Projeto 3

PZYP: Compressão LZSS

Índice

Índice de Ilustrações

1. Introdução e Objetivos

No seguimento da formação em Técnico de Programação, como trabalho final da UFCD 10794- Programação Avançada em Python, foi-nos proposto realizar deste projeto cujo objetivo passa por aplicar os conhecimentos adquiridos no desenvolvimento de aplicações úteis para a manutenção e administração de sistemas.

Com este projeto, pretendeu-se desenvolver uma aplicação para **compressão** e **descompressão de ficheiros**, utilizando um código modular, organizado em funções, com tipos de dados apropriados, recorrendo a algumas bibliotecas e tendo como base o algoritmo LZSS, o sucessor do mais conhecido algoritmo de compressão, o LZ77.

Para que seja possível entender qual o modo de funcionamento da aplicação, procurou-se saber em que consistia o algoritmo LZSS e um pouco a sua história.

O algoritmo LZ77 realiza a compressão através da substituição de uma sequência repetida de bytes por uma referência, que indica a posição original da repetição (**distância** ou **offset**) e a sua dimensão (**comprimento** ou **length**). Ou seja, este algoritmo possui uma janela (buffer) que vai analisando e guardando os bytes processados e, sempre que é lida uma cópia de uma sequência de bytes já existente na janela, o LZ77 substitui essa sequência por uma referência <distância, comprimento>.

A janela possui um tamanho fixo, de modo que, quando os dados a analisar são superiores ao tamanho máximo da janela, ela precisa "deslizar" sobre os dados. Assim, quando o tamanho máximo da janela é atingido, sempre que uma sequência de bytes é adicionada à janela, é removida a mesma quantidade de bytes do lado dos bytes processados mais antigos, criando assim o efeito deslizante. Por este motivo a janela é muitas vezes designada de **janela deslizante**. No processo de descompressão, a janela é também recriada para deslizar pelo ficheiro e utilizada para interpretar as referências que for encontrando.

No entanto, o tamanho da janela pode variar. Em teoria, quanto maior a janela maior a taxa de conversão, ou seja, quanto maior a janela mais bytes podem ser armazenados e, por conseguinte, mais referências podem ser geradas. Porém, isto também torna o processo mais lento uma vez que terá que efetuar pesquisas mais demoradas. Outra consequência de utilizar uma janela maior é a necessidade de ter disponível um maior número de bits para representar quer a distância e quer o comprimento, sendo que tipicamente utilizam-se menos bits para o comprimento. Pode acontecer, no entanto, encontrarmos uma referência com um comprimento maior que a distância. Isto significa que, a sequência repetida depende de bytes que ainda não foram colocados na janela e representa uma **repetição consecutiva** de uma mesma sequência.

Com a compressão, o que se pretende é que o ficheiro comprimido seja mais pequeno que o original e para tal, a referência deve ser menor que a sequência que se pretende substituir. Com o LZ77 isto nem sempre acontece porque este algoritmo substitui cópias de qualquer dimensão por referências. Ou seja, o compressor cria uma referência <distância, comprimento> para uma sequência repetida, mas também cria uma referência com distancia e comprimento 0 para os bytes não comprimidos (**literais**), o que por vezes torna o ficheiro comprimido maior que o original.

Para minimizar estes problemas do LZ77 e torna-lo mais eficiente, James Storer e Thomas Szymanski desenvolveram o algoritmo LZSS, que continua a utilizar o conceito do LZ77 de criação de referências <distância, comprimento> para sequências repetidas, permitindo o descompressor identificar a localização da sequência original, mas, não cria referências de repetições com qualquer tamanho, definindo um **break even point**, e prefixa com uma **flag de compressão** os bytes comprimidos (ou referências) com um bit a 1 e os bytes literais com um bit a 0.

Ao definir um break even point, isto é, um limite mínimo de comprimento de uma repetição passível de ser comprimida, o LZSS só gera referências para repetições cujo comprimento seja superior ao valor definido, isto é, se o break even point for 2 o LZSS procura apenas repetições de sequências com pelo menos 3 bytes, nunca sendo então geradas referências para sequências com 0, 1 e 2 bytes. Como o tamanho máximo de uma sequência é de 15 bytes, o compressor pode utilizar estes valores para representar comprimentos até 18 bytes, se subtrair 3 ao comprimento da repetição. Já o descompressor necessita somar 3 ao comprimento de cada referência antes de recriar a sequência original.

