Casamento Aproximado de Padrões

Mário Massato Harada

Casamento Aproximado de Padrões

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Mário Massato Harada e aprovada pela Comissão Julgadora.

Campinas, 14 de abril de 1994.

Prof. Cláudio Leonardo Lucchesi
Orientador

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

UNICAMP

Aos meus pais Jun e Tereza e à minha irmã Celina

Agradecimentos

Agraceço especialmente à minha família que sempre me apoiou durante toda essa jornada do mestrado. Aos meus pais Jun e Tereza, e a minha irmã Celina meus agradecimentos pelo amor, carinho e apoio que me foram essenciais durante este trabalho.

Meu agradecimento especial à minha namorada e amiga Márcia pelo seu amor e compreensão que foram inestimáveis a minha pessoa.

Aos meus tios o meu agradecimento pelo grande estímulo.

Ao orientador e amigo Cláudio Lucchesi agradeço pela oportunidade oferecida, ao apoio nas mais diversas situações e aos bons conselhos pessoais que recebi.

Aos colegas da turma de 91 (Inês, Heitor, Zacharias, Atta, Carrilho, Juliano...) agradeço pelo apoio e pelos vários momentos de descontração vividos durante o mestrado. Um agradecimento aos companheiros Herbert, Marcelo e Marcus pelas valiosas sugestões dadas para este trabalho. Agradeço também aos amigos Atta, Carmelo, Fábio Lucena, Marcelo Henriques, Marcus Vinícius e Thierson pela companhia constante e alegre.

Agradeço aos professores e a todos os funcionários do departamento de Computação.

Finalmente, agradeço ao Conselho Nacional de Pesquisa e Desenvolvimento (CNPq), à Fundação de Apoio e a Pesquisa do Estado de São Paulo (FAPESP), ao Fundo de Apoio a Pesquisa e ao Ensino da Unicamp (FAEP) pelo suporte financeiro para realização deste trabalho.

Abstract

In this work, we study some algorithms that give solutions to the three variations of the problem of approximate string matching: k-differences, k-mismatches, patterns with don't care symbols. In this last problem we will not study a specific algorithm that solves it but we study a generic algorithm that solves the three problems. Our main objective is to describe and analize clearly and precisely some algorithms for the three problems.

In Chapter 2 we study the algorithm of Ukkonen that gives a basis for some algorithms in Chapter 3.

In Chapter 3 we present solutions to the k-differences problem. It will be presented the algorithm of Ukkonen, the algorithm of Galil and Park and the algorithm of Tarhio and Ukkonen. The algorithm of Ukkonen is a modification of the original one presented in Chapter 2, the algorithm of Galil and Park is an improvement of the algorithm of Ukkonen. The algorithm of Tarhio and Ukkonen uses the ideas from dynamic programming and preprocessing of the pattern.

In Chapter 4, we describe three algorithms that solve the k-mismatches problem: algorithm of Landau and Vishkin, algorithm of Baeza and Gonnet and the algorithm of Tarhio and Ukkonen. The first uses the ideas similar to the ideas of the algorithm of Knuth, Morris and Pratt, the last two algorithms use the pattern shift technique introduced in the algorithm of Boyer and Moore.

Finally, in Chapter 5, we describe the algorithm of Baeza and Gonnet and the algorithm of Wu and Manber, these algorithms are flexible enough to solve the three problems of approximate string matching.

Resumo

Neste trabalho estudaremos alguns algoritmos que fornecem soluções para três variações do problema de casamento aproximado de padrões: k diferenças, k colisões, e padrões com símbolos neutros. Neste último problema não estudaremos um algoritmo específico para solucioná-lo, mas um algoritmo genérico que soluciona os três problemas citados. Nosso objetivo principal é descrever e analisar de forma clara e precisa alguns algoritmos para os três problemas.

No Capítulo 2 estudaremos o algoritmo de Ukkonen que servirá de base para alguns algoritmos do Capítulo 3.

No Capítulo 3 apresentaremos soluções para o problema das k diferenças. Serão apresentados os algoritmos de Ukkonen, o algoritmo de Galil e Park e o algoritmo de Tarhio e Ukkonen. O algoritmo de Ukkonen é uma modificação do algoritmo original apresentado no Capítulo 2, o algoritmo de Galil e Park é uma melhoria do algoritmo de Ukkonen. Já o algoritmo de Tarhio e Ukkonen utiliza as idéias da programação dinâmica e do pré-processamento do padrão.

No Capítulo 4 descreveremos três algoritmos que fornecem soluções para o problema das k colisões: algoritmo de Landau e Vishkin, algoritmo de Baeza-Yates e o algoritmo de Tarhio e Ukkonen. O primeiro utiliza idéias semelhantes às idéias do algoritmo de Knuth, Morris e Pratt, os dois últimos algoritmos usam as idéias de deslocamento do padrão encontradas no algoritmo de Boyer e Moore.

Por fim, no Capítulo 5, apresentamos os algoritmos de Baeza-Yates e Gonnet e o algoritmo de Wu e Manber que apresentam algoritmos flexíveis para resolver os três problemas do casamento aproximado de padrões.

Conteúdo

1 Introdução **1**

1.1 Casamento Aproximado de Padrões 3
1.2 Objetivos e Organização da Dissertação 6
1.3 Casamento exato de padrões: Algoritmos Clássicos 9
1.3.1 Algoritmo Knuth-Morris-Pratt 9
1.3.2 Algoritmo Boyer-Moore 11
1.3.3 Algoritmo de Aho e Corasick 15

2 Distância de Edição **17**

2.1 A distância de Edição 17
2.2 Algoritmo de Ukkonen 18
2.2.1 O algoritmo melhorado 19
2.2.2 Análise de Complexidade 27

3 Algoritmos para casamento aproximado com k diferenças **29**

3.1 Introdução 29
3.2 O algoritmo de Ukkonen 31
3.3 Algoritmo de Galil e Park 49
3.3.1 Introdução 49
3.3.2 Triplas de referência 49
3.3.3 A tabela Pref 50
3.3.4 Uma nova versão de *CalculaM*. 51
3.3.5 Uma nova ordem de cálculo 55
3.3.6 A complexidade do novo algoritmo 58
3.4 Algoritmo de Tarhio e Ukkonen 69
3.4.1 Introdução 69
3.4.2 Operação de marca 69

3.4.3 Operação de deslocamento 76
3.4.4 Análise de complexidade 81

4 Algoritmos para casamento aproximado com k colisões 85
4.1 Algoritmo de Landau e Vishkin 86
4.1.1 Análise do texto 88
4.1.2 Pré-processamento do padrão 94
4.1.3 Análise de Complexidade 97
4.2 Algoritmo de Baeza-Yates 100
4.3 Algoritmo de Tarhio e Ukkonen 108

5 Algoritmos genéricos 113
5.1 Algoritmo de Baeza-Yates e Gonnet 113
5.1.1 Padrões contendo símbolos neutros 117
5.1.2 Casamento aproximado com k colisões 118
5.2 Algoritmo de Wu e Manber 123
5.2.1 Extensões 127
5.2.2 Testes empíricos 128

6 Conclusão 131

Lista de Tabelas

6.1 Tabela comparativa dos algoritmos para o problema das k diferenças. 132
6.2 Tabela comparativa dos algoritmos para o problema das k colisões. 133
6.3 Tabela comparativa dos algoritmos genéricos 133

Lista de Figuras

1.1 Deslocamento efetuado pelo algoritmo Knuth,Morris e Pratt. 10
1.2 Deslocamento efetuado pelo algoritmo de Boyer e Moore dado pela heurística de casamento. 13
1.3 Deslocamento efetuado pelo algoritmo de Boyer e Moore dado pela heurística de ocorrência. 14
1.4 Transições do autômato de Aho e Corasick. 16

2.1 Matriz $D(i,j)$ com grafo de dependência e legenda dos arcos. 20
2.2 Algoritmo *Programação Dinâmica.* 21
2.3 Faixa de diagonais. 24
2.4 Algoritmo *Teste.* 25
2.5 Algoritmo *Distância.* 28

3.1 Exemplo para a matriz D com custos unitários. 34
3.2 Diagrama representativo do algoritmo *CalculaM.* 35
3.3 Algoritmo *CalculaM.* 36
3.4 Algoritmo *Diferenças-UK.* 41
3.5 Algoritmo *Diferença-UK-Normal.* 43
3.6 Algoritmo *Constrói-Pref* 51
3.7 Algoritmo *CalculaM2.* 53
3.8 Algoritmo *Diferenças-GP* 56
3.9 Algoritmo *CalculaM3.* 59
3.10 Diagrama representativo do conjunto de k vizinhança. 72
3.11 Algoritmo *Constrói-Ruim* 76
3.12 Algoritmo *Analisa Diagonal* 77
3.13 Algoritmo *Diferenças-TU* 79
3.14 Algoritmo *Constrói-Des* 81
3.15 Gráficos do teste empírico do algoritmo de Tarhio e Ukkonen. 83

3.16 Gráficos do teste empírico do algoritmo de Tarhio e Ukkonen. 84

4.1 Diagrama de representação do pré-processamento do padrão no algoritmo *Colisões-LV*. 86
4.2 Representação de alinhamentos entre texto e padrão. 87
4.3 Tabela TC para $A = adbc$ e $B = abaadccb$. 89
4.4 Representação das iterações do algoritmo de Baeza-Yates. . . 90
4.5 Representação de iterações do algoritmo *Colisões-LV*. 91
4.6 Algoritmo *Colisões-LV* 93
4.7 Algoritmo *Ingênuo* 94
4.8 Algoritmo *Intercala-Iterações* 95
4.9 Algoritmo *Análise-Padrão* 98
4.10 Diagrama explicativo da obtenção da tabela s_j 102
4.11 Algoritmo *Calcula-s_j* 104
4.12 Gráfico comparativo entre o algoritmo de Baeza-Yates e o algoritmo Ingênuo. 106
4.13 Gráfico comparativo entre o algoritmo de Baeza-Yates e o algoritmo Ingênuo. 107
4.14 Algoritmo *Colisões-TU* 109
4.15 Gráfico comparativo entre os algoritmos *Diferenças-UK* e *Diferenças-TU* 110
4.16 Gráfico comparativo entre os algoritmos *Diferenças-UK* e *Diferenças-TU* 111

5.1 Transição de S^j para S^{j+1} 115
5.2 Diagrama de funcionamento do algoritmo de Baeza-Yates e Gonnet para o problema de casamento exato de padrões . . 116
5.3 Algoritmo para calcular a tabela N. 117
5.4 Diagrama de funcionamento do algoritmo de Baeza-Yates e Gonnet para o problema de casamento aproximado de padrões. 120
5.5 Comparação dos algoritmos de Landau e Vishkin, Baeza-Yates e Gonnet e o método Ingênuo com $k = 1$. 121
5.6 Comparação dos algoritmos de Landau e Vishkin, Baeza-Yates e Gonnet e o método Ingênuo para $k = 2$. 121
5.7 Comparação dos algoritmos de Landau e Vishkin, Baeza-Yates e Gonnet e o método Ingênuo para $k = 3$. 122
5.8 Comparação dos algoritmos de Galil-Park, Ukkonen e Wu-Manber para k variando no intervalo $[0,6]$ e com $|\Sigma| = 2$. . . 129

5.9 Comparação dos algoritmos de Galil-Park, Ukkonen e Wu-Manber para k variando no intervalo $[0,6]$ e com $|\Sigma| = 30$. . 130

Capítulo 1

Introdução

Certamente, há algum tempo o problema de casamento de padrões, e suas generalizações, vem recebendo uma atenção considerável por parte dos pesquisadores que buscam solucioná-lo de maneira mais eficiente possível. Em sua versão básica o problema do casamento exato de padrões é definido da seguinte forma: sejam dadas duas seqüências de símbolos A de tamanho m e B de tamanho n, $m \leq n$, deseja-se encontrar todas as posições i em B onde A ocorre, ou seja, $B(i+1 \ldots i+m) = A(1 \ldots m)$ onde $B(i+1 \ldots i+m)$ representa os caracteres de B que estão nas posições que vão de $i+1$ a $i+m$.

Por exemplo, a solução para A=abc e B=dcfgabcedafbabc são as posições 5 e 13 de B, veja:

dcfgabcedagbabc
abc abc

A solução do problema é aplicável em vários cenários. Uma das formas mais comuns é saber se um dado padrão ocorre em um texto. Nesta mesma linha, sistemas que implementam dicionários valem-se da busca de termos fornecidos pelo usuário para apresentar o significado dos mesmos.

O reconhecimento de padrões em seqüências de caracteres está relacionado com um problema correspondente em tratamento de imagens. De fato, alguns algoritmos de casamento de padrões podem ser generalizados para a forma bidimensional [KS87], pois, a extração do contorno de um objeto bidimensional leva a uma forma unidimensional que, para efeito de solução, é considerada uma seqüência de símbolos, logo, neste caso, a aplicabilidade do problema é imediata.

Na década de 70 vários algoritmos bastante eficientes para casamento exato de padrões foram propostos, dentre os quais destacamos três algoritmos clássicos: o algoritmo de Knuth, Morris e Pratt [KMP77], o algoritmo de Boyer e Moore [BM77], o algoritmo de Aho e Corasick [AC75]. Estes algoritmos, além de sua simplicidade, apresentam comportamento linear em relação ao tamanho do texto, em particular [BM77] apresenta um comportamento sublinear na maioria dos casos. Um estudo detalhado do problema de casamento exato de padrões pode ser encontrado em [AL93].

Uma série de novas aplicações surgiram para o problema de casamento de padrões. Por exemplo, em caso de entrada de dados duvidosos, causada por erro de digitação, a busca por seqüências parecidas ajuda na crítica de dados. Neste mesmo tipo de aplicação, sistemas que implementam dicionários necessitam realizar buscas de palavras que possuam certo grau de semelhança para apresentar um conjunto de sugestões ao usuário. Um exemplo para este tipo de aplicação é dada em [LK93]. Um estudo sobre o problema de casamento exato e aproximado pode ser encontrado em [Gra92].

Note que agora temos uma variação do problema do casamento de padrões, estamos preocupados em fazer um reconhecimento aproximado do padrão. Uma das áreas em que a solução para o problema do casamento aproximado de padrões é muito utilizado é a biologia computacional, que estuda o DNA para reconhecer as subseqüências de nucleotídeos que o compõem.

Diante das inovações das necessidades pelas variações do problema de casamento aproximado de padrões surgiram novas classes do problema, dentre as quais podemos citar:

- casamento aproximado com um número máximo de diferenças;
- casamento aproximado com um número máximo de colisões;
- casamento aproximado com o padrão contendo símbolos neutros.

O problema de casamento de padrões conta ainda com outras variações, tais como: reconhecimento em duas ou três dimensões, reconhecimento da maior subseqüência comum, etc. Para cada uma dessas modalidades do problema de casamento de padrões, existem inúmeras soluções que visam explorar diferentes características de cada situação. Desta forma, podemos notar que uma discussão detalhada das diversas soluções para o problema

de casamento de padrões é um assunto bastante longo. Assim, nesta dissertação vamos nos restringir ao problema do casamento aproximado de padrões.

1.1 Casamento Aproximado de Padrões

Conforme relatamos acima, um estudo detalhado dos diversos problemas do casamento de padrões tornaria esta dissertação bastante longa. Assim, concentraremos no problema do casamento aproximado de padrões. Para este problema, temos várias classes dentre as quais podemos citar: casamento aproximado com no máximo k diferenças, casamento aproximado com no máximo k colisões e casamento aproximado com o padrão contendo símbolos neutros.

O problema do casamento aproximado de padrões com no máximo k diferenças surge quando desejamos reconhecer um padrão em um texto onde se permite um determinado número de diferenças entre uma subseqüência do texto e uma subseqüência do padrão. Basicamente existem três tipos de diferenças:

1. Um símbolo do padrão corresponde a um símbolo diferente do texto. Neste caso seria necessária uma substituição de um símbolo. Esta diferença é chamada de *colisão*;

2. Um símbolo do padrão não tem um correspondente no texto. Neste caso seria necessária uma inserção;

3. Um símbolo do texto não tem um correspondente no padrão. Neste caso seria necessária uma remoção de um símbolo do texto.

Então o problema do casamento aproximado de padrões com no máximo k símbolos diferentes (daqui por diante, o problema das k diferenças) consiste em encontrar as posições em que um padrão A ocorre como subseqüência de um texto B com no máximo k diferenças, onde consideraremos que uma subseqüência de B são caracteres que ocupam posições consecutivas de B.

Por exemplo, considere o texto B=abcdefghi e o padrão A=bxdyegh e $k = 3$. Existe uma ocorrência de A em B, a partir do segundo símbolo, com 3 diferenças. A correspondência obtida é: b (do texto) corresponde a b(do

padrão), c corresponde a x (uma substituição), d com d, nada com y (uma inserção), e com e, f com nada (uma remoção), g com g, h com h. Veja a ilustração abaixo:

```
|bxdye gh|
a|bcd efgh|i
```

Note que este problema é uma generalização do problema clássico de casamento exato de padrões. Basta fazer $k = 0$ e teremos que buscar as ocorrências exatas de B em A. Por outro lado, se desejarmos saber o número de diferenças para transformar A em B ou vice-versa então temos o problema da *distância de edição* entre duas seqüências de caracteres.

A aplicação deste problema é bastante freqüente em biologia computacional. Seu objetivo é estudar as seqüências de nucleotídeos que compõem o DNA. Uma cadeia de DNA pode ser considerada como uma seqüência de símbolos formada sobre o alfabeto Σ={C,G,T,A}, onde cada símbolo representa uma das quatro bases nitrogenadas (citosina, guanina, timina, adenina).

Tem sido de grande importância para esta área descobrir seqüências homólogas entre DNA. Isto se deve ao interesse em descobrir como as seqüências de DNA estão relacionadas e também para obtenção de linhagens melhoradas geneticamente.

Mesmo o problema clássico (reconhecimento exato) é utilizado na biologia: certas doenças (como por exemplo a distrofia muscular) somente se manifestam a partir de uma certa idade, mas é possível o diagnóstico precoce através da busca de certos padrões genéticos. Alguns autores [Miu80, Fra91, Lan91] descrevem de maneira mais técnica as razões do uso de casamento de padrões em biologia.

Devido a mais esta importância várias propostas foram feitas para o problema das k diferenças, dentre estas destacamos os algoritmos de Ukkonen [Ukk85], Galil e Park [GP90], Tarhio e Ukkonen [TU91]. O algoritmo de Ukkonen [Ukk85] é uma melhoria da solução trivial dada pelas técnicas de programação dinâmica, já a proposta de Galil e Park [GP90] melhora a solução de Ukkonen com o pré-processamento do padrão (idéia já usada em [KMP77]), Tarhio e Ukkonen [TU91] generalizaram a idéia de Boyer e Moore [BM77] para o problema das k diferenças.

Uma outra variação do problema do casamento de padrões é conhecido como casamento aproximado de padrões com no máximo k colisões (daqui

para frente, o problema das k colisões). Nesta modalidade, a solução consiste em buscar ocorrências do padrão no texto que possuam no máximo k diferenças do tipo 1, ou seja, permitem-se apenas diferenças ocasionadas por substituição de símbolos.

Por exemplo, considere o texto B=bbababacaacbb e o padrão A=aaaaabaaab e $k = 4$. Neste caso temos uma ocorrência na posição 3 de B com 4 colisões. Veja:

bbababacaacbb
aaaaabaaab
↑ ↑ ↑ ↑

Para esta classe do problema podemos citar as seguintes propostas: Landau e Vishkin [LV86] que realizam um pré-processamento do padrão e utilizam uma idéia semelhante ao algoritmo de Knuth, Morris e Pratt [KMP77], Baeza-Yates [BY89] e Tarhio e Ukkonen [TU91] que generalizam a idéia do algoritmo Boyer e Moore [BM77].

A última variação do problema (padrões com símbolos neutros) possui importantes aplicações. Uma delas é quando se deseja encontrar uma resenha em um texto que possua a ocorrência de duas palavras separadas por um determinado número de caracteres. Por exemplo, podemos querer saber se existe alguma palavra que identifique algum tipo de relação entre um sapo e um rato, para isto basta procurar a ocorrência de sapo seguida de rato com um número k de símbolos neutros entre sapo e rato. Este problema será tratado aqui como o *problema do casamento aproximado com símbolos neutros*, onde os símbolos neutros, denotados por ϕ, são aqueles que não acarretam nenhum tipo de colisão.

Este tipo de busca também é comum em biologia computacional. Por exemplo, o padrão de DNA TATA (timina, adenina, timina, adenina - conhecida como caixa de Hogness [MBY91]) sempre aparece antes da seqüência CAATCT (conhecida como caixa CAAT). Procuremos na seqüência

B=AAGCTACTGCCCTATAGCGCCAGGGATTCAATCTGGCCAAA

a ocorrência de TATA e CAATCT com 12 símbolos neutros entre os mesmos, ou seja, $k = 12$. A solução indica uma ocorrência no décimo quarto símbolo de B. Veja:

AAGCTACTGCCCTATAGCGCCAGGGATTCAATCTGGCCAAA
TATA$\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi$CAATCT

Temos duas variantes deste problema: numa, como no exemplo acima, exigem-se exatamente k símbolos ignorados no texto; na outra variante, aceitam-se até k símbolos ignorados.

De outra forma, o problema de casamento aproximado de padrões com símbolos neutros pode ser definido da seguinte forma: seja ϕ um símbolo neutro, um texto $B(1\ldots n) \in \Sigma^*$ e um padrão $A(1\ldots m) \in (\Sigma^+\phi^+)^*\Sigma^+$, devemos, então, encontrar os índices i tais que $1 \leq i \leq n-m+1$ e $t_i \ldots t_{i+m-1} = p_1 \ldots p_m$, onde dois símbolos são iguais se eles denotam os mesmos elementos de Σ ou se um deles é o símbolo neutro ϕ.

Várias propostas surgiram para este problema, dentre estas podemos citar as soluções de Pinter [Pin85], Baeza-Yates e Gonnet [BYG89], Manber e Baeza-Yates [MBY91] e Wu e Manber [WM91].

Pinter em sua solução realiza uma adaptação do algoritmo de Aho e Corasick [AC75]. Nesta proposta, Pinter considera as subseqüências entre os símbolos neutros como novos padrões e, então, reduz o problema a uma busca múltipla de vários padrões.

Baeza-Yates e Gonnet transformam o padrão e o texto em valores numéricos que são armazenados em um vetor. Através de operações de deslocamento e operações **and** sobre o vetor numérico, Baeza-Yates e Gonnet resolvem o problema de casamento exato de padrões de maneira simples e eficiente.

Manber e Baeza-Yates apresentam uma solução para casamento de padrões com apenas uma única seqüência de símbolos neutros. Esta solução foi obtida através da redução do problema de padrões com símbolos neutros para um problema em geometria computacional.

1.2 Objetivos e Organização da Dissertação

Estabeleceremos agora o objetivo e organização deste trabalho. O objetivo desta dissertação é apresentar (descrever e analisar) alguns algoritmos das três classes do problema do casamento aproximado de padrões.

Agora, vamos estabelecer os objetivos da descrição e da análise de cada algoritmo. O objetivo inicial da descrição dos algoritmos é descrevê-los de maneira clara e simples, nesta descrição pretendemos eliminar a falta

de clareza existente na grande maioria dos artigos da área, além disso, procuraremos descrever os algoritmos de maneira mais didática o possível.

Nesta dissertação os algoritmos serão descritos em *pseudo-código*. Algumas vezes, o símbolo $\ll n \gg$, onde n é um valor numérico, será utilizado nos algoritmos. Estes símbolos não têm qualquer efeito no algoritmo e servirão apenas para referenciar o comando na linha do algoritmo em que o símbolo é encontrado. Outras vezes, alguns comandos dos algoritmos serão marcados com um contorno (*comando*). Estas marcas indicam que o comando não tem influência no algoritmo, tais comandos contornados são utilizados para auxiliar as demonstrações e explicações relacionadas ao algoritmo.

Por fim, o objetivo da análise dos algoritmos é avaliar o comportamento dos algoritmos na situação de pior caso (complexidade de tempo). Consideraremos também em nossas análises questões de complexidade de espaço.

Como de praxe, nas análises dos algoritmo utilizaremos a notação O ("O grande") cujas descrição pode ser encontrada em várias publicações tais como [AHU74], [Ziv93].

Como restringiremos nossos estudos a algoritmos que realizarão buscas de um padrão em um texto, então consideraremos que o comportamento dos algoritmos será determinado pelo número de comparações entre caracteres do texto e do padrão. Não consideraremos outros tipos de comparações, como por exemplo, entre variáveis de controle.

Finalmente, vejamos como esta dissertação foi organizada. Inicialmente, gostaríamos de ressaltar que devido ao grande número de soluções existentes para os variados problemas do casamento aproximado de padrões, escolhemos aqueles algoritmos que apresentavam idéias que geraram várias outras soluções, e que também, em nossa opinião, possuem soluções elegantes, ou por vezes, tem importância histórica.

Descreveremos agora a organização desta dissertação. Inicialmente, no restante deste capítulo, descreveremos de maneira sucinta três algoritmos que são considerados referências importantes na área de casamento de padrões. Os algoritmos são o algoritmo de Knuth, Morris e Pratt [KMP77], o algoritmo de Boyer e Moore [BM77] e o algoritmo de Aho e Corasick [AC75]. Muitos dos algoritmos para as diversas variações do problema de casamento aproximado valem-se das idéias destes algoritmos.

No capítulo 2 estudamos o algoritmo de Ukkonen [Ukk85] para o problema da *distância de edição*. Vale ressaltar que abordamos esse problema pois este algoritmo servirá de base para a descrição de alguns algoritmos

para o problema das k diferenças. Na descrição deste algoritmo conseguimos, em nossa opinião, uma versão mais clara e correta do algoritmo. Correta no sentido de que eliminamos problemas de contorno do algoritmo.

No capítulo 3 apresentamos soluções para o problema das k diferenças. Neste capítulo apresentamos os algoritmos de Ukkonen [Ukk85], o algoritmo de Galil e Park [GP90] e o algoritmo de Tarhio e Ukkonen [TU91]. É interessante destacar que o algoritmo de Ukkonen descrito neste capítulo é uma pequena modificação do algoritmo original que tratava do problema da distância de edição. O algoritmo de Galil e Park consiste de uma melhoria do algoritmo de Ukkonen. Para diminuir o número de comparações realizadas pelo algoritmo de Ukkonen este algoritmo utiliza uma tabela proveniente do pré-processamento do padrão e armazena informações de igualdades entre subseqüências do texto e do padrão, tais informações são obtidas durante a execução do algoritmo. O algoritmo de Tarhio e Ukkonen alia as idéias da programação dinâmica a um pré-processamento do padrão semelhante ao do algoritmo de Boyer e Moore.

No capítulo 4, estudamos soluções para o problema das k colisões. Neste capítulo são estudados os algoritmos de Landau e Vishkin [LV86], o algoritmo de Baeza-Yates [BY89] e o algoritmo de Tarhio e Ukkonen [TU91]. O algoritmo de Landau e Vishkin é divido em duas fases: pré-processamento do padrão e análise do texto. Este algoritmo, de certa forma, utiliza as idéias do algoritmo de Knuth, Morris e Pratt. O algoritmo de Baeza-Yates vale-se do pré-processamento do padrão e realiza deslocamentos do padrão semelhantes aos do algoritmo de Boyer e Moore. Já o algoritmo de Tarhio e Ukkonen utiliza-se também das idéias do algoritmo de Boyer e Moore.

Por fim, no capítulo 5, apresentamos dois algoritmos: algoritmo de Baeza-Yates e Gonnet [BYG89] e o algoritmo de Wu e Manber [WM91]. O algoritmo de Baeza-Yates e Gonnet propõe uma solução para resolver o problema de casamento exato de padrões baseado em um método numérico. A partir desta solução, Baeza-Yates e Gonnet generalizam a idéia para resolver o problema das k colisões e também o problema do casamento de padrões com símbolos neutros. Wu e Manber baseados na idéia do algoritmo de Baeza-Yates e Gonnet desenvolveram um algoritmo bastante flexível que fornece soluções para os diversos problemas de casamento de padrões. Por exemplo: k diferenças, k colisões e casamento de padrões com símbolos neutros e busca de vários padrões no texto.

1.3 Casamento exato de padrões: Algoritmos Clássicos

Nesta seção exporemos de maneira rápida dois algoritmos para o problema de casamento exato de padrões. A razão para existência desta seção deve-se ao fato de que estes algoritmos possuem importância histórica e contribuíram com suas idéias para o surgimento de novas soluções para as variações do problema de casamento de padrões. Os algoritmos são de autoria de Knuth, Morris e Pratt [KMP77], Boyer e Moore [BM77] e Aho e Corasick [AC75].

De maneira formal podemos definir o problema de casamento exato de padrões da seguinte forma: seja $A(1 \ldots m)$ e $B(1 \ldots n)$ seqüências de caracteres tal que $m \leq n$. O problema de casamento exato de padrões consiste em encontrar todas as posições de ocorrência de A em B. Considere $S(T, P)$ o conjunto dessas posições, então um algoritmo que resolve o problema de casamento exato de padrões devolve:

$$S(T,P) = \{l : (1 \leq l \leq n-m) \text{ e } (T(l+i) = P(i)), 1 \leq i \leq m\}$$

Uma maneira imediata de resolver o problema é, em um dado alinhamento, comparar texto e padrão, caracter a caracter, até encontrarmos uma colisão ou até encontrarmos uma ocorrência do padrão. Após as comparações, deslocaríamos o padrão de uma posição para verificar uma outra ocorrência. Este método é conhecido como *método ingênuo.*

Apresentaremos agora os algoritmo clássicos.

1.3.1 Algoritmo Knuth-Morris-Pratt

A idéia básica do algoritmo é tentar aproveitar os caracteres já reconhecidos no texto para conseguir um maior deslocamento do padrão. A vantagem desta abordagem é que evitamos repetir comparações com caracteres que já foram comparados.

O algoritmo vale-se do pré-processamento do padrão que resulta em uma tabela que será consultada quando ocorrer uma colisão. Esta tabela fornecerá a próxima posição do padrão a ser comparada com o texto após uma colisão.

A função desta tabela é determinar a maior parte aproveitável do texto analisado para economia de comparações. Suponha que $A(1 \ldots i)$, $1 \leq i \leq$

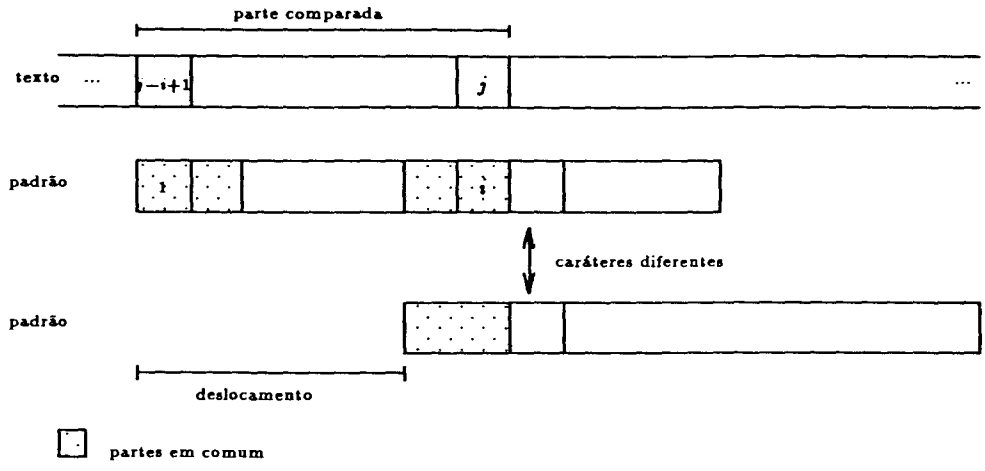

Figura 1.1: Deslocamento efetuado pelo algoritmo Knuth,Morris e Pratt.

m tenha sido comparado com $B(j-i+1\ldots j)$. Além disso, suponha que $A(i+1) \neq B(j+1)$. A tabela resultante do pré-processamento devolverá o maior prefixo de tamanho l de $A(1\ldots m)$ que é sufixo de $A(1\ldots i)$, ou seja, $A(1\ldots l) = A(i-l+1\ldots i)$. Além disso, l é tal que $A(l+1) \neq A(i+1)$. Assim, devemos comparar $A(l+1)$ com $B(j+1)$, pois, $A(1\ldots l) = A(i-l+1\ldots i) = B(j-l+1\ldots j)$. Veja a figura 1.1 que ilustra esta situação.

Esta mesma idéia foi implementada independentemente por Aho e Corasick [AC75] para busca de vários padrões em um texto. A idéia de Aho e Corasick é construir uma trie generalizando a idéia da tabela construída pelo algoritmo de Knuth, Morris e Pratt. Entretanto, o algoritmo depende do tamanho do alfabeto, o que pode levar a uma alta complexidade de espaço.

Agora ilustraremos o algoritmo com um exemplo dado em [KMP77]. Seja B=abcaabcabcabcacabc e A=abcabcacab. Procuraremos uma ocorrência de A em B.

O alinhamento de A na posição 1 de B resulta em uma colisão entre $A(5)$ e $B(5)$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	b	c	a	a	b	c	a	b	c	a	b	c	a	c	a	b	c
a	b	c	a	b	c	a	c	a	b								
				↑													

Note que o maior prefixo de $A(1 \ldots 10)$ que é sufixo de $A(1 \ldots 4)$ é $A(1)$. No entanto, $A(2) = A(5)$. Assim, a próxima comparação a ser realizada é $A(1)$ com $B(5)$. Em outras palavras, podemos deslocar o padrão de 4 posições:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	b	c	a	a	b	c	a	b	c	a	b	c	a	c	a	b	c
				a	b	c	a	b	c	a	c	a	b				
											↑						

Observe que no alinhamento acima temos uma colisão entre $A(8)$ e $B(12)$. Veja que o maior prefixo de $A(1 \ldots 10)$ que é sufixo de $A(1 \ldots 7)$ é $A(1 \ldots 4)$. Além disso, temos que $A(5) \neq A(8)$. Logo, a próxima comparação é $A(5)$ com $B(12)$:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
a	b	c	a	a	b	c	a	b	c	a	b	c	a	c	a	b	c
							a	b	c	a	b	c	a	c	a	b	
																	↑

Note que no alinhamento acima encontramos uma ocorrência de A em B.

A seguir apresentaremos o algoritmo de Boyer e Moore.

1.3.2 Algoritmo Boyer-Moore

A idéia básica do algoritmo de Boyer e Moore para o problema de casamento exato de padrões é que obtemos mais informações se as comparações forem feitas da direita para a esquerda, ou seja, a cada alinhamento do texto e do padrão, os caracteres alinhados são examinados da direita para esquerda a partir do último caracter do padrão. Entre alinhamentos sucessivos o padrão é deslocado para a direita. Considere um padrão A e um texto B de tamanhos m e n, respectivamente.

