# Projeto 2 de Fundamentos de Compressão de Sinais
# LZ77

Giordano Süffert Monteiro - 17/0011160

## I. Descrição do Algoritmo

### A. Codificador

O programa criado, primeiramente espera que o usuário informe o nome do arquivo que será lido a partir do diretório atual e em seguida o diretório do arquivo que será gerado (opcional). Em seguida, essas informações são processadas de forma a criar os arquivos com nomes e caminhos corretos.

A partir daí, são executados os passos do LZ77 de casamentos de padrões e geração de triplas porém sem escrever em um arquivo, e sim contabilizando os índices e comprimentos dessas triplas de forma a gerar um dicionário com os símbolos em bytes e seus probabilidades.

De posse dessas probabilidades, é gerado um código de huffman através de uma árvore seguindo a ordem lexicográfica dos símbolos na escolha dos códigos nessa árvore. Além disso, são gerados 32 bytes indicando os símbolos codificados e uma sequência de bytes com o comprimento dos códigos gerados, ambas essas informações para serem informadas no cabeçalho do arquivo a ser criado, após a assinatura '0xfz' em que z será o tamanho do padding. (Foi reutilizado e adaptado o algoritmo criado no Projeto 1)

Assim, após preenchido o início do arquivo com esses dados, é realizada novamente a execução do algoritmo do LZ77 porém agora escrevendo os resultados no arquivo alvo utilizando a codificação de Huffman para os índices e comprimentos resultantes, gerando assim o arquivo comprimido com a extensão '.bin', informando no espaço reservado do header o padding final utilizado.

Vale ressaltar que execução do algoritmo do LZ77 consistiu nos seguintes passos:

1) **Escolha dos tamanhos máximos dos buffers:** Os limites máximos para os tamanhos do Search Buffer e do Look Ahead Buffer são selecionados como 16 símbolos ou para os valores informados no momento da execução do programa (max 255).
2) **Casamento de padrões:** Procura pelo melhor padrão iniciado pelo primeiro elemento do Look Ahead Buffer no Search Buffer, podendo se estender até o Look Ahead Buffer novamente.
3) **Atualização dos buffers:** Modifica os buffers para se manterem no tamanho permitido e receberem os próximos símbolos.

No casamento de padrões, o tamanho mínimo de casamento utilizado é 1 e a busca nos buffers é feita de maneira linear, comparando byte com byte, de forma que, após identificada uma combinação para o primeiro símbolo, é verificado a igualdade dos símbolos subsequentes até que um não combine ou o Look Ahead Buffer chegue no último símbolo. Assim, é possível armazenar e informar o casamento de maior comprimento para um mesmo Look Ahead Buffer.

Na atualização dos buffers, os símbolos já codificados pelo LZ77 são adicionados ao Search Buffer e se caso fosse ultrapassar seu tamanho máximo, são removidos os símbolos mais antigos necessários. Já o Look Ahead Buffer, como começa já preenchido até o seu tamanho máximo, remove os símbolos codificados pelo LZ77 e adiciona os próximos existentes no arquivo de leitura.

### B. Decodificação

No programa que realiza a decodificação, primeiramente é pedido o nome do arquivo a ser decodificado a partir do diretório atual e, igual ao codificador, pode ser informado um diretório de destino para o resultado, e o tamanho do Search Buffer utilizado na codificação no momento da execução do programa na linha de comando. Dessa forma,é verificada a extensão do arquivo e iniciada a sua leitura, sendo primeiro extraído o byte de header informando o padding existente; seguido pelos 32 bits indicando quais os símbolos existentes, gerando um lista com esses símbolos; e pelos bytes contendo o comprimento do código de cada um deles, gerando uma lista com esses valores, sendo esses símbolos e comprimentos informações para a utilização do código de Huffman.

Com isso, da mesma forma que o codificador, é gerado o código de huffman tendo como resultado o mesmo código que foi utilizado pelo codificador. Assim, é possível dar início ao processo de decodificação do conteúdo do arquivo, ao ser lida uma tripla por vez, de forma a interpretar os dois primeiros símbolos como codificações huffman e o terceiro como o símbolo seguinte.

Assim, utilizando os valores da tripla, é preenchido um Search Buffer idêntico ao da codificação, escrevendo tanto nesse buffer quanto no arquivo a sequência de símbolos que começa no item de valor informado pelo primeiro elementro da tripla (índice) e tendo tamanho informado pelo segundo (comprimento), reescrevendo assim o arquivo e refazendo as operações feitas pelo codificador na construção do Search Buffer, lembrando de sempre ao final da decodificação de uma tripla ajustar o tamanho do Search Buffer para aquele máximo informado (deve ser igual ou maior que o utilizado no codificador).

