estritamente panorâmica e, portanto, algum sacrifício da fluência deve ser depositado no altar do rigor matemático. Além disso, apresentamos uma análise crítica da adequação e desempenho das estruturas de índice estudadas para a resolução do problema da inferência de *motifs* através de algoritmos exatos e combinatórios.

Palavras-chave: casamento de padrões, estruturas de índice, árvores de sufixos, árvores de afixos, DAWGs, autômatos de sufixos, vetores de sufixos, localização de *motifs*, inferência de *motifs*.

ABSTRACT

One of the most efficient ways (remarkably from the computational standpoint) of representing information employed by humanity has been by means of *text*, that is, by means of unidimensional strings of symbols (or *characters*) taken over discrete finite sets (or *alphabets*). The fecund theories, techniques and algorithms aimed at text processing have been playing a major rôle in several realms of Computer Science thus constituting, especially over the last three decades, a field of particular interest in the heart of the major area of Algorithms and Data Structures.

Much of the recent interest on text processing can be credited to the emerging branch of science known as *Computational Molecular Biology* which, roughly speaking, concerns the study, through mathematical and computational techniques, of the structure and function of the bio-molecular artifacts responsible for the conformation and physiological activities of living organisms. The confluence of Molecular Biology and text-related problems is due to the fact that the fundamental macromolecular structures (DNA, RNA, proteins) can be represented by (very long) strings of characters over specific (short) alphabets.

The fundamental problem related to text processing concerns finding the occurences, either exact or approximate, of a (often small) given *pattern* in a (often large) given text—*string pattern matching problem*—this problem admitting numerous variations. The string matching problems can be partitioned into two big classes regarding the previous knowledge or not of the text to be examined. The classical string matching algorithms refer to the case in which the text is not previously known. In such case, each character of the text must be examined at least once thus resulting in solutions whose costs are at least linear on the size of the input text. If, on the contrary, the text is known *a priori*, then it can be preprocessed (tipically in linear time) giving birth to an auxiliary strucuture (typically of linear size) called an *index*, against which patterns can then be confronted in order to have their occurences determined. In this latter case the cost of the optimal solution is linear on the size of the pattern (usually much smaller than the text).

In Computational Molecular Biology, we are frequently interested in locating the occurences of a given molecular subsequence (or *motif*) within bigger structures. In general, those *motifs* stand for highly conserved regions, *i.e.* not much affected by mutations, that perform specific biological functions. This problem of *motif localization* very much corresponds to the string matching problem and therefore can be approached by the same techniques. In other occasions, though, we are not exactly interested in locating *motifs*, but in *inferring* them. That is, given a set of molecular sequences, we wish to discover which subsequences appear repeated in a significant number of those sequences in a sufficiently conserved fashion thus having a good probability of representing a biological object of particular interest.

In this work, we propose to put together in a single piece much of the fundamental

information spread across the literature about the main known full indexes, with emphasis put on their structural properties. As long as our presentation does not intend to be strictly panoramic, some sacrifice of fluency has to be laid upon the altar of mathematical rigour. Besides, we provide a critical analysis of the adequacy and performance of the examined structures to the resolution of the *motif* extraction problem by means of combinatoric exact algorithms.

Keywords: string matching, index structures, suffix trees, affix trees, DAWGs, suffix automata, suffix arrays, *motif* localization, *motif* inference.

SUMÁRIO

LISTA DE FIGURAS

CAPÍTULO 1

INTRODUÇÃO

Catar feijão se limita com escrever:
Joga-se os grãos na água do alguidar
E as palavras na da folha de papel;
E depois, joga-se fora o que boiar
— JOÃO CABRAL DE MELLO NETO

Uma das maneiras mais eficientes e amplamente empregadas pela humanidade para a representação da informação tem sido através da sua codificação na forma de *texto*. Por texto entenda-se, aqui de forma um pouco mais restritiva, uma cadeia unidimensional de símbolos (ou *caracteres*) tomados sobre um determinado conjunto discreto finito (ou *alfabeto*). Essa forma de representação da informação, embora não seja propriamente a mais expressiva, apresenta algumas características que a tornam particularmente apropriada para a manipulação (armazenagem, processamento e transmissão) por meio dos dispositivos computacionais, teóricos ou concretos, tais como os conhecemos hoje.

Os problemas de processamento de texto têm, historicamente, ocupado um papel central em diversos âmbitos da Ciência da Computação, tais como, projeto de compiladores, compressão de dados, recuperação de informação (*information retrieval*), Criptologia, *etc*. Embora ainda classicamente vista como uma sub-área dentro da grande área de "Algoritmos e Estruturas de Dados", o ramo que compreende os problemas relacionados ao processamento de cadeias constituiu-se, sobretudo ao longo das últimas três décadas, num campo de pesquisa de particular importância, com várias publicações e conferências internacionais a ele exclusivamente dedicados.

Não obstante a abrangência dos seus resultados, boa parte do interesse atual no que diz respeito à pesquisa na área de processamento de cadeias está relacionado ao recente esforço científico de caráter internacional para o sequenciamento e análise dos chamados *genomas* de diversos organismos, com destaque para o genoma humano. O problema do sequenciamento de genomas ou, mais geralmente, os diversos problemas relacionados à determinação da estrutura e função das estruturas bio-moleculares responsáveis pela conformação e atividades fisiológicas dos organismos vivos, fez surgir uma nova e multidisciplinar área de pesquisa conhecida como *Biologia Computacional* na qual os algoritmos para processamento de textos desempenham um papel central.

1.1 CASAMENTO DE PADRÕES

Dentre os diversos problemas de processamento de cadeias, o mais importante e freqüente de todos corresponde ao problema de determinar as ocorrências, exatas ou aproximadas, de um *padrão* Y (geralmente pequeno) em um texto X (geralmente grande). Esse problema, que admite inúmeras variações, é genericamente denominado *problema*

*do casamento de padrões**.

Problemas de casamento de padrões apresentam-se, naturalmente, em inúmeros contextos sob as mais diversas variações. Podemos classificá-los, a grosso modo, em dois eixos ortogonais que dizem repeito ao tipo de ocorrência a ser considerada (exata ou aproximada) e ao conhecimento prévio ou não do texto X a ser examinado. Temos, portanto, quatro situações:

i) Casamento exato *on-line* de padrões
ii) Casamento aproximado *on-line* de padrões
iii) Casamento exato de padrões via índices
iv) Casamento aproximado de padrões via índices.

Os dois primeiros casos acima referem-se a situações nas quais o texto X não é conhecido previamente. No primeiro caso, estamos interessados em localizar os trechos de X *idênticos* ao padrão Y. É nesse caso que se enquadram os resultados clássicos sobre casamento de padrões. A solução ótima do problema do casamento exato *on-line* de padrões apresenta custo computacional $O(|X| + |Y|)$ com uma quantidade constante de memória adicional ([GS83], [CR94, Cap. 13]).

Nas aplicações concretas, todavia, dificilmente estaremos interessados apenas nas ocorrências exatas do padrão Y no texto X. Com efeito, erros podem ocorrer tanto no texto quanto no padrão, quer seja devido a uma falha humana, como um erro de digitação, quer seja por um ruído qualquer de transmissão, *etc*. Afora esses "acidentes", existem situações nas quais estamos, de fato, interessados em regiões "parecidas" com o padrão procurado, uma vez que isso pode representar informação útil. Esse é, via de regra, o caso no âmbito da Biologia Computacional, uma vez que sabemos que, além dos potenciais erros durante o processo de seqüenciamento, por exemplo, as moléculas estão naturalmente sujeitas a *mutações* em conseqüência de processos evolutivos, de interações com o meio-ambiente, além das variações naturais entre as espécies.

O Capítulo 2 desta dissertação apresenta uma visão geral das principais abordagens para o problema do casamento exato e aproximado de padrões quando o texto X não é conhecido com antecedência. Embora esse não seja o tema central deste trabalho, essa discussão prévia é importante por três razões principais: primeiro, ela proporciona uma perspectiva histórica da área, segundo, ela favorece a compreensão dos problemas relacionados ao processamento de textos e, por fim, ela representa um contexto mais natural para a introdução de conceitos básicos que nos acompanharão nas discussões posteriores.

Os dois últimos casos acima serão descritos na seção a seguir.

1.2 ÍNDICES

Na seção anterior, consideramos o problema do casamento de padrões quando o texto X não é conhecido *a priori*. Em diversas situações práticas, entretanto, o caso é exatamente o oposto: a cadeia X é conhecida com antecedência e mantém-se fixa enquanto a procura pelas ocorrências do padrão Y é levada a termo repetidas vezes para

*Mais precisamente, estamos considerando o chamado problema do casamento *unidimensional* de padrões ou casamento de cadeias (*string matching* ou *string pattern matching*)

diferentes valores de Y. Esse é exatamente o caso, por exemplo, em diversas aplicações de Biologia Computacional nas quais queremos identificar a presença de determinados padrões em seqüências moleculares (DNA, RNA ou proteínas) previamente conhecidas.

Se o texto X é conhecido previamente, então ele pode ser submetido a um pré-processamento, originando uma estrutura conhecida como *índice*. Um índice pode ser conceitualmente encarado como uma estrutura composta por *chaves* na qual cada uma das chaves representa um ou mais trechos da cadeia considerada. Se um índice representa todos os trechos, ou *fatores* (vide Definição 1.3), de uma cadeia, então ele é dito um índice *completo*, em contraposição aos índices ditos *parciais* que representam apenas um sub-conjunto próprio desses fatores.

Um índice de um texto X normalmente permite que as ocorrências exatas de um padrão Y nesse texto sejam determinadas em tempo $O(|Y|)$ com uma relativa facilidade. Em geral, um *bom* índice de uma cadeia X, $I(X)$, deve apresentar pelo menos três características fundamentais:

i) I deve ser uma estrutura compacta, tipicamente de tamanho $O(|X|)$;
ii) I deve ser construído eficientemente, tipicamente em tempo $O(|X|)$;
iii) I deve permitir que o predicado $Y \subset X$ (Y ocorre em X) seja avaliado de maneira eficiente, tipicamente em tempo $O(|Y|)$.

O problema do casamento aproximado de padrões é, normalmente, resolvido através das mesmas estruturas de índice empregadas no caso exato, mediante alteração do algoritmo de busca [NBYST01]*.

Navarro *et al.* [NBYST01] propõem uma taxonomia para os índices destinados à solução do problema do casamento aproximado de padrões na qual os índices são classificados segundo dois critérios: a estrutura de dados subliminar e a estratégia de busca empregada. As estruturas de dados básicas consideradas nessa taxonomia são as *árvores de sufixos*, os *vetores de sufixos*, os *q-Grams* e os *q-Samples*, as duas primeiras dando origem a índices completos e as duas últimas originando índices parciais.

Nos capítulos 3, 4 e 5 desta dissertação examinaremos em detalhes diversos índices completos. Mais precisamente, serão estudadas as *árvores de sufixos*, as *árvores de afixos*, os *DAWGs* (*autômatos de sufixos*), *DAWGs compactos*, *CDAWGs simétricos*, *vetores de sufixos* e *vetores de sufixos compactos*. Essas estruturas (com destaque para as árvores de sufixos) respondem pela grande maioria dos algoritmos e aplicações práticas. Sagot [Sag02] enumera ainda duas outras estruturas menos utilizadas cujas descrições formais não constam neste trabalho, quais sejam, os *cactos de sufixos* [Kär95] e os *oráculos de fatores* [ACR99].

1.3 APLICAÇÕES À BIOLOGIA MOLECULAR: MOTIFS

1.3.1 Uma Introdução Minimalista à Biologia Molecular

1.3.1.1 DNA No final do Séc. XIX, o monge augustiniano Gregor Mendel, realizando experimentos com ervilhas, estabeleceu a existência de estruturas biológicas denominadas *genes* que seriam responsáveis pela expressão e transmissão de características

*Até o início dos anos 90, esse era considerado um dos principais problemas irresolutos da área de processamento de cadeias.

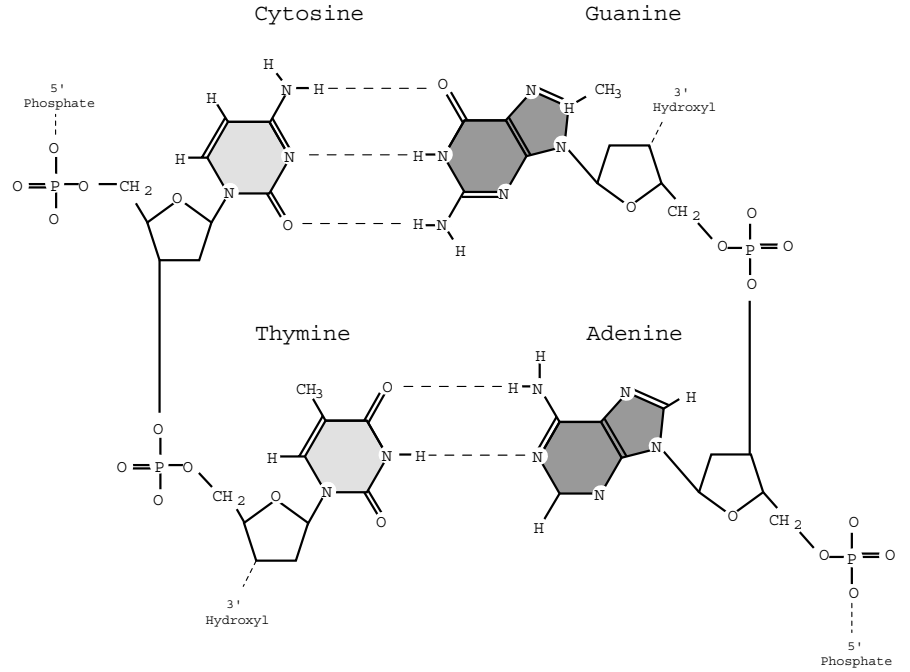
hereditárias nos indivíduos. O trabalho de Mendel deu origem ao ramo da Biologia atualmente conhecido como Genética. Até 1944, acreditava-se que o material genético estava contido em proteínas no interior das células até que Oswald Avery, Maclyn McCarty e Colin McLeod demonstraram que, na verdade, a molécula de DNA é o principal carregador do código genético dos organismos vivos.

O *ácido desoxiribonucléico*, ou *DNA*, é uma macromolécula constituída a partir de estruturas menores denominadas *nucleotídeos*. Cada nucleotídeo, por sua vez, é formado por uma molécula de açúcar (desoxiribose), um grupo fosfato e uma base nitrogenada que o identifica. As bases nitrogenadas são: a adenina (`A`), a guanina (`G`), a timina (`T`) e a citosina (`C`) (Figura 1.1(a)). Em 1953, James Watson e Francis Crick deduziram a estrutura tridimensional da molécula do DNA e, a partir daí, seu processo de replicação. O DNA possui a forma de um helicóide duplo (Figura 1.1(c)). Cada uma das "fitas" helicoidais representa uma seqüência de nucleotídeos. As duas fitas são mantidas juntas através de ligações químicas conhecidas como *pontes de hidrogênio* sendo que, no caso do DNA, essas ligações obedecem a uma regra estrita: as chamadas bases púricas (`A` e `G`) dos nucleotídeos de uma fita são unidas às chamadas bases pirimídicas (`T` e `C`) da outra fita, e vice-versa (Figura 1.1(a,b)), formando os chamados *pares de base* (*base pairs*). Mais precisamente, uma adenina de uma fita sempre se "emparelha" com uma timina da outra fita (e reciprocamente) ao passo que uma guanina de uma fita sempre se "emparelha" com uma citosina da outra (e reciprocamente). Como conseqüência dessa regra de formação, temos que as fitas na molécula do DNA, embora não sejam idênticas, são *complementares*, no sentido que uma determina a outra univocamente. Além disso, essas fitas possuem uma orientação ditada pelo número do átomo de carbono da molécula de açúcar na qual um nucleotídeo se liga ao outro. A orientação positiva é, convencionalmente, denominada orientação 5'-3', indicando que um nucleotídeo se liga ao anterior a ele na fita pelo carbono de número 5 e, ao posterior, pelo carbono de número 3. As orientações das fitas complementares são reversas entre si.

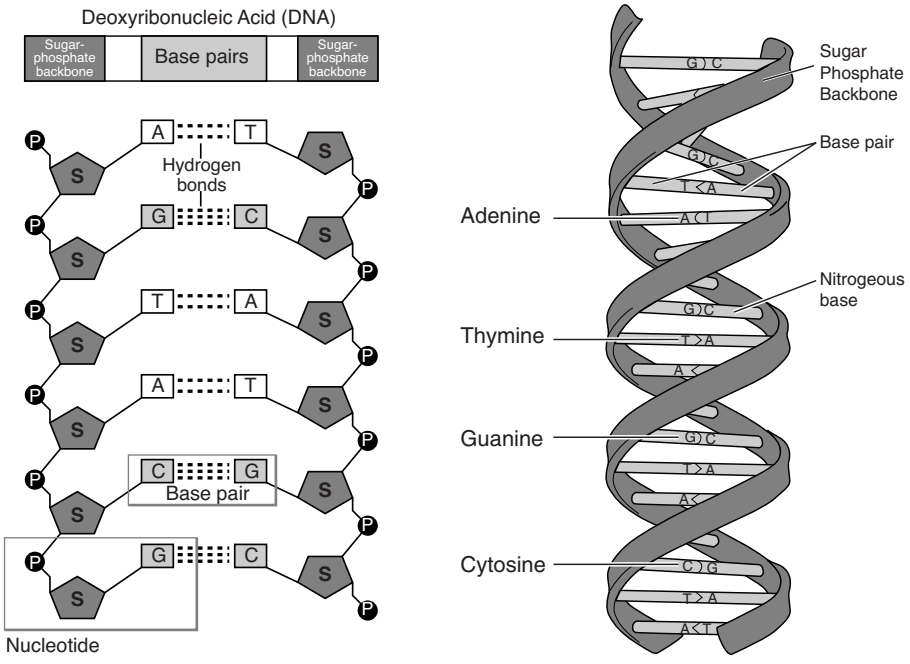
Da estrutura do DNA delineada no parágrafo anterior, podemos concluir diretamente que ele é passível de ser representado por uma cadeia de caracteres tomadas sobre o alfabeto $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Essa cadeia corresponde a uma das fitas na sua orientação positiva.

Um *gene* pode ser visto como um "trecho" ou "região" da molécula de DNA responsável pela codificação e expressão de alguma característica física do indivíduo, o que corresponde, geralmente, à informação necessária para a síntese de uma determinada proteína. O DNA não é ocupado totalmente por genes. Muito pelo contrário, apenas 2%-3% do DNA humano, por exemplo, é constituído por genes. Os restantes 97%-98% do DNA corresponde a material inter-gênico cuja função é, até o presente, desconhecida.

Cada molécula de DNA está, em geral, "empacotada" em um *cromossomo* que é uma estrutura que reside no interior das células (especificamente, no núcleo da célula, no caso dos eucariotes). O número de cromossomos varia de espécie para espécie. A informação correspondente ao DNA de todos os cromossomos de uma determinada espécie compõe aquilo se denomina o *genoma* da espécie. O genoma humano, por exemplo, possui cerca de $3,3 \times 10^9$bp, ou seja, mais de três bilhões de pares de base.



(a)



(b) (c)

Figura 1.1. O ácido desoxiribonucléico—DNA. (a) Composição química dos nucleotídeos e as suas ligações. (b) Os nucleotídeos de uma mesma fita são mantidos unidos através de uma espinha-dorsal de moléculas de açúcar e grupamentos fosfato (*sugar-phosphate backbone*) e ligam-se aos nucleotídeos da fita complementar através de pontes de hidrogênio: duas na ligação $\mathtt{A} = \mathtt{T}$ e três na ligação $\mathtt{G} \equiv \mathtt{C}$. (c) Conformação tridimensional da molécula de DNA. © *National Human Genome Research Institute* (www.genome.gov)

1.3.1.2 RNA e Síntese Protéica Proteínas são polímeros constituídos de moléculas menores (monômeros) denominadas *aminoácidos* unidas entre si através das chamadas *ligações peptídicas*. As proteínas são os elementos básicos constituintes das células, tecidos e órgãos dos indivíduos e também agem como hormônios, enzimas *etc*. A principal atividade celular na qual o DNA está envolvido é justamente a síntese de proteínas. Além do DNA, um outro tipo de ácido nucléico também desempenha um papel fundamental na síntese protéica: o *ácido ribonucléico* ou *RNA*. O RNA, assim como o DNA, é constituído de nucleotídeos, sendo a timina substituída pela *uracila* (`U`). Entretanto, ao contrário do DNA, o RNA é formado por uma única "fita" e apresenta uma conformação tridimensional bem mais variada do que a (dupla hélice) do DNA.

O processo de síntese de proteínas no interior das células ocorre, essencialmente, em duas etapas denominadas *transcrição* e *tradução*, esquematizadas na Figura 1.2 e descritas a seguir.

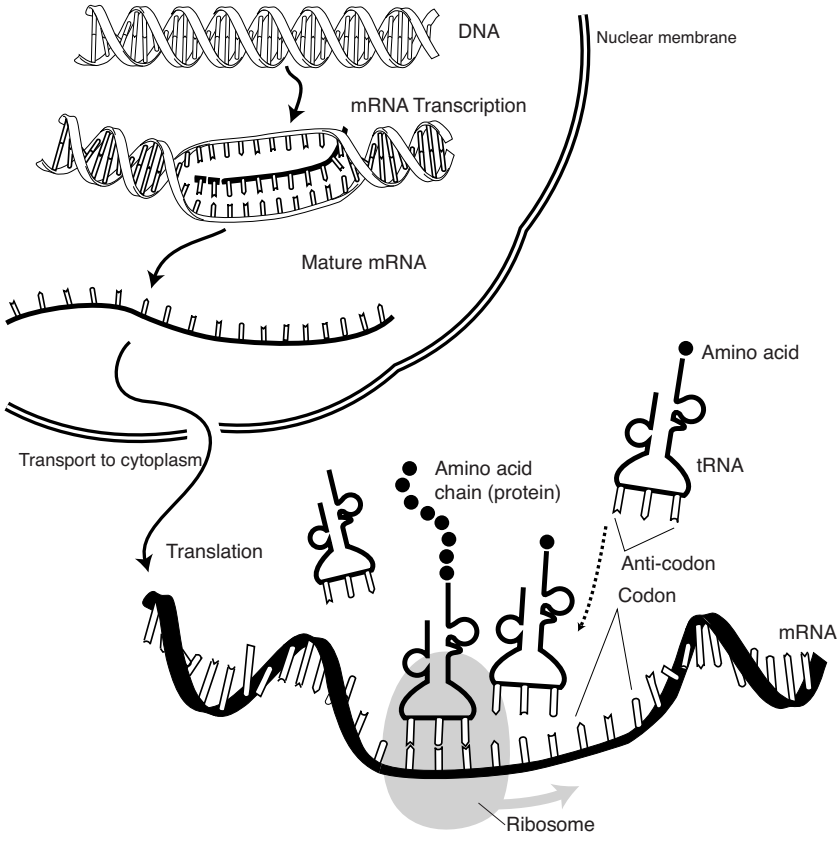


Figura 1.2. Síntese Protéica. © NHGRI (www.genome.gov)

Transcrição A síntese protéica inicia-se com a produção de um tipo de RNA, conhecido como *RNA mensageiro* ou *RNAm*, a partir de um gene. O processo de produção do RNAm ocorre, a grosso modo, da seguinte forma: o aparato celular reconhece o início de um gene a partir de alguns "sinais" (seqüências específicas de nucleotídeos) que

ocorrem nas chamadas *regiões promotoras*, que se estendem por algumas dezenas de pares de base antes do início dos genes. Uma enzima denominada *RNA polimerase* fixa-se ao DNA em pontos específicos da região promotora conhecidos como *sítios de ligação*, provocando uma ruptura localizada entre as suas duas fitas constituintes. A partir de então, a RNA polimerase "desliza" sobre uma das fitas do DNA gênico (no sentido 3'-5'), percorrendo cada desoxiribonucleotídeo (`C`, `G`, `T` ou `A`) e fazendo com que ribonucleotídeos complementares (resp. `G`, `C`, `A` ou `U`) sejam acrescentados à molécula de RNAm que, assim, vai-se formando ao longo do caminho (no sentido 5'-3'). Ao final do gene (também reconhecido pela RNA polimerase por um sinal específico) algumas dezenas de resíduos `A`'s são acrescentados ao RNAm, constituindo a chamada *cauda poli-A*, e a RNA polimerase desliga-se do DNA, que é então reconstituído.

Nos seres *procariotes*, dentre os quais incluem-se formas de vida primitivas como as bactérias e algas cianofíceas, o RNAm assim produzido já pode ser utilizado para a síntese de uma determinada proteína. Já nos seres denominados *eucariotes* (organismos pluricelulares com núcleo celular distinto), dentre os quais incluem-se a maioria dos organismos superiores como as plantas e animais, os genes são extratificados em trechos que contribuem para a síntese protéica chamados *éxons*, intercalados por trechos não-codificantes denominadas *íntrons*. Portanto, nos eucariotes, o RNAm recém-produzido a partir de um gene da maneira acima descrita ainda não está completamente maduro, sendo ainda submetido a um processo denominado *splicing* no qual, através da ação de um complexo de enzimas, os trechos correspondentes aos íntrons são removidos da molécula, permanecendo apenas os trechos codificantes.

Tradução Uma vez maduro, o RNAm abandona o núcleo celular (nos eucariotes) e migra para o *citoplasma* (líquido extra-nuclear no interior da célula). A *tradução* do RNAm em uma proteína dá-se, no interior de organelas celulares denominadas *ribossomos*, com o auxílio de um tipo de RNA conhecido como *RNA transportador* (*RNAt*), da seguinte forma: cada tripla ordenada de nucleotídeos consecutivos ao longo do RNAm constitui aquilo que se denomina um *códon*. Cada códon responde pela codificação de um determinado aminoácido conforme disposto na Figura 1.3. As moléculas de RNAt são, geralmente, pequenas (cerca de 80 resíduos) e possuem uma região com grande afinidade a um determinado códon, denominada *anti-códon*, e uma outra extremidade ligada a um determinado aminoácido correspondente ao códon afim de acordo com código genético da Figura 1.3. À medida em que o RNAm vai passando pelo ribossomo, um RNAt correspondente a cada um dos sucessivos códons acopla-se ao RNAm, trazendo consigo o aminoácido equivalente ao códon em questão. Através da ação de determinadas enzimas, esse aminoácido desprende-se do RNAt e une-se à cadeia polipeptídica que, assim, vai sendo progressivamente formada. A tradução é finalizada assim que um STOP códon (Figura 1.3) é detectado.

1.3.2 Motifs

Em Biologia Molecular Computacional, freqüentemente estamos interessados em localizar as ocorrências de uma determinada subseqüência molecular (ou *motif*) dentro de estruturas maiores como DNA, RNA e proteínas. Esses *motifs* representam, em ge-

1a. Base	2a. Base				3a. Base
	U	C	A	G	
U	Phe (F)	Ser (S)	Tyr (Y)	Cys (C)	U
	Phe (F)	Ser (S)	Tyr (Y)	Cys (C)	C
	Leu (L)	Ser (S)	STOP	STOP	A
	Leu (L)	Ser (S)	STOP	Trp (W)	G
C	Leu (L)	Pro (P)	His (H)	Arg (R)	U
	Leu (L)	Pro (P)	His (H)	Arg (R)	C
	Leu (L)	Pro (P)	Gln (Q)	Arg (R)	A
	Leu (L)	Pro (P)	Gln (Q)	Arg (R)	G
A	Ile (I)	Thr (T)	Asn (N)	Ser (S)	U
	Ile (I)	Thr (T)	Asn (N)	Ser (S)	C
	Ile (I)	Thr (T)	Lys (K)	Arg (R)	A
	Met (M)	Thr (T)	Lys (K)	Arg (R)	G
G	Val (V)	Ala (A)	Asp (D)	Gly (G)	U
	Val (V)	Ala (A)	Asp (D)	Gly (G)	C
	Val (V)	Ala (A)	Glu (E)	Gly (G)	A
	Val (V)	Ala (A)	Glu (E)	Gly (G)	G

Figura 1.3. O código genético. Repare que esse código é degenerado uma vez que os 64 (4^3) possíveis códons originam apenas 20 aminoácidos. Cada aminoácido possui um código de três letras e um outro código de uma letra apenas.

ral, regiões altamente conservadas, *i.e.*, pouco afetadas por mutações, que desempenham funções biológicas específicas, *e.g.* regiões promotoras, sítios de ligação, regiões reguladoras, *etc*. Esse problema de *localização de motifs* limita-se com o problema do casamento de padrões e pode ser abordado através das mesmas técnicas.

Em outras situações, todavia, estamos interessados não em localizar *motifs* mas sim em inferí-los. Isto é, dado um conjunto de seqüências moleculares, queremos descobrir que subseqüências aparecem repetidas em uma quantidade significativa dessas seqüências de maneira suficientemente conservada e que, portanto, possuem uma boa probabilidade de representar um objeto biológico de particular interesse.

O problema da localização e, sobretudo, da inferência de motifs é discutido no Capítulo 6 desta dissertação. Serão considerados os *motifs* constituídos de uma única seqüência contígua de resíduos como também os *motifs* constituídos de mais de uma subseqüência, separadas por intervalos de comprimentos limitados. Em particular, examinaremos a adequação e desempenho das estruturas de índices abordadas ao longo deste trabalho para a solução do problema da inferência de *motifs* através de algoritmos combinatórios e exatos baseados nos chamados *modelos*, introduzidos por Sagot [Sag98] (*modelos simples*) e Marsan-Sagot [MS00, Mar02] (*modelos estruturados*).

1.4 DEFINIÇÕES PRELIMINARES

Finalizamos esta introdução com uma apresentação um pouco mais formal, todavia sucinta, dos objetos principais do nosso discurso, as cadeias, juntamente com outros conceitos elementares que nos deverão acompanhar por toda a extensão deste texto.

Definição 1.1 (Alfabeto) *Um alfabeto $\Sigma = \{a_1, a, \ldots, a_s\}$ é um conjunto finito de símbolos (ou caracteres). O número de elementos distintos (*tamanho*) do alfabeto Σ é denotado por $|\Sigma|$.*

Na maioria das aplicações práticas, estaremos interessados em alfabetos de tamanho pequeno. Em problemas de Biologia Computacional, por exemplo, temos, tipicamente, $|\Sigma| = 4$ ou $|\Sigma| = 20$. Para boa parte das discussões a seguir, não é necessário levar em consideração nenhuma relação de ordem entre os elementos do alfabeto. Quando isto, eventualmente, se fizer necessário, redefiniremos Σ convenientemente.

Definição 1.2 (Cadeia) *Uma* cadeia *X de comprimento $n \in \mathbb{Z}_+$ sobre um alfabeto Σ é uma seqüência de símbolos (caracteres) da forma $X = x_1 \cdots x_n$, onde $x_1, \ldots, x_n \in \Sigma$. O comprimento de X é denotado por $|X|$. A cadeia de comprimento 0 é denominada* cadeia vazia *ou* nula *e denotada por ε.*

Repare que definimos as cadeias como *seqüências*, o que pressupõe uma ordenação dos seus caracteres constituintes. Ou seja, a cadeia $X = \texttt{ABC}$ é essencialmente diferente da cadeia $X' = \texttt{BAC}$ e estas duas são diferentes de $X'' = \texttt{CAB}$. Mais formalmente, definimos a igualdade entre duas cadeias de um dado alfabeto da seguinte forma:

$$x_1 \cdots x_n = y_1 \cdots y_m \iff n = m \ \wedge \ x_i = y_i, \forall i = 1, \ldots, n.$$

O conjunto de todas as cadeias de um alfabeto Σ é denotado por Σ^* e o conjunto de todas as suas cadeias não-nulas denota-se por Σ^+. Podemos definir em Σ^*, a operação *concatenação* ou *justaposição* de cadeias por:

$$\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$$
$$(X = x_1 \cdots x_n, Y = y_1 \cdots y_m) \mapsto X \cdot Y = XY = x_1 \cdots x_n y_1 \cdots y_m.$$

Por definição, $\varepsilon X = X\varepsilon = X$ (elemento neutro ou identidade). Repare que a operação de concatenação é associativa*, ou seja, $(XY)Z = X(YZ), \forall X, Y, Z \in \Sigma^*$, portanto, podemos escrever simplesmente XYZ de forma não-ambígua. Também é usual utilizar-se a notação de potência para expressar concatenações sucessivas, isto é, $X^k = \underbrace{X \cdots X}_{k \text{ vezes}}$.

O *reverso* da cadeia $X = x_1 \cdots x_n$ é a cadeia dada por $X^R = x_n \cdots x_1$.†

Definição 1.3 (Fator) *Dadas duas cadeias $X = x_1 \cdots x_n, Y = y_1 \cdots y_m \in \Sigma^*$, dizemos que Y é um fator de X, ou ainda, que Y ocorre em X, e escrevemos $Y \subset X$ se, e somente se, existem cadeias $U, V \in \Sigma^*$ tais que $X = UYV$. Se $Y \subset X$ e $Y \neq X$, dizemos que Y é um* fator próprio *de X.*

Um fator de uma cadeia $X = x_1 \cdots x_n$ pode ser identificado com um par ordenado de índices (i, j) que representam as suas posições inicial e final em X. Utilizamos a notação $X_{i..j}$, $1 \leq i \leq j \leq n = |X|$, para representar o fator $x_i x_{i+1} \cdots x_j$. Por exemplo, se $X = \texttt{ABCDE}$, então $X_{2..4} = \texttt{BCD}$. Por convenção, o fator nulo ε é representado por qualquer par (i, j) com $i > j$.

Denotamos por $\mathscr{F}(X)$ o conjunto de todos os fatores de uma determinada cadeia $X \in \Sigma^*$. Se $|X| = n$, então $\mathscr{F}(X)$ tem, no máximo, $(n - k + 1)$ fatores de comprimento k, donde

$$\begin{aligned} |\mathscr{F}(X)| &\leq \sum_{k=1}^{n}(n-k+1) = \sum_{k=1}^{n}(n+1) - \sum_{k=1}^{n} k \\ &= n(n+1) - n\frac{(n+1)}{2} = n\frac{(n+1)}{2} \\ &= \frac{n^2}{2} + \frac{n}{2}. \end{aligned} \tag{1.1}$$

A tabela a seguir contém, a título de ilustração, uma enumeração de todos os elementos de $\mathscr{F}(X = \texttt{ABCDE})$.

Compr.	*Qtde.*	*Fatores*				
1	5	A	B	C	D	E
2	4	AB	BC	CD	DE	
3	3	ABC	BCD	CDE		
4	2	ABCD	BCDE			
5	1	ABCDE				

*A tripla $(\Sigma^*, \cdot, \varepsilon)$ constitui-se em uma estrutura algébrica conhecida como *monóide*, isto é, um conjunto munido de uma operação associativa com elemento identidade.

†É comum encontrar, na literatura, X^{-1} para denotar o reverso de X. Não adotamos essa notação por ser ela inconsistente com a notação de potência aqui estabelecida.

Um fator Y de X é dito *aninhado* se ele ocorre em duas ou mais posições distintas de X, *i.e.* $\exists U, U', V, V' \in \Sigma^*$ t.q. $X = UYV = U'YV'$ com $U \neq U'$. Em particular, ε sempre é aninhado seja X qual for. Dizemos que o fator aninhado Y *deriva* à direita (respec. à esquerda) em X se existem caracteres $a \neq b \in \Sigma$ tais que $Ya, Yb \in \mathscr{F}(X)$ (respec. $aY, bY \in \mathscr{F}(X)$). Por exemplo se $X = \texttt{ABCBAB}$, então o $Y = \texttt{AB}$ é aninhado mas não deriva nem à direita nem à esquerda, ao passo que $Y' = \texttt{B}$ deriva à direita e à esquerda.

Há dois tipos de fator de particular interesse: os prefixos e sufixos. O prefixo de comprimento k, ou k-prefixo, da cadeia $X = x_1 \cdots x_n$ é definido como $X_{..k} = x_1 x_2 \cdots x_k$, se $k < n$, ou $X_{..k} = X$, se $k \geq n$. Já o sufixo de X iniciado na posição $k \leq n$ é dado por $X_{k..} = x_k x_{k+1} \cdots x_n$. Também por definição, ε é prefixo e sufixo de X, qualquer que seja X ($\varepsilon = X_{..0} = X_{n+1..}$). O conjunto de todos os prefixos de X será denotado por $\mathscr{P}(X)$ e o conjunto de todos os sufixos de X será denotado por $\mathscr{S}(X)$.

Ainda no que diz respeito aos prefixos e sufixos de uma cadeia, consideramos duas relações binárias $\sqsubset$ ("prefixo de") e $\sqsupset$ ("sufixo de") assim definidas:

$$Y \sqsubset X \iff Y \in \mathscr{P}(X)$$
$$Y \sqsupset X \iff Y \in \mathscr{S}(X)$$

Repare que as relações acima são relações de *ordem parcial* uma vez que, para duas cadeias quaisquer X e Y, temos:

i) $X \sqsubset X$ (reflexiva)
ii) $Y \sqsubset X \wedge X \sqsubset Y \Rightarrow Y = X$ (anti-simétrica)
iii) $Y \sqsubset X \wedge X \sqsubset Z \Rightarrow Y \sqsubset Z$ (transitiva),

o mesmo valendo para $\sqsupset$. Portanto, o conjunto Σ^* munido da relação $\sqsubset$ (igualmente, $\sqsupset$) é dito um *conjunto parcialmente ordenado* ou *poset*.

Definição 1.4 (Borda) *A* borda *de uma cadeia X, $brd(X)$, é definida como sendo o maior fator próprio de X que é, ao mesmo tempo, prefixo e sufixo de X.*

Por exemplo, $brd(\texttt{ABABABA}) = \texttt{ABABA}$, $brd(\texttt{ABACABA}) = \texttt{ABA}$ e $brd(\texttt{ABABABC}) = \varepsilon$.

Notação

Ao longo do texto, manteremos as seguintes convenções:

- Cadeias serão representadas pelas letras latinas maiúsculas U, V, W, X, Y e Z, ao passo que seus caracteres constituintes serão representados pelos seus correspondentes minúsculos indexados, *e.g.* o terceiro caractere da cadeia X será denotado por x_3;
- Caracteres serão representados pelas letras latinas minúsculas a, b, c, d, e;
- Constantes literais serão tipografadas em fonte mono-espaçada, *e.g.* "ABCDE", com exceção da cadeia vazia, representada por ε;
- Números inteiros serão representados pelas letras latinas minúsculas i, j, k, l, m, n, p, q, r, s e t.

- Os nós dos grafos (árvores inclusive) e autômatos serão denotados primariamente pelas letras latinas minúsculas u, v e w. A aresta com rótulo Y, eferente a u e aferente a v será denotada por $u \xrightarrow{Y} v$.

CAPÍTULO 2

CASAMENTO DE PADRÕES

O universo não pode ser lido até que nós tenhamos aprendido a linguagem e nos familiarizado com os caracteres com os quais ele é escrito. Ele é escrito em linguagem matemática e as letras são triângulos, círculos e outras figuras geométricas sem os quais é humanamente impossível compreender uma só palavra.

— GALILEO GALILEI (Opere Il Saggiatore p. 171)

Neste capítulo, versaremos sobre o problema do casamento de padrões na sua forma *exata* e *aproximada*. Estaremos considerando o caso no qual a cadeia a ser varrida não é conhecida *a priori* e, portanto, não pode ser pré-processada. Todavia, cumpre salientar que nosso objetivo não é o de promover, aqui, um estudo extensivo das inúmeras técnicas, estruturas e algoritmos destinados à resolução dessa variação do problema, sob pena de nos desviarmos, de maneira demasiadamente alongada, do tema central deste trabalho que diz respeito, precisamente, ao caso oposto. Intencionamos, antes, prover informação na medida apenas suficiente para a perfeita compreensão do problema, das suas principais dificuldades e suas principais abordagens, até mesmo para estabelecer critérios de comparação com as soluções baseadas no emprego de estruturas de índice.

2.1 CASAMENTO EXATO DE PADRÕES

O problema básico relacionado ao processamento de cadeias que origina toda uma classe de problemas conhecida como *problemas de casamento de padrões* (*string matching problems*) pode ser colocado da seguinte forma:

Problema do casamento de padrões (exato) Dadas duas cadeias* $X, Y \in \Sigma^+$, de comprimentos n e m respectivamente, com $m < n$, determinar todas as ocorrências de Y em X.

Mais formalmente, o problema do casamento exato de padrões resume-se à resolução no universo $\mathbb{U} = \{1, \ldots, n - m + 1\}$, da seguinte equação em i:

$$X_{i..i+m-1} = Y.$$

Por exemplo, se $X = \texttt{ABCBCD}$ e $Y = \texttt{BC}$, então o conjunto solução é dado por $\mathbb{S} = \{2, 4\}$. Por definição, se $Y = \varepsilon$ e $X = x_1 \cdots x_n \neq \varepsilon$ então o conjunto solução é dado por $\mathbb{S} = \{1, \ldots, n\}$.

*Na literatura de casamento de padrões, é comum referir-se a X como *texto* e a Y como *padrão*.

2.1.1 Janela Deslizante

A solução mais imediata que se pode pensar para o problema básico do casamento de padrões baseia-se na idéia simples de comparar, caractere por caractere, o padrão Y com todos os fatores de comprimento m, $X_{i..i+m-1}$, da cadeia de entrada, com o índice i variando de 1 até $n - m + 1$. O Algoritmo 2.1 correspondente a essa abordagem é conhecido como *Algoritmo Força-bruta* ou *Algoritmo Ingênuo*.

```
1  Algoritmo Força-bruta (X = x_1 ··· x_n, Y = y_1 ··· y_m)
2  início
3    para i ← 1 até n − m + 1 faça
4      j ← 0
5      enquanto X_{i+j} = Y_{j+1} e j < m faça
6        j ← j + 1
7      fim-faça
8      se j = m então
9        𝕊 ← 𝕊 ∪ {i}
10     fim-se
11   fim-faça
12 fim
```

Algoritmo 2.1. Algoritmo Ingênuo para casamento exato de padrões.

O custo do Algoritmo Força-bruta é determinado pelo número de comparações de símbolos (linha *5*). Essa complexidade de tempo é da ordem $O(mn)$, o pior caso ocorrendo, por exemplo, quando $X = a^n$ e $Y = a^{m-1}b$.

A técnica empregada pelo Algoritmo 2.1 consiste, conceitualmente, em manter uma *janela* de tamanho m deslocando-se sobre o texto X, comparando-se, a cada passo, o fator de X visível através da janela com o padrão procurado Y. Se essa comparação, ou *tentativa*, resultar em igualdade, então uma ocorrência (*match*) é reportada para a posição de X correspondente à extremidade esquerda da janela. Esse mecanismo é conhecido como *janela deslizante*. A Figura 2.1 ilustra a execução do procedimento para $X =$ `DADABACBADCDACDABACBDCADB` e $Y =$ `DABACB`.

Observe-se que, no caso do Algoritmo Força-bruta, a janela mantém-se deslocando-se sempre para a direita em "passos" de tamanho um. Além disso, o fator de X visível através da janela é comparado a Y da esquerda para a direita, caractere por caractere. O dsipositivo da janela deslizante, entretanto, não está restrito a esse comportamento. Em geral, a largura da janela, o tamanho dos seus deslocamentos e a estratégia da comparação no seu interior podem variar, dando origem a outros algoritmos mais eficientes porém baseados no mesmo princípio. A grande maioria dos algoritmos para casamento exato de padrões sem utilização de índices emprega o modelo da janela deslizante. Charras e Lecroq [CL97a] catalogam mais de trinta algoritmos baseados nesse mecanismo. Além do já apresentado Algoritmo Ingênuo, discorreremos aqui sobre quatro dos algoritmos mais representativos para o problema em questão, tanto do ponto de vista teórico quanto prático, além do aspecto da relevância histórica.

O primeiro algorimo apresentado é devido a Knuth, Morris e Pratt [KMP77], tendo

Figura 2.1. Janela deslizante do Algoritmo Ingênuo. A figura ilustra os passos $i = 1, 2, 3, 15$ e 20. Neste caso, as ocorrências são constatadas para $i = 3$ e 15.

sido esse o primeiro algoritmo linear no pior caso destinado à resolução do problema. O segundo algoritmo, devido a Boyer e Moore [BM77], é contemporâneo do Knuth-Morris-Pratt (KMP) e, apesar de exigir tempo quadrático no pior caso, tem um comportamento sublinear no caso médio, o que o torna bastante atraente e amplamente utilizado em aplicações práticas. Ao contrário do Algoritmo KMP, que realiza a comparação dentro da janela da esquerda para a direita, o Algoritmo Boyer-Moore (BM) faz essa comparação no sentido inverso, ou seja, da direita para a esquerda. O terceiro algoritmo apresentado é um algoritmo probabilístico devido a Karp e Rabin [KR87] que se baseia na utilização de uma função de *hashing* para a comparação do padrão procurado com o conteúdo da janela. Por fim, trataremos do Algoritmo *Shift-Or* [BYG92], que tira proveito, de maneira engenhosa, de características específicas (porém bastante razoáveis) do *hardware* subjacente.

2.1.2 O Algoritmo Knuth-Morris-Pratt

A grande deficiência do Algoritmo Ingênuo reside na sua inabilidade em reaproveitar o resultado de uma tentativa com o objetivo de abreviar a(s) tentativa(s) subseqüente(s).

O algoritmo proposto por Knuth, Morris e Pratt [KMP77], que corresponde a uma otimização de um algoritmo anterior devido apenas a Morris e Pratt, propõe-se a corrigir essa deficiência, conforme veremos a seguir, ao custo do pré-processamento do padrão Y e da conseqüente utilização de $O(m)$ espaço adicional.

Suponha que, na i-ésima tentativa do Algoritmo Força-bruta, tenhamos uma coincidência entre os j primeiros símbolos de Y e do fator delimitado pela janela (*i.e.*, $X_{i..i+m-1}$) e que essas cadeias divirjam pela primeira vez no $(j+1)$-ésimo caractere. Ou seja, $X_{i..i+j-1} = U = Y_{..j}$ e $a = X_{i+j} \neq Y_{j+1} = b$. Tal situação encontra-se ilustrada na Figura 2.2(a). No Algoritmo Ingênuo, essa tentativa resultaria em falha e a janela seria deslocada um caractere à direita, onde teria início uma nova comparação de Y com o conteúdo da janela, caractere a caractere, da esquerda para a direita. Repare, entretanto, que para qualquer ocorrência de Y em X iniciada na posição $i+k$, para $1 \leq k < j$, devemos ter, necessariamente, uma coincidência entre o fator $X_{i+k..i+j-1}$ e o prefixo $Y_{..j-k}$ (Figura 2.2(b)). Observe que, se for esse o caso, então teremos um alinhamento entre um prefixo de Y (também prefixo de U) e um sufixo de U, ambos de comprimento $j-k$ que, na Figura 2.2(b) está representado por V. Da definição de *borda* (página 11), temos que $|V| \leq |brd(U)|$ e, portanto, podemos afirmar que $k \geq (j - |brd(U)|)$.

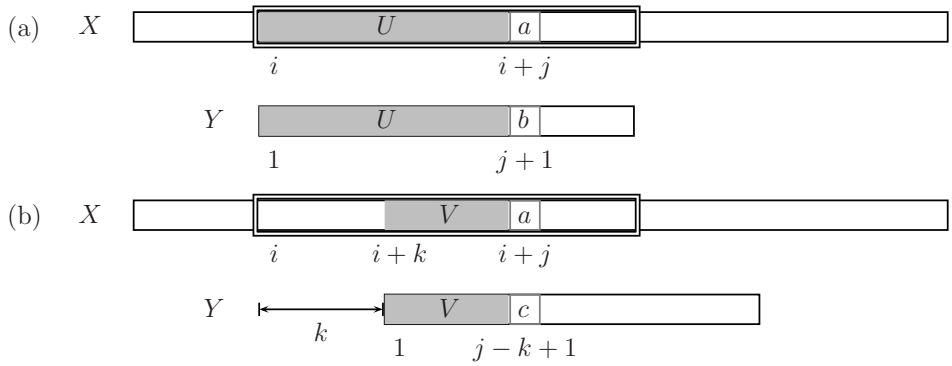


Figura 2.2. Deslocamento da janela no Algoritmo KMP.