Os deslocamentos no algoritmo de Boyer e Moore são realizados de acordo com duas heurísticas. Considere o alinhamento do padrão A na posição $j - i + 1$ do texto, ou seja, $A(1 \ldots m)$ está alinhado com $B(j - i + 1 \ldots j - i + m)$. Suponha que já comparamos $A(i + 1 \ldots m)$ com $B(j + 1 \ldots j - i + m)$ e suponha também que essas comparações resultaram em igualdades, ou seja, $A(i + 1 \ldots m) = B(j + 1 \ldots j - i + m)$. Além disso, suponha que $A(i) \neq B(j)$, ou seja, temos uma colisão. Logo, devemos deslocar o padrão. Este deslocamento será dado pelas seguintes heurísticas:

- se $B(j)$ não ocorre em $A(1 \ldots i - 1)$ então não temos uma ocorrência de A nas posições $j - i + 1, j - i + 2, \ldots, j - 1$. Neste caso podemos deslocar o padrão de i posições. Por outro lado, se $B(j)$ ocorrer na posição l, $1 \leq l \leq i - 1$, de A, ou seja, $B(j) = A(l)$ então temos uma possível ocorrência do padrão quando $B(j)$ estiver alinhado com $A(l)$. Assim, podemos deslocar o padrão de $i - l$ posições. Veja a figura 1.2 que ilustra estes dois casos. Esta heurística é chamada de heurística de casamento;

- se a parte do texto $B(j+1 \ldots j-i+m)$ que já foi comparada reocorrer no padrão à esquerda da posição i então podemos ter uma possível ocorrência do padrão se alinharmos $B(j + 1 \ldots j - i + m)$ com sua reocorrência no padrão. Suponha que $B(j + 1 \ldots j - i + m) = A(l - (m-i-1) \ldots l)$, $l < i+1$, então podemos deslocar o padrão de $m-l+1$ posições. Veja a figura 1.3 que ilustra esta situação. Esta heurística é chamada de heurística de ocorrência.

O algoritmo de Boyer e Moore escolhe o maior dos deslocamentos fornecidos pelas duas heurísticas acima.

Para um melhor entendimento destas heurísticas exigiremos um exemplo retirado de [BM77]. Procuraremos por uma ocorrência de AT-THAT em WHICH-FINALLY-HALTS.-AT-THAT-POINT.

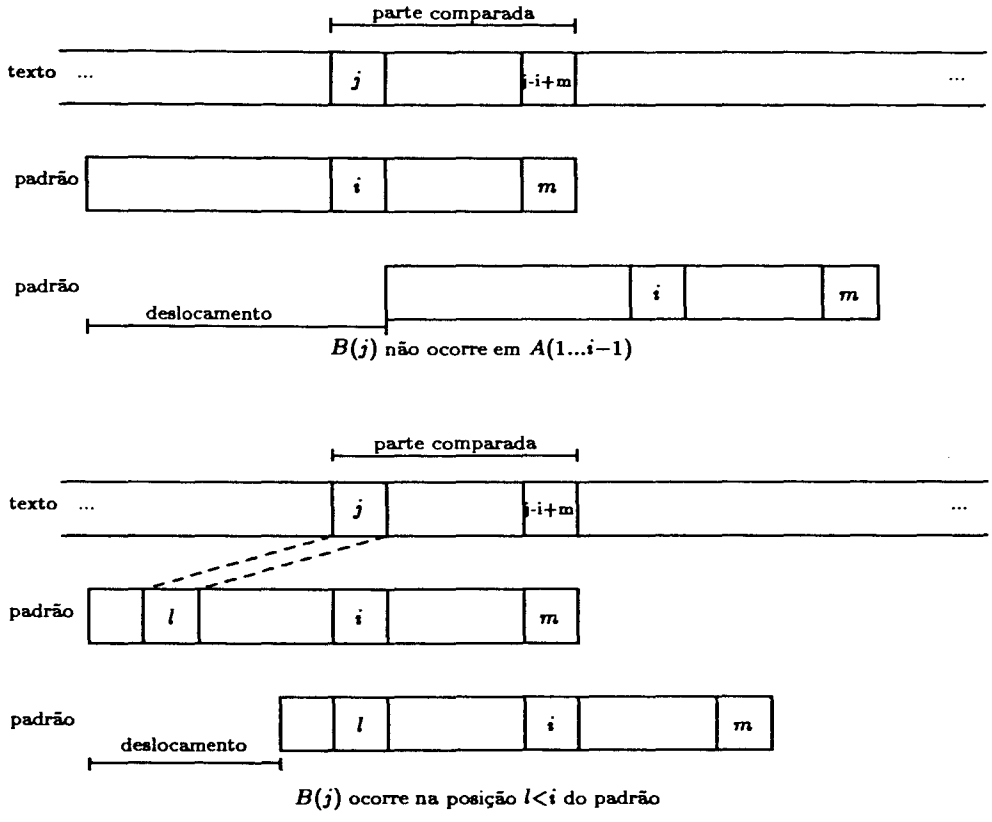

Figura 1.2: Deslocamento efetuado pelo algoritmo de Boyer e Moore dado pela heurística de casamento.

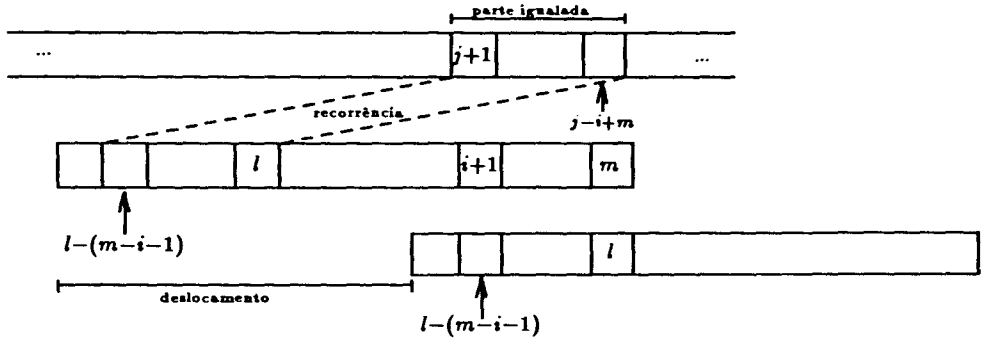

Figura 1.3: Deslocamento efetuado pelo algoritmo de Boyer e Moore dado pela heurística de ocorrência

O primeiro alinhamento indica uma colisão entre T e F:

```
A T-THAT
WHICH - FINALLY-HALTS.--AT-THAT-POINT
        ↑
```

Note que F não ocorre em A. Assim, podemos deslocar A de 7 posições:

```
    AT - THAT
WHICH-F I NALL Y - HALTS.--AT-THAT-POINT
                ↑
```

No alinhamento acima temos uma colisão entre - e T. como o hífen ocorre no padrão então deslocaremos A de 4 posições para a direita:

```
        AT-THAT
WHICH-FINALLY-HALTS.--AT-THAT-POINT
                ↑
```

Veja que temos uma colisão entre A e L. Como L não ocorre em A então deslocaremos o padrão de 6 posições:

```
                AT-THAT
WHICH-FINALLY-HALTS. - - AT-THAT-POINT
                      ↑
```

Observe que neste caso temos duas opções para o deslocamento: ou deslocamos devido à reocorrência de AT em A ou deslocamos devido à reocorrência do hífen. Como o deslocamento provocado pela reocorrência de AT em A é maior então deslocamos o padrão de 5 posições:

```
                                AT-THAT
WHICH-FINALLY-HALTS.--AT-THAT-POINT
                     ↑
```

Neste alinhamento encontramos uma ocorrência de A.
Apresentaremos a seguir o algoritmo de Aho e Corasick.

1.3.3 Algoritmo de Aho e Corasick

O algoritmo de Aho e Corasick baseia-se na construção de um autômato finito para encontrar as ocorrências de vários padrões em um texto. Aho e Corasick constroem três funções para o autômato: função de transição, função de falha e a função de saída. A função de transição Tr indica a transição do estado corrente para outro estado do autômato quando o caracter analisado é válido. Em caso contrário, a função de transição indica que houve uma falha. Já a função de falha F fornecerá um estado alternativo quando a função de transição indicar que houve uma falha. Por último, a função de saída S indicará quando um dos padrões ocorrer no texto.

Para exemplificar o algoritmo, considere o conjunto de padrões $\mathcal{P} = \{\text{he, she, his, hers}\}$ dado em [AC75].

A função de transição Tr tem como parâmetros o estado corrente e o caracter recebido pelo autômato. Para melhor ilustrar a função, considere o grafo de transição dado na figura 1.4, onde os números indicam os estados e os caracteres indicam a condição para transição. Assim, $Tr(0,h) = 1$, $Tr(1,e) = 2$ e $Tr(2,r) = 8$. Note que os prefixos em comum entre os padrões são aproveitados.

A função de falha indicará um estado alternativo quando ocorrer uma falha no estado corrente. Assim F é tal que

estado	1	2	3	4	5	6	7	8	9
F	0	0	0	1	2	0	3	0	3.

Veja que $F(4) = 1$ então isto significa que o estado corrente é o 4 e, portanto, foram reconhecidos os caracteres s e h. Assim, se ocorrer uma falha

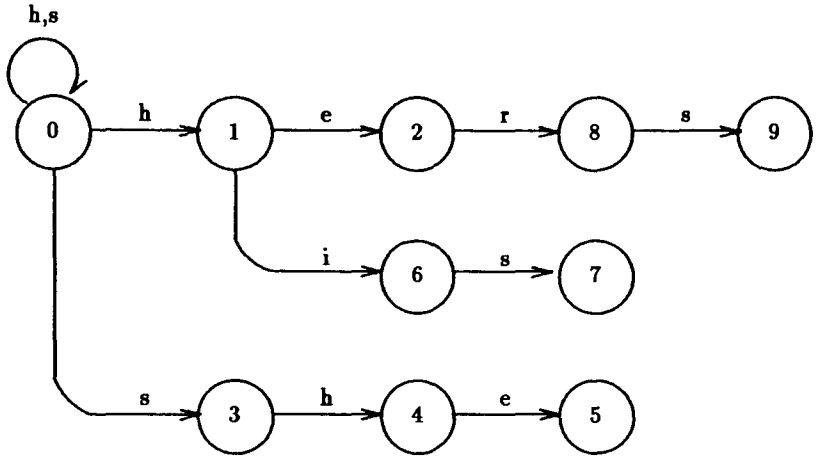

Figura 1.4: Transições do autômato de Aho e Corasick.

no estado 4 então o estado alternativo é o 1. Desta forma, aproveitamos o reconhecimento do caracter h.

A função de saída indicará quando um dos padrões de $\mathcal{P}$ for reconhecido. Assim, temos que os seguintes valores para $\mathcal{P}$

estado	2	5	7	9
S	he	she, he	his	hers

A seguir apresentaremos algumas soluções para o problema de k diferenças.

Capítulo 2

Distância de Edição

2.1 A distância de Edição

A noção de *distância de edição* foi introduzida por Levenshtein [Lev66] e consiste em estabelecer o custo do número mínimo de *operações de edição* para transformar uma seqüência em outra, ambas sobre um alfabeto Σ. Basicamente existem três tipos de operação de edição:

- a inserção de um caracter na seqüência a ser transformada: esta operação será denotada por $\epsilon \rightarrow a$;
- a remoção de um caracter da seqüência a ser transformada: esta operação será denotada por $a \rightarrow \epsilon$;
- a substituição de um caracter por outro caracter na seqüência a ser transformada: esta operação será denotada por $a \rightarrow b$,

onde a e b pertencem a Σ e ϵ é a seqüência vazia.

A cada uma destas três operações pode-se atribuir um *custo*. Assim, a *distância de edição* entre duas seqüências w e x, denotada por $\mathcal{E}(w, x)$, será o custo total do número mínimo de operações de edição para transformar w em x.

Apresentaremos a seguir um algoritmo para calcular a distância de edição, baseado na programação dinâmica e devido a Ukkonen [Ukk85].

2.2 Algoritmo de Ukkonen

A idéia do algoritmo de Ukkonen baseia-se na melhoria do algoritmo clássico da programação dinâmica. A simples metodologia do algoritmo da programação dinâmica tem como objetivo o cálculo de uma forma matricial que é responsável pelo armazenamento dos custos de transformação de subseqüências de uma cadeia em subseqüências de outra cadeia.

Mais especificamente, sejam A e B duas cadeias de caracteres de tamanho m e n, respectivamente. Agora, considere também a matriz D de tamanho $(m+1)$x$(n+1)$, onde cada valor $D(i,j)$, $0 \leq i \leq m$ e $0 \leq j \leq n$ conterá o valor da distância de edição entre $A(1 \ldots i)$ e $B(1 \ldots j)$. Em particular, $D(m,n)$ é a distância $\mathcal{E}(A,B)$.

Os elementos da matriz podem ser calculados da seguinte forma: $D(i,j)$ é:

$$\begin{cases} 0 & \text{se } i = 0 \text{ e } j = 0 \\ D(i,j-1) + \delta(\epsilon \rightarrow B(j)) & \text{se } i = 0 \\ D(i-1,j) + \delta(A(i) \rightarrow \epsilon) & \text{se } j = 0 \\ \min \begin{cases} D(i-1,\ j\ ) + \delta(A(i) \rightarrow\ \epsilon\ ) \\ D(i-1,j-1) + \delta(A(i) \rightarrow B(j)) \\ D(\ i\ ,j-1) + \delta(\ \epsilon\ \rightarrow B(j)) \end{cases} & \text{caso contrário} \end{cases} \tag{2.1}$$

onde δ é a função que fornece o custo da operação de edição.

Para um melhor entendimento da recorrência acima, observe que se a escolha para $D(i,j)$ for:

- $D(i-1,j) + \delta(A(i) \rightarrow \epsilon)$ então isto significa que $D(i,j)$ corresponde ao custo de transformar $A(1 \ldots i-1)$ em $B(1 \ldots j)$ mais o custo de remoção de $A(i)$;

- $D(i-1,j-1) + \delta(A(i) \rightarrow B(j))$ então isto significa que $D(i,j)$ corresponde ao custo de transformar $A(1 \ldots i-1)$ em $B(1 \ldots j-1)$ mais a substituição de $A(i)$ por $B(j)$;

- $D(i,j-1) + \delta(\epsilon \rightarrow B(j))$ então isto significa que $D(i,j)$ corresponde ao custo de transformar $A(1 \ldots i)$ em $B(1 \ldots j-1)$ mais o custo de inserção de $B(j)$.

As dependências entre os valores $D(i,j)$ podem ser ilustradas graficamente através de um arco orientado com origem em (i,j) e destino (i',j') se e somente se o passo de minimização de (2.1) gera $D(i',j')$ a partir de $D(i,j)$. O grafo resultante será chamado de *grafo de dependência.*

Observe que na figura 2.1 temos um exemplo em que A =yxxzy e B =xyxzyz. A função de custo δ utilizada é dada por:

$$\delta(a,b) := \begin{cases} 0 & \text{se } a = b \\ 2 & \text{se } a \neq b \text{ e } (a = \epsilon \text{ ou } b = \epsilon) \\ 3 & \text{nos demais casos.} \end{cases}$$

Observe que na figura 2.1 o elemento $D(0,1)$ teve como escolha mínima o valor $D(0,0) + \delta(\epsilon \rightarrow \text{x})$, logo, no grafo de dependência temos um arco de $(0,0)$ a $(0,1)$. Já o elemento $D(1,2)$ teve como escolha mínima o valor $D(0,1) + \delta(\text{y} \rightarrow \text{y})$, portanto temos um arco de origem em $(0,1)$ e destino $(1,2)$.

Observe que na figura 2.1 existem caminhos alternativos de $(0,0)$ a (m,n), isto se deve ao fato de que ocorreram empates durante os passos de minimização dado pela equação (2.1). No caso dos vértices $(1,2),(2,2),(2,3)$ e $(3,3)$ o custo de ir de $(1,2)$ a $(3,3)$ possui duas alternativas de caminho com mesmo custo: ou se faz uma substituição seguida de uma remoção ou o contrário. Observe que o problema consiste em encontrar um caminho do grafo de dependência.

Com base na recursão (2.1) podemos calcular a matriz D com complexidade $O(mn)$ em tempo e $O(m)$ em espaço (figura 2.2).

No entanto, observações feitas a respeito dos valores da matriz D indicam que a maioria destes não são efetivamente utilizados para o cômputo do resultado final. Ukkonen, baseado nesta observação, limita os cálculos a apenas alguns elementos da matriz D. Estas observações serão apresentadas a seguir.

2.2.1 O algoritmo melhorado

Nesta subseção descreveremos o algoritmo de Ukkonen cuja principal contribuição é a diminuição do número de elementos a serem calculados na matriz de programação dinâmica.

Podemos rotular as arestas do grafo de dependência com o custo $\delta(a \rightarrow b)$ correspondente à operação de edição escolhida e, conseqüentemente, cada

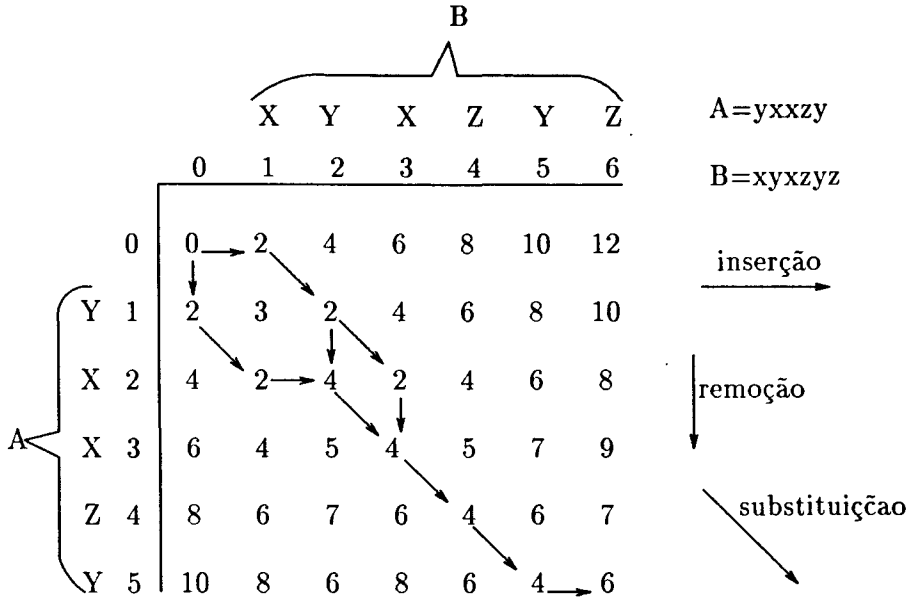

Figura 2.1: Matriz $D(i,j)$ com grafo de dependência e legenda dos arcos. O custo de uma inserção ou de uma remoção possui valor 2, o custo de uma substituição possui valor 3, se os caracteres forem distintos, 0 caso contrário.

Algoritmo Programação Dinâmica.
Entrada: Cadeias de caracteres A e B.
Saída: Matriz D.
Método: Programação Dinâmica.
$D(0,0) \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça**
    $D(i,0) \leftarrow D(i-1,0) + \delta(A(i) \rightarrow \epsilon)$
para $j \leftarrow 1$ **até** n **faça**
início
  $D(0,j) \leftarrow D(0,j-1) + \delta(\epsilon \rightarrow B(j))$
  para $i \leftarrow 1$ **até** m **faça**
  início
    $rem \leftarrow D(i-1,j) + \delta(A(i) \rightarrow \epsilon)$
    $ins \leftarrow D(i,j-1) + \delta(\epsilon \rightarrow B(j))$
    $sub \leftarrow D(i-1,j-1) + \delta(A(i) \rightarrow B(j))$
    $D(i,j) \leftarrow \min\{rem, ins, sub\}$
  fim
fim

Figura 2.2: Algoritmo *Programação Dinâmica.*

valor $D(i,j)$ é a soma dos rótulos de um caminho de $(0,0)$ a (i,j). Portanto, temos a seguinte propriedade, facilmente demonstrada por indução:

Proposição 2.1 *Se o grafo de dependência contém um caminho orientado de (i,j) a (i',j') então $D(i',j') = D(i,j) + s$, onde s representa a soma dos rótulos no caminho.* □

Observe que para calcular o valor $D(m,n)$, o valor que nos interessa, somente as entradas $D(i,j)$ pertencentes aos caminhos de $(0,0)$ a (m,n) são utilizados; mais ainda, nenhum $D(i,j) > D(m,n)$ será usado no cálculo de $D(m,n)$.

Dado um inteiro d, $-m \leq d \leq n$, a diagonal d, denotada por D_d, é o conjunto dos pares (i,j) tais que $0 \leq i \leq m$, $0 \leq j \leq n$ e $j - i = d$. Ou seja,

$$D_d := \{(i, i+d) : \max\{0,-d\} \leq i \leq \min\{m, n-d\}\}.$$

Uma outra observação importante é que se estamos no vértice (i,j) do grafo de dependência e se pelo passo de minimização dado por (2.1) for determinada uma substituição temos que a aresta determinada a partir de (i,j) mantém a diferença $d = j - i$ constante, ou seja, estamos caminhando em cima de uma diagonal.

Por outro lado, se desejamos caminhar do vértice (i,j) para o vértice (i',j') devemos atravessar um número de diagonais no mínimo igual a $|(j'-i') - (j-i)|$: esse valor indica a quantidade mínima de operações de inserção ou remoção necessárias para transformar $A(1 \ldots i')$ em $B(1 \ldots j')$ a partir de edições efetuadas em $A(1 \ldots i) \rightarrow B(1 \ldots j)$.

A partir de agora denotaremos por Δ o menor custo entre inserções e remoções. Em outras palavras,

$$\Delta := \min\{\delta(a \rightarrow \epsilon), \delta(\epsilon \rightarrow a)\},$$

onde $a \in \Sigma$. Temos então, a partir da observação acima, as seguintes propriedades:

Proposição 2.2 *Se o grafo de dependência contém um caminho orientado de (i,j) para (i',j') então $D(i',j') \geq D(i,j) + |(j'-i') - (j-i)| \cdot \Delta$.* □

Lema 2.3 *Se um caminho ótimo de $(0,0)$ a (m,n) cruza uma diagonal d então*

$$|d| \leq \frac{D(m,n)}{\Delta}.$$

Demonstração: Seja (i,j) um ponto do caminho na diagonal d. Pela Proposição 2.2 temos que

$$D(i,j) \geq D(0,0) + |d| \cdot \Delta = |d| \cdot \Delta$$

e, portanto, pela Proposição 2.1 temos que $D(m,n) \geq |d| \cdot \Delta$. □

Observe que este resultado limita o número de elementos a serem calculados a um conjunto de diagonais compreendidas entre as diagonais $-D(m,n)/\Delta$ e $D(m,n)/\Delta$, ou seja, o conjunto de diagonais a analisar consiste de $1 + 2 \cdot D(m,n)/\Delta$ diagonais.

Entretanto um conjunto menor, contendo $1+D(m,n)/\Delta$ diagonais, pode ser determinado:

Lema 2.4 *Se um caminho ótimo C de $(0,0)$ a (m,n) cruza uma diagonal d então*

$$-p + \min\{0, n-m\} \leq d \leq p + \max\{0, n-m\},$$

onde

$$p := \left\lfloor \frac{1}{2} \cdot \left(\frac{D(m,n)}{\Delta} - |n-m| \right) \right\rfloor$$

Demonstração: Sejam $d_0 := \min\{0, n-m\}$ e $d_1 := \max\{0, n-m\}$. Vamos demonstrar a desigualdade

$$d \leq p + d_1. \tag{2.2}$$

Observe inicialmente que, pelo Lema 2.3,

$$|n-m| \leq \frac{D(m,n)}{\Delta}$$

e, portanto,

$$p \geq 0.$$

Assim, a desigualdade (2.2) é imediata se $d \leq d_1$. Podemos, portanto, supor que

$$d > d_1.$$

Seja (i,j) um ponto de C na diagonal d. Dado que C se inicia na diagonal 0, e que $0 \leq d_1 < d$, então C cruza d_1, num ponto (i_1, j_1) anterior a (i,j) em C. Analogamente, C termina na diagonal $n-m$ e $n-m \leq d_1 < d$, portanto

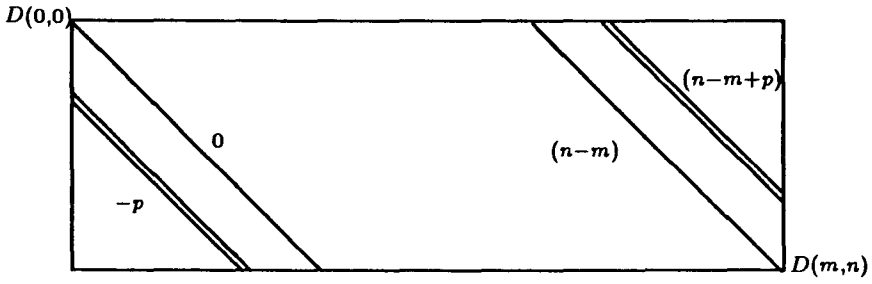

Figura 2.3: Diagonais $-p, 0, n-m$ e $n-m+p$ quando $m \leq n$.

$\mathcal{C}$ cruza novamente a diagonal d_1 num ponto (i_2, j_2) posterior a (i, j) em $\mathcal{C}$. Pela Proposição 2.2 temos que:

$$\begin{array}{rcl} D(m,n) & \geq & D(i_2,j_2) + |(n-m) - d_1| \cdot \Delta \\ D(i_2,j_2) & \geq & D(i,j) + |d - d_1| \cdot \Delta \\ D(i,j) & \geq & D(i_1,j_1) + |d - d_1| \cdot \Delta \\ D(i_1,j_1) & \geq & |d_1| \cdot \Delta. \end{array}$$

Somando as quatro desigualdades, simplificando e dividindo por Δ temos

$$\begin{array}{rcl} D(m,n)/\Delta & \geq & |(n-m) - d_1| + |d_1| + 2\,|d - d_1| \\ & \geq & |n-m| + 2(d - d_1), \end{array}$$

o que implica a desigualdade $d \leq p + d_1$.

A desigualdade $-p + d_0 \leq d$ é demonstrada de forma análoga. □

Observe que o Lema 2.4 estabelece uma faixa de diagonais: no caso em que $m \leq n$, uma das diagonais é obtida a partir da diagonal 0 caminhando-se p diagonais para baixo; a outra diagonal é obtida a partir da diagonal $n-m$ caminhando-se p diagonais para cima. Veja a Figura 2.3.

Dado um valor $t \geq 0$, podemos determinar se $D(m,n) \leq t$ utilizando o Lema 2.4. Para isso, vamos redefinir p da seguinte forma:

$$p := \left\lfloor \frac{1}{2} \cdot \left(\frac{t}{\Delta} - |n-m| \right) \right\rfloor.$$

Além disso, denominaremos de *faixa de diagonais*, denotada por $\mathrm{f}(t)$, o conjunto das diagonais d no intervalo

$$-p + \min\{0, n-m\} \leq d \leq p + \max\{0, n-m\}.$$

Algoritmo Teste. Hipótese: $t \geq |n-m|\,\Delta$.
Entrada: valor t a ser testado com $D(m,n)$.
Saída: *aceite o valor* t se $D(m,n) \leq t$ ou *rejeite o valor* t se $D(m,n) > t$.
Método: calcular os valores $C(i,j)$ através da recorrência (2.1) aplicada apenas a pontos dentro da faixa de diagonais.

$p \leftarrow \lfloor \frac{1}{2} \cdot \left(\frac{t}{\Delta} - |n-m|\right) \rfloor$
$d_0 \leftarrow \min\{0, n-m\}$
$d_1 \leftarrow \max\{0, n-m\}$
% $p \geq 0$
$C(0,0) \leftarrow 0$
para $j \leftarrow 0$ **até** $\min\{p+d_1, n\}$ **faça**
    $C(0,j) \leftarrow C(0,j-1) + \delta(\epsilon \rightarrow B(j))$
para $i \leftarrow 1$ **até** m **faça**
início
  para $d \leftarrow \max\{-i, -p+d_0\}$ **até** $\min\{n-i, p+d_1\}$ **faça**
  início
    $j \leftarrow i+d$
    % $0 \leq j \leq n$ e $-p+d_0 \leq d \leq p+d_1$
    se $d = p+d_1$ **então** $rem \leftarrow \infty$
    senão $rem \leftarrow C(i-1,j) + \delta(A(i) \rightarrow \epsilon)$
    se $j = 0$ **ou** $d = -p+d_0$ **então** $ins \leftarrow \infty$
    senão $ins \leftarrow C(i,j-1) + \delta(\epsilon \rightarrow B(j))$
    se $j = 0$ **então** $sub \leftarrow \infty$
    senão $sub \leftarrow C(i-1,j-1) + \delta(A(i) \rightarrow B(j))$
    $C(i,j) \leftarrow \min\{rem, ins, sub\}$
  fim
fim
se $C(m,n) \leq t$ **então** *aceite o valor* t
senão *rejeite o valor* t

Figura 2.4: Algoritmo *Teste*.

O próximo resultado mostra que para testar se $D(m,n) \leq t$, basta utilizar a recursão (2.1) nos pontos dentro da faixa de diagonais, como é feito pelo algoritmo *Teste* dado na Figura 2.4.

Lema 2.5 *Seja C a matriz calculada pelo algoritmo* Teste. *Então:*

1. *$C(i,j) \geq D(i,j)$ para qualquer $C(i,j)$ calculado;*
2. *para qualquer (i,j) pertencente a um caminho C, com origem em $(0,0)$, no grafo de dependências e dentro da faixa de diagonais $f(t)$, então $C(i,j) = D(i,j)$;*
3. *se $D(m,n) \leq t$ então $C(m,n) = D(m,n)$;*
4. *$D(m,n) \leq t$ se e somente se o algoritmo* Teste *aceita o valor t.*

Demonstração:

1. Para $i = 0$ o valor calculado para $C(0,j)$ coincide com o de $D(0,j)$. Suponha, portanto, que $i > 0$ e adote como hipótese de indução que a afirmação é verdadeira para todos os valores de C anteriormente calculados.

   Seja $d := i - j$. Vamos agora mostrar que *rem*, *ins* e *sub* são no mínimo iguais a $D(i,j)$, ou maiores. Para mostrar que

   $$rem \geq D(i,j),$$

   podemos obviamente supor que $rem < \infty$, o que implica que $d < p + d_1$. Assim o ponto $(i-1,j)$, que pertence à linha $i-1$ e à diagonal $d+1$, foi calculado anteriormente. Por hipótese de indução, $C(i-1,j) \geq D(i-1,j)$ e, portanto, $rem \geq D(i,j)$, por (2.1).

   Analogamente, se $ins < \infty$ então $j > 0$ e $d > -p + d_0$, o que implica que o ponto $(i,j-1)$, que pertence à diagonal $d-1$, foi calculado na iteração anterior. Por hipótese de indução $C(i,j-1) \geq D(i,j-1)$ o que implica que $ins \geq D(i,j)$, por (2.1).

   Finalmente, se $sub < \infty$ então $i > 0$, o que implica que o ponto $(i-1,j-1)$, que pertence à mesma diagonal d, foi calculado anteriormente. Por hipótese de indução, $C(i-1,j-1) \geq D(i-1,j-1)$, e portanto, $sub \geq D(i,j)$, por (2.1).

   Podemos então concluir que $\min\{rem, ins, sub\} \geq D(i,j)$.

2. Provemos por indução no comprimento do caminho $\mathcal{C}$. Para o comprimento de tamanho zero temos que a asserção é trivialmente verdadeira. Seja (i', j') o vértice imediatamente anterior a (i, j) em $\mathcal{C}$. Seja $\mathcal{C}'$ o trecho de $\mathcal{C}$ de $(0,0)$ a (i', j').

   Claramente temos que $\mathcal{C}' \subseteq \mathrm{f}(t)$, pois, $\mathcal{C}' \subseteq \mathcal{C}$. Agora, pela hipótese de indução, $C(i', j') = D(i', j')$.

   Por outro lado, pela recorrência (2.1), para um valor δ adequado, $D(i', j') + \delta = D(i, j)$ e pela especificação do algoritmo temos que $C(i, j) \leq C(i', j') + \delta$. Mas $C(i', j') = D(i', j')$, logo, $C(i, j) \leq D(i', j') + \delta = D(i, j)$. Portanto, pelo item 1, $C(i, j) = D(i, j)$.

3. Suponha que $\mathcal{C}$ seja um caminho ótimo de $(0,0)$ a (m, n), temos claramente que $\mathcal{C} \subseteq \mathrm{f}(t)$, pois, $\mathcal{C} \subseteq \mathrm{f}(D(m, n)) \subseteq \mathrm{f}(t)$. Logo, pelo item 2, temos que $C(m, n) = D(m, n)$.

4. Pelo item 3 temos que se $D(m, n) \leq t$ então $C(m, n) = D(m, n) \leq t$, logo, o algoritmo *Teste* aceita o valor t. Por outro lado, se o algoritmo *Teste* aceita o valor t então $C(m, n) \leq t$, mas, pelo item 1, temos então que $D(m, n) \leq t$. □

É possível determinar o valor $D(m, n)$ com auxílio do algoritmo *Teste*. Quando o algoritmo *Teste* aceitar um valor t isto significa que $C(m, n) = D(m, n)$, pelo Lema 2.5 (itens 3 e 4). Agora, se o algoritmo rejeitar o valor t podemos executar o algoritmo *Teste* com valores crescentes de t (sempre o dobro do valor, por exemplo) até que um valor seja aceito. Com esta metodologia obtemos o algoritmo *Distância*, mostrado na Figura 2.5.

2.2.2 Análise de Complexidade

É possível implementar o algoritmo *Teste* de forma que apenas duas linhas de C sejam armazenadas. Mediante uma indexação alternativa de cada linha, é possível utilizar, para cada linha, espaço proporcional a

$$1 + |n - m| + 2p \leq 1 + \frac{t}{\Delta}.$$

Assim, o algoritmo *Teste* tem complexidade em espaço

$$\mathrm{O}(\min\{n, \frac{t}{\Delta}\}).$$

Algoritmo Distância
Entrada: cadeias A e B com tamanhos m e n, respectivamente.
Saída: valor s da distância de edição entre A e B.
Método: testes com valores crescentes de t através do uso do algoritmo *Teste*. O primeiro valor para o teste de t é dado pelo valor mínimo de edição entre A e B, ou seja, a diferença de comprimento vezes o custo mínimo entre inserções e remoções (Δ). Os valores sucessivos de t a serem testados são sempre o dobro do anterior, isto se deve à maneira com que o parâmetro p é calculado.
$t \leftarrow (|n - m| + 1) \cdot \Delta$
enquanto *Teste*(t)=*rejeite o valor t* **faça** $t \leftarrow 2 \cdot t$
O valor da distância de edição é: $C(m, n)$

Figura 2.5: Algoritmo *Distância*.

