



**Universidade do Minho**

## **Departamento de Informática**

Mestrado Integrado em Engenharia Informática

Comunicação de Dados

# **Trabalho Prático: Implementação da Ferramenta Shafa**

## **Realizado por:**

**(Rodrigo Rodrigues A93201**

**Rui Monteiro A93179**

**Rui Moreira A93232**

**Bernardo Saraiva A93189**

**André Presa A82917)**

Universidade do Minho

Braga, 03 de Janeiro de 2021



Rodrigo Rodrigues



Rui Monteiro



Bernardo Saraiva



Rui Moreira



André Presa

## Índice

<b>Organização do grupo em torno do trabalho realizado .....</b>	<b>3</b>
<b>Estratégias e mecanismos adotados .....</b>	<b>4</b>
Módulo A .....	4
Módulo B.....	4
Módulo C.....	5
<b>Explicação e análise crítica das principais funções .....</b>	<b>6</b>
Módulo A .....	6
Módulo B.....	7
Módulo C.....	8
<b>Resultados da execução dos módulos .....</b>	<b>9</b>
Módulo A .....	9,10,11
Módulo B.....	11
<b>Conclusões.....</b>	<b>12</b>
<b>Referências... ..</b>	<b>13</b>

## **Organização do grupo em torno do trabalho desenvolvido**

A ferramenta a desenvolver é constituída por quatro módulos independentes. A fim de dividir tarefas, o grupo de 8 elementos foi subdividido em 4 grupos de dois elementos e foi atribuído a cada grupo um módulo do trabalho prático. Eis os grupos e os respetivos módulos:

**Módulo A:** Rodrigo Rodrigues (A93201) & Rui Monteiro (A93179)

- Compressão RLE
- Cálculo das Frequências dos símbolos

**Módulo B:** Rui Moreira (A93232) & Bernardo Saraiva (A93189)

- Tabelas de codificação
- Representação dos códigos no ficheiro cod

**Módulo C:** André Gil & André Presa

- Construir a sequência codificada de bits
- Otimização da codificação binária SF com matrizes de bytes

**Módulo D:** Bruno Silva & Ricardo Veloso

- Descodificar a sequência comprimida de bits
- Otimização da descodificação binária SF

Nota: Módulo **D** não foi entregue devido ao incumprimento da sua realização por parte do subgrupo encarregue. Módulo **C** foi entregue apenas por um dos membros (André Presa).

## Estratégias e Mecanismos adotados

- **Módulo A**

Armazenamento dos dados do ficheiro de entrada num Buffer em memória que consiste num array dinâmico, de modo a reduzir a complexidade e aumentar a eficiência do processamento dos dados. Por uma questão de desempenho, fez-se, para cada bloco, a contagem dos símbolos e a compressão RLE.

Implementação de uma função “*rle\_aux*” que faz a compressão individual de um bloco: Recebe um Buffer inicial e percorre o array dos símbolos, um a um, e gera o padrão RLE {0}{símbolo}{nº repeticoes} quando um símbolo se repete pelo menos 4 vezes. Nesta função existe ainda uma exceção (feita com uma função auxiliar “*excecao*”) para o caso de termos o símbolo {0}. Esta função escreve num ficheiro final (“.rle”) e é chamada na função “*rle*” de modo a aplicar a compressão bloco a bloco.

Implementação de uma função “*freq*”: Recebe um Buffer inicial (armazena o conteúdo do bloco) e calcula as frequências dos símbolos de um bloco e organiza-as num array de acordo com a ordem dos símbolos ASCII, guardando num Buffer final. Esta função é chamada na função “*rle\_to\_freq*” de modo a aplicar o cálculo das frequências dos símbolos para cada bloco. Para tal, define um Buffer que armazena os dados do ficheiro (“.rle”), distingue cada bloco através de um índice e vai formatando o Buffer final de acordo com as especificações do enunciado.

- **Módulo B**

Para a realização deste módulo decidimos armazenar o conteúdo do ficheiro que recebemos (“.freq”) num Buffer que consiste num array dinâmico. Optamos por este método para que ao longo de todo o processo não termos de lidar, diretamente, com alocação de memória, e para isso, criamos o header “buffer”, cujas funções lidam automaticamente com essas questões.

No processo de codificação, e para conseguir implementar o processamento por blocos, usamos 3 Buffers: o inicial (como já referido, armazena o conteúdo do ficheiro inicial), o intermédio (que auxilia o processamento armazenando sucessivamente o conteúdo do bloco a processar) e o final (que armazena os códigos já processados com a formatação ideal para escrever no ficheiro a devolver).

A cada bloco, organizamos as frequências num array de inteiros e aplicamos às frequências não nulas o algoritmo Shannon-Fano, que insere os códigos referentes aos símbolos numa matriz também ela dinâmica. A partir daí, inserimos os códigos no Buffer final, pela ordem antes da ordenação implícita no algoritmo.

- **Módulo C**

Escolheu-se dividir as tarefas dos dois elementos do grupo entre;

**“Tarefa Principal”:**

Onde se leva a cabo a manipulação dos ficheiros .txt, .RLE e .cod para criar o ficheiro .shaf e;

Para esta tarefa, determinou-se que se deveria proceder da seguinte forma;

1. Determinar o código ASCII para cada símbolo do ficheiro .txt
2. Usar esse código como “chave” para encontrar a codificação SF do símbolo no ficheiro .cod
3. Substituir cada símbolo do ficheiro .txt pelo seu código SF
4. Resolver o problema de codificação do ficheiro .RLE da mesma forma, evitando a conversão dos símbolos próprios da codificação .RLE, por exemplo: {0}1 e {31}1.

