Aula 6

Compressão de dados

Objetivos

Esperamos que, ao final desta aula, você seja capaz de:

- compreender a importância da compressão de dados;
- conhecer algoritmos de compressão de dados (freqüência de caracteres, Huffman e LZW).

Pré-requisitos

Para compreender satisfatoriamente esta aula, é necessário conhecer as árvores binárias e os tipos de arquivos (estudados nas aulas um e quatro deste caderno, respectivamente). As árvores binárias são importantes porque uma das técnicas aqui discutida (algoritmo de Huffman) faz uso de árvores binárias em sua solução. Além disso, é importante conhecer também os tipos de arquivos, uma vez que todos os algoritmos apresentados nesta aula fazem uso de arquivos.

Introdução

Compactar, comprimir, *zipar*. Eis expressões comuns do jargão da informática que, provavelmente, já foram usadas por você ao se referir à compressão de arquivos.

A compressão de dados ou, em inglês, *data compression*, consiste na utilização de um conjunto de métodos e outros pormenores práticos com o intuito da redução do espaço armazenado em unidades de memória secundária ou mesmo primária de um sistema computacional. Um arquivo comprimido terá seu tamanho reduzido, como saída resultante da aplicação de um algoritmo de compactação de alguma aplicação, como, por exemplo, *gzip*, *winzip* ou *winrar*. Essas aplicações também incluem algoritmos de empacotamento, a fim de permitir que múltiplos arquivos sejam compactados e concatenados dentro de um único arquivo resultado, aumentando a praticidade do processo.

A necessidade de compressão de dados é algo comumente relacionado à vida das pessoas, seja de maneira direta ou indireta. Na *internet*, a compressão ajuda a diminuir a quantidade de tráfego na grande rede, aumentando a velocidade de navegação, realização de *downloads* de arquivos e visualização de vídeos. Na vida *offline*, arquivos compactados são preferíveis quando há interesse de armazenamento de maior número de dados possível no menor espaço de memória secundária disponível, como em *pen-drives*, *memmory cards*, discos rígidos e demais unidades de memória.

Muitos arquivos de extensões famosas, como **pdf** (textos, *e-books*), **mp3** (áudio, músicas), **gif** (imagens, fotos), **zip** (arquivos em geral), **mpg** (vídeos), utilizam algoritmos de compactação em suas concepções. Arquivos são seqüências de *bytes*, caracteres alfabéticos, numéricos e símbolos imprimíveis ou não. Se começarmos a raciocinar sobre essas seqüências, é provável que comecemos a imaginar maneiras de reorganizar ou representar tais seqüências de uma maneira que a quantidade de *bytes* possa ser reduzida.

É dentro desse contexto de raciocínio, que muitos teóricos se tornaram (e tornam-se) criadores de diversos algoritmos de compressão de arquivos, com finalidade de uso nas mais diversas áreas da computação. Cabe a você, a partir do entendimento da importância de se comprimir dados e do conhecimento de alguns algoritmos de compressão de arquivos, obter uma base de conhecimento no assunto. A partir disso, estará pronto para encarar os desafios da área da compactação, quando surgirem diante de você, exigidos pelo próprio mercado.

Estando consciente da importância da compressão de dados para a sociedade e para o profissional da informação, verá, nesta aula, alguns famosos algoritmos de compactação de dados.

6.1 Freqüência de caracteres

Esse algoritmo é utilizado para compactar arquivos contendo texto alfabético. Considere como exemplo o arquivo texto a seguir com 32 caracteres:

AAAAAHHHFGGGGBBPEEECCCCCCDLLLLRR

Sobre esse algoritmo, Szwarcfiter e Markenzon (1994, p. 293) dizem que é necessário determinar "a quantidade de símbolos idênticos consecutivos existentes no texto. Cada uma das subseqüências máximas de símbolos idênticos do texto é substituída por um número indicando a freqüência do símbolo em questão".

O texto exemplificado anteriormente seria compactado como:

5A3H1F4G2B1P3E6C1D4L2R

Essa representação compactada do exemplo possui 22 caracteres, o que resulta em uma economia de 10 *bytes* (lembre-se de que cada caractere ocupa 1 *byte* de memória).

