Universidade Federal de Minas Gerais

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Construção de Arranjo de Sufixos para Texto Comprimido

Edleno S. de Moura

Nivio Ziviani

Relatório Técnico 038/96[1]

Novembro 1996

[1] Available from http://www.dcc.ufmg.br/~nivio

Construção de Arranjo de Sufixos para Texto Comprimido*

Edleno S. de Moura

Nivio Ziviani

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil

Resumo

O objetivo deste trabalho é descrever um esquema eficiente de compressão para ser utilizado em conjunto com sistemas para recuperação de informação em texto completo. Bases de dados de texto completo tendem a se tornarem muito grandes no decorrer do tempo e são essencialmente estáticas. O modelo de compressão utilizado é semi-estático e o código para compressão é o de Huffman para palavras. Como consequência o espaço utilizado pelo texto fica abaixo de 30% do espaço ocupado pelo texto original e a velocidade de leitura sequencial descomprimindo o texto é pelo menos o dobro da velocidade de leitura sequencial do texto original. A implementação usa um índice para textos chamado arranjo de sufixos (ou *Pat Array*). Resultados experimentais preliminares mostram que a compressão não apenas economiza espaço, mas melhora também o desempenho no processamento da consulta e na construção do arranjo de sufixos.

1 Introdução

A popularização dos recursos computacionais fez crescer a utilização de bases de dados textuais do tipo publicações científicas, jornais, automação de escritórios, *World Wide Web (WWW)*, dentre outras. Essas bases de dados podem se tornar enormes, pois sofrem adições periódicas de novos textos ao final do arquivo, e são essencialmente estáticas, condições que as tornam ideais para utilização de compressão. Para sistemas que manipulam grandes bases de dados textuais é necessário preprocessar o texto criar um índice para permitir o acesso eficiente ao texto. Neste trabalho nós descrevemos um esquema eficiente

*Este trabalho tem recebido apoio do CNPq (Auxilio Integrado 520916/94-8) e Projeto RITOS/CYTED.

de compressão para ser usado em conjunto com sistemas para recuperação de informação em texto completo. Um aspecto interessante desta implementação é que a compressão não somente economiza espaço, mas também leva a um desempenho muito melhor na fase de consulta.

O índice utilizado é o arranjo de sufixos [MM90], também conhecido como *Pat Array* [Gon87, GBS92], que é basicamente um conjunto de apontadores para posições de interesse no texto ordenado pelos sufixos. Um *sufixo* é definido por uma posição inicial e se estende para a direita tanto quanto necessário ou até o final do texto. A pesquisa usando arranjo de sufixos é realizada através de uma pesquisa binária, a um custo $O(\log n)$, onde n representa o tamanho da base de dados.

O índice invertido é o tipo de índice mais comum utilizado em sistemas de recuperação de informação. Ele armazena, para cada termo que aparece em uma coleção de documentos, uma lista de endereços no texto correspondente ao termo. A complexidade de tempo e espaço ocupado são equivalentes para arranjo de sufixos e índice invertido. A grande vantagem do uso do arranjo de sufixos sobre o índice invertido é o seu uso potencial para uma enorme variedade de tipos de consultas que são difíceis ou ineficientes para arquivos invertidos [GBS92]. Este é o caso quando se pesquisa uma longa cadeia de caracteres, certos tipos de consultas lógicas, consultas permitindo erros e consulta usando expressões regulares.

O método de compressão utilizado é de Huffman para palavras com modelo semi-estático [Mof89]. O método de Huffman para palavras apresenta taxas de compressão muito altas, deixando o texto comprimido com aproximadamente 25% a 30% do tamanho do texto original [BMNWZ93] Inovações recentes na área de compressão, tais como a modelagem adaptativa e codificação aritmética não se mostram adequadas, devido a necessidade de acesso randômico ao texto e custo computacional elevado [ZM95].

2 Compressão de Textos

A compressão de arquivos consiste de duas etapas distintas, a saber: *modelagem* (ou criação de um modelo[1]), na qual uma probabilidade é atribuída a cada próximo símbolo possível na corrente de símbolos sendo representados; e *codificação*, onde o próximo símbolo real é representado por seu código relativo a distribuição de probabilidades obtida pela atividade de modelagem.

