Testes Cruzados e Divergências no Huffman Adaptativo: Um Estudo de Caso com FGK

Luiz Guilherme Monteiro Padilha, RA: 1924745

Abstract—Este trabalho analisa divergências e incompatibilidades em implementações do algoritmo de Huffman Adaptativo, com ênfase no modelo FGK (Faller-Gallager-Knuth). No contexto do laboratório de Estrutura de Dados, diferentes alunos desenvolveram suas próprias soluções baseadas nas mesmas referências clássicas, mas testes cruzados revelaram sérios problemas de compatibilidade binária. Logs detalhados e diagramas permitiram identificar pontos críticos sujeitos a interpretações distintas, como regras de swaps, manipulação do nó NYT e questões de endianess. Os resultados mostram que pequenas variações de implementação, mesmo quando consideradas corretas individualmente, podem inviabilizar a interoperabilidade dos arquivos comprimidos. O estudo reforça a importância da documentação detalhada, padronização e validação colaborativa para garantir compatibilidade real entre diferentes implementações de algoritmos adaptativos de compressão.

Index Terms—Compressão de Dados, Huffman Adaptativo, Algoritmo FGK

1 Introdução

A compressão de dados é um tema central em Ciência da Computação, desempenhando papel fundamental no armazenamento e transmissão eficiente de informações. Entre os métodos mais consagrados, destaca-se o algoritmo de Huffman, proposto por David A. Huffman em 1952 [1], reconhecido por sua ótima capacidade de gerar códigos de comprimento variável que minimizam a repetição média dos símbolos segundo suas probabilidades de ocorrência.

A codificação de Huffman utiliza uma árvore binária para representar os símbolos, atribuindo códigos mais curtos aos mais frequentes e mais longos aos menos frequentes, resultando em uma média de comprimento próxima à entropia da entrada [1]. Apesar da eficiência, o algoritmo clássico exige conhecimento prévio das estatísticas da entrada, o que limita sua aplicação em fluxos contínuos.

• *Programa de Pós-Graduação em Ciência da Computação (PPGCC), Universidade Tecnológica Federal do Paraná - UTFPR. Área de Concentração: Sistemas e Métodos de Computação; Linha de Pesquisa: Teoria da Computação.*
E-mail: luipad@alunos.utfpr.edu.br

Manuscript received Jul 02, 2025.

Para superar essa limitação, variantes adaptativas foram propostas, permitindo que a árvore de codificação seja ajustada dinamicamente à medida que os dados são processados, sem análise estatística antecipada. Essa abordagem foi desenvolvida por Faller, Gallager e Knuth, culminando no algoritmo FGK (Faller-Gallager-Knuth)[2].

No contexto do terceiro trabalho prático da disciplina de Estrutura de Dados (Lab-03), cada aluno desenvolveu, de forma independente, sua implementação do algoritmo FGK, realizando testes cruzados para avaliar a compatibilidade entre diferentes soluções, durante o processo, surgiram incompatibilidades marcantes, arquivos comprimidos por um programa, ao serem descomprimidos por outro, apresentavam caracteres ilegíveis ou repetições inesperadas, mesmo com todos os alunos alegando seguir rigorosamente o algoritmo e as boas práticas descritas por Ziviani [5].

Este trabalho tem como objetivo identificar e relatar detalhadamente os problemas encontrados na análise dessas incompatibilidades, discutir hipóteses sobre as possíveis causas das divergências e validá-las experimentalmente, fundamentando a discussão nas referências clássicas sobre algoritmos adaptativos de Huff-

man [1], [2], [3].

2 METODOLOGIA

A solução é baseada em uma árvore binária dinâmica, na qual cada nó é implementado tendo um identificador único e decrescente (ordem), um campo de peso (frequência acumulada ou soma dos pesos dos filhos), ponteiros para pai, filho esquerdo e direito, e, no caso das folhas, um símbolo associado. Para agilizar a busca e atualização, um vetor de ponteiros para folhas é mantido, permitindo acesso imediato (O(1)) à folha de qualquer símbolo já presente.

