INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
DE SANTA CATARINA

CAMPUS SÃO JOSÉ

RHENZO HIDEKI SILVA KAJIKAWA

ESTUDO E IMPLEMENTAÇÃO DOS ALGORITMOS DE COMPRESSÃO LZ77 E CODIFICAÇÃO ARITMÉTICA NA BIBLIOTECA KOMM

SÃO JOSÉ

2025

Rhenzo Hideki Silva Kajikawa

Estudo e Implementação dos Algoritmos de Compressão LZ77 e Codificação Aritmética na Biblioteca Komm

Projeto de Trabalho de conclusão de curso apresentado à Coordenadoria do Curso de Engenharia de Telecomunicações do Campus São José do Instituto Federal de Santa Catarina.

Área de concentração: Telecomunicações

Orientador: Prof. Roberto Wanderley da Nóbrega, Dr.

São José

2025

RESUMO

A biblioteca de compressão de dados Komm será estendida com a integração de dois algoritmos sem perdas: LZ77 e codificação aritmética. O cenário de redes de telecomunicações, marcado pelo aumento exponencial de dados, exige técnicas eficientes de compressão para otimização de largura de banda e armazenamento. Os objetivos englobam o estudo teórico dos algoritmos, projeto e implementação de módulos na arquitetura existente em Python, documentação e validação por meio de testes automatizados de correto funcionamento e desempenho. A metodologia inclui revisão bibliográfica, desenvolvimento de software e análise comparativa, visando quantificar ganhos em taxa de compressão, tempo de execução e uso de memória. Espera-se comprovar a viabilidade prática dos algoritmos e oferecer subsídios para futuras ampliações da biblioteca.

Palavras-chave: Compressão sem perdas; LZ77; codificação aritmética; Python.

ABSTRACT

The Komm data compression library will be extended by integrating two lossless algorithms: LZ77 and arithmetic coding. In the telecommunications network context, characterized by exponential data growth, efficient compression techniques are required to optimize bandwidth and storage. The objectives include a theoretical study of these algorithms, the design and implementation of modules in the existing Python architecture, documentation, and validation through automated testing to ensure proper functioning and performance. The methodology encompasses a literature review, software development, and comparative analysis, aiming to quantify improvements in compression ratio, execution time, and memory usage. It is expected that the practical feasibility of the algorithms will be demonstrated, providing a foundation for future expansions of the library.

Keywords: Lossless compression; LZ77; arithmetic coding; Python.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A compressão de dados é essencial para minimizar custos de armazenamento e transmissão, reduzindo o volume de dados sem comprometer o entendimento da informação. No caso da compressão sem perdas, segundo a teoria de Shannon, a entropia da fonte estabelece o limite inferior para a taxa média de bits de qualquer esquema de compressão (MACKAY, 2003). Para aproximar-se desse limite teórico, algoritmos de compressão combinam técnicas de dicionário e codificação estatísticas.

Na prática, formatos de compressão amplamente utilizados empregam essa abordagem híbrida. Por exemplo, o algoritmo DEFLATE, utilizado no formato ZIP, aplica LZ77 em conjunto com a codificação de Huffman para obter compressões melhores. Essa combinação explora tanto os padrões repetitivos de longo alcance quanto as probabilidades de ocorrência dos símbolos, elevando a eficiência global do método (DEUTSCH, 1996).

De modo geral, os algoritmos de compressão sem perdas dividem-se em métodos de codificação estatística e métodos de dicionário (SAYOOD, 2012). Por exemplo, a codificação de Huffman e a codificação aritmética são técnicas estatísticas, está ultima capaz de representar toda a mensagem como um único número fracionário no intervalo $[0, 1[$, alcançando compressões muito próximas do limite de entropia. Por sua vez, os métodos de dicionário exploram redundâncias substituindo sequências repetitivas por referências a ocorrências anteriores já vistas: um dos mais conhecidos é o algoritmo LZ77, proposto por Ziv e Lempel (ZIV; LEMPEL, 1977), que implementa esse princípio construindo dinamicamente um "dicionário" de padrões enquanto lê os dados.

Apesar do uso disseminado dessas técnicas em aplicações, a biblioteca de código aberto Komm não dispunha até o momento de uma implementação do LZ77 e nem de codificação aritmética. Essa biblioteca voltada ao ensino e à simulação em comunicação digital, já inclui diversos algoritmos clássicos de compressão, como os código de Huffman, Shannon–Fano e LZ78, evidenciando a relevância de incorporar as técnicas LZ77 e codificação aritmética para torná-la mais completa. Portanto, este trabalho tem como objetivo desenvolver e integrar um versão do LZ77 e um codificador aritmético na biblioteca Komm, avaliando estes algoritmos em termos de taxa de compressão, tempo de processamento e uso de memória.

1.1 OBJETIVO GERAL

Expandir as capacidades da biblioteca Komm com a implementação dos algoritmos LZ77 e codificação aritmética.

