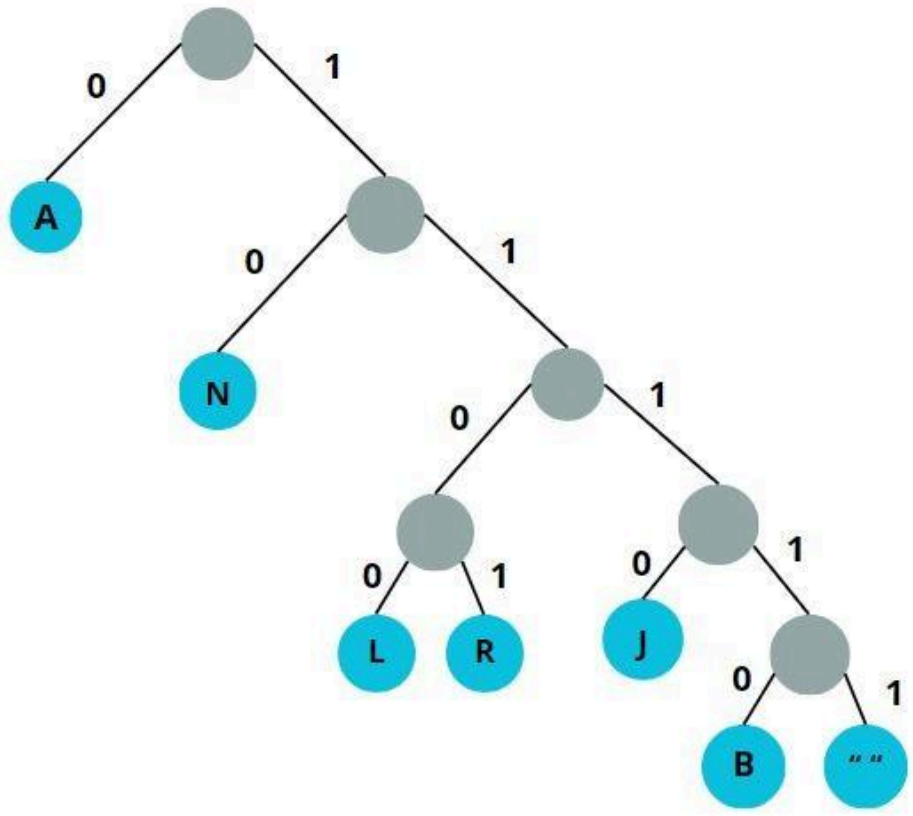
# Documentação do 2º TP de Estrutura de Dados 2025/1




**Leonardo Cheregati de Oliveira Roxo**

**Matteo Chisté Carvalho Trento**

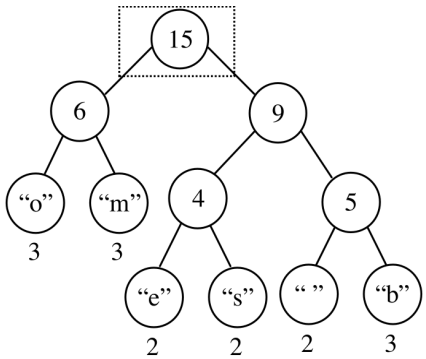
## INTRODUÇÃO

O trabalho desenvolvido é um programa compactador e descompactador de arquivos baseado no algoritmo de Codificação de Huffman. O objetivo do trabalho foi desenvolver um par de programas, "Compactador" e "Descompactador", que operam através da linha de comando para manipular arquivos binários, incluindo arquivos de texto e imagens. A compactação de Huffman é um método de compressão de dados sem perdas que atribui códigos de comprimento variável aos caracteres, onde os caracteres mais frequentes recebem códigos mais curtos, resultando em uma economia significativa de bits. O programa Compactador é responsável por ler um arquivo de entrada, construir uma árvore de Huffman baseada na frequência dos caracteres, e gerar um arquivo compactado que inclui tanto o cabeçalho da árvore quanto os dados codificados. O programa Descompactador, por sua vez, lê o arquivo compactado, reconstrói a árvore a partir do cabeçalho e, em seguida, navega pela árvore para decodificar a sequência de bits e restaurar o arquivo original.

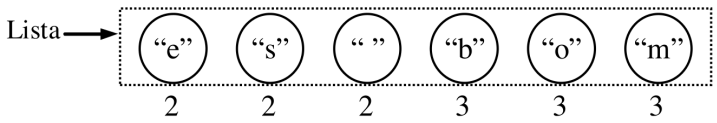
## IMPLEMENTAÇÃO

A implementação foi dividida em vários Tipos Abstratos de Dados (TADs) para promover a modularidade e a clareza do código. Os principais TADs são:

- **TAD `Árvore`**: Representa a árvore binária de Huffman, crucial para a codificação e decodificação. Todos os nós da árvore armazenam a frequência de um caractere ou a soma das frequências de seus filhos. Os nós folha contêm o caractere e um marcador (`ehletra`) para diferenciá-los dos nós internos.
    - Exemplo de árvore para a string “bom esse bombom”:



- **TAD `Lista`**: Uma lista encadeada ordenada foi utilizada para armazenar os nós da árvore durante a construção da árvore de Huffman. A lista é preenchida e mantida ordenada por frequência, garantindo que a cada passo os dois nós com as menores frequências sejam combinados, conforme o algoritmo de Huffman. Além disso, as funções `insereLista e removeLista` são amplamente utilizadas na construção da árvore, pois a cada etapa o nó combinado é criado e seus dois filhos são removidos da lista.
    - Exemplo de lista para a string “bom esse bombom”:




- **TAD `Bitmap`**: Cordialmente cedido pelo Prof. João Paulo Almeida, foi fundamental para a manipulação e escrita de bits individuais em um arquivo binário. Ele permite a adição de bits um a um (`bitmapAppendLeastSignificantBit`) e o acesso ao conteúdo do mapa de bits (`bitmapGetContents`), o que facilitou o trabalho ao permitir escrever dados que não são múltiplos de 8 bits.
- **TAD `Compactador`**: O principal módulo responsável por orquestrar o processo de compactação. Ele gerencia o ciclo de vida da compactação, desde a leitura do arquivo original até a escrita do arquivo compactado. Suas principais funcionalidades incluem:
    - **Leitura de Arquivo**: Analisa o arquivo de entrada para calcular a frequência de cada caractere.
    - **Construção da Árvore**: A partir das frequências, constrói a árvore de Huffman chamando as funções do TAD `Lista` e `Arvore`.
    - **Geração da Tabela de Códigos**: Percorre a árvore de Huffman para gerar uma tabela de códigos binários para cada caractere.
    - **Processamento e Escrita**: Utiliza a tabela de códigos para ler o arquivo original e escrever a sequência de bits compactada no arquivo de saída, junto com o cabeçalho da árvore.
- **TAD `Descompactador`**: Sua lógica é o contraposto do `Compactador`. As principais funcionalidades são:
    - **Reconstrução da Árvore**: Lê o cabeçalho binário no início do arquivo compactado para reconstruir a árvore de Huffman.
    - **Decodificação dos Dados**: Navega pela árvore reconstruída, lendo o fluxo de bits compactado um a um, até encontrar um nó folha.

    - **Escrita do Arquivo Original**: Escreve o caractere do nó folha no arquivo de saída, repetindo o processo até que a sequência de bits de final de arquivo seja encontrada.
- **Bilbioteca `IO:`** Contém a maior parte das definições de limite de tamanho, assim como as funções para leitura e escrita de binários em arquivos.

A lógica principal do programa segue os passos do algoritmo de Huffman. O Compactador calcula a frequência dos caracteres, constrói a árvore de Huffman usando a lista ordenada, e então serializa a árvore em pré-ordem para o arquivo de saída. A serialização da árvore é realizada escrevendo um bit `1` para nós folha (seguido pelo caractere em sua versão inteira, com `9` bits) e um bit `0` para nós internos. Um pseudo-caractere de 9 bits (código 256) foi adicionado à árvore para sinalizar o fim da sequência de bits compactados, evitando problemas com bits de preenchimento do sistema operacional. O Descompactador inverte o processo: lê o cabeçalho bit a bit para reconstruir a árvore, e então lê o restante do arquivo, navegando na árvore até encontrar um nó folha e decodificar o caractere correspondente.

## CONCLUSÃO

As principais dificuldades encontradas nesta implementação se concentraram em torno da manipulação de binários bit a bit, pois, embora o TAD Bitmap tenha facilitado tal manipulação, muitas operações tiveram que ser feitas diretamente na árvore ou nas funções de leitura e escrita dos arquivos. Outro empecilho significativo foi a compreensão de como alcançar todos os caracteres da tabela ASCII, pois em dado momento o código não compactava os arquivos corretamente devido à presença de números negativos representando caracteres. Posteriormente, o problema foi resolvido trocando todas as leituras de char para unsigned char, que cobre os 256 caracteres necessários. Ademais, a impressão da árvore no cabeçalho do arquivo compactado, assim como sua reconstrução no programa Descompactador, gerou muita confusão, embora tenha sido relativamente simples de implementar uma vez compreendida sua lógica, naturalmente recursiva.

Portanto, o desenvolvimento do trabalho foi desafiador, porém gratificante. A principal satisfação encontrada foi conseguir, com sucesso, compactar e descompactar arquivos relativamente grandes, vendo-os reaparecer. Conforme Arthur C. Clarke, qualquer tecnologia suficientemente avançada é indistinta de magia. Contudo, a lógica da Compactação de Huffman nos permitiu compreender de forma mais aprofundada como programas importantes, como winrar  e zip, reduzem o tamanho de arquivos sem

eliminar quaisquer informações, desmistificando tal processo que, para muitos novos desenvolvedores, parece irreal.

## BIBLIOGRAFIA


1. GeeksforGeeks, Huffman Coding in C
2. Programiz, Huffman Coding Algorithm
3. GeeksforGeeks, Serialize and Deserialize a Binary Tree
4. Citação de Arthur C. Clarke