## II. Resultados

TABLE I

COMPRESSÃO DOS ARQUIVOS

| Nome do Arquivo | Entropia 1ª Ordem | Tamanho Original (bytes) | Tamanho Comprimido (bytes) (%)* |
|---|---|---|---|
| dom_casmurro.txt | 4,6430 | 389.670 | 312.705 (80,25%) |
| fonte.txt | 2,8940 | 1.000.000 | 678.699 (67,87%) |
| fonte0.txt | 1,5835 | 1.000 | 553 (55,30%) |
| fonte1.txt | 3,3033 | 1.000.000 | 744.875 (74,49%) |
| TEncEntropy.txt | 5,2719 | 19.415 | 10.107 (52,06%) |
| TEncSearch.txt | 5,1581 | 253.012 | 127.818 (50,52%) |

* 100 * Novo / Original

TABLE II

VARIANDO PARÂMETROS DO LZ77

| Nome do Arquivo | Tamanho Search Buffer | Tamanho Look Ahead Buffer | Compressão (%)* |
|---|---|---|---|
| TEncSearch.txt | 32 | 32 | 86,67% |
| TEncSearch.txt | 32 | 255 | 86,72% |
| TEncSearch.txt | 128 | 32 | 61,62% |
| TEncSearch.txt | 128 | 255 | 60,90% |
| TEncSearch.txt | 255 | 32 | 51,58% |
| TEncSearch.txt | 255 | 255 | 50,52% |
| fonte.txt | 32 | 32 | 78,39% |
| fonte.txt | 32 | 255 | 78,39% |
| fonte.txt | 128 | 32 | 70,74% |
| fonte.txt | 128 | 255 | 70,74% |
| fonte.txt | 255 | 32 | 67,87% |
| fonte.txt | 255 | 255 | 67,87% |

* 100 * Novo / Original

## III. COMPARAÇÕES

TABLE III

DIFERENTES COMPRESSÕES

| Nome Arquivos | Percentuais resultantes da compressão (%) | | | |
|---|---|---|---|---|
| | *Implementado* * | *Deflate* | *RAR* | *Huffman (Proj 1)* |
| dom_casmurro.txt | 80,25% | 36,91% | 34,82% | 58,54% |
| fonte.txt | 67,87% | 41,03% | 42,32% | 36,87% |
| fonte0.txt | 55,30% | 32,00% | 39,20% | 24,70% |
| fonte1.txt | 74,49% | 46,81% | 48,73% | 41,58% |
| TEncEntropy.txt | 52,06% | 21,51% | 22,14% | 66,96% |
| TEncSearch.txt | 50,52% | 14,84% | 13,61% | 64,95% |

* 255 como tamanho para os buffers

## IV. CONCLUSÃO

Sobre o comportamento observado do algoritmo do LZ77 utilizando buffers de 255 elementos e o algoritmo de Huffman para codificar os índices e comprimentos, é possível observar que o algoritmo apresentou uma taxa de compressão não muito boa (resultado maior que 50% do original), sendo pior para os arquivos "fonte*.txt", criados para o contexto desse experimento, não representando uma situação de prática real, porém foi inesperadamente ruim para "dom_casmurro.txt".

Pode-se observar também, baseado na diferença de comportamento do algoritmo ao mudarmos os tamanhos dos buffers, que para um melhor desempenho devemos ter um Search Buffer maior possível, representando uma mudança bastante significativa na compressão dos arquivos. Já sobre o Look Ahead Buffer representa uma melhora menor de compressão com o seu aumento, porém não apresenta uma piora na compressão em nenhum dos casos observados.

Já com os resultados das comparações com algoritmos de compressão utilizados comercialmente (tabela III) e aquele implementado no Projeto 1, é possível observar que o algoritmo implementado aqui só foi melhor do que o Huffman puro no caso dos arquivos "TEnc*.txt", que representam códigos de programas, onde ocorre uma grande quantidade de repetições de pequenos termos relativamente "próximos"um do outro. Porém, comparando o LZ77 implementado com o "Deflate"e o "RAR"podemos ver que ele apresenta uma performance claramente bem pior, sem deixar evidente vantagens para nenhum dos tipos de arquivos testados, sejam eles fabricados para o contexto da aplicação ou baseados em usos reais.

Com isso, é evidente que o algoritmo implementado precisa de melhorias nas escolhas de projeto realizadas, como por exemplo, aumentar bastante o tamanho do buffer para dezenas de kBytes em vez de somente 255, visto que o "Deflate"apresenta uma performance muito melhor sendo baseado no LZ77. Outro fator que pode influenciar o resultado do algoritmo é a aplicação de uma codificação também aos símbolos correspondentes ao "próximo símbolo"nas triplas, utilizando o próprio Huffman por exemplo, algo que também é feito no "Deflate".

Outro fator importante que deve ser levado em consideração é o tempo de processamento dos arquivos. Nas situações testadas, nos maiores arquivos o tempo de espera para codificação foi de, em média, 50 segundos e para decodificação de 25, utilizando um buffer "pequeno", como já foi visto, de 255 bytes; indicando assim que ao aumentar o tamanho do buffer como sugerido, deve-se também buscar fazer aprimoramentos em termos de tempo de processamento do algoritmo ao manipular os dados do arquivo, como por exemplo, utilizar uma árvore de busca para reduzir o tempo de busca pelo melhor casamento. Assim, partindo da implementação atual e realizando os aprimoramentos sugeridos, acredito que o algoritmo implementado possa gerar resultados cada vez mais parecidos com aqueles visto comercialmente.