Universidade do Minho
Escola de Engenharia

Mestrado em Engenharia de Telecomunicações e Informática

Serviços de Rede & Aplicações Multimédia

Docente: Bruno Alexandre Fernandes Dias

Trabalho Prático Nº1 – Compressão LZWd

Rui Filipe Ribeiro Freitas - pg47639@alunos.uminho.pt
Tiago João Pereira Ferreira - pg47692@alunos.uminho.pt

28 de junho de 2022

Índice

1 Introdução

Atualmente existem vários especialistas que acreditam que uma das próximas crises que a humanidade enfrentará será a falta de espaço para guardar informação digital. Com isto, a compressão de dados torna-se uma ferramenta com um papel fundamental na era digital em que vivemos.

Neste relatório é apresentada a nossa implementação do algoritmo LZW e LZWd, bem como a comparação entre os mesmos. A realização deste trabalho teve como objetivo a familiarização com técnicas de compressão de dados assim como o seu armazenamento.

De modo a sermos capazes de cumprir com os objetivos deste trabalho foi necessário pôr em prática conhecimentos adquiridos ao longo do semestre na unidade curricular assim como realizar alguma pesquisa de eventuais dificuldades encontradas.

2 Desenvolvimento

Nesta secção do relatório é apresentada a parte mais prática do projeto onde se encontra a discussão das estratégias escolhidas, as opções tomadas e os mecanismos adotados.

2.1 Estratégias escolhidas

Para o desenvolvimento dos algoritmos LZW e LZWd, este foi implementado em linguagem C, sendo que foi necessário decidir vários aspetos importantes para a implementação do código. O primeiro passo consistiu na leitura do ficheiro a ser comprimido para memória. Para isto o grupo optou por ler o ficheiro em blocos de tamanho definido pelo utilizador. Depois de lido, é aplicado ao bloco o tipo de compressão desejada pelo utilizador.

Outra decisão importante foi a escolha da estrutura de dados onde seriam guardados os diferentes padrões, onde o grupo decidiu implementar uma matriz de inteiros em que cada linha corresponde a um padrão e cada coluna corresponde a um símbolo.

2.2 Análise crítica

Nesta subsecção são apresentadas as funções mais relevantes na implementação da solução.

Primeiro é apresentada a função *checkIfExistsInDictionary()*, apresentada na figura seguinte. Esta tem como objetivo receber um padrão com um dado tamanho e realizar a procura no dicionário desse padrão e caso encontre, retorna o valor da linha do dicionário onde se encontra o maior padrão igual, caso contrário retorna -1.

```
int checkIfExistsInDictionary(char *cod, int size_cod, int dictionary[DICTIONARY_SIZE][DICTIONARY_LENGTH])
{
    int line = -1;
    int t = 1;
    for (int i = 0; i < DICTIONARY_SIZE; i++)
    {
        if (*(cod + t - 1) == dictionary[i][t])
        {
            t++;
            for (int j = t; j < size_cod + 1; j++)
            {
                if (*(cod + j - 1) == dictionary[i][j])
                {
                    t++;
                }
            }
            if (t == size_cod + 1 && dictionary[i][t] == -1)
            {
                line = i;
                break;
            }
            t = 1;
        }
    }
    return line;
}
```

Figura 1: Função para ver se um elemento existe no dicionário.

Como o grupo implementou tanto o algoritmo LZW como LZWd e a unica diferença entre estes é a procura do padrão pK, de seguida é apresentado o excerto de código onde é realizada a distinção entre ambos. Inicialmente é realizada a procura do padrão pJ e após isso, tendo em conta o *type*, decide qual a forma correta de obter o pK.

```
if (type == 1)
{ // LZW
    *(pk) = *(temp_buffer + posicao_no_buffer);
    tries4pm = tries4pk;
    pm[tries4pm] = *(pk);
}
else if (type == 2)
{ // LZWD
    for (size_t triesPk = 0; triesPk < 64; triesPk++)
    {
        *(pk + triesPk) = *(temp_buffer + posicao_no_buffer + triesPk);
        int line = checkIfExistsInDictionary(pk, triesPk + 1, dictionary);
        if (line == -1)
        {
            tries4pm = tries4pk + triesPk;
            for (int i = tries4pk; i < tries4pm; i++)
            {
                pm[i] = *(pk + i - tries4pk);
            }
            triesPk = 64;
        }
    }
}
```

Figura 2: Procura do padrão pK (LZW vs LZWd).

3 Testes e discussão de resultados

De modo a comprovar a solução implementada foram realizados alguns testes cujos resultados se encontram na tabela seguinte.

Ficheiro	Tamanho (Kb)	Tam Bloco (b)	LZW Tempo(s)	LZWd Tempo(s)	LZW Compressão(%)	LZWd Compressão(%)
test.txt	6.4	4095	0.01	0.02	69.34	19.88
		65535	0.01	0.02		
test2.txt	68.02	4095	0.22	0.49	76.31	43.95
		65535	0.24	0.54		
eng_file.txt	3010.2	4095	6.15	12.02	39.49	16.4
		65535	6.25	12.4		

Figura 3: Tabela de resultados.

Através da observação da tabela anterior, onde estão presente os valores obtidos na realização dos testes, é possível afirmar que o ficheiro *eng_file.txt* demorava muito tempo a realizar a compressão devido a este ter 3Mbytes de dados. De salientar que

foi obtida uma taxa de compressão bastante superior no caso do ficheiro *test2.txt* visto que este continha grandes sequências de símbolos iguais.

Uma razão para o qual os valores obtidos não serem os melhores é devido ao grupo ter optado por utilizar a inserção e procura de padrões no dicionário através de matrizes sendo que este é um tipo de procura bastante lento comparado com outros métodos como por exemplo hash tables e matrizes de índices.

Após ser realizada a compressão do ficheiro é realizada a escrita do output num ficheiro com o nome *output.txt*, em que na solução implementada são escritos 2 bytes por cada padrão. Após ser realizada a compressão e escrita no ficheiro de saída é apresentado ao utilizador certas informações relativas à compressão. De seguida é possível observar o que o utilizador tem acesso após ser realizada a compressão LZW com um tamanho de bloco de 64k e um tamanho de ficheiro de 6400 bytes.

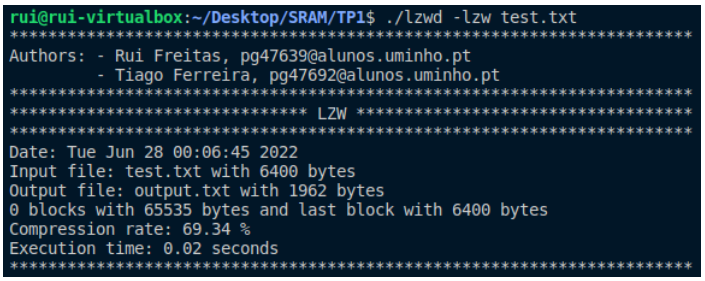

Figura 4: Informações apresentadas.

4 Conclusão

O principal objetivo deste trabalho foi a implementação do algoritmo LZWd e LZW, bem como a comparação entre os mesmos. A principal diferença entre os dois algoritmos é o tamanho do pK, pois no LZW este tem tamanho de 1 símbolo, já no LZWd tem tamanho variável.

No final da implementação foram realizados diferentes testes de modo a comprovar o bom funcionamento da solução implementada pelo grupo. Depois de analisados os resultados obtidos, o grupo sente que realizou um trabalho positivo, no entanto com algumas melhorias que poderiam ser implementadas, nomeadamente na estrutura de dados do dicionário bem como na procura de padrões no mesmo.