ELE-32 Introdução a Comunicações

Aula 2- Codificação de Fonte

September 14, 2016

1 Introdução

Nem sempre uma fonte de dados está sendo representada da forma mais eficiente possível. Havendo limites de quantos bits por segundo podem ser transmitidos através de um canal, devemos representar a informação da forma mais eficiente possível. Este processo se chama codificação de fonte.

A codificação de fonte é uma transformação de uma variável ou sequência aleatória de símbolos em uma palavra ou sequência de palavras-código. Esta transformação tem como objetivo reduzir a repetição de informação, ou seja, reduzir a redundância presente na varável aleatória. Do ponto de vista de Teoria de Informação, isto é o mesmo que reduzir a quantidade de informação mútua entre símbolos a serem transmitidos. A codificação pode ou não causar perda de informação. A perda ocorre se o processo é irreversível, isto é, é impossível obter exatamente o valor original, sempre, a partir da palavra código. Métodos populares sem perda de informação são: Huffman, Golumb-Rice e Lempel Ziv. Métodos populares com perda de informação são por exemplo .mp3, .jpeg e .avi.

1.1 Códigos de Huffman

Um método que não causa perda de informação é a codificação de Huffman. Neste código, a palavra código (que representa um símbolo a ser transmitido) tem comprimento é variável. É um código de prefixo livre: nenhuma palavra código é prefixo de outra. Isto permite identificar o início e término de cada palavra código caso várias sejam transmitidas sequencialmente.

Neste código, as palavras código são atribuídas de acordo com a distribuição de probabilidade (necessariamente conhecida) dos símbolos da fonte. O algoritmo para obtenção das palavras código (binárias) e relação com os símbolos pode ser descrito como:

1. Ordene os símbolos de acordo com a probabilidade deles ocorrerem. Associe a cada símbolo um nó, cujo peso deve ser proporcional à probabilidade do símbolo ocorrer.

2. Junte os dois nós com menor peso existentes, gerando assim um novo nó. O peso deste novo nó é a soma dos pesos dos nós unidos.

3. Repita o procedimento do item anterior até que haja somente um nó com peso obrigatoriamente igual a soma dos pesos dos nós originais. A estrutura resultante é uma árvore com este nó como raiz e os nós associados a cada símbolo nas extremidades.

4. Atribua a cada ramo um valor binário de forma que, a partir da raiz, ramos saindo do mesmo nó tenham valores diferentes. O rótulo binário associado a cada símbolo é a sequência de bits gerada ao se percorrer a árvore da raiz para as extremidades.

Um exemplo deste algoritmo está apresentado na figura abaixo.

