Universidade Federal da Paraíba
Centro de Informática
Introdução a Teoria da Informação

Trabalho Prático

Relatório sobre Implementação do PPM-C com codificador Aritmético

Grupo:
Laura de Faria Maranhão Ayres (20180019070)
Renata Mendes Pereira Campos (20200008786)
Rodrigo Pereira do Nascimento (20200008623)

Professor: Leonardo Vidal Batista

3 de setembro de 2025

Conteúdo

1 Introdução

A teoria da informação, estabelecida por Claude Shannon em 1948, fornece as bases matemáticas para compreensão e quantificação da informação. Um dos campos de aplicação mais diretos desta teoria é a compressão de dados, onde conceitos como entropia e redundância se traduzem em algoritmos práticos capazes de reduzir significativamente o tamanho de arquivos. O algoritmo *Prediction by Partial Matching (PPM)* representa uma das implementações mais elegantes dos princípios da teoria da informação aplicada à compressão. Desenvolvido originalmente por Cleary e Witten em 1984, o PPM utiliza modelos probabilísticos adaptativos baseados em contextos de ordem variável para predizer a ocorrência de símbolos, permitindo codificação eficiente através de técnicas de codificação entrópica. Este trabalho apresenta o desenvolvimento completo de um compressor-descompressor PPM com método de escape C, implementado para processar dados com alfabeto completo de bytes (0-255). O sistema combina uma estrutura de árvore Patricia para gerenciamento eficiente de contextos com codificação aritmética de precisão de 32 bits, resultando em implementação que equilibra eficiência teórica com viabilidade prática.

1.1 Fundamentação Teórica

1.1.1 Algoritmo PPM-C (Prediction by Partial Matching – Compressed)

O **PPM-C** é uma variante do algoritmo de *compressão de dados baseado em previsão por correspondência parcial* (PPM - *Prediction by Partial Matching*), amplamente utilizado em compressão de texto e sequências simbólicas.

O algoritmo funciona estimando a *probabilidade de ocorrência de cada símbolo* em uma sequência com base no contexto histórico, ou seja, nos símbolos anteriores. Diferentemente de métodos de compressão tradicionais que analisam toda a sequência de forma global, o PPM utiliza *modelos condicionais de ordem variável*, ajustando dinamicamente o tamanho do contexto para prever cada símbolo.

A versão **PPM-C** implementa melhorias na forma de lidar com símbolos desconhecidos e na atualização das probabilidades, além de incluir *mecanismos de escape* mais eficientes. Esses mecanismos permitem que, quando um símbolo não foi observado no contexto atual, o algoritmo "escape" para contextos menores, garantindo uma predição mais precisa e uma compressão mais eficiente.

1.1.2 Patricia Tree

A **Patricia Tree**, também conhecida como *Practical Algorithm To Retrieve Information Coded In Alphanumeric* ou simplesmente árvore Patricia, é uma estrutura de dados derivada das tries, projetada para armazenar e recuperar eficientemente strings ou sequências de símbolos.

Diferentemente de uma trie convencional, onde cada nó representa um único caractere, a Patricia Tree compacta os caminhos com apenas um filho, armazenando sequências de caracteres consecutivos em um único nó sempre que possível. Isso reduz drasticamente o número de nós necessários, economizando memória e acelerando buscas.

A estrutura mantém a capacidade de pesquisa, inserção e exclusão em tempo proporcional ao comprimento da chave, sendo particularmente eficiente para grandes conjuntos de dados com muitas strings compartilhando prefixos comuns.

A Patricia Tree apresenta as seguintes características fundamentais:

- **Compressão de Caminhos**: Nós com apenas um filho são eliminados, armazenando múltiplos caracteres em um único nó
- **Eficiência de Memória**: Redução significativa no número de nós comparada às tries tradicionais
- **Complexidade Temporal**: Operações de busca, inserção e remoção em $O(|k|)$, onde $|k|$ é o comprimento da chave
- **Otimização para Prefixos**: Especialmente eficiente quando as strings compartilham prefixos longos

No contexto dos algoritmos PPM (Prediction by Partial Matching), a Patricia Tree desempenha um papel fundamental na implementação eficiente dos modelos de contexto. O PPM necessita armazenar e acessar rapidamente uma grande quantidade de contextos de diferentes ordens, desde contextos de ordem zero até contextos de ordem máxima definida pelo algoritmo.

Armazenamento de Contextos A Patricia Tree é utilizada para armazenar os contextos e suas respectivas estatísticas (frequências de símbolos). Cada caminho da raiz até um nó representa um contexto específico, e cada nó contém as informações sobre as frequências dos símbolos que podem seguir aquele contexto.

Busca Eficiente de Contextos Durante a compressão ou descompressão, o algoritmo PPM precisa constantemente buscar por contextos de diferentes ordens. A Patricia Tree permite essa busca em tempo $O(|c|)$, onde $|c|$ é o comprimento do contexto, independentemente do número total de contextos armazenados.

Mecanismo de Escape Quando um símbolo não é encontrado em um contexto de ordem k, o PPM precisa "escapar" para contextos de ordem menor. A Patricia Tree facilita essa operação, permitindo navegar eficientemente pelos contextos de diferentes ordens através da estrutura hierárquica da árvore.

Otimização de Memória Em aplicações práticas do PPM, especialmente com textos longos, o número de contextos únicos pode ser muito grande. A compressão de caminhos da Patricia Tree é crucial para manter o uso de memória em níveis gerenciáveis, especialmente quando muitos contextos compartilham prefixos comuns.

