Árvores de Ukkonen:
caracterização combinatória e aplicações

Gustavo Akio Tominaga Sacomoto

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Ciência da Computação

Orientador: Prof. Dr. Alair Pereira do Lago

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, fevereiro de 2011

Árvores de Ukkonen:
caracterização combinatória e aplicações

Esta dissertação contém as correções e alterações
sugeridas pela Comissão Julgadora durante a defesa
realizada por (Gustavo Akio Tominaga Sacomoto) em 08/02/2011.
O original encontra-se disponível no Instituto de
Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Alair Peirera do Lago (orientador) - IME-USP
- Prof. Dr. José Coelho de Pina Junior - IME-USP
- Prof. Dr. Guilherme Pimentel Teles - IC-UNICAMP

Agradecimentos

Primeiramente, gostaria de agradecer meus pais: Miltes e João. Este trabalho é tão deles quanto meu. Eles, que sempre me deram toda a liberdade para fazer as minhas próprias escolhas e todo o suporte e incentivo para seguir o caminho escolhido. Agradeço também a minha irmã Natália, por sua companhia e amizade.

Gostaria de agradecer a todos os meus amigos: do colegial, da Unicamp e das Ciências Moleculares. Que estiveram comigo nos momentos de descontração (essenciais para o desenvolvimento deste trabalho!), nas viagens, nos bares e baladas de Campinas e São Paulo.

Agradeço a todos os professores do Curso de Ciências Moleculares por me ensinarem o rigor do pensamento científico e, ao mesmo tempo, abrirem meus olhos para toda a beleza das teorias científicas. Por fim, agradeço a todos os professores que conheci no IME, mesmo não sendo um aluno muito assíduo, aprendi muito com eles. Em especial, agradeço meu orientador Alair, por sua enorme paciência e constante interesse em meu trabalho.

Resumo

A árvore de sufixos é uma estrutura dados, que representa em espaço linear todos os fatores de uma palavra, com diversos exemplos de aplicações práticas. Neste trabalho, definimos uma estrutura mais geral: a *árvore de Ukkonen*. Provamos para ela diversas propriedades combinatórias, dentre quais, a minimalidade em um sentido preciso. Acreditamos que a apresentação aqui oferecida, além de mais geral que as árvores de sufixo, tem a vantagem de oferecer uma descrição explícita da topologia da árvore, de seus vértices, arestas e rótulos, o que não vimos em nenhum outro trabalho.

Como aplicações, apresentamos também a árvore esparsa de sufixos (que armazena apenas um subconjunto dos sufixos) e a árvore de k-fatores (que armazena apenas os segmentos de comprimento k, ao invés dos sufixos) definidas como casos particulares das árvores de Ukkonen.

Propomos para as árvores esparsas um novo algoritmo de construção com tempo $O(n)$ e espaço $O(m)$, onde n é tamanho da palavra e m é número de sufixos. Para as árvores de k-fatores, propomos um novo algoritmo *online* com tempo e espaço $O(n)$, onde n é o tamanho da palavra.

Palavras-chave: estrutura de dados, busca por padrões, recuperação de informação, combinatória de palavras, biologia computacional, stringology.

Abstract

The suffix tree is a data structure that represents, in linear space, all factors of a given word, with several examples of practical applications. In this work, we define a more general structure: the Ukkonen's tree. We prove many properties for it, among them, its minimality in a precise sense. We believe that this presentation, besides being more general than the suffix trees, has the advantage of offering an explicit description of the tree topology, its vertices, edges and labels, which was not seen in any other work.

As applications, we also presents the sparse suffix tree (which stores only a subset of the suffixes) and the k-factor tree (which stores only the substrings of length k, instead of the suffixes), both defined as Ukkonen's tree special cases.

We propose a new construction algorithm for the sparse suffix trees with time $O(n)$ and space $O(m)$, where n is the size of the word and m is the number of suffixes. For the k-factor trees, we propose a new *online* algorithm with time and space $O(n)$, where n is the size of the word.

Keywords: data structure, pattern matching, information retrieval, combinatorics on words, computational biology, stringology.

Sumário

Lista de Figuras

Lista de Algoritmos

Capítulo 1

Introdução

Em 1985, Galil [AG85] cunhou o termo *stringology* para o conjunto de métodos algorítmicos aplicados ao processamento de textos e palavras (*strings*). Muitos dos quais se baseiam em resultados combinatórios sobre palavras, por exemplo, a demonstração de Cole [Col91] para linearidade de uma variante do algoritmo de Boyer-Moore para busca de padrões. Outros se baseiam na teoria de autômatos, como o algoritmo de Knuth-Morris-Pratt [CLRS01] também para a busca de padrões. Ou seja, trata-se de uma "área" com enfoque algorítmico, mas com uma intersecção grande com outras áreas mais teóricas, como combinatória de palavras, teoria de autômatos e teoria de grupos.

O texto, uma sequência de caracteres, sempre foi uma forma fundamental de representação de informação. Neste exato momento milhões de programas estão efetuando algum tipo de processamento de texto. Muitos outros estão produzindo alguma forma de texto, *web crawlers* estão varrendo toda a internet coletando informações sobre as páginas, sequenciadores de DNA de segunda geração [SJ08] estão sequenciando genomas inteiros. O ritmo de crescimento das bases de dados aumentou enormemente no últimos anos, elas vem se tornando gigantescas e como consequência, algoritmos muito eficientes são necessários para lidar com elas. Em diversas aplicações, esta eficiência exige uma forma de indexação, por exemplo do texto, de forma que, depois de um pré-processamento onde sua indexação é feita, não mais o algoritmo depende de seu tamanho. Uma técnica que pode ser utilizada para as bases textuais é a indexação com árvores de sufixos.

As árvores de sufixos foram apresentadas pela primeira vez por Weiner [Wei73] em 1973, em um trabalho que mais tarde foi chamado por Donald Knuth de "Algoritmo do ano de 1973" [Pra73]. O trabalho apresenta uma estrutura de dados, a árvore de sufixos, que representa todos os fatores (*substrings*) de uma determinada palavra e um algoritmo linear para a sua construção. Trata-se de um índice que armazena em espaço linear informações sobre todos os fatores de uma determinada palavra. Com este índice é possível resolver de maneira eficiente

diversos problemas de sobre palavras, por exemplo: a busca de padrões, o problema do maior fator comum entre duas palavras e do fator mais frequente em uma palavra [Gus97].

Trabalhos posteriores de McCreight [McC76] e Ukkonen [Ukk95] apresentaram outros algoritmos lineares para a construção da árvore de sufixos, o primeiro realiza um número menor de operações e é mais eficiente no uso de memória e o segundo é o único *online*. Outro trabalho de Giegerich et al. [GK97] mostra a relação entre esses três algoritmos, aparentemente, bastante distintos. Estes algoritmos partem da hipótese de que o tamanho do alfabeto é constante. Por fim, Farach [Far97] apresentou um algoritmo de construção, completamente diferente dos anteriores e bastante complexo, mas linear mesmo no caso em que o alfabeto é grande, i.e. é da mesma ordem do tamanho da palavra.

O primeiro objetivo deste trabalho é apresentar uma estrutura, inspirada em, mas mais geral que as árvores de sufixo: as árvores de Ukkonen [dLS03]. No primeiro momento, o trabalho tem um enfoque mais matemático, a fim de bem defini-la e caracterizar completamente sua topologia baseando-se unicamente no conjunto de palavras associado e provar a sua minimalidade.

O segundo objetivo deste trabalho é o de aplicar as árvores de Ukkonen a casos particulares; que incluem inclusive as *árvores de sufixos* e as muitas vezes chamadas de *árvores generalizadas de sufixos*. Em seguida, adotamos um enfoque mais algorítmico, definimos as *árvores esparsas de sufixos* [AJS99] e *árvores de k-fatores* [AS04] como casos particulares das árvores de Ukkonen e propomos novos algoritmos ótimos para cada um destes dois casos particulares.

A forma como o trabalho está organizado é apresentada mais detalhadamente a seguir.

No capítulo 3, assentamos as bases sobre as quais todo a resto do trabalho está apoiado, lá definimos a árvore de Ukkonen que reconhece um dado conjunto de palavras. Tal definição se baseia em uma outra estrutura que apresentamos na seção 3.1, a A^+-árvore. Ao longo do capítulo enunciamos e provamos diversas proposições que relacionam a topologia da árvore de Ukkonen com as propriedade combinatórias do conjunto de palavras que ela reconhece. Na seção 3.4, provamos uma lema que se traduz facilmente em um algoritmo ingênuo para a construção das árvores de Ukkonen. No final do capítulo, provamos que as árvore de Ukkonen são minimais, ou mais precisamente, que elas são um *minor* de qualquer A^+-árvore para o mesmo conjunto de palavras.

No capítulo 4, apresentamos as árvores de sufixo e as árvores de sufixo generalizadas como casos particulares das árvores de Ukkonen. Ao longo do capítulo discutimos algumas consequências desta nova definição para estas árvores. Na seção 4.3, discutimos alguns aspectos ligados com a representação computacional das árvores de sufixos. Na seção 4.4, apresentamos de maneira bastante sucinta os algoritmos clássicos para construção linear da árvore de sufixos: Ukkonen, McCreight, Weiner e Farach.

No capítulo 5, apresentamos a árvore esparsa de sufixos. Elas são definidas como casos

particulares das árvores de Ukkonen que reconhecem apenas um subconjunto de m sufixos de uma palavra w de comprimento n. Apresentamos também um novo algoritmo que constrói a árvore esparsa de sufixos em tempo $O(n)$ e espaço linear no número de sufixos considerados, $O(m)$. Observe que é fácil construir esta árvore em tempo linear usando espaço $O(n)$, basta construir a árvore de sufixos (completa) e posteriormente eliminar os sufixos ausentes, mas esta abordagem usa espaço $O(n)$. Na seção 5.4, comparamos o nosso algoritmo com as soluções encontradas na literatura [AJS99, KU96, IT06]. Por fim, na seção 5.5 discutimos um problema em aberto relacionado com a construção das árvores esparsas.

No capítulo 6, apresentamos as árvores de k-fatores. As quais também são definidas como casos particulares das árvore de Ukkonen em que o conjunto de palavras reconhecidas é o conjunto de todos os fatores de comprimento k de uma palavra w de comprimento n. Em seguida, propomos um novo algoritmo *online* com tempo e espaço $O(n)$ para a construção das árvores de k-fatores. Assim como para as árvores esparsas, existe um algoritmo trivial para a construção das árvores de k-fatores, basta construir a árvore completa e remover todos os vértices com altura maior do que k. Este algoritmo também usa espaço e tempo $O(n)$, mas não é online, é menos eficiente que a construção direta e necessariamente usa mais memória, mesmo tendo a mesma complexidade assintótica.

Por fim, no capítulo 7, apresentamos as nossas conclusões sobre este trabalho e, no capítulo 8, discutimos algumas linhas possíveis para a sua continuidade.

Capítulo 2

Definições Gerais

Neste capítulo apresentamos algumas definições que serão utilizadas ao longo do texto. As definições deste capítulo não são exaustivas, outras serão apresentadas ao longo dos outros capítulos, não as colocamos aqui por se tratarem de conceitos com aplicações mais restritas ou que ficariam mais claros se apresentados nos contextos mais adequados.

2.1 Combinatória de Palavras

Um *alfabeto* é um conjunto de símbolos distintos, seus elementos são chamados de *letras*. Seja A um alfabeto finito[1], qualquer sequência finita de letras de A é chamada de *palavra sobre* A. O conjunto de todas as palavras sobre A de comprimento k é denotado por A^k, o conjunto de todas as palavras sobre A (incluindo a palavra vazia 1) é denotado por $A^* = \cup_{k>0} A^k$ e o conjunto $A^* \setminus \{1\}$, todas as palavras não vazias, é denotado por A^+.

O *comprimento* de uma palavra w é o tamanho da sequência de letras associadas a w e é denotado por $|w|$. Também usaremos esta notação para a cardinalidade de conjuntos, i.e. se Σ é um conjunto, então $|\Sigma|$ é a sua cardinalidade. A palavra vazia tem comprimento 0 e é a única com esta propriedade. A palavra w pode ser representada de maneira explicita como uma sequência de letras em A, desta forma $w = w[1]w[2]\dots w[|w|]$, onde $w[i] \in A$ com $1 \leq i \leq |w|$. Para tornar a notação mais concisa, se $1 \leq i \leq j \leq |w|$ a sub-cadeia $w[i]w[i+1]\dots w[j-1]w[j]$ é representada por $w[i,j]$ e possui comprimento $|w[i,j]| = j - i + 1$.

A *concatenação* de duas palavras u e v é definida como a concatenação de suas respectivas sequências, ou seja, é a sequência $u[1,|u|]v[1,|v|]$, e é denotada por uv. Não é difícil ver que está é uma operação associativa, não comutativa e 1 é o seu elemento neutro . Além disso, o comprimento da palavra resultante uv é $|uv| = |u| + |v|$.

[1] A não ser que seja mencionado explicitamente o contrário, todos os alfabetos considerados neste trabalho são finitos.

Seja $w = utv$ uma palavra sobre A, com $u, v, t \in A^*$, então u é *prefixo* de w, v é *sufixo* de w e u, t e v são *fatores* de w. O conjunto de todos os prefixos de w é denotado por $\mathrm{Pref}(w)$, de maneira similar, o conjunto de sufixos é denotado por $\mathrm{Suf}(w)$ e o de fatores é denotado por $\mathrm{Fat}(w)$. Dizemos que x é prefixo (sufixo, fator) *próprio* de w se $\mathrm{Pref}(x) \subset \mathrm{Pref}(w)$ ($\mathrm{Suf}(x) \subset \mathrm{Suf}(w)$, $\mathrm{Fat}(x) \subset \mathrm{Fat}(w)$)

Seja $u \in \mathrm{Pref}(w)$, nós denotamos[2] por $u^{-1}w$ a palavra obtida com remoção do prefixo u de w, ou seja, o sufixo de w com tamanho $|w| - |u|$. De maneira análoga, se $u \in \mathrm{Suf}(w)$ definimos wu^{-1}. Ressaltamos que o símbolo u^{-1} não tem sentido sozinho, somente quando aplicado a uma palavra w que tenha u como prefixo (sufixo).

Se $w \in A^*$ e $|w| \geq k$, então $A^{-k}w$ é a palavra obtida após a remoção do prefixo (só existe um) de comprimento k de w. De maneira análoga, definimos wA^{-k}.

2.2 Grafos e Árvores

Um *grafo dirigido* ou *grafo* G, é um par (V, E), onde V é um conjunto finito qualquer e seus elementos são chamados *vértices*. O conjunto $E \subseteq V \times V$ é um conjunto de pares ordenados, cujos elementos são denominados *arestas*. Dizemos que uma aresta (u, v), *sai* de u e *chega* em v.

Um *caminho* de comprimento k de s para t no grafo $\mathbf{G} = (V, E)$ é uma sequência de vértices $(v_0, v_1, \ldots, v_k)$, dois a dois distintos, tais que $v_0 = s$, $v_k = t$ e $(v_{i-1}, v_i) \in E$ para $i \in [1, k]$. Claro, um caminho também pode ser visto como uma sequência de arestas $(e_1, e_2, \ldots, e_k)$, tais que $e_i = (v_{i-1}, v_i)$ para $i \in [1, k]$. O caminho *contém (ou passa pelos)* os vértices $v_0, v_1, \ldots, v_k$ e as arestas (v_{i-1}, v_i) para $i \in [1, k]$.

Uma *árvore enraizada* ou *árvore* é um grafo dirigido $\mathbf{T} = (V, E)$ com um vértice $r \in V$ chamado *raiz*, tal que todo para vértice $v \in V$, existe um único caminho de r para v. A *profundidade* de $v \in V$ é o comprimento deste único caminho e a *altura* de $\mathbf{T}$ é a profundidade de seu vértice mais profundo. Dizemos que u é *ancestral* de v em $\mathbf{T}$ se u pertence ao caminho da raiz até v. Este ancestral é *próprio* se $u \neq v$. Definimos ainda o *menor ancestral comum (MAC)* entre $u, v \in V$ como o ancestral comum (a raiz é ancestral de todos os vértices) entre u e v que possuí a maior profundidade.

Seja $\mathbf{T} = (V, E)$ uma árvore, dizemos que u é *pai* de v e que v é *filho* de u se $(u, v) \in E$. Segue da nossa definição de árvore que todos os vértices $v \in V$, exceto a raiz, possuem um único pai, o denotamos por $\mathrm{Pai}(v)$. Todos os vértices de $\mathbf{T}$ que não possuem nenhum filho são chamados de *folhas*. Todos os outros vértices são chamados de *vértices internos*.

[2]Este é uma notação conveniente inspirada na teoria de grupos, mas o conjunto A^* com a operação de concatenação não forma um grupo, justamente porque a concatenação de palavras não é uma operação inversível.

Como lidamos exclusivamente com árvores ao longo de todo o trabalho, omitiremos o sentido das arestas[3] em favor da seguinte convenção: seja $\mathbf{T} = (V, E)$ uma árvore e $(u, v) \in E$ uma aresta, então na representação gráfica da árvore $\mathbf{T}$, u estará a cima de v, indicando que a aresta vai de u para v. Como consequência disto, a raiz de qualquer árvore será sempre o vértice acima dos demais.

Na figura 2.1, temos um exemplo de árvore representada em que os vértices são: a, b, c, d, e e f. De acordo com a nossa convenção, a raiz desta árvore é o vértice a e as aresta são: $(a, b), (a, c), (b, d), (b, e)$ e (c, f). Seguindo as nossa definições, b e c são vértice internos e d, e e f são as folhas da árvore.

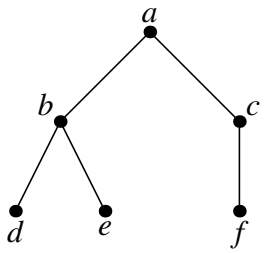


Figura 2.1: *Exemplo de uma árvore*

[3] Com uma única exceção para um tipo especial de aresta, as ligações de sufixo, usadas nos capítulos 5 e 6.

Capítulo 3

Caracterização Combinatória

Neste capítulo definimos as estruturas combinatórias que serão usadas ao longo de todo o trabalho: a A^+-árvore e a árvore de Ukkonen. Começamos definindo as A^+-árvores, das quais as árvores de Ukkonen são casos particulares, na seção 3.1 e expondo suas principais propriedades na seção 3.2. Na seção 3.3, definimos as árvores de Ukkonen e demonstramos que esta definição é boa. Na seção 3.4, temos uma limitação no tamanho das árvores de Ukkonen e um lema que relaciona a estrutura de duas árvores cujos conjuntos de palavras diferem por apenas um elemento. Baseado neste lema, na seção 3.5, propomos um algoritmo ingênuo para a construção da árvore de Ukkonen. Finalmente, na seção 3.6, demonstramos que, fixado um conjunto de palavras W, a árvore de Ukkonen de W é a menor A^+-árvore que reconhece W.

3.1 A^+-árvores

Definição 3.1 (A^+-árvore)**.** *Uma* A^+-árvore $\mathbf{T} = (V, E, \lambda)$ *é uma árvore enraizada* (V, E) *com uma rotulação nas arestas* $\lambda : E \longrightarrow A^+$ *e com a propriedade adicional que quaisquer duas arestas distintas que saem de um mesmo vértice tem rótulos cujas primeiras letras são distintas.*

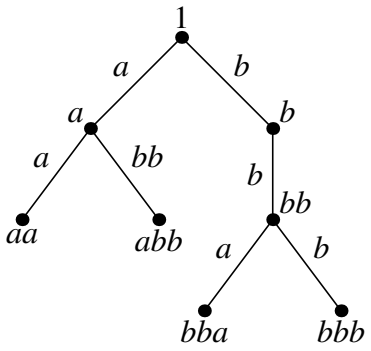


Figura 3.1: *Um exemplo de* A^+*-árvore, com a rotulação das arestas e vértices.*

A partir desta definição, que também aparece em [GK97], temos que cada vértice possui no máximo $|A|$ filhos. Além disso, dado um caminho $p = (e_1, e_2, \ldots, e_k)$, é natural estender a rotulação λ para definir o *rótulo de um caminho* p como a palavra $\lambda(e_1)\lambda(e_2)\cdots\lambda(e_k)$. Como existe somente um caminho p entre raiz até qualquer vértice v da árvore, nós também estendemos a rotulação para definir o *rótulo de um vértice* v como a palavra $\lambda(p)$. Como os rótulos das arestas são não vazios, é imediato verificar que para qualquer vértice v o comprimento de $\lambda(v)$ é um limitante para a profundidade de v. Na figura 3.1, temos um exemplo de A^+-árvore com os rótulos das arestas e vértices representados.