A idéia fundamental do Algoritmo KMP é utilizar as observações do parágrafo anterior de modo que a janela seja deslocada $k \geq 1$ posições à direita sem que nenhuma ocorrência de Y seja negligenciada. Além disso, na tentativa subseqüente, a comparação deve ser iniciada a partir dos símbolos Y_{j-k+1} e X_{i+j}. Vimos que podemos tomar, com segurança, $k = j - |brd(U)|$. Entretanto, se o caractere $c = Y_{j-k+1}$, imediatamente posterior a $V = brd(U)$, for igual a b, então fatalmente teremos uma colisão (*mismatch*) entre c e $a = X_{i+j}$. Essa colisão pode ser prevenida se consideramos o que chamamos de *borda estrita* do prefixo $Y_{..j} = U$, ao invés da sua borda.

Definição 2.1 (Borda Estrita) *Dada uma cadeia $Y = y_1 \cdots y_m$ e um inteiro $j < m$, a* borda estrita *do prefixo $Y_{..j}$, $sbrd(Y_{..j})$ é dada pelo maior prefixo de Y que é também um sufixo próprio de $Y_{..j}$ e cujo caractere subseqüente em Y difere de y_{j+1}, se este prefixo existir. Formalmente:*

$$sbrd(Y_{..j}) = \max\{k \mid Y_{..k} = Y_{j-k+1..j} \wedge y_{k+1} \neq y_{j+1}\}.$$

Por definição, $sbrd(Y_{..m}) = brd(Y)$ e $sbrd(Y_{..0}) = \perp$, onde $\perp$ denota um valor indeterminado.

A Figura 2.3 ilustra a diferença entre as bordas e as bordas estritas dos prefixos de $Y = \texttt{ABAAB}$

Y	A	B	A	A	B
j	1	2	3	4	5
$brd(Y_{..j})$	ε	ε	A	A	AB
$sbrd(Y_{..j})$	ε	$\perp$	A	ε	AB
$next$	0	-1	1	0	2

Figura 2.3. Bordas e bordas estritas dos prefixos de $Y = \texttt{ABAAB}$.

No Algoritmo Knuth-Morris-Pratt (Algoritmo 2.3), o padrão Y é pré-processado, dando origem a um vetor $next(Y) = (next_1, \ldots, next_m)$, onde $next_j = |sbrd(Y_{..j})|$. O vetor $next(\texttt{ABAAB})$ encontra-se ilustrado na última linha da Figura 2.3. Tomamos, por definição, $|\perp| = -1$. A computação eficiente do vetor $next$ (Algoritmo 2.2) corresponde, essencialmente, a um subproduto da execução do próprio Algoritmo KMP comparando o padrão Y com ele mesmo, conforme ilustra a Figura 2.4.

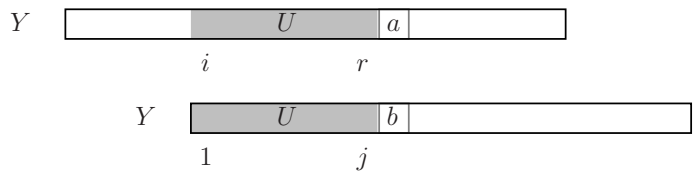


Figura 2.4. Tentativa da função *borda_estrita*. O prefixo $Y_{..j}$ é alinhado com o sufixo de $Y_{..r}$ ($r = i+j-1$) iniciado na posição i. Portanto, $Y_{..j}$ é candidato a borda estrita de $Y_{..r}$, restando, apenas, certificar-se de que $b = y_{j+1} \neq y_{r+1} = a$. Se esse for o caso, então $Y_{..j}$ é, de fato, a borda estrita de $Y_{..r}$. Do contrário, a borda estrita de $Y_{..r}$ é a borda estrita de $Y_{..j}$.

A complexidade do Algoritmo Knuth-Morris-Pratt no que diz respeito ao tempo, está relacionada ao número de comparações de símbolos. Esse número é linear no tamanho da entrada, $O(m+n)$. A função *borda_estrita*, por razões análogas, também demanda tempo $O(m)$, além do espaço adicional requerido pelo vetor $next$, igualmente $O(m)$.

2.1.3 O Algoritmo Boyer-Moore

O Algoritmo proposto por Boyer e Moore [BM77] é um dos mais utilizados em aplicações práticas e difere fundamentalmente do Algoritmo Knuth-Morris-Pratt por efetuar a comparação do padrão Y com o conteúdo da janela da direita para a esquerda, ao contrário daquele, no qual essa comparação é realizada da esquerda para a direita. Em ambos os casos, entretanto, o deslocamento da janela dá-se no sentido esquerda-direita.

```
1  Função borda_estrita (Y = y_1 ··· y_m)
2  início
3     V ← (v_1 = −1, ..., v_m = −1)
4     i ← 2
5     j ← 0
6     enquanto i ≤ m faça
7        enquanto y_{i+j} = y_{j+1} e i + j ≤ m faça
8           j ← j + 1
9           se v_{i+j−1} = −1 então
10             se i + j − 1 = m ou y_j ≠ y_{i+j} então
11                v_{i+j−1} ← j
12             senão
13                v_{i+j−1} ← v_j
14             fim-se
15          fim-se
16       fim-faça
17       i ← i + j − v_j
18       j ← max{0, v_j}
19    fim-faça
20    devolva V
21 fim
```

Algoritmo 2.2. Função *borda_estrita*.

```
1  Algoritmo Knuth-Morris-Pratt (X = x_1 ··· x_n, Y = y_1 ··· y_m)
2  início
3     next(Y) ← borda_estrita(Y)
4     i ← 1
5     j ← 0
6     enquanto i ≤ n − m + 1 faça
7        enquanto x_{i+j} = y_{j+1} e j < m faça
8           j ← j + 1
9        fim-faça
10       se j = |Y| então
11          S ← S ∪ {i}
12       fim-se
13       i ← i + j − next_j
14       j ← max{0, next_j}
15    fim-faça
16 fim
```

Algoritmo 2.3. Algoritmo Knuth-Morris-Pratt.

O Algoritmo Boyer-Moore faz uso simultâneo de duas heurísticas para manter a janela deslocando-se sempre para a direita: a heurística do *mau caractere* (Figura 2.5) e a heurística do *bom sufixo* (Figura 2.6).

A heurística do mau caractere baseia-se no seguinte princípio: suponha uma tentativa do algoritmo na qual a extremidade esquerda da janela está sobre a posição i. O algoritmo procede comparando o padrão Y com o fator $X_{i..i+m-1}$ da direita para a esquerda, até que uma colisão é detectada entre o mau caractere $a = x_{i+j-1}$ e $y_j = b$. Seja $k = \lambda(a) = \max(\{0\} \cup \{r \mid 1 \leq r \leq m \wedge y_r = a\})$, *i.e.* $\lambda(a)$ corresponde ao índice da ocorrência mais à direita de a em Y. Temos duas situações:

i) Se $0 \leq k < j$ então não existe nenhuma ocorrência de Y em X iniciada em $i + l$, para algum $0 < l < j - k$, pois, se houvesse, teríamos alinhados os caracteres $x_{i+j-1} = a = y_{j-l}$, o que contradiz a maximalidade de k, uma vez que $l < j - k \Rightarrow k < j - l$. A Figura 2.5(a) ilustra essa situação no caso em que $k > 0$ e a Figura 2.5(b) ilustra o caso em que $k = 0$, *i.e.*, quando a não ocorre em Y;

ii) Se $j < k \leq m$ então, por um argumento simétrico, podemos afirmar que não existe nenhuma ocorrência de Y em X iniciada em $i + l$, para algum $j - k < l < 0$.

Sumarizando os dois itens acima, podemos dizer que a heurística do mau caractere propõe, nessa situação, um deslocamento "seguro" para a janela de $j - k$ posições. Ocorre que, no segundo caso, o que a heurística sugere é um deslocamento negativo, ou seja, à esquerda, sugestão esta que deve ser desconsiderada. Os valores de $\lambda(a)$ para todos os símbolos $a \in \Sigma$ podem ser facilmente computados pela função *última_ocorrência* (Algoritmo 2.4).

```
1  Função última_ocorrência (Y = y_1 ··· y_m)
2  início
3     % Consideramos uma enumeração do alfabeto Σ = {a_1, ..., a_s}
          e identificamos cada b ∈ Σ com a sua posição, i.e. b ≡ j ⇔ b = a_j.
4     % Λ = (λ_1, ..., λ_s) é um vetor t.q. λ_j = λ(b = a_j)
5     Λ ← (λ_1 = 0, ..., λ_s = 0)
6     para j ← 1 até m faça
7        λ_{y_j} ← j
8     fim-faça
9     devolva Λ
10 fim
```

Algoritmo 2.4. Função *última_ocorrência*.

Para complementar a informação da heurística do mau caractere, o Algoritmo Boyer-Moore lança mão da heurística do bom sufixo, ilustrada na Figura 2.6. Para poder compreendê-la, consideremos a relação binária $\sim$ ("similar a") definida da seguinte maneira:

$$X \sim Y \iff X \sqsupset Y \vee Y \sqsupset X.$$

Suponha, novamente, uma tentativa do algoritmo na qual a extremidade esquerda da janela está sobre a posição i. O algoritmo procede comparando o padrão Y com o fator $X_{i..i+m-1}$ da direita para a esquerda, sendo o sufixo $U = Y_{j+1..}$ alinhado com

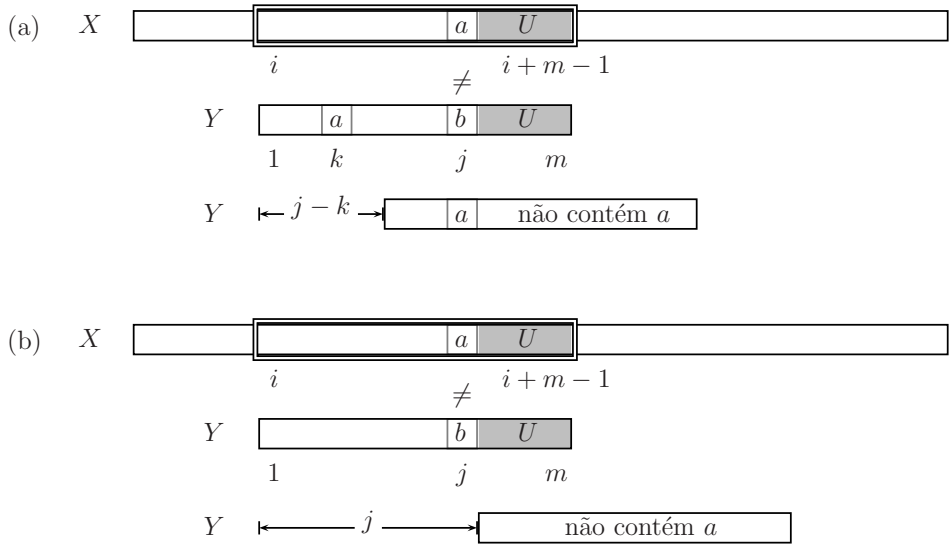


Figura 2.5. Heurística do mau caractere. (a) O padrão Y é comparado com a janela na posição i (de X) da direita para a esquerda. O sufixo $U = Y_{j+1..}$ é alinhado com o fator $X_{i+j..i+m-1}$ e a primeira colisão ocorre em $a = x_{i+j-1} \neq y_j = b$. O padrão (a janela) é deslocado para a direita de forma que a ocorrência mais à direita do *mau caractere* a em Y, y_k, seja alinhada com x_{i+j-1}. Se $k > j$ então esta heurística não é capaz de fornecer ajuda alguma. (b) Mesmo caso anterior sendo que o *mau caractere* não ocorre em Y. Nesse caso a janela pode ser deslocada para a posição $i + j$.

o fator $X_{i+j..i+m-1}$ e a primeira colisão acontecendo em $a = x_{i+j-1} \neq y_j = b$. Seja $k = \delta'(j) = \max\{r \mid 0 \leq r < m \wedge Y_{..r} \sim U = Y_{j+1..}\}$, isto é, $\delta'(j)$ corresponde ao comprimento do maior prefixo próprio de Y similar a $Y_{j+1..}$. Afirmamos que não pode existir uma ocorrência de Y em X iniciada na posição $i + l$, para $0 < l < m - k$ pois, se houvesse, teríamos $U = X_{i+j..i+m-1} = Y_{j+1..} = Y_{j+1-l..m-l} \ \therefore \ Y_{j+1..} \sqsupset Y_{..m-l}$, o que contradiz a maximalidade de k, uma vez que $l < m - k \Rightarrow k < m - l$. Portanto, a heurística do bom sufixo propõe, nesse caso, um deslocamento seguro para a janela de $m - k$ posições à direita.

Para melhor compreender como a computação de δ' pode ser realizada de maneira eficiente, faz-se necessário, antes, estabelecer alguns resultados e definições.

Lema 2.1 *Para todo* $1 \leq j < m$, $\delta'(j) \geq |brd(Y)|$.

Prova Seja $U = Y_{j+1..}$. Sabemos que $U \sqsupset Y$ e $brd(Y) \sqsupset Y$. Se $|U| \leq |brd(Y)|$, então $U \sqsupset brd(Y)$. Se $|U| > |brd(Y)|$, então $brd(Y) \sqsupset U$. Portanto, em qualquer caso, temos que $U \sim brd(Y)$. ∎

Em conseqüência do Lema 2.1, $\delta'(j)$ pode ser reescrito da seguinte forma:

$$\delta'(j) = \max\{r \mid |brd(Y)| \leq r < m \ \wedge \ Y_{..r} \sim Y_{j+1..}\}. \tag{2.1}$$

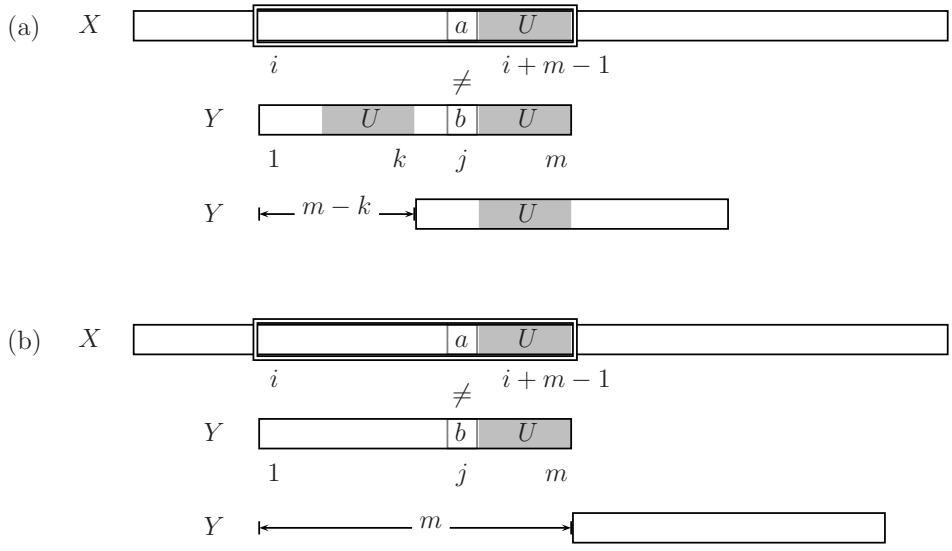


Figura 2.6. Heurística do bom sufixo. (a) O padrão Y é comparado com a janela na posição i da direita para a esquerda. O sufixo $U = Y_{j+1..}$ é alinhado com o fator $X_{i+j..i+m-1}$ e a primeira colisão ocorre em $a = x_{i+j-1} \neq x_j = b$. O padrão (a janela) é deslocado para a direita de forma que a ocorrência mais à direita do *bom sufixo* U em $Y_{..m-1}$, $Y_{k-(m-j)+1..k}$, seja alinhada com $X_{i+j..i+m-1}$. (b) Mesmo caso anterior sendo que o *bom sufixo* não ocorre em Y, nem tampouco nenhum dos seus sufixos ocorre como prefixo de Y. Nesse caso a janela pode ser deslocada para a posição $i + m$.

Lema 2.2 *Se* $|brd(Y)| < r < m$ *e* $Y_{..r} \sim Y_{j+1..}$, *para algum* $1 \leq j < m$, *então* $Y_{j+1..} \sqsupset Y_{..r}$.

Prova Se $Y_{..r} \sim Y_{j+1..}$ então $Y_{..r} \sqsupset Y_{j+1..}$ ou $Y_{..j+1} \sim Y_{r..}$. Suponha que $Y_{..r} \sqsupset Y_{j+1..}$ e $Y_{..r} \neq Y_{j+1..}$. Nesse caso $Y_{..r} \sqsupset Y$ e $r = |Y_{..r}| > |brd(Y)|$, o que contradiz a maximalidade da borda. Logo temos, necessariamente, $Y_{j+1..} \sqsupset Y_{..r}$. ∎

Em conseqüência do Lema 2.2, reescrevemos a expressão (2.1) como:

$$\delta'(j) = \max(\{|brd(Y)|\} \cup \{r \mid |brd(Y)| < r < m \ \wedge \ Y_{j+1..} \sqsupset Y_{..r}\}). \qquad (2.2)$$

Lema 2.3 *Sejam as funções definidas por* $\pi(t) = |brd(Y_{..t})|$ $(1 \leq t \leq m)$ *e* $\pi^R(t) = |brd(Y^R_{..t})|$. *Se* $k = \max\{r \mid 0 < r < m \ \wedge \ Y_{j+1..} \sqsupset Y_{..r}\}$ *então* $\pi^R(l) = m - j$, *onde* $l = (m - k) + (m - j)$.

Prova A demonstração pode ser feita, na sua quase totalidade, graficamente através da Figura 2.7. O resultado segue diretamente do fato de que $brd(Y^R_{..m-j}) = U^R$, o que pode ser facilmente verificado uma vez que, se tivéssemos uma borda maior, teríamos contrariado a maximalidade de k. ∎

Novamente, desta vez com a ajuda do Lema 2.3, reescrevemos a expressão (2.2) como:

$$\delta'(j) = \max(\{\pi(m)\} \cup \{m - l + \pi^R(l) \mid 1 \leq l \leq m \ \wedge \ j = m - \pi^R(l)\}). \qquad (2.3)$$

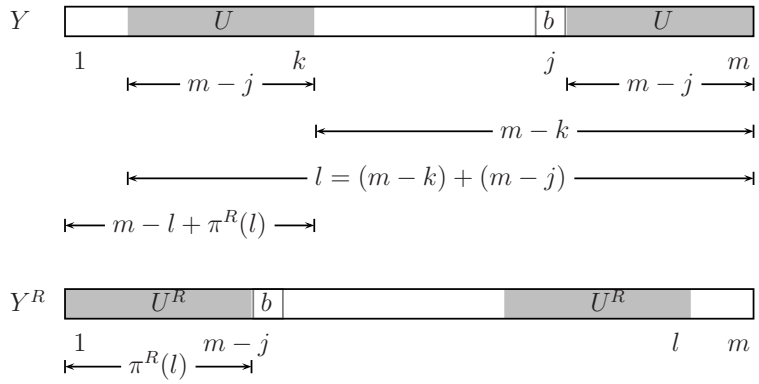


Figura 2.7. Ilustração do Lema 2.3.

Finalmente, de posse da equação (2.3), e sabendo que o deslocamento proposto pela heurística do bom sufixo para o caso em que $Y_{j+1..} = X_{i+j..i+m-1}$ e $x_{i+j-1} \neq y_j$ é dado pela expressão

$$\begin{aligned}\delta(j) &= m - \delta'(j) \\ &= m - \max(\{\pi(m)\} \cup \{m - l + \pi^R(l) \mid 1 \leq l \leq m \ \wedge \ j = m - \pi^R(l)\}) \\ &= \min(\{m - \pi(m)\} \cup \{l - \pi^R(l) \mid 1 \leq l \leq m \ \wedge \ j = m - \pi^R(l)\}),\end{aligned}$$

estamos em condição de apresentar a função *bom_sufixo* (Algoritmo 2.5) que computa previamente esses deslocamentos para todos os valores de j. O Algoritmo 2.5 faz uso da função *borda* que pode ser obtida através de uma ligeira modificação do Algoritmo 2.2, substituindo-se as linhas *10–14* por "$v_{i+j-1} \leftarrow j$". Além disso, temos uma chamada à função *reverte* que reverte uma cadeia em tempo linear no tamanho desta.

O Algoritmo Boyer-Moore (Algoritmo 2.6) corresponde a uma modificação do Algoritmo Força-bruta de forma a contemplar a comparação do padrão com a janela no sentido direita-esquerda e a utilização conjunta das duas heurísticas descritas.

A função *última_ocorrência* demanda, claramente, tempo $O(m + |\Sigma|)$. A função *bom_sufixo*, por sua vez, requer tempo $O(m)$, uma vez que as chamadas às funções *borda* e *reverte* tomam tempo $O(m)$ e os dois laços seguintes são percorridos exatamente $m+1$ e m vezes respectivamente. No pior caso, o Algoritmo Boyer-Moore realiza $n - m + 1$ tentativas, cada uma delas exigindo m comparações de símbolos. Portanto o pior caso do algoritmo demanda tempo $O((n - m + 1)m + |\Sigma|)$. Na prática, entretanto, esse pior caso corresponde a uma situação bastante incomum e de pouco interesse. Um estudo probabilístico do Algoritmo Boyer-Moore [BYGR90] revela, para o caso médio, um comportamento sublinear. Quando levamos em conta o número de comparações necessárias para encontrar apenas a primeira ocorrência do padrão, obtemos um limite ótimo de $3n+\Omega(n/m)$ comparações [Col94]. Embora essa demonstração seja não-trivial, uma prova um tanto mais simples do limite superior de $4n$ comparações pode ser encontrada em [Col94, CR94]. O algoritmo requer ainda $\Theta(m + |\Sigma|)$ espaço adicional correspondente aos vetores auxiliares S e C.

```
1  Função bom_sufixo (Y = y_1 ··· y_m)
2  início
3     % Π = (π_1, ..., π_m) e Π^R = (π_1^R, ..., π_m^R) são vetores t.q. π_j = π(j) e π_j^R = π^R(j)
4     % Δ = (δ_0, ..., δ_m) é um vetor t.q. δ_j = δ(j)
5     Π ← borda(Y)
6     Y^R ← reverte(Y)
7     Π^R ← borda(Y^R)
8     para j ← 0 até m faça
9        δ_j ← m − π_m
10    fim-faça
11    para l ← 0 até m faça
12       j ← m − π^R(l)
13       se δ_j > l − π^R(l) então
14          δ_j ← l − π^R(l)
15       fim-se
16    fim-faça
17    devolva Δ
18 fim
```

Algoritmo 2.5. Função *bom_sufixo*.

```
1  Algoritmo Boyer-Moore (X = x_1 ··· x_n, Y = y_1 ··· y_m)
2  início
3     C ← última_ocorrência(Y)
4     S ← bom_sufixo(Y)
5     i ← 1
6     enquanto i ≤ n − m + 1 faça
7        j ← m
8        enquanto x_{i+j−1} = y_j e j > 0 faça
9           j ← j − 1
10       fim-faça
11       se j = 0 então
12          𝕊 ← 𝕊 ∪ {i}
13          i ← i + S_0
14       senão
15          i ← max{S_j, j − C_{x_{i+j−1}}}
16       fim-se
17    fim-faça
18 fim
```

Algoritmo 2.6. Algoritmo Boyer-Moore.

2.1.4 O Algoritmo Karp-Rabin

O Algoritmo devido a Karp e Rabin [KR87] baseia-se em um princípio distinto dos dois algoritmos clássicos apresentados até o momento. Aqueles são algoritmos exatos, determinísticos que se baseiam na comparação de símbolos, enquanto este é um algoritmo probabilístico que faz forte uso de operações aritméticas, tal como veremos a seguir.

Conforme temos feito até então, estamos interessados em procurar ocorrências de um padrão $Y = y_1 \cdots y_m$ em uma cadeia $X = x_1 \cdots x_n$, ambos tomados sobre um alfabeto Σ de tamanho s (*i.e.* $|\Sigma| = s$). No Algoritmo Karp-Rabin, cada caractere do alfabeto é tomado como sendo um numeral no sistema de numeração da base s e cada cadeia de comprimento k é considerada como um número de k dígitos nessa mesma base. Por exemplo, tomando o alfabeto de nucleotídeos $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, temos $s = 4$ e podemos considerar $\mathtt{A} = 0$, $\mathtt{C} = 1$, $\mathtt{G} = 2$ e $\mathtt{T} = 3$. Nesse caso a cadeia $W = \mathtt{GTACT}$ corresponde ao número $W' = 23013_4 = 711_{10}$. Procedemos, então, de maneira bastante similar ao que fizemos no Algoritmo Força-bruta, comparando cada fator $X_{i..i+m-1}$ (para $i = 1, \ldots, n - m + 1$) no interior da janela deslizante, com o padrão Y. No Algoritmo Karp-Rabin (Algoritmo 2.7), entretanto, a comparação do padrão com o conteúdo da janela é feita não mais através da comparação dos seus caracteres constituintes e sim através da comparação direta dos valores numéricos correspondentes.

```
 1 Algoritmo Karp-Rabin (X = x_1 ··· x_n, Y = y_1 ··· y_m)
 2 início
 3   p ← s^{m-1} mod r
 4   Y', X' ← 0
 5   para i ← 1 até m faça
 6     Y' ← (sY' + y'_i) mod r
 7     X' ← (sX' + x'_i) mod r
 8   fim-faça
 9   para i ← 1 até n − m + 1 faça
10     se Y' = X' então
11       se Y = X_{i..i+m-1} então
12         𝕊 ← 𝕊 ∪ {i}
13       fim-se
14     fim-se
15     se i < n − m + 1 então
16       X' ← (s(X' − px'_i) + x'_{i+m}) mod r
17     fim-se
18   fim-faça
19 fim
```

Algoritmo 2.7. Algoritmo Karp-Rabin.

À luz do exposto até o momento, algumas explicações fazem-se necessárias acerca da eficiência e eficácia do Algoritmo Karp-Rabin. Primeiramente, o valor numérico Y' correspondente ao padrão Y pode ser computado em tempo $O(m)$ (linhas *5–8*) na forma

$$Y' = s(s(s(\cdots s(y'_2 + sy'_1) \cdots) + y'_{m-2}) + y'_{m-1}) + y'_m.$$

O mesmo tipo de computação vale para o valor numérico do primeiro fator de tamanho m de X, ou seja, o prefixo $X_{..m}$. O valor numérico dos fatores subseqüentes podem ser computados cada um em função do seu anterior, em tempo constante, através da seguinte relação (linha *16*):

$$X'_{i+1..i+m} = (s(X'_{i..i+m-1} - x'_i s^{m-1}) + x'_{i+m}),$$

bastando, para tanto, computar previamente (também em tempo linear) o valor da constante $p = s^{m-1}$.

Para evitar problemas com números muito grandes, as operações são realizadas em aritmética módulo r, em geral, para um valor primo de r escolhido convenientemente de forma que sr caiba em uma palavra de máquina. O efeito colateral dessa solução é que podemos ter situações em que $Y' \equiv X'_{i..i+m-1} \mod r$ sem que, necessariamente, $Y = X_{i..i+m-1}$. Todavia, se $Y' \not\equiv X'_{i..i+m-1} \mod r$ então, necessariamente, $Y \neq X_{i..i+m-1}$. Portanto, toda vez que uma "igualdade módulo r" é detectada (linha *10*) entre o valor numérico do padrão e do conteúdo da janela, uma comparação dessas cadeias, caractere a caractere, ainda é necessária (linhas *11–13*), sob pena de que seja reportada uma *ocorrência espúria*.

O Algoritmo Karp-Rabin demanda tempo $O((n-m+1)m)$ no pior caso (pouco interessante e pouquíssimo provável), proporcional ao número de comparações de símbolos (*e.g.*, $X = a^n$ e $Y = a^m$). Se supormos uma distribuição de probabilidade uniforme sobre Σ, uma ocorrência espúria pode acontecer com probabilidade $1/r$. Com efeito, a probabilidade que um número de m dígitos pertença a uma das classes de equivalência da relação $\equiv \mod r$, em particular, à classe de Y', é obtida dividindo-se a quantidade de números dessa classe, s^m/r, pela quantidade total de números de m dígitos, s^m. Logo, o número esperado de ocorrências espúrias é de n/r e o tempo esperado para Algoritmo 2.7 é $O(n)+O(m(v+n/r))$, onde $v = |\mathbb{S}|$ denota a quantidade de ocorrências efetivas de Y em X. Se a quantidade esperada de ocorrências é pequena, digamos $O(1)$, e $r \geq m$, então o tempo esperado para o algoritmo é linear no tamanho da entrada.

2.1.5 O Algoritmo Shift-Or

O Algoritmo *Shift-Or*, apresentado por Baeza-Yates e Gonnet [BYG92], faz forte uso do paralelismo intrínseco das operações binárias (*bit-parallelism*) restritas a uma palavra de máquina no modelo RAM [AHU74] e explora a finitude do alfabeto Σ com o objetivo de acelerar a localização das ocorrências exatas, no texto $X = x_1 \cdots x_n$, do padrão $Y = y_1 \cdots y_m$, idealmente com $|Y| \leq w$, onde w representa o comprimento da palavra de máquina.

Para cada $1 \leq i \leq n$ definimos o número S^i dado em sua forma binária por

$$S^i = s^i_m \cdots s^i_1, \quad \text{tal que } s^i_j = \Phi(Y_{..j}, X_{i-j+1..i}), \tag{2.4}$$

onde $\Phi : \Sigma^* \times \Sigma^* \to \{0,1\}$ é a função característica definida por

$$\Phi(U,V) = \begin{cases} 0, & \text{se } U = V \\ 1, & \text{do contrário.} \end{cases}$$

Equivalentemente, temos $S^i = \sum_{j=1}^{m} \Phi(Y_{..j}, X_{i-j+1..i}) \cdot 2^{i-1}$. Da definição (2.4), temos que Y ocorre exatamente em X a partir da posição $i - m + 1$, se $s_m^i = 0$.

Definimos também, para cada caractere $a \in \Sigma$, o número binário T^a dado por

$$T^a = t_m^a \cdots t_1^a, \quad \text{tal que } t_j^a = \varphi(y_j, a), \tag{2.5}$$

onde $\varphi : \Sigma \times \Sigma \to \{0, 1\}$ é a função característica definida por

$$\varphi(a, b) = \begin{cases} 0, & \text{se } a = b \\ 1, & \text{do contrário,} \end{cases} \tag{2.6}$$

ou seja, T^a funciona como uma máscara binária das ocorrências de a em Y.

A Figura 2.8 ilustra os valores de S^i e T^{x_i} para as cadeias $X =$ `ABDABABABC` e $Y =$ `ABABC` tomadas sobre $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$.

X	A	B	D	A	B	A	B	A	B	C
T^{x_i}	11010	10101	11111	11010	10101	11010	10101	11010	10101	01111
S^i	11110	11101	11111	11110	11101	11010	10101	11010	10101	01111

Figura 2.8. Tentativas do Algoritmo *Shift-Or* para $X =$ `ABDABABABC`, $Y =$ `ABABC` e $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$. Nesse caso, uma ocorrência é detectada na posição 6, assinalada pelo valor 0 do bit mais significativo de S^{10}.

A hipótese de indução do Algoritmo *Shift-Or* corresponde ao seguinte resultado, que dá conta de como os casamentos parciais dos prefixos de Y com fatores de X terminados na posição i podem ser estendidos em casamentos de prefixos de Y com fatores de X terminados na posição subseqüente $i + 1$.

Lema 2.4 *Para todo* $1 \leq i < n$, $S^{i+1} = S^i \ll 1 \mid T^{x_{i+1}}$, *onde* $\ll$ *denota operador binário de deslocamento de bits à esquerda (*shift left*)*[*] *e* $\mid$ *denota o operador booleano* ou-binário *(*bitwise or*).*

Prova Seja $S^{i+1} = s_m^{i+1} \cdots s_1^{i+1}$. Para todo $1 < j \leq m$, temos:

$$\begin{aligned} s_j^{i+1} &= \Phi(Y_{..j}, X_{i+1-j+1..i+1}) \\ &= \Phi(Y_{..j-1}, X_{i-(j-1)+1..i}) \mid \varphi(y_j, x_{i+1}) \\ &= s_{j-1}^i \mid t_j^{x_{i+1}}. \end{aligned}$$

Se $j = 1$, então $s_1^{i+1} = \varphi(y_1, x_{i+1}) = 0 \mid \varphi(y_1, x_{i+1})$. ∎

O Algoritmo *Shift-Or* mantém a janela deslocando-se para a direita um caractere por vez, computando, a cada iteração, com base no Lema 2.4, o valor de S^i, onde i corresponde à extremidade direita da janela. Se $s_m^i = 0$, ou seja, $S^i < 2^{m-1}$, então uma ocorrência é reportada na posição $i - m + 1$.

[*] Se $S = s_m \cdots s_1$, então $S \ll k = s_{m-k} \cdots s_1 \underbrace{0 \cdots 0}_{k}$.

```
1  Algoritmo Shift-Or (X = x_1 ··· x_n, Y = y_1 ··· y_m)
2  início
3     t ← 2^{m-1}
4     T ← máscara_ocorrência(Y)
5     S ← 1···1 (m)
6     para i ← 1 até n faça
7        S ← S ≪ 1 | T^{x_i}
8        se S < t então
9           𝕊 ← 𝕊 ∪ {i − m + 1}
10       fim-se
11    fim-faça
12 fim
```

Algoritmo 2.8. Algoritmo *Shift-Or*.

Como se pode observar, o Algoritmo 2.8 depende do pré-processamento do padrão Y para a computação da tabela de máscaras de ocorrências T, o que é feito pela função *máscara_ocorrência* (Algoritmo 2.9). Essa função de pré-processamento pode ser implementada em tempo linear $O(|\Sigma| + m)$ com o auxílio de dois operadores binários: o "&", que representa o *e-binário* (*bitwise and*), e o "$\sim$", que representa a negação binária (inversão dos bits).

```
1  Função máscara_ocorrência (Y = y_1 ··· y_m)
2  início
3     % Consideramos uma enumeração do alfabeto Σ = {a_1, ..., a_s}
          e identificamos cada b ∈ Σ com a sua posição, i.e. b ≡ i ⇔ b = a_i.
4     % T = (T_1, ..., T_s) é um vetor t.q. T_j = T^{b=a_j}
5     para j ← 1 até s faça
6        T_j ← 1···1 (m)
7     fim-faça
8     k ← 0···01 (m)
9     para j ← 1 até m faça
10       T_{x_j} ← T_{x_j} & ∼k
11       k ← k ≪ 1
12    fim-faça
13    devolva T
14 fim
```

Algoritmo 2.9. Função *máscara_ocorrência*.

A fase de busca do Algoritmo Shift-Or requer tempo $O(n)$ (pior caso e caso médio) se $m \leq w$, isso porque, nestas condições, podemos supor que a operação da linha 7 do Algoritmo 2.8 é realizada em tempo constante, conforme comentamos anteriormente. Em geral, se $m > w$, obtemos tempo $O(\lceil \frac{m}{w} \rceil n)$, onde $\lceil \frac{m}{w} \rceil$ corresponde ao tempo necessário

para computar um deslocamento binário ou operação equivalente em um número de m bits numa máquina com palavras de tamanho w (nas arquiteturas atuais, geralmente $w = 32$ ou $w = 64$). A fase de pré-processamento (Algoritmo 2.9) demanda tempo $O(m+|\Sigma|)$ também para $m \leq w$. Similarmente, se $m > w$, obtemos tempo $O(\lceil \frac{m}{w} \rceil (m + |\Sigma|))$. O espaço extra requerido pelo Algoritmo *Shift-Or* limita-se, essencialmente, com o espaço ocupado pela tabela T, ou seja, $m \cdot |\Sigma|$ bits.

2.2 CASAMENTO APROXIMADO DE PADRÕES

Em boa parte das aplicações práticas, estamos interessados em identificar as ocorrências não necessariamentes exatas do padrão Y no texto X. Esse tipo de problema surge naturalmente em contextos como, por exemplo, o da Biologia Computacional, uma vez que as seqüências moleculares biológicas estão sujeitas a mutações, para não falar da possibilidade de erros durante o processo de sequenciamento (leitura e montagem de fragmentos).

Nesta seção, trataremos do problema do *casamento aproximado de padrões*, ou seja, o problema de casamento de padrões no qual uma determinada quantidade de erros é admissível.

Antes de enunciar o problema em questão, cumpre introduzir a noção de distância entre cadeias. Em geral, uma função de distância entre cadeias ou *métrica* em Σ^* é uma aplicação $d : \Sigma^* \times \Sigma^* \to \mathbb{Z}_+$ que obedece às seguintes propriedades para quaisquer $X, Y, Z \in \Sigma^*$:

i) $d(X, Y) \geq 0$ (definida positiva)
ii) $d(X, Y) = 0 \iff X = Y$
iii) $d(X, Y) = d(Y, X)$ (simétrica)
iv) $d(X, Y) + d(Y, Z) \geq d(X, Z)$ (desigualdade triangular)

A função de distância confere a Σ^* a estrutura de um espaço métrico.

Existem diversas funções de distância estudadas em processamento de cadeias. Neste trabalho, estaremos interessados sobretudo na chamada *Distância de Levenshtein*, d_L, também conhecida como *distância de edição* e, ocasionalmente, na chamada *Distância de Hamming*, d_H.

Definição 2.2 (Distância de Levenshtein) *A distância de Levenshtein entre duas cadeias $X = x_1 \cdots x_n$ e $Y = y_1 \cdots y_m$ pode ser definida através da seguinte relação de recorrência:*

$$d_L(X, Y) = \min \left\{ \begin{array}{c} d_L(X_{..n-1}, Y_{..m-1}) + \varphi(x_m, y_n), \\ d_L(X_{..n-1}, Y) + 1, \\ d_L(X, Y_{..m-1}) + 1 \end{array} \right\}, \tag{2.7}$$

onde φ é a função característica definida em (2.6) e $d_L(\varepsilon, \varepsilon) = 0$ por definição.

Definição 2.3 (Distância de Hamming) *A distância de Hamming entre duas cadeias $X = x_1 \cdots x_n$ e $Y = y_1 \cdots y_n$ pode ser definida através da seguinte relação de recorrência:*

$$d_H(X, Y) = d_H(X_{..n-1}, Y_{..n-1}) + \varphi(x_n, y_n),$$

onde φ é definida como em (2.6) e $d_H(\varepsilon, \varepsilon) = 0$.

A distância de Levenshtein corresponde à quantidade de operações de edição (daí o nome)—substituições, inserções e remoções* —necessárias à conversão de uma cadeia na outra, equanto a distância de Hamming considera apenas as substituições de símbolos. Por exemplo, $d_L(\texttt{ABCABC}, \texttt{BCABCA}) = 2$ (remoção do primeiro caractere mais a inserção de um `A` ao final), enquanto $d_L(\texttt{ABCABC}, \texttt{BCABCA}) = 6$.

De posse do conceito de distância entre cadeias, podemos formular adequadamente o problema do casamento aproximado de padrões.

Problema do casamento aproximado de padrões Dadas duas cadeias $X = x_1 \cdots x_n$ e $Y = y_1 \cdots y_m$ (em geral, $m < n$) e um inteiro $0 \leq r \leq m$, determinar todas as ocorrências de Y em X a uma distância menor ou igual a r.

Colocando de maneira um pouco mais formal, o problema do casamento aproximado de padrões corresponde a resolver no universo $\mathbb{U} = \{1, \ldots, n\}$, a seguinte inequação em j:

$$d(Y, X_{j-k..j}) \leq r, \text{ para algum } 0 \leq k < j.$$

Quando $d = d_H$, temos o que se convencionou denominar na literatura de "casamento de padrões com r colisões", ao passo que, quando $d = d_L$, temos o que comumente se chama de "casamento de padrões com r diferenças". No restante do capítulo, estaremos considerando o segundo caso, ao qual nos referiremos simplesmente como "casamento aproximado de padrões".

A exemplo do seu correspondente exato, o problema do casamento aproximado de padrões tem sido extensivamente estudado nas últimas décadas. Navarro [Nav98, Nav01] propõe uma taxonomia para os algoritmos *on-line*† exatos e combinatórios destinados à resolução do problema. Em seu estudo, baseado na análise de mais de quarenta algoritmos, Navarro postula que esses algoritmos sejam agrupados em quatro grandes classes:

i) Algoritmos baseados na matriz de programação dinâmica. Os algoritmos clássicos destinados à computação da distância de Levenshtein entre duas cadeias baseiam-se em uma técnica que prevê a construção da chamada *matriz de programação dinâmica*. Esta técnica foi, posteriormente, adaptada para o problema da busca aproximada de padrões. Os algoritmos mais eficientes deste grupo possuem complexidade $O(rn)$ no pior caso e $O(rn/\sqrt{|\Sigma|})$ no caso médio. Embora não sejam os algoritmos dessa categoria, necessariamente, os mais eficientes, eles possuem grande importância devido à flexibilidade do modelo empregado além, é claro, do valor teórico e histórico;

ii) Algoritmos baseados na teoria dos autômatos finitos. A exemplo da classe anterior, os resultados nessa área são razoavelmente antigos. Aqui, a idéia central baseia-se na modelagem do processo de busca através de um autômato finito (geralmente não-determinístico). Os algoritmos mais eficientes desta categoria apre-

*A distância definida pela expressão (2.7) é chamada de distância de Levenshtein *simples* uma vez que confere custo 1 a todas as operações de edição. Em geral, poderíamos considerar uma métrica genérica na qual as operações possuem custos distintos.

†Que não se baseiam no emprego de estruturas de índice.

sentam custo $O(n)$ no pior caso mas implicam em uma dependência exponencial entre m e r que limita a sua utilização em problemas concretos;

iii) Algoritmos baseados em técnicas de paralelismo binário (*bit-parallelism*). Esta técnica, introduzida por Ricardo Baeza-Yates em sua tese de doutoramento [*Efficient Text Searching*, Univ. of Waterloo, 1989], consiste na utilização do paralelismo intrínseco das operações binárias, quando restritas a uma palavra de máquina, de forma a reduzir o custo de diversas operações em um fator de até w vezes, onde w representa o comprimento da palavra de máquina. Este princípio já foi apresentado na Seção 2.1.5;

iv) Algoritmos baseados em filtragem. O processo de filtragem fundamenta-se na idéia de descartar, a baixo custo, regiões do texto idealmente "grandes" nas quais podemos assegurar que o padrão não ocorre. Em determinadas condições, pode ser, de fato, menos custoso determinar apenas se o padrão *não* ocorre no texto em uma dada posição do que verificar a sua ocorrência. Para as regiões que não puderem ser descartadas, um outro algoritmo capaz de determinar, de maneira definitiva, a ocorrência do padrão é solicitado. Portanto, os algoritmos que empregam filtros só são capazes de apresentar ganho de eficiência no caso médio, sendo $O(n(r + \log_{|\Sigma|} m)/m)$ o limite ótimo [CM94].

Nas seções vindouras, apresentamos quatro algoritmos, representantes de cada uma das classes acima. Antes, porém, trataremos de forma um pouco mais detalhada, ainda que breve, da questão da computação da distância de edição entre duas cadeias.

2.2.1 Computação da Distância de Edição Entre Cadeias

Embora a definição da distância de edição entre cadeias seja, até certo ponto, intuitiva, a computação eficiente desse valor não é óbvia. O método descrito nesta seção foi descoberto e redescoberto independentemente no passado em diversas áreas, conforme apontam Aho [Aho90, Seção 6.3] e Navarro [Nav01, Seção 5.1].

Dadas as cadeias $X = x_1 \cdots x_n$ e $Y = y_1 \cdots y_m$, definimos*

$$d_{i,j} = d_L(Y_{..i}, X_{..j}).$$

Com base nesta definição, consideramos uma matriz $D(X, Y)$ com $(m + 1)$ linhas e $(n + 1)$ colunas, na qual o elemento da i-ésima linha e j-ésima coluna ($i \in \{0, \ldots, m\}$ e $j \in \{0, \ldots, n\}$) corresponde exatamente a $d_{i,j}$. Desta maneira, o elemento da última linha e última coluna corresponde a $d_{m,n} = d_L(Y_{..m}, X_{..n}) = d_L(Y, X)$. Repare que, com base na Definição 2.2, o valor de $d_{i,j}$ está completamente determinado pelos valores dos elementos adjacentes da linha e da coluna imediatamente anteriores. Portanto, a matriz

*Até então, vínhamos utilizando o índice i para nos referirmos às posições no texto X e j para posições no padrão Y. Essa, de fato, parece ser a praxe na literatura do casamento exato de padrões. No contexto do casamento aproximado de padrões, entretanto, o costume parece ser justamente o contrário, conforme pudemos constatar nas principais referências desta seção [Nav01, Sel80, Ukk85, WM92b, CR94], dentre outras. Optamos então por sacrificar, ligeiramente, a consistência da notação em nome da prática consagrada.

D pode ser computada sequencialmente coluna a coluna (ou linha a linha), conforme disposto no Algoritmo 2.10. A Figura 2.9 ilustra o procedimento para $X = \texttt{ABADAC}$ e $Y = \texttt{CADA}$.

1 **Função** *distância_edição* ($X = x_1 \cdots x_n$, $Y = y_1 \cdots y_m$)
2 **início**
3 % $D = \begin{pmatrix} d_{0,0} & \cdots & d_{0,n} \\ \vdots & \ddots & \vdots \\ d_{m,0} & \cdots & d_{m,n} \end{pmatrix}$ matriz $m+1 \times n+1$
4 **para** $i \leftarrow 0$ **até** m **faça**
5 $d_{i,0} \leftarrow i$
6 **fim-faça**
7 **para** $j \leftarrow 0$ **até** n **faça**
8 $d_{0,j} \leftarrow j$
9 **fim-faça**
10 **para** $j \leftarrow 1$ **até** n **faça**
11 **para** $i \leftarrow 1$ **até** m **faça**
12 $d_{i,j} = \min\{d_{i-1,j-1} + \varphi(x_j, y_i),\ d_{i,j-1} + 1,\ d_{i-1,j} + 1\}$
13 **fim-faça**
14 **fim-faça**
15 **devolva** $d_{m,n}$
16 **fim**

Algoritmo 2.10. Função *distância_edição*.

Observe que o Algoritmo 2.10 baseia-se na técnica de *programação dinâmica*, uma vez que, de (2.7), temos claramente que a solução do problema para duas cadeias pode ser resolvido através da solução deste mesmo problema para seus prefixos, caracterizando um processo indutivo. Por essa razão, a matriz D é comumente denominada *matriz de programação dinâmica*.

	ε	A	B	A	D	A	C
ε	**0**	**1**	2	3	4	5	6
C	1	1	**2**	3	4	5	5
A	2	1	2	**2**	3	4	5
D	3	2	2	3	**2**	3	4
A	4	3	3	2	3	**2**	**3**

Figura 2.9. Matriz de programação dinâmica $D(X, Y)$ para $X = \texttt{ABADAC}$ e $Y = \texttt{CADA}$.

A(s) sequência(s) exata(s) de operações necessárias à conversão da cadeia X na cadeia Y pode(m) ser facilmente recobrada(s) através da matriz $D(X, Y)$. Com efeito, pela definição (2.7), temos três situações a considerar:

i) $d_{i,j} = d_{i-1,j-1} + \varphi(x_j, y_i)$. Neste caso, distância entre $Y_{..i}$ e $X_{..j}$ é determinada pela distância entre $Y_{..i-1}$ e $X_{..j-1}$ mais o custo da comparação entre os últimos

caracteres, x_j e y_i. Na matriz, esta situação é caracterizada quando o valor da entrada é obtido a partir da entrada adjacente na diagonal à esquerda ascendente;

ii) $d_{i,j} = d_{i,j-1} + 1$. Este caso ocorre quando a distância entre $Y_{..i}$ e $X_{..j}$ é determinada pela distância entre $Y_{..i}$ e $X_{..j-1}$ mais o custo da remoção do caractere x_j. Na matriz, esta situação é caracterizada quando o valor da entrada é obtido a partir da entrada adjacente à esquerda;

iii) $d_{i,j} = d_{i-1,j} + 1$. Aqui, a distância entre $Y_{..i}$ e $X_{..j}$ é determinada pela distância entre $Y_{..i-1}$ e $X_{..j}$ mais o custo da inserção de um caractere neutro ao final de X. Na matriz, esta situação é caracterizada quando o valor da entrada é obtido a partir da entrada adjacente acima.