Os métodos de compressão podem ser classificados de acordo com o modelo de dados utilizado podendo ser estáticos, semi-estáticos ou adaptativos. O método estático usa o mesmo modelo para todos os textos, escolhido no momento da instalação do método. Neste caso as taxas de compressão podem ser muito baixas quando o texto sendo codificado não corresponde ao modelo, devendo ser utilizado somente quando a velocidade e a simplicidade são os únicos requisitos exigidos. O método semi-estático resolve este problema através do uso de um modelo diferente para cada texto. Antes da compressão propriamente dita, o

[1] Um modelo é essencialmente uma coleção de distribuições de probabilidades, uma para cada contexto no qual um caractere pode ser codificado.

texto é analisado e um modelo é construído para ser utilizado tanto pelo codificador quanto pelo decodificador. O método adaptativo atualiza o modelo (a partir de um modelo inicial qualquer) a cada leitura de um caractere do texto, evitando assim a necessidade de uma análise prévia do texto [BWC89, ZM95].

Os métodos modernos de compressão são tipicamente adaptativos porque permitem que a compressão seja realizada em uma passada sobre o texto original. Além disso, não há a necessidade de transmitir em separado o modelo usado na codificação para o decodificador iniciar seu trabalho. Para textos longos o modelo adaptativo converge para os valores que seriam obtidos se uma análise estatística preliminar tivesse ocorrido e, embora a taxa de compressão seja ligeiramente menos eficiente durante os estágios iniciais da codificação, a diferença entre uma e duas passadas é desprezível.

Entretanto, os métodos adaptativos mudam dinamicamente a codificação em relação a um caractere durante o processo de compressão, fazendo com que a decodificação tenha que ser realizada sequencialmente do início até o final do arquivo, impossibilitando a descompressão isolada de um dado fragmento de texto. Esta característica dos métodos adaptativos inviabiliza a sua aplicação em sistemas de bases de dados indexadas, onde o acesso aleatório e a necessidade de descompressão de fragmentos são essenciais.

Quanto a etapa de codificação, para os métodos adaptativos a preferência tem sido maior para a codificação aritmética [WNC87] do que para a codificação de Huffman [Huf52], porque a habilidade da codificação aritmética de permitir o uso de *bits* fracionários permite a obtenção de taxas de compressão próximas do ótimo, mesmo quando a distribuição de probabilidades dos caracteres tenda fortemente para um caractere. Entretanto, para sistemas de recuperação de informação em bases de dados de texto completo, Zobel e Moffat [ZM95] mostram que a modelagem adaptativa usando codificação aritmética não é a técnica mais efetiva a ser utilizada.

2.1 Escolha do Modelo

Ao invés de tentar utilizar um modelo adaptativo nós decidimos utilizar um modelo semi-estático, onde é necessário realizar uma passada preliminar sobre todo o texto a ser comprimido antes de realizar a compressão em uma segunda passada. De acordo com argumentos colocados em [ZM95], existem duas razões para se utilizar o modelo semi-estático ao invés do modelo adaptativo. A primeira é que os modelos adaptativos são inerentemente sequenciais, uma vez que o código usado para cada caractere de entrada é uma função da configuração inicial da distribuição de probabilidades e também do caractere anterior, o que não permite a decodificação a partir de pontos intermediários do texto.

A segunda é que os modelos adaptativos assumem uma distribuição de probabilidades inicial qualquer (por exemplo, cada caractere igualmente provável), sem assumir nenhum conhecimento prévio sobre o texto a ser comprimido. Logo, durante os passos iniciais da compressão, em que o modelo está aprendendo sobre os parâmetros do texto, a taxa de compressão é relativamente baixa. Quando o texto é longo este efeito não é significativo pois o modelo aprende rapidamente. Entretanto, para textos pequenos, segundo [ZM95] de até 10 *megabytes*, o resultado da compressão pode ser pobre devido ao custo da aprendizagem.

Em bases de dados de documentos os dois efeitos negativos aparecem combinados na utilização de modelos adaptativos. Para ser possível recuperar documentos separadamente é mandatório que cada documento seja codificado individualmente no modelo adaptativo. Entretanto, cada documento individual geralmente é constituído de poucos *bytes*, quase nunca passando de poucos milhares de *bytes*, o que deteriora a taxa de compressão.

