FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Código para descompressão de ficheiros com base no algoritmo LZ77

Teoria da Informação

Saulo José Piccirilo Mendes
Marco Simões
18 de dezembro de 2022

1. Introdução.....1
2. Estrutura do trabalho.....2
3. Conclusão.....6

1.Introdução

O presente relatório possui o objetivo de explicar o funcionamento do algoritmo de *deflate* desenvolvido para a cadeira de Teoria da Informação, do ano letivo 22/23. O algoritmo baseia-se no método de compressão L77, publicado em 1977 pelos académicos Abraham Lempel e Jacob. Vale ressaltar que o algoritmo, escrito em Python, recorre também ao código de Huffman para interpretar os dados.

2.Estrutura do trabalho

O trabalho foi construído através dos ficheiro gzip.py e huffmantree.py, previamente disponibilizados. Foram adicionados métodos que permitem a total descompressão do ficheiro de maneira eficiente e sem perdas de informação, sendo eles:

2.1- getFinalFileName(self)

Método responsável por verificar se já existe um ficheiro com o mesmo nome. Caso exista, acrescenta-se um número ao nome original até que se encontre um nome disponível.

2.2- getConstants(self)

Método responsável por extrair de cada bloco os respetivos valores das constantes H_LIT,H_DIST, H_CLEN

2.3- getCodeLEN(self, symbol_order, search_limit)

Método responsável por obter os comprimentos individuais de cada símbolo de acordo com uma ordem. No caso deste trabalho, utilizou-se a ordem "symbol_order = [16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15]"

2.4- getLengths(self, CODE_tree, search_limit)

Método responsável por obter os comprimentos dos literais consoante uma árvore de Huffman dos comprimentos individuais dos símbolos. Vale ressaltar que o método leva em conta as exceções do algoritmo de compressão LZ77, ou seja, possui interpretações características deste algoritmo para os número 16,17,18. No caso do número ser:

- 16: repete-se o símbolo anterior por 3 + (valor dos dois bits extras) vezes.
- 17: repete-se o 0 por 3 + (valor dos três bits extras) vezes.
- 17: repete-se o 0 por 11 + (valor dos sete bits extras) vezes.

2.5- getValues(self, length_array)

Método responsável por atribuir valores diferentes a cada símbolo consoante o seu comprimento. Deste modo, símbolos com menores comprimentos são atribuídos valores menores.

2.6- createHuffmanTree(self, lengths_array, values_array)

Método responsável por construir uma árvore de Huffman de acordo com duas determinadas listas de comprimentos e valores. Os valores são convertidos em números binários que representam a posição (folha) de cada símbolo na árvore.

2.6- createAlphabet(self, codigo, values_range, increase_index)

Método responsável por construir um alfabeto de descompressão para os literais (que é o mesmo para os comprimentos), e para as distâncias. Recebe um array que contém os códigos, um array com o primeiro intervalo de valores, e um array com os índices onde aumentam o número de bits extras necessários para obter o valor desejado.

2.7- decodeDist(self, HDIST_tree, dist_alphabet)

Método responsável por determinar a distância que se deve recuar para copiar um determinado comprimento do buffer de output e introduzi-lo outra vez. Baseia-se na árvore das distância e o alfabeto de descompressão para as distância previamente construídos.

2.8- decodeLength(self, val, length_alphabet)

Método responsável por interpretar o comprimento do código que vai ser introduzido outra vez. Baseia-se na árvore dos literais e o alfabeto de descompressão para as distância previamente construídos.

2.9- getData(self, HLIT_tree, HDIST_tree, length_alphabet, dist_alphabet)

Método responsável por ler os bytes do ficheiro comprimido, e interpretá-los como ou literais, ou comprimentos, ou distâncias, consoante as regras do algoritmo de compressão LZ77. Caso o valor seja:

- Menor que 256: é interpretado como um literal
- Igual a 256: é interpretado como o fim do bloco
- Maior que 256: é interpretado como um comprimento, e depois obtém-se a distância

Conclusão

O presente trabalho levou à necessidade de tomada de decisão perante diversos cenários de usabilidade, integridade de dados e organização do projeto. Foi preciso que cada alfabeto e árvore fosse construídos perfeitamente, caso contrário, qualquer erro mínimo poderia levar a uma leitura a mais de bits e a ultrapassar o limite do ficheiro comprimido. Deste modo, foi um projeto enriquecedor do ponto de vista teórico, porque para implementar o processo de deflate foi preciso primeiramente dominar o funcionamento do algoritmo de compressão LZ77 e a teoria do código de Huffman