Com a flag de compressão, o descompressor processa o ficheiro de entrada dependendo do valor do primeiro bit, ou seja, se o primeiro bit estiver a 0, ele lê o byte seguinte e coloca-o diretamente no ficheiro de saída e no fim da janela. Por outro lado, se o primeiro bit estiver a 1, ele lê os 2 bytes seguintes que devem indicar uma referência para uma sequência já lida, utilizando-a para localizar a sequência original na janela, e, a partir desta, recriar a cópia que é depois colocada na janela e no ficheiro de saída.

Com isto, o LZSS consegue resumir sequências maiores e aumentar, ainda que ligeiramente, a taxa de compressão.

Feita esta análise e para uma melhor orientação na criação desta aplicação, elaborou-se um fluxograma com o respetivo algoritmo de compressão e outro com o algoritmo de descompressão.

A elaboração deste projeto terminou com a criação da interface gráfica da aplicação, a partir da framework PyQT. Aqui, é possível anexar um documento proveniente do explorador de ficheiros, escolher a opção de comprimir ou descomprimir documento, selecionar o nível de compressão e por fim, indicar se pretende obter o ficheiro final encriptado e apontar o endereço para o guardar.

2. Desenhos e Estrutura

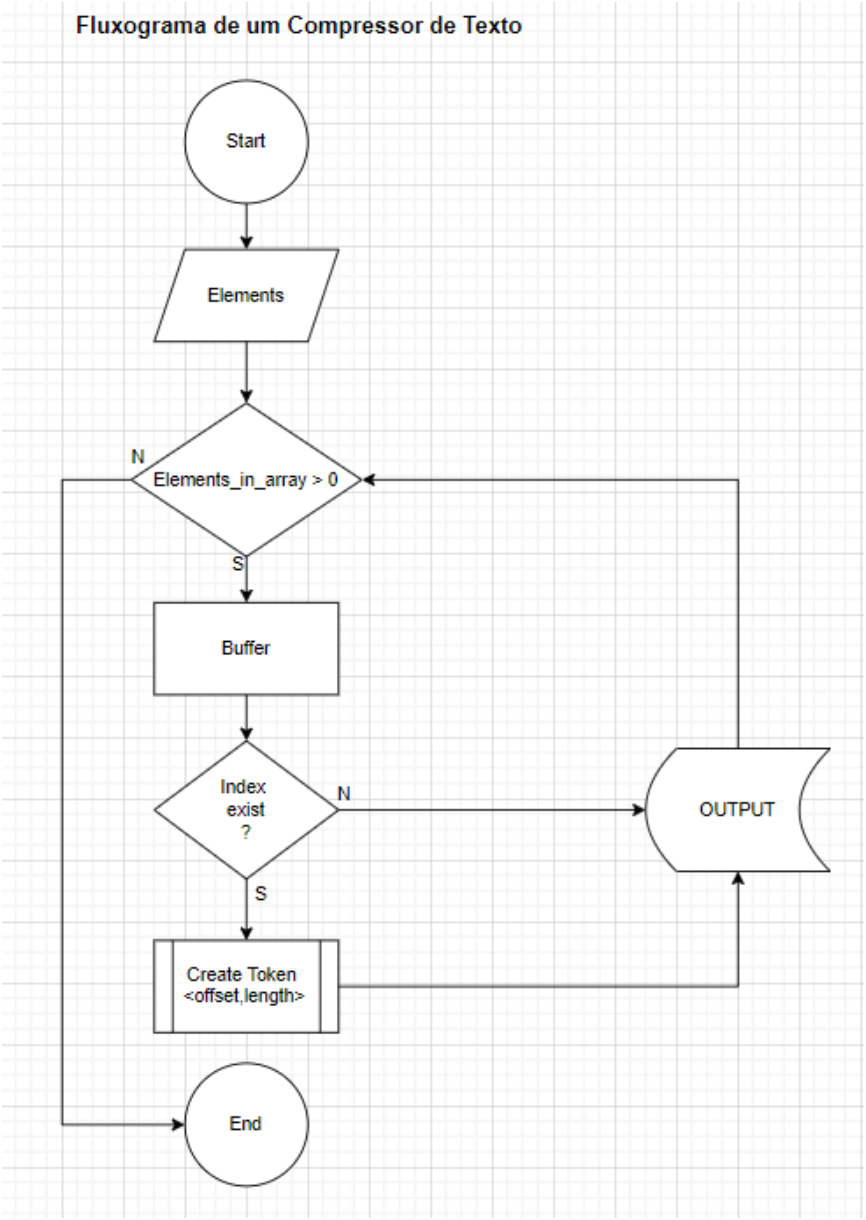

Figura 1- Fluxograma do Compressor de texto PZYP.