1.2 OBJETIVOS ESPECIFICOS

Para alcançar tais objetivos é necessário definir etapas:

- Estudo da literatura sobre o algoritmo LZ77 e suas variantes, identificando aspectos de implementação e aplicações.
- Estudo do algoritmo de codificação aritmética, contemplando modelagem probabilística, renormalização e limitações de precisão.
- Projetar e desenvolver módulos LZ77 e aritmético na arquitetura da Komm, observando padrões de codificação e cobertura de testes e escrita da documentação.
- Validar a compressão e descompressão em arquivos de textos e imagens, assegurando corretude e integridade.
- Realizar análise comparativa de desempenho (taxa de compressão, tempo de execução e uso de memória) em relação a Huffman, LZ78 e LZW.

1.3 ORGANIZAÇÃO DO TEXTO

O trabalho está estruturado em três capítulos. O Capítulo 1 apresenta a introdução ao tema estudado, contextualizando a proposta e estabelecendo os objetivos a serem alcançados no decorrer do projeto.

O Capítulo 2 descreve os fundamentos teóricos dos algoritmos de compressão selecionados, destacando seus princípios de funcionamento e principais características.

Por fim, o Capítulo 3 detalha as etapas metodológicas adotadas para o desenvolvimento da proposta, incluindo aspectos de implementação, validação e análise dos resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção será discutido o funcionamento e as principais características do LZ77 e do algoritmo aritmético, com o propósito de permitir ao leitor uma melhor compreensão dos algoritmos utilizados neste trabalho.

2.1 FUNCIONAMENTO DO LZ77

O algoritmo de compressão LZ77, proposto por Ziv e Lempel em 1977 (ZIV; LEMPEL, 1977), pertence à classe dos métodos baseados em dicionário. No LZ77, o dicionário é construído dinamicamente a partir de partes já codificadas da própria entrada, o que dispensa a necessidade de um dicionário fixo ou pré-definido. Este é utilizado como base em algoritmos amplamente adotados, como o DEFLATE (DEUTSCH, 1996).

Para realizar a codificação, o algoritmo utiliza uma janela deslizante que percorre a sequência de entrada. Essa janela é dividida em duas regiões principais:

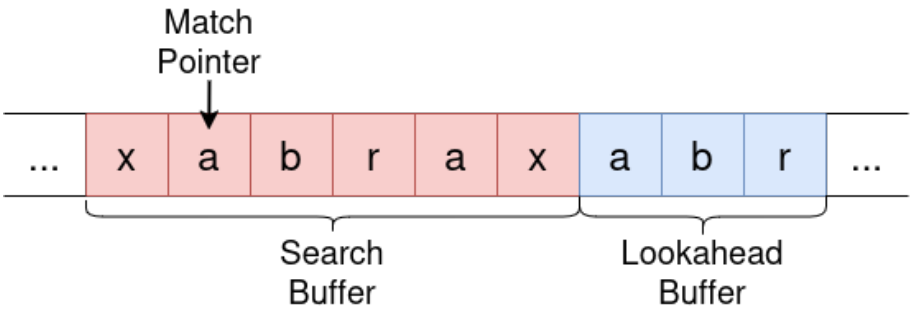
- ***Search buffer***: contém a porção já codificada da sequência, armazenando até n caracteres recentes. Ele funciona como o `"dicionário"` atual, onde o codificador tenta encontrar correspondências com os próximos caracteres a serem codificados.
- ***Lookahead buffer***: armazena os próximos caracteres a serem analisados e codificados. O codificador busca no *search buffer* a maior sequência que coincida com os caracteres do *lookahead buffer*.

Durante a codificação, o algoritmo utiliza um ponteiro de correspondência (*match pointer*) que percorre o *search buffer* para identificar o maior prefixo do *lookahead buffer* que também ocorra ali. Quando uma correspondência é encontrada, ela é codificada como um trio $\langle o, l, c \rangle$, onde:

- o (*offset*): distância entre o início da correspondência no *search buffer* e a posição atual da janela;
- l (*length*): comprimento da sequência que será codificada;
- c (*character*): próximo caractere literal após a sequência correspondente.

A Figura 1 ilustra essa estrutura, destacando as divisões da janela deslizante e o funcionamento da busca por correspondências.

Figura 1 – Divisão da janela deslizante no algoritmo LZ77



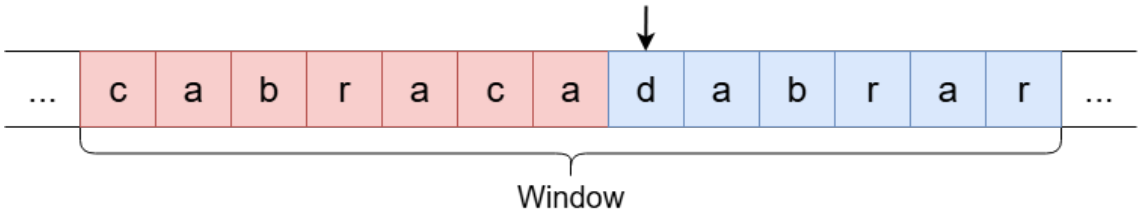
Fonte: Adaptada de Sayood (2012)

Este mecanismo permite substituir sequências repetidas por referências a ocorrências anteriores, resultando em compressões eficazes especialmente para dados com alta redundância.

