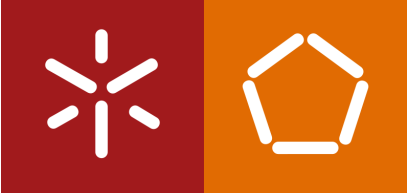




Catarina Pereira
Inês Neves
Leonardo Martins

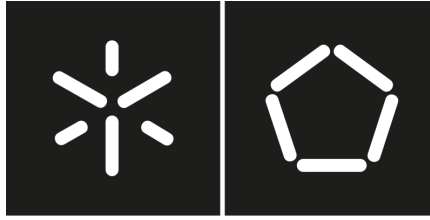
Compressão LZWdR



Universidade do Minho
Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Inês Cabral Neves
Leonardo Dias Martins

Compressão LZWdR



Universidade do Minho

Escola de Engenharia

Catarina da Cunha Malheiro da Silva Pereira
Inês Cabral Neves
Leonardo Dias Martins

Compressão LZWdR


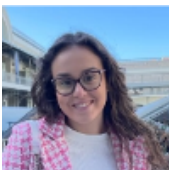

Serviços de Rede e Aplicações Multimédia
Trabalho prático de Mestrado
Engenharia em Telecomunicações e Informática

Trabalho efetuado sob a orientação de:

Professor Doutor Bruno Alexandre Fernandes Dias

Identificação do Grupo

O Grupo 08 é constituído por três deles que são do 1º ano do Mestrado em Engenharia de Telecomunicações e Informática (METI), sendo identificados por Pós-Graduação (PG) seguido dos seus números mecanográficos.

Imagem	Nome / Número Mecanográfico / E-mail institucional
	Catarina da Cunha Malheiro da Silva Pereira PG53733 pg537336@alunos.uminho.pt
	Inês Cabral Neves PG53864 pg53864@alunos.uminho.pt
	Leonardo Dias Martins PG53996 pg53996@alunos.uminho.pt

Índice

Identificação do Grupo	ii
Lista de Figuras	iv
Lista do Código	v
Siglas	v
1 Introdução	1
2 Revisão da Literatura	2
2.1 Algoritmo LZW	2
2.2 Algoritmo LZWdR	2
3 Abordagem Metodológica	4
4 Trabalho Realizado	5
4.1 Extras	7
4.2 Limitações	8
5 Testes	9
6 Conclusão	13

Lista de Figuras

1	Help menu.	9
2	Compressão ficheiro txt sem opções extra.	9
3	Compressão ficheiro .txt com limpeza total do dicionário e tamanho de dicionário de 2^{20}	10
4	Compressão ficheiro .txt apenas com limpeza total do dicionário e tamanho default 2^{16}	10
5	Compressão ficheiro .txt sem limpeza total do dicionário e tamanho de dicionário aumentado 2^{20}	10
6	Compressão ficheiro RAF com parâmetros default.	11
7	Compressão ficheiro RAF com limpeza total do dicionário e tamanho do dicionário aumentado 2^{20}	11
8	Compressão ficheiro RAF com limpeza total do dicionário e tamanho do dicionário default 2^{16}	12
9	SCompressão ficheiro RAF sem limpeza do dicionário e tamanho do dicionário aumentado 2^{20}	12

Lista do Código

int_trie function	5
insertPattern function	5
check_if_in_dici function	6
output_number function	6
pick_PB function	6

Siglas

METI Mestrado em Engenharia de Telecomunicações e Informática.

PG Pós-Graduação.

UC Unidade Curricular.

1 Introdução

Este relatório faz parte do segundo trabalho prático da Unidade Curricular (UC) Serviços de Rede e Aplicações Multimédia, do 2º semestre do 1º ano do Mestrado em Engenharia de Telecomunicações e Informática. Este projeto foi desenvolvido como resposta a um problema apresentado pelo docente.

Este relatório apresenta as características do trabalho prático de compressão LZWdR, uma variante da compressão por padrões (ou por dicionário) conhecida como LZW. A compressão LZWdR é uma técnica de compressão de dados que se baseia na substituição de sequências de símbolos originais por códigos respetivos, permitindo obter taxas de compressão e rendimentos virtuais superiores a 100%.