Com base nas alternativas acima, podemos definir sobre D o chamado *grafo de dependências*, no qual os nós correspondem às entradas $d_{i,j}$ da matriz e as arestas (direcionadas) são representadas por pares $(d_{i',j'}, d_{i,j})$ se o valor de $d_{i,j}$ pode ser determinado por $d_{i',j'}$ conforme disposto nos três itens acima. Portanto, o grafo de dependências de D possui arestas diagonais (item i), horizontais (item ii) e verticais (item iii) que conectam células adjacentes de D. Percorrendo o(s) caminho(s) de $d_{0,0}$ a $d_{m,n}$ no grafo de dependências, podemos reconstituir a(s) seqüência(s) de edições consideradas no cálculo da distância entre X e Y, obtendo assim, um ou mais *alinhamentos* ótimos entre essas cadeias. Na Figura 2.9, um desses caminhos está indicado pelos elementos em destaque, que corresponde ao alinhamento

```
A  B  A  D  A  C
|  |           |
-  C  A  D  A  -
```

Repare que as inserções e remoções em X e Y são complementares, ou seja, uma remoção em X é equivalente a uma inserção em Y e reciprocamente. É comum denotar o caractere inserido por "-".

Quanto à complexidade, o Algoritmo 2.10 demanda tempo $O(mn)$, conforme pode ser trivialmente verificado. Menos óbvio é o fato de que o algoritmo requer apenas $O(m)$ espaço, uma vez que a matriz D não necessita ser armazenada na íntegra, sendo suficiente apenas o conhecimento da coluna atual e da imediatamente anterior.

2.2.2 O Algoritmo Sellers

O algoritmo proposto por Sellers [Sel80] prevê uma ligeira adaptação da matriz $D(X, Y)$ e do Algoritmo 2.10 de forma a comportar a busca aproximada do padrão Y em X com, no máximo, r diferenças.

A idéia subjacente ao Algoritmo 2.11 é, de fato, bastante simples. Primeiramente, cada elemento na última linha da matriz D corresponde ao custo do alinhamento de Y com algum prefixo de X. Se um desses valores é menor ou igual a r, então o prefixo correspondente representa uma ocorrência aproximada de Y em X. No caso geral, entretanto, estamos interessados em ocorrências aproximadas representadas por qualquer

tipo de fator de X, e não apenas prefixos. Portanto, é necessário "permitir" que as ocorrências (aproximadas) de Y sejam consideradas a partir de qualquer posição de X sem qualquer penalidade, *i.e.* o custo do alinhamento do prefixo ε de Y com qualquer prefixo de X deve ser nulo. Na matriz de programação dinâmica, essa premissa equivale a tornar todos os elementos da primeira linha iguais a 0. A Figura 2.10 ilustra a situação para $X = \texttt{ABADAC}$ e $Y = \texttt{CADA}$.

1 **Algoritmo** *Sellers* ($X = x_1 \cdots x_n$, $Y = y_1 \cdots y_m$, r)
2 **início**
3 % $S' = \begin{pmatrix} s'_0 \\ \vdots \\ s'_m \end{pmatrix}$, $S = \begin{pmatrix} s_0 \\ \vdots \\ s_m \end{pmatrix}$ vetores-coluna da matriz de prog. dinâmica
4 **para** $i \leftarrow 0$ **até** m **faça**
5 $s'_i \leftarrow i$
6 **fim-faça**
7 **para** $j \leftarrow 1$ **até** n **faça**
8 $s_0 \leftarrow 0$
9 **para** $i \leftarrow 1$ **até** m **faça**
10 $s_i = \min\{s'_{i-1} + \varphi(x_j, y_i),\ s'_i + 1,\ s_{i-1} + 1\}$
11 $s'_{i-1} \leftarrow s_{i-1}$
12 **fim-faça**
13 $s'_m \leftarrow s_m$
14 **se** $s_m \leq r$ **então**
15 $\mathbb{S} \leftarrow \mathbb{S} \cup \{j\}$
16 **fim-se**
17 **fim-faça**
18 **fim**

Algoritmo 2.11. Algoritmo de Sellers.

	ε	A	B	A	D	A	C
ε	0	0	0	0	0	0	0
C	1	1	1	1	1	1	0
A	2	1	2	1	2	1	1
D	3	2	2	2	1	2	2
A	4	3	3	**2**	**2**	**1**	**2**

Figura 2.10. Matriz de programação dinâmica do Algoritmo de Sellers para $X = \texttt{ABADAC}$, $Y = \texttt{CADA}$ e $r = 2$. Nesse caso, temos $\mathbb{S} = \{3, 4, 5, 6\}$, sinalizadas na matriz pelos elementos em negrito. Mais precisamente, as ocorrências aproximadas de Y em X correspondem aos fatores $X_{1..3} = \texttt{ABA}$, $X_{2..4} = \texttt{BAD}$, $X_{2..5} = \texttt{BADA}$ e $X_{2..6} = \texttt{BADAC}$.

A exemplo do ocorrido na Função *distância_edição*, a matriz é computada coluna por coluna no Algoritmo 2.11. Por economia de espaço, apenas a coluna atual (S) e

a coluna imediatamente anterior (S') são representadas. O Algoritmo de Sellers possui complexidade de tempo $O(mn)$ e o seu custo em termos de espaço é $O(m)$.

2.2.3 O Algoritmo Ukkonen

A idéia da utilização de autômatos finitos para a modelagem do problema do casamento de padrões não é recente, tendo sido introduzida por Knuth *et al.* [KMP77]. Naquele clássico artigo, foi apresentado um AFD como o da Figura 2.11(a), correspondente ao Algoritmo Knuth-Morris-Pratt*, cuja implementação óbvia requer $O(|\Sigma|m)$ espaço. O processo de busca pode então ser reproduzido em tempo $O(n)$, percorrendo-se esse autômato, tomando o texto X como entrada.

No caso do casamento aproximado de padrões, o processo de busca pode ser modelado naturalmente através de um AFN como o da Figura 2.11(b), no qual cada linha corresponde ao número de erros encontrado e cada coluna representa um casamento de um prefixo do padrão Y [BY96].

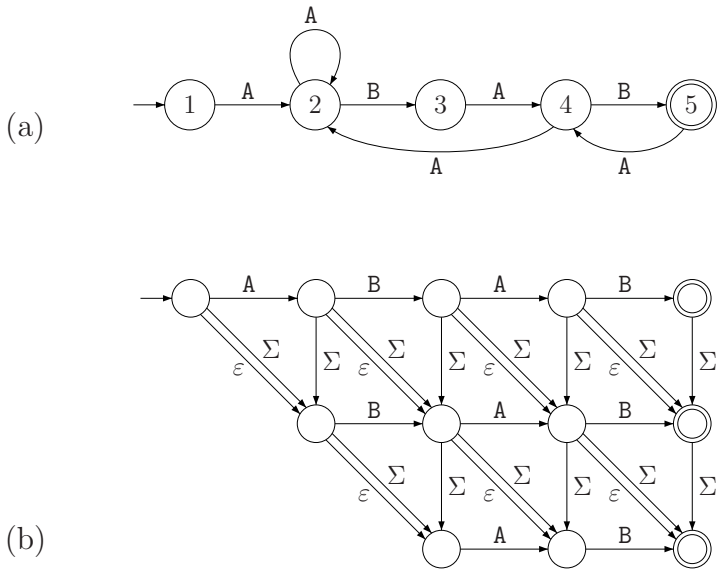


Figura 2.11. Casamento de padrões via autômatos finitos. (a) Busca exata do padrão $Y =$ `ABAB` modelada por um AFD. As demais transições que conduzem ao estado inicial foram omitidas. (b) Busca aproximada do padrão $Y =$ `ABAB` com no máximo $r = 2$ erros modelada por um AFN com ε-transições.

Ukkonen [Ukk85] propôs a representação do problema do casamento aproximado de padrões através de um autômato não explicitamente idêntico ao da Figura 2.11(b). O Autômato de Ukkonen é o AFD dado por $A_Y = (Q, \Sigma, \delta, q_0, F)$ onde:

- O conjunto de estados Q corresponde ao conjunto de todos os valores possíveis de todos os $m + 1$ vetores-coluna da matriz de programação dinâmica $D(X, Y)$ para qualquer possível valor de X;
- O alfabeto Σ é o mesmo alfabeto do problema;

*Em [KMP77] também foi proposta uma modelagem do Algoritmo Boyer-Moore por um AFD.

- A função de transição δ é tal que $\delta(S' = (s'_0, \ldots, s'_m), a) = S = (s_0, \ldots, s_m)$ se, e somente se, $s_0 = s'_0 = 0$ e $s_i = \min\{s'_{i-1} + \varphi(Y_i, a),\ s'_i + 1,\ s_{i-1} + 1\}, \forall 1 \leq i \leq m$. Equivalentemente, existe uma a-transição de S' para S se, e somente se, existe alguma configuração possível para a matriz $D(X, Y)$ na qual S' e S são colunas consecutivas e o caractere de X que determina S é igual a a;
- O estado inicial $q_0 = (0, \ldots, m)$ corresponde à primeira coluna de D;
- O conjunto de estados finais é dado por $F = \cup_{S \in Q}\{S = (s_0, \ldots, s_m) \mid s_m \leq r\}$.

Deve estar claro que percorrer o Autômato de Ukkonen respondendo aos caracteres do texto $X = x_1 \cdots x_n$, reportando uma ocorrência sempre que um estado final é alcançado, corresponde a executar o Algoritmo 2.11. Aqui, entretanto, o procedimento demanda tempo apenas linear no tamanho do texto.

Lema 2.5 *Sejam dois estados $S = (s_0, \ldots, s_m), S' = (s'_0, \ldots, s'_m) \in Q$ tais que $s_i \neq s'_i \Rightarrow s_i, s'_i > r$. Então S e S' são equivalentes,* i.e., *as linguagens aceitas a partir de ambos são idênticas.*

Prova O lema segue diretamente do fato de que qualquer caminho no grafo de dependências de D é monotonicamente não-decrescente. Assim sendo, nenhuma entrada $s_i = d_{i,j} > r$ pode pertencer a um caminho terminado em um certo $s''_m = d''_{m,j''} \leq r$. Portanto, o valor exato de $s_i > r$ não tem influência sobre o reconhecimento ou não de uma certa cadeia. ∎

Lema 2.6 *Seja $S = (s_0, \ldots, s_m) \in Q$ um estado do autômato A. Então $s_i - s_{i-1} = -1, 0$ ou 1 para $1 \leq i \leq m$.*

Prova De (2.7) temos que $d_{i,j} \leq d_{i-1,j} + 1 \Rightarrow d_{i,j} - d_{i-1,j} \leq 1$. Como as entradas de D são todas inteiras, resta mostrar que $d_{i,j} - d_{i-1,j} \geq -1$. A prova é por indução na coluna j. Se $j = 0$, o resultado segue diretamente de (2.7). Se $j > 0$ então, de (2.7), temos que $d_{i-1,j} \leq d_{i-1,j-1} + 1 \Rightarrow d_{i-1,j-1} \geq d_{i-1,j} - 1$ e, da hipótese de indução, temos que $d_{i,j-1} - d_{i-1,j-1} \geq -1 \Rightarrow d_{i,j-1} + 1 \geq d_{i-1,j-1}$. Logo, $d_{i,j} \geq \min\{d_{i-1,j-1}, d_{i,j-1} + 1, d_{i-1,j} + 1\} \geq \min\{d_{i-1,j} - 1, d_{i-1,j} - 1, d_{i-1,j} + 1\} = d_{i-1,j} - 1 \therefore d_{i,j} - d_{i-1,j} \geq -1$. ∎

De posse dos resultados acima, podemos apresentar o Algoritmo 2.12, que constrói o Autômato de Ukkonen. O algoritmo faz uso da função *próxima_coluna* que calcula o valor de uma nova coluna da matriz de Sellers a partir da coluna anterior. Com o objetivo de diminuir o número de estados, uma pequena modificação no cálculo dessa coluna é introduzida (linhas *6–8* do Algoritmo 2.13), de forma que os estados equivalentes, no espírito do Lema 2.5, são todos tratados indistintamente. Cumpre observar também que, em conseqüência do Lema 2.6, o conjunto de estados Q pode ser representado por uma árvore ternária T, de profundidade m, na qual cada nó interno possui, no máximo, três arestas eferentes rotuladas por "-1", "0" e "1". Dessa forma, cada estado $S = (s_0, \ldots, s_m)$ é representado, de forma não ambígua, pela folha de T alcançável, a partir da raiz, através do caminho rotulado por $(s_1 - s_0, s_2 - s_1, \ldots, s_m - s_{m-1})$. Os estados em Q são indexados por inteiros $0, 1, 2, \ldots$ e $index(S)$ denota o índice de S em Q. O conjunto auxiliar de estados N pode ser implementado como uma lista de apontadores para folhas de T e o conjunto de estados finais F como um lista de inteiros correspondentes a índices de estados de Q.

```
1  Algoritmo Ukkonen_AFD (Y = y_1 ··· y_m, Σ, r)
2  início
3     S ← (0, ..., m)
4     index(S) ← 0; i ← 0
5     Q ← {S}; Q' ← {S}; F ← ∅
6     se r ≥ m então
7        F ← F ∪ {0}
8     fim-se
9     enquanto N ≠ ∅ faça
10       S' ← algum elemento de N
11       N ← N \ {S'}
12       para cada a ∈ Σ faça
13          S ← próxima_coluna(S', Y, a)
14          se S ∉ Q então
15             Q ← Q ∪ {S}; N ← N ∪ {S}
16             i ← i + 1;  index(S) ← i
17             se s_m ≤ r então
18                F ← F ∪ {index(S)}
19             fim-se
20          fim-se
21          δ(index(S'), b) ← index(S)
22       fim-faça
23    fim-faça
24 fim
```

Algoritmo 2.12. Construção do Autômato de Ukkonen.

```
1  Função próxima_coluna (S', Y, a)
2  início
3     s_0 ← 0
4     para i ← 1 até m faça
5        s_i ← min{s'_{i-1} + φ(y_i, a),  s'_i + 1,  s_{i-1} + 1}
6        se s_i > r + 1 então
7           s_i ← r + 1
8        fim-se
9     fim-faça
10    devolva S = (s_0, ..., s_m)
11 fim
```

Algoritmo 2.13. Função *próxima_coluna*.

As inserções e remoções em Q correspondem a inserções e remoções de folhas de T que demandam tempo $O(m)$. As inserções e remoções em N e inserções em F podem ser efetuadas em tempo constante. A função *próxima_coluna* requer, claramente, tempo $O(m)$ e, portanto, o Algoritmo 2.12 consome tempo $O(k|\Sigma|m)$, onde k representa o número de estados de Q. Através da análise da estrutura da árvore T e do comportamento das colunas das matrizes $D(X,Y)$ quando Y mantém-se fixo e X varia em Σ^*, levando-se em conta os valores de r e $|\Sigma|$, Ukkonen obteve uma estimativa não-trivial para o número de estados, qual seja, $k = O(\min\{3^m, 2^r \cdot |\Sigma|^r \cdot m^{r+1}\})$. O espaço requerido pelo algoritmo corresponde, essencialmente, ao tamanho da árvore T, $O(mk)$, mais o espaço ocupado por N e F, ambos $O(k)$, mais $O(k|\Sigma|)$ correspondente à matriz $k \times |\Sigma|$ de inteiros utilizada para representar a função de transição δ. Portanto, o algoritmo consome, no total, espaço $O((m + |\Sigma|)k)$.

No seu artigo, baseado no resultado que $d_{i,j} \geq d_{i_1,j-1}$ (ou seja, as diagonais da matriz de programação dinâmica são monotonicamente não-decrescentes), Ukkonen observou que as colunas de D necessitariam ser calculadas apenas parcialmente. Especificamente, se, em uma coluna, a última entrada $\leq r$ acontece na linha l, então, na coluna seguinte, apenas as linhas $0, \ldots, l+1$ necessitam ser computadas e as demais podem ser consideradas como valendo $r+1$. Ukkonen propôs então uma heurística segundo a qual apenas os $3r/2$ primeiros valores de cada coluna deveriam ser computados *a priori*, uma vez que ele conjecturou que, na média, o valor de l seria $O(r)$ (fato posteriormente demonstrado por Chang e Lampe [CL92]).

2.2.4 O Algoritmo Wu-Manber

O Algoritmo proposto por Wu e Manber [WM92b], apresentado nesta seção, corresponde a uma extensão do Algoritmo *Shift-Or* (Seção 2.1.5) para comportar o caso mais geral do casamento de padrões com, no máximo, r erros, sendo, portanto, um representante da classe dos algoritmos para casamento aproximado de padrões baseados no paralelismo binário (*bit-parallelism*).

Dados, como de costume, o texto $X = x_1 \cdots x_n$ e o padrão $Y = y_1 \cdots y_m$, definimos o padrão binário $S^{j,q} = s_m^{j,q} \cdots s_1^{j,q}$ $(q \geq 0,\ 1 \leq j \leq n)$ como sendo tal que

$$s_i^{j,q} = \begin{cases} 0, \text{ se } \exists l, 0 \leq l < j \text{ t.q. } d(Y_{..i}, X_{j-l..j}) \leq q \\ 1, \text{ do contrário.} \end{cases} \tag{2.8}$$

Ou seja, $s_i^{j,q}$ indica a ocorrência terminada na posição j, com, no máximo, q erros, do prefixo $Y_{..i}$ em X. Repare que, dessa forma, uma ocorrência aproximada de Y em X terminada na posição j e com, no máximo, r erros é determinada pelo valor "0" do bit mais à esquerda de $S^{j,r}$. Repare também que, para $q = 0$, temos $S^{j,0} = S^j$, onde S^j é definido exatamente como em (2.4).

O Algoritmo Wu-Manber utiliza não apenas um padrão binário, como faz o Algoritmo *Shift-Or*, mas sim $r+1$ padrões correspondentes a $S^{j,0}, S^{j,1}, \ldots, S^{j,r}$, com o índice j variando ao longo das posições de X. Além disso, a exemplo do *Shift-Or*, o Algoritmo Wu-Manber utiliza o vetor de máscaras binárias $T = (T^a)_{a \in \Sigma}$, onde T^a é definido como em (2.5). A idéia básica é procurar obter extensões dos casamentos aproximados parciais

entre $Y_{..i}$ e fatores de X de forma a obter casamentos aproximados com $Y_{..i+1}$, e assim sucessivamente até obtermos, eventualmente, casamentos aproximados envolvendo todo o padrão Y. A hipótese de indução do algoritmo está representada no lema a seguir.

Lema 2.7 *Para todo* $1 \leq j < n$ *e* $q > 0$, *temos*

$$\begin{aligned} S^{j+1,q} =& (S^{j,q} \ll 1 \mid T^{x_{j+1}}) \\ & \& \ (S^{j,q-1} \ll 1) \\ & \& \ (S^{j+1,q-1} \ll 1) \\ & \& \ S^{j,q-1}. \end{aligned}$$

Prova Da definição (2.8), temos que o valor de $s_i^{j+1,q}$ $(1 < i \leq m)$ indica a ocorrência terminada na posição $j+1$ do prefixo $Y_{..i}$ em X com até q erros. Isso ocorre se, e somente se, pelo menos uma das seguintes quatro condições se verificar:

i) $\exists l, 0 \leq l < j$ t.q. $d(Y_{..i-1}, X_{j-l..j}) \leq q$ e $y_i = x_{j+1}$, ou seja, existe um alinhamento com até q erros entre os primeiros $i-1$ caracteres do prefixo $Y_{..i}$ e algum fator de X terminado na posição j e o último caractere de $Y_{..i}$ coincide com o caractere subseqüente de X. Este caso corresponde à extensão de uma ocorrência válida (com, no máximo, q erros) de $Y_{..i-1}$ em uma ocorrência válida de $Y_{..i}$ através de um casamento (*match*);

ii) $\exists l, 0 \leq l < j$ t.q. $d(Y_{..i-1}, X_{j-l..j}) < q$, ou seja, existe um alinhamento com até $q-1$ erros entre os primeiros $i-1$ caracteres do prefixo $Y_{..i}$ e algum fator de X terminado na posição j. Esta condição é suficiente para assegurar uma extensão de uma ocorrência válida de $Y_{..i-1}$ em uma ocorrência válida de $Y_{..i}$ através de um casamento (*match*) ou de uma substituição (*mismatch*);

iii) $\exists l, 0 \leq l \leq j$ t.q. $d(Y_{..i-1}, X_{(j+1)-l..j+1}) < q$, ou seja, existe um casamento com até $q-1$ erros do prefixo $Y_{..i-1}$ com algum fator de X já terminado na posição $j+1$. Esta condição é suficiente para assegurar a extensão de um casamento válido entre $Y_{..i-1}$ e um fator de X terminado em x_{j+1} em um casamento válido entre $Y_{..i}$ e esse mesmo fator de X mediante a supressão de y_i ou, equivalentemente, da inserção de um caractere neutro em X na posição após $j+1$ (*insert*);

iv) $\exists l, 0 \leq l < j$ t.q. $d(Y_{..i}, X_{j-l..j}) < q$, ou seja, já existe um alinhamento com até $q-1$ erros de todo o prefixo $Y_{..i}$ com algum fator de X terminado na posição j. Esta condição é suficiente para assegurar a extensão desse casamento válido entre $Y_{..i}$ e um fator de X terminado em x_j em um casamento válido entre $Y_{..i}$ e um fator de X terminado em x_{j+1} mediante a supressão deste último caractere (*delete*);

A condição (i) é a mesma condição encontrada no Lema 2.4, sendo representada pelo valor de $s_{i-1}^{j,q} | t_i^{x_{j+1}}$ (0 se verdadeira, 1 se falsa), ou seja, o i-ésimo bit menos significativo de $S^{j,q} \ll 1 \mid T^{x_{j+1}}$. A condição (ii), segundo a definição (2.8), corresponde ao valor de $s_{i-1}^{j,q-1}$, ou seja, o i-ésimo bit menos significativo de $S^{j,q-1} \ll 1$. A condição (iii) é determinada pelo valor de $s_{i-1}^{j+1,d-1}$, ou seja, o i-ésimo bit menos significativo de $S^{j+1,q-1} \ll 1$.

Finalmente, a condição (iv), corresponde ao valor de $s_i^{j,q-1}$, ou seja, o i-ésimo bit menos significativo de $S^{j,q-1}$. O lema segue diretamente dessas observações e da definição de "&". Para o caso particular $i = 1$, a verificação pode ser feita facilmente a partir da hipótese que a operação "$\ll$" sempre atribui valor 0 ao bit menos significativo. ∎

```
1  Algoritmo Wu-Manber (X = x_1 ··· x_n, Y = y_1 ··· y_m, r)
2  início
3    t ← 2^{m-1}
4    T ← máscara_ocorrência(Y)      % Algoritmo 2.9
5    S_0 ← 1···1   (m)
6    para q ← 1 até r faça
7      S_q ← S_{q-1} ≫ 1
8    fim-faça
9    para j ← 1 até n faça
10     S' ← S_0
11     S_0 ← S_0 ≪ 1 | T_j
12     para q ← 1 até r faça
13       S'' ← S_q
14       S_q ← (S_q ≪ 1 | T_j) & (S' ≪ 1) & (S_{q-1} ≪ 1) & S'
15       S' ← S''
16     fim-faça
17     se S_r < t então
18       𝕊 ← 𝕊 ∪ {j}
19     fim-se
20   fim-faça
21 fim
```

Algoritmo 2.14. Algoritmo Wu-Manber.

A fase de pré-processamento do Algoritmo 2.14 demanda o já comentado tempo $O(\lceil \frac{m}{w} \rceil (m+|\Sigma|))$, necessário à construção de tabela de máscaras T, mais tempo $O(r\lceil \frac{m}{w} \rceil)$ gasto na iniciação dos estados $S_0, \ldots, S_r$. A fase de busca requer tempo $O(nr\lceil \frac{m}{w} \rceil)$, conforme pode ser facilmente verificado, levando-se em conta que as operações das linhas *11* e *14* são executadas em tempo $O(1)$. O espaço requerido pelo algoritmo corresponde ao tamanho da tabela T mais o equivalente aos $r + 1$ estados, *i.e.*, $O((|\Sigma| + r)m)$ bits.

2.2.4.1 Filtragem O princípio da *filtragem* explora o fato de que, em determinadas situações, pode ser mais fácil determinar que o padrão Y não ocorre em uma certa posição j do texto X do que verificar se ele ocorre. Esta idéia é, de certa forma, exercitada no Algoritmo Karp-Rabin (Seção 2.1.4). No Algoritmo 2.7, o padrão Y é comparado com o conteúdo da janela $X_{j..j+m-1}$ através de uma função de *hashing*. Se esses valores forem diferentes, então esse *filtro* garante que a posição pode ser descartada. Se, pelo contrário, a posição j passa por esse filtro, as cadeias Y e $X_{j..j+m-1}$ ainda precisam ser comparadas através de um algoritmo exato qualquer. Em geral, o filtro só é capaz de assegurar a

não-ocorrência do padrão e, desta forma, um algoritmo de filtragem deve ser sempre utilizado em conjunto com algum outro algoritmo potencialmente mais custoso, todavia capaz de verificar de maneira definitiva a ocorrência do padrão em uma determinada posição do texto, ou vizinhança dela.

A eficiência dos algoritmos de filtragem está condicionada a diversos parâmetros como, por exemplo, o tamanho do alfabeto, o comprimento do padrão e o limiar de erro permitido. Uma medida importante a ser considerada quando da aplicação de um desses algoritmos é a *taxa de erros* obtida através da razão entre o número de posições não descartadas pelo filtro e a quantidade efetiva de ocorrências encontradas. Essa medida avalia a eficiência do processo de filtragem.

Os algoritmos para casamento aproximado de padrões baseados em técnicas de filtragem constituem uma categoria bastante nova (década de 90) e têm-se mostrado bastante eficientes na prática (ver resultados comparativos em [Nav01] Seção 9). Nesta seção, descreveremos como o Algoritmo Wu-Manber pode ser estendido para comportar um processo de filtragem com vistas a atingir uma melhor desempenho no caso médio.

A abordagem adotada por Wu e Manber baseia-se em um princípio simples introduzido por Rivest [Riv76] que afirma que r erros não podem afetar, simultaneamente, mais de r fatores disjuntos de Y. Mais precisamente, seja $t = \lfloor \frac{m}{r+1} \rfloor$, então $Y = Y^1 Y^2 \cdots Y^{r+1} R$, onde $Y^1, \ldots, Y^{r+1}$ são fatores de comprimento t e R um resíduo de comprimento $< t$. Se Y ocorre em X com, no máximo, r erros, então pelo menos um dos Y^k deve ocorrer em X de forma exata. Assim, podemos filtrar X, procurando por ocorrências exatas de qualquer um dos Y^k, descartando os trechos nos quais nenhum desses fatores ocorre e verificando, efetivamente, a ocorrência de Y apenas em vizinhanças de tamanho m das posições de X nas quais qualquer um dos Y^k ocorre.

Por simplicidade de notação, vamos considerar que $R = \varepsilon$. Em geral, podemos simplesmente desprezar esse resíduo. Temos então

$$Y = y_1^1 \cdots y_t^1 y_1^2 \cdots y_t^2 \cdots y_1^{r+1} \cdots y_t^{r+1} = \underbrace{y_1 \cdots y_t}_{|Y^1|=t} \underbrace{y_{t+1} \cdots y_{2t}}_{|Y^2|=t} \cdots \underbrace{y_{rt+1} \cdots y_{(r+1)t}}_{|Y^{r+1}|=t}$$

e definimos a permutação

$$\tilde{Y} = \tilde{y}_1 \cdots \tilde{y}_m = \underbrace{y_1 y_{t+1} \cdots y_{rt+1}}_{|\cdot|=r+1} \underbrace{y_2 y_{t+2} \cdots y_{rt+2}}_{|\cdot|=r+1} \cdots \underbrace{y_t y_{2t} \cdots y_{(r+1)t}}_{|\cdot|=r+1},$$

ou seja,

$$\tilde{y}_i = y_{\lceil \frac{i}{r+1} \rceil}^{((i-1) \mod (r+1))} = y_{((i-1) \mod (r+1))t + \lceil \frac{i}{r+1} \rceil}. \tag{2.9}$$

Seja também o estado $\tilde{S}^j$ $(1 \leq j \leq n)$ definido por

$$\tilde{S}^j = \tilde{s}_m^j \cdots \tilde{s}_1^j \text{ t.q. } \tilde{s}_i^j = \begin{cases} 0, \text{ se } Y_{\cdot\cdot \lceil \frac{i}{r+1} \rceil}^{(i-1) \mod (r+1)} = X_{j - \lceil \frac{i}{r+1} \rceil \cdot\cdot j} \\ 1, \text{ do contrário,} \end{cases} \tag{2.10}$$

e a máscara de ocorrências $\tilde{T}^a$ $(a \in \Sigma)$ dada por

$$\tilde{T}^a = \tilde{t}_m^a \cdots \tilde{t}_1^a \text{ t.q. } \tilde{t}_i^a = \begin{cases} 0, \text{ se } a = \tilde{y}_i \\ 1, \text{ do contrário.} \end{cases} \tag{2.11}$$

Os valores de $\tilde{S}^j$ e $\tilde{T}^a$ são ilustrados na Figura 2.12 para a instância $X = \texttt{ABAABCABA}$, $Y = \texttt{ABCABA}$ e $r = 1$.

X	A	B	A	A	B	C	A	B	A
$\tilde{T}^{x_j}$	011100	110011	011100	011100	110011	101111	011100	110011	011100
$\tilde{S}^j$	111100	110011	011100	111100	110011	101111	111100	110011	011100

Figura 2.12. Estados do filtro de Wu-Manber para $X = \texttt{ABAABCABA}$, $Y = \texttt{ABCABA}$ e $r = 1$. Temos $Y = Y^1Y^2$, onde $Y^1 = \texttt{ABC}$ e $Y^2 = \texttt{ABA}$. A ocorrência de Y^1 é terminada na posição 6 ao passo que as ocorrências de Y^2 são terminadas nas posições 3 e 9, conforme indicam os bits assinalados.

Lema 2.8 *Para todo* $1 \leq j < n$, $\tilde{S}^{j+1} = \tilde{S}^j \ll (r+1) \mid \tilde{T}^{x_{j+1}}$.

Prova Para todo $(k+1) < i \leq m$,

$$\begin{aligned}
\tilde{s}^{j+1} = 0 &\iff Y^{(i-1) \mod (r+1)}_{..\lceil \frac{i}{r+1} \rceil} = X_{j+1-\lceil \frac{i}{r+1} \rceil + 1..j+1} \\
&\iff Y^{(i-1) \mod (r+1)}_{..\lceil \frac{i}{r+1} \rceil - 1} = X_{j-(\lceil \frac{i}{r+1} \rceil - 1)+1..j} \ \wedge \ y^{(i-1) \mod (r+1)}_{\lceil \frac{i}{r+1} \rceil} = x_{j+1} \\
&\iff Y^{(i-(r+1)-1) \mod (r+1)}_{..\lceil \frac{i-(r+1)}{r+1} \rceil} = X_{j-\lceil \frac{i-(r+1)}{r+1} \rceil + 1..j} \ \wedge \ y^{(i-1) \mod (r+1)}_{\lceil \frac{i}{r+1} \rceil} = x_{j+1}.
\end{aligned}$$

O resultado segue diretamente da expressão acima e das definições de s^j_i (2.10), $y^{(i-1) \mod (r+1)}_{\lceil \frac{i}{r+1} \rceil}$ (2.9) e $\tilde{t}^{x_{j+1}}_i$ (2.11). Para $i = 1, \ldots, (r+1)$, o resultado pode ser facilmente verificado levando-se em conta que o resultado da operação $\tilde{S}^j \ll (r+1)$ possui os $(r+1)$ bits menos significativos iguais a 0. ∎

Com base na definição (2.10), temos que um dos fatores Y^k ocorre em X quando um dos $(r+1)$ bits mais significativos de $\tilde{S}^j$ vale 0. Mais precisamente, $Y^k = X_{j-t+1..j} \iff \tilde{s}^j_{m-(r+1-k)} = 0$. Portanto, o Algoritmo 2.8 pode ser, elegantemente, adaptado para comportar a busca simultânea por $Y^1, \ldots, Y^{r+1}$, bastando substituir o padrão Y pela permutação $\tilde{Y}$ e modificar a linha *7* para representar um deslocamento à esquerda de $r+1$ bits.

O custo do Algoritmo 2.15 corresponde ao custo do *Shift-Or* para um padrão de tamanho t adicionado ao custo de $O(m)$ verificações (linhas *11-15*) para cada posição de X que passar pelo filtro. O custo médio do algoritmo, portanto, é bastante sensível à taxa de erros e a função empregada para a verificação. A vantagem óbvia sobre o Algoritmo 2.14 reside na não-necessidade de manter os padrões $S_0, \ldots, S_q$.

Wu e Manber [WM92b] ainda propuseram outras adaptações do algoritmo original, por exemplo, para comportar a busca por expressões regulares. Esse conjunto flexível de algoritmos deu origem a uma ferramenta para busca aproximada de padrões largamente utilizada na plataforma UNIX—o Agrep [WM92a].

```
1  Algoritmo Wu-Manber_Filtro (X = x_1 ··· x_n, Y = y_1 ··· y_m, r)
2  início
3     l ← 2^{m−(r+1)}
4     t ← ⌊m/(r+1)⌋
5     Ỹ ← permuta(Y_{..m−(m mod t)})
6     T̃ ← máscara_ocorrência(Ỹ)
7     S̃ ← 1···1   (m−(m mod t) uns)
8     para j ← 1 até n faça
9        S̃ ← S̃ ≪ (r + 1) | T̃^{x_j}
10       se S̃ < l então
11          para k ← j até j + m − t faça
12             se distância_edição(Y, X_{k−m+1..k}) ≤ r então
13                𝕊 ← 𝕊 ∪ {k}
14             fim-se
15          fim-faça
16       fim-se
17    fim-faça
18 fim
```

Algoritmo 2.15. Algoritmo Wu-Manber_Filtro.

CAPÍTULO 3

ÁRVORES DE SUFIXOS

Estruturas são as armas do matemático.
— NICOLAS BOURBAKI

A primeira estrutura de índice que vamos examinar em detalhes é amplamente divulgada na literatura sob as mais diversas denominações: *suffix tree*, *subword tree*, *PATRICIA tree*, *compact position tree*, *etc*. Historicamente, a idéia de uma *suffix tree*—ou *árvore de sufixos*— apareceu pela primeira vez em [Mor68]. A primeira exposição sistemática dessa estrutura deveu-se, entretanto, a Weiner [Wei73]. McCreight [McC76] forneceu um outro algoritmo linear para a construção das árvores de sufixos que, sob determinadas condições, chega a representar uma economia de espaço de cerca de 25%. Quase duas décadas mais tarde, Ukkonen [Ukk95] desenvolveu o primeiro algoritmo *on-line* para a construção dessas estruturas.

Em seguida, abordaremos as chamadas *árvores de afixos*, que vêm a ser extensões das árvores de sufixos tradicionais de forma a conferir-lhes a capacidade de representar, simultaneamente, os fatores de uma determinada cadeia e também os seus reversos. As árvores de afixos foram introduzidas por Stoye [Sto95] que também forneceu um algoritmo bidirecional para a construção dessas estruturas. Stoye não conseguiu demonstrar a linearidade do seu processo de construção. Todavia, Maaß [Maa00] desenvolveu, recentemente, um processo de construção fortemente baseado naquele proposto por Stoye, e cuja linearidade pode ser demonstrada.

3.1 ÁRVORES DE SUFIXOS

3.1.1 Definição e Propriedades Elementares

Nossa abordagem segue o modelo menos convencional, todavia mais abrangente, introduzido por Giegerich e Kurtz [GK97], que se propõe a definir as árvores de sufixos a partir de estruturas mais genéricas, as árvores-Σ^+.

Definição 3.1 (Árvore-Σ^+) *Uma árvore-Σ^+ é uma árvore n-ária que apresenta as seguintes propriedades:*

i) Cada aresta é rotulada por uma cadeia não-vazia;
ii) Os rótulos de duas arestas irmãs (provenientes do mesmo vértice) distintas não podem ser iniciados pelo mesmo caractere.

O *rótulo* de um determinado caminho $v_0v_1v_2\cdots v_n$ em uma árvore T cujas arestas são rotuladas por cadeias em Σ^* (não necessariamente uma árvore-Σ^+) é dado pela concatenação $Y_1Y_2\cdots Y_n$, onde Y_i é o rótulo da aresta $v_{i-1} \rightarrow v_i$, $\forall i = 1, \ldots, n$. O rótulo de um caminho trivial formado apenas por um vértice é, por definição, a cadeia

nula ε. Dizemos que o vértice v *representa* a cadeia Y em T ou, ainda, que v é um *locus* de X e denotamos $v = \overline{Y}$, se Y é o rótulo do (único) caminho da RAIZ até v. Nesse caso, escrevemos também $rót(v) = Y$. Em particular, RAIZ $= \overline{\varepsilon}$. Se T é uma árvore-Σ^+ então, em conseqüência da propriedade (ii) da Definição 3.1, o locus de Y em T, se existir, é único. Adicionalmente, definimos a *profundidade* de um vértice v de uma árvore-Σ^+ como sendo o comprimento da cadeia representada por esse vértice, *i.e.*, $prof(v) = |rót(v)|$.

Definição 3.2 (Vértice implícito) *Um* vértice implícito *v da árvore-Σ^+ T é um par ordenado $v = (u, Y)$, onde u é um vértice de T e Y é um prefixo próprio do rótulo do caminho de u até algum descendente w de u.*

Se $v = (u, Y)$, então v *representa* a cadeia $X = rót(u)Y$, onde $rót(u)$ corresponde ao rótulo do caminho RAIZ $\rightsquigarrow u$ (*i.e.*, $\overline{rót(u)} = u$). Nesse caso, dizemos também que v é um *locus* de X e escrevemos $v = \overline{X}$. Repare que existem, potencialmente, diferentes representações para um mesmo vértice implícito v de T. Seja, por exemplo, a árvore da Figura 3.1. Nela, o vértice implícito $v = \overline{\texttt{AAAB}}$ (indicado por uma bola aberta) pode ser escrito como $v = (u, \texttt{AB})$ ou $v = (u', \texttt{AAB})$ para $u = \overline{\texttt{AA}}$ e $u' = \overline{\texttt{A}}$. Se w é o vértice mais profundo tal que v pode ser escrito como $v = (w, Y)$, para algum $Y \in \Sigma^*$, então esta é dita a *forma canônica* de v. No exemplo da Figura 3.1, a forma canônica de v é $v = (u, \texttt{AB})$. Os vértices (explícitos) de T são exatamente aqueles cujas representações canônicas são da forma (u, ε) ou, por simplicidade, u.

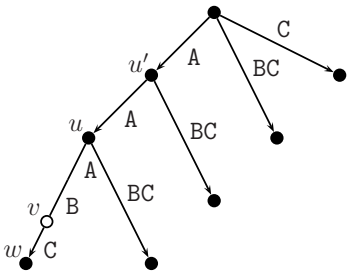


Figura 3.1. Duas possíveis representações para o vértice implícito v: $v = (u, \texttt{AB})$ ou $v = (u', \texttt{AAB})$.

Definição 3.3 (Árvore-Σ^+ atômica) *Uma aresta $u \xrightarrow{Y} v$ de uma árvore-Σ^+ T é dita* atômica *se $|Y| = 1$. T é dita uma árvore-Σ^+ atômica se todas as suas arestas são atômicas.*

As árvores-Σ^+ atômicas são também conhecidas como *Tries* *.

Definição 3.4 (Árvore-Σ^+ compacta) *Uma árvore-Σ^+ T é dita* compacta *se cada vértice interno, exceto a* RAIZ *possui grau maior ou igual a 2.*

*Segundo Knuth [Knu98b], o nome *trie* teria sido sugerido por E. Fredkin [*Comm. of the ACM* **3** (1960), 490–500] em virtude do fato de que estruturas desta natureza seriam empregadas em problemas de *information retrieval* (recuperação de informação). Por carência de um termo correspondente no idioma vernáculo, manteremos a expressão estrangeira.

Em uma *trie*, todos os vértices são explícitos enquanto que, em uma árvore-Σ^+ compacta, todo vértice interno com apenas uma "aresta" eferente é um vértice implícito (vide Figura 3.2).

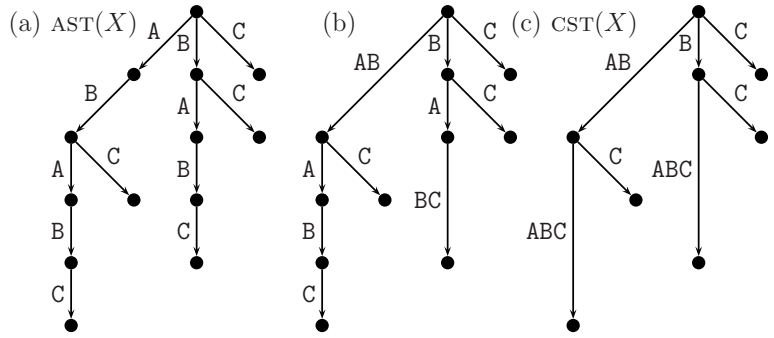


Figura 3.2. Diferentes árvores de sufixos de $X = \texttt{ABABC}$. À esquerda, uma árvore-Σ^+ atômica, à direita, a sua versão compacta e, ao centro, uma versão intermediária.

Denota-se por $\mathit{cadeias}(T)$ o conjunto de cadeias representadas por uma árvore-Σ^+ T. Em símbolos:

$$\mathit{cadeias}(T) = \{Y \mid \exists\, v = (u, V) \in T \text{ t.q. } v = \overline{Y}\}.^*$$

Repare que a aplicação "locus de" $\overline{\cdot} : \mathit{cadeias}(T) \to T$ é bem-definida, ou seja, dada uma cadeia $Y \in \mathit{cadeias}(T)$, existe um único vértice $v \in T$ t.q. $v = \overline{Y}$.

Definição 3.5 (Árvore de sufixos) *Uma árvore-Σ^+ T é dita uma* árvore de sufixos *da cadeia $X = x_1 \cdots x_n \in \Sigma^*$ se $\mathit{cadeias}(T) = \mathscr{F}(X)$.*

Embora, em geral possam existir diferentes árvores de sufixos para uma mesma cadeia X (Figura 3.2), os resultados a seguir mostram que estas ficam completamente determinadas se impomo-lhes restrições de atomicidade ou compaccidade.

Teorema 3.1 *Dada uma cadeia $X = x_1 \cdots x_n \in \Sigma^*$, a árvore de sufixos atômica de X (também denominada* suffix trie *de X), $\text{AST}(X)$, existe e é única.*

Prova (*Existência*) A prova é construtiva e por indução em $|X|$. Se $|X| = 0$, então $\text{AST}(X)$ é a árvore trivial formada apenas pela RAIZ. Se $|X| > 0$, então $X = aX'$ para algum $a \in \Sigma$ e $X' \in \Sigma^*$. Vamos mostrar como obter uma árvore de sufixos atômica de X a partir da árvore de sufixos atômica $\text{AST}(X')$. Seja $u = \overline{Y}$ o locus do maior prefixo comum a X e X', *i.e.*, $X = YbV$ e $X' = YW$ t.q. $b \in \Sigma$, $V, W \in \Sigma^*$ e $b \notin W$. Seja π o ramo[†] rotulado por bV constituído apenas de arestas atômicas. Afirmamos que a árvore T, resultante da adição de π a $\text{AST}(()X')$ a partir de u, é uma *suffix trie* de X.

*Se $G = (V_G, A_G)$ é um grafo, onde V_G representa o seu conjunto de vértices e A_G representa o seu conjunto de arestas, escrevemos, por simplicidade de notação, $u \in G$ e $u \to v \in G$ para denotar $u \in V_G$ e $u \to v \in A_G$ respectivamente. Em suma, quando não houver chance de ambigüidade, identificaremos um grafo (em prticular, uma árvore) com seu conjunto de vértices ou arestas.

[†]Sub-árvore na qual todos os vértices, exceto a folha, possuem grau 1.

Com efeito, todos os fatores de X, com exceção dos prefixos, já estão representados em $\textsc{ast}(X')$ que é uma sub-árvore de T. Os prefixos de X estão todos representados em T no (único) caminho que vai da RAIZ até a folha de π, passando por u. Além disso, T é, claramente, uma árvore-Σ^+ pois T' é uma árvore-Σ^+ e a adição do ramo π a T' não altera essa condição. Com efeito, T só não seria uma árvore-Σ^+ se u já possuisse uma aresta eferente com rótulo b. Todavia, se assim o fosse, então u não poderia ser o *locus* do maior prefixo comum a X e X'.

(*Unicidade*) Sejam T e T' duas árvores de sufixos atômicas de X e seja a aplicação

$$\begin{aligned} \psi : \quad & T \to T' \\ & u = \overline{Y} \mapsto u' = \psi(u) = \overline{Y}. \end{aligned}$$

A prova consiste em mostrar que ψ é um isomorfismo. Com efeito, ψ é bem-definida pois, dado um vértice qualquer $u = \overline{Y} \in T$, para algum $Y \in \mathscr{F}(X)$, o locus de Y também é um vértice explícito bem-definido em T' uma vez que T' é uma árvore de sufixos atômica de X. ψ é, claramente, injetiva e sobrejetiva e, mais ainda, se a aresta $u \xrightarrow{a} v$ ocorre em T, então a aresta $u' \xrightarrow{a} v'$ ocorre em T'. De fato, se $u \xrightarrow{a} v \in T$, temos $u = \overline{Y}$ e $v = \overline{Ya}$, para algum $Y \in \mathscr{F}(X)$. Logo, $u' = \overline{Y}$ e $v' = \overline{Ya}$ são vértices explícitos de T'. Mas então $u' \xrightarrow{a} v' \in T'$ pois, do contrário, temos que o pai de v' em T' é um vértice $w \neq u'$ que representa a cadeia $(Ya)_{..|Ya|-1} = Y$, o que é absurdo. ∎

Para cada árvore-Σ^+ atômica (mais geralmente, não-compacta) existe uma única árvore-Σ^+ compacta equivalente (*i.e.*, que representa o mesmo conjunto de cadeias). Esta pode ser obtida a partir daquela colapsando-se as seqüências maximais de arestas adjacentes $\{v_0 \to v_1, v_1 \to v_2, \ldots, v_k \to v_{k+1}\}$, nas quais $v_1, \ldots, v_k$ são vértices unários e v_0 é a raiz ou possui grau ≥ 2, em uma única aresta $v_0 \to v_{k+1}$ cujo rótulo corresponde à concatenação dos rótulos das arestas originais (vide Figura 3.3). Dessa propriedade e da unicidade das árvores de sufixos atômicas atestada pelo Teorema 3.1, decorre o seguinte fato:

Teorema 3.2 *Dada uma cadeia $X = x_1 \cdots x_n \in \Sigma^*$, a árvore de sufixos compacta de X, $\textsc{cst}(X)$, existe e é única.*

Um vértice (explícito) de uma árvore de sufixos T de X ramifica, *i.e.*, possui grau ≥ 2, se, e somente se, o fator que ele representa deriva à direita em X. Em particular, os vértices internos de $\textsc{cst}(X)$ estão biunivocamente associados aos fatores de X que derivam à direita. Além disso, cada folha de T sempre representa um sufixo (em geral, não-nulo) de X. Entretanto, o recíproco não necessariamente se verifica, ou seja, pode ocorrer que um sufixo de X seja representado por um vértice interno, ou até mesmo implícito, de T. Essa situação encontra-se ilustrada na Figura 3.4 (a). Entretanto, em algumas situações, é desejável que exista uma bijeção entre as folhas de T e os sufixos não-nulos de X. Essa conformação de T se verifica sempre que $X = x_1 \cdots x_n$ é tal que $x_n \neq x_i$, $\forall 1 \leq i < n$. Com efeito, se X obedece a essa propriedade, então nenhum sufixo não-nulo de X é aninhado e, portanto, não pode ser representado por um vértice interno de T. No caso geral, podemos impor essa restrição sobre X concatenando, ao seu final, um caractere especial $\$ \notin \Sigma$ denominado *caratere sentinela*. A árvore de sufixos de $X\$$

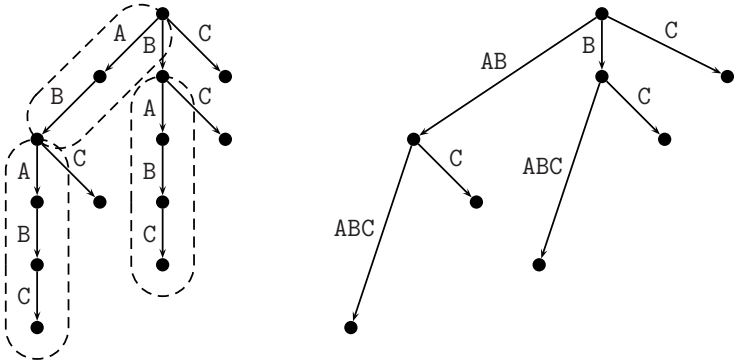


Figura 3.3. À esquerda, AST(ABABC) e, à direita, sua versão compacta CST(ABABC). As seqüências de arestas colapsadas estão indicadas na *suffix trie* pelos ciclos tracejados. Os vértices da árvore de sufixos estão didaticamente dispostos à mesma distância vertical da RAIZ do que os seus correspondentes na *trie*. Essa distância é proporcional ao comprimento do fator representado.

representa todos os fatores de X e os sufixos de X podem ser obtidos a partir do rótulo das folhas desconsiderando-se o último caractere (Figura 3.4 (b)).