Atualização Dinâmica Durante o processamento do PPM, as estatísticas dos contextos precisam ser atualizadas constantemente. A Patricia Tree permite essas atualizações de forma eficiente, mantendo a integridade da estrutura e possibilitando a inserção de novos contextos conforme necessário.

Exemplo de Compactação:

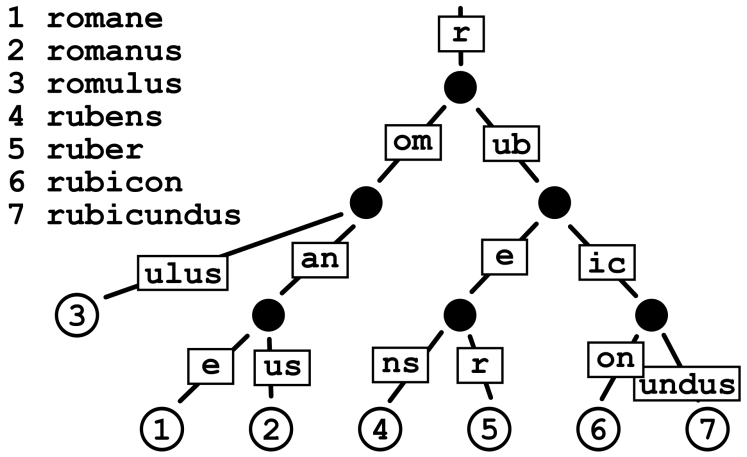

Figura 1: Patricia Tree

1.1.3 Codificação Aritmética

A **codificação aritmética** é uma técnica de compressão de dados baseada em probabilidades que representa uma sequência de símbolos como um único número real dentro do intervalo $[0, 1)$. Diferentemente de outros métodos, como o Huffman, que atribuem códigos inteiros fixos a cada símbolo, a codificação aritmética trabalha com intervalos fracionários, permitindo alcançar compressões mais próximas do limite teórico definido pela entropia de Shannon.

Do ponto de vista teórico, a codificação aritmética apresenta as seguintes vantagens:

- Aproxima-se do **limite de Shannon**, produzindo taxas de compressão muito próximas ao ótimo;
- Permite **codificação fracionária de bits por símbolo**, algo impossível em esquemas baseados em códigos inteiros (como Huffman);
- Evita **overhead para símbolos de baixa frequência**, já que mesmo eventos raros podem ser representados eficientemente.

Na implementação prática, alguns cuidados são necessários para que o algoritmo seja eficiente e numericamente estável:

- Utiliza-se tipicamente **estados de 32 bits** (ou mais, em implementações modernas) para equilibrar precisão e custo computacional, evitando estouros de variáveis;

- É necessário um mecanismo de **renormalização automática**, que garante que os intervalos continuem sendo representados corretamente ao longo da codificação, evitando perda de precisão;
- Técnicas de **tratamento de underflow** são aplicadas para garantir a estabilidade numérica, evitando que valores muito pequenos comprometam a execução correta do algoritmo.

1.1.4 Entropia

A base teórica da compressão de dados reside no conceito de entropia de Shannon, definida como:

$$H(X) = -\sum p(x_i) \log_2 p(x_i)$$

onde x_i representa os símbolos distintos da fonte X (por exemplo, bytes) e $p(x_i)$ é a probabilidade do símbolo x_i. A entropia mede a quantidade média de informação contida nos dados e estabelece o limite teórico inferior para a compressão sem perdas.

Como este trabalho aborda a implementação de um método adaptativo, a entropia é calculada a partir da média das informações dos símbolos codificados em uma mensagem de comprimento N. A informação de cada símbolo na sequência é dada por:

$$I_j = -\log_2 p(x_j)$$

onde x_j é o símbolo na posição j da mensagem. Assim, a entropia é calculada como:

$$H(X) = \frac{1}{N} \sum_{j=1}^{N} I_j$$

1.2 Objetivos

1.2.1 Objetivo Principal

Desenvolver e avaliar um compressor-descompressor PPM-C completo, demonstrando a aplicação prática dos conceitos de teoria da informação.

1.2.2 Objetivos Específicos

Implementação Técnica:

- Sistema completo de compressão/descompressão;
- Estruturas de dados otimizadas (Patricia Tree);
- Codificação aritmética de alta precisão.

Análise de Desempenho:

- Avaliação para $K_{max} = 0, 1, 2, \ldots, 6$;
- Métricas: Entropia, Tempo de Codificação e Decodificação, Tamanho Original, Tamanho Comprimido e Comprimento Médio;
- Comparação com compressores comerciais.

Validação Experimental:

- Testes no Corpus Silesia padrão;
- Verificação de integridade dos dados;
- Análise de comportamento para cada arquivo.

Geração de Texto:

- Criação de corpus inglês processado (100MB+);
- Geração probabilística usando modelo treinado.

2 Metodologia

2.1 Configuração de Software

Tabela 1: Configuração do ambiente de desenvolvimento.

Componente	**Especificação**	**Justificativa**
Linguagem	C++17	Performance nativa, controle de memória
Compilador	GCC 9.0+	Otimizações avançadas, padrões modernos
Flags de Compilação	`-std=c++17 -Wall -Wextra -O2`	Qualidade de código, otimização
Bibliotecas	STL padrão	Portabilidade, confiabilidade

Tabela 2: Especificações dos sistemas utilizados nos experimentos.

Sistema Operacional	**Processador**	**Memória**
Windows 11	Xeon E5 2620 v3, 2.40 GHz, 6 núcleos / 12 threads	16 GB DDR4, 2666 MHz
Linux Ubuntu 22.04.5 LTS	Ryzen 5 3500U, 2.10 GHz, 4 núcleos / 8 threads	12 GB DDR4, 2400 MHz

2.2 Arquitetura de Sistema

2.2.1 Organização Modular

O sistema foi projetado seguindo princípios de engenharia de software com separação clara de responsabilidades:

Tabela 3: Organização modular do sistema desenvolvido.