Podemos melhorar ainda mais essa compactação. Para isso, basta definir que a ausência do número que indica a freqüência do símbolo implica freqüência igual a um. A partir dessa nova situação, a compactação ficaria assim:

5A3HF4G2BP3E6CD4L2R

Resultando em uma economia de mais 3 *bytes*, agora 13 bytes do total da mensagem original. Você deve estar pensando: e se o texto tiver dígitos numéricos? Nesse caso, a freqüência do dígito concatenada com ele mesmo poderia ser confundida com um número de mais de um algarismo. Isso é um problema, mas podemos adotar algum símbolo especial para sanar esse problema. Assim se pode empregar o símbolo @ para indicar que, na seqüência, será apresentado um símbolo do texto original e não uma freqüência. Por exemplo:

KKKK4444PP888TJJJJJ22222NN

Resulta na representação compactada:

4K4@42P3@8T5J5@22N

Como o texto original possui 26 e a representação compactada 18, houve uma economia de 8 *bytes*. A seguir, é apresentada a função que implementa o algoritmo de freqüência de caracteres.

```
public static String compactarSequencia(String sequencia)
{
  StringBuffer sbCompactado = new StringBuffer();
  int i = 0;
  while(i < sequencia.length())
  {
    int nOcorrencias = 0;
    char ch = sequencia.charAt(i);
    int j = i;
    while(j < sequencia.length() &&
       sequencia.charAt(j)==ch){
     nOcorrencias++;
     j++;
     i++;
    }

    String cAdd;
```

```
    if (Character.isDigit(ch))
      cAdd = "@"+ch;
    else
      cAdd = Character.toString(ch);

    if(nOcorrencias==1)
     sbCompactado.append(cAdd);
    else
    if(nOcorrencias>1)
     sbCompactado.append(Integer.
     toString(nOcorrencias)+cAdd);
  }
  return sbCompactado.toString();
}
```

6.2 Algoritmo de Huffman

É, basicamente, a mesma idéia por trás do algoritmo de freqüência de caracteres, no entanto utiliza árvore binária.

Considere um alfabeto e um texto constituído somente por símbolos desse alfabeto. Ao invés de expressarmos a freqüência no novo texto resultante da compactação, como ocorre no algoritmo de freqüência de caracteres, queremos codificar o texto original em um texto codificado somente por *bits* (0 ou 1).

Tenenbaum, Langsan e Augestein (1995) apresentam a idéia desse algoritmo em um exemplo. Suponha que um alfabeto consista nos símbolos A, B, C e D e que códigos são atribuídos a esses símbolos, como segue na Tabela 1.

Tabela 1 *Exemplo de códigos para o algoritmo de Huffman.*

SÍMBOLO	CÓDIGO
A	010
B	100
C	000
D	111

Fonte: Tanenbaum, Langsan e Augenstein (1995).

De acordo com a tabela anterior, o texto ABACCDA seria codificado, substituindo cada símbolo por seu código, como 01010001000000111010. Note que foram usados três *bits* para cada símbolo. Assim é preciso 21 *bits* para o novo texto codificado. Se para cada símbolo forem associados os códigos da Tabela 2, poderemos ter uma codificação menor.

Tabela 2 *Outro exemplo de códigos para o algoritmo de Huffman*.

SÍMBOLO	CÓDIGO
A	0
B	110
C	10
D	111

Fonte: Tanenbaum, Langsan e Augenstein (1995).

Assim a nova mensagem, codificada a partir do texto original, fica 0110010101110, necessitando agora de somente 13 *bits*. Em suma, a idéia por trás do algoritmo de Huffman é encontrar uma codificação capaz de diminuir o tamanho, em *bits*, da mensagem original.

Isso acontece porque o menor código (representando a letra A) aparece com mais freqüência do que os códigos mais extensos (representando as letras B e D). Desse modo, em textos maiores que tenham símbolos que raramente apareçam, a economia é substancial (TENENBAUM; LANGSAN; AUGENSTEIN, 1995).

Você deve estar se perguntando: como posso trocar D (um caractere) por 111 (três caracteres) e conseguir compactação? A resposta está no primeiro período do curso, na codificação ASCII. Lembre-se de que, no código ASCII, todo caractere tem um número associado entre 0 e 255. Para representar todos esses números binariamente, são necessários 8 *bits* ($8^2 = 256$). Assim, para representar cada caractere são necessários 8 *bits*. A idéia do algoritmo de Huffman está em representar os símbolos que ocorrem com maior freqüência com menos *bits* e os que acontecem com menor freqüência com mais *bits* (8, por exemplo). Por isso é possível compactar com Huffman.