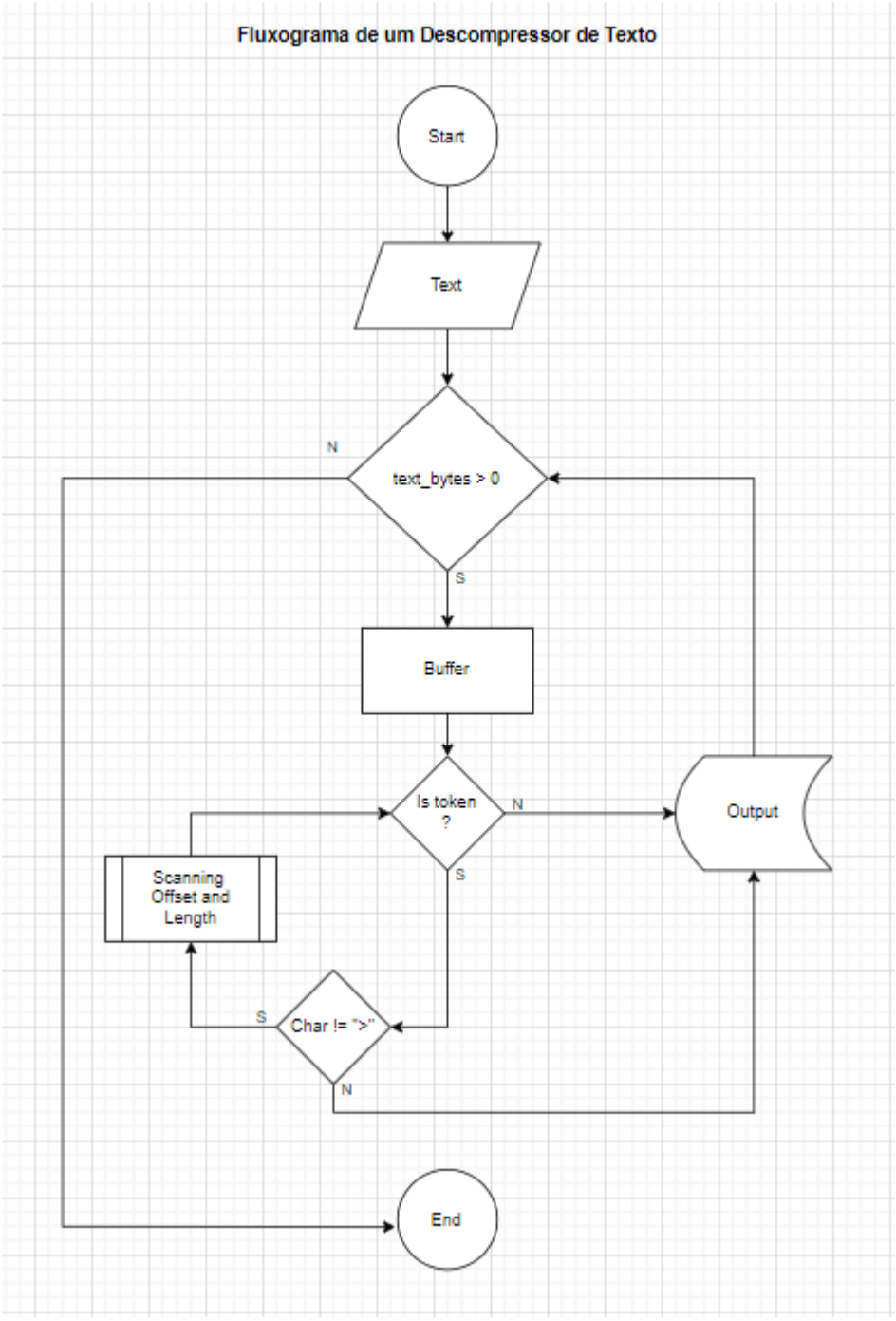

Figura 2- Fluxograma do Descompressor de texto PZYP.

3. Implementação

Para a concretização deste projeto, utilizou-se o editor de código **Visual Studio Code**, versão 1.62, num computador com Sistema Operativo Windows para criar e testar o algoritmo LZYP. No que se refere á componente da criação da Interface gráfica, optou-se por utilizar o programa **Qt Designer**, uma ferramenta que permite compor e personalizar as suas janelas ou diálogos (conhecidos por Qt Widgets) e testá-los usando diferentes estilos e resoluções, e também permite integrar com código gerado, desde que adaptado ao Qt, para que se possa obter o comportamento desejado dos elementos gráficos.