2 Revisão da Literatura

Este capítulo apresenta os conceitos teóricos necessários ao desenvolvimento deste projeto, assim como a respectiva revisão da literatura.

A compressão de dados é uma técnica fundamental em ciência da computação e engenharia de telecomunicações que visa reduzir o tamanho de arquivos para armazenamento e transmissão mais eficiente. Existem dois tipos principais de compressão: compressão sem perda (lossless) e compressão com perda (lossy). A compressão sem perda, como o próprio nome sugere, permite que os dados originais sejam perfeitamente recuperados a partir dos dados comprimidos, enquanto a compressão com perda elimina alguns dados, geralmente irrelevantes, para alcançar maiores taxas de compressão.

2.1 Algoritmo LZW

O algoritmo LZW é uma técnica de compressão sem perda baseada em dicionário, derivada dos algoritmos LZ77 e LZ78. O LZW foi desenvolvido por Abraham Lempel, Jacob Ziv e Terry Welch e é amplamente utilizado em formatos de arquivo como GIF, TIFF e PDF, bem como em utilitários de compressão como o ZIP e o RAR.

A principal característica do LZW é a substituição de sequências repetitivas de símbolos (padrões) por códigos de comprimento fixo. O algoritmo começa com um dicionário pré-inicializado contendo todos os possíveis símbolos únicos da entrada (geralmente os 256 caracteres ASCII). À medida que processa a entrada, o LZW constrói um dicionário de padrões mais longos que encontra, permitindo que sequências repetitivas sejam substituídas por códigos mais curtos.

2.2 Algoritmo LZWdR

A variante LZWdR introduz melhorias ao LZW ao adicionar novos padrões ao dicionário de forma dinâmica e mais eficiente. Enquanto o LZW tradicional adiciona um único novo padrão em cada iteração, o LZWdR pode adicionar múltiplos padrões, formados pela combinação de dois padrões já conhecidos. Isso resulta num crescimento mais rápido do dicionário, o que pode aumentar a eficiência da compressão, embora também aumente o consumo de memória.

A principal diferença operacional entre LZW e LZWdR está no método de atualização do dicionário. No LZW, cada novo padrão é formado pela concatenação de um padrão existente com um símbolo seguinte, enquanto no LZWdR, novos padrões podem ser formados pela combinação de dois padrões já existentes, acelerando assim a formação de padrões longos.

A implementação do LZWdR envolve a criação de uma aplicação codificadora que processa arquivos de entrada em blocos. Cada bloco é lido em um *buffer* na memória, onde é processado para identificar e registrar novos padrões no dicionário. O tamanho do bloco é um parâmetro crucial que afeta tanto a eficiência da compressão quanto o uso de memória. Por padrão, o tamanho do bloco é definido como 64 KBytes, mas pode ser configurável até 64 MBytes.

A ferramenta de codificação LZWdR inclui os seguintes parâmetros:

- 1. Tamanho dos Blocos:** Define o tamanho dos blocos para processamento do arquivo de entrada.

- 2. Tamanho Máximo do Dicionário:** Especifica o número máximo de entradas que o dicionário pode conter.
- 3. Tipo de Limpeza do Dicionário:** Define o tipo de limpeza que é feito quando é atingido o número máximo de padrões que o dicionário pode conter. O valor padrão é que não faz nada, i.e., quando se atinge o número máximo de entradas é usado o dicionário completo sem se acrescentar mais padrões novos.
- 4. Informações Estatísticas:** Define quais as estatísticas sobre o processo de codificação que são calculadas. O valor padrão inclui o número de bytes processados, o número de vezes que os padrões são encontrados, o número de códigos de padrões enviados para o output, o tamanho médio dos padrões inseridos no dicionário e o número de vezes que o tamanho máximo do dicionário foi atingido.

