Relatório
Trabalho Prático nº 2
Descompactação de Ficheiros ‘gzip’

Departamento de Engenharia Informática

Disciplina: Teoria da Informação

Trabalho realizado por:

Diogo Honório, 2021232043

Mateus Queiroz, 2021214102

Gabriel Ferreira, 2021242097

UNIVERSIDADE D
COIMBRA

Índice:

Introdução:

Neste trabalho, pretende-se implementar o descodificador do algoritmo deflate (usado em ficheiros gzip). Em específico e dando ênfase na descompactação de blocos comprimidos com códigos de Huffman dinâmicos.

Exercício 1:

Este exercício tinha como objetivo a leitura de um bloco devolvendo assim os valores correspondentes a HLIT, HDIST e a HCLEN. Para isso, tive-mos de recorrer à função já fornecida 'readBits' lendo assim os bits necessários para cada um dos blocos, ou seja, 5 bits para HLIT, 5 bits para o HDIST e 4 bits para o HCLEN.

Exercício 2:

Para resolver este exercício, criamos um *array* com a ordem pela qual queremos que seja lida as sequências de 3 bits. Em seguida, armazenamos num outro *array* 'clens' as leituras das sequências de 3 bits, estas irão corresponder aos comprimentos dos códigos com base em HCLEN.

Exercício 3:

Neste exercício, começamos por implementar o código fornecido slide 34 do Doc1., este código tem como função guardar os valores que se encontram nas folhas da árvore de Huffman, guardando-os num array *cujo* nome é 'tree_code'.

```
clens = list(clens)

bl_count = np.zeros(HCLEN, dtype=np.int16) #criac
for i in range(len(clens)):
    bl_count[clens[i]] += 1

code = 0 #codigo base
bl_count[0]=0
next_code = np.zeros(HCLEN, dtype=np.int16)
for i in range(1,HCLEN):
    code = (code + bl_count[i-1]) << 1
    next_code[i] = code


tree_code = np.zeros(len(clens), dtype=np.int16)
for i in range(len(clens)):
    length = clens[i]
    if(length != 0):
        tree_code[i] = next_code[length]
        next_code[length] += 1
```

Fig.1 - Implementação do código

```
Code – código base
bl_count [i] – número de símbolos a serem codificados com i bits

code = 0;
bl_count[0] = 0;
for (bits = 1; bits <= MAX_BITS; bits++) {
    code = (code + bl_count[bits-1]) << 1;
    next_code[bits] = code;
}


for (n = 0;  n <= max_code; n++) {
      len = tree[n].Len;
      if (len != 0) {
         tree[n].Code = next_code[len];
         next_code[len]++;
      }
  }
```

Fig.2 - Código do slide 34 do Doc1.

Assim que os valores forem todos guardados no *array* 'tree_code', iremos percorrer esses valores encontrando 1's e 0's. Para isso analisamos o bit menos significativo, obtemo-lo e adicionamo-lo na lista 'huff', de seguida, dividimos o valor por 2 de modo que o bit menos significativo passe a ser o bit seguinte e repetimos todo este processo até que o valor seja menor que 1. Assim conseguimos obter o código de Huffman do valor que estamos a analisar.

De notar também que no caso de o comprimento indicado em 'clens' ser maior que a *len( )* do código de Huffman, então, inserimos 0's à esquerda de modo que ambos os valores sejam iguais.

Por fim, é criada uma árvore de Huffman através da função *HuffmanTree()* onde são adicionados os códigos de Huffman obtidos anteriormente através da função *addNode()*.

Exercício 4 e 5:

Nestas etapas, tínhamos como objetivo criar uma função que fosse capaz de devolver os códigos de literais/comprimentos e os códigos das distâncias. A função que criamos devolve ambos os alfabetos, apenas sendo necessário a troca do parâmetro de entrada 'comp_vet' para o valor de HLIT no caso de se desejar os códigos de literais/comprimentos, ou trocar para o valor de HDIST no caso de se desejar o código das distâncias.

