

# Teoria da Informação

## Trabalho Prático nº 2

### Descompactação de Ficheiros 'gzip'



# Introdução

Período de execução: 6 aulas práticas

## Ritmo de execução esperado para avaliação:

- Semana 1: 1 a 3
- Semana 2: 4
- Semana 3: 5
- Semana 4: 6,7
- Semana 5: 8
- Semana 6: correções finais

Prazo de Entrega: **sábado, 16 de dezembro de 2023**

Esforço extra-aulas previsto: 30 h / aluno

Linguagem de Programação: Python

Objectivo: Pretende-se que o aluno adquira sensibilidade para as questões fundamentais relacionadas com codificação usando árvores de Huffman e dicionários LZ77.

## Trabalho Prático

Neste trabalho, pretende-se implementar o descodificador do algoritmo *deflate* (usado em ficheiros gzip). Em particular, será objectivo levar a cabo a descompactação de blocos comprimidos com **códigos de Huffman dinâmicos**. Todas as restantes situações de descodificação deverão ser ignoradas no âmbito do presente trabalho.

## A. Preparação

1. Leitura dos documentos de apoio ao trabalho prático:
  - **(Doc1)**: Slides fornecidos nas aulas teórico-práticas.
  - **(Doc2)**: Request for Comment (RFC) do deflate.
  - **(Doc3)**: RFC do cabeçalho do gzip.
  - **(Doc4)**: ficheiro *byteStream.txt* → ficheiro com a sequência de bytes após o cabeçalho do gzip (para o exemplo fornecido, *FAQ.txt.gz*).
  - **(Doc5)**: ficheiro *Códigos.xls* → resultados esperados para os códigos de Huffman a obter nas várias etapas do algoritmo, para o exemplo *FAQ.txt.gz*
2. O seguinte código fonte é-lhe fornecido como base de trabalho. Poderá utilizá-lo, caso considere pertinente. Nesse caso, deverá estudar as funcionalidades implementadas.
  - a) Ficheiro **gzip.py**: classe principal para descompactação de um ficheiro no formato gzip:
    - o Linha de comando: `gzip <nome.gz>`
    - o Classes principais
      - **GZIPHeader**: class relativa à leitura do cabeçalho do ficheiro .gz
      - **GZIP**: class relativa à descompactação do ficheiro .gz, lendo primeiro o header e depois passando para a descompressão bloco a bloco (parte a implementar pelos estudantes).
  - b) Ficheiro **huffmantree.py**: contém a class `HuffmanTree` com um conjunto de funções para criação, acesso e gestão de árvores de Huffman:

Observação: o ficheiro auxiliar **testhuffmantree.py**(ver abaixo) contém alguns exemplos de utilização das funções para

manipulação de árvores de Huffman (inserção de um dado código na árvore, pesquisa de um código na árvore, ...); tal como se referiu, o ficheiro `testhuffmantree.py` é um ficheiro auxiliar, de modo que não deverá ser incluído no projecto;

- Classes principais:

- **HFNode:** contém informação relativa a um nó da árvore de Huffman

- Campos da class

- *index*: guarda posição do nó no alfabeto, caso seja folha; -1 caso contrário;
      - *level*: nível na árvore em que se encontra o nó actual;
      - *left, right*: referências para os filhos esquerdo e direito do nó actual;

- **HuffmanTree:** define uma árvore de Huffman

- Campos da class

- *root (HFNode)*: raiz da árvore;
      - *curNode (HFNode)*: nó actual da árvore;

- Funções principais:

- *addNode(self, s, ind, verbose=False)*: adiciona nó à árvore:

- recebe uma string (*s*) com o código (sequência de 0s e 1s), o índice no alfabeto (*ind*) e um campo 'verbose' para visualização ou não de resultados;
    - devolve -1 se o nó já existe; -2 se a inserção implicasse que o código deixasse de ser de prefixo; 0 se adicionou bem;

- *nextNode(self, dir)*: actualiza o nó corrente na árvore com base no nó actual (*curNode* da árvore *hft*) e no próximo bit (*dir*):

- recebe um caracter *dir* com valor '0' ou '1';

- devolve -1 se não encontrou o nó; -2 se encontrou mas não é folha; o índice no alfabeto se é folha.
- **É esta a função a utilizar na pesquisa bit a bit, tal como decorre da leitura de bits referentes a códigos de Huffman**
- *findNode(self, s, cur=None, verbose=False)*: procura código na árvore, a partir de um nó especificado:
  - recebe uma string *s* com o código (sequência de 0s e 1s), o nó a partir do qual a pesquisa deve ser efectuada (se não especificado, começa da raiz) e um campo 'verbose'
  - devolve -1 se não encontrou; -2 se é prefixo de código existente; índice no alfabeto se encontrou;

Nota: esta função procura um código completo (e não de forma iterativa), pelo que, na prática, não deverão ser utilizadas;

- *resetCurNode (self)*: reposiciona *curNode* na raiz da árvore.

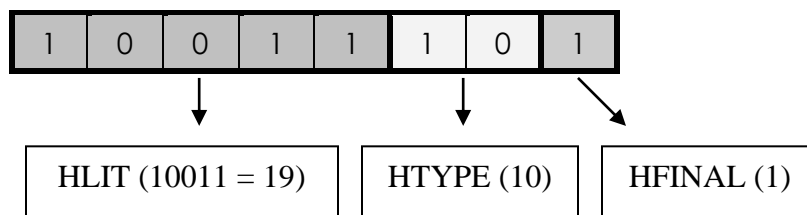
- c) Ficheiro **testhuffmantree.py**: contém exemplos de utilização de árvores de Huffman:
- Apenas contém um script com alguns exemplos.