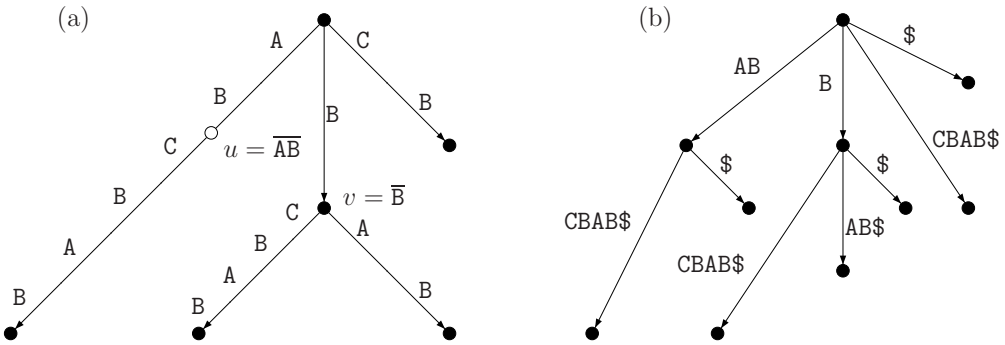


Figura 3.4. (a) Árvore de sufixos compacta CST(ABCBAB). Repare que o sufixo AB está representado pelo vértice implícito u e o sufixo B está representado pelo vértice interno v ambos indicados na figura. Os demais sufixos estão representados pelas folhas da árvore. (b) Árvore de sufixos compacta CST(ABCBAB$). Os sufixos não-nulos estão bijetivamente relacionados com as folhas.

Um último aspecto que observamos, antes de nos devotarmos aos processos de construção das árvores de sufixos, diz respeito ao tamanho dessas estruturas. Na *suffix trie* de X, cada fator de X é representado por um vértice explícito e, portanto, o número de vértices limita-se com a quantidade de fatores distintos de X, ou seja, $O(|X|^2)$ conforme disposto em (1.1). Esse tamanho quadrático no comprimento da cadeia representada faz das árvores de sufixos atômicas estruturas inapropriadas, no espírito da discussão da página 3. Todavia, se consideramos as árvores de sufixos na versão compacta, temos um resultado satisfatório.

Teorema 3.3 CST$(X = x_1 \cdots x_n)$ *possui* $O(n)$ *vértices.*

Prova CST(X) possui, no máximo, n folhas correspondentes aos sufixos distintos não-aninhados de X. Além disso, cada vértice interno de CST(X) possui grau maior ou igual a 2. Logo, o número máximo de vértices de CST(X) ocorre quando a mesma é uma árvore binária de n folhas e, nesse caso, vale $2n - 1 = O(n)$. ∎

3.1.2 Construção

O primeiro a mostrar que as árvores de sufixos podiam ser construídas de maneira direta em tempo linear foi Weiner [Wei73]. Embora o algoritmo de Weiner tenha grande importância histórica, apresentaremos, nesta seção, dois outros algoritmos mais eficientes e mais amplamente utilizados, devidos, respectivamente, a McCreight [McC76] e Ukkonen [Ukk95]. Aquele é ligeiramente mais rápido na prática, enquanto este, além de ser o mais intuitivo, apresenta a desejável propriedade de ser um algoritmo *on-line*. O fato surpreendente é que, embora esses dois algoritmos baseiem-se em idéias bastante distintas, eles são, essencialmente, bastante semelhantes e o primeiro pode, na verdade, ser derivado a partir do segundo [GK97].

3.1.2.1 Construção *à la* McCreight O algoritmo devido a McCreight constrói a árvore de sufixos compacta CST(X) iterativamente do maior para o menor sufixo, ou seja, percorrendo a cadeia X da esquerda para a direita. A cada passo $1 < i \leq n$, uma nova folha $leaf_i$ correspondente ao sufixo $X_{i..}$ é inserida na árvore T_{i-1}, dando origem a T_i, cujas folhas representam os sufixos $X_{j..}$ para $1 \leq j \leq i$.* O Algoritmo 3.1 representa o esquema geral do processo de construção de McCreight. Aqui, $head_i = head(X_{i..}, T_{i-1})$ denota o maior prefixo de $X_{i..}$ que é também prefixo de $X_{j..}$ para algum $1 \leq j < i$ ou, equivalentemente, o maior prefixo de $X_{i..}$ que está representado em T_{i-1} (possivelmente por um vértice implícito).

```
1 Algoritmo Esquema_McCreight (X = x_1 ··· x_n)
2 início
3   crie a árvore inicial T_1 : RAIZ --X--> X̄
4   para i ← 2 até n faça
5     localize head_i
6     insira uma nova folha leaf_i = X_{i..} como filha de head_i
      através da aresta head_i --tail_i--> leaf_i, onde X_{i..} = head_i tail_i.
7   fim-faça
8 fim
```

Algoritmo 3.1. Algoritmo McCreight em alto nível.

A esta altura, deve estar claro que o "Calcanhar de Aquiles" do Algoritmo 3.1 reside na linha *5*, que compreende a localização dos *loci* de $head_i$. Uma busca por força-bruta iniciada, a cada passo, a partir da RAIZ levaria-nos, fatalmente, à complexidade

*O Algoritmo McCreight exige $X = x_1 \cdots x_n$ com $x_n \neq x_i, \forall\ 1 \leq i < n$. Se isso não acontece, recorre-se ao caractere sentinela.

quadrática. Devemos, portanto, desenvolver um artifício que nos permita encurtar essa busca, por exemplo, começando por um vértice mais próximo do ponto que desejamos alcançar. Para tanto, lancemos mão da seguinte propriedade:

Lema 3.1 *Se $head_{i-1} = aY$ para algum caractere $a \in \Sigma$ e uma cadeia $Y \in \Sigma^*$, então Y é prefixo de $head_i$.*

Prova Se $head_{i-1} = aY$, então aY é prefixo de $X_{j..}$ para algum $j < i - 1$. Logo, Y é prefixo de $X_{i..}$ e de $X_{k..}$, para $k = j+1 < i$. Da definição de $head_i$, temos, imediatamente, o enunciado. ∎

Corolário 3.1 *Se $head_{i-1} = aY$ para algum caractere $a \in \Sigma$ e uma cadeia $Y \in \Sigma^*$, então qualquer prefixo de Y é também prefixo de $head_i$.*

Definindo a função auxiliar σ por

$$\begin{aligned} \sigma : \text{CST}(X) \setminus \{\text{RAIZ}\} &\rightarrow \text{CST}(X) \\ v = \overline{aY} &\mapsto \overline{Y} \end{aligned} \tag{3.1}$$

onde $\text{CST}(X)$ é identificado com o seu conjunto de vértices, chegamos ao seguinte resultado fundamental como conseqüência imediata do Corolário 3.1.

Corolário 3.2 *O* locus *de $head_i$ em T_{i-1} é descendente de $\sigma(v)$ para qualquer ancestral v de $\overline{head_{i-1}}$.*

O Algoritmo McCreight prevê que sejam representadas na árvore de sufixos, conexões auxiliares na forma $v \dashrightarrow \sigma(v)$ para os vértices não-terminais de $\text{CST}(X)$. Conexões dessa natureza são denominadas *suffix links*. É também comum referir-se ao vértice $\sigma(v)$ como o *suffix link* de v.

Definição 3.6 (*Locus* contraído) *O* locus contraído *de uma cadeia $Y \in \Sigma^*$ em uma árvore-Σ^+ T, denotado por $\underline{Y}$, é o* locus *do maior prefixo de Y explicitamente representado em T.*

Ilustrando a Definição 3.6 acima, temos, na árvore da Figura 3.1, $\underline{\texttt{AAAB}} = \underline{\texttt{AAA}} = \underline{\texttt{AAB}} = \overline{\texttt{AA}} = u$, $\underline{\texttt{ABA}} = \overline{\texttt{AB}} = u'$ e $\underline{\texttt{BA}} = \underline{\texttt{B}} = \overline{\varepsilon} = \text{RAIZ}$.

A essência da otimização proposta por McCreight consiste em utilizar o resultado do Corolário 3.2 para empreender a busca por $\overline{head_i}$, a cada passo i, não mais a partir da RAIZ, mas sim a partir do *suffix link* do *locus* contraído de $head_{i-1}$ em T_{i-2}. Outro ponto chave é a divisão desse procedimento de busca em duas partes, uma mais "acelerada" e outra mais "lenta", correspondentes aos algoritmos 3.3 e 3.4 respectivamente. A Função *busca_rápida* identifica o *locus* de uma dada cadeia em uma dada sub-árvore desde que, necessariamente, a cadeia em questão esteja representada nessa sub-árvore, fato este que confere maior celeridade à busca. Se o *locus* procurado for um vértice implícito, a Função *divide_aresta* (Algoritmo 3.2) é invocada para transformar esse vértice implícito expresso em forma canônica em um vértice explícito. A Função *busca_lenta*, por sua vez, identifica o *locus* do maior prefixo de uma dada cadeia representado em uma dada

```
1 Função divide_aresta (u --W--> v, l)
2 início
3   crie um novo vértice w
4   crie uma nova aresta u --W_..l--> w
5   crie uma nova aresta w --W_l+1..--> v
6   remova a aresta u --W--> v
7 fim
```

Algoritmo 3.2. A função *divide_aresta* torna explícito o vértice implícito canônico $w = (u, W_{..l})$.

```
1  Função busca_rápida (u, Y)
2  início
3    v ← u
4    i ← 1
5    enquanto i ≤ |Y| faça
6      Z ← rótulo da aresta v → w tal que y_i = z_1
7      se |Y_i..| ≥ |Z| então
8        v ← w
9        i ← i + |Z|
10     senão
11       v ← divide_aresta(v --Z--> w, |Y_i..|)
12       i ← |Y| + 1
13     fim-se
14   fim-faça
15   devolva v
16 fim
```

Algoritmo 3.3. A Função *busca_rápida* localiza o locus de Y na sub-árvore enraizada por u com a pré-condição de que Y esteja representada nessa sub-árvore.

```
1  Função busca_lenta (u, Y)
2  início
3    v ← u
4    i ← 1
5    enquanto ∃ uma da aresta v -Z-> w tal que y_i = z_1 faça
6      j ← 0
7      enquanto j < |Z| e y_{i+j} = z_{1+j} faça
8        j ← j + 1
9      fim-faça
10     se j = |Z| então
11       v ← w
12     senão
13       v ← divide_aresta(v -Z-> w, j)
14     fim-se
15     i ← i + j
16   fim-faça
17   devolva v
18 fim
```

Algoritmo 3.4. A Função *busca_lenta* localiza o locus do maior prefixo de Y representado na sub-árvore enraizada por u mediante varredura de Y caractere-a-caractere.

sub-árvore percorrendo essa cadeia símbolo-a-símbolo. O Algoritmo 3.5 representa o procedimento de McCreight para a construção da árvore de sufixos da cadeia X.

Alguns comentários se fazem necessários acerca do Algoritmo 3.5. Vamos supor que o passo $i-1$ acaba de ser concluído e considerar a i-ésima iteração. Nas linhas *7–14*, é identificado um sufixo $Y \in \Sigma^*$ de $head_{i-1}$ de tal forma que:

i) $head_{i-1} = UVY$;
ii) Se o *locus* contraído de $head_{i-1}$ em T_{i-2}, $\underline{head_{i-1}}$, é diferente da RAIZ, então ele é o *locus* de UV. Do contrário, V é vazio;
iii) U é uma cadeia de, no máximo, um caractere e que é vazia se, e somente se, $head_{i-1}$ for vazio. (3.2)

A seguir, na linha *15*, é feito uso do Corolário 3.2 para encurtar a busca por $head_i$, conforme discutido anteriormente. A execução dessa linha no passo i requer, entretanto, que o *suffix link* de $\underline{head_{i-1}}$ esteja definido em T_{i-1}. Com efeito, na i-ésima iteração, no máximo um novo vértice interno, correspondente a $\overline{head_i}$, é criado. Essa criação se dá durante uma chamada à Função *divide_aresta*. Também durante essa iteração, o *suffix link* de $\overline{head_{i-1}}$ é estabelecido (linha *17*). Portanto temos o seguinte resultado:

Lema 3.2 *Na árvore intermediária T_i, resultante da i-ésima iteração do Algoritmo McCreight, o único vértice interno para o qual o* suffix link *pode não estar definido é $\overline{head_i}$.*

Na linha *16*, o locus de VY, prefixo de $head_i$, é localizado a partir de $u = \overline{V}$ através da função *busca_rápida*. Repare que essa função requer que a cadeia Y esteja representada

1 **Algoritmo** *McCreight* ($X = x_1 \cdots x_n$)
2 **início**
3 crie a árvore inicial T_1 : RAIZ $\xrightarrow{X} \overline{X}$
4 $tail_i \leftarrow X$; $\underline{head_i} \leftarrow$ RAIZ; $\overline{head_i} \leftarrow$ RAIZ
5 **para** $i \leftarrow 2$ **até** n **faça**
6 $tail_{i-1} \leftarrow tail_i$; $\underline{head_{i-1}} \leftarrow \underline{head_i}$; $\overline{head_{i-1}} \leftarrow \overline{head_i}$
7 **se** $\underline{head_{i-1}} \neq \overline{head_{i-1}}$ **então**
8 $Y \leftarrow$ rótulo da aresta $\underline{head_{i-1}} \rightarrow \overline{head_{i-1}}$
9 **se** $\underline{head_{i-1}} =$ RAIZ **então**
10 $Y \leftarrow Y_{2..}$
11 **fim-se**
12 **senão**
13 $Y \leftarrow \varepsilon$
14 **fim-se**
15 $u \leftarrow \sigma(\underline{head_{i-1}})$
16 $v \leftarrow$ *busca_rápida*(u, Y)
17 crie o *suffix link* $\overline{head_{i-1}} \dashrightarrow v = \sigma(\overline{head_{i-1}})$
18 **se** $\overline{head_{i-1}} =$ RAIZ **então**
19 $W \leftarrow X_{i..}$
20 **senão**
21 $W \leftarrow tail_{i-1}$
22 **fim-se**
23 $\overline{head_i} \leftarrow$ *busca_lenta*(d, W)
24 insira uma nova folha $leaf_i = \overline{X_{i..}}$ como filha de $\overline{head_i}$
através da aresta $\overline{head_i} \xrightarrow{tail_i} leaf_i$, onde $X_{i..} = head_i tail_i$.
25 **fim-faça**
26 **fim**

Algoritmo 3.5. Algoritmo McCreight para construção de árvores de sufixos.

na sub-árvore enraizada por u. Isso, de fato, ocorre uma vez que, por definição de $head_{i-1}$, a cadeia UVY é prefixo de algum sufixo $X_{j..}$ para $1 \leq j < i-1$. Mas então a cadeia VY é prefixo de algum sufixo $X_{k..}$ para $1 \leq k \leq i-1$ e, portanto, esse sufixo deve estar representado em T_{i-1}. De acordo com as definições de U, V e Y, o vértice $v = \overline{VY}$ resultante da execução do *busca_rápida* corresponde, precisamente, ao *suffix link* de $head_{i-1}$. Essa informação é adicionada à árvore intermediária T_{i-1} na linha *17*. A partir desse ponto, não há mais nenhuma informação extra que possa servir de auxílio na busca por $\overline{head_i}$. A função *busca_lenta* (linha *23*) é então utilizada para a localização de $\overline{head_i}$ a partir de $v = \overline{VY}$. Para tanto, é feito uso da seguinte propriedade trivial:

Lema 3.3 *Seja $head_{i-1} = UVY$ definidos como em (3.2) e $head_i = VYZ$ para algum $Z \in \Sigma^*$. Se $head_{i-1} \neq \varepsilon$, então Z é prefixo de $tail_{i-1}$. Do contrário, Z é prefixo de $X_{i..}$.*

Uma vez localizado o *locus* de $head_i$, uma nova folha para representar o sufixo $X_{i..}$ é finalmente inserida a partir desse vértice através de uma aresta rotulada por $tail_i$ (linha *24*).

Como exemplo do processo descrito no parágrafo anterior, a Figura 3.5 ilustra a sexta iteração da construção da árvore de sufixos da cadeia $X = \texttt{AABBAABBC}$.

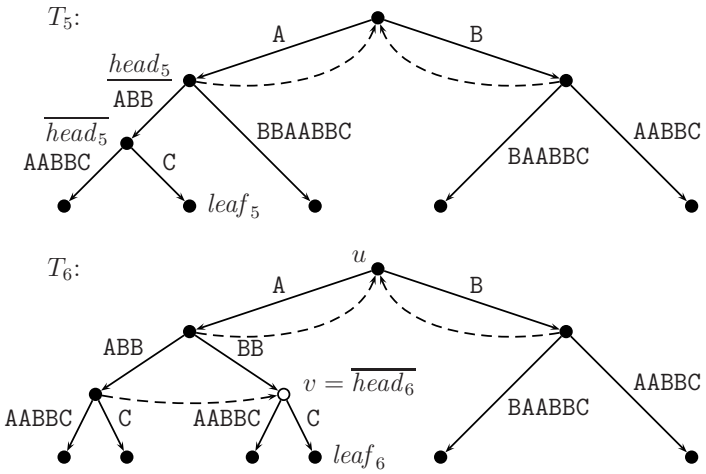


Figura 3.5. Exemplo de iteração do Algoritmo McCreight. A figura ilustra a construção da árvore intermediária T_6 a partir de T_5 para $X = \texttt{AABBAABBC}$. Nessa iteração temos $U = \texttt{A}$, $V = \varepsilon$, $Y = \texttt{ABB}$ e $tail_5 = \texttt{C}$. Os valores de $u = \sigma(\underline{head_5}) = \overline{V}$ e $v = \overline{VY} = busca\_rápida(u, Y)$ estão indicados na árvore inferior. Observe que $\overline{\overline{head_6}} = busca\_lenta(v, tail_5) = v$. Repare também que um novo *suffix link* $\overline{head_5} \dashrightarrow v$ foi inserido nessa iteração.

Teorema 3.4 *O Algoritmo McCreight demanda tempo linear no comprimento da cadeia de entrada.*

Prova A construção da árvore de sufixos $\textsc{cst}(X = x_1 \cdots x_n)$ é executada em n iterações nas quais cada operação, exceto as chamadas às funções *busca_rápida* e *busca_lenta* (linhas *16* e *23*), são executadas em tempo constante. O custo total do algoritmo pode, portanto, ser obtido a partir do tempo total gasto em cada uma dessas funções ao longo das n iterações.

O tempo gasto durante a execução de *busca_rápida(u,Y)* na i-ésima iteração é linearmente proporcional ao número de vértices (explícitos) intermediários, int_i, no caminho entre u e (u, Y). Com efeito, todas as operações da Função *busca_rápida* requerem tempo $O(1)$, inclusive a chamada à Função *divide_aresta* e a verificação da linha *6* que requer, na implementação mais óbvia, um máximo de $|\Sigma|$ comparações (assumimos $|\Sigma|$ constante). Portanto, o custo total das chamadas a *busca_rápida* é dado pela soma $\sum_{i=2}^{n} int_i$, a qual vamos estimar indiretamente. Seja res_i o menor sufixo de $X_{i..}$ submetido às operações de busca durante a i-ésima iteração ($res_i = YZtail_i$ nos termos do Lema 3.3). Repare que cada vértice intermediário encontrado durante o *busca_rápida* de Y corresponde a um fator não-nulo que não pode ocorrer como prefixo em res_{i+1}. De fato, cada vértice intermediário encontrado é um ancestral do *locus* contraído de $head_i$ em T_{i-1} e, como no passo $i+1$ a busca rápida se inicia a partir de $\sigma(\underline{head_i})$, os fatores representados pelas arestas aferentes a esses ancestrais são automaticamente desconsiderados. Portanto, temos que $|res_{i+1}| \leq |res_i| - int_i \ \therefore$

$$|res_n| \leq |res_{n-1}| - int_{n-1} \leq |res_{n-2}| - int_{n-2} - int_{n-1} \leq \cdots \leq |res_2| - \sum_{i=2}^{n-1} int_i,$$

ou seja, $\sum_{i=2}^{n-1} int_i \leq |res_2| - |res_n|$. Como, claramente, $|res_2| = n-1$ e $|res_n| = 0$, temos

$$\sum_{i=2}^{n-1} int_i \leq n-1. \tag{3.3}$$

O custo da chamada à Função *busca_rápida* na i-ésima iteração é linearmente proporcional ao comprimento da cadeia varrida, ou seja, $|Z|$ nos termos do Lema 3.3. Com efeito, todas as instruções da função podem ser assumidas como requerendo tempo constante, à exceção da linha *5* que demanda, no máximo, $|\Sigma|$ comparações. Também do Lema 3.3, segue diretamente que $|Z| = |head_i| - |head_{i-1}| + 1$. Portanto, o custo total das chamadas a *busca_lenta* é limitado pela soma

$$\sum_{i=2}^{n} |head_i| - |head_{i-1}| + 1 = |head_n| - |head_1| + (n-1).$$

Como $head_n = head_1 = \varepsilon$, temos

$$\sum_{i=2}^{n} |head_i| - |head_{i-1}| + 1 = n-1. \tag{3.4}$$

De (3.3) e (3.4), temos que o custo do algortimo é $O(2(n-1)) = O(n)$. ∎

3.1.2.2 Construção *à la* Ukkonen O Algoritmo devido a Ukkonen constrói a árvore de sufixos compacta $\textsc{cst}(X = x_1 \cdots x_n)$ em n iterações de maneira incremental, *i.e.*, na i-ésima iteração, a árvore intermediária $T_i = \textsc{cst}(X_{..i})$ é construída a partir de $T_{i-1} = \textsc{cst}(X_{..i-1})$. Ao contrário do Algoritmo McCreight, o processo de Ukkonen não exige que X não possua sufixos aninhados.

A invariante do Algoritmo Ukkonen corresponde ao seguinte resultado elementar:

Lema 3.4 *Dada a cadeia* $X = x_1 \cdots x_n$ *e o inteiro* $1 < i \leq n$, *temos*

$$\mathscr{F}(X_{..i}) = \mathscr{F}(X_{..i-1})x_i \cup \{\varepsilon\}.$$

O lema acima nos diz que os sufixos do prefixo $X_{..i}$ são obtidos a partir dos sufixos de $X_{..i-1}$ concatenando-se, ao final de cada um destes, o caractere x_i e adicionando-se, ao novo conjunto obtido, o sufixo vazio. O funcionamento geral do Algoritmo Ukkonen pode então ser percebido através do Algoritmo 3.6. A Figura 3.6 ilustra o processo de construção para $X = \texttt{ABABC}$. Um ponto a ser destacado no Algoritmo 3.6 corresponde à linha 7. Repare que o vértice u pode ser um vértice implícito (*e.g.* $\overline{X_{3..4}}$ em T_4 na Figura 3.6) e, nesse sentido, não faz sentido falar em "aresta" partindo de u. Ao invés disso, adotamos um conceito mais geral de "transição". Em geral, dizemos que existe uma *a-transição* partindo de um vértice w em uma árvore-Σ^+ T se o vértice implícito $w' = (w, a)$ está representado em T. Portanto, a transição a que referimos na linha 7 pode ser uma aresta como também pode ser apenas o caractere x_i que representa a mudança do vértice implícito $u = ([(X_{j..i-1})_{..k}], X_{j+k..i-1})$ para o vértice (também possivelmente implícito) $v = ([(X_{j..i-1})_{..k}], X_{j+k..i})$.

```
1  Algoritmo Esquema_Ukkonen (X = x_1 ··· x_n)
2  início
3     crie a árvore inicial T_1 : RAIZ --x_1--> x_1
4     para i ← 2 até n faça
5        para j ← 1 até i − 1 faça
6           u ← X_{j..i-1}
7           se ∄ uma x_i-transição a partir de de u então
8              se u é uma folha então
9                 estenda a aresta aferente a u concatenando x_i
                  ao final do seu rótulo
10             senão
11                insira uma nova folha w = X_{j..i} como filha de u
                  através de uma aresta rotulada por x_i.
12             fim-se
13          fim-se
14       fim-faça
15    fim-faça
16 fim
```

Algoritmo 3.6. Algoritmo Ukkonen em alto nível.

Uma vez apresentada a idéia geral do algoritmo, passamos a discutir os pormenores capazes de torná-lo eficiente.

Definição 3.7 (Fronteira) *A* fronteira *da árvore de sufixos* $\text{CST}(X = x_1 \cdots x_n)$ *é a seqüência de vértices definida por* $(w_1, \ldots, w_n)$ *na qual* $w_1 = \overline{X}$ *e* $w_j = \sigma^{j-1}(w_1) = \sigma(w_{j-1})$ *para* $j > 1$, *onde* σ *é a função definida como em* (3.1).

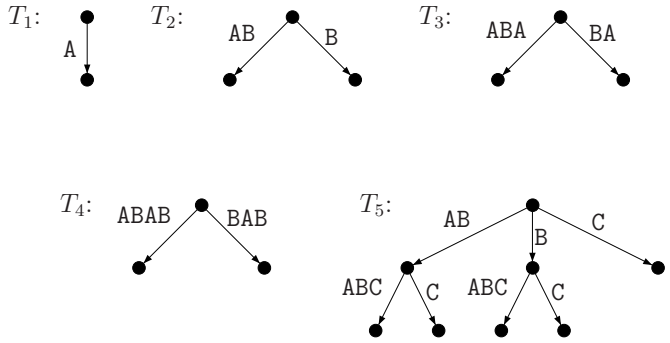


Figura 3.6. Construção de CST(ABABC) pelo Algoritmo Ukkonen.

Observe que a fronteira de $\text{CST}(X)$ corresponde, exatamente, à seqüência dos *loci* de todos os sufixos de X, do maior para o menor. Esses vértices podem ser todos facilmente localizados partindo-se do vértice que representa a cadeia completa e seguindo-se os *suffix links*. Portanto, para obter as sucessivas referências para $\overline{X_{j..i-1}}$ em T_{i-1}, na i-ésima iteração do algoritmo, é apenas necessário adotar esse procedimento, contanto que o caminho de *suffix links* a partir de $\overline{X_{..i-1}}$ esteja apropriadamente definido.

Dada a árvore intermediária $T_{i-1} = \text{CST}(X_{..i-1})$ e a sua fronteira $(w_1, \ldots, w_n)$, definimos o par de índices

$$j' = \min\{1 \leq j \leq i-1 \mid w_j \text{ não é uma folha de } T_{i-1}\} \text{ e}$$

$$j'' = \min\{1 \leq j \leq i-1 \mid w_j \text{ não possui uma } x_i\text{-transição}\}.$$

Os vértices $w_{j'}$ e $w_{j''}$ assim definidos são, repectivamente, chamados de *vértice ativo* e *vértice terminador* de T_{i-1}. Repare que o vértice ativo é bem-definido uma vez que $w_{i-1} = \text{RAIZ}$ não é uma folha. O vértice terminador, entretanto, pode assumir um valor indeterminado. Os vértices na fronteira de T_{i-1} podem, então, ser particionados em três intervalos:

i) $\{w_j \mid 1 \leq j < j'\}$. Os vértices deste intervalo correspondem aos *loci* dos sufixos de $X_{..i-1}$ que são representados por folhas em T_{i-1}. Se $u = \overline{x_j \cdots x_{i-1}}$ é um desses vértices então, durante a i-ésima iteração do algoritmo, a aresta aferente a u tem seu rótulo estendido, através da concatenção do caractere x_i, de forma que u passe a representar o sufixo $x_j \cdots x_i$. Se identificamos o fator $X_{l..r}$, conforme explicado à página 10, com o par (l, r), então apenas o segundo elemento do par correspondente ao rótulo da aresta aferente a u necessita ser alterado (acrescido de 1) para contemplar essa atualização. Todavia, como nesse caso o rótulo dessa aresta sempre corresponde a um sufixo de $X_{..i-1}$ que será modificado para representar um sufixo de $X_{..i}$, essa atualização de forma explícita pode ser evitada bastando, para tanto, representar o rótulo da aresta em questão por um par na forma (l, ∞), originando o que se costuma chamar de uma *aresta aberta*.

ii) $\{w_j \mid j' \leq j < j''\}$. Os vértices deste intervalo não são folhas nem tampouco possuem x_i-transições. Com efeito, se $w_{j'} = \overline{X_{j'..i-1}}$ não é uma folha de T_{i-1}, então

$X_{j'..i-1}$ é prefixo de algum outro sufixo $X_{k..i-1}$ $(k < j')$ de $X_{..i-1}$. Mas então temos que $w_{j'+1} = \sigma(w_{j'}) = \overline{X_{j'+1..i-1}}$, onde $X_{j'+1..i-1}$ é prefixo de $X_{k+1..i-1}$ que, por sua vez, também é sufixo de $X_{..i-1}$ e que, portanto, deve estar representado em T_{i-1}. Logo, $w_{j'+1}$ é ancestral de $\overline{X_{k+1..i-1}}$ e, portanto, não pode ser uma folha de T_{i-1}. Esse resultado se estende, de maneira análoga, para os demais valores de $j > j'$.

iii) $\{w_j \mid j'' \leq j \leq i-1\}$. Os vértices deste intervalo correspondem aos vértices internos de T_{i-1} que possuem x_i-transições mais a RAIZ. Com efeito, se $w_{j''} = \overline{X_{j''..i-1}}$ possui uma x_i-transição, então $X_{j''..i-1}$ é prefixo de um certo sufixo $X_{k..i-1}$ $(k < j'')$ de $X_{..i-1}$ tal que $x_{k+i-j''} = x_i$. Mas então temos que $w_{j''+1} = \sigma(w_{j''}) = \overline{X_{j''+1..i-1}}$, tal que $X_{j''+1..i-1}$ é prefixo de $X_{k+1..i-1}$ (com $x_{(k+1)+i-(j''+1)} = x_i$) que, por sua vez, também é sufixo de $X_{..i-1}$ e que, portanto, deve estar representado em T_{i-1}. Logo, $w_{j''+1}$ é ancestral de $\overline{X_{k+1..i-1}}$ e o rótulo do caminho de $w_{j''+1}$ até $\overline{X_{k+1..i-1}}$ em T_{i-1} é iniciado por x_i. De maneira análoga, temos o mesmo resultado para os demais valores de $j > j''$. Os vértices deste intervalo, não necessitam de atualização na i-ésima iteração do algoritmo.

Das observações acima, temos que, a cada iteração i, apenas os vértices do segundo grupo necessitam ser explicitamente atualizados. Em outras palavras, no início da i-ésima iteração, precisamos posicionar um apontador sobre o vértice ativo de T_{i-1} e percorrer a fronteira até o vértice terminador (ou até a RAIZ) fazendo as atualizações necessárias. O seguinte resultado demonstra como o vértice ativo de T_{i-1} pode ser facilmente localizado no início da i-ésima iteração.

Lema 3.5 *Seja $(u, X_{l..i-1}) = (u, (l, i-1))$ uma representação implícita do vértice terminador $w_{j''}$ de T_{i-1} (se este existir). Então $(u, X_{l..i}) = (u, (l, i))$ é uma representação implícita do vértice ativo de T_i.*

Prova Se $w_{j''}$ é o vértice terminador de T_{i-1} então $w_{j''} = [x_{j''} \cdots x_{i-1}]$ onde $x_{j''} \cdots x_{i-1}$ é o maior sufixo de $X_{..i-1}$ tal que $x_{j''} \cdots x_{i-1}x_i$ também ocorre em $X_{..i-1}$. Logo $\overline{x_{j''} \cdots x_{i-1}x_i} = (u, (l, i))$ também é um vértice de T_{i-1} e, mais ainda, $x_{j''} \cdots x_{i-1}x_i$ é o maior sufixo de $X_{..i}$ que ocorre mais de uma vez em $X_{..i}$ (uma vez que ocorre em $X_{..i-1}$), o que equivale a dizer que $\overline{x_{j''} \cdots x_{i-1}x_i}$ é o vértice ativo de T_i. ∎

O cerne do processo de construção de Ukkonen (Algoritmo 3.10) reside na Função *atualiza*. A Função *atualiza* (Algoritmo 3.9) é invocada no passo i para atualizar T_{i-1}, dando origem a T_i. Essa função recebe como argumento uma referência canônica para o vértice ativo de T_{i-1} e, partindo desse vértice, percorre a fronteira de T_{i-1} até que o vértice terminador ou a RAIZ seja alcançada. A função vale-se do fato de que, se $(u, (l, i-1))$ é uma referência canônica para um determinado vértice w_j na fronteira de T_{i-1}, então $(\sigma(u), (l, i-1))$ é uma referência implícita pra w_{j+1}. Para cada vértice $w_j = \overline{X_{j..i-1}}$ encontrado nesse trajeto pela fronteira, uma nova folha, representando o sufixo $X_{j..i}$, é adicionada como sua filha através de uma aresta aberta rotulada por (i, ∞). Além disso, também são inseridos os *suffix links* da forma $w_{j-1} \dashrightarrow w_j$. A Função *atualiza* devolve uma referência implícita ao vértice terminador de T_{i-1} que será utilizada para a determinação do vértice ativo da recém-computada árvore intermediária T_i.

O Algoritmo Ukkonen ainda faz uso de duas outras funções auxiliares: a Função *testa_e_divide* e a Função *normaliza*. A Função *testa_e_divide* (Algoritmo 3.7) recebe, na i-ésima iteração, vértices implícitos em forma canônica mais um símbolo $a = x_i$ e testa se o vértice fornecido é o vértice terminador (ou a raiz) de T_{i-1}, *i.e.*, se possui uma a-transição. O resultado do teste é devolvido como primeiro elemento de um par. Além disso, caso o resultado do teste seja positivo, o vértice é então tornado explícito e essa referência explícita é devolvida no segundo elemento do par acima referido. A Função *normaliza* (Algoritmo 3.8) simplesmente recebe um vértice expresso em forma implícita e devolve a forma canônica equivalente.

```
1  Função testa_e_divide ((u, (l, r)), a)
2  início
3    se l ≤ r então
4      localize a aresta u --(l',r')--> v t.q. x_l' = x_l
5      se x_{l'+(r-l)+1} = a então
6        devolva (TRUE, u)
7      senão
8        crie um novo vértice w
9        crie uma aresta u --(l',l'+(r-l))--> w
10       crie uma aresta w --(l'+(r-l)+1,r')--> v
11       remova a aresta u → v
12       devolva (FALSE, w)
13     fim-se
14   senão
15     se u = RAIZ ou ∃ uma aresta u → v com rótulo iniciado por a então
16       devolva (TRUE, u)
17     senão
18       devolva (FALSE, u)
19     fim-se
20   fim-se
21 fim
```

Algoritmo 3.7. Função *testa_e_divide*.

Teorema 3.5 *O Algoritmo Ukkonen constrói a árvore de sufixos* $\text{CST}(X = x_1 \cdots x_n)$ on-line *e em tempo* $O(n)$.

Prova O Algoritmo Ukkonen constrói as árvores intermediárias $T_1, T_2, \ldots, T_n$ de maneira incremental sendo que, para a construção de T_i, apenas o prefixo $X_{..i}$ necessita ser conhecido $\therefore$ o algoritmo é *on-line*.

O custo do Algoritmo 3.10 é dado pelo tempo da construção da árvore inicial, que é constante, mais o tempo gasto nas $n-1$ chamadas à Função *atualiza* (linha *6*) mais o tempo gasto nas $n-1$ chamadas à Função *normaliza* (linha *7*). Observe, entretanto, que a Função *atualiza* possui, por sua vez, chamadas à Função *normaliza*. Portanto, procederemos da seguinte forma: primeiro, estimaremos o custo total das chamadas

```
1  Função normaliza ((u, (l, r)))
2  início
3    se r < l então
4      devolva (u, (l, r))
5    senão
6      localize a aresta u --(l',r')--> v t.q. x'_l = x_l
7      enquanto (r' − l') ≤ (r − l) faça
8        l ← l + (r' − l') + 1
9        u ← v
10       se l ≤ r então
11         localize a aresta u --(l',r')--> v t.q. x'_l = x_l
12       fim-se
13     fim-faça
14     devolva (u, (l, r))
15   fim-se
16 fim
```

Algoritmo 3.8. Função *normaliza*.

```
1  Função atualiza ((u, (l, i − 1)))
2  início
3    w' ← NIL
4    (end, w) ← testa_e_divide((u, (l, i − 1)), x_i)
5    enquanto ¬end faça
6      crie um novo vértice v e uma aresta w --(i,∞)--> v
7      se w ≠ NIL então
8        crie o suffix link w' ⇢ w
9      fim-se
10     w' ← w
11     (u, (l, i − 1)) ← normaliza((σ(u), (l, i − 1)))
12     (end, w) ← testa_e_divide((u, (l, i − 1)), x_i)
13   fim-faça
14   se w ≠ NIL então
15     crie o suffix link w' ⇢ w
16   fim-se
17   devolva (u, (l, i − 1))
18 fim
```

Algoritmo 3.9. Função *atualiza*.

```
1 Algoritmo Ukkonen (X = x_1 ··· x_n)
2 início
3   crie a árvore inicial T_1 : RAIZ --(1,∞)--> v = x_1
4   (u, (l, r)) ← (RAIZ, (1, 0))
5   para i ← 2 até n faça
6     (u, (l, r)) ← atualiza(u, (l, r))
7     (u, (l, r)) ← normaliza(u, (l, i))
8   fim-faça
9 fim
```

Algoritmo 3.10. Algoritmo Ukkonen para construção de árvores de sufixos.

a *atualiza* sem levar em consideração as chamadas encadeadas a *normaliza* e, depois, estimaremos o custo total das chamadas a esta função separadamente. O custo do algoritmo é dado pela soma dos dois valores obtidos.

Todas as operações da Função *atualiza*, a menos das chamadas a *normaliza* (que, a princípio, não consideramos), demandam tempo constante. Com efeito, a Função *testa_e_divide* requer tempo $O(|\Sigma|)$ (linhas *4* e *15*), ou seja, $O(1)$ se assumimos $|\Sigma|$ constante. Portanto, a chamada da Função *atualiza*, na i-ésima iteração do algoritmo, demanda (a menos das chamadas a *normaliza*) tempo proporcional ao número de vezes que o laço interno (linhas *5-13* do Algoritmo 3.9) é percorrido ou, equivalentemente, à quantidade de vértices visitados no caminho pela fronteira de T_{i-1} do vértice ativo até o vértice terminador (ou a RAIZ). Seja $w^i_{j'}, \ldots, w^i_{j''}$ essa seqüência de vértices visitados na i-ésima iteração do algoritmo. Temos que $|rót(w^i_{j-1})| - |rót(w^i_j)| = 1$ para $j' < j \leq j''$. Portanto, o número de vértices visitados nessa iteração é $|rót(w^i_{j'})| - |rót(w^i_{j''})| + 1$. Ao longo de todo o processo, temos um total de $\sum_{i=2}^{n}(|rót(w^i_{j'})| - |rót(w^i_{j''})| + 1)$ vértices visitados. Ocorre que, pelo Lema 3.5, temos que $|rót(w^{i+1}_{j'})| \leq |rót(w^i_{j''})| + 1$ e, portanto, $\sum_{i=2}^{n}(|rót(w^i_{j'})| - |rót(w^i_{j''})| + 1) \leq |rót(w^1_{j''})| - |rót(w^n_{j''})| + 2(n-1) < 2n$. Ou seja, o custo total das chamdas a *atualiza*, a menos das chamadas encadeadas a *normaliza* é $O(n)$.

Em seguida, precisamos contabilizar o custo de todas as chamadas a *normaliza*. Cada uma dessas chamadas consome tempo, no máximo, proporcional ao número de vezes em que o laço correspondente às linhas *7-13* do Algoritmo 3.8 é percorrido mais um valor constante, necessário à execução das linhas *3* e *6*. Portanto, o custo total da Função *normaliza* é proporcional ao número de vezes que ela é chamada mais o número total de vezes que o laço supra-mencionado é percorrido. Notamos, primeiro, que a função é invocada $O(n)$ vezes: uma vez para cada vértice visitado na Função *atualiza* (conforme disposto no parágrafo anterior) mais $n-1$ vezes no laço do Algoritmo 3.10. Para, finalmente, estimar o número de vezes que o laço em questão é percorrido, basta reparar que o seu efeito líquido é sempre suprimir um prefixo não-nulo de um fator de X representado por (l, r), o que corresponde a incrementar o valor de l de forma que ainda tenhamos $l \leq r$. Como o valor de r é incrementado apenas na linha *7* do Algoritmo 3.10, uma unidade por vez, chegando a valer, no máximo, n e o valor de l entre chamadas sucessivas a *normaliza* nunca decresce, temos que essas supressões de prefixos não-nulos podem ocorrer, no máximo, n vezes. Portanto, o custo total do laço é $O(n)$. ∎

3.2 ÁRVORES DE AFIXOS

As árvores de afixos introduzidas por Stoye [Sto95] representam uma ampliação na representatividade das árvores de sufixos uma vez que são estruturas *duais*, ou seja, são estruturas capazes de representar, simultaneamente, os fatores e os fatores reversos de uma mesma cadeia.

3.2.1 Motivação e Preliminares

A idéia fundamental das árvores de afixos tem sua origem em uma certa dualidade curiosamente já encontrada nas árvores de sufixos e estruturas equivalentes. Nesta seção, revelamos essa dualidade ao mesmo tempo em que estabelecemos a terminologia e definições elementares utilizadas no porvir.

3.2.1.1 ***Suffix Links*** As arestas auxiliares conhecidas como *suffix links* utilizadas nos processos de construção das árvores de sufixos (Seções 3.1.2.1 e 3.1.2.2) podem ser definidas de maneira geral da seguinte forma:

Definição 3.8 (*Suffix link*) *Seja T uma árvore-Σ^+ e $u = \overline{aY}, v = \overline{V}$ vértices de T com $Y = UV \in \Sigma^*$ tais que V corresponde ao maior sufixo de Y explicitamente representado em T. A aresta $u \overset{aU}{\dashrightarrow} v$ é um* suffix link *de T.*

Um *suffix link* é dito *atômico* se ele é da forma $\overline{aY} \overset{a}{\dashrightarrow} \overline{Y}$. Os suffix links do Algoritmos McCreight e Ukkonen são todos atômicos. Em geral, temos o seguinte:

Teorema 3.6 *Se $u \dashrightarrow v$ é um* suffix link *da árvore de sufixos compacta* $\textsc{cst}(X)$ *tal que u é um vértice interno, então esse* suffix link *é atômico.*

Prova Se u é um vértice interno de $\textsc{cst}(X)$, então $u = \overline{aY}$ para algum fator aninhado que deriva à direita aY de X. Mas então Y também é um fator aninhado de X que deriva à direita e, portanto, $v = \overline{Y}$ é um vértice interno (explícito) de $\textsc{cst}(X)$. ∎

O *suffix link* originado na folha $v = \overline{aY}$ de $\textsc{cst}(X)$ é atômico a menos que o sufixo Y seja aninhado mas não derive à direita. Portanto, os *suffix links* de $\textsc{cst}(X = x_1 \cdots x_n)$ tal que $x_n \neq x_i, \forall 1 \leq i < n$ (em particular, $x_n = \$$) são todos atômicos. os *suffix links* de $\textsc{ast}(X)$ são todos, igualmente, atômicos pois todos os seus vértices são explícitos.

Definição 3.9 (Árvore de *suffix links*) *A árvore de* suffix links *T^R de uma árvore-Σ^+ T é a a árvore cujos vértices são os mesmos vértices de T e as arestas são os* suffix links *de T no sentido inverso, ou seja, o* suffix link *$u \overset{Y}{\dashrightarrow} v$ origina a aresta $v \xrightarrow{Y^R} u$.*

A Definição 3.9 acima encontra-se ilustrada na Figura 3.7. É fácil constatar que T^R é, de fato, uma árvore. Com efeito, cada vértice de T origina exatamente um *suffix link* que aponta para um outro vértice que será o seu pai na árvore de *suffix links*. Além disso, cada seqüência de emphsuffix links em T converge para $\textsc{raiz} = \overline{\varepsilon}$, o que faz do grafo em questão conexo. Por fim, reparamos que a árvore de *suffix links* não pode conter ciclos na

medida em que os comprimentos das cadeias representadas pelos vértices consecutivos em um "caminho" de *suffix links* formam uma seqüência estritamente decrescente.

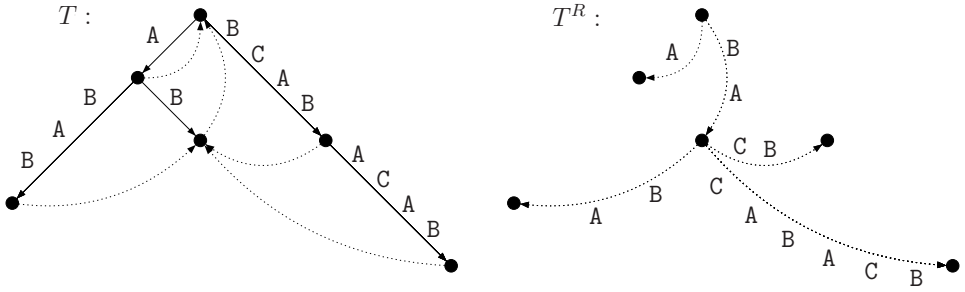


Figura 3.7. À esquerda, uma árvore-Σ^+ T com os seus *suffix links* representados. À direita, a árvore de *suffix links* correspondente T^R. Repare que a árvore de *suffix links* não é, necessariamente, uma árvore-Σ^+.

Teorema 3.7 *Seja T uma árvore-Σ^+. $\overline{Y} \in T \iff \overline{Y^R} \in T^R$.*

Prova $u = \overline{Y} \in T \iff$ existe um caminho de *suffix links* $u \dashrightarrow \cdots \dashrightarrow \text{RAIZ} = \overline{\varepsilon}$ em T $\iff$ existe um caminho $\text{RAIZ} \rightsquigarrow u$ em T^R cujo rótulo é $Y^R \iff u$ é um locus de Y^R em T^R. ∎

3.2.1.2 Dualidade Em geral, duas estruturas são ditas duais entre si se, para cada cadeia representada em uma estrutura, a cadeia reversa correspondente está representada na outra e vice-versa. Em particular, estamos interessados em estruturas que apresentem um aspecto de "auto-dualidade", ou seja, estruturas que sejam os duais delas próprias, ou ainda, estruturas que representam, simultaneamente, um determinado conjunto de cadeias e os seus reversos ($Y \in \mathit{cadeias}(T) \iff Y^R \in \mathit{cadeias}(T)$).