Tenenbaum, Langsan e Augenstein (1995, p. 351) informam que, "em geral, os códigos não são formados pela freqüência de caracteres dentro de uma única mensagem isolada, mas por sua freqüência dentro de um conjunto inteiro de mensagens".

Nesse sentido, textos da língua portuguesa devem ser codificados de acordo com a freqüência relativa de ocorrência dos símbolos na língua portuguesa para obter um melhor resultado.

Devemos tomar cuidado para que o código de um símbolo não seja prefixo de outro código. Isso visa a evitar confusão entre um código e um prefixo. Por exemplo, veja que na Tabela 2 o código do símbolo A é 0 e que 0 não é prefixo dos outros códigos, ou seja, nenhum dos outros códigos começa com 0. Todos os demais símbolos começam seus códigos com 1, o único que tem o segundo *bit* igual a 0 é o símbolo C; e assim por diante.

Szwarcfiter e Markenzon (1994, p. 294) complementam afirmando que "uma vantagem da utilização de códigos prefixo é a facilidade existente para executar as tarefas de codificação e decodificação".

As entradas do algoritmo de Huffman são:

- quantidade de símbolos do alfabeto do texto original;
- um vetor com a quantidade de vezes que cada símbolo aparece no texto (vetor de freqüência).

Os **passos** para **implementação do algoritmo de Huffman** são apresentados a seguir:

1. Varra o texto contando os símbolos e montando o vetor de freqüências.
2. Construa um nó para cada símbolo do alfabeto do texto. O nó deve ter ponteiros suficientes para encadear com o pai e as subárvores esquerda e direita. Cada um desses nós são, inicialmente, raízes de diferentes árvores e não possuem filhos.
3. Enquanto houver mais de uma árvore:
   - 3.1 Encontre nas raízes as duas freqüências que aparecem menos.
   - 3.2 Construa um nó que combine os nós identificados em 3.1 em um novo nó e armazene no novo nó a soma das freqüências.
   - 3.3 Atribua como subárvore esquerda do nó criado em 3.2 o nó com menor freqüência e o outro como subárvore direita.
4. Varra o texto original substituindo cada símbolo por seu código representado na árvore.

Vamos exemplificar o algoritmo passo a passo e entender como os códigos da Tabela 2 foram obtidos. Suponha um texto contendo a mensagem ABACCDA.

1. Varra o texto contando os símbolos e montando o vetor de freqüências.

Símbolos = 4

A = 3

B = 1

C = 2

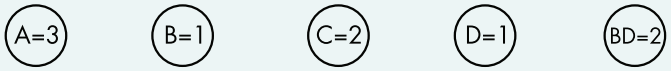
D = 1

2. Construa um nó para cada símbolo do alfabeto do texto.

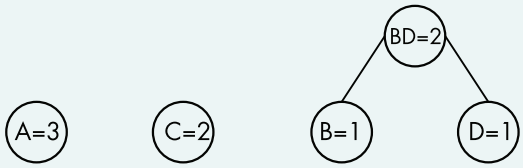
3. Enquanto houver mais de uma árvore:

3.1. Encontre nas raízes as duas freqüências que aparecem menos;

3.2. Construa um nó que combine os nós identificados em 3.1 em um novo nó e armazene no novo nó a soma das freqüências;

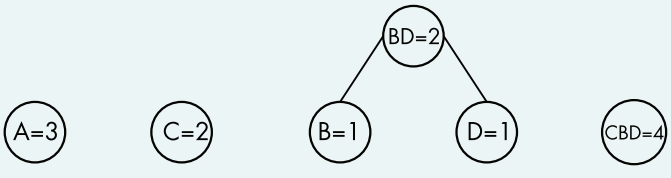


3.3. Atribua como subárvore esquerda do nó criado em 3.2 o nó com menor freqüência e o outro como subárvore direita;

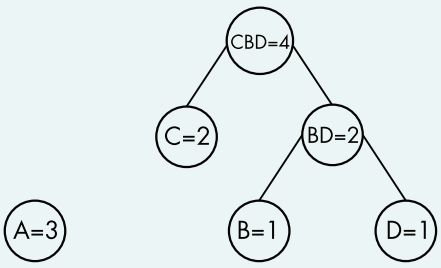


3.1. Encontre nas raízes as duas freqüências que aparecem menos;

3.2. Construa um nó que combine os nós identificados em 3.1 em um novo nó e armazene no novo nó a soma das suas freqüências;