**“Tarefas Auxiliares”:**

Onde são resolvidos problemas como

- Leitura de ficheiros
- Criação de estruturas ARRAY
- Contagem do tempo de execução – ( )
- Impressão de texto de saída na consola

É de referir que grande parte das soluções de programação necessárias para completar as duas tarefas eram desconhecidas dos elementos do grupo e exigiram pesquisa e procura de exemplos.

# Explicação e Análise Crítica das Principais Funções

- **Módulo A**

- Função **exec\_moduloA**: função principal

Esta função encontra-se dividida em etapas:

1. Lê o ficheiro recebido e escreve num Buffer de “origem”
2. Aplica a compressão RLE e faz o cálculo das frequências
3. Escreve no terminal toda a informação sobre a sua execução

Limitações: Incapacidade de forçar compressão

- Função **rle** -> função que aplica a compressão RLE, bloco a bloco, aplicando-lhes a função **rle\_aux**, e que coloca num Buffer os resultados da compressão.

- Função **rle\_aux** -> Faz a compressão individual de um bloco.

Esta função encontra-se dividida em etapas:

1. Distinção entre o caso de exceção (quando temos o símbolo {0}) e os restantes casos
2. Distinção entre símbolos que se repetem pelo menos 4 vezes e os que se repetem menos de 4 vezes
3. Aplicação do padrão RLE {0}{símbolo}{nº repeticoes} aos primeiros e repetição dos segundos tal como estavam originalmente
4. Escrever num ficheiro de saída (“.rle”)

- Função **freq** -> função que faz o cálculo das frequências dos símbolos para um bloco individual e guarda as frequências dos símbolos num array organizado pela ordem dos símbolos ASCII.

Esta função encontra-se dividida em etapas:

1. Identificação dos casos de exceção (\*\*)
2. Armazenar num array as frequências de cada símbolo

\*\*Quando temos{0}{símbolo}{nº repetições}

\*\*Nº repetições pode ter 1, 2 ou 3 algarismos

- Função **rle\_to\_freq** -> Função que calcula as frequências dos símbolos dos blocos e escreve num ficheiro “.freq” a frequência de cada símbolo conforme as especificações do enunciado.

- **Módulo B**

- Função `exec_moduloB` -> função principal

Esta função encontra-se dividida em 2 etapas:

1. Lê o ficheiro recebido e escreve no Buffer inicial e depois da codificação, escreve o Buffer final previamente alocado no ficheiro a retornar.
2. Escreve no terminal toda a informação sobre a sua execução

- Função `code` -> esta função gere a divisão por blocos do Buffer inicial aplicando a estes a função `codeBlock` e coloca no Buffer final os dados que não precisam de processamento.

- Função `codeBlock` -> surge como solução para o processamento individual para cada bloco. Esta função encontra-se dividida em 3 etapas:

1. Organizamos por ordem decrescente as frequências num array (com o algoritmo BubbleSort);
2. Aplicamos a codificação propriamente dita com a função `calcular_codigos_SF` e armazenamos o resultado numa matriz dinâmica (optamos por esta solução por não saber exatamente o tamanho de cada código);
3. Com a codificação finalizada copiamos para o Buffer final os códigos pela ordem correta e para isso, usamos o array de frequências ainda desordenado que surge inicialmente para encontrar o índice onde o código deve ser escrito.

- Função `calcular_codigos_SF` -> algoritmo disponibilizado pelo professor e adaptado para escrever os códigos na matriz dinâmica.

- **Módulo C**

- Função **Main**: função principal

Esta função tem como objetivo ler e correr todas as funções secundárias.

- Função **timeCode** -> função que faz a contagem do tempo que demora a executar o programa.
- Função **fileOpener** -> função que abre e lê o ficheiro.
- Função **Contador** -> Esta função está dividida em etapas:
  1. Abre e lê um ficheiro .cod
  2. Utiliza um contador para contar especificamente 4 ocorrências do carácter “@”
  3. A partir da 4ª ocorrência de “@”, cria um array onde vai guardar todas as ocorrências de vazios (por cada vazio acrescenta no array o “-1”) e todas as ocorrências de números Shannon-Fano
- Função **filetxt** -> Esta função abre e guarda os elementos do ficheiro .txt num array.
- Função **fsize** -> função disponibilizada pelo professor para dar o tamanho dos blocos do ficheiro



## Resultados da Execução dos Módulos

- **Módulo A**

aaa.txt

[illegible]





## Conclusões

No Módulo A, gostaríamos de ter acrescentado a funcionalidade de forçar a compressão. Não foi possível adicioná-la uma vez que as tentativas de implementação causavam erros que comprometiam o bom funcionamento do resto do programa. Decidimos, portanto, manter o “grosso” do programa a funcionar, ainda que sem esta funcionalidade.

A fim de completar a ferramenta *Shafa* e capacitá-la de todas as funcionalidades previstas, gostaríamos de ter implementado os Módulos C e D (que, por incumprimento dos colegas encarregues pelos módulos, não foi possível entregar).

# Referências

- **Bibliográficas**

1. Bibliografia das disciplinas de Comunicação de Dados e Programação Imperativa

- **Recursos da internet**

1. [www.sanfoundry.com](http://www.sanfoundry.com)
2. [www.javatpoint.com](http://www.javatpoint.com)
3. [pt.stackoverflow.com/](http://pt.stackoverflow.com/)