2.1.1 Exemplo de Codificação com LZ77

Nesta seção será demonstrado um exemplo detalhado do funcionamento da janela deslizante e da representação do trio $\langle o, l, c \rangle$ utilizados pelo algoritmo LZ77. A sequência a ser codificada é `"cabracadabrarrarrad"`. Para este exemplo, a janela deslizante possui um tamanho total fixo de 13 caracteres, com o *lookahead buffer* definido em 6 caracteres. O estado inicial da janela pode ser visto na Figura 2.

Figura 2 – Estado inicial da janela deslizante no algoritmo LZ77

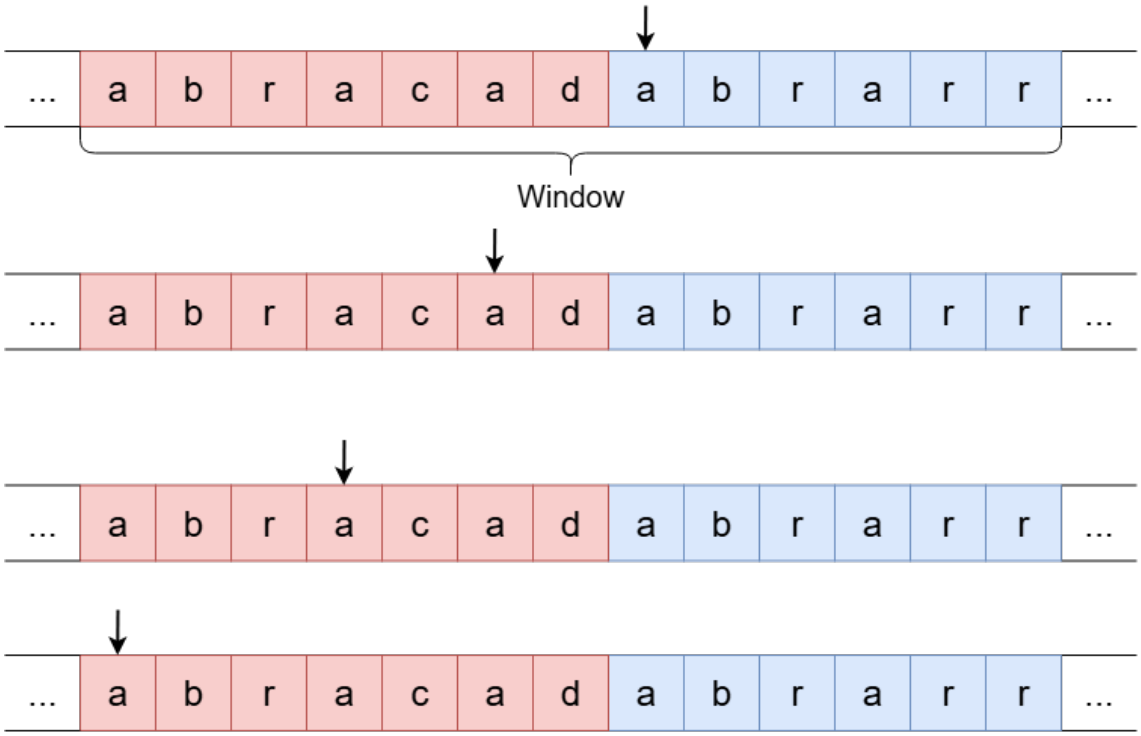


Fonte: Adaptada de Sayood (2012)

Inicialmente, busca-se no *search buffer* algum caractere ou sequência que coincida com o início do *lookahead buffer*. Neste momento, deseja-se codificar o primeiro caractere do *lookahead buffer*, que é `"d"`. Ao observar o *search buffer*, verifica-se que não há nenhuma correspondência prévia para este caractere. Portanto, o algoritmo gera o trio $\langle 0, 0, \texttt{d} \rangle$, indicando que não houve correspondência (0 de deslocamento e 0 de comprimento) e que o caractere literal transmitido é `"d"`.

Após esta codificação inicial, a janela deslizante avança uma posição, o que altera o conteúdo tanto do *search buffer* quanto do *lookahead buffer*, conforme mostrado na Figura 3.

Figura 3 – Estado da janela após primeira codificação no algoritmo LZ77



Fonte: Adaptada de Sayood (2012)

Nesse novo estado, procura-se novamente uma correspondência no *search buffer* para a sequência do *lookahead buffer*, que agora começa com `"a"`. Observando o *search buffer*, é possível encontrar múltiplas ocorrências isoladas do caractere `"a"`, porém, busca-se sempre a correspondência mais longa possível. Notavelmente, existe uma sequência completa `"abra"` previamente codificada, iniciando a 7 caracteres de distância da posição atual da janela. Essa correspondência possui comprimento 4 caracteres.

