Compressão de Textos*

Última alteração: 23 de Março de 2014

*Transparências elaboradas por Fabiano Cupertino Botelho, Charles Ornelas Almeida, Israel Guerra e Nivio Ziviani

Conteúdo do Capítulo

Compressão - Motivação

- Explosão de informação textual disponível *on-line*:
  - Bibliotecas digitais
  - Sistemas de automação de escritórios
  - Bancos de dados de documentos
  - World-Wide Web
- Somente a Web tem hoje bilhões de páginas estáticas disponíveis
- Cada bilhão de páginas ocupando aproximadamente 10 *terabytes* de texto corrido
- Em setembro de 2003, a máquina de busca Google (www.google.com.br) dizia ter mais de 3,5 bilhões de páginas estáticas em seu banco de dados

Características necessárias para sistemas de recuperação de informação

- Métodos recentes de compressão têm permitido:
  1. Pesquisar diretamente o texto comprimido mais rapidamente do que o texto original
  2. Obter maior compressão em relação a métodos tradicionais, gerando maior economia de espaço
  3. Acessar diretamente qualquer parte do texto comprimido sem necessidade de descomprimir todo o texto desde o início (Moura, Navarro, Ziviani e Baeza-Yates, 2000; Ziviani, Moura, Navarro e Baeza-Yates, 2000)
- Compromisso espaço X tempo:
  - vencer-vencer

Porque Usar Compressão

- **Compressão de texto -** maneiras de representar o texto original em menos espaço:
  - Substituir os símbolos do texto por outros que possam ser representados usando um número menor de *bits* ou *bytes*
- **Ganho obtido:** o texto comprimido ocupa menos espaço de armazenamento $\Rightarrow$ menos tempo para ser lido do disco ou ser transmitido por um canal de comunicação e para ser pesquisado.
- **Preço a pagar:** custo para codificar e decodificar o texto
- **Avanço da tecnologia:** De acordo com Patterson e Hennessy (1995), em 20 anos, o tempo de acesso a discos magnéticos tem se mantido praticamente constante, enquanto a velocidade de processamento aumentou aproximadamente 2 mil vezes $\Rightarrow$ melhor investir mais poder de computação em compressão em troca de menos espaço em disco ou menor tempo de transmissão

Razão de Compressão

- Definida pela porcentagem que o arquivo comprimido representa em relação ao tamanho do arquivo não comprimido
- **Exemplo:** se o arquivo não comprimido possui 100 *bytes* e o arquivo comprimido resultante possui 30 *bytes*, então a razão de compressão é de 30%
- Utilizada para medir o ganho em espaço obtido por um método de compressão

Outros Importantes Aspectos a Considerar

Além da economia de espaço, deve-se considerar:

- Velocidade de compressão e de descompressão
- Possibilidade de realizar **casamento de cadeias** diretamente no texto comprimido
- Permitir acesso direto a qualquer parte do texto comprimido e iniciar a descompressão a partir da parte acessada:

  Um sistema de recuperação de informação para grandes coleções de documentos que estejam comprimidos necessitam acesso direto a qualquer ponto do texto comprimido

Abordagens para Compressão de Texto em Linguagem Natural

- Métodos Estatísticos
  - Uma probabilidade é obtida para cada símbolo e o código obtido é baseado na probabilidade
- Métodos de Dicionário
  - Substitui uma sequência de símbolos por um ponteiro que aponta para uma ocorrência anterior da sequência

Métodos de Dicionário

- Usa representação de ponteiros para codificar referências para entradas no dicionário
- Dicionário é uma lista de símbolos (geralmente chamado frases) que são esperados ocorrerem freqüentemente
- Ponteiros para entradas de dicionários necessitam menos espaço que a frase que eles substituem, obtendo assim a compressão

  Ex.: jan, feb,$\cdots$, dec são codificados como 1, 2, $\cdots$, 12
- Nenhuma probabilidade explícita associada as frases
- Métodos de dicionários mais conhecidos: família Ziv-Lempel