O cálculo de cada elemento de uma linha é feito em tempo O(1). Logo, a complexidade em tempo do algoritmo é

$$O(m \cdot \min\{n, \frac{t}{\Delta}\}).$$

Para analisar a complexidade de tempo do algoritmo *Distância*, considere $t_0 = |n - m| \cdot \Delta + 1, t_1 = 2 \cdot t_0, \ldots t_r = 2^r \cdot t_0$, os valores de t usado como parâmetro do algoritmo *Teste*. De acordo com a análise do algoritmo *Teste*, o algoritmo *Distância* possui complexidade em tempo

$$O(m \cdot \sum_{i=0}^{r} \min\{n, \frac{t_i}{\Delta}\}),$$

ou seja, a complexidade de tempo é

$$O(m \cdot \min\{n \cdot \log t_r, \frac{t_r}{\Delta}\}).$$

Como s (custo de edição) é maior do que $\frac{t_r}{2}$ a complexidade de tempo é

$$O(m \cdot \min\{n \log s, \frac{s}{\Delta}\}).$$

O ganho ocorre quando $\min\{n \log s, \frac{s}{\Delta}\} = \frac{s}{\Delta}$.

Capítulo 3

Algoritmos para casamento aproximado com k diferenças

3.1 Introdução

A primeira variante do problema de casamento aproximado de padrões que estudaremos é conhecida como o *problema das k diferenças*. Dados um inteiro não negativo k, um *padrão* A de comprimento m e um *texto* B de comprimento n, ambos sobre um alfabeto Σ, este problema consiste em determinar todas as posições j do texto, $0 \leq j \leq n$, em que o padrão tem sua distância de edição no máximo k em relação a algum sufixo de $B(1 \ldots j)$.

Mais precisamente, para cada j, $0 \leq j \leq n$, podemos definir

$$\mathrm{f}(j) := \min\{\mathcal{E}(A(1 \ldots m), B(i \ldots j)) : 1 \leq i \leq j+1\}.$$

O problema das k diferenças consiste então em determinar o conjunto

$$\mathcal{J} := \{j : 0 \leq j \leq n \text{ e } \mathrm{f}(j) \leq k\}.$$

A rigor, o problema também exige que se conheça não só o conjunto $\mathcal{J}$, mas também, para cada $j \in \mathcal{J}$, o sufixo $B(i \ldots j)$ de $B(1 \ldots j)$ que determina $\mathrm{f}(j) \leq k$ e quais as operações de edição que devem ser feitas para levar $B(i \ldots j)$ em $A(1 \ldots m)$.

Na verdade, a obtenção dessas informações adicionais pode ser feita mediante alterações razoavelmente triviais nos algoritmos a serem apresentados neste capítulo. Limitar-nos-emos apenas a determinar o conjunto $\mathcal{J}$.

Neste capítulo assumiremos que cada operação de edição possui custo unitário, independente dos símbolos envolvidos. Além disso, modificaremos a definição da matriz D, da seguinte maneira: para cada i, j com $0 \leq i \leq m$ e $0 \leq j \leq n$,

$$D(i,j) := \min\{\mathcal{E}(A(1 \ldots i), B(h \ldots j)) : 0 \leq h \leq j+1\}.$$

Assim, a determinação do conjunto $\mathcal{J}$, que constitui o problema das k diferenças, pode ser feita facilmente:

$$\mathcal{J} := \{j : 0 \leq j \leq n \text{ e } D(m,j) \leq k\}.$$

Vamos agora fazer uma restrição importante nos valores de m, n e k, a saber,

$$m \leq n+k \text{ e } k < \frac{m}{2}.$$

Essa restrição é sempre viável, conforme veremos a seguir.

É fácil ver que se $m > n+k$ então o problema não tem solução, ou seja, $\mathcal{J} = \emptyset$. Por outro lado, se $k \geq m/2$ então $O(mn) = O(kn)$, ou seja, qualquer algoritmo ingênuo (prog. dinâmica) é $O(kn)$.

Neste capítulo descreveremos os algoritmos de Ukkonen [Ukk85], Galil e Park [GP90], Tarhio e Ukkonen [TU91].

A partir do algoritmo clássico da programação dinâmica, Ukkonen [Ukk85] desenvolve um algoritmo melhorado cuja complexidade de tempo é $O(mn)$ e complexidade de espaço é $O(n)$.

Galil e Park [GP90] melhoram a idéia de Ukkonen através do pré-processamento do padrão, idéia já utilizada no algoritmo de Knuth, Morris e Pratt [KMP77]. A complexidade do algoritmo de Galil e Park é $O(kn)$ para o processamento do texto, $O(m^2)$ para o pré-processamento do padrão e $O(m^2)$ para armazenar o resultado do pré-processamento do padrão.

Tarhio e Ukkonen [TU91] desenvolveram uma solução para o problema de k diferenças baseada na filosofia do algoritmo de Boyer e Moore [BM77] cuja complexidade de tempo é $O((n/k) \cdot m)$, no caso médio a complexidade de tempo é $O((c/(c-2k)) \cdot k \cdot n \cdot (k/(c+2k^2)) + 1/m)$, a ocupação de memória é da ordem de $O(cm)$ onde c é o tamanho do alfabeto.

A seguir apresentaremos o algoritmo de Ukkonen [Ukk85].

3.2 O algoritmo de Ukkonen

Como conseqüência da mudança da definição de D, temos uma alteração na recorrência (2.1) que agora representará as ocorrências do padrão no texto. Assim, $D(i,j)$ é:

$$\begin{cases} 0 & \text{se } i = 0 \\ i & \text{se } j = 0 \\ \min \begin{cases} D(i-1,\ j\ ) + 1 \\ D(i-1,j-1) + \text{ se } A(i) = B(j) \text{ então 0 senão 1} \\ D(\ i\ ,j-1) + 1 \end{cases} & \text{c. c.} \end{cases} \tag{3.1}$$

Mostraremos agora uma propriedade que será útil para o algoritmo.

Lema 3.1 *Para todo* (i,j), $1 \leq i \leq m$ *e* $1 \leq j \leq n$, $D(i,j) = D(i-1,j-1)$ *ou* $D(i,j) = D(i-1,j-1)+1$.

Demonstração: Como $D(i,j)$ é sempre um inteiro então é suficiente provar que

$$D(i-1,j-1) \leq D(i,j) \leq D(i-1,j-1)+1.$$

A segunda desigualdade é facilmente verificada a partir de (3.1).

A primeira desigualdade será demonstrada por indução em $i+j$. Se $i = 1$ então

$$D(i-1,j-1) = 0 \leq D(i,j).$$

Podemos, portanto, supor que

$$i > 1.$$

Podemos também supor, obviamente, que

$$D(i,j) \neq D(i-1,j-1)+1. \tag{3.2}$$

Considere inicialmente o caso em que

$$D(i,j-1)+1 = D(i,j). \tag{3.3}$$

Neste caso, $j > 1$ pois se $j = 1$ então, por (3.1)

$$D(i,j-1) = i = 1 + D(i-1,j-1) \geq D(i,j),$$

uma contradição. De fato, neste caso $j \geq 2$.

Por hipótese de indução,

$$D(i-1, j-2) \leq D(i, j-1). \tag{3.4}$$

Ademais, por (3.1),

$$D(i-1, j-1) \leq D(i-1, j-2) + 1. \tag{3.5}$$

Somando (3.3), (3.4) e (3.5), obtemos a desigualdade.

Resta considerar agora o caso em que

$$D(i, j-1) + 1 \neq D(i, j). \tag{3.6}$$

De (3.2) e (3.6), por (3.1),

$$D(i-1, j) + 1 = D(i, j).$$

Por hipótese de indução, e por (3.1), respectivamente,

$$\begin{aligned} D(i-2, j-1) &\leq D(i-1, j) \\ D(i-1, j-1) &\leq D(i-2, j-1) + 1. \end{aligned}$$

Somando as três últimas desigualdades obtemos o resultado. □

A Proposição 2.1 e o Lema 3.1 sugerem uma forma alternativa para obter os valores de D, conforme veremos a seguir.

A diagonal d, $-m \leq d \leq n$, foi definida como sendo o conjunto dos pares $(i, i+d)$ tais que $0 \leq i \leq m$ e $0 \leq i + d \leq n$. Assim, o menor i tal que $(i, i+d)$ está na diagonal d é

$$i_d := \max\{0, -d\}$$

e o maior i tal que $(i, i+d)$ está na diagonal d é

$$f_d := \min\{m, n-d\}.$$

Sejam

$$\min D_d := \min\{D(i, i+d) : i_d \leq i \leq f_d\}$$

e

$$\max D_d := \max\{D(i, i+d) : i_d \leq i \leq f_d\}.$$

Pela Proposição 2.1 e por (3.1),

$$\min D_d = D(i_d, i_d + d) = \max\{0, -d\} = i_d \text{ e}$$

$$\max D_d = D(f_d, f_d + d).$$

Proposição 3.2 *Para j no intervalo $0 \leq j < m - k$, temos que $D(m, j) > k$.*

Demonstração: Seja $d := j - m$. Então $-m \leq d < -k$. Portanto, d é uma diagonal e $\min D_d = \max\{0, -d\} > k$. Pelo Lema 3.1, $D(m, j) > k$. □

Proposição 3.3 *A seguinte afirmação é verdadeira para toda diagonal $d < n$:*

$$|\max D_d - \max D_{d+1}| \leq 1.$$

Demonstração: Para demonstrar essa desigualdade consideraremos dois casos, a depender de d. Dado que $\max D_d = D(f_d, f_d + d)$ e que

$$f_d = \min\{m, n - d\},$$

convém conduzir a análise em dois casos, a depender do resultado da comparação entre m e $n - d$.

Vamos considerar inicialmente o caso em que $m < n - d$. Temos então que $f_d = m = f_{d+1}$. Assim,

$$\max D_{d+1} = D(m, m + d + 1)$$

e

$$\max D_d = D(m, m + d).$$

De (3.1), temos que

$$\max D_{d+1} \leq 1 + \max D_d. \tag{3.7}$$

De (3.1) e do Lema 3.1, temos que

$$\max D_d \leq 1 + D(m - 1, m - 1 + d + 1) \leq 1 + \max D_{d+1}. \tag{3.8}$$

De (3.7) e (3.8) segue o resultado.

Resta agora considerar o caso em que $m \geq n - d$. Analogamente, temos que $f_d = n - d = 1 + f_{d+1}$. Portanto,

$$\max D_{d+1} = D(n - (d + 1), n)$$

e

$$\max D_d = D(n - d, n).$$

		0	1	2	3	4
			x	x	z	y
0		0	0	0	0	0
1	y	1	1	1	1	0
2	x	2	1	1	2	1
3	x	3	2	1	2	2
4	y	4	3	2	2	2

Figura 3.1: Matriz D com custos unitários para as cadeias yxxy e xxzy.

De (3.1), temos que

$$\max D_d \leq 1 + \max D_{d+1}.$$

De (3.1) e do Lema 3.1, temos que

$$\max D_{d+1} \leq 1 + D(n-1+d+1, n-1) \leq 1 + \max D_d.$$

Também neste caso vale a desigualdade. □

Para cada diagonal d, é suficiente conhecer os valores de

$$M_{d,v} := \max\{i : D(i, i+d) = v,\ i_d \leq i \leq f_d\}$$

para os valores de v tais que

$$\min D_d \leq v \leq \max D_d.$$

Um exemplo para as cadeias yxxy e xxzy é mostrado na Figura 3.1. Nesta matriz D temos que $M_{-1,1} = 3, M_{0,1} = 2$ e $M_{0,2} = 4$.

Podemos calcular o valor $M_{d,v}$ recursivamente da seguinte forma. Suponha que os valores $M_{d-1,v-1}$, $M_{d,v-1}$ e $M_{d+1,v-1}$ já foram calculados.

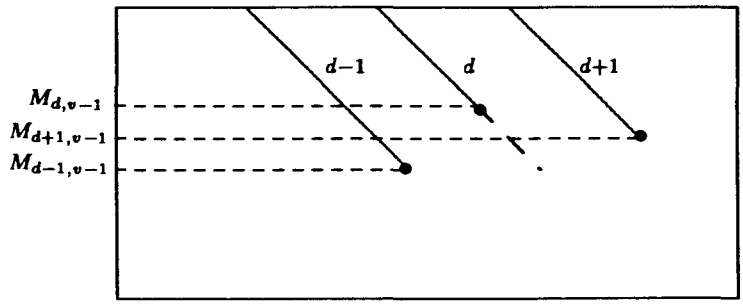

Figura 3.2: Ilustração da igualdade (3.9).

Com estes três valores temos a informação dos locais onde o valor $v-1$ ocorre como último valor nas diagonais $d-1$, d e $d+1$, respectivamente.

Pela propriedade de crescimento uniforme demonstrada pelo Lema 3.1 e pelas informações de M, conforme veremos a seguir podemos encontrar um ponto na diagonal d onde existe o valor v.

De fato, o maior valor dentre $M_{d-1,v-1}$, $M_{d,v-1}$ e $M_{d+1,v-1}$ informa um ponto na diagonal d onde o valor v ocorre, pois, cada um desses valores indica, em suas respectivas diagonais, a última ocorrência de $v-1$. Logo, pelo Lema 3.1 e pela recorrência 3.1, v ocorre na posição

$$t := \max\{M_{d-1,v-1}, M_{d,v-1}+1, M_{d+1,v-1}+1\}. \tag{3.9}$$

Veja a Figura 3.2.

Assim, para calcular o valor $M_{d,v}$ basta fazer comparações entre os caracteres de A e de B a partir das posições $t+1$ e $t+d+1$, respectivamente. Enquanto persistirem igualdades, os valores na diagonal d permanecerão constantes. O último valor de uma seqüência de igualdades será o valor de $M_{d,v}$.

Ilustramos estas idéias no algoritmo *CalculaM*, Figura 3.3. Os valores t_0, t_1 e t_2 indicam a última ocorrência de $v-1$ nas diagonais d, $d-1$ e $d+1$, respectivamente. As restrições impostas na definição daqueles valores devem-se ao problema de limitar os pontos dentro da matriz D. Por exemplo, t_1 só assumirá o valor i_d quando $d=-m$ ou quando $M_{d-1,v-1}$ não fizer sentido, ou seja, $v-1 < \min D_{d-1}$.

Algoritmo CalculaM. Hipótese:$-m \leq d \leq n$ e $\min D_d \leq v \leq \max D_d$.
Entrada: Padrão A e texto B de comprimento m e n, respectivamente.
Saída: Valor $M_{d,v}$.

$$t_0 \leftarrow \begin{cases} i_d & \text{se } v = \min D_d \\ M_{d,v-1} + 1 & \text{caso contrário} \end{cases}$$
$$t_1 \leftarrow \begin{cases} i_d & \text{se } d = -m \text{ ou } v \leq \min D_{d-1} \\ M_{d-1,v-1} & \text{caso contrário} \end{cases}$$
$$t_2 \leftarrow \begin{cases} i_d & \text{se } d = n \text{ ou } v \leq \min D_{d+1} \\ M_{d+1,v-1} + 1 & \text{caso contrário} \end{cases}$$
$t \leftarrow \max\{t_0, t_1, t_2\}$
$l_0 \leftarrow \min\{t, f_d\}$
$l \leftarrow l_0 + 1$
enquanto $l \leq f_d$ **e** $A(l) = B(l+d)$ **faça** $l \leftarrow l + 1$ $\ll 1 \gg$
$M_{d,v} \leftarrow l - 1$

Figura 3.3: Algoritmo *CalculaM*.

Proposição 3.4 *No algoritmo CalculaM, o valor t_0 está bem definido, $i_d \leq t_0 \leq f_d$ e $D(t_0, t_0 + d) = v$.*

Demonstração: Vamos inicialmente considerar o caso em que $\min D_d = v$. Neste caso, por definição de t_0, temos que

$$i_d = t_0 \leq f_d.$$

Além disso,

$$D(t_0, t_0 + d) = D(i_d, i_d + d) = \min D_d = v.$$

Resta agora analisar o caso em que $v \neq \min D_d$. Pela hipótese do algoritmo, $\min D_d < v \leq \max D_d$ e, portanto,

$$i_d \leq M_{d,v-1} < f_d.$$

Por definição de t_0, $t_0 = 1 + M_{d,v-1}$. Assim,

$$i_d < t_0 \leq f_d.$$

Além disso, pelo Lema 3.1,

$$D(t_0, t_0 + d) = 1 + D(M_{d,v-1}, M_{d,v-1} + d) = v. \square$$

Proposição 3.5 *No algoritmo CalculaM, o valor t_1 está bem definido. Mais precisamente, se $v \neq -m$ e $v > \min D_{d-1}$ então*

$$\min D_{d-1} \leq v - 1 \leq \max D_{d-1}.$$

Demonstração: Pela hipótese do algoritmo, $-m \leq d \leq n$. Por hipótese, $d \neq -m$. Assim, $d - 1$ está no intervalo $-m \leq d - 1 < n$.

A desigualdade $\min D_{d-1} \leq v - 1$ equivale à hipótese $v > \min D_{d-1}$ da Proposição.

A desigualdade $v - 1 \leq \max D_{d-1}$ segue da hipótese $v \leq \max D_d$ do algoritmo *CalculaM*, pela Proposição 3.3. $\square$

A demonstração da próxima Proposição é análoga à da Proposição 3.5.

Proposição 3.6 *No algoritmo CalculaM, o valor t_2 está bem definido. Mais precisamente, se $v \neq n$ e $v > \min D_{d+1}$ então*

$$\min D_{d+1} \leq v - 1 \leq \max D_{d+1}.\square$$

Proposição 3.7 *No algoritmo CalculaM, temos que $i_d \leq l_0 \leq f_d$ e $D(l_0, l_0 + d) = v$*

Demonstração: Se $l_0 = f_d$ então $t_0 \leq l_0$, pela Proposição 3.4; se $l_0 = t$ então $t_0 \leq l_0$ pela definição de t. Em ambos os casos, $t_0 \leq l_0$. Pela Proposição 3.4 e pela definição de l_0, temos então que

$$i_d \leq t_0 \leq l_0 \leq f_d,$$

o que implica a primeira parte da asserção. Além disso, pela Proposição 3.4 e pelo Lema 3.1,

$$v = D(t_0, t_0 + d) \leq D(l_0, l_0 + d).$$

Ademais, se $t = t_0$ então $l_0 = t = t_0$ e portanto $D(l_0, l_0 + d) = v$.

Resta, portanto, mostrar que

$$D(l_0, l_0 + d) \leq v,$$

supondo que $t > t_0$.

Assim, temos que $t \in \{t_1, t_2\}$. Analisaremos apenas o caso em que $t = t_1$. O caso em que $t = t_2$ é análogo.

Dado que $t_0 < t$, então $i_d < t$. Lembrando que $t = t_1$, temos que $i_d < t_1$. Pela definição de t_1, temos que

$$t_1 = M_{d-1,v-1}. \tag{3.10}$$

Portanto,

$$D(t, t + d - 1) = v - 1. \tag{3.11}$$

Se $t \leq f_d$, então $l_0 = t$, logo de (3.1),

$$D(l_0, l_0 + d) \leq 1 + D(t, t + d - 1) = v.$$

Por outro lado, se $t > f_d$ então

$$t_1 = t \geq 1 + f_d = 1 + \min\{m, n - d\} \geq \min\{m, n - (d - 1)\} = f_{d-1}.$$

Nesse caso, de (3.10), $t = f_{d-1}$. Logo de (3.11) temos que $\max D_{d-1} = v-1$. Portanto, pela Proposição 3.3,

$$D(l_0, l_0 + d) \leq \max D_d \leq 1 + \max D_{d-1} = v.$$

Em ambos os casos temos que

$$D(l_0, l_0 + d) \leq v,$$

o que completa a demonstração da igualdade $D(l_0, l_0 + d) = v$. □

Lema 3.8 *O algoritmo CalculaM determina corretamente o valor* $M_{d,v}$.

Demonstração: Pela especificação do algoritmo, é calculado o maior valor l tal que

$$l_0 \leq l \leq f_d \tag{3.12}$$

e

$$A(l_0 + 1 \ldots l) = B(l_0 + 1 + d \ldots l + d). \tag{3.13}$$

Pela Proposição 3.7,

$$D(l_0, l_0 + d) = v. \tag{3.14}$$

Assim, de (3.1), (3.13) e (3.14)

$$D(l, l + d) = D(l_0, l_0 + d) = v. \tag{3.15}$$

Se $l = f_d$ então certamente l é o maior valor que satisfaz (3.12) e (3.15) e portanto $l = M_{d,v}$.

Suponhamos então que $l \neq f_d$. Nesse caso,

$$l < f_d \text{ e } A(l+1) \neq B(l+1+d). \tag{3.16}$$

Resta mostrar que

$$D(l+1, l+1+d) = v+1.$$

Em vista de (3.15) e (3.16) por (3.1) isto acontece a menos que

$$\text{ou } -m < d \text{ e } D(l+1, l+d) \leq v-1 \tag{3.17}$$

$$\text{ou } d < n \text{ e } D(l, l+d+1) \leq v-1. \tag{3.18}$$

Vamos mostrar que tanto (3.17) quanto (3.18) são falsas, estabelecendo assim a correção de *CalculaM*.

Dado que $l < f_d$, então $l_0 < f_d$. Por definição de l_0, temos então que $l_0 = t$. Assim,

$$\max\{t_1, t_2\} \leq t = l_0 \leq l < f_d. \tag{3.19}$$

Suponha, por absurdo, que (3.17) vale. Nesse caso, $-m < d$ e

$$\min D_{d-1} \leq v - 1 < v.$$

Por definição,

$$t_1 = M_{d-1,v-1}.$$

Desta igualdade e de (3.19) temos que

$$M_{d-1,v-1} < l + 1,$$

o que contradiz (3.17). De fato, (3.17) é falsa.

Suponhamos agora, por absurdo, que (3.18) vale. Nesse caso, $d < n$ e

$$\min D_{d+1} \leq v - 1 < v.$$

Por definição,

$$t_2 = 1 + M_{d+1,v-1}.$$

Desta igualdade e de (3.19) temos que

$$M_{d+1,v-1} < l,$$

o que contradiz (3.18). De fato, (3.17) e (3.18) são falsas. □

Os valores $M_{d,v}$ podem então ser calculados pelo algoritmo *Diferenças-UK*, Figura 3.4. A variável C_d indica se a diagonal d foi totalmente analisada.

Para podermos demonstrar a correção e a complexidade do algoritmo *Diferenças-UK* vamos inicialmente demonstrar duas propriedades óbvias a respeito do algoritmo.

A primeira propriedade explicita qual o conjunto dos pares (d, v) que o algoritmo analisa. A segunda propriedade é relativa à ordem em que os pares (d, v) são analisados.

Algoritmo Diferenças-UK.
Entrada: Padrão A e texto B de tamanhos m e n, respectivamente; $k \leq m \leq n + k$.
Saída: Valores $M_{d,v}$.
Método: Cálculo dos valores $M_{d,v}$ através do algoritmo *CalculaM*.

```
para d ← −k até n − m + k faça C_d ← verdadeiro
para v ← 0 até k faça
início
   para d ← −v até n − m + k − v faça
   início                                          ≪ 1 ≫
      se C_d então
      início
         CalculaM (d, v)                           ≪ 2 ≫
         C_d ← M_{d,v} < f_d
      fim
      ≪ 3 ≫
   fim
fim
```

Figura 3.4: Algoritmo *Diferenças-UK*.

Proposição 3.9 *No algoritmo Diferenças-UK, são analisados os pares de valores* (d, v) *tais que*

$$-k \leq d \leq n - m + k$$

e

$$\min D_d \leq v \leq \min\{k, n - m + k - d\}.$$

Demonstração: São analisados os pares (d, v) tais que

$$0 \leq v \leq k \tag{3.20}$$

e

$$-v \leq d \leq n - m + k - v. \tag{3.21}$$

Somando $v - d$ aos termos de (3.21) temos então que são analisados os pares (d, v) tais que

$$0 \leq v \leq k$$

e

$$-d \leq v \leq n - m + k - d.$$

Ou, de forma equivalente, são analisados os pares (d, v) tais que

$$\max\{0, -d\} \leq v \leq \min\{k, n - m + k - d\}. \tag{3.22}$$

O intervalo (3.22) é não vazio se e somente se as seguintes desigualdades são válidas:

$$0 \leq k \tag{3.23}$$

$$-d \leq k \tag{3.24}$$

$$0 \leq n - m + k - d \tag{3.25}$$

$$-d \leq n - m + k - d. \tag{3.26}$$

A desigualdade (3.23) é válida por hipótese. A desigualdade (3.26) equivale à desigualdade $m \leq n + k$, válida por hipótese. Assim de (3.24) e (3.25) o intervalo (3.22) é não vazio se e somente se

$$-k \leq d \leq n - m + k. \tag{3.27}$$

De fato, os pares (d, v) analisados são precisamente aqueles que satisfazem (3.22) e (3.27). □

Algoritmo Diferença-UK-Normal.
Entrada: Padrão A e texto B de tamanhos m e n, respectivamente; $k \leq m \leq n + k$.
Saída: Valores $M_{d,v}$.
Método: Cálculo dos valores $M_{d,v}$ através do algoritmo *CalculaM*.
para $d \leftarrow -k$ **até** $n - m + k$ **faça** $C_d \leftarrow$ *verdadeiro*
para cada (d, v) **tal que**
$-k \leq d \leq n - m + k$ **e** $\min D_d \leq v \leq \min\{k, n - m + k - d\}$,
em ordem normal **faça**
início $\ll 1 \gg$
  se C_d **então**
  início
    CalculaM (d, v) $\ll 2 \gg$
    $C_d \leftarrow M_{d,v} < f_d$
  fim
  $\ll 3 \gg$
fim

Figura 3.5: Algoritmo *Diferença-UK-Normal*.

Proposição 3.10 *Se os pares* (d, v) *e* $(x, v - 1)$, *com* $d - 1 \leq x \leq d + 1$, *são ambos analisados pelo algoritmo Diferenças-UK, então a análise do par* $(x, v - 1)$ *precede a do par* (d, v). □

Uma ordem parcial $\preceq$ sobre $\mathcal{Z}^2$ é *normal* se para quaisquer dois pares $(d', v - 1)$ e (d, v), tais que $d - 1 \leq d' \leq d + 1$ temos que $(d', v - 1) \prec (d, v)$. Assim, $\preceq$ denota a ordem de análise dos pares (d, v) no algoritmo *Diferenças-UK*, a Proposição 3.10 afirma que essa ordem é normal.

Com base nessa observação, vamos interromper a análise de *Diferenças-UK* e propor um novo algoritmo, *Diferença-UK-Normal*, Figura 3.5. Note que o algoritmo num certo sentido generaliza o algoritmo *Diferenças-UK*. Assim, na verdade vamos demonstrar a correção do novo algoritmo.

Proposição 3.11 *No algoritmo Diferença-UK-Normal, as seguintes afirmações são verdadeiras:*

1. *no ponto* $\ll 2 \gg$, *as hipóteses de CalculaM para o cálculo de* $M_{d,v}$ *são satisfeitas e os valores porventura necessários de* $M_{x,v-1}$, $d-1 \leq x \leq d+1$, *já foram corretamente calculados;*

2. *no ponto* $\ll 3 \gg$, $C_d = (v < \max D_d)$;

3. *no ponto* $\ll 3 \gg$, *os valores* $M_{d,y} : \min D_d \leq y \leq \min\{v, \max D_d\}$ *já foram corretamente calculados.*

Demonstração: A demonstração será feita por indução.

1. Pela especificação do algoritmo,

$$-k \leq d \leq n - m + k \tag{3.28}$$

e $\min D_d \leq v$. Para provar que as hipóteses de *CalculaM* são satisfeitas devemos mostrar que $-m \leq d \leq n$ e que $v \leq \max D_d$.

De (3.28) e pelo fato de que $k < m/2$, temos trivialmente que $-m \leq d \leq n$.

Resta provar que $v \leq \max D_d$. Esta desigualdade vale trivialmente se $v = \min D_d$. Podemos portanto supor que $v > \min D_d$. Dado que a ordem de análise é normal, então houve uma passagem anterior pelo ponto $\ll 3 \gg$ em que a diagonal d foi analisada e a variável v assumiu o valor $v' := v - 1$, na verdade, essa foi a última passagem em que d foi analisada. Pela hipótese de indução em (2), $C_d = (v' < \max D_d)$, ou seja, $C_d = (v \leq \max D_d)$. Mas como estamos no ponto $\ll 2 \gg$, C_d é verdadeiro. Logo, $v \leq \max D_d$. De fato as hipóteses do algoritmo *CalculaM* são satisfeitas.

Vamos agora mostrar que os valores porventura necessários de $M_{x,v-1}$, $d-1 \leq x \leq d+1$, já foram corretamente calculados. Suponha que $M_{x,v-1}$ é necessário.

Para $x = d$, da primeira parte e da especificação de *CalculaM*, temos que

$$\min D_x < v \leq \max D_x.$$

Para $x = d-1$ ou $x = d+1$, pelas Proposições 3.5 e 3.6 e a especificação de *CalculaM*, temos que

$$\min D_x \leq v - 1 \leq \max D_x.$$

Portanto, para qualquer um dos três valores de x,

$$\min D_x \leq v - 1 \leq maxD_x. \tag{3.29}$$

Por outro lado, pela especificação do algoritmo,

$$v - 1 \leq \min\{k - 1, n - m + k - d - 1\}.$$

Dado que $x \leq d + 1$, segue que

$$v - 1 \leq \min\{k - 1, n - m + k - x\}. \tag{3.30}$$

De (3.29) e (3.30),

$$\min D_x \leq v - 1 \leq \min\{k - 1, n - m + k - x, \max D_x\}. \tag{3.31}$$

Finalmente, lembrando que $\min D_x = \max\{0, -x\}$, temos, de (3.31),

$$-k < x \leq n - m + k.$$

Portanto, a diagonal x é analisada e o valor $v - 1$ está no intervalo dos valores de v considerados para a diagonal. Além disso, $\min D_x \leq v - 1 \leq \max D_x$. Dado que a ordem de análise é normal, houve uma passagem anterior pelo ponto $\ll 3 \gg$, com d valendo x e v valendo $v - 1$. Por indução em (3), $M_{x,v-1}$ foi calculado anteriormente.

2. Se o algoritmo executou a chamada *CalculaM*(d, v) em $\ll 2 \gg$, então, por hipótese de indução em (1) e pelo Lema 3.8, C_d recebeu o valor $(M_{d,v} < f_d) = (v < \max D_d)$. Podemos então supor que a chamada não foi executada. Ou seja, que C_d é falso em $\ll 1 \gg$.

   Mas, C_d foi inicializado verdadeiro. Portanto, a diagonal d já foi analisada anteriormente. Dado que a ordem é normal, o último valor usado foi $v - 1$ no lugar de v. Por indução em (2) temos que $v - 1 \geq \max D_d$. Portanto, $v < \max D_d$ também é falso.

3. Suponhamos inicialmente que $v = \min D_d$. Dado que a ordem é normal, então esta é a primeira vez que a diagonal d é considerada e, como C_d foi inicializado verdadeiro, o ponto $\ll 2 \gg$ foi executado. Logo, por indução em (1), $M_{d,v}$ foi calculado. Assim, (3) vale.

   Vamos agora considerar o caso em que $\min D_d < v$. Então a diagonal d foi analisada anteriormente, e o último valor usado para v foi $v-1$. Por indução em (3), os valores $M_{d,y}$ foram corretamente calculados, para

   $$\min D_d \leq y \leq \min\{v-1, \max D_d\}.$$

   Por indução em (2), $C_d = (v-1 < \max D_d) = (v \leq \max D_d)$

   Se C_d era verdadeiro, isto é, se $v \leq \max D_d$, então $M_{d,v}$ foi calculado corretamente na iteração corrente. Se C_d era falso, então $v > \max D_d$. Em ambos os casos, todos os valores $M_{d,y}$ foram calculados corretamente, para $\min D_d \leq y \leq \min\{v, \max D_d\}$. □

Proposição 3.12 *Seja j uma coluna da matriz D; seja $d := j - m$. Se $j < m - k$ então $D(m,j) > k$. Por outro lado, se $j \geq m - k$ então a diagonal d é analisada pelo algoritmo Diferença-UK-Normal e ao final da sua execução as seguintes afirmações são verdadeiras:*

1. $C_d = (D(m,j) > k)$;
2. *se $D(m,j) \leq k$ então $m = M_{d,D(m,j)}$ foi corretamente calculado.*

Demonstração: Se $j < m - k$ então $D(m,j) > k$, pela Proposição 3.2. Suponha então que $m - k \leq j \leq n$ e, portanto,

$$-k \leq d \leq n - m \leq n - m + k.$$

Logo, d é analisada pelo algoritmo.

Vamos considerar a última passagem do algoritmo pelo ponto $\ll 3 \gg$, quando a diagonal d estiver sendo analisada. Lembrando que $d \leq n - m$, temos que,

$$n - m + k - d \geq k.$$

Assim, pela especificação do algoritmo, temos que $v = k$.

Pela Proposição 3.11(2), $C_d = (k < \max D_d)$, ou seja, $C_d = (D(m,j) > k)$.

Suponhamos ainda que $D(m,j) \leq k$. Ou seja, $\max D_d = \min\{k, \max D_d\}$ e $m = M_{d,D(m,j)}$. Pela Proposição 3.11(3), o valor $M_{d,D(m,j)}$ foi corretamente calculado. □

Corolário 3.13 *O algoritmo Diferença-UK-Normal resolve o problema das k diferenças.* □

Corolário 3.14 *O algoritmo Diferenças-UK resolve o problema das k diferenças.* □

Proposição 3.15 *Se a complexidade de determinação da ordem for linear no número de pares analisados então a complexidade do algoritmo Diferença-UK-Normal é $O(mn)$.*

Demonstração: O custo global das comparações com igualdade na linha $\ll 1 \gg$ de *CalculaM*(d,v) para uma diagonal fixa d é obviamente $O(m)$. Assim, o custo global das comparações com igualdade na linha $\ll 1 \gg$ de *CalculaM* é

$$O((n-m+2k)m) = O(mn), \tag{3.32}$$

pois, $n-m+2k+1$ diagonais são analisadas.

Para uma dada diagonal d, são analisados no máximo $k+1$ valores de v. Assim, por hipótese, o custo da enumeração dos pares é $O(k(n-m+2k)) = O(kn)$.