O algoritmo LZWdR representa uma evolução significativa do LZW, oferecendo potencialmente melhor eficiência de compressão através de um crescimento mais rápido do dicionário. Sua implementação exige uma cuidadosa consideração dos parâmetros operacionais para otimizar o desempenho e a utilização de memória. A compreensão dos fundamentos teóricos por trás desses algoritmos é essencial para a realização de compressão de dados eficiente e eficaz em diversas aplicações.

3 Abordagem Metodológica

O algoritmo LZWdR implementado neste projeto baseia-se no algoritmo de compressão LZW, com adaptações para melhorar a eficiência através da utilização de uma Trie para a gestão do dicionário de padrões e do processamento de dados em blocos. A Trie permite uma inserção e procura rápida dos padrões, enquanto o processamento em blocos facilita o manuseamento de grandes volumes de dados.

A compressão é realizada inicializando a Trie com os primeiros 256 padrões (cada byte individual). Durante o processo de compressão, novos padrões são inseridos na Trie e os dados são lidos e manipulados em blocos. O algoritmo incorpora estratégias de limpeza do dicionário para manter a eficiência na utilização da memória.

Os principais parâmetros utilizados no algoritmo incluem:

- Tamanho do bloco: Define o número de bytes processados em cada bloco. Um tamanho de bloco adequado é essencial para o desempenho do algoritmo.
- Comprimento do buffer: Define o tamanho do buffer utilizado para armazenar temporariamente os dados durante o processamento.
- Tamanho máximo do dicionário: Limita o número de padrões que podem ser armazenados na Trie, garantindo uma utilização eficiente da memória.

O ambiente de desenvolvimento utilizado para este projeto inclui:

- Sistema Operativo: O algoritmo foi desenvolvido e testado num ambiente Windows, devido à sua robustez e suporte a ferramentas de desenvolvimento.
- Editor de Código; Utilizou-se o Visual Studio Code.

A linguagem de programação utilizada foi a linguagem C, a escolha deve-se à sua eficiência e controlo sobre a gestão de memória, fundamentais para a implementação de algoritmos de compressão.

4 Trabalho Realizado

Neste capítulo é para detalhar a implementação do algoritmo de compressão LZWdR, cujo objetivo foi criar um algoritmo eficiente utilizando uma Trie para a gestão do dicionário de padrões e processamento de dados em blocos. O algoritmo, baseado no LZW, otimiza a inserção e procura de padrões, além de implementar estratégias de limpeza do dicionário para garantir eficiência na utilização de memória.

1. Estrutura de Dados

- Trie: A implementação do algoritmo LZWdR utiliza uma Trie para representar o dicionário. Cada nó da Trie contém um vetor de 256 ponteiros para outros nós, um índice, um contador de uso e um boolean que indica se o nó representa o final de um padrão.
- Nó da Trie: A estrutura “TrieNode” é usada para representar cada nó na Trie. Esta estrutura é inicializada com a função “createNode”, que aloca memória para o nó e inicializa os campos.

2. Inicialização do Dicionário

- Função “initTrie”: Inicializa o nó da raiz da Trie, onde cada filho do nó raiz representa um símbolo único de um byte. Assim, o dicionário começa com 256 entradas iniciais.

```
1 void initTrie() {
2     root = createNode();
3     for (int i = 0; i < 256; i++) {
4         unsigned char symbol[1] = { (unsigned char)i };
5         insertPattern(symbol, 1, i);
6     }
7     last_position_of_dici = 256;
8 }
```

Código 4.1: int_trie function

3. Inserção e Procura de Padrões

- Função “insertPattern”: Insere um novo padrão na Trie, utilizando a função “createNode” para criar novos nós conforme necessário.

```
1 void insertPattern(unsigned char *pattern, int length, int index){
2     TrieNode *node = root;
3     for (int i = 0; i < length; i++) {
4         int idx = pattern[i];
5         if (node->children[idx] == NULL) {
6             node->children[idx] = createNode();
7         }
8         node = node->children[idx];
9     }
10    node->isEndOfPattern = true;
11    node->index = index;
12 }
```