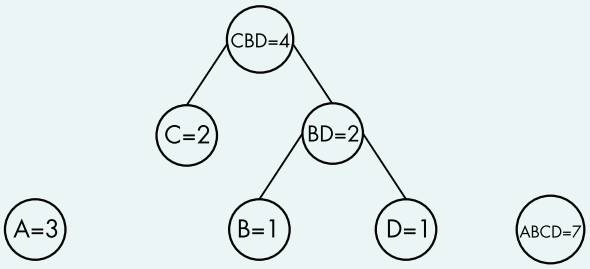


3.3. Atribua como subárvore esquerda do nó criado em 3.2 o nó com menor freqüência e o outro como subárvore direita;

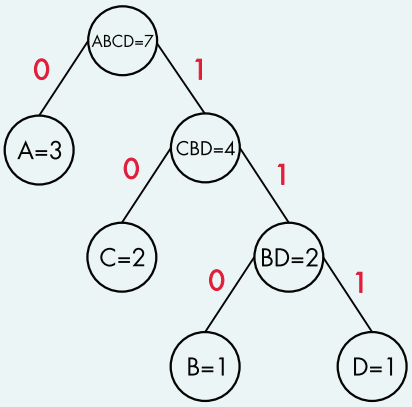


3.1. Encontre nas raízes as duas freqüências que aparecem menos;

3.2. Construa um nó que combine os nós identificados em 3.1 em um novo nó e armazene no novo nó a soma das suas freqüências;



3.3. Atribua como subárvore esquerda do nó criado em 3.2 o nó com menor freqüência e o outro como subárvore direita.



4. Varra o texto original substituindo cada símbolo por seu código representado na árvore.

Desse modo, observe que os códigos surgem do percurso da raiz até a folha que armazena o símbolo do alfabeto original. Para cada visita a um filho à esquerda, um 0 (zero) será obtido, por outro lado, cada visita a um filho à direita, um 1 é obtido. Assim o símbolo com seus respectivos códigos são:

A = 0

B = 110

C = 10

D = 111

Repare que o algoritmo de Huffman não vale somente para a compactação de textos, apesar de ser sua principal aplicação. Dependendo da distribuição de freqüência dos símbolos, o ganho com o método pode ser maior ou menor, ou seja, quanto menos uniforme for a distribuição dos símbolos, maior será o ganho.

6.3 LZW

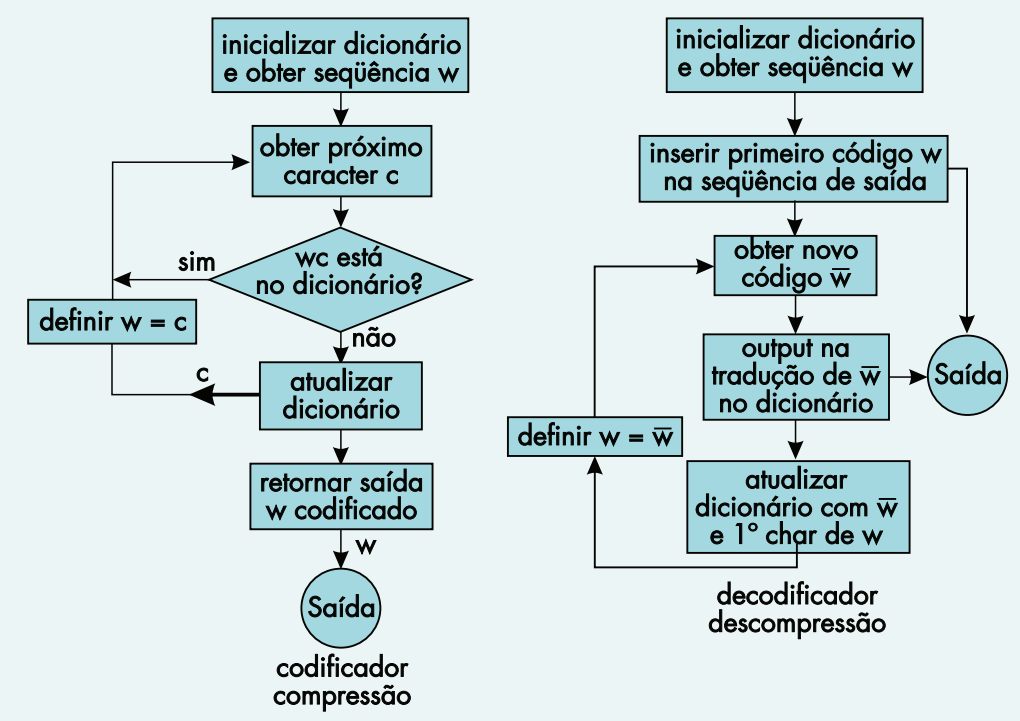
O algoritmo conhecido como LZW (Lempel-Ziv-Welch) é derivado dos nomes dos seus desenvolvedores: Abraham Lempel, Jakob Ziv e Terry Welch.

