Projeto 1 de Fundamentos de Compressão de Sinais
Código de Huffman

Giordano Süffert Monteiro - 17/0011160

I. Descrição do Algoritmo

A. Codificador

O programa criado, primeiramente espera que o usuário informe o nome a partir do diretório atual, com a opção de ter informado um diretório de destino junto com a chamada do programa. Após informado o nome do arquivo de leitura, ele é lido por inteiro e com isso é gerado um dicionário com todos os símbolos e suas probabilidades baseado nas aparições que ocorrem no arquivo.

Com essa informação, é criado um código de huffman qualquer ao seguir os passos esperados:

1) Os símbolos são ordenados de acordo com sua probabilidade, sendo dado o bit 0 para o código de menor probabilidade e o bit 1 àquele de segunda menor probabilidade, sempre à esquerda dos já existentes. Caso o símbolo escolhido seja uma junção de vários outros, o bit é dado a todos seus componentes.
2) Esses símbolos são juntados para criar um novo que considerará a probabilidade de ambos em conjunto e em seguida são removidos.
3) Os dois primeiros passos são repetidos até sobrarem somente dois símbolos, em que um receberá o bit 0 e o outro o bit 1, da mesma forma que foi descrito no primeiro item.

Assim, começam a ser geradas as informações que serão escritas no arquivo da compressão, começando pelo byte header de formato '0xfX', em que X indicará o tamanho do padding em bits no final do arquivo. Em seguida, é gerada uma lista com o tamanho em bits dos códigos criados, além de uma sequência de 256 bits que indicará quais os símbolos que são utilizados no arquivo em questão, sendo numerados de 0 a 255, da direita à esquerda.

Após a definição dessas estruturas, é gerado um novo código de huffman baseado nos comprimentos obtidos na criação do primeiro código, porém dessa vez será utilizada uma estrutura de árvore seguindo a ordem lexicográfica dos símbolos e sempre priorizando um nó à esquerda, sendo explicado em mais detalhes na próxima subseção.

Com isso, é iniciada a escrita no novo arquivo comprimido, que manterá o mesmo nome do arquivo base porém com uma extensão .bin, e estará ou no mesmo diretório desse arquivo ou em outro qualquer caso tenha sido informado na execução do programa. Primeiramente, é escrito o byte do header, seguido pela sequência de 32 bytes que indica os símbolos existentes e pelos comprimentos de cada um deles (1 byte para cada).

Finalmente, o arquivo base é lido novamente e seus símbolos são codificados byte a byte e escritos no novo arquivo, sendo adicionado um padding no final para completar o último bytes caso necessário. O valor desse padding então é colocado nos últimos 4 bits do header '0xfX'.

B. Módulo da árvore

Esse módulo implementa a estrutura de uma árvore criada especificamente para sua utilização na codificação e decodificação em questão. Para tal, é criada uma classe 'Node' e uma função 'make_tree_code'.

Um objeto da classe 'Node' têm um valor próprio em bits e apresenta um método para criar seus filhos (repassando o seu valor e adicionando um bit 0 para o da esquerda e um bit 1 para o da direita), um para indicar para toda a árvore que seu valor está sendo utilizado, e outro para fazer um crescimento ou busca na árvore a fim de gerar um código de tamanho informado.

Já a função 'make_tree_code' têm a única função de, a partir de uma lista de símbolos e comprimentos, iniciar a criação de uma árvore que gerará os códigos de comprimentos informados para esses símbolos, retornando um dicionário com essas informações.

C. Decodificação

No programa que realiza a decodificação, primeiramente é esperado o nome do arquivo a ser decodificado a partir do diretório atual e, igual ao codificador, pode ser informado um diretório de destino para o resultado na chamada do programa. Dessa forma, é iniciada a leitura desse arquivo, sendo primeiro extraído o byte de header informando o padding existente; seguido pelos 32 bits indicando quais os símbolos existentes, gerando um lista com esses símbolos; e pelos bytes contendo o comprimento do código de cada um deles, gerando uma lista com esses valores.

Com isso, da mesma forma que o codificador, é gerado o código de huffman através de uma árvore, informando os símbolos existentes e seus comprimentos, tendo como resultado o mesmo código que foi utilizado pelo codificador e a árvore criada no processo. Assim, é possível dar início ao processo de decodificação do conteúdo do arquivo, sendo ele criado ou no diretório do arquivo lido ou em outro caso tenha sido informando, mantendo o mesmo nome porém sem a extensão '.bin'.

Para essa decodificação, foi utilizada a árvore criada durante a geração dos códigos, de forma a ser lido bit a bit o texto a ser decodificado, em que o um bit 0 significa ir para o nó filho esquerdo e o bit 1 ir ao novo filho direito, até ser

encontrado um nó folha, significando que os bits lidos fazem referência ao símbolo com o código desse nó; retornando para o nó raiz e repetindo o processo até chegar ao final do arquivo, desconsiderando o padding final de tamanho já conhecido.

