Unidade 6 - Conceitos de compressão de dados

6.1 - Primeiras palavras

Uma das utilidades mais óbvias para a compressão de dados é poupar espaço em disco. Se uma determinada quantidade de dados pode ser armazenada em um espaço menor, então mais espaço sobrará para outros dados.

As aplicações da compressão de dados, porém, extrapola esta utilidade imediata, pois envolve também a transferência de informações. Para que se obtenha eficiência, quanto mais dados puderem ser transmitidos em um mesmo intervalo de tempo, melhor será a situação. Isso pode ser obtido aumentando-se a velocidade das linhas de comunicação ou, então, reduzindo-se a representação dos dados para que consumam menos da banda de transmissão.

A compressão de dados é uma forma de reduzir a quantidade de símbolos necessários para guardar uma mesma informação, escolhendo símbolos que consumam menos espaço. Em um sistema de controle de uma adega, por exemplo, os tipos de vinho podem ser indicados por "tinto", "branco" ou "rosé". Um programador pode usar, por exemplo, uma cadeia de caracteres de 7 bytes para armazenar esta informação, pensando em uma linguagem como C. Essa quantidade de bytes é suficiente para guardar qualquer um dos tipos definidos. Porém a escolha de "T", "B" e "R" pode representar a mesma informação, usando apenas um byte, o que representa uma economia de espaço pela redução dos símbolos necessários. Pode-se, ainda, ir além, optando-se por usar apenas 2 bits para o armazenamento, com 00 para tintos, 01 para brancos e 10 para rosé (o símbolo 11 não seria utilizado). Em relação à representação inicial, há uma redução de 7 bytes para 0,25 byte para a representação dos dados.

Os símbolos que são escolhidos podem, ainda, não ter comprimento igual para cada informação diferente. Assim, informações mais frequentes podem usar menos bits, enquanto os símbolos mais longos são deixados para os dados que aparecem menos.

Naturalmente, o código deve poder ser revertido. Assim, a partir dos símbolos adotados para representação de um dado, o dado original deve ser obtido de volta[*].

6.2 - Os símbolos e a compressão

A compressão de dados se dá quando, a partir de uma representação de um conjunto, se obtém outra representação equivalente, esta última usando menos espaço. Por exemplo, se existe um arquivo texto, ocupando s bytes em uma representação qualquer (ASCII, por exemplo), então uma codificação diferente pode ser utilizada, obtendo-se uma versão comprimida com t bytes. O "ganho" obtido com a redução de tamanho pode ser medido pela proporção do número de bytes de diferença entre o arquivo original e o comprimido, proporcionalmente ao tamanho original. Na prática, é calculada a expressão $(s - t)/s$. Este valor dá a proporção do ganho e é conhecido como **taxa de compressão**. A compressão, então, ocorre pela substituição de símbolos por códigos (que também são símbolos) de menor comprimento.

Supondo um arquivo texto contendo apenas as letras A, B e C como exemplo inicial, sabe-se que cada letra ocupa 1 byte, se for usada a tabela ASCII como referência. Se o texto contiver um total de 40 letras, então ocupará 40 bytes. Porém, se às letras forem associados os símbolos binários 00, 01 e 10, respectivamente para A, B e C, então, em um único byte poderão ser armazenadas 4 letras (00100101 guardaria ACBB, por exemplo). Usando este novo código, as mesmas 40 letras ocupariam apenas 10 bytes, com taxa de compressão de 75%. A partir destes 10 bytes "comprimidos" é possível restaurar o arquivo original.

Acrescentando mais uma informação ao exemplo, considere-se que no texto existam 20 letras A, 10 letras B e 10 letras C, em qualquer ordem. Uma segunda opção para escolha de símbolos seria usar o dígito binário 0 para a letra A e os símbolos 10 e 11 para as letras B e C, respectivamente. Assim, 01010110 significaria ABBCA. Cada letra A usaria um único bit, de forma que 20 letras A consumiriam o equivalente 2,5 bytes (20 bits). Para as letras B e C, cada uma usaria também 2,5 bytes. A codificação final, portanto, usaria 7,5 bytes[†]. A taxa de compressão, para esta nova codificação, atinge 81,25%.