Para conseguirmos obter estes códigos tivemos de ler bit a bit da árvore que criamos no exercício 3 e verificar o seu valor:

- No caso de ser entre 0 e 15 o seu valor irá ser representado do mesmo modo.
- No caso de o valor ser igual a 16 então devemos copiar tamanho do código anterior 3 a 6 vezes dependendo dos 2 bits seguintes.
- No caso de o valor ser igual a 17 então devemos copiar tamanho em 0's entre 3 a 10 vezes dependendo dos 3 bits seguintes.
- No caso de o valor ser iguala a 18 então devemos copiar tamanho em 0's entre 11 a 138 vezes dependendo dos 7 bits seguintes.

Assim recorrendo a este critério os bits irão ser adicionados ao *array* 'Hlen' que irá possuir o tamanho de HLIT ou HDIST conforme o parâmetro inserido na função.

Exercício 6:

Neste exercício, tínhamos como objetivo criar os códigos de Huffman dos dois alfabetos desenvolvidos no exercício 4 e no exercício 5 (literais/comprimentos e distâncias). Então apenas usamos a função desenvolvida no exercício 3, no entanto, apenas trocando o parâmetro de entrada 'clens' pelo alfabeto literais/comprimentos e pelo alfabeto das distâncias.

Exercício 7:

Nesta etapa, tínhamos como objetivo descompactar os dados comprimidos através dos códigos de Huffman desenvolvidos do exercício 6 e no algoritmo L277. Para isso, criamos uma função que primeiro obtém o valor da árvore de literais/comprimentos e depois verifica se o valor é menor que 256, caso isso aconteça, o valor pode ser copiado de imediato para lista 'output' (lista onde serão guardados todos os valores descodificados).

No caso, de o valor ser igual a 256 significa que nos encontramos no fim do bloco logo é feito um *break* de modo a parar de executar a função.

Caso o valor lido se encontre entre 257 e 285 então vamos descodificar o comprimento através da função 'comp_decode', nesta função descodificamos o valor do seguinte modo:

- Caso o valor seja menor que 265 então o comprimento é igual ao seu valor menos 254.
- Caso o valor seja 285, o seu comprimento será 258.
- Caso o valor seja maior ou igual a 265 e menor ou igual a 284 então significa que possui um determinado número bits extra, estes bits irão fazer com que os valores do comprimento variem de acordo com a tabela a seguir apresentada:

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			

Fig.3 – valores possíveis de comprimento conforme a variação dos bits extra.

Após o valor do comprimento ter sido descodificado, no caso desse valor ter se encontrado entre 257 e 285, então devemos também descodificar a distância, para isso obtemos o valor da árvore das distâncias e através da função 'dist_decode', descodificamos a distância da seguinte maneira:

- Caso o valor seja menor que 4 então a distância é igual ao valor mais 1.
- Caso o valor seja maior ou igual a 4 e menor ou igual a 29 então significa que possui um determinado número de bits extra, estes bits irão fazer com que os valores da distância variem de acordo com a seguinte tabela:

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

Fig.4 – valores possíveis de distância conforme a variação dos bits extra

A descodificação do comprimento e da distância é usada posteriormente para recuar no 'output' o valor da distância e copiar o valor do comprimento a partir da posição até onde a distância o fez recuar, inserindo depois esse valor no 'output'.

Exercício 8:

Neste exercício, é efetuada a abertura de um ficheiro do tipo escrita e são colocados todos os valores que descodificamos anteriormente no exercício 7. No entanto, é necessário ter em atenção por os valores a *char* de modo que o resultado no ficheiro seja texto legível.

Conclusão:

Com a realização deste trabalho, conseguimos compreender melhor como trabalhar com árvores de Huffman e como descompactar um ficheiro ‘gzip’. Este trabalho de início pareceu-nos bastante fácil e intuitivo, porem, exercícios como o 4 e o 7 causaram-nos alguns problemas, no entanto, achamos um trabalho interessante no geral.