II. Resultados

TABLE I
TAMANHOS DOS ARQUIVOS

Nome Arquivos	**Tamanho *Original (bytes)***	**Tamanho Comprimido (bytes) (%)***	
		Com cabeçalho	***Sem cabeçalho***
dom_casmurro.txt	389.670	228.121 (58,54%)	227.984 (58,51%)
fonte.txt	1.000.000	368.736 (36,87%)	368.693 (36,69%)
fonte0.txt	1.000	247 (24,70%)	209 (20,90%)
fonte1.txt	1.000.000	415.846 (41,58%)	415.793 (41,58%)
lena.bmp	263.290	248.099 (94,23%)	247.810 (94,12%)
TEncEntropy.txt	19.415	13.000 (66,96%)	12.882 (66,35%)
TEncSearch.txt	253.012	164.323 (64,95%)	164.196 (64,90%)

* 100 * Novo / Original

TABLE II
ENTROPIA DOS ARQUIVOS

Arquivos	**Entropia 1ª Ordem**	**Comprimento médio do código (%)***
dom_casmurro.txt	4,6430	4,6806 (0,81%)
fonte.txt	2,8940	2,9495 (1,92%)
fonte0.txt	1,5835	1,6680 (5,34%)
fonte1.txt	3,3033	3,3263 (0,70%)
lena.bmp	7,5097	7,5296 (0,26%)
TEncEntropy.txt	5,2719	5,3079 (0,68%)
TEncSearch.txt	5,1581	5,1917 (0,65%)

* (Comprimento médio / Entropia)*100 - 100

III. Comparações

TABLE III
DIFERENTES COMPRESSÕES

Nome Arquivos	**Percentuais resultantes da compressão(%)***			
	Implementado	***Deflate***	***RAR***	***LZMA2***
dom_casmurro.txt	58,54%	36,91%	34,82%	32,88%
fonte.txt	36,87%	41,03%	42,32%	40,15%
fonte0.txt	24,70%	32,00%	39,20%	46,00%
fonte1.txt	41,58%	46,81%	48,73%	45,91%
lena.bmp	94,23%	82,19%	70,46%	69,58%
TEncEntropy.txt	66,96%	21,51%	22,14%	21,54%
TEncSearch.txt	64,95%	14,84%	13,61%	12,93%

* 100 * Novo / Original

IV. Conclusão

Sobre o comportamento observado do algoritmo de Huffman puro (tabelas I e II), é possível observar que o algoritmo apresentou uma boa taxa de compressão (resultado menor que 50% do original) para os arquivos "fonte*.txt", criados para o contexto desse experimento, não representando uma situação de prática real, porém mesmo assim o comprimento médio do código de "fonte0.txt"foi alta comparada com a entropia calculada (>%2). Já para os outros arquivos, foi surpreendente a pequena compressão realizada em "lena.bmp"mesmo com um comprimento médio do código próximo da entropia calculada, comprimento esse que foi satisfatório para os outros arquivos restante também, porém com uma compressão melhor do que a da "lena.bmp".

Já com os resultados das comparações com algoritmos de compressão utilizados comercialmente (tabela III), é possível observar padrões, como uma melhor eficiência do nosso algoritmo nos arquivos teóricos ("fonte*.txt") e uma clara vantagem dos programas comerciais sobre o restante dos arquivos. Além disso, mesmo entre os arquivos em que os algoritmos comerciais foram melhores, àqueles que eram formados por palavras e padrões da língua inglesa (TEnc*.txt) foram comprimidos muito mais do que o texto em português ("dom_casmurro.txt") e a imagem ("lena.bmp").

Com isso, observando os pontos em que os algoritmos comerciais são bem melhores do que o de Huffman implementado, juntamente com os comprimentos médios dos códigos obtidos e uma perspectiva de utilização real; pode-se ter como estratégia, para melhorar um possível futuro algoritmo baseado no implementado, a consideração de uma dependência entre os símbolos existentes nos arquivos, visto que mesmo para arquivos em que o algoritmo implementado é significativamente pior do que os comerciais, o seu comprimento médio está muito próximo da entropia de 1ª ordem, indicando que para uma entropia de ordem maior, teríamos um menor valor, indicando assim uma relação entre os símbolos utilizados e possibilitando uma melhora no comprimento médio do código para uma valor menor do que a da entropia aqui calculada. Exemplos claros de padrões que poderiam ser utilizados para uma melhor compressão são a existência de palavras nas diferentes línguas, de forma a formarem sequências de símbolos que provavelmente se repetem no decorrer do arquivo("dom_casmurro.txt"e "TEnc*.txt"); e termos que claramente se repetem como regras de sintaxe de uma linguagem de programação ("TEnc*.txt").