A Figura 6-1 mostra ambas as formas de codificação para uma sequência de símbolos. Na Figura 6-1(a) é apresentada a sequência original de símbolos. Usando-se dois bits para cada símbolo, sendo 00

[*] Este texto trata da chamada **compressão sem perdas**, que é aquela em que o dado original é obtido do dado comprimido exatamente como estava antes. Alguns métodos de compressão admitem perda de informação, como é o caso da compressão de áudio e vídeo. Quanto a maior a compressão de um arquivo MP3, por exemplo, pior a qualidade do áudio e, a partir do áudio comprimido, não é mais possível recuperar a qualidade original.

[†] Quando o número de bytes não tem tamanho inteiro, significa que alguns bits do último byte são desprezados por não fazerem parte da representação.

para A, 01 para B e 10 para C, obtém-se a compressão apresentada na Figura 6-1(b). Neste caso, são necessários 5,5 bytes para a representação da cadeia original, de 22 caracteres. Na Figura 6-1(c) é apresentada a compressão usando-se o símbolo 0 para A, 10 para B e 11 para C, chegando-se a uma cadeia de 4,25 bytes após a compressão.

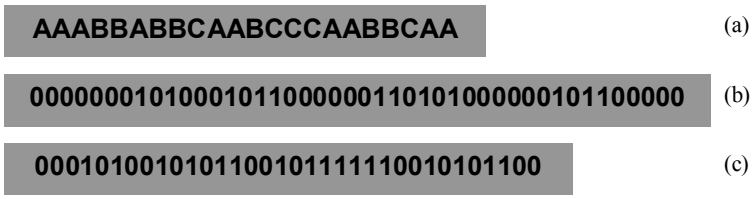

Figura 6-1. Exemplos de codificação: (a) Sequência original de símbolos; (b) Compressão usando código de 2 bits por símbolo; (c) Compressão usando 1 bit para o símbolo A e 2 bits para os símbolos B e C. (Detalhes no texto.)

O segredo da compressão de dados é a escolha de novos símbolos que substituam os símbolos originais e que reduzam o tamanho dos dados ao mínimo. Para obter um bom código para tradução de símbolos, alguns pontos devem ser considerados [6]:

1) Cada palavra de código deve corresponder a um único símbolo.
2) Durante a interpretação do código comprimido, a interpretação deve ser sequencial.
3) O comprimento de um código de símbolos mais frequentes deve ser menor que o de símbolos menos frequentes.
4) Códigos de menor comprimento não devem ser excluídos do conjunto de símbolos.

O primeiro item é apenas uma resolução de ambiguidade, pois estabelece que cada símbolo original é mapeado em um único novo símbolo.

Interpretar sequencialmente o código comprimido, por outro lado, é importante para que não seja necessário "olhar para frente" para tomar uma decisão de como interpretar o código. Em outras palavras, significa que a interpretação do código não pode depender de símbolos que venham depois do símbolo atualmente sendo interpretado. Se, por exemplo, o símbolo X tiver como código o símbolo 1 e o símbolo Y for codificado como 10, então, para decidir o significado uma sequencia iniciada com 1, é necessário olhar o próximo símbolo: se for 0, então significa Y, caso contrário, significa X. Porém, para decidir, é preciso ir para o símbolo seguinte e, depois, retornar para a interpretação. Esta característica é chamada de **propriedade de prefixo**. Isso quer dizer que um dado código nunca é a parte inicial (prefixo) de outro código.

A restrição de que códigos mais longos não sejam associados a símbolos mais frequentes permite codificações de menor comprimento. Se um símbolo A aparece 50 vezes em um conjunto de dados e o símbolo B aparece apenas 5 vezes, então claramente um código menor deve ser associado ao símbolo A e um menor ao B.

Finalmente, não há sentido que códigos curtos não sejam utilizados em uma codificação. Ou seja, se existe um código de menor comprimento sem uso, então ele deve ser empregado no lugar de outro código mais longo. Há, assim, consequente redução do comprimento geral da compressão.

6.3 - Codificação de Huffman

O código de Huffman [3] é um procedimento relativamente simples que, a partir da frequência de ocorrência de cada símbolo, gera um código ótimo.