Definição 3.10 (Dualidade entre árvores-Σ^+) *Duas árvores-Σ^+ T e T' são ditas* duais *entre si se, para cada vértice $\overline{Y} \in T$, existe um vértice $\overline{Y^R} \in T'$ e vice-versa.*

É conveniente generalizar a definição de *locus* e refinar a sua notação, passando a denotar $u = \overrightarrow{Y} = \overline{Y}$ se $Y \in \Sigma^*$ é o rótulo do caminho $\text{RAIZ} = \overrightarrow{\varepsilon} \rightsquigarrow u$ e $u = \overleftarrow{Y} = \overline{Y^R}$ se Y^R é o rótulo do caminho $\text{RAIZ} = \overleftarrow{\varepsilon} \rightsquigarrow u$. Se $u = \overleftarrow{Y}$ então dizemos que u é um *locus reverso* de Y.

Teorema 3.8 (Dualidade das *suffix tries*) $\text{AST}(X)^R = \text{AST}(X^R)$.*

Prova Existe uma aresta $\overleftarrow{Y} \xrightarrow{a} \overleftarrow{aY}$ em $\text{AST}(X)^R \iff$ existe um *suffix link* $\overrightarrow{aY} \dashrightarrow \overrightarrow{Y}$ em $\text{AST}(X) \iff$ existem vértices $\overrightarrow{Y}$ e $\overrightarrow{aY}$ em $\text{AST}(X) \iff$ existem vértices $\overleftarrow{Y}$ e $\overleftarrow{aY}$ em $\text{AST}(X^R) \iff$ existe uma aresta $\overleftarrow{Y} \xrightarrow{a} \overleftarrow{aY}$ em $\text{AST}(X^R)$. ∎

*Uma árvore-Σ^+ T tal que $\mathit{cadeias}(T) = \mathscr{F}(X^R)$ é comumente denominada uma *árvore de prefixos reversos de X*. Portanto, $\text{AST}(X^R)$ é conhecida como "a" árvore de prefixos reversos atômica (*reverse prefix trie*) de X e $\text{CST}(X^R)$ como "a" árvore de prefixos reversos compacta de X.

O Teorema 3.8 acima nos diz que as árvores de sufixos atômicas aumentadas com os seus *suffix links* são estruturas inerentemente duais. Com efeito, as cadeias representadas por $\textsc{ast}(X)$ correspondem aos fatores de X. Logo, as cadeias reversas respectivas correspondem aos reversos dos fatores de X, ou seja, os fatores de X^R, que são exatamente as cadeias representadas por $\textsc{ast}(X^R)$.

Teorema 3.9 (Dualidade fraca das árvores de sufixos compactas) *A árvore de* suffix links $\textsc{cst}(X)^R$ *da árvore de sufixos compacta* $\textsc{cst}(X)$ *é uma árvore-*Σ^+ *que representa um subconjunto das cadeias representadas por* $\textsc{cst}(X^R)$. *Além disso,* $(\textsc{cst}(X)^R)^R = \textsc{cst}(X)$.

Prova Suponha que existam duas arestas $\overleftarrow{Y} \xrightarrow{(Ua)^R} \overleftarrow{UaY}$ e $\overleftarrow{Y} \xrightarrow{(Va)^R} \overleftarrow{VaY}$ em $\textsc{cst}(X)^R$ com $U \neq V$ ambas não-nulas. $\overrightarrow{aY}$ não é um vértice de $\textsc{cst}(X)$ pois, do contrário, $\overrightarrow{UaY} \overset{Ua}{\dashrightarrow} \overrightarrow{Y}$ e $\overrightarrow{VaY} \overset{Va}{\dashrightarrow} \overrightarrow{Y}$ não seriam *suffix links* dessa árvore. Temos 2 casos:

i) Se $\overrightarrow{UaY}$ (ou, igualmente, $\overrightarrow{VaY}$) é um vértice interno de $\textsc{cst}(X)$ então UaY (respec. VaY) deriva à direita em X e igualmente o faz aY. Logo, $\overrightarrow{aY}$ dever ser um vértice interno explícito de $\textsc{cst}(X)$.

ii) Se $\overrightarrow{UaY}$ e $\overrightarrow{VaY}$ são ambas folhas de X então UaY e VaY são ambos sufixos de X. Logo, podemos supor, sem perda de generalidade, que $VaY \sqsupset UaY$. Mas então não pode existir o *suffix link* $\overrightarrow{UaY} \overset{Ua}{\dashrightarrow} \overrightarrow{Y}$ pois $|VaY| > |Y|$.

Das contradições acima, temos que o vértice $\overleftarrow{Y} \in \textsc{cst}(X)^R$ não pode originar duas arestas cujos rótulos iniciam-se por um mesmo a $\therefore$ $\textsc{cst}(X)^R$ é uma árvore-Σ^+.

A seqüência de *suffix links* $\overrightarrow{Y} \dashrightarrow \cdots \dashrightarrow \textsc{raiz} = \overrightarrow{\varepsilon}$ em $\textsc{cst}(X)$ origina um caminho $\textsc{raiz} = \overleftarrow{\varepsilon} \rightsquigarrow \overleftarrow{Y}$ em $\textsc{cst}(X)^R$ cujo rótulo é Y^R o qual, obviamente, também está representado em $\textsc{cst}(X^R)$

Finalmente, para mostrar que $(\textsc{cst}(X)^R)^R = \textsc{cst}(X)$, observamos que, da primeira parte do enunciado, $\textsc{cst}(X)^R$ é uma árvore-Σ^+. Logo, $(\textsc{cst}(X)^R)^R$ está definida e, mais ainda, os seus vértices são os mesmos de $\textsc{cst}(X)$. Existe um *suffix link* $\overleftarrow{YU} \overset{U^R}{\dashrightarrow} \overleftarrow{Y}$ em $\textsc{cst}(X)^R \iff$ não existe um sufixo $V^R \sqsupset (YU)^R$ t.q. $\overleftarrow{V} \in \textsc{cst}(X)^R$ e $|(YU)^R| > |V^R| > |Y^R| \iff$ não existe um prefixo $V \sqsubset YU$ t.q. $\overrightarrow{V} \in \textsc{cst}(X)$ e $|YU| > |V| > |Y| \iff$ existe uma aresta $\overrightarrow{Y} \xrightarrow{U} \overrightarrow{YU}$ em $\textsc{cst}(X)$. ∎

O Teorema 3.9 acima demonstra que, embora não sejam estruturas duais no sentido estrito, as árvores de sufixos compactas ainda apresentam um certo grau de dualidade. A pergunta que surge naturalmente é se seria possível estender as árvores de sufixos compactas, acrescentando-lhes alguns vértices e algumas arestas, de forma a obter a mesma dualidade intrínseca das *suffix tries* todavia sem comprometer a complexidade linear do espaço requerido. A resposta é sim e as estruturas resultantes recebem o nome de árvores de afixos. Antes de passar para a definição destas, entretanto, é necessário caracterizar o tipo de estrutura que elas representam.

3.2.1.3 União reversa

Definição 3.11 (Extensão de árvores-Σ^+) *Dadas duas árvores-Σ^+ S e P, a extensão de S por P (ou P-extensão de S), denotada por S_P, é a árvore-Σ^+ obtida a partir de S se tornamos explícitos os seus vértices implícitos $\overrightarrow{Y} = (u, V)$ tais que $\overleftarrow{Y}$ é um vértice explícito de P.*

A Figura 3.8 ilustra a Definição 3.11 acima. Observe que a extensão de uma árvore-Σ^+ não altera o conjunto de cadeias por ela representadas. Repare também que, como conseqüência imediata da definição, se T e T' são árvores-Σ^+ duais entre si, então $T_{T'} = T$ e, da mesma forma, $T'_T = T'$.

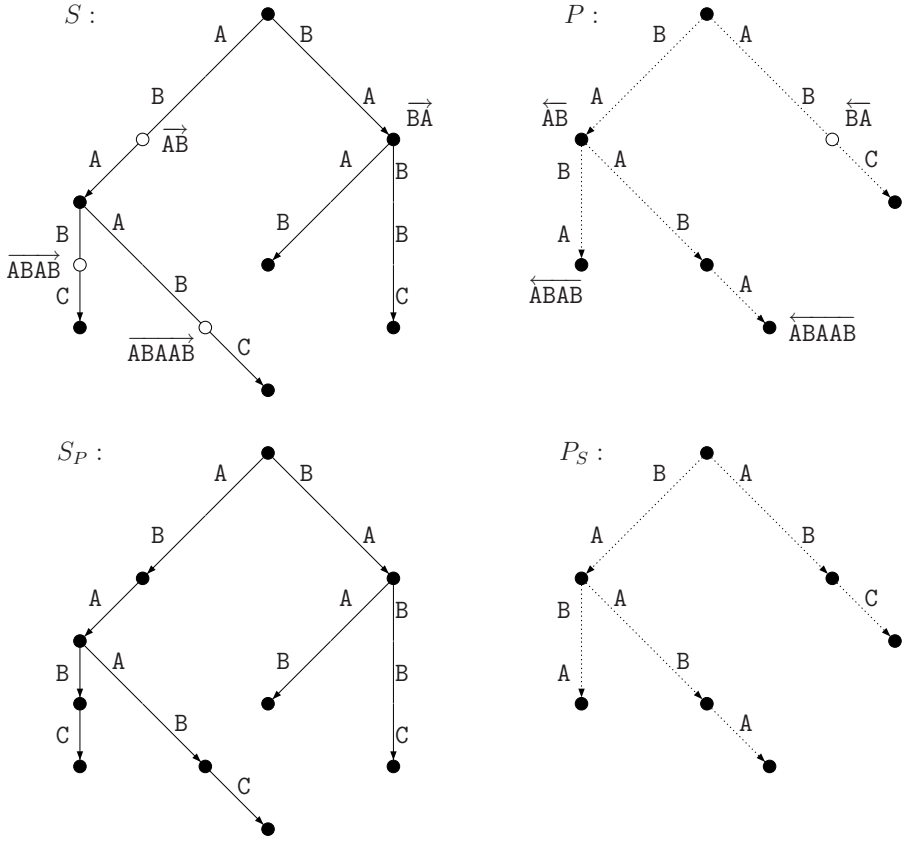


Figura 3.8. Extensão de árvores-Σ^*. Acima, as árvores originais S e P. Abaixo, as suas extensões mútuas S_P e P_S. Os vértices implícitos feitos explícitos estão indicados na parte superior pelos círculos não-preeenchidos.

Definição 3.12 (União reversa de árvores-Σ^+) *A união reversa de duas árvores-Σ^+ S e P, denotada por $S\tilde{\cup}P$, é o multigrafo* direcionado e enraizado definido da seguinte maneira:*

*Na realidade, um bi-grafo, ou seja, um grafo tal que, para cada par de vértices (u, v), existem, no máximo, duas arestas $u \xrightarrow{W} v$.

i) O conjunto de vértices de $S\tilde{\cup}P$ corresponde à reunião dos conjuntos de vértices de S_P e P_S sendo que cada vértice $\overrightarrow{Y}$ de S_P é identificado com o vértice $\overleftarrow{Y}$ de P_S, se este existir, originando o vértice $\overrightarrow{Y} = \overleftarrow{Y^R}$ e reciprocamente. Em símbolos:

$$\textit{vértices}(S\tilde{\cup}P) = \textit{vértices}(S_P)\ \tilde{\cup}\ \textit{vértices}(P_S), \textit{onde}$$

$$\begin{aligned} A\ \tilde{\cup}\ B =& \{\overrightarrow{Y} = \overleftarrow{Y^R} \mid \overrightarrow{Y} \in A\ \wedge\ \overleftarrow{Y} \in B\} \\ & \cup \{\overrightarrow{Y} \mid \overrightarrow{Y} \in A\ \wedge\ \overleftarrow{Y} \notin B\} \\ & \cup \{\overleftarrow{Y^R} \mid \overleftarrow{Y} \in B\ \wedge\ \overrightarrow{Y} \notin A\} \end{aligned}$$

ii) O conjunto de arestas de $S\tilde{\cup}P$ corresponde à reunião dos conjuntos de arestas de S_P e P_S

$$\textit{arestas}(S\tilde{\cup}P) = \textit{arestas}(S_P)\ \cup\ \textit{arestas}(P_S).$$

A figura Figura 3.9 ilustra a Definição 3.12 acima. Observe que a união reversa $S\tilde{\cup}P$ contém as sub-árvores S_P e P_S.

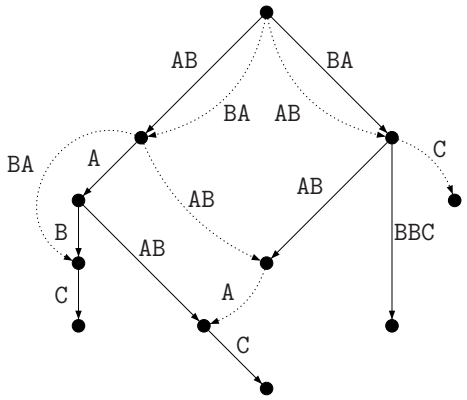


Figura 3.9. União reversa $S\tilde{\cup}P$ das árvores-Σ^+ S e P apresentadas na Figura 3.8.

Definição 3.13 (União reversa dual) *A união reversa $S\tilde{\cup}P$ é dita* dual *se cada um dos seus vértices é, ao mesmo tempo, um vértice da sub-árvore S_P e da sub-árvore P_S.*

Teorema 3.10 *A união reversa $S\tilde{\cup}P$ é dual se, e somente se, as sub-árvores S_P e P_S são duais.*

Prova $S\tilde{\cup}P$ é dual $\iff \forall u = \overrightarrow{Y} = \overleftarrow{Y^R} \in \textit{vértices}(S\tilde{\cup}P),\ u \in \textit{vértices}(S_P)\ \wedge\ u \in \textit{vértices}(P_S) \iff \overrightarrow{Y} \in \textit{vértices}(S_P)$ se, e somente se, $\overleftarrow{Y} \in \textit{vértices}(P_S) \iff S_P$ e P_S são duais. ∎

Teorema 3.11 *Se a união reversa $S\tilde{\cup}P$ é dual então P_S é a árvore de* suffix links *de S_P e reciprocamente.*

Prova Sejam os vértices $\overrightarrow{aY}, \overrightarrow{V} \in S_P$ com $a \in \Sigma$ e $Y = UV \in \Sigma^*$ tais que V é o maior sufixo de Y explicitamente representado por um vértice $\overrightarrow{V} \in S_P$, *i.e.*, $\overrightarrow{aY} \overset{aU}{\dashrightarrow} \overrightarrow{V}$ é um *suffix link* de S_P. Vamos mostrar que a sub-árvore P_S contém uma aresta $\overrightarrow{V} \xrightarrow{(aU)^R} \overrightarrow{aY}$. Com efeito, como $S\tilde{\cup}P$ é dual, então todos os vértices de S_P, em particular $\overrightarrow{aY} = \overleftarrow{(aY)^R}$ e $\overrightarrow{V} = \overleftarrow{V^R}$ são também vértices de P_S. Além disso, como $V^R \sqsubset (aY)^R$, então $\overleftarrow{V^R}$ é ancestral de $\overleftarrow{(aY)^R}$ na sub-árvore P_S. Suponha agora que existe um vértice intermediário no caminho $\overleftarrow{V^R} \rightsquigarrow \overleftarrow{(aY)^R}$ em P_S, ou seja, existe um $W^R \sqsubset aY^R$ t.q. $\overleftarrow{W^R} \in P_S$ e $|V^R| < |W^R| < |aY^R|$. Mas então, como $S\tilde{\cup}P$ é dual, existe um $W \sqsupset aY$ t.q. $\overrightarrow{W} \in S_P$ e $|V| < |W| < |aY|$ e, portanto, não pode existir o *suffix link* $\overrightarrow{aY} \overset{U}{\dashrightarrow} \overrightarrow{V}$. Da contradição, temos que $\overleftarrow{V^R}$ é o pai de $\overleftarrow{(aY)^R}$ em P_S. Adicionalmente, como não há vértices de P_S que não estejam em S_P, cada aresta $u \xrightarrow{Z} v$ em P_S representa, por um argumento inverso, algum *suffix link* de S_P $\therefore$ $P_S = (S_P)^R$

Devido à simetria de $S\tilde{\cup}P$, $S_P = (P_S)^R$ pode ser demonstrado de maneira análoga. ∎

Em outras palavras, o Teorema 3.11 acima nos diz que, se a árvore dual T' de uma árvore-Σ^+ T existe, então ela é, precisamente, a árvore de *suffix links* de T. Com efeito, T e T' são duais $\overset{\text{Def. 3.11}}{\Longrightarrow}$ $T_{T'} = T$ e $T'_T = T'$ são duais $\overset{\text{Teo. 3.10}}{\Longrightarrow}$ $T\tilde{\cup}T'$ é dual $\overset{\text{Teo. 3.11}}{\Longrightarrow}$ $T'_T = T' = (T = T_{T'})^R$.

3.2.2 Definição e Propriedades Elementares

Definição 3.14 (Árvore de afixos) *Uma* árvore de afixos *de uma cadeia $X \in \Sigma^*$ é dada pela união reversa de uma árvore de sufixos e uma árvore de prefixos reversos de X.*

A Figura 3.10 ilustra a Definição 3.14 acima. Repare que uma árvore de afixos corresponde a uma espécie de *fusão* entre uma árvore de sufixos de uma cadeia e uma árvore de sufixos do seu reverso, o que já representa um forte indício da característica dual dessas estruturas.

A *árvore de afixos atômica* de X, $\text{AAT}(X)$, é definida como sendo a união reversa da árvore de sufixos atômica de X e da árvore de prefixos reversos atômica de X. Em símbolos: $\text{AAT}(X) = \text{AST}(X)\tilde{\cup}\text{AST}(X^R)$. Repare que, em conseqüência do Teorema 3.8, a árvore de afixos atômica $\text{AAT}(X)$ corresponde, exatamente, à *suffix trie* $\text{AST}(X)$ aumentada com os seus *suffix links* no sentido inverso (Figura 3.10(a)). De maneira análoga, definimos a *árvore de afixos compacta* de X, $\text{CAT}(X)$, como sendo a união reversa da árvore de sufixos compacta de X com a sua árvore de prefixos reversos compacta, *i.e.*, $\text{CAT}(X) = \text{CST}(X)\tilde{\cup}\text{CST}(X^R)$ (Figura 3.10(c)).

Teorema 3.12 (Dualidade das árvores de afixos) *As árvores de afixos são duais.*

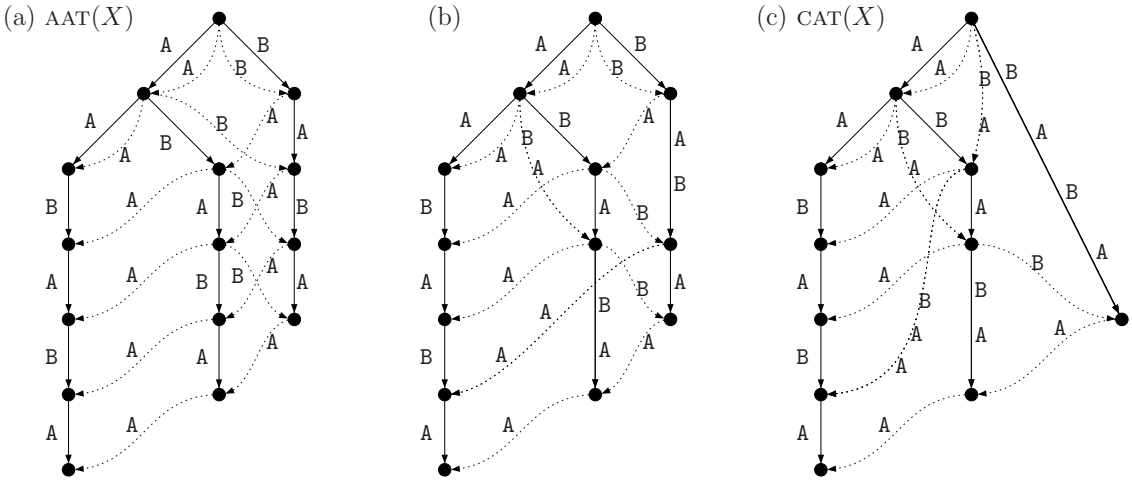


Figura 3.10. Diferentes árvores de afixos de $X =$ `AABABA`. À esquerda, a versão atômica, à direita, a versão compacta e, ao centro, uma versão intermediária. (Exemplo retirado de [Sto00])

Prova Seja S uma árvore de sufixos de $X \in \Sigma^*$ e P uma árvore de prefixos reversos da mesma cadeia. A prova consiste em mostrar que S_P e P_S são duais, *i.e.*,

$$\overrightarrow{Y} \in S_P \iff \overleftarrow{Y} \in P_S.$$

($\Longrightarrow$) Seja $\overrightarrow{Y}$ um vértice explícito de S_P. Então $Y \in \mathit{cadeias}(S_P)$ e, conseqüentemente, $Y \in \mathit{cadeias}(S)$. Logo, $Y^R \in \mathit{cadeias}(P)$ e, portanto, $\overleftarrow{Y}$ é um vértice de P quer seja explícito ou implícito. Se $\overleftarrow{Y}$ é explícito em P, então ele também o é em P_S. Se $\overleftarrow{Y}$ é implícito em P então $\overrightarrow{Y}$ é explícito em S ou não seria explícito em S_P. Mas então $\overleftarrow{Y}$ é feito explícito em P_S pela S-extensão. ($\Longleftarrow$) O recíproco demonstra-se de maneira análoga. ∎

O Teorema 3.12 acima atesta que as árvores de afixos possuem, de fato, a característica dual a que vínhamos nos referindo, ou seja, a capacidade de representar, simultaneamente, os fatores de uma cadeia X e os seus reversos. Como bons índices, entretanto, elas ainda devem demonstrar-se estruturas eficientes (*i.e.*, lineares) sob o aspecto do espaço requerido. Esse fato é verificado no teorema a seguir.

Teorema 3.13 *Seja a cadeia* $X = x_1 \cdots x_n \in \Sigma^*$. $\textsc{cat}(X)$ *possui, no máximo,* $N = 4n - 4$ *vértices e* $2N - 2$ *arestas.*

Prova Sabemos, do Teorema 3.3, que $\textsc{cst}(X)$ possui, no máximo $2n - 1$ vértices. O mesmo vale, obviamente, para $\textsc{cst}(X^R)$. Portanto, como o número de vértices da união reversa $S\tilde{\cup}P$ é, no máximo, a soma dos totais de vértices de S e P, temos um limite superior inicial $4n - 2$. Ocorre que pelo menos dois vértices de $\textsc{cst}(X)_{\textsc{cst}(X^R)}$ são identificados com vértices de $\textsc{cst}(X^R)_{\textsc{cst}(X)}$, quais sejam $\overrightarrow{\varepsilon}$ e $\overrightarrow{X}$, donde concluímos o limite superior teórico $N = 4n - 4$.

Como a árvore de afixos é dual, então as sub-árvores $\textsc{cst}(X)_{\textsc{cst}(X^R)}$ e $\textsc{cst}(X^R)_{\textsc{cst}(X)}$ possuem o mesmo número de vértices e, conseqüentemente, de arestas. Se ambas possuem N vértices, então, por serem árvores possuem $N-1$ arestas e, logo, a união reversa possui $2N-2$ arestas. ∎

Em sua dissertação [Sto95], Stoye não foi capaz de fornecer um exemplo no qual os limites superiores do Teorema 3.13 se verificam. Entretanto, ele mostrou que para $\Sigma = \{a_1, \ldots, a_s\}$ e $X_{(s)} = a_1(a_2 \cdots a_{s-1})^s a_s$ ($n = |X_{(s)}| = s^2 - 2s + 2$), o desvio relativo em relação ao limite superior para o número de vértices torna-se arbitrariamente pequeno à medida que o comprimento da cadeia aumenta. Mais precisamente, $\lim_{s\to\infty} \frac{|\textsc{cat}(X_{(s)})|-4n}{4n} = 0$.

CAPÍTULO 4

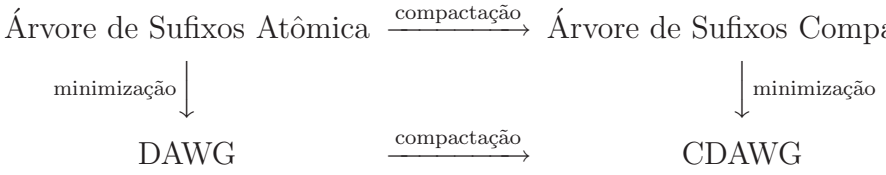
DAWGs

Tal é a vantagem de uma linguagem bem construída
que a sua notação simplificada freqüentemente
torna-se a fonte de profundas teorias.

— PIERRE-SIMON DE LAPLACE

Neste capítulo, estudaremos a família de índices completos originados na Teoria do Autômatos Finitos e que tomam a forma de *grafos direcionados acíclicos de fatores*, ou *DAWGs* (*Directed Acyclic Word Graphs*). Os DAWGs foram introduzidos por Blumer *et al.* [BBH+85], juntamente com o primeiro algoritmo linear e *on-line* para sua construção. O DAWG proposto por Blumer *et al.* é um AFD capaz de reconhecer a linguagem constituída pelos fatores de uma dada cadeia. Crochemore [Cro86] propôs uma ligeira variação dos DAWGs originando o autômato mínimo a reconhecer a linguagem formada pelos sufixos de uma cadeia: o *autômato de sufixos*.

Os DAWGs correspondem, essencialmente, a uma minimização de árvores de sufixos atômicas equivalentes, que pode ser efetuada através da identificação das subárvores isomorfas [CR94]. Não obstante a maior eficiência em termos de espaço já alcançada pelos DAWGs (estruturas lineares) em comparação com as suas equivalentes *suffix tries* (estruturas quadráticas), Blumer *et al.* [BBHM87] propuseram estruturas ainda mais sucintas—os *DAWGs Compactos* ou *CDAWGs* (*Compact Directed Acyclic Word Graphs*)—que podem ser, intuitivamente, encaradas como as estruturas derivadas dos DAWGs mediante o processo de compactação de caminhos semelhante ao que são submetidos as árvores de sufixos atômicas para originarem as árvores de sufixos compactas. Em geral, temos o seguinte diagrama:



No mesmo trabalho, Blumer *et al.* propuseram o primeiro algoritmo capaz de construir os CDAWGs em tempo linear todavia de forma indireta, ou seja, construindo primeiro o DAWG correspondente para então compactá-lo. Crochemore e Vérin [CV97b, CV97a] forneceram o primeiro algoritmo linear para a construção dos CDAWGs de maneira direta, baseados no Algoritmo de McCreight [McC76] para a construção de árvores de sufixos. Recentemente, Inenaga *et al.* [IHS+01b], baseados no trabalho de Ukkonen [Ukk95], desenvolveram o primeiro algoritmo linear e *on-line* para a construção de CDAWGs.

Os DAWGs, a exemplo das suas correspondentes *suffix tries*, apresentam uma certa característica dual. Mais precisamente, o DAWG associado a uma certa cadeia, acrescido

de algumas arestas auxiliares, contém a árvore de sufixos compacta do reverso dessa cadeia (observando-se, apenas, uma pequena restrição sobre a cadeia em questão) [CR94]. Blumer *et al.* [BBHM87], observando que os vértices de um CDAWG são invariantes mediante reversão da cadeia representada, propuseram uma ligeira extensão dessa estrutura, originando uma estrutura dual conhecida como *CDAWG Simétrico* ou *SCDAWG* (*Symetric Compact Direted Acyclic Word Graph*). Inenaga *et al.* [IHS+01a] estenderam o algoritmo supra-mencionado destinado à construção de CDAWGs para contemplar a construção *on-line* em tempo linear dos SCDAWGs.

4.1 DAWGs

4.1.1 Preliminares: o Teorema de Myhill-Nerode

Definição 4.1 (Invariância de uma relação de equivalência) *Uma relação de equivalência R sobre um monóide** $(Q, \cdot, e)$ *é dita* invariante à direita *(*esquerda*) se, dados quaisquer $x, y \in Q$, $xRy \implies xzRyz,\ \forall z \in Q$ (respec., $xRy \implies zxRzy,\ \forall z \in Q$).*

Seja $A = (Q, \Sigma, \delta, q_0, F)$ um AFD†. Baseados em A, podemos definir, sobre o monóide Σ^*, uma relação de equivalência R_A dada por

$$XR_AY \iff \delta(q_0, X) = \delta(q_o, Y).^{\ddagger} \tag{4.1}$$

Ou seja, duas cadeias são equivalentes se, quando tomadas como entrada pelo autômato A, conduzem-no ao mesmo estado. Repare que R_A é bem-definida pois A é determinístico, *i.e.*, dado $X \in \Sigma^*$, $\exists! q \in Q$ t.q. $q = \delta(q_0, X)$. Além disso, R_A é, de fato, uma relação de equivalência uma vez que o "=" de (4.1) o é. Cada estado $q \in Q$ define uma classe de equivalência dada por $\{X \in \Sigma^* \mid \delta(q_0, X) = q\}$.

Proposição 4.1 *A relação de equivalência R_A, definida como em (4.1), é invariante à direita.*

Prova Sejam as cadeias $X, Y \in \Sigma^*$ tais que XR_AY e $Z \in \Sigma^*$ uma cadeia arbitrária. Por definição de R_A, temos

$$XZ\, R_A\, YZ \iff \delta(q_0, XZ) = \delta(q_0, YZ) \iff \delta(\delta(q_0, X), Z) = \delta(\delta(q_0, Y), Z).$$

Como XR_AY, então $\delta(q_0, X) = \delta(q_0, X) \ \therefore\ XZ\, R_A\, YZ$. ∎

Teorema 4.1 (Myhill-Nerode, [AHU74]) *As seguintes proposições são equivalentes:*

i) O conjunto $\mathscr{L} \subset \Sigma^*$ é aceito por algum AFD.

*Vide nota de rodapé à página 10.

†A quíntupla $A = (Q, \Sigma, \delta, q_0, F)$ define um autômato finito no qual Q corresponde ao conjunto de estados, Σ corresponde ao alfabeto de entrada definido como na Definição 1.1, δ corresponde à função de transição de estados, q_0 denota o estado inicial e F corresponde ao conjunto de estados finais.

‡$\delta(q, X = x_1 \cdots x_n) = \begin{cases} q, & \text{se } n = 0 \\ \delta(\delta(q, X_{..n-1}), x_n), & \text{se } n > 0. \end{cases}$

ii) $\mathscr{L}$ *é obtido a partir da reunião de classes de equivalência de uma relação invariante à direita de índice finito*[*].

iii) Seja a relação de equivalência $R_{\mathscr{L}}$ *definida por*

$$X R_{\mathscr{L}} Y \overset{def.}{\iff} \forall Z \in \Sigma^*, XZ \in \mathscr{L} \Leftrightarrow YZ \in \mathscr{L}.$$

Então $R_{\mathscr{L}}$ *tem índice finito.*

Prova (i) $\Longrightarrow$ (ii). Seja $A = (Q, \Sigma, \delta, q_0, F)$ uma AFD que reconhece $\mathscr{L}$ e seja R_A a relação de equivalência invariante à direita definida como em (4.1). $\mathscr{L}$ é obtido a partir da reunião das classes de equivalência definidas pelos estados finais em F. R_A possui, obviamente, índice finito pois A é finito.

(ii) $\Longrightarrow$ (iii). Seja E uma relação uma relação de equivalência de índice finito invariante à direita t.q. $\mathscr{L}$ possa ser obtido a partir da união de classes de equivalência de E. Para mostrar que $R_{\mathscr{L}}$ tem índice finito, vamos mostrar que E *refina* $R_{\mathscr{L}}$, ou seja, que cada classe de equivalência de E está completamente contida em alguma classe de equivalência de $R_{\mathscr{L}}$ e, portanto, como E e $R_{\mathscr{L}}$ estão definidos ambos sobre o mesmo Σ^*, $R_{\mathscr{L}}$ não pode possuir mais classes de equivalência do que E. Com efeito, se XEY então, como E é invariante à direita, $XZ\,EYZ$, $\forall Z \in \Sigma^*$. Como $\mathscr{L}$ é formado a partir de classes de equivalência de E, então, para um Z qualquer, se XZ pertence a uma dessas classes e, portanto, pertence também a $\mathscr{L}$, YZ também pretence, visto que $XZ\,EYZ$. Pela mesma razão, se XZ não pertence a uma dessas classes e, portanto, não pertence a $\mathscr{L}$, então YZ também não pertence. Ou seja, $XZ \in \mathscr{L} \Leftrightarrow YZ \in \mathscr{L} \therefore XEY \implies X R_{\mathscr{L}} Y$.

(iii) $\Longrightarrow$ (i). A prova é por construção. Primeiramente, precisamos mostrar que $R_{\mathscr{L}}$ é invariante à direita, *i.e.*, $\forall W \in \Sigma^*$, $X R_{\mathscr{L}} Y \implies XW\, R_{\mathscr{L}}\, YW$, ou seja, $\forall Z \in \Sigma^*$, $XWZ \in \mathscr{L} \Leftrightarrow YWZ \in \mathscr{L}$. Com efeito, tome um W arbitrário. Como $X R_{\mathscr{L}} Y$, então $XV \in \mathscr{L} \Leftrightarrow YV \in \mathscr{L}$, $\forall V \in \Sigma^*$, em particular para $V = WZ$, qualquer que seja $Z \therefore XW\, R_{\mathscr{L}}\, YW$.

Seja então $Q' = \bigcup_{X \in \Sigma^*} [X]$, onde $[X]$ denota a classe de equivalência de X com respeito a $R_{\mathscr{L}}$. Defina a função de transição $\delta'([X], a) = [Xa]$. δ' é bem-definida pois $R_{\mathscr{L}}$ é invariante à direita. De fato, $X R_{\mathscr{L}} Y, X \neq Y \implies Xa R_{\mathscr{L}} Ya \implies [Xa] = [Ya]$. Sejam ainda $q_0' = [\varepsilon]$ e $F' = \bigcup_{X \in \mathscr{L}} [X]$. Então o AFD $A' = (Q', \Sigma, \delta', q_0', F')$ reconhece $\mathscr{L}$, uma vez que $\delta'([\varepsilon], X) = [X]$ e, portanto, $X \in \mathscr{L} \iff [X] \in F$. ∎

Teorema 4.2 *O autômato mínimo*[†] *a reconhecer uma dada linguagem* $\mathscr{L} \subset \Sigma^*$ *é, a menos de isomorfismo, o autômato* A' *da demonstração do Teorema 4.1.*

Prova Foi visto, na demonstração do Teorema 4.1, que se $\mathscr{L}$ é reconhecida por algum AFD $A = (Q, \Sigma, \delta, q_0, F)$, então A induz uma relação de equivalência natural, R_A, invariante à direita e de índice finito tal que R_A refina a relação $R_{\mathscr{L}}$, ou seja, A possui pelo

[*] O *índice* de uma relação de equivalência corresponde à quantidade de classes de equivalência que ela define.

[†] Com menor número de estados.

menos tantos estados (classes de equivalência de R_A) quantos forem os estados de A' (classes de equivalência de $R_{\mathscr{L}}$). Se, todavia, tivermos $|Q| = |Q'|$, definimos a aplicação

$$\begin{aligned} \psi : \quad & Q \to Q' \\ & q = \delta(q_0, X) \mapsto \psi(q) = q' = \delta'(q'_0, X) \end{aligned}$$

e afirmamos que ψ é um isomorfismo. Primeiramente, mostramos que ψ é bem-definida. Com efeito, para qualquer $q \in Q$, $\exists X \in \Sigma^*$ t.q. $q = \delta(q_0, X)$ pois, caso contrário, q poderia ser "retirado" de Q, originando um autômato menor. Mais ainda, se $X \neq Y$ são duas cadeias tais que $q = \delta(q_0, X) = \delta(q_0, Y)$, então, da segunda parte da demonstração do Teorema 4.1, temos $XR_AY \implies XR_{\mathscr{L}}Y \implies \delta'(q'_0, X) = \delta'(q'_0, Y)$. ψ é uma bijeção em conseqüência do fato que R_A refina $R_{\mathscr{L}}$ e $|Q| = |Q'|$. Finalmente, ψ é um homomorfismo pois $\delta(q, a) = \delta(\delta(q_0, X), a) = \delta(q_0, Xa) \implies \delta'(q', a) = \delta'(\delta'(q'_0, X), a) = \delta'(q'_0, Xa) = \psi(\delta(q_0, Xa))$. ∎

4.1.2 Definição e Propriedades Elementares

Definição 4.2 (Términos) *Dada uma cadeia $X = x_1 \cdots x_n \in \Sigma^*$, o* conjunto de términos *da cadeia não-nula $Y \in \Sigma^+$ em X é dado por*

$$\mathscr{T}_X(Y) = \{i \mid Y = X_{i-|Y|+1..i}\}.$$

Também, por definição, $\mathscr{T}_X(\varepsilon) = \{0, 1, \ldots, n\}$.

Como exemplo da Definição 4.2 acima, seja $X = \texttt{AABBABC}$. Então temos $\mathscr{T}_X(\texttt{A}) = \{1, 2, 5\}$, $\mathscr{T}_X(\texttt{AB}) = \{3, 6\}$, $\mathscr{T}_X(\texttt{AABB}) = \{4\}$ e $\mathscr{T}_X(\texttt{ABA}) = \varnothing$ (caso degenerado).

Definição 4.3 (Cadeias término-equivalentes) *Dizemos que cadeias $Y, Z \in \Sigma^*$ são* término-equivalentes *em $X \in \Sigma^*$ e denotamos $Y \equiv_{\mathscr{T}_X} Z$, se $\mathscr{T}_X(Y) = \mathscr{T}_X(Z)$.*

A relação binária $\equiv_{\mathscr{T}_X}$ constitui-se em uma relação de equivalência sobre Σ^* pois a igualdade de conjuntos que a define é uma relação de equivalência sobre $\mathbb{P}(\{1, \ldots, n\}) \cup \{0, \ldots, n\}$, onde $\mathbb{P}(S)$ denota o conjunto das partes de S. Denotamos por $[Y]_{\mathscr{T}_X}$ a classe de equivalência de Y segundo $\equiv_{\mathscr{T}_X}$. Usando o mesmo exemplo acima ($X = \texttt{AABBABC}$), temos $[\texttt{BB}]_{\mathscr{T}_X} = [\texttt{AABB}]_{\mathscr{T}_X}$, $[\texttt{BA}]_{\mathscr{T}_X} = [\texttt{AABBA}]_{\mathscr{T}_X}$ e $[\texttt{C}]_{\mathscr{T}_X} = [X]_{\mathscr{T}_X}$.

A seguinte proposição contempla algumas propriedades elementares da relação de término-equivalência.

Proposição 4.2

i) $\equiv_{\mathscr{T}_X}$ *é invariante à direita e tem índice finito.*

ii) Se $Y, Z \in \mathscr{F}(X)$ são tais que $\mathscr{T}_X(Y) \cap \mathscr{T}_X(Z) \neq \varnothing$, então $Y \sqsupset Z$ ou $Z \sqsupset Y$. A fortiori, *se dois fatores de X são término-equivalentes então um deles é sufixo do outro.*

iii) Duas cadeias YZ e Z são término-equivalentes em X se, e somente se, toda ocorrência de Z em X é precedida por uma ocorrência de Y.

iv) Um fator $Y \in \mathscr{F}(X)$ é o maior elemento da classe de equivalência $[Y]_{\mathscr{T}_X}$ se, e somente se, Y é prefixo de X ou Y deriva à esquerda em X.

Prova

i) Sejam $Y, Z \in \Sigma^*$ tais que $Y \equiv_{\mathscr{T}_X} Z$ e tome $W \in \Sigma^*$ arbitrário. Então $\mathscr{T}_X(Y) = \mathscr{T}_X(Z)$ e, portanto,

$$\begin{aligned}\mathscr{T}_X(YW) &= \{i + |W| \mid W = X_{i+1..i+|W|} \ \wedge \ i \in \mathscr{F}(Y)\} \\ &= \{i + |W| \mid W = X_{i+1..i+|W|} \ \wedge \ i \in \mathscr{F}(Z)\} \\ &= \mathscr{T}_X(ZW).\end{aligned}$$

Para qualquer cadeia $V \notin \mathscr{F}(X)$, temos $\mathscr{T}_X(V) = \varnothing$ e, logo, $[V]_{\mathscr{T}_X} = \Sigma^* \setminus \mathscr{F}(X)$. Como $\mathscr{F}(X)$ é finito, então ele está particionado em um número finito de classes de equivalência.

ii) Seja $i \in \mathscr{T}_X(Y) \cap \mathscr{T}_X(Z)$ e suponha, sem perda de generalidade, que $|Y| \leq |Z|$. Então $Y = x_{i-|Y|+1} \cdots x_i$ e $Z = x_{i-|Z|+1} \cdots x_i \therefore Y \sqsupset Z$.

iii) ($\Longrightarrow$) Suponha, do contrário, que $\exists W \not\sqsupset Y$ t.q. $|W| \leq |Y| \ \wedge \ WZ = X_{i-|WZ|+1..i}$. Então $i \in \mathscr{T}_X(Z) \wedge i \notin \mathscr{T}_X(Y)Z \implies Z \not\equiv_{\mathscr{T}_X} YZ$. ($\Longleftarrow$) Da Definição 4.2, temos $\mathscr{T}_X(YZ) \subset \mathscr{T}_X(Z)$ e, da hipótese, temos $\mathscr{T}_X(Z) \subset \mathscr{T}_X(YZ) \therefore \mathscr{T}_X(Y) = \mathscr{T}_X(YZ)$.

iv) Primeiramente, observamos que, devido à finitude de $\mathscr{F}(X)$ e a (ii), cada classe de equivalência de $\equiv_{\mathscr{T}_X}$ possui um, e apenas um, representante máximo com respeito ao comprimento, o qual denominamos *representante canônico* da classe. ($\Longrightarrow$) Se Y é o representante canônico de $[Y]_{\mathscr{T}_X}$, então, por (ii), todo elemento de $[Y]_{\mathscr{T}_X}$ é sufixo de Y. Portanto, se $[Y]_{\mathscr{T}_X}$ contém algum prefixo $Z \sqsubset X$, temos $Z \sqsupset Y \implies |Z| \leq |Y|$. Por outro lado, temos $|Z| \in \mathscr{T}_X(Z) = \mathscr{T}_X(Y) \implies |Y| \leq |Z| \therefore |Y| = |Z| \therefore Y = Z$. Se, ao contrário, $[Y]_{\mathscr{T}_X}$ não possui prefixo algum de X, então Y deriva à esquerda em X. Com efeito, suponha que Y não deriva à esquerda em X, *i.e.*, $\exists a \in \Sigma$ tal que toda ocorrência de Y em X é precedida por a. Então, por (iii), temos que $aY \equiv_{\mathscr{T}_X} Y \implies aY \in [Y]_{\mathscr{T}_X}$, o que contraria a maximalidade de Y. ($\Longleftarrow$) Se Y é prefixo de X, então Y é o maior elemento de $[Y]_{\mathscr{T}_X}$ conforme discutido anteriormente. Senão, suponha que Y deriva à esquerda em X e que exista um $W \in [Y]_{\mathscr{T}_X}$ t.q. $|W| > |Y|$. Por (ii), temos que $Y \sqsupset W = ZaY$ para algum $Z \in \Sigma^*$ e $a \in \Sigma$. Como $Y \equiv_{\mathscr{T}_X} W$, então, por (iii), toda ocorrência de Y em X é precedida por a e, logo, Y não pode derivar à esquerda. Da contradição, temos que $\nexists W \in [Y]_{\mathscr{T}_X}$ t.q. $|W| > |Y|$. ∎

Definição 4.4 (Grafo Direcionado Acíclico de Fatores—DAWG)
Na terminologia da Teoria dos Autômatos Finitos: *Dada a cadeia $X \in \Sigma^*$,* $\text{DAWG}(X) = (Q, \Sigma, \delta, q_0, F)$ *é o AFD incompleto no qual*

$$\begin{aligned}&Q = \mathscr{F}(X)/\equiv_{\mathscr{T}_X} = \{[Y]_{\mathscr{T}_X} \mid Y \in \mathscr{F}(X)\}, \\ &\delta([Y]_{\mathscr{T}_X}, a) = [Ya]_{\mathscr{T}_X}, \ \textit{com } Y, Ya \in \mathscr{F}(X), \\ &q_0 = [\varepsilon]_{\mathscr{T}_X}, \\ &F = Q.\end{aligned}$$

Na terminologia da Teoria dos Grafos: $\text{DAWG}(X) = (V_{\text{DAWG}(X)}, A_{\text{DAWG}(X)})$ *é o grafo direcionado acíclico no qual*

$$V_{\text{DAWG}(X)} = \mathscr{F}(X)/\equiv_{\mathscr{T}_X} = \{[Y]_{\mathscr{T}_X} \mid Y \in \mathscr{F}(X)\},$$
$$A_{\text{DAWG}(X)} = \{[Y]_{\mathscr{T}_X} \xrightarrow{a} [Ya]_{\mathscr{T}_X} \mid Y, Ya \in \mathscr{F}(X),\ a \in \Sigma\}.$$

Estritamente falando, $\mathscr{F}(X)$ não é um monóide e, portanto, a definição do "quociente" $\mathscr{F}(X)/\equiv_{\mathscr{T}_X}$ é imprecisa. Abusamos dessa notação para representar a partição de $\mathscr{F}(X)$ em classes de equivalência, induzida pela relação $\equiv_{\mathscr{T}_X}$ sobre Σ^*, ou seja, $\mathscr{F}(X)/\equiv_{\mathscr{T}_X} = \Sigma^*/\equiv_{\mathscr{T}_X} \cap \mathbb{P}(\mathscr{F}(X))$. Repare que a função de transição δ da Definição 4.4 é , de fato, bem-definida em conseqüência da invariância à direita da relação $\equiv_{\mathscr{T}_X}$ (Proposição 4.2(i)). Com efeito, se $Y \equiv_{\mathscr{T}_X} Z$ são dois representantes distintos de $[Y]_{\mathscr{T}_X}$, temos $Ya \equiv_{\mathscr{T}_X} Za,\ \forall a \in \Sigma$ e, portanto, $\delta([Y]_{\mathscr{T}_X}, a) = [Ya]_{\mathscr{T}_X} = [Za]_{\mathscr{T}_X} = \delta([Z]_{\mathscr{T}_X}, a)$. A Figura 4.1 ilustra a Definição 4.4.

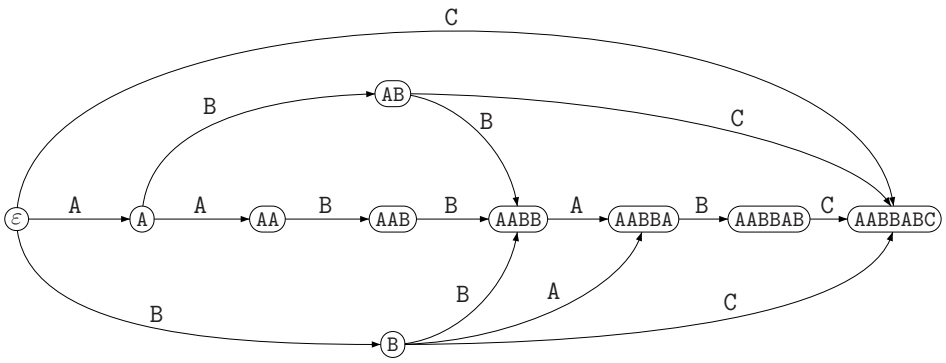


Figura 4.1. DAWG(AABBABC). Os rótulos dos vértices indicam os representantes canônicos das classes de $\equiv_{\mathscr{T}_X}$ a que eles correspondem.

A seguinte proposição contempla algumas propriedades elementares de $\text{DAWG}(X)$ diretamente provenientes da Definição 4.4.

Proposição 4.3

i) Para cada estado $q = [Y]_{\mathscr{T}_X} \in Q$, existe um caminho $[\varepsilon]_{\mathscr{T}_X} \rightsquigarrow q$ cujo rótulo é igual a Y. Mais ainda, se Y é o representante canônico de q, então o rótulo de qualquer caminho $[\varepsilon]_{\mathscr{T}_X} \rightsquigarrow q$ é um sufixo de Y.*

ii) $\text{DAWG}(X)$ *possui, exatamente, uma* fonte *e um* sorvedouro† *dados, respectivamente, por* $[\varepsilon]_{\mathscr{T}_X}$ *e* $[X]_{\mathscr{T}_X}$.