O custo das partes restantes, por par analisado, é $O(1)$. Assim, o custo global das partes restantes também é $O(kn)$.

Portanto a complexidade total do algoritmo é $O(mn)$. □

Corolário 3.16 *A complexidade do algoritmo Diferenças-UK é $O(mn)$.* □

Agora vamos analisar a complexidade de espaço do algoritmo *Diferenças-UK*. Para o cálculo de M, relativos a um determinado v são necessários apenas os valore de M relativos a $v-1$. Assim, basta um vetor com $O(n)$ posições, dado que o número de diagonais é $n-m+2k+1$. Para armazenar os valores $\max D_d$, outro vetor de mesmo tamanho é suficiente. Assim, a complexidade em espaço é $O(n)$.

Terminaremos esta seção com duas observações importantes.

Vale ressaltar que o algoritmo apresentado aqui possui algumas diferenças em relação ao algoritmo original de Ukkonen onde o autor considera

o problema de transformar uma seqüência A em uma outra seqüência B, ou seja, a recorrência usada é a da equação (2.1) com custos unitários.

A complexidade do tempo de execução do algoritmo *Diferenças-UK* não é melhor do que a do algoritmo padrão de programação dinâmica, e no entanto é mais complicado. A razão de apresentarmos este algoritmo é preparar a apresentação do algoritmo de Galil e Park, para próxima seção, que fará um uso mais eficiente de *CalculaM*. Mais especificamente a parcela (3.32) da complexidade é reduzida para $O(kn)$, fazendo com que essa seja a complexidade total do algoritmo de Galil e Park. Para tanto, a ordem de enumeração, embora normal, será distinta da enumeração do algoritmo *Diferenças-UK*.

3.3 Algoritmo de Galil e Park

3.3.1 Introdução

O algoritmo descrito nesta seção apresenta uma solução para o problema das k diferenças. Assim como no algoritmo de Ukkonen os custos das operações de edição são unitários. Além disso, o algoritmo de Galil e Park também utilizará a propriedade do crescimento monotônico dos valores em uma diagonal. Vale ressaltar que esse algoritmo é melhor que o algoritmo de Ukkonen, pois algumas comparações entre o texto e o padrão são evitadas.

A idéia principal é diminuir o número de comparações realizadas no algoritmo *CalculaM* para o cômputo dos valores $M_{d,v}$, mais especificamente, diminuir as comparações realizadas no ponto $\ll 1 \gg$ indicado na figura 3.3 (página 36).

Conforme foi visto ao final da seção anterior, o custo global dessas comparações onde um caracter de A é igual a um caracter de B é $O(mn)$, ao passo que o custo do restante do algoritmo é $O(kn)$.

A contribuição do algoritmo de Galil e Park é a de baixar o custo global das comparações com igualdade, baixando-o para $O(kn)$.

Para conseguir esse objetivo, o algoritmo registra as igualdades obtidas, para uso posterior, em *triplas de referência*, conforme veremos a seguir.

Para poder fazer uso dessa informação, é realizada uma pré-computação no padrão cujo custo é $O(m^2)$. Nesta pré-computação é obtida uma tabela denotada Pref, conforme será visto na subseção 3.3.3.

Finalmente, a ordem de cálculo, ainda normal, é diferente daquela usada em *Diferenças-UK*.

3.3.2 Triplas de referência

Uma tripla de inteiros $\tau := (p, q, d)$ é denominada uma *tripla de referência* se as seguintes propriedades forem satisfeitas:

$$-m \leq d \leq n \tag{3.33}$$

$$i_d \leq p - 1 \leq q \leq f_d \tag{3.34}$$

$$A(p \ldots q) = B(p + d \ldots q + d) \tag{3.35}$$

$$\text{se } p \leq q < f_d \text{ então } A(q + 1) \neq B(q + 1 + d). \tag{3.36}$$

Assim, se τ é uma tripla de referência então d denota uma diagonal, $A(p \ldots q)$ e $B(p+d \ldots q+d)$ são segmentos iguais no padrão e no texto, ambos de comprimento $q-(p-1) \geq 0$. Esses segmentos são maximais à direita, ou seja, não é válida a igualdade com $q+1$ no lugar de q por uma de duas razões: ou por condições de contorno ($q = f_d$), ou porque os caracteres $A(q+1)$ e $B(q+1+d)$ são distintos.

Convém ressaltar ainda que esses segmentos podem ser vazios, com $p = q+1$.

Proposição 3.17 *Ao final do algoritmo CalculaM, $(l_0+1, M_{d,v}, d)$ é uma tripla de referência.*

Demonstração: Pela hipótese do algoritmo *CalculaM*, d é uma diagonal, que satisfaz a desigualdade (3.33).

Pela Proposição 3.7,

$$i_d \leq l_0.$$

Pela especificação de *CalculaM*,

$$l_0 \leq M_{d,v} \leq f_d.$$

Assim a tripla satisfaz (3.34), com l_0+1 no lugar de p e $M_{d,v}$ no lugar de q.

Finalmente, pela especificação de *CalculaM*, (3.35) e (3.36) também são satisfeitas. □

Proposição 3.18 *A tripla $(1,0,0)$ é de referência.* □

3.3.3 A tabela Pref

A tripla de referência obtida no algoritmo *CalculaM* (Proposição 3.17) como tal não tem utilidade pois os pares de caracteres iguais $A(i)$ e $B(i+d)$, $p \leq i \leq q$ jamais serão comparados novamente. No entanto, $B(i+d)$ será comparado com outros caracteres do padrão, em outras posições $j \neq i$. Nesse caso, se soubermos de antemão o resultado da comparação entre $A(i)$ e $A(j)$ podemos determinar previamente o resultado da comparação entre $A(j)$ e $B(i+d)$.

O mesmo raciocínio pode então ser estendido a todo um segmento do texto B, não apenas ao caracter $B(i+d)$, que esteja incluído no segmento $B(p+d \ldots q+d)$.

Algoritmo Constrói-Pref
Entrada: Padrão A de tamanho m.
Saída: Tabela Pref.

```
para j ← 1 até m faça
      se A(m) ≠ A(j) então Pref(m, j) ← 0
      senão Pref(m, j) ← 1
para i ← m − 1 até 1 faça
início
   para j ← 1 até m − 1 faça
         se A(i) ≠ A(j) então Pref(i, j) ← 0
         senão Pref(i, j) ← 1 + Pref(i + 1, j + 1)
   se A(i) ≠ A(m) então Pref(i, m) ← 0
   senão Pref(i, m) ← 1
fim
```

Figura 3.6: Algoritmo *Constrói-Pref*

Precisamos entretanto saber de antemão informações sobre segmentos iguais no padrão A.

Para armazenar tais informações utilizaremos uma tabela com m linhas e m colunas, a qual denominaremos de Pref, tal que Pref(i,j) fornece o comprimento do maior prefixo comum dos sufixos $A(i \ldots m)$ e $A(j \ldots m)$ do padrão A. É fácil ver que para i e j tais que $1 \leq i, j \leq m$:

$$\text{Pref}(i,j) = \begin{cases} 1 + \text{Pref}(i+1, j+1) & \text{se } i,j < m \text{ e } A(i) = A(j) \\ 1 & \text{se } (i = m \text{ ou } j = m) \text{ e } A(i) = A(j) \\ 0 & \text{caso contrário.} \end{cases}$$

Com base nesta propriedade, um algoritmo $O(m^2)$ é imediato (figura 3.6).

3.3.4 Uma nova versão de *CalculaM*.

A utilidade das triplas de referência e da tabela Pref ficará mais clara após a leitura da nova versão do algoritmo *CalculaM*, chamada de *CalculaM2*

(Figura 3.7) e das Proposições 3.19 e 3.20, que serão apresentadas mais adiante.

Nesta nova versão de *CalculaM*, supomos que há um conjunto de triplas de referência disponível, que é inicializado como vazio. À medida que a execução do algoritmo *Diferenças* progride, o próprio *CalculaM2* acrescenta triplas de referência ao conjunto, em consonância com a Proposicao 3.17.

Proposição 3.19 *As seguintes afirmações são verdadeiras nos pontos indicados do algoritmo CalculaM2:*

$$i_d < l \leq f_d \tag{3.37}$$

no ponto $\ll 1 \gg$ *e*

$$1 \leq l, l + d - d' \leq m \tag{3.38}$$

no ponto $\ll 3 \gg$ *e*

$$r \text{ está bem definido, } r \geq 0 \tag{3.39}$$

no ponto $\ll 3 \gg$ *e*

$$s \geq 1. \tag{3.40}$$

no ponto $\ll 4 \gg$.

Demonstração: Por indução. Vamos inicialmente provar que $i_d < l \leq f_d$ no ponto $\ll 1 \gg$. Certamente $l \leq f_d$, pela restrição utilizada no comando **enquanto**. Na primeira passagem por $\ll 1 \gg$, o valor de l é $l_0 + 1$, por sua vez, maior do que i_d pela Proposição 3.7. Em passagens posteriores, temos que:

1. se l foi incrementado em $\ll 5 \gg$ então $r \geq 0$ e $s \geq 1$ por hipótese de indução, o que implica que o incremento Δl é não negativo;

2. se l foi incrementado em $\ll 7 \gg$ então o incremento é positivo.

Assim, em ambos os casos o incremento em l é não negativo e portanto a desigualdade $i_d < l$ segue da hipótese de indução.

A desigualdade $1 \leq l \leq m$ em $\ll 3 \gg$ segue imediatamente da desigualdade anterior em $\ll 1 \gg$. Por outro lado, pela definição de tripla de referência e a restrição $\ll 2 \gg$ temos em $\ll 3 \gg$ que

$$i_{d'} < p \leq l + d - d' \leq q \leq f_{d'} \leq m.$$

Algoritmo CalculaM2. Hipótese: $-m \leq d \leq n$ e $\min D_d \leq v \leq \max D_d$.
Entrada: Padrão A e texto B de comprimento m e n, respectivamente.
Saída: Valor $M_{d,v}$.

$$t_0 \leftarrow \begin{cases} i_d & \text{se } v = \min D_d \\ M_{d,v-1} + 1 & \text{caso contrário} \end{cases}$$
$$t_1 \leftarrow \begin{cases} i_d & \text{se } d = -m \text{ ou } v \leq \min D_{d-1} \\ M_{d-1,v-1} & \text{caso contrário} \end{cases}$$
$$t_2 \leftarrow \begin{cases} i_d & \text{se } d = n \text{ ou } v \leq \min D_{d+1} \\ M_{d+1,v-1} + 1 & \text{caso contrário} \end{cases}$$

$t \leftarrow \max\{t_0, t_1, t_2\}$
$l_0 \leftarrow \min\{t, f_d\}$
$l \leftarrow l_0 + 1$
$achou \leftarrow falso$
enquanto $l \leq f_d$ **e não** *achou* **faça**

$\ll 1 \gg$

se $\exists$ tripla de referencia $\tau := (p, q, d')$ **tal que**
$p \leq l + d - d' \leq q$ $\ll 2 \gg$
então
início
$r \leftarrow \text{Pref}(l, l + d - d')$ $\ll 3 \gg$
$s \leftarrow q + d' - (l + d) + 1$ $\ll 4 \gg$
$\Delta l \leftarrow \min\{r, s\}$
$l \leftarrow l + \Delta l$ $\ll 5 \gg$
$achou \leftarrow (r \neq s)$
fim
senão
se $A(l) = B(l + d)$ $\ll 6 \gg$
então $l \leftarrow l + 1$ $\ll 7 \gg$
senão $achou \leftarrow verdadeiro$
$M_{d,v} \leftarrow l - 1$
acrescente $(l_0 + 1, M_{d,v}, d)$ ao conjunto de triplas de referencia

Figura 3.7: Algoritmo *CalculaM2*.

Assim, $1 \leq l + d - d' \leq m$ em $\ll 3 \gg$. Portanto, r está bem definido. Pela definição de Pref, $r \geq 0$.

A desigualdade $s \geq 1$ em $\ll 4 \gg$ segue da restrição $\ll 2 \gg$.

De fato,

1. a desigualdade (3.37) vale na primeira passagem por $\ll 1 \gg$;

2. a validade de (3.37) numa passagem por $\ll 1 \gg$ implica a validade de (3.38), (3.39) e (3.40) nos pontos indicados e conseqüêntemente a validade de (3.37) na próxima passagem por $\ll 1 \gg$. □

Proposição 3.20 *Seja u o comprimento do maior prefixo comum a $A(l \ldots f_d)$ e $B(l+d \ldots f_d+d)$, onde o valor de l é tomado no ponto $\ll 1 \gg$ do algoritmo CalculaM2. Suponha ainda que uma tripla de referência τ que satisfaz a restrição $\ll 2 \gg$ foi encontrada. Então $l + \Delta l \leq 1 + f_d$ e $u - \Delta l$ é igual ao comprimento do maior prefixo comum entre $A(l + \Delta l \ldots f_d)$ e $B(l + \Delta l + d \ldots f_d + d)$. Ademais se $r = s$ então $\Delta l > 0$ e se $r \neq s$ então $u = \Delta l$.*

Demonstração: Pela Proposição 3.19, $r \geq 0$ e $s \geq 1$. Portanto, $\Delta l \geq 0$. Ademais se $r = s$ então $\Delta l > 0$.

Seja $\Delta d := d - d'$.

Por definição da tabela Pref e de r, temos que

$$A(l \ldots l + r - 1) = A(l + \Delta d \ldots l + \Delta d + r - 1). \tag{3.41}$$

Por outro lado, a restrição $\ll 2 \gg$ equivale a

$$p + d' \leq l + d \leq q + d'. \tag{3.42}$$

Portanto, pela definição de tripla de referência e de s, temos

$$A(l + \Delta d \ldots l + \Delta d + s - 1) = B(l + d \ldots l + d + s - 1). \tag{3.43}$$

Assim, de (3.41) e (3.43), pela definição de Δl, temos

$$A(l \ldots l + \Delta l - 1) = B(l + d \ldots l + \Delta l - 1 + d). \tag{3.44}$$

Portanto, $u - \Delta l$ é igual ao maior prefixo comum entre

$$A(l + \Delta l \ldots f_d) \text{ e } B(l + \Delta l + d \ldots f_d + d).$$

Ademais,

$$l + \Delta l - 1 \leq \min\{m, n - d\} = f_d.$$

Resta agora mostrar que se $r \neq s$ então $\Delta l = u$. Suponhamos então que $r \neq s$. Obviamente, se $l + \Delta l = 1 + f_d$ então (3.44) pode ser reescrita da seguinte forma,

$$A(l \ldots f_d) = B(l + d \ldots f_d + d)$$

o que claramente implica que $\Delta l = u$.

Podemos então supor que $l + \Delta l \leq f_d$. Assim,

$$l + \Delta l \leq m$$

e

$$A(l + \Delta l).$$

$$l + \Delta l + d \leq n.$$

Se $r > s$, então, de (3.41) e da definição de r,

$$A(l + \Delta l) = A(l + \Delta d + \Delta l) \neq B(l + \Delta l + d).$$

Se $r < s$ então de (3.43) e da definição de s,

$$B(l + \Delta l + d) = A(l + \Delta d + \Delta l) \neq A(l + \Delta l)$$

Em ambos os casos,

$$A(l + \Delta l) \neq B(l + \Delta l + d)$$

e portanto $\Delta l = u$. □

Corolário 3.21 *O algoritmo CalculaM2 determina corretamente o valor de* $M_{d,v}$. □

3.3.5 Uma nova ordem de cálculo

Na ordem utilizada por *Diferenças-UK*, para cada valor de v, $O(n)$ diagonais são examinadas. Mesmo que fosse mantida apenas uma tripla de referência, por diagonal, a simples busca de uma tripla conveniente em *CalculaM2* custaria $O(n)$, o que inviabilizaria a nova estratégia.

Algoritmo Diferenças-GP. Hipótese: $k < m/2$ e $m \leq n + k$
Entrada: Padrão A de tamanho m, texto B de tamanho n e o número máximo k de diferenças.
Saída: Valores $M_{d,v}$.

```
para d ← −k até n − m + k faça C_d ← verdade
para i ← 0 até k faça τ'_i ← (1,0,0)
para c ← 0 até n − m + k faça
início
   para i ← 0 até k faça τ_i ← τ'_i
   g ← 0
   para v ← 0 até k faça
   início
      d ← c − v
      τ'_v ← τ_v
      se C_d então
      início
         CalculaM3 (d, v)
         C_d ← (M_{d,v} < f_d)
      fim
                                                  ≪ 1 ≫
   fim
fim
```

Figura 3.8: Algoritmo *Diferenças-GP*

A idéia então é mudar a ordem, mantendo-a normal, porém examinando $O(k)$ diagonais por iteração. Em cada iteração, o valor de v varia de 0 a k, mantendo-se a soma $d + v$ constante para o cálculo de $M_{d,v}$.

Em cada iteração estarão disponíveis $k + 1$ triplas de referência $\tau_i := (p_i, q_i, d_i)$ com $(0 \leq i \leq k)$. Ao final da iteração já estarão geradas outras $k + 1$ triplas de referência, $\tau'_i := (p'_i, q'_i, d'_i)$ com $(0 \leq i \leq k)$ que serão utilizadas na iteração seguinte.

A Figura 3.8 apresenta o algoritmo *Diferenças-GP* e a Figura 3.9, o algoritmo *CalculaM3*.

No algoritmo *Diferenças-GP*, assim como no algoritmo *Diferenças-UK*, Figura 3.4, a hipótese do algoritmo *CalculaM* é respeitada. Mais ainda, da mesma forma que na versão do algoritmo *Diferenças-UK*, os valores $M_{d-1,v-1}$, $M_{d,v-1}$ e $M_{d+1,v-1}$, quando necessários para o cálculo de $M_{d,v}$, já foram calculados. A demonstração de tais afirmações será dada a seguir.

Proposição 3.22 *No algoritmo Diferenças-GP, são analisados os pares de valores* (d, v) *tais que*

$$-k \leq d \leq n - m + k$$

e

$$\min D_d \leq v \leq \min\{k, n - m + k - d\}.$$

Demonstração: São analisadas as diagonais d com os valores v tais que

$$\begin{array}{lcccl} d & = & c - v, & & \\ 0 & \leq & c & \leq & n - m + k \\ 0 & \leq & v & \leq & k. \end{array}$$

Assim, o par (d, v) é analisado se e somente se

$$0 \leq d + v \leq n - m + k$$

e

$$0 \leq v \leq k.$$

Ou, de forma equivalente, se e somente se,

$$-v \leq d \leq n - m + k - v$$

e

$$0 \leq v \leq k.$$

Estes dois intervalos são precisamente os intervalos de variação do algoritmo *Diferenças-UK*. O resultado segue então da Proposição 3.9. □

Proposição 3.23 *A ordem de análise dos pares (d, v) do algoritmo Diferenças-GP é normal.*

Demonstração: Seja $c := d + v$. Então

$$\begin{aligned} d-1 &= c-2-(v-1) \\ d &= c-1-(v-1) \\ d+1 &= c-(v-1). \end{aligned}$$

Assim,

$$(d-1, v-1), (d, v-1), (d+1, v-1) \prec (d, v),$$

onde $\prec$ denota a ordem do algoritmo *Diferenças-GP*. □

Comparando *CalculaM2* com *CalculaM3*, convém observar que apenas houve uma mudança na forma de se buscar uma tripla de referência útil, através da variável g. Em ambas as versões uma tripla (p_g, q_g, d_g) é usada somente se

$$p_g + d_g \leq l + d \leq q_g + d_g.$$

Corolário 3.24 *O algoritmo Diferenças-GP resolve o problema das k diferenças.* □

3.3.6 A complexidade do novo algoritmo

A análise da complexidade do Algoritmo *Diferenças-GP* é feita mediante a soma de três parcelas, além do custo $O(m^2)$ para o cálculo da tabela Pref. Lembrando que *CalculaM3* tem uma malha

enquanto $l \leq f_d$ **e não** *achou* **faça**

as três parcelas são as seguintes:

1. a complexidade de *Diferenças-GP* sem levar em conta a complexidade de execução da malha;

Algoritmo CalculaM3. Hipótese: $-m \leq d \leq n$ e $\min D_d \leq v \leq \max D_d$.
Entrada: Padrão A e texto B de comprimento m e n, respectivamente.
Saída: Valor $M_{d,v}$.

$$t_0 \leftarrow \begin{cases} i_d & \text{se } v = \min D_d \\ M_{d,v-1} + 1 & \text{caso contrário} \end{cases}$$
$$t_1 \leftarrow \begin{cases} i_d & \text{se } d = -m \text{ ou } v \leq \min D_{d-1} \\ M_{d-1,v-1} & \text{caso contrário} \end{cases}$$
$$t_2 \leftarrow \begin{cases} i_d & \text{se } d = n \text{ ou } v \leq \min D_{d+1} \\ M_{d+1,v-1} + 1 & \text{caso contrário} \end{cases}$$

$t \leftarrow \max\{t_0, t_1, t_2\}$
$l_0 \leftarrow \min\{t, f_d\}$
$l \leftarrow l_0 + 1$
$achou \leftarrow falso$
enquanto $l \leq f_d$ **e não** *achou* **faça**
início
  $\boxed{x \leftarrow l + d}$ $\ll 1 \gg$
  enquanto $\ll 2 \gg$ $g \leq k$ **e** $l + d > q_g + d_g$ **faça** $g \leftarrow g + 1$
  se $g \leq k$ **e** $p_g + d_g \leq l + d$
  então
  início
    $r \leftarrow \text{Pref}(l, l + d - d'_g)$
    $s \leftarrow q_g + d' - (l + d) + 1$
    $\Delta l \leftarrow \min\{r, s\}$
    $l \leftarrow l + \Delta l$
    $achou \leftarrow (r \neq s)$
  fim
  senão
    se $A(l) = B(l + d)$ **então** $l \leftarrow l + 1$
    senão $achou \leftarrow verdadeiro$
fim
$M_{d,v} \leftarrow l - 1$
se $M_{d,v} + d \geq q_v + d_v$ **então** $\tau'_v \leftarrow (l_0 + 1, M_{d,v}, d)$

Figura 3.9: Algoritmo *CalculaM3*.

2. complexidade total de execuções da malha para $g > k$, sem levar em conta a sua complexidade para $g \leq k$;

3. complexidade total de execuções da malha para $g \leq k$, sem levar em conta a sua complexidade quando $g > k$.

Denotaremos estas três parcelas por α, β e γ, respectivamente. Uma simples inspeção do algoritmo revela que $\alpha = O(kn)$. Conforme veremos nas Proposições 3.31 e 3.32, $\beta = O(kn)$ e $\gamma = O(kn)$.

Assim, a complexidade do algoritmo *Diferenças-GP* é $O(kn + m^2)$, onde a parcela m^2 vem do pré-processamento do padrão.

Para podermos avaliar β e γ, necessitamos de uma série de resultados auxiliares, estabelecidos nas Proposicões 3.25, 3.26, 3.27, 3.28, 3.29 e 3.30, demonstradas a seguir.

Proposição 3.25 *Se $v > 0$ e $M_{d,v}$ foi calculado no algoritmo Diferenças-GP então $M_{d+1,v-1}$ já foi calculado.*

Demonstração: Seja $d = c - v$. Pela Proposição 3.22, e usando a desigualdade $k < m/2$, temos

$$0 < v \leq n - m + k - d \leq n - d,$$

logo

$$d < n. \tag{3.45}$$

Na iteração $(c, v-1)$, a diagonal $d+1$ é analisada. Pela Proposição 3.22, $\min D_{d+1} \leq v - 1$, logo

$$v > \min D_{d+1}. \tag{3.46}$$

De (3.45) e (3.46) e pela especificação do algoritmo *CalculaM3* temos que

$$t_2 = 1 + M_{d+1,v-1}.$$

Pela normalidade da ordem de cálculo do algoritmo *Diferenças-GP*, $M_{d+1,v-1}$ foi previamente calculado: esse cálculo foi realizado na iteração anterior, uma vez que não há repetição de pares (c, v). □

Proposição 3.26 *Durante a execução do algoritmo Diferenças-GP, para uma iteração em que a variável c está fixa, considere a seqüência χ de valores atribuídos à variável x em CalculaM3, no ponto $\ll 1 \gg$. As seguintes afirmações são verdadeiras.*

1. *a subseqüência de χ correspondente a uma sub-iteração com v fixo é estritamente crescente;*

2. *a seqüência χ é (não estritamente) crescente.*

Demonstração:

1. Os incrementos que l sofre dentro do comando

   enquanto $l \leq f_d$ e **não** *achou* **faça**

   são ou 1 ou Δl; pela Proposicão 3.20, $\Delta l \geq 0$, com igualdade somente na última iteração do comando no algoritmo *CalculaM3*.

2. Seja v no intervalo $0 < v \leq k$. Seja $d := c - v$. Suponhamos que *CalculaM3*(d, v) foi executado. Suponhamos ainda que, durante essa execução, a variável x sofreu uma ou mais atribuições. Seja x_1 o valor usado pela primeira dessas atribuições.

   Dado que houve tais atribuições, então $l_0 < f_d$ e, portanto, $t_2 \leq t = l_0$. Assim,

   $$t_2 + d + 1 \leq l_0 + d + 1 = x_1. \tag{3.47}$$

   Pela Proposição 3.25 temos que

   $$t_2 = 1 + M_{d+1,v-1}. \tag{3.48}$$

   Seja $\bar{l}$ o último valor assumido pela variável l durante o cálculo de $M_{d+1,v-1}$ na iteração $(c, v-1)$. Então

   $$\bar{l} = 1 + M_{d+1,v-1} = t_2 \leq f_{d+1}. \tag{3.49}$$

   Assim, a atribuição em $\ll 1 \gg$ foi executada durante o cálculo de $M_{d+1,v-1}$. Seja x_0 o último valor atribuído a x durante esse cálculo.

   Observe que x_0 imediatamente precede x_1 na seqüência χ.

   Pela parte (1),

   $$x_0 \leq \bar{l} + (d + 1).$$

De (3.49) e (3.47) temos

$$x_0 \leq t_2 + d + 1 \leq x_1.$$

De fato, $x_0 \leq x_1$. Demonstramos então que se é atribuído um valor a x durante o cálculo de $M_{d,v}$ $(v > 0)$ então o último valor de x_0 atribuído não excede o primeiro valor x_1 atribuído na iteração (c, v). Assim, levando em conta a parte (1) temos a validade de (2). □

Proposição 3.27 *As seguintes afirmações envolvendo as novas triplas de referência são verdadeiras no ponto* $\ll 1 \gg$ *do algoritmo Diferenças-GP:*

1. $q'_v + d'_v = \max\{M_{i-v,v} + i - v : 0 \leq i \leq c \text{ e } \min D_{i-v} \leq v \leq \max D_{i-v}\}$;
2. *se* $v > 0$ *então* $p'_v + d'_v \leq 2 + q'_{v-1} + d'_{v-1}$.

Demonstração: Por indução em c. Seja $d := c - v$.

1. Temos que

$$v = c - d \geq -d \geq \min D_d. \tag{3.50}$$

Vamos considerar o caso em que $c = 0$. Neste caso, $d \leq 0$ e, portanto,

$$v = -d = \min D_d.$$

Logo, o único elemento do conjunto cujo máximo se pretende determinar é

$$M_{d,v} + d \geq 0.$$

Pela inicialização das triplas com $(1, 0, 0)$,

$$q_v + d_v = 0.$$

Assim, $q'_v = M_{d,v}$, $d'_v = d$ e, portanto a asserção vale.

Resta agora considerar o caso em que $c > 0$. Por hipótese de indução

$$q_v + d_v = \max\{M_{i-v,v} + i - v : 0 \leq i \leq c-1 \text{ e } \min D_{i-v} \leq v \leq \max D_{i-v}\}. \tag{3.51}$$

Se $v > \max D_d$ então a asserção segue trivialmente de (3.51). Se $v \leq \max D_d$ então, por (3.50), $M_{d,v}$ é calculado, e portanto

$$q'_v + d'_v = \max\{q_v + d_v, M_{d,v} + d\};$$

novamente, a asserção segue de (3.50).

2. Se $c = 0$ então a inicialização das triplas forneceu à variável p_v o valor 1 e a q_v, d_v, d_{v-1} o valor 0. Se $c > 0$ podemos usar a hipótese de indução. Em ambos os casos,

$$p_v + d_v \leq 2 + q_{v-1} + d_{v-1}.$$

Pela especificação do algoritmo,

$$q_{v-1} + d_{v-1} \leq q'_{v-1} + d'_{v-1}$$

e, portanto,

$$p_v + d_v \leq 2 + q'_{v-1} + d'_{v-1}.$$

Se $p'_v + d'_v = p_v + d_v$ então a asserção segue trivialmente. Podemos portanto supor que $M_{d,v}$ foi calculado e daí

$$p'_v = 1 + l_0 \text{ e } d'_v = d.$$

Assim, lembrando que

$$l_0 = \min\{t, f_d\} \leq \max\{t_0, t_1, t_2\},$$

resta mostrar que

$$t_x + d \leq 1 + q'_{v-1} + d'_{v-1} \tag{3.52}$$

para todo x em $0 \leq x \leq 2$.

Vamos então considerar inicialmente o caso em que $v = \min D_d$. Neste caso,

$$t_0 = i_d. \tag{3.53}$$

Por hipótese, $v > 0$. Mas $v = \min D_d = \max\{0, v - c\}$. Assim, $d < 0$ e $c = 0$. Portanto,

$$\text{ou } d = -m \text{ ou } (d > -m \text{ e } v < \min D_{d-1})$$

e

$$v - 1 = \min D_{d+1} = i_d - 1.$$

Podemos então concluir que

$$t_1 = i_d \tag{3.54}$$

$$t_2 = 1 + M_{d+1,v-1} \geq i_d. \tag{3.55}$$

Assim, de (3.53), (3.54) e (3.55) e por (1)

$$t_0 + d = t_1 + d = i_d + d \leq t_2 + d = M_{d+1,v-1} + d + 1 \leq q'_{v-1} + d'_{v-1}$$

e (3.52) é nesse caso uma desigualdade estrita.

Resta agora demonstrar (3.52) para o caso em que $v > \min D_d$. Nesse caso,

$$t_0 = 1 + M_{d,v-1}$$

e portanto, por (1)

$$i_d + d \leq t_0 + d = 1 + M_{d,v-1} + d \leq 1 + q'_{v-1} + d'_{v-1}.$$

Assim, (3.52) vale para t_0, e para t_x se $t_x = i_d$. Se $t_1 \neq i_d$ então por (1),

$$t_1 + d = M_{d-1,v-1} + d \leq 1 + q'_{v-1} + d'_{v-1}.$$

Se $t_2 \neq i_d$ então, por (1)

$$t_2 + d = 1 + M_{d+1,v-1} + d \leq 1 + q'_{v-1} + d'_{v-1}.$$

De fato, (3.52) vale também nos casos em que $v > \min D_d$. □

Proposição 3.28 *No algoritmo CalculaM3, se* $v = 0$ *então* $l_0 = 0$.

Demonstração: O valor da variável d é

$$v = c - d = c \geq 0.$$

Assim,

$$\min D_d = 0$$

e

$$\text{ou } d = -m \text{ ou } 0 \leq \min D_{d-1}$$

e

$$\text{ou } d = n \text{ ou } 0 = \min D_{d+1}.$$

Portanto pela especificação de *CalculaM3*,

$$t_0 = t_1 = t_2 = i_d = 0.$$

Logo,

$$l_0 = \min\{t, f_d\} = t = 0.\square$$

Proposição 3.29 *As seguintes invariantes são válidas no ponto* $\ll 2 \gg$*:*

1. *se* $g \leq k$ *então* $x \geq p_g + d_g - 1$;
2. $x > q_i + d_i$ *para todo* i *no intervalo* $0 \leq i \leq g - 1$.

Demonstração: Por indução.

Vamos inicialmente demonstrar a validade da asserção por ocasião da primeira passagem pelo ponto $\ll 2 \gg$. Consideraremos que a diagonal analisada é $d := c - v$. Assim, necessariamente, $g = 0$. Pela Proposição 3.25, $v = 0$. Em resumo,

$$d = c \text{ e } v = g = 0.$$

Assim, pela Proposição 3.28,

$$x = l_0 + 1 + d = 1 + c.$$

Portanto, se τ_0 é a tripla inicial $(1, 0, 0)$ então a desigualdade em (1) vale de forma estrita. Por outro lado, se τ_0 for uma tripla calculada, em que a variável c tinha o valor c_0, então, pela Proposição 3.28

$$p_0 + d_0 = 1 + c_0 \leq 1 + c,$$

e, novamente a desigualdade em (1) vale de forma estrita.

A desigualdade em (2) é vacuosamente verdadeira neste caso.

Vamos considerar agora outras passagens por $\ll 2 \gg$ que não a primeira, para um valor de c fixo. Suponhamos, como hipótese de indução, que a asserção é válida por ocasião da passagem anterior, em que a variável x tinha o valor x' e g o valor g'.

Vamos considerar inicialmente o caso em que $g = g'$. Pela Proposição 3.26,

$$x \geq x'.$$

Por indução

$$\begin{array}{lll} x' & \geq & p_g + d_g - 1 \quad \text{se } g \leq k \\ x' & > & q_i + d_i \qquad \forall i \; 0 \leq i \leq g - 1 \end{array}$$

e, portanto, a asserção vale.

Resta agora considerar o caso em que $g = g' + 1$ (e $x = x'$). Necessariamente,

$$x > q_{g-1} + d_{g-1}. \tag{3.56}$$

e, por hipótese de indução,

$$x > q_i + d_i \ \forall i \ 0 \leq i \leq g - 2. \tag{3.57}$$

Se $g \leq k$,

$$q_{g-1} + d_{g-1} \geq p_g + d_g - 2. \tag{3.58}$$

A validade desta desigualdade vem do valor $(1, 0, 0)$ de inicialização das triplas, se $c = 0$, e da Proposição 3.27(2) se $c > 0$. A validade de (1) segue então de (3.56) e (3.58), e a validade de (2) segue de (3.56) e (3.57). □

Proposição 3.30 *Para cada c no intervalo $0 \leq c \leq n - m + k$, defina*

$$z_c := \max\{q_i + d_i : 0 \leq i \leq k\}$$

Então seqüência $z_0, z_1, \ldots, z_{n-m+k}$ é crescente. Se para uma iteração (c, v) (do algoritmo Diferenças-GP) o algoritmo CalculaM3(d, v) é executado e, além disso, uma comparação entre $A(l)$ e $B(x)$ é efetuada e $g = k+1$ nesse instante então

$$x > z_c.$$

Ademais, se a comparação resulta em igualdade e $c < n - m + k$ então

$$z_{c+1} \geq x.$$

Demonstração: A desigualdade

$$z_c \leq z_{c+1}$$

segue do fato que na iteração c

$$q_i' + d_i' \geq q_i + d_i.$$

A desigualdade

$$x > z_c$$

segue da Proposição 3.29(2).