Família de Métodos de Compressão Ziv-Lempel

- Substitui uma sequência de símbolos por um apontador para uma ocorrência anterior daquela sequência
- A compressão é obtida porque os apontadores ocupam menos espaço do que a sequência de símbolos que eles substituem
- Os métodos Ziv-Lempel são populares pela sua velocidade, economia de memória e generalidade
- Já o método de Huffman baseado em palavras é muito bom quando a cadeia de caracteres constitui texto em linguagem natural

Desvantagens dos Métodos de Ziv-Lempel para Ambiente de Recuperação de Informação

- É necessário iniciar a decodificação desde o início do arquivo comprimido $\Rightarrow$ Acesso randômico muito caro
- É muito difícil pesquisar no arquivo comprimido sem descomprimir
- Uma possível vantagem do método Ziv-Lempel é o fato de não ser necesário armazenar a tabela de símbolos da maneira com que o método de Huffman precisa
- No entanto, isso tem pouca importância em um ambiente de recuperação de informação, já que se necessita o vocabulário do texto para criar o índice e permitir a pesquisa eficiente

Relacionamento entre Probabilidades e Códigos

- Claude Shannon (1948) em seu teorema:

  Em um esquema de codificação ótimo, um símbolo que é esperado ocorrer com probabilidade p deve ser atribuído um código de tamanho $\log_2 \frac{1}{p}$ bits

- O número de bits no qual o símbolo é melhor codificado representa o conteúdo informacional do símbolo

- A quantidade média de informação por símbolo sobre o alfabeto todo é chamado de *entropia* da distribuição de probabilidade, que é dada por:

$$E = \sum p_i \log_2 \frac{1}{p_i}$$

  Medida em bits por símbolo e representa um limite que nunca pode ser batido em um dado modelo

Métodos de Huffman Baseados em Palavras: Vantagens

- Permitem acesso randômico a palavras dentro do texto comprimido
- Considerar palavras como símbolos significa que a tabela de símbolos do codificador é exatamente o vocabulário do texto
- Isso permite uma integração natural entre o método de compressão e o arquivo invertido
- Permitem acessar diretamente qualquer parte do texto comprimido sem necessidade de descomprimir todo o texto desde o início

Compressão de Huffman Usando Palavras

- Técnica de compressão mais eficaz para textos em linguagem natural
- O método considera cada palavra diferente do texto como um símbolo
- Conta suas frequências e gera um código de Huffman para as palavras
- A seguir, comprime o texto substituindo cada palavra pelo seu código.
- Assim, a compressão é realizada em duas passadas sobre o texto:
  1. Obtenção da frequência de cada palavra diferente
  2. Realização da compressão

Forma Eficiente de Lidar com Palavras e Separadores

- Um texto em linguagem natural é constituído de palavras e de separadores
- Separadores são caracteres que aparecem entre palavras: espaço, vírgula, ponto, ponto e vírgula, interrogação, e assim por diante
- Uma forma eficiente de lidar com palavras e separadores é representar o espaço simples de forma implícita no texto comprimido
- Nesse modelo, se uma palavra é seguida de um espaço, então, somente a palavra é codificada
- Senão, a palavra e o separador são codificados separadamente.
- No momento da decodificação, supõe-se que um espaço simples segue cada palavra, a não ser que o próximo símbolo corresponda a um separador