*Referir-nos-emos a $[Y]_{\mathscr{T}_X}$ como "classe", "estado" ou "vértice" indistintamente.

†Se $G = (V_G, A_G)$ denota um grafo, o *corte* associado ao subconjunto de vértices $V'_G \subset V_G$ é o subconjunto de arestas $A'_G \subset A_G$ incidentes a vértices de V'_G tais que, para cada uma dessas arestas, uma das suas extremidades pertence a V'_G e a outra pertence a $V_G \setminus V'_G$. Se todas as arestas do corte associado a V'_G são eferentes (aferentes) a vértices de V'_G, então V'_G é dito uma *fonte* (resp. *sorvedouro*) de G [LSS+79].

Teorema 4.3 DAWG(X) *reconhece* $\mathscr{F}(X)$.

Prova O enunciado decorre, imediatamente, da demonstração do Teorema 4.1 observando-se que $\mathscr{F}(X)$ pode ser obtido a partir da reunião das classes de equivalência de $\equiv_{\mathscr{T}_X}$ (invariante à direita e de índice finito) correspondentes aos estados de $Q = F$ e que $R_{\text{DAWG}(X)}$ e $\equiv_{\mathscr{T}_X}$ são idênticas. ∎

O AFD DAWG(X) não é o autômato mínimo a reconhecer o conjunto $\mathscr{F}(X)$ [BBH+85]. Entretanto, essa minimalidade se verifica caso estejamos interessados em reconhecer apenas $\mathscr{S}(X)$.

Definição 4.5 (Autômato de sufixos) *O* autômato de sufixos *de uma cadeia* $X \in \Sigma^*$, SA(X) *é o AFD obtido a partir de* DAWG(X) *se consideramos como estados finais apenas as classes de equivalência dos sufixos de* X*, ou seja,* $F = \{[Y]_{\mathscr{T}_X} \in Q \mid Y \in \mathscr{S}(X)\}$.

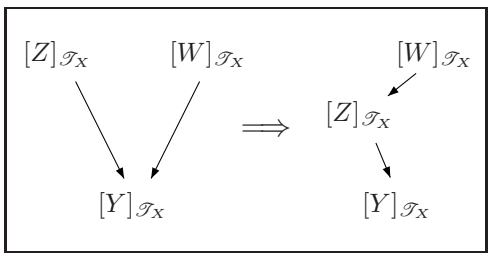
Teorema 4.4 (Crochemore) SA(X) *é o menor AFD incompleto a reconhecer o conjunto* $\mathscr{S}(X)$.

Prova A prova de que SA(X) reconhece $\mathscr{S}(X)$ é trivial. A minimalidade de SA(X) decorre do Teorema 4.2 se mostarmos que SA(X) corresponde ao autômato A' da terceira parte da demonstração do Teorema 4.1 quando $\mathscr{L} = \mathscr{S}(X)$, o que equivale a provar que as relações $\equiv_{\mathscr{T}_X}$ e $R_{\mathscr{S}(X)}$ são idênticas, *i.e.*,

$$Y \equiv_{\mathscr{T}_X} Z \iff YW \in \mathscr{S}(X) \text{ exatamente quando } ZW \in \mathscr{S}(X),\ \forall W \in \Sigma^*.$$

($\Longrightarrow$) Trivial, sabendo-se que $\mathscr{T}_X(Y) = \mathscr{T}_X(Z)$. ($\Longleftarrow$) Suponha, sem perda de generalidade, que $\exists i \in \mathscr{T}_X(Y)$ t.q. $i \notin \mathscr{T}_X(Z)$ e seja $W = X_{i+1..}$. Nesse caso, temos $YW \sqsupset X$ e $ZW \not\sqsupset X$, o que é uma contradição $\therefore \mathscr{T}_X(Y) = \mathscr{T}_X(Z)$. ∎

4.1.2.1 ***Suffix links*** Se identificamos cada classe de equivalência não-degenerada de $\equiv_{\mathscr{T}_X}$ com o conjunto de términos a que ela corresponde, então, pela Proposição 4.2(ii), temos que $\equiv_{\mathscr{T}_X}$ induz uma estrutura de árvore entre as suas classes de equivalência, correspondente à relação de inclusão desses conjuntos de términos. Com efeito, $\forall Y \in \mathscr{F}(X)$, $\mathscr{T}_X(Y) \subset \mathscr{T}_X(\varepsilon)$ (*i.e.*, $[Y]_{\mathscr{T}_X} \subset [\varepsilon]_{\mathscr{T}_X}$) $\therefore \forall Y \in \mathscr{F}(X) \setminus \{\varepsilon\}$, $\exists Z \in \mathscr{F}(X)$ t.q. $[Y]_{\mathscr{T}_X} \subsetneq [Z]_{\mathscr{T}_X}$. Mais ainda, se $Y, Z, W \in \mathscr{F}(X)$ são tais que $[Y]_{\mathscr{T}_X} \subsetneq [Z]_{\mathscr{T}_X}$ e $[Y]_{\mathscr{T}_X} \subsetneq [W]_{\mathscr{T}_X}$, então, $[Z]_{\mathscr{T}_X} \cap [W]_{\mathscr{T}_X} \neq \varnothing \overset{\text{Prop. 4.2(ii)}}{\Longrightarrow} W \sqsupset Z$ (ou $Z \sqsupset W$) $\Longrightarrow$ $[Z]_{\mathscr{T}_X} \subset [W]_{\mathscr{T}_X}$. Esta situação encontra-se ilustrada no diagrama ao lado na qual uma seta $A \to B$ é utilizada para representar a inclusão $A \supset B$.



À luz da discussão do parágrafo anterior, podemos definir, a exemplo do que fizemos para as árvores de sufixos, conexões auxiliares em DAWG(X) também denominadas *suffix links*.

Definição 4.6 (*Suffix link*) *Sejam* $[Z]_{\mathscr{T}_X} \neq [\varepsilon]_{\mathscr{T}_X}$ *e* $[Y]_{\mathscr{T}_X}$ *classes não-degeneradas de* $\equiv_{\mathscr{T}_X}$*, representadas canonicamente por* $Z = UY$ *(*$U \in \Sigma^+$*) e* Y *respectivamente, sendo* $[Y]_{\mathscr{T}_X}$ *a menor classe de* $\equiv_{\mathscr{T}_X}$ *tal que* $[Z]_{\mathscr{T}_X} \subsetneq [Y]_{\mathscr{T}_X}$ *(no sentido da inclusão dos conjuntos de términos correspondentes). A "aresta"* $[Y]_{\mathscr{T}_X} \overset{U^R}{\dashrightarrow} [Z]_{\mathscr{T}_X}$ *é dita um* suffix link *de* DAWG(X)*.*

Definição 4.7 (Árvore de *suffix links* de um DAWG) *A árvore formada pelos vértices de* DAWG(X) *cuja raiz corresponde a* $[\varepsilon]_{\mathscr{T}_X}$ *e cujas arestas correspondem aos seus* suffix links *é denominada* árvore de *suffix links* de DAWG(X) *e denotada por* DAWG$(X)^R$*.*

A Figura 4.2 ilustra a Definição 4.7 acima. Na Seção 4.3.1 provaremos, assim como o fizemos no caso das árvores de sufixos, um certo aspecto de dualidade existente entre DAWG(X) e DAWG$(X)^R$, mais precisamente, DAWG$(\$X)^R$ = CST$(X^R\$)$.

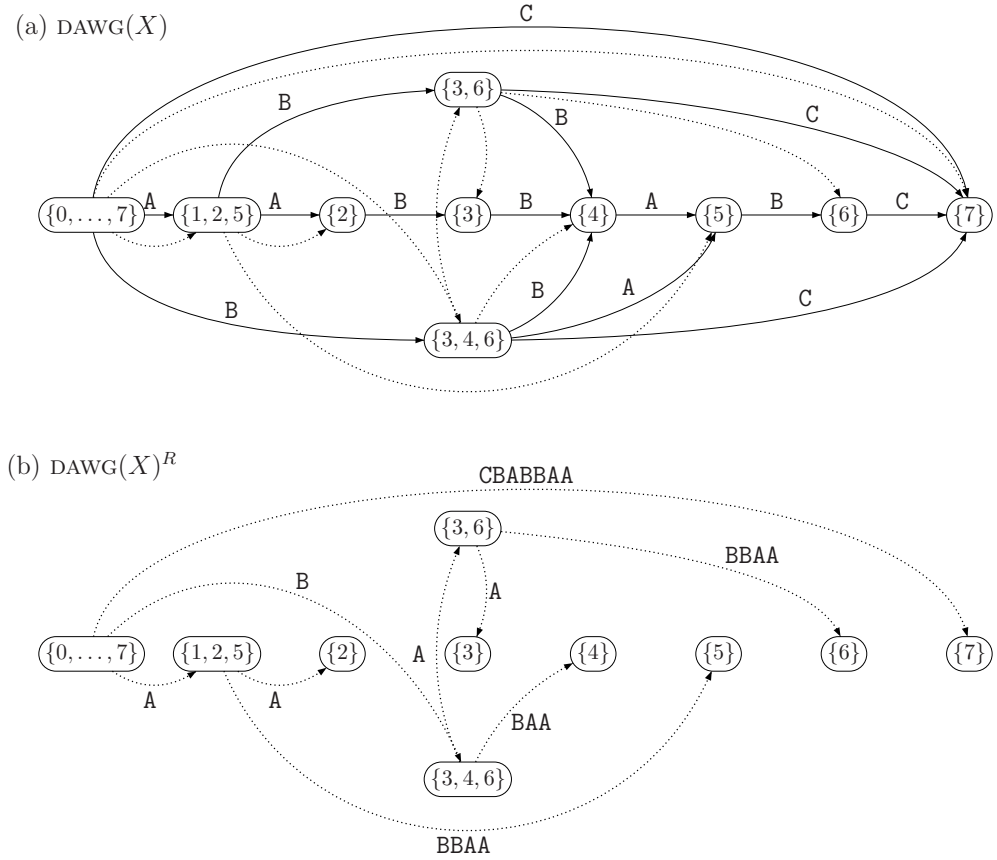


Figura 4.2. (a) DAWG(AABBABC) com os seus *suffix links* (linhas pontilhadas). Os rótulos dos vértices indicam os conjuntos de términos a que eles correspondem. (b) DAWG(AABBABC)^R.

O seguinte lema revela uma interessante propriedade estrutural das árvores de *suffix links* dos DAWGs, ao mesmo tempo em que desempenha um papel importante no estudo da complexidade espacial dessas estruturas conforme verificaremos adiante.

Lema 4.1 *Se* Y *representa canonicamente o vértice* $[Y]_{\mathscr{T}_X} \in$ DAWG(X)*, então os filhos desse vértice em* DAWG$(X)^R$ *são os vértices da forma* $[aY]_{\mathscr{T}_X}$ t.q. $a \in \Sigma$ *e* $aY \in \mathscr{F}(X)$*.*

Prova Os filhos de $[Y]_{\mathscr{T}_X} \in \text{DAWG}(X)^R$, vistos como conjuntos de términos, são os sub-conjuntos próprios maximais de $\mathscr{T}_X(Y)$ que também representam classes de equivalência de $\equiv_{\mathscr{T}_X}$. Cada um desses subconjuntos é da forma $\mathscr{T}_X(UY)$ para algum $U \in \Sigma^+$ t.q. $UY \in \mathscr{F}(X)$ e $UY \not\equiv_{\mathscr{T}_X} Y$. Como Y é o maior representante de $[Y]_{\mathscr{T}_X}$, já temos essas condições satisfeitas com $|U| = 1$. ∎

4.1.2.2 Limites Superiores Os DAWGs, a exemplo das *suffix tries*, são grafos *atômicos*, *i.e.*, grafos nos quais as arestas possuem rótulos unitários. No caso das árvores de sufixos atômicas, essa vantagem vem a custa de um número quadrático de vértices, o que torna aquelas estruturas, em muitos casos, inviáveis. Todavia, no que diz repeito aos DAWGs, esse compromisso não se verifica, conforme demonstra o teorema a seguir.

Teorema 4.5 *Seja* $\text{DAWG}(X) = (V_{\text{DAWG}(X)}, A_{\text{DAWG}(X)})$ *para a cadeia* $X = x_1 \cdots x_n$ *com* $n > 1$. *Então* $|V_{\text{DAWG}(X)}| \leq 2n - 1$ *e* $|A_{\text{DAWG}(X)}| \leq 3n - 3$.

Prova Se vistos como conjuntos de términos, as folhas de $\text{DAWG}(X)^R$ são sub-conjuntos disjuntos dois-a-dois de $\{1, \ldots, n\}$ (apenas a classe $[\varepsilon]_{\mathscr{T}_X}$ contém o "0" e esta sabemos tratar-se da RAIZ de $\text{DAWG}(X)^R$). Logo, $\text{DAWG}(X)$ possui, no máximo n folhas, o que não implica na tese quanto ao número de vértices uma vez que $\text{DAWG}(X)^R$ ainda pode conter vértices internos de grau 1.

Consideremos a partição

$$\begin{aligned} V_{\text{DAWG}(X)} = \{[Y]_{\mathscr{T}_X} \mid Y \text{ é repr. canônico de } [Y]_{\mathscr{T}_X} \in V_{\text{DAWG}(X)} \ \wedge \ Y \sqsubset X\} \\ \cup \{[Y]_{\mathscr{T}_X} \mid Y \text{ é repr. canônico de } [Y]_{\mathscr{T}_X} \in V_{\text{DAWG}(X)} \ \wedge \ Y \not\sqsubset X\} \end{aligned}$$

e estimemos a cardinalidade de cada sub-conjunto separadamente. Obviamente, o primeiro conjunto possui, exatamente, $n + 1$ elementos correspondentes aos $n + 1$ prefixos de X. Se $[Y]_{\mathscr{T}_X}$ pertence ao segundo conjunto, então, pela Proposição 4.2(iv), Y deriva à esquerda em X. Logo, pelo Lema 4.1, $[Y]_{\mathscr{T}_X}$ possui pelo menos dois filhos em $\text{DAWG}(X)^R$ sendo, portanto, um vértice interno dessa árvore. Como sabemos que $\text{DAWG}(X)^R$ possui, no máximo, n folhas, então o segundo conjunto possui, a princípio, até $n - 1$ elementos. Ocorre que o vértice interno $[\varepsilon]_{\mathscr{T}_X}$ já foi contabilizado no primeiro sub-conjunto e, logo, o segundo conjunto possui menos de $n - 1$ elementos $\therefore \ |V_{\text{DAWG}(X)}| < (n + 1) + (n - 1) < 2n \leq 2n - 1$.

Seja T uma árvore geradora de $\text{DAWG}(X)$ com raiz em $[\varepsilon]_{\mathscr{T}_X}$ e que contém o caminho $[\varepsilon]_{\mathscr{T}_X} \rightsquigarrow [X]_{\mathscr{T}_X}$ rotulado por X. Para estimar o número de arestas em $\text{DAWG}(X)$, consideremos a partição

$$\begin{aligned} A_{\text{DAWG}(X)} = \{u \to v \mid u \to v \in A_{\text{DAWG}(X)} \ \wedge \ u \to v \in T\} \\ \cup \{u \to v \mid u \to v \in A_{\text{DAWG}(X)} \ \wedge \ u \to v \notin T\}, \end{aligned}$$

e estimemos a cardinalidade de cada sub-conjunto separadamente. Obviamente, o primeiro conjunto possui, exatamente, $|V_{\text{DAWG}(X)}| - 1$ arestas uma vez que T é uma árvore geradora de $\text{DAWG}(X)$. Para cada aresta $u \to v$ no segundo conjunto, associamos o caminho em $\text{DAWG}(X)$ $\pi(u \to v) = [\varepsilon]_{\mathscr{T}_X} \to t_1 \to t_2 \to \cdots \to u \to v \to w_1 \to w_2 \to \cdots \to [X]_{\mathscr{T}_X}$, tal que o caminho $[\varepsilon]_{\mathscr{T}_X} \to t_1 \to t_2 \to \cdots \to u$ corresponde ao único caminho em T

da sua raiz $[\varepsilon]_{\mathscr{T}_X}$ até o vértice u e $v \to w_1 \to w_2 \to \cdots \to [X]_{\mathscr{T}_X}$ é algum caminho de v até o sorvedouro de DAWG(X). Pela Proposição 4.3(i), o rótulo de $\pi(u \to v)$ é um sufixo de X e, mais ainda, se $u \to v$ e $u' \to v'$ são duas arestas distintas do segundo conjunto, os rótulos de $\pi(u \to v)$ e $\pi(u' \to v')$ são distintos pois estes caminhos são, claramente diferentes (eles diferem na primeira aresta fora de T). Logo, o número de arestas do segundo conjunto limita-se como o número de sufixos distintos de X menos ε e o próprio X, que só podem ser produzidos por caminhos totalmente contidos em T $\therefore\ |A_{\text{DAWG}(X)}| \leq (|V_{\text{DAWG}(X)}| - 1) + (|\mathscr{S}(X)| - 2) \leq (2n-2) + (n-1) \leq 3n-3.$ ∎

4.2 DAWGs COMPACTOS: CDAWGs

O Teorema 4.5 acima comprova que os DAWGs são estruturas lineares e que, portanto, sob o aspecto do espaço requerido, são apropriadas para utilização como estruturas de índices. Ocorre que, em algumas aplicações (notadamente, em Biologia Computacional) as constantes multiplicativas envolvidas nessa complexidade são, de fato, significativas. Nesse sentido, também a exemplo do que foi feito para as árvores de sufixos atômicas, podemos recorrer a alguma estratégia de compactação da estrutura, retirando-lhe alguns vértices que, de certa maneira, puderem ser subentendidos. As estruturas resultantes da compactação dos DAWGs recebem o nome de *DAWGs Compactos* ou, simplesmente, CDAWGs.

4.2.1 Preliminares

Definição 4.8 (Inícios) *Dada uma cadeia $X = x_1 \cdots x_n \in \Sigma^*$, o* conjunto de inícios *da cadeia não-nula $Y \in \Sigma^+$ em X é dado por*

$$\mathscr{I}_X(Y) = \{i \mid Y = X_{i..i+|Y|-1}\}.$$

Por definição, $\mathscr{I}_X(\varepsilon) = \{0, \ldots, n\}$.

Como exemplo da Definição 4.8 acima, seja $X = \texttt{AABBABC}$. Então temos $\mathscr{I}_X(\texttt{A}) = \{1, 2, 5\}$, $\mathscr{I}_X(\texttt{AB}) = \{2, 5\}$ e $\mathscr{I}_X(\texttt{AAA}) = \varnothing$ (caso degenerado).

Definição 4.9 (Cadeias início-equivalentes) *Dizemos que cadeias $Y, Z \in \Sigma^*$ são* início-equivalentes *em $X \in \Sigma^*$ e denotamos $Y \equiv_{\mathscr{I}_X} Z$, se $\mathscr{I}_X(Y) = \mathscr{I}_X(Z)$.*

A relação binária $\equiv_{\mathscr{I}_X}$ também se constitui em uma relação de equivalência sobre Σ^* uma vez que a igualdade de conjuntos que a define é uma relação de equivalência sobre $\mathbb{P}(\{1, \ldots, n\}) \cup \{0, \ldots, n\}$. Denotamos por $[Y]_{\mathscr{I}_X}$ a classe de equivalência de Y segundo $\equiv_{\mathscr{I}_X}$. Para $X = \texttt{AABBABC}$, temos $[\texttt{BB}]_{\mathscr{T}_X} = [\texttt{BBA}]_{\mathscr{T}_X}$ e $[\texttt{AAB}]_{\mathscr{T}_X} = [X]_{\mathscr{T}_X}$.

A seguinte proposição é análoga à Proposição 4.2 e contempla algumas propriedades elementares da relação de início-equivalência.

Proposição 4.4

i) $\equiv_{\mathscr{I}_X}$ é invariante à esquerda e tem índice finito.

ii) *Se $Y, Z \in \mathscr{F}(X)$ são tais que $\mathscr{I}_X(Y) \cap \mathscr{I}_X(Z) \neq \varnothing$, então $Y \sqsubset Z$ ou $Z \sqsubset Y$.* A fortiori, *se dois fatores de X são início-equivalentes então um deles é prefixo do outro.*

iii) *Duas cadeias YZ e Y são início-equivalentes em X se, e somente se, toda ocorrência de Y em X é sucedida por uma ocorrência de Z.*

iv) *Um fator $Y \in \mathscr{F}(X)$ é o maior elemento da classe de equivalência $[Y]_{\mathscr{I}_X}$ se, e somente se, Y é sufixo de X ou Y deriva à direita em X.*

No que se segue, denotaremos por $\overleftarrow{Y}^{X}$ e $\overrightarrow{Y}^{X}$ os representantes canônicos da classes de equivalência $[Y]_{\mathscr{T}_X}$ e $[Y]_{\mathscr{I}_X}$ respectivamente*.

Definição 4.10 (Contexto) *O* contexto *do fator $Y \in \mathscr{F}(X)$ em X é definido como sendo o fator $\overleftrightarrow{Y}^{X} = UYV$ de X com $U, V \in \Sigma^*$ tais que $UY = \overleftarrow{Y}^{X}$ e $YV = \overrightarrow{Y}^{X}$.*

Ilustrando a Definição 4.10 acima, se $X = \texttt{AABBABC}$, então $\overleftrightarrow{\texttt{A}}^{X} = \texttt{A}$, $\overleftrightarrow{\texttt{AB}}^{X} = \texttt{AB}$ e $\overleftrightarrow{\texttt{BA}}^{X} = \texttt{AABBABC}$. Também decorre da referida definição de contexto que

$$\overleftrightarrow{Y}^{X} = (\overrightarrow{\overleftarrow{Y}^{X}}^{X}) = (\overleftarrow{\overrightarrow{Y}^{X}}^{X}) \,. \tag{4.2}$$

Se $Y \in \mathscr{F}(X)$ é tal que $\overleftrightarrow{Y}^{X} = Y$, então dizemos que Y é um *fator primo* de X.

Proposição 4.5 *Se $Y = \overleftrightarrow{Y}^{X}$, então Y é um prefixo de X ou Y é um sufixo de X ou Y deriva à esquerda e à direita em X. Em particular, se $Y = \overleftrightarrow{Y}^{X}$ é um fator primo não-aninhado de X, então $Y = X$.*

Prova Das proposições 4.2(iv) e 4.4(iv), temos:

$$Y = \overleftrightarrow{Y}^{X} \Longrightarrow \begin{cases} Y = \overleftarrow{Y}^{X} \Longrightarrow\ Y \sqsubset X \text{ ou } Y \text{ deriva à esquerda em } X \\ Y = \overrightarrow{Y}^{X} \Longrightarrow\ Y \sqsupset X \text{ ou } Y \text{ deriva à direita em } X. \end{cases}$$

Se Y não é prefixo nem sufixo de X, então, das implicações acima, temos que Y deriva à esquerda e à direita em X. Se Y é um fator primo não-aninhado de X então, obviamente, Y não deriva à esquerda e logo, pela primeira implicação acima, $Y \sqsubset X$. Analogamente, Y não deriva à direita em X e portanto, pela segunda implicação, $Y \sqsupset X \ \therefore Y = X$. ∎

*A seta sobre Y indica a direção na qual ele deve ser estendido para dar origem ao representante canônico da classe em questão.

Definição 4.11 (Equivalência de Contexto) *Dada uma cadeia $X \in \Sigma^*$, definimos a relação de* equivalência de contexto *da seguinte forma:*

$$Y \equiv_X Z \iff \overset{X}{\overleftrightarrow{Y}} = \overset{X}{\overleftrightarrow{Z}} .$$

Se $Y \equiv_X Z$, então dizemos que Y e Z são contexto-equivalentes* *em X.*

Ainda no exemplo $X = \texttt{AABBABC}$, temos $\overset{X}{\overleftrightarrow{\texttt{AA}}} = \overset{X}{\overleftrightarrow{\texttt{BA}}} = \overset{X}{\overleftrightarrow{\texttt{C}}} = \texttt{AABBABC}$. A relação $\equiv_X$ é, claramente, uma relação de equivalência sobre $\mathscr{F}(X)$ pois é definida em termos de uma igualdade entre cadeias.

Proposição 4.6 *$\equiv_X$ é o fecho transitivo de $\equiv_{\mathscr{I}_X} \cup \equiv_{\mathscr{T}_X}$.*

Prova ($\supset$) Sejam $Y, Z \in \mathscr{F}(X)$ tais que $Y \equiv_{\mathscr{T}_X} Z$ ($Y \equiv_{\mathscr{I}_X} Z$), então $\overset{X}{\overleftarrow{Y}} = \overset{X}{\overleftarrow{Z}}$ ($\overset{X}{\overrightarrow{Y}} = \overset{X}{\overrightarrow{Z}}$). Logo, de (4.2), temos $\overset{X}{\overleftrightarrow{Y}} = (\overset{X}{\overrightarrow{\overleftarrow{Y}}}) = (\overset{X}{\overrightarrow{\overleftarrow{Z}}}) = \overset{X}{\overleftrightarrow{Z}}$ ($\overset{X}{\overleftrightarrow{Y}} = (\overset{X}{\overleftarrow{\overrightarrow{Y}}}) = (\overset{X}{\overleftarrow{\overrightarrow{Z}}}) = \overset{X}{\overleftrightarrow{Z}}$) $\therefore Y \equiv_X Z$. ($\subset$) Reciprocamente, se $Y, Z \in \mathscr{F}(X)$ são tais que $\overset{X}{\overleftrightarrow{Y}} = UYV = U'ZV' = \overset{X}{\overleftrightarrow{Z}}$ (para $U, U', V, V' \in \Sigma^*$ t.q. $\overset{X}{\overleftarrow{Y}} = UY$, $\overset{X}{\overrightarrow{Y}} = YV$, $\overset{X}{\overleftarrow{Z}} = U'Z$ e $\overset{X}{\overrightarrow{Z}} = ZV'$), então

$$Y \equiv_{\mathscr{T}_X} \overset{X}{\overleftarrow{Y}} \equiv_{\mathscr{I}_X} (\overset{X}{\overrightarrow{\overleftarrow{Y}}}) = UYV = U'ZV' = (\overset{X}{\overrightarrow{\overleftarrow{Z}}}) \equiv_{\mathscr{I}_X} \overset{X}{\overleftarrow{Z}} \equiv_{\mathscr{T}_X} Z. \blacksquare$$

4.2.2 Definição e Propriedades Elementares

Definição 4.12 (DAWG Compacto—CDAWG) *O grafo direcionado acíclico compacto de fatores da cadeia $X \in \Sigma^*$ é dado por* $\textsc{cdawg}(X) = (V_{\textsc{cdawg}(X)}, A_{\textsc{cdawg}(X)})$ *onde*

$$V_{\textsc{cdawg}(X)} = \mathscr{F}(X)/\equiv_X = \{[Y]_X \mid Y \in \mathscr{F}(X)\} \simeq \{\overset{X}{\overleftrightarrow{Y}} \mid Y \in \mathscr{F}(X)\},$$
$$A_{\textsc{cdawg}(X)} = \{\overset{X}{\overleftrightarrow{Y}} \xrightarrow{aV} \overset{X}{\overleftrightarrow{Ya}} \mid Y, Ya \in \mathscr{F}(X),\ a \in \Sigma,\ V \in \Sigma^*,\ \overset{X}{\overrightarrow{Ya}} = YaV \text{ e } \overset{X}{\overrightarrow{Y}} \neq \overset{X}{\overrightarrow{Ya}}\}$$

A Figura 4.3 ilustra a Definição 4.12 acima. Comparando a Figura 4.3 com a Figura 4.1, reparamos que $\textsc{cdawg}(\texttt{AABBABC})$ pode ser obtido a partir de $\textsc{dawg}(\texttt{AABBABC})$ se compactamos os caminhos maximais $v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \xrightarrow{a_3} \cdots \xrightarrow{a_m} v_m$ nos quais a única aresta eferente ao vértice v_i é a aresta $v_i \xrightarrow{a_{i+1}} v_{i+1}$, $\forall i, 0 \leq i < m$, dando origem a uma única aresta $v_0 \xrightarrow{a_1 \cdots a_m} v_m$. Essa observação se verifica em geral e decorre da Proposição 4.6. Com efeito, seja $Y \in v_i$ para algum $0 < i \leq m$. Por definição de DAWG, $Y \equiv_{\mathscr{T}_X} Z$, $\forall Z \in v_i$, em particular $Y \equiv_{\mathscr{T}_X} \overset{X}{\overleftarrow{Y}}$. Além disso, de acordo com a Proposição

*Ou, simplesmente, "equivalentes" quando não houver margem para ambigüidades.

4.3(i), $\exists U \in v_{i-1}$ t.q. $Ua_i \sqsupset \overleftarrow{Y}^{X}$ e, mais ainda, como $v_{i-1} \xrightarrow{a_i} v_i$ é a única aresta eferente a v_{i-1}, então toda ocorrência de U em X é sucedida por a_i. Logo, pela Proposição 4.4(iii), $U \equiv_{\mathscr{I}_X} Ua_i \equiv_{\mathscr{T}_X} Y$. Ou seja, dado um fator arbitrário em qualquer uma das classes v_i $(0 < i \leq m)$, sempre existe um fator na classe "anterior" que se relaciona com o fator dado através de uma seqüência de fatores término-equivalentes ou início-equivalentes. Estendendo este resultado, temos que dois fatores quaisquer do conjunto de classes $\{v_0, \ldots, v_m\}$ pertencem à mesma classe do fecho transitivo de $\equiv_{\mathscr{I}_X} \cup \equiv_{\mathscr{T}_X}$, ou seja, são contexto-equivalentes.

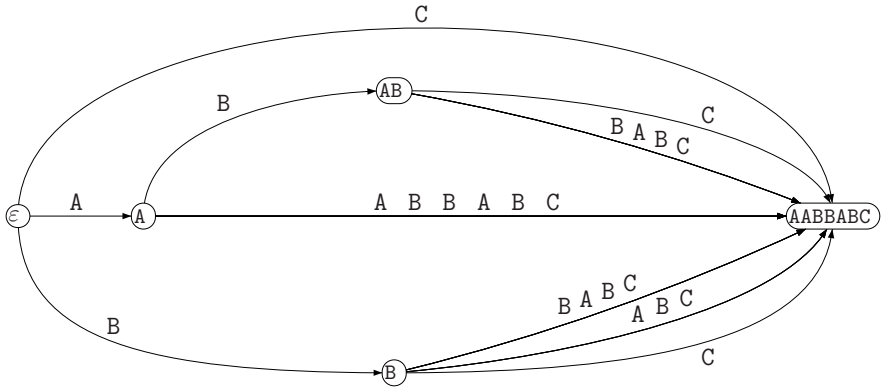


Figura 4.3. CDAWG(AABBABC).

Um outro aspecto de grande importância a ser observado é que a compactação de DAWG(X) preserva os *suffix links*. Em outras palavras, se um determinado vértice de DAWG(X) é mantido em CDAWG(X), então o seu pai na árvore de *suffix links* também o é. Desta forma, os *suffix links* remanescentes ainda formam uma árvore, denotada por CDAWG$(X)^R$, que é, na realidade, uma sub-árvore de DAWG$(X)^R$ (Figura 4.4). Esse fato pode ser constatado a partir da proposição a seguir.

Proposição 4.7 *Seja* $[Y]_{\mathscr{T}_X} \dashrightarrow [Z]_{\mathscr{T}_X}$ *um* suffix link *em* DAWG(X) *tal que* $\overleftarrow{Z}^{X} = \overleftrightarrow{Z}^{X}$. *Então* $\overleftarrow{Y}^{X} = \overleftrightarrow{Y}^{X}$.

Prova Da definição de *suffix link*, temos que $\overleftarrow{Y}^{X} \sqsupset \overleftarrow{Z}^{X}$. Como $\overleftarrow{Z}^{X} = \overleftrightarrow{Z}^{X}$, então $\overleftarrow{Z}^{X} = (\overrightarrow{\overleftarrow{Z}^{X}})$ e logo, pela Proposição 4.4(iv), $\overleftarrow{Z}^{X} \sqsupset X$ ou $\overleftarrow{Z}^{X}$ deriva à direita em X. Mas então $\overleftarrow{Y}^{X} \sqsupset X$ ou $\overleftarrow{Y}^{X}$ deriva à direita em X e, portanto, pela Proposição 4.4(iv), $(\overrightarrow{\overleftarrow{Y}^{X}}) = \overleftarrow{Y}^{X} \Longrightarrow \overleftarrow{Y}^{X} = \overleftrightarrow{Y}^{X}$. ∎

O teorema a seguir revela que a compactação dos DAWGs originam estruturas com cerca da metade da quantidade de vértices e apenas dois terços do número original de arestas, o que, em determinadas situações, pode traduzir-se em um ganho de memória de cerca de 50% [CV97b, CV97a].

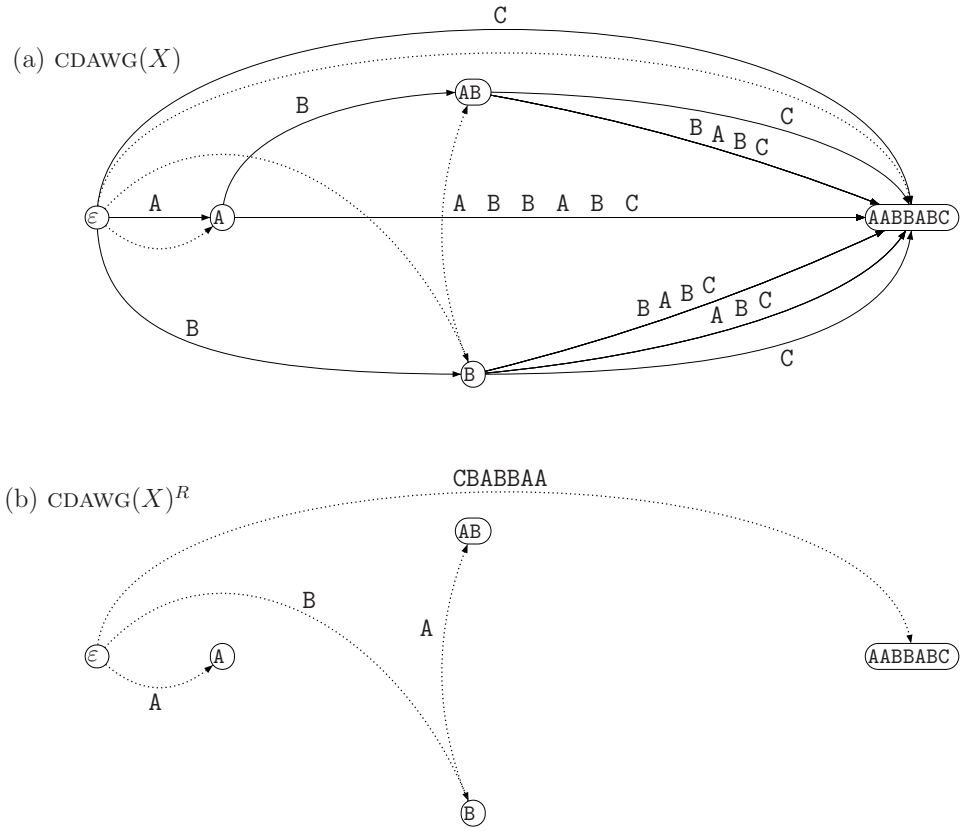


Figura 4.4. (a) CDAWG(AABBABC) com os seus *suffix links*. (b) CDAWG(AABBABC)^R. Observe que esta árvore é uma subárvore da árvore da Figura 4.2(b).

Teorema 4.6 *Seja* CDAWG$(X) = (V_{\text{CDAWG}(X)}, A_{\text{CDAWG}(X)})$ *para a cadeia* $X = x_1 \cdots x_n$ *com* $n > 1$. *Então* $|V_{\text{CDAWG}(X)}| \leq n + 1$ *e* $|A_{\text{CDAWG}(X)}| \leq 2n - 2$.

Prova Sabemos que CST$(X\$)$ possui $n+1$ folhas, correspondentes aos $n+1$ sufixos não-nulos de $X\$$, e, no máximo, n vértices internos, que estão bijetivamente relacionados aos fatores de $X\$$ que derivam à direita. Utilizaremos esses limites superiores para estimar o número de fatores primos de X, que correspondem aos vértices de CDAWG(X). Se Y é um fator primo aninhado de X, então Y deriva à direita em X ou Y é um sufixo (aninhado) de X. Em ambos os casos, Y está representado por algum vértice interno de CST$(X\$)$. Logo, o número de fatores primos aninhados de X limita-se com o número de vértices internos de CST$(X\$)$, ou seja, n. Por outro lado, sabemos, pela Proposição 4.5, que o único fator primo não-aninhado é o fator trivial $\therefore\ |V_{\text{CDAWG}(X)}| \leq n + 1$.

Se $X = a^n$, então CDAWG(X) possui, exatamente, n arestas. Consideremos então, o caso em que X possui, pelo menos, dois caracteres distintos. Seja $(Y = \overleftrightarrow{Y}^X) \xrightarrow{aV} \overleftrightarrow{Ya}^X \in A_{\text{CDAWG}(X)}$. Temos $Y = \overrightarrow{Y}^X$ e, portanto, ou Y deriva à direita em X ou Y é um sufixo aninhado de X (aninhado pois $Ya \in X$). Em ambos os casos, pelo mesmo raciocínio acima, Y é representado por um vértice interno v em CST$(X\$)$. Mais ainda, v possui

uma aresta eferente com rótulo iniciado por a. Dessa maneira, temos uma relação injetiva entre as arestas de $\textsc{cdawg}(X)$ e as arestas de $\textsc{cst}(X\$)$. Logo, $|A_{\textsc{cdawg}(X)}|$ é limitado pela quantidade de arestas de $\textsc{cst}(X\$)$. $\textsc{cst}(X\$)$ possui $n+1$ folhas e, no máximo, $n-1$ vértices internos (a RAIZ possui grau ≥ 3), possuindo, portanto, $2n-1$ arestas. Observamos, por fim, que a aresta $\overline{\varepsilon} \rightarrow \overline{\$}$ não corresponde a nenhuma aresta de $\textsc{cdawg}(X)$ $\therefore |A_{\textsc{cdawg}(X)}| \leq 2n-2$. ∎

4.3 DAWGs SIMÉTRICOS: SCDAWGs

Nesta seção, trataremos dos SCDAWGs, que vêm a ser extensões duais dos CDAWGs. Antes porém, vamos nos ocupar em examinar uma certa dualidade já encontrada nos DAWGs. Essa dualidade é natural uma vez que os DAWGs decorrem de um processo de minimização das árvores de sufixos atômicas conforme discutido informalmente no início deste capítulo.

4.3.1 Dualidade

A seguinte proposição envolve os resultados fundamentais que nos fornecem subsídios para o estudo da dualidade dos DAWGs e, posteriormente, dos CDAWGs.

Proposição 4.8 *Se $Y \in \mathscr{F}(X)$, então as seguintes afirmações se verificam:*

i) $Y = \overleftarrow{Y}^{X} \iff Y^R = \overrightarrow{Y^R}^{X^R}$.

ii) $Y = \overrightarrow{Y}^{X} \iff Y^R = \overleftarrow{Y^R}^{X^R}$.

iii) $Y = \overleftrightarrow{Y}^{X} \iff Y^R = \overleftrightarrow{Y^R}^{X^R}$.

Prova

i) $Y = \overleftarrow{Y}^{X} \iff \begin{array}{c} Y \sqsubset X \text{ ou} \\ Y \text{ der. à esq. em } X \end{array} \iff \begin{array}{c} Y^R \sqsupset X^R \text{ ou} \\ Y^R \text{ der. à dir. em } X^R \end{array} \iff Y^R = \overrightarrow{Y^R}^{X^R}$.

ii) Análogo ao item (i).

iii) $Y = \overleftrightarrow{Y}^{X} \iff Y = \overleftarrow{Y}^{X} \wedge Y = \overrightarrow{Y}^{X} \iff Y^R = \overrightarrow{Y^R}^{X^R} \wedge Y^R = \overleftarrow{Y^R}^{X^R} \iff Y^R = \overleftrightarrow{Y^R}^{X^R}$. ∎

Considere a árvore de sufixos compacta $\textsc{cst}(W)$ de uma cadeia $W = w_1 \cdots w_n$ tal que $w_n \neq w_i$, $\forall 1 \leq i < n$. Os vértices internos de $\textsc{cst}(W)$ representam bijetivamente os fatores aninhados de W que derivam à direita e as folhas representam bijetivamente

os sufixos de W. Portanto, em conseqüência da Proposição 4.4(iv), $\textsc{cst}(W)$ pode ser redefinida como $\textsc{cst}(W) = (V_{\textsc{cst}(W)}, A_{\textsc{cst}(W)})$ tais que

$$\begin{aligned} V_{\textsc{cst}(W)} &= \{\overrightarrow{Y}^{W} \mid Y \in \mathscr{F}(W)\} \\ A_{\textsc{cst}(W)} &= \{\overrightarrow{Y}^{W} \xrightarrow{aV} \overrightarrow{Ya}^{W} \mid Y, Ya \in \mathscr{F}(W),\ a \in \Sigma,\ \overrightarrow{Ya}^{W} = YaV \text{ e } \overrightarrow{Y}^{W} \neq \overrightarrow{Ya}^{W}\}. \end{aligned} \tag{4.3}$$

Observe que se W possui um sufixo aninhado Y que, todavia, não deriva à direita, então temos $Y = \overrightarrow{Y}^{W} \in V$ quando sabemos que, na verdade, o locus de Y em $\textsc{cst}(W)$ é um vértice implícito. Daí a restrição de que último caractere de W não pode ocorrer em qualquer outra posição nessa cadeia.

Teorema 4.7 *Se $X = x_1 \cdots x_n$ é uma cadeia tal que $x_1 \neq x_i$, $\forall 1 < i \leq n$, então* $\textsc{dawg}(X)^R \simeq \textsc{cst}(X^R)$.

Prova Em virtude da restrição sobre X do enunciado, podemos definir $\textsc{cst}(X^R)$, de maneira análoga ao que foi feito em (4.3), como $\textsc{cst}(X^R) = (V_{\textsc{cst}(X^R)}, A_{\textsc{cst}(X^R)})$ tais que

$$\begin{aligned} V_{\textsc{cst}(X^R)} &= \{\overrightarrow{Y^R}^{X^R} \mid Y^R \in \mathscr{F}(X^R)\} \overset{\text{Prop. 4.8(i)}}{\simeq} \{\overleftarrow{Y}^{X} \mid Y \in \mathscr{F}(X)\} \\ A_{\textsc{cst}(X^R)} &= \{\overrightarrow{Y^R}^{X^R} \xrightarrow{aV^R} \overrightarrow{Y^Ra}^{X^R} \mid Y^R, Y^Ra \in \mathscr{F}(X^R),\ a \in \Sigma,\ \overrightarrow{Y^Ra}^{X^R} = Y^RaV^R \text{ e } \overrightarrow{Y^R}^{X^R} \neq \overrightarrow{Y^Ra}^{X^R}\} \\ &\simeq \{\overleftarrow{Y}^{X} \xrightarrow{aV^R} \overleftarrow{aY}^{X} \mid Y, aY \in \mathscr{F}(X),\ a \in \Sigma,\ \overleftarrow{aY}^{X} = VaY \text{ e } \overleftarrow{Y}^{X} \neq \overleftarrow{aY}^{X}\}. \end{aligned} \tag{4.4}$$

Comparando (4.4) com a Definição 4.4, constatamos que os conjuntos de vértices são, de fato, equivalentes. A equivalência das arestas é atestada pelo Lema 4.1. ∎

A Figura 4.2 ilustra o Teorema 4.7 acima*. Em geral, podemos impor a restrição do Teorema 4.7 sobre uma cadeia X, adicionando-lhe, ao seu início, o caractere sentinela `$`.

4.3.2 Definição e Propriedades Elementares

Sabemos (Teorema 3.8) que as árvores de sufixos atômicas apresentam a desejável propriedade de possuir um conjunto de vértices invariante (a menos de rótulo) sob a reversão da cadeia representada. Vimos também (Teorema 3.9) que as árvores de sufixos compactas apresentam essa invariância mas apenas parcialmente, necessitando assim, serem complementadas com alguns vértices de tal sorte a constituirem-se em estruturas duais conhecidas como árvores de afixos. A Proposição 4.8(iii) nos diz que, a exemplo

*Curiosamente, a cadeia $X =$ `AABBABC` não respeita a hipótese do Teorema 4.7 e, ainda assim, $\textsc{dawg}(\texttt{AABBABC})^R = \textsc{cst}(X^R)$. Isso ocorre porque, na realidade, esse teorema admite uma hipótese mais fraca, qual seja, que X (X^R) não possua prefixos (sufixos) aninhados que não derivem à esquerda (direita). Repare que o único sufixo aninhado de $X^R =$ `CBABBAA` é `A`, que deriva à direita, sendo representado, portanto, por um vértice explícito de $\textsc{cst}(X^R)$.

do que acontece com as *suffix tries*, os vértices de CDAWG(X) são invariantes sob a reversão da cadeia representada. Dessa forma, Blumer *et al.* [BBHM87] mostraram que os CDAWGs podem ser facilmente estendidos em estruturas duais denominadas *CDAWGs Simétricos* (SCDAWGs) mediante mera adição de arestas complementares.

Definição 4.13 (CDAWG Simétrico—SCDAWG) *O grafo direcionado acíclico compacto simétrico de fatores da cadeia $X \in \Sigma^*$ é o multigrafo*

$$\text{SCDAWG}(X) = (V_{\text{SCDAWG}(X)}, A_{\text{SCDAWG}(X)}, A^R_{\text{SCDAWG}(X)})$$

tal que

$$V_{\text{SCDAWG}(X)} = \mathscr{F}(X)/\equiv_X = \{[Y]_X \mid Y \in \mathscr{F}(X)\} \simeq \{\overleftrightarrow{Y}^X \mid Y \in \mathscr{F}(X)\},$$

$$A_{\text{SCDAWG}(X)} = \{\overleftrightarrow{Y}^X \xrightarrow{aV} \overleftrightarrow{Ya}^X \mid Y, Ya \in \mathscr{F}(X),\ a \in \Sigma,\ V \in \Sigma^*,\ \overrightarrow{Ya}^X = YaV \text{ e } \overrightarrow{Y}^X \neq \overrightarrow{Ya}^X\}$$

$$A^R_{\text{SCDAWG}(X)} = \{\overleftrightarrow{Y}^X \xrightarrow{aU^R} \overleftrightarrow{aY}^X \mid Y, Ya \in \mathscr{F}(X),\ a \in \Sigma,\ U \in \Sigma^*,\ \overleftarrow{aY}^X = UaY \text{ e } \overleftarrow{Y}^X \neq \overleftarrow{aY}^X\}$$

A Figura 4.5 ilustra a Definição 4.13 acima. Repare que SCDAWG(X) contém CDAWG(X) e CDAWG(X^R).

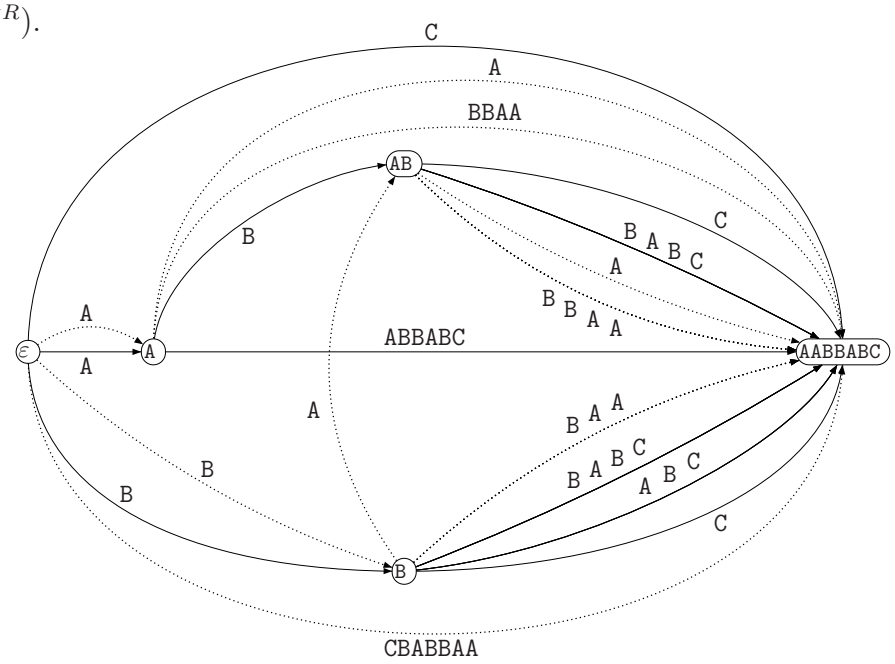


Figura 4.5. SCDAWG(AABBABC). As arestas de CDAWG(CBABBAA) estão indicadas na figura pelas linhas pontilhadas.