A A^+-árvore é dita *compacta* se todos os vértices internos, com exceção da raiz, possuem pelo menos dois filhos. No exemplo da figura 3.1 temos uma A^+-árvore que não é compacta, o vértice interno com rótulo b possui apenas um filho. Na figura 3.2, temos um exemplo de A^+-árvore compacta.

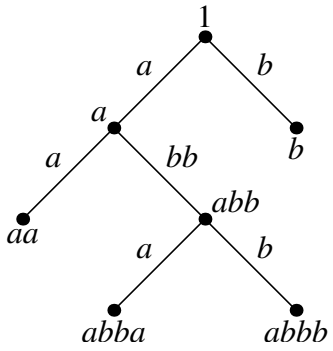


Figura 3.2: *Um exemplo de A^+-árvore compacta.*

Seja uma A^+-árvore $\mathbf{T} = (V, E, \lambda)$. Definimos um *lugar em* $\mathbf{T}$ como sendo um par ordenado (u, x) onde $u \in V$ é um vértice qualquer e $x \in A^*$ é a palavra vazia ou um prefixo próprio não vazio do rótulo de uma aresta que parte de u. Caso x seja vazio, permitimo-nos dizer que o lugar é o vértice u. Caso contrário, dizemos que o lugar está numa aresta, a saber, a única aresta que parte de u e que começa com a primeira letra de x; neste caso, dizemos que o lugar é um vértice implícito. Também estendemos a rotulação de forma a definir o *rótulo do lugar* (u, x) como sendo a palavra $\lambda((u, x)) = \lambda(u)x$.

Dizemos que uma palavra $w \in A^*$ *é uma palavra de (é representada em)* $\mathbf{T}$ se w é prefixo do rótulo de algum vértice de $\mathbf{T}$. Além disso, dizemos que $\mathbf{T}$ *reconhece* w se existe algum vértice cujo rótulo é w. Repare que se w é reconhecido por $\mathbf{T}$ então w é uma palavra de $\mathbf{T}$, mas a recíproca não é verdadeira. Estendemos esta última definição para conjuntos de palavras da maneira apresentada a seguir.

Definição 3.2. *Seja W um conjunto de palavras. Dizemos que a A^+-árvore $\mathbf{T} = (V, E, \lambda)$* reconhece *$W$ se existe um subconjunto de vértices $X \subseteq V$ tal que $\lambda(X) = \{\lambda(x) \mid x \in X\} = W$.*

Chamamos estes vértices de vértices finais.

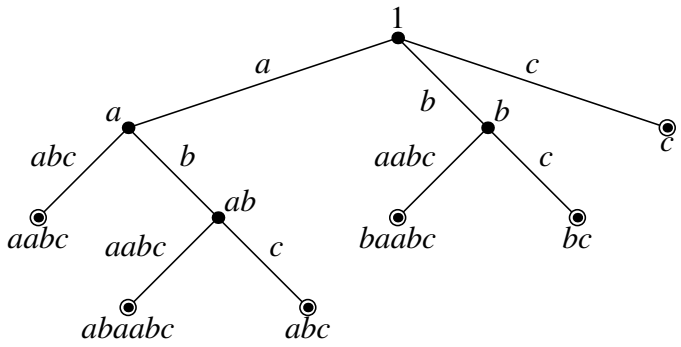


Figura 3.3: *Uma A^+-árvore que reconhece* $\mathrm{Suf}(abaabc) \setminus \{1\}$

Na figura 3.3, as folhas da A^+-árvore são os vértices finais e estão destacados. Repare que o conjunto W de palavras reconhecidas pelo conjunto X das folhas da A^+-árvore na figura 3.3 é o conjunto

$$\{abaabc, baabc, aabc, abc, bc, c\},$$

i.e., $\mathrm{Suf}(abaabc) \setminus \{1\}$, o conjunto de sufixos não vazios da palavra $abaabc$.

3.2 Propriedades Gerais

Esta seção esta organizada da seguinte forma: na proposição 3.3 vemos uma propriedade que relaciona a ancestralidade entre dois vértices com os prefixos de seus rótulos e no corolário 3.4 vemos que não há vértices distintos com mesmos rótulos. A proposição 3.5 relaciona o *menor ancestral comum (MAC)* de dois vértices com o prefixo comum mais longo de seus rótulos. Finalmente, a proposição 3.6 mostra-nos que a rotulação λ estabelece uma bijeção entre os lugares na A^+-árvore $\mathbf{T}$ e as palavras de $\mathbf{T}$. Assim, dada uma palavra w de $\mathbf{T}$, podemos então definir o *lugar de w em* $\mathbf{T}$ como sendo o lugar $\lambda(w) = (u, \lambda(u)^{-1}w)$, onde u é o vértice mais profundo tal que $\lambda(u)$ é prefixo de w.

Estas proposições, especialmente a 3.3 e a 3.5, mostram que a topologia de $\mathbf{T}$ está fortemente ligada ao *conjunto de palavras de* $\mathbf{T}$. Elas serão necessárias na seção 3.3 para que seja possível dar uma boa definição de A^+-árvore minimal que reconhece determinado conjunto de palavras, ou seja, a árvore de Ukkonen.

Proposição 3.3. *Sejam u e v dois vértices de uma A^+-árvore $\mathbf{T} = (V, E, \lambda)$. Então u é um ancestral de v se e só se $\lambda(u)$ for prefixo próprio de $\lambda(v)$.*

Prova. Suponha que o vértice u seja um ancestral de v. Então o passeio p da raiz até v pode

ser decomposto como a concatenação dos passeios q da raiz até u e o passeio não vazio r de u até v. Assim $\lambda(u) = \lambda(q)$ é prefixo próprio de $\lambda(q)\lambda(r) = \lambda(qr) = \lambda(p) = \lambda(v)$.

Suponha agora que $\lambda(u)$ seja um prefixo próprio de $\lambda(v)$. Provaremos que u é ancestral de v por indução em $|\lambda(u)|$. Se $|\lambda(u)| = 0$, temos que u é a raiz. Como v não é a raiz u já que $\lambda(v) \neq 1 = \lambda(u)$, como u é vértice do passeio da raiz a v, temos que u é ancestral de v. Suponhamos agora que $|\lambda(u)| > 0$ e que a hipótese de indução seja válida. Seja w o pai de u. Assim w é o início da última aresta do único caminho da raiz a u e temos que $\lambda(u) = \lambda(w) \cdot \lambda((w, u))$. Ademais, temos que $\lambda(w) \in \text{Pref}(\lambda(u))$ é prefixo próprio de $\lambda(v)$ e w é ancestral de v por hipótese de indução. Assim existem passeios p da raiz a v, q da raiz a w e r de w a v tais que $p = qr$ e $\lambda(v) = \lambda(p) = \lambda(q)\lambda(r) = \lambda(w)\lambda(r)$. Seja a a primeira letra do rótulo da aresta de w a u. Assim, $\lambda(w)a \in \text{Pref}(\lambda(w)\lambda((w, u))) = \text{Pref}(\lambda(u)) \subset \text{Pref}(\lambda(v)) = \text{Pref}(\lambda(w)\lambda(r))$ implica que a seja a primeira letra de $\lambda(r)$, a primeira letra do rótulo da primeira aresta de r. Como existe uma única aresta, a saber (w, u), que parte de w e cujo rótulo começa com a, esta é a primeira aresta de r. Assim u é vértice de r, e portanto de $p = qr$. É imediato que $u \neq v$ já que $\lambda(u)$ é prefixo próprio de $\lambda(v)$. □

Corolário 3.4. *Sejam u e v dois vértices de uma A^+-árvore $\mathbf{T} = (V, E, \lambda)$. Então $u = v$ se, e só se, $\lambda(u) = \lambda(v)$.*

O corolário 3.4 garante que não existem dois vértices distintos de $\mathbf{T}$ com um mesmo rótulo, desta forma o conjunto de vértices finais X (definição 3.2) é unicamente determinado pelo conjunto de palavras W reconhecidas por $\mathbf{T}$ e vice-versa.

Proposição 3.5. *Seja $T = (V, E, \lambda)$ uma A^+-árvore. Sejam u e $v \in V$ e $z = \text{MAC}(u, v)$. Então, $\lambda(z)$ é a palavra mais comprida em $\text{Pref}(\lambda(u)) \cap \text{Pref}(\lambda(v))$.*

Prova. Temos pela proposição 3.3 que se z é ancestral de v então $\lambda(z) \in \text{Pref}(\lambda(v))$. Como o mesmo vale para u, temos que $\lambda(z) \in \text{Pref}(\lambda(u)) \cap \text{Pref}(\lambda(v))$.

Seja P_u o único caminho de z até u. Ele pode ser escrito da forma $P_u = (z, u_0)(u_0, u_1) \cdots (u_n, u)$. Analogamente, seja $P_v = (z, v_0)(v_0, v_1) \cdots (v_n, v)$ o único caminho de z até v. Como z é o MAC de u e v, não há arestas em comum entre P_u e P_v e $(z, v_0) \neq (z, u_0)$, o que implica que as primeiras letras dos rótulos destas arestas são distintas. Sejam elas a e b respectivamente. Assim $a \in \text{Pref}(\lambda(z, u_0)) \subseteq \text{Pref}(\lambda(P_u)) = \text{Pref}(\lambda(z)^{-1}\lambda(u))$. Desta forma $\lambda(z)a \in \lambda(z)\text{Pref}(\lambda(z)^{-1}\lambda(u)) = \text{Pref}(\lambda(u))$. Analogamente, $\lambda(z)b \in \text{Pref}(\lambda(v))$. Isto prova que não existe outro prefixo comum de $\lambda(u)$ e de $\lambda(v)$ mais comprido que $\lambda(z)$, pois caso houvesse um tal prefixo w, ele seria um prefixo de $\lambda(u)$ e teria comprimento não menor que o de $\lambda(z)a$, implicando que $\lambda(z)a$ seria prefixo de w. Analogamente, $\lambda(z)b \in \text{Pref}(w)$, o que levaria a uma contradição já que $a \neq b$. □

Proposição 3.6. *Seja uma A^+-árvore $\mathbf{T} = (V, E, \lambda)$. A função λ estabelece uma bijeção entre os lugares de $\mathbf{T}$ e as palavras de $\mathbf{T}$. Além disso, se w é uma palavra de $\mathbf{T}$ e u é o vértice mais profundo cujo rótulo seja prefixo de w, então $\lambda^{-1}(w) = (u, \lambda(u)^{-1}w)$.*

Prova. Seja (u, x) um lugar qualquer em $\mathbf{T}$. Suponha que $x = 1$. Assim, $\lambda((u, x)) = \lambda(u)x = \lambda(u)$ é uma palavra de $\mathbf{T}$. Suponha agora que $x \neq 1$. Seja então a aresta $e = (u, v)$ tal que $x \in \text{Pref}(\lambda(e))$. Assim, $\lambda((u, x)) = \lambda(u)x \in \lambda(u)\text{Pref}(\lambda(e)) \subseteq \text{Pref}(\lambda(u)\lambda(e)) = \text{Pref}(\lambda(v))$ é uma palavra de $\mathbf{T}$.

Seja agora w uma palavra de $\mathbf{T}$. Nem que seja a raiz, podemos escolher u o vértice mais profundo cujo rótulo seja prefixo de w. Assim, a palavra $\lambda(u)^{-1}w$ é definida. Primeiramente provaremos que $(u, \lambda(u)^{-1}w)$ é um lugar em $\mathbf{T}$. Suponha primeiro o caso em que $\lambda(u) = w$. Então $\lambda(u)^{-1}w = 1$ e $(u, \lambda(u)^{-1}w)$ é certamente um lugar em $\mathbf{T}$, a saber, o próprio vértice u. Suponha agora o caso em que $\lambda(u)^{-1}w \neq 1$. Como w é uma palavra de $\mathbf{T}$, seja um vértice v' tal que w seja um prefixo de seu rótulo e seja p um passeio da raiz até v'. Como $\text{Pref}(\lambda(u)) \subset \text{Pref}(w) \subseteq \text{Pref}(\lambda(v'))$, temos que u é um ancestral de v' devido à proposição 3.3. Podemos definir v o primeiro vértice após u dentro do passeio p. Devido à escolha de u, temos que $\lambda(v) \notin \text{Pref}(w)$. Como w e $\lambda(v)$ são prefixos de $\lambda(p)$, temos que w é prefixo de $\lambda(v)$. Assim $\lambda(u)^{-1}w$ é prefixo de $\lambda(u)^{-1}\lambda(v)$, que é o rótulo da aresta (u, v). Isto prova que $(u, \lambda(u)^{-1}w)$ é um lugar em $\mathbf{T}$, que está na aresta (u, v) neste caso. Por fim, $\lambda((u, \lambda(u)^{-1}w)) = \lambda(u)\lambda(u)^{-1}w = w$. □

3.3 A árvore de Ukkonen

Nesta seção vamos definir a árvore de Ukkonen, a principal estrutura combinatória deste trabalho. Retomando o exemplo da figura 3.3, uma A^+-árvore que reconhece $W = \text{Suf}(abaabc) \setminus \{1\}$, observe que os rótulo dos vértices internos são

$$\{1, a, b, ab\}.$$

Cada palavra deste conjunto é prefixo próprio de pelo menos duas palavras distintas de W. Por exemplo, ab é prefixo de abc e de $abaabc$. Observe que a árvore mostrada é compacta. Como sua última letra c ocorre somente uma vez, não existem dois sufixos tais que um é prefixo do outro. Queremos generalizar esta construção, mas antes precisamos de algumas definições.

Dado um conjunto de palavras W, dizemos que W é *livre de prefixos* se não existirem duas palavras distintas em W tais que uma delas seja prefixo próprio da outra. Dizemos que uma palavra x é *prefixo de* W se x for prefixo de alguma palavra de W. Dizemos ainda que

x é uma *bifurcação de* W se x for prefixo de W e se existirem letras distintas $a, b \in A$ tais que xa e xb também sejam prefixos de W. Como vimos acima $x = ab$ é uma bifurcação de $W = \mathrm{Suf}(abaabc) \setminus \{1\}$. Observe que isto implica que W tenha ao menos duas palavras não vazias já que xa e xb não são prefixos da palavra vazia e não podem ser prefixos de uma mesma palavra.

O *conjunto das bifurcações de* W é denotado por $\mathrm{Bifurc}(W)$. Observe que o conjunto $W = \mathrm{Suf}(abaabc) \setminus \{1\}$ reconhecido pela A^+-árvore da figura 3.3 é livre de prefixos e que cada rótulo de cada vértice interno é uma bifurcação deste conjunto. Dizemos que uma palavra x é *prefixo próprio de* W se x for prefixo próprio de alguma palavra de W. Como uma bifurcação de W é prefixo próprio de W, $\mathrm{Bifurc}(W)$ é disjunto de W se W for livre de prefixos. Finalmente, podemos generalizar esta construção, da mesma maneira que foi feita em [dLS03], para um conjunto qualquer de palavras W.

Definição 3.7 (árvore de Ukkonen)**.** *Dado um conjunto de palavras* W*, a* árvore de Ukkonen de W *é a* A^+*-árvore* $\mathbf{T} = (V, E, \lambda)$ *tal que,*

$$
\begin{aligned}
&V = W \cup \mathrm{Bifurc}(W) \cup \{1\},\\
&E = \{(u, v) \in V \times V \mid u = \max_{|x|}\{x \in V \mid x \text{ é prefixo próprio de } v\}\},\\
&\lambda : \begin{array}{ccc} E & \longrightarrow & A^+ \\ (u,v) & \longrightarrow & u^{-1}v. \end{array}
\end{aligned}
$$

A A^+-árvore da figura 3.3 é a árvore de Ukkonen do conjunto $\mathrm{Suf}(abaabc) \setminus \{1\}$. Contudo, não é óbvio que a árvore de Ukkonen definida desta maneira é realmente uma A^+-árvore ou mesmo uma árvore. De fato, o teorema 3.8 [dLS03] prova que a árvore de Ukkonen $\mathbf{T} = (V, E, \lambda)$ definida em 3.7 é uma A^+-árvore que reconhece W. Este teorema, além de garantir que a árvore de Ukkonen está bem definida, demonstra que as palavras de $\mathbf{T}$ são exatamente os prefixos de W e caracteriza as folhas e os vértices internos de $\mathbf{T}$.

Teorema 3.8. *Seja* W *um conjunto qualquer de palavras, seja* $\mathbf{T} = (V, E, \lambda)$ *sua árvore de Ukkonen e seja* $F \subseteq W$ *o conjunto das palavras de* W *que não são prefixos próprios de* W*. Então:*

1. $\mathbf{T}$ *é uma árvore enraizada trivial*[1] *se e só se* $W = \emptyset$ *ou* $W = \{1\}$*;*
2. $\mathbf{T}$ *é uma* A^+*-árvore (com raiz* 1*);*
3. $\lambda(v) = v$ *para todo vértice* v*;*
4. $\mathbf{T}$ *reconhece* W *(com vértices finais* W*);*

[1] Uma árvore enraizada trivial é uma árvore com apenas um vértice (a raiz) e nenhuma aresta.

5. *o conjunto das palavras de* $\mathbf{T}$ *é o conjunto dos prefixos de* W*;*

6. *as folhas de* $\mathbf{T}$ *são* F *e os nós internos são* $\text{Bifurc}(W) \cup \{1\} \cup (W \setminus F)$*;*

7. *os nós internos com pelo menos dois filhos são* $\text{Bifurc}(W)$*;*

Prova. A partir da definição, temos claramente que $V = \emptyset$ se e só se $W \subseteq \{1\} = V$, o que prova o item 1.

Vamos provar o item 2. Caso $W = \emptyset$, temos que $\mathbf{T}$ é A^+-árvore trivial. Suporemos que $W \neq \emptyset$ a partir de agora. Seja $v \in V$. Se $v = 1$, como não há prefixo próprio de v temos que v não é término de nenhuma aresta. Se $v \neq 1$, temos que $1 \in V$ é um prefixo próprio de v e que v é término de uma única aresta: daquela que parte de u, seu prefixo próprio mais comprido que está em V. Por indução no comprimento de v podemos provar que existe um único caminho de 1 a v em $\mathbf{T}$: aquele caminho de 1 a u mais a aresta (u, v). Isto prova que $\mathbf{T}$ é árvore com raiz 1. As arestas (u, v) têm rótulos não vazios $u^{-1}v$ já que u é prefixo próprio de v. Sejam duas arestas distintas (u, v) e (u, v'). Seja x o mais comprido prefixo comum de v e de v'. Suponha por absurdo que $x = v'$. Da escolha da aresta (u, v'), temos que $\text{Pref}(u) \subset \text{Pref}(v')$. Da escolha de x, temos que $\text{Pref}(v') = \text{Pref}(x) \subseteq \text{Pref}(v)$. Como as arestas (u, v) e (u, v') são distintas, temos que $\text{Pref}(v) \neq \text{Pref}(v')$ e, portanto, $\text{Pref}(u) \subset \text{Pref}(v') \subset \text{Pref}(v)$ e u não é a palavra de V mais comprida que é prefixo próprio de v, o que contradiz com a escolha da aresta (u, v). Donde, $x \neq v'$. Analogamente, $x \neq v$. Da escolha de x, temos que a primeira letra de $x^{-1}v$, chamemo-la a, é distinta da primeira letra de $x^{-1}v'$, chamemo-la b. Como $V \subseteq \text{Pref}(W)$, temos que $xa \in \text{Pref}(x\,x^{-1}v) = \text{Pref}(v) \subseteq \text{Pref}(W)$ e que $xb \in \text{Pref}(x\,x^{-1}v') = \text{Pref}(v') \subseteq \text{Pref}(W)$. Assim $x \in \text{Bifurc}(W) \subseteq V$. Como u é um prefixo comum de v e de v', da escolha de x temos que $\text{Pref}(u) \subseteq \text{Pref}(x) \subset \text{Pref}(v)$. Do fato de u ser o prefixo mais comprido de v que está em V e $x \in V$, temos que $x = u$, que a é a primeira letra de $x^{-1}v = u^{-1}v = \lambda((u, v))$, e que b é a primeira letra de $x^{-1}v' = u^{-1}v' = \lambda((u, v'))$ e que $a \neq b$. Isto completa a prova de que $\mathbf{T}$ é uma A^+-árvore de raiz 1.

Vamos provar os itens 3 e 4. Seja $v_0, v_1, v_2, \ldots, v_k$, com $1 = v_0$ e $v_k = v$, a sequência de vértices percorridos pelo passeio p da raiz até um vértice qualquer v. Então

$$\begin{aligned}
\lambda(v) &= \lambda(p) \\
&= \lambda((v_0, v_1))\,\lambda((v_1, v_2))\,\lambda((v_2, v_3)) \cdots \lambda((v_{k_1}, v_k)) \\
&= (v_0^{-1}v_1)(v_1^{-1}v_2)(v_2^{-1}v_3) \cdots (v_{k-1}^{-1}v_k) \\
&= v_0^{-1}v_k = 1^{-1}v \\
&= v.
\end{aligned}$$

Assim, $\lambda(v) = v$ e $W \subseteq V$ é reconhecida por $\mathbf{T}$ em $X = W$.