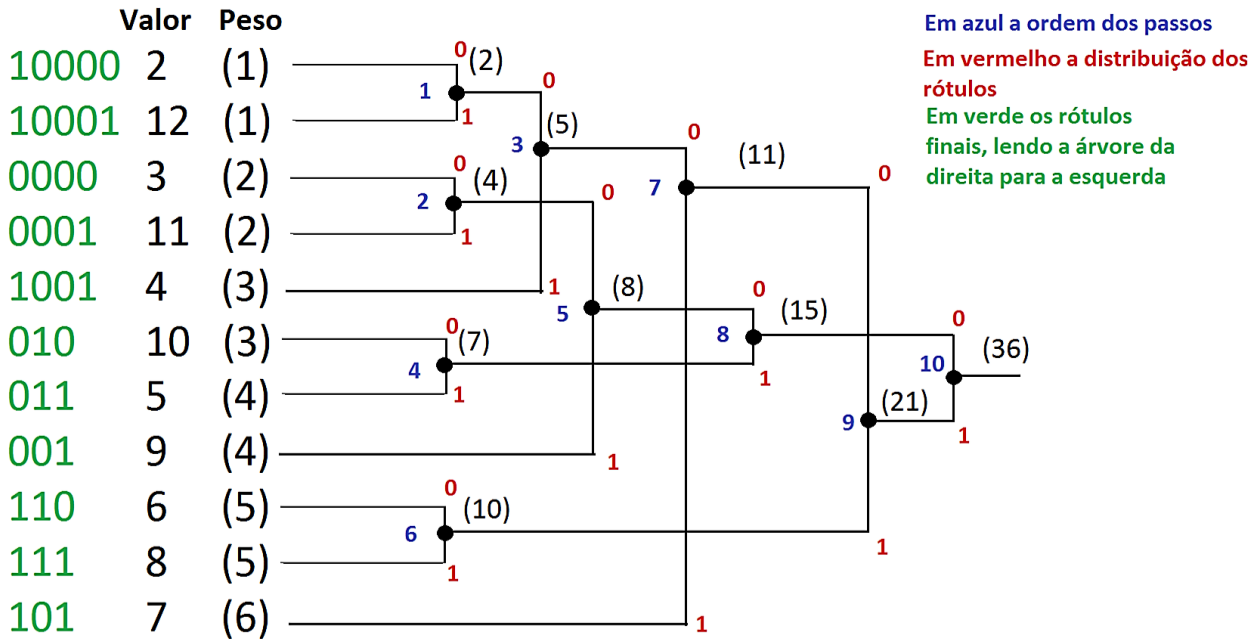

Figure 1: Exemplo de árvore e rótulos resultantes para os símbolos "Valor" com peso "Peso". As probabilidades de cada símbolo são

1.2 Códigos Lempel Ziv

Um problema dos códigos de Huffman é que é necessário o conhecimento a priori exato da distribuição dos símbolos para que o código gere a menor representação possível dos símbolos. Além disso, só é possível representar a informação com uma quantidade de bits igual a sua entropia se as probabilidades de todos os símbolos for igual a 2^{-k}, onde k é um valor inteiro e não precisa ser o mesmo para todos os símbolos.

O algoritmo de Lempel Ziv e suas variantes elimina a necessidade de conhecimento a priori da distribuição dos símbolos. Ele trabalha de forma iterativa, se beneficiando de repetições presentes na sequência de símbolos. Há variações: LZ77, LZ78, LZW, LZSS, etc..

Conceitualmente, estes algoritmos constroem um dicionário indexado de sequências, isto é, sequências são armazenadas e indexadas em um dicionário $Dic[i]$. Quando uma sequência diferente das já armazenadas aparece, ela é descrita através das sequências já presentes. O dicionário pode no início conter os símbolos fundamentais, o que no caso de texto equivale a dizer que contém todas as letras, símbolos de pontuação de texto, símbolos de organização (EOL, EOF), etc.

1.2.1 Algoritmo de codificação

Há dois algoritmos: um de codificação (compactação) e outro de decodificação(descompactação). O algoritmo de compactação deve realizar os seguintes passos:

1. Inicie o dicionário $Dic[i]$ com todas as sequências que possuem somente um símbolo. O primeiro índice i deve ser igual a 1 (em binário), o segundo 10, etc.. Armazene o número de entradas no dicionário como sendo igual a i.

2. Incremente o valor de i em 1.

3. Leia a sequência de símbolos do arquivo a ser compactado até que uma sequência não presente no dicionário se apresente. Esta sequência é necessariamente a concatenação de uma sequência S já existente no dicionário com um novo símbolo c. A sequência S está presente na posição $Dic[j]$, com $j < i$

4. Transmita o endereço da sequência S já existente no dicionário, isto é, o valor de j, utilizando bits. O número de bits necessários para representar o endereço de S depende do número de entradas no dicionário: Encontre b tal que $i \leq 2^b$. Assim, por exemplo, quando $i = 128$, são necessários $b = 7$ bits para descrever qualquer um dos endereços anteriores. Quando $i = 129$, são necessários $b = 8$ bits.

5. O símbolo c do item 3 será o primeiro símbolo da próxima sequência a ser processada.

6. Se ainda há símbolos a serem processados, retorne ao passo 2.

7. Repita o procedimento até que todos os símbolos da fonte tenham sido processados. O último símbolo é necessariamente EOF (end of file).

8. A qualquer momento interrompa o algoritmo se a sequência de entrada terminou. Transmita a última sequência não processada.

Deve-se transmitir tanto o dicionário obtido no item 1 quanto a sequência de bits que representam os endereços obtida no item 3.

A complexidade deste algoritmo aumenta na medida em que o valor de i aumenta, pois é necessário comparar a sequência $S + c$* com os i elementos do dicionário. Por este motivo é comum limitar o valor de i e reiniciar o dicionário sempre que $i >$ algum limite preestabelecido

Exemplo:

- Sequência de entrada: *abcbababcbcabac*
- Sequência de saída: $abc \cdot 010100110101000100101000111000 \cdot c$

O dicionário resultante está na tabela 1.

