**UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO**
CURSO DE ENGENHARIA DE COMPUTAÇÃO

ELDER RIBEIRO STORCK

# COMPACTADOR/DESCOMPACTADOR DE HUFFMAN

VITÓRIA
2021

ELDER RIBEIRO STORCK

# COMPACTADOR/DESCOMPACTADOR DE HUFFMAN

Trabalho apresentado no curso de engenharia de computação da Universidade do espírito santo

Professor:

VITÓRIA
2021

# SUMÁRIO

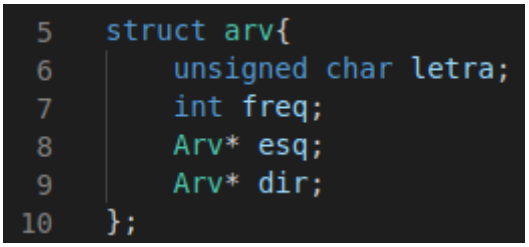
# INTRODUÇÃO

No trabalho foi feito um codificador e um decodificador usando a linguagem C e aplicando os métodos de codificação de huffman.
A codificação de huffman é um sistema de compressão de dados que consiste em criar para cada símbolo um código de tamanho variado com base na sua frequência em que esses símbolos aparecem no arquivo, depois de ter os códigos o arquivo é escrito com base nessa codificação.
O programa consiste em dois executáveis diferentes que usam os mesmos TIPO ABSTRATO DE DADO (TAD).O primeiro é um compactador que deve ler o arquivo de entrada e montar uma tabela, onde em cada linha deve ter um código de tamanho variável para um símbolo do arquivo, e depois deve gravar o arquivo em binário usando os códigos da tabela. O segundo é um descompactador que deve lê o arquivo binário e monta uma tabela do mesmo tipo e logo após escrever o arquivo descompactado.

# DESENVOLVIMENTO

## 1 TIPOS ABSTRATOS DE DADOS (TADs)

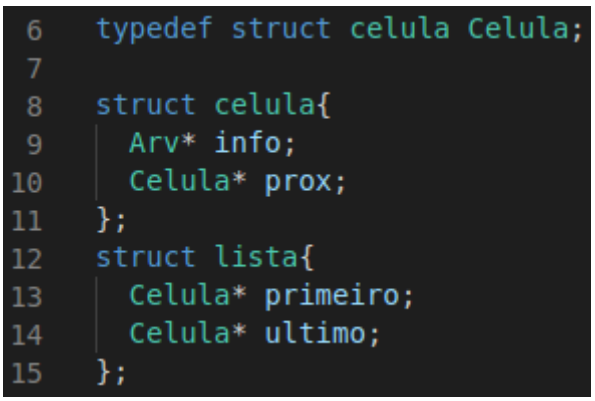
O primeiro TAD a ser desenvolvido é o TAD árvore, com a finalidade de manipular a estrutura da figura 1.



```
5   struct arv{
6       unsigned char letra;
7       int freq;
8       Arv* esq;
9       Arv* dir;
10  };
```

Figura 01: Estrutura do TAD árvore

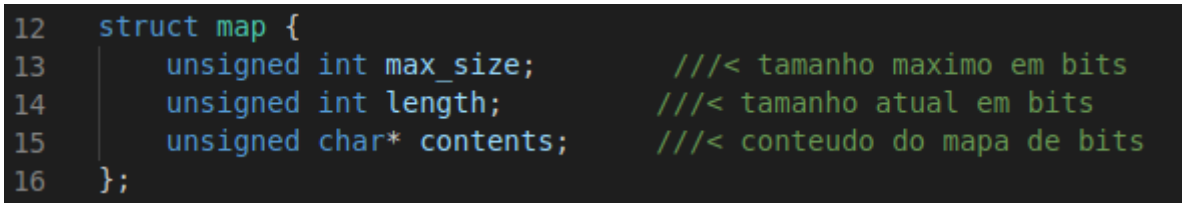
O segundo TAD a ser desenvolvido é o TAD lista de árvores, com a finalidade de manipular a estrutura da figura 2.



```
6   typedef struct celula Celula;
7
8   struct celula{
9     Arv* info;
10    Celula* prox;
11  };
12  struct lista{
13    Celula* primeiro;
14    Celula* ultimo;
15  };
```