Vamos provar o item 5. Seja w uma palavra de $\mathbf{T}$ e seja $v \in V$ tal que $w \in \text{Pref}(\lambda(v))$. Assim, $w \in \text{Pref}(\lambda(v)) = \text{Pref}(v) \subseteq \text{Pref}(W)$ já que toda palavra em V é prefixo de W. Seja agora u um prefixo de W e seja $w \in W$ tal que $u \in \text{Pref}(w)$. Como $w \in W \subseteq V$ e $u \in \text{Pref}(w) = \text{Pref}(\lambda(w))$, temos que u é palavra de $\mathbf{T}$.

Vamos provar o item 6. Suponha que $u \in V$ seja uma folha. Assim u não é ancestral de nenhum vértice em W e, devido à proposição 3.3, $\lambda(u) = u$ não é prefixo próprio de $\lambda(w) = w$ para nenhum $w \in W$. Assim $u \in F$. Suponha agora que $u \in V$ seja um vértice interno. Assim u é pai de algum vértice-filho $v \in V$ e existe a aresta (u, v). Assim, u é prefixo próprio de $v \in V \subseteq \text{Pref}(W)$, e $u \notin F$. Isto prova que as folhas de $\mathbf{T}$ são F. Como $F \cap \text{Bifurc}(W) = \emptyset$ já que toda bifurcação de W é prefixo próprio de W, como $\{1\} \cap F = \emptyset$ pois 1 é prefixo próprio de W sempre que W possui uma palavra não vazia, temos que os vértices internos são

$$\begin{aligned} V \setminus F &= (\text{Bifurc}(W) \cup \{1\} \cup W) \setminus F \\ &= ((\text{Bifurc}(W) \cup \{1\}) \setminus F) \cup (W \setminus F) \\ &= (\text{Bifurc}(W) \cup \{1\}) \cup (W \setminus F). \end{aligned}$$

Vamos provar o item 7. Seja $u \in \text{Bifurc}(W)$ um vértice interno. Como u é uma bifurcação de W, sejam a e b letras distintas tais que ua e ub sejam prefixos de W. Assim ua é palavra de $\mathbf{T}$. Suponha o caso em que $ua \in V$. Neste caso, (u, ua) é certamente uma aresta de E cujo rótulo começa com a. Suponha agora o caso em que $ua \notin V$. Neste caso, usando a proposição 3.6, o lugar de ua em $\mathbf{T}$ é o lugar (u, a), que está na aresta que parte de u e cujo rótulo começa com a. Em qualquer caso, existe uma aresta que parte de u e cujo rótulo começa com a. Analogamente, existe uma aresta que parte de u e cujo rótulo começa com b. Assim u tem pelo menos dois filhos. Seja x um vértice interno com pelo menos dois filhos, y e z. Sejam a a primeira letra do rótulo da aresta (x, y) e b a primeira letra do rótulo da aresta (x, z). Como $\mathbf{T}$ é A^+-árvore temos que $a \neq b$. Por definição, (x, a) e (x, b) são dois lugares em $\mathbf{T}$. Pela proposição 3.6, temos que $\lambda((x, a)) = \lambda(x)a = xa$ e $\lambda((x, b)) = xb$ são duas palavras de $\mathbf{T}$, e portanto são dois prefixos de W. Assim, x é uma bifurcação de W, completando a prova. □

Se tivermos que $W \neq \{1\}$ e W livre de prefixos, além das hipótese do teorema 3.8. Então, $W = F$ (as folhas da árvore são W) e todos vértices internos, exceto a raiz, tem pelo menos dois filhos. Desta forma, segue imediatamente o corolário 3.9.

Corolário 3.9. *Se além das hipóteses do último teorema tivermos que W é livre de prefixos, então temos que: as folhas são W; os vértices internos são* $\text{Bifurc}(W) \cup \{1\}$ *e $\mathbf{T}$ é uma A^+-árvore compacta.*

A árvore de Ukkonen da figura 3.3 é uma árvore compacta, todos os seus vértices internos

tem pelo menos dois filhos. Esta árvore reconhece o conjunto $W = \mathrm{Suf}(abaabc) \setminus \{1\}$, que é livre de prefixos. A figura 3.4 mostra um exemplo de árvore de Ukkonen não compacta, ela reconhece o conjunto $W = \mathrm{Suf}(abaab) \setminus \{1\}$ que não é livre prefixos, uma vez que ab é prefixo de $abaab$. O conjunto das bifurcações desta árvore é $\{1, a\}$, que são justamente os vértices internos com mais de um filho. Os outros vértices internos $\{ab, b\} = W \setminus F$ possuem apenas um filho. E as folhas são $\{abaab, baab, aab\} = F$, o conjunto das palavras de W que são livres de prefixo.

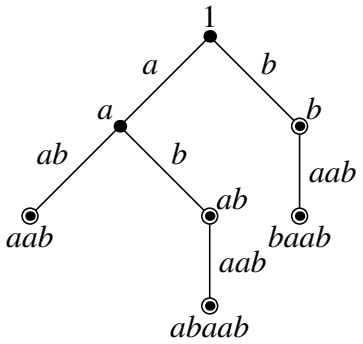


Figura 3.4: *Árvore de Ukkonen de* $\mathrm{Suf}(abaab) \setminus \{1\}$*, um exemplo de árvore não compacta.*

3.4 Tamanho e Bifurcações

Da maneira como foi definida a árvore de Ukkonen de um dado conjunto de palavras W, tem a sua topologia determinada por W, nesta seção comparamos duas árvores de Ukkonen cujos os conjuntos de palavras diferem por apenas uma palavra. Como uma consequência do lema 3.10, que faz esta comparação entre as duas árvores, obtemos uma limitação no número de vértices e arestas para uma árvore de Ukkonen que reconhece um conjunto de n palavras, enunciado no teorema 3.11

Se $W \neq \emptyset$, mesmo que uma palavra $w \in A^*$ não seja uma palavra da árvore de Ukkonen $\mathbf{T}$ de W, sempre existe um prefixo de w que está representado em $\mathbf{T}$ (no pior caso, a palavra vazia), ou seja, o conjunto $\mathrm{Pref}(W) \cap \mathrm{Pref}(w)$ é não vazio. Chamamos o maior prefixo deste conjunto de *cabeça* de w em $\mathbf{T}$, e o denotamos h. O próximo lema faz uma comparação entre o conjunto de vértices das árvores de Ukkonen $\mathbf{T}$ e $\mathbf{T}'$ dos conjuntos W e W', bem como o conjunto de bifurcações de W e W'. Os vértices w e h (possivelmente igual a w) são os dois vértices que eventualmente precisam ser adicionados a $\mathbf{T}$ para que se obtenha $\mathbf{T}'$.

Lema 3.10. *Seja w um palavra de A^*, W e W' dois conjuntos de palavras não vazios tais que $W' = W \cup \{w\}$. Sejam $\mathbf{T} = (V, E, \lambda)$ e $\mathbf{T}' = (V', E', \lambda')$ as suas respectivas árvore de Ukkonen*

e h a cabeça de w em $\mathbf{T}$. Então,

$$\begin{aligned} V' &= V \cup \{h, w\}, \\ \text{Bifurc}(W) \subseteq \text{Bifurc}(W') &\subseteq \text{Bifurc}(W) \cup \{h\}. \end{aligned}$$

Sendo que $\text{Bifurc}(W') \neq \text{Bifurc}(W)$ *se, e somente se,* $h \neq w$, $h \notin \text{Bifurc}(W)$ *e* $\exists y \in W$ *tal que* $\text{Pref}(h) \subset \text{Pref}(y)$.

Prova. Vamos provar que $\text{Bifurc}(W') \setminus \text{Bifurc}(W) \subseteq \{h\}$. Se existe $x \in \text{Bifurc}(W') \setminus \text{Bifurc}(W)$, então existem duas letras distintas a, b tais que $x, xa \in \text{Pref}(w)$ e $x, xb \in \text{Pref}(W)$. Como x não é uma bifurcação de W, segue que $xa \notin \text{Pref}(W)$, ou seja, x é a palavra mais comprida em $\text{Pref}(w) \cap \text{Pref}(W)$, o que implica que $x = h$ e $\text{Bifurc}(W') \setminus \text{Bifurc}(W) \subseteq \{h\}$. Portanto, $\text{Bifurc}(W') \subseteq \text{Bifurc}(W) \cup \{h\}$.

A outra inclusão $\text{Bifurc}(W) \subseteq \text{Bifurc}(W')$ segue diretamente do fato de que o conjunto das bifurcações de qualquer conjunto de palavras depende somente do conjunto de prefixos destas palavras e por hipóteses temos que $W \subseteq W'$.

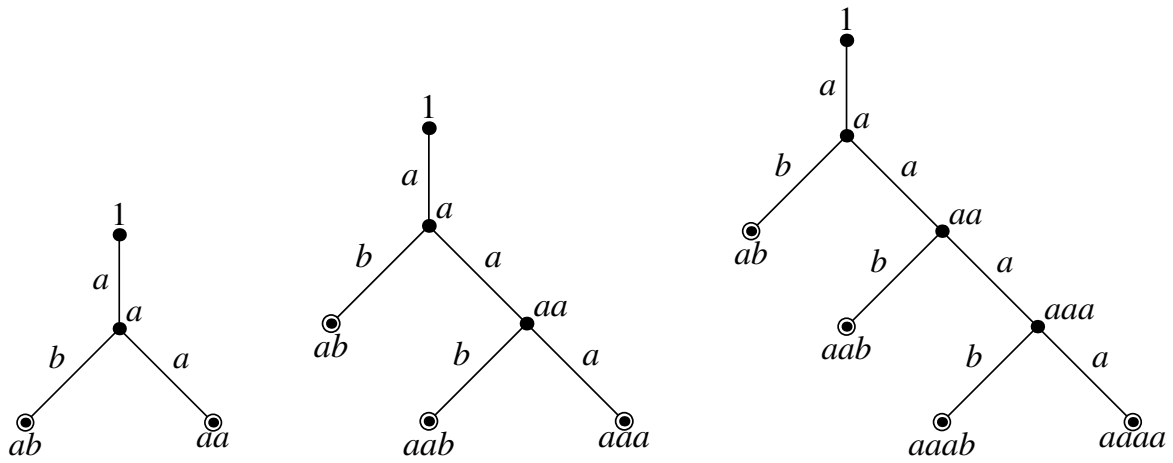
Vamos provar que $\text{Bifurc}(W') \neq \text{Bifurc}(W)$ se, e somente se, $w \neq h$, $\exists y \in W$ tal que $\text{Pref}(h) \subset \text{Pref}(y)$ e $h \notin \text{Bifurc}(W)$. Suponha que $w \neq h$ e $h \notin \text{Bifurc}(W)$, então existem duas letras distintas a, b tais que $ha \in \text{Pref}(w)$ e $hb \in \text{Pref}(W)$. Logo h é uma bifurcação de W' e a terceira hipótese garante que $h \notin \text{Bifurc}(W)$. Portanto, $\text{Bifurc}(W') \neq \text{Bifurc}(W)$. A outra direção segue dos seguintes três fatos. Se $h = w$, temos que $w \in \text{Pref}(W) = \text{Pref}(W')$ e $\text{Bifurc}(W') = \text{Bifurc}(W)$. Se $\nexists y \in W$ tal que $\text{Pref}(h) \subset \text{Pref}(y)$, h não é uma bifurcação de W' e como $\text{Bifurc}(W') \subseteq \text{Bifurc}(W) \cup \{h\}$, segue que $\text{Bifurc}(W') = \text{Bifurc}(W)$. Por fim, se $h \in \text{Bifurc}(W)$, temos que $\text{Bifurc}(W) \subseteq \text{Bifurc}(W') \subseteq \text{Bifurc}(W) \cup \{h\} = \text{Bifurc}(W)$.

Vamos provar que $V' = V \cup \{h, w\}$. Se $\text{Bifurc}(W') \neq \text{Bifurc}(W)$, temos que $\text{Bifurc}(W') = \text{Bifurc}(W) \cup \{h\}$ e daí segue que $V' = V \cup \{h, w\}$. Por outro lado, se tivermos que $\text{Bifurc}(W') = \text{Bifurc}(W)$, existem duas possibilidades. Se $h = w$ ou $h \in \text{Bifurc}(W)$, $V \cup \{h, w\} = V \cup \{w\} = V'$. Se $\nexists y \in W$ tal que $\text{Pref}(h) \subset \text{Pref}(y)$, como $h \in \text{Pref}(w) \cap \text{Pref}(W)$, temos que $h \in W \subseteq V$ e $V \cup \{h, w\} = V \cup \{w\} = V'$. □

Teorema 3.11. *Seja W um conjunto com n palavras e $\mathbf{T} = (V, E, \lambda)$ a árvore de Ukkonen de W. Então, temos que $|V| \leq 2n$ e $|E| \leq 2n - 1$.*

Prova. Da definição da árvore de Ukkonen temos que $|V| = |\text{Bifurc}(W)| + |W| + 1$. Vamos limitar o tamanho do conjunto de bifurcações. Se W tem somente uma palavra, então $\text{Bifurc}(W)$ é vazio, usando isto como o caso base da indução e a segunda parte do lema 3.10 como passo indutivo, temos que $|\text{Bifurc}(W)| \leq n - 1$. Logo, $|V| \leq 2n$ e como $\mathbf{T}$ é uma árvore $|E| \leq 2n - 1$. □

A limitação para o número de vértices dada por este teorema é justa. Na figura 3.5 temos exemplos de árvore de Ukkonen com $n = 2, 3, 4$ palavras que possuem $2n$ vértices. É fácil generalizar este exemplo para um número qualquer de palavras.



(a) $W = \{aa, ab\}$ (b) $W = \{ab, aab, aaa\}$ (c) $W = \{ab, aab, aaab, aaaa\}$

Figura 3.5: *Exemplos de árvores de Ukkonen com número máximo de vértices.*

3.5 Um Algoritmo Ingênuo

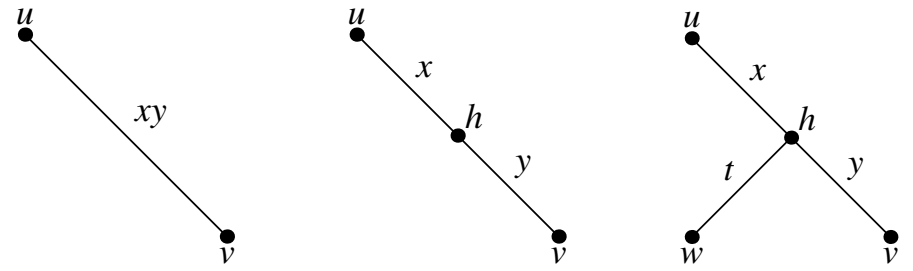
Nesta seção usaremos o lema 3.10 para obter um algoritmo incremental simples que constrói a árvore de Ukkonen de um conjunto qualquer de palavras. O lema 3.10 garante que ao adicionarmos uma palavra w à árvore de Ukkonen $\mathbf{T}$ de W, temos que adicionar, se ainda não estiverem presentes, dois vértices: w e h (a cabeça de w em $\mathbf{T}$). Para tanto, o algoritmo 3.2 faz uso da função auxiliar EncontraCabeça$(w, \mathbf{T})$, que percorre a árvore $\mathbf{T}$ a partir da raiz e retorna a cabeça h e o seu lugar (u, x) em $\mathbf{T}$.

Depois que o lugar de h em $\mathbf{T}$ é encontrado o algoritmo 3.2 atualiza a árvore $\mathbf{T}$, inserindo, possivelmente, dois novos vértices h e w. Se o lugar (u, x) de h estiver em uma aresta, devemos quebrar esta aresta, adicionando o novos vértices h e w e as arestas correspondentes, como mostrado na figura 3.6. Se o lugar estiver em um vértice, inserimos apenas w e a aresta correspondente. Por fim, se $h = w$ não há nada a fazer. Na figura 3.7 temos um exemplo de aplicação do algoritmo 3.2 com conjunto $W = \mathrm{Suf}(abaabc) \setminus \{1\}$.

A corretude do algoritmo 3.2 segue diretamente do lema 3.10 e da definição de árvore de Ukkonen. A complexidade do algoritmo é dominada pelo tempo gasto com a função EncontraCabeça, que é proporcional ao comprimento de cada cabeça h. Logo, o tempo total gasto com o algoritmo é limitado por $\sum_{w \in W} |w|$. Se não fizermos nenhuma hipótese adicional sobre o conjunto

Algoritmo 3.1 Função EncontraCabeça

```
EncontraCabeça(w, T)
 1  ▷ w: palavra cuja cabeça deve ser encontrada
 2  ▷ devolve o lugar (u, x) da cabeça de w em T
 3  u ← 1
 4  s ← w
 5  enquanto s ≠ 1 e existe uma aresta e = (u, v) ∈ E tal que λ(e) ∈ Pref(s) faça
 6          s ← λ(e)^{-1} s
 7          u ← v
 8  se existe uma aresta e = (u, v) ∈ E tal que s[1] ∈ Pref(λ(e)) então
 9          x ← o prefixo comum mais comprido entre λ(e) e s
10  senão
11          x ← 1
12  devolva (λ(u)x, u, x)
```



(a) Aresta (u, v) de rótulo xy

(b) Quebra de (u, v) em (u, h) mais (h, v)

(c) Aresta (h, w) de rótulo t

Figura 3.6: *Quebrando arestas*

Algoritmo 3.2 Algoritmo de construção da árvore de Ukkonen

```
UkkonenTree(W)
 1  ▷ W: conjunto de palavras que serão reconhecidos pela árvore de Ukkonen
 2  T ← árvore com raiz 1
 3  para cada w ∈ W faça
 4          (h, u, x) ← EncontraCabeça(w, T)
 5          t ← h^{-1} w
 6          se |x| > 0 então
 7                  (u, v) ← a aresta associada ao lugar (u, x) em T
 8                  acrescente novo vértice h
 9                  remova a aresta (u, v)
10                  acrescente aresta de u para h de rótulo x
11                  acrescente aresta de h para v de rótulo h^{-1} v
12          se |t| > 0 então
13                  acrescente nova folha w
14                  acrescente aresta de h para w de rótulo t
15  devolva T
```

W, qualquer algoritmo de construção da árvore de Ukkonen de W deve ter uma cota inferior de $\Omega(\sum_{w \in W} |w|)$, pois é necessário ler toda a entrada. Ou seja, este algoritmo ingênuo é também um algoritmo ótimo para a construção das árvores de Ukkonen.