Código 4.2: insertPattern function

- Função “check_if_in_dici”: Verifica se um padrão já está presente na Trie. Esta função percorre a Trie a partir da raiz seguindo os filhos correspondentes aos símbolos do padrão.

```

1 bool check_if_in_dici(unsigned char *pattern, int length) {
2     TrieNode *node = root;
3     for (int i = 0; i < length; i++) {
4         int idx = pattern[i];
5         if (node->children[idx] == NULL) {
6             return false;
7         }
8         node = node->children[idx];
9     }
10    return node->isEndOfPattern;
11 }

```

Código 4.3: check_if_in_dici function

- Função “output_number”: Obtém o índice de um padrão na Trie. Se o padrão não for encontrado, a função retorna -1.

```

1 int output_number(unsigned char *pattern, int length) {
2     TrieNode *node = root;
3     for (int i = 0; i < length; i++) {
4         int idx = pattern[i];
5         if (node->children[idx] == NULL) {
6             return -1; // Padrão não encontrado
7         }
8         node = node->children[idx];
9     }
10    node->usageCount++;
11    return node->index;
12 }

```

Código 4.4: output_number function

4. Processamento de Blocos

- Leitura em Blocos: O arquivo de entrada é lido em blocos de tamanho configurável (padrão de 64 KB). Cada bloco é processado individualmente, o que facilita a manipulação de grandes arquivos e permite a implementação de diferentes estratégias de compressão em blocos.
- Função “pick_PB”: Seleciona o próximo padrão PB a ser processado a partir do buffer de entrada.

```

1 void pick_PB(unsigned char *pb, int *t) {
2     unsigned char aux[64];
3     aux[0] = buffer_of_file[pos_bloco][pos_bufferfile + 1];
4     pos_bufferfile++;
5     int tamanho = 1;
6     bool check = true;
7     while (check) {
8         if (check_if_in_dici(aux, tamanho)) {
9             pb[tamanho - 1] = buffer_of_file[pos_bloco][pos_bufferfile];
10            pos_bufferfile++;
11            aux[tamanho] = buffer_of_file[pos_bloco][pos_bufferfile];
12            tamanho++;
13            if (pos_bufferfile > tam_blocos - 1) {
14                tamanho--;
15                pos_bufferfile--;

```

```

16         check = false;
17     }
18     } else {
19         check = false;
20     }
21 }
22 *t = tamanho;
23 }

```

Código 4.5: pick_PB function

5. Codificação

- Função “compress”: Esta função principal implementa o processo de compressão. Utiliza as funções auxiliares para inserir e procura de padrões na Trie e gera a saída codificada.
- Buffer de Saída: Um buffer é usado para armazenar os códigos gerados durante a compressão antes de serem escritos no arquivo de saída.

6. Estatísticas e Informações

- Cálculo de Estatísticas: Durante a execução, são calculadas diversas estatísticas, como o número de padrões encontrados e inseridos, o tamanho do arquivo processado e o tempo total de execução.
- Impressão de Informações: A aplicação imprime informações sobre os padrões encontrados, o tamanho dos blocos processados e outras métricas relevantes.

7. Limpeza do Dicionário

- Estratégias de Limpeza: Implementação de diferentes estratégias de limpeza do dicionário (sem limpeza, limpeza total e limpeza parcial), configuráveis pelo utilizador. Esta funcionalidade é crucial para gerir eficientemente o espaço do dicionário e manter o desempenho da compressão.

4.1 Extras

O trabalho desenvolvido foi aprimorado com a inclusão de várias funcionalidades adicionais que aumentam a flexibilidade e eficiência do algoritmo LZWdR. Entre estas melhorias, destacam-se as seguintes:

- O tamanho dos blocos para processamento do ficheiro de entrada pode agora ser configurado para valores diferentes de 64 KByte, permitindo uma adaptação mais precisa às necessidades específicas dos dados a serem comprimidos.
- O tamanho máximo do dicionário foi ajustado para ser configurável, permitindo valores entre 2^{12} e 2^{24} . Esta flexibilidade garante uma utilização otimizada da memória e pode melhorar a taxa de compressão para diferentes conjuntos de dados.
- Foram implementados diferentes tipos de limpeza do dicionário, incluindo a opção de uma limpeza completa (reset), que restaura o dicionário ao seu estado inicial quando o número máximo de entradas é atingido, e uma limpeza seletiva, que elimina apenas os padrões menos usados. Estas estratégias de gestão do dicionário asseguram que o algoritmo mantém a sua eficiência ao longo do tempo.