A sua implementação foi realizada em 3 fases.

Na **primeira fase**, começou-se por esquematizar o comportamento do compressor e descompressor de texto, conforme o pretendido (figuras 1 e 2, respetivamente). Depois, começou-se a elaborar o código de modo que o programa conseguisse ler e comprimir um documento de texto. Nesta fase, recorreu-se à importação das bibliotecas sys e os.path, para que pudéssemos invocar o programa na linha de comandos por inspeção direta do sys.argv e para manipulação dos ficheiros com informação comprimida e/ou descomprimida, respetivamente. O programa que corresponde a esta fase, deve ser compilado num terminal ou linha de comandos e invocado com o comando:

$ python pzyp.py -c [-d] File.

Feita a confirmação de que a compressão e descompressão estava a ser realizada corretamente, avançou-se para a **segunda fase**. Aqui, a intensão era que o processo de compressão e descompressão fosse realizado tendo em conta vários pontos:

1. Que pudesse ser utilizado ficheiros de qualquer tipo;
2. Que pudesse ser escolhido um nível de compressão;
3. Que nos ficheiros LZS aparecesse um cabeçalho com resumo dos dados;
4. Que houvesse a possibilidade de encriptar o ficheiro comprimido.

Assumindo todos estes requisitos, na linha de comandos, o programa passa a ser invocado com o comando:

$ python pzyp.py -c [-d] -l [level] -s -h -p [password] File ,

Sendo que:

- o **-c** e **-d** são a indicação para compressão/ descompressão, respetivamente;
- **-l** a indicação do nível de compressão;
- **-s** se pretender exibir o sumário;
- **-h** exibe uma mensagem com os tópicos de ajuda e depois sai do programa;
- **-p** a indicação de realização de um processo de compressão/descompressão para/de um ficheiro encriptado, respetivamente.

Para que o ponto 1. seja realizado, os dados são lidos em modo binário e convertidos para o formato LZSS, onde depois o LZSSwriter escreve uma string codificada, as ditas referências <distância, comprimento>, ou descodificada, que poderá ser um objeto binário com apenas um byte, para um stream de output (figura 3).

```
def _lzs_encode(window):
    with lzss_io.io.BytesIO() as out:
        with lzss_io.LZSSWriter(out, ctx=ctx) as writer:
            for byte_int in window:
                writer.write(bytes((byte_int,)))
        #print('O ficheiro de saída tem os seguintes dados: ')
        out.seek(0)
        dados_comp = out.read()
        compressed_file.write(dados_comp)
```

Figura 3-Escrita de uma string comprimida.

A escolha do nível de compressão (ponto 2.) reflete-se no tamanho do buffer e por conseguinte, no tamanho final do ficheiro comprimido. Neste caso, os níveis variam entre 1 e 4 e não é possível assumir um valor fora deste intervalo (figura 4). Quanto maior o nível, maior o buffer e mais pequeno será o tamanho final do ficheiro. De notar que, nesta situação, o processo de compressão/descompressão será também mais lento devido à quantidade de bytes que tem que analisar para encontrar a sequência repetida. Pode ainda acontecer que não seja atribuído o nível de compressão na invocação do programa e aqui, ele assumirá um nível default que corresponde ao nível 2 (figura 5).

```
def set_level(level):

    if level not in range(1,5):
        print(f' The specified compression level {level} it is outside possible interval [1-4]')
        print("Exiting...")
        sys.exit(2)
    else:
        match level:
            case 1:
                print("Compression level 1 settings | window 1kb")
                comp_lvl = 1024
            case 2:
                print("Compression level 2 settings | window 4kb")
                comp_lvl = 4096
            case 3:
                print("Compression level 3 settings | window 16kb")
                comp_lvl = 16384
            case 4:
                print("Compression level 4 settings | windows 32kb")
                comp_lvl = 32768
    return comp_lvl
```

Figura 4-Definição dos níveis de compressão.

```
def encode(text_bytes, max_sliding_window_size=4096): # 4096 is default if argument is omited
```

Figura 5-Definição do nível Default.