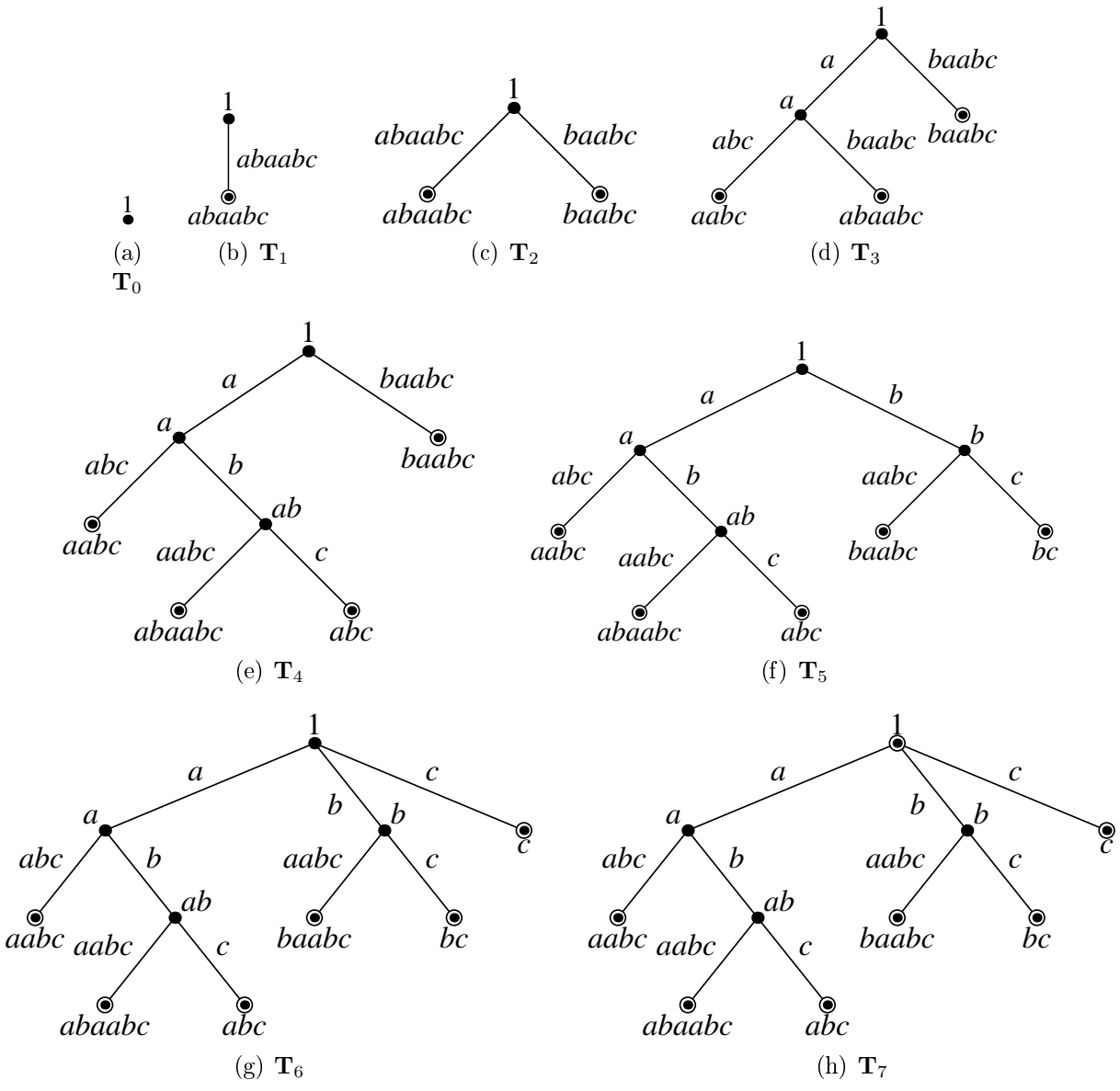


Figura 3.7: *Sucessivas árvores de Ukkonen* $\mathbf{T_i}$ *de* W_i*, onde* $W_0 = \emptyset$ *e* $W_i = W_{i-1} \cup \{s[i..|s|]\}$ *para* $s = abaabc$ *e* $i = 1, \ldots, |s| + 1$.

Além disso, observando mais atentamente o algoritmo 3.2 fica claro que a ordem com que as palavras são inseridas não é importante, elas podem ser inseridas em qualquer ordem. No exemplo da figura 3.7 a ordem escolhida foi decrescente com o comprimento da palavra, mas poderia ter sido qualquer outra.

3.6 Minimalidade

Na próxima proposição e em seu corolário veremos que, dado um conjunto de palavras W, sua árvore de Ukkonen é a menor A^+-árvore $\mathbf{T} = (V, E, \lambda)$ que reconhece W. Para ser mais preciso, é a menor no sentido em que sempre existe uma injeção, que preserva rótulos e caminhos, dos vértices da árvore de Ukkonen para os vértices de qualquer A^+-árvore $\mathbf{T}$ que reconhece o mesmo conjunto de palavras e cada aresta na árvore de Ukkonen corresponde a um caminho em $\mathbf{T}$ com o mesmo rótulo. Em outras palavras, a árvore de Ukkonen é um *minor* de qualquer A^+-árvore que reconhece o mesmo conjunto de palavras.

Estamos usando *minor* no sentido usual encontrado na literatura da teoria de grafos. Dadas duas A^+-árvores $\mathbf{T}$ e $\mathbf{T}'$, dizemos que $\mathbf{T}$ *é um minor de* $\mathbf{T}'$ se $\mathbf{T}$ é isomórfico à A^+-árvore obtida a partir de $\mathbf{T}'$ por sucessivas remoções de folhas, suas respectivas arestas e/ou contração de arestas. Dados três vértices u, v e w, com arestas de u para v e de v para w, a *contração dessas arestas*, remove essas duas arestas, o vértice v e acrescenta uma aresta de u para w com rótulo igual a concatenação dos rótulos das arestas anteriores, desta maneira os rótulos dos vértices u e w permanecem inalterados.

Proposição 3.12. *Seja $W \subseteq A^*$ e seja $\mathbf{T} = (V, E, \lambda)$ uma A^+-árvore que reconhece W. Então, $W \cup \text{Bifurc}(W) \cup \{1\} \subseteq \lambda(V)$.*

Prova. Os casos em que $|W| \leq 1$ são triviais. Vamos assumir então que W tem pelo menos duas palavras. Pela definição de A^+-árvore que reconhece W, $W \subseteq \lambda(V)$ e $\{1\} \subseteq \lambda(V)$. Faltanos provar que $\text{Bifurc}(W) \subseteq \lambda(V)$. Seja $x \in \text{Bifurc}(W)$. Existem $a, b \in \Sigma$, com $a \neq b$, tais que $xa, xb \in \text{Pref}(W)$. Assim, existem $u, v \in W \subseteq \lambda(V)$ distintos tais que $xa \in \text{Pref}(u)$ e $xb \in \text{Pref}(v)$. Como T reconhece $W \supseteq \{u, v\}$, temos que $\lambda^{-1}(u), \lambda^{-1}(v)$ são vértices de T e existe $\text{MAC}(\lambda^{-1}(u), \lambda^{-1}(v))$. Como, x é o prefixo comum mais comprido de u e de v e sabemos pela proposição 3.5 que $x = \lambda(\text{MAC}(\lambda^{-1}(u), \lambda^{-1}(v)))$, temos que $x \in \lambda(V)$. Isto prova que $\text{Bifurc}(W) \subseteq \lambda(V)$, o que completa a demonstração. □

Usando a a proposição anterior e a proposição 3.3, o próximo corolário segue imediatamente.

Corolário 3.13. *Seja $W \subseteq A^*$ e $T = (V, E, \lambda)$ uma A^+-árvore que reconhece W. Toda aresta (u, v) na árvore de Ukkonen de W corresponde a um caminho de $\lambda^{-1}(u)$ até $\lambda^{-1}(v)$ em T. Além disso, eles possuem o mesmo rótulo $u^{-1}v$.*

Removendo todos os vértices que não pertencem a nenhum caminho da raiz até um vértice final e usando a última proposição e corolário, segue imediatamente o próximo teorema.

Teorema 3.14. *Seja $W \subseteq A^*$ um conjunto qualquer de palavras, a árvore de Ukkonen de W é um* minor *de qualquer A^+-árvore que reconhece W.*

Capítulo 4

Árvore de Sufixos (Generalizada)

Face à quantidade de aplicações, de certa forma as árvores de sufixos dispensam maiores apresentações. Desde que foram introduzidas por Weiner [Wei73] elas encontraram um extenso uso na resolução de problemas sobre palavras. O problema do fator comum mais longo [Gus97] é uma destas aplicações: dadas duas palavras s e t queremos encontrar o fator comum mais longo entre estas palavras. Uma das soluções eficientes mais simples para este problema [dLS03] envolve a construção da árvore de sufixos para s e t conjuntamente, ou seja, uma única árvore de sufixos que contém todos os sufixos de s e de t. Esta generalização natural das árvores de sufixos é chamada de *árvore de sufixos generalizada* [Gus97].

Muito autores[1] [Wei73, Ukk95, GK97, Far97, Gus97] definem as árvores de sufixos como *tries* compactas ou árvores PATRICIA [Mor68] para o conjunto de sufixos não vazios de uma palavra w. Uma trie [CHL07] é um autômato finito determinístico que reconhece os sufixos de w em que dois caminhos distintos partindo de um mesmo estado terminam em estados distintos, ou equivalentemente, é uma A^+-árvore que reconhece $\mathrm{Suf}(w)$ em que todas as arestas tem rótulos de comprimento um. Uma trie compacta ou árvore PATRICIA é uma trie em que todos vértices com apenas um filho são contraídos (vértices implícitos). Nesta definição geralmente se exige que a última letra de w seja distinta. Trata-se de uma definição *intrínseca*, mas não *explícita*, a estrutura da árvore não é óbvia *a priori*. Com exceção das folhas (que são os sufixos de w) e da raiz, não sabemos quais são os vértices, as arestas e nem os seu rótulos.

Neste capítulo apresentaremos as árvores de sufixos e as árvores de sufixos generalizadas como um caso particular das árvores de Ukkonen. Uma vantagem desta abordagem é que a definição 3.7 é tanto *intrínseca*, a topologia da árvore é completamente determinada pelo conjunto de palavras que ela reconhece, quanto *explícita*, toda a estrutura da árvore (quais são os vértices, arestas e rótulos) é dada na definição. Além disso, as propriedade estudas no capítulo 3 também

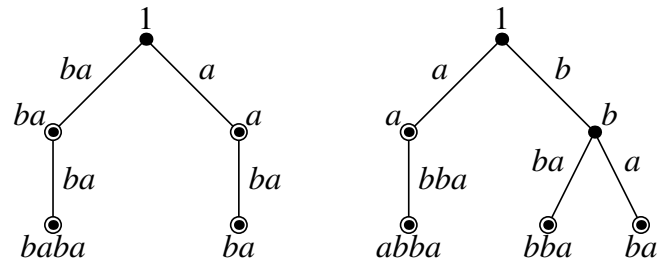
[1] Gusfield [Gus97] e Weiner [Wei73] não citam explicitamente as tries, mas as suas definições são muito próximas daquelas que usam tries.

são válidas para estes casos especiais, em particular o teorema 3.14, a propriedade minimal. Em outras palavras, se definidas desta forma a árvore de sufixos e a árvore generalizada são as menores estruturas em forma de árvore que reconhecem um conjunto de sufixos de uma palavra e o super conjunto de sufixos de várias palavras, respectivamente.

4.1 Definições

A árvore de sufixos [Wei73, McC76, Ukk95, Gus97, dLS03] de uma palavra w é árvore de Ukkonen que reconhece o conjunto dos sufixos não vazios de w. Os exemplos das figuras figuras 3.3 e 3.4 são as árvores de sufixo de *abaabc* e *abaab*, respectivamente. Na figura 4.1, são apresentados mais dois exemplos de árvore de sufixos. E a definição formal, como um caso particular da árvore de Ukkonen, é apresentada na definição 4.1.

Definição 4.1 (Árvore de sufixos)**.** *Seja $w \in A^*$ uma palavra e seja $W = \mathrm{Suf}(w) \setminus \{1\}$ o conjunto dos sufixos não vazios de w. A* árvore de sufixos *de w é a árvore de Ukkonen de W.*



(a) Árvore de sufixo de $w = baba$

(b) Árvore de sufixo de $w = abba$

Figura 4.1: *Exemplos de árvores de sufixos.*

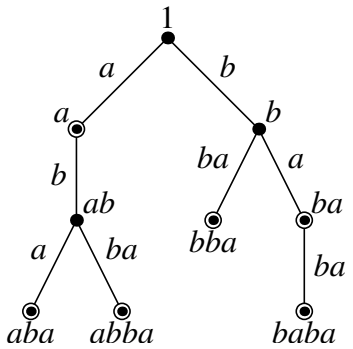


Figura 4.2: *Árvore de sufixos generalizada de $H = \{abba, baba\}$.*

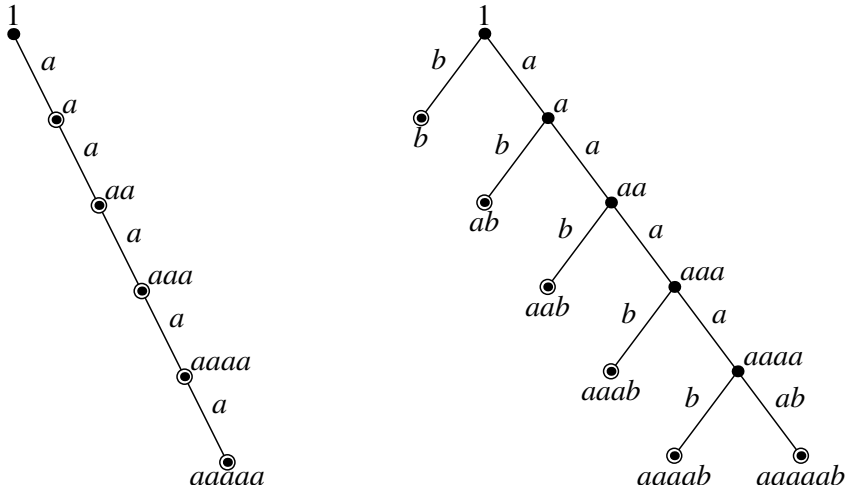
A árvore de sufixos generalizada [Gus97] de um conjunto de palavras H é a árvore de

Ukkonen do conjunto de todos os sufixos não vazios das palavras em H. A definição formal é apresentada na definição 4.2 e um exemplo com duas palavras é dado na figura 4.2.

Definição 4.2 (Árvore de sufixos generalizada)**.** *Seja $H \subset A^*$ um conjunto finito de palavras e seja $W = \cup_{w \in H} \text{Suf}(w) \setminus \{1\}$ o conjunto de todos os sufixos não vazios das palavras em H. A* árvore de sufixos generalizada *de H é a árvore de Ukkonen de W.*

4.2 A Última Letra Distinta

Vale a pena observar que na definição 4.1, diferentemente[2] de [Gus97, Ukk95, McC76, Wei73], não exigimos que a última letra de w não tenha aparecido antes, ou seja, que $\text{Suf}(w)$ seja livre de prefixos. A consequência disto, como garante o corolário 3.9, é que nem todos os sufixos serão necessariamente folhas e nossa árvore não será necessariamente compacta. Na figura 4.3(a), temos um uma árvore de sufixos não compacta e, na figura 4.3(b), temos uma árvore compacta.



(a) Árvore de sufixos de $w = aaaaa$

(b) Árvore de sufixos de $w = aaaaab$

Figura 4.3: *Exemplos de árvores de sufixos.*

Nestes exemplos, estão representadas as árvores de sufixos de *aaaaa* e *aaaaab*. Ou seja, na figura 4.3(b) temos a árvore de sufixos de *aaaaa* após a concatenação de uma última letra distinta. Repare que a árvore da figura 4.3(a) possui 6 vértices, enquanto a outra possui 11 vértices. Houve um aumento considerável no número de vértices. Em geral, se $|V|$ é número de vértices da árvore de sufixos $\mathbf{T} = (V, E, \lambda)$ de w, então a árvore de sufixos de $w\#$ pode ter até $2|V| - 1$ vértices, como ocorre nos exemplos da figura 4.3. Portanto, não exigindo que

[2] Crochemore et al. [CHL07] também não exige que a última letra seja distinta.

última letra seja distinta, a definição 4.1 das árvores de sufixo resulta em uma estrutura mais econômica no uso de memória do que as apresentadas em [Gus97, Ukk95, McC76, Wei73].

4.3 A Representação das Árvores de Sufixos

Na seção 3.5 argumentamos que a cota inferior para construção da árvore de Ukkonen de W era $\Omega(\sum_{x\in W}|x|)$. No caso da árvore de sufixos de uma palavra w de comprimento n, teríamos que

$$\sum_{x\in W}|x| = \sum_{x\in \mathrm{Suf}(w)}|x| = \sum_{i=1}^{n} i = \Omega(n^2).$$

Ou seja, a cota inferior para construção das árvores de sufixos seria $\Omega(n^2)$. Aqui temos, porém, uma informação adicional importante sobre W. Aquela argumentação não se aplica ao caso das árvores de sufixos porque é possível "ler" todas as palavras de W em tempo $O(n)$, afinal *todas as palavras de W são sufixos de w.*

Existe ainda um outro ponto que poderia proibir um algoritmo linear para a construção das árvores de sufixos: a representação dos rótulos. Qualquer representação computacional da árvore de Ukkonen $\mathbf{T} = (V, E, \lambda)$ de W, deve incluir pelo menos os rótulos da arestas. No caso da árvore de sufixos de w, com $n = |w|$, se associássemos a cada aresta uma palavra, seriam necessários $\sum_{i=1}^{n} i = \Omega(n^2)$ letras para representar todos os rótulos das arestas. O que, naturalmente, impossibilitaria um algoritmo linear. Este problema é resolvido se em cada rótulo ao invés de guardarmos uma cópia de um fator de w, guardarmos dois números: a posição de início[3] da ocorrência do fator em w e o seu tamanho. Assumiremos esta representação compacta dos rótulos de agora em diante, inclusive para a árvore esparsa de sufixos e para a árvore de k-fatores definidas nos capítulos seguintes [Gus97].

4.4 Algoritmos Clássicos

Nesta seção pretendemos descrever de maneira bastante superficial os algoritmos clássicos para a construção da árvore de sufixos e árvore generalizada. Contudo, no próximo capítulo apresentaremos em detalhes um novo algoritmo para construção da árvore esparsa de sufixos que pode ser usado diretamente para construção da árvore de sufixos e, com poucas modificações, também para construção da árvore generalizada.

Os quatro algoritmos clássicos para a construção da árvore de sufixos de w em tempo $O(n)$, onde $n = |w|$, são: Weiner [Wei73], McCreight [McC76, dLS03], Ukkonen [Ukk95] e

[3] No caso de várias ocorrências, qualquer uma serviria.

Farach [Far97].

4.4.1 Ukkonen

O algoritmo do Ukkonen [Ukk95] é o único algoritmo linear que é também *online*, ou seja, ele processa a palavra da esquerda para a direita e a cada etapa mantém uma árvore de sufixos completa para o prefixo lido até então. A estratégia assume que o alfabeto tem tamanho constante e usa o conceito de ligações de sufixos[4], um tipo especial de aresta na árvore que liga dois vértices u e v tais que $A^{-1}\lambda(u) = \lambda(v)$, para acelerar as computações. Ukkonen baseou o seu algoritmo em um algoritmo quadrático simples para a construção da trie de sufixos [Ukk95]. Além disso, este algoritmo pode ser facilmente modificado para a construção da árvore de sufixos generalizada [Gus97].

4.4.2 McCreight

O algoritmo do McCreight [McC76], entre os algoritmos lineares clássicos, é o mais eficiente, possuindo a menor constante de tempo [GK97], independente da implementação. Este algoritmo na forma como foi apresentado em [dLS03] é a base para os algoritmos que desenvolvemos nos próximos capítulos. Além disso, assim como Ukkonen este algoritmo assume um alfabeto constante e pode ser facilmente modificado para a construção da árvore de sufixos generalizada [dLS03].

Trata-se de um algoritmo iterativo que insere na árvore um sufixos de cada vez começando com o mais comprido (a palavra toda). Diferentemente do Ukkonen, este algoritmo mantém a árvore de Ukkonen para conjunto de sufixos inseridos até então ao invés de uma árvore de sufixos completa. Assim como o algoritmo ingênuo da seção 3.5, para inserir um novo sufixo na árvore este algoritmo primeiro busca pela cabeça deste sufixo na árvore. Contudo, ele faz uso das ligações de sufixos para acelerar estas computações e atingir tempo total $O(n)$.

4.4.3 Weiner

O algoritmo do Weiner [Wei73] foi o primeiro algoritmo linear para a construção da árvore de sufixos. Este algoritmo é mais lento e usa consideravelmente mais memória do que os dois já mencionados [GK97], além de ser mais complexo. Por isso, não é um algoritmo utilizado na prática e estamos apresentando aqui por seu valor histórico.

Trata-se também de um algoritmo iterativo que em cada iteração insere um sufixo na árvore. No entanto, os sufixos são inseridos na ordem inversa ao de McCreight, ou seja, começando com

[4]Uma explicação mais detalhada sobre as ligações de sufixos é dada nos capítulos 5 e 6

o mais curto sufixo próprio não vazio (a última letra). Assim como o McCreight, este algoritmo mantém a árvore de Ukkonen para o conjunto de sufixos até então, e ao inserir um novo sufixo, busca primeiro pela sua cabeça na árvore. Uma estratégia similar às ligações de sufixos, mas percorridas na ordem inversa [Gus97, GK97], é usada para acelerar as computações.

4.4.4 Farach

O algoritmo do Farach [Far97] é o único algoritmo que constrói a árvore de sufixos em tempo linear independente do tamanho do alfabeto. Todos os outros possuem ao menos uma dependência $\Omega(\lg |A|)$ com o tamanho do alfabeto $|A|$. Mas como se assume que o alfabeto é constante a complexidade final é $O(n)$, ou seja, linear. Este algoritmo constrói a árvore de sufixos em tempo $O(n)$ para um alfabeto de números inteiros limitados por $O(n)$. A estratégia utilizada difere completamente dos outros algoritmos clássicos. Trata-se de um algoritmo de divisão e conquista, que constrói recursivamente a árvore de Ukkonen de todos os sufixos que começam em posições ímpares, a partir dela constrói a árvore de Ukkonen para as posições pares e combina as duas para obter a árvore de sufixos final. Não conhecemos nenhuma maneira de adaptar este algoritmo para construir a árvore de sufixos generalizada.