Estas funcionalidades adicionais permitem uma maior adaptabilidade do algoritmo a diversas situações, otimizando o processo de compressão conforme as características específicas dos dados processados.

4.2 Limitações

Apesar da eficácia do algoritmo LZWdR na compressão de dados, existem algumas limitações a considerar. Dependendo dos parâmetros utilizados durante o processo de compressão, como o tamanho do bloco e do buffer, o ficheiro comprimido pode acabar por ser maior do que o ficheiro original, especialmente em casos de dados que não se beneficiam da compressão. Além disso, o algoritmo está restrito a dicionários com um tamanho máximo de 2^{20} entradas. Esta limitação pode ser insuficiente para alguns conjuntos de dados, onde um dicionário maior poderia potencialmente aumentar a taxa de compressão. Estas restrições destacam a importância de uma cuidadosa seleção de parâmetros e a necessidade de futuras melhorias para adaptar o algoritmo a diferentes tipos de dados e cenários.

5 Testes

Este capítulo apresenta os resultados obtidos com a implementação do algoritmo de compressão LZWdR. Os testes foram realizados com diferentes tipos de arquivos e parâmetros de configuração para avaliar a eficiência e desempenho do algoritmo.

A Figura 1 apresenta o menu de ajuda do programa. Este menu fornece uma visão geral das opções e parâmetros disponíveis para a execução do algoritmo LZWdR. Ele detalha as funcionalidades, comandos e exemplos de utilização, servindo como uma referência para os utilizadores que desejam compreender melhor as capacidades do programa e como utilizá-lo de forma eficiente.

```
PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -h
Usage: program [OPTIONS]
Options:
-h, --help            Show this help message and exit
-i, --input            Specify the input file
-o, --output           Specify the output file
-c, --cleanup 0|1|2   Specify the cleanup option (0, 1, or 2)
                      (0-> No cleanup, 1 -> Full cleanup, 2-> Partial cleanup)
-d, --dictionary 0|1|2 Specify the dictionary size option (0, 1, or 2)
                      (0-> 2^16 , 1 -> 2^18, 2-> 2^20)
```

Figura 1: Help menu.

A Figura 2 mostra o resultado da compressão de um ficheiro .txt utilizando o algoritmo LZWdR sem opções adicionais. Este teste serve como linha de base, permitindo avaliar a eficácia do algoritmo em condições padrão. A análise inclui a taxa de compressão obtida e o tempo de execução.

```
PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.txt -o saida.lzwdr
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.txt
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 65536
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Sem Limpeza.
-----
Tempo de execucao final: 0.7890 segundos
Numero de padroes no dicionario: 65536
Quantidade de dicionarios cheios: 0
Tamanho do ficheiro: 5776694 bytes
Numero de blocos : 89
Tamanho dos blocos: 65536 bytes
Tamanho do ultimo bloco: 9526 bytes
Tamanho do ficheiro de saida: 2947768 bytes
Taxa de compressao: 48.97%
PS C:\Users\35196\Desktop\tp1-sram\SRAM>
```

Figura 2: Compressão ficheiro txt sem opções extra.