Considerando-se um conjunto inicial de dados, a obtenção do código binário de Huffman é dada pelos seguintes passos:

1) Calcular a frequência de ocorrência de cada símbolo diferente, gerando para ele uma árvore de um único nó, contendo o símbolo e sua probabilidade de ocorrência (dada pelo número de ocorrências dividido pelo número total de símbolos).
2) Criar uma lista com as árvores, em ordem crescente do valor da probabilidade de ocorrência armazenada no nó raiz.
3) Remover da lista as duas primeiras árvores, criando uma nova árvore. Gerar, para isso, uma nova raiz, ficando como ramo direito a primeira árvore da lista e como ramo esquerdo a segunda. Na raiz deve ser inserido o valor de probabilidade dado pela soma das probabilidades de cada uma das duas raízes originais. A nova árvore criada deve ser reinserida na lista, mantendo a ordenação pelas probabilidades de ocorrência. Este processo deve ser repetido até que a lista contenha uma única árvore (para cuja raiz a probabilidade de ocorrência será 1).
4) Associar símbolo (código binário) 0 a cada ramo esquerdo da árvore, enquanto o símbolo 1 deve ser associado a cada ramo direito.

5) Criar um símbolo binário, de comprimento variável, para cada símbolo original. Notando-se que cada símbolo original é uma folha da árvore gerada, o novo símbolo binário que deve substituir o original é dado pelos bits da travessia da árvore, iniciando-se na raiz e terminando na folha correspondente.

Como exemplo, considere-se um arquivo texto contendo apenas as letras A, B, C e D. O número de letras A presente no arquivo é 67, o de letras B é 32 e os de C e D, respectivamente, 49 e 93. O tamanho total do arquivo é 241. Assim, as probabilidades de ocorrência dos símbolos A, B, C e D são, respectivamente, 0,278, 0,133, 0,203 e 0,386.

A criação de uma lista com árvores de apenas um nó contendo os símbolos e probabilidades é ilustrado na (a). A lista já se encontra ordenada crescentemente pela probabilidade de ocorrência.

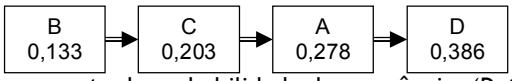

Figura 6-2. Lista em ordem crescente de probabilidade de ocorrência. (Detalhes no texto.)

A partir da lista, o procedimento do item 3 descrito acima é aplicado. Inicialmente são unidos os nós de símbolos B e C, criando-se para isso uma raiz com probabilidade 0,336, dada pela soma das probabilidades das duas raízes. Esta nova árvore, reinserida na lista, ocupa a segunda posição, para manter a ordenação. A Figura 6-3(a) mostra este primeiro resultado.

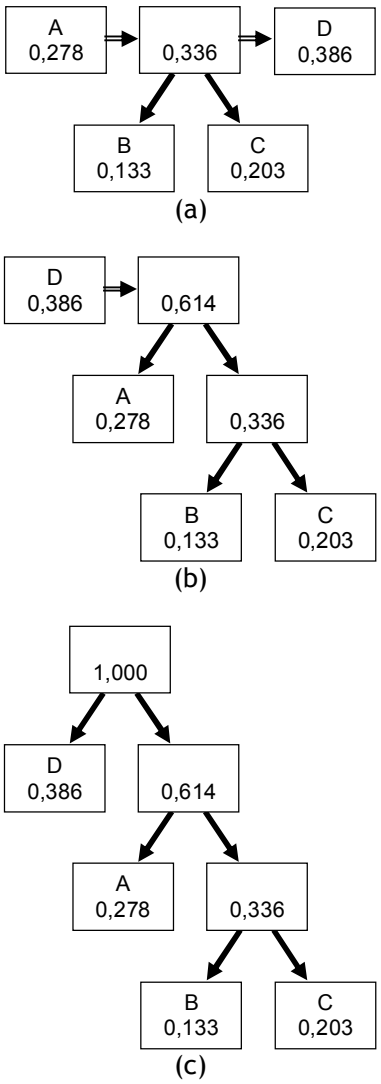

Figura 6-3. Passos da criação da árvore de códigos. (Detalhes no texto.)