Existe uma alternativa consideravelmente menos complexa para o algoritmo do Farach que também constrói a árvore de sufixos em tempo linear independente do alfabeto. Ela se baseia na construção linear do vetor sufixos para um alfabeto de números inteiros apresentado em [KSB06]. Este algoritmo ordena lexicograficamente todos os sufixos de uma dada palavra sobre um alfabeto de inteiros e calcula os maiores prefixos comuns entre duas palavras adjacentes, tudo em tempo linear no tamanho da palavra independente do alfabeto. A partir deste vetor de sufixos é possível construir [CR03] a árvore de sufixos também em tempo linear independente do alfabeto. Os dois algoritmos envolvidos na construção são relativamente simples. Portanto, esta pode ser uma alternativa prática para a construção das árvores de sufixo para alfabetos grandes.

Capítulo 5

Árvore Esparsa de Sufixos

Uma árvore esparsa de sufixos (também denominada em alguns trabalhos como *sparse suffix tree* ou *word suffix tree*) tem a mesma estrutura que uma árvore de sufixos tradicional, mas ao invés de representar todos os sufixos de uma palavra, representa apenas um subconjunto deles. Assim como a árvore de sufixos tradicional ela é um caso particular da árvore de Ukkonen.

As árvores esparsas de sufixos são estrutura particularmente adequadas para a indexação de textos de linguagens naturais. Nestes textos, geralmente, queremos que os sufixos representados na árvore sejam somente aqueles que se iniciam em uma palavra. Por exemplo, na frase *"O rato roeu a roupa do rei de Roma."* os sufixos *"O rato roeu a roupa..."*, *"rato roeu a roupa..."* e *"roeu a roupa..."* nos interessam, mas *"ato roeu a roupa..."* e *"to roeu a roupa..."* não nos interessam. Geralmente, quando fazemos buscas em um texto em português não estamos interessados em sub-cadeias (de letras), mas sim em sub-cadeias de palavras.

As aplicações para as árvores esparsas não se restringem a indexação de textos em linguagem natural, existem outros contextos em que são relevantes somente alguns dos sufixos, por exemplo textos comprimidos com o código de Huffman [CLRS01]. Nestes textos comprimidos os símbolos originais são representados por palavras de tamanho variável, novamente os únicos sufixos que nos interessam são aqueles que quando decodificados correspondem a sufixos no texto original (não comprimido).

O trabalho de Karkkainen et al. [KU96] é a base para uma terceira aplicação das árvores esparsas. Nele é descrito um algoritmo de busca por um padrão P em um texto T, com tempo de execução, sem considerar o pré-processamento de T, ligeiramente superior a $O(l+|P|)$, onde l é o número de ocorrências de P em T. Ao invés de usar a árvore de sufixos completa para T, o algoritmo necessita apenas de uma árvore esparsa de T. Ou seja, este algoritmo faz a busca pelo padrão P em todo T, não apenas nos sufixos representados na árvore esparsa, mas usando apenas esta árvore. Desta forma, podemos reduzir o consumo de memória (armazenando somente a árvore esparsa) com um pequeno aumento no tempo de *query*, esta é uma troca que

muitas vezes pode valer a pena e pode ser a única opção quando a árvore completa não cabe na memória.

O algoritmo trivial para a construção das árvores esparsas de sufixos para um subconjunto de m sufixos de uma palavra w, com comprimento n, constrói primeiro a árvore completa para w usando algum algoritmo linear [Wei73, McC76, Ukk95, Far97], remove os sufixos que não interessam e percorre a árvore contraindo vértices que tenham apenas um filho. É fácil verificar que todos os passos deste algoritmo podem ser feitos em tempo $O(n)$. Ademais, a memória utilizada, pelo teorema 3.11, é $O(n)$, mas o tamanho da estrutura final é proporcional ao *número de sufixos* representados na árvore esparsa, mais precisamente $O(m)$. Este algoritmo usa mais memória, $O(n)$, do que o tamanho da estrutura final, $O(m)$.

Neste capítulo apresentaremos um novo algoritmo que consome tempo $O(n)$ e usa espaço proporcional ao número de sufixos representados, $O(m)$. Ou seja, trata-se de um algoritmo ótimo em tempo, pois é necessário ler toda a palavra, e espaço, pois a árvore final tem tamanho proporcional ao número de sufixos. Este algoritmo se baseia no algoritmo do Mccreight [McC76] da forma em que foi apresentado em [dLS03].

5.1 Definição

Como já foi dito uma árvore esparsa de sufixos é uma árvore de sufixos tradicional com alguns sufixos removidos, ou melhor, uma árvore de Ukkonen que reconhece um subconjunto de sufixos de uma determinada palavra. A definição formal é apresentada em seguida.

Definição 5.1 (Árvore Esparsa de Sufixos)**.** *Seja w uma palavra e W um subconjunto de* $\mathrm{Suf}(w)$*, o conjunto dos sufixos de w. A árvore esparsa de sufixos de W é a árvore de Ukkonen de W.*

Na figura 5.1, temos um exemplo de uma árvore esparsa de sufixos para a palavra *abbaabbc*, sendo que o subconjunto dos sufixos representados são todos aqueles que começam em posição ímpar, em outra palavras, $W = \{abbaabbc, baabbc, abbc, bc\}$. Assim como no exemplo, geralmente exige-se que a última letra de w não tenha aparecido antes. Desta forma, o seu conjunto de sufixos é livre de prefixos. Em nossa abordagem, assim como no capítulo 4, esta hipótese não é necessária. A única diferença é que, em nosso caso, nem todos os sufixos representados serão folhas.

Ao longo deste capítulo assumiremos que w é uma palavra sobre um alfabeto A. Seu comprimento $|w|$ é n; o subconjunto dos sufixos de w que queremos representar é $W \subseteq \mathrm{Suf}(w)$ e a cardinalidade de W é m. Sem perda de generalidade, podemos supor que $w \in W$ pois do contrário poderíamos trocar w pelo sufixo mais comprido em W. Além disso, w pode ser fatorado

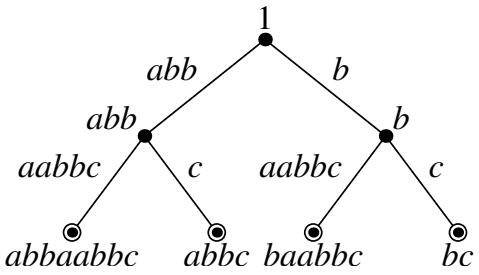


Figura 5.1: *Árvore esparsa de sufixos para o subconjunto* $W = \{abbaabbc, baabbc, abbc, bc\}$ *dos sufixos de* $abbaabbc$.

como

$$w = f_1 f_2 \dots f_m,$$

de forma que cada sufixo de W seja $f_i f_{i+1} \dots f_m$ para algum $i \in [1, m]$. *Os fatores* f_i *representam os gaps entre os sufixos de* W. Denotamos o conjunto de todos os fatores f_i por F, que também será suposto fixado ao longo do capítulo.

Por exemplo, na árvore esparsa da figura 5.1, os fatores são $F = \{ab, ba, ab, bc\}$ e os sufixos são $W = \{abbaabbc, baabbc, abbc, bc\}$.

Assumiremos ainda uma importante hipótese adicional sobre o conjunto dos fatores F:

Hipótese 5.2. *O conjunto de fatores* $F = \cup_{i=1}^{m}\{f_i\}$ *é livre de prefixos, ou seja, se* $f_i \in \text{Pref}(f_j)$, *então* $f_i = f_j$.

Como veremos mais adiante esta hipótese é necessária para que seja possível definir de maneira satisfatória as ligações de sufixo, dadas na definição 5.3. Ademais, a simples eliminação desta hipótese invalidaria o lema 5.4. É importante ressaltar que esta restrição praticamente não tira a generalidade do algoritmo descrito neste trabalho, pois podemos considerar um conjunto F arbitrário e "torná-lo" livre de prefixos, concatenando um caractere especial $\# \notin \Sigma$ a cada $f_i \in F$. Assim, $\#$ pode ser visto como um terminador de cada palavra f_i.

Para tornar a notação mais concisa denotamos os sufixos de W por

$$w\langle i, m\rangle = f_i f_{i+1} \dots f_m, \text{ para } 1 \leq i \leq m.$$

O fator $w\langle i, m\rangle$ pode ser visto como o fator de w onde as "letras" são na verdade $f_j \in F$ da fatoração de w induzida por F. É importante notar que esta notação é diferente de $w[i, m]$, que representa o fator de w dado por $w[i]w[i+1] \dots w[m]$, onde cada $w[j]$ é uma letra de A. Por exemplo, para $w = ababba$ com $F = \{a, b, ab, ba\}$, temos que $w\langle 3, 4\rangle = f_3 f_4 = abba$, mas $w[3, 4] = w[3]w[4] = ab$. Claro, se todos os fatores de F possuem somente uma letra as duas notações coincidem.

Além disso, para $i \in [0, m]$, definimos a sequência de conjuntos W_i, e a sequência de árvores esparsas de sufixos $\mathbf{T}_i$ como sendo

$$\begin{aligned} W_i &= \cup_{k=1}^{i} \{w\langle k, m\rangle\} = \begin{cases} \emptyset, & \text{se } i = 0, \\ W_{i-1} \cup \{w\langle i, m\rangle\}, & \text{se } i > 0, \end{cases} \\ \mathbf{T}_i &= \text{árvore de Ukkonen de } W_i. \end{aligned}$$

Para $i \in [1, m]$, definimos:

- a cabeça h_i, a maior palavra do conjunto $\text{Pref}(w\langle i, m\rangle) \cap \text{Pref}(W_{i-1})$;
- a cauda $t_i = h_i^{-1} w\langle i, m\rangle$;
- o lugar[1], ou vértice implícito, (u_i, x_i) de h_i em $\mathbf{T}_{i-1}$;
- e o vértice v_i, o término da aresta associada a este lugar, quando o lugar está numa aresta, $x_i \neq 1$.

De acordo com o lema 3.10, h_i e $w\langle i, m\rangle$ são os dois vértice que devem ser adicionados, se já não estiverem presentes, à árvore $\mathbf{T}_{i-1}$ para se obter $\mathbf{T}_i$.

5.2 Ligações de Sufixo

Como já foi dito, o nosso algoritmo para a construção das árvores esparsas de sufixos se baseia no algoritmo do Mccreight [McC76], que assim como outros algoritmos clássicos [Wei73, Ukk95] faz uso das *ligações de sufixo*. Nestes trabalhos, a ligação de sufixo de um vértice com rótulo x é definida como o vértice com rótulo $A^{-1}x$ (x com a primeira letra removida), ou seja, é seu maior sufixo próprio. Em nosso algoritmo, também precisaremos das ligações de sufixo, mas será necessário generalizá-las. A ligação de sufixo que usaremos aqui liga dois vértices removendo uma palavra do rótulo, não apenas uma letra. A definição formal é dada a seguir.

Definição 5.3 (Ligação de Sufixo)**.** *Seja T uma A^+-árvore, u e v dois vértices quaisquer. Se existe $f_i \in F$ tal que $\lambda(u) = f_i\lambda(v)$, então a ligação de sufixo do vértice u é o vértice v. Denotaremos isso por* $\text{Sufixo}(u) = v$.

Não está explícito na definição 5.3, mas cada vértice tem no máximo uma ligação de sufixo. Isto é consequência da hipótese de que F é livre de prefixos. Pois se existem dois vértices distintos v' e v'' tais que $\lambda(u) = f_i\lambda(v') = f_j\lambda(v'')$, devemos ter que v' e v'' tem tamanhos distintos, s.p.g. $|\lambda(v')| > |\lambda(v'')|$, desta forma f_j é prefixo próprio de f_i, contrariando a nossa

[1] O vértice u_i mais x_i, o prefixo de uma aresta que tem início em u_i. Como definido na seção 3.1.

hipótese. Ademais, é fácil ver que se a ligação de sufixo de um vértice v remove um prefixo f_i, então a ligação de sufixo de qualquer ancestral, deve remover o mesmo prefixo f_i se, obviamente, o comprimento do rótulo deste ancestral for pelo menos $|f_i|$, ou seja, se estiver definida.

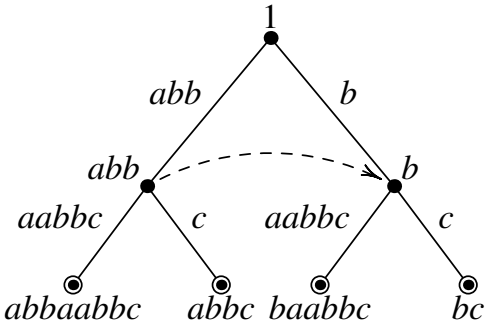


Figura 5.2: *Árvore esparsa de sufixos para o subconjunto $W = \{abbaabbc, baabbc, abbc, bc\}$ dos sufixos de abbaabbc, com a ligação de sufixo de abb representada*

Na figura 5.2 temos a árvore esparsa de sufixos para $W = \{abbaabbc, baabbc, abbc, bc\}$, ou seja, os sufixos da palavra *abbaabbc* com o conjunto de fatores $F = \{ab, ba, ab, bc\}$. A ligação de sufixo do vértice *abb* é o vértice *b* e está representada com a linha tracejada na figura. Observe que neste caso o fator removido foi *ab*.

O próximo lema é central para a linearidade do algoritmo a ser apresentado e é também a razão pela qual definimos a ligação de sufixo da maneira que fizemos. Este lema relaciona as cabeças h_{i-1} e h_i. Mais precisamente, ele estabelece que $f_{i-1}^{-1}h_{i-1}$ é prefixo de h_i. Além disso, ressaltamos que hipótese 5.2 é necessária para a demonstração e sem ela sua tese é falsa. Apresentaremos um contra-exemplo na seção 5.5.

Lema 5.4. *Suponha que $i \in [3, m]$ e que $|h_{i-1}| \geq |f_{i-1}|$. Então, a palavra $f_{i-1}^{-1}h_{i-1}$ é prefixo de h_i. Ademais, se este prefixo é próprio temos que $f_{i-1}^{-1}h_{i-1}$ é um vértice de $\mathbf{T}_{i-1}$.*

Prova. Como $|h_{i-1}| \geq |f_{i-1}|$ e f_{i-1} é prefixo de h_{i-1}, temos que $f_{i-1}^{-1}h_{i-1}$ está bem definido e $f_{i-1}^{-1}h_{i-1} \in \text{Pref}(f_{i-1}^{-1}w\langle i-1, m\rangle) = \text{Pref}(w\langle i, m\rangle)$. Sabemos ainda que existe $j \in [1, i-2]$ tal que $h_{i-1} \in \text{Pref}(w\langle j, m\rangle)$, pois $h_{i-1} \in \text{Pref}(w\langle i-1, m\rangle) \cap \text{Pref}(W_{i-2})$. Como f_{i-1} é prefixo de h_{i-1}, temos que $f_{i-1}^{-1}w\langle j, m\rangle$ está bem definido, ou seja, $f_{i-1} \in \text{Pref}(f_j)$ ou $f_j \in \text{Pref}(f_{i-1})$, mas por hipótese F é livre de prefixos, assim $f_{i-1} = f_j$. Logo, $f_{i-1}^{-1}h_{i-1} = f_j^{-1}h_{i-1} \in \text{Pref}(f_j^{-1}w\langle j, m\rangle) = \text{Pref}(w\langle j+1, m\rangle) \subseteq \text{Pref}(W_{i-1})$, pois $j \leq i-2$. Portanto, $f_{i-1}^{-1}h_{i-1} \in \text{Pref}(w\langle i, m\rangle) \cap \text{Pref}(W_{i-1})$, como h_i é a maior palavra deste conjunto, temos que $|h_i| \geq |f_{i-1}^{-1}h_{i-1}|$, ou seja, $f_{i-1}^{-1}h_{i-1}$ é prefixo de h_i.

Vamos demonstrar agora a segunda parte do lema, se $f_{i-1}^{-1}h_{i-1}$ é prefixo próprio de h_i, então existe $a \in A$ tal que $f_{i-1}^{-1}h_{i-1}a \in \text{Pref}(h_i) \subseteq \text{Pref}(W_{i-1})$. Por outro lado, $h_{i-1} \in \text{Pref}(w\langle i-1, m\rangle)$, desta forma $|h_{i-1}| \leq |w\langle i-1, m\rangle| < |w\langle j, m\rangle|$, pois $j < i-1$ e $|f_{i-1}| > 0$. Ou

seja, h_{i-1} é prefixo próprio de $w\langle j, m\rangle$, assim existe $b \in A$ tal que $h_{i-1}b \in \text{Pref}(w\langle j, m\rangle)$, por isso $f_{i-1}^{-1}h_{i-1}b \in \text{Pref}(f_{i-1}^{-1}w\langle j, m\rangle) = \text{Pref}(w\langle j+1, m\rangle)$, assim $f_{i-1}^{-1}h_{i-1}b \in \text{Pref}(W_{i-1})$. Se tivermos que $a \neq b$, então $f_{i-1}^{-1}h_{i-1}b$ é uma bifurcação W_{i-1}. Ora, sabemos que $h_{i-1} \in \text{Pref}(w\langle j, m\rangle) \subseteq \text{Pref}(W_{i-2})$, além disso $h_{i-1}a \in \text{Pref}(w\langle i-1, m\rangle)$, pois temos que $f_{i-1}^{-1}h_{i-1}a \in \text{Pref}(w\langle i, m\rangle)$ e $h_{i-1}a \in \text{Pref}(f_{i-1}w\langle i, m\rangle) = \text{Pref}(w\langle i-1, m\rangle)$. Portanto, $a \neq b$ pois h_{i-1} é a maior palavra de $\text{Pref}(w\langle i-1, m\rangle) \cap \text{Pref}(W_{i-2})$. □

Este lema garante que se $|h_{i-1}| \geq |f_{i-1}|$, então existem duas possibilidades: a ligação de sufixo de h_{i-1} já está em $\mathbf{T}_{i-1}$ ($f_{i-1}^{-1}h_{i-1}$ é prefixo próprio de h_i); ou a ligação de sufixo de h_{i-1} é h_i ($f_{i-1}^{-1}h_{i-1} = h_i$), que será inserido na iteração i. Desta forma, com uma indução simples, usando este lema como passo indutivo, temos que se $u \in \text{Bifurc}(W_{i-2})$ então a ligação de sufixo de u está definida em $\mathbf{T}_{i-1}$. Ou seja, se u é um nó interno de $\mathbf{T}_{i-2}$, então a sua ligação de sufixo está definida em $\mathbf{T}_{i-1}$. Além disso, é fácil ver que todas as folhas de $\mathbf{T}_{i-2}$ também tem as suas ligações de sufixo em $\mathbf{T}_{i-1}$. Com isso provamos o próximo corolário.

Corolário 5.5. *Seja u um vértice de $\mathbf{T}_{i-1}$ já presente em $\mathbf{T}_{i-2}$ com algum $j \in [1, m]$ tal que $f_j \in \text{Pref}(u)$, então $f_j^{-1}u \in \mathbf{T}_{i-1}$. Ou seja, a ligação de sufixo de u está bem definida em $\mathbf{T}_{i-1}$.*

5.3 Um Algoritmo Ótimo

Nesta seção apresentaremos um novo algoritmo que constrói a árvore esparsa de sufixos em tempo linear e com espaço linear no *número m de sufixos*. O pseudo-código para o algoritmo está descrito no algoritmo 5.3. Vale observar que trata-se de um algoritmo ótimo, tanto no tempo como no espaço, pois temos a cota inferior $\Omega(n)$ no tempo, uma vez que é necessário ler toda a palavra; e no espaço $\Omega(m)$, pois temos que representar todos os sufixos.

O algoritmo insere um sufixo de cada vez em ordem, começando com o mais longo, ou seja, a palavra inteira w. Ele faz uso de duas funções auxiliares SOLETRA$(l, r, \mathbf{T})$ e RESOLETRA$(l, r, \mathbf{T})$, os pseudo-códigos estão descritos nos algoritmos 5.1 e 5.2, respectivamente. Ambas partem do vértice r e descem a árvore $\mathbf{T}$ até encontrar palavra mais comprida $l' \in \text{Pref}(l)$ tal que $\lambda(r)l'$ é uma palavra de $\mathbf{T}$.

A diferença entre essas funções é que a segunda pressupõe que $l' = l$ e por isso não precisa comparar todas as letras de l, para cada aresta basta comparar a primeira letra de cada aresta. Isso é possível usando a representação dos rótulos dada na seção 4.3, o rótulo de cada aresta é representado por dois inteiros: a posição da ocorrência em w e o comprimento. Desta forma é possível em tempo constante encontrar a aresta certa e saltar até o final do seu rótulo, por isso o tempo total gasto é proporcional ao *número de arestas percorridas*. Já a primeira função compara todas as letras de l' e tem um tempo proporcional ao *comprimento de l'*. As duas

funções retornam o prefixo mais comprido de $\lambda(r)l$ contido na árvore, o seu lugar (u, x), e o vértice v, que é término da aresta associada a este lugar.