Por outro lado, os métodos semi-estáticos permitem a utilização de um modelo que seja ótimo em relação a base de dados como um todo, mesmo que não seja ótimo para cada documento individualmente. Além deste aspecto os modelos semi-estáticos são mais rápidos na fase de descompressão (ou decodificação), pelo fato de não haver necessidade de atualizar o modelo a cada caractere lido. Uma possível desvantagem do modelo semi-estático é a necessidade de duas passadas pelo texto e também a necessidade de armazenar os parâmetros do modelo como parte da base de dados. Mas no caso de grandes bases de dados estáticas estes aspectos não são significativos uma vez que o esforço de criação do índice para permitir pesquisar eficientemente é consideravelmente maior, mais o fato de que o espaço ocupado pelo índice é ordens de grandeza maior do que o espaço ocupado pelos parâmetros do modelo.

2.2 Escolha do Método de Codificação

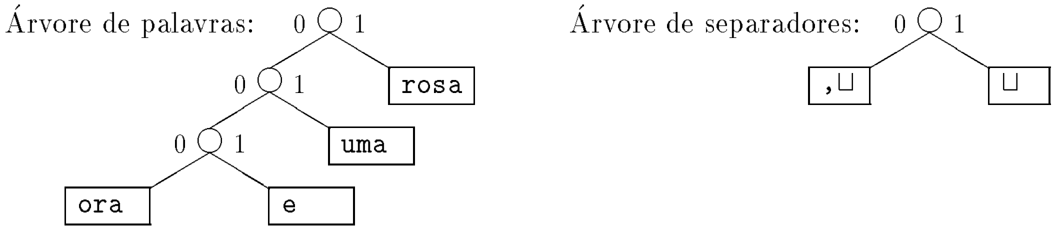
Um aspecto muito importante em textos muito grandes é a velocidade de descompressão. O método de codificação de Huffman [Huf52] é muito mais eficiente na decodificação do que o método aritmético. Uma das razões para que o código de Huffman tenha sido menos utilizado do que o código aritmético nos últimos anos é devido a dificuldade de obter resultados ótimos quando a distribuição de probabilidades é tendenciosa para alguns caracteres. Entretanto, para os tipos de texto que estamos considerando, Gallager [Gal78] mostrou que a ineficiência causada pelo arredondamento de probabilidades é de no máximo 9% quando comparado com o código aritmético empregado em modelos baseados em palavras, e o ganho obtido em velocidade de decodificação compensa esta penalidade.

2.3 Método de Huffman para Palavras

O método de Huffman semi-estático [Huf52] é um dos mais populares métodos de compressão de dados existentes. O método de Huffman consiste na construção de uma árvore de prefixos que busca associar aos caracteres mais frequentes uma codificação com o menor número de *bits* possível. Com o método pode-se determinar precisamente em que *bit* do arquivo comprimido uma determinada palavra do texto fonte irá começar. Isto faz com que o método e suas variações tornem-se adequados para a utilização de compressão em conjunto com índices para pesquisa em texto [ZM95, BMNWZ93].

Uma das variações é o método de Huffman para palavras [Mof89, BMNWZ93]. A principal alteração proposta neste método em relação ao método de Huffman original é a utilização do vocabulário do texto como alfabeto para a construção da árvore de Huffman, construindo uma outra árvore para os símbolos que separam as palavras do texto.

A figura 1 mostra um exemplo de compressão utilizando o método. A lista de símbolos para a árvore de palavras é composta pelas palavras "e", "ora", "rosa" e "uma", cujas frequências são 1, 1, 3 e 2, respectivamente. A lista de símbolos para a árvore de separadores é composta pelos separadores "␣" e ",␣", onde ␣ representa um espaço em branco, com as frequências 5 e 1, respectivamente. De acordo com o código de Huffman, os símbolos mais frequentes recebem um código menor. Logo, a palavra "rosa" é codificada como 1, a palavra "uma" é codificada como 01, a palavra "e" é codificada como 001 e a palavra "ora" é codificada como 000. Da mesma forma, o separador "␣," é codificado como 0 e o separador "␣" é codificado como 1. Para identificarmos qual a codificação de um determinado símbolo no texto comprimido, basta percorrermos a árvore da raiz até a folha onde se encontra o símbolo a ser codificado.



Texto Exemplo:	`ora rosa, uma rosa e uma rosa`
Texto Comprimido:	`000 1 1 0 01 1 1 1 001 1 01 1 1`

Figura 1: Compressão usando método de Huffman para palavras

A decodificação é realizada através da alternação na utilização das árvores de palavras e de separadores. Isto significa que devemos saber se o dado a ser descomprimido inicia com uma palavra ou se o mesmo inicia com uma sequência que não é uma palavra.

