⌨️

Compressão de Textos

Column	☐
Files	https://www.moodlepresencial.ufop.br/pluginfile.php/954477/mod_resource/content/2/bcc203_compressao-textos.pdf
Unit/Module	Compressão de Textos
Video	https://www.youtube.com/watch?v=4z1fTblbC4M
Week	

👩‍💻 Clara Ribas - 20.1.4008

👩‍💻 Lívia Sousa - 20.1.4029

Introdução

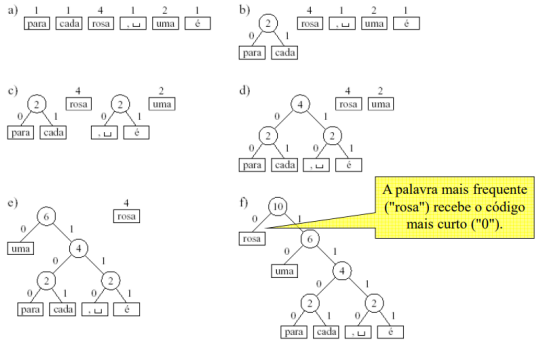
- A compressão do texto consiste em representar o texto original em menos espaço. Para isso, substitui-se os símbolos do texto por outros que ocupam um número menor de bits ou bytes **(Codificação)**. Em seguida, cria-se um novo arquivo com as substituições.
- **Ganho obtido:** O texto comprimido ocupa menos espaço de armazenamento, ocasionando em um menor tempo para realizar pesquisa, leitura, casamento de cadeia, etc neste.
- **Preço a pagar:** Custo computacional para codificar e decodificar o texto, por isso deve-se utilizar uma técnica eficiente.
- Aspectos importantes a serem considerados:
  - **Velocidade de compressão e descompressão:** A compressão pode ser feita de forma mais demorada já que é off-line, mas a descompressão deve ser rápida pois ocorre em tempo real.
  - **Possibilidade de realizar casamento de cadeias diretamente em texto comprimido:** Ocorre de forma muito mais rápida.
  - **Acesso direto a qualquer parte do texto comprimido, possibilitando o início da descompressão a partir da parte acessada:** É importante que o algoritmo permita que a descompressão aconteça em qualquer parte do arquivo, de forma que não seja preciso descomprimir o arquivo inteiro para acessar determinada parte deste.
- A **métrica** que compara a eficiência dos métodos de compressão é a **razão de compressão**, que corresponde à porcentagem que o arquivo comprimido representa em relação ao arquivo original. A ideia é que a razão seja a menor possível.

Compressão de Huffman

- Método de codificação mais utilizado atualmente que propõe:
  - Um **código único**, de tamanho variável, é atribuído a cada símbolo diferente do texto.
  - Os códigos mais curtos são atribuídos a símbolos com frequências altas.
  - As implementações tradicionais do método de Huffman consideram caracteres como símbolo, as implementações atuais consideram as palavras como símbolos, gerando uma razão de compressão bem menor.

- Em termos de compressão, o primeiro passo é realizar um primeira leitura no arquivo à fim de gerar o vocabulário e para cada palavra deste estabelecer a **frequência** da palavra dentro do texto.
- A frequência de cada palavra é utilizada para estabelecer o código de cada palavra. Em seguida, cada palavra é substituída por seu respectivo código.
- A compressão é realizada em **duas passadas sobre o texto**:
  - Obtenção da frequência de cada palavra diferente.
  - Realização da compressão.
- Um texto é composto por **separadores** *(espaço, vírgula, ponto, etc)*, uma maneira eficiente de lidar com eles é considerar o espaço em branco como **verdadeiro separador**, já a vírgula, ponto, etc serão considerados **palavras independentes**, ou seja, terão seus próprios códigos.

Árvore de Codificação



a) Estabelecer vocabulário e frequência de palavras.

b) e **c)** Cada palavra, juntamente com sua frequência, constitui um nó folha da Árvore, a partir das folhas a Árvore é construída de baixo para cima, de forma que os nó folha de **menor frequência** são combinados em uma **sub-árvore**, colocando a **soma da frequência** das folhas consideradas como **nó pai**.

d) É interessante no processo de concepção da árvore, considerar sempre unir as sub-árvores já formadas primeiro, para gerar uma árvore canônica mais efetiva.