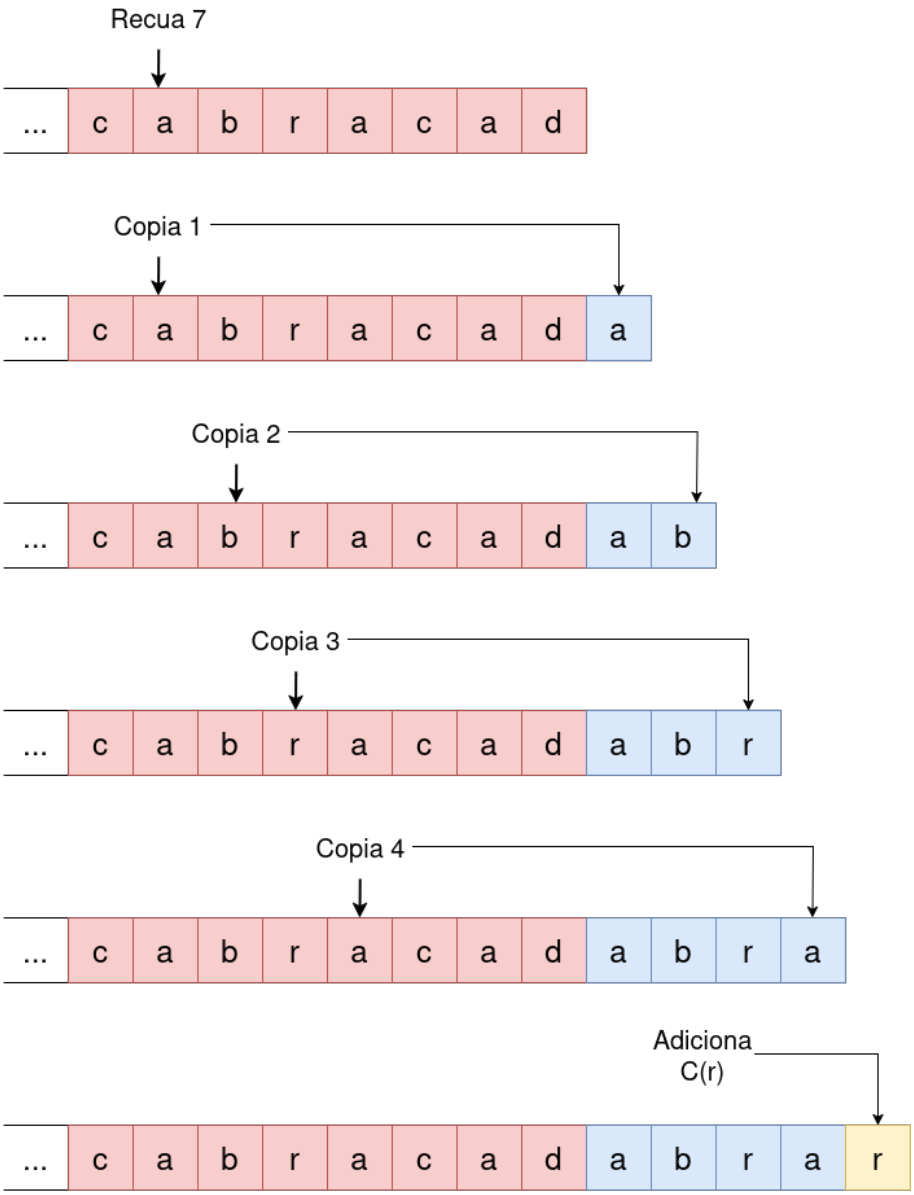
Dessa forma, o algoritmo codifica a sequência encontrada como o trio $\langle 7, 4, \texttt{r} \rangle$, onde 7 indica a distância até o início da correspondência no *search buffer*, 4 indica o comprimento da correspondência encontrada (`"abra"`), e `"r"` é o caractere seguinte imediatamente após essa sequência, ainda não codificado. Após isso, a janela avança em 5 posições (4 caracteres da sequência codificada mais 1 caractere literal).

Para realizar o processo inverso, ou seja, decodificar o trio recebido $\langle 7, 4, \texttt{r} \rangle$, o decodificador utiliza o mesmo princípio do algoritmo LZ77, porém no sentido inverso.

Inicialmente, ele utiliza o offset (o) para retornar exatamente 7 posições na sequência já decodificada até o momento. A partir dessa posição inicial encontrada, copia-se uma sequência de comprimento 4 (valor l), obtendo o trecho `"abra"`. Em seguida, adiciona-se ao final desta sequência copiada o caractere literal adicional (c), que neste exemplo é `"r"`.

Esse processo é ilustrado passo a passo na Figura 4. Inicialmente, há o estado parcial da decodificação com o *buffer* já reconstruído. Em seguida, avança-se caractere a

Figura 4 – Decodificação do exemplo $\langle 7, 4, \texttt{r} \rangle$



Fonte: Adaptada de Sayood (2012)

caractere, copiando-se do *buffer* reconstruído e adicionando o caractere literal no final. O resultado final após a decodificação deste trio será `"abrar"`.