Forma Eficiente de Lidar com Palavras e Separadores

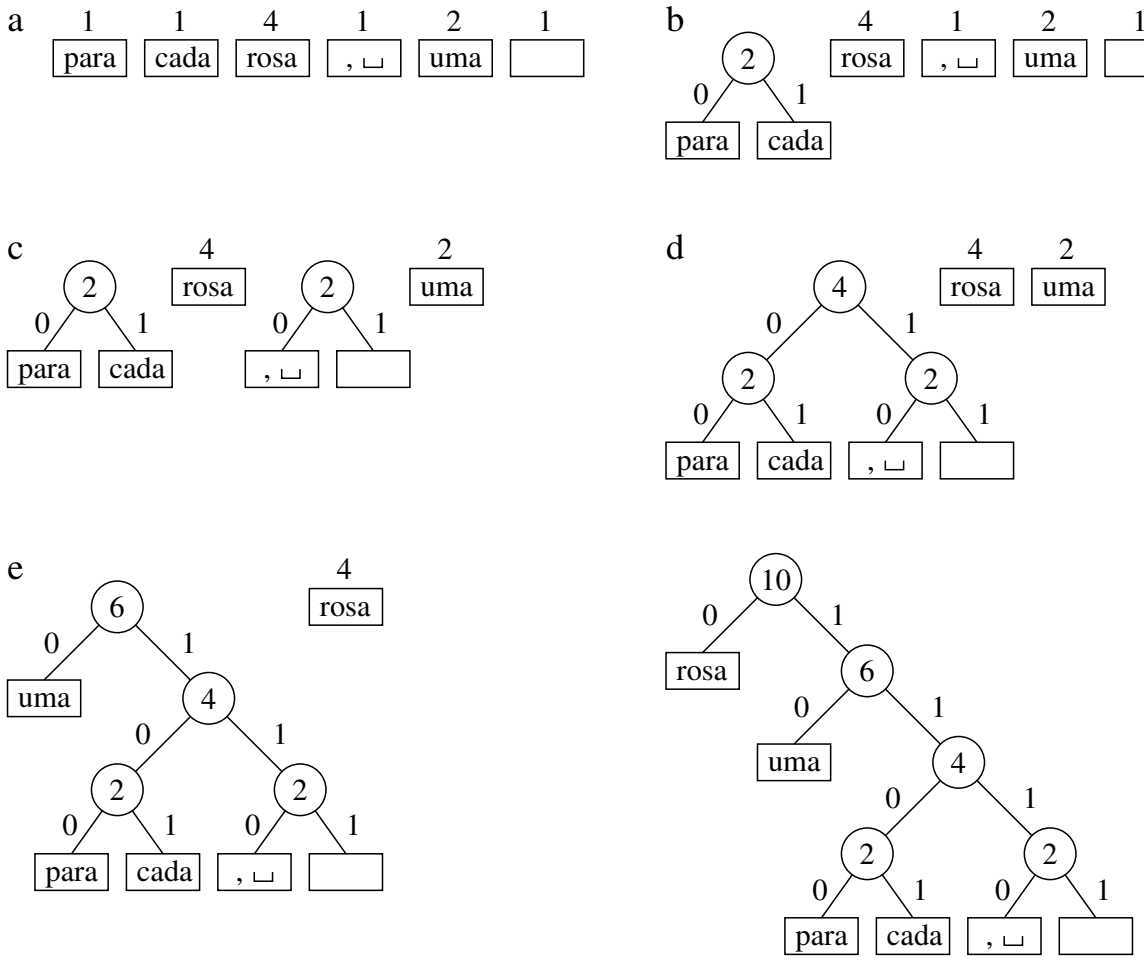
Arquivos	Palavras (p)	Espaços simples(s)	s/p
AP	38.977.670	30.020.803	0,77
DOE	28.505.125	22.001.366	0,77
FR	34.455.982	25.506.763	0,74
WSJ	42.710.250	33.909.610	0,79
ZIFF	39.675.248	27.225.864	0,69

Algoritmo da Árvore de Huffman

1. Liste todos os possíveis símbolos com suas probabilidades
2. Considere os dois símbolos com as menores probabilidades
3. Substitua estes dois por um único conjunto, cuja probabilidade é a soma das probabilidades individuais
4. Repita até que a lista contenha somente um membro

Compressão usando codificação de Huffman

Exemplo: "`para cada rosa rosa, uma rosa é uma rosa`"



Código: `0000 0001 1 1 0010 01 1 0011 01 1`

Árvore de Huffman

- O método de Huffman produz a árvore de codificação que minimiza o comprimento do arquivo comprimido
- Existem diversas árvores que produzem a mesma compressão.
- Por exemplo, trocar o filho à esquerda de um nó por um filho à direita leva a uma árvore de codificação alternativa com a mesma razão de compressão
- Entretanto, a escolha preferencial para a maioria das aplicações é a **árvore canônica**
- Uma árvore de Huffman é canônica quando a altura da subárvore à esquerda de qualquer nó nunca é menor que a altura da subárvore à direita