Os vértices de SCDAWG(X) são exatamente os mesmos de CDAWG(X) enquanto que as arestas correspondem às arestas de CDAWG(X) mais as arestas CDAWG(X^R). Logo, temos, trivialmente, o seguinte teorema.

Teorema 4.8 *Seja* $\text{SCDAWG}(X) = (V_{\text{SCDAWG}(X)}, A_{\text{SCDAWG}(X)}, A^R_{\text{SCDAWG}(X)})$ *para a cadeia* $X = x_1 \cdots x_n$ *com* $n > 1$. *Então* $|V_{\text{SCDAWG}(X)}| \leq n+1$ *e* $|A_{\text{SCDAWG}(X)} \cup A^R_{\text{SCDAWG}(X)}| \leq 4n-4$.

CAPÍTULO 5

VETORES DE SUFIXOS

A natureza tem perfeições para mostrar que é a imagem de Deus,
e defeitos para mostrar que é apenas a imagem.

— BLAISE PASCAL

O último grupo de estruturas de índice que vamos examinar é constituído pelas estruturas unidimensionais (ao contrário dos demais índices estudados anteriormente) denominadas *vetores de sufixos*. Os vetores de sufixos foram introduzidos por Manber e Myers [MM93]. Embora os vetores de sufixos já representem uma substancial economia de espaço sobre as demais estruturas, eles ainda são passíveis de compactação, originando os chamados *vetores de sufixos compactos*, apresentados, recentemente, por Mäkinen [Mäk00].

5.1 VETORES DE SUFIXOS

5.1.1 Definição e Propriedades Elementares

Definição 5.1 (Ordem lexicográfica) *Seja $\Sigma = \{a_1, \ldots, a_s\}$ uma enumeração de um dado alfabeto induzida por uma relação de ordem parcial $\preceq$, ou seja,*

$$a_i \preceq a_j \iff i \leq j.$$

Podemos estender essa relação $\preceq$ para o produto cartesiano Σ^*, dando origem a uma relação de ordem parcial denominada ordem lexicográfica*, definida da seguinte maneira:*

$$\begin{cases} \varepsilon \preceq X, \ \forall X \in \Sigma^* \\ X = x_1 \cdots x_n \preceq Y = y_1 \cdots y_m, \ \textit{se} \begin{cases} x_1 \neq y_1 \ \wedge \ x_1 \preceq y_1 \ \ \textit{ou} \\ x_1 = y_1 \ \wedge \ X_{2..} \preceq Y_{2..} \end{cases} \end{cases}$$

De maneira análoga à Definição 5.1 acima, podemos definir em Σ^* as relações "$\succeq$" ($X \succeq Y \iff Y \preceq X$) e "$\asymp$" ($X \asymp Y \iff X \preceq Y \wedge Y \preceq X$)*. Usamos, por simplicidade, "$X \prec Y$" e "$X \succ Y$" para denotar $X \preceq Y \wedge X \neq Y$ e $X \succeq Y \wedge X \neq Y$ respectivamente. No que se segue, estaremos considerando os alfabetos constituídos de caracteres latinos ordenados segundo a ordem alfabética convencional ($\texttt{A} \prec \texttt{B} \prec \texttt{C} \prec \cdots \prec \texttt{X} \prec \texttt{Y} \prec \texttt{Z}$).

Definição 5.2 (Ordem lexicográfica de k-prefixos) *Dado $k \in \mathbb{Z}_+$, definimos a relação de* ordem lexicográfica de k-prefixos *em Σ^*, $\preceq_k$, através da seguinte expressão*

$$X \preceq_k Y \iff X_{..k} \preceq Y_{..k}$$

*Como conseqüência imediata da definição, temos $X \asymp Y \iff X = Y$. Portanto, usaremos o sinal de igualdade "=" convencional.

Cadeia de entrada:	$X = \texttt{ABBCAAB}$
Sufixos ordenados:	$X_{5..} = \texttt{AAB}$ $X_{6..} = \texttt{AB}$ $X_{1..} = \texttt{ABBCAAB}$ $X_{7..} = \texttt{B}$ $X_{2..} = \texttt{BBCAAB}$ $X_{3..} = \texttt{BCAAB}$ $X_{4..} = \texttt{CAAB}$
Vetor de sufixos:	$\textsc{sarr}(X) = (5, 6, 1, 7, 2, 3, 4)$

Figura 5.1. Vetor de sufixos da cadeia $X = \texttt{ABBCAAB}$.

De maneira análoga à Definição 5.2 acima, podemos definir em Σ^* as relações "$\succeq_k$" ($X \succeq_k Y \iff Y \preceq_k X$) e "$=_k$" ($X =_k Y \iff X \preceq_k Y \wedge Y \preceq_k X \iff X_{..k} = Y_{..k}$). Também por simplicidade, usamos "$X \prec_k Y$" e "$X \succ_k Y$" para denotar $X \preceq_k Y \wedge X \neq Y$ e $X \succeq_k Y \wedge X \neq Y$ respectivamente.

A seguinte proposição contempla propriedades básicas da ordem lexicográfica.

Proposição 5.1

i) $X \preceq Y \implies X \preceq_k Y$, $\forall k \in \mathbb{Z}_+$

ii) Dadas as cadeias $X, Y \in \Sigma^*$ *tais que* $X =_k Y$ *para algum* $k \in \mathbb{Z}_+$, *se existe algum* $W \in \Sigma^*$ t.q. $X \preceq W \preceq Y$, *então* $X =_k W =_k Y$.

Prova

i) Trivial a partir das definições de $\preceq$ e $\preceq_k$.

ii) Suponha $X \preceq W \preceq Y$ tais que $X =_k Y$ e $X \neq_k W$. Logo, temos $W \prec_k X$ ou $W \succ_k X$. Se $W \prec_k X$, então $W \prec X$. Se $W \succ_k X$, então $W \succ_k Y \implies W \succ Y$. Dessas contradições, temos $X =_k W$. Analogamente, $Y =_k W$. ∎

Definição 5.3 *O* vetor de sufixos *de uma cadeia* $X = x_1 \cdots x_n \in \Sigma^*$ *é o vetor* n*-dimensional de inteiros não-negativos* $\textsc{sarr}(X) = (A_1, \ldots, A_n)$ *tal que*

$A_j = i \iff X_{i..}$ *é o* j*-ésimo sufixo não-nulo de* X *com respeito à ordem lexicográfica*,

ou seja,

$$X_{A_0..} \prec X_{A_1..} \prec \cdots \prec X_{A_n..}.$$

Em outras palavras, o vetor de sufixos de uma determinada cadeia corresponde, simplesmente, a uma lista ordenada dos seus sufixos não-nulos com respeito à ordem lexicográfica. A Figura 5.1 ilustra essa definição.

O vetor de sufixos de uma cadeia $X = x_1 \cdots x_n$ requer, na prática, meros $2n$ inteiros (adicionamo-lhe n inteiros correspondentes aos comprimentos de *máximos prefixos comuns* de alguns pares de sufixos de X, utilizados para acelerar a busca por padrões nessa cadeia).

5.1.2 Casamento de Padrões via Vetores de Sufixos

A utilização das estruturas de índice estudadas anteriormente na resolução do problema básico do casamento de padrões é bastante intuitiva. Em suma, se estamos preocupados em identificar ocorrências de um padrão Y em um texto X, construímos um desses índices tomando X como entrada e o percorremos, partindo da sua raiz ($\overline{\varepsilon}$ no caso das árvores de sufixos, $[\varepsilon]_{\mathscr{T}_X}$ no caso dos DAWGs), seguindo pelas arestas rotuladas pelos sucessivos caracteres $y_1, y_2, \ldots, y_n$. Se, dessa maneira, conseguimos "consumir" todo o padrão Y, teremos identificado a sua ocorrência. As posições exatas dessas ocorrências podem ser, em geral, recuperadas através de algumas anotações que podem ser feitas nas estruturas.

A utilização dos vetores de sufixos para a resolução do problema do casamento de padrões é, todavia, muito menos óbvia. Nesta seção, apresentaremos um algoritmo para a resolução do problema do casamento exato de padrões através de um vetor de sufixos, com o objetivo de ilustrar a utilização dessa estrutura. O algoritmo baseia-se no lema a seguir.

Lema 5.1 *Seja o texto $X = x_1 \cdots x_n \in \Sigma^*$ e $\textsc{sarr}(X) = (A_1, \ldots, A_n)$ o seu vetor de sufixos. Dado o padrão $Y = y_1 \cdots y_m$, definimos*

$$L_Y = \min(\{j \mid Y \preceq_m X_{A_j..}\} \cup \{n+1\})$$
$$R_Y = \max(\{j \mid Y \succeq_m X_{A_j..}\} \cup \{0\}).$$

Então

$$Y = X_{i..i+m-1} \iff i = A_j, \text{ para algum } j \text{ t.q. } L_Y \leq j \leq R_Y.$$

Prova ($\Longrightarrow$) Seja $Y = X_{i..i+m-1}$, ou seja, $Y =_m X_{i..}$. Seja j tal que $A_j = i$. Necessitamos mostrar que $L_Y \leq j \leq R_Y$. Se supomos $j < L_Y$, então temos $Y \preceq_m X_{A_j..}$ e $j < L_Y$, o que contraria a minimalidade de L_Y. Da contradição, temos $L_Y \leq j$. Analogamente, $R_Y \geq j$.

($\Longleftarrow$) Suponha que tenhamos $L_Y \leq j \leq R_Y$ e seja $i = A_j$. Da definição de vetor de sufixos, temos $X_{A_{L_Y}..} \preceq X_{A_{R_Y}..} \overset{\text{Prop. 5.1(i)}}{\Longrightarrow} X_{A_{L_Y}..} \preceq_m X_{A_{R_Y}..}$. Por outro lado, pelas definições de L_Y e R_Y, temos $X_{A_{R_Y}..} \preceq_m Y \preceq_m X_{A_{L_Y}..} \therefore X_{A_{L_Y}..} =_m Y =_m X_{A_{R_Y}..}$. Novamente, pela definição de vetores de sufixos, temos $X_{A_{L_Y}..} \preceq X_{A_j..} \preceq X_{A_{R_Y}..} \overset{\text{Prop. 5.1(ii)}}{\Longrightarrow}$ $X_{A_{L_Y}..} =_m X_{A_j..} =_m X_{A_{R_Y}..} \therefore Y =_m X_{A_j..} = X_{i..}$ ∎

O Lema 5.1 acima encontra-se ilustrado no diagrama da Figura 5.2. Através desse resultado, concluímos que a busca pelas ocorrências de Y em X corresponde tão somente à determinação dos valores de L_Y e R_Y. Se $L_Y \leq R_Y$, então Y ocorre em X nas posições correspondentes aos valores entre L_Y e R_Y (inclusive). A determinação dos valores de L_Y e R_Y pode ser realizada através de uma espécie de busca binária sobre os valores de $\textsc{sarr}(X)$ uma vez que estes representam uma ordenação dos sufixos de X. O Algoritmo 5.1 contempla a busca por L_Y. O processo é análogo para R_Y.

Por tratar-se de uma busca binária, o Algoritmo 5.1 efetua a comparação do padrão $Y = y_1 \cdots y_m$ com um fator de $X = x_1 \cdots x_n$ (linha *11*), no máximo, $\log n$ vezes. Cada uma dessas comparações implica na comparação de, no máximo, m símbolos. Portanto,

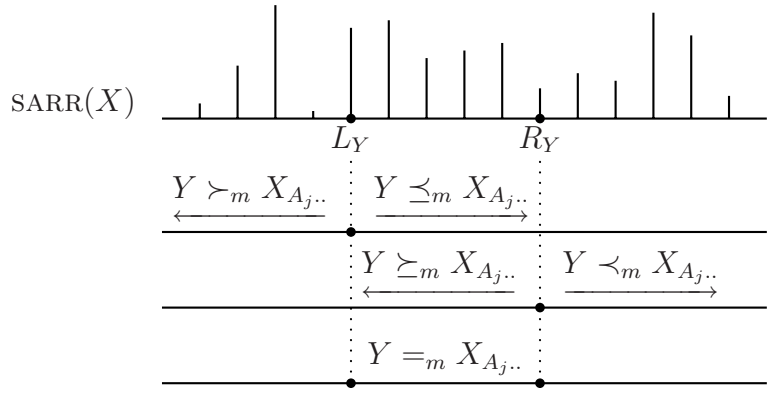


Figura 5.2. Ilustração do Lema 5.1. As linhas verticais representam os sufixos de X ordenados segundo $\preceq$.

```
1  Algoritmo busca_L_Y (X = x_1 ··· x_n, SARR(X) = (A_1, ..., A_n), Y = y_1 ··· y_m)
2  início
3    se Y ⪯_m X_{A_0..} então
4      L_Y ← 0
5    senão-se Y ≻_m X_{A_n..} então
6      L_Y ← n + 1
7    senão
8      (l, r) ← (0, n)
9      enquanto r − l > 1 faça
10       m ← (l + r)/2
11       se Y ⪯_m X_{A_m..} então
12         r ← m
13       senão
14         l ← m
15       fim-se
16     fim-faça
17   fim-se
18 fim
```

Algoritmo 5.1. Algoritmo *busca_*L_Y.

essa solução do problema do casamento de padrões via vetores de sufixos demanda tempo $O(m \cdot \log n)$. Manber e Myers [MM93] apresentaram uma versão aprimorada do algoritmo acima descrito em tempo $O(m + \log n)$. Esse novo algoritmo faz uso de informações pré-computadas acerca dos *máximos prefixos comuns* entre os sufixos $X_{l..}$, $X_{m..}$ e $X_{r..}$, envolvidos nas comparações supra-mencionadas.

5.2 VETORES DE SUFIXOS COMPACTOS

Um vetor de sufixos pode ser encarado como uma espécie de árvore de sufixos simplificada na qual apenas as folhas são representadas. A busca por um padrão na árvore de sufixos de um determinado texto pode ser, de certa maneira, simulada por uma busca binária no vetor de sufixos correspondente. Como sabemos, as árvores de sufixos implicam em uma certa redundância que se manifesta na forma de sub-árvores isomorfas. Essa redundância pode ser removida através de um processo de identificação dessas sub-árvores isormorfas que acaba por dar origem aos (C)DAWGs. Sendo os vetores de sufixos versões simplificadas das árvores de sufixos, seria razoável indagar se eles são passíveis de serem também compactados de alguma maneira. A resposta é sim e as estruturas resultantes são denominadas *vetores de sufixos compactos* [Mäk00].

Em uma árvore de sufixos, duas sub-árvores isormorfas representam os mesmos sufixos a menos de um prefixo comum a todos os sufixos representados por uma delas. Mais precisamente, se T e T' são duas sub-árvores isomorfas de uma árvore de sufixos de X e T é mais "profunda" do que T', então, se o conjunto de sufixos representados por T' é $S' \subset \mathscr{S}(X)$, o conjunto de sufixos representados por T é $U \cdot S'$ para algum $U \in \Sigma^*$ (U é, na realidade, a cadeia representada pela raiz de T). A idéia da compactação dos vetores de sufixos é, de certa forma, baseada nesse princípio e consiste em identificar trechos que representam as mesmas seqüências de sufixos (e não somente conjuntos, o que é natural uma vez que a estrutura é ordenada) a menos de um prefixo unitário comum a todos os sufixos de um dos trechos. Mais precisamente, dado o vetor de sufixos $\textsc{sarr}(X) = (A_1, \ldots, A_n)$, se existem dois trechos $A_i, \ldots, A_{i+\ell}$ e $A_j, \ldots, A_{j+\ell}$, para algum $\ell > 0$, tais que $A_{i+k} = A_{j+k} + 1,\ \forall 0 \leq k \leq \ell$, então substituimos o trecho $A_i, \ldots, A_{i+\ell}$ por um apontador para o trecho $A_j, \ldots, A_{j+\ell}$. Esse apontador passa a ocupar o lugar da entrada A_i e as demais posições $A_{i+1}, \ldots, A_{i+\ell}$ são desconsideradas. Essa compactação pode ser revertida por um processo inverso de descompactação no qual os apontadores são substituídos por trechos. A Figura 5.3 ilustra a compactação dos vetores de sufixos.

Definição 5.4 *O* vetor de sufixos compacto *de uma cadeia $X = x_1 \cdots x_n$ é o vetor $\textsc{csarr}(X) = (A'_1, \ldots, A'_{n'})$ (com $n' \leq n$) tal que cada entrada é dada por uma tripla $A'_i = (B, j, \ell)$. B é um valor booleano que indica se a entrada denota um sufixo ($B =$ `FALSE`) ou um apontador ($B =$ `TRUE`). Se $B =$ `FALSE`, então $X_{j..}$ é o i-ésimo sufixo não-nulo de X com respeito à ordem lexicográfica. Se $B =$ `TRUE`, então a $j + \ell$ indica a posição apontada após a descompactação do vetor.*

Mäkinen [Mäk00] demonstrou empiricamente que, em alguns casos, a compactação dos vetores de sufixos representa um ganho de memória de cerca de 50%. Um fato importante a ser destacado é que, na prática, o grau de compactação pode ser controlado através da imposição de limites sobre o comprimento dos trechos compactados e sobre

SARR(X)

i	$X_{A_i..}$	A_i
1	AAAB	6
2	AAB	7
3	AB	8
4	ACDAAAB	3
5	B	9
6	CDAAAB	4
7	CDACDAAAB	1
8	DAAAB	5
9	DACDAAAB	2

$\Rightarrow$

i	$X_{A_i..}$	A_i
1	AAAB	6
2	AAB	7
3	AB	8
4	ACDAAAB	3
5	$\longrightarrow$	(3)
6		
7	CDACDAAAB	1
8	$\longrightarrow$	(6)
9		

$\Rightarrow$

CSARR(X)

i	$X_{A'_i..}$	A'_i
1	AAAB	6
2	AAB	7
3	AB	8
4	ACDAAAB	3
5	$\longrightarrow$	(3,0)
6	CDACDAAAB	1
7	$\longrightarrow$	(5,1)

Figura 5.3. Vetor de sufixos compacto de $X =$ CDACDAAAB. À esquerda, temos o vetor de sufixos não-compactado. Ao centro, uma versão intermediária ilustrativa. Observe que os trechos A_5, A_6 e A_8, A_9 foram substituídas por apontadores para os trechos A_3, A_4 e A_6, A_7 respectivamente. Na prática, apenas a posição de início do trecho apontado necessita ser armazenada. Ocorre que o apontador armazenado em A_8 aponta para uma posição cujo trecho já foi compactado em um apontador. Nesse caso, na versão final do vetor compactado (à direita), o apontador passa a indicar o início do trecho compactado mais um valor correspondente ao deslocamento dentro do trecho após a sua descompactação. No exemplo, a entrada $A'_7 \rightarrow (A'_5, 1)$ indica que esse apontador aponta para a segunda posição do trecho resultante da descompactação do apontador $A'_5 \rightarrow (A'_3, 0)$.

o nível de "aninhamento" dos apontadores. Existe um compromisso entre o grau de compactação e o custo da resolução do problema do casamento de padrões via vetores de sufixos compactos [Mäk00, Mäk01]. O tempo esperado para a resolução do problema do casamento exato de padrões via vetores de sufixos compactos é $O((\frac{2n-n'}{n'})^2(m + k \log n))$, onde n denota o comprimento do texto, n' o tamanho do vetor de sufixos compacto, m o comprimento do padrão e k o número de ocorrências.

CAPÍTULO 6

INFERÊNCIA DE MOTIFS

*Se possuíssemos um conhecimento profundo de todas as partes do embrião de um animal qualquer (*e.g. *o homem), poderíamos, a partir disso apenas, por razões estritamente matemáticas e precisas, deduzir a conformação completa de cada um dos seus membros e, reciprocamente, se conhecêssemos as diversas peculiaridades dessa conformação, nós poderíamos, a partir delas, deduzir a natureza do embrião.*

— RENÉ DESCARTES

Neste capítulo, versaremos sobre o problema da inferência de *motifs* (biológicos). *Motifs* são determinados trechos das cadeias moculares biológicas que, em geral, correspondem a regiões altamente conservadas e que desempenham funções biológicas específicas (regiões promotoras, regiões reguladoras, sítios de ligação, *etc*). Estaremos interessados em estudar a inferência de *motifs* através de algoritmos combinatórios exatos. Em particular, examinaremos as soluções propostas por Sagot [Sag98] (*motifs* simples) e Marsan-Sagot [MS00] (*motifs* estruturados). Ambas as soluções supra-mencionadas, baseiam-se na utilização da árvore de sufixos do conjunto de cadeias sobre as quais os *motifs* devem ser inferidos. Ao final deste capítulo, apresentamos uma breve análise crítica sobre a adequação e desempenho das demais estruturas de índice abordadas nesta dissertação com respeito ao problema da inferência de *motifs* aqui apresentado.

6.1 SIMILARIDADE ENTRE CADEIAS

6.1.1 Descrição Geométrica de Similaridade

Vamos fazer uma breve digressão no sentido de justificar, intuitivamente, a noção de similaridade entre cadeias a ser adotada no restante do capítulo.

Considere, por exemplo, um conjunto discreto de pontos dispostos sobre um plano (ou em um espaço n-dimensional qualquer). Dizemos, intuitivamente, que dois pontos são *similares* quando eles são *próximos*, ou seja, quando a distância (euclideana) entre eles não é maior do que um limite $2r$ pré-estabelecido. Quanto menor for o valor de r, mais restritiva será a idéia de similaridade. Em particular, se $r = 0$, então dois pontos são tidos como similares apenas se eles são idênticos. Repare que essa noção de similaridade, embora seja reflexiva (x é próximo a x) e simétrica (x próximo a $y \iff y$ próximo a x), não é transitiva, ou seja, se x é próximo a y e y próximo a z, não necessariamente temos x próximo a z (como contra-exemplo, considere os pontos x, y e z da Figura 6.1).

Podemos agrupar os pontos similares em "classes de similaridade" que correspondem a conjuntos de pontos no interior de círculos de raio r (Figura 6.1). Repare que, em conseqüência da não-transitividade da relação de similaridade, um determinado ponto

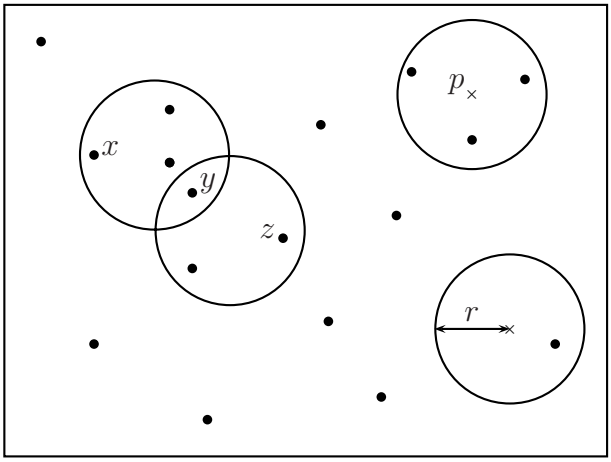


Figura 6.1. Classes de similaridade de pontos no plano.

pode pertencer a duas classes distintas (*e.g.* y na Figura 6.1). Denotamos por $B_r(p)$ o círculo de raio r com centro no ponto p. Como, no caso das classes de similaridade, cada um dos círculos está completamente determinado pelo seu centro p (o raio é fixo), podemos dizer que p "representa" uma classe de similaridade. Note-se que p não necessariamente pertence ao conjunto de pontos dados originalmente (*e.g.* ponto p na Figura 6.1). Parafraseando a nossa definição informal de similaridade, podemos dizer que, dado um conjunto de pontos, um sub-conjunto desses pontos constitui-se em uma classe de pontos similares entre si, se existe um ponto central p (não necessariamente pertencente ao conjunto de pontos dados) tal que a distância de cada um dos pontos do sub-conjunto a esse ponto central é menor ou igual ao limite pré-estabelecido e, além disso, todos os pontos fora do sub-conjunto estão a uma distância de p superior a esse limite.

6.1.2 Modelos

As considerações geométricas (sobremaneira intuitivas) sobre similaridade da subseção anterior aplicam-se *ipsis litteris*, se substituimos os pontos do plano por cadeias em Σ^*, e a distância euclideana por uma métrica qualquer, d, nesse espaço, *e.g.* a distância de Hamming (d_H) ou a distância de Levenshtein (d_L). Mais precisamente, adotaremos a seguinte definição de similaridade:

Definição 6.1 (Similaridade entre cadeias) *Dados uma métrica d em Σ^* e um inteiro $r \geq 0$, dizemos que duas cadeias $X, Y \in \Sigma^*$ são* r-similares *(ou simplesmente* similares*), se existe uma cadeia $M \in \Sigma^*$ tal que $d(M, X) \leq r$ e $d(M, Y) \leq r$. Em símbolos:*

$$X \simeq_r Y \iff \exists M \in \Sigma^* \text{ t.q. } d(M, X) \leq r \wedge d(M, Y) \leq r.$$

A definição de similaridade acima induz o conceito de "classe de similaridade" a seguir.

Definição 6.2 (Classe de similaridade) *Dados uma métrica d em Σ^*, um inteiro $r \geq 0$ e um conjunto de cadeias $X^* = \{X_1, \ldots, X_n\}$, dizemos que um subconjunto $\{X_{k_1}, \ldots, X_{k_m}\} \subset X^*$ é uma* classe de similaridade *de X^*, se existe uma cadeia $M \in \Sigma^*$* t.q. $\forall j \in \{k_1, \ldots, k_m\}, d(M, X_j) \leq r$ *e* $\forall j \in (\{1, \ldots, n\} \setminus \{k_1, \ldots, k_m\}), d(M, X_j) > r$.

A cadeia M na Definição 6.2 acima é denominada um *modelo*. Repare que um modelo é o equivalente a um ponto central de um círculo de equivalência no nosso exemplo geométrico.

Definição 6.3 (Ocorrência de um modelo em uma cadeia) *Dados uma métrica d em Σ^* e um inteiro $r \geq 0$, dizemos que um modelo $M \in \Sigma^*$ ocorre em uma cadeia $X \in \Sigma^*$, se existe um fator $Y \in \mathscr{F}(X)$* t.q. *$d(M, Y) \leq r$. Nesse caso, Y é dita uma* r-ocorrência *(ou, simplesmente, uma* ocorrência*) do modelo M em X.*

A Definição 6.3 acima pode ser, naturalmente, generalizada para um conjunto de cadeias.

Definição 6.4 (Ocorrência de um modelo em um conjunto de cadeias) *Dados uma métrica d em Σ^* e um inteiro $r \geq 0$, dizemos que um modelo $M \in \Sigma^*$ ocorre em um conjunto de cadeias $X^* = \{X_1, \ldots, X_n\}$, se M ocorre em pelo menos uma cadeia $X_j \in X^*$. Se M ocorre em q cadeias distintas de X^*, então q é dito o* quorum *de M em X^*.*

Definindo a *r-vizinhança* de M como sendo o conjunto de cadeias a uma distância não superior a r de M, *i.e.* $B_r(M) = \{Y \in \Sigma^* \mid d(M, Y) \leq r\}^*$, temos que o conjunto de r-ocorrências de M em X^* é dado por $\bigcup_{X_i \in X^*} (\mathscr{F}(X_i) \cap B_r(M))$.

6.2 MOTIFS SIMPLES

O primeiro problema de inferência que vamos examinar diz respeito aos *motifs* simples, ou seja, *motifs* constituídos de uma única cadeia.

Problema 1 (Inferência de *motifs* simples) *Dado um conjunto de cadeias não-alinhadas $X^* = \{X_1, \ldots, X_n\}$, um limite erro $r \geq 0$ e um* quorum *mínimo $2 \leq q \leq n$, determinar todos os modelos M que ocorrem em X^* com* quorum *maior ou igual a q.*

Um modelo é dito *válido*, se ele satisfaz as condições do Problema 1.

6.2.1 Estrutura de Dados

A solução do Problema 1 apresentada por Sagot [Sag98] baseia-se na utilização de um estrutura de dados conhecida como *árvore de sufixos generalizada*, definida, nos termos da Definição 3.5, da seguinte forma:

*Na terminlogia de espaços métricos, a *bola* de raio r centrada em M

Definição 6.5 (Árvore de sufixos generalizada) *Dado um conjunto de cadeias* $X^* = \{X_1, \ldots, X_n\}$, *uma* árvore de sufixos generalizada T *de* X^* *é uma árvore-*Σ^+ *tal que*

$$cadeias(T) = \bigcup_{X_i \in X^*} \mathscr{F}(X_i).$$

A Figura 6.2(a) ilustra a Definição 6.5 acima.

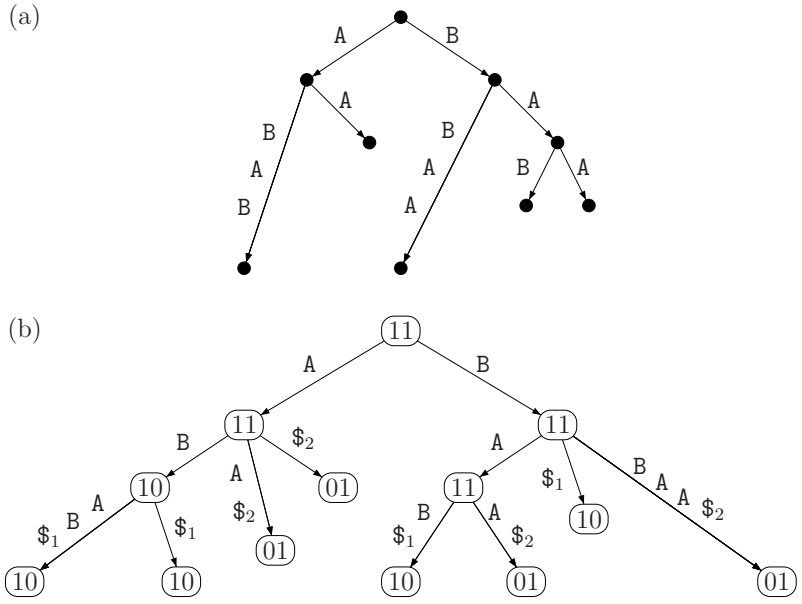


Figura 6.2. (a) Árvore de sufixos generalizada de $X^* = \{\texttt{ABAB}, \texttt{BBAA}\}$. (b) Árvore de sufixos generalizada de $X^* = \{\texttt{ABAB\$}_1, \texttt{BBAA\$}_2\}$ (os *loci* de $\$_1$ e $\$_2$ foram omitidos). Na figura, cada vértice v está rotulado com o padrão binário C_v correspondente.

Em geral, estaremos interessados na árvore de sufixos generalizada compacta de um conjunto de cadeias X^*, ou seja, $\textsc{cst}(X^*)$. É conveniente, para esta aplicação, que cada folha de $\textsc{cst}(X^*)$ represente um, e apenas um, sufixo de X^* (*i.e.*, um sufixo de X_i para algum $i = 1, \ldots, n$) e, reciprocamente, que cada sufixo de X^* seja representado por uma única folha de $\textsc{cst}(X^*)$. Essa situação ocorre, naturalmente, se cada sufixo de X^* não é aninhado em X^*, ou seja, $Y \sqsupset X_i \implies Y \notin \mathscr{F}(X_i) \setminus \mathscr{S}(X_i) \wedge Y \notin \mathscr{F}(X_j)$, para $j \neq i$. Para garantir que essa condição seja satisfeita, em geral adicionam-se caracteres sentinelas $\$_1, \ldots, \$_n \notin \Sigma$, distintos dois a dois, de forma a obter o conjunto $\{X_1\$_1, \ldots, X_n\$_n\}$ cuja árvore de sufixos generalizada compacta respeita as restrições acima (Figura 6.2(b)). Se $\textsc{cst}(X^*)$ respeita essas restrições, então cada vértice interno v representa um prefixo de mais de um sufixo de X^*. A sub-árvore enraizada em v possui folhas cujas arestas aferentes são rotuladas por sufixos da forma $Y\$_j$ para alguns valores de j em $\{0, \ldots, n\}$. Dessa forma, temos que a cadeia representada por v ocorre em cada uma das cadeias $X_j \in X^*$.

Um ponto chave para a solução que descreveremos a seguir é o pré-processamento de $\textsc{cst}(X^*)$ de maneira a associar, a cada vértice v, um padrão binário $C_v = c_1 \cdots c_n$ tal

que

$$c_i = \begin{cases} 1, \text{ se } \mathit{rót}(v) \in \mathscr{F}(X_i) \\ 0, \text{ do contrário.} \end{cases}$$

Essa anotação de $\text{CST}(X^*)$ pode ser feita, simplesmente, percorrendo-a em profundidade, compondo o vetor C_v a partir dos vetores dos seu filhos, conforme disposto no Algoritmo 6.1. O algoritmo é invocado pela primeira vez com $v =$ RAIZ de $\text{CST}(X^*)$.

```
1  Algoritmo anota_árvore (v)
2  início
3    se v é uma folha então
4      Seja Y$_j o rótulo da aresta aferente a v
5      C_v = c_1 ··· c_{j-1} c_j c_{j+1} ··· c_n ← 0 ··· 010 ··· 0
6    senão
7      C_v ← 0 ··· 0
8      para cada aresta v → w faça
9        anota_árvore(w)
10       C_v ← C_v || C_w
11     fim-faça
12   fim-se
13 fim
```

Algoritmo 6.1. Pré-processamento da árvore de sufixos generalizada. O parâmetro v corresponde à raiz da (sub-)árvore a ser anotada.

Em geral, a construção de árvores de sufixos generalizadas pode ser feita de maneira similar à construção das árvores de sufixos, simplesmente concatenando-se as cadeias a serem representadas e desconsiderando-se, dos rótulos das arestas aferentes às folhas, os sufixos iniciados após o primeiro caractere sentinela. Uma análise cuidadosa do processo de construção de McCreight (Seção 3.1.2.1), por exemplo, revela que a construção de uma árvore de sufixos gerenalizada por ser feita, de maneira incremental, construindo-se a árvore de sufixos de cada cadeia do conjunto sobre a árvore de sufixos acumulada das cadeias anteriores, essencialmente com o mesmo algoritmo. Uma construção mais eficiente é proposta por Bieganski *et al.* [BRCR94].

Definição 6.6 (Nó-ocorrência) *Dados uma métrica d em Σ^* e um inteiro $r \geq 0$, dizemos que o par (v, k) é uma* nó-ocorrência *de um modelo M em uma árvore de sufixos generalizada T de $X^* = \{X_1, \ldots, X_n\}$, se v é um vértice (possivelmente implícito) de T tal que $\mathit{rót}(v)$ é uma ocorrência de M em X^* (*i.e.*, uma ocorrência de M em alguma cadeia $X_i \in X^*$) com $d(M, \mathit{rót}(v)) = k \leq r$.*

Repare que uma nó-ocorrência de um modelo em uma árvore de sufixos de X^* corresponde, em geral, a um conjunto de ocorrências do modelo nas cadeias que constituem o conjunto X^*.

6.2.2 Algoritmo

A solução para o Problema 1 proposta por Sagot baseia-se no seguinte lema*:

Lema 6.1 *Dado um limite de substituições $r \geq 0$, o par (v, k) é uma nó-ocorrência do modelo $M' = Ma$ ($M \in \Sigma^m$, $a \in \Sigma$) em $\text{CST}(X^*)$ se, e somente se, uma das condições abaixo é satisfeita:*

*i) (*Coincidência*) Se $(pai(v), k)^\dagger$ é uma nó-ocorrência de M e a transição‡ $pai(v) \rightarrow v$ possui rótulo igual a a.*

*ii) (*Substituição*) Se $(pai(v), k-1)$ é uma nó-ocorrência de M, $k \leq r$ e a transição $pai(v) \rightarrow v$ possui rótulo $b \neq a$.*

Prova Imediato pelas definições de nó-ocorrência e distância de Hamming. ∎

Os modelos válidos de comprimento ℓ podem ser extraídos percorrendo-se (de forma simulada) a *trie* lexicográfica de todos os modelos possíveis§ até a profundidade ℓ, verificando, para cada cadeia considerada, a sua validade enquanto modelo, com o auxílio da árvore de sufixos generalizada anotada. Esse procedimento corresponde ao Algoritmo 6.2.

No Algoritmo 6.2, o parâmetro M corresponde a um modelo cuja ocorrência em X^* está comprovada, inclusive satisfazendo a restrição de *quorum*. O_M corresponde ao seu conjunto de nó-cocorrências em $\text{CST}(X^*)$. O algoritmo é inicialmente invocado com $M = \varepsilon$ e $O_M = \{(\text{RAIZ}, 0)\}$. Se o modelo atual possui o comprimento desejado (ℓ) então ele é considerado com uma das soluções do problema (linhas *3–4*). Senão, são consideradas todas as possíveis extensões de M por caracteres em Σ (os filhos de $\overline{M}$ na árvore lexicográfica). Para cada extensão Ma considerada, as suas nó-ocorrências são determinadas a partir das nó-ocorrências de M de acordo com o Lema 6.1 (linhas *7–22*). Se, finalmente, uma extensão satisfaz a restrição de *quorum*, então ela á considerada para efetivação ou extensão (linhas *24–26*). Uma última observação acerca do Algoritmo 6.2 diz respeito à linha *29*: se v' é um nó implícito, então $C_{v'}$ não existe explicitamente na árvore de sufixos anotada. Nesse caso, entretanto, temos $C_{v'} = C_w$, onde w corresponde ao nó explícito menos profundo que é "descendente" de v', ou seja, se $rót(v') = Y$, então w é locus da menor extensão de Y explicitamente representada em $\text{CST}(X^*)$.¶

*No restante da seção, por simplicidade na apresentação, adotaremos a distância de Hamming como a métrica de Σ^*. A solução do mesmo problema levando-se em conta a distância de edição é esboçada em [Sag98].

†O nó (implícito) u é dito o pai de um nó (implícito) v em uma árvore-Σ^+ T, se o nó explícito correspondente a u é o pai do nó explícito correspondente a v na árvore-Σ^+ atômica equivalente a T. Nesse caso, escrevemos $pai(v) = u$.

‡Repare que utilizamos o termo "transição" e não "aresta" (a exemplo da Seção 3.1.2.2) pois estamos tratando de nós possivelmente implícitos. Todavia, abusamos da notação e denotamos por $u \rightarrow v$ a transição de u para v.

§ *Trie* na qual cada nó possui $|\Sigma|$ arestas eferentes, cada uma delas rotulada por um elemento de Σ.

¶McCreight [McC76] denomina w o *locus estendido* de Y na árvore de sufixos.

```
1  Algoritmo motifs_simples (M = m_1 ··· m_m, O_M)
2  início
3    se m = ℓ então
4      𝕊 ← 𝕊 ∪ {M}
5    senão-se m < ℓ então
6      para cada a ∈ Σ faça
7        O_Ma ← ∅
8        C_Ma ← 0 ··· 0
9        para cada (u, k) ∈ O_M faça
10         para cada transição u --b∈Σ--> u' faça
11           % u e u' são, possivelmente, implícitos
12           se b ≠ a então
13             k' ← k + 1
14           senão
15             k' ← k
16           fim-se
17           se k' ≤ r então
18             O_Ma ← O_Ma ∪ {(u', k')}
19             C_Ma ← C_Ma || C_v'
20           fim-se
21         fim-faça
22       fim-faça
23     fim-faça
24     se quantidade de bits 1 em C_Ma ≥ q então
25       motifs_simples(Ma, O_Ma)
26     fim-se
27   fim-se
28 fim
```

Algoritmo 6.2. Inferência de *motifs* simples de comprimento ℓ.

6.2.3 Complexidade

Teorema 6.1 *Considerando um conjunto de cadeias $X^* = \{X_1, \ldots, X_n\}$ de comprimento (médio) t e um limite de r substituições por ocorrência, a inferência de modelos simples de comprimento ℓ demanda tempo $O(\ell t n^2 V_H(r,\ell))^*$, onde $V_H(r,\ell)$ denota o número de elementos em uma r-vizinhança $B_r(Y)$ de uma cadeia Y de comprimento ℓ, com respeito à distância de Hamming.*

Prova O custo total da solução para o problema da inferência de *motifs* simples aqui apresentada corresponde ao custo da construção da árvore de sufixos generalizada $\textsc{cst}(X^*)$, mais o custo do pré-processamento dessa estrutura (Algoritmo 6.1), mais o custo da execução do Algoritmo 6.2.

A construção de $\textsc{cst}(X^*)$ pode ser feita, conforme discutido anteriormente, a partir da cadeia $X_1X_2\cdots X_n$, em tempo linear, ou seja, $O(tn)$. O pré-processamento dessa estrutura (Algoritmo 6.1) compreende uma visita a cada um dos seus $O(tn)$ vértices. A visita ao nó v implica na atualização de C_v (linha *10*), no máximo, $|\Sigma|$ vezes (número máximo de filhos de um vértice em uma árvore-Σ^+). Se C_v é implementado como um vetor de *bits* para uma palavra de máquina de comprimento w, então o C_v é composto de $\lceil \frac{n}{w} \rceil$ palavras, cada uma das quais podendo ser atualizada em tempo constante (em virtude do paralelismo binário). Logo, o custo total do pré-processamento de $\textsc{cst}(X^*)$ é $O(tn|\Sigma|\lceil \frac{n}{w} \rceil) = O(tn^2)$, supondo-se $|\Sigma|$ fixo.

Extrair um modelo válido M de comprimento ℓ significa percorrer caminhos em $\textsc{cst}(X^*)$ da RAIZ até vértices de profundidade ℓ, soletrando, a cada nível, um caractere de M (todos os vértices de profundidade ℓ assim alcançados, são nó-ocorrências de M). Para cada vértice v t.q. $\mathit{prof}(v) = \ell$, existem $V_H(r,\ell)$ maneiras distintas de soletrar um modelo M de forma que v seja uma nó-ocorrência de M. Portanto, cada vértice v de profundidade ℓ pode ser visitado, no máximo, $V_H(r,\ell)$ vezes (determinados percursos RAIZ $\rightsquigarrow v$ podem ser precocemente interrompidos, se o *quorum* mínimo não já for satisfeito em uma profundidade $< \ell$). Existem $s_\ell \leq tn$ vértices de profundidade ℓ e, portanto, o número de visitas a vértices de profundidade ℓ é limitado por $tnV_H(r,\ell)$. Cada visita a um vértice v de profundidade ℓ implica, na realidade, na visita de ℓ vértices no caminho RAIZ $\rightsquigarrow v$, cada uma dessas visitas demandando tempo $O(\lceil \frac{n}{w} \rceil) = O(n)$ (linha *19*). Portanto, o custo total das visitas a vértices de $\textsc{cst}(X^*)$ ao longo da execução do algoritmo é $O(\ell t n^2 V_H(r,\ell))$. Finalmente, a verificação da linha *24* demanda, no máximo, $O(n)$ operações por modelo. Como existem, no máximo, $n(t-\ell)V_H(r,\ell) \leq tnV_H(r,\ell)$ modelos possíveis, então o custo acumulado dessa operação é $O(tn^2 V_H(r,\ell))$.

Sumarizando, o custo total da inferência de *motifs* simples é, no pior caso, $O(tn + tn^2 + \ell t n^2 V_H(r,\ell) + tn^2 V_H(r,\ell)) = O(\ell t n^2 V_H(r,\ell))$. ∎

Teorema 6.2 *Nas mesmas condições do Teorema 6.1, a inferência de modelos simples requer $O((\ell + n)tn)$ espaço.*

Prova A árvore de sufixos generalizada de X^* requer, como sabemos, $O(\|X^*\| = tn)$ espaço. Cada vetor C_v ocupa $O(\lceil \frac{n}{w} \rceil)$ espaço, onde w corresponde ao tamanho da palavra de máquina. Portanto, a anotação de $\textsc{cst}(X^*)$ implica em $O(tn\lceil \frac{n}{w} \rceil) = O(tn^2)$ espaço

*Sagot [Sag98] apresenta uma prova para o limite mais justo $O(tn^2 V_H(r,\ell))$.

adicional. Em cada instante do algoritmo, o espaço requerido para a armazenagem de todas as ocorrências de um modelo válido de comprimento $\leq \ell$ e de todos os seus prefixos não pode ser superior a $O(\ell tn)$ ∴ o requisito total de espaço para a inferência de *motifs* simples é $O(tn + tn^2 + \ell tn) = O((\ell + n)tn)$. ∎

6.3 MOTIFS ESTRUTURADOS

Nesta seção, vamos examinar o problema da inferência dos *motifs* ditos *estruturados*, ou seja, *m*otifs constituídos de mais de uma cadeia separadas por intervalos de comprimento restrito.

6.3.1 Modelos Estruturados

Definição 6.7 (Modelo estruturado) *Um* modelo estruturado *é um par* (M^*, D^*) *de forma que*

- M^* *é uma p-tupla de modelos simples (denominados* blocos*)* $M^* = (M_1, \ldots, M_p)$
- D^* *é uma* $(p-1)$*-tupla de triplas* $D^* = ((d_{min_1}, d_{max_1}, \delta_1), \ldots, (d_{min_{p-1}}, d_{max_{p-1}}, \delta_{p-1}))$

com $p \in \mathbb{Z}_+$, $d_{min_i}, d_{max_i} (d_{min_i} \leq d_{max_i}), \delta_i \in \mathbb{Z}_+^*$.

Definição 6.8 (Ocorrência de um modelo estruturado em uma cadeia) *Dados uma cadeia* $X = x_1 \cdots x_n \in \Sigma^*$, *uma métrica d em* Σ^* *e um limite de erro* $r \geq 0$, *dizemos que o modelo estruturado de p blocos* (M^*, D^*) *ocorre em X se existem ocorrências* $Y_1 = y_1^1 \cdots y_{l_1}^1, \ldots, Y_p = y_1^p \cdots y_{l_p}^p$ *consecutivas e disjuntas de* $M_1, \ldots, M_p$ *em X tais que*

$$y_{l_j}^j = x_t \text{ e } y_1^{j+1} = x_{t'} \implies d_{min_j} \leq (t' - t - 1) \leq d_{max_j}, \ \forall j = 1, \ldots, p-1.$$

Definição 6.9 (Modelo estruturado válido) *Dado um conjunto de seqüências* $X^* = \{X_1, \ldots, X_n\}$, *uma métrica d em* Σ^*, *um limite de erro* $r \geq 0$ *e um* quorum *mínimo q, dizemos que o modelo estruturado* (M^*, D^*) *é* válido *em* X^* *se ele ocorre em* $k \geq q$ *cadeias distintas de* X^* *de forma que, para cada lista de ocorrências* $Y_1 = y_1^1 \cdots y_{l_1}^1, \ldots, Y_p = y_1^p \cdots y_{l_p}^p$ *de* $M_1, \ldots, M_p$, *temos*

- *Para cada* $i = 1, \ldots, p-1$ *existe um* $d_i, d_{min_i} + \delta_i \leq d_i \leq d_{max_i} - \delta_i$ *tal que a "distância" que separa o final de* Y_i *e o início de* Y_{i+1} *é igual a* $d_i \pm \delta_i$, *ou seja,*

$$y_{l_i}^i = x_t \text{ e } y_1^{i+1} = x_{t'} \implies d_{min_i} + \delta_i \leq (t' - t - 1) = d_i \leq d_{max_i} - \delta_i;$$

- *Para todo* $i = 1, \ldots, p-1$, d_i *é o mesmo para as k ocorrências do modelo.*

Para que um modelo estruturado seja considerado válido, não basta que ele ocorra em em uma quantidade suficiente de cadeias. É também necessário que a distância que separa dois de seus blocos constituintes consecutivos não varie muito ao longo dessas ocorrências, mas se mantenha dentro de um sub-intervalo, respeitando, é claro, os limites mínimo e máximo permitidos. A Figura 6.3 ilustra a Definição 6.9 acima.