- A ideia da Árvore de Codificação é que ela seja uma árvore **canônica**, ou seja, tenha um lado maior que o outro em termos de altura.
- A aresta **esquerda** representa o **bit 0** e a aresta **direita** representa o **bit 1** do código gerado. Logo, o código será a sequência de bits que se leva da raiz até determinada palavra.
- Constrói-se uma **árvore de pesquisa** de forma que, para cada palavra é guardado o código referente a ela.
- Dependendo do tamanho do texto, a construção da árvore pode ser muito custosa. Dessa forma, é interessante simular a construção da árvore por meio de um **vetor**.

Algoritmo de Moffat e Katajainen

- Algoritmo que, baseado na codificação canônica, simula a árvore em um vetor, apresentando comportamento linear em tempo e em espaço.
- O algoritmo **não calcula os códigos** propriamente ditos, mas o comprimento deles. Somente após o cálculo dos comprimentos realiza-se a codificação.
- O primeiro passo do algoritmo constitui na construção de um vetor em que cada posição corresponde à uma palavra do vocabulário.

- Para o texto "para cada rosa rosa, uma rosa é uma rosa", o vetor A é:

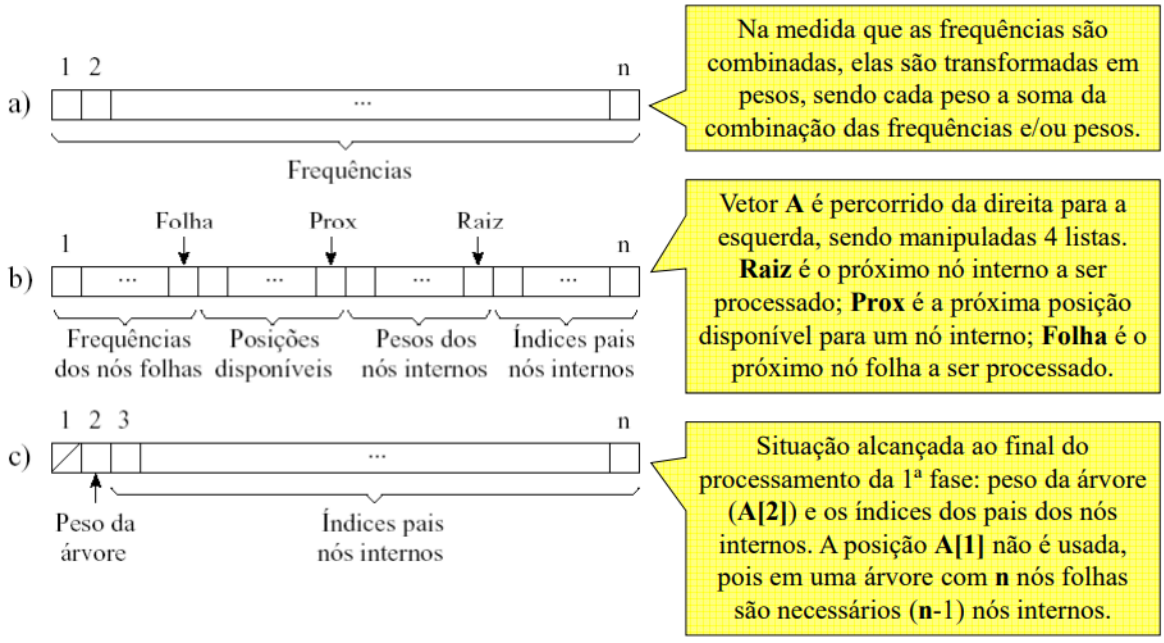
4	2	1	1	1	1

Cada posição guarda a taxa de frequência de determinada e palavra e a estrutura deve ser ordenada de forma decrescente.

- O algoritmo é dividido em **três fases** que ocorrem dentro do próprio vetor, sem a necessidade de construir outra estrutura. São criados sub-vetores temporários que coexistem dentro do próprio vetor.