Assim como no algoritmo de Huffman, o LZW procura substituir seqüências de símbolos por códigos. Para obter a compressão de dados, os códigos devem ser menores que as seqüências representadas por eles. Na prática, esse algoritmo consegue um maior nível de compressão de dados que os outros métodos citados anteriormente.

O LZW é usado para compactar arquivos binários em geral, a exemplo das imagens, vídeos e dos textos. Iremos nos ater na exemplificação de tal algoritmo por meio de seu pseudocódigo. Cabe a você procurar implementá-lo, em Java, a fim de exercitar sua capacidade nessa área.

A seguir, na Figura 1, é apresentado em fluxograma os algoritmos de codificação (compressão) e decodificação (descompressão) do LZW.

Figura1 *O cerne do algoritmo LZW. Codificador (compressão) e decodificador (descompressão).*



Fonte: UCPEL ([s.d.]).

Saiba mais

Para obter mais informações sobre os métodos de compactação de Huffman e LZW, inclusive, com direito a um *applet* Java animado sobre a compressão com Huffman e LZW, acesse o sítio: <http://www.cs.sfu.ca/CC/365/li/squeeze/>. Esse sítio também permite o *download* do código fonte em Java do *applet* que anima o algoritmo LZW.

Após estudarmos alguns algoritmos de compressão de dados, chegamos ao fim da aula. Não se esqueça de pôr em prática os novos conceitos aprendidos. Use a linguagem Java para implementar e fazer seu próprio compactador de dados. Você pode desenvolver um novo aplicativo para concorrer com o Winzip e o Winrar.

Síntese da aula

Nesta aula, tomamos consciência da importância dos **algoritmos de compactação**, ou compressão para a sociedade da informação, incluindo suas implementações para suprir diversas facetas que a computação tem. Tomamos conhecimento de três **métodos de compressão**. Um deles foi o **algoritmo de freqüência de caracteres**, que considera somente vetores alfanuméricos, executando compressão em caso de repetição de caracteres, um após o outro, dentro de uma seqüência. Vimos o **algoritmo de Huffman**, seguindo a mesma idéia de freqüência de caracteres, porém utilizando-se de uma árvore binária. Por fim, observamos o coração do **método LZW**, muito famoso e utilizando para compactar todo tipo de arquivo, binário ou textual.

Atividades

1. Após ter estudado sobre a definição de compressão de dados e a sua importância, analise as afirmativas e, em seguida assinale a alternativa correta.

   I. Uma das utilizações comumente conhecidas e úteis das ferramentas de compressão de dados é o empacotamento de dois ou mais arquivos de dados (xls, txt, doc, etc.) em um único arquivo compactado.

   II. A compactação de um arquivo é útil em várias situações. Entre elas, o armazenamento de cópia de segurança (*backup*), com finalidade de transporte, distribuição ou *upload*.

   III. Uma vez realizada a compressão de arquivos via um aplicativo popular, como o Winzip, teremos um arquivo resultante contendo o empacotamento de arquivos e diretórios. Assim, para que possamos utilizar os arquivos do pacote, será necessária, antes, a extração dos arquivos

desejados, ou seja, executar um processo para descomprimir e salvar em disco os arquivos na forma original.

IV. A *internet* é uma das maiores beneficiárias da compressão de dados, pois arquivos menores são mais rapidamente transmitidos na grande rede, proporcionando um ganho maior de performance e velocidade de navegação, por parte dos usuários.

a) Todas as afirmativas estão corretas.

b) Somente as afirmativas I, II e III estão corretas.

c) Somente as afirmativas II, III e IV estão corretas.

d) Somente as afirmativas I, III e IV estão corretas.