Algoritmo 5.1 Função SOLETRA

```
SOLETRA(l, r, T)
 1  ▷ r: vértice de origem, tipicamente, a raiz
 2  ▷ l: palavra a ser soletrada a partir do vértice r
 3  ▷ devolve a cabeça de λ(r)l em T, seu lugar e aresta associada
 4  u ← r
 5  s ← l
 6  enquanto |s| > 0 e ∃e = (u, v) | λ(e) = λ(u)^{-1}λ(v) ∈ Pref(s) faça
 7          s ← λ(e)^{-1}s
 8          u ← v
 9  se |s| > 0 e ∃e = (u, v) | s[1] ∈ Pref(λ(e)) então
10          x ← o mais comprido prefixo de λ(e) e de s
11          ▷ λ(u)x é prefixo próprio de λ(v)
12  senão
13          x ← 1
14          v ← u
15  devolva (λ(u)x, u, x, v)
```

Baseado no lema 5.4, na iteração i do algoritmo 5.3, ao buscar a posição para inserir o vértice h_i, o algoritmo evita processar novamente seu o prefixo $f_{i-1}^{-1}h_{i-1}$ usando, sempre que for possível, a ligação de sufixo do vértice h_{i-1} ou de seu pai. A principal diferença com o algoritmo do McCreight é neste ponto, quando a árvore contém todos os sufixos, a ligação de sufixo do pai de h_{i-1} sempre está definida, mesmo quando a de h_{i-1} não está. No caso presente, isso não é verdade, haverá casos em que nenhuma das duas ligações de sufixos estará definida (linhas 9 e 16) do algoritmo 5.3) e será necessário processar h_i inteiro, sem eliminar prefixo nenhum. Contudo, mostraremos no teorema 5.7 que isso não é o suficiente para que o algoritmo perca a linearidade.

Além disso, o algoritmo precisa atualizar as ligações de sufixo a cada iteração. Isto é feito nas linhas 21, 25, 36. Na iteração i, o algoritmo vai atualizar a ligação de sufixo do vértice h_{i-1} somente se ele tiver sido inserido na iteração anterior $i-2$ (se o *else* da linha 13 for executado), pois caso contrário a sua ligação de sufixo já foi atualizada em uma iteração anterior. Há dois casos possíveis para Sufixo(h_{i-1}):

- na linha 21, a ligação de sufixo de h_{i-1} já estava presente em $\mathbf{T}_{i-1}$ (no lema 5.4, é o caso em que $f_{i-1}^{-1}h_{i-1}$ é prefixo próprio de h_i) e é o vértice u que encontramos nesta iteração;
- na linha 25, a ligação de sufixo de h_{i-1} será inserida nesta iteração (no lema 5.4, é o caso

Algoritmo 5.2 Função ReSoletra

```
ReSoletra(l, r, T)
 1  ▷ l é soletrado a partir de r em T
 2  ▷ Restrição importante: λ(r)l é palavra de T
 3  ▷ devolve a cabeça de λ(r)l em T, seu lugar e aresta associada
 4  u ← r
 5  s ← l
 6  x ← 1
 7  enquanto |s| > 0 faça
 8        (u, v) ← aresta e | s[1] ∈ Pref(λ(e))   ▷ λ(e) = λ(u)^{-1}λ(v)
 9        se |s| ≥ |λ(e)| então
10              s ← λ(e)^{-1}s
11              u ← v
12        senão
13              x ← s
14              s ← 1
15  devolva (λ(u)x, u, x, v)
```

em que $f_{i-1}^{-1}h_{i-1}$ não é prefixo próprio de h_i, ou seja, $f_{i-1}^{-1}h_{i-1} = h_i$) e é o vértice h_i que será inserido nesta iteração.

Não é difícil transformar a argumentação dos dois parágrafos anteriores em uma demonstração formal de que o algoritmo 5.3 mantém os seguintes invariantes na linha 4:

1. $\mathbf{T}$ guarda $\mathbf{T}_{i-1}$, a árvore de Ukkonen de W_{i-1};
2. As ligações de sufixos de $\text{Bifurc}(W_{i-2}) \cup W_{i-2} \setminus \{1\}$ estão definidas;
3. (u_{i-1}, x_{i-1}) é o lugar de h_{i-1} em $\mathbf{T}_{i-2}$.

A partir destes invariantes, o teorema 5.6 segue imediatamente. Por fim, no teorema 5.7, provamos a linearidade do algoritmo 5.3.

Teorema 5.6. *O algoritmo 5.3 constrói corretamente a árvore esparsa de sufixos para o subconjunto W de sufixos de w.*

Teorema 5.7. *O algoritmo 5.3 constrói a árvore esparsa de sufixos para o subconjunto $W \subseteq \text{Suf}(w)$ em tempo $O(n)$ e espaço $O(m)$.*

Prova. Em cada iteração o algoritmo 5.3 mantém somente a árvore atual $\mathbf{T}_i$ e as suas ligações de sufixo. O número de ligações de sufixo é limitado pelo número de vértices, por isso o espaço utilizado pelo algoritmo é proporcional ao número de vértices na árvore final. O teorema 3.11

Algoritmo 5.3 Algoritmo de construção da árvore esparsa de sufixos

SparseSuffixTree(w, F)

```
 1  ▷ w: palavra para qual a árvore será construída
 2  ▷ F: conjunto dos fatores tais que w = f_1 f_2 ... f_m
 3  inicialize T com a árvore só com a raiz 1
 4  para i de 1 até m faça
 5          ▷ T = T_{i-1}
 6          se i = 1 então
 7                  (h_i, u_i, x_i, v_i) ← Soletra(w, 1, T)
 8          senão-se |h_{i-1}| < |f_{i-1}| então
 9                  (h_i, u_i, x_i, v_i) ← Soletra(f_{i-1}^{-1} h_{i-1} t_{i-1}, 1, T)
10          senão-se x_{i-1} = 1 então
11                  ▷ h_{i-1} é vértice de T_{i-2}
12                  (h_i, u_i, x_i, v_i) ← Soletra(t_{i-1}, Sufixo(h_{i-1}), T)
13          senão
14                  ▷ h_{i-1} não é vértice de T_{i-2} mas u_{i-1} = Pai(h_{i-1}) é
15                  se |u_{i-1}| < |f_{i-1}| então
16                          (h, u, x, v) ← ReSoletra(f_{i-1}^{-1} u_{i-1} x_{i-1}, 1, T)
17                  senão
18                          (h, u, x, v) ← ReSoletra(x_{i-1}, Sufixo(u_{i-1}), T)
19                  se x = 1 então
20                          (h_i, u_i, x_i, v_i) ← Soletra(t_{i-1}, u, T)
21                          Sufixo(h_{i-1}) ← u
22                  senão
23                          (h_i, u_i, x_i, v_i) ← (h, u, x, v)
24                          ▷ h_i será acrescentado nesta iteração
25                          Sufixo(h_{i-1}) ← h_i
26          t_i ← h_i^{-1} w⟨i, m⟩
27          ▷ Atualiza T de T_{i-1} para T_i
28          se |x_i| > 0 então
29                  acrescente novo vértice h_i
30                  remova a aresta (u_i, v_i)
31                  acrescente aresta de u_i para h_i de rótulo x_i
32                  acrescente aresta de h_i para v_i de rótulo h_i^{-1} v_i
33          se |t_i| > 0 então
34                  acrescente nova folha w⟨i, m⟩
35                  acrescente aresta de h_i para w⟨i, m⟩ de rótulo t_i
36                  Sufixo(w⟨i − 1, m⟩) ← w⟨i, m⟩
37  devolva T
```

garante que o número de vértice na árvore esparsa é limitado por $2m$. Portanto, o espaço utilizado pelo algoritmo 5.3 é $O(m)$

O tempo de execução do algoritmo é dominado pelas chamadas das funções SOLETRA e RESOLETRA, que por sua vez tem tempos proporcionais ao tamanho da palavra a ser soletrada e ao número de arestas percorridas ao resoletrar a palavra, respectivamente. Vamos analisar os dois casos separadamente.

Observe que em cada iteração i as chamadas da função SOLETRA são feitas na tentativa de soletrar t_{i-1} ou $f_{i-1}^{-1}h_{i-1}t_{i-1}$ (sempre que este caso ocorre, temos que $|f_{i-1}^{-1}h_{i-1}t_{i-1}| < |t_{i-1}|$), em ambos os casos o tamanho do prefixo soletrado é menor ou igual a $|t_{i-1}t_i^{-1}|$. Assim, ao final de todas as iterações, o número de letras soletradas por SOLETRA será limitado superiormente por

$$\sum_{i=1}^{m} |t_{i-1}t_i^{-1}| = |t_0| - |t_m| = n - |t_m| \leq n$$

e o tempo total gasto em SOLETRA é $O(n)$.

Por fim, vamos analisar o tempo gasto com a função RESOLETRA. Para uma dada iteração i, seja k_i o número de arestas visitadas com uma eventual chamada de RESOLETRA (linha 16, linha 18, ou nenhuma chamada) e $\phi(i)$ o tamanho[2] da palavra que resta a ser soletrada com SOLETRA ou RESOLETRA (veja que $\phi(i) \leq |x_i t_i|$).

Na iteração $i+1$ o número máximo de arestas que podemos subir na árvore é $|f_i| + 1$. Isto ocorre com a chamada da função RESOLETRA na linha 16: subimos uma aresta de h_i para o seu pai u_i; e de u_i para a raiz, neste caso temos $|u_i| < |f_i|$, ou seja, subimos no máximo $|f_i|$ arestas. No pior caso, na iteração i, pelo menos $k_i - (|f_i|+1)$ dentre as k_i arestas visitadas nesta iteração terão sido definitivamente soletradas[3].

Como o rótulo de cada aresta é não vazio, temos que pelo menos um prefixo com comprimento $k_i - (|f_i| + 1)$ foi definitivamente soletrado na iteração i, desta forma $\phi(i+1) \leq \phi(i) - (k_i - (|f_i| + 1))$, ou seja,

$$k_i \leq \phi(i) - \phi(i+1) + (|f_i| + 1).$$

Assim, o tempo total despendido com as chamadas à função RESOLETRA é proporcional a

$$\sum_{i=1}^{m} k_i \leq \phi(1) - \phi(m+1) + \sum_{i=1}^{m} |f_i| + m = 2n + m \leq 3n$$

e o tempo total gasto com RESOLETRA é $O(n)$.

[2]Para facilitar a exposição definimos $\phi(i) = 0$ para $i > m$.

[3]Ou seja, a concatenação dos rótulos destas arestas correspondem a um prefixo de $x_i t_i$ que não será mais soletrado em iterações futuras.

□

É interessante observar que quando a ligação de sufixo de um vértice ou de seu pai está definida o algoritmo se comporta exatamente como o do McCreight. Quando nenhuma delas está definida, o algoritmo se comporta com o algoritmo ingênuo da seção 3.5, que procura a posição para inserir h_i usando SOLETRA a partir da raiz. Como vimos a complexidade do algoritmo ingênuo é proporcional à soma dos comprimentos das palavras inseridas. Se aplicado a este problema seria $\sum_{i=1}^{m} w\langle i, m\rangle = O(n^2)$. A razão pela qual isso não interfere na linearidade do algoritmo, é uma consequência do lema 5.4. De acordo com este lema, as únicas cabeças que não terão as ligações de sufixo definidas são aqueles em que $|h_i| < |f_i|$ e neste caso o algoritmo vai se comportar como o ingênuo. Assim, no pior caso, o algoritmo constrói a árvore para o conjunto $F = \{f_i | 1 \leq i \leq m\}$ como o ingênuo, que tem a complexidade da soma dos tamanhos das palavras do conjunto, ou seja, $\sum_{i=1}^{m} |f_i| = n$. Portanto, o tempo total se mantém $O(n)$.

5.4 Comparação com Outros Algoritmos

Até onde sabemos, na literatura existem dois algoritmos que resolvem o problema da construção da árvore esparsa de sufixos em tempo $O(n)$ e espaço $O(m)$. Ambos fazem a mesma hipótese que fizemos aqui sobre o conjunto W, que ele seja livre de prefixos. Esta parece ser uma restrição necessária para a construção linear em espaço $O(m)$. Discutiremos isso melhor na próxima seção.

O primeiro algoritmo que resolve o problema da construção das árvores esparsas de sufixos foi apresentado por Andersson et al. [AJS99]. Este algoritmo resolve o problema em espaço $O(m)$, no pior caso, e em tempo *esperado* $O(n)$. Foi utilizada uma abordagem completamente diferente da apresentada neste trabalho. Eles reduziram o problema dos subconjuntos de sufixos para o problema de todos os sufixos transformando o conjunto dos fatores $F = \cup_{i=1}^{m}\{f_i\}$ em um alfabeto de inteiros. Este algoritmo tem valor teórico, mas não serve para aplicações práticas, pois as transformações utilizadas pelo algoritmo fazem com que ele fique bastante complexo e a constante de tempo seja alta. Para se ter uma idéia, uma das subrotinas utilizadas pelo algoritmo é o método de Harel e Tarjan [HT84, dLS03] para encontrar o menor ancestral comum em árvores em tempo constante.

Mais recentemente Inenaga et al. apresentaram em [IT06] um algoritmo *online*, baseado no trabalho do Ukkonen [Ukk95], linear em que o espaço utilizado é $O(m)$. Neste trabalho eles também fazem uma generalização das ligações de sufixo que difere da nossa, mas é possível provar equivalência entre as duas. Ao contrário do algoritmo do Andersson, este pode ser usado na prática com bom desempenho em tempo e espaço. Contudo, independentemente da imple-

mentação, sua constante de tempo deve ser maior que a nossa, pois o nosso algoritmo é baseado em McCreight que de acordo com o trabalho de Giegerich et al. [GK97] efetua intrinsecamente menos operações do que o algoritmo de Ukkonen.

Karkkainen et al. [KU96] apresentou um algoritmo para construção linear da árvore esparsa de sufixos no caso em que todos os f_i tem o mesmo tamanho, que é um caso particular da nossa hipótese de que F é livre de prefixos. O algoritmo deles se baseia em uma extensão natural da ligação de sufixo para o caso em que os f_i tem o mesmo comprimento. O tratamento que damos neste trabalho é mais geral no sentido em que, não só a nossa restrição é mais fraca, mas também nossa definição de ligação de sufixo contém a deles. Ademais, o principal lema apresentado em [KU96], é um caso particular do lema 5.4.

5.5 Problema em Aberto e a Hipótese 5.2

Em todos os trabalhos [AJS99, IT06, KU96] que tratam o problema de construção linear da árvore esparsa de sufixos com espaço $O(m)$, exige-se que o conjunto dos fatores F seja *livre de prefixos* (ou hipótese ainda mais restritiva). O problema mais geral, que não inclui esta restrição, ainda continua em aberto. Talvez ainda mais surpreendente seja que no trabalho recente de Ferragina et al. [FF07] para a construção do *vetor* esparso de sufixos, esta mesma hipótese seja necessária.

Não é difícil ver que o lema 5.4, fundamental para a corretude e linearidade do algoritmo 5.3, não é válido se F não for livre de prefixos. Tome por exemplo $F = \{ab, d, a, b, c\}$ e $w = abdabc$. Neste caso, $h_3 = ab$ e $h_4 = 1$. Desta forma $f_3^{-1}h_3 = b$, que não é prefixo de h_4. A árvore $\mathbf{T}$ para este caso é mostrada na figura 5.3. Nesta figura, podemos observar que a ligação de sufixo do vértice com rótulo ab, removendo $f_3 = a$, teria rótulo b, mas não existe vértice com este rótulo na árvore.

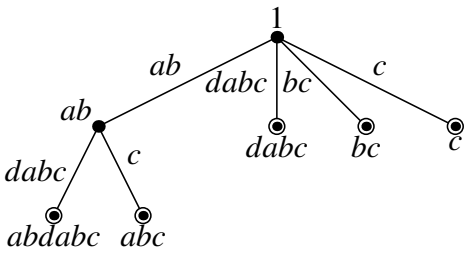


Figura 5.3: *Árvore esparsa de sufixos para $w = abdabc$ e $F = \{ab, d, a, b, c\}$.*

A hipótese 5.2 não é importante apenas para a demonstração do lema 5.4. Sem ela não é possível garantir a unicidade das ligações de sufixo. Um dado vértice poderia ter duas ligações de sufixo possíveis. Na figura 5.4, o vértice aab tem duas ligações de sufixo possíveis: removendo

$f_4 = a$, temos o vértice ab; removendo $f_6 = ab$, temos o vértice b. Poderíamos cogitar generalizar ainda mais a definição da ligação de sufixo, permitindo que um vértice possua um conjunto de ligações, mas o espaço extra necessário seria no pior caso $O(m)$ *por vértice*, o que possivelmente não permitiria que o algoritmo tivesse um consumo total de espaço $O(m)$.

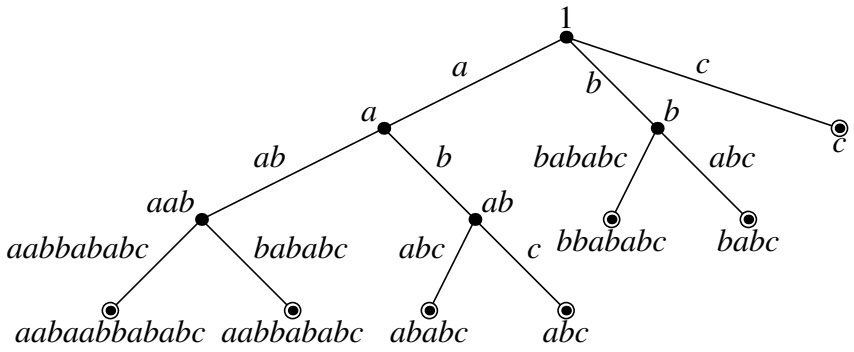


Figura 5.4: *Árvore esparsa de sufixos para* $w = aabaabbbababc$ *e* $F = \{aab, aa, bb, a, b, ab, c\}$.

Capítulo 6

Árvore de k-fatores

Os k-fatores de uma palavra w são todos os fatores de w que possuem comprimento exatamente k. A árvore de k-fatores (também denominada em outro trabalho *k-factor tree* ou *k-deep factor tree*) tem a estrutura semelhante à árvore de sufixos tradicional em que todos os vértice com rotulo de comprimento estritamente maior do que k são removidos. Da mesma forma que as árvores de sufixos e as árvores esparsas de sufixo, ela pode ser definida como um caso particular das árvores de Ukkonen.

Esta estrutura de dados pode ser usada para resolver de forma eficiente alguns problemas de análise de sequências em biologia computacional. Existem alguns algoritmos de filtragem [RSM05, BCF+99, PSdL+09] para alinhamento local que precisam indexar todos os k-fatores (q-grams) de uma dada palavra (sequência) w. A solução usada em [PSdL+09] envolve um *hash* perfeito com tempo de execução e memória proporcionais a $|A|^k + |w|$, mas uma abordagem baseada na árvores de k-fatores usaria somente $|w|$ (com uma constante maior), o que é muito interessante para alfabetos grandes (proteínas) ou fatores grandes.

Uma outra aplicação das árvores de k-fatores é o problema da busca de diversas palavras em um texto T. Uma solução usual para este problema envolve a construção da árvore de sufixos para T e soletrar cada palavra P nesta árvore. No entanto, se soubermos que os padrões buscados tem um tamanho máximo $|P|_{\max}$ podemos usar a árvore de k-fatores de T com $k = |P|_{\max}$ no lugar da árvore de sufixos, uma vez que todos os fatores de comprimento até k estão representados na árvore de k-fatores. Este é o caso da busca de *motifs* em biologia computacional [Sag98].