1. Combinação de nós

- Etapa que realiza a simulação da árvore de codificação.

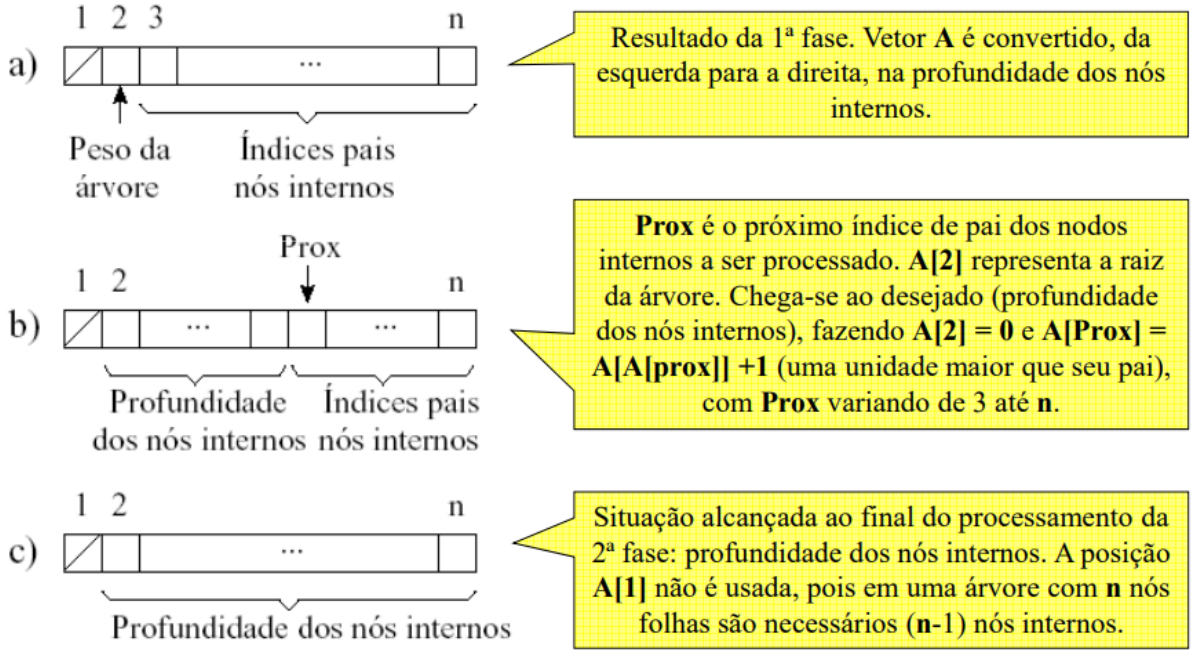


- A primeira posição do vetor final é vazia já que a quantidade de nós internos é menor que a quantidade de frequências.
- A segunda posição guarda o índice da **raiz**. O peso da árvore.
- Para a transformação *(vetor **a** em **c**)* são criados sub-vetores que são controlados pelas variáveis:
  - **Folha:** Controla as frequências que são lidas de trás pra frente no vetor original. Inicialmente tem valor **n** e a medida que as frequências são lidas o valor vai **diminuindo em 1**.
  - **Prox:** A medida em que as folhas são lidas, os pesos são somados, e essa soma é jogada nessa variável, que representa um sub-vetor de posições disponíveis. A partir deles é possível calcular o índice dos pais dos nós internos.
  - **Raiz:** Tais índices são jogados nessa variável, que controla os índices que são gerados.

```
PrimeiraFase (A, n)
{ Raiz = n; Folha = n;
  for (Prox = n;Prox>= 2; Prox--)
  { /* Procura Posicao */
    if ((nao existe Folha) || ((Raiz > Prox) && (A[Raiz] <= A[Folha])))
    { A[Prox] = A[Raiz];  A[Raiz] = Prox;  Raiz = Raiz - 1; /* No interno */ }
    else { A[Prox] = A[Folha];  Folha = Folha - 1; /* No folha */ }
    /* Atualiza Frequencias */
    if ((nao existe Folha) || ((Raiz > Prox) && (A[Raiz] <= A[Folha])))
    { /* No interno */
      A[Prox] = A[Prox] + A[Raiz]; A[Raiz] = Prox; Raiz = Raiz - 1;
    }
    else { A[Prox] = A[Prox] + A[Folha]; Folha = Folha - 1; /* No folha */ }
  }
}
```

2. Determinação das profundidades dos nós internos

- Dado o vetor resultante da primeira fase, calcula-se a profundidade dos nós internos (distância do nó interno em relação à raiz), cujos índices já foram definidos.