A Figura 3 ilustra a compressão de um ficheiro .txt utilizando uma limpeza total do dicionário e um tamanho de dicionário aumentado para 2^{20} entradas. Este teste avalia o impacto de uma maior capacidade do dicionário e da estratégia de limpeza completa na taxa de compressão e no desempenho. A limpeza total do dicionário ajuda a evitar a saturação da memória, enquanto um dicionário maior pode potencialmente melhorar a compressão ao armazenar mais padrões.

```

PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.txt -o saida.lzwdr -c 1 -d 2
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.txt
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 1048576
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Total.
-----
Tempo de execucao final: 6.7710 segundos
Numero de padroes no dicionario: 788947
Quantidade de dicionarios cheios: 2
Tamanho do ficheiro: 5776694 bytes
Numero de blocos : 89
Tamanho dos blocos: 65536 bytes
Tamnho do ultimo bloco: 9526 bytes
Tamanho do ficheiro de saida: 1822166 bytes
Taxa de compressao: 68.46%

```

Figura 3: Compressão ficheiro .txt com limpeza total do dicionário e tamanho de dicionário de 2^{20} .

A Figura 4 apresenta os resultados da compressão de um ficheiro .txt utilizando a limpeza total do dicionário com o tamanho padrão de 2^{16} entradas. Este teste permite avaliar a eficácia da limpeza total do dicionário sem a alteração do tamanho do dicionário. Comparando com os resultados anteriores, podemos observar a influência do tamanho do dicionário no desempenho e na taxa de compressão.

```

PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.txt -o saida.lzwdr -c 1
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.txt
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 65536
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Total.
-----
Tempo de execucao final: 10.6490 segundos
Numero de padroes no dicionario: 23562
Quantidade de dicionarios cheios: 70
Tamanho do ficheiro: 5776694 bytes
Numero de blocos : 89
Tamanho dos blocos: 65536 bytes
Tamnho do ultimo bloco: 9526 bytes
Tamanho do ficheiro de saida: 2661988 bytes
Taxa de compressao: 53.92%

```

Figura 4: Compressão ficheiro .txt apenas com limpeza total do dicionário e tamanho default 2^{16} .

A Figura 5 mostra a compressão de um ficheiro .txt com o tamanho do dicionário aumentado para 2^{20} entradas, mas sem a limpeza total do dicionário. Este teste avalia como a ausência de uma estratégia de limpeza afeta o desempenho e a eficiência da compressão, especialmente quando o dicionário atinge a capacidade máxima.

```

PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.txt -o saida.lzwdr -d 2
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.txt
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 1048576
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Sem Limpeza.
-----
Tempo de execucao final: 2.3490 segundos
Numero de padroes no dicionario: 1048576
Quantidade de dicionarios cheios: 0
Tamanho do ficheiro: 5776694 bytes
Numero de blocos : 89
Tamanho dos blocos: 65536 bytes
Tamnho do ultimo bloco: 9526 bytes
Tamanho do ficheiro de saida: 1767182 bytes
Taxa de compressao: 69.41%

```

Figura 5: Compressão ficheiro .txt sem limpeza total do dicionário e tamanho de dicionário aumentado 2^{20} .

A Figura 6 apresenta a compressão de um ficheiro RAF utilizando os parâmetros padrão do algoritmo LZWdR. Este teste serve como referência para a compressão de dados binários mais complexos, permitindo comparar a eficácia do algoritmo em diferentes tipos de ficheiros. A análise inclui a taxa de compressão e o tempo de execução, fornecendo uma base para avaliar as melhorias introduzidas pelas funcionalidades adicionais.

```
PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.RAF -o saida.lzwdr
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.RAF
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 65536
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Sem Limpeza.
-----
Tempo de execucao final: 4.3010 segundos
Numero de padroes no dicionario: 65536
Quantidade de dicionarios cheios: 0
Tamanho do ficheiro: 50569728 bytes
Numero de blocos : 772
Tamanho dos blocos: 65536 bytes
Tamnho do ultimo bloco: 41472 bytes
Tamanho do ficheiro de saida: 68438436 bytes
Taxa de compressao: -35.33%
```

Figura 6: Compressão ficheiro RAF com parâmetros default.