Para o ponto 3., começou-se por criar um cabeçalho e guardar lá informação sobre os dados utilizados, como por exemplo o nome do ficheiro, a data em que o ficheiro foi comprimido, o nível de compressão escolhido e respetivo tamanho da janela e os autores. Caso se pretenda que este sumário seja exibido, basta colocar o comando -s aquando da invocação do programa e o resultado será algo semelhante ao indicado na figura 6.

```
Header data summary:

Name of compressed file : [fich_texto.LZS]
Timestamp : [1646150559.0221937]
Window size : [1024 KB] 10 Bits | length size : [Default]
Compression date / time : [03/01/2022 16:02:39]
Author(s): Ana Graça, Nuno Guerra, Sónia Jardim

The compressed file fich_texto.LZS has been sucessfuly created !
```

Figura 6- Resumo dos dados.

No que se refere ao ponto 4, ao introduzir uma palavra-passe estará a indicar que pretende encriptar o ficheiro resultante do processo de compressão ou se irá descomprimir um ficheiro encriptado. Para que isto fosse possível, recorreu-se a alguns módulos da biblioteca cryptografy, mais precisamente os módulos Fernet, hashes e à classe PBKDF2HMAC do módulo pbkdf2, e à biblioteca base64, que permite criar e decifrar a palavra-passe.

Para executar esta fase do projeto foi necessário recorrer a outras bibliotecas, para além das anteriormente referidas, tais como a biblioteca docopt para podermos exibir uma mensagem de ajuda guardada numa docstring e chama-la na linha de comandos através do comando -h, biblioteca struct para auxiliar na criação do cabeçalho, biblioteca time que obtermos uma leitura correta da data e hora no momento de compressão e das bibliotecas io, math, typing, bitarray e bitstruct para a leitura dos dados em modo binário e conversão em formato LZS.

A **terceira e última fase** deste projeto foi dedicada ao desenvolvimento da interface gráfica, recorrendo ao Qt Designer. Procurou-se criar uma interface (figura 7) que assumisse as funcionalidades

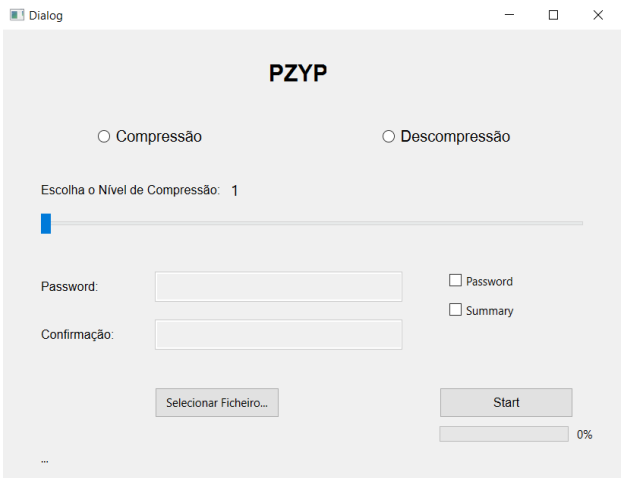

Figura 7- Interface Gráfica da Aplicação PZYP.

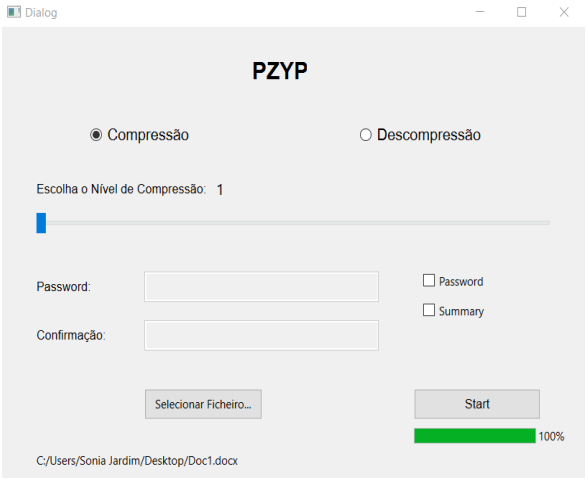

Figura 8- Interface após um processo de Compressão com o nível 1 e sem encriptação.

pensadas para a aplicação, tais como, que permita escolher o processo (compressão ou descompressão), selecionar o nível de compressão, opção de exibir o resumo dos dados, se pretendemos encriptar o ficheiro comprimido ou descomprimir um ficheiro encriptado, que possibilite selecionar o ficheiro para comprimir ou descomprimir e ter um botão para gerar a ação (botão "Start"). Optou-se também por colocar uma barra de progressão, para que o utilizador possa ter conhecimento que o processo terminou (a barra de progressão assume um valor de 100%, conforme indica a figura 8).