O algoritmo inicia com o nó especial NYT ("Not Yet Transmitted"), representando todos os símbolos ainda não transmitidos, ao processar cada símbolo, se for inédito, codifica o caminho até o NYT, escreve seu valor em 8 bits, e o NYT é dividido, à esquerda um novo NYT, à direita uma folha para o novo símbolo, se o símbolo já estiver presente, codifica o caminho da raiz até sua folha correspondente. Após cada símbolo processado, inicia a atualização da árvore, da folha até a raiz, identifica em cada etapa o líder do bloco (nó de maior ordem com mesmo peso); se o líder não for o próprio nó nem seu pai, realiza o swap, incrementa o peso e o procedimento prossegue para o pai.

A ordenação única dos nós, seguindo Knuth e Vitter [2], [3], é essencial para a manutenção da *sibling property* e execução correta dos swaps, estes são permitidos entre quaisquer nós de mesmo peso e maior ordem relativa, exceto entre um nó e seu pai imediato, conforme expresso por Knuth [2], essa regra é fundamental para evitar degenerações e preservar a propriedade de prefixo dos códigos.

Para garantir a auditabilidade e facilitar a identificação de divergências em testes cruzados, o algoritmo gera logs detalhados de cada atualização, swap e do estado da árvore. Além disso, o endianess dos dados foi fixado em little-endian, conforme Ziviani [5], para assegurar compatibilidade entre sistemas distintos.

A Figura 1 ilustra um exemplo prático da árvore gerada ao final do processamento de uma sequência de entrada, destacando os campos essenciais (peso, ordem e símbolo) e evidenciando o alinhamento prático com o modelo teórico do algoritmo FGK.

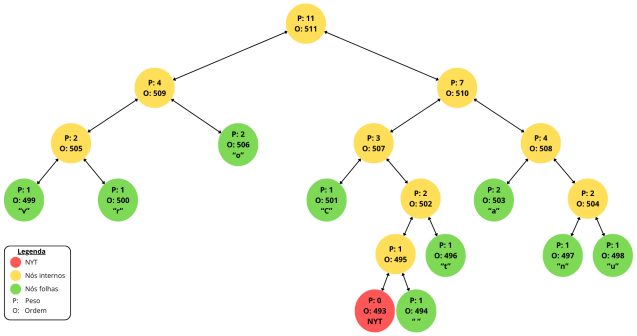

Fig. 1. Árvore binária gerada a partir do log da execução da implementação FGK para a entrada "Corvonauta "

3 TESTES E EXPERIMENTOS CRUZADOS

Para validar a implementação do algoritmo FGK e analisar sua robustez, conduzimos uma bateria de testes com arquivos de diferentes naturezas: textos curtos, arquivos binários e conjuntos compactados (.tar), tanto em ambiente local quanto em testes cruzados entre implementações independentes dos colegas de disciplina.

O primeiro passo foi a validação individual, cada implementação foi utilizada para comprimir e descomprimir arquivos de teste, confirmando que o arquivo resultante era bit a bit idêntico ao original, na etapa seguinte, realizamos os testes cruzados, arquivos comprimidos por um programa eram descomprimidos por outro, o que revelou problemas generalizados de incompatibilidade, na maioria dos casos, os arquivos descomprimidos apresentavam repetições de apenas 3 símbolos ou caracteres ilegíveis, apesar do uso das mesmas referências e boas práticas de Ziviani [5].

Logs detalhados e diagramas gerados durante a execução evidenciaram que pequenas diferenças na implementação do FGK, especialmente no tratamento de swaps, atualização do NYT, ou ordem dos bits emitidos, são suficientes para levar a árvores divergentes e, consequentemente, à falha na decodificação cruzada.