Módulo	**Arquivos**	**Responsabilidade Principal**
Controle Principal	`main.cpp`	Interface de usuário, fluxo de controle
Entrada de Dados	`ReadData.cpp/.hpp`	Leitura de arquivos; validação de entrada
Árvore Patrícia	`PatriciaTree.cpp/.hpp, PatriciaNode.hpp`	Gerenciamento de contextos e frequências
Codificação	`ArithmeticCoder.cpp/.hpp`	Codificação/decodificação aritmética
Streams de Bits	`BitStream.cpp/.hpp`	Manipulação de dados binários
Frequências	`FrequencyTable.cpp/.hpp`	Modelos probabilísticos
Funções Centrais	`generalFunctions.cpp/.hpp`	Algoritmos principais do PPM e geração de texto

2.2.2 Estruturas de Dados Fundamentais

PatriciaNode - Estrutura Central:
A estrutura PatriciaNode representa cada nó da árvore e encapsula toda informação necessária para modelagem estatística:

- **prefixo**: string que armazena sequência compactada de caracteres do nó pai.
- **freq**: vetor de 256 posições com frequências de cada símbolo.
- **distintos**: contador de símbolos únicos observados no contexto, útil para verificar se todos símbolos já foram vistos.
- **filhos**: mapa associando primeiro caractere ao nó filho correspondente.

PatriciaTree - Estrutura de Controle:
A classe PatriciaTree gerencia a árvore completa, fornecendo interface de alto nível para operações fundamentais:

- Inserção de contextos com atualização automática das frequências.
- Busca eficiente de contextos específicos.
- Visualização e da tabela gerada para análise.
- Gerenciamento automático de memória com limitação do histórico.

2.3 Algoritmos Implementados

2.3.1 Inserção na Árvore Patrícia

A função `inserirContexto` implementa a construção da árvore Patrícia, tratando três cenários fundamentais:

Cenário 1 - Criação de Novo Caminho:
Quando não existe caminho iniciando com o primeiro caractere do contexto, um novo nó é criado contendo todo o contexto restante como prefixo. Esta abordagem exemplifica o princípio de compactação da Patrícia Tree, armazenando sequências longas em nós únicos.

Cenário 2 - Navegação em Caminho Existente:

Quando o contexto coincide completamente com prefixo de nó existente, a navegação continua pelo caminho apropriado. Este caso representa a operação mais comum e otimizada da estrutura.

Cenário 3 - Divisão de Nó (Split):

Quando há coincidência parcial entre contexto e prefixo existente, o algoritmo executa divisão do nó, criando nó intermediário com prefixo comum e redistribuindo nós filhos. Esta operação mantém propriedades da Patrícia Tree enquanto acomoda novos padrões.

Atualização Estatística:

Após localizar ou criar o nó apropriado, frequências são atualizadas incrementalmente. O contador de símbolos distintos é atualizado apenas na primeira ocorrência de cada símbolo.

2.3.2 Algoritmo de Codificação Principal

A função `encode_master` implementa o algoritmo PPM-C completo, processando mensagens símbolo por símbolo através de abordagem hierárquica:

Inicialização do Sistema:

O sistema cria árvore Patrícia vazia, histórico de contexto e conjunto de símbolos observados. Arquivo de saída é inicializado com cabeçalho contendo tamanho original da mensagem, informação crucial para o decodificador.

Processo de Codificação Hierárquica:

Para cada símbolo, o algoritmo implementa uma busca decrescente por ordens de contexto, iniciando pela maior ordem disponível. Esta abordagem garante utilização das frequências do maior contexto existente para cada símbolo a ser codificado.

Construção de Tabelas de Frequência:

Quando contexto é encontrado, tabela de frequência é construída baseada nas estatísticas do nó, incluindo símbolo de escape com frequência igual ao número de símbolos distintos, para quando nem todos os símbolos foram vistos. Além disso, símbolos excluídos de contextos superiores são removidos da tabela, implementando mecanismo de exclusão do PPM.

Codificação de Símbolos e Escapes:

Se símbolo é encontrado no contexto atual, é imediatamente codificado usando codificação aritmética. Caso contrário, símbolo de escape é codificado, símbolos do contexto são adicionados ao conjunto de exclusões, e busca continua em contexto de ordem inferior.

Fallback para Equiprobabilidade:

Quando todos os contextos são esgotados, sistema utiliza modelo de ordem -1 com equiprobabilidade para símbolos ainda não observados na mensagem. Este mecanismo garante que qualquer símbolo possa ser codificado.

Atualização Adaptativa do Modelo:

Após codificação, modelo é atualizado inserindo símbolo em todos os contextos relevantes. Histórico é mantido como janela deslizante de tamanho KMAX, balanceando especificidade com uso de memória.

2.3.3 Algoritmo de Decodificação

A função `decode_master` implementa processo espelhado de decodificação, mantendo sincronização perfeita com o encoder:

Sincronização de Estado Inicial:
Decodificador inicializa estruturas de dados idênticas ao encoder: árvore Patrícia vazia, histórico vazio, conjunto vazio de símbolos vistos. Esta sincronização inicial é fundamental para correção do processo.

Reconstrução de Tabelas de Frequência:
Para cada símbolo a decodificar, sistema reconstrói exatamente as mesmas tabelas de frequência utilizadas durante codificação. Isto inclui aplicação das mesmas exclusões baseadas em contextos anteriores e adição do símbolo de escape com frequência idêntica.