Além disso, se a comparação resultar em igualdade então

$$M_{d,v} \geq l$$

e portanto

$$q'_v + d'_v \geq M_{d,v} + d \geq l + d = x.$$

Assim,

$$z_{c+1} \geq x. \square$$

Proposição 3.31 *A parcela β de complexidade do algoritmo Diferenças-GP é $O(kn)$.*

Demonstração: Na situação $g = k + 1$, as triplas de referência não são usadas, e portanto, a cada volta na malha o caracter em uma posição l do padrão A é comparado com um caracter em uma posição x do texto B. O custo de uma dessas volta é O(1). A cada comparação em que $A(l) \neq B(x)$, a execução da malha é terminada.

Assim, para um c fixo o custo total das voltas da malha em que $g = k+1$ e $A(l) \neq B(x)$ é $O(k)$. Portanto, o custo total das voltas da malha em que $g = k + 1$ e $A(l) \neq B(x)$ é $O(kn)$, levando em conta todos os possíveis valores de c.

Vamos agora estimar o custo total das voltas de malha em que $g = k+1$ e $A(l) = B(x)$, para x fixo e l assumindo todos os valores possíveis no intervalo $1 \leq l \leq m$.

Seja c_0 o menor valor de c tal que para algum valor de v e de l a comparação $A(l) = B(x)$ é efetuada com $g = k + 1$.

Pela Proposição 3.30,

$$z_c \geq x > z_{c_0} \quad \forall c: \ c_0 + 1 \leq c \leq n - m + k.$$

Portanto, se $g = k + 1$ então a posição x do texto B somente é utilizada em comparações com igualdade com caracteres do padrão, quando $c = c_0$.

Pela Proposição 3.26 a posição x é utilizada para comparações no máximo $k+1$ vezes durante a iteração c_0. Podemos então concluir que, para uma posição fixa x do texto, o custo total das malhas em que $g = k + 1$ e a posição x é utilizada para comparação com igualdade é $O(k)$.

Logo, $\beta = O(kn)$. $\square$

Proposição 3.32 *A parcela γ de complexidade do algoritmo Diferenças-GP é $O(kn)$.*

Demonstração: Quando $g \leq k$, no ponto $\ll 2 \gg$ do algoritmo *CalculaM3*, podemos ter uma das seguintes situações, pela Proposição 3.29:

1. $x = p_g + d_g - 1$;
2. $p_g + d_g \leq x \leq q_g + d_g$;
3. $x > q_g + d_g$.

Na situação (1), é realizada uma comparação entre padrão e texto que ou acarreta o término da execução da malha (uma desigualdade) ou incrementa de 1 o valor de l (uma igualdade). No segundo caso, na nova passagem por $\ll 2 \gg$ estaremos na situação (2) ou (3).

Na situação (2), a tripla é utilizada; se $r \neq s$ então a execução da malha é terminada; se $r = s$ então l é incrementado de forma que na passagem seguinte por $\ll 2 \gg$

$$x = q_g + d_g + 1$$

e estaremos assim, na situação (3).

Na situação (3) o valor de g é incrementado de 1.

Podemos então concluir que se estamos no ponto $\ll 2 \gg$ da malha, em O(1) voltas

ou a execução da malha termina

ou g é incrementado.

Conseqüêntemente, para c fixo o custo total das execuções da malha com $g \leq k$ é $O(k)$. Assim, $\gamma = O(kn)$. □

3.4 Algoritmo de Tarhio e Ukkonen

3.4.1 Introdução

O algoritmo proposto nesta seção resolverá problema das k diferenças através do método da programação dinâmica (equação 3.1). Ao invés de calcular todos os elementos da matriz D, o algoritmo calculará apenas alguns elementos. Mais precisamente, o método da programação dinâmica será utilizado apenas em algumas diagonais, ditas *diagonais marcadas*. Para efeito de utilização da equação (3.1), os valores nas diagonais não marcadas poderão ser tomados como ∞, sem prejuízo de obtenção das soluções do problema.

Convém recordar que no problema das k diferenças que estamos considerando neste capítulo temos um padrão A de comprimento m, um texto B de comprimento n e operações de edição com custo unitário, o que nos permite adotar a hipótese,

$$k < \frac{m}{2} \text{ e } m \leq n + k$$

conforme já foi visto na seção 3.1.

Para limitar o número de diagonais a serem analisadas na matriz D, o algoritmo descrito nesta seção analisará se um determinado alinhamento entre o padrão e o texto está suficientemente próximo de outro alinhamento que possa levar a uma possível ocorrência aproximada. Em caso afirmativo, diagonais próximas à diagonal relativa ao alinhamento são marcadas.

Após ter realizado a análise descrita acima, deslocaremos o padrão, utilizando as idéias do algoritmo de Boyer e Moore [BM77], para analisar outros alinhamentos.

Podemos, então, dizer que este algoritmo está dividido em duas operações. Uma operação é a de marcar as diagonais e a outra é a de deslocar o padrão.

Na seção 3.4.2 descreveremos a operação de marcar as diagonais e na seção 3.4.3 descreveremos como deslocar o padrão.

3.4.2 Operação de marca

Nesta seção descreveremos um método para marcar as diagonais da matriz D. É nas diagonais marcadas que o método da programação dinâmica será

aplicado.

O método da programação dinâmica gera uma dependência entre os valores $D(i,j)$ cuja representação gráfica é chamada de *grafo de dependência.* Neste grafo existe um arco orientado com origem em (i,j) e destino (i',j') se, e somente se, o passo de minimização da equação (3.1) (página 31) gera $D(i',j')$ a partir de $D(i,j)$.

Observe que a aresta do grafo de dependência que sai de (i,j) e entra em (i',j') é de um dos seguintes tipos:

- *horizontal* se $i' = i$ e $j' = j+1$;
- *diagonal* se $i' = i+1$ e $j' = j+1$;
- *vertical* se $i' = i+1$ e $j' = j$.

Dado um caminho C no grafo de dependência, denotaremos por $H(C)$, $D(C)$ e $V(C)$, respectivamente, o conjunto das arestas de C que são horizontais, diagonais e verticais. Dentre as arestas diagonais, denotamos ainda por $D_1(C)$ o conjunto daquelas que correspondem a uma colisão e por $D_0(C)$ o conjunto das demais. Assim, o custo de C é $|V(C)| + |H(C)| + |D_1(C)|$

Proposição 3.33 *Para todo caminho C no grafo de dependências,*

$$|V(C)| - |H(C)| = d_0 - d_1,$$

onde d_0 e d_1 denotam, respectivamente, as diagonais que passam pelos pontos inicial e final de C.

Demonstração: Por indução. Se o caminho C não tem arestas então todas as parcelas que participam dos termos da igualdade são nulos.

Se o caminho C tem precisamente uma aresta a afirmação segue imediatamente das definições de aresta horizontal, diagonal e vertical.

Podemos então supor que C passa por um ponto $p := (j-d, j)$, distinto dos pontos inicial e final de C. Seja C_1 e C_2 os sub-caminhos do seu ponto inicial a p e de p ao seu ponto final, respectivamente. Por hipótese de indução,

$$\begin{aligned} |V(C_1)| - |H(C_1)| &= d_0 - d \\ |V(C_2)| - |H(C_2)| &= d - d_1 \end{aligned}$$

A soma destas igualdades fornece a igualdade enunciada. □

Denominaremos de *caminho ótimo* todo caminho do grafo de dependência que tem origem em $(0, j)$ e destino (m, h) tal que $D(m, h) \leq k$, onde $0 \leq j \leq n$ e $0 \leq h \leq n$.

Proposição 3.34 *Todo caminho ótimo C intercepta no máximo $k + 1$ diagonais.*

Demonstração: Seja

$$\{d : d_* \leq d \leq d^*\}$$

o conjunto das diagonais que C intercepta. Sejam p_* e p^* pontos de C nas diagonais d_* e d^*, respectivamente. Se p_* precede p^* em C, seja C' o subcaminho de C que vai de p_* de a p^*. Se p^* precede p_* em C, seja C' o subcaminho de C que vai de p^* a p_*.

Se p_* precede p^*, então, pela Proposição 3.33,

$$d^* - d_* = |H(C')| - |V(C')| \leq |H(C)| \leq k.$$

Analogamente, se p^* precede p_* então

$$d^* - d_* = |V(C')| - |H(C')| \leq |V(C)| \leq k.$$

Em ambos os casos, $d^* - d_* \leq k$. □

Dada uma posição i do padrão ($1 \leq i \leq m$), definimos a *vizinhança* S_i como sendo o seguinte conjunto de caracteres:

$$S_i := \{A(j) : 1 \leq j \leq m \text{ e } |j - i| \leq k\}.$$

Assim, S_i é o conjunto dos caracteres que ocorrem no padrão em posições que distam no máximo k da posição i.

Seja d uma diagonal tal que

$$0 \leq d \leq n - m + k.$$

Uma coluna j é *ruim com relação à diagonal* d se

$$k < j - d \leq \min\{m, n - d\} \text{ e } B(j) \notin S_{j-d}.$$

Assim, uma coluna j é ruim se e somente se está a mais do que k colunas à direita do seu ponto $(0, d)$, intercepta a diagonal, e o caracter $B(j)$ não está na vizinhança da posição $j - d$ do padrão (Figura 3.10). Vamos denotar por R_d o conjunto das colunas ruins com relação à diagonal d.

O resultado apresentado a seguir tem um papel fundamental na marcação de diagonais:

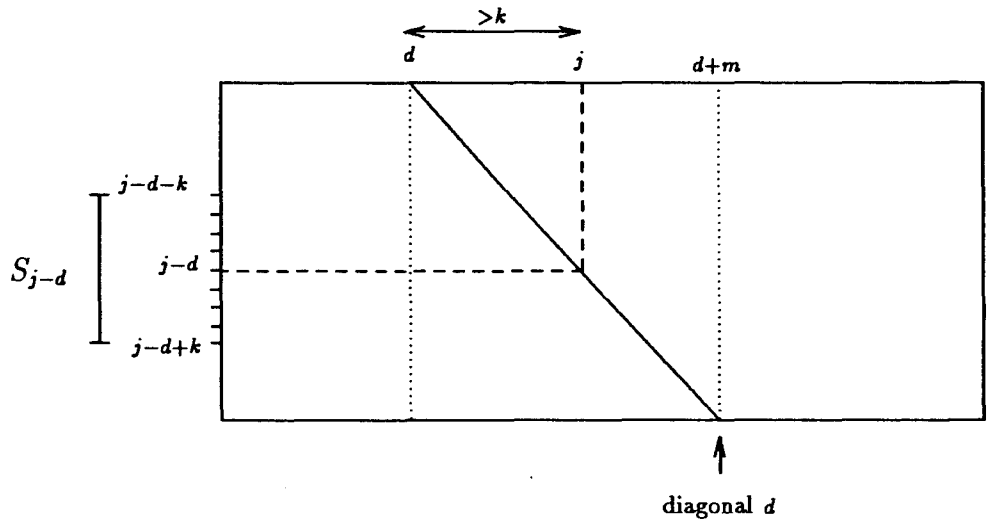

Figura 3.10: O conjunto S_{j-d} indica os caracteres de A onde devemos procurar a ocorrência do caracter $B(j)$.

Teorema 3.35 *Seja d uma diagonal no intervalo*

$$0 \leq d \leq n - m + k.$$

Seja C um caminho ótimo cujos pontos estão todos em diagonais no intervalo

$$\{x : d - k \leq x \leq d + k\}.$$

Se C tem pelo menos um ponto em uma diagonal $x \geq d$, então

$$|R_d| \leq k.$$

Demonstração: Seja d_0 a diagonal que passa pelo ponto inicial de C, d_1 a diagonal que passa pelo seu ponto final. Assim, os pontos inicial e final de C são, respectivamente, $(0, d_0)$, $(m, m + d_1)$.

Sejam

$$X := \{j : j \in R_d, d_0 < j \leq m + d_1\}$$

e

$$Y := R_d \backslash X.$$

Assim, X é o conjunto das colunas ruins em relação à diagonal d que interceptam o caminho C em pontos à direita do seu ponto inicial e Y é o conjunto das demais colunas de R_d.

Pelas Proposições 3.36 e 3.37, abaixo enunciadas,

$$|R_d| = |X| + |Y| \leq |H(C)| + |D_1(C)| + |V(C)| \leq k.$$

A demonstração do Teorema se reduz assim à demonstração das duas Proposições.

Proposição 3.36 $|X| \leq |H(C)| + |D_1(C)|$.

Proposição 3.37 $|Y| \leq |V(C)|$.

As demonstrações das Proposições 3.36 e 3.37 serão dadas a seguir.

Demonstração da Proposição 3.36: Para cada $j \in X$, temos, por definição de X que j intercepta C. Seja i o menor inteiro tal que $(i, j) \in C$.

Por definição de X, j intercepta C em pontos à direita do seu ponto inicial $(0, d_0)$.

Assim, a aresta de $(i, j) \neq (0, d_0)$. Seja então α a aresta de C que entra no ponto (i, j).

Pela definição de i, a aresta α não é vertical. Assim,

$$\alpha \in H(C) \cup D(C).$$

Vamos agora mostrar que na verdade

$$\alpha \in H(C) \cup D_1(C).$$

Por hipótese, $j \in R_d$. Assim,

$$B(j) \notin S_{j-d}. \tag{3.59}$$

Também por hipótese, a diagonal $j - i$, uma das diagonais que C intercepta, satisfaz

$$d - k \leq j - i \leq d + k$$

o que equivale a

$$j - d - k \leq i \leq j - d + k.$$

Assim,

$$A(i) \in S_{j-d}. \tag{3.60}$$

De (3.59) e (3.60), concluímos que

$$A(i) \neq B(j).$$

Portanto, a aresta α, se for diagonal, corresponde a uma colisão. De fato,

$$\alpha \in H(C) \cup D_1(C).$$

Finalmente, diferentes colunas j em X correspondem a diferentes aresta α em $H(C) \cup D_1(C)$ que entram em j. Daí a desigualdade enunciada. □

Demonstração da Proposição 3.37: Sejam

$$T := \{j : k < j - d \leq \min\{m, n - d\}\}$$

e

$$W := \{j : d_0 < j \leq m + d_1\}.$$

Da definição de R_d e de Y temos então que

$$Y \subseteq T \backslash W.$$

Vamos inicialmente avaliar $|T \cap W|$. Uma coluna j está em $T \cap W$ se e somente se

$$k + d < j \leq \min\{m + d, n\}$$

e

$$d_0 < j \leq m + d_1.$$

Assim,

$$|T \cap W| = \min\{m + d, n, m + d_1\} - \max\{k + d, d_0\}.$$

Mas $m + d_1$ é a coluna que contém o ponto final de C e portanto $m + d_1 \leq n$. Por hipótese, d_0, uma diagonal que intercepta C, satisfaz $d_0 \leq d + k$.

Portanto,

$$\begin{aligned} |T \cap W| &= \min\{m + d, m + d_1\} - (d + k) \\ &= m - k + \min\{0, d_1 - d\} \\ &= m - k - \max\{0, d - d_1\}. \end{aligned}$$

Por outro lado, obviamente

$$|T| = \min\{m, n-d\} - k \leq m - k.$$

Assim,

$$|Y| \leq |T \backslash W| \leq \max\{0, d - d_1\}.$$

Se $d_1 \geq d$ então

$$|Y| \leq 0 \leq |V(C)|.$$

Podemos portanto supor que $d_1 < d$, o que implica que

$$|Y| \leq d - d_1.$$

Por hipótese, C contém pontos em diagonais $x \geq d$. Portanto, C e d se interceptam, uma vez que d_1 é uma diagonal que intercepta C e $d_1 < d$. Seja p um ponto nessa interseção, seja C' o subcaminho de C que vai de p até o ponto final de C. Pela Proposição 3.33,

$$d - d_1 = |V(C')| - |H(C')| \leq |V(C)|.$$

Independentemente do resultado da comparação entre d e d_1 temos então a desigualdade enunciada. □

A demonstração da Proposição 3.37 completa a demonstração do Teorema 3.35.

O Teorema 3.35 sugere a seguinte metodologia para marcar as diagonais: para a diagonal d e para i variando de k a $\min\{m, n-d\}$, compare $B(i+d)$ com os caracteres de S_i. Se o número total de colunas ruins encontradas for no máximo k então marcaremos as diagonais $d-k, \ldots, \min\{n-m, d\}+k$, pois, a existência de um número maior do que k colunas ruins indica que não pode haver um caminho ótimo com pontos em diagonais $x \geq d$ mas totalmente contido em diagonais no intervalo $d - k \leq x \leq d + k$.

Após ter analisado a diagonal d devemos determinar a próxima diagonal cuja vizinhança será possivelmente marcada. Esta operação será descrita na seção 3.4.3. No restante desta seção descreveremos como obter as colunas ruins de forma mais eficiente do que a indicada acima.

Para encontrar as colunas ruins construiremos a tabela Ruim com a seguinte definição:

$$\text{Ruim}(i, a) := a \notin S_i \ (k < i \leq m, a \in \Sigma).$$

Algoritmo Constrói-Ruim
Entrada: Padrão A e seu respectivo tamanho m.
Saída: Tabela Ruim.

para $i \leftarrow k+1$ **até** m **faça**
início
  para cada $a \in \Sigma$ **faça** $\text{Ruim}(i, a) \leftarrow$ verdadeiro
  para $j \leftarrow i-k$ **até** $\min\{m, i+k\}$ **faça**
  $\text{Ruim}(i, A(j)) \leftarrow$ falso
fim

Figura 3.11: Algoritmo *Constrói-Ruim*

Assim,

$$j \in R_d \text{ se e somente se } k < j - d \leq \min\{m, n-d\} \text{ e } \text{Ruim}(j-d, B(j)).$$

A tabela pode ser calculada através de uma varredura no padrão A como é feito no algoritmo *Constrói-Ruim*, Figura 3.11.

O processo de análise de uma diagonal d para possível marcação das diagonais $d-k \ldots \min\{n-m, d\}+k$ é então muito simples, sua descrição está na Figura 3.12.

3.4.3 Operação de deslocamento

Para cada posição i no padrão no intervalo $m-k \leq i \leq m$, e para cada caracter a de Σ, vamos definir

$$\text{Des}(i, a) := \min[\{i-h : 1 \leq h \leq i, A(h) = a\} \cup \{i\}].$$

Assim, $\text{Des}(i, a)$ é o menor deslocamento para a direita que se deve fazer no padrão em que a posição i está alinhada com uma ocorrência do caracter a no texto, para que não haja colisão naquela posição do texto.

Dada uma diagonal d tal que $0 \leq d \leq n-m+k$, definimos

$$\delta_d := \max\{k+1, \min\{\text{Des}(i, B(i+d)) : m-k \leq i \leq \min\{m, n-d\}\}.$$

Os próximos resultados são cruciais para a operação de deslocamento e esclarece o papel de Des e δ.

Algoritmo Analisa Diagonal
Entrada: Diagonal d, $0 \leq d \leq n - m + k$.
Saída: Possível marcação das diagonais $d - k \ldots \min\{n - m, d\} + k$.

$r \leftarrow 0$
$i \leftarrow \min\{m, n - d\}$
enquanto $i > k$ **e** $r \leq k$ **faça**
início
  se Ruim$(i, B(i + d))$ **então** $r \leftarrow r + 1$
  $i \leftarrow i - 1$
fim
se $r \leq k$ **então** *marque* $d - k \ldots \min\{n - m, d\} + k$

Figura 3.12: Algoritmo *Analisa Diagonal*

Teorema 3.38 *Seja d uma diagonal no intervalo $0 \leq d \leq n - m + k$, C um caminho ótimo que contenha pontos em diagonais $x \geq d$. Uma das seguintes alternativas é verdadeira:*

ou (1) *todos os pontos de C estão em diagonais no intervalo $\{x : d - k \leq x \leq d + k\}$;*

ou (2) *o caminho C tem pontos em diagonais $x \geq d + \delta_d$.*

Demonstração: Suponhamos que a alternativa (1) seja falsa. Dado que C tem pontos em diagonais $x \geq d$, então, pela Proposição 3.34, todo ponto de C está em diagonais $x \geq d - k$. A falsidade da alternativa (1) implica então que

$$C \text{ tem pontos em diagonais } x > d + k. \tag{3.61}$$

Se $\delta_d = k + 1$ então C tem pontos em diagonais $x \geq d + \delta_d$. Podemos supor que

$$\delta_d = \min\{\text{Des}(i, B(i + d)) : m - k \leq i \leq \min\{m, n - d\}\}.$$

Em particular, lembrando que $d \leq n - m + k$, temos que $m - k \leq \min\{m, n - d\}$ e portanto

$$\delta_d \leq m - k,$$

pois, $\mathrm{Des}(i, a) \leq i$ para todo $a \in \Sigma$. Podemos portanto supor que todo ponto de C está em diagonais $x < m - k + d$.

Sejam d_0 e d_1 as diagonais que passam pelos pontos inicial e final de C, respectivamente. Temos que

$$d_0 < m - k + d$$

e

$$d_1 > d, \tag{3.62}$$

onde a segunda desigualdade segue de (3.61), pela Proposição 3.34. Portanto, todas as colunas j no intervalo

$$m - k + d \leq j \leq m + d \tag{3.63}$$

interceptam o caminho C à direita do seu ponto inicial. Mas o número dessas colunas é $k + 1$, logo nem todas as arestas de C que entram nessas colunas podem ter custo 1. Assim, pelo menos uma delas está em $D_0(C)$. Seja $(j-c, j)$ o ponto em que uma aresta de $D_0(C)$ entra, com j no intervalo (3.63).

Seja

$$i := j - d.$$

Então

$$B(i + d) = B(j) = A(j - c) = A(i - (c - d))$$

e

$$m - k \leq i \leq m = \min\{m, n - d\}$$

onde a última igualdade segue de (3.62), pois $m + d_1 \leq n$. Logo,

$$\mathrm{Des}(i, B(i + d)) \leq c - d$$

e, portanto,

$$\delta_d \leq c - d.$$

Assim, o ponto $(j - c, j)$ de C está na diagonal c, que por sua vez satisfaz

$$d + \delta_d \leq c. \square$$

Com base no Teorema 3.38 damos na Figura 3.13 o algoritmo *Diferenças-TU*

Algoritmo Diferenças-TU
Entrada: Tabelas Ruim e Des.
Saída: Diagonais da matriz D marcadas.

$d \leftarrow 0$
repita
  $r \leftarrow 0$
  $i \leftarrow \min\{m, n-d\}$
  $\delta \leftarrow m - k$
  enquanto $r \leq k$ **e** $i > k$ **faça**
  início
    se $m - k \leq i$ **então** $\delta \leftarrow \min\{\delta, \mathrm{Des}(i, B(i+d))\}$
    se $\mathrm{Ruim}(i, B(i+d))$ **então** $r \leftarrow r + 1$
    $i \leftarrow i - 1$
  fim
  se $r \leq k$ **então** *marque as diagonais* $d - k \ldots \min\{n - m, d\} + k$
  $\delta \leftarrow \max\{k + 1, \delta\}$ $\ll 1 \gg$
  $d \leftarrow d + \delta$
até que $d > n - m + k$

Figura 3.13: Algoritmo *Diferenças-TU*

Convém observar que, tal como apresentado, o algoritmo determina corretamente se $R_d \leq k$ para cada diagonal d analisada. No entanto, nem todos os valores de $\mathrm{Des}(i, B(i+d))$ com

$$m - k \leq i \leq n - d$$

são utilizados, apenas aqueles cujos correspondentes i satisfazem

$$k < i.$$

Esta restrição adicional é inócua se $k < m - k$.

Vamos analisar o que ocorre no caso em que $k \geq m - k$. Essa desigualdade equivale a $k \geq m/2$. Assim, conforme visto na seção 3.1, os algoritmos $O(mn)$ são também $O(kn)$. Portanto, o algoritmo não é eficiente nesse caso.

Teorema 3.39 *As diagonais marcadas pelo algoritmo Diferenças-TU incluem todos os pontos de todos os caminhos ótimos.*

Demonstração: Seja C um caminho ótimo. Por definição, o ponto inicial de C está numa diagonal não negativa. Assim, C tem pontos em diagonais $x \geq 0$; note que 0 é uma das diagonais analisadas pelo algoritmo (a primeira delas). Seja d a última diagonal analisada pelo algoritmo tal que C tem pontos em diagonais $x \geq d$.

O valor δ calculado na iteração em que d é analisada é precisamente δ_d.

Se d não for a última diagonal analisada pelo algoritmo então todo ponto de C pertence a diagonais $x < d + \delta_d$, por definição d.

Alternativamente, se d for a última diagonal analisada pelo algoritmo então

$$d + \delta_d > n - m + k.$$

Nenhuma diagonal $x > n - m + k$ contém pontos de caminhos ótimos, pela Proposição 3.34, pois o ponto final de C está, por definição de C numa diagonal que não excede $n - m$.

Assim, seja d uma diagonal que não a última diagonal analisada pelo algoritmo, todo ponto de C está em diagonais $x < d + \delta_d$. Pelo Teorema 3.38, todo ponto de C está em diagonais x no intervalo

$$d - k \leq x \leq d + k.$$

Pelo Teorema 3.35, todas as diagonais que passam por pontos de C são marcadas. Finalmente, esta conclusão vale para todo caminho ótimo C. □

Algoritmo Constrói-Des
Entrada: padrão A e seu respectivo tamanho m.
Saída: Tabela *Des*.

para cada $a \in \Sigma$ **faça**
início
  $\mathrm{I}(a) \leftarrow m + 1$
  para $i \leftarrow m - k$ **até** m **faça** $\mathrm{Des}(i, a) \leftarrow i$
fim
para $h \leftarrow m$ **descendo até** 1 **faça**
início
  para $i \leftarrow \mathrm{I}(A(h)) - 1$ **descendo até** $\max\{h, m - k\}$ **faça**
    $\mathrm{Des}(i, A(h)) \leftarrow i - h$
  $\mathrm{I}(A(h)) \leftarrow h$
fim

Figura 3.14: Algoritmo *Constrói-Des*

Vamos construir esta seção com o algoritmo *Constrói-Des*, Figura 3.14, que determina corretamente a matriz Des. O valor de $\mathrm{I}(a)$, para cada $a \in \Sigma$, indica a ocorrência de a mais à direita no padrão, porém não à direita da posição h corrente.

3.4.4 Análise de complexidade

Indicaremos o tamanho do alfabeto Σ por c. Assim, o pré-processamento do padrão A para a obtenção da tabela Ruim possui uma complexidade $\mathrm{O}((k + c) \cdot m)$.

A obtenção da tabela Des possui complexidade $\mathrm{O}(k(m+c))$. Claramente, o espaço ocupado pelas duas tabelas é $\mathrm{O}(cm)$.

O algoritmo *Diferenças-TU* possui complexidade $\mathrm{O}((n/k) \cdot m)$, pois, podemos incrementar d de pelo menos $k + 1$ em cada iteração e o laço interno pode ser executado no pior caso $m - k$ vezes.

Observe que a complexidade do algoritmo *Diferenças-TU* quando comparado com os algoritmos apresentados anteriormente não é tão satisfatório.

Entretanto, a análise do caso médio da complexidade de tempo do algoritmo é:

$$O\left(\frac{c}{c-2k}\cdot k\cdot n\left(\frac{k}{c+2k^2}+\frac{1}{m}\right)\right),$$

conforme demonstrado em [TU91].

Nas figuras 3.15 e 3.16 apresentamos alguns gráficos resultantes dos testes realizado em [TU91] para analise empírica do algoritmo. Os teste foram realizados usando-se textos aleatórios com comprimento de 100.000 caracteres e alfabetos de diferentes tamanhos. O tempo de execução é dado em unidades de 10 milisegundos. Nestes gráficos são apresentados também o desempenho do algoritmo de Ukkonen [Ukk85].

Na figura 3.15 é exibido o tempo de execução do algoritmo quando o tamanho do padrão e do alfabeto variam para $k = 4$.

Na figura 3.16 é exibido o tempo de execução quando k e o tamanho do alfabeto variam para $m = 8$.

Note que o algoritmo possui um desempenho melhor com alfabetos maiores e o tempo de execução decresce à medida que o padrão aumenta. Essas características devem-se à utilização da heurística do algoritmo de Boyer-Moore. No entanto, um aumento em k diminui o desempenho do algoritmo.

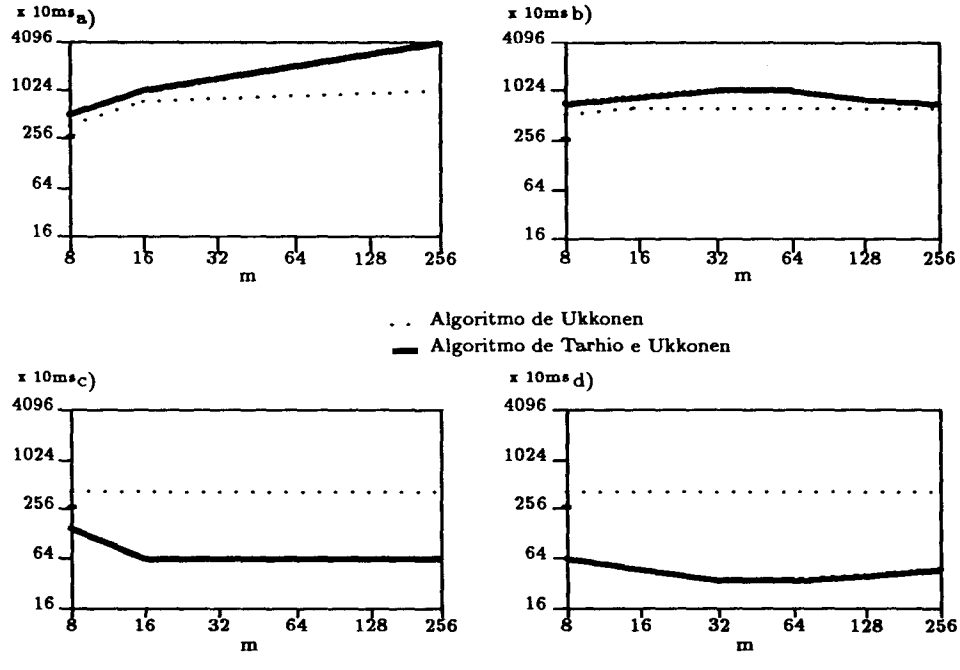

Figura 3.15: O valor de k é 4 enquanto que m varia e o tamanho do alfabeto é $2, 4, 30, 90$ para as figuras a, b, c e d respectivamente.

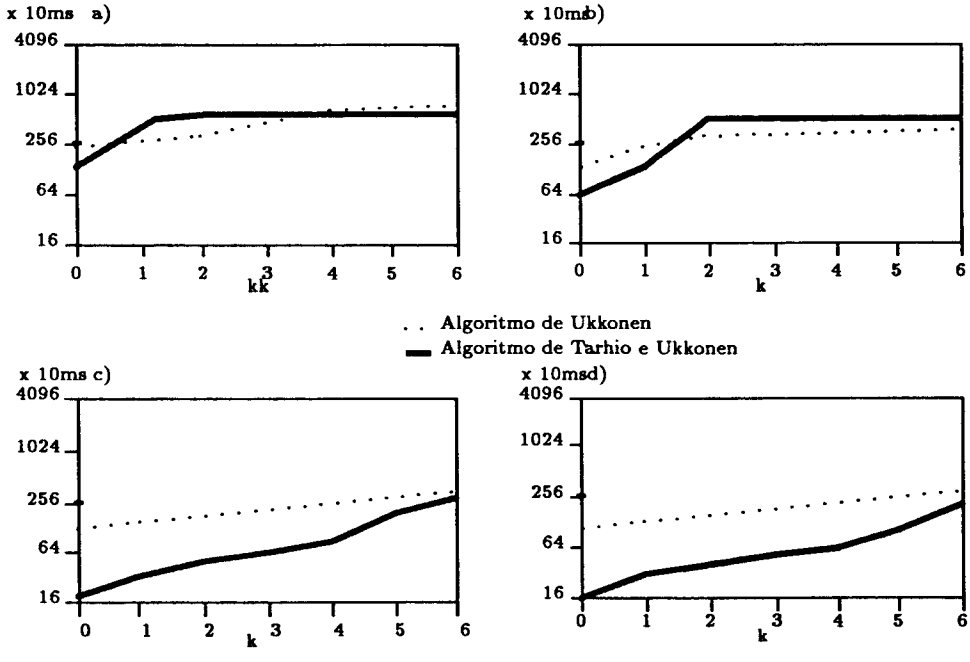

Figura 3.16: O valor de m é 8 enquanto que k varia e o tamanho do alfabeto é $2, 4, 30, 90$ para as figuras a, b, c e d respectivamente.

Capítulo 4

Algoritmos para casamento aproximado com k colisões

Neste capítulo abordaremos o problema das k colisões, ou seja, desejamos encontrar ocorrências do padrão no texto com no máximo k colisões. Note que o problema das k colisões é uma particularização do problema das k diferenças, isto é, no problema das k colisões não são permitidas as operações de inserção e remoção. Como sempre, suporemos um padrão A de comprimento m, um texto B de comprimento n, tais que $m \leq n$. Obviamente podemos também supor que $k < m$, caso contrário todos os alinhamentos são satisfatórios.

Neste capítulo serão apresentados os algoritmos de Landau e Vishkin [LV86], Baeza-Yates [BY89] e Tarhio e Ukkonen [TU91].

Landau e Vishkin desenvolveram a primeira solução para o problema utilizando tempo $O(kn)$ para procurar as ocorrências aproximadas e para pré-processar o padrão o algoritmo utiliza tempo $O(km \log m)$. Além disso, o algoritmo requer $O(k(n+m))$ de espaço.

Baeza-Yates desenvolveu uma outra solução baseada em uma heurística do algoritmo de Boyer e Moore [BM77] cuja complexidade de tempo para encontrar as ocorrências aproximadas é, no pior caso, $O(mn)$. No entanto, no caso médio o algoritmo apresenta um desempenho um pouco melhor. A complexidade de espaço do algoritmo de Baeza-Yates é $O(m-k)$.

Tarhio e Ukkonen também desenvolveram uma solução baseada em uma heurística do algoritmo de Boyer e Moore utilizando, no caso médio, $O(kn(1/(m-k)+k/|\Sigma|))$ para procurar as ocorrências do padrão no texto.