Vale destacar que o decodificador reconstrói o buffer de busca dinamicamente, conforme recebe e processa novos trios, permitindo a reconstrução exata dos dados originais sem perda alguma.

Utilizando o exemplo anterior do trio $\langle 7, 4, \texttt{r} \rangle$, onde a janela total possui 13 caracteres, temos a seguinte definição dos campos em bits:

- O deslocamento (o) pode variar de 0 a 7, portanto requer 3 bits;
- O comprimento (l) pode variar de 0 a 6, logo exige também 3 bits;

- O caractere literal (c) é codificado em ASCII, utilizando 8 bits (1 byte).

Dado que na implementação adotada os tamanhos em bits para cada campo do trio são

$$\underbrace{3}_{\substack{\text{bits para}\\ \text{offset } (o)}} + \underbrace{3}_{\substack{\text{bits para}\\ \text{length } (l)}} + \underbrace{8}_{\substack{\text{bits para}\\ \text{character } (c)}} = 14 \text{ bits},$$

temos um formato de código de comprimento fixo, no qual cada trio $\langle o, l, c\rangle$ ocupará exatamente 14 bits. Em outras palavras, após definido o tamanho da janela (*offset*) e do *lookahead buffer* (*length*), bem como o padrão de 8 bits para o caractere literal, toda e qualquer codificação produzida por esse esquema terá comprimento contínuo de 14 bits por símbolo codificado (NELSON, 2008).

Realizando a codificação especificamente para o exemplo dado ($\langle 7, 4, \texttt{r}\rangle$), obtêm-se:

- O valor do *offset* $o = 7$, em 3 bits é $111_{(2)}$.
- O comprimento $l = 4$, em 3 bits é $100_{(2)}$.
- O caractere r, em ASCII binário (8 bits), é $01110010_{(2)}$.

Portanto, a representação completa do trio em bits será:

$$111 \mid 100 \mid 01110010$$

Resultando na sequência binária final: $11110001110010_{(2)}$.

Esse processo ilustra o funcionamento fundamental do algoritmo LZ77, mostrando como ele explora redundâncias por meio de correspondências encontradas em trechos já codificados, reduzindo o volume de dados transmitidos ou armazenados.

2.2 ALGORITMO ARITMÉTICO

O algoritmo aritmético de compressão foi desenvolvido independente e simultamente por Jorma J. Rissanen, da IBM Research, e por Richard Pasco, da Universidade de Stanford, tendo ambos publicado seus artigos em maio de 1976 (PASCO, 1976), (RISSANEN, 1976). Para uma explicação mais detalhada, pode-se consultar também a série de vídeos *"Information Theory"* por Jeffrey W. Miller.

O algoritmo aritmético é um método de codificação por fluxo (*stream code*) que representa uma sequência completa de símbolos por meio de um único número real fracionário pertencente ao intervalo $[0, 1[$. Considerando que o conjunto dos números reais nesse intervalo é infinito, torna-se possível atribuir uma *tag* exclusiva para cada sequência distinta de símbolos, estreitando progressivamente esse intervalo conforme cada símbolo é processado.

Ao contrário de métodos que atribuem códigos fixos ou códigos prefixos, como o Huffman, o algoritmo aritmético utiliza diretamente as probabilidades de cada símbolo para subdividir e refinar o intervalo, alcançando, teoricamente, taxas de compressão próximas ao limite da entropia da fonte.

2.2.1 Algoritmo com precisão infinita

Para esta primeira análise, será assumido que não há problemas relacionados à divisão infinita dos valores. Portanto, supõe-se precisão infinita durante os cálculos realizados pelo algoritmo.

Observação: Em compressões de dados é comum, e altamente recomendável, reservar um símbolo especial para indicar o fim da mensagem, conhecido como *End of File (EoF)* ou *End of Data*. Isso se deve ao fato de que a codificação resulta em um único número fracionário e, sem o *EoF*, o decodificador não teria como saber quando a mensagem termina, a menos que o seu comprimento total seja previamente conhecido. Assim, nos exemplos apresentados para o algoritmo aritmético, será adotado o uso do símbolo *End of File* para garantir a correta decodificação das mensagens.