2. Após ler sobre a importância da compressão de dados, coloque-se em lugar de um consultor de empresas e explique como uma empresa de serviço de hospedagem de sítios poderia economizar dinheiro a partir da utilização da compressão de dados.

3. Como vimos nesta aula, existem diversos algoritmos de compressão de dados, cada um com suas características peculiares. Sobre esses algoritmos de compressão de dados, analise os itens a seguir e assinale a alternativa correta.

I. O algoritmo de freqüência de caracteres recebe como entrada um vetor de caracteres alfanuméricos e, como saída, devolve um vetor, também de caracteres alfanuméricos, com exceção do caractere '@', sempre menor, no mínimo em um caractere do que a seqüência de entrada.

II. O algoritmo de Huffman, entre os três explicados na aula, é o que apresenta a saída com maior nível de compressão para todos os casos.

III. O algoritmo LZW é utilizado para comprimir qualquer tipo de arquivo, diferentemente do algoritmo de freqüência de caracteres, que é indicado para compressão de seqüência de caracteres alfabéticos e numéricos.

IV. O algoritmo de Huffman segue a mesma idéia do algoritmo de freqüência de caracteres, porém utilizando-se de árvore binária.

a) Somente as afirmativas I e II e estão corretas.

b) Somente as afirmativas I e III estão corretas.

c) Somente as afirmativas II e III estão corretas.

d) Somente as afirmativas III e IV estão corretas.

4. Escreva um pequeno programa em Java, que possibilite a decodificação do algoritmo de compressão por freqüência de caracteres. Ele será justamente

o "caminho de volta" (descompressão) do algoritmo "de ida" (compressão). Teste seu algoritmo com as entradas expostas nesta aula, bem como outras entradas de sua preferência.

Comentário das atividades

Na **atividade um**, a resposta correta é a letra **(a)**. Todas as afirmativas estão de acordo com os assuntos explanados em aula e estudados em materiais referenciados.

Na **atividade dois**, você deve ter pensado como um consultor de tecnologia da informação e focado em uma empresa de hospedagem de sítios. Assim, para a empresa economizar dinheiro, a partir do uso da compactação, ela poderia manter os arquivos de usuários compactados, economizando espaço em disco, diminuindo o gasto com aquisição de novas unidades de disco rígido ou de fitas de *backup*.

Na **atividade três**, a resposta correta é a letra **(d)**. A afirmativa (I) erra ao afirmar que a saída é um vetor sempre menor que o de entrada: no caso da entrada não tem nenhuma repetição de caracteres adjacentes, a *string* resultante será igual à *string* de entrada. A alternativa (II) também está errada, pois o algoritmo LZW é o que oferece um maior nível de compressão, entre os três algoritmos expostos em aula. As demais alternativas estão de acordo com os assuntos explanados em aula e estudados em materiais referenciados.

Finalmente, na **atividade quatro**, você deve ter desenvolvido um aplicativo em Java para decodificação de seqüências de caracteres (*strings*) compactadas com o algoritmo de compressão por freqüência de caracteres. Você deve ter dado uma olhada na função *public static String compactarSequencia(String sequencia)* exposta nesta aula e até a copiado no programa feito, para poder gerar saídas codificadas que pudessem ser copiadas, servindo de teste para a função de decodificação que você criou. A função deu certo, uma vez que a saída que ela retorna, mostrada em tela, por exemplo, pela mensagem *System.out.println(resultado)*, é a seqüência de caracteres original, que servira de entrada para a função de compactação.

Se você respondeu corretamente a essas questões, atingiu os dois objetivos propostos para esta aula: compreender a importância da compressão de dados e conhecer algoritmos de compressão de dados (freqüência de caracteres, Huffman e LZW).

Referências

SZWARCFITER, Jayme Luiz; MARKENZON, Lílian. **Estruturas de dados e seus algoritmos**. Rio de Janeiro: LTC, 1994.

TENENBAUM, Aaron M.; LANGSAN, Yedidyah; AUGENSTEIN, Moshe J. **Estrutura de dados usando C**. São Paulo: Makron Books, 1995.

UCPEL. **Compressão – LZW (Lempel-Ziv-Welch)**. Disponível em: <http://atlas.ucpel.tche.br/~tst/lzw.html>. Acesso em: 17 set. 2008.

Na próxima aula

Você conhecerá os grafos. Um grafo é uma estrutura de dados com capacidade de representar problemas complexos e resolvê-los com algoritmos relativamente simples. Que venham os grafos!

Anotações