Figura 02: Estrutura do tad lista

O terceiro TAD usado foi o TAD bitmap que foi desenvolvido pelo João Paulo Andrade (jpalmeida@inf.ufes.br), com a finalidade de ajudar a manipular os bits para a escrita e leitura no arquivo binário.



```
12  struct map {
13      unsigned int max_size;        ///< tamanho maximo em bits
14      unsigned int length;         ///< tamanho atual em bits
15      unsigned char* contents;     ///< conteudo do mapa de bits
16  };
```

Figura 03: Estrutura do TAD bitmap

Além disso, as funções adicionais usadas no compactador e no descompactador foi implementada em um outro arquivo chamado “tad_funcoes.c” e declaradas no “tad_funcoes.h”

# 2 COMPACTADOR

Depois de abrir o arquivo a ser compactado.
O primeiro passo é percorrer o arquivo lendo cada byte e montar um lista com a frequência em que eles aparecem, para isso foi usado um vetor com 255 posições e para cada ocorrência em que o byte aparecia o vetor na posição do byte era incrementado em uma unidade.
O segundo passo é para cada byte em que a frequência é diferente de zero, criar um nó folha do tipo ponteiro para árvore que contém o byte e a frequência em que ele aparece no arquivo, em seguida o nó deve ser adicionado a uma lista de árvores.
O terceiro passo é usar a lista com as árvores montadas no segundo passo para montar uma única árvore que contenha todas as árvores da lista. Para isso precisa retirar dois nós da lista que contém a menor frequência e criar um terceiro nó em que os dois nos retirados seriam adicionados como filhos nele e a frequência dele é a soma da frequência dos outros dois, depois de criar o terceiro nó basta inserir na lista o novo nó, depois de inserir o novo nó na lista o tamanho da lista ficará menor em uma unidade e assim por diante até existir um único nó.
O quarto passo é criar uma tabela com os códigos de cada símbolo baseado na árvore do terceiro passo. Para isso, basta percorrer cada nó da árvore e quando acessar o nó da direita o código recebe o bit 1 e ao acessar a esquerda o código recebe o bit 0 até chegar a um nó folha, quando um nó folha é encontrado o código é adicionado à tabela, quando todos os nós folhas da árvore estiverem mapeados a tabela estará completa.
O último passo é percorrer o arquivo a ser compactado lendo cada byte e escrever o novo arquivo binário. no arquivo binário primeiro é escrito o vetor com a frequência em que cada byte aparece e depois é escrito o texto compactado usando os códigos da tabela do quarto passo  com a ajuda do TAD bitmap.

# 3 DESCOMPACTADOR

Depois de abrir o arquivo binário a ser descompactado.
O primeiro passo é ler a lista com a frequência em que cada byte aparece no arquivo, obtendo a lista de frequência, o programa executa o segundo passo e o terceiro passo descrito no compactador e monta a árvore e a tabela com os códigos.
O segundo passo é ler cada bit e comparar com a árvore, gerando assim a sequência de bytes do arquivo antes de ser compactado. Assim, com a sequência de bytes original é só escrever no arquivo.

# CONCLUSÃO

Ao analisar os resultados dos arquivos compactados é possível concluir que apesar de o compactador apresentar uma boa taxa de compactação em arquivos texto, o mesmo não acontece com arquivos imagem. O arquivo do tipo imagem quando compactado, em alguma das vezes ocupa um espaço maior na memória do que o arquivo original.

# BIBLIOGRAFIA

Celes, Waldemar.
Introdução a estruturas de dados: com técnicas de programação em C / Waldemar Celes, Renato Cerqueira, José Lucas Rangel. – 2. ed. Rio de Janeiro : Elsevier, 2016.

Codificação de Huffman. Disponível em: (https://pt.wikipedia.org/wiki/Codifica%C3%A7%C3%A3o_de_Huffman ). Acesso em: 01/10/2021.