3 Arranjo de Sufixos e Compressão de Textos

Um dos principais enfoques para bases de dados de texto completo enxerga o texto como uma única cadeia de caracteres [Knu73, Gon87]. Cada posição de interesse no texto é chamada de sufixo. A base de dados pode ser vista como tendo um número infinito de caracteres nulos ao final do texto. Os sufixos podem ser lexicograficamente comparados caractere a caractere, e assim dois sufixos em posições diferentes nunca são iguais. A parte inferior da figura 2 ilustra um texto exemplo com sete sufixos a partir do início das palavras.

Sob o ponto de vista prático a estrutura de dados mais adequada para pesquisar no conjunto de sufixos é o *arranjo de sufixos* [MM90], também conhecido como *Pat array* [Gon87, GBS92]. O arranjo de sufixos é um conjunto de apontadores para posições de interesse no texto ordenado pelos sufixos. A parte superior da figura 2 apresenta o arranjo de sufixos para os sete pontos de indexação do texto exemplo. Para pesquisar a palavra `uma` nós realizamos uma pesquisa binária indireta sobre o arranjo de sufixos e obtemos [6,7]

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Arranjo de Sufixos:	20	1	26	15	5	22	11

Texto Exemplo: `ora rosa, uma rosa e uma rosa`

1 5 11 15 20 22 26

Figura 2: Arranjo de sufixos

como resposta. No caso, a posição 6 do arranjo aponta para o sufixo que inicia na posição 22 do texto, e a posição 7 do arranjo de sufixos aponta para o sufixo que inicia na posição 11 do texto. O tamanho do intervalo no arranjo de sufixos corresponde ao tamanho da resposta, que no caso é 2.

Para pesquisar todas as ocorrências de uma palavra o custo é no máximo $2m \log n$ comparações de caracteres e $4 \log n$ acessos a disco, onde m representa o número de caracteres da chave de consulta e n representa o número de sufixos de interesse na base de dados. O custo para construir o arranjo de sufixos é $O(n \log n)$, equivalente ao custo para ordenar os sufixos. O espaço ocupado pelo arranjo de sufixos é de apenas um apontador para cada sufixo, o que equivale a aproximadamente 60% do espaço ocupado pelo texto.

3.1 Arranjo de Sufixos Usando Huffman para Palavras

Nosso algoritmo de construção do arranjo de sufixos utiliza apenas o texto comprimido (ao invés de usar o texto original), realizando a descompressão durante o processo de ordenação para obter a ordem lexicográfica do texto original. A figura 3 mostra como fica o arranjo de sufixos para o texto exemplo codificado. Para pesquisar a palavra `uma` basta realizar a busca binária descomprimindo o texto a cada comparação.

	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Arranjo de Sufixos:	11	1	18	9	5	15	7

Texto Exemplo: `ora rosa, uma rosa e uma rosa`

Texto Comprimido: `0001 10 01 11 1001 101 11`

1 5 7 9 11 15 18

Figura 3: Compressão usando método Huffman para Palavras

A principal vantagem deste método é a sua enorme economia de espaço. A redução no tamanho do arquivo faz com que o tempo de *seek* durante o processo de busca seja menor no texto comprimido do que no texto original. Isto faz com que o tempo gasto na descompressão das informações seja compensado, permitindo assim que a busca no texto comprimido possa ter um desempenho melhor do que a busca no texto fonte. Este mesmo aspecto faz com que a construção do arranjo de sufixos usando o texto comprimido também seja mais eficiente.

Arranjos de sufixos permitem tratar com eficiência consultas que são difíceis ou ineficientes para índices invertidos, tais como consultas envolvendo longas cadeias de palavras, proximidade de palavras e também consultas permitindo erros[2] [AZ96]. Neste trabalho o algoritmo é baseado em uma extensão do arranjo de sufixos, onde é construída uma tabela contendo o vocabulário de palavras do texto e o intervalo no arranjo de sufixos onde cada palavra ocorre. O aspecto mais positivo do algoritmo é sua capacidade de pesquisa exata ou aproximada sem a necessidade de consultar o arranjo de sufixos ou o texto quando a chave de pesquisa contém apenas uma palavra.