Decodificação Hierárquica:
Utilizando tabela de frequência reconstruída, decodificador aritmético determina qual símbolo foi codificado. Se resultado for símbolo de escape, processo continua para contexto de ordem inferior, replicando exatamente a lógica do encoder.

Atualização Sincronizada:
Após decodificar cada símbolo, decodificador atualiza seu estado de forma idêntica ao encoder, garantindo que modelos permaneçam sincronizados para processamento do próximo símbolo.

2.3.4 Geração de Texto Probabilística

A função `generate_text` demonstra capacidade do modelo treinado de gerar texto seguindo distribuições aprendidas:

Amostragem Probabilística:
Sistema utiliza distribuições discretas ponderadas pelas frequências observadas para gerar símbolos. Quando símbolo de escape é selecionado, processo recua para contexto de ordem inferior, simulando mecanismo de escape durante compressão.

Manutenção de Contexto:
Gerador mantém histórico em janela deslizante, permitindo que contextos de diferentes ordens influenciem geração de acordo com disponibilidade e especificidade.

2.4 Fluxograma de Funcionamento

O sistema implementa uma arquitetura modular com funcionalidades principais integradas, conforme ilustrado na Figura 2. A estrutura principal apresenta um módulo central que se ramifica em três funcionalidades distintas: compressão, geração de tabela e geração de texto.

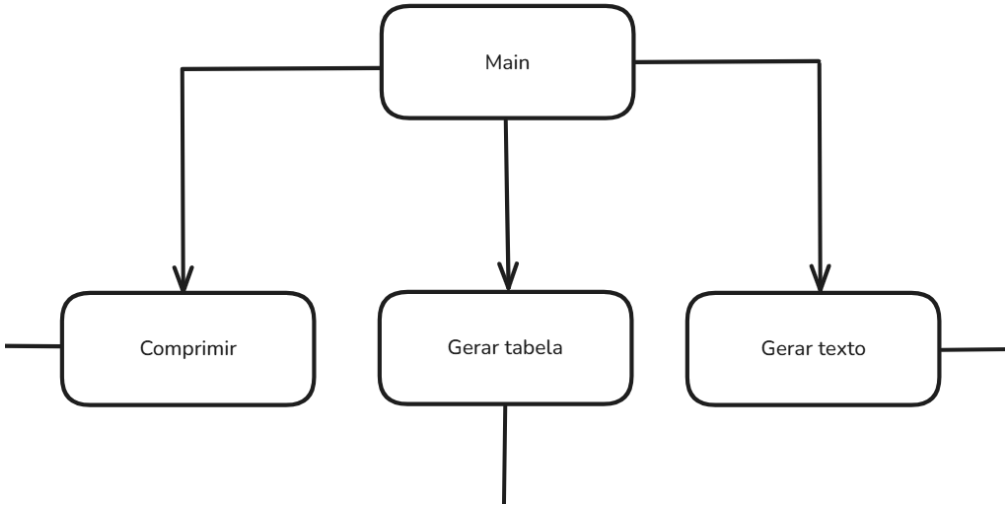

Figura 2: Fluxograma principal do sistema mostrando a arquitetura modular

2.4.1 Módulo de Compressão

O módulo de compressão implementa o algoritmo PPM-C (Prediction by Partial Matching) utilizando estrutura de árvore Patrícia para armazenamento eficiente de contextos. O fluxo detalhado deste processo é apresentado na Figura 3.

2.4.2 Fluxo Principal de Compressão

O processo inicia com a inicialização de uma árvore Patrícia vazia e entrada pelo módulo "Comprimir". Para cada símbolo da mensagem de entrada, o sistema executa uma busca hierárquica por contextos, testando ordens decrescentes desde KMAX até 0, buscando encontrar correspondência ou esgotar as possibilidades disponíveis.

Símbolos encontrados em contextos conhecidos são imediatamente codificados e a árvore é atualizada. Símbolos não encontrados geram códigos de escape (ESC) e adicionam exclusões para tentativas futuras em contextos específicos. Quando o contexto atual chega a 0 sem encontrar o símbolo, o sistema utiliza equiprobabilidades (K = -1) para codificação.