Considere o exemplo definido pelos seguintes parâmetros:

$$\begin{aligned} X &= \{a, b, c\} \qquad \text{EoF} : a \\ p &= (p_a, p_b, p_c) = (0.2,\ 0.4,\ 0.4) \\ c &= (c_a, c_b, c_c) = (0,\ 0.2,\ 0.6) \end{aligned}$$

onde:

- X é o alfabeto dos símbolos possíveis;
- p_i é a probabilidade individual associada a cada símbolo i;
- c_i é a probabilidade acumulada, definida por:

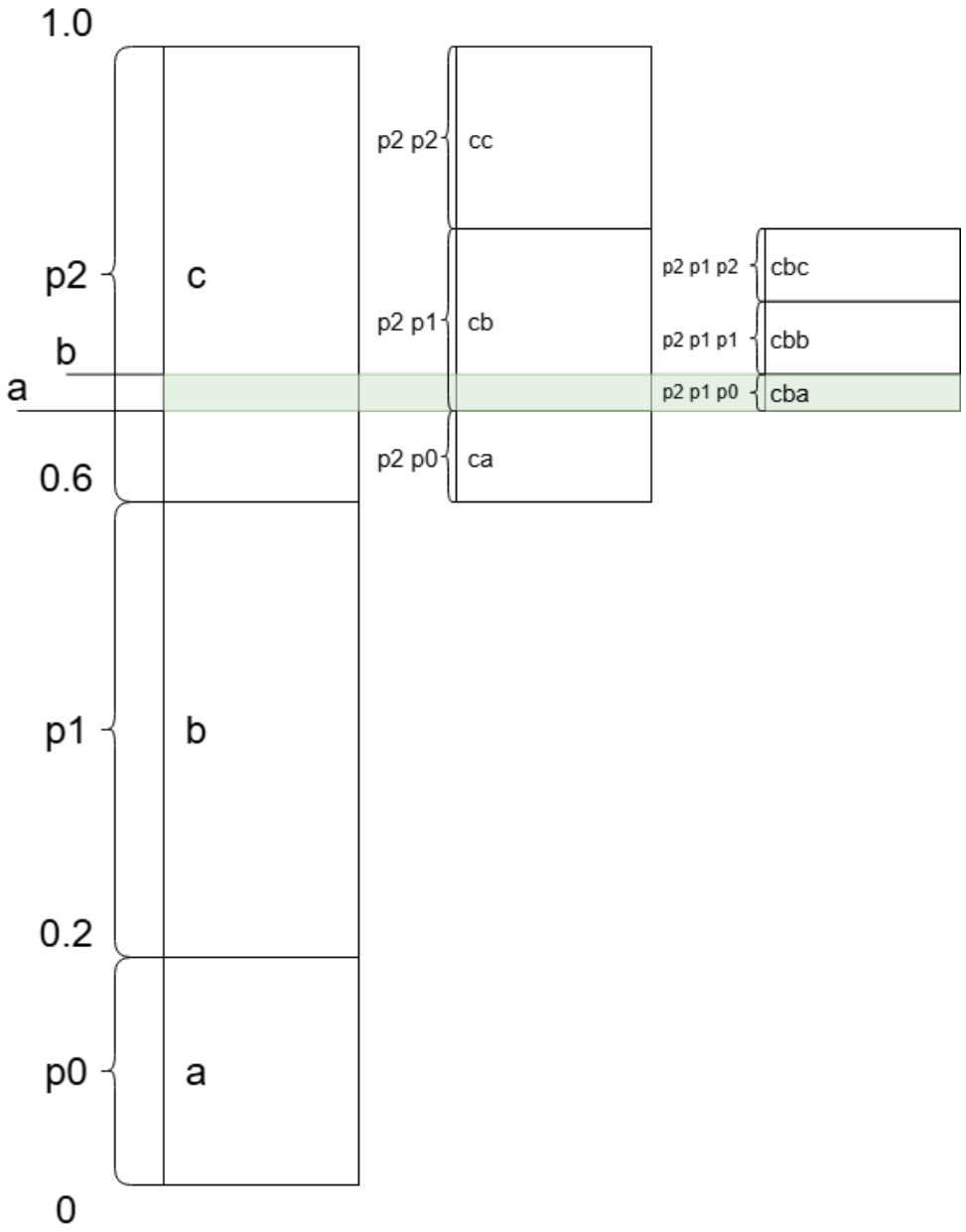
$$c_i = \sum_{j=0}^{i-1} p_j.$$

A Figura 5 ilustra o processo de codificação aritmética para a sequência `"cba"`.

O fluxo de codificação é contínuo, e todos os símbolos da mensagem são codificados de forma conjunta. Isso resulta em um código dinâmico, particularmente eficiente para fontes com distribuições probabilísticas variáveis.

O processo inicia com o intervalo inicial $[low, high[= [0, 1[$. A cada símbolo processado, esse intervalo é subdividido proporcionalmente às probabilidades acumuladas dos símbolos.

Figura 5 – Codificação da palavra `"cba"` utilizando codificação aritmética



Fonte: Elaborada pelo autor

Formalmente, para cada símbolo s correspondente ao índice j, o subintervalo é calculado por

$$[\, low + (high - low)c_{j-1},\ low + (high - low)c_j\,[\,,$$

garantindo que a largura do subintervalo resultante seja exatamente a probabilidade associada ao símbolo.

