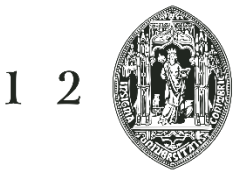

UNIVERSIDADE D COIMBRA

Relatório do Trabalho Prático Nº 2

Teoria da Informação

Feito por:

Filipe Rodrigues

Nº 2021228054

Joás Silva

Nº 2021226149

André Louro

Nº 2021232388

Relatório

Exercício 1:

Para devolver o valor correspondente a HLIT, HDIST e HCLEN, foi criado um método de nome ***readBlock*** que devolve os três valores pretendidos. Estes valores são obtidos através da soma dos bits lidos através da função ***readBits*** com um inteiro de forma a cumprir os valores mínimos correspondentes.

Exercício 2:

De forma a armazenar num array os comprimentos dos códigos do "alfabeto de comprimentos de códigos", com base em HCLEN, foi criado um método, com o nome ***readLengths***, que, tendo em atenção a ordem indicada no enunciado, armazena os comprimentos obtidos através do HCLEN num array, sendo este devolvido no final do método.

Exercício 3:

Utilizando o código da página 34 do Doc1, foi criada a função ***ex3*** que possui como parâmetro o array ***lengths*** com os comprimentos do código de Huffman da árvore a ser criada. Esta função devolve o array ***code*** que contém os códigos de Huffman de cada um dos símbolos (code[**i**] = código de Huffman do símbolo **i**).

Também foi criada a função ***toBinary*** que converte um número inteiro em uma string do número em binário (com a designado comprimento (***length***)).

Neste exercício foi ainda criada a lista ***h*** utilizando a função do exercício 1 que contém HLIT, HDIST e HCLEN. É então chamada a função **readLengths** criada no exercício anterior com a lista criada, guardando o resultado na variável ***lengths*** (comprimentos dos códigos do "alfabeto de comprimentos de códigos"). Após isso é chamada a função ***ex3*** com *lengths*, sendo o resultado guardado na variável ***tree***.

Por último, é criada a árvore de Huffman do alfabeto de comprimentos de códigos, sendo guardada na variável ***hft***.

Exercício 4:

Para armazenar num array os HLIT + 257 comprimentos dos códigos do alfabeto de literais/comprimentos, percorremos a árvore de Huffman do alfabeto de comprimentos de códigos (criada no exercício anterior) *hft* bit a bit (lidos do ficheiro) e caso o valor obtido da árvore esteja entre 0 e 15, o valor é adicionado ao array de comprimentos de códigos, caso o valor lido seja entre 16 e 18, terão de ser lidos bits extra:

- Caso seja 16:
    - Serão lidos mais 2 bits do ficheiro e depois é somado 3 ao valor dos bits lidos. O número obtido será a quantidade de vezes que o número anterior do array será copiado e adicionado.
- Caso seja 17:
    - Serão lidos mais 3 bits do ficheiro e depois é somado 3 ao valor dos bits lidos. O número obtido será a quantidade de vezes que o número "0" será adicionado ao array.
- Caso seja 18:
    - Parecido ao caso do "17", sendo que serão lidos 7 bits ao invés de 3 bits, e soma-se 11 ao invés de 3. O número obtido é o número de "0"s a serem adicionados ao array.

Exercício 5:

De modo a ler e armazenar num array os HDIST + 1 comprimentos de código referentes ao alfabeto de literais/comprimentos, codificados segundo o código de Huffman de comprimentos de códigos, foi criado um método, com o nome ***ex5***, que recorrendo às funções do ficheiro huffmantree.py e tomando atenção aos códigos que requerem a leitura de extra bits, vai devolver um array com os valores pretendidos.

Exercício 6:

Para determinar os códigos de Huffman referentes aos dois alfabetos (literais/comprimentos e distâncias) e armazená-los num array, foi utilizado o método do exercício 3 (***ex3***) para o array devolvido em ***ex4*** e em ***ex5***.

Exercício 7:

Para poder descompactar os dados comprimidos, primeiro foram criadas as duas árvores de Huffman, a árvore de HLIT e de HDIST, percorrendo os seus array de comprimento de código e caso seja maior que 0, adiciona-se o node com o código de Huffman correspondente ao símbolo (criado no exercício anterior).

Depois é utilizada a função ***descomprimir*** que leva como parâmetros as árvores de Huffman de HLIT e HDIST. Nela é percorrida a árvore de HLIT bit a bit (lidos do ficheiro) e caso o número obtido da folha seja menor que 256, é adicionado ao array do ficheiro descomprimido, caso o valor esteja entre 257 e 284 (sendo 285 um caso especial com comprimento = 258), significa que o valor a ser lido é um comprimento, então calcula-se o valor de bits extra (***extraBits***) a serem lidos: ((posição - 265)//4 + 1), e o comprimento é obtido através da fórmula:

$$2^{extraBits}(pos - (261 + 4extraBits)) + (2^{extraBits+2} + 3) + readBits(extraBits)$$

A fórmula foi obtida após fatorizar a fórmula dos casos obtidos com *n* bits do Doc2, sendo *pos* o valor lido do ficheiro entre 257 e 284 e *extraBits* o nº de bits extra calculado anteriormente.

Após isso, a árvore HDIST é percorrida bit a bit (lidos do ficheiro), até encontrar uma folha da árvore, guardando o símbolo da folha na variável ***pos2***. Caso pos2 esteja entre 0 e 3, a distância (***dist***) será *pos2* + 1, caso esteja entre 4 e 29 vão ter de ser lidos bits extra, utilizando a fórmula: (*pos2* // 2) -1 sendo este número guardado na variável ***aux2***. Depois, a distância é calculada com a seguinte fómula:

$$2^{axu2}(pos2 - (4 + 2^{aux2-1})) + (2^{(aux2\ +\ 1)} + 1) + readBits(aux2)$$

Como a fórmula anterior, esta foi obtida ao fatorizar os casos com *n* bits do Doc2, sendo *pos2* o valor lido entre 4 e 29 e *aux2* o número de bits extra a serem lidos do ficheiro.

Depois de obter o valor da distância e do comprimento, é utilizado o algoritmo LZ77, percorrendo no array ***dist*** casas para trás e copiando ***length*** dígitos à frente.

A função ***descomprimir*** acaba quando o número lido da árvore de Huffman de HLIT for igual a 256, devolvendo então o array dos símbolos descodificados.

Exercício 8:

Para poder escrever o resultado do programa em um ficheiro, foi criada a variável **output** para criar um ficheiro com o nome original (sem o .gzip) e nele foi escrito em binário o array obtido no exercício anterior convertido em binário, utilizando a função *bytes()*.