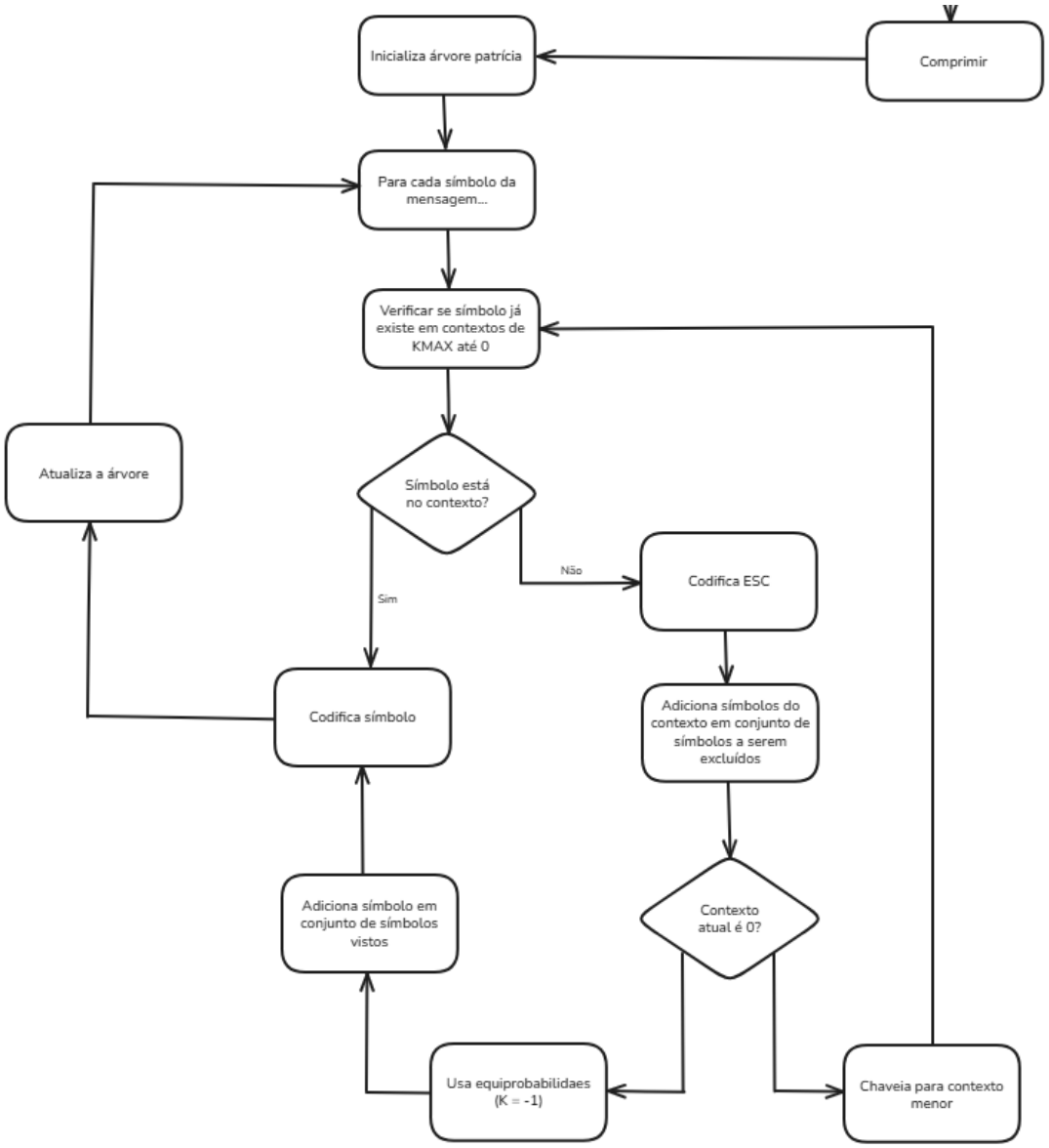

Figura 3: Fluxograma detalhado do módulo de compressão

2.4.3 Módulo de Geração de Tabela

O módulo de geração de tabela, ilustrado na Figura 4, é uma funcionalidade dedicada à construção e visualização da árvore Patrícia sem realizar compressão efetiva, gerando a um arquivo .txt que poderá ser posteriormente usado na geração de textos. O fluxo é linear e direto: inicializa a árvore Patrícia, processa cada símbolo da mensagem inserindo contextos e símbolos na estrutura, e finalmente salva a representação completa da árvore em arquivo de texto.

Figura 4: Fluxograma do módulo de geração de tabela

2.4.4 Módulo de Geração de Texto

O módulo de geração de texto, apresentado na Figura 5, utiliza uma árvore Patrícia pré-treinada para gerar texto seguindo as probabilidades aprendidas durante o processo de treinamento. O sistema inicia com contexto vazio, construindo gradualmente um histórico que permite influência de contextos de ordem crescente na geração.

No processo de geração, o fluxo inicia com a leitura do arquivo contendo a árvore previamente salva e sua reconstrução em memória. Em seguida, gera o primeiro símbolo aleatoriamente baseado nas probabilidades de contexto K=0. Para cada símbolo subsequente, verifica sua existência em contextos de KMAX até 0, adicionando-o à mensagem se encontrado ou chaveando para contexto menor caso contrário.

O processo continua até que a mensagem gerada atinja um tamanho predefinido (superior a 2000 caracteres), momento em que a geração é finalizada.