Aplicando esses cálculos ao exemplo, os subintervalos a cada etapa ficam definidos da seguinte maneira:

1. **Símbolo `"c"`**:
$$[\, low, high\,[\ =\ [\, c_2,\ c_2 + p_2\,[\ =\ [\,0{,}6;\ 1\,[\,.$$

2. **Símbolo `"b"`** (novo intervalo sobre o intervalo anterior):

$$[\, low, high\,[= [\, 0.6 + (1 - 0.6)c_1,\ 0.6 + (1 - 0.6)(c_1 + p_1)\,[= [\, 0.68,\ 0.84\,[\,.$$

3. **Símbolo `"a"`** (novo intervalo sobre o intervalo anterior):

$$[\, low, high\,[= [\, 0.68 + (0.84 - 0.68)c_0,\ 0.68 + (0.84 - 0.68)(c_0 + p_0)\,[= [\, 0.68,\ 0.712\,[\,.$$

Esse intervalo final representa a sequência completa codificada.

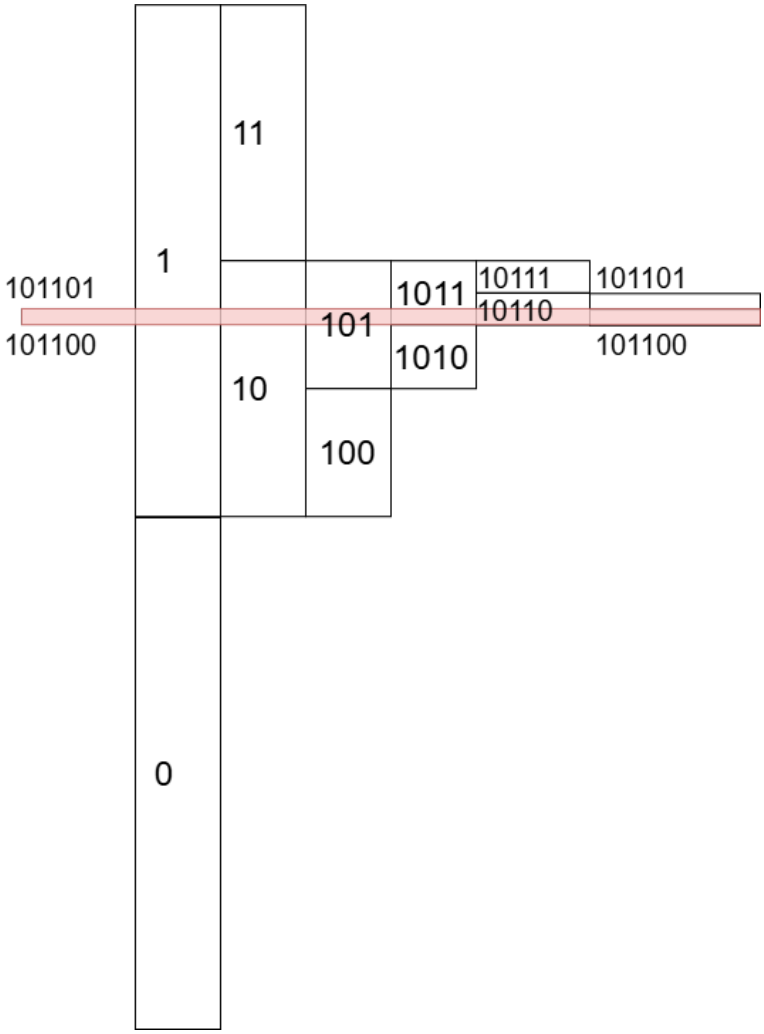
No exemplo em questão, o intervalo final obtido é [0.68, 0.712[. A transformação desse intervalo para uma sequência binária pode ser visualizada na Figura 6, com as seguintes subdivisões:

- Inicialmente, divide-se o intervalo unitário $[0; 1[$ em duas partes iguais: $[0, 0.5[$ e [0.5, 1[. Como o intervalo desejado, [0.68, 0.712[, encontra-se totalmente na segunda metade, define-se o primeiro bit como `"1"`.
- Posteriormente, subdivide-se a metade superior [0.5, 1[novamente, gerando os intervalos [0.5, 0.75[(prefixo `"10"`) e [0.75, 1[(prefixo `"11"`). Dado que o intervalo final permanece totalmente dentro de [0.5, 0.75[, define-se o segundo bit como `"0"`, resultando no prefixo `"10"`.
- Esse procedimento é repetido sucessivamente, subdividindo os intervalos gerados (tais como `"100"` e `"101"`), até obter-se um subintervalo contido integralmente no intervalo alvo original, [0.68, 0.712[.

Analisando o diagrama, identifica-se o prefixo binário:

$$\texttt{0.101100}_2$$

Figura 6 – Codificação da palavra "cba" em bits



Fonte: Elaborada pelo autor

Esse prefixo define precisamente o subintervalo binário

$$[0.101100_2, 0.101101_2[= [0.6875,\ 0.703125[,$$

que está completamente contido no intervalo alvo original $[0.68,\ 0.712[$. Assim, a sequência binária mínima que representa fielmente a mensagem original, sob hipótese de precisão infinita, é dada por "101100".