O algoritmo trivial para a construção das árvores de k-fatores de uma palavra w, assim como o para as árvores esparsas, constrói inicialmente a árvore de sufixos completa de w usando algum algoritmo linear [Wei73, McC76, Ukk95, Far97]. Em seguida, percorre a árvore removendo todos vértices cujo rótulo tenha comprimento maior que k e, eventualmente, adicionando novas folhas

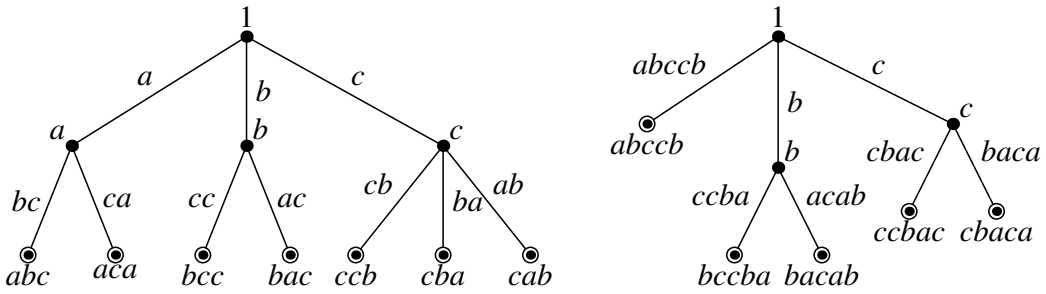
correspondentes aos fatores de comprimento k. Obtendo desta forma a árvore de k-fatores[1] de w. Não é difícil ver que este algoritmo ingênuo pode ser executado em tempo e espaço $O(n)$, onde n é o comprimento de w. No entanto, veremos neste capítulo que esta abordagem, para alfabetos e fatores pequenos, usa mais memória que o necessário.

Neste capítulo, definimos as árvores de k-fatores de uma palavra w como um caso particular das árvores de Ukkonen. Propomos também um algoritmo *online* ótimo para sua construção baseado no algoritmo do McCreight [McC76]. Este algoritmo também utiliza as das ligações de sufixo e as funções SOLETRA e RESOLETRA, definidas no capítulo 5, como subrotinas.

6.1 Definição

Definição 6.1 (Árvore de k-fatores)**.** *Seja w uma palavra e $W = \{f \in \mathrm{Fat}(w) \mid |f| = k\}$, o conjunto dos fatores de w com comprimento k. A árvore de k-fatores de w é a árvore de Ukkonen de W.*

Na figura 6.1, temos dois exemplo de árvores de k-fatores para a palavra $w = abccbacab$: um com $k = 3$; outro com $k = 5$. Na figura, observa-se que as duas árvores são consideravelmente diferentes, apesar da palavra base ser mesma nos dois casos. Isto ocorre em geral, porque a topologia da árvore de Ukkonen é determinada pelo conjunto W, e não pela palavra w. Com $k = 3$, temos $W = \{abc, bcc, ccb, cba, bac, aca, cab\}$. Com $k = 4$, temos $W = \{abccb, bccba, ccbac, cbaca, bacab\}$.



(a) $w = abccbacab$ com $k = 3$ (b) $w = abccbacab$ com $k = 5$

Figura 6.1: *Dois exemplos de árvores de k-fatores*

De maneira análoga ao capítulo 5, para $i \in [0, n-k+1]$, definimos a sequência de conjuntos W_i e a sequência de árvores de Ukkonen $\mathbf{T}_i$ como sendo:

[1] Rigorosamente falando, a árvore obtida desta forma não é a árvore de k-fatores de w. Os vértices correspondentes aos sufixos de w com comprimento estritamente menor do que k precisam ser removidos também. Isso pode ser feito com mais um percurso na árvore. Uma outra alternativa seria na construção da árvore completa não inserir os sufixos $w[i, |w|]$, com $i > |w| - k + 1$, na árvore de sufixos inicial, mas isso só é possível se o algoritmo utilizado for o McCreight (parando o algoritmo após a iteração $n - k$).

$$\begin{aligned} W_i &= \cup_{j=1}^{i}\{w[j, j+k-1]\} = \begin{cases} \emptyset, & \text{se } i = 0, \\ W_{i-1} \cup \{w[i, i+k-1]\}, & \text{se } i > 0, \end{cases} \\ \mathbf{T}_i &= \text{árvore de Ukkonen de } W_i. \end{aligned}$$

É fácil ver que todos os fatores de comprimento k de w tem a forma $w[i, i+k-1]$, com $1 \leq i \leq n-k+1$. A última árvore desta sequência, $\mathbf{T}_{n-k+1}$, é precisamente a árvore de k-fatores de w. Além disso, diferentemente de seus análogos da seção 5.1, todos os conjuntos W_i são livres de prefixo, pois todas as palavras de W_i tem o mesmo tamanho. Com efeito, o corolário 3.9 garante que todos os vértices internos de $\mathbf{T}_i$, exceto possivelmente a raiz, são bifurcações e as folhas são W_i.

As árvores de Ukkonen $\mathbf{T}_i$ dos conjuntos W_i deste capítulo possuem uma outra propriedade importante. Para $i > 0$, o conjunto W_i contém todos os fatores de comprimento k da palavra $w[1, i+k-1]$, que é prefixo de w. Desta forma, $\mathbf{T}_i$ é árvore de k-fatores de $w[1, i+k-1]$, ou seja, não é só a última árvore desta sequência que é uma árvore de k-fatores. As árvores intermediárias também o são. Isto não ocorre com as árvores da seção 5.1, onde somente a última árvore é uma árvore esparsa de sufixos. Observe ainda que a árvore de k-fatores dos prefixos $w[1, j]$, para $j < k$, é a árvore trivial. Pela definição anterior, $\mathbf{T}_0$ também é a árvore trivial.

Para $i \in [1, n-k+1]$, definimos:

- a cabeça h_i, a palavra mais comprida do conjunto $\text{Pref}(w[i, i+k-1]) \cap \text{Pref}(W_{i-1})$;
- a cauda $t_i = h_i^{-1} w[i, i+k-1]$;
- o lugar (u_i, x_i) de h_i em $\mathbf{T}_{i-1}$;
- o vértice v_i, o término da aresta associada a (u_i, x_i), quando o lugar está numa aresta, $x_i \neq 1$.

De acordo com lema 3.10, h_i e $w[i, i+k-1]$ são os dois vértice que devem ser adicionados à árvore $\mathbf{T}_{i-1}$ para se obter $\mathbf{T}_i$, se já não estiverem presentes.

6.2 Tamanho

O teorema 3.11 estabelece a melhor limitação possível (os exemplos mostram que trata-se de uma limitação justa) para o tamanho máximo de qualquer árvore de Ukkonen que reconhece n palavras. Aplicando este teorema diretamente para a árvore de k-fatores de uma palavra w, temos que o número de vértices desta árvore é limitado por $2(|w|-k+1)$, ou seja, uma limitação

em função do tamanho da palavra w e de k. Na figura 6.2, temos a árvore de k-fatores da palavra $w = ababbaabbbaaaababab$ com $k = 3$, usando o limitante anterior obtemos um número máximo de 34 vértices, mas esta árvore possui apenas 15 vértices.

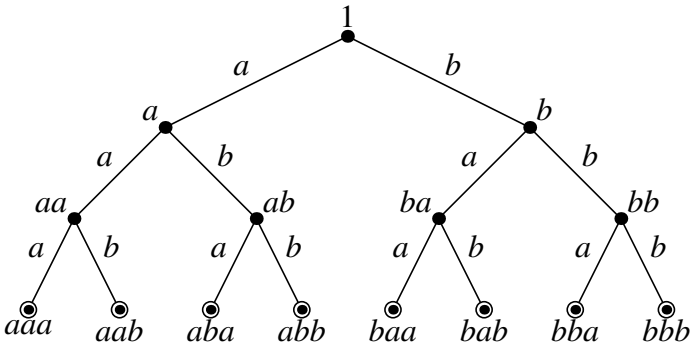


Figura 6.2: *Árvore de k-fatores de $w = ababbaabbbaaaababab$ para $k = 3$.*

Há algo de especial neste exemplo, todas as palavras possíveis de tamanho 3 em $A^* = \{a, b\}^*$ são vértices desta árvore. Ou seja, esta é a maior árvore de k-fatores possível para $k = 3$ e $A = \{a, b\}$. Seguindo este raciocínio, seria possível obter uma limitação mais precisa para o tamanho da árvore de k-fatores se considerássemos o número de palavras distintas com comprimento k em A^*, ou seja, a cardinalidade de A^k. Desta forma obteríamos o número máximo de folhas possíveis para árvore de k-fatores de $w \in A^*$ e a partir daí uma limitação no número total de vértices. Na proposição 6.2 fazemos isto de uma forma ligeiramente diferente, mas que resulta em um limitante um pouco mais preciso.

Proposição 6.2. *Seja $w \in A^*$ uma palavra de comprimento n e $\mathbf{T} = (V, E, \lambda)$ a árvore de k-fatores de w. Então, se $\alpha = \min\{\frac{|A|^{k+1}-1}{|A|-1}, 2(n-k+1)\}$ temos que $|V| \leq \alpha$ e $|E| \leq \alpha - 1$.*

Prova. O número de fatores de tamanho k em uma palavra de comprimento n é $n - k + 1$, ou seja, o número de palavra reconhecidas por $\mathbf{T}$ é $|W| = n - k + 1$. Desta forma, o teorema 3.11 garante que

$$|V| \leq 2(n - k + 1).$$

Por outro lado, a profundidade de cada vértice é limitada pelo comprimento de seu rótulo (os rótulos são não vazios), logo a altura máxima de $\mathbf{T}$ é k. Além disso, cada vértice tem no máximo $|A|$ filhos. Então, somando nível a nível a partir da raiz, temos que

$$|V| \leq \sum_{j=0}^{j=k} |A|^j = \frac{|A|^{k+1} - 1}{|A| - 1}.$$

Tomando o menor entre os dois limitantes superiores obtemos o resultado. □

6.3 Ligações de Sufixo

Assim como na construção da árvore esparsa, na construção da árvore de k-fatores em tempo linear precisaremos das ligações de sufixo. No entanto, neste capítulo as ligações de sufixo de um vértice v não removem mais uma palavra $w \in A^+$ do seu rótulo, mas apenas uma letra de A.

Definição 6.3 (Ligação de Sufixo)**.** *Seja T uma A^+-árvore e u, v dois vértices quaisquer. Se existe $a \in A$ tal que $\lambda(u) = a\lambda(v)$, então a ligação de sufixo do vértice u é o vértice v. Denotaremos isso por* $\text{Sufixo}(u) = v$.

Esta definição de ligação de sufixo é a usualmente encontrada na literatura [McC76, Ukk95, dLS03, Gus97, CHL07] e é um caso particular da definição 5.3, basta tomar F como um subconjunto do alfabeto A, que é claramente livre de prefixos. Na figura 6.3, temos um exemplo de uma ligação de sufixo do vértice ab na árvore de k-fatores de $w = babbaaaa$ com $k = 3$.

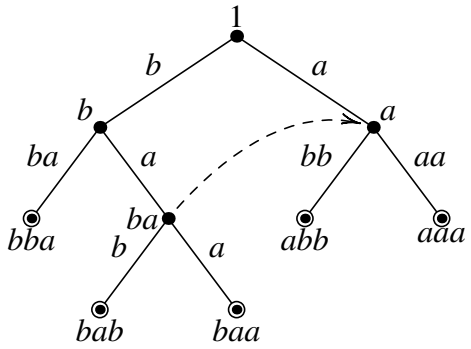


Figura 6.3: *Árvore de k-fatores de $w = babbaaaa$ para $k = 3$, com a ligação de sufixo de ab representada.*

Assim como o lema 5.4, o próximo lema é central para a corretude e linearidade do algoritmo. Ele relaciona duas cabeças consecutivas h_{i-1} e h_i. Este lema será necessário para justificar o uso da função ReSoletra e das ligações de sufixo.

Lema 6.4. *Suponha que $i \in [3, n-k+1]$. Então, a palavra $A^{-1}h_{i-1}$ é prefixo de h_i. Ademais, se este prefixo é próprio e $|h_{i-1}| < k$, temos que $A^{-1}h_{i-1}$ é um vértice de $\mathbf{T}_{i-1}$.*

Prova. Sabemos que $h_{i-1} \in \text{Pref}(w[i-1, i+k-2]) \cap \text{Pref}(W_{i-2})$, ou seja, existe um $j \in [1, i-2]$ tal que $h_{i-1} \in \text{Pref}(w[i-1, i+k-2]) \cap \text{Pref}(w[j, j+k-1])$. Desta forma, temos que $A^{-1}h_{i-1} \in \text{Pref}(w[i, i+k-2]) \cap \text{Pref}(w[j+1, j+k-1])$. É claro que $w[i, i+k-2]$ e $w[j+1, j+k-1]$ são prefixos de $w[i, i+k-1]$ e $w[j+1, j+k]$, respectivamente. Como $j+1 \leq i-1$, temos que $A^{-1}h_{i-1} \in \text{Pref}(w[i, i+k-1]) \cap \text{Pref}(W_{i-1})$. Por fim, como h_i é a maior palavra deste conjunto, $A^{-1}h_{i-1}$ é prefixo de h_i.

Vamos demonstrar a segunda parte do lema, se $A^{-1}h_{i-1}$ é prefixo próprio de h_i, então existe $a \in A$ tal que $A^{-1}h_{i-1}a \in \text{Pref}(h_i) \subseteq \text{Pref}(W_{i-1})$. Por outro lado, $h_{i-1} \in \text{Pref}(w[j, j+k-1])$ e como $|h_{i-1}| < k$, temos que este prefixo é próprio. Ou seja, existe um $b \in A$ tal que $h_{i-1}b \in \text{Pref}(w[j, j+k-1]) \subseteq \text{Pref}(W_{i-2})$. Como $j+1 \leq i-1$, temos que $A^{-1}h_{i-1}b \in \text{Pref}(w[j+1, j+k-1]) \subseteq \text{Pref}(w[j+1, j+k]) \subseteq \text{Pref}(W_{i-1})$. Ou seja, se $a \neq b$, então $A^{-1}h_{i-1}$ é uma bifurcação de W_{i-1}. Ora, sabemos que $A^{-1}h_{i-1}a \in \text{Pref}(h_i) \subseteq \text{Pref}(w[i, i+k-1])$, por isso $h_{i-1}a \in \text{Pref}(w[i-1, i+k-1])$, mas como $|h_{i-1}a| \leq k$, temos também que $h_{i-1}a \in \text{Pref}(w[i-1, i+k-2])$. Portanto, $h_{i-1}a \in \text{Pref}(w[i-1, i+k-2])$ e $h_{i-1}b \in \text{Pref}(W_{i-2})$, como h_{i-1} é a maior palavra na intersecção destes conjuntos, temos que $a \neq b$. □

Este lema garante que se $|h_{i-1}| < k$ (ou seja, $h_{i-1} \neq w[i-1, i+k-2]$), então existem duas possibilidades: a ligação de sufixo de h_{i-1} está presente em T_{i-1} ($A^{-1}h_{i-1}$ é um prefixo próprio); ou a ligação de sufixo de h_{i-1} é h_i ($A^{-1}h_{i-1} = h_i$) que será inserido no final na iteração i. Usando este lema como o passo indutivo obtemos o próximo corolário.

Corolário 6.5. *Seja $u \in \text{Bifurc}(W_{i-1})$ já presente em $\mathbf{T}_{i-2}$, então $A^{-1}u \in \mathbf{T}_{i-1}$. Ou seja, a ligação de sufixo de u está presente $\mathbf{T}_{i-1}$.*

Repare que nada foi dito a respeito das ligações de sufixo dos vértices em W_i. Isto ocorre porque em geral a ligação de sufixo destes vértices não estará definida. No entanto, estas ligações de sufixos não serão necessárias para o algoritmo, as únicas ligações de sufixos necessárias serão as dos nós internos, que neste caso, como W_i é livre de prefixos (corolário 3.9), são exatamente as bifurcações e a raiz.

Vale a pena observar que a hipótese que $|h_{i-1}| < k$ é realmente necessária para a segunda parte do lema 6.4. Como um contra-exemplo tome a palavra $aaaaaaa$ e $k = 3$, é fácil ver que qualquer árvore intermediária T_i ($i \geq 1$) tem apenas a raiz e um vértice de rótulo aaa. Além disso, $h_2 = aaa$ e $h_3 = aaa$, logo $A^{-1}h_2 = aa$ é prefixo próprio de h_3, mas claramente aa não é vértice de T_2. Veja ainda que a hipótese $|h_{i-1}| < k$ só é usada na segunda parte do lema, se ela fosse usada também na primeira parte do lema 6.4 o algoritmo 6.1 não seria linear[2].

6.4 Um Algoritmo Ótimo

Assim como o algoritmo para construção das árvores esparsas de sufixos, este algoritmo se baseia no trabalho de McCreight [McC76]. Trata-se de um algoritmo incremental que insere um k-fator em cada iteração, começando com o mais a "esquerda", $w[1, k]$. Sempre que possível ele usa as ligações de sufixos para acelerar as computações, evitando processar o prefixo $A^{-1}h_{i-1}$ de h_i.

[2]Não seria possível usar a função ReSoletra, que é essencial para a linearidade do algoritmo.

Neste algoritmo, também são usadas as funções SOLETRA$(l, r, \mathbf{T})$ e RESOLETRA$(l, r, \mathbf{T})$ descritas no capítulo 5. A primeira função desce a árvore $\mathbf{T}$ a partir do vértice r procurando o maior l', prefixo de l, tal que $\lambda(r)l'$ é uma palavra de $\mathbf{T}$. A segunda função faz o mesmo, mas assumindo a hipótese extra que $l = l'$, ou seja, $\lambda(r)l$ é uma palavra de $\mathbf{T}$. Desta forma ela tem consumo de tempo proporcional ao *número de arestas visitadas*, enquanto que a outra tem consumo de tempo proporcional ao tamanho do prefixo l'. Sempre que possível o algoritmo irá usar a função RESOLETRA.

Diferentemente do algoritmo 5.3 de construção das árvores esparsas, o algoritmo 6.1 é *online*. Ou seja, o algoritmo mantém a árvore de k-fatores completa para o prefixo de w lido até então. Como foi visto na seção 6.1, a árvore intermediaria $\mathbf{T}_i$ é exatamente a árvore de k-fatores para o prefixo $w[1, i+k-1]$ processado até então. Ademais, se este prefixo tem comprimento menor que k, a sua árvore de k-fatores é a árvore trivial, que é igual a $\mathbf{T}_0$.

Por fim, no teorema 6.6 demonstramos a corretude do algoritmo 6.1 e, no teorema 6.7, demonstramos a sua linearidade.

Teorema 6.6. *O algoritmo 6.1 constrói corretamente a árvore de k-fatores de w.*

Prova. No fim da primeira iteração temos que $\mathbf{T}_1$ é a árvore de Ukkonen de W_1, $h_1 = 1, u_1 = 1$ e $x_1 = 1$. Na iteração seguinte o algoritmo 6.1 chama a função SOLETRA (linha 9) para encontrar o lugar (u_2, x_2) de h_2 em $\mathbf{T}_1$ e o utiliza para, possivelmente, adicionar os vértices h_2 e $w[2, 2+k-1]$ em $\mathbf{T}_1$ obtendo desta forma $\mathbf{T}_2$. Observe que $\text{Bifurc}(W_1) = \emptyset$. Agora, vamos provar que o algoritmo 6.1, durante as próximas iterações ($i \in [3, n-k+1]$), mantém os seguintes invariantes na linha 7:

1. $\mathbf{T}$ guarda $\mathbf{T}_{i-1}$, a árvore de Ukkonen de W_{i-1};
2. As ligações de sufixos de $\text{Bifurc}(W_{i-2}) \setminus \{1\}$ estão definidas;
3. (u_{i-1}, x_{i-1}) é o lugar de h_{i-1} em $\mathbf{T}_{i-2}$.

Suponha que $x_{i-1} = 1$ e $|h_{i-1}| < k$ (linha 10). Desta forma h_{i-1} é um vértice interno de $\mathbf{T}_{i-2}$. Se h_{i-1} não é a raiz, a sua ligação de sufixo está definida (invariante 2) e podemos usá-la para encontrar $A^{-1}h_{i-1}$, que é um prefixo de h_i de acordo com o lema 6.4. A partir daí SOLETRA é usada no sufixo restante de h_i (linha 15) para encontrar (u_i, x_i) o lugar de h_i em $\mathbf{T}_{i-1}$. Se $h_{i-1} = 1$, SOLETRA é usada com as modificações necessárias (linha 13), mas com o mesmo propósito.