Árvore de Huffman

- A representação do código na forma de árvore facilita a visualização
- Sugere métodos de codificação e decodificação triviais:
  - **Codificação:** a árvore é percorrida emitindo *bits* ao longo de suas arestas
  - **Decodificação:** os *bits* de entrada são usados para selecionar as arestas
- Essa abordagem é ineficiente tanto em termos de espaço quanto em termos de tempo

Código Canônico

- Os comprimentos dos códigos obedecem ao algoritmo de Huffman
- Códigos de mesmo comprimento são inteiros consecutivos
- A partir dos comprimentos obtidos, o primeiro código é composto apenas por zeros e, para os demais, adiciona-se 1 ao código anterior
- Codificação canônica obtida:

i	Símbolo	Código Canônico
1	`rosa`	`1`
2	`uma`	`01`
3	`para`	`0000`
4	`cada`	`0001`
5	`,␣`	`0010`
6	`é`	`0011`

Árvore de Huffman Canônica

- Folha mais profunda na posição mais a esquerda (elemento com menor probabilidade) conterá somente zeros
- Os códigos seguintes estarão em ordem crescente dentro de cada nível
- A cada mudança de nível adiciona-se 0 ao código anterior e faz-se um deslocamento à esquerda

Código de Huffman Canônico

- Seqüência ordenada S de pares (x_i, y_i), $1 \leq i \leq \ell$, onde x_i representa o número de elementos no nível i e y_i representa o valor numérico do primeiro código no nível i
- Para o exemplo, a seqüência ordenada é:
  $S = \langle (1,1), (1,1), (0,\infty), (4,0) \rangle$
- O par $(4,0)$ em S corresponde ao quarto nível e indica que existem quatro nós neste nível e que o nodo mais a esquerda é atribuído código com valor 0 (código 0000 para o quarto nível)
- Conjuntos de códigos tendo o mesmo tamanho são representações binárias de inteiros consecutivos
- Interpretados como inteiros, os códigos de 4 bits são 0, 1, 2 e 3, código de 2 bits é 1 e código de 1 bit também é 1

Código de Huffman Canônico: Como Decodificar

- Código: `0000 0001 1 1 0010 01 1 0011 01 1`
- Se o primeiro caractere é 1, um código foi identificado e o símbolo correspondente pode ser decodificado
- Se este valor é 0, um segundo bit é adicionado e os dois bits são novamente interpretados como um inteiro e usado para indexar a tabela e identificar o símbolo correspondente
- Uma vez que nós lemos “00” nós sabemos que o código tem quatro bits e portanto nós podemos ler mais dois e usá-los como um índice dentro da tabela
- Este fato pode ser explorado para possibilitar codificação e decodificação eficiente com pequeno *overhead* de tempo e muito menos memória é requerida, o que é especialmente importante para grandes vocabulários

Algoritmo Baseado na Codificação Canônica com Comportamento Linear em Tempo e Espaço

- O algoritmo é atribuído a Moffat e Katajainen (1995)
- Calcula os comprimentos dos códigos em lugar dos códigos propriamente ditos
- A compressão atingida é a mesma, independentemente dos códigos utilizados
- Após o cálculo dos comprimentos, há uma forma elegante e eficiente para a codificação e a decodificação

O Algoritmo

- A entrada do algoritmo é um vetor A contendo as frequências das palavras em ordem não-crescente
- Frequências relativas à frase exemplo: "`para cada rosa rosa, uma rosa é uma rosa`"

4	2	1	1	1	1

- Durante execução são utilizados vetores logicamente distintos, mas capazes de coexistirem no mesmo vetor das frequências (*in place*)
- O algoritmo divide-se em três fases:
  1. Combinação dos nós
  2. Conversão do vetor no conjunto das profundidades dos nós internos
  3. Calculo das profundidades dos nós folhas

Primeira Fase - Combinação dos nós

- A primeira fase é baseada em duas observações:
  1. A frequência de um nó só precisa ser mantida até que ele seja processado
  2. Não é preciso manter apontadores para os pais dos nós folhas, pois eles podem ser inferidos