A análise desses experimentos mostrou que cada ponto passível de interpretação no algoritmo multiplica exponencialmente o número

de variantes de implementação possíveis, prejudicando drasticamente a compatibilidade binária entre soluções. Entre os principais pontos de divergência observados, destacam-se:

- As regras para realização de swaps (folhas apenas, folhas e internos, bloqueio de trocas entre ancestrais, etc.);
- O critério de atualização e ordenação do nó NYT;
- A atualização dos pesos e ordens dos nós;
- A escolha da ordem dos bits na codificação (MSB/LSB-first);
- O formato e endianess do cabeçalho dos arquivos comprimidos.

A tabela 1 ilustra, de modo simplificado, como o número de variantes possíveis pode crescer rapidamente mesmo com poucos pontos de decisão:

TABLE 1
Crescimento combinatório das variantes de implementação do FGK conforme pontos de divergência

Ponto de Divergência	Opção 1	Opção 2
Swaps permitidos	Apenas folhas	Folhas e internos
Swaps entre ancestrais	Bloqueia só pai	Bloqueia qualquer ancestral
Ordem dos bits na codificação	MSB-first	LSB-first
Endianess do cabeçalho	Little-endian	Big-endian
Inserção do NYT	Ordem decrescente	Ordem crescente
Total de combinações		$2^5 = 32$

Com apenas cinco pontos de decisão, cada um com duas opções, já são 32 combinações possíveis de comportamento entre as implementações, se cada aluno adota uma configuração diferente, a chance de compatibilidade binária diminui drasticamente, e, na prática, existem ainda mais variantes, pois algumas decisões podem envolver múltiplas alternativas ou detalhes de implementação não especificados nos artigos clássicos que podem ter sido implementados pelos alunos.

4 Conclusões

Os experimentos realizados mostram que, mesmo quando diferentes implementações do algoritmo FGK seguem as mesmas referências clássicas e boas práticas, pequenas diferenças de interpretação em aspectos como critérios de swap, manipulação do nó NYT e tratamento do endianess podem levar a incompatibilidades graves nos testes cruzados. Detalhes aparentemente sutis, se não padronizados e explicitados, tornam os arquivos comprimidos incompatíveis entre diferentes soluções, comprometendo a interoperabilidade.

A análise dos logs e diagramas gerados permitiu identificar precisamente os principais pontos sujeitos a escolhas alternativas, evidenciando que cada fator de divergência — como ilustrado na Tabela 1 — multiplica o número de variantes possíveis e reduz drasticamente a chance de compatibilidade binária. Assim, mesmo implementações "corretas" sob óticas individuais podem ser mutuamente incompatíveis, se não houver consenso em todos os detalhes operacionais.

Esses resultados reforçam a importância de uma documentação precisa, do respeito estrito às definições dos algoritmos, como discutido por Knuth [2] e Ziviani [5], e, principalmente, de uma comunicação eficiente entre todos os envolvidos no desenvolvimento. Para garantir compatibilidade real, é fundamental documentar, padronizar e validar em conjunto cada etapa do protocolo de compressão, do cabeçalho à emissão dos bits.

Por fim, a experiência evidencia que práticas como geração de logs detalhados, documentação clara e validação sistemática por meio de testes cruzados são essenciais não só para o desenvolvimento de software robusto, mas também para fomentar uma cultura colaborativa e científica no ambiente acadêmico e profissional.

References

[1] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", *Proceedings of the IRE*, vol. 40, n. 9, pp. 1098–1101, 1952.
[2] D. E. Knuth, "Dynamic Huffman Coding", *Journal of Algorithms*, vol. 6, pp. 163–180, 1985.
[3] J. S. Vitter, "Design and Analysis of Dynamic Huffman Codes", *Journal of the ACM*, vol. 34, n. 4, pp. 825–845, 1987.
[4] J. S. Vitter, "Algorithm 673: Dynamic Huffman Coding", *ACM Transactions on Mathematical Software*, vol. 15, n. 2, pp. 158–167, 1989.
[5] N. Ziviani, *Projeto de Algoritmos: com implementações em Pascal e C*, 3. ed. Rio de Janeiro: Elsevier, 2010.