Por outro lado (linha 16), temos duas possibilidades:

- se $|h_{i-1}| = k$, então h_{i-1} é uma folha de $\mathbf{T}_{i-2}$ e $\text{Pai}(h_{i-1})$ está definido em $\mathbf{T}_{i-1}$;

Algoritmo 6.1 Algoritmo de construção da árvore de k-fatores

kFactorTree(w, k)

```
1  ▷ w: palavra que contém os fatores da árvore
2  ▷ k: tamanho dos fatores
3  inicialize T como a árvore com raiz 1 (T_0)
4  se n < k então
5          devolva T
6  para i de 1 até n − k + 1 faça
7          ▷ T = T_{i−1}
8          se i ≤ 2 então
9                  (h_i, u_i, x_i, v_i) ← Soletra(w[i, k], 1, T)
10         senão-se x_{i−1} = 1 and |h_{i−1}| < k então
11                 ▷ h_{i−1} é um vértice interno de T_{i−2}
12                 se h_{i−1} = 1 então
13                         (h_i, u_i, x_i, v_i) ← Soletra(A^{−1} t_{i−1} w[i + k − 1], 1, T)
14                 senão
15                         (h_i, u_i, x_i, v_i) ← Soletra(t_{i−1} w[i + k − 1], Sufixo(h_{i−1}), T)
16         senão
17                 ▷ h_{i−1} não é um vértice interno de T_{i−2} mas Pai(h_{i−1}) é
18                 y ← rótulo da aresta (Pai(h_{i−1}), h_{i−1})
19                 se Pai(h_{i−1}) = 1 então
20                         (h, u, x, v) ← ReSoletra(A^{−1} y, 1, T)
21                 senão
22                         (h, u, x, v) ← ReSoletra(y, Sufixo(Pai(h_{i−1})), T)
23                 se |h_{i−1}| < k então
24                         se x = 1 então
25                                 (h_i, u_i, x_i, v_i) ← Soletra(t_{i−1} w[i + k − 1], u, T)
26                                 Sufixo(h_{i−1}) ← u
27                         senão
28                                 (h_i, u_i, x_i, v_i) ← (h, u, x, v)
29                                 Sufixo(h_{i−1}) ← novo vértice h_i
30                 senão
31                         se w[i, i + k − 1] é uma folha de T então
32                                 (h_i, u_i, x_i, v_i) ← (w[i, i + k − 1], w[i, i + k − 1], 1, w[i, i + k − 1])
33                         senão
34                                 (h_i, u_i, x_i, v_i) ← (h, u, x, v)
35         t_i ← h_i^{−1} w[i, i + k − 1]
36         ▷ Atualiza T de T_{i−1} para T_i
37         se |x_i| > 0 então
38                 acrescente novo vértice h_i
39                 remova a aresta (u_i, v_i)
40                 acrescente aresta de u_i para h_i de rótulo x_i
41                 acrescente aresta de h_i para v_i de rótulo h_i^{−1} v_i
42         se |t_i| > 0 então
43                 acrescente nova folha w[i, i + k − 1]
44                 acrescente aresta de h_i para w[i, i + k − 1] de rótulo t_i
45 devolva T
```

- se $x_{i-1} \neq 1$, então h_{i-1} não é um vértice de $\mathbf{T}_{i-2}$ e, como $h_{i-1} \neq 1$, Pai(h_{i-1}) está definido em $\mathbf{T}_{i-1}$.

Em ambos os casos, Pai(h_{i-1}) está definido e é um vértice interno de $\mathbf{T}_{i-1}$. Se não for a raiz, sua ligação de sufixo também está definida (invariante 2). Além disso, nos dois casos, a primeira parte do lema 6.4 é válida e garante que $A^{-1}h_{i-1}$ é um prefixo de h_i. Desta forma, ReSoletra é usada a partir de Sufixo(Pai(h_{i-1})) (linha 22) para encontrar (u, x) o lugar de $A^{-1}h_{i-1}$ em $\mathbf{T}_{i-1}$. Se Pai$(h_{i-1}) = 1$, ReSoletra é usada com as modificações necessárias (linha 20), mas com o mesmo propósito.

Na sequência (linha 23), se $|h_{i-1}| < k$, a segunda parte do lema 6.4 é válida. Ela garante que se $A^{-1}h_{i-1}$ é um vértice de $\mathbf{T}_{i-1}$, então $A^{-1}h_{i-1} = u$ é um prefixo próprio de h_i. Por isso (linha 24), se $x = 1$, então u é vértice de $\mathbf{T}_{i-1}$; Soletra é usado com o sufixo restante para encontrar o lugar (u_i, x_i) de h_i em T_{i-1} (linha 25); e a ligação de sufixo de h_{i-1} é u. Caso contrário (linha 27), $A^{-1}h_{i-1} = h_i$, o lugar de h_i em $\mathbf{T}_{i-1}$ é $(u_i, x_i) = (u, x)$ (linha 28) e a ligação de sufixo de h_{i-1} é h_i. Observe que h_{i-1} é uma bifurcação de W_{i-1} somente se $|h_{i-1}| < k$, e este é único vértice de Bifurc(W_{i-1}) para o qual a ligação de sufixo, possivelmente, ainda não está definida.

Por outro lado (linha 30), se $|h_{i-1}| = k$, h_{i-1} é uma folha e sua ligação de sufixo não precisa ser definida. Temos agora duas possibilidades para h_i: $w[i, i+k-1]$ é uma folha de $\mathbf{T}_{i-1}$, $h_i = w[i, i+k-1]$ e o seu lugar (u_i, x_i) em $\mathbf{T}_{i-1}$ é $(w[i, i+k-1], 1)$; ou $h_i = A^{-1}h_{i-1}$ e seu lugar (u_i, x_i) em $\mathbf{T}_{i-1}$ é (u, x).

Por fim (linha 36), o lugar (h_i, x_i) de h_i em $\mathbf{T}_{i-1}$ já foi encontrado. Então o algoritmo usa esta informação para inserir os vértices, caso ainda não estejam presentes, h_i e $w[i, i+k-1]$ em $\mathbf{T}_{i-1}$. Obtendo, desta forma, a árvore $\mathbf{T}_i$. □

Teorema 6.7. *O algoritmo 6.1 constrói a árvore de k-fatores de w em tempo $O(n)$ e espaço $O(\min\{\frac{|A|^{k+1}-1}{|A|-1}, 2(n-k+1)\})$.*

Prova. Em cada iteração, o algoritmo 6.1 mantém somente a árvore atual $\mathbf{T}_i$ e a suas ligações de sufixo. O número de ligações de sufixo é limitado pelo número de vértices. Por isso o espaço utilizado pelo algoritmo é proporcional ao número de vértices na árvore final. A proposição 6.2 garante que o número de vértices na árvore final é limitado por $\min\{\frac{|A|^{k+1}-1}{|A|-1}, 2(n-k+1)\}$.

Assim como no algoritmo 5.3, o tempo de execução do algoritmo 6.1 é dominado pelas chamadas das funções Soletra e ReSoletra, que por sua vez tem tempos proporcionais ao tamanho da palavra a ser soletrada e ao número de arestas percorridas ao resoletrar a palavra, respectivamente. Vamos analisar os dois casos separadamente.

Observe que em cada iteração i as chamadas da função Soletra são feitas na tentativa de soletrar $t_{i-1}w[i+k-1]$. Assim, ao final de todas as iterações, o número de letras soletradas por

SOLETRA será limitado superiormente por

$$\sum_{i=1}^{n-k+1} (|t_{i-1}t_i^{-1}| + 1) = |t_0| - |t_{n-k+1}| + n - k = k - |t_{n-k+1}| + n - k \leq n,$$

e o tempo total gasto em SOLETRA é $O(n)$.

Por fim, vamos analisar o tempo gasto com a função RESOLETRA. Para uma dada iteração i, seja k_i o número de arestas visitadas com uma eventual chamada de RESOLETRA (linha 20 ou 22 ou nenhuma chamada); seja[3] $\phi(i)$ o tamanho da palavra que resta a ser soletrada com SOLETRA ou RESOLETRA (note que $\phi(i) \leq |x_i t_i|$).

Na iteração $i + 1$, podemos subir no máximo uma aresta na árvore. Isto ocorre com a chamada da função RESOLETRA na linha 20: subimos uma aresta de h_i para o seu pai u_i. No pior caso, na iteração i, pelo menos $k_i - 1$ dentre as k_i arestas visitadas nesta iteração terão sido definitivamente soletradas[4].

Como o rótulo de cada aresta é não vazio, temos que pelo menos um prefixo com comprimento $k_i - 1$ foi definitivamente soletrado na iteração i. Desta forma, $\phi(i + 1) \leq \phi(i) - (k_i - 1))$. De forma que

$$k_i \leq \phi(i) - \phi(i + 1) + 1.$$

Assim, o tempo total despendido com as chamadas à função RESOLETRA é proporcional a

$$\sum_{i=1}^{n-k+1} k_i \leq \phi(1) - \phi(n - k + 1) + n - k + 1 = 2n - k + 1$$

e o tempo total gasto com RESOLETRA é $O(n)$. □

6.5 Comparação com Outros Algoritmos

Até onde sabemos existe apenas um trabalho na literatura que também trata do problema da construção da árvore de k-fatores. Trata-se do trabalho de Allali et al. [AS04], em que os autores propõe um algoritmo *online* baseado no trabalho do Ukkonen [Ukk95] para a construção das árvores de k-fatores.

O algoritmo proposto por Allali et al. não é apenas um resultado teórico, ele possui um bom desempenho na prática. Os autores realizaram diversos testes comparando o tempo de

[3] Para facilitar a exposição definimos $\phi(i) = 0$ para $i > n - k + 1$.

[4] Ou seja, a concatenação dos rótulos destas arestas correspondem a um prefixo de $x_i t_i$ que não será mais soletrado em iterações futuras.

execução e a memória usada pelo algoritmo proposto com o algoritmo de Ukkonen tradicional aplicados ao problema de busca de padrões com tamanho limitado. O algoritmo proposto obteve resultados melhores em todos os testes, com ganhos particularmente expressivos nos testes em que o alfabeto era pequeno. Este algoritmo possui as mesmas complexidades assintóticas de tempo e espaço que o nosso, apresentadas no teorema 6.7. No entanto, ele usa uma estrutura de controle adicional: uma fila de folhas. Como visto no capítulo anterior, baseado no trabalho de [GK97], que mostra que a abordagem de McCreight é a mais rápida que a de Ukkonen por realizar menos operações a cada iteração, é esperado que o algoritmo 6.1 aqui proposto tenha uma constante melhor independente da implementação.

Capítulo 7

Conclusão

Na primeira parte deste trabalho, a caracterização combinatória no capítulo 3, retomamos e estendemos o estudo das propriedades combinatórias das árvores de Ukkonen, primeiramente estudas em [dLS03]. A minimalidade da árvore de Ukkonen demonstrada no teorema 3.14 é resultado original desta dissertação. O enfoque nesta parte é mais matemático, consistindo em enunciados e demonstrações de diversas propriedades das árvores de Ukkonen, que servem de base para todo o resto do trabalho. Tais propriedades, em geral, procuram relacionar propriedades combinatórias das palavras com propriedades estruturais da árvore. Acreditamos que a caracterização oferecida pela definição 3.7 oferece uma descrição explícita dos objetos por ela descritos, como por exemplo as árvores de sufixo, que normalmente não se encontram em outros trabalhos onde, quase sempre, a estrutura é caracterizada como fruto da execução de um algoritmo.

O principal resultado original da primeira parte é o enunciado do teorema 3.14 sobre a minimalidade das árvores de Ukkonen, cuja demonstração faz uso de várias propriedades provadas anteriormente. Além disso, no capítulo 4 as árvores de sufixos são definidas como casos particulares das árvores de Ukkonen, ou seja, este teorema também se aplica para elas e justifica a noção intuitiva de que as árvores de sufixos são as menores A^+-árvores que representam todos os sufixos de uma determinada palavra.

Na segunda parte, nos capítulos 5 e 6, o enfoque é mais algoritmo. Nela, são estudadas duas estruturas de dados: árvore esparsa de sufixos e árvore de k-fatores. Elas também são definidas como casos particulares das árvores de Ukkonen, herdando a minimalidade e todas as outras propriedades já provadas. Além disso, propomos novos algoritmos ótimos, tanto em tempo quanto em espaço, para a construção destas árvores.

Nossos algoritmos não são apenas propostas de valor teóricos, eles podem ser aplicados na prática, sua implementação não é muito complexa e devem ser eficientes mesmo para instâncias pequenas. Apesar de possuírem as mesmas complexidades que alguns dos melhores algoritmos

presentes na literatura, o número operações que eles efetuam é intrinsecamente menor [GK97]. Contudo, não foram realizados testes experimentais para confirmar estas predições.

Capítulo 8

Trabalhos Futuros

Neste capítulo, serão discutidas algumas possibilidades para a continuidade deste trabalho. Temos quatro linhas possíveis, três delas mais aplicadas e uma mais teórica, com graus de dificuldades variados. Possivelmente, a mais difícil é a descrita na seção 8.4.

8.1 Aplicação da Árvore Esparsa de Sufixos à Detecção de Fraudes

Maranzato et al. [MNPdL09, MNPdL10] apresentaram nestes dois trabalho técnicas para a detecção de fraudes em *e-markets* (i.e. Mercado Livre, eBay e Toda Oferta). Esta técnicas se baseiam principalmente em *logs* de transações e não levam em conta o conteúdo dos anúncios, as páginas HTML dos produtos vendidos.

Utilizando os métodos descritos neste trabalho, em particular o algoritmo de construção da árvore esparsa de sufixos, seria possível indexar todas a base de anúncios e levantar estatísticas, por exemplo fatores mais comuns, e com isso tentar aumentar a eficiência do método de detecção de fraudes.

8.2 Comparação com o Vetor de Sufixos Esparso

Em trabalho recente [FF07] Ferragina et al. estudaram o problema de construção do vetor de sufixos esparso e propuseram um algoritmo com tempo $O(n)$ e espaço $O(m)$. Neste trabalho, a hipótese 5.2 sobre o conjunto de fatores também é necessária. Seria interessante fazer um estudo aprofundado deste trabalho para tentar encontrar relações entre este algoritmo e o que nós desenvolvemos, o que poderia ajudar a responder as questões da seção 8.4. Um estudo experimental comparativo entre este algoritmo e o algoritmo 5.3 seria importante para determinar

qual é a melhor solução prática.

8.3 Aplicação da Árvore Esparsa de Sufixos a *Document Clustering*

O problema que se quer resolver em *document clustering* é o seguinte: dada uma coleção de documentos queremos agrupá-los em *clusters* de forma que os documentos de um mesmo *cluster* sejam similares entre si e não similares aos demais.

O STC (*suffix tree clustering*) introduzido por Zamir et al. em [ZE98] é um algoritmo clássico que resolve este problema. Nele, cada documento da coleção é representado por uma árvores de sufixos. Acreditamos que o uso de árvores esparsas de sufixos pode melhorar o uso de memória. Ademais, acreditamos ser possível melhorar a qualidade dos *clusters* produzidos.

8.4 Construção da Árvore Esparsa de Sufixos sem a Hipótese 5.2

Na seção 5.5 nós discutimos a importância da hipótese 5.2 para as ligações de sufixo e corretude do algoritmo, mas *não demos nenhuma demonstração da impossibilidade de construir a árvore esparsa de sufixos em tempo linear e espaço* $O(m)$. Além disso, todos os trabalhos que encontramos na literatura [AJS99, IT06, KU96, FF07] impõem esta hipótese no conjunto de fatores. Seria interessante um estudo mais aprofundado do problema geral de construção e a necessidade do conjunto de fatores ser livre de prefixo, para dar uma resposta definitiva para este problema, seja afirmativa na forma de um algoritmo ou uma demonstração da impossibilidade da construção sem a hipótese 5.2.

Referências Bibliográficas

[AG85] Alberto Apostolico e Zvi Galil, editors. *Combinatorial Algorithms on Words*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1985. 1

[AJS99] Arne Andersson, N. Jesper, e Larsson Kurt Swanson. Suffix trees on words. *Algorithmica*, 23:102–115, 1999. 2, 3, 39, 40, 58

[AS04] Julien Allali e Marie-France Sagot. The at most k-deep factor tree. Relatório Técnico 2004-03, Institut Gaspard Monge, Université de Marne la Vallée, 2004. 2, 52

[BCF+99] Stefan Burkhardt, Andreas Cramer, Paolo Ferragina, Hans peter Lenhof, Eric Rivals, e Martin Vingron. q-gram based database searching using a suffix array (QUASAR). Em *Proceedings of the third annual international conference on Computational molecular biology (Recomb 99)*, 1999. 43

[CHL07] Maxime Crochemore, Christophe Hancart, e Thierry Lecroq. *Algorithms on Strings*. Cambridge University Press, New York, NY, USA, 2007. 23, 25, 47

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, e Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001. 1, 29

[Col91] Richard Cole. Tight bounds on the complexity of the boyer-moore string matching algorithm. Em *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*, SODA '91, páginas 224–233, Philadelphia, PA, USA, 1991. Society for Industrial and Applied Mathematics. 1

[CR94] Maxime Crochemore e Wojciech Rytter. *Text algorithms*. Oxford University Press, Inc., New York, NY, USA, 1994.

[CR03] Maxime Crochemore e Wojciech Rytter. *Jewels of Stringology*. World Scientific Publishing, 2003. 28

[dLS03] Alair Pereira do Lago e Imre Simon. *Tópicos em Algorítmos sobre Sequências*. IMPA, 2003. 2, 14, 23, 24, 26, 27, 30, 39, 47, 55

[Far97] Martin Farach. Optimal suffix tree construction with large alphabets. Em *FOCS '97: Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, página 137, Washington, DC, USA, 1997. IEEE Computer Society. 2, 23, 27, 28, 30, 43

[FF07] Paolo Ferragina e Johannes Fischer. Suffix arrays on words. Em *In Proceedings of the 18th Annual Symposium on Combinatorial Pattern Matching, volume 4580 of LNCS*, páginas 328–339. Springer, 2007. 40, 57, 58

[GK97] Robert Giegerich e Stefan Kurtz. From Ukkonen to Mccreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997. 2, 10, 23, 27, 28, 40, 53, 56

[Gus97] Dan Gusfield. *Algorithms on strings, trees, and sequences*. Cambridge University Press, Cambridge, 1997. Computer science and computational biology. 2, 23, 24, 25, 26, 27, 28, 47

[HT84] Dov Harel e Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. 39

[IT06] Shunsuke Inenaga e Masayuki Takeda. On-line linear-time construction of word suffix trees. Em *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, páginas 60–71, 2006. 3, 39, 40, 58

[KSB06] Juha Kärkkäinen, Peter Sanders, e Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, November 2006. 28

[KU96] Juha Kärkkäinen e Esko Ukkonen. Sparse suffix trees. Em *Computing and Combinatorics, Second Annual International Conference, COCOON '96, Hong Kong, June 17-19, 1996, Proceedings*, páginas 219–230, 1996. 3, 29, 40, 58

[McC76] Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, 1976. 2, 24, 25, 26, 27, 30, 32, 43, 44, 47, 48

[MM90] Udi Manber e Gene Myers. Suffix arrays: A new method for on-line string searches. Em *SODA*, páginas 319–327, 1990.

[MNPdL09] Rafael Maranzato, Marden Neubert, Adriano Pereira, e Alair Pereira do Lago. Feature extraction for fraud detection in electronic marketplaces. Em *LA-WEB 2009: 7th Latin American Web Congress*, Mérida, México, 2009. IEEE Computer Society. 57

[MNPdL10] Rafael Maranzato, Marden Neubert, Adriano Pereira, e Alair Pereira do Lago. Feature detection in reputation systems in e-markets using logistic regression. Em *ACM-SAC 2010: 25th Symposium On Applied Computing*, 2010. 57

[Mor68] Donald R. Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15(4):514–534, 1968. 23

[Pra73] V.R. Pratt. Improvements and applications of the Weiner repetition finder. Cambridge, MA, 1973. 1

[PSdL+09] Pierre Peterlongo, Gustavo Akio Tominaga Sacomoto, Alair Pereira do Lago, Nadia Pisanti, e Marie-France Sagot. Lossless filter for multiple repeats with bounded edit distance. *Algorithms for Molecular Biology*, 4, 2009. 43

[RSM05] Kim R. Rasmussen, Jens Stoye, e Eugene W. Myers. Efficient q-gram filters for finding all epsilon-matches over a given length. Em *Research in Computational Molecular Biology, 9th Annual International Conference, RECOMB 2005, Cambridge, MA, USA, May 14-18, 2005, Proceedings*, páginas 189–203, 2005. 43

[Sag98] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. Em *Proceedings of the Third Latin American Symposium on Theoretical Informatics*, LATIN '98, páginas 374–390, London, UK, 1998. Springer-Verlag. 43

[SJ08] Jay Shendure e Hanlee Ji. Next-generation dna sequencing. *Nature Biotechnology*, 26(10):1135–1145, 2008. 1

[Ukk95] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995. 2, 23, 24, 25, 26, 27, 30, 32, 39, 43, 47, 52

[Wei73] Peter Weiner. Linear pattern matching algorithms. Em *14th Annual Symposium on Foundations of Computer Science, 15-17 October 1973, The University of Iowa, USA*, páginas 1–11, 1973. 1, 23, 24, 25, 26, 27, 30, 32, 43

[ZE98] Oren Zamir e Oren Etzioni. Web document clustering: A feasibility demonstration. Em *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, páginas 46–54, 1998. 58