     Exemplo: nós internos nas profundidades $[0, 1, 2, 3, 3]$ teriam nós folhas nas profundidades $[1, 2, 4, 4, 4, 4]$

Exemplo de Codificação

- Para a palavra $i = 4$ (“`cada`”):
  1. Códigos de mesmo comprimento são inteiros consecutivos
  2. Verifica-se que é um código de comprimento 4
  3. Verifica-se também que é o segundo código com esse comprimento
  4. Assim, seu código é `0001`

i	Símbolo	Código Canônico
1	`rosa`	`1`
2	`uma`	`01`
3	`para`	`0000`
4	`cada`	`0001`
5	`,␣`	`0010`
6	`é`	`0011`

Codificação de Huffman Usando Bytes

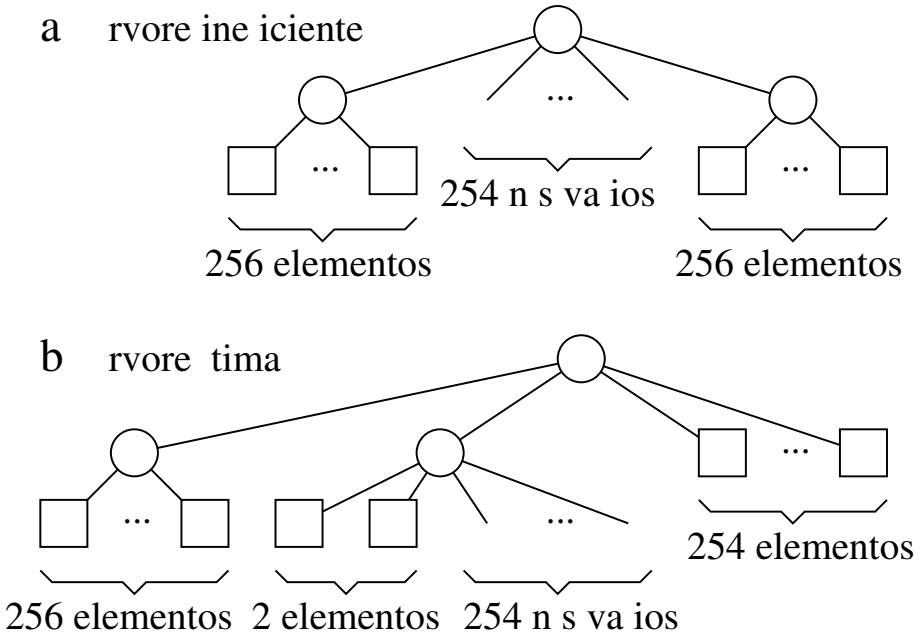
- O método original proposto por Huffman (1952) tem sido usado como um código binário
- Moura, Navarro, Ziviani e Baeza-Yates (2000) modificaram a atribuição de códigos de tal forma que uma sequência de *bytes* é associada a cada palavra do texto
- Consequentemente, o grau de cada nó passa de 2 para 256. Essa versão é chamada de *código de Huffman pleno*
- Outra possibilidade é utilizar apenas 7 dos 8 *bits* de cada *byte* para a codificação, e a árvore passa então a ter grau 128
- Nesse caso, o oitavo *bit* é usado para marcar o primeiro *byte* do código da palavra, sendo chamado de *código de Huffman com marcação*

Exemplo de Códigos Plenos e com Marcação

- O código de Huffman com marcação ajuda na pesquisa sobre o texto comprimido
- **Exemplo:**
  - Código pleno para "`uma`" com 3 *bytes*: "`47 81 8`"
  - Código com marcação para "`uma`" com 3 *bytes*: "`175 81 8`"
  - Note que o primeiro *byte* é $175 = 47 + 128$
- Assim, no código com marcação o oitavo *bit* é 1 quando o *byte* é o primeiro do código, senão ele é 0

Árvore de Huffman Orientada a *Bytes*

- A construção da árvore de Huffman orientada a *bytes* pode ocasionar o aparecimento de nós internos não totalmente preenchidos:



- Na Figura, o alfabeto possui 512 símbolos (nós folhas), todos com a mesma frequência de ocorrência
- O segundo nível tem 254 espaços vazios que poderiam ser ocupados com símbolos, mudando o comprimento de seus códigos de 2 para 1 *byte*

Movendo Nós Vazios para Níveis mais Profundos

- Um meio de assegurar que nós vazios sempre ocupem o nível mais baixo da árvore é combiná-los com os nós de menores frequências
- O objetivo é movê-los para o nível mais profundo da árvore.
- Para isso, devemos selecionar o número de símbolos que serão combinados com os nós vazios, dada pela equação:

  $$1 + ((n - \text{BaseNum}) \mod (\text{BaseNum} - 1))$$

- No caso da Figura da transparência anterior é igual a $1 + ((512 - 256) \mod 255) = 2$

Código para Fazer a Compressão

- O Código para fazer a compressão é dividido em três etapas:
  1. Na primeira, as palavras são extraídas do texto a ser comprimido e suas respectivas frequências são contabilizadas
  2. Na segunda, são gerados os vetores Base e Offset, os quais são gravados no arquivo comprimido seguidamente do vocabulário. Para delimitar os símbolos do vocabulário no disco, cada um deles é separado pelo caractere zero
  3. Na terceira, o arquivo texto é percorrido pela segunda vez, sendo seus símbolos novamente extraídos, codificados e gravados no arquivo comprimido

Descrição do Código para Fazer a Descompressão

- O primeiro passo é recuperar o modelo usado na compressão. Para isso, lê o alfabeto, o vetor Base, o vetor Offset e o vetor Vocabulario
- Em seguida, inicia a decodificação, tomando o cuidado de adicionar um espaço em branco entre dois símbolos que sejam palavras
- O processo de decodificação termina quando o arquivo comprimido é totalmente percorrido

Resultados Experimentais

- Mostram que não existe grande degradação na razão de compressão na utilização de *bytes* em vez de *bits* na codificação das palavras de um vocabulário
- Por outro lado, tanto a descompressão quanto a pesquisa são muito mais rápidas com uma codificação de Huffman usando *bytes* do que uma codificação de Huffman usando *bits*, isso porque deslocamentos de *bits* e operações usando máscaras não são necessárias
- Os experimentos foram realizados em uma máquina PC Pentium de 200 MHz com 128 *megabytes* de *RAM*

Comparação das Técnicas de Compressão: Arquivo WSJ

Dados sobre a coleção usada nos experimentos:

Texto		Vocabulário		Vocab./Texto	
Tam (bytes)	#Palavras	Tam (bytes)	#Palavras	Tamanho	#Palavras
262.757.554	42.710.250	1.549.131	208.005	0,59%	0,48%

Método	Razão de Compressão	Tempo (min) de Compressão	Tempo (min) de Descompressão
Huffman binário	27,13	8,77	3,08
Huffman pleno	30,60	8,67	1,95
Huffman com marcação	33,70	8,90	2,02
Gzip	37,53	25,43	2,68
Compress	42,94	7,60	6,78

Pesquisa em Texto Comprimido

- Uma das propriedades mais atraentes do método de Huffman usando *bytes* em vez de *bits* é que o texto comprimido pode ser pesquisado exatamente como qualquer texto não comprimido
- Basta comprimir o padrão e realizar uma pesquisa diretamente no arquivo comprimido
- Isso é possível porque o código de Huffman usa *bytes* em vez de *bits*; de outra maneira, o método seria complicado ou mesmo impossível de ser implementado