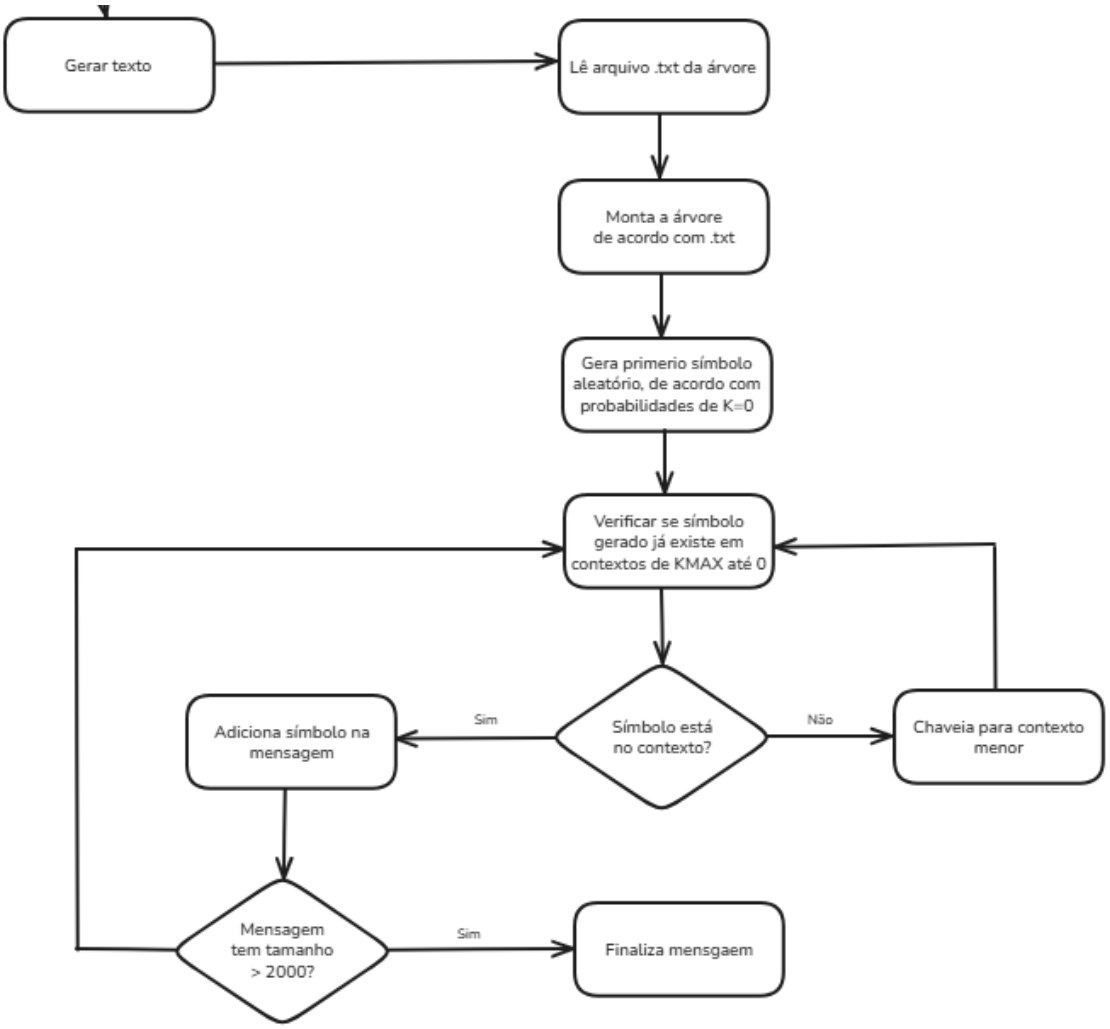

Figura 5: Fluxograma do módulo de geração de texto

2.4.5 Pontos de Controle

Arquitetura inclui múltiplos pontos de validação: verificação de integridade na decodificação, limites de recursos para controle de memória, condições de parada apropriadas, e diagnóstico detalhado de problemas quando detectados.

2.5 Corpus em inglês

O desenvolvimento de um corpus específico para treinamento de um modelo de geração de textos envolveu processo automatizado de coleta e processamento de textos do Project Gutenberg, conforme implementado no sistema de construção de corpus.

O corpus processado atende às seguintes especificações rigorosas:

- **Tamanho mínimo:** 100 MB de texto puro processado

- **Alfabeto restrito:** 27 símbolos (a-z + espaço)
- **Processamento:** remoção completa de diacríticos e pontuação
- **Normalização:** sequências múltiplas de espaços reduzidas a espaço único
- **Preservação:** separação entre palavras mantida cuidadosamente

A construção do corpus seguiu metodologia sistemática em três etapas principais:

Etapa 1 - Descoberta de Livros: Sistema automatizado testa IDs aleatórios do Project Gutenberg em faixas otimizadas (1-1000, 1000-5000, etc.), evitando duplicatas com 128 livros já conhecidos e 10 livros problemáticos previamente identificados. O algoritmo valida conteúdo substancial (maior que 3KB após remoção de metadados) e extrai títulos automaticamente.

Etapa 2 - Download e Agregação: Processo de download de 144 livros validados do Project Gutenberg, com remoção automática de metadados (cabeçalhos "START OF"/"END OF") mantendo formatação original. Cada livro é separado por marcadores "NEXT BOOK "no arquivo final.

Etapa 3 - Processamento Linguístico: Aplicação de processamento rigoroso que converte todo texto para minúsculas, remove quebras de linha (substituindo por espaços), mantém apenas letras a-z e espaços simples, elimina números, pontuação, acentos e símbolos especiais, e substitui múltiplos espaços por espaços únicos.

2.6 Métricas de Avaliação

Para avaliar o desempenho do sistema de compressão PPM-C, utilizam-se as seguintes métricas:

2.6.1 Tempo de Decodificação

O tempo de decodificação mede o desempenho do algoritmo na reconstrução da mensagem original a partir do arquivo comprimido. O cálculo segue os passos abaixo:

1. Registre o momento antes de iniciar a descompressão.
2. Execute o processo completo de decodificação, incluindo leitura do arquivo comprimido, decodificação aritmética e reconstrução da mensagem.
3. Registre o momento após terminar o processo.
4. Calcule a diferença temporal entre os registros inicial e final.

2.6.2 Tamanho Original

O tamanho original representa a quantidade de dados antes da compressão. Para medi-lo:

1. Conte o número total de bytes do arquivo de entrada.
2. Pode ser medido abrindo o arquivo e verificando seu tamanho em bytes.
3. Alternativamente, se o arquivo já estiver carregado na memória, conte os caracteres contidos.

2.6.3 Tamanho Comprimido

O tamanho comprimido corresponde à quantidade de bytes do arquivo após o processo de compressão. Para obtê-lo:

1. Meça o tamanho total do arquivo comprimido ou fluxo de saída.
2. Esse valor é utilizado no cálculo de outras métricas, como a taxa de compressão e o comprimento médio por símbolo.

