Universidade Federal de Ouro Preto
Departamento de Computação – DECOM
Disciplina: BCC203 - Estrutura de Dados II
Professor: Guilherme Tavares de Assis
Resumo IV – Compressão de Textos

Turma: 11
Nome: Pedro Lucas Damasceno Silva
Matrícula: 20.1.4003

- ## INTRODUÇÃO

Consiste na representação de textos originais ocupando menos espaço através da substituição dos símbolos do texto original por outros que ocupam menos *bits* ou *bytes*. Apesar do custo computacional para codificar e decodificar o texto comprimido, a compressão de texto, além de propiciar menor ocupação de memória, também decrementa o tempo de pesquisa (casamento de cadeias) e transferência por um canal de comunicação.

Outros aspectos relevantes no âmbito da compressão de textos são: velocidade de compressão e descompressão (priorizadas de acordo com o contexto), possibilidade de realização de casamento de cadeias diretamente no texto comprimido, a possibilidade de acessar diretamente qualquer uma de suas partes, descomprimindo apenas uma parcela do texto conforme a necessidade, e a razão de compressão entre o arquivo comprimido e o não comprimido (quanto menor, melhor).

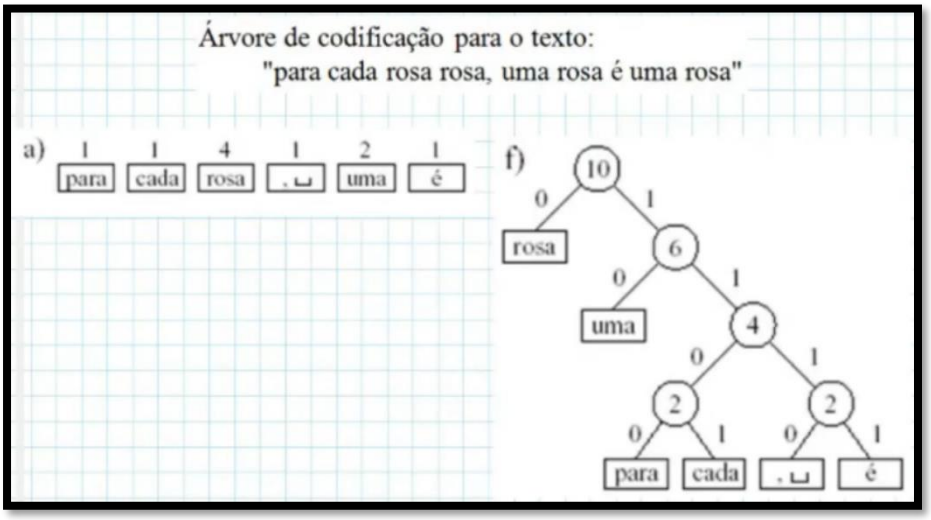
- ## COMPRESSÃO DE HUFFMAN

Proposto em 1952, corresponde à técnica de compressão mais eficaz para textos em linguagem natural e consiste na atribuição de um código único, de tamanho variável, a cada símbolo distinto do texto. Códigos mais curtos são atribuídos a símbolos (tradicionalmente considerados como caracteres) com muita recorrência. Atualmente, palavras são consideradas como símbolos a serem codificados, ao contrário da implementação tradicional. Em média, métodos de Huffman baseados em caracteres e em palavras comprimem o texto em uma razão de 60% e 25% respectivamente, o que evidencia a maior eficiência das implementações atuais em comparação às tradicionais.

Inicialmente, é realizada uma leitura no arquivo com a finalidade de gerar o vocabulário, determinando a frequência de cada palavra de maneira similar ao arquivo invertido estudado anteriormente. Caracteres separadores (vírgulas, pontos, espaços, etc) podem ser tratados como palavras separadas, possuindo também seus códigos associados, ou como parte de outras palavras, o que pode ampliar o vocabulário com variações de uma mesma palavra e, por consequência, diminuir um pouco a eficiência do algoritmo. Em seguida, a frequência é utilizada para atribuir os códigos às palavras (atribuindo os menores às palavras de maior frequência), o texto é comprimido a partir da substituição de cada palavra do texto pelo seu código correspondente e uma árvore de codificação é construída.

- **ÁRVORE DE CODIFICAÇÃO**

Árvore canônica (desbalanceada, possuindo um lado muito mais alto do que outro), construída a partir do vocabulário extraído do texto original. A frequência é considerada de forma que as palavras mais recorrentes são inseridas mais próximas da raiz, e as menos recorrentes em nós mais baixos, construindo assim a árvore desigual em prol da eficiência da pesquisa. O número de nós internos é sempre 1 unidade menor do que a quantidade de nós externos.



