Relatório Trabalho Prático TP1
Teoria de Informação - 2023/2024

Francisco Pereira - 2022217071
Francisco Caetano - 2022217054
João Vaz - 2022231087
Docente: Rui Pedro Pinto de Carvalho e Paiva

Ponto 1:

O método *getBlockFormatReader* desempenha a leitura do formato do bloco, conforme especificado no enunciado.

Este chama a função *readBits* para obter os valores de HLIST, HDIST e HCLEN, ou seja, os primeiros 5 bits destinado para o HLIST, os próximos 5 bits para o HDIST e os últimos 4 bits para o HCLEN.

Ponto 2:

A função *hclenParaCodeComp* tem como função gerar um array que armazena os comprimentos dos códigos do "alfabeto de comprimentos de códigos".

Este utiliza um array, *order*, que representa a ordem dos códigos a serem lidos. Inicialmente, é criado um array preenchido com zeros, *alphaCodeLen*, utilizando o numpy. Em seguida, um loop que percorre os primeiros *numCLCode*, elementos de *order* e associa a cada *posiçao* do array alphaCodeLen o valor lido a partir de 3 bits, utilizando *readBits*. O array resultante contém os comprimentos dos códigos do "alfabeto de comprimentos de códigos".

Ponto 3:

A função *alphaParaHuffman* desempenha a conversão dos comprimentos de códigos, contidos na lista *alphaCodeLen*, em códigos de Huffman. Esta gera uma tabela que associa cada comprimento ao seu código de Huffman correspondente.

Primeiramente, é criada uma lista, lenCounts, para armazenar a contagem da frequência de cada comprimento de código Cada índice dessa lista representa um comprimento, e os valores armazenados indicam quantas vezes esse comprimento aparece na lista original. Esta lista é inicializada com zeros de tamanho igual a alphaCodeLen. Percorre-se a lista alphaCodeLen e incrementa a contagem de cada comprimento em lenCounts.

Em seguida, são gerados os códigos de Huffman que irão ser armazenados na lista nxtCode.

Por fim, é criada uma lista de tuplos, codeTable, contendo o comprimento e o código correspondente.

Ponto 4, 5 e 6:

Os pontos 4,5 e 6 foram feitos todos na mesma função *litDistToHuffman*.
Esta função cria uma árvore de Huffman para os comprimentos de código codeLengthsTreeRoot = HuffmanTree(), constrói a árvore de Huffman usando a tabela de códigos fornecida.

De seguida, inicializa arrays para armazenar os comprimentos de código para códigos literais/comprimentos e códigos de distância e inicializa uma árvore de Huffman para percorrer os comprimentos de código.

O loop principal para compreender os comprimentos de códigos consiste em percorrer a árvore de Huffman com base no próximo bit lido, verificar se o nó atual na árvore representa um código válido, depois obter o índice do nó atual, assegurar que o índice está dentro do intervalo esperado [0, 18], interpretar o código e atualizar os arrays conforme necessário para 15,16 e 17 por fim retorna os arrays contendo os comprimentos de código para códigos literais/comprimentos e códigos de distância.

Ponto 7 e 8:

O ponto 7 e 8 estão na mesma função: *decodeHuffman*.
A primeira parte da função consiste em criar duas árvores de Huffman, uma para os códigos de literais e outra para os códigos da distância.
Após isso é executado o scope while que irá ler os bits se o curNode, ou seja, o nó atual da árvore for -1 passa-se para o seguinte e atualiza-se index para o index do curNode.
Caso esse index seja 256 significa que chegamos ao fim do bloco e saímos com break, caso o index seja menor que 256(else) damos append ao nosso array hist do index desse nó e caso o tamanho desse array ultrapasse histSize(32768) que é a distância máxima é removido do array a descodificação mais antiga.
no caso do index ser maior que 256, temos de acessar as tabelas que nos foram fornecidas no documento 2:

	Extra			Extra			Extra	
Code	Bits	Length(s)	Code	Bits	Lengths	Code	Bits	Length(s)
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

	Extra			Extra			Extra	
Code	Bits	Dist	Code	Bits	Dist	Code	Bits	Distance
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

Essas tabelas foram armazenadas em dois dicionários que foram declarados no ínicio do nosso código e através delas acessamos o numExtraBits e tam o que nos permite calcular através da função readBits a length e a distância que serão usadas para armazenar no array hist as descodificações.No final, criamos um objeto de bytes de hist (*bytesarray*) e escrevemos para o ficheiro.