2.6.4 Comprimento Médio (Bits por Símbolo)

O comprimento médio indica a eficiência da codificação em termos de quantidade de bits utilizados por símbolo. Calcula-se pela fórmula:

$$\text{Bits por Símbolo (BPS)} = \frac{\text{Tamanho do arquivo comprimido (bytes)} \times 8}{\text{Tamanho original (bytes)}}$$

2.6.5 Entropia

Utiliza-se da fórmula já mencionada:

$$H(X) = \frac{1}{N} \sum_{j=1}^{N} I_j$$

3 Resultados

Os experimentos computacionais foram realizados utilizando valores de K variando de 0 até 6. As tabelas a seguir apresentam os resultados obtidos para cada arquivo do corpus de teste Silesia.

É importante destacar que os experimentos com $K = 0$ a $K = 3$ foram conduzidos em um sistema Linux Ubuntu 22.04.5 LTS, com processador Ryzen 5 3500U (2,10 GHz, 4 núcleos / 8 threads) e 12 GB de RAM DDR4 a 2400 MHz. Já os experimentos com $K = 4$ a $K = 6$ foram realizados em um sistema Windows 11, com processador Xeon E5-2620 v3 (2,40 GHz, 6 núcleos / 12 threads) e 16 GB de RAM DDR4 a 2666 MHz.

A mudança de sistema operacional ocorreu porque, a partir de $K = 4$, o consumo de memória RAM atingia o limite total, fazendo com que o Linux encerrasse o processo automaticamente. Portanto, somente no Windows foi possível executar os experimentos com valores maiores de K.

Adicionalmente, experimentos com K superiores a 6 não foram realizados, principalmente devido ao aumento significativo no tempo de execução necessário para o benchmarking, o que inviabilizaria cumprir o prazo estabelecido para a entrega deste trabalho.

Tabela 4: Métricas de compressão para K=0.

Arquivo	**Tempo cod.**	**Tempo decod.**	**Tam. original**	**Tam. compri.**	**Entropia**	**Comp. médio**
dickens	9.933	9.989	9991663	5596649	4.481054	4.481055
mozilla	44.349	47.003	51220480	39839836	6.222485	6.222485
mr	8.432	8.909	9970564	4592189	3.684597	3.684597
nci	30.521	32.015	33553445	10188917	2.429298	2.429299
ooffice	5.424	5.718	6152192	5106506	6.640241	6.640243
osdb	8.832	9.374	10085684	8312872	6.593799	6.593799
reymont	6.258	6.663	6627202	4011511	4.842478	4.842479
samba	18.623	19.953	21606400	16460303	6.094603	6.094603
sao	6.505	6.720	7251944	6821820	7.525508	7.525508
webster	38.389	40.572	40527865	24957073	4.926403	4.926403
xml	5.115	5.309	5345280	3687246	5.518507	5.518508
x-ray	7.413	8.096	8474240	6996439	6.604900	6.604900

Tabela 5: Métricas de compressão para K=1.

Arquivo	**Tempo cod.**	**Tempo decod.**	**Tam. original**	**Tam. compri.**	**Entropia**	**Comp. médio**
dickens	10.775	11.945	9991663	4466453	3.576144	3.576144
mozilla	110.270	133.959	51220480	30037027	4.691409	4.691409
mr	9.600	10.275	9970564	4013162	3.220008	3.220008
nci	32.409	34.462	33553445	8166061	1.946998	1.946998
ooffice	7.360	7.758	6152192	3920436	5.097936	5.097937
osdb	11.514	11.922	10085684	6451583	5.117418	5.117418
reymont	7.025	7.489	6627202	2284089	2.757228	2.757229
samba	22.354	23.377	21606400	11437372	4.234809	4.234809
sao	8.496	8.734	7251944	5330205	5.880028	5.880029
webster	42.106	44.097	40527865	17714248	3.496705	3.496705
xml	5.544	5.939	5345280	2334090	3.493309	3.493310
x-ray	9.110	9.446	8474240	5920156	5.588849	5.588849

Tabela 6: Métricas de compressão para K=2.

Arquivo	Tempo cod.	Tempo decod.	Tam. original	Tam. compri.	Entropia	Comp. médio
dickens	13.813	13.434	9991663	3556106	2.847258	2.847259
mozilla	98.902	100.366	51220480	24035496	3.754044	3.754045
mr	12.727	13.676	9970564	2496832	2.003362	2.003363
nci	36.041	38.263	33553445	5608084	1.337111	1.337111
ooffice	13.116	13.707	6152192	3182702	4.138625	4.138625
osdb	20.723	21.573	10085684	3590268	2.847813	2.847813
reymont	8.143	8.475	6627202	1685965	2.035206	2.035206
samba	32.755	34.257	21606400	8394472	3.108143	3.108143
sao	25.008	25.850	7251944	5125515	5.654224	5.654225
webster	49.655	51.908	40527865	12660694	2.499158	2.499158
xml	6.610	6.797	5345280	1395762	2.088962	2.088964
x-ray	16.412	17.143	8474240	3996134	3.772500	3.772500

Tabela 7: Métricas de compressão para K=3.

Arquivo	Tempo cod.	Tempo decod.	Tam. original	Tam. compri.	Entropia	Comp. médio
dickens	16.688	16.990	9991663	2886957	2.311493	2.311493
mozilla	175.772	175.956	51220480	20245619	3.162113	3.162113
mr	18.463	18.966	9970564	2457898	1.972123	1.972124
nci	42.556	45.232	33553445	4118944	0.982061	0.982062
ooffice	23.565	22.974	6152192	2979317	3.874152	3.874153
osdb	38.310	37.705	10085684	2901014	2.301094	2.301095
reymont	10.391	10.568	6627202	1418355	1.712161	1.712161
samba	49.791	50.242	21606400	6381653	2.362875	2.362875
sao	47.082	45.002	7251944	5209958	5.747378	5.747378
webster	69.001	71.103	40527865	9591941	1.893402	1.893402
xml	8.456	8.649	5345280	926044	1.385961	1.385961
x-ray	31.479	31.568	8474240	3997478	3.773769	3.773769

Tabela 8: Métricas de compressão para K=4.