A Figura 7 mostra os resultados da compressão de um ficheiro RAF utilizando uma limpeza total do dicionário e um tamanho de dicionário aumentado para 2^{20} entradas. Este teste avalia como a combinação de uma maior capacidade do dicionário e uma estratégia de limpeza completa afeta a compressão de ficheiros binários complexos.

```
PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.RAF -o saida.lzwdr -c 1 -d 2
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.RAF
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 1048576
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Total.
-----
Tempo de execucao final: 166.2210 segundos
Numero de padroes no dicionario: 214416
Quantidade de dicionarios cheios: 47
Tamanho do ficheiro: 50569728 bytes
Numero de blocos : 772
Tamanho dos blocos: 65536 bytes
Tamnho do ultimo bloco: 41472 bytes
Tamanho do ficheiro de saida: 35099436 bytes
Taxa de compressao: 30.59%
```

Figura 7: Compressão ficheiro RAF com limpeza total do dicionário e tamanho do dicionário aumentado 2^{20} .

A Figura 8 ilustra a compressão de um ficheiro RAF com a limpeza total do dicionário e o tamanho padrão de 2^{16} entradas. Este teste permite comparar a eficácia da limpeza total do dicionário sem alterar o tamanho do dicionário, destacando a importância de estratégias de gestão de memória na compressão de dados complexos. A análise inclui a taxa de compressão e o tempo de execução.

```

PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.RAF -o saida.lzwdr -c 1
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.RAF
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 65536
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Total.
-----
Tempo de execucao final: 97.1200 segundos
Numero de padroes no dicionario: 10952
Quantidade de dicionarios cheios: 758
Tamanho do ficheiro: 50569728 bytes
Numero de blocos : 772
Tamanho dos blocos: 65536 bytes
Tamanho do ultimo bloco: 41472 bytes
Tamanho do ficheiro de saida: 48206234 bytes
Taxa de compressao: 4.67%

```

Figura 8: Compressão ficheiro RAF com limpeza total do dicionário e tamanho do dicionário default 2^{16} .

A Figura 9 apresenta os resultados da compressão de um ficheiro RAF com o tamanho do dicionário aumentado para 2^{20} entradas, mas sem a implementação de uma estratégia de limpeza do dicionário. Este teste avalia o impacto da ausência de limpeza na eficiência da compressão e no desempenho, especialmente em ficheiros binários complexos.

```

PS C:\Users\35196\Desktop\tp1-sram\SRAM> .\lzwdr.exe -i exemplo.RAF -o saida.lzwdr -d 2
Autores: Catarina Pereira PG53733, Ines Neves PG53864 e Leonardo Martins PG53996
Data de criacao: 30/05/2024 - 12/06/2024
Ficheiro de entrada: exemplo.RAF
Ficheiro de Saida: saida.lzwdr
Tamanho maximo do dicionario: 1048576
Tipo de dicionario inicial: 256
Tipo de limpeza de dicionario: Sem Limpeza.
-----
Tempo de execucao final: 8.2280 segundos
Numero de padroes no dicionario: 1048576
Quantidade de dicionarios cheios: 0
Tamanho do ficheiro: 50569728 bytes
Numero de blocos : 772
Tamanho dos blocos: 65536 bytes
Tamanho do ultimo bloco: 41472 bytes
Tamanho do ficheiro de saida: 39079124 bytes
Taxa de compressao: 22.72%

```

Figura 9: SCompressão ficheiro RAF sem limpeza do dicionário e tamanho do dicionário aumentado 2^{20} .

6 Conclusão

O desenvolvimento do programa codificador de ficheiros que implementa o algoritmo LZWdR foi realizado sucesso, apesar de certas dificuldades iniciais. Além dos cálculos de dados estatísticos pretendidos, o grupo decidiu desenvolver funcionalidades e otimizações adicionais, de forma a tornar o trabalho mais eficiente.

A aplicação desenvolvida foi capaz de processar ficheiros de entrada em blocos e gerar ficheiros de saída comprimidos utilizando o algoritmo LZWdR. Além disso, a aplicação calcula e imprime várias estatísticas sobre o processo de codificação, incluindo o número de bytes processados, o número de vezes que os padrões são encontrados, o número de códigos de padrões enviados para o output, o tamanho médio dos padrões inseridos no dicionário e o número de vezes que o tamanho máximo do dicionário foi atingido.