O código de cada palavra é a sequência de *bits* construída a partir do caminhamento na árvore (binária), na qual cada aresta à esquerda representa o *bit* zero e, à direita, o *bit* um, a partir da raiz até o nó folha no qual a palavra se encontra. Na árvore ilustrada acima, a palavra ‘para’ possui o código 1100. Para otimizar a compressão em vocabulários extensos, é possível armazenar os códigos referentes a cada palavra juntos ao vocabulário em uma estrutura de dados de pesquisa como uma árvore B, de complexidade logarítmica, por exemplo. Já para otimizar a descompressão é possível simplesmente caminhar na árvore de codificação a partir do código da palavra presente no arquivo comprimido.

- **ALGORITMO DE MOFFAT E KATAJAINEN**

Haja vista o tempo de construção da árvore de codificação e o espaço ocupado por ela a partir de vocabulários muito extensos, o algoritmo de Moffat e Katajainen foi elaborado a partir da codificação canônica e apresenta comportamento linear em tempo e espaço. O algoritmo calcula o comprimento dos códigos ao contrário dos próprios códigos, obtendo a mesma compressão independente dos códigos utilizados.

A entrada do algoritmo é um vetor contendo as frequências das palavras em ordem decrescente. Durante a execução, são criados subvetores distintos temporários que coexistem no mesmo vetor. No caso da árvore de codificação ilustrada acima, o vetor é:

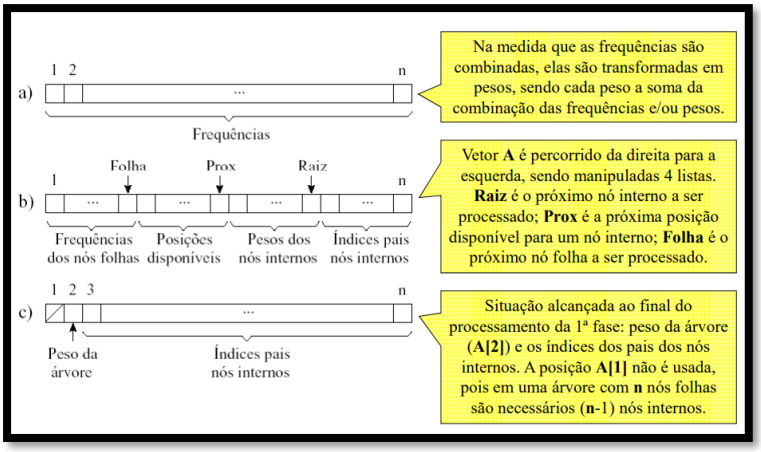
4	2	1	1	1	1

1. Combinação dos nós

A partir do vetor de frequências é gerado um vetor que possui o primeiro índice vazio, o segundo possui o peso da árvore de codificação e os seguintes possuem os índices dos nós internos a partir do segundo nó interno. Considerando a árvore anterior, o vetor obtido durante essa etapa é:

-	10	2	3	4	4

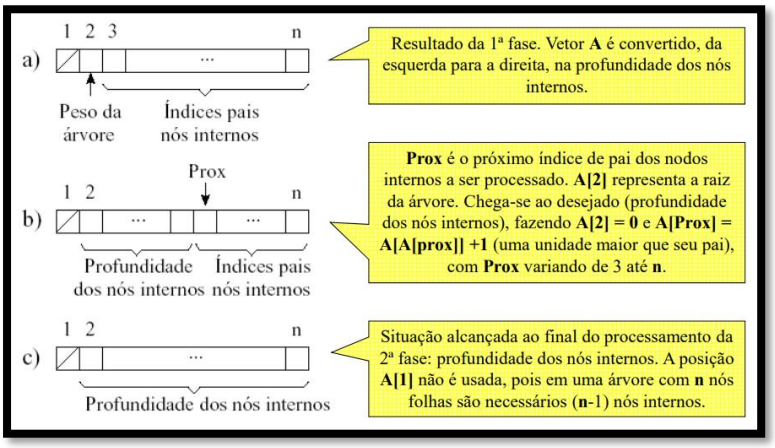
Durante a criação desse vetor, as frequências (lidas a partir da direita para a esquerda) são transferidas ao subvetor controlado pela variável 'Prox', submetidas a um processo e transferidas ao subvetor controlado por raiz até que todos os índices de pais internos sejam calculados. O processo termina quando não houverem mais frequências a serem lidas.



2. Cálculo da profundidade dos nós internos

Substituição dos pesos no vetor anterior pela profundidade dos nós internos. A raiz possui profundidade zero e, consequentemente, os próximos nós possuem profundidade igual à anterior + 1. A partir do vetor anterior, obtemos a partir da segunda fase:

-	0	1	2	3	3

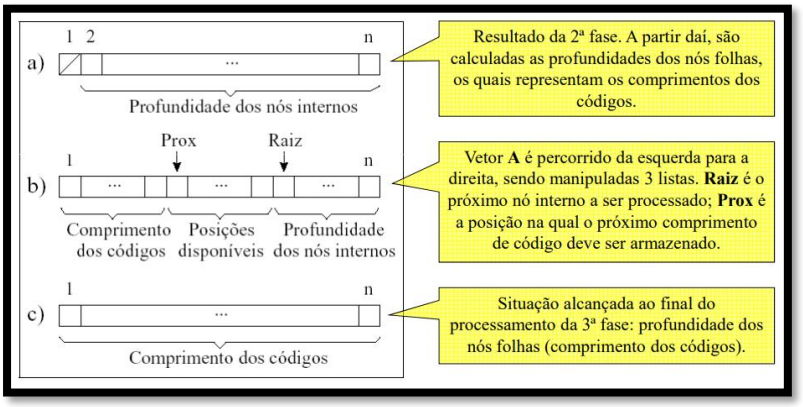


3. Cálculo da profundidade dos nós externos

O vetor de profundidade dos nós internos é lido no sentido normal (da esquerda para a direita). A partir dessa leitura, o cálculo da profundidade dos nós externos é realizado com o auxílio de 2 variáveis (Prox e Raiz),

que definem a posição na qual o próximo comprimento de código deve ser armazenado e o próximo nó interno a ser processado, respectivamente. O vetor obtido a partir da terceira fase é:

1	2	4	4	4	4



- **CÓDIGOS CANÔNICOS**

A partir do vetor de comprimentos dos códigos obtido previamente durante a execução do algoritmo de Moffat e Katajainen, calculamos os códigos canônicos da seguinte forma:

1. O primeiro código (referente à palavra mais frequente do vocabulário) é composto apenas por zeros.
2. Para os demais, adiciona-se 1 ao código anterior e faz-se um deslocamento à esquerda para obtenção do comprimento adequado quando necessário.
3. Os comprimentos dos códigos seguem o algoritmo de Huffman e códigos de mesmo comprimento são inteiros consecutivos.

i	Símbolo	Código Canônico
1	Rosa	0 (0)
2	Uma	10 (2)
3	Para	1100 (12)
4	Cada	1101 (13)
5	,_	1110 (14)
6	É	1111 (15)

Para a realização da descompressão, o vocabulário e seus códigos associados são armazenados no arquivo comprimido antes do próprio conteúdo. Dessa forma, grande parte do volume do texto comprimido se deve ao vocabulário.

- **CODIFICAÇÃO E DECODIFICAÇÃO**

Para diminuir o espaço ocupado pelo vocabulário e os códigos canônicos, é possível gerar os códigos canônicos de forma dinâmica, pulando as etapas de criação dos códigos, conforme a necessidade. Possuindo o resultado obtido a partir do algoritmo de Moffat e Katajainen (vetor de tamanhos dos códigos), calculamos 2 vetores (*base* e *offset*) indexados pelo comprimento do código.

1. Vetor *Base*: para um dado comprimento c, indica o valor inteiro do 1º código com tal comprimento. É calculado de acordo com a fórmula:

$$\text{Base}[c] = \begin{cases} 0 & \text{se } c = 1, \\ 2 \times (\text{Base}[c-1] + w_{c-1}) & \text{caso contrário,} \end{cases}$$

Nº de códigos com comprimento (c-1).

2. Vetor *Offset*: indica o índice no vocabulário da 1ª palavra de comprimento *c*.

A partir dos dois vetores, obtidos a partir do resultado do algoritmo de Moffat e Katajainen, é possível calcular todos os códigos canônicos a partir da posição da palavra no vocabulário sem a necessidade de armazenamento. Basta calcular '*i* – *offset*[*c*] + *base*[*c*]' para obter o valor inteiro e convertê-lo em código binário. Para decodificar, calculamos 'código canônico + *offset*[c] – *base*[c]', que retorna o índice da palavra que referencia o código utilizado no cálculo.

c	Base[c]	Offset[c]
1	0	1
2	2	2
3	6	2
4	12	3

- **COMPRESSÃO**

1. Percorrer o arquivo texto e gerar o vocabulário e a frequência de cada palavra (uma tabela *hash* com tratamento de colisão é utilizada para garantir o custo O(1) das operações de inserção e pesquisa no vetor de vocabulário.
2. Ordenar o vetor vocabulário a partir das frequências, calcular o comprimento dos códigos (Moffat e Katajainen), construir e gravar os vetores *Base*, *Offset* e *Vocabulário* no arquivo comprimido e reconstruir a tabela *hash*, caso tenha se perdido, a partir da leitura do vocabulário no disco.
3. Percorrer novamente o arquivo texto, codificar as palavras extraídas e gravar os códigos correspondentes no arquivo comprimido.

- **DESCOMPRESSÃO**

1. Ler os vetores *Base*, *Offset* e *Vocabulário* presentes no início do arquivo comprimido.
2. Ler os códigos do arquivo comprimido, decodifica-los e gravar as palavras correspondentes no arquivo texto.

A descompressão é evidentemente mais simples e rápida do que a de compressão, dada sua prioridade. O usuário normalmente pode aguardar um pouco mais para arquivar dados, mas é desejável que o acesso aos dados seja o mais próximo possível de imediato.