## B. Implementação do descompactador:

Notas:

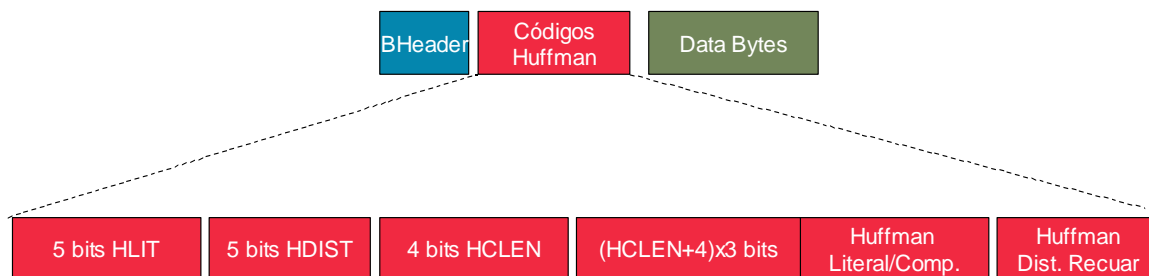
- As alíneas seguintes são apenas sugestões de implementação. Poderá seguir outra estratégia que considere mais adequada.
- Todas as funções desenvolvidas, assim como partes do código particularmente complexas, deverão estar **comentadas** de forma compreensível.

- É apresentada uma proposta de planeamento temporal, ao longo das 6s semanas do projecto.
- **Importante! Ordem dos bits nos bytes:**
  - o No *deflate*, a sequência de bits correspondente a códigos de Huffman é ordenada no byte começando com o bit *mais* significativo.
    - Exemplo: byte '**0110**1011', em que os bits em negrito correspondem a um código de Huffman:
      - Atendendo à regra acima, e lendo bit a bit, o código será 10110 (i.e., pela ordem inversa).
  - o Em elementos que não sejam códigos de Huffman, os bits são ordenados no byte começando com o bit *menos* significativo.
    - Exemplo: determinação dos comprimentos dos códigos do alfabeto de comprimentos de códigos, em que um dado byte tem a informação '**001000**10' (os bits em negrito denotam os 3 bits do comprimento a ler);
    - Com base nesta regra, o comprimento será 100 = 4 bits.
  - o Num byte que contenha vários elementos, os mesmos são armazenados da "direita para esquerda no byte"
    - Exemplo: block header



## 1ª Semana

1. Crie um método que leia o formato do bloco (i.e., devolva o valor correspondente a HLIT, HDIST e HCLEN), de acordo com a estrutura de cada bloco, apresentada na figura seguinte:



2. Crie um método que armazene num array os comprimentos dos códigos do "alfabeto de comprimentos de códigos", com base em HCLEN:
  - Tenha em atenção que as sequências de 3 bits a ler correspondem à ordem 16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15 no array de códigos (ver Doc1, Doc2; resultados a obter: Doc5)

## **2ª Semana**

3. Crie um método que converta os comprimentos dos códigos da alínea anterior em códigos de Huffman do "alfabeto de comprimentos de códigos" (ver Doc5);

## **3ª Semana**

4. Crie um método que leia e armazene num array os HLIT + 257 comprimentos dos códigos referentes ao alfabeto de literais/comprimentos, codificados segundo o código de Huffman de comprimentos de códigos (ver Doc5):
  - Recorra às funções do ficheiro huffmantree.py, nomeadamente às funções de pesquisa de códigos de forma sequencial (i.e., bit a bit);

- Tenha em atenção que alguns códigos requerem a leitura de alguns bits extra (nomeadamente os índices 16, 17 e 18 do alfabeto);

#### **4ª Semana**

5. Crie um método que leia e armazene num array os HDIST + 1 comprimentos de código referentes ao alfabeto de literais/comprimentos, codificados segundo o código de Huffman de comprimentos de códigos (ver Doc5):
  - Recorra às funções do ficheiro huffmantree.py, nomeadamente às funções de pesquisa de códigos de forma sequencial (i.e., bit a bit);
  - Tenha em atenção que alguns códigos requerem a leitura de alguns bits extra (nomeadamente os índices 16, 17 e 18 do alfabeto);

#### **5ª Semana**

6. Usando o método do ponto 3), determine os códigos de Huffman referentes aos dois alfabetos (literais / comprimentos e distâncias) e armazene-os num array (ver Doc5).
7. Crie as funções necessárias à descompactação dos dados comprimidos, com base nos códigos de Huffman e no algoritmo LZ77 (ver Doc1 e Doc2).
  - Recorra funções do ficheiro huffmantree.py, nomeadamente às funções de pesquisa de códigos de forma sequencial (i.e., bit a bit);
  - Tenha em atenção que alguns códigos requerem a leitura de alguns bits extra (por exemplo os índices 265 a 285 no alfabeto de literais/comprimentos ou os índices 4 a 29 no alfabeto de distâncias);

#### **6ª Semana**



8. Grave os dados descompactados num ficheiro com o nome original (consulte a estrutura *gzipHeader*, nomeadamente o campo *fName* e analize a função *getHeader* do ficheiro *gzip.cpp*).