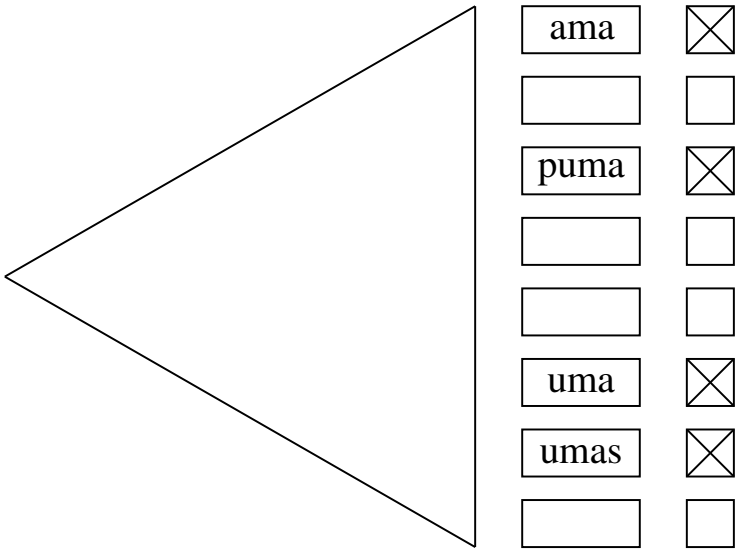
Casamento Exato: Algoritmo

- Buscar a palavra no vocabulário, podendo usar busca binária nesta fase:
  - Se a palavra for localizada no vocabulário, então o código de Huffman com marcação é obtido
  - Senão a palavra não existe no texto comprimido
- A seguir, o código é pesquisado no texto comprimido usando qualquer algoritmo para casamento exato de padrão
- Para pesquisar um padrão contendo mais de uma palavra, o primeiro passo é verificar a existência de cada palavra do padrão no vocabulário e obter o seu código:
  - Se qualquer das palavras do padrão não existir no vocabulário, então o padrão não existirá no texto comprimido
  - Senão basta coletar todos os códigos obtidos e realizar a pesquisa no texto comprimido

Casamento Aproximado: Algoritmo

- Pesquisar o padrão no vocabulário. Nesse caso, podemos ter:
  - Casamento exato, o qual pode ser uma **pesquisa binária** no vocabulário, e uma vez que a palavra tenha sido encontrada a folha correspondente na árvore de Huffman é marcada
  - Casamento aproximado, realizado por meio de pesquisa sequencial no vocabulário, usando o algoritmo Shift-And. Nesse caso, várias palavras do vocabulário podem ser encontradas e a folha correspondente a cada uma na árvore de Huffman é marcada
- A seguir, o arquivo comprimido é lido *byte* a *byte*, ao mesmo tempo que a árvore de decodificação de Huffman é percorrida
- Ao atingir uma folha da árvore, se ela estiver marcada, então existe casamento com a palavra do padrão
- Seja uma folha marcada ou não, o caminhamento na árvore volta à raiz ao mesmo tempo que a leitura do texto comprimido continua

Esquema Geral de Pesquisa para a Palavra "*uma*" Permitindo 1 Erro



Casamento Aproximado Usando uma Frase como Padrão

- **Frase:** sequência de padrões (palavras), em que cada padrão pode ser desde uma palavra simples até uma expressão regular complexa permitindo erros

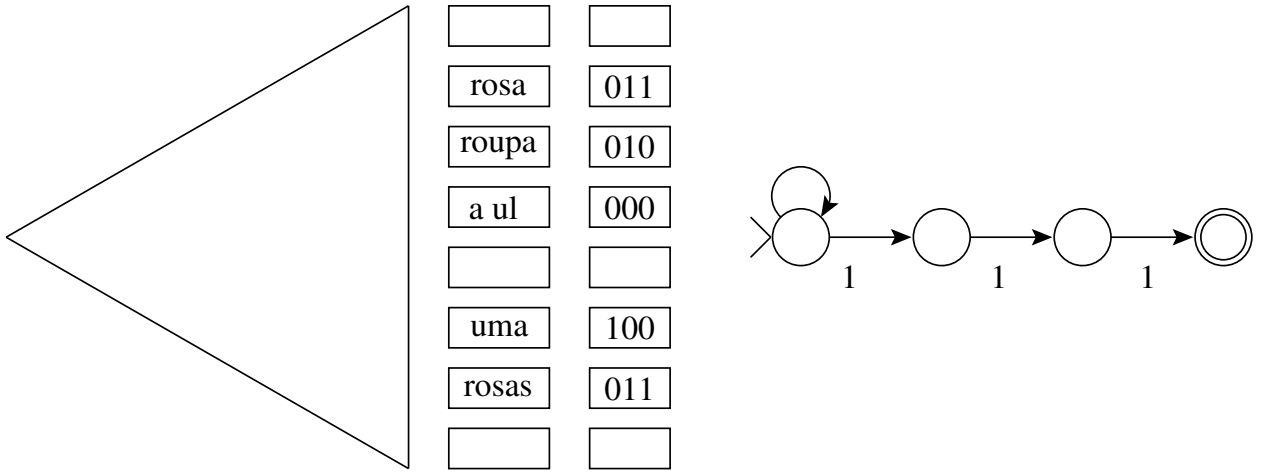
Pré-Processamento:

- Se uma frase tem j palavras, então uma máscara de j *bits* é colocada junto a cada palavra do vocabulário (folha da árvore de Huffman)
- Para uma palavra x da frase, o i-ésimo *bit* da máscara é feito igual a 1 se x é a i-ésima palavra da frase
- Assim, cada palavra i da frase é pesquisada no vocabulário e a i-ésima posição da máscara é marcada quando a palavra é encontrada no vocabulário

Casamento Aproximado Usando uma Frase como Padrão

- O estado da pesquisa é controlado por um **autômato finito não-determinista** de $j + 1$ estados
- O autômato permite mover do estado i para o estado $i + 1$ sempre que a i-ésima palavra da frase é reconhecida
- O estado zero está sempre ativo e uma ocorrência é relatada quando o estado j é ativado
- Os *bytes* do texto comprimido são lidos e a árvore de Huffman é percorrida como antes
- Cada vez que uma folha da árvore é atingida, sua máscara de *bits* é enviada para o autômato
- Um estado ativo $i - 1$ irá ativar o estado i apenas se o i-ésimo *bit* da máscara estiver ativo
- O autômato realiza uma transição para cada palavra do texto

Esquema Geral de Pesquisa para Frase "`uma ro* rosa`"



- O autômato pode ser implementado eficientemente por meio do algoritmo Shift-And
- Separadores podem ser ignorados na pesquisa de frases
- Artigos, preposições etc., também podem ser ignorados se conveniente, bastando ignorar as folhas correspondentes na árvore de Huffman quando a pesquisa chega a elas
- Essas possibilidades são raras em sistemas de pesquisa *online*

Tempos de Pesquisa (em segundos) para o Arquivo WSJ, com Intervalo de Confiança de 99%

Algoritmo	$k = 0$	$k = 1$	$k = 2$	$k = 3$
Agrep	23,8 $\pm$ 0,38	117,9 $\pm$ 0,14	146,1 $\pm$ 0,13	174,6 $\pm$ 0,16
Pesquisa direta	14,1 $\pm$ 0,18	15,0 $\pm$ 0,33	17,0 $\pm$ 0,71	22,7 $\pm$ 2,23
Pesquisa com autômato	22,1 $\pm$ 0,09	23,1 $\pm$ 0,14	24,7 $\pm$ 0,21	25,0 $\pm$ 0,49

Referências

- Moura, E. S., Navarro, G., Ziviani, N. and Baeza-Yates, R. “Fast Searching on Compressed Text Allowing Errors”, *ACM SIGIR*, 1998.
- Moura, E. S., Navarro, G., Ziviani, N. and Baeza-Yates, R. “Fast and Flexible Word Searching on Compressed Text”, *ACM Transactions on Information Systems 18*(2), 2000, 113-139.
- Ziviani, N., Moura, E. S., Navarro, G., and Baeza-Yates, R. “Compression: A Key for Next-Generation Text Retrieval Systems”, *IEEE Computer, 33*(11), 2000, 37-44.
- Gonzalo, N. and Ziviani, N. “Documents: Languages and Properties”. In: Baeza-Yates, R. and Ribeiro-Neto, B. Modern Information Retrieval, Chapter 6, Addison-Wesley, 2011, 203-254 (second edition).
- Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, Cengage Learning, 2010, terceira edição.