Como há, ainda, três árvores na lista da Figura 6-3(a), as duas primeiras são unidas em uma nova árvore, com probabilidade acumulada de 0,641. Ao ser inserida na lista, a nova árvore ocupa a última posição. O resultado é apresentado na Figura 6-3(b), na qual ainda existem duas árvores na lista.

Uma última união reúne os dois itens da lista em uma única árvore, obtendo-se a árvore final, ilustrada na Figura 6-3(c).

Esta árvore tem associado a cada ramo os dígitos 0 e 1, respectivamente para os ramos esquerdo e direito de cada nó. Esta árvores de códigos é apresentada na Figura 6-4.

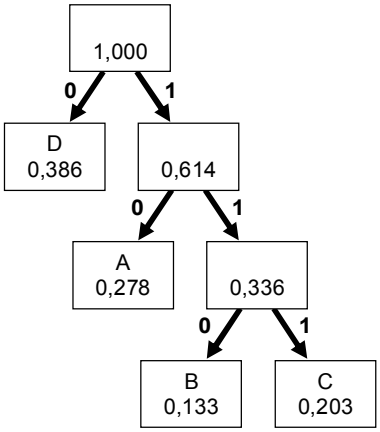

Figura 6-4. Árvore de códigos obtida pelo algoritmo de Huffman. (Detalhes no texto.)

O símbolo que deve ser associado, então, a cada letra, é dado pelos dígitos binários dos ramos, seguindo-se o caminho descendente na árvore, da raiz até a folha que contém o símbolo. Desta forma, o símbolo 0 é associado à letra D; o símbolo 10 à letra A, 110 à letra B e, finalmente, o símbolo 111 à letra C.

Para finalizar o processo de compressão, cada letra do arquivo original é substituída pelo código binário derivado pela árvore, gerando uma sequência binária total. Como ilustração do processo, se no texto houver a sequência BCDA, a sequência binária 110111010 a substituirá. A cada 8 bits da saída do processo de codificação será montado um novo byte para o arquivo compactado, sendo que o último byte pode ser "incompleto", o seja, somente alguns de seus bits podem ser significativos. No arquivo destino devem ser armazenados tanto os códigos novos (comprimidos) quanto a tabela com os símbolos originais, para que a conversão de volta seja possível.

O processo de descompressão dos dados segue o caminho inverso. Dada a tabela de associação dos símbolos originais e dos códigos utilizados, a sequência de bits comprimida é percorrida e interpretada. Assim, reconhecido um zero, o símbolo D é colocado na saída. Caso seja um 1, o próximo valor é consultado; se for 0, então a saída é o símbolo A e, se não for, o próximo bit decidirá entre o B e o C. É importante destacar que os quatro pontos destacados anteriormente como requisitos para um código de compressão são atendidos pela codificação de Huffman.

6.4 - Codificação LZW

O algoritmo conhecido por LZW foi proposto por Abraham Lemple, Jakob Ziv e Terry Welch. É caracterizado como um algoritmo adaptativo, ou seja, um algoritmo que se adapta aos dados de entrada para gerar os símbolos de saída. A codificação é uma evolução dos algoritmos LZ77, proposto por Lemple e Ziv em 1977, e LZ78, de 1978.

O princípio de operação do método é substituir sequências de símbolos do conjunto de dados original por códigos de menor comprimento, obtendo assim a compressão dos dados. As sequências de símbolos que são substituídas são armazenadas em uma tabela, conhecida como **dicionário**.

O Algoritmo 6-1, adaptado diretamente de [6], mostra um pseudo-código de alto nível para a compressão pelo LZW.

Algoritmo 6-1

```
1  crie uma tabela com todas as letras
2  crie uma cadeia de entrada s com a primeira letra da entrada
3
4  enquanto houver dados de entrada faça
5        obtenha o próximo caractere de entrada c
6        se a sequência s + c existe na tabela então
7              s ← s + c
8        senão
9              gere como saída um símbolo para s
10             insira a cadeia s + c na tabela
11             s ← c  { recomeça a cadeia com o novo caractere }
12       fim-se
13 fim-enquanto
14
```

```
15 gere como saída um símbolo para s
```

Na prática, para cada símbolo diferente há um código, mas também há códigos para cadeias de símbolos. Quanto maiores forem as cadeias e maior seu número de ocorrências, maior será a compressão, visto que apenas um código mais curto substituirá toda uma série de símbolos.