1.2.2 Algoritmo de decodificação

O algoritmo de decodificação deve saber o dicionário com todas as sequências que contém somente um símbolo, isto é, o dicionário obtido no item 1 do algoritmo de compactação. Este dicionário inicial contém i entradas. A partir disso, é possível determinar o restante da sequência de símbolos da seguinte forma:

1. Leia uma quantidade b de bits da sequência compactada, onde b é o menor valor tal que $i \leq 2^b$. Este é o endereço e.

2. Determine $S = Dic[e]$. Não determinamos nesta etapa o valor de c, mas este valor será determinado na próxima iteração.

3. Concatene S à saída descompactada. Faça $i = i + 1$.

4. Caso ainda haja mais endereços a serem processados, retorne ao primeiro passo. Caso contrário, concatene o restante do arquivo compactado à saída.

*Aqui, a soma indica a concatenação da sequência S com o símbolo c.

i (decimal)	$Dic[i]$	j (decimal)	b	j (binário)
1	a	-	1	-
2	b	-	1	-
3	c	-	2	-
4	ab	1	2	01
5	bc	2	3	010
6	cb	3	3	011
7	ba	2	3	010
8	aba	4	3	100
9	abc	4	4	0100
10	cbc	6	4	1010
11	ca	3	4	0011
12	abac	8	4	1000

Table 1: Dicionário resultante da compactação da sequência do exemplo

2 Atividade

2.1 Programa

Gere um programa que analisa um texto qualquer e aplica o algoritmo Lempel-Ziv como descrito anteriormente, gerando assim um arquivo compactado. O arquivo compactado deve conter toda a informação necessária para sua decodificação.

Aplique-os a diversos textos (grandes) do site https://www.gutenberg.org/ (ou https://www.gutenberg.org/MIRRORS.ALL[†])com idiomas diferentes. Utilize as versões .txt para realizar a análise. É possível que o texto em si influencie no resultado. Espera-se no mínimo a análise de pelo menos 4 idiomas, 3 textos para cada idioma. Elimine o texto padrão inicial e final de cada arquivo. Considere símbolos especiais tais como pontuação,espaços como caracteres, EOL, EOF, etc..

Limite o tamanho do seu dicionário em L entradas. Quando o tamanho do dicionário for igual a L, reinicie-o(mantendo todas as sequências com um símbolo). Este valor é conhecido pelo algoritmo de descompactação. Utilize $L = 100, 1000, 10000, ...$, o quanto o seu computador permitir.

Pode-se utilizar qualquer idioma de programação.

Os programas são considerados aceitáveis se eles conseguem, ao compactar e descompactar o arquivo, retornar ao arquivo original sem nenhuma diferença.

2.2 A entregar

Espera-se no relatório:

- Uma descrição sucinta do seu algoritmo de compactação e de descompactação
- Uma tabela com os resultados, indicando qual o tamanho em bytes do arquivo original, do arquivo compactado, % de compactação, tempo de processamento, etc.
- A resposta para as seguintes perguntas:
    1. Qual é a entropia do arquivo compactado? Trate o seu arquivo compactado como uma sequência de bits.

[†]Caso seja necessário utilizar um mirror do site principal, utilize o arquivo GUTINDEX.zip para encontrar livros. Abra o arquivo incluso GUTINDEX.ALL com um editor de texto simples como por exemplo Notepad.

2. Como a entropia do seu arquivo compactado se compara com a entropia do texto, como calculada no laboratório anterior? Explique bem estes resultados.
3. Qual é a influência de L no tempo de compactação, de descompactação e no tamanho do arquivo compactado?
4. Como o seu algoritmo se compara em relação a algoritmos comerciais como o WinZip, WinRar, 7Zip, etc?
5. O livro/idioma selecionado tem influência na eficiência do seu algoritmo?

- Gráficos, equações, diagramas, etc., que forem úteis.
- **Desafio**: Trate a sua sequência de entrada como uma sequência de bits. O resultado final é melhor ou pior?

Seja sucinto. É possível responder todas as perguntas de forma satisfatória, incluindo o desafio, em menos de 5 páginas. Siga o formato de artigos do IEEE, disponível em https://www.ieee.org/publications_standards/publications/authors/author_templates.html (Templates for Transactions)

3 Referências

- https://en.wikipedia.org/wiki/Entropy_(information_theory)
- https://www.gutenberg.org/ : fonte de textos
- https://en.wikipedia.org/wiki/UTF-8 : Descrição de UTF-8, se necessário.