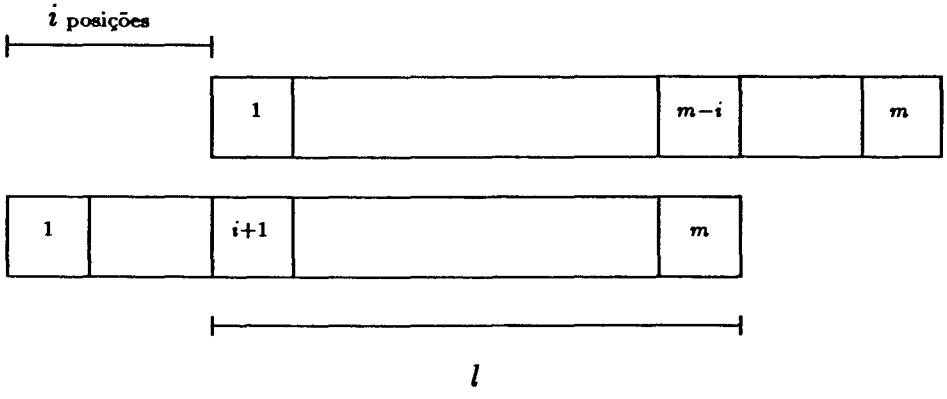

Figura 4.1: O pré-processamento devolverá as colisões existente no intervalo l.

No pior caso o algoritmo é O(mn). Para pré-processar o padrão o algoritmo utiliza $O(m + k\,|\Sigma|)$. Além disso, o algoritmo requer um espaço de $O(kc)$.

4.1 Algoritmo de Landau e Vishkin

O algoritmo descrito nesta seção é composto por duas partes: o pré-processamento do padrão e a análise do texto.

O pré-processamento do padrão fornecerá as colisões existentes na sobreposição de duas cópias do padrão deslocadas de i posições, $1 \leq i \leq m-1$. Em outras palavras, o pré-processamento devolverá as colisões existentes entre $A(1 \ldots m-i)$ e $A(i+1 \ldots m)$. Veja a figura 4.1.

A análise do texto utilizará as informações obtidas pelo pré-processamento do padrão e, conforme veremos a seguir, utilizará também as informações de alinhamentos entre o texto e o padrão que já tenham sido analisados.

Sejam a e b dois alinhamentos entre o texto e o padrão tal que o alinhamento a esteja à esquerda do alinhamento b. Além disso, suponha que o alinhamento a já tenha sido analisado, ou seja, as colisões entre o padrão e o texto no alinhamento a já são conhecidas. O alinhamento b é o alinhamento que devemos analisar. Veja a figura 4.2.

No alinhamento b teríamos que comparar os caracteres do padrão com

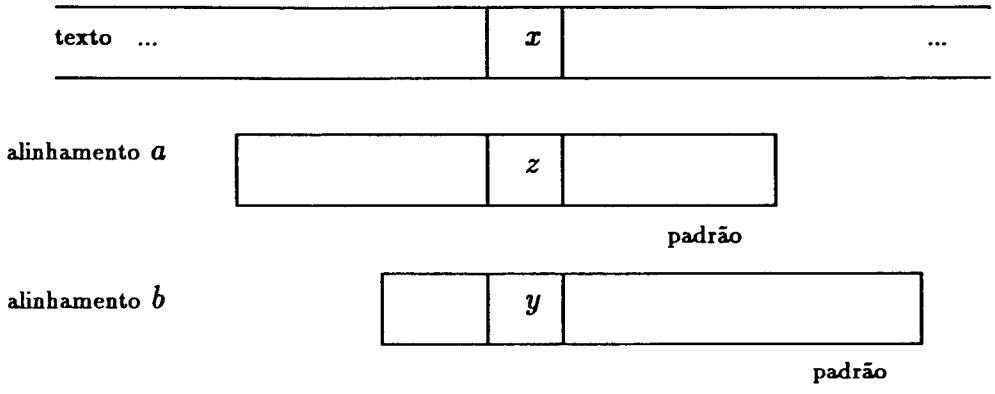

Figura 4.2: Alinhamentos a e b.

os do texto a partir do início desse alinhamento, no entanto, como veremos a seguir, em alguns casos podemos evitar algumas dessas comparações.

Observe que os alinhamentos a e b correspondem a um alinhamento entre duas cópias do padrão deslocadas de algumas posições. Além disso, note que pelo pré-processamento do padrão sabemos quais as colisões existentes na sobreposição das cópias do padrão, isto é, se y e z são caracteres do padrão alinhados na sobreposição (veja a figura 4.2) então sabemos se eles são iguais ou não.

Além disso, temos as colisões existentes entre o texto e o padrão no alinhamento a, ou seja, neste alinhamento sabemos se um caracter x do texto é igual ou não a um caracter z do padrão.

Portanto, com as informações acima podemos evitar comparações entre o texto e o padrão no alinhamento b. Para isto, basta fazer uma análise de casos.

Suponha que os caracteres x do texto, y e z do padrão estejam alinhados (figura 4.2). Podemos decidir se existe uma colisão entre x e y através da análise dos seguintes casos:

- se $x = z$ e $z = y$ então $x = y$;
- se $x = z$ e $z \neq y$ então $x \neq y$;
- se $x \neq z$ e $z = y$ então $x \neq y$.

Agora se $x \neq z$ e $z \neq y$ então devemos comparar x com y.

Descreveremos, com maiores detalhes, a análise do texto na seção 4.1.1 e o pré-processamento do padrão será descrito na seção 4.1.2.

4.1.1 Análise do texto

Na fase de análise do texto precisamos de duas tabelas. Uma que contém as informações provenientes do pré-processamento do padrão (tabela PD) e outra (tabela TC) que armazenará as colisões existentes entre o texto e o padrão obtidas em iterações anteriores durante a execução do algoritmo. A tabela PD possui $m-1$ linhas e $2k+1$ colunas, enquanto que a tabela TC possui $n-m+1$ linhas e $k+1$ colunas. Descreveremos a seguir essas duas tabelas.

A linha i da tabela PD contém as posições das primeiras $2k+1$ colisões existentes entre $A(1 \ldots m-i)$ e $A(i+1 \ldots m)$, ou seja, contém as colisões existentes entre duas cópias do padrão deslocadas de i posições. Assim, se $A(l) \neq A(i+l)$ e esta é a u-ésima colisão, da esquerda para a direita, com $1 \leq u \leq 2k+1$, que existe na sobreposição de $A(1 \ldots m-i)$ com $A(i+1 \ldots m)$, então $\mathrm{PD}(i,u) = l$. Mais adiante veremos por que devemos armazenar $2k+1$ colisões para cada uma destas sobreposições do padrão.

Agora, seja c o número de colisões entre $A(1 \ldots m-i)$ e $A(i+1 \ldots m)$. Se c for menor do que $2k+1$ então atribuiremos o valor $m+1$ em PD para a coluna $c+1$, que funcionará como uma "sentinela". As colunas seguintes, de $c+2$ até $2k+1$, ficam com valor indefinido.

A linha i da tabela TC contém as primeiras $k+1$ posições em A que levam a uma colisão entre $B(i+1 \ldots i+m)$ e $A(1 \ldots m)$, ou seja, contém as $k+1$ primeiras colisões do alinhamento do padrão com o texto na posição $i+1$. Assim, se $B(i+l) \neq A(l)$ e esta é a v-ésima colisão, da esquerda para a direita, com $1 \leq v \leq k+1$, que existe entre $B(i+1 \ldots i+m)$ e $A(1 \ldots m)$, então $\mathrm{TC}(i,v) = l$.

Agora, seja c o número de colisões entre $B(i+1 \ldots i+m)$ e $A(1 \ldots m)$. Se c for menor do que $k+1$ então atribuiremos o valor $m+1$ em TC para a coluna $c+1$, que também funcionará como uma "sentinela". As colunas seguintes, de $c+2$ até $2k+1$, ficam com valor indefinido.

Na figura 4.3 temos um exemplo para a tabela TC, onde $B = abaadccb$, $A = adbc$ e $k = 2$. Na tabela da figura 4.3 a primeira linha exibe as colisões no seguinte alinhamento:

	1	2	3
0	2	3	4
1	1	2	3
2	2	3	5
3	3	5	?
4	1	2	3

Figura 4.3: Tabela TC para $A = adbc$ e $B = abaadccb$.

a b a a d c c b
a d b c

A análise do texto consiste em verificar a cada iteração i, $0 \leq i \leq n-m$, o alinhamento do padrão na posição $i+1$ do texto. Em cada alinhamento compararemos texto e padrão, caracter a caracter, até encontrarmos $k+1$ colisões, ou até encontrarmos uma ocorrência com no máximo k colisões. Sempre que possível evitaremos efetuar as comparações entre o texto e o padrão utilizando as tabelas TC e PD. Após a análise de um alinhamento deslocaremos o padrão de uma posição. Além disso, à medida que analisamos um alinhamento atualizaremos a tabela TC.

Agora descreveremos como o algoritmo utiliza as tabelas TC e PD.

Seja i uma iteração onde verificaremos a ocorrência do padrão na posição $i+1$ do texto. Como o método prevê a utilização das iterações anteriores então é conveniente escolher uma iteração r, $0 \leq r < i$, tal que a última posição j_r do texto analisada na iteração r seja a mais à direita possível. Em outras palavras, escolheremos r tal que j_r seja máximo para $0 \leq r < i$. Por simplicidade denotaremos j_r por j. Veja a figura 4.4 que ilustra a iteração i e a iteração r.

Temos algumas condições que devem ser analisadas: $j > i$ e $j \leq i$. Se $j \leq i$, ou seja, o caso em que as análises das posições entre o texto e o padrão na iteração r não ultrapassam a posição i, então na iteração i devemos comparar texto e padrão, caracter a caracter.

Se $j > i$, isto é, quando as análises da iteração r ultrapassaram a posição i, (indicado por s_1 na figura 4.4) então podemos utilizar as informações das tabelas TC e PD para analisar o intervalo do texto correspondente a s_1.

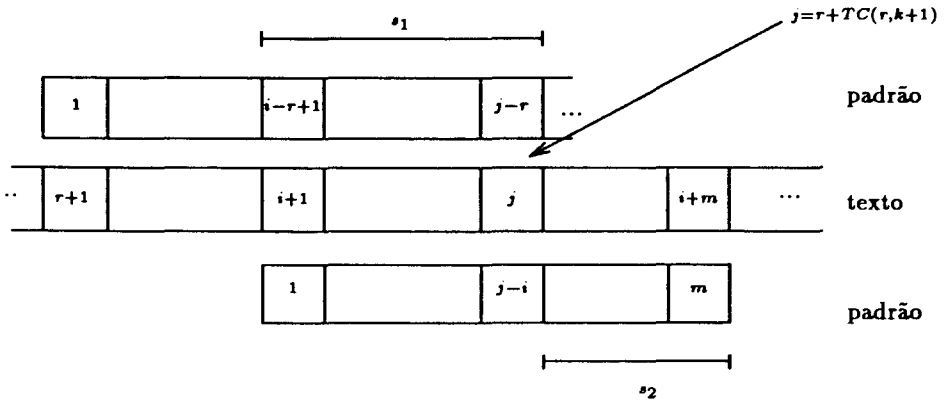

Figura 4.4: A iteração r é anterior à iteração i. O intervalo s_1 é a parte que já foi comparada com o padrão na iteração r. O intervalo s_2 é a parte não sobreposta pelas iterações i e r.

Para analisar este intervalo necessitamos da linha r da tabela TC, ou seja, $\mathrm{TC}(r, 1 \ldots k+1)$, pois, esta linha indica as colisões na iteração r. Mais especificamente, necessitamos apenas das posições do texto maiores do que i onde as colisões ocorreram na iteração r, pois, é a partir da posição $i+1$ do texto que começa a análise na iteração i. Então os valores apropriados da tabela TC para analisar o intervalo em questão são $\mathrm{TC}(r, q \ldots k+1)$ onde q é o menor inteiro tal que $r + \mathrm{TC}(r, q) > i$.

Da tabela PD necessitamos da linha $i - r$, já que os alinhamentos da iteração i e r posicionam as cópias do padrão deslocadas de $i - r$ posições. Necessitamos, então, de $\mathrm{PD}(i - r, 1 \ldots t)$ onde t é o menor possível tal que ou $\mathrm{PD}(i - r, t) > j - i$ ou $t = 2k + 1$ (veja a figura 4.5).

Seja p uma posição no texto tal que $i+1 \leq p \leq j$ (figura 4.5). A posição p pode satisfazer uma, nenhuma ou ambas as condições abaixo:

a) considerando a iteração r (padrão alinhado na posição $r+1$), a posição p corresponde a uma colisão, ou seja, $B(p) \neq A(p - r)$ e esta é a v-ésima colisão para $q \leq v \leq k + 1$, isto é, $\mathrm{TC}(r, v) = p - r$;

b) com as duas cópias do padrão deslocadas de $i - r$ posições, existe uma colisão entre os respectivos caracteres do padrão alinhados com $B(p)$,

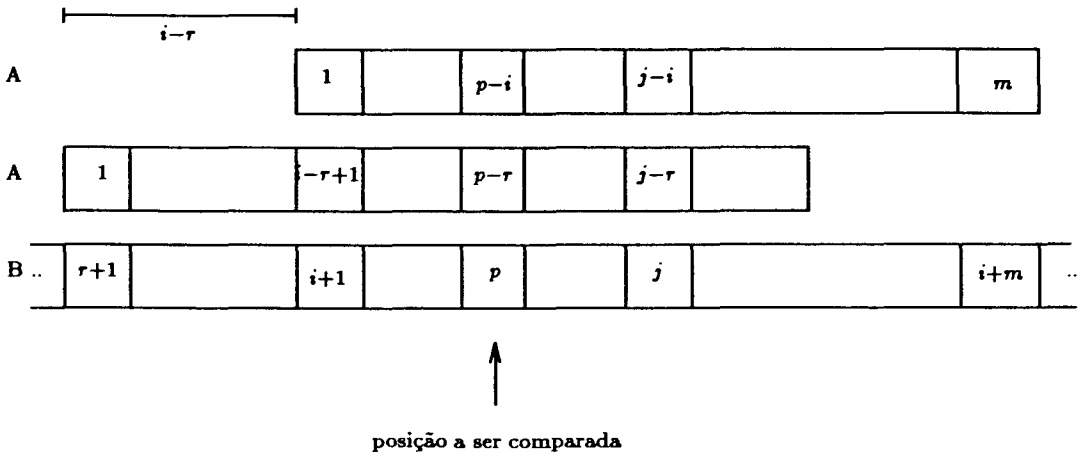

Figura 4.5: Iterações i e r.

isto é, $A(p-r) \neq A(p-i)$ e esta é a u-ésima colisão, $1 \leq u \leq t$, entre as cópias do padrão para este deslocamento, ou seja, $\mathrm{PD}(i-r, u) = p-i$.

Como estamos analisando a iteração i (alinhamento do padrão na posição $i+1$ do texto) então devemos decidir se $B(p)$ colide ou não com $A(p-i)$. Para isto, basta fazer uma análise de casos com relação as condições (a) e (b):

- caso 1: a posição p não satisfaz nem (a) e nem (b): então $B(p) = A(p-r)$ e $A(p-r) = A(p-i)$. Assim, $B(p) = A(p-i)$;
- caso 2: a posição p satisfaz apenas uma das duas condições. No caso em que apenas a condição (a) é satisfeita temos que $B(p) \neq A(p-i)$, pois $B(p) \neq A(p-r)$ e $A(p-r) = A(p-i)$. Analogamente, o caso em que somente a condição (b) é satisfeita então $B(p) \neq A(p-i)$, pois, $B(p) = A(p-r)$ e $A(p-r) \neq A(p-i)$;
- caso 3: ambas as condições são satisfeitas. Neste caso não temos informações suficientes para decidir se $B(p)$ é igual ou não a $A(p-i)$. Portanto devemos fazer uma comparação direta entre $B(p)$ e $A(p-i)$.

Observe que nos casos acima se uma colisão for encontrada então podemos atualizar a tabela TC da seguinte forma: $\mathrm{TC}(i, b) = p-i$, onde b indica o número da colisão encontrada.

Uma maneira particular de implementar o método acima é utilizar uma estratégia semelhante à estratégia do algoritmo de ordenação "merge-sort", ou seja, intercalar duas seqüências crescentes de valores em uma única seqüência.

Uma das seqüências utilizadas é a seqüência de posições que levam a colisões entre as duas cópias do padrão deslocadas de $i-r$ posições, ou seja,

$$\mathrm{PD}(i-r,1),\ldots,\mathrm{PD}(i-r,t).$$

A outra seqüência utilizada é a seqüência de posições no padrão (considere o padrão alinhado na posição $i+1$ do texto) que estão alinhadas com as posições no texto que levaram a colisões entre o texto e o padrão na iteração r, ou seja,

$$r+\mathrm{TC}(r,q)-i,\ldots,r+\mathrm{TC}(r,k+1)-i.$$

Ao invés de intercalarmos as duas seqüências exatamente como é realizado no algoritmo "merge-sort", verificaremos se cada posição analisada pertence a um dos três casos (casos 1,2 e 3).

Por exemplo, se $\mathrm{PD}(i-r,1) > r+\mathrm{TC}(r,q)-i$ então isto significa que a posição $r+\mathrm{TC}(r,q)-i$ está à esquerda da posição $\mathrm{PD}(i-r,1)$. Em outras palavras, a primeira colisão entre o texto e o padrão na iteração r no intervalo $[i+1,j]$ ocorre antes da primeira colisão entre as duas cópias do padrão deslocadas de $i-r$ posições. Logo, pelo caso 1 temos que $\mathrm{TC}(i,1) = r+\mathrm{TC}(r,q)-i$.

Podemos repetir esse método de analisar os casos até que uma das seguintes restrições sejam satisfeitas:

1. $b = k+1$, isto significa que foram encontradas $k+1$ colisões;

2. $v = k+2$, pois, v deve estar no intervalo $[q,k+1]$; ao atingir o valor $k+2$ significa que passamos da posição j no texto e comparações diretas entre o padrão e o texto precisam ser feitas;

3. $\mathrm{PD}(i-r,u) > j-i$ e $\mathrm{TC}(r,v) = m+1$. Isto ocorre quando todas as posições do alinhamento r que causavam uma colisão entre o texto e o padrão e todas as posições que levavam a colisões entre as cópias do padrão à esquerda da posição j no texto (figura 4.5) já foram analisadas. Novamente, comparacões diretas entre o padrão e o texto precisam ser feitas.

Algoritmo Colisões-LV
Entrada: Padrão $A(1 \ldots m)$ e Texto $B(1 \ldots n)$.
Saída: Tabela $\mathrm{TC}(0 \ldots n-m, 1 \ldots k+1)$.

```
r ← 0
j ← 0
para i ← 0 até n − m faça
início
   b ← 0
   se i < j então Intercala-Iterações(i, r, j, b)
   se b < k + 1 então
   início
      r ← i                                          ≪ 1 ≫
      Ingênuo(i, j, b)
      se b < k + 1 então TC(i, b + 1) ← m + 1
   fim
fim
```

Figura 4.6: Algoritmo *Colisões-LV*

O algoritmo que implementa estas idéias está na figura 4.8.

A figura 4.6 exibe o algoritmo *Colisões-LV*. Neste algoritmo, usamos o algoritmo *Intercala-Iterações* (figura 4.8) enquanto as posições do texto são menores do que ou iguais a j. As posições à direita de j são analisadas pelo algoritmo *Ingênuo* (figura 4.7) que também atualiza o valor de j.

Agora mostraremos que as $2k+1$ colisões armazenadas pela tabela PD são suficientes para que o algoritmo *Intercala-Iterações* encontre todas as colisões, ou pelo menos $k+1$ colisões, no intervalo $[i+1, j]$.

O valor de u é incrementado em 2 comandos do algoritmo *Intercala-Iterações*, a saber, no caso 2, condição (b), e nos casos 1 e 3. No caso 2 condição (a) ou condição (b), o valor b também é incrementado. Nos casos 1 ou 3, v também é incrementado. Depois de no máximo $2k+1$ incrementos de u, ou b foi incrementado k vezes (e portanto $b = k+1$) ou v foi incrementado $k+1$ vezes (e portanto $v = k+2$).

Em ambos os casos a execução do algoritmo *Intercala-Iterações* termina.

Algoritmo Ingênuo
Entrada: índices i (iteração), j (texto) e b (número de colisões)
Saída: Tabela TC atualizada.

enquanto $(b < k+1)$ **e** $(j \leq i+m)$ **faça**
início
  $j \leftarrow j+1$
  se $B(j) \neq A(j-i)$ **então**
  início
    $b \leftarrow b+1$
    $\mathrm{TC}(i,b) \leftarrow j-i$
  fim
fim

Figura 4.7: Algoritmo *Ingênuo*

De fato, durante a execução desse algoritmo, $1 \leq u \leq 2k+1$.

Assim, as $2k+1$ posições armazenadas pela tabela PD são suficientes para encontrar todas as colisões, ou pelo menos $k+1$ colisões.

O leitor atento deve ter observado que, na verdade, apenas duas linhas da matriz TC são necessárias para o algoritmo: a linha r e a linha i. Assim, pode-se alterar o algoritmo, mantendo apenas dois vetores de $k+1$ posições, um correspondendo à linha r, o outro, à linha i. A atualização do primeiro é feita atribuindo-lhe o valor do segundo, no lugar da atribuição $r \leftarrow i$ (ponto $\ll 1 \gg$) no algoritmo *Colisões-LV*.

A seguir apresentaremos o pré-processamento do padrão.

4.1.2 Pré-processamento do padrão

Descreveremos agora o pré-processamento do padrão onde calcularemos a tabela $\mathrm{PD}(1 \ldots m-1, 1 \ldots 2k+1)$.

Convém recordar que $\mathrm{PD}(i,u) = l$ indica que $A(i+l) \neq A(l)$ e esta é a u-ésima colisão existente na sobreposição de duas cópias do padrão deslocadas de i posições.

Para calcular a tabela PD utilizaremos a mesma estratégia do algoritmo

Algoritmo Intercala-Iterações
Entrada: índices i (iteração), r (iteração anterior), j (última posição analisada) e b (número de colisões)
Saída: Tabela TC atualizada.

```
u ← 1
v ← min{q' : TC(r,q') > i − r} % i < j
enquanto b < k + 1 e v < k + 2 e (
            PD(i − r,u) ≤ j − i ou TC(r,v) < m + 1 ) faça
início
   se PD(i − r,u) > r + TC(r,v) − i então
   % caso 2, condição (a)
   início
      b ← b + 1
      TC(i,b) ← r+TC(r,v) − i
      v ← v + 1
   fim
   senão se PD(i − r,u) < r + TC(r,v) − i então
         início
            % caso 2, condição (b)
            b ← b + 1
            TC(i,b) ← PD(i − r,u)
            u ← u + 1
         fim
         % casos 1 ou 3
         senão início
                  se B(i + PD(i − r,u)) ≠ A(PD(i − r,u)) então
                  início% caso 3
                     b ← b + 1
                     TC(i,b) ← PD(i − r,u)
                  fim
                  u ← u + 1
                  v ← v + 1
               fim
fim
```

Figura 4.8: Algoritmo *Intercala-Iterações*

que realiza a análise do texto. Desta forma, necessitamos de um texto T e um padrão P para poder aplicar esta estratégia. Como estamos interessados em encontrar colisões entre duas cópias do padrão então P e T serão subseqüências do padrão A.

Vamos supor que m é uma potência de 2. O algoritmo de pré-processamento consiste de $\log_2 m$ (daqui para frente simplesmente $\log m$) *estágios.*

No primeiro estágio será calculada a primeira linha de PD. No segundo estágio serão calculados a segunda e terceira linhas. Em geral, no estágio l ($1 \leq l \leq \log m$) serão calculadas as seguintes 2^{l-1} linhas de PD:

$$2^{l-1}, 2^{l-1}+1, \ldots, 2^l - 1$$

ou seja, em cada estágio calcularemos as colisões existentes na sobreposição de duas cópias do padrão deslocadas de $2^{l-1}, 2^{l-1}+1, \ldots, 2^l - 1$ posições. Assim, podemos dizer que em cada estágio temos 2^{l-1} iterações.

Assim, para que possamos utilizar a mesma estratégia do algoritmo que realiza a análise do texto, definiremos, para cada estágio l, o "texto" T_l e o seu comprimento n_l da seguinte forma:

$$\begin{aligned} T_l &:= A(2^{l-1}+1 \ldots m) \\ n_l &:= m - 2^{l-1}. \end{aligned}$$

Para cada iteração i, $1 \leq i \leq 2^{l-1}$, do estágio l definiremos o "padrão" P_{l_i} e o seu comprimento m_{l_i} da seguinte forma:

$$\begin{aligned} P_{l_i} &:= A(1 \ldots m - (2^{l-1}+i)+1) \\ m_{l_i} &:= m - (2^{l-1}+i)+1. \end{aligned}$$

A tabela resultante do estágio l é denotada por TC_l. Esta tabela armazenará as colisões entre T_l e P_{l_i}, $1 \leq i \leq 2^{l-1}$. A matriz auxiliar que será usada no cálculo de TC_l e que faz o papel de PD será denotada por PD_l. Note que no primeiro estágio ($l = 1$) PD_1 é irrelevante, pois, não temos estágios anteriores de onde poderíamos utilizar as informações para realizar a análise de casos do algoritmo *Colisões-LV*. Assim, o estágio 1 consiste apenas de uma iteração, em que apenas o algoritmo ingênuo é utilizado. Para os estágios seguintes, em que $l > 1$,

$$\mathrm{PD}_l = \mathrm{TC}_{l-1}.$$

Resta agora definir o limite de colisões k_{l_i} no estagio l para cada iteração i:

$$k_{l_i} := \min\{2^{1-l+\log m}, m_{l_i}\}.$$

Vamos agora justificar a definição de k_{l_i}. Evidentemente, não faz sentido ter mais colunas em TC_l do que $1 + m_{l_i}$. Assim, k_{l_i} deve ser o mínimo entre m_{l_i} e algum outro parâmetro. Pelos mesmos argumentos de que são necessárias $2k + 1$ colisões em PD para obter k colisões para a tabela TC, e pelo fato de que TC_{l-1} =PD_l temos então que o número de colisões, no estágio l, que devem ser armazenadas por TC_l é $2^{-l+\log m}2k + 1$. Desta forma, no último estágio ($l = \log m$) são armazenadas $2k + 1$ colisões em $TC_{\log m}$. Na figura 4.9 ilustramos a versão do algoritmo *Colisões-LV* para pré-processar o padrão.

Apresentaremos a seguir a análise de complexidade do algoritmo de *Colisões-LV* e do algoritmo de pré-processamento do padrão.

4.1.3 Análise de Complexidade

Para cada iteração i do algoritmo *Colisões-LV* o número de operações realizadas é dado pelo número de comparações efetuadas pelo algoritmo *Ingênuo* mais o número de operações efetuadas pelo algoritmo *Intercala-Iterações.*

O número total de operações efetuadas pelo algoritmo *Ingênuo*, no pior caso, é $O(n)$.

Já o número de operações, por iteração, efetuadas pelo algoritmo *Intercala-Iterações* é dada pelo número de consultas à tabela TC mais o número de consultas à tabela PD.

O total de elementos nessas tabelas é $3k + 2$. Além disso, as análises são realizadas em ordem crescente de posições no texto. Assim, o número de operações, neste algoritmo, é $O(k)$.

Como o número de iterações i é $O(n)$ então a complexidade do algoritmo de *Colisões-LV* é $O(kn)$.

A análise de complexidade do pré-processamento é feita de maneira análoga.

O número total de comparações realizadas pelo algoritmo *Ingênuo2* é $O(m)$.

Já o número total de operações realizadas pelo algoritmo *Intercala-Iterações2* é $O(\min\{2^{\log m-l+1}k + 1, m - 2^l\}) = O(2^{\log m-l+1}k + 1)$.

Algoritmo Analise-Padrão
Entrada: Padrão $A(1 \ldots m_{l_i})$ e texto $A(2^{l-1}+1 \ldots m)$.
Saída:PD$(2^{l-1}+1 \ldots 2^l, 1 \ldots \min\{2^{\log m - l + 1}k + 1, m - 2^l\})$.
Obs: Os algoritmos *Intercala-Iterações2* e Algoritmo *Ingênuo2* são simples modificações dos algoritmos utilizados no algoritmo *Colisões-LV* e, portanto, não serão descritos.

$r \leftarrow 0$
$j \leftarrow 0$
para $i \leftarrow 2^{l-1}$ **até** $2^l - 1$ **faça**
início
  $b \leftarrow 0$
  se $i < j$ **então** *Intercala-Iterações2*(i, r, j, b)
  $m_{l_i} \leftarrow m - i + 1$
  se $b < \min\{2^{1-l+\log m}, m_{l_i}\}$ **então**
  início
    $r \leftarrow i$
    Ingênuo(i, j, b)
    se $b < \min\{2^{1-l+\log m}, m_{l_i}\}$ **então** TC$(i, b+1) \leftarrow m + 1$
  fim
fim

Figura 4.9: Algoritmo *Análise-Padrão*

Como no estágio l ($2^{l-1} \leq i \leq 2^l$) temos 2^{l-1} iterações então o número de operações no estágio l é $O(2^{l-1}(2^{\log m - l + 1}k + 1)) = O(km)$.

Além disso, como $1 \leq l \leq \log m$ então o pré-processamento do padrão é $O(\sum_{l=1}^{\log m} km) = O(km \log m)$.

Finalmente a complexidade de espaço é $O(k(n+m))$ para armazenar as tabelas TC e PD.

Note que o algoritmo *Colisões-LV* sempre desloca o padrão de uma posição para procurar uma ocorrência no texto. De certa forma está estratégia não é muito eficiente, pois, conforme mostraremos, podemos deslocar o padrão mais de uma posição.

4.2 Algoritmo de Baeza-Yates

O algoritmo de Baeza-Yates [BY89] descrito nesta seção utiliza a mesma estratégia do algoritmo de Boyer e Moore [BM77]. O método utilizado neste algoritmo realiza as comparações entre o texto e o padrão da direita para a esquerda, até encontrar $k+1$ colisões ou até que as comparações entre o texto e o padrão terminem resultando em uma ocorrência aproximada. Neste ponto, utilizando uma tabela, deslocaremos o padrão para a direita para analisarmos um outro alinhamento.

Suponha que em um alinhamento entre o texto e o padrão foram efetuadas j comparações tal que k dessas comparações resultaram em colisões, ou seja, as posições $m-j+1,\ldots,m$ do padrão foram comparadas com o texto e k colisões foram encontradas. Além disso, suponha que a próxima comparação também resulte em uma colisão, logo, devemos deslocar o padrão. Ou alternativamente, suponha que o alinhamento corresponda a uma ocorrência aproximada do padrão com até k colisões. Ao contrário do deslocamento de uma posição realizado pelo algoritmo de Landau e Vishkin, neste algoritmo o deslocamento poderá ser maior.

Suponha que deslocamos o padrão de i posições onde $1 \leq i \leq m-1$ posições. Note que qualquer deslocamento do padrão menor do que m posições dispõe duas cópias do padrão alinhadas entre si de tal forma que exista uma sobreposição entre elas. Agora, considere duas cópias do padrão sobrepostas devido ao deslocamento de i posições. Analisaremos o caso em que existem pelo menos $2k+1$ colisões entre as cópias do padrão. Assim, suponha que existam pelo menos $2k+1$ colisões entre estas duas cópias e que estas colisões estejam no intervalo $[m-j+1, m]$, ou seja, as colisões estão no intervalo que foi comparado antes do deslocamento.

Veja que se deslocarmos o padrão de i posições tal que existam pelo menos $2k+1$ posições que levam a colisões entre as cópias do padrão no intervalo $[m-j+1, m]$ então podemos ter algumas dessas $2k+1$ posições coincidindo com algumas das k posições, do alinhamento anterior, que levam a colisões entre o texto e o padrão.

Veremos agora como se relacionam os caracteres do padrão deslocado e do texto associados a essas posições coincidentes. Seja l uma posição de A no intervalo $[m-j+1, m]$. Nesse intervalo, pelo menos $2k+1$ posições correspondem a colisões entre posições do padrão distantes i posições; destas posições no máximo k correspondem a colisões entre o texto e o padrão na

primeira posição. Portanto, no novo alinhamento há pelo menos $k+1$ colisões entre padrão e texto. Mais especificamente, existem pelo menos $2k+1$ valores l no intervalo

$$m-j+1 \leq l \leq m,$$

tais que $A(l) \neq A(i+l)$. Existem também no máximo k valores l no mesmo intervalo tais que

$$A(l) \neq B(l+r),$$

onde $r+1$ é a posição do texto que no primeiro alinhamento estava alinhada com a posição 1 do padrão.

Assim, para pelo menos $k+1$ valores de l no referido intervalo,

$$B(l+r) = A(l) \neq A(i+l).$$

Logo, é inútil deslocar o padrão de i posições.

Como buscamos um deslocamento que forneça um alinhamento com no máximo k colisões então um deslocamento que forneça $2k+1$ colisões entre as cópias do padrão no intervalo $[m-j+1, m]$ não é viável. Por outro lado, um deslocamento que forneça menos do que $2k+1$ colisões entre as cópias do padrão poderá ser favorável a um alinhamento que leve a um casamento aproximado.

Assim, definiremos s_j como o menor deslocamento tal que sobrepondo duas cópias do padrão deslocadas de l posições, $1 \leq l \leq s_j - 1$, existam no máximo $2k$ colisões entre os caracteres do padrão no intervalo $[m-j+1, m]$, ou seja, um deslocamento de s_j posições é o deslocamento que fornece menos do que $2k+1$ colisões entre os caracteres do padrão. Observe que ao deslocarmos o padrão procuramos por uma reocorrência da parte já comparada que possivelmente possua menos do que k colisões.

Note que temos trivialmente que $s_j = 1$ para $1 \leq j \leq 2k$. Uma outra observação é que $s_{j+1} \geq s_j$, pois, se qualquer deslocamento l, $1 \leq l \leq s_j - 1$ fornece pelo menos $2k+1$ colisões no intervalo $[m-j+1, m]$ então para o intervalo $[m-j, m]$ um deslocamento l, $1 \leq l \leq s_j - 1$, também fornece pelo menos $2k+1$ colisões. Daí, $s_{j+1} \geq s_j$. Na figura 4.10 temos um exemplo para o cálculo de s_j onde $j = 6$ e $k = 2$, isto é, consideramos que foram comparados os 6 últimos caracteres do padrão.