Para converter o intervalo final em uma sequência de bits, considera-se inicialmente a representação infinita do espaço $[0, 1[$ em subdivisões binárias. Cada dígito binário ("0" ou "1") corresponde a uma metade do intervalo atual, conforme ilustra o diagrama da Figura 6. A codificação prossegue extraindo o menor prefixo binário que, como número real, cai totalmente dentro do intervalo final calculado para a sequência "210".

3 PROPOSTA

O presente trabalho tem como objetivo expandir as funcionalidades da biblioteca de comunicação Komm, escrita em Python, com a implementação de dois algoritmos de compressão: a codificação LZ77 e a codificação aritmética. A proposta contempla o estudo teórico, o projeto, a implementação, a validação e a análise comparativa desses algoritmos dentro da arquitetura existente da biblioteca.

3.1 PROJETO E IMPLEMENTAÇÃO

A implementação será feita em conformidade com a arquitetura atual da biblioteca Komm, desenvolvida em Python. Para isso, serão analisadas as estruturas existentes e definidos os pontos de integração. O código será desenvolvido com à legibilidade, modularidade e aderência às práticas já utilizadas no projeto. As funcionalidades desenvolvidas incluirão rotinas de codificação e decodificação para os dois algoritmos propostos, com documentação no formato padrão da biblioteca, de forma a facilitar o uso e a manutenção por futuros colaboradores.

Testes unitários serão criados para cada funcionalidade, garantido a robustez das implementações e permitindo que possíveis erros sejam identificados e corrigidos de forma sistemática.

3.2 VALIDAÇÃO E TESTES

A validação da implementação envolverá a compressão e descompressão de arquivos com diferentes formatos e tamanhos. O foco será verificar a corretude e a integridade dos dados reconstruídos, garantindo que a descompressão resulte fielmente nos dados originais. Além disso, serão coletadas métricas de desempenho, como taxa de compressão, tempo de execução e uso memória. Esses testes buscarão avaliar a viabilidade prática e a eficiência dos algoritmos no contexto da biblioteca Komm.

3.3 ANÁLISE COMPARATIVA

Como etapa final, será realizada uma comparação sistemática entre os algoritmos implementados (LZ77 e Codificação Aritmética) e os já existentes na biblioteca Komm, como Huffman, Tunstall, LZ78 e LZW. Essa análise será baseada em critérios objetivos obtidos nos testes de desempenho e integridade, buscando destacar os contextos nos quais cada algoritmo apresenta melhor desempenho. Os resultados obtidos serão organizados

Tabela 1 – Cronograma de Atividades

Atividades	**Ago**	**Set**	**Out**	**Nov**	**Dez**
A1	X				
A2		X	X		
A3			X	X	
A4				X	X
A5	X	X	X	X	X

em tabelas e gráficos comparativos que sintetizem os pontos fortes e limitações de cada técnica no ambiente estudado.

Com isso, espera-se que o trabalho contribua significativamente para o enriquecimento da biblioteca Komm e ofereça uma base sólida para futuras extensões e estudos relacionadas à compressão de dados sem perda.

3.4 CRONOGRAMA

- **A1**: Estudo e levantamento teórico sobre os algoritmos LZ77 e Codificação Aritmética;
- **A2**: Projeto da arquitetura e integração com a biblioteca Komm;
- **A3**: Implementação dos algoritmos e testes unitários;
- **A4**: Validação dos resultados e comparação dos resultados com outros algoritmos.
- **A5**: Redação do TCC.

REFERÊNCIAS

DEUTSCH, L. Peter. **DEFLATE Compressed Data Format Specification version 1.3**. RFC Editor, mai. 1996. 17 p. RFC 1951. (Request for Comments, 1951). DOI: 10.17487/RFC1951. Disponível em: <https://www.rfc-editor.org/info/rfc1951>.

MACKAY, D.J.C. **Information Theory, Inference and Learning Algorithms**. Cambridge University Press, 2003. ISBN 9780521642989. Disponível em: <https://books.google.com.br/books?id=AKuMj4PN_EMC>.

NELSON, M. **The Data Compression Book**. BPB Publications, 2008. ISBN 9788170297291. Disponível em: <https://books.google.com.br/books?id=pndwnAEACAAJ>.

PASCO, Richard Clark. **Source coding algorithms for fast data compression.** 1976. Tese (Doutorado) – Stanford University, Stanford, CA, USA. AAI7626055.

RISSANEN, J. J. Generalized Kraft Inequality and Arithmetic Coding. **IBM Journal of Research and Development**, v. 20, n. 3, p. 198–203, 1976. DOI: 10.1147/rd.203.0198.

SAYOOD, K. **Introduction to Data Compression**. Elsevier Science, 2012. (The Morgan Kaufmann Series in Multimedia Information and Systems). ISBN 9780124157965. Disponível em: <https://books.google.com.br/books?id=mkCMxnHm6hsC>.

ZIV, J.; LEMPEL, A. A universal algorithm for sequential data compression. **IEEE Transactions on Information Theory**, v. 23, n. 3, p. 337–343, 1977. DOI: 10.1109/TIT.1977.1055714.