Arquivo	Tempo cod.	Tempo decod.	Tam. original	Tam. compri.	Entropia	Comp. médio
dickens	182.332	178.418	10192446	2549863	2.001374	2.001375
mozilla	1342.327	1405.837	51220480	18961001	2.961472	2.961472
mr	192.607	195.747	9970564	2447424	1.963720	1.963720
nci	535.139	544.418	34393996	2893430	0.673008	0.673008
ooffice	169.338	182.742	6152192	2871447	3.733884	3.733885
osdb	242.011	246.557	10085684	2859513	2.268175	2.268176
reymont	110.934	112.826	6627202	1316520	1.589232	1.589232
samba	429.343	438.602	21606400	5517628	2.042960	2.042961
sao	260.538	274.232	7251944	5221962	5.760620	5.760620
webster	712.037	723.918	41458703	8163802	1.575312	1.575313
xml	90.896	92.354	5345280	717080	1.073216	1.073216
x-ray	248.103	251.767	8474240	4279210	4.039734	4.039735

Tabela 9: Métricas de compressão para K=5.

Arquivo	Tempo cod.	Tempo decod.	Tam. original	Tam. compri.	Entropia	Comp. médio
dickens	221.558	225.149	10192446	2432493	1.909251	1.909252
mozilla	2901.136	3270.324	51220480	18544875	2.896478	2.896478
mr	246.045	248.550	9970564	2501456	2.007073	2.007073
nci	647.725	658.552	34393996	2487520	0.578594	0.578594
ooffice	219.452	227.045	6152192	2838664	3.691255	3.691255
osdb	298.923	309.641	10085684	2870853	2.277170	2.277171
reymont	138.717	138.631	6627202	1250385	1.509397	1.509397
samba	536.291	551.047	21606400	5148738	1.906375	1.906375
sao	368.873	395.309	7251944	5251716	5.793443	5.793444
webster	892.000	903.627	41458703	7363961	1.420973	1.420973
xml	111.597	113.309	5345280	633072	0.947484	0.947486
x-ray	342.839	356.157	8474240	4383521	4.138208	4.138208

Tabela 10: Métricas de compressão para K=6.

Arquivo	Tempo cod.	Tempo decod.	Tam. original	Tam. compri.	Entropia	Comp. médio
dickens	304.664	282.958	10192446	2415789	1.896140	1.896141
mozilla	4771.761	5284.255	51220480	18372822	2.869606	2.869606
mr	305.758	308.910	9970564	2539837	2.037868	2.037868
nci	767.270	773.977	34393996	2433701	0.566076	0.566076
ooffice	274.782	297.898	6152192	2831869	3.682419	3.682420
osdb	360.884	377.825	10085684	2892571	2.294397	2.294397
reymont	165.535	167.622	6627202	1228829	1.483376	1.483376
samba	654.428	681.255	21606400	4974973	1.842037	1.842037
sao	654.038	728.846	7251944	5303084	5.850109	5.850110
webster	1089.233	1104.422	41458703	7155036	1.380658	1.380658
xml	134.235	135.229	5345280	588760	0.881166	0.881166
x-ray	533.306	570.102	8474240	4460129	4.210529	4.210529

Tabela 11: Comparação dos tamanhos em bytes de compressão da implementação do PPM-C com K=6, do WinRAR e do 7zip.

Arquivo	PPM-C	WinRAR	7zip
dickens	2415789	3116689	3653879
mozilla	18372822	15217118	18560533
mr	2539837	2784205	3485895
nci	2433701	2083200	3039001
ooffice	2831869	2309136	3010237
osdb	2892571	3311144	3628989
reymont	1228829	1563970	1705129
samba	4974973	4149319	5193011
sao	5303084	5547658	5244727
webster	7155036	9433667	11518065
xml	588760	495345	667094
x-ray	4460129	4201012	5763032

4 Conclusão

Com base nos métodos utilizados, observou-se que os resultados obtidos foram comparáveis aos de ferramentas gratuitas disponíveis no mercado em suas configurações padrão. No entanto, percebe-se um elevado consumo de memória RAM, diretamente proporcional ao nível de contexto empregado. Além disso, quanto maior o contexto, maior também o tempo de execução.

O compressor proposto atendeu aos objetivos estabelecidos, apresentando desempenho superior em alguns casos quando comparado a concorrentes. Apesar disso, ainda se faz necessária uma etapa de otimização, especialmente para reduzir o tempo de execução e implementar mecanismos que garantam a estabilidade do sistema operacional, mesmo diante do uso intensivo de memória.

Referências

[1] BATISTA, Leonardo Vidal. *Introdução ao Processamento Digital de Imagens*. 2025. Disponível em: `https://sig-arq.ufpb.br/arquivos/2025151006a900768259042a059637481/ITI2025.1.pdf`. Acesso em: 2 set. 2025.

[2] NAYUKI. *Reference Arithmetic Coding*. 2025. Disponível em: `https://github.com/nayuki/Reference-arithmetic-coding`. Acesso em: 2 set. 2025.

[3] SALOMON, David. *Data Compression: The Complete Reference*. 3. ed. Springer, 2004. 899 p. ISBN 978-0387406978.