Assim, uma maneira de calcular s_j é dispor duas cópias do padrão deslocadas de uma posição e deslorcamos continuamente uma das cópias

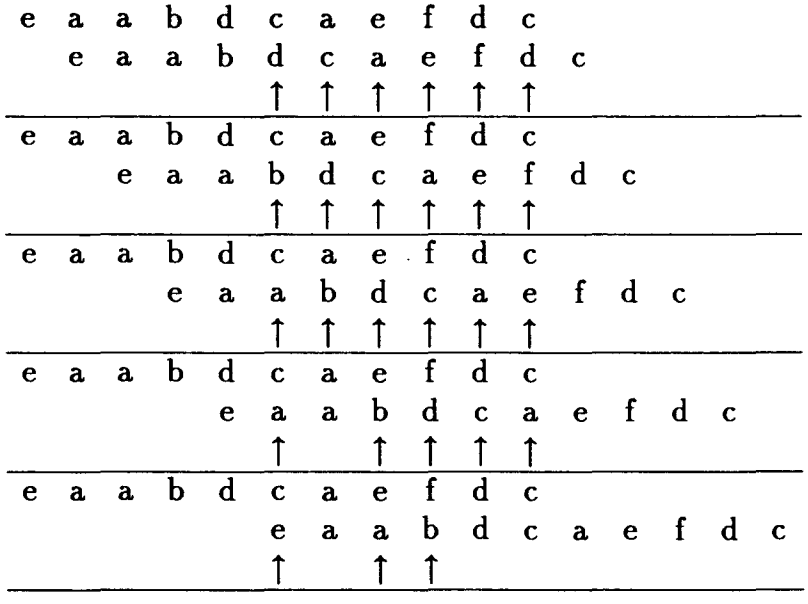

Figura 4.10: Cálculo para s_6 considerando que $k = 2$, ou seja, consideramos que os últimos 6 caracteres do padrão (caefdc) foram comparados. Nesta figura as setas indicam as colisões. Veja que $s_6 = 5$.

para direita até encontrarmos menos do que $2k+1$ colisões no intervalo $[m-j+1, m]$. Pela observação acima, para calcular s_{j+1} continuamos do deslocamento obtido em s_j, mas agora analisamos as colisões no intervalo $[m-j, m]$.

Como as comparações entre o texto e o padrão serão realizadas da direita para a esquerda então podemos calcular s_j considerando que j é posição do padrão que foi comparada. Desta forma, $s_j = 1$ para $m-2k \leq j \leq m$, pois, se $j \in [m-2k, m]$ então menos do que $2k$ caracteres foram comparados. Uma implementação do algoritmo para se calcular s_j é dada na figura 4.11.

Note que desta forma são analisadas $m-2k$ alinhamentos e como no pior caso podemos fazer $O(m)$ comparações em cada alinhamento então este algoritmo possui uma complexidade de $O(m(m-2k))$. Uma outra alternativa para obter s_j, conforme sugerido em [BY89], é utilizar as idéias do algoritmo de Knuth, Morris e Pratt [KMP77] obtendo um algoritmo com complexidade $O(km)$. A justificativa para a escolha do algoritmo descrito para obtenção de s_j é a de deixar mais evidente que a heurística de deslocamento baseia-se na mesma heurística da reocorrência do algoritmo de Boyer e Moore.

Assim, o algoritmo de Baeza-Yates para k colisões [BY89] consiste em comparar o padrão com o texto da direita para a esquerda até que sejam encontradas $k+1$ colisões ou até encontrar uma ocorrência aproximada. Neste ponto utilizamos a tabela s para deslocar o padrão e analisar outro alinhamento.

Este algoritmo, no pior caso, efetua $O(mn)$ comparações e requer um espaço de $O(m-2k)$ para armazenar a tabela s_j. Citamos a seguir alguns resultados sobre a complexidade média obtidos por Baeza-Yates [BY89].

A probabilidade de um caracter ocorrer em uma posição fixa de uma seqüência é $p = 1/c$, onde c é o tamanho do alfabeto.

Proposição 4.1 *[BY89] Seja $\overline{C}_m$ o número médio de comparações necessárias para decidir se existe no máximo k colisões entre duas seqüências de comprimento m. Então*

$$\overline{C}_m = \frac{k+1}{1-p} - O(p^{m-k}(1-p)^{k+1}m^k)\square$$

Teorema 4.2 *O número médio de comparações $\overline{C}_n$ efetuadas pelo algoritmo ingênuo para encontrar uma ocorrência aproximada com k colisões de um*

Algoritmo Calcula-s_j
Entrada: Padrão A
Saída: Tabela s_j.

```
para j ← m descendo até m − 2k faça s_i ← 1
para j ← m − 2k − 1 descendo até 0 faça
início
   s_j ← s_{j−1} − 1
   repita
      s_j ← s_j + 1
      colisoes ← 0
      i ← max{1, j − s_j}
      enquanto i ≤ m − s_j e colisoes ≤ 2k faça
      início
         se A(i) ≠ A(i + s_j) então colisoes ← colisoes + 1
         i ← i + 1
      fim
   até que colisoes ≤ 2k
fim
```

Figura 4.11: Algoritmo *Calcula-s_j*

padrão de tamanho m em um texto de tamanho n satisfaz a desigualdade

$$\overline{C}_n \leq 2(k+1)n \square$$

Teorema 4.3 *O número médio de comparações* $\overline{C}$ *do algoritmo de Baeza-Yates é tal que*

$$\overline{C} \geq \frac{\overline{C}_m}{1+(m-2k-1)P_{2k+1,k}}(n-m+1)$$

onde $P_{2k+1,k}$ *é a probabilidade de encontrar k colisões em* $2k+1$ *comparações.* □

Terminaremos esta seção com alguns resultados experimentais obtidos por Baeza-Yates [BY89].

Em média o algoritmo de Baeza-Yates [BY89] é um pouco melhor que o algoritmo ingênuo. Sua performance melhora à medida que o padrão aumenta de tamanho. A seguir exibiremos dois gráficos apresentados em [BY89] que ilustram o comportamento teórico e prático do algoritmo de Baeza-Yates.

A figura 4.12 exibe o comportamento do algoritmo do algoritmo ingênuo para um texto com 50.000 caracteres e um alfabeto de 32 símbolos.

A figura 4.13 exibe o comportamento do algoritmo de Baeza-Yates para k colisões. Note que o desempenho do algoritmo melhora à medida que o padrão aumenta.

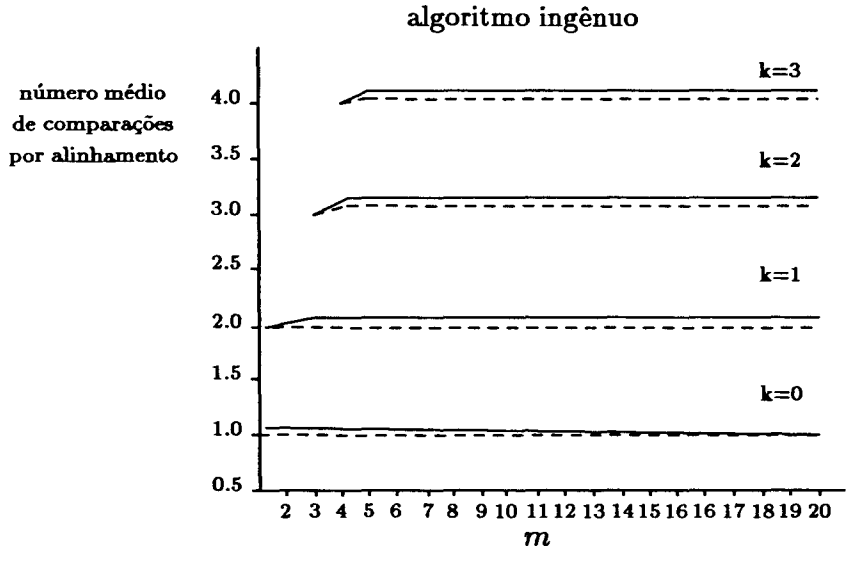

Figura 4.12: A linha tracejada indica o resultado teórico para um texto aleatório e a linha cheia indica o resultado experimental para um texto em inglês.

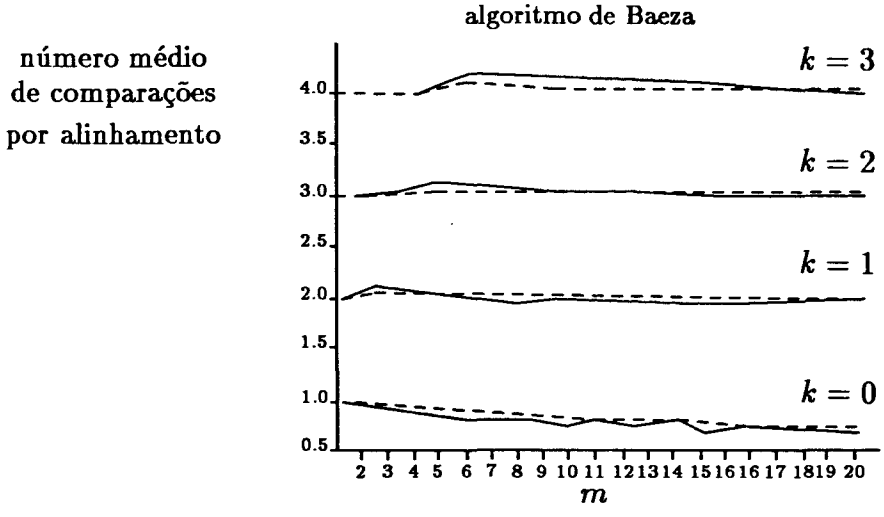

Figura 4.13: A linha tracejada indica o resultado teórico (limite inferior) para um texto aleatório e a linha cheia indica o resultado experimental para um texto em inglês.

4.3 Algoritmo de Tarhio e Ukkonen

O algoritmo descrito nesta seção é uma simplificação do algoritmo de Tarhio e Ukkonen [TU91] descrito na seção 3.4 do capítulo anterior. Assim como no algoritmo de Boyer e Moore, as comparações deste algoritmo serão realizadas da direita para a esquerda. Já os deslocamentos serão dados apenas pela heurística da tabela Des explicada no algoritmo da seção 3.4. Na figura 4.14 exibimos o algoritmo *Colisões-TU* para o problema das k colisões.

O número de comparações, no pior caso, para o algoritmo *Colisões-TU* é $O(mn)$. Isto ocorre, por exemplo, quando utilizamos $A = a^n$ e $B = a^m$. Citamos a seguir alguns resultados sobre a complexidade média obtidos por Baeza-Yates [BY89].

Consideraremos que o texto e o padrão são seqüências aleatórias e que os caracteres de A e B são escolhidos uniformemente em Σ.

Assim, seja C a variável aleatória que denota para c e k fixos o número de comparações em um alinhamento do padrão entre dois deslocamentos sucessivos. Seja $\overline{C}$ seu valor esperado.

Lema 4.4 $\overline{C} = \frac{k+1}{c-1}c$. □

Seja $S(B)$ a variável aleatória que denota o comprimento do deslocamento do padrão B para c e k fixos.

Lema 4.5 $\overline{S}(B_0) \geq \frac{1}{2}\min\{\frac{c}{k+1}, m-k\}$ □

Lema 4.6 *O número médio de comparações do algoritmo de Tarhio e Ukkonen é*

$$O(\frac{nk^2}{c} + \frac{nk}{m-k}).$$

A seguir apresentamos alguns gráficos resultantes do teste realizados em [TU91].Os teste foram realizados usando-se textos aleatórios com comprimento de 100.000 caracteres e alfabetos de diferentes tamanhos. O tempo de execução é dado em unidades de 10 milisegundos. Ambos os testes foram comparados com um algoritmo baseado em programação dinâmica para o problema de k colisões.

Na figura 4.15 é exibido o tempo de execução do algoritmo quando o tamanho do padrão e do alfabeto variam para $k = 4$.

Algoritmo Colisões-TU
Entrada: Tabela Des, padrão A e o texto B.
Saída: Posições no texto cujo alinhamento com o padrão resultam em uma ocorrência aproximada do padrão no texto.

$h \leftarrow m$
enquanto $j \leq n + k$ **faça**
início
  $j \leftarrow h$
  $i \leftarrow m$
  $colisoes \leftarrow 0$
  $\delta \leftarrow m - k$
  enquanto $i > 0$ **e** $colisoes \leq k$ **faça**
  início
    se $i \geq m - k$ **então** $\delta \leftarrow \min\{\delta, \mathrm{Des}(i, B(j))\}$
    se $A(i) \neq B(j)$ **então** $colisoes \leftarrow colisoes + 1$
    $i \leftarrow i - 1$
    $j \leftarrow j - 1$
  fim
  se $colisoes \leq k$ **então** devolva $h - m$
  $h \leftarrow h + \delta$
fim

Figura 4.14: Algoritmo *Colisões-TU*

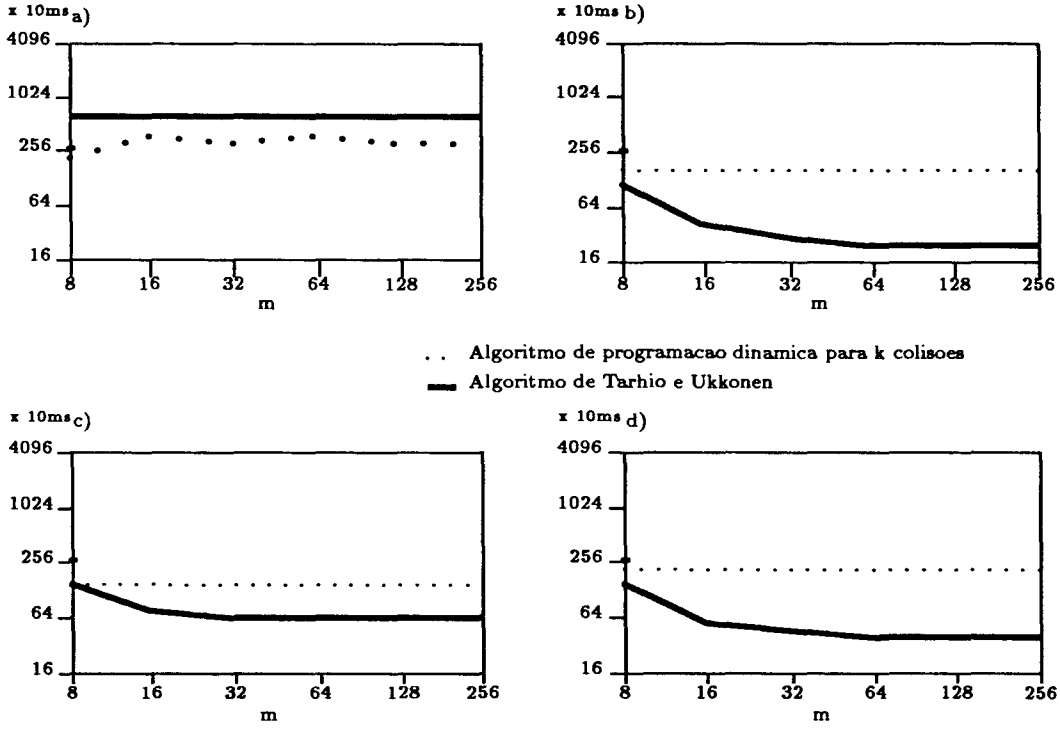

Figura 4.15: O valor de k é 4 enquanto que m varia e $c = 2, 4, 30, 90$ para as figuras a, b, c e d respectivamente.

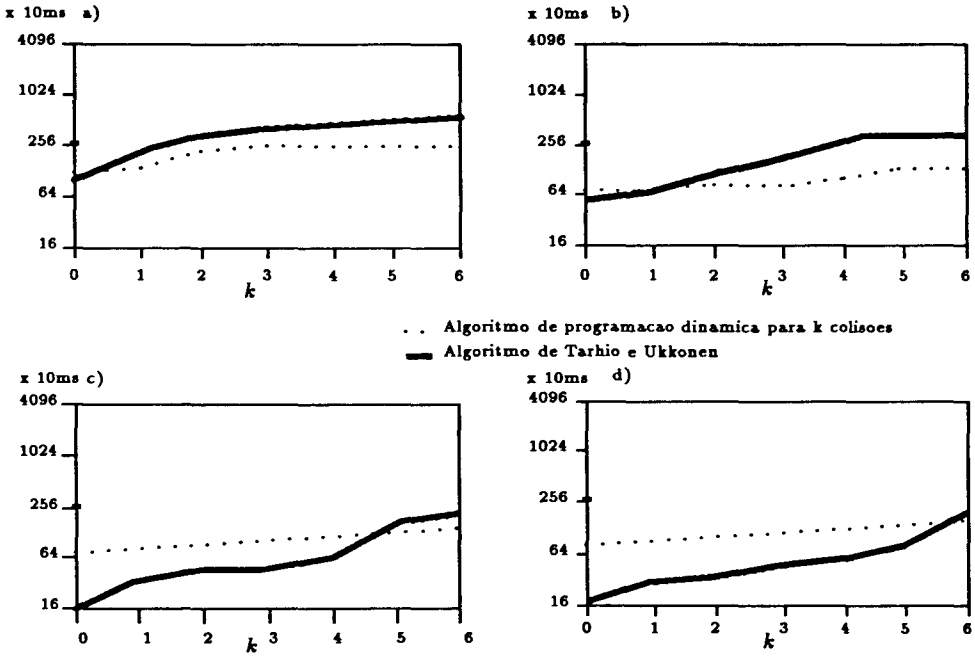

Figura 4.16: O valor de m é 8 enquanto que k varia e $c = 2, 4, 30, 90$ para as figuras a, b, c e d respectivamente.

Na figura 4.16 é exibido o tempo de execução quando k e o tamanho do alfabeto variam para $m = 8$.

Note que quanto maior o padrão e maior o tamanho do alfabeto melhor o desempenho do algoritmo. Esta é uma característica presente nos algoritmos que utilizam as idéias do algoritmo de Boyer e Moore [BM77].

Capítulo 5

Algoritmos genéricos

Neste capítulo apresentaremos os algoritmos de Baeza-Yates e Gonnet [BYG89], Wu e Manber [WM91].

O algoritmo de Baeza-Yates e Gonnet propõe uma solução para resolver o problema de casamento exato de padrões baseado em um método numérico. A partir desta solução, Baeza-Yates e Gonnet generalizaram a idéia para resolver o problema das k colisões e também para o problema de casamento de padrões com símbolos neutros, ou seja, padrões que possuem símbolos ϕ que casam com qualquer outro símbolo.

Wu e Manber, baseados na idéia do algoritmo de Baeza-Yates e Gonnet, desenvolveram um algoritmo bastante flexível que fornece soluções para os diversos problemas de casamento de padrões. Por exemplo: k diferenças, k colisões, padrões com símbolos neutros e busca de vários padrões no texto.

Descreveremos inicialmente o algoritmo de Baeza-Yates e Gonnet e, na seqüência, o algoritmo de Wu e Manber.

5.1 Algoritmo de Baeza-Yates e Gonnet

Descreveremos inicialmente o algoritmo para casamento exato de padrões.

Considere um alinhamento entre o texto e o padrão tal que $A(1 \ldots i)$ esteja alinhado, caracter a caracter, com $B(j - i + 1 \ldots j)$. Neste caso temos duas situações que podem ocorrer: $A(1 \ldots i) = B(j - i + 1 \ldots j)$ ou $A(1 \ldots i) \neq B(j - i + 1 \ldots j)$. Posto isto, representaremos estas duas situações através de uma tabela, denotada por S. Sejam $1 \leq j \leq n$ e $1 \leq i \leq m$. Se $A(1 \ldots i) = B(j - i + 1 \ldots j)$, $j \geq i$, então $S_{j,i} = 1$. Em

caso contrário, $S_{j,i} = 0$. Note que se $S_{j,m} = 1$, $j \geq m$, então temos uma ocorrência exata do padrão na posição $j - m + 1$ do texto.

Podemos obter $S_{j+1,i+1}$ a partir de $S_{j,i}$ da seguinte forma:

$$S_{j+1,i+1} = \begin{cases} 1 & \text{se } S_{j,i} = 1 \text{ e } A(i+1) = B(j+1) \\ 0 & \text{caso contrário,} \end{cases} \tag{5.1}$$

onde presumiremos que $S_{j,0} = 1$ para $1 \leq j \leq n$ e $S_{0,0} = 1$. Em outras palavras, se $A(1 \ldots i) = B(j - i + 1 \ldots j)$ e $A(i+1) = B(j+1)$ então $S_{j+1,i+1} = 1$. Em caso contrário, $S_{j+1,i+1} = 0$.

Observe que as informações de ocorrências de caracteres do texto no padrão também podem ser armazenadas em uma tabela. Para isto definiremos N tal que

$$N_{i,c} = \begin{cases} 1 & \text{se } c = A(i) \\ 0 & \text{caso contrário,} \end{cases}$$

onde $c \in \Sigma$ e $1 \leq i \leq m$. Veja que se $N_{i,c} = 1$ então c ocorre na posição i de A.

Assim, podemos reescrever (5.1) da seguinte forma:

$$S_{j+1,i+1} = \begin{cases} 1 & \text{se } S_{j,i} = 1 \text{ e } N_{i+1,B(j+1)} = 1 \\ 0 & \text{caso contrário.} \end{cases}$$

Resta agora estabelecer como obter a transição de $S_{j,i}$ para $S_{j+1,i+1}$ de maneira eficiente. Observe que cada valor $S_{j,i}$ pode ser representado por um bit, pois, $S_{j,i} = 1$ ou $S_{j,i} = 0$. Analogamente, $N_{i,c}$ também pode ser representado por um bit. Assim, podemos obter $S_{j+1,i+1}$ da seguinte forma:

$$S_{j+1,i+1} = S_{j,i} \textbf{ and } N_{i+1,B(j+1)}. \tag{5.2}$$

Como cada valor $S_{j,i}$ e cada valor $N_{i,c}$ pode ser representado por um bit então representaremos por S^j o vetor de bits formado pelos valores $S_{j,i}$ para $1 \leq i \leq m$. Da mesma forma, representaremos por N^c o vetor de bits formado pelos valores $N_{i,c}$ para $1 \leq i \leq m$.

Portanto, de (5.2) temos que

$$S^{j+1} = \vec{\mathrm{D}}\,(S^j) \textbf{ and } N^{B(j+1)},$$

onde $\vec{\mathrm{D}}$ indica a operação de deslocar S^j de 1 bit para a direita. Presumiremos que $S^0 = 00 \ldots 0$ (m zeros). Além disso, após o deslocamento a posição mais à esquerda do vetor será preenchida com o valor 1.

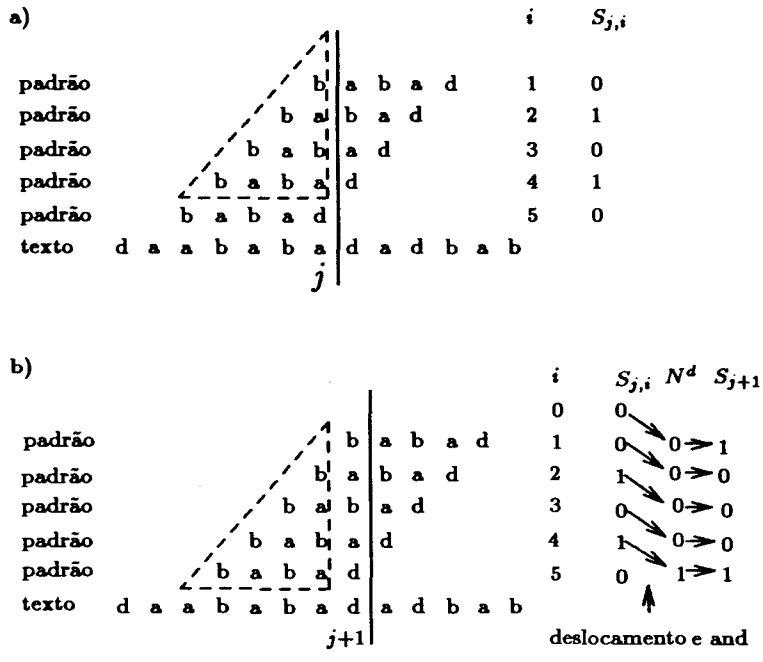

Figura 5.1: Transição de S^j para S^{j+1}

Para melhor visualizar esta operação considere o padrão A =babad e o texto B=daababadadbab. Na figura 5.1a são exibidos os valores de $S_{j,i}$ para j fixo e a figura 5.1b exibe a transição de S^j para S^{j+1}.

Assim, o algoritmo consiste em calcular os valores S^j, $1 \leq j \leq n$, ($S^0 = 0$) e verificar se o m-ésimo elemento de S^j possui o valor 1.

Na figura 5.2 exibimos um exemplo de como o algoritmo funciona. Nesta figura consideramos que A =babad e B =daababadad.

A vantagem de representar as informações da tabela S e N em bits é que as operações realizadas para obter os valores de S são simples e rápidas. Além disso, podemos armazenar S^j eficientemente através de um número na base 2 da seguinte forma:

$$S^j = \sum_{i=0}^{m-1} 2^i S_{j,i+1}.$$

Da mesma forma, N^c também pode ser representado por um valor

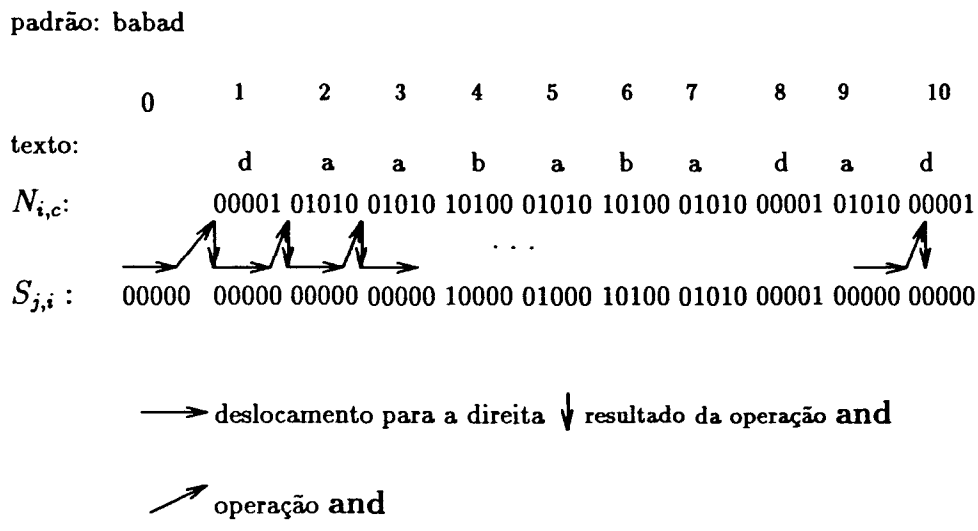

Figura 5.2: Diagrama de funcionamento do algoritmo de Baeza-Yates e Gonnet para o problema de casamento exato de padrões

numérico:

$$N^c = \sum_{i=0}^{m-1} 2^i N_{i+1,c}. \tag{5.3}$$

Agora analisaremos a complexidade do algoritmo.

Observe que durante o cálculo de S^j, $1 \leq j \leq n$, é realizado um número constante de operações (deslocamentos e operações **and**) em $\lceil m/w \rceil$ palavras, onde w é o tamanho da palavra utilizada pela máquina. Assim, a complexidade do algoritmo é $O(\lceil m/w \rceil n)$.

Para armazenar a tabela N serão necessários $\lceil m/w \rceil |\Sigma|$ palavras, pois, N deve conter todos os símbolos de Σ e necessitamos de $\lceil m/w \rceil$ palavras para armazenar cada N^c.

Podemos calcular a tabela N da figura 5.3.

Assim, para calcular os valores $N_{i,c}$ realizamos $O(m\,|\Sigma|)$ operações. Para obter os vetores N^c, $c \in \Sigma$, realizaremos $O(\lceil m/w \rceil\,|\Sigma|)$ operações, pois, para obter cada N^c são realizadas $O(\lceil m/w \rceil)$ operações (equação (5.3)). A complexidade do algoritmo para buscar as ocorrências do padrão no texto é $O(\lceil m/w \rceil n)$.

para $c \in \Sigma$ **faça**
    para $i \leftarrow 1$ **até** m **faça** $N_{i,c} \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça** $N_{i,A(i)} \leftarrow 1$

Figura 5.3: Algoritmo para calcular a tabela N.

5.1.1 Padrões contendo símbolos neutros

Um símbolo neutro é o símbolo ϕ cuja comparação com qualquer outro símbolo é sempre verdadeira, ou seja, a comparação $\phi \stackrel{?}{=} c$, $c \in \Sigma$, é sempre verdadeira.

A adaptação do algoritmo de Baeza-Yates e Gonnet para padrões com símbolos neutros é bastante simples. Basta fazer $N^{\phi} = \sum_{i=0}^{m-1} 2^i$, pois, $N_{i,\phi} = 1$ onde $1 \leq i \leq m$ mantendo a mesma estratégia para calcular os vetores S^j.

Pinter [Pin85] elaborou um algoritmo para resolver este problema utilizando as idéias do algoritmo para busca de vários padrões de Aho e Corasick [AC75]. Para isto, Pinter dividiu o padrão em partes tais que estas não possuam símbolos neutros. Por exemplo, para, A=ab$\phi\phi$cd$\phi\phi$ef Pinter dividiu o padrão nas seguintes partes: ab, cd, ef. A complexidade do algoritmo de Pinter é $O(n \log^2 m \log\log m \log|\Sigma|)$. No entanto, conforme observado por Baeza-Yates e Gonnet em [BYG89], este é um resultado teórico e o algoritmo não é prático.

Manber e Baeza-Yates [MBY91] elaboraram um algoritmo para padrões contendo apenas uma seqüência de símbolos neutros ($B = abc\phi\phi\phi de$, por exemplo). A idéia do algoritmo consiste em dividir o padrão em duas partes B_1 e B_2 tais que estas partes não contenham símbolos neutros. Assim, para $A = abc\phi\phi\phi de$ a divisão do padrão resulta nas partes $A_1 = abc$ e $A_2 = de$. Posteriormente, o algoritmo procura as ocorrências de A_1 e A_2 no texto, e verifica quais os pares de ocorrências de A_1 e A_2 que distam de 3 caracteres (número de símbolos neutros entre A_1 e A_2).

Uma observação importante é que adaptações dos algoritmos de Knuth, Morris e Pratt [KMP77] e de Boyer e Moore [BM77] não são viáveis para resolver o problema de padrões contendo símbolos neutros, pois, a transitividade da igualdade não é mais válida. Por exemplo, se $a = \phi$ e $\phi = b$ então não podemos dizer que $a = b$.

Devido à flexibilidade do algoritmo de Baeza-Yates e Gonnet podemos permitir que uma posição do padrão possa ser:

- c: um caracter do alfabeto Σ;
- ϕ: um símbolo neutro;
- $[a_1, \ldots, a_l]$: uma classe de caracteres, ou seja, qualquer caracter do texto comparado com a classe será igual se o caracter for igual a um dos símbolos $a_1, \ldots, a_l$. Por exemplo, para A=xyz$[m, n, o]$p temos que qualquer caracter c do alfabeto que for comparado com $A(4) = [m, n, o]$ será igual se $c \in [m, n, o]$.
- $\overline{c}$: símbolo complementar, ou seja, um caracter t é igual a $\overline{c}$ desde que $t \neq c$.

Para permitir padrões desses tipos basta que adaptemos o cálculo da tabela N. Por exemplo, para classe de caracteres $[a_1, \ldots, a_l]$, $N_{i,c}$ é tal que:

$$N_{i,c} = \begin{cases} 1 & \text{se } c \in [a_1, \ldots, a_l] \\ 0 & \text{caso contrário.} \end{cases}$$

Mostraremos agora mais uma variante do algoritmo que resolve o problema de k colisões.

5.1.2 Casamento aproximado com k colisões

Para resolver o problema das k colisões adotaremos a mesma estratégia do algoritmo para casamento exato que foi descrito anteriormente. Para isto, definiremos $C_{j,i}$, $1 \leq i \leq m$, $1 \leq j \leq n$ e $j \geq i$ como sendo o número de colisões existentes entre $A(1 \ldots i)$ e $B(j - i + 1 \ldots j)$. Assim, se $C_{j,m} \leq k$, $1 \leq j \leq n$, então temos uma ocorrência aproximada do padrão no texto na posição $j - m + 1$ do texto.

Definiremos a tabela $\overline{N}$ da seguinte forma:

$$\overline{N}_{i,c} = \begin{cases} 0 & \text{se } c = A(i) \\ 1 & \text{caso contrário} \end{cases}$$

Assim, a transição de $C_{j,i}$ para $C_{j+1,i+1}$ pode ser feita da seguinte forma:

$$C_{j+1,i+1} = C_{j,i} + \overline{N}_{i+1,B(j+1)}, \tag{5.4}$$

onde $C_{j,0} = 0$, $1 \leq j \leq n$ e $C_{0,0} = 0$. Em outras palavras, verificamos o número de colisões entre $A(1 \ldots i)$ e $B(j-i+1 \ldots j)$ e se $B(j+1) = A(i+1)$ então $C_{j+1,i+1} = C_{j,i}$. Em caso contrário $C_{j+1,i+1} = C_{j,i} + 1$.

Mostraremos agora como obter a transição de $C_{j,.i}$ para $C_{j+1,i+1}$ de maneira eficiente. Note que cada valor $C_{j,i}$ pode ser representado por $b := \lceil \log_2(k+1) \rceil$ bits, pois, $C_{j,i}$ pode ter no máximo $k+1$ valores. O valor $\overline{N}_{i,c}$ também pode ser representado por b bits ($\overline{N}_{i,c} = \overbrace{00\ldots0}^{b \text{ bits}}$ ou $\overline{N}_{i,c} = \overbrace{00\ldots01}^{b \text{ bits}}$). Observe que a representação em bits de $C_{j,i}$ indicará o valor de $C_{j,i}$ na base 2^b, o mesmo acontece para $N_{i,c}$.

Da mesma forma que no algoritmo anterior representaremos por C^j o vetor de bits formado pelos valores $C_{j,i}$ para $1 \leq i \leq m$. Analogamente, $\overline{N}^c$ representará o vetor de bits formado pelos valores $\overline{N}_{i,c}$ para $1 \leq i \leq m$.

Assim, de (5.4) temos que

$$C^{j+1} = \vec{\mathrm{D}}_b\,(C^j) + \overline{N}^{B(j+1)}, \tag{5.5}$$

onde $\vec{\mathrm{D}}_b$ indica a operação de deslocamento de b bits para a direita. Presumiremos que $C^0 = 00\ldots0$ (m zeros). Além disso, suporemos que posição mais à esquerda é preenchida com o valor 0.