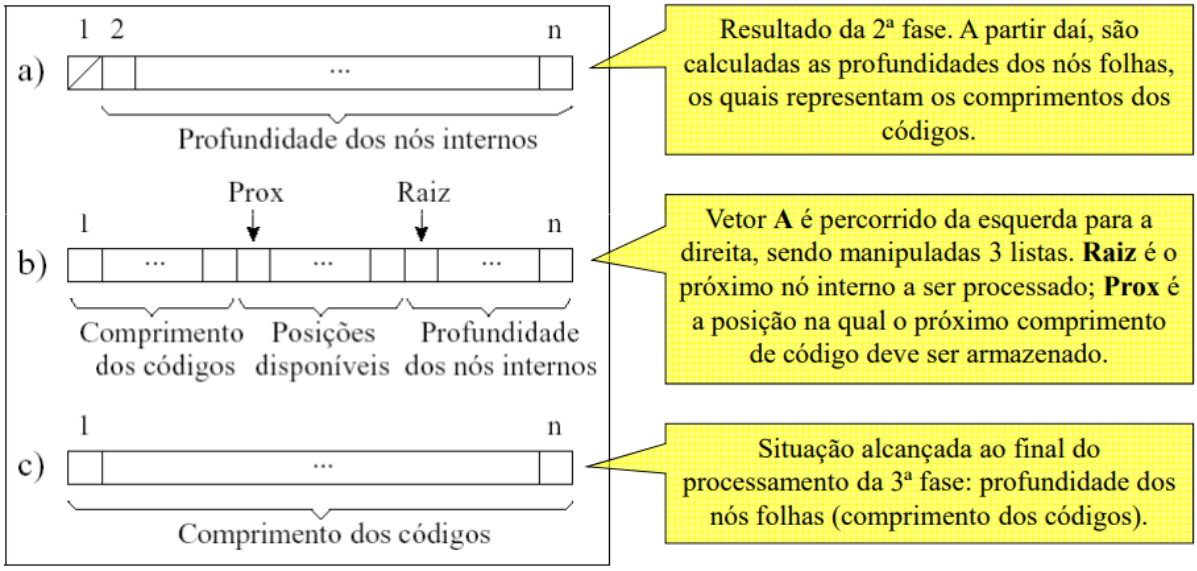


- A altura da própria raiz é **0**, para calcular os níveis seguintes basta **somar 1** ao anterior.

```
SegundaFase (A, n)
{ A[2] = 0;
  for (Prox = 3; Prox <= n; Prox++) A[Prox] = A[A[Prox]] + 1;
}
```

3. Determinação das profundidades dos nós folhas (Comprimento dos Códigos)

- Etapa que calcula a profundidade dos nós externos.



- Utiliza-se de dois sub-vetores para transformar a profundidade dos nós internos no comprimento dos códigos.
- A variável **Raiz** controla a leitura das profundidades dos nós internos a partir da posição 2. É aplicado um cálculo em cima dos valores lidos e o valor do comprimento é colocado na parte anterior do vetor, controlada pela variável **Prox**.

Disp armazena quantos nós estão disponíveis no nível **h** da árvore.
u indica quantos nós do nível **h** são internos.

```
TerceiraFase (A, n)
{ Disp = 1; u = 0; h = 0; Raiz = 2; Prox = 1;
  while (Disp > 0)
  { while (Raiz <= n && A[Raiz] == h) { u = u + 1; Raiz = Raiz + 1; }
    while (Disp > u) { A[Prox] = h; Prox = Prox + 1; Disp = Disp – 1; }
    Disp = 2 * u; h = h + 1; u = 0;
  }
}
```