No caso particular em que $\delta_i = (d_{max_i} - d_{min_i})/2$ (ou seja, a distância entre blocos consecutivos pode variar livremente no intervalo $d_{min_i} \ldots d_{max_i}$), escrevemos, simplesmente, $(M^*, D^*) = ((M_1, \ldots, M_p), \ ((d_{min_1}, d_{max_1}), \ldots, (d_{min_{p-1}}, d_{max_{p-1}})))$.

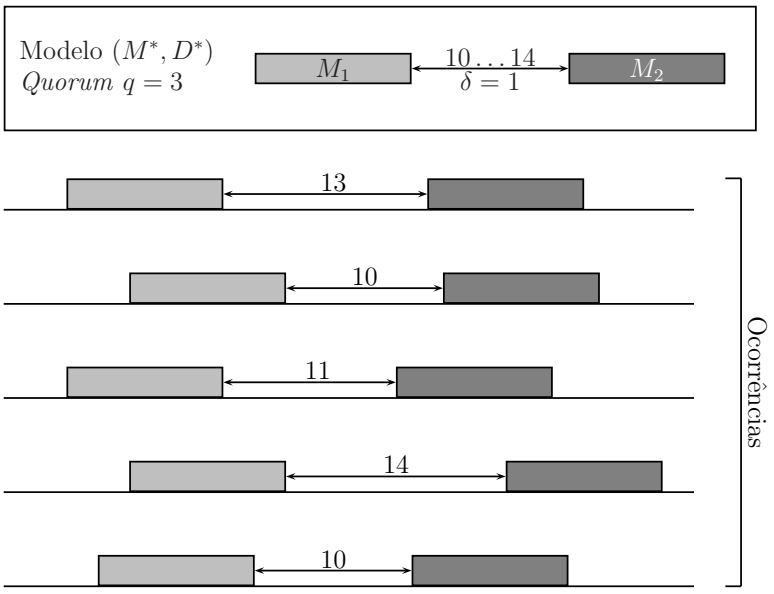


Figura 6.3. No exemplo, estamos considerando a validade do modelo estruturado composto por dois blocos $(M^*, D^*) = ((M_1, M_2), (10, 14, 1))$ para um *quorum* mínimo $q = 3$. Temos, portanto, três sub-intervalos possíveis para a distância entre os blocos M_1 e M_2, quais sejam, 11 ± 1, 12 ± 1 e 13 ± 1. Para $d_1 = 11$, temos a segunda, a terceira e a última ocorrência, satisfazendo, assim, o *quorum* mínimo. Para $d_1 = 12$, temos apenas a primeira e a terceira ocorrência. Para $d_1 = 13$ temos apenas a primeira e a quarta ocorrência. Portanto, o modelo é válido apenas para $d_1 = 11$. (Exemplo baseado em [Mar02])

6.3.2 Problema

Marsan e Sagot [MS00] propõem um série de problemas relacionados à inferência de *motifs* (modelos) estruturados em ordem crescente de complexidade e generalidade, culminando no seguinte enunciado:

Problema 2 (Inferência de *motifs* estruturados) *Dado um conjunto de cadeias não-alinhadas* $X^* = \{X_1, \ldots, X_n\}$, *um limite de erro* $r \geq 0$ *e um* quorum *mínimo* $2 \leq q \leq n$, *determinar todos os modelos estruturados válidos de* p *blocos* $(M^*, D^*) = ((M_1, \ldots, M_p),\ ((d_{min_1}, d_{max_1}, \delta_1), \ldots, (d_{min_{p-1}}, d_{max_{p-1}}, \delta_{p-1})))$.

Vamos examinar em detalhes a solução para o caso particular apontado ao final da subseção anterior, quando temos $\delta_i = (d_{max_i} - d_{min_i})/2$. O caso geral pode ser tratado de maneira similar com umas poucas modificações no algoritmo a ser apresentado (mais detalhes podem ser encontrados em [MS00, Mar02, SW02])*.

*Para efeito de simplicidade na apresentação, vamos, mais uma vez, considerar a distância de Hamming como métrica de Σ^* e supor que todos os modelos simples constituintes do modelo estruturado obedecem às mesmas restrições locais (comprimento ℓ e máximo número de substituições r).

6.3.3 Estrutura de Dados

A solução proposta por Marsan-Sagot [MS00] para o Problema 2 também baseia-se no emprego da árvore de sufixos generalizada do conjunto de cadeias dado*.

Embora o conceito de nó-ocorrência de um modelo simples seja, de certa forma, intuitivo, a idéia de uma nó-ocorrência de um modelo estruturado é um pouco mais sutil. Intuitivamente, uma ocorrência de um modelo estruturado $M^* = (M_1, \ldots, M_p)$ em uma determinada cadeia X representa uma ocorrência de M_1 seguida (alguns caracteres depois) por uma ocorrência de M_2, e assim sucessivamente até uma ocorrência de M_p. O fator $Y \subset X$ iniciado no primeiro caractere da ocorrência de M_1, estendendo-se até o último caractere da ocorrência de M_p, corresponde, como um todo, a uma ocorrência do modelo estruturado. Portanto, nada mais natural que o *locus* de Y na árvore de sufixos correspondente represente uma nó-ocorrência do modelo estruturado. Entretanto, a menos que estejamos interessados em impor uma restrição global sobre o erro acumulado ao longo das ocorrências dos p blocos, não faz muito sentido associar um valor de erro a esta nó-ocorrência (ao contrário dos modelos simples). Outra coisa importante a ser observada, é que a ocorrência de um bloco M_i corresponde a um trecho (de comprimento ℓ) do caminho RAIZ $\rightsquigarrow \overline{Y}$ e que o vértice que representa o início do trecho correspondente a M_{i+1} é um descendente do vértice que representa o fim do trecho correspondente a M_i, estando aquele de d_{min_i} a d_{max_i} vértices "acima" deste. A Figura 6.4 ilustra essas observações no caso $p = 2$.

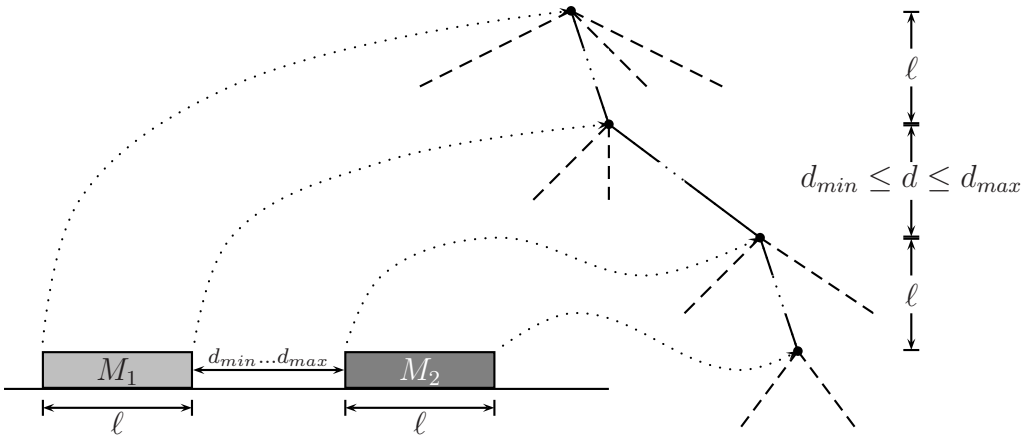


Figura 6.4. A nó-ocorrência do modelo estruturado $((M_1, M_2), (d_{min}, d_{max}))$ é um descendente da nó ocorrência de M_1 e diferença entre as profundidades desses nós situa-se no intervalo $\ell + d_{min} \ldots \ell + d_{max}$.

A partir das observações precedentes, formalizamos o lema a seguir.

Lema 6.2 *Se o vértice v é uma nó-ocorrência de um modelo estruturado $(M_1, \ldots, M_p)$ de $p \geq 2$ blocos de comprimento ℓ em* AST(X^*)*, então existe um ancestral u de v tal*

*Também para efeito de simplicidade na apresentação, vamos considerar a árvore de sufixos generalizada atômica de X^*, AST(X^*), ao invés da sua árvore de sufixos compacta.

que $\ell + d_{min_{p-1}} \leq prof(v) - prof(u) \leq \ell + d_{max_{p-1}}$ *e* u *é uma nó-ocorrência do modelo estruturado* $(M_1, \ldots, M_{p-1})$

6.3.4 Algoritmo

O Algoritmo 6.3 faz uso dos lemas 6.2 e 6.1 para inferir, de maneira incremental, os modelos estruturados $(M_1), (M_1, M_2), \ldots, (M_1, \ldots, M_p)$, da maneira descrita a seguir. Primeiramente, um modelo simples M_1 é inferido sobre AST(X^*) através do Algoritmo 6.2, sendo suas nó-ocorrências armazenadas em O_{M_1}. Pelo Lema 6.2, as nó-ocorrências de um modelo estruturado (M_1, M_2), para um certo M_2, são descencentes dos vértices de O_{M_1} situados de $\ell + d_{min_1}$ a $\ell + d_{max_1}$ vértices "abaixo" na árvore de sufixos. Esses descendentes são todos armazenados em uma lista N de *potenciais* nó-ocorrências de (M_1, M_2). Cada vértice $v' \in N$ representa um fator Y de X^* tal que $2\ell + d_{min_i} \leq |Y| = m \leq 2\ell + d_{max_i}$. Se v' é, de fato, uma nó-ocorrência de um modelo estruturado (M_1, M_2), então $v'' = \overline{Y_{m-\ell+1..}}$ é uma nó-ocorrência de M_2 em X^*. v'' é um vértice de profundidade ℓ que pode ser alcançado seguindo-se o caminho de *suffix links* a partir de v'. Se considerarmos a "sub-árvore" T_2 de AST(X^*) enraizada na própria RAIZ de AST(X^*) e cujas folhas são todos os v'' assim determinados a partir dos elementos de N, então os modelos M_2 que estendem o modelo M_1 podem ser todos determinados sobre T_2 através do Algoritmo 6.2. Uma vez inferido um modelo M_2 que estende o modelo M_1, as extensões M_3 são consideradas de maneira análoga, e assim sucessivamente, até que um modelo $(M_1, \ldots, M_i)$ $(i < p)$ não possa ser mais estendido ou até que tenhamos uma tupla válida $(M_1, \ldots, M_p)$, quando então a recursão retrocede.

Um detalhe sutil, porém de fundamental importância, diz respeito à árvore T_i, mencionada no parágrafo anterior, cujas folhas correspondem a vértices $v'' = \overline{Y_{m-\ell+1..}}$ de profundidade ℓ alcançáveis a partir dos elementos $v' = \overline{Y} = \overline{y_1 \cdots y_m} \in N$ por caminhos de *suffix links*. Na realidade, $C_{\overline{Y_{m-\ell+1..}}}$ indica *todas* as ocorrências de $Y_{m-\ell+1..}$ em X^*, e não apenas aquelas que sucedem ocorrências de $(M_1, \ldots, M_{i-1})$. Portanto, é necessário "transportar" os valores de $C_{v'}$ para os respectivos $C_{v''}$ (linha *15*) e, antes de inferir M_i, atualizar os valores de C_w para todos os nós w de T_i, o que pode ser feito de maneira análoga ao Algoritmo 6.1 (linha *18*). Essa atualização, obviamente, desconfigura a árvore original T_{i-1} sendo necessário, portanto, salvaguardar os valores originais de $C_{v''}$ em uma lista L_{i-1} (linha *9*) para a posterior restauração daquela árvore (linha *36*). Um outro detalhe importante é que as nó-ocorrências de um modelo M_i, determinadas sobre T_i pelo Algoritmo 6.2 (linha *23*), na realidade, não são as veradeiras nó-ocorrências de $(M_1, \ldots, M_i)$ e, portanto, não podem ser diretamente utilizadas para a composição de N. Ao invés disso, cada folha u $(\in F)$ de T_i (vértice de profundidade ℓ em AST(X^*)) possui uma lista de apontadores $P_{i-1}(u)$ (linha *16*) para os vértices (de profundidade entre $i\ell + (i-1)d_{min_{i-1}}$ e $i\ell + (i-1)d_{max_{i-1}}$) que o determinam (através de um caminho de *suffix links*). Por fim, observamos que, como o algoritmo está sempre percorrendo caminhos de *suffix links* partindo de vértices v' de profundidade $i\ell + (i-1)d_{min_{i-1}} \leq prof(v') \leq i\ell + (i-1)d_{max_{i-1}}$, $\forall i = 1, \ldots, p$, até vértices v'' de profundidade ℓ, então a AST(X^*) pode ser pré-processada de forma que esses caminhos sejam substituídos por "atalhos" que denominamos *suffix-links rápidos* (linha *8*).

Algoritmo *modelos_estruturados* $((M^*, D^*), O_{M^*})$
início
  se $i > 1$ **então**
    para cada nó-ocorrência $v = P_{i-1}(u)$ com $u \in O_{M^*}$ **faça**
      $N \leftarrow \{v' \mid v'$ descendentes de $v \wedge$
        $i\ell + (i-1)d_{min_{i-1}} \leq prof(v) \leq i\ell - (i-1)d_{max_{i-1}}\}$
    fim-faça
    para cada $v' \in N$ **faça**
      $v'' \leftarrow$ vértice de prof. ℓ atingido a partir de v' através do *suffix-link* rápido
      $L_{i-1} \leftarrow L_{i-1} \cup \{C_{v''}\}$
      se esta é a primeira visita a v'' **então**
        $C_{v''} \leftarrow 0 \cdots 0$
        $P_i(v'') \leftarrow \varnothing$
        $F \leftarrow F \cup \{v''\}$
      fim-se
      $C_{v''} \leftarrow C_{v''} \mid\mid C_{v'}$
      $P_i(v'') \leftarrow P_i(v'') \cup \{v'\}$
    fim-faça
    $T_i \leftarrow$ árvore anotada cujas folhas estão em F (análogo ao Algoritmo 6.1)
    $T \leftarrow T_i$
  senão
    $T \leftarrow \text{AST}(X^*)$
  fim-se
  para cada modelo válido M_i obtido a partir de T pelo Algoritmo 6.2 **faça**
    se $i = 1$ **então**
      para cada $u \in O_{M_1}$ **faça**
        $P_i(u) \leftarrow \{u\}$
      fim-faça
    fim-se
    se $i < p$ **então**
      modelos_estruturados$((M^*, D^*) = ((M_1, \ldots, M_i), ((d_{min_1}, d_{max_1}), \ldots, (d_{min_i}, d_{max_i}))),$
        $O_{M^*}, i+1)$
    senão
      $\mathbb{S} \leftarrow \mathbb{S} \cup \{((M_1, \ldots, M_p), ((d_{min_1}, d_{max_1}), \ldots, (d_{min_i}, d_{max_p})))\}$
    fim-se
  fim-faça
  se $i > 1$ **então**
    Restaure a árvore T_{i-1} com a ajuda de L_{i-1} (análogo ao Algoritmo 6.1)
  fim-se
fim

Algoritmo 6.3. Inferência de *motifs* estruturados.

6.3.5 Complexidade

Os seguintes teoremas oferecem estimativas para o custo computacional da solução do problema da inferência de *motifs* estruturados aqui apresentada, tanto em termos de tempo quanto em termos de espaço. As demonstrações serão omitidas e podem ser encontradas em [MS00].

Teorema 6.3 *Para extrair, a partir de um conjunto* $X^* = \{X_1, \ldots, X_n\}$ *de cadeias de comprimento (médio)* t*, todos os modelos estruturados de* p *blocos de comprimento* ℓ*,* $((M_1, \ldots, M_p), ((d_{min_1}, d_{max_1}), \ldots, (d_{min_p}, d_{max_p})))$*, com um limite de* r *substituições por bloco, e supondo um alfabeto de tamanho fixo, o Algoritmo 6.3 requer, no pior caso, tempo* $O(t \cdot n^2 + n \cdot s_\ell \cdot V_H^p(r, \ell) + n \cdot s_{p\ell+(p-1)d_{max}} \cdot V_H^{p-1}(r, \ell))$*, onde* s_j *denota o número de nós de* $\text{AST}(X^*)$ *de profundidade* j*.*

Teorema 6.4 *Nas mesmas condições do Teorema 6.3, o Algoritmo 6.3 demanda espaço* $O((p \cdot (\ell + d_{max}) + n) \cdot t \cdot n + s_{p\ell+(p+1)d_{max}} \cdot n \cdot (p-1))$

6.4 DISCUSSÃO SOBRE ESTRUTURAS DE DADOS

6.4.1 Árvores de Sufixos

As árvores de sufixos têm sido exaustivamente estudadas, tendo suas virtudes salientadas na literatura especializada [Apo85]. Com efeito, as árvores de sufixos conseguem reunir um alto poder expressivo aliado a uma boa eficiência tanto em termos do espaço ocupado quanto com respeito ao tempo de construção (ambos de comportamento assintótico linear). Esse poder e elegância das árvores de sufixos têm feito dessa estrutura a escolha natural em diversas aplicações, incluindo as soluções para o problema da inferência de *motifs* apresentadas neste capítulo.

Com respeito aos algoritmos para inferência de *motifs* aqui apresentados, Marsan [Mar02] observa que:

> (...) a estrutura de árvores de sufixos (...) não é indissociável desses algoritmos de extração e pode ser substituída por qualquer outra estrutura que permita:
>
> - Um percurso recursivo eficaz, ou seja, poder passar, em tempo constante, de uma ocorrência W a uma ocorrência Wa (com $W \in \Sigma^+$ e $a \in \Sigma$) e inversamente;
> - A avaliação em tempo constante do número de ocorrências de um dado modelo (possível na árvore através de ligeiras modificações durante a sua construção);
> - A determinação das posições de um fator (mesma observação).

À luz dessas observações, fica claro porque uma escolha natural recai sobre as árvores de sufixos.

Uma crítica freqüente às árvores de sufixos diz respeito ao espaço ocupado. Embora linear no pior caso, o requisito de espaço dessa estrutura ainda é alvo de restrições,

sobretudo se consideramos aplicações (*e.g.* Biologia Computacional) nas quais o volume de dados (*i.e.* o comprimento total das cadeias de entrada) é muito grande. Uma medida comumente utilizada para a aferir a eficiência de estruturas de índices em termos do espaço ocupado é dada pela razão

$$\rho = \frac{\text{Tamanho total da estrutura (em } \textit{bytes})}{\text{Comprimento total da(s) cadeia(s) representada(s)}},$$

que fornece uma taxa média de *bytes* por caractere de entrada.

Sabemos que, para cadeias do tipo $X = x_1 \cdots x_{n-1}\$$, a árvore de sufixos compacta possui, exatamente n folhas e, no máximo, $n-1$ vértices internos (a RAIZ também ramifica), perfazendo um total de, no máximo, $2n-1$ vértices. Na implementação mais eficiente proposta por McCreight [McC76] (Figura 6.6(a)) esse valor corresponde a uma taxa $\rho = 28$bpc (*bytes* por caractere). Kurtz [Kur99] aponta uma curiosa inconsistência entre os alegados valores de ρ para diversas implementações de árvores de sufixos propostas na literatura. No mesmo artigo, Kurtz propõe uma implementação na qual os nós internos são representados por uma lista de (no máx. $n-1$) registros compostos de cinco inteiros cada e as n folhas são representadas em uma simples lista de inteiros (Figura 6.6(b)) . Essa respresentação corresponde a uma taxa de 24bpc* no pior caso. Explorando uma série de redundâncias da árvore de sufixos, Kurtz consegue uma representação ainda mais sucinta, chegando a obter uma taxa $\rho = 20$bpc no pior caso e $\rho = 10.1$bpc no caso médio, comprovada empiricamente por experimentos com mais de 40 arquivos de diferentes tipos e tamanhos.

6.4.2 (C)DAWGs

Os DAWGs podem ser utilizados de maneira bastante similar às árvores de sufixos para a resolução do problema da inferência de *motifs*. Com efeito, primeiramente observamos que, a exemplo do que foi feito com as árvores de sufixos, os DAWGs podem ser estendidos para representar um conjunto de cadeias† (Figura 6.5). Mais ainda, como DAWG(X^*) (ou CDAWG(X^*)) é acíclico e pode ser construído de tal forma que cada cadeia $X_i \in X^*$ determine um sorvedouro‡, então DAWG(X^*) pode ser anotado de maneira bastante similar ao Algoritmo 6.1, percorrendo-se, em profundidade, todas as arestas do grafo.

Uma ressalva a ser feita aqui diz respeito ao problema da inferência de *motifs* estruturados. Embora os DAWGs não possuam uma estrutura de *suffix links* que permitam "retroceder a uma certa profundidade" de maneira óbvia, os *motifs* estruturados ainda podem ser inferidos de maneira similar ao Algoritmo 1 (saltando na árvore de sufixos) apresentado em [MS00].

Uma vez demonstrada a adequação dos DAWGs ao problema da inferência de *motifs*, resta-nos analisar a eficiência dessas estruturas em termos do espaço ocupado, sobretudo em comparação com as árvores de sufixos. Autômatos são, classicamente, implementados

*Supomos um inteiro ocupando 4 *bytes*.

†De fato, os DAWGs compactos foram introduzidos já nessa forma mais geral [BBHM87].

‡Basta recorrer aos caracteres sentinelas.

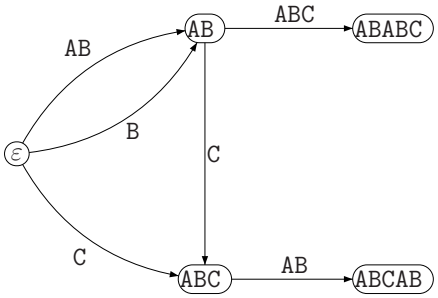


Figura 6.5. CDAWG generalizado de $X^* = \{\texttt{ABABC}, \texttt{ABCAB}\}$.

```
1  Algoritmo anota_dawg (v)
2  início
3    se v é um sorvedouro então
4      Seja Y$_j o repr. canônico de v
5      C_v = c_1 ··· c_{j-1} c_j c_{j+1} ··· c_n ← 0 ··· 010 ··· 0
6    senão
7      C_v ← 0 ··· 0
8      para cada aresta v → w com w não-visitado faça
9        anota_dawg(w)
10       marque o vértice w como visitado
11       C_v ← C_v || C_w
12     fim-faça
13   fim-se
14 fim
```

Algoritmo 6.4. Pré-processamento do DAWG generalizado.

através das chamadas *matrizes de transição* ou através das *listas de adjacências* [CLR89, Cap. 23]. Uma matriz de transição $A = (a_{ij})$ é uma matriz $|V| \times |\Sigma|$, onde $|V|$ denota o número de estados (vértices) do autômato (grafo) e $|\Sigma|$ o tamanho do alfabeto. A entrada a_{ij} contém o estado alcançado a partir do i-ésimo vértice, seguindo-se a transição rotulada pelo j-ésimo caractere do alfabeto. Dessa forma, as transições podem ser determinadas em tempo constante mas, se o alfabeto for longo e a matriz esparsa, então podemos ter um significativo desperdício de espaço. Alternativamente, podemos ter, para cada vértice u, uma lista encadeada na qual cada elemento indica um caractere $a \in \Sigma$ e um vértice v de forma que a transição $u \xrightarrow{a} v$ pertence ao grafo. Se as listas forem ordenadas pelos rótulos das transições, então estas podem ser determinadas em tempo logarítmico no tamanho do alfabeto, por exemplo, por busca binária. No caso de alfabetos pequenos (*e.g.*, o alfabeto de nucleotídeos $\{\texttt{A}, \texttt{C}, \texttt{G}, \texttt{T(U)}\}$), então, geralmente, a matriz de transição é a melhor escolha [CV97a].

O número de vértices de $\textsc{dawg}(X)$ é compatível com o número de vértices de $\textsc{cst}(X)$ (afinal, $\textsc{dawg}(X)^R \simeq \textsc{cst}(X^R)$), sendo que naquela estrutura, as arestas ainda apresentam a conveniência de serem atômicas. Na prática, entretanto, os DAWGs ainda apresentam um maior consumo de memória, sendo essa desvantagem amenizada através da compactação dos DAWGs em CDAWGs. Crochemore e Vérin [CV97a] propõem imple-

mentações para DAWG(X) e CDAWG(X) através de matrizes de transição que apresentam taxas $\rho = 27,8$bpc e $\rho = 22,78$bpc no caso médio, com base nas estimativas teóricas fornecidas por Blumer *et al.* [BEH89] acerca do número de estados dessas estruturas em função de $|X|$, supondo-se um alfabeto de tamanho 4. Esses valores demonstram que, também sob o aspecto do espaço requerido, os (C)DAWGs são estruturas viáveis e competitivas com a implementação de McCreight das árvores de sufixos. A implementação de Kurtz, todavia, mostra-se superior, mesmo se considerarmos que a anotação da estrutura afeta mais as árvores de sufixos (em relação aos CDAWGs) pelo fato de possuirem um maior número de vértices.

6.4.3 Vetores de Sufixos

A principal (única?) vantagem dos vetores de sufixos e estruturas afins (vetores de sufixos compactos, árvore de busca binária de sufixos [IL00], *etc*) sobre as árvores de sufixos é a economia de espaço. Com efeito, Manber e Myers [MM93] demonstram que os vetores de sufixos anotados com informações sobre máximos prefixos comuns apresentam uma taxa $\rho = 9$bpc no pior caso e no caso médio, o que representa, de fato, uma economia substancial sobre as demais estruturas.

Nada obstante a patente superioridade no quesito espaço, os vetores de sufixos carecem de poder expressividade e versatilidade. Com efeito, com respeito aos três requisitos levantados por Marsan e descritos acima, os vetores de sufixos, a princípio, só atendem, razoavelmente, ao terceiro (que é justamente o problema de busca para o qual eles foram projetados).

6.4.4 Estruturas Duais

Embora as estruturas duais consideradas neste trabalho, ou seja, as árvores de afixos e os SCDAWGs possam, perfeitamente, substituir os seus correspondentes não-duais, ou seja, as árvores de sufixos e os CDAWGs respectivamente, essa substituição implica em um maior consumo de espaço e em uma maior complexidade no processo de construção da estrutura de índice. Em boa parte das aplicações, esse custo extra não se justifica, incluindo-se aqui, o caso geral da inferência de *motifs* simples ou estruturados. Todavia, existe uma situação onde vislumbramos a possibilidade de tirar proveito da dualidade intrínseca dessas estruturas, qual seja, na inferência de *motifs* palíndromos.

No restante da subseção, estaremos tratando da inferência de *motifs* que apresentam uma certa feição palindrômica. O tom empregado nessa discussão será peculiarmente informal, uma vez que as idéias aqui apresentadas ainda não foram devidamente elaboradas nem tampouco implementadas. O princípio básico do raciocínio é inspirado em [SV97, Sag02]. Por fim, cumpre salientar que as observações aqui emitidas dizem respeito, a princípio, a árvores de afixos embora, com as devidas adaptações, apliquem-se, da mesma forma, aos DAWGs simétricos.

Inferir palíndromos é um problema de interesse algorítmico em geral, e de particular biológico uma vez que a ocorrência desse tipo de *motif* em uma molécula de RNA, por exemplo, pode ser um indicativo de alguma conformação estrutural característica como um laço. A exemplo do que tínhamos para os *motifs* genéricos, também temos a definição

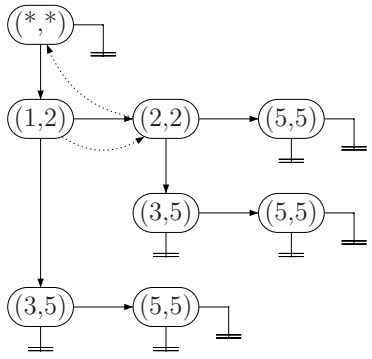
de um modelo para *motifs* palindrômicos*

Definição 6.10 *Dados uma métrica d em Σ^* e os inteiros não-negativos r, d_{min} e d_{max}, uma cadeia $M \in \Sigma^+$ é dita um* modelo para um palíndromo aproximado *em uma cadeia $X \in \Sigma^+$, se existem dois fatores $U, V \in \mathscr{F}(X)$ com $X = YUZVW$ ($Y, Z, W \in \Sigma^*$)* t.q.*:*

- $d(M, U) \leq r$ *e* $d(M, V^R) \leq r$
- $d_{min} \leq d \leq d_{max}$*, onde* $d = |Z|$ *é a "distância" entre o último caractere de U e o primeiro caractere de V.*

Modelos de palíndromos aproximados de comprimento ℓ podem ser inferidos na árvore de afixos CAT(X), de maneira semelhante ao Algoritmo 6.2 percorrendo-se, simultaneamente, caminhos na sub-árvore de sufixos e na sub-árvore de prefixos reversos da RAIZ até vértices de profundidade ℓ em ambas as sub-árvores. Os vértices assim encontrados, combinados dois a dois em pares ordenados, sendo o primeiro deles da sub-árvore de sufixos e o segundo da sub-árvore de prefixos reversos, constituem-se em potenciais "nó-ocorrências" do modelo, restando apenas verificar se as ocorrências que eles representam respeitam o intervalo de separação $[d_{min}, d_{max}]$ (possivelmente, através de alguma anotação na árvore de afixos). A princípio, a verificação dessa restrição da distância para todos os possíveis pares de ocorrências para cada modelo implicaria em um alto custo, portanto, seria também interessante desenvolver algum filtro capaz de eliminar, precocemente, algumas possibilidades.

*Para fins dessa breve discussão informal, relaxamos a definição de [SV97].



(a)

T_{folha}					
folha	$\overline{\texttt{ABABC}}$	$\overline{\texttt{BABC}}$	$\overline{\texttt{ABC}}$	$\overline{\texttt{BC}}$	$\overline{\texttt{C}}$
no. da folha j	1	2	3	4	5
$T_{\text{folha}}[j]$	**3**	**4**	NIL	NIL	NIL

T_{interno}			
vért. interno	RAIZ	$\overline{\texttt{AB}}$	$\overline{\texttt{B}}$
no. do vért.	1	2	3
primeiro_filho	2	**1**	**2**
irmão	NIL	3	**5**
profundidade	0	2	1
posição_head	1	3	4
suffix_link		3	1

(b)

Figura 6.6. Duas implementações para a árvore de sufixos compacta CST(ABABC) da Figura 3.3(c): (a) Implementação proposta por McCreight [McC76]. A árvore é representada na sua forma binária [Knu98b, pp. 334–335,338]. Uma seta vertical partindo de um vértice representa um apontador para o primeiro filho desse vértice, enquanto que uma seta horizontal representa um apontador para um vértice irmão. O rótulo de um vértice representa o par de índices correspondente ao rótulo da aresta aferente ao mesmo. Finalmente, as linhas pontilhadas correspondem aos *suffix links*. (b) Implementação proposta por Kurtz [Kur99]. Cada folha é representada por uma entrada em uma lista T_{folha} de inteiros que contém o índice da folha irmã (se existir) nessa mesma lista. Cada vértice interno é representado por um registro de cinco campos em uma lista de registros. O campo "posição_head" corresponde à posição inicial da primeira ocorrência do fator representado pelo vértice em um novo contexto à direita.

CAPÍTULO 7

CONCLUSÃO E TRABALHOS POSTERIORES

Computadores são inúteis.
Eles podem fornecer apenas respostas.
— PABLO PICASSO

7.1 COMENTÁRIOS FINAIS

Nesta dissertação, apresentamos algumas das principais estruturas de índice presentes na literatura especializada bem como nas aplicações práticas relacionadas a problemas de processamento de texto. Nosso objetivo foi o de estudar, de maneira razoavelmente aprofundada e rigorosa, sobretudo a estrutura desses índices.

Dentre as diversas possíveis aplicações de estruturas de índices em problemas de processamento de texto, aquela sobre a qual concentramos nossa atenção diz respeito à inferência de *motifs* biológicos. Em particular, apresentamos soluções para o problema da inferência de *motifs* simples e estruturados fazendo uso da noção de um *modelo* para esses padrões. Nas soluções apresentadas, esses modelos são inferidos sobre a árvore de sufixos generalizada que representa o conjunto de cadeias não-alinhadas nas os *motifs* devem aparecer de maneira repetida e conservada.

Apresentamos uma breve análise crítica (teórica) acerca da adequação e desempenho das estruturas de índices estudadas para a inferência de *motifs*, apresentando as suas principais virtudes e deficiências. As árvores de sufixos mostraram, de fato, reunir o maior conjunto de qualidades e tiveram, portanto, a sua escolha devidamente justificada. Os DAWGs e CDAWGs também apresentam-se como alternativas viáveis, ao contrário dos vetores de sufixos.

7.2 TRABALHOS POSTERIORES

O presente trabalho admite, naturalmente, uma vasta gama de melhoramentos e extensões, tanto em "profundidade" quanto em "amplitude". Algumas dessas extensões mantém o viés de "estudo aprofundado e análise" que caracteriza este trabalho. Outras, entretanto, possuem uma conotação um tanto mais criativa.

Com respeito ao estudo das estruturas de índice *per se*, destacamos:

- Um estudo ainda mais aprofundado sobre estruturas de índice aqui apresentadas: não obstante o razoável grau de detalhamento deste trabalho, alguns aspectos ainda mereceriam um exame mais detido, notadamente, no que tange aos processos de construção dessas estruturas e ao espaço requerido por elas no caso médio;

- O desenvolvimento de representações mais eficientes para os índices conhecidos (no espírito de [Kur99]) uma vez que, conforme discutido anteriormente, o enorme volume de dados em aplicações de Biologia Computacional, por exemplo, impõe

requisitos ainda mais restritivos sobre o desempenho dessas estruturas, tanto do ponto de vista do espaço quanto de tempo.

- Um estudo mais abrangente das estruturas de índices existentes. Diversas estruturas não foram expostas em detalhes no presente trabalho por questão de escopo, dentre as quais salientamos os *cactos de sufixos* [Kär95], os *oráculos de fatores* [ACR99], as *compressed level tries* [AN93], as *position end-set trees* [LI93], *etc.*

- Um estudo de outras aplicações das estruturas de índice (além do problema da inferência de *motifs* aqui considerado), *e.g.* o casamento aproximado de padrões.

Acerca do problema da inferência de *motifs*, Sagot [Sag02] levanta as seguintes questões:

- Não existem, na prática, algoritmos combinatórios exatos eficientes para inferência de *motifs* longos e sutis (aqueles para os quais existe uma grande quantidade de similaridades espúrias). Os algoritmos existentes não são suficientemente rápidos nem precisos [PS00]. As teorias, técnicas e heurísticas relativas ao problema desenvolvidas até o presente são todas potencialmente passíveis de serem melhoradas, tanto para *motifs* simples quanto estruturados.

- Uma heurística clássica para resolução de problemas de busca com satisfação de restrição (*constraint satisfaction problems*) é conhecida como *most restricted first* e consiste em iniciar a busca pelo trecho mais restrito, o que potencializa a chance de detecção precoce de insucesso. No caso da inferência de *motifs* estruturados, não existe nenhum algoritmo eficiente conhecido que permita iniciar a busca pelo bloco mais restrito, *i.e.*, para o qual se permitem menos erros. Essa questão está diretamente relacionada aos índices que têm sido empregados para solução do problema. A extensão de estruturas conhecidas para adequá-las a essa variação do problema e até a proposição novas estruturas para este fim, constituem-se em tópicos de pesquisa de interesse.

- Ainda para o problema da inferência e localização de *motifs*, pode ser interessante, por razões meramente algorítmicas e, sobretudo por razões biológicas, considerar *motifs* (simples ou estruturados) de feição palindrômica. Aqui novamente, a questão da adequação das estruturas de índice reside no cerne do problema. Acreditamos que a natureza simétrica de estruturas como os SCDAWGs e árvores de afixos aponta para um horizonte bastante promissor que pode ser explorado com boa chance de êxito.

Ainda na linha dos algoritmos combinatórios exatos, também podem ser exploradas outras abordagens para a inferência de *motifs* que não através dos modelos.

REFERÊNCIAS BIBLIOGRÁFICAS

[ACR99] Cyril Allauzen, Maxime Crochemore, and Mathieu Raffinot. Factor oracle: a new structure for pattern matching. In *Proceedings of the 26th Conference on Current Trends in Theory and Practice of Informatics (SOFSEM'99)*, volume 1725 of *Lecture Notes in Computer Science*, pages 291–306, Milovy, Czech Republic, 1999. Springer-Verlag.

[Aho90] Alfred V. Aho. Algorithms for finding patterns in strings. In Jan van Leeuwen, editor, *Algorithms and Complexity*, volume A of *Handbook of Theoretical Computer Science*, chapter 5, pages 255–300. Elsevier, Amsterdam, 1990.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachussets, 1974.

[AN93] Arne Andersson and Stefan Nilsson. Improved behaviour of tries by adaptive branching. *Information Processing Letters*, 46(6):295–300, 1993.

[Apo85] Alberto Apostolico. The myriad virtues of subword trees. In Zvi Galil and Alberto Apostolico, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO ASI Series F*, pages 85–96. Springer-Verlag, New York, 1985.

[BBH+85] Anselm Blumer, J. Blumer, David Haussler, Andrzej Ehrenfeucht, M. T. Chen, and Joel Seiferas. The smallest automaton recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[BBHM87] Anselm Blumer, J. Blumer, David Haussler, and Ross McConnell. Complete inverted files for efficient text retrieval and analysis. *Journal of The Association for Computing Machinery*, 34(3):578–595, July 1987.

[BEH89] Anselm Blumer, Andrzej Ehrenfeucht, and David Haussler. Average size of suffix trees and dawgs. *Discrete Applied Mathematics*, 24:37–45, 1989.

[BM77] Robert S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, October 1977.

[BRCR94] Paul Bieganski, John Riedl, John Carlis, and Ernest F. Retzel. Generalized suffix trees for biological sequence data: Applications and implementations. In *Proc. of the 27th Hawai International Conference on Systems Sciences*, pages 35–44. IEEE Computer Society Press, 1994.

[BY96] Ricardo Baeza-Yates. A unified view of pattern matching problems. In *Proc. 22nd Latin American Conference on Informatics*, Bogota, Colombia, June 1996.

[BYG92] Ricardo A. Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Communications of the ACM*, 35(10):74–82, October 1992.

[BYGR90] Ricardo Baeza-Yates, Gaston Gonnet, and Mireille Régnier. Analysis of Boyer-Moore-type string searching algorithms. In *First ACM-SIAM Symposium on Discrete Algorithms*, pages 328–343, San Francisco, January 1990.

[CH98] Maxime Crochemore and Christophe Hancart. Pattern matching in strings. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook*, chapter 11, pages 1–2. CRC Press, Boca Raton, 1998.

[CL92] William I. Chang and Jordan Lampe. Theoretical and empirical comparisons of approximate string matching algorithms. In *Proc. of the 3rd Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, volume 644 of *Lecture Notes in Computer Science*, pages 172–181, Tucson, Arizona, 1992. Springer Verlag.

[CL97a] Christian Charras and Thierry Lecroq. Handbook of exact string-matching algorithms. `http://www-igm.univ-mlv.fr/~lecroq`, 1997.

[CL97b] Maxime Crochemore and Thierry Lecroq. Pattern matching and text compression algorithms. In Allen B. Tucker, editor, *The Computer Science and Engineering Handbook*, chapter 8, pages 162–202. CRC Press, Boca Raton, 1997.

[CLR89] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachussets, 1989.

[CM94] William I. Chang and Thomas G. Marr. Approximate string matching and local similarity. In *Proc. of the 5th Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, volume 807 of *Lecture Notes in Computer Science*, pages 259–273, Tucson, Arizona, 1994. Springer-Verlag.

[Col94] Richard Cole. Tight bounds on the complexity of the Boyer-Moore string matching algorithm. *SIAM Journal on Computing*, 23(5):1075–1091, October 1994.

[CR94] Maxime Crochemore and Wojciech Rytter. *Text Algorithms*. Oxford University Press, New York, 1994.

[Cro86] Maxime Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45(1):63–86, 1986.

[CV97a] Maxime Crochemore and Renaud Vérin. Direct construction of compact directed acyclic word graphs. In Alberto Apostolico and Jotun Hein, editors, *Proc. 8th Annual Symposium on Combinatorial Pattern Matching (CPM'97)*, volume 1264 of *Lecture Notes in Computer Science*, pages 116–129, Aarhus, Denmark, 1997. Springer-Verlag.

[CV97b] Maxime Crochemore and Renaud Vérin. On compact directed acyclic word graphs. In J. Mycielski, G. Rozenberg, and A. Salomaa, editors, *Structures in Logic and Computer Science, a selection of essays in honor of A. Ehrenfeucht*, volume 1261, pages 192–211. Springer-Verlag,, Berlin, 1997.

[FBY92] William B. Frakes and Ricardo A. Baeza-Yates, editors. *Information Retrieval: Data Structures & Algorithms*. Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[GK97] Robert Giegerich and Stefan Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19:331–353, 1997.

[GS83] Zvi Galil and Joel Seiferas. Time-space optimal string matching. *Journal of Computer and Systems Science*, pages 280–294, 1983.

[IHS+01a] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, and Setsuo Arikawa. On-line construction of symmetric compact directed acyclic word graphs. In *Proc. 8th International Symposium on String Processing and Information Retrieval (SPIRE'01)*, pages 96–110, Laguna de San Rafael, Chile, 2001. IEEE Computer Society.

[IHS+01b] Shunsuke Inenaga, Hiromasa Hoshino, Ayumi Shinohara, Masayuki Takeda, Setsuo Arikawa, Giancarlo Mauri, and Giulio Pavesi. On-line construction of compact directed acyclic word graphs. In *Proc. 12th Annual Symposium on Combinatorial Pattern Matching (CPM'01)*, volume 2089 of *Lecture Notes in Computer Science*, pages 169–180, Jerusalem, Israel, 2001. Springer-Verlag.

[IL00] Robert W. Irving and Lorna Love. The suffix binary search tree and suffix avl tree. Technical Report TR-2000-54, Computing Science Department, Glasgow University, UK, February 2000.

[Ine02] Shunsuke Inenaga. On data structures for string searching problems. Master's thesis, Department of Informatics, Kyushu University, Japan, February 2002.

[Kär95] Juha Kärkkäinen. Suffix cactus: A cross between suffix tree and suffix array. In *Proc. 6th Symposium on Combinatorial Pattern Matching (CPM '95)*, volume 937 of *Lecture Notes in Computer Science*, pages 191–204, Espoo/Helsinki, Finland, 1995. Springer-Verlag.

[KMP77] Donald E. Knuth, Jim H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, June 1977.

[Knu98a] Donald E. Knuth. Fundamental algorithms. In *The Art of Computer Programming*, volume 1, pages 334–346. Addison-Wesley, Reading, Massachusetts, third edition, 1998.

[Knu98b] Donald E. Knuth. Sorting and searching. In *The Art of Computer Programming*, volume 3, pages 492–507. Addison-Wesley, Reading, Massachusetts, second edition, 1998.

[KR87] Richard M. Karp and Michael O. Rabin. Efficient randomized string matching algorithms. *IBM J. Res. Dev.*, 31:249–260, 1987.

[Kur99] Stefan Kurtz. Reducing the space requirement of suffix trees. *Software—Practice and Experience*, 29(13):1149–1171, 1999.

[LI93] Christophe Lefèvre and Joh-E. Ikeda. The position end-set tree: a small automaton for word recognition in biological sequences. *Computer Applications in the Biosciences*, 9(3):343–348, June 1993.

[LSS+79] Cláudio Lucchesi, Imre Simon, Istvan Simon, Janos Simon, and Tomasz Kowaltowski. *Aspectos Teóricos da Computação*. Projeto Euclides. Instituto de Matemática Pura e Aplicada, Rio de Janeiro, 1979.

[Maa00] Moritz G. Maaß. Linear bidirectional on-line construction of affix trees. In *Proc. of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00)*, volume 1848 of *Lecture Notes in Computer Science*, pages 320–334, Montreal, Canada, 2000. Springer-Verlag.

[Mäk00] Veli Mäkinen. Compact suffix array. In *Proc. of the 11th Annual Symposium on Combinatorial Pattern Matching (CPM'00)*, volume 1848 of *Lecture Notes In Computer Science*, pages 305–319, Montreal, Canada, 2000. Springer-Verlag.

[Mäk01] Veli Mäkinen. Trade off between compression and search times in compact suffix array. In *Proc. 3rd Workshop on Algorithm Engineering and Experimentation (ALENEX'01)*, volume 2153 of *Lecture Notes In Computer Science*, pages 189–201, Washington D.C., January 2001. Springer-Verlag.

[Mar02] Laurent Marsan. *Inférence de Motifs Structurés: Algorithmes et Outils Appliqués a la Détection de Sites de Fixation Dans les Séquences Génomiques*. PhD thesis, Université de Marne-la-Vallée, France, April 2002.

[McC76] Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[MM93] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22:935–948, 1993.

[Mor68] Donald R. Morrison. PATRICIA—Practical Algorithm To Retrieve Information Coded In Alphanumeric. *Journal of the ACM*, 15(4):514–534, October 1968.

[MS00] Laurent Marsan and Marie-France Sagot. Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification. *Journal of Computational Biology*, 7:345–360, 2000.

[Nav98] Gonzalo Navarro. *Approximate Text Searching*. PhD thesis, Departamento de Ciencias de Computación, Universidad de Chile, 1998.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, March 2001.

[NBYST01] Gonzalo Navarro, Ricardo Baeza-Yates, Erkki Sutinen, and Jorma Tarhio. Indexing methods for approximate string matching. *IEEE Data Engineering Bulletin*, 24(4):19–27, 2001.

[PS00] Pavel Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *8th International Conference fon Intelligent Systems for Molecular Biology ISMB*, pages 269–278, 2000.

[Raf99] Mathieu Raffinot. *Structures pour la Localisation de Motifs*. PhD thesis, Université de Marne-la-Vallée, France, October 1999.

[Riv76] Ronald L. Rivest. Partial-match retrieval algorithms. *SIAM Journal on Computing*, 5(1):19–50, 1976.

[Sag98] Marie-France Sagot. Spelling approximated repeated or common motifs using a suffix tree. In Cláudio Lucchesi and Arnaldo Moura, editors, *LATIN'98: Theoretical Computer Science*, volume 1380 of *Lecture Notes in Computer Science*, pages 111–127. Springer-Verlag, 1998.

[Sag02] Marie-France Sagot. Comunicação pessoal, April 2002.

[Sel80] Peter H. Sellers. The theory and computation of evolutionary distances: Pattern recognition. *Journal of Algorithms*, 1:359–373, 1980.

[SM97] João Setubal and João Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Company, Boston, 1997.

[Sto95] Jens Stoye. Affixbäume. Master's thesis, Technische Fakultät der Universität Bielefeld, Germany, May 1995. Traduzido em [Sto00].

[Sto00] Jens Stoye. Affix trees. Technical Report 2000-04, Technische Fakultät der Universität Bielefeld, Germany, 2000. Tradução de [Sto95].

[SV97] Marie-France Sagot and Alain Viari. Flexible identification of structural objects in nucleic acid sequences: palindromes, mirror repeats, pseudoknots and triple helices. In Alberto Apostolico and Jotun Hein, editors, *Proc. 8th Annual Symposium on Combinatorial Pattern Matching (CPM'97)*, volume 1264 of *Lecture Notes in Computer Science*, pages 224–246, Aahus, Denmark, 1997. Springer-Verlag, Berlin.

[SW02] Marie-France Sagot and Yoshiko Wakabayashi. Pattern inference under many guises. In Bruce Reed and Cláudia Sales, editors, *Recent Advances in Algorithms and Combinatorics*. Springer-Verlag, 2002.

[Ukk85] Esko Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[Ukk95] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[Wei73] Peter Weiner. Linear pattern-matching algorithms. In *Proc. of the 14th IEEE Annual Symposium on Switching and Automata Theory*, pages 1–11, New York, 1973.

[WM92a] Sun Wu and Udi Manber. Agrep—a fast approximate pattern-matching toll. In *Usenix Winter 1992 Technical Conference*, pages 153–162, San Francisco, January 1992.

[WM92b] Sun Wu and Udi Manber. Fast text searching allowing errors. *Communications of the ACM*, 35(10):83–91, October 1992.