Note que como realizamos operações de adição podemos ter um "vai-um" que pode alterar o valor do cálculo de C^j. Mais especificamente, ao aplicarmos (5.4) para calcular $C_{j,i}$ pode ocorrer um "vai-um" que pode afetar o valor de $C_{j,i+1}$, ou seja, $C_{j,i+1}$ pode assumir um valor que não representa o verdadeiro resultado do número de colisões entre $A(1 \ldots i+1)$ e $B(j-i \ldots j)$. Assim, para evitar este problema, para cada $C_{j,i}$ (grupo de b bits de C^j) utilizaremos um bit extra para armazenar o "vai-um". Após o cálculo de C^j os bits de "vai-um" são armazenados em um vetor de *overflow* V^j para "lembrar" que houve o "vai-um", posteriormente o bit de *overflow* é zerado antes começar o calculo de C^{j+1}. Assim, o algoritmo consiste em calcular os vetores C^j e verificar se o m-ésimo elemento de C^j mais o valor correspondente do vetor de *overflow* é no máximo k.

Na figura 5.4 exibimos um exemplo de como do algoritmo funciona. Nesta figura consideramos que A=babad, B=daababadad e $k = 3$.

Da mesma forma que no algoritmo de casamento exato, durante o cálculo de C^j, $1 \leq j \leq n$, é realizado um número constante de operações em

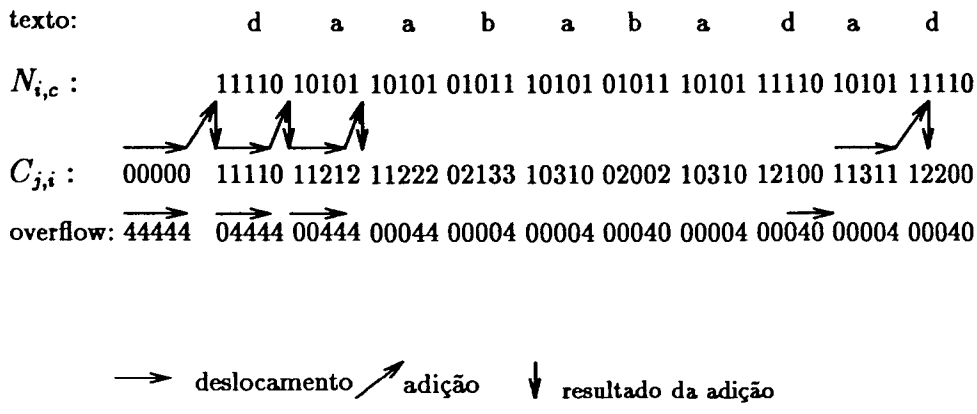

Figura 5.4: Diagrama de funcionamento do algoritmo de Baeza-Yates e Gonnet para o problema de casamento aproximado de padrões.

$\lceil mb/w \rceil$ palavras, onde w é o tamanho da palavra da máquina. Assim, a complexidade do algoritmo é $O(\lceil mb/w \rceil n)$.

As figuras 5.5, 5.6 e 5.7 exibem resultados empíricos da comparação (realizados em [BY89]) de três algoritmos para o problema das k colisões. Os algoritmos comparados são: o algoritmo ingênuo (comparação caracter a caracter e deslocamento de uma posição), o algoritmo de Landau e Vishkin e o algoritmo de Baeza-Yates e Gonnet. Os experimentos foram realizados com um texto em inglês com 50000 caracteres e com 1000 padrões de tamanhos variando de 2 a 8 caracteres. O desempenho do algoritmo de Baeza-Yates e Gonnet é superior aos demais.

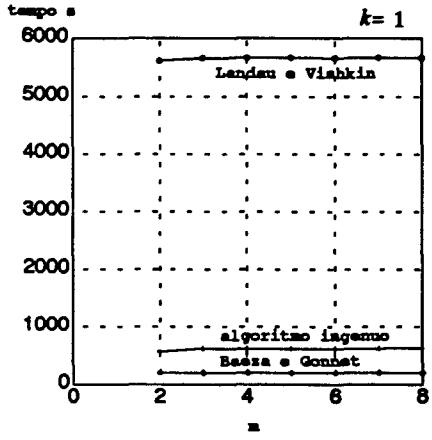

Figura 5.5: Comparação dos algoritmos de Landau e Vishkin, Baeza-Yates e Gonnet e o método Ingênuo com $k = 1$.

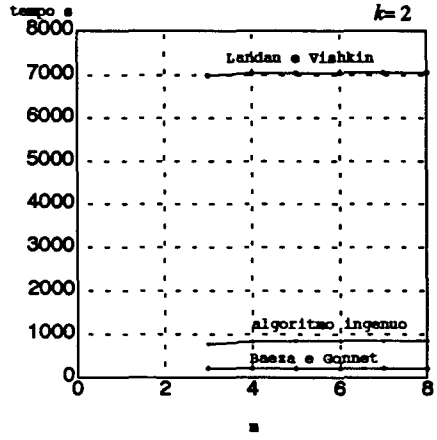

Figura 5.6: Comparação dos algoritmos de Landau e Vishkin, Baeza-Yates e Gonnet e o método Ingênuo para $k = 2$.

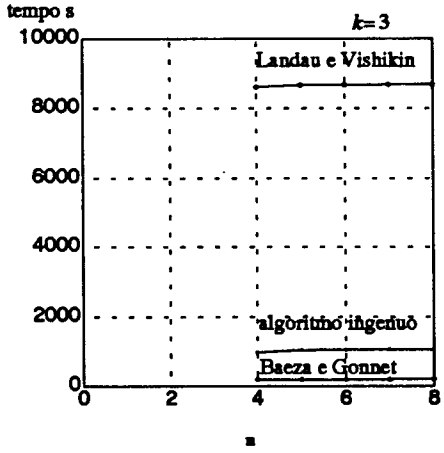

Figura 5.7: Comparação dos algoritmos de Landau e Vishkin, Baeza-Yates e Gonnet e o método Ingênuo para $k = 3$.

5.2 Algoritmo de Wu e Manber

Este algoritmo é uma extensão do algoritmo de Baeza-Yates e Gonnet. Wu e Manber adaptaram o algoritmo descrito na seção anterior para resolver o problema das k diferenças.

Começaremos a descrição do algoritmo com a análise de um caso simples: apenas uma substituição é permitida em qualquer posição do padrão. Em outras palavras, analisaremos o problema das k colisões onde $k = 1$.

Note que neste caso temos duas situações que podem acontecer, ou o padrão ocorre no texto com mais do que uma substituição, ou ocorre no texto com no máximo uma substituição. Desta forma, podemos representar estas informações com o uso de uma tabela como foi realizado no algoritmo de Baeza-Yates e Gonnet. Assim, definiremos uma tabela, denotada por ${}^1\mathrm{E}$ que indicará as ocorrências do padrão no texto com no máximo uma substituição. Mais precisamente, ${}^1\mathrm{E}_{j,i} = 1$, $1 \leq i \leq m$, $1 \leq j \leq n$ e $j \geq i$ se, e somente se, $A(1 \ldots i)$ é igual a $B(j-i+1 \ldots j)$ realizando-se máximo uma substituição em $A(1 \ldots i)$.

Mostraremos agora como obter ${}^1\mathrm{E}_{j+1,i+1}$ a partir de ${}^1\mathrm{E}_{j,i}$. Para isto utilizaremos as tabelas S e N definidas na seção anterior.

Existem dois casos em que $A(1 \ldots i+1)$ é igual a $B(j-i+1 \ldots j+1)$ com no máximo uma substituição:

- existe um casamento exato entre $A(1 \ldots i)$ e $B(j-i+1 \ldots j)$ e, além disso, $A(i+1) = B(j+1)$ ou $A(i+1) \neq B(j+1)$; este caso corresponde a uma substituição de $A(i+1)$ por $B(j+1)$ independentemente da igualdade entre $B(j+1)$ e $A(i+1)$.

- existe um casamento aproximado entre $A(1 \ldots i)$ e $B(j-i+1 \ldots j)$ com uma substituição e, além disso, $B(j+1) = A(i+1)$.

Desta forma, podemos dizer que ${}^1\mathrm{E}_{j+1,i+1}$ é tal que:

$$ {}^1\mathrm{E}_{j+1,i+1} = \begin{cases} 1 & \text{se } S_{j,i} = 1 \text{ ou } ({}^1\mathrm{E}_{j,i} = 1 \text{ e } B(j+1) = A(i+1)) \\ 0 & \text{caso contrário,} \end{cases} \tag{5.6} $$

onde ${}^1\mathrm{E}_{0,0} = 1$ e ${}^1\mathrm{E}_{j,0} = 1$ para $1 \leq j \leq n$.

Utilizando a mesma estratégia do algoritmo e Baeza-Yates e Gonnet representaremos cada valor ${}^1\mathrm{E}_{j,i}$ por um bit. Assim, podemos reescrever (5.6) da seguinte forma:

$$^1E_{j+1,i+1} = S_{j,i} \textbf{ or } (^1E_{j,i} \textbf{ and } N_{i+1,B(j+1)}). \quad (5.7)$$

Representaremos por $^1E^j$ o vetor de bits formado pelos valores $^1E_{j,i}$. Portanto, de (5.7) temos que

$$^1E^j = S^j \textbf{ or } (\vec{D}\ (^1E^j) \textbf{ and } N^{B(j+1)}),$$

onde presumiremos que $^1E^0 = 00\ldots0$ e, além disso, ao deslocarmos $^1E^j$ o bit mais à esquerda é preenchido com o valor 1.

Analisemos agora o problema de permitir no máximo uma inserção no padrão. Considere a nova definição de ^1E: $^1E_{j,i} = 1$ se e somente se $A(1\ldots i)$ é igual a $B(j-i\ldots j)$ com no máximo uma inserção em $A(1\ldots i)$.

É importante ressaltar que a análise deste caso é semelhante ao caso anterior, a diferença está na manipulação dos índices de A e B.

Mostraremos agora como obter a transição para ^1E. Mais especificamente, como obter $^1E_{j+1,i+1}$ a partir de $^1E_{j,i}$. Existem dois casos em que $A(1\ldots i+1)$ é igual a $B(j-i\ldots j+1)$ com no máximo uma inserção:

- existe um casamento exato entre $A(1\ldots i+1)$ e $B(j-i\ldots j)$; este caso corresponde à inserção de $B(j+1)$ no final de $A(1\ldots i+1)$;
- existe um casamento entre $A(1\ldots i)$ e $B(j-i\ldots j)$ com uma inserção e, além disso, $A(i+1) = B(j+1)$.

Desta forma, podemos dizer que $^1E_{j+1,i+1}$ é tal que :

$$^1E_{j+1,i+1} = \begin{cases} 1 & \text{se } S_{j,i+1} = 1 \text{ ou } (^1E_{j,i} = 1 \text{ e } A(i+1) = B(j+1)) \\ 0 & \text{caso contrário,} \end{cases} \quad (5.8)$$

onde $^1E_{0,0} = 1$ e $^1E_{j,0} = 1$ para $1 \leq j \leq n$.

Se representarmos cada valor $^1E_{j+1,i+1}$ por um bit então podemos reescrever (5.8) da seguinte forma:

$$^1E_{j+1,i+1} = S_{j,i+1} \textbf{ or } (^1E_{j,i} \textbf{ and } N_{i+1,B(j+1)}).$$

Analogamente ao caso anterior temos que

$$^1E^{j+1} = S^j \textbf{ or } (\vec{D}\ (^1E^j) \textbf{ and } N^{B(j+1)}),$$

onde presumiremos que ${}^1\mathrm{E}^0 = 00\ldots0$ e, além disso, ao deslocarmos ${}^1\mathrm{E}^j$ o bit mais à esquerda é preenchido com o valor 1.

Agora analisemos o problema de permitir no máximo uma remoção. Neste caso, ${}^1\mathrm{E}_{j,i} = 1$ se, e somente se, $A(1\ldots i)$ é igual a $B(j-i+2\ldots j)$ realizando-se uma remoção em $A(1\ldots i)$. Mostraremos como obter ${}^1\mathrm{E}_{j+1,i+1}$ a partir de ${}^1\mathrm{E}_{j,i}$. Existem dois casos em que $A(1\ldots i+1)$ é igual a $B(j-i+2\ldots j+1)$ com no máximo uma remoção:

- existe um casamento exato entre $A(1\ldots i)$ e $B(j-i+2\ldots j+1)$; este caso corresponde a remoção de $A(i+1)$;
- existe um casamento aproximado entre $A(1\ldots i)$ e $B(j-i+2\ldots j)$ com uma remoção em A e, além disso, $A(i+1) = B(j+1)$.

Desta forma, podemos dizer que ${}^1\mathrm{E}_{j+1,i+1}$ é tal que:

$$ {}^1\mathrm{E}_{j+1,i+1} = \begin{cases} 1 & \text{se } S_{j+1,i} = 1 \text{ ou } ({}^1\mathrm{E}_{j,i} = 1 \text{ e } A(i+1) = B(j+1)) \\ 0 & \text{caso contrário,} \end{cases} \tag{5.9} $$

onde ${}^1\mathrm{E}_{0,0} = 1$ e ${}^1\mathrm{E}_{j,0} = 1$ para $1 \leq j \leq n$.

Equivalentemente, temos que

$$ {}^1\mathrm{E}^{j+1} = S^{j+1} \textbf{ or } (\vec{\mathrm{D}}\,({}^1\mathrm{E}^j) \textbf{ and } N^{B(j+1)}), $$

onde presumiremos que ${}^1\mathrm{E}^0 = 00\ldots0$ e, além disso, ao deslocarmos ${}^1\mathrm{E}^j$ o bit mais à esquerda é preenchido com o valor 1.

Resta agora generalizar o método para permitir no máximo k diferenças onde uma diferença ou é uma inserção, ou uma substituição ou uma remoção. Para resolver este problema manteremos k tabelas: ${}^1\mathrm{E}, {}^2\mathrm{E}, \ldots, {}^k\mathrm{E}$, onde as tabelas ${}^d\mathrm{E}$, $1 \leq d \leq k$, indicam os casamentos com no máximo d diferenças. Mais especificamente, se ${}^d\mathrm{E}_{j,i} = 1$ então isto indica que um alinhamento de $A(1\ldots i)$ terminando na posição j do texto resulta em um casamento aproximado com no máximo d diferenças.

Mostraremos agora como obter a transição para ${}^d\mathrm{E}$. Existem quatro possibilidades de obter um casamento aproximado, com no máximo d diferenças, em um alinhamento de $A(1\ldots i)$ terminando na posição j do texto:

- existe um casamento com no máximo d diferenças no alinhamento de $A(1\ldots i-1)$ terminando na posição $j-1$ do texto e, além disso, $A(i) = B(j)$, ou seja, se $({}^d\mathrm{E}_{j-1,i-i} = 1$ e $A(i) = B(j))$;

- existe um casamento com no máximo $d-1$ diferenças no alinhamento de $A(1 \ldots i-1)$ terminando na posição $j-1$ do texto , ou seja, ${}^{d-1}\mathrm{E}_{j-1,i-1} = 1$, este caso corresponde à substituição de $A(i)$ por $B(j)$ independente de $A(i)$ e $B(j)$ serem iguais ou não;

- existe um casamento com no máximo $d-1$ diferenças no alinhamento de $B(1 \ldots i-1)$ terminando na posição j do texto, ou seja, ${}^{d-1}\mathrm{E}_{j,i-1} = 1$,este caso corresponde à remoção de $A(i)$;

- existe um casamento com no máximo $d-1$ diferenças no alinhamento de $A(1 \ldots i)$ terminando na posição $j-1$ do texto, ou seja, ${}^{d-1}\mathrm{E}_{j-1,i} = 1$, este caso corresponde à inserção de $B(j)$ no final de $A(1 \ldots i)$.

Desta forma podemos dizer que ${}^{d}\mathrm{E}_{j,i}$ é tal que

$$ {}^{d}\mathrm{E}_{j,i} \begin{cases} 1 & \text{se } ({}^{d}\mathrm{E}_{j-1,i-i} = 1 \text{ e } A(i) = B(j)) \text{ ou } ({}^{d-1}\mathrm{E}_{j-1,i-1} = 1) \text{ ou} \\ & \quad ({}^{d-1}\mathrm{E}_{j,i-1} = 1) \text{ ou } ({}^{d-1}\mathrm{E}_{j-1,i} = 1) \\ 0 & \text{caso contrário,} \end{cases} $$

onde ${}^{0}\mathrm{E}_{j,i} = S_{j,i}$ e ${}^{d}\mathrm{E}_{0,i} = 1$ para $1 \leq i \leq d$.

Considerando que ${}^{d}\mathrm{E}_{j,i}$ pode ser representado por um bit então

$$ {}^{d}\mathrm{E}^{j} = (\vec{\mathrm{D}}\,({}^{d}\mathrm{E}^{j-1}) \textbf{ and} N^{B(j)}) \textbf{ or } \vec{\mathrm{D}}\,({}^{d-1}\mathrm{E}^{j-1}) \textbf{ or } \vec{\mathrm{D}}\,({}^{d-1}\mathrm{E}^{j}) \textbf{ or } {}^{d-1}\mathrm{E}^{j-1} $$

$$ = (\vec{\mathrm{D}}\,({}^{d}\mathrm{E}^{j-1}) \textbf{ and } N^{B(j)} \textbf{ or } \vec{\mathrm{D}}\,({}^{d-1}\mathrm{E}^{j-1} \textbf{ or } {}^{d-1}\mathrm{E}^{j}) \textbf{ or } {}^{d-1}\mathrm{E}^{j-1} \quad (5.10) $$

Observe que no total temos quatro deslocamentos, uma operação **and** e três operações **or** para cada ${}^{d}\mathrm{E}$. Portanto, temos $O((k+1)n)$ operações para calcular ${}^{d}\mathrm{E}$, $1 \leq d \leq k$.

A complexidade do pré-processamento é a mesma do algoritmo de Baeza-Yates e Gonnet [BYG89], ou seja, $O(m\,|\Sigma|)$. Já a complexidade da análise do texto é $nk\,\lceil m/w \rceil$, pois são realizadas $k\,\lceil m/w \rceil$ por caracter do texto.

Uma característica importante deste algoritmo é a flexibilidade com que ele pode ser adaptado para outros tipos de padrão. Na seção seguinte descreveremos algumas das extensões possíveis para este algoritmo.

5.2.1 Extensões

O algoritmo de Wu e Manber também permite o uso de padrões contendo classes de caracteres. Da mesma forma que no algoritmo de Baeza-Yates e Gonnet, a única modificação necessária é pré-processar o padrão de maneira adequada. Por exemplo, suponha que o padrão desejado seja $xyz[0,1,2,\ldots,9]wz$, ou seja, desejamos um número na posição 4 do padrão. Então basta fazer $N_{4,c} = 1$ para $c \in [0,1,2,\ldots,9]$.

Analogamente ao algoritmo de Baeza-Yates e Gonnet , o algoritmo de Wu e Manber também permite a utilização de padrões com símbolos neutros. Por exemplo, para o padrão $xyz\phi w$ basta fazer $N_{4,c} = 1$ para qualquer $c \in \Sigma$.

Além desta característica o algoritmo permite a ocorrência consecutiva de um número indeterminado de símbolos neutros entre dois símbolos quaisquer do padrão. Denotaremos a seqüência de símbolos neutros de tamanho indeterminado por #. Assim, se $A = algo\#m$ então entre os símbolos o e m existe um número indeterminado de símbolos neutros. Os índices dos símbolos não neutros de um padrão que contém seqüências # serão dados pela ordem em que ele aparecem no padrão. Assim temos que $A(1) = a$, $A(2) = l$, $A(3) = g$, $A(4) = o$ e $A(5) = m$.

Agora, considere o texto $B = dcdabalgoritmdl$ e o padrão A definido acima. Note que A ocorre na posição 6 de B:

$$\begin{array}{cccccccccccccc} d & c & d & a & b & a & l & g & o & r & i & t & m & d & l \\ & & & & & a & l & g & o & \phi & \phi & \phi & m & & \end{array}$$

Mostraremos agora como analisar os alinhamentos no texto para este tipo de padrão. Suponha que após $A(i)$, $1 \leq i \leq m-1$, exista uma seqüência #. Além disso, suponha que um alinhamento de $A(1\ldots i)$ que termina na posição j do texto resulta em um casamento com no máximo d diferenças. Assim, ao analisarmos qualquer posição r de B, $r > j$, podemos começar a análise com $B(r)$ e $A(i+1)$. Note que ignoramos os caracteres entre $B(j)$ e $B(r-1)$, pois, após $A(i)$ existe um número indeterminado de símbolos neutros. Em outras palavras, um alinhamento de $A(1\ldots i)$ terminando na posição j com no máximo d diferenças é válido da posição j em diante.

Uma outra flexibilidade do algoritmo é a de permitir erros apenas em algumas partes do padrão. Por exemplo, suponha que no padrão abc123 permitimos erros apenas na parte que contém números, ou seja, na parte 123. Para solucionar este problema, construiremos uma tabela I_i, $1 \leq i \leq m$ tal que $I_i = 0$ se $A(i)$ pertence à parte que não deve conter erros, em caso

contrário $I_i = 1$. Assim, para permitir erros em determinadas partes do padrão basta reescrever (5.10) da seguinte forma:

$$^{d}\mathrm{E}^{j} = ((\vec{\mathrm{D}}\,(^{d}\mathrm{E}^{j-1}) \textbf{ and } N^{B(j)}) \textbf{ or } (\vec{\mathrm{D}}\,(^{d-1}\mathrm{E}^{j-1} \textbf{ or } ^{d-1}\mathrm{E}^{j})) \textbf{ or } ^{d-1}\mathrm{E}^{j-1}) \textbf{ and } I$$

O algoritmo de Wu e Manber também permite que as operações de edição possuam custos não uniformes. Por exemplo, se desejamos que as inserções possuam custo 3. Então, uma inserção adiciona 3 unidades, ao invés de 1 unidade à tabela $^{d}\mathrm{E}$. Em outras palavras, uma inserção leva a uma casamento com d diferenças se anteriormente tínhamos um casamento com $d-3$ diferenças. Desta forma, basta que a parcela $^{d-1}\mathrm{E}^{j-1}$ que contribui para inserções seja substituída por $^{d-3}\mathrm{E}^{j-1}$.

Wu e Manber ainda sugerem para o problema das k colisões que quando $k << n$, então podemos deslocar a tabela $^{d}\mathrm{E}$ mais do que 1 bit. Ilustraremos esta idéia com um exemplo retirado de [WM91]. Seja A =abcdefghijkl ($m = 12$) e seja $k = 3$. Dividiremos o padrão em $k+1$ blocos, ou seja, abc, def, ghi e jkl e construiremos o padrão A' formado pela intercalação dos quatro blocos, ou seja, A' =adgjbehkcfil. Agora, calcularemos as tabelas $^{d}\mathrm{E}$ como na equação (5.10). No entanto, ao invés de deslocarmos a tabela de apenas um bit podemos deslocá-la de quatro bits, pois, A' é uma intercalação de blocos de A de tamanho 3. Além disso, apenas um bloco deverá casar corretamente com o texto.

Wu e Manber em [WM91] fornecem outras sugestões para tratamento de casos como: padrão de tamanho grande, alfabetos de tamanho grande e padrões com expressões regulares.

Apresentaremos a seguir alguns resultados empíricos realizados por Wu e Manber.

5.2.2 Testes empíricos

Para realizar os teste empíricos Wu e Manber utilizaram um texto aleatório com 1000000 caracteres e um padrão com 20 caracteres. Para efeito de comparação foram também testados os algoritmos de Ukkonen [Ukk85], Galil e Park [GP90].

Na figura 5.8 foram comparados os três algoritmos. Utilizou-se um alfabeto de tamanho 2 e variou-se o número de diferenças. Claramente, o

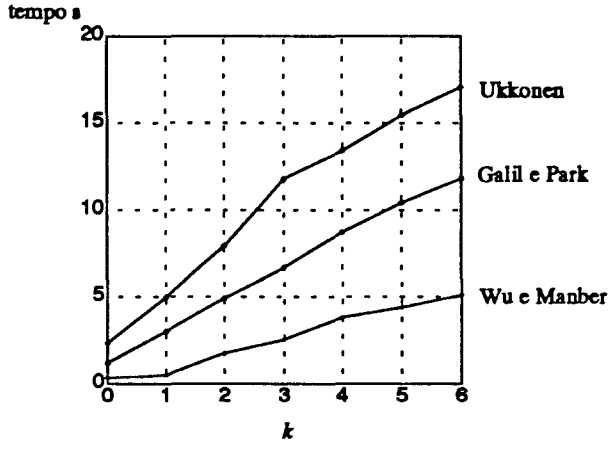

Figura 5.8: Comparação dos algoritmos de Galil-Park, Ukkonen e Wu-Manber para k variando no intervalo $[0, 6]$ e com $|\Sigma| = 2$.

algoritmo de Wu e Manber é superior aos demais. Na figura 5.9 o tamanho do alfabeto é de 30 símbolos e também variou-se o número de diferenças.

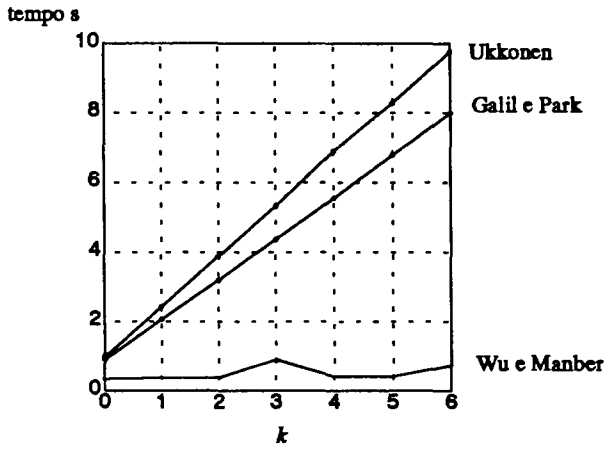

Figura 5.9: Comparação dos algoritmos de Galil-Park, Ukkonen e Wu-Manber para k variando no intervalo $[0,6]$ e com $|\Sigma| = 30$.

Capítulo 6

Conclusão

Conforme estabelecido no início da dissertação, os objetivos deste trabalho são de descrever e analisar de forma clara e precisa alguns algoritmos em três diferentes temas do problema de casamento aproximado.

Em nossa opinião alcançamos tal objetivo, descrevemos os algoritmos de maneira mais didática e organizada do que originalmente propostos.

Alcançar este objetivo é, a nosso modo de ver, uma grande contribuição, visto que muitos artigos estudados apresentavam suas idéias de maneira difusa, e em algumas vezes, alguns algoritmos apresentavam problemas de contorno e omissão de demonstrações.

No desenvolver do nosso trabalho acrescentamos algumas pequenas contribuições como demonstrações mais claras, por vezes, exibimos demonstrações que não constavam no artigo. Além disso, apresentamos uma nova versão do algoritmo de Ukkonen para o problema das k diferenças. Eliminamos muito do "ruído" existente nos artigos de Galil e Park [GP90] e Tarhio e Ukkonen [TU91]. Além disso, acreditamos que proporcionamos ao leitor uma visão ampla do problema do casamento aproximado de padrões.

Durante o desenvolver do nosso trabalho pudemos tirar algumas conclusões a respeito dos algoritmos estudados. Para a classe do problema das k diferenças acreditamos que o algoritmo de Galil e Park é o que apresenta o melhor desempenho em relação aos algoritmos de Ukkonen [Ukk85] e Tarhio e Ukkonen [TU91]. A evidência disto deve-se à complexidade do algoritmo de Tarhio e Ukkonen que é $O((n/k)m)$ e aos testes empíricos realizados em [WM91] cujos resultados são exibidos na figura 5.8 e 5.9. Além disso, acreditamos que os algoritmos apresentados para o problema das k diferenças

Algoritmo	Análise do Texto	Pré-Processamento
Ukkonen [Ukk85]	$O(m \min\{n \log s, s/\Delta\})$	$O(\min\{n, s/\Delta\})$
Galil e Park [GP90]	$O(kn)$	$O(m^2)$
Tarhio e Ukkonen [TU91]	$O((n/k)m)$	$O(mc)$

Tabela 6.1: Tabela comparativa dos algoritmos para o problema das k diferenças.Os valores s e c indicam o custo de edição e o tamanho do alfabeto, respectivamente.

possuem o mesmo grau de dificuldade para implementação. Na tabela 6.1 exibimos um quadro comparativo das complexidades de análise do texto, pré-processamento do padrão no pior caso.

Para a classe do problema das k colisões o algoritmo de Landau e Vishkin [LV86] é o que apresenta, teoricamente, o melhor desempenho. No entanto, análises empíricas realizadas em [BYG89] mostram que o algoritmo de Landau e Vishkin possui um desempenho pior do que o algoritmo ingênuo (figuras 5.5 e 5.6). Por outro lado, análises realizadas em [BY89] mostram que o algoritmo de Baeza-Yates apresenta um desempenho um pouco melhor que o algoritmo ingênuo para k colisões. Assim, na prática o algoritmo de Baeza-Yates possui um desempenho melhor que o algoritmo de Landau e Vishkin. Acreditamos também que o algoritmo de Tarhio e Ukkonen também possua um desempenho semelhante ao algoritmo de Baeza-Yates, pois o algoritmo de Tarhio e Ukkonen também se baseia em heurísticas semelhantes à heurística de Boyer e Moore. Uma outra característica importante a ressaltar é a simplicidade dos algoritmos de Baeza-Yates e Tarhio e Ukkonen quando comparados com a idéia do algoritmo de Landau e Vishkin. Na tabela 6.2 apresentamos um quadro comparativo dos algoritmos para k colisões.

Por fim, para os algoritmos genéricos concluímos que o algoritmo de Baeza-Yates e Gonnet e o algoritmo Wu e Manber [WM91] inauguraram uma nova geração de algoritmos para o problema de casamento aproximado. Ambos os algoritmos são de fácil entendimento, fácil implementação e utilizam idéias simples. Além disso, a utilização de operações com bits são rápidas. Nesta mesma abordagem temos o algoritmo de Baeza-Yates e Perleberg [BYP92] que resolvem o problema das k colisões e k diferenças

Algoritmo	Análise do Texto	Pré-Processamento
Landau e Vishkin [LV86]	$O(kn)$	$O(km \log m)$
Baeza-Yates [GP90]	$O(mn)$	$O(m^2)$
Tarhio e Ukkonen [TU91]	$O(mn)$	$O(k(m+c))$

Tabela 6.2: Tabela comparativa dos algoritmos para o problema das k colisões.O valor c indica o tamanho do alfabeto.

Algoritmo	Análise do Texto	Pré-Processamento
Baeza-Yates e Gonnet [BYG89]	$O(n(mb/w))$	$O((m/w)\|\Sigma\|)$
Wu e Manber [WM91]	$O(nk\lceil m/w \rceil)$	$O((m/w)\|\Sigma\|)$

Tabela 6.3: Tabela comparativa dos algoritmos genéricos. O valor w indica o tamanho da palavra da máquina, e o valor b é o número de bits deslocados no problema das k colisões. Se o problema é para casamento exato ou padrões com símbolos neutros então $b = 1$.

com o uso de contadores associados a cada caracter do texto.

Podemos concluir que o algoritmo de Wu e Manber é melhor que o algoritmo de Baeza-Yates e Gonnet em aspectos gerais, pois ele é mais flexível para resolver os diversos problemas do casamento aproximado e, além disso, o algoritmo é simples e prático. Porém, é importante ressaltar que o algoritmo Baeza-Yates possui seus méritos, pois as idéias são simples e o algoritmo de Wu e Manber é uma generalização das idéias do algoritmo de Baeza-Yates e Gonnet.

Para encerrar este capítulo, exibimos na tabela 6.3 um quadro comparativo entre o algoritmo Baeza-Yates e Gonnet e o algoritmo de Wu e Manber.

Bibliografia

[AC75] A. Aho e M. Corasick. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18:333–340, 1975.

[AHU74] A. V. Aho, J. E. Hopcroft, e J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison Wesley, 1974.

[AL93] M. V. A Andrade e C. L. Lucchesi. Algoritmos para reconhecimento eficiente de padrões. Master's thesis, Universidade Estadual de Campinas, 1993.

[BM77] R. Boyer e S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[BY89] R. Baeza-Yates. String matching with mismatches. *Phd Thesis*, pp. 93–104, 1989.

[BYG89] R. Baeza-Yates e G. H. Gonnet. A new approach to text search. *Proc. 12th ACM SIGIR*, pp. 168–175, 1989.

[BYP92] R. Baeza-Yates e C. H. Perleberg. Fast and pratical approximate string matching. *Lecture Notes on Computer Science*, 644:185–192, April/May de 1992.

[Fra91] K. Frankel. The human genome project and informatics. *Communications of the ACM*, 34(11):33–39, 1991.

[GP90] Z. Galil e K. Park. An improved algorithm for approximate string matching. *SIAM Journal on Computing*, 19(6):989–999, 1990.

[Gra92] A. S. Grahan. String search. Technical report, School of Eletronic Engineering Science, University College of North Wales, 1992.

[KMP77] D Knuth, J. Morris, e V. Pratt. Fast pattern matching in strings. *Siam Journal on Computing*, 6(2):323–350, 1977.

[KS87] K. Kristhivasan e R. Sitalakshmi. Efficient two-dimensional pattern matching in the presence of errors. *Information Sciences*, 43:169–184, 1987.

[Lan91] E. Lander. Mapping and interpreting biological information. *Communications of the ACM*, 34(11):33–39, 1991.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics Control Theory*, 10(8):707–720, 1966.

[LK93] C. L. Lucchesi e T. Kowaltowski. Application of finite automata representing large vocabularies. *Software Pratice and Experience*, 23(1):15–30, January de 1993.

[LV86] G. Landau e U. Vishkin. Efficient string matching with k-mismatches. *Theoretical Computer Science*, 43:239–249, 1986.

[MBY91] U. Manber e R. Baeza-Yates. An algorithm for string maching with a sequence of don't cares. *Information Processing Letters*, 45(37):133–136, 1991.

[Miu80] R. Miura. Some mathematical questions in biology- DNA sequence analysis. *Lectures on Mathematical in Life Sciences*, 17:1–29, 1980.

[Pin85] Y. R. Pinter. Efficient string matching with don't care pattern. In A. Apostolico e Z. Galil, editores, *Combinatorial algorithms on words*, pp. 11–29, Springer,Berlin, 1985. NATO: Advanced Science Institute.

[TU91] J. Tarhio e E. Ukkonen. Boyer-moore approach to approximate string-matching. *Lecture Notes on Computer Science*, 447:348–359, 1991.

[Ukk85] E. Ukkonen. Algorithms for approximate string-matching. *Information and Control*, 64:100–118, 1985.

[WM91] S. Wu e U. Manber. Fast text searching with error. Technical Report TR-91-11, Departament of Computer Science, University of Arizona, Tucson, 1991.

[Ziv93] N. Ziviani. *Projetos de Algoritmos com Implementação em Pascal e C.* Livraria Pioneira, 1993.