O dicionário é criado, inicialmente, tendo uma entrada para cada símbolo esperado. Retomando o exemplo do texto que contém apenas as letras A, B, C e D, haverá uma entrada para cada uma destas quatro letras. A Tabela 6-I mostra os valores iniciais para este conjunto de símbolos. A posição é sequencial e indica o código de saída. Como o número de linhas depende dos valores dos dados usados na entrada, o tamanho final do dicionário é, em princípio, indeterminado.

Tabela 6-I. Tabela que representa o dicionário de símbolos inicial da compressão LZW.

Posição	Símbolo
0	A
1	B
2	C
3	D

A partir da criação inicial do dicionário, a cadeia de entrada pode ser processada. Assim, o algoritmo é seguido e o dicionário recebe mais entradas a partir do momento em que concatenações de símbolos vão sendo acrescentadas.

Como exemplo, pode-se supor que o início do conjunto de dados seja composto pela sequência AABBBAACBBBA... Deste modo, a cadeia de interpretação se inicia com o primeiro símbolo, no caso o A e é feita a leitura do próximo símbolo, também um A. Como resultado desta situação, é gerada a saída do código 0 (posição de A na tabela) e é acrescentada ao dicionário a sequência AA na posição 4. Ainda neste passo, a sequência de entrada é substituída pelo segundo símbolo, coincidentemente um A. O novo ciclo se inicia com a obtenção do próximo caractere, o B. Como AB não está no dicionário, então é feita a saída para o símbolo A (valor 0 novamente), a sequência AB é inserida no dicionário e a cadeia em análise é reiniciada para o símbolo B. O processo se repete para as demais entradas.

A Tabela 6-II mostra, passo a passo, o acompanhamento do algoritmo para a criação do dicionário para a cadeia de entrada deste exemplo.

Tabela 6-II. Dicionário e acompanhamento das entradas e saídas para o processamento da cadeia AABBBAACBBBA para a compressão LZW.

Posição	Símbolo
0	A
1	B
2	C
3	D
4	AA
5	AB
6	BB
7	BBA
8	AAC
9	CB
10	BBB
11	BA

Entrada	Caractere atual	Saída
A	A	0
A	B	0
B	B	1
B	B	
BB	A	6
A	A	
AA	C	4
C	B	2
B	B	
BB	B	6
B	A	1
A	...	

A descodificação segue a interpretação dos símbolos codificados, fazendo a consulta na tabela e gerando de volta os símbolos originais. Naturalmente tanto a tabela quanto a sequência codificada são necessárias para obter a versão completa novamente.

As questões que envolvem este algoritmo são os tempos de busca na tabela, que pode chegar a tamanhos relativamente grandes. Assim, otimizações que envolvam o tempo de busca e a economia de espaço são envolvidas no algoritmo, mas não fazem parte da discussão deste texto. Uma discussão interessante sobre estes aspectos pode ser vista no livro de Drozdek [6].

6.5 - Considerações finais

A compressão de dados é um assunto importante e, como tal, envolve muito mais que os exemplos citados neste texto. Os algoritmos discutidos envolvem apenas compressão de dados sem perdas, dando uma visão limitada da extensão do assunto. Porém, para compressão de dados sem perdas, tanto Huffman

quanto LZW permitem uma visão interessante sobre duas perspectivas diferentes de se obter a redução do volume físico de dados sem perder a informação armazenada.

Muitos programas disponíveis no mercado que fazem compressão de dados utilizam combinações de métodos de compressão diferentes, combinando as vantagens de cada um deles. Estas vantagens e desvantagens devem levar em conta não somente as taxas de compressão obtidas, mas também a complexidade do algoritmo e o tempo que é consumido para comprimir e descomprimir os dados. Em particular, o tempo é importante quando a compressão é utilizada em meios de transmissão de dados, na qual a redução do volume de bytes que devem ser transmitidos não pode ser prejudicada pelo tempo que a compressão leva para ser feita em uma ponta e desfeita na outra.