Criada a interface, viramo-nos para a ligação desta ao programa criado anteriormente e já testado na linha de comandos. No Qt Designer, cada interface gera um código em formato XML com a descrição dos objetos utilizados, o qual, para conectá-lo ao programa, foi necessário traduzi-lo para Python, com recurso ao comando:

$ pyside6-uic [caminho_ficheiro_ui] -o [caminho_ficheiro_py].

Uma vez traduzido, avançou-se para a gestão dos comportamentos e criação dos eventos correspondentes a cada objeto, de modo que as respostas fossem as esperadas, desta vez com recurso à biblioteca PySide6.

Procurou-se também minimizar os possíveis erros de preenchimento, criando-se validações, de forma que o utilizador consiga obter o pretendido sem dificuldade. Assim, aparecerá uma mensagem de

erro a indicar se não foi assinalado o processo pretendido, se, caso selecione a opção de encriptação, a repetição da password não coincidir com a primeira introdução ou estiver vazia, se o ficheiro a descomprimir não tiver um formato LZS e/ou se não for selecionado nenhum ficheiro e der a ordem de execução. Da mesma forma, no final de cada processo, será exibida uma mensagem a indicar que o programa foi realizado com sucesso e a localização e formato do mesmo. As figuras 9 e 10 mostram um exemplo de uma mensagem de um processo de compressão com encriptação e de um processo de descompressão de um ficheiro encriptado, respetivamente.

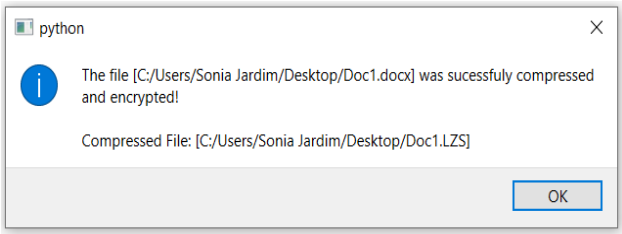

Figura 9- Exemplo de uma mensagem de conclusão do processo de Compressão e encriptação.

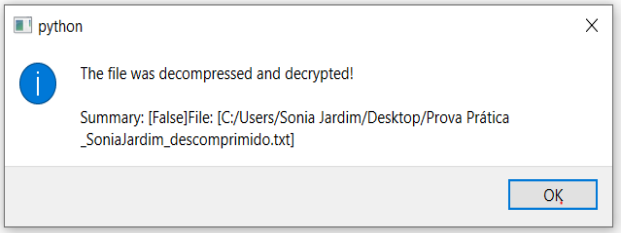

Figura 10-Exemplo de uma mensagem de conclusão do processo de Descompressão e descriptação.

4. Conclusão

Neste projeto, procurou-se desenvolver uma aplicação de compressão e descompressão de ficheiros, utilizando como base o algoritmo LZSS. Com este algoritmo, percebemos que para ocorrer compressão, seria necessário existir uma janela deslizante que percorresse o ficheiro e fosse capaz de identificar e codificar sequências repetidas, substituindo-as pela indicação da sua localização (distância) e comprimento da sequência. Percebemos também que esta mesma janela teria que estar presente no processo de descompressão, para novamente deslizar sobre o ficheiro comprimido e identificar as posições e tamanho das sequências repetidas, repondo-as no ficheiro de saída ou ficheiro descomprimido.

Para a sua realização, apercebemo-nos que os recursos às bibliotecas são de facto uma grande ajuda para a simplificação do código e que a construção do código por meio de funções e classes, torna o código mais organizado e percetível.

Com este projeto, criou-se uma aplicação que permite:

- comprimir qualquer tipo de ficheiro, guarda-o com formato LZS e encripta-o, se for a intensão;
- descomprimir ficheiros com formato LZS e guardá-los com o formato original,

e terminou com a criação de uma interface gráfica que permite levar esta aplicação a qualquer utilizador. Criou-se uma interface fácil de utilizar, com alertas orientadores caso o seu preenchimento não esteja correto para desempenhar as suas funções com sucesso.

5. Webgrafia

https://go-compression.github.io/algorithms/lzss/

https://en.wikipedia.org/wiki/Deflate

https://doc.qt.io/qt-5/qtdesigner-manual.html

https://doc.qt.io/qtforpython/tutorials/index.html

https://www.tutorialspoint.com/pyqt/