Acontece que a estrutura de dados adicional proposta em [AZ96] é essencialmente a árvore de Huffman para palavras, o que torna a associação das duas propostas extremamente adequada. Como o vocabulário de palavras de um texto literário é muito menor do que o tamanho do texto original, a tabela de palavras pode permanecer na memória principal durante toda a fase de consulta. A tabela 1 mostra algumas estatísticas sobre cinco grandes arquivos de texto. Os textos foram extraídos de uma coleção de 2 *gigabytes* chamada TIPSTER [Har95], os quais são: *AP Newswire* (1989), *Short Abstracts (DOE Publications)*, *Wall Street Journal* (1987, 1988, 1989) e artigos da *Computer Select (Ziff-Davis Publishing)*.

Arquivos	Texto		Vocabulário		Vocab./Texto (%)	
	Tam. (*bytes*)	# Pal. (n)	Tam. (*bytes*)	# Palavras	Tam.	# Pal.
AP	266,527,343	41,849,862	1,889,417	234,587	0.71	0.56
DOE	192,723,670	28,997,047	1,860,024	214,707	0.97	0.74
FR	272,320,498	39,560,978	1,195,986	150,881	0.44	0.38
WSJ	279,534,695	43,392,799	1,550,382	204,587	0.55	0.47
ZIFF	254,022,399	39,459,716	1,639,662	216,480	0.65	0.55

Tabela 1: Arquivos de texto da coleção TIPSTER

[2] A consulta permitindo erros (ou consulta aproximada) ocorre quando a chave de pesquisa e/ou o texto não são exatos, devido a algum erro de digitação ou porque estamos procurando por chaves aproximadas. O problema da pesquisa aproximada em textos busca todas as ocorrências de uma chave de pesquisa p de acordo com uma distância de edição. Uma cadeia de caracteres p está a uma distância k de uma cadeia de caracteres c se p puder ser transformado para ficar igual a c com uma sequência de k inserções, retiradas ou substituições de caracteres únicos em pontos arbitrários de p. Este é um problema difícil de ser tratado eficientemente em grandes bases de dados textuais indexadas

4 Resultados Experimentais Preliminares

Uma medida de desempenho importante é a velocidade de leitura sequencial do texto a partir de um ponto qualquer. Esta operação ocorre com frequência, seja durante a construção do arranjo de sufixos, seja durante a pesquisa, ou seja para apresentar resultados aos usuários. A tabela 2 apresenta a velocidade de leitura (*megabytes* por segundo) em arquivos comprimidos com o método de Huffman para palavras e os arquivos originais sem compressão. Foram utilizados 4 arquivos de tamanhos 4, 8, 16 e 32 *megabytes* e a leitura foi realizada caractere a caractere, sendo que a quantidade de *bytes* lidos do arquivo considera o total de *bytes* obtidos após a descompressão. A velocidade de leitura do arquivo comprimido é aproximadamente o dobro, consequência da redução para menos de 30% do tamanho dos arquivos originais e da boa velocidade de descompressão do método.

Taxa de Leitura em Mb/s	
Sem Compressão	2,20
Huffman para Palavras	4,1

Tabela 2: Velocidade de acesso sequencial em texto comprimido

Uma outra medida importante é o tempo de acesso a posições aleatórias no texto. O acesso aleatório ocorre durante a pesquisa no arranjo de sufixos. A tabela 3 apresenta o tempo de acesso a 100 posições do texto escolhidas aleatoriamente. Para cada acesso realizado ao texto fonte, foi realizado um acesso à posição correspondente no texto comprimido sendo lidos 10 *bytes* a partir de cada posição acessada. O texto utilizado é parte contendo 50 *megabytes* do arquivo sobre o *Wall Street Journal* da coleção TIPSTER [Har95].

Tempo de Acesso Aleatório	
Arquivo	Tempo de Acesso
Huffman de Palavras	0,16
Sem Compressão	1,15

Tabela 3: Tempo para acesso a posições Aleatórias

Os dados confirmam que o tempo de acesso ao arquivo comprimido é bem menor do que o tempo de acesso ao arquivo original. Este resultado se deve a redução no tempo de *seek* aliada a alta velocidade de descompressão do método.

5 Considerações Finais

Nós descrevemos um esquema eficiente de compressão para arquivos de texto para ser utilizado em sistemas de recuperação de informação usando arranjo de sufixos como índice. O método de Huffman para palavras produz arquivos comprimidos com menos de 30% do tamanho do arquivo original, e a velocidade de descompressão de texto lido do disco tem sido acima de 4 milhões de caracteres por segundo.

Resultados experimentais preliminares mostram que a compressão, além de economia de espaço, proporciona maior desempenho nas fases de construção do índice e de pesquisa na base de dados. No momento estamos nos preparando para a realização de experimentos completos nos arquivos que constituem a coleção TIPSTER [Har95].

Em termos de trabalho futuro seria interessante estudar o impacto da compressão de textos no desenvolvimento de algoritmos paralelos para a construção do arranjo de sufixos [RKZ96, KRZ96].

Referências

[AZ96] M. D. Araújo e N. Ziviani. Large text searching allowing errors. *Fourth Annual European Symposium on Algorithms (ESA'96)*, Barcelona, 1996 (submetido).

[BMNWZ93] T. C. Bell, A. Moffat, C. G. Nevill-Manning, I. H. Witten e J. Zobel. Data compression in full-text retrieval systems. *Journal of the american society for information science*, 44: 508-531, 1993.

[BKR93] A. Bookstein, S. T. Klein e T. Raita. Is Huffman Coding dead?. *ACM-SIGIR*, 1993.

[BWC89] T. C. Bell, I. H. Witten. e J. G. Cleary. Modelling for text compression. *ACM Computing Surveys*, 21(4): 557-591, 1989.

[FHBL92] E. A Fox, D. K. Harman, R. Baeza-Yates e L. C. R. & Lee. Inverted files. In W. B. Frakes & R. Baeza-Yates (Eds.). Information Retrieval: Data Structures and Algorithms. Prentice-Hall, 1992.

[Gal78] R. G. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory*, IT-24(6): 668-674, 1978.

[GBS92] G. H. Gonnet, R. Baeza-Yates and T. Snider. PAT arrays: an alternative to inverted files. In W. B. Frakes and R. Baeza-Yates (Eds.), *Information Retrieval Data Structures and Algorithms*, 66-82, Prentice-Hall, Englewoods Cliff, N.J., 1992.

[GBY91] G. H. Gonnet e R. Baeza-Yates. Handbook of Algorithms and Data Structures. Addison-Wesley, 1991.

[Gon87] G. H. Gonnet. *PAT 3.1: An Efficient Text Searching System.* User's Manual. Center for the New Oxford English Dictionary. University of Waterloo, Waterloo, Canada, 1987.

[Har95] D. K. Harman. Overview of the Third Text REtrieval Conference. In *Proc. Third Text REtrieval Conference (TREC-3)*, pages 1-19, National Institute of

Standards and Technology Special Publication 500-207, Gaithersburg, Maryland, 1995.

[Huf52] D. A. Huffman. A Method for the construction of minimum-redundancy codes. *Proceedings of the I.R.E.*, 40(9): 1090-1101, 1952.

[Knu73] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley, Reading, Mass., 1973.

[KRZ96] J. P. W. Kitajima, B. A. N. Ribeiro e N. Ziviani. Network and Memory Analysis in Distributed Paralell Generation of PAT Arrays. *VIII Simpósio Brasileiro de Arquitetura de Computadores*, Recife, Agosto de 1996 (submetido).

[LS93] G. Linoff e C. Stanfill. Compression of indexes with full positional information in very large text databases. *ACM-SIGIR*, 1993.

[MB95] A. Moffat e T. C. Bell. In situ generation of compressed inverted files. *Journal of the american society for information science*, 46: 537-550, 1995.

[MM90] U. Manber, e G. Myers. Suffix arrays: A new method for on-line string searchs. *1st ACM-SIAM Symposium on Discrete Algorithms*, 319-327, São Francisco, 1990.

[Mof89] A. Moffat. Word-based text compression. *Software Pratice and Experience*, 19(2): 185-198, 1989.

[Man93] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. Thechnical Report 93-07. Department of Computer Science, University of Arizona, March 1993.

[RKZ96] B. A. N. Ribeiro, J. P. W. Kitajima e N. Ziviani. Distributed Paralell Generation of PAT Arrays. *Fourth Annual European Symposium on Algorithms (ESA'96)*, Barcelona, 1996 (submetido).

[ZM95] J. Zobel e A. Moffat. Adding compression to full-text retrieval system. *Software Pratice and Experience*, 25(8): 891-903, 1995.

[WNC87] I. H. Witten, R. Neal, e J. G. Cleary. Aritmetic codding for data compression. *Communications of the ACM*, 29(4): 320-330, 1986.